如果您看到本PDF，请发邮件给 21504965@qq.com，让我知道我的付出没有白费。谢谢！

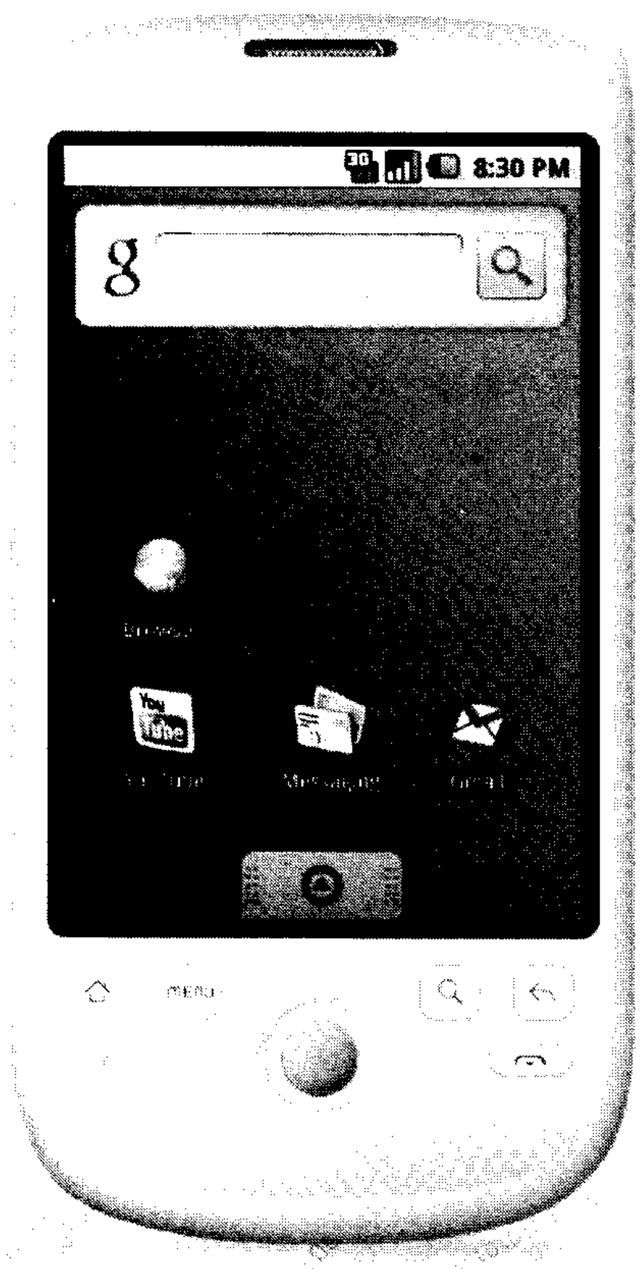
自制GPS群：19504537 （电台（短波）+GPS（GPRS选配）+陀螺仪（MEMS））

本项目征集 志愿者：

1、工程师：硬件/软件/模具（外壳）

2、VC：疑似头脑简单型。本项目暂时看不到盈利的希望。

3、测试者。喜欢户外的同学，测试产品，写测试报告，提供产品的建议和意见。



Android 应用开发详解

郭宏志 编著

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

内 容 简 介

作为一本 Android 应用开发书籍，本书既适合 Android 初学者，也适合具备了一定 Android 开发经验但需要开发案例的高级读者。

本书分为三个部分，共 18 章，由浅入深地详细介绍了 Android 的每个开发细节。

本书基础翔实，实例丰富，案例真实。从基础到案例覆盖了 Android 应用开发的三大领域：基础应用、网络应用和游戏应用。读者所需要学习的，正是本书描述的。

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 应用开发详解 / 郭宏志编著. —北京：电子工业出版社，2010.6
ISBN 978-7-121-10875-4

Ⅰ. ①A…　Ⅱ. ①郭…　Ⅲ. ①移动通信—携带电话机—应用程序—程序设计　Ⅳ. ①TN929.53

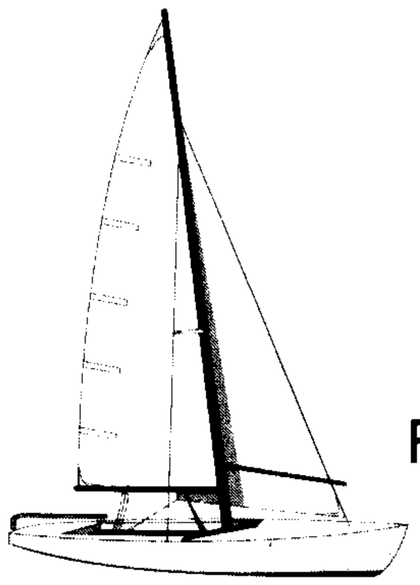
中国版本图书馆 CIP 数据核字（2010）第 087706 号

责任编辑：胡辛征
文字编辑：江　立
印　　刷：北京市天竺颖华印刷厂
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：31.75　字数：755 千字
印　　次：2010 年 6 月第 1 次印刷
印　　数：4000 册　　定价：59.80 元（含光盘 1 张）

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

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

服务热线：（010）88258888。



FOREWORD

前　言

移动互联网时代来临，一个崭新的时代开始了。所谓移动互联网就是将移动通信和互联网整合在一起，使移动设备（包括手机和上网本）可以随时随地地访问互联网资源和应用。互联网时代创造了一个经济神话，也造就了很多时代英雄，他们一个个令人仰慕。试想为数亿的手机用户和数亿的网民建立一个共同的平台，使其应用到企业、商业和和农村之间，又会是怎样一个惊天动地的伟业呢？新时代开始了，你愿意输在起跑线上吗？

智能手机是移动互联网时代一个标志性的客户端工具，它和传统的手机是有区别的，它就像一台“小电脑”，具有独立的操作系统，可以自由安装、卸载软件，具有强大的计算和存储能力，可以通过移动通信网络来实现无线网络接入。智能手机一般具备如下特点：高速度处理芯片、大存储芯片和存储扩展能力、面积大、标准化、可触摸的显示屏、摄像头至少 300 万像素、支持播放式的手机电视、必须支持 GPS 导航、操作系统必须支持新应用的安装等。

互联网的竞争格局基本定型，那么移动互联网时代竞争的焦点在智能手机终端上，软件部分包括智能手机操作系统和应用软件。

目前智能手机操作系统有：诺基亚的 Symbian、微软的 Windows Mobile 和 Google 的 Android 等。

Google 于 1998 年 9 月 7 日创立，经过十几年在搜索引擎方面的精耕细作，成为全球互联网巨头，尤其在地图搜索的应用更是引人注目。Google 于 2007 年 11 月 5 日宣布的基于 Linux 平台的开源手机操作系统，名称为 Android，中文译为“机器人”。这意味着 Google 在移动互联网时代开始抢跑并领跑。

Android 是一个真正意义上的开源智能手机操作系统，该系统由底层的 Linux、

中间层的软件包和上层的 Java 应用程序组合而成。该系统一经推出立即受到了全球移动设备厂商和开发者的热捧。

2008 年 9 月 22 日，美国运营商 T-Mobile USA 在纽约正式发布第一款 Google 手机——T-Mobile G1。该款手机为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，它支持 WCDMA/HSPA 网络，理论下载速率为 7.2Mbps，并支持 Wi-Fi。到了 2010 年 1 月，Google 开始发布自家品牌手机 Nexus One。该款手机使用的操作系统是 Android 2.1。如下左图为 G1，右图为 Nexus One。

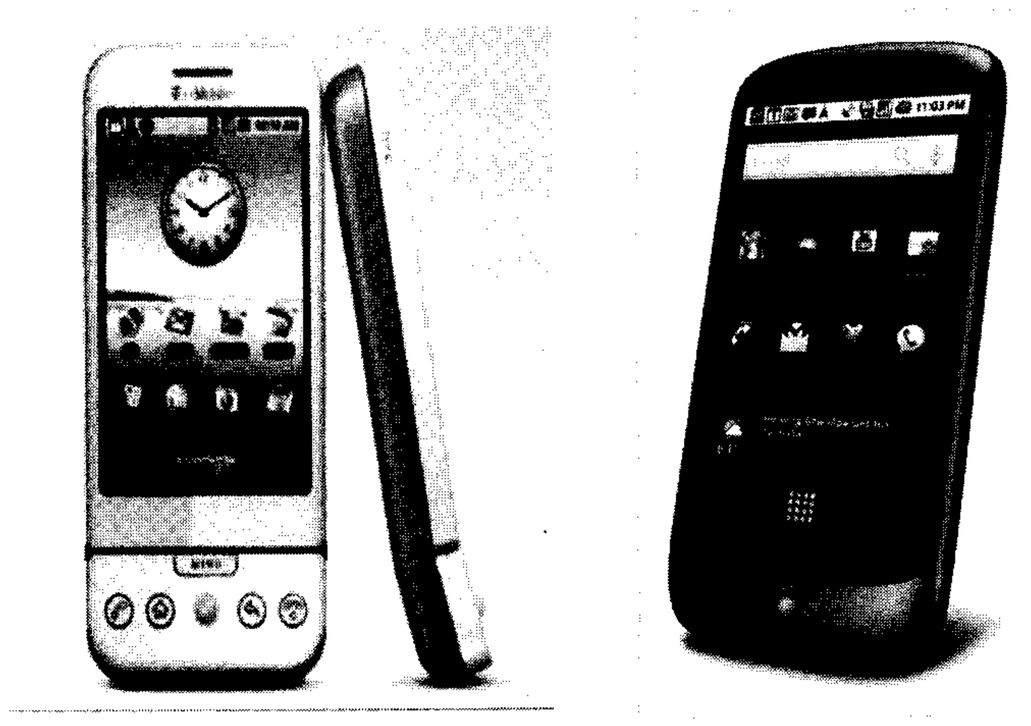


图 1　G1 和 Nexus One

内容简介

本书分为三个部分，包括基础篇、技术篇和应用篇。由浅入深地讲述了 Android 应用开发的方方面面。

篇　名	章　名	内容简介
第一篇　基础篇	第 1 章　Android 概述	Android 概述，讲述了 Android 的前生后世、架构和特点、Android Market、应用程序组件和 Android 与 Java ME 的区别及联系
	第 2 章　Android 开发基础	Android 开发基础，讲述了 Android 开发环境的搭建、Android 常用工具的使用和第一个 Android 应用程序的开发
第二篇　技术篇	第 3 章　Android 中的资源访问	Android 中的资源访问，讲述了如何定义和访问 Android 中的外部资源。
	第 4 章　Android 用户界面	Android 用户界面，讲述了 Android 中的事件处理机制、布局管理和常用组件的使用。
	第 5 章　Android 基本程序单元 Activity	Android 基本程序单元 Activity，讲述了 Android 中重要组件 Activity 的创建、启动和生命周期等内容

续表

篇　名	章　名	内容简介
第二篇　技术篇	第 6 章　Android 组件之间的信使 Intent	Android 组件之间的信使 Intent，讲述了 Intent 对象及其属性、Intent 的实现策略和 Intent 的常见应用
	第 7 章　Android Service 组件	Android Service 组件，讲述了 Android 中的后台服务 Service 的概念、创建和使用，并详细讲解了远程服务的调用
	第 8 章　Android 广播事件处理 Broadcast Receiver	Android 广播事件处理 Broadcast Receiver，讲述了广播事件处理机制、Notification、NotificationManager 和 AlarmManager 的使用
	第 9 章　Android 中的数据存取	Android 中的数据存取，讲述了 Android 的四种数据存取方法：Preference、File、SQLite 和 Network
	第 10 章　Content Provider	Content Provider，讲述了 Android 不同应用程序之间相互共享数据的机制，包括 ContentProvider 和 ContentResolver
	第 11 章　Android 中的多媒体应用	Android 中的多媒体应用，讲述了 Android 的图片应用、音频及视频播放、音频及视频录制和照相机的使用
	第 12 章　Android 中的图形图像	Android 中的图形图像，讲述了 Android 中的图片、动画、图形绘制和图形特效
	第 13 章　Android 中的互联网应用	Android 中的互联网应用，讲述了 Android 中的各种网络应用，包括 Socket、URL、HTTP、Web Service 和 WebView 组件
	第 14 章　Android 中的 GPS 应用	Android 中的 GPS 应用，讲述了 LocationManager、LocationProvider、跟踪、定位、Geocoder 正逆向编解码和可视化位置服务
第三篇　应用篇	第 15 章　Android 应用案例——移动警务通	Android 应用案例——移动警务通，通过一个真实的商业案例讲解了 Android 的各种应用，本项目包括信息采集、信息查询、照片上传、GPS 定位等综合警务应用
	第 16 章　Android 应用案例——雷电游戏	Android 应用案例——雷电游戏，通过一个完整的雷电游戏，讲述了 Android 的游戏开发思路、框架和具体实现，并比较了 Android 游戏开发和 Java ME 游戏开发的异同，成功移植了 Java ME 游戏 API 到 Android 当中
	第 17 章　Android 应用案例——备忘录	Android 应用案例——备忘录，通过一个 Android 基础应用项目综合应用了 Android 中的各种组件，包括 Activity、Service、Broadcast Receiver、ContentProvider、Intent 和 View 的应用
	第 18 章　Android 应用案例——无线点餐系统	Android 应用案例——无线点餐系统，通过一个真实的商业案例讲解了 Android 的各种应用，本项目包括操作员登录、点餐、结算、转台、并台、查台、更新数据和退出系统等餐厅点餐的常用操作

本书特点

1. 实例丰富，步步为“赢”

作为一名从业多年的开发人员，笔者深刻体会到技术型书籍中的实例代码对读者的重要意义，所以本书实例占据了本书的很大篇幅；作为一名多年的培训讲师，笔者也掌握了如何“传道”的技巧，这种技巧就是“编程思路”，也叫“编程步骤”，所以本书中每个实例开始部分都有概括性的步骤总结，使读者有“画竹必先有成竹于胸”的感受。

2. 代码整齐，注释清晰

为了使读者更好更快地学习 Android，书中的代码非常整齐，并且每行代码都有注释。下面的代码段演示了有注释的程序和没有注释的程序。

❋有注释

```
package com.amaker.ch03.dimen;

import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.Button;
import com.amaker.test.R;

/**
*
* @author 郭宏志
* 测试尺寸资源
*/
public class TestDimensionActivity extends Activity {
private Button myButton;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
// 设置当前 Activity 的内容布局视图
setContentView(R.layout.test_dimen);
// 通过 findViewById 方法获得 Button 实例
myButton = (Button)findViewById(R.id.Button01);
// 获得 Resources 实例
Resources r = getResources();
// 通过 getDimension 方法获得尺寸值
float btn_h = r.getDimension(R.dimen.btn_height);
float btn_w = r.getDimension(R.dimen.btn_width);
// 设置按钮的宽
myButton.setHeight((int)btn_h);
// 设置按钮的高
myButton.setWidth((int)btn_w);
}
}
```

❋没有注释

```
package com.amaker.ch03.dimen;

import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.Button;

import com.amaker.test.R;

public class TestDimensionActivity extends Activity {
private Button myButton;
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.test_dimen);
myButton = (Button)findViewById(R.id.Button01);
Resources r = getResources();
float btn_h = r.getDimension(R.dimen.btn_height);
float btn_w = r.getDimension(R.dimen.btn_width);
myButton.setHeight((int)btn_h);
myButton.setWidth((int)btn_w);
}
}
```

3. 案例真实，覆盖全面

本书分为三篇，从 Android 的基础知识讲到 Android 的项目实战，覆盖了 Android 应用开发的三大领域：基础开发、互联网应用开发和游戏开发。

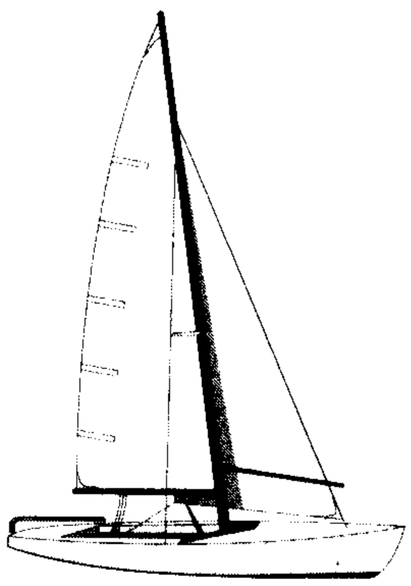
致谢

最后衷心地感谢我的父母、家人、朋友、同学和同行的伙伴们。再次感谢我的父母，感谢你们对我的培养和寄予的厚望，感谢你们拖着年迈的身体为我照料那个让人可爱又可恨的女儿。

感谢电子工业出版社的胡辛征老师在写作风格和布局谋篇方面给予的建议和帮助，感谢江立老师耐心细致的编辑、修正。他们的丰富经验和认真负责的态度，使得本书从开始粗糙的初稿变成最后精美的成书。

郭宏志

2010 年 3 月



CONTENTS

目　　录

第一篇　基础篇

第 1 章　Android 概述……2

1.1　Android 的前世今生……3

1.1.1　Android 的产生……3

1.1.2　Android 的发展……3

1.2　Android 的平台架构及特性……4

1.2.1　Android 平台特性……5

1.2.2　Android 平台架构……5

1.3　Android Market……7

1.4　Android 应用程序组件……8

1.4.1　Activity……9

1.4.2　Service……9

1.4.3　Broadcast Receiver……9

1.4.4　ContentProvider……9

1.4.5　View……10

1.4.6　Intent……10

1.5　Android 与 Java ME 的区别与联系……10

1.5.1　二者的区别……10

1.5.2　二者的联系……10

1.5.3　各自的优势……11

第 2 章　Android 开发基础……12

2.1　Android 开发环境的搭建……12

2.1.1　下载 Android SDK……13

2.1.2　下载安装 JDK……15

2.1.3　下载 Eclipse……16

2.1.4　下载安装 ADT……16

2.2　Android 常用工具的使用……18

2.2.1　创建 Android Virtual Devices（AVD）……18

2.2.2 模拟器（Emulator）的使用……20
2.2.3 Android Debug Bridge（ADB）的使用……20
2.2.4 Dalvik Debug Monitor Service（DDMS）的使用……21
2.2.5 Android Asset Packaging Tool（AAPT）的使用……22
2.2.6 DX 的使用……22
2.2.7 mksdcard 的使用……22
2.3 我的第一个 Android 应用……22
2.3.1 纯手工创建一个 Android 应用……22
2.3.2 使用 Eclipse 创建一个 Android 应用……25

第二篇 技术篇

第 3 章 Android 中的资源访问……32
3.1 资源简介……33
3.1.1 资源的类型和布局……33
3.1.2 资源文件的使用……33
3.2 使用颜色（color）资源……35
3.2.1 颜色值定义……35
3.2.2 颜色资源 XML 文件的定义……35
3.2.3 使用颜色资源……36
3.3 使用字符串（string）资源……37
3.3.1 字符串资源 XML 文件的定义……37
3.3.2 字符串资源 XML 文件的使用……38
3.4 使用尺寸（dimen）资源……39
3.4.1 Android 中支持的尺寸单位……39
3.4.2 尺寸资源 XML 文件的定义……40
3.4.3 尺寸资源 XML 文件的使用……40
3.5 使用原始 XML 资源……42
3.5.1 原始 XML 资源文件的定义……42
3.5.2 原始 XML 文件的使用……42
3.6 使用 drawables 资源……45
3.7 使用布局（layout）资源……47
3.7.1 布局文件的定义……47
3.7.2 布局文件的使用……48
3.8 使用菜单（menu）资源……49
3.8.1 菜单资源文件的定义……50
3.8.2 菜单资源文件的使用……51

第 4 章　Android 用户界面 ……55
4.1　菜单 ……55
4.1.1　选项菜单（Option Menu） ……56
4.1.2　上下文菜单（Context Menu） ……57
4.1.3　子菜单（Sub Menu） ……59
4.2　对话框 ……61
4.3　提示信息（Toast） ……64
4.4　事件处理 ……66
4.4.1　事件处理机制 ……66
4.4.2　Android 中的事件监听器 ……68
4.4.3　事件处理步骤 ……69
4.5　布局管理（Layout） ……72
4.5.1　线性布局 ……73
4.5.2　帧布局 ……75
4.5.3　表格布局 ……76
4.5.4　相对布局 ……77
4.5.5　绝对布局 ……78
4.6　组件（Widget） ……79
4.6.1　常用组件 ……79
4.6.2　自动完成文本框（AutoCompleteTextView） ……86
4.6.3　选项卡（Tab） ……88
4.6.4　进度条（ProgressBar） ……91
4.6.5　日期、时间选择对话框（DataPickerDialog、TimePickerDialog） ……95
4.6.6　列表视图（ListView） ……97
4.6.7　网格视图（GridView） ……99
4.6.8　画廊视图（Gallery） ……101
4.6.9　地图视图（MapView） ……105
4.6.10　网络视图（WebView） ……108
第 5 章　Android 基本程序单元 Activity ……111
5.1　何谓回调 ……111
5.2　Activity 简介 ……113
5.2.1　Activity 的创建 ……113
5.2.2　启动另一个 Activity ……115
5.2.3　Activity 之间传递数据 ……117
5.2.4　启动另一个 Activity 并返回结果 ……121

5.3 Activity 的生命周期 …… 126
第 6 章 Android 组件之间的信使 Intent …… 130
6.1 Intent 对象及其属性 …… 131
6.1.1 Intent 的 ComponentName 属性 …… 131
6.1.2 Intent 的 Action 属性 …… 135
6.1.3 Intent 的 Data 属性 …… 140
6.1.4 Intent 的 Category 属性 …… 141
6.1.5 Intent 的 Extras 属性 …… 142
6.2 系统标准 Activity Action 应用 …… 144
6.2.1 和打电话相关的标准 Activity Action 应用 …… 145
6.2.2 访问浏览器和地图 …… 147
6.2.3 发邮件 …… 149
6.3 Intent 的实现策略 …… 151
第 7 章 Android Service 组件 …… 159
7.1 Service 简介 …… 159
7.1.1 创建一个 Service …… 159
7.1.2 启动和停止 Service …… 160
7.1.3 绑定一个已经存在的 Service …… 161
7.1.4 Service 实例演示 …… 161
7.2 远程 Service 调用 …… 166
7.2.1 创建一个 AIDL 文件 …… 166
7.2.2 实现 AIDL 文件生成的 Java 接口 …… 170
7.2.3 将你的接口暴露给客户端 …… 171
7.2.4 客户端调用 …… 171
第 8 章 Android 广播事件处理 Broadcast Receiver …… 173
8.1 自己定义 Broadcast Receiver 来处理广播事件 …… 173
8.2 系统广播事件的使用 …… 176
8.3 Notification 和 NotificationManager 的使用 …… 178
8.3.1 Notification 和 NotificationManager 简介 …… 178
8.3.2 通知实例演示 …… 179
8.4 AlarmManager 的使用 …… 186
第 9 章 Android 中的数据存取 …… 190
9.1 Preference …… 190

9.1.1 Preference 简介 ……190
9.1.2 Preference 应用实例——保存临时短信 ……191
9.2 File ……193
9.3 SQLite ……195
9.3.1 SQLiteDatabase ……195
9.3.2 SQLiteOpenHelper ……199
9.3.3 SQLite 应用实例——收藏管理 ……201
第 10 章 Content Provider ……208
10.1 Content Provider 简介 ……208
10.1.1 Content Provider 的常用方法 ……208
10.1.2 ContentResolver ……209
10.1.3 URI ……209
10.1.4 查询系统 ContentProvider 内容 ……210
10.1.5 添加系统 ContentProvider 内容 ……211
10.1.6 添加系统 ContentProvider 图片内容 ……211
10.2 自定义 ContentProvider ……212
10.2.1 创建 ContentProvider 的步骤 ……212
10.2.2 ContentProvider 实例 ……213
第 11 章 Android 中的多媒体应用 ……221
11.1 音频及视频播放 ……221
11.1.1 从源文件中播放 ……221
11.1.2 从文件系统中播放 ……222
11.1.3 从网络中播放 ……222
11.2 迷你音乐播放器 ……223
11.3 迷你视频播放器 ……228
11.4 音视频的录制 ……229
11.5 Camera 照相 ……233
第 12 章 Android 中的图形图像 ……238
12.1 在 Android 中访问图片 ……238
12.1.1 使用图片文件创建 Drawable 对象 ……239
12.1.2 使用 XML 文件定义 Drawable 属性 ……240
12.1.3 Bitmap 和 BitmapFactory ……241
12.2 Android 中的动画 ……242
12.2.1 Tween 动画 ……242

12.2.2 Frame 动画 ····· 251
12.3 动态图形绘制 ····· 253
12.3.1 动态图形绘制的基本思路 ····· 253
12.3.2 动态图形绘制类简介 ····· 256
12.3.3 绘制几何图形 ····· 257
12.4 图形特效 ····· 261
12.4.1 使用 Matrix 实现旋转、缩放和平移 ····· 261
12.4.2 使用 Shader 类渲染图形 ····· 264
第 13 章 Android 中的互联网应用 ····· 268
13.1 通过 Socket、ServerSocket 进行网络编程 ····· 268
13.1.1 Socket、ServerSocket 编程模型 ····· 268
13.1.2 Socket 编程实例 ····· 269
13.2 通过 URL 进行网络编程 ····· 271
13.3 通过 HTTP 进行网络编程 ····· 273
13.3.1 使用 HttpURLConnection ····· 273
13.3.2 使用 Apache HTTP 客户端 ····· 278
13.4 通过 Web Service 进行网络编程 ····· 280
13.5 直接使用 WebView 视图组件显示网页 ····· 285
13.5.1 使用 WebView 打开网页 ····· 285
13.5.2 使用 WebView 加载 HTML ····· 286
第 14 章 Android 中的 GPS 应用 ····· 287
14.1 LocationManager 和 LocationProvider 简介 ····· 287
14.1.1 LocationManager ····· 288
14.1.2 LocationProvider ····· 288
14.2 通过模拟器测试位置服务 ····· 289
14.3 获得 LocationProvider ····· 292
14.3.1 通过名称获得 LocationProvider ····· 292
14.3.2 获得当前可利用的 LocationProvider ····· 292
14.3.3 根据 Criteria 条件获得 LocationProvider ····· 292
14.4 定位和跟踪 ····· 293
14.4.1 定位 ····· 294
14.4.2 跟踪 ····· 296
14.5 趋近警告 ····· 298
14.6 Geocoder 正逆向编解码 ····· 300
14.6.1 正向编码 ····· 301
14.6.2 反向编码 ····· 303

第三篇　应用篇

第 15 章　Android 应用案例——移动警务通……306

15.1　移动警务通需求分析……306

15.2　移动警务通总体设计……307

15.2.1　系统架构……307

15.2.2　技术选型……307

15.2.3　系统功能……308

15.3　移动警务通详细设计……309

15.3.1　系统包及其资源规划……309

15.3.2　Activity 界面规划及其程序执行流程……310

15.3.3　系统数据库设计……312

15.4　系统编码实现……313

15.4.1　登录模块 Android 客户端实现……313

15.4.2　登录模块服务器端实现……320

15.4.3　程序主菜单实现……325

15.4.4　信息查询子菜单实现……326

15.4.5　在逃人员查询模块客户端实现……327

15.4.6　在逃人员查询模块服务器端实现……332

15.4.7　信息采集子菜单实现……338

15.4.8　机动车违章信息采集 Android 客户端实现……339

15.4.9　机动车违章信息采集服务器端实现……345

15.4.10　文件上传子菜单实现……348

15.4.11　现场照片上传 Android 客户端实现……349

15.4.12　现场照片上传服务器端实现……360

15.4.13　GPS 定位功能子菜单实现……363

15.4.14　GPS 定位“我的位置”的实现……364

15.4.15　GPS 定位“按坐标查询”的实现……369

15.4.16　GPS 定位“按地址查询”的实现……374

第 16 章　Android 应用案例——雷电游戏……378

16.1　Android 游戏开发基本框架……378

16.1.1　Android 游戏开发基础……378

16.1.2　Android 游戏开发基本框架……379

16.2　将 Java ME 游戏 API 嫁接到 Android……384

16.2.1　Layer 类的移植……384

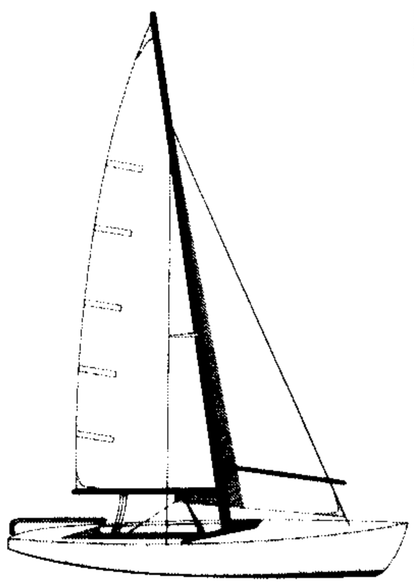
16.2.2 Sprite 类的移植 …… 386
16.2.3 TiledLayer 类的移植 …… 386
16.2.4 LayerManager 类的移植 …… 386
16.3 Android 版雷电游戏的实现 …… 387
16.3.1 雷电游戏简介 …… 387
16.3.2 雷电游戏的实现 …… 387
第 17 章 Android 应用案例——备忘录 …… 396
17.1 备忘录数据存储实现 …… 396
17.1.1 数据库表设计 …… 396
17.1.2 备忘录 ContentProvider 实现 …… 397
17.2 备忘录列表展示 …… 402
17.3 备忘录维护 …… 405
17.4 备忘录定时提醒 …… 412
17.4.1 定义提醒广播接收器 …… 412
17.4.2 显示提醒 Activity …… 413
第 18 章 Android 应用案例——无线点餐系统 …… 417
18.1 无线点餐系统需求分析 …… 417
18.2 无线点餐系统概要设计 …… 417
18.2.1 系统物理架构 …… 418
18.2.2 技术选型 …… 418
18.2.3 系统功能 …… 419
18.3 无线点餐系统详细设计 …… 422
18.3.1 系统包及其资源规划 …… 422
18.3.2 系统数据库设计 …… 423
18.4 系统编码实现 …… 424
18.4.1 登录模块 Android 客户端实现 …… 424
18.4.2 登录模块服务器端实现 …… 432
18.4.3 程序主菜单实现 …… 437
18.4.4 点餐功能客户端实现 …… 440
18.4.5 点餐功能服务器端实现 …… 450
18.4.6 结算模块 Android 客户端实现 …… 455
18.4.7 结算模块服务器端实现 …… 458
18.4.8 查台模块 Android 客户端实现 …… 466
18.4.9 查台模块服务器端实现 …… 470

18.4.10 更新模块 Android 客户端实现 ····· 473
18.4.11 更新模块服务器端实现 ····· 477
18.4.12 转台模块 Android 客户端实现 ····· 480
18.4.13 转台模块服务器端实现 ····· 482
18.4.14 并台模块 Android 客户端实现 ····· 485
18.4.15 并台模块服务器端实现 ····· 488

第一篇

基 础 篇

- 第 1 章　Android 概述
- 第 2 章　Android 开发基础



第 1 章　Android 概述

时势造英雄，PC 时代微软凭借 Windows 操作系统造就了微软帝国，互联网时代 Google 凭借搜索引擎造就了 Google 帝国。那么，移动互联网时代来临，谁将是真正的英雄呢？这个我们很难定论，但是一个不确定的答案就是，掌握移动互联网技术的人将是这个时代的弄潮儿，将成为这一时代真正的英雄。

这里我们思考一下，为什么能产生了两个帝国，而不是一个帝国呢？设想如果微软既掌握了操作系统技术又掌握了搜索引擎技术，那么微软将成为唯一帝国；如果 Google 既掌握了操作系统技术又掌握了搜索引擎技术，那么 Google 将成为唯一帝国。就是因为他们各有所长，所以两个帝国才能并存。俗话说，一山不能容二虎，二者都有垄断唯一霸主地位的野心。

在这个移动互联网时代，Google 历经数年，耗资数亿，研发了手机系统 Android。其实 Android 不光是一个手机操作系统，它可以延伸到各种移动设备和 PC。有了自己的互联网技术和自己的操作系统，也就有了自己唯一的霸主地位。这就是 Google 的野心吧！

Android 一经推出，受到了业界前所未有的热捧。全世界的 Android 社区热火朝天，开发人员痴迷学习，手机设备厂商不断推出 Android 系统的手机，为什么会有这样的效果呢？可以用一句话来概括："博采众长，为它所用。" Android 采用了开源的 Linux 操作系统，底层使用了访问硬件速度最快的 C 语言，应用层采用了简单而强大的 Java 语言，这使得 Android 无处不闪烁着耀眼的光芒。

Android 实现了全部开源，这为 Android 的使用提供了无限可能。硬件厂商可以在原有基础上，根据自己的需求扩展或裁剪现有功能，软件开发人员可以开发出更优秀的软件替换现有软件。

我们都知道，之前的手机市场为少数实力雄厚的公司所垄断，Android 的推出将使得这个市场重新"洗牌"。王者归来，个人英雄主义又将重现。你可以开一家公司提供定制的 Android 系统，也可以开一家公司开发 Android 应用软件、游戏在 Android Market 上出售。

本章我们将学习如下内容：

- Android 的前世今生
- Android 的平台架构及特性
- Android Market
- Android 应用程序组件
- Android 与 Java ME 的区别与联系

1.1 Android 的前世今生

Android 是“机器人”的意思，是 Google 公司于 2007 年 11 月 5 日发布的一个基于 Linux 平台的开源手机操作系统。该系统由底层的 Linux 操作系统、中间件和核心应用程序组成。Android 应用程序由强大的 Java 语言来编写，也支持其他一些语言如 C、Perl 等。

1.1.1 Android 的产生

Android 一开始并不是 Google 自己研发的产品，而是收购了一家刚刚创业 22 个月的公司的产品，该公司的创始人是 Andy Rubin（如图 1.1 所示），也就是现在 Google Android 产品负责人罗宾。Google 收购 Android 没有向媒体透露任何消息，可以说是悄悄进行的。罗宾拒绝对 Android 公司或出售给 Google 发表评论。Google 的一名发言人也拒绝就这一交易发表评论，只是指出“我们收购 Android 公司的原因是，它有天才的工程师和了不起的技术，我们对他们的加盟感到由衷的高兴”。

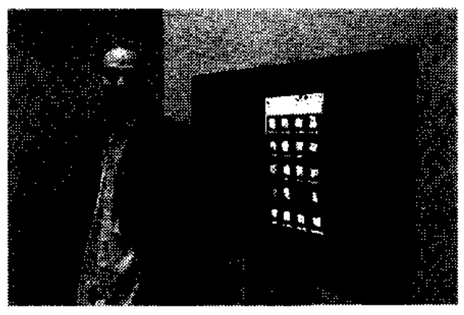


图 1.1　Android 发明者 Andy Rubin

Google 于 2007 年 11 月 5 日发布了 Android 1.0 手机操作系统，号称是首个为移动终端打造的真正开放和完整的移动软件，后经版本不断更新，到笔者成书时的最新版本是 Android 2.1。

同日，Google 宣布组建了一个全球性的联盟组织。这一联盟将会支持 Google 可能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几类。目前，联盟成员数量已经达到了 34 家。移动运营商包括中国移动、中国电信、中国联通、美国 Sprint Nextel、德国 T-Mobile 等；半导体制造商包括 ARM、英特尔、高通等；电话制造商包括摩托罗拉、三星、华硕、华为、宏达电子等。

1.1.2 Android 的发展

Android 软件经推出之后，版本升级非常快，几乎每隔半年就有一个新的版本发布。

版本先后经历了 1.0、1.1、1.5、1.6，目前的最新版本是 Android 2.1。

2008 年 9 月 22 日，美国运营商 T-MobileUSA 在纽约正式发布第一款 Google 手机——T-Mobile G1，如图 1.2 所示。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，理论下载速率为 7.2Mbps，并支持 Wi-Fi。

随后，各大手机厂商纷纷发布基于 Android 操作系统的手机，包括摩托罗拉、宏达电、三星、联想、华为等。如图 1.3 所示是摩托罗拉 Android 2.0 触屏 Droid 手机，Droid 采用了基于 ARM Cortex-A8 的 TI OMAP3430 处理器，其速度不逊色于当前的任何一款 Android 手机。它采用 3.7 英寸 FWVGA 触控屏，480×854 像素，支持多点触控操作，配置包括 500 万像素摄像头，支持 A-GPS、WiFi、256MB RAM、512MB ROM，可扩展 16GB 存储卡。

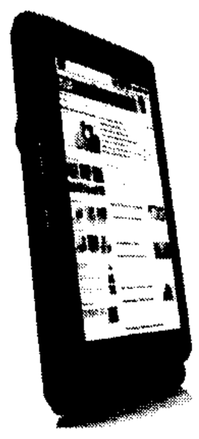


图 1.2　第一款 Android 手机 T-MobileG1

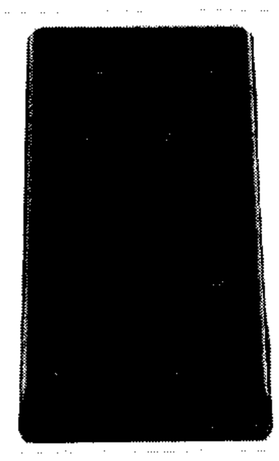


图 1.3　摩托罗拉 Android 2.0 触屏 Droid 手机

2010 年 1 月 6 日 Google 推出其自主研发手机 Nexus One（如图 1.4 所示），Google 终于跳入智能型手机直接销售的杀戮战场。Nexus One 由 HTC 量身打造，使用了最新版本的 Android 2.1 操作系统。

图 1.4　Google 自主研发的手机 Nexus One

Nexus One 拥有 3.7 英寸液晶显示屏，使用高通公司 Qualcomm 的 1 千兆赫芯片，是当今市场上运行速度最快的芯片。除此之外，这款手机拥有清晰的色点、传感电池技术、指南针、GPS、加速计，以及屏幕光敏电池管理，而且它非常轻巧，只有 11.5 毫米厚，130 克重。

Nexus One 拥有 500 万像素的相机和 LED flash，可以拍摄照片和录制视频。轻松一点就可以上传到 YouTube。配有 3.5 毫米耳机插口和实时噪声消除功能，两个话筒，前后各一个，非常酷!音质一流，环绕手机的金属边框还可供用户按个人喜好替换。该款手机无论是从功能还是外观上都可以和现在最为流行的 iPhone 有一拼了。

1.2 Android 的平台架构及特性

Android 平台采用了整合的策略思想，包括底层 Linux 操作系统、中间层的中间件和

上层的 Java 应用程序。本小节我们将详细讲解有关 Android 的特性及其架构体系结构。

1.2.1　Android 平台特性

Android 平台有如下特性：

- 应用程序框架支持组件的重用与替换。这在之前的手机操作系统当中是很难想象的。这意味着我们可以把系统中不喜欢的应用程序替换掉，安装我们自己喜欢的应用程序，例如，打电话应用程序、文件管理器等。
- Dalvik 虚拟机专门为移动设备做了优化。Android 应用程序将由 Java 编写、编译的类文件通过 DX 工具转换成一种后缀名为.dex 的文件来执行。Dalvik 虚拟机是基于寄存器的，相对于 Java 虚拟机速度要快很多。
- 内部集成浏览器基于开源的 WebKit 引擎。有了内置的浏览器，这将意味着 WAP 应用的时代即将结束，真正的移动互联网时代已经来临，手机就是一台“小电脑”，可以在网上随意遨游。
- 优化的图形库包括 2D 和 3D 图形库，3D 图形库基于 OpenGL ES 1.0。强大的图形库给游戏开发带来福音。在我看来 3G 最为重要的应用莫过于手机上网和手机游戏了。
- SQLite 用作结构化的数据存储。
- 多媒体支持包括常见的音频、视频和静态印象文件格式（如 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG、GIF）。
- GSM 电话（依赖于硬件）。
- 蓝牙（Bluetooth）、EDGE、3G、WiFi（依赖于硬件）。
- 照相机、GPS、指南针和加速度计（依赖于硬件）。
- 丰富的开发环境包括设备模拟器、调试工具、内存及性能分析图表和 Eclipse 集成开发环境插件。Google 提供了 Android 开发包 SDK，其中包含了大量的类库和开发工具。并且专门开发了针对 Eclipse 的可视化开发插件 ADT，这部分内容将在后续章节详细讲述。

1.2.2　Android 平台架构

如图 1.5 所示的是 Android 操作系统的体系结构。每一部分将会在下面具体描述。

从图 1.5 中我们可以看出 Android 操作系统体系结构分为 4 层，由上而下依次是应用程序、应用程序框架、核心类库和 Linux 内核。其中在第三层还包括 Android 运行时环境。下面分别来讲解各个部分。

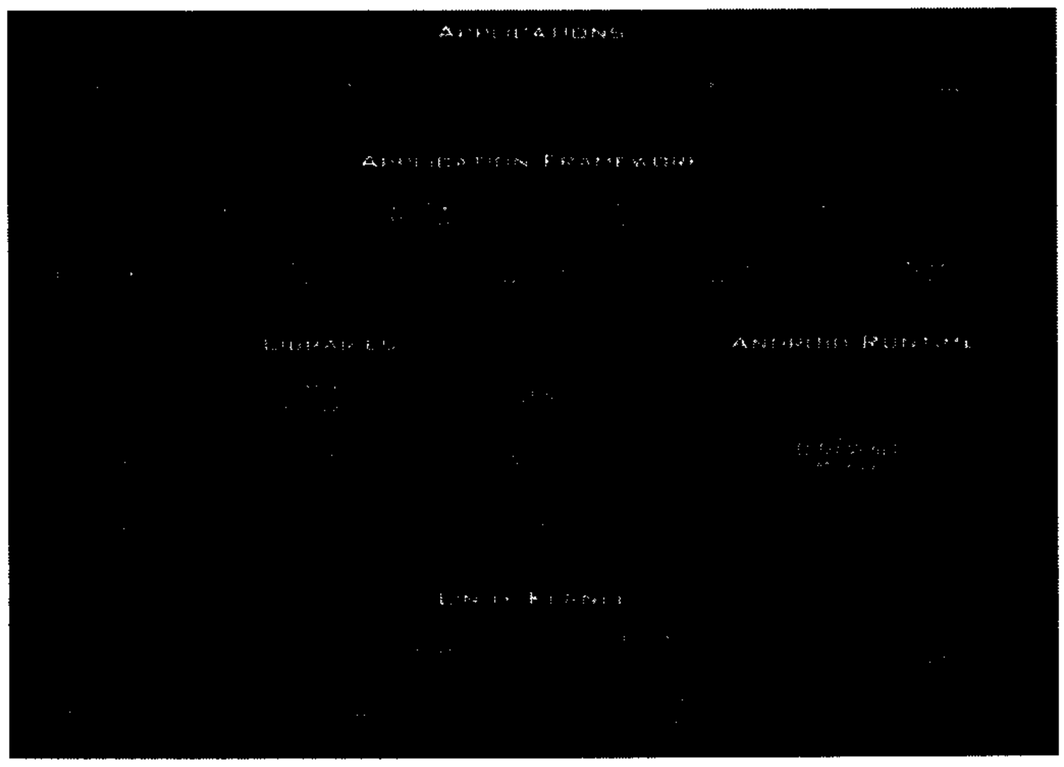


图 1.5　Android 操行系统的体系结构

1．应用程序

Android 连同一个核心应用程序包一起发布，该应用程序包包括 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序等。所有的应用程序都是用 Java 编写的。

2．应用程序框架

开发者完全可以访问核心应用程序所使用的 API 框架。该应用程序架构用来简化组件软件的重用，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其所发布的功能块（不过得遵循框架的安全性限制）。该应用程序重用机制使得组件可以被用户替换。

以下所有的应用程序都由一系列的服务和系统组成，包括：

1）一个可扩展的视图（Views）可以用来创建应用程序，包括列表（lists）、网格（grids）、文本框（text boxes）、按钮（buttons），甚至是一个可嵌入的 Web 浏览器。

2）内容管理器（Content Providers）使得应用程序可以访问另一个应用程序的数据（如联系人数据库），或者共享它们自己的数据。

3）一个资源管理器（Resource Manager）提供非代码资源的访问，如本地字符串、图形和分层文件（layout files）。

4）一个通知管理器（Notification Manager）使得应用程序可以在状态栏中显示客户通知信息。

5）一个活动类管理器（Activity Manager）用来管理应用程序生命周期并提供常用的导航回退功能。

3．Android 程序库

Android 包括一个被 Android 系统中各种不同组件所使用的 C/C++库集。该库通过

Android 应用程序框架为开发者提供服务。以下是一些主要的核心库。

1）系统 C 库：一个从 BSD 继承来的标准 C 系统函数库（libc），专门为基于 Embedded Linux 的设备定制。

2）媒体库：基于 PacketVideo OpenCORE；该库支持录放，并且可以录制许多流行的音频视频格式，还有静态映像文件包括 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG。

3）Surface Manager：对显示子系统的管理，并且为多个应用程序提供 2D 和 3D 图层的无缝融合。

4）LibWebCore：一个最新的 Web 浏览器引擎，用来支持 Android 浏览器和一个可嵌入的 Web 视图。

5）SGL：一个内置的 2D 图形引擎。

6）3D libraries：基于 OpenGL ES 1.0 APIs 实现；该库可以使用硬件 3D 加速（如果可用）或者使用高度优化的 3D 软加速。

7）FreeType：位图（bitmap）和向量（vector）字体显示。

8）SQLite：一个对于所有应用程序可用、功能强劲的轻型关系型数据库引擎。

4. Android 运行库

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

每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 是针对同时高效地运行多个 VMs 来实现的。Dalvik 虚拟机执行.dex 的 Dalvik 可执行文件，该格式文件针对最小内存使用做了优化。该虚拟机是基于寄存器的，所有的类都经由 Java 汇编器编译，然后通过 SDK 中的 DX 工具转化成.dex 格式由虚拟机执行。

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

5. Linux 内核

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

1.3 Android Market

2008 年 8 月 28 日，Google 宣布推出 Android Market（如图 1.6 所示）。Android Market 运行用户浏览、下载使用由第三方开发商为 Android 开发的应用程序。用户可以购买或免费使用这些应用程序，这些应用程序可以直接下载到 Android 手机中。Android Market 的网址是 http://www.android.com/market/。用户可以在这里找到很多需要的 Android 应用程序。

我们可以开发自己的 Android 应用程序并发布到 Android Market，不过在发布自己的应用程序之前，需要申请一个 Market 账户。申请一个 Market 账户，需要填写如图 1.7 所示的基本信息并缴纳 25 美元服务费，因此需要一张国际信用卡，可以是 VISA 或 Master。申请网址是 http://market.android.com/publish/signup。

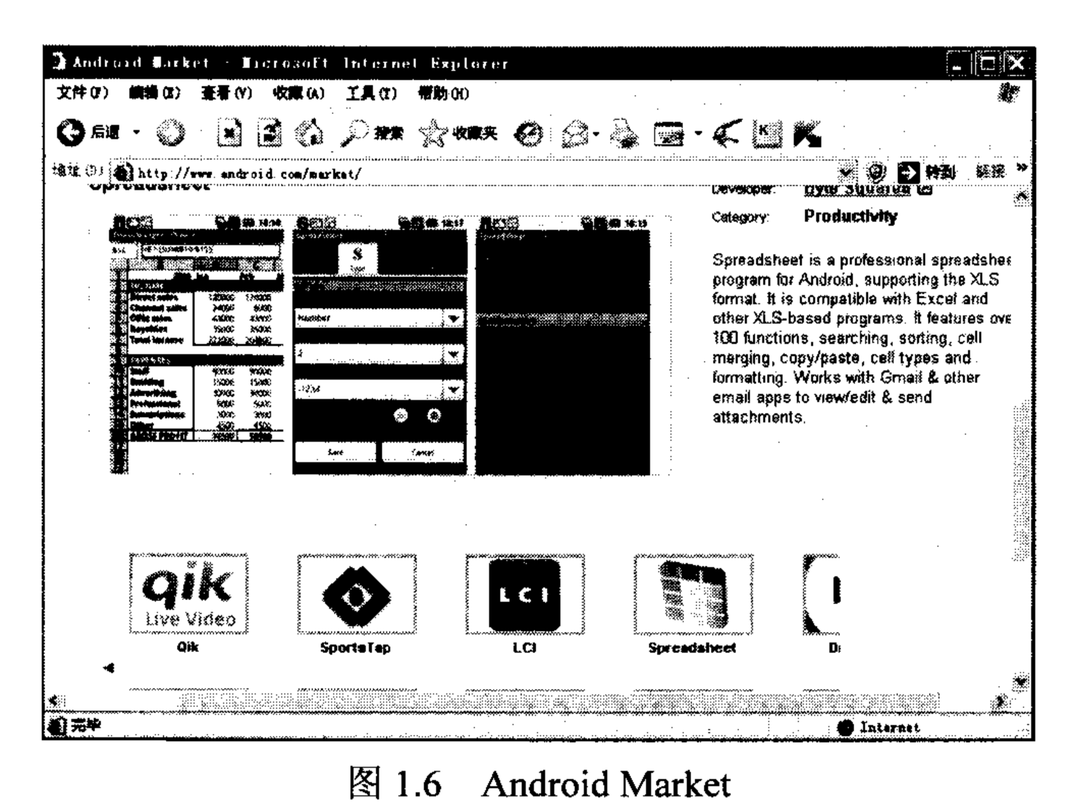


图 1.6　Android Market

有了一个 Market 账户，我们就可以开发自己的 Android 应用程序并在 Android Market 上赚钱了。这也成了 Android 开发人员除了上班赚钱的另外一条致富途径了。

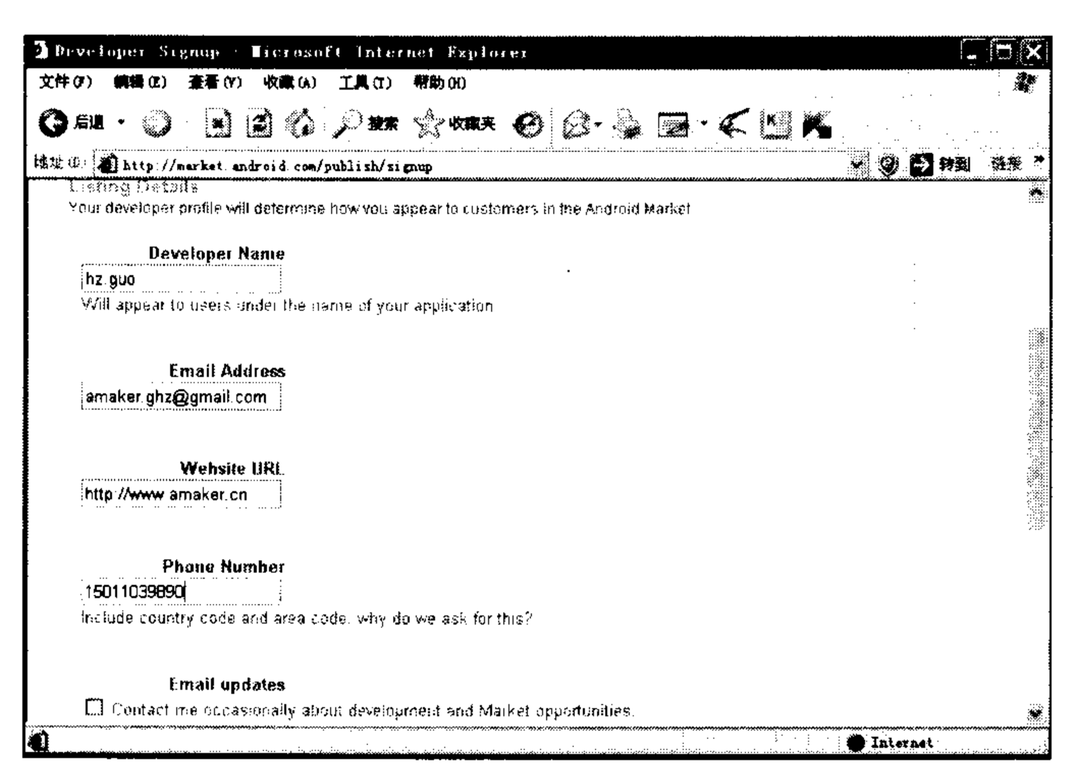


图 1.7　Android Market 账户申请

1.4 Android 应用程序组件

如果你想从事 Android 应用程序开发，那么了解 Android 应用程序的思想是非常必要的。Android 应用程序没有统一的入口（例如 main()方法），各个应用之间是相互独立的，并且运行在自己的进程当中。根据完成的功能不同，Android 划分了四类核心的组件类：Activity、Service、Broadcast Receiver 和 ContentProvider。相同组件和不同组件之间的导航通过 Intent 来完成。Android 还定义了 View 类来显示可视化界面，例如菜单、对话框、下拉列表等。本小节将详细讲述各个组件的意义和用法。

1.4.1　Activity

Activity 是 Android 组件中最基本也是最为常用的一种组件，在一个 Android 应用中，一个 Activity 通常就是一个单独的屏幕。每一个 Activity 都被实现为一个独立的类，并且继承于 Activity 这个基类。这个 Activity 类将会显示由几个 Views 控件组成的用户接口，并对事件做出响应。大部分的应用都会包含多个屏幕。例如，一个短消息应用程序将会有一个屏幕用于显示联系人列表，第二个屏幕用于写短消息，同时还会有用于浏览旧短消息及进行系统设置的屏幕。每一个这样的屏幕就是一个 Activity。

通过调用 startActivity()方法可以从一个屏幕导航到另一个屏幕，打开 Activity 的条件被封装在 Intent 中。

当一个新的屏幕打开后，前一个屏幕将会暂停，并保存在历史堆栈中。用户可以返回到历史堆栈中的前一个屏幕。当屏幕不再使用时，还可以从历史堆栈中删除。默认情况下，Android 将会保留从主屏幕到每一个应用的运行屏幕。

1.4.2　Service

一个 Service 是一种长生命周期的、没有用户界面的程序。比较好的一个例子就是正在从播放列表中播放歌曲的媒体播放器。在一个媒体播放器的应用中，应该会有多个 Activity，让使用者可以选择歌曲并播放歌曲。然而，音乐重放这个功能并没有对应的 Activity，因为使用者会认为在导航到其他屏幕时音乐应该还在播放。在这个例子中，媒体播放器这个 Activity 会使用 Context.startService()来启动一个 Service，从而可以在后台保持音乐的播放。同时，系统也将保持这个 Service 一直执行，直到这个 Service 运行结束。

另外，我们还可以通过使用 Context.bindService()方法连接到一个 Service 上（如果这个 Service 还没有运行则将启动它）。当连接到一个 Service 之后，我们还可以通过 Service 提供的接口与它进行通信。拿媒体播放器这个例子来说，我们还可以进行暂停、重播等操作。

1.4.3　Broadcast Receiver

Broadcast Receiver 是为了实现系统广播而提供的一种组件。例如，我们可以发出一种广播来检测手机电量的变化，这时候就可以定义一个 Broadcast Receiver 来接收广播，当手机电量较低时提示用户。

1.4.4　ContentProvider

在前面的学习中，我们知道 Android 应用程序之间是相互独立的，各个组件运行在不

同的进程当中，这就意味着数据是不能共享的。如何使得不同组件数据的共享呢？Android 通过使用 ContentProvider 来实现不同组件之间数据的共享。

1.4.5 View

View 是 Android 中图形用户界面的基类，提供了可视化界面的展示。Android 的图形界面展示可以分为三层：底层是 Activity；Activity 上面是 Window；Window 上面是 Views。View 又可以分为 View 和 ViewGroup。View 是指基本的控件，例如按钮、单选框、多选框、菜单等；ViewGroup 是指布局控件，即用来控制界面中的控件如何布局摆放的。

1.4.6 Intent

Intent 是不同组件之间相互导航的纽带，封装了不同组件之间导航查找的条件。在 Intent 的描述结构中，有两个最重要的部分：动作和动作对应的数据。典型的动作类型有 MAIN（Activity 的门户）、VIEW、PICK、EDIT 等。而动作对应的数据则以 URI 的形式进行表示。例如，要查看一个人的联系方式，你需要创建一个动作类型为 VIEW 的 Intent，以及一个表示这个人的 URI。

1.5 Android 与 Java ME 的区别与联系

本书是讲解 Android 应用程序开发的书籍，有一定 Java ME 开发经验的读者，自然会想到 Java ME 也是做手机应用的，二者到底有何区别与联系，本节将对这一问题做详细讲解。

1.5.1 二者的区别

概括地讲，Android 与 Java ME 的区别在于，Android 是一个完整的移动设备操作系统平台，由 Linux 操作系统、中间件、C 类库和核心应用程序组成，而 Java ME 只是 Java 的一个微型版本，针对移动设备来开发应用程序的开发包，它必须有底层操作系统的支持，如 Symbian、Win CE 等。

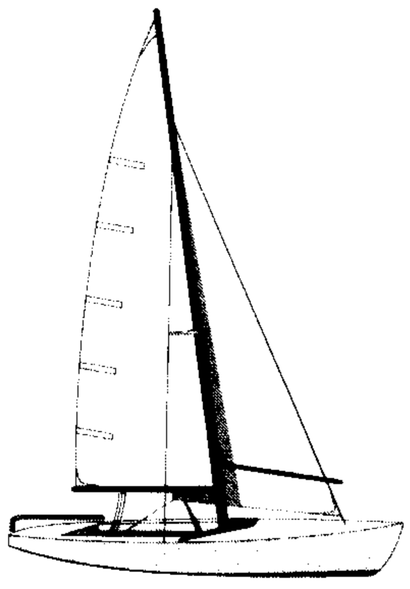
1.5.2 二者的联系

Android 和 Java ME 联系并不大，如果非要说一说它们的联系的话，应该在应用程序的编程语言上。Android 应用程序和 Java ME 都是 Java 编程语言。有过 Java 编程经验的读者可以轻松学习 Java ME 和 Android 应用程序的开发。笔者在本书的第 16 章中，详细介绍了如何将 Java ME 中的游戏 API 移植到 Android 中。

1.5.3　各自的优势

我们可以把 Android、Symbian 和 Win CE 放在一起来比较各自的优缺点，但是，我们很难讲 Android 与 Java ME 谁好谁坏，因为，它们的出发点和应用场合不一样。

Android 的优势在于它的开放性、开源等优势，是一个优秀的移动设备操作系统。Java ME 是一个移动设备开发软件包，跨平台性是它最主要的特点。有人预言 Android 的出现将导致 Java ME 的死亡，笔者并不认同这种看法，Symbian、Win CE 等其他手机操作系统存在一天，Java ME 的应用就不会减少，甚至我们可以在 Android 中开发 Java ME 应用，二者并不冲突。



第 2 章　Android 开发基础

“工欲善其事，必先利其器”，要想高效、快速地开发 Android 应用程序，好的开发工具和开发环境是必不可少的。

在 Android SDK 中提供了一系列工具，它们有模拟硬件设备的 Emulator（模拟器）、Android Asset Packaging Tool（AAPT）Android 资源打包工具、Dalvik Debug Monitor Service（DDMS）Dalvik 调试监视服务、Android Debug Bridge（adb）Android 调试桥和将.class 字节码文件转换为.dex 文件的 DX 工具等。

上述的这些工具我们可以直接在 DOS 命令行中调用，可以进行开发、调试、编译、打包、部署等工作，但是这种开发效率太低了。Android 提供了针对 Eclipse 的开发插件 Android Development Tools（ADT，Android 开发工具）。有了 ADT 我们可以在 Eclipse 开发环境中快速创建 Android 应用程序，自动生成一些代码。调用 Android SDK 中的工具（如 DDMS 进行调试、调用 AAPT 打包等）可以可视化地在布局文件中添加、删除视图组件，极大提高了开发效率。

ADT 必须有 Eclipse 和 Android SDK 的支持，而 Eclipse 必须有 JDK 的支持。安装环境的配置是这样的：下载 Android SDK、下载安装 JDK、下载 Eclipse、下载安装 ADT。

本章我们将讲述 Android 开发环境的搭建、Android SDK 中的常用命令的使用以及如何手工和使用 Eclipse 开发一个简单的 Android 应用。

2.1 Android 开发环境的搭建

Android 应用程序使用 Java 语言编写，使用 Java 语言必须有 JDK，所以我们需要下载和安装 JDK。针对 Java 语言的集成开发环境有很多，如早期的 JBuilder、Sun 公司的 NetBeans，当然使用最为广泛的是 IBM 公司开发的 Eclipse 平台，本节将介绍如何搭建 Eclipse 的 Android 开发环境。

2.1.1　下载 Android SDK

Android SDK 的官方下载网站是 http://developer.android.com./sdk/index.html。在 Android 1.5 之前 Google 提供了 Android SDK 开发包的完整下载，在 Android SDK 1.6 之后 Google 只提供了安装工具在线安装。Android SDK 的下载链接如图 2.1 所示。

December 2009

The Android SDK has changed! If you've worked with the Android SDK before, you will notice several important differences:

If you are new to the Android SDK, please read the Quick Start, below, for an overview of how to install and set up the SDK.

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r04-windows.zip	23069119 bytes	c43b407de852ba483869f17337e90997
Mac OS X (intel)	android-sdk_r04-mac_86.zip	19657927 bytes	b08512765aa9b0369bb9b8fecdf763e3
Linux (i386)	android-sdk_r04-linux_86.tgz	15964887 bytes	ef84b08fd9da84f4c4ae77564fe4eaee

图 2.1　Android SDK 下载链接

我们这里以 Windows 操作系统为例来讲述如何安装配置 Android 开发环境。如图 2.1 所示，单击“android-sdk_r04-windows.zip”链接进行 SDK 下载，下载该文件后解压该文件到操作系统的任意目录下。打开该目录内容如表 2.1 所示。

表 2.1　Android SDK 下载包说明

目录结构或文件	说　　明
add-ons	空目录保存 Google 插件工具
platforms	空目录保存不同版本 SDK
tools	SDK 工具
SDK Setup.exe	在线安装 SDK 的可执行文件
SDK Readme.txt	说明文件

下载完成后，打开目录找到 SDK Setup.exe 可执行文件，双击执行该文件，运行界面如图 2.2 所示。

如图 2.2 所示，左侧列表的第一项可以创建 AVD；第二项显示已经安装的包；第三项显示可以安装的包；第四项是设置选项；第五项是说明选项。

如果读者是第一次安装，那么我们选择 Available Packages，右侧列表显示可以安装的不同版本的软包。这里列表的内容有 SDK 开发文档、不同版本的 SDK 和 Google API。选择你要安装的 SDK 版本和 Google API 版本和文档，界面如图 2.3 所示。

我们选择了要安装的软件包之后，单击“Install Selected”按钮弹出确认对话框，如图 2.4 所示。

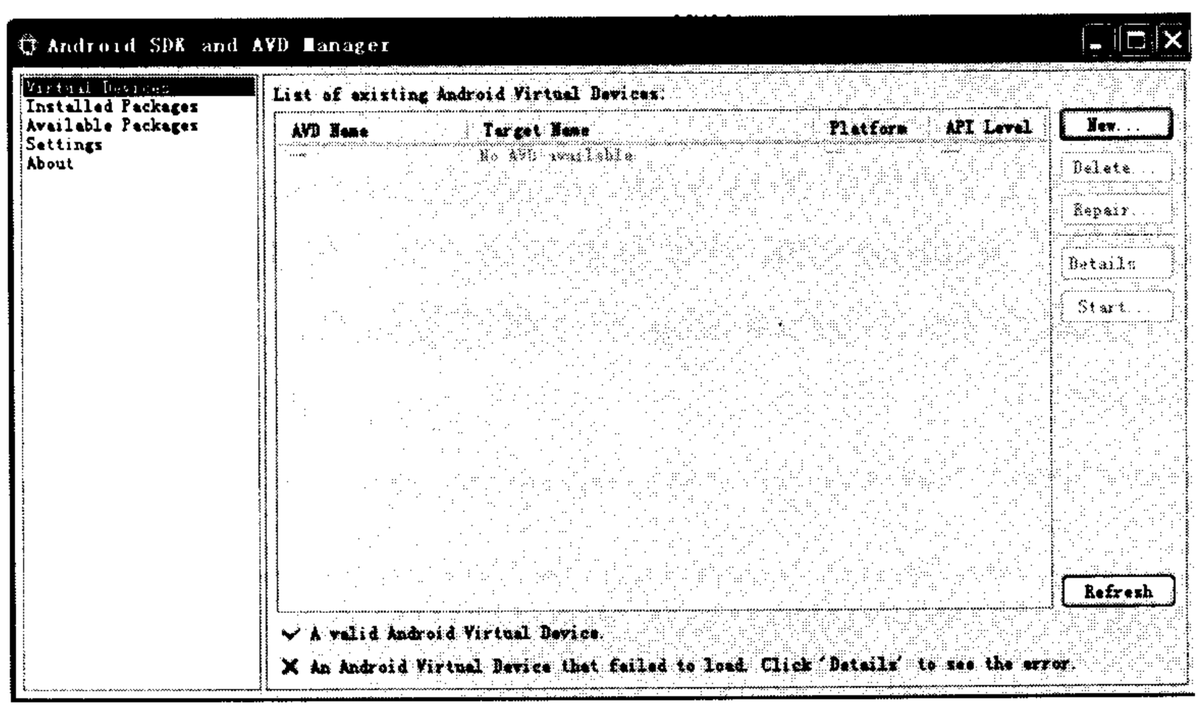


图 2.2　Android SDK 在线安装界面 1

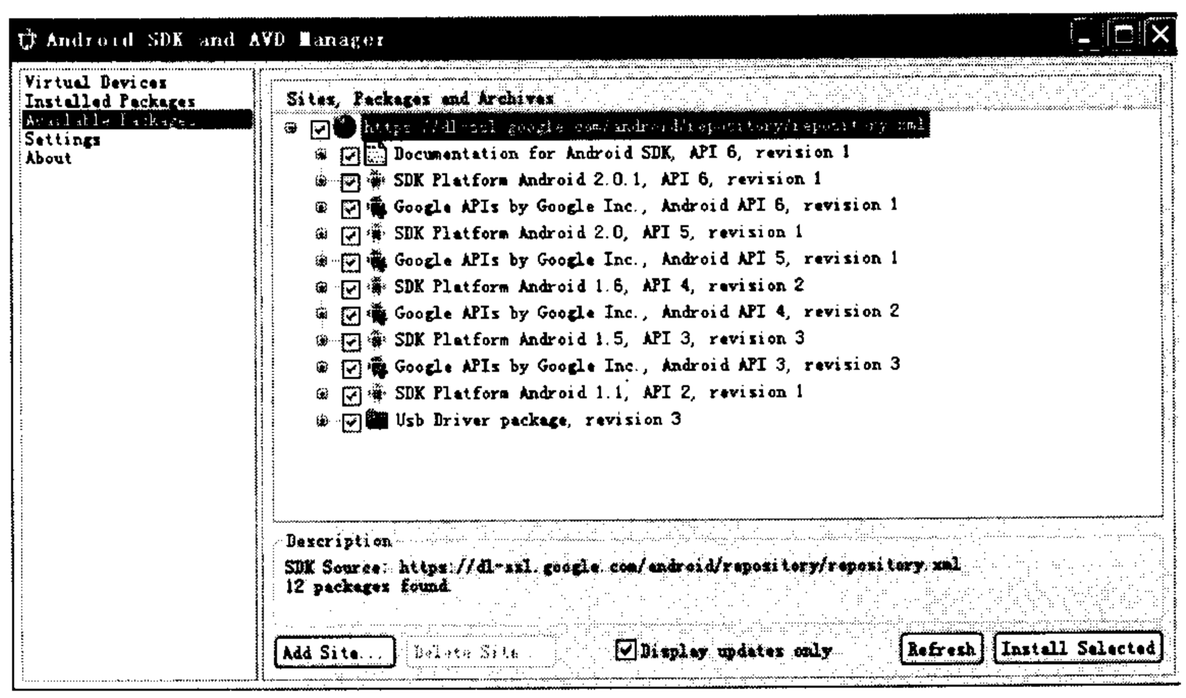


图 2.3　Android SDK 在线安装界面 2

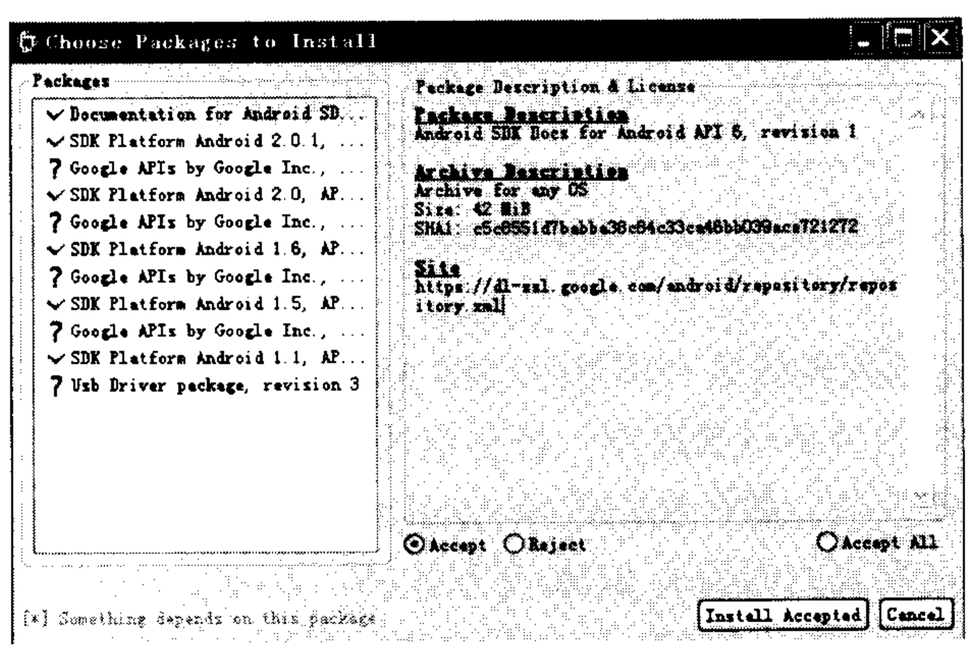


图 2.4　Android SDK 在线安装界面 3

选择“Accept All”单选按钮，再单击“Install Accepted”按钮就可以进行在线安装了。

这里需要注意的是，如果你的“Available Packages”右侧的地址不能打开，请选择左侧列表的“Settings”选项，在右侧的设置中勾选“Fore https://...”。设置界面如图 2.5 所示。

安装结束后的目录结构如图 2.6 所示。

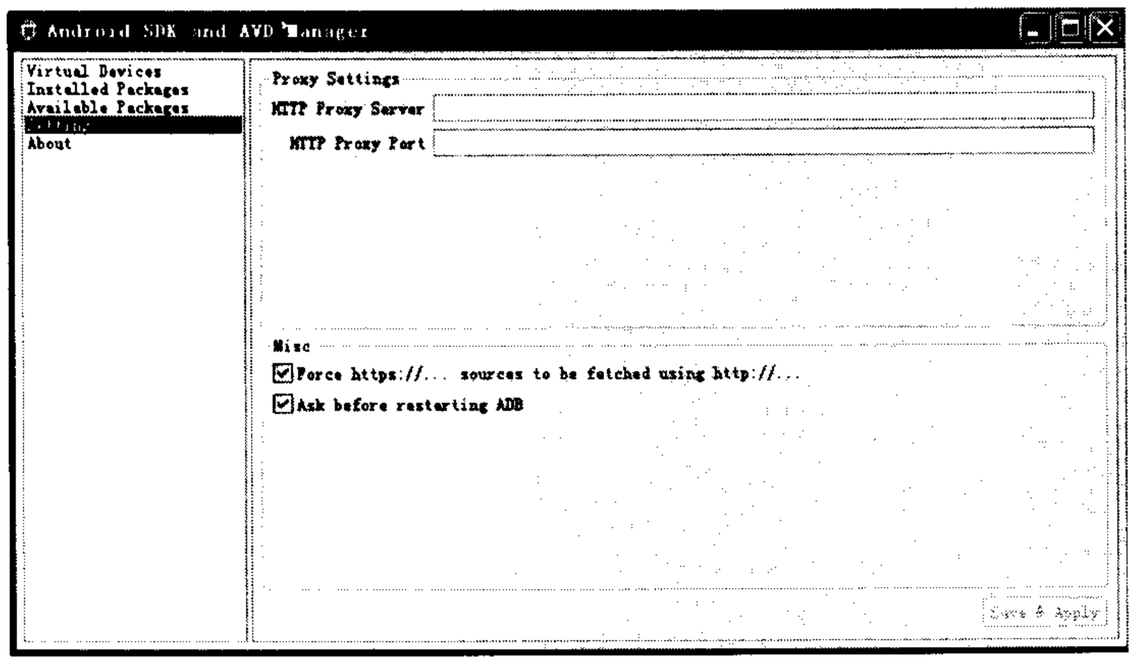


图 2.5　Android SDK 在线安装界面 4

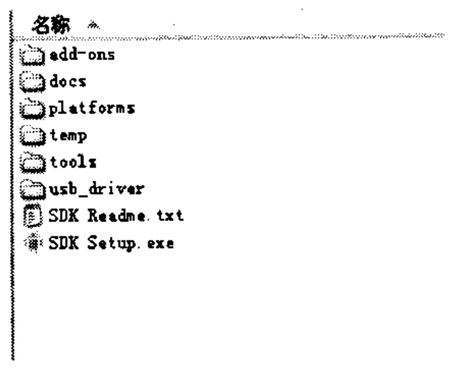


图 2.6　Android SDK 在线安装界面 5

Google API 在 add-ons 目录中，文档在 docs 目录中，不同版本的 SDK 在 platforms 目录中，tools 目录是 Android 的工具文件，usb_drivers 是 USB 驱动。

2.1.2　下载安装 JDK

在下载了 Android SDK 之后，要想进行开发还需要下载安装 Java 的开发环境，因为 Android SDK 的应用层使用 Java 语言，所以需要 Java 开发环境。有过 Java 开发经验的读者，对 JDK 的配置应该并不陌生。下面将介绍 JDK 的下载和安装。

① 下载 JDK。Android SDK 需要 JDK5 以上的版本。JDK6 的下载地址是 http://java.sun.com/javase/downloads/widget/jdk6.jsp。

② 安装 JDK。下载完成之后就可以进行 JDK 的安装，JDK 安装包是一个可执行文件，双击执行就可以了，这里不再赘述。

③ 配置 Java 环境变量。为了使用 Java 工具进行编译、运行，Java 程序需要配置 Java 路径 Path 和 classpath。

3.1 配置 Java Home。复制 Java 的安装路径，右键单击“我的电脑”→“属性”→“高级”→“环境变量”中新建环境变量 java_home，变量值为 Java 安装路径。配置过程如图 2.7 所示。

3.2 配置 Path。为了能够使用 Java 的编译、运行等命令工具，需要配置这些命令的 Path。右键单击“我的电脑”→“属性”→“高级”→“环境变量”，在“系统环境变量”中编辑 Path 变量，添加 Java 的 bin 目录到其中。变量中间使用分号“;”分隔。配置过程如图 2.8 所示。

3.3 配置 classpath。为了能够成功运行 Java 类，需要配置 Java 的类路径 classpath。右键单击“我的电脑”→“属性”→“高级”→“环境变量”，在“系统环境变量”中新建一个系统变量名称为“classpath”，变量值为半角句号“.”。配置过程如图 2.9 所示。

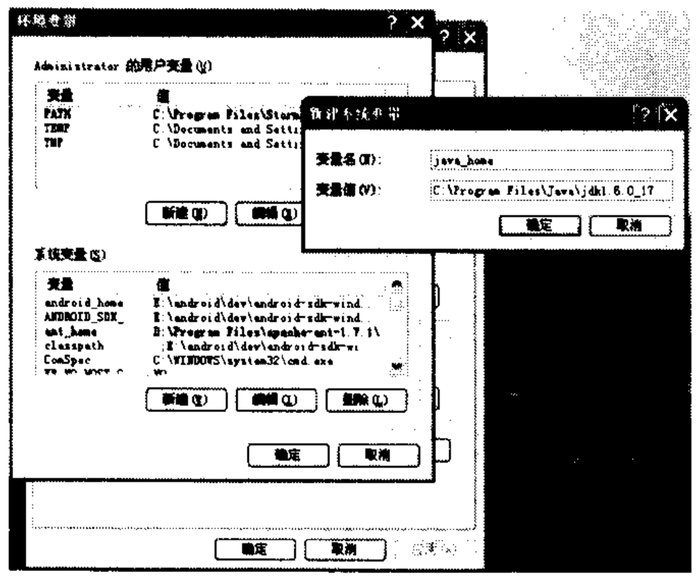


图 2.7　配置 Java Home

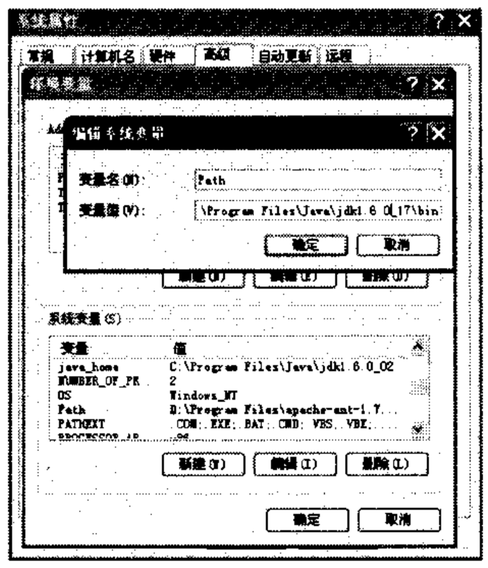


图 2.8　配置 Path

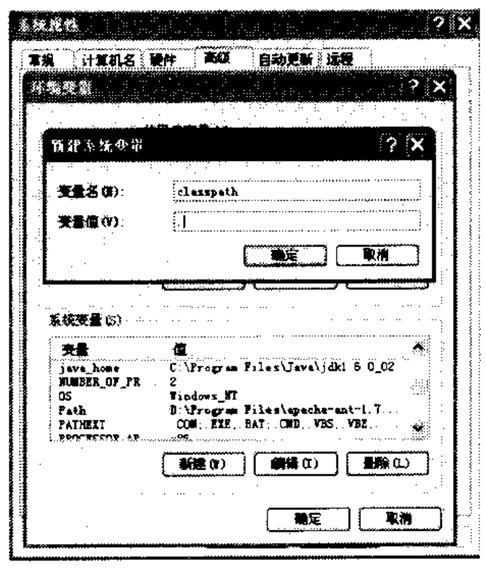


图 2.9　配置 classpath

2.1.3　下载 Eclipse

Google 提供 Android 的集成开发环境 Eclipse 的开发插件 Android Development Tools（ADT），为了使用该插件，首先需要下载并安装 Eclipse。

ADT 插件要求 Eclipse 的版本是 3.3 以上，Eclipse 的下载网址是 http://www.eclipse.org/downloads/。下载后解压可以直接使用 Eclipse，其运行界面如图 2.10 所示。

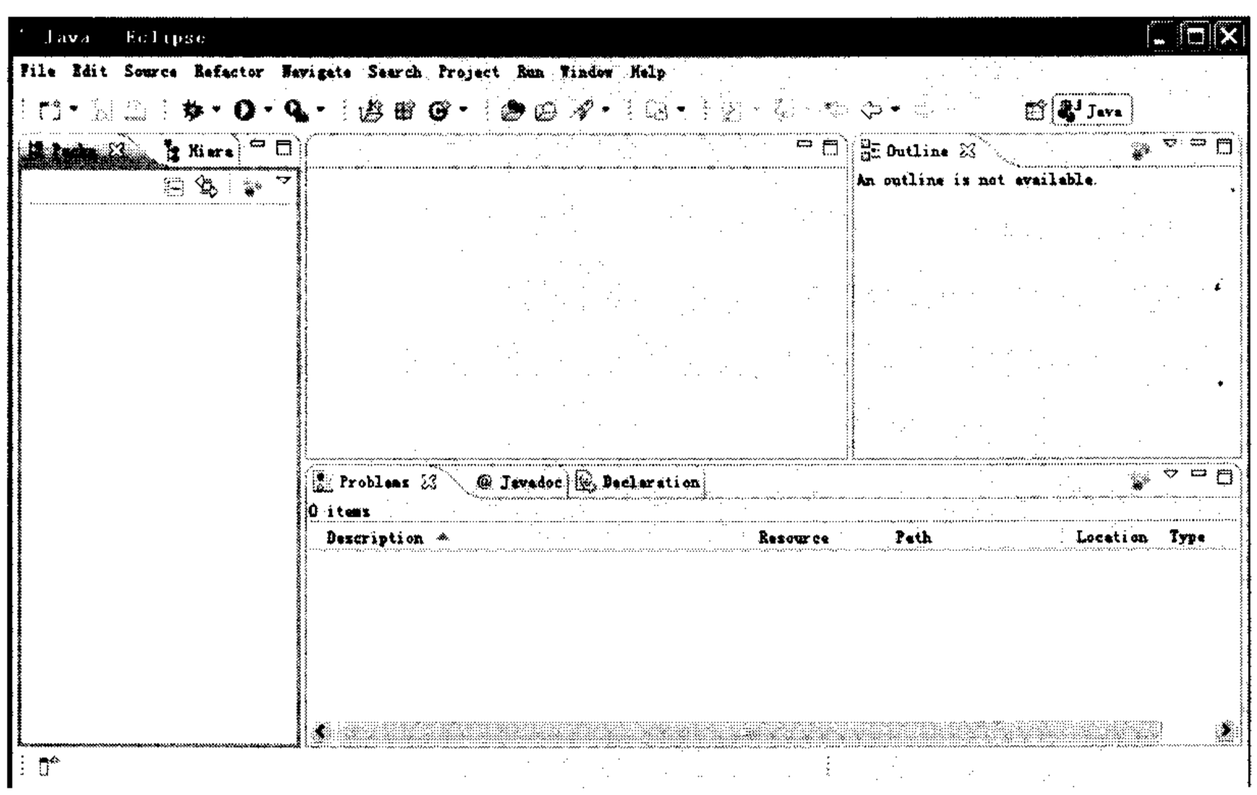


图 2.10　Eclipse 运行界面

2.1.4　下载安装 ADT

Google 公司提供了针对 Eclipse 的 Android 开发插件 ADT。通过 ADT 可以进行集成开

发，包括代码的自动生成、调试、编译、打包、拖曳式界面生成等功能。

ADT 的配置过程有两种：一种是通过 Eclipse 在线更新；另一种是下载 ADT 插件包手动配置。下面将分别介绍这两种配置过程。

1. 通过 Eclipse 在线更新

① 启动 Eclipse，选择“Help”→“Install New Software...”，弹出如图 2.11 所示的对话框。

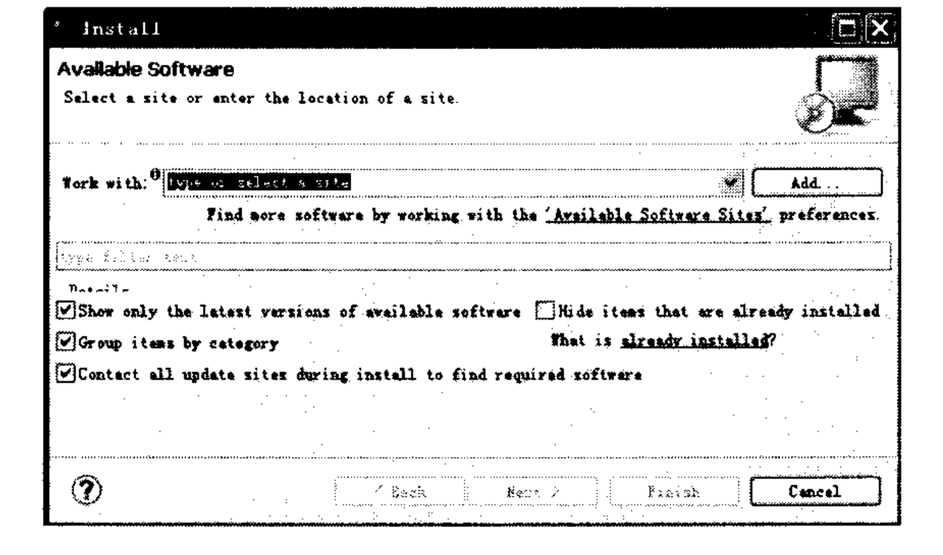


图 2.11　安装 ADT

② 单击“Add...”按钮，添加一个更新站点（注意：这里添加的站点 http 后不要加 s），否则可能连接失败，如图 2.12 所示。

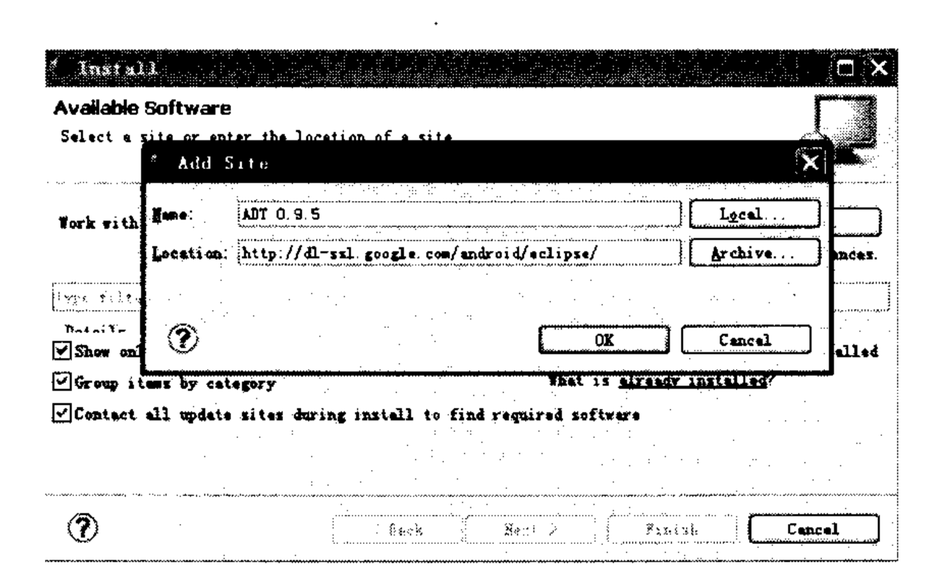


图 2.12　安装 ADT

③ 添加好后，勾选“Developer Tools”选项，单击“Next”按钮进行更新安装。

2. 下载 ADT 插件包手动配置

① 直接在 Android 的官方网站下载 ADT，下载地址是 http://dl-ssl.google.com/android/ADT-0.9.5.zip。

② 下载完成后，解压将 plugins 包和 features 包中的内容复制到 Eclipse 对应的 plugins 包和 features 包中，重新启动 Eclipse。

安装成功后，重新启动 Eclipse，选择 Window→Preferences 菜单，在弹出的对话框中多了一项“Android”。选择 Android 选项，在右边选择 Android SDK 的安装路径“SDK

Location”，下面会列出当前可用的 SDK 版本和 Google API 版本，如图 2.13 所示。

选择“File”→“New”命令，我们就可以使用该插件创建 Android 工程了，如图 2.14 所示。

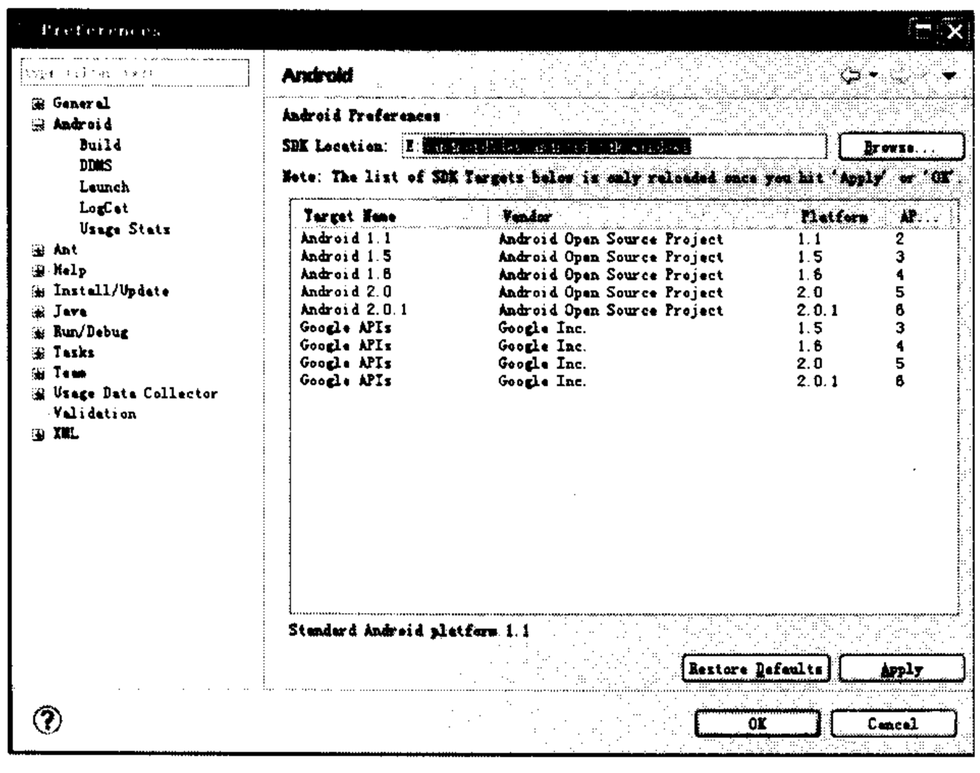


图 2.13　安装 ADT

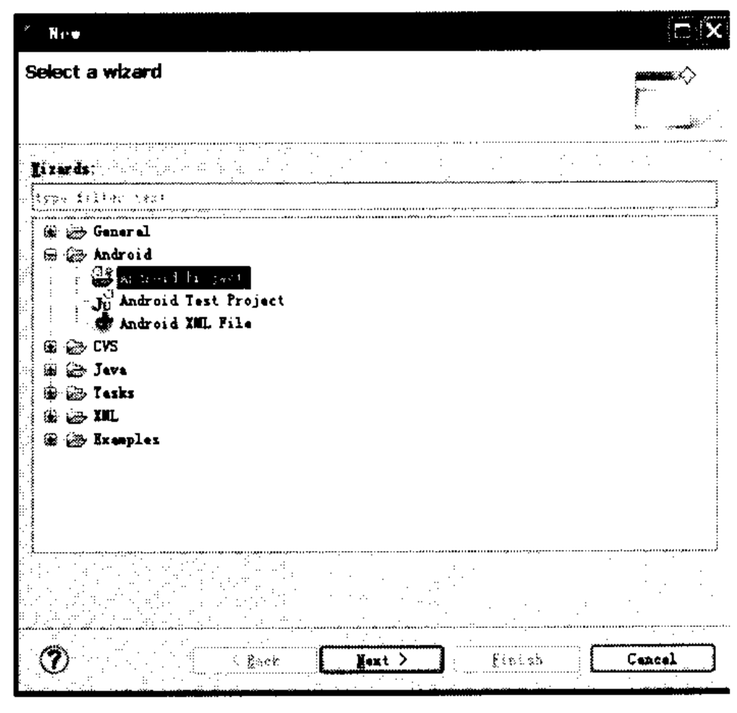


图 2.14　使用 ADT 创建 Android 项目

2.2 Android 常用工具的使用

在我们下载的 Android SDK 中有一个 tools 目录，该目录中包含了 Android 中的常用工具。另外，在 platforms 目录下针对不同的版本也有一个 tools 文件夹，该文件夹下面也有一些工具。例如，AATP、DX 等。其实，我们使用的 ADT 插件就是使用了这些工具来进行可视化开发的，所以了解这些工具的使用是非常有必要的。本节将详细讲述这些工具的使用。

2.2.1　创建 Android Virtual Devices（AVD）

1. 使用 DOS 命令行方式来创建

AVD 是模拟器的一个配置，让用户更好地来模拟真实设备。一个 AVD 包含如下几部分内容：一个硬件配置（如是否有照相机、键盘的指定和内存的大小等）、版本的选择、设备的屏幕尺寸和外观、SD 卡的大小，以及 AVD 文件的存储位置。

创建 AVD 我们使用 tools 目录下的 android 命令，打开 DOS 窗口将目录定位到 tools 目录下，按照如下命令格式输入：android create avd -n <name> -t <targetID> [-<option> <value>] ...

