



孙更新 邵长恒 宾 晟 等编著

# Android

# 入门到精通

# Android 从入门到精通

孙更新 邵长恒 宾 晟 等编著

电子工业出版社

Publishing House of Electronics Industry

北京 • BEIJING

资源分享  
PDG



## 内 容 简 介

本书注重实际动手能力的培养,在遵循技术研发知识体系的严密性的同时,在容易产生错误、不易理解的环节配上翔实的开发实例呈现给读者。每一个实例都经过精心挑选,解释详尽,使读者在实例学习中掌握 Android 的编程思想和编程技巧。本书配有源代码,读者可快速、无障碍地学习 Android 实战开发技术。

本书适合具备一定软件开发经验,想快速进入 Android 开发领域的程序员,以及具备一些手机开发经验的开发者和 Android 开发爱好者阅读,也适合作为相关培训学校的 Android 培训教材。

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

Android 从入门到精通 / 孙更新等编著. —北京:电子工业出版社, 2011.10

ISBN 978-7-121-14647-3

I. ①A... II. ①孙... III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆 CIP 数据核字(2011)第 192685 号

责任编辑:李红玉

印 刷: 三河市鑫金马印装有限公司

装 订:

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编: 100036

北京市海淀区翠微东里甲 2 号 邮编: 100036

开 本: 787×1092 1/16 印张: 27.25 字数: 698 千字

印 次: 2011 年 10 月第 1 次印刷

定 价: 54.00 元

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

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn), 盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线:(010) 88258888。



## 前 言

随着计算机技术和无线通信技术的发展,移动信息设备正在深刻地改变着人们的生活,以手机、PDA 等为代表的移动信息设备已经渗透到人们的生活中。一方面,新的移动设备与移动应用不断涌现。另一方面,人们从网络信息服务中受益,并正以前所未有的主动性去创建信息、共享信息。因此,移动信息设备编程将成为今后计算机软件开发热点之一。

作为一款 Linux 内核的操作系统,Android 系统因其移植性、跨平台性以及开放性被广大移动终端商广泛采用。它涵盖移动信息设备工作所需的全部软件,包括操作系统、用户界面和应用程序。Android 系统不但可以应用于智能手机,在平板电脑市场也在急速扩张。Android 正在逐渐成为目前移动信息设备应用程序开发的最主要的平台,而且必将成为今后移动信息设备应用程序开发的主流工具。

Android 平台采用了软件堆层(Software Stack,又名软件叠层)的架构,主要分为四部分:底层以 Linux 核心为基础,并包含各种驱动,只提供基本功能。中间层包括程序库(Libraries)和 Android 运行时环境。再往上一层是 Android 提供的应用程序框架,最上面一层是各种应用软件,包括通话程序、短信程序等,这些应用软件由开发人员自行开发。

本书按照循序渐进的原则组织内容,由易到难、从入门到精通讲解 Android 关键技术以及应用开发。为了加深读者理解,本书采用先实现后分析的方法描述 Android 中的组件。书中每个知识点都包含一个实例实现以及实例分析,内容详实,示例生动丰富、可操作性强。本书不仅涵盖了 Android 平台从基础概念到高级应用的所有主题,而且深入浅出地介绍了几种常见的 Android 项目,力图将传统互联网的内容/服务与移动平台紧密结合起来,也就是将移动和互联网紧密结合,使 Web 应用的开发者能够很方便地将之前的应用程序转换到 Android 平台上来。

本书具有以下特点:

- 内容丰富:涵盖了 Android 平台从基础概念到高级应用的所有主题。
- 采用最新平台:基于最新的 SDK 进行设计和开发,详细介绍每个知识点的重要接口。
- 大量示例代码:所有知识点都包含至少一个实例。
- 参考价值强:按照循序渐进的方式组织内容。

除了封面署名作者之外,参与本书编写的还有杨霞等,在此一并表示感谢。

由于作者水平和编写时间有限,书中难免存在错误和疏漏之处,欢迎广大读者给予批评指正。

---

为方便读者阅读,若需要本书配套资料,请登录“北京美迪亚电子信息有限公司”(<http://www.medias.com.cn>),在“资料下载”页面进行下载。



目 录

第 1 章 初识 Android .....	1	2.5.3 Eclipse 中 Android 项目的 调试和运行 .....	24
1.1 什么是 Android .....	1	第 3 章 Android 中的 Activity .....	25
1.1.1 移动信息设备分类 .....	1	3.1 Activity 的作用 .....	25
1.1.2 Open Handset Alliance 和 Android .....	2	3.2 单 Activity 的 Android 应用 .....	25
1.2 Android 简介 .....	4	3.2.1 Activity 的生命周期 .....	25
1.2.1 Andriod 的历史 .....	4	3.2.2 Activity 类的结构 .....	27
1.2.2 Andriod 的未来 .....	5	3.3 Activity 的两种界面设计方式 .....	29
1.2.3 Andriod 平台的技术架构 .....	6	3.3.1 基于 XML 的界面设计 .....	29
1.3 Android 应用程序构成 .....	7	3.3.2 基于代码的界面设计 .....	30
1.3.1 活动 (Activity) .....	8	3.4 应用实例：在界面中显示图片 .....	32
1.3.2 意图 (Intent) .....	8	第 4 章 Android 人机界面和常用组件 .....	35
1.3.3 服务 (Service) .....	8	4.1 用户人机界面元素分类 .....	35
1.3.4 内容提供者 (ContentProvider) .....	8	4.1.1 视图组件 (View) .....	35
1.4 Android 网上资源 .....	9	4.1.2 视图容器组件 (ViewGroup) .....	36
第 2 章 搭建 Android 开发环境 .....	10	4.1.3 布局组件 (Layout) .....	36
2.1 Android 开发环境要求 .....	10	4.1.4 布局参数 (LayoutParams) .....	37
2.2 JDK 的安装和配置 .....	10	4.2 常用 Widget 组件 .....	38
2.2.1 安装 JDK .....	11	4.2.1 文本框视图 (TextView) .....	38
2.2.2 配置 JDK .....	12	4.2.2 按钮 (Button) .....	42
2.3 Android SDK 的下载和安装 .....	13	4.2.3 图片按钮 (ImageButton) .....	48
2.3.1 下载 Android SDK .....	13	4.2.4 编辑框 (EditText) .....	52
2.3.2 安装 Android SDK .....	14	4.2.5 多项选择 (CheckBox) .....	56
2.3.3 创建 Android 虚拟设备 .....	15	4.2.6 单项选择 (RadioGroup) .....	61
2.4 Eclipse 的下载和安装 .....	18	4.2.7 下拉列表 (Spinner) .....	64
2.4.1 下载和安装 Eclipse .....	18	4.2.8 自动完成文本框视图 (AutoCompleteTextView) .....	66
2.4.2 安装和配置 Eclipse 中的 Android 插件 .....	19	4.2.9 日期选择器 (DatePicker) .....	68
2.5 使用 Eclipse 开发 Android 应用程序 .....	22	4.2.10 时间选择器 (TimePicker) .....	72
2.5.1 使用 Eclipse 创建 Android 项目 .....	22	4.2.11 数字时钟 (DigitalClock) .....	74
2.5.2 Eclipse 中 Android 项目架构 .....	22	4.2.12 表状时钟 (AnalogClock) .....	80
		4.2.13 进度条 (ProgressBar) .....	83
		4.2.14 拖动条 (SeekBar) .....	90
		4.2.15 评分条 (RatingBar) .....	95

<b>第 5 章 Android 中的视图组件</b>	100
5.1 视图组件	100
5.1.1 图片视图 (ImageView)	100
5.1.2 滚动视图 (ScrollView)	111
5.1.3 网格视图 (GridView)	120
5.1.4 列表视图 (ListView)	127
5.1.5 切换图片 (ImageSwitcher 和 Gallery)	134
5.1.6 标签切换 (Tab)	141
5.2 通用 XML 属性	146
<b>第 6 章 Android 菜单和布局设计</b>	148
6.1 菜单 (Menu)	148
6.1.1 上下文菜单 (ContextMenu)	148
6.1.2 选项菜单 (OptionsMenu)	154
6.1.3 基于 XML 的菜单结构	169
6.2 界面布局设计	179
6.2.1 基于 XML 的布局设计	179
6.2.2 线性布局 (LinearLayout)	181
6.2.3 相对布局 (RelativeLayout)	183
6.2.4 表格布局 (TableLayout)	186
6.2.5 绝对布局 (AbsoluteLayout)	188
6.3 界面中的字体	192
6.3.1 设置系统字体	192
6.3.2 引用用户自定义字体	198
6.4 应用实例详解: 制作 手机桌面	199
6.4.1 实例分析	199
6.4.2 实例实现	200
<b>第 7 章 Android 中的核心 Intent</b>	209
7.1 Intent 的作用	209
7.1.1 多 Activity 的 Android 应用	210
7.1.2 Activity 之间的消息传递	214
7.2 Intent 的分类	220
7.2.1 Action Intent	221
7.2.2 Broadcast Intent	222
7.3 解析 Intent 的实现	224
7.3.1 Intent Receiver	224
7.3.2 Intent Filter	227
7.4 设置 Activity 许可	230
7.5 应用实例详解: 电话拨号程序	233

7.5.1 实例分析	233
7.5.2 实例实现	235
<b>第 8 章 Android 中的后台服务 Service</b>	241
8.1 Service 的作用	241
8.2 Service 的实现	242
8.2.1 创建 Service	242
8.2.2 启动 Service	244
8.3 Toast 和 Notification 应用	245
8.3.1 使用 Notification 通知 用户服务启动	245
8.3.2 使用 Toast 显示通知信息	250
8.4 应用实例详解: 播放 背景音乐	255
8.4.1 实例分析	255
8.4.2 实例实现	256
<b>第 9 章 Android 中的数据存储</b>	260
9.1 使用 Preferences 存储数据	260
9.1.1 访问 Preferences 的 API	260
9.1.2 使用 XML 存储 Preferences 数据	261
9.2 使用文件存储数据	267
9.2.1 访问应用中的文件数据	267
9.2.2 访问设备中独立的文件数据	267
9.3 使用 SQLite 数据库存储数据	271
9.3.1 SQLite 数据库简介	271
9.3.2 SQLite 数据库操作	272
9.3.3 使用 SQLiteDatabase 对象 操作数据库	272
9.3.4 Cursor 的使用	275
9.4 使用 ContentProvider	280
9.4.1 定义 ContentProvider	280
9.4.2 使用 ContentProvider 进行 CRUD 操作	281
9.5 应用实例详解: 创建音乐 播放列表	288
9.5.1 实例分析	288
9.5.2 实例实现	288
<b>第 10 章 Android 的网络通信</b>	298
10.1 访问 Internet	298
10.1.1 使用 WebKit 组件访问	



Internet .....	298	11.6.2 实例实现 .....	352
10.1.2 使用 Apache HttpComponents 访问 Internet .....	307	<b>第 12 章 Android 的搜索引擎和 Gtalk 开发 .....</b>	<b>354</b>
10.2 Socket 通信 .....	316	12.1 搜索引擎在手机中的应用 .....	354
10.3 应用实例详解: 手机 短信程序 .....	327	12.1.1 本地搜索 .....	354
10.3.1 实例分析 .....	327	12.1.2 Web 搜索 .....	355
10.3.2 实例实现 .....	329	12.2 Android 搜索引擎 API 简介 .....	358
<b>第 11 章 Android 的 GPS 应用开发 .....</b>	<b>338</b>	12.3 应用实例详解: 过滤式搜索 引擎程序 .....	<b>358</b>
11.1 GPS 在手机中的应用 .....	338	12.3.1 实例分析 .....	358
11.2 Android Location-Based API 简介 .....	339	12.3.2 实例实现 .....	359
11.3 Android 模拟器支持的 GPS 定位文件 .....	339	12.4 Android 的 GTalk 应用开发 .....	364
11.3.1 KML .....	339	12.4.1 GTalk 在手机中的应用 .....	365
11.3.2 NMEA .....	340	12.4.2 Android GTalk API 简介 .....	366
11.4 应用实例详解: 确定当前 位置的 GPS 程序 .....	341	12.4.3 为 GTalk 配置 Android 模拟器 .....	367
11.4.1 实例分析 .....	341	12.5 应用实例详解: Google GTalk 程序 .....	368
11.4.2 实例实现 .....	343	12.5.1 实例分析 .....	368
11.5 基于 Google Map 的应用 .....	347	12.5.2 实例实现 .....	370
11.5.1 使用 MapView 显示地图 .....	347	<b>第 13 章 Android 综合案例开发: 俄罗斯方块 .....</b>	<b>376</b>
11.5.2 使用 MapController 控制地图缩放 .....	350	13.1 俄罗斯方块游戏功能需求 .....	376
11.6 应用实例详解: 普通地图和 卫星地图切换 .....	351	13.2 俄罗斯方块游戏 UI 设计 .....	377
11.6.1 实例分析 .....	351	13.3 俄罗斯方块游戏功能实现 .....	392
		13.4 俄罗斯方块游戏演示 .....	423



## 第 1 章 初识 Android

Android 是 Google 开发的基于 Linux 的开源移动信息设备应用程序开发平台,该平台由操作系统、中间件、用户界面和应用软件组成,是首个为移动终端打造的真正开放和完整的软件开发平台。本章将首先对 Android 的历史、发展和功能进行简单介绍,并在此基础上详细介绍 Android 应用程序的各组成部分,为后续的应用程序开发打下良好的基础。

### 1.1 什么是 Android

Android 涵盖移动信息设备工作所需的全部软件,包括操作系统、用户界面和应用软件等,正在逐渐成为目前移动信息设备应用程序开发的最主要的平台,而且必将成为今后移动信息设备应用程序开发的主流工具。

#### 1.1.1 移动信息设备分类

随着计算机技术和无线通信技术的发展,移动信息设备正在深刻地改变着人们的生活,以手机、PDA 等为代表的移动信息设备已经渗透到人们生活的各个角落。一方面,新的移动设备与移动应用不断涌现。另一方面,人们从网络信息服务中受益,并正以前所未有的主动性去创建信息、共享信息。这些事实必将带来移动设备上大量应用程序的需要,因此,移动信息设备编程将成为今后计算机软件开发的重点之一。

移动信息设备不像 PC,它有许多的平台可以选择。从世界市场占有率来看,PC 中的 Windows 系列占了 90%以上的市场,而移动信息设备中的操作系统却呈现出群雄割据的局面。通常使用的操作系统有: Symbian、Windows Mobile、iPhone OS、Linux (含 Android、Maemo 和 WebOS)、Palm OS 和 BlackBerry OS。它们之间的应用软件互不兼容,所以移动信息设备中的应用程序需要根据不同的操作系统进行专门的开发。

Symbian 是一家软件公司,研发与授权 Symbian 操作系统。Symbian 将代表全球行业标准的 Symbian OS,授权给全球手机领导厂商使用,包括摩托罗拉、诺基亚、三星、西门子与索尼爱立信。目前, Symbian OS 的获授权厂商的销售额已超过全球手机总销售额的 50%。运行于 Symbian OS 之上的应用程序需要使用由 Symbian 公司发布的指定版本的 Symbian OS C++ SDKs 构建。一个 SDK 包含工具、应用程序接口、类库和文档等,以方便开发者能够开发新的应用程序。Symbian 手机如图 1-1 所示。

在以前,移动信息设备中的应用程序开发基本上都是面向 Symbian OS 和 Windows Mobile 系统的。但自从 iPhone 上市以来,使用 iPhone 的用户越来越多。iPhone 系统由苹果公司的 Mac OS X 发展而成,结合了多种功能于一体,如网络、桌面级的电子邮件、网页浏览及地图搜索等。全新的用户界面基于一个大型综合触摸显示屏。iPhone 平台采用 Object-c 作为开发语言, Object-c 的内核使用 C 语言构建,并基于 C 语言实现了一些面向对象的特性。iPhone 手机如图 1-2 所示。





图 1-1 Symbian 手机



图 1-2 iPhone 手机

BlackBerry (黑莓) 是 RIM 公司的手提无线通信设备品牌。其特色是支援推动式电子邮件、移动电话、文字短信、互联网传真、网页浏览及其他无线资讯服务。较新的型号亦加入个人数码助理 (PDA) 功能以及电话簿、行事历、语音通信等功能。大部分 BlackBerry 设备附设小型但完全的 QWERTY 键盘, 方便用户输入文字。BlackBerry 开发平台分为三部分, 分别是: BlackBerry Browser Development (黑莓浏览器开发)、Rapid Application Development (快速程序开发) 和 Java Application Development (Java 程序开发)。它既支持标准 Java ME 程序, 也可以开发黑莓专用的 Java 程序。BlackBerry 手机如图 1-3 所示。

网络巨头 Google 于 2007 年 11 月 5 日宣布基于 Linux 平台的开源手机操作系统 Android 诞生, 标志着移动信息设备的开发平台进入一个崭新的领域。该平台由操作系统、中间件、用户界面和应用软件组成, 是首个为移动终端打造的真正开放和完整的移动软件开发平台。Android 上的应用程序开发使用 Java 语言, 并提供了专门的 SDK。Android 手机如图 1-4 所示。



图 1-3 BlackBerry 手机



图 1-4 Android 手机

### 1.1.2 Open Handset Alliance 和 Android

BlackBerry 和 iPhone 都提供了受欢迎的、高容量的移动平台, 但是却分别针对两个不同的消费群体。BlackBerry 是企业业务用户的不二选择。但是, 作为一种消费设备, 它在应用程序的易用性和新奇性等方面难以和面向普通个人用户的 iPhone 抗衡。Android 则是一个年轻的、不断完善中的平台, 它有潜力同时涵盖移动通信设备的两个不同消费群体, 甚至可能缩小工作和娱乐之间的差别。

Android 平台是 Open Handset Alliance (开放手机联盟) 的成果, Open Handset Alliance 组织由一群共同致力于构建更好的手持移动信息设备的公司组成。这个组织由 Google 领导, 包括移动运营商、手持设备制造商、零部件制造商、软件解决方案和平台提供商以及市场营销公司。

2007 年 11 月, Google 宣布 34 家终端和运营企业加入开放手机联盟。Google、中国移动、

T-Mobile、宏达电子 (HTC)、高通、摩托罗拉等领军企业将通过开放手机联盟携手开发 Android 及其上的应用程序。

首先让我们来看看这个联盟中的成员:

### 1. 手机制造商及运营商

- 中国台湾宏达国际电子 (Palm 等多款智能手机的代工厂)
- 美国摩托罗拉 (美国最大的手机制造商)
- 韩国三星电子 (仅次于诺基亚的全球第二大手机制造商)
- 韩国 LG 电子
- 中国移动 (全球最大的移动运营商, 7.03 亿用户)
- 日本 KDDI (2900 万用户)
- 日本 NTT DoCoMo (5200 万用户)
- 美国 Sprint Nextel (美国第三大移动运营商, 5400 万用户)
- 意大利电信 (意大利主要的移动运营商, 3400 万用户)
- 西班牙 Telefónica (在欧洲和拉美有 1.5 亿用户)
- T-Mobile (德意志电信旗下公司, 在美国和欧洲有 1.1 亿用户)

### 2. 半导体公司

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

### 3. 软件公司

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





这 34 家公司中并不包含把持 Symbian 的诺基亚, 以及凭借着 iPhone 占有目前市场绝对份额的苹果公司, 当然微软公司也没有加入, 独树一帜的加拿大 RIM 和其 Blackberry 也被挡在门外。

随着 Android 平台的发展, 越来越多的相关企业加入开放手机联盟, 最新的开放手机联盟成员名单可以在其官方网站 [http://www.openhandsetalliance.com/oha\\_members.html](http://www.openhandsetalliance.com/oha_members.html) 中查看到。像我国的电信、移动、联通这三大运营商以及华为、中兴等通信设备制造商都已经加入。

开放手机联盟旨在开发多种技术, 大幅削减移动设备和服务的开发和推广成本。因为开放手机联盟中的厂商都将基于 Android 平台开发手机的新型业务, 应用之间的通用性和互联性将在最大程度上得到保持。开放手机联盟表示, Android 平台可以促使移动设备的创新, 让用户体验到最优越的移动服务, 同时, 开发商也将得到一个新的开放级别, 可更方便地进行协同合作, 从而保障新型移动设备的研发速度。随着越来越多的移动运营商和手机厂商的 Android 手机的推出, Android 平台的发展必然进入到一个全新的快速发展的阶段。

## 1.2 Android 简介

Google 公司的 Android 平台就像 Google 其他产品一样出人意料, 在正式推出之前已经传得沸沸扬扬, 可当 Android 轰轰烈烈推出的时候, 原来并非手机产品, 而是手机操作系统。下面我们就带领大家揭开 Android 的神秘面纱。

### 1.2.1 Android 的历史

虽然出现时间不长, 但作为移动信息设备的操作系统中的重量级一员, Android 开发平台正吸引越来越多的追随者投入她的怀抱, 其中包括开发者、设备生产商、软件开发商等。

通过 Android 发展历程中的大事记, 我们可以看到 Android 迅猛发展的势头。

2007 年 11 月 5 日, Google 公司宣布组建一个全球性的开放手机联盟。这一联盟将会支持 Google 发布的手机操作系统或者应用软件, 共同开发名为 Android 的开放源代码的移动系统。开放手机联盟包括手机制造商、手机芯片厂商和移动运营商等。创建时, 联盟成员数量已经达到了 34 家。

2008 年 9 月 22 日, 美国运营商 T-Mobile 在纽约正式发布第一款 Google 手机——T-Mobile G1。该款手机为中国台湾宏达电代工制造, 是世界上第一部使用 Android 操作系统的手机, 支持 WCDMA/HSPA 网络, 理论下载速率 7.2Mbps, 并支持 Wi-Fi。

2009 年 1 月 1 日, Google 的 Android 应用程序市场 (App Market) 将在 2009 年初开始出售 Android 付费应用程序。

2009 年 12 月 9 日, 宏达电将逐渐放弃 Windows Mobile 系统, 继而转向 Android 系统。

2009 年 12 月 23 日, 据知情人士透露, Google 将于 2010 年 1 月 1 日在中国大陆推出中文版 Android Market。国内已经有开发者推出针对国内用户的 Android Market。易联致远 CEO 靳岩介绍称, 其公司已经推出名为 eoeMarket 的专门针对国内用户的第三方 Android Market。

2009 年 11 月 25 日, AdMob 的调查显示, 在美国, 10 月份使用苹果 iPhone 操作系统所浏览的智能手机广告量占美国市场的 55%; 排第二位的是 Android 系统, 占 20%。至于全球市场, 10 月份通过 iPhone 系统浏览的广告量, 以市场占有率 50% 居冠; 其次是 Symbian 操作系统, 占 25%; 接着是 Android 系统, 占 11%, 居于第三位。作为一个智能手机平台的新成员来说,



Android 系统的受欢迎程度正在快步上升。

2010 年 1 月 6 日, Google 正式发布首款自有品牌手机 Nexus One, 该机采用 Android 2.1 操作系统, 裸机的定价为 529 美元 (约合人民币 3600 元)。

2010 年 2 月 24 日, 全球瞩目的世界移动大会 (Mobile World Congress 2010) 如期而至, 华为公司在此次大会上展出了 5 款 Android 终端, 并创造性地把 Android 平台运用到家庭互联网终端上, 首次发布了其 SmaKit S7 Tablet。

2010 年 3 月 3 日, 运营商 AT&T 宣布本月即将推出首款 Android 手机, 但默认搜索引擎却不是 Google, 而是雅虎。

2010 年 3 月 3 日, 网络分析公司 Quantcast 最新报告显示, 今年 2 月份, Google Android 和 RIM 移动互联网流量份额增长, 而苹果 iPhone 份额则下滑。报告指出, Android 份额在过去一年中几乎翻番, RIM 份额增长 7.5%, iPhone 份额同期下滑 10.2%。但 iPhone 仍是移动互联网流量份额的遥遥领先者, 2 月份份额近 64%; 其次是 Android, 份额约 15%; RIM 份额约 9%。

将上面的 Android 发展大事记串联起来, 就会明显感受到 Android 的咄咄逼人和当仁不让的气势。Android 的市场占有率正飞速攀升, 其带来的周边利益也越来越被从事相关产品开发的业界人士所关注和重视。

### 1.2.2 Android 的未来

Android 作为一个出现不久的移动信息设备开发平台, 因为具有一些巨大的先天优势, 所以发展前景良好。Android 的优势主要体现在:

#### 1. 系统的开放性和免费性

Android 最震撼人心之处在于 Android 手机系统的开放性和服务免费。Android 是一个对第三方软件完全开放的平台, 开发者在为其开发程序时拥有更大的自由度, 突破了 iPhone 等只能添加为数不多的固定软件的枷锁, 同时与 Windows Mobile、Symbian 等操作系统不同, Android 操作系统免费向开发人员提供, 这一点对开发者、厂商来说是最大的诱惑。

#### 2. 移动互联网的发展

Android 采用 WebKit 浏览器引擎, 具备触摸屏、高级图形显示和上网功能, 用户能够在手机上查看电子邮件、搜索网址和观看视频节目等, 比 iPhone 等具有更强的搜索功能, 界面更强大, 可以说是一种融入全部 Web 应用的互联网络平台。这正顺应了移动互联网这个大潮流, 也必将有助于 Android 的推广及应用。

#### 3. 相关厂商的大力支持

Android 项目目前正在从手机运营商、手机制造厂商、开发者和消费者那里获得大力支持。Google 移动平台主管鲁宾表示, 与软件开发合作伙伴的密切接触正在进行中。从组建开放手机联盟开始, Google 一直在向服务提供商、芯片厂商和手机销售商提供 Android 平台的技术支持。

但是 Android 也不是一个完美的系统, 它同样面临着许多挑战:

#### 1. 技术的进一步完善

目前, Android 系统在技术上仍有许多需要完善的地方, 例如: 不支持桌面同步功能, 还有自身系统的一些 bug。这些都是 Android 需要去继续完善的地方。

#### 2. 开放手机联盟模式的挑战

Android 由开放手机联盟开发、维护、完善, 还有未来的创新。很多人会担心, 最终的结局是否会像当年的 Linux 和 Windows 操作系统之争那样? 这种开发式联盟的模式, 对 Android



未来的发展、定位是否存在阻碍作用？这些未知的隐忧，也会影响到一些开发者的信心。

### 3. 其他技术的竞争

提到移动信息设备，特别是智能手机，永远都要注意 Windows Mobile，因为它的背后是微软公司，微软拥有 PC 操作系统市场最大、最牢不可破的占有率。而智能手机与 PC 互相连动，实现无缝对接，这都是智能手机的一个发展趋势。在这方面，Android 就显得稍逊一筹。此外，即使在智能手机自身的操作系统上，苹果公司的 iPhone 目前也占有绝对的霸主地位，还有 Nokia 公司的 Symbian 以及 RIM 的 Blackberry 都会与 Android 展开激烈的竞争。

## 1.2.3 Andriod 平台的技术架构

Android 平台采用了软件堆层（Software Stack），又名软件叠层的架构，主要分为四部分：底层以 Linux 核心为基础，并包含各种驱动，只提供基本功能。中间层包括程序库（Libraries）和 Android 运行时环境。再往上一层是 Android 提供的应用程序框架。最上层是各种应用软件，包括通话程序、短信程序等，这些应用软件由开发人员自行开发。

Android 平台的架构如图 1-5 所示。



图 1-5 Android 平台的架构

### 1. 应用程序（Applications）

Android 会附带一系列核心应用程序包，这些应用程序包包括 E-mail 客户端、SMS 短信程序、日历、地图、浏览器、联系人管理程序等。Android 中所有的应用程序都是使用 Java 语言编写的。

### 2. 应用程序框架（Application Framework）

开发者也可以访问 Android 应用程序框架中的 API。该应用程序架构简化了组件的重用，任何一个应用程序都可以发布它的功能块，并且任何其他的应用程序都可以使用这些发布的功能块（应该遵循框架的安全性限制）。同样，该应用程序的重用机制也使用户可以方便地替换程序组件。

隐藏在每个应用程序后面的是 Android 提供的一系列的服务和管理器，其中包括：

- 丰富而又可扩展的视图（Views）：包括列表（Lists）、网格（Grids）、文本框（Text Boxes）、

按钮 (Buttons), 甚至包括可嵌入的 Web 浏览器, 这些视图可以用来构建应用程序。

- 内容提供者 (Content Providers): 使得应用程序可以访问另一个应用程序的数据 (例如, 联系人数据库), 或者可以共享它们自己的数据。
- 资源管理器 (Resource Manager): 提供非代码资源的访问, 例如本地字符串、图形和布局文件 (Layout Files) 等。
- 通知管理器 (Notification Manager): 使得应用程序可以在状态栏中显示自定义的提示信息。
- 活动管理器 (Activity Manager): 用来管理应用程序生命周期, 并且提供常用的导航回退功能。

### 3. 程序库 (Libraries)

Android 平台包含一些 C/C++ 库, Android 系统中的组件可以使用这些库。它们通过 Android 应用程序框架为开发者提供服务。这些程序库主要包括:

- 系统 C 库: 一个从 BSD 继承的标准 C 系统函数库, 它是专门为基于嵌入式 Linux 设备定制的。
- 媒体库: 基于 PacketVideo 的 OpenCORE, 该库支持多种常用的音频、视频格式文件的回放和录制, 同时支持静态图像文件, 编码格式包括 MPEG4、H.264、MP3、AAC、AMR、JPG 和 PNG 等。
- Surface Manager: 管理显示子系统, 并且为多个应用程序提供 2D 和 3D 图层的无缝融合。
- LibWebCore: 一个最新的 Web 浏览器引擎, 支持 Android 浏览器和一个可嵌入的 Web 视图。
- SGL: 底层的 2D 图形引擎。
- 3D 库: 基于 OpenGL ES 1.0 API 实现, 该库可以使用 3D 硬件加速或者使用高度优化的 3D 软加速。
- FreeType: 用于位图和矢量字体显示。
- SQLite 库: 一个对于所有应用程序可用的、功能强劲的轻型关系型数据库引擎。

### 4. Android 运行时环境

Android 运行时环境由一个核心库和 Dalvik 虚拟机组成。核心库提供 Java 编程语言核心库的大多数功能。每一个 Android 应用程序都在自己的进程中运行, 都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备可以同时高效地运行多个虚拟系统。它依赖于 Linux 内核的一些功能, 例如线程机制和底层内存管理机制等。Dalvik 虚拟机执行 .dex 的 Dalvik 可执行文件, 该格式文件针对小内存的使用进行了优化, 同时虚拟机是基于寄存器的, 所有的类由 Java 编译器编译, 然后通过 SDK 中的 “dx” 工具转化成 .dex 格式, 最后由虚拟机执行。

### 5. inux 内核

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

## 1.3 Android 应用程序构成

在通常情况下, 一个 Android 应用程序是由以下 4 个组件构成的: 活动 (Activity)、意图 (Intent)、服务 (Service) 和内容提供者 (Content Provider)。这 4 个组件是构成 Android 应用



程序的基础，但并不是每个 Andorid 应用程序都必须包含这 4 个组件，除了 Activity 是必要组分之外，其余组件都是可选的。

### 1.3.1 活动 (Activity)

活动 (Activity) 是最基本的 Andorid 应用程序组件。在应用程序中，一个活动通常就是一个单独的屏幕。每个活动都是通过继承活动基类被实现为一个独立的类，活动类将会显示由视图控件组成的用户接口，并对事件做出响应。

大多数的应用程序都是由多个屏幕显示组成。例如，一个发送信息的应用也许有一个显示发送消息的联系人列表屏幕，第二个屏幕用来写文本消息和选择收件人，第三个屏幕可以查看历史消息或者进行消息设置操作等。这里每个屏幕都是一个活动，很容易实现从一个屏幕到一个新屏幕并且完成新的活动。因为 Android 会把每个从主菜单打开的程序保留在堆栈中，所以当打开一个新屏幕时，之前的屏幕会被置为暂停状态并且压入历史堆栈中。用户可以通过回退操作回到以前打开过的屏幕，也可以选择性地移去一些没有必要保留的屏幕。

### 1.3.2 意图 (Intent)

Intent 用来描述应用程序想做什么。它是一种运行时绑定机制，能在程序运行的过程中连接两个不同的组件。通过 Intent，应用程序可以向 Android 表达某种请求或者意愿，Android 会根据意愿的内容选择适当的组件来响应。

与 Intent 相关的两个类分别是 IntentFilter 和 IntentReceiver。当 Intent 请求做某个动作时，IntentFilter 用于描述一个活动或者广播接收器能够操作哪些 Intent。而 IntentReceiver 可使应用程序对外部事件做出响应。

Intent 中两个最重要的部分是动作和动作对应的数据。典型的动作类型有活动的查看 (View)、选取 (Pick)、编辑 (Edit) 等，而动作对应的数据则以 URI 形式进行表示。

### 1.3.3 服务 (Service)

服务是 Android 应用程序中具有较长的生命周期但是没有用户界面的代码程序。它在后台运行，并且可以与其进行交互。它跟 Activity 的级别差不多，但是不能自己运行，需要通过某一个 Activity 来调用。

Android 应用程序的生命周期是由 Android 系统来决定的，不由具体的应用程序的线程来左右。当应用程序要求在没有界面显示的情况还能正常运行 (要求有后台线程，而且直到线程结束，后台线程是不会被系统回收的)，这个时候就需要用到 Service 了。

Service 典型的例子是一个具有播放列表功能的正在播放歌曲的媒体播放器。在媒体播放器应用中，可能会有一个或多个活动，让使用者可以选择并播放歌曲。然而活动本身并不处理音乐播放功能，因为用户期望在切换到其他屏幕后，音乐应该还在后台继续播放。

### 1.3.4 内容提供者 (ContentProvider)

Android 应用程序可以使用文件或 SQLite 数据库来存储数据。ContentProvider 提供了一种多应用间数据共享的方式。当开发者希望自己的应用数据能与其他应用共享时，内容提供者将会非常有用。一个内容提供者类实现了一组标准的方法，能够让其他的应用保存或读取此内容提供者处理的各种数据类型。

也就是说,一个应用程序可以通过实现一个 `ContentProvider` 的抽象接口将自己的数据暴露出去。外界根本看不到,也不用看到这个应用程序暴露的数据在应用程序当中是如何存储的,但是外界可以通过一套标准及统一的接口和应用程序里的数据打交道,可以读取应用程序的数据,也可以删除应用程序的数据。

## 1.4 Android 网上资源

Google 为 Android 平台和基于该平台的 Android 应用程序开发提供了大量的信息和有用的服务。例如扩展 Android 平台的外部库、Android 应用程序、托管的服务和 API、Android 开发人员竞赛等。这些信息和服务都在 Google 为 Android 设置的官方网站中。此网站的所有内容均由 Google 为了 Android 开发人员的利益而提供。

如果要查找关于 Android 的一般信息,请访问 [www.android.com](http://www.android.com) 网站。如果对开发用于 Android 设备的应用程序感兴趣,请访问 Android 开发人员网站,网址是 [developer.android.com](http://developer.android.com)。

除了 Google 提供的 Android 官方网站之外,还有许多 Android 爱好者和组织构建了一些相关的技术网站和论坛,Android 开发者和初学者可以通过这些网上资源进行学习和技术交流。

91 手机娱乐门户: <http://android.sj.91.com/>

Android 手机网: <http://www.android123.com/>

Android 手机资讯网: <http://android.hk.cn/>

Android 开发者: <http://www.androidin.com/>

Android 开发网: <http://www.android123.com.cn/>

Android 论坛: <http://bbs.android123.com/>

Android 实验室: <http://www.androidlab.cn/>

Android 论坛中文论坛: <http://www.androidin.net/bbs/index.php>

Android 中文网: <http://www.androidcn.net/>

Google Android 爱好者论坛: <http://www.loveandroid.com/>

中国台湾 Android 中文资源站: <http://android.cool3c.com/>

Android 手机资源中文共享社区: <http://www.apkcn.com/>

Google Android 论坛: <http://www.android1.net/>

Android 开发者论坛: <http://bbs.androidin.com/>

Android 开发资源下载: <http://www.androidhere.cn/>

新学网  
PDG



## 第 2 章 搭建 Android 开发环境

“工欲善其事，必先利其器”，要进行 Android 应用程序开发，必须首先搭建 Android 开发环境。本章将以快速熟悉开发条件、开发环境为目的，详细介绍搭建 Android 开发环境所需要的操作系统、Android SDK、IDE 等，并在此基础上重点介绍在 Windows 操作系统中搭建开发环境的过程和步骤，为读者学习开发 Android 应用做好前期准备。

### 2.1 Android 开发环境要求

搭建 Android 开发环境需要预先准备如表 2-1 所示的作业环境以及程序。

表 2-1 Android 开发环境所需项目

所需项目	版本要求	说明
操作系统	Microsoft Windows XP/Vista 及以上操作系统或 Mac OS X 10.4.8 或更新的版本（硬件必须是 x86 版本）或 Linux	本书以 Windows 操作系统为例
SDK	Android SDK 1.0r2 以上 （本书所有实例皆以最新的 Android SDK 2.1 版本为开发环境）	截至目前，最新版本为 2.1
IDE	Eclipse 3.3 以上 （本书所有实例皆以 Eclipse-SDK-3.5.2-Win32 版本为编译环境）	使用 Eclipse IDE for Java Developers 版本
开发插件 ADT	ADT（Android Development Tools）0.9.5 以上	Eclipse 开发 Android 应用的必需插件
其他	Java Development Kit（JDK）v5.0 以上 （本书使用 JDK 6.0）	不可以只有 JRE，必须要有 JDK

JDK 6.0 可以到 Sun 公司的官方网站 <http://java.sun.com/javase/downloads/index.jsp> 下载。

Eclipse 3.5 可以从网址 <http://www.eclipse.org/downloads/> 下载。

Android SDK 2.1 可以从 Google 公司的 Android 开发网站 <http://developer.android.com> 下载。

下面将详细介绍表 2-1 中各种软件的下载、安装以及配置的详细步骤和所需注意事项。

### 2.2 JDK 的安装和配置

JDK 的全称是 Java Development Kit，翻译成中文就是 Java 开发工具包，主要包括 Java 运行环境（Java Runtime Environment）、一些 Java 命令工具和 Java 基础的类库文件。JDK 是开发任何类型 Java 应用程序的基础。因为开发 Android 应用程序时，使用的开发语言是 Java，而且安装 Eclipse 集成开发环境也需要 JDK 的支持，如果没有 JDK，则启动 Eclipse 时将会报错。所以首先在系统中必须正确地安装和配置 JDK。

### 2.2.1 安装 JDK

JDK 程序安装包可以从 Sun 公司的官方网站免费下载，目前最新的版本是 6.0。一般情况下，一个版本的 JDK 会同时支持不同操作系统的多个版本，所以在下载时，用户要根据所使用的操作系统来选择。本书介绍的 Android 开发是基于 Windows 操作系统的，所以下面以 Windows 下的 JDK 为例，介绍其安装的具体步骤。

- (1) 双击下载的 JDK 安装文件，弹出如图 2-1 所示的初始化安装界面。

(2) 初始化完成后，将进入如图 2-2 所示的许可证协议对话框。

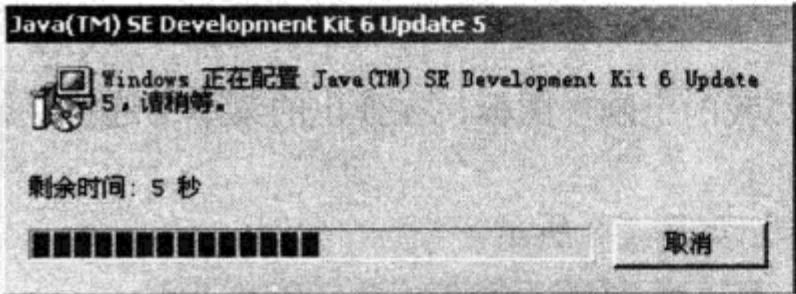


图 2-1 初始化安装界面

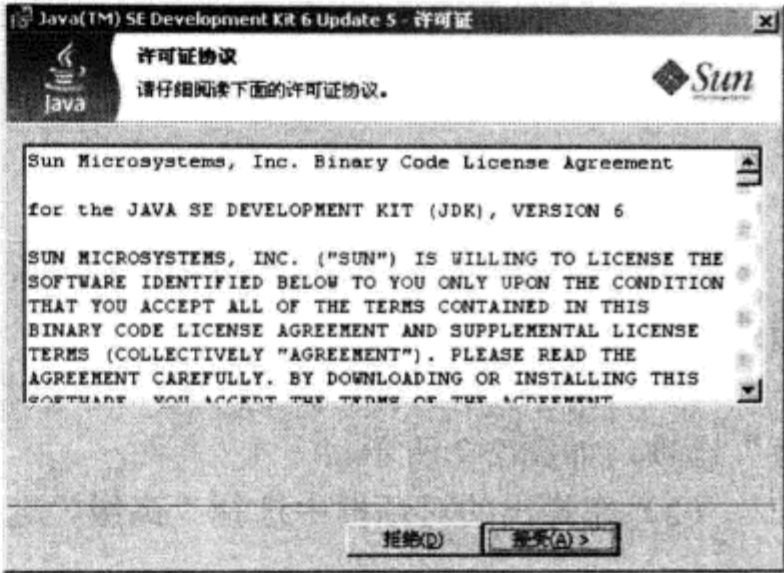


图 2-2 许可证协议对话框

- (3) 阅读安装许可协议后，单击“接受”按钮，将显示如图 2-3 所示的自定义安装对话框，在其中可以选择安装的组件以及更改安装路径。

(4) 单击“下一步”按钮，将自动开始 JDK 的安装，显示如图 2-4 所示的正在安装对话框。

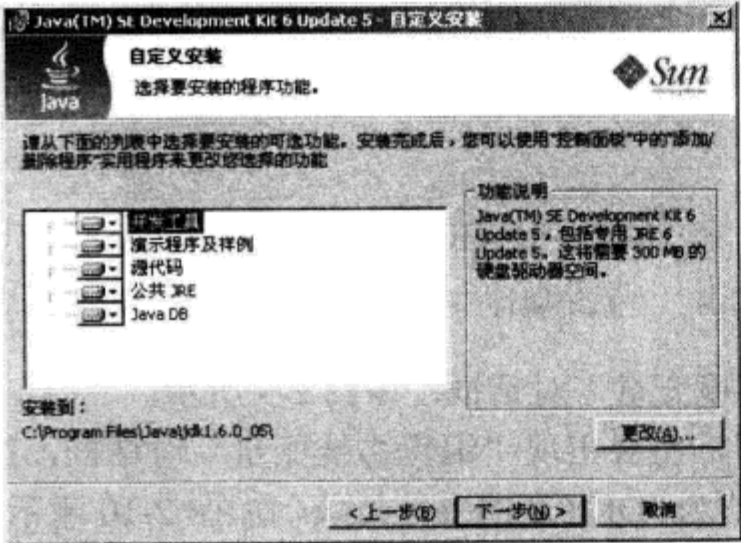


图 2-3 自定义安装对话框

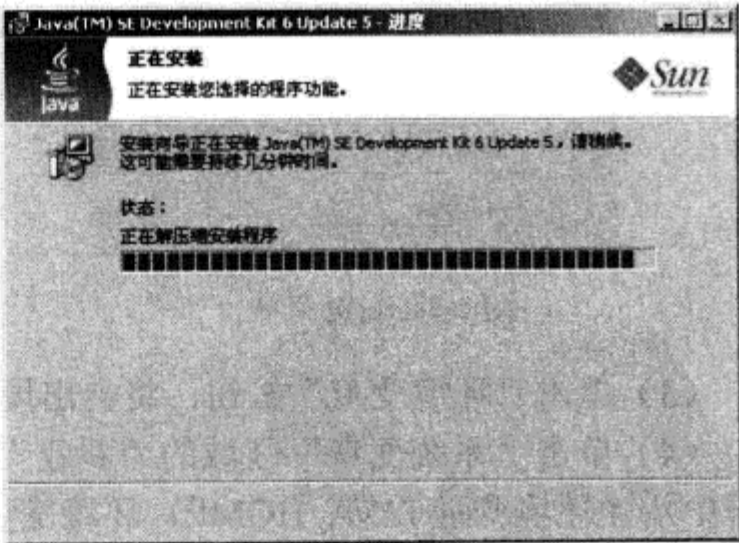


图 2-4 正在安装对话框

- (5) 在安装完 JDK 后自动进入到 JRE 的安装，JRE 的安装步骤同 JDK，将显示如图 2-5 所示的自定义安装对话框。

(6) 安装完成后将显示如图 2-6 所示的安装成功界面。
- 至此，已经成功地在系统中安装了 JDK 6.0。



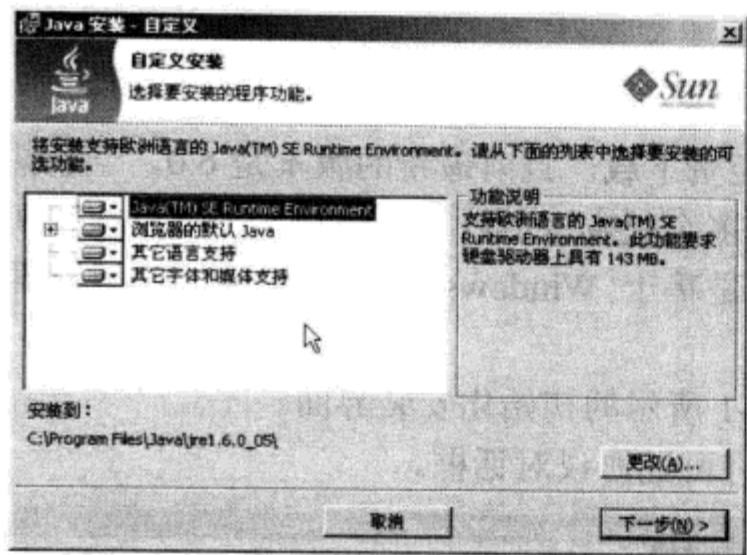


图 2-5 JRE 自定义安装对话框

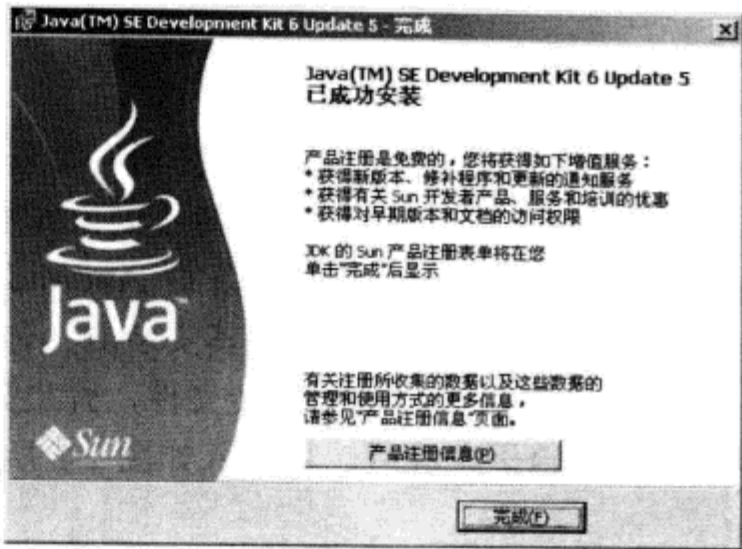


图 2-6 JDK 安装成功界面

2.2.2 配置 JDK

JDK 安装成功后，必须要对其进行配置，才能够正确使用。JDK 的配置主要是手动设置环境变量，具体步骤如下：

- (1) 在 Windows 操作系统中右击桌面上的“我的电脑”图标，在弹出的菜单中选择“属性”选项，如图 2-7 所示。
- (2) 在弹出的对话框中选择“高级”选项卡，如图 2-8 所示。

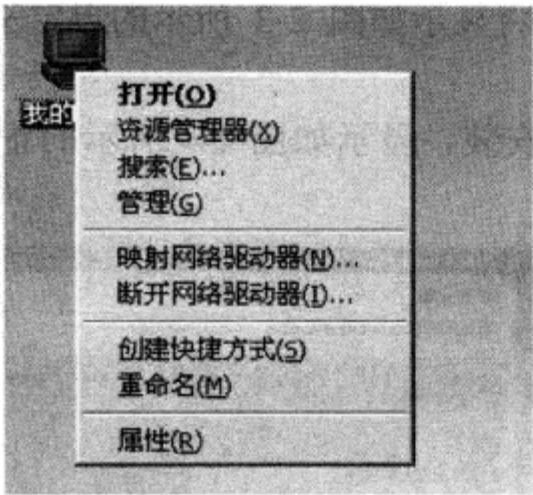


图 2-7 右键菜单

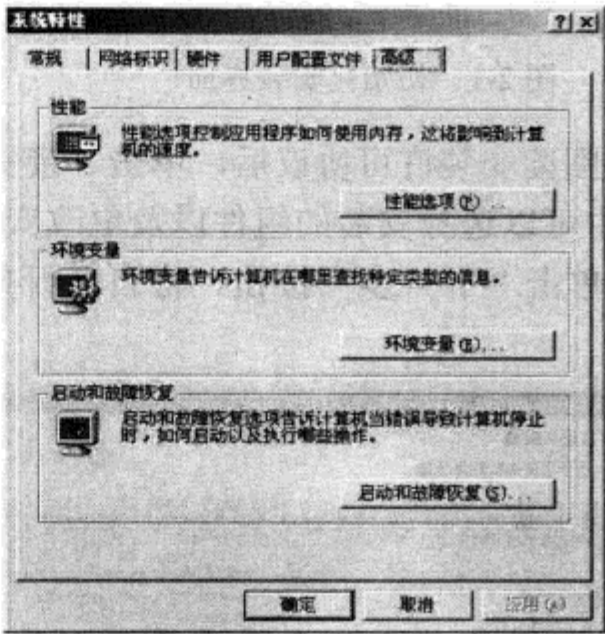


图 2-8 “系统属性”对话框的“高级”选项卡

- (3) 单击“环境变量”按钮，将会出现“环境变量”对话框，如图 2-9 所示。
- (4) 单击“系统变量”区域的“新建”按钮，将会出现“编辑系统变量”对话框，在对话框中新建环境变量 JAVA\_HOME，环境变量的值为 JDK 6.0 的安装目录，如图 2-10 所示。
- (5) 设置 JAVA\_HOME 环境变量后，在“系统变量”区域中选择“Path”环境变量，然后单击“编辑”按钮，弹出“编辑系统变量”对话框，在其中添加 JDK 6.0 安装目录下的 bin 子目录的路径，如图 2-11 所示。
- (6) JDK 安装配置完毕后，可以检验配置是否正确。在 Windows 操作系统中单击“开始”|“运行”，在其中输入“cmd”命令启动 DOS 窗口，如图 2-12 所示。
- (7) 在 DOS 窗口的命令提示符下输入“java -version”命令，出现如图 2-13 所示的运行界面，说明环境变量配置成功。

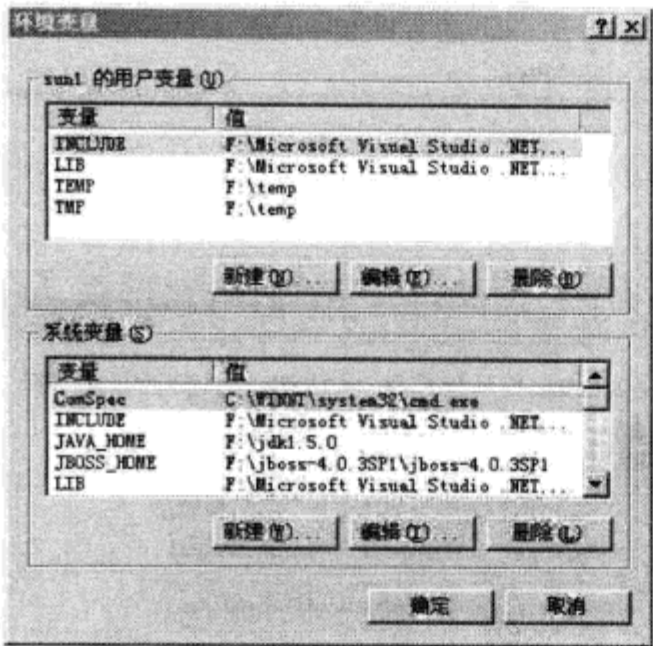


图 2-9 “环境变量”对话框

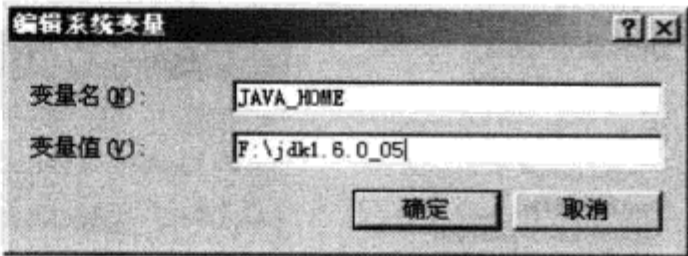


图 2-10 设置 JAVA\_HOME 环境变量

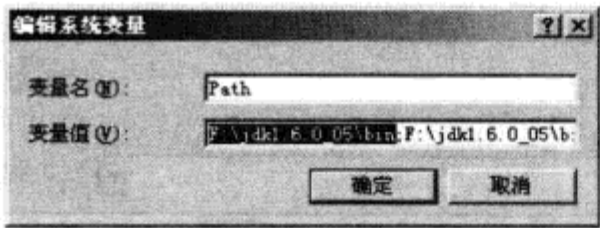


图 2-11 向 Path 环境变量添加 JDK 路径

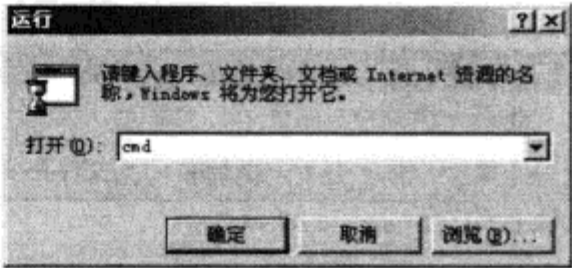


图 2-12 输入命令

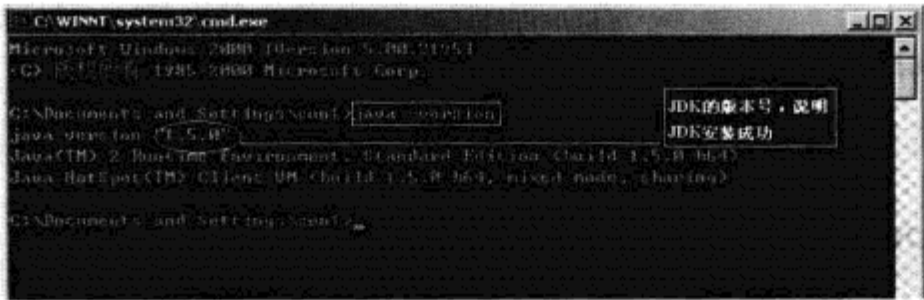


图 2-13 JDK 6.0 安装配置成功

### 2.3 Android SDK 的下载和安装

有过软件开发经验的读者可能都知道，SDK 是软件开发工具包，是进行软件开发的基础。与其他开发工具的 SDK 一样，Android SDK 也是进行 Android 应用程序开发的基础，所以要进行 Android 应用程序开发，必须首先在系统中安装 Android SDK。

#### 2.3.1 下载 Android SDK

Android SDK 的官方开发网站是 <http://developer.android.com>，可以从该网站下载最新版的 Android SDK，目前最新版本是 2.1。Android SDK 的下载页面如图 2-14 所示。

选择对应的操作系统所使用的 Android SDK 后，单击对应的链接，就将开始下载，下载后的 Android SDK 为压缩文件，请将它解压缩到磁盘中。解压缩后的 Android SDK 目录如图 2-15 所示。



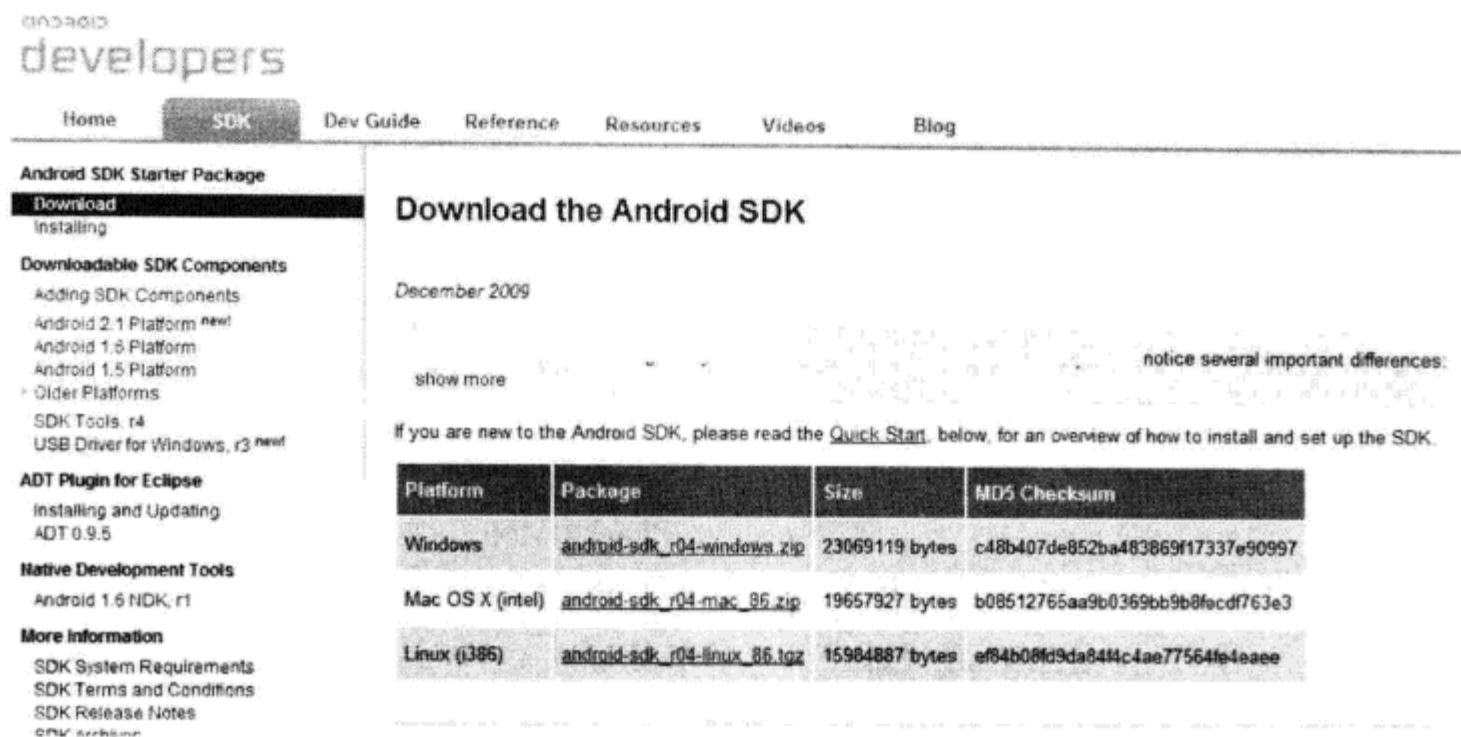


图 2-14 Android SDK 下载页面

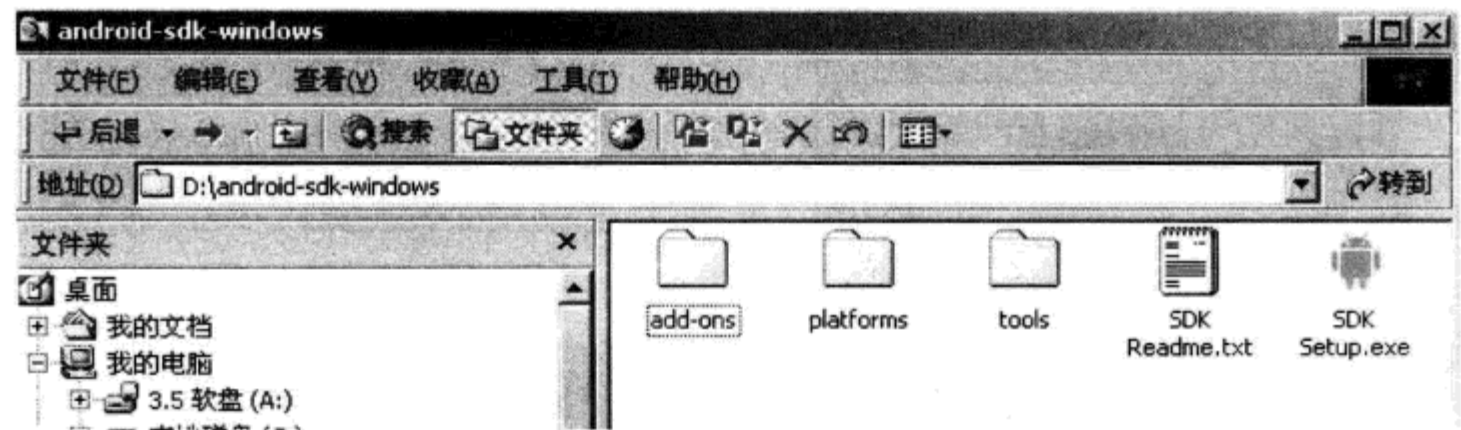


图 2-15 解压缩后的 Android SDK 目录

2.3.2 安装 Android SDK

- (1) 双击 Android SDK 目录中如图 2-16 所示的 Setup 程序就将开始安装 Android SDK。
- (2) 安装开始将显示如图 2-17 所示的更新资源窗口。



图 2-16 Android SDK 安装程序

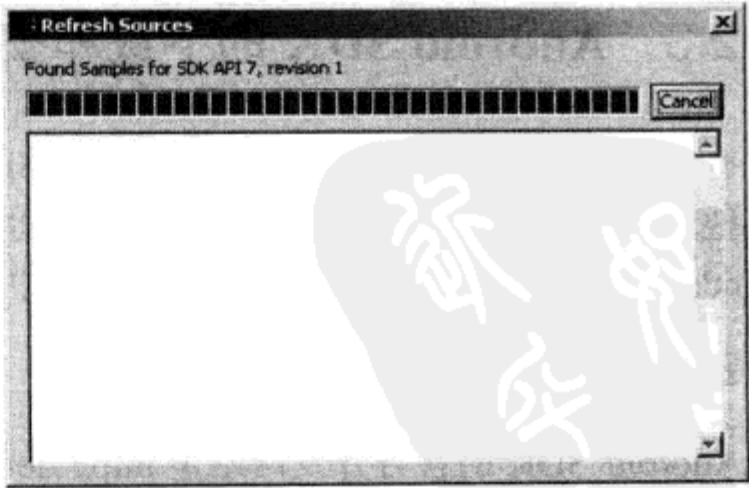


图 2-17 更新资源窗口

(3) 如果在更新过程中遇到消息为“Failed to fetch URL...”的错误提示，那么就需要将获取更新信息的协议由 HTTPS 方式改为 HTTP 方式。这就需要在“Android SDK and AVD Manager”窗口的左侧选择“Settings”选项，然后在窗口的右侧选择“Force https://...”选项，

如图 2-18 所示，这样将重新更新资源。

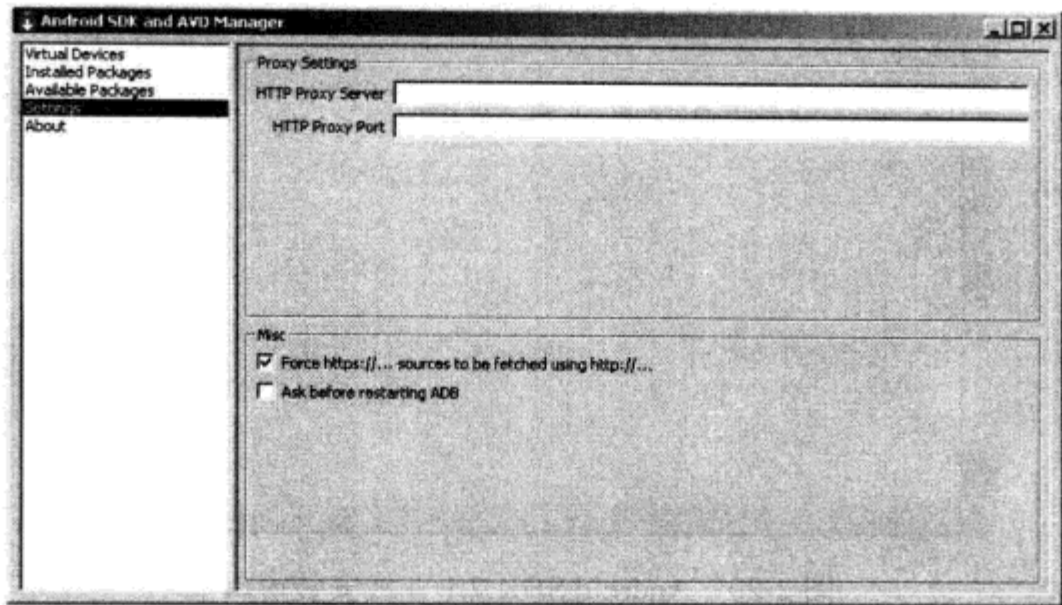


图 2-18 更改更新资源的协议方式

(4) 当成功获取要更新的资源信息后，将显示如图 2-19 所示的选择要安装的包窗口。在该窗口中可以选择要安装的 API 版本、驱动、文档以及实例代码等。如果只要使用 Android 2.1 的模拟器，那么只选择“SDK Platform Android 2.1, API 7, revision 1”即可，如果要使用 SDK 开发应用程序和游戏应用，那么就需要选择“Accept All”选项，接受并遵守所有许可内容。然后单击“Install”按钮开始下载所选许可内容。

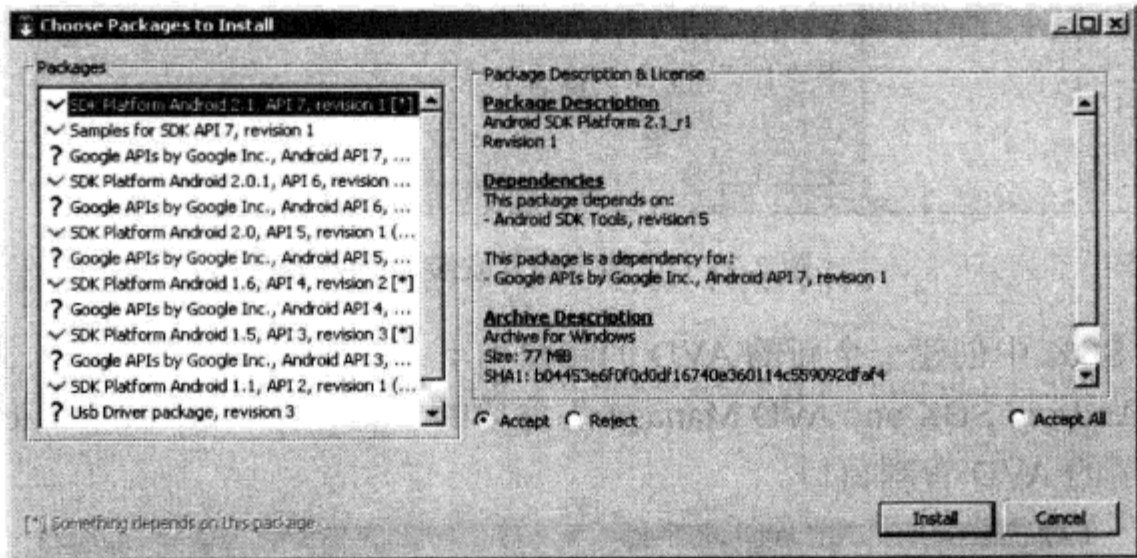


图 2-19 选择要安装的包

(5) Android SDK 2.1 需要在线下载安装，安装窗口如图 2-20 所示。

(6) 下载并安装完成后，在如图 2-21 所示的“Android SDK and AVD Manager”窗口的左侧选择“Installed Packages”选项，将在窗口右侧显示出已经正确下载并安装好的包。

2.3.3 创建 Android 虚拟设备

在 Android SDK 1.5 版本以后的 Android 开发中，必须创建至少一个 AVD，AVD 的全称为 Android Virtual Device（Android 虚拟设备），每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核、系统图像和数据分区，还可以有自己的 SD 卡 and 用户数据以及外观显示等。所以在 Android SDK 安装完成之后，必须创建 AVD，才能够使用 SDK 来开发和运行 Android 应用程序。



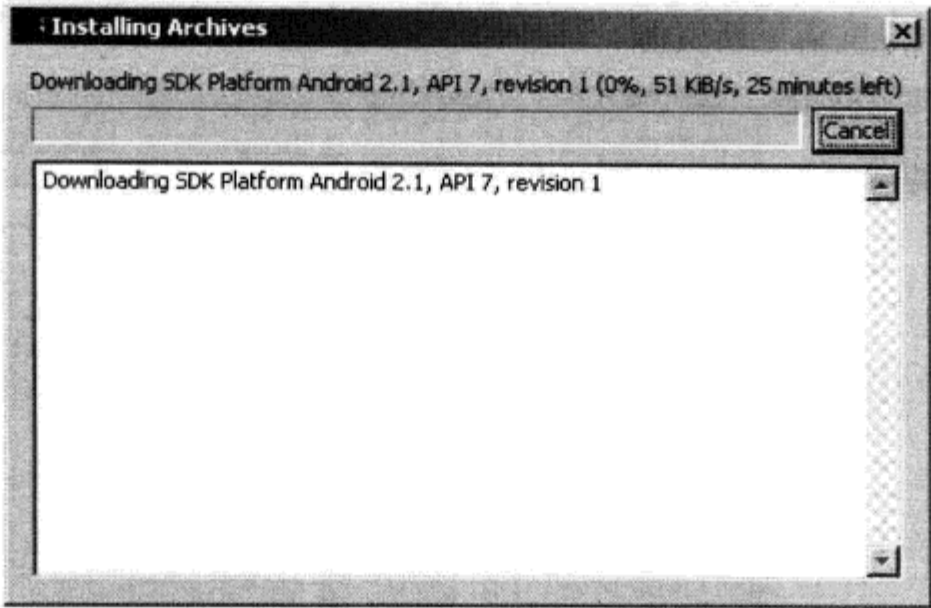


图 2-20 Android SDK 2.1 在线安装窗口

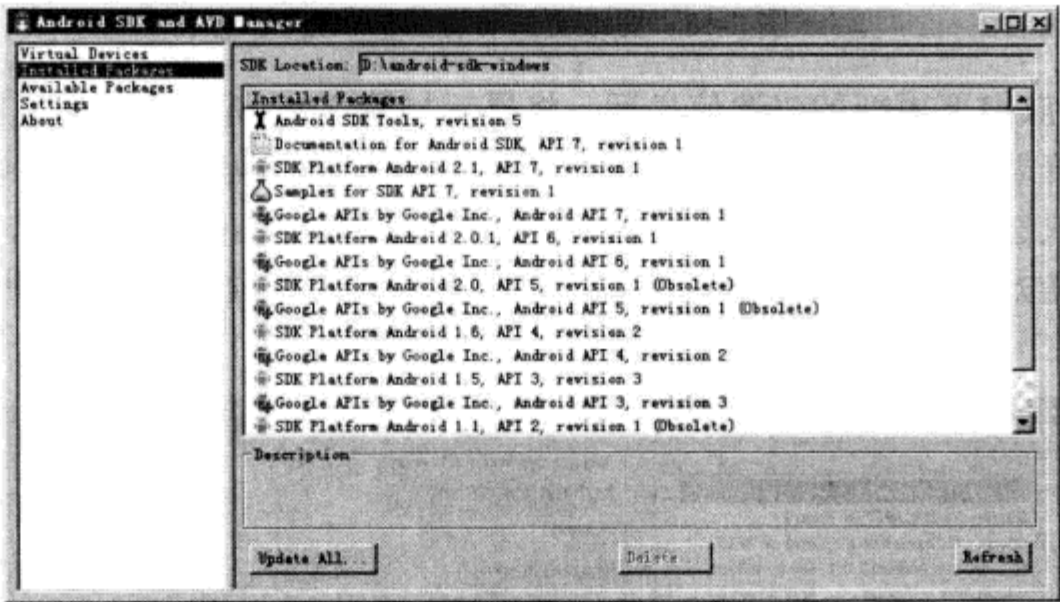


图 2-21 正确下载并安装好的包

在 Android SDK 中创建一个新的 AVD 的步骤如下：

（1）在“Android SDK and AVD Manager”窗口的左侧选择“Virtual Devices”选项，将显示如图 2-22 所示的 AVD 管理窗口。

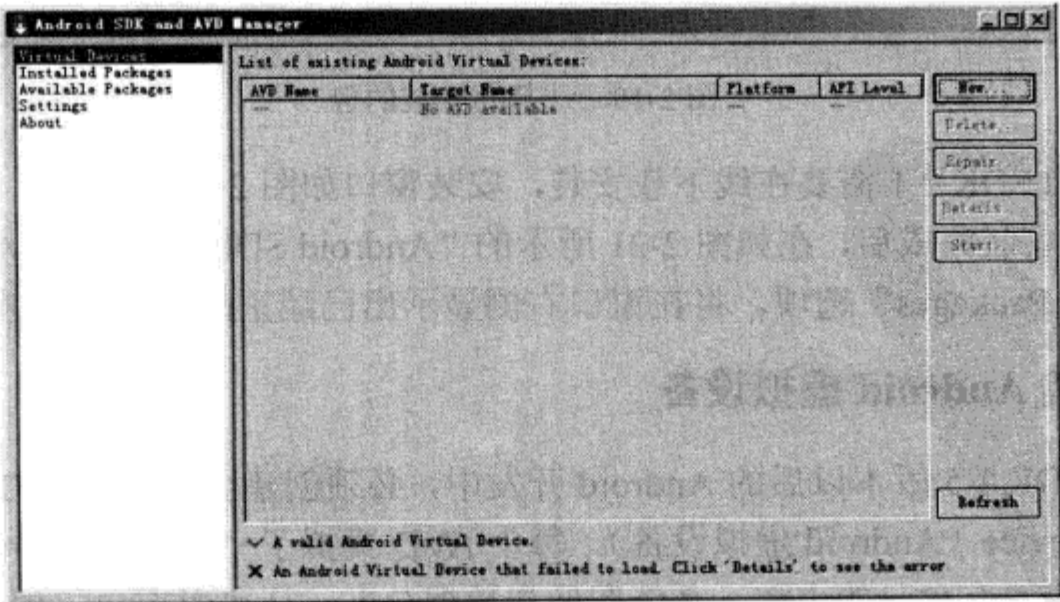


图 2-22 AVD 管理窗口

（2）单击“New...”按钮，将弹出如图 2-23 所示的创建新的 AVD 窗口。在“Name”文

本框中输入新创建的 AVD 的名字，在“Target”下拉列表框中选择目标平台版本规范，在“SD Card”中设置模拟的 SD Card 的容量大小，在“Hardware”中设置模拟的硬件设备的属性参数。

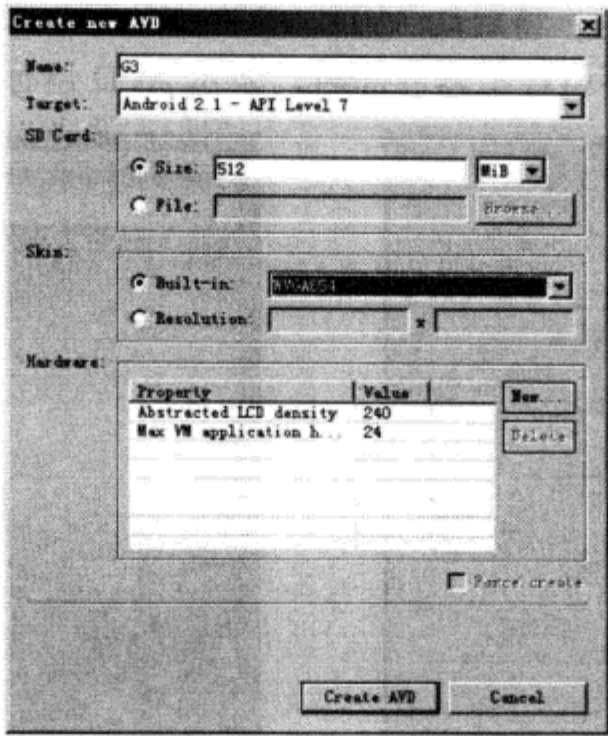


图 2-23 创建新的 AVD 窗口

(3) 设置完成后，单击“Create AVD”按钮，将显示如图 2-24 所示的创建 AVD 成功提示框。

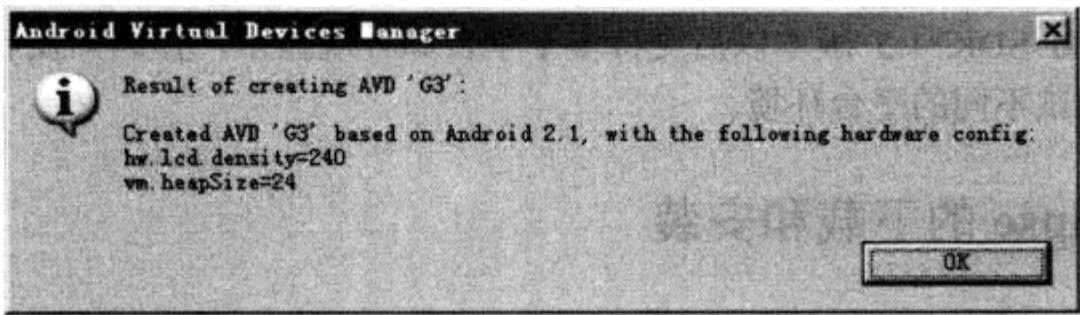


图 2-24 创建 AVD 成功提示框

(4) 新的 AVD 创建成功后，将会出现在如图 2-25 所示的现存的 AVD 列表中。

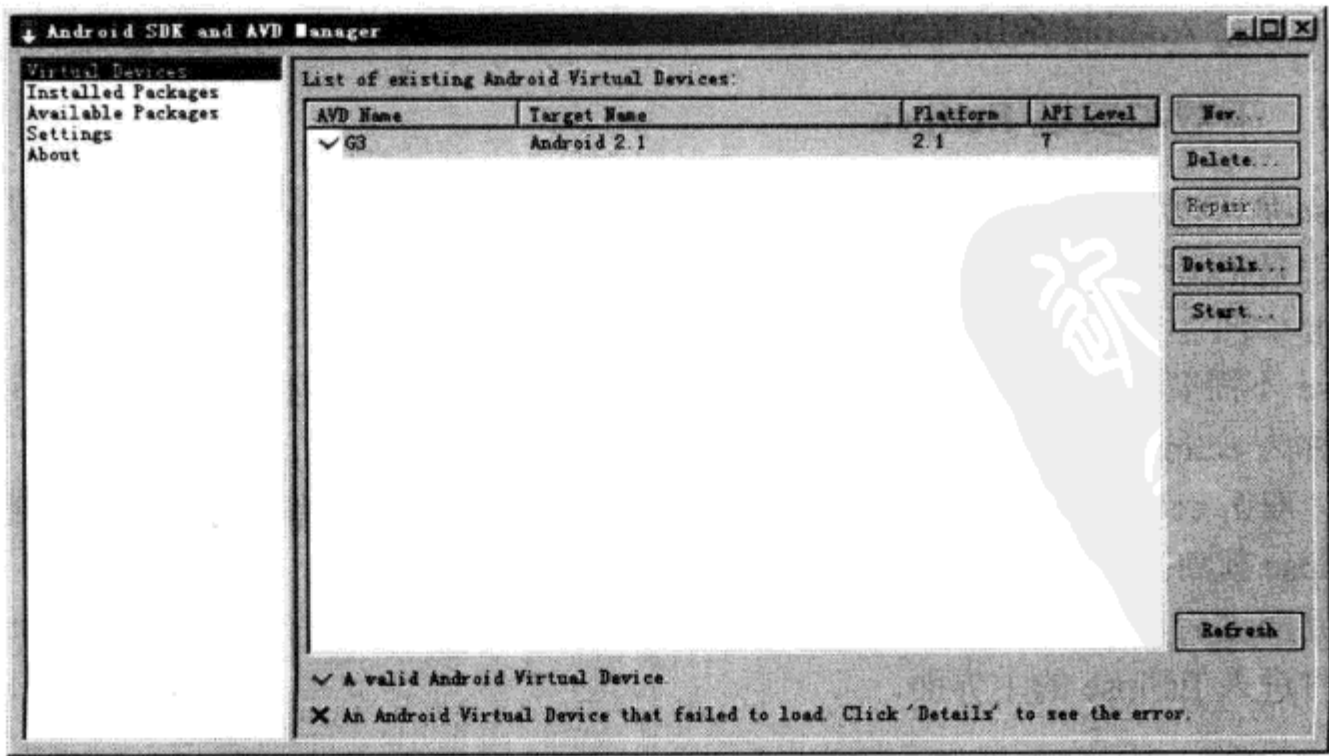


图 2-25 现存的 AVD 列表



(5) 选中要运行的 AVD，单击“Start”按钮，将显示如图 2-26 所示的加载运行选项对话框。在该对话框中可以设置 AVD 运行时的外观大小等属性。

(6) 单击“Launch”按钮，将启动 AVD，AVD 首次运行较慢，需要几分钟的时间。当显示如图 2-27 所示的 Android 手机模拟器界面时，表示 AVD 运行成功。

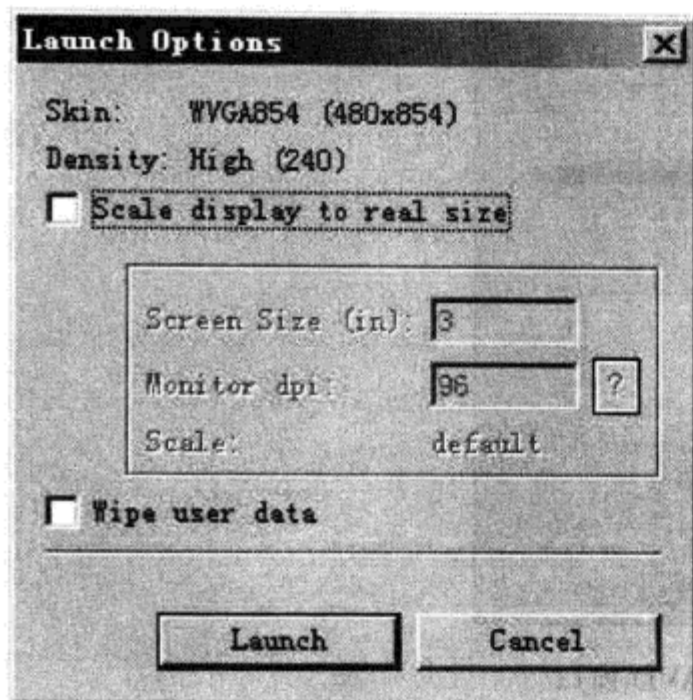


图 2-26 AVD 加载运行选项对话框

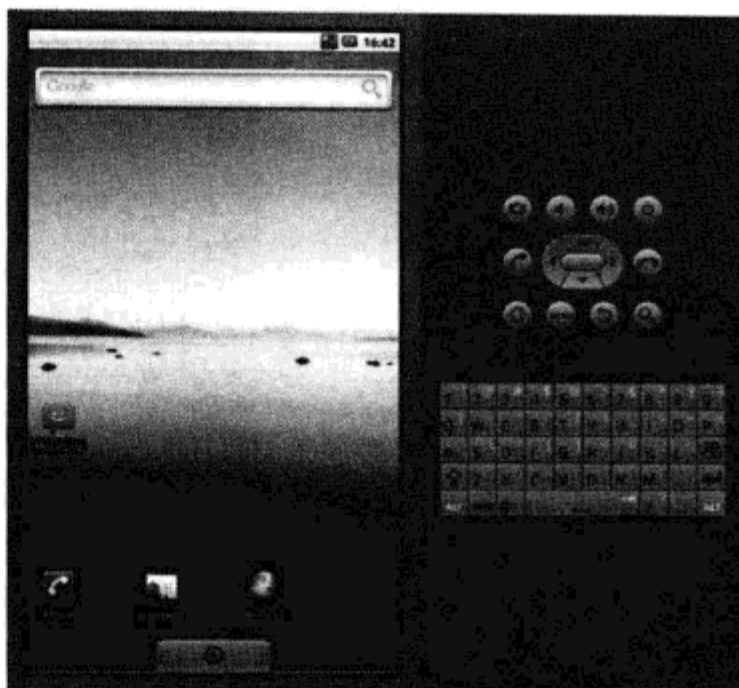


图 2-27 Android 手机模拟器界面

由于 Android SDK 1.5 版本以后支持多个平台和外观显示，作为开发者可以创建不同的 AVD 来模拟和测试不同的平台环境。

## 2.4 Eclipse 的下载和安装

虽然在正确安装 Android SDK 之后，就可以进行 Android 应用程序的开发。但是 Android SDK 仅仅提供了 Android 应用程序的编译和执行工具，并没有提供程序代码编写的环境。通过使用 Android 的 Eclipse 插件 ADT (Android Development Tools)，就可以在强大的 Eclipse 集成开发环境中构建 Android 应用程序。

### 2.4.1 下载和安装 Eclipse

Eclipse 是可以免费使用的软件，从 Eclipse 的官方网站 <http://www.eclipse.org> 可以直接下载。本书使用的是 Eclipse 的最新版本 3.5。该版本需要在 JDK 5.0 及以上版本下运行。Eclipse 3.5 的下载页面如图 2-28 所示。

Eclipse 不需要安装，下载完成后，将 Eclipse 压缩文件直接解压到某个路径下面，解压后的目录名称为 eclipse。

然后，双击 eclipse 文件夹中的可执行文件 eclipse.exe，如果系统中已经正确安装和配置过 JDK，Eclipse 就将正确启动，显示如图 2-29 所示的启动界面。

启动界面过后，将显示如图 2-30 所示的对话框，在其中设置工作台的路径后，单击“OK”按钮，就将进入 Eclipse 的主界面。



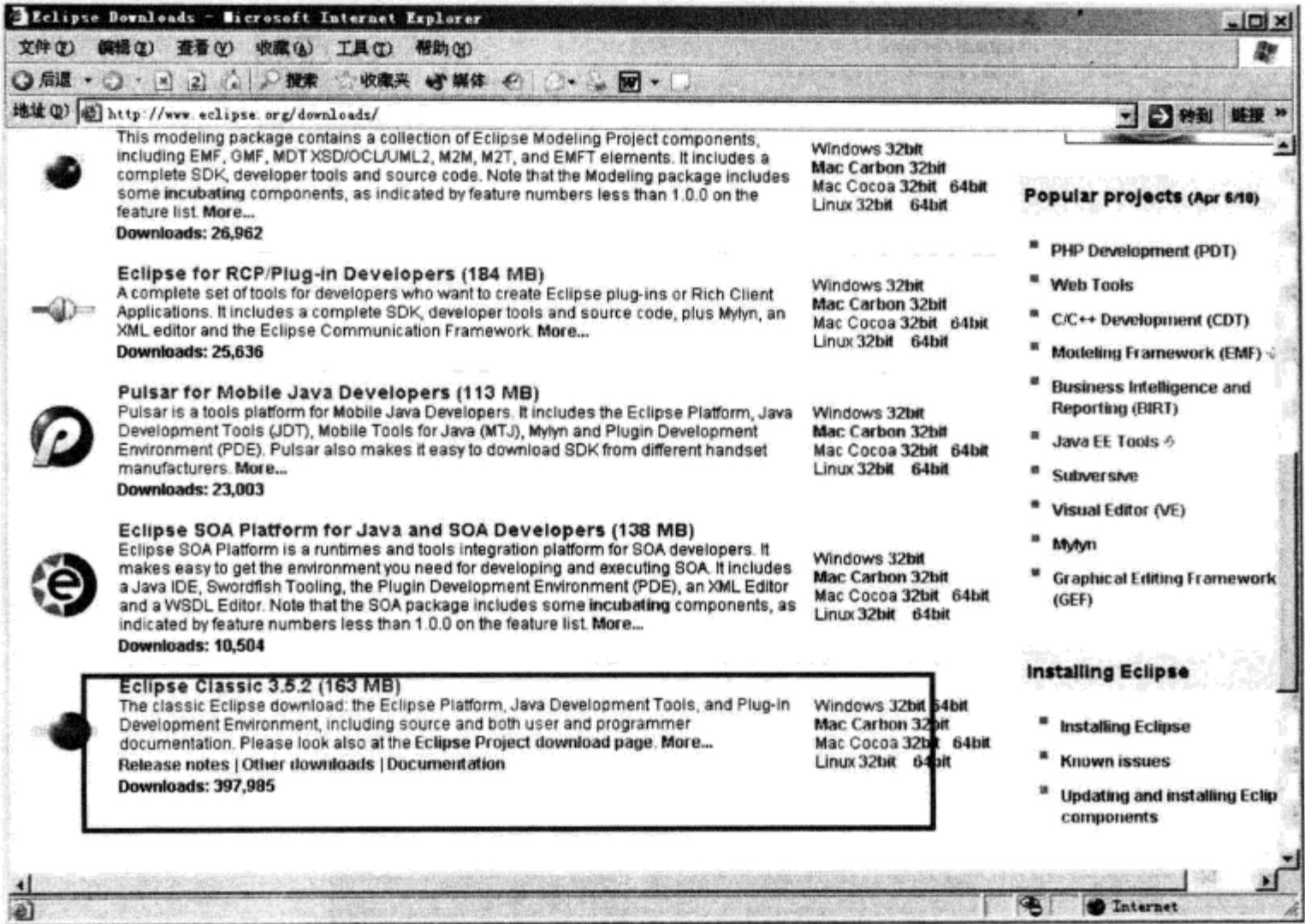


图 2-28 Eclipse 3.5 的下载页面

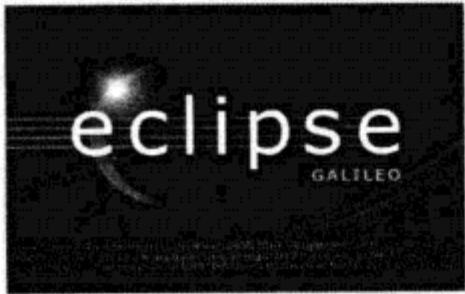


图 2-29 Eclipse 启动界面

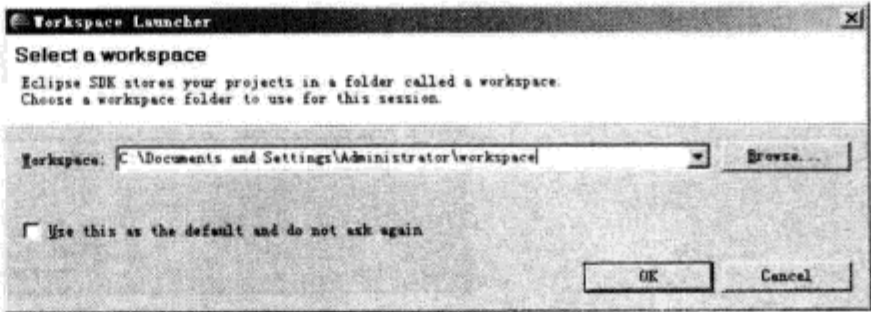


图 2-30 设置 Eclipse 工作台的路径

2.4.2 安装和配置 Eclipse 中的 Android 插件

要使 Eclipse 支持 Android 应用程序开发，必须首先在 Eclipse 中安装 Android 插件 ADT，具体安装和配置过程如下：

- (1) 单击 Eclipse 菜单栏中的“Help”|“Install New Software...”命令，将显示如图 2-31 所示的软件安装窗口。
- (2) 单击“Add...”按钮，将弹出如图 2-32 所示的添加站点对话框。其中，在“Name”文本框中输入站点的名称，在“Location”文本框中输入站点的地址。
- (3) 安装 ADT 插件时，可以选择在线安装，也可以选择将插件下载到本地进行安装。在线安装时，在“Location”文本框中输入如下地址：<http://dl-ssl.google.com/android/eclipse/>，本地安装时，单击“Archive...”按钮，选择对应的 zip 插件文件。本书采用的是本地安装方式，选择插件后如图 2-33 所示。
- (4) 设置完成后，单击“OK”按钮，将显示安装向导窗口，该窗口中包括可以安装的内容，如图 2-34 所示。



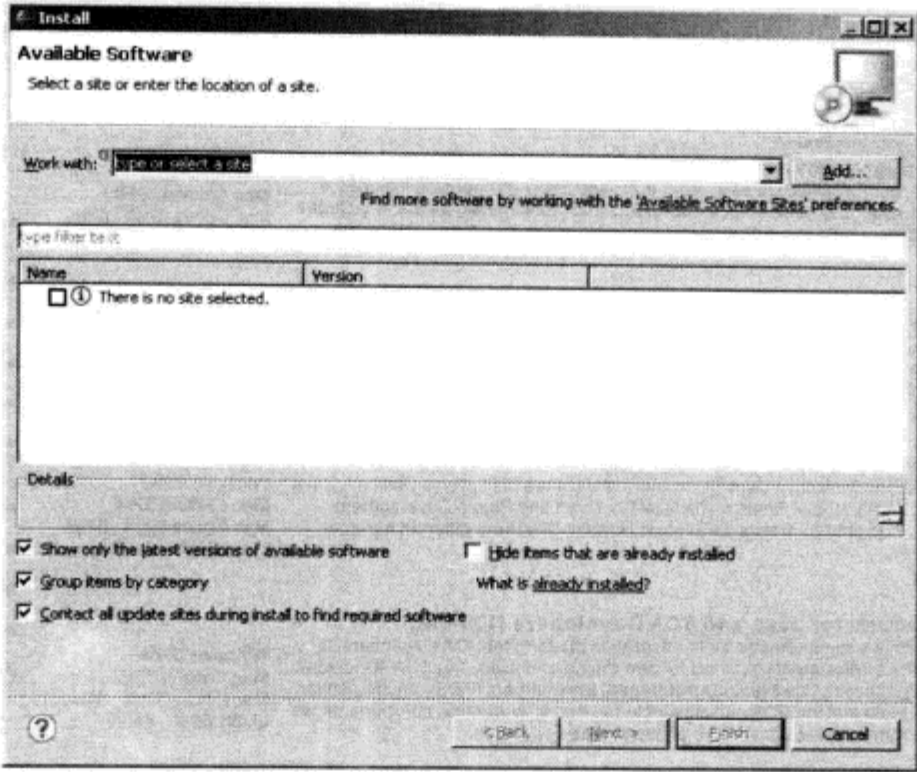


图 2-31 软件安装窗口

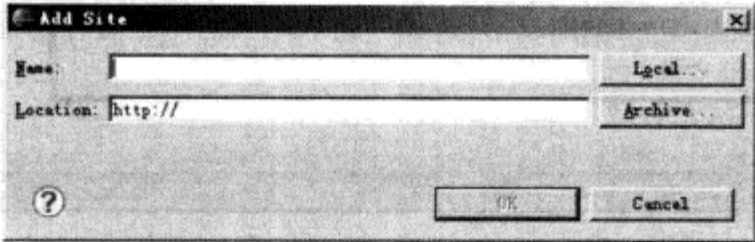


图 2-32 添加站点对话框

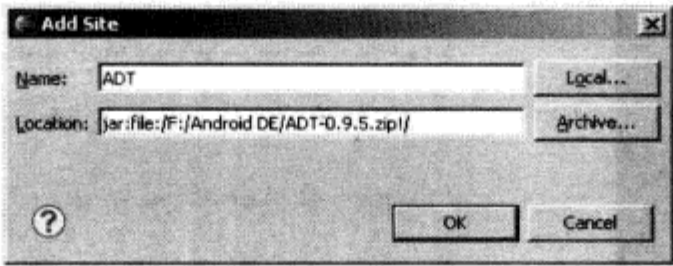


图 2-33 本地安装 ADT 插件

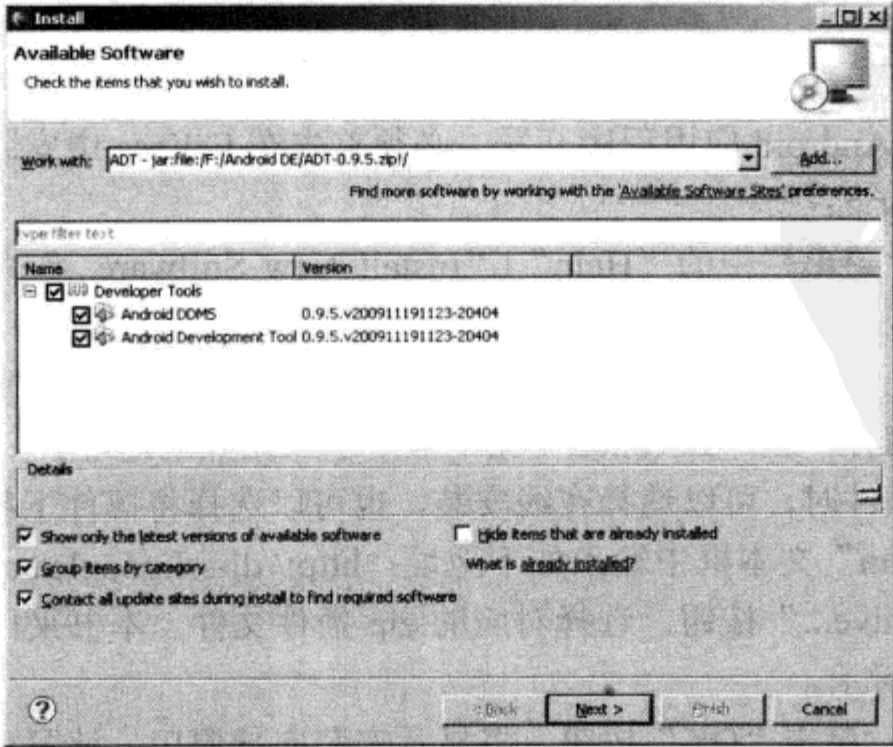


图 2-34 显示要安装的内容

(5) 在选中要安装的内容之后，单击“Next”按钮，将显示如图 2-35 所示的检查安装软件的信息窗口。该窗口显示两项必须安装的程序：Android Development Tools 与 Android DDMS。

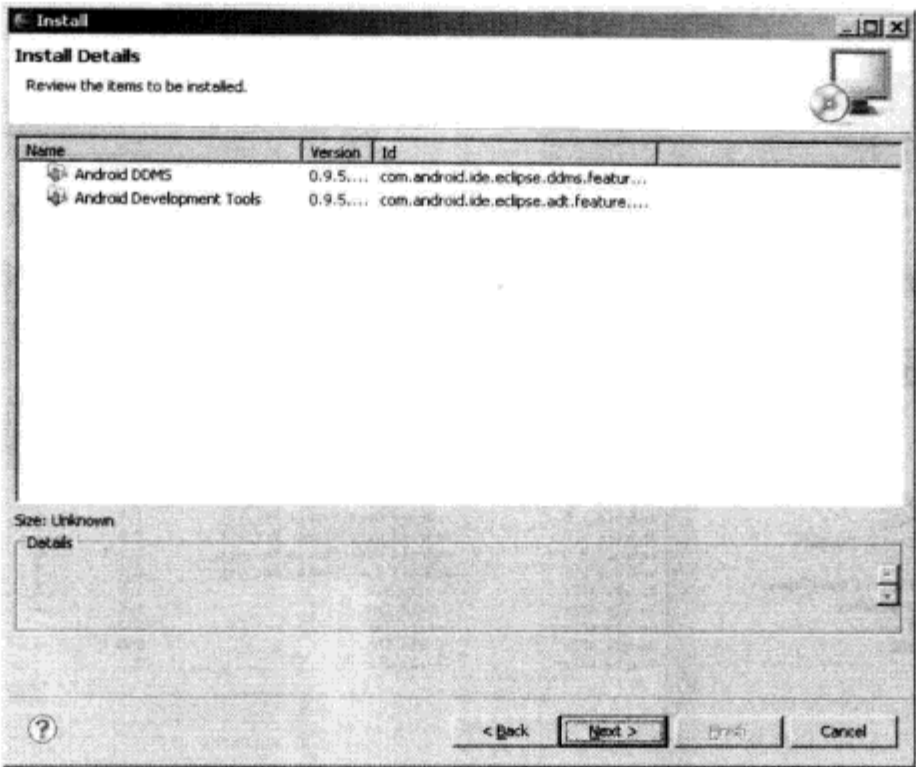


图 2-35 检查安装软件的信息窗口

(6) 确认无误后，单击“Next”按钮，将显示如图 2-36 所示的安装授权协议窗口。

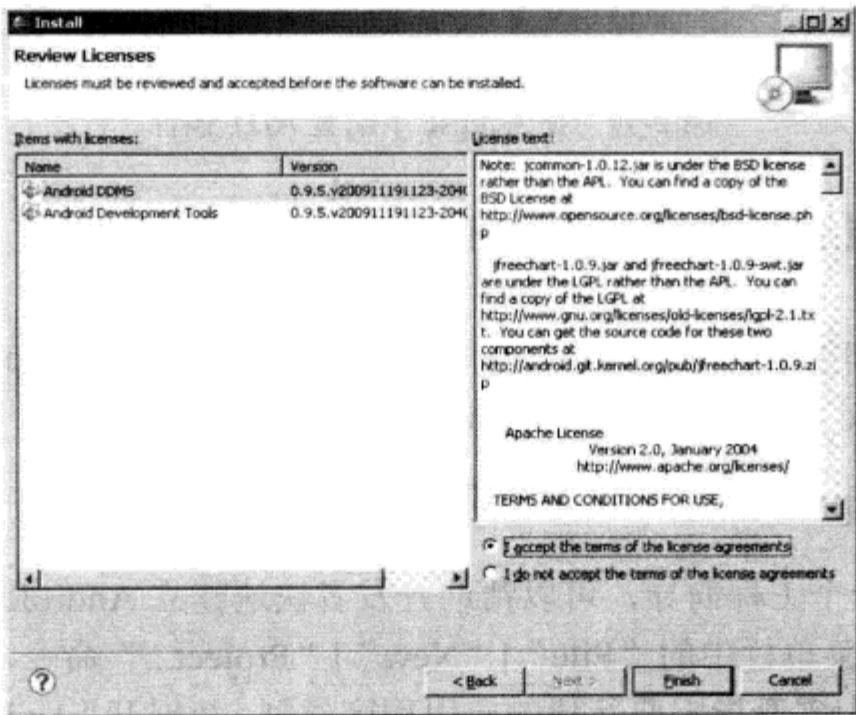


图 2-36 安装授权协议窗口

(7) 选择“I accept the terms of the license agreements”选项，接受安装协议后，单击“Finish”按钮，将开始安装 ADT 插件，并显示如图 2-37 所示的安装进度窗口。

(8) 当安装完成后，将显示如图 2-38 所示的软件更新成功对话框，单击“Yes”按钮重新启动 Eclipse，则安装在 Eclipse 中的 ADT 插件将生效。

(9) ADT 插件安装完成之后，需要在 Eclipse 中进行配置。单击 Eclipse 菜单栏中的“Window”|“Preferences”命令，在弹出的如图 2-39 所示的窗口的左侧选择“Android”选项，在窗口的右侧选择 Android SDK 的安装目录，这时下面的列表中将显示所有可用的目标平台版本规范。至此所有准备工作都已经就绪，随时可以开始建立 Android 项目。



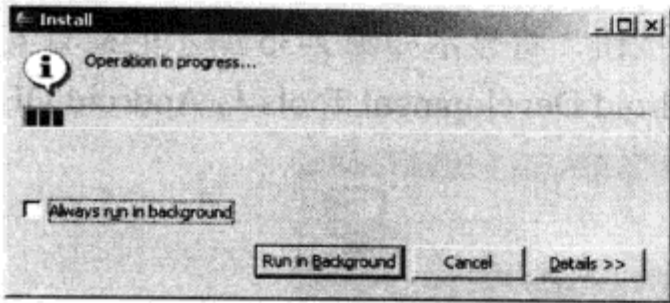


图 2-37 安装进度窗口

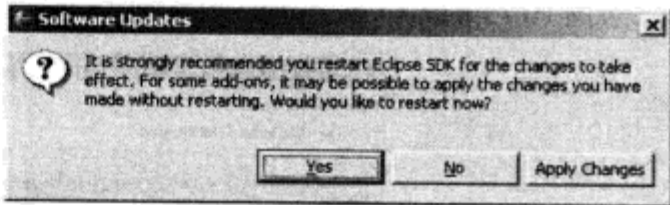


图 2-38 软件更新成功对话框

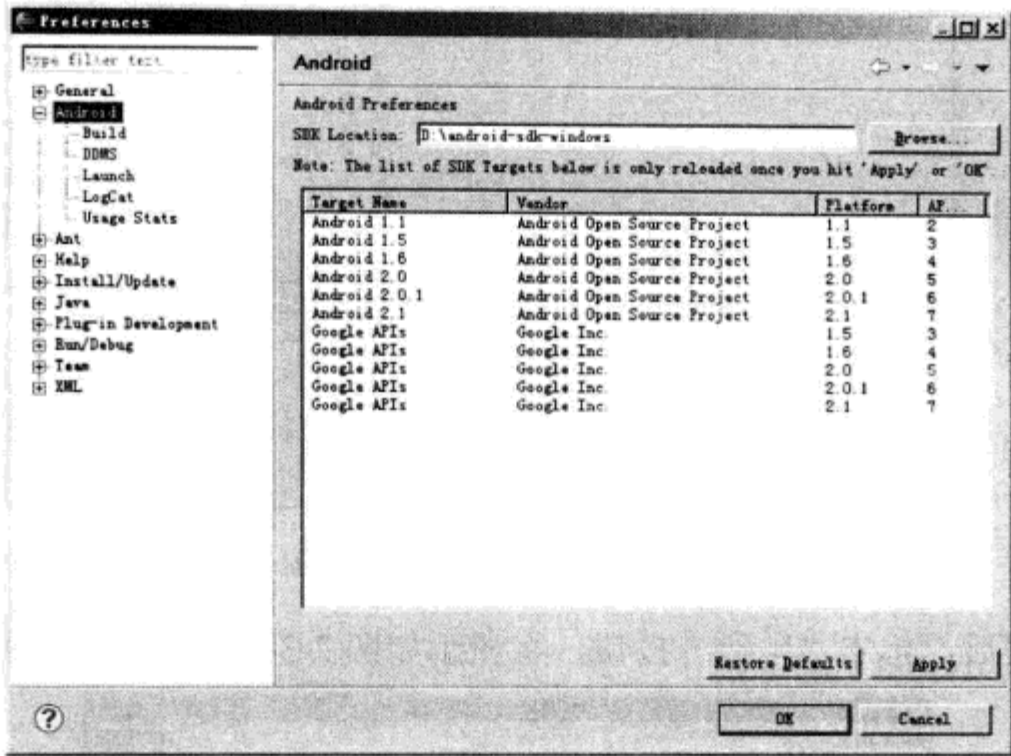


图 2-39 在 Eclipse 中配置 ADT 插件

2.5 使用 Eclipse 开发 Android 应用程序

在 Eclipse 中安装和配置 Android 插件成功之后，就可以使用 Eclipse 开发 Android 应用程序了，并可以对 Android 应用程序进行调试和运行。

2.5.1 使用 Eclipse 创建 Android 项目

ADT 插件提供了一个工程向导，可以帮助开发者快速建立 Android 项目，具体步骤如下：

- （1）单击 Eclipse 菜单栏中的“File”|“New”|“Project...”命令，将显示如图 2-40 所示的新建项目向导。在其中需要指定要新建的应用程序类型，故展开“Android”后，单击“Android Project”，表示要新建 Android 项目。
- （2）单击“Next”按钮，将显示如图 2-41 所示的新建 Android 项目窗口，在该窗口的“Project name”框中输入项目名称，在“Build Target”列表中选择要使用的目标设备版本，在“Properties”区域设置应用程序的名称、项目中类的包名以及项目中 Activity 的名称。
- （3）单击“Finish”按钮，就可完成 Android 项目的创建。

2.5.2 Eclipse 中 Android 项目架构

在 Eclipse 的“Package Explorer”透视图中，新创建的 Android 项目的架构如图 2-42 所示。通过该透视图可以看到，项目的根目录 HelloWorld 中包含一些自动生成的文件和文件夹，

它们是组成 Android 应用程序的必需部分，它们在应用程序中所起到的作用和主要功能如下：

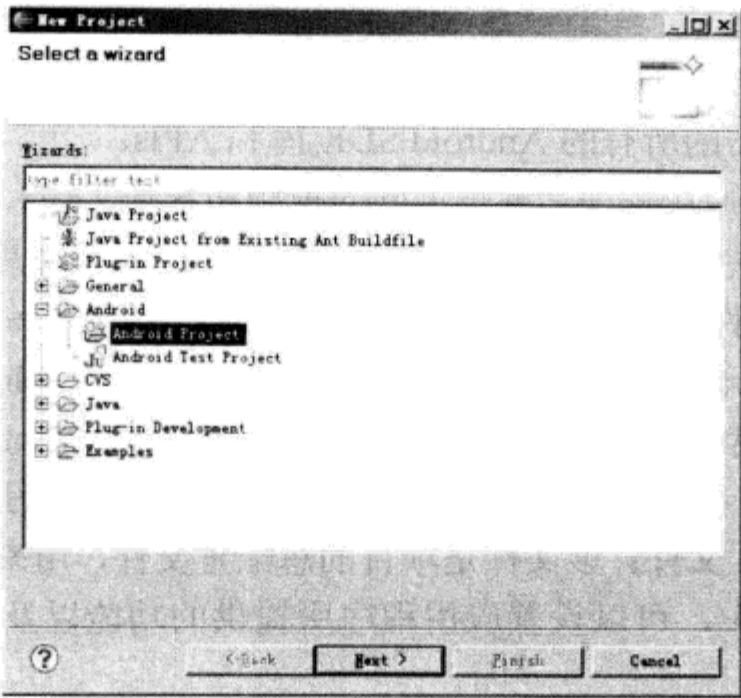


图 2-40 选择项目类型为 “Android Project”

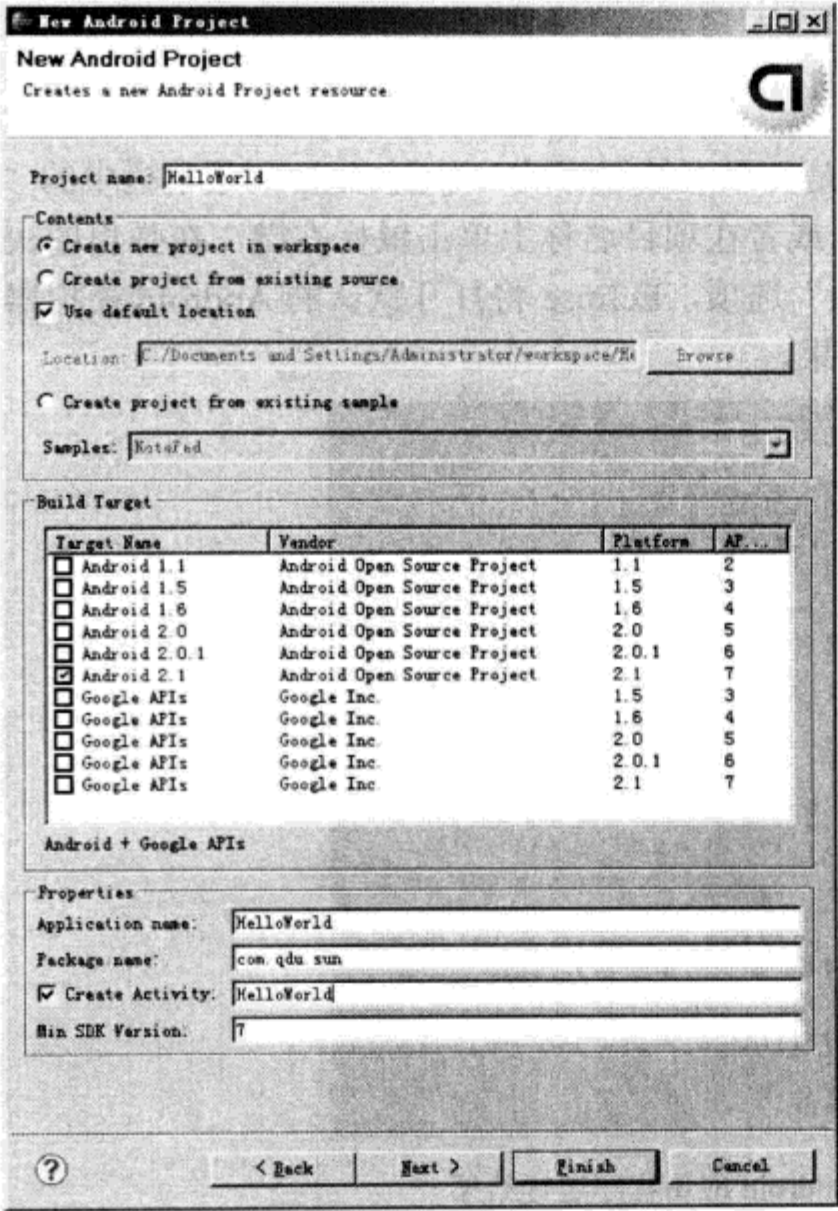


图 2-41 新建 Android 项目窗口



图 2-42 Android 项目架构

- **src 文件夹：**该文件夹用来存放项目中的所有源文件，当项目刚创建时，该文件夹中包含 activity 的源文件，以后用户创建的所有源文件也都将存放在该文件夹中。
- **gen 文件夹：**该文件夹中包含一个在创建项目时自动生成的 R.java 文件，该文件是只读



模式的，不能手动更改。R.java 文件中包含很多静态类，这些静态类用来表示项目中所有资源的引用。

- **Android 2.1 文件夹**：该文件夹中包含 android.jar 文件，这是一个 Java 归档文件，其中包含构建应用程序所需的所有的 Android SDK 库和 APIs。
- **assets 文件夹**：包含应用程序需要使用到的视频和音频文件。
- **res 文件夹**：该文件夹是资源目录，包含项目中的资源文件并将其编译进应用程序。向此目录添加资源文件时，会被 R.java 自动记录。该文件夹下会有 5 个子文件夹：drawabel-hdpi、drawabel-ldpi、drawabel-mdpi、layout 和 values。其中，三个 drawabel 开头的文件夹中包含一些应用程序中使用的图标文件，layout 文件夹中包含界面布局文件 main.xml，values 文件夹中包含程序中要使用到的字符串引用文件 strings.xml。
- **AndroidManifest.xml 文件**：该文件是项目的总配置文件，用来配置应用中所使用的各种组件。在这个文件中，可以设置应用程序所提供的功能以及应用程序使用到的服务和 Activity。
- **default.properties 文件**：该文件负责记录项目所需要的环境信息，例如 Android 的版本信息等。

### 2.5.3 Eclipse 中 Android 项目的调试和运行

使用 Eclipse 新创建的 Android 项目是可以直接运行的。单击 Eclipse 菜单栏中的“Run”|“Run As”|“Android Application”命令或者在项目名称上单击鼠标右键，在弹出的快捷菜单中选择“Run As”|“Android Application”选项，Eclipse 将打开默认的 Android 模拟器，显示如图 2-43 所示的 HelloWorld 程序运行结果。

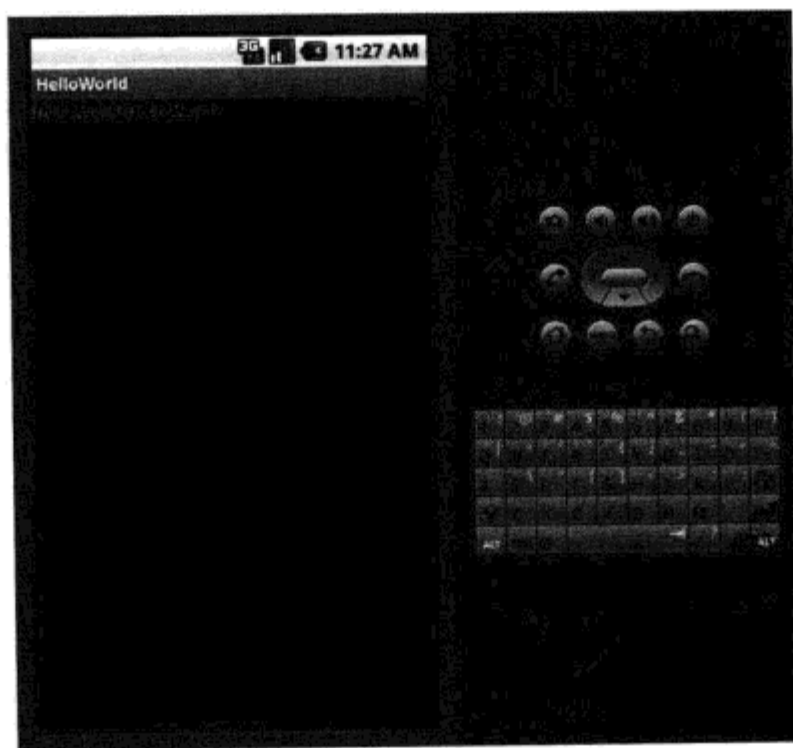


图 2-43 Android 应用程序运行结果

如果要对 Android 应用程序进行调试，可以单击 Eclipse 菜单栏中的“Run”|“Debug As”|“Android Application”命令或者在项目名称上单击鼠标右键，在弹出的快捷菜单中选择“Debug As”|“Android Application”选项，就将进入程序调试过程。

## 第 3 章 Android 中的 Activity

Activity 是 Android 应用的基本组成单位。在本章中，将创建一个完整的单 Activity 的 Android 应用。通过该应用，详细介绍 Activity 的程序结构和生命周期，并在此基础上讲解应用程序界面设计的两种方式。通过本章的学习，读者将对 Android 应用，特别是对应用中的 Activity 有更深层的认识。

### 3.1 Activity 的作用

在 Android 应用程序中，Activity 主要负责创建显示窗口，一个 Activity 通常就代表了一个单独的屏幕。它是用户唯一可以看得到的东西，所以几乎所有的 Activity 都是用来与用户进行交互的。对于熟悉 Windows 编程或者 Java ME 编程的读者来说，可以将 Activity 理解为 Windows 编程中的 WinForm 类或者 Java ME 编程中的 Display 类。

在具体实现时，每个 Activity 都被定义为一个独立的类，并且以 Android 中的 `android.app.Activity` 作为基类。在 Activity 类中将使用 `setContentView (View)` 方法来显示由视图控件组成的用户界面，并对用户通过这些视图控件所触发的事件做出响应。

大多数的应用程序根据功能的需要都是由多个屏幕显示组成，因此大部分的 Android 应用中也就必须包含多个 Activity 类。这些 Activity 可以通过一个 Activity 栈来进行管理。当一个新的 Activity 启动的时候，它首先会被放置在 Activity 栈顶部并成为运行状态的 Activity，而之前正在运行的 Activity 也在 Activity 栈中，它将被保存在这个新的 Activity 的下面，只有当这个新的 Activity 退出以后，之前的 Activity 才能重新回到前台界面。

由于多个 Activity 之间必然存在着较为复杂的跳转和切换的关系，所以接下来，本章将首先以一个单 Activity 的 Android 应用为例，详细介绍 Activity 的相关知识。

### 3.2 单 Activity 的 Android 应用

在 Android 应用中，如果需要包括要显示的界面，则在应用中至少应包含一个 Activity 类。本节将使用 Eclipse 创建一个名称为 HelloWorldText 的单 Activity 的 Android 应用，通过对该应用的分析来详细说明 Activity 类的程序结构和生命周期。

#### 3.2.1 Activity 的生命周期

为了更好地理解 Activity 类的结构，有必要首先熟悉一下 Activity 的生命周期。

Activity 共有四种状态，分别是：

- 激活或者运行状态，这时 Activity 运行在屏幕的前台。
- 暂停状态，这时 Activity 失去焦点，但是仍然对用户可见（例如这个 Activity 之上遮挡了一个透明的或者非全屏的 Activity）。



- 停止状态，这时 Activity 被其他 Activity 覆盖而完全变暗。
- 终止状态，这时 Activity 将会被系统清理出内存。

需要注意的是，处于暂停状态和停止状态的 Activity 仍然保存了其所有的状态和成员信息，直到被系统终止。当被系统终止的 Activity 需要重新显示的时候，必须完全重新启动并且将其关闭之前的状态全部恢复。

Activity 的生命周期状态转换如图 3-1 所示。

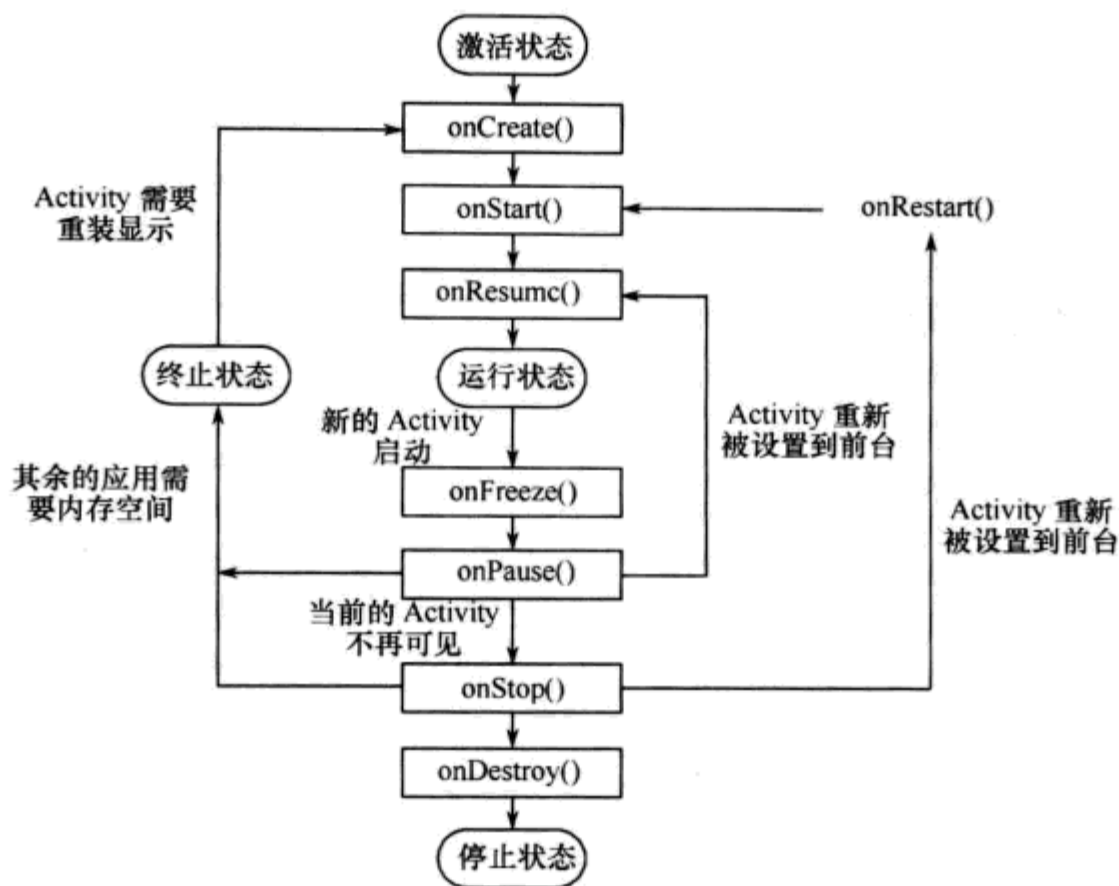


图 3-1 Activity 生命周期状态转换图

在图 3-1 所示的 Activity 生命周期状态转换图中，椭圆形表示的是 Activity 所处的状态。直角矩形代表了回调方法，开发者可以实现这些方法从而使 Activity 在改变状态的时候执行用户定义的操作。

Activity 的生命周期又可以根据不同的标准分为完整生命周期、可见生命周期和前台生命周期。

- 从 Activity 最初调用 onCreate()方法到最终调用 onDestroy()方法的这个过程称为完整生命周期。Activity 会在 onCreate()方法中进行所有全局状态的设置，在 onDestroy()方法中释放其持有的所有资源。
- 从 Activity 调用 onStart()方法开始，到调用对应的 onStop()方法为止的这个过程称为可见生命周期。在这段时间内，用户可以在屏幕上看到这个 Activity，尽管并不一定是在前台显示，也不一定可以与其交互。在这两个方法之间，用户可以维护 Activity 在显示时所需的资源。因为每当 Activity 显示或者隐藏时都会调用相应的方法，所以 onStart()方法和 onStop()方法在整个生命周期中可以多次被调用。
- 从 Activity 调用 onResume()方法开始，到调用对应的 onPause()方法为止的这个过程称为前台生命周期。这段时间当前的 Activity 处于其他所有 Activity 的前面，且可以与用户交互。

3.2.2 Activity 类的结构

应用中的每个 Activity 类在定义时都必须以 Android 中的 android.app.Activity 作为父类，该类定义了 Activity 生命周期中所包含的全部方法，具体定义代码如下：

```

public class Activity extends ApplicationContext {
    protected void onCreate(Bundle icle);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onFreeze(Bundle outIcicle);
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}

```

android.app.Activity 类中的方法定义了 Activity 完整的生命周期。这些方法在生命周期中的作用以及其相互之间的转换关系如表 3-1 所示。

表 3-1 Activity 类中的方法

方法	功能描述	跳转到的方法
onCreate()	Activity 初次创建时被调用，在该方法中一般进行一些静态设置	onStart() 或 onRestart()
onStart()	当 Activity 对用户即将可见的时候调用	onRestart() 或 onResume()
onRestart()	当 Activity 从停止状态重新启动时调用	onResume()
onResume()	当 Activity 将要与用户交互时调用	onFreeze()
onFreeze()	当 Activity 被暂停而其他的 Activity 恢复与用户交互的时候，这个方法将会被调用	onPause()
onPause()	当系统要启动一个其他的 Activity 时（其他的 Activity 显示之前），这个方法将被调用	onResume() 或 onStop()
onStop()	当另外一个 Activity 恢复并遮盖住当前的 Activity，导致其对用户不再可见时，这个方法将被调用	onStart() 或 onDestroy()
onDestroy()	在 Activity 被销毁前所调用的最后一个方法	无

开发者可以重载这些方法从而使自定义的 Activity 在状态改变时执行用户所期望的操作。当然这些方法不是必须要求被实现的，一般情况下，所有 Activity 都应该实现自己的 onCreate() 方法来进行初始化设置，大部分还应该实现 onPause()方法来准备终止与用户的交互。至于其他的方法则可以在需要时进行实现，当实现这些方法的时候需要注意的是，一定要覆盖父类中的对应方法。

下面根据 2.5 节中所述的步骤，创建一个名称为 HelloWorldText 的单 Activity 的 Android 应用，其中创建的 Activity 类名也是 HelloWorldText。创建项目的对话框如图 3-2 所示。



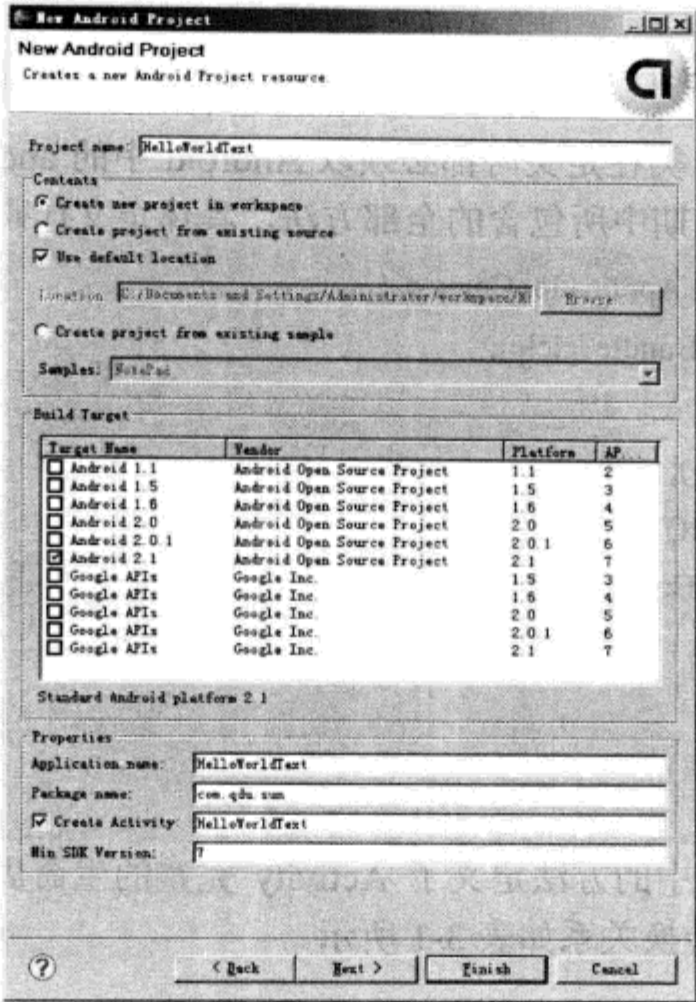


图 3-2 创建 HelloWorldText 项目的对话框

项目创建完成后，HelloWorldText 类中的缺省代码如下所示：

```

package com.qdu.sun;

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

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