其中，n 表示 AVD 名称；-t 表示 AVD 使用的 SDK 版本。我们可以使用 android list target 命令列出当前可用的 SDK 版本。例如，我们在 DOS 下输入 android list target 命令，输出结果如图 2.15 所示。

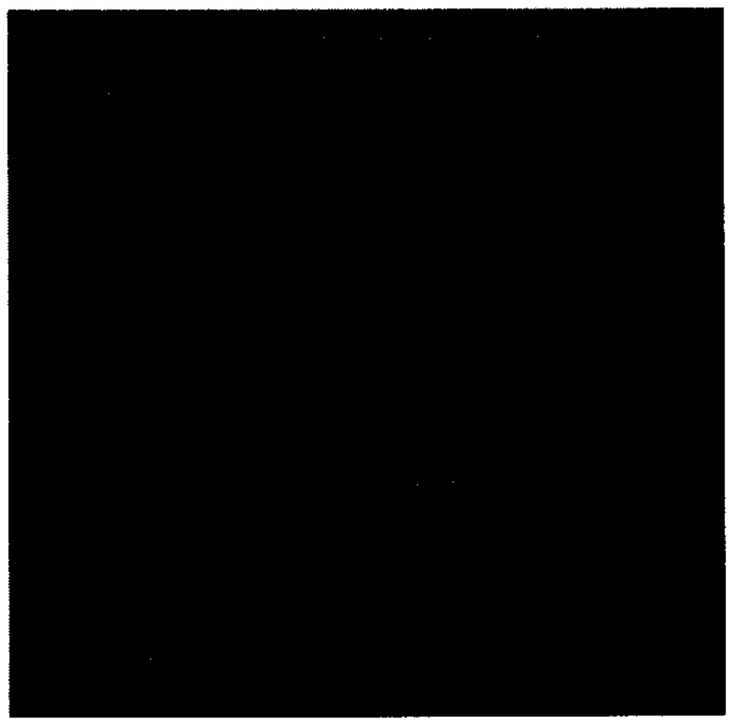


图 2.15　列出 Android SDK 版本信息

-option 是一些配置项信息。

例如，我们输入如下命令：android create avd -n MyAVD1.5 -t 2，系统将提示我们是否自定义一个硬件配置，我们选择 NO，将很快创建一个 AVD 配置。AVD 的保存位置会根据一个叫做“ANDROID_SDK_HOME”的环境变量来指定，如果没有指定该变量，则它会在系统默认的目录下创建。在 Windows 系统中的默认路径是 C:\Users\<user>\.android\。在这里我们配置的“ANDROID_SDK_HOME”环境变量是“E:\android\avds\.android\avd”，那么打开该目录，发现下面会有一个“.android”目录该目录中有一个“avd”目录，该目录下就是我们创建的 AVD 配置了。下面会有两个文件：config.ini（配置信息）和 userdata.img（用户数据镜像）。

2. 可视化配置

我们打开 DOS 窗口，将目录定位到 tools 目录下，输入 android 命令，将出现一个如图 2.16 所示的窗口。

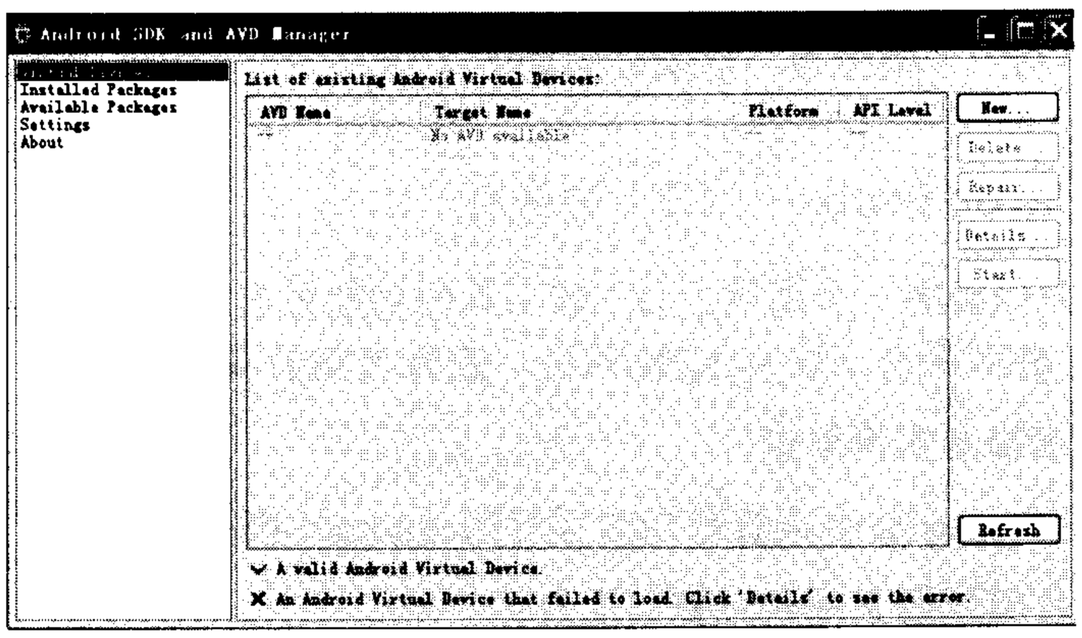


图 2.16　使用 android 命令创建 AVD1

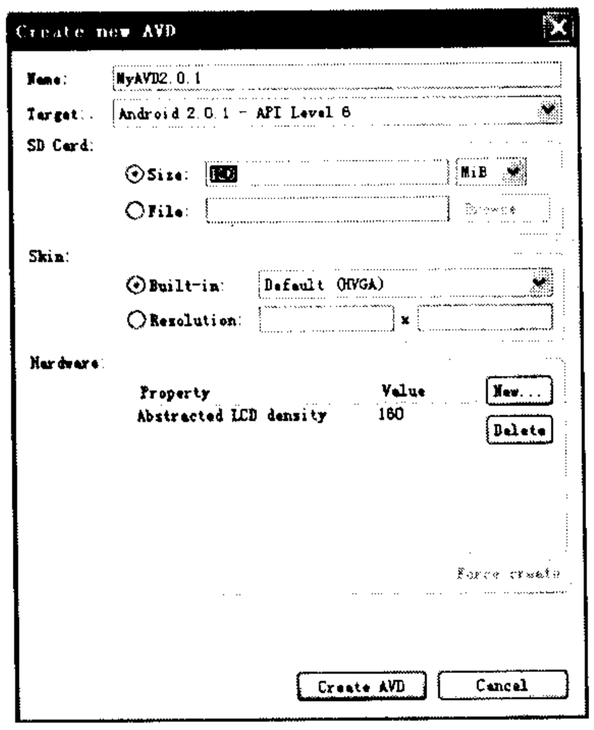


图 2.17　使用 android 创建 AVD2

单击“New”按钮弹出创建 AVD 对话框，如图 2.17 所示。

在该窗口中添加 AVD 名称、Target（Android 版本指定）、SD Card 大小和外观特征。单击“Create AVD”按钮将成功创建一个 AVD。

2.2.2　模拟器（Emulator）的使用

在 Android SDK 的 tools 目录下有一个 emulator.exe 可执行文件，该文件是硬件设备的一个仿真模拟器。我们可以通过该模拟器开发、测试 Android 应用程序。

Android 模拟器几乎提供了大多数物理硬件设备的硬件和软件特性。当然，它和真机还是有一些区别的。例如，它不能接打电话、不能进行拍照等。Emulator 通过一个叫做 Android Virtual Device（AVD，Android 虚拟设备）的配置来指定 Android 的版本、硬件选项和模拟器外观。

Android 通过 QEMU（QEMU 是一套由 Fabrice Bellard 编写的模拟处理器的自由软件）仿真模拟器来模拟 ARM 处理器，通过磁盘镜像来模拟存储。在 QEMU 的支持下，模拟器具有很多的硬件设备的特征：

- 支持 ARM5 处理器和响应的内存管理单元（MMU）；
- 支持一个 16 位的 LCD 显示屏；
- 一个或者多个键盘；
- 具有音频输入和输出的能力；
- 通过磁盘镜像使用闪存；
- 一个 GSM 调制解调器，包括一个模拟的 SIM 卡。

启动一个模拟器使用如下命令：emulator -avd <avd_name> 。avd_name 是 AVD 配置名称，例如，可以使用 emulator –avd MyAVD1.5 命令启动一个模拟器。启动界面如图 2.18 所示。

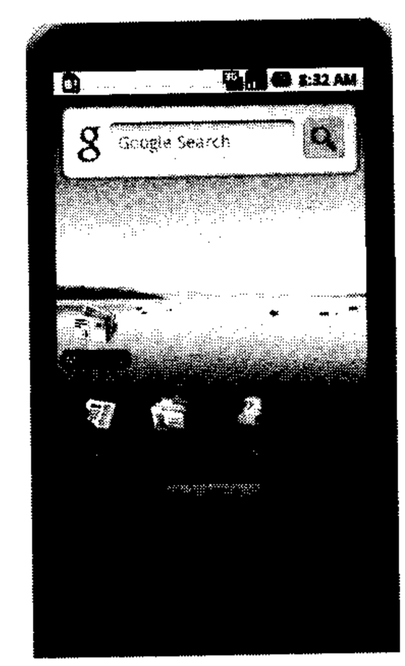


图 2.18　模拟器运行界面

2.2.3　Android Debug Bridge（ADB）的使用

ADB 是管理模拟器的一个通用工具，该工具的功能很多，例如，将系统文件复制到设备、从设备复制文件到系统、安装 APK 项目、查看当前设备等。

1．查询当前模拟器实例数量

可以使用 adb devices 命令来查看当前运行的 Android 模拟器实例。

2．本地机器和模拟器之间相互复制文件

可以使用 adb push 命令从系统复制文件到设备，使用 adb pull 命令从设备复制文件到

系统。例如，如下命令是将 D 盘根命令下面的一个 test.txt 文件复制到设备的 sdcard 里面。

```
D:\>adb push d:\test.txt /sdcard/
```

下面的命令是将设备中的 test.txt 文件复制到系统。

```
D:\>adb pull /sdcard/test.txt d:/
```

3. 安装 APK 应用程序

可以使用 adb install 命令安装一个 APK 应用程序到设备。注意在安装之前应先把应用程序复制到设备再进行安装。例如，如下命令可以安装一个名为 test.apk 的程序到设备。

```
adb push d:\test.apk /sdcard/
adb install test.apk
```

4. 使用 shell 命令

ADB 提供了一个 ash shell 允许使用系统中的各种命令，这些命令位于设备或者模拟器文件系统的/system/bin/目录下面。在 DOS 命令行输入 adb shell 便进入 shell 命令行了，这里我们可以使用 ls 来显示当前目录下的文件内容，可以使用 cd 来改变当前路径，也可以使用 exit 退出 shell。

2.2.4 Dalvik Debug Monitor Service（DDMS）的使用

DDMS 全称为 Dalvik Debug Monitor Service，即 Dalvik 调试监控服务，是一个可视化的调试监控工具。在 DOS 命令窗口中输入 ddms 会弹出如图 2.19 所示的界面。

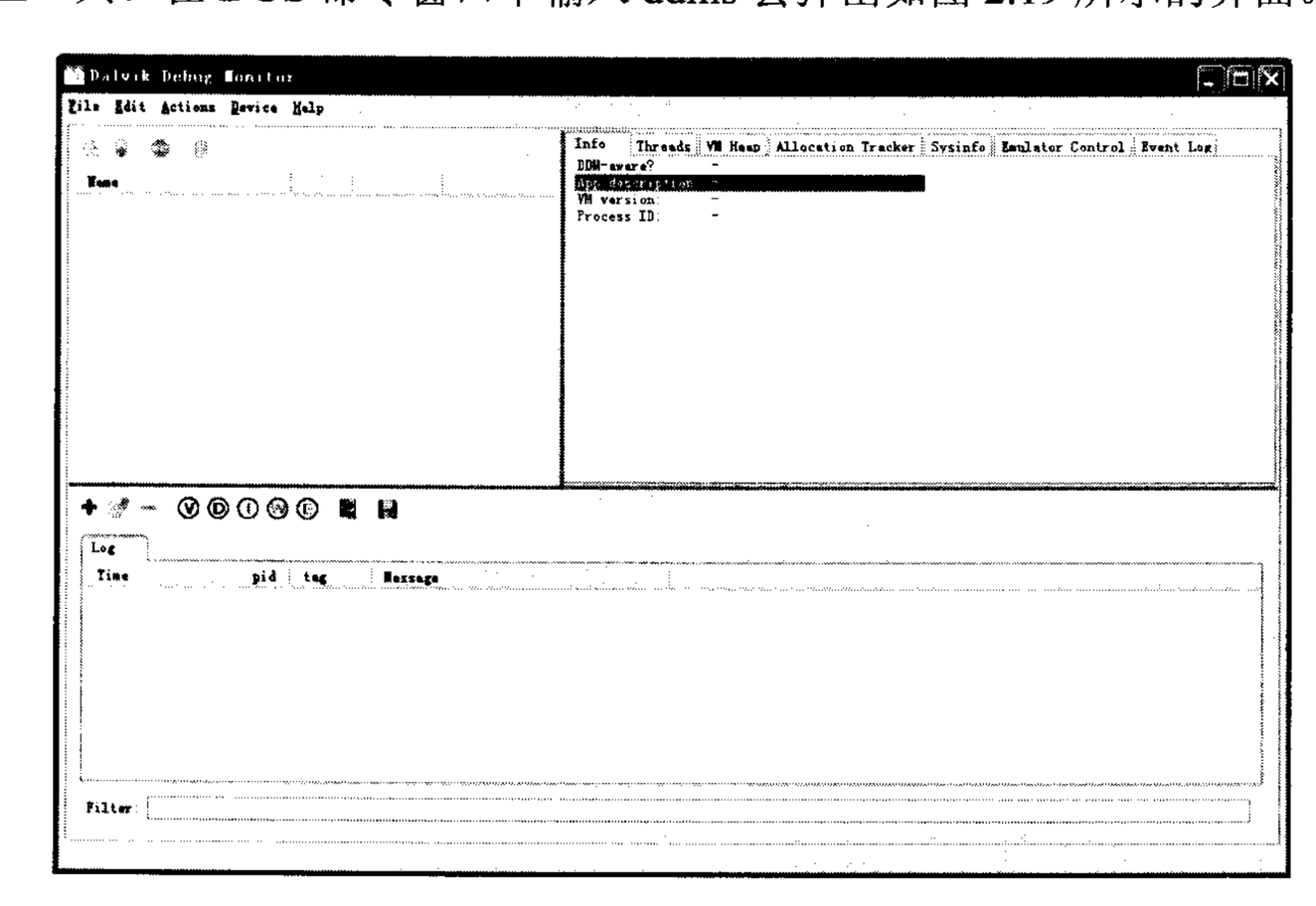


图 2.19　DDMS 窗口

该窗口主要是对系统运行后台日志的监控，还有系统线程、虚拟机状态的监控。另外，该窗口还可以模拟发送短信、拨打电话和发送 GPS 位置信息。我们后面在 Eclipse 中使用

的 DDMS 就是对该工具的集成。

2.2.5 Android Asset Packaging Tool（AAPT）的使用

AAPT 是 Android Asset Packaging Tool 的缩写，即 Android 资源打包工具，是对 Android 工程中的资源进行打包的工具。例如，音频、图片等原生文件；XML 配置文件等。下面的命令将 Android 工程下面的所有资源（包括 res 目录下面的、assets 目录下面的和 AndroidManifest.xml 配置文件）打包为\bin\目录下的 resources.ap_压缩文件。

```
aapt p -A assets -S res -M AndroidManifest.xml -F \bin\resources.ap_
```

2.2.6 DX 的使用

Android 应用程序虽然使用 Java 语言编写，但是不能直接使用编译后的类文件（.class）文件，而是需要编译成一种.dex 文件才能运行。dx 命令是将 Java 编译后的类文件（.class 文件）转换成 Android Dalvik 虚拟机可执行的文件，即.dex 文件（Dalvik Executable Format）。例如，如下命令将 D:\MyAndroidPro\bin 目录下面的所有类文件转换成 D:\MyAndroidPro\bin\下面的 classes.dex 文件。

```
dx --dex --dump-to=D:\MyAndroidPro\bin\classes.dex --core-library D:\MyAndroidPro\bin
```

2.2.7 mksdcard 的使用

我们都知道真正的 Android 设备当中是有数据存储设备的。例如，SDCard、扩展卡等。但是，我们使用 Android 模拟器时并没有真正的数据存储设备，这时候我们可以将硬盘空间镜像为数据存储设备。mksdcard 命令就能够将硬盘空间镜像为数据存储设备。例如，下面命令是在 D 盘根目录下创建一个名称为 mysdcard.img 且大小为 64MB 的存储卡。

```
mksdcard 64M d:\mysdcard.img
```

2.3 我的第一个 Android 应用

通过上面两节的学习，我们对 Android 中的常用命令及其开发环境有了一个整体的了解。接下来我们将开始创建第一个 Android 应用，为了使读者有一个更深入的了解，我们将通过两节来讲解 Android 的第一个应用。从而使读者能够了解到更多 IDE 背后的事情。

2.3.1 纯手工创建一个 Android 应用

因为当前我们还不是很熟悉 Android 的程序结构以及一些配置文件和类文件的结构，所以这里我们参考 Android SDK 中的一些模板文件，这些文件位于 Android SDK\platforms\android-1.5\templates 目录下面。

该工程的创建步骤说明如下。

① 创建包及其目录结构。在文件系统的 D 盘（任意）根目录下创建一个文件夹 MyAndroidPro。在该目录下再创建如下 src 目录及 res 目录：src 保存 Java 源文件，在该目录下创建包目录 com/amaker/app；res 保存资源文件，在该目录下创建 layout 保存布局文件，创建 values 保存一些常量值文件。目录结构如下所示。

```
MyAndroidPro
    |—src
        |—com
            |—amaker
                |—app
    |—res
        |—layout
        |—values
```

② 创建 Android 应用的 Activity 类。在 com.amaker.app 包中创建一个 MainActivity 类，该类继承 Activity，覆盖父类的 onCreate()方法，代码如下所示：

```
package com.amaker.app;

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

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

③ 创建界面布局文件。Android 中一种很好的设计思想就是可以把界面上的组件声明在配置文件当中，这样既降低了程序的耦合性，又提高了程序的运行效率。在 res\layout\ 目录下创建一个 main.xml 布局文件。该布局文件内参考 Android SDK\platforms\ android-1.5\templates 目录下面的 layout.template 文件，将该文件中的内容复制粘贴到 main.xml 文件中。

```
<?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>
```

该文件中的第一行是 XML 文件版本和编码。下面是该文件的布局管理类，现在使用的是线型布局。android:orientation="vertical"属性指定垂直布局，android:layout_width="fill_parent"和 android:layout_height="fill_parent"属性指定内容充满父窗口。下面是一个文本视图，显示一个字符串。android:layout_width="fill_parent"属性指定内容文本宽充满父窗口，android:layout_height="wrap_content"属性指定内容高依赖文本内容高，android:text="@string/hello"属性表示引用字符串常量显示内容。

④ 在 res/values/目录下创建一个 strings.xml 配置文件。该文件用来声明一些在程序中使用的字符串常量。

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

⑤ 在 com.amaker.app 包中创建一个 R.java 类，该类是 Java 源文件和引用资源之间的一个索引文件。该文件中针对不同的资源又创建了不同的内部类。例如，下面的 main 属性代表指向 main.xml 布局文件的一个索引，可以在 Java 源文件中引用。

```
package com.amaker.app;
public final class R {
   public static final class attr {
   }
   public static final class drawable {
   }
   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;
   }
}
```

⑥ 创建 Android 应用程序清单文件 AndroidManife.xml。Android 中的所有组件必须在该文件中声明后才能使用。另外，该文件也会声明一些权限信息和其他配置信息。在 MyAndroidPro 文件夹下创建一个名为 AndroidManifest.xml 的清单文件，这里我们可以参考 Android SDK\platforms\android-1.5\templates 目录下面的 AndroidManifest.template 文件。将给文件内容复制到 AndroidManifest.xml 文件中，修改相应属性如下所示：

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

```
            android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
  </application>
</manifest>
```

该文件的第一行是 XML 文件的版本和编码声明，下面是<mainfest>根元素，其中声明了命名空间、包名称和版本等信息。<application>子元素代表整个应用程序，其中的 android:label="@string/app_name"属性引用字符串常量，显示标签。<activity>元素是用来声明 Activity 组件的，其属性分别是类名称和标签。<intent-filter>子元素代表该 Activity 的访问能力，<action>和<category>代表当前的 Activity 是应用程序的入口程序。

⑦ 编译 MainActivity 和 R 类，并使用 DX 工具将 class 文件编译成 dex 文件，命令如下所示：

```
dx --dex --dump-to=D:\MyAndroidPro\bin\classes.dex --core-library D:\MyAndroidPro\bin
```

⑧ 使用 appt 命令打包资源文件。

```
aapt p -A assets -S res -M AndroidManifest.xml -F \bin\resources.ap_
```

⑨ 使用 apkbuilder 命令打包生成 apk 文件。

```
apkbuilder AndroidPro.apk -z resources.ap_ -f  classes.dex
```

⑩ 使用 emulator 命令启动模拟器。

```
emulator -avd MyAVD1.5
```

⑪ 使用 adb 命令安装 apk 文件。

```
adb push AndroidPro.apk /sdcard/
adb install AndroidPro.apk
```

⑫ 运行程序结果如图 2.20 所示。

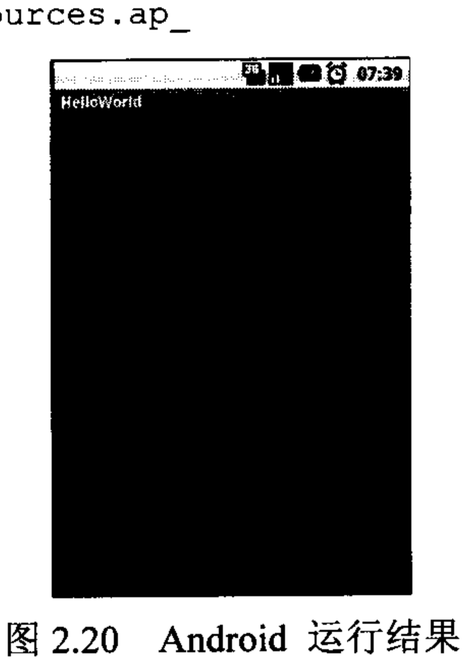


图 2.20　Android 运行结果

2.3.2　使用 Eclipse 创建一个 Android 应用

使用 Eclipse 插件 ADT 创建一个 Android 应用程序比较简单。下面我们将讲述如何使用 Eclipse 来创建一个 Android 应用程序。创建步骤说明如下。

① 启动 Eclipse，创建一个 Android 工程。我们选择 Eclipse 的“New”快捷方式将弹出创建工程对话框，如图 2.21 所示。

② 在弹出的对话框中选择“Android Project”并单击“Next”按钮进入创建 Android 工程属性选择对话框，如图 2.22 所示。

在 Project name 中输入 Android 项目名称，这里我们输入 Chaper02_HelloWorld。在 Build

Target 中选择你使用的 Android SDK 版本，这里我们选择 Android 1.5。在 Application name 中输入应用程序名称，这里我们输入 HelloWorld。在 Package name 中输入包名称，这里是 com.amaker.ch02.app。在 Create Activity 中输入 Activity 名称，这里我们输入 MainActivity。

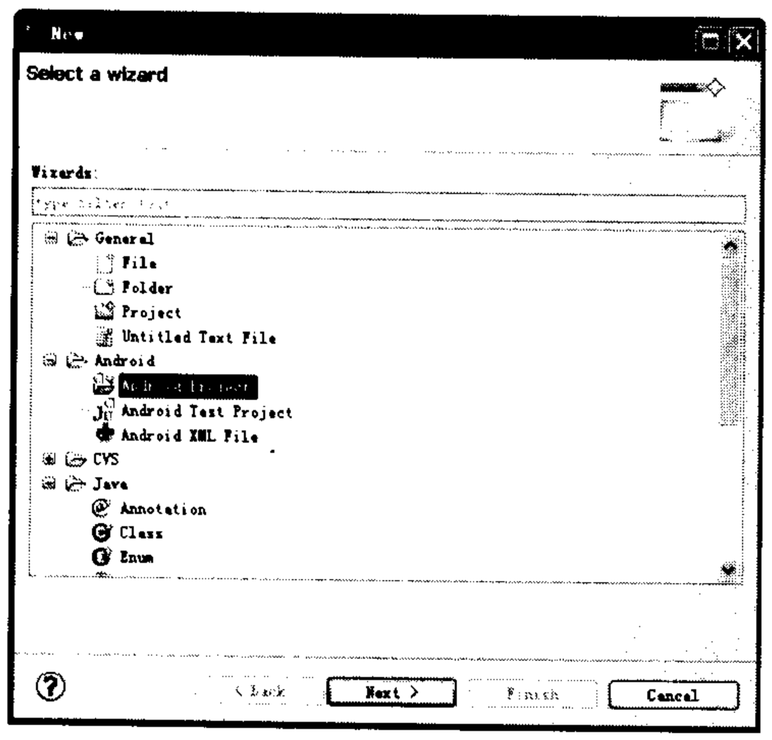


图 2.21　创建 Android 工程界面 1

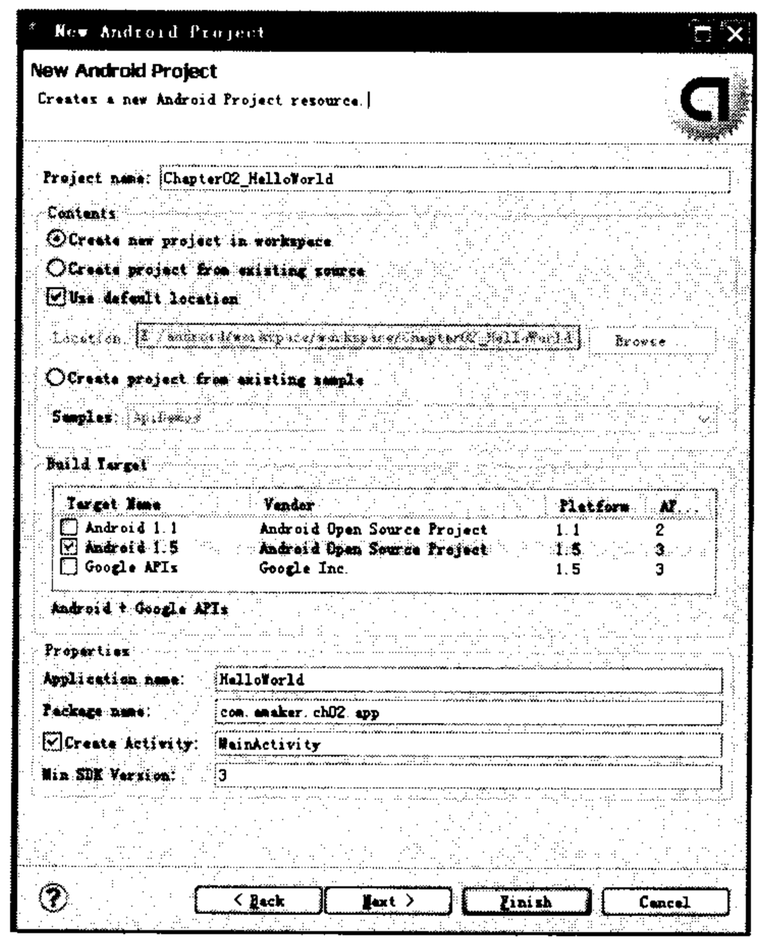


图 2.22　创建 Android 工程界面 2

③ 单击“Next”按钮进入下一个对话框，如图 2.23 所示。

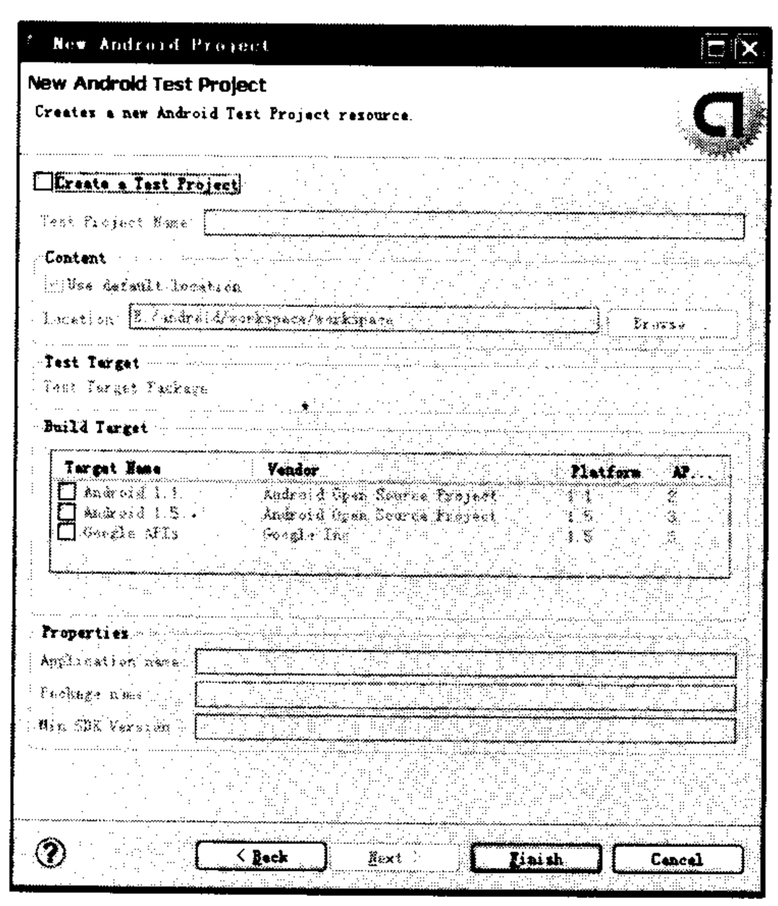


图 2.23　创建 Android 工程界面 3

该对话框中提示我们是否创建测试工程，这里我们暂时不创建测试工程，单击“Finish”按钮工程创建完成。

④ 右键单击工程，在弹出的菜单中选择“Run as”→“Android Application”命令，程序运行结果如图 2.24 所示。

如上所述，我们没有书写一句程序代码，一个 Android 应用便创建成功了。Android 应用程序如此简单吗？当然不仅如此而已。别忘了这只是一个“HelloWorld”。这里我们将详细讲述 Android 的应用程序结构，这样的 Android 应用程序结构在后续章节中会经常用到。所以掌握 Android 应用程序的基本结构还是非常必要的。

我们将上一节工程的目录结构展开，如图 2.25 所示。

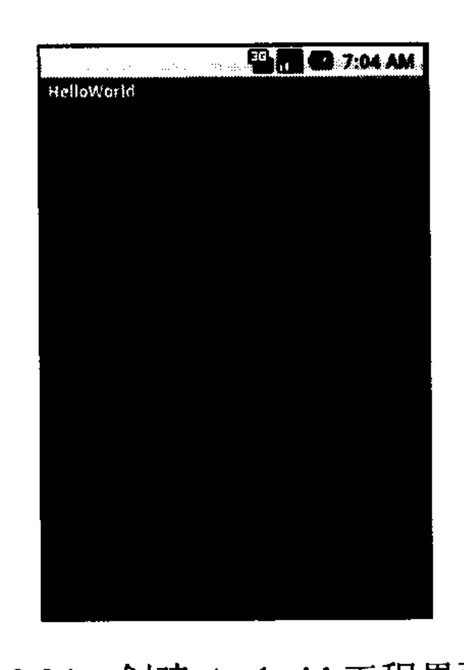


图 2.24　创建 Android 工程界面 4

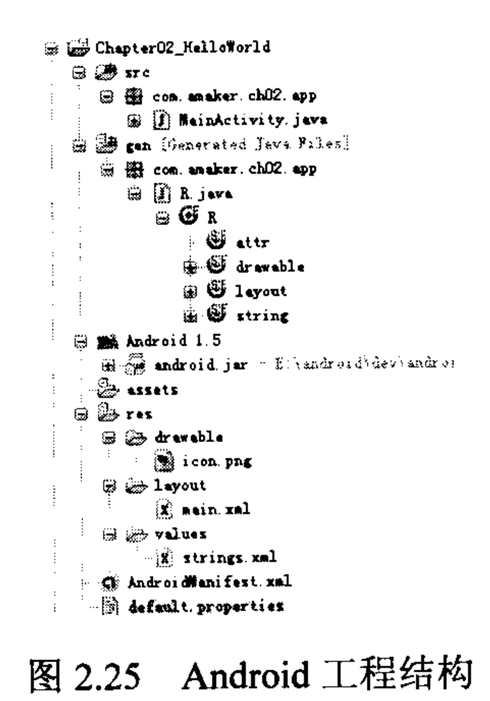


图 2.25　Android 工程结构

我们通过表 2.2 来分析 Android 工程的程序结构。

表 2.2　Android 应用程序结构

目录或文件名称	说　　明
Chaper02_HelloWorld	工程名
src	源文件文件夹
com.amaker.ch02.app	包名称
MainActivity.java	Activity 类
gen	保存自动生成的 R 资源类文件夹
gen->com.amaker.ch02.app	包名称
R.java	工程自动生成的资源索引类
Android 1.5->android.jar	Android SDK Jar 文件
assets	保存原始资源文件的文件夹
res	资源文件夹
drawable drawable-icon.png	保存图片等资源文件夹 图标文件
layout	界面布局文件夹
layout->main.xml	界面布局文件

续表

目录或文件名称	说　明
values	简单值配置文件夹
values->strings.xml	字符串配置文件
AndroidManifest.xml	Android 配置清单文件
default.xml	属性文件

接下来我们将重点分析程序的代码。

1. MainActivity.java 程序代码

```
package com.amaker.ch02.app;

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

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

该类比较简单，MainActivity 类继承了 Activity 类，并覆盖了 onCreate()方法，在该方法中调用父类构造方法，然后调用 setContentView()方法展示视图界面。R.layout.main 是 R.java 资源类中的一个属性。

2. R.java 程序代码

```
package com.amaker.ch02.app;

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;
    }
}
```

该类是一个资源索引类，由系统自动生成，无须修改。该类里面根据不同的资源类型又包含了不同的静态内部类，attr 中声明属性；drawable 中声明一些图片资源；layout 中声

明布局文件；string 中声明字符串。现在你应该明白了 setContentView（R.layout.main）这句话的意思了吧！它是通过访问资源类 R 的内部类 layout 的 main 属性来访问工程 layout 文件夹下的 main.xml 布局文件，在界面上展示视图组件。

3. string.xml 文件代码

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

该文件是一个 XML 文件，该文件中声明了系统中使用到的字符串常量。这样有两个好处：一是降低了程序的耦合性；二是 Android 通过一种特殊的方式来使用字符串，提高了程序的运行效率。在我们运行结果中显示的程序标题和文本内容都来自该文件。

4. main.xml 布局文件

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

该文件也是一个 XML 文件，该文件中声明了程序中使用到的视图组件。Android 通过这种巧妙的方法将程序的表现层和控制层分开，降低了程序的耦合性，提高了程序的可配置性。当然，我们也可以在程序中编码实现视图组件。

该文件的第一行是 XML 文件的版本和编码声明。第二行是一个线型布局，该布局以垂直或水平方式摆放视图组件。android:orientation="vertical"属性表明组件以垂直方式摆放；android:layout_width="fill_parent"属性表明布局的宽充满父组件；android:layout_height="fill_parent"表明布局的高充满父组件。下面是一个文本视图，android:layout_width="fill_parent"属性表明该组件的宽充满父组件；android:layout_height="wrap_content"属性表明组件的高依靠自身内容的高；android:text="@string/hello"属性表明文本内容引用 string.xml 文件中的 hello 元素。

5. AndroidManifest.xml 文件代码

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch02.app"
     android:versionCode="1"
```

```
  android:versionName="1.0">
<application
android:icon="@drawable/icon"
android:label="@string/app_name">

    <activity
        android:name=".MainActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

  </application>
  <uses-sdk android:minSdkVersion="3" />
</manifest>
```

每一个 Android 工程都有一个名为“AndroidManifest.xml”的配置文件，在所有项目中该文件的名称不变。该文件是 Android 工程的一个全局配置文件，所有 Android 中使用到的组件（如 Activity、Service、ContentProvider 和 Broadcast Receiver）都要在该文件中声明，并且该文件中还可以声明一些权限以及 SDK 的最低版本等信息。

该文件的第一行是 XML 文件版本和编码的声明。下面是 manifest 根元素，该元素中指定了命名空间、包名称、版本代码号和版本名称等信息。

application 子元素中的两个属性分别指定程序的图标和标题。下面是 Activity 组件的声明，Activity 有两个属性表明 Activity 的类名称和标题。

<intent-filter>是找到该 Activity 的过滤器，这里的 action 表明该 Activity 是程序的入口。category 属性表明在加载程序时运行。

<uses-sdk>表明使用的 SDK 最低版本。

我们通过对上述代码的分析，基本上了解了一个 Android 应用程序的基本结构和运行机制。

第二篇

技 术 篇

- 第 3 章　Android 中的资源访问
- 第 4 章　Android 用户界面
- 第 5 章　Android 基本程序单元 Activity
- 第 6 章　Android 组件之间的信使 Intent
- 第 7 章　Android Service 组件
- 第 8 章　Android 广播事件处理 Broadcast Receiver
- 第 9 章　Android 中的数据存取
- 第 10 章　Content Provider
- 第 11 章　Android 中的多媒体应用
- 第 12 章　Android 中的图形图像
- 第 13 章　Android 中的互联网应用
- 第 14 章　Android 中的 GPS 应用

第 3 章　Android 中的资源访问

概括地讲，Android 中的资源是指非代码部分。例如，在我们的 Android 程序中要使用一些图片来设置桌面，要使用一些音频文件来设置铃声，要使用一些动画来显示特效，要使用一些字符串来显示提示信息。那么，这些图片、音频、动画和字符串等就叫做 Android 中的资源文件。

我们在创建一个 Android 工程时，和 src 源文件夹并列的有两个文件夹，分别叫做 res 和 assets。这两个文件夹是用来保存资源文件的。在这两个文件夹中保存的文件有所不同，在 assets 中保存的一般是原生的文件，例如，一个 MP3 文件，Android 程序不能直接访问，必须通过 AssetManager 类以二进制流的形式来读取。而 res 中的资源可以通过 R 资源类直接访问。assets 中的资源很少用到，而 res 中的资源经常使用。如图 3.1 所示，在该工程中和 src 并列的有两个文件夹 assets 和 res，res 目录下又有 anim、drawable、layout、menu、raw、values 和 xml 文件夹，分别用来保存动画、图片、布局文件、菜单、原生文件、常量值和 XML 文件。

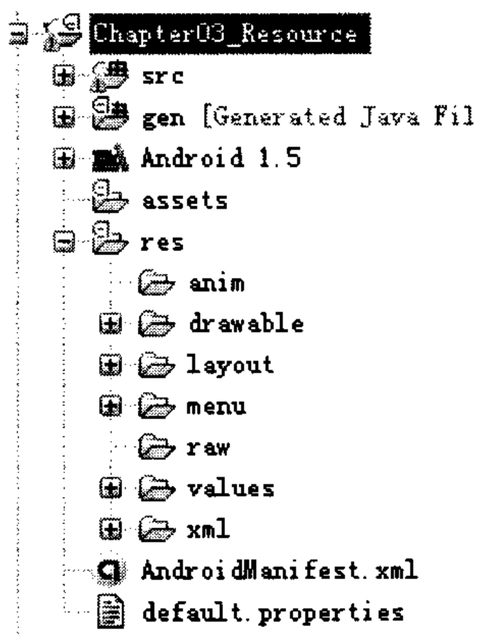


图 3.1　资源目录结构

3.1 资源简介

Android 中的资源是在代码中使用的外部文件。这些文件作为应用程序的一部分，被编译到应用程序当中。Android 中支持大量的资源文件，如 XML 文件、图片文件、音频和视频文件。XML 文件的格式有不同的写法，详细内容请参考本章后续小节。

本节我们将讲解如何创建资源文件，以及如何在代码中使用和如何在其他资源文件中引用该资源。在代码中我们使用 Context 的 getResources()方法得到 Resources 对象，该对象提供了获得各种类型资源的方法。在其他资源中引用资源的一般格式是这样的：@[包名称:]资源类型/资源名称。例如，下面的代码是在布局文件中引用颜色资源、字符串资源和尺寸资源：

```
<TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/styled_welcome_message"
    android:textColor="@color/opaque_red"
    android:textSize="@dimen/sixteen_sp" />
```

3.1.1 资源的类型和布局

我们可以在工程的 res 目录下创建和保存各种不同类型的资源文件。这些资源文件要有一定的布局和格式。Android 使用资源编译器工具 AAPT 来编译这些资源文件。表 3.1 展示了 Android 中资源的布局和类型。

表 3.1　Android 资源布局类型表

目 录 结 构	资 源 类 型
res/anim/	XML 动画文件
res/drawable/	一些位图文件
res/layout/	XML 布局文件
res/values/	各种 XML 资源文件 arrays.xml：XML 数组文件 colors.xml：XML 颜色文件 dimens.xml：XML 尺寸文件 styles.xml：XML 样式文件
res/xml/	任意的 XML 文件
res/raw/	直接复制到设备中的原生文件
res/menu/	XML 菜单文件

3.1.2 资源文件的使用

资源文件的使用分为在代码中使用和在其他资源文件中引用该资源文件。在我们编译

一个 Android 应用时，Android 会自动生成一个 R 类，在该类中根据不同的资源类型又生成了相应的内部类，该类包含了系统中使用到的所有资源文件的标示，其内容如下所示。

```
package com.amaker.mp;
// 资源类
public final class R {
     // 数组
    public static final class array {
        public static final int faultRecords=0x7f060000;
    }
    // 属性
    public static final class attr {
    }
    // 颜色
    public static final class color {
        public static final int black=0x7f040001;
        public static final int red=0x7f040000;
    }
    // 图片
    public static final class drawable {
        public static final int icon=0x7f020001;
        public static final int logo2=0x7f020002;
   }
    // ID 标示
    public static final class id {
        public static final int licenseEditText=0x7f070022;
        public static final int lngEditText=0x7f070001;
    }
    // 布局
    public static final class layout {
        public static final int custom_dialog=0x7f030000;
       public static final int custom_dialog1=0x7f030001;
    }
    // 字符串
    public static final class string {
        public static final int app_name=0x7f050001;
        public static final int hello=0x7f050000;
    }
}
```

1. 在代码中使用资源文件

在代码中访问资源文件，我们是通过使用 R 资源类中定义的资源文件类型和资源文件名称来访问的。具体格式为：R.资源文件类型.资源文件名称。例如：

```
// 设置 Activity 显示的布局视图
setContentView(R.layout.login_system);
// 获得 Button 实例
cancelBtn = (Button)findViewById(R.id.cancelButton);
loginBtn = (Button)findViewById(R.id.loginButton);
// 获得 TextView 实例
userEditText = (EditText)findViewById(R.id.userEditText);
```

```
        pwdEditText = (EditText)findViewById(R.id.pwdEditText);}
```

另外，除了访问用户自己定义的资源文件，还可以访问系统中的资源文件。大部分的资源文件被定义在 android 包下的 R 类中。访问系统中的资源文件格式为：android.R.资源文件类型.资源文件名称。例如：

```
int i ;
// 动画
i = android.R.anim.fade_in;
// 数组
i = android.R.array.emailAddressTypes;
// 颜色
i = android.R.color.darker_gray;
// 尺寸
i = android.R.dimen.app_icon_size;
// 可绘制图片
i = android.R.drawable.title_bar;
// 字符串
i = android.R.string.cancel;
```

2. 在其他资源文件中引用资源文件

我们经常会在布局文件的元素属性中引用其他资源文件，经常用到的有字符串、图片、颜色等资源。例如，下列布局文件中的 TextView 组件引用了一个字符串来表示文本内容，引用了一个颜色来表示文本颜色，引用了一个尺寸来表示文本尺寸。

```
<TextView android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/styled_welcome_message"
android:textColor="@color/opaque_red"
android:textSize="@dimen/sixteen_sp" />
```

3.2 使用颜色（color）资源

3.2.1 颜色值定义

颜色值的定义是通过 RGB 三原色和一个 alpha 值来定义的。颜色值定义的开始是一个井号（#），后面是 Alpha-Red-Green-Blue 的格式。例如：

#RGB

#ARGB

#RRGGBB

#AARRGGBB

3.2.2 颜色资源 XML 文件的定义

下面通过一个表格来展示颜色资源 XML 文件的定义，如表 3.2 所示。

表 3.2　颜色的定义和使用

资 源 位 置	res/values/colors.xml
颜色 XML 文件格式	使用<?xml version="1.0" encoding="utf-8"?> <resources>根元素 <color>子元素：<color name=color_name>#color_value</color>
获得颜色方法	Resources.getColor()
引用资源格式	Java 代码中：R.color.color_name XML 文件中：@[package:]color/color_name

3.2.3　使用颜色资源

下面通过一个实例来演示颜色资源的使用，本实例的功能是定义两个颜色资源：红色背景和蓝色文字。实例步骤说明如下。

① 定义一个名称为“Chapter03_Resource”的 Android 工程，在该工程的 res\values\目录下，定义一个 colors.xml 颜色资源文件，内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="red_bg">#f00</color>
   <color name="blue_text">#0000ff</color>
</resources>
```

② 在该工程的 res\layout\目录下定义一个布局资源文件，在该文件中添加一个 TextView 视图组件，引用颜色资源，设置视图组件 TextView 的文字颜色为蓝色。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">
<TextView
   android:text="测试颜色资源，红色背景，蓝色文字"
   android:id="@+id/TextView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textColor="@color/blue_text"
   />
</LinearLayout>
```

③ 定义一个 TestColorActivity 类，引用颜色资源文件，设置背景色为红色。

```
package com.amaker.ch03.color;
import android.app.Activity;
import android.os.Bundle;
import com.amaker.test.R;

public class TestColorActivity extends Activity {
   @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.test_color);
        // 引用颜色资源，设置背景色为红色
        getWindow().setBackgroundDrawableResource(R.color.red_bg);
    }
}
```

④ 运行程序，结果如图 3.2 所示。

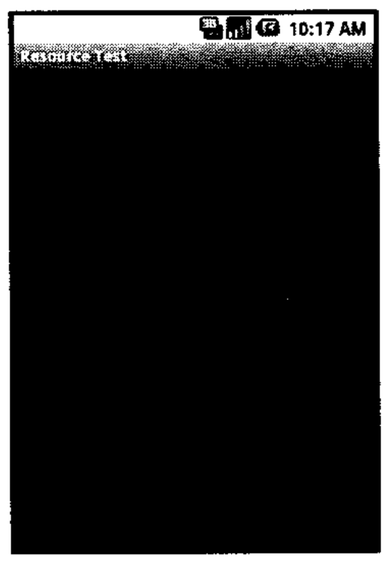


图 3.2　颜色资源应用

3.3 使用字符串（string）资源

在一个 Android 工程中，我们可能会使用到大量的字符串作为提示信息。这些字符串都可以作为字符串资源声明在配置文件中，从而实现程序的可配置性。

在代码中我们使用 Context.getString()方法，通过传递资源 ID 参数来得到该字符串，也可以在其他资源文件中引用字符串资源，引用格式为："@string/字符串资源名称。

3.3.1　字符串资源 XML 文件的定义

我们通过表 3.3 来说明字符串资源是如何定义的，包括资源的位置、XML 文件的格式、获得资源的方法和引用资源的方法等。

表 3.3　字符串资源得定义和使用

资 源 位 置	res/values/strings.xml
字符串 XML 文件格式	使用<?xml version="1.0" encoding="utf-8"?> <resources>根元素 <string>子元素：<string name=color_name>string_value</string>
获得字符串资源的方法	Resources.getString()
引用字符串资源的格式	Java 代码中：R.string.string_name XML 文件中：@[package:]string/string_name

3.3.2 字符串资源 XML 文件的使用

下面将通过一个实例来演示资源文件的用法。在该实例中用到两个字符串资源：一个在布局文件中引用；另一个在 Java 代码中引用。实例步骤说明如下。

① 在该工程的 res\values\目录下，创建一个字符串资源文件 stirngs.xml，内容如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Test Resources</string>
    <string name="test_str1">从代码中引用！ </string>
    <string name="test_str2">从资源文件引用！ </string>
</resources>
```

② 在该工程的 res\layout\目录下，定义一个布局文件 test_string.xml。在该布局文件中添加两个 TextView 视图对象：第一个 TextView 的文本内容直接引用 strings.xml 文件中的资源；第二个 TextView 的文本内容在代码中设置。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:text="@string/test_str1"
        android:id="@+id/myTextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <TextView
    android:text=""
    android:id="@+id/myTextView02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

③ 在工程的 com.amaker.ch03.string 包中，创建一个 TestStringActivity 类。在该类的 onCreate()方法中，设置当前的视图布局，并获得 TextView 实例。通过 Context.getString()方法，从字符串资源中获得字符串常量，并将其设置为 TextView 的文本内容。

```
package com.amaker.ch03.string;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import com.amaker.test.R;
/**
 *
```

```
 * @author 郭宏志
 * 测试字符串资源
 */
public class TestStringActivity extends Activity {
   private TextView myTextView;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
      setContentView(R.layout.test_string);
      myTextView = (TextView)findViewById(R.id.myTextView02);

      String str = getString(R.string.test_str2).toString();
      myTextView.setText(str);
   }
}
```

④ 运行程序，结果如图 3.3 所示。

图 3.3　字符串资源应用

3.4 使用尺寸（dimen）资源

我们可以使用一些常用的尺寸单位来定义一些文字尺寸、视图组件的宽和高等。尺寸资源是一个数字类型的数据，被定义在 res\values\dimens.xml 文件中。

3.4.1 Android 中支持的尺寸单位

在计算机中一般我们会用到的尺寸单位有厘米（cm）、毫米（mm）、像素（px）、英尺（in）等。Android 中支持的尺寸单位如表 3.4 所示。

表 3.4　Android 中支持单位

单位表示	单位名称	单位说明
px	像素	屏幕上的真实像素表示
in	英尺	基于屏幕的物理尺寸

续表

单位表示	单位名称	单位说明
mm	毫米	基于屏幕的物理尺寸
pt	点	英尺的 1/72
dp	和密度无关的像素	相对屏幕物理密度的抽象单位
sp	和精度无关的像素	和 dp 类似

3.4.2 尺寸资源 XML 文件的定义

尺寸资源是定义在 XML 文件中的一些整型数值。有关尺寸资源的定义如表 3.5 所示。

表 3.5 尺寸资源的定义和使用

资源位置	res/values/dimens.xml
尺寸 XML 文件格式	使用<?xml version="1.0" encoding="utf-8"?> <resources>根元素 <dimen>子元素：<dimen name=dimen_name>dimen_value</dimen>
获得尺寸资源的方法	getResources().getDimension()
引用尺寸资源的格式	Java 代码中：R.dimen.dimen_name XML 文件中：@[package:]dimen/dimen_name

3.4.3 尺寸资源 XML 文件的使用

下面还是通过一个实例来演示尺寸资源的用法。该实例在布局文件中添加一个 TextView 和一个 Button，分别使用尺寸资源文件来定义它们的宽和高。

① 在工程的 res\values\目录下创建一个 dimens.xml 尺寸资源文件。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <dimen name="text_width">150px</dimen>
   <dimen name="text_height">100px</dimen>
   <dimen name="btn_width">30mm</dimen>
   <dimen name="btn_height">10mm</dimen>
</resources>
```

② 在工程的 res\layout\目录下创建一个 test_dimen.xml 布局文件。在该布局文件中添加一个 TextView 和一个 Button。TextView 的宽和高引用尺寸资源来设置。Button 的宽和高在代码中设置。

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

   <TextView
```

```
        android:text="@string/test_dimen"
        android:id="@+id/myDimenTextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="@dimen/text_width"
        android:height="@dimen/text_height"
        android:background="@color/red_bg"
        />

    <Button
        android:text="@string/test_dimen1"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 在 com.amaker.dimen 包中，创建一个 TestDimensionActivity 类。在该类顶部声明使用的 Button 视图组件，在 onCreate()方法中实例化该组件，并定义尺寸资源设置其宽和高。

```
package com.amaker.ch03.dimen;

import android.app.Activity;
import android.content.res.Resources;
import android.os.Bundle;
import android.widget.Button;

import com.amaker.test.R;
/**
 *
 * @author 郭宏志
 * 测试尺寸资源
 */
public class TestDimensionActivity extends Activity {
    private Button myButton;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置当前 Activity 的内容布局视图
       setContentView(R.layout.test_dimen);
       // 通过 findViewById 方法获得 Button 实例
       myButton = (Button)findViewById(R.id.Button01);
       // 获得 Resources 实例
       Resources r = getResources();
      // 通过 getDimension 方法获得尺寸值
       float btn_h = r.getDimension(R.dimen.btn_height);
       float btn_w = r.getDimension(R.dimen.btn_width);
       // 设置按钮的宽
       myButton.setHeight((int)btn_h);
      // 设置按钮的高
       myButton.setWidth((int)btn_w);
    }
```

```
}
```

④ 运行程序，结果如图 3.4 所示。

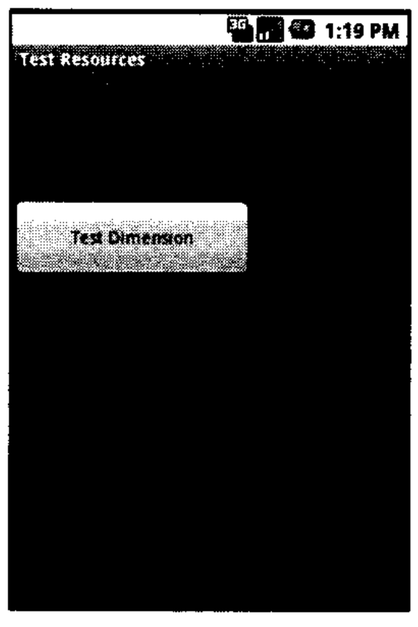


图 3.4　尺寸资源应用

3.5 使用原始 XML 资源

如果项目中使用到了一些原始的 XML 文件，那么，我们可以定义一些 XML 文件供工程使用。XML 文件定义在工程的 res\xml\目录下，通过 Resources.getXML()方法来访问。

3.5.1 原始 XML 资源文件的定义

原始 XML 资源文件的定义信息通过表 3.6 来描述。

表 3.6　原始 XML 资源文件定义和使用

资 源 位 置	res/xml/test.xm（文件名称任意）
原始 XML 文件格式	使用<?xml version="1.0" encoding="utf-8"?> <resources>根元素 <someElement>子元素：<someElement name=value/>子元素及属性名称任意
获得 XML 资源的方法	getResources().getXml()
引用 XML 资源的格式	Java 代码中：R.xml.xml_name

3.5.2 原始 XML 文件的使用

获得原始 XML 文件的基本思路是，通过 getResources().getXml()获得 XML 原始文件，得到 XmlResourceParser 对象，通过该对象来判断是文档的开始还是结尾、是某个标签的开始还是结尾，并通过一些获得属性的方法来遍历 XML 文件，从而访问 XML 文件的内容。下面的实例演示了如何访问 XML 文件内容，并将内容显示在一个 TextView 中。

① 在“Chapter03_Resource”工程中的 res\xml\目录下，创建一个 test.xml 文件。该

文件中定义了两条客户信息，属性信息有姓名、年龄、性别和 E-mail 地址，内容如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <customer name="tom" age="20" gender="male" email="tom@yahoo.com"/>
   <customer name="kite" age="21" gender="male" email="kite@yahoo.com"/>
</resources>
```

② 在该工程的 res\layout\目录下，创建一个 test_xml.xml 布局文件，该布局文件中添加一个 Button 和一个 TextView 视图组件。Button 用于响应单击事件访问 XML 内容，TextView 用于显示 XML 内容。

```
<?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:text="获得 XML 内容"
   android:id="@+id/xmltTestButton01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></Button>

   <TextView
   android:text=""
   android:id="@+id/xmlContentTextView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   />

</LinearLayout>
```

③ 在该工程的 com.amaker.ch03.xml 包中创建一个 TestXmlActivity 类。在该类顶部声明使用到的 TextView 和 Button，在 onCreate()方法中实例化，添加 Button 的单击事件，获得 XML 内容显示在 TextView 中。

```
package com.amaker.ch03.xml;

import java.io.IOException;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

import android.app.Activity;
import android.content.res.Resources;
import android.content.res.XmlResourceParser;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
```

```
import com.amaker.test.R;
/**
 *
 * @author 郭宏志
 * 测试 xml 资源
 */
public class TestXmlActivity extends Activity {
    private TextView myTextView;
    private Button myButton;
    @Override
    public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     // 设置当前 Activity 的内容布局视图
    setContentView(R.layout.test_xml);
     // 通过 findViewById 方法获得 TextView 实例
    myTextView = (TextView)findViewById(R.id.xmlContentTextView01);
     // 通过 findViewById 方法获得 Button 实例
    myButton = (Button)findViewById(R.id.xmltTestButton01);
     // 设置按钮的单击事件监听器
    myButton.setOnClickListener(new OnClickListener() {
     @Override
     public void onClick(View v) {
          // 定义计数器
          int counter = 0;
          // 实例化 StringBuilder
          StringBuilder sb = new StringBuilder("");
          // 获得 Resources 实例
          Resources r = getResources();
          // 通过 Resources，获得 XmlResourceParser 实例
          XmlResourceParser xrp = r.getXml(R.xml.test);
          try {
               // 如果没有到文件尾继续循环
               while (xrp.getEventType() != XmlResourceParser.END_DOCUMENT) {
                    // 如果是开始标签
                    if (xrp.getEventType() == XmlResourceParser.START_TAG) {
                         // 获得标签名称
                         String name = xrp.getName();
                         // 判断标签名称是否等于 customer
                         if(name.equals("customer")){
                           counter++;
                           获得标签属性追加到 StringBuilder 中
                           sb.append("第"+counter+"条客户信息: "+"\n");
                           sb.append(xrp.getAttributeValue(0)+"\n");
                           sb.append(xrp.getAttributeValue(1)+"\n");
                           sb.append(xrp.getAttributeValue(2)+"\n");
                           sb.append(xrp.getAttributeValue(3)+"\n\n");
                         }
                    } else if (xrp.getEventType() == XmlPullParser.END_TAG) {
                    } else if (xrp.getEventType() == XmlPullParser.TEXT) {
                    }
                     // 下一个标签
```

```
                    xrp.next();
                }
            // 将 StringBuilder 设置为 TextView 的文本
            myTextView.setText(sb.toString());
        } catch (XmlPullParserException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
}
}
```

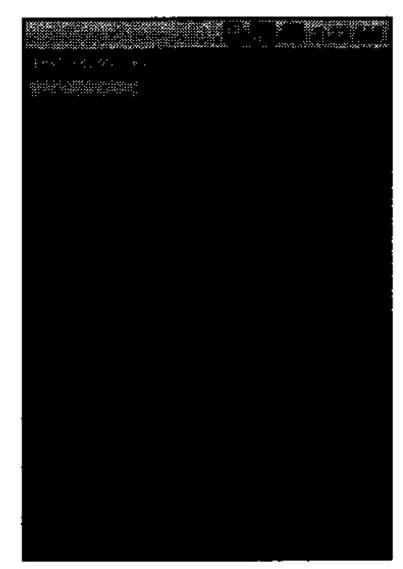


图 3.5　XML 资源应用

④ 运行程序，结果如图 3.5 所示。

3.6 使用 drawables 资源

drawable 资源是一些图片或者颜色资源，主要用来绘制屏幕，通过 Resources.getDrawable()方法获得。drawable 资源分为三类：Bitmap File（位图文件）、Color Drawable（颜色）、Nine-Patch Image（九片图片）。这里只讲述常用的位图文件的使用。

Android 中支持的位图文件有 png、jpg 和 gif。表 3.7 描述了位图文件的定义细节。

表 3.7　尺寸资源的定义和使用

资 源 位 置	res/drawable/file_name.png /file_name.jpg / file_name.gif.
获得位图资源的方法	Resources.getDrawable()
引用位图资源的格式	Java 代码中：R.drawable.file_name XML 文件中：@[package:]drawable/file_name

下面通过实例的方式来演示位图文件的使用。本实例首先在 res\drawable\目录下添加两个位图文件 g1.jpg 和 moto.jpg，并将这两个位图文件显示在 Activity 的 ImageView 中，第一个通过在布局文件中直接引用，第二个在 Java 代码中引用。实例步骤说明如下。

① 在“Chapter03_Resource”工程的 res\drawable\目录下添加两张位图文件 g1.jpg 和 moto.jpg。

② 创建一个布局文件 test_bitmap。在该布局文件中添加两个 ImageView 组件用来显示图标，其中第一个 ImageView 组件直接引用 g1.jpg 文件，第二个在 Java 代码中进行设置。

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

   <TextView
      android:text="测试位图资源"
```

```
        android:id="@+id/bitmapTextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

    <ImageView
        android:id="@+id/bitmapImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/g1"
        ></ImageView>

    <ImageView
        android:id="@+id/bitmapImageView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></ImageView>

</LinearLayout>
```

③ 在工程的 com.amaker.ch03.drawable 包中创建 TestBitmapActivity 类。在该类顶部声明一个 ImageView 视图组件，在 onCreate()方法中实例化该组件，并通过 Resources.getDrawable()方法获得位图资源，将 ImageView 组件设置为可显示的图片。

```
package com.amaker.ch03.drawable;

import android.app.Activity;
import android.content.res.Resources;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.widget.ImageView;

import com.amaker.test.R;
/**
 *
 * @author 郭宏志
 * 测试Bitmap资源
 */
public class TestBitmapActivity extends Activity {
    // 声明ImageView对象
    private ImageView myImageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前内容布局视图
        setContentView(R.layout.test_bitmap);
        // 通过findViewById方法获得ImageView实例
        myImageView = (ImageView)findViewById(R.id.bitm- apImageView02);
        // 获得Resources实例
        Resources r = getResources();
        // 通过Resources 获得Drawable实例
        Drawable d = r.getDrawable(R.drawable.moto);
        // 设置ImageView的ImageDrawable属性显示图片
        myImageView.setImageDrawable(d);
    }
}
```

④ 运行程序，结果如图 3.6 所示。

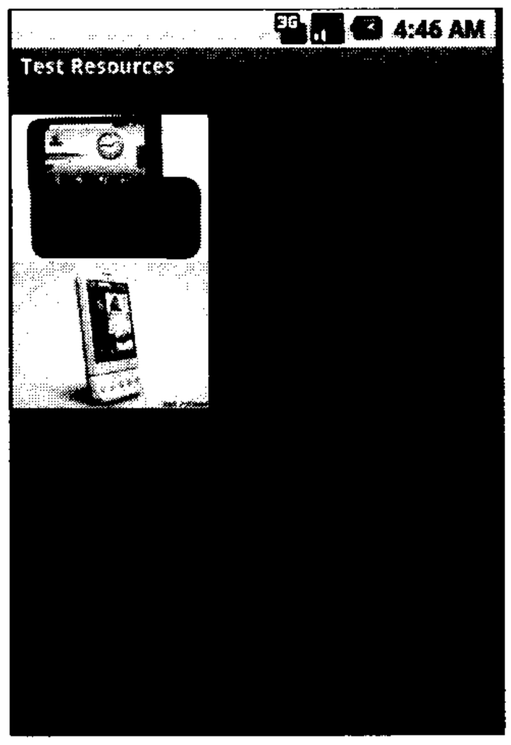


图 3.6　位图资源应用

3.7 使用布局（layout）资源

布局资源是 Android 中最常使用的一种资源，Android 可以将屏幕中组件的布局方式定义在一个 XML 文件中，这有点像 Web 开发中的 HTML 页面。我们可以调用 Activity.setContentView()方法，将布局文件展示在 Activity 上。Android 通过 LayoutInflater 类将 XML 文件中的组件解析为可视化的视图组件。布局文件保存在 res\layout\文件夹中，文件名称任意。

3.7.1　布局文件的定义

我们将通过表 3.8 来说明布局文件的定义情况。

表 3.8　布局文件的定义

资 源 位 置	res/layout/my_layout.xml(文件名称任意)
布局 XML 文件格式	使用<?xml version="1.0" encoding="utf-8"?> <布局类 xmlns:android="http://schemas.android.com/apk/res/android" id="@+id/string_name" (属性)> <视图组件或者其他嵌套布局类> <requestFocus/> </布局类>
获得 XML 资源的方法	Activity.setContentView()
引用 XML 资源的格式	Java 代码中：R.layout.my_layout XML 文件中：@[package:]layout/my_layout

3.7.2 布局文件的使用

下面通过一个实例来演示布局文件的用法。该实例定义一个布局文件，在该布局文件中添加一个 TextView、一个 EditText 和一个 Button，分别设置其属性，并且使用 Activity.setContentView()方法将其设置为 Activity 的界面，使用 findViewById()方法来得到布局中的组件。

① 在工程的 res\layout\目录下创建一个 test_layout.xml 布局文件，在该布局文件中使用 LinearLayout 嵌套 TableLayout 进行布局管理，其中添加 TextView、EditText 和 Button 三个视图组件，并为其设置属性。

```
<?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">
   <!-- 以上四个属性分别是命名空间、 组件布局方向(这里是垂直)、布局的宽(充满屏幕)和高(充满屏
幕)-->

   <!--  以下嵌套一个 TableLayout -->
   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">
      <TableRow>
         <TextView
         android:text="测试 Layout: "
         android:id="@+id/layoutTextView01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="@color/red_bg"/>

   <!-- 以上五个属性分别是：文本内容、引用组件的 ID、该组件的宽(内容的宽)、该组件的高(内容的高)、
文件颜色 -->

         <EditText
         android:text=""
         android:id="@+id/EditText01"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>

      <TableRow android:gravity="right">
         <Button
         android:text="Test"
         android:id="@+id/layoutButton01"
         android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            />
        </TableRow>
    </TableLayout>
</LinearLayout>
```

② 在工程的 com.amaker.ch03.layout 包中创建一个 TestLayoutActivity 类，在该类的顶部声明 TextView、EditText 和 Button。在 onCreate()方法中定义 setContentView()方法，将布局文件设置为 Activity 的界面，使用 findViewById()方法实例化以上三个视图组件。

```
   <EditText
package com.amaker.ch03.layout;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import com.amaker.test.R;
/**
 *
 * @author 郭宏志
 * 测试布局资源
 */
public class TestLayoutActivity extends Activity {
   private TextView myTextView;
   private EditText myEditText;
   private Button myButton;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置 Activity 的界面布局
      setContentView(R.layout.test_layout);
      // 通过 findViewByIdff 获得 TextView 实例
     myTextView = (TextView)findViewById(R.id.layoutText View01);
      // 通过 findViewById 方法获得 EditText 实例
     myEditText = (EditText)findViewById(R.id.layoutEdit Text01);
      // 通过 findViewById 方法获得 Button 实例
      myButton = (Button)findViewById(R.id.layoutButton01);
   }
}
```

③ 运行程序，结果如图 3.7 所示。

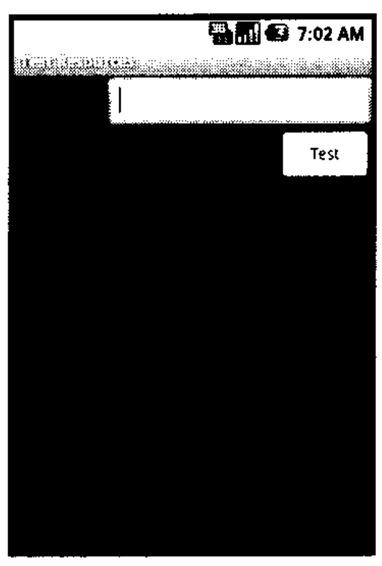


图 3.7　布局资源应用

3.8 使用菜单（menu）资源

任何视图组件的创建方式都有两种：一种通过在布局文件中声明创建；另一种通过在

代码中创建。菜单也不例外，我们既可以在资源文件中声明，也可以在代码中创建。

Android 中的菜单分为选项菜单、上下文菜单和子菜单，都可以在 XML 文件中声明定义，在代码中通过 MenuInflater 类使用。本节我们主要讲述如何使用菜单资源来声明创建菜单。

3.8.1 菜单资源文件的定义

菜单资源文件也是一个 XML 文件，该菜单文件位于工程的 res\menu\my_menu.xml 目录下。通过 R.menu.my_menu 的方式来引用。

典型菜单资源文件的结构是这样的：<menu>根元素，在<menu>根元素里面会嵌套<item>和<group>子元素，<item>元素中也可嵌套<menu>形成子菜单。

<menu>根元素没有属性，它包含<item>和<group>子元素。

<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：唯一标示该菜单组的引用 id。
- menuCategory：对菜单进行分类，定义菜单的优先级，有效值为 container、system、secondary 和 alternative。
- orderInCategory：一个分类排序整数。
- checkableBehavior：选择行为，单选、多选还是其他。有效值为 none、all 和 single。
- visible：是否可见，true 或者 false。
- enabled：是否可用，true 或者 false。

<item>表示菜单项，包含在<menu>或<group>中的有效属性。<item>元素的属性说明如下。

- id：唯一标示菜单的 ID 引用。
- menuCategory：菜单分类。
- orderInCategory：分类排序。
- title：菜单标题字符串。
- titleCondensed：浓缩标题，适合标题太长的时候使用。
- icon：菜单的图标。
- alphabeticShortcut：字符快捷键。
- numericShortcut：数字快捷键。
- checkable：是否可选。
- checked：是否已经被选。
- visible：是否可见。
- enabled：是否可用。

3.8.2　菜单资源文件的使用

下面还是通过一个实例来演示菜单资源的使用。本实例定义了一个文件系统的菜单信息，主菜单包括 File、Edit 和 Help 三个菜单项。File 菜单有 New、Open 和 Save 子菜单项；Edit 菜单有 Cut、Copy 和 Paste 子菜单项；Help 菜单有 About 和 Exit 子菜单项。其中主菜单分别添加了图标。File 子菜单带有快捷键，Edit 子菜单采用单选按钮。Help 子菜单可用响应单击事件，分别获得帮助和退出程序。

① 在工程的 res\menu\目录下创建 file_menu.xml 菜单资源文件。

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:title="File"
      android:icon="@drawable/file">
      <menu>
         <group
            android:id="@+id/noncheckable_group"
            android:checkableBehavior="none">

            <item
            android:id="@+id/newFile"
            android:title="New"
            android:alphabeticShortcut="n"/>

            <item
            android:id="@+id/openFile"
             android:title="Open"
             android:alphabeticShortcut="o"/>

            <item
            android:id="@+id/saveFile"
            android:title="Save"
            android:alphabeticShortcut="s"/>
         </group>
      </menu>
   </item>

   <item android:title="Edit" android:icon="@drawable/edit">
      <menu>
         <group android:id="@+id/edit_group"
               android:checkableBehavior="single">

            <item android:id="@+id/cut"
                 android:title="Cut" />

            <item android:id="@+id/copy"
                 android:title="Copy"/>
```

```
            <item android:id="@+id/past"
                 android:title="Past"/>
        </group>
    </menu>
</item>

<item android:title="Help" android:icon="@drawable/help">
    <menu>
        <group android:id="@+id/help_group">
            <item android:id="@+id/about"
                   android:title="About" />
            <item android:id="@+id/exit"
                   android:title="Exit" />
        </group>
    </menu>
</item>
</menu>
```

② 在工程 res\layout\目录下创建一个 test_menu.xml 布局文件，在该文件中添加一个 TextView 视图组件用来显示一个提示信息。

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

   <TextView
      android:text="测试菜单资源"
      android:id="@+id/menuTextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></TextView>

</LinearLayout>
```

③ 在工程的 com.amaker.ch03.menu 包中创建一个名为 TestMenuActivity 的 Activity 类。在该类的顶部声明 MenuInflater 类，在 onCreate()方法中实例化该对象，用来加载菜单 XML 资源。

```
package com.amaker.ch03.menu;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

import com.amaker.test.R;
/**
 * @author 郭宏志
```

```
 * 测试菜单资源
 */
public class TestMenuActivity extends Activity {
   private MenuInflater mi;
   @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
     // 设置 Activity 界面的布局
     setContentView(R.layout.test_menu);
     // 实例化 MenuInflater 对象
     mi = new MenuInflater(this);
   }
}
```

④ 覆盖 Activity 的 onCreateOptionsMenu()方法，在其中创建菜单。

```
/*
 * 创建菜单
 */
public boolean onCreateOptionsMenu(Menu menu) {
    // 调用 MenuInflater 的 inflate 方法，通过配置文件创建菜单
    mi.inflate(R.menu.file_menu, menu);
    return true;
}
```

⑤ 覆盖 Activity 的 onOptionsItemSelected()方法，响应菜单单击事件，当单击 About 时显示帮助，当单击 Exit 时退出程序。

```
@Override // 菜单项的单击事件
 public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    // 显示关于对话框
    case R.id.about:
       aboutAlert("本实例演示的是如何使用 XML 菜单资源来定义菜单！");
       break;
    // 显示退出对话框
    case R.id.exit:
       exitAlert("真的要退出吗？");
       break;
    }
    return true;
 }

 // 显示对话框
 private void exitAlert(String msg){
     // 实例化 AlertDialog. Builder
     AlertDialog.Builder builder = new AlertDialog.Builder(this);
     // 设置显示信息
     builder.setMessage(msg)
          .setCancelable(false)
           // 确定按钮
          .setPositiveButton("确定", new DialogInterface.OnClickListener() {
```

```
                public void onClick(DialogInterface dialog, int id) {
                  // 结束Activity
                finish();
                }
            // 取消按钮
            }).setNegativeButton("取消", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                 return;
                }
            });
        // 创建对话框
        AlertDialog alert = builder.create();
         // 显示对话框
        alert.show();
    }

    // 显示对话框
    private void aboutAlert(String msg){
         // 实例化AlertDialog. Builder
         AlertDialog.Builder builder = new AlertDialog.Builder(this);
         // 设置显示信息
         builder.setMessage(msg)
             .setCancelable(false)
              // 设置确定按钮
             .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog, int id) {
                 }
             });
         // 创建对话框
         AlertDialog alert = builder.create();
         // 显示对话框
        alert.show();
    }
```

⑥ 运行程序，结果如图 3.8 所示。

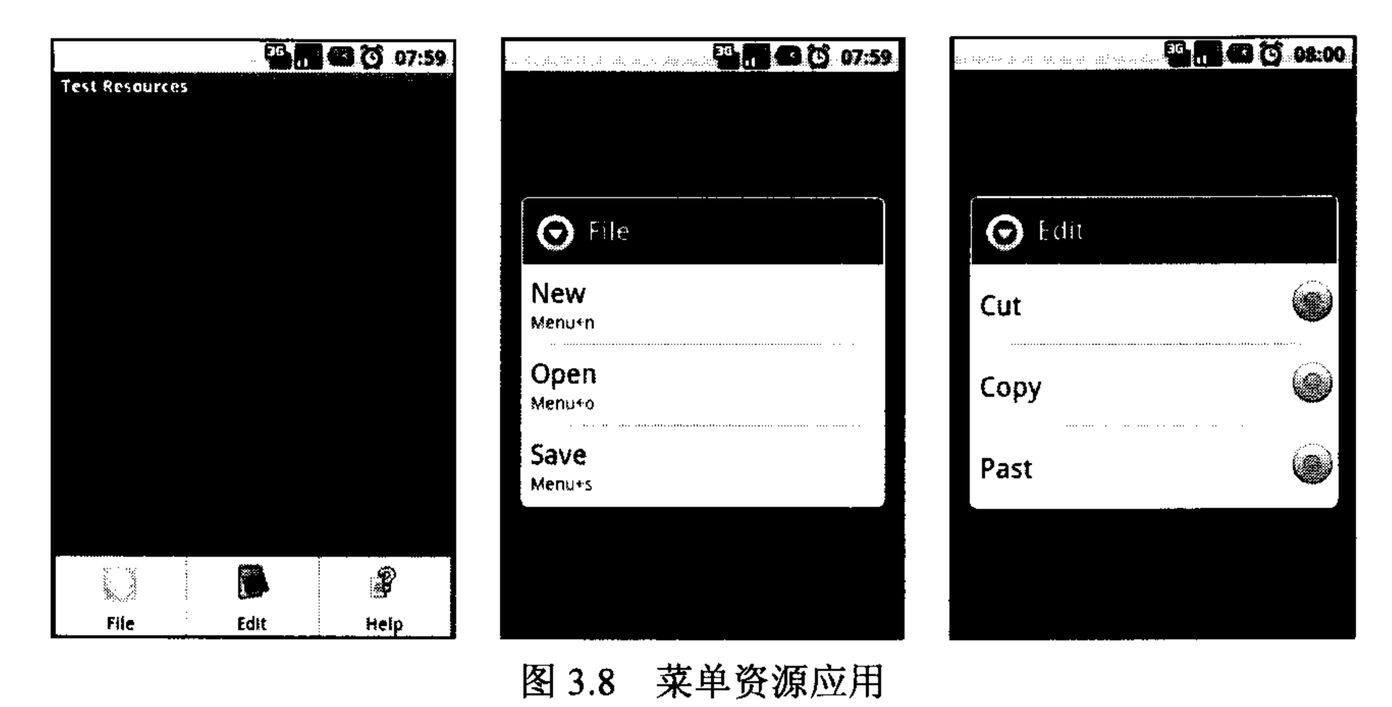
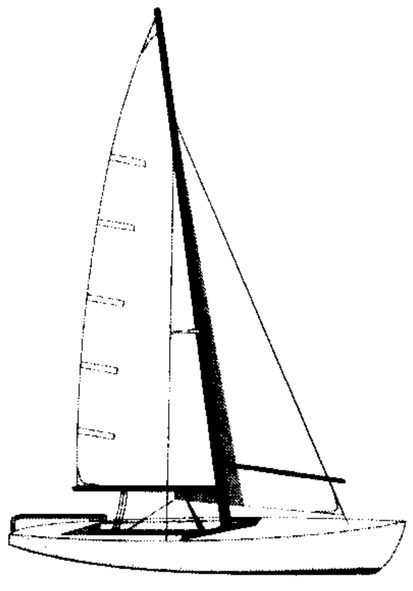


图 3.8　菜单资源应用



第 4 章　Android 用户界面

Android 系统为我们提供了丰富的可视化用户界面组件，包括菜单、对话框、按钮、下拉列表等。Android 系统借用了 Java 里的 UI 设计思想，包括事件响应机制和布局管理，所以有过 Java GUI 开发经验的读者，学习这一章会很轻松。

Android 系统中的所有 UI 类都是建立在 View 和 ViewGroup 这两个类的基础之上的。所有 View 的子类称为“Widget”，所有 ViewGroup 的子类称为“Layout”。

View 和 ViewGroup 之间采用了组合设计模式（Composite），可以使得“部分-整体”同等对待。ViewGroup 作为布局容器类在最上层，布局容器里面又可以有 View 和 ViewGroup，其层次结构如图 4.1 所示。

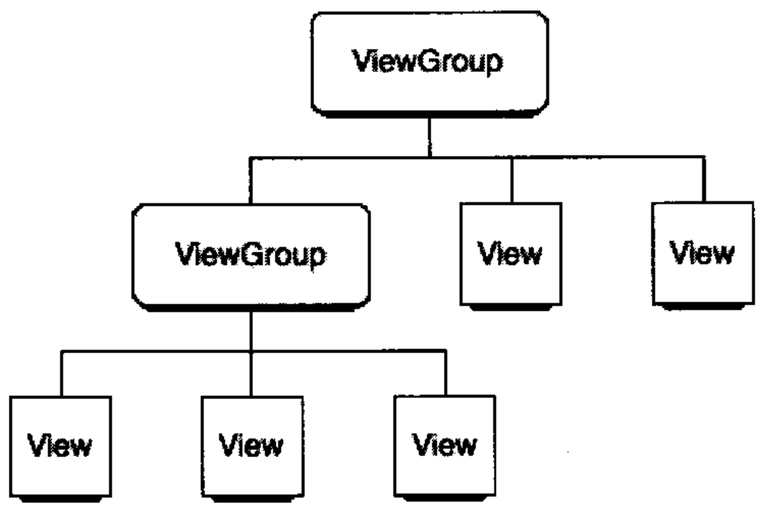


图 4.1　View 和 ViewGroup 的层次关系

4.1 菜单

菜单的设计在人机交互中可以说是非常人性化的，它提供了不同功能分组展示的能力。Android 中菜单分为三种类型：选项菜单（Option Menu）、上下文菜单（Context Menu）和子菜单（Sub Menu）。

4.1.1 选项菜单（Option Menu）

当用户单击设备上的菜单（Menu）按键时，弹出的菜单就是选项菜单。选项菜单的菜单项最多只能有六个，超过六个第六个自动显示“更多”选项来展开显示。我们先来看一个 Android 系统自带的选择菜单功能，当我们单击“Menu”按键时，将弹出如图 4.2 所示的选项菜单。

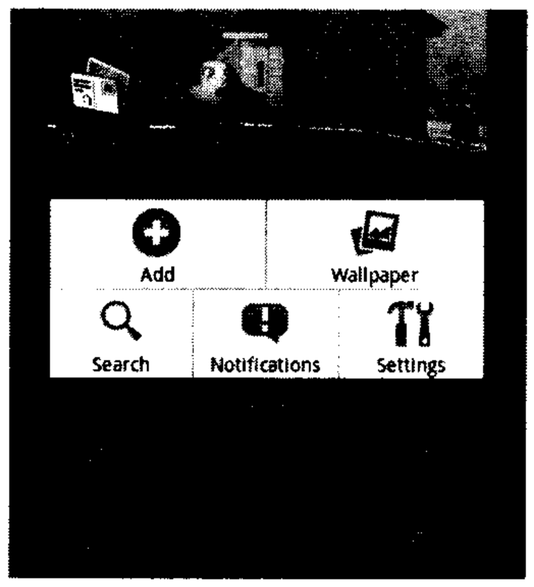


图 4.2　Android 选项菜单

我们也经常会在程序里自己创建选项菜单。例如，我们设计了一款游戏，就需要设计一个选项菜单给用户提供交互接口。创建一个选项菜单需要如下步骤。

① 覆盖 Activity 的 onCreateOptionsMenu(Menu menu)方法，当我们第一次打开菜单时该方法被自动调用。

② 调用 Menu 的 add()方法添加菜单项（MenuItem），可以调用 MenuItem 的 setIcon()方法来为菜单项设置图标。

③ 当菜单项（Menuitem）被选择时，覆盖 Activity 的 onOptionsItemSelected()方法来响应事件。

下面是一个创建并响应菜单选项的实例代码：

```
package com.amaker.test;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
/**
 *
 * @author 郭宏志
 * 选项菜单实例
 */
public class MainActivity extends Activity {
   // 菜单项 ID 常量
   private static final int ITEM1 = Menu.FIRST;
```

```
    // 菜单项 ID 常量
    private static final int ITEM2 = Menu.FIRST+1;
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 的界面布局
        setContentView(R.layout.main);
    }

    /**
     * 覆盖该方法添加菜单项
     */
    public boolean onCreateOptionsMenu(Menu menu) {
        // 添加菜单项
        menu.add(0, ITEM1, 0, "开始");
        // 添加菜单项
        menu.add(0, ITEM2, 0, "推出");
        return true;
    }

    /**
     * 覆盖该方法，响应菜单选项被单击事件
     */
    public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
          // 菜单项 1 被选择
          case ITEM1:
              // 设置 Activity 标题
              setTitle("开始游戏！");
              break;
          // 菜单项 2 被选择
          case ITEM2:
              // 设置 Activity 标题
              setTitle("退出！");
              break;
          }
      return true;
    }
}
```

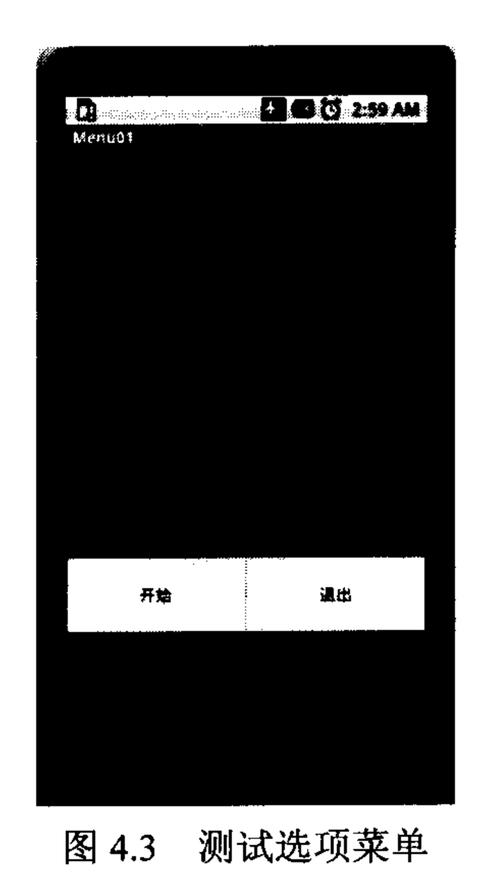


图 4.3　测试选项菜单

单击“Menu”按钮，程序运行结果如图 4.3 所示。

4.1.2　上下文菜单（Context Menu）

当用户长时间按键不放时，弹出的菜单称为上下文菜单。我们经常在 Windows 中用鼠标右键单击弹出的菜单就是上下文菜单。我们还是先来看看 Android 系统中的上下文菜单应用吧！当我们在设备的启动界面长时间按键不放时，会弹出如图 4.4 所示的上下文菜单。

在程序中创建一个上下文菜单需要如下步骤。

① 覆盖 Activity 的 onCreateContextMenu()方法，调用 Menu 的 add 方法添加菜单项

（MenuItem）。

② 覆盖 onContextItemSelected()方法，响应菜单单击事件。

③ 调用 registerForContextMenu()方法，为视图注册上下文菜单。

下面实例演示了在 TextView 视图上注册上下文菜单来改变 TextView 的背景色：

```
package com.amaker.test;
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.TextView;

/**
 *
 * @author 郭宏志 测试上下文菜单，改变背景色
 */
public class MainActivity extends Activity {
    // 菜单项 ID
    private static final int ITME1 = Menu.FIRST;
    // 菜单项 ID
    private static final int ITME2 = Menu.FIRST+1;
    // 菜单项 ID
    private static final int ITME3 = Menu.FIRST+2;
    // 声明 TextView 文本视图对象
   private TextView myTV;
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // 设置当前 Activity 界面布局
       setContentView(R.layout.main);
       // 通过 findViewById 方法获得 TextView 实例
       myTV = (TextView)findViewById(R.id.TextView01);
       // 注册上下文菜单
       registerForContextMenu(myTV);
   }

   @Override
   public void onCreateContextMenu(ContextMenu menu, View v,
          ContextMenuInfo menuInfo) {
       // 添加菜单项
       menu.add(0, ITME1, 0, "红色背景");
       // 添加菜单项
       menu.add(0, ITME2, 0, "绿色背景");
       // 添加菜单项
       menu.add(0, ITME3, 0, "白色背景");
   }
```

图 4.4　系统上下文菜单

```
    @Override
    public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        // 菜单项 1 被选择
        case ITME1:
            // 设置 TextView 背景色
            myTV.setBackgroundColor(Color.RED);
            break;
        // 菜单项 2 被选择
        case ITME2:
            // 设置 TextView 背景色
            myTV.setBackgroundColor(Color.GREEN);
            break;
        // 菜单项 3 被选择
        case ITME3:
            // 设置 TextView 背景色
            myTV.setBackgroundColor(Color.WHITE);
            break;
        }
        return true;
    }
}
```

程序运行结果如图 4.5 所示。

4.1.3　子菜单（Sub Menu）

很好理解，子菜单就是将相同功能的分组进行多级显示的一种菜单，比如，Windows 的“文件”菜单中又有“新建”、“打开”、“关闭”等子菜单。Android 系统中的子菜单也很多见，如“Settings”（设置）菜单就有其子菜单，如图 4.6 所示。

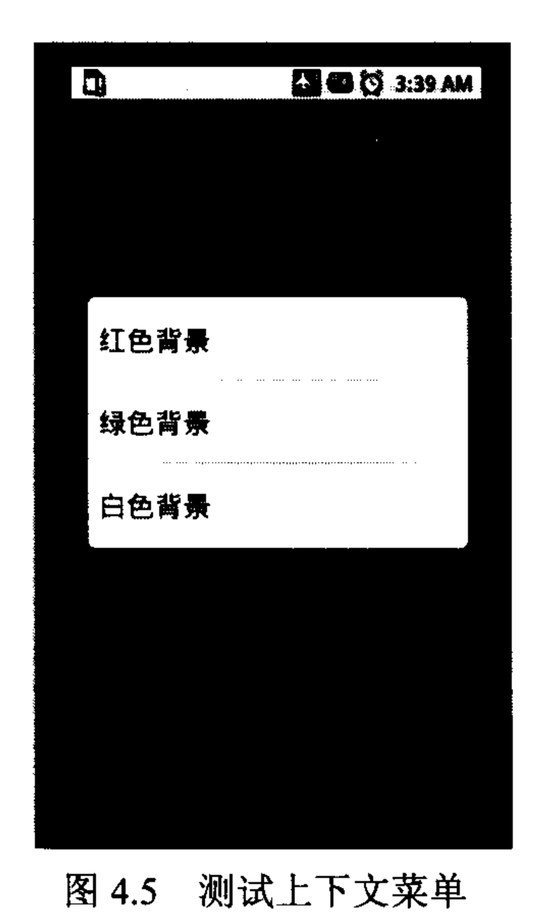


图 4.5　测试上下文菜单

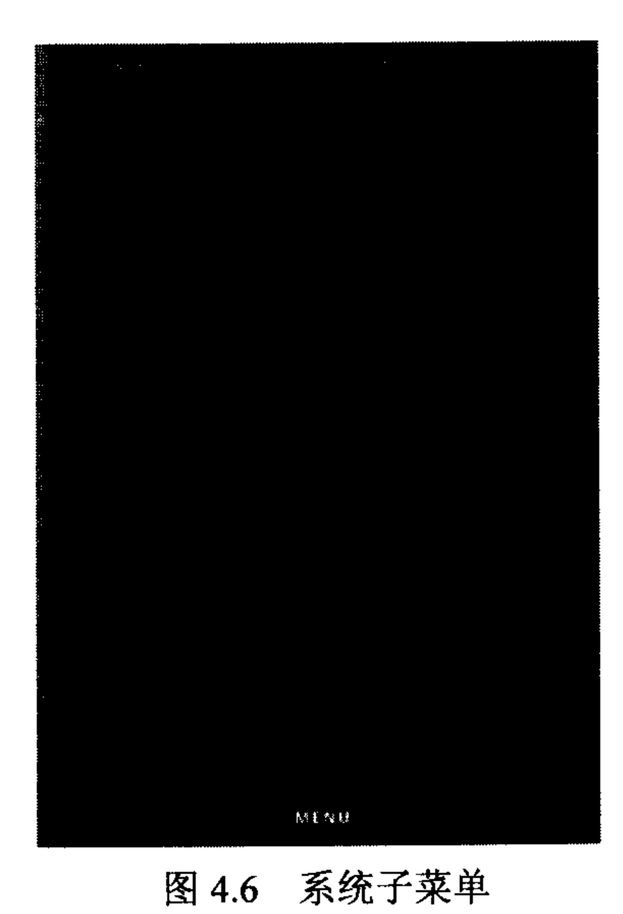


图 4.6　系统子菜单

在程序中创建子菜单需要如下步骤。

① 覆盖 Activity 的 onCreateOptionsMenu ()方法，调用 Menu 的 addSubMenu()方法添加子菜单项（Sub Menu）。

② 调用 SubMenu 的 add()方法，添加子菜单（Sub Menu）。

③ 覆盖 onContextItemSelected()方法，响应菜单单击事件。

如下代码演示了创建文件菜单及其子菜单的过程。

```
package com.amaker.test;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
/**
 *
 * @author 郭宏志
 * 子菜单实例
 */
public class MainActivity extends Activity {
    // 菜单项 ID 常量
    private static final int ITEM1 = Menu.FIRST;
    // 菜单项 ID 常量
    private static final int ITEM2 = Menu.FIRST+1;

   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置 Activity 界面布局
      setContentView(R.layout.main);
   }

   /**
    * 覆盖该方法添加子菜单项
    */
   public boolean onCreateOptionsMenu(Menu menu) {
       // 添加子菜单
       SubMenu file = menu.addSubMenu("文件");
       // 添加子菜单
       SubMenu edit = menu.addSubMenu("编辑");
       // 为子菜单添加菜单项
       file.add(0, ITEM1, 0, "新建");
       // 为子菜单添加菜单项
       file.add(0, ITEM2, 0, "打开");
       return true;
   }

   /**
    * 覆盖该方法，响应菜单选项被单击事件
    */
   public boolean onOptionsItemSelected(MenuItem item) {
```

```
        switch (item.getItemId()) {
            // 菜单项 1 被选择
            case ITEM1:
                // 设置 Activity 标题
                setTitle("新建文件！");
                break;
            // 菜单项 2 被选择
            case ITEM2:
                // 设置 Activity 标题
                setTitle("打开文件！");
                break;
            }
        return true;
        }
    }
```

程序运行结果如图 4.7 所示。

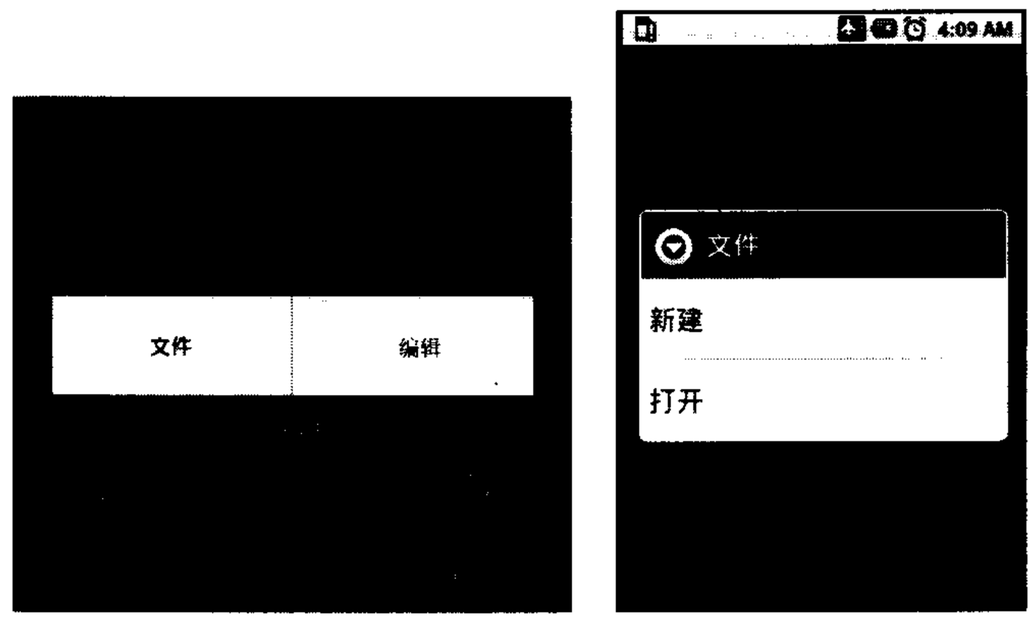


图 4.7　测试子菜单

4.2 对话框

对话框是程序运行中的弹出窗口。例如，当用户要删除一个联系方式时，会弹出一个对话框，让用户确认是否真的要删除。Android 系统提供了四种对话框：警告对话框（AlertDialog）、进度对话框（ProgressDialog）、日期选择对话框（DatePickerDialog）和时间选择对话框（TimePickerDialog），本节重点讲述 AlertDialog，其他对话框的应用将在后续章节中进行讲述。

AlertDialog 是一个提示窗口，要求用户作出选择。该对话框中一般会有几个选择按钮、标题信息和提示信息。

在程序中创建对话框需要如下步骤。

① 获得 AlertDialog 的静态内部类 Builder 对象，由该类来创建对话框。

② 通过 Buidler 对象设置对话框的标题、按钮以及按钮将要响应的事件。

③ 调用 Builder 的 create()方法创建对话框。

④ 调用 AlertDialog 的 show()方法显示对话框。

如下代码创建了具有“是”、“否”按钮和提示信息的对话框，用来确认用户的操作。

```
package com.amaker.test;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
/**
 *
 * @author amaker
 * AlertDialog 测试
 */
public class MainActivity extends Activity {
   // 声明 TextView 对象
   private TextView myTV;
   // 声明 Button 对象
   private Button myBtn;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // 设置 Activity 界面布局
       setContentView(R.layout.main);
       // 通过 findViewById 方法获得 TextView 实例
       myTV = (TextView)findViewById(R.id.TextView01);
       // 通过 findViewById 方法获得 Button 实例
       myBtn = (Button)findViewById(R.id.Button01);
       // 实例化 AlertDialog.Builder 对象
       final AlertDialog.Builder builder = new AlertDialog.Builder(this);
       // 添加按钮的单击监听器
       myBtn.setOnClickListener(new OnClickListener() {
          public void onClick(View v) {
                // 设置显示信息
                builder.setMessage("真的要删除该记录吗？").
                // 设置确定按钮
                setPositiveButton("是", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                        // 设置 TextView 文本
                        myTV.setText("删除成功！");
                }
          }).
              // 取消按钮
              setNegativeButton("否", new DialogInterface.OnClickListener() {
              // 单击事件
                  public void onClick(DialogInterface dialog, int which) {
```

```
                        // 设置 TextView 文本内容
                        myTV.setText("取消删除！");
                    }
            });
              // 创建对话框
              AlertDialog ad = builder.create();
               // 显示对话框
               ad.show();
        }
          }
        );
    }
}
```

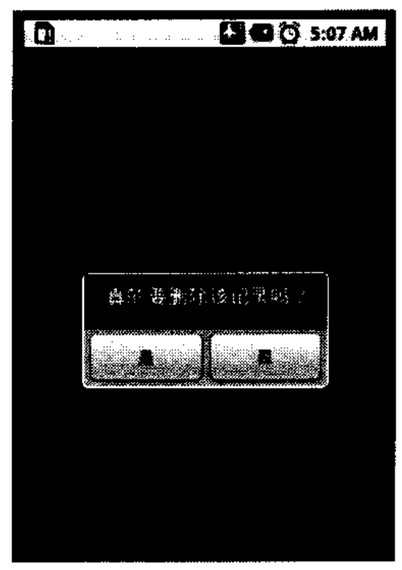


图 4.8　测试对话框

程序运行结果如图 4.8 所示。

除了上述对话框之外，还可以像上下文菜单一样来设置多个条目选项，供用户选择。

```
package com.amaker.test;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    // 声明文本视图 TextView
    private TextView myTV;
    // 声明按钮 Button
    private Button myBtn;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 TextView 实例
        myTV = (TextView)findViewById(R.id.TextView01);
        // 通过 findViewById 方法获得 Button 实例
        myBtn = (Button)findViewById(R.id.Button01);
        // 对话框选项数组
        final String[] items = {"奥尔良鸡腿堡","麻辣鸡腿堡","咖啡"};
        // 实例化 1 AlertDialog.Builder 对象
        final AlertDialog.Builder builder = new AlertDialog.Builder(this);
        // 添加按钮单击监听器
        myBtn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 设置标题和选项
                //builder.setTitle(" 请 点 餐 ").setItems(items,  new  DialogInterface.
```

```
OnClickListener() {
                // 如果改为如下方法，以单项按钮样式显示
                builder.setTitle("请点餐").setSingleChoiceItems(items,-1,
                    new DialogInterface.OnClickListener() {
                    // 单击事件
                    public void onClick(DialogInterface dialog, int which) {
                        // 设置 TextView 文本内容为选中的选项
                        myTV.setText(items[which]);
                    }
                });
                 // 创建对话框
                AlertDialog ad = builder.create();
                 // 显示对话框
                ad.show();
            }
        });
    }
}}
```

运行程序，结果如图 4.9 所示。

图 4.9　测试对话框

4.3 提示信息（Toast）

提示信息（Toast）在程序中很常用，使用也很简单。当用户执行某个动作之后，自动显示一个提示信息，过一会儿将自动消失，这就是 Toast。

在程序中创建 Toast 的步骤说明如下。

① 调用 Toast 的静态方法 makeText()添加显示文本和时长。

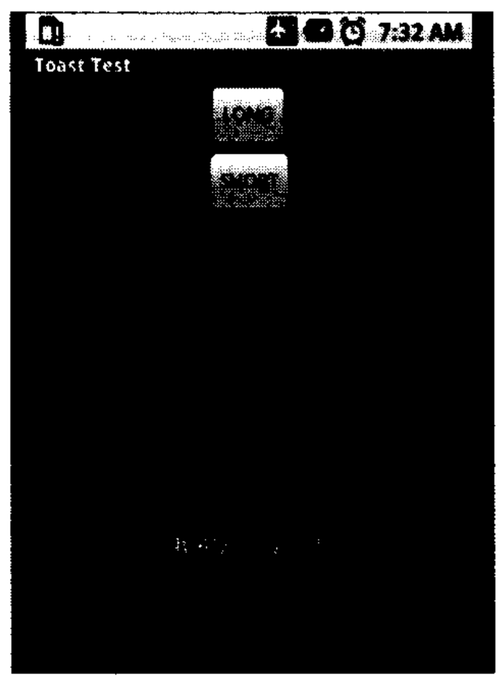
② 调用 Toast 的 show()显示。

下面实例在屏幕上添加两个按钮：第一个按钮响应事件显示一个长时间 Toast；第二个按钮响应事件显示短时间 Toast。

```
package com.amaker.test;
import android.app.Activity;
```

```
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends Activity {
    // 声明 Button 对象
    private Button b1,b2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button)findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 Button 对象
        b2 = (Button)findViewById(R.id.Button02);
        // 定义 Toast 显示时间属性
        final int l = Toast.LENGTH_LONG;
        // 定义 Toast 显示时间属性
        final int s = Toast.LENGTH_SHORT;
        // 定义 Toast 显示内容属性
        final String s1 = "我多显示一会儿！";
        // 定义 Toast 显示内容属性
        final String s2 = "我少显示一会儿！";
        // 为 Button 添加单击监听器
        b1.setOnClickListener(new OnClickListener() {
            // 单击事件
            public void onClick(View v) {
                // 创建 Toast
                Toast t1 = Toast.makeText(getApplicationContext(), s1, l);
                // 显示 Toast
                t1.show();
            }
        });
        // 为 Button 添加单击事件监听器
        b2.setOnClickListener(new OnClickListener() {
            // 单击事件
            public void onClick(View v) {
                // 创建 Toast
                Toast t2 = Toast.makeText(getApplicationContext(), s2, s);
                // 显示 Toast
                t2.show();
            }
        });
    }
}
```

运行程序结果如图 4.10 所示。



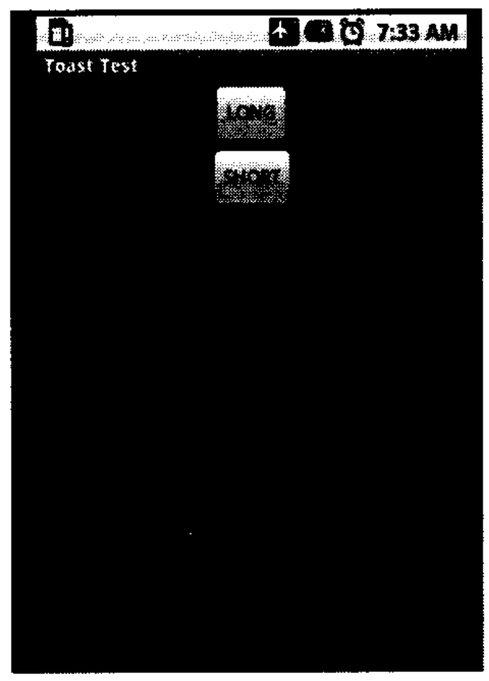


图 4.10　测试 Toast

4.4 事件处理

在 Android 系统中引用 Java 中的事件处理机制，包括事件、事件源和事件监听器三个方面。事件可以是鼠标事件、键盘事件、触摸事件或者鼠标移动事件等；事件源是指产生时间的组件；事件监听器是组件产生事件时响应的接口。

举例来说明三者的关系，现在高级轿车都有防盗功能，当车产生强烈震动时就会报警。在这里事件是震动，事件源是车，事件监听器是报警器。

4.4.1　事件处理机制

Java 中的事件角色有三个。

- event object：就是事件产生时具体的“事件”，用于 listener 的相应方法之中，作为参数，一般存在于 listerner 的方法之中。
- event source：具体接受事件的实体，比如说，你单击一个 button，那么 button 就是 event source。
- event listener：事件监听器，当有其对应的 event object 产生的时候，它就调用相应的方法进行处理。这种方式也被称为方法回调（callback）。

下面我们举一个生活中的实例来说明事件处理机制。在我们房间里经常会安装防火监听器，当房子着火时，监听器就会报警。下面的实例代码演示了该功能。

1．着火事件类，绑定事件源

```
package com.amaker.test;
import java.util.EventObject;
// 事件类
public class FireEvent extends EventObject {
    // 定义事件源对象
```

```
    private Object eventSource;
    // 构造方法
    public FireEvent(Object obj) {
        super(obj);
        this.eventSource = obj;
    }
    // 获得事件源
    public Object getEventSource() {
        return eventSource;
    }
}
```

2. 着火监听器，监听着火事件

```
package com.amaker.test;
import java.util.EventListener;
// 事件监听器接口
public interface Listener extends EventListener{
    // 事件方法
    public void listen(FireEvent e);
}
```

3. 着火监听器实现类

```
// 事件监听器实现类
public class FireListener implements Listener{
    @Override
    public void listen(FireEvent e) {
        System.out.println("房子着火啦！");
    }
}
```

4. 事件源房子类

```
package com.amaker.test;
import java.util.ArrayList;
import java.util.List;
// 事件源
public class House {
    // 定义标示
    private boolean flag = false;
    // 监听器列表
    private List listeners = new ArrayList();
    // 获得标示
    public boolean getFlag() {
        return flag;
    }
    // 设置标示
    public void setFlag(boolean flag) {
        this.flag = flag;
        if(flag){
            ntf();
        }
    }
```

```
    // 添加监听器
    public void addListener(Listener l) {
        listeners.add(l);
    }
    // 获得监听器
    public List getListeners() {
        return listeners;
    }
    // 通知方法
    public void ntf() {
        // 遍历监听器
        for (int i = 0; i < listeners.size(); i++) {
            // 获得监听器
            Listener l = (Listener) listeners.get(i);
            // 监听
            l.listen(new FireEvent(this));
        }
    }
}
```

5. 测试类

```
package com.amaker.test;
public class Test {
    public static void main(String[] args) {
        // 事件源房子
        House h = new House();
        // 添加监听器
        h.addListener(new FireListener());
        // 匿名内部类实现
        h.addListener(new Listener(){
            public void listen(FireEvent e) {
                System.out.println("冒烟啦！");
            }
        });
        h.setFlag(true);
    }
}
```

程序运行结果如图 4.11 所示。

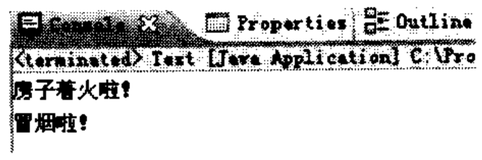


图 4.11　测试事件

4.4.2　Android 中的事件监听器

在 Android 系统中常见的事件监听器有如下几种：

- 单击事件（View.OnClickListener）：当用户触碰到某个组件或者方向键被按下时产生该事件，该事件的处理方法是 onClick()。
- 焦点事件（View.OnFocusChangeListener）：组件得到或者失去焦点时产生该事件，事件处理方法是 onFocusChange()。
- 按键事件（View.OnKeyListener）：用户按下或者释放设备上的某个按键时产生，事

件处理方法是 onKey()。

- 触碰事件（View.OnTouchListener）：设备具有触摸屏功能时，触碰屏幕产生该事件，事件处理方法是 onTouch()。
- 创建上下文菜单事件（View.OnCreateContextMenuListener）：创建上下文菜单时产生该事件，事件处理方法是 onCreateContextMenu()。

4.4.3 事件处理步骤

要实现事件处理，有如下几个步骤。

① 创建事件监听器。

② 给要响应事件的组件注册事件监听器。

③ 在事件处理方法中编写实现代码。

这里我们看一个用户登录的表单界面是如何来响应用户事件的。

```
package com.amaker.test;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnFocusChangeListener;
import android.view.View.OnKeyListener;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Toast;
import android.widget.CompoundButton.OnCheckedChangeListener;
/**
 *
 * @author 郭宏志
 * 测试事件
 */
public class MainActivity extends Activity {

    // 定义编辑文本框 EditText
    private EditText myEdit1, myEdit2;
    // 定义多选框 CheckBox
    private CheckBox cb1;
    // 定义按钮 Button
    private Button b1, b2;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
       setContentView(R.layout.main);
```

```
 // 通过 findViewById 方法获得 EditText 对象
myEdit1 = (EditText) findViewById(R.id.EditText01);
 // 通过 findViewById 方法获得 EditText 对象
myEdit2 = (EditText) findViewById(R.id.EditText02);
 // 通过 findViewById 方法获得 ChecBox 对象
cb1 = (CheckBox) findViewById(R.id.CheckBox01);
 // 通过 findViewById 方法获得 Button 对象
b1 = (Button) findViewById(R.id.Button01);
 // 通过 findViewById 方法获得 Button 对象
b2 = (Button) findViewById(R.id.Button02);

// 编辑文本框的按键事件
myEdit1.setOnKeyListener(new OnKeyListener() {
      // 按键方法
   public boolean onKey(View v, int keyCode, KeyEvent event) {
         // 清空 EditText
        myEdit1.setText("");
        return false;
   }
});

// 编辑文本框的按键事件
myEdit2.setOnKeyListener(new OnKeyListener() {
     // 按键方法
     public boolean onKey(View v, int keyCode, KeyEvent event) {
          // 清空 EditText
         myEdit2.setText("");
         return false;
   }
});

// 编辑文本框的焦点事件
myEdit1.setOnFocusChangeListener(new OnFocusChangeListener() {
   // 焦点改变方法
   public void onFocusChange(View v, boolean hasFocus) {
         // 使用 Toast 显示 EditText 内容
        Toast.makeText(getApplicationContext(), myEdit1.getText(),
                Toast.LENGTH_LONG);
   }
});
// 编辑文本框的焦点事件
myEdit2.setOnFocusChangeListener(new OnFocusChangeListener() {
   // 焦点改变方法
   public void onFocusChange(View v, boolean hasFocus) {
        // 使用 Toast 显示 EditText 的内容
         Toast.makeText(getApplicationContext(),   myEdit2.getText(),Toast.
LENGTH_LONG);
   }
});
// 多选框的选择事件
cb1.setOnCheckedChangeListener(new OnCheckedChangeListener() {
   // 事件方法
```

```
            public void onCheckedChanged(CompoundButton buttonView,
                    boolean isChecked) {
                // 使用 Toast 显示 CheckBox 是否被选中
                Toast.makeText(getApplicationContext(), cb1.isChecked()+""  ,
Toast. LENGTH_LONG);
            }
        });
        // 按钮的选择事件
        b1.setOnClickListener(new OnClickListener() {
            // 单击方法
            public void onClick(View v) {
                // 使用 Toast 显示 Button 内容
                Toast.makeText(getApplicationContext(), b1.getText(),  Toast.LENGTH_
LONG);
            }
        });
        // 按钮的选择事件
        b2.setOnClickListener(new OnClickListener() {
            // 单击方法
            public void onClick(View v) {
                // 使用 Toast 显示 Button 文本内容
                Toast.makeText(getApplicationContext(), b2.getText(),  Toast.LENGTH_
LONG);
            }
        });
    }
}
```

布局文件代码如下所示。

```
<?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">

<TableLayout
   android:id="@+id/TableLayout01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

   <TableRow>
      <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="用户名称: "></TextView>
      <EditText
      android:id="@+id/EditText01"
      android:text="admin"
      android:layout_width="fill_parent"
      android:scrollHorizontally="true"></EditText>
   </TableRow>
```

```
    <TableRow>
        <TextView
        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="用户密码："></TextView>
        <EditText
        android:id="@+id/EditText02"
        android:text="123"
        android:password="true"
        android:layout_width="fill_parent"
        android:scrollHorizontally="true" ></EditText>
    </TableRow>

    <TableRow>
        <TextView
        android:id="@+id/TextView03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="自动登录："></TextView>
        <CheckBox
        android:text=""
        android:id="@+id/CheckBox01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></CheckBox>
    </TableRow>

    <TableRow android:gravity="right">
        <Button android:text="登录" android:id="@+id/Button01">
    </Button>
        <Button android:text="取消" android:id="@+id/Button02">
    </Button>
    </TableRow>
    </TableLayout>
    </LinearLayout>
```

图 4.12　测试事件

程序运行结果如图 4.12 所示。

4.5 布局管理（Layout）

所谓布局就是组件在 Activity 中的呈现方式，即组件大小、间距和对齐方式等。Android 提供了以下两种创建布局的方式：

- 在 XML 配置文件中声明：这种方式是将需要呈现的组件在配置文件中进行声明，在程序中通过 setContentView(R.layout.main)方法将视图呈现在 Activity 中，通过 findViewById 方法获得组件实例（推荐）。
- 在程序中通过“硬代码”直接实例化布局及其组件。

在 Android 中常见的布局方式有如下几种：

- 线性布局（LinearLayout）：按照垂直或者水平方向布局组件。
- 帧布局（FrameLayout）：组件从屏幕的左上角坐标布局组件。
- 表格布局（TableLayout）：按照行列方式布局组件。
- 相对布局（RelativeLayout）：相对其他组件的布局方式。
- 绝对布局（AbsoluteLayout）：按照绝对坐标来布局组件。

4.5.1　线性布局

线性布局是将子组件按照垂直或者水平方向来布局，方向控制由“android:orientation”属性来控制，属性值有垂直（vertical）和水平（horizontal）两种。另外一个常用属性是“android:gravity”，来控制左右上下对齐方式，其属性值有上（top）、下（bottom）、左（left）、右（right）。

下面的实例是将几个 TextView 通过 LinearLayout 以不同的方向来布局。其中使用了 LinearLayout 外层垂直布局、内层上面水平布局和下面垂直布局。

```
<?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">


   <LinearLayout
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:layout_weight="1">

   <TextView
      android:text="red"
      android:gravity="center_horizontal"
      android:background="#aa0000"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent"
      android:layout_weight="1"/>

   <TextView
      android:text="green"
      android:gravity="center_horizontal"
      android:background="#00aa00"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent"
      android:layout_weight="1"/>

   <TextView
```

```
        android:text="blue"
        android:gravity="center_horizontal"
        android:background="#0000aa"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>

    <TextView
        android:text="yellow"
        android:gravity="center_horizontal"
        android:background="#aaaa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>

    </LinearLayout>

    <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">

    <TextView
        android:text="row one"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    <TextView
        android:text="row two"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    <TextView
        android:text="row three"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    <TextView
        android:text="row four"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>

    </LinearLayout>
```

程序运行结果如图 4.13 所示。

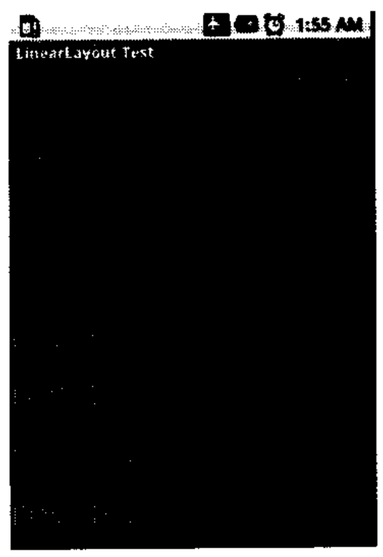


图 4.13　测试 LinearLayout

4.5.2　帧布局

帧布局是从屏幕的左上角（0,0）坐标开始布局，多个组件层叠排序，后面的组件覆盖前面的组件。下面的实例在 FrameLayout 布局中放置了三个 TextView，分别设置 TextView 的大小、背景色及层叠显示。

```
<?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">

   <FrameLayout android:id="@+id/FrameLayout01"
      android:layout_width="wrap_content" android:layout_height="wrap_content">

      <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:width="150px"
      android:height="150px"
      android:background="#aa0000"
      >
      </TextView>

      <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:width="100px"
      android:height="100px"
      android:background="#00aa00"
      >
      </TextView>

      <TextView
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:width="80px"
      android:height="80px"
      android:background="#0000aa"
      >
      </TextView>
   </FrameLayout>

</LinearLayout>
```

程序运行结果如图 4.14 所示。

图 4.14　测试 FrameLayout

4.5.3 表格布局

表格布局以行、列表格的方式布局子组件。TableLayout 中使用 TableRow 对象来定义多行。

```
<?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"
    >

<TableLayout
android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:collapseColumns="2"
android:stretchColumns="1"
>

    <TableRow
    android:id="@+id/TableRow01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="用户名称"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

        <EditText
        android:text="admin"
        android:id="@+id/EditText01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></EditText>
    </TableRow>

    <TableRow
    android:id="@+id/TableRow01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="用户密码"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

        <EditText
        android:text="123"
        android:id="@+id/EditText01"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:password="true"

        ></EditText>
    </TableRow>

    <TableRow
    android:id="@+id/TableRow01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <Button
        android:text="注册"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

        <Button
        android:text="登录"
        android:id="@+id/Button02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
    </TableRow>

</TableLayout>
</LinearLayout>
```

程序运行结果如图 4.15 所示。

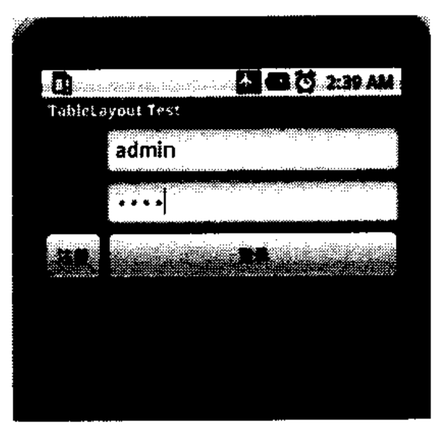


图 4.15　测试 TableLayout

4.5.4　相对布局

相对布局是指按照组件之间的相对位置来布局，如在某个组件的左边、右边、上面和下面等。例如，android:layout_below 属性指定在某个组件的下面；android:layout_toLeftOf 属性指定在某个组件的左边等。

下面的实例通过相对布局，在屏幕上放置四个按钮，B 在 A 的右边，C 在 A 的下面，D 在 B 的，下面 C 的右边。

```
<?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">

    <RelativeLayout
        android:id="@+id/RelativeLayout01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <Button
```

```
        android:text="A"
        android:id="@+id/a"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

        <Button
        android:text="B"
        android:id="@+id/b"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/a"
        ></Button>

        <Button
        android:text="C"
        android:id="@+id/c"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/a"
        ></Button>

        <Button
        android:text="D"
        android:id="@+id/d"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/b"
        android:layout_toRightOf="@+id/c"
        ></Button>

    </RelativeLayout>
</LinearLayout>
```

程序运行结果如图 4.16 所示。

图 4.16　测试 RelativeLayout

4.5.5　绝对布局

通过指定子组件的确切 *X*、*Y* 坐标来确定组件的位置，在 Android 2.0 API 文档中标示该类已过期，可以使用 FrameLayout 或者 RelativeLayout 代替之。

下面的实例是通过绝对布局来排列四个按钮，主要设置参数是 android:layout_x 及 android:layout_y。

```
<?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">
<AbsoluteLayout
    android:id="@+id/AbsoluteLayout01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
```

```
    <Button
    android:text="A"
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="20px"
    android:layout_y="20px"
    ></Button>

    <Button
    android:text="B"
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="60px"
    android:layout_y="20px"
    ></Button>

    <Button
    android:text="C"
    android:id="@+id/Button03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="20px"
    android:layout_y="80px"
    ></Button>

    <Button
    android:text="D"
    android:id="@+id/Button04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="60px"
    android:layout_y="80px"
    ></Button>

  </AbsoluteLayout>
  </LinearLayout>
```

程序运行结果如图 4.17 所示。

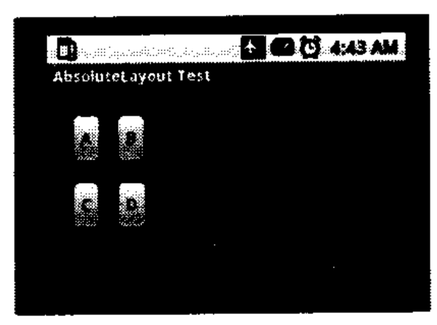


图 4.17　测试 AbsoluteLayout

4.6 组件（Widget）

Android 中的组件可谓应有尽有，从简单的按钮（Button）到复杂的浏览器（WebView）都是无所不能、无所不有。

4.6.1　常用组件

在这里我们以一个用户注册会员为例，演示了有关文本框（TextView）、编辑框

（EditText）、密码文本框（EditText）、单选按钮（RadioButton）、复选按钮（CheckBox）、开关按钮（ToggleButton）和下拉列表（Spinner）的用法。

MainActivity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.RadioButton;
import android.widget.Spinner;
import android.widget.ToggleButton;

public class MainActivity extends Activity {

    // 声明按钮 Button
    private Button register,cancel;
    // 声明 ToggleButton
    private ToggleButton marriged;
    // 声明单选按钮
    private RadioButton male,female;
    // 声明文本编辑框 EditText
    private EditText username,password;
    // 声明下拉列表
    private Spinner position;
    // 声明多选按钮
    private CheckBox reading,swimming;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 获得 EditText 对象
        username = (EditText)findViewById(R.id.username);
        password = (EditText)findViewById(R.id.password);
        // 通过 findViewById 方法获得 RadioButton
        male = (RadioButton)findViewById(R.id.male);
        female = (RadioButton)findViewById(R.id.female);
        // 通过 findViewById 方法获得 CheckBox 实例
        reading = (CheckBox)findViewById(R.id.reading);
        swimming = (CheckBox)findViewById(R.id.swimming);
        // 通过 findViewById 方法获得 ToggleButton 实例
        marriged = (ToggleButton)findViewById(R.id.marriged);
        // 通过 findViewById 方法 获得 Spinner 实例
```

```
        position = (Spinner)findViewById(R.id.position);
        // 下拉列表项数组
        String[] str = {"CEO","CFO","PM"};
        // 数组下拉列表适配器
        ArrayAdapter aa = new ArrayAdapter(this, android.R.layout.simple_spinner_
item,str);
        // 设置下拉列表适配器
        position.setAdapter(aa);
        // 通过 findViewById 方法获得 Button 实例
        register = (Button)findViewById(R.id.register);
        cancel = (Button)findViewById(R.id.cancel);
        // 添加按钮的单击事件监听器
        register.setOnClickListener(new OnClickListener() {
            // 点击事件方法
            public void onClick(View v) {
                // 实例化 Bundle 对象，保存属性
                Bundle b = new Bundle();
                // 在 Bundle 中添加用户名称和用户密码
                b.putString("username", "用户名称："+username.getText().toString());
                b.putString("password", "用户密码："+password.getText().toString());
                // 在 Bundle 添加性别
                if(male.isChecked()){
                    b.putString("gender", "性别：男");
                }else{
                    b.putString("gender", "性别：女");
                }
                String temp = "爱好：";
                if(reading.isChecked()){
                    temp+="阅读";
                }
                if(swimming.isChecked()){
                    temp+=" ";
                    temp+="游泳";
                }
                // 在 Bundle 添加爱好
                b.putString("hobby", temp);
                // 在 Bundle 添加婚否
                if(marriged.isChecked()){
                    b.putString("marriged", "婚否：已婚");
                }else{
                    b.putString("marriged", "婚否：未婚");
                }
                // 在 Bundle 中添加职位
                b.putString("position","职位："+position.getSelectedItem().toString());
                // 实例化 Intent，跳转到 ResultActivity
                Intent intent = new Intent(MainActivity.this,ResultActivity.class);
                // 将 Bundle 添加到 Intent
                intent.putExtra("data", b);
                // 启动 Activity
                startActivity(intent);
            }
        });
```

```
    }
}
```

ResultActivity 代码如下所示：

```
package com.amaker.test;

import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class ResultActivity extends Activity{
   // 声明 ListView
   private ListView listView;

   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置当前 Activity 界面布局
      setContentView(R.layout.result);
      // 通过 findViewById 方法获得 ListView 对象
      listView = (ListView) findViewById(R.id.ListView01);
      // 获得 Intent
      Intent intent = this.getIntent();
      // 从 Intent 中获得 Bundle
      Bundle b = intent.getBundleExtra("data");

      // 实例化 List
      List list =  new ArrayList();
      // 从 Bundle 中获得属性，添加到 List
      list.add(b.getString("username"));
      list.add(b.getString("password"));
      list.add(b.getString("position"));
      list.add(b.getString("gender"));
      list.add(b.getString("hobby"));
      list.add(b.getString("marriged"));
      // 实例化数组适配器
      ArrayAdapter adapter = new ArrayAdapter(this,android.R.layout.simple_list_
item_checked,list);
      // 为 ListView 设置适配器
      listView.setAdapter(adapter);
   }
}
```

main.xml 布局文件代码如下所示：

```
<?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"
    >

<TableLayout
    android:id="@+id/TableLayout01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:collapseColumns="3"
    android:stretchColumns="1"
    >


    <TableRow
    android:id="@+id/TableRow01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="用户名称"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

        <EditText
        android:text=""
        android:id="@+id/username"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        ></EditText>
    </TableRow>

    <TableRow
    android:id="@+id/TableRow02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="用户密码"
        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

        <EditText
        android:text=""
        android:id="@+id/password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:password="true"

        ></EditText>
    </TableRow>
```

```
    <TableRow
    android:id="@+id/TableRow03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="性别"
        android:id="@+id/TextView03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

        <RadioGroup
        android:id="@+id/gender_g"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <RadioButton
        android:text="男"
        android:id="@+id/male"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></RadioButton>

        <RadioButton
        android:text="女"
        android:id="@+id/female"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></RadioButton>


        </RadioGroup>
    </TableRow>


    <TableRow
    android:id="@+id/TableRow04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
        <TextView
        android:text="婚否"
        android:id="@+id/TextView04"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>


    <ToggleButton
    android:text="@+id/ToggleButton01"
    android:id="@+id/marriged"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ToggleButton>
    </TableRow>

    <TableRow
    android:id="@+id/TableRow05"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
            <TextView
            android:text="爱好"
            android:id="@+id/hobby"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>

            <CheckBox
            android:text="阅读"
            android:id="@+id/reading"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></CheckBox>
            <CheckBox
            android:text="游泳"
            android:id="@+id/swimming"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></CheckBox>


        </TableRow>


        <TableRow
        android:id="@+id/TableRow06"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
            <TextView
            android:text="职务"
            android:id="@+id/TextView05"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>

            <Spinner
            android:id="@+id/position"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Spinner>
        </TableRow>



        <TableRow
        android:id="@+id/TableRow07"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
            <Button
            android:text="取消"
            android:id="@+id/cancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

            <Button
```

```
        android:text="注册"
        android:id="@+id/register"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    </TableRow>


</TableLayout>
</LinearLayout>

</LinearLayout>
```

result.xml 布局文件代码如下所示：

```
<?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"
    >
    <ListView
    android:id="@+id/ListView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></ListView>
</LinearLayout>
```

程序运行结果如图 4.18 所示。

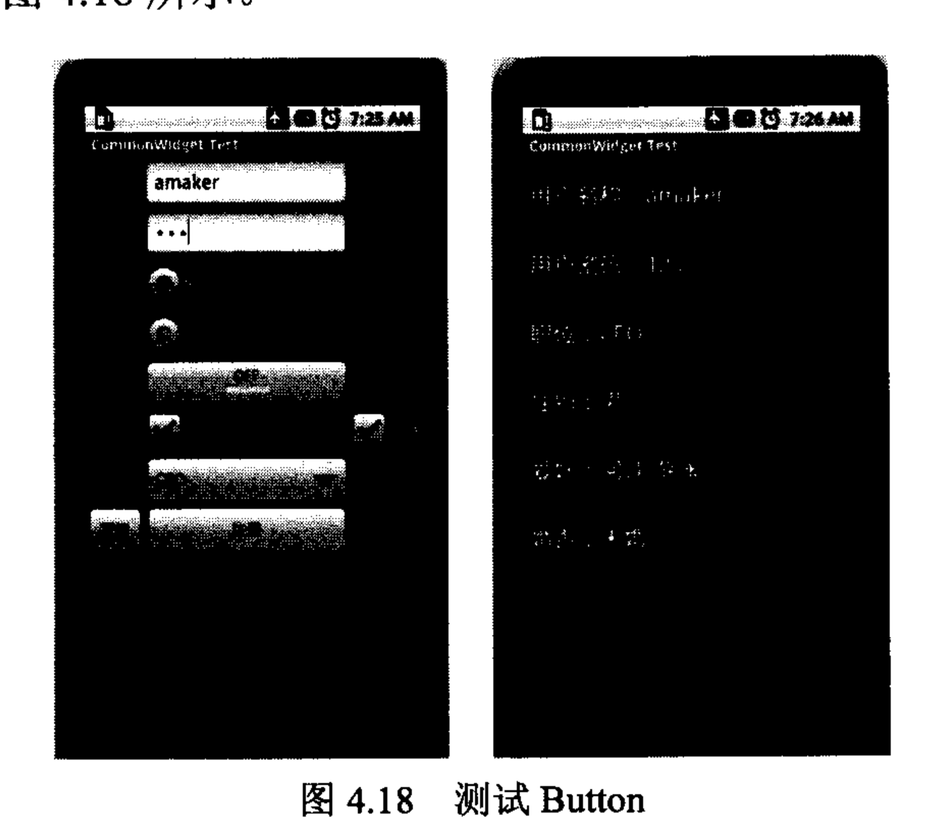


图 4.18　测试 Button

4.6.2　自动完成文本框（AutoCompleteTextView）

当我们在 Google 网站的搜索栏里输入类似“Android”等内容时，和 Android 相关的选项将被列出来，供用户选择。该组件的功能和它类似。

在该组件中，主要是设置要想显示资源的适配器（Adapter）。下面看一个实例。

Activity 代码如下所示：

```
package com.amaker.test;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class MainAcitivity extends Activity {
    // 声明 AutoCompleteTextView
    private AutoCompleteTextView atv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 AutoCompleteTextView 实例
        atv = (AutoCompleteTextView) findViewById(R.id.AutoCompleteTextView01);
        // 声明字符串数组
        String[] strs = { "abc", "abcd", "bcd", "bcde" };
        // 创建适配器
        ArrayAdapter adapter = new ArrayAdapter(this,
                android.R.layout.simple_dropdown_item_1line, strs);
        // 为 AutoCompleteTextView 设置适配器
        atv.setAdapter(adapter);
    }
}
```

布局文件代码如下所示：

```
<?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"
    >
    <TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="自动完成文本框演示"></TextView>

    <AutoCompleteTextView
    android:text=""
    android:id="@+id/AutoCompleteTextView01"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"></AutoCompleteTextView>

</LinearLayout>
```

程序运行结果如图 4.19 所示。

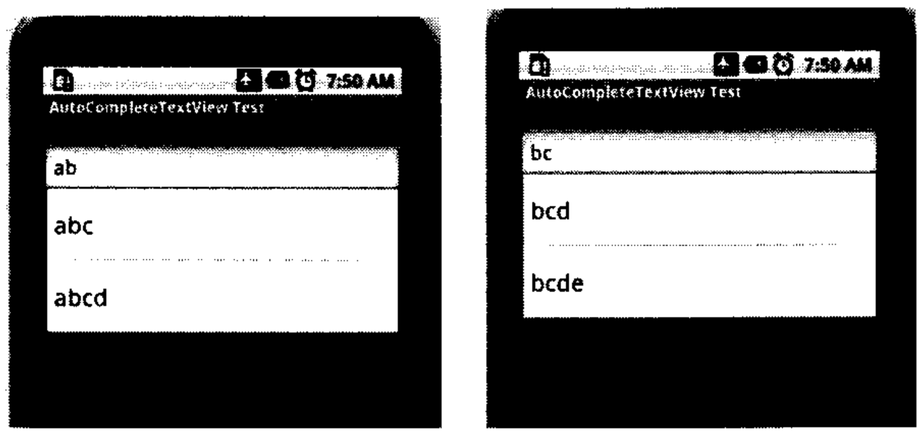


图 4.19　测试自动完成文本框

4.6.3　选项卡（Tab）

选项卡是一个非常方便的分类组件。例如，你可以把通话记录使用选项卡（Tab）来分类，将所有通话记录、已接来电、未接来电等分类显示。下面的实例就是使用 Tab 来完成这个功能的。

使用 Tab 组件的步骤说明如下。

① 在布局文件中使用 FrameLayout 列出 Tab 组件及 Tab 中的内容组件。

② Activity 要继承 TabActivity。

③ 调用 TabActivity 的 getTabHost()方法获得 TabHost 对象。

④ 通过 TabHost 创建 Tab 选项。

Activity 代码如下所示：

```
package com.amaker.test;
import android.app.TabActivity;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.widget.TabHost;
// 继承 TabActivity
public class MainActivity extends TabActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 获得 TabHost
        TabHost th = getTabHost();

        LayoutInflater.from(this).inflate(R.layout.main, th.getTabContentView(), true);
        // 设置 Tab 的标签内容和显示内容
        th.addTab(th.newTabSpec("all").setIndicator("所有通话记录").setContent(R.id.
TextView01));
        // 设置 Tab 的标签内容和显示内容
        th.addTab(th.newTabSpec("ok").setIndicator("已接来电").setContent(R.id.Text
View02));
        // 设置 Tab 的标签内容和显示内容
        th.addTab(th.newTabSpec("cancel").setIndicator("未接来电").setContent(R.id.
TextView03));
```

```
    }
}
```

布局文件代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/FrameLayout01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

   <TabHost
   android:id="@+id/TabHost01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TabHost>

   <TextView
   android:id="@+id/TextView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="所有通话记录"></TextView>

   <TextView
   android:id="@+id/TextView02"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="已接来电"></TextView>

   <TextView
   android:id="@+id/TextView03"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="未接来电"></TextView>

</FrameLayout>
```

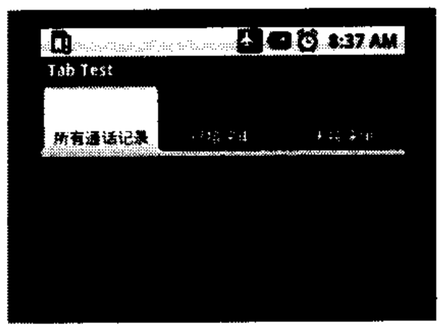


图 4.20　测试选项卡

程序运行结果如图 4.20 所示。

另外，关于 Tab 的内容还可以通过实现一个接口 TabHost.TabContentFactory 的 createTabContent 方法来指定，参考下面的实例。

```
package com.amaker.test;

import java.util.ArrayList;
import java.util.List;

import android.app.TabActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
```

```
import android.widget.ListView;
import android.widget.TabHost;
// 继承 TabActivity 实现 TabHost.TabContentFactory 接口
public class MainActivity extends TabActivity implements
        TabHost.TabContentFactory {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 获得 TabHost 对象
        TabHost th = getTabHost();
        // 为 Tab 添加标签内容和显示内容
        th.addTab(th.newTabSpec("all").setIndicator("所有通话记录").setContent(this));
        // 为 Tab 添加标签的内容和显示内容
        th.addTab(th.newTabSpec("ok").setIndicator("已接来电").setContent(this));
        // 为 Tab 添加标签内容和显示内容
        th.addTab(th.newTabSpec("cancel").setIndicator("未接来电").setContent(this));
    }
    // 创建 Tab 方法
    public View createTabContent(String tag) {
        // 实例化列表视图 ListView
        ListView lv = new ListView(this);
        // 实例化 List
        List<String> list = new ArrayList<String>();
        // 在 list 中添加标签内容
        list.add(tag);
        // 在 list 中添加显示内容
        if(tag.equals("all")){
            list.add("tom");
            list.add("kite");
            list.add("rose");
        }else if(tag.equals("ok")){
            list.add("tom");
            list.add("kite");
        }else{
            list.add("rose");
        }
        // 实例化数组适配器
        ArrayAdapter adapter = new ArrayAdapter(this,
                android.R.layout.simple_list_item_checked, list);
        // 为 ListView 设置适配器
        lv.setAdapter(adapter);
        return lv;
    }
}
```

程序运行结果如图 4.21 所示。

另外，setContent(Intent i)也可以传递一个 Intent 对象，用来添加内容。

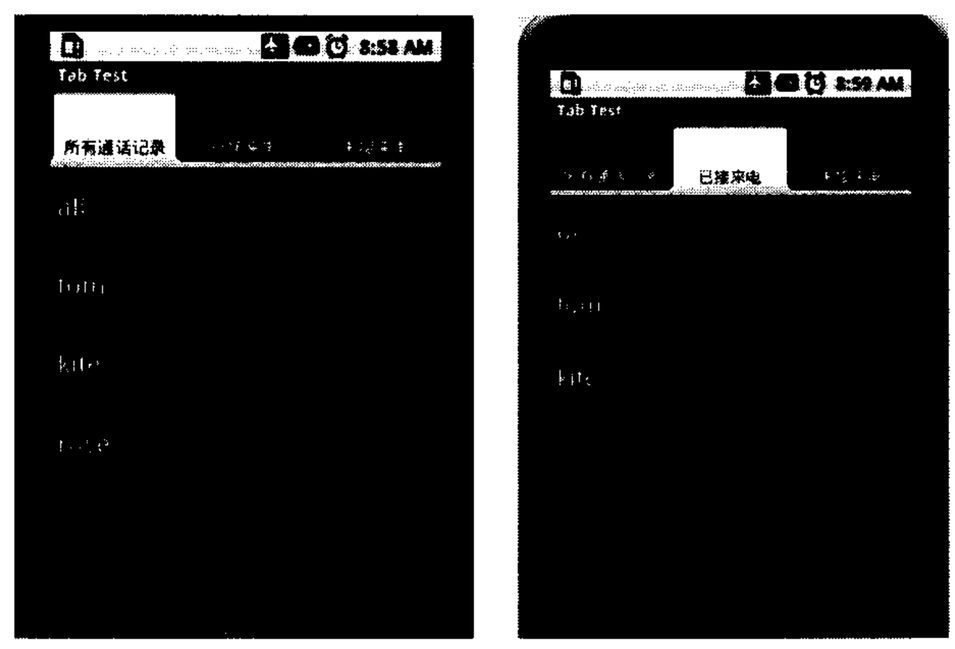


图 4.21　测试选项卡

4.6.4　进度条（ProgressBar）

我们在安装一些软件时，经常会看到进度条显示安装进度。或者我们在执行一些比较耗时的程序时，系统会弹出一个进度对话框提示用户等待。

在 Android 系统中进度条有很多种：对话框进度条、标题栏进度条和水平进度条。我们首先来看看对话框进度条。

1．对话框进度条

创建对话框进度条需要如下步骤：

① 覆盖 Activity 的 onCreateDialog()方法，并在其中创建对话框。

② 调用 Activity 的 showDialog()方法，显示进度条对话框。

Activity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
   private Button myBtn;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置 Activity 界面布局
      setContentView(R.layout.main);
      // 通过 findViewById 方法获得 Button 对象
```

```
        myBtn = (Button)findViewById(R.id.Button01);
        // 为 Button 设置单击事件监听器
        myBtn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 显示对话框
                showDialog(0);
            }
        });
    }

    @Override
    protected Dialog onCreateDialog(int id) {
        // 实例化进度条对话框 ProgressDialog
        ProgressDialog dialog = new ProgressDialog(this);
        // 可以不显示标题
        dialog.setTitle("测试对话框");
        dialog.setIndeterminate(true);
        dialog.setMessage("程序正在加载请稍候！");
        dialog.setCancelable(true);
        return dialog;
    }
}
```

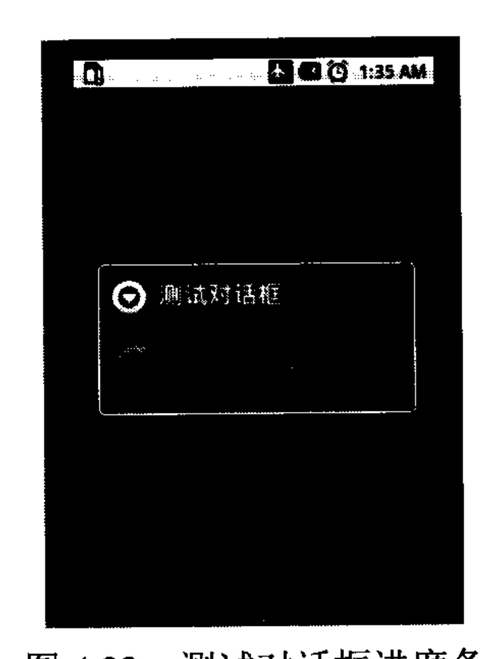


图 4.22　测试对话框进度条

程序运行结果如图 4.22 所示。

2. 标题栏进度条

创建标题栏进度条的步骤说明如下。

① 调用 Activity 的 requestWindowFeature()方法，获得进度条。

② 调用 Activity 的 setProgressBarIndeterminateVisibility()方法显示进度条。

Activity 代码如下所示：

```
package com.amaker.test;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
    // 声明 Button 对象
    private Button b1,b2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置窗口特征
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 对象
```

```
        b1 = (Button)findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 Button 对象
        b2 = (Button)findViewById(R.id.Button02);
        // 为按钮设置单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            // 在窗口标题栏显示进度条
            setProgressBarIndeterminateVisibility(true);
         }
        });

        b2.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            // 在窗口标题栏取消进度条
            setProgressBarIndeterminateVisibility(false);
         }
        });
    }
}
```

程序运行结果如图 4.23 所示。

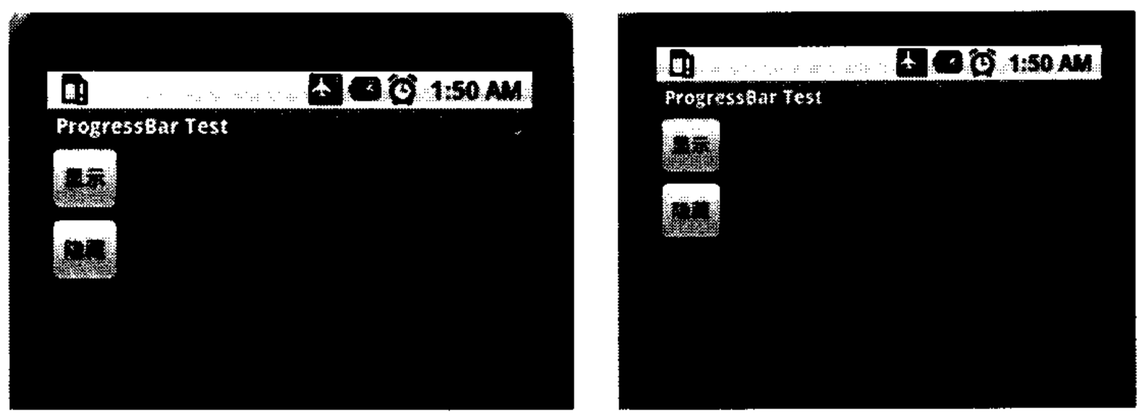


图 4.23　测试对话框进度条

3. 水平进度条

创建水平进度条的步骤说明如下。

① 在布局文件中声明 ProgressBar。

② 在 Activity 中获得 ProgressBar 实例。

③ 调用 ProgressBar 的 incrementProgressBy()方法增加或减少进度。

Activity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;

public class MainActivity extends Activity {
```

```
 // 声明按钮
private Button b1,b2;
 // 声明进度条 ProgressBar
ProgressBar progressBar;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 设置当前 Activity 界面布局
    setContentView(R.layout.main);
    // 通过 findViewById 方法获得 Button 实例
    b1 = (Button)findViewById(R.id.Button01);
    // 通过 findViewById 方法获得 Button 实例
    b2 = (Button)findViewById(R.id.Button02);
    // 通过 findViewById 方法获得 ProgressBar 实例
    progressBar = (ProgressBar)findViewById(R.id.ProgressBar01);
    // 为按钮添加单击事件监听器
    b1.setOnClickListener(new OnClickListener() {
       public void onClick(View v) {
              // 增加进度
             progressBar.incrementProgressBy(1);
       }
    });
    // 为按钮添加单击事件监听器
    b2.setOnClickListener(new OnClickListener() {
       public void onClick(View v) {
            // 减少进度
           progressBar.incrementProgressBy(-1);
       }
    });
  }
}
```

布局文件代码如下所示：

```
<?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"
   >
<ProgressBar
   android:id="@+id/ProgressBar01"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="200dip"
   android:layout_height="wrap_content"
   android:max="100"
   android:progress="50"
></ProgressBar>

<Button
android:id="@+id/Button01"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:text="增加"></Button>

<Button
android:id="@+id/Button02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="减少"></Button>

</LinearLayout>
```

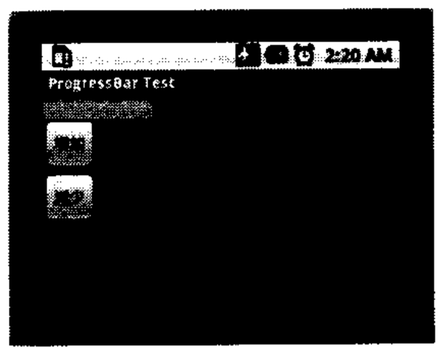


图 4.24　测试对话框进度条

程序运行结果如图 4.24 所示。

4.6.5　日期、时间选择对话框（DataPickerDialog、TimePickerDialog）

在系统中我们经常会用到一些日期、时间的选择。比如，更改系统时间、设置闹钟、输入生日等。Android 为我们提供了非常人性化的日期、时间选择组件。

创建日期、时间选择对话框的步骤说明如下。

① 调用 Activity 的 onCreateDialog()方法创建对话框。

② 分别在 OnDateSetListener 的 onDateSet()方法和 OnTimeSetListener 的 onTimeSet()事件方法中更改日期和时间。

③ 调用 Activity 的 showDialog()方法显示对话框。

Activity 代码如下所示：

```
package com.amaker.test;

import java.util.Calendar;
import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.app.DatePickerDialog.OnDateSetListener;
import android.app.TimePickerDialog.OnTimeSetListener;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;

public class MainActivity extends Activity {
   // 声明按钮
   private Button b1,b2;
   // 声明文本视图
   private TextView tv1,tv2;
   // 声明日历
   private Calendar c;
```

```
    // 年、月、日
    private int m_year,m_month,m_day;
    // 小时、分钟
    private int m_hour,m_minute;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 对象
        b1 = (Button)findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 Button 对象
        b2 = (Button)findViewById(R.id.Button02);
        // 获得日历实例
        c = Calendar.getInstance();
        // 获得年
        m_year = c.get(Calendar.YEAR);
        // 获得月
        m_month = c.get(Calendar.MONTH);
        // 获得日
        m_day = c.get(Calendar.DAY_OF_MONTH);
        // 获得小时
        m_hour = c.get(Calendar.HOUR);
        // 获得分钟
        m_minute = c.get(Calendar.MINUTE);
        // 通过 findViewById 方法获得 TextView 实例
        tv1 = (TextView)findViewById(R.id.TextView01);
        // 为 TextView 设置文本内容，显示日期
        tv1.setText(m_year+":"+(m_month+1)+":"+m_day);
        // 通过 findViewById 方法获得 TextView 对象
        tv2 = (TextView)findViewById(R.id.TextView02);
        // 为 TextView 设置文本内容，显示时间
        tv2.setText(m_hour+":"+m_minute);
        // 为按钮添加单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 显示日期对话框
                showDialog(0);
            }
        });
        // 为按钮添加单击事件监听器
        b2.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 显示时间对话框
                showDialog(1);
            }
        });
    }
    // 显示对话框方法
    protected Dialog onCreateDialog(int id) {
        if(id==0)
```

```
        // 显示日期
            return new DatePickerDialog(this,
                    l1,
                    m_year, m_month, m_day);
        else
        // 显示时间
            return new TimePickerDialog(this,
                    l2, m_hour, m_minute, false);
    }
    // 日期设置监听器
    private OnDateSetListener l1 = new OnDateSetListener() {
        public void onDateSet(DatePicker view, int year, int monthOfYear,
                int dayOfMonth) {
            m_year = year;
            m_month = monthOfYear;
            m_day = dayOfMonth;
            // 为 TextView 设置文本内容，重新显示当前日期
            tv1.setText(m_year+":"+(m_month+1)+":"+m_day);
        }
    };
    // 时间设置监听器
    private OnTimeSetListener l2 = new OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            m_hour = hourOfDay;
            m_minute = minute;
            // 为 TextView 设置文本内容，重新显示当前时间
            tv2.setText(m_hour+":"+m_minute);
        }
    };
}
```

程序运行结果如图 4.25 所示。

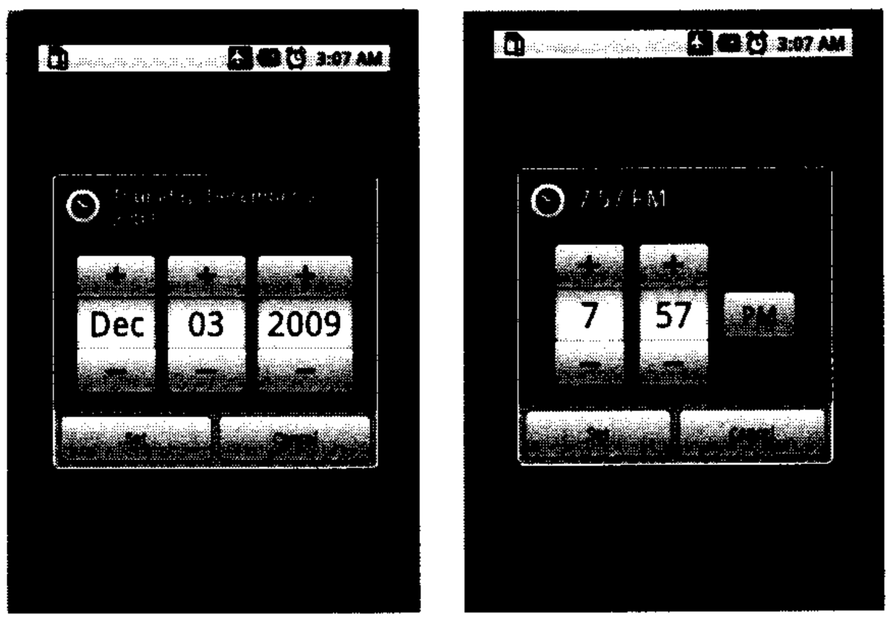


图 4.25　测试日期、时间选择对话框

4.6.6　列表视图（ListView）

列表视图可以说是 Android 中最为常用的一种视图组件，它以垂直列表的方式列出需

要显示的列表项。例如，显示联系人名单、系统设置项等，都用到了 ListView。

创建 ListView 既可以使用 ListView 组件，也可以继承 ListActivity。在使用中重要的是如何设置选项内容，也就是如何设置 Adapter，Adapter 既可以是一个简单的数组 Adapter（ArrayAdapter），也可以是一个游标 Adapter（SimpleCursorAdapter），还可以继承 BaseAdapter 展示其他视图（例如图片等）。

1. 使用 ArrayAdapter 添加列表视图内容

其具体实现步骤说明如下。

① 创建 ArrayAdapter。

② 调用 ListActivity 的 setListAdapter()方法显示列表项。

ListActivity 代码如下所示：

```
package com.amaker.test;
import android.app.ListActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;

public class MainActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 列表项数组
        String[] strs = {"Java","C","C++","VB"};
        // 数组适配器
        ArrayAdapter<String>
adapter = new ArrayAdapter<String>(this,android.R.layout.
simple_list_item_1,strs);
        // 为 ListView 设置适配器
        setListAdapter(adapter);
    }
}
```

程序运行结果如图 4.26 所示：

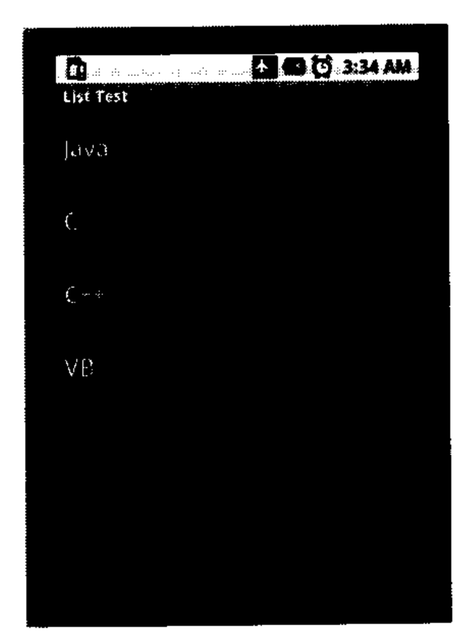


图 4.26　测试 ListView

2. 使用 SimpleCursorAdapter 添加列表视图内容

在这里我们使用 ListView 将 Android 中的联系人显示出来。

① 调用 getContentResolver()方法返回 ContentResolver 对象，通过该对象查询得到 Cursor 对象。

② 创建 SimpleCursorAdapter 对象。

③ 调用 ListActivity 的 setListAdapter()方法显示列表项。

ListActivity 代码如下所示：

```
package com.amaker.test;

import android.app.ListActivity;
```

```
import android.database.Cursor;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;

public class MainActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 获得通讯录联系人游标对象 Cursor
        Cursor c = getContentResolver().query(People.CONTENT_URI, null, null, null, null);
        startManagingCursor(c);

          // 实例化列表适配器
          ListAdapter adapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_1,
                c,
                new String[] {People.NAME} ,
                new int[] {android.R.id.text1});
        // 为 ListView 设置列表适配器
        setListAdapter(adapter);
    }
}
```

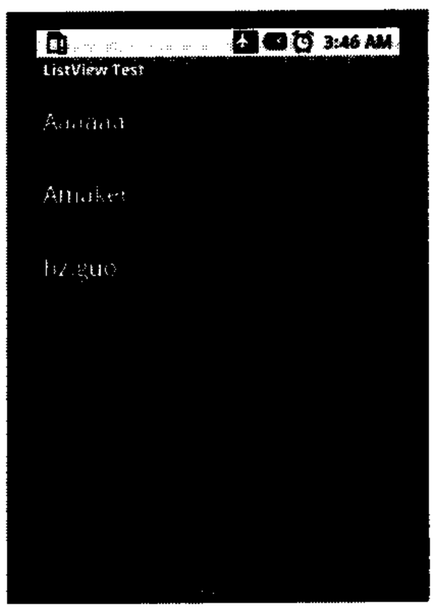


图 4.27　测试 ListView

程序运行结果如图 4.27 所示。

4.6.7　网格视图（GridView）

GridView 按照行列的方式来显示内容，一般适合显示图标、图片等内容，主要用于设置 Adapter。

在这里主要是基础 BaseAdapter 类，重写其中的方法，主要是重写 getView()方法设置图片显示格式。

Activity 代码如下所示：

```
package com.amaker.test;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class MainActivity extends Activity {
    // 声明 GridView 对象
    private GridView gv;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 GridView 对象
        gv = (GridView)findViewById(R.id.GridView01);
        // 设置 GridView 列数
        gv.setNumColumns(4);
        // gv.setNumColumns(3);
        // 在 GridView 中显示字符串
        // String[] strs = {"a","a1","a2","b","b1","b2","c","c1","c2"};
        //ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.
layout.simple_gallery_item,strs);
        // 为 GridView 设置适配器
        gv.setAdapter(new MyAdapter(this));
    }
    // 自定义适配器
    class MyAdapter extends BaseAdapter{
    // 图片 ID 数组
    private Integer[] imgs = {
                R.drawable.gallery_photo_1,
                R.drawable.gallery_photo_2,
                R.drawable.gallery_photo_3,
                R.drawable.gallery_photo_4,
                R.drawable.gallery_photo_5,
                R.drawable.gallery_photo_6,
                R.drawable.gallery_photo_7,
                R.drawable.gallery_photo_8,

                R.drawable.gallery_photo_1,
                R.drawable.gallery_photo_2,
                R.drawable.gallery_photo_3,
                R.drawable.gallery_photo_4,
                R.drawable.gallery_photo_5,
                R.drawable.gallery_photo_6,
                R.drawable.gallery_photo_7,
                R.drawable.gallery_photo_8
    };
     // 上下文对象
     Context context;
     // 构造方法
     MyAdapter(Context context){
        this.context = context;
     }
        // 获得数量
public int getCount() {
            return imgs.length;
        }
        // 获得当前选项
        public Object getItem(int item) {
            return item;
```

```
        }
        // 获得当前选项 ID
        public long getItemId(int id) {
            return id;
        }
        // 创建 View 方法
        public View getView(int position, View convertView, ViewGroup parent) {
            ImageView imageView;
              if (convertView == null) {
                  // 实例化 ImageView 对象
                imageView = new ImageView(context);
                  // 设置 ImageView 对象布局
                imageView.setLayoutParams(new GridView.LayoutParams(45, 45));
                  // 设置边界对齐
                imageView.setAdjustViewBounds(false);
                  // 设置刻度类型
                imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                  // 设置间距
                imageView.setPadding(8, 8, 8, 8);
              } else {
                imageView = (ImageView) convertView;
              }
               // 为 ImageView 设置图片资源
              imageView.setImageResource(imgs[position]);
              return imageView;
        }
    }
}
```

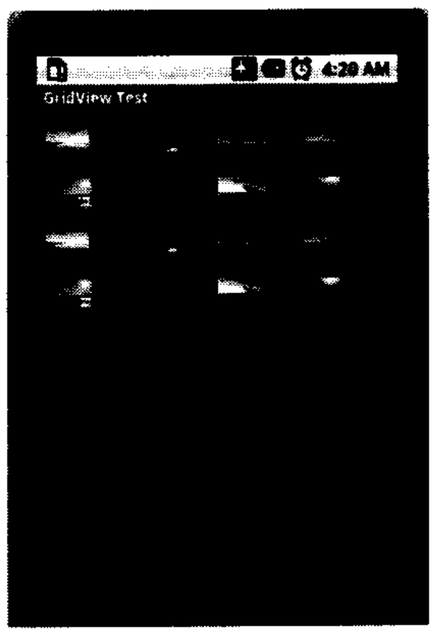


图 4.28　测试 GridView

程序运行结果如图 4.28 所示。

4.6.8　画廊视图（Gallery）

Gallery 能够水平方向显示其内容，一般用来浏览图片，被选中的选项位于中间，并且可以响应事件显示信息。下面结合 ImageSwitcher 组件来实现一个通过缩略图来浏览照片的程序。程序步骤说明如下。

① 创建一个 Android 工程“chapter04_UI_gallary”，该工程的入口 Activity 类 MainActivity 继承 Activity 并实现 OnItemSelectedListener 接口和 ViewFactory 接口，来实现图片单击事件和视图创建。

```
public class MainActivity extends MapActivity {

package com.amaker.test;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
```

```
import android.view.Window;
import android.view.animation.AnimationUtils;
import android.widget.AdapterView;
import android.widget.BaseAdapter;
import android.widget.Gallery;
import android.widget.ImageSwitcher;
import android.widget.ImageView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.Gallery.LayoutParams;
import android.widget.ViewSwitcher.ViewFactory;
// 继承Activity，实现OnItemSelectedListener和ViewFactory接口
public class MainActivity extends Activity implements OnItemSelectedListener,
        ViewFactory {
    @Override// Activity创建时调用
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override// 选项被选择方法
    public void onItemSelected(AdapterView<?> adapter, View v, int position,
            long id) {
    }

    @Override// 没有任何选择方法
    public void onNothingSelected(AdapterView<?> arg0) {
    }

    @Override // 创建视图方法
    public View makeView() {
        return null;
    }
}
```

② 在工程的 res\drawable\目录下添加 7 张图片和对应的缩略图片。

③ 在工程的 res\layout\目录下创建一个布局文件 main.xml。在其中添加一个 Gallery 组件和一个 ImageSwitcher 组件，并设置相应属性。

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

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

    <Gallery android:id="@+id/gallery"
        android:background="#55000000"
```

```
        android:layout_width="fill_parent"
        android:layout_height="60dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:gravity="center_vertical"
        android:spacing="16dp"
    />

</RelativeLayout>
```

④ 在 MainActivity 顶部声明使用到的 ImageSwitcher 实例和图片资源 Integer 数组。

```
    // 声明 ImageSwitcher
private ImageSwitcher mSwitcher;
    // 图片 ID 数组
    private Integer[] mThumbIds = { R.drawable.sample_thumb_0,
          R.drawable.sample_thumb_1, R.drawable.sample_thumb_2,
          R.drawable.sample_thumb_3, R.drawable.sample_thumb_4,
          R.drawable.sample_thumb_5, R.drawable.sample_thumb_6,
          R.drawable.sample_thumb_7 };
    // 图片 ID 数组
    private Integer[] mImageIds = { R.drawable.sample_0, R.drawable.sample_1,
          R.drawable.sample_2, R.drawable.sample_3, R.drawable.sample_4,
          R.drawable.sample_5, R.drawable.sample_6, R.drawable.sample_7 };
```

⑤ 在 MainActivity 的 onCreate()方法中，将窗口样式设置为无标题，设置当前布局视图，获得 ImageSwitcher 实例，并设置渐进渐出动画，获得 Gallery 实例。

```
    // 设置窗口特征无标题
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    // 设置 Activity 界面布局
    setContentView(R.layout.main);
    // 通过 findViewById 方法获得 ImageSwitcher 对象
    mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
    // 为 ImageSwitcher 设置工厂
    mSwitcher.setFactory(this);
    // 设置动画效果
    mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
          android.R.anim.fade_in));
    // 设置动画效果
    mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
          android.R.anim.fade_out));
     // 通过 findViewById 方法获得 Gallery 对象
    Gallery g = (Gallery) findViewById(R.id.gallery);
```

⑥ 创建内部类 ImageAdapter，该类继承 BaseAdapter。为 Gallery 设置 Adapter 实例。

```
public class ImageAdapter extends BaseAdapter {
    // 构造方法
    public ImageAdapter(Context c) {
        mContext = c;
    }
      // 获得数量
```

```
        public int getCount() {
            return mThumbIds.length;
        }
         // 获得当前选项
        public Object getItem(int position) {
            return position;
        }
         // 获得当前选项 ID
        public long getItemId(int position) {
            return position;
        }
         // 获得 View 对象
        public View getView(int position, View convertView, ViewGroup parent) {
             // 实例化 ImageView 对象
             ImageView i = new ImageView(mContext);
             // 设置图片资源
             i.setImageResource(mThumbIds[position]);
             // 设置边界对齐
             i.setAdjustViewBounds(true);
             // 设置布局参数
             i.setLayoutParams(new Gallery.LayoutParams(
                    LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
             // 设置背景资源
             i.setBackgroundResource(R.drawable.picture_frame);
             return i;
        }
        private Context mContext;
    }
```

⑦ 实现 onItemSelected()方法，更换图片。

```
    @Override
    public void onItemSelected(AdapterView<?> adapter, View v, int position,
            long id) {
        // 设置图片资源
        mSwitcher.setImageResource(mImageIds[position]);
    }
```

⑧ 实现 makeView()方法，为 ImageView 设置布局格式。

```
    @Override
    public View makeView() {
         // 创建 ImageView
         ImageView i = new ImageView(this);
         // 设置背景色
         i.setBackgroundColor(0xFF000000);
         // 设置精度类型
         i.setScaleType(ImageView.ScaleType.FIT_CENTER);
         // 设置布局参数
         i.setLayoutParams(new ImageSwitcher.LayoutParams(
                LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
         return i;
    }
```

⑨ 为 Gallery 添加 Adapter 并添加 OnItemSelectedListener 监听器。

```
g.setAdapter(new ImageAdapter(this));
g.setOnItemSelectedListener(this);
```

程序运行结果如图 4.29 所示。

图 4.29　测试 Gallery

4.6.9　地图视图（MapView）

Google 的地图服务在全世界来说也是最抢眼的，在 Android 中也可以使用地图服务，使用 MapView 组件便可以实现。使用 MapView 时需要注意以下几点：

- 在创建项目时“Build Target”要选择“Google APIs”，也就是要添加 Google 的 API jar 文件 map.jar。
- 在选择“AVD”时要使“Target”为“Google APIs”，具体配置如图 4.30 所示。

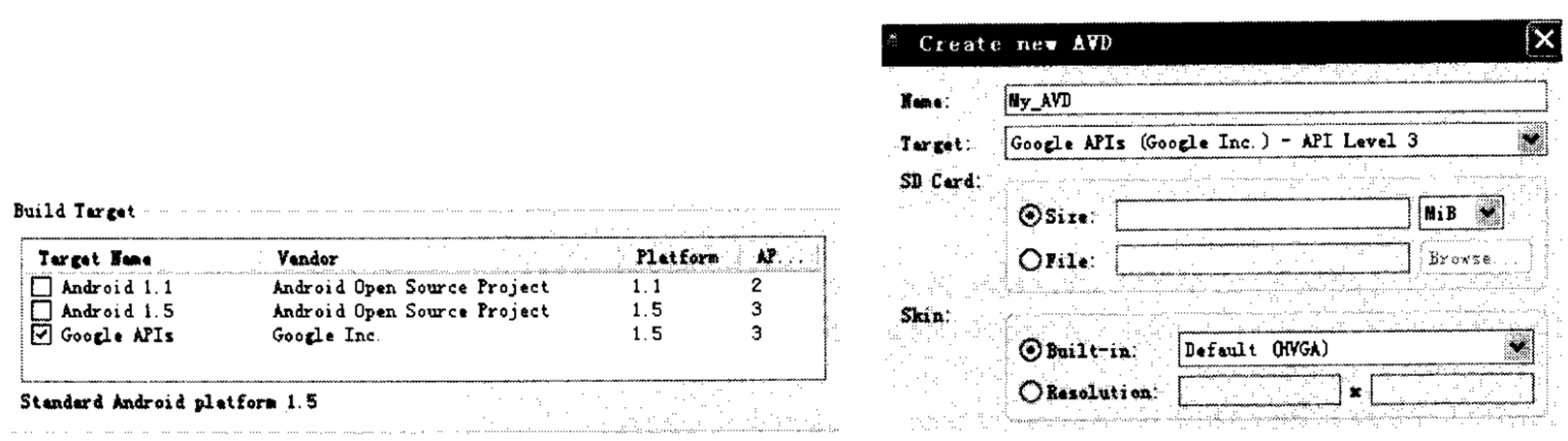


图 4.30　MapView 项目配置

- 在 AndroidManifiest.xml 配置文件中添加<uses-library android:name="com.google.android.maps"/>。
- 在 AndroidManifiest.xml 配置文件中声明权限<uses-permission android:name="android.permission.INTERNET" />。
- 获得“Google Map API key”，使用 JDK 的 keytool 生成 MD5 key，如图 4.31 所示。

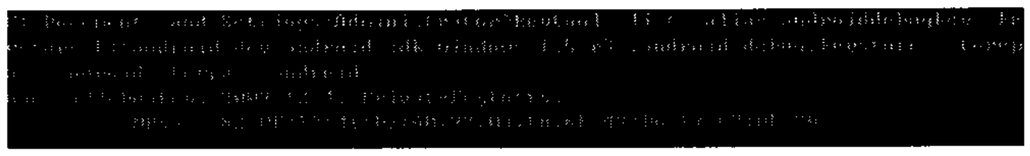


图 4.31　keytool 生成 key

- 获得 MD5 认证指纹后到 Google 网站申请 ApiKey。申请地址为 http://code.google.com/intl/zh-CN/android/maps-api-signup.html。当输入 MD5 认证指纹后，生成结果如图 4.32 所示。

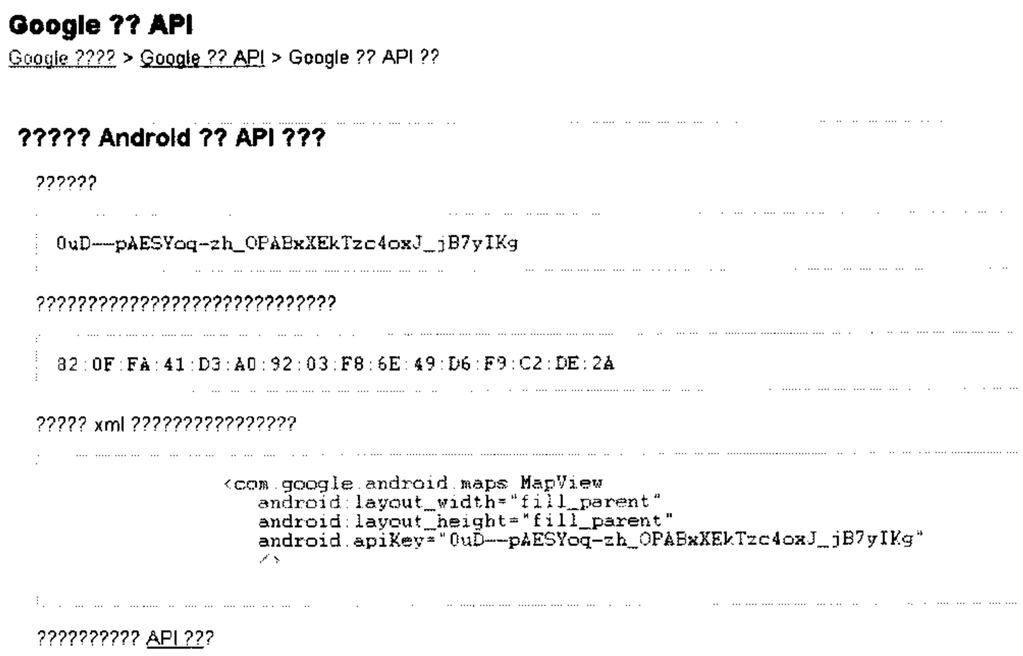


图 4.32　输入 MD5 认证指纹后的结果

- Activity 要继承 MapActivity。

  Activity 代码如下所示：

```
package com.amaker.test;

import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.ZoomControls;

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;

public class MainActivity extends MapActivity {
    // 声明线性布局
    LinearLayout linearLayout;
    // 声明地图视图
    MapView mapView;
    // 声明缩放控制
    ZoomControls mZoom;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
```

```
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 LinearLayout 实例
        linearLayout = (LinearLayout) findViewById(R.id.zoomview);
        // 通过 findViewById 获得 MapView 实例
        mapView = (MapView) findViewById(R.id.mapview);
        // 从 MapView 中获得 ZoomControls 实例
        mZoom = (ZoomControls) mapView.getZoomControls();
        // 在 LinearLayout 中添加 ZoomControls
        linearLayout.addView(mZoom);
    }

    @Override// 是否显示路径
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

布局文件代码如下所示：

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

    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0uD--pAESYoq-zh_OPABxXEkTzc4oxJ_jB7yIKg"
    />

    <LinearLayout
    android:id="@+id/zoomview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@id/mapview"
    android:layout_centerHorizontal="true"
/>

</RelativeLayout>
```

清单文件代码如下所示：

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

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <uses-library android:name="com.google.android.maps" />
</application>

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />

<uses-sdk android:minSdkVersion="3" />

</manifest>
```

程序运行结果如图 4.33 所示。

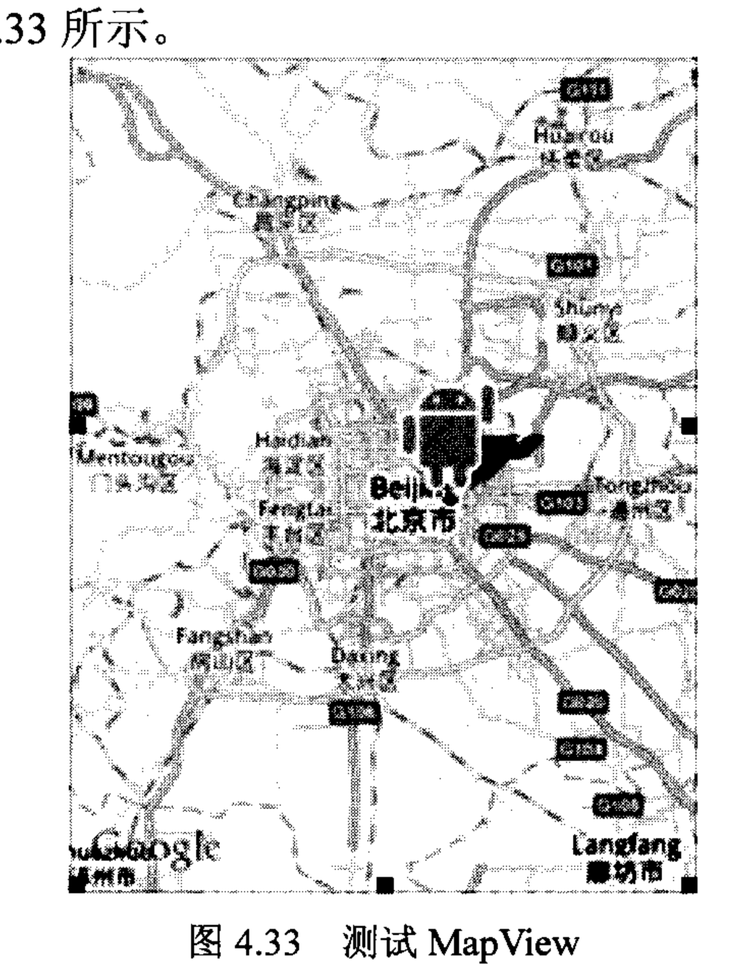


图 4.33　测试 MapView

4.6.10　网络视图（WebView）

Android 的网络功能很强大，WebView 组件支持直接加载网页，可以将其视为一个浏览器。要实现该功能需要如下步骤：

① 在布局文件中声明 WebView。

② 在 Activity 中实例化 WebView。

③ 调用 WebView 的 loadUrl()方法，加载指定的 URL 地址网页。

④ 为了让 WebView 能够响应超链接功能，调用 setWebViewClient()方法设置 Web View 客户端。

⑤ 为了让 WebView 支持回退功能，覆盖 onKeyDown()方法。

⑥ 在清单文件中添加访问互联网权限：

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

Activity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MainActivity extends Activity {
    WebView webview;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 WebView 对象
        webview = (WebView) findViewById(R.id.webview);
        // 设置 WebView 属性，能够执行 JavaScript 脚本
        webview.getSettings().setJavaScriptEnabled(true);
        // 加载 Url 内容
        webview.loadUrl("http://www.google.com");
        // 设置 Web 视图客户端
        webview.setWebViewClient(new HelloWebViewClient ());
    }

    @Override// 设置回退
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && webview.canGoBack()) {
            webview.goBack();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }

    // Web 视图客户端
    private class HelloWebViewClient extends WebViewClient {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            view.loadUrl(url);
            return true;
        }
    }
}
```

布局文件代码如下所示：

```
<?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"
    >

    <WebView
        android:id="@+id/webview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />

</LinearLayout>
```

程序运行结果如图 4.34 所示。

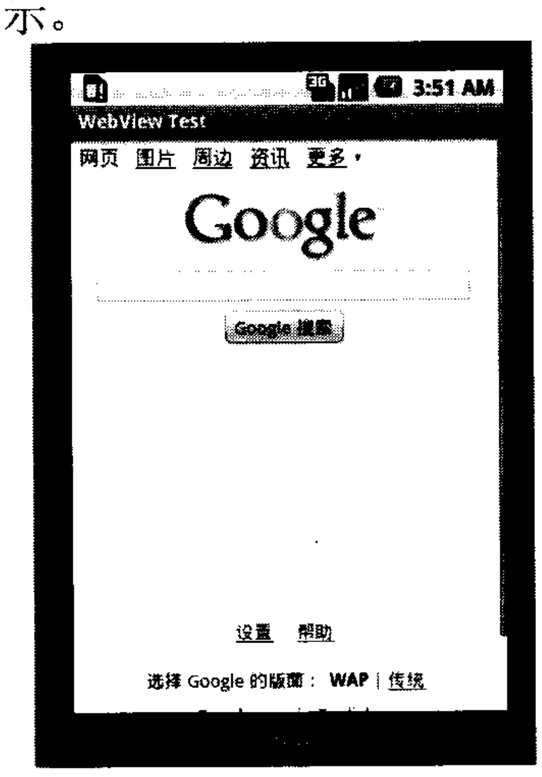
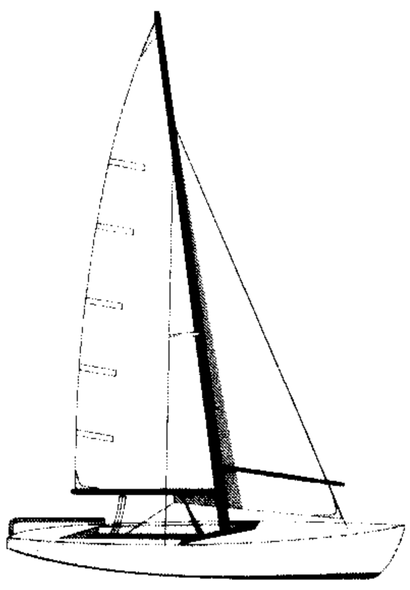


图 4.34　测试 WebView



第 5 章　Android 基本程序单元 Activity

在前面的学习中我们提到了，Android 系统由 Activity、Service、Broadcast Receiver 和 Content Provider 组成。其中 Activity 是使用频率最高、最重要的组件了。在 Android 系统中 Activity 提供可视化的用户界面，一个 Android 应用通常由多个 Activity 组成。多个 Activity 组成了 Activity 栈（Stack），当前活动的 Activity 处于栈顶。Activity 有自己的生命周期，由 Android 系统来控制。

在 Activity 的上面是一个 Window 对象，在 Window 之上通常是一个布局容器对象（如一个 LinearLayout），再上面是我们要展示的组件，如按钮、文本框、列表框等。

本章我们将对 Activity 的生命周期、Activity 的创建和使用做全面的讲解。

5.1 何调回调

在 4.4 节里面，我们讲述了事件处理机制，其实这里面就有方法的回调。本章我们将要学到很多其他的回调方法，所以有必要把回调拿出来介绍一下。

我们都知道在 C、C++或者 Java Script 中经常有回调方法，这种回调是通过指针来实现的。那么，在 Java 中不允许直接操作指针，回调又该如何实现呢？在 Java 当中回调是通过接口来实现的。

方法回调是功能定义和功能实现分离的一种手段，是一种松耦合设计思想。作为一种系统架构，必须有自己的运行环境，并且提供用户的实现接口。

例如，在 Activity 中定义了很多生命周期中不同状态要调用的方法，这些方法都是空实现，系统框架要调用，用户也要调用来实现。

下面通过实例来模拟一下 Android 中 Activity 的方法回调思想。

1. Activity 接口（为了说明问题这里只定义三个方法）

```
package com.amaker.test;
// 定义接口
public interface Activity {
    // 创建时调用的方法
   public void onCreate();
    // 启动时调用的方法
   public void onStart();
    // 销毁时调用的方法
   public void onDestroy();
}
```

2. Activity 接口的实现类 MyActivity

```
package com.amaker.test;
// 定义一个类实现 Activity 接口
public class MyActivity implements Activity{
   @Override// 实现创建方法，简单输出提示信息
   public void onCreate() {
       System.out.println("onCreate..........");
   }
   @Override// 实现启动方法，简单输出信息
   public void onStart() {
       System.out.println("onStart..........");
   }
   @Override// 实现销毁方法，简单输出信息
   public void onDestroy() {
       System.out.println("onDestroy..........");
   }
}
```

3. 系统运行环境类 AndroidSystem

```
package com.amaker.test;
// 系统运行环境类
public class AndroidSystem {
    // 定义创建常量
   public static final int CREATE=1;
    // 定义启动常量
   public static final int START=2;
    // 定义销毁常量
   public static final int DESTORY=3;
    // 运行方法
   public void run(Activity a,int state){
       switch (state) {
        // 创建
       case CREATE:
         a.onCreate();
         break;
        // 启动
       case START:
         a.onStart();
```

```
            break;
          // 销毁
         case DESTORY:
            a.onDestroy();
            break;
        }
    }
}
```

4. 测试类

```
package com.amaker.test;
// 测试类
public class Test {
    // 主方法
   public static void main(String[] args) {
          // 实例化 AndroidSystem
      AndroidSystem system = new AndroidSystem();
          // 实例化 MyActivity
      Activity a = new MyActivity();
          //  创建
      system.run(a, AndroidSystem.CREATE);
          // 启动
      system.run(a, AndroidSystem.START);
          // 销毁
      system.run(a, AndroidSystem.DESTORY);
   }
}
```

通过上述代码我们可以看出，接口（系统框架）是系统提供的，接口的实现是用户实现的。这样可以达到接口统一，实现不同。

系统通过在不同的状态“回调”我们的实现类，来达到接口和实现的分类。

程序运行结果如图 5.1 所示。

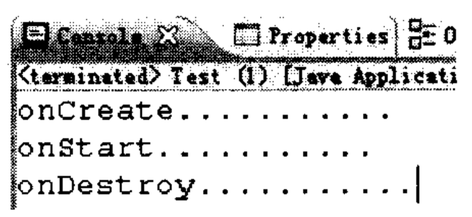


图 5.1　测试方法回调

5.2 Activity 简介

5.2.1　Activity 的创建

Activity 提供了和用户交互的可视化界面。创建一个 Activity 一般是继承 Activity（当然也可以继承 ListActivity、MapActivity 等），覆盖 Activity 的 onCreate()方法，在该方法中调用 setContentView()方法展示要显示的视图，调用 findViewById()方法实例化组件。注意 Activity 只有在清单文件中声明才能使用。

下面的代码演示了在 MainActivity 中添加 LinearLayout 布局视图，在该视图中添加一个按钮和一个文本框，覆盖 Activity 的 onCreate()方法，在该方法中调用 setContentView()方法展示要显示的视图，并调用 findViewById()方法实例化组件。

Activity 代码如下所示：

```
package com.amaker.test;

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

// 继承 Activity
public class MainActivity extends Activity {
    // 声明要使用的组件
    private TextView myTextView;
    private Button myButton;
    // 覆盖 onCreate 方法
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前视图
        setContentView(R.layout.main);
        // 通过 findViewById()  方法实例化组件
        myTextView = (TextView) findViewById(R.id.TextView01);
        myButton = (Button) findViewById(R.id.Button01);
    }
}
```

布局文件代码如下所示：

```
<?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"
    >

    <TextView android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="简单的 Activity"></TextView>

    <Button android:text="@+id/Button01"
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

</LinearLayout>
```

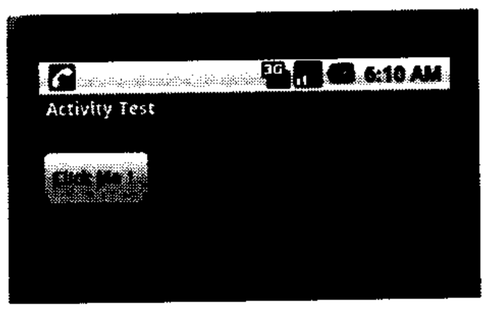


图 5.2　创建 Activity

清单文件代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.test"
     android:versionCode="1"
     android:versionName="1.0">
```

```
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MainActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

5.2.2　启动另一个 Activity

像我们在做 Web 开发一样，经常会从一个页面跳转到另一个页面。在 Android 中我们也经常会从一个 Activity 跳转到另一个 Activity 做出一些处理。从一个 Activity 启动另一个 Activity 可以使用 startActivity 方法或者 startActivityForResult()方法（能够返回结果）。这两个方法要传递的参数是 Android 中的另外一个非常重要的组件 Intent，Intent 是相同或不同组件的信使，有关 Intent 的内容请参考第 6 章的内容。

下面我们来演示一个实例，在这个实例中有两个 Activity：FirstActivity 和 SecondActivity。每个 Activity 中放置一个按钮，FirstActivity 中的按钮响应事件跳转到 SecondActivity，SecondActivity 中的按钮响应事件跳转到 FirstActivity。

FirstActivity 代码如下所示：

```
package com.amaker.test;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class FirstActivity extends Activity {
    //  声明 Button 对象实例
   private Button b1;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // 设置当前 Activity 界面布局
       setContentView(R.layout. first);
       // 通过 findViewById 方法获得 Button 实例
       b1 = (Button) findViewById(R.id.Button01);
       // 响应按键事件
       b1.setOnClickListener(new OnClickListener() {
           @Override
           public void onClick(View v) {
               // 显示方式声明 Intent,直接启动 SecondActivity
```

```
                Intent intent = new Intent(FirstActivity.this,SecondActivity.class);
                 // 启动 Activity
                startActivity(intent);
            }
        });
    }
}}
```

SecondActivity 代码如下所示：

```
package com.amaker.test;

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

public class SecondActivity extends Activity {
    // 声明 Button
    private Button b2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.second);
        // 通过 findViewById 方法获得 Button 对象
        b2 = (Button) findViewById(R.id.Button02);
        // 响应按键事件
        b2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                 // 显示方式声明 Intent,直接启动 SecondActivity
                Intent intent = new Intent(SecondActivity.this,FirstActivity.class);
                startActivity(intent);
            }
        });
    }
}}
```

First.xml 布局文件代码如下所示：

```
<?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"
    >

    <Button android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:text="Second Activity"></Button>

</LinearLayout>
```

Second.xml 布局文件代码如下所示：

```
<?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">

    <Button android:id="@+id/Button02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="返回"></Button>
</LinearLayout>
```

清单文件代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.amaker.test"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".FirstActivity"
                  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="SecondActivity"/>
    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

程序运行结果如图 5.3 所示。

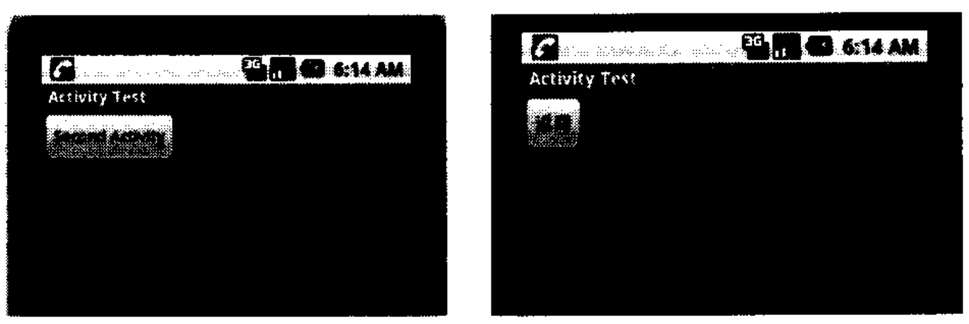


图 5.3　启动另一个 Activity

5.2.3　Activity 之间传递数据

在 Web 开发中我们经常把数据放在某个 Scope 中（如 request、session）来实现数据共

享。在 Android 系统中不同的 Activity 之间又是如何传递数据的呢？这里要用到另外一个对象 Bundle，我们将要传递的信息封装到该对象里面，并通过 Intent 对象传递到另一个 Intent 中。

在 Android 系统中提供了一个联系人管理应用程序，本实例是在这个应用程序基础上实现的，在这个实例的第一个 Activity 中要求用户输入用户名称，通过用户名称查询用户联系方式信息并在第二个 Activity 中显示，这里需要将用户名称传递给第二个 Activity。

MainActivity 代码如下所示：

```
package com.amaker.test;

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

public class MainActivity extends Activity {
    // 声明 Button 对象
   private Button b1;
    // 声明 EditText 实例
   private EditText myEditText;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       // 设置当前 Activity 界面布局
       setContentView(R.layout.main);
       // 通过 findViewById 方法获得 Button 实例
        b1 = (Button)findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 EditText 实例
        myEditText = (EditText)findViewById(R.id.EditText01);
        // 设置单击监听器
        b1.setOnClickListener(new OnClickListener() {
          public void onClick(View v) {
               // 获得姓名字符串
                String name = myEditText.getText().toString();
               // 实例化 Bundle 对象
                Bundle data = new Bundle();
               // 将姓名信息添加到 Bundle
                data.putString("name", name);
              // 实例化 Intent
                Intent intent = new Intent(MainActivity.this,ResultActivity.class);
              // 为 Intent 添加 Bundle
                intent.putExtras(data);
              // 启动 Activity
                startActivity(intent);
          }
       });
```

```
    }
}
```

ResultActivity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Contacts.People;
import android.widget.TextView;

public class ResultActivity extends Activity {
    // 声明 TextView
    private TextView tv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.result);
        // 通过 findViewById 方法获得 TextView 实例
         tv = (TextView)findViewById(R.id.TextView02);
        // 获得 Intent
        Intent intent = getIntent();
        // 从 Intent 中获得 Bundle 对象
        Bundle b = intent.getExtras();
        // 从 Bundle 中获得 name
        String name = b.getString("name");
         // 创建查询列数组
        String[] projection = new String[] { People._ID,
                People.NAME, People.NUMBER };
        // 查询 Uri 对象
        Uri contacts = People.CONTENT_URI;
        // 查询参数
        String[] args = {name};
        // 根据姓名查询联系人
        Cursor managedCursor = managedQuery(contacts, projection,
                "name=?",
                args,
                People.NAME + " ASC");
        // 遍历游标
        if(managedCursor.moveToFirst()){
            // 获得姓名
            String name1 = managedCursor.getString(1);
            // 获得电话号码
            String number = managedCursor.getString(2);
            // 设置 TextView 的文本内容为姓名和电话号码
            tv.setText(name1+":"+number);
        }
```

```
    }
}
```

布局文件 main.xml 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 xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <TextView
      android:text="请输入用户名称："
      android:id="@+id/TextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></TextView>

      <EditText
      android:text=""
      android:id="@+id/EditText01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"></EditText>
   </LinearLayout>

   <Button
   android:text="查询"
   android:id="@+id/Button01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_gravity="right"></Button>

</LinearLayout>
```

布局文件 result.xml 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:text=""
   android:id="@+id/TextView02"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TextView>

</LinearLayout>
```

清单文件 AndroidManifest.xml 代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.test"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity android:name="ResultActivity"/>

   </application>
   <uses-sdk android:minSdkVersion="3" />

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

程序运行结果如图 5.4 所示。

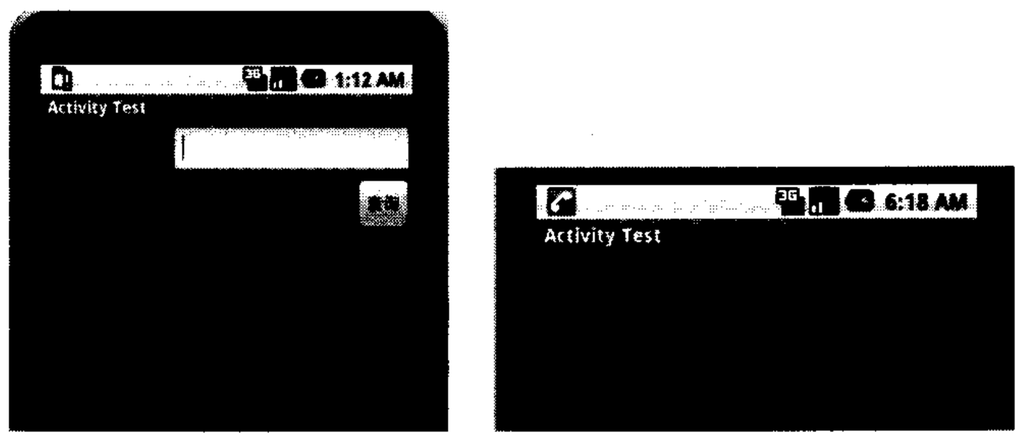


图 5.4　Activity 之间传递数据

5.2.4　启动另一个 Activity 并返回结果

在 5.2.2 节里，我们讲述了如何启动另一个 Activity，有的时候我们需要启动另一个 Activity，并返回一个结果。这里我们使用另外一个方法 startActivityForResult()。

例如，在注册一个系统会员的时候经常需要多个步骤，我们经常使用“上一步”、“下一步”来完成，那么当用户在返回“上一步”时应如何保存信息呢？

MainActivity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
    // 声明 EditText 对象
    private EditText username,password;
    // 声明 Button 对象
    private Button b1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button)findViewById(R.id.Button01);
        // 为 Button 添加单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 通过 findViewById 方法获得 EditText 实例
                username = (EditText)findViewById(R.id.username);
                // 通过 findViewById 方法获得 EditText 实例
                password = (EditText)findViewById(R.id.password);
                // 获得用户名称
                String str_username = username.getText().toString();
                // 获得密码
                String str_password = password.getText().toString();
                // 实例化 Bundle
                Bundle b = new Bundle();
                // 为 Bundle 添加用户名称、密码
                b.putString("username", str_username);
                b.putString("password", str_password);
                // 实例化 Intent
                Intent intent = new Intent(MainActivity.this,NextActivity.class);
                // 将 Bundle 添加到 intent
                intent.putExtras(b);
                // 启动 Activity
                startActivityForResult(intent, 0);
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            // 从 Intent 中获得 Bundle
            Bundle b = data.getExtras();
            // 从 Bundle 中获得用户名称、密码
```

```
            String str_username = b.getString("username");
            String str_password = b.getString("password");
            // 将用户名称和密码赋值给 EditText
            username.setText(str_username);
            password.setText(str_password);
    }
}
```

NextActivity 代码如下所示：

```
package com.amaker.test;

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

public class NextActivity extends Activity {
    // 声明 Button 对象
    private Button b2;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.next);
        // 通过 findViewById 方法获得 Button 对象实例
        b2 = (Button)findViewById(R.id.Button02);
        // 为 Button 添加事件监听器
        b2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 获得 Intent
                Intent intent = getIntent();
                // 设置返回结果
                NextActivity.this.setResult(0, intent);
                // 结束当前 Activity
                NextActivity.this.finish();
            }
        });
    }
}
```

布局文件 main.xml 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:text="用户名称："
android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

<EditText
android:text=""
android:id="@+id/username"
android:layout_width="fill_parent"
android:layout_height="wrap_content"></EditText>

<TextView
android:text="用户密码："
android:id="@+id/TextView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"

></TextView>

<EditText
android:text=""
android:id="@+id/password"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:password="true"
></EditText>

<Button
android:text="下一步"
android:id="@+id/Button01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></Button>

</LinearLayout>
```

布局文件 next.xml 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:text="Email："
android:id="@+id/email"
android:layout_width="wrap_content"
android:layout_height="wrap_content"></TextView>

<EditText
android:text=""
android:id="@+id/EditText01"
```

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

    <TextView
    android:text="Mobile: "
    android:id="@+id/mobile"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></TextView>

    <EditText
    android:text=""
    android:id="@+id/EditText02"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"></EditText>

    <Button
    android:text="上一步"
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