```

该类开头的 import 语句表示从 Android SDK 的 android.jar 中导入特定的类，这两个导包语句是必需的。在定义用户的 Activity 类时，必须继承 Android 提供的 android.app.Activity。该类中包含的代码用来定义如何创建、显示和运行应用程序，在该类的缺省代码中只包含了一个 onCreate()方法，在该方法中首先调用父类中的 onCreate()方法，然后调用 setContentView()方法，该方法的作用是根据 main.xml 文件中的配置代码来设置 Activity 的界面内容。该方法中所需的参数是“R.layout.main”，其中 R 表示在创建项目时自动生成的 R.java 文件，该文件中的代码不需要手工修改。



创建项目后，该文件的缺省代码如下所示：

```
package com.qdu.sun;

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

通过上述代码可以看到，该类中定义了很多静态类，实际上这些静态类是指向项目中资源的指针。这时候理解 HelloWorldText 类中的代码 setContentView(R.layout.main)就很容易了，该方法的参数表示 R 类中的 layout 内部类中的 main 变量，通过该变量就可以引用 main.xml 文件了。

### 3.3 Activity 的两种界面设计方式

Activity 的界面设计可以使用两种不同的方式来实现，一种是使用基于 XML 的方式来获取界面组件从而实现界面设计，另一种是直接使用代码来创建界面组件从而实现界面设计。

#### 3.3.1 基于 XML 的界面设计

项目的 res/layout 目录中包含一个 XML 文件 main.xml，该文件中定义了 Activity 界面的布局以及界面中所需的组件的声明，因此 Activity 界面的设计实际上可以通过编写该 XML 文件来完成。

HelloWorldText 项目中 main.xml 文件的缺省代码如下所示：

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



```
android:layout_height="wrap_content"
android:text="@string/hello"
/>
```

```
</LinearLayout>
```

其中,<LinearLayout>表示界面采用线性布局方式。<TextView>标签用来声明一个 TextView 组件对象,在该标签中,android:text 属性表示 TextView 组件所显示的内容,该属性的属性值 “@string/hello” 表示引用项目 res/values 目录中的 strings.xml 文件中 name 为 “hello” 的字符串。

strings.xml 文件的缺省代码如下所示:

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

通过上述代码可以看到,strings.xml 文件中 name 为 “hello” 的字符串是 “Hello World, HelloWorldText!”,也就是说,在 Activity 界面中将通过 TextView 组件显示字符串 “Hello World, HelloWorldText!”。当然,用户可以修改要显示的字符串内容,例如,如果要在屏幕中显示 “This is my first Android Application” 字符串,则需要将 strings.xml 中对应的代码修改如下:

```
<string name="hello">This is my first Android Application</string>
```

修改完成后,运行 HelloWorldText 项目,在打开的 Android 模拟器中将显示如图 3-3 所示的运行结果。



图 3-3 显示基于 XML 设计的界面

### 3.3.2 基于代码的界面设计

除了使用 Android 建议的基于 XML 的界面设计方式之外,也可以与普通 Java 程序一样,使用代码的方式直接进行界面设计。

接下来,我们再创建一次 HelloWorldText 项目,只不过这次使用代码而不是 main.xml 文件来进行程序界面的设计。

项目创建完成后,首先打开 main.xml 文件,将其中的<TextView>标签及其内容全部删除。删除后的 main.xml 文件代码如下所示:

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



```
</LinearLayout>
```

这样，main.xml 就成为一个空白的文件了，里面没有跟界面设计相关的任何代码。然后打开 HelloWorldText.java 源文件，将下面这行代码删除：

```
setContentView(R.layout.main);
```

因为在本例中将使用代码直接创建界面，而不是通过 main.xml 文件，因此，在 HelloWorldText 类中就没有必要再引用 main.xml 文件了，所以需要将对应的代码删除。

接下来，在 HelloWorldText 类中添加如下导包的代码：

```
import android.widget.TextView;
```

因为要在 HelloWorldText 类中使用代码创建界面，所以要创建 TextView 类的实例，因此必须在类的开头先使用 import 语句导入对应的类。

然后在 onCreate() 方法中创建 TextView 类的一个实例，具体代码如下：

```
TextView HelloWorldTextView = new TextView(this);
```

上述代码创建了一个名称为 HelloWorldTextView 的 TextView 类的对象，并对其进行了实例化。

在创建完 TextView 类的实例后，可以调用该类的 setText() 方法为其设置要显示的字符串，具体代码如下：

```
HelloWorldTextView.setText("Hello World!");
```

上述代码表示为 HelloWorldTextView 对象设置要显示的字符串为 “Hello World!”。

最后，调用 setContentView() 方法将 TextView 对象显示在屏幕上，具体代码如下：

```
setContentView(HelloWorldTextView);
```

至此，就完成了使用代码进行界面设计的工作。

因此，可以总结出基于代码的界面设计的主要步骤如下：

- (1) 创建要显示在界面中的各个组件的实例。
- (2) 为各个组件设置其要显示的内容。
- (3) 使用 setContentView() 方法将这些组件显示在屏幕上。

修改后的 HelloWorldText.java 源文件的完整代码如下所示：

```
package com.qdu.sun;
```

```
import android.app.Activity;
```

```
import android.os.Bundle;
```

```
import android.widget.TextView;;
```

```
public class HelloWorldText extends Activity {
```

```
    /** Called when the activity is first created. */
```

```
    @Override
```

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

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

```
        TextView HelloWorldTextView = new TextView(this);
```



```

HelloWorldTextView.setText("Hello World!");
setContentView(HelloWorldTextView);
}
}

```

代码编写完后，运行 HelloWorldText 项目，在打开的 Android 模拟器中将显示如图 3-4 所示的运行结果。

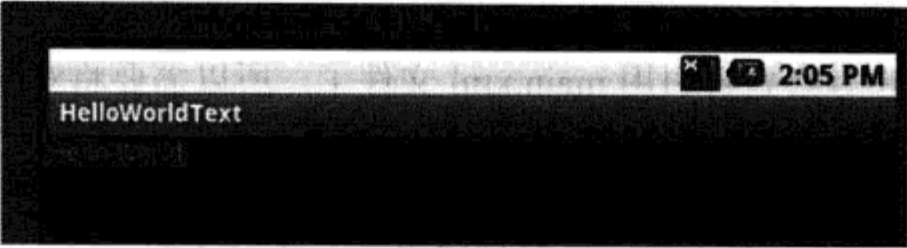


图 3-4 显示基于代码设计的界面

3.4 应用实例：在界面中显示图片

在本章之前的实例中，都是使用 TextView 组件显示静态文本信息，为了给用户带来更好的应用体验，往往需要在屏幕中显示图片。本应用实例将详细介绍如何使用 ImageView 组件在屏幕中显示图片。

首先创建名称为 HelloWorldImage 的 Android 项目，具体步骤如下：

(1) 单击 Eclipse 菜单栏中的“File”|“New”|“Project...”命令，将显示如图 3-5 所示的新建项目向导。在其中需要指定要新建的应用程序类型，故展开“Android”后，单击“Android Project”，表示要新建 Android 项目。

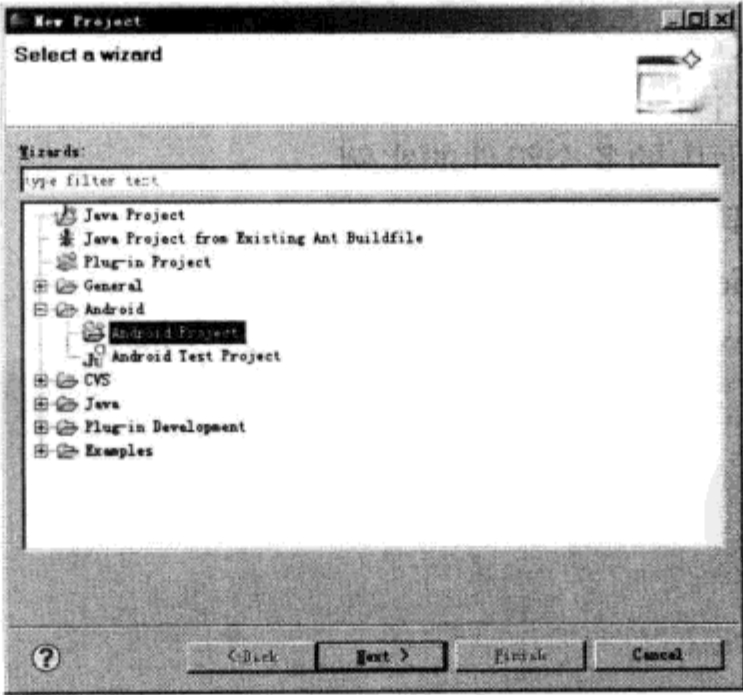


图 3-5 选择项目类型为“Android Project”

(2) 单击“Next”按钮，将显示如图 3-6 所示的新建 Android 项目窗口，在该窗口的“Project name”框中输入项目名称“HelloWorldImage”，在“Build Target”列表中选择要使用的目标设备版本为“Android 2.1”，在“Properties”区域设置应用的名称和项目中 Activity 的名称都是“HelloWorldImage”，项目中类的包名为“com.qdu.sun”。

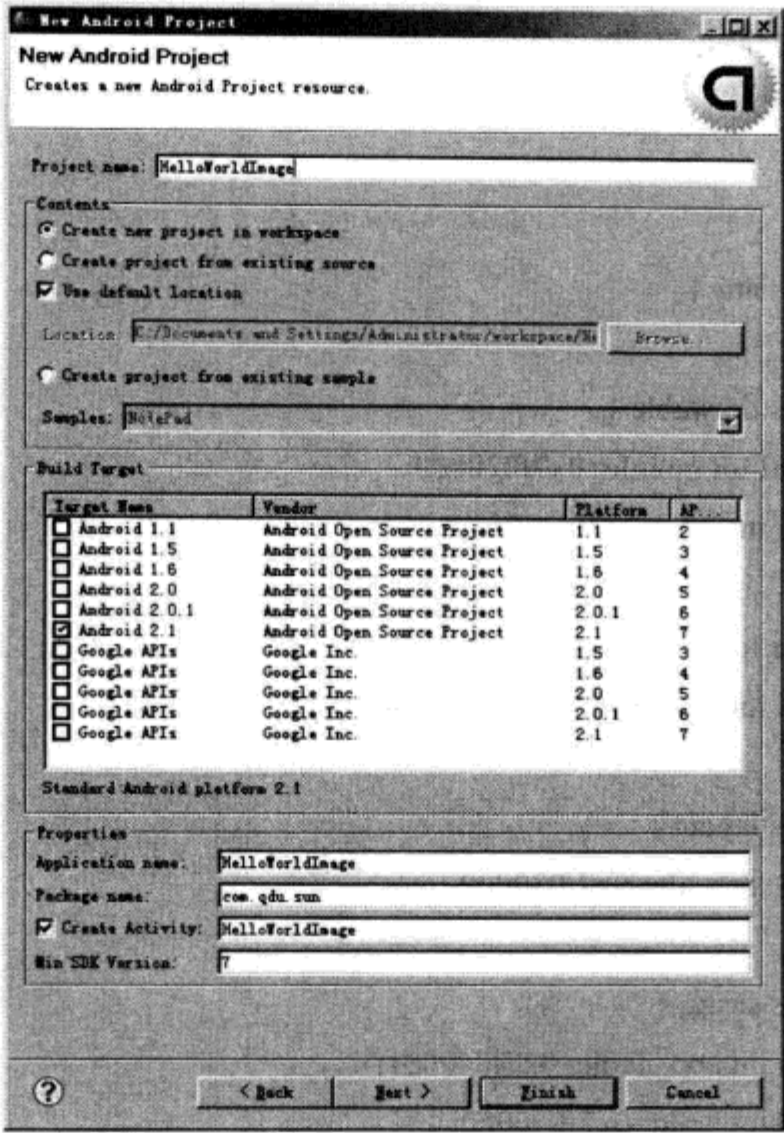


图 3-6 新建 HelloWorldImage 项目窗口

(3) 将要在屏幕中显示的.png 图片 google.png 复制到项目的 res/drawable-mdpi 目录中，如图 3-7 所示。读者可能注意到了，在项目的 res 目录中有三个名称类似的子目录，分别是 drawable-ldpi、drawable-mdpi 以及 drawable-hdpi。这三个目录都是用来存放项目中要使用的图片的，其中 drawable-hdpi 里面主要存放高分辨率的图片，drawable-mdpi 里面主要存放中等分辨率的图片，而 drawable-ldpi 里面主要存放低分辨率的图片，系统会根据模拟器的分辨率自动到这几个文件夹中去寻找对应的图片。所以在开发程序时，为了兼容不同屏幕，建议在各文件夹中根据需求均存放不同版本的图片。在将图片复制到正确的目录之后，在项目名称上单击鼠标右键，在弹出的快捷菜单中选择 “Refresh” 选项来刷新项目，google.png 图片将会出现在项目的视图中。

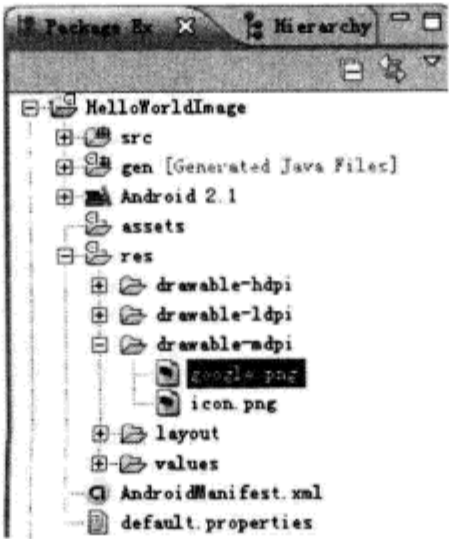


图 3-7 在项目中引入图片



(4) 打开项目中的 R.java 文件，其中的代码如下所示：

```
package com.qdu.sun;

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

    public static final class drawable {
        public static final int google=0x7f020000;
        public static final int icon=0x7f020001;
    }

    public static final class id {
        public static final int imageview=0x7f050000;
    }

    public static final class layout {
        public static final int main=0x7f030000;
    }

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

粗体显示的代码就是 Eclipse 自动为 google.png 图片增加的一个指针。

(5) 打开项目中的 main.xml 文件，将文件中的<TextView>标签内容删除，在该文件中添加用于显示图片的<ImageView>标签。



## 第 4 章 Android 人机界面和常用组件

人机界面组件是 Andriod 构建屏幕，实现人机交互的基本要素。在本章中，通过实现基本的 Andriod 界面，详细介绍 Andriod 中的基本 UI 设计方法、UI 的基本属性。并在此基础上讲述 Android 生成用户界面的两种方式：XML 文件和代码生成方式。通过本章的学习，读者将掌握 Android 人机界面的设计。

### 4.1 用户人机界面元素分类

用户人机界面由视图、视图容器、布局等组件构成。一个复杂的 Andriod 界面往往需要不同的组件组合才能实现。本节将介绍 Andriod 主要组件的特点及其功能。

#### 4.1.1 视图组件（View）

作为 Android 平台中用户界面的基础元素，View 对象存储了 Andriod 屏幕上一个特定的矩形区域的布局和内容属性的数据体。Andriod 的窗体功能是通过 Widget 类实现的，而 View 类是 Widget（窗体部件）的基类。View 类包含的子类如表 4-1 所示。

表 4-1 View 类的主要子类

类名	功能描述	事件监听器
TextView	文本框视图	OnKeyListener
EditText	编辑框	OnEditorActionListener
Button	按钮	OnClickListener
Checkbox	复选框	setOnCheckedChangeListener
RadioGroup	单选按钮	OnCheckedChangeListener
Spinner	下拉列表	OnItemSelectedListener
AutoCompleteTextView	自动完成文本框视图	OnKeyListener
DataPicker	日期选择器	OnDateChangeListener
TimePicker	时间选择器	OnTimeChangeListener
DigitalClock	数字时钟	OnKeyListener
AnalogClock	表状时钟	OnKeyListener
ProgressBar	进度条	OnProgressBarChangeListener
RatingBar	评分条	OnRatingBarChangeListener
SeekBar	拖动条	OnSeekBarChangeListener
GridView	网格视图	OnKeyDown,OnKeyUp
LsitView	列表视图	OnKeyDown,OnKeyUp
ScrollView	滚动视图	OnKeyDown,OnKeyUp



4.1.2 视图容器组件（**ViewGroup**）

ViewGroup 就是 View 的容器，一个 ViewGroup 对象是一个 Android.view.Viewgroup 的实例。通过这种容器可将 View 添加进来，一个 ViewGroup 也可以加入到另外一个 Viewgroup 里。ViewGroup 的类层次关系如下：

```

java.lang.Object
  android.view.View
    android.view.ViewGroup

```

ViewGroup 类中嵌套了两个类和一个接口：

- ViewGroup.LayoutParams
- ViewGroup.MarginLayoutParams
- ViewGroup.OnHierarchyChangeListener（这是一个接口类）

ViewGroup 类提供的主要方法如表 4-2 所示。

表 4-2 ViewGroup 类的主要方法

方法	功能描述	返回值
ViewGroup	ViewGroup 的构造方法，ViewGroup 包含三种构造函数： <ul style="list-style-type: none"> <li>● ViewGroup(Context context)</li> <li>● ViewGroup(Context context, AttributeSet attrs)</li> <li>● ViewGroup(Context context, AttributeSet attrs, int defStyle)</li> </ul>	null
addView(View child)	addView 方法用于添加子视图，除此之外 ViewGroup 还提供了其他几种添加视图的方法： <ul style="list-style-type: none"> <li>● void addView(View child, ViewGroup.LayoutParams params)</li> <li>● void addView(View child, int index)</li> <li>● void addView(View child)</li> <li>● void addView(View child, int width, int height)</li> </ul>	void
bringChildToFront(View child)	该方法将参数指定的视图移动到所有视图的前面显示	void
clearChildFocus(View child)	该方法清除参数指定的视图的焦点	boolean
dispatchKeyEvent(KeyEvent event)	该方法将参数指定的键盘事件分发给当前焦点路径的视图。分发判断事件时，按照焦点路径查找合适的视图。若本视图为焦点，则将键盘事件发送给自己；否则发送给焦点视图	boolean
dispatchPopulateAccessibilityEvent(AccessibilityEvent event)	该方法将参数指定的事件发给当前焦点路径的视图	boolean
dispatchSetSelected(boolean selected)	该方法为所有的子视图调用 SetSelected 方法	boolean

4.1.3 布局组件（**Layout**）

在 UI 设计中，除了要清楚控件的作用和接口之外，还需要熟悉控件布局方式。Android 提供了许多种布局，常用的有以下几种：

（1）LinearLayout（线性布局）

LinearLayout 是一种线性排列的布局，在该布局中子元素之间呈线性排列，即顺序排列。

由于布局显示在二维空间里,其顺序排列是在某一方向上的顺序排列,常见的有水平顺序排列、垂直顺序排列。

## (2) RelativeLayout (相对布局)

RelativeLayout 是一种根据相对位置排列元素的布局,这种方式允许子元素指定它们相对于其他元素或父元素的位置(通过 ID 指定)。这种方式相对于线性布局,没有规律性。需要注意,线性布局不需要特别指定其父元素,相对布局使用之前必须指定其参照物。只有指定参照物之后,才能定义其相对位置。

## (3) TableLayout (表格布局)

同 LinearLayout 类似,TableLayout 是一种表格布局,这种布局将子元素的位置分配到行或列中,即按照表格的顺序排列。一个表格布局有多个“表格行”,而每个表格行又包含多个表格单元。需要注意,表格布局并不是真正意义上的表格,只是按照表格的方式组织元素。在表格布局中,元素之间并没有实际表格中的分界线。

## (4) AbsoluteLayout (绝对布局)

相对布局需要指定其参照的父元素,AbsoluteLayout 与相对布局相反,绝对布局不需要指定其参照物,绝对布局使用整个手机界面作为坐标系,通过坐标系的两个偏移量(水平偏移量和垂直偏移量)来唯一指定其位置。

### 4.1.4 布局参数 (LayoutParams)

LayoutParams 是用来设置视图布局的基类,基本的 LayoutParams 类只是用来描述视图的宽度和高度。Android 提供的布局类都是 LayoutParams 的子类,LayoutParams 的子类有:

- AbsListView.LayoutParams
- AbsoluteLayout.LayoutParams
- Gallery.LayoutParams
- ViewGroup.MarginLayoutParams
- WindowManager.LayoutParams
- AbsListView.LayoutParams
- ViewGroup.MarginLayoutParams
- WindowManager.LayoutParams
- FrameLayout.LayoutParams
- LinearLayout.LayoutParams
- RadioGroup.LayoutParams
- RelativeLayout.LayoutParams
- TableLayout.LayoutParams
- TableRow.LayoutParams

其中常用的是 RelativeLayout.LayoutParams、AbsoluteLayout.LayoutParams、LinearLayout.LayoutParams。在以后的章节里将详细介绍这些子类的作用,本章不再赘述。



4.2 常用 Widget 组件

4.2.1 文本框视图（TextView）

对于用户来说 TextView 是屏幕中一块用于显示文本的区域，它属于 `andriod.Wiget` 包并且继承 `andriod.view.View` 类。从层次关系上来说，TextView 类继承了 View 类的方法和属性，同时又是 Button、CheckedTextView、Chronometer、DigitaClock 以及 EditText 的父类。TextView 类的层次关系如下：

```

java.lang.Object
  android.view.View
    android.widget.TextView
      Button, CheckedTextView, Chronometer, DigitalClock, EditText

```

TextView 提供了用于控制文本显示的方法，如表 4-3 所示。

表 4-3 TextView 类的方法

方法	功能描述	返回值
TextView	TextView 的构造方法	null
getDefaultMovementMethod	获取默认的箭头按键移动方式	MovementMethod
getText	取得 TextView 对象的文本	CharSequence
length	获取 TextView 中文本长度	int
getEditableText	取得文本的可编辑对象,通过这个对象可对 TextView 的文本进行操作,如在光标之后插入字符	android.text.Editable
getCompoundPaddingBottom	返回 TextView 的底部填充物	int
setCompoundDrawables	设置 Drawable 图像显示的位置,在设置该 Drawable 资源之前需要调用 setBounds(Rect)	void
setCompoundDrawablesWithinIntrinsicBounds	设置 Drawable 图像显示的位置,但其边界不变	void
setPadding	根据位置设置填充物	void
getAutoLinkMask	返回自动链接的掩码	int
setTextColor	设置文本显示的颜色	void
setHighlightColor	设置选中时文本显示的颜色	void
setShadowLayer	设置文本显示的阴影颜色	void
setHintTextColor	设置提示文字的颜色	void
setLinkTextColor	设置链接文本的颜色	void
setGravity	设置当 TextView 超出了文本本身时横向以及垂直对齐	void
getFreezesText	设置该视图是否包含整个文本,如果包含则返回真值,否则返回假值	boolean

使用 TextView 类时，必须导入其所在的包路径，即 `android.widget.TextView`。TextView 定义了文本框操作的基本方法，它是一个不可编辑的文本框，往往用来在屏幕中显示静态字符串，功能类似于 Java 语言中 swing 包的 JLabel 组件。下面以一个具体的例子说明 TextView 的基本用法。



### 【实例 4-1】TextView 组件的应用。

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView; //导入 TextView 类

public class HelloAndroid extends Activity {
    @Override //标识 onCreate 方法是重写父类的方法
    public void onCreate(Bundle savedInstanceState) {
        TextView myTextView; //声明一个 TextView 的对象
        String str="Welcome to Android World!"; //定义 TextView 中显示的字符串
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myTextView=(TextView) this.findViewById(R.id.myTextViewID);
        /*调用 setText 方法设置 Andriod 屏幕上显示的字符,否则 Andriod 屏幕上显示的字符是 main.xml
        中的 text 引用的字符*/
        myTextView.setText(str);
    }
}
```

【代码说明】实例 4-1 首先导入 `android.app.Activity`、`android.os.Bundle` 和 `android.widget.TextView` 包。`Activity` 是 Android 应用程序最基本的组成单位。在 Android 应用程序中, `Activity` 主要负责创建显示窗口, 一个 `Activity` 通常就代表了一个单独的屏幕。它是用户唯一可以看得到的东西, 所以几乎所有的 `Activity` 都是用来与用户进行交互的。因此, 需要导入 `Activity` 类才能建立用户的人机交互界面。一个 `Activity` 包含完整生命周期和可视生命周期。

- 完整生命周期: 从调用 `onCreate()` 开始到 `onDestroy()` 为止是一个 `Activity` 完整的生命周期。`onCreate()` 用于设置 `Activity` 中所有“全局”状态以初始化系统资源, `onDestroy()` 用于释放所有系统资源。例如, 如果 `Activity` 有一个线程在后台运行以从网络上传文件, 它会以 `onCreate()` 创建那个线程, 而以 `onDestroy()` 销毁那个线程。
- 可视生命周期: 从调用 `onStart()` 开始到 `onStop()` 为止是一个 `Activity` 的可视生命周期。用户可以在这个周期期间, 在终端屏幕上看到这个 `Activity`。`onStart()` 和 `onStop()` 方法可被多次调用, 从而实现应用程序对用户可见或者不可见。

`onCreate()` 方法使用 `Bundle` 对象作为参数, `Bundle` 类用于在不同的 `Activity` 之间传递参数, 通常需要结合 `Intent` 类来实现不同 `Activity` 之间的交互。`TextView` 类包含了用户控制和显示文本框视图的操作方法, 需要导入该类才能在 Android 中显示文本框视图。

`@Override` 标志后面定义的方法是从父类或者接口中继承过来的, 需要重写。在本例中, `HelloAndroid` 类重载了父类的 `onCreate()` 方法。Android 应用程序没有类似于 C 语言的 `main` 入口, 会以这个方法为程序执行的入口点。`onCreate()` 方法包含了一个 `Bundle` 类型的参数, 这个参数可用于不同 `Activity` 之间的消息传递。在本章中只定义了一个 `Activity`, 在以后会详细介绍 `Activity` 之间的消息传递。

然后, `HelloAndroid` 类的 `onCreate()` 方法通过 `super` 关键字调用了父类的同名方法。在这里



super 关键字不能省略，否则程序会默认调用 HelloAndroid 类的 onCreate()方法。虽然可以成功编译省略了 super 关键字的 Android 应用程序，但是在 Andriod 模拟器中执行该应用程序时会出现“应用异常终止”的错误。

HelloAndroid 类的 onCreate()方法接着调用了 setContentView()方法。这个方法指定了这个 Activity 的界面布局，这个布局是在 R.layout.main 中定义的。如果不指定布局，执行之后会生成一个空白的屏幕。随后 onCreate()方法调用了 findViewById()方法，这个方法是重载父类的方法。这个方法的作用是加载在 XML 文件中定义的 TextView，findViewById()方法的参数是一个 Widget 类的句柄，这个句柄可用来唯一标识一个 Widget 对象。可以不通过在布局中添加 Widget 组件并且调用 setContentView()加载 TextView 组件。这个方法并不是加载 TextView 组件必须调用的，只有需要修改在 XML 中定义的 TextView 组件属性时，例如本例需要重新设置文本框视图中显示的字符串，findViewById()才需要被调用。通过 findViewById()返回的句柄来修改组件的属性。最后通过调用 TextView 的 setText()方法重新设置 myTextViewID 组件显示的字符串。

R.layout.main 定义了本程序所显示的界面布局，而这个布局实际上是一个 XML 文件，这个文件是 Res/layout/main.xml 文件。main.xml 定义布局的代码如下：

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

main.xml 包含了一个 LinearLayout 标签，这个标签定义了整个程序显示的布局。在 LinearLayout 布局中，android:orientation 用于定义布局中子元素的排列方式，布局包含两种排列方式：vertical（垂直排列）和 horizontal（水平排列）。因此，在这个布局中 vertical 声明此布局中的子元素要竖直排列，这样如果在这个布局中有两个子元素的话，那么这两个子元素将各占一行，如果将这个值设置为 horizontal，就变成了水平排列，那么两个子元素将各占一列。

android:layout\_width 定义了元素布局的宽度，可以通过三种方式来指定宽度。

- fill\_parent: 宽度和父元素相同。
- wrap\_content: 宽度随组件本身的内容调整。
- 通过指定 px 值来设置宽度。

android:layout\_height 定义了元素布局的高度，可以通过三种方式来指定高度。

- fill\_parent: 宽度和父元素相同。
- wrap\_content: 宽度随组件本身的内容调整。



- 通过指定 px 值来设置高度。

LinearLayout 标签包含一个 TextView 的子标签，这个标签定义了一个 TextView 的对象。程序会根据这个标签的定义加载一个文本框。在该标签中，使用@+id 声明了一个句柄，实际上 “@+id/myTextViewID” 定义了该对象的唯一的标识 myTextViewID，这个标识的值（myTextViewID）是在 R.java 文件中定义的。如果所定义的组件（TextView）没有被 Java 代码引用，不需要在 XML 文件中定义 android:id 属性。例如，一个定义了显示方式的静态 TextView 组件，这个组件定义的属性不需要动态改变。而在 Android 应用程序中，这种类型的组件不需要定义 android:id 属性。但是任何 Java 代码引用的组件都要通过 android:id 属性来定义该组件的句柄。android:layout\_width="fill\_parent"指定了布局的宽度和父元素相同，不会随组件本身的内容而变化；android:layout\_height="wrap\_content"指定了布局的高度随组件本身的内容而变化，不一定跟父元素的宽度相同；android:text 属性表示 TextView 组件所显示的内容，该属性的属性值 “@string/hello” 表示引用 res/values 目录的 strings.xml 文件中 name 为 “hello” 的字符串。在 strings.xml 中"hello"字符串定义如下：

```

<string name="hello">Welcome you! </string>
    
```

TextView 标签提供了用于设置 TextView 的属性，如表 4-4 所示。

表 4-4 TextView 标签的属性

属性	描述
android:autoLink	设置是否当文本为 URL 链接、E-mail、电话号码等时，文本显示为可单击的链接。可选值 none/web/email/phone/map/all
android:capitalize	设置英文字母大写类型。此处无效果，需要弹出输入法才能看得到
android:cursorVisible	设定光标为显示/隐藏，默认显示
android:digits	设置允许输入哪些字符。如 “1234567890.+*/%\\n()”
android:drawableBottom	在 text 的下方输出一个 drawable 对象，如图片。如果指定一个颜色的话会把 text 的背景设为该颜色，同时和 background 使用时覆盖后者
android:drawableLeft	在 text 的左边输出一个 drawable 对象，如图片
android:drawablePadding	设置 text 与 drawable(图片)的间隔，与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用，可设置为负数，单独使用没有效果
android:drawableRight	在 text 的右边输出一个 drawable 对象，如图片
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型

在 main.xml 文件生成之后，编译器会在 R.java 文件中添加一个静态类，并为 main.xml 文件@+id 定义的 TextView 组件生成 32 位的静态常量。可以通过这个常量唯一标识 XML 中定义的组件对象。在实例 4-1 中，R.Java 中生成的静态常量如下：

```

public static final class id {
    public static final int myTextViewID=0x7f050000;
}
    
```

这个标识作为对象的句柄，可唯一标识一个对象。Android 资源管理器用这个标识来加载相应的对象。

程序设计完成后运行该 Andriod 程序，Andriod 屏幕上没有显示 TextView 标签定义的 “Welcome you!” 而显示 “Welcome to Android World!”。这是因为在实例 4-1 中，程序通过句



柄重新了设置对象的显示属性。实例 4-1 的运行结果如图 4-1 所示。

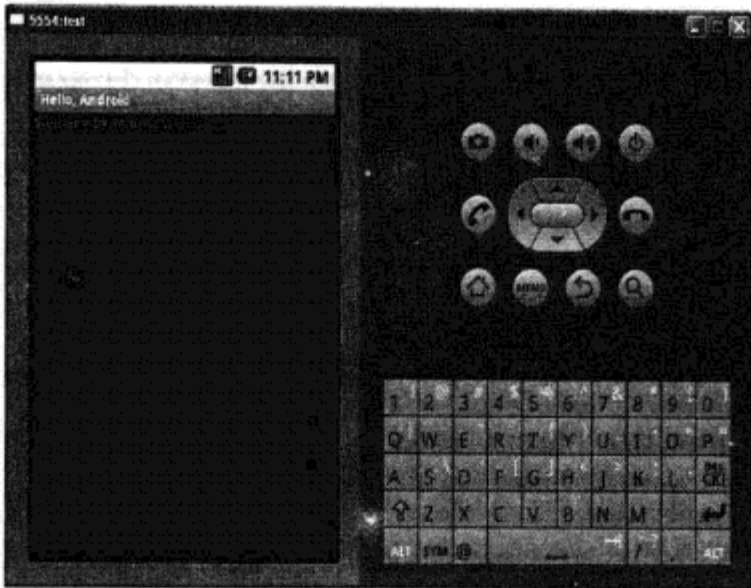


图 4-1 TextView 中显示的字符串

### 4.2.2 按钮（Button）

Button 类提供了控制按钮的功能，本节将介绍 Button 类的主要方法以及属性，并通过实例介绍 Button 类的用法。Button 类属于 andriod.Wiget 包并且继承 android.widget.TextView 类。从层次关系上来说，Button 类继承了 TextView 类的方法和属性，同时又是 CompoundButton、CheckBox、RadioButton 以及 ToggleButton 的父类。

Button 类提供了操纵控制按钮的方法和属性。事实上除了构造函数之外，Button 类没有自己定义的方法，主要通过继承父类的方法实现对按钮组件的操作。

表 4-5 列举了 Button 类的常用方法。

表 4-5 Button 类的方法

方法	功能描述	返回值
Button	Button 类的构造方法	null
onKeyDown	当用户按键时，该方法被调用。通过指定按键值以及按键对象，当相应时间发生时（按键）该方法被调用	Boolean
onKeyUp	当用户按键弹起后，该方法被调用。通过指定按键值以及按键对象，当相应时间发生时（按键弹起）该方法被调用	Boolean
onKeyLongPress	当用户保持按键时，该方法被调用。通过指定按键值、按键次数以及按键对象，当相应时间发生时（保持按键）该方法被调用	Boolean
onKeyMultiple	当用户多次按键时，该方法被调用。通过指定按键值以及按键对象，当相应时间发生时（多次按键）该方法被调用	Boolean
invalidateDrawable	刷新 Drawable 对象。执行该方法将会设置 view 为无效，最终导致 onDraw 方法被重新调用	void
scheduleDrawable	定义动画方案的下一帧	void
unscheduleDrawable	取消 scheduleDrawable 定义的动画方案	void
onPreDraw	设置视图显示，例如在视图显示之前调整滚动轴的边界	Boolean



续表

方法	功能描述	返回值
sendAccessibilityEvent	发送事件类型指定的 AccessibilityEvent。发送请求之前，该方法在需要检查 Accessibility 是否打开	void
sendAccessibilityEventUnchecked	发送事件类型指定的 AccessibilityEvent。发送请求之前，该方法在不需要检查 Accessibility 是否打开	void
setOnKeyListener	设置按键监听	void

默认情况下，Button 使用 Android 系统提供的默认背景。因此在不同平台上或者是设备上，Button 显示的风格也不相同。Android 支持修改 Button 默认的显示风格，可通过 Drawable 状态列表替换默认的背景。下面以一个具体例子说明 Button 类的基本用法。

**【实例 4-2】**Button 组件的应用。

```

public class Ex_42 extends Activity {
    TextView textview;//声明为全局变量

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final Button button1 = (Button) findViewById(R.id.myButton1);
        final Button button2 = (Button) findViewById(R.id.myButton2);
        final Button button3 = (Button) findViewById(R.id.myButton3);
        /*因为上面定义的 Button 对象被另外一个程序（setOnClickListener）引用，需将该对象声明为
常量，否则编译失败。*/
        textview = (TextView) findViewById(R.id.myTextView);
        Resources resource = this.getContext().getResources();
        final Drawable red_Drawable = resource.getDrawable(R.drawable.RED);
        final Drawable blue_Drawable = resource.getDrawable(R.drawable.BLUE);
        final Drawable yellow_Drawable = resource.getDrawable(R.drawable.YELLOW);
        /*因为 Drawable 对象被另外一个方法（setOnClickListener）引用，需将该对象声明为常量。
否则编译失败。*/
        button1.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                String str = "You have clicked " + button1.getText().toString();
                textview.setText(str);
                //当单击 button1 时，修改 textview 中的文本
                if (textview.getBackground() != red_Drawable)
                    textview.setBackgroundDrawable(red_Drawable);
                //若当前 textview 背景颜色不是红色，则修改背景颜色为红色
            }
        });
        //实现 button1 的单击监听方法
        button2.setOnClickListener(new View.OnClickListener() {

```



```

        public void onClick(View v) {
            String str = "You have clicked " + button2.getText().toString();
            textview.setText(str);
            //当单击 button2 时, 修改 textview 为 str 包含的字符串
            if (textview.getBackground() != blue_Drawable)
                textview.setBackgroundDrawable(blue_Drawable);
            //若当前 textview 背景颜色不是蓝色, 则修改背景颜色为蓝色
        }
    });
    //实现 button2 的单击监听方法
    button3.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            String str = "You have clicked " + button3.getText().toString();
            textview.setText(str);
            //当单击 button3 时, 修改 textview 为 str 包含的字符串
            if (textview.getBackground() != yellow_Drawable)
                textview.setBackgroundColor(Color.YELLOW);
            //若当前 textview 背景颜色不是黄色, 则修改背景颜色为黄色
        }
    });
    //实现 button3 的单击监听方法
}
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    CharSequence charseq="You have pressed ";
    charseq = charseq + "a key!";
    textview.setText(charseq);
    return super.onKeyDown(keyCode, event);
}
//实现 onKeyDown 接口方法。当用户按键时, 该方法被调用
public boolean onKeyUp(int keyCode, KeyEvent event)
{
    CharSequence charseq="Change your color here!";
    textview.setText(charseq);
    textview.setBackgroundColor(Color.WHITE);
    return super.onKeyUp(keyCode, event);
}
//实现 onKeyUp 接口方法。当释放之前的按键时, 该方法被调用
}

```

【代码说明】实例 4-2 首先通过 XML 文件创建 Button 控件和 Drawable 颜色对象, 然后注



册 Button 控件单击事件监听以及 TextView 的按键与释放按键监听的方法。单击 Button 控件时，OnClickListener 类触发 onClick()方法，该方法通过 Drawable 对象改变 TextView 的背景颜色。使用 OnClickListener 前，需在实例中通过“import android.view.KeyEvent;”、“import android.graphics.drawable.Drawable;import android.content.res.Resources;”、“import android.graphics.Color”等语句导入所必需的包。KeyEvent 提供了获取键盘事件的方法，如按键的时间、按键值等，Drawable 类提供了引用 XML 文件中定义的 Drawable 资源的方法，Resources 包提供了处理 Drawable 对象的方法，如本实例中的 final Drawable red\_Drawable = resource.getDrawable(R.drawable.RED)。

本实例使用 Android 自定义的颜色设置 TextView 的背景。颜色是在 UI 设计中重要的属性，Android 采用 alpha+RGB 提供丰富的颜色方案。需要使用 4 个字节来表示颜色，前一个字节为 alpha 值，该值代表透明度；后面 3 个字节是 RGB 值。Android 有两种使用颜色的方法：

- Color 类：该类定义了常见的 12 种颜色常量（如表 4-6 所示）。
  - Drawable 标签：通过指定颜色的值生成颜色常量，从而可以获得任意颜色。
- 为了比较上述两种用法，实例 4-2 使用了这两种方法设置背景颜色。

表 4-6 Color 类定义的颜色

名称	整型值	十六进制	颜色
BLACK	-16777216	0xFF000000	黑
DKGRAY	-12303292	0xFF444444	暗灰
GRAY	-7829368	0xFF888888	灰
LTGRAY	-3355444	0xFFCCCCCC	亮灰
WHITE	-1	0xFFFFFFFF	白
RED	-65536	0xFFFF0000	红
GREEN	-16711936	0xFF00FF00	绿
BLUE	-16776961	0xFF0000FF	蓝
YELLOW	-256	0xFFFF0000	绿
CYAN	-16711681	0xFF00FFFF	蓝绿
MAGENTA	-65281	0xFFFF00FF	品红
TRANSPARENT	0	0x00000000	透明

实例 4-2 通过 XML 生成包含 3 个 Button 的 UI，其 UI 的 XML 定义如下：

```

<TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:layout_gravity="top"
    android:background="@drawable/WHITE"
/>
<!-- 定义 myTextView 控件，设置背景颜色为白色 -->
    
```



```
<Button
    android:id="@+id/myButton1"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/myButtonText1"
    android:layout_gravity="center"
    android:layout_weight="1"
/>
<!-- 定义 myButton1 控件, 设置布局位置为居中 -->
<Button
    android:id="@+id/myButton2"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/myButtonText2"
    android:layout_gravity="center"
    android:layout_weight="2"
/>
<!-- 定义 myButton2 控件, 设置布局位置为居中 -->
<Button
    android:id="@+id/myButton3"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:text="@string/myButtonText3"
    android:layout_gravity="bottom"
    android:layout_weight="3"
/>
<!-- 定义 myButton3 控件, 设置布局位置为底部 -->
```

该界面包含 3 个 Button 控件以及一个 TextView 控件。在布局定义中, 使用了 `layout_gravity` 和 `layout_weight` 两个属性。`layout_gravity` 用于指定控件的位置, `layout_gravity` 提供了 `top`、`bottom`、`left`、`right`、`center_vertical`、`fill_vertical`、`fill_horizontal`、`center`、`fill` 和 `clip_vertical` 共 10 种位置。

- `top`: 控件位于布局的顶端。
- `center`: 控件位于布局的中间。
- `bottom`: 控件位于布局的底端。
- `left`: 控件位于布局的左端。
- `right`: 控件位于布局的右端。

`layout_weight` 用于设置控件在整个布局的比重, 所以 `myButton1`、`myButton2` 和 `myButton3` 在整个布局中比例为 1 : 2 : 3。需要注意的是, `layout_weight` 值需要加引号。如果不加引号, 如 `android:layout_weight=3`, 则编译出错。

Button 标签提供了许多用于设置控制按钮的属性, 表 4-7 列举了常用的 Button 标签属性。



表 4-7 Button 标签的属性

属性	描述
android:layout_height	设置控件高度。可选值 fill_parent、wrap_contect、px
android:layout_weight	设置控件宽度。可选值 fill_parent、wrap_contect、px
android:text	设置控件名称。可选值：任意字符串
android:layout_gravity	设置控件在布局中的位置。可选值 top、bottom、left、right、center_vertical、fill_vertical、fill_horizontal、center、fill、clip_vertical
android:layout_weight	设置控件在布局中的比重。可选值：任意数字，如“3”
android:textColor	设置文字颜色。可选值：任意颜色值，如 0xFFFFFFFF
android:bufferType	设置取得的文本类别。可选值 normal、spannable、editable
android:hint	设置文本为空时显示的字符。可选值：任意字符串，如“请单击按钮”
Android:textColorHighlight	设置文本被选中时，高亮显示的颜色。可选值：任意颜色值，如 0xFFFFFFFF
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型。可选值 none、text、textCapCharacters、textWords、textCapSentences 等

OnClickListener()方法引用了外部定义的 Button 对象,这些对象被 final 关键字声明成常量。如果去掉 final 关键字,不能引用外部的非常量而使编译失败。需通过 final 关键字将被引用对象声明成常量对象（只能引用不能改变其值）,才能使得 setOnClickListener()方法引用该对象。

接着通过 getDrawable()方法,Resource 对象获取了 XML 中定义的 Drawable 资源。Drawable 资源可被看成图像的抽象对象,Drawable 资源可能是一张位图 (BitmapDrawable),也可能是一个图形 (ShapeDrawable),还有可能是一个图层 (LayerDrawable)。这些 Drawable 资源可通过 getDrawable()方法获取并绘制到屏幕上。

然后 Button 控件通过实现 setOnClickListener()方法设置单击监听方法,该方法在用户单击按钮时被调用。本实例实现了两种按键监听方法 onKeyDown 和 onKeyUp,这两个方法实现了按键重置 TextView 背景颜色的功能。读者注意到,本实例采用的键盘监听处理方法中使用 String 和 Charsequence 两种对象设置文本。CharSequence 与 String 都能用于定义字符串,但 CharSequence 的值是可读可写序列,而 String 的值是只读序列。

实例实现了通过 3 个 Button 控件控制 TexView 显示不同背景颜色的功能:

- 单击 button1 时,设置 TexView 的背景颜色为红色。
- 单击 button2 时,设置 TexView 的背景颜色被蓝色。
- 单击 button2 时,设置 TexView 的背景颜色被黄色。
- 当用户按下任意键并释放时,TextView 的颜色被置为白色。

Button 控件和 TextView 的事件监听方法分别采用了不同的颜色设置方式。前者通过 Drawable 对象设置背景颜色,后者通过 Color 类提供的颜色常量设置背景颜色。Drawable 颜色的 XML 定义如下:

```

<resources>
    <drawable name="BLUE">#FF0000FF</drawable>
    <drawable name="BLACK">#FF000000</drawable>
    <drawable name="RED">#FFFF0000</drawable>
    <drawable name="YELLOW">#FFFFFF00</drawable>

```



```

<drawable name="WHITE">#FFFFFF</drawable>
</resources>

```

这些 Drawable 对象可以通过 “R.drawable.<名字>” 引用，例如 R.drawable.BLUE 代表 drawable 标签中定义的 BLUE 所指定的颜色，其值为 #FF0000FF。

至此，控制颜色显示的程序就完成了。运行该 Andriod 程序后，TextView 的背景颜色为白色，3 个按钮按照定义的位置（layout\_gravity）和比例（layout\_weight）在布局中显示。程序启动后，界面如图 4-2 所示。

在单击 Blue button 时，Blue button 的单击事件监听方法被调用。该方法设置 TextView 的背景颜色为蓝色。单击 Blue button 后，程序运行结果如图 4-3 所示。

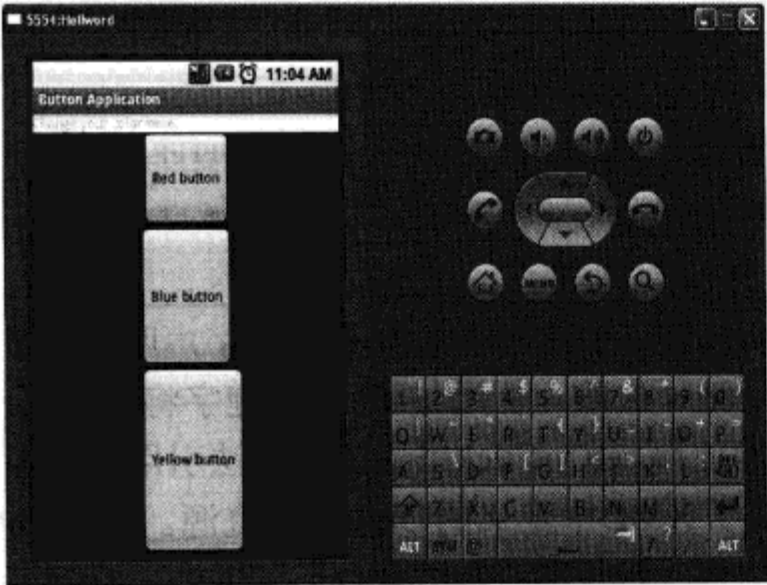


图 4-2 实例 4-2 启动时的界面

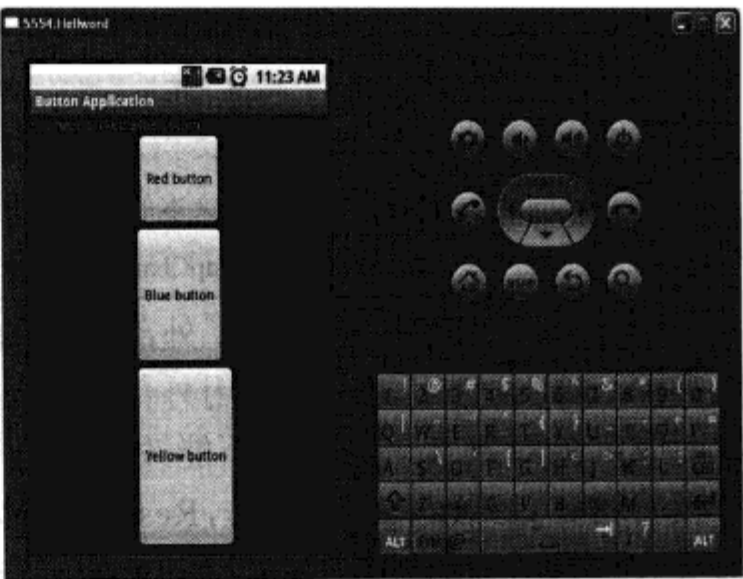


图 4-3 单击 Blue button 层的界面

当按下任意键时，TextView 的 onKeyUp()方法被调用。释放按键后，TextView 的 onKeyDown()方法被调用。onKeyDown()方法将 TextView 的背景颜色恢复成白色。按下任意键后，程序运行结果如图 4-4 所示。

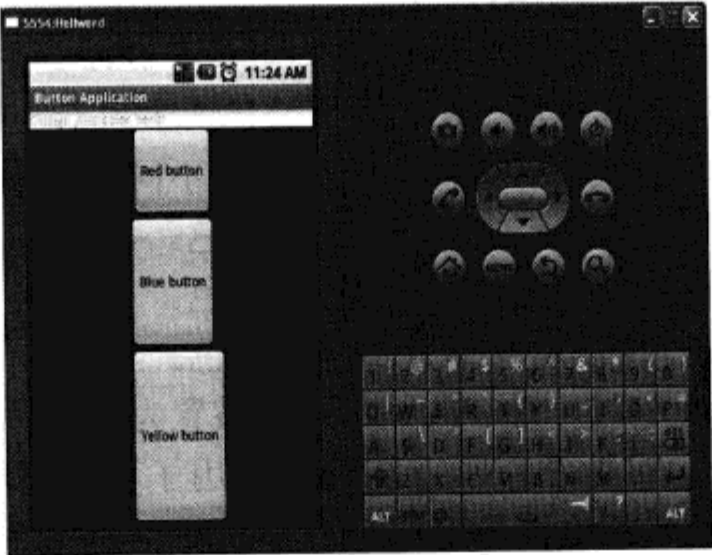


图 4-4 按键后的界面

4.2.3 图片按钮（ImageButton）

ImageButton 与 Button 功能基本类似，主要区别是 ImageButton 可通过图像表示按钮的外观。ImageButton 显示一个可以被用户单击的图片按钮，默认情况下，ImageButton 看起来像一个普通的按钮。可通过 android:src 属性或 setImageResource()方法指定 ImageButton 显示的图片。



本节将介绍 `ImageButton` 类的主要方法以及属性，并通过实例介绍 `ImageButton` 类的用法。`ImageButton` 类属于 `andriod.Wiget` 包并且继承 `android.widget.ImageView` 类。

除了构造函数之外，`ImageButton` 类只定义了一个方法：`onSetAlpha(int alpha)`。`ImageButton` 主要通过继承父类的方法提供对图片按钮控件的操作。

表 4-8 描述了 `ImageButton` 常用的方法。

表 4-8 `ImageButton` 常用的方法

方法	功能描述	返回值
<code>ImageButton</code>	<code>ImageButton</code> 类的构造方法	<code>null</code>
<code>setAdjustViewBounds</code>	设置是否保持高宽比。需要结合 <code>maxWidth</code> 和 <code>maxHeight</code> 一起使用	<code>Boolean</code>
<code>getDrawable</code>	获取 <code>Drawable</code> 对象。若获取成功，则返回 <code>Drawable</code> 对象，否则返回 <code>null</code>	<code>Drawable</code>
<code>getScaleType</code>	获取视图的填充方式	<code>ScaleType</code>
<code>setScaleType</code>	设置视图的填充方式。Android 提供了包括矩阵、拉伸等七种填充方式	<code>void</code>
<code>SetAlpha</code>	设置图片透明度。透明值范围为 0~255，其中 0 为完全透明，255 为完全不透明	<code>void</code>
<code>setMaxHeight</code>	设置按钮控件的最大高度	<code>void</code>
<code>setMaxWidth</code>	设置按钮控件的最大宽度	<code>void</code>
<code>setImageURI</code>	设置图片地址。图片地址使用 <code>URI</code> 指定	<code>void</code>
<code>setImageResource</code>	设置图片资源库	<code>void</code>
<code>setOnTouchListener</code>	设置 <code>ImageButton</code> 单击事件监听	<code>Boolean</code>
<code>setColorFilter</code>	设置颜色过滤。需要制定颜色过滤矩阵	<code>void</code>

`ImageButton` 类提供了图片按钮的功能，`ImageButton` 的单击事件监听方法不同于 `Button` 的单击事件监听方法。前者使用 `setOnTouchListener()` 方法设置事件监听方法，而后者使用 `setOnClickListener()` 方法。下面以一个具体的例子说明 `ImageButton` 类的基本用法。

【实例 4-3】`ImageButton` 组件的应用。

```

public class Ex_43 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ImageButton btn = (ImageButton) findViewById(R.id.imagebutton);
        /*单击时的颜色过滤 */
        final float[] CLICKED=new float[] {
            2, 0, 0, 0, 2,
            0, 2, 0, 0, 2,
            0, 0, 2, 0, 2,
            0, 0, 0, 1, 0 };
        /*单击结束时的颜色过滤 */
    }
}
    
```



```

final float[] CLICKED_OVER=new float[] {
    1, 0, 0, 0, 0,
    0, 1, 0, 0, 0,
    0, 0, 1, 0, 0,
    0, 0, 0, 1, 0 };

btn.setBackgroundResource(R.drawable.touch_up); //设置 touch_up.png 为背景图像
btn.setOnTouchListener(new ImageButton.OnTouchListener(){
    @Override
    public boolean onTouch(View view, MotionEvent event) {
        if(event.getAction() == MotionEvent.ACTION_DOWN){
            view.setBackgroundResource(R.drawable.touch_down);
            view.getBackground().setColorFilter(new ColorMatrixColorFilter(CLICKED));
            //取得背景颜色过滤矩阵
            view.setBackgroundDrawable(view.getBackground());
            //设置背景为指定的过滤颜色
        }
        /*当单击时，设置背景颜色为 CLICKED 的过滤颜色*/
        else if(event.getAction() == MotionEvent.ACTION_UP){
            view.setBackgroundResource(R.drawable.touch_up);
            view.getBackground().setColorFilter(new ColorMatrixColorFilter(CLICKED_OVER));
            //取得背景颜色过滤矩阵
            view.setBackgroundDrawable(view.getBackground());
            //设置背景为指定的过滤颜色
        }
        /*当单击结束时，设置背景颜色为 CLICKED_OVER 的过滤颜色*/
        return false;
    }
});
//实现 ImageButton 的鼠标单击事件监听
}
}

```

【代码说明】实例 4-3 通过 XML 创建了图片按钮，可在不同单击事件下显示不同图片。可以通过两种方式实现 ImageButton 的单击事件处理：一种是覆盖 setOnTouchListener()方法；另外一种是通过 XML 的 selector 标签实现，例如：

```

<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:drawable="@drawable/img_pressed" /> <!-- 单击时，显示 img_pressed.png-->

```



```

<item android:state_focused="true"
      android:drawable="@drawable/img_focused" /> <!-- 聚焦时，显示 img_focused.png-->
<item android:drawable="@drawable/img_normal" /> <!-- 默认显示 img_normal.png-->
</selector>

```

保存上面的 XML 文件到 res/drawable/文件夹下（注意文件名大小写!），将文件名作为一个参数设置到 ImageButton 的 android:src 属性（如 XML 文件名为 myselector.xml，那么设置为 “@drawable/myselector”，设置 android:background 也是可以的，但效果不太一样）。<item>元素的顺序很重要，因为是根据这个顺序判断是否适用于当前按钮状态的，这也是为什么正常（默认）状态指定的图片放在最后，因为它只会在 pressed 和 focused 都判断失败之后才会被采用。例如，按钮被按下时是同时获得焦点的，但是获得焦点并不一定按了按钮，所以这里会按顺序查找，找到合适的就不继续找了。

实例 4-3 通过 XML 定义 ImageButton，其代码如下：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <ImageButton
        android:id="@+id/imagebutton"
        android:layout_gravity="center"
        android:layout_width="150px"
        android:layout_height="150px" />
</LinearLayout>

```

上述布局定义了一个图片按钮，该图片按钮以居中方式显示，高度和宽度都为 150 像素。除了这些之外，XML 还提供了其他 ImageButton 属性。表 4-9 列举了 ImageButton 标签的常用属性。

表 4-9 ImageButton 标签的常用属性

属性	描述
android:adjustViewBounds	设置是否保持宽高比，可选值 true、false
android:cropToPadding	是否截取指定区域用空白代替。单独设置无效果，需要与 scrolly 一起使用。可选值 true、false
android:maxHeight	设置图片按钮的最大高度
android:maxLength	设置图片按钮的最大宽度
android:scaleType	设置图片的填充方式，可选值 0~7
android:src	设置图片按钮的 drawable
android:tint	设置图片为渲染颜色

至此启动该 Android 程序，ImageButton 在不同单击状态时使用不同的图片作为背景（左图为单击时界面，右图为单击释放时界面）。程序运行结果如图 4-5 所示。



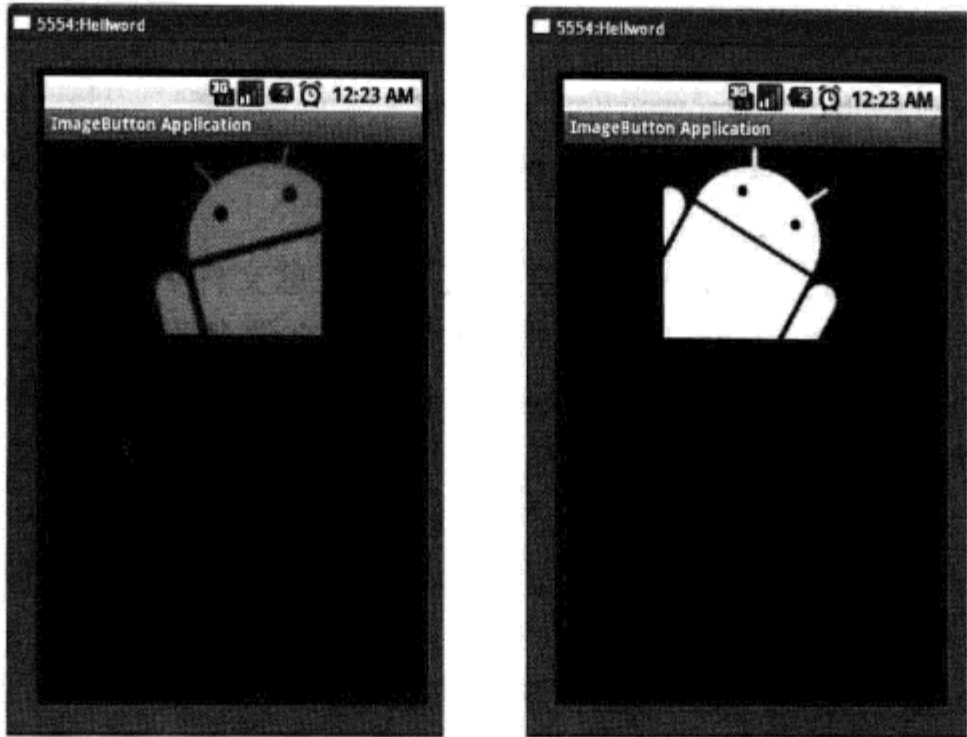


图 4-5 ImageButton 应用实例

4.2.4 编辑框（EditText）

EditText 同 TextView 功能基本类似，它们之间的主要区别是 EditText 提供了可编辑的文本框。本节将介绍 EditText 类的主要方法以及属性，并通过实例让读者了解使用 EditText 创建 UI 界面的方法。EditText 类是 TextView 的子类，同时 EditText 类又派生出两个子类（Atuo CompleteText View 和 ExtractEditText）。EditText 类的层次关系如下：

```

java.lang.Object
android.view.View
android.widget.TextView
android.widget.EditText
AtuoCompleteTextView ExtractEditText

```

EditText 是用户与系统之间的文本输入接口，用户通过这个组件可以把数据传给 Android 系统，然后得到想要的数
 据。EditText 提供了许多用于设置和控制文本框功能的方法，下面列举了 EditText 定义的方法：

```

EditText(Context context)
EditText(Context context, AttributeSet attrs)
EditText(Context context, AttributeSet attrs, int defStyle) //构造函数
void extendSelection(int index) //public 类型方法
void selectAll() //public 类型方法
void setEllipsize(TextUtils.TruncateAt ellipsis) //public类型方法
void setSelection(int index) //public 类型方法
void setSelection(int start, int stop) //public 类型方法
boolean getDefaultEditable() //Protected 类型方法
MovementMethod getDefaultMovementMethod() //Protected 方法
/*以上是 EditText 自身定义的方法*/

```

除了自身定义的方法之外，表 4-10 列举了 EditText 类其他常用的方法。



表 4-10 EditText 常用的方法

方法	功能描述	返回值
setImeOptions	设置软键盘的 Enter 键	void
getImeActionLabel	设置 IME 动作标签	Charsequence
getDefaultEditable	获取是否默认可编辑	boolean
setEllipsize	设置当文字过长时控件显示的方式	void
setFreezesText	设置保存文本内容以及光标位置	void
getFreezesText	获取保存文本内容以及光标位置	boolean
setGravity	设置文本框在布局中的位置	void
getGravity	获取文本框在布局中的位置	int
setHint	设置文本框为空时，文本框默认显示的字符串	void
getHint	获取文本框为空时，文本框默认显示的字符串	CharSequence
setIncludeFontPadding	设置文本框是否包含底部和顶端的额外空白	void
setMarqueeRepeatLimit	在 ellipsize 指定 marquee 的情况下，设置重复滚动的次数， 当设置为 marquee_forever 时表示无限次	void

EditText 提供了可编辑的文本框功能，下面以一个具体例子说明 EditText 的基本用法。

【实例 4-4】EditText 组件的应用。

```

package Text.Ex_44;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView.OnEditorActionListener;//提供编辑事件监听接口
import android.view.KeyEvent;//键盘事件包
import android.widget.EditText; //导入可编辑文本框类
import android.widget.TextView; //导入不可编辑文本框类

public class Ex_44 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);//设置界面布局
        EditText ET_phone=(EditText)findViewById(R.id.ET_phonenumber);
        EditText ET_password=(EditText)findViewById(R.id.ET_password);
        final TextView text=(TextView)findViewById(R.id.myTextView); //获取 XML 定义的资源
        ET_phone.setOnEditorActionListener(new OnEditorActionListener() {
            public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
                text.setText("Editing ET_phonenumber");//当编辑 ET_phone 时，修改文本框内容
                return false;
            }
        })
    }
}
    
```



```

    });

    /*ET_password.setOnEditorActionListener 实现 ET_phonenumber 文本框编辑事件监听。当编辑 ET_phonenumber 时，ET_phonenumber 的 onEditorAction 被调用。*/
    ET_password.setOnEditorActionListener(new OnEditorActionListener() {
        public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
            text.setText("Editing ET_password");//当编辑 ET_password 时，修改文本框内容
            return false;
        }
    });

    /*ET_password.setOnEditorActionListener 实现 ET_password 文本框编辑事件监听。当编辑 ET_password 时，ET_password 的 onEditorAction 被调用。*/
    }
}

```

【代码说明】实例 4-4 实现了两个 EditText (ET\_password 和 ET\_phone) 的编辑事件监听。当编辑相应的文本框时，编辑事件处理方法 onEditorAction 就会被调用。当编辑 ET\_phonenumber 时，ET\_phonenumber 的 onEditorAction 被调用。当编辑 ET\_password 时，ET\_password 的 onEditorAction 被调用。这两个文本框在 XML 中的定义如下：

```

<TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
<!-- 不可编辑文本框，适用于没有文本交互的应用-->
<EditText
    android:id="@+id/ET_phonenumber"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:maxLength="40"
    android:hint="Please enter your phone"
    android:textColorHint="#FF000000"
    android:phoneNumber="true"
    android:imeOptions="actionGo"
    android:inputType="text"/>
<!-- 用于输入数字的文本框，可作为只接受电话号码输入的文本框-->
<EditText
    android:id="@+id/ET_password"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:maxLength="40"
    android:hint="Please enter your password"
    android:password="true"

```



```

        android:textColorHint="#FF000000"
        android:imeOptions="actionSearch"/>
        <!-- 用于输入密码的文本框 -->
    </LinearLayout>

```

上述 XML 代码定义了两个 EditText：ET\_password 和 ET\_phonenumber，通过属性设置了这两个文本框的功能。这两个编辑框实现了一个简单的手机用户登录界面。其中，ET\_password 编辑框可作为密码输入框，而 ET\_phonenumber 可作为电话号码输入框。下面重点说明本实例中使用的 EditText 属性：

属性 phoneNumber 用于设置编辑框是否只接受数字，包含两个取值：

- phoneNumber="true"：编辑框只接受数字，不显示非数字字符。
- phoneNumber="false"：编辑框不只接受数字，可显示任意的字符。默认情况下，phoneNumber="false"。

属性 imeOptions 用于设置键盘的 Enter 键图标，取值包括：

- actionSearch：放大镜图标。
- actionNone：回车键，按下后光标移到下一行。
- actionGo：Go。
- actionSend：Send。
- actionNext：Done。

属性 inputType 用于设置编辑文本框对应的虚拟键盘，取值包括：

- date：时间键盘（如图 4-6 左）。
- text：文本键盘（如图 4-6 中）。
- phone：拨号键盘（如图 4-6 右）。



图 4-6 inputType 对应的键盘

除了 phoneNumber、imeOptions 和 inputType 之外，EditText 标签还包含其他属性，如表 4-11 所示。

表 4-11 EditText 标签的主要属性

属性	描述
android:editable	设置文本框是否可编辑，可选值 true、false
android:ellipsize	设置文本框文字过长时的显示方式，可选值：start-开头显示省略号、end-结尾显示省略号、middle-中间显示省略号、marquee-动态横向显示
android:freezesText	设置保存文本的内容以及光标的位置
android:imeOptions	设置 Enter 键的图标，可选值 normal、actionUnspecified、actionNone、actionGo、actionSearch、actionSend、actionNext、actionDone、flagNoExtractUi、flagNoAccessoryAction、flagNoEnterAction



续表

属性	描述
android:imeActionId	设置 IME 动作标识, 该标识在 onEditorAction 中捕获
android:imeActionLabel	设置 IME 动作标签
android:includeFontPadding	设置顶部和底部额外空白是否包含在文本, 可选值 true、false
android:marqueeRepeatLimit	设置动态横向重复移动的次数
android:lineSpacingExtra	行间距设置, 可选值数字
android:lineSpacingMultiplier	行间距的倍数, 可选值数字

两个文本框构成了一个简单的用户登录界面。其中一个文本框接收数字输入, 不显示非数字字符; 另外一个文本框接收密码输入, 输入的字符被显示成一个点。启动该 Android 程序, 程序运行结果如图 4-7 所示。

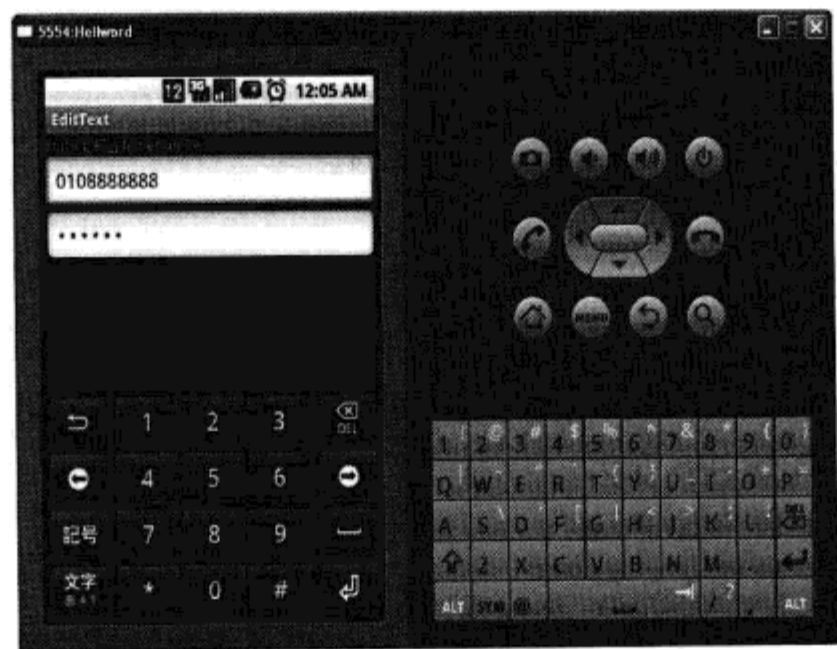


图 4-7 EditText 实例运行结果

4.2.5 多项选择（CheckBox）

多项选择（CheckBox）组件也被称为复选框, 该组件通常用于某选项的打开或关闭。CheckBox 表明一个特定的状态是勾选（on, 值为 1）还是不勾选（off, 值为 0), 在应用程序中为用户提供“真”或“假”选择。复选框状态彼此独立, 因此可同时选择任意多个。本节将介绍 CheckBox 类的主要方法以及属性, 并通过实例让读者掌握该组件的用法。CheckBox 类是 TextView 的子类, 同时 EditText 类又派生出两个子类（AtuoCompleteTextView 和 ExtractEditText)。CheckBox 类的层次关系如下:

```

android.widget.Button
android.widget.CompoundButton
android.widget.CheckBox

```

CheckBox 类提供了用于设置和控制复选框的方法, 表 4-12 列举了 CheckBox 类主要的方法。

复选框的状态只有两种: 选中或者未选中, 因此复选框状态变化包含两种情况:

- 复选框由选中状态变成未选中状态。
- 复选框由未选中状态变成选中状态。



表 4-12 CheckBox 常用的方法

方法	功能描述	返回值
dispatchPopulateAccessibilityEvent	在子视图创建时，分派一个辅助事件	boolean（True-完成辅助事件分发 /False-没有完成辅助事件分发）
isChecked	判断组件状态是否勾选	boolean（True-该组件被勾选/False- 该组件未被勾选）
onRestoreInstanceState	设置视图恢复以前的状态，这个状态是 onSaveInstanceState() 方法生成的	void
performClick	执行 Click 动作，该动作会触发事件监听器	boolean（True-调用事件监听器 /False-没有调用事件监听器）
setButtonDrawable	根据 Drawable 对象设置组件的背景	void
setChecked	设置组件的状态。若参数为真，则置组件为选中状态；否则 置组件为未选中状态	void
setOnCheckedChangeListener	CheckBox 常用的设置事件监听器的方法。组件状态改变时， 该监听器被调用	void
toggle	改变按钮的当前状态	void
onCreateDrawableState	获取文本框为空时，文本框默认显示的字符串	CharSequence
onCreateDrawableState	为当前视图生成新的 Drawable 状态	int[]

通过鼠标单击复选框，可触发复选框状态的改变。复选框会从当前状态变到另一种状态。

下面以一个实例讲述 CheckBox 的基本用法。本实例创捷了多个复选框组件，当选择相应的复选框时，程序会弹出一个 Toast 对话框提示复选框状态。

【实例 4-5】CheckBox 组件的应用。

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);    //设置界面布局
    /*根据 XML 中控件标签定义的属性生成控件*/
    checkbox1 = (CheckBox) findViewById(R.id.CheckBox01);
    checkbox2 = (CheckBox) findViewById(R.id.CheckBox02);
    checkbox3 = (CheckBox) findViewById(R.id.CheckBox03);
    checkbox4 = (CheckBox) findViewById(R.id.CheckBox04);
    button = (Button) findViewById(R.id.Submit);

    /*注册 checkbox1、checkbox2、checkbox3、checkbox4 以及 button 监听事件*/
    checkbox1.setOnCheckedChangeListener(new CheckBoxListener());
    /*checkbox1 状态改变（选中或取消选中）监听器*/
    checkbox2.setOnCheckedChangeListener(new CheckBoxListener());
    /*checkbox2 状态改变（选中或取消选中）监听器*/
    checkbox3.setOnCheckedChangeListener(new CheckBoxListener());
    /*checkbox3 状态改变（选中或取消选中）监听器*/

```



```
checkbox4.setOnCheckedChangeListener(new CheckBoxListener());
/*checkbox2 状态改变（选中或取消选中）监听器*/
button.setOnClickListener((OnClickListener) new ButtonClickListener());//button 单击监听器

}

/*定义复选框勾选状态监听器。当复选框状态发生改变时（选中变成未选中或未选中变成选中），
onCheckedChangeListener 方法被调用*/
class CheckBoxListener implements OnCheckedChangeListener{
    public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked) {
        if(isChecked){
            toast=Toast.makeText(Ex_45.this, buttonView.getText()+"被选择",Toast.LENGTH_SHORT );
            toast.setGravity(Gravity.CENTER, 5, 5);
            toast.show();
        }else{
            toast=Toast.makeText(Ex_45.this, buttonView.getText()+"取消选择",Toast.LENGTH_SHORT );
            toast.setGravity(Gravity.CENTER, 5, 5);
            toast.show();
        }
    }
}

/*定义按钮单击监听器。当单击按钮时，onClick 方法被调用*/
class ButtonClickListener implements OnClickListener{
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        String str = "";
        if(checkbox1.isChecked())
            str = str+checkbox1.getText();
        if(checkbox2.isChecked())
            str = str +checkbox2.getText();
        if(checkbox3.isChecked())
            str = str +checkbox3.getText();
        if(checkbox4.isChecked())
            str = str +checkbox4.getText();
        Toast.makeText(Ex_45.this, str+"被选择", Toast.LENGTH_LONG).show();
    }
}
```

【代码说明】本实例利用 CheckBox 实现了情景模式的 UI 设计，提供了 4 种手机情景模式（上班模式、家庭模式、旅游模式、会议模式）。四种模式的 XML 定义如下：

```
<string name="app_name">CheckBox</string>
```



```
<string name="Ttile">请选择喜欢的情景模式</string>
<string name="Profile1">上班模式</string>
<string name="Profile2">家庭模式</string>
    <string name="Profile3">旅游模式</string>
    <string name="Profile4">会议模式</string>
<string name="Submit">Submit</string>
```

实例导入相应的类包后,接着调用 `OnCreate()`方法加载 `Activity` 应用。`OnCreate()`方法生成 UI 布局,并根据 XML 布局中定义的组件属性生成复选框组件对象。程序通过 `id` 找到相应的 XML 组件属性,这个 `id` 是程序编译完成后由系统自动生成的。

然后通过 `setOnCheckedChangeListener()`方法注册复选框组件状态改变监听器 `OnCheckedChangeListener`。`OnCheckedChangeListener` 默认是一个接口,需要添加相应的处理逻辑到 `onCheckedChanged()`方法中,否则除了复选框的状态发生变化之外,复选框不做任何处理。`onCheckedChanged()`方法有两个参数: `CompoundButton buttonView` 和 `boolean isChecked`。`CompoundButton` 类是 `CheckBox` 的父类, `CheckBox` 继承了 `CompoundButton` 的方法和属性,故 `CompoundButton` 对象 (`buttonView`) 可调用 `getText()`方法获取复选框的文本内容。参数 `isChecked` 代表当前 `CheckBox` 的选中状态,当单击复选框时,该值会发生变化。可通过该值判断当前 `CheckBox` 是否被勾选, `isChecked` 有两个取值:

- `true` (真值): 复选框处于选中状态。此时若单击复选框, `isChecked` 值变为 `false` (假值)。
- `false` (假值): 复选框处于未选中状态。此时若单击复选框, `isChecked` 值变为 `true` (真值)。

`onCheckedChanged()`方法使用了 `Toast` 作为消息提示的方式。`Toast` 是 Android 中用来显示信息的一种机制,和 `Dialog` 不一样的是, `Toast` 是没有焦点的,而且 `Toast` 显示的时间有限,过一定的时间就会自动消失。通过 `makeText()`方法,可设置消息的显示字符串以及持续时间。`Toast` 类定义了两个 `makeText()`方法:

- `public static Toast makeText (Context context, Charsequence charset, int duration)`: 生成包含文本的标准 `Toast` 对象。`context` 是使用的上下文,通常是本程序的 `Application` 或 `Activity` 对象; `charset` 是显示的字符串; `duration` 是信息的存续期间,值为 `LENGTH_SHORT` 或 `LENGTH_LONG`。
- `public static Toast makeText (Context context, int resourceid int duration)`: 生成从资源中取得的文本的标准 `Toast` 对象。`context` 是使用的上下文,通常是本程序的 `Application` 或 `Activity` 对象; `resourceid` 是要使用的字符串资源 ID,可以是已格式化的文本; `duration` 是信息的存续期间,值为 `LENGTH_SHORT` 或 `LENGTH_LONG`。

通过 `makeText ()`方法生成标准 `Toast` 对象后, `Toast` 对象通过 `setGravity()`方法设置提示信息在屏幕上的显示位置。`setGravity()`方法包含三个参数:

- `Gravity`: `Toast` 对象的显示位置, `Gravity` 提供了多种位置,表 4-13 列举了 `Gravity` 类常用的位置常量。
- `xOffset`: 左右移动偏量,用于水平位置微调。`xOffset` 大于 0 向右移,小于 0 向左移。
- `yOffset`: 上下移动偏量,用于垂直位置微调。`yOffset` 大于 0 向上移,小于 0 向下移。

最后调用 `Toast` 的 `show()`方法将 `Toast` 对象显示出来。除 `show()`、`setGravity()`方法之外, `Toast` 类的主要方法还有:



表 4-13 Gravity 位置常量

名称	整型值	十六进制
AXIS_CLIP	8	0x00000008
AXIS_PULL_AFTER	4	0x00000004
AXIS_Y_SHIFT	4	0x00000004
BOTTOM	80	0x00000050
CENTER	17	0x00000011
CENTER_HORIZONTAL	1	0x00000001
CENTER_VERTICAL	16	0x00000010
CLIP_HORIZONTAL	8	0x00000008
CLIP_VERTICAL	128	0x00000080
VERTICAL_GRAVITY_MASK	112	0x00000070

- `public void setMargin (float horizontalMargin, float verticalMargin)`: 设置视图的栏外空白。参数 `horizontalMargin` 为容器的边缘与提示信息的横向空白（与容器宽度的比）。`verticalMargin` 为容器的边缘与提示信息的纵向空白（与容器高度的比）。
- `public void setText (int resId)`: 更新之前通过 `makeText()` 方法生成的 `Toast` 对象的文本内容。参数 `resId` 为 `Toast` 指定的新的字符串资源 ID。
- `public void setText (CharSequence s)`: 更新之前通过 `makeText()` 方法生成的 `Toast` 对象的文本内容。参数 `s` 为 `Toast` 指定的新的文本。
- `public void setView (View view)`: 设置要显示的 `View`。

`Toast` 提供了一种特殊效果的视图组件，该视图以浮于应用程序之上的形式显示。与其他组件不同的是，它不获得焦点，它的目标是尽可能已不显眼的方式，使用户看到程序所提供的信息。

至此，一个情景模式的 UI 设计就完成了，该 UI 利用 `CheckBox` 提供手机情景模式（上班模式、家庭模式、旅游模式、会议模式）选择的界面。当单击 UI 提供的复选框时，程序弹出一个 `Totast` 消息提示相应的复选框被选中或未被选中。但在很短的时间之后，`Toast` 提示框会消失。单击 `Submit` 时，程序弹出一个 `Totast` 消息提示用户已经选择的复选框，此时 `Totast` 消息持续较长一段时间才会消失。这是因为复选框中的 `Toast` 对象使用 `LENGTH_SHORT` 作为持续时间，而按钮中的 `Toast` 对象使用 `LENGTH_LONG` 作为持续时间。启动该 `Android` 程序，然后单击“家庭模式”复选框，程序会显示“家庭模式被选择”的 `Toast` 消息。程序运行结果如图 4-8 所示。



图 4-8 CheckBox 实例运行结果



4.2.6 单项选择（RadioGroup）

单项选择（RadioGroup）组件也被称为单选按钮组。单选按钮同复选框类似，表明一个特定的状态是勾选（on，值为1）还是不勾选（off，值为0）。同复选框的区别是，复选框状态彼此独立，所以可同时选择任意多个，而多个单选按钮通常结合在一起，之间相互不独立。一组单选按钮有且只能有一个被选中。RadioGroup 类用于创建按钮之间相互排斥的单选按钮组，在同一个单选按钮组中勾选一个按钮，则会取消该组中其他已经勾选的按钮的选中状态。初始状态下，所有的单选按钮都未勾选，虽然不能取消一个特定的单选按钮的勾选状态，但可以通过单选按钮组消除它的勾选状态，根据 XML 布局文件中的单选按钮的唯一 ID 标识指定的选择信息。本节将介绍 RadioGroup 类的主要方法以及属性，并通过实例让读者掌握该组件的用法。RadioGroup 类是 LinearLayout 的子类，RadioGroup 类的层次关系如下：

```

android.view.ViewGroup
android.widget.LinearLayout
android.widget.RadioGroup

```

RadioGroup 类提供了用于设置和控制单选按钮组的方法。

表 4-14 列举了 RadioGroup 类主要的方法。

表 4-14 RadioGroup 常用的方法

方法	功能描述	返回值
addView	根据布局指定的属性添加一个子视图	void
check	当传递-1 作为指定的选择标识符时，此方法同 clearCheck()方法作用等效	void
generateLayoutParams	返回一个新的布局实例，这个实例是根据指定的属性集合生成的	ViewGroup.LayoutParams
setOnCheckedChangeListener	注册单选按钮状态改变监听器，单击按钮时，OnCheckedChangeListener 方法被调用	void
setOnHierarchyChangeListener	注册层次结构变化（当子内容添加到该视图或者从该视图中移除时）监听器	void
generateDefaultLayoutParams	返回默认的布局参数。当 View 作为参数传递给 addView(View)而没有布局参数时，就会请求这些参数。如果返回 null，则 addView 会抛出异常	LinearLayout.LayoutParams（默 认的布局参数）
getCheckedRadioButtonId	返回单选按钮组中所选择的单选按钮的标识 ID	int（如果没有勾选，则返回-1）

单选按钮同复选框一样，包含勾选或者不勾选两个状态。复选框提供给用户两种状态（真或假）的选择，但复选框无法提供多个状态的选择。RadioGroup（单选按钮组）可以解决这样的问题，提供多个可选择的状态。通常多个单选按钮构成一个组，单选按钮之间相互影响，最多只能有一个单选按钮被选中。下面以一个实例讲述 RadioGroup 的基本用法。本实例创捷了多个 RadioButton 组件，当选择相应的 RadioButton 时，程序会弹出一个 Toast 消息框提示选择状态。

【实例 4-6】RadioGroup 组件的应用。

```

Public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main); //设置程序布局
}

```



```

/*根据 XML 定义生成取得 TextView、RadioGroup、RadioButton 对象*/
textview = (TextView) findViewById(R.id.textview); //生成 TextView 对象
RG = (RadioGroup) findViewById(R.id.RG); //生成 RadioGroup 对象, 该组包含 6 个单选按钮
RB1 = (RadioButton) findViewById(R.id.RB1); //生成第一个单选按钮对象
RB2 = (RadioButton) findViewById(R.id.RB2); //生成第二个单选按钮对象
RB3 = (RadioButton) findViewById(R.id.RB3); //生成第三个单选按钮对象
RB4 = (RadioButton) findViewById(R.id.RB4); //生成第四个单选按钮对象
RB5 = (RadioButton) findViewById(R.id.RB5); //生成第五个单选按钮对象
RB6 = (RadioButton) findViewById(R.id.RB6); //生成第六个单选按钮对象

RG.setOnCheckedChangeListener(ChangeRadioGroup);
/*使用 setOnCheckedChangeListenerRadioGroup 注册单选按钮状态改变监听器*/
}

private RadioGroup.OnCheckedChangeListener ChangeRadioGroup = new
RadioGroup.OnCheckedChangeListener()
{
    public void onCheckedChanged(RadioGroup group, int checkedId)
    {
        if(checkedId==RB1.getId() && RB1.isChecked()){
            textview.setText(RB1.getText());
            Toast.makeText(Ex_46.this, RB1.getText()+"被选择", Toast.LENGTH_LONG).show();
        }
        /*若 RB1 被选中, textview 显示 RB1 的内容: Android*/
        else if(checkedId==RB2.getId() && RB2.isChecked()){
            textview.setText(RB2.getText());
            Toast.makeText(Ex_46.this, RB2.getText()+"被选择", Toast.LENGTH_LONG).show();
        }
        /*若 RB2 被选中, textview 显示 RB2 的内容: Sysbian*/
        else if(checkedId==RB3.getId() && RB3.isChecked()){
            textview.setText(RB3.getText());
            Toast.makeText(Ex_46.this, RB3.getText()+"被选择", Toast.LENGTH_LONG).show();
        }
        /*若 RB3 被选中, textview 显示 RB3 的内容: WinCE*/
        else if(checkedId==RB4.getId() && RB4.isChecked()){
            textview.setText(RB4.getText());
            Toast.makeText(Ex_46.this, RB4.getText()+"被选择", Toast.LENGTH_LONG).show();
        }
        /*若 RB4 被选中, textview 显示 RB4 的内容: PalmOS*/
        else if(checkedId==RB5.getId() && RB5.isChecked()){
            textview.setText(RB5.getText());

```



```

        Toast.makeText(Ex_46.this, RB5.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB5 被选中, textview 显示 RB5 的内容: Linux*/
    else if (checkedId==RB6.getId() && RB6.isChecked()) {
        textView.setText(RB6.getText());
        Toast.makeText(Ex_46.this, RB6.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB6 被选中, textview 显示 RB6 的内容: iPhoneOS*/
}

};

/*定义 RadioGroup 状态改变事件监听器。当单选按钮的状态发生改变时, onCheckedChanged 方法被调用*/

```

**【代码说明】** 随着手机的发展, 同时支持多种操作系统的手机会越来越普及。本实例利用 RadioGroup 实现了选择手机操作系统的 UI 设计, RadioGroup 提供了 6 种手机操作系统 (Android、Sysbian、WinCE、PalmOS、Linux 以及 iPhoneOS) 的选择。可通过 XML 定义 RadioGroup:

```

<RadioGroup
<!--添加 RadioGroup 属性定义-->
>
<!--第一个 RadioButton -->
<RadioButton
<!--添加 RadioButton 属性-->
/>
<!--第二个 RadioButton -->
<RadioButton
<!--添加 RadioButton 属性-->
/>
<!--第三个 RadioButton -->
<RadioButton
<!--添加 RadioButton 属性-->
/>
<!--根据需要添加更多 RadioButton-->
</RadioGroup>

```

完成 RadioGroup 的 XML 定义后, 通过 findViewById() 方法加载 RadioGroup 布局。本实例的 RadioGroup 包含 6 个单选按钮, 每个单选按钮代表一种操作系统类型。同属一个组, 这些单选按钮之间相互影响, 同时最多只有一个单选按钮被选中。RadioGroup 特殊的 UI 工作模式在很多方面得到了应用, 如调查问卷的单项选择等。

RadioGroup 常用的监听器有 setOnHierarchyChangeListener 和 OnCheckedChangeListener 两种, 其中 OnCheckedChangeListener 监听单选按钮状态改变事件。本实例通过实现 OnCheckedChangeListener 监听器, 在触发相应的事件时, 调用 onCheckedChanged() 方法。onCheckedChanged() 方法通过 checkedId 同每个单选按钮进行比较, 以判断当前发生状态变化的单选



按钮，然后设置 textview 文本视图显示当前被选中的操作系统，并使用 Toast 提示选中的操作系统。

onCheckedChanged()方法包含两个参数：

- RadioGroup group: RadioGroup 对象。
- int checkedId: 当前发生状态改变的单选按钮的 ID，这个 ID 可唯一标识一个单选按钮。

因此可通过 getId()方法同 checkedId 比较，来判断当前发生状态变化的单选按钮。

至此，一个选择手机操作系统的 UI 设计就完成了，该 UI 利用 RadioGroup 提供操作系统选择的界面。启动该 Android 程序，然后单击 Android 单选按钮，程序运行结果如图 4-9。

图 4-9 表明当 Android 单选按钮被选中时，文本框视图的内容为该操作系统名称，并且程序弹出一个 Totast 消息提示相应的单选按钮被选中，此时 Totast 消息持续较长一段时间才会消失。这是因为 RadioGroup 的 Toast 对象使用 LENGTH\_SHORT 作为持续时间。

然后单击 Linux 单选按钮，程序运行结果如图 4-10 所示。



图 4-9 单击 Android 单选按钮后的界面



图 4-10 单击 Linux 单选按钮后的界面

图 4-10 表明单击另一个单选按钮时，当前被选中的单选按钮的状态会变为未选中状态。这是因为同属一个组的单选按钮之间相互影响，最多只能有一个单选按钮被选中。一个单选按钮的状态变化会引起同组其他单选按钮的状态变化。

4.2.7 下拉列表（Spinner）

Spinner 功能类似于 RadioGroup，相比 RadioGroup，Spinner 提供了体验性更强的 UI 设计模式。一个 Spinner 对象包含多个子项，每个子项只有两种状态：选中或者未被选中。Spinner 类是 AbsSpinner 的子类，Spinner 类的层次关系如下：

```

android.view.ViewGroup
android.widget.AdapterView<T extends android.widget.Adapter>
android.widget.AbsSpinner
    android.widget.Spinner
    
```

Spinner 类提供了用于设置和控制下拉列表的方法。

表 4-15 列举了 Spinner 类主要的方法。



表 4-15 Spinner 常用的方法

方法	功能描述	返回值
getBaseline	获取组件文本基线的偏移	int（组件文本基线的偏移量，如果这个控件不支持基线对齐，那么返回-1）
getPrompt	获取被聚焦时的提示消息	CharSequence
performClick	效果同鼠标单击一样，会触发 OnClickListener	Boolean（true: 调用指定 OnClickListener。false: 调用指定的 OnClickListener 失败）
setOnItemClickListener	此方法不可用。Spinner 不支持 item 单击事件，调用此方法将引发异常	void
setPromptId	设置对话框弹出时显示的文本	void
setOnItemSelectedListener	设置下拉列表子项被选中监听器	void

Spinner 同 RadioGroup 一样，多个子元素组合成一个 Spinner。多个子元素之间相互影响，最多只能有一个被选中。Spinner 提供了更为丰富的 UI 设计模式，下面以一个实例讲述 Spinner 的基本用法。

【实例 4-7】Spinner 组件的应用。

```

Public class Ex_47 extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //设置布局

        Spinner spinner = (Spinner)findViewById(R.id.spinner); //根据 XML spinner 定义生成 Spinner 对象
        ArrayAdapter<?> adapter = ArrayAdapter.createFromResource(this, R.array.color, t, android.R.
            layout.simple_spinner_item );
        /*根据 appList 以 simple_spinner_item 方式生成数组适配器*/
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        //设置下拉列表样式
        spinner.setAdapter(adapter); //通过 setAdapter 读取 ArrayAdapter 中的数据
        OnItemSelectedListener itemSelectedListener= new OnItemSelectedListener() {
            public void onItemSelected(AdapterView<?> parent, View view,
                int position, long id) {
                Toast.makeText(Ex_47.this,"选择的色彩: " + parent.getItemAtPosition(position). toString(),
                    Toast.LENGTH_LONG).show();
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0) {
                Toast.makeText(Ex_47.this,"Nothing is selected", Toast.LENGTH_LONG).show();
            }
        };
    }
};
    
```



```

/*OnItemSelectedListener 定义子元素选择监听器。当子元素被选择时，该类的 onItemSelected
被调用*/
spinner.setOnItemSelectedListener(itemSelectedListener);
/*spinner 通过 setOnItemSelectedListener 注册下拉列表子元素选择监听器*/
}
}

```

【代码说明】 本实例利用 Spinner 实现颜色选择的下拉列表。Spinner 通过数组适配器读取 XML 中定义的颜色子元素。这种设计方式被称为适配器模式，适配器模式建议定义一个包装类，包装有不兼容接口的对象。这个包装类指的就是适配器（Adapter），它包装的对象就是适配者（Adaptee）。适配器提供客户类需要的接口，适配器接口的实现是把客户类的请求转化为对适配者的相应接口的调用。换句话说，当客户类调用适配器的方法时，在适配器类的内部调用适配者类的方法，这个过程对客户类是透明的，客户类并不直接访问适配者类。因此，适配器能使由于借口不兼容而不能交互的类一起工作。Android 系统主要提供了三种适配器，不同适配器对应不同的应用：

- ArrayAdapter: 需要把数据放入一个数组以便显示。
- SimpleCursorAdapter: 数据库应用有关的适配器。
- SimpleAdapter: SimpleAdapter 可定义多种布局，包括 ImageView（图片）、Button（按钮）以及 CheckBox（复选框）。

createFromResource() 方法用于生成适配器，其原型为：ArrayAdapter.createFromResource (Context context, int textArrayResId, int textViewResId)。createFromResource 方法包含三个参数：

- context: 程序上下文，一般使用 this。
- textArrayResId: 数组资源 ID。本实例通过 string-array 标签生成数组，编译后系统会生成一个 ID 来标识该数组。
- textViewResId: 文本视图资源 ID，如 android.R.layout.simple\_spinner\_item 等。

创建 ArrayAdapter 对象后，可通过 setAdapter() 来读取 ArrayAdapter 中的数据。

最后 Spinner 对象通过 setOnItemSelectedListener (itemSelectedListener) 方法注册下拉列表子元素选择监听器。ItemSelected 是 Spinner 组件常用的监听事件，当子元素被选择时，该类的 onItemSelected() 方法被调用。onItemSelected() 方法包含四个参数：

- AdapterView<?> parent: 父类对象。
- View view: 视图对象。
- int position: 位置参数。
- long id: 标识。

至此，完成了颜色选择的下拉列表 UI 设计。该 UI 利用 Spinner 提供下拉列表的界面。启动该 Android 程序，运行结果如图 4-11 所示。

单击 Spinner 组件，界面显示一个下拉列表，如图 4-12 所示。

#### 4.2.8 自动完成文本框视图（AutoCompleteTextView）

上网搜索时，只要输入几个文字，搜索框就会显示相近的关键字。Android 提供了实现这种功能的控件——AutoCompleteTextView。AutoCompleteTextView 是一个文本框视图



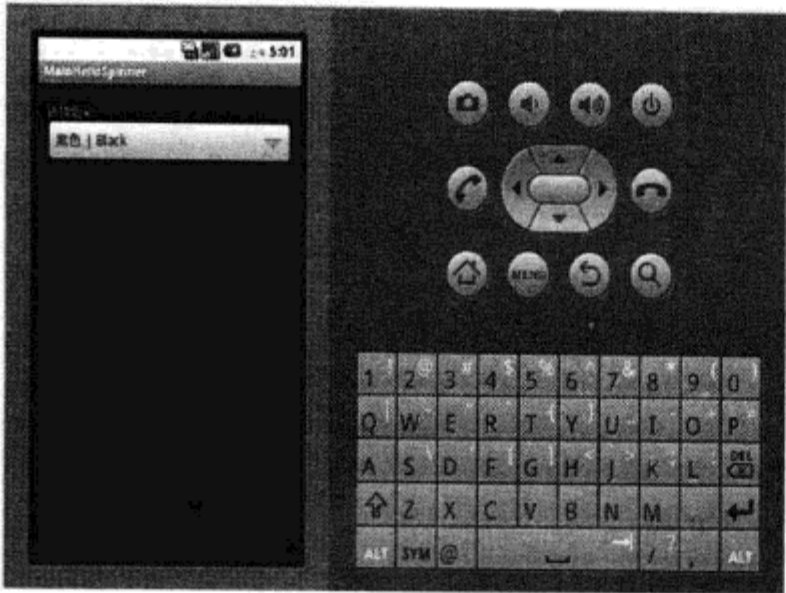


图 4-11 启动界面

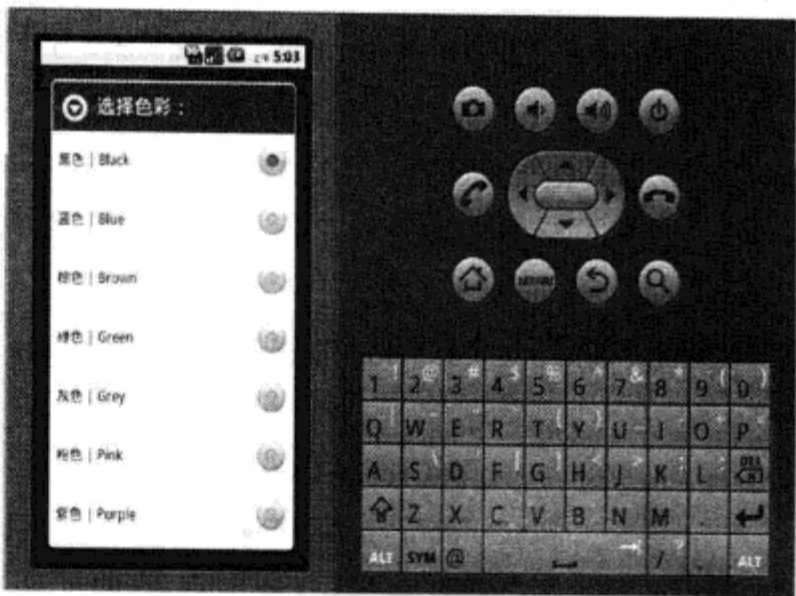


图 4-12 单击下拉列表

组件，它提供了自动完成文本功能。在 Android 中，AutoCompleteTextView 类是 EditText 类的子类，Auto CompleteTextView 派生出 MultiAutoCompleteTextView 类。AutoCompleteTextView 类的层次关系如下：

```

java.lang.Object
android.view.View
android.view.TextView
android.widget.EditText
android.widget.AutoCompleteTextView
MultiAutoCompleteTextView

```

AutoCompleteTextView 组件可以对用户输入的文本进行有效的扩充提示，而不需要用户输入整个文本内容。用户只需输入一部分内容，剩下的部分系统会给予提示。使用 AutoCompleteTextView 时，必须提供一个 MultiAutoCompleteTextView.Tokenizer 对象以用来区分不同的子串。

AutoCompleteTextView 类提供了用于设置和控制自动完成文本的方法。表 4-16 列举了 AutoCompleteTextView 类主要的方法。

表 4-16 AutoCompleteTextView 常用的方法

方法	功能描述	返回值
setMarqueeRepeatLimit	在指定 marquee 的情况下，设置重复滚动的次数，当设置为 marquee_forever 时表示无限次	void
enoughToFilter	当文本长度超过阈值时过滤	boolean(true/false)
performValidation	确定文本中的单个符号的有效性	void
setTokenizer	设置分词组件，该组件决定用户正在输入文本的范围	void
performFiltering	过滤从函数 findTokenStart()到函数 getSelectionEnd()获得的长度为 0 或者超过了预定义的文本内容	void
replaceText	根据参数的文本替换从函数 findTokenStart()到函数 getSelectionEnd()得到的文本	void

下面以一个实例说明 AutoCompleteTextView 的功能，实例 4-8 利用 AutoCompleteTextView 实现了电话号码自动输入的 UI。通过该实例，读者能掌握 AutoCompleteTextView 的基本用法。



#### 【实例 4-8】AutoCompleteTextView 组件的应用。

```
Public class Ex_48 extends Activity {
    private static final String[] phonenumStr = new String[] { "88888888", "85668888", "7777777",
        "86666666", "7377777" }; //自动输入字符串库

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //加载 main.xml 布局

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_dropdown_
            item_1line, phonenumStr);
        /* 以 phonenumStr 字符串数组生成 ArrayAdapter 对象 */
        AutoCompleteTextView autoCompleteTextView = (AutoCompleteTextView) findViewById(R.id.auto
            CompleteTextView);
        /* 以 findViewById()取得 AutoCompleteTextView 对象 */
        autoCompleteTextView.setAdapter(adapter);
        /* 通过 setAdapter()读取 ArrayAdapter 中的数据 :phonenumStr*/

    }
}
```

【代码说明】 本实例首先在 Layout 中部署一个 AutoCompleteTextView 组件：

```
<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="Please enter phone number"
/>
```

然后通过 ArrayAdapter 构造函数将预先设置好的电话号码数组 phonenumStr 放入 ArrayAdapter 中，最后 AutoCompleteTextView 调用 setAdapter()方法读取 ArrayAdapter 中的数据。当用户输入字符串时，AutoCompleteTextView 就会在 ArrayAdapter 中查找匹配的字符串。若仅存在一个字符串，AutoCompleteTextView 控件就会显示出该字符串。例如输入 85，AutoCompleteTextView 控件就会显示相应字符串，如图 4-13 所示。

#### 4.2.9 日期选择器 (DatePicker)

作为提供手机应用开发的系统，Android 系统提供了 DatePicker 和 TimePicker 组件用于实现时间选择器。其中，DatePicker 是一个选择日期的布局视图，它提供了日期选择器的功能。从层次关系上看，DatePicker 是 FrameLayout 和 ViewGroup 的子类，直接继承 FrameLayout 类。DatePicker 类的层次关系如下：



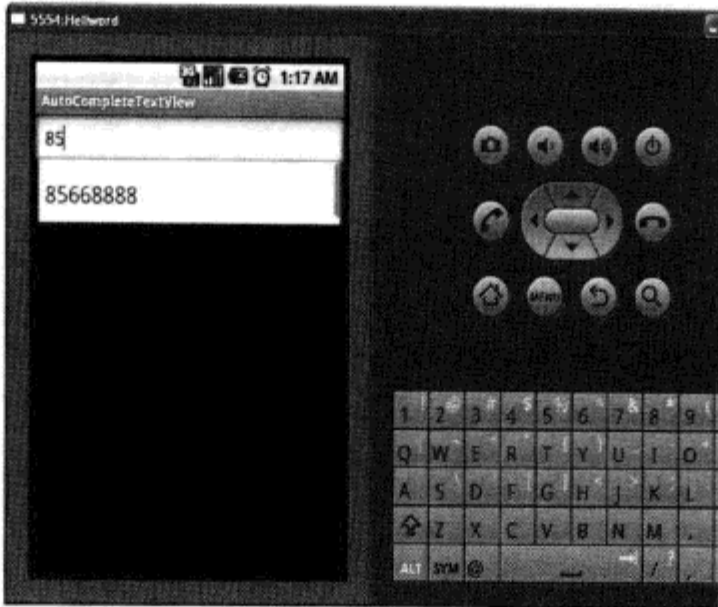


图 4-13 实例 4-8 运行结果

java.lang.Object  
 android.view.View  
 android.view.ViewGroup  
 android.widget.FrameLayout  
 android.widget.DatePicker

表 4-17 列举了 DatePicker 类主要的方法。

下面以一个实例说明 DatePicker 的功能，实例 4-9 利用 DatePicker 和 Calender 实现了日期选择器。

表 4-17 DatePicker 常用的方法

方法	功能描述	返回值
setonDateChangeListener	注册日期改变监听器，当日期发生改变时，onDateChangd 被触发	void
getDayOfMonth	该方法用于获取月份中的日期	int
getMonth	该方法用于返回月份值	int（返回月份减一值，故返回值范围为 0~11）
getYear	该方法用于返回年份值	int
init	该方法用于重置年月日值	void
updateDate	该方法用于日期的更新	void
dispatchRestoreInstanceState	重新应用以前状态，该状态通过 onSaveInstanceState 方法生成。该方法使用的状态参数不能为空	void
onSaveInstanceState	生成一个代表内部的状态，该状态可被 dispatchRestoreInstanceState 方	Parcelable
performFiltering	过滤从函数 findTokenStart()到函数 getSelectionEnd()获得的长度为 0 或	void
replaceText	根据参数的文本替换从函数 findTokenStart()到函数 getSelectionEnd()得	void
	到的文本	

【实例 4-9】DatePicker 组件的应用。

Public class Ex\_49 extends Activity {



```

private DatePicker datepicker; //声明一个私有的时间选择器对象
private TextView textview; //声明一个私有文本框视图
Calendar calendar; //声明 Calendar 对象
int cur_year, cur_month, cur_day; //声明日期变量

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
    setContentView(R.layout.main); //根据 main.xml 生成布局

    datepicker = (DatePicker) this.findViewById(R.id.DatePicker);
    /*根据 XML 的 DatePicker 标签中的定义生成 datepicker*/
    textview = (TextView) this.findViewById(R.id.TextView);
    /*根据 XML 的 TextView 标签中的定义生成 textview*/
    calendar = Calendar.getInstance(); //使用 getInstance 方法生成 Calendar 对象
    cur_year = calendar.get(Calendar.YEAR); //获取当前的年
    cur_month = calendar.get(Calendar.MONTH + 1); //获取当前的月
    cur_day = calendar.get(Calendar.DAY_OF_MONTH); //获取当前的天

    textview.setText("当前时间: " + cur_year + "年" + cur_month + "月" + cur_day + "日");
    /*显示当前的日期*/
    datepicker.init(cur_year, cur_month, cur_day, new MyDateChangeListener());
    //注册日期改变监听器
}
/* 首次启动 Activity 时, onCreate 方法被调用。若再次启动 Service, 不会再执行 onCreate 方法,
而是直接执行 onStart 方法 */
private class MyDateChangeListener implements OnDateChangeListener
{
    public void onDateChanged(DatePicker view, int year,
        int monthOfYear, int dayOfMonth) {
        // TODO Auto-generated method stub
        cur_year = year;
        cur_month = monthOfYear;
        cur_day = dayOfMonth;
        textview.setText("当前时间: " + cur_year + "年" + cur_month + "月" + cur_day + "日");
    }
}
/*MyDateChangeListener 类实现日期改变监听器的功能, 当日期改变时, 调用 onDateChanged 方法
更新日期*/
}

```

**【代码说明】** 本实例通过 DatePicker 实现了日期选择器功能, 用户可以通过日期选择器



修改日期。调整后的日期会在文本框视图 textview 中显示。

首次启动 Activity 时, onCreate()方法被调用。onCreate()方法首先使用 Calendar 对象获取当前的年、月、日,而 DatePicker 对象可使用 init()方法生成日期选择器并监听日期的改变。

DatePicker 类的 init(int cur\_year, int cur\_month, int cur\_day, OnDateChangeListener OnDateChangeListener) ()方法包含四个参数:

- cur\_year: 表示年。
- cur\_month: 表示月。
- cur\_day: 表示天。
- OnDateChangeListener: 日期改变监听器。

对于 DatePicker, 其常用的事件就是日期变化。而日期变化对应的监听器就是 OnDateChangeListener。当日期发生改变时, OnDateChangeListener 的 onDateChanged()方法被调用。这个过程同 Button 类的 OnClickListener 相似,当单击按钮时,其对应的监听器的 onClick()方法被调用。由于 OnDateChangeListener 是一个接口类,需要使用关键字 implements 继承 OnDateChangeListener 并实现 OnDateChangeListener 的 onDateChanged()方法,否则系统日期的变化不会引起 DatePicker 的日期的变化。因此,本实例定义了 MyDateChangeListener 类,该类继承并实现了 OnDateChangeListener 的 onDateChanged()方法。

onDateChanged()方法声明为 onDateChanged(DatePicker view, int year,int monthOfYear, int dayOfMonth) 。其参数如下:

- DatePicker view: 当前发生变化的时间选择器。
- int year: 当前时间选择器的年份。
- int monthOfYear,: 当前时间选择器的月份。
- int dayOfMonth: 当前时间选择器的日期。

因此,当 DatePicker 的日期发生变化时,DatePicker 会将发生变化后的日期值通过传递参数的形式传送给 onDateChanged()方法。这些形参显示在文本框视图 textview 中,若用户改变日期选择器的时间,当前日期值可在 textview 中显示出来。

在 Eclipse 中,将该 Android 应用程序部署到模拟器中。在 Android 模拟器中,单击启动该程序。系统启动后,界面如图 4-14 所示。

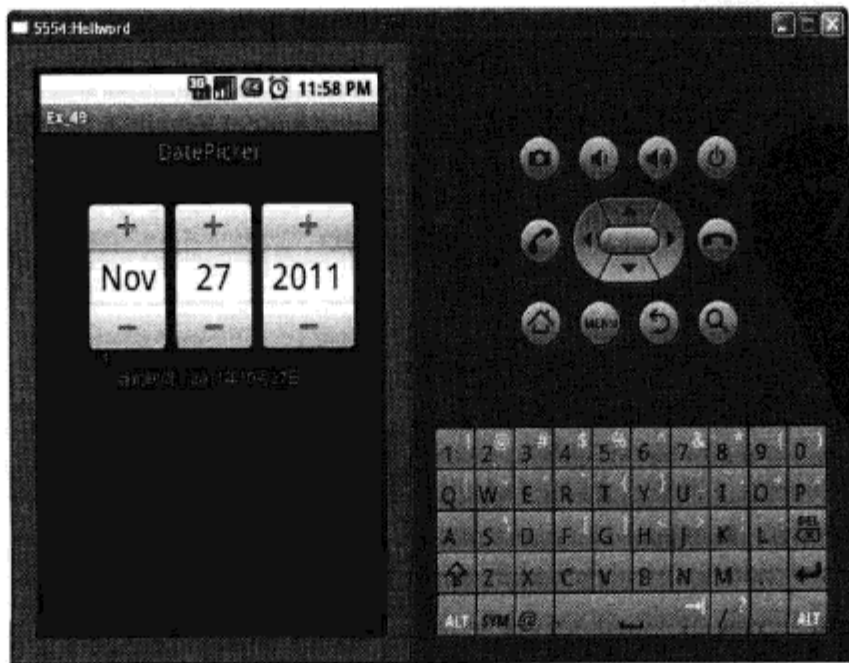


图 4-14 日期选择器运行结果



图 4-14 是时间选择器的运行结果，每个日期属性（年、月、日）分别对应一个可修改的组件。可通过单击“+”或者“-”来调整时间，不难理解单击“+”可将属性值变大，而单击“-”可将属性值变小。

至此，通过系统提供的 DatePicker，仅需修改少量代码就可实现一个界面友好的日期选择器。并且用户无须关心其实现的复杂性，只须导入开发包即可实现包含该功能的日期选择器。到这里读者会问，DatePicker 只提供了日期的选择，不提供时间（时、分）选择的功能。在 Android 系统中，是否具有时间选择器功能的类呢？答案是肯定的，TimePicker（时间选择器）可实现时间选择的功能。

4.2.10 时间选择器（TimePicker）

上一节讲述了通过 DatePicker 实现日期选择器的方法。Android 系统不仅提供了日期选择器，还提供了 TimePicker 组件用于提供时间选择器的功能。DatePicker 支持 24 小时制及上午/下午模式。小时、分钟及上午、下午都可以用垂直滚动条来调整。TimePicker 的层次关系与 DatePicker 一样，TimePicker 是 FrameLayout 和 ViewGroup 的子类，并直接继承 FrameLayout 类。TimePicker 类的层次关系如下：

```

android.widget.FrameLayout
android.widget.TimePicker

```

表 4-18 列举了 TimePicker 类主要的方法。

表 4-18 TimePicker 常用的方法

方法	功能描述	返回值
TimePicker	提供了三个构造函数：  <code>public TimePicker(android.content.Context context)</code> <code>public TimePicker(Context contextandroid.util.AttributeSet attrs)</code> <code>public TimePicker(Context context, AttributeSet attrs, int defStyle)</code>	null
setOnTimeChangeListener	注册时间改变监听器，当时间发生改变时，onTimeChanged 被触发	void
setEnabled	设置当前视图是否可以编辑	int（返回月份减一值，故返回值范围为 0~11）
onSaveInstanceState	生成一个代表内部的状态，该状态可被 dispatchRestoreInstanceState 方法用来重构状态。这种状态应该只包含非持久或不能够重建的信息	Parcelable
onRestoreInstanceState	允许一个视图恢复到某个状态，这个状态是用 onSaveInstanceState 方法保存的状态。注意，state 参数不能为空	void
getCurrentHour	获取当前时间对应的小时	void
setIs24HourView	设置是否使用 24 小时制表示时间	void
getCurrentMinute	获取当前时间对应的分钟	void
getBaseline	返回窗口空间的文本基准线到其顶边界的偏移量。如果不支持基准线对齐，则返回-1	Int（不支持基准线对齐，则返回-1）

下面以一个实例说明 TimePicker 的功能，实例 4-10 利用 TimePicker 和 Calender 实现了时间选择器。



### 【实例 4-10】TimePicker 组件的应用。

```

Public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
    setContentView(R.layout.main); //根据 main.xml 生成布局
    timepicker = (TimePicker) this.findViewById(R.id.TimePicker);
    /*根据 XML 的 TimePicker 标签中的定义生成 timepicker*/
    textview = (TextView) this.findViewById(R.id.TextView);
    /*根据 XML 的 TextView 标签中的定义生成 textview*/
    calendar = Calendar.getInstance(); //使用 getInstance 方法生成 Calendar 对象
    cur_hour = calendar.get(Calendar.HOUR); //获取当前的小时
    cur_minute = calendar.get(Calendar.MINUTE); //获取当前的分钟
    textview.setText("当前时间: " + cur_hour + "时" + cur_minute + "分"); //显示当前的时间
    timepicker.setIs24HourView(true); //设置时间为 24 小时制, 而非上午/下午模式
    timepicker.setOnTimeChangedListener(new MyTimeChangedListener()); //注册时间改变监听器
}
/* 首次启动 Activity 时, onCreate 方法被调用。若再次启动 Service, 不会再执行 onCreate 方法,
   而是直接执行 onStart 方法 */
private class MyTimeChangedListener implements OnTimeChangedListener
{
    public void onTimeChanged(TimePicker view, int hourOfDay, int minute)
    {
        // TODO Auto-generated method stub
        cur_hour = hourOfDay;
        cur_minute = minute;
        textview.setText("当前时间: " + cur_hour + "时" + cur_minute + "分"); //显示当前的日期
    }
}
/*MyTimeChangedListener 类实现时间改变监听器的功能, 当时间改变时, 调用 onTimeChanged 方法更新日期*/

```

**【代码说明】** 本实例通过 TimerPicker 实现了时间选择器功能, 用户可以通过时间选择器修改时间。调整后的时间会在文本框视图 textview 中显示。

启动 Activity 后, 程序首先根据 XML 定义生成 UI, 接着获取当前的小时以及当前的分钟。本实例使用 getInstance() 方法获取 Calendar 对象, 注意 getInstance() 方法同使用 new() 方法新建对象是有区别的。使用 getInstance() 方法被称为 Singleton 模式, 这种模式保证了一个类只有一个实例对象存在。多次调用 getInstance() 方法返回同一个对象, 这种机制在很多应用中都非常有帮助, 比如建立网络连接、文件目录等。然后时间选择器使用 setIs24HourView() 方法设置时间模式。当参数为 true 时, setIs24HourView() 方法设置时间模式为 24 小时制, 而非上午/下午模式。时间选择器使用 setOnTimeChangedListener() 方法注册时间改变监听器, 其监听器对象为 MyTimeChanged Listener 对象。MyTimeChangedListener 是 OnTimeChangedListener 的子类, 该类实现了 onTime Changed() 方法。onTimeChanged() 方法声明为 onTimeChanged(TimePicker



view, int hourOfDay, int minute) 。其参数如下：

- DatePicker view： 当前发生变化的时间选择器。
- int hourOfDay： 当前时间选择器的小时。
- int minute,： 当前时间选择器的分钟。

当 TimerPicker 的时间发生变化时, TimerPicker 会将发生变化后的时间值通过传递参数的形式传送给 onTimeChanged()方法。本例使用文本框视图 textview 显示这些形参，若用户改变日期选择器的时间，当前日期值可在 textview 中显示出来。在 Eclipse 中，将该 Android 应用程序部署到模拟器中。在 Android 模拟器中，单击启动该程序。系统启动后，界面如图 4-15 所示。

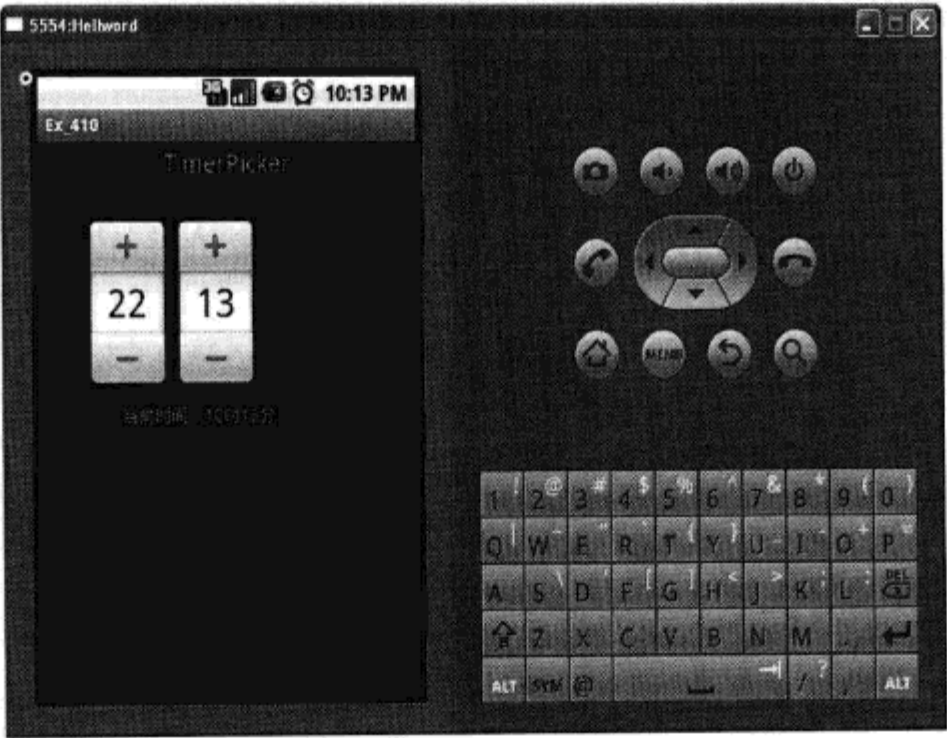


图 4-15 时间选择器运行结果

4.2.11 数字时钟（DigitalClock）

除了日期和时间选择器外,Android 系统本身还提供了两个时钟类:数字时钟(DigitalClock)和表状时钟 (AnalogClock)。DigitalClock 是显示类似于电子日历的时钟的类，这种时钟可以显示系统的标准时间。它可以选择 12 小时制或者 24 小时制，支持时区设置，可以调整颜色。图 4-16 显示了两个数字时钟的例子。



图 4-16 数字时钟（DigitalClock）

事实上 DigitalClock 可以被看成一个文本框视图，因为 DigitalClock 确实是 android.widget.TextView 的子类。Android.widget.DigitalClock 类的层次关系如下：

```

java.lang.Object
android.view.View
android.widget.TextView
android.widget.DigitalClock

```

DigitalClock 类主要使用父类（android.widget.TextView）提供的方法，DigitalClock 本身提



供的方法不多，只有四个。其中包括两个构造函数：`publicDigitalClock (Context context)`和 `publicDigitalClock (Context context, AttributeSet attrs)`，还包括两个窗体事件的响应方法：`onAttachedToWindow` 和 `onDetachedFromWindow`。表 4-19 列举了 `DigitalClock` 类主要的方法。

表 4-19 `DigitalClock` 常用的方法

方法	功能描述	返回值
<code>DigitalClock</code>	提供了两个构造函数：  <code>publicDigitalClock (Context context)</code> <code>publicDigitalClock (Context context, AttributeSet attrs)</code>	<code>null</code>
<code>onAttachedToWindow()</code>	该方法在视图附加到窗体时调用。当视图附加到窗体时，视图将开始绘制用于显示的界面。注意，要保证该方法被调用之前调用了 <code>onDraw(Canvas)</code> 方法	<code>void</code>
<code>computeVerticalScrollExtent()</code>	继承的 <code>android.widget.TextView</code> 的方法。用于计算滚动条把手在纵向滚动范围内占用的幅度。该值用于计算滚动条把手在滚动条滑道中的长度。注意 <code>computeVerticalScrollExtent</code> 默认的长度是视图的可绘制高度。计算单位与 <code>computeVerticalScrollRange()</code> 和 <code>computeVerticalScrollOffset()</code> 相同	<code>int</code>
<code>computeScroll()</code>	继承的 <code>android.widget.TextView</code> 的方法。用于计算滚动条把手在纵向滚动范围内占用的幅度。该值用于计算滚动条把手在滚动条滑道中的长度	<code>void</code>
<code>onDetachedFromWindow()</code>	从窗体分离事件的响应方法。当视图（ <code>DigitalClock</code> ）从窗体上分离（移除）时调用	<code>void</code>
<code>computeVerticalScrollRange()</code>	继承的 <code>android.widget.TextView</code> 的方法。用于计算滚动条代表的纵向范围。该方法使用的范围与 <code>computeVerticalScrollExtent()</code> 和 <code>computeVertical Scroll Offset()</code> 相同	<code>int</code>
<code>dispatchPopulateAccessibilityEvent(AccessibilityEvent event)</code>	继承的 <code>android.widget.TextView</code> 的方法。用于分发 <code>AccessibilityEvent</code> 事件到子视图中	<code>Boolean</code> (成功分发返回 <code>true</code> ，否则返回 <code>false</code> )
<code>drawableStateChanged()</code>	继承的 <code>android.widget.TextView</code> 的方法。该方法在视图状态的变化影响到所显示的可绘制对象的状态时调用	<code>void</code>
<code>getBaseline</code>	继承的 <code>android.widget.TextView</code> 的方法。返回窗口空间的文本基准线到其顶边界的偏移量。如果不支持基准线对齐，这个方法返回-1	<code>int</code> （不支持基准线对齐，则返回-1）
<code>getDefaultEditable()</code>	继承的 <code>android.widget.TextView</code> 的方法。子类覆盖该方法，以在默认 XML 选项未指定可编辑时设置 <code>KeyListener</code>	<code>Boolean</code> (设置成功返回 <code>true</code> ，否则返回 <code>false</code> )
<code>getDefaultMovementMethod()</code>	继承的 <code>android.widget.TextView</code> 的方法。子类覆盖该方法，以指定默认动作方法	<code>MovementMethod</code>
<code>getText()</code>	继承的 <code>android.widget.TextView</code> 的方法。返回 <code>TextView</code> 显示的文本。如果使用 <code>BufferType.SPANNABLE</code> 或 <code>BufferType.EDITABLE</code> 参数调用 <code>setText</code> 方法，可以将本方法的返回值分别转换为 <code>Spannable</code> 或 <code>Editable</code> 。注意，返回值的内容不能修改	<code>CharSequence</code>
<code>dispatchDraw(Canvas canvas)</code>	继承的 <code>android.view.View</code> 的方法。该方法绘出子视图。可被派生类重写，以便在其子项被画出之前取得控制权。此方法由 <code>draw</code> 方法在绘制子视图时调用。子类可以重写该方法，在绘制子视图之前获得控制权	<code>void</code>



下面以一个实例说明 DigitalClock 的功能，实例 4-11 利用 DigitalClock 和 Calender 实现了数字时钟功能。

【实例 4-11】DigitalClock 组件的应用。

主程序 DigitalClockApplication.java 实现，该程序使用 main.xml 初始化界面：

```
public class DigitalClockApplication extends Activity { //创建一个 Activity
    /** Called when the activity is first created. */
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
        setContentView(R.layout.main); //使用 main.xml 初始化界面
    }
}
```

程序 DigitalClock.java 实现：

```
public DigitalClock(Context context, AttributeSet attributeSet) {
    super(context, attributeSet); //调用父类的 DigitalClock 方法
    initialization(context); //根据上下文初始化时间
}

/*DigitalClock(Context context, AttributeSet attrs)是 DigitalClockde 的构造函数*/
public DigitalClock(Context context) {
    super(context); //调用父类的 DigitalClock 方法
    initialization(context); //根据上下文初始化时间
}

/*DigitalClock(Context context)是 DigitalClockde 的构造函数*/

private void initialization(Context context) {
    Resources resource = context.getResources(); //获取上下文的资源
    Uri uri = Settings.System.CONTENT_URI; //定义 uri 为 Settings.System.CONTENT_URI
    formatChangeObserver = new DigitalClockHandler(); //创建格式改变的监视器
    try {
        if (calendar == null) {
            calendar = Calendar.getInstance(); //若日历对象为空，则生成日历 Calendar 类的实例
        }
        Context mycontext = getContext(); //使用 getContext 获取本地上下文
        ContentResolver contentResolver = mycontext.getContentResolver();
        //使用上下文对象获取 ContentResolver
        contentResolver.registerContentObserver(uri, true, formatChangeObserver);
        //使用 registerContentObserver 方法注册事件监听器
        setTimeformat(); //设置格式
    }
}

/*可能出现异常的代码*/
```



```

catch (Exception e)
{
    Toast.makeText(context, "Exception", LONGTIME).show();
    //若出现异常，系统弹出异常的 Toast 消息
}
/*捕获异常的代码*/
Finally
{
    //最后处理的代码
}
}
/*initClock(Context context)根据上下文初始化时间*/
@Override
protected void onAttachedToWindow() {
    ticker = false; //置 ticker 为假值
    super.onAttachedToWindow(); //调用父类的 onAttachedToWindow
    handler = new Handler(); //创建 Handler 对象
    thread = new Runnable() {
        public void run() { //需要实现 Runnable 接口的 run 方法，线程使用 run 作为入口
            if (ticker) return; //只有 ticker 为假值时，才设置时间；否则直接返回
            calendar.setTimeInMillis(System.currentTimeMillis()); //利用日历对象设置时间
            setText(DateFormat.format(format, calendar)); //使用时间格式设置文本
            invalidate();
            long now = SystemClock.uptimeMillis(); //获取当前时间
            long next = now + (1000 - now % 1000); //计算下一个要显示的时间，以毫秒为单位
            handler.postAtTime(thread, next); //设置显示的时间
        }
    };
    /*使用 Runnable 方法创建线程*/
    thread.run(); //运行 thread 线程
}
/*重载父类 DigitalClock 的 onAttachedToWindow 方法。该方法在视图附加到窗体时调用
 * 当视图附加到窗体时，视图将开始绘制用于显示的界面
 * 注意，要保证该方法被调用之前调用了 onDraw(Canvas)方法*/
@Override
protected void onDetachedFromWindow() {
    super.onDetachedFromWindow(); //调用父类的 onDetachedFromWindow
    ticker = true; //置 ticker 为真值
}
/*重载父类 DigitalClock 的 onDetachedFromWindow 方法。该方法为 DigitalClock 从窗体分离事件的

```



响应方法

\* 当视图 (DigitalClock) 从窗体上分离 (移除) 时调用, 这时不再进行画面绘制\*/

```
private boolean return24HourMode() {
    boolean ret = android.text.format.DateFormat.is24HourFormat(getContext());
    //使用 DateFormat 的方法 is24HourFormat 判断是否为 24 小时制模式
    return ret; //返回 bool 值来表示是否为 24 小时制模式
}

private void setTimeformat() {
    if (return24HourMode())
    {
        format = format_24; //format_24="k:mm"
    }
    /*使用 return24HourMode 方法获取当前的时间模式, 若为 24 小时制模式, 则设置 format 为
    k:mm*/
    else
    {
        format = format_12; //format_12="h:mm aa"
    }
    /*使用 return24HourMode 方法获取当前的时间模式, 若为 12 小时制模式, 则设置 format 为
    h:mm aa*/
}

/*setTimeformat 方法通过设置 format 来设置当前的时间模式 (12 小时制模式或者 24 小时制模式)
*该方法调用 return24HourMode 方法来判断当前的时间模式, 根据当前的系统模式来设置 format 值*/
private class DigitalClockHandler extends ContentObserver {
    public DigitalClockHandler() {
        super(new Handler());
    }
    //定义 DigitalClockHandler 的构造方法
    @Override
    public void onChange(boolean flag) {
        setTimeformat();
    }
}
```

//DigitalClockHandler 类继承 ContentObserver 类, 用于监听内容变化

**【代码说明】** 本实例通过 DigitalClock 实现了数字时钟功能, 该实例包含两个文件: 一个是主程序, 另外一个用于生成 DigitalClock。布局为 LinearLayout 且包含一个 Textview 和一个 DigitalClock。本实例的 DigitalClock 类重载父类 android.widget.DigitalClock 的 onAttachedToWindow 方法。该方法在视图附加到窗体时调用。当视图附加到窗体时, 视图将开始绘制用于显示的界面。注意, 要保证该方法被调用之前已经调用了 onDraw(Canvas)方法。用户可以通过



时间选择器修改时间，调整后的时间会在文本框视图 `textview` 中显示。

本实例使用 `Runnable` 接口在视图附加到窗体时创建线程。Android 提供了两种实现线程的方式：`Thread` 和 `Runnable`。其中，`Thread` 是在 `java.lang` 中定义的类，`Runnable` 是在 `java.lang` 中定义的接口。在使用 `Runnable` 接口时需要建立一个 `Thread` 实例，实现了 `Runnable` 接口的类的实例。要注意的是，`Thread` 类也实现了 `Runnable` 接口，因此需要建立 `Thread` 类或它的子类的实例，才能使用 `Thread` 类或 `Runnable` 接口建立线程。

启动 Android 程序后，Android 会自动创建一个名称为“main”的主线程。通常一个程序里面有一个主线程，普通的 Java 程序一般使用包含 `main` 的类作为主线程。然而 Android 程序，使用实现 `onCreate` 方法的类作为主线程。这个由 Android 系统创建的主线程负责初始化界面、设置时间监听事件等。在系统资源出现不足时，Android 会按照进程的优先级停止释放低优先级进程的资源给其他新的进程使用。按照优先级，Android 的进程被分成五个类别：

#### (1) 前台进程

前台进程是当前正在使用的进程，这种进程的优先级最高。

#### (2) 可见进程

这种进程的优先级比前台进程低，但比其他进程要高。除非前台进程需要获取它的资源，不然不会被中止。

#### (3) 服务进程

这种进程由 `startService()` 方法启动。这种进程提供类似于后台进程的功能，服务进程的优先级比前台和可见进程低，但比后台和空进程高。

#### (4) 后台进程

后台进程的优先级低于服务进程，但高于空进程。

#### (5) 空进程

空进程的优先级最低，这种进程是在系统出现资源短缺时最有可能被释放的进程。

最后，在 Eclipse 中启动该程序，程序显示了一个 12 小时制模式的数字时钟，如图 4-17 所示。



图 4-17 数字时钟运行结果



4.2.12 表状时钟（AnalogClock）

上一节讲述了通过 DigitalClock 实现数字时钟的方法。我们知道，Android 系统本身提供了两种时钟：数字时钟（DigitalClock）和表状时钟（AnalogClock）。图 4-18 所示为表状时钟。

表状时钟的功能同数字时钟一样，都是提供时间的显示。但表状时钟与数字时钟不同的是，AnalogClock 不是文本框视图的子类。事实上，AnalogClock 可以被看成一个视图（而非文本框视图），因为 AnalogClock 确实是 android.view.View 的子类。android.widget.AnalogClock 类的层次关系如下：