    </LinearLayout>
```

清单文件 AndroidManifest.xml 代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.test"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name="NextActivity"/>
   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

程序运行结果如图 5.5 所示。

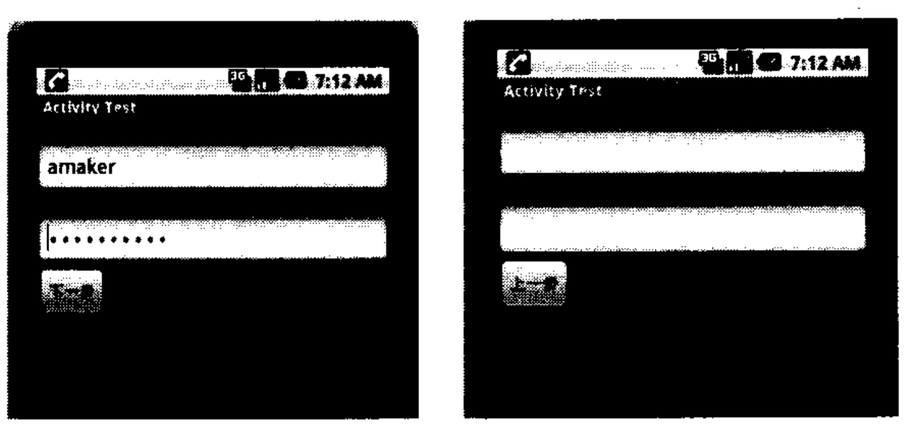


图 5.5　启动另一个 Activity 并返回结果

5.3 Activity 的生命周期

在 Android 系统中 Activity 作为 Activity 栈（Stack）被管理，当前活动的 Activity 处于栈顶，之前的非活动 Activity 被压入下面成为非活动 Activity，等待是否可能被恢复为活动状态。

在 Activity 的生命周期中有四个重要的状态，如表 5.1 所示：

表 5.1　Activity的生命周期

状　态	状 态 描 述
活动	在屏幕的前面（在栈顶），有焦点并可见
暂停	失去了焦点，但是它仍然可见
停止	失去焦点，不可见
销毁	被系统或进程结束

Activity 各种状态及方法调用时机如图 5.6 所示。

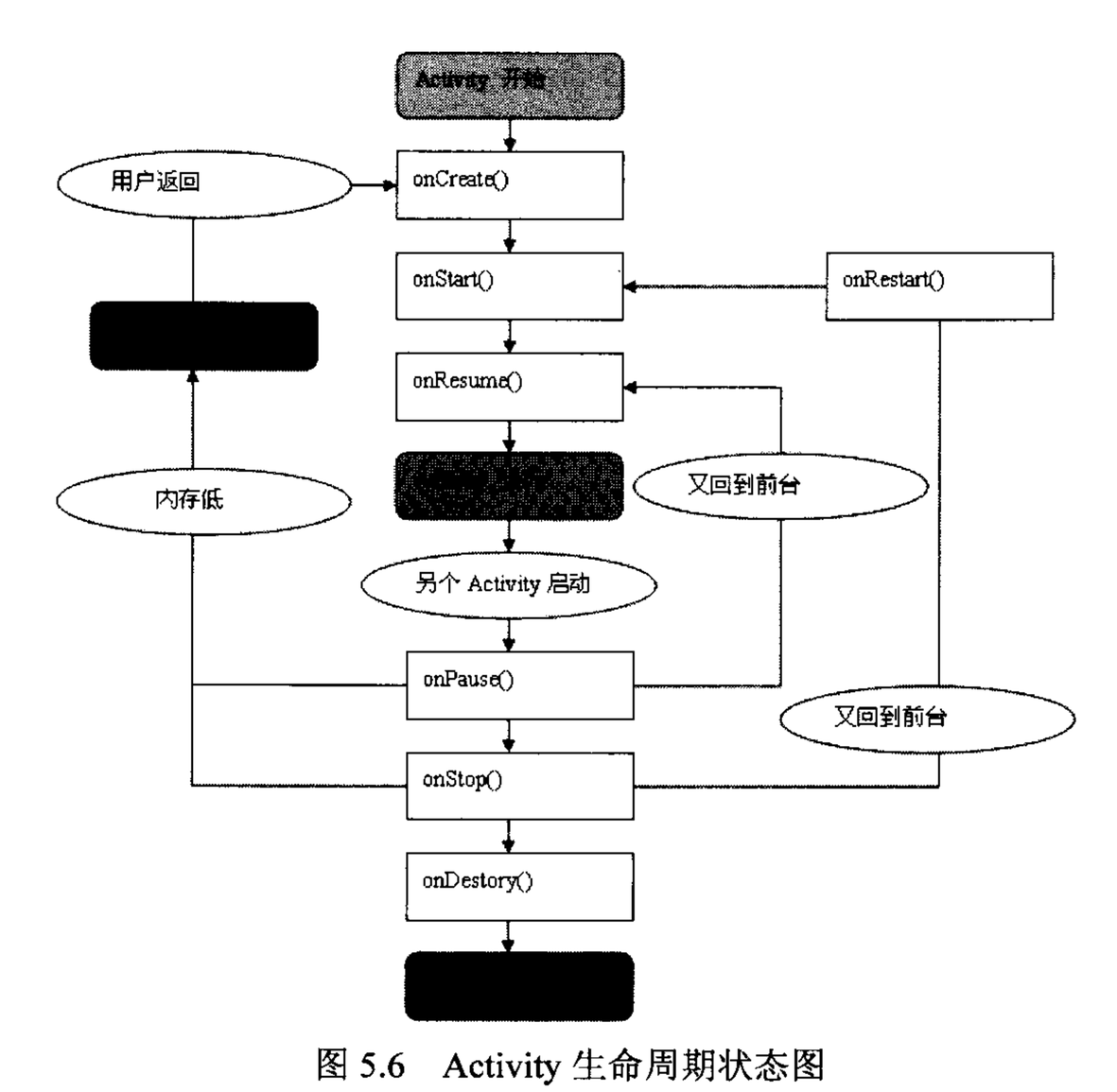


图 5.6　Activity 生命周期状态图

从图 5.6 中我们可以将 Activity 的生命周期分为以下三个过程：

- 整个完整生命周期：从 onCreate()方法开始到 onDestory()方法结束。
- 可见生命周期：从 onStart 开始到 onStop 结束。
- 前台可见生命周期：从 onResume()方法开始到 onPause()结束。

如下是整个 Activity 生命周期中的所有方法：

```
public class Activity extends ApplicationContext {
   // 创建时调用
    protected void onCreate(Bundle savedInstanceState);
   // 启动时调用
    protected void onStart();
    // 重新启动时调用
    protected void onRestart();
   // 恢复时调用
    protected void onResume();
   // 暂停时调用
    protected void onPause();
   // 停止时调用
    protected void onStop();
   // 销毁时调用
    protected void onDestroy();
}
```

可以根据我们程序的需要来覆盖相应的方法。当然，最常用的方法是 onCreate()和 onPause()。onCreate()用来初始化组件并设置 Activity 视图等。onPause()用来持久保存数据。例如，当我们正在编辑短信，突然来了一个电话，这时候就可以在这个方法中将短信内容持久地保存起来。

下面通过一个实例来测试一下 Activity 的生命周期中各个方法的调用情况，在这里我们把 Activity 的所有方法都覆盖，并通过日志的方式来观察运行结果。

我们在界面中放置一个按钮，当单击按钮时结束 Activity，并观察日志输出。

当程序启动时日志输出结果如图 5.7 所示。

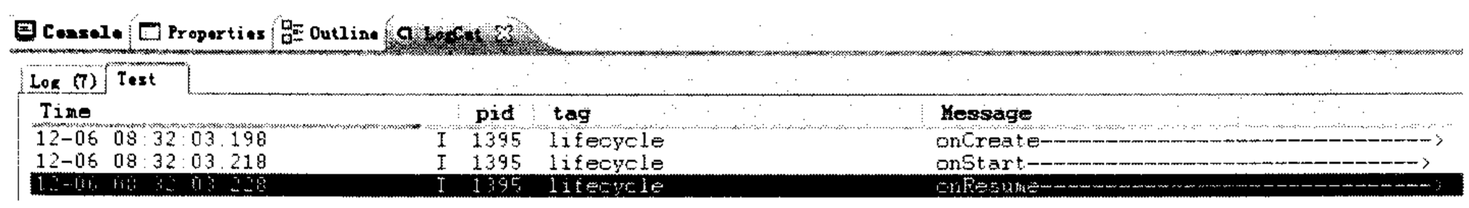


图 5.7　Activity 生命周期

当单击结束 Activity 按钮时日志输出结果如图 5.8 所示。

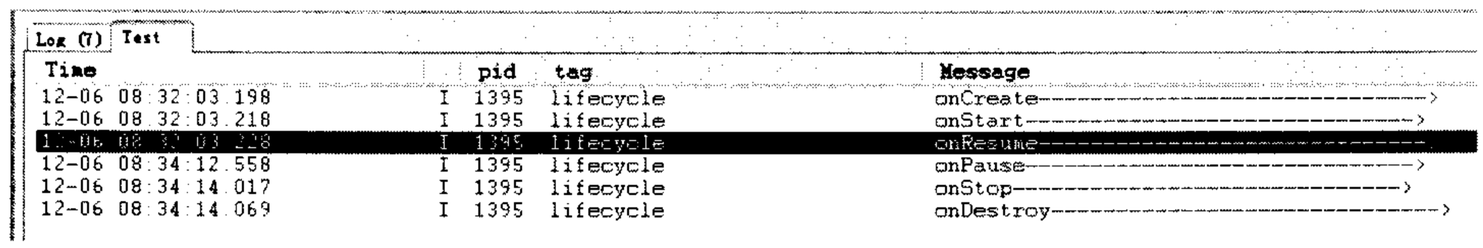


图 5.8　Activity 生命周期

MainActivity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
```

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
    // 声明按钮对象
    private Button b1;
    // 定义日志标签
    private static final String TAG="lifecycle";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 输出日志
        Log.i(TAG, "onCreate------------------------------->");
        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button)findViewById(R.id.Button01);
        // 为 Button 设置单击监听器
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 结束当前 Activity
                MainActivity.this.finish();
            }
        });
    }

    @Override// 启动
    protected void onStart() {
    super.onStart();
    // 输出日志
    Log.i(TAG, "onStart------------------------------->");
    }

    @Override// 重启
    protected void onRestart() {
    super.onRestart();
    // 输出日志
    Log.i(TAG, "onRestart------------------------------->");
    }

    @Override// 恢复
    protected void onResume() {
    super.onResume();
    // 输出日志
    Log.i(TAG, "onResume------------------------------->");
    }

    @Override// 暂停
```

```
    protected void onPause() {
    super.onPause();
    // 输出日志
    Log.i(TAG, "onPause------------------------------->");
    }

    @Override// 停止
    protected void onStop() {
    super.onStop();
    // 输出日志
    Log.i(TAG, "onStop------------------------------->");
    }

    @Override// 销毁
    protected void onDestroy() {
    super.onDestroy();
    // 输出日志
    Log.i(TAG, "onDestroy------------------------------->");
    }

}
```

布局文件代码如下所示：

```
<?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"
    >
<TextView
android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="测试 Activity 的生命周期"></TextView>

<Button android:id="@+id/Button01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="结束 Activity"></Button>

</LinearLayout>
```

程序运行结果如图 5.9 所示。

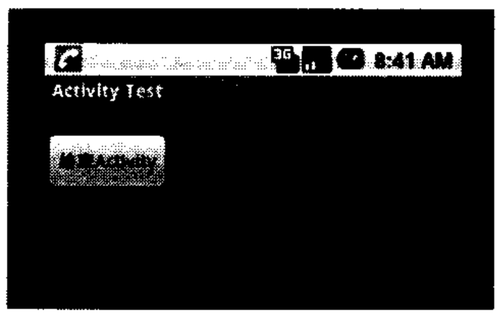
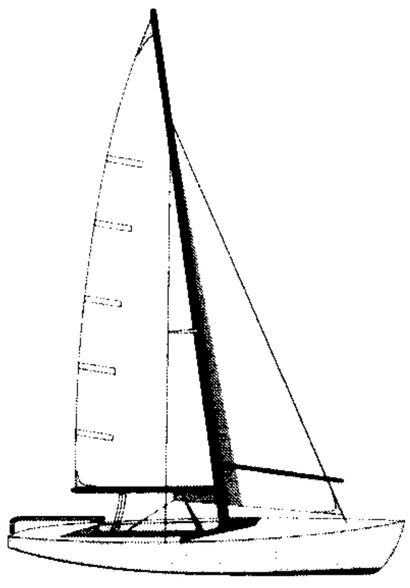


图 5.9　Activity 生命周期



第 6 章　Android 组件之间的信使 Intent

Intent 的中文意思是“意图，意向”，可以把它理解为不同组件之间通信的“媒介”或者“信使”。也就是“把我的意思/意图告诉它”。举个例子来说吧，一位姑娘想找个对象，然后她找了一个媒婆，把她的要求（四有新人，即有型、有钱、有车、有房）告诉了这位媒婆，然后媒婆按照她的要求去寻找目标对象了。在这里的“姑娘”是源组件（例如，一个 Activity），“媒婆”是 Intent，“目标对象”是目标组件。这样源组件通过 Intent 找到了目标组件。

目标组件要通过 Intent 来声明自己的条件，一般通过组件中的<intent-filter>元素来实现。

Intent 到底都能干什么呢？Intent 可以启动一个 Activity，也可以启动一个 Service，还可以发起一个广播（Broadcast）。具体方法如表 6.1 所示。

表 6.1　Intent 启动不同组件的方法

组 件 名 称	方 法 名 称
Activity	startActivity() startActivityForResult()
Service	startService() bindService()
Broadcasts	sendBroadcast() sendOrderedBroadcast() sendStickyBroadcast()

Intent 由以下几个部分组成：动作（Action）、数据（Data）、分类（Category）、类型（Type）、组件（Component）和扩展信息（Extra）。通过这些可以启动其他组件并携带信息。

Intent 在寻找目标组件时有两种方法：第一，通过组件名称直接指定；第二，通过 Intent Filter 过滤指定。还是上面那个姑娘找对象的例子，可以有两种方法：第一，姑娘告诉媒婆

我就找张三了；第二，姑娘提出条件（如要四有新人），那么就要在符合条件的人中过滤筛选。

6.1 Intent 对象及其属性

Intent 是对它要完成操作的一种抽象描述，我们可以使用 Intent 来启动一个 Activity、发起一个 Broadcast 和启动或绑定一个 Service。Intent 使应用程序代码在运行时动态绑定成为可能，这也降低了不同代码之间的耦合性。Intent 最常使用的方法是用来启动一个 Activity。Intent 封装了它要执行动作的属性，这些属性最常见的是 Action 和 Data。

6.1.1 Intent 的 ComponentName 属性

在后续的内容中我们将介绍 Intent 的查找组件策略，其中的一种方法是显式查找，就是直接通过组件名称（Component name）来查找。Intent 的组件名称对象由 ComponentName 类来封装，组件名称包含包名称和类名称，被声明在 AndroidManifest.xml 文件中。

组件名称通过 setComponent()、setClass()、setClassName()设置，通过 getComponent()获得。

下面我们通过一个实例来演示 Intent 组件名称属性的使用，该实例是通过一个组件名称来直接启动一个 Activity 的。实例步骤说明如下。

① 创建一个名为“Chapter06_Intent_ComponentName”的项目，在“com.amaker.ch06.app”包中创建一个 MainActivity 类。

```
import android.os.Bundle;
package com.amaker.ch06.app;

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

/**
 * @author 郭宏志
 * 测试 Intent 的 ComponentName 属性
 */
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置视图布局
        setContentView(R.layout.main);
```

```
    }
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在其中添加一个 Button 组件。

```
<?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:text="测试 Intent 的组件名称属性"
        android:id="@+id/myButton01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 在 MainActivity 类的顶部声明 Button 组件，在 onCreate()方法中实例化，并为其添加单击事件监听器。

```
    private Button btn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置视图布局
        setContentView(R.layout.main);
        // 实例化 Button
        btn = (Button)findViewById(R.id.myButton01);
        // 添加单击监听器
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 实例化组件名称
                ComponentName cn =
                    new  ComponentName(MainActivity.this,  "com.amaker.ch06.app1.
MyActivity");
                // 实例化 Intent
                Intent intent = new Intent();
                // 为 Intent 设置组件名称属性
                intent.setComponent(cn);
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
```

单击方法里创建了组件名称对象，指向了一个 Activity，实例化 Intent 并为其设置组件名称属性，启动这个 Activity。

④ 在该工程中创建另外一个包“com.amaker.ch06.app1”，并在其中创建一个

MyActivity 类。

```
package com.amaker.ch06.app1;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

import com.amaker.ch06.app.R;

/**
 * @author 郭宏志
 * 测试 Intent 的 ComponentName 属性
 */
public class MyActivity extends Activity {
    // 声明 TextView
    private TextView tv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // 设置视图布局
        super.onCreate(savedInstanceState);
    }
}
```

⑤ 在该工程的 res\layout\目录下创建一个布局文件 my_layout.xml，在其中添加一个 TextView 组件。

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

    <TextView
        android:text="@+id/TextView01"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

</LinearLayout>
```

⑥ 在 MyActivity 类的顶部声明 TextView，在 onCreate()方法中实例化它，获得 Intent 从而获得组件名称对象，从组件名称对象中获得包名称和类名称，设置在 TextView 中显示。

```
    // 声明 TextView
    private TextView tv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
    // 设置视图布局
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_layout);
        // 获得 Intent
        Intent intent = this.getIntent();
        // 获得组件名称对象
        ComponentName cn = intent.getComponent();
        // 获得包名称
        String packageName = cn.getPackageName();
        // 获得类名称
        String className = cn.getClassName();
        // 实例化 TextView
        tv = (TextView)findViewById(R.id.TextView01);
        // 显示
        tv.setText("组件包名称: "+packageName+"\n"+"组件类名称: "+className);
    }
```

⑦ 在 AndroidManifest.xml 文件中声明 MyActivity。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch06.app"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name="com.amaker.ch06.app1.MyActivity"/>
   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

程序运行结果如图 6.1 所示。

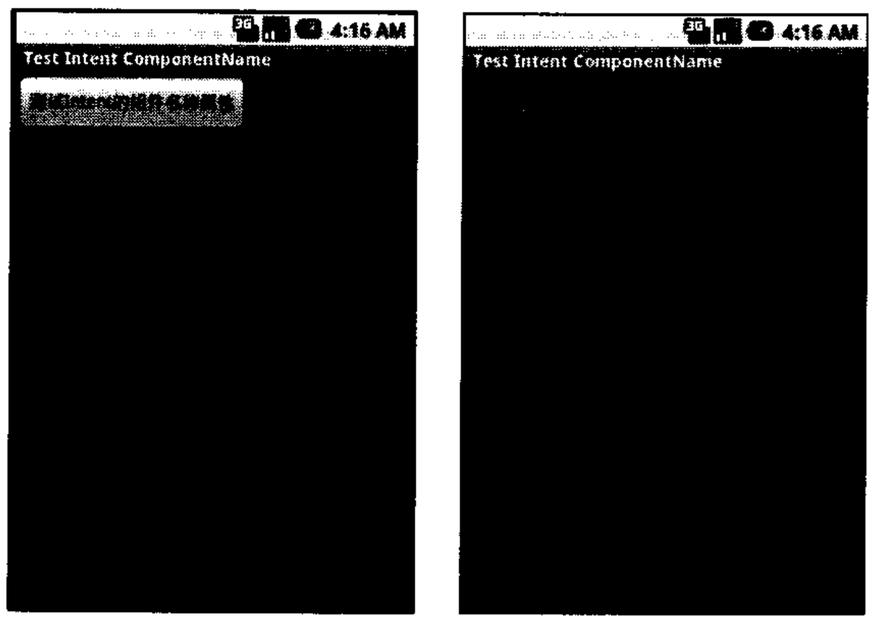


图 6.1　测试 Intent 组件名称属性

6.1.2　Intent 的 Action 属性

Action 是指 Intent 要完成的动作，是一个字符串常量。在 Intent 类里面定义了大量的 Action 常量属性，例如，ACTION_CALL（打电话）、ACTION_EDIT（编辑数据）、ACTION_BATTERY_LOW（电量低广播 Action）等。我们也可以自己定义 Action 来使用。

我们可以使用 setAction()来设置 IntentAction 属性，使用 getAction 来获得 Intent 的 Action 属性。

1．自定义 Action 属性

我们可以为 Intent 定义一个 Action 属性来访问，Action 属性是一个字符串，我们在程序中定义，并在要访问组件（例如，Activity）的 IntentFilter 中声明就可以使用了。下面的实例将演示如何自定义一个 Intent Action 属性。

① 创建一个工程“Chapter06_Intent_TestAction”，在包“com.amaker.ch06.app”中创建一个 MainActivity 类。

```
package com.amaker.ch06.app;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
/**
 *
 * @author 郭宏志
 * 测试 Intent Action 属性
 */
public class MainActivity extends Activity {
    // 定义 Action 属性常量
    public static final String MY_ACTION="com.amaker.ch06.app.MY_ACTION";
    // 声明 Button
    private Button btn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置布局视图
        setContentView(R.layout.main);
    }
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在其中添加一个 Button 组件。

```
<?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:text="测试 Intent 的 Action 属性"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 在 MainActivity 类的顶部声明 Action 属性字符串常量和 Button 实例，并在 onCreate 方法中实例化它。为 Button 添加单击监听器，在 onClick()方法中创建 Intent，设置 Action 属性，并启动 Activity。

```
// 定义 Action 属性常量
public static final String MY_ACTION="com.amaker.ch06.app.MY_ACTION";
// 声明 Button
private Button btn;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 设置布局视图
    setContentView(R.layout.main);
    // 实例化 Button
    btn = (Button)findViewById(R.id.Button01);
    btn.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 实例化 Intent
            Intent intent = new Intent();
            // 为 Intent 设置 Action 属性
            intent.setAction(MY_ACTION);
            // 启动 Activity
            startActivity(intent);
        }
    });
}
```

④ 在工程的 res\layout\目录下创建一个 my_layout.xml 布局文件，在其中添加一个 TextView 视图组件。

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

    <TextView
```

```
        android:text="@+id/TextView01"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>

</LinearLayout>
```

⑤ 在“com.amaker.ch06.app”包中创建一个 MyActivity 类，在其顶部声明 TextView 组件，在 onCreate()方法中实例化它，获得 Intent 实例，从而获得 action 属性并显示在 TextView 中。

```
package com.amaker.ch06.app;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
/**
 * @author 郭宏志
 * 测试Intent Action 属性
 */
public class MyActivity extends Activity {
    // 声明TextView
    private TextView tv;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置视图布局
        setContentView(R.layout.my_layout);
        // 获得Intent 对象
        Intent intent = getIntent();
        // 获得Action
        String action = intent.getAction();
        // 获得TextView
        tv = (TextView)findViewById(R.id.TextView01);
        // 设置内容
        tv.setText(action);
    }
}
```

⑥ 在 AndroidManifest.xml 配置文件中添加一个 Activity 声明，在 IntentFilter 元素中指定 Action 属性。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch06.app"
     android:versionCode="1"
     android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">

        <activity android:name=".MainActivity"
```

```
            android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name="MyActivity">
        <intent-filter>
            <action android:name="com.amaker.ch06.app.MY_ACTION" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>

    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

程序运行结果如图 6.2 所示。

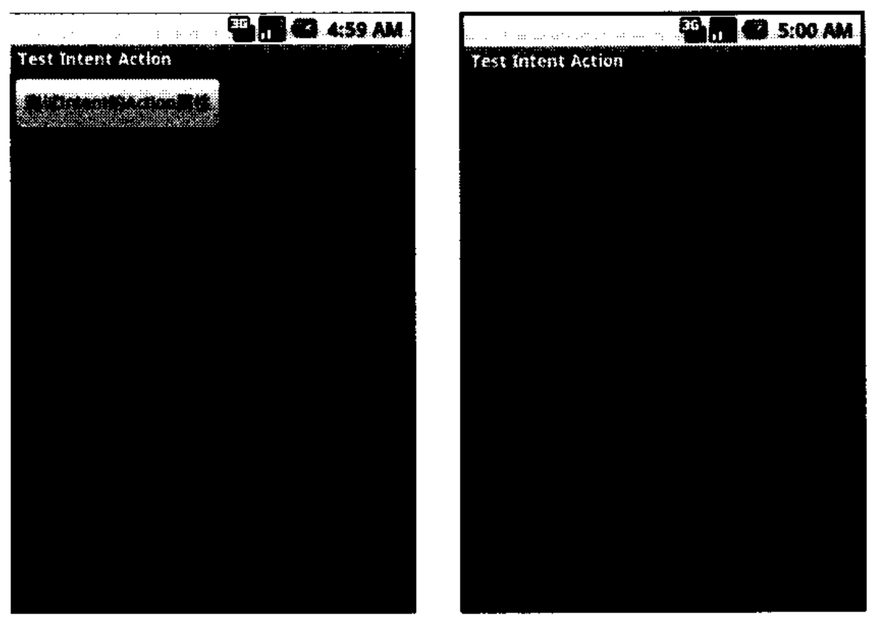


图 6.2　测试 Intent Action 属性

2. 访问系统 Action 属性

Android Intent 类中定义了很多 Action 属性常量，这些有标准 Activity 的和 Broadcast Receiver 的。上面我们讲述了如何访问自定义 Action 属性。下面我们将讲述如何访问系统 Action 属性。

例如，我们实现了一个发送短信的程序，在这个程序当中，我们需要选择对方的电话号码，这时候我们可以调用系统电话本来查找电话号码。我们可以使用 Intent.ACTION_GET_CONTENT 常量调用系统 Activity 来实现该功能。实例步骤说明如下。

① 创建一个工程“Chapter06_Intent_Action2”，在“com.amaker.ch06.app”包中创建一个 MainActivity。

```
package com.amaker.ch06.app;
```

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

/**
 *
 * @author 郭宏志
 * 测试 Intent Action 属性
 */
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前布局视图
        setContentView(R.layout.main);
      }
}
```

② 在工程 res\layout\目录中创建一个 main.xml 布局文件，在其中添加一个 Button 组件。

```
<?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:text="测试 Intent 的 Action 属性"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 在 MainActivity 顶部声明 Button 组件，在 onCreate()方法中实例化，设置当前布局，为 Button 添加单击监听器，在 onClick 方法中创建 Intent 对象，设置 Action 属性为“Intent.ACTION_GET_CONTENT”，设置 Type 属性为“vnd.android.cursor.item/phone”。

```
    // 声明 Button
    private Button btn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前布局视图
        setContentView(R.layout.main);
        // 实例化 Button
        btn = (Button)findViewById(R.id.Button01);
        btn.setOnClickListener(new OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                // 创建 Intent
                Intent intent = new Intent();
                // 设置 Intent Action 属性
                intent.setAction(Intent.ACTION_GET_CONTENT);
                // 设置 Intent Type 属性
                intent.setType("vnd.android.cursor.item/phone");
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
```

程序运行结果如图 6.3 所示。

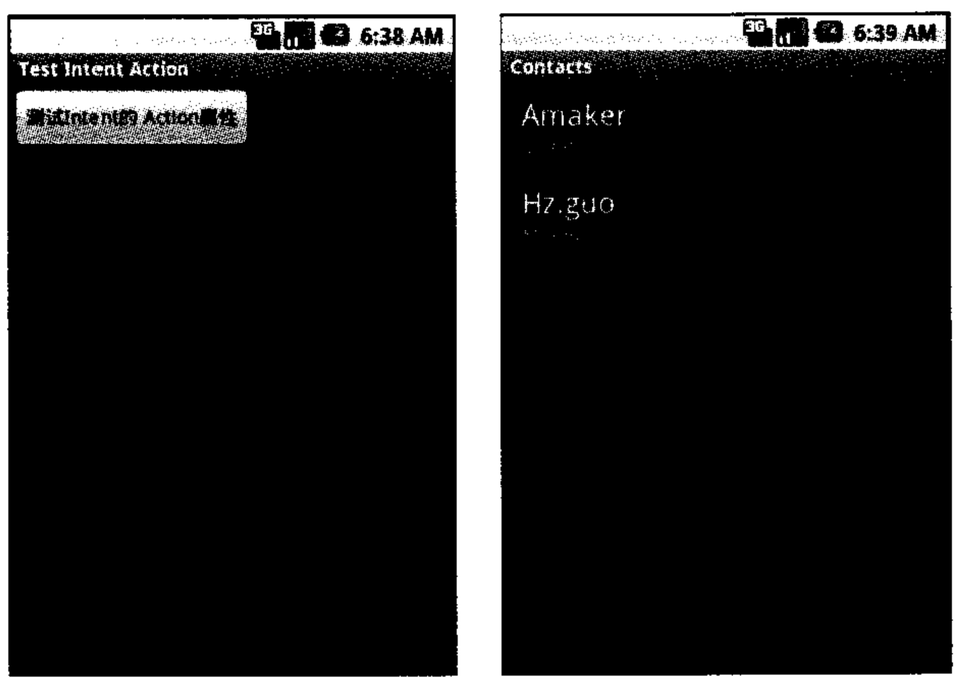


图 6.3　测试 Intent Action 属性

6.1.3　Intent 的 Data 属性

Intent 的 Data 属性是执行动作的 URI 和 MIME 类型，不同的 Action 有不同的 Data 数据指定。例如，ACTION_EDIT Action 应该和要编辑的文档 URI Data 匹配，ACTION_VIEW 应用和要显示的 URI 匹配。表 6.2 列出了一些常见的 Action 和 Data 的匹配应用。

表 6.2　Intent 的 Action 和 Data 属性匹配

Action 属性	Data 属性	说　明
ACTION_VIEW	content://contacts/people/1	显示_id 为 1 的联系人信息
ACTION_DIAL	content://contacts/people/1	将_id 为 1 的联系人电话号码显示在拨号界面中
ACTION_VIEW	tel:123	显示电话为 123 的联系人信息
ACTION_VIEW	http://www.google.com	在浏览器中浏览该网址
ACTION_VIEW	file:///sdcard/mymusic.mp3	播放 MP3
ACTION_VIEW	geo:39.3256,116.2312	显示地图

6.1.4　Intent 的 Category 属性

Intent 中的 Category 属性是一个执行 Action 的附加信息。例如，CATEGORY_LAUNCHER 意味着在加载程序时，Activity 出现在最上面。还有 CATEGORY_HOME 则表示回到 Home 界面。

当我们创建一个应用程序的时候，程序第一个加载的 Activity 中有如下的代码：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.test"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

上述代码表明当前的 Activity 是加载程序时第一个出现的界面。

下面的实例演示了如何回到 Home 界面。

```
package com.amaker.test;

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

/**
 * @author 郭宏志
 * 测试Intent 的 Category属性
 */
public class MainActivity extends Activity {
   // 声明 Button
   private Button b1;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置当前布局
      setContentView(R.layout.main);
      // 实例化 Button
```

```
        b1 = (Button)findViewById(R.id.Button01);
        // 为 Button 添加监听器
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 实例化 Intent
                Intent i = new Intent();
                // 添加 Action 属性
                i.setAction(Intent.ACTION_MAIN);
                // 添加 Category 属性
                i.addCategory(Intent.CATEGORY_HOME);
                // 启动 Activity
                startActivity(i);
            }
        });
    }
}
```

当我们单击“回到 Home”按钮时，程序回到 Home 界面。这里是通过设置 Intent 的 Category 属性来实现的。程序运行结果如图 6.4 所示。

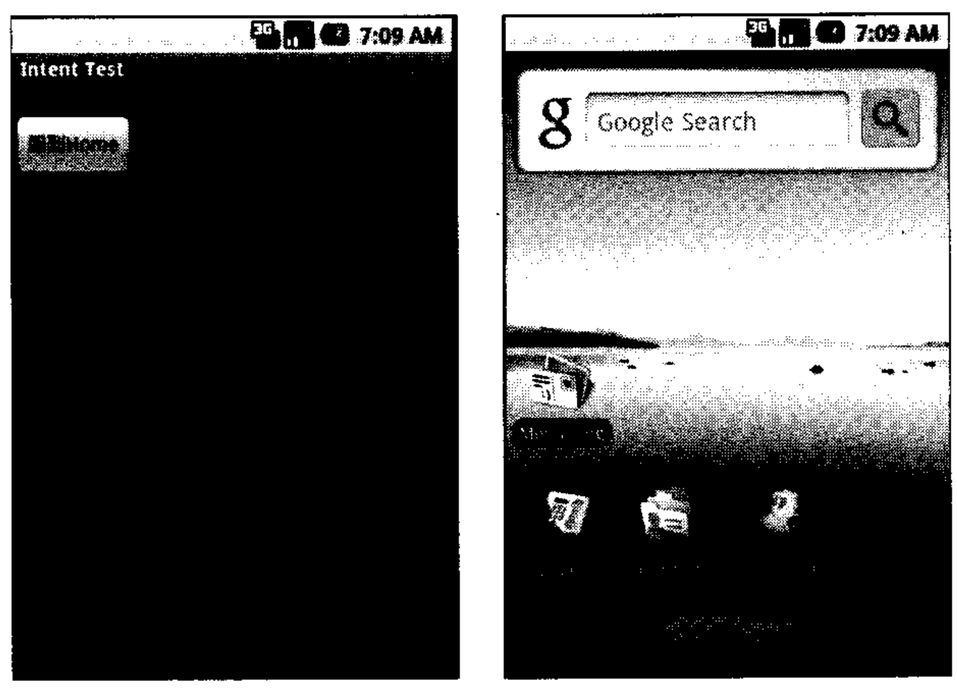


图 6.4　测试 Intent Category 属性

6.1.5　Intent 的 Extras 属性

Intent 的 Extras 属性是添加一些组件的附加信息。例如，如果我们要通过一个 Activity 来发送一个 E-mail，就可以通过 Extras 属性来添加 subject 和 body。

下面的实例在第一个 Activity 的 EditText 中输入年龄，该年龄保存在 Intent 的 Extras 属性中，当单击按钮时在第二个 Activity 中从 Intent 的属性中取出显示。

MainActivity 代码如下所示：

```
package com.amaker.test;

import android.app.Activity;
```

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
    // 声明 Button 对象
    private Button b1;
    // 声明 EditText 对象
    private EditText et;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 的界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button)findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 EditText 实例
        et = (EditText)findViewById(R.id.EditText01);
        // 为 Button 添加单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                  // 实例化 Intent
                Intent intent = new Intent();
                // 设置 Intent 的 class 属性，跳转到 ResultActivity
                intent.setClass(MainActivity.this, ResultActivity.class);
                // 为 intent 添加额外信息
                intent.putExtra("age", et.getText().toString());
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
}
```

ResultActivity 代码如下所示：

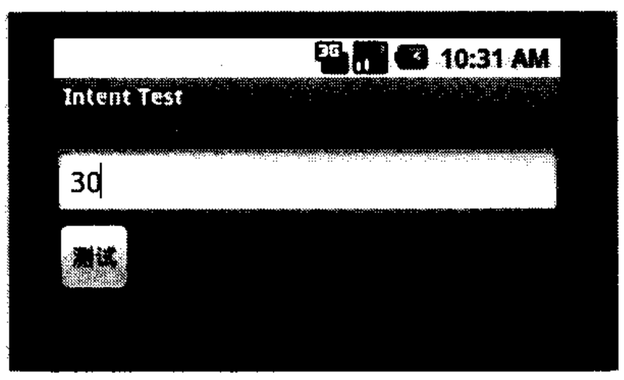
```
package com.amaker.test;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class ResultActivity extends Activity {
    // 声明 TextView 实例
    private TextView tv;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.result);
        // 获得 Intent
        Intent intent = this.getIntent();
        // 通过 findViewById 方法获得 TextView 实例
        tv = (TextView)findViewById(R.id.TextView01);
        // 从 Intent 获得额外信息，设置为 TextView 的文本
        tv.setText(intent.getStringExtra("age"));
    }
}}
```

程序运行结果如图 6.5 所示。



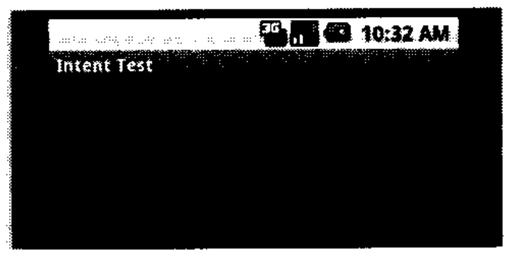


图 6.5　Intent 的 extras 属性

6.2 系统标准 Activity Action 应用

在上面的实例中我们应用了 Intent 的 Action 和 Data 来启动一个系统 Activity。Action 是一个字符串名称，用来标示组件的动作。在 Android 系统中定义了一系列 Action 常量，分为 Activity Action 和 Broadcast Action。标准 Activity Action 如表 6.3 所示。

表 6.3　Activity Action 常量

常量名称	常量值	意义
ACTION_MAIN	android.intent.action.MAIN	应用程序的入口
ACTION_VIEW	android.intent.action.VIEW	显示数据给用户
ACTION_ATTACH_DATA	android.intent.action.ATTACH_DATA	指明附加给其他地方的一些数据
ACTION_EDIT	android.intent.action.EDIT	显示可编辑的数据
ACTION_PICK	android.intent.action.PICK	选择数据
ACTION_CHOOSER	android.intent.action.CHOOSER	显示一个 Activity 选择器
ACTION_GET_CONTENT	android.intent.action. GET_CONTENT	获得内容
ACTION_DIAL	android.intent.action. GET_CONTENT	显示打电话面板
ACTION_CALL	android.intent.action. DIAL	直接打电话
ACTION_SEND	android.intent.action. SEND	直接发送短信
ACTION_SENDTO	android.intent.action. SENDTO	选择发短信
ACTION_ANSWER	android.intent.action. ANSWER	应答电话

续表

常量名称	常量值	意义
ACTION_INSERT	android.intent.action. INSERT	插入数据
ACTION_DELETE	android.intent.action. DELETE	删除数据
ACTION_RUN	android.intent.action. RUN	运行数据
ACTION_SYNC	android.intent.action. SYNC	同步数据
ACTION_PICK_ACTIVITY	android.intent.action. PICK_ACTIVITY	选择 Activity
ACTION_SEARCH	android.intent.action. SEARCH	搜索
ACTION_WEB_SEARCH	android.intent.action. WEB_SEARCH	Web 搜索
ACTION_FACTORY_TEST	android.intent.action. FACTORY_TEST	工厂测试入口点

6.2.1　和打电话相关的标准 Activity Action 应用

这里我们通过一个实例来演示和打电话相关的标准 Activity Action 的应用，我们可以通过这些标准 Activity Action 来查看联系人信息（ACTION_VIEW）、编辑联系人信息（ACTION_EDIT）、显示打电话界面（ACTION_ DIAL）、直接拨打电话（ACTION_CALL）等。

下面的实例定义一个 Activity 继承 ListActivity，在 ListView 中显示不同的应用类型。当该选项被选择时启动一个 Activity 演示效果。实例步骤说明如下。

① 创建一个工程“Chapter06_Intent_SystemAction”。在该项目的“com.amaker.ch06.app”包中创建一个 MainActivity 继承 ListActivity。

```
package com.amaker.ch06.app;

import android.app.ListActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends ListActivity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }
}
```

② 在 onCreate()方法中，为 ListView 添加列表项。

```
      // 菜单项数组
      String[] menus = { "查看电话信息", "编辑电话信息", "显示拨打电话界面","直接打电话"};
      // 将菜单项数组设置为 ListView 的列表项展示
      setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, menus));
```

```
        getListView().setTextFilterEnabled(true);
```

③ 覆盖 onListItemClick()方法，响应列表项单击事件。

```
protected void onListItemClick(ListView l, View v, int position, long id) {
    // 实例化 Intent
    Intent intent = new Intent();
    // 声明 Uri
    Uri uri ;
    // 声明数据字符串
    String data;
    switch (position) {
    // 查看_id 为 1 的用户电话信息
    case 0:
        // 字符串 Uri
        data = "content://contacts/people/1";
        // 通过 Uri 字符串获得 Uri 实例
        uri = Uri.parse(data);
        // 设置 Intent 的 Action 属性
        intent.setAction(Intent.ACTION_VIEW);
        // 设置 Intent 的 data 属性
        intent.setData(uri);
        // 启动 Activity
        startActivity(intent);
        break;
    // 编辑_id 为 1 的用户电话信息
    case 1:
        // 字符串 Uri
        data = "content://contacts/people/1";
        // 将字符串 Uri 传唤为 Uri 实例
        uri = Uri.parse(data);
        // 设置 Intent 的 Action 属性
        intent.setAction(Intent.ACTION_EDIT);
        // 设置 Intent 的 data 属性
        intent.setData(uri);
        // 启动 Activity
        startActivity(intent);
        break;
    // 显示拨打电话界面
    case 2:
        // 字符串 Uri
        data = "tel:13800138000";
        // 将字符串 Uri 传唤为 Uri 实例
        uri = Uri.parse(data);
        // 设置 Intent 的 Action 属性
        intent.setAction(Intent.ACTION_DIAL);
        // 设置 Intent 的 data 属性
        intent.setData(uri);
        // 启动 Activity
        startActivity(intent);
        break;
    // 直接打电话
```

```
        case 3:
            // 字符串 Uri
            data = "tel:13800138000";
            // 将字符串 Uri 传唤为 Uri 实例
            uri = Uri.parse(data);
            // 为 Intent 设置 Action 属性
            intent.setAction(Intent.ACTION_CALL);
            // 设置 Intent 的 data 属性
            intent.setData(uri);
            // 启动 Activity
            startActivity(intent);
            break;
        default:
            break;
        }
    }
```

程序运行结果如图 6.6 和图 6.7 所示。

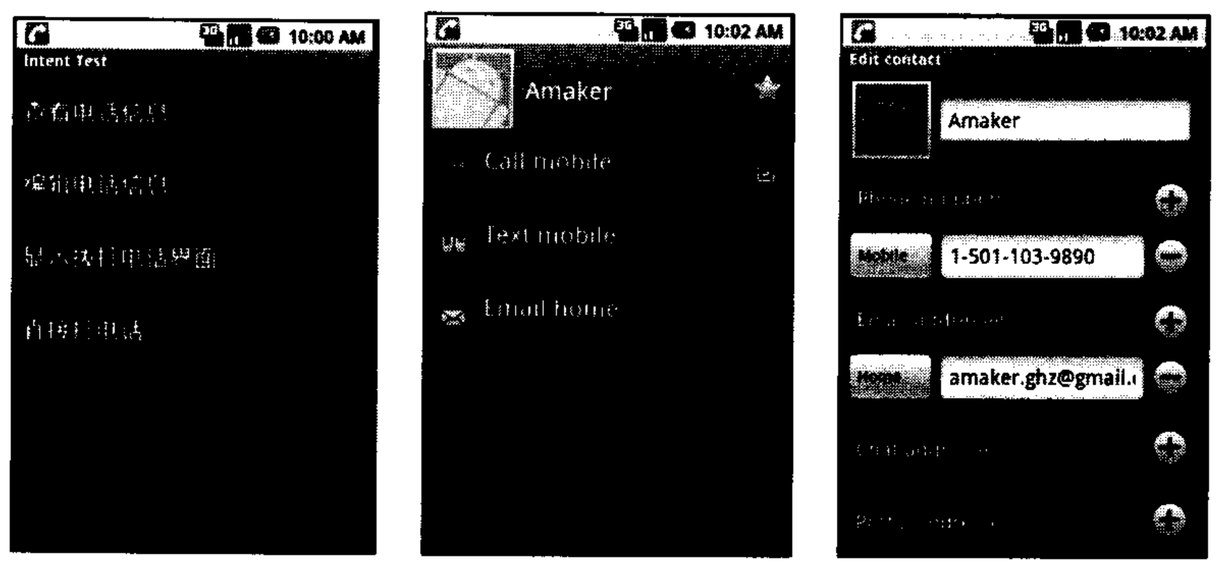


图 6.6　标准 Activity Action1

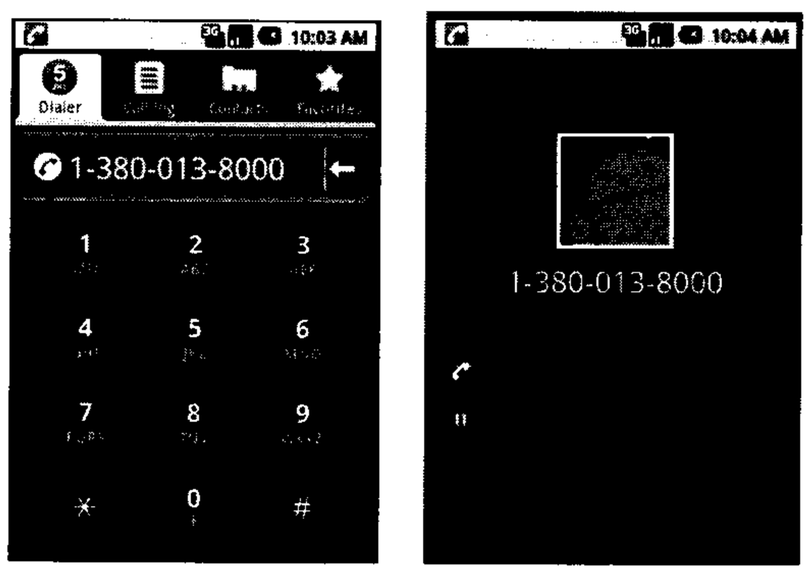


图 6.7　标准 Activity Action2

6.2.2　访问浏览器和地图

下面的实例将要演示的是如何访问浏览器和地图服务，在这里使用的标准 Activity

Action 是 ACTION_VIEW，通过构建不同的 URI 数据来调用不同的 Activity。浏览器 URI 的格式是 http://www.xxx.yyy。地图 URI 的格式是“geo:经度，纬度”。实例步骤说明如下。

- 在上面实例的基础上添加两个列表项：“访问浏览器”和“访问地图”。
- 在 onListItemClick()方法中，响应列表项单击事件。

```
        // 访问浏览器
    case 4:
        // 字符串 Uri
        data = "http://www.google.com";
        // 将字符串 Uri 传唤为 Uri 实例
        uri = Uri.parse(data);
        // 设置 Intent 的 Action 属性
        intent.setAction(Intent.ACTION_VIEW);
        // 设置 Intent 的 data 属性
        intent.setData(uri);
        // 启动 Activity
        startActivity(intent);
        break;
    // 访问地图
    case 5:
        // 字符串 Uri
        data = "geo:39.92,116.46";
        // 将字符串 Uri 传唤为 Uri 实例
        uri = Uri.parse(data);
        // 实例化 Intent
        intent = new Intent(Intent.ACTION_VIEW,uri);
        // 启动 Activity
        startActivity(intent);
        break;
```

程序运行结果如图 6.8 所示。

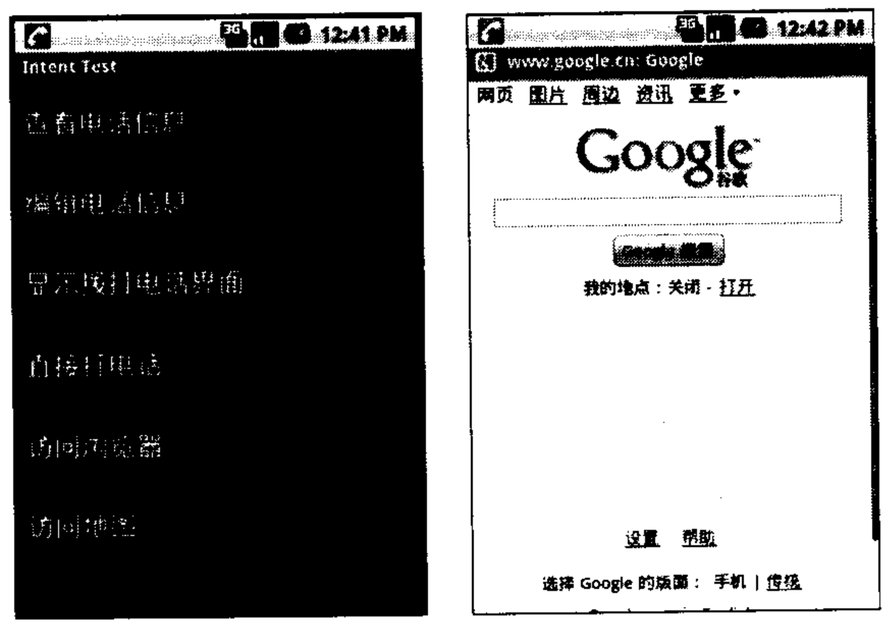


图 6.8　标准 Activity Action

6.2.3　发邮件

在移动互联网时代，手机邮件已经不是什么新鲜事。通过 Android 可以很方便地发送邮件。我们可以使用内置的 Gmail 引擎来发送邮件，也可以使用 SMTP 来发送 E-mail。下面实例演示了如何使用 Gmail 客户端来发送 E-mail。实例步骤说明如下。

① 创建一个项目“Chapter06_Intent_Email”，在该项目的“com.amaker.ch06.app”包中创建一个 MainActivity 类。

```
package com.amaker.ch06.app;

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

/**
 * @author 郭宏志
 * 发送 Email
 */
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置当前 Activity 的界面布局
        setContentView(R.layout.main);
    }
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在该布局文件中添加三个 TextView、三个 EditText 和一个 Button。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:text="收件人："
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>
    <EditText
        android:text=""
        android:id="@+id/toEditText01"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"></EditText>
    <TextView
        android:text="主题："
        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>
    <EditText
        android:text=""
        android:id="@+id/subjectEditText01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"></EditText>
    <TextView
        android:text="内容："
        android:id="@+id/TextView03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>
    <EditText
        android:text=""
        android:id="@+id/contentEditText01"
        android:layout_width="fill_parent"
        android:layout_height="100px"></EditText>
    <Button
        android:text="发送"
        android:id="@+id/sendButton01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 在 MainActivity 的顶部声明使用到的视图组件，在 onCreate()方法中实例化它们。

```
// 声明视图组件
private EditText toEditText,subjectEditText,contentEditText;
private Button sendBtn;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 设置 Activity 的界面布局
    setContentView(R.layout.main);
    // 通过 findViewById 方法获得 EditText 对象
    toEditText = (EditText)findViewById(R.id.toEditText01);
    // 通过 findViewById 方法获得 EditText 对象
    subjectEditText = (EditText)findViewById(R.id.subjectEditText01);
    // 通过 findViewById 方法获得 EditText 对象
    contentEditText = (EditText)findViewById(R.id.contentEditText01);
    // 通过 findViewById 方法获得 Button 对象
    sendBtn = (Button)findViewById(R.id.sendButton01);
}
```

④ 创建一个单击事件监听器。在监听器的 onClick()方法中创建 Intent，并设置其额外信息来发送 E-mail。

```
    // 发送按钮单击监听器
```

```
private OnClickListener listener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 邮件目标地址
        String to = toEditText.getText().toString();
        // 邮件主题
        String subject = subjectEditText.getText().toString();
        // 邮件内容
        String content = contentEditText.getText().toString();
        // 创建 Intent
        Intent emailIntent = new Intent(android.content.Intent.ACTION_SEND);
        // 设置内容类型
        emailIntent.setType("plain/text");
        // 设置额外信息
        emailIntent.putExtra(android.content.Intent.EXTRA_EMAIL, new String[] {to});
        emailIntent.putExtra(android.content.Intent.EXTRA_SUBJECT, subject);
        emailIntent.putExtra(android.content.Intent.EXTRA_TEXT, content);
        // 启动 Activity
        startActivity(Intent.createChooser(emailIntent, "发
送邮件..."));
    }
};
```

⑤ 在 onCreate()方法中，为按钮添加单击事件监听器。

```
// 为按钮添加单击监听器
sendBtn.setOnClickListener(listener);
```

程序运行结果如图 6.9 所示。

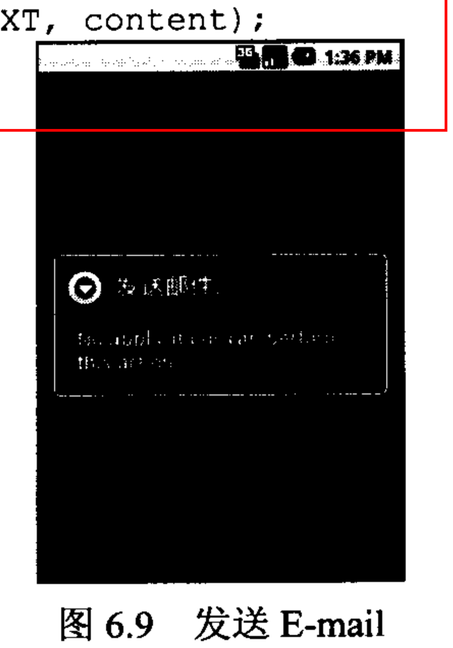


图 6.9　发送 E-mail

> **注意**：由于 Android 模拟器没有内置 Gmail 客户端，所以程序会显示“No applications can perform this action”。在真实设备中运行是正常的。

6.3 Intent 的实现策略

Intent 是如何找到目标组件的呢？Intent 寻找目标组件的方式可以分为两种。一种是显式 Intent，这种方式是通过指定 Intent 组件名称来实现的，这种方式在我们上面讲到的 Intent 组件名称属性时提到过，它一般用在源组件知道目标组件名称的前提下，一般是在相同应用程序内部实现的。那么，不同应用程序之间呢？我们并不知道目标组件的名称，又该如何寻找到目标组件呢？在这种情况下，我们使用第二种方式，隐式 Intent，这种方式是通过 Intent Filter 实现的。

显式 Intent 比较简单，这里不再赘述，本节重点讲述隐式 Intent 的实现策略。在使用 Intent Filter 进行过滤时 Intent 通常考虑的三个属性是 Action、Data 和 Category。

对于显式 Intent，Android 不需要去做解析，因为目标组件已经很明确，Android 需要

解析的是那些间接 Intent，通过解析，将 Intent 映射给可以处理此 Intent 的 Activity、IntentReceiver 或 Service。

Intent 解析机制主要是通过查找已注册在 AndroidManifest.xml 中的所有 IntentFilter 及其中定义的 Intent，最终找到匹配的 Intent。在这个解析过程中，Android 是通过 Intent 的 Action、Category 和 Data 这三个属性来进行判断的，判断方法说明如下。

1. Action 测试

如果 Intent 指明定了 Action，则目标组件 IntentFilter 的 Action 列表中就必须包含这个 Action，否则不能匹配。如果 Intent 没有指定 Action，则 Action 测试自动通过。

Action 的声明在 AndroidManifest.xml 中。每个 Android 组件（例如，Activity、Service 等）都会有一个<intent-filter>元素来声明 Intent 过滤器。Action 作为<intent-filter>的子元素来声明。例如：

```
<intent-filter . . . >
    <action android:name="com.amaker.ch06.app.TEST_ACTION1" />
    <action android:name=" com.amaker.ch06.app.TEST_ACTION2" />
    <action android:name=" com.amaker.ch06.app.TEST_ACTION3" />
    . . .
</intent-filter>
```

<intent-filter>列表中的 Action 不能为空，否则程序将阻塞不能通过。如果 Intent 对象指定了一个 Action 属性，那么要想通过 Action 测试，Intent 对象指定的属性必须和 Intent Filter 中的一个匹配，否则不能通过测试。如果 Intent 对象没有指定 Action 属性，则自动通过测试。

下面我们通过实例来演示 Action 的测试情况。实例步骤说明如下。

① 创建一个 Android 工程“Chapter06_Intent_Filter”，在“com.amaker.ch06.app”包中创建一个 MainActivity。

```
package com.amaker.ch06.app;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
/**
 *
 * @author 郭宏志
 * Intent Filter 测试
 */
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
       // 设置 Activity 界面布局
        setContentView(R.layout.main);
    }
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在其中添加一个 Button 组件。

```
<?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:text="测试 Intent Filter"
      android:id="@+id/Button01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 在工程的“com.amaker.ch06.app”包中创建一个 TestActivity 类。

```
package com.amaker.ch06.app;

import android.app.Activity;
import android.os.Bundle;
/**
 * @author 郭宏志
 * 测试 Intent Filter
 */
public class TestActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
     // 设置 Activity 界面布局
      setContentView(R.layout.main1);
   }
}
```

④ 在工程的 res\layout\目录中创建一个 main1.xml 布局文件，添加一个 TextView 组件。

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

```
    <TextView
        android:text="测试 Intent Filter"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></TextView>
</LinearLayout>
```

⑤ 将 TestActivity 类声明在 AndroidManifest.xml 配置文件中。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch06.app"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity android:name="TestActivity" >
      <intent-filter>
         <action android:name="com.amaker.ch06.app.TEST_ACTION1"/>
         <action android:name="com.amaker.ch06.app.TEST_ACTION2"/>
         <action android:name="com.amaker.ch06.app.TEST_ACTION3"/>

         <category android:name="android.intent.category.DEFAULT"/>
      </intent-filter>
      </activity>

   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

在这里，我们在 TestActivity 的 Intent Filter 中声明了三个 Action 属性。

⑥ 在 MainActivity 的顶部声明 Button 实例并在 onCreate()方法中实例化该 Button，为 Button 添加单击监听器，在 onClieck 方法中，声明 Intent 对象，添加其 Action 属性。

```
// 声明 Button
  private Button btn;
  private static final String ACTION1 = "com.amaker.ch06.app.TEST_ACTION1";
  private static final String ACTION2 = "com.amaker.ch06.app.TEST_ACTION2";
  private static final String ACTION3 = "com.amaker.ch06.app.TEST_ACTION3";
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     // 设置内容布局
     setContentView(R.layout.main);
     //实例化按钮
```

```
        btn = (Button)findViewById(R.id.Button01);
        // 添加单击监听器
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                // 实例化 Intent
                Intent intent = new Intent();
                // 设置 Intent 的 Action 属性
                intent.setAction(ACTION1);
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
```

这里我们声明了三个字符串常量，分别和 AndroidManifest.xml 中的 Action 属性对应，并在 onClick()方法中实例化 Intent 对象，为其设置 Action 属性，程序运行结果如图 6.10 所示，我们看到 Action 测试成功，通过该 Intent 正常启动了 TestActivity。

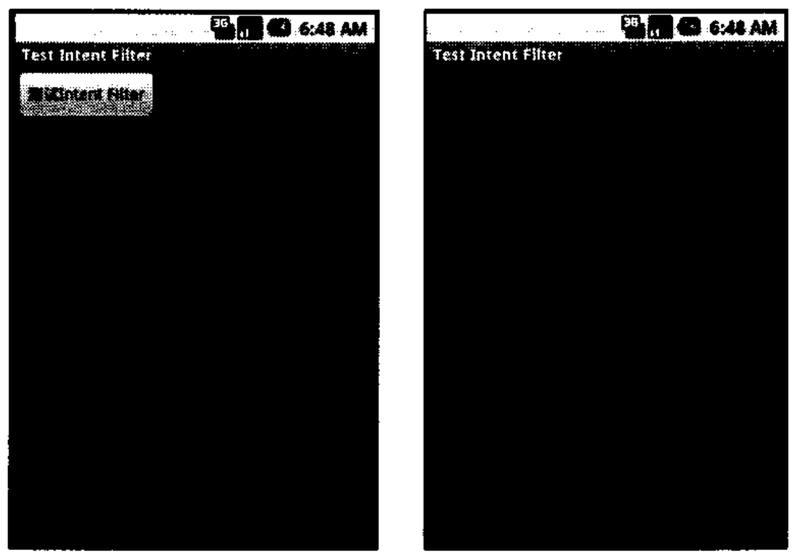


图 6.10　测试 Intent Filter

那么，如果我们没有对 Intent 对象设置 Action 属性呢？还能通过 Action 测试吗？答案是肯定的。在上面实例的基础上，我们去掉 Intent 的 Action 属性，添加一个 Data 属性（这里 Data 属性是任意的），仍然能成功启动 TestActivity。

MainActivity 中的代码改变如下：

```
public void onClick(View view) {
            // 实例化 Intent
            Intent intent = new Intent();
          // intent.setAction(ACTION1);
            // 实例化 Uri
            Uri data = Uri.parse("content://com.amaker.ch07.app/abc");
            // 设置 Intent 的 data 属性
            intent.setData(data);
            // 启动 Activity
           startActivity(intent);
        }
});
```

AndroidManifest.xml 配置文件中的代码改变如下：

```
<activity android:name="TestActivity" >
<intent-filter>
   <action android:name="com.amaker.ch07.app.TEST_ACTION1"/>
   <action android:name="com.amaker.ch07.app.TEST_ACTION2"/>
   <action android:name="com.amaker.ch07.app.TEST_ACTION3"/>

   <data android:scheme="content" android:path="com.amaker.ch07.app/abc"/>

   <category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

以上是我们对 Action 属性测试的验证，下面来看看 Category 属性的验证。

2. Category 测试

Category 属性也是作为<intent-filter>子元素来声明的。例如：

```
<intent-filter>
  <category android:name="android.intent.category.DEFAULT"/>
      <category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

在 Intent 对象中出现的 Category 属性在 Intent Filter 中必须出现，否则不能通过测试。在上面的实例基础上，我们在 MainActivity 中为 Intent 添加了一个 Category，在 AndroidManifest 的 TestActivity 的<intent-filter>中添加了三个<category>，我们发现通过 Intent 对象能够成功启动 TestActivity。

MainActivity 改变代码如下所示：

```
// 类型字符串常量
private static final String CATEGORY1 = "com.amaker.ch07.app.CATEGORY1";
public void onClick(View view) {
      // 实例化 Intent
      Intent intent = new Intent();
     // intent.setAction(ACTION1);
      // 获得 Uri 实例
      Uri data = Uri.parse("content://com.amaker.ch07.app/abc");
      // 设置 Intent 的 data 属性
      intent.setData(data);
      // 设置 Intent 的 Category 属性
      intent.addCategory(CATEGORY1);
      // 启动 Activity
      startActivity(intent);
}
```

注意：“android.intent.category.DEFAULT”属性是启动 Activity 默认的属性，这个必须添加，否则 Category 测试失败。

3. Data 测试

Data 属性是 Android 要访问的数据，和 Action 和 Category 声明方式相同，也是在<intent-filter>中。例如：

```
<intent-filter>
   <category android:name="android.intent.category.DEFAULT"/>
       <category android:name="android.intent.category.BROWSABLE" />
       <category android:name="com.amaker.ch06.app.CATEGORY1"/>
</intent-filter>
```

Data 属性的声明中要指定访问数据的 URI 和 MIME 类型。可以在<category>元素中通过一些属性（android:scheme、android:path、android:port、android:mimeType、android:host、android:scheme 等）来设置，通过这些属性来对应一个典型的 URI 格式 scheme://host:port/path，例如 http://www.google.com。

Data 测试规则说明如下。

- 如果 Intent 对象中没有包含 Data，<intent-filter>列表中也没有包含 Data，则通过测试，这种情况一般是通过 Action 属性来过滤的。例如，在我们上面实例中的 Action 测试中。
- 如果 Intent 对象包含 URI 但是没有包含类型，并且类型不能从 URI 中自动识别，那么<intent-filter>列表中也只能包含 URI，这样才能通过测试。
- 相反，如果 Intent 对象只包含类型，没有包含 URI，那么在<intent-filter>中也只能包含类型，不能包含 URI。
- 如果 Intent 对象中既包含 URI 又包含 Type，那么在<intent-filter>中也必须二者都包含才能通过测试。

下面我们来看一种特殊情况，我们定义的 Intent 和系统中的重合又该如何呢？例如，我们都知道如果一个 Intent 的 Action 被定义为“android.intent.action.VIEW”，Data 被定义为 http://www.google.com。它将能访问 Google 网站。那么，我们自己定义的 Intent 和它的 Action 及 Data 属性相同，又该如何呢？在这种情况下，系统会提示我们选择哪一个组件来运行。下面我们来做一个测试。

① 在 AndroidManifest.xml 的 TestActivity 的<intent-filter>中添加一个 Action 属性“<action android:name="android.intent.action.VIEW"/>”和一个 Data 属性“<data android: scheme="http" android:path="www.google.com" />”。

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <data android:scheme="http" android:path="www.google.com" />
</intent-filter>
```

② 为 MainActivity 中的 Intent 对象设置 Action 和 Data 属性同上。

```
intent.setAction("android.intent.action.VIEW");
   intent.setData(Uri.parse("http://www.google.com"));
```

程序运行结果如图 6.11 所示。

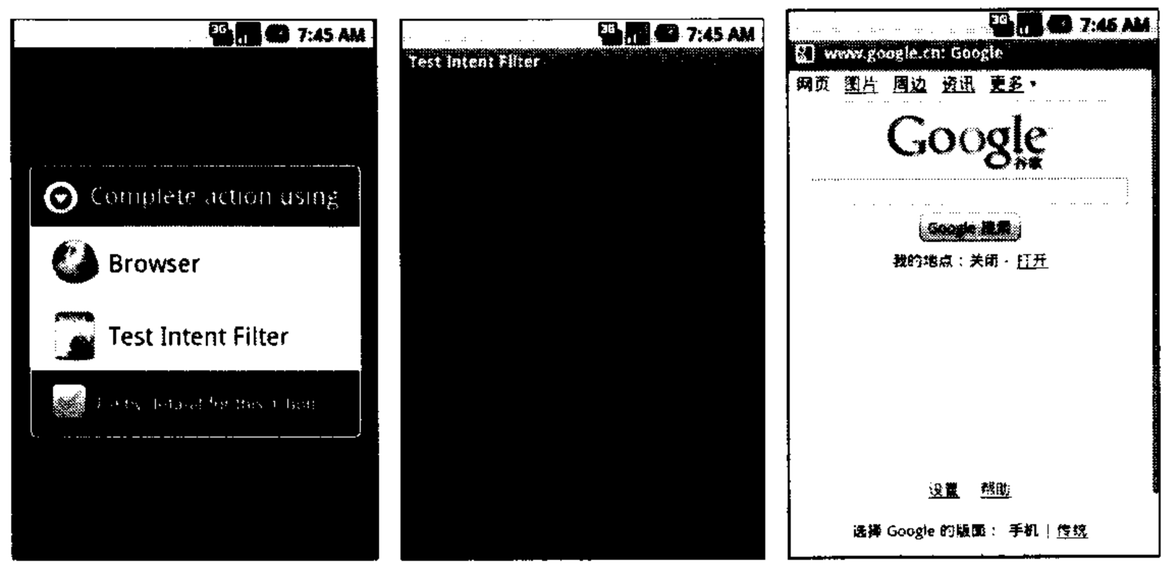
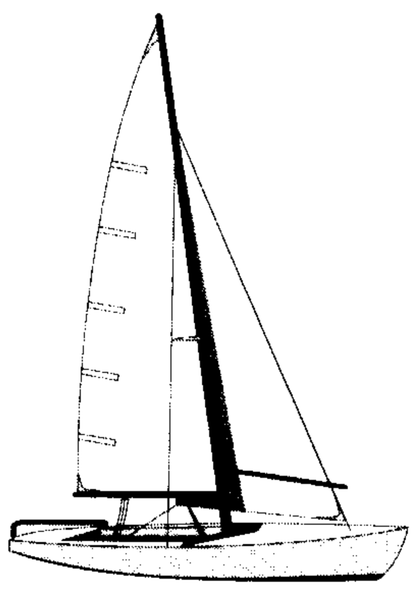


图 6.11　测试 Intent Filter



第 7 章　Android Service 组件

Service 是 Android 系统提供的四种组件之一，它的地位和 Activity 是并列的，只不过没有 Activity 的使用频率高。顾名思义 Service 就是运行在后台的一种服务程序，一般很少和用户交互，因此没有可视化界面。

定义一个 Service 类比较简单，只要继承 Service 类，实现其生命周期中的方法就可以了。一个定义好的 Service 必须在 AndroidManifest.xml 配置文件中通过<service>元素声明才能使用。

Service 有自己的生命周期，我们可以调用 startService()启动一个 Service 或者使用 bindService()方法来绑定一个存在的 Service，还可以通过 RPC（远程进程调用）机制来实现不同进程间 Service 的调用。

7.1 Service 简介

7.1.1 创建一个 Service

创建一个 Service 类比较简单，只要定义一个类继承 Service，覆盖该类中相应的方法就可以了。Service 中定义了一系列和自身声明周期相关的方法，这些方法有：

- onBind（Intent intent）：是必须实现的一个方法，返回一个绑定的接口给 Service。
- onCreate()：当 Service 第一次被创建时，由系统调用。
- onStart（Intent intent, int startId）：当通过 startService()方法启动 Service 时，该方法被调用。
- onDestroy()：当 Service 不再使用，系统调用该方法。

下面的代码演示了如何创建一个 Service。

```
/**
 * @author 郭宏.志
```

```
 * 测试 Service
 */
public class MyService extends Service{
    // 可以返回 null，通常返回一个有 aidl 定义的接口
    public IBinder onBind(Intent intent) {
        return null;
    }
    // Service 创建时调用
    public void onCreate() {
        super.onCreate();
    }
    // 当客户端调用 startService()方法启动 Service 时，该方法被调用
    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
    }
    // 当 Service 不再使用时调用
    public void onDestroy() {
        super.onDestroy();
    }
}
```

要想使用该 Service 必须在 AndroidManif.xml 配置文件中使用<service>元素声明该 Service。在<service>元素中添加<intent-filter>指定如何访问该 Service。

```
<service android:name="MyService">
    <intent-filter>
        <action android:name="com.amaker.ch08.app.action.MY_SERVICE"/>
    </intent-filter>
</service>
```

7.1.2 启动和停止 Service

一旦定义好一个 Service 就可以在其他组件中启动该 Service 来使用它了。启动一个 Service 使用 Context.startService（Intent intent）方法，你会发现这和启动一个 Activity 非常相似，也是传递一个 Intent。当我们调用 startService()方法时，被调用的 Service 会调用它的 onCreate()方法（如果该 Service 还未创建），接着调用 onStart()方法。一旦 Service 启动后将一直运行直到调用了 Context.stopService()或者 stopSelf()。下面的代码演示了如何启动和停止一个 Service。

```
// 创建 Intent
Intent intent = new Intent();
// 设置 Action 属性
intent.setAction("com.amaker.ch08.app.action.MY_SERVICE");
// 启动该 Service
startService(intent);
// 停止一个 Service
stopService(intent);
```

7.1.3　绑定一个已经存在的 Service

我们可以调用 startService()方法来启动一个 Service 方法，也可以通过 bindService()方法来绑定一个 Service。和调用 startService()方法一样，Service 会调用 onCreate()方法来创建 Service（如果它未创建），但是它不会调用 onStart()方法而是调用 onBind()返回客户端一个 IBinder 接口。绑定 Service 一般是用在远程 Service 调用。有关这个话题的详细内容下面将详细介绍。

绑定 Service 需要三个参数：bindService(intent, conn, Service.BIND_AUTO_CREATE)；第一个是 Intent；第二个是 ServiceConnection 对象，我们创建该对象实现其 onServiceConnected()和 onServiceDisconnected()来判断连接成功或断开连接；第三个参数是如何创建 Service，一般指定绑定时自动创建。

```
// 连接对象
ServiceConnection conn = new ServiceConnection() {
@Override// 连接时调用
   public void onServiceConnected(ComponentName name, IBinder service) {
       Log.i("SERVICE", "连接成功！");
   }
   @Override// 断开时调用
   public void onServiceDisconnected(ComponentName name) {
       Log.i("SERVICE", "断开连接！");
   }
};

// 绑定 Service
bindService(intent, conn, Service.BIND_AUTO_CREATE);
```

7.1.4　Service 实例演示

下面通过一个实例来演示如何创建、启动、停止及绑定一个 Service。实例步骤说明如下。

① 创建一个 Android 工程“Chapter07_Service_Example”，在“com.amaker.ch07.app”包中创建一个 MainActivity。在该类的顶部声明四个 Button 对象，分别用来启动 Service、停止 Service、绑定 Service 和解除绑定 Service。

```
package com.amaker.ch07.app;

import android.app.Activity;
import android.app.Service;
import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
```

```
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

/**
 * @author 郭宏志
 * 测试Service
 */

public class MainActivity extends Activity {
    // 声明Button
    private Button startBtn,stopBtn,bindBtn,unbindBtn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置Activity的界面布局
        setContentView(R.layout.main);
        // 通过findViewById方法获得Button实例
        startBtn = (Button)findViewById(R.id.startButton01);
        stopBtn = (Button)findViewById(R.id.stopButton02);
        bindBtn = (Button)findViewById(R.id.bindButton03);
        unbindBtn = (Button)findViewById(R.id.unbindButton04);

        // 添加监听器
        startBtn.setOnClickListener(startListener);
        stopBtn.setOnClickListener(stopListener);
        bindBtn.setOnClickListener(bindListener);
        unbindBtn.setOnClickListener(unBindListener);

    }
    // 启动Service监听器
    private OnClickListener startListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 创建Intent
            Intent intent = new Intent();
            // 设置Action属性
            intent.setAction("com.amaker.ch07.app.action.MY_SERVICE");
            // 启动该Service
            startService(intent);
        }
    };
    // 停止Service监听器
    private OnClickListener stopListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 创建Intent
            Intent intent = new Intent();
            // 设置Action属性
            intent.setAction("com.amaker.ch07.app.action.MY_SERVICE");
```

```
            // 启动该 Service
            stopService(intent);
        }
    };

    // 连接对象
    private ServiceConnection conn = new ServiceConnection() {
        @Override // 连接时调用
        public void onServiceConnected(ComponentName name, IBinder service) {
            // 输出日志
            Log.i("SERVICE", "连接成功！");
            // 通过 Toast 显示信息
            Toast.makeText(MainActivity.this, "连接成功！", Toast.LENGTH_LONG).show();
        }
        @Override // 断开时调用
        public void onServiceDisconnected(ComponentName name) {
            // 输出日志
            Log.i("SERVICE", "断开连接！");
            // 通过 Toast 显示信息
            Toast.makeText(MainActivity.this, "断开连接！", Toast.LENGTH_LONG).show();
        }
    };

    // 绑定 Service 监听器
    private OnClickListener bindListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 创建 Intent
            Intent intent = new Intent();
            // 设置 Action 属性
            intent.setAction("com.amaker.ch07.app.action.MY_SERVICE");
            // 绑定 Service
            bindService(intent, conn, Service.BIND_AUTO_CREATE);
        }
    };


    // 解除绑定 Service 监听器
    private OnClickListener unBindListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 创建 Intent
            Intent intent = new Intent();
            // 设置 Action 属性
            intent.setAction("com.amaker.ch07.app.action.MY_SERVICE");
            // 解除绑定 Service
            unbindService(conn);
        }
    };
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在该文件中添加四个

Button 组件。

```
<?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:text="启动 Service"
        android:id="@+id/startButton01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button
        android:text="停止 Service"
        android:id="@+id/stopButton02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button
        android:text="绑定 Service"
        android:id="@+id/bindButton03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button
        android:text="解除绑定"
        android:id="@+id/unbindButton04"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 创建一个 MyService 类继承 Service，覆盖其生命周期中的方法，并在各个方法中显示信息。

```
package com.amaker.ch07.app;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

/**
 * @author 郭宏志
 * 测试 Service
 */
public class MyService extends Service{

    // 可以返回 null，通常返回一个有 aidl 定义的接口
```

```
    public IBinder onBind(Intent intent) {
        Log.i("SERVICE", "onBind.............");
        Toast.makeText(MyService.this, "onBind.............", Toast.LENGTH_LONG).
show();
        return null;
    }
    // Service 创建时调用
    public void onCreate() {
        Log.i("SERVICE", "onCreate.............");
        Toast.makeText(MyService.this, "onCreate.............", Toast.LENGTH_LONG).
show();
    }
    // 当客户端调用 startService()方法启动 Service 时，该方法被调用
    public void onStart(Intent intent, int startId) {
        Log.i("SERVICE", "onStart.............");
        Toast.makeText(MyService.this, "onStart.............", Toast.LENGTH_LONG).
show();
    }
    // 当 Service 不再使用时调用
    public void onDestroy() {
        Log.i("SERVICE", "onDestroy.............");
       Toast.makeText(MyService.this, "onDestroy.............", Toast.LENGTH_LONG).
show();
    }
}
```

④ 在 AndroidManifest.xml 配置文件中声明 Activity 和 Service。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch07.app"
     android:versionCode="1"
     android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">

        <activity android:name=".MainActivity"
                 android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:name="MyService">
        <intent-filter>
            <action android:name="com.amaker.ch07.app.action.MY_SERVICE"/>
        </intent-filter>
        </service>

    </application>
    <uses-sdk android:minSdkVersion="3" />

</manifest>
```

程序运行结果如图 7.1 和图 7.2 所示。

图 7.1　测试 Service1

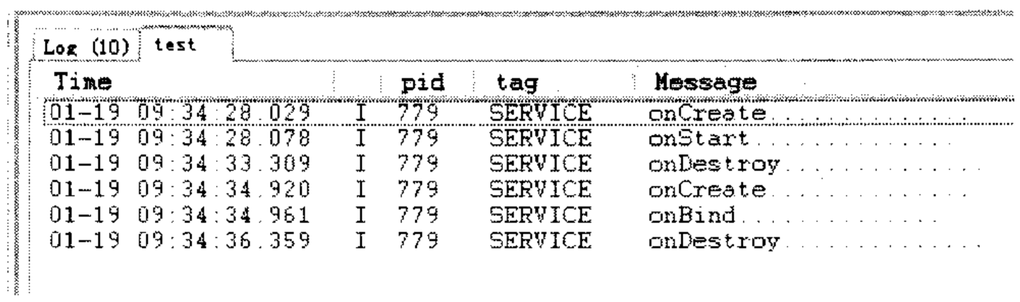


图 7.2　测试 Service2

7.2 远程 Service 调用

在 Android 平台中，各个组件运行在自己的进程中。它们之间是不能相互访问的，但是在程序之间不可避免地要传递一些对象，在进程之间相互通信。为了实现进程之间的相互通信，Android 采用了一种轻量级的实现方式 RPC（Remote Procedure Call，远程进程调用）来完成进程之间通信，并且 Android 通过接口定义语言（Android Interface Definition Language，AIDL）来生成两个进程之间相互访问的代码。例如，你在 Activity 里的代码需要访问 Service 中的一个方法，那么就可以通过这种方式来实现了。

AIDL RPC 机制是通过接口来实现的，类似 Windows 中的 COM 或者 Corba，但它是轻量级的，客户端和被调用实现之间是通过代理模式实现的。代理类和被代理类实现同一个接口即 Ibinder 接口。

7.2.1　创建一个 AIDL 文件

定义一个 AIDL 文件的语法和定义一个 Java 接口的语法相似，只不过文件的扩展名是“.aidl”。在 AIDL 文件中可以声明任意多个方法，方法可以带参数也可以有返回值，参数和返回值可以是任意类型。需要注意的是，你必须导入除了内建类型（例如：int、boolean 等）外的任何其他类型，即使它们在同一个包中。具体规则如下：

- Java 原始类型不需要导入。
- String、List、Map 和 CharSequence 不需要导入。

定义好的 AIDL 文件可以使用 Android SDK→Tools 下面的 AIDL 工具生产 Java 代码，如果使用基于 Eclipse 的 ADT 插件，代码会自动生成。下面我们使用 Eclipse 的 ADT 插件创建一个 IPerson.aidl 文件。该文件中声明三个方法，分别是 setName（String name）设置名称、setAge（int age）设置年龄和 String display()显示名字和年龄。

```
package com.amaker.ch07.app;
// Iperson 接口
interface IPerson {
   // 设置年龄方法
   void setAge(int age);
   // 设置姓名方法
   void setName(String name);
   // 显示信息方法
   String display();
}
```

当我们创建好上述文件之后，刷新该文件系统自动在 gen 目录下的“com.amaker.ch07.app”包中创建一个 Java 实现接口。

```
package com.amaker.ch07.app;

import java.lang.String;
import android.os.RemoteException;
import android.os.IBinder;
import android.os.IInterface;
import android.os.Binder;
import android.os.Parcel;

// AIDL 文件生成的 Java 接口实现类
public interface IPerson extends android.os.IInterface {
   // 存根类，调用该类的 asInterface 返回目标接口
   public static abstract class Stub extends android.os.Binder implements
         com.amaker.ch07.app.IPerson {
      private static final java.lang.String DESCRIPTOR = "com.amaker.ch07.app.
IPerson";

      public Stub() {
         this.attachInterface(this, DESCRIPTOR);
      }

      /**
       * 转换一个 IBinder 对象成一个 Iperson 接口，生成一个代理
       */
      public static com.amaker.ch07.app.IPerson asInterface(
            android.os.IBinder obj) {
         if ((obj == null)) {
            return null;
```

```
        }
        android.os.IInterface iin = (android.os.IInterface) obj
                .queryLocalInterface(DESCRIPTOR);
        if (((iin != null) && (iin instanceof com.amaker.ch08.app.IPerson))) {
            return ((com.amaker.ch07.app.IPerson) iin);
        }
        // 调用代理类实现
        return new com.amaker.ch07.app.IPerson.Stub.Proxy(obj);
    }

    public android.os.IBinder asBinder() {
        return this;
    }
    // 处理方法
    public boolean onTransact(int code, android.os.Parcel data,
            android.os.Parcel reply, int flags)
            throws android.os.RemoteException {
        switch (code) {
        case INTERFACE_TRANSACTION: {
            reply.writeString(DESCRIPTOR);
            return true;
        }
        case TRANSACTION_setAge: {
            data.enforceInterface(DESCRIPTOR);
            int _arg0;
            _arg0 = data.readInt();
            this.setAge(_arg0);
            reply.writeNoException();
            return true;
        }
        case TRANSACTION_setName: {
            data.enforceInterface(DESCRIPTOR);
            java.lang.String _arg0;
            _arg0 = data.readString();
            this.setName(_arg0);
            reply.writeNoException();
            return true;
        }
        case TRANSACTION_display: {
            data.enforceInterface(DESCRIPTOR);
            java.lang.String _result = this.display();
            reply.writeNoException();
            reply.writeString(_result);
            return true;
        }
        }
        return super.onTransact(code, data, reply, flags);
    }
    // 静态内部代理类，实现了 IPerson 接口
    private static class Proxy implements com.amaker.ch07.app.IPerson {
        // IBinder 实例
        private android.os.IBinder mRemote;
```

```
        // 构造方法
        Proxy(android.os.IBinder remote) {
            mRemote = remote;
        }
        public android.os.IBinder asBinder() {
            return mRemote;
        }
        // 获得接口描述常量
        public java.lang.String getInterfaceDescriptor() {
            return DESCRIPTOR;
        }
        // 设置年龄实现
        public void setAge(int age) throws android.os.RemoteException {
            android.os.Parcel _data = android.os.Parcel.obtain();
            android.os.Parcel _reply = android.os.Parcel.obtain();
            try {
                _data.writeInterfaceToken(DESCRIPTOR);
                _data.writeInt(age);
                mRemote.transact(Stub.TRANSACTION_setAge, _data, _reply, 0);
                _reply.readException();
            } finally {
                _reply.recycle();
                _data.recycle();
            }
        }
        // 设置名称实现
        public void setName(java.lang.String name)
                throws android.os.RemoteException {
            android.os.Parcel _data = android.os.Parcel.obtain();
            android.os.Parcel _reply = android.os.Parcel.obtain();
            try {
                _data.writeInterfaceToken(DESCRIPTOR);
                _data.writeString(name);
                mRemote
                        .transact(Stub.TRANSACTION_setName, _data, _reply,
                                0);
                _reply.readException();
            } finally {
                _reply.recycle();
                _data.recycle();
            }
        }
        // 显示信息方法实现
        public java.lang.String display() throws android.os.RemoteException {
            android.os.Parcel _data = android.os.Parcel.obtain();
            android.os.Parcel _reply = android.os.Parcel.obtain();
            java.lang.String _result;
            try {
                _data.writeInterfaceToken(DESCRIPTOR);
                mRemote
                        .transact(Stub.TRANSACTION_display, _data, _reply,
                                0);
                _reply.readException();
                _result = _reply.readString();
            } finally {
                _reply.recycle();
```

```
                _data.recycle();
            }
            return _result;
        }
    }
    // 处理设置年龄方法标示常量
    static final int TRANSACTION_setAge = (IBinder.FIRST_CALL_TRANSACTION + 0);
    // 处理设置姓名方法标示常量
    static final int TRANSACTION_setName = (IBinder.FIRST_CALL_TRANSACTION + 1);
    // 处理设置显示方法标示常量
    static final int TRANSACTION_display = (IBinder.FIRST_CALL_TRANSACTION + 2);
}
// 设置年龄方法
public void setAge(int age) throws android.os.RemoteException;
// 设置姓名方法
public void setName(java.lang.String name)
        throws android.os.RemoteException;
// 显示方法
public java.lang.String display() throws android.os.RemoteException;
}
```

认真分析上述代码，我们会发现它是使用代理模式来实现的。我们一般是定义该接口的静态内部类 Stub 的 asInterface()方法，返回我们的接口。

7.2.2 实现 AIDL 文件生成的 Java 接口

AIDL 会生成一个和.aidl 文件同名的 Java 接口文件，该接口中有一个静态抽象内部类 Stub，该类中声明了 AIDL 文件中定义的所有方法。其中有一个重要的方法是 asInterface()，该方法通过代理模式返回 Java 接口的实现。

我们可以定义一个实现类（例如 PersonImpl），该类继承 Stub 类，实现我们定义的三个方法。

```
package com.amaker.ch07.app;
import android.os.RemoteException;
/**
 *
 * @author 郭宏志
 * IPerson 接口实现类
 */
public class IPersonImpl extends IPerson.Stub{
    // 声明两个变量
    private int age;
    private String name;
    @Override
    // 显示 name 和 age
    public String display() throws RemoteException {
        return "name:"+name+";age="+age;
    }
    @Override
    // 设置 age
    public void setAge(int age) throws RemoteException {
        this.age = age;
    }
    @Override
```

```
    // 设置 name
    public void setName(String name) throws RemoteException {
        this.name = name;
    }
}
```

7.2.3　将你的接口暴露给客户端

现在我们已经实现了 Iperson 接口，接下来我们要看一看如何将该接口暴露给客户端调用。一般我们通过定义一个 Service 来实现。在 Service 的 onBind()方法中返回该接口，当我们绑定该接口时调用该方法。

```
package com.amaker.ch08.app;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

import com.amaker.ch08.app.IPerson.Stub;

/**
 *
 * @author 郭宏志
 * 使用 Service 将接口暴露给客户端
 */
public class MyRemoteService extends Service{
    // 声明 IPerson 接口
    private Stub iPerson = new IPersonImpl();
    @Override
    public IBinder onBind(Intent intent) {
        return iPerson;
    }
}
```

7.2.4　客户端调用

接下来我们定义一个 Activity 来绑定远程 Service，获得 Iperosn 接口，通过 RPC 机制调用接口中的方法。

```
/**
 * @author 郭宏志
 * RPC 测试
 */
public class MainActivity extends Activity {
    // 声明 IPerson 接口
    private IPerson iPerson;
    // 声明 Button
    private Button btn;
    // 实例化 ServiceConnection
    private ServiceConnection conn = new ServiceConnection() {
        @Override
```

```
        synchronized public void onServiceConnected(ComponentName name, IBinder
service) {
            // 获得 IPerson 接口
            iPerson = IPerson.Stub.asInterface(service);
            if (iPerson != null)
                try {
                    // RPC 方法调用
                    iPerson.setName("hz.guo");
                    iPerson.setAge(30);
                    String msg = iPerson.display();
                    // 显示方法调用返回值
                    Toast.makeText(MainActivity.this, msg, Toast.LENGTH_LONG)
                            .show();
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {

        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前视图布局
        setContentView(R.layout.main);
        // 实例化 Button
        btn = (Button) findViewById(R.id.Button01);
        //为 Button 添加单击事件监听器
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 实例化 Intent
                Intent intent = new Intent();
                // 设置 Intent Action 属性
                intent

    .setAction("com.amaker.ch07.app.action.MY_REMOTE_SERV
ICE");
                // 绑定服务
                bindService(intent,   conn,   Service.BIND_
AUTO_CREATE);
            }
        });
    }
}
```

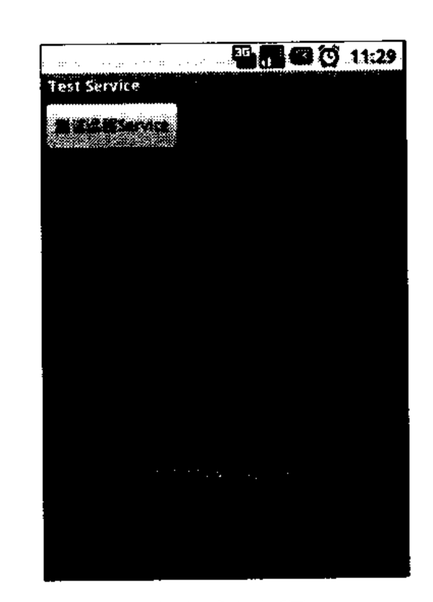
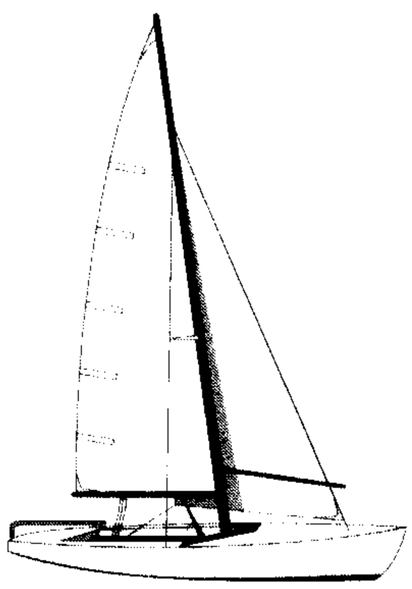


图 7.3　测试远程 Service

程序运行结果如图 7.3 所示。



第 8 章　Android 广播事件处理 Broadcast Receiver

Broadcast Receiver 顾名思义是指广播接收器，它和事件处理机制类似，只不过事件处理机制是程序组件级别的（例如，某个按钮的单击事件），而广播事件处理机制是系统级别的。到目前为止我们可以使用 Intent 来启动一个程序组件，我们还可以通过使用 sendBroadcast()方法来发起一个系统级别的事件广播来传递消息。我们可以在你的应用程序中实现 Broadcast Receiver 来监听和响应这些广播的 Intent。

事件的广播比较简单，同样还是构建 Intent 对象，然后调用 sendBroadcast()方法将广播发出。事件的接收是通过定义一个继承 BroadcastReceiver 的类来实现的，继承该类后覆盖其 onReceive ()方法，在该方法中响应事件。

Android 系统中定义了很多标准的 Broadcast Action 来响应系统广播事件。例如，ACTION_TIME_CHANGED（时间改变时触发）、ACTION_BOOT_COMPLETED（系统启动完成后触发）、ACTION_PACKAGE_ADDED（添加包时触发）、ACTION_BATTERY_CHANGED（电量低时触发）。当然，我们也可以自己定义 Broadcast Receiver 接收广播事件。

通过本章的学习你将掌握如下内容：

- 自己定义 Broadcast Receiver 来处理广播事件
- 系统广播事件的使用
- Notification 和 NotificationManager 的使用
- AlarmManager 的使用

8.1 自己定义 Broadcast Receiver 来处理广播事件

自己定义一个广播是比较简单的。首先在你的程序组件里构建你要广播的 Intent，使

用 sendBroadcast 方法发送出去。其次定义一个广播接收器，该广播接收器继承 BroadcastReceiver，并且覆盖 onReceive()方法来响应事件。最后注册该广播接收器，我们可以在代码中注册，也可以在 AndroidManifest.xml 配置文件中注册。下面通过一个实例来演示 Intent 发出广播和 Broadcast Receiver 处理广播的过程，实例步骤说明如下。

① 创建一个工程“Chapter08_Broadcast_Receiver1”，在“com.amaker.ch08.app”包中创建一个 MainActivity。在该类的顶部声明一个字符串定义广播 Action，声明一个 Button 对象响应单击事件发出广播。在 onCreate()方法中设置当前布局视图，实例化 Button 对象，为 Button 添加单击监听器，在 onClick()方法中创建 Intent 对象，为其设置 Action 属性和 Extra 属性，使用该 Intent 发出广播。

```
package com.amaker.ch08.app;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
/**
 *
 * @author 郭宏志
 * 发出广播
 */
public class MainActivity extends Activity {
    // 定义一个 Action 常量
    private static final String MY_ACTION = "com.amaker.ch08.action.MY_ACTION";
    // 定义一个 Button 对象
    private Button btn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前布局视图
        setContentView(R.layout.main);
        btn = (Button)findViewById(R.id.Button01);
        // 为按钮设置单击监听器
        btn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                 // 实例化 Intent 对象
                Intent intent = new Intent();
                // 设置 Intent action 属性
                intent.setAction(MY_ACTION);
                // 为 Intent 添加附加信息
                intent.putExtra("msg", "地瓜地瓜，我是土豆，收到请回复，收到请回复！");
                // 发出广播
                sendBroadcast(intent);
            }
        });
```

```
    }
}
```

② 在工程的 res\layout\目录下创建一个布局文件 main.xml，在其中添加一个 Button 组件。

```
<?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:text="发出广播..."
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

③ 在工程的“com.amaker.ch08.app”包中定义一个 MyReceiver 类，该类继承 BroadcastReceiver 类，覆盖 onReceive()方法来接收广播并显示收到的消息。

```
package com.amaker.ch08.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

/**
 * @author 郭宏志
 * 接收广播
 */
public class MyReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context cxt, Intent intent) {
        // 从 Intent 中获得信息
        String msg = intent.getStringExtra("msg");
        // 使用 Toast 显示
        Toast.makeText(cxt, msg, Toast.LENGTH_LONG).show();
    }
}
```

④ 在工程的 AndroidManifest.xml 配置文件中声明广播接收器组件。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch08.app"
     android:versionCode="1"
```

```
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
     <activity android:name=".MainActivity"
             android:label="@string/app_name">
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>

        <receiver android:name="MyReceiver">
        <intent-filter>
           <action
android:name="com.amaker.ch08.action.MY_ACTION"/>
        </intent-filter>
        </receiver>

    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

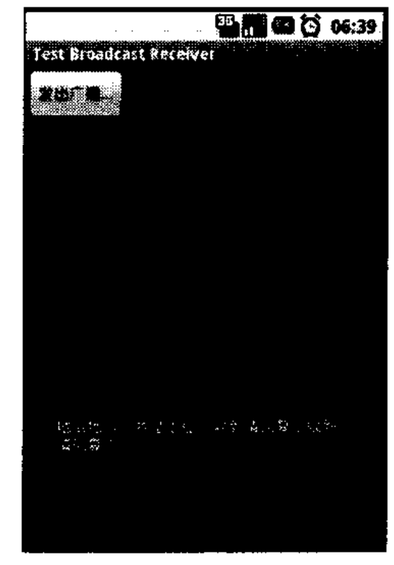


图 8.1　测试广播事件

程序运行结果如图 8.1 所示。

8.2 系统广播事件的使用

除了上面我们自己定义的广播事件，Android 系统还提供了很多标准广播 Action。这些广播是系统自动发出的，我们直接定义事件接收器进行接收就可以使用这些系统消息了。常见的标准广播 Action 常量如表 8.1 所示。

表 8.1　标准广播Action常量

常量名称	常量值	意义
ACTION_BOOT_COMPLETED	android.intent.action.BOOT_COMPLETED	系统启动完成
ACTION_TIME_CHANGED	android.intent.action. ACTION_TIME_CHANGED	时间改变
ACTION_DATE_CHANGED	android.intent.action. ACTION_DATE_CHANGED	日期改变
ACTION_TIMEZONE_CHANGED	android.intent.action. ACTION_TIMEZONE_CHANGED	时区改变
ACTION_BATTERY_LOW	android.intent.action. ACTION_BATTERY_LOW	电量低
ACTION_MEDIA_EJECT	android.intent.action. ACTION_MEDIA_EJECT	插入或拔出外部媒体
ACTION_MEDIA_BUTTON	android.intent.action. ACTION_MEDIA_BUTTON	按下媒体按钮
ACTION_PACKAGE_ADDED	android.intent.action. ACTION_PACKAGE_ADDED	添加包
ACTION_PACKAGE_REMOVED	android.intent.action. ACTION_PACKAGE_REMOVED	删除包

下面我们通过实例来看看如何接收系统发出的广播，本实例我们通过接收系统启动完成（ACTION_BOOT_COMPLETED）广播来测试系统广播事件。接收系统发出的广播，我们不需要自己再定义发出广播的 Intent，只需要定义接收器就可以了。实例步骤说明如下。

① 在 8.1 节的工程基础上，在“com.amaker.ch08.app”包中创建一个 MyReceiver 类，该类继承 BroadcastReceiver，覆盖 onReceive()显示系统启动完成广播信息。

```
package com.amaker.ch08.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

/**
 *
 * @author 郭宏志
 * 显示系统启动完成广播接收器
 */
public class MyReceiver2 extends BroadcastReceiver{
   @Override
   public void onReceive(Context context, Intent intent) {
       // 显示广播信息
       Log.i("my_tag", "BOOT_COMPLETED~~~~~~~~~~~~~~~~"); }
}
```

② 在 AndroidManifest.xml 配置文件中注册该接收器，注意这里<intent-filter>中的<action>属性必须是“android.intent.action.BOOT_COMPLETED”，才能接收系统启动完成广播事件。

```
      <receiver android:name="MyReceiver2">
       <intent-filter>
          <action  android:name="android.intent.action.BOOT_COMPLETED"/>
       </intent-filter>
      </receiver> }
```