```

java.lang.Object
  android.view.View
    android.widget..AnalogClock
    
```

AnalogClock 类主要使用父类（android.widget.TextView）提供的方法，它本身只提供了四个方法：onAttachedToWindow()、onDetachedFromWindow()、onDraw (Canvas canvas)、onMeasure (int widthMeasureSpec,int heightMeasureSpec)方法。表 4-20 列举了 AnalogClock 类主要的方法。



图 4-18 表状时钟（AnalogClock）

表 4-20 AnalogClock 常用的方法

方法	功能描述	返回值
AnalogClock	提供了 3 个构造函数：publicAnalogClock (Context context) publicAnalogClock (Context context, AttributeSet attrs) publicAnalogClock (Context context, AttributeSet attrs, int defStyle)	null
onAttachedToWindow()	该方法在视图附加到窗体时调用。当视图附加到窗体时，视图将开始绘制用于显示的界面。注意，要保证该方法被调用之前调用了 onDraw (Canvas)方法	void
onDraw (Canvas canvas)	当绘制视图时，该方法被调用	void
onMeasure (int widthMeasureSpec, int heightMeasureSpec)	当该方法被重写时，必须调用 setMeasuredDimension(int, int)来存储已测量的视图的高度和宽度，否则将通过 measure(int, int)抛出一个 IllegalStateException 异常	void
onDetachedFromWindow()	从窗体分离事件的响应方法。当视图（AnalogClock）从窗体上分离（移除）时调用	void
addFocusables(ArrayList<View> views, int direction)	继承的 android.view.View 的方法。该方法为当前 ViewGroup 中的所有子视图添加焦点获取能力	void
addTouchables(ArrayList<View> views)	继承的 android.view.View 的方法。该方法为子视图添加触摸能力	void
getBaseline()	继承的 android.view.View 的方法。返回窗口空间的文本基准线到其顶边界的偏移量。如果不支持基准线对齐，这个方法返回-1	int（不支持基准线对齐，则返回-1）



续表

方法	功能描述	返回值
dispatchDisplayHint(int hint)	继承的 android.view.View 的方法。该方法分发视图是否显示的提示	void
dispatchDraw(Canvas canvas)	继承的 android.view.View 的方法。调用此方法来绘制子视图。可被派生类重写，以便在子视图被画出之前取得控制权。此方法由 draw 方法在绘制子视图时调用。子类可以重写该方法，在绘制其子视图之前获得控制权	void
dispatchPopulateAccessibilityEvent(AccessibilityEvent event)	分发 AccessibilityEvent 事件到视图的子视图中	boolean（分发成功返回 true，否则返回 false）

下面以一个实例说明 AnalogClock 的功能，实例 4-12 利用 DigitalClock 和 AnalogClock 实现了数字时钟和表状时钟切换的功能。

**【实例 4-12】AnalogClock 组件的应用。**

主程序 AnalogClockApplication.java 实现，该程序使用 analogclock.xml 初始化界面：

```

public class AnalogClockApplication extends Activity {
    private Button analogbutton; //声明按钮控件，该按钮用于启动表状时钟
    private TextView analogtextView; //声明文本框视图控件
    private AnalogClock analogclock; //声明表状时钟控件
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.analogclock); //使用 analogclock.xml 初始化界面
        analogbutton = (Button) findViewById(R.id.analogClockButton); //根据 XML 定义创建按钮对象
        analogtextView = (TextView) findViewById(R.id.analogClockTextView);
        //根据 XML 定义创建文本框视图对象
        analogclock = (AnalogClock) findViewById(R.id.analogClock);
        //根据 XML 定义创建表状时钟对象

        analogtextView.setText("Current clock is AnalogClock");
        //设置文本框视图显示文字，提示当前时钟为表状时钟
        analogbutton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent myintent = new Intent();
                myintent.setClass(AnalogClockApplication.this, DigitalClockApplication.class);
                /*新建一个 Intent 对象，并指定启动程序 analogClockApplication*/
                AnalogClockApplication.this.startActivity(myintent);
                /*程序 AnalogClockApplication 利用 startActivity 调用新的 Activity，这个 Activity 是由 set
                Class 方法指定的*/
                AnalogClockApplication.this.finish(); //关闭当前的 Activity
            }
        });
    }
}

```



/\*定义按钮 analogbutton 的单击监听器。当单击 analogbutton 按钮时，onClick 方法被调用\*/

```
}
}
```

主程序 DigitalClockApplication.java 实现，该程序使用 digitalclock.xml 初始化界面：

```
public class DigitalClockApplication extends Activity {
    /** Called when the activity is first created. */
    private Button digitalbutton; //声明按钮控件，该按钮用于启动数字时钟
    private TextView digitaltexview; //声明文本框视图控件
    private DigitalClock digitalclock; //声明数字时钟控件
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.digitalclock); //使用 digitalclock.xml 初始化界面
        digitalbutton = (Button) findViewById(R.id.digitalClockButton); //根据 XML 定义创建按钮对象
        digitaltexview = (TextView) findViewById(R.id.digitalClockTextView);
        //根据 XML 定义创建文本框视图对象
        digitalclock = (DigitalClock) findViewById(R.id.digitallock); //根据 XML 定义创建数字时钟对象
        digitaltexview.setText("Current clock is DigitalClock");
        //设置文本框视图显示文字，提示当前时钟为数字时钟
        digitalclock.setTextColor(Color.GREEN); //设置数字时钟的颜色为绿色
        digitalbutton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Intent myintent = new Intent();
                myintent.setClass(DigitalClockApplication.this, AnalogClockApplication.class);
                /*新建一个 Intent 对象，并指定启动程序 AnalogClockApplication*/
                DigitalClockApplication.this.startActivity(myintent);
                /*程序 DigitalClockApplication 利用 startActivity 调用新的 Activity，这个 Activity 是由
                setClass 方法指定的*/
                DigitalClockApplication.this.finish(); //关闭当前的 Activity
            }
        });
        /*定义按钮 digitalbutton 的单击监听器。当单击 digitalbutton 按钮时，onClick 方法被调用*/
    }
}
```

**【代码说明】** 本实例通过 DigitalClock 和 AnalogClock 实现了数字时钟和表状时钟切换的功能。本实例包含两个 activity: DigitalClockApplication 和 AnalogClockApplication。DigitalClockApplication 用于实现数字时钟的功能，AnalogClockApplication 用于实现表状时钟的功能。这两个 activity 之间使用 Intent 实现消息传递以及相互调用。对于 DigitalClockApplication，当单击切换按钮时，按钮的单击事件方法被调用。该方法新建一个 Intent 对象，并指定启动程序 analogClockApplication，然后调用 AnalogClockApplication.this.startActivity(myintent)，该方法将调用新的 Activity，这个 Activity 是由 setClass 方法指定的。AnalogClockApplication 的处理过



程同 DigitalClockApplication 类似。

启动该 Android 程序，显示一个表状时钟，如图 4-19 所示。

单击“切换到数字时钟”按钮，由数字时钟代替表状时钟显示时间，如图 4-20 所示。

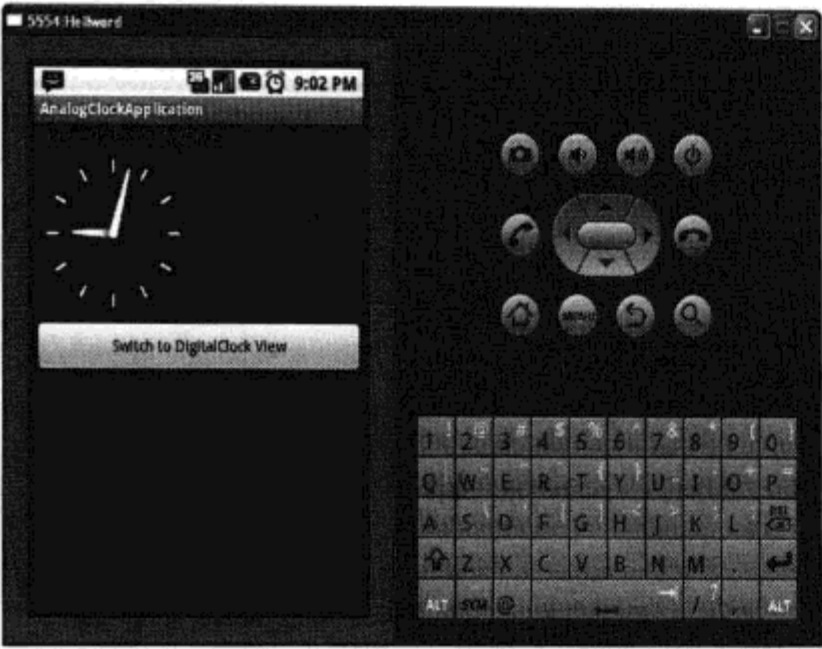


图 4-19 表状时钟（AnalogClock）



图 4-20 数字时钟（DigitalClock）

4.2.13 进度条（ProgressBar）

Android 为用户提供了丰富的应用，本节将为大家介绍一个显示进度的控件——进度条（ProgressBar）。使用手机时，经常会碰到进度条的应用，如打开一个程序时加载界面。进度条可以很形象地提示应用（如正在下载的应用）正在处理中，用户需要等待。图 4-21 展示了一个圆圈进度条。

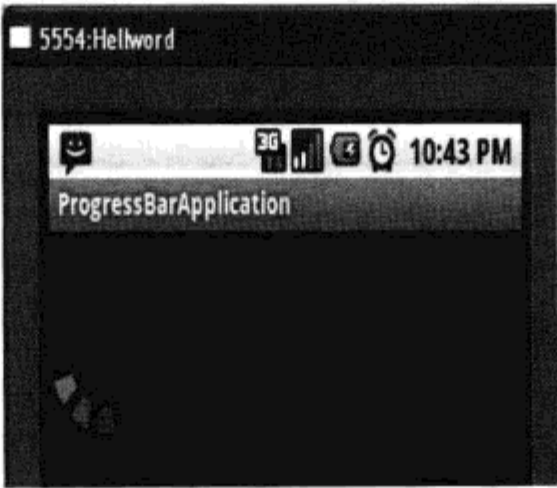


图 4-21 进度条（ProgressBar）

进度条（ProgressBar）是 View 的子类，其在 Android 系统中的层次关系如下：

```

java.lang.Object
android.view.View
android.widget.ProgressBar

```

表 4-21 列举了 ProgressBar 类主要的方法。

下面以一个实例说明 ProgressBar 类的功能，实例 4-13 实现了 3 种进度条（长方形进度条、大圆圈进度条、小圆圈进度条）。



表 4-21 ProgressBar 常用的方法

方法	功能描述	返回值
ProgressBar	提供了 3 个构造函数：  <div> <div>ProgressBar(Context context)</div> <div>ProgressBar(Context context, AttributeSet attrs)</div> <div>ProgressBar(Context context, AttributeSet attrs, int defStyle)</div> </div>	null
onAttachedToWindow()	该方法在视图附加到窗体时调用。当视图附加到窗体时，视图将开始绘制用于显示的界面。注意，要保证该方法被调用之前调用了 onDraw(Canvas)方法	void
onDraw (Canvas canvas)	当绘制视图时，该方法被调用	void
onMeasure (int widthMeasureSpec, int heightMeasureSpec)	当该方法被重写时，必须调用 setMeasuredDimension(int, int)来存储已测量视图的高度和宽度，否则将通过 measure(int, int)抛出一个 IllegalStateException 异常	void
onDetachedFromWindow()	从窗体分离事件的响应方法。当视图（ProgressBar）从窗体上分离（移除）时调用	void
addFocusables(ArrayList<View> views, int direction)	继承的 android.view.View 的方法。该方法为当前 ViewGroup 中的子视图添加焦点获取能力	void
addTouchables(ArrayList<View> views)	继承的 android.view.View 的方法。该方法为子视图添加触摸能力	void
getBaseline	继承的 android.view.View 的方法。返回窗口空间的文本基准线到其顶边界的偏移量	int（不支持基准线对齐，则返回-1）
dispatchDisplayHint(int hint)	继承的 android.view.View 的方法。该方法分发视图是否显示的提示。	void
dispatchDraw(Canvas canvas)	继承的 android.view.View 的方法。调用此方法来绘制子视图。可被派生类重写，以便在子视图被画出之前取得控制权。此方法由 draw 方法在绘制子视图时调用。子类可以重写该方法，在绘制其子视图之前获得控制权	void

**【实例 4-13】ProgressBar 组件的应用。**

主程序 ProgressBarApplication.java 实现，该程序使用 progressbar.xml 初始化界面：

```

public void onCreate(Bundle savedInstanceState)
{
    /** Called when the activity is first created. */
    super.onCreate(savedInstanceState);    //调用父类的 onCreate 方法
    setContentView(R.layout.main);    //使用 main.xml 初始化程序 UI
    button = (Button)findViewById(R.id.button);    //根据 XML 定义创建 button
    textviewHorizontal = (TextView)findViewById(R.id.textviewHorizontal);
    //根据 XML 定义创建 textviewHorizontal
    textviewLarge = (TextView)findViewById(R.id.textviewLarge);
    //根据 XML 定义创建 textview Large
    textviewSmall = (TextView)findViewById(R.id.textviewSmall);
}
    
```



```
//根据 XML 定义创建 textview Small
/* 设置 ProgressBar widget 对象 */
progressBarStyleHorizontal = (ProgressBar)findViewById(R.id.progressBarStyleHorizontal);
//根据 XML 定义创建 ProgressBar 对象 progressBarStyleHorizontal
progressBarStyleLarge = (ProgressBar)findViewById(R.id.progressBarStyleLarge);
//根据 XML 定义创建 ProgressBar 对象 progressBarStyleLarge
progressBarStyleSmall = (ProgressBar)findViewById(R.id.progressBarStyleSmall);
//根据 XML 定义创建 ProgressBar 对象 progressBarStyleSmall
progressBarStyleHorizontal.setIndeterminate(false);
//设置 progressBarStyleHorizontal 的 indeterminate 模式为 false
progressBarStyleLarge.setIndeterminate(false);
//设置 progressBarStyleHorizontal 的 indeterminate 模式为 false
progressBarStyleSmall.setIndeterminate(false);
//设置 progressBarStyleHorizontal 的 indeterminate 模式为 false

button.setOnClickListener(new Button.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // TODO Auto-generated method stub

        textViewHorizontal.setText("水平进度条开始"); //设置 textViewHorizontal 的文本
        textViewLarge.setText("大圆圈进度条开始"); //设置 textViewLarge 的文本
        textViewSmall.setText("小圆圈进度条开始"); //设置 textViewSmall 的文本
        progressBarStyleHorizontal.setVisibility(View.VISIBLE);
        //将隐藏的 progressBarStyle Horizontal 显示出来
        progressBarStyleLarge.setVisibility(View.VISIBLE);
        //将隐藏的 progressBarStyleLarge 显示出来
        progressBarStyleSmall.setVisibility(View.VISIBLE);
        //将隐藏的 progressBarStyle Small 显示出来
        progressBarStyleHorizontal.setMax(100); //指定 Progress 为最多 100
        progressBarStyleLarge.setMax(100); // 指定 Progress 为最多 90
        progressBarStyleSmall.setMax(100); // 指定 Progress 为最多 80

        progressBarStyleHorizontal.setProgress(0); //初始 progressBarStyleHorizontal 为 0
        progressBarStyleLarge.setProgress(10); //初始 progressBarStyleLarge 为 10
        progressBarStyleSmall.setProgress(50); //初始 progressBarStyleSmall 为 50
        /* 开始一个进程 */
        new Thread(new Runnable()
        {
```



```

public void run()
{
    for (int i=0;i<10;i++)
    {
        try
        {
            counter = (i+1)*20;    /* 成员变量, 用以识别加载进度 */
            Thread.sleep(1000);    /* 每运行一次循环, 即暂停 1 秒 */
            /* 当 Thread 运行 5 秒后显示运行结束 */
            if(i==3)
            {
                Message messageHorizontal = new Message();
                //新建信息, 传递参数给 Handler
                Message messageLarge = new Message();
                //新建信息, 传递参数给 Handler
                Message messageSmall = new Message();
                //新建信息, 传递参数给 Handler
                messageHorizontal.what = ProgressBarApplication.StopHandlerHorizontal; /*以 what 属性指定消息为 StopHandlerHorizontal*/
                messageLarge.what = ProgressBarApplication.StopHandlerLarge;
                /*以 what 属性指定 StopHandlerLarge 消息 */
                messageSmall.what = ProgressBarApplication.StopHandlerSmall;
                /*以 what 属性指定 StopHandlerSmall 消息 */

                ProgressBarApplication.this.myMessageHandler.sendMessage(messageHorizontal);
                //发送 messageHorizontal 消息
                ProgressBarApplication.this.myMessageHandler.sendMessage(messageLarge);
                //发送 messageLarge 消息
                ProgressBarApplication.this.myMessageHandler.sendMessage(messageSmall);
                //发送 messageSmall 消息

                break;
            }
            else
            {
                Message messageHorizontal = new Message();
                //新建信息, 传递参数给 Handler
                Message messageLarge = new Message();
                //新建信息, 传递参数给 Handler
                Message messageSmall = new Message();
                //新建信息, 传递参数给 Handler
                messageHorizontal.what = ProgressBarApplication.StartHandler

```



```

Horizontal;
messageLarge.what = ProgressBarApplication.StartHandlerLarge;
messageSmall.what = ProgressBarApplication.StartHandlerSmall;

ProgressBarApplication.this.myMessageHandler.sendMessage(messageHorizontal);
//发送 messageHorizontal 消息

ProgressBarApplication.this.myMessageHandler.sendMessage(messageLarge);
//发送 messageLarge 消息

ProgressBarApplication.this.myMessageHandler.sendMessage(messageSmall);
//发送 messageSmall 消息

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

}).start();
}    /* 默认 0 至 9, 共运行 10 次的循环语句 */
});
}
/* 单击按钮后, 开始进程工作 */
/* Handler 构建之后, 会监听传来的信息代码 */
Handler myMessageHandler = new Handler()
{
    // @Override
    public void handleMessage(Message msg)
    {
        switch (msg.what)
        {
            case ProgressBarApplication.StopHandlerHorizontal:

                textViewHorizontal.setText("progressBarStyleHorizontal 进度条结束");
                /* 显示运行结束 */
                textViewHorizontal.setVisibility(View.GONE); /* 设置 ProgressBar Widget 为隐藏 */
                Thread.currentThread().interrupt();
                break;

            case ProgressBarApplication.StopHandlerLarge:

```



```

        textViewLarge.setText("progressBarStyleLarge 进度条结束");    /* 显示运行结束 */
        textViewLarge.setVisibility(View.GONE); /* 设置 ProgressBar Widget 为隐藏 */
        Thread.currentThread().interrupt();
        break;
    case ProgressBarApplication.StopHandlerSmall:

        textViewSmall.setText("progressBarStyleSmall 进度条结束");    /* 显示运行结束 */
        textViewSmall.setVisibility(View.GONE); /* 设置 ProgressBar Widget 为隐藏 */
        Thread.currentThread().interrupt();
        break;
    case ProgressBarApplication.StartHandlerHorizontal:
        if(!Thread.currentThread().isInterrupted())
        {
            progressBarStyleHorizontal.setProgress(counter);
            /* 将显示进度显示于 TextView 中 */
            textViewHorizontal.setText
            (
                getResources().getText(counter, "进度条开始")+ "("+Integer.toString
                (counter)+"%)\n"+
                "Progress:"+
                Integer.toString(progressBarStyleHorizontal.getProgress())+
                "\n"+"Indeterminate:"+"
                Boolean.toString(progressBarStyleHorizontal.isIndeterminate())
            );
            //设置 textViewHorizontal 的文本
        }
        break;
    case ProgressBarApplication.StartHandlerLarge:
        if(!Thread.currentThread().isInterrupted())
        {
            progressBarStyleLarge.setProgress(counter);
            /* 将显示进度显示于 TextView 中 */
            textViewLarge.setText
            (
                getResources().getText(counter, "进度条开始")+ "("+Integer.toString
                (counter)+"%)\n"+
                "Progress:"+
                Integer.toString(progressBarStyleLarge.getProgress())+
                "\n"+"Indeterminate:"+"

```



```

        Boolean.toString(progressBarStyleLarge.isIndeterminate())
    );
    //设置 textviewLarge 的文本
}
break;
case ProgressBarApplication.StartHandlerSmall:
    if(!Thread.currentThread().isInterrupted())
    {
        progressBarStyleSmall.setProgress(counter);
        /* 将显示进度显示于 TextView 中 */
        textviewSmall.setText
        (
            getResources().getText(counter, "进度条开始")+ "("+Integer.toString
            (counter)+"%)\n"+
            "Progress:"+
            Integer.toString(progressBarStyleSmall.getProgress())+
            "\n"+"Indeterminate:"+
            Boolean.toString(progressBarStyleSmall.isIndeterminate())
        );
    }
    break;
    //设置 textviewSmall 的文本
}
super.handleMessage(msg);
}
};

```

**【代码说明】** 本实例实现了 3 种进度条：progressBarStyleHorizontal（长方形进度条）、progressBarStyleLarge（大圆圈进度条）、progressBarStyleSmall（小圆圈进度条）。可通过在 ProgressBar 中设置 style 属性来指定进度条的类型，style 属性是一个整型值。除了本实例实现的进度条类型，Android 还提供了其他的进度条类型。android.R.attr 类中定义了 Android 使用的进度条，表 4-22 列举了 android.R.attr 类定义的进度条类型。

表 4-22 android.R.attr 类定义的进度条类型

名称	类型	整型值
progressBarStyle	默认进度条	16842871(0x01010077)
progressBarStyleHorizontal	水平进度条	16842872 (0x01010078)
progressBarStyleLargeInverse	倒转圆圈进度条	16843399 (0x01010287)
progressBarStyleLarge	大圆圈进度条	16842874 (0x0101007a)
progressBarStyleSmall	小圆圈进度条	16842873 (0x01010079)
progressBarStyleSmallInverse	倒转小圆圈进度条	16843279 (0x0101020f)



本实例首先使用 main.xml 初始化程序 UI，该 UI 包含 3 个文本框视图控件(textviewHorizontal、textviewLarge 和 textviewSmall)、3 个进度条控件（progressBarStyleHorizontal、progressBarStyleLarge 和 progressBarStyleSmall）和一个按钮控件。其中，progressBarStyle Horizontal 是一个水平的进度条，该控件的宽度和高度随内容进行调整； progressBarStyleLarge 是一个大圆圈的控制条，该控件的宽度和高度也随内容进行调整； progressBarStyleSmall 是一个小圆圈的控制条，该控件的宽度和高度也随内容进行调整。3 个文本框视图控件分别用于显示相应进度条控件的状态，textviewHorizontal、textviewLarge 和 textviewSmall 分别对应 progressBarStyle Horizontal、progressBarStyleLarge 和 progressBarStyleSmall 的状态。

启动该 Android 程序，程序显示 3 个进度条，如图 4-22 所示。

单击“Control”按钮，进度条开始累加，运行结果如图 4-23 所示。

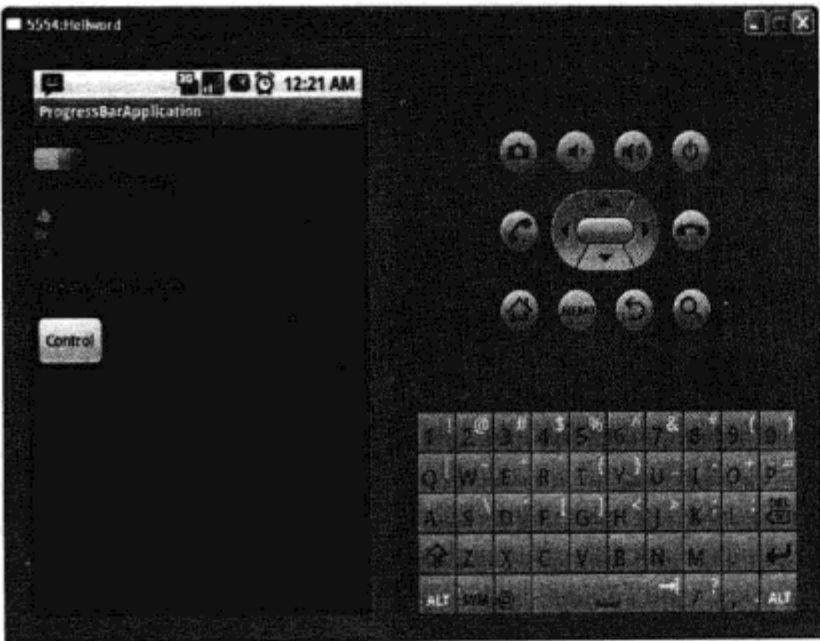


图 4-22 显示 3 个进度条

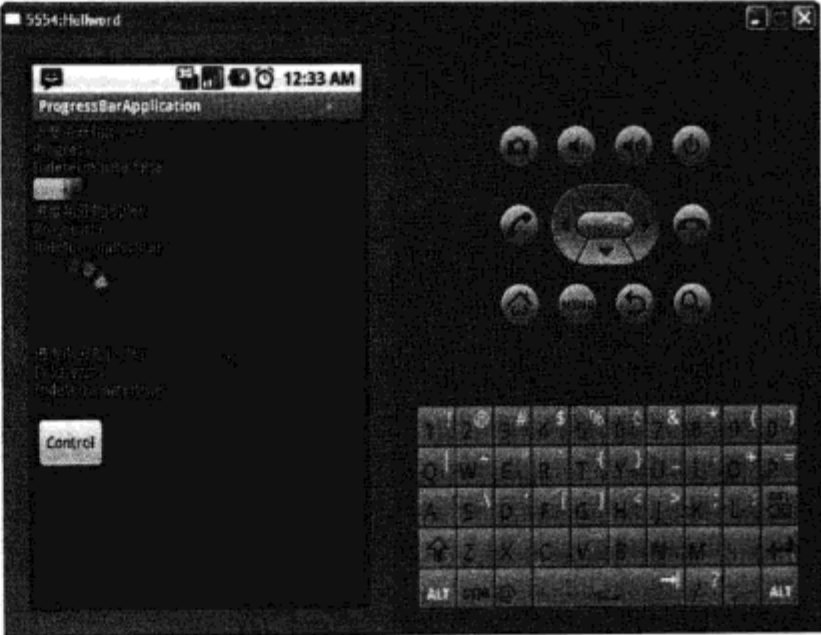


图 4-23 进度条状态改变

4.2.14 拖动条（SeekBar）

SeekBar 同 PorgressBar 不一样，SeekBar 可实现拖动进度条的功能。SeekBar 扩展了 ProgressBar 的功能，在 ProgressBar 基础上增加了一个可拖动的图标。通过 SeekBar 类，开发人员可以很方便地实现进度条的拖动功能，这种功能在许多场景中得到应用，如拖动视频、拖动音量等。图 4-24 显示了一个拖动条。

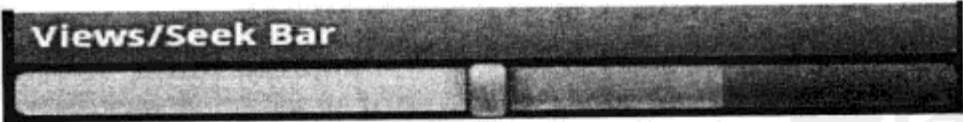


图 4-24 拖动条（SeekBar）

通过 SeekBar 类可实现拖动条的功能。SeekBar 是 PorgressBar 的子类，SeekBar 在 Android 系统中的层次关系如下：

```

android.widget..ProgressBar
android.widget.AbsSeekBar
android.widget.SeekBar

```

作为 ProgressBar 的子类，SeekBar 类主要继承父类的方法和属性，其本身自定义的方法和属性并不多。除了构造函数以外，SeekBar 类只定义了方法 setOnSeekBarChangeListener 和属性 thumb。这个方法用于注册拖动条的监听器，这个监听器用于监听改变拖动条状态的事件。



属性 thumb 用于指定拖动条对应的图标。属性 thumb 是对其他资源的参照，是形式为“@[+][package:]type:name”或“?[package:][type:]name”的主题属性。

表 4-23 列举了 SeekBar 类主要的方法。

表 4-23 SeekBar 常用的方法

方法	功能描述	返回值
SeekBar	提供了 3 个构造函数：  SeekBar(Context context)  SeekBar(Context context, AttributeSet attrs)  SeekBar(Context context, AttributeSet attrs, int defStyle)	null
setOnSeekBarChangeListener (SeekBar.OnSeekBarChangeListener l)	用于注册拖动条的监听器，这个监听器用于监听改变拖动条状态的事件	void
drawableStateChanged()	继承的 android.widget.ProgressBar 的方法。该方法在视图状态的变化影响到所显示的可绘制对象的状态时调用。注意，覆盖该方法时，需要使用 super.drawableStateChanged()调用父类的方法	void
onAttachedToWindow()	继承的 android.widget.ProgressBar 的方法。该方法在视图附加到窗体时调用。当视图附加到窗体时，视图将开始绘制用于显示的界面。注意，要保证在该方法被调用之前调用了 onDraw(Canvas)方法	void
onDraw (Canvas canvas)	继承的 android.widget.ProgressBar 的方法。该方法在绘制视图时被调用	void
onMeasure (int w, int h)	继承的 android.widget.ProgressBar 的方法。当该方法被重写时，必须调用 setMeasuredDimension(int, int)来存储已测量视图的高度和宽度，否则将通过 measure(int, int)抛出一个 IllegalStateException 异常	void

下面以一个实例说明 SeekBar 类的功能，实例 4-14 使用 SeekBar 实现了音频和音量控制功能。

【实例 4-14】SeekBar 组件的应用。

主程序 SeekBarApplication.java 实现，该程序使用 main.xml 初始化界面：

```

public class SeekBarApplication extends android.app.Activity {
    private static int cur_volume=0; //cur_volume 记录了当前音量大小，初始设置为 0
    private final static int MIN_VOLUME=0; //MIN_VOLUME 定义了最小音量值
    private final static int MAX_VOLUME=15; //MAX_VOLUME 定义了最大音量值
    /*定义音量参数： cur_volume、MIN_VOLUME 以及 MAX_VOLUME*/
    private static int cur_audio=0; //cur_voice 记录了当前音频大小，初始设置为 0
    private final static int MIN_AUDIO=0; //MIN_AUDIO 定义了最小音频值
    private final static int MAX_AUDIO=15; //MAX_AUDIO 定义了最大音频值
    /*定义音频参数： cur_voice、MIN_AUDIO 以及 MAX_AUDIO*/
    private int maxvolumeProgress ;//定义 volumesseekbar 最大进度
    private int maxaudioProgress ;//定义 volumesseekbar 最大进度
    private SeekBar volumesseekbar;    //声明拖动条控件对象 volumesseekbar
    private SeekBar audioseekbar;    //声明拖动条控件对象 audioseekbar
    private TextView volumetextview;    //声明文本框视图控件对象 volumetextview

```



```
private TextView audiotextview; //声明文本框视图控件对象 audiotextview

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
    setContentView(R.layout.main); //使用 main.xml 初始化程序 UI
    volumeseekbar = (SeekBar) this.findViewById(R.id.volumeseekbar );
    //根据 XML 定义的控件创建 volumeseekbar, 该控件用于调节音量
    audioseekbar = (SeekBar) this.findViewById(R.id.audioseekbar );
    //根据 XML 定义的控件创建 audioseekbar , 该控件用于调节音频
    volumetextview= (TextView) this.findViewById(R.id.volumetextview);
    //根据 XML 定义的控件创建 volumetextview, 该文本框视图用于显示 volumeseekbar 的状态
    audiotextview= (TextView) this.findViewById(R.id.audiotextview);
    //根据 XML 定义的控件创建 audiotextview, 该文本框视图用于显示 audioseekbar 的状态
    maxvolumeProgress=volumeseekbar.getMax(); //通过 getMax 获取 volumeseekbar 的 Progress
    maxaudioProgress=audioseekbar.getMax(); //通过 getMax 获取 maxaudioProgress 的 Progress
    try
    {
        volumeseekbar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
        {

            @Override
            public void onProgressChanged(SeekBar seekBar, int progress,
                boolean fromUser) {
                cur_volume=(((progress*100)/ maxvolumeProgress )*MAX_VOLUME)/100 ;
                volumetextview.setText("当前音量: "+cur_volume);
            } //拖动时, 该方法被调用

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
                Toast.makeText(SeekBarApplication .this,
                    "volumeseekbar 拖动中...",Toast.LENGTH_LONG).show();
            } //开始拖动时, 该方法被调用

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
                Toast.makeText(SeekBarApplication .this,
                    "volumeseekbar 拖动完毕",Toast.LENGTH_LONG).show();
            } //结束拖动时, 该方法被调用

        }
    );
    /*注册 volumeseekbar 的事件监听器, 需要实现监听器的 onProgressChanged、onStart
    TrackingTouch 和 onStopTrackingTouch 方法*/
}
```



```

audioseekbar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
{
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress,
        boolean fromUser) {
        cur_audio=((progress*100)/maxaudioProgress)*MAX_AUDIO/100;
        //根据拖动条的进度计算当前的音频
        audiotextview.setText("当前音频: "+cur_audio); //显示当前音频
    } //拖动时, 该方法被调用

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
        Toast.makeText(SeekBarApplication.this,
            "audioseekbar 拖动中...", Toast.LENGTH_LONG).show();
    } //开始拖动时, 该方法被调用

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {

        Toast.makeText(SeekBarApplication.this,
            "audioseekbar 拖动完毕", Toast.LENGTH_LONG).show();

    } //结束拖动时, 该方法被调用
});
/*注册 audioseekbar 的事件监听器, 需要实现监听器的 onProgressChanged、onStart
TrackingTouch 和 onStopTrackingTouch 方法*/

} //可能出现异常的代码
catch(Exception e)
{
    Toast.makeText(SeekBarApplication.this, "出现异常" + e.toString(), Toast.LENGTH_LONG).
show();
    //显示异常的 Toast 消息
}
/*捕获异常*/
finally
{
    //添加最后处理代码
}
}
}

```



【代码说明】 本实例通过 SeekBar 实现了音频和音量控制功能。实例首先使用 main.xml 中定义的控件初始化程序界面，采用线性布局生成控件。子元素的排列方式为垂直排列，整个布局的高度和宽度同父元素相同。main.xml 包含 2 个 TextView 控件（volumetextview 和 audiotextview）和 2 个 SeekBar 控件（volumesekbar 和 audioseekbar）。volumetextview 是显示音量拖动条的文本框视图，该控件的宽度同父元素相同，高度随本身的内容进行调整。audiotextview 是显示音频拖动条的文本框视图，该控件的宽度同父元素相同，高度随本身的内容进行调整。volumesekbar 是控制音量改变的拖动条，该控件的宽度同父元素相同，高度随本身的内容调整。audioseekbar 是控制音频改变的拖动条，该控件的宽度同父元素相同，高度随本身的内容进行调整。

然后注册 volumesekbar 的事件监听器，在用户拖动 volumesekbar 时被调用。需要实现监听器的 onProgressChanged、onStartTrackingTouch 和 onStopTrackingTouch 方法。其中，onStartTrackingTouch 在拖动时被调用，onProgressChanged 在拖动的过程中被调用，onStopTrackingTouch 在拖动结束后被调用。当用户拖动拖动条时，onStartTrackingTouch、onProgressChanged 和 onStopTrackingTouch 这 3 个方法依次被调用，即先调用 onStartTracking Touch，然后调用 onProgressChanged，最后调用 onStopTrackingTouch。

接着注册 audioseekbar 的事件监听器，用户拖动 audioseekbar 时被调用。需要实现监听器的 onProgressChanged、onStartTrackingTouch 和 onStopTrackingTouch 方法。

需要注意两点：（1）onProgressChanged 方法的形参同 onStartTrackingTouch 不同，onProgressChanged 方法的形参不仅包含拖动条对象，还包含当前的进度。（2）在用户拖动过程中，onProgressChanged 方法实时计算当前的值。计算当前的音频值或者音量值时，计算公式为： $((\text{progress} \times 100) / \text{maxaudioProgress}) \times \text{MAX\_AUDIO} / 100$ 。若采用“(progress/maxaudioProgress) \* MAX\_) / 10”公式计算当前的值，则返回值总是 0。这是因为系统会将两个整数的运算结果自动转换成整数，而 progress/ maxaudioProgress 的结果是一个小数，所以系统自动将其转换成 0。

启动该 Android 程序，程序显示两个拖动条。拖动控制音量的拖动条，程序根据拖动条的进度计算当前的音量值并显示在文本框视图中，如图 4-25 所示。

拖动控制音频的拖动条，程序根据拖动条的进度计算当前的音频值并显示在文本框视图中，如图 4-26 所示。

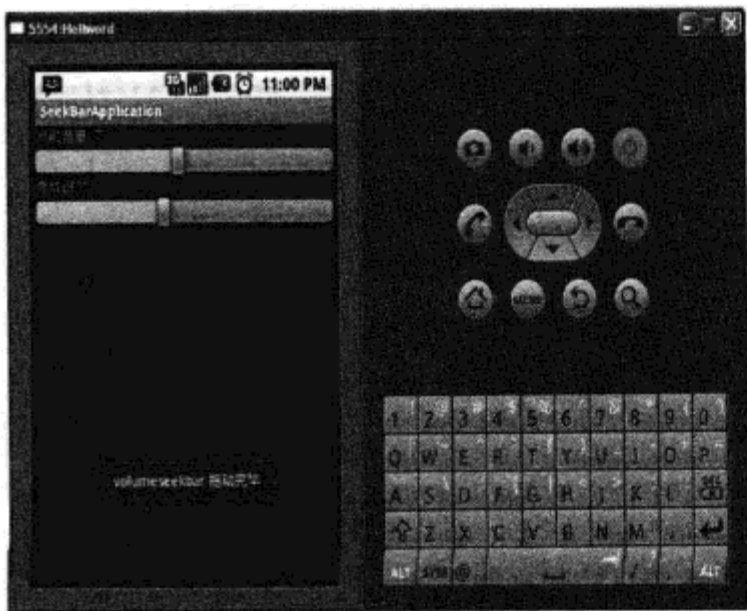


图 4-25 调节音量拖动条



图 4-26 调节音频拖动条



4.2.15 评分条（RatingBar）

RatingBar 是基于 SeekBar 和 ProgressBar 的扩展,用星型来显示等级评定。在使用 RatingBar 的默认大小时,用户可以触摸、拖动或使用方向键来设置评分。当使用可以支持用户交互的 RatingBar 时,无论将小部件放在它的左边还是右边都是不合适的。只有当控件宽度设置为随自身内容变化时,设置的星星数量（通过函数 setNumStars(int)或者在 XML 布局文件中定义）才显示出来。若宽度设为同父元素相同,则评分条组件可能不能正确地被显示出来。注意,评分条组件的次级进度被当做星型部分内部的填充背景,次级进度一般不应该被修改。

RatingBar 和 SeekBar 都是 ProgressBar 和 AbsSeekBar 的子类,故其功能同 SeekBar 和 ProgressBar 类似,但 SeekBar 的变化范围比 RatingBar 更规整。通过 RatingBar,用户只能每次增加或减少半个星的幅度。图 4-27 显示了一个评分条。



图 4-27 评分条（SeekBar）

通过 RatingBar 可实现评分条。RatingBar 是 PorgressBar 和 AbsSeekBar 的子类,RatingBar 在 Android 系统中的层次关系如下:

```

java.lang.Object
android.view.View
android.widget..ProgressBar
android.widget.AbsSeekBar
android.widget.RatingBar

```

在开发中,评分条组件很少被用到。相对于拖动条,RatingBar 本身提供了许多属性和方法,开发人员可通过这些方法和属性创建一个合适的评分条组件。

表 4-24 列举了 RatingBar 类主要的方法。

表 4-24 RatingBar 常用的方法

方法	功能描述	返回值
RatingBar	提供了 3 个构造函数: RatingBar(Context context) RatingBar(Context context, AttributeSet attrs) RatingBar(Context context, AttributeSet attrs, int defStyle)	null
getNumStars()	该方法返回显示在界面上的星星的数目	int
getOnRatingBarChangeListener()	该方法获取该评分条组件的监听器（OnRatingBarChangeListener）	RatingBar.OnRatingBarChangeListener
setIsIndicator(boolean isIndicator)	该方法用于设置该评分条组件是否可被改变。若不可被改变,则该组件是一个指示器。参数 isIndicator 为 true 时,表示该组件不可修改,否则该组件可被修改	void



续表

方法	功能描述	返回值
getRating()	获取当前已经被填充的星星的数量，例如当前评分条组件含有 6 个星星且步长为 0.5，其中有两个半星星被填充，则返回 2.5	float
getStepSize()	获取评分条的步长	float
setMax(int max)	设置进度条的范围	void
onSizeChanged (int w, int h, int oldw, int oldh)	继承的 android.widget.ProgressBar 的方法。该方法在视图的大小发生改变时被调用。如果只是添加到视图，调用时显示的是旧值  0。onSizeChanged 定义了四个形参： w：视图当前宽度 h：视图当前高度 oldw：视图以前的宽度 oldh：视图以前的高度	void
addFocusables(ArrayList<View> views, int direction)	继承的 android.view.View 的方法。该方法为当前 ViewGroup 中的所有子视图添加焦点获取能力	void
addTouchables(ArrayList<View> views)	继承的 android.view.View 的方法。该方法为子视图添加触摸能力	void
setNumStars(int numStars)	设置显示的星星的数量。为了能够正常显示它们，建议将当前小部件的布局宽度设置为“wrap content”。也可通过 android:numStars 属性设置	void
setOnRatingBarChangeListener(RatingBar.OnRatingBarChangeListener listener)	该方法用于注册评分组件改变事件监听器	void
setRating(float rating)	设置分数（星星的数量）。也可通过 android:rating 属性设置	void
setStepSize(float stepSize)	设置当前评分条的步长。也可通过 android:stepSize 属性设置	void

下面以一个实例说明 RatingBar 类的功能，实例 4-15 实现了一个简单的评分条，用户可以通过单击评分条设置分数。

**【实例 4-15】RatingBar 组件的应用。**

主程序 RatingBarApplication.java 实现，该程序使用 main.xml 初始化界面：

```

public class RatingBarApplication extends Activity {
    private RatingBar ratingBarStyleSmall; //声明评分条控件 ratingBarStyleSmall
    private RatingBar ratingBarStyleIndicator; //声明评分条控件 ratingBarStyleIndicator
    private RatingBar ratingBarDefault; //声明评分条控件 ratingBarStyle
    private TextView textviewIndicator; //声明文本框视图控件 textviewIndicator
    private TextView textviewSmall; //声明文本框视图控件 textviewSmall
    private TextView textviewDefault; //声明文本框视图控件 textviewSmall
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
    }
}
    
```



```

super.onCreate(savedInstanceState); //必须在子类的 onCreate 方法中调用父类的 onCreate 方法
setContentView(R.layout.main); //setContentView 加载 main.xml 作为程序布局
textViewIndicator = (TextView) findViewById(R.id.textViewIndicator);
//根据 main.xml 控件定义创建 textViewIndicator
textViewSmall = (TextView) findViewById(R.id.textViewSmall);
//根据 main.xml 控件定义创建 textViewSmall
textViewDefault = (TextView) findViewById(R.id.textViewDefault);
//根据 main.xml 控件定义创建 textViewDefault
ratingBarStyleIndicator = (RatingBar) findViewById(R.id.ratingBarStyleIndicator);
//根据 main.xml 控件定义创建 ratingBarStyleIndicator
ratingBarStyleSmall = (RatingBar) findViewById(R.id.ratingBarStyleSmall);
//根据 main.xml 控件定义创建 ratingBarStyleSmall
ratingBarDefault = (RatingBar) findViewById(R.id.ratingBarDefault);
//根据 main.xml 控件定义创建 ratingBarDefault

ratingBarStyleSmall.setOnRatingBarChangeListener(new OnRatingBarChangeListener()
{
    public void onRatingChanged(RatingBar ratingBar, float rating,
        boolean fromUser) {
        int numStars = ratingBar.getNumStars(); //获取 ratingBarStyleSmall 的星星的总量
        float currating = rating; //获取当前评分, 即已被选择的星星的数量
        textViewIndicator.setText(numStars + "个星" + "已选" + currating);
    }
    /*需要覆盖父类的 onRatingChanged 方法。当 ratingBarStyleSmall 的状态发生改变时, 该
    方法被调用*/
});
/*注册 ratingBarStyleSmall 的状态改变监听器, 需要在该监听器中实现 onRatingChanged 方法*/
ratingBarStyleIndicator.setOnRatingBarChangeListener(new OnRatingBarChangeListener()
{
    public void onRatingChanged(RatingBar ratingBar, float rating,
        boolean fromUser) {
        int numStars = ratingBar.getNumStars(); //获取 ratingBarStyleSmall 的星星的总量
        float currating = rating; //获取当前评分, 即已被选择的星星的数量
        textViewSmall.setText(numStars + "个星" + "已选" + currating);
    }
    /*需要覆盖父类的 onRatingChanged 方法。当 ratingBarStyleIndicator 的状态发生改变时,
    该方法被调用*/
});
/*注册 ratingBarStyleIndicator 的状态改变监听器, 需要在该监听器中实现 onRatingChanged 方
法*/

```



```

ratingBarDefault.setOnRatingBarChangeListener(new OnRatingBarChangeListener()
{
    public void onRatingChanged(RatingBar ratingBar, float rating,
                                boolean fromUser) {
        int numStars = ratingBar.getNumStars(); //获取 ratingBarDefault 的星星的总量
        float currating = rating; //获取当前评分，即已被选择的星星的数量
        textViewDefault.setText(numStars + "个星" + "已选" + currating);
    }
    /*需要覆盖父类 onRatingChanged 方法。当 ratingBarDefault 的状态发生改变时，该方法
    被调用*/
});
/*注册 ratingBarDefault 的状态改变监听器，需要在该监听器中实现 onRatingChanged 方法*/
}
}

```

**【代码说明】** 本实例实现了 3 种类型的评分条。同 SeekBar 和 ProgressBar 一样，android.R.attr 类定义了 Android 使用的评分条类型。表 4-25 列举了 android.R.attr 类定义的评分条类型。

表 4-25 android.R.attr 类定义的评分条类型

名称	类型	整型值
ratingBarStyle	默认评分条	16842876 (0x0101007c)
ratingBarStyleIndicator	指示功能评分条，该评分条不可与用户交互	16843280 (0x01010210)
ratingBarStyleSmall	小评分条，该评分条不可与用户交互	16842877 (0x0101007d)

本实例首先使用 main.xml 中定义的空间初始化程序界面，采用线性布局生成控件，子元素的排列方式为垂直排列，整个布局的高度和宽度同父元素相同。main.xml 包含 3 个 TextView 控件（textViewDefault、textViewSmall 和 textViewIndicator）和 3 个 RatingBar 控件（ratingBarDefault、ratingBarStyleSmall 和 ratingBarStyleIndicator）。其中，textViewDefault 是显示默认进度的文本框视图，该控件的宽度随本身的内容进行调整，高度也随本身的内容进行调整。默认评分条 ratingBarDefault 未指定其自身的 Style，该评分条可与用户进行交互，即用户可通过单击鼠标改变其状态。该控件的宽度随本身的内容进行调整，高度也随本身的内容进行调整。textViewSmall 是显示 ratingBarStyleSmall 进度的文本框视图，该控件的宽度随本身的内容进行调整，高度也随本身的内容进行调整。ratingBarStyleSmall 的宽度随本身的内容进行调整，高度也随本身的内容进行调整。该控件通过 style="?android:attr/ratingBarStyleSmall"指定评分条为小评分条。textViewIndicator 是显示 ratingBarStyleIndicator 进度的文本框视图，其宽度随本身的内容进行调整，高度也随本身的内容进行调整。ratingBarStyleSmall 通过 style="?android:attr/ratingBarStyleIndicator"指定其类型为指示评分条，该控件的宽度随本身的内容进行调整，高度也随本身的内容进行调整。

生成程序中的控件之后，每个评分条都被注册了状态改变的监听器。RatingBar 对应的监听器为 OnRatingBarChangeListener，该监听器是 RatingBar 的私有类。需要使用 android.widget.



RatingBar.OnRatingBarChangeListener 引用该监听器，因此程序开始时使用“import android.widget. RatingBar. OnRatingBarChangeListener;” 语句导入评分条监听器类。注册该监听器的方法为 setOnRatingBarChange Listener(OnRatingBarChangeListener listener), RatingBar 控件需要调用该方法才能完成注册，并且需要实现监听器 OnRatingBarChangeListener 的 onRatingChanged (RatingBar ratingBar, float rating,boolean fromUser) 方法。当 RatingBar 控件状态发生变化时，该方法被调用。

需要注意，虽然本实例为每个 RatingBar 控件注册了监听器，但是 ratingBarStyleSmal 和 ratingBarStyleIndicator 不能与用户进行交互，用户的单击事件不会响应控件。笔者故意在程序中设计其相应的监听器，为的是让用户通过该程序体验不同类型评分条的交互性。

通过 Eclipse 启动该 Android 程序，程序显示 3 个评分条组件和 3 个文本框视图。图 4-28 所示是初始状态下，程序中的文本框视图显示相应的评分条的类型，且每个评分条组件包含六个星星。

单击每个评分组件（ratingBarStyleSmall、ratingBarStyleSmall 和 ratingBar StyleIndicator），图 4-29 显示了单击的运行结果。



图 4-28 程序启动后的界面

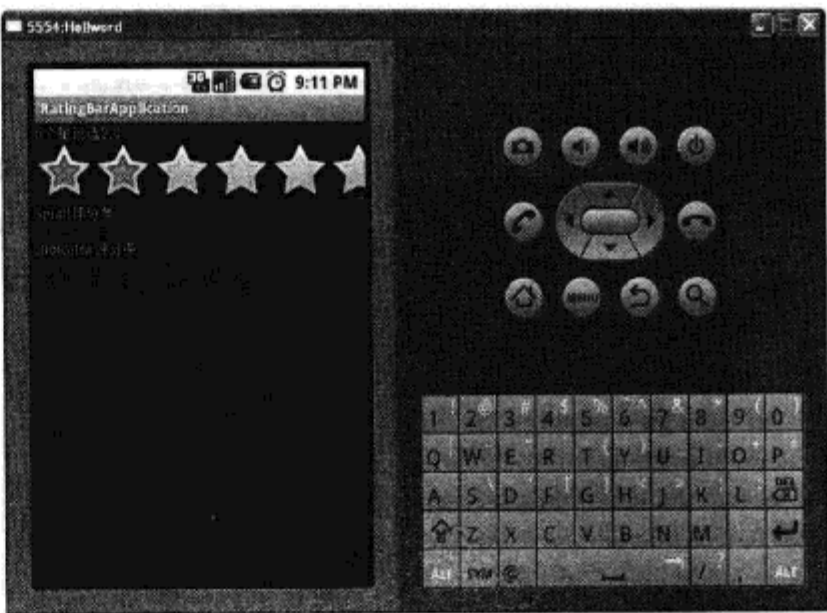


图 4-29 单击评分条组件

如图 4-29 所示，用户单击时，第一个评分条组件（ratingBarDefault）状态发生变化；后面两个评分条组件（ratingBarStyleSmall 和 ratingBarStyleIndicator）的状态则不发生任何变化。





## 第 5 章 Android 中的视图组件

除了常用的组件之外，Android 还提供了视图相关的组件。这些组件提供了视图相关的接口，通过这些接口可实现视图相关的应用。本章将详细介绍这些组件的功能、用法以及相应的接口。

### 5.1 视图组件

#### 5.1.1 图片视图（ImageView）

ImageView 与 TextView 功能基本类似，主要区别是显示的资源不同。ImageView 可显示图像资源，而 TextView 只能显示文本资源。在应用开发中，图片视图是一个常用且重要的组件。ImageView 是 ImageButton 的父类，ImageButton 可显示一个可以被用户单击的图片按钮。ImageView 可通过两种方式设置资源：一种是通过 setImageBitmap 方法设置图片资源；第二种是通过 <ImageView>XML 元素的 android:src 属性或 setImageResource(int)方法指定 ImageView 的图片。图 5-1 显示了一个图片视图。



图 5-1 图片视图（ImageView）

ImageView 类属于 android.Wiget 包并且继承 android.widget.View 类，ImageView 类派生了 ImageButton、Zoom Button 等子类。ImageView 类提供了操作图片视图的方法和属性，开发人员可根据这些方法设计图片视图相关的应用。表 5-1 列举了 ImageView 常用的方法。

ImageView 提供了显示图片的功能，下面以一个具体的例子说明 ImageView 的基本用法。

表 5-1 ImageView 常用的方法

方法	功能描述	返回值
ImageView	提供了 3 个构造函数： <div>                     ImageView(Context context)   ImageView(Context context, AttributeSet attrs)   ImageView(Context context, AttributeSet attrs, int defStyle)                 </div>	null
setAdjustViewBounds	设置是否保持高宽比。需要结合 maxWidth 和 maxHeight 一起使用	Boolean
getDrawable	获取 Drawable 对象。若获取成功，返回 Drawable 对象，否则返回 null	Drawable
getScaleType	获取视图的填充方式	ScaleType



		续表
方法	功能描述	返回值
setScaleType	设置视图的填充方式。Android 提供了包括矩阵、拉伸等七种填充方式	void
SetAlpha	设置图片透明度。透明值范围为 0~255，其中 0 为完全透明，255 为完全不透明	void
setMaxHeight	设置按钮控件的最大高度	void
setMaxWidth	设置按钮控件的最大宽度	void
setImageURI	设置图片地址。图片地址使用 URI 指定	void
setImageResource	设置图片资源库	void
setOnTouchListener	设置 ImageButton 单击事件监听	Boolean
setColorFilter	设置颜色过滤。需要制定颜色过滤矩阵	void

【实例 5-1】ImageView 组件的应用。

ImageViewApplication.java 实现，该类实现了图片变化的功能：

```

public class ImageViewApplication extends Activity {
    /** Called when the activity is first created. */
    private ImageView imageview; //声明 ImageView 对象 imageview
    private TextView textview; //声明 TextView 对象 textview
    private Button helpButton; //帮助按钮
    private Bitmap bitmap; //图片资源 Bitmap

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //使用 main.xml 生成程序 UI
        imageview = (ImageView)findViewById(R.id.imageview); //根据 XML 定义创建 imageview 对象
        textview = (TextView)findViewById(R.id.textview); //根据 XML 定义创建 textview 对象
        textview.setText("按 1 放大 按 2 缩小 \n 按 3 左转 按 4 右转");
        bitmap = BitmapFactory.decodeResource(this.getResources(), R.drawable.motor);
        //使用 BitmapFactory 的 decodeResource 方法获取 R.drawable.motor 的位示图
        imageview.setImageBitmap(bitmap); //设置 imageview 的背景为 R.drawable.motor 指定的图像
    }

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch(keyCode)
        {
            case KeyEvent.KEYCODE_1:
                Toast.makeText(this, "正在放大图片", Toast.LENGTH_SHORT).show();
                break;
                //按 1，图片将被放大

```



```

        case KeyEvent.KEYCODE_2:
            Toast.makeText(this, "正在缩小图片", Toast.LENGTH_SHORT).show();
            break;
            //按 2, 图片将被放大
        case KeyEvent.KEYCODE_3:
            Toast.makeText(this, "正在左转图片", Toast.LENGTH_SHORT).show();
            break;
            //按 3, 图片将被左转
        case KeyEvent.KEYCODE_4:
            Toast.makeText(this, "正在右转图片", Toast.LENGTH_SHORT).show();
            break;
            //按 4, 图片将被右转
        default:
            Toast.makeText(this, "无效按键", Toast.LENGTH_SHORT).show();
            break;
            //按其余键无效
    }
    /*根据用户按键, 判断用户的操作并显示出来*/
    return super.onKeyDown(keyCode, event); //必须返回父类的 onKeyDown 方法
}

/*实现 onKeyDown 接口方法。当释放之前的按键时, 该方法被调用
 * 根据 keyCode 判断用户操作:
 * 按 1, 放大图片
 * 按 2, 缩小图片
 * 按 3, 左转图片
 * 按 4, 右转图片
 * 否则, 输入无效*/

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch(keyCode)
    {
        case KeyEvent.KEYCODE_1:
        {
            float scaleWidth = 2; //设置图片宽度放大的比例
            float scaleHight = 2; //设置图片高度放大的比例
            ZoomoutImageView(scaleWidth, scaleHight); //按 1, 图片将被放大
            break;
        }
        case KeyEvent.KEYCODE_2:

```



```

        {
            float scaleWidth = (float) 0.5; //设置图片宽度缩小的比例
            float scaleHight = (float) 0.5; //设置图片高度缩小的比例
            ZoominImageView(scaleWidth, scaleHight);    //按 2, 图片将被放大
            break;
        }
        case KeyEvent.KEYCODE_3:
        {
            float degree = -10; //设置旋转角度, 负数表示向左转
            RotateLeftImageView(degree); //按 3, 图片将被左转
            break;
        }
        case KeyEvent.KEYCODE_4:
        {
            float degree = 10; //设置旋转角度, 正表示向右转
            RotateRightImageView(degree); //按 4, 图片将被右转
            break;
        }
        default:
            break;
        //按其余键, 什么都不做
    }
    /*根据用户按键, 判断用户的操作并显示出来*/
    return super.onKeyDown(keyCode, event);
}

/*实现 onKeyUp 接口方法。当释放之前的按键时, 该方法被调用
 * 根据 keyCode 判断用户操作:
 * 按 1, 放大图片
 * 按 2, 缩小图片
 * 按 3, 左转图片
 * 按 4, 右转图片
 * 否则, 输入无效*/

private void ZoomoutImageView(float width, float hight)
{
    float scaleWidth = width; //设置图片宽度放大的比例
    float scaleHight = hight; //设置图片高度放大的比例
    int bitmapWidth = bitmap.getWidth(); //使用 getWidth 方法获取 Bitmap 资源的宽
    int bitmapHeight = bitmap.getHeight(); //使用 getHeight 方法获取 Bitmap 资源的高
    try

```



```

{
    Matrix matrix = new Matrix(); //新建 Matirx 对象, Matirx对象用来存储图像相关的数据
    matrix.postScale(scaleWidth, scaleHight); //使用 matrix 记录图像的缩放比例
    Bitmap resizeBmp = Bitmap.createBitmap(bitmap, 0, 0, bitmapWidth, bitmapHeight, matrix,
        true);
    /*使用 createBitmap 重构缩放后的图片 Bitmap 对象, 该方法需要指定原来图像的 Bitmap
    对象、原来图像的宽度和高度以及图像的缩放比例 (缩放比例是由 Matrix 指定的) */
    imageView.setImageBitmap(resizeBmp);
    //使用 setImageBitmap 方法设置 imageView 的资源图片为 resizeBmp
}
/*try 块包含可能产生异常的代码*/
catch (Exception e)
{
    Toast.makeText(this, "放大图片时产生异常"+ e.toString(), Toast.LENGTH_SHORT).show();
    //若产生异常, 则弹出相应的 Toast 消息
}
/*捕获异常*/
finally
{
    //添加最后处理代码
}
}
/*ZoomoutImageView 方法用于放大图片*/

private void ZoominImageView(float width, float hight){
    float scaleWidth = width; //设置图片宽度缩小的比例
    float scaleHight = hight; //设置图片高度缩小的比例
    int bitmapWidth = bitmap.getWidth(); //使用 getWidth 方法获取 Bitmap 资源的宽
    int bitmapHeight = bitmap.getHeight(); //使用 getHeight 方法获取 Bitmap 资源的高
    try
    {
        Matrix matrix = new Matrix(); //新建 Matirx 对象, Matirx对象用来存储图像相关的数据
        matrix.postScale(scaleWidth, scaleHight); //使用 matrix 记录图像的缩放比例
        Bitmap resizeBmp = Bitmap.createBitmap(bitmap, 0, 0, bitmapWidth, bitmapHeight, matrix,
            true);
        /*使用 createBitmap 重构缩放后的图片 Bitmap 对象, 该方法需要指定原来图像的 Bitmap
        对象、原来图像的宽度和高度以及图像的缩放比例 (缩放比例是由 Matrix 指定的) */
        imageView.setImageBitmap(resizeBmp);
        //使用 setImageBitmap 方法设置 imageView 的资源图片为 resizeBmp
    }
}

```



```

/*try 块包含可能产生异常的代码*/
catch (Exception e)
{
    Toast.makeText(this, "放大图片时产生异常"+ e.toString(), Toast.LENGTH_SHORT).show();
    //若产生异常, 则弹出相应的 Toast 消息
}
/*捕获异常*/
finally
{
    //添加最后处理代码
}
}

/*ZoominImageView 方法用于缩小图片*/

private void RotateLeftImageView(float degree)
{
    int bitmapWidth = bitmap.getWidth();//使用 getWidth 方法获取 Bitmap 资源的宽
    int bitmapHeight = bitmap.getHeight();//使用 getHeight 方法获取 Bitmap 资源的高
    try
    {
        Matrix matrix = new Matrix();//新建 Matirx 对象, Matirx 对象用来存储图像相关的数据
        matrix.postScale(1, 1);//由于图片只是旋转, 所以保持原有的高度和宽度
        //degree = currentDegree-10;
        matrix.setRotate(degree);//设置旋转角度, 负数表示向左转
        Bitmap resizeBmp = Bitmap.createBitmap(bitmap, 0, 0, bitmapWidth, bitmapHeight,
        matrix, true);
        /*使用 createBitmap 重构缩放后的图片 Bitmap 对象, 该方法需要指定原来图像的 Bitmap
        对象、原来图像的宽度和高度以及图像的缩放比例 (缩放比例是由 Matrix 指定的) */
        imageView.setImageBitmap(resizeBmp);
        //使用 setImageBitmap 方法设置 imageView 的资源图片为 resizeBmp
    }
    /*try 块包含可能产生异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(this, "放大图片时产生异常"+ e.toString(), Toast.LENGTH_SHORT).show();
        //若产生异常, 则弹出相应的 Toast 消息
    }
    /*捕获异常*/
    finally
    {

```



```

        //添加最后处理代码
    }
}
/*RotateLeftImageView 方法用于左转图片*/

private void RotateRightImageView(float degree){
    int bitmapWidth = bitmap.getWidth();//使用 getWidth 方法获取 Bitmap 资源的宽
    int bitmapHeight = bitmap.getHeight();//使用 getHeight 方法获取 Bitmap 资源的高
    try
    {
        Matrix matrix = new Matrix();//新建 Matirx 对象, Matirx 对象用来存储图像相关的数据
        matrix.postScale(1, 1);//由于图片只是旋转, 所以保持原有的高度和宽度
        //degree = currentDegree+10;
        matrix.setRotate(degree);//设置旋转角度, 正数表示向右转
        Bitmap resizeBmp = Bitmap.createBitmap(bitmap, 0, 0, bitmapWidth, bitmapHeight, matrix,
            true);
        /*使用 createBitmap 重构缩放后的图片 Bitmap 对象, 该方法需要指定原来图像的 Bitmap
        对象、原来图像的宽度和高度以及图像的缩放比例(缩放比例是由 Matrix 指定的)*/
        imageView.setImageBitmap(resizeBmp);//使用 setImageBitmap 方法设置 imageView 的资源
        图片为 resizeBmp
    }
    /*try 块包含可能产生异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(this, "放大图片时产生异常"+ e.toString(), Toast.LENGTH_SHORT).show();
        //若产生异常, 则弹出相应的 Toast 消息
    }
    /*捕获异常*/
    finally
    {
        //添加最后处理代码
    }
}
/*RotateRightImageView 方法用于右转图片*/
}

```

布局 main.xml 实现, 该布局包含一个 TextView、一个 ImageView 子元素。控件排列方式为垂直排列, 整个布局高度同父元素相同, 且宽度同父元素相同。

<!--TextView 控件。该控件高度和宽度随内容变化-->

```

<TextView
    android:id="@+id/textview"

```



```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />
<!--ImageView 控件。该控件的高度和高度为 200px-->
<ImageView
    android:id="@+id/imageview"
    android:layout_x="40dp"
    android:layout_y="80dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</ImageView>
</LinearLayout>

```

**【代码说明】** 本实例通过 `Bitmap` 和 `BitmapFactory` 实现了图片的放大、缩小、左转和右转。`Bitmap` 可被看成由像素点组成的矩阵，这些点可以进行不同的排列和染色以构成图样。`Bitmap` 将图像定义为由点（像素）组成的矩阵，每个点可以由多种色彩表示（包括 2、4、8、16、24 和 32 位色彩）。例如，一幅  $1024 \times 1024$  分辨率的 32 位真彩图片，其所占存储空间为： $1024 \times 1024 \times 4 = 4\text{MB}$ 。位图文件图像效果好，但是非压缩格式的，需要占用较大的存储空间，不利于在网络上传送，JPG 格式则恰好弥补了位图文件这个缺点。当放大位图时，可以看见构成整个图像的无数单个方块。扩大位图尺寸的结果是增多单个像素，从而使线条和形状显得参差不齐。然而，如果从稍远的位置观看它，位图图像的颜色和形状又显得是连续的。由于每一个像素都是单独染色的，所以可以通过以每次一个像素的频率操作选择区域而产生近似相片的逼真效果，诸如加深阴影和加重颜色。缩小位图尺寸也会使原图变形，因为此举是通过减少像素来使整个图像变小的。由于位图图像是以排列的像素集合体形式创建的，所以不能单独操作（如移动）局部位图。有两种位示图的编码方法：`RGB` 和 `CMYK`。`RGB` 用红、绿、蓝三原色的光学强度来表示一种颜色，这是最常见的位图编码方法，可以直接用于屏幕显示。`CMYK` 用青、品红、黄、黑四种颜料含量来表示一种颜色，这种方法也是常用的位图编码方法，可以直接用于彩色印刷。

Android 系统提供了 `android.graphics.Bitmap` 和 `android.graphics.BitmapFactory` 类来实现位示图的应用，下面是位示图的处理过程：

步骤一：获取位示图。

通过 `BitmapFactory` 类获取位示图，获取位示图的方法分为两种：使用资源获取和使用文件获取。本实例使用资源获取的方法：`BitmapFactory.decodeResource(Resource res, int id)`，该方法需要指定 `Resource` 资源对象和图片资源。

还可使用文件获取的方法 `BitmapFactory.decodeFile(String file)`，该方法需要指定图像资源的文件名。例如，若在手机模拟器 SD 卡上 `/data/data/` 包路径下存储一张 PNG 图片（`motor.png`），可通过 `BitmapFactory.decodeFile` 获取手机模拟器 SD 卡上的 `motor.png` 图片。注意，需要使用 DDMS 的 File 浏览器存储 PNG 图片。首先在 DDMS 的文件浏览器中展开 `/data/data/text.Ex_51`，单击右上角的“Push a file onto the device”按钮上传图片，如图 5-2 所示。



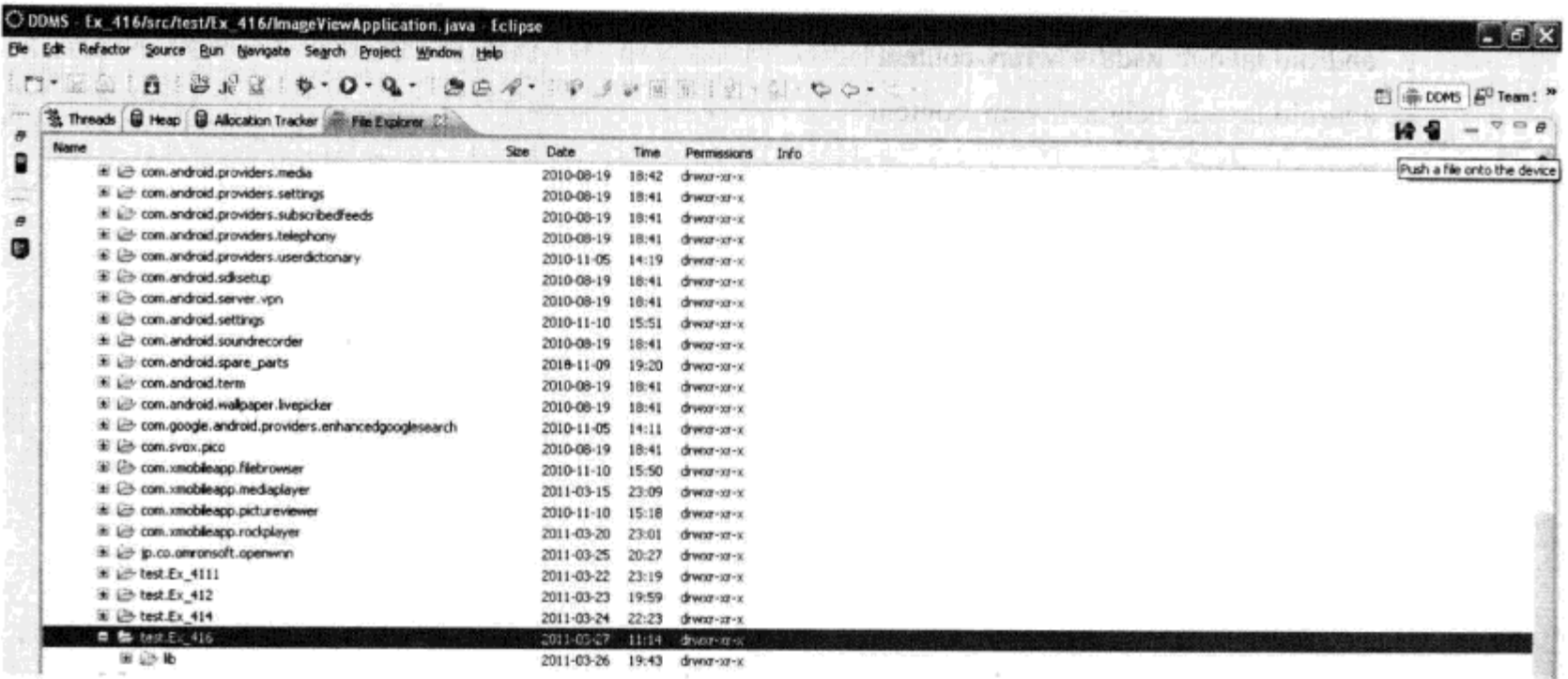


图 5-2 单击上传文件按钮

然后选择 motor.png 文件上传。上传成功后，展开/data/data/text.Ex\_51 文件夹，会在该文件夹下看到 motor.png 文件（如图 5-3 所示）。

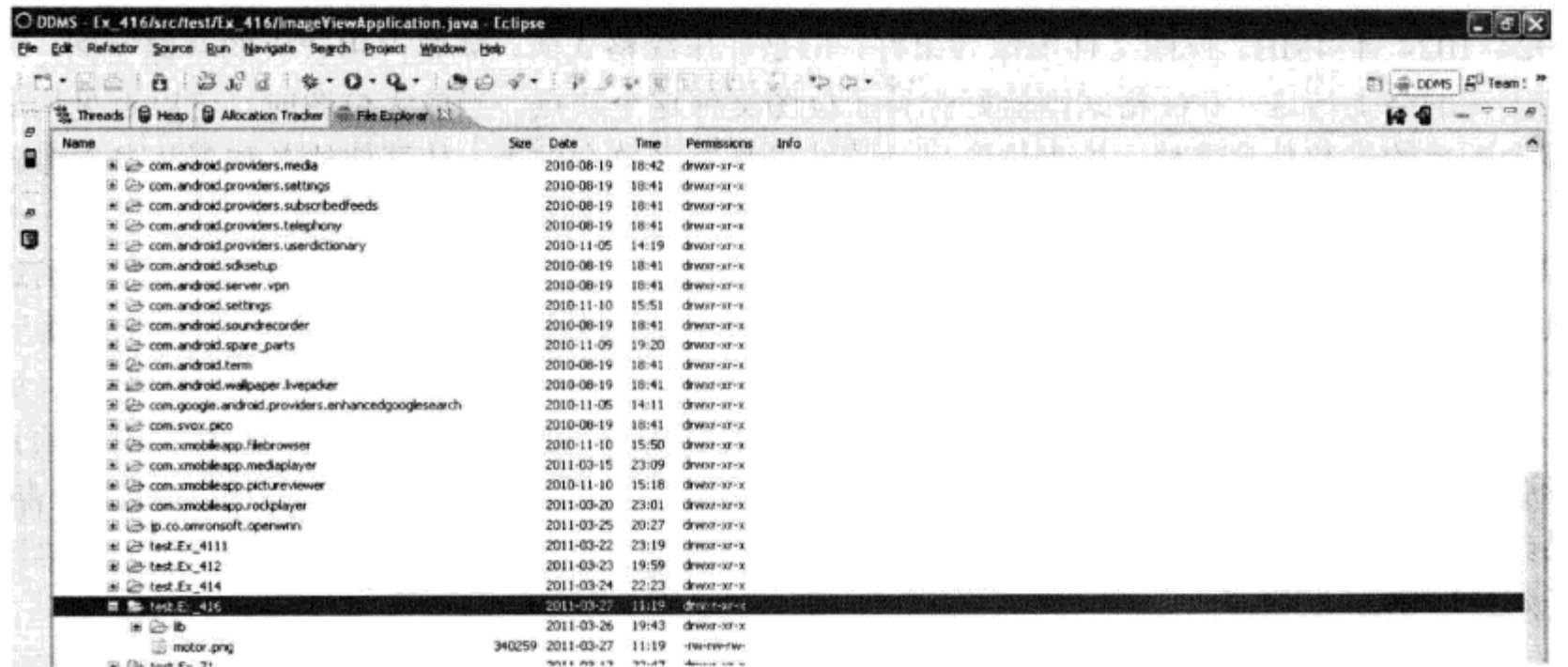


图 5-3 文件上传成功

上传成功后，可使用 BitmapFactory.decodeFile 方法生成该图片的位示图。  
步骤二：获取位示图的信息。  
生成完位示图之后，就可根据生成的位示图获取位示图的信息。Bitmap 提供了获取位示图信息的方法，常见的是获取位示图的高和宽。例如：

```

int bitmapWidth = bitmap.getWidth();//使用 getWidth 方法获取 Bitmap 资源的宽
int bitmapHeight = bitmap.getHeight();//使用 getHeight 方法获取 Bitmap 资源的高

```

除此之外，还可获取透明度、颜色格式等信息。开发人员可以查看 Android SDK 的 Bitmap 文档，该文档详细描述了 Bitmap 的功能、方法和属性。需要注意，Bitmap 使用 Bitmap.Config 定义颜色格式，而 Bitmap.Config 没有包含所有的颜色格式，仅定义了 ALPHA\_8、ARGB\_4444、ARGB\_8888、RGB\_565 等格式。另外，开发人员可使用 Bitmap 的 compress()接口来压缩图片，



但该压缩方法只支持 PNG、JPG 格式的压缩。

步骤三：位示图处理。

获取位示图的信息是为了进一步处理位示图，处理位示图的操作一般包含缩放和旋转。缩放位示图是指改变位示图的大小，可被细化为两种操作：放大和缩小。旋转位示图是指改变位示图的角度，可被细化为两种操作：左转和右转。下面讲述如何实现这些操作。

首先使用语句“`Matrix matrix = new Matrix();`”新建 `Matirx` 对象，这个 `Matirx` 对象用来存储图像相关的信息。

若为缩放操作，则使用 `Matrix` 的 `postScale(float scaleWidth, float scaleHight)` 方法设置缩放比例。其中，参数 `scaleWidth` 是缩放宽度的比例，参数 `scaleHight` 为缩放高度的比例。若缩放比例大于 1 表示该操作为放大操作，反之若小于 1，则该操作为缩小操作。下面是缩小操作的例子：

```
float scaleWidth = width; //设置图片宽度缩小的比例
float scaleHight = hight; //设置图片高度缩小的比例
int bitmapWidth = bitmap.getWidth(); //使用 getWidth 方法获取 Bitmap 资源的宽
int bitmapHeight = bitmap.getHeight(); //使用 getHeight 方法获取 Bitmap 资源的高
Matrix matrix = new Matrix(); //新建 Matirx 对象，Matirx 对象用来存储图像相关的数据
matrix.postScale(scaleWidth, scaleHight); //使用 matrix 记录图像的缩放比例
```

若为旋转操作，则使用 `Matrix` 的 `etRotate(float degree)` 方法设置旋转的角度。其中，参数 `degree` 指定了旋转的角度。如果是旋转，设置为 0，表示不旋转，设置的角度是负数，则向左转，设置的角度是正数，则向右转。下面是旋转操作的例子：

```
Matrix matrix = new Matrix(); //新建 Matirx 对象，Matirx 对象用来存储图像相关的数据
degree = -10;
matrix.setRotate(degree); //设置旋转角度，负数表示向左转
```

步骤四：重新创建位示图。

使用 `Matrix` 存储图像相关操作信息后，使用 `createBitmap` 方法重构位示图。Android 提供了 6 种 `createBitmap` 方法：

```
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)
static Bitmap createBitmap(int width, int height, Bitmap.Config config)
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height)
static Bitmap createBitmap(int[] colors, int offset, int stride, int width, int height, Bitmap.Config config)
static Bitmap createBitmap(Bitmap src)
static Bitmap createBitmap(int[] colors, int width, int height, Bitmap.Config config)
```

本实例使用了 `createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)` 方法重构位示图。其中，参数 `source` 为原来的 `Bitmap` 资源，参数 `x` 为 `Bitmap` 所处的左上角坐标，参数 `y` 为 `Bitmap` 所处的右上角坐标，参数 `width` 为 `Bitmap` 资源宽度，参数 `height` 为 `Bitmap` 资源高度，参数 `m` 为位示图的处理信息，如旋转度或者缩放比例，参数 `filter` 指定资源是否过滤。

上面讲述了位示图的处理过程，除此之外还需要设计驱动位示图的处理操作。本实例可通过实现键盘的监听，根据用户的不同按键驱动位示图处理。键盘操作包含按键和释放按键，这两个操作的监听分别通过 `onKeyDown` 和 `onKeyUp` 实现。`onKeyDown` 方法在用户按键时被调



用，而 `onKeyUp` 方法在用户释放按键时被调用。

在程序运行时按键，`onKeyDown` 方法被调用。`onKeyDown` 方法跟据 `keyCode` 判断用户操作：若按 1，则弹出放大图片的消息；按 2，则弹出缩小图片的消息；按 3，则弹出左转图片消息；按 4，则弹出右转图片消息；按其他键，则弹出输入无效消息。释放按键时，`onKeyUp` 方法被调用。`onKeyUp` 方法跟据 `keyCode` 判断用户操作：若按 1，则放大图片；按 2，则缩小图片；按 3，左转图片；按 4，则右转图片；按其他键，则什么都不做。注意，在程序中使用 `KeyEvent` 类定义的 `keyCode` 判断按键的类型，使用 `KeyEvent` 的 `keyCode` 非常方便，不需要记住按键对应的值就可使用。图 5-4 是一个手机键盘界面，`KeyEvent` 类定义了键盘上所有按键的值。

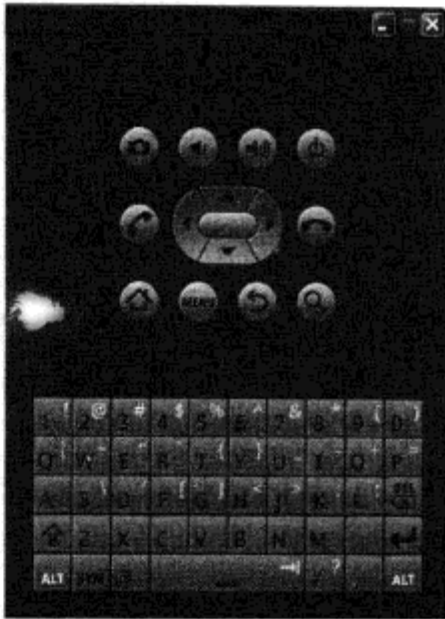


图 5-4 手机键盘

表 5-2 列举了常用的键盘按键的定义。

表 5-2 按键定义

名称	类型	整型值
KEYCODE_0	键盘上的 0 键	7 (0x00000007)
KEYCODE_1	键盘上的 1 键	8 (0x00000008)
KEYCODE_2	键盘上的 2 键	9 (0x00000009)
KEYCODE_3	键盘上的 3 键	10 (0x0000000a)
KEYCODE_4	键盘上的 4 键	11 (0x0000000b)
KEYCODE_5	键盘上的 5 键	12 (0x0000000c)
KEYCODE_6	键盘上的 6 键	13 (0x0000000d)
KEYCODE_7	键盘上的 7 键	14 (0x0000000e)
KEYCODE_8	键盘上的 8 键	14 (0x0000000f)
KEYCODE_9	键盘上的 9 键	16 (0x00000010)
KEYCODE_A	键盘上的 A 键	29 (0x0000001d)
KEYCODE_Z	键盘上的 Z 键	54 (0x00000036)
KEYCODE_ALT_LEFT	键盘上的左 ALT 键	57 (0x00000039)
KEYCODE_ALT_RIGHT	键盘上的右 ALT 键	58 (0x0000003a)
KEYCODE_APOSTROPHE	键盘上的右顿号键	75 (0x0000004b)
KEYCODE_AT	键盘上的右@键	77 (0x0000004d)
KEYCODE_BACK	键盘上的返回键	4 (0x00000004)
KEYCODE_BACKSLASH	键盘上的斜线 ‘\’ 键	73 (0x00000049)
KEYCODE_COMMA	键盘上的逗号 ‘,’ 键	55 (0x00000037)
KEYCODE_ENDCALL	键盘上的挂电话键	6 (0x00000006)
KEYCODE_ENTER	键盘上的回车键	66 (0x00000042)
KEYCODE_ENVELOPE	键盘上的发邮件键	65 (0x00000041)
KEYCODE_EQUALS	键盘上的等号 ‘=’ 键	70 (0x00000046)
KEYCODE_LEFT_BRACKET	键盘上的左括号 ‘(’ 键	71 (0x00000047)
KEYCODE_MENU	键盘上的菜单键	82 (0x00000052)
KEYCODE_MINUS	键盘上的减号 ‘-’ 菜单键	69 (0x00000045)



通过 Eclipse 启动该 Android 程序，程序显示一个摩托车的图片视图和一个提示用户按键的文本框视图（如图 5-5 所示）。

分别按键盘上的 1 键、2 键、3 键、4 键，图片分别被放大、缩小、左转以及右转。图 5-6 显示了图片缩小后的效果。

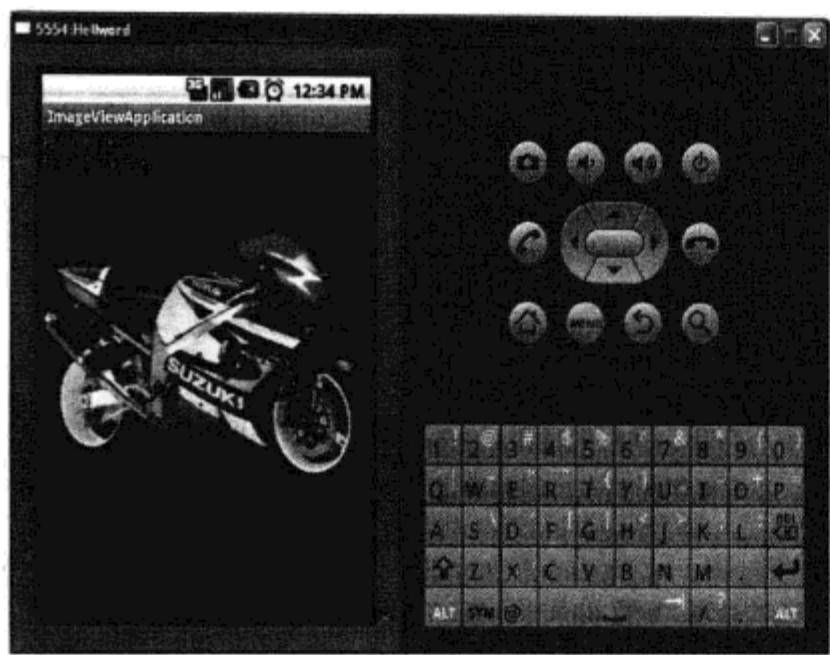


图 5-5 程序启动后的界面



图 5-6 缩小图片

5.1.2 滚动视图（ScrollView）

ScrollView 是 Android 提供的滚动视图类，用于实现滚动效果。ScrollView 可被看成一种容器，这种容器通过滚动的方式来显示内容。用户可通过滑动滑块来实现 ScrollView 界面的滚动，这种功能类似于翻页功能。ScrollView 的子元素可以是一个复杂的对象布局管理器，通常用的子元素是垂直方向的线性布局。注意，ScrollView 只支持垂直方向的滚动，不支持水平方向的移动。图 5-7 是一个滚动视图的例子。

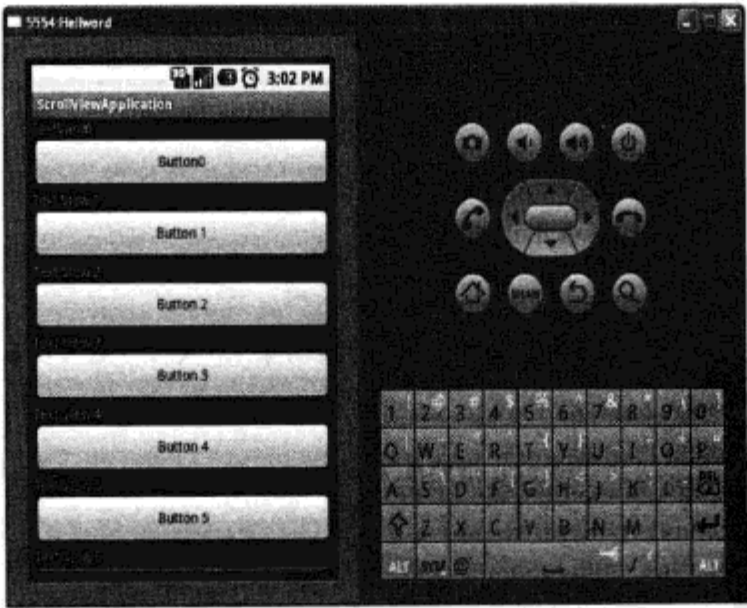


图 5-7 滚动视图（ScrollView）

ScrollView 类属于 andriod.Wiget 包并且继承了 android.widget.FrameLayout 类，而 android.widget.FrameLayout 类又继承了 android.widget.ViewGroup 类。ScrollView 类的层次关系如下：

```

java.lang.Object
android.view.View
android.widget.ViewGroup

```



android.widget.FrameLayout

android.widget.ScrollView

ScrollView 类提供了操作滚动视图的方法和属性，开发人员可根据这些方法实现滚动视图相关应用。表 5-3 列举了 ScrollView 常用的方法。

表 5-3 ScrollView 常用的方法

方法	功能描述	返回值
ScrollView	提供了 3 个构造函数： <div>                         ScrollView(Context context)                         ScrollView(Context context, AttributeSet attrs)                         ScrollView(Context context, AttributeSet attrs, int defStyle)                     </div>	null
dispatchKeyEvent (KeyEvent event)	该方法将参数指定的键盘事件分发给当前焦点路径的视图。分发键盘事件时，按照焦点路径查找合适的视图。若本视图为焦点，则将键盘事件发送给自己；否则发送给焦点视图	Boolean(分发成功返回 true，否则返回 false)
arrowScroll (int direction)	该方法响应单击上下箭头时对滚动条的处理。参数 direction 指定了滚动的方向	Boolean
addView (View child)	该方法用于子视图的添加。参数 view 指定了添加的子视图。 注意，除此方法之外，还有 3 种 addView 方法： <div>                         public void addView (View child, int index)                         public void addView (View c, int index, ViewGroup.LayoutParams params)                         public void addView (View child, ViewGroup.LayoutParams params)                     </div>	void
computeScroll ()	更新子视图的值 (mScrollX 和 mScrollY)，该方法常被父视图调用	void
onTouchEvent (MotionEvent ev)	该方法用于运动事件，这个运动事件是在处理触摸屏幕时产生的	boolean
isSmoothScrollingEnabled ()	按箭头方向滚动时，是否显示滚动的平滑效果	boolean
setMaxWidth	设置按钮控件的最大宽度	void
setImageURI	设置图片地址，图片地址使用 URI 指定	void
setImageResource	设置图片资源库	void
setOnTouchListener	设置 ImageButton 单击事件监听	Boolean
setColorFilter	设置颜色过滤，需要制定颜色过滤矩阵	void
executeKeyEvent (KeyEvent event)	当接收到键盘事件时，此函数执行滚动操作，类似于处理由视图体系发送的事件	Boolean(执行成功返回 true，否则返回 false)
fullScroll (int direction)	将视图滚动到 direction 指定的方向	Boolean(若按照指定方向滚动成功返回 true，否则返回 false)
onInterceptTouchEvent (MotionEvent me)	此方法用于拦截用户的触屏事件	boolean

上面列举了 ScrollView 常用的方法，通过这些方法用户可以实现通过滚动的方式显示内容。下面以一个手机报的例子说明 ScrollView 的基本用法。

【实例 5-2】ScrollView 组件的应用。



ScrollViewApplication.java 实现, 该类实现了手机报的功能:

```
import android.app.Activity; //导入 Activity 类
import android.os.Bundle; //导入 Bundle 类
/*ScrollViewApplication 实现了一个手机应用*/
public class ScrollViewApplication extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
        setContentView(R.layout.main); //使用 main.xml 生成程序布局
    }
}
```

布局 main.xml 实现, 该布局通过 ScrollView 控件生成一个简单的手机报。控件排列方式为默认的垂直排列方式, 整个布局的高度随内容变化且宽度同父元素相同。

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <!--TextView 控件。该控件的高度和宽度随内容变化-->
    <TextView
        android:id="@+id/textview1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30px"
        android:text="          旅游信息          " />
    <!--TextView 控件。该控件的高度和宽度随内容变化-->
    <TextView
        android:id="@+id/textview2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20px"
        android:text="旅游信息导读\n" />
    <!--TextView 控件。该控件的高度和宽度随内容变化-->
    <TextView
        android:id="@+id/textview3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="15px"
        android:text="1.我和春天有个约会 " />
```



<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text=" 2.看美丽宝岛" />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview4"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text=" 3.箭扣长城攻略" />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview6"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text=" 4.杭州西溪湿地 " />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview7"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text=" 5.三月下江南 " />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview8"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text="\n 手机正文 \n1.我和春天有个约会 " />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview8"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="10px"



android:text="\n 婺源是江西最西北部的一个县城，东临浙江省衢州市，北临安徽省黄山市。解放前是属于安徽的，无论是建筑还是民风，都属于典型的徽派文化。由于地处三省交界处的山区，交通非常不方便，导致当地经济条件相对落后。也正由于这个原因，当地的古建筑保存得很完整。" />

<!--ImageView 控件。该控件的高度和宽度随内容变化-->

<ImageView

android:id="@+id/imageview1"

android:layout\_width="300px"

android:layout\_height="300px"

android:src="@drawable/pic1"/>

<!--Text 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview9"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="10px"

android:text="黄山归来不看山，婺源归来不看村。婺源乡村之美，在于浑然天成的和谐。“青山向晚盈轩翠，碧水含春傍槛流”，无论是民居还是村落的设造，无不讲究人与天、地、山、水的融洽关系，或枕山面水，或临溪而居，山山水水皆成人“家”。"/>

<!--ImageView 控件。该控件的高度和宽度为 300px-->

<ImageView

android:id="@+id/imageview2"

android:layout\_width="300px"

android:layout\_height="300px"

android:src="@drawable/pic2"/>

<!--ImageView 控件。该控件的高度和宽度为 300px-->

<ImageView

android:id="@+id/imageview3"

android:layout\_width="300px"

android:layout\_height="300px"

android:src="@drawable/pic3"/>

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView

android:id="@+id/textview10"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="15px"

android:text="\n\n2.看美丽宝岛" />

<!--TextView 控件。该控件的高度和宽度随内容变化-->

<TextView



```

        android:id="@+id/textview11"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10px"
        android:text="\n 阿里山是台湾著名的风景区之一，有“不到阿里山，不知阿里山之美，不知
        阿里山之富，更不知阿里山之伟大”的说法。由于山区气候温和，盛夏时依然清爽宜人，
        加上林木葱翠，阿里山也是理想的避暑胜地。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
        android:id="@+id/imageview4"
        android:layout_width="300px"
        android:layout_height="300px"
        android:src="@drawable/pic4"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
        android:id="@+id/textview12"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10px"
        android:text="\n 由于山区气候温和，盛夏时依然清爽宜人，加上林木葱翠，阿里山也是理想
        的避暑胜地。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
        android:id="@+id/imageview5"
        android:layout_width="300px"
        android:layout_height="300px"
        android:src="@drawable/pic5"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
        android:id="@+id/textview13"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="15px"
        android:text="\n\n3.箭扣长城攻略\n" />
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
        android:id="@+id/textview14"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10px"

```



```

        android:text="箭扣长城是一段“野长城”。北京境内的长城有 629 公里，其中 600 公里属自然状态中的“野长城” " />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview6"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic6"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview15"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="那残破、古旧、朴拙的砖石绵延于崇山峻岭之中，荒凉不事雕琢的自然美更能令人感受长城金戈铁马的千年不屈风骨。" />
<!--ImageView 控件。该控件的高度和高度为 300px-->
<ImageView
    android:id="@+id/imageview7"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic7"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview16"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="厌倦了游人摩肩接踵，过度开发的成熟景点，“野长城”越来越显示出其独特的魅力。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview8"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic8"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview17"
    android:layout_width="wrap_content"

```



```

        android:layout_height="wrap_content"
        android:textSize="15px"
        android:text="\n\n4.杭州西溪湿地\n" />
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview18"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="来西溪湿地要看野鸭。野鸭最佳的观赏点在鸭子湾，行走在烟水渔庄岸边，或
    是泛舟于此，总是可以看到野鸭在湖面上悠闲地游着，有时三五成群，有时则两两做伴。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview9"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic9"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview19"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="西溪的景色特点是天然质朴，无论是动物还是植物，都向世人展示了最本真的
    面目。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview10"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic10"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview20"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="在这里每个人遇到的总是不同的小鸟、野鸭、小鱼，都会有一段属于自己的美
    好经历。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->

```



```

<ImageView
    android:id="@+id/imageview11"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic11"/>
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview21"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="15px"
    android:text="\n\n5.三月下江南 \n" />
<!--TextView 控件。该控件的高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview22"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="10px"
    android:text="里水桥相连，房屋依水而建，有水的地方就有人家，流水把整个小镇一分为二。
    临河的屋子，除了一小部分做了店铺，大多依旧是住在小镇的居民。" />
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview12"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic12"/>
<!--ImageView 控件。该控件的高度和宽度为 300px-->
<ImageView
    android:id="@+id/imageview13"
    android:layout_width="300px"
    android:layout_height="300px"
    android:src="@drawable/pic13"/>
</LinearLayout>

```

**【代码说明】** 本实例的主程序（ScrollViewApplication.java）比较简单，只有几行代码。ScrollViewApplication.java 只完成了最基本的功能，包括导入 Activity 类、Bundle 类，并且在自己的 onCreate 方法中调用父类的 onCreate 方法初始化环境，最后使用 main.xml 生成程序布局。

以前讲过 Activity 的界面设计可以使用两种不同的方式来实现：一种是采用基于 XML 的方式来获取界面组件从而实现界面设计，另一种是直接使用代码来创建界面组件从而实现界面设计。虽然这两种方式在功能上是等价的，但这两种方式在程序的 UI 设计的应用场合是不一样的。一般来讲，对于程序运行之后改变较少的控件，应该采用 XML 实现。而对于程序运行之



后需要动态变化的控件，应该使用代码实现。由于手机报只是用来显示指定的内容，程序运行后控件不发生变化，因此笔者主要采用 XML 来实现。

从组织层次上，main.xml 最高层只有一个元素：ScrollView 控件。该控件的排列方式为默认的垂直排列方式，整个布局的高度随内容变化且宽度同父元素相同。该元素包含了若干个 TextView 和 ImageView 控件。ImageView 控件主要用于显示手机报中的图片，而 TextView 控件用于显示手机报中的文字信息。

ScrollView 元素可以包含若干子元素，只需在该元素中放入需要滚动显示的子元素，当子元素过多而超过手机屏幕显示的限制时，ScrollView 元素就提供了滚动查看的功能。手机用户只需滑动滑块，就能滚动查看 ScrollView 元素中的子元素。

通过 Eclipse 启动该 Android 程序，程序显示一个关于旅游信息的手机报，如图 5-8 所示。

为了实现文字显示的层次感，对于不同类型的文字设置不同的大小。设置文字大小是通过 android 标签的属性 android:textSize 进行的。对于总标题，通过 android:textSize 设置其大小为 30px。而对于旅游信息导读和子信息标题则设置文字大小分别为 20px 和 15px。而对于普通文字则设置文字大小为 10px。

图 5-8 显示了用户当前只能查看标题 1 的内容，可通过滚动界面来查看其余的内容。图 5-9 显示了通过滚动界面显示的其他内容。

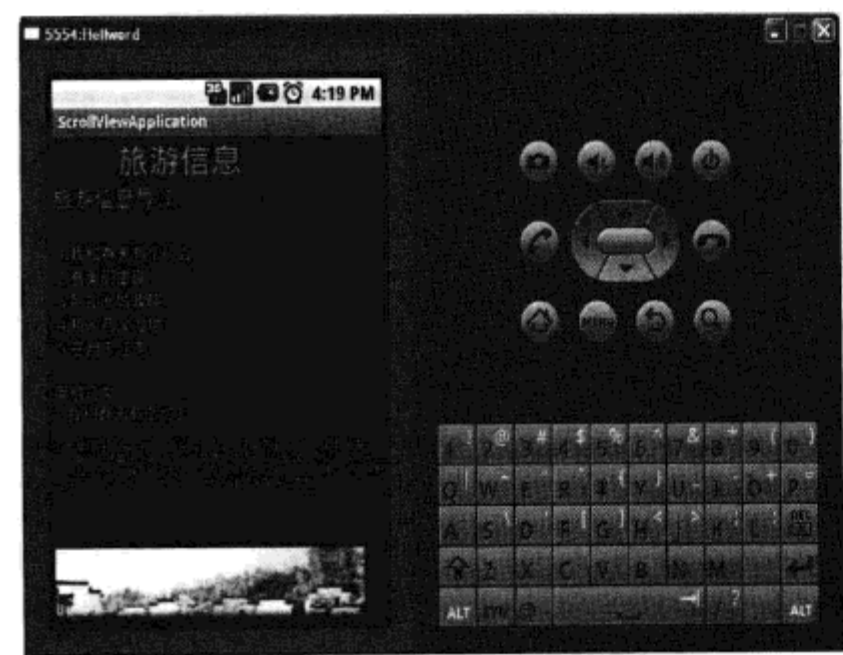


图 5-8 手机报显示界面一

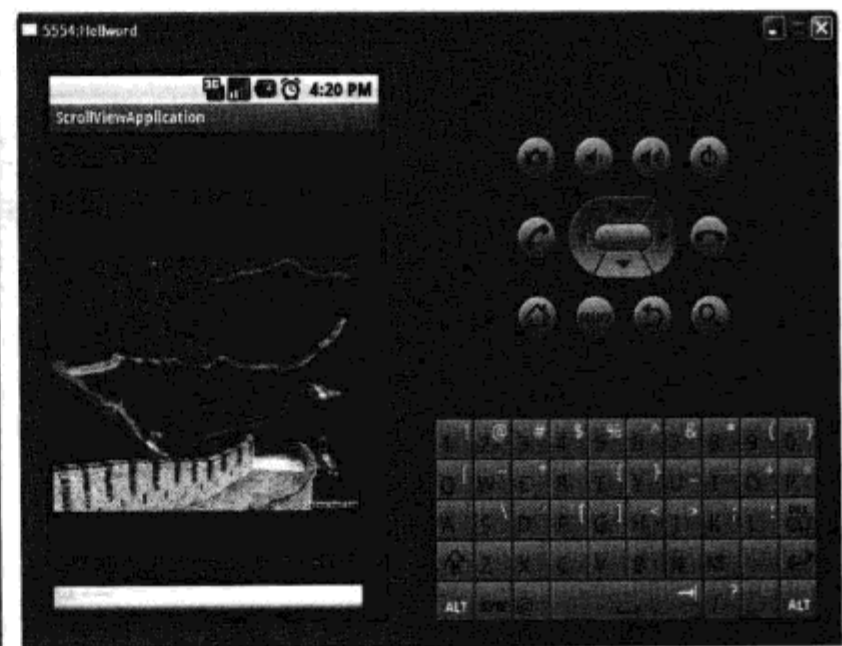


图 5-9 手机报显示界面二

### 5.1.3 网格视图（GridView）

网格视图（GridView）也是一种特殊的视图组织方式，网格视图将其子元素组织成网格状（如图 5-10 所示）。一个网格视图通常需要一个列表适配器（ListAdapter），这个适配器包含网格视图的子元素组件。网格视图的视图排列方式与矩阵类似，当屏幕上有很多元素（文字、图片或其他元素）需要显示时，可以使用网格视图。既然有多个元素要显示，就需要使用 ListAdapter 来存储这些元素。用户可能会选择其中一个元素进行操作，这就需要设置事件监听 setOnItemClickListener 来捕捉和处理事件。



图 5-10 网格视图（GridView）

网格视图能够以数据网格形式显示子元素，并能够对这些子元素进行分页、自定义样式等操作。GridView 类是



andriod.Wiget 包中的一个应用，该类继承了 AdapterView 类。

GridView 类提供了操作网格视图的方法和属性，开发人员可根据这些方法实现网格视图相关应用（如手机桌面、九宫图）。表 5-4 列举了 GridView 常用的方法。

表 5-4 GridView 常用的方法

方法	功能描述	返回值
GridView	提供了 3 个构造函数：  <div> <div>GridView(Context context)</div> <div>GridView(Context context, AttributeSet attrs)</div> <div>GridView(Context context, AttributeSet attrs, int defStyle)</div> </div>	null
setGravity(int gravity)	设置此组件中的内容在组件中的位置	void
setColumnWidth(int)	设置网格视图的宽度	void
getAdapter()	获取该视图的适配器 Adapter	ListAdapter
setAdapter(ListAdapter adapter)	根据参数指定的适配器，设置网格视图对应的适配器	void
setStretchMode(int)	该方法用于设置缩放模式，也可通过 android:stretchMode 设置，有多个缩放模式：NO_STRETCH、STRETCH_SPACING、STRETCH_SPACING_UNIFOR 或 STRETCH_COLUMN_WIDTH	void
onKeyMultiple(int keyCode,int repeatCount, KeyEvent event)	多次按键时的处理方法。当连续发生多次按键时，该方法被调用。其中，keyCode 为按键对应的整型值，repeatCount 是按键的次数，event 是按键事件	Boolean(若继续处理事件，返回 true，否则返回 false)
setSelection(int p)	设置当前被选中的网格视图的子元素	void
onKeyUp(int k, KeyEvent e)	释放按键时的处理方法。释放按键时，该方法被调用。其中，k 为按键对应的整型值，e 是按键事件	boolean
onKeyDown(int keyCode, KeyEvent event)	按键时的处理方法。按键时，该方法被调用。其中，keyCode 为按键对应的整型值，event 是按键事件。注意，用户按键的过程中，onKeyDown 先被调用，然后用户释放按键后，调用 onKeyUp	boolean
setHorizontalSpacing(int c)	当网格视图有多列时，该方法设置网格视图同一行子元素之间的水平间距	void
setNumColumns(int)	设置网格视图包含的子元素的列数	void
getHorizontalSpacing()	获取网格视图同一行子元素之间的水平间距	int
getNumColumns(int)	获取网格视图包含的子元素的列数	int
getSelection()	获取当前被选中的网格视图的子元素	int

上面列举了 GridView 的常用方法，通过这些方法用户可以实现网格视图。下面通过一个例子说明 GridView 的基本用法。

【实例 5-3】GridView 组件的应用。

GridViewApplication.java 实现，该类实现了网格视图的功能：

```

public class GridViewApplication extends Activity {
    /** Called when the activity is first created. */
    private List<MyGrid> gridlist; //声明 MyGrid 对象列表
    private SelfAdapter gridadapter; //声明 SelfAdapter 对象，该适配器类是本程序定义的适配器

```



```
private GridView mygridview;    //声明网格视图对象 mygridview
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) { //重载 onCreate 方法
    super.onCreate(savedInstanceState);    //调用父类的 onCreate 方法
    setContentView(R.layout.main);        //使用 main.xml 生成程序布局
    mygridview = (GridView) findViewById(R.id.gridview);
    //根据 XML 定义的 gridview 生成 Grid View 对象
    try
    {
        gridlist = new ArrayList<MyGrid>();    //使用数组列表创建 MyGrid 的对象列表
        gridadapter = new SelfAdapter(this);    //创建自定义的 SelfAdapter 对象
        gridlist.add(new MyGrid("Grid_1"));
        gridlist.add(new MyGrid("Grid_2"));
        gridlist.add(new MyGrid("Grid_3"));
        gridlist.add(new MyGrid("Grid_4"));
        gridlist.add(new MyGrid("Grid_5"));
        gridlist.add(new MyGrid("Grid_6"));
        gridlist.add(new MyGrid("Grid_7"));
        gridlist.add(new MyGrid("Grid_8"));
        gridlist.add(new MyGrid("Grid_9"));
        gridlist.add(new MyGrid("Grid_10"));
        gridlist.add(new MyGrid("Grid_11"));
        gridlist.add(new MyGrid("Grid_12"));
        gridlist.add(new MyGrid("Grid_13"));
        gridlist.add(new MyGrid("Grid_13"));
        /*使用 ArrayList 的 add 方法添加 MyGrid 对象*/
        gridadapter.setList(gridlist);        //自定义的适配器设置关联的列表
        mygridview.setAdapter(gridadapter);    //通过 setAdapter 建立同自定义适配器对象的关联
    }
    /*try 包含可能出现异常的代码*/
    catch(Exception e)
    {
        Toast.makeText(this, e.toString(), Toast.LENGTH_SHORT).show();
    }
    /*catch 块捕获异常*/
    finally
    {
        //TODO
    }
}
```



```

    }
    /*finally 块为最后的处理代码*/
}

public class MyGrid {
    private String objname;        //MyGrid 类只有一个属性，这个属性为对象的名字

    /*MyGrid 类定义了 3 个方法：MyGrid 、getObjName 和 setObjName */
    public MyGrid(String objname) {
        super();
        this.objname = objname;    //通过关键字 this 来区分类属性 objname 和 MyGrid 形参
        objname
    }
    /*MyGrid(String name)是自定义的构造函数*/

    public String getObjName() {
        return objname;
    }
    /*getObjName 方法用来获取 objname*/

    public void setObjName(String objname) {
        this.objname = objname;    //通过关键字 this 来区分类属性 objname 和 setObjName 形参
        objname
    }
    /*setObjName 方法用来获取 objname*/
}

public class SelfAdapter extends BaseAdapter {

    private Context context;        //声明 Context 对象 context
    private List<MyGrid> gridlist;    //声明 List 对象 gridlist
    private LayoutInflater layoutinflater; //声明 LayoutInflater 对象 layoutinflater
    private class Grider {
        ImageView imageview; //声明类的属性 imageview
        TextView textview;    //声明类的属性 textview
    }
    /*SelfAdapter 的私有类 Grider，该类只有两个属性，没有方法。这个类的功能类似于一个特殊
    的数据结构*/

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

```



```

        this.context = context;    //根据构造函数指定的 Context 对象构造 SelfAdapter 对象
    }
    /*SelfAdapter 的构造函数，调用构造函数需要指定一个上下文对象*/

    public void setList(List<MyGrid> gridlist) {
        this.gridlist = gridlist;    //通过关键字 this 来区分类属性 gridlist 和形参 gridlist
        layoutinflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_
            SERVICE);
        //getService 使用 Context 提供的 LAYOUT_INFLATER_SERVICE 属性构造 LayoutInflater 对象
    }
    /*setList 方法设置本适配器对象的 MyGrid 列表*/

    public int getCount() {
        // TODO
        return gridlist.size(); //返回列表的元素个数
    }
    /*返回本适配器的列表的元素个数*/

    @Override
    public long getItemId(int index) {
        return index;
    }
    /*getItem 方法获取索引标识，事实上这个方法返回指定的索引*/

    @Override
    public Object getItem(int index) {
        return gridlist.get(index);    //返回列表中第 index 个元素
    }
    /*getItem 方法根据索引，获取 gridlist 对应的元素，即 MyGrid 对象*/

    @Override
    public View getView(int index, View view, ViewGroup parent) {
        Grider gridholder;    //声明 Grider 对象 gridholder

        if (view != null)
        {
            gridholder = (Grider) view.getTag(); //获取标记
        }
        else
    
```



```

    {
        gridholder = new Grider();    //创建 Grider 对象 gridholder
        view = layoutInflater.inflate(R.layout.grid, null); //获取 R.layout.grid 对应的 View 对象
        gridholder.imageView = (ImageView)view.findViewById(R.id.XMLimageView);
        //根据 grid.xml 定义的文本框视图组件生成 gridholder 的 imageView 对象
        gridholder.textView = (TextView)view.findViewById(R.id.XMLtextView);
        //根据 grid.xml 定义的文本框视图组件生成 gridholder 的 textView 对象
        view.setTag(gridholder); //设置标记
    }
    /*若 view 为空, 则设置标记, 否则使用 getTag 方法获取其标记*/
    MyGrid grid = gridlist.get(index);    //根据索引获取 MyGrid 对象
    if (grid != null)
    {
        gridholder.textView.setText(grid.getObjName()); //设置其对应的文本
    }
    return view;    //返回 view 对象
}
}
}

```

main.xml 实现, 该布局包含一个 GridView 控件, 该控件的高度和宽度随内容变化, 缩放模式为 columnWidth。

```

<?xml version="1.0" encoding="utf-8"?>
<!--GridView 控件。该控件的高度和宽度随内容变化, 缩放模式为 columnWidth-->
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:horizontalSpacing="10dp"
    android:columnWidth="90dp"
    android:stretchMode="columnWidth"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:id="@+id/gridview"
    android:gravity="center" />
<?xml version="1.0" encoding="utf-8"?>
<!--程序采用相对布局。该布局包含一个 TextView、一个 ImageView 子元素。整个布局高度同和宽度同
父元素相同-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content"
    android:paddingBottom="8dip"

```



```

        android:layout_width="wrap_content">
        <!--ImageView 控件。该控件的高度和宽度随内容变化-->
        <ImageView
            android:layout_height="wrap_content"
            android:id="@+id/XMLimageview"
            android:layout_width="wrap_content"
            android:layout_centerHorizontal="true"
            android:src="@drawable/icon">
        </ImageView>
        <!--TextView 控件。该控件的高度和宽度随内容变化-->
        <TextView
            android:layout_width="wrap_content"
            android:layout_below="@+id/itemImage"
            android:layout_height="wrap_content"
            android:layout_centerHorizontal="true"
            android:id="@+id/XMLtextview">
        </TextView>
    </RelativeLayout>

```

【代码说明】 本实例首先导入基本适配器类（`android.widget.BaseAdapter`）和网格视图类（`android.widget.GridView`）等类，适配器类和网格视图类是实现网格视图功能的核心类。基本适配器类是一个公共基类适配器，该类是基于列表视图和下拉列表共同实现的。`GridView` `Application` 类自定义了两个子类：`MyGrid` 类和 `SelfAdapter` 类。

`MyGrid` 是自定义的一个网格子元素类，本实例的网格视图用于对这些子元素进行管理。`MyGrid` 类只有一个属性（`objname`），这个属性为对象的名字。`MyGrid` 类定义了 3 个方法：`MyGrid`、`getObjName` 和 `setObjName`。其中，`MyGrid` 是自定义的构造函数，`getObjName` 方法用来获取 `objnam`，`setObjName` 方法用来设置 `objname`。注意，`MyGrid` 类的属性（`objname`）同 `MyGrid` 的构造函数和 `etObjName` 的形参一样。通过关键字 `this` 来区分类属性 `objname` 和 `MyGrid` 的形参 `objname`。若不加区分，程序会出现编译错误。

`SelfAdapter` 是自定义的适配器类，该类是 `BaseAdapter` 的子类。该类包含 4 个成员：`Context` 对象 `context`、`List` 对象 `gridlist`、`LayoutInflater` 对象 `layoutinflater` 和 `Grider`。`Grider` 是 `SelfAdapter` 的私有类，该类只有两个属性，没有任何方法，这个类的功能类似于一个特殊的数据结构。`SelfAdapter` 定义了 `SelfAdapter(Context context)`、`setList(List<MyGrid>gridlist)`、`getCount()`、`getItemId` 和 `getView(int index, View view, ViewGroup parent)` 方法。其中，`SelfAdapter` 方法是构造函数，调用构造函数需要指定一个上下文对象。方法 `setList` 用于设置本适配器对象的 `MyGrid` 列表，方法 `getCount` 使用 `gridlist.size()` 返回本适配器的列表的元素个数；`getItemId` 根据索引使用 `gridlist.get(index)` 获取 `gridlist` 对应的元素，即 `MyGrid` 对象。`getView` 方法用于生成网格视图中的子元素 `MyGrid`，而这个子元素使用 `R.layout.grid` 作为程序布局。

本实例定义了两个布局：`main.xml` 和 `gird.xml`。`main.xml` 布局是程序的总布局，其包含一个 `GridView` 控件，该控件的高度和宽度随内容变化，缩放模式为 `columnWidth`。`gird.xml` 是网格视图的子元素所采用的布局，`SelfAdapter` 类使用这个布局生成 `MyGrid` 子元素。`gird.xml`



采用相对布局，该布局包含一个 TextView、一个 ImageView 子元素，整个布局的高度和宽度同父元素相同。

本实例使用 ArrayList 构造了一个 MyGrid 的列表，ArrayList 是一个特殊的容器。除了 ArrayList 之外，Andorid 还提供了其他类型的容器，包括 Collection、List、Vector、Stack、Hashtable、Map、Set 等。

同 LinkedList 一样，ArrayList 也是线程不安全的容器。ArrayList 是一个大小可变化的数组，与其他容器不同，这个容器允许空元素存在。ArrayList 支持基本的容器操作，如读取、添加、删除以及更新。每个 ArrayList 实例都有一个容量 (Capacity)，即用于存储元素的数组的大小。这个容量的增长算法并没有定义，其容量随着新元素不断添加而自动增加。ArrayList 的插入效率较低，其复杂度同容量成正比，即为  $O(n)$ 。当需要插入大量元素时，在插入前可以调用 ensureCapacity 方法来增加 ArrayList 的容量以提高插入效率。因此，本实例可进行如下改动以提高插入效率：

```
gridlist = new ArrayList<MyGrid>(); //使用数组列表创建 MyGrid 的对象列表
gridadapter = new SelfAdapter(this); //创建自定义的 SelfAdapter 对象
ensureCapacity(13); //增加 ArrayList 的容量以提高插入效率
gridlist.add(new MyGrid("Grid_1"));
gridlist.add(new MyGrid("Grid_2"));
gridlist.add(new MyGrid("Grid_3"));
gridlist.add(new MyGrid("Grid_4"));
gridlist.add(new MyGrid("Grid_5"));
gridlist.add(new MyGrid("Grid_6"));
gridlist.add(new MyGrid("Grid_7"));
gridlist.add(new MyGrid("Grid_8"));
gridlist.add(new MyGrid("Grid_9"));
gridlist.add(new MyGrid("Grid_10"));
gridlist.add(new MyGrid("Grid_11"));
gridlist.add(new MyGrid("Grid_12"));
gridlist.add(new MyGrid("Grid_13"));
gridlist.add(new MyGrid("Grid_13"));
/*插入网格子元素*/
```

通过 Eclipse 启动该 Android 程序，程序显示一个简单的网格视图，如图 5-11 所示。

#### 5.1.4 列表视图 (ListView)

不同于网格视图 (GridView)，列表视图 (ListView) 将元素按照条目的方式自上而下列出。通常，每一列只有一个元素，如图 5-11 所示。图 5-12 是 Android 手机的设置界面，该界面以列表的方式组织子元素。

列表视图将子元素以列表的方式组织，用户可通过滑动滑块来显示界面之外的元素。ListView 类是 andriod.Wiget 包中的一个应用，该类继承了 AdapterView 类。ListVeiw 类的层次关系如下：



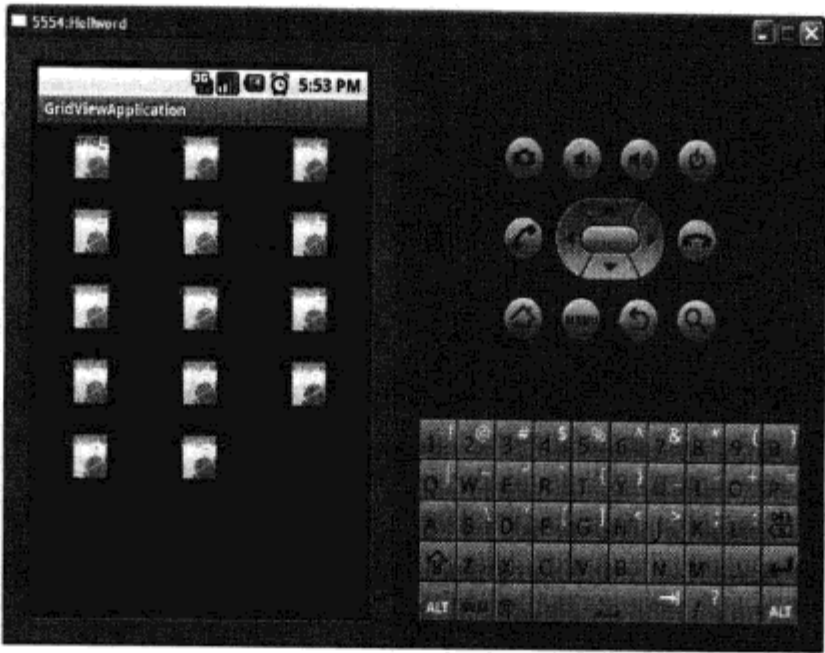


图 5-11 网格视图实例运行界面

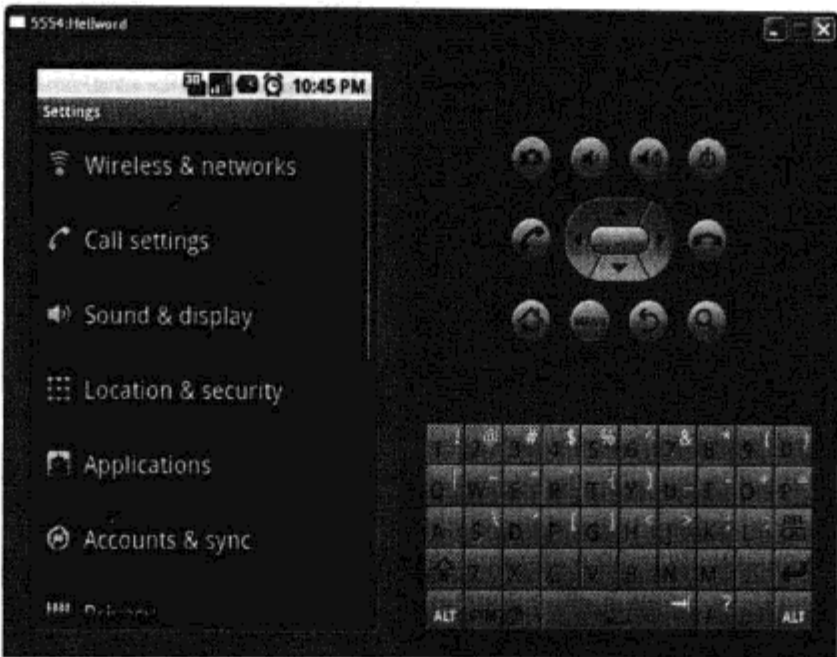


图 5-12 列表视图（ListView）

android.widget.ViewGroup  
 android.widget.AdapterView  
 android.widget.ListVeiw

ListView 类提供了操作列表视图的方法和属性，开发人员可根据这些方法实现列表视图相关应用（如手机设置界面）。表 5-5 列举了 ListView 常用的方法。

表 5-5 GridView 常用的方法

方法	功能描述	返回值
ListView	提供了 3 个构造函数： ListView(Context context) ListView(Context context, AttributeSet attrs) ListView(Context context, AttributeSet attrs, int defStyle)	null
getCheckedItemPosition()	返回当前被选中的子元素的位置	int
addFooterView(View view)	该方法给视图添加脚注，通常脚注位于列表视图的底部。其中，参数 view 指定了要添加脚注的视图	void



续表

方法	功能描述	返回值
getMaxScrollAmount()	该方法获取列表视图的最大滚动数量	int
getDividerHeight()	该方法获取子元素之间分隔符的宽度	void
setStretchMode(int)	该方法用于设置缩放模式，也可通过 android:stretchMode 设置，有多个缩放模式：NO_STRETCH、STRETCH_SPACING、STRETCH_SPACING_UNIFOR 或 STRETCH_COLUMN_WIDTH	void
onKeyMultiple(int keyCode,int repeatCount, KeyEvent event)	多次按键时的处理方法。当连续发生多次按键时，该方法被调用。其中，keyCode 为按键对应的整型值，repeatCount 是按键的次数，event 是按键事件	Boolean(若继续处理事件，返回 true，否则返回 false)
setSelection(int p)	设置当前被选中的列表视图的子元素	void
onKeyUp(int keyCode, KeyEvent event)	释放按键时的处理方法。释放按键时，该方法被调用。其中，keyCode 为按键对应的整型值，event 是按键事件	Boolean
onKeyDown(int k, KeyEvent e)	按键时的处理方法。按键时，该方法被调用。其中，k 为按键对应的整型值，e 是按键事件。注意，用户按键的过程中，onKeyDown 先被调用，然后用户释放按键后，调用 onKeyUp	Boolean
isChecked (int position)	该方法判断指定位置 position 的元素是否被选中	Boolean（若被选中，返回 true，否则返回 false）
addHeaderView(View view)	该方法给视图添加头注，通常头注位于列表视图的顶部。其中，参数 view 指定了要添加头注的视图	void
getChoiceMode()	返回当前的选择模式	int

上面列举了 ListView 的常用方法，通过这些方法用户可以实现列表视图。下面通过一个例子说明 ListView 的基本用法。

**【实例 5-4】**ListView 组件的应用。

ListViewApplication.java 实现，该类实现了列表视图的功能：

```

public class ListViewApplication extends Activity {
    /** Called when the activity is first created. */
    private List<MyList> mylist;    //声明 MyList 对象列表
    private SelfAdapter gridadapter; //声明 SelfAdapter 对象，该适配器类是本程序定义的适配器
    private ListView mylistview;    //声明列表视图对象 mylistview
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) { //重载 onCreate 方法
        super.onCreate(savedInstanceState);    //调用父类的 onCreate 方法
        setContentView(R.layout.main);        //使用 main.xml 生成程序布局
        mylistview = (ListView) findViewById(R.id.listview);
        //根据 XML 定义的 mylistview 生成 ListView 对象
        try
    
```



```

{
    mylist = new ArrayList<MyList>();    //使用数组列表创建 MyGrid 的对象列表
    gridadapter = new SelfAdapter(this); //创建自定义的 SelfAdapter 对象
    mylist.add(new MyList("Grid_1"));
    mylist.add(new MyList("Grid_2"));
    mylist.add(new MyList("Grid_3"));
    mylist.add(new MyList("Grid_4"));
    mylist.add(new MyList("Grid_5"));
    mylist.add(new MyList("Grid_6"));
    mylist.add(new MyList("Grid_7"));
    mylist.add(new MyList("Grid_8"));
    mylist.add(new MyList("Grid_9"));
    mylist.add(new MyList("Grid_10"));
    mylist.add(new MyList("Grid_11"));
    mylist.add(new MyList("Grid_12"));
    mylist.add(new MyList("Grid_13"));
    mylist.add(new MyList("Grid_13"));
    /*使用 ArrayList 的 add 方法添加 MyGrid 对象*/
    gridadapter.setList(mylist);        //自定义的适配器设置关联的列表
    mylistview.setAdapter(gridadapter); //通过 setAdapter 建立同自定义适配器对象的关联
}

/*try 包含可能出现异常的代码*/
catch(Exception e)
{
    Toast.makeText(this, e.toString(), Toast.LENGTH_SHORT).show();
}
/*catch 块捕获异常*/
finally
{
    //TODO
}
/*finally 块为最后的处理代码*/
}

public class MyList {
    private String objname;    //MyList 类只有一个属性，这个属性为对象的名字

    /*MyList 类定义了 3 个方法： MyList、getObjName 和 setObjName */
    public MyList(String objname) {

```



```

        super();
        this.objname = objname;    //通过关键字 this 来区分类属性 objname 和 MyList 形参
        objname
    }
    /*MyList (String name)是自定义的构造函数*/

    public String getObjName() {
        return objname;
    }
    /*getObjName 方法用来获取 objname*/

    public void setObjName(String objname) {
        this.objname = objname;    //通过关键字 this 来区分类属性 objname 和 setObjName 形参
        objname
    }
    /*setObjName 方法用来获取 objname*/
}

public class SelfAdapter extends BaseAdapter {

    private Context context;           //声明 Context 对象 context
    private List<MyList> gridlist;     //声明 List 对象 gridlist
    private LayoutInflater layoutinflater; //声明 LayoutInflater 对象 layoutinflater
    private class Grider {
        ImageView imageview; //声明类的属性 imageview
        TextView textview;   //声明类的属性 textview
    }
    /*SelfAdapter 的私有类 Grider, 该类只有两个属性, 没有方法。这个类的功能类似于一个特殊
    的数据结构*/

    public SelfAdapter(Context context) {
        super();
        this.context = context;    //根据构造函数指定的 Context 对象构造 SelfAdapter 对象
    }
    /*SelfAdapter 的构造函数, 调用构造函数需要指定一个上下文对象*/

    public void setList(List<MyList> gridlist) {
        this.gridlist = gridlist;    //通过关键字 this 来区分类属性 gridlist 和形参 gridlist
        layoutinflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_INFLATER_
        SERVICE);
    }
}

```



```

        //getService 使用 Context 提供的 LAYOUT_INFLATER_SERVICE 属性构造 Layout
        Inflater 对象
    }
    /*setList 方法设置本适配器对象的 MyList 列表*/

    public int getCount() {
        // TODO
        return gridlist.size(); //返回列表的元素个数
    }
    /*返回本适配器的列表的元素个数*/

    @Override
    public long getItemId(int index) {
        return index;
    }
    /*getItem 方法获取索引标识, 事实上这个方法返回指定的索引*/

    @Override
    public Object getItem(int index) {
        return gridlist.get(index); //返回列表中第 index 个元素
    }
    /*getItem 方法根据索引, 获取 gridlist 对应的元素, 即 MyGrid 对象*/

    @Override
    public View getView(int index, View view, ViewGroup parent) {
        Grider gridholder; //声明 Grider 对象 gridholder

        if (view != null)
        {
            gridholder = (Grider) view.getTag(); //获取标记
        }
        else
        {
            gridholder = new Grider(); //创建 Grider 对象 gridholder
            view = layoutinflater.inflate(R.layout.grid, null); //获取 R.layout.grid 对应的 View 对象
            gridholder.imageview = (ImageView)view.findViewById(R.id.XMLimageview);
            //根据 grid.xml 定义的文本框视图组件生成 gridholder 的 imageview 对象
            gridholder.textview = (TextView)view.findViewById(R.id.XMLtextview);
            //根据 grid.xml 定义的文本框视图组件生成 gridholder 的 textview 对象
        }
    }

```







这种适配器是一种针对数据库的 SimpleAdapter，该适配器可以以列表的形式显示数据库的内容。

### (3) 简单适配器 (SimpleAdapter)

该适配器具有扩充性强、灵活度高的优点，通过该适配器可定义各种效果。

### (4) 基本适配器 (BaseAdapter)

该适配器提供了适配器的基本操作。

SelfAdapter 类包含 4 个成员：Context 对象 context、List 对象 gridlist、LayoutInflater 对象 layoutinflater 和 Grider。Grider 是 SelfAdapter 的私有类，该类只有两个属性，没有任何方法，这个类的功能类似于一个特殊的数据结构。SelfAdapter 定义了 SelfAdapter(Context context)、setList(List<MyList> gridlist)、getCount()、getItemId 和 getView(int index, View view, ViewGroup parent)方法。其中，SelfAdapter 方法是构造函数，调用构造函数需要指定一个上下文对象。方法 setList 用于设置本适配器对象的 MyList 列表，方法 getCount 使用 gridlist.size()返回本适配器的列表的元素个数；getItemId 根据索引使用 gridlist.get(index)获取 gridlist 对应的元素即 MyList 对象。getView 方法用于生成网格视图中的子元素 MyList，而这个子元素使用 R.layout.grid 作为程序布局。