③ 当我们重新启动程序，且系统启动完成时，程序运行结果如图 8.2 所示。

图 8.2　测试广播事件

我们既可以在 AndroidManifest 中注册一个广播接收器，也可以通过代码的方式来注册。当然，我们也可以注销一个广播接收器。一般我们是在 Activity.onResume()方法中使用 Context.registerReceiver()方法来注册一个广播接收器，在 Activity.onPause()中使用 unregisterReceiver(r)方法来注销一个广播接收器。下面的代码片段显示了如何使用一个 Intent Filter 注册 Broadcast Receiver。

```
// 实例化 IntentFilter
IntentFilter filter = new IntentFilter();
// 实例化 Receiver
MyReceiver2 r = new MyReceiver2 ();
```

```
// 注册 Receiver
registerReceiver(r, filter);
```

为了注销一个 Broadcast Receiver，应使用 Context.unregisterReceiver 方法，传入一个 Broadcast Receiver 实例。

```
// 注销 Receiver
unregisterReceiver(r);
```

8.3 Notification 和 NotificationManager 的使用

Broadcast Receiver 组件并没有提供可视化的界面用来显示广播信息。这里我们可以使用 Notification 和 NotificationManager 来实现可视化的信息显示。通过使用它们我们可以显示广播信息的内容、图标以及振动等信息。

8.3.1 Notification 和 NotificationManager 简介

使用 Notification 和 NotificationManager 也比较简单，一般获得系统级的服务 NotificationManager，然后实例化 Notification，设置其属性，通过 NotificationManager 发出通知就可以了。基本步骤说明如下。

① 获得系统级的服务 NotificationManager，这里比较简单，通过调用 Context.getSystemService()方法即可实现。

```
String service = NOTIFICATION_SERVICE;
NotificationManager nm = (NotificationManager)getSystemService(service);
```

② 实例化 Notification 对象，并设置其属性。

```
// 实例化 Notification
Notification n = new Notification();
// 设置显示图标，该图标会在状态栏显示
int icon = n.icon = R.drawable.icon;
// 设置显示提示信息，该信息也会在状态栏显示
String tickerText = "Test Notification";
// 显示时间
long when = System.currentTimeMillis();
n.icon = icon;
n.tickerText = tickerText;
n.when = when;

// 也可以通过这种构造方法来设置
Notification n1 = new Notification(icon, tickerText, when);
```

③ 调用 setLatestEventInfo()方法在视图中设置图标和时间。

```
// 实例化 Intent
Intent intent = new Intent(this, MainActivity.class);
```

```
// 获得 PendingIntent
PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
// 设置事件信息
n.setLatestEventInfo(this, "My Title", "My Content", pi);
```

④ 发出通知。

```
// 标示该通知的 ID
int ID = 1;
// 发出通知
nm.notify(ID, n);
```

8.3.2　通知实例演示

下面通过一个实例来演示通知的基本用法，本实例在界面上添加两个按钮：一个用来发出通知；一个用来取消通知。实例步骤说明如下。

① 创建一个 Android 工程“Chapter08_Notification1”，在“com.amaker.ch08.app”包中创建一个 MainActivity 类。在该类的顶部声明使用到的 Button、Notification、NotificationManager 等。在 onCreate()方法中实例化它们，并为 Notification 对象设置属性。

```
package com.amaker.ch08.app;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

/**
 * @author 郭宏志
 * 测试通知
 */

public class MainActivity extends Activity {
   // 声明按钮
   private Button sendBtn,cancelBtn;
   // 声明 Notification
   private Notification n ;
   // 声明 NotificationManager
   private NotificationManager nm;
   // Notification 标示 ID
   private static final int ID = 1;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
```

```
        // 实例化按钮
        sendBtn = (Button)findViewById(R.id.sendButton01);
        cancelBtn = (Button)findViewById(R.id.cancelButton02);

        // 获得 NotificationManager 实例
        String service = NOTIFICATION_SERVICE;
        nm = (NotificationManager)getSystemService(service);

        // 实例化 Notification
        n = new Notification();
        // 设置显示图标，该图标会在状态栏显示
        int icon = n.icon = R.drawable.icon;
        // 设置显示提示信息，该信息也会在状态栏显示
        String tickerText = "Test Notification";
        // 显示时间
        long when = System.currentTimeMillis();
        n.icon = icon;
        n.tickerText = tickerText;
        n.when = when;
        // 为按钮添加监听器
        sendBtn.setOnClickListener(sendListener);
        cancelBtn.setOnClickListener(cancelListener);
    }
}
```

② 在工程的 res\layout\目录下，创建一个 main.xml 布局文件，添加两个 Button 组件。

```
<?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="测试 Notification"
    />

<Button
    android:text="发出通知"
    android:id="@+id/sendButton01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

<Button
    android:text="取消通知"
    android:id="@+id/cancelButton02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 创建两个单击事件监听器：一个用来发出通知；一个用来取消通知。

```
// 发送通知监听器
private OnClickListener sendListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 实例化 Intent
        Intent intent = new Intent(MainActivity.this, MainActivity.class);
        // 获得 PendingIntent
        PendingIntent pi = PendingIntent.getActivity(MainActivity.this, 0, intent, 0);
        // 设置事件信息
        n.setLatestEventInfo(MainActivity.this, "My Title", "My Content", pi);
        // 发出通知
        nm.notify(ID, n);
    }
};
// 取消通知监听器
 private OnClickListener cancelListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 取消通知
        nm.cancel(ID);
    }
};
```

④ 为按钮添加单击事件监听器。

```
// 为按钮添加监听器
sendBtn.setOnClickListener(sendListener);
cancelBtn.setOnClickListener(cancelListener);
```

程序运行结果如图 8.3 所示。

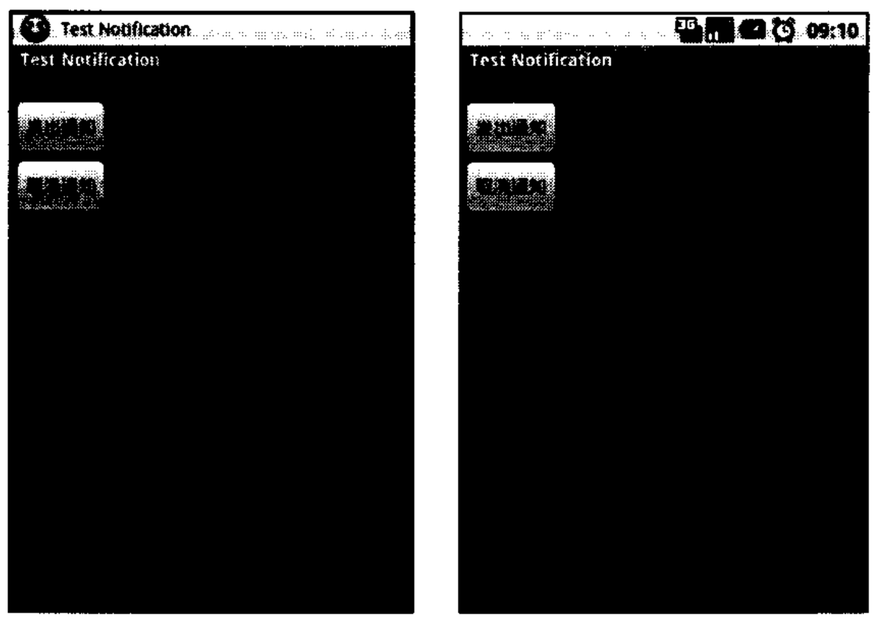


图 8.3　测试 Notification

在上面的实例中我们只是使用 Notification 和 NotificationManager 来显示通知，下面我们来看另外一个实例，该实例是 Notification、NotificationManager 和 Broadcast Receiver 的一个综合实例。这里我们定义一个 MainActivity 发出广播，定义一个 MyReceiver 类接收广

播，当接收到广播后，启动另一个 DisplayActivity，在该类中通过 Notification 和 NotificationManager 来可视化显示广播通知。实例步骤说明如下。

① 创建一个工程“Chapter08_Notification2”，在“com.amaker.ch08.app”包中创建一个 MainActivity 类。在该类的顶部声明一个 Button 和一个 Action 常量。在 onCreate()方法中实例化 Button，并为其添加单击事件监听器，在事件方法中创建 Intent，发出广播。

```
package com.amaker.ch08.app;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
/**
 * @author 郭宏志
 * 测试广播和通知
 */
public class MainActivity extends Activity {
    // 声明 Button
    private Button btn;
    // 定义 Broadcast Receiver action
    private static final String MY_ACTION = "com.amaker.ch08.app.MY_ACTION";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前布局视图
        setContentView(R.layout.main);
        // 实例化 Button
        btn = (Button)findViewById(R.id.Button01);
        // 添加事件监听器
        btn.setOnClickListener(listener);
    }
    // 创建事件监听器
    private OnClickListener listener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 实例化 Intent
            Intent intent = new Intent();
            // 设置 Intent action 属性
            intent.setAction(MY_ACTION);
            // 发起广播
            sendBroadcast(intent);
        }
    };
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在其中添加一个 Button 组件。

```
<?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:text="发出广播"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 创建一个接收广播类 MyReceiver，该类继承 BroadcastReceiver。当收到广播后启动另外一个 Activity 来显示通知。

```
package com.amaker.ch08.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
/**
 * @author 郭宏志
 */
public class MyReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        // 实例化 Intent
        Intent i = new Intent();
        // 在新的任务中启动 Activity
        i.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        // 设置 Intent 启动的组件名称
        i.setClass(context, DisplayActivity.class);
        // 启动 Activity 显示通知
        context.startActivity(i);
    }
}
```

④ 定义一个 DisplayActivity 来显示通知，实现方式和上一个实例相似，这里不再赘述。

```
package com.amaker.ch08.app;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.Button;
/**
 * @author 郭宏志
 */
public class DisplayActivity extends Activity {
    // 声明按钮
    private Button cancelBtn;
    // 声明 Notification
    private Notification n ;
    // 声明 NotificationManager
    private NotificationManager nm;
    // Notification 标示 ID
    private static final int ID = 1;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main2);
        // 实例化按钮
        cancelBtn = (Button)findViewById(R.id.cancelButton02);
        // 获得 NotificationManager 实例
        String service = NOTIFICATION_SERVICE;
        nm = (NotificationManager)getSystemService(service);

        // 实例化 Notification
        n = new Notification();
        // 设置显示图标，该图标会在状态栏显示
        int icon = n.icon = R.drawable.happy;
        // 设置显示提示信息，该信息也会在状态栏显示
        String tickerText = "Test Notification";
        // 显示时间
        long when = System.currentTimeMillis();
        n.icon = icon;
        n.tickerText = tickerText;
        n.when = when;

        // 实例化 Intent
        Intent intent = new Intent(this, MainActivity.class);
        // 获得 PendingIntent
        PendingIntent pi = PendingIntent.getActivity(this, 0, intent, 0);
        // 设置事件信息
        n.setLatestEventInfo(this, "My Title", "My Content", pi);
        // 发出通知
        nm.notify(ID, n);

        // 为按钮添加监听器
        cancelBtn.setOnClickListener(cancelListener);
    }

    // 取消通知监听器
    private OnClickListener cancelListener = new OnClickListener() {
        @Override
```

```
        public void onClick(View v) {
            // 取消通知
            nm.cancel(ID);
        }
    };
}
```

程序运行结果如图 8.4 所示。

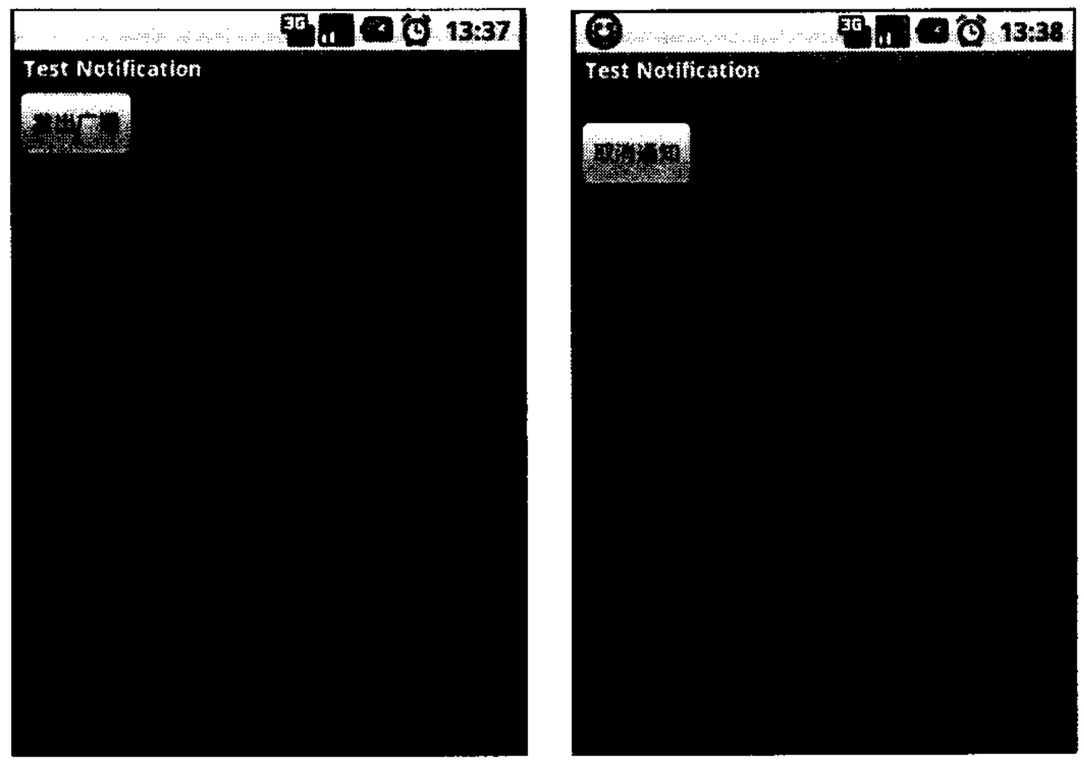


图 8.4　测试 Notification

正如我们上面看到的那样，可以为 Notification 对象设置图标、显示文字等信息。除了这些 Notification，还有很多其他属性可以用来进行提示。例如，设置声音、振动和闪光灯等。设置方式如下所示：

- 提示音

```
n.defaults |= Notification.DEFAULT_SOUND;
n.sound = Uri.parse("file:///sdcard/sound.mp3");
n.sound = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

- 振动

```
n.defaults |= Notification.DEFAULT_VIBRATE;
long[] vibrate = {0,50,100,150};
n.vibrate = vibrate;
```

- 闪光灯

```
n.defaults |= Notification.DEFAULT_LIGHTS;
n.ledARGB = 0xff00ff00;
n.ledOnMS = 300;
n.ledOffMS = 1000;
n.flags |= Notification.FLAG_SHOW_LIGHTS;
```

8.4 AlarmManager 的使用

现在的手机普遍都会有一个闹钟的功能，如果使用 Android 来实现一个闹钟，可以使用 AlarmManager 来实现。AlarmManager 提供了一种系统级的提示服务，允许你安排在将来的某个时间执行一个服务。AlarmManager 对象一般不直接实例化，而是通过 Context.getSystemService(Context.ALARM_SERVICE) 方法获得。表 8.2 列出了 AlarmManager 类的一些常用属性和方法。

表 8.2 AlarmManager常用属性和方法

属性或方法名称	说　明
ELAPSED_REALTIME	设置闹钟时间，从系统启动开始
ELAPSED_REALTIME_WAKEUP	设置闹钟时间，从系统启动开始，如果设备休眠则唤醒
INTERVAL_DAY	设置闹钟时间，间隔一天
INTERVAL_FIFTEEN_MINUTES	间隔 15 分钟
INTERVAL_HALF_DAY	间隔半天
INTERVAL_HALF_HOUR	间隔半小时
INTERVAL_HOUR	间隔 1 小时
RTC	设置闹钟时间，从系统当前时间开始（System.currentTimeMillis()）
RTC_WAKEUP	设置闹钟时间，从系统当前时间开始，设备休眠则唤醒
set(int type, long triggerAtTime, PendingIntent operation)	设置在某个时间执行闹钟
setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)	设置在某个时间重复执行闹钟
setInexactRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)	设置在某个时间重复执行闹钟，但不是间隔固定时间

AlarmManager 的使用步骤说明如下。

① 获得 AlarmManager 实例。

```
AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
```

② 闹钟一般是通过发出一个广播来实现的，所以要定义一个 PendingIntent 发出广播。

③ 调用 AlarmManager 的方法，设置定时或重复提醒。

下面通过一个实例来演示 AlarmManager 的使用。在该实例中定义一个 MainActivity，在该类中添加一个 Button，单击 Button 设置闹钟。定义一个 BroadcastReceiver 接收广播，并显示信息。实例步骤说明如下。

① 创建一个 Android 工程“Chapter08_Broadcast_AlarmManager”，在“com.amaker.ch08.app”包中创建一个 MainActivity 类。在该类的顶部声明两个 Button 实例和一个广播 Action 属性，在 onCreate()方法中实例化 Button，为两个按钮添加单击监听事件：一个

用于设置闹钟，一个用于取消闹钟。

```
package com.amaker.ch08.app;

import android.app.Activity;
import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

/**
 *
 * @author 郭宏志
 * 测试AlarmManager
 */
public class MainActivity extends Activity {
    // 声明Button
    private Button setBtn, cancelBtn;
    // 定义广播Action
    private static final String BC_ACTION = "com.amaker.ch08.app.action.BC_ACTION";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前布局视图
        setContentView(R.layout.main);

        // 实例化Button
        setBtn = (Button) findViewById(R.id.Button01);
        cancelBtn = (Button) findViewById(R.id.Button02);

        // 获得AlarmManager实例
        final AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);

        // 实例化Intent
        Intent intent = new Intent();
        // 设置Intent action属性
        intent.setAction(BC_ACTION);
        intent.putExtra("msg", "你该去开会啦！");
        // 实例化PendingIntent
        final PendingIntent pi = PendingIntent.getBroadcast(MainActivity.this, 0,
                intent, 0);
        // 获得系统时间
        final long time = System.currentTimeMillis();

        // 设置按钮单击事件
        setBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
            // 重复提示，从当前时间开始，间隔 5 秒
            am.setRepeating(AlarmManager.RTC_WAKEUP, time,
                    8 * 1000, pi);
        }
    });

    // 设置按钮单击事件
    cancelBtn.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            am.cancel(pi);
        }
    });
    }
}
```

② 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在其中添加两个按钮：一个用于设置闹钟；一个用于取消闹钟。

```
<?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:text="设置闹钟"
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>
<Button
    android:text="取消闹钟"
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>
</LinearLayout>
```

③ 创建一个 MyReceiver 类继承 BroadcastReceiver，在 onReceive()方法中显示提示信息。

```
package com.amaker.ch08.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
/**
 *
 * @author 郭宏志
 *
 */
public class MyReceiver extends BroadcastReceiver {
```

```
    @Override
    public void onReceive(Context context, Intent intent) {
        // 获得提示信息
        String msg = intent.getStringExtra("msg");
        // 显示提示信息
        Toast.makeText(context, msg, Toast.LENGTH_LONG).show();
    }
}
```

④ 在 AndroidManifest.xml 配置文件中注册 MyReceiver。

```
package com.amaker.ch08.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;
/**
 *
 * @author 郭宏志
 *
 */
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // 获得提示信息
        String msg = intent.getStringExtra("msg");
        // 显示提示信息
        Toast.makeText(context, msg, Toast.LENGTH_LONG). show();
    }
}
```

程序运行结果如图 8.5 所示。

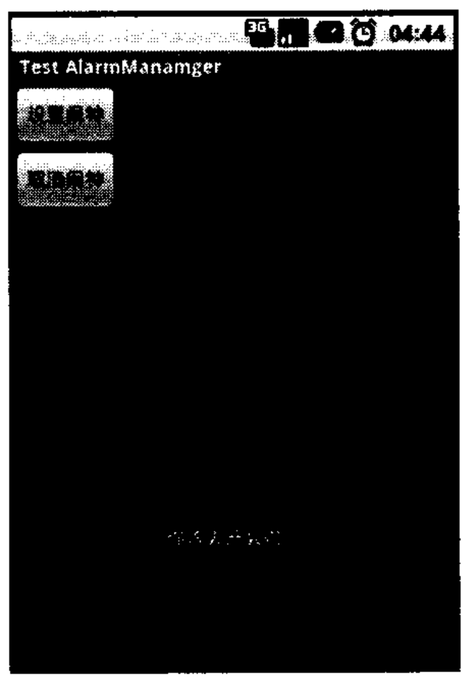
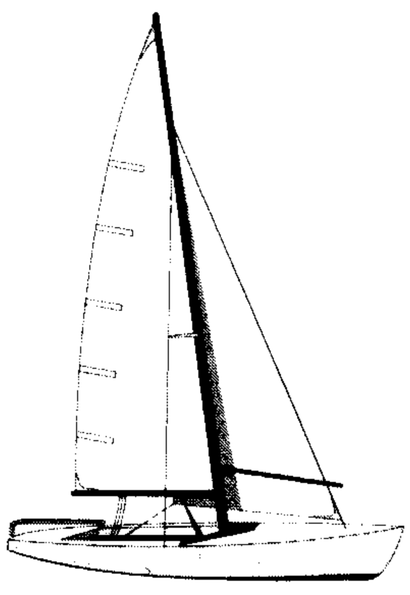


图 8.5　测试 AlarmManager



第 9 章　Android 中的数据存取

程序是数据的输入、处理和输出的过程，不管是操作系统还是应用程序都不可避免要用到大量的数据。由于内存容量有限，且不能长期保存数据，因此一般把它们以文件的形式保存在磁盘等持久的存储介质中。操作系统一般是以文件的形式来保存数据的，而应用程序一般以数据库来保存数据。

在手机这种特殊设备里，也经常会存取一些数据。例如，音频文件、视频文件、图片文件和通讯录等数据。Android 作为一种手机操作系统，提供了如下几种存取数据的方式：Preference（配置）、File（文件）、SQLite 数据和网络，可以根据程序的实际需要选择合适的存取方式。在本章里我们将详细讲述各种存取方式的使用。

另外，在 Android 中各个应用程序组件之间是相互独立的，彼此的数据不能共享。为了实现数据的共享，Android 提供了 Content Provider 组件来实现应用程序之间数据的共享。

9.1 Preference

Preference 提供了一种轻量级的数据存取方法，应用场合主要是数据比较少的配置信息。它以“键-值”（是一个 Map）对的方式将数据保存在一个 XML 配置文件中。例如，我们设置了一个手机开机问候语，可以将其以 Preference 方式来进行配置。

9.1.1 Preference 简介

使用 Preference 方式来存取数据，用到了 SharedPreferences 接口和 SharedPreferences 的一个内部接口 SharedPreferences.Editor，这两个接口在 android.content 包中。

调用 Context. getSharedPreferences (String name, int mode)方法得到 SharedPreferences 接口。该方法的第一个参数是文件名称，第二个参数是操作模式。操作模式有三种：MODE_PRIVATE（私有）、MODE_WORLD_READABLE（可读）和 MODE_WORLD_

WRITEABLE（可写）。

SharedPreferences 提供了获得数据的方法，如 getString(String key, String defValue)、getInt(String key, int defValue) 等。调用 SharedPreferences 的 edit() 方法返回 SharedPreferences.Editor 内部接口，该接口中提供了保存数据的方法，如 putString(String key, String value)、putInt(String key, int value)等，调用该接口的 commit()方法可以将数据进行保存。SharedPreferences 和 SharedPreferences.Editor 的常用方法如表 9.1 和表 9.2 所示。

表 9.1　SharedPreferences 常用方法

方 法 名 称	方 法 描 述
edit()	返回 SharedPreferences 的内部接口 SharedPreferences.Editor
contains(String key)	判断是否包含该键值
getAll()	返回所有配置信息 Map
getBoolean(String key, boolean defValue)	获得一个 boolean 值
getFloat(String key, float defValue)	获得一个 float 值
getInt(String key, int defValue)	获得一个 int 值
getLong(String key, long defValue)	获得一个 long 值
getString(String key, String defValue)	获得一个 String 值

表 9.2　SharedPreferences.Editor 常用方法

方 法 名 称	方 法 描 述
clear()	清除所有值
commit()	保存
getAll()	返回所有配置信息 Map
putBoolean(String key, boolean value)	保存一个 boolean 值
putFloat(String key, float value)	保存一个 float 值
putInt(String key, int value)	保存一个 int 值
putLong(String key, long value)	保存一个 long 值
putString(String key, String value)	保存一个 String 值
remove(String key)	删除该键对应的值

9.1.2　Preference 应用实例——保存临时短信

我们经常会有这样的经历，当我们在发送短信时，突然来了一个电话。我们停止发短信，去打电话。那么，当我们打完电话回来，我们发了一半的短信内容应该还在才对。这里我们通过 Preference 来保存临时短信内容。

创建一个 Activity，在其中放置一个 EditText 保存短信内容，放置一个 Button 用于发送短信。在 onCreate()中通过 getSharedPreferences()方法获得 SharedPreferences 接口，调用接口的 getString()方法，获得保存内容，将内容设置到 EditText 中。

在 onStop()方法中保存内容，使用 getSharedPreferences().edit()方法获得 Shared Preferences.Editor 接口，调用 SharedPreferences.Editor 的 putString()方法保存短信内容，调用 commit()方法提交内容。

```
package com.amaker.test;

import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
   // 声明 EditText 实例
   private EditText myEditText;
   // 声明 Button 对象
   private Button b1;
   //
   private static final String TEMP_SMS="temp_sms";
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置 Activity 界面布局
      setContentView(R.layout.main);
      // 通过 findViewById 方法实例化 EditText
      myEditText = (EditText)findViewById(R.id.EditText01);
      // 通过 findViewById 方法实例化 Button
      b1 = (Button)findViewById(R.id.Button01);

      // 获得 SharedPreferences 实例
      SharedPreferences pre = getSharedPreferences(TEMP_SMS, MODE_WORLD_READABLE);
      // 从 SharedPreferences 中获得短信内容
      String content = pre.getString("sms_content", "");
      // 在 EditText 中显示短信内容
      myEditText.setText(content);

   }

   @Override
   protected void onStop() {
      super.onStop();
   // 获得编辑器
   SharedPreferences.Editor editor =
   getSharedPreferences(TEMP_SMS, MODE_WORLD_WRITEABLE).edit();
   // 将 EditText 中的文本内容添加到编辑器
   editor.putString("sms_content", myEditText.getText().toString());
   // 提交编辑器内容
   editor.commit();
   }
}
```

当我们编辑了一些内容，退出屏幕再返回时，发现原来的内容显示出来了。其实，它

是把要保存的内容作为 XML 文件保存到了/data/data/package/shared_prefs/下面了。如图 9.1 所示是生成的 XML 文件。

程序运行结果如图 9.2 所示。

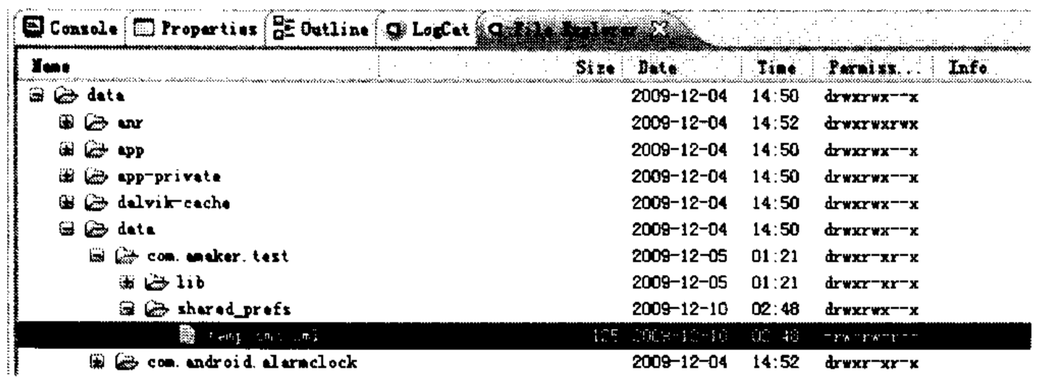


图 9.1　Preference 测试

图 9.2　Preference 测试

9.2 File

我们可以将一些数据直接以文件的形式保存在设备中。例如，一些文本文件、PDF 文件、音视频文件和图片等。Android 提供了文件读写的方法。

通过 Context.openFileInput()方法获得标准 Java 文件输入流（FileInputStream），通过 Context.openFileOutput()方法获得标准 Java 文件输出流（FileOutputStream）。使用 Resources.openRawResource (R.raw.myDataFile)方法返回 InputStream。

下面通过实例来演示 File 读写文件。创建一个 Activity，在其中放置两个 EditText 和两个 Button。第一个 EditText 和 Button 将 EditText 的内容写到文件名为 temp.txt 的文件中。第二个 EditText 和 Button 将内容从文件 temp.txt 中读出来显示。文件被默认保存在/data/data/package/files 下面。如图 9.3 所示是该文本文件。

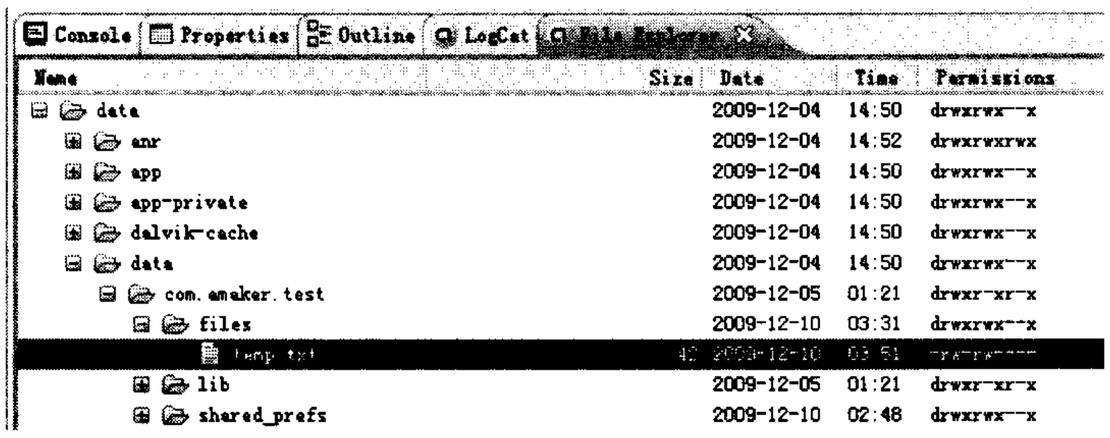


图 9.3　File 测试

```
package com.amaker.test;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import android.app.Activity;
import android.os.Bundle;
```

```
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity {
    // 文件名称字符串常量
    private static final String FILE_NAME="temp.txt";
    // 声明 Button 实例
    private Button b1,b2;
    // 声明 EditText 实例
    private EditText et1,et2;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 的界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 对象
        b1 = (Button)findViewById(R.id.Button01);
        b2 = (Button)findViewById(R.id.Button02);
        // 通过 findViewById 方法获得 EditText 对象
        et1 = (EditText)findViewById(R.id.EditText01);
        et2 = (EditText)findViewById(R.id.EditText02);
        // 为按钮设置单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 写内容
                write(et1.getText().toString());
            }
        });
        // 为按钮添加单击事件监听器
        b2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 读内容
                et2.setText(read());
            }
        });
    }
    // 写文件方法
    private String read(){
        try {
            // 实例化文件输入流对象
            FileInputStream fis = openFileInput(FILE_NAME);
            // 定义缓存数组
            byte[] buffer = new byte[fis.available()];
            // 读到缓冲区
            fis.read(buffer);
            return new String(buffer);
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
        return null;
    }
    // 写方法
    private void write(String content){
        try {
            // 实例化文件输出流
            FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
            // 写内容
            fos.write(content.getBytes());
            // 关闭文件流
            fos.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

图 9.4　File 测试

程序运行结果如图 9.4 所示。

9.3 SQLite

Android 中通过 SQLite 数据库引擎来实现结构化数据存储。SQLite 是一个嵌入式数据库引擎，针对内存等资源有限的设备（如手机、PDA、MP3）提供的一种高效的数据库引擎。

SQLite 数据库不像其他的数据库（如 Oracle），它没有服务器进程。所有的内容包含在同一个单文件中。该文件是跨平台的，可以自由复制。基于其自身的先天优势，SQLite 在嵌入式领域得到了广泛应用。Android 也没有重复发明“轮子”，而是直接使用了 SQLite 数据库。

9.3.1 SQLiteDatabase

Android 提供了创建和使用 SQLite 数据库的 API。SQLiteDatabase 代表一个数据库对象，提供了操作数据库的一些方法，另外还有一个 SQLiteOpenHelper 工具类提供了更简洁的功能。在 Android 的 SDK 目录下有 sqlite3 工具，我们可以利用它来创建数据库、创建表和执行一些 SQL 语句。下面我们先来看看 SQLiteDatabase 的常用方法（如表 9.3 所示）。

表 9.3　SQLiteDatabase 常用方法

方法名称	方法描述
openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)	打开或者创建数据库
insert(String table, String nullColumnHack, ContentValues values)	添加一条记录
delete(String table, String whereClause, String[] whereArgs)	删除一条记录
query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	查询记录

续表

方法名称	方法描述
update(String table, ContentValues values, String whereClause, String[] whereArgs)	修改记录
execSQL(String sql)	执行一条 SQL 语句
close()	关闭数据库

1. 打开或者创建数据库

可以使用 SQLiteDatabase 的静态方法 openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)打开或者创建一个数据库。该方法的第一个参数是数据库创建路径，注意这个路径一定是数据库全路径。例如：/data/data/paackage/databases/dbname.db。第二个参数是指定返回一个 Cursor 子类的工厂，如果没有指定（null）则使用默认工厂。

下面的代码创建了一个 temp.db 数据库。

```
SQLiteDatabase.openOrCreateDatabase("/data/data/com.amaker.test/databases/temp.db
", null);
```

2. 创建表

创建一张表很简单。首先，编写创建表的 SQL 语句，然后，调用 SQLiteDatabase 的 execSQL()方法便可以创建一张表了。

下面的代码创建了一张用户表，属性列为：_id（主键并且自动增加）、username（用户名称）、password（密码）。

```
private void createTable(SQLiteDatabase db){
      // 创建表 SQL 语句
      String sql = " create table UserTbl(_id integer primary key autoincrement,
username text,password text)";
      // 执行 SQL 语句
db.execSQL(sql);
}
```

3. 插入数据

插入数据有两种方法：一种是调用 SQLiteDatabase 的 insert(String table, String nullColumnHack, ContentValues values)方法，该方法的第一个参数是表名称，第二个参数是空列的默认值，第三个参数是 ContentValues 类型的一个封装了列名称和列值的 Map；另一种方法是编写插入数据的 SQL 语句、直接调用 SQLiteDatabase 的 execSQL()方法来执行。

下面的代码演示了如何插入一条记录到数据库。

```
// 插入数据
private void insert(SQLiteDatabase db){
// 插入数据 SQL 语句
      String sql = " insert into UserTbl(username,password) values('amaker','123') ";
   // 执行 SQL 语句
      db.execSQL(sql);
```

```
 }
 // 插入数据 2
private void insert2(SQLiteDatabase db){
        // 实例化常量值
   ContentValues cv =new ContentValues();
        // 添加用户名
   cv.put("username","ghz");
        // 添加密码
   cv.put("password", "456");
        // 插入
   db.insert("UserTbl", null, cv);
}
```

4. 删除数据

和插入数据类似，删除数据也有两种方法：一种是调用 SQLiteDatabase 的 delete(String table, String whereClause, String[] whereArgs)方法，该方法的第一个参数是表名称，第二个参数是删除条件，第三个参数是删除条件值数组；另一种方法是编写删除 SQL 语句，调用 SQLiteDatabase 的 execSQL()方法来执行删除。

下面的代码演示了如何删除记录。

```
  // 删除
private void delete(SQLiteDatabase db){
  // 删除 SQL 语句
    String sql = " delete from UserTbl where _id=2 ";
 // 执行 SQL 语句
    db.execSQL(sql);
}
// 删除 2
private void delete2(SQLiteDatabase db){
 // 删除条件
String whereClause = "_id=?";
// 删除条件参数
String[] whereArgs = {String.valueOf(1)};
// 执行删除
    db.delete("UserTbl", whereClause, whereArgs);
}
```

5. 查询数据

查询数据相对比较复杂，因为查询可能会带有很多条件。不过熟悉 SQL 查询的读者都知道这只不过是把查询 SQL 封装成方法而已。下面看一个查询方法。

```
public Cursor query (String table, String[] columns, String selection, String[]
selectionArgs, String groupBy, String having, String orderBy, String limit)。
```

各个参数的意义说明如下。

- table：表名称
- colums：列名称数组

- selection：条件子句，相当于 where
- selectionArgs：条件子句，参数数组
- groupBy：分组列
- having：分组条件
- orderBy：排序类
- limit：分页查询限制
- Cursor：返回值，相当于结果集 ResultSet

Cursor 是一个游标接口，提供了遍历查询结果的方法，如移动指针方法 move()、获得列值方法 getString()等。Cursor 各个方法的详细介绍如表 9.4 所示。

表 9.4　Cursor 游标常用方法

方法名称	方法描述
getCount()	总记录条数
isFirst()	判断是否第一条记录
isLast()	判断是否最后一条记录
moveToFirst()	移动到第一条记录
moveToLast()	移动到最后一条记录
move(int offset)	移动到指定记录
moveToNext()	移动到下一条记录
moveToPrevious()	移动到上一条记录
getColumnIndexOrThrow(String columnName)	根据列名称获得列索引
getInt(int columnIndex)	获得指定列索引的 int 类型值
getString(int columnIndex)	获得指定列索引的 String 类型值

下面的代码演示了如何查询数据记录。

```
    // 查询
private void query(SQLiteDatabase db){
 // 查询获得游标
      Cursor c = db.query("UserTbl", null, null, null, null, null, null);
    // 判断游标是否为空
      if(c.moveToFirst()){
       // 遍历游标
         for (int i = 0; i < c.getCount(); i++) {
             c.move(i);
             // 获得 ID
             int id = c.getInt(0);
             // 获得用户名
             String username = c.getString(1);
             // 获得密码
             String password = c.getString(2);
             // 输出用户信息
             System.out.println(id+":"+username+":"+password);
```

```
        }
    }
}
```

6. 修改数据

和添加删除相同，修改数据有两种方式。一是调用 SQLiteDatabase 的 update(String table, ContentValues values, String whereClause, String[] whereArgs)方法。该方法的第一个参数是表名称，第二个参数是更新列 ContentValues 类型的键-值对（Map），第三个参数是更新条件（where 子句），第四个参数是更新条件值数组。二是编写更新的 SQL 语句，调用 SQLiteDatabase 的 execSQL 执行更新。

下面的代码演示了如何更新数据记录。

```
    // 修改
private void update(SQLiteDatabase db){
    // 修改 SQL 语句
        String sql = " update UserTbl set password=888 where _id=1 ";
    // 执行 SQL
        db.execSQL(sql);
    }
    // 修改 2
    private void update2(SQLiteDatabase db){
     // 实例化内容值
        ContentValues values = new ContentValues();
     // 在 values 中添加内容
        values.put("password", "999");
    // 修改条件
        String whereClause = "_id=?";
    // 修改添加参数
        String[] whereArgs = {String.valueOf(1)};
    // 修改
        db.update("UserTbl", values, whereClause, whereArgs);
    }
```

9.3.2 SQLiteOpenHelper

SQLiteOpenHelper 是 SQLiteDatabase 的一个帮助类，用来管理数据库的创建和版本更新。一般的用法是定义一个类继承之，并实现其两个抽象方法 onCreate(SQLiteDatabase db)和 onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)来创建和更新数据库。如表 9.5 所示是 SQLiteOpenHelper 常用方法。

表 9.5　SQLiteOpenHelper 常用方法

方法名称	方法描述
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	构造方法，一般是传递一个要创建的数据库名称 name 参数
onCreate(SQLiteDatabase db)	创建数据库时调用

续表

方法名称	方法描述
onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	版本更新时调用
getReadableDatabase()	创建或打开一个只读数据库
getWritableDatabase()	创建或打开一个读写数据库

下面的代码演示了通过继承 SQLiteOpenHelper 来实现创建数据库、创建表、插入数据和查询数据的过程。

```
 // 修改
package com.amaker.test;
import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.os.Bundle;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置Activity界面布局
        setContentView(R.layout.main);
       // 实例化数据库帮助类
        MyDBHelper helper = new MyDBHelper(this);
       // 插入
        helper.insert();
       // 查询
        helper.query();
    }
    // 数据库帮助类
    class MyDBHelper extends SQLiteOpenHelper{
    // 创建表SQL语句
    private static final String CREATE_TABLE_SQL = " create table TempTbl(_id integer,
name text) ";
    // SQLiteDatabase实例
        private SQLiteDatabase db;
    // 构造方法
        MyDBHelper(Context c){
            super(c,"test.db", null, 2);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
            // 创建表
            db.execSQL(CREATE_TABLE_SQL);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
    }
    // 插入方法
    private void insert(){
        // 插入 SQL 语句
        String sql = "insert into TempTbl(_id,name)values(1,'amaker') ";
        // 执行插入
        getWritableDatabase().execSQL(sql);
    }
    // 查询方法
    private void query(){
        // 查询获得游标
        Cursor c  = getWritableDatabase().query("TempTbl", null, null, null, null,
null, null);
        // 判断游标是否为空
        if(c.moveToFirst()){
            // 遍历游标
            for (int i = 0; i < c.getCount(); i++) {
                c.move(i);
                int id = c.getInt(0);
                String name = c.getString(1);
                System.out.println(id+":"+name);
            }
        }
    }
  }
}
```

9.3.3　SQLite 应用实例——收藏管理

在当今这个移动互联网时代，大家不可避免地要通过手机上网。那么，很多网站如何进行收藏和管理呢？在这里可以通过使用 SQLite 对自己比较喜欢的网站进行收藏管理。收藏信息包括编号、网站名称、URL 和网站内容描述，可以实现的功能有添加、删除和浏览等。

要完成该实例需要如下步骤。

① 创建操作数据库的工具类 DBHelper，该类继承 SQLiteOpenHelper。覆盖 onCreate()和 onUpgrade()方法，并添加 insert()、del()和 query()方法，分别用来添加、删除和查询数据。代码如下所示。

```
package com.amaker.test;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
// 数据库帮助类
public class DBHelper extends SQLiteOpenHelper {
```

```
    // 数据库名称
    private static final String DB_NAME = "coll.db";
    // 表名称
    private static final String TBL_NAME = "CollTbl";
    // 创建表 SQL 语句
    private static final String CREATE_TBL = " create table "
            + " CollTbl(_id integer primary key autoincrement,name text,url text, desc
text) ";
    // SQLiteDatabase 实例
    private SQLiteDatabase db;
    // 构造方法
    DBHelper(Context c) {
        super(c, DB_NAME, null, 2);
    }
    @Override // 创建表
    public void onCreate(SQLiteDatabase db) {
        this.db = db;
        db.execSQL(CREATE_TBL);
    }
    // 插入方法
    public void insert(ContentValues values) {
        // 获得 SQLiteDatabase 实例
        SQLiteDatabase db = getWritableDatabase();
        // 插入
        db.insert(TBL_NAME, null, values);
        db.close();
    }
    // 查询方法
    public Cursor query() {
        // 获得 SQLiteDatabase 实例
        SQLiteDatabase db = getWritableDatabase();
        // 查询获得 Cursor
        Cursor c = db.query(TBL_NAME, null, null, null, null, null, null);
        return c;
    }
    // 删除方法
    public void del(int id) {
        if (db == null)
        // 获得 SQLiteDatabase
            db = getWritableDatabase();
        // 执行删除
        db.delete(TBL_NAME, "_id=?", new String[] { String.valueOf(id) });
    }
    // 关闭数据库
    public void close() {
        if (db != null)
            db.close();
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
}
```

② 创建视图布局文件 add.xml，在其中添加三个 TextView 和三个 EditText，分别用来输入网站名称、URL 和网站描述信息，一个 Button 响应单击事件，当用户单击时将信息保存到数据库。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" android:layout_gravity="center_vertical">

   <TextView
   android:text="网站名称"
   android:id="@+id/TextView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TextView>

   <EditText
   android:text=""
   android:id="@+id/EditTextName"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"></EditText>

   <TextView
   android:text="URL" android:id="@+id/TextView02"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TextView>

   <EditText
   android:text=""
   android:id="@+id/EditTextUrl"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"></EditText>


   <TextView
   android:text="网站描述"
   android:id="@+id/TextView03"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TextView>

   <EditText
   android:text=""
   android:id="@+id/EditTextDesc"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:height="100px"></EditText>

   <Button
   android:text="添加"
   android:id="@+id/ButtonAdd"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></Button>
```

```
</LinearLayout>
```

收藏管理添加界面如图 9.5 所示。

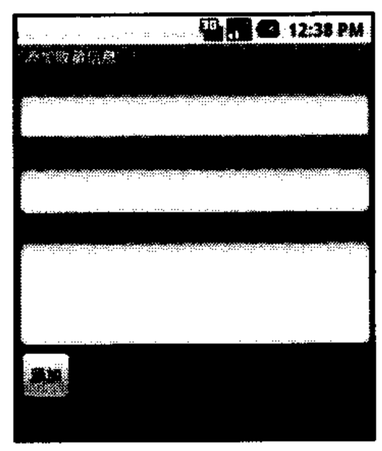


图 9.5 收藏管理添加界面

③ 创建 Activity AddActivity。在该类中声明用到的组件类 EditText 和 Button，并在 onCreate()方法中进行初始化。响应 Button 的单击事件方法，该方法从编辑框中获得数据项，封装到 ContentValues 对象中，调用 DBHelpter 进行保存。之后跳转到显示收藏记录 QueryActivity 界面。代码如下所示。

```
package com.amaker.test;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class AddActivity extends Activity {
    // 声明 EditText 实例
    private EditText et1, et2, et3;
    // 声明 Button 实例
    private Button b1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.add);
        // 设置当前 Activity 标题
        this.setTitle("添加收藏信息");
        // 通过 findViewById 方法获得 EditText 实例
        et1 = (EditText) findViewById(R.id.EditTextName);
        et2 = (EditText) findViewById(R.id.EditTextUrl);
        et3 = (EditText) findViewById(R.id.EditTextDesc);
        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button) findViewById(R.id.ButtonAdd);
        // 为 Button 添加单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 姓名
                String name = et1.getText().toString();
                // url
                String url = et2.getText().toString();
                // 描述
                String desc = et3.getText().toString();
                // 内容值实例
```

```
            ContentValues values = new ContentValues();
            // 在 values 中添加信息
            values.put("name", name);
            values.put("url", url);
            values.put("desc", desc);
            // 实例化数据库帮助类
            DBHelper helper = new DBHelper(getApplicationContext());
            // 插入数据
            helper.insert(values);
            // 实例化 Intent
            Intent intent = new Intent(AddActivity.this,
                    QueryActivity.class);
            // 启动 Activity
            startActivity(intent);
        }
    });

    }
}
```

④ 创建显示记录 Activity QueryActivity。该类继承 ListAcitivity，将从数据库查询的记录通过 ListView 显示在列表中。并且当用户单击某条记录时，通过对话框提示用户是否删除此记录。这里的难点是如何将从数据库查询的 Cursor 构建成 SimpleCursorAdapter，将该数据绑定到 LisView。要把 Cursor 中的每一条记录展现成 ListView 中的每一项，这里我们需要将表中的每一列绑定到特定的 View 上面，我们创建一个 row.xml 布局文件，在其中放置四个 TextView 组件，用来和表中的_id、name、url 和 desc 对应。这样我们就可以将 Cursor 中的一条记录展现成为 ListView 中的一项了。

添加 ListView 组件的选项单击事件，当用户单击某一项时弹出对话框，提示用户是否删除该条记录。

row.xml 布局文件代码如下所示。

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

   <TextView
   android:id="@+id/text0"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:paddingRight="10px" />

   <TextView
android:id="@+id/text1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:paddingRight="10px"/>
```

```
    <TextView
    android:id="@+id/text2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingRight="10px" />

    <TextView
    android:id="@+id/text3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="10px" />

</LinearLayout>
```

QueryActivity 代码如下所示。

```
package com.amaker.test;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.DialogInterface;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.AdapterView.OnItemClickListener;
// 继承 ListActivity
public class QueryActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 标题
        this.setTitle("浏览收藏信息");
        // 实例化数据库帮助类
        final DBHelper helpter = new DBHelper(this);
        // 查询获得游标
        Cursor c = helpter.query();
        // 列表项数组
        String[] from = { "_id", "name", "url", "desc" };
        // 列表项 ID
        int[] to = { R.id.text0, R.id.text1, R.id.text2, R.id.text3 };
        // 适配器
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                R.layout.row, c, from, to);
        // 列表视图
        ListView listView = getListView();
        // 为列表视图添加适配器
        listView.setAdapter(adapter);

        // 提示对话框
        final AlertDialog.Builder builder = new AlertDialog.Builder(this);
        // 为 ListView 设置单击监听器
```

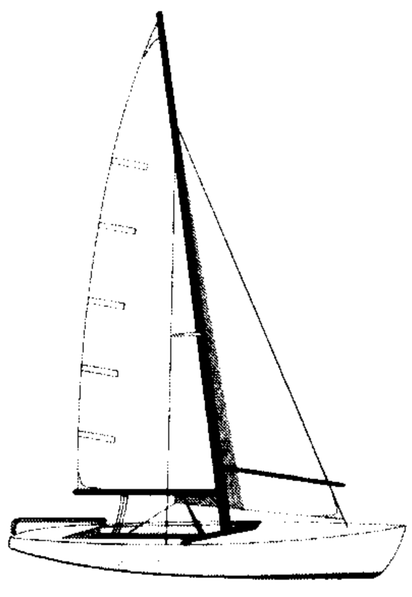
```
        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                    long arg3) {
                final long temp = arg3;
                builder.setMessage("真的要删除该记录吗？").setPositiveButton("是",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int which) {
                                // 删除数据
                                helpter.del((int)temp);
                                // 重新查询
                                Cursor c = helpter.query();
                                String[] from = { "_id", "name", "url", "desc" };
                                int[] to = { R.id.text0, R.id.text1, R.id.text2,
                                R.id.text3 };

                                SimpleCursorAdapter adapter =
new SimpleCursorAdapter(getApplicationContext(),s
                                        R.layout.row, c, from, to);
                                ListView listView = getListView();
                                listView.setAdapter(adapter);
                            }
                        }).setNegativeButton("否",
                        new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog,
                                    int which) {

                            }
                        });
                // 创建提示对话框
                AlertDialog ad = builder.create();
                // 显示提示对话框
                ad.show();
            }
        });
        helpter.close();
    }
}
```

程序运行结果如图 9.6 所示。

图 9.6　收藏管理界面



第 10 章　Content Provider

在前面的章节中我们讲述过，在 Android 中的应用程序组件有四种，它们分别是：Activity、Service、Broadcast Receiver 和 Content Provider。前三个我们都分别做了阐述，在这一章里我们将详细讲述有关 Content Provider 的有关内容。

在 Android 中，应用程序之间是相互独立的，分别运行在自己的进程中。如果应用程序之间相互共享数据又该如何实现呢？例如，当我们要发送一条短信时，可能要用到联系人应用程序，从中选择要发送的人。在这种情况下，Android 提供了应用程序之相互访问的统一接口，这些接口被定义在 Content Provider 中，其中包括添加、删除、修改和查询等操作。

10.1 Content Provider 简介

Content Provider 用来保存和检索数据，并且使应用程序之间相互访问数据成为可能，它是跨应用程序共享数据的唯一方法。

Android 为常用的数据类型（如：音视频、图片和联系方式等）提供了大量的 Content Provider。它们被定义在 android.provider 包下面。通过这样定义好的 Content Provider 我们可以方便地进行数据操作。当然我们必须拥有适当的权限。

我们也可以自己来定义 Content Provider 共享我们的数据，方便用户的访问。

10.1.1 Content Provider 的常用方法

ContentProvider 定义在 android.content 包下面。我们定义一个 ContentProvider 必须实现几个抽象方法，这些抽象方法是：

- query（Uri, String[], String, String[], String）查询
- insert（Uri, ContentValues）插入
- update（Uri, ContentValues, String, String[]）更新

- delete（Uri, String, String[]）删除
- getType（Uri）获得 MIME 数据类型

有关 ContentProvider 的常用方法如表 10.1 所示。

表 10.1　ContentProvider 常用方法

方 法 名 称	方 法 描 述
insert(Uri, ContentValues)	插入
delete(Uri, String, String[])	删除
update(Uri, ContentValues, String, String[])	修改
query(Uri, String[], String, String[], String)	查询
getType(Uri)	获得 MIME 数据类型
onCreate()	当 ContentProvider 创建时调用
getContext()	获得 Context 对象

10.1.2　ContentResolver

我们是在 ContentProvider 中实现我们实际操作数据的方法的。但是，客户端调用时，我们用到了另外的一个接口，它就是 ContentResolver。ContentResolver 中提供和 ContentProvider 中对应的方法。我们是间接地通过操作 ContentResolver 来操作 ContentProvider 的。

ContentResolver 通过应用程序的 getContentResolver()方法获得。一般情况下，ContentProvider 是单实例的，但是可以有多个 ContentResolver 在不同的应用程序和不同的进程之间和 ContentProvider 交互。

10.1.3　URI

ContentProvider 又是如何共享其数据的呢？这里就要知道另外的一个概念，这便是 URI。ContentProvider 就是通过该对象来共享其数据的。

一个 URI 对象必须以“content://”开头，接下来是 URI 的授权部分，这部分内容要和 AndroidManifest.xml 配置文件中声明的授权内容一致，后面还可能有数据类型和记录 ID。

通过 URI 可以使得 ContentResolver 知道和哪个 ContentProvider 对应，并且来操作哪些表以及哪些记录。

完整的一个 URI 如图 10.1 所示。

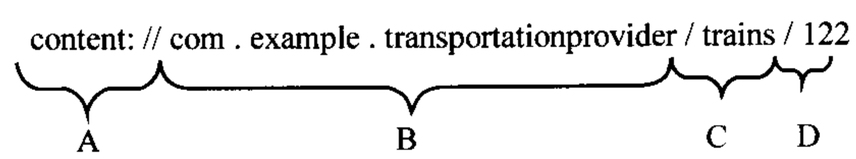


图 10.1　URI 格式

10.1.4 查询系统 ContentProvider 内容

在 Android SDK API 中提供了大量的数据 ContentProvider。这些内容在 android.content 包中。要实现一个对现有系统 ContentProvider 的查询需要如下步骤。

① 通过对应 getContentResolver()方法，获得 ContentResolver 对象。

② 获得 ContentProvider 的 URI 标示。

③ 列出想要查询的列。

④ 调用 ContentResolver 的 query 方法执行查询。

如下代码实现了对系统联系人的有条件查询。

```
private void query() {
    // 获得 ContentResolver 实例
    ContentResolver cr = getContentResolver();
    // 定义 Uri
    Uri uri = Contacts.People.CONTENT_URI;
    // 定义查询类数组
    String[] projection = { Contacts.PeopleColumns.NAME,
            Contacts.PeopleColumns.NOTES };
    // 查询条件
    String selection = Contacts.PeopleColumns.NAME + "=?";
    // 查询条件参数
    String[] selectionArgs = { "Amaker" };
    // 排序列
    String sortOrder = Contacts.PeopleColumns.NAME;
    // 查询获得游标
    Cursor c = cr.query(uri, projection, selection, selectionArgs,
            sortOrder);
    // 判断游标是否为空
    if (c.moveToFirst()) {
        // 遍历游标
        for (int i = 0; i < c.getCount(); i++) {
            c.moveToPosition(i);
            // 获得列索引
            int idx = c.getColumnIndexOrThrow(PeopleColumns.NAME);
            // 输出日志，列值
            Log.i("Test", c.getString(idx));
        }
    }
}
```

程序运行结果如图 10.2 所示。

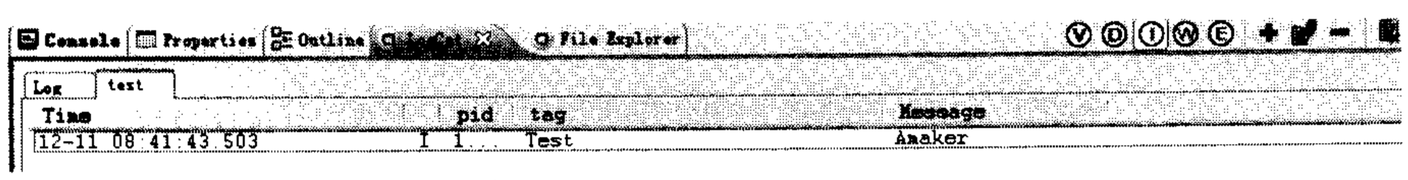


图 10.2 查询 ContentProvider

10.1.5　添加系统 ContentProvider 内容

除了查询，我们还可以对现有的 ContentProvider 执行添加操作，在 Android 中提供了添加联系人的程序。在这里我们自己写一个程序来执行添加操作。

执行添加操作需要如下步骤。

① 通过对应的 getContentResolver()方法获得 ContentResolver 对象。

② 获得 ContentProvider 的 URI 标示。

③ 将添加的信息封装到 ContentValues 对象中。

④ 调用 ContentResolver 对象的 insert 方法执行添加。

如下代码添加了联系人姓名和备注列信息。

```
private void insert(){
    // 获得 ContentResolver 实例
    ContentResolver cr = getContentResolver();
    // 实例化 ContentValues
    ContentValues values = new ContentValues();
    // 定义 Uri
    Uri url = Contacts.People.CONTENT_URI;
    // 添加姓名
    values.put(People.NAME, "tom");
    // 添加备注
    values.put(People.NOTES, "I'm tom!");
    // 插入数据
    cr.insert(url, values);
}
```

运行程序，我们打开联系人就会发现多了一个联系人信息，结果如图 10.3 所示。

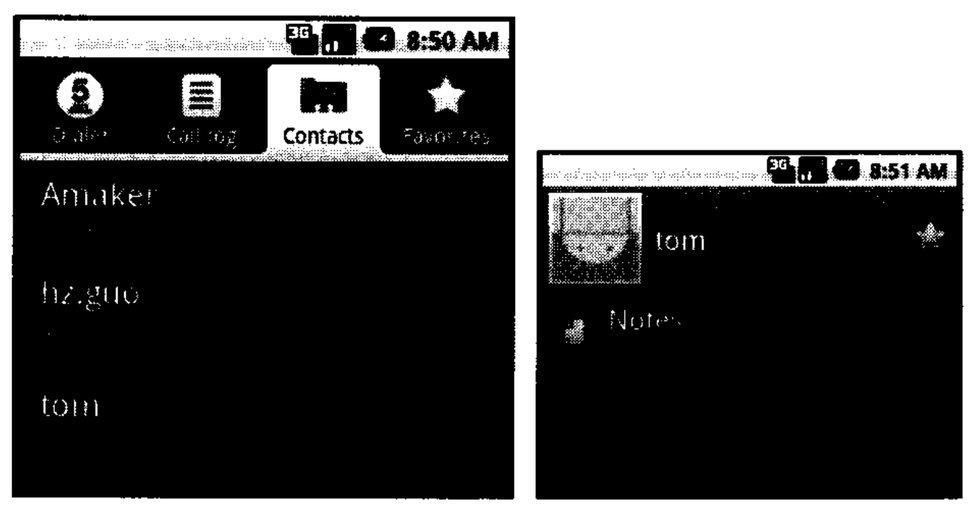


图 10.3　添加 ContentProvider

10.1.6　添加系统 ContentProvider 图片内容

在 Android.content 中提供了 MediaStoreContentProvider 类，该类用来实现一些多媒体文件的操作，例如，图片音频、视频文件等。

如下代码实现了添加一张图片的功能。

```
        private void insert2(){
        // 实例化 ContentValues
        ContentValues values = new ContentValues(3);
        // 添加图片名称、描述和类型
        values.put(Media.DISPLAY_NAME, "Girl");
        values.put(Media.DESCRIPTION, "Beauty");
        values.put(Media.MIME_TYPE, "image/jpeg");
        // 插入数据
        Uri uri = getContentResolver().insert(Media.EXTERNAL_CONTENT_URI, values);
            // 获得 Bitmap 实例
            Bitmap  sourceBitmap  =  BitmapFactory.decodeFile("/data/data/com.amaker.
test/files/girl.jpg");
            // 定义输出流
            OutputStream outStream;
            try {
                // 获得输出流
                outStream = getContentResolver().openOutputStream
(uri);
                // 压缩输出流
    sourceBitmap.compress(Bitmap.CompressFormat.JPEG,        50,
outStream);
                outStream.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
    }
```

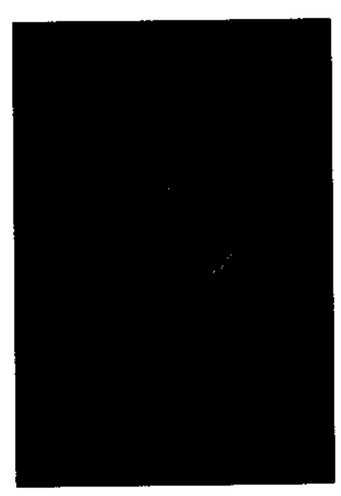


图 10.4　添加图片 ContentProvider

程序运行结果如图 10.4 所示。

10.2 自定义 ContentProvider

在 10.1 节中我们操作的是 Android 系统自带的核心应用程序中的 ContentProvider 对象，一个是 Contact（通讯录），另一个是 MediaStore（媒体存储）。当然，我们也可以定义自己的 ContentProvider 对象来共享数据。本节将详细讲述如何定义并使用 ContentProvider。

10.2.1　创建 ContentProvider 的步骤

要使用 ContentProvider 来操作数据，必须要有保存数据的场所。可以使用文件或 SQLite 数据库的方式来保存数据，通常使用 SQLite 数据库。

为了访问数据，就要提供访问数据的接口。这里需要基础 ContentProvider 类，实现其中访问数据的抽象方法，这些方法有：

- query()：查询
- insert()：插入
- update()：更新

- delete()：删除
- getType()：获得类型
- onCreate()：创建时调用

另外，定义好的 ContentProvider 必须在 AndroidManifest.xml 配置文件中声明才能使用。声明中必须添加的参数是授权属性“android:authorities”。

综上所述，创建 ContentProvider 的步骤说明如下。

① 创建保存数据的文件或数据库。

② 定义一个类继承 ContentProvider，实现抽象方法。

③ 将定义好的 ContentProvider 在 AndroidManif.xml 配置文件中声明，以便使用。

10.2.2　ContentProvider 实例

本节将通过一个实例来演示如何创建和使用 ContentProvider。该实例演示的是员工信息维护，包括添加、删除、修改和查询员工信息。为了重点讲解 ContentProvider 的使用，这里员工信息比较简单，只定义编号、姓名、性别和年龄。

为了完成本实例需要如下步骤。

① 创建一个 Android 工程，工程的名称为“Chapter10_ContentProvider_01_Test02”。

② 在“com.amaker.ch10.app”包中创建一个 MainActivity 类。

```
package com.amaker.ch10.app;

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

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

③ 创建一个常量类，该类主要是用来声明一些常量，如列名称、排序列、URI 和内容类型等。该类有一个内部类 Employee 实现 BaseColumns 接口，BaseColumns 接口是一个常量接口，里面有一个“_id”和“_count”字符串常量，分别用来标示记录 ID 和记录总数。

```
package com.amaker.ch10.app;

import android.net.Uri;
import android.provider.BaseColumns;
/**
 * @author 郭宏志
 * 通讯录常量类
```

```
 */
public final class Employees {
    // 授权常量
    public static final String AUTHORITY = "com.amaker.provider.Employees";
    private Employees() {}
    // 内部类
    public static final class Employee implements BaseColumns {
        // 构造方法
        private Employee() {}
        // 访问 Uri
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
"/employee");
        // 内容类型
        public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.amaker.
employees";
        public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.
amaker.employees";

        // 默认排序常量
        public static final String DEFAULT_SORT_ORDER = "name DESC";// 按姓名排序
        // 表字段常量
        public static final String NAME = "name";                    // 姓名
        public static final String GENDER= "gender";                 // 性别
        public static final String AGE = "age";                      // 年龄
    }
}
```

④ 创建一个数据库帮助类，该类继承 SQLiteOpenHelper 类，在其中声明了数据库名称、数据库版本和表名称一些常量，该类有三种方法：DBHelper 方法用来创建数据库；onCreate 方法用来创建表；onUpgrade 方法是当数据库版本改变时更新表。

```
package com.amaker.ch10.app;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import com.amaker.ch10.app.Employees.Employee;
/**
 * @author 郭宏志
 * 数据库工具类
 */
public class DBHelper extends SQLiteOpenHelper{
    // 数据库名称常量
    private static final String DATABASE_NAME = "Employees.db";
    // 数据库版本常量
    private static final int DATABASE_VERSION = 1;
    // 表名称常量
    public static final String EMPLOYEES_TABLE_NAME = "employee";
    // 构造方法
    public DBHelper(Context context) {
        // 创建数据库
```

```
        super(context, DATABASE_NAME,null, DATABASE_VERSION);
    }

    // 创建时调用
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + EMPLOYEES_TABLE_NAME + " ("
                + Employee._ID + " INTEGER PRIMARY KEY,"
                + Employee.NAME + " TEXT,"
                + Employee.GENDER + " TEXT,"
                + Employee.AGE + " INTEGER"
                + ");"));
    }

    // 版本更新时调用
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // 删除表
        db.execSQL("DROP TABLE IF EXISTS employee");
        onCreate(db);
    }
}
```

⑤ 创建一个 EmployeeProvider 类，该类继承 ContentProvider，实现其抽象方法来操作数据库。该类引用 DBHelper 类来获得数据库实例，引用 UriMatcher 工具类来对 URI 进行匹配比较，引用 HashMap 保存查询列，在 onCreate 方法中实例化 DBHelper 获得数据库实例，在 insert 方法中实现插入数据，在 delete 方法中实现删除数据，在 query 方法中实现查询数据，在 update 方法中实现更新数据。

```
package com.amaker.ch10.app;

import java.util.HashMap;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

import com.amaker.ch10.app.Employees.Employee;

public class EmployeeProvider extends ContentProvider{
    // 数据库帮助类
    private DBHelper dbHelper;
    // Uri 工具类
    private static final UriMatcher sUriMatcher;
    // 查询、更新条件
    private static final int EMPLOYEE = 1;
    private static final int EMPLOYEE_ID = 2;
    // 查询列集合
```

```
    private static HashMap<String, String> empProjectionMap;
    static {
    // Uri 匹配工具类
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(Employees.AUTHORITY, "employee", EMPLOYEE);
        sUriMatcher.addURI(Employees.AUTHORITY, "employee/#", EMPLOYEE_ID);
        // 实例化查询列集合
        empProjectionMap = new HashMap<String, String>();
        // 添加查询列
        empProjectionMap.put(Employee._ID, Employee._ID);
        empProjectionMap.put(Employee.NAME, Employee.NAME);
        empProjectionMap.put(Employee.GENDER, Employee.GENDER);
        empProjectionMap.put(Employee.AGE, Employee.AGE);
    }

    // 创建是调用
    public boolean onCreate() {
        // 实例化数据库帮助类
        dbHelper = new DBHelper(getContext());
        return true;
    }
    // 添加方法
    public Uri insert(Uri uri, ContentValues values) {
        // 获得数据库实例
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        // 插入数据，返回行 ID
        long rowId = db.insert(DBHelper.EMPLOYEES_TABLE_NAME, Employee.NAME, values);
        // 如果插入成功返回 uri
        if (rowId > 0) {
            Uri empUri = ContentUris.withAppendedId(Employee.CONTENT_URI, rowId);
            getContext().getContentResolver().notifyChange(empUri, null);
            return empUri;
        }
        return null;
    }
    // 删除方法
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // 获得数据库实例
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        // 获得数据库实例
        int count;
        switch (sUriMatcher.match(uri)) {
        // 根据指定条件删除
        case EMPLOYEE:
            count = db.delete(DBHelper.EMPLOYEES_TABLE_NAME, selection, selectionArgs);
            break;
        // 根据指定条件和 ID 删除
        case EMPLOYEE_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.delete(DBHelper.EMPLOYEES_TABLE_NAME, Employee._ID + "=" +
noteId
                + (!TextUtils.isEmpty(selection) ? " AND (" + selection + ')' : ""),
```

```
selectionArgs);
            break;

        default:
            throw new IllegalArgumentException("错误的 URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    // 获得类型
    public String getType(Uri uri) {
        return null;
    }

    // 查询方法
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
            switch (sUriMatcher.match(uri)) {
            // 查询所有
            case EMPLOYEE:
                qb.setTables(DBHelper.EMPLOYEES_TABLE_NAME);
                qb.setProjectionMap(empProjectionMap);
                break;
            // 根据 ID 查询
            case EMPLOYEE_ID:
                qb.setTables(DBHelper.EMPLOYEES_TABLE_NAME);
                qb.setProjectionMap(empProjectionMap);
                qb.appendWhere(Employee._ID + "=" + uri.getPathSegments().get(1));
                break;
            default:
                throw new IllegalArgumentException("Uri 错误！ " + uri);
            }

            // 使用默认排序
            String orderBy;
            if (TextUtils.isEmpty(sortOrder)) {
                orderBy = Employee.DEFAULT_SORT_ORDER;
            } else {
                orderBy = sortOrder;
            }

            // 获得数据库实例
            SQLiteDatabase db = dbHelper.getReadableDatabase();
            // 返回游标集合
            Cursor c = qb.query(db, projection, selection, selectionArgs, null, null,
orderBy);
            c.setNotificationUri(getContext().getContentResolver(), uri);
            return c;
    }
```

```
    // 更新方法
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {
    // 获得数据库实例
        SQLiteDatabase db = dbHelper.getWritableDatabase();
        int count;
        switch (sUriMatcher.match(uri)) {
        // 根据指定条件更新
        case EMPLOYEE:
            count  =  db.update(DBHelper.EMPLOYEES_TABLE_NAME,  values,  selection,
selectionArgs);
            break;
        // 根据指定条件和 ID 更新
        case EMPLOYEE_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.update(DBHelper.EMPLOYEES_TABLE_NAME, values, Employee._ID + "="
+ noteId
                    + (!TextUtils.isEmpty(selection) ? " AND (" + selection + ')' : ""),
selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("错误的 URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
}
```

⑥ ContentProvider 必须声明才能使用，因此，需要在 AndroidManifest.xml 配置文件中声明。声明很简单，需要类名称和授权名称两个属性。

```
        <provider android:name="EmployeeProvider"
        android:authorities="com.amaker.provider.Employees"/>}
```

⑦ 测试，在 MainActivity 中创建添加、删除、更新和查询方法。在 onCreate 方法中调用测试。

```
package com.amaker.ch10.app;

import android.app.Activity;
import android.content.ContentUris;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

import com.amaker.ch10.app.Employees.Employee;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 添加
        insert();
        // 查询
        query();
        // 更新
        update();
        // 查询
        query();
        // 删除
        del();
        // 查询
        query();
    }
    // 删除方法
    private void del(){
    // 删除 ID 为 1 的记录
    Uri uri = ContentUris.withAppendedId(Employee.CONTENT_URI, 1);
    // 获得 ContentResolver，并删除
    getContentResolver().delete(uri, null, null);
    }
    // 更新
    private void update(){
    // 更新 ID 为 1 的记录
    Uri uri = ContentUris.withAppendedId(Employee.CONTENT_URI, 1);
    ContentValues values = new ContentValues();
    // 添加员工信息
    values.put(Employee.NAME, "hz.guo");
    values.put(Employee.GENDER, "male");
    values.put(Employee.AGE,31);
    // 获得 ContentResolver，并更新
        getContentResolver().update(uri, values, null, null);
    }
    // 查询
    private void query(){
    // 查询列数组
       String[] PROJECTION = new String[] {
                  Employee._ID,        // 0
                  Employee.NAME,       // 1
                  Employee.GENDER,     // 2
                  Employee.AGE         // 3
        };
    // 查询所有备忘录信息
        Cursor c = managedQuery(Employee.CONTENT_URI, PROJECTION, null,
                null, Employee.DEFAULT_SORT_ORDER);
        // 判断游标是否为空
        if (c.moveToFirst()) {
            // 遍历游标
            for (int i = 0; i < c.getCount(); i++) {
                c.moveToPosition(i);
                // 获得姓名
```

```
                String name = c.getString(1);
                String gender = c.getString(2);
                int age = c.getInt(3);
                // 输出日志
                Log.i("emp", name+":"+gender+":"+age);
            }
        }
    }
    // 插入
    private void insert(){
    // 声明 Uri
    Uri uri = Employee.CONTENT_URI;
    // 实例化 ContentValues
    ContentValues values = new ContentValues();
    // 添加员工信息
    values.put(Employee.NAME, "amaker");
    values.put(Employee.GENDER, "male");
    values.put(Employee.AGE,30);
    // 获得 ContentResolver，并插入
    getContentResolver().insert(uri, values);
    }
}
```

程序运行结果如图 10.5 所示。

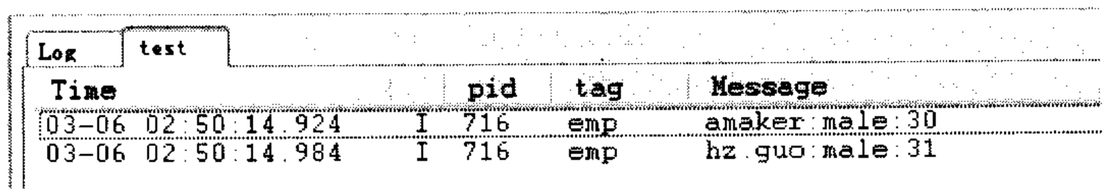
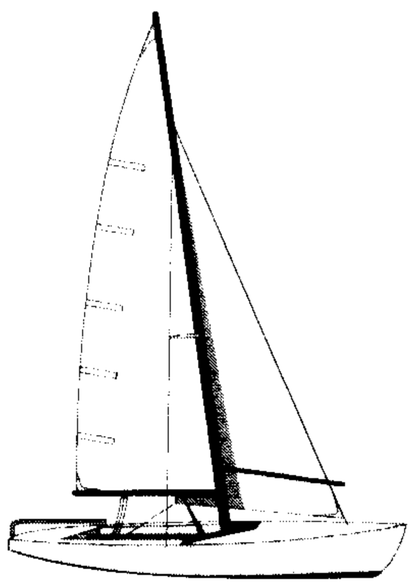


图 10.5　ContentProvider 实例



第 11 章　Android 中的多媒体应用

Android 提供了常见媒体的编码、解码机制，因此可以非常容易地集成音频、视频和图片等多媒体文件到应用程序中。我们可以调用 Android 提供的现有 API，非常容易地实现相册、播放器、录音和摄像等应用程序，当然，有些需要硬件的支持。通过我们前面学习的 Activity 和 Intent，可以非常直接地来访问这些媒体文件。

播放音频及视频我们会用到 MediaPlayer 和 JetPlayer 类，录制音频及视频我们会用到 MediaRecorder 类。文件的来源可以是本地文件系统文件、存储在 Android 中的资源文件，还可以是通过网络的文件流。

Android 对常用媒体格式提供了支持，支持的图片格式有 JPEG、GIF、PNG 和 BMP，支持的音频格式有 3GP、MP3 和 WAV，支持的视频格式有 3GP、MP4 等。

11.1 音频及视频播放

音频及视频的播放我们会用到 MediaPlayer 类。该类提供了播放、暂停、停止和重复播放等方法。该类位于 android.media 包中，详细信息请参考 API 文档。

我们播放的文件来源很多，可以是本地文件系统中的文件、资源中的源文件，还可以是网络中的文件流等。

11.1.1　从源文件中播放

使用 MediaPlayer 类播放音频及视频，最简单的一种方式就是从源文件中播放。要实现该功能需要如下步骤。

① 在项目的 res/raw 文件夹下面放置一个 Android 支持的媒体文件，例如一个 MP3 文件。

② 创建一个 MediaPlayer 实例，可以使用 MediaPlayer 的静态方法 create 来完成。

③ 调用 start()方法开始播放，调用 pause()方法暂停播放，调用 stop()方法停止播放。如果你希望重复播放，在调用 start()方法之前，必须调用 reset()和 prepare()方法。

程序代码如下所示。

```
 // 实例化 MediaPlayer
MediaPlayer player = MediaPlayer.create(this, R.raw.test);
// 开始播放
 player.start();
```

11.1.2 从文件系统中播放

从文件系统中播放也是比较常见的一种方式。例如，在我们的 SDCard 中有一些音频及视频文件，那么我们就可以直接进行播放。

从文件系统中播放需要如下步骤。

① 实例化一个 MediaPlayer。

② 调用 setDataSource()方法来设置想要播放文件的路径。

③ 首先调用 prepare()，然后调用 start()来进行播放。

程序代码如下所示。

```
// 实例化 MediaPlayer
MediaPlayer mp = new MediaPlayer();
// 播放路径
   String path = "/sdcard/test.mp3";
    // 为 MediaPlayer 设置数据源
   mp.setDataSource(path);
   // 准备
   mp.prepare();
   // 播放
   mp.start();
```

11.1.3 从网络中播放

随着 3G 的不断完善和推广，资费不断降低，直接利用网络资源已经不是什么问题。移动互联网时代已经到了。这里我们就看一看如何通过网络来播放音视频文件。

要实现该功能有两种方法。

方法一

① 创建网络 URI 实例。

② 创建一个 MediaPlayer 实例，可以使用 MediaPlayer 的静态方法 create，通过传递 URI 参数来完成。

③ 调用 start()方法开始播放。

程序代码如下所示。

```
// 播放路径
String path = "http://ms1.m.mop.com/2008/08/27/012/12198150127399164.mp3";
// 将字符串 Uri 解析为 Uri 实例
Uri uri = Uri.parse(path);
 // 实例化 MediaPlayer
MediaPlayer player = MediaPlayer.create(this, uri);
// 播放
player.start();
```

方法二

① 实例化一个 MediaPlayer。

② 调用 setDataSource()方法来设置想要播放文件的路径，该路径是网络上的可用路径。

③ 首先调用 prepare()，然后调用 start()来进行播放。

程序代码如下所示。

```
// 实例化 MediaPlayer
MediaPlayer mp = new MediaPlayer();
// 播放路径
String path = "http://ms1.m.mop.com/2008/08/27/012/12198150127399164.mp3";
try {
    // 设置数据源
    mp.setDataSource(path);
    // 准备
    mp.prepare();
   // 播放
   mp.start();
} catch (IllegalArgumentException e) {
    e.printStackTrace();
} catch (IllegalStateException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

11.2 迷你音乐播放器

通过上面的学习我们已经对媒体播放，以及 MediaPlayer 类有一些了解。在这一节我们要讲述一个具有控制界面的音乐播放器。

该程序需要的步骤说明如下。

① 在工程的 res/drawable 目录下放置三个图片文件，用来设置控制按钮的图片。

② 在布局文件 main.xml 中添加四个 View 组件，其中 TextView 用来显示标题，三个图片按钮用来控制音乐的播放、暂停和停止。

③ 创建 Activity 类 MainActivity 并实现 MediaPlayer.OnCompletionListener 接口，当音乐播放完成时进行控制。

④ 声明三个 ImageButton 实例。

⑤ 创建 loadClip()方法，实例化 MediaPlayer，并添加事件监听器 OnCompletion Listener()。

⑥ 在 onCreate()方法中实例化。

⑦ 创建 play()方法，调用 MediaPlayer 的 play()方法播放音乐，同时使播放按钮失效，使暂停按钮和停止按钮生效。

⑧ 创建 pause()方法，调用 MediaPlayer 的 pause()方法暂停播放音乐，同时使暂停按钮失效，使播放按钮和停止按钮生效。

⑨ 创建 stop()方法，调用 MediaPlayer 的 stop()方法停止播放音乐，调用 MediaPlayer 的 prepare()方法，然后调用 MediaPlayer 的 seekTo()方法定位到播放音乐的开始。同时使播放按钮生效，暂停和停止按钮失效。

⑩ 创建 error()方法捕获异常。

⑪ 实现监听接口的 onCompletion()方法，调用 stop()方法停止音乐播放。

⑫ 创建 setup()方法，调用 loadClip()，使播放按钮生效，使停止按钮和暂停按钮失效。

⑬ 为播放按钮添加单击事件，调用 play()方法播放音乐。

⑭ 为暂停按钮添加单击事件，调用 pause()方法暂停播放。

⑮ 为停止按钮添加单击事件，调用 stop()方法停止播放。

布局文件 main.xml 的代码如下所示。

```
Str <?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="迷你音乐播放器" />

   <LinearLayout
   android:orientation="horizontal" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

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

   <ImageButton
   android:id="@+id/pause"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/pause"
```

```
    />

    <ImageButton
    android:id="@+id/stop"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/stop"
    />
    </LinearLayout>
</LinearLayout>
```

MainActivity 代码如下所示。

```
package com.amaker.test;

import android.app.Activity;
import android.app.AlertDialog;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

public class MainActivity extends Activity implements MediaPlayer.OnCompletion
Listener {
    // 声明图片按钮 ImageButton
    private ImageButton play,pause,stop;
    // 声明 MediaPlayer 实例
    private MediaPlayer mp;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
setContentView(R.layout.main);
        // 通过 findViewById 方法获得按钮实例
        play = (ImageButton)findViewById(R.id.play);
        pause = (ImageButton)findViewById(R.id.pause);
        stop = (ImageButton)findViewById(R.id.stop);
        // 为按钮添加单击事件监听器
        play.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                // 播放
                play();
            }
        });
        //为按钮添加单击事件监听器
        pause.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                // 暂停
                pause();
            }
        });
        // 为按钮添加单击事件监听器
        stop.setOnClickListener(new View.OnClickListener() {
```

```
            public void onClick(View view) {
                // 停止
                stop();
            }
        });
         // 准备
        setup();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (stop.isEnabled()) {
            // 停止
            stop();
        }
    }

    public void onCompletion(MediaPlayer mp) {
        // 停止
        stop();
    }
    // 播放
    private void play() {
        // 播放
        mp.start();
        // 使播放按钮失效
        play.setEnabled(false);
         // 使暂停按钮生效
        pause.setEnabled(true);
        // 使停止按钮生效
        stop.setEnabled(true);
    }
    // 停止
    private void stop() {
        // 停止
        mp.stop();
        // 使暂停按钮失效
        pause.setEnabled(false);
        // 是停止按钮失效
        stop.setEnabled(false);

        try {
            // 准备
            mp.prepare();
            // 定位到开始
            mp.seekTo(0);
            // 是播放按钮生效
            play.setEnabled(true);
        }
        catch (Throwable t) {
            error(t);
        }
    }
```

```
    // 暂停
    private void pause() {
        // 暂停
        mp.pause();
        // 使播放按钮生效
        play.setEnabled(true);
        // 是暂停按钮失效
        pause.setEnabled(false);
        // 是停止按钮生效
        stop.setEnabled(true);
    }
    // 加载播放文件
    private void loadClip() {
        try {
            // 实例化 MediaPlayer
            mp=MediaPlayer.create(this, R.raw.test);
            // 设置监听器
            mp.setOnCompletionListener(this);
        }
        catch (Throwable t) {
            error(t);
        }
    }

    // 准备
    private void setup() {
        // 加载播放文件
        loadClip();
        // 使播放按钮生效
        play.setEnabled(true);
        // 是暂停按钮失效
        pause.setEnabled(false);
        // 是停止按钮失效
        stop.setEnabled(false);
    }
    // 出错提示
    private void error(Throwable t) {
        // 获得对话框构造器
        AlertDialog.Builder builder=new AlertDialog.Builder(this);
        Builder
            // 设置标题
            .setTitle("报错啦!")
            // 设置提示信息
            .setMessage(t.toString())
            // 确定按钮
            .setPositiveButton("确定", null)
            // 显示
            .show();
    }
}
```

程序运行结果如图 11.1 所示。

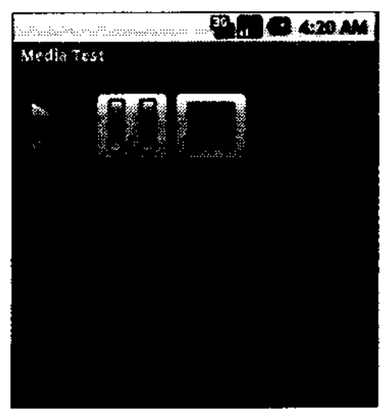


图 11.1　迷你音乐播放器

11.3 迷你视频播放器

借助 VideoView 和 MediaController 组件可以很方便地实现一个视频播放器。要实现该功能需要的步骤说明如下。

① 在文件系统中放置一个 Android 支持的视频文件，例如一个 MP4 文件。

② 在布局管理文件中添加一个 VideoView 组件。

③ 创建一个 Activity，声明 VideoView 和 MediaController 组件。

④ 在 onCreate()方法中实例化这两个对象。

⑤ 创建文件对象指向 MP4 文件。

⑥ 为 VideoView 设置播放路径。

⑦ 建立 VideoView 和 MediaController 组件之间的关系。

布局文件 main.xml 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"
      >
   <VideoView
       android:id="@+id/video"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
      />
</LinearLayout>
```

Activity 代码如下所示。

```
package com.amaker.test;

import java.io.File;
import android.app.Activity;
import android.graphics.PixelFormat;
import android.os.Bundle;
import android.widget.MediaController;
import android.widget.VideoView;

public class VideoDemo extends Activity {
   // 声明视频视图 VideoView
   private VideoView video;
   // 声明媒体控制组件 MediaController
   private MediaController ctlr;

   @Override
   public void onCreate(Bundle icicle) {
      super.onCreate(icicle);
```

```
        // 设置窗口特征
        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 实例化 File
        File clip=new File("/sdcard/test.mp4");
        // 判断文件是否存在
        if (clip.exists()) {
            // 通过 findViewById 方法获得 VideoView 实例
            video=(VideoView)findViewById(R.id.video);
            // 设置视频播放路径
            video.setVideoPath(clip.getAbsolutePath());
            // 实例化 MediaController
            ctlr=new MediaController(this);
            // MediaController 和 MediaPlayer 相互关联
            ctlr.setMediaPlayer(video);
            // MediaController 和 MediaPlayer 相互关联
            video.setMediaController(ctlr);
            // 使 VideoView 获得焦点
            video.requestFocus();
        }
    }
}
```

程序运行结果如图 11.2 所示。

图 11.2　迷你视频播放器

11.4 音视频的录制

现在大部分的手机都提供了音频录制的功能，Android 系统也提供了对音频及视频录制的支持，当然这也需要有硬件的支持。要想录制音频及视频需要通过 MediaRecorder 来完成。

MediaRecorder 类位于 android.media 包中，提供了相关方法。常用的方法有：start()开始录制、pause()暂停录制、stop()停止录制、prepare()准备录制、reset()重新开始、release()释放资源等。

要实现音频及视频的录制需要的步骤说明如下。

① 在工程的 res/drawable 目录下放置两个图片文件，用来设置控制按钮的图片。

② 在布局文件 main.xml 中添加三个 View 组件，其中 TextView 用来显示标题，两个图片按钮用来控制开始录制和停止录制。

③ 创建 Activity 类 MainActivity。

④ 声明两个 ImageButton 实例。

⑤ 创建 loadClip()方法，实例化 MediaRecorder。

⑥ 在 onCreate()方法中实例化。

⑦ 创建 record()方法，调用 MediaRecorder 的 start()方法录制，同时使录制按钮失效，使停止按钮生效。

⑧ 创建 stop()方法，调用 MediaRecorder 的 stop()方法停止录制。

⑨ 创建 error()方法捕获异常。

⑩ 创建 setProperty()方法，设置 MediaRecorder 的属性，包括音频输入源（麦克风）、音频输出格式和文件保存路径等。

⑪ 创建 setup()方法，调用 loadClip()，使录制按钮生效，使停止按钮失效。

⑫ 为录制按钮添加单击事件，调用 play()方法录制。

⑬ 为停止按钮添加单击事件，调用 stop()方法停止录制。

布局文件 main.xml 的代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android: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="迷你录音器" />

   <LinearLayout
   android:orientation="horizontal" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <ImageButton
   android:id="@+id/record"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/record"
   />

   <ImageButton
   android:id="@+id/stop"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
    android:src="@drawable/stop"
    />
    </LinearLayout>
</LinearLayout>
```

Activity 代码如下所示。

```
package com.amaker.test;

import java.io.File;
import java.io.IOException;

import android.app.Activity;
import android.app.AlertDialog;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

public class MainActivity extends Activity {
    // 声明图片按钮 ImageButton
    private ImageButton record, stop;
    // 声明 MediaRecorder
    private MediaRecorder mr;
    // 声明路径
    private String path;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 ImageButton 实例
        record = (ImageButton) findViewById(R.id.record);
        // 通过 findViewById 方法获得 ImageButton 实例
        stop = (ImageButton) findViewById(R.id.stop);
        // 为录制按钮设置单击事件监听器
        record.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                // 录制
                record();
            }
        });
        // 为停止按钮设置单击事件监听器
        stop.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                // 停止录制
                stop();
            }
        });
        // 准备
        setup();
```

```
    }
    // 设置路径
    private void setPath() throws IOException {
        // 路径为 sd 扩展卡
        path = "/sdcard/test1.mp3";
        // 获得 sd 扩展卡状态
        String state = android.os.Environment.getExternalStorageState();
        // 判断是否安装扩展卡
        if (!state.equals(android.os.Environment.MEDIA_MOUNTED)) {
            throw new IOException("SD 没有安装. 它的状态是" + state
                    + ".");
        }
        // 获得文件父目录
        File directory = new File(path).getParentFile();
        if (!directory.exists() && !directory.mkdirs()) {
            throw new IOException("文件不能被创建.");
        }

    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (stop.isEnabled()) {
            // 停止录制
            stop();
        }
    }
    // 录制方法
    private void record() {
        try {
            // 准备
            mr.prepare();
        } catch (IllegalStateException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 开始录制
        mr.start();
        // 是录制按钮失效
        record.setEnabled(false);
        // 是停止按钮生效
        stop.setEnabled(true);
    }
    // 停止
    private void stop() {
        // 停止录制
        mr.stop();
        // 使停止按钮失效
        stop.setEnabled(false);
        // 使录制按钮生效
        record.setEnabled(true);
```

```
        // 释放资源
        mr.release();
    }
    // 设置属性
    private void setProperty() {
        try {
            // 实例化 MediaRecorder
            mr = new MediaRecorder();
            // 设置音频源
            mr.setAudioSource(MediaRecorder.AudioSource.MIC);
            // 设置输出格式
            mr.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
            // 设置音频编码器
            mr.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
            try {
                // 设置路径
                setPath();
            } catch (IOException e) {
                e.printStackTrace();
            }
            // 设置输出文件
            mr.setOutputFile(path);
        } catch (Throwable t) {
            error(t);
        }
    }
    // 准备
    private void setup() {
        // 设置属性
        setProperty();
        // 是录制按钮生效
        record.setEnabled(true);
        // 是停止按钮失效
        stop.setEnabled(false);
    }
    // 错误提示
    private void error(Throwable t) {
        // 对话框
        AlertDialog.Builder builder = new AlertDialog.Builder
(this);
        builder.setTitle("报错啦!").setMessage(t.toString()).
setPositiveButton(
                "确定", null).show();
    }
}
```

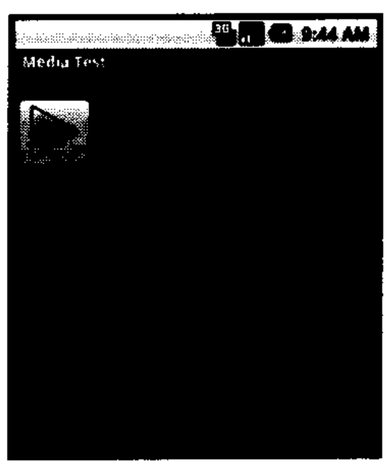


图 11.3　迷你录音器

程序运行结果如图 11.3 所示。

11.5 Camera 照相

手机拍照已经不是什么新鲜事儿，Android 提供了手机拍照功能 API。当然，这还需要硬件的支持。要使用 Android 系统进行拍照，使用到的类有很多。具体介绍如下：

- SurfaceView：该类是一个视图组件，实现了照片的预览功能。
- SurfaceHolder：一个抽象接口，是 SurfaceView 的支持类，一般通过 SurfaceView 的 getHolder()方法获得。允许你控制界面的尺寸和格式、编辑界面的像素，以及监控界面尺寸的改变。
- SurfaceHolder.Callback：SurfaceHolder 的一个内部接口，可以实现该接口获得界面改变的信息。该接口中有三个方法，surfaceCreated()方法在界面创建时被调用，一般在该方法中打开照相机并设置预览。surfaceChanged()方法在界面尺寸改变时被调用，一般在该方法中设置相机的参数（大小、格式等）。surfaceDestroyed()方法在界面被销毁时调用，在该方法中清除相机实例，释放资源。
- Camera：相机类，实现拍照功能。
- Camera.PictureCallback：Camera 的一个内部接口，处理照片准备好后的回调。

通过上面类的介绍，我们可以实现一个拍照程序了。下面是程序的实现步骤。

① 创建一个 Android 工程“Chapter11_Camera”，在该工程的“com.amaker.ch11.app”包中，创建一个 MainActivity 类，在该类的顶部声明 SurfaceView、SurfaceHolder 和 Camera 对象。

```
package com.amaker.ch11.app;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

import android.app.Activity;
import android.graphics.PixelFormat;
import android.hardware.Camera;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Environment;
import android.view.KeyEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MainActivity extends Activity {
    // 声明拍照界面组件 SurfaceView
    private SurfaceView surfaceView;
    // 声明界面控制组件 SurfaceHolder
    private SurfaceHolder surfaceHolder;
    // 声明照相机
    private Camera camera;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
```

```
    };
}
```

② 在该工程的 res\layout\目录下创建一个布局文件 main.xml，在其中添加一个 SurfaceView 视图组件。

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

③ 在 onCreate()方法中获得 SurfaceView 和 SurfaceHolder 实例对象。

```
// 实例化拍照界面组件
surfaceView = (SurfaceView)findViewById(R.id.preview);
// 从 SurfaceView 中获得 SurfaceHolder
surfaceHolder = surfaceView.getHolder();
```

④ 实现 SurfaceHolder.Callback 回调接口，实现该接口中的三个方法。

```
// SurfaceHodler 回调，处理打开相机，关闭相机以及照片尺寸的改变
   SurfaceHolder.Callback surfaceCallback = new SurfaceHolder.Callback() {
      @Override
      public void surfaceCreated(SurfaceHolder holder) {
         // 打开相机
         camera = Camera.open();
         try {
            // 设置预览
            camera.setPreviewDisplay(holder);
         } catch (IOException e) {
            e.printStackTrace();
         }
      }

      @Override
      public void surfaceChanged(SurfaceHolder holder, int format, int width,
            int height) {
         // 获得相机参数
         Camera.Parameters parameters = camera.getParameters();
         // 设置照片大小
         parameters.setPreviewSize(width, height);
         // 设置照片格式
         parameters.setPictureFormat(PixelFormat.JPEG);
         // 设置相机参数
         camera.setParameters(parameters);
         // 开始预览
         camera.startPreview();
      }
```

```
        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            // 停止预览
            camera.stopPreview();
            // 释放相机资源
            camera.release();
            camera = null;
        }
    };
```

⑤ 为 SurfaceHolder 添加回调，并设置其类型。

```
    // 为 SurfaceHolder 添加回调
    surfaceHolder.addCallback(surfaceCallback);
    surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

⑥ 创建一个保存照片任务类，该类的主要功能是将从硬件获得的字节流保存到设备文件中。

```
    class SavePictureTask extends AsyncTask<byte[],String,String>{
        @Override
        protected String doInBackground(byte[]... params) {
            // 创建文件
           File picture = new File(Environment.getExternalStorageDirectory(),"picture.
jpg");
            // 如果文件存在删除掉
            if(picture.exists())picture.delete();
            try {
                // 获得文件输出流
                FileOutputStream  fos = new FileOutputStream(picture.getPath());
                // 写到该文件
                fos.write(params[0]);
                // 关闭文件流
                fos.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }
    }
```

⑦ 实现 Camera.PictureCallback 回调方法，执行保存照片任务。

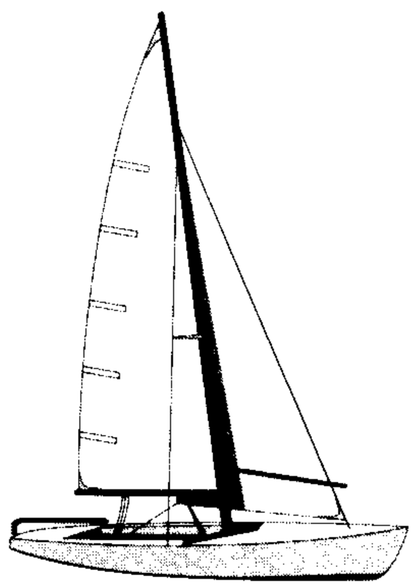
```
// 照片回调
    Camera.PictureCallback pictureCallback = new Camera.PictureCallback() {
        @Override
        public void onPictureTaken(byte[] data, Camera camera) {
            new SavePictureTask().execute(data);
            camera.startPreview();
        }
    };
```

⑧ 创建保存照片方法。

```
// 拍照方法
private void takePic(){
    // 停止预览
    camera.stopPreview();
    // 拍照
    camera.takePicture(null,null, pictureCallback);
}
```

⑨ 响应按键事件，当拍照键或搜索键被按下时进行拍照。

```
// 响应按键事件
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if(keyCode==KeyEvent.KEYCODE_CAMERA||keyCode==KeyEvent.KEYCODE_SEARCH){
        takePic();
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```



第 12 章　Android 中的图形图像

Android 系统的图形处理能力非常强大，对于 2D 图形的处理它并没有使用 Java 的图形处理类，而是自定义了一系列 2D 图形处理类，这些类分别位于 android.graphics、android.graphics.drawable.shapes 和 android.view.animation 包中。对于 3D 图形的处理，Android 集成了 OpenGL ES 1.0 提供的高效 3D 图形处理技术，这些类分别位于 javax.microedition.khronos.opengles 和 android.opengl 包中。

Android 系统中的图形处理基本可以分为两类。一类是静态图形处理，也就是这些图片不经常变化，例如，一些图标、Logo、动画等，针对这些图形的处理，一般是将图形作为资源文件添加到工程（res\drawable\、res\anim）当中，然后通过各种 Drawable 类来处理使用。另一类是动态图形处理，也就是这些图片要经常变化（包括大小、位置、形状等），需要不断地重新绘制，例如，游戏中的各种场景、人物和道具等。

本章的内容要点如下：

- 在 Android 中访问图片，包括使用图片文件创建 Drawable 对象，使用 XML 文件定义 Drawable 属性以及 Bitmap 和 BitmapFactory 的使用
- Tween 动画和 Frame 动画实现
- 动态图片绘制的基本思路，继承 View 类还是继承 SurfaceView 类
- 动态图形类 Canvas、Color、Paint 和 Path 简介
- 绘制几何图形
- 使用 Matrix 和 Shader 实现图形特效

12.1 在 Android 中访问图片

在 Android 系统中很多地方都在使用图片。例如，列表图标、快捷方式图标、图片按钮等。在 Android 中操作图片是通过使用 Drawable 类来完成的。Drawable 类有很多个子类，

如 BitmapDrawable 用来操作位图；ColorDrawable 用来操作颜色；ShapeDrawable 用来操作各种形状。

有三种方法来实例化 Drawable 对象：一是使用保存在工程中的一个图片文件；二是使用 XML 文件定义 Drawable 属性；三是构造方法实例化，这种方法在实际开发中一般用不到。

12.1.1　使用图片文件创建 Drawable 对象

最简单的一种方式是在工程的资源文件下面保存图片，该图片会自动在 R 类中创建引用，然后可以通过 R.drawable.my_image 使用该图片对象。下面我们看一个实例，实例步骤说明如下。

① 在工程的资源文件夹下放入一个 test.jpg 图片文件。

② 创建布局文件 main.xml 并在其中添加一个 ImageView 组件。

③ 创建 Activity，并实例化 ImageView 组件对象。

④ 调用 ImageView 的 setImageResource()方法，引用资源 id。

Activity 代码如下所示。

```
package com.amaker.test;

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

public class MainActivity extends Activity {
   // 声明图片视图 ImageView
   private ImageView myImageView;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
     // 设置当前 Activity 界面布局
      setContentView(R.layout.main);
     // 通过 findViewById 方法获得 ImageView
      myImageView = (ImageView)findViewById(R.id.ImageView01);
      // 为 ImageView 设置图片资源
      myImageView.setImageResource(R.drawable.test);
   }
```

布局文件 main.xml 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="Drawable Test" />
```

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

程序运行结果如图 12.1 所示。

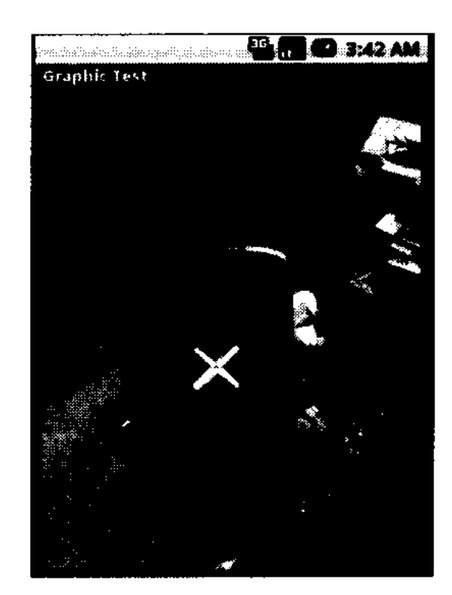


图 12.1　从资源文件加载图片

12.1.2　使用 XML 文件定义 Drawable 属性

我们也经常在布局文件中定义 Drawable 属性。例如，可以在布局文件中定义图片按钮的图片及应用程序的图标等。

下面的代码演示了如何在 AndroidManifest.xml 配置文件中引用资源图标。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.test"
     android:versionCode="1"
     android:versionName="1.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".MainActivity"
              android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

   </application>
   <uses-sdk android:minSdkVersion="3" />

</manifest>
```

在上一个例子中我们也可以通过这种方式来配置 ImageView 的图片资源。

```
<?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="Drawable Test" />

   <ImageView
      android:id="@+id/ImageView01"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:src="@drawable/test"
        ></ImageView>
</LinearLayout>
```

12.1.3　Bitmap 和 BitmapFactory

如果我们的图片文件保存在 SDCard 中又该如何呢？在这种情况下，我们可以通过 Bitmap 和 BitmapFactory 两个类来读取文件。下面的代码演示了如何从 SDCard 中读取图片文件并将其设置为壁纸。

程序步骤说明如下。

① 在 SDCard 中添加一个名称为 wallpaper.jgp 的图片文件。

② 创建 Activity。

③ 在 Activity 的 onCreate()方法中通过 BitmapFactory 的 decodeFile()方法传递文件路径，获得 Bitmap 对象。

④ 调用 setWallpaper()方法设置桌面。

程序代码如下所示。

```
package com.amaker.test;

import java.io.IOException;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置当前Activity界面布局
        setContentView(R.layout.main);
       // 图片路径
        String path="/sdcard/wallpaper.jpg";
       // 通过BitmapFactory获得Bitmap实例
        Bitmap bm = BitmapFactory.decodeFile(path);
        try {
            // 设置桌面
            setWallpaper(bm);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

图 12.2　Bitmap 和 BitmapFactory 的使用

程序运行后桌面被改变，如图 12.2 所示。

12.2 Android 中的动画

在 Android 系统中提供了两种动画实现方式：一种是 Tween 动画，这种实现方式可以使视图组件移动、放大、缩小以及产生透明度的变化；另一种是 Frame 动画，这是一种传统的动画方法，通过顺序播放排列好的图片来实现，类似电影。

12.2.1 Tween 动画

Tween 动画能完成一系列简单的变化（如位置、尺寸、透明度和旋转等）。例如，在你的程序中有一个 ImageView 组件，我们通过 Tween 动画可以使该视图实现放大、缩小、旋转、渐变等。Tween 动画类位于 android.view.animation 包中，该包中包含了一些常用的动画实现类。

- Animation：动画抽象类，其他几个实现类继承该类。
- ScaleAnimation：控制尺寸变化的动画类。
- AlphaAnimation：控制透明度变化的动画类。
- RotateAnimation：控制旋转变化的动画类。
- TranslateAnimation：控制位置变化的动画类。
- AnimationSet：定义动画属性集合类。
- AnimationUtils：动画工具类。

总体来讲，Android 系统 Tween 动画提供了四种实现方式，详细说明如表 12.1 所示。

Tween 动画的实现方式有两种：一种是直接通过硬编码的方式在程序代码中实现；另一种是在配置文件中定义。Android 系统推荐使用配置文件的方法，这种方式可扩展性较高。

首先，我们来看看硬编码方式是如何实现的。下面还是通过一个实例来演示各种动画的实现。在该实例中，在界面中添加一个 ImageView 组件和四个 Button 组件。ImageView 组件用于显示动画的图标。四个 Button 分别响应单击事件实现不同的动画效果。实例步骤说明如下。

① 创建一个 Android 工程“Chapter12_Graphic_Animation”。

② 在工程的 res\drawable\目录中添加一个 girl.png 图片资源。

表 12.1 Tween 动画实现类

名　称	实 现 类	常用构造方法	参 数 说 明
Alpha（渐变动画）	AlphaAnimation	AlphaAnimation(float fromAlpha, float toAlpha)	fromAlpha：动画开始透明度 toAlpha：动画结束透明度（取值范围 0.0～1.0）

续表

名　称	实　现　类	常用构造方法	参 数 说 明
Scale（尺寸变化动画）	ScaleAnimation	public ScaleAnimation (float fromX, float toX, float fromY, float toY, int pivot XType, float pivotXValue, int pivotYType, float pivot YValue)	fromX：起始 *X* 坐标上的伸缩尺寸 toX：结束 *X* 坐标上的伸缩尺寸 fromY：起始 *Y* 坐标上的伸缩尺寸 toY：结束 *Y* 坐标上的伸缩尺寸 pivotXType: *X* 坐标伸缩模式（取值有 Animation.ABSOLUTE、Animation.RELATIVE_TO_SELF 、 Animation.RELATIVE_TO_PARENT） pivotXValue：相当于 *X* 坐标伸缩值 pivotYType: *Y* 坐标伸缩模式（取值有 Animation.ABSOLUTE、Animation.RELATIVE_TO_SELF 、 Animation.RELATIVE_TO_PARENT） pivotYValue：相当于 *Y* 坐标伸缩值
Translate（位置变化动画）	TranslateAnimation	TranslateAnimation(float from XDelta, float toXDelta, float fromYDelta, float to YDelta)	fromXDelta：起始 *X* 坐标 toXDelta：结束 *X* 坐标 fromYDelta：起始 *Y* 坐标 toYDelta：结束 *Y* 坐标
Rotate（旋转动画）	RotateAnimation	RotateAnimation (float from Degrees, float to Degrees, int pivotXType, float pivotXValue,int pivotYType, float pivotYValue)	fromDegrees：旋转开始角度 toDegrees：旋转结束角度 pivotXType: *X* 坐标伸缩模式（取值有 Animation.ABSOLUTE、Animation.RELATIVE_TO_SELF、Animation.RELATIVE_TO_PARENT） pivotXValue：相当于 *X* 坐标伸缩值 pivotYType: *Y* 坐标伸缩模式（取值有 Animation.ABSOLUTE、Animation.RELATIVE_TO_SELF、Animation.RELATIVE_TO_PARENT） pivotYValue：相当于 *Y* 坐标伸缩值

③ 在该工程的 res\layout\目录中创建一个 main.xml 布局文件，在该布局文件中添加 4 个 Button 组件和 1 个 ImageView 组件，并为 ImageView 组件设置资源引用 android:src 属性。

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

   <ImageView
      android:id="@+id/ImageView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/girl"
      ></ImageView>
```

```
    <Button
        android:text="Test Scale..."
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button
        android:text="Test Alpha..."
        android:id="@+id/Button02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button android:text="Test Translate..."
            android:id="@+id/Button03"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

    <Button android:text="Test Rotate..."
            android:id="@+id/Button04"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
</LinearLayout>
```

④ 在 MainActivity 类的顶部声明使用到的 Button 和 ImageView 组件，在 onCreate() 方法中实例化之。

```
package com.amaker.ch12.app;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.RotateAnimation;
import android.view.animation.ScaleAnimation;
import android.view.animation.TranslateAnimation;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
    // 声明使用到的 Button 视图组件
    private Button b1, b2, b3, b4;
    // 声明使用到的 ImageView 组件
    private ImageView girlImage;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前视图布局
        setContentView(R.layout.main);
        // 实例化视图组件
        girlImage = (ImageView) findViewById(R.id.ImageView01);
```

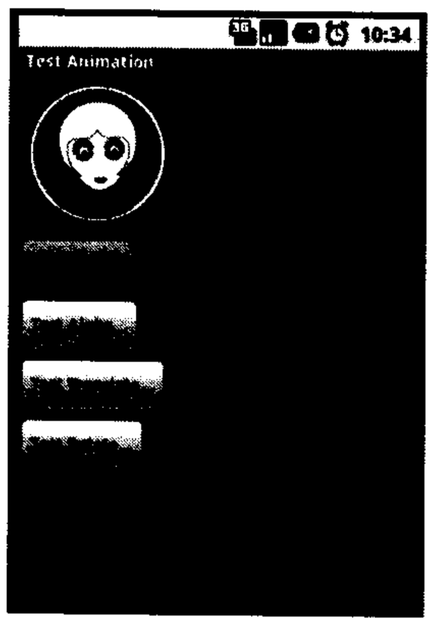
```
        b1 = (Button) findViewById(R.id.Button01);
        b2 = (Button) findViewById(R.id.Button02);
        b3 = (Button) findViewById(R.id.Button03);
        b4 = (Button) findViewById(R.id.Button04);
    }
```

⑤ 分别为四个按钮添加单击事件监听器，在监听器的方法中通过构造方法定义不同的动画，调用动画的 setDuration()方法设置动画持续时间，调用 ImageView 的 startAnimation()方法开始动画。

```
// 为按钮添加监听事件
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 Sacle(尺寸)变化动画
                Animation scaleAnimation = new ScaleAnimation(0f, 1f, 0f, 1f,
                        Animation.RELATIVE_TO_SELF, 0.5f,
                        Animation.RELATIVE_TO_SELF, 0.5f);
                // 设置动画持续时间
                scaleAnimation.setDuration(3000);
                // 开始动画
                girlImage.startAnimation(scaleAnimation);
            }
        });
        // 为按钮添加监听事件
        b2.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 Alpha(渐变)动画
                Animation alphaAnimation = new AlphaAnimation(0.1f, 1.0f);
                // 设置动画持续时间
                alphaAnimation.setDuration(3000);
                // 开始动画
                girlImage.startAnimation(alphaAnimation);
            }
        });
        // 为按钮添加监听事件
        b3.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 translate(位置变化)动画
                Animation translateAnimation = new TranslateAnimation(10, 100,
                        10, 100);
                // 设置动画持续时间
                translateAnimation.setDuration(3000);
                // 开始动画
                girlImage.startAnimation(translateAnimation);
            }
        });
        // 为按钮添加监听事件
        b4.setOnClickListener(new OnClickListener() {
```

```
        @Override
        public void onClick(View v) {
            // 创建 rotate(旋转)动画
            Animation rotateAnimation = new RotateAnimation(0f, +360f,
                    Animation.RELATIVE_TO_SELF, 0.5f,
                    Animation.RELATIVE_TO_SELF, 0.5f);
            // 设置动画持续时间
            rotateAnimation.setDuration(3000);
            // 开始动画
            girlImage.startAnimation(rotateAnimation);
        }
    });
```

程序运行结果如图 12.3 和图 12.4 所示。



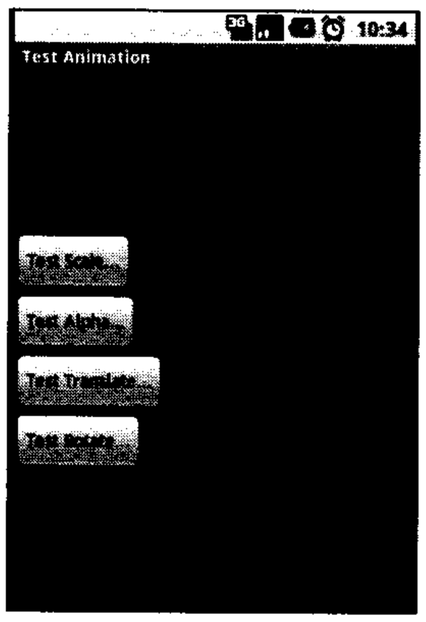
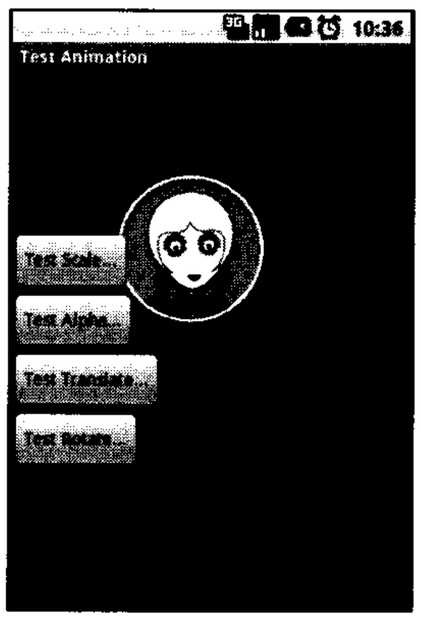


图 12.3　动画效果 Scale 和 Alpha



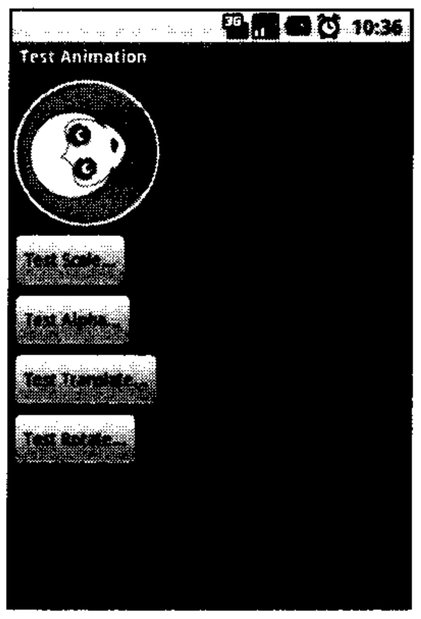


图 12.4　动画效果 Translate 和 Rotate

以上讲述的是通过硬编码的方式来实现的四种不同的动画效果，其实 Android 系统更推荐我们使用 XML 配置文件的方式来实现，该 XML 文件位于工程的 res\anim\目录中。XML 文件的格式如下所示。

```
<set android:shareInterpolator=boolean>
  <alpha android:fromAlpha=float
        android:toAlpha=float >   |
  <scale android:fromXScale=float
        android:toXScale=float
        android:fromYScale=float
        android:toYScale=float
        android:pivotX=string
        android:pivotY=string >    |
  <translate android:fromX=string
           android:toX=string
           android:fromY=string
           android:toY=string >   |
  <rotate android:fromDegrees=float
         android:toDegrees=float
         android:pivotX=string
         android:pivotY=string > |
  <interpolator tag>
```

```
  <set>
</set>
```

如上所示，必须要有一个 set 根元素，根元素里面定义不同的动画，如渐变、尺寸、位置变化和旋转等。

下面通过实例来演示如何使用 XML 配置文件实现各种不同的动画效果。基本思路是，在工程的 res\anim\目录下定义不同的动画配置文件。通过调用 AnimationUtils.loadAnimation()方法获得动画实例，调用视图组件的 startAnimation()方法开始动画。实例具体步骤说明如下。

① 创建一个 Android 工程“Chapter12_Graphic_Animation1”。

② 在工程的 res\drawable\目录中添加一个 girl.png 图片资源。

③ 在该工程的 res\layout\目录中创建一个 main.xml 布局文件，在该布局文件中添加 4 个 Button 组件和 1 个 ImageView 组件，并为 ImageView 组件设置资源引用 android:src 属性。

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

   <ImageView
      android:id="@+id/ImageView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/girl"
      ></ImageView>

   <Button
      android:text="Test Scale..."
      android:id="@+id/Button01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></Button>

   <Button
      android:text="Test Alpha..."
      android:id="@+id/Button02"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></Button>

   <Button android:text="Test Translate..."
         android:id="@+id/Button03"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"></Button>

   <Button android:text="Test Rotate..."
         android:id="@+id/Button04"
         android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"></Button>
</LinearLayout>
```

④ 在 MainActivity 类的顶部声明使用到的 Button 和 ImageView 组件，在 onCreate() 方法中实例化之。

```
package com.amaker.ch12.app;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.RotateAnimation;
import android.view.animation.ScaleAnimation;
import android.view.animation.TranslateAnimation;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
   // 声明使用到的 Button 视图组件
   private Button b1, b2, b3, b4;
   // 声明使用到的 ImageView 组件
   private ImageView girlImage;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置当前视图布局
      setContentView(R.layout.main);
      // 实例化视图组件
      girlImage = (ImageView) findViewById(R.id.ImageView01);
      b1 = (Button) findViewById(R.id.Button01);
      b2 = (Button) findViewById(R.id.Button02);
      b3 = (Button) findViewById(R.id.Button03);
      b4 = (Button) findViewById(R.id.Button04);
}
```

⑤ 在工程的 res\anim\目录下创建各种动画的 XML 配置文件。

- Alpha：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
  <alpha android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="5000"
         />
</set>
```

- Translate：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
 <translate
     android:fromXDelta="10"
```

```
        android:toXDelta="100"
        android:fromYDelta="10"
        android:toYDelta="100"
    >
    </translate>
</set>
```

- Scale：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:fromXScale="0.0"
        android:toXScale="1.0"
        android:fromYScale="0.0"
        android:toYScale="1.0"

        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000"
    >

    </scale>
</set>
```

- Rotate：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="0"
        android:toDegrees="-180"

        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000"
    >
    </rotate>
</set>
```

⑥ 分别为四个按钮添加单击事件监听器，在监听器的方法中通过 AnimationUtils.loadAnimation()方法定义不同的动画，调用 ImageView 的 startAnimation()方法开始动画。

```
// 为按钮添加监听事件
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 Sacle(尺寸)变化动画
                Animation scaleAnimation =
    AnimationUtils.loadAnimation(MainActivity.this,R.anim.my_scale);
                // 开始动画
                girlImage.startAnimation(scaleAnimation);
            }
        });
        // 为按钮添加监听事件
        b2.setOnClickListener(new OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                // 创建 Alpha(渐变)动画
                Animation alphaAnimation =

AnimationUtils.loadAnimation(MainActivity.this,R.anim.my_alpha);
                // 开始动画
                girlImage.startAnimation(alphaAnimation);
            }
        });
        // 为按钮添加监听事件
        b3.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 translate(位置变化)动画
                Animation translateAnimation =
AnimationUtils.loadAnimation(MainActivity.this,R.anim.my_translate);
                // 开始动画
                girlImage.startAnimation(translateAnimation);
            }
        });
        // 为按钮添加监听事件
        b4.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 创建 rotate(旋转)动画
                Animation rotateAnimation =
AnimationUtils.loadAnimation(MainActivity.this,R.anim.my_rotate);
                // 设置动画持续时间
                rotateAnimation.setDuration(3000);
                // 开始动画
                girlImage.startAnimation(rotateAnimation);
            }
        });
```

程序运行结果如图 12.5 和图 12.6 所示。

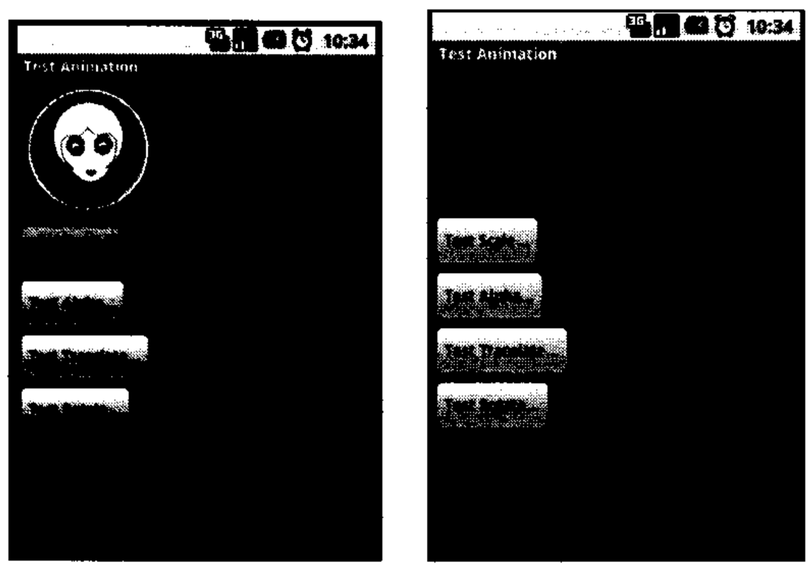


图 12.5　动画效果 Scale 和 Alpha

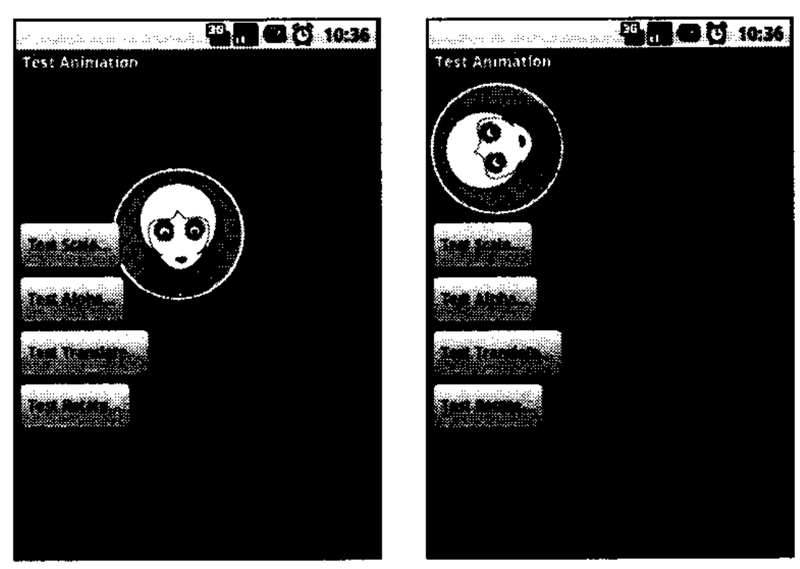


图 12.6　动画效果 Translate 和 Rotate

12.2.2　Frame 动画

Frame 动画是顺序播放图片来产生动画效果的，类似电影。例如，我们要实现一个人物走动的动画，可以通过三张图片来实现，第一张两脚都着地，第二张左脚着地，第三张右脚着地。然后，按顺序播放就实现人物行走的动画效果了。

Frame 动画是通过 AnimationDrawable 类实现的，该类中的两个重要方法是 start()和 stop()，分别用来开始和停止动画。动画一般通过 XML 配置文件来进行配置，在工程的 res\anim\目录下创建一个 XML 配置文件，该配置文件中有一个<animation-list>根元素和若干个<item>子元素。

下面还是通过一个实例来演示 Frame 动画的实现。该实例通过顺序播放 6 张图片来实现一个人跳舞的动画效果。六张静态图片如图 12.7 所示。

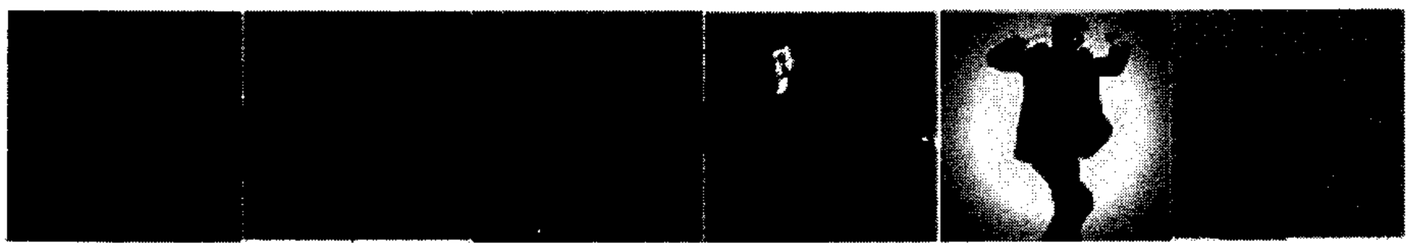


图 12.7　Frame 动画静态图片

实例步骤说明如下。

① 创建一个工程“Chapter12_Graphic_Animation2”。

② 在该工程的 res\drawable\目录下添加上述六张资源文件。

③ 在该工程的 res\anim\目录下创建一个 XML 动画文件。

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
   android:oneshot="true">
   <item android:drawable="@drawable/p01" android:duration="500" />
   <item android:drawable="@drawable/p02" android:duration="500" />
   <item android:drawable="@drawable/p03" android:duration="500" />

   <item android:drawable="@drawable/p04" android:duration="500" />
   <item android:drawable="@drawable/p05" android:duration="500" />
   <item android:drawable="@drawable/p06" android:duration="500" />
</animation-list>
```

④ 在工程的 res\layout\目录下创建一个 main.xml 布局文件，在该布局文件中添加两个 Button，一个用来开始动画，一个用来停止动画。添加一个 ImageView 显示动画。设置 ImageView 的背景色 android:background="@anim/dance"。

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

   <ImageView
```

```
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@anim/dance"
        ></ImageView>

    <Button
        android:text="Start..."
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>

    <Button
        android:text="Stop..."
        android:id="@+id/Button02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

⑤ 在 MainActivity 类的顶部声明 Button、ImageView 和 AnimationDrawable 实例。在 onCreate()方法中实例化之。获得 ImageView 视图的背景色，并转换为 AnimationDrawable 对象。为按钮添加单击事件监听器，在事件方法中分别开始动画和结束动画。

```
package com.amaker.ch12.app;

import android.app.Activity;
import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends Activity {
    // 声明使用到的 Button 视图组件
    private Button b1, b2;
    // 声明使用到的 ImageView 组件
    private ImageView myImage;
    // 声明 AnimationDrawable
    private AnimationDrawable danceAnimation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前视图布局
        setContentView(R.layout.main);
        // 实例化视图组件
        myImage = (ImageView) findViewById(R.id.ImageView01);
        b1 = (Button) findViewById(R.id.Button01);
        b2 = (Button) findViewById(R.id.Button02);

        // 获得背景色，并转换为 AnimationDrawable 对象
```

```
        danceAnimation = (AnimationDrawable) myImage.getBackground();

        // 为按钮添加监听事件
        b1.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 开始动画
                danceAnimation.start();
            }
        });
        // 为按钮添加监听事件
        b2.setOnClickListener(new
OnClickListener() {
            @Override
            public void onClick(View v) {
                // 停止动画
                danceAnimation.stop();
            }
        });

    }
}
```

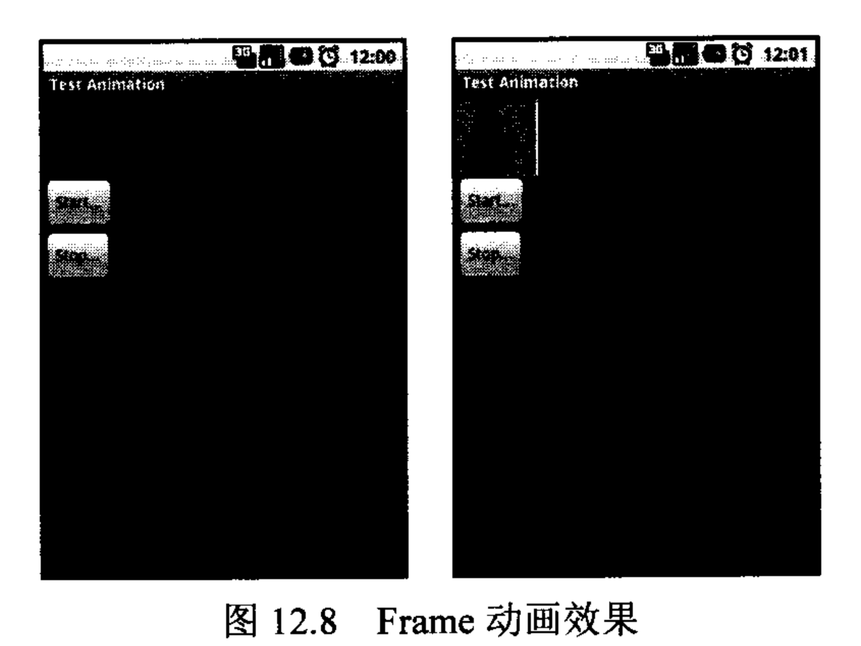


图 12.8　Frame 动画效果

程序运行结果如图 12.8 所示。

12.3 动态图形绘制

12.3.1　动态图形绘制的基本思路

动态图形绘制的基本思路是，创建一个类继承 View 类（或者继承 SurfaceView 类）。覆盖 onDraw()方法，使用 Canvas 对象在界面上绘制不同的图形，使用 invalidate()方法刷新界面。下面通过一个弹球实例来讲述动态图形绘制的基本思路。该实例是在界面上动态绘制一个小球，当小球触顶或者触底时自动改变方向继续运行。实例步骤说明如下。

① 创建一个 Android 工程，入口 Activity 的名称为 MainActivity。

```
package com.amaker.test.app;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.AttributeSet;
import android.view.View;
```

```
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

② 在 MainActivity 类中创建一个 MyView 内部类，该类实现 Runnable 接口支持多线程。

2.1 在 onDraw()方法中，定义 Paint 画笔并设置画笔颜色，使用 Canvas 的 drawCircle()方法画圆。

2.2 定义一个 update()方法，用于更新 *Y* 坐标。

2.3 定义一个消息处理器类 RefreshHandler，该类继承 Handler 并覆盖 handleMessage()方法，在该方法中处理消息。

2.4 在线程的 run()方法中设置并发送消息。

2.5 在构造方法中启动线程。

```
// 自定义视图类
class MyView extends View implements Runnable {
    // 图形当前坐标
    private int x = 20, y = 20;
    // 构造方法
    public MyView(Context context, AttributeSet attrs) {
        super(context, attrs);
        // 获得焦点
        setFocusable(true);
        // 启动线程
        new Thread(this).start();
    }
    @Override
    public void run() {
        RefreshHandler mRedrawHandler = new RefreshHandler();
        while (!Thread.currentThread().isInterrupted()) {
            // 通过发送消息更新界面
            Message m = new Message();
            m.what = 0x101;
            mRedrawHandler.sendMessage(m);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        // 实例化画笔
        Paint p = new Paint();
```

```
        // 设置画笔颜色
        p.setColor(Color.GREEN);
        // 画圆
        canvas.drawCircle(x, y, 10, p);
    }

    // 更新界面处理器
    class RefreshHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            if (msg.what == 0x101) {
                MyView.this.update();
                MyView.this.invalidate();
            }
            super.handleMessage(msg);
        }
    };
    // 更新坐标
    private void update() {
        int h = getHeight();
        y += 5;
        if (y >= h)
            y = 20;
    }
}
```

③ 在 MainActivity 类的 onCreate()方法中实例化 MyView 类，并将其设置为 Activity 内容视图。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyView v = new MyView(this, null);
    setContentView(v);
}
```

④ 程序运行结果如图 12.9 所示。

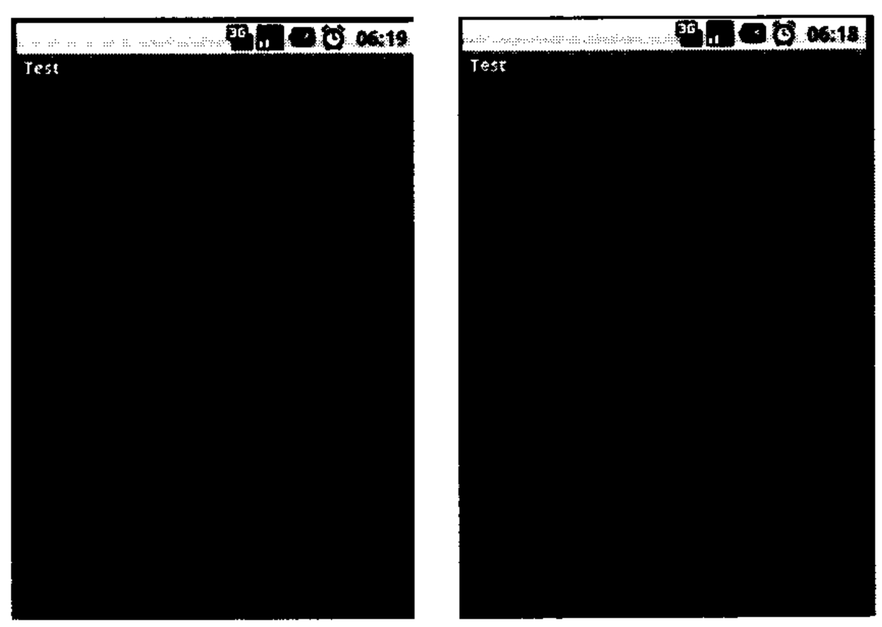


图 12.9　绘制图形

12.3.2 动态图形绘制类简介

如何动态绘制图形呢？其实，在 Android 中设计的这些工具类都很形象。我们想一想真正地画一张画需要哪些东西呢？首先我们需要一张画布，这里就是 Android 中的 Canvas。其次还需要画笔，这里就是 Android 中的 Paint。再次需要不同的颜色，这里就是 Android 中的 Color。接下来如果要画线还需要连接路径，这里就是 Android 中的 Path。我们还可以借助工具直接画出各种图形如圆、椭圆、矩形等，这里就是 Android 中的 ShapeDrawable 类，当然它还有很多子类，例如 OvalShape（椭圆）、RectShape（矩形）等。

1. Canvas

Canvas 就是我们所说的画布，位于 android.graphics 包中，提供了一些画各种图形的方法，例如矩形、圆、椭圆等。该类的详细方法如表 12.2 所示。

表 12.2 Canvas 常用方法

方法名称	方法描述
drawText(String text, float x, float y, Paint paint)	在屏幕上写字
drawPoint(float x, float y, Paint paint)	画点
drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	画线
drawCircle(float cx, float cy, float radius, Paint paint)	画圆
drawOval(RectF oval, Paint paint)	画椭圆
drawRect(RectF rect, Paint paint)	画矩形
drawRoundRect(RectF rect, float rx, float ry, Paint paint)	画圆角矩形
clipRect(float left, float top, float right, float bottom)	剪辑矩形
clipRegion(Region region)	剪辑区域

2. Paint

Paint 就是涂料的意思，用来描述图形的颜色和风格，如线宽、颜色、字体等信息。Paint 位于 android.graphics 包中，该类的详细方法如表 12.3 所示。

表 12.3 Paint 常用方法

方法名称	方法描述
Paint()	构造方法，使用默认设置
setColor(int color)	设置颜色
setStrokeWidth(float width)	设置线宽
setTextAlign(Paint.Align align)	设置文字对齐
setTextSize(float textSize)	设置文字尺寸
setShader(Shader shader)	设置渐变
setAlpha(int a)	设置 alpha 值
reset()	复位 Paint 默认设置

3. Color

Color 类定义了一些颜色常量和一些创建颜色的方法。颜色的定义一般使用 RGB 三原色来定义。Color 位于 android.graphics 包中，其常用属性和方法如表 12.4 所示。

表 12.4　Color 常用属性和方法

方法或者属性名称	方 法 描 述
BLACK	黑色
BLUE	蓝色
CYAN	青色
DKGRAY	深灰色
GRAY	灰色
GREEN	绿色
LTGRAY	浅灰色
MAGENTA	紫色
RED	红色
TRANSPARENT	透明
WHITE	白色
YELLOW	黄色

4. Path

当我们想要画一个圆的时候，我们只需指定圆心（点）和半径就可以了。那么，如果要画一个梯形呢？这里我们需要有点和连线。Path 一般用来从某个点移动到另一个点连线。Path 位于 android.graphics 包中，
详细方法如表 12.5 所示。

表 12.5　Path 方法

方法或者属性名称	方 法 描 述
lineTo(float x, float y)	从最后点到指定点划线
moveTo(float x, float y)	移动到指定点
reset()	复位

12.3.3　绘制几何图形

通过上述四个类的学习，我们可以使用它们来画出一些几何图形了。下面的实例演示了如何使用这些类来画出一些常见几何图形。程序步骤说明如下。

① 创建一个 Activity。

② 创建 Activity 的内部类 MyView 继承 View。

③ 覆盖 View 的 onDraw()方法。

④ 在 onDraw()方法中创建 Paint 对象、Path 对象，设置 Canvas 属性画出各种图形。

⑤ 在 Activity 的 onCreate()方法中实例化 MyView，调用 Activity 的 setContentView()方法，将其设置为当前视图。

程序代码如下所示。

```
package com.amaker.test;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.RectF;
import android.graphics.Shader;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(new MyView(this));
    }

    // 自定义视图类
    private class MyView extends View{
        public MyView(Context context) {
            super(context);
        }

        @Override
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
             // 设置 Canvas 颜色
              canvas.drawColor(Color.WHITE);
              // 实例化 Paint
              Paint paint = new Paint();
              paint.setAntiAlias(true);
              // 设置颜色
              paint.setColor(Color.RED);
              // 设置样式
              paint.setStyle(Paint.Style.STROKE);
              // 设置笔画粗细
              paint.setStrokeWidth(3);
              // 画圆
              canvas.drawCircle(40,40,30, paint);
              // 画矩形
```

```
canvas.drawRect(10,90,70,150,paint);
// 画矩形
canvas.drawRect(10,170,70,200,paint);
// 声明矩形
RectF re=new RectF(10,220,70,250);
// 画椭圆
canvas.drawOval(re, paint);
// 实例化路径
Path path = new Path();
// 移动到指定点
path.moveTo(10,330);
// 画线
path.lineTo(70,330);
// 画线
path.lineTo(40,270);
// 关闭路径
path.close();
// 画路径
canvas.drawPath(path, paint);
// 实例化路径
Path path1 = new Path();
// 移动到指定点
path1.moveTo(10,410);
// 画线
path1.lineTo(70,410);
// 画线
path1.lineTo(55,350);
// 画线
path1.lineTo(25,350);
// 关闭路径
path1.close();
// 画路径
canvas.drawPath(path1, paint);
// 设置样式
paint.setStyle(Paint.Style.FILL);
// 设置颜色
paint.setColor(Color.BLUE);
// 画圆
canvas.drawCircle(120, 40, 30, paint);
// 画矩形
canvas.drawRect(90,90,150,150,paint);
// 画矩形
canvas.drawRect(90,170,150,200,paint);
// 实例化矩形
RectF re2=new RectF(90,220,150,250);
// 画椭圆
canvas.drawOval(re2, paint);
// 实例化路径
Path path2 = new Path();
// 移动到指定点
path2.moveTo(90,330);
// 画线
```

```
path2.lineTo(150,330);
// 画线
path2.lineTo(120,270);
// 关闭路径
path2.close();
// 画路径
canvas.drawPath(path2, paint);
// 实例化路径
Path path3 = new Path();
// 移动到指定点
path3.moveTo(90,410);
// 画线
path3.lineTo(150,410);
// 画线
path3.lineTo(135,350);
// 画线
path3.lineTo(105,350);
 // 关闭路径
path3.close();
 // 画路径
canvas.drawPath(path3, paint);
// 线性渲染
Shader mShader=new LinearGradient(0, 0,100,100,
    new int[]{Color.RED, Color.GREEN,Color.BLUE,Color.YELLOW},
    null, Shader.TileMode.REPEAT);
 // 为paint 设置线性渲染
paint.setShader(mShader);
// 画圆
canvas.drawCircle(200,40, 30, paint);
// 画矩形
canvas.drawRect(170,90,230,150,paint);
// 画矩形
canvas.drawRect(170,170,230,200,paint);
// 矩形
RectF re3=new RectF(170,220,230,250);
// 画椭圆
canvas.drawOval(re3, paint);
// 实例化路径
Path path4 = new Path();
// 移动到指定点
path4.moveTo(170,330);
// 画线
path4.lineTo(230,330);
 // 画线
path4.lineTo(200,270);
  // 关闭路径
path4.close();
// 画路径
canvas.drawPath(path4, paint);
// 实例化路径
Path path5 = new Path();
// 移动到指定点
```

```
            path5.moveTo(170,410);
              // 画线
            path5.lineTo(230,410);
            // 画线
            path5.lineTo(215,350);
            // 画线
            path5.lineTo(185,350);
            // 关闭路径
            path5.close();
            // 画路径
            canvas.drawPath(path5, paint);
            // 设置文本大小
            paint.setTextSize(24);
           // 写文字
            canvas.drawText(getResources().getString(R.string.str_text1),
                        240,50,paint);
           // 写文字
            canvas.drawText(getResources().getString(R.string.str_text2),
                        240,120,paint);
           // 写文字
            canvas.drawText(getResources().getString(R.string.str_text3),
                        240,190,paint);
           // 写文字
            canvas.drawText(getResources().getString(R.string.str_text4),
                        240,250,paint);
           // 写文字

canvas.drawText(getResources().getString(R.string.str_text5),
                        240,320,paint);
           // 写文字

canvas.drawText(getResources().getString(R.string.str_text6),
                        240,390,paint);

        }
    }
}
```

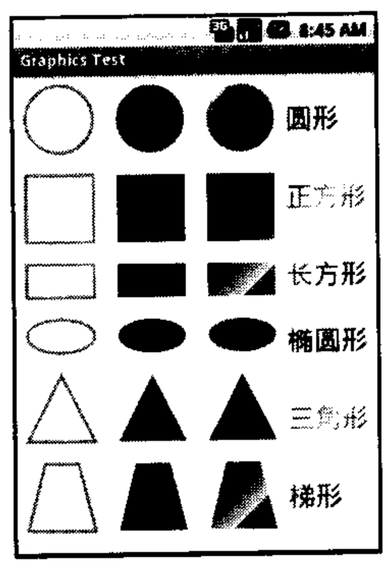


图 12.10　2D 图形

程序运行结果如图 12.10 所示。

12.4 图形特效

12.4.1　使用 Matrix 实现旋转、缩放和平移

在 Android 图形 API 中提供了一个 Matrix 矩阵类，该类具有一个 3×3 的矩阵坐标。通过该类可以实现图形的旋转、平移和缩放。使用 reset()方法来初始化矩阵类，使用 setScale()、setTranslate()和 setRotate()方法来设置矩阵缩放、平移和旋转属性。

下面我们通过一个实例来演示 Matrix 的具体应用，在本实例中我们自定义一个 View 类，在该类中拥有一个 Bitmap 和 Matrix 实例，Bitmap 实例从系统资源加载一张图片，覆盖 View 类的 onDraw()方法，在该方法中通过 reset()方法初始化 Matrix，并设置其旋转或缩放属性，使用 Canvas 的 drawBitmap()方法将 Bitmap 重新绘制在视图中。通过键盘事件 onKeyDown()实现旋转属性和缩放属性的改变，调用 postInvalidate()方法重新绘制 Bitmap。实例步骤说明如下。

① 创建一个 Android 工程“Chapter12_Graphic_Matrix”，入口 Activity 的名称为 MainActivity。

```
package com.amaker.ch12.app;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;

public class MainActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       MyView myView = new MyView(MainActivity.this);
       // 设置当前视图布局
       setContentView(myView);
   }
}
```

② 在该工程的 res\drawable\目录下添加一张图片资源 girl.png。

③ 在 MainActivity 中定义一个内部类 MyView 继承 View。在该类的顶部声明使用到的变量，在构建方法中初始化变量，覆盖 onDraw()方法和 onKeyDown()方法。

```
// 自定义视图类
class MyView extends View {
    // 位图实例
    private Bitmap bm;
    // Matrix 实例
    private Matrix matrix = new Matrix();
    // 旋转角度
    private float angle = 0.0f;
    // 位图宽和高
    private int w, h;
    // 缩放比例
    private float scale = 1.0f;
    // 判断缩放还是旋转
```

```
    private boolean isScale = false;

    // 构造方法
    public MyView(Context context) {
        super(context);
        // 获得位图
        bm = BitmapFactory.decodeResource(this.getResources(),
                R.drawable.girl);
        // 获得位图宽
        w = bm.getWidth();
        // 获得位图高
        h = bm.getHeight();
        // 使当前视图获得焦点
        this.setFocusable(true);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        // 重置 Matrix
        matrix.reset();
        if (!isScale) {
            // 旋转 Matrix
            matrix.setRotate(angle);
        } else {
            // 缩放 Matrix
            matrix.setScale(scale, scale);
        }
        // 根据原始位图和 Matrix 创建新视图
        Bitmap bm2 = Bitmap.createBitmap(bm, 0, 0, w, h, matrix, true);
        // 绘制新视图
        canvas.drawBitmap(bm2, matrix, null);
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        // 向左旋转
        if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
            isScale = false;
            angle++;
            postInvalidate();
        }
        // 向右旋转
        if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
            isScale = false;
            angle--;
            postInvalidate();
        }
        // 放大
        if (keyCode == KeyEvent.KEYCODE_DPAD_UP) {
            isScale = true;
            if (scale < 2.0)
                scale += 0.1;
            postInvalidate();
        }
        // 缩小
```

```
        if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN) {
            isScale = true;
            if (scale > 0.5)
                scale -= 0.1;
            postInvalidate();
        }

        return super.onKeyDown(keyCode, event);
    }
```

④ 在 MainActivity 的 onCreate()方法中，实例化 MyView 并将其设置为当前 Activity 的视图内容。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyView myView = new MyView(MainActivity.this);
    // 设置当前视图布局
    setContentView(myView);
}
```

程序运行结果如图 12.11 所示。

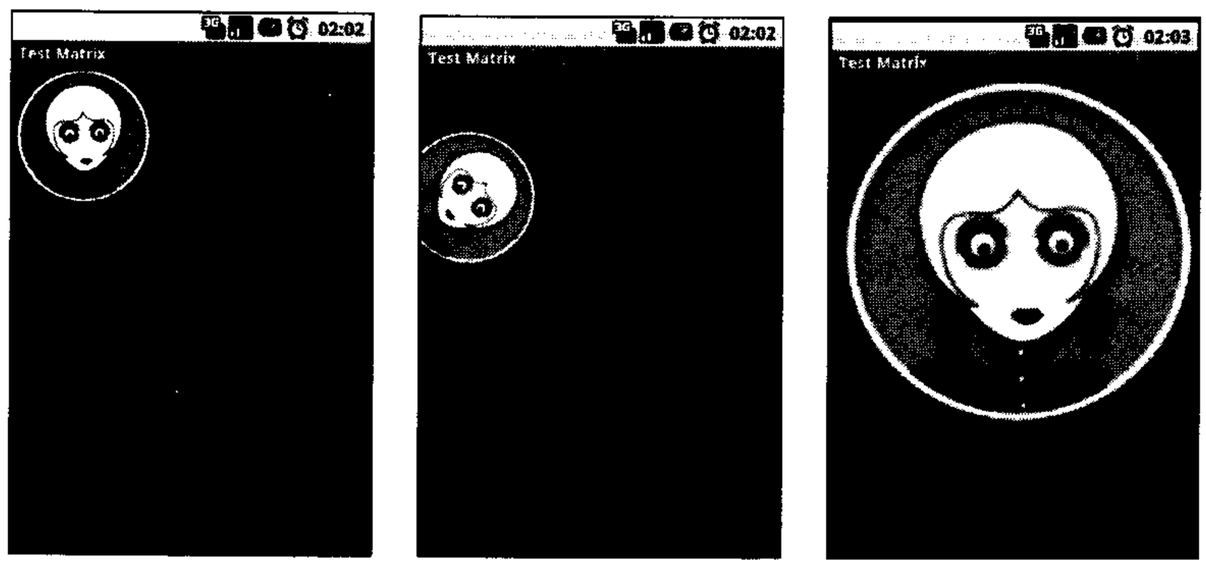


图 12.11　使用 Matrix 旋转、缩放

12.4.2　使用 Shader 类渲染图形

我们经常会看到一些屏保或者一些音乐播放器的界面中有一些非常绚丽的图形，这些其实是在一些原有图形的基础上进行渲染的结果。Android 系统中提供了 Shader 渲染类来实现这一功能。Shader 是一个抽象父类，其子类有很多个，如 BitmapShader（位图渲染）、ComposeShader（混合渲染）、LinearGradient（线性渲染）、RadialGradient（光束渲染）和 SweepGradient（梯度渲染）等。通过 Paint 对象的 paint.setShader(shader)方法来使用 Shader。

下面通过一个实例来演示不同 Shader 的渲染效果。该实例自定义一个 MyView 继承 View 类。在该类中构建不同的 Shader 对象，在 onDraw()方法中绘制各种渲染后的图形。实例步骤说明如下。

① 创建一个 Android 工程“Chapter12_Graphic_Shader”，入口 Activity 类的名称为

MainActivity。

```
package com.amaker.test.app;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ComposeShader;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.PorterDuff;
import android.graphics.RadialGradient;
import android.graphics.Shader;
import android.graphics.SweepGradient;
import android.graphics.Shader.TileMode;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.View;

public class MainActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
   }
}
```

② 在工程的 res\drawable\目录下添加一个 girl.png 资源文件。

③ 在 MainActivity 中自定义一个 MyView 类，该类继承 View 类并覆盖其 onDraw() 方法。

3.1 在该类的顶部声明各种渲染对象及 Paint 对象。

3.2 在构造方法中进行实例化。

3.3 在 onDraw 方法中根据当前的渲染绘图。

3.4 在 onKeyDown()方法中通过键盘按键改变渲染对象。

```
// 自定义视图类
class MyView extends View{
   // 声明 Bitmap 对象
   private Bitmap bm ;
   // 声明位图渲染对象
   private Shader bitmapShader;
   // 声明线性渲染对象
   private Shader linearGradient;
   // 声明光束渲染对象
   private Shader radialGradient;
   // 声明梯度渲染对象
   private Shader sweepGradient;
   // 声明混合渲染对象
   private Shader composeShader;
   // 声明画笔
```

```
        private Paint paint;
        // 声明颜色数组
        private int[] colors;
        private boolean isFirst = true;

        public MyView(Context context) {
            super(context);
            // 获得 Bitmap 实例
            bm = BitmapFactory.decodeResource(getResources(), R.drawable.girl);
            // 实例化画笔
            paint = new Paint();
            colors = new int[]{Color.RED,Color.GREEN,Color.BLUE};
            // 实例化位图渲染对象,x 坐标方向重复图形，y 坐标方向镜像图形
            bitmapShader = new BitmapShader(bm, TileMode.REPEAT, TileMode.MIRROR);
            // 实例化线性渲染
            linearGradient = new LinearGradient(0, 0, 100, 100,
                    colors, null,  TileMode.REPEAT);
            // 实例化光束渲染
            radialGradient = new RadialGradient(100, 100, 80,
                    colors, null,  TileMode.REPEAT);
            // 实例化梯度渲染
            sweepGradient =  new SweepGradient(200, 200, colors, null);

            // 实例化混合渲染
            composeShader =
new ComposeShader(linearGradient, radialGradient,PorterDuff.Mode.DARKEN);
            // 使其获得焦点
            setFocusable(true);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
            if(isFirst){
                //写字，用来提示用户操作
                String content = "按上/下/左/右/中间键测试！";
                paint.setColor(Color.BLUE);
                canvas.drawText(content, 0, content.length()-1, 20, 20, paint);
            }else{
                // 全屏画矩形
                canvas.drawRect(0, 0, getWidth(), getHeight(), paint);
            }
        }

        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
            isFirst = false;
            if(keyCode==KeyEvent.KEYCODE_DPAD_UP){
                // 将画笔渲染设置为位图渲染
                paint.setShader(bitmapShader);
            }
            if(keyCode==KeyEvent.KEYCODE_DPAD_DOWN){
                // 将画笔渲染设置为线性渲染
                paint.setShader(linearGradient);
            }
            if(keyCode==KeyEvent.KEYCODE_DPAD_LEFT){
                // 将画笔渲染设置为光束渲染
```

```
                paint.setShader(radialGradient);
            }
            if(keyCode==KeyEvent.KEYCODE_DPAD_RIGHT){
                // 将画笔渲染设置为梯度渲染
                paint.setShader(sweepGradient);
            }
            if(keyCode==KeyEvent.KEYCODE_DPAD_CENTER){
                // 将画笔渲染设置为混合渲染
                paint.setShader(composeShader);
            }
            // 重绘界面
            postInvalidate();
            return super.onKeyDown(keyCode, event);
        }
    }
    }
```

④ 在 MainActivity 类的 onCreate()方法中，实例化 MyView 并将其设置为 Activity 的内容视图。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyView v = new MyView(this);
    setContentView(v);
}
```

程序运行结果如图 12.12 和图 12.13 所示。

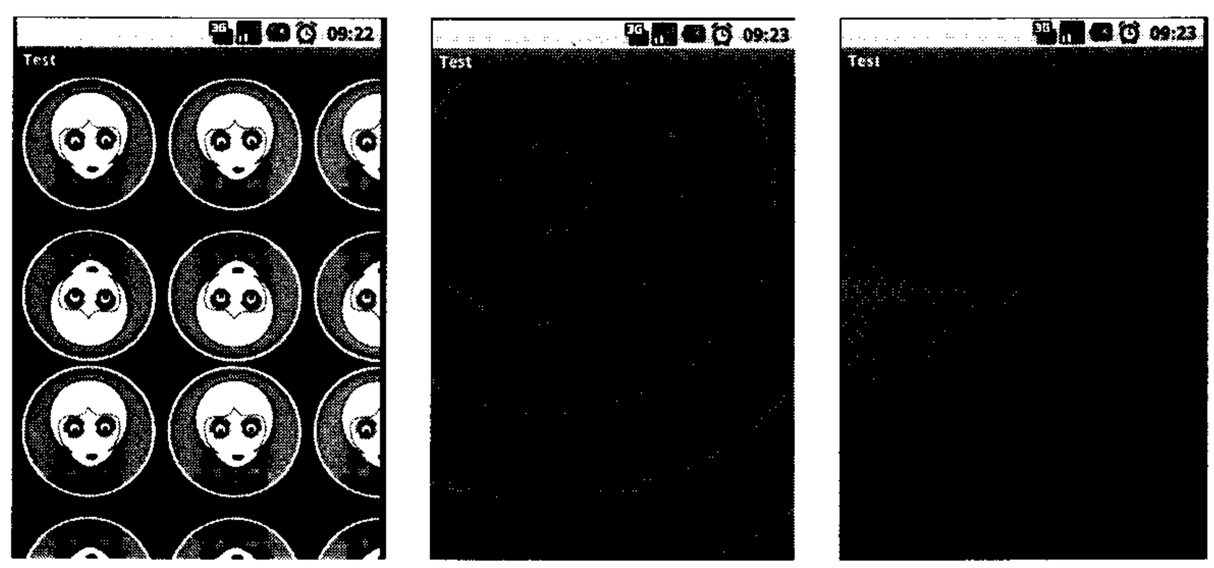


图 12.12　使用 Shader 渲染效果 1

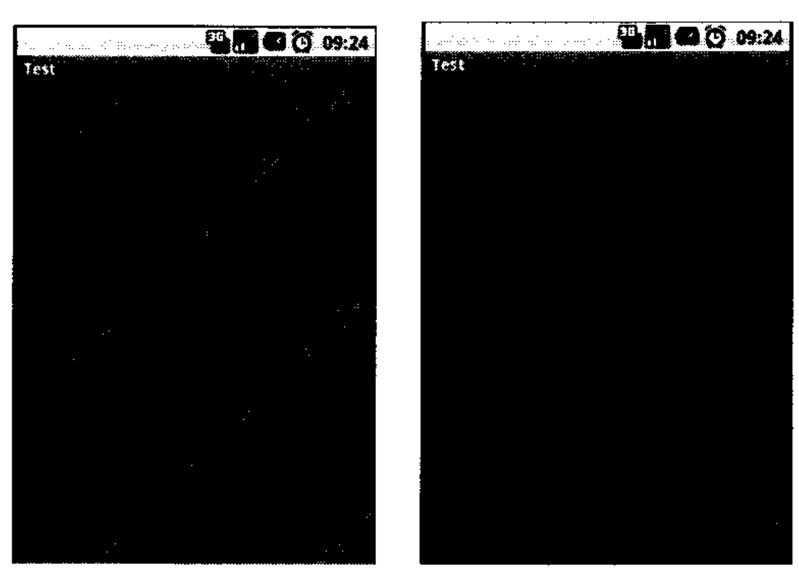
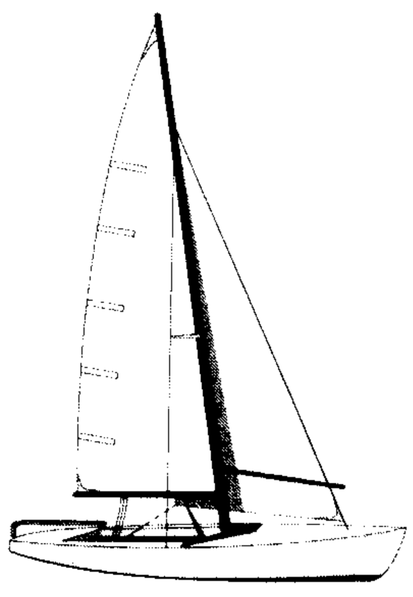


图 12.13　使用 Shader 渲染效果 2



第 13 章　Android 中的互联网应用

Google 是一个互联网公司，通过核心业务搜索引擎，在互联网领域独占鳌头。Android 手机操作系统的发布标志着 Google 将触角伸向了移动互联网领域。Android 出自 Google 之手，那么其网络功能自然更加强大。

Google 的应用层采用 Java 语言，查看 Android API 文档会发现 Java 里面提供的网络编程方式，在 Android 中都提供了支持。具体的网络编程方式有：

- 针对 TCP/IP 的 Socket、ServerSocket
- 针对 UDP 的 DatagramSocket、DatagramPackage
- 针对直接 URL 的 URL、URLConnection 和 HttpURLConnection
- Google 集成了 Apache HTTP 客户端，这使得使用 HTTP 进行网络编程成为可能
- 我们也可以使用 Web Service 进行网络编程
- 直接使用 WebView 视图组件显示网页。

另外，谈到网络编程我们就离不开 I/O 流，Android 集成了 Java 里的 I/O 编程包，所以 I/O 的操作方式和 Java 里的没有区别。I/O 编程类位于 java.io 包中。

13.1 通过 Socket、ServerSocket 进行网络编程

Socket、ServerSocket 编程方式可以说是比较底层的网络编程方式，其他的高级协议（如 HTTP）都是建立在此基础之上的，而且 Socket 编程是跨平台的编程，可以在异构语音之间进行通信，所以掌握 Socket 网络编程是掌握其他网络编程方式的基础。

13.1.1　Socket、ServerSocket 编程模型

在 Socket 编程模式中，Socket 类用来建立客户端程序，ServerSocket 类用来建立服务器端程序。首先，要有服务器端程序。通过 ServerSocket 建立服务器端程序，一般是指定

一个监听端口，用来等待客户端的连接。客户端 Socket 要指定服务器端的 IP 地址和端口号。一旦连接建立 ServerSocket 就可以获得一个 Socket 实例，通过 Socket 打开输入、输出流进行通信。具体编程模型如图 13.1 所示。

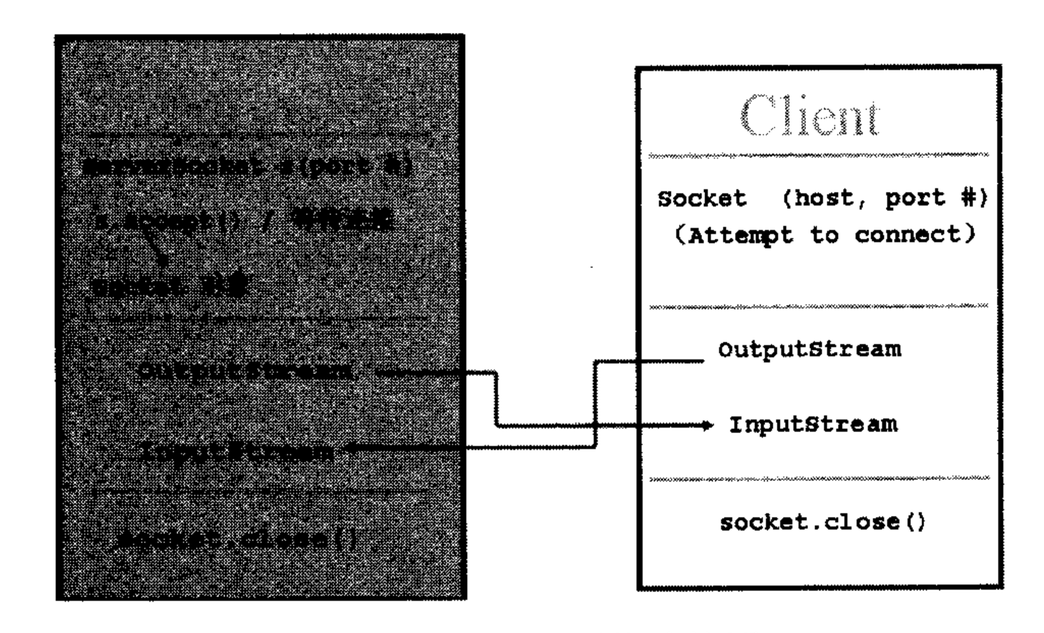


图 13.1　Socket 网络连接图

13.1.2　Socket 编程实例

下面我们通过一个实例来演示如何实现 Socket 在 Android 中的应用。一般 Socket 在 Android 中的编程应用中，Android 都作为客户端，因为无线上网 IP 地址的分配是由移动运营公司动态分配的，没有固定 IP 所以不能作为服务器端来使用。下面的实例是通过 Android 建立 Socket 客户端连接 ServerSocket 服务器端实现简单的通信。具体步骤说明如下。

① 创建一个名为“Chapter_13_Networking_server”的 Java 工程。

② 创建一个名为“MyServer”的 Java 类作为服务器，响应 Android 客户端连接。当客户端连接时，向客户端写“Hello Android!”字符。

```
package com.amaker.test;

import java.io.IOException;
import java.io.OutputStream;
import java.net.ServerSocket;
import java.net.Socket;
/**
 *
 * @author 郭宏志
 * Socket 服务器，向客户端写 Hello Android!
 */
public class MyServer {
   public static void main(String[] args) {
       try {
           // 实例化服务器套接字
           ServerSocket server = new ServerSocket(8888);
           // 永真循环
```

```
            while(true){
                // 获得客户端套接字
                Socket client = server.accept();
                // 获得输出流
                OutputStream out = client.getOutputStream();
                // 输出字符串
                String msg = "Hello Android!";
                // 写字符串
                out.write(msg.getBytes());
                // 关闭输出流
                client.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

③ 创建一个名称为“Chapter13_Networking_Client”的 Android 工程。

④ 在 Android 工程的 res\layout\目录下，创建一个“socket.xml”布局文件。在该布局文件中添加一个 TextView，用来显示接收到服务器的信息。

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

    <TextView
    android:text="等待接收服务器信息..."
    android:id="@+id/msgTextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></TextView>

</LinearLayout>
```

⑤ 创建一个名为“TestSocketActivity”的 Activity 类来接收服务器信息。该类首先获得 TextView 实例，建立 Socket 连接，从 Socket 连接中获得输入流，最后读取输入流中的信息并显示在 TextView 中。

```
import android.app.Activity;
package com.amaker.ch13.socket;

import java.io.IOException;
import java.io.InputStream;
import java.net.Socket;
import java.net.UnknownHostException;

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

```
import android.widget.TextView;
import com.amaker.ch13.R;
/**
 *
 * @author 郭宏志
 * Android  Socket 客户端接收服务器信息
 */
public class TestSocketActivity extends Activity {
    // 声明文本视图 TextView
    private TextView myTextView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.socket);
        // 通过 findViewById 方法获得 TextView
        myTextView = (TextView)findViewById(R.id.msgTextView01);
        try {
            // 实例化 Socket
            Socket socket = new Socket("192.168.1.101",8888);
            // 获得输入流
            InputStream in = socket.getInputStream();
            // 缓冲区
            byte[] buffer = new byte[in.available()];
            // 读到缓冲区
            in.read(buffer);
            // 转换为字符串
            String msg = new String(buffer);
            // 使用 TextView 显示
            myTextView.setText(msg);
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

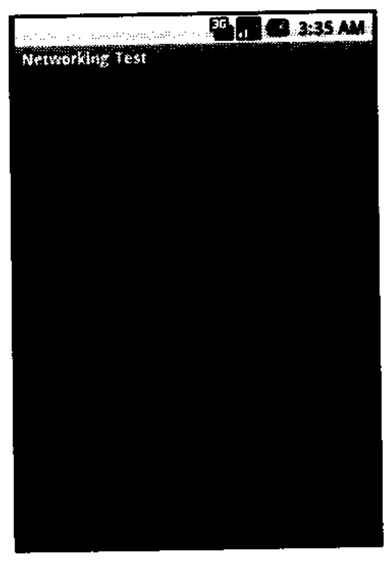


图 13.2　Android Socket 网络编程

⑥ 启动服务器等待监听。

⑦ 运行 Android 客户端程序，结果如图 13.2 所示。

13.2 通过 URL 进行网络编程

如果我们知道网络上某个资源的 URL（如一些图片、音乐和视频文件等），那么我们就可以直接使用 URL 来进行网络连接。

在这里我们可以使用 URL、URLConnection 和 HttpURLConnection 进行 URL 网络连接，下面的实例要展示的是通过 URL 连接，读取服务器的一张图片，并使用 ImageView 组件显示出来。

① 在“Chapter13_Networking_Client”的 res\layout\目录下创建一个“test_url.xml”布局文件，在 test_url.xml 中添加一个 ImageView 组件。

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

    <ImageView
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></ImageView>

</LinearLayout>
```

② 创建一个名为“TestURLActivity”的 Activity 类，该类声明并实例化一个 ImageView 组件视图。声明连接服务器 URL 字符串，创建 URL 对象（这里我们可以使用 URL、URLConnection 和 HttpURLConnection 三种方式中的任何一种），打开输入流，将输入通过 BitmapFactory 类解码成 Bitmap 类并通过 ImageView 来显示。

```
package com.amaker.test;
package com.amaker.ch13.url;

import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;

import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.widget.ImageView;

import com.amaker.ch13.R;
/**
 * @author 郭宏志
 * 通过 URL 进行网络连接
 */
public class TestURLActivity extends Activity {
    // 声明图片视图 ImageView
    private ImageView imageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.test_url);
        // 通过 findViewById 方法获得 ImageView
        imageView = (ImageView)findViewById(R.id.ImageView01);
```

```
        // 要显示图片路径
        String  urlStr  =  "http://192.168.1.101:8080/Chapter_13_Networking_server/
upload/zs.jpg";

        try {
            // 实例化 URL
            URL url = new URL(urlStr);
            // 1. 直接使用 URL 获得输入流
            //InputStream in = url.openStream();

            // 2. 获得 URLconnection
            URLConnection conn = url.openConnection();
            InputStream in = conn.getInputStream();

            // 3. 如果是 HTTP 协议可以使用 HttpURLConnection
            //HttpURLConnection  httpConn  =  (HttpsURL
Connection)conn;
            //in = httpConn.getInputStream();
            // 通过 BitmapFactory 获得 Bitmap 实例
            Bitmap bm = BitmapFactory.decodeStream(in);
            // 为 ImageView 设置图片
            imageView.setImageBitmap(bm);

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

图 13.3　Android URL 网络编程

程序运行结果如图 13.3 所示。

13.3 通过 HTTP 进行网络编程

在有线互联网领域，基于 HTTP 的应用是最为广泛的，也就是我们所说的 B/S 结构程序，即浏览器/服务器结构的程序。随着移动互联网时代的来临，基于 HTTP 的手机等移动终端的 B/S 结构应用会更加广泛。

在 Android 中针对 HTTP 进行网络通信有两种：一是 HttpURLConnection；二是 Apache HTTP 客户端。本节就针对这两种情况进行详细阐述。

13.3.1　使用 HttpURLConnection

如果我们知道访问资源的 URL，并且是基于 HTTP 的，那么我们就可以使用 HttpURLConnection 进行请求/响应。通过 HttpURLConnection 我们可以发送请求并获得响应。

下面我们通过一个用户登录实例，来比较学习使用 HttpURLConnection 和使用 Apache HTTP 客户端的异同。具体步骤说明如下。

① 在“Chapter13_Networking_Client”工程的 res\layout\目录下，创建一个“http1.xml”布局文件。该布局文件使用 LinearLayout 嵌套 TableLayout 进行布局管理。其中添加两个 TextView 和两个 EditText 用于用户输入用户名称和密码，添加两个按钮用于提交和取消用户登录。

```
<?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">

   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">
      <TableRow>
         <TextView
            android:text="用户名称："
            android:id="@+id/TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            ></TextView>

         <EditText
            android:text=""
            android:id="@+id/userEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"></EditText>
      </TableRow>


      <TableRow>
         <TextView
            android:text="用户密码："
            android:id="@+id/TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            ></TextView>

         <EditText
            android:text=""
            android:id="@+id/pwdEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:password="true"></EditText>
      </TableRow>

      <TableRow android:gravity="right">
         <Button
            android:text="取消"
            android:id="@+id/cancelButton"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"></Button>

        <Button
            android:text="登陆"
            android:id="@+id/loginButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
    </TableRow>

  </TableLayout>

</LinearLayout>
```

② 在 com.amaker.ch13.http 包中创建一个“LoginActivity”。

2.1 在该类的顶部声明使用到的视图组件，并在 onCreate()方法中实例化这些组件。

```
package com.amaker.ch13.http;

import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.amaker.ch13.R;
public class LoginActivity extends Activity {
   // 声明 Button 实例
   private Button cancelBtn,loginBtn;
   // 声明 EditText 实例
   private EditText userEditText,pwdEditText;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置当前 Activity 界面布局
      setContentView(R.layout.http1);
      // 通过 findViewById 方法获得 Button 实例
      cancelBtn = (Button)findViewById(R.id.cancelButton);
      // 通过 findViewById 方法获得 Button 实例
      loginBtn = (Button)findViewById(R.id.loginButton);
      // 通过 findViewById 方法获得 EditText 实例
      userEditText = (EditText)findViewById(R.id.userEditText);
      // 通过 findViewById 方法获得 EditText 实例
      pwdEditText = (EditText)findViewById(R.id.pwdEditText);
}
}
```

2.2 创建一个显示对话框方法 showDialog()，用于显示提示信息。

```
// 定义一个显示提示信息的对话框
private void showDialog(String msg){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage(msg)
          .setCancelable(false)
          .setPositiveButton("确定", new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int id) {
             }
          });
    AlertDialog alert = builder.create();
    alert.show();
}
```

2.3 创建 login()方法进行登录。该方法首先声明请求的 URL，通过该地址创建 URL 对象，通过 URL 对象打开连接，获得 HttpURLConnection 对象。调用 HttpURLConnection 对象的 getResponseCode()方法判断是否请求成功，如果成功获得输入流，则读入提示信息并显示在对话框中提示用户。

```
    // 通过用户名称和密码进行查询，发送 Post 请求，获得响应结果。
    private void login(String username,String password){
        // 请求 URL
        String  urlStr  =  "http://192.168.1.101:8080/Chapter_13_Networking_server/
servlet/LoginServlet?";
        // 查询字符串
        String queryString = "username="+username+"&password="+password;
        urlStr+=queryString;
        try {
            // 实例化 URL
            URL url = new URL(urlStr);
            // 获得 HttpURLConnection 实例
            HttpURLConnection conn = (HttpURLConnection)url.openConnection();
            // 如果请求成功
            if(conn.getResponseCode()==HttpURLConnection.HTTP_OK){
                // 获得输入流
                InputStream in = conn.getInputStream();
                // 数组缓存
                byte[] b = new byte[in.available()];
                // 读数据到缓存
                in.read(b);
                // 转换为字符串
                String msg = new String(b);
                // 显示对话框
                showDialog(msg);
                // 关闭输入流
                in.close();
            }
            // 断开连接
            conn.disconnect();
        } catch (Exception e) {
```

```
            showDialog(e.getMessage());
        }
    }
```

2.4 为提交按钮添加单击事件，调用 Login 方法登录。为取消按钮添加单击事件，结束当前 Activity。

```
        //设置登录监听器
        loginBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                    // 获得用户名和密码
                    String username = userEditText.getText().toString();
                    String pwd = pwdEditText.getText().toString();
                    // 登录
                    login(username,pwd);
            }
        });
        // 设置取消监听器
        cancelBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 结束当前 Activity
                finish();
            }
        });
    }
```

3 在服务器端创建 LoginServlet，响应请求并返回登录结果。该 Servlet 只做简单判断，没有连接后台数据库，在实际项目当中将连接数据库进行查询。

```
package com.amaker.http;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author 郭宏志
 * 响应登录 Servlet
 */
public class LoginServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 从请求中获得用户名和密码
        String username = request.getParameter("username");
```

```
        String password = request.getParameter("password");
        // 设置内容类型
        response.setContentType("text/html");
        // 设置字符编码
        response.setCharacterEncoding("utf-8");
        // 获得打印输出流
        PrintWriter out = response.getWriter();
        String msg = null;
        // 判断登录成功还是失败
        if(username!=null&&username.equals("admin")&&password!=null&&password.
equals("1")){
            msg = "登录成功!";
        }else{
            msg = "登录失败!";
        }
        // 返回信息给客户端
        out.print(msg);
        // 刷新输出流
        out.flush();
        // 关闭输出流
        out.close();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }

}
```

程序运行结果如图 13.4 所示。

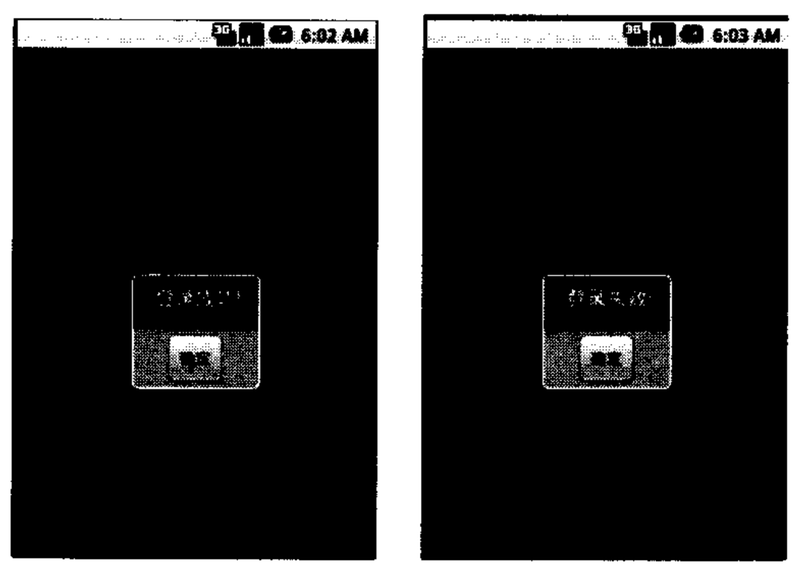


图 13.4　使用 HttpURLConnection 进行登录

13.3.2　使用 Apache HTTP 客户端

Android 集成了 Apache HTTP 客户端，使得针对 HTTP 的编程更加方便、高效。有过 B/S 编程经验的读者都知道，HTTP 是一种请求/响应机制。在 Servlet 编程中我们会用

HttpServletRequest 和 HttpServletResponse 来表示请求和响应。Apache HTTP 客户端也对请求和响应进行封装，根据请求方法的不同，我们会用到 HttpGet 和 HttpPost 两个对象。响应对象是 HttpResponse，使用 DefaultHttpClient 执行请求获得响应。

本实例在上面实例的基础上只是把 login 方法改写成使用 Apache HTTP Client，其他的都没有变化。login()方法内容如下所示。

```
    // 通过用户名称和密码进行查询，发送 Post 请求，获得响应结果。
    private void login(String username,String password){
        // 2. 使用 Apache HTTP 客户端实现
        String  urlStr  =  "http://192.168.1.101:8080/Chapter_13_Networking_server/
servlet/LoginServlet";
        HttpPost request = new HttpPost(urlStr);
        // 如果传递参数个数比较多的话，我们可以对传递的参数进行封装
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        // 添加用户名和密码
        params.add(new BasicNameValuePair("username", username));
        params.add(new BasicNameValuePair("password", password));
        try {
            // 设置请求参数项
            request.setEntity( new UrlEncodedFormEntity(params,HTTP.UTF_8));
            // 执行请求返回响应
            HttpResponse response = new DefaultHttpClient().execute(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得响应信息
                String msg = EntityUtils.toString(response.getEntity());
                // 在对话框中显示
                showDialog(msg);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

程序运行会得到同样的结果，如图 13.5 所示。

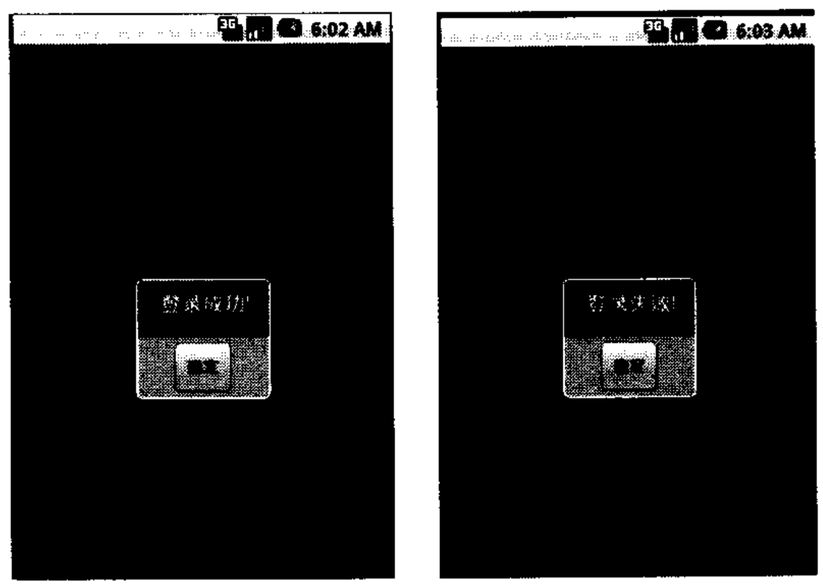


图 13.5　使用 Apache HTTP 客户端进行登录

13.4 通过 Web Service 进行网络编程

Web Service 是实现异构程序之间方法调用的一种机制。Web Service 通过一种 XML 格式的特殊文件来描述方法、参数、调用和返回值，这种格式的 XML 文件称为 WSDL（Web Service Description Language），即 Web 服务描述语言。Web Service 采用的通信协议是 SOAP（Simple Object Access Protocol），即简单对象访问协议。

Java 领域的 Web Service 实现有很多种方式，例如 Sun 公司的 XML-RPC、Apache 的 Axis 以及 Codehaus 的 Xfire，但是这些都不适合在资源有限的移动设备中使用。早期的 Java ME 开发当中使用到了 KSOAP，近期 Google 也发布了针对 Android 的 KSOAP jar 包。该 jar 文件的访问地址为 http://code.google.com/p/ksoap2-android/downloads/list。

下面的实例我们将通过 KSOAP 和 Apache HTTP 客户端来实现一个天气预报。具体步骤说明如下。

① 到 Google 的网站下载支持 Android 的 KSOAP jar 文件，下载地址为 http://code.google.com/p/ksoap2-android/downloads/list。

② 将该 jar 文件添加到“Chapter13_Networking_Client”工程的 classpath 下。

③ 在该工程的 res\layout\目录下创建一个布局文件 weather.xml。该布局文件采用 LinearLayout 实现布局管理。一个 TextView 用来显示天气预报标题，另一个 TextView 用来显示选择城市的提示，一个 Spinner 用来显示要选择的城市，最后一个 TextView 用来显示天气预报信息。由于信息较多，所以采用 ScrollView 来滚动显示。

```
<?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:text="天气预报"
      android:id="@+id/titleTextView01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"></TextView>

   <LinearLayout
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <TextView
         android:text="请选择城市："
         android:id="@+id/cityTextView02"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"></TextView>
```

```
    <Spinner
        android:id="@+id/citySpinner01"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"></Spinner>
</LinearLayout>

<ScrollView
    android:id="@+id/ScrollView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

        <TextView
        android:text="@+id/displayTextView03"
        android:id="@+id/displayTextView03"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"></TextView>

    </ScrollView>
</LinearLayout>
```

④ 在该工程下创建一个 com.amaker.ch13.webservice 包，在该包中创建一个 Weather Activity 类，在该类顶部声明使用到的视图组件，并在 onCreate()方法中实例化它们。

```
package com.amaker.ch13.webservice;

import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.AdapterView.OnItemSelectedListener;
import com.amaker.ch13.R;

/**
 * @author 郭宏志
 * 显示天气预报
 */
public class WeatherActivity extends Activity {
    // 声明 TextView 对象
    private TextView displayTextView;
    // 声明 Spinner 对象
    private Spinner spinner;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置 Activity 界面布局
        setContentView(R.layout.weather);
        // 通过 findViewById 方法获得 TextView 实例
```

```
        displayTextView = (TextView) findViewById(R.id.displayTextView03);
        // 通过 findViewById 方法获得 Spinner 实例
        spinner = (Spinner) findViewById(R.id.citySpinner01);
    }
}
```

⑤ 在 com.amaker.ch13.webservice 包中创建一个 WebServiceUtil 工具类。在该类中声明一个 getCityList()方法，该方法通过使用 KSOAP 调用 http://WebXml.com.cn/网站的一个免费天气预报 Web Service。

```
    /*
     * 使用 ksoap，获得城市列表
     */
    public static List<String> getCityList() {
        // 命名空间
        String serviceNamespace = "http://WebXml.com.cn/";
        // 请求 URL
        String serviceURL = "http://webservice.webxml.com.cn/WebServices/WeatherWS.
asmx";
        // 调用的方法
        String methodName = "getRegionProvince";
        // 实例化 SoapObject 对象
        SoapObject request = new SoapObject(serviceNamespace, methodName);
        // 获得序列化的 Envelope
        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
                SoapEnvelope.VER11);
        envelope.bodyOut = request;
        (new MarshalBase64()).register(envelope);

        // Android 传输对象
        AndroidHttpTransport ht = new AndroidHttpTransport(serviceURL);
        ht.debug = true;

        try {
            // 调用
            ht.call("http://WebXml.com.cn/getRegionProvince", envelope);
            if (envelope.getResponse() != null) {
                return parse(envelope.bodyIn.toString());
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (XmlPullParserException e) {
            e.printStackTrace();
        }
        return null;
}
```

⑥ 创建一个 parse()方法对上述得到的 XML 信息进行解析。

```
    /*
     * 对得到的城市 XML 信息进行解析
     */
```

```
    private static List<String> parse(String str) {
        String temp;
        List<String> list = new ArrayList<String>();
        if (str != null && str.length() > 0) {
            int start = str.indexOf("string");
            int end = str.lastIndexOf(";");
            temp = str.substring(start, end - 3);
            String[] test = temp.split(";");
            for (int i = 0; i < test.length; i++) {
                if (i == 0) {
                    temp = test[i].substring(7);
                } else {
                    temp = test[i].substring(8);
                }
                int index = temp.indexOf(",");
                list.add(temp.substring(0, index));
            }
        }
        return list;
    }
}
```

⑦ 创建一个 getWeatherMsgByCity()方法，通过城市名称获得当前城市的天气预报信息。

```
    /*
     * 通过传递城市名称获得天气信息
     */
    public static String getWeatherMsgByCity(String cityName) {
        String url = "http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx/
getWeather";
        HttpPost request = new HttpPost(url);
        List<NameValuePair> params = new ArrayList<NameValuePair>();
        params.add(new BasicNameValuePair("theCityCode", cityName));
        params.add(new BasicNameValuePair("theUserID", ""));
        String result = null;
        try {
            UrlEncodedFormEntity entity = new UrlEncodedFormEntity(params,
                    HTTP.UTF_8);
            request.setEntity(entity);
            HttpResponse response = new DefaultHttpClient().execute(request);
            if (response.getStatusLine().getStatusCode() == 200) {
                result = EntityUtils.toString(response.getEntity());
                return parse2(result);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

⑧ 创建一个 parse2()方法对得到的天气预报 XML 信息进行解析。

```
    /*
     * 对天气信息 XML 文件进行解析
     */
    private static String parse2(String str){
```

```
        String temp;
        String[] temps;
        List list = new ArrayList();
        StringBuilder sb = new StringBuilder("");
        if(str!=null&&str.length()>0){
            temp = str.substring(str.indexOf("<string>"));
            temps = temp.split("</string>");
            for (int i = 0; i < temps.length; i++) {
                sb.append(temps[i].substring(12));
                sb.append("\n");
            }
        }
        return sb.toString();
    }
```

⑨ 将通过工具类 WebServiceUtil 的 getCityList()方法得到的城市信息列表添加到 Spinner 视图组件中。

```
        List<String> citys = WebServiceUtil.getCityList();
        ArrayAdapter a = new ArrayAdapter(this,
                android.R.layout.simple_spinner_dropdown_item, citys);
        spinner.setAdapter(a);    }
```

⑩ 为 Spinner 添加选项被选择事件，获得不同城市的天气信息。

```
        spinner.setOnItemSelectedListener(new OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1,
                    int arg2, long arg3) {
                String msg = WebServiceUtil.getWeatherMsgByCity(spinner.getSelected-
Item(). toString());
                displayTextView.setText(msg);
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0) {

            }
        });
```

程序运行结果如图 13.6 所示。

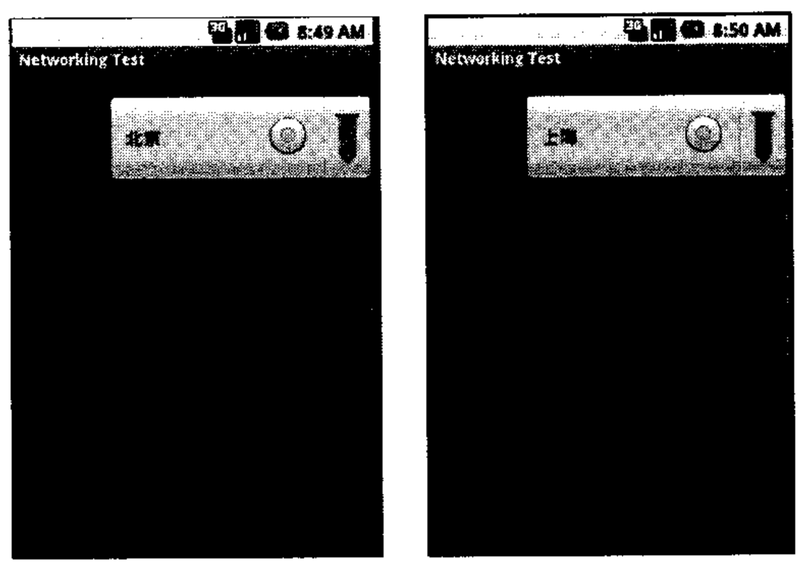


图 13.6　Web Service 天气预报

13.5 直接使用 WebView 视图组件显示网页

Android 提供了内置的浏览器，该浏览器使用了开源的 WebKit 引擎。在 Android 中使用浏览器需要通过 WebView 视图组件来实现。WebView 提供了一些浏览器方法。例如，使用 loadUrl()可以直接打开一个 Web 地址页面；使用 loadData 可以显示 HTML 格式的网页内容。

13.5.1　使用 WebView 打开网页

创建了 WebView 视图组件，就可以直接使用 WebView 的 loadUrl()方法来显示该 URL 内容。下面的实例就演示了该功能。

① 在“Chapter13_Networking_Client”工程的 res\layout\目录下，创建一个 test_webview.xml 布局文件，在该布局文件中添加一个 WebView 视图组件。

```
<?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"
    >
     <WebView
        android:id="@+id/mywebview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>
```

② 在“Chapter13_Networking_Client”工程中创建 com.amaker.ch13.webview 包。在该包中创建一个“TestWebViewActivity”。

```
package com.amaker.ch13.webview;

import android.app.Activity;
import android.os.Bundle;
import android.webkit.WebView;
import com.amaker.ch13.R;
/**
 * @author 郭宏志
 * 通过WebView浏览网络
 */
public class TestWebViewActivity extends Activity {
    private WebView webView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.test_webview);
```

```
        webView = (WebView)findViewById(R.id.mywebview);
        String url = "http://www.google.com";
        webView.loadUrl(url);
    }
}
```

程序运行结果如图 13.7 所示。

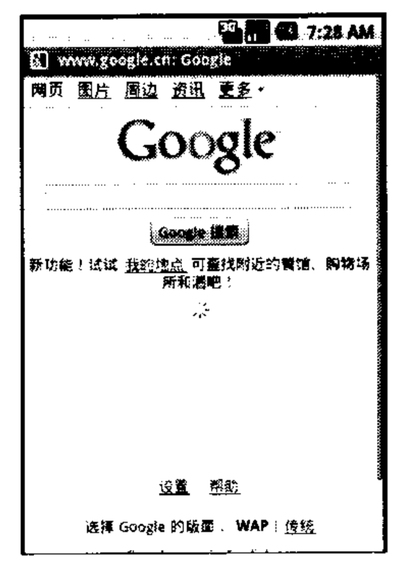


图 13.7 使用 WebView 浏览网络

13.5.2 使用 WebView 加载 HTML

以上实例是通过 WebView 的 loadUrl()方法来访问网页的，我们也可以使用 loadData()方法传递静态 HTML 页面来显示网页内容。下面的实例将上面的实例进行改编，改编后的内容如下所示。

```
package com.amaker.ch13.webview;

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

import com.amaker.ch13.R;
/**
 * @author 郭宏志
 * 通过 WebView 浏览网络
 */
public class TestWebViewActivity extends Activity {
    private WebView webView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.test_webview);
        webView = (WebView)findViewById(R.id.mywebview);
       /* String url = "http://www.google.com";
        webView.loadUrl(url);*/

        String html = "";
        html+="<html>";
            html+="<body>";
            html+="<a href=http://www.google.com>Google Home</a>";
            html+="</body>";
        html+="</html>";

        webView.loadData(html, "text/html", "utf-8");