本实例定义了两个布局：main.xml 和 gird.xml。main.xml 布局是程序的总布局，其包含一个 GridView 控件，该控件高度和宽度随内容进行变化，缩放模式为 columnWidth。gird.xml 是网格视图的子元素所采用的布局，SelfAdapter 类使用这个布局生成 MyGrid 子元素。gird.xml 采用相对布局。该布局包含一个 TextView、一个 ImageView 子元素。整个布局高度同父元素相同且宽度同父元素相同。

通过 Eclipse 启动该 Android 程序，程序显示一个简单的列表视图，如图 5-13 所示。

可通过滑动滑块，来显示其余的元素，如图 5-14 所示。

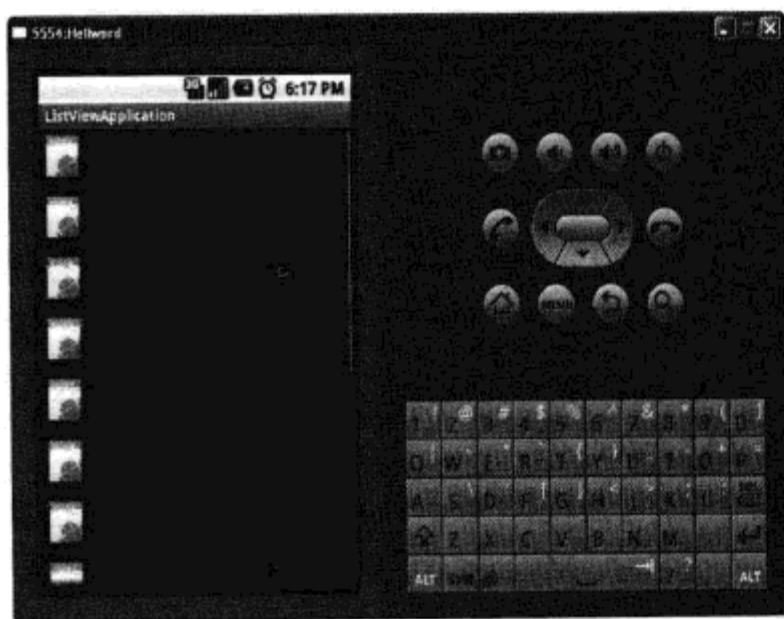


图 5-13 列表视图实例运行结果



图 5-14 滑动列表视图

## 5.1.5 切换图片 (ImageSwitcher 和 Gallery)

ImageSwitcher 类和 Gallery 类可以说是一对“兄弟”，ImageSwitcher 通常用于设定图片的切换动画，而 Gallery 是存储图片的一个特殊容器。本节将讲述 ImageSwitcher 类和 Gallery 类的用法，图 5-15 是一个实例。

ImageSwitcher 类是 android.widget.ViewAnimator 类的子类，其层次关系如下所示：



```

java.lang.Object
android.view.View
android.widget.ViewGroup
android.widget.FrameLayout
android.widget.ViewAnimator
android.widget.ImageSwitcher
    
```

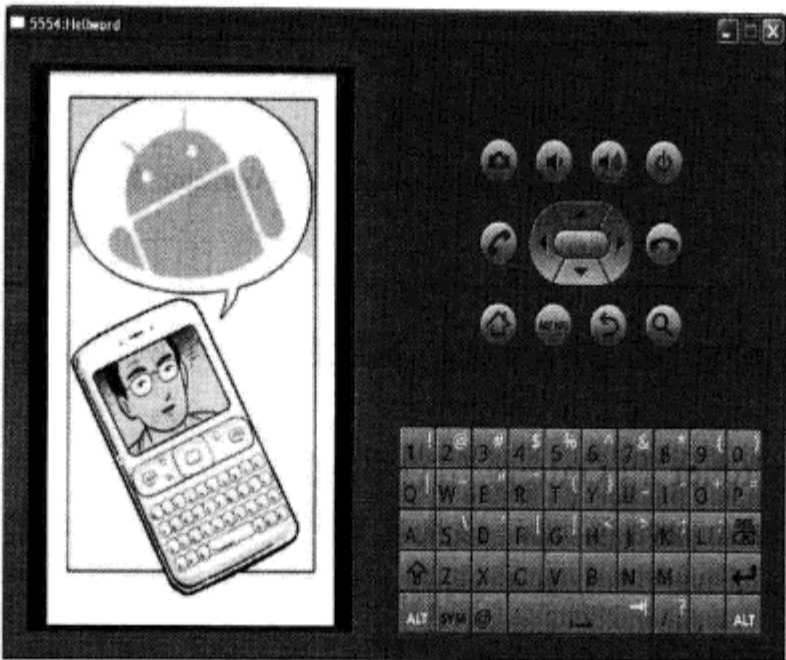


图 5-15 应用实例

ImageSwitcher 类提供了显示图片以及图片切换的动画，开发人员可以利用 ImageSwitcher 类实现图片切换的相关应用（如动感影集）。表 5-6 列举了 ImageSwitcher 常用的方法。

表 5-6 ImageSwitcher 常用的方法

方法	功能描述	返回值
ImageSwitcher	提供了 2 个构造函数：  ImageSwitcher(Context context)  ImageSwitcher(Context context, AttributeSet attrs)	null
setImageDrawable(Drawable drawable)	设置图片为参数 drawable 指定的资源	void
setImageResource(int resid)	设置图片为参数 resid 指定的资源。resid 是编译器为资源生成的唯一标识，例如 R.id.pic1	void
setInAnimation(Context context, int id)	设置图片进入屏幕时，所显示的动画	void
setOutAnimation(Context context, int id)	设置图片退出屏幕时，所显示的动画	void
setImageURI(Uri uri)	设置图片为参数 uri（例如一个图片的 web 链接）指定的资源	void
addView(View child, int index, ViewGroup.LayoutParams params)	继承的 android.widget.ViewSwitcher 的方法。该方法用于添加视图。其中，child 是要添加的视图，params 是视图的布局	void
getNextView()	继承的 android.widget.ViewSwitcher 的方法。该方法用于获取视图	View
getCurrentView()	继承的 android.widget.ViewAnimator 的方法。该方法用于获取视图	View
getDisplayedChild()	该方法返回当前显示的子视图的索引	int
removeAllViews()	继承的 android.widget.ViewAnimator 的方法。调用此方法可以在视图中移除所有的子视图	void



方法	功能描述	续表 返回值
removeViewAt(int index)	继承的 android.widget.ViewAnimator 的方法。该方法用于将 index 指定的视图移除。可使用此方法删除一张显示的图片	void
setAnimateFirstView(boolean animate)	继承的 android.widget.ViewAnimator 的方法。设置视图是否在首次显示时播放动画	void
removeViews(int start, int count)	继承的 android.widget.ViewAnimator 的方法。该方法移除从 start 开始的 count 个子视图	void
showNext()	继承的 android.widget.ViewAnimator 的方法。该方法用于显示下一个视图	void

上面列举了 ImageSwitcher 的常用方法，下面通过一个例子说明 ImageSwitcher 和 Gallery 的用法。

【实例 5-5】ImageSwitcher 和 Gallery 组件的应用。

ImageSwitcherApplication.java 实现，该类实现了图片变化的功能：

```

import android.widget.Gallery;           //导入 Gallery 类
import android.widget.ImageSwitcher;    //导入图片切换器类
import android.widget.ImageView;         //导入图片视图类
import android.widget.Toast;             //导入 Toast 类
import android.app.Activity;             //导入活动类
import android.content.Context;          //导入 Context 类
import android.os.Bundle;               //导入 Bundle 类
import android.widget.AdapterView.OnItemClickListener; //导入 AdapterView 的子元素选择监听器类
import android.view.View;                //导入视图类
import android.view.ViewGroup;           //导入视图组类
import android.widget.Gallery.LayoutParams; //导入布局参数类
import android.widget.ViewSwitcher.ViewFactory; //导入视图工厂类
import android.view.Window;              //导入视图窗口类
import android.view.animation.AnimationUtils; //导入动画应用类
import android.widget.AdapterView;          //导入适配器视图类
import android.widget.BaseAdapter;        //导入基本适配器类

/*ImageSwitcherApplication 继承 Activity 类、android.widget.ViewSwitcher.ViewFactory 接口
和 android.widget.AdapterView.OnItemClickListener 接口*/
public class ImageSwitcherApplication extends Activity implements
    android.widget.ViewSwitcher.ViewFactory, android.widget.AdapterView.OnItemClickListener{
    private Integer[] drawableResource = { R.drawable.pic1, R.drawable.pic2, R.drawable.pic3, R.drawable.
        pic4,
        R.drawable.pic5,R.drawable.pic6, R.drawable.pic7,  R.drawable.pic8,  R.drawable.pic9,

```



```

        R.drawable.pic10,R.drawable.pic11, R.drawable.pic12, R.drawable.pic13, R.drawable.pic14,
        R.drawable.pic15, R.drawable.pic16, R.drawable.pic17, R.drawable.pic18, R.drawable.
        pic19,};

/*drawableResource 数组包含了程序中的图片资源*/
private ImageSwitcher imageSwitcher; //声明图片切换器对象 imageSwitcher
private Gallery gallery; //声明 Gallery 对象 gallery
private int InAnimation = android.R.anim.fade_in; //定义视图进入屏幕的动画
private int OutAnimation = android.R.anim.fade_out; //定义视图退出屏幕的动画
private SelfAdapter adapter; //声明自定义的适配器对象
private int FILLPARENT = LayoutParams.FILL_PARENT;
//定义填充方式变量, 该变量表示为与父元素相同
private int WRAPCONTENT = LayoutParams.WRAP_CONTENT;
//定义填充方式变量, 该变量表示为随内容变化
private int backgroundcolor = 0xFF000000; //定义背景颜色
private android.widget.FrameLayout.LayoutParams params; //声明 LayoutParams 对象

@Override
protected void onCreate(Bundle savedInstanceState) {
    /** Called when the activity is first created. */
    super.onCreate(savedInstanceState); //调用父类的 onCreate 方法
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    //启用窗体的扩展特性, 设置窗体没有标题
    setContentView(R.layout.imageswitcherpage); //加载程序布局
    try
    {
        gallery = (Gallery) findViewById(R.id.gallery); //根据 XML 定义生成 Gallery 对象
        imageSwitcher = (ImageSwitcher) findViewById(R.id.iamgeswitcher);
        //根据 XML 定义生成 ImageSwitcher 对象
        adapter=new SelfAdapter(this);
        imageSwitcher.setFactory(this); //设置用来生成将在视图转换器中切换的两个视图的工厂
        imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,InAnimation));
        //设置视图进入屏幕的动画
        imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,OutAnimation));
        //设置视图退出屏幕的动画
        gallery.setAdapter(adapter); //通过 setAdapter 设置适配器
        gallery.setOnItemClickListener(this); //注册 gallery 的子元素被选中的监听器
    }
    /*try 包含可能出现异常的代码*/
    catch(Exception e)
    {
        Toast.makeText(this, e.toString(), Toast.LENGTH_SHORT).show();
    }
}

```



```

    }
    /*catch 块捕获异常*/
    finally
    {
        //TODO
    }
    /*finally 块为最后的处理代码*/
}

@Override
public View makeView() {
    ImageView imageview = new ImageView(this); //创建 ImageView 对象
    params = new ImageSwitcher.LayoutParams(FILLPARENT, FILLPARENT);
    imageview.setBackgroundColor(backgroundcolor); //设置背景颜色
    imageview.setScaleType(ImageView.ScaleType.FIT_CENTER); //设置 ImageView 的大小
    imageview.setLayoutParams(params); //根据 LayoutParams 对象设置 imageview 的布局
    return imageview; //返回设置后的 imageview
}
/*makeView 方法用于生成视图*/

public class SelfAdapter extends BaseAdapter {

    private Context context; //SelfAdapter 类的属性 context
    public SelfAdapter(Context context) {
        this.context = context; //使用 context 构造 SelfAdapter 对象
    }
    /*SelfAdapter 的构造函数*/

    public View getView(int position, View view, ViewGroup parent) {
        ImageView imageview = new ImageView(context); //根据 context 创建 ImageView 对象
        LayoutParams params = new Gallery.LayoutParams(WRAPCONTENT, WRAPCONTENT);
        //定义 Gallery 的布局参数, 其高度和宽度随内容调整
        imageview.setImageResource(drawableResource[position]);
        //设置当前视图为 drawable Resource[position]对应的图片
        imageview.setAdjustViewBounds(true); //允许调整视图边界
        imageview.setLayoutParams(params); //根据 params 设置 imageview 的布局参数
        imageview.setBackgroundResource(R.drawable.e); //设置背景资源
        return imageview; //返回视图
    }
}

```



```

public int getCount() {
    return drawableResource.length; //返回 drawableResource 数组长度
}

/*getCount 返回图片资源数组的个数，即图片数*/

public Object getItem(int position) {
    return drawableResource[position]; //返回图片对象
}

/*getItem 根据索引返回其对应的对象*/

public long getItemId(int position) {
    return position; //返回位置
}

/*getItemId 根据索引返回其位置，事实上该函数没做任何处理*/
}

@Override
public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
    //imageSwitcher.setTag( "tag");
    imageSwitcher.setImageResource(drawableResource[position]); //设置 imageSwitcher 的显示图片
}

/*当选择图片时，该方法被调用*/

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

}

```

【代码说明】本实例通过 ImageSwitcher 和 Gallery 实现了图片浏览器的功能。ImageSwitcher 类继承了 Activity 类、android.widget.ViewSwitcher.ViewFactory 接口和 android.widget.AdapterView.OnItemClickListener 接口。在该类中，定义了如下方法：

```
protected void onCreate(Bundle savedInstanceState)
```

该方法是程序指定的入口，首先调用父类的 onCreate 方法，之后使用 requestWindowFeature 方法启用窗体的扩展特性，设置窗体没有标题。requestWindowFeature 方法是 android.view.Window 类提供的方法，该方法用于设定串口的特性。除了 Window.FEATURE\_NO\_TITLE 之外，android.view.Window 类还定义了其他的属性，表 5-7 列举了 android.view.Window 提供的属性。



表 5-7    android.view.Window 提供的属性

名称	功能描述	整数值
FEATURE_INDETERMINATE_PROGRESS	内部终止进度	5 (0x00000005)
FEATURE_LEFT_ICON	窗口左边显示图标	3 (0x00000003)
FEATURE_NO_TITLE	窗口没有标题	1 (0x00000001)
FEATURE_OPTIONS_PANEL	启用“选项面板”功能，默认已启用	0 (0x00000000)
FEATURE_PROGRESS	进度指示器功能标志	2 (0x00000002)
FEATURE_RIGHT_ICON	窗口右边显示图标	4 (0x00000004)

然后使用 main.xml 加载程序布局，布局定义如下：

```

<?xml version="1.0" encoding="utf-8"?>
<!--采用相对布局作为程序布局。整个布局的高度和宽度同父元素相同-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!--TextView 布局设置。布局的高度和宽度随内容变化-->
    <TextView
        android:id="@+id/textview"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <!--ImageSwitcher 布局设置。布局高度和宽度同父元素相同-->
    <ImageSwitcher
        android:id="@+id/iamgeswitcher"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
    />
    <!--Gallery 布局设置。布局高度为 59dp，宽度同父元素相同-->
    <Gallery android:id="@+id/gallery"
        android:background="#660000FF"
        android:layout_width="fill_parent"
        android:layout_height="59dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:gravity="center_vertical"
        android:spacing="17dp"
    />
</RelativeLayout>
    
```



生成完布局后, 根据 XML 定义生成 Gallery 对象和 ImageSwitcher 对象。使用 SelfAdapter 创建自定义的适配器对象, 本实例自动创建的适配器类继承了基本适配器类 BaseAdapter。在 SelfAdapter 中, 构造函数需根据 context 创建 ImageView 对象, 并定义 Gallery 的布局参数, 其高度和宽度随内容调整。然后设置当前视图为 drawableResource[position]对应的图片并且允许调整视图边界。接着设置用来生成将在视图转换器中切换的两个视图的工厂、视图进入屏幕的动画和视图退出屏幕的动画。

最后使用 Gallery 的 setOnItemSelectedListener 方法注册 gallery 的子元素被选中的监听器。需要在监听器中实现以下两个方法:

```
public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
public void onNothingSelected(AdapterView<?> parent)
```

当图片被选中时, 使用 ImageSwitcher.setImageResource 设置 ImageSwitcher 的显示图片。

通过 Eclipse 启动该 Android 程序, 程序显示一个图片浏览器, 如图 5-16 所示。

可以通过单击下面的图片, 来显示该图片。单击图片时, OnItemSelectedListener 监听到选中图片的事件, 然后监听器执行 onItemSelected 方法, 该方法将获取被选中图片的位置, 并使用 setImageResource 设置当前图片视图显示的图片。程序运行结果如图 5-17 和图 5-18 所示。

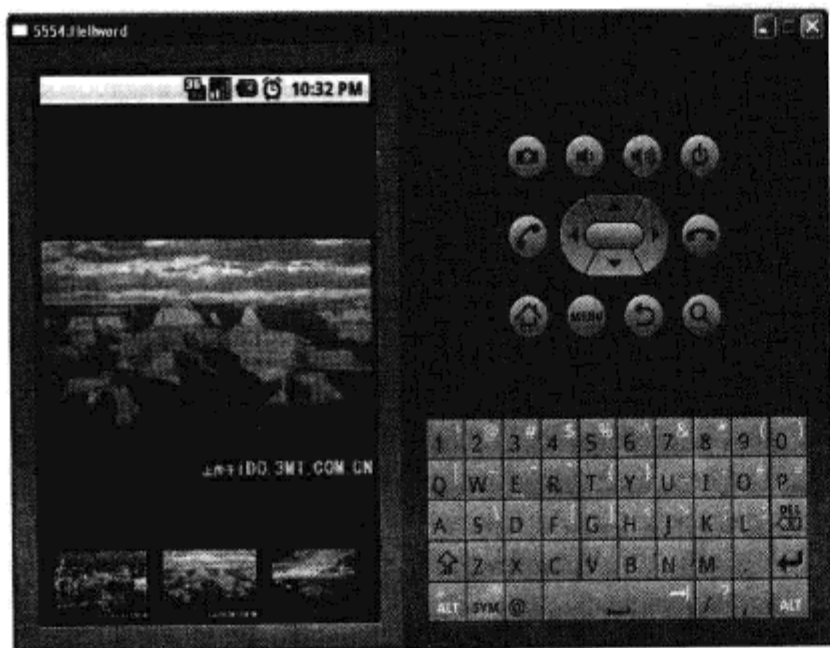


图 5-16 程序启动后的界面



图 5-17 程序运行结果一

### 5.1.6 标签切换 (Tab)

普通视图大多只能上下移动, 可使用标签 (Tab) 实现视图的左右移动。标签 (Tab) 是一种常用的视图显示方式, 图 5-19 显示了一个标签的应用。

Tab 是 android.widget.FrameLayout 的子类, 其层次关系如下所示:

```
java.lang.Object
android.view.View
android.widget.ViewGroup
android.widget.FrameLayout
```

下面通过一个例子说明 Tab 的用法。

【实例 5-6】Tab 组件的应用。

TabApplication.java 实现, 该类使用 Tab 实现标签切换的功能:





图 5-18 程序运行结果二



图 5-19 应用实例

```
import android.widget.LinearLayout; //导入 LinearLayout 类
import android.widget.RadioButton; //导入 RadioButton 类
import android.widget.RadioGroup; //导入 RadioGroup 类
import android.widget.RelativeLayout; //导入 RelativeLayout 类
import android.widget.TabHost; //导入 TabHost 类
import android.widget.TabWidget; //导入 TabWidget 类
import android.widget.TextView; //导入 TextView 类
import android.widget.Toast; //导入 Toast 类
import android.widget.TabHost.OnTabChangeListener; //导入 OnTabChangeListener 类

public class TabApplication extends TabActivity {

    TabHost myTabController = null; //定义 TabHost 控件对象 myTabController
    String Table1 = "Tab1";
    String Table2 = "Tab2";
    String Table3 = "Tab3";
    String Table4 = "Tab4";
    String Table5 = "Tab5";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle);
            setContentView(R.layout.main);
            myTabController = this.getTabHost();
            MyTabLayout tablayout = new MyTabLayout(this);
```



```

        myTabController.addTab(myTabController.newTabSpec(Table1).setIndicator("Tab 1",
        getResources().getDrawable(R.drawable.icon)).setContent(R.id.view1)); //设置 Table1 的内容
        myTabController.addTab(myTabController.newTabSpec(Table2).setIndicator("Tab 2",
        getResources().getDrawable(R.drawable.icon)).setContent(R.id.view2)); //设置 Table2 的内容
        myTabController.addTab(myTabController.newTabSpec(Table3).setIndicator("Tab 3").setContent
        (R.id.view3)); //设置 Table3 的内容
        myTabController.addTab(myTabController.newTabSpec(Table4).setIndicator("Tab 4").setContent
        (R.id.view4)); //设置 Table4 的内容
        myTabController.setOnTabChangeListener(new MyOnTabChangeListener());
        //注册 myTab Controller 的监听器
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(TabApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
        show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}

class MyOnTabChangeListener implements OnTabChangeListener
{

    @Override
    public void onTabChanged(String arg0) {
        // TODO Auto-generated method stub
        Toast.makeText(TabApplication.this, "Tab is changed", Toast.LENGTH_LONG).show();
    }

}

/*MyOnTabChangeListener 监听 Tab 的变化。当 Tab 变化时, onTabChanged 方法被调用*/

public class MyTabLayout implements TabHost.TabContentFactory {
    Activity myacti;
    LayoutInflater layoutInflater;

```



```

LinearLayout linearlayout;

public MyTabLayout (Activity a) {
    myacti = a;
    layoutInflater = a.getLayoutInflater();
}
/*MyTabLayout 的构造函数*/

public View generateTab(String string) {
    View tabview = addCustomView(string);
    return tabview;
}
/*generateTab 创建 Tab 视图*/

public View addCustomView(String viewid){

    linearlayout = new LinearLayout(myacti);
    linearlayout.setOrientation(LinearLayout.VERTICAL);

    if(viewid.equals(Table1)){
        Images imageview = new ImageView(myacti);    //使用 Activity 创建图片视图
        imageview.setImageResource(R.drawable.pic2);
        linearlayout.addView(imageview,
                                new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
LinearLayout.LayoutParams.FILL_PARENT));
    }
    /*若该视图是 Table1，则添加 Table1 标签对应的视图*/

    else if(viewid.equals(Table2)){
        EditText edittext = new EditText(myacti);
        linearlayout.addView(edittext,
                                new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
LinearLayout.LayoutParams.FILL_PARENT));

        Button button = new Button(myacti);
        button.setText("OK");
        button.setWidth(100);
    }
}

```



```

        linearlayout.addView(button,
                                new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
LinearLayout.LayoutParams.FILL_PARENT));

        RadioGroup radiogroup = new RadioGroup(myacti);
        radiogroup.setOrientation(LinearLayout.HORIZONTAL);
        RadioButton radioButton1 = new RadioButton(myacti);
        radioButton1.setText("Radio A");
        radiogroup.addView(radioButton1);
        RadioButton radioButton2 = new RadioButton(myacti);
        radioButton2.setText("Radio B");
        radiogroup.addView(radioButton2);

        linearlayout.addView(radiogroup,
                                new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT));
    }
    /*若该视图是 Table2, 则添加 Table2 标签对应的视图*/

    else if(viewid.equals(Table3)){

        LinearLayout.LayoutParams Table3_LP =
            new LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
            LinearLayout.LayoutParams.FILL_PARENT);

        linearlayout.addView(layoutInflator.inflate(R.layout.hello, null),Table3_LP);
    }
    /*若该视图是 Table3, 则添加 Table3 标签对应的视图*/

    else if(viewid.equals(Table4)){
        TextView textview = new TextView(myacti);
        textview.setText("HelloTags!");
        textview.setGravity(Gravity.CENTER);
        linearlayout.addView(textview);    //在该视图中添加文本视图
    }
    /*若该视图是 Table4, 则添加 Table4 标签对应的视图*/
    return linearlayout;
}

```



```

        }

    }
}
```

【代码说明】本实例通过 TabActivity 和 TabHost 实现了 Tab 的应用。TabHost 有两个重要的方法：一个是 addTab 方法，该方法用于增加标签；第二个是 setOnTabChangeListener 方法，该方法用于注册监听器。

在 Eclipse 下，运行该程序，运行结果如图 5-20 所示。

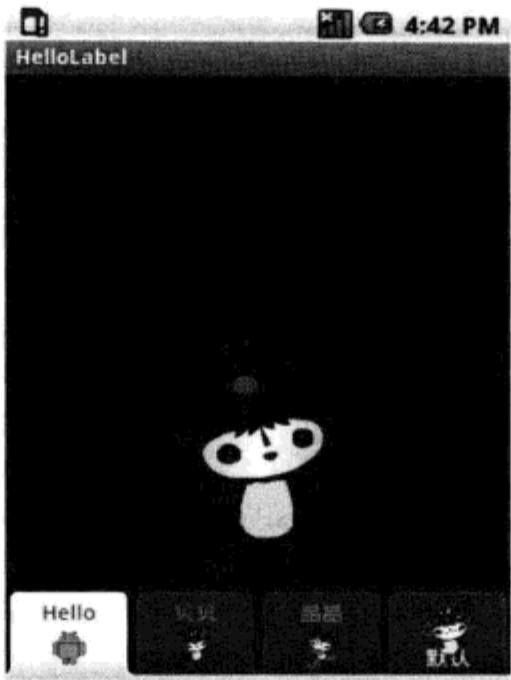


图 5-20 程序运行结果

## 5.2 通用 XML 属性

前面讲述了常用的组件，这些组件几乎囊括了 UI 的所有重要组件。有两种设计 UI 的方法：XML 文件和 Java 代码。其中，第一种方式由于简单易用，被广泛使用。Widget 所有的组件几乎都属于 View 类，有些属性在这些组件之间是通用的。表 5-8 列举了 Widget 组件通用的属性。

表 5-8 Widget 组件通用的属性

属性名称	描述
android:id	设置控件的索引，Java 程序可通过“R.id.<索引>”引用该控件
android:layout_height	设置布局高度，可以通过三种方式来指定高度： fill_parent：高度和父元素相同 wrap_content：高度随本身的内容进行调整 通过指定 px 值来设置高度
android:layout_width	设置布局宽度，可以通过三种方式来指定宽度： fill_parent：宽度和父元素相同 wrap_content：宽度随本身的内容进行调整 通过指定 px 值来设置宽度



续表

属性名称	描述
android:autoLink	设置是否当文本为 URL 链接、E-mail、电话号码、map 时，文本显示为可单击的链接。可选值为 none、web、email、phone、map、all
android:autoText	如果设置，将自动执行输入值的拼写纠正
android:bufferType	指定 getText()方法取得的文本类别。editable 类似于 StringBuilder 可追加字符，也就是说在 getText 方法后可调用 append 方法设置文本内容。spannable 则可在给定的字符区域使用样式
android:capitalize	设置英文字母大写类型
android:cursorVisible	设置光标为显示/隐藏，默认显示
android:digits	设置允许输入哪些字符，如 “1234567890.+-%\n()”
android:drawableBottom	在 text 的下方输出一个 drawable，如图片。如果指定一个颜色的话，会把 text 的背景设为该颜色，并且同时和 background 使用时覆盖后者
android:drawableLeft	在 text 的左边输出一个 drawable 对象，如图片
android:drawablePadding	设置 text 与 drawable(图片)的间隔，与 drawableLeft、drawableRight、drawableTop、drawableBottom 一起使用，可设置为负数，单独使用没有效果
android:drawableRight	在 text 的右边输出一个 drawable 对象，如图片
android:inputType	设置文本的类型，用于帮助输入法显示合适的键盘类型
android:cropToPadding	设置是否截取指定区域用空白代替。单独设置无效果，需要与 scrollY 一起使用。可选值为 true、false
android:maxHeight	设置 View 的最大高度





## 第 6 章 Android 菜单和布局设计

本章主要讲述 Android 提供的菜单组件以及布局组件,并且通过实例介绍这些组件的使用方式。其中菜单是一种特殊的组件,提供了层次结构的方式;而布局用来组织界面元素之间的位置。

### 6.1 菜单 (Menu)

#### 6.1.1 上下文菜单 (ContextMenu)

ContextMenu 是 android.view.Menu 的子类,提供了菜单的设计接口。不同于 Windows 系统中的菜单,使用 Android 上下文菜单时,需要长时间按住才能显示其子菜单。图 6-1 是一个上下文菜单的例子。

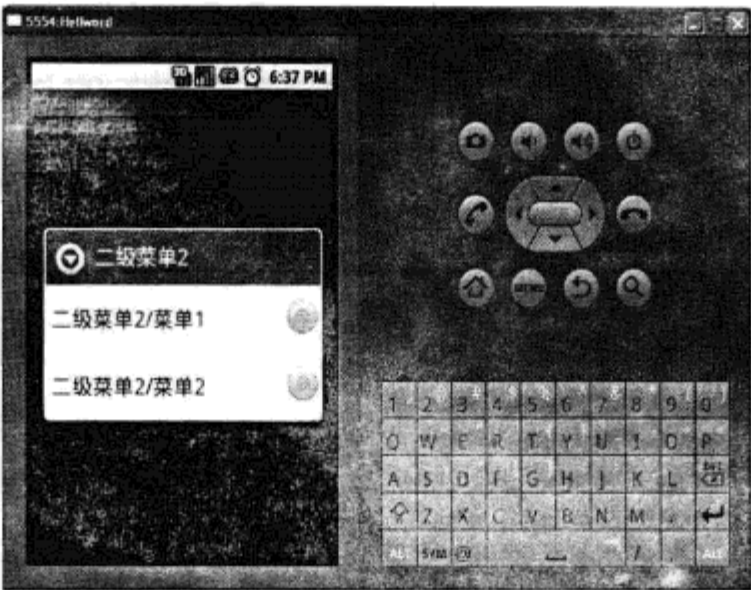


图 6-1 上下文菜单

ContextMenu 提供了用于创建和添加菜单的接口,表 6-1 列举了其常用的方法。

表 6-1 ContextMenu 的常用方法

方法	功能描述	返回值
setHeaderIcon(int iconRes)	设置上下文菜单的图标	ContextMenu
setHeaderIcon(Drawable icon)	设置上下文菜单的图标	ContextMenu
setHeaderTitle(CharSequence title)	设置上下文菜单的标题	ContextMenu
setHeaderTitle(int titleRes)	设置上下文菜单的标题	ContextMenu
add (int groupId, int itemId, int order, CharSequence title)	使用 add 方法添加子菜单: groupId: 组 ID ieemId: 菜单项 ID Order: 顺序号 title: 菜单项标题	Menu



下面以一个 ContextMenu 的应用为例说明上下文菜单的用法。

【实例 6-1】ContextMenu 组件的应用。

CMAApplication.java 实现，该类使用 ContextMenu 实现一个简单文件管理器的功能：

```
public class CMAApplication extends Activity
{
    /** Called when the activity is first created. */
    private TextView textview1; //声明文本框视图控件 textview1
    private TextView textview2; //声明文本框视图控件 textview2
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        try
        {
            setContentView(R.layout.main);
            init(); //初始化空间
            createComponent(); //根据 XML 创建组件
            register(); //将组件添加到到上下文菜单中
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(CMAApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
                show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
    /*onCreate 方法是程序的入口*/

    public void init()
    {
        textview1 = null;
        textview2 = null;
    }
}
```



/\*init 方法初始化文本框视图控件为空\*/

```
public void createComponent()
```

```
{
```

```
    textview1 = (TextView) this.findViewById(R.id.textview1);
```

```
    textview2 = (TextView) this.findViewById(R.id.textview2);
```

```
}
```

/\*createComponent 方法根据 XML 属性创建控件\*/

```
public void register()
```

```
{
```

```
    this.registerForContextMenu(textview1);
```

```
    this.registerForContextMenu(textview2);
```

```
}
```

/\*register 方法使用 registerForContextMenu 将控件注册到上下文菜单中\*/

@Override

```
public void onCreateContextMenu(ContextMenu contextMenu, View view, ContextMenuInfo contextMenuInfo)
```

```
{
```

```
    super.onCreateContextMenu(contextMenu, view, contextMenuInfo);
```

//指定父类的 onCreateContext Menu 方法

```
    if (view == textview1)
```

```
    {
```

```
        contextMenu.setHeaderIcon(R.drawable.icon); //设置上下文菜单的图标
```

```
        contextMenu.setHeaderTitle("My Menu"); //设置上下文菜单的标题
```

```
        contextMenu.add(1,0,0,"菜单 1");
```

/\*使用 add 方法添加子菜单

\* 第一个参数: 组 ID 为 1

\* 第二个参数: 菜单项 ID 为 0

\* 第三个参数: 顺序号为 0

\* 第四个参数: 菜单项上显示的内容为"菜单 1"\*/

```
        contextMenu.add(1,1,1,"菜单 2");
```

/\*使用 add 方法添加子菜单

\* 第一个参数: 组 ID 为 1

\* 第二个参数: 菜单项 ID 为 1

\* 第三个参数: 顺序号为 1

\* 第四个参数: 菜单项上显示的内容为"菜单 2"\*/

```
        contextMenu.add(1,1,2,"菜单 3");
```

/\*使用 add 方法添加子菜单



```

        * 第一个参数: 组 ID 为 1
        * 第二个参数: 菜单项 ID 为 1
        * 第三个参数: 顺序号为 2
        * 第四个参数: 菜单项上显示的内容为"菜单 3"*/
contextMenu.add(1,1,2,"菜单 4");
/*使用 add 方法添加子菜单
    * 第一个参数: 组 ID 为 1
    * 第二个参数: 菜单项 ID 为 1
    * 第三个参数: 顺序号为 3
    * 第四个参数: 菜单项上显示的内容为"菜单 4"*/
contextMenu.add(1,1,2,"菜单 5");
/*使用 add 方法添加子菜单
    * 第一个参数: 组 ID 为 1
    * 第二个参数: 菜单项 ID 为 1
    * 第三个参数: 顺序号为 4
    * 第四个参数: 菜单项上显示的内容为"菜单 5"*/
contextMenu.add(1,1,2,"菜单 6");
/*使用 add 方法添加子菜单
    * 第一个参数: 组 ID 为 1
    * 第二个参数: 菜单项 ID 为 1
    * 第三个参数: 顺序号为 5
    * 第四个参数: 菜单项上显示的内容为"菜单 6"*/
contextMenu.add(1,1,2,"菜单 7");
/*使用 add 方法添加子菜单
    * 第一个参数: 组 ID 为 1
    * 第二个参数: 菜单项 ID 为 1
    * 第三个参数: 顺序号为 6
    * 第四个参数: 菜单项上显示的内容为"菜单 7"*/
}

else if(view == textview2)
{
    SubMenu submenu1 = contextMenu.addSubMenu("二级菜单 1");
    //使用 addSubMenu 方法添加子菜单, 参数指定子菜单显示的内容
    submenu1.setHeaderIcon(R.drawable.icon); //设置子菜单的图标
    submenu1.add(0, 0, 0, "二级菜单 1/菜单 1"); //添加子菜单
    /*使用 add 方法添加子菜单
        * 第一个参数: 组 ID 为 0
        * 第二个参数: 菜单项 ID 为 0
        * 第三个参数: 顺序号为 0
    */
}

```



```

    * 第四个参数: 菜单项上显示的内容为"菜单 1"*/
    submenu1.add(0, 1, 1, "二级菜单 1/菜单 2");    //添加子菜单
    submenu1.setGroupCheckable(1, true, true);      //设置整个组可选
    SubMenu submenu2 = contextMenu.addSubMenu("二级菜单 2");
    submenu2.setIcon(R.drawable.icon);              //设置子菜单 submenu2 的图标
    submenu2.add(1, 0, 0, "二级菜单 2/菜单 1");    //添加子菜单
    submenu2.add(1, 1, 1, "二级菜单 2/菜单 2");    //添加子菜单
    submenu2.setGroupCheckable(1, true, true);      //设置整个组可选
    SubMenu submenu3 = contextMenu.addSubMenu("二级菜单 3");
    submenu3.setIcon(R.drawable.icon);              //设置子菜单 submenu3 的图标
    submenu3.add(1, 0, 0, "二级菜单 3/菜单 1");    //添加子菜单
    submenu3.add(1, 1, 1, "二级菜单 3/菜单 2");    //添加子菜单
    submenu3.setGroupCheckable(1, true, true);      //设置整个组可选
    SubMenu submenu4 = contextMenu.addSubMenu("二级菜单 4");
    submenu4.setIcon(R.drawable.icon);              //设置子菜单 submenu4 的图标
    submenu4.add(1, 0, 0, "二级菜单 4/菜单 1");    //添加子菜单
    submenu4.add(1, 1, 1, "二级菜单 4/菜单 2");    //添加子菜单
    submenu4.setGroupCheckable(1, true, true);      //设置整个组可选
    SubMenu submenu5 = contextMenu.addSubMenu("二级菜单 5");
    submenu5.setIcon(R.drawable.icon);              //设置子菜单 submenu5 的图标
    submenu5.add(1, 0, 0, "二级菜单 5/菜单 1");    //添加子菜单
    submenu5.add(1, 1, 1, "二级菜单 5/菜单 2");    //添加子菜单
    submenu5.setGroupCheckable(1, true, true);      //设置整个组可选
    SubMenu submenu6 = contextMenu.addSubMenu("二级菜单 6");
    submenu6.setIcon(R.drawable.icon);              //设置子菜单 submenu6 的图标
    submenu6.add(1, 0, 0, "二级菜单 6/菜单 1");    //添加子菜单
    submenu6.add(1, 1, 1, "二级菜单 6/菜单 2");    //添加子菜单
    submenu6.setGroupCheckable(1, true, true);      //设置整个组可选
}
}
/*重写父类的 onCreateContextMenu 方法, 该方法用于创建上下文菜单*/

```

**【代码说明】**本实例利用 ContextMenu 实现一个简单的文件管理器的功能。为了创建一个上下文菜单, 需要重写 onCreateContextMenu() 方法, 这个方法用于在创建上下文菜单的时候被调用。并且通过 registerForContextMenu() 方法将视图添加到上下文菜单中。

程序使用 setContentView() 方法初始化布局, setContentView() 方法使用如下的布局:

```

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

```



```

android:layout_width="fill_parent"
android:layout_height="fill_parent"    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="长时间按住显示菜单"    />
    <TextView
        android:id="@+id/textview1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="主菜单 2"    />
    <TextView
        android:id="@+id/textview2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="主菜单 1"    />
</LinearLayout>

```

该布局如图 6-2 所示。



图 6-2 程序布局

然后使用 `init` 初始化控件，设置两个文本框视图对象为空：

```

textview1 = null;
textview2 = null;

```

初始化之后，使用 `createComponent()` 方法根据 XML 创建组件。`createComponent()` 方法调用了 `findViewById` 来创建控件：

```

textview1 = (TextView) this.findViewById(R.id.textview1);
textview2 = (TextView) this.findViewById(R.id.textview2);

```

最后使用 `register()` 方法将组件添加到上下文菜单中，该方法封装了 `registerForContextMenu()` 方法：



```
this.registerForContextMenu(textview1);
this.registerForContextMenu(textview2);
```

registerForContextMenu()方法注册视图时，onCreateContextMenu()方法被调用。onCreateContextMenu()方法用于为指定的视图添加菜单，其包含三个参数：

- 参数 ContextMenu 指定当前的上下文菜单。
- 参数 View 是注册的视图，如 textview1。
- 参数 ContextMenuInfo 是当前的上下文菜单信息。

使用 registerForContextMenu()方法注册后，程序会自动调用重载的 onCreateContextMenu()方法。这个方法首先执行父类的方法来初始化环境，然后判断当前请求注册的视图。因为笔者故意使用不同的视图创建不同的应用。若为 textview1，则使用 add()方法添加菜单。add()方法是 ContextMenu 用于添加菜单的方法，其包含四个参数：

- 第一个参数为组标识。
- 第二个参数为菜单 ID。
- 第三个参数为菜单顺序号。
- 第四个参数为菜单标题。

若请求注册的视图是 textview2，过程就相对复杂了。程序使用 addSubMenu()方法添加子菜单，参数指定子菜单显示的内容，这个子菜单还可以再嵌套其他的子菜单。本程序在请求注册的视图是 textview2 的情况下，创建了 5 个子菜单，而每个子菜单又对应 2 个子菜单。这种层次结构类似于文件夹的组织结构，因而可通过这种方式实现文件浏览器。

至此，一个简单的文件浏览器就实现了。使用 Eclipse 将该程序部署到 Android 上，运行后可看到界面中只显示了几个文本框视图。可通过长时间按住“主菜单 1”或者“主菜单 2”来显示其对应的上下文菜单。图 6-3 显示了按住“主菜单 2”时，程序弹出的上下文菜单。

这些二级菜单又包含两个子菜单，单击二级菜单时，程序会弹出二级菜单对应的子菜单。例如单击“二级菜单 1”时，程序弹出二级菜单对应的子菜单，如图 6-4 所示。

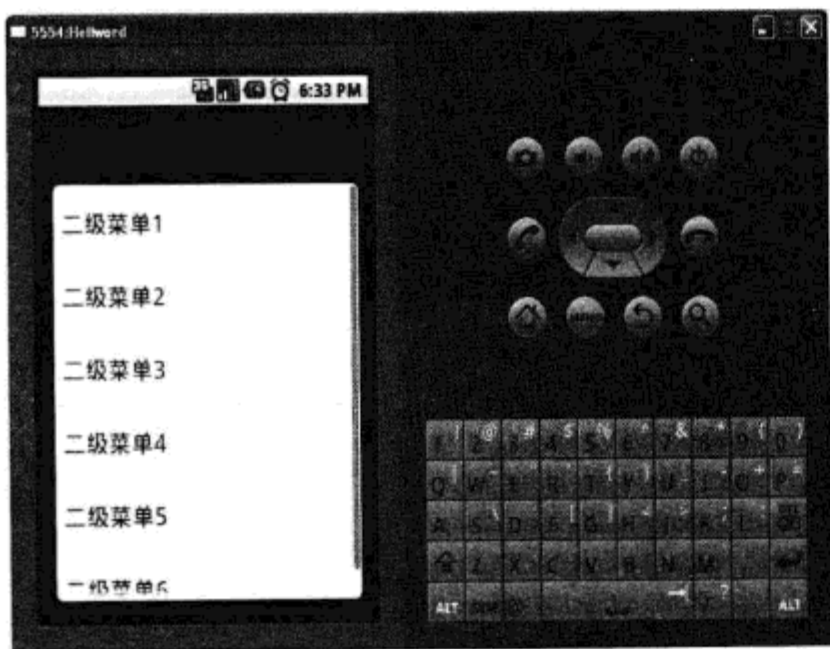


图 6-3 主菜单的上下文菜单

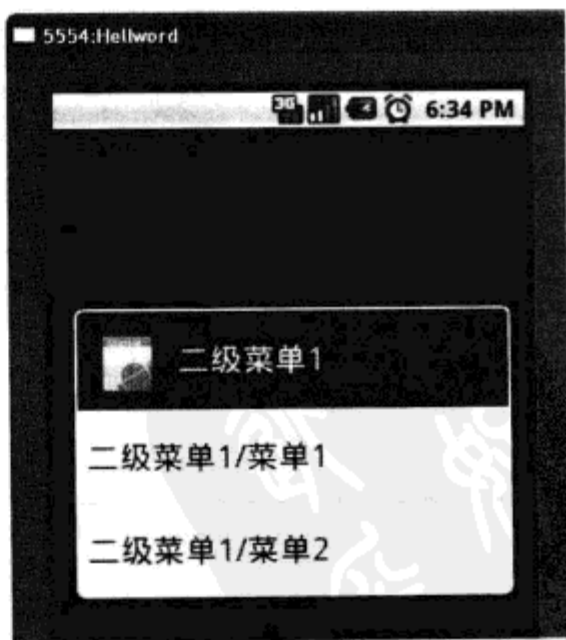


图 6-4 二级菜单的子菜单

### 6.1.2 选项菜单 (OptionsMenu)

Android 手机上有一个 Menu 按键，按下的时候，每个 Activity 都可以选择处理这一请求，在屏幕底部弹出一个菜单，这个菜单称为选项菜单 (OptionsMenu)。选项菜单提供了一种特殊



的菜单显示方式，这种菜单从视图底部弹出选项供用户选择。不同于上下文菜单，选项菜单没有对应的视图即用户无法通过单击屏幕上的视图来加载菜单选项。一般可通过按手机键盘上的 Menu 键来显示菜单选项。

图 6-5 所示是 Android 手机的选项菜单。

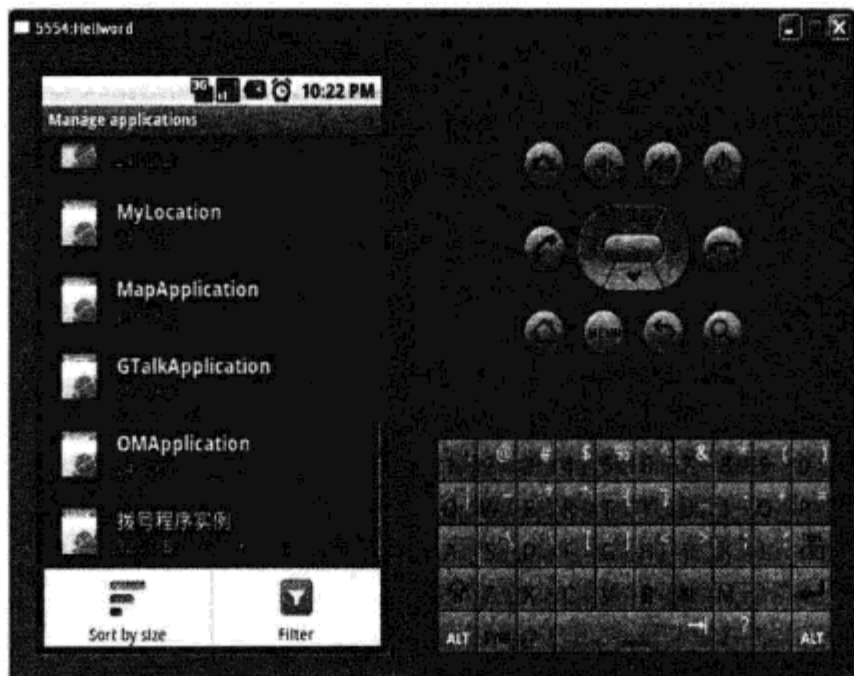


图 6-5 选项菜单

Android 开放了选项菜单的应用接口并屏蔽了其实现的复杂性，开发人员只需调用几个关键的方法就可以实现一个非常美观的选项菜单。下面通过例子讲述如何实现选项菜单（OptionsMenu）的功能。

### 【实例 6-2】OptionsMenu 组件的应用。

OMApplication.java 实现，该类实现了图片变化的功能：

```
package test.OMApplication;

import android.app.Activity; //导入活动类
import android.content.Intent; //导入意图类
import android.os.Bundle; //导入 Bundle 类
import android.view.Menu; //导入菜单类
import android.view.MenuItem; //导入菜单项类
import android.widget.Toast; //导入 Toast 类

public class OMAApplication extends Activity {
    /** Called when the activity is first created. */
    private MenuItem item_delete; //声明“删除”子菜单
    private MenuItem item_save; //声明“保存”子菜单
    private MenuItem item_help; //声明“帮助”子菜单
    private MenuItem item_add; //声明“添加”子菜单
    private MenuItem item_detail; //声明“详细”子菜单
    private MenuItem item_send; //声明“发送”子菜单
    private Intent item_delete_intent; //声明菜单项 item_delete 对应的意图
```



```

private Intent item_save_intent; //声明菜单项 item_save 对应的意图
private Intent item_help_intent; //声明菜单项 item_help 对应的意图
private Intent item_add_intent; //声明菜单项 item_add 对应的意图
private Intent item_detail_intent; //声明菜单项 item_add 对应的意图
private Intent item_send_intent; //声明菜单项 item_add 对应的意图
@Override
public void onCreate(Bundle bundle) {
    try
    {
        super.onCreate(bundle);    //执行父类的 onCreate 方法
        setContentView(R.layout.main); //使用 main.xml 作为程序布局
        init();
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(OMApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
            show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
private void init()
{
    item_delete = null; //置 item_delete 为 null
    item_save = null; //置 item_save 为 null
    item_help = null; //置 item_help 为 null
    item_add = null; //置 item_add 为 null
    item_detail = null; //置 item_detail 为 null
    item_send = null; //置 item_send 为 null
    item_delete_intent = null;
    item_save_intent = null;
    item_add_intent = null;
    item_add_intent = null;
    item_detail_intent = null;
    item_send_intent = null;
}

```



```

    }
    /*初始化菜单项 item_delete、item_save、item_help、item_add、item_detail 和 item_send 为 null*/

    public boolean onCreateOptionsMenu(Menu optionmenu) {
        try
        {
            item_delete = optionmenu.add(Menu.NONE, Menu.FIRST + 1, 1, "删除");
            /*使用 add 方法添加子菜单
            * 第一个参数: 组 ID 为 0
            * 第二个参数: 菜单项 ID 为 Menu.FIRST+1
            * 第三个参数: 顺序号为 0
            * 第四个参数: 菜单项上显示的内容为"菜单 1"*/
            item_delete.setIcon(android.R.drawable.ic_menu_delete);
            item_save = optionmenu.add(Menu.NONE, Menu.FIRST + 2, 2, "保存");
            item_save.setIcon( android.R.drawable.ic_menu_edit);
            item_help = optionmenu.add(Menu.NONE, Menu.FIRST + 3, 3, "帮助");
            item_help.setIcon(android.R.drawable.ic_menu_help);
            item_add = optionmenu.add(Menu.NONE, Menu.FIRST + 4, 4, "添加");
            item_add.setIcon(android.R.drawable.ic_menu_add);
            item_detail= optionmenu.add(Menu.NONE, Menu.FIRST + 5, 5, "详细");
            item_detail.setIcon(android.R.drawable.ic_menu_info_details);
            item_send = optionmenu.add(Menu.NONE, Menu.FIRST + 6, 6, "发送");
            item_send.setIcon(android.R.drawable.ic_menu_send);
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(OMApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
                show();
            return false;
        }
        /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
        return true;
    }
}
/*按 Menu 键时, 系统调用当前 Activity 的 onCreateOptionsMenu 方法, 并传送一个实现了 Menu

```



接口的 menu 对象供使用\*/

@Override

```
public boolean onOptionsItemSelected(MenuItem menuItem) {

    switch (menuItem.getItemId()) {
        case Menu.FIRST + 1:
            item_delete_intent = new Intent(OMApplication.this, delete.class); //创建 Intent
            startActivity(item_delete_intent); //启动 delete.class
            Toast.makeText(this, "删除菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 2:
            item_save_intent = new Intent(OMApplication.this, save.class); //创建 Intent
            startActivity(item_save_intent); //启动 save.class
            Toast.makeText(this, "保存菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 3:
            item_help_intent = new Intent(OMApplication.this, help.class); //创建 Intent
            startActivity(item_help_intent); //启动 help.class
            Toast.makeText(this, "帮助菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 4:
            item_add_intent = new Intent(OMApplication.this, add.class); //创建 Intent
            startActivity(item_add_intent); //启动 add.class
            Toast.makeText(this, "添加菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 5:
            item_add_intent = new Intent(OMApplication.this, detail.class); //创建 Intent
            startActivity(item_detail_intent); //启动 detail.class
            Toast.makeText(this, "详细菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case Menu.FIRST + 6:
            item_send_intent = new Intent(OMApplication.this, send.class); //创建 Intent
            startActivity(item_send_intent); //启动 send.class
            Toast.makeText(this, "发送菜单被单击了", Toast.LENGTH_LONG).show();
            break;
    }

    return false;
}
```



```

    }
    /*选择菜单项时，调用该方法*/

    @Override
    public void onOptionsMenuClosed(Menu menu){
        Toast.makeText(this, "选项菜单关闭了", Toast.LENGTH_LONG).show();
    }
    /*选项菜单被关闭事件，菜单被关闭有三种情形：Menu 键被再次按下、back 按钮被单击或者用户
    选择了某一个菜单项*/

    @Override
    public boolean onPrepareOptionsMenu(Menu menu){
        //TODO
        return true;
    }
    /*菜单被显示之前的事件*/
}

```

【代码说明】本实例实现了选项菜单的功能。主程序 `OMApplication.java` 使用 `main.xml` 作为程序布局，`main.xml` 定义了两个文本框视图。注意，这两个文本框视图不是菜单的父元素，不能通过长时间单击菜单来加载选项菜单。因为选项菜单没有对应的父元素，只能通过按键盘上的 Menu 键来加载。本实例的布局文件用做提示，程序 `main.xml` 定义如下：

```

<?xml version="1.0" encoding="utf-8"?>
<!-- 线性布局，高度和宽度同父元素相同，垂直分布 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:background="#102277"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 文本框视图控件，高度和宽度随内容变化 -->
    <TextView
        android:id="@+id/textview1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        android:layout_marginTop="6dp"
        android:text="Option Menu Application" >
    </TextView>
    <!-- 文本框视图控件，高度和宽度随内容变化 -->

```



```

<TextView
    android:id="@+id/textview2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:layout_marginTop="6dp"
    android:text="单击 Menu 键 OptionMenu" >
</TextView>
</LinearLayout>

```

加载布局之后，就可以使用程序创建选项菜单了。实现一个选项菜单的过程比较简单，需要重写 OptionsMenu 的 onPrepareOptionsMenu、onCreateOptionsMenu、onOptionsItemSelected 和 onOptionsMenuClosed 方法。这些方法的执行顺序为 onPrepareOptionsMenu→onCreateOptionsMenu→onOptionsItemSelected→onOptionsMenuClosed，这四个方法也标识了一个选项菜单的生命周期。

onPrepareOptionsMenu 方法在生成选项按钮之前被调用，使用该方法可以对选项按钮进行预处理。onCreateOptionsMenu 方法在 onPrepareOptionsMenu 之后调用，用于创建选项菜单。当用户选择子选项时，onOptionsItemSelected 方法被调用。退出选项菜单时，onOptionsMenuClosed 方法被调用。

在 onCreateOptionsMenu 方法中使用 add 方法为选项菜单中添加了：

```

private MenuItem item_delete;    //声明“删除”子菜单
private MenuItem item_save;      //声明“保存”子菜单
private MenuItem item_help;      //声明“帮助”子菜单
private MenuItem item_add;       //声明“添加”子菜单
private MenuItem item_detail;    //声明“详细”子菜单
private MenuItem item_send;      //声明“发送”子菜单

```

六个子菜单，这些子菜单使用 setIcon 方法设置选项菜单显示的图标，例如 android.R.drawable.ic\_menu\_delete。不难发现，本实例使用了 Android 系统自定义的图标。除了 android.R.drawable.ic\_menu\_delete、android.R.drawable.ic\_menu\_add、android.R.drawable.ic\_menu\_help、android.R.drawable.ic\_menu\_add、android.R.drawable.ic\_menu\_add、android.R.drawable.ic\_menu\_info\_details 和 android.R.drawable.ic\_menu\_send 图标之外，Android 还定义了其他图标，如表 6-2 所示。

表 6-2 android.R.drawable 属性

类型	名称	值
static final int	alert_dark_frame	17301504
static final int	alert_light_frame	17301505
static final int	arrow_down_float	17301506
static final int	bottom_bar	1730165
static final int	arrow_up_float	17301507
static final int	btn_default	17301508



续表		
类型	名称	值
static final int	btn_default_small	17301509
static final int	btn_dialog	17301527
static final int	btn_dropdown	17301510
static final int	btn_minus	17301511
static final int	btn_plus	17301512
static final int	btn_radio	17301513
static final int	btn_star	17301514
static final int	btn_star_big_off	17301515
static final int	btn_star_big_on	17301516
static final int	button_onoff_indicator_off	17301518
static final int	button_onoff_indicator_on	17301517
static final int	checkbox_off_background	17301519
static final int	checkbox_on_background	17301520
static final int	dark_header	17301669
static final int	dialog_frame	17301521
static final int	divider_horizontal_bright	17301522
static final int	divider_horizontal_dark	17301524
static final int	divider_horizontal_dim_dark	17301525
static final int	divider_horizontal_textfield	17301523
static final int	edit_text	17301526
static final int	editbox_background	17301528
static final int	editbox_background_normal	17301529
static final int	divider_horizontal_bright	17301522
static final int	editbox_dropdown_dark_frame	17301530
static final int	editbox_dropdown_light_frame	17301531
static final int	gallery_thumb	17301532
static final int	editbox_dropdown_dark_frame	17301530
static final int	ic_btn_speak_now	17301668
static final int	ic_delete	17301533
static final int	ic_dialog_alert 3	1730154
static final int	ic_dialog_dialer	1730154

当单击选项菜单时，onOptionsItemSelected 方法被调用。本实例重写的 onOptionsItemSelected 方法会根据被单击的菜单做相应的处理：

若单击“删除”子菜单，首先创建一个指向处理删除的类“delete.class”的意图，然后使用 startActivity 方法在当前的活动中启动 delete.class。下面是处理删除的类（delete.java）实现：

```

import android.app.Activity; //导入活动类
import android.os.Bundle;   //导入 Bundle 类

```



```
import android.widget.Toast; //导入 Toast 类

public class delete extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle);    //执行父类的 onCreate 方法
            setContentView(R.layout.delete); //使用 delete.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(delete.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}
```

其中布局实现如下:

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

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:background="#102277"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:text="删除菜单处理" >
    </TextView>
</LinearLayout>
```

若单击“保存”子菜单, 首先创建一个指向处理保存的类“save.class”意图, 然后使用



startActivity 方法在当前的活动中启动该类。下面是处理保存的类 (save.java) 实现:

```
import android.app.Activity; //导入活动类
import android.os.Bundle;    //导入 Bundle 类
import android.widget.Toast; //导入 Toast 类

public class save extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle);    //执行父类的 onCreate 方法
            setContentView(R.layout.save); //使用 save.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(save.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}
```

其中布局实现如下:

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

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:background="#102277"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:text="保存菜单处理">
```



```
</TextView>
```

```
</LinearLayout>
```

若单击“添加”子菜单，首先创建一个指向处理添加的类“add.class”意图，然后使用 `startActivity` 方法在当前的活动中启动该类。下面是处理添加的类（`add.java`）实现：

```
import android.app.Activity; //导入活动类
import android.os.Bundle;    //导入 Bundle 类
import android.widget.Toast; //导入 Toast 类

public class add extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle); //执行父类的 onCreate 方法
            setContentView(R.layout.add); //使用 add.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(add.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}
```

其中布局实现如下：

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

```
<LinearLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
android:orientation="vertical" android:background="#102277"
```

```
android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent">
```

```
<TextView
```



```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="40sp"
android:text="添加菜单处理" >
</TextView>
</LinearLayout>
```

若单击“帮助”子菜单，首先创建一个指向处理帮助类“help.class”意图，然后使用 startActivity 方法在当前的活动中启动该类。下面是处理帮助类（help.java）实现：

```
import android.app.Activity; //导入活动类
import android.os.Bundle;   //导入 Bundle 类
import android.widget.Toast; //导入 Toast 类

public class help extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle); //执行父类的 onCreate 方法
            setContentView(R.layout.help); //使用 help.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(help.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}
```

其中布局实现如下：

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

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```



```

android:orientation="vertical" android:background="#102277"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="40sp"
android:text="帮助菜单处理">
</TextView>
</LinearLayout>

```

若单击“详细”子菜单，首先创建一个指向处理详细的类“detail.class”意图，然后使用 startActivity 方法在当前的活动中启动该类。下面是处理详细的类（detail.java）实现：

```

import android.app.Activity; //导入活动类
import android.os.Bundle;   //导入 Bundle 类
import android.widget.Toast; //导入 Toast 类

public class detail extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle); //执行父类的 onCreate 方法
            setContentView(R.layout.detail); //使用 detail.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(detail.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}

```

其中布局实现如下：

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



```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical" android:background="#102277"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="40sp"
android:text="详细菜单处理" >
</TextView>
</LinearLayout>
```

若单击“发送”子菜单，首先创建一个指向处理发送的类“send.class”意图，然后使用 startActivity 方法在当前的活动中启动该类。下面是处理发送的类（send.java）实现：

```
import android.app.Activity; //导入活动类
import android.os.Bundle;    //导入 Bundle 类
import android.widget.Toast; //导入 Toast 类

public class send extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle);    //执行父类的 onCreate 方法
            setContentView(R.layout.send); //使用 send.xml 作为程序布局
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(send.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}
```



```
}
```

其中布局实现如下:

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

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:background="#102277"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:text="发送菜单处理">
    </TextView>
</LinearLayout>
```

最后, 需要注意在清单文件中添加处理这些子菜单的 Activity (add、delete、save、help、detail、send), 否则程序会出现异常。清单文件如下:

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".OMApplication"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".add" android:label="@string/app_name"/>
    <activity android:name=".delete" android:label="@string/app_name"/>
    <activity android:name=".save" android:label="@string/app_name"/>
    <activity android:name=".send" android:label="@string/app_name"/>
    <activity android:name=".detail" android:label="@string/app_name"/>
    <activity android:name=".help" android:label="@string/app_name"/>
</application>
```

至此, 实现了一个简单的选项菜单。使用 Eclipse 将该程序部署到 Android, 程序启动后界面上没有任何可单击的按钮, 如图 6-6 所示。

由于选项菜单没有对应的父元素, 可通过按键盘上的 Menu 键来加载选项视图。图 6-7 显示了按 Menu 键后, 程序加载一个包含六个子菜单的选项菜单。





图 6-6 程序启动后的界面

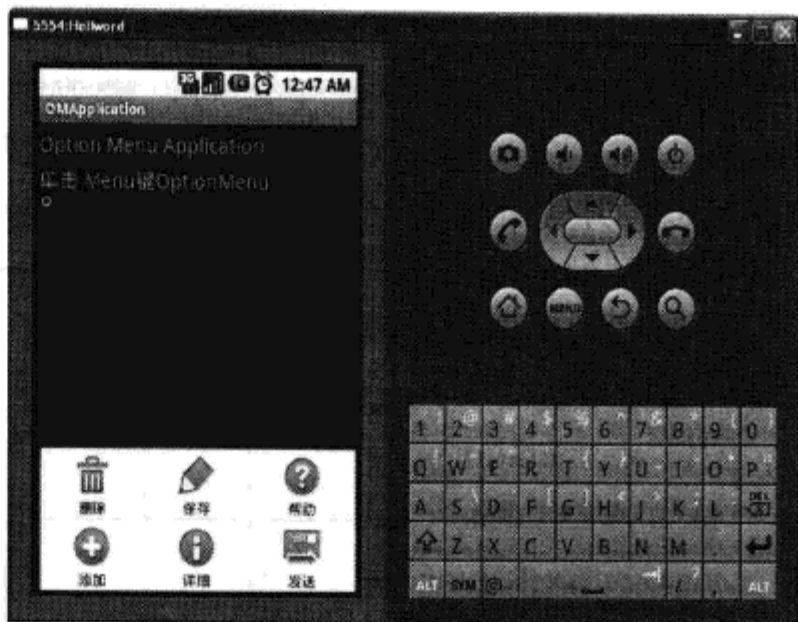


图 6-7 按 Menu 键弹出选项菜单

### 6.1.3 基于 XML 的菜单结构

上面两节讲到了上下文菜单和选项菜单，最后通过实例说明这两个菜单的用法。实例 6-1 和 6-2 主要通过菜单类提供的 API 接口来实现指定的菜单，例如使用 `add()` 方法添加相应的子菜单。前面利用 Java 代码实现菜单，本节将介绍使用 XML 创建菜单的方法，通常使用 Android 提供的 `menu` 标签来实现菜单。一般将 XML 的菜单文件存储到 `res\menu\` 目录下，然后使用 `R.menu` 来引用该资源。例如，下面是一个菜单的基本结构：

```
<MENU xmlns:android="http://schemas.android.com/apk/res/android">
  <GROUP android:id="@+id/group1" android:title="">
    <ITEM>
      <!--添加属性-->
    </ITEM>
    <ITEM>
      <!--添加属性-->
    </ITEM>
  </GROUP>
</MENU>
```

这里每个标签的定义如下：

(1) `<menu>` 是根元素，在 `<menu>` 根元素里面会嵌套 `<item>` 和 `<group>` 子元素，`<menu>` 根元素没有属性。

(2) `<item>` 元素中也可嵌套 `<menu>` 形成子菜单，这种嵌套同 `addSubMenu` 方法等效。

(3) `<group>` 表示一个菜单组，相同的菜单组可以一起设置其属性，例如 `visible`、`enabled` 和 `checkable` 等。`<group>` 元素的属性说明如下：

- `id`：唯一标识菜单组。
- `menuCategory`：对菜单进行分类，定义菜单的优先级。
- `orderInCategory`：组内的序号。
- `checkableBehavior`：规定选择行为，单选、多选还是其他。
- `visible`：是否可见，`true` 或者 `false`。



- enabled: 是否可用, true 或者 false。

使用 XML 创建好菜单之后, 需要使用 `MenuInflater` 来生成该菜单。`android.view.MenuInflater` 是 `java.lang.Object` 的子类, 该类用于将 XML 文件中定义类实例化为菜单对象。可使用两种方式来构造 `MenuInflater` 类的对象:

- 使用 `MenuInflater` 定义的构造函数: `publicMenuInflater (Context context)`。
- 除了构造函数之外, `Activity` 类提供了生成 `MenuInflater` 的方法: `getMenuInflater()`。一般使用这种方法生成 `MenuInflater` 对象。

假设 `res\menu\menu.xml` 定义了 menu 菜单结构, 可使用如下方法生成菜单:

```
MenuInflater inflater = getMenuInflater();
inflater.inflate(R.menu.menu, menu);
```

下面笔者使用 XML 分别实现了实例 6-1 和实例 6-2 的功能。

实例 6-1 实现了一个上下文菜单, 该实例展示了使用 `ContextMenu` 创建一个嵌套的菜单, 即多级菜单。下面的 XML 代码实现了实例 6-1 的功能:

```
<MENU xmlns:android="http://schemas.android.com/apk/res/android">
<!-- 第一个菜单 -->
<GROUP android:id="@+id/gruop1" android:title="">
    <ITEM
        android:id="@+id/menu1"
        android:title="菜单 1"
        android:orderInCategory="0"
        android:icon="@drawable/icon"
    ></ITEM>
    <ITEM
        android:id="@+id/menu2"
        android:title="菜单 2"
        android:orderInCategory="1"
        android:icon="@drawable/icon"
    ></ITEM>
    <ITEM
        android:id="@+id/menu3"
        android:title="菜单 3"
        android:orderInCategory="2"
        android:icon="@drawable/icon"
    ></ITEM>
    <ITEM
        android:id="@+id/menu4"
        android:title="菜单 4"
        android:orderInCategory="3"
        android:icon="@drawable/icon"
    ></ITEM>
```



```

        <ITEM
        android:id="@+id/menu5"
        android:title="菜单 5"
        android:orderInCategory="4"
        android:icon="@drawable/icon"
        ></ITEM>

        <ITEM
        android:id="@+id/menu6"
        android:title="菜单 6"
        android:orderInCategory="5"
        android:icon="@drawable/icon"
        ></ITEM>
    </GROUP>
</MENU>
<!-- 第二个菜单 -->
<MENU xmlns:android="http://schemas.android.com/apk/res/android">
    <GROUP android:id="@+id/gruop1" android:title="">
        <ITEM android:id="@+id/submenu1" android:title="二级菜单 1"
            android:icon="@drawable/icon">
            <MENU>
                <ITEM
                android:id="@+id/submenu11"
                android:title="二级菜单 1/菜单 1"
                android:orderInCategory="0"
                android:icon="@drawable/icon"
                >
                </ITEM>
                <ITEM
                android:id="@+id/submenu12"
                android:title="二级菜单 1/菜单 2"
                android:orderInCategory="0"
                android:icon="@drawable/icon"
                >
                </ITEM>
            </MENU>
        </ITEM>
        <ITEM android:id="@+id/submenu2" android:title="二级菜单 2"
            android:icon="@drawable/icon">
            <MENU>
                <ITEM

```



```

        android:id="@+id/submenu21"
        android:title="二级菜单 2/菜单 1"
            android:orderInCategory="0"
        android:icon="@drawable/icon"
    >
</ITEM>
<ITEM
    android:id="@+id/submenu22"
    android:title="二级菜单 2/菜单 2"
    android:orderInCategory="0"
    android:icon="@drawable/icon"
    >
</ITEM>
</MENU>
</ITEM>
<ITEM android:id="@+id/submenu3" android:title="二级菜单 3"
    android:icon="@drawable/icon">
    <MENU>
        <ITEM
            android:id="@+id/submenu31"
            android:title="二级菜单 3/菜单 1"
                android:orderInCategory="0"
            android:icon="@drawable/icon"
        >
    </ITEM>
    <ITEM
        android:id="@+id/submenu32"
        android:title="二级菜单 3/菜单 2"
        android:orderInCategory="0"
        android:icon="@drawable/icon"
    >
    </ITEM>
    </MENU>
</ITEM>
<ITEM android:id="@+id/submenu4" android:title="二级菜单 4"
    android:icon="@drawable/icon">
    <MENU>
        <ITEM
            android:id="@+id/submenu41"
            android:title="二级菜单 4/菜单 1"

```



```

        android:orderInCategory="0"
        android:icon="@drawable/icon"
    >
</ITEM>
<ITEM
    android:id="@+id/submenu42"
    android:title="二级菜单 4/菜单 2"
    android:orderInCategory="0"
    android:icon="@drawable/icon"
    >
</ITEM>
</MENU>
</ITEM>
<ITEM android:id="@+id/submenu5" android:title="二级菜单 5"
    android:icon="@drawable/icon">
    <MENU>
        <ITEM
            android:id="@+id/submenu51"
            android:title="二级菜单 5/菜单 1"
            android:orderInCategory="0"
            android:icon="@drawable/icon"
            >
        </ITEM>
        <ITEM
            android:id="@+id/submenu52"
            android:title="二级菜单 5/菜单 2"
            android:orderInCategory="0"
            android:icon="@drawable/icon"
            >
        </ITEM>
    </MENU>
</ITEM>
<ITEM android:id="@+id/submenu6" android:title="二级菜单 6"
    android:icon="@drawable/icon">
    <MENU>
        <ITEM
            android:id="@+id/submenu61"
            android:title="二级菜单 6/菜单 1"
            android:orderInCategory="0"
            android:icon="@drawable/icon"

```



```
>
</ITEM>
<ITEM
    android:id="@+id/submenu62"
    android:title="二级菜单 6/菜单 2"
    android:orderInCategory="0"
    android:icon="@drawable/icon"
>
</ITEM>
</MENU>

</ITEM>
</GROUP>
</MENU>
```

实例 6-2 实现了包含六个子菜单（“删除”子菜单、“保存”子菜单、“帮助”子菜单、“发送”子菜单、“添加”子菜单、“详细”子菜单）的选项菜单：

```
item_delete = optionmenu.add(Menu.NONE, Menu.FIRST + 1, 1, "删除");
item_delete.setIcon(android.R.drawable.ic_menu_delete);
item_save = optionmenu.add(Menu.NONE, Menu.FIRST + 2, 2, "保存");
item_save.setIcon( android.R.drawable.ic_menu_edit);
item_help = optionmenu.add(Menu.NONE, Menu.FIRST + 3, 3, "帮助");
item_help.setIcon(android.R.drawable.ic_menu_help);
item_add = optionmenu.add(Menu.NONE, Menu.FIRST + 4, 4, "添加");
item_add.setIcon(android.R.drawable.ic_menu_add);
item_detail= optionmenu.add(Menu.NONE, Menu.FIRST + 5, 5, "详细");
item_detail.setIcon(android.R.drawable.ic_menu_info_details);
item_send = optionmenu.add(Menu.NONE, Menu.FIRST + 6, 6, "发送");
item_send.setIcon(android.R.drawable.ic_menu_send); -->
item_delete = optionmenu.add(Menu.NONE, Menu.FIRST + 1, 1, "删除");
/*使用 add 方法添加子菜单
 * 第一个参数：组 ID 为 0
 * 第二个参数：菜单项 ID 为 Menu.FIRST+1
 * 第三个参数：顺序号为 0
 * 第四个参数：菜单项上显示的内容为"菜单 1"*/
item_delete.setIcon(android.R.drawable.ic_menu_delete);
item_save = optionmenu.add(Menu.NONE, Menu.FIRST + 2, 2, "保存");
item_save.setIcon( android.R.drawable.ic_menu_edit);
item_help = optionmenu.add(Menu.NONE, Menu.FIRST + 3, 3, "帮助");
item_help.setIcon(android.R.drawable.ic_menu_help);
item_add = optionmenu.add(Menu.NONE, Menu.FIRST + 4, 4, "添加");
item_add.setIcon(android.R.drawable.ic_menu_add);
```



```
item_detail= optionmenu.add(Menu.NONE, Menu.FIRST + 5, 5, "详细");
item_detail.setIcon(android.R.drawable.ic_menu_info_details);
item_send = optionmenu.add(Menu.NONE, Menu.FIRST + 6, 6, "发送");
item_send.setIcon(android.R.drawable.ic_menu_send);
```

使用 Android 的 menu 标签可实现上述菜单:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 使用 menu 标签创建选项菜单, 该菜单包含六个子菜单: “删除” 子菜单、“保存” 子菜单
“帮助” 子菜单、“发送” 子菜单、“添加” 子菜单、“详细” 子菜单-->
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- “删除” 选项定义 -->
    <item
        android:id="@+id/delete"
        android:title="删除"
        android:orderInCategory="1"
        android:icon="@android:drawable/ic_menu_delete" />
    <!-- “保存” 选项定义 -->
    <item
        android:id="@+id/save"
        android:title="保存"
        android:orderInCategory="2"
        android:icon="@android:drawable/ic_menu_edit" />
    <!-- “帮助” 选项定义 -->
    <item
        android:id="@+id/help"
        android:title="帮助"
        android:orderInCategory="3"
        android:icon="@android:drawable/ic_menu_help" />
    <!-- “添加” 选项定义 -->
    <item
        android:id="@+id/add"
        android:title="详细"
        android:orderInCategory="4"
        android:icon="@android:drawable/ic_menu_add" />
    <!-- “详细” 选项定义 -->
    <item
        android:id="@+id/details"
        android:title="详细"
        android:orderInCategory="5"
        android:icon="@android:drawable/ic_menu_info_details" />
    <!-- “发送” 选项定义 -->
```



```
<item
    android:id="@+id/send"
    android:title="发送"
    android:orderInCategory="6"
    android:icon="@android:drawable/ic_menu_send" />
</menu>
```

为了帮助开发人员理解 XML 创建菜单的机制，下面使用 XML 来实现例 6-2 的功能：

```
import android.app.Activity; //导入活动类
import android.content.Intent; //导入意图类
import android.os.Bundle; //导入 Bundle 类
import android.view.Menu; //导入菜单类
import android.view.MenuInflater;
import android.view.MenuItem; //导入菜单项类
import android.widget.Toast; //导入 Toast 类

public class XMLOMApplication extends Activity {
    /** Called when the activity is first created. */
    private Intent item_delete_intent; //声明菜单项 item_delete 对应的意图
    private Intent item_save_intent; //声明菜单项 item_save 对应的意图
    private Intent item_help_intent; //声明菜单项 item_help 对应的意图
    private Intent item_add_intent; //声明菜单项 item_add 对应的意图
    private Intent item_detail_intent; //声明菜单项 item_add 对应的意图
    private Intent item_send_intent; //声明菜单项 item_add 对应的意图
    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle); //执行父类的 onCreate 方法
            setContentView(R.layout.main); //使用 main.xml 作为程序布局
            init();
        }
        /**try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(XMLOMApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_
                LONG).show();
        }
        /**catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {

```



```

        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
private void init()
{
    item_save_intent = null;
    item_add_intent = null;
    item_add_intent= null;
    item_detail_intent= null;
    item_send_intent= null;
}
/*初始化菜单 item_delete、item_save、item_help、item_add、item_detail 和 item_send 为 null*/
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.layout.om, menu);
    return true;
}
/*使用 MenuInflater 加载菜单布局*/

public boolean onOptionsItemSelected(MenuItem menuItem) {

    switch (menuItem.getItemId()) {
        case R.id.delete:
            item_delete_intent = new Intent(XMLOMApplication.this,delete.class); //创建 Intent
            startActivity(item_delete_intent); //启动 delete.class
            Toast.makeText(this, "删除菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case R.id.save:
            item_save_intent = new Intent(XMLOMApplication.this,save.class); //创建 Intent
            startActivity(item_save_intent); //启动 save.class
            Toast.makeText(this, "保存菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case R.id.help:
            item_help_intent = new Intent(XMLOMApplication.this,help.class); //创建 Intent
            startActivity(item_help_intent); //启动 help.class
            Toast.makeText(this, "帮助菜单被单击了", Toast.LENGTH_LONG).show();
            break;
        case R.id.add:
            item_add_intent = new Intent(XMLOMApplication.this,add.class); //创建 Intent
    }
}

```



```

        startActivity(item_add_intent); //启动 add.class
        Toast.makeText(this, "添加菜单被单击了", Toast.LENGTH_LONG).show();
        break;
    case R.id.details:
        item_detail_intent = new Intent(XMLOMApplication.this,detail.class); //创建 Intent
        startActivity(item_detail_intent); //启动 detail.class
        Toast.makeText(this, "详细菜单被单击了", Toast.LENGTH_LONG).show();
        break;
    case R.id.send:
        item_send_intent = new Intent(XMLOMApplication.this,send.class); //创建 Intent
        startActivity(item_send_intent); //启动 send.class
        Toast.makeText(this, "发送菜单被单击了", Toast.LENGTH_LONG).show();
        break;
    }
    return false;
}
/*菜单项被选择时，该方法被调用*/
}

```

上述代码的功能同实例 6-2 的功能等效。主要使用 MenuInflater 类和 menu 标签来实现菜单结构，而实例 6-2 使用 add 方法来增加菜单。

使用 Eclipse 将上面的程序部署到 Android，程序启动后界面上没有任何可单击的按钮。按键盘上的 Menu 键，图 6-8 显示程序加载了一个包含六个子菜单的选项菜单。

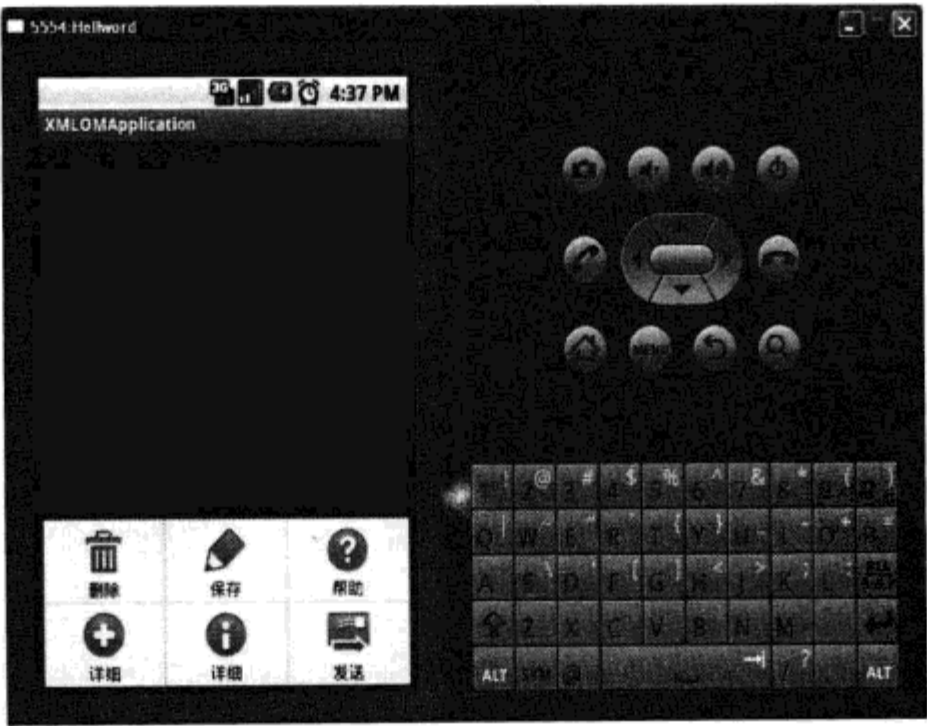


图 6-8 使用 XML 实现选项菜单



## 6.2 界面布局设计

### 6.2.1 基于 XML 的布局设计

Android 包含 LinearLayout、RelativeLayout、TableLayout 和 AbsoluteLayout 等多种布局。有两种实现布局的方式：一种是通过布局参数类提供的方法设置，另一种是通过 XML 属性设置。下面是用 XML 生成一个综合线性布局和相对布局的 UI：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- android:orientation 指定该布局为垂直线性排列，且高度和宽度同父元素相同-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 布局中的第一个按钮-->
<Button
    android:id="@+id/Button1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="First Button"
    />
    >
<!-- 布局中的第二个按钮-->
<Button
    android:id="@+id/Button2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Second Button"
    />
    <!-- 布局中的第三个按钮-->
<Button
    android:id="@+id/Button3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Third Button"
    />
<!-- 相对布局-->
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
```



```
>
<!-- 布局中的第四个按钮-->
<Button
    android:id="@+id/Button4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Fourth Button"
    android:layout_marginLeft="120px"
    android:layout_marginTop="120px"
/>
<!-- 布局中的第五个按钮-->
<Button
    android:id="@+id/Button5"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Fifth Button"
    android:layout_marginLeft="160px"
    android:layout_marginTop="160px"
/>
</RelativeLayout>
</LinearLayout>
```

布局生成效果如图 6-9 所示。

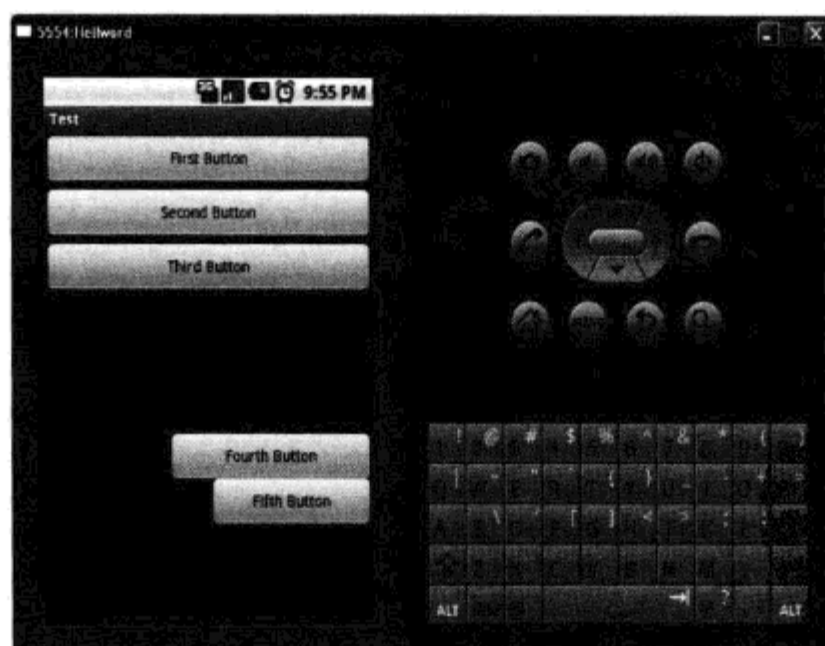


图 6-9 使用 XML 生成布局

布局设计是程序设计中最为核心的部分，Android 中常用的布局有以下几种：

#### (1) LinearLayout

LinearLayout 是一种线性排列的布局，在该布局中子元素之间呈线性排列，即顺序排列。由于布局是显示在二维空间中的，因此顺序排列是指在某一方向上的顺序排列，常见的有水平顺序排列和垂直顺序排列。



(2) RelativeLayout

RelativeLayout 是一种根据相对位置排列元素的布局，这种方式允许子元素指定它们相对于其他元素或父元素的位置（通过 ID 指定）。相对于线性布局，子元素可任意放置，没有规律性。需要注意，线性布局不需要特殊指定其父元素，但在使用之前必须指定其参照物，只有指定参照物之后，才能定义其相对位置。

(3) TableLayout

同 LinearLayout 类似，TableLayout 是一种表格布局，这种布局将子元素的位置分配到行或列中，即按照表格的数序排列。一个表格布局有多个“表格行”，而每个表格行又包含表格单元。需要注意，表格布局并不是真正意义上的表格，只是按照表格的方式组织子元素的位置。在表格布局之中，子元素之间并没有实际表格中的分隔线。

(4) AbsoluteLayout

相对布局需要指定其参照的父元素，AbsoluteLayout（绝对布局）与相对布局相反，绝对布局不需要指定其参照物，绝对布局使用整个手机界面作为坐标系，通过坐标系的两个偏移量（水平偏移量和垂直偏移量）来唯一指定其位置。

6.2.2 线性布局（LinearLayout）

在线性布局中，只有两种排列方式 vertical（垂直排列）和 horizontal（水平排列）。可通过属性 android:orientation 定义布局中子元素的排列方式。android.widget.LinearLayout 类是 android.view.ViewGroup 的子类，其派生了 RadioGroup、TabWidget、TableLayout、TableRow、ZoomControls 等类。

LinearLayout 类提供了定义线性布局的方法和属性，开发人员可根据这些方法实现线性布局相关应用。表 6-3 列举了 LinearLayout 常用的方法。

表 6-3 LinearLayout 常用的方法

方法	功能描述	返回值
LinearLayout	提供了 2 个构造函数：  LinearLayout(Context context)  LinearLayout(Context context, AttributeSet attrs)	null
getOrientation()	获取布局的方向设置。其中 0 代表水平方向，1 代表垂直方向	int
isBaselineAligned()	判断布局是否按照基线对齐	boolean
setBaselineAligned(boolean baselineAligned)	根据参数设置基线对齐	void
setGravity(int gravity)	根据指定的重力设置元素的大小	void
setHorizontalGravity(int gravity)	设置水平方向的重力	void
setVerticalGravity(int gravity)	设置垂直方向的重力	void
generateDefaultLayoutParams	返回包含宽度和高度的布局参数的集合	LayoutParams
setGravity(int gravity)	设置布局的重力	void

下面使用线性布局实现一个简单的图片浏览器，代码如下：

```

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

```



```

<!-- 线性布局垂直分布，高度和宽度同父元素相同 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 文本框视图，字体大小为 20px -->
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20px"
    android:text="风景图片 1"
    />
<!-- 图片视图，宽度和高度为 200px -->

<ImageView
    android:layout_width="100px"
    android:layout_height="100px"
    android:src="@drawable/pic12"
    />
<!-- 文本框视图，字体大小为 20px -->

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20px"
    android:text="风景图片 2"
    />
<!-- 图片视图，宽度和高度为 200px -->
<ImageView
    android:layout_width="100px"
    android:layout_height="100px"
    android:src="@drawable/pic2"
    />
<!-- 文本框视图-->
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20px"
    android:text="风景图片 3"

```



```

/>
<!-- 图片视图-->
<ImageView
    android:layout_width="100px"
    android:layout_height="100px"
    android:src="@drawable/pic3"
/>
</LinearLayout>

```

上述布局将图片按照线性方式垂直排列，实现了一个简单的图片浏览器功能。图 6-10 是上述 XML 的运行结果。



图 6-10 线性布局运行结果

### 6.2.3 相对布局（RelativeLayout）

android.widget.RelativeLayout 类是 android.view.ViewGroup 的子类，其层次关系如下所示：

```

java.lang.Object
android.view.View
android.widget.ViewGroup
android.widget.RelativeLayout
DialerFilter, TwoLineListItem

```

下面是使用 XML 实现相对布局的例子：

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/textview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Enter you sentence"/>

```



```

<EditText
    android:id="@+id/et"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@id/label"/>
</RelativeLayout>

```

RelativeLayout 类提供了定义相对布局的方法和属性，开发人员可根据这些方法实现相对布局相关应用。表 6-4 列举了 RelativeLayout 常用的方法。

表 6-4 RelativeLayout 常用的方法

方法	功能描述	返回值
RelativeLayout	提供了 2 个构造函数： <div>RelativeLayout(Context context)</div> <div>RelativeLayout(Context context, AttributeSet attrs)</div>	null
checkLayoutParams(ViewGroup.LayoutParams p)	检查参数指定的布局参数是否是 LayoutParams 实例	boolean
isBaselineAligned()	判断布局是否按照基线对齐	boolean
setBaselineAligned(boolean baselineAligned)	根据参数设置基线对齐	void
setGravity(int gravity)	根据指定的重力设置元素的大小	void
setHorizontalGravity(int gravity)	设置水平方向的重力	void
setVerticalGravity(int gravity)	设置垂直方向的重力	void
generateDefaultLayoutParams()	生成默认的布局参数实例	ViewGroup.LayoutParams

下面以一个具体的实例说明相对布局的用法，代码如下：

```

<?xml version="1.0" encoding="utf-8"?>
<!--相对布局-->
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#770000ff"
    android:padding="10dip">

    <TextView
        android:id="@+id/textview1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Username: " />

```



<!--这个 EditText 放置在上边 id 为 label 的 TextView 的下边-->

```
<EditText
    android:id="@+id/entry"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="@android:drawable/editbox_background"
    android:layout_below="@id/label" />
```

<!-- “取消” 按钮和容器的右边齐平，并且设置左边的边距为 10dip-->

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/entry"
    android:layout_alignParentRight="true"
    android:layout_marginLeft="10dip"
    android:text="取消" />
```

<!-- “确定” 按钮在 “取消” 按钮的左侧，并且和 “取消” 按钮的高度齐平-->

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/cancel"
    android:layout_alignTop="@id/cancel" android:text="确定" />
```

```
</RelativeLayout>
```

上面的程序使用相对布局实现了一个用户名输入界面。其中用户名输入框使用 `android:layout_below` 属性指定其位置。图 6-11 是相对布局的运行结果。



图 6-11 相对布局运行结果



6.2.4 表格布局（TableLayout）

事实上，TableLayout 是 LinearLayout 的子类，读者可以尝试使用 LinearLayout 实现表格布局的功能。在本章的最后一节，笔者使用 LinearLayout 实现了表格布局的功能，这里不再赘述。TableLayout 的层次关系如下：

```

java.lang.Object
android.view.View
android.widget.ViewGroup
android.widget.LinearLayout
android.widget.TableLayout
    
```

TableLayout 类提供了定义表格布局的方法和属性，开发人员可根据这些方法实现表格布局相关应用。 需要注意，表格布局不能设置其子元素的宽度属性（layout\_widt），该属性必须同父元素相同（FILL\_PARENT）。表 6-5 列举了 TableLayout 常用的方法。

表 6-5 TableLayout 常用的方法

方法	功能描述	返回值
TableLayout	提供了 2 个构造函数：  TableLayout(Context context)  TableLayout(Context context, AttributeSet attrs)	null
addView(View child)	将参数指定的视图添加到表格布局中	void
addView(View child)	判断布局是否按照基线对齐。	void
isColumnCollapsed(int columnIndex)	该方法判断指定列的折叠情况	boolean
isColumnShrinkable(int c)	判断参数指定的列是否可收缩	boolean
isShrinkAllColumns()	该方法不同于 isColumnShrinkable(int c)，用来判断所有的列是否可收缩	boolean

下面以一个具体的实例说明表格布局的用法，代码如下：

```

import android.app.Activity; //导入活动类
import android.os.Bundle;    //导入 Bundle 类
import android.provider.MediaStore.Images; //导入图片类
import android.view.View;    //导入视图类
import android.view.ViewGroup; //导入视图组类
import android.view.ViewGroup.LayoutParams; //导入布局参数类
import android.widget.ImageView; //导入图片视图类
import android.widget.TableLayout; //导入表格布局类
import android.widget.TableRow; //导入表格行类
import android.widget.TextView; //导入文本框视图类
import android.widget.Toast; //必须导入消息类

public class TLayout extends Activity {
    /** Called when the activity is first created. */
    
```



```
@Override
public void onCreate(Bundle bundle) {
    try
    {
        super.onCreate(bundle);    //必须执行父类的 onCreate 方法
        setContentView(R.layout.main);    //使用 main.xml 初始化程序布局
        TableLayout tableLayout = (TableLayout)findViewById(R.id.layout);
        //创建 TableLayout 对象
        tableLayout.setStretchAllColumns(true);

        for(int row=0;row<3;row++)
        {
            TableRow tableRow=new TableRow(this);
            TextView tv=new TextView(this);
            ImageView iv=new ImageView(this);
            iv.setMaxHeight(1);
            iv.setMaxWidth(1);
            tv.setText("风景");
            iv.setBackgroundResource(R.drawable.pic12);
            tableRow.addView(tv); //表格行添加文本框视图对象
            tableRow.addView(iv); //表格行添加图片视图对象
            tableLayout.addView(tableRow, new TableLayout.LayoutParams(ViewGroup.LayoutParams.FILL_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
            //新建的 TableRow 添加到 TableLayout
        }
        /*循环添加 3 个表格布局列*/
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(TLayout.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
```



```
}
```

上面的程序实现了一个表格布局。其中使用 `main.xml` 初始化布局, 该布局中包含一个 `TableLayout` 元素:

```
<TableLayout
    android:id="@+id/layout"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</TableLayout>
```

表格布局循环添加了 3 个表格行, 通过表格行添加表格元素。 `TableRow` 开放了添加视图元素的方法, 可使用 `addView` 方法添加指定的视图。添加视图之后, 按照行顺序组织。 `TableLayout` 的 `addView` 方法需要指定两个参数:

- 表格行: 该参数指定要添加到布局的行。
- 布局参数: 该参数指定要添加到布局的布局参数。

由于循环添加了 3 个表格行, 因而可在布局中看到 3 个表格行, 而每行又包含一个文本框视图和一个图片视图。图 6-12 是表格布局的运行结果。



图 6-12 表格布局运行结果

### 6.2.5 绝对布局 (AbsoluteLayout)

对于 `AbsoluteLayout` 标签, 主要使用的两个参数是:

- `android:layout_x`: 指定  $x$  坐标的位置。
- `android:layout_y`: 指定  $y$  坐标的位置。

下面是一个绝对布局的例子:

```
<AbsoluteLayout>
<TextView
    android:text="textview"
    android:id="@+id/tv"
    android:layout_height="wrap_content"
    android:layout_y="20px"
```



```

android:layout_width="wrap_content"
android:layout_x="40px"/>
</AbsoluteLayout>

```

同其他布局一样，`android.widget.AbsoluteLayout` 是 `android.view.ViewGroup` 类的子类，其层次关系如下：

```

java.lang.Object
android.view.View
android.widget.ViewGroup
android.widget.AbsoluteLayout

```

`android.widget.AbsoluteLayout` 常用的方法如表 6-6 所示。

表 6-6 `AbsoluteLayout` 常用的方法

方法	功能描述	返回值
<code>AbsoluteLayout</code>	提供了 3 个构造函数： <code>public AbsoluteLayout (Context context)</code> <code>public AbsoluteLayout (Context context, AttributeSet attrs)</code> <code>Public AbsoluteLayout (Context context,AttributeSet attrs, int defStyle)</code>	<code>null</code>
<code>checkLayoutParams(ViewGroup.LayoutParams p)</code>	检查参数指定的布局参数是否是 <code>LayoutParams</code> 实例	<code>boolean</code>
<code>onLayout (boolean changed, int l, int t, int r, int b)</code>	视图的布局改变时，该方法被调用	<code>boolean</code>
<code>setBaselineAligned(boolean baselineAligned)</code>	根据参数设置基线对齐	<code>void</code>
<code>onMeasure (int widthMeasureSpec, int h)</code>	该方法被 <code>measure</code> 调用，用于测量视图的宽度和高度	<code>void</code>
<code>setHorizontalGravity(int gravity)</code>	设置水平方向的重力	<code>void</code>

下面以一个具体的实例说明绝对布局的用法。

AL.java 实现：

```

public class TLayout extends Activity {
    /** Called when the activity is first created. */

    @Override
    public void onCreate(Bundle bundle) {
        try
        {
            super.onCreate(bundle);    //必须执行父类的 onCreate 方法
            setContentView(R.layout.main);    //使用 main.xml 初始化程序布局

        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {

```



```

        Toast.makeText(TLayout.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
}

```

布局实现:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- 采用绝对布局, 高度和宽度同父元素相同 -->
<AbsoluteLayout android:id="@+id/left"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/t1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="40px"
        android:layout_y="10px"
        android:textSize="20px"
        android:text="风景图片 1"
    />
    <ImageView android:id="@+id/p1"
        android:src="@drawable/pic12"
        android:layout_width="100px"
        android:layout_height="100px"
        android:layout_x="40px"
        android:layout_y="40px"
    />

    <TextView android:id="@+id/t2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="40px"
        android:layout_y="140px"
        android:textSize="20px"
        android:text="风景图片 2"
    />

```



```

/>

<ImageView android:id="@+id/p2"
    android:src="@drawable/pic2"
    android:layout_width="100px"
    android:layout_height="100px"
    android:layout_x="40px"
    android:layout_y="170px"
/>

<TextView android:id="@+id/t3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="40px"
    android:layout_y="270px"
    android:textSize="20px"
    android:text="风景图片 3"
/>

<ImageView android:id="@+id/p3"
    android:src="@drawable/pic3"
    android:layout_width="100px"
    android:layout_height="100px"
    android:layout_x="40px"
    android:layout_y="300px"
/>

</AbsoluteLayout>

```

上面的程序使用绝对布局实现了线性布局的功能。线性布局不需要指定子元素的位置，子元素是按照顺序排列的，而线性布局可以随意指定子元素的位置。图 6-13 是绝对布局的运行结果。

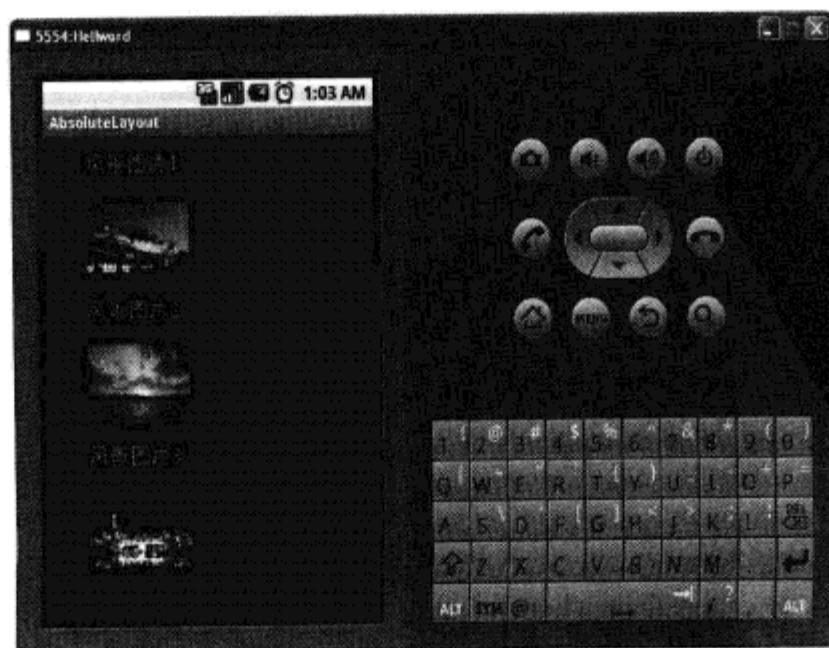


图 6-13 绝对布局运行结果



6.3 界面中的字体

在布局设计中，除了位置之外，还需要设置字体、文本大小、颜色等。Android 提供两种设置字体的方式：

- 通过 android 类提供的字体库设置字体。
- 使用自定义字体库设置字体。

这两种方式都需要用到 android.graphics.Typeface 类，本节将介绍这两种设置字体的方式。

6.3.1 设置系统字体

设置字体是 UI 的一个重要方面，合适的字体不仅有利于阅读，而且有利于信息的传达。例如，在一篇几万字的文章里，可以将重点内容加粗来提示读者注意。布局杂乱无章，字体不规整，会降低信息的可读性。

Android 提供了 android.graphics.Typeface 类来设置字体，android.graphics.Typeface 类是 java.lang.Object 类的子类。android.graphics.Typeface 提供了用于设置字体的方法和属性，表 6-7 列举了 android.graphics.Typeface 常用的方法。

表 6-7 android.graphics.Typeface 常用的方法

方法	功能描述	返回值
create(Typeface family, int style)	根据参数指定的字型和字体创建 Typeface	Typeface
getStyle()	该方法用于获取字体的风格	int
isBold()	该方法判断字体是否为粗体	boolean
isItalic()	该方法判断字体是否为斜体	boolean
create(String familyName, int style)	根据参数指定的字型和字体创建 Typeface	Typeface
createFromAsset(AssetManager mgr, String path)	根据参数指定的 AssetManager 和路径创建 Typeface	Typeface
createFromFile(File path)	根据参数指定的文件路径创建 Typeface	Typeface
defaultFromStyle(int style)	返回默认的 Typeface	Typeface

除了定义字体相关的方法之外，android.graphics.Typeface 还定义了五种字型和四种字体常量：

- Typeface DEFAULT：默认。
- Typeface DEFAULT\_BOLD：默认黑体。
- Typeface MONOSPACE：等宽字型。
- Typeface SANS\_SERIF：无衬线字型。
- Typeface SERIF：衬线字型。
- int BOLD：黑体。
- int BOLD\_ITALIC：粗斜体。
- int ITALIC：斜体。
- int NORMAL：正常字体。

下面以一个实例说明 Typeface 的基本用法。

【实例 6-3】设置系统字体。





Typeface.java 实现, 该类实现了控制显示多种字体的功能。

```
import android.app.Activity;    //导入活动类
import android.graphics.Color;  //导入颜色类
import android.graphics.Paint;  //导入画笔类
import android.graphics.Typeface; //导入字体类, 提供设置字体的接口
import android.os.Bundle;       //导入 Bundle 类
import android.widget.EditText;  //导入文本框视图类
import android.widget.RadioButton; //导入单选按钮类
import android.widget.RadioGroup; //导入单选按钮组类
import android.widget.TextView;  //导入文本框视图类
import android.widget.Toast;     //导入消息类

public class Ex_427 extends Activity {
    /** Called when the activity is first created. */
    private TextView textview;    //声明 TextView 对象
    private EditText edittext;    //声明 EditText 对象
    private RadioGroup RG;       //声明 RadioGroup 对象
    private RadioButton RB1;     //声明 RadioButton 对象
    private RadioButton RB2;     //声明 RadioButton 对象
    private RadioButton RB3;     //声明 RadioButton 对象
    private RadioButton RB4;     //声明 RadioButton 对象
    private RadioButton RB5;     //声明 RadioButton 对象
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        try
        {
            init(); //初始化
            setContentView(R.layout.main);
            edittext = (EditText) findViewById(R.id.edittext); //生成 EditText 对象, 显示输入
            textview = (TextView) findViewById(R.id.textview); //生成 TextView 对象, 显示规定字体
            RG = (RadioGroup) findViewById(R.id.RG); //生成 RadioGroup 对象, 该组包含 5 个单选按钮
            RB1 = (RadioButton) findViewById(R.id.RB1); //生成第一个单选按钮对象, 选择默认字体
            RB2 = (RadioButton) findViewById(R.id.RB2); //生成第二个单选按钮对象, 选择粗体字体
            RB3 = (RadioButton) findViewById(R.id.RB3); //生成第三个单选按钮对象, 选择斜体字体
            RB4 = (RadioButton) findViewById(R.id.RB4); //生成第四个单选按钮对象, 选择粗斜体字体
            RB5 = (RadioButton) findViewById(R.id.RB5); //生成第五个单选按钮对象, 选择仿粗体字体

            //将视图加入到布局中
            RG.setOnCheckedChangeListener(ChangeRadioGroup);
            /*使用 setOnCheckedChangeListenerRadioGroup 注册单选按钮状态改变监听器*/
        }
    }
}
```



```

    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(Ex_427.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/

}
/*onCreate 方法是程序入口点*/

private void init()
{
    textview = null;
    edittext = null;
    RG = null;
    RB1 = null;
    RB2 = null;
    RB3 = null;
    RB4 = null;
    RB5 = null;
}
/*init 方法置控件为空*/

private RadioGroup.OnCheckedChangeListener ChangeRadioGroup = new
RadioGroup.OnCheckedChangeListener()
{
    @Override
    public void onCheckedChanged(RadioGroup group, int checkedId)
    {
        // TODO Auto-generated method stub
        if(checkedId==RB1.getId() && RB1.isChecked()){
            textview.setText(edittext.getText());
            textview.setTextColor(Color.BLUE);
            textview.setTypeface(Typeface.DEFAULT, Typeface.NORMAL);
        }
    }
}

```



```

        //设置字型为默认黑体, 正常字体
        Toast.makeText(Ex_427.this, RB1.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB1 被选中, textview 显示 edittext 的内容为默认字体*/
    else if(checkedImage==RB2.getId() && RB2.isChecked()){
        textview.setText(edittext.getText());
        textview.setTextColor(Color.CYAN);
        textview.setTypeface(Typeface.DEFAULT_BOLD, Typeface.BOLD);
        //设置字型为默认黑体, 粗体字体
        Toast.makeText(Ex_427.this, RB2.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB2 被选中, textview 显示 edittext 的内容为粗体字体*/
    else if(checkedImage==RB3.getId() && RB3.isChecked()){
        textview.setText(edittext.getText());
        textview.setTextColor(Color.GREEN);
        textview.setTypeface(Typeface.MONOSPACE, Typeface.ITALIC);
        //设置字型为等宽字型, 斜体字体
        Toast.makeText(Ex_427.this, RB2.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB3 被选中, textview 显示 edittext 的内容为斜体字体*/
    else if(checkedImage==RB4.getId() && RB4.isChecked()){
        textview.setText(edittext.getText());
        textview.setTextColor(Color.RED);
        textview.setTypeface(Typeface.MONOSPACE, Typeface.BOLD_ITALIC);
        //设置字型为等宽字型, 斜粗体字体
        Toast.makeText(Ex_427.this, RB2.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB4 被选中, textview 显示 edittext 的内容为粗斜体字体*/
    else if(checkedImage==RB5.getId() && RB5.isChecked()){
        textview.setText(edittext.getText());
        textview.setTextColor(Color.YELLOW);
        textview.setTypeface(Typeface.DEFAULT_BOLD, Typeface.BOLD);
        //设置字型为默认粗体
        textview.getPaint().setFakeBoldText(true); //使用 setFakeBoldText 设置仿粗体
        Toast.makeText(Ex_427.this, RB2.getText()+"被选择", Toast.LENGTH_LONG).show();
    }
    /*若 RB5 被选中, textview 显示 edittext 的内容为仿粗体字体*/
}

};
/*定义 RadioGroup 状态改变事件监听器。当单选按钮的状态发生改变时, onCheckedChanged 方法

```



被调用\*/

}

【代码说明】本实例包含一个文本输入框用于接收用户的输入，还有一个可选的单选按钮组和一个显示相应字体的文本框视图。用户可在输入框中输入字符，然后选择相应的单选按钮，对应风格的文字会在文本框视图中显示出来。界面定义如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 线性布局垂直分布，高度和宽度同父元素相同 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <!--文本输入框，接受输入-->
    <EditText
        android:id="@+id/edittext"
        android:layout_width="300px"
        android:layout_height="60px"
        android:textSize="30sp"
        android:hint="@string/texthint"
    />
    <!--创建一个 RadioGroup，该组包含 5 个单选按钮-->
    <RadioGroup
        android:id="@+id/RG"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/RG"
    >
        <!--第一个 RadioButton -->
        <RadioButton
            android:id="@+id/RB1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/RB1"
        />
        <!--第二个 RadioButton -->
        <RadioButton
            android:id="@+id/RB2"
            android:layout_width="wrap_content"
```



```

android:layout_height="wrap_content"
android:text="@string/RB2"
/>
<!-- 第三个 RadioButton -->
<RadioButton
android:id="@+id/RB3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/RB3"
/>
<!-- 第四个 RadioButton -->
<RadioButton
android:id="@+id/RB4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/RB4"
/>
<!-- 第五个 RadioButton -->
<RadioButton
android:id="@+id/RB5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/RB5"
/>
</RadioGroup>
<!-- 文本框视图，根据字体显示 -->
<TextView
android:id="@+id/textview"
android:layout_width="300px"
android:layout_height="60px"
android:textSize="30sp"
android:text="@string/texthint"
/>
</LinearLayout>

```

该布局定义的界面如图 6-14 所示。

本实例使用单选按钮作为选择字体的方式，五个单选按钮分别对应六种字体：粗体、斜体、粗斜体、正常字体、仿粗体以及默认字体。单选按钮的监听器是 `OnCheckedChangeListener`，设置监听器的方法是 `setOnCheckedChangeListener`。

字体设置是通过 `setTypeface` 方法实现的，几乎每个控件都有 `setTypeface` 方法。`setTypeface` 方法的原型为：



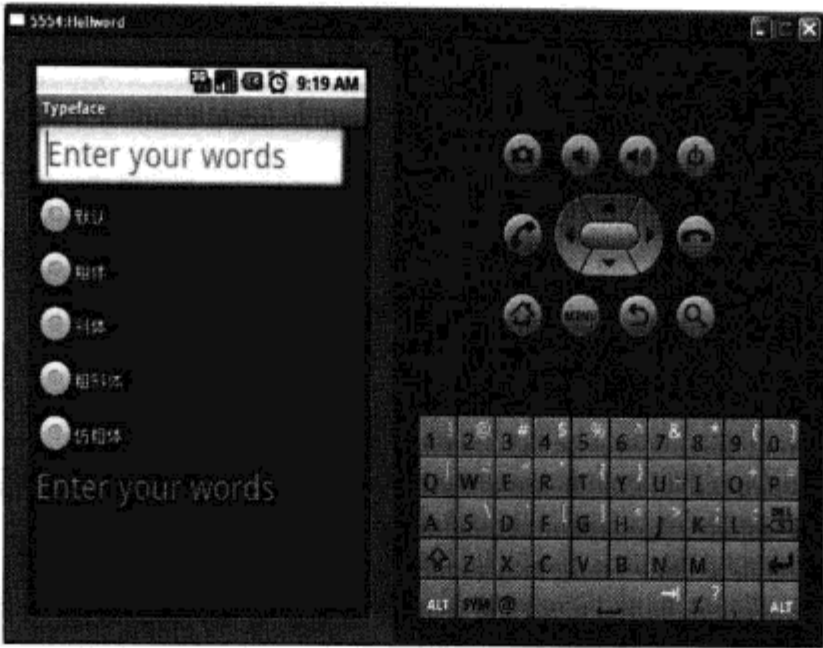


图 6-14 程序界面

```
void setTypeface(Typeface typeface, int style)
```

其中，参数 `typeface` 指定显示的字型，有五种取值：`Typeface.DEFAULT`、`Typeface.DEFAULT_BOLD`、`Typeface.MONOSPACE`、`Typeface.SERIF`；参数 `style` 指定显示的字体，有四种取值：`Typeface.BOLD`、`Typeface.BOLD_ITALIC`、`Typeface.ITALIC`、`Typeface.NORMAL`。

- 输入字符之后单击单选按钮，程序会根据单选按钮设置相应的字体：
- 当单击“粗体”单选按钮时，程序会调用如下方法：
 

```
textView.setTypeface(Typeface.DEFAULT_BOLD, Typeface.BOLD);
```

 该方法设置字型为默认粗体，粗体字体。
  - 当单击“斜体”单选按钮时，程序会调用如下方法：
 

```
textView.setTypeface(Typeface.MONOSPACE, Typeface.ITALIC);
```

 该方法设置字型为等宽字型，斜体字体。
  - 当单击斜“粗体”单选按钮时，程序会调用如下方法：
 

```
textView.setTypeface(Typeface.MONOSPACE, Typeface.BOLD_ITALIC);
```

 该方法设置字型为等宽字型，斜粗体字体。
  - 当单击“斜体”单选按钮时，程序会调用如下方法：
 

```
textView.setTypeface(Typeface.MONOSPACE, Typeface.BOLD_ITALIC);
```

 该方法设置字型为默认粗体，并使用 `Paint` 类的 `setFakeBoldText` 方法设置字体为仿粗体。
 运行程序并在输入框中输入字符，单击“斜体”单选按钮，文本框视图将输入框的文本以斜体的方式显示，如图 6-15 所示。

### 6.3.2 引用用户自定义字体

在 Android 中，不仅可以通过 `android` 类提供的字体库设置字体，还可以通过自定义字体库设置字体。设置自定义的字体时，需要新建存储自定义字体的 TTF 文件。TTF 的全称是 TrueTypeFont（真实类型字体），这是一种文件格式。当前常用的字体文件格式有两种：`postscript` 和 `TTF`。`TTF` 文件格式是由苹果和微软两家公司共同推出的字体文件格式。相比其他字体文字格式，该格式还可将字体轮廓转换成曲线，并支持曲线的填充以及颜色和效果的制作。



不仅如此，该格式可以进一步变形，制作特殊效果字体，因此经常用来制作一些标题字或花样字。

下面是 Android 系统引用自定义字体的过程（如图 6-16 所示）：



图 6-15 改变字体

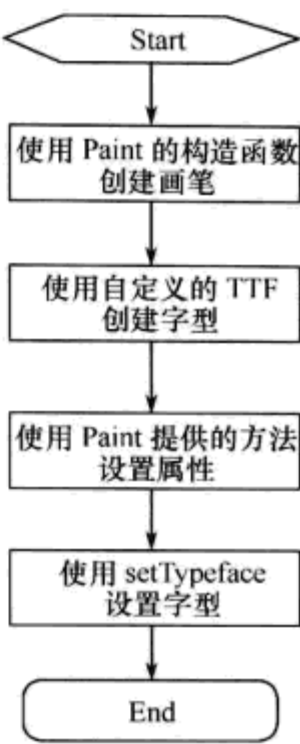


图 6-16 引用自定义字体过程

- （1）创建画笔。  
 例如，使用 Paint 的构造函数创建画笔对象：

```

Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);

```

- （2）第二步 使用自定义的 TTF 文件创建字型对象。  
 例如，使用 Typeface 的 createFromAsset 方法创建字型对象：

```

Typeface tf = Typeface.createFromAsset(getAssets()," Myfont.ttf");

```

- （3）设置画笔属性，如画笔大小、字体大小。

```

paint.setColor(0xff666666);
paint.setTextSize(30);

```

- （4）使用 setTypeface 设置字型。

```

paint.setTypeface(tf)

```

## 6.4 应用实例详解：制作手机桌面

至此，我们学习了大部分的 UI 相关的组件、布局以及字体等。这些组件在应用的设计中是不可缺少的，全面的 UI 知识是 Android 开发中必备的技能。下面以制作手机桌面为例讲述 UI 的完整设计过程。

### 6.4.1 实例分析

手机桌面是手机的门户，通常手机桌面会放置系统提供的应用程序以及应用服务。这些应用程序一般包含一个图标和一个名称。因此，一个桌面可以简单理解为一个存放应用程序图标的容器。图 6-17 是一个手机桌面的示意图。





图 6-17 手机桌面示意图

实现一个手机桌面应用，需要实现以下三个功能：

- 应用程序图标布局，即排放顺序。手机桌面一般采用网格组织结构。
- 应用程序图标素材的选择，这个主要和应用程序本身相关。一般来讲，一个形象的应用程序图标可以在某种程度上提高认可度。例如，电子邮件应用程序采用信封作为图标，这样不仅一目了然，还可提高认可度。
- 监听器管理。需要为每个应用程序提供监听器，以便在触摸或者单击的时候打开该应用程序。

对于应用程序图标的布局设计，本实例采用多级线性布局实现网格结构：

- 元素级。
- 行级。
- 桌面级。

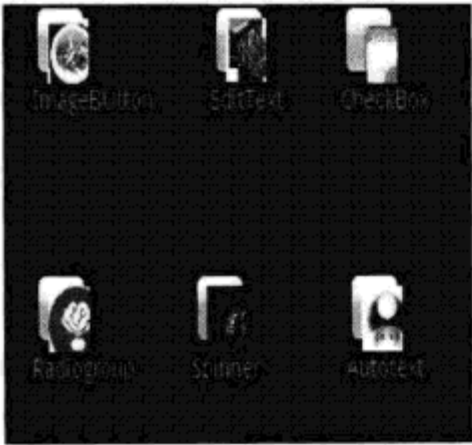


图 6-18 元素级布局

一个桌面级布局包含其所有的应用程序图标及标题，桌面级布局包含多行行级布局。对于手机桌面的每一行使用一个线性布局，这个线性布局是水平排列的。而一行线性布局又包含若干个元素级的线性布局。每个元素级的线性布局对应一个应用程序的图标以及标题。为了实现图标下面显示标题的效果，元素级布局采用垂直排列方式。图 6-18 所示是一个元素级的布局。

为了实现单击图标启动对应的应用程序，使用图片按钮包含相应的图标。每个图片按钮，都被注册单击事件监听。本实例实现了一个简单的桌面，单击图标时跳转到相应的程序执行。

6.4.2 实例实现

【实例 6-4】制作手机桌面。

布局实现，采用多级线性布局实现网格结构：

```

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

```



```

<!-- 线性布局垂直分布，高度和宽度同父元素相同 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

    <!-- 桌面第一行的应用，包含 3 个 ImageButton，高度和宽度与父元素相同，水平排列 -->
    <LinearLayout
        android:id="@+id/linearLayout1"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
        >

        <TextView
            android:layout_width="20px"
            android:layout_height="40px"
            />

        <!-- ImageButton Application -->
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="wrap_content"
            android:layout_height="70px"
            >

            <ImageButton
                android:id="@+id/imageButton11"
                android:layout_width="40px"
                android:layout_height="40px"
                android:focusableInTouchMode="true"
                android:src="@drawable/icon1"
                />

            <TextView
                android:text="ImageBUtton"
                android:layout_width="wrap_content"
                android:layout_height="20px"
                />

        </LinearLayout>

        <TextView
            android:layout_width="40px"

```



```

        android:layout_height="40px"
    />
    <!-- EditText Application -->
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="70px"
    >
        <ImageButton
            android:id="@+id/imageButton12"
            android:layout_width="40px"
            android:layout_height="40px"
            android:focusableInTouchMode="true"
            android:src="@drawable/icon2"
        />
        <TextView
            android:text="EditText"
            android:layout_width="wrap_content"
            android:layout_height="20px"
        />
    </LinearLayout>
    <TextView
        android:layout_width="40px"
        android:layout_height="40px"
    />
    <!-- CheckBox Application -->
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="70px"
    >

        <ImageButton
            android:id="@+id/imageButton13"
            android:layout_width="40px"
            android:layout_height="40px"
            android:focusableInTouchMode="true"
            android:src="@drawable/icon3"
        />
        <TextView

```



```

        android:text="CheckBox"
        android:layout_width="wrap_content"
        android:layout_height="20px"
    />
</LinearLayout>
</LinearLayout>

<!-- 桌面第二行的应用，包含 3 个 ImageButton，高度和宽度与父元素相同，水平排列 -->
<LinearLayout
    android:id="@+id/linearLayout1"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    >
    <TextView
        android:layout_width="20px"
        android:layout_height="40px"
    />
    <!-- Radiogroup Application -->
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="70px"
    >

        <ImageButton
            android:id="@+id/imageButton21"
            android:layout_width="40px"
            android:layout_height="40px"
            android:focusableInTouchMode="true"
            android:src="@drawable/icon9"
        />
        <TextView
            android:text="Radiogroup"
            android:layout_width="wrap_content"
            android:layout_height="20px"
        />
    </LinearLayout>
    <TextView

```



```

        android:layout_width="40px"
        android:layout_height="40px"
    />

    <!-- Spinner Application -->
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="70px"
    >

        <ImageButton
            android:id="@+id/imageButton22"
            android:layout_width="40px"
            android:layout_height="40px"
            android:focusableInTouchMode="true"
            android:src="@drawable/icon8"
        />

        <TextView
            android:text="Spinner"
            android:layout_width="wrap_content"
            android:layout_height="20px"
        />
    </LinearLayout>

    <TextView
        android:layout_width="60px"
        android:layout_height="40px"
    />

    <!-- Autotext Application -->
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="70px"
    >

        <ImageButton
            android:id="@+id/imageButton23"
            android:layout_width="40px"
            android:layout_height="40px"
            android:focusableInTouchMode="true"
            android:src="@drawable/icon6"

```



```

        />
        <TextView
            android:text="Autotext"
            android:layout_width="wrap_content"
            android:layout_height="20px"
        />
    </LinearLayout>
</LinearLayout>
</LinearLayout>

```

主程序主要实现了控件监听器的注册以及监听事件发生时的处理：

```

import android.app.Activity;    //导入活动类
import android.content.Intent;  //导入意图类
import android.os.Bundle;       //导入 Bundle 类
import android.view.View;       //导入视图类
import android.view.View.OnClickListener; //导入单击事件监听器
import android.widget.ImageButton; //导入图片按钮类
import android.widget.Toast;     //导入 Toast 消息

public class MyDesktop extends Activity {
    /** Called when the activity is first created. */

    private ImageButton IB_app1;
    private ImageButton IB_app2;
    private ImageButton IB_app3;
    private ImageButton IB_app4;
    private ImageButton IB_app5;
    private ImageButton IB_app6;
    /*声明图片按钮控件 IB_app1、IB_app2、IB_app3、IB_app4、IB_app5、IB_app6*/
    private Intent Intent_app1;
    private Intent Intent_app2;
    private Intent Intent_app3;
    private Intent Intent_app4;
    private Intent Intent_app5;
    private Intent Intent_app6;
    /*声明意图 Intent_app1、Intent_app2、Intent_app3、Intent_app4、Intent_app5、Intent_app6*/
    public void onCreate(Bundle savedInstanceState) {
        try
        {
            super.onCreate(savedInstanceState);
            init(); //初始化控件

```



```

        setContentView(R.layout.main);
        createComponent(); //使用 createComponent 方法创建组件
        register(); //注册桌面图标的监听器
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(MyDesktop.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
/*程序入口点*/

private void init()
{
    IB_app1 = null ;
    IB_app2 = null ;
    IB_app3 = null ;
    IB_app4 = null ;
    IB_app5 = null ;
    IB_app6 = null ;
    Intent_app1 = null ;
    Intent_app2 = null ;
    Intent_app3 = null ;
    Intent_app4 = null ;
    Intent_app5 = null ;
    Intent_app6 = null ;
}
/*初始化控件和意图*/

private void createComponent()
{
    IB_app1 = (ImageButton) this.findViewById(R.id.imageButton11);
    IB_app2 = (ImageButton) this.findViewById(R.id.imageButton12);
    IB_app3 = (ImageButton) this.findViewById(R.id.imageButton13);

```



```

IB_app4 = (ImageButton) this.findViewById(R.id.imageButton21);
IB_app5 = (ImageButton) this.findViewById(R.id.imageButton22);
IB_app6 = (ImageButton) this.findViewById(R.id.imageButton23);
}
/*createComponent 方法创建程序组件*/

private void register()
{
    IB_app1.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app1 = new Intent(MyDesktop.this,Ex43.class); //创建 Intent
            startActivity(Intent_app1); //启动 Intent 指定的应用
        }
    }); //注册 IB_app1 单击事件监听器

    IB_app2.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app2 = new Intent(MyDesktop.this,Ex44.class); //创建 Intent
            startActivity(Intent_app2); //启动 Intent 指定的应用
        }
    }); //注册 IB_app12 单击事件监听器

    IB_app3.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app3 = new Intent(MyDesktop.this,Ex45.class); //创建 Intent
            startActivity(Intent_app3); //启动 Intent 指定的应用
        }
    }); //注册 IB_app12 单击事件监听器

    IB_app4.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app4 = new Intent(MyDesktop.this,Ex46.class); //创建 Intent
            startActivity(Intent_app4); //启动 Intent 指定的应用
        }
    }); //注册 IB_app14 单击事件监听器

    IB_app5.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app5 = new Intent(MyDesktop.this,Ex47.class); //创建 Intent
            startActivity(Intent_app5); //启动 Intent 指定的应用
        }
    });
}

```



```

    }
    }); //注册 IB_app15 单击事件监听器

    IB_app6.setOnClickListener(new OnClickListener(){
        public void onClick(View v) {
            Intent_app6 = new Intent(MyDesktop.this,Ex48.class); //创建 Intent
            startActivity(Intent_app6); //启动 Intent 指定的应用
        }
    }); //注册 IB_app16 单击事件监听器
}
/*register 方法注册单击事件监听器*/
}

```

【代码说明】本实例使用图片按钮以及线性布局实现了一个简单的手机桌面。当用户单击图标时，对应的单击监听方法被调用。线性布局用来组织桌面上的图标，本实例使用多级线性布局实现网格结构的桌面。





## 第 7 章 Android 中的核心 Intent

大多数手机应用程序之间相互独立、相互隔离，应用程序与硬件和原始组件之间没有交互的行为，然而交互是手机应用功能扩展必需的因素，有了交互，手机才能支持复杂应用的开发。在当前的应用需求下，不支持应用交互的手机是没有意义的。鉴于此，Android 系统提供了用于开发应用程序交互功能的组件，这些组件包括 Broadcast Receivers、Intent、Adapters、Content Providers。

Intent 是一种利用消息进行交互的机制。Intent 对象描述了应用中一次操作的动作、动作涉及的数据和附加数据，系统通过该对象的描述调用对应的应用。调用的应用可以是一个应用程序，也可以是一个 Activity 或者 Service。在 Android 系统中，Intent 类继承 Object 类的属性和方法，派生 LabeledIntent 类。Intent 的派生关系如下所示：

```

java.lang.Object
  android.content.Intent
    android.content.pm.LabeledIntent
    
```

本章将讲述 Intent 的作用，以及 Intent 的两种分类(Action Intent 和 Broadcast Intent)、Activity 许可设置方法，最后以一个电话拨号程序实例讲解 Intent 的用法。

### 7.1 Intent 的作用

Intent 提供了应用程序之间的交互机制，Intent 负责对应用中一次操作的动作、动作涉及的数据、附加数据进行描述，Android 则根据此 Intent 的描述，负责找到对应的组件，将 Intent 传递给调用的组件，并完成组件的调用。Intent 消息是一种同一或不同应用程序中的组件之间延迟运行时绑定的机制。Android 应用程序的三个核心组件是 Activity（活动）、Services（服务）、Broadcast Receiver（广播接收器）。Intent 消息描述了操作的抽象，这个抽象被描述成将要执行的操作数据结构。对于 Activity（活动）、Services（服务）、Broadcast Receiver（广播接收器）这三个组件，Intent 组件有不同的处理方式（详见表 7-1）。

表 7-1 不同组件的 Intent 处理方式

核心组件	调用方法	作用
Activity	Context.startActivity()	启动一个 Activity 或使一个已存在的 Activity 去做新的工作
	Activity.startActivityForResult()	
Services	Context.startService()	初始化一个 Service 或传递一个新的操作给当前正在运行的 Service
Broadcast Receiver	Context.sendBroadcast()	对所有想接受消息的 Broadcast Receiver 传递消息
	Context.sendOrderedBroadcast()	
	Context.sendStickyBroadcast()	



### 7.1.1 多 Activity 的 Android 应用

Intent 对象描述了应用中一次操作的动作、动作涉及的数据和附加数据，系统通过 Intent 对象的描述调用对应的应用。它提供了多个 Activity 之间进行交互的方式，应用程序可通过 startActivity 方法指定相应的 Intent 对象来启动另外一个 Activity。下面以一个选择操作系统类型的实例讲述多 Activity 的 Android 应用。

【实例 7-1】多 Activity 的 Android 应用。

第一个 Activity 代码：

/\*Ex\_71\_1.java 代码，该类为主 Activity\*/

```
package test.Ex_71;
```

```
import android.app.Activity;
```

```
import android.os.Bundle;
```

```
import android.content.Intent; //导入 Intent 包
```

```
import android.view.View;
```

```
import android.widget.Button;
```

```
import android.view.View.OnClickListener; //导入单选监听器包
```

```
import android.widget.RadioButton; //导入单选按钮包
```

```
import android.widget.RadioGroup; //导入单选按钮组包
```

```
public class Ex_71_1 extends Activity {
```

```
    /** Called when the activity is first created. */
```

```
    @Override
```

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

```
        RadioGroup RG_OS;
```

```
        RadioButton RG_OS_RB1, RG_OS_RB2, RG_OS_RB3;
```

```
        Button button_submit, button_back;
```

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

```
        setContentView(R.layout.activity1); //根据布局文件 activity1.xml 生成程序界面
```

```
        /*根据 XML 定义生成 RadioGroup、RadioButton、Button 对象*/
```

```
        RG_OS = (RadioGroup) findViewById(R.id.RG_OS);
```

```
        /*生成 RadioGroup 对象，该组包含 3 个单选按钮：RG_OS_RB1, RG_OS_RB2 和 RG_OS_RB3。单选按钮对应关系如下：
```

```
        *                RG_OS_RB1 Andirod
```

```
        *                RG_OS_RB2 Symbian
```

```
        *                RG_OS_RB2 Other
```

```
        */
```

```
        RG_OS_RB1 = (RadioButton) findViewById(R.id.RG_OS_RB1); //生成第一个单选按钮对象
```

```
        RG_OS_RB2 = (RadioButton) findViewById(R.id.RG_OS_RB2); //生成第二个单选按钮对象
```

```
        RG_OS_RB3 = (RadioButton) findViewById(R.id.RG_OS_RB3); //生成第三个单选按钮对象
```

```
        button_submit = (Button) findViewById(R.id.button_submit); //生成按钮对象
```



```

        button_submit.setOnClickListener(new ButtonClickListener());
        /*使用 setOnClickListener 注册单选按钮单击事件监听器*/
    }

    /*定义按钮 button_submit 单击事件监听器。当单击 button_submit 时，onClick 方法被调用*/
    class ButtonClickListener implements OnClickListener{
        public void onClick(View arg0) {
            Intent myintent = new Intent();
            myintent.setClass(Ex_71.this, Ex_71_2.class);
            /*新建一个 Intent 对象，并指定启动程序 Ex_71*/
            Ex_71.this.startActivity(myintent);
            /*程序 Ex_71_2 利用 startActivity 调用新的 Activity，这个 Activity 是由 setClass 方法指定的*/
            Ex_71.this.finish();//关闭当前的 Activity
        }
    }
}

```

第二个 Activity 代码:

```

/*Ex_71_2.java 代码，该 Activity 被 Ex_71_1 的 Activity 调用*/
package test.Ex_71;
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent; //导入 Intent 包
import android.view.View;
import android.widget.Button;
import android.view.View.OnClickListener; //导入单选监听器包

public class Ex_71_2 extends Activity {
    Button button_back;//定义 Button 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity2);//根据布局文件 activity2.xml 生成程序界面
        button_back = (Button) findViewById(R.id.button_back);//生成按钮对象
        button_back.setOnClickListener(new ButtonClickListener());
        /*使用 setOnClickListener 注册单选按钮单击事件监听器*/
    }

    /*定义按钮 button_submit 单击事件监听器。当单击 button_submit 时，onClick 方法被调用*/
    class ButtonClickListener implements OnClickListener{
        public void onClick(View arg0) {
            Intent myintent = new Intent();

```



```

        myintent.setClass(Ex_71_2.this, Ex_71.class);
        /*新建一个 Intent 对象, 并指定启动程序 Ex_71*/
        Ex_71_2.this.startActivity(myintent);
        /*程序 Ex_71_2 利用 startActivity 调用新的 Activity, 这个 Activity 是由 setClass 方法指定的*/
        Ex_71_2.this.finish(); //关闭当前的 Activity
    }
}
}

```

布局 activity1.xml 定义:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="First Activity"
        />
    <!-- 创建一个选择操作系统的 RadioGroup, 该组包含 3 个单选按钮 -->
    <RadioGroup
        android:id="@+id/RG_OS"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="选择操作系统类型"
        >
        <!-- 第一个 RadioButton -->
        <RadioButton
            android:id="@+id/RG_OS_RB1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Android"
            />
        <!-- 第二个 RadioButton -->
        <RadioButton
            android:id="@+id/RG_OS_RB2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

```



```

        android:text="Symbian"
    />
    <!-- 第三个 RadioButton -->
    <RadioButton
        android:id="@+id/RG_OS_RB3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Other"
    />
    <Button
        android:id="@+id/button_submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit"
    />
</RadioGroup>
</LinearLayout>

```

【代码说明】 本实例定义了两个 Activity: Ex\_71\_1 和 Ex\_71\_2, 两个 Activity 之间使用 Intent 实现相互调用。Ex\_71\_1 使用 activity1.xml 生成程序界面, 该布局包含一个 RadioGroup 和一个 Button 组件。RadioGroup 提供了选择操作系统的单选按钮, Button 负责生成单击事件的监听器。当该按钮的单击事件发生时, 程序会使用 Intent 对象调用另外一个 Activity。

首先 Intent 的构造函数会产生一个 Intent 对象 myintent, 该对象利用自身的方法 setClass 设置要调用的 Activity。setClass 的原型为:

```
setClass(Context packageContext, Class<?> cls)
```

setClass 方法包含两个参数:

- Context packageContext 为当前 Activity, 例如 Ex\_71\_2.this。
- Class<?> cls 为被调用的 Activity, 如 Ex\_71\_2.class。注意, 后缀 class 不能省略, 因为 Activity Ex\_71\_2 是在 Ex\_71\_2.java 中定义的, 需要指定其可执行文件名字才能调用 Activity Ex\_71\_2。

然后通过 startActivity(myintent)启动 myintent 中指定的 Activity, 当前的 Activity Ex\_71\_1 调用 finish 方法关闭。这时程序的控制权就转给了 Activity Ex\_71\_2。

Ex\_71\_2 使用 activity2.xml 生成程序界面, 该布局包含一个 Button 组件。单击该按钮时, 程序会使用 Intent 对象调用 Activity Ex\_71\_1。这时程序的控制权又返回给 Activity Ex\_71\_2, 因此该按钮实现了类似于“界面后退”的功能。

请注意, 在使用 Eclipse 创建 Android 工程时, 系统在 AndroidManifest.xml 中自动生成了主 Activity Ex\_71\_1 的定义, 没有生成关于 Activity Ex\_71\_2 的定义, 需要在 AndroidManifest.xml 中添加 Activity Ex\_71\_2 的代码, 否则运行时, 系统会因找不到 Ex\_71\_2 而出现异常终止的错误, 如图 7-1 所示。

```

<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Ex_71_1"

```



```

        android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".Ex_71_2" android:label="@string/app_name"/>
</application>

```



图 7-1 找不到 Activity Ex\_71\_2 异常终止错误

最后启动该 Android 程序，单击 Ex\_71\_1 的提交按钮（图 7-2 左图所示），这时手机界面切换到另外一个 Activity 的界面（图 7-2 右图所示）。至此，实现了多 Activity 的 Android 应用，该应用通过 Intent 对象来启动另外一个 Activity。

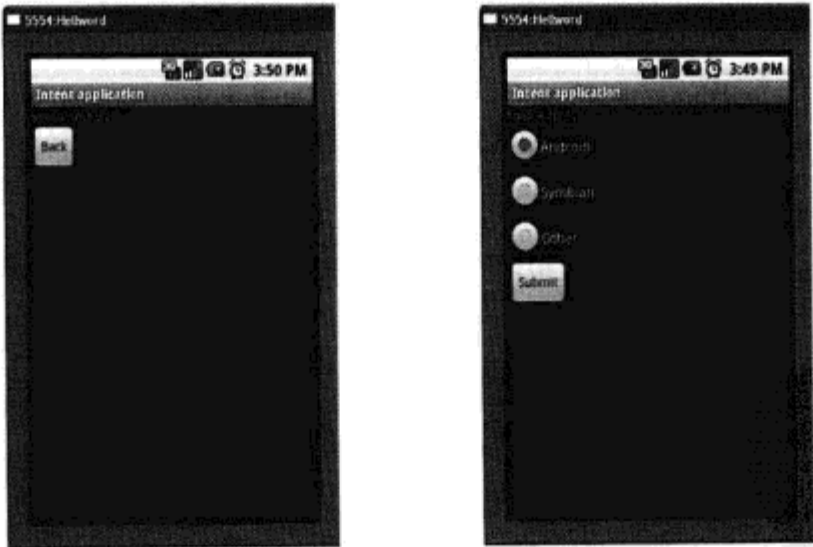


图 7-2 实例 7-1-1 运行结果

7.1.2 Activity 之间的消息传递

如果 Activity 之间需要传递数据，Intent 可利用 Bundle 组件实现。Bundle 对象可被看成一个哈希表，该映射表建立了将关键字（标识）同其值（传递的数据）的映射关系，可通过 Bundle 类的 putXXX(Key, Value)方法将数据封装到 Bundle 对象中，如 putString(String key, String value)。putXXX 方法的第一参数是传递数据的关键字，可通过 getXXX(String key)取得关键字对应的数



据。表 7-2 列举了 Bundle 类常用的方法。

表 7-2 Bundle 类常用的方法

方法	功能描述	返回值
Bundle	Bundle 类的构造方法，Bundle 类提供了四个构造函数	null
get(String key)	获取关键字 key 对应的数据，返回值为一个 Object 对象	Object
getBoolean(String key)	获取关键字 key 对应的布尔值，若找不到关键字的记录，则返回 false	Boolean
getBoolean(String key, boolean defaultValue)	获取关键字 key 对应的布尔值，若找不到关键字的记录，则返回 defaultValue	Boolean
getBundle(String key)	获取关键字 key 对应的 Bundle 对象，若找不到关键字的记录，则返回 null。Bundle 注意，Bundle 对象可包含一个 Bundle 对象的映射关系，即 Bundle 对象可嵌套包含	
getChar(String key)	获取关键字 key 对应的 char 值，若找不到关键字的记录，则返回 0	char
getChar(String key, char defaultValue)	获取关键字 key 对应的 char 值，若找不到关键字的记录，则返回 defaultValue	char
hasFileDescriptors	Bundle 对象是否包含文件描述符，返回值 true 表示 Bundle 对象包含文件描述符，false 表示 Bundle 对象不包含文件描述符	Boolean
putAll(Bundle map)	插入 map 到该 Bundle 对象中	void
putBoolean(String key, boolean value)	插入布尔值 value 到该 Bundle 对象中，若关键字 key 已存在，则原有值被 value 替代。Bundle 对象通过 key 关键字获取该值	void
putBundle(String key, Bundle value)	插入 Bundle 对象 value 到该 Bundle 对象中，若关键字 key 已存在，则原有值被 value 替代。Bundle 对象通过 key 关键字获取该值	void
putByte(String key, byte value)	插入字节值 value 到该 Bundle 对象中，若关键字 key 已存在，则原有值被 value 替代。Bundle 对象通过 key 关键字获取该值	void
remove(String key)	移除该 Bundle 对象关键字为 key 的记录	void
size()	获取 Bundle 对象的关键字个数	int

下面将详细讲述如何通过 Intent 和 Bundle 在 Activity 之间进行消息传递。

【实例 7-2】Activity 之间的消息传递。

第一个 Activity 代码：

```

/*Ex_71_2_1.java 代码，该类为主 Activity*/
package test.Ex_71_2;
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent; //导入 Intent 包
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.view.View.OnClickListener; //导入单选监听器包
import android.widget.RadioButton; //导入单选按钮包
    
```



```
import android.widget.RadioGroup; //导入单选按钮组包
```

```
public class Ex_71_2_1 extends Activity {
```

```
    RadioGroup RG_OS;
```

```
    RadioButton RG_OS_RB1, RG_OS_RB2, RG_OS_RB3;
```

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

```
        Button button_submit, button_back;
```

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

```
        setContentView(R.layout.first_activity); //根据布局文件 first_activity.xml 生成程序界面
```

```
        /*根据 XML 定义生成 RadioGroup、RadioButton、Button 对象*/
```

```
        RG_OS = (RadioGroup) findViewById(R.id.RG_OS);
```

```
        /*生成 RadioGroup 对象，该组包含 3 个单选按钮：RG_OS_RB1, RG_OS_RB2 和 RG_OS_RB3。单选按钮对应关系如下：
```

```
        *                RG_OS_RB1 Andirod
```

```
        *                RG_OS_RB2 Symbian
```

```
        *                RG_OS_RB2 Other
```

```
        */
```

```
        RG_OS_RB1 = (RadioButton) findViewById(R.id.RG_OS_RB1); //生成第一个单选按钮对象
```

```
        RG_OS_RB2 = (RadioButton) findViewById(R.id.RG_OS_RB2); //生成第二个单选按钮对象
```

```
        RG_OS_RB3 = (RadioButton) findViewById(R.id.RG_OS_RB3); //生成第三个单选按钮对象
```

```
        button_submit = (Button) findViewById(R.id.button_submit); //生成单击按钮对象
```

```
        button_submit.setOnClickListener(new ButtonClickListener());
```

```
        /*使用 setOnClickListener 注册单选按钮单击事件监听器*/
```

```
    }
```

```
    /*定义按钮 button_submit 单击事件监听器。当单击 button_submit 时，onClick 方法被调用*/
```

```
    class ButtonClickListener implements OnClickListener{
```

```
        public void onClick(View arg0) {
```

```
            Intent myintent = new Intent();
```

```
            myintent.setClass(Ex_71_2_1.this, Ex_71_2_2.class);
```

```
            /*新建一个 Intent 对象，并指定启动程序 Ex_71_2_2*/
```

```
            Bundle mybundle = new Bundle();
```

```
            /*创建 Bundle 对象，该对象用于记录被传送的数据*/
```

```
            if(RG_OS_RB1.isChecked())
```

```
                mybundle.putString("selected_radiobutton", (String) RG_OS_RB1.getText());
```

```
            else if (RG_OS_RB2.isChecked())
```

```
                mybundle.putString("selected_radiobutton", (String) RG_OS_RB2.getText());
```

```
            else if (RG_OS_RB3.isChecked())
```

```
                mybundle.putString("selected_radiobutton", (String) RG_OS_RB3.getText());
```

```
            else
```

```
                mybundle.putString("selected_radiobutton", "null");
```



```

        /*判断当前被选中的单选按钮，利用 putString 方法将单选按钮的名字存储到 mybundle 中*/
        myintent.putExtras(mybundle);
        /*将数据封装到 Intent 对象中，通过该 Intent 对象将数据传送给相应的 Activity*/
        Ex_71_2_1.this.startActivity(myintent);
        /*程序 Ex_71_2_1 利用 startActivity 调用新的 Activity，这个 Activity 是由 setClass 方法指定的*/
        Ex_71_2_1.this.finish();//关闭当前的 Activity
    }
}
}

```

第二个 Activity 代码:

```

/*Ex_71_2_2.java 代码，该类为辅助 Activity*/
public class Ex_71_2_2 extends Activity {
    Button button_back;//定义 Button 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_activity);//根据布局文件 second_activity.xml 生成程序界面
        button_back = (Button) findViewById(R.id.button_back);//生成按钮对象
        TextView textview = (TextView) findViewById(R.id.textview);//生成文本框视图对象
        button_back.setOnClickListener(new ButtonClickListener());
        /*使用 setOnClickListener 注册单选按钮单击事件监听器*/
        Intent myintent = this getIntent();//获取 Activity 传递的 Intent
        Bundle mybundle = myintent.getExtras();//获取 Intent 的 Bundle 对象，该对象记录了传送的数据
        String selected_radiobutton = mybundle.getString("selected_radiobutton");
        if (selected_radiobutton == "null")
            textview.setText("Not selected any OS");
        else
            textview.setText(selected_radiobutton+" is selected");
    }
    /*定义按钮 button_submit 单击事件监听器。当单击 button_submit 时，onClick 方法被调用*/
    class ButtonClickListener implements OnClickListener{
        public void onClick(View arg0) {
            Intent myintent = new Intent();
            myintent.setClass(Ex_71_2_2.this, Ex_71_2_1.class);
            /*新建一个 Intent 对象，并指定启动程序 Ex_71_2_1*/
            Ex_71_2_2.this.startActivity(myintent);
            /*程序 Ex_71_2_2 利用 startActivity 调用新的 Activity，这个 Activity 是由 setClass 方法指定的*/
            Ex_71_2_2.this.finish();//关闭当前的 Activity
        }
    }
}

```



}

布局 first\_activity.xml 定义:

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="First Activity"
    />
<!--创建一个选择操作系统的 RadioGroup , 该组包含 3 个单选按钮-->
<RadioGroup
    android:id="@+id/RG_OS"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="选择操作系统类型"
    android:layout_x="100px"
    android:layout_y="70px"
    >
<!-- 第一个 RadioButton -->
<RadioButton
    android:id="@+id/RG_OS_RB1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Android"
    />
<!-- 第二个 RadioButton -->
<RadioButton
    android:id="@+id/RG_OS_RB2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Symbian"
    />
<!-- 第三个 RadioButton -->
<RadioButton
    android:id="@+id/RG_OS_RB3"
```



```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Other"
    />
</RadioGroup>

<Button
    android:id="@+id/button_submit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="100px"
    android:layout_y="200px"
    android:text="Submit"
/>
</AbsoluteLayout>

```

【代码说明】本实例定义了两个 Activity: Ex\_71\_2\_1 和 Ex\_71\_2\_2, Ex\_71\_2\_1 通过 Bundle 绑定单选按钮值, 将当前被选中的单选按钮值传送给另外一个 Activity: Ex\_71\_2\_2。这样两个 Activity 之间使用 Intent 和 Bundle 不仅可以实现相互调用, 还能进行数据传递。Ex\_71\_2\_1 包含一个 RadioGroup 和一个 Button 组件, 单击按钮后, 通过 Bundle 的 putString 方法将被选中的单选按钮值封装到该 Bundle 对象中。例如, 选中 RG\_OS\_RB2 并单击按钮, 该单选按钮的值 (RG\_OS\_RB2.getText()) 被封装到 Bundle 对象中, 该值的关键字为 selected\_radiobutton:

```
putString("selected_radiobutton", (String) RG_OS_RB2.getText());
```

然后 Intent 对象通过调用 putExtras 方法将 Bundle 对象捆绑到 Intent 对象。Intent 类包含 putExtras 方法:

- public Intent putExtras (Bundle extras): 将 Bundle 对象 extras 复制到 Intent 对象中。
- public Intent putExtras (Intent src): 将 Intent 对象 src 复制到 Intent 对象中。

本实例使用 Intent putExtras (Intent src) 方法将 Bundle 对象 mybundle 封装到 Intent 对象中, 对象 mybundle 包含了当前被选中的单选按钮的名称。接着程序利用 startActivity 将控制权从 Ex\_71\_2\_1 转到 Ex\_71\_2\_2。

Ex\_71\_2\_2 包含一个 TextView 和一个 Button 组件, 布局方式为 AbsoluteLayout, 布局中的组件可通过指定在 Android 坐标系统中的绝对位置实现定位。Android 坐标系统是一个二维的坐标系统, 需要通过两个参数 layout\_x 和 layout\_y 确定绝对位置, 横向位置通过 layout\_x 指定, 纵向位置通过 layout\_y 指定。

```

<!--second activity 代码-->
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
>

```

```
<!--在水平 80px, 垂直 20px 的位置上创建 TextView, 用于显示当前被选中的操作系统-->
```



```
<TextView
    android:id="@+id/textview"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_x="80px"
    android:layout_y="20px"
/>
<!--在水平 100px, 垂直 100px 的位置上创建 Button, 用于返回到主 Activity-->
<Button
    android:id="@+id/button_back"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="100px"
    android:layout_y="100px"
    android:text="Back"
/>
</AbsoluteLayout>
```

Ex\_71\_2\_1 调用 Ex\_71\_2\_2 后, Ex\_71\_2\_1 传递给 Ex\_71\_2\_2 一个 Intent 对象并将控制权转给 Ex\_71\_2\_2。这时 Ex\_71\_2\_2 需要获取该 Intent 对象才能读取 Bundle 对象, Bundle 对象包含了传递的 String 类型数据。因此 Ex\_71\_2\_2 通过 getIntent 方法获取从 Ex\_71\_2\_1 传过来的 Intent 对象。接着该 Intent 对象使用 getExtras 方法得到封装单选按钮值的 Bundle 对象 mybundle, 对象 mybundle 调用 getString 方法获取 selected\_radiobutton 对应的值。

至此, 通过 Bundle 和 Intent 组件实现了 Activity 之间的消息传递。启动该 Android 程序, 单击 Ex\_71\_2\_1 的提交按钮(图 7-3 左图所示), 这时手机界面切换到另外一个 Activity 的界面, 该界面显示前一个界面被选中的单选按钮值(图 7-3 右图所示)。

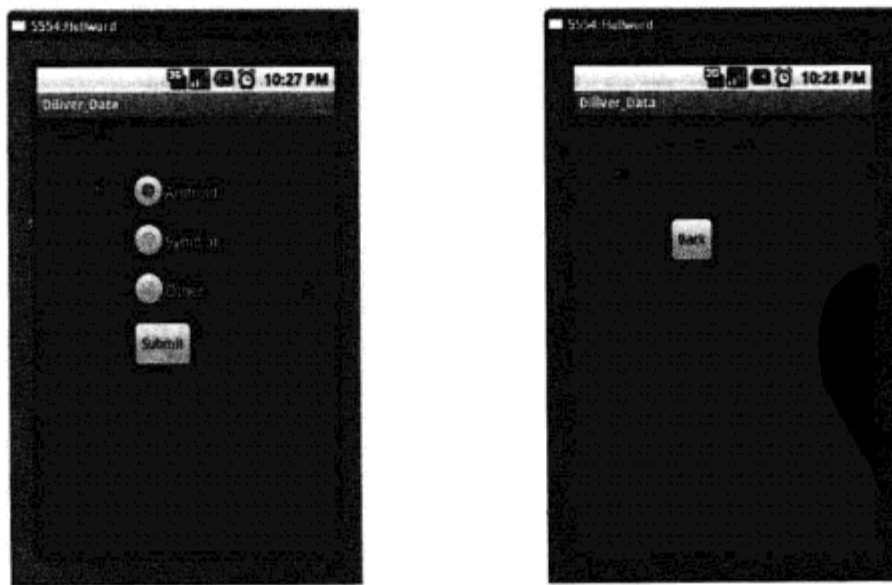


图 7-3 实例 7-1-2 运行结果

## 7.2 Intent 的分类

Intent 是一种同一或不同应用程序中的组件之间互相调用的机制。一般来讲, Android 根据



Intent 的描述找到对应的组件，并将 Intent 传递给调用的组件。可通过 startActivity 启动一个活动，利用 broadcastIntent 把 Intent 对象广播给所有插入的 BroadcastReceiver 组件。

按照 Intent 的处理方式，Intent 可以分为显式 Intent 和隐式 Intent 两类。显式 Intent 通过名字指定被调用的应用（Activity/Service），如 Intent("android.intent.action.CALL",Uri.parse("tel:"+string))。显式 Intent 通常用于应用程序中多个 Activity 之间的交互。因为开发者通常不知道其他应用程序的组件名字，显式 Intent 通常用于应用程序内部消息，如一个活动启动从属的服务或启动一个姐妹活动。显式方式通常被称为 Action Intent。

相对于显式 Intent，Android 还提供了隐式 Intent。这种方式并不指定目标应用的名字。为了找到合适的目标应用，隐式方式需要广播 Intent 消息。隐式方式通常被称为 BroadcastIntent。

7.2.1 Action Intent

Intent 用来呼叫应用程序以外的活动，只有一个活动可以处理 Intent。例如，对于网页浏览器，需要打开网页浏览器活动来显示一个页面。Intent 类定义了一些动作常量，如表 7-3 所示。

表 7-3 动作常量

Action 常量	行为描述	使用组件
ACTION_CALL	发起一个电话应用	activity
ACTION_EDIT	显示数据以供用户编辑	activity
ACTION_MAIN	初始化操作，这个操作既没有输入也没有输出	Boolean
ACTION_SYNC	同步服务器与移动设备之间的数据	Boolean
ACTION_BATTERY_LOW	警告设备电量低	Broadcast Receiver
ACTION_HEADSET_PLUG	插入或者拔出耳机	Broadcast Receiver
ACTION_SCREEN_ON	打开移动设备屏幕	Broadcast Receiver
ACTION_TIMEZONE_CHANGED	移动设备时区发生变化	Broadcast Receiver

其他的动作定义在 Android API 中，我们还可以定义自己的动作字符串以在我们的应用程序中激活组件。自定义动作字符串应该包含应用程序包名前缀，如“com.example.project.SHOW\_COLOR”。

动作很大程度上决定了剩下的 Intent 如何构建，特别是数据（data）和附加（extras）字段，就像一个方法名决定了参数和返回值。正是这个原因，应该尽可能明确指定动作，并紧密关联其他 Intent 字段。换句话说，应该定义组件能够处理的 Intent 对象的整个协议，而不仅仅是单独地定义一个动作。一个 Intent 对象的动作通过 setAction 方法设置，通过 getAction 方法读取。

不同的动作有不同的数据规格。例如，如果动作常量是 ACTION\_EDIT，数据（data）字段将包含用于编辑的文档的 URI；如果动作常量是 ACTION\_CALL，数据（data）字段将是一个 tel:URI 和将拨打的号码；如果动作常量是 ACTION\_VIEW，数据（data）字段将是一个 http:URI，接收活动将被调用去下载和显示 URI 指向的数据。

当匹配一个 Intent 到一个能够处理数据的组件时，通常需要知道数据的类型（它的 MIME 类型）和 URI。例如，一个组件能够显示图像数据，不应该被调用去播放一个音频文件。

在许多情况下，数据类型能够从 URI 中推测，特别是 content:URI，它表示位于设备上的数据被内容提供者（content provider）控制。但是类型也能够显式地设置，setData 方法用于指



定数据的 URI，setType 方法用于指定 MIME 类型，setDataAndType 方法用于指定数据的 URI 和 MIME 类型。通过 getData 方法可以读取 URI，getType 方法可以读取类型。

此外，可以在一个 Intent 对象中指定任意数量的种类描述。Intent 类定义的一些种类常量如下：

- Constant
- CATEGORY\_BROWSABLE
- CATEGORY\_GADGET
- CATEGORY\_HOME
- CATEGORY\_LAUNCHER
- CATEGORY\_PREFERENCE

addCategory 方法用于添加一个种类到 Intent 对象，removeCategory 方法用于删除一个之前添加的种类，getCategories 方法用于获取 Intent 对象中的所有种类。

### 7.2.2 Broadcast Intent

通过 7.2.1 小节，可以知道一个 Action Intent 只能指定一个 Activity 处理。如果 Intent 需要不止一个 Activity 处理，那么如何实现把 Intent 传递给多个 Activity。Andorid 提供了 Broadcast Intent 的机制来处理这种情况，这种机制可广播 Intent 到多个 Activity。例如，当手机的电量较低时，需要当前运行的活动都做出反应，就可通过 Broadcast Intent 机制实现。

Broadcast Intent 机制的实现包含四个步骤：第一步需要注册相应的 BroadcastReceiver，BroadcastReceiver 是接收广播消息并对消息做出反应的组件，如电量较低时的通知信息。第二步发送广播，这个过程将消息内容和用于过滤的信息封装起来，并广播给 BroadcastReceiver。第三步满足条件的 BroadcastReceiver 执行 onReceiver 方法，第四步执行 onReceiver 方法，销毁 BroadcastReceiver。具体流程如图 7-4 所示。

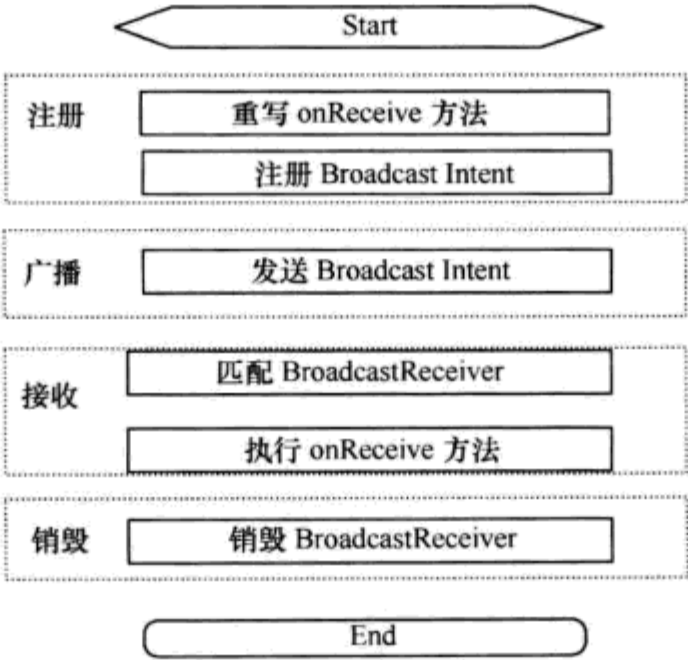


图 7-4 Broadcast Intent 机制的处理过程

第一步：注册。  
首先继承 BroadcastReceiver，并重写 onReceive 方法。例如：

```

public class MyReceiver extends BroadcastReceiver {
    @Override

```



```
public void onReceive(Context context, Intent intent) {
    /*添加 onReceive 代码处理*/
}
}
```

然后根据 IntentFilter 注册 Broadcast Intent。Android 提供了两种注册方法:Java 注册和 XML 注册。

#### (1) Java 注册

创建 IntentFilter 和 Receiver 对象,然后在需要的地方调用 Context.registerReceiver 进行注册。同样,可使用 Context.unregisterReceiver 取消注册:

```
IntentFilter myfilter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
MyReceiver myreceiver = new MyReceiver();
Context.registerReceiver(myreceiver, myfilter);
```

#### (2) XML 注册

在 AndroidManifest.xml 的 application 标签中,在 intent-filter 中添加如下行为:

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

第二步:广播。

有三种发送广播的方式,这三种方式是由 Context 类提供的:

- Context.sendBroadcast: 广播 Intent 到 BroadcastReceiver,满足条件的 BroadcastReceiver 都会执行 onReceive 方法。这种方式不严格保证执行顺序。
- Context.sendOrderedBroadcast: 广播 Intent 到 BroadcastReceiver,满足条件的 BroadcastReceiver 都会执行 onReceive 方法。这种方式保证执行顺序,根据 BroadcastReceiver 注册时 IntentFilter 设置的优先级的顺序来执行 onReceive 方法,高优先级的 BroadcastReceiver 执行先于低优先级的 BroadcastReceiver。
- Context.sendStickyBroadcast: 广播 Intent 到 BroadcastReceiver,满足条件的 BroadcastReceiver 都会执行 onReceive 方法。这种方式一直保存 sendSticky Broadcast 发送的 Intent,这样以后使用 registerReceiver 注册接收器时,新注册的接收器的 Intent 对象为该 Intent 对象。

第三步:接收。

BroadcastReceiver 收到广播 Intent,对 Intent 进行判断。如果该接收器满足条件,执行 onReceive 方法。

第四步:销毁。

在 Android 中,每次广播消息到来时都会创建 BroadcastReceiver 实例并执行 onReceive 方法, onReceive 方法执行完后, BroadcastReceiver 实例就会被销毁。执行 onReceive 方法时,Android 系统会启动一个程序计时器。如果在一定的时间内 onReceive 方法没有完成,会被认为该程序无响应,因此 onReceive 方法需要包含快速执行的逻辑,否则会弹出程序无响应 (Application No Response) 的对话框。



收到 Broadcast Intent 后, 所有包含相匹配的 IntentFilter 的 BroadcastReceiver 就会被激活。只有 BroadcastReceiver 才能接收 Broadcast Intent 消息, 然而 Activity 或 Service 不会接收 Broadcast Intent 消息。Activity 只接收由 startActivity 方法传递的消息, Service 只接收 startService 方法传递的消息。Broadcast Intent 机制被广泛运用于通知设备或系统的状态变化。例如, 当手机设备的电池电量低于一个阈值时, 系统会发送一个广播, 该广播的 Action 为 ACTION\_BATTERY\_LOW。收到该广播后, 所有包含相匹配的 IntentFilter 的 Broadcast Receiver 就会执行 onReceive 方法的处理代码, 比如进入节电模式。onReceive 方法的代码如下:

```
public void onReceive(Context mycontext, Intent myintent) {
    if (myintent.getAction().equals(Intent.ACTION_BATTERY_LOW)) {
        //添加低电量的处理, 如关闭 WiFi 和 GPS 以节电模式运行
    }
}
```

## 7.3 解析 Intent 的实现

### 7.3.1 Intent Receiver

Broadcast Intent 传递的消息需要 Receiver 接收, 在使用 Receiver 接收之前, 需要将其注册到系统中。Android 提供了 Java 和 XML 两种方式实现 Intent Receiver 注册。使用 Java 注册首先创建 IntentFilter 和 Receiver 对象, 然后在需要的地方调用 Context.registerReceiver 进行注册。同样, 可使用 Context.unregisterReceiver 取消注册。使用 XML 注册首先需要在 AndroidManifest.xml 的 application 的 receiver 标签中使用 android:name 属性指定接收器的名字, 然后在 intent-filter 中添加相应的行为、类别或者类型:

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

当 Intent Filter 的行为同广播的 Intent 匹配时, 系统执行该广播接收器的 onReceive 方法。需要注意, 应用程序会弹出应用无响应 (Application No Response) 的对话框。不需要在收到广播 Intent 之前启动 Broadcast Receiver, 该接收器会在匹配广播 Intent 的时候被激活。这种特殊的处理方式适合资源管理, 可通过这种方式创建关闭或销毁的事件驱动应用程序, 并以安全的方式对广播事件做出响应。Broadcast Receiver 会更新内容、启动服务、更新 Activity 的 UI 或使用通知管理器来通知用户。下面以一个具体例子说明 Intents Receiver 的基本用法。

【实例 7-3】Intent Receiver 应用。

Ex\_73.java 实现:

```
/*Broadcast Intent */
package test.Ex_73;
import android.app.Activity;
import java.io.File;
import android.content.Intent;
```



```
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.View;
import android.widget.Button;
/*导入程序中需要的包*/
public class Ex_73 extends Activity {
    public static final String My_NEW_LIFEFORM ="com.china.ui.NEW_LIFEFORM";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final Intent intent = new Intent(My_NEW_LIFEFORM);
        /*新建 Intent 对象, 指定启动应用为 com.china.ui.NEW_LIFEFORM
        Button button = (Button) findViewById(R.id.sendBroadcastIntent);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                sendBroadcast(intent);
                /*利用 sendBroadcast 方法广播 Intent 给广播接收器*/
            }
        }); //实现单击按钮事件处理, 当单击按钮时, onClick 方法被调用
    }
}
```

Receiver.java 实现:

```
/*负责接收广播 Intent*/
package test.Ex_73; //将该文件打包到 test.Ex_73 中
import android.content.BroadcastReceiver; //导入广播接收器类
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
import java.lang.CharSequence;
public class Receiver extends BroadcastReceiver {

    public void onReceive(Context context, Intent intent) {
        CharSequence string = "收到广播消息";
        Toast.makeText(context, string, Toast.LENGTH_LONG).show(); //显示 Toast 消息
    }
}
```

在 AndroidManifest.xml 中实现:

<!--在 AndroidManifest.xml 中添加该行为的 receiver 标签 -->



```
<receiver android:name="Receiver" android:enabled="true">
    <intent-filter>
        <action android:name="com.china.ui.NEW_LIFEFORM" />
    </intent-filter>
</receiver>
```

布局实现:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Sender"
/>

<Button
    android:id="@+id/sendBroadcastIntent"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Broadcast Intent"
/>
```

【代码说明】 为了实现 Intent 的广播和接收功能，需要在工程中至少包含两个 Java 文件：一个用于广播 Intent，另一个用于接收广播的 Intent。本实例包含 Ex\_73.java 和 Receiver.java 实现用于广播 Intent 和接收 Intent。Ex\_73.java 实现了广播 Intent 的功能，Receiver.java 实现了接收 Intent 的功能。Ex\_73.java 包含一个单选按钮，该单选按钮监听器的 onClick 方法设置了广播 Intent 的功能。单击单选按钮时，onClick 方法调用 sendBroadcast 发送载有行为 com.china.ui.NEW\_LIFEFORM 的 Intent。发送后，系统会匹配已注册的广播接收器，广播接收器可以通过 Java 注册也可以通过 XML 注册。本例采用 XML 的注册方式，因此系统在 AndroidManifest.xml 中匹配包含该 Intent 的接收器的名字。由于 android:name="Receiver" 的 receiver 标签包含 com.china.ui.NEW\_LIFEFOR，则 Receiver.java 的 Receiver 类被激活。该类继承 BroadcastReceiver 类，收到匹配的 Intent 后，该类的 onReceive 方法被调用执行。

程序运行结果如图 7-5 所示。



图 7-5 Intent Receiver 应用



7.3.2 Intent Filter

应用程序的核心组件（活动、服务和广播接收器）通过意图被激活。意图是描述期望动作的信息包（一个 Intent 对象），包括要操作的数据，执行该动作的组件类别，以及其他有关指令。Android 寻找一个合适的组件来响应这个意图，如果需要会启动这个组件，并传递给这个意图对象。

组件通过意图过滤器（Intent Filter）通告它们所具备的能力，即能响应的意图类型。由于 Android 系统在启动一个组件前必须知道该组件能够处理哪些意图，那么意图过滤器需要在 manifest 中以<intent-filter>标签指定。一个组件可以拥有多个过滤器，每一个描述不同的能力。

一个显式命名目标组件的意图将会激活那个组件，过滤器不起作用。但是一个没有指定目标的意图只在它能够通过组件过滤器过滤时才能激活该组件。

BroadcastReceiver 用于监听 Broadcast 的 Intent。为了激活一个 BroadcastReceiver，需要在代码或在程序中注册。当注册一个 BroadcastReceiver 时，必须使用 Intent Filter 来定义该接收器可接收的 Intent。在查找 BroadcastReceiver 时，程序可能未指定对应的处理该 Intent 的应用。在 Android 系统中，Intent Filter 被用来实现 Intent 的查找，本小节将讲述 Intent Filter 的功能。顾名思义，Intent Filter 提供了 Intent 过滤的功能。通过 Intent Filter，可以基于行为（Action）、类别（Category），以及数据（Data）进行 Intent 的过滤。Intent Filter 定义了接收 Intent 的能力，这种能力表示系统活动（Activity）、服务（Service）、广播接收者（BroadcastReceiver）等应用能够接收的 Intent，每个活动（Activity）、服务（Service）、广播接收者（BroadcastReceiver）可以有一个或多个 Intent Filter。例如，手机通讯簿活动有两个过滤器：一个用于启动一个指定的通讯录，用户可以查看和编辑；另一个用于建立一个新的通讯录，用户能够编辑并保存。

同其他组件一样，Android 提供了两种生成 Intent Filter 的方式。一种是通过 IntentFilter 类生成，另一种是通过在应用程序的清单文件（AndroidManifest.xml）中定义<intent-filter>生成。表 7-4 列举了 IntentFilter 类的常用方法。

表 7-4 IntentFilter 常用的方法

方法	功能描述	返回值
IntentFilter	提供了四个构造函数：  IntentFilter()、IntentFilter(String action)、IntentFilter(String action, String dataType)和 IntentFilter(IntentFilter o)	null
addAction(String action)	为 IntentFilter 添加匹配的行为。例如，添加电量低行为：addAction(ACTION_BATTERY_LOW)	void
addCategory(String category)	为 IntentFilter 添加匹配类别，如 addCategory(CATEGORY_LAUNCHER)	void
addDataAuthority(String host, String port)	获取 IntentFilter 的数据验证，如 addDataAuthority(myhost, 8888)host 参数可以包含通配符“*”，表示匹配任意字符，port 为空表示匹配任意端口	void
countActions()	计算 IntentFilter 包含的 Action 数量	int
countDataAuthorities()	计算 IntentFilter 包含的 DataAuthority 数量	int
getDataAuthority(int index)	根据 index 获取 IntentFilter 的 DataAuthority	IntentFilter.AuthorityEntry



续表

方法	功能描述	返回值
getAction(int index)	根据 index 获取 IntentFilter 的 Action	String
setPriority(int priority)	设置 IntentFilter 的优先级，默认优先级为 0。通常 priority 值介于 -1000~1000 之间。Android 系统根据优先级匹配 Intent	void
getPriority()	获取 IntentFilter 的优先级	int
hasCategory(String category)	判断 category 是否在 Intent 中，若包含返回 ture，否则返回 false	boolean
matchCategories(Set<String> categories)	基于 categories 匹配 IntentFilter，若匹配 IntentFilter 的所有类别返回 null，否则返回第一个不匹配的类别名字	String

IntentFilter 提供了 Intent 的过滤功能，因此需要知道一个 Intent 所包含的内容。通常构造一个 Intent，需要指定以下一个或多个部分：

- **Component**: 指定包名或者类名来调用，这些包或者类通常是用户自定的类或者包，例如 Ex\_71\_2\_2.class。
- **Action**: 指定 Intent 的行为，这些行为通常是由系统提供的，例如 android.provider.Telephony.SMS\_RECEIVED。
- **Data**: 指定 Intent 的数据，这个数据通常是一个 URI。一个 URI 包含 scheme、host、port、path 四个部分(scheme://host:port/path)，例如 http://com.example.myandroid: 8888/androidfolder。
- **Type**: 指定过滤的类型，例如图片类型 JPEG。
- **Category**: 指定过滤范围，例如 CATEGORY\_LAUNCHER、CATEGORY\_ALTERNATIVE。
- **Extras**: 指定 Intent 所带的数据，这个数据通过 Bundle 类生成。例如：

```
Bundle mybundle = new Bundle();
mybundle.putString("selected_radiobutton", (String) RG_OS_RB1.getText());
```

- **Flags**: 标志位，例如列出所有的应用：ACTION\_ALL\_APPS。

一个 IntentFilter 包含了 Intent 对象的动作、数据、种类的字段。过滤器要进行动作过滤、数据过滤以及种类过滤，然后 Android 系统会根据过滤结果将 Intent 传递给过滤后的应用。下面讲述每个过滤的方式：

(1) 动作过滤

在 AndroidManifest.xml 的<intent-filter>标签中添加<action>子标签，例如：

```
<intent-filter ...>
    <action android:name="android.intent.ACTION_BATTERY_LOW" />
    <action android:name="android.intent.ACTION_BATTERY_OKAY" />
    <action android:name="android.intent.ACTION_POWER_CONNECTED" />
    <action android:name="android.intent.ACTION_POWER_DISCONNECTED" />
</intent-filter>
```

上述是一个包含电量变化 Action 的 IntentFilter，由此可见多个 Action 可以包含在一个过滤器中。需要注意，如果过滤器不包含任何 Action，即 Action 列表为空，所有的 Intent 都会因匹配失败而被阻塞。所以一个过滤器需要包含至少一个<action>标签，这样系统才能处理 Intent 消息。

(2) 种类过滤

在 AndroidManifest.xml 的<intent-filter>标签中添加<category>子标签，例如：

```
<intent-filter ...>
```



```
<category android:name="android.intent.category.CATEGORY_SELECTED_ALTERNATIVE" />
<category android:name="android.intent.category.CATEGORY_LAUNCHER" />
<category android:name="android.intent.category.CATEGORY_DEFAULT" />
</intent-filter>
```

对于一个 Intent 要通过种类检测, Intent 对象中的每个种类必须匹配过滤器中的一个。即过滤器能够列出额外的种类, 但是 Intent 对象中的种类都必须能够在过滤器中找到, 只要有一个种类在过滤器列表中没有, 就算种类检测失败。

因此, 原则上如果一个 Intent 对象中没有种类 (即种类字段为空) 应该总是能通过种类测试, 而不管过滤器中有什么种类。但是有个例外, Android 对待所有传递给 Context.startActivity() 的隐式 Intent 好像至少包含 android.intent.category.DEFAULT (对应 CATEGORY\_DEFAULT 常量)。因此, 活动想要接收隐式 Intent 必须要在 Intent 过滤器中包含 android.intent.category.DEFAULT。

注意: android.intent.action.MAIN 和 android.intent.category.LAUNCHER 设置, 它们分别标记活动开始新的任务和带到启动列表界面。它们可以包含 android.intent.category.DEFAULT 到种类列表, 也可以不包含。

### (3) 数据过滤

类似的, 清单文件中的 <intent-filter> 标签以 <data> 子标签列出数据, 例如:

```
<intent-filter ...>
  <data android:mimeType="video/mpeg" android:scheme="http://com.example.android:8888/1" />
  <data android:mimeType="audio/mpeg" android:scheme="http://com.example.android:8888/2" />
  <data android:mimeType="audio/mpeg" android:scheme="http://com.example.android:8888/3" />
</intent-filter>
```

每个 <data> 指定一个 URI 和数据类型 (MIME 类型)。它有四个属性 scheme、host、port、path。scheme 是 content, host 是 com.example.project, port 是 200, path 是 folder/subfolder/etc。host 和 port 一起构成 URI 的凭据 (authority), 如果 host 没有指定, port 也被忽略。这四个属性都是可选的, 但它们之间并不都是完全独立的。要让 authority 有意义, scheme 必须也要指定。要让 path 有意义, scheme 和 authority 也都必须要指定。

当比较 Intent 对象和过滤器的 URI 时, 仅仅比较过滤器中出现的 URI 属性。例如, 如果一个过滤器仅指定了 scheme, 所有有此 scheme 的 URI 都匹配过滤器; 如果一个过滤器指定了 scheme 和 authority, 但没有指定 path, 所有匹配 scheme 和 authority 的 URI 都通过检测, 而不管它们的 path; 如果四个属性都指定了, 要都匹配才能算是匹配。然而, 过滤器中的 path 可以包含通配符来要求匹配 path 中的一部分。<data> 的 type 属性指定数据的 MIME 类型。Intent 对象和过滤器都可以用 “\*” 通配符匹配子类型字段, 例如 “text/\*”、“audio/\*” 表示任何子类型。数据检测既要检测 URI, 也要检测数据类型。规则如下:

- 一个 Intent 对象既不包含 URI, 也不包含数据类型: 仅当过滤器不指定任何 URI 和数据类型时, 才不能通过检测, 否则都能通过。
- 一个 Intent 对象包含 URI, 但不包含数据类型: 仅当过滤器不指定数据类型, 同时它们的 URI 匹配, 才能通过检测。例如, mailto: 和 tel: 都不指定实际数据。
- 一个 Intent 对象包含数据类型, 但不包含 URI: 仅当过滤器只包含数据类型且与 Intent 相同时, 才通过检测。



- 一个 Intent 对象既包含 URI, 也包含数据类型 (或数据类型能够从 URI 推断出): 数据类型部分, 只有与过滤器中之一匹配才算通过; URI 部分, 它的 URI 要出现在过滤器中, 或者它有 content: 或 file: URI, 又或者过滤器没有指定 URI。换句话说, 如果它的过滤器仅列出了数据类型, 组件假定支持 content: 和 file: 。

如果一个 Intent 能够通过不止一个活动或服务的过滤器, 用户可能会被询问哪个组件被激活。如果没有找到目标, 会产生一个异常。

#### (4) 其他过滤

上面最后一条规则表明组件能够从文件或内容提供者获取本地数据。因此, 它们的过滤器仅列出数据类型且不必明确指出 content: 和 file:scheme 的名字。这是一种典型的情况, 一个 <data> 像下面这样:

```
<data android:mimeType="image/*" />
```

告诉 Android 这个组件能够从内容提供者获取 image 数据并显示它。因为大部分可用数据由内容提供者分发, 过滤器指定一个数据类型但没有指定 URI 或许最通用。

另一种通用配置是过滤器指定一个 scheme 和一个数据类型。例如, 一个 <data> 像下面这样:

```
<data android:scheme="http" android:type="video/*" />
```

告诉 Android 这个组件能够从网络获取视频数据并显示它。当用户单击一个 Web 页面上的链接时, 浏览器应用程序会做什么? 它首先会试图去显示数据 (如果链接时是一个 HTML 页面, 就能显示)。如果它不能显示数据, 它将把一个隐式 Intent 加到 scheme 和数据类型, 去启动一个能够做此工作的活动。如果没有接收者, 它将请求下载管理者去下载数据。这将在内容提供者的控制下完成, 因此一个潜在的大活动池 (它们的过滤器仅有数据类型) 能够响应。

## 7.4 设置 Activity 许可

同 API 一样, Android 系统开放了许多的底层应用 (如 ACTION\_CALL) 供用户调用。同其他系统不同, Android 系统有自己特殊的调用底层应用的方式。Android 系统会在运行时检查该用户程序是否有权限调用该底层应用, 因此需要通过某种方式设置 Activity 许可才能运行相应的应用。这种方式提供了程序使用系统应用的安全性保证, 底层应用只在相应的权限许可下才能被用户程序使用。设置 Activity 许可的方式是通过清单文件, 否则程序运行会出现错误。例如, 打电话应用需要调用系统提供的电话底层处理 ACTION\_CALL 行为, 这时需要在清单文件 (AndroidManifest.xml) 中的 uses-permission 添加打电话的许可属性。

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

这样用户就能使用 ACTION\_CALL 来激活打电话应用。如果不在清单文件 (AndroidManifest.xml) 中设置许可, 则运行打电话应用时会弹出提示用户缺少相应权限的异常错误。

当运行没有设置 android.permission.CALL\_PHONE 许可的 ACTION\_CALL 应用时, 系统会弹出安全异常错误 (java.lang.SecurityException)。程序缺少许可时的运行结果如图 7-6 所示。

从图 7-6 可以看出, 异常错误提示了发生异常的详细原因, 包括触发异常的行为以及所需要的许可。可以判断出用户程序出现异常错误是因为使用了 android.Intent.action.CALL 行为, 而该行为拨打了一个号码为 88888888 的电话。这种行为需要 android.permission.CALL\_PHONE 的许可。



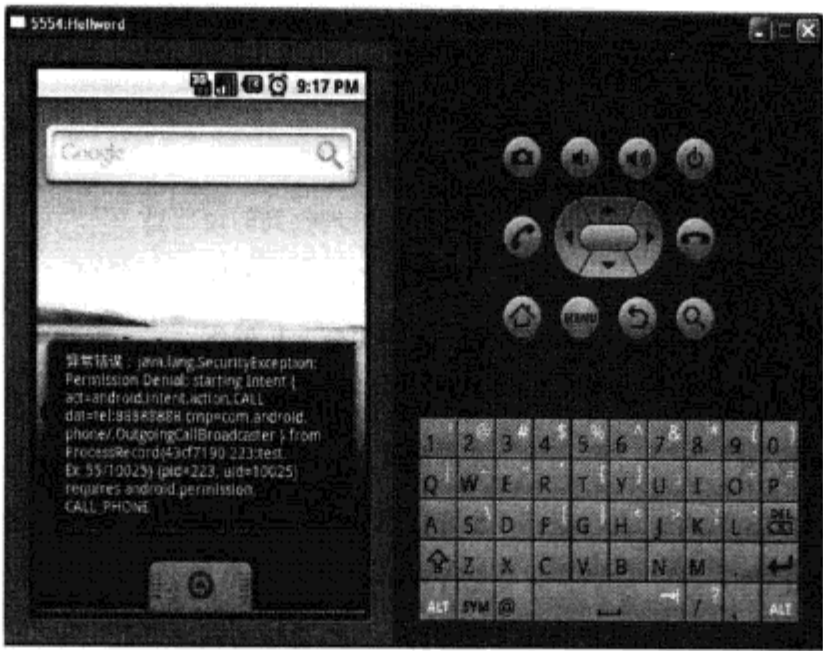


图 7-6 缺少相应权限的异常错误

需要注意，uses-permission 标签包含在 manifest 中，并与 application 标签属于同一级别：

```

<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission android:name="android.permission.CALL_PHONE">    </uses-permission>
    <application>
        <activity>
            <intent-filter>
                </intent-filter>
            </activity>
        </application>
    </manifest>

```

除 android.permission.CALL\_PHONE 之外，Android 系统还提供了许多其他许可。用户使用相应底层服务时，需要在 AndroidManifest.xml 中添加相应的权限。表 7-5 列举了 Android 系统提供的主要的许可。

表 7-5 Android 系统提供的许可

许可名字	许可功能
android.permission.ACCESS_CHECKIN_PROPERTIES	允许读写 checkin 数据库中的 properties 表
android.permission.ACCESS_COARSE_LOCATION	允许程序通过访问 CellID 或 WiFi 热点来获取粗略的位置
android.permission.BLUETOOTH	允许程序同匹配的蓝牙设备建立连接
android.permission.CALL_PHONE	允许程序拨打电话，该行为无需通过拨号器的用户界面确认
android.permission.CLEAR_APP_CACHE	允许用户清除设备上的所有安装程序的缓存
android.permission.CLEAR_APP_USER_DATA	允许程序清除用户数据
android.permission.CONTROL_LOCATION_UPDATES	允许启用/禁止无线模块的位置更新
android.permission.PROCESS_OUTGOING_CALLS	允许程序监视、修改或者删除已拨电话
android.permission.READ_INPUT_STATE	允许程序获取当前按键状态
android.permission.REBOOT	请求用户设备重启的操作
android.permission.RECEIVE_BOOT_COMPLETED	允许一个程序接收到系统启动后的广播 ACTION_BOOT_COMPLETED



续表

许可名字	许可功能
android.permission.RECEIVE_MMS	允许程序处理收到的 MMS 彩信
android.permission.RECEIVE_SMS	允许程序处理收到的短信息
android.permission.SET_TIME_ZONE	允许程序设置系统时区
android.permission.SET_WALLPAPER	允许程序设置手机壁纸
android.permission.STATUS_BAR	允许程序打开、关闭或禁用状态栏及图标
android.permission.WRITE_CALENDAR	允许程序写入但不读取用户日历
android.permission.WRITE_CONTACTS	允许程序写入但不读取用户联系人数据
android.permission.WRITE_GSERVICES	允许程序修改 Google 服务地图
android.permission.WRITE_SETTINGS	允许程序读取或修改系统设置
android.permission.WRITE_SMS	允许程序修改短信
android.permission.DELETE_CACHE_FILES	允许程序删除缓存文件
android.permission.DELETE_PACKAGES	允许程序删除包
android.permission.DEVICE_POWER	允许访问底层电源管理
android.permission.DISABLE_KEYGUARD	允许程序禁用键盘锁
android.permission.DUMP	允许程序获取系统服务的 dump 信息
android.permission.GET_ACCOUNTS	允许访问 Accounts 服务中的账户列表
android.permission.GET_PACKAGE_SIZE	允许程序获取任何 Package 占用空间大小
android.permission.GET_TASKS	允许程序获取当前或最近运行的任务概要信息
android.permission.HARDWARE_TEST	允许访问程序系统硬件
android.permission.INJECT_EVENTS	允许程序截获用户事件，如按键、触摸、回滚等
android.permission.INSTALL_PACKAGES	允许程序安装包
android.permission.INTERNAL_SYSTEM_WINDOW	允许打开系统用户界面窗口
android.permission.EXPAND_STATUS_BAR	允许程序拉伸或者缩小状态栏
android.permission.INTERNET	允许程序打开网络套接字
android.permission.MODIFY_AUDIO_SETTINGS	允许程序修改系统音频设置
android.permission.MODIFY_PHONE_STATE	允许修改电话状态，如充电
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	允许挂载和反挂载移动设备
android.permission.SET_ACTIVITY_WATCHER	允许程序监视和控制系统活动的启动
android.permission.SET_ALWAYS_FINISH	允许程序控制是否活动在处于后台时立即结束
android.permission.SET_DEBUG_APP	配置一个用于调试的程序
android.permission.SET_ORIENTATION	允许通过底层应用设置屏幕方向
android.permission.SET_PREFERRED_APPLICATIONS	允许程序修改默认程序列表
android.permission.SET_PROCESS_FOREGROUND	允许程序强制将当前运行程序转到前台运行
android.permission.SET_PROCESS_LIMIT	允许设置最大的系统当前运行进程数量
android.permission.ACCESS_LOCATION_EXTRA_COMMANDS	允许应用程序使用额外的位置提供命令
android.permission.ACCESS_MOCK_LOCATION	允许程序创建用于测试的模拟位置
android.permission.ACCESS_NETWORK_STATE	允许程序获取网络状态信息



续表

许可名字	许可功能
android.permission.ACCESS_SURFACE_FLINGER	允许程序获取 SurfaceFlinger 底层特性
android.permission.ACCESS_WIFI_STATE	允许程序获取 WiFi 网络信息
android.permission.ADD_SYSTEM_SERVICE	允许程序发布系统级服务
android.permission.BATTERY_STATS	允许程序更新手机电池统计信息
android.permission.BLUETOOTH_ADMIN	允许程序发现和配对蓝牙设备
android.permission.BROADCAST_PACKAGE_REMOVED	允许程序广播一个包已经移除的提示消息
android.permission.BROADCAST_STICKY	允许程序广播带数据的 Intents
android.permission.CAMERA	请求使用照相设备
android.permission.CHANGE_COMPONENT_ENABLED_STATE	允许程序启用或禁用其他组件
android.permission.CHANGE_CONFIGURATION	允许程序修改当前设置
android.permission.CHANGE_NETWORK_STATE	允许程序改变网络连接状态
android.permission.CHANGE_WIFI_STATE	允许程序改变 WiFi 连接状态
android.permission.READ_SYNC_SETTINGS	允许程序读取同步设置
android.permission.READ_CONTACTS	允许程序读取用户联系人数据

7.5 应用实例详解：电话拨号程序

手机设备提供的基本功能之一是打电话，Andorid 系统提供了打电话的底层应用 ACTION\_CALL。用户程序无需关心其实现方式，只需要通过 Activity 调用该应用就能实现电话拨号功能。本节将介绍如何使用 Activity 调用系统提供的底层应用实现电话拨号功能。

7.5.1 实例分析

本实例实现了一个电话拨号的功能，读者可通过该实例掌握拨号功能的实现方法。本实例包含主程序 main.xml、AndroidManifest.xml 以及 Ex\_75.java。

main.xml 中定义了拨号实例的界面包含的元素以及布局方式，该界面包含一个 AutoCompleteTextView 和 ImageButton 控件。AutoCompleteTextView 实现了具有自动提示功能的号码输入框，ImageButton 实现了带有电话图标的拨号按钮。

AndroidManifest.xml 添加了 android.permission.CALL\_PHONE 用户许可，这样系统才能激活 ACTION\_CALL 应用。

Ex\_75.java 是本实例的主程序，该程序首先声明 android.app.Activity、android.os.Bundle、android.content.Intent、android.view.View.OnClickListener、android.widget.AdapterView、android.widget.AutoCompleteTextView、android.widget.ImageButton、android.widget.Toast、import android.net.Uri、java.util.regex.Matcher、java.util.regex.Pattern 等类。

接着定义一个号码表 phonenumberStr 用于提示用户输入，这个号码表需要添加到数组适配器（ArrayAdapter）中。AutoCompleteTextView 通过 setAdapter 方法将 phonenumberStr 添加到自动完成文本控件的搜索库里。当用户在自动完成文本框中输入电话号码时，系统自动从搜索库里搜索包含当前字符串的电话号码。如果在 phonenumberStr 中搜索到包含当前字符串的



电话号码, 则在该文本框下方显示出来。例如, 当用户输入 88 时, 文本框在 `phonenumStr` 中搜索出号码 88888888 含有前缀 88, 则 88888888 号码在文本框下方显示出来。这个功能类似于手机电话查找, 程序会自动显示与当前输入匹配的电话号码。

然后设置 `ImageButton` 单击事件监听器。当该控件被单击时, `MyButtonClickListener` 类的 `onClick` 方法被调用。在 `onClick` 方法中, 使用了 `try...catch` 机制捕捉程序的异常。在 Android 中, 我们使用 `try...catch` 语句来处理运行时的异常, 它的基本语法如下:

```
try
{
    // 此处是可能产生异常的语句
}
catch(error)
{
    // 此处是负责处理异常的语句
}
finally
{
    // 此处是出口语句
}
```

上述代码中, `try` 块中的语句首先被执行。如果运行过程中发生了错误, 控制就会转移到位于 `catch` 块中的语句, 其中括号中的 `error` 参数被作为例外变量传递, 否则 `catch` 块的语句被跳过不执行。无论是发生错误 `catch` 块中的语句执行完毕, 或者没有发生错误 `try` 块中的语句执行完毕, 最后都将执行 `finally` 块中的语句。`finally` 块一般包含程序的结束处理语句, 如回收资源, 因此本实例中关闭当前的 `Activity` 的语句被放在 `finally` 块中。

`onClick` 方法调用 `ValidDigitalPhoneNumber(String phone_number, String regexp)` 方法, 该方法判断字符串 `phone_numbe` 是否匹配正则表达式 `regexp`, 并返回一个 `boolean` 值作为判断依据。`ValidDigitalPhoneNumber` 方法包含两个参数: 第一个参数是需要匹配的字符串, 第二个参数是正则表达式。若第二个参数能够匹配第一个参数, 则返回真值, 否则返回假值。

正则表达式 (Regular Expression) 是包含特殊意义的字符串, 这个特殊的字符串定义了一个模式来搜索匹配字符串。如果曾经用过 Perl 或任何其他内建正则表达式支持的语言, 一定知道用正则表达式处理文本和匹配模式是多么简单。Perl、PHP、Python、JavaScript 和 JScript 等都支持正则表达式, 一些文本编辑器用正则表达式实现高级“搜索-替换”功能。例如, 需从一个文本中查找所有的 IPv4 地址。由于 IPv4 的地址包含 4 个字节, 每个字节之间使用一个点作为分割。因此, IP 地址中的每一个字节有至少一个、最多三个数字。IPv4 地址的正则表达式为:

`[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}`

本例定义了两个正则表达式 `invalid_expression="([a-zA-Z]+)$"` 和 `valid_expression="([0-9]+)$"`。在正则表达式中, 表达式“0-9”代表任意单个数字 (从 0 到 9), 同理 “a-z” 或者 “A-Z” 代表小写或者大写字母。因此 `valid_expression="([0-9]+)$"` 用来匹配仅以数字作为开头的字符串, `invalid_expression="([a-zA-Z]+)$"` 用来匹配以字符 (a-z 或者 A-Z) 作为开头的字符串。

Android 系统也提供了正则表达式的支持, 需要导入 `java.util.regex.Matcher`、`java.util.regex.`



Pattern 两个类才能使用正则表达式。下面是使用正则表达式匹配的过程:

(1) 使用 java.util.regex.Pattern 类的方法 compile 生成正则表达式的 Pattern 对象。如:

```
Pattern pattern = Pattern.compile(regex);
```

(2) Pattern 对象 pattern 使用 matcher 方法匹配字符, 判断被匹配的字符是否满足正则表达式的模式并返回 Matcher 对象。如:

```
Matcher matcher = pattern.matcher(phone_number);
```

(3) 第 (2) 步返回的 Matcher 对象调用 matches 方法, 判断是否匹配成功。若匹配成功则返回真值, 否则返回假值。如:

```
if(matcher.matches())
{
    //匹配成功处理
}
else
{
    //匹配失败处理
}
```

若自动完成文本框中输入的字符串满足格式要求, 则在 onClick 方法中新建一个 Intent 对象 myIntent, 并指定被启动的应用为: android.intent.action.CALL。myIntent 的 Intent, 传入 ACTION\_CALL 与 Uri.parse(), 注意传入 Uri 的数据, 电话的前缀为 “tel:”。这种方式需要用户通过键盘输入, 也可使用虚拟键盘来拨打电话。使用虚拟键盘只需要在自定义 Intent 时将 Action.CALL 改为 Action.DIAL 即可:

```
Intent myIntent = new Intent("android.intent.action.DIAL",Uri.parse("tel:"+string));
/*新建一个 Intent 对象, 并指定被启动的应用为: android.intent.action.DIAL*/
startActivity(myIntent);
```

若自动完成文本框中输入的字符串不满足格式要求, 则程序弹出一个无效电话号码格式的 Toast 消息。

读者需要特别注意, 需要 AndroidManifest.xml 添加和 android.intent.action.CALL 相关的权限, 否则调用 android.intent.action.CALL 时, 系统会弹出需要权限的异常错误。android.intent.action.CALL 相关的权限是 android.permission.CALL\_PHONE, 因此需要在 AndroidManifest.xml 中添加:

```
<!-- 添加拨出电话的权限 -->
<uses-permission android:name="android.permission.CALL_PHONE">
```

## 7.5.2 实例实现

主程序 Ex\_75.java 实现:

```
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent; //导入 Intent 包
import android.view.View;
```



```
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.ImageButton;
import android.widget.Toast;
import android.net.Uri;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Ex_75 extends Activity {

    private static final String[] phonenumStr = new String[] { "88888888", "85668888", "7777777",
        "86666666", "7377777" }; //用于自动完成提示的号码表
    /*声明 Button 与 AutoCompleteTextView 对象名称*/
    private ImageButton button_phone;
    private AutoCompleteTextView autoCompletePhone;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        autoCompletePhone = (AutoCompleteTextView) findViewById(R.id.autoCompletePhone);

        /* 以 findViewById()取得 AutoCompleteTextView 对象 */
        button_phone = (ImageButton) findViewById(R.id.button_phone);
        button_phone.setBackgroundResource(R.drawable.phone);
        /*通过 findViewById 构造器来构造 EditText 与 Button 对象*/
        ArrayAdapter<String>adapter=new ArrayAdapter<String>(this,android.R.layout.simple_dropdown_
            item_1line, phonenumStr);
        /* 以 phonenumStr 字符串数组生成 ArrayAdapter 对象 */
        autoCompletePhone.setAdapter(adapter);
        /* 通过 setAdapter()来读取 ArrayAdapter 里的数据: phonenumStr*/
        button_phone.setOnClickListener(new MyButtonClickListener());
        /*设置 Button 对象的 OnClickListener 来监听 OnClick 事件*/
    }
    private class MyButtonClickListener implements OnClickListener{
        public void onClick(View view) {
            try
            {
                String string = autoCompletePhone.getText().toString();
                String valid_expression = "([0-9]+)$";
```



```

        //设置 vaild_expression 为仅包含数字的字符串
        String invalid_expression ="([a-zA-Z]+)$";
        //设置 vaild_expression 为仅包含字母的字符串
        if (ValidDigitalPhoneNumber(string,vaild_expression) && ! ValidDigitalPhoneNumber
            (string,invalid_expression)&& VaildNumberLen(string, 4, 9))
        {
            Intent myIntent = new Intent("android.intent.action.CALL",Uri.parse("tel:"+
                string));
            /*新建一个 Intent 对象, 并指定被启动的应用为: android.intent.action.CALL*/
            startActivity(myIntent);
            /*调用 myIntent 指定的 android.intent.action.CALL, 这个应用是由 Android 系统提供
            的*/
            autoCompletePhone.setText("");
        }
        /*若输入的号码包含数字, 不包含字母, 且字符串长度在 4-9 之间, 则调用 Android
        系统的拨号应用*/
        else
        {
            autoCompletePhone.setText("");
            Toast.makeText(Ex_75.this, "无效电话号码格式",Toast.LENGTH_LONG).
                show();
        }
    }
    catch (Exception e)
    {
        Toast.makeText(Ex_75.this, "异常错误: " + e.toString(),Toast.LENGTH_LONG).
            show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        Ex_75.this.finish(); //关闭当前的 activity
    }
    /*finally 中可添加处理异常的代码*/
}

}

public static boolean VaildNumberLen(String phone_number, int mixlen, int maxlen)
{
    int len = phone_number.length(); //获取电话号码的长度
    if (len>=mixlen && len<=maxlen) //判断电话号码是否在 mixlen 与 maxlen 之间

```



```

        return true;
    }
    return false;
}
/*判断输入的电话号码的长度是否在 mixlen 与 maxlen 之间, 若长度正确返回 true; 否则返回 false*/
public static boolean ValidDigitalPhoneNumber(String phone_number, String regexp)
{
    Pattern pattern = Pattern.compile(regexp);
    /*根据正则表达式 regexp 生成模式对象*/
    Matcher matcher = pattern.matcher(phone_number);
    /*模式对象利用正则表达式 regexp 匹配 phone_number, 返回 Matcher 对象*/
    if(matcher.matches())
    {
        return true;
    }
    return false;
    /*通过 Matcher 对象的 matches 方法判断匹配是否成功,
    * 即 phone_number 是否含有正则表达式 regexp 定义的字符串。
    * 若包含 regexp 定义的字符串, 则返回真值 (true);
    * 否则返回假值 (false)。
    */
}
/*检查电话号码是否包含正则表达式定义的字符串, 若包含返回 true; 否则返回 false*/
}

```

在 AndroidManifest.xml 中添加打电话的 uses-permission:

```

<uses-permission android:name="android.permission.CALL_PHONE"> </uses-permission>
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Ex_75"
        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>

```

main.xml 实现:

```

<!-- AbsoluteLayout 布局 -->
<!-- AutoCompleteTextView 作为号码输入框 -->
<AutoCompleteTextView
    android:id="@+id/autoCompletePhone"
    android:layout_width="fill_parent"

```



```

        android:layout_height="wrap_content"
        android:layout_x="0px"
        android:layout_y="30px"
        android:hint="Please enter phone number"
    />

    <!-- ImageButton 作为拨号按钮 -->
    <ImageButton
        android:id="@+id/button_phone"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="150px"
    />

</AbsoluteLayout>
    
```

在 `res/drawable-hdpi` 文件夹下添加电话按钮图标，该图标作为拨号按钮的图标，如图 7-7 所示。

在 Android 环境下，运行该程序。输入电话号码时，自动完成文本框根据库中包含的字符提示用户。

例如，当输入 88 时，自动完成文本框从库中搜索到匹配的字符串 88888888 并提示给用户，如图 7-8 所示。双击弹出框，提示的字符串被自动输入到文本框中，如图 7-9 所示。

单击拨号按钮，程序判断输入的号码符合规定的格式，调用 `Action.CALL` 进行拨号，如输入图 7-10 所示。



图 7-7 电话按钮图标

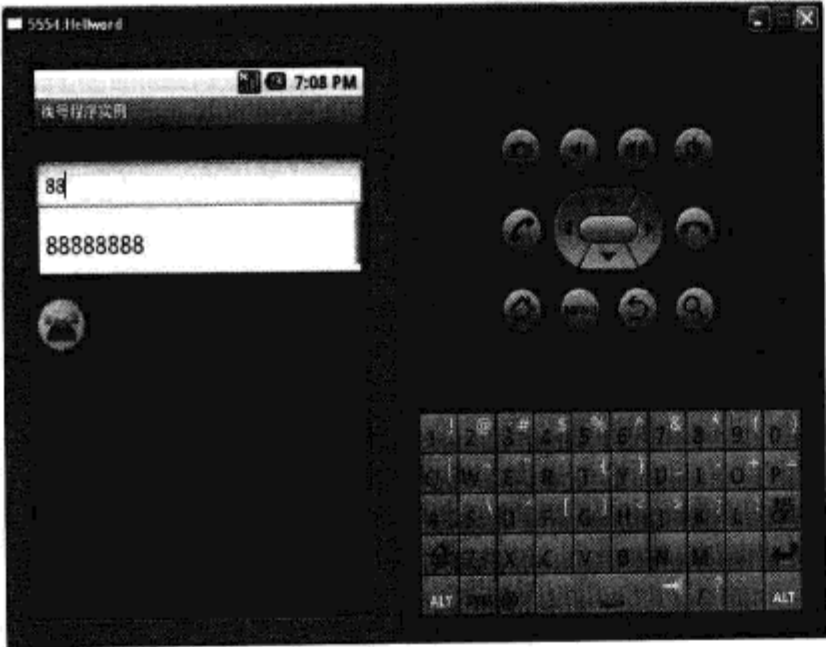


图 7-8 提示用户



图 7-9 提示字符串自动输入



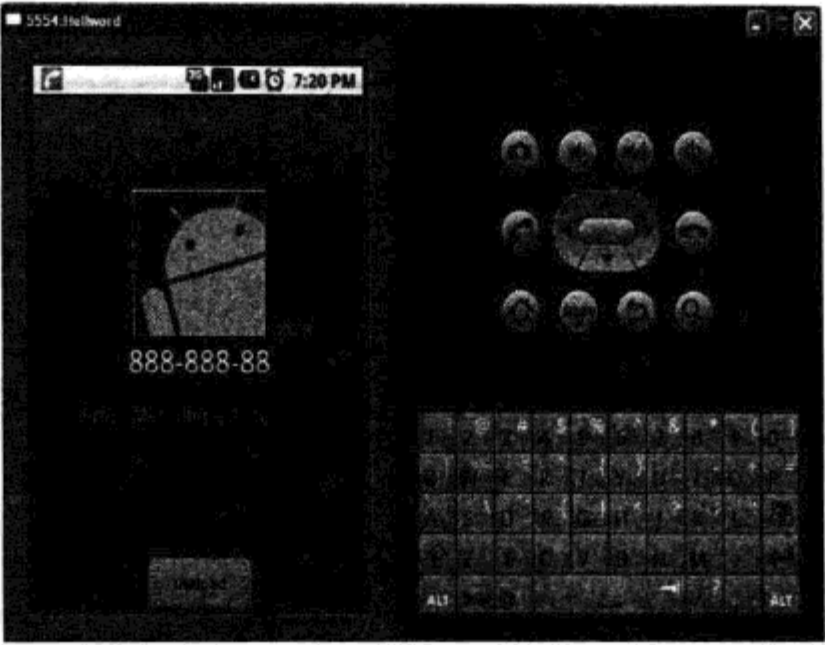


图 7-10 调用 Action.CALL 拨号





## 第 8 章 Android 中的后台服务

应用程序按照工作的方式可分为前台程序和后台服务两种，Android 中的 Activity 是前台程序。Android 系统使用 startActivity 方法调用 Intent 指定的活动，这时活动控制权由当前活动转到 Intent 指定的活动。本章将介绍另一种不同类型的程序——后台服务。Android 提供了 Service 类来实现后台服务，Service 按照类型分为两种：

- 本地服务 (Local Service)

这种服务主要用于程序内部，实现应用程序自己的一些耗时任务，比如自动下载程序。

- 远程服务 (Remote Service)

这种服务主要用于 Android 系统内部的应用程序之间，一个应用程序可以调用其他应用程序的服务。

在 Android 系统中，Service 类继承 ContextWrapper 类的属性和方法，并且 Service 类派生了 AbstractInputMethodService、AccessibilityService、IntentService、RecognitionService、WallpaperService 等类。Service 的派生关系如下所示：

```
java.lang.Object
  android.content.Context
    android.content.ContextWrapper
      android.app.Service
```

### 8.1 Service 的作用

在 Android 开发中，当需要创建在后台运行的程序的时候，就要使用到 Service。Service 的功能类似于 Linux 系统中的守护进程，需要长时间运行以提供后台服务，甚至可能会在系统启动时开始运行到系统关闭时结束。Activity 可与用户进行交互（例如电话拨号程序，该程序需要用户输入号码，并判断用户输入号码的有效性），并且 Activity 运行时会获取当前程序的控制权。然而 Service 与 Activity 完全相反，Service 一般不与用户进行交互，并且 Service 运行时，不会改变当前应用程序的控制权。

Service 不是一个独立的进程，它通常是应用的一部分。因此，Service 本身其实很简单，提供两个主要特点：

- 应用程序的一种设施：告诉系统有关的事情要在后台进行（甚至当用户没有直接的互动申请时）。通过调用 Context.startService 方法来启动 Service，这个方法会一直运行直到 Service 被终止。
- 为应用程序提供交互功能：通过调用 Context.bindService 方法启动一个长期运行的 Service，以与其进行交互。

注意，虽然 Service 本身很简单，但是可以将其按照需要设计成复杂的 Service 或者简单的 Service。可以把 Service 作为一种本地 Java 对象，直接在方法中调用。



Service 最终也是一段代码，而这段代码是在后台运行的。Service 可以根据应用的需要决定其运行方式，共有两种：一种是运行在它自己的进程中，另外一种是在其他应用程序进程的上下文中。其他的组件可以通过 Context.bindService 方法捆绑到 Service，通过远程过程调用（RPC）来调用这个方法。例如，接收短信的 Service，虽然用户没有选择运行短信接收服务，但开机时短信接收服务就会启动，一直运行到用户关机。

有两种方式启动 Service。一种是通过调用 Context.startService 方法启动，通过调用 Context.stopService 方法终止，Context.stopService 方法可以传递参数给 Service。在调用 Context.startService 方法之后，系统首先获取相应的 Service（根据需要调用 onCreate 方法创建），然后通过用户提供的参数调用 onStartCommand 方法，之后这个 Service 会一直在系统中运行，直到调用 Context.stopService 方法或 stopSelf 方法终止。注意，多次调用 Context.startService 方法不会引起嵌套（虽然导致多次调用 onStartCommand 方法），所以不管 Service 被启动多少次，一旦调用 Context.stopService 方法或 stopSelf 方法，都会被终止。注意，Context.stopService 方法和 stopSelf 方法的区别，Context.stopService 方法强行终止当前 Service，而 stopSelf 方法直到 Intent 处理完后才终止 Service。根据 onStartCommand 的返回值不同，启动 Service 有两个附加的模式：

- START\_STICKY：在该模式下，Service 可被显式启动和停止。
- START\_NOT\_STICKY 或 START\_REDELIVER\_INTENT：在该模式下，Service 只在有命令需要处理时才运行。

另外一种是通过调用 Context.bindService 方法启动，调用 Context.unbindService 方法终止，这种方式通过 Service Connection 访问 Service。

Android 除了提供 Service 类以使用户能够实现自己的服务之外，还提供了一些特殊的类，如 AbstractInputMethodService、AccessibilityService、IntentService、RecognitionService、WallpaperService。例如 AccessibilityService 类，当 AccessibilityEvent 被启动时，AccessibilityService 会接收回调函数运行于后台，这些事件指的是用户接口间的状态转换，比如焦点变化、按钮被单击等。一些辅助服务继承此类并且实现它的抽象方法，像这样的服务和和其他服务一样在 AndroidManifest.xml 中被声明，但必须指定操作“android.accessibilityservice.AccessibilityService”的意图。

综上所述，可总结出 Service 具有以下特点：

- 没有用户界面，不与用户进行交互。
- 长期运行，不占用程序控制权。
- 比 Activity 的优先级高，不会轻易被系统终止 即使被系统终止，在系统资源恢复后，也将自动恢复运行状态。
- 用于进程间通信（Inter Process Communication, IPC），解决两个不同 Android 应用程序进程之间的调用和通信问题。

## 8.2 Service 的实现

### 8.2.1 创建 Service

创建 Service 需要使用 extends 关键字继承 Service 类，并且覆盖 Service 类提供的 onCreate、onStart 以及 onDestroy 等方法。在 Service 生命周期中，这些方法分别代表 Service



的不同阶段:

- 方法 onCreate 完成 Service 的初始化工作, 标志 Service 生命周期的开始。
- 方法 onStart 启动一个 Service, 代表 Service 进入了运行状态。
- 方法 onDestroy 释放 Service 所占用的资源, 标志 Service 生命周期的结束。

可按照如下方式创建 Service 类:

/\*继承 Service 类的方法\*/

```
public class MyService extends Service {
```

```
@Override
```

```
public IBinder onBind(Intent intent) {
```

/\*这个方法会在 Service 被绑定到其他程序上时被调用。onBind 将返回给客户端一个 IBinder 接口实例, IBinder 允许客户端回调 Service 的方法, 比如得到 Service 运行的状态\*/

```
return binder;
```

```
}
```

```
@Override
```

```
public void onCreate() {
```

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

```
//添加创建 Service 的代码
```

```
}
```

```
@Override
```

```
public void onStart() {
```

```
super.onStart();
```

```
//通常 Intent 对象会传递给 onStart()方法, 这样可以得到 Intent 里面的数据
```

```
}
```

```
@Override
```

```
public void onDestroy() {
```

```
super.onDestroy();
```

```
//添加释放 Service 资源代码, 例如释放内存
```

```
}
```

```
}
```

```
}
```



8.2.2 启动 Service

上一节讲到，有两种启动 Service 的方式：

- 第一种是通过调用 Context.startService()方法启动，通过调用 Context.stopService()方法终止，startService()方法可以传递参数给 Service。启动一个 Service 的过程为：context.startService() → onCreate() → onStart()。其中，onCreate()方法执行服务的初始化工作，onStart()则执行服务的启动工作。启动方式的生命周期如图 8-1 所示。
- 第二种是通过调用 Context.bindService()方法启动，通过调用 Context.unbindService()方法终止，这种方式通过 ServiceConnection 访问 Service。停止一个 Service 的过程如下：context.stopService() → onDestroy()。捆绑方式的生命周期如图 8-2 所示。

上述两种方式（启动方式和捆绑方式）之间并不是完全独立的，可以混合使用。以 MP3 播放器为例，后台工作的 Service 通过 Context.startService()启动音乐播放器。在播放过程中如果用户需要暂停音乐播放，则需要通过 Context.bindService()获取服务链接和 Service 对象。再通过调用 Service 提供的方法，暂停音乐播放并保存相关信息。在这种情况下，Context.stopService()方法并不能够完全停止 Service，在所有的服务链接关闭后才能使 Service 真正停止。

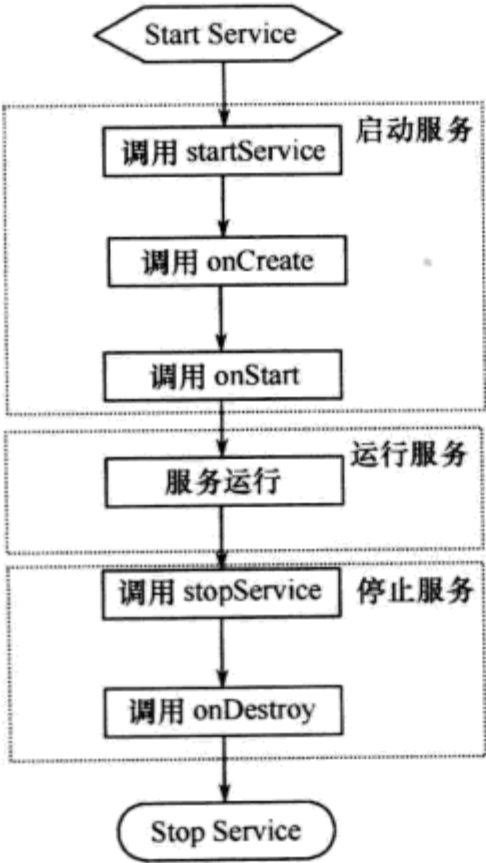


图 8-1 启动方式的 Service 生命周期

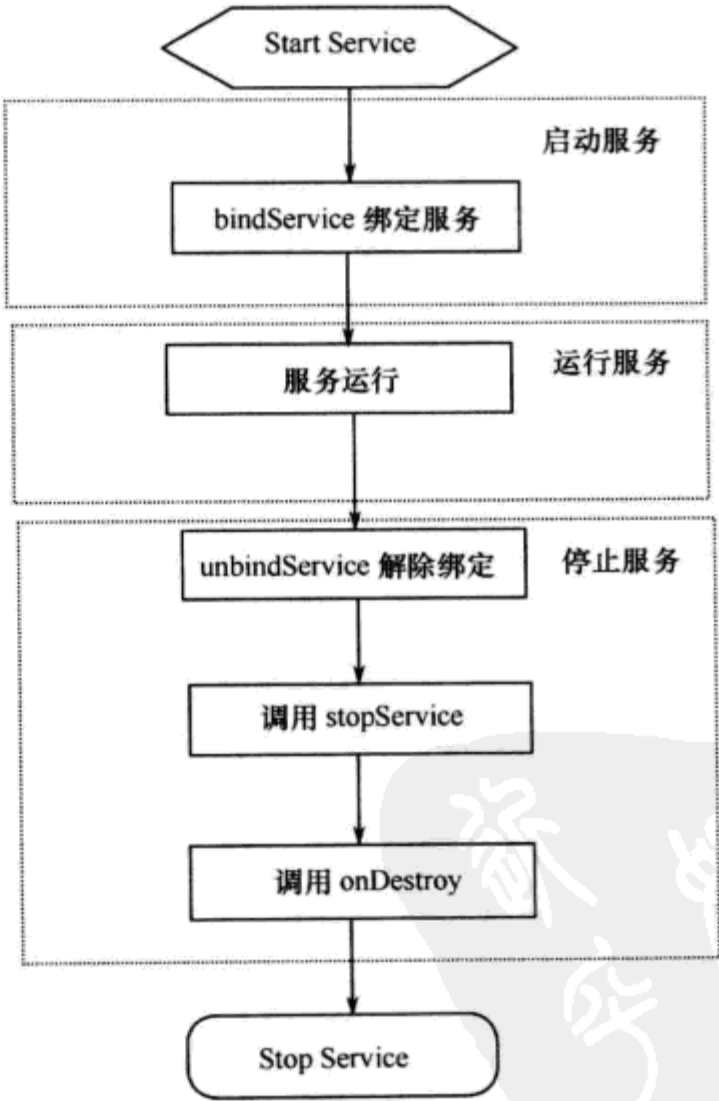


图 8-2 捆绑方式的 Service 生命周期



## 8.3 Toast 和 Notification 应用

Android 支持两种通知方式: Toast 和 Notification。Notification 的通知在状态栏上显示,而 Toast 通知可以显示在屏幕上的任何位置。

### 8.3.1 使用 Notification 通知用户服务启动

对于手机用户来说,会熟悉这样的场景:当有未接来电或者未读短信时,手机顶部状态栏上会显示未接电话或未读短信的图标,该图标提示用户有未接来电或者未读短信;通过单击状态栏,可查看相应信息。Android 提供了 Notification 机制来管理手机状态栏,从而实现用户服务的通知。Notification 被用来通知用户某个事件发生了,比如未读短消息。Notification 允许设置标题,这样可以在“通知窗口”中浏览通知列表。当用户从通知列表中单击某个通知时,Notification 设置的 Intent 就会被触发。通常,这个 Intent 用来启动某个 Activity。

只通过 Notification 类本身提供的方法并不能实现通知机制,还需要结合 NotificationManager 才能实现通知机制。NotificationManager 用于管理 Notification 对象,可通过 getSystemService(Context.NOTIFICATION\_SERVICE)获得 NotificationManager 对象,然后调用 notify 方法将 Notification 对象传递给用户。下面以一个实例讲解 Notification 的用法。

【实例 8-1】Android 的 Notification 应用。

```
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager; //通知管理器, 可发送通知或者取消通知
import android.app.PendingIntent; //导入 PendingIntent 类
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
/*导入程序中使用的类*/

public class Ex_81 extends Activity {

    private NotificationManager nm; //声明 NotificationManager 对象, 该对象用于管理 Notification
    private Button button_sunny; //声明显示 Sunny Notification 的按钮
    private Button button_cloud; //声明显示 Cloud Notification 的按钮
    private Button button_rain; //声明显示 Rain Notification 的按钮
    private Button button_clear;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_notification); //根据 activity_notification.xml 设置程序 UI
```



```

nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
/*NotificationManager 是 OPhone 平台的系统服务, 通过
    getSystemService(Context.NOTIFICATION_SERVICE)可以获得 NotificationManager 对象*/
button_sunny = (Button) findViewById(R.id.button_sunny); //生成 Button 对象 button_sunny
button_cloud = (Button) findViewById(R.id.button_cloud); //生成 Button 对象 button_cloud
button_rain = (Button) findViewById(R.id.button_rain); //生成 Button 对象 button_rain
button_clear = (Button) findViewById(R.id.button_clear); //生成 Button 对象 button_clear

button_sunny.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        dispalyWeather("Sunny day", "天气预报", "晴空万里", R.drawable.sun);
    }
});
/*注册 button_sunny 单击监听器, 当单击 button_sunny 时, 使用 Notification 显示当前的天气为"Sunny
day"*/
button_cloud.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        dispalyWeather("Cloud day", "天气预报", "阴云密布", R.drawable.cloudy);
    }
});
/*注册 button_cloud 单击监听器, 当单击 button_cloud 时, 使用 Notification 显示当前的天气为"Cloud
day"*/
button_rain.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        dispalyWeather("Rainy day", "天气预报", "大雨连绵", R.drawable.rain);
    }
});
/*注册 button_rain 单击监听器, 当单击 button_rain 时, 使用 Notification 显示当前的天气为"Rainy
day"*/
button_clear.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
        nm.cancel(R.layout.activity_notification);
    }
});
/*注册 button_clear 单击监听器, 当单击 button_clear 时, 之前的通知被取消。假如是一个短暂的通
知, 试图将其隐藏, 假如是一个持久的通知, 将从状态条中移走*/
}

private void dispalyWeather(String tickerText, String title, String content,
    int drawable) {

```



```
Notification notification = new Notification(drawable, tickerText, System.currentTimeMillis());
/*创建一个 Notification 对象用于显示通知。该通知的图标为 drawable 对应的图像, 标题为 tickerText
对应的文本, 通知的发送时间为当前时间*/
PendingIntent myIntent = PendingIntent.getActivity(this, 0,
    new Intent(this, Ex_81.class), 0); //生成该通知的 PendingIntent 对象
notification.setLatestEventInfo(this, title, content, myIntent); //设置 Notification 详细参数, 如指定 Intent
nm.notify(R.layout.activity_notification, notification);
/*调用 notify 方法将通知发送到布局 activity_notification 的状态栏中*/
}
}
```

布局 Main.xml 实现:

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

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">

            <Button
                android:id="@+id/button_sunny"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="晴空万里" />

            <Button
                android:id="@+id/button_cloud"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="阴云密布" />

            <Button
```



```

        android:id="@+id/button_rain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="大雨连绵" />

    </LinearLayout>

    <Button android:id="@+id/button_clear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dip"
        android:text="清除 notification" />

</LinearLayout>

</ScrollView>

```

**【代码说明】** 实例使用 Notification 实现了显示天气的状态栏。Notification 的实现可分为下面几个步骤：

(1) 通过 getSystemService(NOTIFICATION\_SERVICE)创建 NotificationManager 对象来管理 Notification。注意，创建 NotificationManager 对象不需要使用构造函数，可通过调用 getSystemService 方法获得相应的对象。系统根据 getSystemService 指定的服务名字返回相应的对象。

(2) 使用 Notification 的构造函数创建 Notification 对象：Notification notification=new Notification()。

(3) 创建 Notification 对象之后，设置 Notification 的属性。例如，设置 Notification 在状态栏上的图标、内容、时间等：Notification(drawable, tickerText, System.currentTimeMillis())。注意，除了通过 Notification 构造函数设置属性之外，还可通过 Notification 类本身的属性设置。语句“Notification(drawable, tickertext, System.currentTimeMillis())”等价于：icon=drawable, tickerText=tickertext, when=System.currentTimeMillis()。

(4) 创建 Intent 对象，并在该对象中指定 Notification 对象：Intent intent=new Intent(notification, class)。

(5) 根据创建的 Intent 对象创建 PendingIntent 对象：PendingIntent m\_PendingIntent=PendingIntent.getActivity(NotificationName.this, 0, intent, 0)。

(6) 根据 PendingIntent 设置 Notification 参数：setLatestEventInfo(this, title, content, myIntent)。

(7) 调用 NotificationManager 的 notify 方法将通知发送到布局 activity\_notification 的状态栏中：notify(R.layout.activity\_notification, notification)。

(8) 使用 NotificationManager 的 cancel 方法删除 Notification。

Notification 包含许多设置属性，表 8-1 列举了 Notification 常用的属性。



表 8-1 Notification 常用的属性

属性名称	描述
audioStreamType	当声音响起时，所用的音频流类型。这些音频流类型必须在 android.media.AudioManager 中定义
contentIntent	设置单击通知条目时所执行的 Intent
contentView	设置在状态栏上显示通知时显示的视图
defaults	设置默认值。例如，DEFAULT_LIGHTS（默认灯）、DEFAULT_SOUND（默认声音）、DEFAULT_VIBRATE（默认震动）、DEFAULT_ALL（以上默认）
deleteIntent	设置单击 Clear All Notifications 按钮时所执行的 Intent
icon	设置状态栏上显示的图标
iconLevel	设置显示图标级别
ledARGB	设置 led 的颜色
ledOffMS	设置关闭 led 时的闪烁时间
ledOnMS	设置开启 led 时的闪烁时间。可以设置 ledOnMS 属性为 1，ledOffMS 属性为 0 来让 led 始终亮着；或者将两者均设置为 0 来将 led 关闭。注意，设置 led 时，必须添加 FLAG_SHOW_LIGHTS 标志位
sound	设置一个音频文件作为 Notification，其值为一个 URI
tickerText	设置状态栏上显示的消息内容
vibrate	设置 Notification 的振动模式，通常需要给 Notification 的 vibrate 属性设定一个时间数组，如 long[] vibrate = new long[] { 1000, 1000, 1000, 1000, 1000 }。注意，使用震动之前，需要在清单文件中添加震动权限：<uses-permission android:name="android.permission.VIBRATE"/>
when	通知发生时的时间，通常使用 System.currentTimeMillis()作为时间戳

需要注意，getService 是 Android 重要的一个 API。这个方法根据传入的 NAME 来取得对应的 Object，然后转换成相应的服务对象。例如指定 NOTIFICATION\_SERVICE，则 getService 方法返回 NotificationManager 对象。表 8-2 列举了 getService 支持的服务。

表 8-2 Andirod 系统服务

Service 名字	作用	返回对象
WINDOW_SERVICE	管理打开的窗口，例如获得屏幕的宽和高	android.view.WindowManager
LAYOUT_INFLATER_SERVICE	布局泵（LayoutInflater）根据 XML 布局文件来绘制视图（View）对象	android.view.LayoutInflater
ACTIVITY_SERVICE	管理 Activity，ActivityManager 为系统中所有运行着的 Activity 交互提供了接口	android.app.ActivityManager
NOTIFICATION_SERVICE	管理通知服务	android.app.NotificationManager
KEYGUARD_SERVICE	管理键盘锁的服务	android.app.KeyguardManager
LOCATION_SERVICE	管理 Location，用于提供位置信息	android.location.LocationManager
SEARCH_SERVICE	管理系统搜索服务	android.app.SearchManager
VEBRATOR_SERVICE	管理手机震动服务	android.os.Vibrator
CONNECTIVITY_SERVICE	管理网络连接服务	android.net.ConnectivityManager
WIFI_SERVICE	管理 WiFi 连接	android.net.wifi.WifiManager
TELEPHONY_SERVICE	管理电话服务	android.telephony.TelephonyManager
SENSOR_SERVICE	管理传感器的访问	android.os.storage.StorageManager
INPUT_METHOD_SERVICE	管理输入法服务	android.view.inputmethod.InputMethodManager

启动程序，单击“晴空万里”按钮，状态栏上显示 Sunny day 图标，如图 8-3 所示。





图 8-3 显示 Sunny day 的 Notification

8.3.2 使用 Toast 显示通知信息

Toast 是 Android 支持两种通知方式之一，通过 Toast 可以产生一些特殊效果，例如音量调节器。当用户改变声音音量时，手机会弹出一个显示当前音量的特殊消息。操作完之后一段时间，该特殊消息会自动消失。Toast 不同于 Notification，该消息的显示时间较短，且可显示在屏幕上的任何位置。Toast 提供了一种特殊效果的视图，该视图以浮于应用程序之上的形式显示。与其他组件不同的是，它不获得焦点，不会影响当前用户的操作。它的目标是尽可能以不显眼的方式，使用户看到程序所提供的信息。下面通过实例 8-2 讲述 Toast 的使用方式。

【实例 8-2】Android 的 Toast 应用——音量调节器。

```

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
/*导入 android 类*/

public class VoiceAdjuster extends Activity {
    /** Called when the activity is first created. */
    private Button add_voice;
    private Button sub_voice;
    private Button mute_voice;
    /*声明 3 个 Button 对象，这些对象用于调节音量*/
    private static int cur_voice=0; //cur_voice 记录了当前音量大小，初始设置为 0

```



```
private final static int MIN_VOICE=0; //MIN_VOICE 定义了最小音量值
private final static int MAX_VOICE=8; //MAX_VOICE 定义了最大音量值
/*定义音量参数: cur_voice、MIN_VOICE 以及 MAX_VOICE*/
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main); //根据 main.xml 生产用户界面
    add_voice = (Button) findViewById(R.id.add_voice); //根据 XML 定义生成 Button 对象 add_voice
    sub_voice = (Button) findViewById(R.id.sub_voice); //根据 XML 定义生成 Button 对象 sub_voice
    mute_voice = (Button) findViewById(R.id.mute_voice); //根据 XML 定义生成 Button 对象 mute_voice

    add_voice.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            if(cur_voice < MAX_VOICE)
                cur_voice = cur_voice + 1;
            //若音量值未达到最大值, 则增加当前的音量值。否则当前音量已为最大, 音量值不变
            String voicetext = (String)generateVoice(cur_voice);
            //根据音量值构造音量显示文本, 即每个竖杠代表一个音量
            Toast.makeText(VoiceAdjuster.this, "音量" + cur_voice + "\n" + voicetext, Toast.LENGTH_LONG).
                show(); //根据音量值显示一个 Toast 消息
        }
    });
    /*注册 add_voice 单击事件监听器。当单击该按钮时, 增加当前音量值, 并且使用 Toast 显示增加后的音量*/
    sub_voice.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            if(cur_voice > MIN_VOICE)
                cur_voice = cur_voice - 1;
            //若音量值未达到最小值, 则减小当前的音量值。否则当前音量已为最小, 音量值不变
            String voicetext = (String)generateVoice(cur_voice);
            //根据音量值构造音量显示文本, 即每个竖杠代表一个音量
            Toast.makeText(VoiceAdjuster.this, "音量" + cur_voice + "\n" + voicetext, Toast.LENGTH_LONG).
                show();
            //根据音量值显示一个 Toast 消息
        }
    });
    /*注册 sub_voice 单击事件监听器。当单击该按钮时, 减小当前音量值, 并且使用 Toast 显示减小后的音量*/
    mute_voice.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            cur_voice = 0;
```



```

        //置音量值为 0
        String voicetext = (String)generateVoice(cur_voice);
        //根据音量值构造音量显示文本, 即每个竖杠代表一个音量
        Toast.makeText(VoiceAdjuster.this, "音量" + cur_voice + "\n" + voicetext, Toast.LENGTH_LONG).
        show();
        //根据音量值显示一个 Toast 消息
    }
});
/*注册 mute_voice 单击事件监听器。当单击该按钮时, 当前音量值被置为 0, 并且使用 Toast 显示音
量为 0*/
}
private CharSequence generateVoice(int voice)
{
    CharSequence str = ""; //声明并初始化 CharSequence 对象
    while ( voice > 0)
    {
        str= str + "|";
        voice --;
    }
    /*根据音量值构造 Toast 显示的文本*/
    return str; //返回 CharSequence 对象 str
}
/*generateVoice 方法根据音量值返回音量文本, 该文本的竖杠个数等于音量值。如音量值为 3 时, 返回 3
个竖杠*/
}

```

清单文件定义:

```

package="test.Ex_82"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".Ex_82"
        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>

```

**【代码说明】** 实例 8-2 使用 Toast 实现了音量调节功能。在用户单击音量调节按钮后, 音量调节器会显示当前的音量, 调节完之后, 音量调节器会自动消失。



实例 8-2 定义了三个控制按钮 add\_voice、sub\_voice 以及 mute\_voice，这三个按钮分别用于实现音量增加、音量减小以及音量清除功能：

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <!-- 音量增加按钮 -->
    <Button
        android:id="@+id/add_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="20px"
        android:text="增加"
        />
    <!-- 音量减小按钮 -->
    <Button
        android:id="@+id/sub_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="40px"
        android:text="减小"
        />
    <!-- 静音按钮 -->
    <Button
        android:id="@+id/mute_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="80px"
        android:text="静音"
        />
</LinearLayout>
```



当单击 `add_voice` 按钮时，程序首先判断音量值是否达到最大值（`MAX_VOICE`）。若未达到最大值，则增加当前的音量值。否则当前音量已为最大，则音量值不变。当单击 `sub_voice` 按钮时，程序首先判断音量值是否达到最小值（`MIN_VOICE`）。若未达到最小值，则减小当前的音量值。否则当前音量已为最小，则音量值不变。当单击 `mute_voice` 按钮时，程序会设置当前音量为 0。

实例通过 `Toast` 来实现音量显示，`Toast` 的实现步骤比 `Notification` 简单，分为两步：

（1）创建 `Toast` 对象并设置 `Toast` 参数。通过调用 `Toast` 类的静态方法 `makeText` 来设置 `Toast` 对象，该方法会返回一个 `Toast` 对象。`Toast` 类定义了两个 `makeText` 方法：

- `public static Toast makeText (Context context, Charsequence charset, int duration)`：生成包含文本的标准 `Toast` 对象。`context` 是使用的上下文，通常是本程序的 `Application` 或 `Activity` 对象；`charset` 是显示的字符串；`duration` 是信息的存续期间，值为 `LENGTH_SHORT` 或 `LENGTH_LONG`。
- `public static Toast makeText (Context context, int resourceid int duration)`：生成从资源中取得的文本的标准 `Toast` 对象。`context` 是使用的上下文，通常是本程序的 `Application` 或 `Activity` 对象；`resourceid` 是要使用的字符串资源 ID，可以是已格式化的文本；`duration` 是信息的存续期间，值为 `LENGTH_SHORT` 或 `LENGTH_LONG`。

（2）调用 `show` 方法显示 `Toast` 消息。

启动该 `Android` 程序，图 8-4 显示了程序初始运行界面。

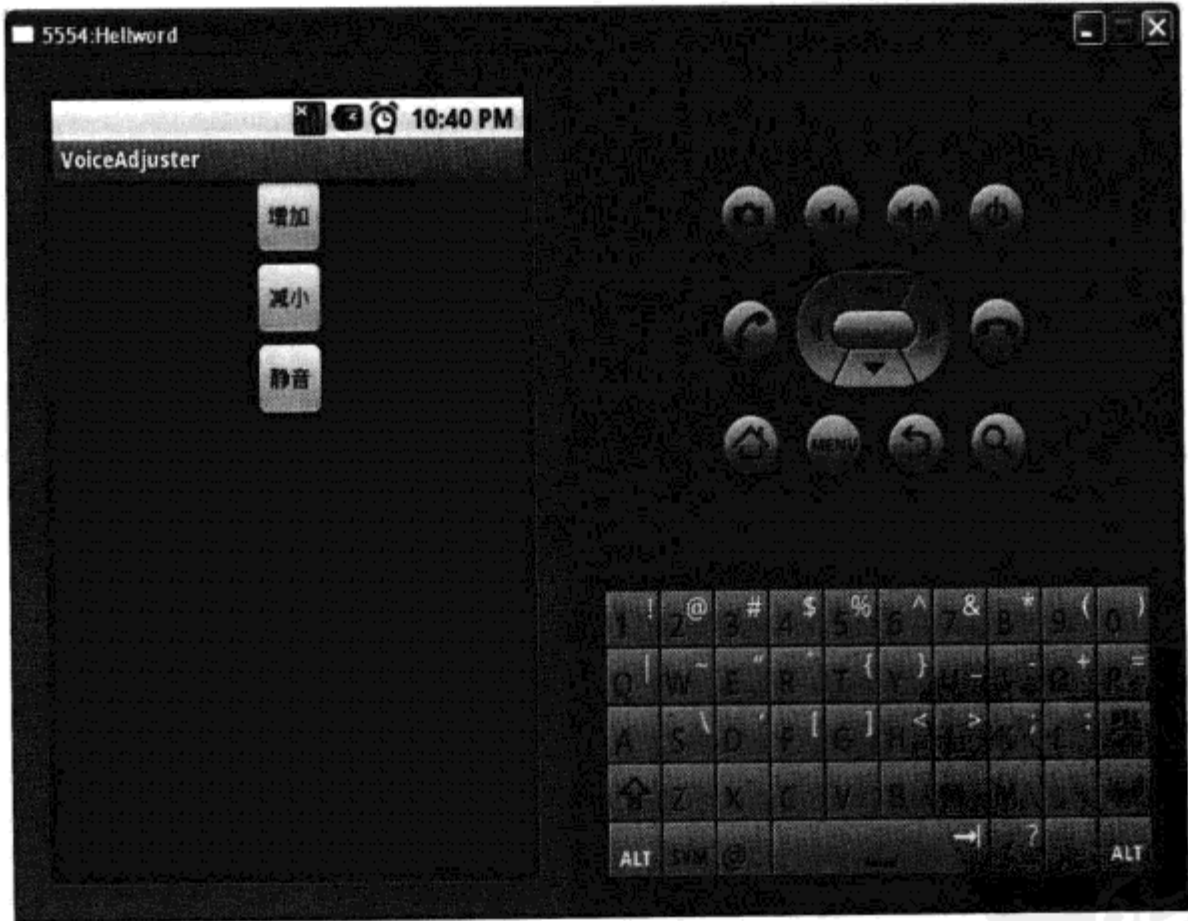


图 8-4 程序初始运行界面

图 8-4 所示界面包含 3 个按钮，这些按钮用于调整音量并显示 `Toast` 消息。例如，单击“增加”按钮，程序会显示一个音量为 1 的 `Toast` 消息。程序运行结果如图 8-5 所示。





图 8-5 显示音量调节器

## 8.4 应用实例详解：播放背景音乐

### 8.4.1 实例分析

实现播放背景音乐至少需要实现两个 Java 类。一个是播放背景音乐类，这个类通过继承 android.app.Service 类将播放音乐功能封装成一个具有后台程序功能的类。在这个类中，需要重载 Service 类的 onStart 以及 onDestroy 方法：

```

public void onStart(Intent intent, int startId) {
    //启动播放音乐服务
}

public void onDestroy() {
    //停止播放音乐服务
}

```

另外一个类提供了播放背景音乐 Service 的控制接口，这个类用于启动或者停止背景音乐的播放。因此该类需要使用 startService 和 stopService 方法来启动或者停止背景音乐的播放。这两个方法（startService 和 stopService）需要一个 Intent 对象作为参数，这个 Intent 对象指定了需要被调用的 Service 类：

```

Intent myIntent= new Intent("SERVICE_NAME");
stopService(myIntent);

```

Intent 对象中指定的 Service 类是通过清单文件（AndroidManifest.xml）中的 Service 标签定义的：

```

<service android:name=".Music">
    <intent-filter>
        <action android:name=""SERVICE_NAME" />
        <!--定义 Service 的名字，与 startService 或 stopService 中的 Service 名字一致-->
    
```



```
<category android:name="android.intent.category.default" />
</intent-filter>
</service>
```

## 8.4.2 实例实现

【实例 8-3】播放背景音乐。

主文件 Playbackground.java 实现，该程序提供了两个按钮，分别用于启动或者结束播放背景音乐 Service。

```
package test.Ex_83; //程序打包

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
/*引入程序所需要的类*/
public class PlaybackgroundMusic extends Activity {
    private Button start;
    private Button stop;
    //声明两个 Button 对象：start 和 stop。start 按钮控制播放背景音乐，stop 按钮用于停止背景音乐
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //根据 main.xml 生成界面
        start = (Button)findViewById(R.id.start); //根据 XML 定义生成按钮对象 start
        stop = (Button)findViewById(R.id.stop); //根据 XML 定义生成按钮对象 stop

        start.setOnClickListener(new android.view.View.OnClickListener() {
            public void onClick(android.view.View v) {
                Intent myIntent= new Intent("test.Ex_83.PlaybackgroundMusic.START_AUDIO_
                SERVICE");
                startService(myIntent);
                //使用 startService 方法启动 Service，其 Intent 为"test.Ex_83.PlugBackgroundMusic.START_
                AUDIO_SERVICE"
            }
        });
        /*注册 start 单击事件监听器。单击该按钮时，其对应的 onClick 方法被调用*/
        stop.setOnClickListener(new android.view.View.OnClickListener() {
            public void onClick(android.view.View v) {
                //stopService(new Intent("test.Ex_83.PlaybackgroundMusic.START_AUDIO_SERVICE"));
                Intent myIntent= new Intent("test.Ex_83.PlaybackgroundMusic.START_AUDIO_
                SERVICE");
```



```

        stopService(myIntent);
        //使用 stopService 方法停止 Service, 其 Intent 为"test.Ex_83.PlagBackgroundMusic.
        START_AUDIO_SERVICE"
        finish();
        //使用 finish 释放 Service 资源
    }
});
/*注册 stop 单击事件监听器。单击该按钮时, 其对应的 onClick 方法被调用*/
}
}

```

布局 main.xml 定义:

```

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

    <Button
        android:id="@+id/start"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Start Play"
    />

    <Button
        android:id="@+id/stop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Stop Play"
    />

</LinearLayout>

```

播放背景音乐类 (Music.java) 实现, 这个类通过继承 android.app.Service 类将播放音乐功能封装成一个具有后台程序功能的类。在这个类中, 需要重载 Service 类的 onStart 以及 onDestroy 方法:

```

package test.Ex_83;

import android.app.Service; //导入 Service 类
import android.content.Intent; //导入 Intent 类, 用于 Activity 与 Service 之间的消息传递
import android.media.MediaPlayer; //导入媒体播放器类, 该类提供了播放媒体的方法

```



```
import android.os.IBinder; //导入捆绑类

public class Music extends Service { //Music 类使用关键字 extends 继承 Service 类
    private MediaPlayer player; //声明媒体播放器对象

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId); //使用关键字 super 来调用父类的 onStart 方法
        player = MediaPlayer.create(this, R.raw.music); //根据音乐文件创建媒体播放器对象
        player.start(); //播放 R.raw.music 指定的音频文件
    }
    /*Music 类启动时, 会自动调用 onStart 方法*/
    public void onDestroy() {
        super.onDestroy(); //使用关键字 super 来调用父类的 onDestroy 方法
        player.stop(); //停止播放 R.raw.music 指定的音频文件
    }
}
```

【代码说明】本实例实现了背景音乐播放功能。

在清单文件（AndroidManifest.xml）中定义了 Service 标签，该标签的 action 元素属性 android:name 与 Playbackground.java 中的 stopService 和 startService 服务一致：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="test.Ex_83"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".PlaybackgroundMusic"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".Music">
```



```

<intent-filter>
    <action android:name="test.Ex_83.PlayBackgroundMusic.START_AUDIO_SERVICE" />
    <category android:name="android.intent.category.default" />
</intent-filter>
</service>
</application>

</manifest>

```

AndroidManifest.xml 是 Android 程序的清单文件，每个应用程序在工程根目录下都有一个 AndroidManifest.xml 文件（一定是这个名字）。这个清单文件给 Android 系统提供了关于这个应用程序的基本信息，系统能在运行任何程序之前通过清单文件获取这些信息。AndroidManifest.xml 主要包含以下功能：

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

启动该 Android 程序，图 8-6 显示了程序运行结果。

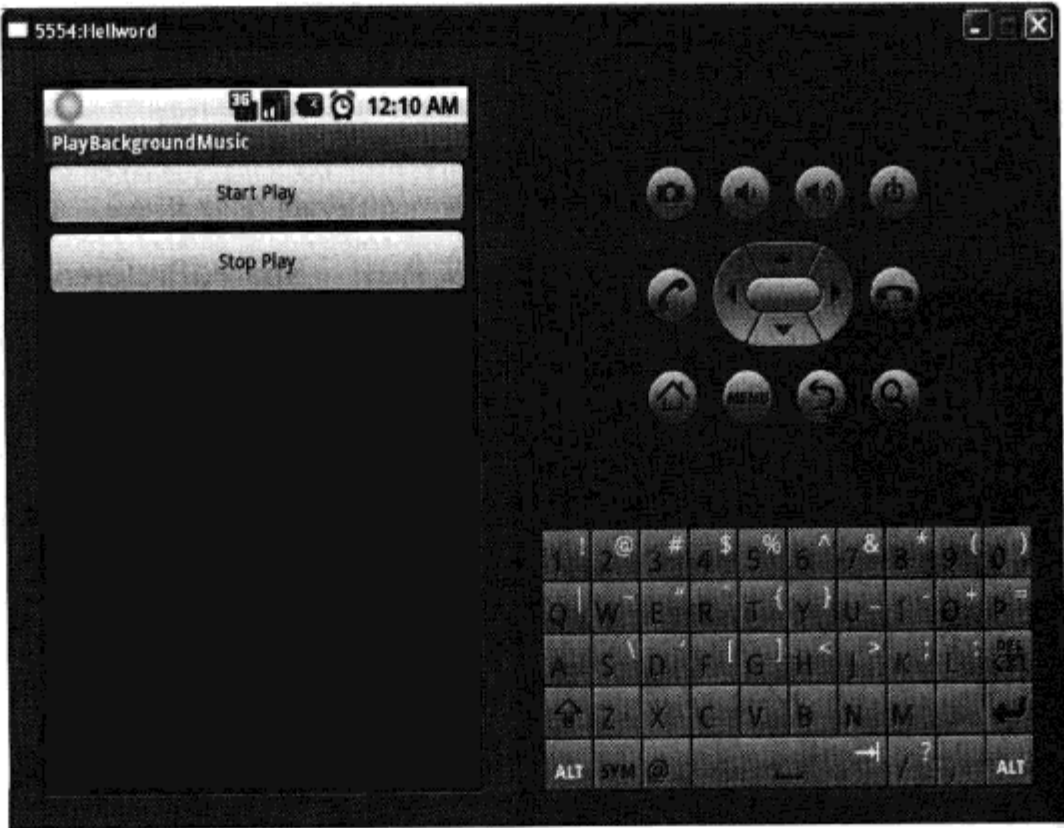


图 8-6 程序运行结果



## 第 9 章 Android 中的数据存储

程序可被理解为数据输入、输出以及处理的过程，通常需要读取处理数据并将处理结果保存起来。Android 提供了五种数据存储方式：

### (1) 使用 Preferences 存储数据

Preferences 采用“键-值”对组织和管理数据，其数据存储存储在 XML 文件中。相对于其他方式，它是一个“轻量级”的存储机制。该方式实现比较简单，适合简单数据的存储。

### (2) 使用文件存储数据

从存储量来看，文件存储是一个“重量级”的存储机制，它比 Preferences 方式更适合存储较大的数据。从存储结构来看，这种方式不同于 SQLite，不适合结构化的数据存储。

### (3) 使用数据库 (SQLite) 存储数据

SQLite 使用数据库作为存储介质，它是一个“重量级”的存储机制。这种方式适合大数据量的数据存储，通过这种方式能够对数据库进行增加、插入、删除、更新等操作。相比 Preferences 和文件存储，这种方式实现较为复杂。

### (4) 使用内容提供者 (ContentProvider) 存储数据

内容提供者可以使用数据库或者文件作为存储介质，通常使用 SQLite 作为存储方式。相比其他方式，这种方式支持多个应用程序之间的数据交换。

本章将详细介绍 Android 中的数据存储机制，通过本章的学习读者将熟悉 Android 存储数据的方式。

## 9.1 使用 Preferences 存储数据

### 9.1.1 访问 Preferences 的 API

SharedPreferences 是 Android 中的一种“轻量级”的数据存储机制，它使用“键-值”对的方式将数据存储到程序 apk 的安装目录下的 XML 文件中。SharedPreferences 类似于 Web 开发的 Cookie，这是一种常用的存储方式。

相比其他方式，SharedPreferences 方式实现较为简单。SharedPreferences 数据存储过程可分为以下几个步骤：

(1) 使用 `getSharedPreferences` 生成 `SharedPreferences` 对象。调用 `getSharedPreferences` 时，需要指定两个参数：存储“键-值”对的 XML 文件名和打开模式。

(2) 使用 `SharedPreferences` 内部类 `Editor` 的 `putXXX` 方法为数据建立“键-值”对。注意，要根据数据类型调用相应的 `putXXX` 方法。例如，调用 `putString("key", "Hello Android")` 为字符串“Hello Android”建立“键-值”对。

(3) 使用 `Editor` 类的 `commit()` 方法将上一步骤建立的“键-值”对写到 XML 文件中。这个 XML 文件存储在 `/data/data/包名/shared_prefs/` 目录下，其文件名是由 `getSharedPreferences` 指



定的。若该文件不存在，则创建一个新的文件。

(4)使用 SharedPreferences 类的 getXXX(String key, XXX defaultValue)方法获取关键字 key 在 getSharedPreferences 指定的 XML 文件中相关联的值。若 XML 文件中包含 key，则返回对应的值，否则返回 getXXX 方法指定的 defaultValue。

Editor 和 SharedPreferences 类使用 putXXX 和 getXXX 方法存储和查看 SharedPreferences 数据。SharedPreferences 只支持简单数据类型的存储操作 (string、float、int 和 long)。

### 9.1.2 使用 XML 存储 Preferences 数据

通过 SharedPreferences, Preferences 数据被存储到/data/data/包名/shared\_prefs/目录下的 XML 文件中。下面以编辑用户的 Preferences 数据为例，介绍使用 XML 存储 Preferences 数据的方法。

【实例 9-1】编辑用户的 Preferences 数据。

```
package test.Ex_91;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences; //导入 SharedPreferences 类
import android.content.SharedPreferences.Editor; //导入 SharedPreferences 编辑器，通过该编辑器存储数据
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class SharedPreference extends Activity {
    /** Called when the activity is first created. */
    private Button add_voice;
    private Button sub_voice;
    private Button mute_voice;
    /*声明 3 个 Button 对象，这些对象用于调节音量*/
    private SharedPreferences sharedPreferences;
    private static int cur_voice=0; //cur_voice 记录了当前音量大小，初始设置为 0
    private final static int MIN_VOICE=0; //MIN_VOICE 定义了最小音量值
    private final static int MAX_VOICE=8; //MAX_VOICE 定义了最大音量值
    /*定义音量参数：cur_voice、MIN_VOICE 以及 MAX_VOICE*/
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //根据 main.xml 生成程序 UI
        sharedPreferences = getSharedPreferences("preferences",
            Context.MODE_PRIVATE);
        /*getSharedPreferences 生成 SharedPreferences 对象，数据存储在 preferences.xml 中*/
    }
}
```



```

add_voice = (Button) findViewById(R.id.add_voice); //根据 XML 定义生成 Button 对象 add_voice
sub_voice = (Button) findViewById(R.id.sub_voice); //根据 XML 定义生成 Button 对象 sub_voice
mute_voice = (Button) findViewById(R.id.mute_voice); //根据 XML 定义生成 Button 对象 mute_voice
cur_voice = getVoicevalue(sharedPreferences);
//通过 getVoicevalue 方法获取 SharedPreferences 存储的音量值, 即最近一次设置的音量值
Toast.makeText(SharedPreferences.this, "上次设置音量: " + cur_voice, 1).show();
//显示上次音量值的 Toast 消息

add_voice.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if(cur_voice < MAX_VOICE)
            cur_voice = cur_voice + 1;
        //若音量值未达到最大值, 则增加当前的音量值; 否则当前音量已为最大, 音量值不变
        String voicetext = (String)generateVoice(cur_voice);
        //根据音量值构造音量显示文本, 即每个竖杠代表一个音量
        Toast.makeText(SharedPreferences.this, "音量" + cur_voice + "\n" + voicetext, Toast.
            LENGTH_LONG).show();
        //根据音量值显示一个 Toast 消息
        saveVoicevalue(cur_voice, sharedPreferences);
        //将音量存储到 SharedPreferences 对象指定的 XML 文件中
    }
});
/*注册 add_voice 单击事件监听器。当单击该按钮时, 当前音量值被增加, 并且使用 Toast 显示增加
后的音量*/
sub_voice.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        if(cur_voice > MIN_VOICE)
            cur_voice = cur_voice - 1;
        //若音量值未达到最小值, 则减小当前的音量值; 否则当前音量已为最小, 音量值不变
        String voicetext = (String)generateVoice(cur_voice);
        //根据音量值构造音量显示文本, 即每个竖杠代表一个音量
        Toast.makeText(SharedPreferences.this, "音量" + cur_voice + "\n" + voicetext, Toast.
            LENGTH_LONG).show();
        //根据音量值显示一个 Toast 消息
        saveVoicevalue(cur_voice, sharedPreferences);
    }
});
/*注册 sub_voice 单击事件监听器。当单击该按钮时, 当前音量值被减小, 并且使用 Toast 显示减小
后的音量*/
mute_voice.setOnClickListener(new View.OnClickListener() {

```



```

public void onClick(View v) {
    cur_voice = 0;
    //置音量值为 0
    String voicetext = (String)generateVoice(cur_voice);
    //根据音量值构造音量显示文本, 即每个|代表一个音量
    Toast.makeText(SharedPreferences.this, "音量" + cur_voice + "\n" + voicetext, Toast.LENGTH_
    LONG).show();
    //根据音量值显示一个 Toast 消息
    saveVoicevalue(cur_voice, sharedPreferences);
}

});
/*注册 mute_voice 单击事件监听器。当单击该按钮时, 当前音量值被置为 0, 并且使用 Toast 显示音
量为 0*/
}

private CharSequence generateVoice(int voice)
{
    CharSequence str = ""; //声明并初始化 CharSequence 对象
    while ( voice > 0)
    {
        str= str + "|";
        voice --;
    }
    /*根据音量值构造 Toast 显示的文本*/
    return str; //返回 CharSequence 对象 str
}

/*generateVoice 方法根据音量值返回音量文本, 该文本的竖杠个数等于音量值。如音量值为 3 时, 返回"|||"*/
void saveVoicevalue(int voicevalue, SharedPreferences sharedPreferences)
{
    Editor editor = sharedPreferences.edit();
    //生成 SharedPreferences 编辑对象, 通过该对象将数据放入到 SharedPreferences 中
    editor.putInt("key", voicevalue);
    /*指定 XML 中存储 voicevalue 值的标签为 key, 即 SharedPreferences 中包含以下信息:
    * <string name="key">$voicevalue</string>
    */
    boolean ret = editor.commit(); //将音量值放入 SharedPreferences 中, 该值通过 key 引用
    if (ret == true)
        Toast.makeText(SharedPreferences.this, "保存成功", 1).show();
    else
        Toast.makeText(SharedPreferences.this, "保存失败", 1).show();
}

```



```

        /*根据 editor.commit()判断数据存储是否成功*/
    }
    /*saveVoicevalue 方法用于将当前的音量值存储到 SharedPreferences 中*/
    int getVoicevalue(SharedPreferences sharedPreferences)
    {
        int ret = sharedPreferences.getInt("key", 0);
        //通过调用 SharedPreferences 对象的 getXXX 方法获取 SharedPreferences 中存储的值
        return ret; //返回结果
    }
    /*getVoicevalue 方法用于获取当前的音量值*/
}

```

【代码说明】实例 9-1 是一个音量调节程序，该程序使用 SharedPreferences 存储用户设置的音量值。程序布局定义如下：

```

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 音量增加按钮 -->
    <Button
        android:id="@+id/add_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="20px"
        android:text="增加"
    />
    <!-- 音量减小按钮 -->
    <Button
        android:id="@+id/sub_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="40px"
        android:text="减小"
    />
    <!-- 静音按钮 -->
    <Button

```



```

        android:id="@+id/mute_voice"
        android:layout_gravity="center"
        android:layout_width="50px"
        android:layout_height="50px"
        android:layout_x="0px"
        android:layout_y="60px"
        android:text="静音"
    />

```

```
</LinearLayout>
```

实例首先通过 `getVoicevalue` 方法获取 `SharedPreferences` 存储的音量值,即最近一次设置的音量值。

若用户改变音量,程序会使用 `Editor` 类的 `commit()` 方法将设置的音量值写到 XML 文件中。这个 XML 文件存储在 `/data/data/包名/shared_prefs/` 目录下,其文件名是由 `getSharedPreferences` 第一个参数指定的。

可通过 `SharedPreferences` 提供的 `getInt` 方法从 XML 文件中获取其值。这个 XML 文件是在调用 `getSharedPreferences` 方法生成 `SharedPreferences` 对象时指定的:

```
getSharedPreferences("preferences", Context.MODE_PRIVATE);
```

`getSharedPreferences` 方法定义了两个形参,其中第一个参数为存储 Preferences 数据的 XML 文件名字,第二个参数是打开模式。注意,第一个参数不需要指定后缀 (`.xml`),即系统会在这个文件名之后添加 `.xml` 后缀并创建之。

`getSharedPreferences` 指定的 Preferences 文件存储在 APK 的安装目录下,可通过 `Andoird DDMS` (`Dalvik Debug Monitor Service`) 查看 Preferences 文件。在 Eclipse 下单击 `Windows→Open Perspective→MMDS`,打开 MMDS。DDMS 为 IDE 和 Emultor、真正的 Android 设备架起了一座桥梁。开发人员可以通过 DDMS 看到目标机器上运行的进程状态,可以查看进程的 Heap 信息,可以查看 LogCat 信息,可以查看进程的内存分配情况,可以给 Android 发送地理位置信息。在 MMDS 中展开 `/data/data/text.Ex_91/shared_prefs`,可看到在该目录下有一个 `preferences.xml` 文件,如图 9-1 所示。

创建好 Preferences 文件后,`android.content.SharedPreferences` 的内部类 `Editor` 使用 `put` 方法为存储数据建立“键-值”对。然后 `android.content.SharedPreferences.Editor` 使用 `commit` 方法将音量值的“键-值”对存储到 XML 文件中。例如,音量为 0 时,Preferences 文件的内容如下:

```

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<int name="key" value="0" />
</map>

```

然后使用 `getXXX` 方法通过 `key` 引用其关联的存储数据。

在 Eclipse 中启动该程序,单击“增加”、“减小”或者“静音”按钮时,音量会自动保存到 Preferences 文件中。程序运行界面如图 9-2 所示。

重新启动应用程序,通过 Preferences 文件获取最近一次保存的数据,如图 9-3 所示。



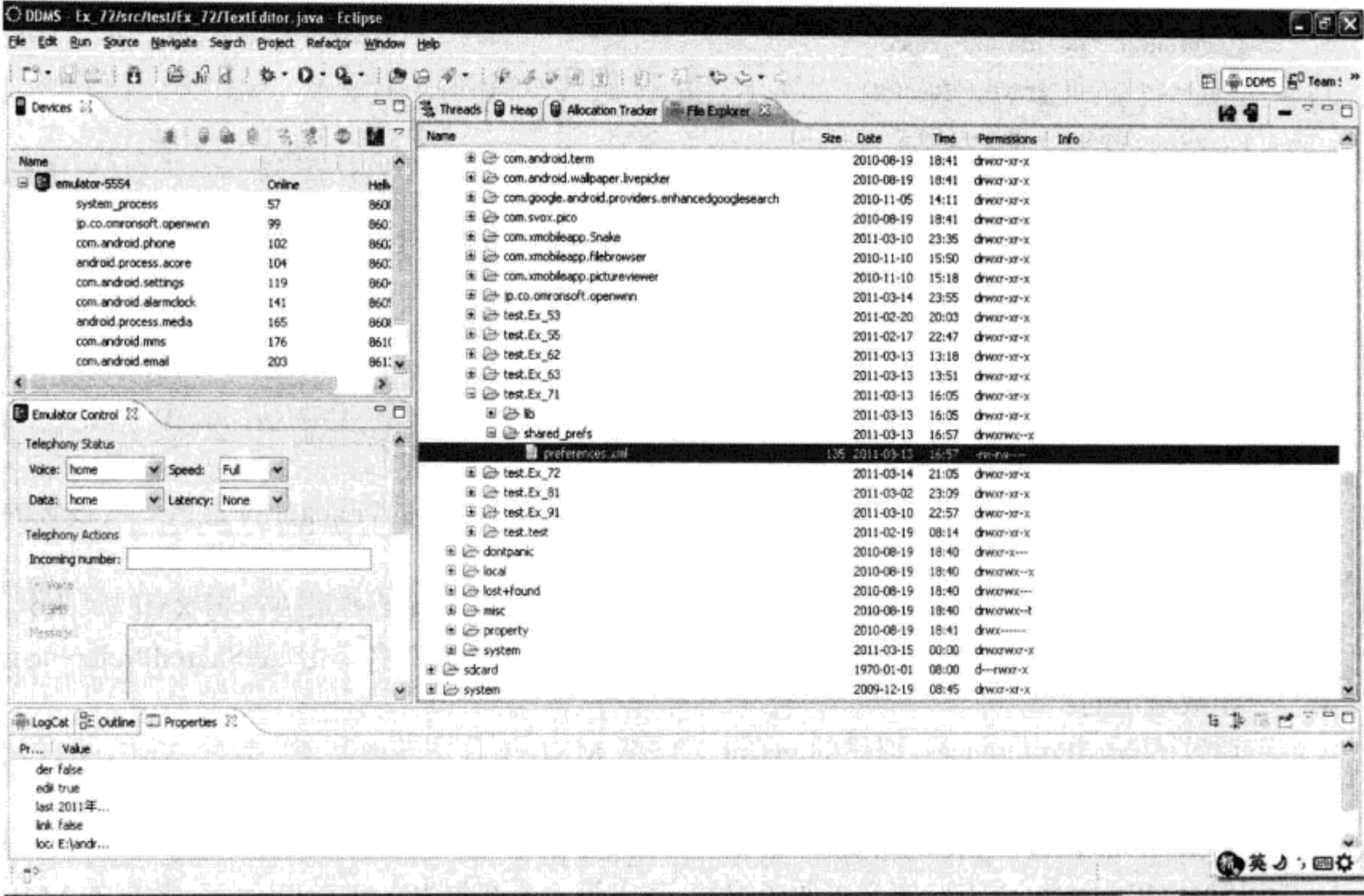


图 9-1 通过 DDMS 查看文件

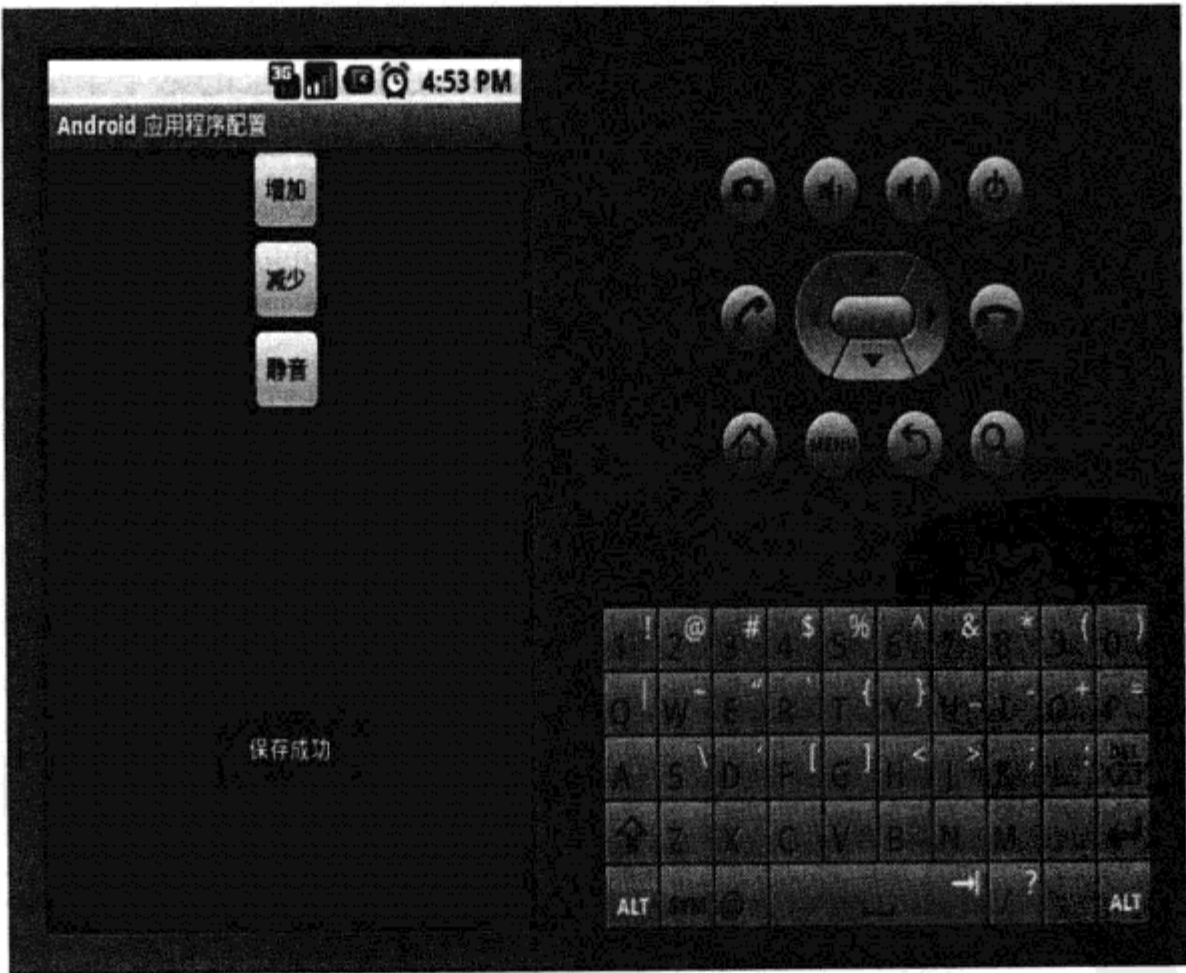


图 9-2 保存音量值





图 9-3 获取音量值

## 9.2 使用文件存储数据

### 9.2.1 访问应用中的文件数据

前面介绍了 Preferences 存储方式，使用这种方式可方便地存储数据。但其只适合存储比较简单的数据，如 int、string、float 格式数据，不适合存储复杂的数据。如果需要存储更复杂的数据，例如结构体或者一段文字，可选择的方式有好几种，这里先给读者介绍文件存储的方式。不同于 Preferences 存储，文件存储方式没有类型限制。和传统的 Java 中实现 I/O 的程序类似，Android 提供了 openFileInput 和 openFileOutput 方法读取设备上的文件。下面看一个创建输入和输出流的例子，具体如下所示：