    }
}
```

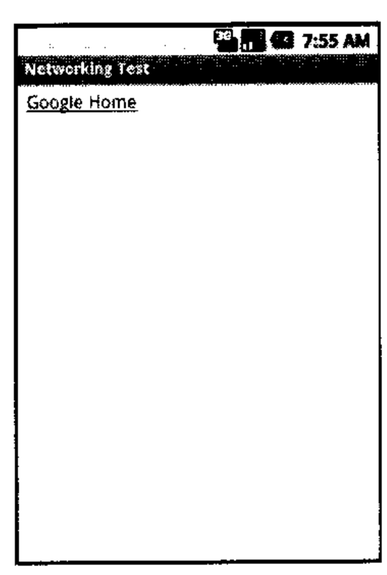
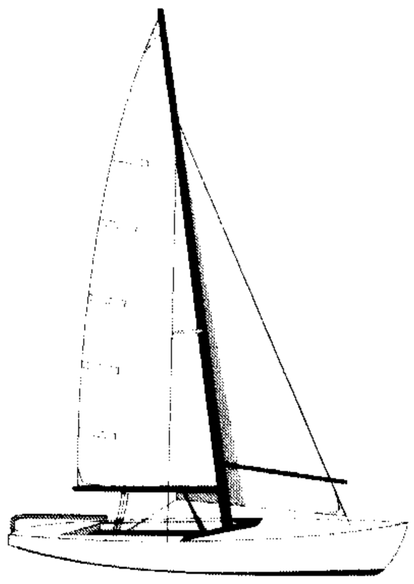


图 13.8 使用 WebView 浏览网络

该方法的第一个参数是 html 类型，第二个参数是 mine 类型，第三个参数是字符编码类型。

程序运行结果如图 13.8 所示。



第 14 章　Android 中的 GPS 应用

GPS（Global Position System，全球定位系统）是 20 世纪 70 年代由美国陆海空三军联合研制的新一代空间卫星导航定位系统。

24 颗 GPS 卫星在离地面 1 万 2 千公里的高空上，以 12 小时的周期环绕地球运行，使得在任意时刻，在地面上的任意一点都可以同时观测到 4 颗以上的卫星。

由于卫星的位置精确，在 GPS 观测中，我们可以得到卫星到接收机的距离，利用三维坐标中的距离公式和 3 颗卫星，就可以组成 3 个方程式，解出观测点的位置（*X*,*Y*,*Z*）。考虑到卫星的时钟与接收机时钟之间的误差，实际上有 4 个未知数，*X*、*Y*、*Z* 和钟差，因而需要引入第 4 颗卫星，形成 4 个方程式进行求解，从而得到观测点的经纬度和高程。

2005 年 2 月 Google 推出了 Google Maps，该服务为 Google 的搜索服务加重了砝码。Android 出自 Google 之手，那么将 GPS 应用在 Android 中自然不在话下。

在 Android 中可以通过 MapActivity 和 MapView 直接调用 Google Map，也可以进行地图的缩放、查找、定位等各项功能。

通过本章的学习我们将掌握如下内容：

- 通过模拟器来测试定位服务
- 跟踪和查找设备
- 创建趋近警告
- 地址和经纬度之间的相互解码

14.1 LocationManager 和 LocationProvider 简介

在 Android 的位置服务中，两个重要类是 LocationManager 和 LocationProvider。LocationManager 位于 android.location 包中，该类提供了系统位置访问的方法。Location Provider 定义了位置服务的提供方法，例如，是由 GPS 设备提供还是通过网络提供等。

14.1.1 LocationManager

通过 LocationManager 可以实现设备的定位、跟踪和趋近提示。可以通过 Context.getSystemService(Context.LOCATION_SERVICE)方法获得该类的实例。该类的常用属性和方法如表 14.1 所示。

表 14.1 LocationManager 常用属性和方法

属性或方法名称	属性或方法描述
GPS_PROVIDER	静态字符串常量，表明 LocationProvider 是 GPS
NETWORK_PROVIDER	静态字符串常量，表明 LocationProvider 是网络
addGpsStatusListener(GpsStatus.Listener listener)	添加一个 GPS 状态监听器
addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)	添加一个趋近警告
getAllProviders()	获得所有 LocationProvider 列表
getBestProvider(Criteria criteria, boolean enabledOnly)	根据 Criteria 返回最适合的 LocationProvider
getLastKnownLocation(String provider)	根据 Provider 获得位置信息
getProvider(String name)	获得指定名称的 LocationProvider
getProviders(boolean enabledOnly)	获得可利用的 LocationProvider 列表
removeProximityAlert(PendingIntent intent)	删除趋近警告
requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)	通过给定的 Provider 名称，周期性地通知当前 Activity

14.1.2 LocationProvider

LocationProvider 用来描述位置提供者，设置位置提供者的一些属性。可以通过 Criteria 类来为 LocationProvider 设置条件，获得合适的 LocationProvider。LocationProvider 的相关属性和方法如表 14.2 所示。

表 14.2 LocationProvider 常用属性和方法

属性或方法名称	属性或方法描述
AVAILABLE	静态整形常量，标示是否可利用
OUT_OF_SERVICE	静态整形常量，不在服务区
TEMPORARILY_UNAVAILABLE	静态整形常量，临时不可利用
getAccuracy()	获得精度
getName()	获得名称
getPowerRequirement()	获得电源需求
hasMonetaryCost()	花钱的还是免费的
requiresCell()	是否需要访问基站网络

续表

属性或方法名称	属性或方法描述
requiresNetwork()	是否需要 Internet 网络数据
requiresSatellite()	是否需要访问卫星
supportsAltitude()	是否能够提供高度信息
supportsBearing()	是否能够提供方向信息
supportsSpeed()	是否能够提供速度信息

14.2 通过模拟器测试位置服务

在真实的设备当中要想获得位置服务需要有 GPS 硬件的支持，但是在开发和测试当中使用模拟器是最为方便的。Android 的 Eclipse 开发插件 ADT 提供了支持。我们可以将 Eclipse 视图模式切换到 DDMS 模式，在 Emulator Control 中可以找到位置服务选项，该选项提供了手动发送经纬度、GPX 和 KML 格式数据来测试位置服务。如图 14.1 所示是模拟器控制台的可视化界面。

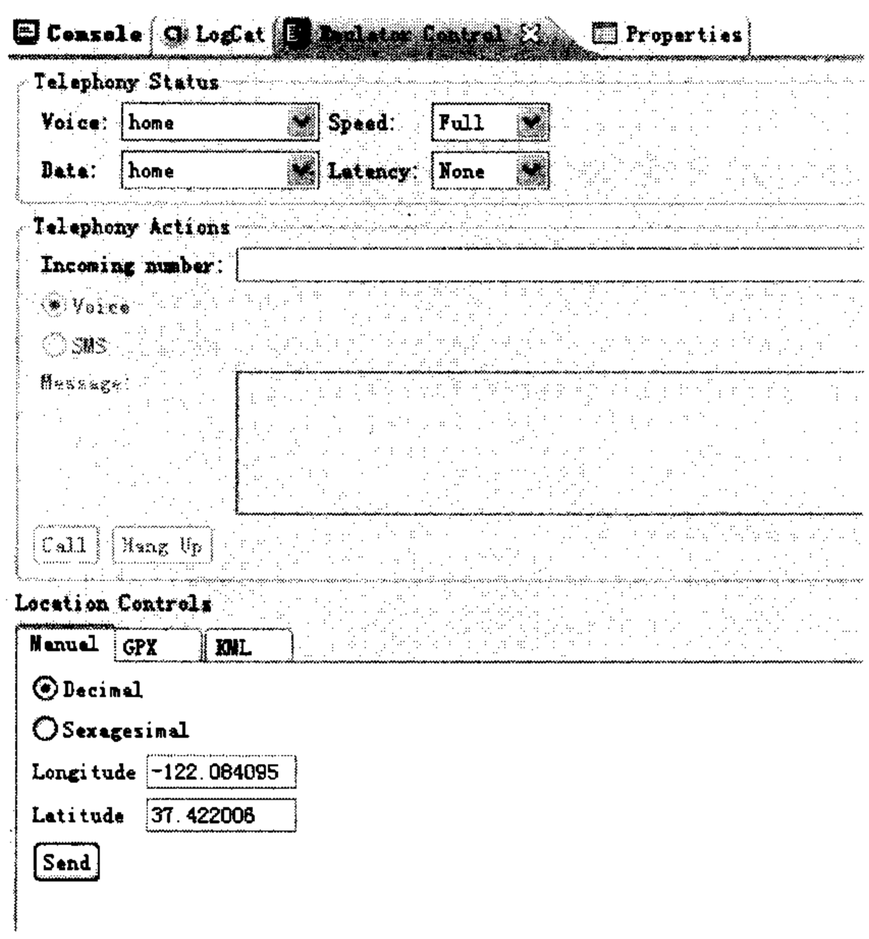


图 14.1　位置服务模拟测试

在本节的实例中，我们创建一个项目，通过模拟器发送经纬度来测试位置服务。要完成该实例需要的步骤说明如下。

① 创建一个项目 Chapter14_Map_01，注意这里通过 Eclipse 创建项目时，target build 要选择 Google APIs，这是为了添加选项 jar 文件 maps.jar。MainActivity 的内容如下所示。

```
package com.amaker.test;
```

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

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
    }
}
```

② 引入 LocationManager 实例。

```
   // 声明定位管理器 LocationManager
   private LocationManager locationManager;
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
      // 通过 getSystemService 方法获得 LocationManager 实例
       locationManager    =    (LocationManager)    getSystemService(Context.LOCATION_
SERVICE);
   }
```

③ 在布局文件 main.xml 中添加一个 TextView 组件。

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

<TextView
android:text="@+id/myTextView01"
android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>

</LinearLayout>
```

④ 创建一个定位方法 locate()。

```
    private void locate(){
    // 通过 findViewById 方法获得 TextView 实例
    TextView tv = (TextView)findViewById(R.id.myTextView01);
    // 实例化 StringBuilder
    StringBuilder builder = new StringBuilder("可利用的providers:");
        // 获得获得 Provider 列表
        List<String> providers = locationManager.getProviders(true);
        // 声明位置服务监听器
        LocationListener ll = new LocationListener(){
          // 位置改变时调用
          public void onLocationChanged(Location location) {
```

```
        }
        // Provider 失效时调用
        public void onProviderDisabled(String provider) {
        }

        // Provider 生效时调用
        public void onProviderEnabled(String provider) {
        }

        // 状态改变时调用
        public void onStatusChanged(String provider, int status,
                Bundle extras) {
        }
    };
    // 循环 Provider ，根据 Provider 获得位置信息（经纬度）
    for(String provider:providers){
    // 请求位置更新
    locationManager.requestLocationUpdates(provider, 0, 1000, 11 );
    // 添加 Provider 到 StringBuilder
    builder.append("\n").append(provider).append(":");
    // 获得 Location
    Location location = locationManager.getLastKnownLocation(provider);
    if(location!=null){
        // 获得经度
        double lat = location.getLatitude();
        // 获得纬度
        double lng = location.getLongitude();
        // 添加经纬度到 StringBuilder
        builder.append("(");
        builder.append(lat);
        builder.append(",");
        builder.append(lng);
        builder.append(")");
}else{
        builder.append("没有位置信息");
    }
    }
    // 在文本视图显示
    tv.setText(builder);
}
```

⑤ 在 AndroidManifest.xml 配置文件中添加权限。

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

⑥ 在 MainActivity 的 onCreate()方法中调用 locate()方法，程序运行结果如图 14.2 所示。

图 14.2　获得设备经纬度

14.3 获得 LocationProvider

14.3.1 通过名称获得 LocationProvider

根据设备，可以使用不同的定位技术来实现位置服务，就是不同的 LocationProvider，不同的 LocationProvider 决定了定位能力，包括费用、耗电多少、精确度，还会有一些方向、速度和高度等信息。

可以通过指定名称来获得 LocationProvider 实例。

```
// 通过 getSystemService 方法获得 LocationManager 实例
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
// Provider 名称常量
String name = LocationManager.GPS_PROVIDER;
// 根据 Provider 名称获得 LocationProvider
LocationProvider myProvider;
myProvider = locationManager.getProvider(name);
```

14.3.2 获得当前可利用的 LocationProvider

可以根据 LocationManager 中的静态常量 GPS_PROVIDER 和 NETWORK_PROVIDER 来获得 gps Provider 和 Network Provider。另外，可以调用 locationManager.getProviders (enabledOnly)方法获得可利用的 LocationProvider 实例。

```
// 根据 getSystemService 方法获得 LocationManager 实例
LocationManager locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
// LocationProvider 的两种方式
String name = LocationManager.GPS_PROVIDER;
String name1 = LocationManager.NETWORK_PROVIDER;
// 根据 Provider 名称获得 LocationProvider
LocationProvider myProvider;
myProvider = locationManager.getProvider(name);

boolean enabledOnly = true;
// 获得所有可利用的 Provider 名称列表
List<String> providers = locationManager.getProviders(enabledOnly);
// 根据名称获得 Provider
myProvider = locationManager.getProvider(name);
```

14.3.3 根据 Criteria 条件获得 LocationProvider

Criteria 封装了获得 LocationProvider 的条件，可以根据指定的 Criteria 条件来过滤获得 LocationProvider。Criteria 的常用属性和方法如表 14.3 所示。

表 14.3　Criteria 常用属性和方法

属性或方法名称	属性或方法描述
ACCURACY_COARSE	粗糙精确度
ACCURACY_FINE	较高精确度
POWER_HIGH	用电高
POWER_LOW	用电低
isAltitudeRequired()	返回 Provider 是否需要高度信息
isBearingRequired()	返回 Provider 是否需要方位信息
isSpeedRequired()	返回 Provider 是否需要速度信息
isCostAllowed()	是否允许产生费用
setAccuracy(int accuracy)	设置 Provider 的精确度
setAltitudeRequired(boolean altitudeRequired)	设置 Provider 是否需要高度信息
setBearingRequired(boolean bearingRequired)	设置 Provider 是否需要方位信息
setSpeedRequired(boolean speedRequired)	设置 Provider 是否需要速度信息
setCostAllowed(boolean costAllowed)	设置是否需要产生费用
getAccuracy()	获得精确度

下面的代码描述了根据 Criteria 条件获得 LocationProvider。

```
// LocationProvider 的查询条件实例
Criteria criteria = new Criteria();
// 设置精确度
criteria.setAccuracy(Criteria.ACCURACY_COARSE);
// 设置电耗
criteria.setPowerRequirement(Criteria.POWER_LOW);
// 是否需要高度信息
criteria.setAltitudeRequired(false);
// 是否需要方位信息
criteria.setBearingRequired(false);
// 是否需要速度信息
criteria.setSpeedRequired(false);
// 是否产生费用
criteria.setCostAllowed(true);
// 获得符合条件最好的 Provider
String bestProvider = locationManager.getBestProvider(criteria, true);
// 获得符合条件的 Provider
List<String> matchingProviders = locationManager.getProviders(criteria,
    false);
```

14.4 定位和跟踪

如果你到了一个陌生的地方迷了路怎么办呢？打开手机的位置服务确定一下自己的方位。又如你的手机丢了，想知道它丢哪里了，也可以通过手机的位置服务确定一下。在这一节里我们将讲述如何实现定位和跟踪。

14.4.1 定位

定位就是确定设备的位置，在这里除了用到了 LocationManager、LocationProvider 还用到了 Location 类，该类描述了当前设备的地理位置信息，包括经纬度、方向、高度和速度等。通过 LocationManager 的 getLastKnownLocation(String provider)方法可以获得 Location 实例。

Location 的常见属性和方法如表 14.4 所示。

表 14.4 Location 常用属性和方法

属性或方法名称	属性或方法描述
getLongitude()	获得经度
getLatitude()	获得纬度
getAccuracy()	获得精确度
getAltitude()	获得高度
getBearing()	获得方向
getSpeed()	获得速度

下面的实例演示了如何定位当前设备，并显示详细信息。

```
package com.amaker.test;

import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
   // 声明定位服务管理器 LocationManager 实例
   private LocationManager locationManager;
   // 声明文本视图 TextView 实例
   private TextView tv;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置当前 Activity 界面布局
      setContentView(R.layout.main);
      // 通过 getSystemService 方法获得 LocationManager
      locationManager = (LocationManager) getSystemService(Context.LOCATION_ SERVICE);
      // 通过 findViewById 方法获得 TextView 实例
      tv = (TextView) findViewById(R.id.myTextView01);
      // 获得位置信息
      Location location = locationManager
```

```
            .getLastKnownLocation(LocationManager.GPS_PROVIDER);
        // 显示位置信息
        printMsg(location);
    }
    // 显示位置信息方法
    private void printMsg(Location l) {
        // 声明 StringBuilder
        StringBuilder builder = new StringBuilder("可利用的 providers:");

        if (l != null) {
            // 获得经度
            double lat = l.getLatitude();
            // 获得纬度
            double lng = l.getLongitude();
            builder.append("(");
            builder.append(lat);
            builder.append(",");
            builder.append(lng);
            builder.append(")");

            if(l.hasAccuracy()){
                builder.append("\n 精度: ");
                builder.append(l.getAccuracy());
            }

            if(l.hasAltitude()){
                builder.append("\n 高度: ");
                builder.append(l.getAltitude());
            }

            if(l.hasBearing()){
                builder.append("\n 方向: ");
                builder.append(l.getBearing());
            }

            if(l.hasSpeed()){
                builder.append("\n 速度: ");
                builder.append(l.getSpeed());
            }

        } else {
            builder.append("没有位置信息");
        }
        tv.setText(builder);
    }

}
```

图 14.3　定位结果

程序运行结果如图 14.3 所示。

14.4.2 跟踪

跟踪是通过注册监听器来实现的。声明 LocationListener 监听器，在 onLocationChanged (Location l)方法中实现位置变化的代码。通过 LocationManager 的 requestLocationUpdates() 方法注册监听器，当设备位置发生变化时监听器被触发，onLocationChanged(Location l)方法被调用。

下面的代码实现了设备的实时跟踪。

```
package com.amaker.test;

import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends Activity {
    // 声明位置服务管理器 LocationManager 实例
    private LocationManager locationManager;
    // 声明文本视图 TextView 实例
    private TextView tv;
    // 声明 StringBuilder
    private StringBuilder builder = new StringBuilder("位置信息:\n");

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.main);
        // 通过 getSystemService 方法获得 LocationManager 实例
        locationManager = (LocationManager) getSystemService(Context.LOCATION_ SERVICE);
        // 通过 findViewById 方法获得 TextView 实例
        tv = (TextView) findViewById(R.id.myTextView01);
        // 获得 GPS Provider
        String provider = LocationManager.GPS_PROVIDER;
        // 获得位置信息
        Location location = locationManager
                .getLastKnownLocation(provider);
        // 更新信息
        updateMsg(location);
        // 声明位置监听器
        LocationListener ll = new LocationListener(){
            // 位置更新时调用
            public void onLocationChanged(Location location) {
                updateMsg(location);
```

```
            }
            // Provider 失效时调用
            public void onProviderDisabled(String provider) {
            }

            // Provider 生效时调用
            public void onProviderEnabled(String provider) {
            }

            // 状态改变时调用
            public void onStatusChanged(String provider, int status,
                    Bundle extras) {
            }
        };
        // 请求位置更新
        locationManager.requestLocationUpdates(provider, 2000, 10,
                ll);

    }
    // 更新信息
    private void updateMsg(Location l) {

        if (l != null) {
            // 获得经度
            double lat = l.getLatitude();
            // 获得纬度
            double lng = l.getLongitude();
            builder.append("(");
            builder.append(lat);
            builder.append(",");
            builder.append(lng);
            builder.append(")");

            if(l.hasAccuracy()){
                builder.append("\n 精度: ");
                builder.append(l.getAccuracy());
            }

            if(l.hasAltitude()){
                builder.append("\n 高度: ");
                builder.append(l.getAltitude());
            }

            if(l.hasBearing()){
                builder.append("\n 方向: ");
                builder.append(l.getBearing());
            }

            if(l.hasSpeed()){
                builder.append("\n 速度: ");
                builder.append(l.getSpeed());
            }
```

```
            builder.append("\n");

        } else {
            builder.append("没有位置信息");
        }
        tv.setText(builder);
    }

}
```

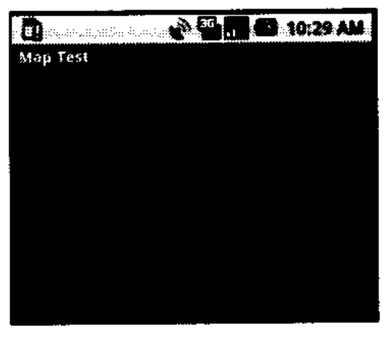


图 14.4　跟踪结果

程序运行结果如图 14.4 所示。

14.5 趋近警告

如果有这样一种位置服务，能够对我们进入或退出某个设定的区域进行提示，这样的应用是很有意义的。LocationManager 类提供这一功能，LocationManager 实例提供了一个方法 addProximityAlert (double latitude, double longitude, float radius, long expiration, PendingIntent intent)，该方法用来实现这一功能。

addProximityAlert 方法中需要提供 5 个参数：前两个是经纬度；第三个是区域半径；第四个是是否过期；第五个一般是一个广播 PendingIntent。

要实现这一功能需要两个步骤：一是获得 LocationManager 实例，调用其方法 addProximityAlert 并添加趋近提示；二是定义一个广播接收器，当设备进入设定区域时提醒用户。下面是程序的实现步骤。

① 创建一个 Android 工程，工程名称为“Chapter14_Map_04”。

② 在布局文件中声明一个 TextView 视图组件和一个 Button 组件，TextView 用来提示，Button 用来设定趋近提示。

```
<?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="测试趋近位置服务"
    />

<Button
    android:text="设定"
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 在 MainActivity 类中声明一个方法 set，该方法获得 LocationManager 实例，并且调用其方法 addProximityAlert 设定趋近提示。

```
// 趋近提示方法
private void set() {
    // 定位服务常量
    String locService = Context.LOCATION_SERVICE;
    // 定位服务管理器实例
    LocationManager locationManager;
    // 通过 getSystemService 方法获得 LocationManager 实例
    locationManager = (LocationManager)getSystemService(locService);
    // 声明经度
    double lat = 37.4;
    // 声明纬度
    double lng = 55.0;
    // 声明半径（单位米）
    float radius = 200f;
    // 不过期
    long expiration = -1;
    // 声明 Intent
    Intent intent = new Intent(PROXIMITY_ALERT_ACTION_NAME);
    // 声明 PendingIntent
    PendingIntent pi =
    PendingIntent.getBroadcast(this, -1,intent,0);
    // 添加趋近警告
    locationManager.addProximityAlert(lat, lng, radius, expiration,pi);
}
```

④ 在 MainActivity 类的 onCreate 方法中获得 Button 实例，为其添加单击事件监听器，在事件方法中调用 set 方法，设定趋近提示。

```
public class MainActivity extends Activity {
    // Intent Action 常量
    private    static    String    PROXIMITY_ALERT_ACTION_NAME    =    "com.amaker.ch14.
ProximityAlert";
    // 声明按钮 Button 实例
    private Button btn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.main);
        // 通过 findViewById 方法获得 Button 实例
        btn = (Button)findViewById(R.id.Button01);
        btn.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 设定
                set();
            }
        });
    }
```

⑤ 定义广播接收器，用来检测当设备进入设定区域时提示用户。

```
package com.amaker.ch14.app01;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.location.LocationManager;
import android.widget.Toast;

public class ProximityAlertReciever extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {
        // 趋近关键字
        String key = LocationManager.KEY_PROXIMITY_ENTERING;
        // 从 Intent 获得额外信息，判断是否进入设置区域
        Boolean isEnter = intent.getBooleanExtra(key, false);
        if(isEnter){
            Toast.makeText(context, "你已经进入海淀区！", Toast.LENGTH_LONG);
        }
    }
}
```

⑥ 在 AndroidManifest.xml 配置文件中声明广播接收器。

```
        <receiver android:name="ProximityAlertReciever">
        <intent-filter>
            <action
android:name="com.amaker.ch14.ProximityAlert" />
            <category
android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        </receiver>
```

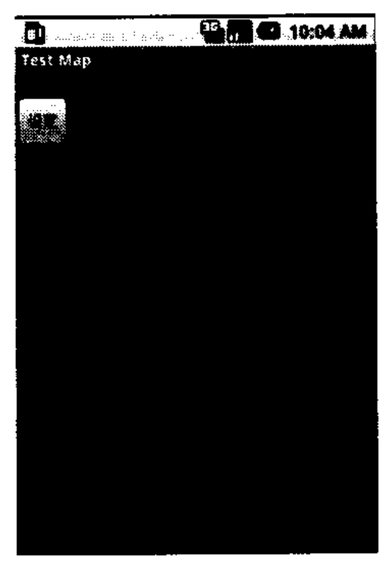


图 14.5　趋近提示

程序运行结果如图 14.5 所示。

14.6 Geocoder 正逆向编解码

Geocoder 可以完成位置信息和经纬度坐标之间的相互转换。例如，你知道了某个地方的名称，想知道它的经纬度坐标，或者你知道了某个地方的经纬度坐标，想知道其名称，那么 Geocoder 就可以完成这一功能了。

解码方式有以下两种：

- 正向编码：通过位置名称获得经纬度坐标。
- 反向编码：通过经纬度坐标获得位置名称。

获得 Geocoder 需要两个参数：一个是上下文 Context 对象；另一个是 Locale 对象。

```
        Geocoder coder =
            new Geocoder(getApplicationContext(),Locale.getDefault());
```

14.6.1　正向编码

正向编码是通过位置名称（如街道名称等）来获得经纬度坐标。可以通过调用 Geocoder 的 public List<Address> getFromLocationName (String locationName, int maxResults)方法来获得 Address 实例列表，该方法需要两个参数：一个是位置名称；另一个是最大结果数。下面通过实例来演示如何通过 Geocoder 实现正向编码。

① 创建一个工程，名称为“Chapter14_Map_05”。

② 在布局文件 main.xml 中，添加一个 TextView 用来显示位置信息，添加两个 Button 用来设置正向、反向编码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:text="显示位置信息："
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></TextView>

<Button
    android:text="正向编码"
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

<Button
    android:text="反向编码"
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>

</LinearLayout>
```

③ 在 MainActivity 中创建一个方法 forward，在该方法中实例化 Geocoder，调用该实例的 getFromLocationName 方法获得 Address 实例列表，通过 Address 实例获得经纬度信息，并在 TextView 中显示。

```
    // 正向编码
    private void forward() {
    // 实例化 Geocoder
    Geocoder gc = new Geocoder(this, Locale.getDefault());
    // 地址
    String address = "北京天安门";
    // 位置列表
```

```
List<Address> locations = null;
try {
    // 获得位置类别
    locations = gc.getFromLocationName(address, 10);
    // 如果 Locations 不为空
    if(locations.size()>0){
        // 获得 Address 实例
        Address a = locations.get(0);
        // 获得经纬度
        double lat = a.getLatitude();
        double lng = a.getLongitude();
        // 显示信息
        tv.setText(lat+":"+lng);
    }
  } catch (IOException e) {}
 }
```

④ 在 MainActivity 的 onCreate 方法中，获得 TextView 和 Button 实例，设置 Button 的单击事件监听器，在事件方法中调用 forward 方法。

```
package com.amaker.ch14.app02;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import android.app.Activity;
import android.content.Context;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    // 声明 TextView
    private TextView tv;
    // 声明 Button
    private Button b1,b2;
    // 定位服务管理器实例
    private LocationManager locationManager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
    // 设置当前 Activity 的界面布局
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 定位服务常量
        String locService = Context.LOCATION_SERVICE;
```

```
        // 通过 getSystemService 方法获得 LocationManager 实例
        locationManager = (LocationManager)getSystemService(locService);
        // 通过 findViewById 方法获得 TextView 实例
        tv = (TextView) findViewById(R.id.TextView01);

        // 通过 findViewById 方法获得 Button 实例
        b1 = (Button) findViewById(R.id.Button01);
        // 通过 findViewById 方法获得 Button 实例
        b2 = (Button) findViewById(R.id.Button02);
        // 为按钮添加单击事件监听器
        b1.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // 正向编码
                forward();
            }
        });
    }
}
```

⑤ 在 AndroidManifest.xml 配置文件中声明权限。

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

程序运行结果如图 14.6 所示。

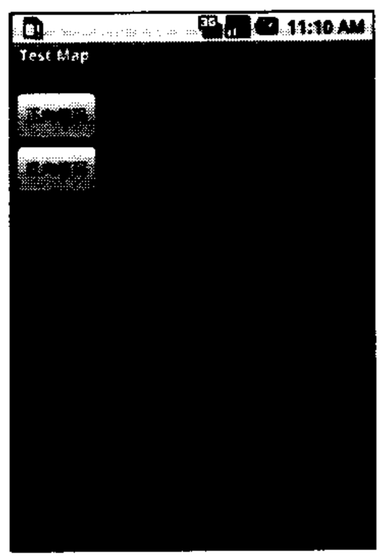


图 14.6　Geocoder 正向编码

14.6.2　反向编码

反向编码是通过经纬度来获得具体的位置信息。可以通过 Geocoder 的 public List<Address> getFromLocation (double latitude, double longitude, int maxResults)方法获得 Address 实例列表。该方法需要传递三个参数，分别是经纬度和最大结果数。下面的实例演示了如何通过经纬度坐标获得位置信息。

① 在以上工程的基础上创建一个 reserse 方法，在该方法中获得 Geocoder 实例，调用该实例的 getFromLocation 方法获得 Address 实例列表，通过 Address 实例获得位置信息。

```
    // 反向编码
    private void reverse() {
        // 经度
        double lng = 116.46;
        // 纬度
        double lat = 39.92;
        // 实例化 Geocoder
        Geocoder gc = new Geocoder(this, Locale.getDefault());
        // 声明地址列表
        List<Address> addresses = null;
        try {
            // 获得 Address 实例列表
            addresses = gc.getFromLocation(lat, lng, 10);
```

```
            // 声明 StringBuilder，保存位置信息
            StringBuilder sb = new StringBuilder();
            // 判断 addresses 是否为空
            if(addresses.size()>0){
                // 获得 Address
                Address a = addresses.get(0);
                for (int i = 0; i < a.getMaxAddressLineIndex(); i++){
                    // 地址
                    sb.append(a.getAddressLine(i)).append("\n");
                    // 地点
                    sb.append(a.getLocality()).append("\n");
                    // 邮编
                    sb.append(a.getPostalCode()).append("\n");
                    // 国家名称
                    sb.append(a.getCountryName());
                }
                tv.setText(sb.toString());
            }
        } catch (IOException e) {}
    }
```

② 在 MainActivity 的 onCreate 方法中，为 b2 按钮添加单击事件监听器，在事件方法中调用 reverse 方法。

```
    // 为按钮添加单击事件监听器
    b2.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            // 反向编码
            reverse();
        }
    });
```

③ 程序运行结果如图 14.7 所示。

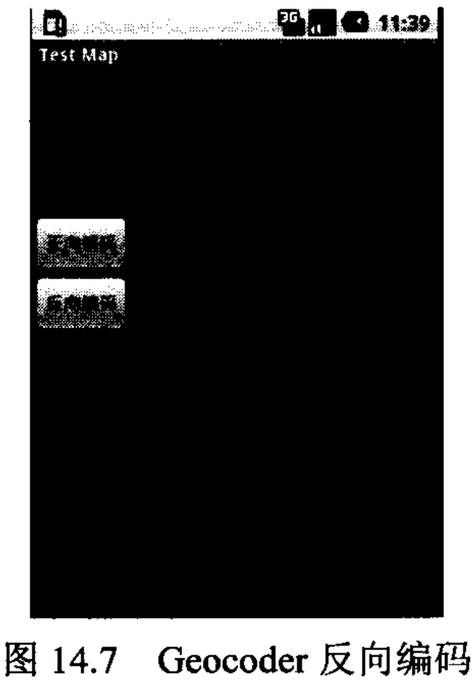
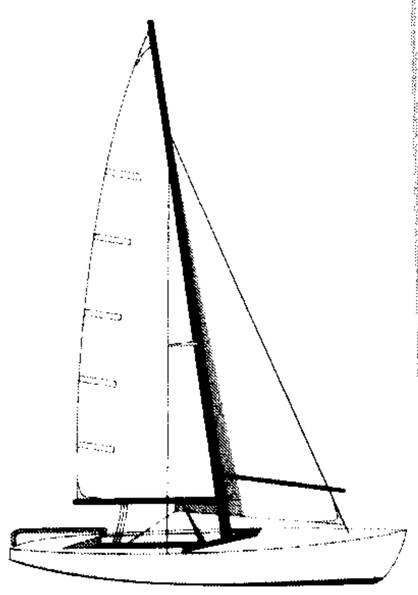


图 14.7　Geocoder 反向编码

第三篇

应 用 篇

- 第 15 章　Android 应用案例——移动警务通
- 第 16 章　Android 应用案例——雷电游戏
- 第 17 章　Android 应用案例——备忘录
- 第 18 章　Android 应用案例——无线点餐系统



第 15 章　Android 应用案例——移动警务通

从本章开始我们将进入案例实战阶段，通过真实的商业案例来综合学习 Android 中的内容。本案例（移动警务通）是笔者在 5 年前做开发时的一个真实案例。当然，在那时还没有 Android，当时项目采用的技术架构是客户端 Java ME+Web 服务+服务器端 Java EE 技术实现的。现在有了 Android 所以笔者就把它移植到了 Android 中，而且使用更方便，功能更强大。

从应用的角度来讲，Android 和 Java ME 有一些相似，二者都主要是应用在一些移动增值服务上，目前 Java ME 应用最多的是手机游戏。但是 Android 比 JavaME 更强大、编程更方便，尤其体现在网络服务方面和位置服务方面。在网络服务方面 Android 集成了 Apache 的 HTTP 客户端，这使得直接使用 HTTP 成为可能。在位置服务方面 Android 直接集成了 Google 地图服务，可以非常方便地完成地图的应用。

本系统将从需求分析、总体设计、详细设计和系统编码等方面，详细介绍移动警务通的应用、功能及具体实现。

15.1 移动警务通需求分析

随着信息科技的高速发展，特别是网络技术和移动通信技术在全世界的推广，人们获取信息的手段发生了巨大的变化，尤其是那些对实时信息要求强烈、移动性强的行业，移动计算和无线数据技术将为他们的工作带来崭新的体验以及前所未有的效率。

公安行业是一个具有工作移动性、突发性、紧急性强的行业，一线部门需要实时和公安数据中心交换信息，对在逃人员、常住人口、暂住人口、重点人口、车辆等信息进行及时、迅速的查询，特别是照片和相关图片的传输应用，能够极大地提高公安人员的办案效率。

目前无论是公安刑警、巡警、交警都已具备基本的综合管理系统，如以 CCIC 为核心

的“金盾工程”、户籍管理系统、案/事件业务、交管业务、监管业务、旅店业登记、出入境、进口机动车核查、指纹处理、综合应用等综合信息系统。但这些都仅仅限于桌面应用，不能够解决公安行业移动性、突发性、紧急性的工作特点之要求，极大地限制了现有系统的使用效率。

针对以上分析，根据“数字警察”的理念，本系统采用先进的移动技术，提出了结合 GPRS、CDMA、3G 等多种无线网络及多种移动终端的移动警务通系统解决方案，实现公安系统移动互联及相关的业务应用，将现有系统的功能通过移动互联技术扩展到每个警员的掌中，做到随时、随地、任意使用综合信息资源，不仅提高现有系统的使用频率，而且大大提高警员的工作效率，为公安行业提升自己的社会和经济效益提供有力的技术保障。

15.2 移动警务通总体设计

为了满足系统的需求，我们将从系统架构、技术选型、系统功能几个方面介绍该系统。

15.2.1　系统架构

系统的物理架构是这样的：客户端 Android 系统智能手机通过移动信号塔访问移动警务服务器，如果需要数据访问，则访问后台数据库。

系统物理结构如图 15.1 所示。

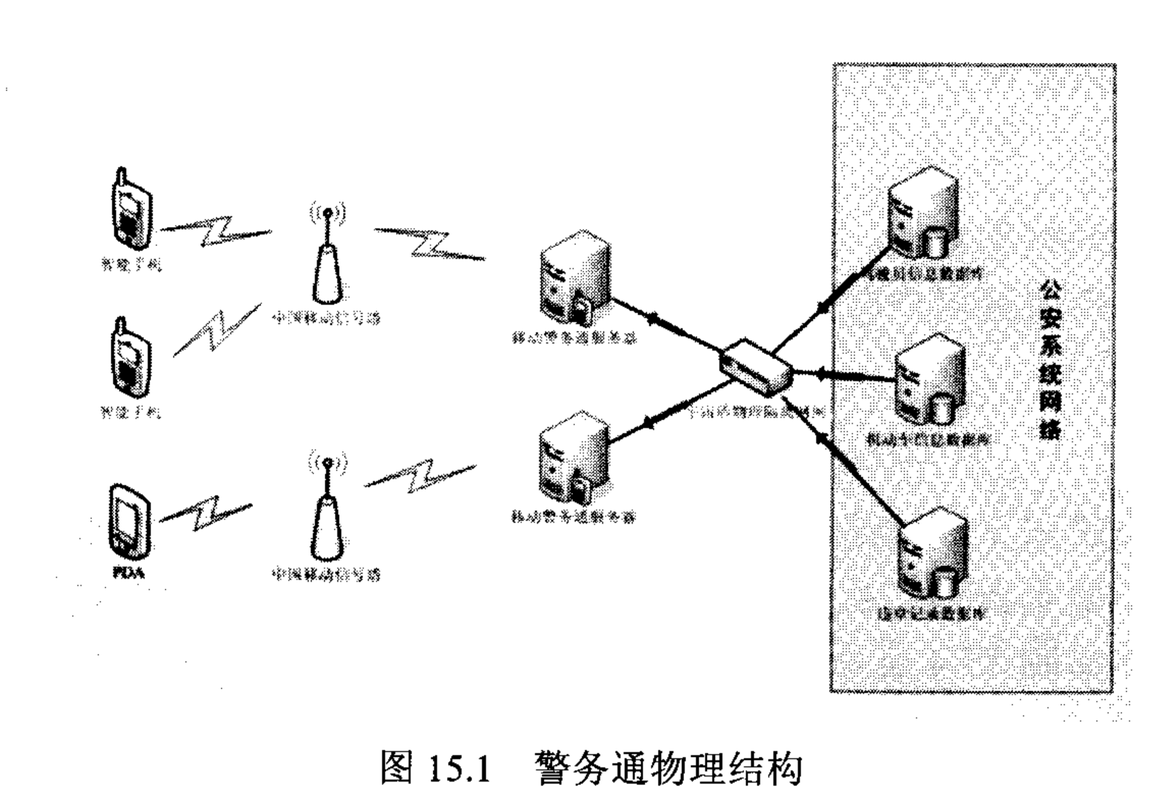


图 15.1　警务通物理结构

15.2.2　技术选型

Android 客户端应用使用 Java 技术，网络通信使用 Apache HTTP 客户端，中间 Web 服务器采用 Servlet 响应客户请求，后台数据库采用 JDBC 访问数据库。

其实在 Web 服务器层，我们可以使用 Java EE 现在流行的任何框架，例如可以使用 Struts2+Spring+Hibernate，但是考虑到 Servlet 和 JDBC 更具有一般意义，所以本系统采用 Servlet+JDBC 实现，毕竟这不是一本讲述 Java EE 的书籍。

Web 服务器我们采用开源的 Tomcat，数据库我们采用 MySQL。实际项目中的服务器可能要采用商业的 WebLogic 或者 WebSphere，当然也可以使用 JBoss，数据库要采用 Oracle，不过这种移植也很方便。图 15.2 描述了技术选型。

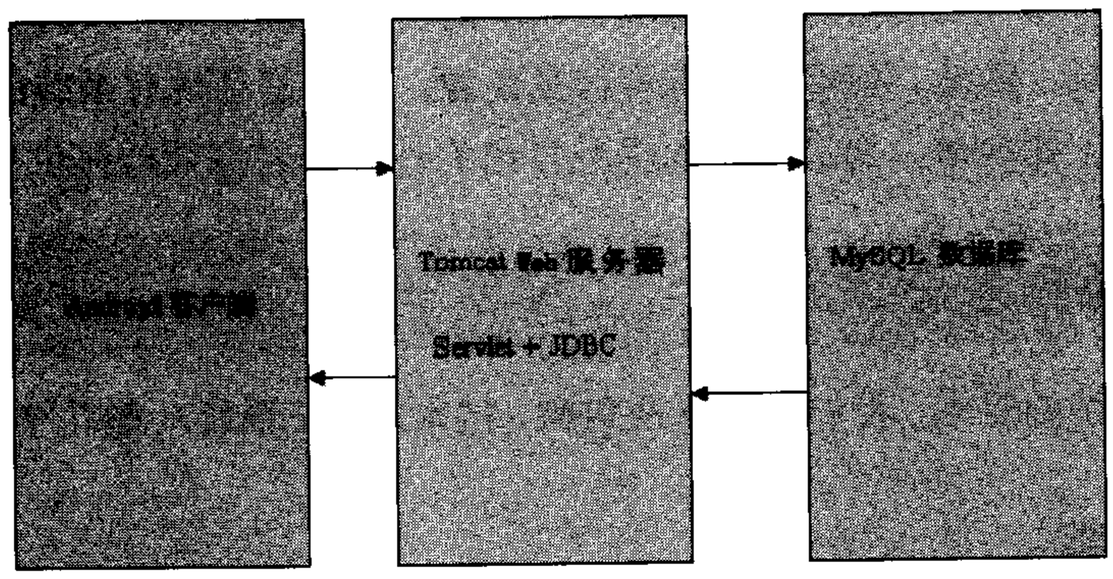


图 15.2　技术选型

15.2.3　系统功能

通过对系统进行需求分析，我们了解了系统的需求。通过系统物理架构和技术选择，我们确定了项目的可行性。接下来我们将对系统功能进行介绍。

1. 系统登录

为了增强系统的使用安全，所有使用系统之前必须登录系统，登录过程是通过无线网络，在后台的数据库通过用户名称密码进行查询。有该注册用户方可使用该系统。系统登录界面如图 15.3 所示。

2. 信息查询

信息查询可以通过无线网络对远端公安信息库进行各种查询，例如在逃人员查询、被盗车辆查询、人口信息查询、社区信息查询、出入境资料查询、案件查询和公文查询等。

信息查询菜单如图 15.4 所示。

3. 信息采集

信息采集功能是对公安人员现场采集的信息进行及时高效的保存。可以采集的内容包括：区警务信息采集、治安信息上报、机动车违章信息文字采集、交通违法信息采集、现场取证笔录、现场违法事件处理等。

信息采集菜单如图 15.5 所示。

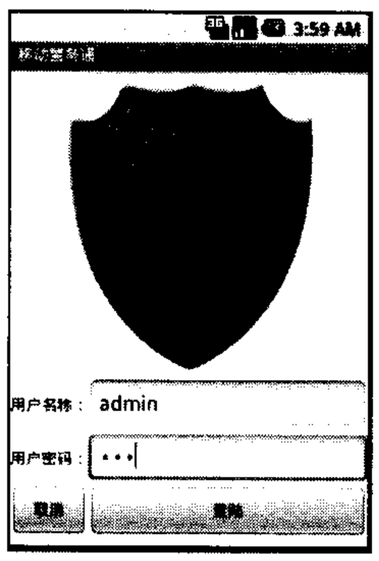


图 15.3　系统登录界面

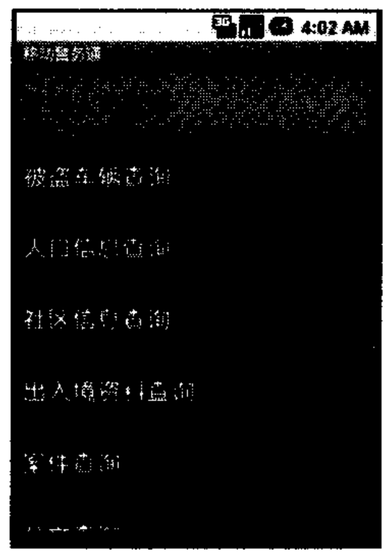


图 15.4　信息查询菜单

4．文件上传

文件上传功能主要是对现场照片（如：交通事故）、现场录音及文件进行及时上传。文件上传包括现场照片、现场录音和文件。

文件上传菜单如图 15.6 所示。

5．位置服务（GPS 跟踪定位功能）

位置服务是通过 GPS 硬件和 Google 的位置服务来实现位置查询、跟踪和定位等功能。位置服务包括：我的位置、按坐标查询、按地址查询和跟踪轨迹等。

位置服务菜单如图 15.7 所示。

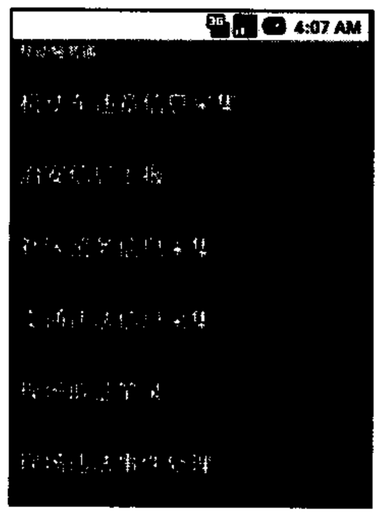


图 15.5　信息采集菜单

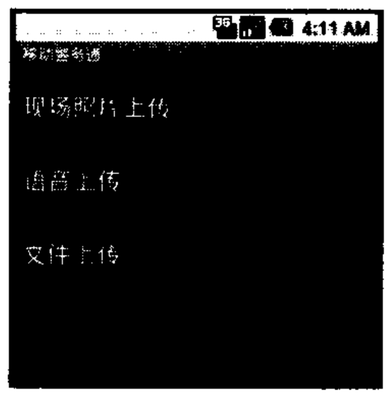


图 15.6　文件上传菜单

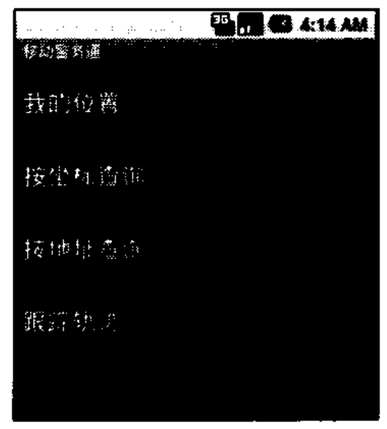


图 15.7　位置服务菜单

15.3 移动警务通详细设计

确定了系统功能之后，我们就可以进行图形的详细设计了。详细设计里主要包括：系统包及其资源规划、Activity 界面规划及其程序执行流程、系统数据库设计等。

15.3.1　系统包及其资源规划

根据系统功能设计，本系统分为五个模块：登录系统、信息查询、信息采集、文件上传

和 GPS 定位。那么，系统的包规划也是按照五个模块来划分的。包结构说明如表 15.1 所示。

表 15.1　系统包说明

包　名　称	包　描　述
com.amaker.mp	系统根包，放置登录和主菜单 Activity 类
com.amaker.mp.fileupload	文件上传功能包
com.amaker.mp.infoquery	信息查询功能包
com.amaker.mp.infocollect	信息采集功能包
com.amaker.mp.location	GPS 定位功能包
com.amaker.util	工具类包

包及其资源结构如图 15.8 所示。

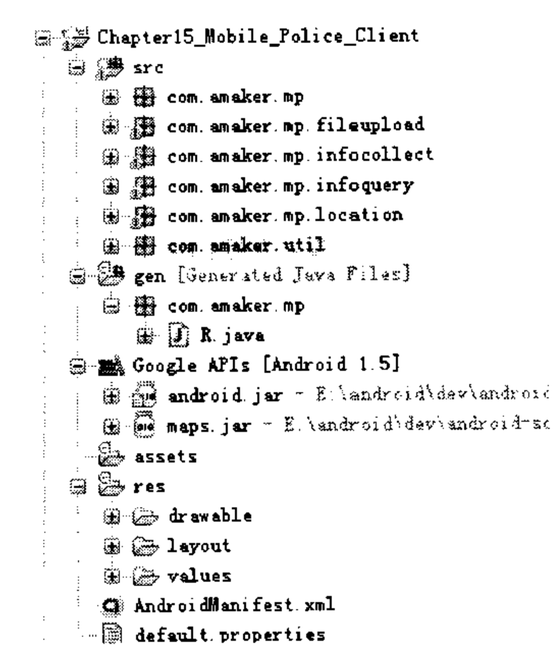


图 15.8　系统包及其资源结构

15.3.2　Activity 界面规划及其程序执行流程

万事预则立，不预则废。修改图纸要比修改大楼更方便且成本更低，所以在进行项目之前我们需要把系统展示的界面做一个规划。本系统用到了大量的 Activity，并且每个模块的展现结构都是大同小异，所以在这里只针对每个模块的不同功能做一个介绍。

1．系统登录界面

系统登录界面是系统的入口，该界面包括一个图片视图 ImageView、两个 TextView、两个 EditText 和两个 Button，其界面如图 15.9 所示。

2．系统主菜单

系统用户登录成功后将进入系统主菜单。系统主菜单由 ListView 展示，其中包括信息查询、信息采集、文件上传和 GPS 定位，其界面如图 15.10 所示。

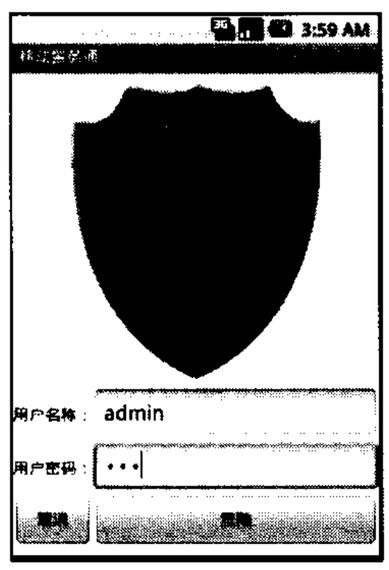


图 15.9　系统登录界面

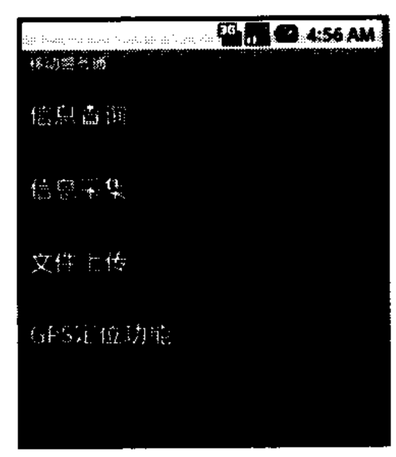


图 15.10　系统主菜单界面

3. 系统子菜单

进入系统主菜单后，选择菜单项将进入该菜单的子菜单。例如，信息查询的子菜单如图 15.11 所示。

4. 查询功能界面

单击查询功能项进入查询功能界面。查询界面与登录界面很相似，一般要输入查询条件，例如身份证号码、车牌照等。输入查询条件后选择查询按钮进行查询，下面将显示查询结果。例如：在逃人员查询界面如图 15.12 所示。

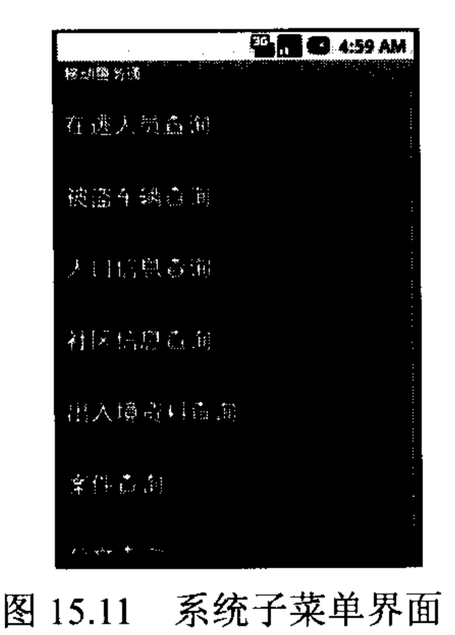


图 15.11　系统子菜单界面

图 15.12　在逃人口查询界面

5. 信息采集功能界面

选择信息采集子菜单将进入信息采集功能界面。信息采集主要是让用户输入采集的信息。输入好信息后，选择单击提交或取消按钮，提交或取消提交。提交后使用对话框显示提交成功或失败信息。如机动车违章信息采集界面如图 15.13 所示。

6. 文件上传功能界面

选择文件上传子菜单将进入文件上传功能界面。文件上传功能界面要求用户选择上传文件路径。选择单击提交或取消按钮，进行文件上传或取消上传。例如，现场照片上传界面如图 15.14 所示。

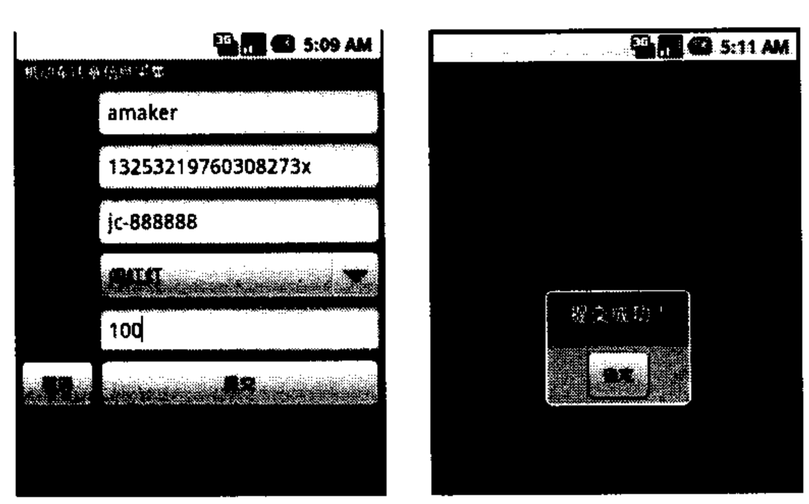


图 15.13　机动车违章信息采集界面

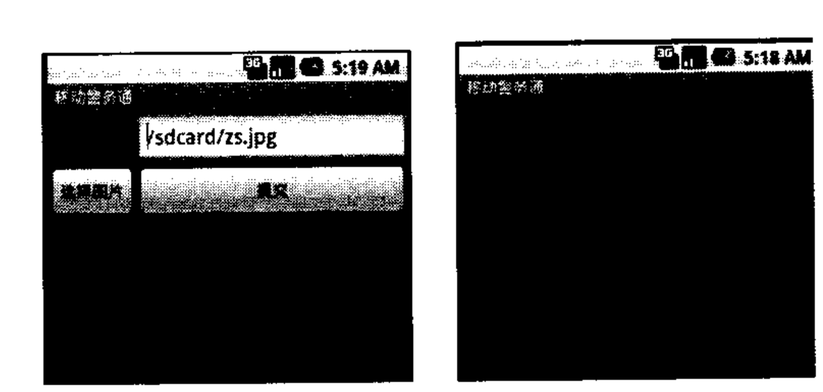


图 15.14　现场照片上传界面

7. GPS 定位界面

选择 GPS 定位功能子菜单将进入 GPS 定位功能界面。GPS 定位功能主要是使用 MapView 来显示地图位置信息。例如，我的位置功能界面如图 15.15 所示。

15.3.3 系统数据库设计

程序无非就是输入、处理和输出。本系统也不例外，除了 Android 客户端应用程序和中间无线网络，还要有强大的数据库做支持。例如，在“在逃人口查询”模块中，没有在逃人口库的支持，程序是无法实现的。本系统采用 MySQL 数据库来模拟后台数据。实际的数据我没有这个权限得到！呵呵！这是公安系统保密的。本节将对系统使用到的表结构进行详细讲解。

1. 系统用户表（UserTbl），登录系统使用。表结构如图 15.16 所示。

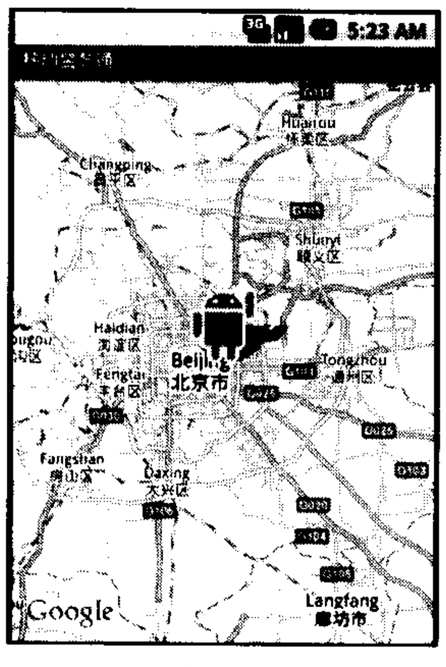


图 15.15 我的位置功能界面

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
username	VARCHAR	20	0	☐	Null	用户名称
password	VARCHAR	20	0	☐	Null	用户密码

图 15.16 系统用户表结构

2. 在逃人口表（EscapedPeopleTbl），查询在逃人口使用。表结构如图 15.17 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
name	VARCHAR	20	0	☐	Null	姓名
gender	VARCHAR	20	0	☐	Null	性别
age	INTEGER	11	0	☐	Null	年龄
IDNo	VARCHAR	20	0	☐	Null	身份证号码
address	VARCHAR	100	0	☐	Null	籍贯
crimeRecord	VARCHAR	200	0	☐	Null	犯罪记录
pic	VARCHAR	100	0	☐	Null	照片路径

图 15.17 在逃人口表结构

3. 机动车违章信息采集表（VehicleInfoTbl），添加机动车违章信息使用。表结构如图 15.18 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
name	VARCHAR	20	0	☐	Null	姓名
idno	VARCHAR	20	0	☐	Null	身份证号码
license	VARCHAR	20	0	☐	Null	车牌照号码
createTime	VARCHAR	30	0	☐	Null	该记录添加时间
faultRecord	VARCHAR	300	0	☐	Null	违章记录
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Null	[illegible]

图 15.18 机动车违章信息采集表结构

4．上传文件表（UploadFileTbl），上传文件使用。表结构如图 15.19 所示。

Field Name	Field Type	Size	Precisi...	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
uploadTime	VARCHAR	20	0	☐	Null	上传时间
FileDesc	VARCHAR	200	0	☐	Null	文件描述
FilePath	VARCHAR	100	0	☐	Null	文件保存路径

图 15.19　文件上传表结构

15.4 系统编码实现

有了以上的准备工作，接下来我们要开始进行系统编码实现了。本系统程序分为两个部分，前端 Android 实现和后端服务器实现。我们主要讲解前端 Android 实现，至于后端服务器实现只做简单讲解。对 Java 服务器端编程不是很熟悉的读者，请参阅 Java EE 相关书籍。

15.4.1　登录模块 Android 客户端实现

为了提高系统的安全性，要想使用该系统必须进行系统登录。登录的基本流程是：通过用户输入的用户名称和用户密码，到后台数据库做查询，如果有该用户存在则进入系统，否则提示用户输入错误。

为了完成系统登录功能，需要 Android 客户端程序和后台服务器程序。本节讲述的是有关 Android 客户端的设计实现。

要完成登录模块的 Android 客户端实现，需要的步骤说明如下。

① 创建一个 Android 工程，名称为“Chapter15_Mobile_Police_Client”，该工程为了支持定位服务，在使用 Eclipse 创建工程时需要选择“Google APIs”。

② 在工程的 res/layout 目录下，创建一个名称为“login_system.xml”的布局文件，布局文件外层是 LinearLayout，内层嵌套 TableLayout 完成布局管理。该布局文件中有一个 ImageView 组件用于显示图标、两个 TextView 用于显示用户名称和密码、两个 EditText 用于输入用户名称和密码、两个 Button 用于登录系统和取消登录。具体代码如下所示。

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

   <ImageView
      android:id="@+id/ImageView01"
      android:layout_height="280px"
      android:layout_width="wrap_content"
      android:src="@drawable/logo2"
      android:layout_gravity="center"></ImageView>

   <TableLayout
```

```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="1">
            <TableRow>
                <TextView
                    android:text="用户名称："
                    android:id="@+id/TextView"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:textColor="@color/black"
                    ></TextView>

                <EditText
                    android:text=""
                    android:id="@+id/userEditText"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"></EditText>
            </TableRow>
            <TableRow>
                <TextView
                    android:text="用户密码："
                    android:id="@+id/TextView"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:textColor="@color/black"
                    ></TextView>

                <EditText
                    android:text=""
                    android:id="@+id/pwdEditText"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:password="true"></EditText>
            </TableRow>

            <TableRow android:gravity="right">
                <Button
                    android:text="取消"
                    android:id="@+id/cancelButton"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"></Button>

                <Button
                    android:text="登录"
                    android:id="@+id/loginButton"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"></Button>
            </TableRow>
        </TableLayout>
    </LinearLayout>
```

③ 创建一个名称为“LoginActivity”的 Activity，在类的顶部声明用到的 EditText 和

Button 组件，在 onCreate()方法中调用 setContentView(R.layout.login_system)方法设置布局视图。通过 findViewById()方法实例化 EditText 和 Button 对象。程序代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
package com.amaker.mp;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
public class LoginActivity extends Activity {
    // 声明使用到的 Button 实例
    private Button cancelBtn,loginBtn;
    // 声明使用到的 EditText 实例
    private EditText userEditText,pwdEditText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.login_system);
        // 通过 findViewById 方法获得 Button 实例
        cancelBtn = (Button)findViewById(R.id.cancelButton);
        loginBtn = (Button)findViewById(R.id.loginButton);
        // 通过 findViewById 方法获得 EditText 实例
        userEditText = (EditText)findViewById(R.id.userEditText);
        pwdEditText = (EditText)findViewById(R.id.pwdEditText);
}
}
```

④ 为了实现登录功能，Android 需要通过网络访问后台的数据库进行查询。Android 访问网络的实现方法有很多，例如 Web Service、HTTP、直接使用 WebView 浏览器。Android 集成了 Apache 的 Http Client 模块，这样使得使用 HTTP 访问网络成为可能。在这里我们使用 HTTP 访问网络。HTTP 是请求（Request）/响应（Response）的一种机制，这里我们定义了一个工具类 HttpUtil 来获得 HttpRequest 对象和 HttpResponse 对象，以及发送 get 和 post 请求获得返回 Response 信息的方法。

```
package com.amaker.util;

import java.io.IOException;

import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
```

```
public class HttpUtil {
    // 声明 Base URL 常量
    public static final String BASE_URL="http://192.168.1.101:8080/Chapter15_Mob-
ile_Police_Server/";

    // 通过 url 获得 HttpGet 对象
    public static HttpGet getHttpGet(String url){
        // 实例化 HttpGet
        HttpGet request = new HttpGet(url);
        return request;
    }
    // 通过 URL 获得 HttpPost 对象
    public static HttpPost getHttpPost(String url){
         // 实例化 HttpPost
         HttpPost request = new HttpPost(url);
         return request;
    }
    // 通过 HttpGet 获得 HttpResponse 对象
    public static HttpResponse getHttpResponse(HttpGet request) throws Client
ProtocolException, IOException{
        // 实例化 HttpResponse
        HttpResponse response = new DefaultHttpClient().execute(request);
        return response;
    }
    // 通过 HttpPost 获得 HttpResponse 对象
    public static HttpResponse getHttpResponse(HttpPost request) throws Client
ProtocolException, IOException{
        // 实例化 HttpResponse
        HttpResponse response = new DefaultHttpClient().execute(request);
        return response;
    }


    // 通过 url 发送 post 请求，返回请求结果
    public static String queryStringForPost(String url){
        // 获得 HttpPost 实例
        HttpPost request = HttpUtil.getHttpPost(url);
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得返回结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        } catch (IOException e) {
```

```
            e.printStackTrace();
            result = "网络异常！";
            return result;
        }
        return null;
    }

    // 通过 HttpPost 发送 get 请求，返回请求结果
    public static  String queryStringForGet(String url){
        // 获得 HttpGet 实例
        HttpGet request = HttpUtil.getHttpGet(url);
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得返回结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        } catch (IOException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        }
        return null;
    }

    // 通过 HttpPost 发送 post 请求，返回请求结果
    public static String queryStringForPost(HttpPost request){
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得请求结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        } catch (IOException e) {
            e.printStackTrace();
            result = "网络异常！";
```

```
            return result;
        }
        return null;
    }
}
```

⑤ 定义一个显示提示信息的对话框。

```
// 定义一个显示提示信息的对话框
private void showDialog(String msg){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage(msg)
          .setCancelable(false)
          .setPositiveButton("确定", new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int id) {
             }
          });
    AlertDialog alert = builder.create();
    alert.show();
}
```

⑥ 定义 validate()方法，对用户名称和密码进行非空验证。

```
// 对用户名称和密码非空进行验证
private boolean validate(){
    String username = userEditText.getText().toString();
    if(username.equals("")){
       showDialog("用户名称是必填项！");
       return false;
    }
    String pwd = pwdEditText.getText().toString();
    if(pwd.equals("")){
       showDialog("用户密码是必填项！");
       return false;
    }
    return true;
}
```

⑦ 定义一个 query()方法，通过用户名称和密码进行查询，发送 Post 请求，获得响应结果。

```
// 通过用户名称和密码进行查询，发送 Post 请求，获得响应结果。
private String query(String username,String password){
    // 查询字符串
    String queryString = "username="+username+"&password="+password;
    // 查询 URL
    String url = HttpUtil.BASE_URL+"servlet/LoginServlet?"+queryString;
    // 查询并返回结果
    return HttpUtil.queryStringForPost(url);
}
```

⑧ 定义 login()方法，该方法调用 query()方法，并对返回结果进行判断。如果结果不为空并且结果等于 1，证明登录成功，返回 true，否则证明登录失败返回 false。

```
// 登录方法
private boolean login(){
    // 获得用户名称
    String username = userEditText.getText().toString();
    // 获得密码
    String pwd = pwdEditText.getText().toString();
    // 查询返回结果
    String result=query(username,pwd);
    // 对结果进行判断
    if(result!=null&&result.equals("1")){
        return true;
    }else{
        return false;
    }
}
```

⑨ 为登录按钮添加单击监听器，在 onClick()方法中，首先调用 validate()方法进行验证，如果验证通过再使用 login()方法，如果登录成功，则系统进入主菜单界面，否则提示登录失败信息。如果验证失败提示验证失败，重新输入。

```
        // 为按钮添加单击事件监听器
loginBtn.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 判断验证
            if(validate()){
                // 判断是否登录成功
                if(login()){
                    // 启动 Activity
                    Intent intent = new Intent(LoginActivity.this,MainActivity.class);
                    startActivity(intent);
                }else{
                    // 显示对话框
                    showDialog("用户名称或者密码错误，请重新输入！");
                }
            }
        }
    });
```

⑩ 为取消按钮添加单击监听器，在 onClick()方法中调用 finish()方法结束该 Activity。

```
// 设置取消监听器
cancelBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 结束当前 Activity
        finish();
    }
});
```

⑪ 程序运行结果如图 15.20 所示。

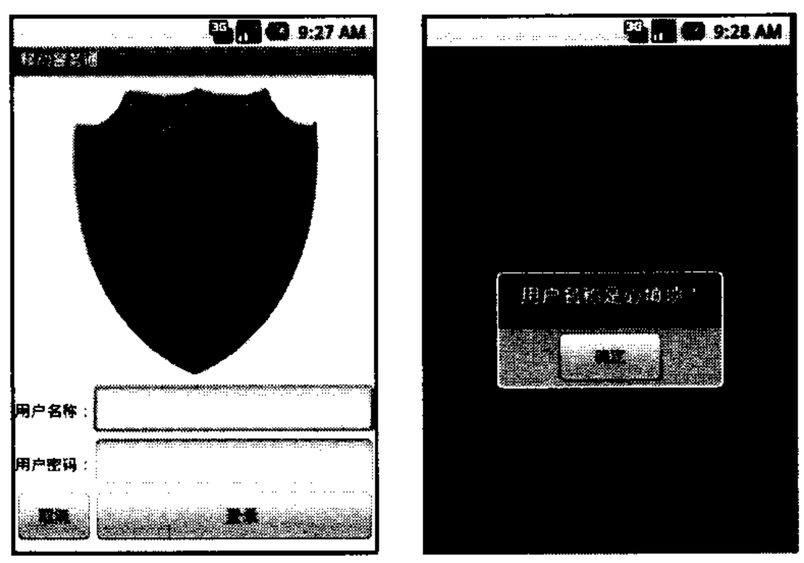


图 15.20　用户登录

15.4.2　登录模块服务器端实现

后台的服务器实现采用 JDBC 访问后台数据库，Servlet 响应 HttpRequest 请求，并返回响应结果。为了完成该功能我们需要安装 MySQL 数据并配置 Tomcat 服务器，这部分内容将不再赘述。有不太明白的读者请参阅 Java EE 相关书籍。

为了程序更加清晰易懂，我们依然采用 MVC+DAO 的设计模式及分层开发思想。

要完成登录模块的服务器端实现需要的步骤说明如下。

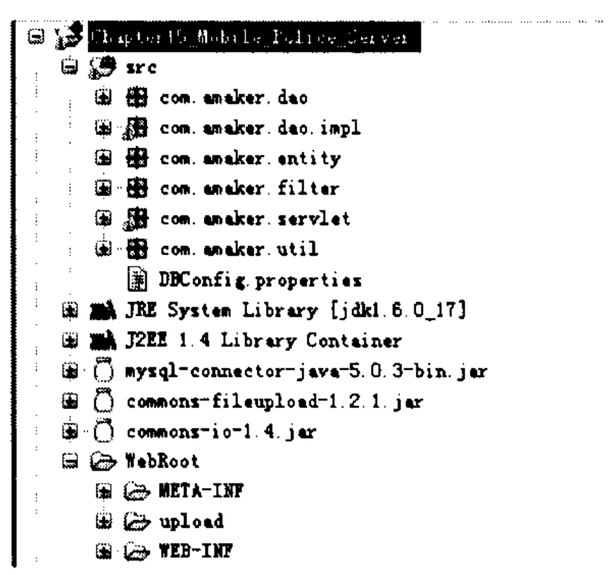


图 15.21　后台系统包结构

① 创建一个名称为“Chapter15_Mobile_Police_Server”的 Web 工程。

② 创建程序需要的包结构。dao 包用于放置采用 DAO 设计模式实现的访问数据库接口及其实现类；entity 包用于放置对应数据库表的实体类；servlet 包用于放置 Servlet 类；filter 包用于放置过滤器；util 包是工具类。其内容如图 15.21 所示。

③ 因为程序要访问 MySQL 数据库，所以要添加 MySQL 驱动包“mysql-connector-java-5.0.3-bin.jar”。将该 jar 文件添加到工程的 classpath 下。本工程可以直接将其复制到 lib 目录下即可。

④ 创建访问数据库的工具类 DBUtil，该类的主要功能是打开和关闭数据库连接。为了程序更具可扩展性，我们将配置信息保存到配置文件当中，这样可以在不修改程序的前提下移植数据库。

配置文件 DBConfig.properties 的内容如下所示。

```
// 数据库驱动
driver=com.mysql.jdbc.Driver
// 数据库连接 url
url=jdbc:mysql://localhost:3306/police_db?useUnicode=true&characterEncoding=utf-8
```

```
// 用户名称
username=root
// 密码
password=1
```

DBUtil 工具类的内容如下所示。

```
package com.amaker.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
/**
 *
 * @author hz.guo
 * 数据库工具类
 */
public class DBUtil {
    // 关闭数据库连接
   public void closeConn(Connection conn){
       try {
          conn.close();
       } catch (SQLException e) {
          e.printStackTrace();
       }
   }

   // 打开数据库连接
   public Connection openConnection() {
       // 实例化 Properties
       Properties prop = new Properties();
       // 驱动
       String driver = null;
       // 数据库连接 url
       String url = null;
       // 用户名称
       String username = null;
       // 用户密码
       String password = null;

       try {
          // 加载配置文件
          prop.load(this.getClass().getClassLoader().getResourceAsStream(
                  "DBConfig.properties"));
          // 获得数据库驱动
          driver = prop.getProperty("driver");
          // 获得数据库 url
          url = prop.getProperty("url");
          // 获得用户名称
          username = prop.getProperty("username");
          // 获得用户密码
          password = prop.getProperty("password");
```

```
            // 注册驱动
            Class.forName(driver);
            // 返回数据库连接
            return DriverManager.getConnection(url, username, password);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

⑤ 创建用于封装 UserTbl 表信息的实体类 User 类。

```
package com.amaker.entity;

/**
 *
 * @author hz.guo
 * 用于封装 UserTbl 表的实体类
 */
public class User {
    // 编号
    private int id;
    // 用户名称
    private String username;
    // 密码
    private String password;

    getters()…
    setters()…
```

⑥ 创建访问数据库的 DAO 接口类 UserDao。

```
package com.amaker.dao;
import com.amaker.entity.User;
public interface UserDao {
    // 登录方法，返回 User 实例
    public User login(String username,String password);
}
```

⑦ 创建 UserDao 接口实现类 UserDaoImpl。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.amaker.entity.User;

import com.amaker.dao.UserDao;
import com.amaker.util.DBUtil;
```

```
/**
 *
 * @author 郭宏志
 * 用户登录 DAO 实现类
 */
public class UserDaoImpl implements UserDao {

    // 通过用户名称和密码登录，登录成功返回 User 对象，登录失败返回 null
   public User login(String username, String password) {
       // SQL 查询语句
       String sql = " select id,username,password from UserTbl where username=? and
password=? ";
       // 实例化数据库工具类
       DBUtil util = new DBUtil();
       // 获得数据库连接
       Connection conn = util.openConnection();
       try {
          // 创建预定义语句
          PreparedStatement pstmt = conn.prepareStatement(sql);
          // 设置查询参数
          pstmt.setString(1, username);
          // 设置查询参数
          pstmt.setString(2, password);
          // 结果集
          ResultSet rs = pstmt.executeQuery();
          // 判断该用户是否存在
          if (rs.next()) {
              // 获得编号
              int id = rs.getInt(1);
              // 实例化 User
              User u = new User();
              // 设置 User 属性
              u.setId(id);
              u.setPassword(password);
              u.setUsername(username);
              return u;
          }
       } catch (SQLException e) {
          e.printStackTrace();
       } finally {
          util.closeConn(conn);
       }
       return null;
   }
}
```

⑧ 创建 LoginServlet 类响应请求。该类通过 request.getParameter()方法获得请求参数，调用 UserDaoImpl 的 login()方法，并根据登录结果返回响应结果信息。

```
package com.amaker.servlet;

import java.io.IOException;
```

```
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.UserDao;
import com.amaker.dao.impl.UserDaoImpl;
import com.amaker.entity.User;
/**
 *
 * @author 郭宏志
 * 响应 Android 客户端发来的请求
 */
public class LoginServlet extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // 设置响应类型
      response.setContentType("text/html");
      // 获得打印输出流
      PrintWriter out = response.getWriter();
      // 实例化 Dao
      UserDao dao = new UserDaoImpl();
      // 获得客户端请求参数
      String username = request.getParameter("username");
      String password = request.getParameter("password");
      // 登录返回 User 实例
      User u = dao.login(username, password);
      // 判断是否登录成功
      if(u!=null){
         // 响应客户端内容，登录成功
         out.print("1");
      }else{
         // 响应客户端内容，登录失败
         out.print("0");
      }
      // 刷新流
      out.flush();
      // 关闭流
      out.close();
   }

   public void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // 调用 doGet 方法
      doGet(request,response);
   }
   // 初始化方法
   public void init() throws ServletException {
   }
```

```
    // 构造方法
    public LoginServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }

}
```

到此为止，系统登录模块的前后台实现已全部完成。

15.4.3　程序主菜单实现

用户成功登录之后便可以进入系统主菜单界面了。主菜单是通过一个 ListView 来展示的。在这里是创建一个 Activity 使其继承 ListActivity。

程序主菜单的实现需要的步骤说明如下。

① 创建一个名称为“MainActivity”的 Activity，并继承 ListActivity。

② 在 onCreate()方法中声明菜单项，并为 ListView 设置菜单项。

③ 覆盖 onListItemClick()方法，通过对用户选择菜单项的判断，跳转到不同的子菜单。

```
package com.amaker.mp;

import com.amaker.mp.fileupload.FileUploadActivity;
import com.amaker.mp.infocollect.InfoCollectActivity;
import com.amaker.mp.infoquery.InfoQueryActivity;
import com.amaker.mp.location.GpsLocationActivity;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

/**
 *
 * @author 郭宏志
 * 程序主菜单实现
 */
public class MainActivity extends ListActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 菜单项
        String[] menus = { "信息查询", "信息采集", "文件上传","GPS 定位功能" };
        // 为 List 设置菜单项
```

```
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, menus));
        getListView().setTextFilterEnabled(true);
    }


    // 响应菜单单击事件，跳转到不同的子菜单
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Intent intent=null;
        switch (position) {
        case 0:
            // 跳转到信息查询
            intent = new Intent(MainActivity.this,InfoQueryActivity.class);
            startActivity(intent);
            break;
        case 1:
            // 跳转到信息采集
            intent = new Intent(MainActivity.this,InfoCollectActivity.class);
            startActivity(intent);
            break;
        case 2:
            // 跳转到文件上传
            intent = new Intent(MainActivity.this,FileUploadActivity.class);
            startActivity(intent);
            break;
        case 3:
            // 跳转到 GPS 定位服务
            intent = new Intent(MainActivity.this,GpsLocat ionActivity.class);
            startActivity(intent);
            break;
        }
    }
}
```

程序运行结果如图 15.22 所示。

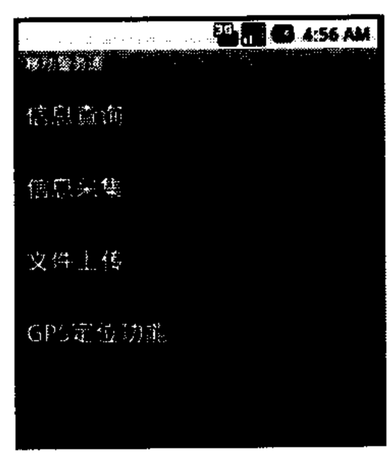


图 15.22　系统主菜单界面

15.4.4　信息查询子菜单实现

进入程序主菜单之后，单击信息查询子菜单便进入信息查询子菜单界面。信息查询子菜单的实现和主菜单的实现类似。具体步骤说明如下。

① 创建一个名称为“InfoQueryActivity”的 Activity，该 Activity 继承 ListActivity。

② 覆盖 onCreate()方法，在其中声明菜单项，并将其设置为 ListView 的选项。

③ 覆盖 onListItemClick()方法，响应菜单的单击事件，跳转到相应的查询界面。

```
package com.amaker.mp.infoquery;

import android.app.ListActivity;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

/**
 *
 * @author 郭宏志
 * 信息查询子菜单实现
 */
public class InfoQueryActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 声明子菜单项
        String[] menus = { "在逃人员查询", "被盗车辆查询", "人口信息查询", "社区信息查询" , "
出入境资料查询", "案件查询" ,"公文查询"};
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, menus));
        getListView().setTextFilterEnabled(true);
    }

    // 响应子菜单单击事件
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Intent intent = null;
        switch (position) {
        case 0:
            // 跳转到在逃人口查询界面
            intent = new Intent(InfoQueryActivity.this,EscapedQueryActivity.class);
            startActivity(intent);
            break;
        case 1:
            break;
        case 2:
            break;
        case 3:
            break;
        }
    }
}
```

程序运行结果如图 15.23 所示。

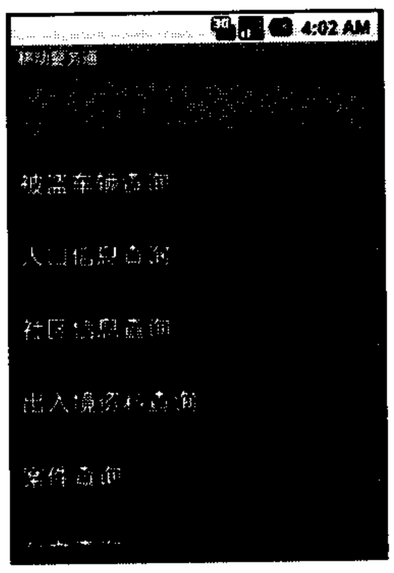


图 15.23　信息查询子菜单界面

15.4.5　在逃人员查询模块客户端实现

在逃人员查询是本程序的一个重要功能，通过该功能为公安人员的灵活、随机办公提供了方便。该功能是通过输入被查人员的身份证号码，通过无线网络到后台人口库中查询是否有犯罪记录。

该功能和用户登录功能相似，具体实现步骤说明如下。

① 在工程的 res\layout\目录下创建一个名称为“escape_people_query.xml”的布局文件，该布局文件使用 LinearLayout 并嵌套 TableLayout 进行布局管理。

1.1 添加一个 TextView 和一个 EditText 用于提示用户输入身份证号码。

1.2 添加两个 Button，一个用于响应查询基本信息事件，一个用于响应查询照片事件。

1.3 添加两个 TextView 用于显示查询基本信息结果。

1.4 添加一个 ImageView 用于显示照片。

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

   <TextView
   android:text="请输入身份证号："
   android:id="@+id/TextView01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"></TextView>

   <EditText
      android:text=""
      android:id="@+id/EditText01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"></EditText>

   <TableLayout
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <TableRow android:gravity="right">
         <Button
            android:text="查询信息"
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right"></Button>

         <Button
            android:text="查询照片"
            android:id="@+id/Button02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

      </TableRow>
      <TextView
         android:text="查询结果"
         android:id="@+id/TextView02"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"></TextView>
```

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

        <ImageView
            android:id="@+id/ImageView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></ImageView>
    </TableLayout>
</LinearLayout>
```

② 在“com.amaker.mp.infoquery”包中创建一个名称为“EscapedQueryActivity”的 Activity，并继承 Activity。在该类顶部声明使用到的视图组件。覆盖 onCreate()方法，设置当前布局,并在其中实例化使用到的视图组件。

```
package com.amaker.mp.infoquery;

import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;

import com.amaker.mp.R;
import com.amaker.util.HttpUtil;

/**
 *
 * @author 郭宏志
 * 在逃人口查询实现
 */
public class EscapedQueryActivity extends Activity {
    // 声明程序使用的 EditText 实例，输入身份证号码
    private EditText myEditText;
    // 声明使用到的 Button 实例，查询用户信息和照片信息
    private Button myBtn,myBtn2;
    // 声明 TextView 实例，显示查询内容
    private TextView myTextView;
```

```
    // 声明 ImageView 显示照片
    private ImageView myImageView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.escape_people_query);
        // 通过 findViewById 方法获得组件实例
        myEditText = (EditText)findViewById(R.id.EditText01);
        myTextView = (TextView)findViewById(R.id.resultTextView);
        myImageView = (ImageView)findViewById(R.id.ImageView01);
        myBtn = (Button)findViewById(R.id.Button01);
        myBtn2 = (Button)findViewById(R.id.Button02);
    }
}
```

③ 创建显示提示对话框方法 showAlert()。

```
    // 显示对话框
    private void showAlert(){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage("请输入身份证号！")
              .setCancelable(false)
              .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                  }
              });
        // 创建对话框
        AlertDialog alert = builder.create();
        // 显示对话框
        alert.show();
    }
```

④ 创建对身份证号进行非空验证方法 validate()。

```
    // 非空验证
    private boolean  validate(){
        // 获得身份证号
        String no = myEditText.getText().toString();
        // 判断身份证是否为空
        if(no==null||no.equals("")){
            // 显示提示信息
            showAlert();
            return false;
        }
        return true;
    }
```

⑤ 创建查询方法 query()，发送 POST 请求，并返回响应结果。该方法使用到了 HttpUtil 工具类，详细请参阅 15.4.1 节。

```
    // 通过身份证号查询基本信息
    private String query(String idno){
```

```
        // 查询字符串
        String queryString = "idno="+idno;
        // 查询 url
        String url = HttpUtil.BASE_URL+"servlet/PeopleServlet?"+queryString;
        // 返回查询结果
        return HttpUtil.queryStringForPost(url);
    }
```

⑥ 创建 queryPicPath()，通过身份证号码查询照片路径。

```
    // 查询照片路径
    private String queryPicPath(String idno){
        // 查询字符串
        String queryString = "idno="+idno;
        // 查询 url
        String url = HttpUtil.BASE_URL+"servlet/PeopleImgServlet?"+queryString;
        // 返回查询结果
        return HttpUtil.queryStringForPost(url);
    }
```

⑦ 创建 showPic()方法，通过照片路径获得照片字节流，并通过 BitmapFactory 的 decodeStream(in)方法，将其转化为 Bitmap 对象，通过 ImageView 来显示。具体是将 url 路径字符串转换为 URL 对象，调用 URL 的 openConnection()方法，该方法返回 URLConnection 对象。调用 URLConnection 的 connect()方法建立连接。调用 getInputStream()方法打开输入流 InputStream。

```
    // 通过 url，查询照片方法
    private void showPic(String urlStr){
        try {
            // 实例化 URL
            URL url = new URL(urlStr);
            获得连接
            URLConnection conn = url.openConnection();
            // 进行连接
            conn.connect();
            // 获得输入流
            InputStream in = conn.getInputStream();
            // 获得 Bitmap 实例
            Bitmap map = BitmapFactory.decodeStream(in);
            // 显示图片
            myImageView.setImageBitmap(map);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

⑧ 设置查询基本信息的按钮的单击事件，进行查询。

```
    // 设置单击事件，查询基本信息
        myBtn.setOnClickListener(new OnClickListener() {
            @Override
```

```
        public void onClick(View v) {
            // 判断验证情况
            if(validate()){
                // 获得身份证号码
                String idno = myEditText.getText().toString();
                // 获得查询信息
                String msg = query(idno);
                // 判断信息是否为空
                if(msg!=null&&!msg.equals(""))
                    // 显示查询结果
                    myTextView.setText(msg);
                else
                    // 显示查无此人!
                    myTextView.setText("查无此人! ");
            }
        }
    });
```

⑨ 为显示照片按钮设置单击事件，查询照片。

```
// 设置单击事件，查询照片
myBtn2.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 判断验证
        if(validate()){
            // 获得身份证号码
            String idno = myEditText.getText().toString();
            // 查询照片路径
            String urlStr = queryPicPath(idno);
            // 判断是否为空

  if(urlStr!=null&&!urlStr.equals("")){
                    // 显示照片
                    showPic(urlStr);
            }
        }
    }
});
```

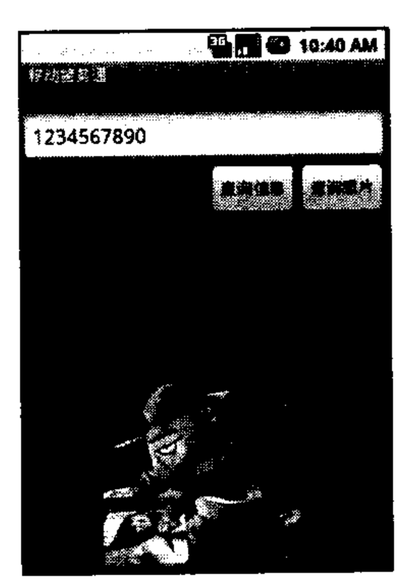


图 15.24　在逃人口查询结果

程序运行结果如图 15.24 所示。

查询模块的其他查询项目和在逃人口查询流程类似，这里不再赘述，读者可以根据该模块内容进行扩展学习。

15.4.6　在逃人员查询模块服务器端实现

在逃人口查询服务器端实现和用户登录的服务器端实现类似，客户端发送请求到服务器，服务器通过 Servlet 响应客户端的调用，并且 Servlet 通过 JDBC 访问数据库实现查询。

服务器端实现采用了 DAO+MVC 的设计模式，并采用分层的开发思想使得程序耦合

性更低，结构更清晰。

在逃人口查询模块的服务器端实现需要的步骤说明如下。

① 在 com.amaker.entity 包中创建实体类 People 对应数据库表 escapedpeopletbl。

```
package com.amaker.entity;

/**
 *
 * @author 郭宏志
 * 实体类对应数据库 escapedpeopletbl 表结构
 */
public class People {
    // 编号
    private int id;
    // 姓名
    private String name;
    // 年龄
    private int age;
    // 性别
    private String gender;
    // 地址
    private String address;
    // 身份证号码
    private String idno;
    // 犯罪记录
    private String crimerecord;
    //  照片
    private String pic;
    getters()…
    setters()…

    /**
     * 覆盖 toString()方法返回 People 信息字符串
     */
    public String toString() {
        StringBuilder sb = new StringBuilder("");

        sb.append("身份证号: ");
        sb.append(idno);
        sb.append("\n");

        sb.append("姓名: ");
        sb.append(name);
        sb.append("\n");

        sb.append("性别: ");
        sb.append(gender);
        sb.append("\n");

        sb.append("年龄: ");
        sb.append(age);
```

```
        sb.append("\n");

        sb.append("籍贯: ");
        sb.append(address);
        sb.append("\n");

        sb.append("犯罪记录: ");
        sb.append(crimerecord);
        sb.append("\n");

        return sb.toString();
    }
}
```

② 在 com.amaker.dao 包中创建查询接口 PeopleDao。

```
package com.amaker.dao;
import com.amaker.entity.People;
/**
 * @author 郭宏志
 * 在逃人口查询接口
 */
public interface PeopleDao {
   // 返回 People 信息字符串
   public String getString(String idno);
   // 返回 People 对象
   public People get(String idno);
}}
```

③ 在 com.amaker.dao.impl 包中创建 PeopleDao 接口对应的实现类 PeopleDaoImpl。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.amaker.dao.PeopleDao;
import com.amaker.entity.People;
import com.amaker.util.DBUtil;

/**
 *
 * @author 郭宏志
// 在逃人口查询实现
public class PeopleDaoImpl implements PeopleDao {
   //返回 People 信息字符串
   public String getString(String idno) {
       return get(idno).toString();
   }
   // 返回 People 对象
   public People get(String idno) {
```

```
        // 查询 SQL 语句
        String sql = " select id,name,age,gender,address,crimerecord,idno,pic from
escapedpeopletbl where idno=? ";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得数据库连接
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置查询参数
            pstmt.setString(1, idno);
            // 获得查询结果
            ResultSet rs = pstmt.executeQuery();
            // 判断是否有此人
            if (rs.next()) {
                // 获得人口信息
                int id = rs.getInt(1);
                // 姓名
                String name = rs.getString(2);
                // 年龄
                int age = rs.getInt(3);
                // 性别
                String gender = rs.getString(4);
                // 地址
                String address = rs.getString(5);
                // 犯罪记录
                String crimerecord = rs.getString(6);
                //String idno = rs.getString(6);
                // 照片
                String pic = rs.getString(8);
                // 实例化 People
                People p = new People();
                // 设置 People 属性
                p.setId(id);
                p.setName(name);
                p.setAge(age);
                p.setAddress(address);
                p.setCrimerecord(crimerecord);
                p.setGender(gender);
                p.setPic(pic);

                p.setIdno(idno);
                return p;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
        return null;
    }
}
```

④ 在 com.amaker.servlet 包中创建名称为“PeopleServlet”的 Servlet 来响应请求。即获得请求参数，调用后台 Dao 实现查询，通过 Response 返回查询结果。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.amaker.dao.PeopleDao;
import com.amaker.dao.impl.PeopleDaoImpl;

/**
 *
 * @author 郭宏志
 * 在逃人口查询 Servlet
 */
public class PeopleServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // 获得请求参数
      String idno = request.getParameter("idno");
      // 调用后台 Dao 实现查询
      PeopleDao dao = new PeopleDaoImpl();
      // 获得查询结果
      String msg = dao.getString(idno);
      // 设置内容类型
      response.setContentType("text/html");
      // 获得打印输出流
      PrintWriter out = response.getWriter();
      // 将查询信息写到客户端
      out.write(msg);
      // 刷新打印输出流
      out.flush();
      // 关闭输出流
      out.close();
   }

   public void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // 调用 doGet 方法
      doGet(request,response);
   }
   // 初始化方法
   public void init() throws ServletException {
   }
   // 构造方法
   public PeopleServlet() {
      super();
```

```
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
}
```

⑤ 在 com.amaker.servlet 包中创建一个名称为“PeoplePicServlet”的 Servlet 类，获得照片路径。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.amaker.dao.PeopleDao;
import com.amaker.dao.impl.PeopleDaoImpl;

/**
 *
 * @author 郭宏志
 * 在逃人口查询，获得照片路径
 */
public class PeopleImgServlet extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      // 获得请求参数，身份证号码
      String idno = request.getParameter("idno");
      // 调用后台 Dao 执行查询
      PeopleDao dao = new PeopleDaoImpl();
      // 获得照片路径
      String path = dao.get(idno).getPic();
      // 设置响应类型
      response.setContentType("text/html");
      // 获得打印输出流
      PrintWriter out = response.getWriter();
      // 响应请求，将查询结果写到客户端
      out.write(path);
      // 刷新
      out.flush();
      // 关闭输出流
      out.close();
   }

   public void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      // 调用 doGet 方法
```

```
        doGet(request,response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
    // 构造方法
    public PeopleImgServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }

}
```

至此，在逃人口查询的前后台程序已经全部编写完毕。

15.4.7 信息采集子菜单实现

信息采集子菜单和信息查询子菜单实现类似，创建菜单项，将菜单项设置为 ListView 的列表项来进行展示。信息采集子菜单实现步骤说明如下。

① 在 com.amaker.mp.infocollect 包中创建名称为“InfoCollectActivity”的 Activity，该 Activity 继承 ListActivity。

② 覆盖 onCreate()方法，并在其中声明菜单项数组，将数组设置为 ListView 的列表项来进行展示。

③ 覆盖 onListItemClick()方法，响应菜单项单击事件。

```
package com.amaker.mp.infocollect;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
/**
 *
 * @author 郭宏志
 * 信息采集子菜单
 */
public class InfoCollectActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 菜单项数组
        String[] menus = { "机动车违章信息采集", "治安信息上报", "社区警务信息采集", "交通违
法信息采集" , "现场取证笔录", "现场违法事件处理"};
```

```
        // 将菜单项数组设置为 ListView 的列表项
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, menus));
        getListView().setTextFilterEnabled(true);
    }

    // 响应菜单项的单击事件
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Intent intent = null;
        switch (position) {
        case 0:
            // 跳转到机动车违章信息采集
            intent = new Intent(InfoCollectActivity.this,VehicleInfoCollActivity.class);
            startActivity(intent);
            break;
        case 1:
            break;
        case 2:
            break;
        case 3:
            break;
        }
    }
}
```

程序运行结果如图 15.25 所示。

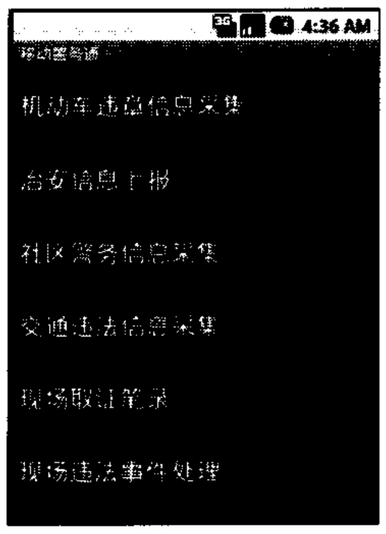


图 15.25　信息菜单子菜单

15.4.8　机动车违章信息采集 Android 客户端实现

机动车违章信息采集 Android 客户端实现，类似我们做 Web 开发中的提交表单。Android 集成了 Apache 的 HTTP 客户端，其实现机制和提交 Web 表单是相同的，只不过写法有一些不同。典型的 Form 表单如下代码所示。

```
<form name="f1"  action="" method="post">
    用户名称: <input type="text" name="login" id="login"><br/>
    用户密码: <input type="password" name="password" id="password"><br/>
    <input type="submit" value="保存">
</form>
```

下面我们就来看一下 Android 客户端该如何实现。实现机动车违章信息采集 Android 客户端，需要的步骤说明如下。

① 在工程的 res\layout\目录下，创建名称为“vehicle_info_collect.xml”的布局文件。该布局文件采用 LinearLayout 嵌套 TableLayout 实现布局管理。添加 TextView、EditText、Spinner 提示用户输入信息，添加 Button 响应用户提交或取消提交。其中 Spinner 中的下拉列表项来源于资源文件中的字符串数组。

```
<?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">

    <TableLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="1">
        <TableRow>
            <TextView
                android:text="姓名: "
                android:id="@+id/TextView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"></TextView>

            <EditText
                android:text=""
                android:id="@+id/nameEditText"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"></EditText>
        </TableRow>


        <TableRow>
            <TextView
                android:text="身份证号: "
                android:id="@+id/TextView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"></TextView>

            <EditText
                android:text=""
                android:id="@+id/idnoEditText"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"></EditText>
        </TableRow>


        <TableRow>
            <TextView
                android:text="驾驶证号: "
                android:id="@+id/TextView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"></TextView>

            <EditText
                android:text=""
                android:id="@+id/licenseEditText"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"></EditText>
        </TableRow>

        <TableRow>
```

```
        <TextView
            android:text="违章类型: "
            android:id="@+id/TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>

        <Spinner
            android:id="@+id/faultRecordSpinner"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:entries="@array/faultRecords"></Spinner>

    </TableRow>

    <TableRow>
        <TextView
            android:text="罚金: "
            android:id="@+id/TextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>

        <EditText
            android:text=""
            android:id="@+id/penaltyEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"></EditText>
    </TableRow>

    <TableRow android:gravity="right">
        <Button
            android:text="取消"
            android:id="@+id/cancelButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

        <Button
            android:text="提交"
            android:id="@+id/submitButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
    </TableRow>
  </TableLayout>

</LinearLayout>
```

② 在 com.amaker.mp.infocollect 包中创建名称为“VehicleInfoCollActivity”的 Activity，在该类的顶部声明使用到的视图组件。覆盖 onCreate()方法，实例化使用到的视图组件。

```
package com.amaker.mp.infocollect;

import java.io.UnsupportedEncodingException;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;

import com.amaker.mp.R;
import com.amaker.util.HttpUtil;