```

String FILE = "myfile"; // 定义读取的文件名
FileOutputStream fileOutputStream= openFileOutput (FILE, Context.MODE_PRIVATE);
//根据指定模式创建文件输出流
FileInputStream fileInputStream = openFileInput (_NAME); //创建写入流
    
```

openFileOutput 方法使用 Context.MODE\_PRIVATE 模式创建文件输出流。Context.MODE\_PRIVATE 模式类似于类中的私有变量。在 Context.MODE\_PRIVATE 模式下，该文件只能被包含文件的应用读取。若其他应用读取该目录的文件，程序会抛出“找不到文件”的异常。对该模式文件进行读写时，原文件内容会被写入的数据覆盖。如果允许其他应用读写该文件，可以指定其模式为 Context.WORLD\_WRITEABLE 或者 Context.WORLD\_READABLE。下面以一个具体实例讲述文件存储方式的用法。

### 9.2.2 访问设备中独立的文件数据

【实例 9-2】简单的文本编辑器。



```

public class TextEditor extends Activity {
    /** Called when the activity is first created. */
    private EditText file_name; //定义输入文件名的文本框
    private EditText file_content; //定义输入文件内容的文本框
    private Button read_button; //定义读取文件的按钮
    private Button save_button; //定义保存文件的按钮
    public static int LONGTIME = Toast.LENGTH_LONG; //定义 Toast 消息显示的时间

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.texteditor);
        setContentView(R.layout.main); //使用 main.xml 初始化程序界面
        read_button = (Button) findViewById(R.id.read_file);
        save_button = (Button) findViewById(R.id.save_file);
        file_name = (EditText) findViewById(R.id.file_name);
        file_content = (EditText) findViewById(R.id.file_content);
        /*获取 XML 定义的控件对象 read_button、save_button、file_name 和 file_contentet*/

        save_button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) { //重载 OnClickListener 的 onClick 方法
                String str_file_name = file_name.getText().toString(); //获取保存的文件名
                if( str_file_name == "")//判断是否指定文件名
                    Toast.makeText(TextEditor.this, "文件名为空", LONGTIME).show();
                else
                {
                    String str_file_content = file_content.getText().toString(); //获取 file_contentet 的内容
                    try
                    {
                        save(str_file_name, str_file_content); //保存内容 str_file_content 到文件
                        str_file_name
                        Toast.makeText(TextEditor.this, "保存成功", LONGTIME).show();
                    }
                    catch (Exception e)
                    {
                        Toast.makeText(TextEditor.this, "保存失败", LONGTIME).show();
                    }
                }
                /*若保存成功，则显示保存成功的 Toast 消息，否则程序抛出异常，显示保存失败的 Toast 消息*/
            }
        })
    }
}

```



```

});
/*注册 save_button 单击事件监听器。当单击该按钮时，保存当前编辑的文件*/

read_button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) { //重载 OnClickListener 的 onClick 方法
        String str_file_name = file_name.getText().toString(); //获取读取的文件名
        if( str_file_name == "" ) //判断是否指定文件名
            Toast.makeText(TextEditor.this, "文件名为空", LONGTIME).show();
        else
        {
            try
            {
                String str_fiel_content = read(str_file_name); //读取文件 str_file_name 内容
                file_content.setText(str_fiel_content);
            }
            catch (Exception e)
            {
                Toast.makeText(TextEditor.this, R.string.file_read_failed, LONGTIME).show();
            }
            /*若保存成功，则显示保存成功的 Toast 消息，否则程序抛出异常，显示保存失败的 Toast 消息*/
        }
    }
});
}

/*注册 read_button 单击事件监听器。当单击该按钮时，根据文件名读取内容*/

public void save(String file, String fileContent) throws Exception {
    // Context 类提供了系统核心操作接口，它是 Activity 的父类
    FileOutputStream fileOutputStream = openFileOutput(file, Context.MODE_PRIVATE);
    //以 MODE_PRIVATE 模式获取指定文件的文件输出流
    fileOutputStream.write(fileContent.getBytes()); //将 fileContent 写入文件
}

/*方法 save 负责将指定的内容（fileContent）写入到文件*/

public String read(String file) throws Exception {
    byte[] buf = new byte[1024];
    int size = 0;
    FileInputStream fileInputStream = openFileInput(file); //获取指定文件的文件输入流
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();

```



```

//生成字节数据输入流对象，该对象实现按字节读取内容
while ((size = fileInputStream.read(buf)) > 0) {
    byteArrayOutputStream.write(buf, 0, size);
}
/*若 fileInputStream 有数据，则将其写入 ByteArrayOutputStream 对象 byteArrayOutputStream 中*/
return byteArrayOutputStream.toString();
}
/*方法 read 负责读取指定的文件*/
}

```

【代码说明】本例基于文件存储方式实现了简单的文本编辑器。对于文本编辑器，核心的操作是存储和读取文件。在本实例中，存储和读取操作分别由函数 save(String file, String fileContent) 和 read(String file)来实现。这两个函数使用 FileInputStream 和 FileOutputStream 对文件进行操作。

对于函数 save(String file, String fileContent) ，使用 FileOutputStream 进行数据存储操作。Save 函数定义了两个形参：其中第一个参数是文件名；第二个参数是文件内容。在 save 函数中，使用 openFileOutput(file, Context.MODE\_PRIVATE)创建文件输出流，该方法定了两个形参：其中第一个参数用于指定文件名称，创建的文件保存在/data/data/目录；第二个参数用于指定文件读写模式，如 MODE\_PRIVATE，这个模式是文件的默认模式，在该模式下写入的数据会覆盖原文件内容。Android 定义了四种文件读写模式，这四种模式由 Context 类定义。表 9-1 列举了这四种模式。

表 9-1 四种文件读写模式

模式	功能描述	整型值
Context.MODE_PRIVATE	私有模式：这种模式创建的文件是私有文件，因而创建的文件只能被应用本身访问。在该模式下，写入的内容会覆盖原文件的内容	0
Context.MODE_APPEND	附加模式：该模式会首先检查文件是否存在，若文件不存在，则创建新文件，若文件存在，则在原文件后追加内容	32968
Context.MODE_WORLD_READABLE	可读模式：表示当前文件可以被其他应用修改	1
Context.MODE_WORLD_READABLE	可读模式：表示当前文件可以被其他应用读取	2

创建的文件可通过单击 Eclipse 菜单“Window→Open Perspective→DDMS”找到，如图 9-4 所示。

在 Eclipse 中运行该程序，程序界面包含两个文本框和两个按钮，如图 9-5 所示。两个文本框分别用于输入文件名和文件内容，两个按钮分别用于触发读取和保存文件操作。程序启动后，在文件名文本框中输入“text1”，在文件内容文本框中输入“Hello android”，然后单击“保存”按钮，文件内容会被写入到指定的文件中。由于该文件不存在，Android 系统会创建一个文件名为“text1”的文件。

在文件浏览器中展开/data/data/test.Ex\_92 目录就可以看到该文件。图 9-6 显示了程序在目录/data/data/text.Ex\_92 下创建了一个“text1”文件。

默认情况下，使用 openFileOutput 方法创建的文件只能被其调用的应用使用，其他应用无法读取这个文件。如果需要在不同的应用中共享数据，可以使用 Content Provider 实现。关于 Content Provider 我们将在稍后的章节中介绍。



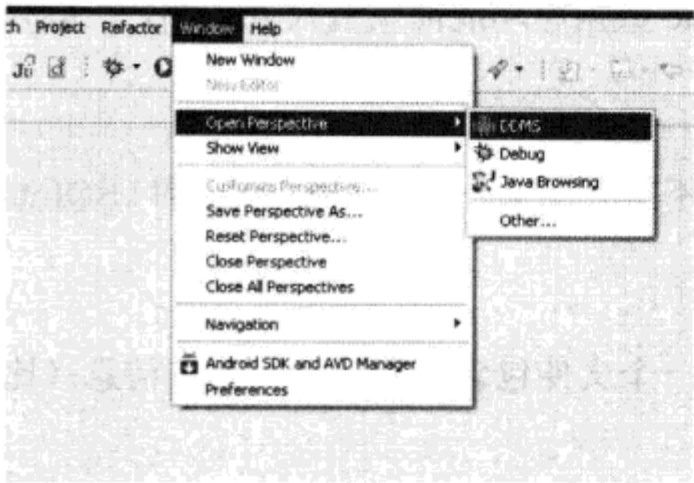


图 9-4 打开 DDMS

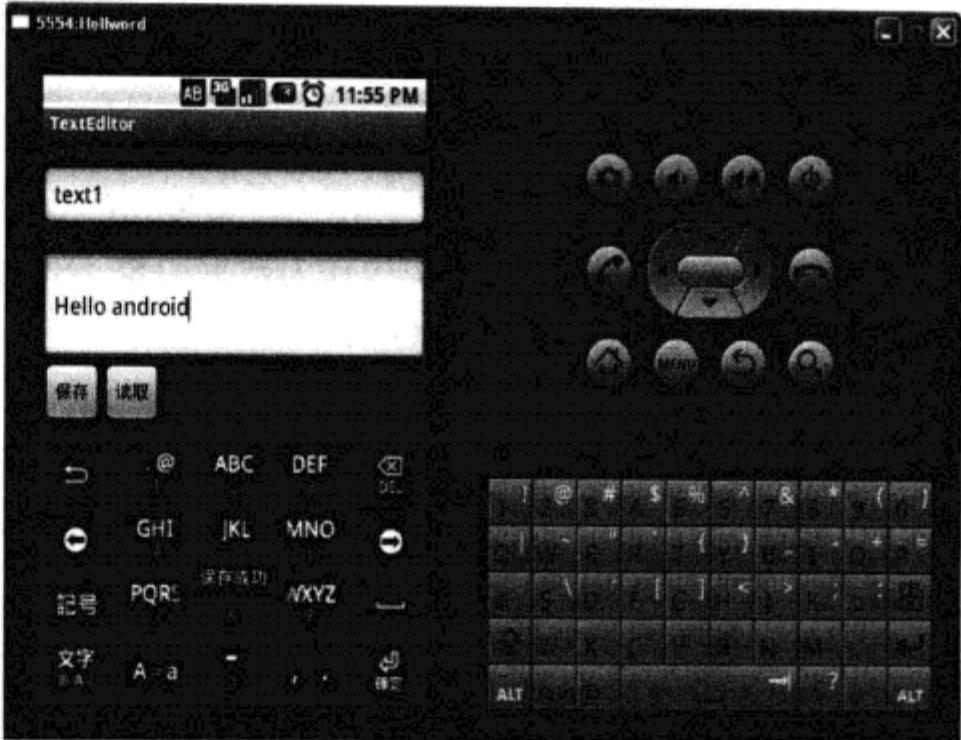


图 9-5 简单的文本编辑器界面

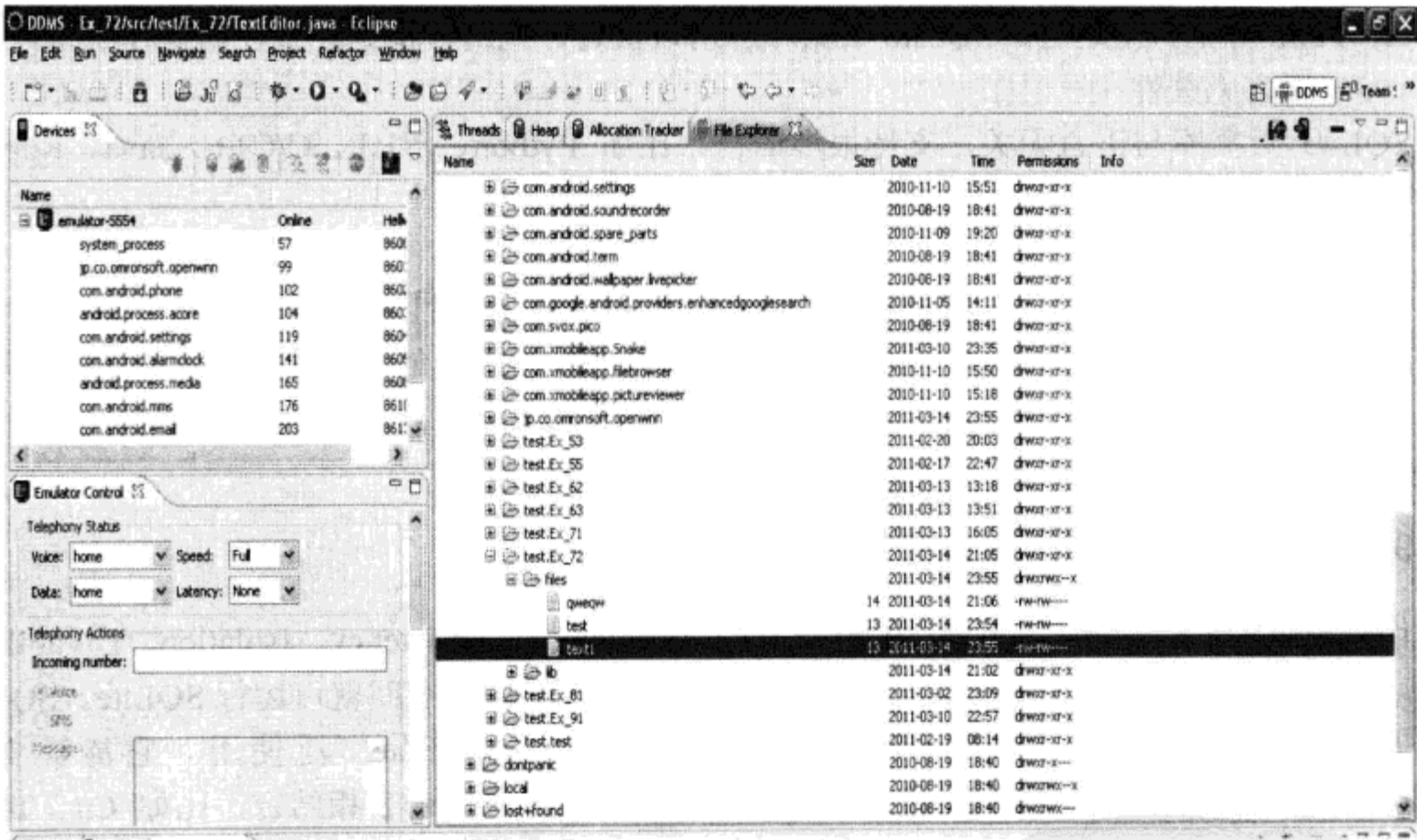


图 9-6 在文件浏览器中查看创建的文件

9.3 使用 SQLite 数据库存储数据

9.3.1 SQLite 数据库简介

除了 Preferences 和文件存储之外，Andorid 系统还支持数据库存储方式。相比 Preferences，该方式适合大数据量的数据存储，并且通过这种方式能够很容易地对数据库进行增加、插入、删除、更新等操作。

Android 使用 SQL 数据库——SQLite，作为存储数据库，SQLite 数据库是一种免费开源且底层无关的数据库。SQLite 数据库是基于 C 语言设计开发的开源数据库，最大支持 2048GB



数据。它具有如下特征:

#### (1) 轻量型数据库

大多数数据库的读写模型是基于 C/S 架构设计的,该架构下的数据库分为客户端和服务端。C/S 架构数据库是重量型的数据库,系统功能复杂且尺寸较大。SQLite 和 C/S 模式的数据库软件不同,它不使用分布式架构作为数据引擎。SQLite 数据库功能简单且尺寸较小,一般只需要带上一个 DDL 就可使用 SQLite 数据库。

#### (2) 不依赖第三方软件

SQLite 数据库与底层操作系统无关,其核心引擎既不需要安装,也不依赖任何软件。SQLite 几乎能在所有的操作系统上运行,具有较高的独立性。

#### (3) 方便管理和维护

SQLite 数据库具有较强的数据隔离性。SQLite 的一个文件包含了数据库的所有信息(比如表、视图、触发器),有利于数据的管理和维护。

#### (4) 可移植性

SQLite 数据库与底层系统无关,可以在大部分操作系统,如 Android、Windows Mobile、Sysbin、Palm 等平台上运行。SQLite 数据库应用可快速、无缝移植到几乎任何操作系统。

#### (5) 语言无关性

SQLite 数据库与语言无关,支持很多语言,比如 Python、.NET、C/C++、Java、Ruby、Perl 等,因而 SQLite 被开发者广泛使用。

#### (6) 事务性

SQLite 数据库采用独立事务处理机制,使用数据库的独占性和共享锁处理事务。这种方式规定必须获得该共享锁后,才能执行写操作。因而 SQLite 既允许数据库被多个进程并发读取,又保证最多只有一个进程写数据。这种方式可有效防止读脏数据、不可重复读、丢失修改等异常。

### 9.3.2 SQLite 数据库操作

作为轻量级的数据库,SQLite 遵守 ACID (Atomicity、Consistency、Isolation、Durability) 原则。SQLite 不要求系统提供太大的资源,仅不到 1MB 的内存空间就可运行 SQLite。SQLite 主要应用在小规模的嵌入式设备中,目前它被很多嵌入式产品广泛使用。它能够支持 Windows/Linux/UNIX 等主流的操作系统,同时能够跟很多程序语言相结合,比如 C#、Tcl、Java 等。SQLite 的处理速度较快,甚至不亚于 MySQL、PostgreSQL 等开源数据库系统。

SQLite 数据库存储了 Android 系统的许多应用信息,如联系人、来电记录、短信息记录等。为了方便存储、管理、查询数据,Android 向开发者开放了 SQLite 数据库相关的 API。通过这些 SQLite 的 API,开发者可实现对基本数据库、表以及记录的操作,包括数据库创建、数据库删除、表创建、表删除、记录插入、记录删除、记录更新、记录查询等。

### 9.3.3 使用 SQLiteDatabase 对象操作数据库

SQLite 提供了基本数据库、表以及记录的操作,下面讲述 Android 中 SQLite 数据库的基本操作。

#### 1. 数据库操作

##### (1) 数据库创建与打开



SQLite 提供了 `openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)` 方法打开指定的数据库。其中, `path` 指定数据库的路径; `factory` 用于构造查询时的游标, 若 `factory` 为 `null`, 则表示使用默认的 `factory` 构造游标; `flags` 指定数据库打开的模式, SQLite 定义了四种数据库打开模式。这四种模式分别为: `OPEN_READONLY` (以只读的方式打开数据库)、`OPEN_READWRITE` (以可读可写的方式打开数据库)、`CREATE_IF_NECESSARY` (首先检查数据库是否存在, 若不存在则创建数据库)、`NO_LOCALIZED_COLLATORS` (打开数据库时, 不按照本地化语言对数据进行排序)。若同时指定多个模式, 可使用 “|” 分隔开。也可使用 `openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)` 打开数据库。使用这种方式打开数据库时, 数据库不按照本地化语言对数据进行排序, 其作用同 `openDatabase(path, factory, CREATE_IF_NECESSARY)` 一样。创建 SQLite 数据库就是在文件系统中创建一个 SQLite 数据库的文件, 应用程序必须对创建数据库的目录有可写的权限, 否则会抛出异常。

下面是使用 `openDatabase` 方法打开指定的数据库代码:

```
/*使用 openDatabase 打开数据库*/
try {
    SQLiteDatabase sqliteDatabase = this.openDatabase("qdu_Student.db", null, NO_LOCALIZED_COLLATORS);
    //打开已经存在的数据库
}
catch (Exception e) {
    db = null;    //捕获创建数据库抛出的异常
}
/*若指定数据库不存在, openDatabase 方法抛出 FileNotFoundException 异常*/
```

下面是使用 `openOrCreateDatabase` 方法打开指定的数据库代码:

```
/*使用 openOrCreateDatabase 打开数据库*/
try {
    SQLiteDatabase sqliteDatabase = this.openOrCreateDatabase("qdu_Student.db", null);
    //打开已经存在的数据库
}
catch (Exception e) {
    db = null;    //捕获创建数据库抛出的异常
}
/*若指定数据库不存在, openDatabase 方法抛出 FileNotFoundException 异常*/
```

## (2) 数据库删除

SQLite 提供了 `deleteDatabase` 删除指定的数据库, 例如:

```
this.deleteDatabase("qdu_Student.db"); //删除数据库 qdu_Student.db
```

## (3) 数据库关闭

SQLite 提供了 `close` 方法关闭数据库, 例如:

```
sqliteDatabase.close(); //关闭数据库
```

## 2. 表操作

### (1) 表创建

数据库包含多个表, 每个表可存储多条记录。数据库创建之后, 下一步需要创建表。SQLite



没有提供专门的方法创建表, 通过 `execSQL` 方法并指定 SQL 语句创建表:

```
String SQL_CT = "CREATE TABLE student (ID INTEGER PRIMARY KEY, age INTEGER, name TEXT)";
/*创建表的 SQL 语句*/
sqliteDatabase.execSQL(SQL_CT); //执行该 SQL 语句创建表
```

### (2) 表删除

通过 `execSQL` 方法并指定 SQL 语句删除表:

```
String SQL_DROP_TABLE = "DROP TABLE student"; //删除表的 SQL 语句
sqliteDatabase.execSQL(SQL_DROP_TABLE); //删除表 student
```

## 3. 记录操作

### (1) 记录插入

每个表可存储多条记录, 有两种方式实现记录的插入。第一种是通过 `SQLiteDatabase` 的 `insert` 方法插入, 第二种是通过 `execSQL` 方法执行插入记录的 SQL 语句。使用 `insert` 方法插入记录时, 需要把插入记录的每个数据项放到 `android.content.ContentValues` 对象中。具体实现如下:

```
ContentValues contentValues = new ContentValues(); //创建 ContentValues 对象
contentValues.put(ID, 1); //将数据项 ID 放入 contentValues
contentValues.put(age, 26); //将数据项 age 放入 contentValues
contentValues.put(name, "StudentA"); //将数据项 name 放入 contentValues
sqliteDatabase.insert(student, null, contentValues); //在数据库 sqliteDatabase 的 student 表中插入记录
```

也可通过 `execSQL` 方法执行插入记录的 SQL 语句, 具体实现如下:

```
String SQL_INSERT = "INSERT INTO student (ID, age, name) values (1, 26, 'StudentA')";
//插入记录的 SQL 语句
sqliteDatabase.execSQL(SQL_INSERT); //在数据库 sqliteDatabase 的 student 表中插入记录
```

### (2) 记录更新

`SQLite` 提供了两种更新记录的方法: 第一种是通过 `SQLiteDatabase` 的 `update` 方法更新, 第二种是通过 `execSQL` 方法执行 SQL 更新记录的语句。使用 `update` 方法更新记录时, 需要把更新记录的每个数据项放到 `android.content.ContentValues` 对象中。 `update` 方法的原型为: `update(String table, ContentValues values, String whereClause, String[] whereArgs)`。下面我们修改 `StudentA` 的年龄为例, 说明 `update` 方法的使用:

```
ContentValues contentValues = new ContentValues(); //创建 ContentValues 对象
contentValues.put(ID, 1); //将数据项 ID 放入 contentValues
contentValues.put(age, 25); //将数据项 age 放入 contentValues
contentValues.put(name, "StudentA"); //将数据项 name 放入 contentValues
sqliteDatabase.update(student, contentValues, "name=StudentA", null);
//在数据库 sqliteDatabase 的 student 表中更新记录
```

也可通过 `execSQL` 方法执行更新记录的 SQL 语句, 具体实现如下:

```
String SQL_UPDATE = "UPDATE student SET age=25 where name='StudentA'"; //更新记录的 SQL 语句
sqliteDatabase.execSQL(SQL_UPDATE); //在数据库 sqliteDatabase 的 student 表中更新记录
```

### (3) 记录查询

`SQLite` 提供了两种查询记录的方法: 第一种是通过 `SQLiteDatabase` 的 `query` 方法查询, 第



二种是通过 execSQL 方法执行查询记录的 SQL 语句。SQLiteDatabase 提供了 6 种 query 方法用于不同方式的查询, 其中常用的方法是: public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit), 这个方法返回一个游标对象。下面我们以查询 StudentA 的记录为例, 说明 query 方法的使用:

```
sqliteDatabase.query(true, student, null, "name=StudentA", null, null, null, null, null);
//在数据库 sqliteDatabase 的 student 表中查询 StudentA 记录
```

也可通过 execSQL 方法执行查询记录的 SQL 语句, 具体实现如下:

```
String SQL_QUERY= "SELECT age, name from student WHERE name=\"StudentA\""; //定义删除 SQL 语句
sqliteDatabase.execSQL(SQL_QUERY); //在数据库 sqliteDatabase 的 student 表中查询记录
```

#### (4) 记录删除

SQLite 提供了两种删除记录的方法: 第一种是通过 SQLiteDatabase 的 delete 方法删除, 第二种是通过 execSQL 方法执行删除记录的 SQL 语句。使用 delete 方法删除记录时, 不需要把删除记录的每个数据项放到 android.content.ContentValues 对象中。update 方法的原型为: public int delete (String table, String whereClause, String[] whereArgs)。下面我们以删除 StudentA 记录例, 说明 delete 方法的使用:

```
sqliteDatabase.update(student, "name=StudentA", null);
//在数据库 sqliteDatabase 的 student 表中删除 StudentA 记录
```

### 9.3.4 Cursor 的使用

在数据库中, 游标 (Cursor) 是一个十分重要的概念。游标提供了一种对从表中检索出的数据进行操作的灵活手段。就本质而言, 游标实际上是一种能从包括多条数据记录的结果集中每次提取一条记录的机制。游标总是与一条 SQL 选择语句相关联, 因为游标由结果集 (可以是零条、一条或由相关的选择语句检索出的多条记录) 和结果集中指向特定记录的游标位置组成。当决定对结果集进行处理时, 必须声明一个指向该结果集的游标。如果曾经用 C 语言写过对文件进行处理的程序, 那么游标就像打开文件所得到的文件句柄一样。只要文件打开成功, 该文件句柄就可代表该文件。对于游标而言, 其道理是相同的。可见游标能够按传统程序读取平面文件类似的方式处理来自基础表的结果集, 从而把表中数据以平面文件的形式呈现给程序。游标允许应用程序对查询语句 SELECT 返回的行结果集中每一行进行相同或不同的操作, 而不是一次对整个结果集进行同一种操作, 可以基于游标位置对表中数据进行删除或更新, 而且正是游标把作为面向集合的数据库管理系统和面向行的程序设计两者联系起来, 使两个数据处理方式能够进行沟通。

游标是系统为用户开设的一个数据缓冲区, 存放 SQL 语句的执行结果。每个游标区都有一个名字。用户可使用 SQL 语句逐一从游标中获取记录, 赋给主变量并交由主语言进一步处理。主语言是面向记录的, 一组主变量一次只能存放一条记录。仅使用主变量并不能完全满足 SQL 语句向应用程序输出数据的要求。嵌入式 SQL 引入了游标的概念, 用来协调这两种不同的处理方式。在数据库开发过程中, 当检索的数据只是一条记录时, 所编写的事务代码往往使用 SELECT INTO 语句。但是我们常常会从某一结果集中逐一地读取一条记录。那么如何解决这种问题呢? 游标为我们提供了一种极为优秀的解决方案。

游标可分为三种类型: Transact\_SQL 游标、API 服务器游标和客户游标。Andorid 系统提



供 Cursor 作为游标类，表 9-2 列出了 Cursor 类的主要方法。

表 9-2 Cursor 类的主要方法

方法	功能描述	返回值
move (int offset)	以当前的位置为基准,将 Cursor 移动到偏移量为 offset 的位置。若移动成功返回 true,失败返回 false。注意,offset 为正值时,游标向前移动;为负值时,向后移动	boolean
moveToPosition (int position)	将 Cursor 移动到绝对位置 position 处。若移动成功返回 true,失败返回 false。注意,moveToPosition 移动到一个绝对位置,而 move 移动以当前位置为基准	boolean
moveToNext ()	将 Cursor 向前移动一个位置。若移动成功返回 true,失败返回 false。其功能等同于 move (1)	boolean
moveToLast ()	将 Cursor 移动到最后一条记录。若移动成功返回 true,失败返回 false。若当前记录数为 count,则其功能等同于 moveToPosition (count)	boolean
moveToFisrt ()	将 Cursor 移动到第一条记录。若移动成功返回 true,失败返回 false。其功能等同于 moveToPosition (1)	boolean
isBeforeFirst()	判断 Cursor 是否指向第一项数据之前。若指向第一项数据之前,返回 true,否则返回 false	boolean
isAfterLast()	判断 Cursor 是否指向最后一项数据之后。若指向最后一项数据之后,返回 true,否则返回 false	boolean
isClosed ()	判断 Cursor 是否关闭。若 Cursor 关闭,返回 true,否则返回 false	boolean
isFirst ()	判断 Cursor 是否指向第一项记录	boolean
isLast()	判断 Cursor 是否指向最后一条记录	boolean
isNull (int columnIndex)	判断指定的位置 columnIndex 的记录是否存在	boolean
getCount ()	获取当前表的行数,即记录总数	int

下面以个人信息管理系统为例，为读者讲述 SQLite 的用法。

【实例 9-3】个人信息管理系统。

```

private LayoutParams layoutParams; //布局参数
private final static int DB_MODE = MODE_PRIVATE; //数据库读取模式
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    linearLayout = new LinearLayout(this); //创建 LinearLayout 对象
    listView = new ListView(this); // 创建 ListView 对象
    layoutParams = new LinearLayout.LayoutParams(LinearLayout.LayoutParams.FILL_PARENT,
        LinearLayout.LayoutParams.WRAP_CONTENT); //设置布局的内容填充方式

    linearLayout.setOrientation(LinearLayout.VERTICAL); //设置线性布局的元素方向为垂直分布
    linearLayout.setBackgroundColor(android.graphics.Color.GREEN); //设置布局背景颜色为绿色
    listView.setBackgroundColor(android.graphics.Color.BLACK); //设置列表视图背景颜色为黑色
    linearLayout.addView(listView, layoutParams); //在 linearLayout 中添加 listView
    setContentView(linearLayout); //使用 linearLayout 初始化程序界面 */
    sqliteDatabase = this.openOrCreateDatabase(DB, DB_MODE, null); // 打开已经存在的数据库
    try
    {
        /* 在数据库 sqliteDatabase 中创建一个表 */
        sqliteDatabase.execSQL(CREATE_TABLE);
    }
}

```



```

    }
    //获取数据库 Phones 的 Cursor
    catch (Exception e)
    {
        UpdateData(); //将数据更新到 listView 中
    }
}

public boolean onKeyUp(int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_1:
            OP_Record_CREATE();    //插入记录
            break;
        case KeyEvent.KEYCODE_2:
            OP_Record_DELETE(); //删除记录
            break;
        case KeyEvent.KEYCODE_3:
            OP_Record_UPDATE();    //更新数据库
            break;
        case KeyEvent.KEYCODE_4:
            OP_Tab_DELETE(); //删除表
            break;
        case KeyEvent.KEYCODE_5:
            OP_DB_DELETE(); //删除数据库
            break;
    }
    /*根据按钮决定执行的数据库操作：
    * 按下按钮 1--插入一条数据库记录
    * 按下按钮 2--删除一条数据库记录
    * 按下按钮 3--更新数据库记录
    * 按下按钮 4--删除表
    * 按下按钮 5--删除数据库*/
    return true;
}

/*处理按钮操作方法，当用户释放按钮时，该方法被调用*/

```



```

public void OP_DB_DELETE()
{
    this.deleteDatabase(DB); //删除 DB
    this.finish();
}

/* 删除数据库 */

public void OP_Tab_DELETE()
{
    sqliteDatabase.execSQL("DROP TABLE " + PI_Table); //删除表 PI_Table
    this.finish();
}

/* 删除一个表 */

/* 更新一条数据 */
public void OP_Record_UPDATE()
{
    ContentValues contentValues = new ContentValues(); //创建 ContentValues 对象存储记录
    contentValues.put(PI_Table_Name, "Jim");
    contentValues.put(PI_Table_Age, 23);
    contentValues.put(PI_Table_Gender, "F");
    sqliteDatabase.update(PI_Table, contentValues, "PI_Table_Age" + "=" + 24, null); // 更新数据
    UpdateData(); //将数据更新到 listView 中
}

public void OP_Record_CREATE()
{
    ContentValues contentValues = new ContentValues(); //创建 ContentValues 对象存储记录
    contentValues.put(PI_Table_Name, "John");
    contentValues.put(PI_Table_Age, 21);
    contentValues.put(PI_Table_Gender, "F");

    sqliteDatabase.insert(PI_Table, null, contentValues); //插入 contentValues 定义的记录
    UpdateData(); //将数据更新到 listView 中
}

/* 向表中添加一条记录 */
public void OP_Record_DELETE()
{
    sqliteDatabase.execSQL("DELETE FROM " + PI_Table + " WHERE PI_Table_ID=" + Integer.toString(pointer));
    //删除数据
    pointer--;
}
    
```



```

        if (pointer < 0)
        {
            pointer = 0;
        }
        UpdateData(); //将数据更新到 listView 中
    }
    /* 从表中删除指定的一条数据 */

    public void UpdateData()
    {
        Cursor cur = sqliteDatabase.query(PI_Table, new String[] { PI_Table_ID, PI_Table_Name, PI_Table_Age,
        PI_Table_Gender }, null, null, null, null, null); // 获取数据库 sqliteDatabase 的 Cursor
        pointer = cur.getCount(); //返回记录的总数
        if (cur != null && cur.getCount() >= 0)
        {
            ListAdapter adapter = new SimpleCursorAdapter(this, android.R.layout.simple_list_item_2,
            cur, new String[] { PI_Table_Name, PI_Table_Age, PI_Table_Gender },
            new int[] { android.R.id.text1, android.R.id.text2 });
            /*创建 ListAdapter 对象, 该对象用于关联 ListView 和后台数据*/
            listView.setAdapter(adapter);
            /* 将 adapter 添加到 listView 中 */
        }
    }
    /* UpdateData 更新视图显示, 将数据更新到 listView */

    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        if (keyCode == KeyEvent.KEYCODE_BACK)
        {
            sqliteDatabase.close(); //退出时, 关闭数据库
            this.finish(); //结束应用
            return true; //返回结果
        }
        return super.onKeyDown(keyCode, event);
    }
    /* 按钮事件处理 */

```

**【代码说明】**本例使用 SQLiteDatabase 开放的 API 实现了一个简单的个人信息管理系统。查看/data/data/text.Ex\_93/databases 目录, 该目录下包含所创建的数据库, 如图 9-7 所示。



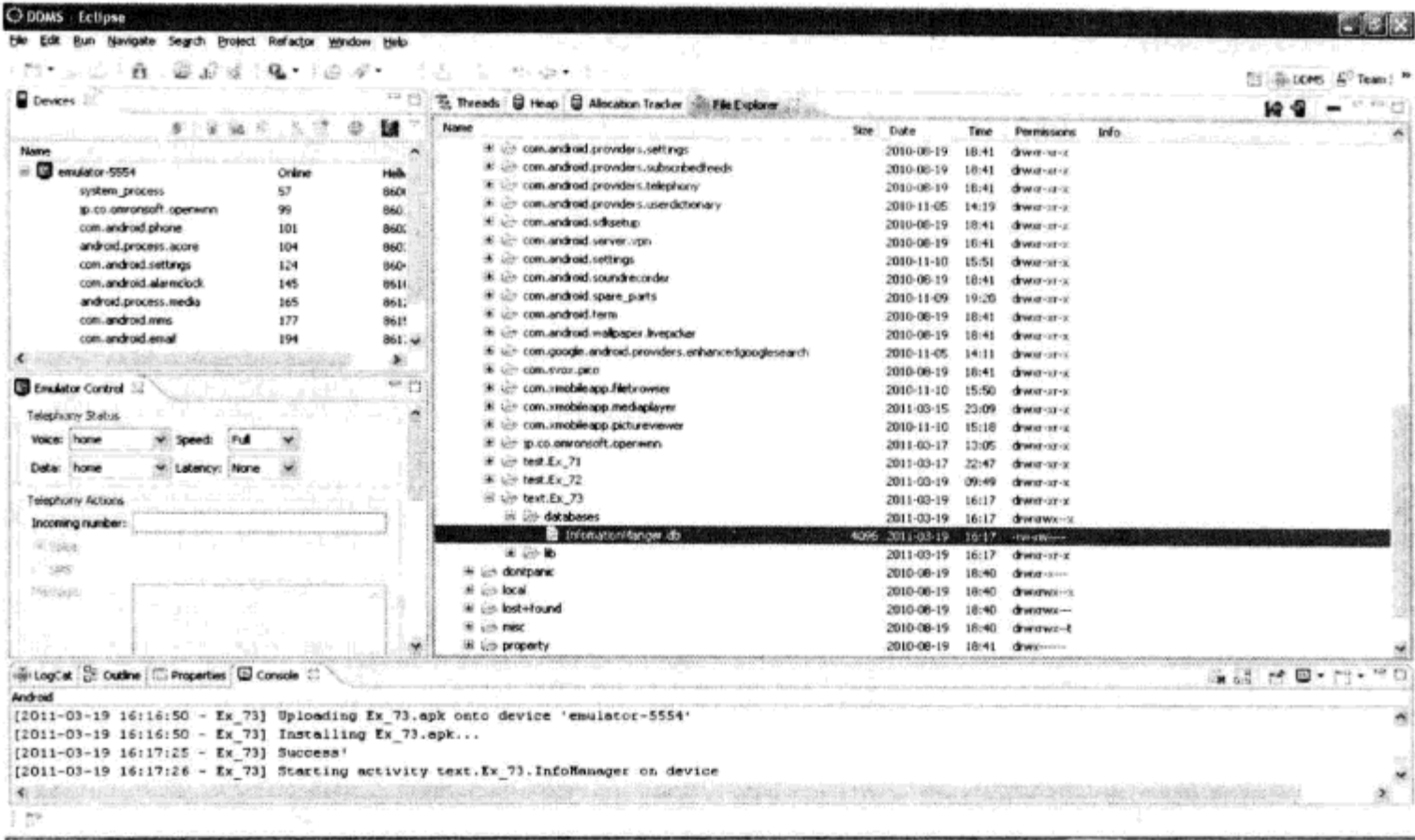


图 9-7 查看 databases 目录

## 9.4 使用 ContentProvider

### 9.4.1 定义 ContentProvider

ContentProvider（内容提供器）属于 Android 应用程序的组件之一，作为应用程序之间唯一的共享数据的途径，ContentProvider 主要的功能就是存储、检索数据以及向其他应用程序提供访问数据的接口。Android 系统为一些常见的数据类型（如音乐、视频、图像、手机通信录、联系人信息等）内置了一系列的 ContentProvider，这些都位于 android.provider 包下。通过持有特定的许可，可以在开发的应用程序中访问这些 ContentProvider。

和其他应用程序共享数据有两种方式：第一种是创建自己的 ContentProvier（即继承自 ContentProvider 类的子类），第二种是将自己的数据添加到已有的 ContentProvider 中去。后者需要保证现有的 ContentProvider 和自己的数据类型相同且具有该 ContentProvider 的写入权限。对于 ContentProvider，最重要的两个概念是数据模型（data model）和 URI。

#### 1. 数据模型

ContentProvider 将其存储的数据以数据表的形式提供给访问者，在数据表中每一行对应一条记录，每一列为具有特定类型和意义的数据。每一条数据记录都包括一个“\_ID”数值字段，用于唯一标识一条记录。

#### 2. URI

每一个 ContentProvider 都对外提供一个能够唯一标识自己数据集（data set）的公开 URI。如果一个 ContentProvider 管理多个数据集，其将会为每个数据集分配一个独立的 URI。所有的 ContentProvider 的 URI 都以“content://”开头，其中“content:”用来标识 ContentProvider 管理的 schema。在几乎所有的 ContentProvider 的操作中都会用到 URI，一般来讲，若自己开发 ContentProvider，最好将 URI 定义为常量，这样在简化开发的同时也提高了代码的可维护性。



访问 `ContentProvider` 中的数据主要通过 `ContentResolver` 对象, `ContentResolver` 类提供了可以用来对 `ContentProvider` 中的数据进行查询、插入、修改和删除等操作的成员方法。以查询为例, 查询一个 `ContentProvider` 需要掌握如下信息: 唯一标识 `ContentProvider` 的 URI、需要访问的数据字段名称和该数据字段的数据类型。

#### 9.4.2 使用 `ContentProvider` 进行 CRUD 操作

查询 `ContentProvider` 的方法有两个: `ContentResolver.query()` 和 `Activity.managedQuery()`, 二者接收的参数均相同且返回的都是 `Cursor` 对象。唯一不同的是, 使用 `managedQuery()` 方法可以用 `Activity` 来管理 `Cursor` 的生命周期。Android 提供了一个 `SQLiteDatabase` 类, 该类封装了一些操作数据库的 API, 使用该类可以完成数据的添加 (Create)、查询 (Retrieve)、更新 (Update) 和删除 (Delete) 操作 (这些操作简称为 CRUD)。 `SQLiteDatabase` 包含两个重要方法: `execSQL()` 和 `rawQuery()`。 `execSQL()` 方法可以执行 INSERT、DELETE、UPDATE 和 CREATE TABLE 之类的有更改行为的 SQL 语句; `rawQuery()` 方法可以执行 SELECT 语句。 `SQLiteDatabase` 还专门提供了对应于添加、删除、更新、查询的操作方法: `insert()`、`delete()`、`update()` 和 `query()`。

无论 `query()` 还是 `managedQuery()`, 它们的第一个参数都是 `ContentProvider` 的 `URI-CONTENT_URI` 常量。这个常量用来标识某个特定的 `ContentProvider` 和数据集 (参见前面的 URI)。为了限制只能对一条记录进行查询, 可以在 URI 后面扩展这个记录的 `_ID` 值, 也就是在 URI 路径部分的最后加上匹配这个 ID 的字符串。

有一些辅助方法, 如 `ContentUris.withAppendedId()` 和 `Uri.withAppendedPath()`, 可以很简单地在 URI 中添加 ID。比如要在联系人数据库中查找记录 41, 可能需要构造如下的查询语句:

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI for the contact with _ID == 41.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 41);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri uri = Uri.withAppendedPath(People.CONTENT_URI, 41);

// Then query for this specific record:
Cursor cur = managedQuery(uri, null, null, null, null);
```

更多的查询细节可以通过 `query()` 和 `managedQuery()` 方法的其他参数进行限定。下面是获取一个联系人名字和首选电话号码列表的例子:

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
```



```

        People._ID,
        People._COUNT,
        People.NAME,
        People.NUMBER
    };

    // Get the base URI for the People table in the Contacts content provider.
    Uri contacts = People.CONTENT_URI;

    // Make the query.
    Cursor managedCursor = managedQuery(contacts,
        projection, // Which columns to return
        null,        // Which rows to return (all rows)
        null,        // Selection arguments (none)
        // Put the results in ascending order by name
        People.NAME + " ASC");
    
```

这个查询从联系人内容提供者中获取了数据。它得到名字、首选电话号码，以及每个联系人的唯一记录 ID。同时它在每个记录的 `_COUNT` 字段记录返回的记录数目。列名的常量被定义在不同的接口中：`_ID` 和 `_COUNT` 定义在 `BaseColumns` 里，`NAME` 定义在 `PeopleColumns` 里，`NUMBER` 定义在 `PhoneColumns` 里。由于 `Contacts.People` 类已经实现了这些接口，上面的代码只需要使用类名就可以引用这些接口。

查询返回的结果是一个集合，这个集合可以由零个或更多数据库记录组成。所有内容提供者都有一个包含了每条记录的唯一 ID 的 `_ID` 列。另外，记录数目可以通过 `_COUNT` 列返回，这是所有的内容提供者都可以做到的。

通过一个游标 `Cursor` 对象就可以把获取到的数据显示出来，如果想在结果集中前后浏览数据，可以借助于游标实现。

- 如果执行增加、修改和删除等操作，必须使用 `ContentResolver` 对象。
- 如果访问结果记录集，则可以通过查询返回的游标对象来实现。
- 如果查询操作是由一个指定的 ID 来执行的，那么这个集合里只有一个值，否则查询返回多个数值。

数据的读取可以通过表格中的特定字段来实现，但前提是需要知道这个字段的数据类型。游标对象针对不同的数据类型有不同的读取方法，比如 `getString()`、`getInt()` 和 `getFloat()`。如果调用 `getString()` 方法，则游标对象将这个数据的字符串表示作为结果返回。

游标可以实现按列名索引，下面的代码片段演示了如何从前述查询结果中读取名字和电话号码：

```

import android.provider.Contacts.People;

private void getColumnData(Cursor cursor){
    if (cursor.moveToFirst()) {
    
```



```
String name;
String phone;
int nindex= cursor.getColumnIndex(People.NAME);
int pindex = cursor.getColumnIndex(People.NUMBER);
String imagePath;

do {
    // Get the field values
    name = cursor.getString(nindex);
    phone= cursor.getString(pindex);

    //TODO
    ...

} while (cursor.moveToNext());
}
```

如果一个查询返回的是二进制数据,那么这个数据可直接被输入到表格或表格条目中,也可由 `content:URI` 字符串来获取数据。一般而言,直接存放到表格中的数据都比较小(例如,20~50KB 或更小),可以通过调用 `Cursor.getBlob()` 来获取。

当表格条目是一个 `content:URI` 时,不应该试图直接打开和读取该文件,可能会由于权限问题而导致操作失败。应该首先调用 `ContentResolver.openInputStream()` 获得一个 `InputStream` 对象,然后再使用 `InputStream` 对象来读取数据。可以通过下面的方法修改保存在内容提供器中的数据:

- 新记录的增加。
- 将新的数据添加到已有的记录。
- 对已有记录进行批量更新。
- 记录的删除。

通过使用 `ContentResolver.openInputStream()` 方法可以实现数据的修改操作。对于一些内容提供器来说,写数据的权限约束比读数据更强。在一个没有写权限的内容提供器上执行 `ContentResolver.openInputStream()` 方法,结果是失败的。如果要为一个内容提供器增加一个新的记录,需进行以下操作:

首先在 `ContentValues` 对象里构建一个“键-值”对映射,这里每个键跟内容提供器的列名是一致的,而值是新记录里特定列所期望的值。然后调用 `ContentResolver.insert()` 方法,并给它发送这个 `ContentValues` 映射图和内容提供器的 `URI`。通过这个方法可以将新记录的 `URI` 全名返回,该 `URI` 内容是由内容提供器的 `URI` 加上该新记录的扩展 ID 得到的。

可以通过该 `URI` 来查询并获得新记录上的一个游标,然后可以对这个记录做进一步的修改。下面是一个例子:

```
import android.provider.Contacts.People; //导入联系人类
```



```
import android.content.ContentResolver; //导入内容解析器
import android.content.ContentValues; //导入 ContentValues

ContentValues contentValues = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
contentValues.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, contentValues);
```

如果记录已经存在，就可以添加新的信息或修改已有信息。上例中的下一步就是添加联系人信息，如一个电话号码或一个即时通信 IM 或电子邮箱地址到新的条目中。在联系人数据库中增加一条记录的最好的办法是在该记录 URI 后扩展表名，然后通过这个修正的 URI 来添加新的数据值。

为此，每个联系人表显示一个 `CONTENT_DIRECTORY` 常量的表名。接上面的例子，下面的代码可以实现为刚刚创建的记录添加一个电话号码和电子邮件地址：

```
Uri puri= null;
Uri euri= null;
values.clear();
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, "1233214567");
getContentResolver().insert(phoneUri, values);
euri= Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);
values.clear();
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
values.put(People.ContactMethods.DATA, "test@example.com");
values.put(People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
getContentResolver().insert(emailUri, values);
```

如果想把少量的二进制数据放到一张表格里，可以通过调用接收字节流的 `ContentValues.put()` 来实现。如果需要添加大量的二进制数据，比如一首完整的歌曲或一张照片，则需要在表里存放到该数据的 `content:URI`，然后调用 `ContentResolver.openOutputStream()` 方法，由该文件的 URI 来完成。

内容提供者把数据保存在一个文件里并且在记录的一个隐藏字段中记录文件路径。考虑到这个原因，Android 提供了 `MediaStore` 内容提供者。该内容提供者主要用于实现分发图像、音频和视频数据，该内容提供者规定了获取关于这个二进制数据的元信息的 `query()` 或 `managedQuery()` 方法使用的 URI，同样可以使用 `openInputStream()` 方法获取数据。类似的，`insert()` 方法使用的 URI，同样可以被 `openOutputStream()` 方法用来存放二进制数据，`insert()` 是实现把元信息放进一个 `MediaStore` 记录里的方法。为了说明这个使用方式，看一下下面的代码片段：



```
import android.provider.MediaStore.Images.Media; //导入媒体类
import android.content.ContentValues; //导入 ContentValues 类
import java.io.OutputStream; //导入输出流

ContentValues contentValues= new ContentValues(3);
values.put(Media.DISPLAY_NAME, "road_trip_1");
values.put(Media.DESCRPTION, "Day 1, trip to Los Angeles");
values.put(Media.MIME_TYPE, "image/jpeg");

Uri uri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, values);
try {
    OutputStream outputStream = getContentResolver().openOutputStream(uri);
    sourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outputStream);
    outputStream.close();
} catch (Exception e) {
    Log.e(TAG, "exception while writing image", e);
}
```

- 如果要批量对一组记录（例如，把所有字段中的“NY”改为“New York”）进行更新操作，可以调用 `ContentResolver.update()` 方法，通过给该方法传递需要改变的列和值参数来实现记录的更新。
- 如果要对单个记录执行删除操作，可以调用 `ContentResolver.delete()` 方法。可以通过给该方法传递一个特定行的 URI 参数来实现单位记录的删除。
- 如果要对多行记录执行删除操作，可以通过传递需要被删除的记录类型的 URI 参数来调用 `ContentResolver.delete()` 方法（例如，`android.provider.Contacts.People.CONTENT_URI`）以及可以由一个 SQL WHERE 语句来确定要删除的行。

【实例 9-4】使用 ContentProvider 访问联系人信息。

```
public class CPApplication extends Activity {
    Uri contact_uri = ContactsContract.Contacts.CONTENT_URI; //联系人的 URI
    TextView textview; //声明 TextView 对象
    int textcolor = Color.GREEN; //定义文本颜色
    String [] contact = {
        ContactsContract.Contacts.DISPLAY_NAME, //联系人名字
        ContactsContract.Contacts._ID, //联系人记录的键值
    };

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //根据 main.xml 设置程序 UI
        textview = (TextView)findViewById(R.id.textview); //创建 textview 对象
        String result = getContactInfo(); //获取联系人的信息
        textview.setTextColor(textcolor); //设置文本框的颜色
    }
}
```



```

        textview.setTextSize(20.0f); //定义字体大小
        textview.setText("记录\t 名字\n" + result); //设置文本框的文本
    }

    public String getContactInfo(){
        String result = "";
        ContentResolver resolver = getContentResolver();//获取 ContentResolver 对象
        Cursor cursor = resolver.query(contact_uri, contact, null, null, null); //查询联系人
        int idIndex = cursor.getColumnIndex(ContactsContract.Contacts._ID);//获得_ID 字段的索引
        int nameIndex = cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME);
        //获得 Name 字段的索引
        int phoneIndex = cursor.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER);
        //获得 HAS_PHONE_NUMBER 字段的索引
        for (cursor.moveToFirst();(!cursor.isAfterLast());cursor.moveToNext()) {
            result = result + cursor.getString(idIndex) + "\t\t\t";
            result = result + cursor.getString(nameIndex)+ "\t\n";
        }
        //遍历 Cursor 提取数据
        cursor.close(); //使用 close 方法关闭游标
        return result; //返回结果
    }

    //getContactInfo 获取联系人列表的信息，返回 String 对象
}

```

**【代码说明】**本实例使用 ContentProvider 访问联系人信息。在 Android 中，用户可使用 Contact 应用来添加、修改、编辑以及删除联系人。

图 9-8 显示了 Android 手机中联系人的列表。

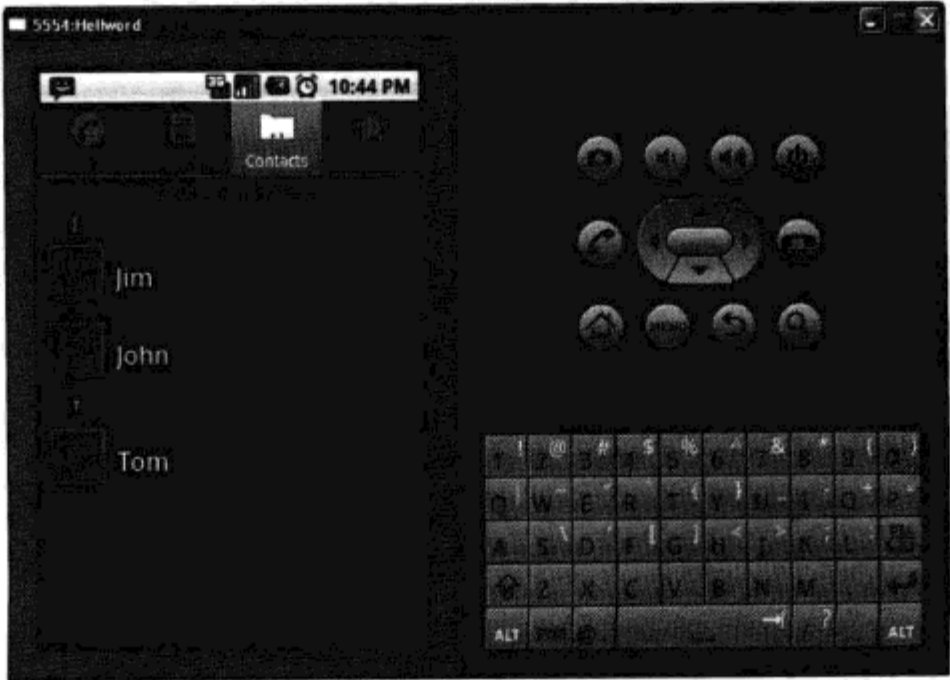


图 9-8 Contact 中的联系人列表



Contact 是一个 SQLite 数据库，用于存放联系人信息。对 Contact 的操作结果最终要存储到一个 SQLite 数据库中，其位于/data/data/provides/contacts/目录下。图 9-9 显示了 Contact 数据库的存放位置。

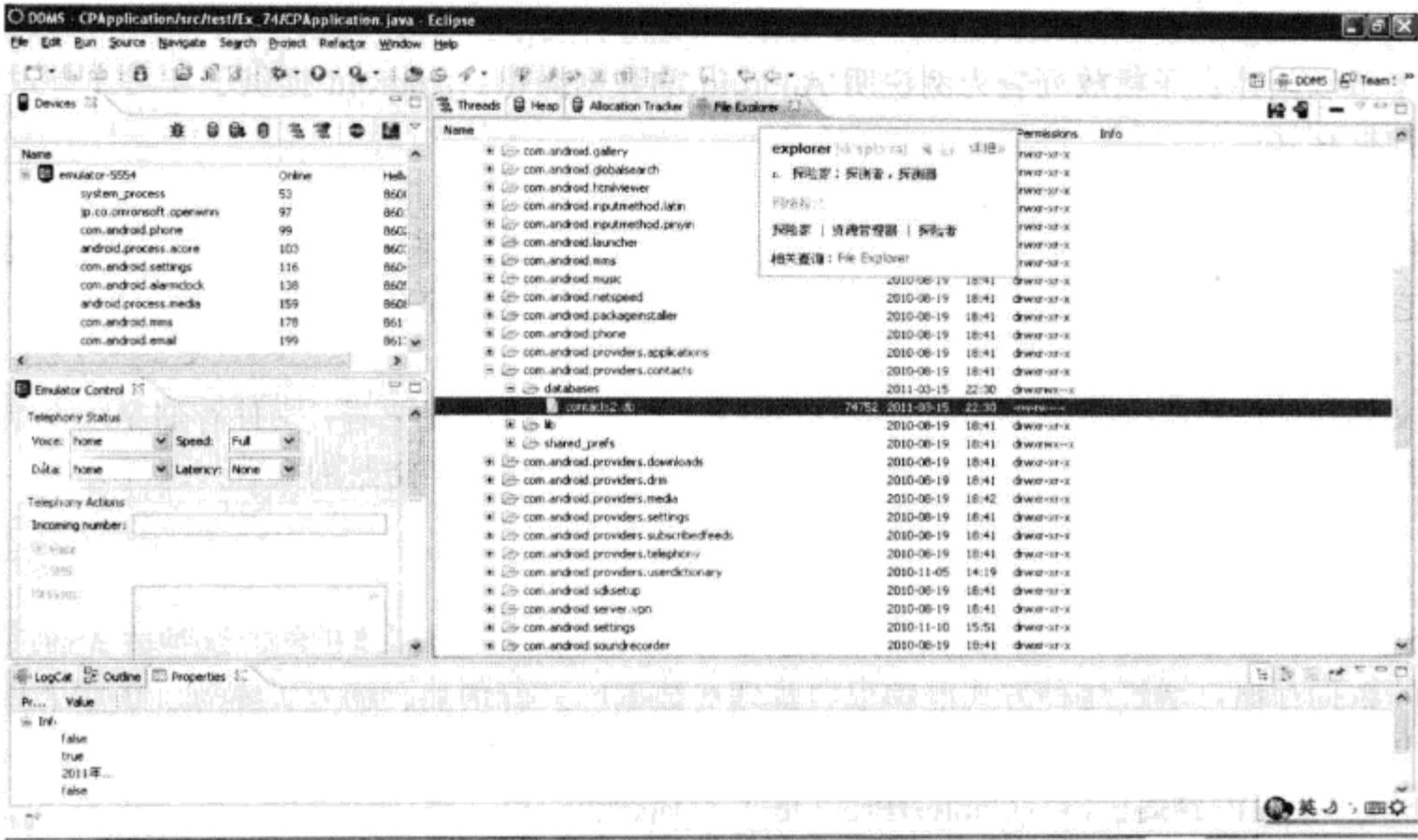


图 9-9 Contact 数据库

需要获取 ContentResolver 对象才能查询联系人，ContentResolver 提供了 query()方法根据 URI 查询 ContentProvider 的信息。执行完 query()方法后，会返回一个游标对象。通过该对象可获取表对应字段的索引，游标类通过 getString 方法获取该索引对应的值。

程序启动后，会读取并显示手机中所有联系人记录的 ID 以及名字。程序运行结果如图 9-10 所示。

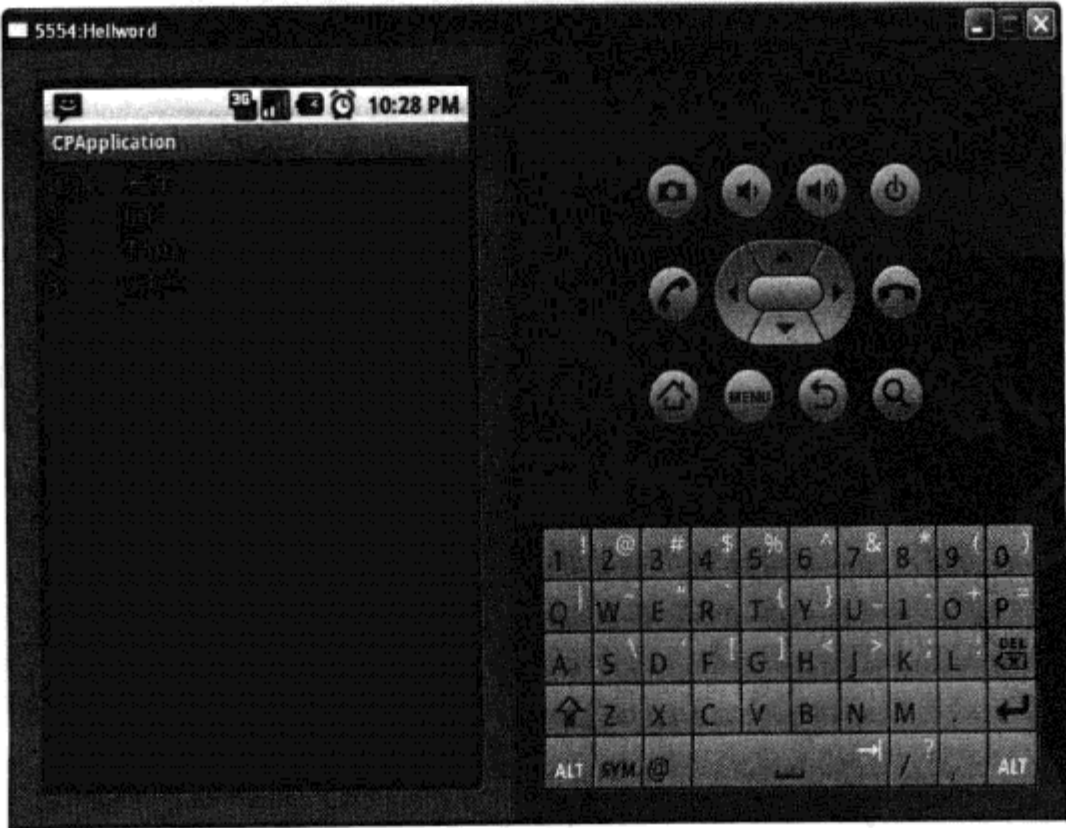


图 9-10 使用 ContentProvider 访问联系人信息



## 9.5 应用实例详解：创建音乐播放列表

### 9.5.1 实例分析

本节以创建音乐播放列表为例说明 Android 的数据操作。Android 提供了 4 种主要的方式实现数据存储：

#### (1) 使用 Preferences 存储数据

Preferences 采用“键-值”对组织和管理数据，其数据存储在 XML 文件中。相对于其他方式，它是一个轻量级的存储机制。该方式实现比较简单，适合简单数据的存储。

#### (2) 使用文件存储数据

文件存储的特点介于 Preferences 与 SQLite 之间。从存储量来看，文件存储是一个“重量级”存储机制，比 Preferences 方式更适合存储较大的数据。从存储结构来看，这种方式不同于 SQLite，不适合结构化的数据存储。

#### (3) 使用数据库 (SQLite) 存储数据

SQLite 使用数据库作为存储介质，它是一个“重量级”的存储机制，这种方式适合大数据量的数据存储，通过这种方式能够很容易地对数据库进行增加、插入、删除、更新等操作。相比 Preferences 和文件存储，这种方式实现较为复杂。

#### (4) 使用内容提供者 (ContentProvider) 存储数据

内容提供者可以使用数据库或者文件作为存储介质，通常使用 SQLite 作为存储数据库。相比其他方式，这种方式支持多个应用程序之间的数据交换。

### 9.5.2 实例实现

【实例 9-5】创建音乐播放列表。

VoicePlayer.java 实现：

```
public class VideoPlayerView extends Activity implements FeedConstants {

    /**
     * TODO: Set the path variable to a streaming video URL or a local media
     * file path.
     */
    private String path = "";
    private VideoView mVideoView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.videoview);
        mVideoView = (VideoView) findViewById(R.id.surface_view);
    }
}
```



```

        Bundle extras = getIntent().getExtras();

        path = extras.getString(KEY_URL);

        /*
         * Alternatively, for streaming media you can use
         *
         */
        // mVideoView.setVideoPath(path);
        mVideoView.setVideoURI(Uri.parse(path));
        MediaController mc = new MediaController(this);

        mVideoView.setMediaController(mc);
        mVideoView.requestFocus();

        mVideoView.start();

    }
}

```

PlayerList.java 实现:

```

public class FeedsList extends ListActivity implements FeedConstants {

    private FeedsDB feedDB;
    private List<Feed> feeds;

    @Override
    protected void onCreate(Bundle icle) {
        try {
            super.onCreate(icle);

            setContentView(R.layout.feeds_list);
            feedDB = FeedsDB.getInstance(this);

            fillData();

        } catch (Throwable e) {
            Log.e("NewsDroid", e.toString());
        }
    }
}

```



```

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);

        MenuItem mItem = menu.add(0, ACTIVITY_INSERT, Menu.NONE, R.string.menu_insert);
        //mItem.setIcon(R.drawable.thumb_nathan);

        mItem = menu.add(0, ACTIVITY_DELETE, Menu.NONE, R.string.menu_delete);

        return true;
    }

    /* (non-Javadoc)
     * @see android.app.Activity#onMenuItemSelected(int, android.view.MenuItem)
     */
    @Override
    public boolean onOptionsItemSelected(int featureId, MenuItem item) {

        super.onOptionsItemSelected(featureId, item);

        switch(item.getItemId()) {
            case ACTIVITY_INSERT:
                createFeed();
                break;
            case ACTIVITY_DELETE:
                feedDB.deleteFeed(feeds.get((int)getSelectedItemId()).feedId);
                fillData();
        }

        return true;
    }

    private void createFeed() {
        Intent i = new Intent(this, URLEditor.class);
    }

```



```

startActivityForResult(i, ACTIVITY_CREATE);

}

/* (non-Javadoc)
 * @see android.app.Activity#onActivityResult(int, int, android.content.Intent)
 */
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    super.onActivityResult(requestCode, resultCode, data);
    fillData();
}

@Override
protected void onListItemClick(ListView l, View v, int position, long id) {

    super.onListItemClick(l, v, position, id);

    Intent i = new Intent(this, ArticlesList.class);

    i.putExtra(FeedConstants.KEY_FEED_ID, feeds.get(position).feedId);
    i.putExtra(FeedConstants.KEY_TITLE, feeds.get(position).title);
    i.putExtra(FeedConstants.KEY_URL, feeds.get(position).url.toExternalForm());

    startActivityForResult(i, ACTIVITY_CREATE);
}

private void fillData() {

    List<String> items = new ArrayList<String>();

    feeds = feedDB.getFeeds();

    for (Feed feed : feeds) {
        items.add(feed.title);
    }
}

```



```

        ArrayAdapter<String> notes =
            new ArrayAdapter<String>(this, R.layout.feeds_row, items);
        setListAdapter(notes);
    }
}

```

FeedsDB.java 实现，用于操作数据库：

```

public class FeedsDB {

    private static final String CREATE_TABLE_FEEDS = "create table feeds (feed_id integer primary key
autoincrement, "
        + "title text not null, url text not null, imageurl text not null);";

    private static final String CREATE_TABLE_ARTICLES = "create table articles (article_id integer primary key
autoincrement, "
        + "feed_id int not null, title text not null, url text not null, imageurl text not null);";

    private static final String FEEDS_TABLE = "feeds";
    private static final String ARTICLES_TABLE = "articles";
    private static final String DATABASE_NAME = "feedsdb2";
    private static final int DATABASE_VERSION = 2;

    /**
     * This class helps open, create, and upgrade the database file.
     */
    private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
            try
            {
                db.execSQL(CREATE_TABLE_FEEDS);
                db.execSQL(CREATE_TABLE_ARTICLES);
            }
            catch (Exception e)

```



```

        {
            Log.i(DATABASE_NAME, "tables already exist");
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(DATABASE_NAME, "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS feeds");
        db.execSQL("DROP TABLE IF EXISTS articles");

        onCreate(db);
    }
}

private DatabaseHelper mOpenHelper;

private FeedsDB (Context ctx)
{
    mOpenHelper = new DatabaseHelper(ctx);
}

public boolean insertFeed(String title, URL url, URL imageUrl) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    ContentValues contentValues= new ContentValues();
    values.put("title", title);
    values.put("url", url.toString());

    if (imageUrl == null)
        values.put("imageurl", "");
    else
        values.put("imageurl", imageUrl.toString());

    boolean resp = (db.insert(FEEDS_TABLE, null, values) > 0);

    return resp;
}

```



```

public boolean deleteFeed(Long feedId) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    boolean resp = (db.delete(FEEDS_TABLE, "feed_id=" + feedId.toString(), null) > 0);

    db.close();

    return resp;
}

public boolean insertArticle(Long feedId, String title, URL url, URL imageUrl) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    ContentValues contentValues= new ContentValues();
    values.put("feed_id", feedId);
    values.put("title", title);
    values.put("url", url.toString());

    if (imageUrl == null)
        values.put("imageurl", "");
    else
        values.put("imageurl", imageUrl.toString());

    boolean resp = (db.insert(ARTICLES_TABLE, null, values) > 0);

    db.close();

    return resp;
}

public boolean deleteArticles(Long feedId) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    boolean resp = (db.delete(ARTICLES_TABLE, "feed_id=" + feedId.toString(), null) > 0);

    db.close();

    return resp;
}

```



```

public List<Feed> getFeeds() {
    ArrayList<Feed> feeds = new ArrayList<Feed>();
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();

    try {

        Cursor c = db.query(FEEDS_TABLE, new String[] { "feed_id", "title",
            "url", "imageurl" }, null, null, null, null, null);

        int numRows = c.getCount();
        c.moveToFirst();
        for (int i = 0; i < numRows; ++i) {
            Feed feed = new Feed();
            feed.feedId = c.getLong(0);
            feed.title = c.getString(1);
            feed.url = new URL(c.getString(2));

            if (c.getString(3) != null && c.getString(3).length() > 0)
                feed.imageUrl = new URL(c.getString(3));

            feeds.add(feed);
            c.moveToNext();
        }
    } catch (SQLException e) {
        Log.e("NewsDroid", e.toString());
    } catch (MalformedURLException e) {
        Log.e("NewsDroid", e.toString());
    }
    finally
    {
        db.close();
    }
    return feeds;
}

```

```

public List<Article> getArticles(Long feedId) {
    ArrayList<Article> articles = new ArrayList<Article>();
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    try {

```



```

        Cursor c = db.query(ARTICLES_TABLE, new String[] { "article_id",
            "feed_id", "title", "url", "imageurl" },
            "feed_id=" + feedId.toString(), null, null, null, null);

        int numRows = c.getCount();
        c.moveToFirst();
        for (int i = 0; i < numRows; ++i) {
            Article article = new Article();
            article.articleId = c.getLong(0);
            article.feedId = c.getLong(1);
            article.title = c.getString(2);

            if (c.getString(3) != null && c.getString(3).length() > 0)
                article.url = new URL(c.getString(3));

            if (c.getString(4) != null && c.getString(4).length() > 0)
                article.imageUrl = new URL(c.getString(4));

            articles.add(article);
            c.moveToNext();
        }
    } catch (SQLException e) {
        Log.e("NewsDroid", e.toString());
    } catch (MalformedURLException e) {
        Log.e("NewsDroid", e.toString());
    }
    finally
    {
        db.close();
    }
    return articles;
}

```

【代码说明】本实例使用数据库存储实现了创建音乐播放列表功能。数据库存储操作主要通过 FeedsDB.java 来实现，FeedsDB 使用 SQLiteOpenHelper 存储数据。

SQLiteOpenHelper 是 Android 的数据库辅助类，该类用来创建或打开数据库。使用 SQLiteOpenHelper 时，需要重写三个重要的方法 onCreate、onUpgrade 和 onOpen：

- onCreate(SQLiteDatabase sqLiteDatabase)：该方法在数据库被首次创建时调用。



- onUpgrade(SQLiteDatabase sqliteDatabase, int ov,int nv): 该方法在数据库升级时被调用。
- onOpen((SQLiteDatabase sqliteDatabase): 该方法在数据库打开时被调用。

在 Eclipse 下运行该程序, 运行结果如图 9-11 所示。

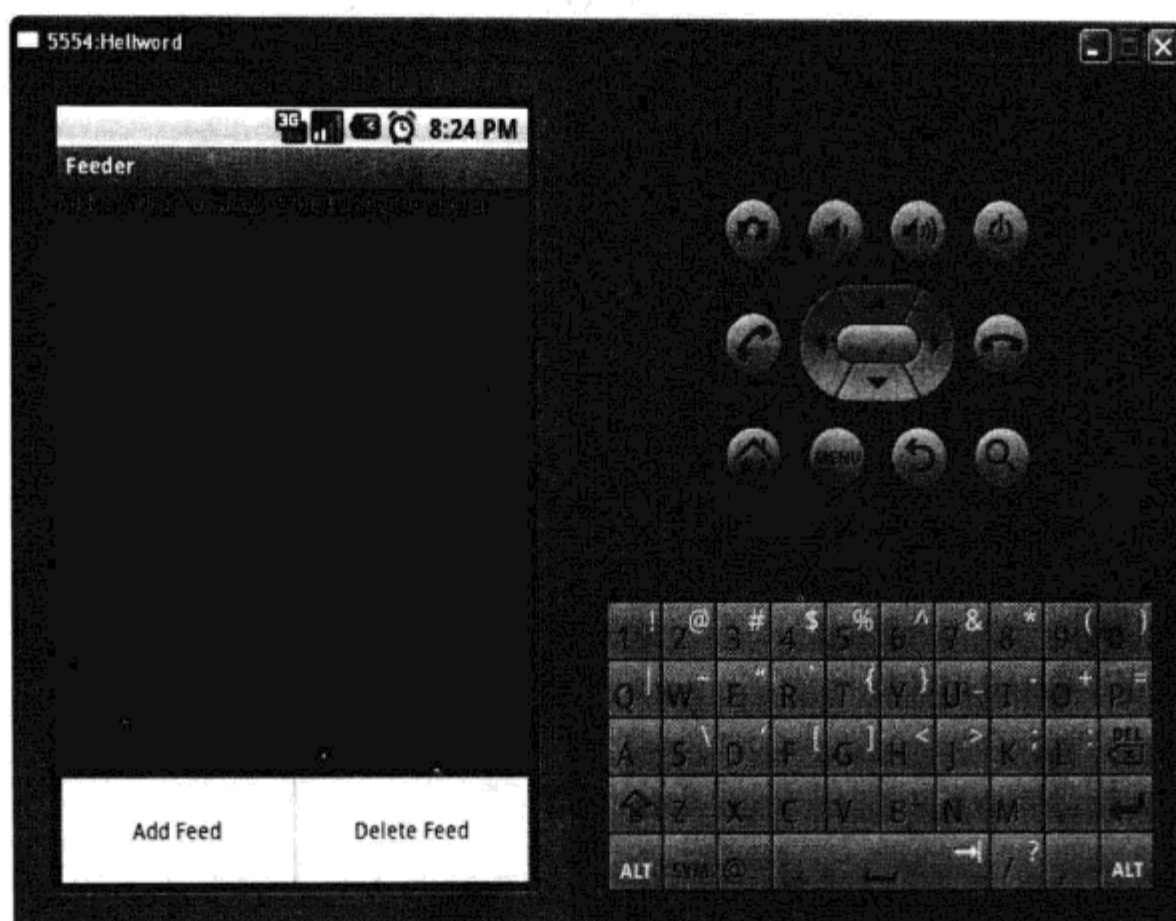


图 9-11 程序运行结果





## 第 10 章 Android 的网络通信

网络通信应该包含三部分的内容：接收方、发送方以及协议栈。接收方和发送方是参与通信的主体，协议栈是接收方和发送方进行通信的规约。按照服务类型，网络通信可分为面向连接和无连接两种方式。面向连接是在通信前建立通信链路，而通信结束后释放该链路。无连接方式则不需要在通信前建立通信链路，这种方式不保证传输的质量。在电话通信中，常用的协议比较多，如 SS7（Signaling System 7）、SIP（Session Initiation Protocol）等。Andorid 提供了多种网络通信的方式：Webkit、HttpComponents 以及 Socket。本章将讲述 Andorid 的网络通信机制，通过本章的学习读者可掌握 Andorid 的通信原理。

### 10.1 访问 Internet

#### 10.1.1 使用 WebKit 组件访问 Internet

现在浏览新闻、发布微博、网络聊天和网络购物等已经成为我们生活中的必不可少的行为，这些行为都需要网络的支持，也就是上网。可以说“上网”是我们无时无刻都在进行的一个应用，是生活中不可缺少的应用。手机上网目前已经成为一种时尚，被越来越多的手机用户接纳。通过手机上网，用户可以在任何时间、任何地点以及任何方式浏览新闻、发布微博、聊天和购物等，如图 10-1 所示。



图 10-1 使用手机上网





Android 系统提供了多种方式访问 Internet:

- 使用 WebKit 访问 Internet。
- 使用 Apache HttpComponents 访问 Internet。

本节将讲述使用 Webkit 访问 Internet 的方法。

WebKit 是一个开源的浏览器引擎，这个引擎被当前常用的手机操作系统所使用，其中包括 Android。Android 向用户开放了 WebKit 引擎的 API，可通过导入 android.webkit 包来使用 WebKit 作为浏览器引擎。android.webkit 包包含了许多用于访问 Internet 的类和接口。

相比其他浏览器引擎，WebKit 浏览器引擎不仅具有较好的渲染效果，而且兼容 Web 标准、可扩展性好，因而 WebKit 引擎项目被多种手机操作系统所采纳。在 Android 平台中，WebKit 引擎可分为 Java 引擎库和 WebCore 引擎库两个部分。这两个库之间使用 JNI 和 Bridge 相互调用。Java 引擎库使用 JavaScript 实现，该引擎负责与 Android 应用程序进行通信；而 WebCore 引擎库负责处理实际的网页生成与版面元素。图 10-2 显示了 WebKit 引擎子模块的调用关系。

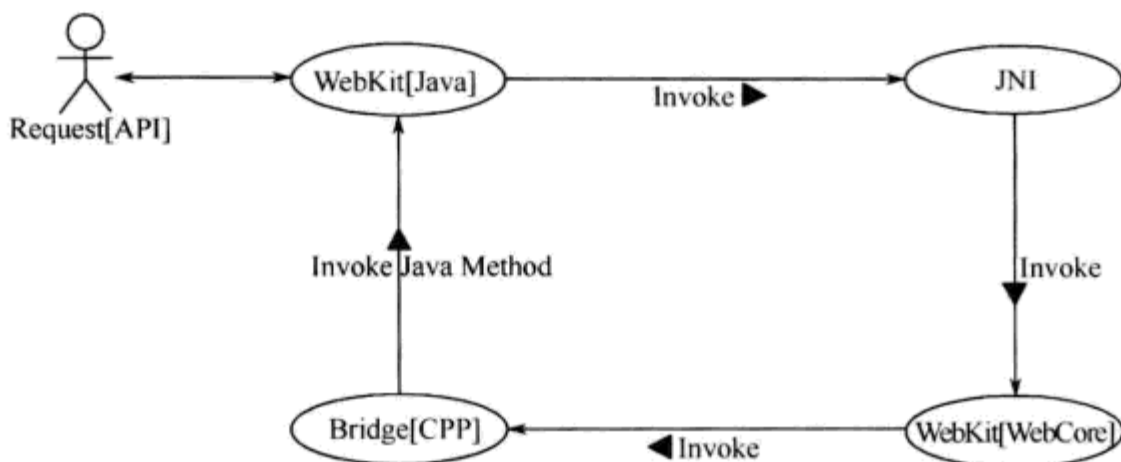


图 10-2 WebKit 引擎子模块的调用关系

访问 Internet 时，Android 的 WebKit 模块通过 Java 引擎库的加载器来加载相应类型的数据。Android 的加载器都是 StreamLoader（流加载器）和 Handler 的子类。StreamLoader 类定义了 3 个方法，其中包括一个公有的方法（load）和两个抽象保护方法（setupStreamAndSendStatus 和 buildHeaders）：

- load 方法是 StreamLoader（流加载器）的公有方法。该方法用于加载指定类型的内容，在载入数据时被调用。当数据载入事件发生时，WebKit 库会通知 BrowserFrame 并且更新加载的进度。该类接收到进度条事件后会通过回调代理通知 View 类。
- setupStreamAndSendStatus 方法是 StreamLoader（流加载器）的保护方法。该方法主要用于构造数据流和发送状态，其中这些数据流是和通信协议相关的，而状态发送给 LoadListener（加载监听器）。载入器的状态是在 StreamLoader 类定义的，该类定义了 4 中载入器的状态：MSG\_STATUS（发送状态消息）、MSG\_DATA（发送数据消息）、MSG\_HEADERS（发送消息头消息）、MSG\_END（数据发送完毕消息）。
- buildHeaders 方法是 StreamLoader（流加载器）的保护方法。该方法负责构造特定协议的消息头。

StreamLoader 包含三种类型的载入器：

- CacheLoader（缓存加载器），加载缓存内容；
- DataLoader（数据加载器），用于载入网页数据。



● **FileLoader**（文件加载器），将文件数据加载到 **Frame** 中。

加载器在构造时，首先启动一个事件处理线程，相应的加载工作由该线程负责。通过 **Webkit** 包提供的类，可实现 **Internet** 的访问。**Webkit** 包中主要类的功能如表 10-1 所示。

表 10-1 Webkit 引擎主要类的功能

类名	功能
BrowserFrame	BrowserFrame 类是一个封装器，封装了 WebCore 库中的 Frame 类。使用该类可创建 WebCore 中定义的 Frame，并且创建该对象的事件监听方法
WebView	WebView 类是 WebKit 模块 Java 层的视图类，所有需要使用 Web 浏览功能的 Android 应用程序都要创建该视图对象显示和处理请求的网络资源。目前，WebKit 模块支持 HTTP、HTTPS、FTP 以及 JavaScript 请求。WebView 作为应用程序的 UI 接口，为用户提供了一系列的网页浏览、用户交互接口，客户程序通过这些接口访问 WebKit 核心代码
HttpAuthHandler	Http 认证处理类。该类对象作为参数传递给 BrowserCallback.displayHttpAuthDialog 方法
DownloadManagerCore	DownloadManagerCore 类负责管理下载网络资源，所有的网络资源的下载均由该类管理。该类实例运行在 WebKit 线程当中，通过调用 CallbackProxy 对象与 UI 线程交互
CacheLoader	缓存加载器。用于加载缓存内容
DataLoader	数据加载器。用于用于载入网页数据
FileLoader	文件加载器。用于将文件数据加载到 Frame 中
WebViewDatabase	Web 视图数据库类。该类封装了 WebKit 引擎对 SQLiteDatabase 的操作。WebViewDatabase 是一个 single 模式的类，需要通过 getInstance 方法获取实例对象。通过该类可实现 Web 相关的数据库操作，如存储浏览器的缓冲数据、历史浏览数据等
CacheSyncManager	Cache 同步管理器。该类负责同步浏览器缓存数据
Network	该类封装了网络连接逻辑，为开发人员提供更为高级的网络连接
CallbackProxy	回调代理类，该类用于 UI 和 WebCore 之间的交互。与用户相关的通知方法是在该类中定义的。当 WebCore 完成相应的数据处理时，会调用 CallbackProxy 类中对应的方法，这些方法通过消息方式间接调用相应处理对象的处理方法
SslErrorHandler	Ssl 错误处理器。该类提供了处理 SSL 错误的方法
JsResult	Js 结果类。该类用于用户交互
WebChromeClient	WebChrome 客户端。该类定义了一系列的事件，这些事件与浏览窗口修饰相关，例如接收到 Title、进度变化等
CellList	CellList 类定义了图片集合中的 Cell，该类用于管理 Cell 图片的绘制、状态变化以及索引
LoadListener	加载监听器。当有下载事件时，该类的 DownloadFileMethod 会被调用
WebViewClient	Web 视图客户端类。该类定义了页面载入、资源载入、页面访问错误等情况发生时的处理方法
DragClient	拖曳客户端。该类定义了与页面拖曳相关的处理
StreamLoader	包含三种类型的载入器： CacheLoader（缓存加载器），加载缓存内容 DataLoader（数据加载器），用于载入网页数据 FileLoader（文件加载器），将文件数据加载到 Frame 中

下面以一个实例说明 Webkit 引擎的功能，通过该实例，读者可掌握 Webkit 的基本用法。

【实例 10-1】Webkit 的应用。

主程序 WebkitApplication.java 实现，该程序使用 main.xml 初始化界面。

```

public class WebKitApplication extends Activity {
    /** Called when the activity is first created. */
    WebView webwidget;           //声明 WebView 对象，使用该对象浏览网页
    String regex = "http.*";     //声明匹配 Uri 的正则表达式
    UriRegex uriRegex;           //声明 UriRegex 类
    private TextView textview;    //声明 TextView 对象
    private Button loaduri;       //声明 Button 对象

```



```
private EditText uri_edittext; //声明 EditText 对象
private String uri;           //声明 uri 对象
public void onCreate(Bundle bundle) {
    super.onCreate(bundle); //必须执行父类的 onCreate 方法
    setContentView(R.layout.main); //使用 main.xml 初始化程序布局
    try
    {
        textview = (TextView) findViewById(R.id.textview);
        //使用 R.id.textview 资源指定的参数创建 TextView 对象
        loaduri = (Button) findViewById(R.id.loaduri);
        //使用 R.id.loaduri 资源指定的参数创建 Button 对象
        uri_edittext = (EditText) findViewById(R.id.uri);
        //使用 R.id.uri 资源指定的参数创建 EditText 对象
        webwidget = new WebView(this); //创建 webwidget 对象
        loaduri.setOnClickListener(new MyButtonClickListener());
        //注册按钮 loaduri 的单击事件监听器
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(WebKitApplication.this, e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}

private class MyButtonClickListener implements OnClickListener{
    public void onClick(View view) {
        try
        {
            uriRegex = new UriRegex(regex);
            //使用构造函数 UriRegex(regex)创建 UriRegex 对象 uriRegex
            uri = uri_edittext.getText().toString(); //获取输入的 Uri
            boolean isValidUri = ValidUri(uri, uriRegex.getUriRegex());
            //判断输入的 uri 的格式是否有效
            if (isValidUri)
```



```

        {
            textView.setText("Loading " + uri); //设置 textView 显示
            webwidget.loadUrl(uri); //使用 WebView 的 loadUrl 方法加载指定的 Uri
        }
        else
            Toast.makeText(WebKitApplication.this, "无效 Uri 格式 "+uri, Toast.LENGTH_LONG).
                .show();
        /*判断输入的 Uri 是否有效, 若有效则加载该 Uri; 否则弹出提示无效 Uri 格式的 Toast 消息*/
        //webView.setDownloadListener(new WebDownloadListener());
    }
    catch (Exception e)
    {
        Toast.makeText(WebKitApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG)
            .show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}

/*实现单击事件监听器。当该监听器监听到单击事件时, onClick 方法被调用*/

public static boolean ValidUri(String uri, String uriregex)
{
    Pattern pattern = Pattern.compile(uriregex);
    /*根据正则表达式 uriregex 生成模式对象*/
    Matcher matcher = pattern.matcher(uri);
    /*模式对象利用正则表达式 uriregex 匹配 uri, 返回 Matcher 对象*/
    if(matcher.matches())
    {
        return true;
    }
    return false;
    /*通过 Matcher 对象的 matches 方法判断匹配是否成功
    * 即 uri 是否含有正则表达式 regexp 定义的字符串
    * 若包含 regexp 定义的字符串, 则返回真值 (true)
    * 否则返回假值 (false)
    */
}

```



```

    **/
}
/*检查输入的网址是否有效：若包含返回 true；否则返回 false*/

private class UriRegex
{
    String regex; //定义对象的属性
    UriRegex(String regex)
    {
        this.regex = regex;
    }
    /*构造函数 UriRegex(String regex)，使用该构造函数设置 regex 属性*/

    public String getUrlRegex()
    {
        String regex = this.regex;
        return regex;
    }
    /*getUrlRegex 返回本对象的 regex*/

    public void setUrlRegex(String regex)
    {
        this.regex = regex;
    }
    /*setUrlRegex 设置该对象的 regex 属性*/
}
/*自定义的类 UriRegex，该类提供了对正则表达式的操作*/
}

```

main.xml 采用 AbsoluteLayout 布局，子元素垂直分布，高度和宽度同父元素相同。该布局包含 1 个 TextView 控件、1 个 EditText 控件和 1 个 Button 控件。

【代码说明】 本实例实现了使用 WebKit 引擎的 WebView 访问 Internet 的功能。WebView 类提供了浏览指定的网页的方法。若需要在程序中加入浏览器的支持，可将该类的实例或者派生类的实例作为视图，然后调用 Activity 类的 setContentView 方法显示出来。

使用 WebView 可访问 Internet，WebView 对象创建主要涉及 3 个类：WebViewCore（Web 视图核心）、CallbackProxy（回调代理）以及 WebViewDatabase（视图数据库）：

- WebViewCore（Web 视图核心）是 WebKit 的核心层，负责初始化 WebKit 模块类库；
- CallbackProxy 为 WebKit 模块中 UI 线程和 WebKit 类库提供交互功能；
- WebViewDatabase 为 WebKit 模块运行时缓存、数据存储提供支持。

WebKit 会在第一次生成 WebView 对象时进行初始化。WebKit 模块初始化流程如图 10-3 所示。



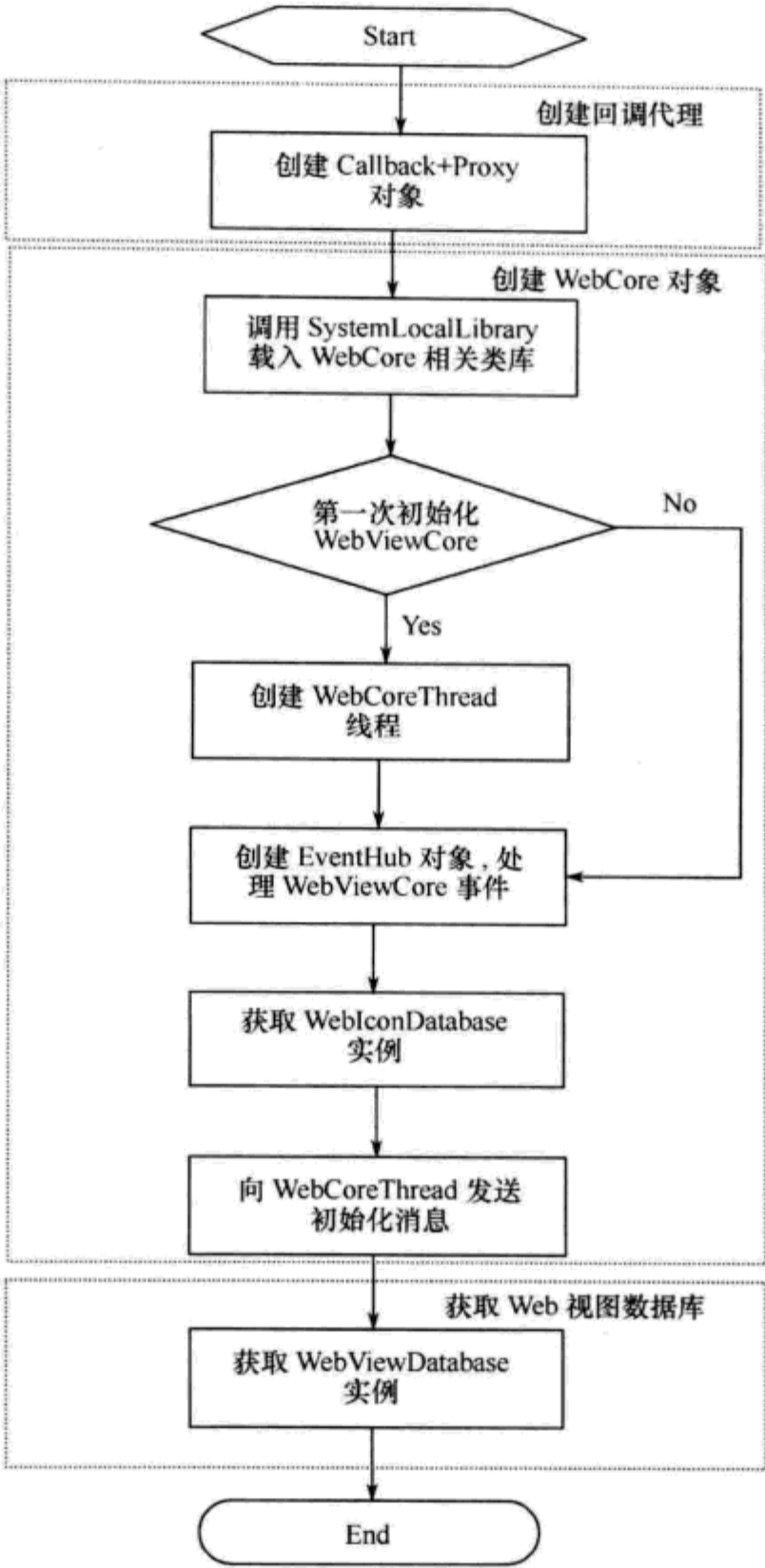


图 10-3 WebKit 模块初始化流程

注意，由于该实例需要访问 Internet，因而需要在清单文件中添加访问 Internet 的许可。否则运行该程序时，会出现没有相应许可的异常错误。在该工程下的 AndroidMainfest 添加用户权限的标签，其 android:name 属性为 android.permission.INTERNET：

```

<uses-permission android:name="android.permission.INTERNET"/>
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".WebKitApplication"
        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>
```

本实例使用 `WebView` 类的 `loadUrl` 方法请求访问指定的 URL 网页数据。`WebView` 对象中保存着 `WebViewCore` 的引用, 由于 `WebView` 属于 UI 线程, 而 `WebViewCore` 属于后台线程, 因此 `WebView` 对象的 `loadUrl` 被调用时, 会通过消息的方式将 URL 信息传递给 `WebViewCore` 对象。该对象会调用成员变量 `mBrowserFrame` 的 `loadUrl` 方法, 进而调用 `WebKit` 库完成数据的载入。网络数据的载入分别由 Java 层和 C 层共同完成, Java 层完成用户交互、资源下载等操作, 而 C 层主要完成数据分析(建立 DOM 树、分析页面元素等)操作。由于 UI 线程和 `WebCore` 线程运行在不同的两个层中, 因此当用户请求访问网络资源时, 通过消息的方式向 `WebViewCore` 对象发送载入资源请求。在 Java 层的 `WebKit` 模块中, 所有与资源载入相关的操作都是由 `BrowserFrame` 类中对应的方法完成的, 这些方法是本地方法, 会直接调用 `WebCore` 库的 C 层函数完成数据载入请求, 以及资源分析等操作。C 层的 `FrameLoader` 类是浏览框架的资源载入器, 该类负责检查访问策略以及向 Java 层发送下载资源请求等。

本实例使用 `main.xml` 初始化程序布局, 程序采用 `AbsoluteLayout` 布局, 子元素垂直分布, 高度和宽度同父元素相同。该布局包含 1 个 `TextView`、1 个 `EditText` 控件和 1 个 `Button` 控件。其中, `TextView` 控件作为上网提示按钮, 高度和宽度随内容变化。 `EditText` 控件作为网址输入框, 控件在指定位置 (0px, 30px) 上显示, 高度和宽度随内容变化。 `Button` 控件作为加载网页按钮, 控件在指定位置 (100px, 100px) 上显示, 高度和宽度随内容变化。 `Button` 控件通过 `setOnClickListener(new MyButtonClickListener())` 方法注册按钮的单击事件监听器。当该监听器监听到单击事件时, `MyButtonClickListener` 的 `onClick` 方法被调用。 `onClick` 方法首先使用构造函数 `UriRegex(regex)` 创建 `UriRegex` 对象 `uriRegex` 并获取输入框的 `Uri`, 然后调用 `ValidUri(String uri, String uriRegex)` 方法来判断判断输入的 `uri` 的格式是否符合正则表达式 `uriRegex`。 `ValidUri` 方法通过 `Matcher` 对象的 `matches` 方法判断匹配是否成功, 即 `uri` 是否含有正则表达式 `regex` 定义的字符串。若包含 `regex` 定义的字符串, 则返回真值 (`true`); 否则返回假值 (`false`)。若编辑框的 `Uri` 与正则表达式 `uriRegex` 相匹配, 则使用 `WebView` 的 `loadUrl` 方法加载指定的 `Uri`, 否则弹出无效 `Uri` 格式的 `Toast` 消息。在 `Eclipse` 下运行该 `Android` 工程, 图 10-4 显示了程序的启动界面。

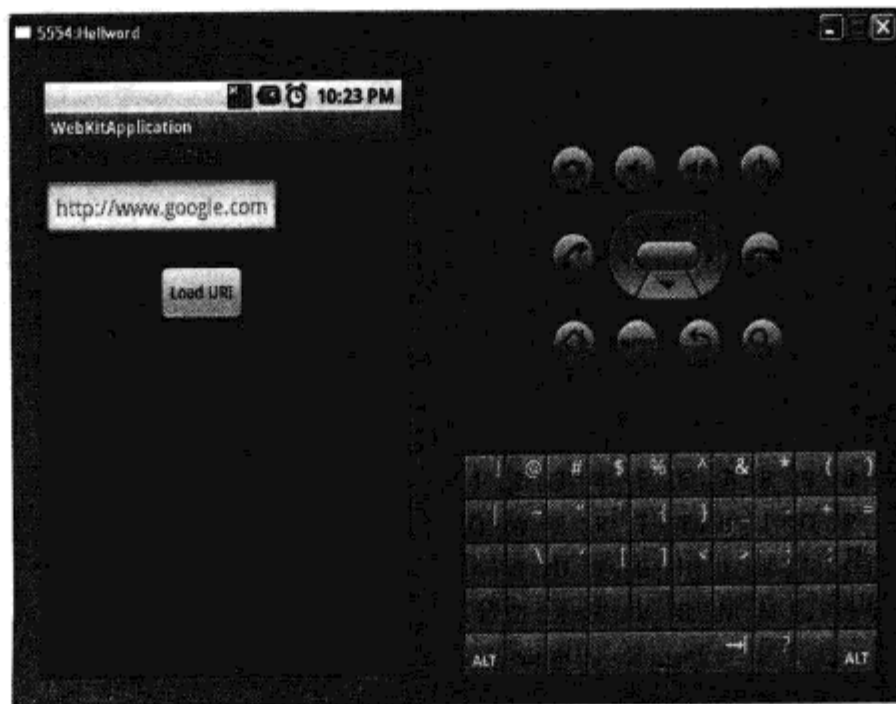


图 10-4 程序启动界面



程序启动后，在编辑框中输入网址，如 `http://www.google.hk`，如图 10-5 所示。

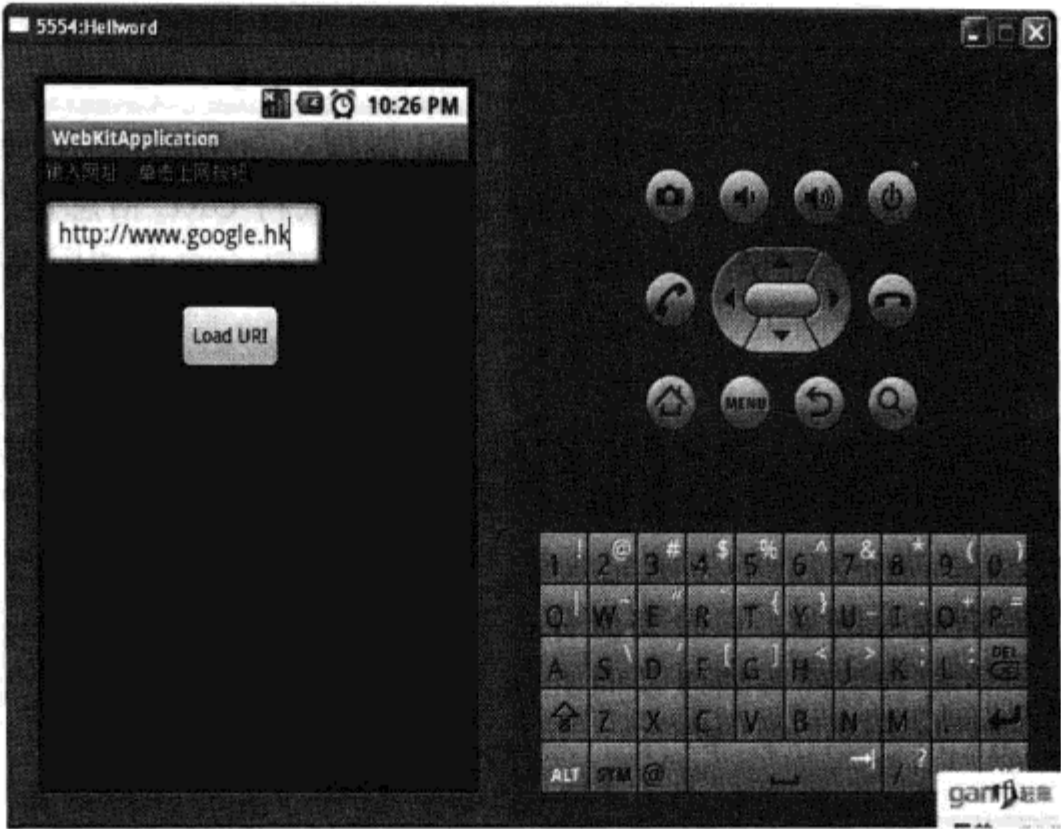


图 10-5 在编辑框中输入有效的 Uri

然后单击“Load URI”按钮，该按钮对应的事件监听器 `MyButtonClickListener` 的 `onClick` 方法被调用。`onClick` 方法根据编辑框的 Uri 来判断是否使用 `WebView` 的 `loadUrl` 方法加载。若编辑框的 Uri 与正则表达式 `uriregex` 相匹配，则使用 `WebView` 的 `loadUrl` 方法加载指定的 Uri，否则弹出无效 Uri 格式的 Toast 消息。本实例使用“`http*`”作为匹配 Uri 的正则表达式，该表达式会匹配所有以 `http` 为开头的网址。由于输入的 Uri 为 `http://www.google.hk`，因而这是一个有效的 Uri。这个 Uri 会被 `WebView` 使用 `loadUrl` 方法加载。图 10-6 显示了使用 `WebView` 加载 `http://www.google.hk` 后的界面。

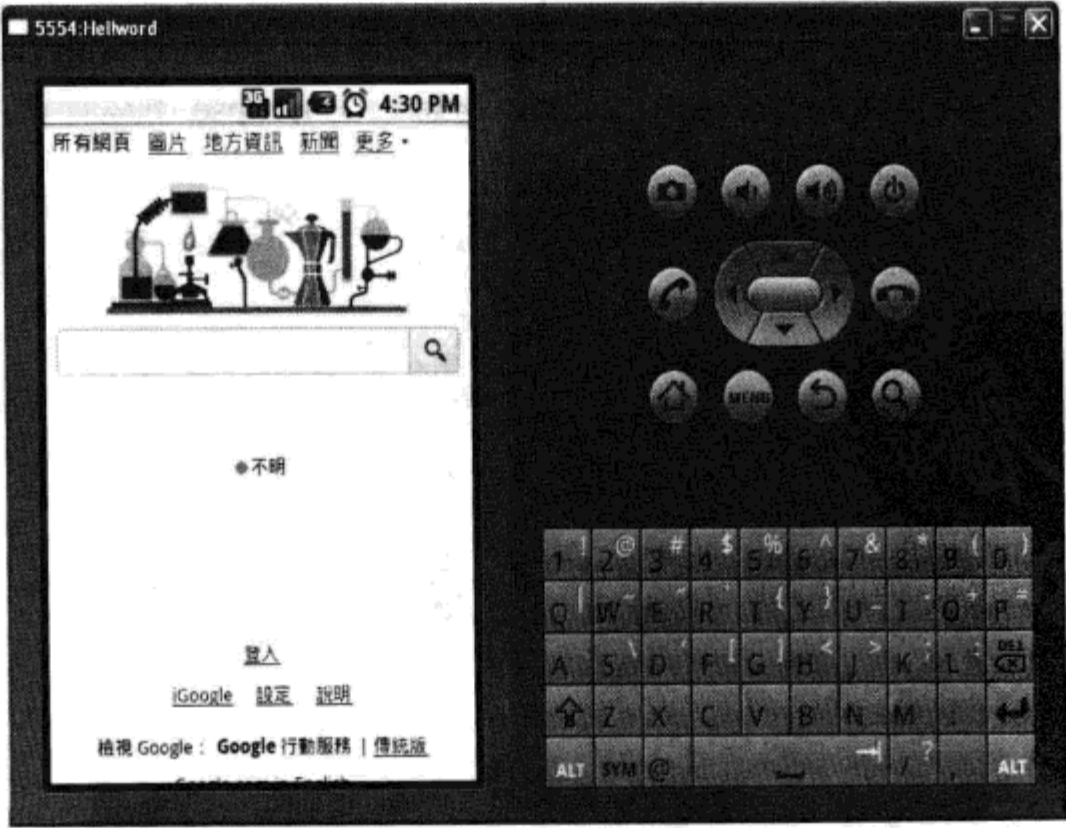


图 10-6 使用 WebView 加载 Uri

若编辑框的 Uri 与正则表达式 `uriregex` 不匹配，则弹出无效 Uri 格式的 Toast 消息。在编辑



框中输入 `hddp://www.google.hk`，如图 10-7 所示。



图 10-7 在编辑框中输入无效的 Uri

然后单击“Load URI”按钮，由于输入的 Uri (`hddp://www.google.hk`) 是一个无效的 Uri，因而程序会弹出无效 Uri 格式的 Toast 消息，如图 10-8 所示。



图 10-8 弹出无效 Uri 格式的 Toast 消息

10.1.2 使用 Apache HttpComponents 访问 Internet

Apache HttpComponents 项目负责创建和维护低水平的 Java 组件，这个组件主要关注 HTTP 和相关协议。这个项目属于软件基金会 (<http://www.apache.org>)，是开发者和使用者社区群体的一部分。HttpComponents 项目的前身是 HttpClient 项目，它已经从一个单纯的 HTTP 客户端



组件转变成了兼容客户端服务器的组件。HttpComponents 项目包含 HTTP 协议的几个方面的底层库。在服务器或客户端通信方面有高级需求的用户会发现，它对于建构定制的 HTTP 协议服务是一个必不可少的工具集。HttpCore 是一套低水平 HTTP 传输元件，它以最小空间建立客户端和服务器的 HTTP 服务。

HttpComponents 包括以下两个组件：

#### (1) HttpComponents Core

HttpCore 是一组底层 HTTP 传输组件，通过该组件使用简单的代码就能实现客户端和服务器的 HTTP 服务。HttpCore 支持两种 I/O 模型：基于 Java I/O 的阻塞模型和基于 Java NIO 的非阻塞模型。阻塞模型更适合数据敏感、低延迟的情况。而非阻塞模型更适合高延迟的情况。

#### (2) HttpComponents Client

HttpClient 本身不是一个浏览器，而是一个客户端的 HTTP 传输库，其目的是为了 Let HttpClient 接收和发送 HTTP 消息。HttpClient 提供了可重用的组件，例如客户端认证、HTTP 状态管理、HTTP 连接管理。HttpClient 继承和使用了 HttpClient 3.x。HttpClient 不会尝试缓存内容，不执行嵌入在 HTML 网页中的 JavaScript，不试图查看内容类型，不格式化 URI 请求或重定向位置。HttpClient 最重要的作用是执行 HTTP 方法。执行一个 HTTP 方法涉及一个或几个 HTTP 请求/HTTP 响应的交互，通常在内部由 HttpClient 处理。请求对象是由用户来提供的，而 HttpClient 负责将该请求转送到目标服务器并返回一个相应的响应对象，或执行失败时抛出一个异常。

下面是一个请求执行过程的简单例子：

```
HttpClient httpclient = new DefaultHttpClient();
String uri = "http://test/";
int count;
byte[] bytearray = new byte[2048];
HttpGet httpget = new HttpGet(uri);
HttpResponse httpResponse = httpclient.execute(httpget);
HttpEntity httpentity = httpResponse.getEntity();
if (httpentity != null) {
    InputStream inputStream = httpentity.getContent();
    count = inputStream.read(bytearray);
    while (count != -1) {
    }
}
```

一个 HTTP 请求包含一个请求行，该请求行包含请求方法名、请求 URI 以及 HTTP 协议版本。HttpClient 支持 HTTP/1.1 定义的方法：GET、HEAD、POST、PUT、DELETE、TRACE 和 OPTIONS。每一种方法对应一个特定的类：HttpGet、HttpHead、HttpPost、HttpPut、HttpDelete、HttpTrace 和 HttpOptions。请求 URI 是一个统一的资源标识符，这个资源标识符标识申请请求的资源位置。HTTP 请求 URI 由协议的方案、主机名称、可选的端口、资源的路径、可选的查询和可选的片段组成。HttpClient 提供了大量的实用方法来建立和修改请求 URI，使用这些接口可操作 URI：

```
URI uri = URIUtils.createURI("http", "www.google.com", -1, "/search",
```



```
"q=httpclient&btnG=Google+Search&aq=f&oq=", null);
HttpGet httpget = new HttpGet(uri);
```

HTTP 响应是在服务器收到请求后, 服务器端返回给客户端的消息。响应消息的第一行包含协议版本、状态码和它的相关文本的短语, 如 HTTP/1.1 200 OK。

HttpClient 接口代表了最基本的 HTTP 请求执行规约。它没有在请求执行的过程上强加任何限制或特定的具体细节, 不关心连接管理细节、状态管理细节、认证和重定向处理的实现。这使得使用额外的功能(内容缓存)实现接口变得容易。HttpClient 通常使用 DefaultHttpClient 生成 HttpClient 对象, DefaultHttpClient 是默认的 HttpClient 类。这个类用来处理 HTTP 协议的某一方面功能, 如重定向或认证处理、关于保持连接和保活时间的决策。

DefaultHttpClient 同时负责维护一个用于处理发送请求和收到的响应的协议翻译器链表, 并提供方法来管理这些翻译器。新协议翻译器可自由地加入协议处理器链或从该链表中离开。协议翻译器被存储在一个简单的 java.util.ArrayList 中。DefaultHttpClient 是线程安全的, 建议多个请求重用这个类的实例:

```
DefaultHttpClient httpclient = new DefaultHttpClient();
httpclient.removeRequestInterceptorByClass(RequestUserAgent.class);
httpclient.addRequestInterceptor(new HttpRequestInterceptor() {

    public void process(
        HttpRequest request, HttpContext context)
        throws HttpException, IOException {
        request.setHeader(HTTP.USER_AGENT, "My-own-client");
    }
});
```

实例 DefaultHttpClient 不再需要时, 必须通过调用 ClientConnectionManager 的 shutdown() 方法关闭。下面这些参数用于自定义默认 HttpClient 行为:

- ClientPNames.HANDLE\_REDIRECTS = 'http.protocol.handle-redirects': 定义是否自动重定向。如果这个参数没有被设置, HttpClient 将自动重定向。
- ClientPNames.REJECT\_RELATIVE\_REDIRECT = 'http.protocol.reject-relative-redirect': 定义是否拒绝相关的重定向。HTTP 标准要求这个位置是一个绝对的 URI。如果这个参数没有被设置, HttpClient 将允许相关的重定向。
- ClientPNames.MAX\_REDIRECTS = 'http.protocol.max-redirects': 定义最多的重定向数。限制重定向数是为了防止因服务器端脚本而导致的死循环。如果这个参数没有被设置, HttpClient 最多允许 100 个重定向。
- ClientPNames.ALLOW\_CIRCULAR\_REDIRECTS = 'http.protocol.allow-circular-redirects': 定义是否允许循环重定向。HTTP 标准并没有清楚定义是否允许循环重定向。如果这个参数没有被设置, 则禁止循环重定向。
- ClientPNames.CONNECTION\_MANAGER\_FACTORY\_CLASS\_NAME = 'http.connection-manager.factory-class-name': 定义的类型名的默认 ClientConnectionManager 实现。如果这个参数没有被设置, 则默认使用 SingleClientConnManager。
- ClientPNames.VIRTUAL\_HOST = 'http.virtual-host': 定义虚拟主机名称用于主机头而不



是物理主机上的名字。如果这个参数没有被设置，则使用目标主机的 IP 地址。

- ClientPNames.DEFAULT\_HEADERS = 'http.default-headers': 定义发送请求表头。
- ClientPNames.DEFAULT\_HOST = 'http.default-host': 定义默认主机。如果目标主机没有在请求 URI 指定，则使用默认值。

下面以一个实例说明 Http Components 的功能，通过该实例，读者可掌握 Http Components 的基本用法。

【实例 10-2】HttpComponents 的应用。

主程序 HttpApplication.java 实现，该程序使用 main.xml 初始化界面：

```
public class HttpApplication extends Activity {
    /** Called when the activity is first created. */
    HttpClient client; //声明 HttpClient 对象，使用该对象浏览网页
    String regex = "http.*"; //声明匹配 Uri 的正则表达式
    UriRegex uriRegex; //声明 UriRegex 类
    private TextView textview; //声明 TextView 对象
    private Button loaduri; //声明 Button 对象
    private EditText uri_edittext; //声明 EditText 对象
    private String uri; //声明 uri 对象
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle); //必须执行父类的 onCreate 方法
        setContentView(R.layout.main); //使用 main.xml 初始化程序布局
        try
        {
            textview = (TextView) findViewById(R.id.textview);
            //使用 R.id.textview 资源指定的参数创建 TextView 对象
            loaduri = (Button) findViewById(R.id.loaduri);
            //使用 R.id.loaduri 资源指定的参数创建 Button 对象
            uri_edittext = (EditText) findViewById(R.id.uri);
            //使用 R.id.uri 资源指定的参数创建 EditText 对象
            loaduri.setOnClickListener(new MyButtonClickListener());
            //注册按钮的单击事件监听器
        }
        /*try 块包含可能出现异常的代码*/
        catch (Exception e)
        {
            Toast.makeText(HttpApplication.this, e.toString(), Toast.LENGTH_LONG).show();
        }
        /*catch 捕捉异常，若出现异常，则显示异常的 Toast 消息*/
        finally
        {
            //TODO
        }
    }
}
```



```

    }
    /*finally 中可添加处理异常的代码*/
}

private class MyButtonClickListener implements OnClickListener{
    public void onClick(View view) {
        try
        {
            uriRegex = new UriRegex(regex);
            //使用构造函数 UriRegex(regex)创建 UriRegex 对象 uriRegex
            uri = uri_edittext.getText().toString(); //获取输入的 uri
            boolean isValidUri = ValidUri(uri,uriRegex.getUriRegex());
            //判断输入的 uri 的格式是否有效
            if (isValidUri)
            {
                textView.setText("Loading " + uri); //设置 textView 显示
                client = new DefaultHttpClient();
                HttpGet get = new HttpGet(uri); //根据 uri 创建 HttpGet 对象
                HttpResponse response = client.execute(get);
                BufferedReader bufferedreader = new BufferedReader(new InputStreamReader(
                    response.getEntity().getContent()));
                for (String string = bufferedreader.readLine(); string != null; string = bufferedreader.readLine())
                {
                    Log.v("response", string);
                }
            }
            else
            {
                Toast.makeText(HttpApplication.this, "无效 Uri 格式 "+uri,Toast.LENGTH_LONG).
                    show();
                /*判断输入的 Uri 是否有效, 若有效加载该 Uri; 否则提示无效 Uri 格式的 Toast
                消息*/
                //webView.setDownloadListener(new WebDownloadListener());
            }
        }
        catch (Exception e)
        {
            Toast.makeText(HttpApplication.this, "异常错误: " + e.toString(),Toast.LENGTH_LONG).
                show();
        }
        /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    }
    finally

```



```

        {
            //TODO
        }
        /*finally 中可添加处理异常的代码*/
    }
}

/*实现单击事件监听器，当该监听器监听到单击事件时，onClick 方法被调用*/

public static boolean ValidUri(String uri, String uriregex)
{
    Pattern pattern = Pattern.compile(uriregex);
    /*根据正则表达式 uriregex 生成模式对象*/
    Matcher matcher = pattern.matcher(uri);
    /*模式对象利用正则表达式 uriregex 匹配 uri，返回 Matcher 对象*/
    if(matcher.matches())
    {
        return true;
    }
    return false;
    /* 通过 Matcher 对象的 matches 方法判断匹配是否成功
     * 即 uri 是否含有正则表达式 regexp 定义的字符串
     * 若包含 regexp 定义的字符串，则返回真值 (true)
     * 否则返回假值 (false)
     */
}

/*检查输入的网址是否有效：若包含返回 true；否则返回 false*/

private class UriRegex
{
    String regex; //定义对象的属性
    UriRegex(String regex)
    {
        this.regex = regex;
    }
    /*构造函数 UriRegex(String regex)，使用该构造函数设置 regex 属性*/

    public String getUrlRegex()
    {
        String regex = this.regex;
        return regex;
    }
}

```



```

    }
    /*getUrlRegex 返回本对象的 regex*/

    public void setUrlRegex(String regex)
    {
        this.regex = regex ;
    }
    /*setUrlRegex 设置该对象的 regex 属性*/
}
/*自定义的类 UriRegex, 该类提供了对正则表达式的操作*/
}

```

布局 main.xml 实现, 采用 AbsoluteLayout 布局, 子元素垂直分布, 高度和宽度同父元素相同。包含 1 个 TextView、1 个 EditText 控件和 1 个 Button 控件。

```

<?xml version="1.0" encoding="utf-8"?>
<!-- 采用 AbsoluteLayout 布局, 子元素垂直分布, 高度和宽度同父元素相同 -->
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    _
    >
<!-- EditText 作为上网提示按钮, 高度和宽度随内容变化-->
<TextView
    android:id="@+id/textview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="输入网址, 单击上网按钮"
    />
<!-- EditText 作为网址输入框, 控件在指定位置 (0px, 30px) 上显示, 高度和宽度随内容变化-->
<EditText
    android:id="@+id/uri"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="0px"
    android:layout_y="30px"
    android:hint="http://www.google.com"
    />
<!-- Button 作为加载网页按钮, 控件在指定位置 (100px, 100px) 上显示, 高度和宽度随内容变化 -->
<Button
    android:id="@+id/loaduri"
    android:layout_gravity="center"

```



```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="100px"
        android:layout_y="100px"
        android:text="Load URI"
    />
</AbsoluteLayout>

```

**【代码说明】** 本实例使用 Apache HttpComponents 实现了访问 Internet 的功能。HttpClient 接口代表了最基本的 HTTP 请求执行规约，它没有在请求执行的过程上强加任何限制或特定的细节，不关心连接管理细节、状态管理细节、认证和重定向处理的实现。这使得使用额外的功能（内容缓存）实现接口变得容易。HttpClient 通常使用 DefaultHttpClient 生成 HttpClient 对象。DefaultHttpClient 是默认的 HttpClient 类。这个类用来负责处理 HTTP 协议的某一方面功能，如重定向或认证处理、关于保持连接和保持活动时间的决策。

本例使用 Log 工具将 Response 打印到 Android 的日志中。Log 工具是在 android.util.Log 中定义的，其提供了输出日志的方法和属性。可通过 Eclipse 查看 Log 工具的输出，选择“Window → Show View → Other”命令，如图 10-9 所示。

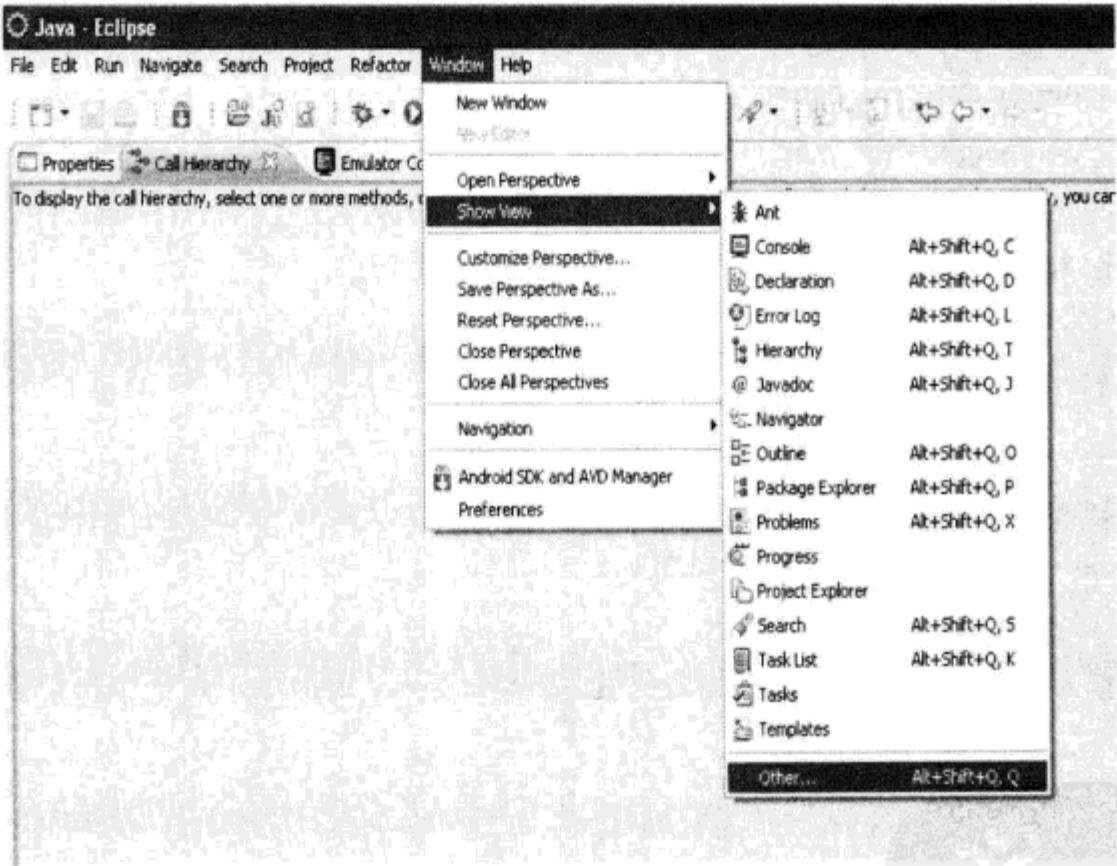


图 10-9 选择命令

Eclipse 会弹出 ShowView 对话框，找到 LogCat 选项并单击 OK 按钮，如图 10-10 所示。这时就进入了 LogCat 视图。该视图包含使用 Log 类输出的日志，如图 10-11 所示。LogCat 提供了过滤日志的控制按钮（图 10-11 的右上角），这些控制按钮可用来过滤相应的日志。过滤级别分别为：V（Verbose）、D（Debug）、I（Info）、W（Warning）、E（Error）。过滤日志时是向低级别兼容的，例如过滤 Error 级别的日志时，视图会不仅显示 E（Error）级别的日志，还显示 V（Verbose）、D（Debug）、I（Info）和 W（Warning）级别的日志。

“http\*”用来匹配所有以 http 开头的字符串，这是一个简单的正则表达式。正则表达式可被用来匹配更为复杂的文本，表 10-2 列举了常用的正则表达式。



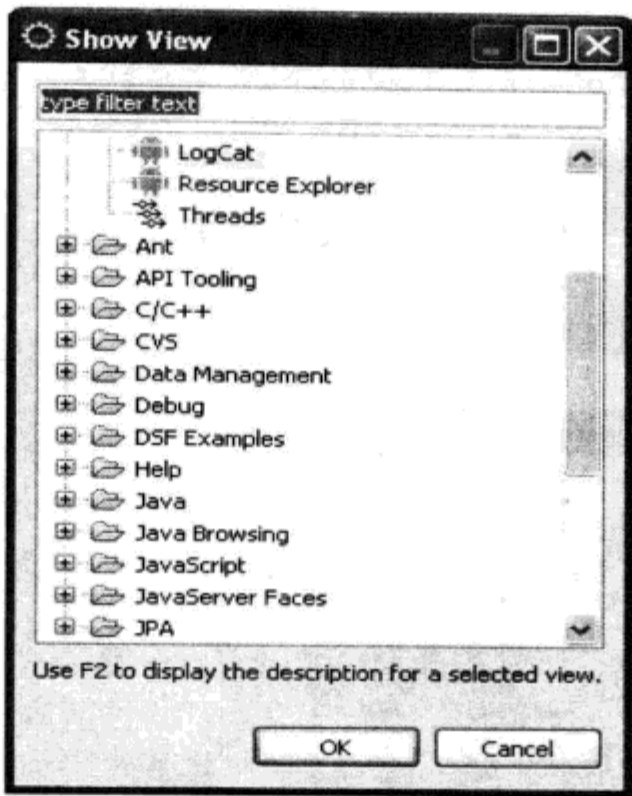


图 10-10 选择 LogCat 选项

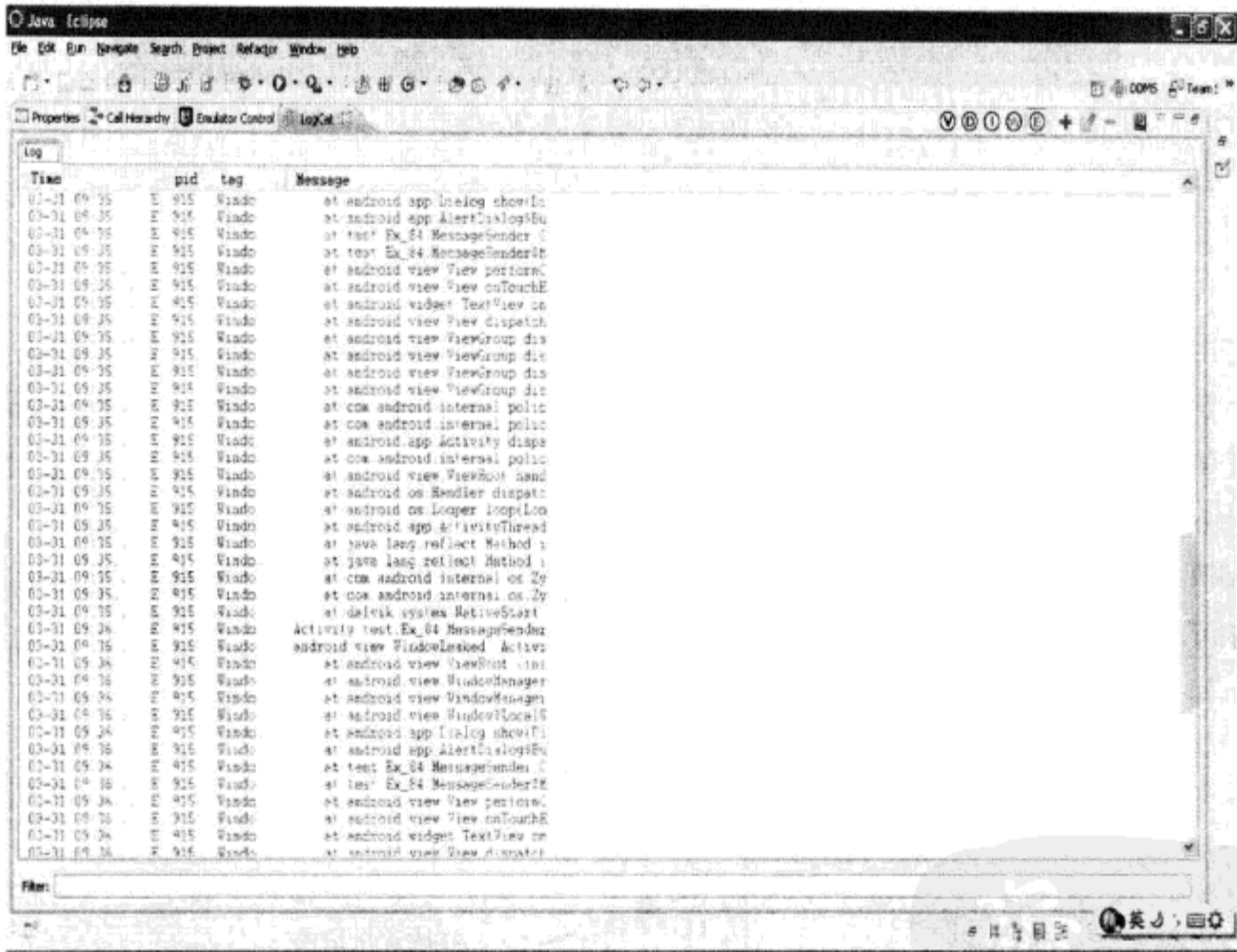


图 10-11 LogCat 视图

表 10-2 常用的正则表达式

匹配对象	正则表达式
中文字符	<code>[\u4e00-\u9fa5]</code>
电话号码	<code>(\d{3}-\d{4})?(\d{8} \d{7})?</code>
邮件地址	<code>\w+([-+.]\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*</code>
空行	<code>\n[\s]*\r</code>
网址	<code>^[a-zA-Z]+://(\w+(-\w+)*)(\.(\w+(-\w+)*))*(\?S*)?\$</code>
HTML 标记	<code>/&lt;(.*?)&gt;.*&lt;/&gt; &lt;(.*?) /&gt;</code>
首尾空格	<code>(^s*)(s*\$)</code>



在 Eclipse 下运行该 Android 工程，程序使用 main.xml 作为布局。图 10-12 显示了程序的启动界面。



图 10-12 程序启动界面

10.2 Socket 通信

本节将讲述 Socket 通信，Socket 通信是指通信双方采用 Socket 机制交换数据。Socket 通常也称做“套接字”，这个套接字用来描述通信链的句柄：IP 地址和端口。通过套接字，应用程序之间可传输信息。按照通信质量，网络通信可分为 TCP（Transmission Control Protocol）和 UDP（User Datagram Protocol）两种。TCP 协议是可靠的、面向连接的协议，这种方式需要在通信前建立通信双方的连接链路，通信结束后会释放该链路。而 UDP 协议是不可靠的、无连接的协议，这种协议不需要在通信前建立通信双方的连接。因而 UDP 使用可靠性来换取传输开销，其传输开销比 TCP 小。图 10-13 描述了 TCP 连接建立的过程。



图 10-13 TCP 连接建立过程



在 Android 系统中，java.net 包提供了 Socket 通信的接口。按照信息的传输方向，Socket 通信可分为客户端的 Socket 和服务端端的 Socket。java.net.Socket 是客户端的 Socket 对应的类，表 10-3 列举了 java.net.Socket 常用的方法。

表 10-3 java.net.Socket 常用方法

方法	功能描述	返回值
Socket	Socket 构造函数，Socket 提供了 7 种构造函数	null
bind(SocketAddress localAddr)	将该 Socket 同参数 localAddr 指定的地址和端口绑定	void
getInetAddress()	获取该 Socket 连接的目标主机的 IP 地址	InetAddress
getReceiveBufferSize()	获取该 Socket 的接收缓冲区的尺寸	synchronized int
close()	关闭 Socket	synchronized void
getInputStream()	获取该 Socket 的输入流，这个输入流用来读取数据	InputStream
isBound()	判断该 Socket 是否同本地的地址和端口绑定	boolean
isOutputShutdown()	判断该 Socket 的输出管道是否关闭	boolean
getLocalSocketAddress()	获取此 Socket 的本地地址和端口	SocketAddress
getPort()	获取端口号	int

java.net.ServerSocket 是服务器端的 Socket 对应的类，这个类表示一个等待客户端连接的服务器端套接字。ServerSocket 类处理连接要求并发送一个适当的答复。表 10-4 列举了 java.net.ServerSocket 常用的方法。

表 10-4 java.net.ServerSocket 常用方法

方法	功能描述	返回值
ServerSocket	ServerSocket 构造函数	null
accept()	等待客户端的连接	Socket
bind(SocketAddress localAddr)	将该服务器套接字同指定的 Socket 地址绑定。最多允许 50 个未建立的连接	void
bind(SocketAddress localAddr, int backlog)	将该服务器套接字同指定的 Socket 地址绑定	void
close()	关闭服务器 Socket	void
getReceiveBufferSize()	获取接收缓冲区的尺寸	int
isBound()	判断该 Socket 是否同本地的地址和端口绑定	boolean
isOutputShutdown()	判断该 Socket 的输出管道是否关闭	boolean
getLocalSocketAddress()	获取此 Socket 的本地地址和端口	SocketAddress
getLocalPort()	获取端口号	int
getInetAddress()	获取该 Socket 的 IP 地址	InetAddress
isClosed()	判断连接是否关闭	boolean
getReuseAddress()	获取地址	boolean
setSocketFactory(SocketImplFactory aFactory)	设置服务器套接字为该套接字工厂类的实例	void
setPerformancePreferences(int connectionTime, int latency, int bandwidth)	设置连接的性能偏好	void
setSoTimeout(int timeout)	设置接收的超时时间	void

下面以一个实例说明 Socket 的功能，通过该实例，读者可掌握 Socket 的基本用法。

【实例 10-3】Socket 的应用。

主程序 Server.java 实现，作为服务器端等待客户端的连接。该程序运行在 Windows 系统中，而不是 Android 系统。

```

import java.io.BufferedReader;    //导入 BufferedReader 类

```



```
import java.io.InputStream;           //导入输入流类, 用于获取输入数据
import java.io.InputStreamReader;    //导入 InputStreamReader 类
import java.io.OutputStream;         //导入输出流类, 用于获取输出数据
import java.io.PrintWriter;         //导入 PrintWriter 类
import java.net.ServerSocket;        //导入 ServerSocket 流
import java.net.Socket;              //导入套接字流

public class Server {
    private int ServerPort = 8886;    //定义端口监听
    private ServerSocket serversocket = null; //声明服务器套接字
    private OutputStream outputStream = null; //声明输出流
    private InputStream inputStream = null;  //声明输入流
    private PrintWriter printWriter = null;  //声明 PrintWriter 对象, 用于将数据发送给对方
    private Socket socket = null;           //声明套接字, 注意同服务器套接字不同
    private BufferedReader reader = null;    //声明 BufferedReader 对象, 用于读取接收的数据

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new Server();
    }
    /*程序入口, 程序从 main 函数开始执行*/

    public Server() {
        try
        {
            serversocket = new ServerSocket(ServerPort);
            //根据指定的端口号, 创建套接字。该套接字在指定的端口上提供网络通信
            System.out.println("Waiting client...");
            socket = serversocket.accept();           //用 accept 方法等待客户端的连接
            System.out.println("Client is connected\n");
        }
        catch (Exception ex)
        {
            ex.printStackTrace(); //打印异常信息
        }

        try
        {
            outputStream = socket.getOutputStream(); //获取套接字输出流
            inputStream = socket.getInputStream();     //获取套接字输入流
        }
    }
}
```



```

        printWriter = new PrintWriter(outputStream, true); //根据 outputStream 创建 PrintWriter 对象
        reader = new BufferedReader(new InputStreamReader(inputStream));
        //根据 inputStream 创建 BufferedReader 对象
        BufferedReader in = new BufferedReader ( new InputStreamReader(System.in));
        //根据 System.in 创建 BufferedReader 对象
        while (true)
        {
            String message = reader.readLine(); //接收客户端的传输信息
            System.out.println("Client: " + message); //将接收的信息打印出来
            if (message.equals("Bye") || message.equals("bye"))
                break;
            //若消息为 Bye 或者 bye, 则结束通信
            message = in.readLine(); //接收键盘输入
            printWriter.println(message); //将输入的信息向客户端输出
        }
        outputStream.close(); //关闭输出流
        inputStream.close(); //关闭输入流
        socket.close(); //关闭套接字
        serversocket.close(); //关闭服务器套接字
        System.out.println("Client is disconnected");
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        e.printStackTrace(); //打印异常信息
    }
    /*catch 捕获 try 块产生的异常*/
    finally
    {
        //TODO
    }
    /*finally 最后被调用*/
}
/*Server 类的构造函数*/
}

```

Client.java 实现, 作为客户端运行在 Android 系统中。客户端通过套接字绑定服务器端的 IP 地址和端口号。

```
package test.Ex_103;
```

```
import android.app.Activity; //导入活动类, 创建应用
```



```

import android.os.Bundle;           //导入 Bundle 类
import java.io.BufferedReader;       //导入 BufferedReader 类, 读取输入流
import java.io.BufferedWriter;       //导入 BufferedWriter 类, 读取输出流
import java.io.InputStreamReader;   //导入 InputStreamReader 类, 读取输出流
import java.io.OutputStreamWriter;   //导入 OutputStreamWriter 类
import java.io.PrintWriter;         //导入 PrintWriter 类
import java.net.InetAddress;         //导入 InetAddress 类, 用于表示网络上的 IP 地址
import java.net.Socket;              //导入套接字类
import android.app.AlertDialog;      //导入提示对话框类
import android.content.DialogInterface; //导入对话框接口类
import android.os.Handler;          //导入处理器类
import android.os.Message;           //导入信息类
import android.util.Log;             //导入日志类
import android.view.View;            //导入视图类
import android.widget.Button;         //导入按钮类
import android.widget.EditText;       //导入编辑框类
import android.widget.TextView;       //导入文本框视图类


public class Client extends Activity implements Runnable {
    /** Called when the activity is first created. */
    private TextView chatmessage = null; //声明文本框视图 chatmessage, 用于显示聊天记录
    private EditText sendmessage = null; //声明编辑框 sendmessage, 用于用户输入短信内容
    private Button send_button = null;   //声明按钮 send_button, 用于发送短信
    private static final String HOST = "192.168.1.100"; //服务器的 IP 地址
    private static final int PORT = 8886; //服务器的端口号
    private Socket socket = null;        //声明套接字类, 传输数据
    private BufferedReader bufferedReader = null; //声明 BufferedReader 类, 读取接收的数据
    private PrintWriter printWriter = null;    //声明 printWriter 类, 用于将数据发送给对方
    private String string = "";              //声明字符串变量

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //必须执行 onCreate 方法
        setContentView(R.layout.main);    //使用 main.xml 生成程序 UI
        chatmessage = (TextView) this.findViewById(R.id.chatmessage); //根据 XML 指定的属性创建 chatmessage
        sendmessage = (EditText) this.findViewById(R.id.sendmessage); //根据 XML 指定的属性创建 sendmessage
        send_button = (Button) this.findViewById(R.id.SendButton); //根据 XML 指定的属性创建 SendButton
    }
    try

```



```

{
    socket = new Socket(InetAddress.getByName(HOST), PORT); //指定 IP 和端口号创建套接字
    bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    //使用套接字的输入流构造 BufferedReader 对象
    printWriter = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.
        getOutputStream()),true);
    //使用套接字的输出流构造 PrintWriter 对象
}
/*try 块包含可能出现异常的代码*/
catch (Exception e)
{
    e.printStackTrace(); //打印异常
    CreateDialog(e.getMessage()); //调用 CreateDialog 方法生成对话框
}
/*catch 块捕获异常*/

send_button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View view) {
        String message = sendmessage.getText().toString(); //获取短信输入框的内容
        if (socket.isConnected())
        {
            if (!socket.isOutputShutdown()) {
                printWriter.println(message); //将短信输入框内容发送到服务器
                chatmessage.setText( chatmessage.getText().toString()+ "\n" + "Client: " +
                    message);
                //设置 chatmessage 的内容
                sendmessage.setText(""); //清空 sendmessage 的内容以便下次输入
            }
        }
        /*判断 Socket 是否连接*/
    }
});
/*注册 send_button 的鼠标单击监听器, 当单击按钮时, 发送指定的短信*/
new Thread(this).start(); //启动线程
}

public void CreateDialog(String msmessage) {
    android.app.AlertDialog.Builder builder = new AlertDialog.Builder(this);
    //首先获取 AlertDialog 的 Builder 类, 该 Builder 对象用于构造对话框
    builder.setTitle("异常"); //指定对话框的标题
    builder.setMessage(msmessage); //设置显示的信息
}

```



```

builder.setPositiveButton("Yes",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
        }
    });
//设置 PositiveButton 的名称以及监听器
builder.setNegativeButton("No",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
        }
    });
//设置 NegativeButton 的名称以及监听器
builder.show(); //显示对话框
}
/*CreateDialog 产生对话框*/

public void run() {
    try {
        while (true) {
            if(socket.isConnected()){
                if(!socket.isInputShutdown()){
                    if ((string = bufferedReader.readLine()) != null) {
                        Log.i("TAG", "++ "+string);
                        string += " ";
                        messegner.sendMessage(messegner.obtainMessage());
                    }
                }
            } else{
                //TODO
            }
        }
        //若套接字同服务器的连接存在且输入流存在，则发送消息
    }

}

} catch (Exception ex) {
    ex.printStackTrace(); //显示异常信息
    Log.w("TAG", "-- "+ex.toString()); //将信息输出到日志
}

```



```

}
/*线程运行的代码*/

public Handler messegner = new Handler() {
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        Log.i("TAG", "-- "+msg); //将消息打印到日志
        chatmessage.setText( chatmessage.getText().toString() + "\n" + "Server: " + string );
    }
};
/*创建 Hnadler 对象 messegner*/
}

```

【代码说明】 本实例实现了使用 Socket 进行通信。使用 Socket 进行双方通信需要至少实现客户端和服务端两部分。

#### (1) 服务器端实现

①使用 ServerSocket 类的构造函数创建服务器套接字, ServerSocket 提供了 4 种构造函数:

- ServerSocket()
- ServerSocket(int aprot)
- ServerSocket(int aprot, int backlog)
- ServerSocket(int aprot, int backlog, InetAddress localAddr)

本实例使用 ServerSocket(ServerPort)创建套接字。这个函数会根据指定的端口号和本地的 IP 地址创建套接字。该套接字在指定的端口上提供网络通信。若有多个本地 IP 地址, 可使用 ServerSocket(int aprot, int backlog, InetAddress localAddr) 方法指定创建套接字的地址。

②accept 方法等待客户端的连接。该方法会阻塞程序运行, 直到监听到客户端的连接。若有客户端连接, 则返回客户端的 Socket 对象。

③处理输入和输出信息。根据 accept 方法返回的 Socket 对象的 getInputStream()方法和 getOutputStream()方法获取其输入流和输出流。输入流可获取客户端的 Socket 携带的信息, 而输出流将服务端的信息发送给客户端。除了使用 getOutputStream()方法获取其输出流外, 服务器端需要创建一个输入流来获取本地的输入, 如键盘输入。然后将此输入作为信息通过 getOutputStream()方法获取的输出流发送给客户端。处理输出和输出信息的步骤如图 10-14 所示。

输入和输出流是由 Java.io 包提供的, 它支持两种类型的数据流: 字节流 (binary stream) 和字符流 (character stream)。虽然字节流与和字符流的使用方式相似, 但读取文件的方式是不同的。

字符流的操作单位是一个 Unicode (两个字节), 且字符流不能直接读写文件, 需通过缓冲区才能读写文件。而字节流的操作单位是一个字节, 可不通过缓冲区直接读写文件。字节流又可分为字节输入流和字节输出流, 这两种输入流对应的超类分别为 InputStream 和 OutputStream。字符流也可分为字节输入流和字节输出流, 这两种输入流对应的超类分别为 Reader 和 Writer。



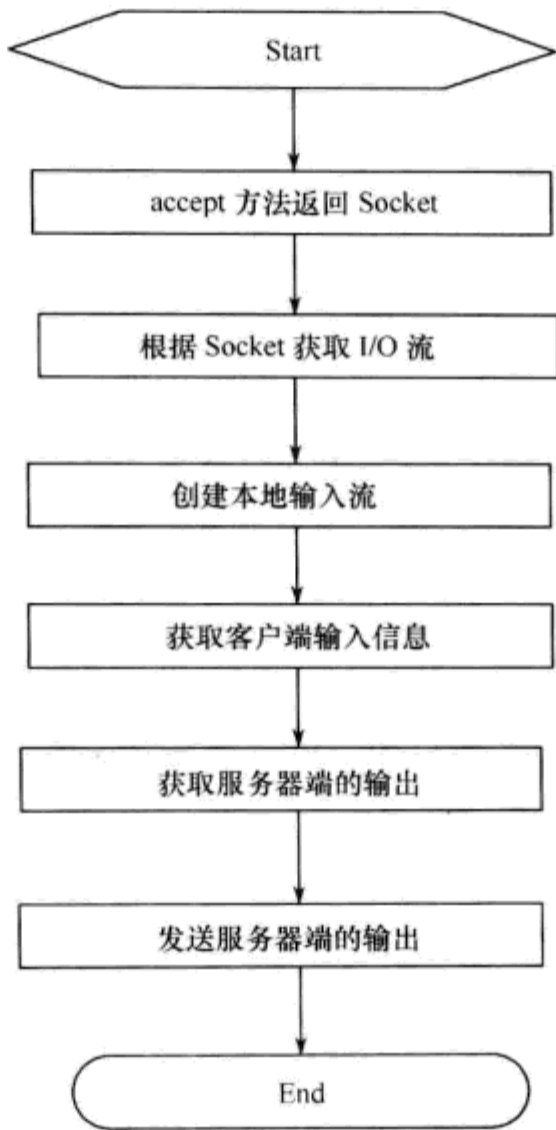


图 10-14 处理输出和输出信息

④若通信结束，则使用 close 方法关闭创建的对象，如：

```

    outputStream.close(); //关闭输出流
    inputStream.close(); //关闭输入流
    socket.close(); //关闭套接字
    serversocket.close(); //关闭服务器套接字
    
```

(2) 客户端实现

使用客户端 Socket 类构造 Socket 对象。例如：socket = new Socket(InetAddress.
 getByName(HOST), PORT)。注意，这里的 IP 地址是服务器的 IP 地址。即使服务器端和 Android
 的模拟器在同一机器上运行，也不能使用回环地址（127.0.0.1）作为服务器的 IP 地址。需要在
 DOS 环境下通过 ipconfig 命令查看本地的 IP 地址，这个地址就是服务器的 IP 地址，如图 10-15
 所示。



图 10-15 获取服务器端的 IP 地址



因此，在客户端中指定的 IP 地址应为本地地址，如 192.168.1.100，若指定回环地址，程序会出现拒绝连接的错误。

本实例使用 Handler 发送服务器端的消息。Handler 为 Android 提供了消息处理机制，这种处理机制是异步的。对于发送和接收信息，Handler 有不同的处理方式，向消息队列中发送消息时会立即返回，而从消息队列中接收消息时会阻塞，其中从消息队列中读取消息时会执行 Handler 中的 `public void handleMessage(Message msg)` 方法。因此，在创建 Handler 时应该使用匿名内部类重写该方法，在该方法中添加读取到消息后的操作。例如，使用 Handler 的 `obtainMessage()` 来获得消息对象，使用 Handler 的 `post` 方法将 Runnable 对象放到 Handler 的线程队列中。注意，该 Runnable 的执行其实并未单独开启线程，而是仍然在当前 Activity 线程中执行，Handler 只是调用了 Runnable 对象的 `run` 方法。

下面运行该实例，首先启动服务器端。运行 DOS，使用 `javac` 命令编译 `Server.java`，使用 `java Server` 命令运行该程序。图 10-16 显示了服务器端的运行结果。

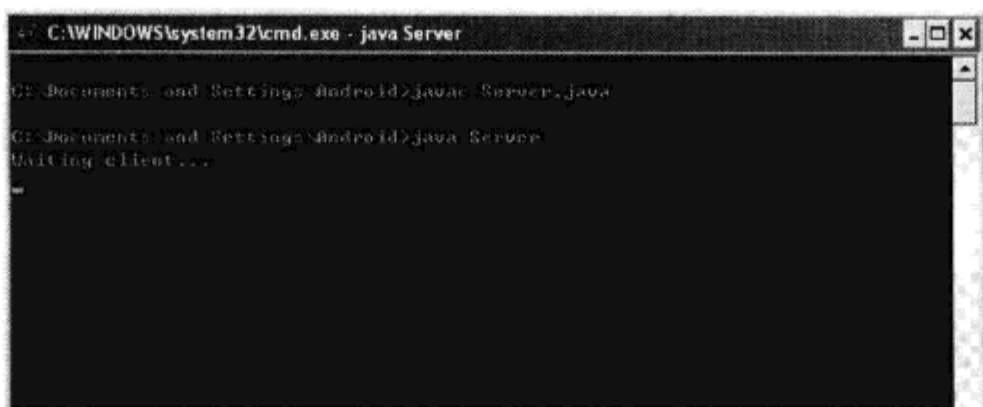


图 10-16 服务器端的运行结果

然后在 Eclipse 下运行 Client 程序。该程序运行在 Android 系统中，程序启动界面如图 10-17 所示。

Client 程序启动后，Client 建立了同服务器端的连接。图 10-18 显示服务器端监听到客户端的连接。

在客户端中发送“Hello, Server”给服务器，图 10-19 显示服务器端接收到该消息。

接收到客户端的消息后，服务器端给客户端发送“Hello, Client. I hava received your message”。图 10-20 显示了服务器给客户端发送的消息。

客户端将接收到的服务器的消息显示在文本框视图中，如图 10-21 所示。

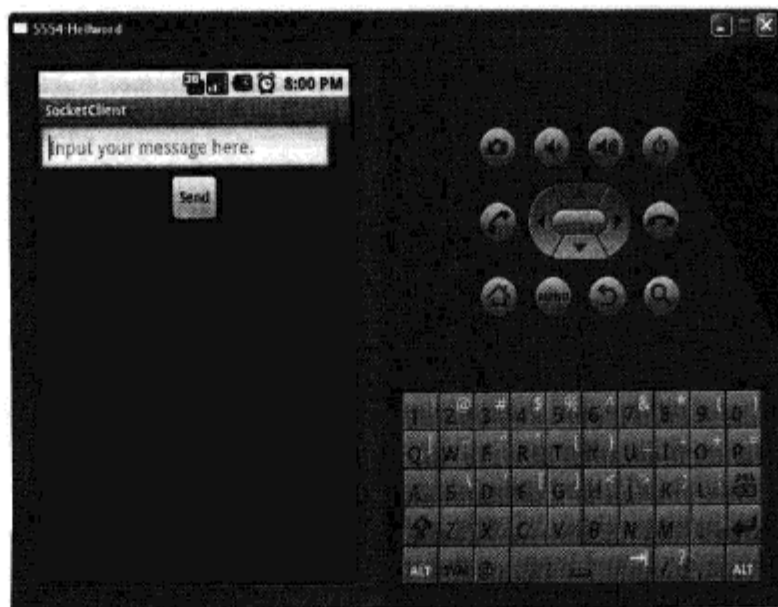


图 10-17 服务器端的运行结果





图 10-18 服务器端监听到客户端

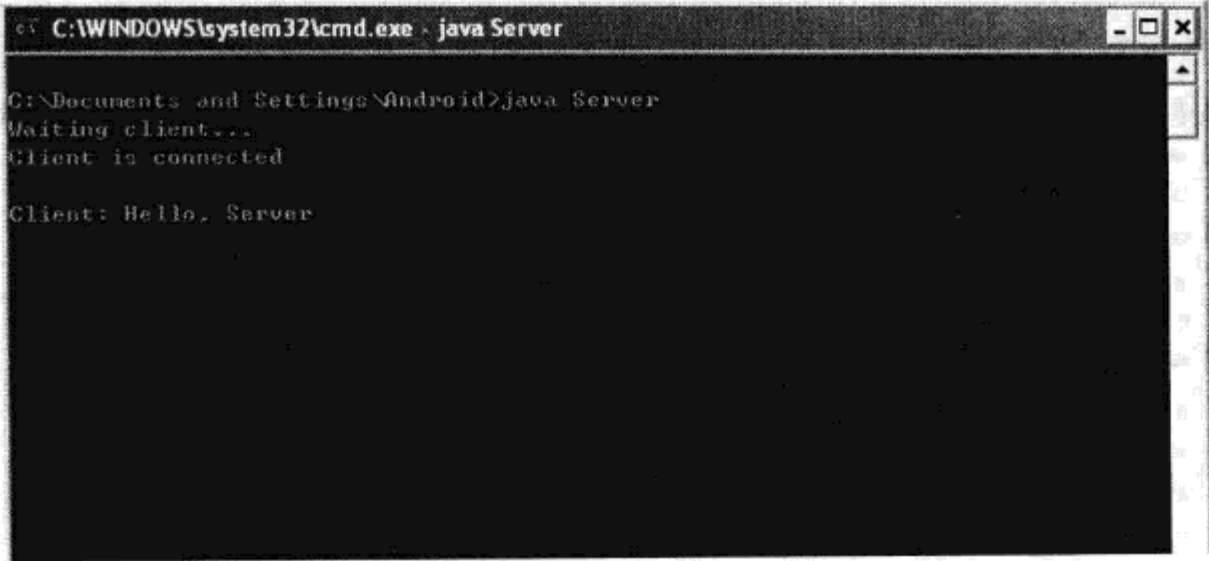


图 10-19 服务器端接收到客户端的消息

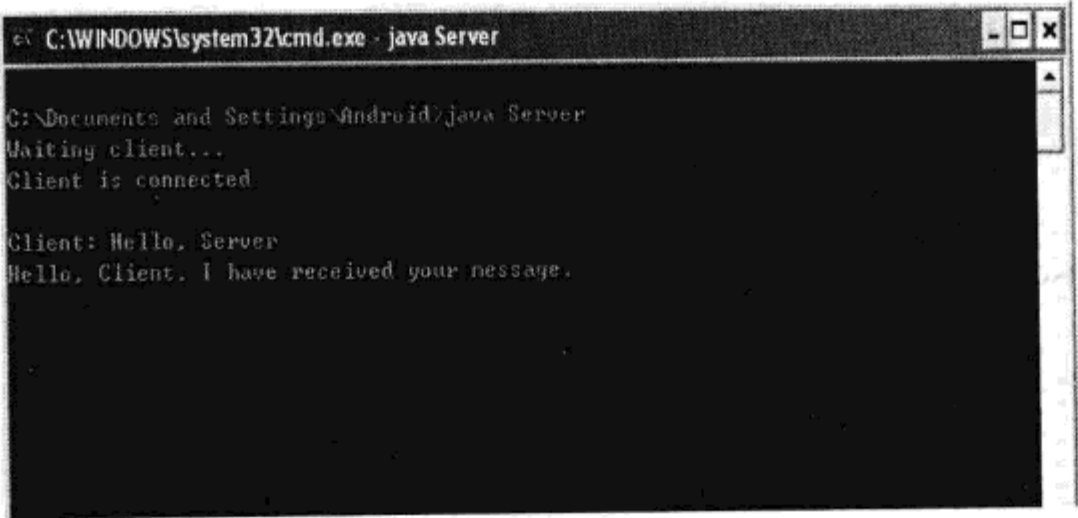


图 10-20 服务器端给客户端发送消息



图 10-21 客户端接收到服务器端的消息



10.3 应用实例详解：手机短信程序

10.3.1 实例分析

手机提供的基本服务是发短信和打电话，其中短信是手机的不可缺少的应用。通过手机提供的短信功能，用户可以与好友进行聊天、发送祝福等。在 Android 系统中，如何才能实现短信的应用呢？本节将以一个手机短信的实例为读者讲述开发手机短信应用的过程。

在前几章的实例中，很多使用 Toast 作为消息的显示方式。Toast 消息本身不会拥有程序焦点，不会与用户进行交互，并且在显示一段时间后自动消失。Toast 通常用来提示用户的操作的状态，如存储数据库成功。除了 Toast 消息之外，还有一种信息的提示方式是 Dialog（对话框）。Dialog 的提示方式与 Toast 大不相同，其在执行时会拥有程序的控制权，且可与用户进行交互。而程序往往通过用户的操作来决定下一步的操作。通常，Dialog 提供了两个用户交互按钮（例如 PositiveButton 和 NegativeButton），这两个按钮分别代表“是”和“否”两个状态。AlertDialog.Builder 提供了多种方法来定制对话框，表 10-5 列举了 Builder 的常用方法。

表 10-5 Builder 常用方法

方法	功能描述	返回值
Builder	Builder 构造函数	null
setAdapter(ListAdapter adapter, DialogInterface.OnClickListener listener)	在 AlertDialog 上设置多个选项，选项由参数 adapter 指定，监听器由参数 listener 指定	AlertDialog.Builder
setNegativeButton(CharSequence text, DialogInterface.OnClickListener listener)	设置对话框的“否”按钮显示的标题及其监听器。参数 text 指定按钮显示的文本，参数 text 指定该按钮的监听器。当用户单击该按钮时，listener 的 onClick 方法被调用	AlertDialog.Builder
setPositiveButton(CharSequence text, DialogInterface.OnClickListener listener)	设置对话框的“是”按钮显示的标题及其监听器。参数 text 指定按钮显示的文本，参数 text 指定该按钮的监听器。当用户单击该按钮时，listener 的 onClick 方法被调用	AlertDialog.Builder
setTitle(CharSequence title)	设置对话框显示的标题，参数 title 指定显示的标题	AlertDialog.Builder
setTitle(int titleId)	设置对话框显示的标题，参数 titleId 指定显示的标题的资源	AlertDialog.Builder
setNeutralButton(CharSequence text, DialogInterface.OnClickListener listener)	设置对话框的“取消”按钮显示的标题及其监听器。参数 text 指定按钮显示的文本，参数 text 指定该按钮的监听器。当用户单击该按钮时，listener 的 onClick 方法被调用	AlertDialog.Builder

Android 提供了 android.telephony.SmsManager 类用于发送短信，该类是 java.lang.Object 的子类。

android.telephony.SmsManager 类提供了短信业务的操作，例如发送数据、文本、SMS 消息。SmsManager 类只提供了五个方法，且没有构造函数。只能通过调用静态方法 SmsManager.getDefault()获取短信管理器对象。表 10-6 列举了 SmsManager 的常用方法。

表 10-6 SmsManager 常用方法

方法	功能描述	返回值
getDefault()	SmsManager 类没有构造函数，只能通过该方法获取短信管理器对象，该方法是一个静态的方法	SmsManager
divideMessage(String text)	该方法将过长的短信分割，参数 text 指定欲分割的短信。该方法返回一个字符串的数组列表	ArrayList<String>



续表

方法	功能描述	返回值
sendDataMessage(String dest,String scAddress, short port, byte[] data, PendingIntent p1, PendingIntent p2)	发送数据短信，参数 dest 指定目的地址，scAddress 指定源地址，port 指定接收信息的端口号，data 指定发送的消息，参数 p1 指定消息成功发送或失败时广播的 PendingIntent，参数 p2 指消息成功传送到接收者时广播的 PendingIntent	void
sendMultipartTextMessage(String dest, String scAddress, ArrayList <String> parts, ArrayList<Pending Intent>p1, PendingIntent, 参数 p2 指消息成功传送到接收者时广播的 PendingIntent rrayList<PendingIntent> p2)	发送多条短信，参数 dest 指定目的地址，scAddress 指定源地址，parts 指定发送的消息数组列表，参数 p1 指定消息成功发送或失败时广播的 PendingIntent，参数 p2 指消息成功传送到接收者时广播的 PendingIntent	void
sendTextMessage(String dest, String scAddress, String text, PendingIntent p1, PendingIntent p2)	发送文本短信，参数 dest 指定目的地址，scAddress 指定源地址，text 指定发送的消息文本，参数 p1 指定消息成功发送或失败时广播的 PendingIntent，参数 p2 指消息成功传送到接收者时广播的 PendingIntent	void

下面讲述程序的实现过程，首先在清单文件中添加用户权限：

```

<uses-permission android:name="android.permission.SEND_SMS"/>
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".MessageSender"
        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>
    
```

然后构造程序的布局 main.xml，程序采用 AbsoluteLayout 布局，子元素垂直分布，高度和宽度同父元素相同。布局包含一个 AutoCompleteTextView、一个 EditText 和一个 Button 控件。AutoCompleteTextView 作为号码输入框，EditText 作为短信输入框，在指定的位置（0px，90px）上显示。Button 控件在指定的位置（0px，30px）上显示,高度随内容变化且宽度同父元素相同。

接着实现主程序 MessageSender.java，本程序定义了 6 个方法和 3 个私有类：

1. MessageSender.java 的方法

（1）public void onCreate(Bundle savedInstanceState)

该方法是 Activity 的入口，程序运行后该方法被执行。该方法首先需要执行父类的 onCreate 方法，然后使用 main.xml 初始化程序 UI。通过 findViewById()取得 AutoCompleteTextView 对象、Button 以及 EditText 对象。接着使用 SmsManager.getDefault()创建一个 SmsManager 对象。最后设置 Button 对象的 OnClickListener 来监听 OnClick 事件。

（2）public static boolean VaildNumberLen(String phone\_number, int mixlen, int maxlen)

该方法从号码输入框中获取号码的长度，并判断输入的电话号码的长度是否在 mixlen 跟 maxlen 之间：若长度正确返回 ture；否则返回 false。

（3）public static boolean ValidDigitalPhoneNumber(String phone\_number, String regexp)

该方法检查电话号码是否包含正则表达式定义的字符串：若包含返回 ture；否则返回 false。其中，参数 phone\_number 指定电话号码，参数 regexp 指定正则表达式。通过 Matcher 对象的 matches 方法判断匹配是否成功，即 phone\_number 是否含有正则表达式 regexp 定义的字符串。



(4) `public void sendMessage(SmsManager smsManager, String destination, String source, String strMessage, PendingIntent pintent, PendingIntent dintent)`

该方法用于发送短信。首先判断发送短信的长度是否超过最大限制,若超过最大限制,则使用 `divideMessage` 方法拆分短信后发送,否则不拆分短信直接发送。

(5) `public int MessageLen(String message)`

用于计算短信长度,该方法在 `sendMessage` 中被调用。

(6) `void CreateDialog(String PositiveButtonText, String NegativeButtonText, String title, PositiveButtonListener positiveButtonListener, NegativeButtonListener negativeButtonListener)`

该方法使用 `Builder` 构造对话框 `PositiveButton` 的标题和单击监听器、`NegativeButton` 的标题和单击监听器,并使用 `show()` 方法显示对话框。其中,参数 `PositiveButtonText` 为 `PositiveButton` 显示的标题,参数 `NegativeButtonText` 为 `NegativeButton` 显示的标题,参数 `title` 为 `Dialog` 的标题,参数 `positiveButtonListener` 为 `PositiveButton` 的单击监听器,参数 `negativeButtonListener` 为 `NegativeButton` 的单击监听器。该方法被私有类 `MyButtonClickListener` 的 `onClick` 方法调用,用来同用户交互是否发送短信。

## 2. MessageSender.java 的私有类

(1) `private class MyButtonClickListener implements OnClickListener`

该类是 `Button` 控件的单击事件监听器,该类实现了 `onClick` 方法。当用户单击“发送”按钮时,使用 `ValidDigitalPhoneNumber` 和 `VaildNumberLen` 判断号码的有效性。当号码有效时,根据短信的长度弹出对话框。若为空信息,弹出对话框提示用户是否“确定发送空信息?”;若长度大于 70,弹出对话框提示用户是否“确定发送超长信息?”;若长度小于 70,弹出对话框提示用户是否“确定发送信息?”。若输入的号码包含字母或长度不在 4~9 之间,则该号码为无效号码,程序弹出“异常错误”的 `Toast` 消息。

(2) `private class NegativeButtonListener implements DialogInterface.OnClickListener`

定义私有类 `NegativeButtonListener` 用于实现 `PositiveButton` 的单击事件监听,该类实现对话框监听事件的接口类 `DialogInterface.OnClickListener`。

(3) `private class PositiveButtonListener implements DialogInterface.OnClickListener`

定义私有类 `PositiveButtonListener` 用于实现 `PositiveButton` 的单击事件监听,该类实现对话框监听事件的接口类 `DialogInterface.OnClickListener`。

为了模拟发送短信的应用,需要两个 Android 终端。通过 Eclipse 只能创建一个终端,可通过 Emulator 提供的命令创建另外一个模拟器。Emulator 提供了许多命令用于管理终端,在 Dos 下输入 `emulator -help` 命令进行查看。在 DOS 下输入 `emulator -data sms`,可创建 Android 客户端。

## 10.3.2 实例实现

`MessageSender.java` 实现,用于将短信发送给 Emulator 启动的 Android 终端。

```
package test.Ex_104;                                //程序打包

import android.app.Activity;                        //导入活动类,创建应用程序
import android.app.AlertDialog;                    //导入对话框类
import android.app.PendingIntent;                  //导入 PendingIntent 类
import android.app.AlertDialog.Builder;            //导入 Builder 类,该类用于设置对话框的属性
```



```
import android.os.Bundle;           //导入 Bundle 类
import android.content.DialogInterface; //导入对话框接口类, 该类提供了对话框的单击事件监听器
import android.content.Intent;      //导入 Intent 类, 传递消息
import android.telephony.SmsManager; //导入短信管理器类, 使用该类发送短信
import android.view.View;           //导入视图类
import android.view.View.OnClickListener; //导入单击事件监听器
import android.widget.AdapterView;      //导入数组适配器类, 存储自动完成文本的号码库
import android.widget.AutoCompleteTextView; //导入自动完成文本框视图类, 提供号码的自动提示
import android.widget.EditText;      //导入编辑框类
import android.widget.ImageButton; //导入图片按钮控件类
import android.widget.Button;        //导入按钮控件类
import android.widget.Toast;         //导入 Toast 类, 显示 Toast 消息
import android.net.Uri;             //导入 Uri 类
import java.util.ArrayList;
import java.util.regex.Matcher;      //导入匹配器类, 用于匹配指定的正则表达式
import java.util.regex.Pattern;      //导入模式类, 用于创建正则表达式

public class MessageSender extends Activity {
    /** Called when the activity is first created. */
    private static final String[] phonenumbers = new String[] { "88888888", "85668888", "7777777", "86666666",
        "7377777" }; //用于自动完成提示的号码表
    /*声明 Button 与 AutoCompleteTextView 对象名称*/
    private Button send_message_button; //声明短信发送按钮
    private EditText send_message_text; //声明短信文本编辑框
    private AutoCompleteTextView autoCompletePhone; //定义自动提示输入文本控件
    private SmsManager smsManager; //声明短信管理器对象 smsManager
    PendingIntent pendingIntent; //声明 PendingIntent 对象
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //必须执行父类的 onCreate 方法
        setContentView(R.layout.main); //使用 main.xml 初始化程序 UI
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        autoCompletePhone = (AutoCompleteTextView) findViewById(R.id.autoCompletePhone);
        /* 以 findViewById()取得 AutoCompleteTextView 对象 */
        send_message_button = (Button) findViewById(R.id.send_message_button);
        //构造 send_message_button 对象
        send_message_text = (EditText) findViewById(R.id.send_message_text);
        //构造 send_message_text 对象
        smsManager = SmsManager.getDefault(); //创建一个默认模式的 SmsManager 对象
        PendingIntent.getBroadcast(MessageSender.this, 0, new Intent(), 0); //创建一个广播的 Intent
    }
}
```



```

/*通过 findViewById 构造器来构造 EditText 与 Button 对象*/
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, android.R.layout.simple_
dropdown_item_1line, phonenumberStr);
/* 以 phonenumberStr 字符串数组生成 ArrayAdapter 对象 */
autoCompletePhone.setAdapter(adapter);
/* 通过 setAdapter()来读取 ArrayAdapter 里的数据: phonenumberStr*/
send_message_button.setOnClickListener(new MyButtonClickListener());
/*设置 Button 对象的 OnClickListener 来监听 onClick 事件*/
}

private class MyButtonClickListener implements OnClickListener{
    public void onClick(View view) {
        try
        {
            String receiver = autoCompletePhone.getText().toString();
            String message = send_message_text.getText().toString();
            String valid_expression = "[0-9]+$"; //设置 valid_expression 为仅包含数字的字符串
            String invalid_expression = "[a-zA-Z]+$"; //设置 valid_expression 为仅包含字母的字符串
            if (ValidDigitalPhoneNumber(receiver,valid_expression) && ! ValidDigitalPhoneNumber
                (receiver,invalid_expression)&& ValidNumberLen(receiver, 4, 9))
            {
                if(MessageLen(message)==0)
                    CreateDialog("Send anyway", "Cancel", "确定发送空信息?", new PositiveButtonListener(),
                        new NegativeButtonListener());
                //若为空信息, 弹出对话框提示用户是否发送
            }
            else if (MessageLen(message)>70)
                CreateDialog("Send anyway", "Cancel", "确定发送超长信息?", new
                    PositiveButtonListener(), new NegativeButtonListener());
                //若长度大于 70, 弹出对话框提示用户是否发送
            else
                CreateDialog("Send", "Cancel", "确定发送信息?", new PositiveButtonListener(),
                    new NegativeButtonListener());
                //若长度小于 70, 弹出对话框提示用户是否发送
            }
            /*若输入的号码包含数字, 不包含字母, 且长度在 4~9 之间, 则调用 Android 系统
            的拨号应用*/
            else
            {
                autoCompletePhone.setText("");
            }
        }
    }
}

```



```

        send_message_text.setText("");
        Toast.makeText(MessageSender.this, "无效电话号码格式", Toast.LENGTH_LONG).
            show();
    }
}
catch (Exception e)
{
    Toast.makeText(MessageSender.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
        show();
}
/*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
finally
{
    //TODO
}
/*finally 中可添加处理异常的代码*/
}
}

public static boolean VaildNumberLen(String phone_number, int mixlen, int maxlen)
{
    int len = phone_number.length(); //获取电话号码的长度
    if (len >= mixlen && len <= maxlen) //判断电话号码是否在 mixlen 跟 maxlen 之间
        return true;
    return false;
}
/*判断输入的电话号码的长度是否在 mixlen 跟 maxlen 之间: 若长度正确返回 ture; 否则返回 false*/

public static boolean ValidDigitalPhoneNumber(String phone_number, String regexp)
{
    Pattern pattern = Pattern.compile(regexp);
    /*根据正则表达式 regexp 生成模式对象*/
    Matcher matcher = pattern.matcher(phone_number);
    /*模式对象利用正则表达式 regexp 匹配 phone_number, 返回 Matcher 对象*/
    if (matcher.matches())
    {
        return true;
    }
    return false;
}
/* 通过 Matcher 对象的 matches 方法判断匹配是否成功

```



```

    * 即 phone_number 是否含有正则表达式 regexp 定义的字符串
    * 若包含 regexp 定义的字符串, 则返回真值 (true)
    * 否则返回假值 (false)
    **/
}

/*检查电话号码是否包含正则表达式定义的字符串: 若包含返回 true; 否则返回 false*/

public void sendMessage(SmsManager smsManager, String destination, String source, String strMessage,
PendingIntent pintent, PendingIntent dintent)
{
    if (MessageLen(strMessage)>70)
    {
        ArrayList<String> messagearray = smsManager.divideMessage(strMessage);
        for (String msg : messagearray) {
            smsManager.sendTextMessage(destination, source, msg, pintent, dintent);
        }
    }
    else
        smsManager.sendTextMessage(destination, source, strMessage, pintent, dintent);
    /*若长度大于 70, 则使用 divideMessage 方法拆分短信*/
}

/*sendMessage 方法用于发送短信*/

public int MessageLen(String message)
{
    return message.length();
}

/*sendMessage 方法计算短信长度*/

void CreateDialog(String PositiveButtonText,String NegativeButtonText, String title, PositiveButtonListener
positiveButtonListener, NegativeButtonListener negativeButtonListener)
{
    Builder mybuilder = new AlertDialog.Builder(this); //新建 Builder 对象, 该 Builder 对象用于构造对话框
    mybuilder.setPositiveButton(PositiveButtonText, positiveButtonListener);
    //设置 PositiveButton 的标题和单击监听器
    mybuilder.setNegativeButton(NegativeButtonText, negativeButtonListener);
    //设置 NegativeButton 的标题和单击监听器
    mybuilder.setTitle(title); //设置对话框的标题
    mybuilder.show(); //显示对话框
}

```



```

/* CreateDialog 根据指定的属性产生对话框:
 * PositiveButtonText: PositiveButton 显示的标题
 * NegativeButtonText: NegativeButton 显示的标题
 * title: Dialog 的标题
 * positiveButtonListener: PositiveButton 的单击监听器
 * negativeButtonListener: NegativeButton 的单击监听器
 */

```

```

private class PositiveButtonListener implements DialogInterface.OnClickListener
{
    public void onClick(DialogInterface dialoginterface,int which)
    {
        String receiver = autoCompletePhone.getText().toString(); //获取接收方地址
        String sender = null; //发送方地址, 可以置为 null (5554)
        String message = send_message_text.getText().toString(); //发送消息
        PendingIntent pintent = pendingIntent;
        PendingIntent dintent = null;
        sendMessage(smsManager,receiver , sender , message , pintent , dintent ); //发送短信
    }
    //发生单击事件时, 该发方法被执行
}

```

/\*定义私有类 PositiveButtonListener 用于实现 PositiveButton 的单击事件监听, 该类实现对话框监听事件的接口类 DialogInterface.OnClickListener\*/

```

private class NegativeButtonListener implements DialogInterface.OnClickListener
{
    public void onClick(DialogInterface dialoginterface,int which)
    {
        autoCompletePhone.setText("");
        send_message_text.setText("");
        //取消发送短信, 清空短信文本框和号码文本框
    }
    //发生单击事件时, 该发方法被执行
}

```

/\*定义私有类 NegativeButtonListener 用于实现 PositiveButton 的单击事件监听, 该类实现对话框监听事件的接口类 DialogInterface.OnClickListener\*/

```

}

```

Main.xml 实现程序布局, 采用 AbsoluteLayout 布局, 子元素垂直分布, 高度和宽度同父元素相同。

```

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

```



```

<!-- 采用 AbsoluteLayout 布局，子元素垂直分布，高度和宽度同父元素相同 -->
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

    <!-- AutoCompleteTextView 作为号码输入框，在指定的位置（0px，30px）上显示，高度随内容变化且宽度同父元素相同 -->
    <AutoCompleteTextView
        android:id="@+id/autoCompletePhone"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_x="0px"
        android:layout_y="30px"
        android:hint="Please enter phone number"
    />

```

**【代码说明】**本实例使用 SmsManager 实现了短信通信。在 Eclipse 下，启动该程序，图 10-22 显示了程序的运行界面。



图 10-22 程序运行界面

使用 Emulator 的 data 命令创建 Android 终端。在 DOS 下输入 emulator -data sms 命令，创建名为 5556 的 Android 客户端。这个 5556 是短信发送的号码，如图 10-23 所示。

在 SmsManager 程序的界面中，输入号码 5556 以及短消息，单击“Send Message”按钮，如图 10-24 所示。





图 10-23 使用 `emulator -data` 命令创建客户端

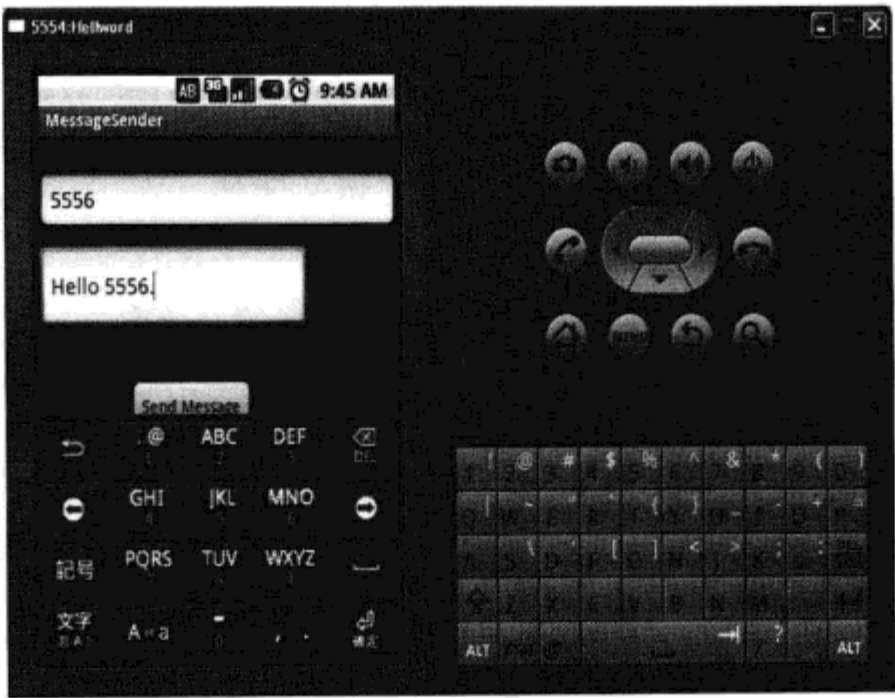


图 10-24 发送短消息

这时程序会弹出一个对话框，这个对话框用于同用户交互是否“确定发送信息？”，单击“Send”按钮，如图 10-25 所示。



图 10-25 确定发送短消息



客户端 5556 收到客户端 5554 发送的短消息，图 10-26 显示了客户端 5556 的短信记录。

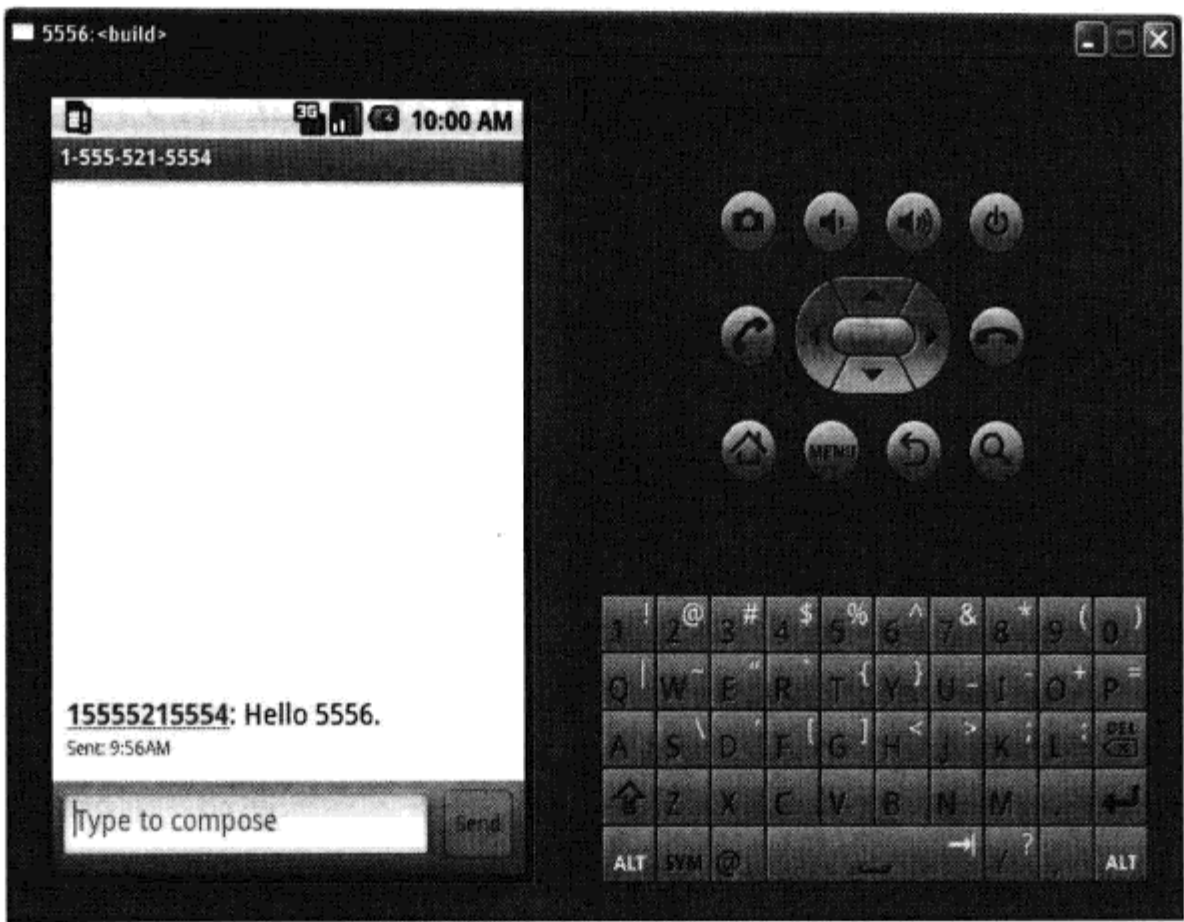


图 10-26 客户端 5556 的短信记录

客户端 5556 也可发送短消息给客户端 5554，图 10-27 显示客户端 5556 发送给客户端 5554 的消息。

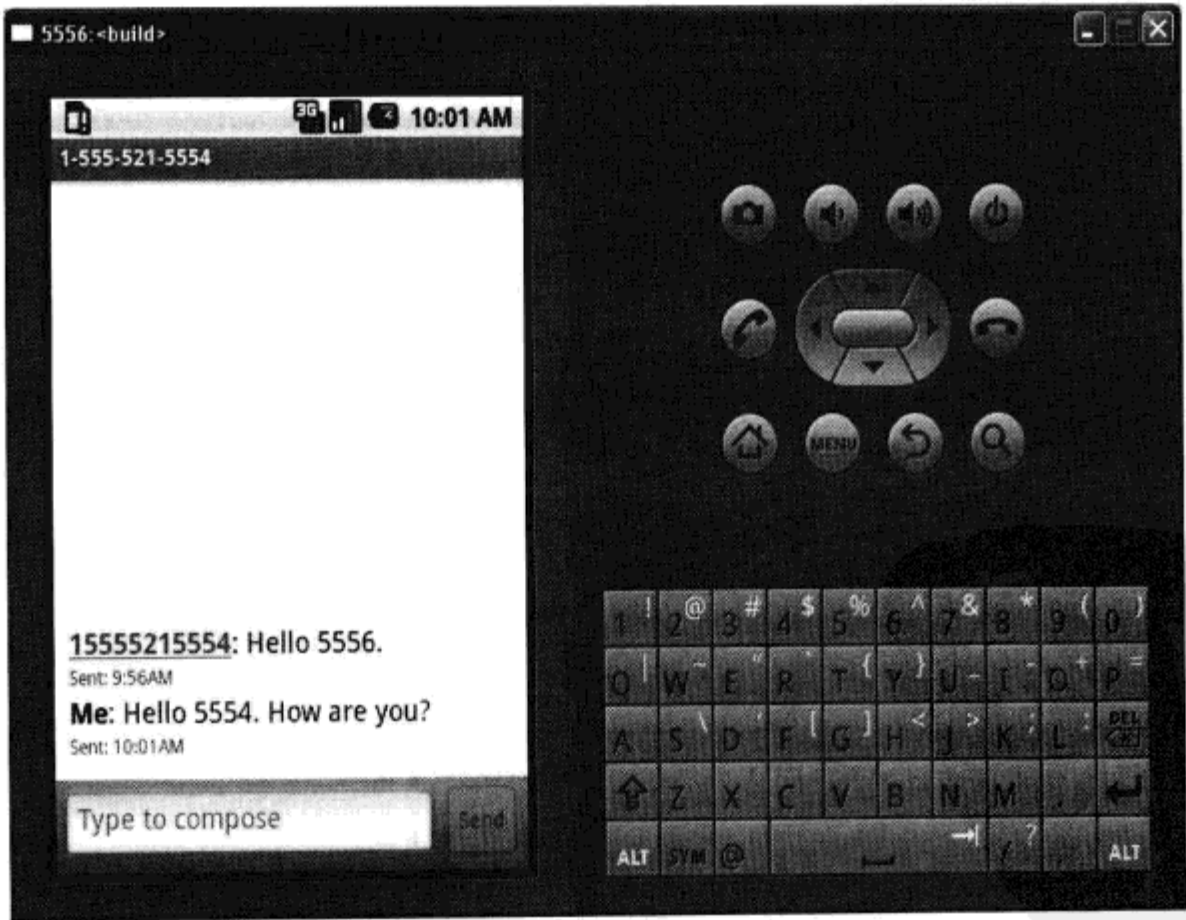


图 10-27 客户端 5556 发送给客户端 5554 的短消息



## 第 11 章 Android 的 GPS 应用开发

### 11.1 GPS 在手机中的应用

GPS（Global Positioning System）即全球定位系统，其最初是为陆、海、空三大领域提供实时、全天候和全球性的导航服务，并用于情报收集、核爆监测和应急通讯等一些军事目的。随着手机的普及，GPS 已成为手机的一个基本功能。通过手机的 GPS，使用者可以很方便地进行定位、查找以及查看好友的位置。一个完整的 GPS 由三部分组成：空间部分、地面控制系统以及用户设备。GPS 的空间部分是由若干个导航卫星组成的，它们均匀分布在 6 个轨道面上。由于这些卫星特殊的分布，全球任何地方、任何时间都可观测到 4 颗以上的卫星。GPS 的卫星因为大气摩擦等问题，随着时间的推移，导航精度会逐渐降低。地面控制系统由监测站（Monitor Station）、主控制站（Master Monitor Station）、地面天线（Ground Antenna）所组成。用户设备部分的主要功能是捕获按一定卫星截止角所选择的待测卫星，并跟踪这些卫星的运行。接收机捕获到跟踪的卫星信号后，就可测量出接收天线至卫星的伪距离和距离的变化率，解调出卫星轨道参数等数据。根据这些数据，接收机中的微处理计算机就可按定位解算方法进行定位计算，计算出用户所在地理位置的经纬度、高度、速度、时间等信息。

GPS 定位的基本原理是根据高速运动的卫星瞬间位置作为已知的起算数据，采用空间距离后方交汇的方法，确定待测点的位置。图 11-1 展示了 GPS 位置计算原理。

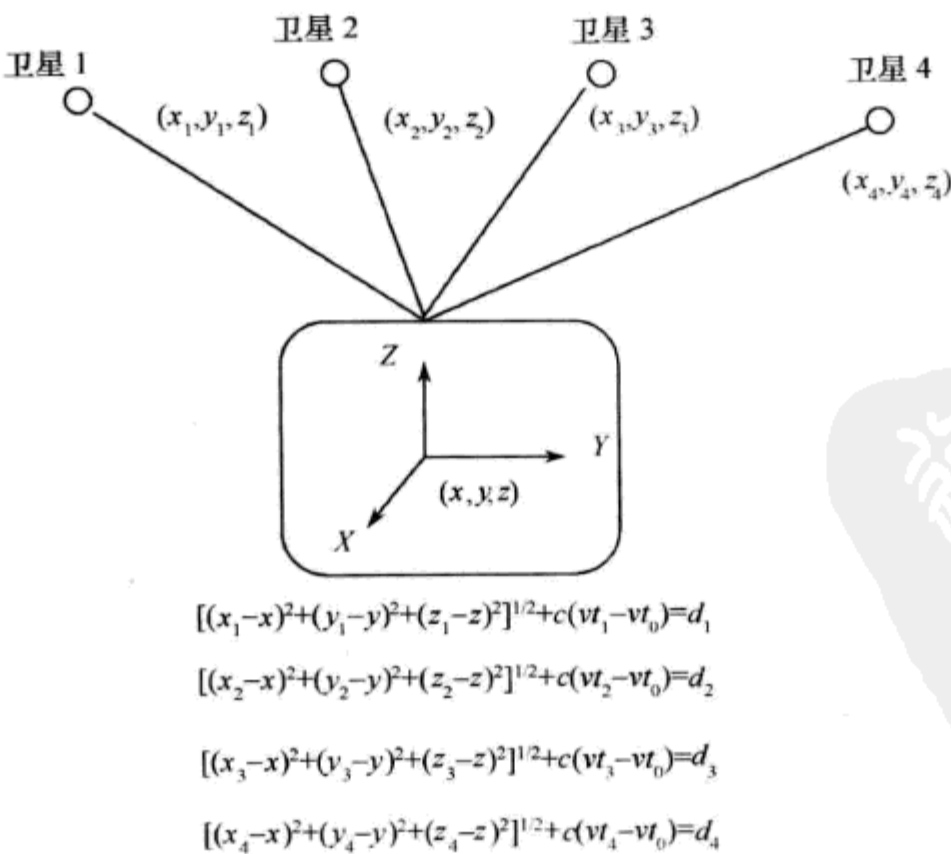


图 11-1 GPS 位置计算原理



11.2 Android Location-Based API 简介

Android 提供了有关位置的 API，这些 API 封装在 android.location 包里。通过这些 API，开发人员可实现位置相关的应用，下面是一个应用实例：

```

public Location getLocation(Context context) {
    LocationManager locMan = (LocationManager) context
        .getSystemService(Context.LOCATION_SERVICE);
    Location location = locMan
        .getLastKnownLocation(LocationManager.GPS_PROVIDER);
    if (location == null) {
        location = locMan
            .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
    }
    return location;
}

```

android.location 定义了 3 个接口（GpsStatus.Listener、GpsStatus.NmeaListener 和 LocationListener），这些接口是位置相关的监听器。另外，还定义了 7 个类（Criteria、Geocoder、GpsSatellite、GpsStatus、Location、LocationManager 和 LocationProvider），这些类提供了位置相关的接口。

表 11-1 列举了 android.location 包的主要类和接口。

表 11-1 android.location 包的主要类和接口

类或接口名	功能描述	类型
GpsStatus.Listener	GPS 状态改变监听器	接口
GpsStatus.NmeaListener	收到 NMEA 信息的监听器	接口
LocationListener	位置改变监听器	接口
Address	描述位置的类	类
Geocoder	该类提供了位置编码和反编码的功能	类
GpsSatellite	代表当前卫星状态的类	类
GpsStatus	代表 GPS 以前状态的类	类
Location	在某一特定时间内的地理位置	类
LocationManager	系统位置服务类	类
LocationProvider	位置提供商类	类

11.3 Android 模拟器支持的 GPS 定位文件

可通过 GPS 定位文件进行位置定位，Android 模拟器支持两种 GPS 定位格式：KML 和 NMEA。其中，KML（Keyhole Markup Language）是 Keyhole 标记语言，而 NMEA（National Marine Electronics Association）是美国海军电子协会制定的协议。

11.3.1 KML

KML 文件用于描述和保存地理信息（如点、线、图像、多边形和模型等），采用 XML 语法与格式，可以被 Google Earth 和 Google Map 识别并显示。KML 使用包含名称、属性的标



签 (tag) 来确定显示方式。Google Earth 处理 KML 文件的方式与网页浏览器处理 HTML 和 XML 文件的方式类似, Google Earth 用户可以使用 KML 来分享位置信息。

KML 是 XML 方案, 这个方案用于表达基于因特网、二维地图和三维地球地图浏览器的地理注释和可视化。KML 是为 Google Earth 而开发的一种语言, 是 Open Geospatial Consortium 的一个国际标准。Google Earth 是第一个够浏览和编辑 KML 文件的程序。为了在 Google Earth, Google Map 和移动终端上实现 KML, KML 指定一组特殊的标签 (位置标志、图像、多边形、3D 模型、文本描述等)。位置是由经度和纬度构成的, 其他的信息可以使视图更直观, 如倾斜、航向、海拔等。KML 共享了 GML 的部分语法, 因此一些 KML 信息不能在 Google Earth 中查看。下面是一个 KML 代码的例子:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
      at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683, 37.42228990140251, 0</coordinates>
    </Point>
  </Placemark>
</kml>
```

### 11.3.2 NMEA

NMEA 同时也是数据传输标准, 有几种不同的格式, 最常用的格式为 “GGA”, 它包含了定位时间、纬度、经度、高度、定位所用的卫星数、DOP 值、差分状态和校正时段等, 其他的有速度、跟踪、日期等。NMEA 实际上已成为所有的 GPS 接收机最常用的数据输出格式, 同时它也被用于 GPS 接收机接口的大多数的软件包里。例如:

```
$GPGGA,092204.999,4250.5589,S,14718.5084,E,1,04,24.4,19.7,M,0.0,0000*1F
```

字段 0: 语句 ID, 表明该语句为 Global Positioning System Fix Data (GGA) GPS 定位信息。

字段 1: UTC 时间 hhmmss.sss, 时分秒格式。

字段 2: 纬度 ddm. mmm, 度分格式 (前导位数不足则补 0)。

字段 3: 纬度 N (北纬) 或 S (南纬)。

字段 4: 经度 dddmm. mmm, 度分格式 (前导位数不足则补 0)。

字段 5: 经度 E (东经) 或 W (西经)。

字段 6: GPS 状态, 0=未定位, 1=非差分定位, 2=差分定位, 3=无效 PPS, 6=正在估算。

字段 7: 正在使用的卫星数量 (00~12) (前导位数不足则补 0)。

字段 8: HDOP 水平精度因子 (0.5~99.9)。

字段 9: 海拔高度 (-9999.9~99999.9)。

字段 10: 地球椭球面相对大地水准面的高度。

字段 11: 差分时间 (从最近一次接收到差分信号开始的秒数, 如果不是差分定位将为空)。

字段 12: 差分站 ID 号 0000~1023 (前导位数不足则补 0, 如果不是差分定位将为空)。



字段 13：校验值。

11.4 应用实例详解：确定当前位置的 GPS 程序

11.4.1 实例分析

本实例使用 android.location.Criteria 设置筛选位置的标准。android.location.Criteria 提供的常量如表 11-2 所示。

表 11-2 android.location.Criteria 提供的常量

常量	功能描述	值
ACCURACY_COARSE	模糊精度	2 (0x00000002)
ACCURACY_FINE	精确精度	1 (0x00000001)
NO_REQUIREMENT	没有要求	0 (0x00000000)
POWER_HIGH	高耗电	3 (0x00000003)
POWER_LOW	低耗电	1 (0x00000001)
POWER_MEDIUM	介于高耗电与低耗电之间	2 (0x00000002)
ACCURACY_MEDIUM	介于高精确与低精确之间	2 (0x00000002)
ACCURACY_LOW	低精确	1 (0x00000001)
ACCURACY_HIGH	高精确	3 (0x00000003)

android.location.Criteria 还提供了用于设置过滤标准的方法，表 11-3 列举了 android.location.Criteria 提供的方法。

表 11-3 android.location.Criteria 提供的方法

方法	功能描述	返回值
setAccuracy(int accuracy)	设置位置解析的精度, Criteria.ACCURACY_FINE 为精确模式和 Criteria.ACCURACY_COARSE 为模糊模式	void
setAltitudeRequired(boolean altitudeRequired)	是否提供海拔高度信息	void
setBearingRequired(boolean bearingRequired)	是否提供方向信息	void
setCostAllowed(boolean costAllowed)	是否允许运营商计费	void
setPowerRequirement(int level)	电池消耗, 如 Criteria.NO_REQUIREMENT	void
setSpeedRequired(boolean speedRequired)	是否提供速度信息	void

使用 Android 模拟器测试 GPS 程序时，需要设置 GPS 位置信息。有两种设置 GPS 位置信息的方法：DDMS 和 cmd。

第一种方法是使用 DDMS 的 Emulator Control 设置经纬度。在 Eclipse 下，单击 Window→Open Perspective→DDMS→Emulator Control→Manual，然后进行设置，如图 11-2 所示。

第二种方法是使用 cmd 设置经纬度。首先在 DOS 环境下，使用 telnet 命令连接 Android 模拟器，如图 11-3 所示。

注意，使用 telnet 命令指定的端口号是跟模拟器一致的（图 11-4 显示的标题，即 5554，就是该模拟器的端口号）。



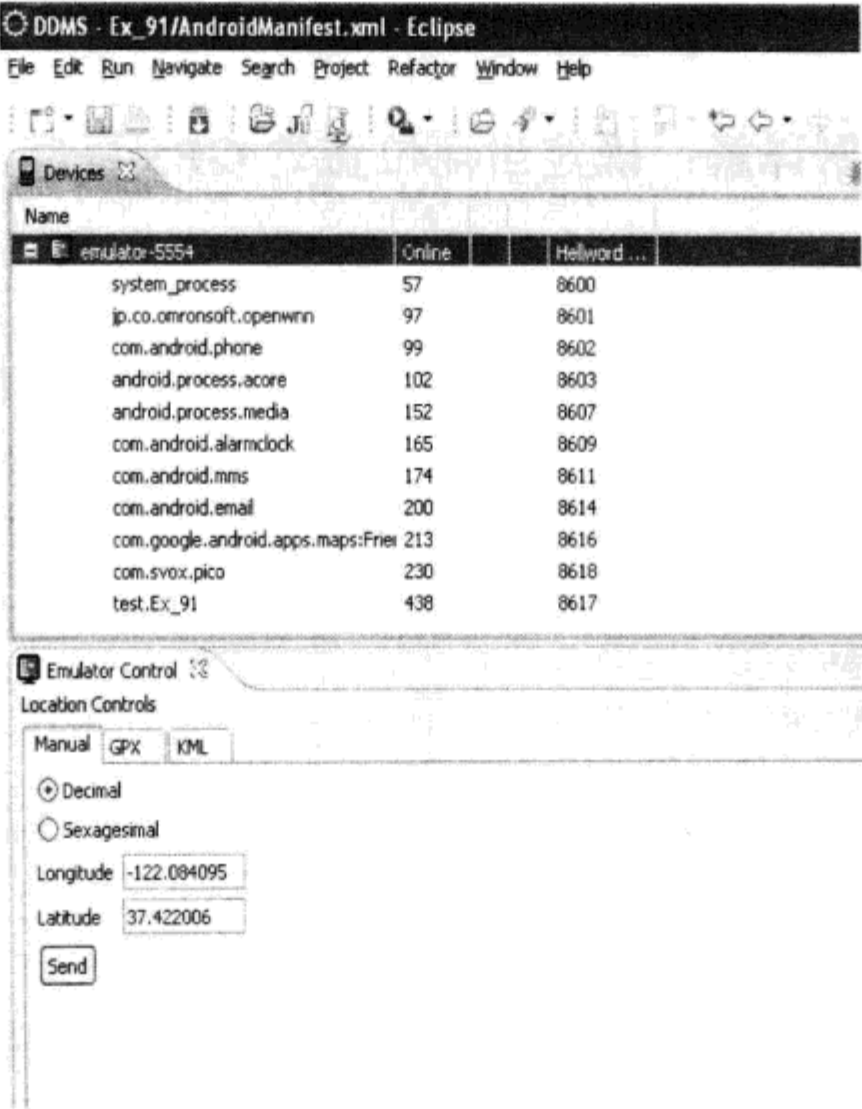


图 11-2 使用 DDMS 的 Emulator Control 设置经纬度

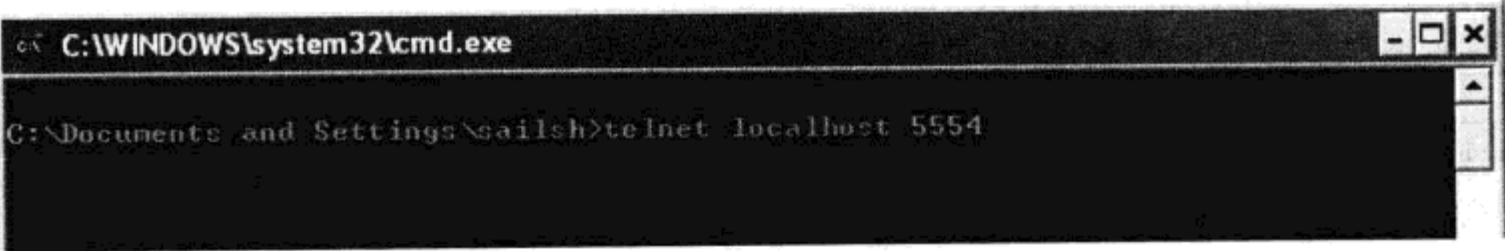


图 11-3 连接模拟器终端



图 11-4 模拟器端口号

接入模拟器后，进入 Android 的命令行界面，如图 11-5 所示。



图 11-5 命令行界面

然后使用“geo fix 经度 纬度”命令发送经纬度信息给 Android 模拟器。



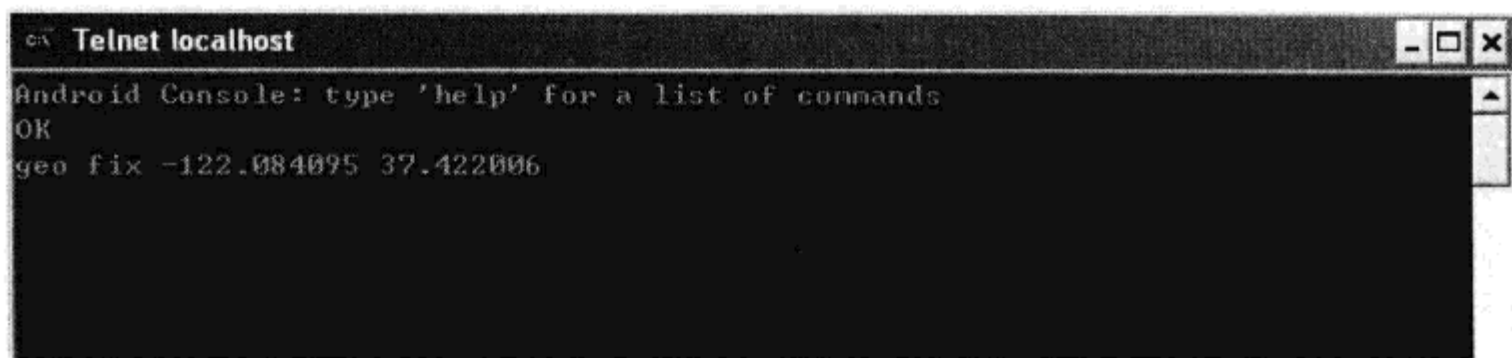


图 11-5 发送经纬度信息

输入“geo fix”命令之后，即发送该信息给 Android 模拟器。发送成功后，返回 OK 信息，如图 11-6 所示。

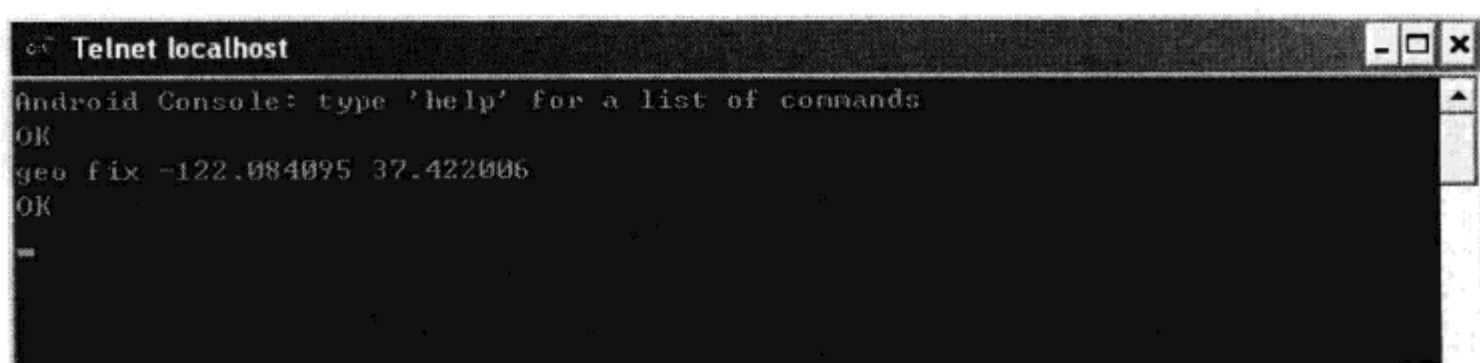


图 11-6 发送成功

## 11.4.2 实例实现

下面以一个“确定当前位置的 GPS 程序”为例讲述 GPS 的应用。

【实例 11-1】确定当前位置的 GPS 程序。

MyLocation.java 实现，该类实现了确定当前位置的 GPS 程序的功能。

```
package test.Ex_111;

import android.app.Activity;           //导入 Activity 类
import android.content.Context;        //导入上下文类
import android.location.Criteria;      //导入 Criteria 类
import android.location.Location;      //导入 Location 类
import android.location.LocationListener; //导入位置监听器类
import android.location.LocationManager; //导入位置管理器类
import android.os.Bundle;              //导入 Bundle 类
import android.util.Log;               //导入日志类
import android.widget.TextView;         //导入文本视图类
import android.widget.Toast;           //导入 Toast 类

public class MyLocation extends Activity
{
    /** Called when the activity is first created. */
    private LocationManager locationManager; //声明位置管理器对象
    private LocationListener locationlistener; //声明位置监听器对象
}
```



```
String locationprovider;           //声明字符串变量
private TextView textview;         //声明文本视图
public void onCreate(Bundle bundle)
{
    super.onCreate(bundle);         //必须执行父类的 onCreate 方法, 参数和子类方法的参数一致
    setContentView(R.layout.main); //使用 main.xml 初始化程序 UI
    textview = (TextView) findViewById(R.id.textview); //根据 XML 定义创建文本视图对象
    try
    {
        Criteria locationcriteria = new Criteria();           //新建 Criteria 类
        locationcriteria.setAccuracy(Criteria.ACCURACY_FINE); //设置精确度为精准模式
        locationcriteria.setAltitudeRequired(false);          //是否提供海拔高度信息
        locationcriteria.setBearingRequired(false);            //是否提供方向信息
        locationcriteria.setCostAllowed(true);                 //是否允许运营商计费
        locationcriteria.setPowerRequirement(Criteria.POWER_LOW); //设置电池消耗为低耗模式
        String context = Context.LOCATION_SERVICE;
        locationManager = (LocationManager) getSystemService(context);
        //使用 getSystemService 方法
        //获取位置管理器对象
        Toast.makeText(MyLocation.this, "getSystemService" , Toast.LENGTH_LONG).show();
        if (checkgps()) //检查 GPS 功能是否开启
        {
            locationManager.setTestProviderEnabled("gps", true); //激活 GPS
            locationprovider = locationManager.getBestProvider(locationcriteria, true);
            //设置位置提供商
            Log.d("provider", locationprovider);
            locationlistener = new MyLocationListener();           //注册位置监听器
        }
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(MyLocation.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
```



```

}
/*onCreate 方法是 Activity 的执行入口*/

private class MyLocationListener implements LocationListener
{
    public void onLocationChanged(Location location)
    {
        // TODO Auto-generated method stub
        Log.i("MyLocationListener onLocationChanged", "Invoke");
        if (location != null)
        {
            String display = "Current altitude = " + location.getAltitude() + "\nCurrent latitude = " +
                location.getLatitude();
            textView.setText(display);
        }
        //若位置有效, 则显示当前经纬度
        locationManager.removeUpdates(this);
        locationManager.setTestProviderEnabled(locationprovider, false);
    }
    //若位置发生变化, onLocationChanged 方法被调用
    @Override
    public void onProviderDisabled(String provider)
    {
        // TODO Auto-generated method stub
        Log.i("MyLocationListener onProviderDisabled", "Invoke");
    }
    //若屏蔽提供商, 该方法被调用
    @Override
    public void onProviderEnabled(String arg0) {
        // TODO Auto-generated method stub
        Log.i("MyLocationListener onProviderEnabled", "Invoke");
    }
    //若激活提供商, 该方法被调用
    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
        // TODO Auto-generated method stub
        Log.i("MyLocationListener onStatusChanged", "Invoke");
    }
    //若状态发生变化, 该方法被调用

```



```

}
/*MyLocationListener 为地理位置监听器, 需要实现 onStatusChanged、onProviderEnabled、onProviderDisabled 和
 * onProviderDisabled 等方法*/

private boolean checkgps()
{
    boolean                providerEnabled
locationManager.isProviderEnabled(android.location.LocationManager.GPS_PROVIDER);
    //判断 Provider 是否被激活
    if (providerEnabled == true )
    {
        Toast.makeText(this, "GPS 模块正常", Toast.LENGTH_SHORT).show();
        return true;
    }
    //若被激活, 返回真值
    else
    {
        Toast.makeText(this, "请开启 GPS!", Toast.LENGTH_SHORT).show();
        return false;
    }
    //若未被激活, 则返回假值
}
}

```

【代码说明】本实例通过 android.location 开发包实现了地理位置的获取。注意, 需要在清单文件中添加权限 ACCESS\_FINE\_LOCATION、ACCESS\_COARSE\_LOCATION 和 UPDATE\_DEVICE\_STATS 才能实现 GPS 的应用:

```

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION">
</uses-permission>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION">
</uses-permission>
<uses-permission android:name="android.permission.UPDATE_DEVICE_STATS">
</uses-permission>
<application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".MyLocation"
        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>

通过 Eclipse 启动该 Android 程序，程序获取到了当前的经纬度并在 TextView 中显示出来，如图 11-7 所示。



图 11-7 程序获取经纬度

11.5 基于 Google Map 的应用

11.5.1 使用 MapView 显示地图

MapView 类属于 com.google.android.maps 包，是用来显示地图的视图。MapView 提供了地图的不同显示模式：

(1) 交通模式

使用 setTraffic(boolean mode)方法设置交通模式。若参数为 true，则当前地图为交通模式，否则地图不是交通模式。图 11-8 显示了交通模式的例子。

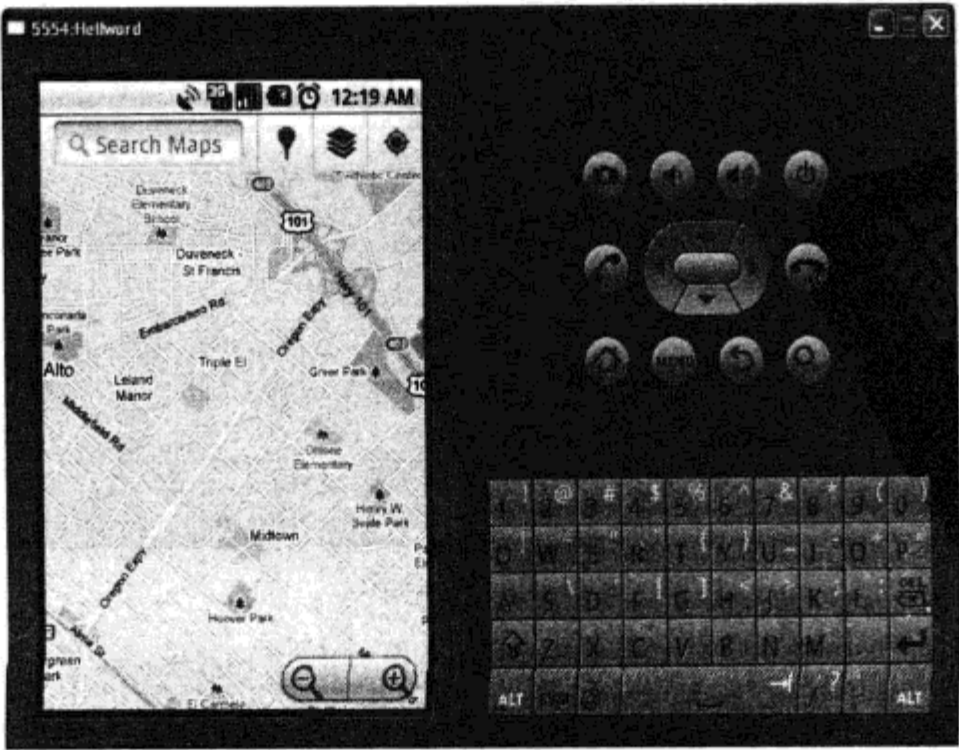


图 11-8 交通模式



(2) 街道模式

使用 `setStreetView(boolean mode)`方法设置街道模式。若参数为 `true`，则当前地图为街道模式，否则地图不是街道模式。图 11-9 显示了街道模式的例子。

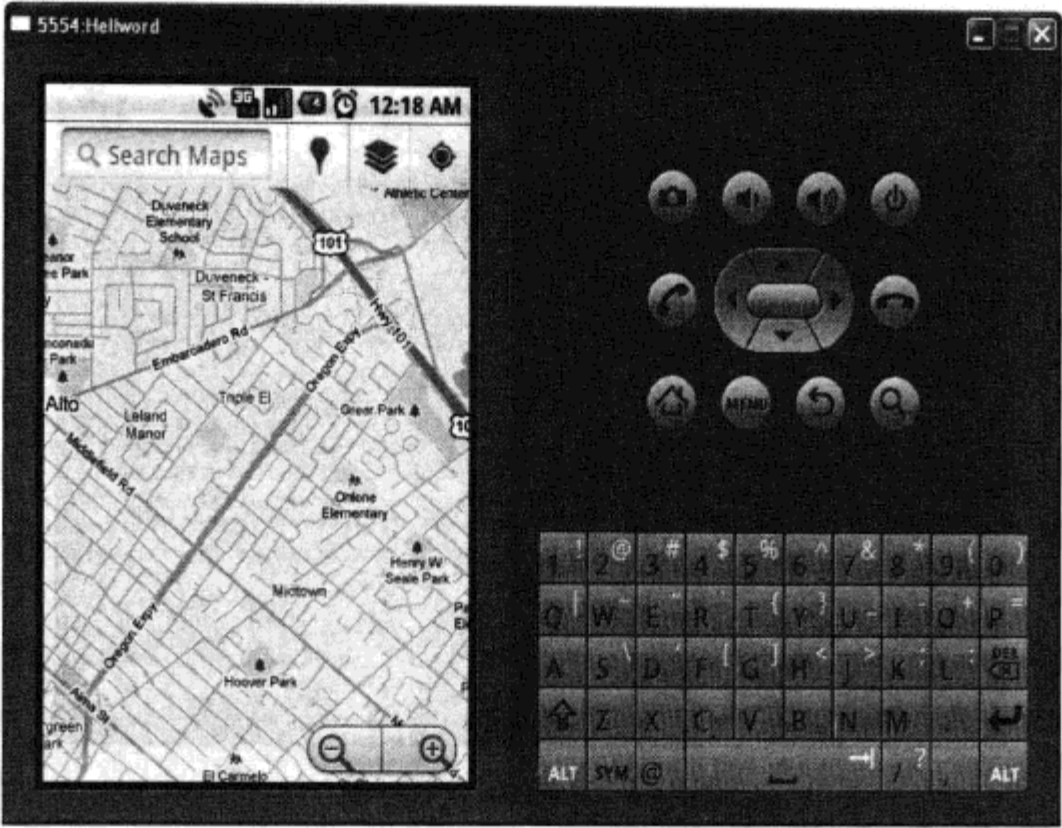


图 11-9 街道模式

(3) 卫星模式

使用 `setSatellite((boolean mode)`方法设置卫星模式。若参数为 `true`，则当前地图为卫星模式，否则地图不是卫星模式。图 11-10 显示了卫星模式的例子。



图 11-10 卫星模式

MapView 只能由 MapActivity 创建,这是因为 MapView 需要通过后台的线程来连接网络或者文件系统。MapActivity 是一个抽象类,任何想要显示 MapView 的 Activity 都需要派生自 MapActivity。使用 MapView 时需要申请 APIKey,下面讲述申请 APIKey 的步骤。



(1) 打开 Eclipse, 单击 Window→Preferences→Android→Build, 查看 Default debug keystore 位置, 作者的是 C:\Documents and Settings\sailsh\.android\debug.keystore, 如图 11-11 所示。

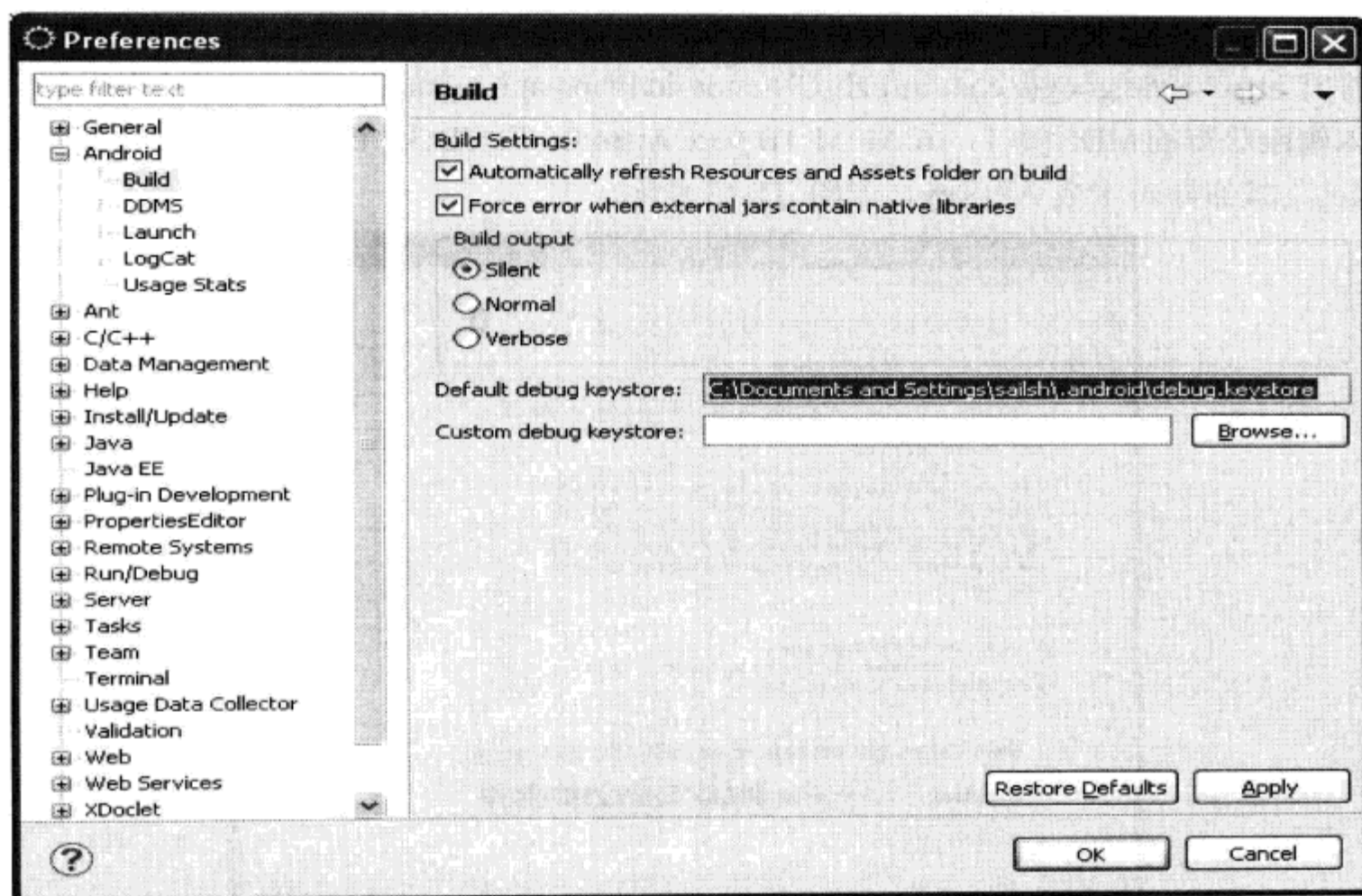


图 11-11 查看 keystore 位置

(2) 获取指纹认证。

查看 keystore 位置之后, 使用 keytool 命令获取其对应的 MD5 值: `keytool -list -alias androiddebugkey -keystore "C:\Documents and Settings\sailsh\.android\debug.keystore" -storepass android -keypass android`, 如图 11-12 所示。



图 11-12 获取 MD5 值



图 11-12 显示 keytool 命令返回一个认证指纹的 MD5 值（17:16:AF:51:1B:93:CA:B4:C7:71:7D:30:42:94:81:A1），通过该值可获取 APIKey。

### （3）生成 APIKey。

打开 <http://code.google.com/intl/zh-CN/android/maps-api-signup.html>，在 MD5 对应的文本框中填入刚刚获得的 MD5 值（17:16:AF:51:1B:93:CA:B4:C7:71:7D:30:42:94:81:A1）。单击“Generate API Key”按钮即可生成 APIKey，如图 11-13 所示。

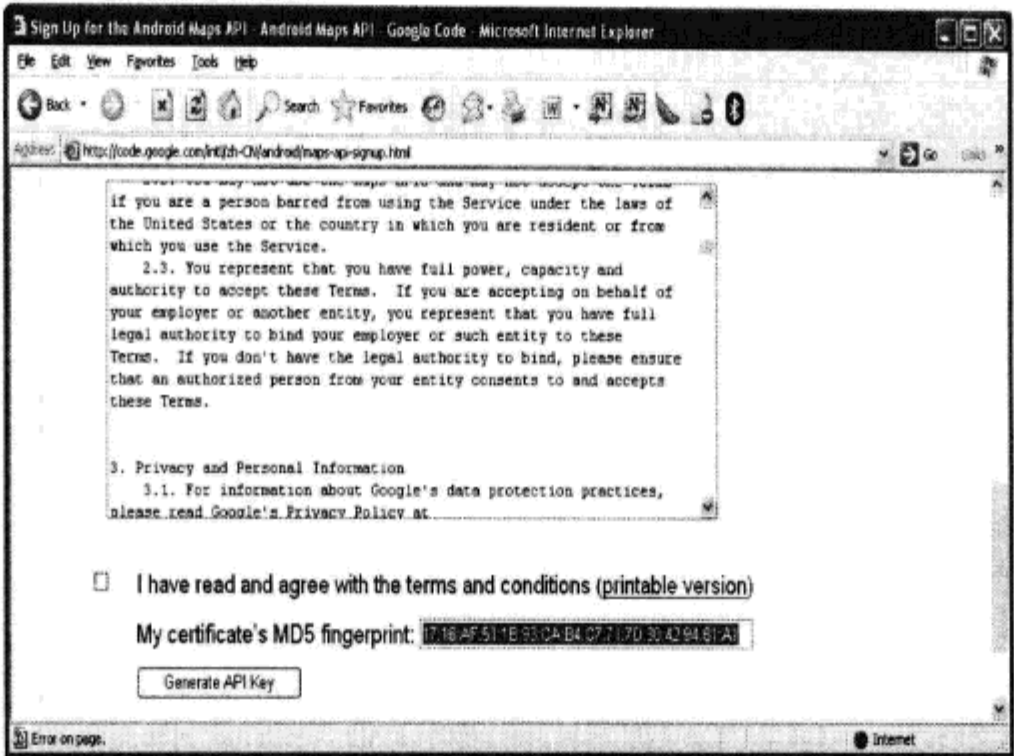


图 11-13 生成 APIKey

## 11.5.2 使用 MapController 控制地图缩放

电子地图常用的操作就是查找、放大和缩小等。图 11-14 显示了放大地图的例子（右图为放大图片，左图为原图）。



图 11-14 放大地图



图 11-15 显示了缩小地图的例子（右图为缩小图片，左图为原图）。

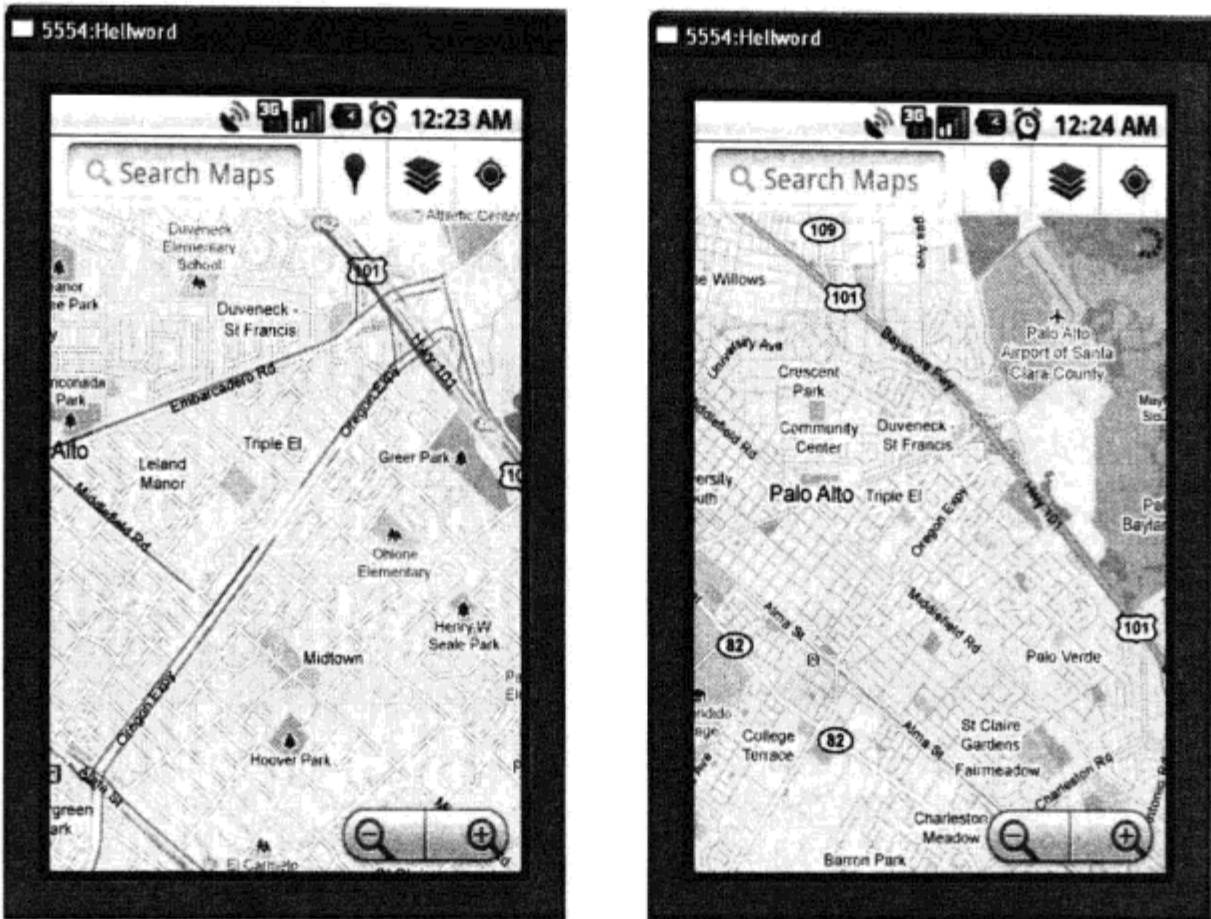


图 11-15 缩小地图

Android 提供了 MapController 类来控制地图的处理，如转换地点、放大和缩小等。使用 animateTo 方法可将地图转移到参数指定的位置，使用 setZoom 方法可以按照参数指定的比例缩放地图。表 11-4 列举了 MapController 常用的方法。

表 11-4 MapController 的常用方法

方法	功能描述	返回值
animateTo(GeoPoint point, java.lang.Runnable runnable)	根据给定的 GeoPoint，开始显示地图	void
scrollBy(int x, int y)	按照给定的像素滚动。其中，参数 x 指定水平方向移动的像素，y 指定垂直方向移动的像素	void
onKey(android.view.View v, int keyCode, android.view.KeyEvent event)	处理按键事件，把事件变换为适度的地图平移	boolean
setZoom(int zoomLevel)	设置地图的缩放级别，范围是[1,21]	int
zoomOutFixing(intx, int y)	缩小一个级别。这个缩放操作将把地图平移到屏幕的一个固定点上，通过像素坐标来设定固定点	boolean
zoomIn()	放大一个级别	boolean

11.6 应用实例详解：普通地图和卫星地图切换

11.6.1 实例分析

卫星地图是从卫星的角度观察的地图，普通地图则一般是街道地图或者交通地图。通过 MapView 类的 setSatellite((boolean mode)方法可以设置当前的地图是否为卫星视图，而使用



setTraffic(boolean mode)方法可以设置是否为交通模式。

11.6.2 实例实现

下面以一个“普通地图和卫星地图切换”为例讲述 MapView 的应用。

**【实例 11-2】**普通地图和卫星地图切换。

Switch.java 实现，该类实现了普通地图和卫星地图的切换功能。因此，需要在相应的按钮监听器中设置卫星地图和普通地图的转换：

```

Traffic_map_button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        setTraffic(true); //设置交通模式
    }
});
Street_map_button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        setTraffic(true); //设置街道模式
    }
});
Satellite_map_button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        setTraffic(true); //设置普通模式
    }
});
    
```

**【代码说明】**实例 11-2 通过 Map 包实现了普通地图和卫星地图的切换。  
 通过 Eclipse 启动该 Android 程序，程序显示一个普通地图，如图 11-16 所示。



图 11-16 普通地图



可为地图选择不同的模式，图 11-17 显示了选择模式的界面。  
 图 11-18 为用户选择街道模式时地图的显示效果。

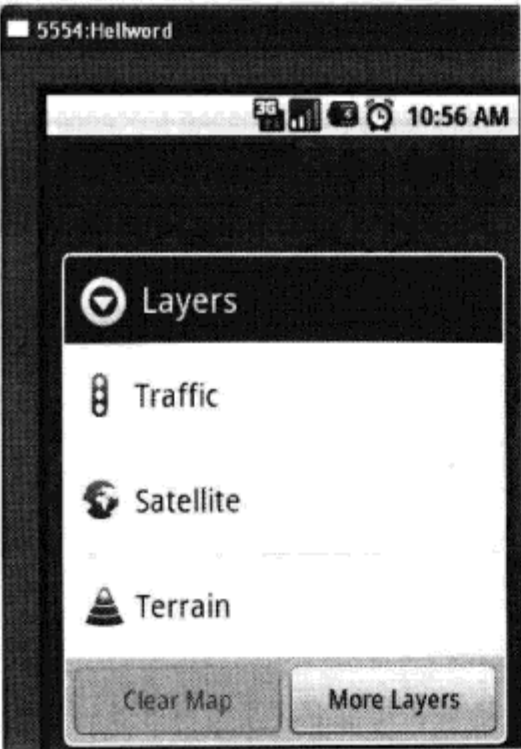


图 11-17 选择模式



图 11-18 选择街道模式

图 11-19 为地图放大时的效果。

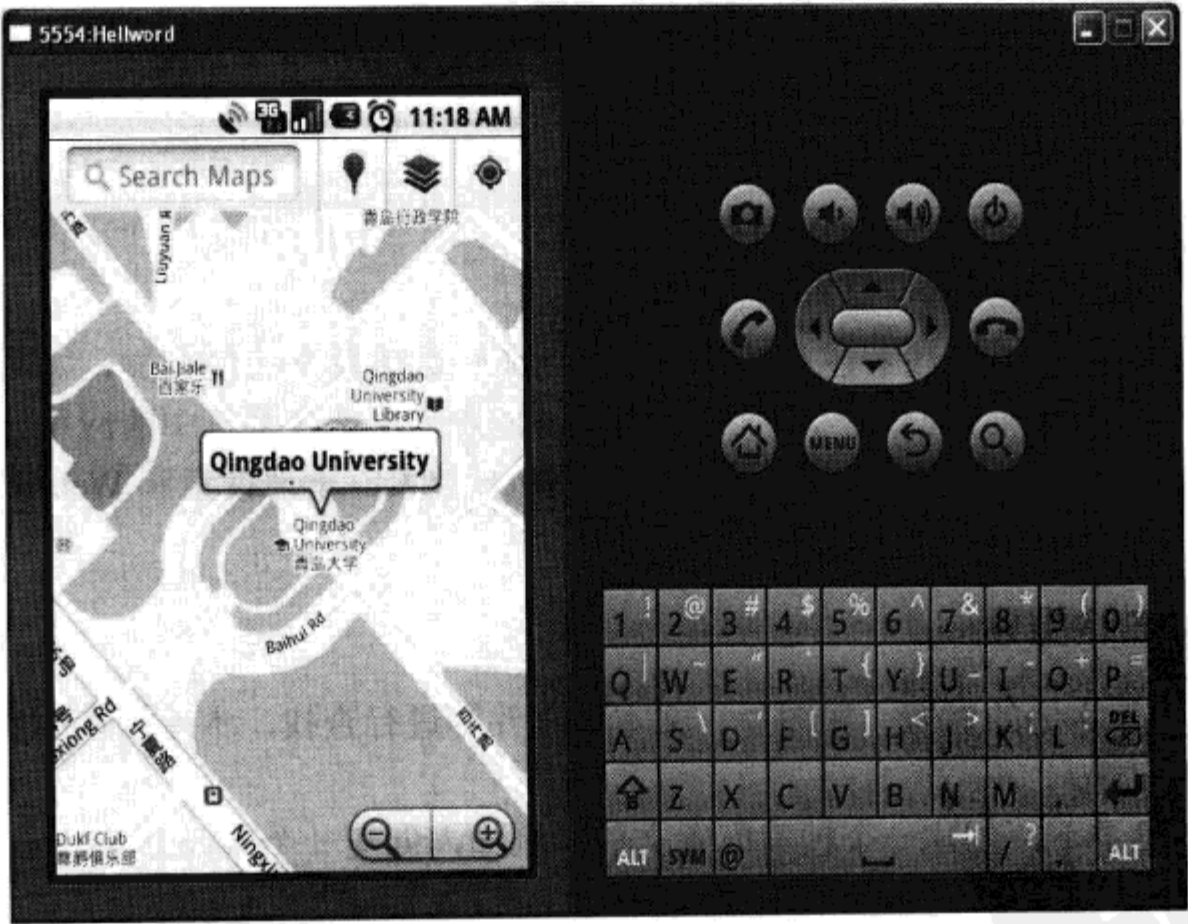


图 11-19 放大地图



## 第 12 章 Android 的搜索引擎和 Gtalk 开发

搜索引擎是一种查找信息的方式，使用搜索引擎，用户可快速检索到感兴趣的信息。Android 手机支持搜索引擎的开发，开放了搜索相关的 API。图 12-1 所示是利用手机搜索单词的例子。



图 12-1 利用手机搜索单词

### 12.1 搜索引擎在手机中的应用

手机作为智能终端，承载着越来越多的业务。随着数据爆发式地增加，如何使用手机进行信息过滤是一个非常重要的问题，这一节我们将讲述搜索引擎在手机中的应用。按照搜索类型的不同，Android 的搜索服务可分为本地搜索和 Web 搜索，本地搜索和 Web 搜索使用不同的 Intent。

#### 12.1.1 本地搜索

本地搜索基于本地的资源，如图片、文本、文件等进行查找，本地搜索的步骤为：

(1) 搜索配置。

在 res/xml 目录下，创建一个搜索配置文件。例如，创建一个 searchable.xml 文件：

```

<searchable xmlns:android="http://schemas.android.com/apk/res/android"
<!-- android:label 指定显示的文本，hint 指定文本框的提示文本-->
    android:label="Search" android:hint="Search Application" android:searchMode="showSearchLabelAsBadge"
<!-- 语音搜索配置 -->
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer"
    android:voiceLanguageModel="free_form" android:voicePromptText="@string/search_invoke"

```



```
<!-- 配置搜索建议 -->
android:searchSuggestAuthority="com.android.cbin.SearchSuggestionSampleProvider"
android:searchSuggestSelection=" ? " />
```

搜索框配置文件是一个用来配置应用程序中搜索框设置的 XML 文件, 这个文件一般命名为 searchable.xml, 并且必须保存在项目的 res/xml/ 目录下。

然后在清单文件中创建本地搜索的 Intent 过滤器:

```
<intent-filter>
    <action android:name="android.intent.action.SEARCH" />
</intent-filter>
```

在清单文件中添加搜索配置的元数据:

```
<meta-data android:name="android.app.searchable"
    android:resource="@xml/searchable" />
</activity>
```

在清单文件中添加默认搜索配置:

```
<meta-data
    android:name="android.app.default_searchable"
    android:value=".SearchApplication" >
</meta-data>
```

- (2) 生成本地搜索的 Intent。
- (3) 启动新任务标志。
- (4) 生成搜索信息。
- (5) 启动 Web 搜索。

下面是一个本地搜索的例子:

```
String key = editText.getText();
Intent search = new Intent(Intent.ACTION_SEARCH); //生成本地搜索的 Intent
search.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK); //启动新任务标志
search.putExtra(SearchManager.QUERY, key); //放入所要搜索的文字
search.putExtra(SearchManager.APP_DATA, getIntent().getBundleExtra(SearchManager.APP_DATA));
startActivity(search); //启动本地搜索
```

图 12-2 显示了一个本地搜索的例子。在输入框中输入关键字, Android 会根据关键字查找相应的应用。

### 12.1.2 Web 搜索

Web 搜索是 Google 擅长的搜索方式, Google 包含强大的搜索引擎来支持用户的 Web 信息查找。Web 搜索和本地搜索的步骤类似:

(1) 搜索配置。在 res/xml 目录下, 创建一个搜索配置文件。例如创建一个 searchable.xml 文件:

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    <!-- android:label 指定显示的文本, hint 指定文本框的提示文本 -->
    android:label="Search" android:hint="Search Application" android:searchMode="showSearchLabelAsBadge"
```



```

<!-- 语音搜索配置 -->
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer"
android:voiceLanguageModel="free_form" android:voicePromptText="@string/search_invoke"
<!-- 配置搜索建议 -->
    android:searchSuggestAuthority="com.android.cbin.SearchSuggestionSampleProvider"
android:searchSuggestSelection=" ? " />
    
```



图 12-2 本地搜索

搜索框配置文件是一个用来配置应用程序中搜索框设置的 XML 文件，这个文件一般命名为 searchable.xml，并且必须保存在项目的 res/xml/目录下。

然后在清单文件中创建本地搜索的 Intent 过滤器：

```

<intent-filter>
    <action android:name="android.intent.action.WEB_SEARCH" />
</intent-filter>
    
```

在清单文件中添加搜索配置的元数据：

```

<meta-data android:name="android.app.searchable"
    android:resource="@xml/searchable" />
</activity>
    
```

在清单文件中添加默认搜索配置：

```

<meta-data
    android:name="android.app.default_searchable"
    android:value=".SearchApplication" >
</meta-data>
    
```

- (2) 生成 Web 搜索的 Intent。
- (3) 启动新任务标志。
- (4) 生成搜索信息。



(5) 启动 Web 搜索。

下面是一个 Web 搜索的例子：

```

String key = editText.getText();
Intent search = new Intent(Intent.ACTION_WEB_SEARCH); //生成 Web 搜索的 Intent
search.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK); //启动新任务标志
search.putExtra(SearchManager.QUERY, key ); //放入所要搜索的文字
search.putExtra(SearchManager.APP_DATA, getIntent().getBundleExtra(SearchManager.APP_DATA));
startActivity(search); //启动 Web 搜索
    
```

图 12-3 显示了一个 Web 搜索的例子。在 Web 输入框中输入关键字，Android 会根据关键字查找相匹配的网页。

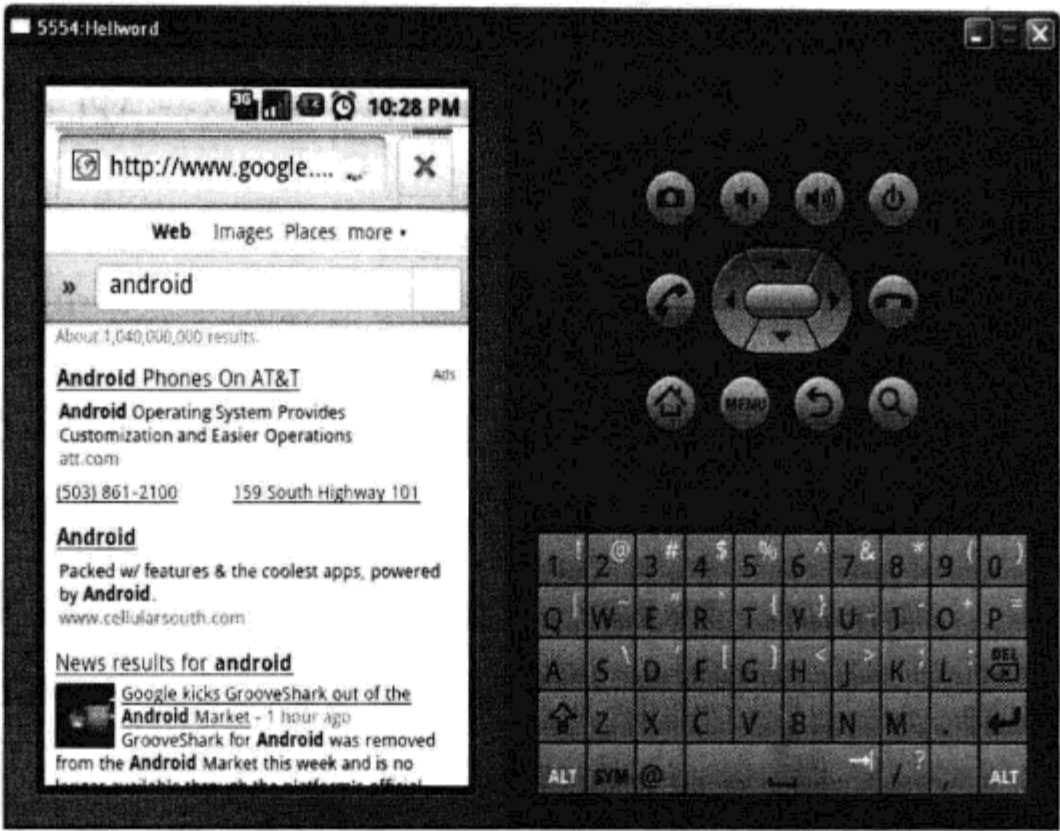


图 12-3 Web 搜索

注意，上面使用了 addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK)方法启动任务，addFlags 指定了任务的启动方式，表 12-1 列举了常用的 flag。

表 12-1 常用的 flag

名称	功能描述	整型值
FLAG_ACTIVITY_BROUGHT_TO_FRONT	正常情况下，该标志不应该由应用程序设置，而是由单任务启动模式的系统设置	4194304 (0x00400000)
FLAG_ACTIVITY_CLEAR_TOP	如果设置该标志，若某个 Activity 已经在当前的任务中运行，不会创建该 Activity 的实例，并且所有其他的 Activity 都会被关闭。这个 Intent 会作为一个新的 Intent 传给已经运行的 Activity	67108864 (0x04000000)
FLAG_ACTIVITY_EXCLUDE_FROM_RECENTS	如果设置该标志，任务不会保存在最近启动活动的列表里	8388608 (0x00800000)
FLAG_ACTIVITY_FORWARD_RESULT	如果设置该标志，已存在的任务启动一个新的任务，当前任务的返回值会给新的任务。这种方式能够获取之前任务的返回值	33554432 (0x02000000)
FLAG_ACTIVITY_LAUNCHED_FROM_HISTORY	正常情况下，该标志不应该由应用程序设置，而是由系统设置	1048576 (0x00100000)



## 12.2 Android 搜索引擎 API 简介

搜索引擎是 Android 的核心应用，用户通过搜索引擎能够搜索可用的数据，这些数据可能位于本地设备上，也可位于互联网上。搜索服务在整个系统中应该是无缝和一致的，Android 为用户提供了一个搜索框架，开发人员可根据搜索服务为用户提供一个熟悉的搜索对话框和友好界面的搜索体验。Android 的搜索框架提供了一个用户界面，用户可以使用该框架执行查询和交互，所以不必去建立自己的搜索活动。相反，一个搜索对话框出现在界面的上方，不会影响当前的活动。搜索框架管理搜索对话框，当用户执行一项搜索时，搜索框架将查询文本传给实施查询的应用。

Android 提供了 `android.app.SearchManager` 提供搜索功能，该类输入 `java.lang.Object` 包，继承了 `DialogInterface.OnCancelListener` 和 `DialogInterface.OnDismissListener` 接口。`android.app.SearchManager` 向终端用户开放了 `SearchManager` 的 API，使用这些 API 可定制相应的搜索功能。表 12-2 列举了 `SearchManager` 的常用方法

表 12-2 SearchManager 常用的方法

方法	功能描述	返回值
<code>getSearchableInfo(Component Name c)</code>	返回搜索活动的信息	<code>SearchableInfo</code>
<code>getSearchablesInGlobalSearch()</code>	返回全局搜索的 <code>Searchable</code> 活动的列表	<code>List&lt;SearchableInfo&gt;</code>
<code>onCancel(DialogInterface dialog)</code>	过时的方法，不赞成使用	<code>void</code>
<code>onDismiss(DialogInterface dialog)</code>	过时的方法，不赞成使用	<code>void</code>
<code>setOnCancelListener(SearchManager.OnCancelListener listener)</code>	该方法用来设置取消搜索 UI 的监听器，其中参数 <code>listener</code> 用于指定监听器	<code>void</code>
<code>startSearch(String i, boolean s, ComponentName l, Bundle a, boolean g)</code>	启动搜索	<code>void</code>
<code>stopSearch()</code>		<code>void</code>
<code>triggerSearch(String q, ComponentName l, Bundle a)</code>	激活搜索	<code>void</code>

## 12.3 应用实例详解：过滤式搜索引擎程序

### 12.3.1 实例分析

上面讲述了搜索的步骤以及 API，下面以一个搜索引擎的例子讲解 Android 搜索的实现过程。

首先为了实现浮动搜索框的功能，使用 `setDefaultKeyMode` 方法设置按键启动搜索框的功能。`setDefaultKeyMode(int mode)` 可用来设置 Activity 按键所启动的应用。`mode` 是由 Activity 类定义的常量，表 12-3 列举了常用的模式。



表 12-3 常用的模式

常量	功能描述	值
DEFAULT_KEYS_DISABLE	使用 setDefaultKeyMode (DEFAULT_KEYS_DISABLE) 设置键盘屏蔽模式，Activity 对按键不做任何响应	0 (0x00000000)
DEFAULT_KEYS_DIALER	使用 setDefaultKeyMode (DEFAULT_KEYS_DIALER) 设置键盘拨号模式，Activity 监听按键后触发拨号程序	1 (0x00000001)
DEFAULT_KEYS_SEARCH_GLOBAL	使用 setDefaultKeyMode (DEFAULT_KEYS_SEARCH_GLOBAL) 设置键盘全局搜索模式，Activity 监听按键后触发全局搜索	4 (0x00000004)
DEFAULT_KEYS_SEARCH_LOCAL	使用 setDefaultKeyMode (DEFAULT_KEYS_SEARCH_LOCAL) 设置键盘本地搜索模式，Activity 监听按键后触发本地搜索	3 (0x00000003)
DEFAULT_KEYS_SHORTCUT	使用 setDefaultKeyMode (DEFAULT_KEYS_SHORTCUT ) 设置键盘快捷键模式，Activity 监听按键后触发快捷键	2 (0x00000002)

12.3.2 实例实现

下面以一个“过滤式搜索引擎”为例讲述搜索引擎的开发流程。

【实例 12-1】过滤式搜索引擎。

SearchApplication.java 实现，该类实现了确定当前位置的 Web 搜索引擎的功能：

```

package test.Ex_121;           //程序打包

import android.app.Activity;   //导入 Activity 类
import android.app.SearchManager; //导入 SearchManager 类，使用该类实现搜索服务
import android.content.Intent; //导入 Intent 类
import android.os.Bundle;      //导入 Bundle 类
import android.view.Menu;       //导入菜单类
import android.view.MenuItem;   //导入菜单元素类
import android.widget.Toast;     //导入 Toast 消息类

public class SearchApplication extends Activity {
    private final int MENU = 1;           //定义私有整型变量，该变量值在运行后不能修改
    private Intent intent;                 //声明 Intent 对象
    private final int KeyMode= Activity.DEFAULT_KEYS_SEARCH_LOCAL; //定义键盘模式为按键搜索模式
    private final int icon = android.R.drawable.ic_menu_search;    //使用 Android 自定义的图标

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); //必须执行父类的 onCreate 方法
        try{
            setContentView(R.layout.main); //使用 main 加载布局
            setDefaultKeyMode(KeyMode); //设置键盘模式，按键弹出搜索框
            intent = getIntent(); //获取 Intent 对象
            searchWeb(intent);
        }
    }
}
    
```



```

/*try 块包含可能出现异常的代码*/
catch (Exception e)
{
    Toast.makeText(SearchApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
}
/*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
finally
{
    //TODO
}
/*finally 中可添加处理异常的代码*/
}

/*onCreate 为 Activity 的执行入口*/

private void searchWeb(Intent intent) {
    String action = intent.getAction(); //获取意图的行为
    if (Intent.ACTION_WEB_SEARCH.equals(action))
    {
        searchHandler(intent); //调用 searchHandler
    }
    else
    {
        //TODO
    }
    //若意图的行为为 Web 搜索, 则搜索
}

/*searchWeb 根据意图的行为判断是否是 Web 搜索*/

@Override
protected void onNewIntent(Intent intent) {
    setIntent(intent); //设置 Intent
    searchWeb(intent);
}

/*重写父类的方法*/

private void searchHandler(Intent intent) {
    final String query = intent.getStringExtra(SearchManager.QUERY);
    Toast.makeText(this, query, Toast.LENGTH_LONG).show();
}

/*searchHandler 获取 SearchManager 的查询*/

```



```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    boolean result = super.onCreateOptionsMenu(menu);
    MenuItem menuItem = menu.add(0, MENU, 0, getText(R.string.searchMenu)); //添加菜单选项
    menuItem.setIcon(icon); //设置图标
    return result; //返回结果
}

/*onMenuItemSelected 实现*/

@Override
public boolean onOptionsItemSelected(int featureId, MenuItem item) {
    switch (item.getItemId()) {
        case MENU:
            onSearchRequested();
            return true;
        default:
            return super.onOptionsItemSelected(featureId, item);
    }
    //若选中元素 1, 则返回真值, 否则返回假值
    //return super.onOptionsItemSelected(featureId, item);
}

/*onMenuItemSelected 实现*/
}

```

在清单文件中添加 android.intent.action.WEB\_SEARCH 的意图过滤器:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="test.Ex_121" android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SearchApplication" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.WEB_SEARCH" />
            </intent-filter>
            <meta-data android:name="android.app.searchable"
                android:resource="@xml/searchable" />
        </activity>
    </application>
</manifest>

```



```
</application>
<meta-data
    android:name="android.app.default_searchable"
    android:value=".SearchApplication" />
</manifest>
```

res/xml 下的 searchable.xml 实现:

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="搜索"
    android:hint="输入搜索条件"
    android:voiceSearchMode="showVoiceSearchButton|launchRecognizer"/>
```

【代码说明】本实例通过 SearchManager 来实现搜索功能, SearchApplication 类定义了如下方法。

#### (1) onCreate 方法

该方法必须执行父类的 onCreate 方法, 然后使用 setContentView 加载布局, 使用 setDefaultKeyMode 方法设置键盘搜索模式, 当用户按键时弹出搜索框。

#### (2) private void searchWeb(Intent intent)方法

该方法根据意图的行为判断是否 Web 搜索, 若调用意图的行为与 Intent.ACTION\_WEB\_SEARCH 相同, 则调用 searchHandler。

#### (3) protected void onNewIntent(Intent intent)方法

该方法重写父类的方法, 该方法会调用 searchWeb 方法。

#### (4) void searchHandler(Intent intent) 方法

该方法获取 SearchManager 的查询:

```
final String query = intent.getStringExtra(SearchManager.QUERY);
Toast.makeText(this, query, Toast.LENGTH_LONG).show();
```

#### (5) public boolean onCreateOptionsMenu(Menu menu)方法

该方法重写父类的 onCreateOptionsMenu 方法, 实现菜单选项创建监听器:

```
boolean result = super.onCreateOptionsMenu(menu);
MenuItem menuItem = menu.add(0, MENU, 0, getText(R.string.searchMenu)); //添加菜单选项
menuItem.setIcon(icon); //设置图标
```

#### (6) public boolean onOptionsItemSelected(int featureId, MenuItem item)方法

该方法重写父类的 onOptionsItemSelected 方法, 实现菜单元素选择监听器:

```
switch (item.getItemId()) {
    case MENU:
        onSearchRequested();
        return true;
    default:
        return super.onOptionsItemSelected(featureId, item);
}
```

//若选中元素 1, 则返回真值, 否则返回假值



另外，程序还需要实现 res/xml 下的 searchable.xml，并且在清单中添加相应的过滤器。添加 Web 搜索的过滤器：

```

<intent-filter>
    <action android:name="android.intent.action.WEB_SEARCH" />
</intent-filter>

```

通过 Eclipse 启动该 Android 程序，程序启动界面不包含搜索框，如图 12-4 所示。



图 12-4 程序启动界面

这是因为使用 setDefaultKeyMode(DEFAULT\_KEYS\_SEARCH\_LOCAL)设置键盘本地搜索模式，Activity 监听按键后触发本地搜索。程序刚启动用户没有输入任何按键，本地搜索不会显示出来。程序启动后，按下键盘上的任意键，可以看到一个浮动的本地搜索框，如图 12-5 所示。

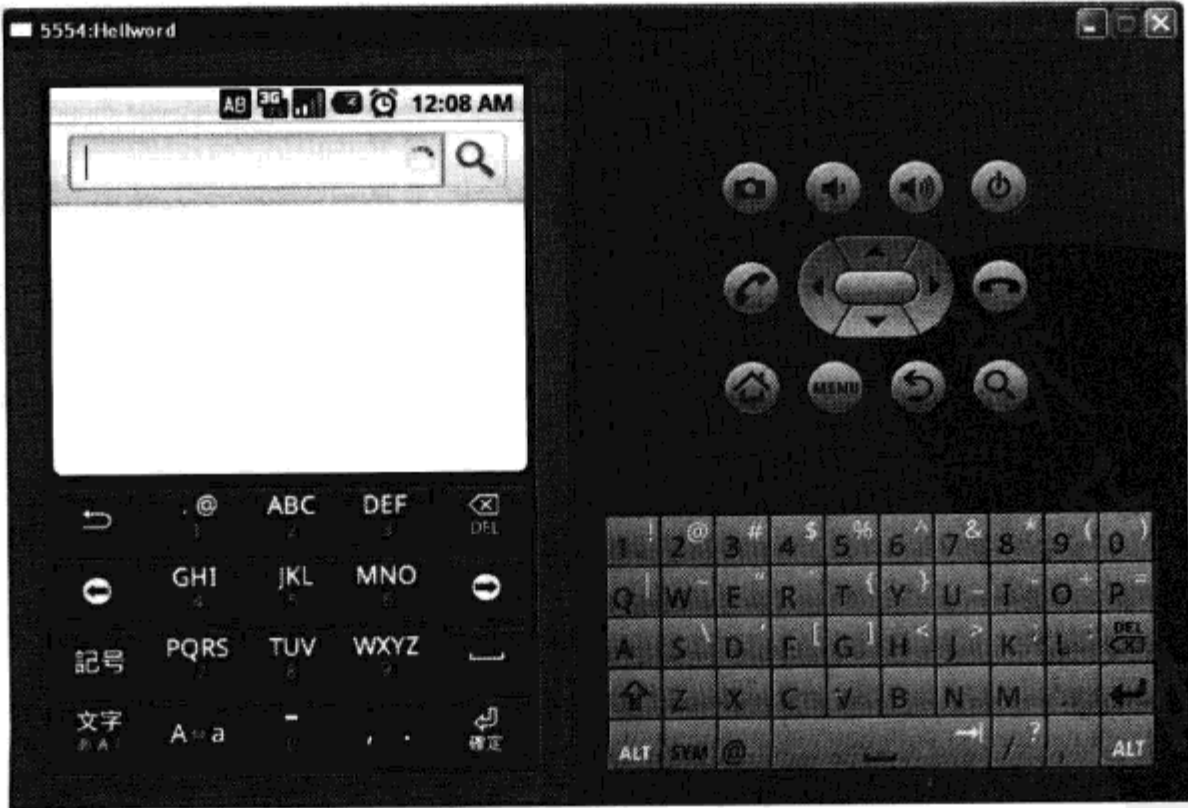


图 12-5 启动搜索框



输入“android os”关键字搜索相应的资源，如图 12-6 所示。

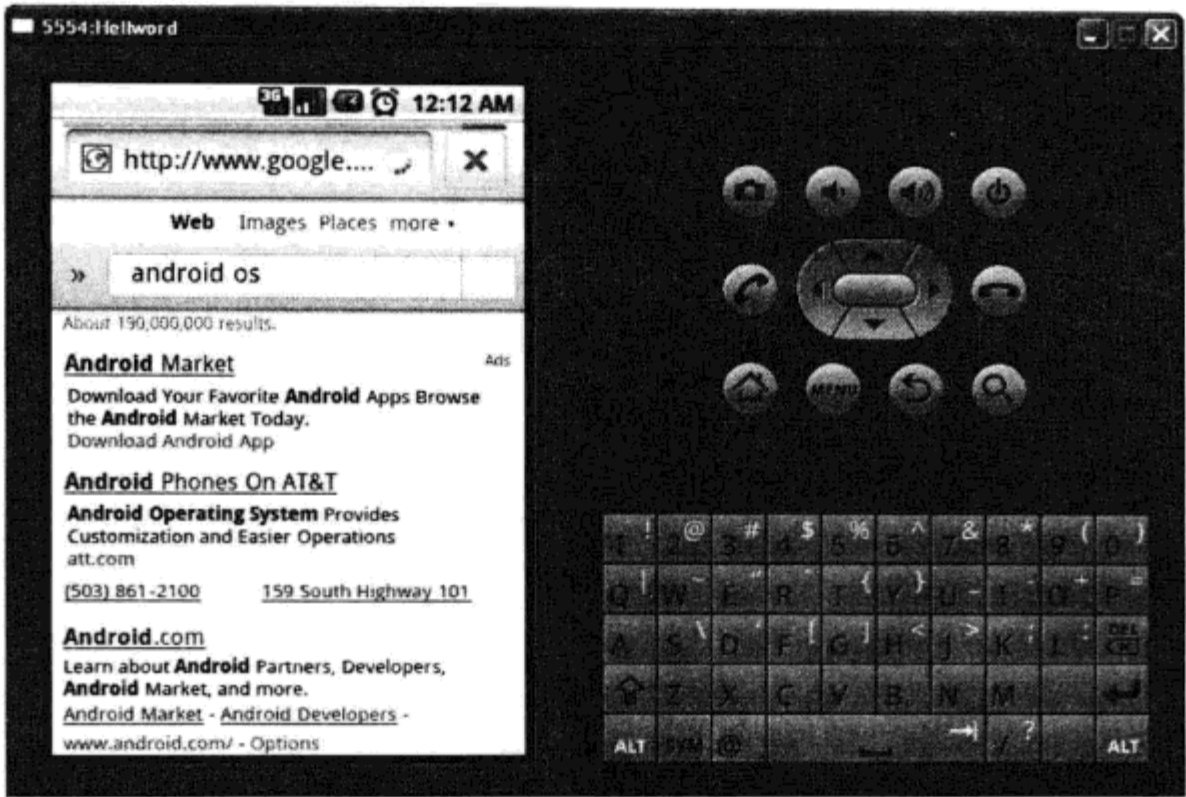


图 12-6 搜索指定的关键字

### 12.4 Android 的 GTalk 应用开发

Google 是一家提供信息搜索服务与通信的公司，它使得全球的信息能够被普通用户使用。Google 比较常用的业务是使用搜索引擎从网络上检索与查看信息，除此之外 Google 还致力于终端应用的开发，如 Gmail、Android 系统、浏览器、Google 地图和即时通信软件等。Gtalk 是 Google 推出的即时聊天工具的客户端，用户通过该客户端可同其他用户相互通信。Google Talk（又称 GTalk）使得用户能够与朋友、家人、同事通过语音电话和即时信息进行通信，这也提供了通信无处不在的功能。Google 致力于开放通信标准，这使得通信服务独立于客户端、运营商和平台，即 Google 用户和其他运营商可根据需要自由选择客户端、运营商和平台进行通信。图 12-7 显示了一个 Google Talk 的客户端。



图 12-7 GTalk 的客户端





### 12.4.1 GTalk 在手机中的应用

除了 GTalk 客户端之外,还有许多其他即时通信的客户端,如腾讯 QQ、微软 MSN、中国移动的飞信、电信 Live、Skype 等。如何使得这些客户端的用户之间能够进行无缝通信,是需要解决的问题。由于不同客户端采用不同的通信协议,可通过两种方式解决异网互通的问题:

- 第一种方式是不同客户端使用相同的通信标准。
- 第二种方式是在不同运营商之间设置通信网关,这种网关主要负责通信协议的转换。

第二种方式虽然允许客户端使用不同的通信标准,但成本高且不利于标准化。采用统一的通信标准是可行的,这样不同运营商之间的即时通信能够无缝进行。为了推动标准化,GTalk 采用标准的 XMPP (Extensible Messaging and Presence Protocol) 协议作为通信标准来实现验证 (Authentication)、呈现 (Presence) 和通信 (Messaging)。

XMPP 是基于 XML 的可扩展通信和表示协议,这种协议可用于实现验证 (Authentication)、呈现 (Presence) 和通信 (Messaging)。为了实现即时消息和呈现,Internet 工程任务组 (Internet Engineering Task Force, IETF) 基于 XML 流协议和 Jabber 开放协议对 XMPP 进行了标准化。

XMPP 具有以下特点:

#### (1) 可扩展性

由于基于 XML 协议, XMPP 协议具有较强的可扩展性以及开放性。经过扩展以后的 XMPP 可以通过发送扩展的信息来处理用户的需求,以及在 XMPP 的顶端建立如内容发布系统和基于地址的服务等应用程序。XMPP 的核心部分就是一个在网络上分片段发送 XML 的流协议。这个流协议是 XMPP 的即时通信指令的传递基础,也是一个非常重要的可以被进一步利用的网络基础协议。

#### (2) 快速部署

XMPP 定义了服务器端的通信协议,开发者通过这种定义良好的协议能够快速创建和部署通信相关的应用。

#### (3) 多媒体业务

XMPP 扩展了 Jingle 协议,实现了点对点 (P2P) 的多媒体交互会话控制,如语音交互 (VOIP)、视频交互。目前,GTalk 支持多种语音和视频编码,支持的语音编码如下:

##### ①PCMU

ITU-T 制定的语音编码,频宽为 64Kbps。

##### ②PCMA

ITU-T 制定的语音编码,频宽为 64Kbps。

##### ③iLBC

专为提供稳健的 IP 语音通信而开发的语音 codec,具有 8 kHz 的采样率。iLBC codec 支持两种基本的帧长度: 30 ms 和 20 ms。

##### ④G.722

多频率语音编码,支持比特率为 64kbps、56 kbps 和 48kbps 的语音。G.722 与 3.6kHz 频率的语音编码相比较,可以处理信号宽带达 7kHz。

##### ⑤Speex

窄带 (8kHz)、宽带 (16kHz) 和超宽带 (32kHz) 压缩于同一位流,适合网络上的语音服务。



此外, 支持的视频编码如下:

#### ①H.264

高度压缩数字视频编解码器标准, 由视频编码专家组 (VCEG) 和 ISO/IEC 动态图像专家组 (MPEG) 联合组成的联合视频组 (JVT, Joint Video Team) 提出。

#### ②H.264/SVC

扩展 H.264 标准, 该标准在编码产生的编码视频时间上 (帧率)、空间上 (分辨率) 可扩展, 可产生不同帧速率、分辨率或质量等级的视频。

#### ③H.263-1998

ITU 编码标准, 是一种低码流通信标准。

#### (4) 独立于角色

XMPP 通信应用不仅支持客户端之间, 还支持客户端、服务器以及网管之间的任意通信。为了使得通信独立于角色, XMPP 详细定义了客户端 (Client)、服务器 (Server)、网关 (Gateway) 三个角色:

- 客户端是通信中的普通用户, 通常通信是在这种角色中产生的。
- 服务器用于记录客户端的信息、管理连接和路由。
- 网关用于不同即时通信客户端之间的互连, 例如 SMS (短信) 之间的互通。

下面以一个例子说明 XMPP 的应用:

```
<!--客户端 XML Stream 标签-->
<stream:stream
to='GTalk_Client1'
xmlns:stream='ClientStream'
>
<message>Client message1</message>
<message>Client message2</message>
</stream:stream>
<!--服务器 XML Stream 标签-->
<stream:stream
from='GTalk_Server'
id='10000010'
xmlns:stream='ServerStream'>
<message>Server message1</message>
<message>Client message2</message>
</stream:stream>
```

## 12.4.2 Android GTalk API 简介

Google 开放了 GTalk 的 API, 可通过导入 GTalk 的 API 包 com.google.android.gtalkservice 来开发 GTalk 相关的应用。com.google.android.gtalkservice 包含了以下类:

```
com.google.android.gtalkservice.IGTalkSession
com.google.android.gtalkservice.IGTalkService
com.google.android.gtalkservice.IGTalkService.Stub
```



```

com.google.android.gtalkservice.GTalkServiceConstants
com.google.android.gtalkservice.IChatSession

```

这些类提供了用于开发 GTalk 相关应用的方法，表 12-4 列举了 com.google.android.gtalkservice 常用的方法。

表 12-4 gtalkservice 常用的方法

方法	功能描述	返回值
createGTalkSession	该方法创建 GTalk 会话	IGTalkSession
getDefaultSession	该方法返回一个默认的会话	IGTalkSession
IGTalkService.Stub.asInterface	获取 IGTalkService 的实例	IGTalkService
isConnected	判断会话是否连接，若连接返回 true，否则返回 false	Boolean
getConnectionState	该方法获取连接状态，若返回 4 表示已连接，5 表示正在请求好友信息，0 表示网络有问题	int
requestRoster	请求获取好友列表	void
sendDataMessage	该方法用于发送消息	void

12.4.3 为 GTalk 配置 Android 模拟器

上面讲了开发中常用的 GTalk API，下面将讲述在 Android 中配置和开发 GTalk 的过程。在 Android 中开发 GTalk 可分为以下几步（流程如图 12-8 所示）：

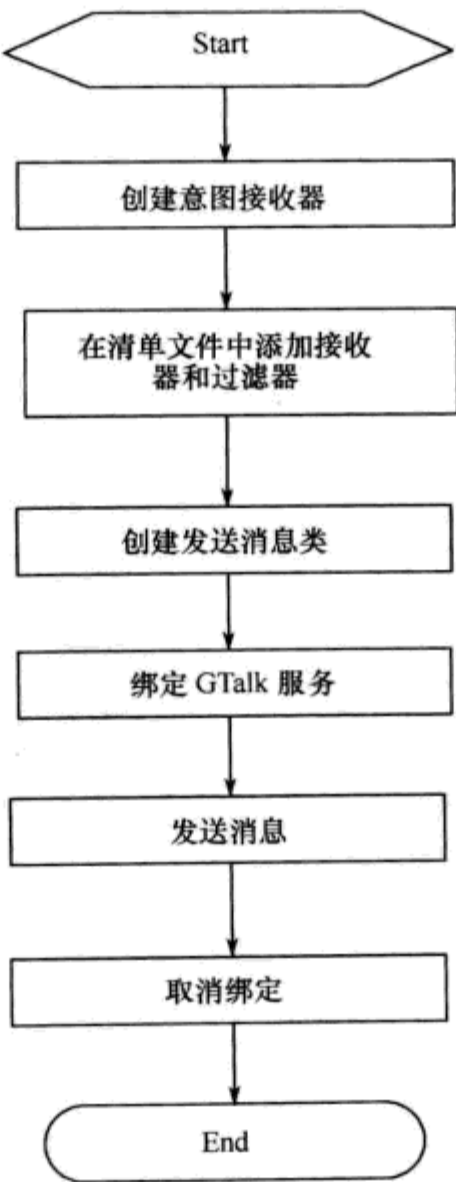


图 12-8 GTalk 开发流程

(1) 创建消息的接收器 Receiver。



创建一个意图接收器，并需要重写该接收器的 `onReceiveIntent` 方法。当接收到指定的意图时，`onReceiveIntent` 方法被调用。

(2) 清单文件中添加接收器和过滤器。

打开 `AndroidManifest.xml` 文件，在该文件添加第一步定义的接收器和过滤器。

(3) 定义发送消息类。

实现消息发送类，用于实现消息的发送。

(4) 绑定 GTalk 服务。

使用 `bindService` 方法实现发送消息和 GTalk 服务的绑定。

(5) 发送信息。

使用调用 `IGTalkSession` 的 `sendDataMessage` 方法发送消息。

(6) 取消绑定。

使用 `unbindService` 方法取消绑定。

## 12.5 应用实例详解：Google GTalk 程序

### 12.5.1 实例分析

本实例使用 GTalk 开放的接口实现其应用，在 `GTalkApplication` 中定义了以下方法：

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

该方法是 `GtalkApplication` 的入口，首先调用 `super.onCreate(savedInstanceState)` 方法执行父类的 `onCreate` 方法，然后调用 `setContentView` 方法利用 `main.xml` 初始化程序的布局。该布局包含两个文本框和两个按钮，`main.xml` 定义如下：

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 采用线性布局，垂直分布且宽度和高度同父元素相同 -->
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<!-- 输入用户名的编辑框，高度和宽度随内容变化-->
<EditText
android:id="@+id/username"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:hint="Please input your username"/>
<!-- 输入密码的编辑框，高度和宽度随内容变化-->
<EditText
android:id="@+id/password"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```



```

        android:hint="Please input your password"/>
<!-- 登录按钮, 高度和宽度随内容变化-->
<Button
    android:id="@+id/login"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Login GTalk">
</Button>
<!--退出按钮, 高度和宽度随内容变化-->
<Button
    android:id="@+id/quit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Quit GTalk">
</Button>
</LinearLayout>

```

接着调用 `findViewById` 方法获取这些标签对应的对象, 以便在 Java 程序中进行控制。

最后创建服务连接类, 注册登录按钮和退出按钮的单击事件监听器。其中, 服务连接类继承了 `ServiceConnection` 接口, 需要实现这个接口的 `onServiceConnected(ComponentName componentName, IBinder iBinder)` 和 `onServiceDisconnected(ComponentName componentname)`。`onServiceConnected` 方法在 `bindService` 绑定成功后被调用, 而 `onServiceDisconnected` 方法在取消绑定后被调用。

在 `MyServiceConnection` 的 `onServiceConnected` 方法中, 首先使用 `asInterface` 创建 `IGTalkService` 对象, 接着通过文本框输入的用户名和密码创建 Gtalk 会话。若会话创建成功且会话已连接上, 则根据连接状态处理 (假设用户名为 `GTalk_Client1`):

- 连接状态为 0 时表示连接失败, 弹出 “GTalk\_Client1 fails to connect.” 的 Toast 消息。
- 连接状态为 1 时表示自动重连, 弹出 “GTalk\_Client1 is reconnecting.” 的 Toast 消息。
- 连接状态为 2 时表示正在连接, 弹出 “GTalk\_Client1 is connecting to server.” 的 Toast 消息。
- 连接状态为 3 表示未验证, 弹出 “GTalk\_Client1 not authenticated yet.” 的 Toast 消息。
- 连接状态为 4 表示登录成功, 弹出 “GTalk\_Client1 login server.” 的 Toast 消息。
- 连接状态为 5 表示获取联系人信息, 设置呈现并创建联系人, 弹出 “GTalk\_Client1 requests roster.” 的 Toast 消息。
- 连接状态为其他值表示未知连接, 弹出 “GTalk\_Client1 unknown connection!” 的 Toast 消息。

类 `login_buttonOnClickListener` 是 `login_button` 的监听器类。单击 `quit_button` 按钮时, 该类的 `onClick` 方法被调用。该监听器的 `onClick` 方法指定了组件名或类对象的 `Intent` 为显式意图, 显式意图明确指定了 `Intent` 应该传递给哪个组件。然后使用 `bindService` 绑定 `myServiceConnection` 和 `gtalkIntent`。而类 `quit_buttonOnClickListener` 是 `quit_button` 的监听器类。单击 `quit_button` 按钮时, 该类的 `onClick` 方法被调用。该监听器的 `onClick` 方法首先使用



unbindService 取消 myServiceConnection 的绑定, 然后置 iGTalkService 对象为空。

(2) private void popUpWithToast(CharSequence message)

popUpWithToast 方法封装了 Toast 的方法, 该方法用于弹出 Toast 消息。

(3) private void createContact()

createContact 方法用于创建联系人。

(4) protected void onActivityResult(int request, int result, String data, Bundle bundle)

onActivityResult 方法返回 Activity 的执行结果。

## 12.5.2 实例实现

【实例 12-2】GTalk 的应用。

GTalkApplication.java 实现, 通过 gtalkservice 开放的接口实现 GTalk 应用。

```
package test.GTalkApplication;                                //程序打包

import com.google.android.gtalkservice.IGTalkService; //导入 GTalk 服务类
import com.google.android.gtalkservice.IGTalkSession; //导入 GTalk 会话类
import com.google.android.gtalkservice.Presence;      //导入呈现类
import android.widget.Button;                          //导入按钮控件类
import android.widget.EditText;                        //导入编辑框类
import android.content.Context;                       //导入上下文类
import android.content.Intent;                       //导入意图类
import android.widget.Toast;                          //导入 Toast 消息类
import android.view.View;                             //导入视图类
import android.os.IBinder;                            //导入绑定器类
import android.app.Activity;                         //导入活动类
import android.content.ComponentName;                //导入组件名称类
import android.content.ServiceConnection;            //导入服务连接类
import android.os.Bundle;                            //导入 Bundle 类

public class GTalkApplication extends Activity {
    private EditText username_GTalk; //声明编辑框对象 username_GTalk, 接收输入的用户名
    private EditText password_GTalk; //声明编辑框对象 password_GTalk, 接收输入的密码
    private Button login_button;     //声明登录按钮对象
    private Button quit_button;      //声明退出按钮对象
    public IGTalkSession iGTalkSession; //声明 IGTalkSession 对象 iGTalkSession
    private static final int activity = 0;
    public IGTalkService iGTalkService; //声明 IGTalkService 对象 iGTalkService
    private static final int flag_bindService = Context.BIND_AUTO_CREATE; //定义绑定服务标志
    private MyServiceConnection myServiceConnection; //声明服务连接类
```

@Override



```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);    //执行父类的 onCreate 方法
    try{
        setContentView(R.layout.main);    //使用 main.xml 作为程序布局
        username_GTalk = (EditText)findViewById(R.id.username);
        //根据 XML 定义创建 username_GTalk
        password_GTalk = (EditText)findViewById(R.id.password);
        //根据 XML 定义创建 password_GTalk
        login_button = (Button)findViewById(R.id.login);    //根据 XML 定义创建 login_button
        quit_button = (Button)findViewById(R.id.quit); //根据 XML 定义创建 quit_button
        myServiceConnection = new MyServiceConnection(); //创建服务连接类
        login_button.setOnClickListener(new login_buttonOnClickListener());
        //注册 login_button 的单击事件监听器
        quit_button.setOnClickListener(new quit_buttonOnClickListener());
        //注册 quit_button 的单击事件监听器
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(GTalkApplication.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).
            show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}

/*GTalkApplication 入口*/

private class login_buttonOnClickListener implements View.OnClickListener
{
    @Override
    public void onClick(View view) {
        // TODO Auto-generated method stub
        Intent gtalkIntent = new Intent(); //新建 Intent
        gtalkIntent.setComponent(com.google.android.gtalkservice.GTalkServiceConstants.GTALK_SERVICE_COMPONENT);
        //指定了组件名或类对象的 Intent 为显式意图, 显式意图明确指定了 Intent 应该传递给哪个组件
    }
}

```



```

        bindService(gtalkIntent, myServiceConnection, flag_bindService);
        //表示绑定此服务之后, 自动创建此服务
    }
}
/* login_button 的监听器类。单击 login_button 按钮时, 该类的 onClick 方法被调用*/

private class quit_buttonOnClickListener implements View.OnClickListener
{
    @Override
    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        unbindService(myServiceConnection);           //取消绑定
        iGTalkService = null;                          //置 iGTalkService 对象为空
        popUpWithToast("Unbind service"); //弹出取消绑定的 Toast 消息
    }
}
/* quit_button 的监听器类。单击 quit_button 按钮时, 该类的 onClick 方法被调用*/

private class MyServiceConnection implements ServiceConnection
{
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
        iGTalkService = iGTalkService.Stub.asInterface(iBinder);
        //使用 asInterface 创建 iGTalkService 对象
        try {
            String username = username_GTalk.getText().toString();
            //获取 username_GTalk 输入框作为用户名
            String password = username_GTalk.getText().toString();
            //获取 username_GTalk 输入框作为密码
            iGtalkSession = iGTalkService.createGTalkSession(username, password);
            //根据用户名和密码创建会话
            iGtalkSession.requestRoster(); //获取好友信息

            if(iGtalkSession == null){
                popUpWithToast("Failed to create GTalkSession");
                return;
            } //若 iGtalkSession 为空, 则创建会话失败

            else{
                boolean connection = false; //定义布尔变量 connection, 该变量记录是否连接的布尔值
            }
        }
    }
}

```



```

try {
    connection = iGtalkSession.isConnected(); //使用 isConnected 判断是否连接
}
/*try 块包含可能出现异常的代码*/
catch (Exception e)
{
    Toast.makeText(GTalkApplication.this, "异常错误: " + e.toString(), Toast.
        LENGTH_LONG).show();
}
/*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
int state = iGtalkSession.getConnectionState(); //获取连接状态
if(connection)
{
    switch(state)
    {
        case 4:
            popUpWithToast(username + " login server");
            break;
            /*4 表示登录成功*/

        case 5:
            Presence presence = new Presence(android.provider.Im.PresenceColumns.
                AVAILABLE, "Available");
            iGtalkSession.setPresence(presence);          //设置呈现
            createContact();          //创建联系人
            popUpWithToast(username + " requests roster");
            break;
            /*5 表示获取联系人信息*/

        case 0:
            popUpWithToast(username + " fails to connect");
            break;
            /*0 表示连接失败*/

        case 2:
            popUpWithToast(username + " is connecting to server.");
            break;
            /*2 表示正在连接*/
    }
}

```



```

        case 3:
            popUpWithToast(username + " not authenticated yet.");
            break;
            /*3 表示用户已连接上, 但未验证*/

        case 1:
            popUpWithToast(username + " is reconnectting.");
            break;
            /*1 表示自动重连*/

        default:
            popUpWithToast(username + " unknown connection!");
            break;
            /*其他值表示未知连接*/
    }
    /*根据连接状态处理*/
}

} /*try 块包含可能出现异常的代码*/
catch (Exception e)
{
    Toast.makeText(GTalkApplication.this, "异常错误: " + e.toString(), Toast.
        LENGTH_LONG).show();
}
/*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
finally
{
    //TODO
}
/*finally 中可添加处理异常的代码*/
}

/*重写父类的 onServiceConnected 方法*/

public void onServiceDisconnected(ComponentName componentname) {
    iGTalkService = null;
}
/*重写父类的 onServiceDisconnected 方法*/
}
/*定义服务连接类的子类 MyServiceConnection*/

```



```
private void popUpWithToast(CharSequence message) {
    Toast toast = Toast.makeText(GTalkApplication.this,message, Toast.LENGTH_SHORT);
    toast.show();
}
/*popUpWithToast 方法用于弹出 Toast 消*/

private void createContact() {
    Intent contact = new Intent(this,GTalkApplication.class); //创建 Intent 对象
    startSubActivity(contact, contacts);
}
/*createContact 方法用于创建联系人*/

protected void onActivityResult(int request, int result, String data, Bundle bundle) {
    super.onActivityResult(request, result, data, bundle);
    switch(request) {
        case activity:
            break;
    }
}
/*onActivityResult 返回 Activity 的执行结果*/
}
```

【代码说明】本例使用 com.google.android.gtalkservice 开发包实现了 GTalk 应用。程序运行界面如图 12-9 所示。

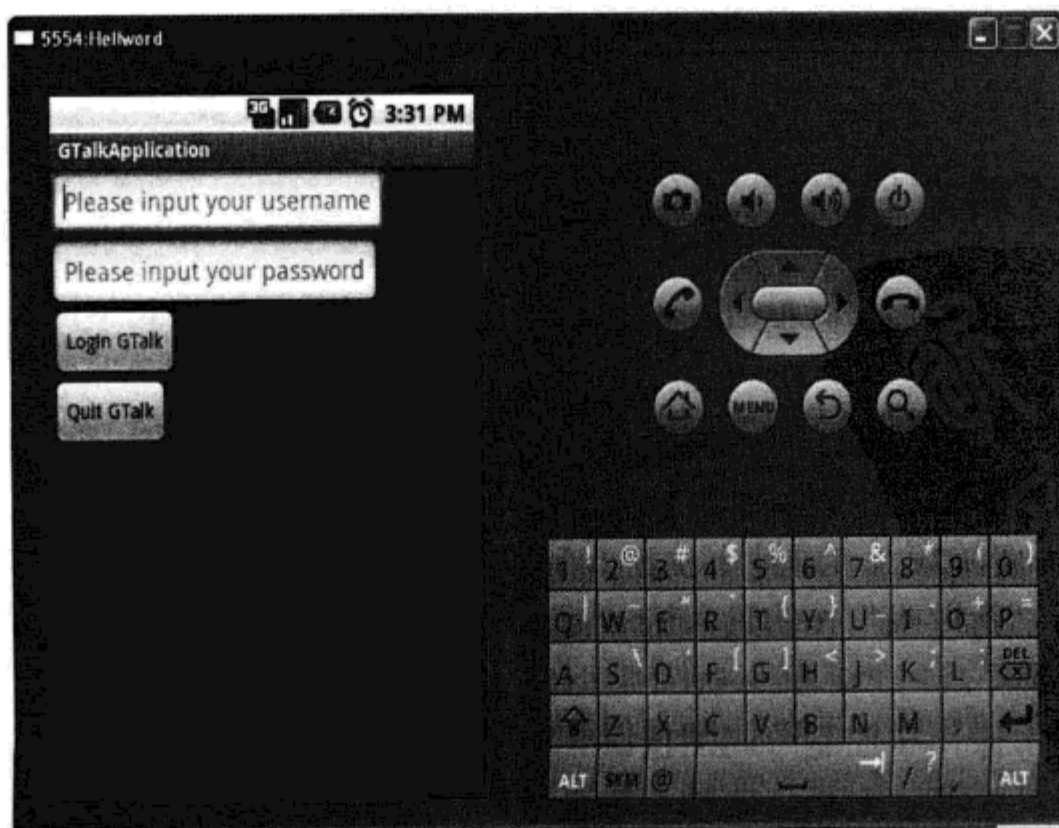


图 12-9 程序运行界面



## 第 13 章 Android 综合案例开发：俄罗斯方块

在前面的章节中介绍了 Android 的基本组件和关键应用，以及 Widget、意图、Service、网络通信、数据存储、GPS 和 GTalk 等。本文以“俄罗斯方块”游戏为例，讲述如何实现综合案例的开发。游戏是手机中不可或缺的应用，几乎每个手机上都提供了手机游戏，如贪吃蛇等。俄罗斯方块是一款比较经典的游戏，这款游戏是由俄罗斯人阿列克谢·帕基特诺夫设想出来的。俄罗斯方块的操作非常简单，主要是将给定的方块通过左移、右移、旋转等操作来尽可能构造完整的一行或多行并且消除得分。如图 13-1 所示为一款俄罗斯方块游戏的例子。

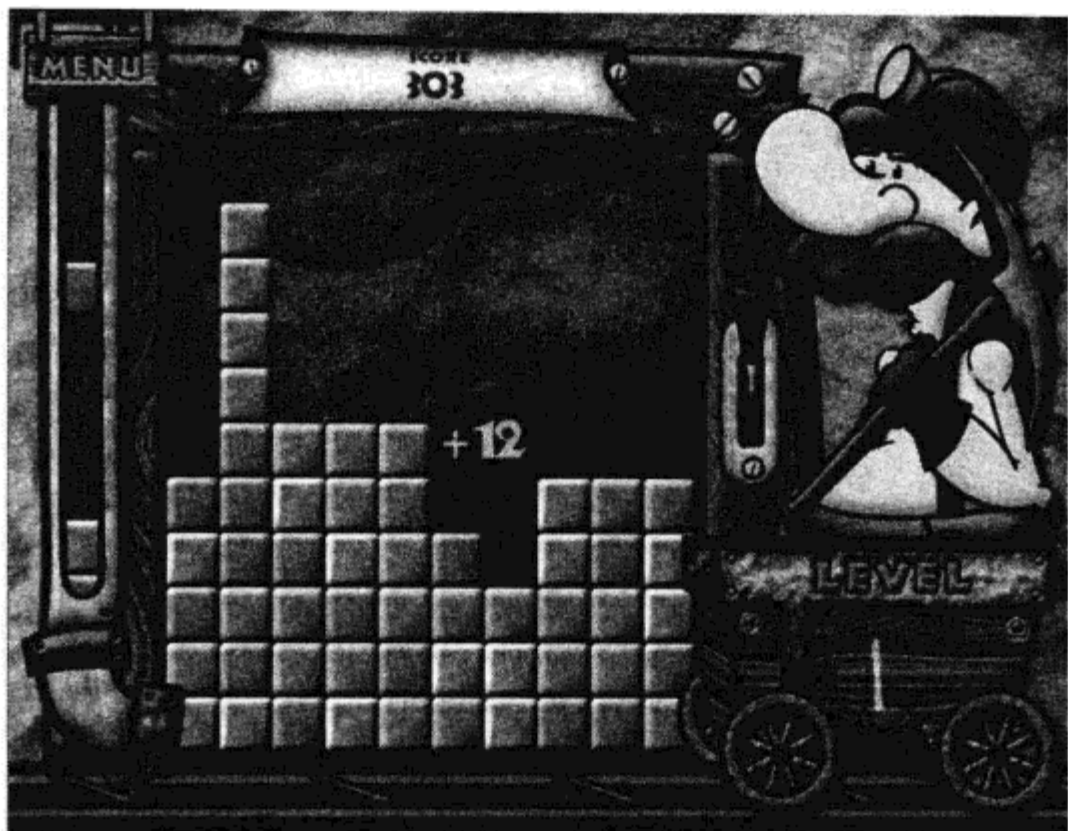


图 13-1 俄罗斯方块游戏

### 13.1 俄罗斯方块游戏功能需求

按照功能划分，俄罗斯方块游戏需要设计如下部分：

#### (1) 游戏界面

游戏需要提供一个友好且美观的界面。而游戏界面又可分主界面、游戏处理界面、帮助界面和评分界面。主界面用于控制其他界面，该界面在启动时被加载。通过单击主界面包含的按钮或者其他控件（“新游戏”、“继续上次游戏”、“帮助”等），可以从主界面进入按钮或控件对应的界面，如单击“新游戏”按钮进入游戏处理界面。

#### (2) 游戏按键

游戏按键是用于操作和控制游戏的按键。对于俄罗斯方块，其操作比较简单。俄罗斯方块游戏的操作主要包括左移、右移、旋转、加速下降、直接下降。例如可以如下定义按键：

①方向键←：左移。



- ②方向键→: 右移。
- ③方向键↑: 旋转。
- ④方向键↓: 加速下降。
- ⑤回车键: 直接下降。

### (3) 游戏处理

用户进入游戏后, 程序进入游戏处理界面。在该界面中需要不断产生新的方块, 并且根据消去的行数实时更新分数。当用户暂停时, 能实时地保存本次游戏。在游戏结束后, 能对用户的得分情况进行排名。

## 13.2 俄罗斯方块游戏 UI 设计

在本章实现的游戏, 使用 XML 定义游戏的界面。

游戏主界面使用线性布局, 垂直分布且宽度和高度同父元素相同。该布局包含了 8 个子相对布局, 分别用于继续上次游戏、新建游戏、显示游戏帮助、排名、设置难度、提示设置难度、设置静音和退出游戏。

### (1) “继续上次游戏” 布局

该布局只包含一个“继续上次游戏”按钮:

```
<Button
    android:id="@+id/bt_continue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="继续上次游戏"
    android:layout_centerHorizontal="true"
/>
```

### (2) “新游戏” 布局

该布局只包含一个“新游戏”按钮:

```
<Button
    android:id="@+id/bt_new"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="新游戏"
    android:layout_centerHorizontal="true"
/>
```

### (3) “帮助” 布局

该布局只包含一个“帮助”按钮:

```
<Button
    android:id="@+id/bt_help"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="帮助"
/>
```



```
android:layout_centerHorizontal="true"
/>
```

#### (4) “排行榜” 布局

该布局只包含一个“排行榜”按钮：

```
<Button
    android:id="@+id/bt_rank"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="排行榜"
    android:layout_centerHorizontal="true"
/>
```

#### (5) “提示选择难度” 布局

该布局只包含一个用于提示用户操作的文本视图：

```
<TextView
    android:id="@+id/tv_level"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="选择难度"
    android:textColor="#ee000000"
    android:layout_centerHorizontal="true"
/>
```

#### (6) “选择难度” 布局

该布局只包含一个用于显示难度的文本视图、一个增加难度的按钮和一个减小难度的按钮：

```
<TextView
    android:id="@+id/tv_speed"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:text="1"
    android:textSize="20px"
    android:textColor="#ffffff00"
    android:background="#6600ee00"
/>

<Button
    android:id="@+id/bt_pre"
    android:layout_width="50px"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/tv_speed"
    android:text="&lt;&lt;"
    android:textSize="20px"
```



```

android:background = "#70dfdfd"
android:textColorHighlight = "#ffff0000"
android:layout_marginRight = "10px"
/>
<Button
android:id = "@+id/bt_next"
android:layout_width = "50px"
android:layout_height = "wrap_content"
android:layout_toRightOf = "@id/tv_speed"
android:text = ">>"
android:textSize = "20px"
android:background = "#80cfcfcf"
    android:layout_marginLeft = "10px"
/>

```

### (7) “设置静音” 布局

该布局只包含一个用于提示设置静音的文本视图和设置静音的复选框:

```

<CheckBox
    android:id = "@+id/cb_voice"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:checked = "true"
    android:layout_centerHorizontal = "true"
/>
<!-- 该布局包含了用于提示的文本视图-->
<TextView
    android:id = "@+id/tv_voice"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "声音开关:"
    android:textColor = "#ee000000"
    android:layout_toLeftOf = "@id/cb_voice"
/>

```

### (8) “退出游戏” 布局

该布局只包含一个用于控制游戏退出的按钮:

```

<Button
    android:id = "@+id/bt_exit"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "退出"
    android:layout_centerHorizontal = "true"

```



/>

整个布局定义如下:

<!-- 程序主界面, 使用线性布局, 垂直分布且宽度和高度同父元素相同。该布局包含了 8 个子相对布局-->

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical"

android:background="@drawable/tetris"

>

<!-- 第一个相对布局。宽度同父元素相同, 高度随内容变化-->

<RelativeLayout

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_marginTop="50px"

>

<!-- 该布局包含了一个按钮, 用于控制游戏从上次保存的位置开始-->

<Button

android:id="@+id/bt\_continue"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="继续上次游戏"

android:layout\_centerHorizontal="true"

/>

</RelativeLayout>

<!-- 第二个相对布局。宽度同父元素相同, 高度随内容变化-->

<RelativeLayout

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

>

<!-- 该布局包含了一个按钮, 用于新建游戏, 而不是从上次保存的位置开始-->

<Button

android:id="@+id/bt\_new"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="新游戏"

android:layout\_centerHorizontal="true"

/>

</RelativeLayout>



```

<!-- 第三个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
android:layout_width = "fill_parent"
android:layout_height = "wrap_content"
>
    <!-- 该布局包含了一个按钮，用于控制从主界面进入帮助界面-->
    <Button
        android:id = "@+id/bt_help"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "帮助"
        android:layout_centerHorizontal= "true"
    />
</RelativeLayout>
<!-- 第四个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
android:layout_width = "fill_parent"
android:layout_height = "wrap_content"
>
    <!-- 该布局包含了一个按钮，用于控制从主界面进入排名界面-->
    <Button
        android:id = "@+id/bt_rank"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "排行榜"
        android:layout_centerHorizontal= "true"
    />
</RelativeLayout>
<!-- 第五个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
android:layout_width = "fill_parent"
android:layout_height = "wrap_content"
>
    <!-- 该布局包含了一个文本视图，用于提示用户操作-->
    <TextView
        android:id = "@+id/tv_level"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "选择难度"
        android:textColor = "#ee000000"
    />
</RelativeLayout>

```



```

    android:layout_centerHorizontal="true"
/>
</RelativeLayout>
<!-- 第六个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
>
<!-- 该布局包含了一个文本视图，用于显示游戏难度（默认设置为 1）-->
<TextView
    android:id="@+id/tv_speed"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:text="1"
    android:textSize="20px"
    android:textColor="#ffffff00"
    android:background="#6600ee00"
/>
<!-- 该布局包含了一个减小难度的按钮-->
<Button
    android:id="@+id/bt_pre"
    android:layout_width="50px"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/tv_speed"
    android:text="&lt;&lt;"
    android:textSize="20px"
    android:background="#70dfdfdf"
    android:textColorHighlight="#ffff0000"
    android:layout_marginRight="10px"
/>
<!-- 该布局包含了一个增加难度的按钮-->
<Button
    android:id="@+id/bt_next"
    android:layout_width="50px"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/tv_speed"
    android:text=">>"
    android:textSize="20px"
    android:background="#80cfcfcf"

```



```

        android:layout_marginLeft = "10px"
    />

</RelativeLayout>
<!-- 第七个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:layout_marginTop = "15px"
>
    <!-- 该布局包含了一个控制是否静音的复选框-->
    <CheckBox
        android:id = "@+id/cb_voice"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:checked = "true"
        android:layout_centerHorizontal = "true"
    />
    <!-- 该布局包含了用于提示的文本视图-->
    <TextView
        android:id = "@+id/tv_voice"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "声音开关:"
        android:textColor = "#ee000000"
        android:layout_toLeftOf = "@id/cb_voice"
    />
</RelativeLayout>
<!-- 第八个相对布局。宽度同父元素相同，高度随内容变化-->
<RelativeLayout
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:layout_marginTop = "15px"
>
    <Button
        android:id = "@+id/bt_exit"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "退出"
        android:layout_centerHorizontal = "true"

```



```

/>
</RelativeLayout>
</LinearLayout>

```

图 13-2 所示是根据上述 XML 定义的游戏主界面。



图 13-2 游戏主界面

单击“新游戏”和“继续上次游戏”按钮可进入游戏处理界面。由于游戏处理界面是动态变化的，因此程序使用 Paint 对象来实时绘制界面。下面是绘制游戏画面的实现代码：

```

private void paintTetrisGame(Canvas TetrisCanvas)
{
    mycourt.paintCourt(TetrisCanvas); //绘制游戏背景
    currentTile.paintTile(TetrisCanvas); //绘制当前方块
    Tertrispaint.setTextSize(20); //设置字体
    paintTetrisNextTile(TetrisCanvas); //绘制下一个方块
    paintGameLevel(TetrisCanvas); //绘制游戏等级
    paintGameScore(TetrisCanvas); //绘制游戏得分
    paintGameDecline(TetrisCanvas); //绘制消去的行数
}

```

下面介绍绘制游戏界面的每个过程：

（1）绘制“游戏等级”文本以及游戏等级数值

使用 Paint 的 drawText 方法在指定位置绘制“游戏等级”文本和数值。“游戏等级”文本显示的位置需要使用整个控件的 X 和 Y 坐标来确定，其中“游戏等级”文本的 X 和 Y 坐标为：

```

getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
getBlockDistance(9);

```

游戏等级数值的 X 和 Y 坐标为：

```

getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
getBlockDistance(11);

```

getBlockDistance 方法和 getRightMarginToCourt 方法定义如下：



```
private float getBlockDistance(float blockNum)
{
    return blockNum * Court.BLOCK_WIDTH;
}

private float getRightMarginToCourt()
{
    return (float)10.0;
}
```

绘制过程实现如下:

```
private void paintGameLevel(Canvas TetrisCanvas)
{
    String GameLevel_title = "游戏等级:";
    float GameLevel_title_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
    //设置"游戏等级"文本的 x 坐标
    float GameLevel_title_y = getBlockDistance(9); //设置"游戏等级"文本的 y 坐标
    String GameLevel = String.valueOf(gamelevel);
    float GameLevel_x = getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
    //设置游戏等级数值的 x 坐标
    float GameLevel_y = getBlockDistance(11); //设置游戏等级数值的 y 坐标

    Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
    TetrisCanvas.drawText(GameLevel_title,GameLevel_title_x, GameLevel_title_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
    Tertrispaint.setColor(Color.RED); //设置画笔颜色
    TetrisCanvas.drawText(GameLevel,GameLevel_x, GameLevel_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
}

/*使用 paintGameLevel 方法绘制"游戏等级"文本以及游戏等级数值*/
```

(2) 绘制“游戏分数”文本以及游戏分数数值

使用 Paint 的 drawText 方法在指定位置绘制“游戏分数”文本和数值。其中“游戏分数”文本的 X 和 Y 坐标为:

```
getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
getBlockDistance(12);
```

游戏分数的 X 和 Y 坐标为:

```
getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
getBlockDistance(15);
private void paintGameScore(Canvas TetrisCanvas)
{
```

```
    String score_title = "游戏得分:";
    float score_title_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
```



```

        //设置"游戏得分"文本的 x 坐标
        float score_title_y = getBlockDistance(13); //设置"游戏得分"文本的 y 坐标
        String score = String.valueOf(gamescore); //将分数转换成字符串
        float score_x = getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
        //设置游戏分数的 x 坐标
        float score_y = getBlockDistance(15); //设置游戏分数的 y 坐标
        Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
        TetrisCanvas.drawText(score_title,score_title_x, score_title_y,Tertrispaint);
        //使用画笔在指定的位置绘制文本
        Tertrispaint.setColor(Color.RED); //设置画笔颜色
        TetrisCanvas.drawText(score,score_x, score_y,Tertrispaint); //使用画笔在指定的位置绘制文本
    }
    /*使用 paintGameScore 方法绘制"游戏得分"文本以及游戏分数*/

```

### (3) 绘制“消去行数”文本以及消去行数数值

使用 Paint 的 drawText 方法在指定位置绘制“消去行数”文本和数值。其中“消去行数”文本的 X 和 Y 坐标为:

```

getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
getBlockDistance(17);

```

消去行数数值的 X 和 Y 坐标为:

```

getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
getBlockDistance(19);
private void paintGameDecline(Canvas TetrisCanvas)
{
    String decline_titlr = "消去行数:";
    float decline_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt(); /
    //设置"消去行数"文本的坐标
    float decline_y = getBlockDistance(17); //设置"消去行数"文本的 y 坐标
    String declinenum = String.valueOf(gameDecline);
    float declinenum_x = getBlockDistance(Court.COURT_WIDTH)+2*getRightMarginToCourt();
    //设置消去行数数值的 x 坐标
    float declinenum_y = getBlockDistance(19); //设置消去行数数值的 y 坐标

    Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
    TetrisCanvas.drawText(decline_titlr,decline_x, decline_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
    Tertrispaint.setColor(Color.RED); //设置画笔颜色
    TetrisCanvas.drawText(declinenum,declinenum_x, declinenum_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
}
/*使用 paintGameDecline 方法绘制"消去行数"文本以及消去行数数值*/

```



#### (4) 绘制下一个方块

由于绘制的是个图像，因此使用 `paintImage` 在指定位置绘制：

```
(int)(Court.BEGIN_DRAW_X+getBlockDistance(Court.COURT_WIDTH) + getBlockDistance((float)
(i+0.5)) );
(int)( getBlockDistance((float)(j+0.5)));
private void paintTetrisNextTile(Canvas TetrisCanvas)
{
    int i,j;
    for(i = 0;i<4;i++)
    {
        for(j = 0;j<4;j++)
        {
            if(nextTile.mTile[i][j] != 0)
            {
                int title_x = (int)(Court.BEGIN_DRAW_X+getBlockDistance(Court.COURT_WIDTH)
+ getBlockDistance((float) (i+0.5)) );
                //设置方块的 x 坐标
                int title_y = (int)( getBlockDistance((float)(j+0.5))); //设置方块的 y 坐标
                DrawTool.paintImage( TetrisCanvas,resourceStore.getBlock(nextTile.getColor()-1),
title_x,title_y );
                //使用画笔在指定的位置绘制图像
            }
        }
    }
}
```

/\*使用 `paintTetrisNextTile` 方法绘制下一个方块。注意，这里绘制的方块是个图像，因此使用 `paintImage` 方法\*/

图 13-3 显示了本实例设计的游戏开始界面。

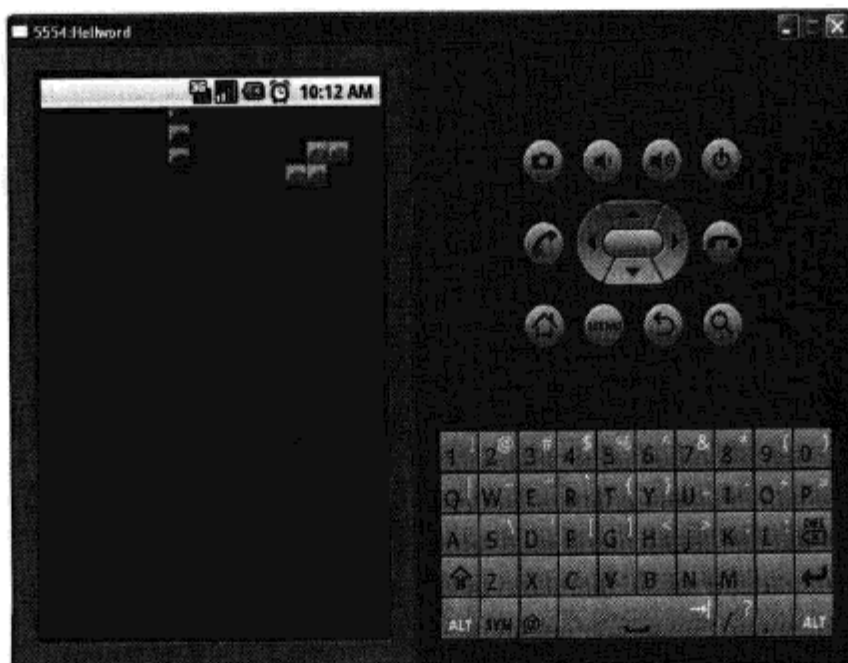


图 13-3 游戏开始界面



### (5) 绘制游戏结束画面

游戏结束时需要重新绘制整个画面。paintGameOver 方法用于游戏结束画面的绘制。图 13-4 显示的是游戏结束界面。

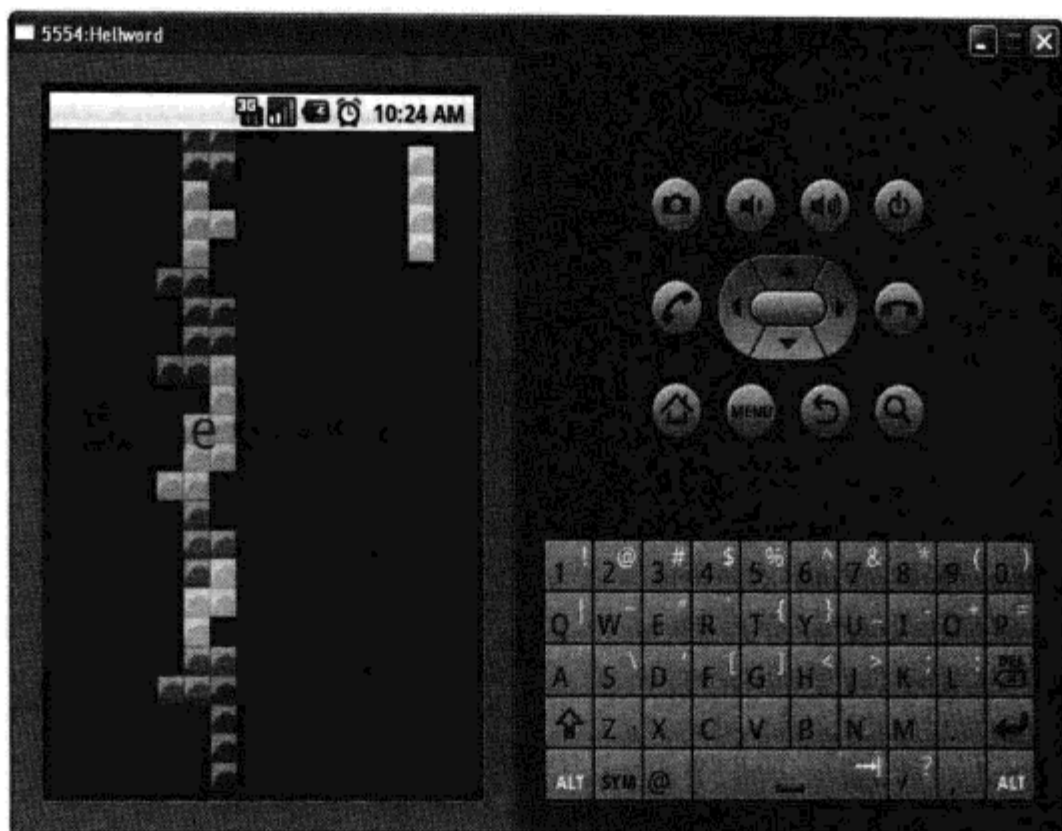


图 13-4 游戏结束界面

下面是其实现过程：

```
private void paintGameOver(Canvas TetrisCanavs)
{
    int textsize = 40;
    boolean AntiAlias = true;
    String Gameover = "Game Over";
    float Gameover_x = getBlockDistance(1); //设置 Game Over 文本的 x 坐标
    float Gameover_y = getBlockDistance(Court.COURT_HEIGHT/2-2);
    //设置 Game Over 文本的 y 坐标
    Paint mypaint = new Paint();//新建画图对象

    paintTetrisGame(TetrisCanavs); //生成游戏界面
    mypaint.setTextSize(textsize); //设置字体大小
    mypaint.setAntiAlias(AntiAlias); //设置模糊边界
    mypaint.setARGB(0xe0,0xff,0x00,0x00); //设置颜色
    TetrisCanavs.drawText(Gameover,Gameover_x,Gameover_y,mypaint);//绘制文本
}

/*paintGameOver 方法游戏结束时被调用*/
```

单击“帮助”按钮进入游戏帮助界面，游戏帮助界面包含游戏的信息提示。下面是本例的游戏帮助界面实现代码：

```
<!-- 帮助界面使用线性布局，垂直分布，高度和宽度同父元素相同。其包含 4 个文本视图，用于信息提示
```



```
-->
<!-- 提示"按 UP 方向键：旋转方块"的文本视图，高度随内容变化且宽度同父元素相同-->
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:textSize = "20px"
    android:textColor = "#ff000000"
    android:background = "#a09090ff"
    android:text = "按 UP 方向键：旋转方块"
/>

<!-- 提示"按 DOWN 方向键：加速移动"的文本视图，高度随内容变化且宽度同父元素相同-->
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:textSize = "20px"
    android:textColor = "#ff000000"
    android:background = "#a09090ff"
    android:text = "按 DOWN 方向键：加速移动"
/>

<!-- 提示"按 LEFT 方向键：左移方块 "的文本视图，高度随内容变化且宽度同父元素相同-->
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:textSize = "20px"
    android:textColor = "#ff000000"
    android:background = "#a09090ff"
    android:text = "按 LEFT 方向键：左移方块 "
/>

<!-- 提示"按 RIGHT 方向键：右移方块"的文本视图，高度随内容变化且宽度同父元素相同-->
<TextView
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:textSize = "20px"
    android:textColor = "#ff000000"
    android:background = "#a09090ff"
    android:text = "按 RIGHT 方向键：右移方块 "
/>

<!-- 提示"按回车键：下落到底"的文本视图，高度随内容变化且宽度同父元素相同-->
<TextView
    android:layout_width = "fill_parent"
```



```

android:layout_height = "wrap_content"
android:textSize = "20px"
android:textColor = "#ff000000"
android:background = "#a09090ff"
android:text = "按回车键：下落到底"
android:layout_weight = "1"
/>

```

图 13-5 显示了本实例设计的游戏帮助界面。

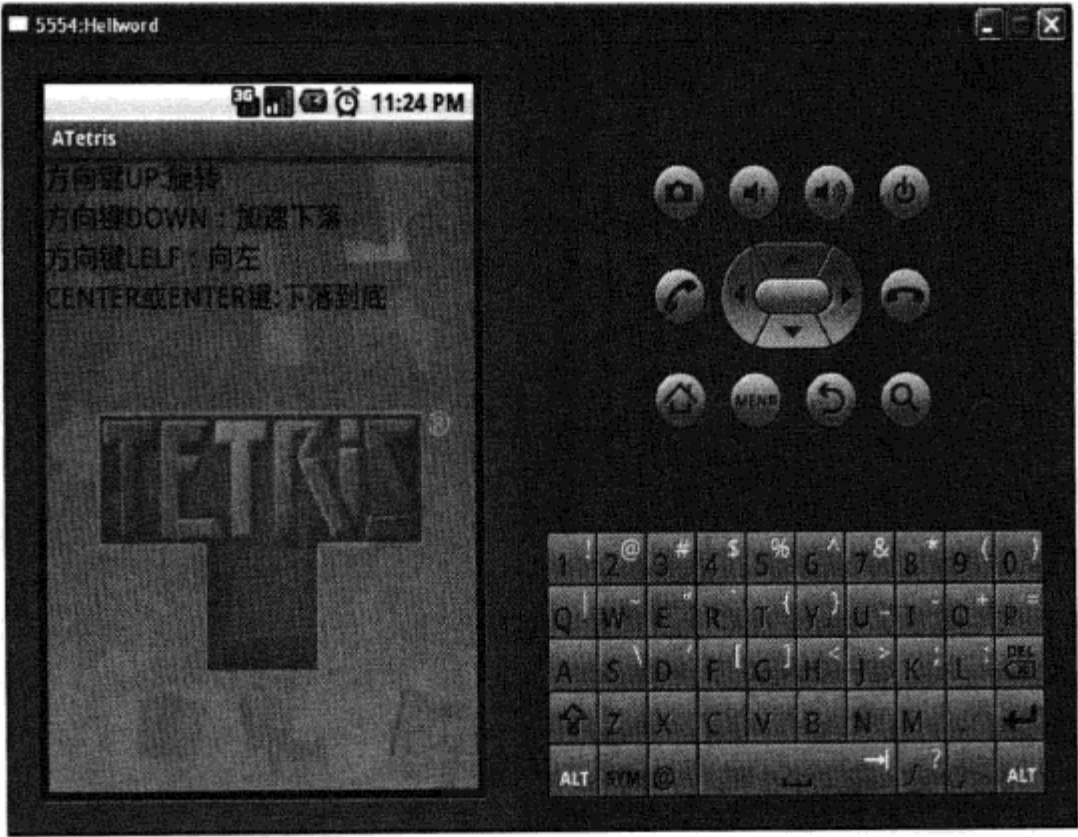


图 13-5 游戏帮助界面

单击“排行榜”按钮可进入游戏排名界面。排名界面使用线性布局，垂直分布，高度和宽度同父元素相同，其中包含一个相对布局和一个列表视图。排名界面定义如下：

```

<?xml version="1.0" encoding="utf-8"?>
<!-- 排名界面使用线性布局，垂直分布，高度和宽度同父元素相同，其中包含一个相对布局和一个列表视图 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation = "vertical"
    android:background = "@drawable/tetris"
    >
    <!-- 相对布局的宽度同父元素相同，高度随内容变化，包含三个用于显示名次、得分以及签名的文本视图-->
    <RelativeLayout
        android:layout_width = "fill_parent"

```



```

        android:layout_height = "wrap_content"
    >
    <!-- 显示名次的文本视图，高度和宽度随内容变化-->
    <TextView
        android:id = "@+id/item_rank"
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "名次"
        android:layout_alignParentLeft = "true"
        android:layout_alignParentTop = "true"
        android:textSize = "20px"
        android:textColor = "#ff000000"
    />
    <!-- 显示得分的文本视图，高度和宽度随内容变化-->
    <TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "得分"
        android:layout_centerHorizontal = "true"
        android:layout_marginLeft = "20px"
        android:textSize = "20px"
        android:textColor = "#ff000000"
    />
    <!-- 显示签名的文本视图，高度和宽度随内容变化-->
    <TextView
        android:layout_width = "wrap_content"
        android:layout_height = "wrap_content"
        android:text = "签名"
        android:layout_alignParentRight = "true"
        android:layout_alignParentTop = "true"
        android:textSize = "20px"
        android:textColor = "#ff000000"
    />
    </RelativeLayout>
    <!-- 显示排名情况的列表视图，宽度同父元素相同，高度随内容变化-->
    <ListView
        android:id = "@+id/rank_list"
        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:background = "#a09090ff"
    />

```



```
android:layout_weight = "1"
/>
</LinearLayout>
```

图 13-6 显示了本实例设计的游戏排名界面。

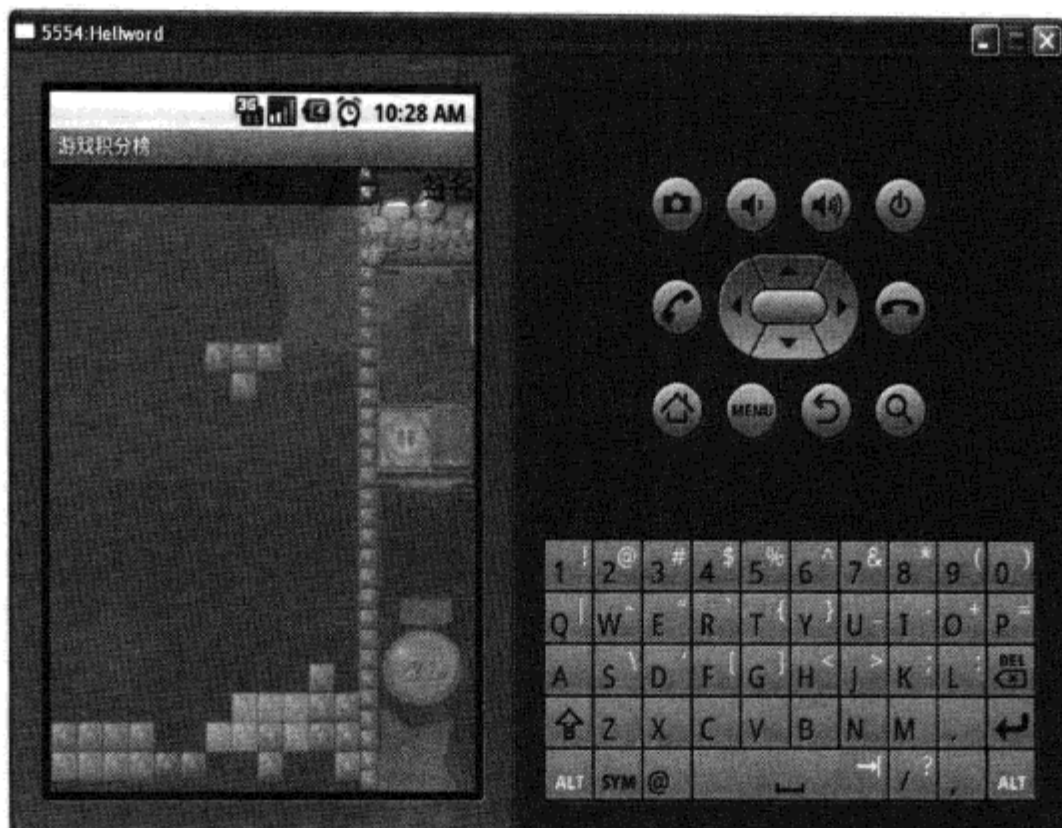


图 13-6 游戏排名界面

### 13.3 俄罗斯方块游戏功能实现

主程序 Tetris.java 实现可分为以下几个步骤：

(1) 主程序使用 menu.xml 加载布局。

(2) 主程序使用 init 方法初始化控件和参数。初始化的控件和参数有：

- 新建游戏的按钮控件 BUTTON\_NEW\_GAME。
- 继续游戏的按钮控件 BUTTON\_CONTINUE。
- 获取游戏帮助的按钮控件 BUTTON\_Game\_HELP。
- 评分的按钮控件 BUTTON\_RATING。
- 增加难度的按钮控件 BUTTON\_PREVIOUS。
- 减小难度的按钮控件 BUTTON\_NEXT。
- 退出游戏的按钮控件 BUTTON\_NEXT。
- 显示难度的文本视图 TEXTVIEW\_LEVEL。
- 控制是否静音的复选框 CHECKBOX\_VOICE。
- 调用 NewTetrisGame 活动的意图 CONTINUE\_Game\_Intent。
- 调用 TetrisGameHelp 活动的意图 HELP\_Game\_Intent。
- 调用 RankTetrisGame 活动的意图 RANK\_Game\_Intent。
- 存储游戏的设置的 SharedPreferences 对象。

(3) 使用 creatComponent 方法根据 XML 定义的标签创建控件对象，执行完之后弹出 Toast 消息：



```

BUTTON_NEW_GAME = (Button)findViewById(R.id.bt_new);
BUTTON_CONTINUE = (Button)findViewById(R.id.bt_continue);
BUTTON_Game_HELP = (Button)findViewById(R.id.bt_help);
BUTTON_RANK = (Button)findViewById(R.id.bt_rank);
BUTTON_PREVIOUS = (Button)findViewById(R.id.bt_pre);
BUTTON_NEXT = (Button)findViewById(R.id.bt_next);
BUTTON_EXIT = (Button)findViewById(R.id.bt_exit);
TEXTVIEW_LEVEL = (TextView)findViewById(R.id.tv_speed);
CHECKBOX_VOICE = (CheckBox)findViewById(R.id.cb_voice);
/*根据 XML 定义的标签创建 BUTTON_NEW_GAME、BUTTON_CONTINUE、BUTTON_Game_HELP、
BUTTON_RANK、BUTTON_PREVIOUS、BUTTON_NEXT、BUTTON_EXIT、TEXTVIEW_LEVEL
和 CHECKBOX_VOICE 控件对象*/
Toast.makeText(Tetris.this, "Finish to creat component", Toast.LENGTH_LONG).show();
//在该活动上 (Tetris) 弹出 Toast 消息

```

(4) 使用 `createIntent` 方法创建活动跳转的意图, 其中跳转关系如下:

- 意图 `NEW_Game_Intent` 负责跳转到 `NewTetrisGame.class`, 这个类负责处理游戏。
- 意图 `RANK_Game_Intent` 负责跳转到 `NewTetrisGame.class`, 这个类负责游戏排名。
- 意图 `CONTINUE_Game_Intent` 负责跳转到 `NewTetrisGame.class`, 这个类负责处理游戏。
- 意图 `HELP_Game_Intent` 负责跳转到 `TetrisGameHelp.class`, 这个类弹出帮助信息。

(5) 使用 `registerListener` 方法注册控件的监听器:

- `BUTTON_NEW_GAME` 控件对应的监听器为 `BUTTON_NEW_GAMEListener`, 该类重写了 `View.OnClickListener` 方法。在该方法中首先设置新游戏标志, 然后将声音和难度信息放到 `NEW_Game_Intent` 中, 最后执行 `NewTetrisGame.class` 的活动。
- `BUTTON_CONTINUE` 控件对应的监听器为 `BUTTON_CONTINUEListener`, 该类重写了 `View.OnClickListener` 方法。在该方法中首先设置继续上次游戏的标志, 然后将声音和难度信息放到 `CONTINUE_Game_Intent` 中, 最后跳转执行 `NewTetrisGame.class` 的活动。
- `BUTTON_Game_HELP` 控件对应的监听器为 `BUTTON_Game_HELPListener`, 该类重写了 `View.OnClickListener` 方法。在该方法中使用 `startActivity` 方法跳转执行 `TetrisGameHelp.class` 的活动。
- `BUTTON_RANK` 控件对应的监听器为 `BUTTON_RANKListener`, 该类重写了 `View.OnClickListener` 方法。该方法跳转执行 `RANK_Game_Intent.class` 的活动。
- `BUTTON_PREVIOUS` 控件对应的监听器为 `BUTTON_PREVIOUSListener`, 该类重写了 `View.OnClickListener` 方法。该方法减小显示的难度。
- `BUTTON_NEXT` 控件对应的监听器为 `BUTTON_NEXTListener`, 该类重写了 `View.OnClickListener` 方法。该方法增加显示的难度。

(6) 使用 `loadLastSettings` 方法加载上一次设置的参数, 例如游戏难度、是否静音等。

本实例使用 `SharedPreferences` 对象存储设置的参数, 可调用 `getSharedPreferences` 方法获取 `SharedPreferences` 对象。通过该对象获取存储的数据, `getSharedPreferences` 需要指定两个参数: 存储键值对的 XML 文件名和打开模式, 然后根据获取的 `GameLevel` 和 `GameVoice`, 设置界面显示难度的文本视图和复选框的状态。



(7) 游戏退出时, onStop 方法被调用。

该方法使用 saveTetrisGameSettings 保存游戏设置的参数。saveTetrisGameSettings 方法使用 SharedPreferences 存取数据。Preferences 文件存储在 APK 的安装目录下, 可通过 Andoird DDMS (Dalvik Debug Monitor Service) 查看 Preferences 文件:

```
private void saveTetrisGameSettings()
{
    Editor editor = sharedPreferences.edit(); //根据 sharedPreferences 获取编辑器对象
    editor.putInt(GameLevel, mylevel);        //将 mylevel 的键值对放入编辑器中
    editor.putBoolean(GameVoice, CHECKBOX_VOICE.isChecked());
    //将复选框状态的键值对放入编辑器中
    editor.commit(); //将 mylevel 和 CHECKBOX_VOICE.isChecked() 的键值对存储到 sharedPreferences 中
}

/*saveTetrisGameSettings 方法使用 Preference 保存该游戏设置的参数*/
```

下面是 Tetris.java 的实现过程:

```
/*Tetris Main function*/
import android.app.Activity;           //导入 Activity 类
import android.content.Intent;        // 导入 Intent 类
import android.content.SharedPreferences; //导入 SharedPreferences 类, 使用 Preferences 存储数据
import android.content.SharedPreferences.Editor;
import android.os.Bundle;              //导入 Bundle 类
import android.view.View;              //导入 View 类
import android.widget.Button;           //导入按钮类
import android.widget.CheckBox;         //导入复选框类
import android.widget.TextView;         //导入文本视图类
import android.widget.Toast;           //导入 Toast 类

public class Tetris extends Activity {

    public static final int IS_NEW_GAME = 0; //定义整型常量 IS_NEW_GAME, 用于标识是否是新游戏
    public static final int IS_CONTINUE = 1; //定义整型常量 IS_CONTINUE, 用于标识是否继续上次游戏
    public static final String GameLevel = "level"; //定义字符串常量 GameLevel
    public static final String GameVoice = "voice"; //定义字符串常量 GameVoice
    public static final String Settings = "settingInfo"; //定义保存设置信息的文件名
    private int mylevel;                  //定义整型变量, 用于记录用户设置的难度
    private Button BUTTON_NEW_GAME; //声明新建游戏的按钮控件 BUTTON_NEW_GAME
    private Button BUTTON_CONTINUE; //声明继续游戏的按钮控件 BUTTON_CONTINUE
    private Button BUTTON_Game_HELP; //声明获取游戏帮助的按钮控件 BUTTON_Game_HELP
    private Button BUTTON_RANK; //声明评分的按钮控件 BUTTON_RANK
    private Button BUTTON_PREVIOUS; //声明按钮控件 BUTTON_PREVIOUS
    private Button BUTTON_NEXT; //声明按钮控件 BUTTON_NEXT
```



```
private Button BUTTON_EXIT; //声明退出游戏的按钮控件 BUTTON_NEXT
private TextView TEXTVIEW_LEVEL; //声明显示难度的文本视图控件 BUTTON_NEXT
private CheckBox CHECKBOX_VOICE; //声明静音复选框控件 CHECKBOX_VOICE
private SharedPreferences sharedPreferences; //声明 SharedPreferences 对象, 用于存储游戏的设置
private Intent NEW_Game_Intent; //声明用于调用 NewTetrisGame 活动的意图
private Intent CONTINUE_Game_Intent; //声明用于调用 NewTetrisGame 活动的意图
private Intent HELP_Game_Intent; //声明用于调用 TetrisGameHelp 活动的意图
private Intent RANK_Game_Intent; //声明用于调用 RankTetrisGame 活动的意图
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle bundle) {
    super.onCreate(bundle); //执行父类的 onCreate 方法
    setContentView(R.layout.menu); //使用 menu.xml 加载布局
    try {
        init(); //使用 init 方法初始化控件和参数
        creatComponent(); //根据 XML 定义的标签创建控件对象
        createIntent(); //创建活动跳转的意图
        registerListener(); //注册控件的监听器
        loadLastSettings(); //使用 loadLastSettings 方法加载上一次设置的参数, 例如游戏难度、是否静音等
    }
    /*try 块包含可能出现异常的代码*/
    catch (Exception e)
    {
        Toast.makeText(Tetris.this, "异常错误: " + e.toString(), Toast.LENGTH_LONG).show();
    }
    /*catch 捕捉异常, 若出现异常, 则显示异常的 Toast 消息*/
    finally
    {
        //TODO
    }
    /*finally 中可添加处理异常的代码*/
}
/*onCreate 方法是 Tetris 的入口, 程序从 onCreate 开始执行*/

private void init()
{
    BUTTON_NEW_GAME = null;
    BUTTON_CONTINUE = null;
    BUTTON_Game_HELP = null;
    BUTTON_RANK = null;
}
```



```

    BUTTON_PREVIOUS = null;
    BUTTON_NEXT = null;
    BUTTON_EXIT = null;
    TEXTVIEW_LEVEL = null;
    CHECKBOX_VOICE = null;
    /*设置 BUTTON_NEW_GAME、BUTTON_CONTINUE、BUTTON_Game_HELP、BUTTON_RATING、
    BUTTON_PREVIOUS、BUTTON_NEXT、BUTTON_EXIT、TEXTVIEW_LEVEL 和 CHECKBOX_
    VOICE 控件为空*/
    mylevel = 1; //初始化 mylevel
    NEW_Game_Intent = null;
    CONTINUE_Game_Intent = null;
    HELP_Game_Intent = null;
    RANK_Game_Intent = null;
    /*设置 NEW_GAME_Intent、CONTINUE_Intent、Game_HELP_Intent 和 Game_RATING_Intent 为空*/
    sharedPreferences = getSharedPreferences(Settings,0);
    /*调用 getSharedPreferences 生成 SharedPreferences 对象
    *调用 getSharedPreferences 时，需要指定两个参数：存储键值对的 XML 文件名和打开模式*/
    Toast.makeText(Tetris.this, "Finish to init", Toast.LENGTH_LONG).show();
    //在该活动上（Tetris）弹出 Toast 消息
}
/*使用 init 方法初始化控件和参数，执行完之后弹出"Finish to init"的 Toast 消息*/

private void creatComponent()
{
    BUTTON_NEW_GAME = (Button)findViewById(R.id.bt_new);
    BUTTON_CONTINUE = (Button)findViewById(R.id.bt_continue);
    BUTTON_Game_HELP = (Button)findViewById(R.id.bt_help);
    BUTTON_RANK = (Button)findViewById(R.id.bt_rank);
    BUTTON_PREVIOUS = (Button)findViewById(R.id.bt_pre);
    BUTTON_NEXT = (Button)findViewById(R.id.bt_next);
    BUTTON_EXIT = (Button)findViewById(R.id.bt_exit);
    TEXTVIEW_LEVEL = (TextView)findViewById(R.id.tv_speed);
    CHECKBOX_VOICE = (CheckBox)findViewById(R.id.cb_voice);
    /*根据 XML 定义的标签创建 BUTTON_NEW_GAME、BUTTON_CONTINUE、BUTTON_Game_HELP、
    BUTTON_RATING、BUTTON_PREVIOUS、BUTTON_NEXT、BUTTON_EXIT、TEXTVIEW_LEVEL
    和 CHECKBOX_VOICE 控件对象*/
    Toast.makeText(Tetris.this, "Finish to creat component", Toast.LENGTH_LONG).show();
    //在该活动上（Tetris）弹出 Toast 消息
}
/*使用 creatComponent 方法创建控件，执行完之后弹出"Finish to creat component"的 Toast 消息*/

```



```
private void registerListener()
{
    BUTTON_NEW_GAME.setOnClickListener(BUTTON_NEW_GAMEListener);
    //注册 BUTTON_NEW_GAME 控件的单击事件监听器
    BUTTON_CONTINUE.setOnClickListener(BUTTON_CONTINUEListener);
    //注册 BUTTON_CONTINUE 控件的单击事件监听器
    BUTTON_Game_HELP.setOnClickListener(BUTTON_Game_HELPListener);
    //注册 BUTTON_Game_HELP 控件的单击事件监听器
    BUTTON_RANK.setOnClickListener(BUTTON_RANKListener);
    //注册 BUTTON_RATING 控件的单击事件监听器
    BUTTON_PREVIOUS.setOnClickListener(BUTTON_PREVIOUSListener);
    //注册 BUTTON_PREVIOUS 控件的单击事件监听器
    BUTTON_NEXT.setOnClickListener(BUTTON_NEXTListener);
    //注册 BUTTON_NEXT 控件的单击事件监听器
    BUTTON_EXIT.setOnClickListener(BUTTON_EXITListener);
    //注册 BUTTON_EXIT 控件的单击事件监听器
    Toast.makeText(Tetris.this, "Finish to register listener", Toast.LENGTH_LONG).show();
    //在该活动上 (Tetris) 弹出 Toast 消息
}
/*使用 creatComponent 方法注册控件的监听器, 执行完之后弹出"Finish to register listener"的 Toast 消息*/

private void createIntent()
{
    NEW_Game_Intent = new Intent(Tetris.this, NewTetrisGame.class); //创建意图 NEW_Game_Intent
    RANK_Game_Intent = new Intent(Tetris.this, RankTetrisGame.class); //创建意图 RANK_Game_Intent
    CONTINUE_Game_Intent = new Intent(Tetris.this, NewTetrisGame.class);
    //创建意图 CONTINUE_Game_Intent
    HELP_Game_Intent = new Intent(Tetris.this, TetrisGameHelp.class); //创建意图 HELP_Game_Intent
    Toast.makeText(Tetris.this, "Create intent", Toast.LENGTH_LONG).show(); //在该活动上弹出 Toast 消息
}
/*使用 createIntent 方法创建用户活动之间转换的 Intent*/

private Button.OnClickListener BUTTON_NEW_GAMEListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {
        NEW_Game_Intent.setFlags(IS_NEW_GAME); //设置新游戏标志
        NEW_Game_Intent.putExtra(GameVoice, CHECKBOX_VOICE.isChecked());
    }
}
```



```

        //BUTTON_CONTINUEListener 携带 GameVoice 信息
        NEW_Game_Intent.putExtra(GameLevel,mylevel);
        //BUTTON_CONTINUEListener 携带 GameLevel 信息
        startActivity(NEW_Game_Intent); //执行 NewTetrisGame 的活动
        return;
    }
};
/*BUTTON_NEW_GAME 控件的单击事件监听器。当单击 BUTTON_NEW_GAME 时, 该类的 onClick 被
调用*/

private Button.OnClickListener BUTTON_CONTINUEListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {
        CONTINUE_Game_Intent.setFlags(IS_CONTINUE); //设置继续上次游戏的标志
        CONTINUE_Game_Intent.putExtra(GameVoice,CHECKBOX_VOICE.isChecked());
        //BUTTON_CONTINUEListener 携带 GameVoice 信息
        startActivity(CONTINUE_Game_Intent); //执行 NewTetrisGame 的活动
        return;
    }
};
/*BUTTON_CONTINUE 控件的单击事件监听器。当单击 BUTTON_CONTINUE 时, 该类的 onClick 被调
用*/

private Button.OnClickListener BUTTON_Game_HELPListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {
        startActivity(HELP_Game_Intent); //执行 TetrisGameHelp 的活动
        return;
    }
};
/*BUTTON_Game_HELP 控件的单击事件监听器。当单击 BUTTON_CONTINUE 时, 该类的 onClick 被调
用*/

private Button.OnClickListener BUTTON_RANKListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {

```



```

        startActivity(RANK_Game_Intent); //执行 RankTetrisGame 的活动
        return;
    }
};

/*BUTTON_RANK 控件的单击事件监听器。当单击 BUTTON_RANK 时, 该类的 onClick 被调用*/

private Button.OnClickListener BUTTON_PREVIOUSListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {
        BUTTON_PREVIOUS.setBackgroundColor(0xffc0c0c0); //设置背景颜色
        String string = TEXTVIEW_LEVEL.getText().toString(); //获取文本视图的等级
        int level = Integer.parseInt(string);
        --level;
        level = (level-1+TetrisView.MAX_LEVEL) % TetrisView.MAX_LEVEL;
        ++level;
        string = String.valueOf(level);
        TEXTVIEW_LEVEL.setText(string);
        mylevel = level;
        BUTTON_PREVIOUS.setBackgroundColor(0x80cfcfcf);
        return;
    }
};

/*BUTTON_PREVIOUS 控件的单击事件监听器。当单击 BUTTON_PREVIOUS 时, 该类的 onClick 被调用*/

private Button.OnClickListener BUTTON_NEXTListener = new Button.OnClickListener()
{
    @Override
    public void onClick(View v) {
        BUTTON_NEXT.setBackgroundColor(0xffc0c0c0);
        String string = TEXTVIEW_LEVEL.getText().toString();
        int level = Integer.parseInt(string);
        --level;
        level = (level+1) % TetrisView.MAX_LEVEL;
        ++level;
        string = String.valueOf(level);
        TEXTVIEW_LEVEL.setText(string);
        mylevel = level;
    }
};

```



```

        BUTTON_NEXT.setBackgroundColor(0x80cfcfcf);
        return;
    }
};

/*BUTTON_NEXT 控件的单击事件监听器。当单击 BUTTON_NEXT 时, 该类的 onClick 被调用*/

private Button.OnClickListener BUTTON_EXITListener = new Button.OnClickListener()
{

    @Override
    public void onClick(View v) {
        Tetris.this.finish();
    }
};

/*BUTTON_EXIT 控件的单击事件监听器。当单击 BUTTON_EXIT 时, 该类的 onClick 被调用*/

private void saveTetrisGameSettings()
{
    Editor editor = sharedPreferences.edit(); //根据 SharedPreferences 获取编辑器对象
    editor.putInt(GameLevel, mylevel); //将 mylevel 的键值对放入编辑器中
    editor.putBoolean(GameVoice, CHECKBOX_VOICE.isChecked());
    //将复选框状态的键值对放入编辑器中
    editor.commit(); //将 mylevel 和 CHECKBOX_VOICE.isChecked()的键值对存储到 sharedPreferences 中
}

/*saveTetrisGameSettings 方法使用 Preference 保存该游戏设置的参数*/

private void loadLastSettings()
{
    SharedPreferences sharedPreferences = getSharedPreferences(Settings, 0);
    /*调用 getSharedPreferences 生成 SharedPreferences 对象
    *调用 getSharedPreferences 时, 需要指定两个参数: 存储键值对的 XML 文件名和打开模式*/
    mylevel = sharedPreferences.getInt(GameLevel, 1); //获取 sharedPreferences 中 GameLevel 对应的值
    boolean hasVoice = sharedPreferences.getBoolean(GameVoice, true);
    //获取 sharedPreferences 中 GameVoice 对应的值
    TEXTVIEW_LEVEL.setText(String.valueOf(mylevel));
    //根据获取的 GameLevel, 设置界面显示难度的文本视图
    CHECKBOX_VOICE.setChecked(hasVoice); //根据获取的 GameVoice, 设置复选框的状态
}

/*loadLastSettings 方法根据上一次存储的游戏设置参数, 将游戏设置回滚到上一次的设置*/

```



```
public void onStop()
{
    super.onStop();           //执行父类的 onStop 方法
    saveTetrisGameSettings(); //保存该游戏设置的参数
}
/*Activity 停止时及游戏结束时, onStop 方法被调用*/
}
```

当单击 `BUTTON_NEW_GAME` 控件或者 `BUTTON_CONTINUE` 控件时, `NewTetrisGame` class 活动被执行。`NewTetrisGame` 是单击“新游戏”按钮和“继续上次游戏”按钮被调用的类, `NewTetrisGame` 也是一个活动类。

`NewTetrisGame` 首先从 `onCreate` 方法开始执行。需要注意, 在 `onCreate` 方法中要使用 `super.onCreate` 方法执行父类的 `onCreate` 方法。然后使用 `requestWindowFeature (Window.FEATURE_NO_TITLE)` 方法设置该 `Activity` 的窗口没有标题, 实现全屏的效果。`requestWindowFeature` 方法用于设置窗口属性, 除了 `FEATURE_NO_TITLE` 之外还有其他属性: `DEFAULT_FEATURES`、`FEATURE_CONTEXT_MENU`、`FEATURE_CUSTOM_TITLE`、`FEATURE_INDETERMINATE_PROGRESS`、`FEATURE_LEFT_ICON`、`FEATURE_NO_TITLE`、`FEATURE_OPTIONS_PANEL`、`FEATURE_PROGRESS`、`FEATURE_RIGHT_ICON`。

最后使用自定义的 `inti_NewTetrisGame` 方法设置游戏参数, 创建用于控制和管理俄罗斯方块的视图 `TetrisView`, 并创建意图对象 `myintent`。使用该意图对象获取 `flag` 值, `flag` 值在主程序中如下定义:

```
public static final int IS_NEW_GAME = 0; //定义整型常量 IS_NEW_GAME, 用于标识是否是新游戏
public static final int IS_CONTINUE = 1; //定义整型常量 IS_CONTINUE, 用于标识是否是继续上次游戏
```

若单击“继续上次游戏”按钮, `flag` 的值为 1 (`IS_CONTINUE`), 程序使用 `restoreGame` 从上一次的游戏开始执行。若单击“新游戏”按钮, 则程序不从上一次的游戏开始执行。程序实现如下:

```
import android.app.Activity; //导入 Activity 类
import android.content.Intent; //导入 Intent 类
import android.os.Bundle; //导入 Bundle 类
import android.view.Window; //导入窗口类
/*NewTetrisGame 是单击“新游戏”按钮和“继续上次游戏”按钮被调用的类*/

public class NewTetrisGame extends Activity {

    TetrisView tetrisView = null;

    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState); //执行父类的 onCreate 方法
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        //使用 requestWindowFeature 方法设置该 Activity 的窗口没有标题, 类似于全屏的效果
```



```

        inti_NewTetrisGame(); //初始化
    }

    /*onCreate 方法是 NewTetrisGame 的入口，程序从 onCreate 开始执行*/

    private void inti_NewTetrisGame()
    {
        tetrisView = new TetrisView(this); //TetrisView 提供了用于控制和管理俄罗斯方块的视图
        Intent myintent = getIntent(); //创建意图对象 myintent
        int level = myintent.getIntExtra(Tetris.GameLevel,1); //获取设置的难度
        tetrisView.setGameLevel(level); //根据获取的 Tetris 的游戏设置，设置 tetrisView 难度
        int flag = myintent.getFlags(); //获取标志
        if(flag == Tetris.IS_CONTINUE)
        {
            tetrisView.restoreGame();
        }
        /*判断该 Activity 是否是由“继续上次游戏”按钮触发的。若该 Activity 是由“继续上次游戏”按钮
        触发的，恢复之前保存的游戏*/
        // voice setting influence last game
        boolean gameVoice = myintent.getBooleanExtra(Tetris.GameVoice,true);
        //获取 Tetris 设置的声音参数（是否静音）
        tetrisView.setVoice(gameVoice); //根据 Tetris 设置的声音参数（是否静音），设置游戏是否静音
        setContentView(tetrisView); //使用 tetrisView 设置游戏界面的视图
    }

    public void onPause()
    {
        tetrisView.onPause(); //暂停 tetrisView
        super.onPause();
    }
    /*暂停该 Activity，onPause 方法被调用*/

    public void onResume()
    {
        super.onResume(); //执行父类的 onResume 方法
        tetrisView.onResume(); //暂停 tetrisView
    }
    /*继续该活动时，onResume 方法被调用*/

    public void onStop()
    {

```



```

super.onStop();
tetrisView.saveGame(); //退出时, 保存当前游戏, 如得分、消掉的行数以及当前游戏的界面等
tetrisView.freeResources(); //释放资源
}
/*停止该活动时, onStop 方法被调用*/
}

```

下面是 TetrisView 的实现, TetrisView 定义了四个属性: GAME\_MENU、GAME\_PLAY (玩游戏)、GAME\_PAUSE (暂停游戏) 和 GAME\_OVER (结束游戏)。TetrisView 使用 countScore(int line)方法计算分数和级别, countScore 方法调用了自定义的 int setGamescoreAccordingLine(int line, int gamescore)方法, 根据消去的行数计算总分。参数 line 指定了消去的行数, 参数 gamescore 指定了上次游戏的得分。分数计算规则如下:

- 若同时消去 1 行则加 100 分。
- 若同时消去 2 行则加 400 分。
- 若同时消去 3 行则加 800 分。
- 若同时消去 4 行则加 1400 分。
- 若同时消去 5 行则加 2000 分。
- 若超过 5 行, 则超过的行数每行加 400 分。

然后使用 setGameLevelAccordingScore 方法根据得分计算级别, 参数 score 指定了当前游戏总分。评级的规则如下:

- 若分数 gamescore >= 10000, 则通过 setGameLevel 设置级别为 6。
- 若分数 gamescore >= 8000 && gamescore < 10000, 则通过 setGameLevel 设置级别为 5。
- 若分数 gamescore >= 6000 && gamescore < 8000, 则通过 setGameLevel 设置级别为 4。
- 若分数 gamescore >= 4000 && gamescore < 6000, 则通过 setGameLevel 设置级别为 3。
- 若分数 gamescore >= 2000 && gamescore < 4000, 则通过 setGameLevel 设置级别为 2。
- 否则设置级别为 1。

下面是具体的实现:

```

/*TetrisView 继承 View 类和 Runnable 接口*/
public class TetrisView extends View implements Runnable{
    final int GAME_MENU    = 0;
    final int GAME_PLAY    = 1;
    final int GAME_PAUSE   = 2;
    final int GAME_OVER    = 3;
    /*以上定义了游戏的四种状态*/
    final static int SCREEN_WIDTH = 320;
    final static int SCREEN_HEIGHT = 455;
    public static final int MAX_LEVEL = 6;
    public static final String TAG = "TetrisView";
    public static final String DATAFILE = "save.dt";
    int gameStat = GAME_PLAY;
    int gamescore = 0;
}

```



```

int gamelevel = 1;
int gameDecline = 0;
/*声明游戏的四个属性：GAME_MENU、GAME_PLAY（玩游戏）、GAME_PAUSE（暂停游戏）和
GAME_OVER（结束游戏）*/
boolean IS_Combo = false; //
boolean IS_PAUSED = false;
boolean IS_Voice = true;
long moveDelay = 600; //定义方块运动的延时
long lastMove = 0;
private Context mycontext = null;
private Paint Tertrispaint = new Paint(); //定义画笔对象
RefreshHandler rh = null; //RefreshHandler 是本程序自定义的类
TileView currentTile = null;
TileView nextTile = null;
Court mycourt = null;
ResourceStore resourceStore = null;
MediaPlayer musicPlayer = null;
/*声明 TetrisView 类的属性，并置 currentTile、nextTile、mycourt、resourceStore 和 musicPlayer 为空*/
public TetrisView(Context context) {
    super(context);
    init(context);
    // TODO Auto-generated constructor stub
}
/*TetrisView 的构造函数，该函数使用 init 方法进行初始化*/

protected void init(Context context)
{
    mycontext = context;
    currentTile = new TileView(mycontext);
    nextTile = new TileView(mycontext);
    mycourt = new Court(mycontext);
    rh = new RefreshHandler(this);
    resourceStore = new ResourceStore(mycontext);
    musicPlayer = new MediaPlayer(context);
    setGameLevel(1);
    Tertrispaint.setAntiAlias(true);
    Tertrispaint.setColor(Color.RED); //设置画笔的颜色
    setFocusable(true); //设置焦点
    new Thread(this).start(); //新建线程并启动
}

```



/\*init 方法用于初始化\*/

```
public void logicHandler()
```

```
{
```

```
    switch(gameStat)
```

```
    {
```

```
        case GAME_MENU:
```

```
            gameStat = GAME_PLAY;
```

```
            break;
```

```
        case GAME_PLAY:
```

```
            playTetrisGame();
```

```
            break;
```

```
            //状态为 GAME_PLAY 时, 使用 playTetrisGame 方法进入玩游戏的逻辑
```

```
        case GAME_PAUSE:
```

```
            break;
```

```
        case GAME_OVER:
```

```
            break;
```

```
        default;
```

```
    }
```

```
}
```

/\*logicHandler 方法根据游戏的状态用于逻辑处理\*/

```
public void startGame()
```

```
{
```

```
    gameStat = GAME_PLAY;
```

```
    mycourt.clearCourt();
```

```
    currentTile = new TileView(mycontext); //当前的方块
```

```
    nextTile     = new TileView(mycontext); //下一个方块
```

```
    setGameLevel(1);
```

```
    gamescore = 0;    //设置得分为 0
```

```
    gameDecline = 0;  //设置消去的行数为 0
```

```
    IS_PAUSED = false; //开始游戏时, 设置 IS_PAUSED 标志为假
```

```
    IS_Combo = false;  //开始游戏时, 设置 IS_Combo 标志为假
```

```
    playTetrisGame(); //使用 playTetrisGame 方法进入“玩游戏”的逻辑
```

```
}
```

/\*startGame 方法设置“玩游戏”参数并进入“玩游戏”的逻辑\*/

```
public void playTetrisGame()
```

```
{
```

```
    long current = System.currentTimeMillis(); //获取当前的时间
```



```

if(current - lastMove > moveDelay)
{
    if(IS_PAUSED)
    {
        return; //若停止游戏，则直接返回
    }

    if(IS_Combo)
    {
        mycourt.placeTile(currentTile);
        musicPlayer.playMoveVoice(); //播放方块移动的声音

        if(mycourt.isGameOver() )
        {
            gameStat = GAME_OVER;
            return;
        }
        int line = mycourt.removeLines(); //
        if(line > 0 )
        {
            musicPlayer.playBombVoice(); //若消去行，则播放消去的声音
        }
        gameDecline += line; //计算至今为止已经消去的行
        countScore(line); //根据消去的行计算得分
        currentTile = nextTile; //下一个方块为当前方块
        nextTile = new TileView(mycontext); //重新获取下一个方块
        IS_Combo = false;
    }
    moveDown(); //向下移动

    lastMove = current;
}
}
/*使用 playTetrisGame 方法进入"玩游戏"的逻辑*/

private void countScore(int line)
{
    gamescore = setGamescoreAccordingLine(line, gamescore);
    /*根据消去的行数，计算分数*/
    setGameLevelAccordingScore(gamescore);
}

```



```

        /*根据得分计算级别*/
    }
    /*countScore 方法根据行数计算分数和级别*/

    public int setGamescoreAccordingLine(int line, int gamescore)
    {
        int score = gamescore;
        switch(line)
        {
            case 0:
                break;
            case 1:
                score += 100;
                break;
                //同时消去 1 行则加 100 分
            case 2:
                score += 400;
                break;
                //同时消去 2 行则加 400 分
            case 3:
                score += 1000;
                break;
                //同时消去 3 行则加 800 分
            case 4:
                score += 1400;
                break;
                //同时消去 4 行则加 1400 分
            case 5:
                score += 1400;
                break;
                //同时消去 5 行则加 2000 分
            default:
                score = (score-5)* 400 + 2000;
                ;
                //超过 5 行, 则超过的行数每行加 400 分
        }
        /*根据消去的行数, 计算分数*/
        return score;
    }
    /*setGamescoreAccordingLine 方法根据消去的行数计算分数。参数 line 指定了消去的行数, 参数 gamescore

```



指定了上次游戏的得分\*/

```
public void setGameLevelAccordingScore(int score)
{
    if(gamescore >= 10000)
    {
        setGameLevel(6);
    }
    else if (gamescore >= 8000 && gamescore < 10000)
    {
        setGameLevel(5);
    }
    else if(gamescore >= 6000 && gamescore < 8000)
    {
        setGameLevel(4);
    }
    else if(gamescore >= 4000 && gamescore < 6000)
    {
        setGameLevel(3);
    }
    else if
    (gamescore >= 2000 && gamescore <4000)
    {
        setGameLevel(2);
    }
    else
        setGameLevel(1);
}
```

/\*setGameLevelAccordingScore 方法根据得分计算级别, 参数 score 指定了当前游戏的总分\*/

```
protected void onDraw(Canvas canvas)
```

```
{
    switch(gameStat)
    {
        case GAME_MENU:
            paintMenu(canvas);
            break;
        case GAME_PLAY:
            paintTetrisGame(canvas);
            break;
    }
}
```



```

        case GAME_PAUSE:
            paintGamePause(canvas);
            break;
        case GAME_OVER:
            paintGameOver(canvas);
            break;
        default:
    }
}

public boolean isGameOver()
{
    return mycourt.isGameOver();
}

public boolean onKeyDown(int keyCode,KeyEvent event)
{
    switch(keyCode)
    {
        case KeyEvent.KEYCODE_DPAD_UP:
            if(gameStat == GAME_PLAY)
            {
                if(!IS_PAUSED)
                {
                    rotate();
                    musicPlayer.playMoveVoice();
                }
            }
            else if(gameStat == GAME_PAUSE)
            {
            }
            else if(gameStat == GAME_MENU)
            {
            }
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            if(gameStat == GAME_PLAY)
            {
                if(!IS_PAUSED)

```



```

        {
            moveDown();
            musicPlayer.playMoveVoice();
        }
    }
    else if(gameStat == GAME_PAUSE)
    {
    }
    else if(gameStat == GAME_MENU)
    {

    }
    break;
case KeyEvent.KEYCODE_DPAD_LEFT:
    if(gameStat == GAME_PLAY)
    {
        if(!IS_PAUSED)
        {
            moveLeft();
            musicPlayer.playMoveVoice();
        }
    }
    else if(gameStat == GAME_PAUSE)
    {
    }
    else if(gameStat == GAME_MENU)
    {

    }
    break;
case KeyEvent.KEYCODE_DPAD_RIGHT:
    if(gameStat == GAME_PLAY)
    {
        if(!IS_PAUSED)
        {
            moveRight();
            musicPlayer.playMoveVoice();
        }
    }
    else if(gameStat == GAME_PAUSE)

```



```

    {
    }
    else if(gameStat == GAME_MENU)
    {

    }
    break;
case KeyEvent.KEYCODE_ENTER: ;
case KeyEvent.KEYCODE_DPAD_CENTER:
    if(gameStat == GAME_PLAY)
    {
        if(!IS_PAUSED)
        {
            fastDrop();
            musicPlayer.playMoveVoice();
        }
    }
    else if(gameStat == GAME_PAUSE)
    {
    }
    else if(gameStat == GAME_MENU)
    {
    }
    break;
//
case KeyEvent.KEYCODE_S:
    if(gameStat == GAME_PLAY)
    {
        IS_PAUSED = true;
    }
    else if(gameStat == GAME_PAUSE)
    {
        IS_PAUSED = false;
    }
    else if(gameStat == GAME_MENU)
    {

    }
    break;
case KeyEvent.KEYCODE_SPACE:

```



```

        IS_PAUSED = !IS_PAUSED;
        if(IS_PAUSED)
        {
            rh.pause();
        }
        else
        {
            rh.resume();
        }
        break;

    default: ;
    }
    return super.onKeyDown(keyCode,event);
}

private void rotate()
{
    // check
    if(!IS_Combo)
        currentTile.rotateOnCourt(mycourt);
}

private void moveDown()
{
    if(!IS_Combo)
    {
        if( ! currentTile.moveDownOnCourt(mycourt) )
            IS_Combo = true;
    }
}

private void moveLeft()
{
    if(!IS_Combo)
    {
        currentTile.moveLeftOnCourt(mycourt);
    }
}

```



```

    }

    private void moveRight()
    {
        if(!IS_Combo)
        {
            currentTile.moveRightOnCourt(mycourt);
        }
    }

    private void fastDrop()
    {
        if(!IS_Combo)
        {
            currentTile.fastDropOnCourt(mycourt);
            IS_Combo = true;
        }
    }

    private void paintMenu(Canvas TetrisCanvas)
    {
        DrawTool.paintImage(TetrisCanvas,resourceStore.getMenuBackground(),0,0);
        DrawTool.paintImage(TetrisCanvas,resourceStore.getMenu(),0,SCREEN_HEIGHT/2
resourceStore.getMenu().getHeight()/2 );
    }

    private void paintTetrisGame(Canvas TetrisCanvas)
    {
        mycourt.paintCourt(TetrisCanvas); //绘制游戏背景
        currentTile.paintTile(TetrisCanvas); //绘制当前方块
        Tertrispaint.setTextSize(20); //设置字体
        paintTetrisNextTile(TetrisCanvas); //绘制下一个方块
        paintGameLevel(TetrisCanvas); //绘制游戏等级
        paintGameScore(TetrisCanvas); //绘制游戏得分
        paintGameDecline(TetrisCanvas); //绘制消去的行数
    }
    /*使用 paintTetrisNextTile 方法绘制下一个方块。注意，这里绘制的方块是个图像，因此使用 paintImage

```



绘制\*/

```
private void paintTetrisNextTile(Canvas TetrisCanvas)
{
    int i,j;
    for(i = 0;i<4;i++)
    {
        for(j = 0;j<4;j++)
        {
            if(nextTile.mTile[i][j] != 0)
            {
                int title_x = (int)(Court.BEGIN_DRAW_X+getBlockDistance(Court.COURT_WIDTH)
                + getBlockDistance((float) (i+0.5)) );
                //设置方块的 x 坐标
                int title_y = (int)( getBlockDistance((float)(j+0.5))); //设置方块的 y 坐标
                DrawTool.drawImage(TetrisCanvas,resourceStore.getBlock(nextTile.getColor()-1), itle_x,
                title_y );
                //使用画笔在指定的位置绘制图像
            }
        }
    }
}
```

/\*使用 paintTetrisNextTile 方法绘制下一个方块。注意，这里绘制的方块是个图像，因此使用 paintImage  
绘制\*/

```
private void paintGameLevel(Canvas TetrisCanvas)
{
    String GameLevel_title = "游戏等级:";
    float GameLevel_title_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
    //设置"游戏等级"文本的 x 坐标
    float GameLevel_title_y = getBlockDistance(9); //设置"游戏等级"文本的 y 坐标
    String GameLevel = String.valueOf(gamelevel);
    float GameLevel_x = getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
    //设置游戏等级数值的 x 坐标
    float GameLevel_y = getBlockDistance(11); //设置游戏等级数值的 y 坐标

    Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
    TetrisCanvas.drawText(GameLevel_title,GameLevel_title_x, GameLevel_title_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
    Tertrispaint.setColor(Color.RED); //设置画笔颜色
```



```

TetrisCanvas.drawText(GameLevel,GameLevel_x, GameLevel_y,Tertrispaint);
//使用画笔在指定的位置绘制文本
}
/*使用 paintGameLevel 方法绘制"游戏等级"文本以及游戏等级数值*/

private void paintGameScore(Canvas TetrisCanvas)
{
    String score_title = "游戏得分:";
    float score_title_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
    //设置"游戏得分"文本的 x 坐标
    float score_title_y = getBlockDistance(13); //设置"游戏得分"文本的 y 坐标
    String score = String.valueOf(gamescore); //将分数转换成字符串
    float score_x = getBlockDistance(Court.COURT_WIDTH)+ 2*getRightMarginToCourt();
    //设置游戏分数 x 坐标
    float score_y = getBlockDistance(15); //设置游戏分数的 y 坐标
    Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
    TetrisCanvas.drawText(score_title,score_title_x, score_title_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
    Tertrispaint.setColor(Color.RED); //设置画笔颜色
    TetrisCanvas.drawText(score,score_x, score_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
}
/*使用 paintGameDecline 方法绘制"游戏得分"文本以及游戏分数*/

private void paintGameDecline(Canvas TetrisCanvas)
{
    String decline_titr = "消去行数:";
    float decline_x = getBlockDistance(Court.COURT_WIDTH)+getRightMarginToCourt();
    //设置"消去行数"文本的 x 坐标
    float decline_y = getBlockDistance(17); //设置"消去行数"文本的 y 坐标
    String declinenum = String.valueOf(gameDecline);
    float declinenum_x = getBlockDistance(Court.COURT_WIDTH)+2*getRightMarginToCourt();
    //设置消去行数数值的 x 坐标
    float declinenum_y = getBlockDistance(19); //设置消去行数数值的 y 坐标

    Tertrispaint.setColor(Color.BLUE); //设置画笔颜色
    TetrisCanvas.drawText(decline_titr,decline_x, decline_y,Tertrispaint);
    //使用画笔在指定的位置绘制文本
    Tertrispaint.setColor(Color.RED); //设置画笔颜色
    TetrisCanvas.drawText(declinenum,declinenum_x, declinenum_y,Tertrispaint);
}

```



```

        //使用画笔在指定的位置绘制文本
    }
    /*使用 paintGameDecline 方法绘制"消去行数"文本以及消去行数数值*/

    private float getBlockDistance(float blockNum)
    {
        return blockNum * Court.BLOCK_WIDTH;
    }

    private float getRightMarginToCourt()
    {
        return (float)10.0;
    }

    private void paintGamePause(Canvas canvas)
    {
        //TODO
    }
    /*paintGameOver 方法在游戏暂停时被调用*/

    private void paintGameOver(Canvas TetrisCanavs)
    {
        int textsize = 40;
        boolean AntiAlias = true;
        String Gameover = "Game Over";
        float Gameover_x = getBlockDistance(1); //设置"Game Over"文本的 x 坐标
        float Gameover_y = getBlockDistance(Court.COURT_HEIGHT/2-2);
        //设置"Game Over"文本的 y 坐标
        Paint mypaint = new Paint();//新建画图对象

        paintTetrisGame(TetrisCanavs); //生成游戏界面
        mypaint.setTextSize(textsize); //设置字体大小
        mypaint.setAntiAlias(AntiAlias); //设置模糊边界
        mypaint.setARGB(0xe0,0xff,0x00,0x00); //设置颜色
        TetrisCanavs.drawText(Gameover,Gameover_x,Gameover_y,mypaint); //绘制文本
    }
    /*paintGameOver 方法在游戏结束时被调用*/

    @Override
    public void run() {

```



```
// TODO Auto-generated method stub
while(!Thread.currentThread().isInterrupted() )
{
    Message ms = new Message();
    ms.what = RefreshHandler.MESSAGE_REFRESH;
    this.rh.sendMessage(ms);
    try
    {
        Thread.sleep(/*RefreshHandler.DELAY_MILLIS*/moveDelay);
    }
    catch(InterruptedException e)
    {
        Thread.currentThread().interrupt();
    }
}

}

public void setGameLevel(int level)
{
    gamelevel = level;
    moveDelay = (long) (600*(1.0 - (double)gamelevel / 7.0 ) );
}
/*使用 setGameLevel 方法设置游戏等级*/

public void setVoice(boolean isVoice)
{
    IS_Voice = isVoice;
    musicPlayer.setMute(!IS_Voice);
}
/*使用 setVoice 方法设置声音*/

public void restoreGame()
{
    Properties pro = new Properties();
    try
    {
        FileInputStream in = mycontext.openFileInput(DATAFILE);
```



```

        pro.load(in);
        in.close();
    }
    catch(IOException e)
    {
        Log.i(TAG,"file open failed in restoreGame()");
        return;
    }

    gameStat = Integer.valueOf(pro.get("gamestate").toString() );
    gamelevel = Integer.valueOf(pro.get("speed").toString() );
    setGameLevel(gamelevel);
    gamescore = Integer.valueOf(pro.get("score").toString() );
    gameDecline = Integer.valueOf(pro.get("deLine").toString() );
    IS_Voice = Boolean.valueOf(pro.get("isVoice").toString() );
    IS_Combo = Boolean.valueOf(pro.get("isCombo").toString() );
    IS_PAUSED = Boolean.valueOf(pro.get("isPaused").toString() );

    restoreCourt(pro);
    restoreTile(pro,currentTile);
    restoreTile(pro,nextTile);
}
/*使用 restoreGame 方法恢复上次保存的游戏*/

```

```

private void restoreCourt(Properties pro)
{
    int[][] matrix = mycourt.getMatrix();
    int i,j;
    for(i = 0;i<Court.COURT_WIDTH;i++)
    {
        for(j = 0;j<Court.COURT_HEIGHT;j++)
        {
            matrix[i][j] = Integer.valueOf(pro.get("courtMatrix"+i+j).toString() );
        }
    }
}
/*使用 restoreCourt 方法恢复上次保存的游戏界面*/

```

```

private void restoreTile(Properties pro,TileView tile)
{

```



```

int[][] matrix = tile.getMatrix();
int i,j;
for(i = 0;i<4;i++)
{
    for(j = 0;j<4;j++)
    {
        matrix[i][j] = Integer.valueOf(pro.get("tileMatrix"+i+j).toString() );
    }
}
tile.setColor(Integer.valueOf(pro.get("tileColor").toString() ));
tile.setShape(Integer.valueOf(pro.get("tileShape").toString() ));
tile.setOffsetX(Integer.valueOf(pro.get("tileOffsetX").toString() ));
tile.setOffsetY(Integer.valueOf(pro.get("tileOffsetY").toString() ));
}
/*使用 restoreTile 方法恢复上次保存的方块*/

public void saveGame()
{
    Properties pro = new Properties();

    pro.put("gamestate",String.valueOf(gameStat));
    pro.put("speed",String.valueOf(gamelevel));
    pro.put("score",String.valueOf(gamescore));
    pro.put("deLine",String.valueOf(gameDecline));
    Boolean b = new Boolean(IS_Voice);
    pro.put("isVoice",b.toString());
    b = new Boolean(IS_Combo);
    pro.put("isCombo",b.toString());
    b = new Boolean(IS_PAUSED);
    pro.put("isPaused",b.toString());

    saveCourt(pro);
    saveTile(pro,currentTile);
    saveTile(pro,nextTile);

    try
    {
        FileOutputStream stream = mycontext.openFileOutput(DATAFILE,Context.MODE_WORLD_WRITEABLE);
        pro.store(stream,"");
        stream.close();
    }
}

```



```

    }
    catch(IOException e)
    {
        Log.i(TAG,"ioexception in saveGame()");
        return;
    }
}

/*使用 saveGame 方法保存游戏*/

private void saveCourt(Properties pro)
{
    int[][] court = mycourt.getMatrix();
    int i,j;
    for(i = 0;i<Court.COURT_WIDTH;i++)
    {
        for(j = 0;j<Court.COURT_HEIGHT;j++)
        {
            pro.put("courtMatrix"+i+j,String.valueOf(court[i][j]) );
        }
    }
}

/*使用 saveCourt 方法保存游戏界面*/

private void saveTile(Properties pro,TileView tile)
{
    int[][] matrix = tile.getMatrix();
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j = 0;j<4;j++)
        {
            pro.put("tileMatrix"+i+j,String.valueOf(matrix[i][j]) );
        }
    }
    pro.put("tileColor",String.valueOf(tile.getColor() ) );
    pro.put("tileShape",String.valueOf(tile.getShape() ) );
    pro.put("tileOffsetX",String.valueOf(tile.getOffsetX() ) );
    pro.put("tileOffsetY",String.valueOf(tile.getOffsetY() ) );
}

```



```

    }
    /*使用 saveCourt 方法保存游戏方块*/

    public void onPause() {
        rh.pause();
        IS_PAUSED = true;
    }
    /*onPause 方法在活动停止时被调用*/

    public void onResume() {
        rh.resume();
        IS_PAUSED = false;
    }
    /*onResume 方法在活动停止后又继续时被调用*/

    public void freeResources()
    {
        musicPlayer.free();
    }
    /*freeResources 活动使用的音乐播放器的资源*/

```

可通过单击“排名”按钮查看用户的成绩以及排名等信息, 这些信息是存储在数据库中的。下面是本实例的数据操作的实现:

```

    private static final String TAG = "RankDatabase";
    private static final String DB_NAME = "rank.db";
    private static final String DB_TABLE = "table1";
    private static final int DB_VERSION = 1;

    private static final String KEY_ID = "_id";
    public static final String KEY_RANK = "rank";
    public static final String KEY_SCORE = "score";
    public static final String KEY_NAME = "name";

    private static final String DB_CREATE = "CREATE TABLE "
        + DB_TABLE + " ("
        + KEY_ID + " INTEGER PRIMARY_KEY,"
        + KEY_RANK + " INTEGER,"
        + KEY_SCORE + " INTEGER"
        + KEY_NAME + " TEXT)";

```



```
private Context mContext = null;
private SQLiteDatabase mDatabase = null;
private DatabaseHelper mHelper = null;

public void RankDatabase(Context context)
{
    mContext = context;
}

private static class DatabaseHelper extends SQLiteOpenHelper
{
    public DatabaseHelper(Context context)
    {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(DB_CREATE); //使用 execSQL 创建 DB
    }

    public void open()
    {
        mHelper = new DatabaseHelper(mContext);
        mDatabase = mHelper.getWritableDatabase();
    }

    public void close()
    {
        mHelper.close();
    }
}
```

除此之外，还需要实现根据不同的动作播放音乐的功能：

```
private MediaPlayer paly1 = null;
private MediaPlayer play2 = null;
private boolean IS_MUTE = false;
public MusicPlayer(Context context)
```



```
{
    paly1 = MediaPlayer.create(context,R.raw.move);
    play2 = MediaPlayer.create(context,R.raw.bomb);
}
/*构造函数使用 MediaPlayer 创建两个媒体*/

public void playMoveVoice()
{
    if(this.IS_MUTE)
        return; //若静音, 则返回
    paly1.start();
}
/*使用 playMoveVoice 播放移动方块的声音*/

public void playBombVoice()
{
    if(this.IS_MUTE)
        return; //若静音, 则返回
    play2.start();
}
/*使用 playBombVoice 播放成功消去行的声音*/

public void setMute(boolean IS_MUTE)
{
    this.IS_MUTE = IS_MUTE;
}
/*setMute 设置静音*/

public void free()
{
    paly1.release();
    play2.release();
}
```

## 13.4 俄罗斯方块游戏演示

通过 Eclipse 启动该游戏, 程序首先初始化控件和参数 (如图 13-7 所示), 并且注册控件的监听器 (如图 13-8 所示)。





图 13-7 游戏初始化



图 13-8 注册控件监听器

与用户交互的主界面中包括“继续上次游戏”按钮、“新游戏”按钮、“帮助”按钮、“排行榜”按钮、“增加难度”按钮、“减小难度”按钮、“声音开关”复选框和“退出”按钮。这些控件分别注册了监听器，当单击相应的控件时相应的方法会被调用。单击一次“增加难度”按钮，文本框视图显示的难度会从初始的 1 增加到 2。图 13-9 显示了单击“增加难度”按钮后的运行结果。

单击“帮助”按钮，程序从主界面转到帮助界面。图 13-10 显示了帮助界面。





图 13-9 单击“增加难度”按钮

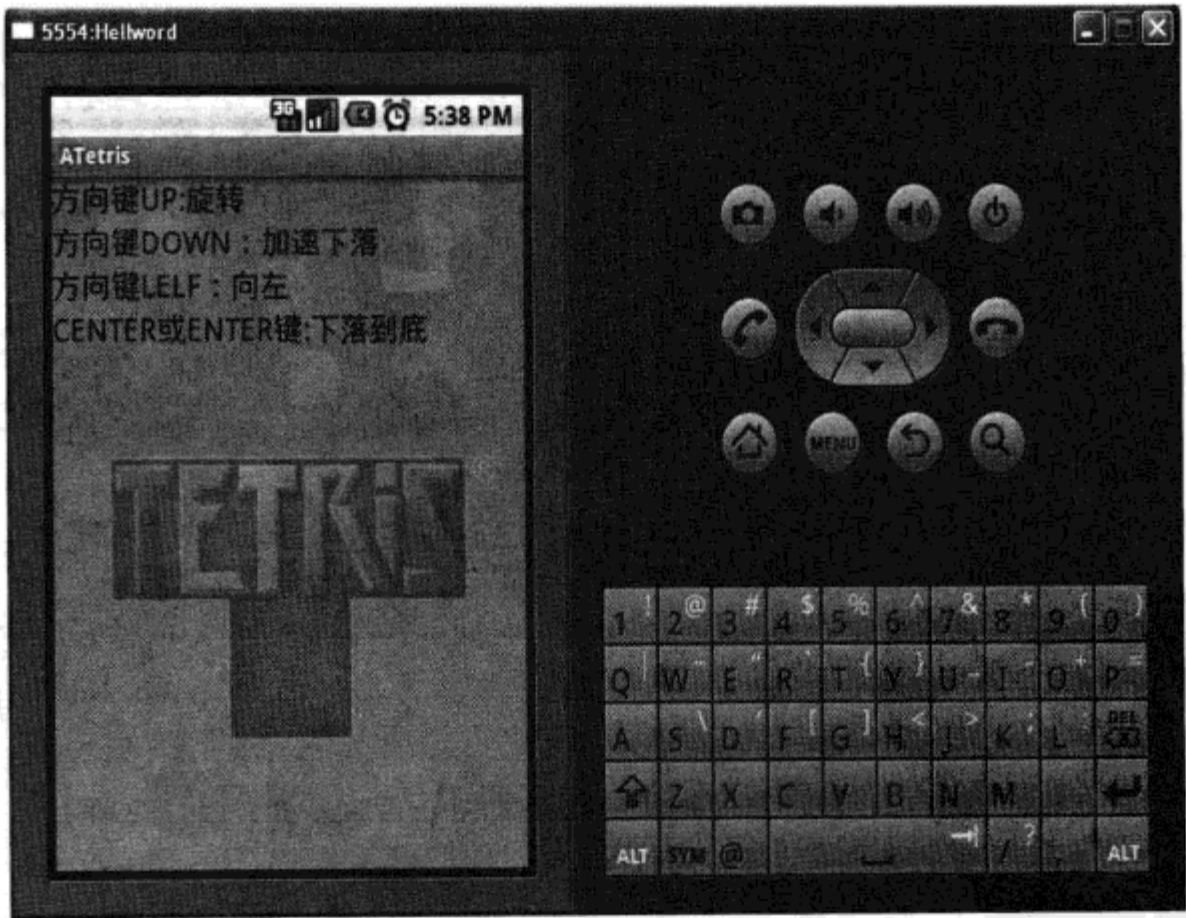


图 13-10 帮助界面

单击“新游戏”按钮，进入了游戏处理界面（如图 13-11 所示）。游戏处理界面包含游戏等级、游戏得分、消去行数以及下一个方块的图标。游戏等级显示为 2，这个数值是在主界面中设置的。



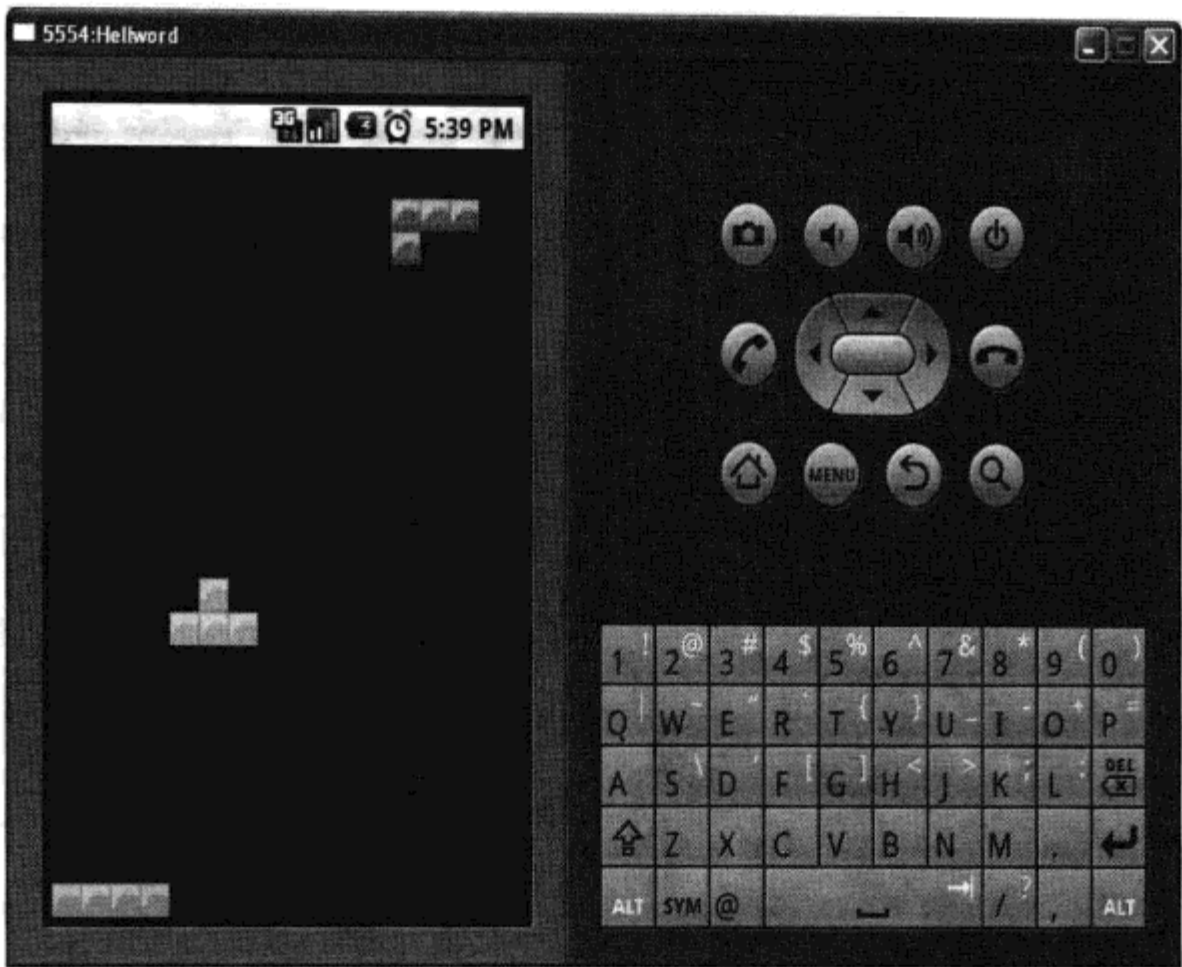


图 13-11 游戏处理界面

可通过“上箭头”按键旋转方块方向、“左箭头”按钮左移方块、“右箭头”按钮右移方块。图 13-12 是旋转前的游戏界面。

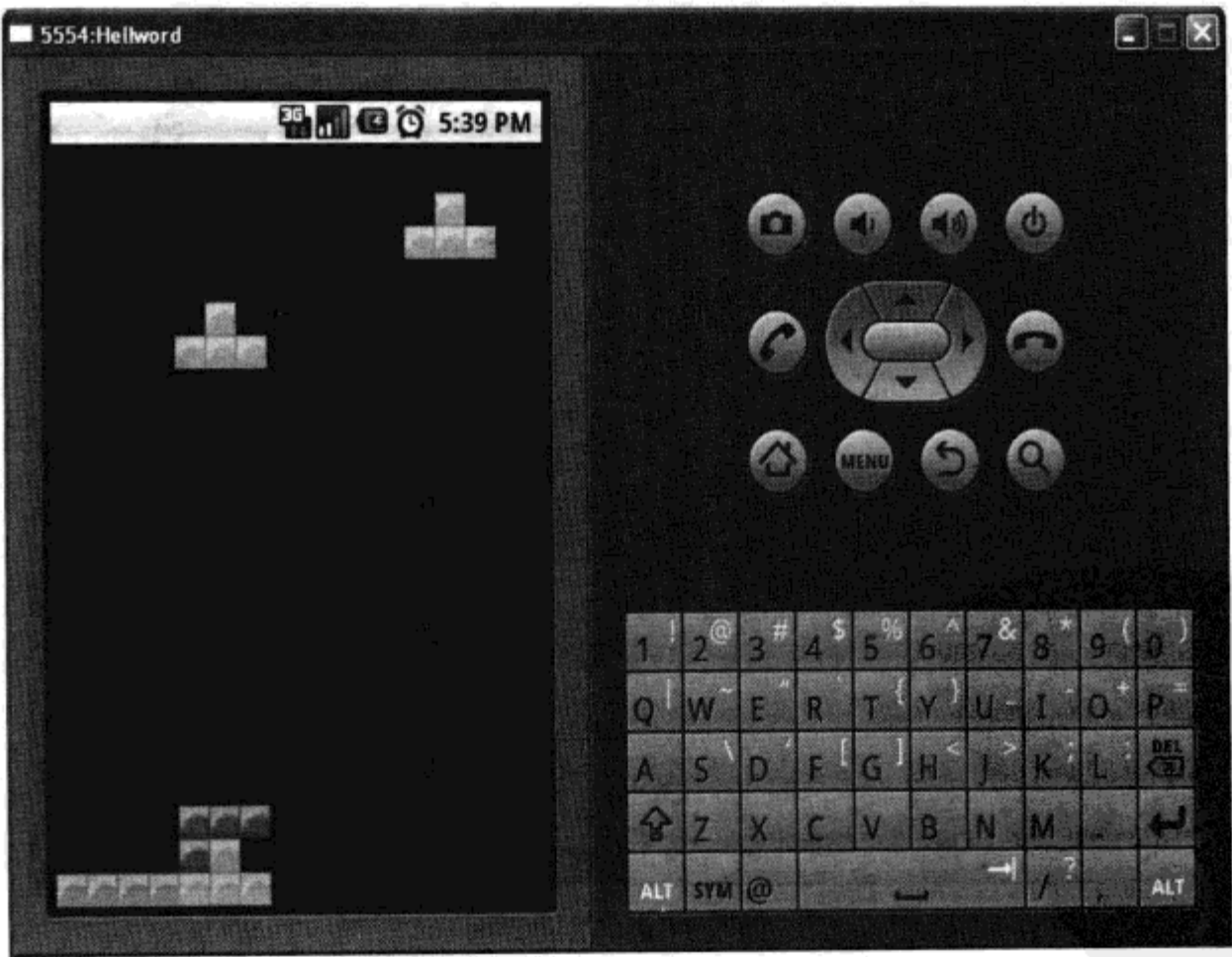


图 13-12 旋转前的游戏界面

按下“上箭头”按键，界面中的方块被旋转。图 13-13 是旋转后的游戏界面。消去一行时，游戏得分和消去行数被相应更新。图 13-14 显示了得分变化的情况。



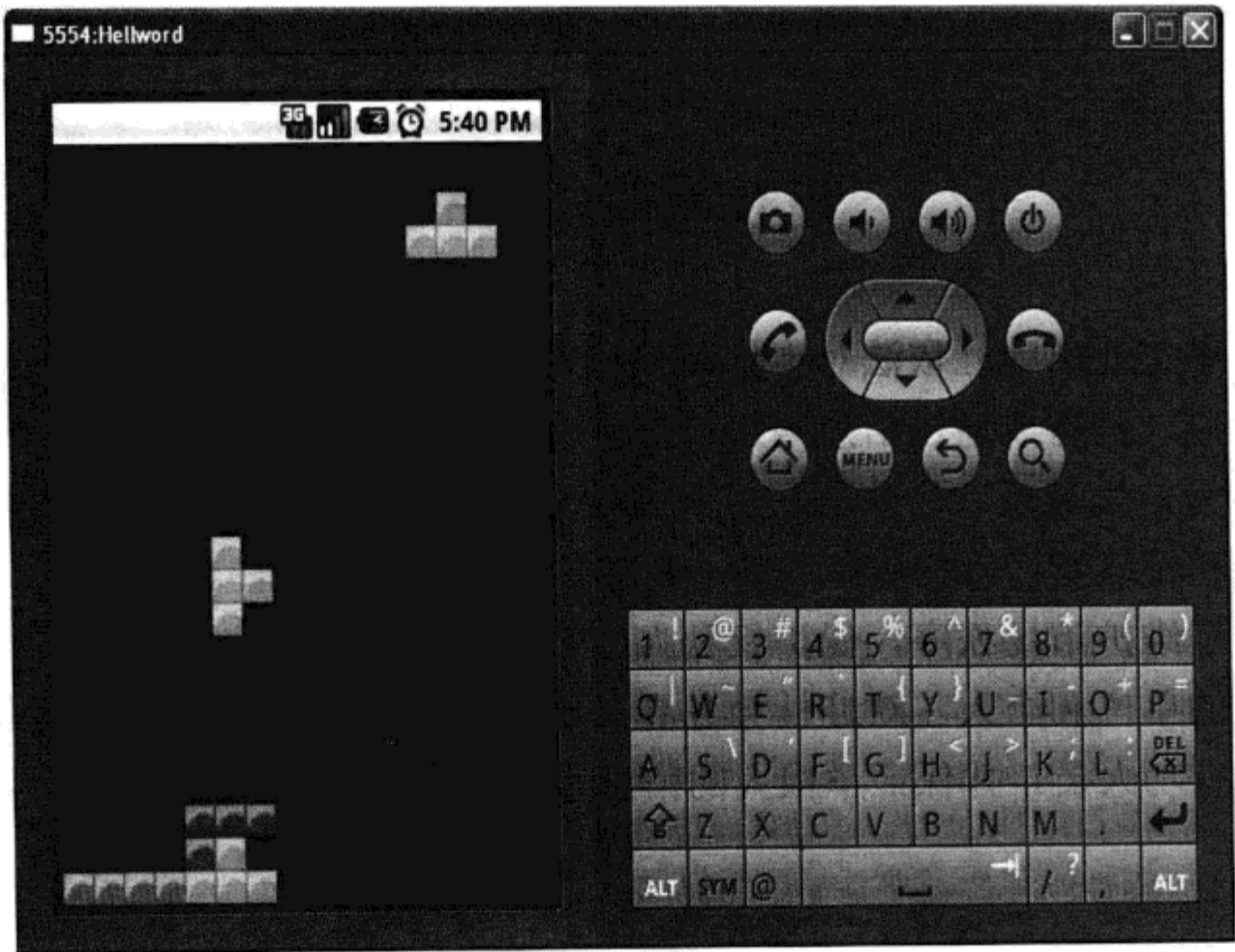


图 13-13 旋转后的游戏界面

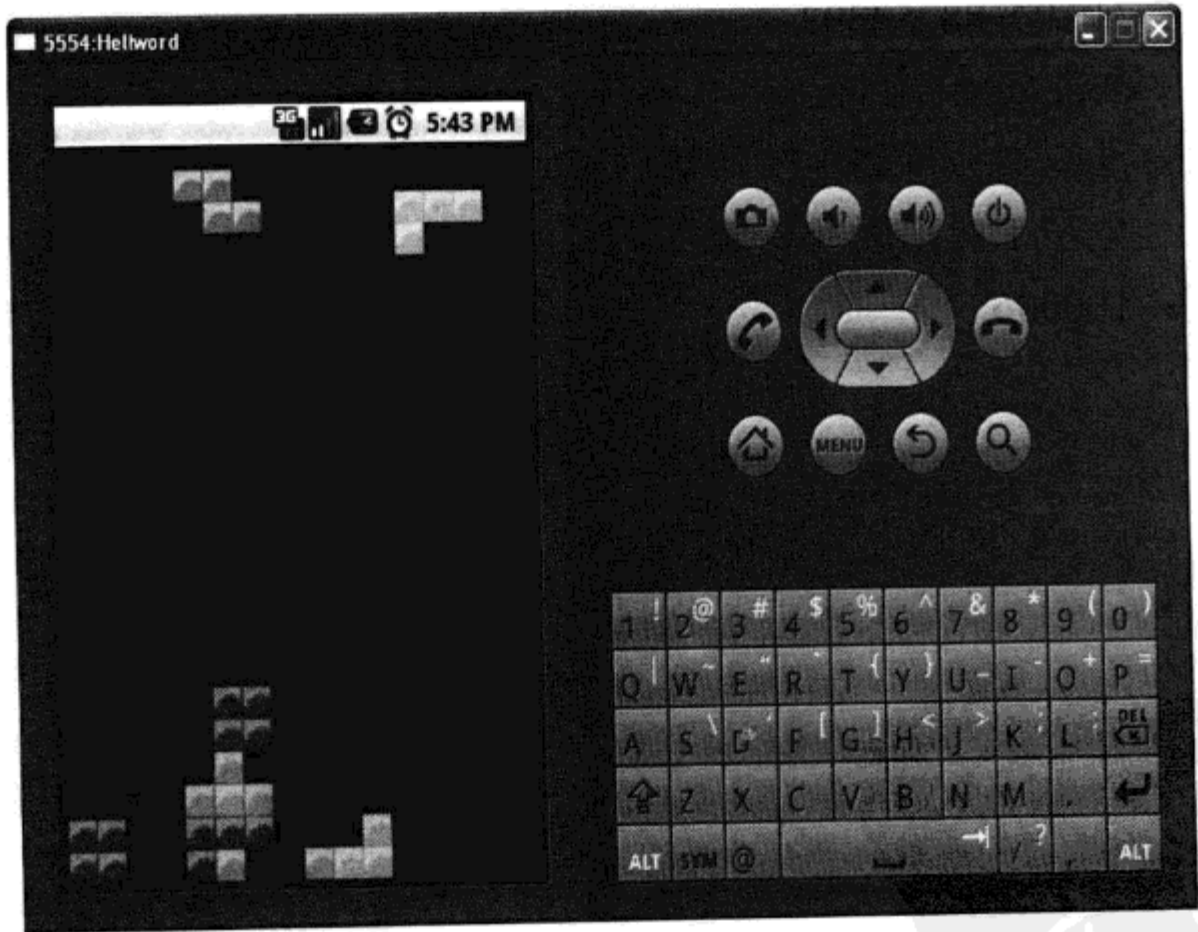


图 13-14 得分变化情况



## 反侵权盗版声明

电子工业出版社依法对本作品享有专有出版权。任何未经权利人书面许可，复制、销售或通过信息网络传播本作品的行为；歪曲、篡改、剽窃本作品的行为，均违反《中华人民共和国著作权法》，其行为人应承担相应的民事责任和行政责任，构成犯罪的，将被依法追究刑事责任。

为了维护市场秩序，保护权利人的合法权益，我社将依法查处和打击侵权盗版的单位和个人。欢迎社会各界人士积极举报侵权盗版行为，本社将奖励举报有功人员，并保证举报人的信息不被泄露。

举报电话：(010) 88254396；(010) 88258888

传 真：(010) 88254397

E-mail: dbqq@phei.com.cn

通信地址：北京市万寿路 173 信箱

电子工业出版社总编办公室

邮 编：100036

新华书店  
PDG



## Android 从入门到精通

一线资深开发人员倾力打造

本书的主要特点如下:

- 由浅入深, 介绍翔实
- 实例丰富, 经典实用
- 易于学习, 轻松掌握
- 经验与教学有机结合

Android从入门到精通

SAS 9.2从入门到精通

Red Hat Enterprise Linux 6从入门到精通

MATLAB 2010从入门到精通

软件测试从入门到精通

设计模式从入门到精通

C语言从入门到精通

Visual C# 2010从入门到精通

Visual Basic 2010中文版从入门到精通

UML与Rational Rose 2003从入门到精通

Java SE 6从入门到精通

Oracle 11g从入门到精通

SQL从入门到精通

SQL Server 2008中文版从入门到精通

Windows 7中文版从入门到精通

Windows Server 2008从入门到精通

Project 2010中文版从入门到精通

Word 2010中文版从入门到精通

Excel 2010中文版从入门到精通

PowerPoint 2010中文版从入门到精通

Crystal Reports 2008水晶报表从入门到精通



责任编辑: 李红玉  
封面设计: 李娜