/**
 *
 * @author 郭宏志
 * 机动车违章信息采集
 */
public class VehicleInfoCollActivity  extends Activity {
    /*
     * 声明使用到的视图组件
     */
    private EditText myEditText1,myEditText2,myEditText3,myEditText4;
    private Spinner mySpinner ;
    private Button cancelBtn,submitBtn;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 界面布局
        setContentView(R.layout.vehicle_info_collect);
        // 设置标题
        setTitle("机动车违章信息采集");
        // 通过 findViewById 方法获得组件实例
        myEditText1 = (EditText)findViewById(R.id.nameEditText);
        myEditText2 = (EditText)findViewById(R.id.idnoEditText);
        myEditText3 = (EditText)findViewById(R.id.licenseEditText);
        myEditText4 = (EditText)findViewById(R.id.penaltyEditText);
        mySpinner = (Spinner)findViewById(R.id.faultRecordSpinner);
        cancelBtn = (Button)findViewById(R.id.cancelButton);
        submitBtn = (Button)findViewById(R.id.submitButton);
    }
}
```

③ 创建 showAlert()方法显示对话框。

```
// 显示对话框
    private void showAlert(String msg){
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage(msg)
              .setCancelable(false)
              .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                  public void onClick(DialogInterface dialog, int id) {
                  }
              });
        // 创建对话框
        AlertDialog alert = builder.create();
        // 显示对话框
        alert.show();
    }
```

④ 创建 makeEntity()方法，封装要提交的参数。在用户登录模块和信息查询模块我们也有要传递的查询参数，在这两个模块中我们是直接添加到 URL 后面，作为查询参数提交的。但是，如果要提交的参数比较多，那么使用这种方法并不方便。这里我们使用 List 来封装要发送的参数，List 里面添加的对象必须是 NameValuePair 类的实例，这里我们使用 BasicNameValuePair 类来封装提交参数。并且将要发送的参数通过 UrlEncodedFormEntity 类进行字符转换。

```
// 封装要提交的参数
private UrlEncodedFormEntity makeEntity(){
    // 姓名
    String name = myEditText1.getText().toString();
    // 身份证号码
    String idno = myEditText2.getText().toString();
    // 驾照
    String license = myEditText3.getText().toString();
    // 处罚
    String penalty = myEditText4.getText().toString();
    // 记录
    String faultRecord = mySpinner.getSelectedItem().toString();
    // 实例化List
    List<NameValuePair> params = new ArrayList<NameValuePair>();
    // 创建日期
    Date date = new Date();
    String dateStr = new SimpleDateFormat("yy-MM-dd HH:mm").format(date);

    // 添加参数
    params.add(new BasicNameValuePair("name", name));
    params.add(new BasicNameValuePair("idno", idno));
    params.add(new BasicNameValuePair("license", license));
    params.add(new BasicNameValuePair("penalty", penalty));
    params.add(new BasicNameValuePair("faultRecord", faultRecord));
    params.add(new BasicNameValuePair("createTime", dateStr));
    try {
```

```
        // 返回封装好的表单项
        return new UrlEncodedFormEntity(params,HTTP.UTF_8);
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
    return null;
}
```

⑤ 创建 submit()方法提交请求。

```
// 提交请求
private boolean submit(){
    // 请求 URL
    String url = HttpUtil.BASE_URL+"servlet/VehicleFaultInfoServlet";
    // 获得 HttpPost 对象
    HttpPost request = HttpUtil.getHttpPost(url);
    // 设置请求参数项
    request.setEntity(makeEntity());
    // 获得查询结果
    String result= HttpUtil.queryStringForPost(request);
    // 判断查询结果
    if(result!=null&&result.equals("1"))return true;
    return false;
}
```

⑥ 为提交按钮添加单击事件监听器，调用 submit()方法，提交请求。

```
// 响应提交按钮事件
submitBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 判断验证
        if(validate()){
            // 判断提交结果
            if(submit()){
                showAlert("提交成功！");
            }else{
                showAlert("提交失败！");
            }
        }else{
            showAlert("输入错误，请重新输入！");
        }
    }
});
```

⑦ 为取消按钮添加单击事件监听器，调用 finish()方法，取消提交请求。

```
// 响应取消按钮事件
cancelBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 结束当前 Activity
        finish();
```

```
    }
});
```

程序运行结果如图 15.26 所示。

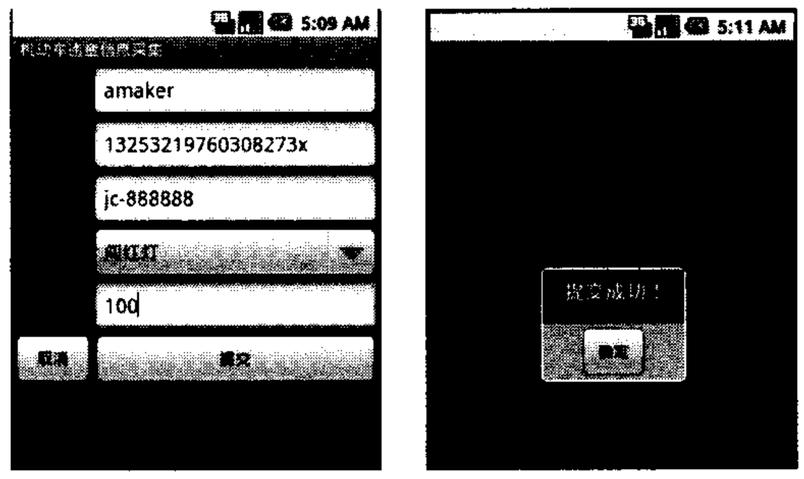


图 15.26　机动车违章信息采集运行结果

15.4.9　机动车违章信息采集服务器端实现

该模块的服务器端基本处理流程和前面的两个模块相似，创建 Servlet 响应客户端请求，获得请求参数。将请求参数封装到类里面，调用后台 DAO 实现信息的保存。具体步骤说明如下。

① 在工程的 com.amaker.entity 包中创建名称为“VehicleFaultInfo”的实体类。

```
package com.amaker.entity;
/**
 *
 * @author 郭宏志
 * 机动车违章信息采集实体类
 */
public class VehicleFaultInfo {
    // 编号
    private int id;
    // 姓名
    private String name;
    // 身份证号码
    private String idno;
    // 创建时间
    private String createTime;
    // 驾照
    private String license;
    // 违章记录
    private String faultRecord;
    // 罚款
    private double penalty;
    getters()….
    Setters()…
}
```

② 在 com.amaker.dao 包中创建名称为“VehicleDao”的 DAO 接口。

```
package com.amaker.dao;
```

```
import com.amaker.entity.VehicleFaultInfo;
/**
 *
 * @author 郭宏志
 * 保存信息的 Dao 接口
 */
public interface VehicleDao {
   public boolean save(VehicleFaultInfo v);
}
```

③ 在 com.amaker.dao.impl 包中，创建名称为“VehicleFaultInfoDaoImpl”的 DAO 接口实现类。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import com.amaker.dao.VehicleDao;
import com.amaker.entity.VehicleFaultInfo;
import com.amaker.util.DBUtil;

/**
 *
 * @author 郭宏志
 * 保存机动车违章信息 Dao 接口实现类
 */
public class VehicleFaultInfoDaoImpl implements VehicleDao {
   // 保存机动车违章记录
   public boolean save(VehicleFaultInfo v) {
      // 插入 SQL 语句
      String sql = " insert into vehicleinfotbl(name,idno,license,createTime,
faultRecord,penalty)" +
            " values(?,?,?,?,?,?) ";
      // 数据库工具类
      DBUtil util = new DBUtil();
      // 获得数据库连接
      Connection conn = util.openConnection();
      try {
         // 创建预定义语句
         PreparedStatement pstmt = conn.prepareStatement(sql);
         // 设置参数
         pstmt.setString(1, v.getName());
         pstmt.setString(2, v.getIdno());
         pstmt.setString(3, v.getLicense());
         pstmt.setString(4, v.getCreateTime());
         pstmt.setString(5, v.getFaultRecord());
         pstmt.setDouble(6, v.getPenalty());
         // 是否插入成功
         if (pstmt.executeUpdate() > 0)
            return true;
```

```
            else
                return false;
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
        return false;
    }
}
```

④ 在 com.amaker.servlet 包中创建响应客户端请求的 Servlet。该 Servlet 将获得请求参数，将其封装到实体类中，调用 DAO 保存信息，通过 Response 返回响应结果给客户端。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.VehicleDao;
import com.amaker.dao.impl.VehicleFaultInfoDaoImpl;
import com.amaker.entity.VehicleFaultInfo;

/**
 *
 * @author 郭宏志
 * 响应机动车违章信息保存的Servlet
 */
public class VehicleFaultInfoServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 设置响应类型
        response.setContentType("text/html");
        // 获得打印输出流
        PrintWriter out = response.getWriter();
        // 姓名
        String name = request.getParameter("name");
        // 身份证号码
        String idno = request.getParameter("idno");
        // 驾照
        String license = request.getParameter("license");
        // 违章记录
        String faultRecord = request.getParameter("faultRecord");
        // 创建时间
        String createTime = request.getParameter("createTime");
        // 罚款
```

```
        double penalty = Double.parseDouble(request.getParameter("penalty"));

        // 封装到实体类中
        VehicleFaultInfo v = new VehicleFaultInfo();
        // 设置属性
        v.setName(name);
        v.setCreateTime(createTime);
        v.setIdno(idno);

        v.setFaultRecord(faultRecord);
        v.setPenalty(penalty);
        v.setLicense(license);
        // 调用 DAO 保存信息
        VehicleDao dao = new VehicleFaultInfoDaoImpl();
        if (dao.save(v))
            // 响应保存结果
            out.print("1");
        else
            // 响应保存结果
            out.print("0");
        // 刷新输出流
        out.flush();
        // 关闭
        out.close();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 调用 doGet 方法
        doGet(request, response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
    // 构造方法
    public VehicleFaultInfoServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
}
```

至此机动车违章信息采集模块的前后台程序已全部完成。

15.4.10 文件上传子菜单实现

单击文件上传菜单将进入文件上传子菜单，文件上传功能包括图片上传、语音上传和文件上传。文件上传子菜单实现和上面各个模块的子菜单实现类似。实现步骤说明如下。

① 在 com.amaker.mp.fileupload 包中创建名称为“FileUploadActivity”的 Activity。
② 覆盖 onCreate()方法，声明菜单项数组，并将其设置为 ListView 的列表项。
③ 覆盖 onListItemClick()方法，响应菜单项的单击事件。

```
package com.amaker.mp.fileupload;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
/**
 *
 * @author 郭宏志
 * 文件上传子菜单
 */
public class FileUploadActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 菜单项数组
        String[] menus = { "现场照片上传", "语音上传", "文件上传"};
        // 将菜单项数组设置为 ListView 的列表项展示
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, menus));
        getListView().setTextFilterEnabled(true);
    }

    // 菜单项单击事件
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Intent intent = null;
        switch (position) {
        case 0:
            intent = new Intent(FileUploadActivity.this,UploadPicActivity.class);
            startActivity(intent);
            break;
        case 1:
            break;
        case 2:
            break;
        case 3:
            break;
        }
    }
}
```

15.4.11　现场照片上传 Android 客户端实现

现场照片上传是指公安人员在外办公过程当中，遇到一些需要保存现场证据的情况

下，将拍摄的照片上传到服务器的过程。例如，交警经常遇到交通事故，为了不影响正常通行，他将现场照片拍下，让事故车辆离开现场，事后根据现场照片证据进行解决。

上传文件比较特殊，不能通过直接发送 HTTP 请求传递参数的方式来实现，也不能通过 NameValuePair 的方式封装表单数据，只能通过 I/O 流的方式实现。所以我们必须了解上传文件时以 I/O 流的方式实现表单的结构。下面我们做一个实验来看看表单的结构。

① 创建一个 Web 工程，并创建一个上传文件的表单。

```
<form name="f1" id="f1" action="servlet/MyServlet" method="post" enctype="multi-
part/form-data">
        <input type="text" name="login" id="login">
        <input type="password" name="password" id="password">
        <input type="file" name="f">
        <input type="submit" value="Save...">
</form>
```

② 创建一个 Servlet，响应表单的提交请求，并将请求信息输出到控制台。

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
// 获得输入流
ServletInputStream sis = request.getInputStream();
// 缓冲区
byte buffer[] = new byte[1024];
// 控制台输出流
PrintStream out = System.out;
// 循环将内容打印到控制台
int len = sis.read(buffer, 0, 1024);
while (len != -1)
{
   out.write(buffer, 0, len);
   len = sis.readLine(buffer, 0, 1024);
}
out.close();
sis.close();
}
```

③ 控制台的打印信息如下所示。

```
-----------------------------7d92033a409a0
Content-Disposition: form-data; name="login"

aaa
-----------------------------7d92033a409a0
Content-Disposition: form-data; name="password"

aaa
-----------------------------7d92033a409a0
Content-Disposition: form-data; name="f"; filename="d:\我的文档\桌面\temp.txt"
Content-Type: text/plain
```

```
Hello !
-----------------------------7d92033a409a0--
```

④ 请求输入流内容分析。

4.1 第一行是一个分隔符，分隔符内容可以任意，分隔符结尾是一个回车换行（\r\n）。

4.2 第二行是内容布置描述，标示提交表单字段的名称，结尾也是一个回车换行（\r\n）。

4.3 第三行又是一个回车换行（\r\n）。

4.4 第四行是表单字段的内容，结尾也是一个回车换行（\r\n）。

4.5 第五、六、七、八行重复以上4.1、4.2、4.3、4.4步骤，只不过字段内容不同。

4.6 第九行是文件上传字段，除了有字段名称，还有文件名称和内容类型。

4.7 最后是结束分隔符，注意要以“--”结尾。

⑤ 明白了以上的表单请求结构，我们就可以根据规律自己拼一个上传文件的提交表单了。

文件上传是要将用户提交的文件上传到后台服务器，并且将上传的信息保存到数据库中方便以后查询。要添加的信息有上传时间、文件描述、文件上传路径等内容。

在选择上传文件时，由于 Android 没有文件浏览管理器，所以在这里我们自己实现了一个文件浏览管理器，供用户选择上传文件。

现场照片上传功能需要的步骤说明如下。

① 创建文件浏览器。

1.1 在工程的 res\layout\目录下创建布局文件“file_explorer.xml”。添加一个 ListView 和一个 TextView 用于显示文件列表和列表项。

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

  <ListView
       android:id="@id/android:list"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"/>
  <TextView
       android:id="@id/android:empty"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/no_files"/>

</LinearLayout>
```

1.2 在 com.amaker.mp.fileupload 包下创建名称为“FileExplorerActivity”的 Activity，该 Activity 继承 ListActivity。在其顶部声明 List 保存文件列表。覆盖 onCreate()方法，设置当前内容视图。

```
package com.amaker.mp.fileupload;
```

```
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.amaker.mp.R;

/**
 *
 * @author 郭宏志
 * 文件浏览器
 */
public class FileExplorerActivity extends ListActivity {
    // 封装文件选项
    private List<String> items = null;

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        // 设置当前Activity界面布局
        setContentView(R.layout.file_explorer);
        // 文件父目录
        fill(new File("/").listFiles());
    }
}
}
```

1.3 创建 fill()方法，该方法首先添加一个返回根目录的列表项，接着循环将文件数组内容添加到 ListView 进行展示。

```
    // 使用文件数组填充列表项
    private void fill(File[] files) {
        // 文件项列表
        items = new ArrayList<String>();
        // 添加返回
        items.add(getString(R.string.to_top));
        // 遍历文件
        for (File file : files)
            items.add(file.getPath());
        // 实例化适配器
        ArrayAdapter<String> fileList = new ArrayAdapter<String>(this,
                R.layout.file_row, items);
        // 为List设置适配器
        setListAdapter(fileList);
    }
```

1.4 创建 fillWithRoot()方法将根目录中的内容添加到 ListView。

```
// 使用根目录下的文件填充列表项
private void fillWithRoot() {
    fill(new File("/").listFiles());
}
```

1.5 覆盖 onListItemClick()方法，当前列表项是目录时继续下一级目录展示，当前列表项是文件时返回文件路径。

```
// 响应列表项的单击事件，如果是文件返回当前文件路径
protected void onListItemClick(ListView l, View v, int position, long id) {
    // 当前类别位置
    int selectionRowID = (int) position;
    // 返回父目录
    if (selectionRowID == 0) {
        fillWithRoot();
    } else {
        // 实例化文件对象
        File file = new File(items.get(selectionRowID));
        // 判断是否目录
        if (file.isDirectory()) {
            fill(file.listFiles());
        } else {
            // 返回结果
            String path = file.getAbsolutePath();
            Intent intent = this.getIntent();
            intent.putExtra("filePath", path);
            FileExplorerActivity.this.setResult(0, intent);
            FileExplorerActivity.this.finish();
        }
    }
}
}
```

② 在工程的 res\layout\目录下，创建名称为“upload_pic.xml”的布局文件。布局文件通过 LinearLayout 嵌套 TableLayout 实现布局管理。添加两个 TextView 和两个 EditText 用于用户输入照片描述和上传照片路径。添加两个按钮用于选择照片和提交表单。

```
<?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">

    <TableLayout android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:stretchColumns="1">

        <TableRow>
            <TextView
                android:text="图片描述："
                android:id="@+id/picDescTextView"
                android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"></TextView>

        <EditText
            android:text=""
            android:id="@+id/picDescEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"></EditText>
    </TableRow>

    <TableRow>
        <TextView
            android:text="图片路径: "
            android:id="@+id/picPathTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></TextView>

        <EditText
            android:text=""
            android:id="@+id/picPathEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"></EditText>
    </TableRow>

    <TableRow android:gravity="right">
        <Button
            android:text="选择图片"
            android:id="@+id/selectButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

        <Button
            android:text="提交"
            android:id="@+id/submitPicButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>

    </TableRow>
  </TableLayout>

</LinearLayout>
```

③ 在 com.amaker.mp.fileupload 包下创建名称为“UploadPicActivity”的 Activity。在其顶部声明使用到的视图组件，覆盖 onCreate()方法，并在其中实例化使用到的视图组件。

```
package com.amaker.mp.fileupload;
import java.io.FileInputStream;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.amaker.mp.R;
import com.amaker.util.FormFile;
import com.amaker.util.HttpFormUtil;
import com.amaker.util.HttpUtil;

/**
 *
 * @author 郭宏志 现场照片上传实现类
 */
public class UploadPicActivity extends Activity {

    // 声明使用到的视图组件
    private Button submitBtn, selectBtn;
    private EditText myEditText,myEditText2;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.upload_pic);
        // 实例化使用到的视图组件
        myEditText = (EditText) findViewById(R.id.picPathEditText);
        myEditText2 = (EditText) findViewById(R.id.picDescEditText);

        submitBtn = (Button) findViewById(R.id.submitPicButton);
        selectBtn = (Button) findViewById(R.id.selectButton);
    }
}
```

④ 覆盖 onActivityResult()方法，获得选择文件的路径，并赋值给 EditText。

```
    // 返回文件路径
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == 1) {
            String path = data.getStringExtra("filePath");
            myEditText.setText(path);
        }
    }
```

⑤ 声明 validate()方法验证文件类型是图片。

```
    // 验证文件类型
    private boolean validate(String fileName) {
        int idx = fileName.indexOf(".");
        String subfix = fileName.substring(idx);
```

```
        if (fileName.equals(""))
            return false;
        if (!subfix.equals("jpg") || !subfix.equals("png")
                || !subfix.equals("jpeg")) {
            return false;
        } else {
            return true;
        }
    }
```

⑥ 在 com.amaker.util 包中创建名称为“FormFile”的工具类，该类用来封装要上传的文件信息。包括文件名称、字段名称、文件类型和字节流数据。

```
package com.amaker.util;
/**
 *
 * @author 郭宏志
 * 封装上传文件属性
 */
public class FormFile {
    private byte[] data; // 文件字节流
    private String filname; // 文件名称
    private String formname; // 表单属性名称
    private String contentType = "image/jpeg"; //内容类型

    /*
     * 构造方法
     */
    public FormFile(String filname, byte[] data, String formname,
            String contentType) {
        this.data = data;
        this.filname = filname;
        this.formname = formname;
        if (contentType != null)
            this.contentType = contentType;
    }
    gettters()…
    setters()…
}
```

⑦ 在 com.amaker.util 包中创建名称为“HttpFormUtil”的类。该类是文件上传的核心，根据本节开篇介绍的方法，动态拼上传表单，发送请求。

```
package com.amaker.util;

import java.io.DataOutputStream;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
```

```
/**
 *
 * @author 郭宏志
 * 提交表单工具类，同时可以上传文件
 */
public class HttpFormUtil {
    /**
     * @param actionUrl 服务器请求路径
     * @param params 普通文本域集合
     * @param files 上传文件数组
     * @return 上传成功返回 "1"，失败返回"0"
     */
    public static String post(String actionUrl, Map<String, String> params,
            FormFile[] files) {

        try {
            String enterNewline = "\r\n";  // 回车换行
            String fix = "--";             // 分隔符前后缀
            String boundary = "######";    // 分隔符

            URL url = new URL(actionUrl);  // URL 对象
                                                                    // 打开 Http 连接
            HttpURLConnection con = (HttpURLConnection) url.openConnection();
            con.setDoInput(true);          // 可以输入
            con.setDoOutput(true);         // 可以输出
            con.setUseCaches(false);       // 禁止使用缓存
            con.setRequestMethod("POST"); // 请求方法为 POST
            con.setRequestProperty("Connection", "Keep-Alive");      // 连接状态
            con.setRequestProperty("Charset", "UTF-8");              // 字符集
            con.setRequestProperty("Content-Type",
                    "multipart/form-data;boundary=" + boundary);     // 内容类型
                                                                     // 获得输出流
            DataOutputStream ds = new DataOutputStream(con.getOutputStream());

            Set<String> keySet = params.keySet();
            Iterator<String> it = keySet.iterator();

            // 循环写入普通文本域
            while(it.hasNext()){
                String key = it.next();
                String value = params.get(key);
                ds.writeBytes(fix + boundary + enterNewline);
                ds.writeBytes("Content-Disposition: form-data; "
                        + "name=\"" + key + "\"" + enterNewline);
                ds.writeBytes(enterNewline);
                ds.writeBytes(value);
                ds.writeBytes(enterNewline);
            }
            // 循环写入上传文件
            if(files!=null&&files.length>0){
                for (int i = 0; i < files.length; i++) {
```

```
                    ds.writeBytes(fix + boundary + enterNewline);
                    ds.writeBytes("Content-Disposition: form-data; "
                            + "name=\"" + files[0].getFormname() + "\"" +";filen-
ame=\"" + files[0].getFilname() + "\"" + enterNewline);
                    ds.writeBytes(enterNewline);
                    byte[] buffer = files[0].getData();

                    ds.write(buffer);
                    ds.writeBytes(enterNewline);
                }
            }

            ds.writeBytes(fix + boundary + fix + enterNewline);
            ds.flush();

            // 获得响应结果
            InputStream is = con.getInputStream();
            int ch;
            StringBuffer b = new StringBuffer();
            while ((ch = is.read()) != -1) {
                b.append((char) ch);
            }

            ds.close();

            return b.toString().trim();
        } catch (Exception e) {
            return "" + e;
        }
    }
}

}
```

⑧ 在 UploadPicActivity 类中创建 uploadFile()方法，调用以上工具类上传文件。

```
// 上传文件
private String uploadFile(String fileName,String fileDesc) {
    String urlStr = HttpUtil.BASE_URL + "servlet/UploadFileServlet";
    String result = "0";
    FileInputStream fStream;
    try {
        fStream = new FileInputStream(fileName);
        byte[] buffer = new byte[fStream.available()];
        fStream.read(buffer);
        FormFile ff = new FormFile(fileName,buffer,"picFile","image/gif");

        Map<String, String> params = new HashMap<String, String>();
        Date date = new Date();
        String dateStr = new SimpleDateFormat("yy-MM-dd HH:mm").format(date);

        params.put("uploadtime", dateStr);
        params.put("filedesc", fileDesc);
```

```
        result = HttpFormUtil.post(urlStr, params, new FormFile[]{ff});

    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
```

⑨ 为选择文件按钮添加单击事件，调用文件浏览器，选择上传文件。

```
selectBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(UploadPicActivity.this,
                FileExplorerActivity.class);
        startActivityForResult(intent, 1);
    }
});
```

⑩ 为提交按钮添加单击事件，提交上传文件。

```
    submitBtn.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String fileName = myEditText.getText().toString();
        if (validate(fileName)) {
            showAlert("无效图片文件！");
        } else {
            String fileDesc = myEditText2.getText().toString();
            if (uploadFile(fileName,fileDesc).equals("1")) {
                showAlert("上传成功！");
            } else {
                showAlert("上传失败！");
            }
        }
    }
});
```

程序运行结果如图 15.27 所示。

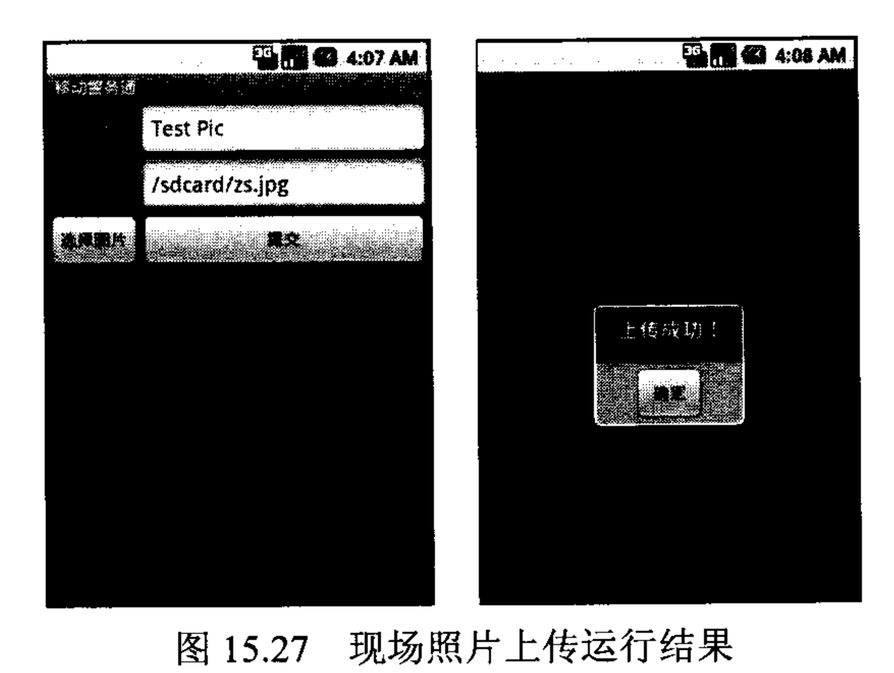


图 15.27　现场照片上传运行结果

15.4.12 现场照片上传服务器端实现

要实现服务器端的文件上传，这里我们用到的是Apache的commons-fileupload工具类，下载地址为 http://commons.apache.org/downloads/download_fileupload.cgi。另外，还需要一个 commons-io 包，下载地址为 http://commons.apache.org/io/。下载后将这两个文件复制到 classpath 下，本工程可以复制到 lib 目录下。有了这两个 jar 文件就可以实现文件上传功能了。

现场照片上传服务器端实现步骤说明如下。

① 在 com.amaker.entity 包中创建实体类 UploadFile 类。

```
package com.amaker.entity;
/**
 *
 * @author 郭宏志
 * 实体类对应数据库 uploadfiletbl 表
 */
public class UploadFile {
   private int id;             // 编号
   private String uploadTime; // 上传时间
   private String fileDesc;   // 文件描述
   private String filePath;   // 文件保存路径
   getters()....
   setters()....
}
```

② 在 com.amaker.dao 包中创建上传文件 DAO 接口 UploadFileDao。

```
package com.amaker.dao;
import com.amaker.entity.UploadFile;

/**
 *
 * @author 郭宏志
 * 文件上传接口
 */
public interface UploadFileDao {
   public boolean save(UploadFile f);
}
```

③ 在 com.amaker.dao.impl 包中创建上传文件接口实现类 UploadFileDaoImpl。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import com.amaker.dao.UploadFileDao;
import com.amaker.entity.UploadFile;
```

```
import com.amaker.util.DBUtil;
/**
 *
 * @author 郭宏志
 * 上传文件 DAO 实现
 */
public class UploadFileDaoImpl implements UploadFileDao {
    // 保存上传文件
    public boolean save(UploadFile f) {
        String sql = " insert into UploadFileTbl(uploadTime,fileDesc,filePath)"
                + " values(?,?,?)  ";
        DBUtil util = new DBUtil();
        Connection conn = util.openConnection();
        try {
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, f.getUploadTime());
            pstmt.setString(2, f.getFileDesc());
            pstmt.setString(3, f.getFilePath());

            if (pstmt.executeUpdate() > 0)
                return true;
            else
                return false;
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
        return false;
    }
}
```

④ 在 com.amaker.servlet 包中创建 UploadFileServlet。

```
package com.amaker.servlet;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
```

```
import com.amaker.dao.UploadFileDao;
import com.amaker.dao.impl.UploadFileDaoImpl;
import com.amaker.entity.UploadFile;

/**
 *
 * @author 郭宏志
 * 文件上传 Servlet
 */
public class UploadFileServlet extends HttpServlet {

    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 获得文件项工厂
        FileItemFactory factory = new DiskFileItemFactory();
        // 获得 Servlet 文件上传对象
        ServletFileUpload upload = new ServletFileUpload(factory);

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        UploadFileDao dao = new UploadFileDaoImpl();
        UploadFile uf = new UploadFile();
        String tempPath = "";
        try {
            // 解析请求
            List items = upload.parseRequest(request);
            Iterator iter = items.iterator();
            // 迭代文件域
            while (iter.hasNext()) {
                FileItem item = (FileItem) iter.next();
                // 判断是普通文本域
                if (item.isFormField()) {
                    String fieldName = item.getFieldName();
                    if(fieldName.toLowerCase().equals("uploadtime")){
                        uf.setUploadTime(item.getString());
                    }

                    if(fieldName.toLowerCase().equals("filedesc")){
                        uf.setFileDesc(item.getString());
                    }
                } else {
                    // 判断是文件域
                    String fullFileName = item.getName();
                    int idx = fullFileName.lastIndexOf(".");
                    String subfix = fullFileName.substring(idx);
                    String fileName = new Date().getTime() + subfix;
                    String path = this.getServletContext().getRealPath(
                            "\\upload");

                    File f = new File(path + "\\" + fileName);
```

```
                tempPath = "\\upload\\"+fileName;
                item.write(f);

            }
        }

        uf.setFilePath(tempPath);
        dao.save(uf);

    } catch (Exception e) {
        e.printStackTrace();
        out.print("0");
    }
    out.print("1");
    out.flush();
    out.close();
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    doPost(request, response);
}

public UploadFileServlet() {
    super();
}
public void destroy() {
    super.destroy();
}
public void init() throws ServletException {
}
}

}
```

至此现场照片上传已全部实现，本节的难点是：

- 如何动态拼上传表单
- 服务器端如何接收上传文件，并保存文件信息到数据库

15.4.13　GPS 定位功能子菜单实现

GPS 定位功能子菜单的实现和前面几个模块的子菜单实现类似，基本流程是：创建 Activity 继承 ListActivity，在 onCreate()方法中声明子菜单项，将菜单项添加到 ListView，覆盖 onListItemClick()方法响应单击事件，跳转到不同的 Activity。

```
package com.amaker.mp.location;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.ArrayAdapter;
import android.widget.ListView;

/**
 *
 * @author 郭宏志
 * GPS 定位子菜单实现
 */
public class GpsLocationActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 声明菜单项
        String[] menus = { "我的位置", "按坐标查询", "按地址查询","跟踪轨迹" };
        // 将菜单项添加到 ListView
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, menus));
        getListView().setTextFilterEnabled(true);
    }

    // 响应单击事件，跳转到不同的 Activity
    protected void onListItemClick(ListView l, View v, int position, long id) {
        Intent intent=null;
        switch (position) {
        case 0:
            intent = new Intent(GpsLocationActivity.this,MyPositionActivity.class);
            startActivity(intent);
            break;
        case 1:
            intent  =  new  Intent(GpsLocationActivity.this,QueryPositionByLLActivi-
ty.class);
            startActivity(intent);
            break;
        case 2:
            intent  =  new  Intent(GpsLocationActivity.this,
QueryPositionByAddressActivity.class);
            startActivity(intent);
            break;
        case 3:
            //intent  =  new  Intent(MainActivity.this,GpsApp
Activity.class);
            //startActivity(intent);
            break;
        }
    }
}
```

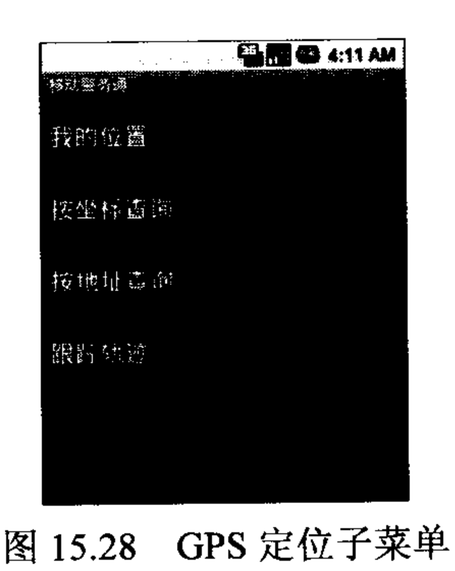


图 15.28　GPS 定位子菜单运行结果

程序运行结果如图 15.28 所示。

15.4.14　GPS 定位“我的位置”的实现

本功能是根据当前手机 GPS 提供的经纬度坐标，在 MapView 中显示用户当前的位置，

并在此基础上使用 Overlay 标记。具体实现步骤说明如下。

① 在工程的 res\layout\目录下创建布局文件“my_position.xml”。该布局文件采用 RelativeLayout 管理布局。在其中添加一个 MapView 视图组件和另外一个 LinearLayout 视图组件用于显示地图和地图控制器。要想通过 MapView 使用 Google 地图，必须获得一个 APIKey。该 APIKey 的申请过程说明如下。

1.1 获得 keystore 的 MD5 编码。这里需要使用 Java JDK 的 keytool 命令。命令格式如图 15.29 所示。

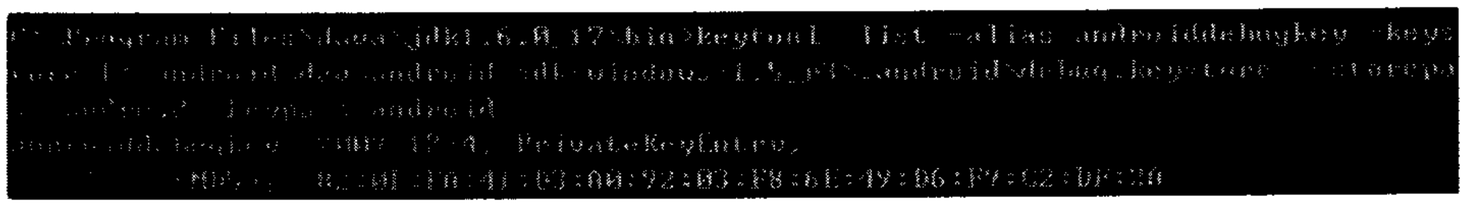


图 15.29　获得认证指纹

这里的 keystore 文件可以使用 Eclipse 开发环境下默认的 keystore 文件，位置如图 15.30 所示。

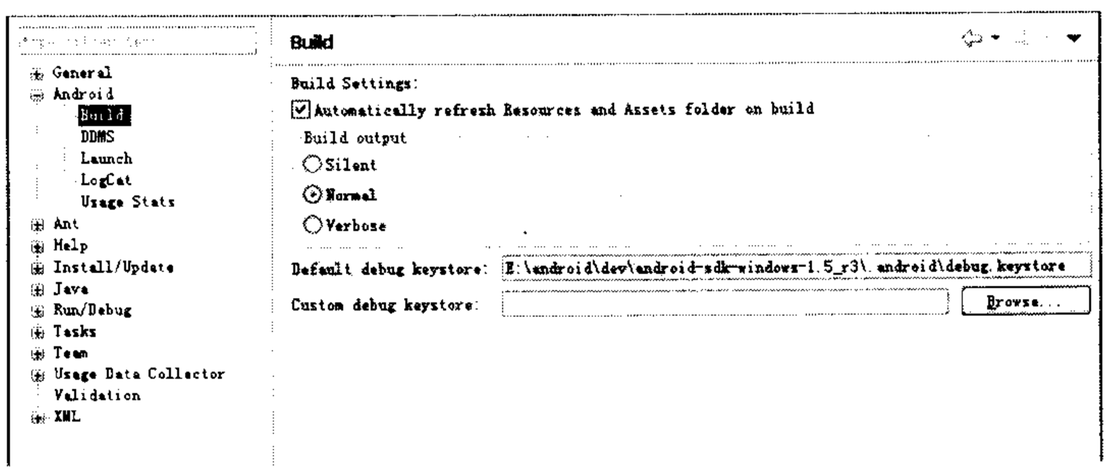


图 15.30　Eclipse 开发环境下默认的 keystore 文件位置

1.2 获得 MD5 认证指纹后到 Google 网站申请 ApiKey。申请地址为 http://code.google.com/intl/zh-CN/android/maps-api-signup.html。当输入 MD5 认证指纹后，生成结果如图 15.31 所示。

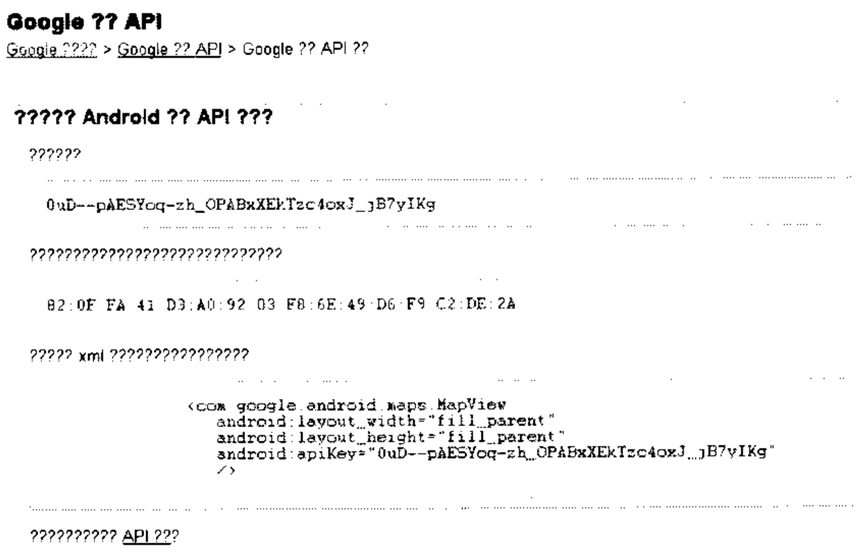


图 15.31　到 Google 网站申请 ApiKey

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

    <com.google.android.maps.MapView
        android:id="@+id/mapview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0uD--pAESYoq-zh_OPABxXEkTzc4oxJ_jB7yIKg"
    />

    <LinearLayout
    android:id="@+id/zoomview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@id/mapview"
    android:layout_centerHorizontal="true"
/>

</RelativeLayout>
```

② 在 com.amaker.mp.location 包中创建 MyPositionActivity，该类继承 MapActivity，覆盖 isRouteDisplayed()方法。在该类的顶部声明使用到的视图组件。覆盖 onCreate()方法并实例化使用到的相关视图组件。

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
package com.amaker.mp.location;

import java.util.List;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.ZoomControls;

import com.amaker.mp.R;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;

/**
```

```
 *
 * @author 郭宏志
 * 确定用户的当前位置
 */
public class MyPositionActivity extends MapActivity {

    // 声明使用到的视图组件
    private LinearLayout linearLayout;
    private MapView mapView;
    private ZoomControls mZoom;
    private LocationManager locationManager;

    private List<Overlay> mapOverlays;
    private Drawable drawable;
    private MyPositionItemizedOverlay itemizedOverlay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_position);
        // 实例化相关视图组件
        linearLayout = (LinearLayout) findViewById(R.id.zoomview);
        mapView = (MapView) findViewById(R.id.mapview);
        mZoom = (ZoomControls) mapView.getZoomControls();
        linearLayout.addView(mZoom);

        mapOverlays = mapView.getOverlays();
        drawable = this.getResources().getDrawable(R.drawable.androidmarker);

    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

③ 定义一个 locate()方法，该方法获得经纬度坐标，通过 MapController 对象定位到该点。

```
    // 获得当前经纬度信息，通过 MapController 定位到该点
    private GeoPoint locate(MapController controller) {
        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERV-
ICE);
        String provider = LocationManager.GPS_PROVIDER;
        Location location = locationManager.getLastKnownLocation(provider);

        double lat=0.0;
        double lng = 0.0;
        if (location != null) {
            lat = location.getLatitude();
            lng = location.getLongitude();
```

```
        }else{
            lat = 39.92;
            lng = 116.46;
        }

        GeoPoint point = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));
        controller.animateTo(point);
        return point;
    }
```

该方法通过 getSystemService()方法获得 LocationManager 对象。该对象通过 getLastKnownLocation()方法获得当前位置对象 Location。如果位置对象不为空则获得当前坐标，否则取得默认值（北京经纬度坐标）。创建 GeoPoint 对象，定义 MapController 的 animateTo()方法定位到该点，并返回当前 GeoPoint 对象。

④ 在 onCreate()方法中，从 MapView 中获得 MapController 对象，调用 locate()方法完成定位。

```
        MapController controller = mapView.getController();
        GeoPoint point = locate(controller);
```

⑤ 为用户当前所在位置添加标示图层。

5.1 在工程的 Drawable 目录下添加一个标示图标。

5.2 在代码中获得该图标对象 Drawable。

```
    drawable = this.getResources().getDrawable(R.drawable.androidmarker);
```

5.3 在 com.amaker.mp.location 包中创建一个 MyPositionItemizedOverlay 类，继承 ItemizedOverlay 用来显示图层标示。

```
package com.amaker.mp.location;

import java.util.ArrayList;

import android.graphics.drawable.Drawable;

import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;

/**
 *
 * @author 郭宏志
 * 图层标示类
 */
public class MyPositionItemizedOverlay extends ItemizedOverlay {

    private ArrayList<OverlayItem> mOverlays = new ArrayList<OverlayItem>();
    public MyPositionItemizedOverlay(Drawable defaultMarker) {
        super(boundCenterBottom(defaultMarker));
    }
```

```
    public void addOverlay(OverlayItem overlay) {
        mOverlays.add(overlay);
        populate();
    }
    @Override
    protected OverlayItem createItem(int i) {
         return mOverlays.get(i);
    }
    @Override
    public int size() {
        return mOverlays.size();
    }
}
```

5.4 在 onCreate()方法中为当前所在点添加图层标示。

```
MapController controller = mapView.getController();
GeoPoint point = locate(controller);

drawable = this.getResources().getDrawable(R.drawable.androidmarker);
mapOverlays = mapView.getOverlays();
itemizedOverlay = new MyPositionItemizedOverlay(drawable);
OverlayItem overlayitem = new OverlayItem(point, "", "");


itemizedOverlay.addOverlay(overlayitem);
mapOverlays.add(itemizedOverlay);
```

程序运行结果如图 15.32 所示。

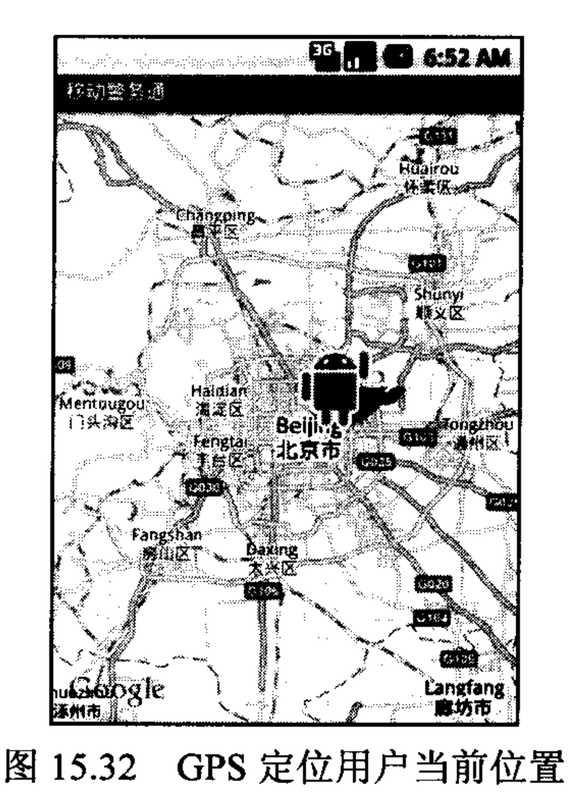


图 15.32　GPS 定位用户当前位置

15.4.15　GPS 定位“按坐标查询”的实现

“按坐标查询”实现和上一节的“我的位置”的实现类似，只不过用户可以自己输入想要查询的经纬度坐标。为了让屏幕能更多地显示地图信息，所以坐标的输入我们采用自定义输入对话框（因为 Android 没有输入对话框），并通过单击菜单来显示对话框让用户输入经纬度信息，单击确定按钮进行定位。

实现步骤说明如下。

① 在工程的 res\layout\目录下定义 query_position_by_ll.xml 布局文件，该布局文件和上一节的类似，这里不再赘述。

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

```
    <com.google.android.maps.MapView
        android:id="@+id/mapview1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0uD--pAESYoq-zh_OPABxXEkTzc4oxJ_jB7yIKg"
    />

    <LinearLayout
    android:id="@+id/zoomview1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@id/mapview"
    android:layout_centerHorizontal="true"
/>
</RelativeLayout>
```

② 在 com.amaker.mp.location 包中定义一个 QueryPositionByLLActivity 类，该类和上一节的内容类似，不同的内容在于：一是添加了一个选项菜单，用于显示输入坐标对话框；二是对话框需要自己定义。

```
<?xml version="1.0" encoding="utf-8"?>
package com.amaker.mp.location;

import java.util.List;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ZoomControls;

import com.amaker.mp.R;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;

public class QueryPositionByLLActivity extends MapActivity {
    // 声明使用到的视图组件
    private LinearLayout linearLayout;
```

```
    private MapView mapView;
    private ZoomControls mZoom;
    private LocationManager locationManager;

    private List<Overlay> mapOverlays;
    private Drawable drawable;
    private MyPositionItemizedOverlay itemizedOverlay;

    private MapController controller;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.query_position_by_ll);
        // 实例化相关视图组件
        linearLayout = (LinearLayout) findViewById(R.id.zoomview1);
        mapView = (MapView) findViewById(R.id.mapview1);
        mZoom = (ZoomControls) mapView.getZoomControls();
        linearLayout.addView(mZoom);

        double lat = 39.92;
        double lng = 116.46;
        locate(lat,lng);
    }

    // 获得当前经纬度信息，通过 MapController 定位到该点
    private void locate(double lat,double lng) {
        if (lat==0.0&&lng==0.0) {
            lat = 39.92;
            lng = 116.46;
        }

        controller = mapView.getController();
        GeoPoint point = new GeoPoint((int) (lat * 1E6), (int) (lng * 1E6));
        controller.animateTo(point);

        drawable = this.getResources().getDrawable(R.drawable.androidmarker);
        mapOverlays = mapView.getOverlays();
        itemizedOverlay = new MyPositionItemizedOverlay(drawable);
        OverlayItem overlayitem = new OverlayItem(point, "", "");


        itemizedOverlay.addOverlay(overlayitem);
        mapOverlays.add(itemizedOverlay);
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}
```

2.1 在工程的 res\layout\目录下定义一个布局文件 custom_dialog.xml，该布局文件的内

容将显示在对话框中。

```
<?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">

   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">
      <TableRow>
         <TextView
            android:text="经度(longitude): "
            android:id="@+id/lngTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            ></TextView>

         <EditText
            android:text=""
            android:id="@+id/lngEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"></EditText>
      </TableRow>


      <TableRow>
         <TextView
            android:text="纬度(latitude): "
            android:id="@+id/latTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            ></TextView>

         <EditText
            android:text=""
            android:id="@+id/latEditText"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            ></EditText>
      </TableRow>

   </TableLayout>

</LinearLayout>
```

该布局文件采用 LinearLayout 嵌套 TableLayout 进行布局管理。在其中添加两个 TextView 和两个 EditText 提示用户输入经纬度坐标。

2.2 在 QueryPositionByLLActivity 类中定义一个 customDialog()方法，用来显示自定义

对话框。

```
    /*
     * 自定义对话框
     */
    private void customDialog(){
        try{
         LayoutInflater mInflater01 = LayoutInflater.from(QueryPositionByLLActivi-
ty.this);
         View myView = mInflater01.inflate(R.layout.custom_dialog, null);
         final  EditText    lngEditText=(EditText)myView.findViewById(R.id.lngEditT-
ext);
         final  EditText     latEditText=(EditText)myView.findViewById(R.id.latEdi-
tText);

          new AlertDialog.Builder(this)
          .setView(myView).setTitle("请输入经纬度坐标")
          .setPositiveButton("确定",
          new DialogInterface.OnClickListener(){
            public void onClick(DialogInterface dialog, int whichButton){
            locate(Double.parseDouble(latEditText.getText().toString()),
                    Double.parseDouble(lngEditText.getText().toString()));
            }
          }).show();
        }
        catch(Exception e)
        {
          e.printStackTrace();
        }
      }
```

该方法首先获得 LayoutInflater 实例，根据该实例对象从布局文件获得 View 对象，将该View对象添加到对话框中。对话框的单击事件获得用户输入的经纬度坐标，并定义locate方法进行重新定位。

2.3 添加选项按钮，当用户单击菜单时显示自定义对话框。

```
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, 1, 0, "显示输入坐标对话框");
    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case 1:
    customDialog();
        return true;
    }
    return false;
}
```

程序运行结果如图 15.33 所示。

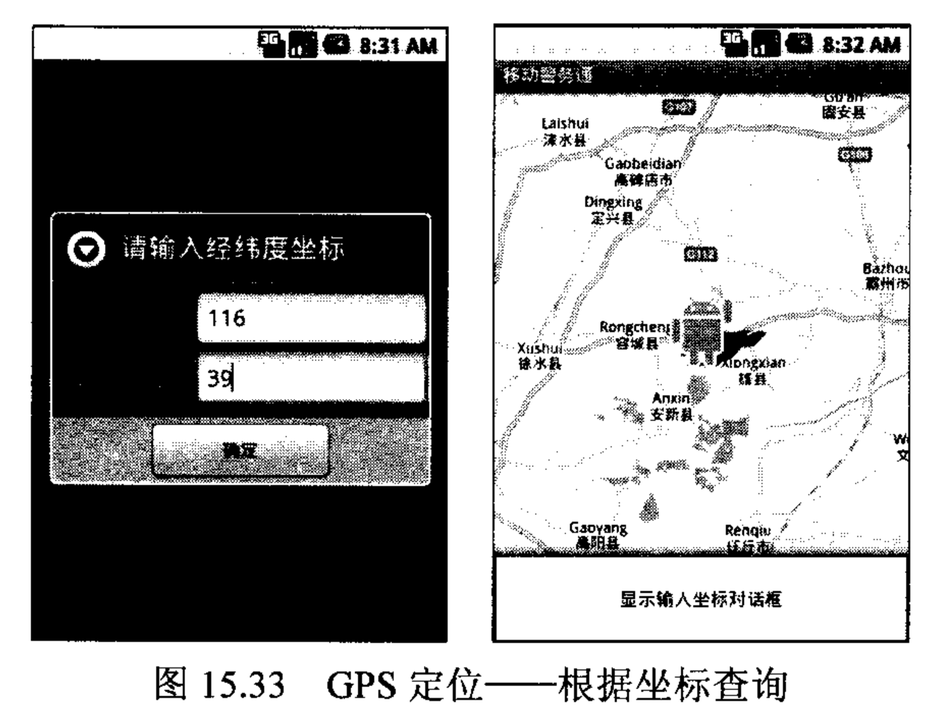


图 15.33　GPS 定位——根据坐标查询

15.4.16　GPS 定位“按地址查询”的实现

按地址查询和按坐标查询类似，只是需要把输入的地址转换成 GeoPoint 对象。具体步骤说明如下。

① 在工程的 res\layout\目录下创建布局文件 query_position_by_address.xml。

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

   <com.google.android.maps.MapView
      android:id="@+id/mapview2"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:clickable="true"
      android:apiKey="0uD--pAESYoq-zh_OPABxXEkTzc4oxJ_jB7yIKg"
   />

   <LinearLayout
   android:id="@+id/zoomview2"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_alignBottom="@id/mapview"
   android:layout_centerHorizontal="true"
/>

</RelativeLayout>
```

② 在 com.amaker.mp.location 包下创建 QueryPositionByAddressActivity 类。

```
package com.amaker.mp.location;
```

```
import java.util.List;
import java.util.Locale;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;
import android.location.Address;
import android.location.Geocoder;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ZoomControls;

import com.amaker.mp.R;
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;

public class QueryPositionByAddressActivity extends MapActivity {
    // 声明使用到的视图组件
    private LinearLayout linearLayout;
    private MapView mapView;
    private ZoomControls mZoom;

    private List<Overlay> mapOverlays;
    private Drawable drawable;
    private MyPositionItemizedOverlay itemizedOverlay;

    private MapController controller;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.query_position_by_address);
        // 实例化相关视图组件
        linearLayout = (LinearLayout) findViewById(R.id.zoomview2);
        mapView = (MapView) findViewById(R.id.mapview2);
        mZoom = (ZoomControls) mapView.getZoomControls();
        linearLayout.addView(mZoom);

        String defaultAddress = "beijing";

        GeoPoint point = getGeoByAddress(defaultAddress);
        locate(point);
```

```
    }

    // 获得当前经纬度信息，通过 MapController 定位到该点
    private void locate(GeoPoint point) {

        controller = mapView.getController();

        controller.animateTo(point);
        drawable = this.getResources().getDrawable(R.drawable.androidmarker);
        mapOverlays = mapView.getOverlays();
        itemizedOverlay = new MyPositionItemizedOverlay(drawable);
        OverlayItem overlayitem = new OverlayItem(point, "", "");


        itemizedOverlay.addOverlay(overlayitem);
        mapOverlays.add(itemizedOverlay);
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }


    public boolean onCreateOptionsMenu(Menu menu) {
        menu.add(0, 1, 0, "显示输入查询地址对话框");
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case 1:
        customDialog();
            return true;
        }
        return false;
    }


    /*
     * 根据地址获得 GeoPoint 对象
     */
    private GeoPoint getGeoByAddress(String strSearchAddress)
      {
        GeoPoint gp = null;
        try
        {
          if(strSearchAddress!="")
          {
            Geocoder geoCoder = new Geocoder(QueryPositionByAddressActivity.this,
Locale.getDefault());
            List<Address> lstAddress = geoCoder.getFromLocationName(strSearchAddress,
1);
            if (!lstAddress.isEmpty())
            {
              Address adsLocation = lstAddress.get(0);
              double geoLatitude = adsLocation.getLatitude()*1E6;
              double geoLongitude = adsLocation.getLongitude()*1E6;
```

```
            gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
          }
        }
      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
      return gp;
    }

  /*
   * 自定义对话框
   */
  private void customDialog(){
      try{
       LayoutInflater mInflater01 = LayoutInflater.from(QueryPositionByAddressAc-
tivity.this);
       View myView = mInflater01.inflate(R.layout.custom_dialog1, null);
       final EditText  addressEditText=(EditText)myView.findViewById(R.id.addres-
sEditText);

        new AlertDialog.Builder(this)
        .setView(myView).setTitle("请输入查询地址坐标")
        .setPositiveButton("确定",
        new DialogInterface.OnClickListener(){
          public void onClick(DialogInterface dialog, int whichButton){
          GeoPoint point = getGeoByAddress(addressEditText.getText().toString());
          locate(point);
          }
        }).show();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
}
```

程序运行结果如图 15.34 所示。

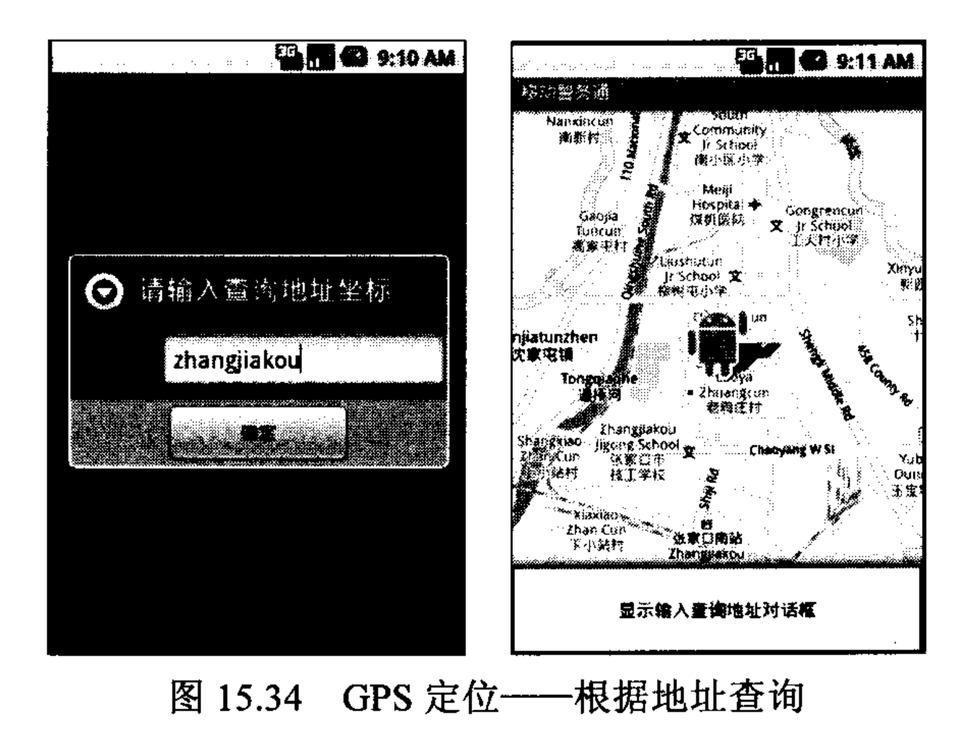
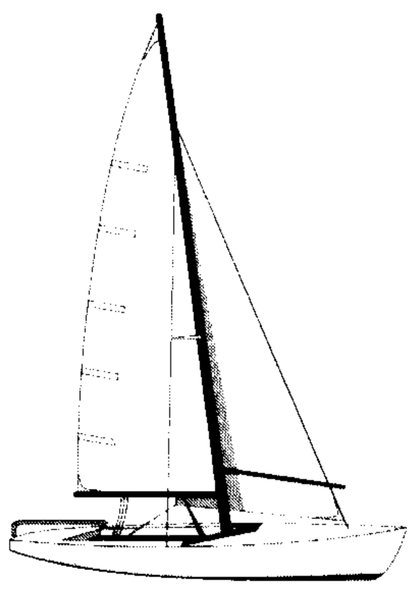


图 15.34　GPS 定位——根据地址查询



第 16 章　Android 应用案例——雷电游戏

在笔者看来，Android 中的应用大致分为三类：一是 Android 基本应用，包括传统的拨打电话、收发短信等；二是作为移动互联网的客户端网络应用，即“手机小电脑”；三是手机游戏。

随着手机硬件配置的不断提升，加上无线网络的普及。手机游戏会有更多的提升空间，例如，3D 游戏、联网游戏将大量涌现。本章首先讲述 Android 游戏开发的基本框架，接着讲述如何将 Java ME 中的游戏 API 移植到 Android，最后通过一个经典的雷电游戏实例来讲述 Android 游戏开发的全过程。

16.1 Android 游戏开发基本框架

16.1.1　Android 游戏开发基础

在使用 Android 进行游戏开发之前，掌握一些 Android 游戏开发的基础知识是非常必要的。例如，Android 中坐标的确定、屏幕的宽和高的确定、边界的确定和视图的移动等，本节将详细讲述这些基础知识。

1．Android 中的坐标系统

在 Android 系统中，屏幕的左上角是坐标系统的原点(0,0)坐标。原点向右延伸是 X 轴正方向，原点向下延伸是 Y 轴正方向，如图 16.1 所示。

2．屏幕的宽和高

为了在屏幕中的合适位置绘制图形，我们需要使用屏幕的宽和高作为参考，来确定绘制图形的位置。要获得屏幕的宽和高，首先从 Activity 对象中获得 WindowManager 对象，

然后从 WindowManager 对象中获得 Display 对象，再从 Display 对象中获得屏幕的宽和高。

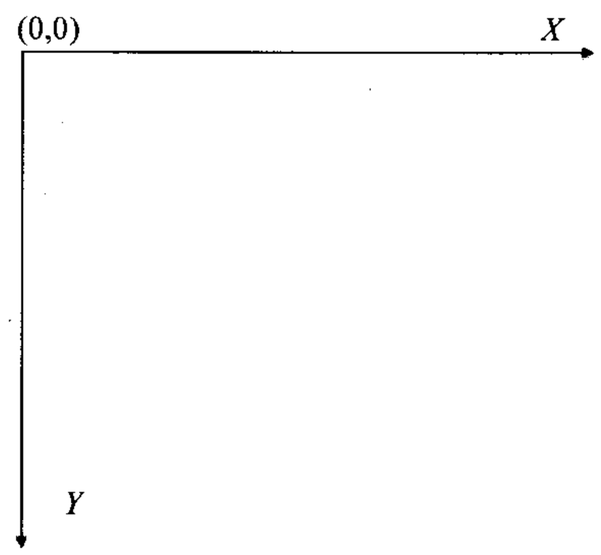


图 16.1　Android 坐标系统图

```
// 获得屏幕宽和高
WindowManager windowManager = getWindowManager();
Display display = windowManager.getDefaultDisplay();
int screenWidth = display.getWidth();
int screenHeight = display.getHeight();
```

3. 边界的确定

在很多游戏中都需要对绘制在屏幕中的视图进行边界的确定。例如，在射击类游戏中我们就需要判断玩家、敌人、子弹等视图的边界位置。边界的判断无非是对上、下、左、右屏幕边界的判断。

如果当前视图的 X 坐标小于零，则当前视图左越界。如果当前视图的 X 坐标大于屏幕的宽，则右越界。

如果当前视图的 Y 坐标小于零，则当前视图上越界。如果当前视图的 Y 坐标大于屏幕的高，则下越界。

4. 视图的移动

游戏的实现过程其实很简单，就是不断地改变视图的坐标位置，然后重新将它们绘制在屏幕上。不过这种坐标的位置改变和绘制过程是通过一定逻辑来控制实现的。视图的移动就是通过改变视图坐标位置来实现的。改变了坐标再重新绘制，给我们的感觉是视图在移动。

如果视图水平向左移动，X 坐标减小；如果视图水平向右移动，X 坐标增大。

如果视图垂直向上移动，Y 坐标减小；如果视图水平向下移动，Y 坐标增大。

16.1.2　Android 游戏开发基本框架

Android 游戏开发框架基本对象有三个：一是图层对象，该图层对象定义图层的宽和高、图层的位置、图层的移动以及绘制方法等；二是视图对象，视图对象的主要作用是绘

制图层对象、响应键盘事件和处理视图线程等；三是一个 Activity 控制游戏流程，例如启动游戏、暂停游戏、停止游戏等。

1. 图层对象

我们以面向对象的设计思想设计一个图层对象，该对象定义图层的坐标、宽和高、移动方法、位置方法、是否可见方法以及绘制方法。

```
package com.amaker.game;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Rect;

public class Layer {
    // x、y坐标
    private int x,y;
    // 图层宽和高
    private int width,height;
    // 是否可见
    private boolean isVisible;
    // 设置图层位置
    public void setPosition(int x,int y){
        this.x = x;
        this.y = y;
    }
    // 移动图层
    public void move(int dx,int dy){
        this.x+= dx;
        this.y+= dy;
    }

    // 绘制图层
    public void paint(Canvas c, int x, int y, Bitmap src, int sx, int sy,
          int w, int h) {
        Rect rect_src = new Rect();
        rect_src.left = sx;
        rect_src.right = sx +w;
        rect_src.top = sy;
        rect_src.bottom = sy+h;

        Rect rect_dst = new Rect();
        rect_dst.left = x;
        rect_dst.right = x +w;
        rect_dst.top = y;
        rect_dst.bottom = y+h;

        c.drawBitmap(src,rect_src, rect_dst, null);
    }

    public int getX() {
```

```
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }
    public int getWidth() {
        return width;
    }
    public void setWidth(int width) {
        this.width = width;
    }
    public int getHeight() {
        return height;
    }
    public void setHeight(int height) {
        this.height = height;
    }
    public boolean isVisible() {
        return isVisible;
    }
    public void setVisible(boolean isVisible) {
        this.isVisible = isVisible;
    }
}
```

2. 视图对象

视图对象主要是将图层绘制在屏幕上，另外需要响应用户交互，例如处理键盘事件、触屏事件等。下面是视图对象的典型结构。

```
package com.amaker.test;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.view.KeyEvent;
import android.view.View;
public class GameView extends View implements Runnable{
    // 图片
    private Bitmap bitmap;
    // 图层实例
    private Layer player;
    // 屏幕宽和高
    private int screenWidth,screenHeight;
    // 图层当前 x、y 坐标
```

```
    private int x,y;
    // 构造方法
    public GameView(Context context,int screenWidth,int screenHeight) {
        super(context);
        // 实例化位图
        bitmap   =   BitmapFactory.decodeResource(context.getResources(),R.drawable.
player);
        // 实例化图层
        player = new Layer(bitmap.getWidth(),bitmap.getHeight());
        // 设置图层位置
        player.setPosition((screenWidth-player.getWidth())/2, screenHeight-80);
        // 设置屏幕宽和高
        this.screenWidth = screenWidth;
        this.screenHeight = screenHeight;
        //获得焦点
        setFocusable(true);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        if(player.isVisible()){
            // 绘制图层
            player.paint(canvas,
                    player.getX(),player.getY(),bitmap, x, y,
                    player.getWidth(),player.getHeight());
        }
        super.onDraw(canvas);
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        // 处理键盘事件
        switch (keyCode) {
        case KeyEvent.KEYCODE_DPAD_UP:
            player.move(x, -3);
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            player.move(x, 3);
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            player.move(-3, y);
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            player.move(3, y);
            break;
        }
        return super.onKeyDown(keyCode, event);
    }

    @Override
    public void run() {
        // 图层线程
```

```
        while(true){
            postInvalidate();
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    // 启动线程
    public void start(){
        new Thread(this).start();
    }
}
```

3. Activity 对象

Activity 对象的主要功能是控制游戏流程，例如启动游戏、暂停游戏、结束游戏等。下面是 Activity 的典型结构。

```
package com.amaker.test;

import android.app.Activity;
import android.os.Bundle;
import android.view.Display;
import android.view.WindowManager;

public class MainActivity extends Activity {
    // 游戏视图对象
    private GameView gameView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 获得屏幕宽和高
        WindowManager windowManager = getWindowManager();
        Display display = windowManager.getDefaultDisplay();
        int screenWidth = display.getWidth();
        int screenHeight = display.getHeight();
        // 实例化
        gameView = new GameView(this,screenWidth,screenHeight);
        setContentView(gameView);
    }

    @Override
    protected void onResume() {
        // 启动游戏线程
        gameView.start();
        super.onResume();
    }
}
```

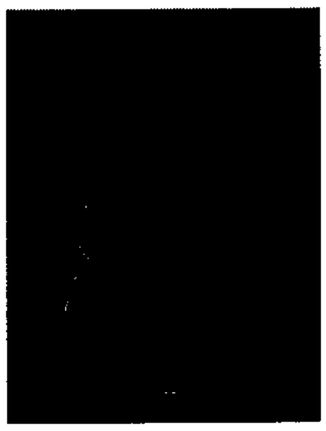


图 16.2　Android 游戏基本框架

程序运行结果如图 16.2 所示。

16.2 将 Java ME 游戏 API 嫁接到 Android

Java ME MIDP 2.0 版本之后提供了专门进行游戏开发的 API，并且非常成熟，Android 和 Java ME 都使用 Java 语言编写，API 有很多相似之处，可以非常轻松地将 Java ME 中的游戏 API 移植到 Android 中。这样一来，一是给 Java ME 的开发者提供了方便，二是减少了 Android 游戏开发的工作量。在这里我们移植了 Layer、Sprite、LayerManaer 和 TiledLayer，GameCanvas 关联的其他 API 比较多，我们不做移植。在 Java ME 里面表示图片的类是 Image，而在 Android 中表示图片的类是 Bitmap，因此将 Image 替换为 Bitmap；在 Java ME 中绘制图片的类是 Graphics，而在 Android 中绘制图片的类是 Canvas，因此将 Graphics 类替换为 Canvas 类。

16.2.1 Layer 类的移植

Layer 类描述了游戏中的可视化图层，该类是一个抽象类，它规定了基本的图层框架和基本属性。基本方法有设置图层位置、移动图层、绘制图层等；基本属性有图层当前坐标、图层宽和高、图层是否可见等。Layer 类的移植步骤说明如下。

① 下载 MIDP 2.0 源码，将 Game 包中的 Layer、Sprite、LayerManaer 和 TiledLayer 类复制到 Android 工程中，更改包名称。

② 将 Layer 类中 paint 方法的参数 Graphics 替换为 Android 中的 Canvas。

③ 在 Layer 类中添加一个绘制图片的方法 drawBitmap()。

```
package com.amaker.game;
import android.graphics.Canvas;

public abstract class Layer {
    // 坐标 x
    int x;
    // 坐标 y
    int y;
    // 图层宽
    int width;
    // 图层高
    int height;
    // 是否可见
    boolean visible = true;
    // 构造方法
    Layer(int width, int height) {
        setWidthImpl(width);
        setHeightImpl(height);
    }
    // 设置位置
    public void setPosition(int x, int y) {
```

```
        this.x = x;
        this.y = y;
    }
    // 移动
    public void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
    public final int getX() {
        return x;
    }
    public final int getY() {
        return y;
    }
    public final int getWidth() {
        return width;
    }
    public final int getHeight() {
        return height;
    }

    public void setVisible(boolean visible) {
        this.visible = visible;
    }
    public final boolean isVisible() {
        return visible;
    }
    // 绘制
    public abstract void paint(Canvas g);
    // 绘制图片
    public void drawBitmap(Canvas c, int x, int y, Bitmap src, int sx, int sy,
            int w, int h) {
        Rect rect_src = new Rect();
        rect_src.left = sx;
        rect_src.right = sx +w;
        rect_src.top = sy;
        rect_src.bottom = sy+h;

        Rect rect_dst = new Rect();
        rect_dst.left = x;
        rect_dst.right = x +w;
        rect_dst.top = y;
        rect_dst.bottom = y+h;
        c.drawBitmap(src,rect_src, rect_dst, null);
    }
    void setWidthImpl(int width) {
        if (width < 0) {
            throw new IllegalArgumentException();
        }
        this.width = width;
    }
    void setHeightImpl(int height) {
```

```
        if (height < 0) {
            throw new IllegalArgumentException();
        }
        this.height = height;
    }
}
```

16.2.2 Sprite 类的移植

Sprite 是一个基本的动画图层类，可以显示几帧动画中的其中一帧。该类还提供了图片转换和图层碰撞检测等方法。移植 Sprite 类的步骤说明如下。

① 复制 Sprite 类到 Android 工程中，更改包名称。

② 将 Sprite 类中的 Graphics 和 Image 类分别替换为 Canvas 和 Bitmap 类。

③ 修改 paint 方法的实现，将 g.drawRegion 方法替换为父类 Layer 方法中的 drawBitmap 方法。

④ 将 doPixelCollision 方法中的 Bitmap1.getRGB()、Bitmap2.getRGB()代码注释掉。

16.2.3 TiledLayer 类的移植

我们在开发游戏时会用到很多图片，有些图片会很大，例如背景图片。为了方便、高效地操作和加载这些图片，我们将图片分成若干张小图片。TiledLayer 就是这样的一个类，它是按照行列单元格的方式来显示图片的。移植 TiledLayer 类的步骤说明如下。

① 将 TiledLayer 类复制到 Android 工程中，更改包名称。

② 将 TiledLayer 类中的 Graphics 和 Image 类分别替换成 Canvas 和 Bitmap 类。

③ 修改 paint 方法的实现，将 g.drawRegion 方法替换为父类 Layer 方法中的 drawBitmap 方法。

16.2.4 LayerManager 类的移植

LayerManager 类提供了管理图层的功能，能够将多个 Layer 对象添加到其中，集中维护和显示。移植 LayerManager 类的步骤说明如下。

① 将 LayerManager 复制到 Android 工程中，更改包名称。

② 将 TiledLayer 类中的 Graphics 和 Image 类分别替换成 Canvas 和 Bitmap 类。

③ 修改 paint 方法的实现，将如下代码注释掉。

```
int clipX = g.getClipX();
int clipY = g.getClipY();
int clipW = g.getClipWidth();
int clipH =  g.getClipHeight();
g.setClip(clipX, clipY, clipW, clipH);
```

④ 在 for 循环下面添加如下代码。

```
c.restore();
c.translate(-x + viewX, -y + viewY);
```

16.3 Android 版雷电游戏的实现

雷电是一款经典的射击类游戏，该游戏最初源于日本的街机游戏，后来被移植到 PC 中，雷电游戏有 Java ME 的实现版本。本节将详细讲述 Android 版雷电游戏的实现。

16.3.1　雷电游戏简介

雷电游戏虽然有不同的版本，但是基本实现大同小异。一般都有玩家、敌人、子弹等角色。我们将这些角色定义为图层。这些图层都需要绘制在屏幕上，我们需要定义视图类绘制这些图层，并且需要控制游戏的流程，例如开始、暂停、结束等，所以需要定义 Activity 来控制游戏流程。

16.3.2 雷电游戏的实现

雷电游戏需要的实现步骤说明如下。

① 创建一个 Android 工程“Chapter16_Game_Pro”，在该工程中创建一个名称为“com.amaker.game”的包，将上一节移植好的 Java ME 游戏 API 类复制到该包中。

② 创建一个“com.amaker.game.app”包，在该包中创建一个 Player 类，该类继承 Sprite 类，代表玩家图层。在该类中定义屏幕宽/高、玩家图片宽/高、移动方向、移动速度、是否存活等属性。构造方法初始化玩家图片宽、高，定义设置屏幕宽、高的方法，设置存活方法、移动玩家和初始化方法。

```
package com.amaker.game.app;

import android.graphics.Bitmap;
import com.amaker.game.Sprite;

/**
 *
 * @author 郭宏志
 * 玩家实现类
 */
public class Player extends Sprite{
    // 定义屏幕宽和高
    private int screenWidth,screenHeight;
    // 定义玩家宽和高
    private int frameWidth,frameHeight;
    // 定义玩家是否活着
    private boolean isAlive = false;
```

```
    // 定义移动方向常量，上、下、左、右
    public static final int UP = 1;
    public static final int DOWN = 2;
    public static final int LEFT = 3;
    public static final int RIGHT = 4;
    // 定义移动速度
    public static int SPEED = 4;

    // 构造方法
    public Player(Bitmap Bitmap,int frameWidth,int frameHeight) {
        super(Bitmap);
        this.frameWidth = frameWidth;
        this.frameHeight = frameHeight;
        defineReferencePixel(frameWidth/2, frameHeight/2);
    }
    // 设置屏幕尺寸
    public void setScreenSize(int screenWidth,int screenHeight){
        this.screenHeight = screenHeight;
        this.screenWidth = screenWidth;
    }
    // 激活玩家
    public void setAlive(boolean isAlive){
        this.isAlive = isAlive;
        setPosition(screenWidth/2, screenHeight/2);
    }
    // 获得玩家存活状态
    public boolean getAlive(){
        return this.isAlive;
    }
    // 移动玩家
    public void move(int direction){
        // 向上移动
        if(direction==UP){
            move(0, -SPEED);
            if(getY()<0){
                setPosition(getX(), 0);
            }
        }
        // 向下移动
        if(direction==DOWN){
            move(0, SPEED);
            if(getY()>screenHeight-frameHeight){
                setPosition(getX(), screenHeight-frameHeight);
            }
        }

        // 向左移动
        if(direction==LEFT){
            move(-SPEED, 0);
            if(getX()<0){
                setPosition(0, getY());
            }
```

```
        }
        // 向右移动
        if(direction==RIGHT){
            move(SPEED, 0);
            if(getX()>screenWidth-frameWidth){
                setPosition(screenWidth-frameWidth,getY());
            }
        }
    }
    // 初始化玩家
    public void init(){
        SPEED = 4;
        setAlive(true);
    }
}
```

③ 定义敌人图层类，该类和玩家类中的属性和方法类似，这里不再赘述。

```
package com.amaker.game.app;

import java.util.Random;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import com.amaker.game.Sprite;
/**
 * @author 郭宏志
 * 敌人图层类
 */
public class Enemy extends Sprite {
    // 定义敌人宽和高
    private int frameWidth,frameHeight;
    // 定义屏幕宽和高
    private int screenWidth,screenHeight;
    // 判断敌人是否存活
    private boolean isAlive;
    // 随机数
    Random r;
    public Enemy(Bitmap Bitmap, int frameWidth, int frameHeight) {
        super(Bitmap, frameWidth, frameHeight);
        this.frameWidth = frameWidth;
        this.frameHeight = frameHeight;
        defineReferencePixel(frameWidth / 2, frameHeight / 2);
        r = new Random();
    }
    // 设置屏幕的宽和高
    public void setScreenSize(int screenWidth,int screenHeight){
        this.screenHeight = screenHeight;
        this.screenWidth = screenWidth;
    }

    // 设置敌人存活
    public void setAlive(){
```

```
        isAlive = true;
        int x = (r.nextInt() & 0x7fffffff)%(screenWidth-frameWidth);
        setPosition(x,0);
    }

    public void setAlive(boolean alive){
        this.isAlive = alive;
    }
    // 移动
    public void move(){
        if(isAlive){
            move(0, 3);
            if(getY()>screenHeight){
                isAlive=false;
            }
        }
    }

    public boolean isAlive(){
        return isAlive;
    }
    // 绘制敌人
    public void draw(Canvas canvas){
        if(isAlive){
            paint(canvas);
        }
    }
}
```

④ 定义子弹类，该类表示玩家按开火键发射的子弹。

```
package com.amaker.game.app;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import com.amaker.game.Sprite;
/**
 * @author 郭宏志
 * 子弹类
 */
public class Bullet extends Sprite{
    // 帧宽和高
    private int frameWidth,frameHeight;
    // 屏幕宽和高
    private int screenWidth,screenHeight;
    // 是否存活
    private boolean isAlive;
    public Bullet(Bitmap Bitmap, int frameWidth, int frameHeight) {
        super(Bitmap, frameWidth, frameHeight);
        this.frameHeight = frameHeight;
        this.frameWidth = frameWidth;
        defineReferencePixel(frameWidth/2, frameHeight/2);
```

```
    }
    // 设置屏幕尺寸
    public void setScreenSize(int screenWidth,int screenHeight){
        this.screenHeight = screenHeight;
        this.screenWidth = screenWidth;
    }
    // 激活玩家
    public void setAlive(boolean isAlive,int x,int y){
        this.isAlive = isAlive;
        setPosition(x,y);
    }

    // 激活玩家
    public void setAlive(boolean alive){
        this.isAlive = alive;
    }
    // 获得玩家存活状态
    public boolean getAlive(){
        return this.isAlive;
    }
    // 移动子弹
    public void move(){
        if(isAlive){
            move(0,-3);
        }
        if(getY()<0){
            isAlive = false;
        }
    }
    // 绘制子弹
    public void draw(Canvas c){
        if(isAlive)
        paint(c);
    }
}
```

⑤ 定义游戏视图类，该类继承 View 并实现 Runnable 接口，主要功能是根据程序逻辑将图层绘制在屏幕上，并响应键盘事件，处理游戏线程刷新屏幕，更新屏幕显示。

```
package com.amaker.game.app;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.view.KeyEvent;
import android.view.View;

/**
 *
 * @author 郭宏志
 * 游戏视图类
 */
```

```
public class GameView extends View implements Runnable{
    // 玩家 Bitmap
    private Bitmap playerBitmap;
    // 子弹 Bitmap
    private Bitmap bulletBitmap;
    // 敌人 Bitmap
    private Bitmap enemyBitmap;
    // 玩家
    private Player player;
    // 子弹
    private Bullet[] bullets = new Bullet[3];
    // 敌人
    private Enemy enemy;
    // Context
    private Context context;
    // 屏幕宽、高
    public int screenWidth,screenHeight;
    // 爆炸图片
    public Bitmap exploreBitmap;
    // 判断是否爆炸
    private boolean isExplore;

    // 构造方法
    public GameView(Context context,int screenWidth,int screenHeight) {
        super(context);
        this.context = context;
        this.screenWidth = screenWidth;
        this.screenHeight = screenHeight;
        setFocusable(true);
    }
    // 绘制方法
    @Override
    public void draw(Canvas canvas) {
        // 画玩家
        player.paint(canvas);
        // 画子弹
        for (int i = 0; i <bullets.length; i++) {
            bullets[i].draw(canvas);
        }
        // 绘制敌人
        enemy.draw(canvas);

        if(isExplore){
            canvas.drawBitmap(exploreBitmap, enemy.getY(), enemy.getX(), null);
            isExplore = false;
            enemy.setAlive(false);
        }

        super.draw(canvas);
    }

    // 键盘事件
```

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    switch (keyCode) {
    case KeyEvent.KEYCODE_DPAD_UP:
        player.move(Player.UP);
        break;
    case KeyEvent.KEYCODE_DPAD_DOWN:
        player.move(Player.DOWN);
        break;
    case KeyEvent.KEYCODE_DPAD_LEFT:
        player.move(Player.LEFT);
        break;
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        player.move(Player.RIGHT);
        break;
    case KeyEvent.KEYCODE_DPAD_CENTER:
        for (int i = 0; i < bullets.length; i++) {
            if(bullets[i].getAlive()==false){
                bullets[i].setAlive(true,
                        player.getRefPixelX()-bullets[i].getWidth()/2,
                        player.getRefPixelY()-player.getHeight());
                break;
            }
        }
        break;
    }
    return super.onKeyDown(keyCode, event);
}

// 线程方法实现
public void run() {
    load();
    init();
    while(true){
        if(!enemy.isAlive()){
            enemy.setAlive();
        }
        tick();
        postInvalidate();
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
        }
    }
}

// 加载资源
public void load(){
    // 获得玩家图片
    playerBitmap = GameHelper.getBitmap(context,R.drawable.player);
    // 实例化玩家
    player  =  new  Player(playerBitmap,playerBitmap.getWidth(),playerBitmap.get
```

```
Height());
        // 设置屏幕宽和高
        player.setScreenSize(screenWidth, screenHeight);
        // 获得子弹图片
        bulletBitmap = GameHelper.getBitmap(context,R.drawable.bullet);

        exploreBitmap = GameHelper.getBitmap(context,R.drawable.explored);
        for(int i=0;i<bullets.length;i++){
            // 实例化子弹
          bullets[i] = new Bullet(bulletBitmap,bulletBitmap.getWidth(),bulletBitmap.
getHeight());
            // 设置屏幕宽和高
            bullets[i].setScreenSize(screenWidth, screenHeight);
        }
        // 获得敌人图片
        enemyBitmap = GameHelper.getBitmap(context, R.drawable.e6);
        // 实例化敌人
        enemy = new Enemy(enemyBitmap,enemyBitmap.getWidth(),enemyBitmap.getHeight());
        // 设置屏幕宽和高
        enemy.setScreenSize(screenWidth, screenHeight);
    }

    // 初始化
    public void init(){
        // 初始化玩家
        player.init();
        for (int i = 0; i < bullets.length; i++) {
            bullets[i].setAlive(false);
        }
    }

    // 启动线程
    public void start(){
        new Thread(this).start();
    }

    public void tick(){
        for (int i = 0; i < bullets.length; i++) {
            bullets[i].move();
            if(bullets[i].collidesWith(enemy, false)){
                isExplore = true;
            }
        }
        enemy.move();

    }
}
```

⑥ 定义游戏流程控制类 MainActivity，该类继承 Activity，主要功能是控制游戏流程。

```
package com.amaker.game.app;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.Display;
import android.view.Window;
import android.view.WindowManager;

public class MainActivity extends Activity {
    // 声明游戏视图
    public GameView gameView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置窗口特征
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        // 获得屏幕宽和高
        WindowManager windowManager = getWindowManager();
        Display display = windowManager.getDefaultDisplay();
        int screenWidth = display.getWidth();
        int screenHeight = display.getHeight();
        gameView = new GameView(this,screenWidth,screen Height);
        setContentView(gameView);

        gameView.start();

    }
}
```

程序运行结果如图 16.3 所示。

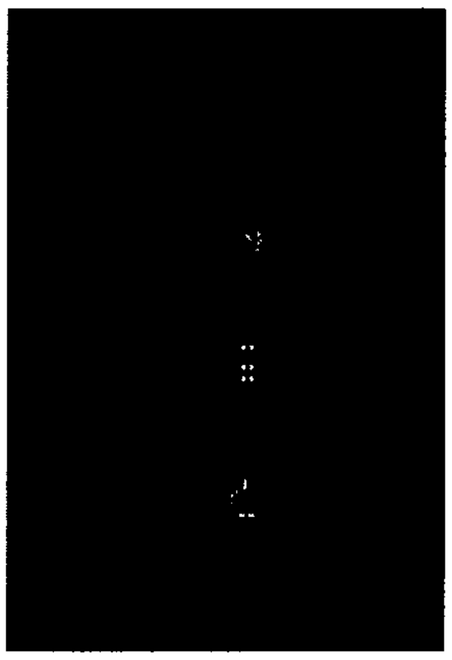
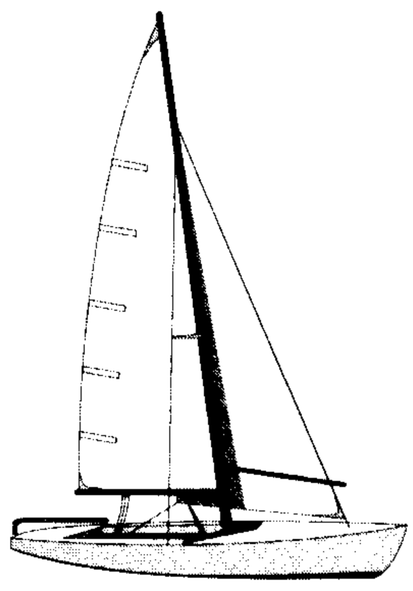


图 16.3　雷电游戏运行结果



第 17 章　Android 应用案例——备忘录

在当今快节奏的生活当中，繁忙的工作难免会让我们顾此失彼。如果能提前对所做的工作有一个计划安排，在一定的时间提醒用户完成某项工作，是一个不错的想法，备忘录就是这样一种应用程序。

在 PDA（个人数字助理）中一个重要的功能就是备忘录，也可以把它称为工作计划安排。现在的手机里也有类似的应用程序，就是为了防止遗忘某个时间的重要工作，将其记录下来，在特定时间通过声音、文字或振动等提示用户的一个应用程序。本节将详细介绍 Android 版本的一个备忘录应用程序的实现。

Android 备忘录应用程序综合应用了 Android 中各个方面的知识，包括基本应用程序组件的应用，例如，Activity、Intent、BroadcastReceiver、ContentProvider、View 等，还有 SQLite 持久数据存储等知识。通过本章的学习，读者可以更好地回顾前面学习的基本知识，并使基本知识得到综合应用。

17.1 备忘录数据存储实现

为了持久地将用户的备忘信息保存下来，需要将这些信息存储在 SQLite 数据库中。本节将详细讲述如何设计存储数据表以及如何创建 ContentProvider 来实现数据的增、删、查、改等操作。

17.1.1　数据库表设计

备忘录需要保存的信息有：编号（ID）、提醒日期（Date1）、提醒时间（Time1）、提醒内容（Content）、是否启用（ON_OFF）、是否声音提醒（ALARM）、创建时间（CREATED）等。

在操作备忘录数据表时，一些属性在几个类中使用，为此我们将其定义为一个常量类，在该常量类中声明。常量类名称为 TaskList，该类有一个内部类 Tasks 继承 BaseColumns，在该类中声明 URI 和列常量等，方便后面使用。

```
package com.amaker.ch17.app;

import android.net.Uri;
import android.provider.BaseColumns;

/**
 * @author 郭宏志
 * 备忘录常量类
 */
public final class TaskList {
   // 授权常量
   public static final String AUTHORITY = "com.amaker.provider.TaskList";

   private TaskList() {}
   // 内部类
   public static final class Tasks implements BaseColumns {

      private Tasks() {}
      // 访问 Uri
      public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
"/taskLists");
      public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.amaker.
tasklist";
      public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.
amaker.tasklist";

      // 默认排序常量
      public static final String DEFAULT_SORT_ORDER = "created DESC";
      // 内容
      public static final String CONTENT = "content";
      // 创建时间
      public static final String CREATED= "created";
      // 日期
      public static final String DATE1 = "date1";
      // 时间
      public static final String TIME1 = "time1";
      // 是否开启
      public static final String ON_OFF = "on_off";
      // 警告
      public static final String ALARM = "alarm";

   }
}
```

17.1.2　备忘录 ContentProvider 实现

为了使备忘录数据表具有通用的数据共享访问机制，需要通过 ContentProvider 来实现。

这里我们定义一个 TaskListProvider 类，该类继承 ContentProvider，实现了 onCreate、query、getType、insert、delete、update 等方法，来支持数据的增、删、查、改等操作。在该类内部还定义了一个 DatabaseHelper 内部类继承 SQLiteOpenHelper，该类提供了创建或打开数据库、创建或删除表的功能。

```
package com.amaker.ch17.app;

import java.util.HashMap;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;
import com.amaker.ch17.app.TaskList.Tasks;

/**
 *
 * @author 郭宏志
 * 提供数据维护方法
 */
public class TaskListProvider extends ContentProvider {
    // 数据库名称常量
    private static final String DATABASE_NAME = "task_list.db";
    // 数据库版本常量
    private static final int DATABASE_VERSION = 1;
    // 表名称常量
    private static final String TASK_LIST_TABLE_NAME = "taskLists";
    // 查询列集合
    private static HashMap<String, String> sTaskListProjectionMap;
    // 查询、更新条件
    private static final int TASKS = 1;
    private static final int TASK_ID = 2;
    // Uri 工具类
    private static final UriMatcher sUriMatcher;
    // 数据库工具类实例
    private DatabaseHelper mOpenHelper;
    // 内部工具类，创建或打开数据库，创建或删除表
    private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        // 创建表
```

```
        @Override
        public void onCreate(SQLiteDatabase db) {
            db.execSQL("CREATE TABLE " + TASK_LIST_TABLE_NAME + " ("
                    + Tasks._ID + " INTEGER PRIMARY KEY,"
                    + Tasks.DATE1 + " TEXT,"
                    + Tasks.TIME1 + " TEXT,"
                    + Tasks.CONTENT + " TEXT,"
                    + Tasks.ON_OFF + " INTEGER,"
                    + Tasks.ALARM + " INTEGER,"
                    + Tasks.CREATED + " TEXT"
                    + ");");
        }
        // 删除表
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS taskLists");
            onCreate(db);
        }
    }

    // 创建或打开数据库
    @Override
    public boolean onCreate() {
        mOpenHelper = new DatabaseHelper(getContext());
        return true;
    }

    // 查询
    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs,
            String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

        switch (sUriMatcher.match(uri)) {
        // 查询所有
        case TASKS:
            qb.setTables(TASK_LIST_TABLE_NAME);
            qb.setProjectionMap(sTaskListProjectionMap);
            break;
        // 根据 ID 查询
        case TASK_ID:
            qb.setTables(TASK_LIST_TABLE_NAME);
            qb.setProjectionMap(sTaskListProjectionMap);
            qb.appendWhere(Tasks._ID + "=" + uri.getPathSegments().get(1));
            break;
        default:
            throw new IllegalArgumentException("Uri 错误！ " + uri);
        }

        // 使用默认排序
        String orderBy;
```

```
        if (TextUtils.isEmpty(sortOrder)) {
            orderBy = TaskList.Tasks.DEFAULT_SORT_ORDER;
        } else {
            orderBy = sortOrder;
        }

        // 获得数据库实例
        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
        // 返回游标集合
        Cursor c = qb.query(db, projection, selection, selectionArgs, null, null,
orderBy);
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }
    // 获得类型
    @Override
    public String getType(Uri uri) {
        switch (sUriMatcher.match(uri)) {
        case TASKS:
            return Tasks.CONTENT_TYPE;
        case TASK_ID:
            return Tasks.CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("错误的 URI: " + uri);
        }
    }

    // 保存数据
    @Override
    public Uri insert(Uri uri, ContentValues initialValues) {
        if (sUriMatcher.match(uri) != TASKS) {
            throw new IllegalArgumentException("错误的 URI: " + uri);
        }
        ContentValues values;
        if (initialValues != null) {
            values = new ContentValues(initialValues);
        } else {
            values = new ContentValues();
        }
        // 获得数据库实例
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        // 保存数据返回行 ID
        long rowId = db.insert(TASK_LIST_TABLE_NAME, Tasks.CONTENT, values);
        if (rowId > 0) {
            Uri taskUri = ContentUris.withAppendedId(TaskList.Tasks.CONTENT_URI, rowId);
            getContext().getContentResolver().notifyChange(taskUri, null);
            return taskUri;
        }
        throw new SQLException("插入数据失败 " + uri);
    }
```

```
    // 删除数据
    @Override
    public int delete(Uri uri, String where, String[] whereArgs) {
        // 获得数据库实例
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        int count;
        switch (sUriMatcher.match(uri)) {
        // 根据指定条件删除
        case TASKS:
            count = db.delete(TASK_LIST_TABLE_NAME, where, whereArgs);
            break;
        // 根据指定条件和 ID 删除
        case TASK_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.delete(TASK_LIST_TABLE_NAME, Tasks._ID + "=" + noteId
                    + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : ""), whereArgs);
            break;

        default:
            throw new IllegalArgumentException("错误的 URI " + uri);
        }
        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }

    // 更新数据
    @Override
    public int update(Uri uri, ContentValues values, String where, String[] whereArgs)
{
        // 获得数据库实例
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        int count;
        switch (sUriMatcher.match(uri)) {
        // 根据指定条件更新
        case TASKS:
            count = db.update(TASK_LIST_TABLE_NAME, values, where, whereArgs);
            break;
        // 根据指定条件和 ID 更新
        case TASK_ID:
            String noteId = uri.getPathSegments().get(1);
            count = db.update(TASK_LIST_TABLE_NAME, values, Tasks._ID + "=" + noteId
                    + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : ""), whereArgs);
            break;
        default:
            throw new IllegalArgumentException("错误的 URI " + uri);
        }

        getContext().getContentResolver().notifyChange(uri, null);
        return count;
    }
```

```
    static {
        // Uri 匹配工具类
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(TaskList.AUTHORITY, "taskLists", TASKS);
        sUriMatcher.addURI(TaskList.AUTHORITY, "taskLists/#", TASK_ID);

        // 实例化查询列集合
        sTaskListProjectionMap = new HashMap<String, String>();
        // 添加查询列
        sTaskListProjectionMap.put(Tasks._ID, Tasks._ID);
        sTaskListProjectionMap.put(Tasks.CONTENT, Tasks.CONTENT);
        sTaskListProjectionMap.put(Tasks.CREATED, Tasks.CREATED);

        sTaskListProjectionMap.put(Tasks.ALARM, Tasks.ALARM);
        sTaskListProjectionMap.put(Tasks.DATE1, Tasks.DATE1);
        sTaskListProjectionMap.put(Tasks.TIME1, Tasks.TIME1);

        sTaskListProjectionMap.put(Tasks.ON_OFF, Tasks.ON_OFF);

    }
}
```

上面定义了 ContentProvider 类之后，还需要在 AndroidManifest.xml 配置文件中声明才能访问。

```
        <provider android:name="TaskListProvider"
        android:authorities="com.amaker.provider.TaskList"/>
```

17.2 备忘录列表展示

用户可能要创建很多条备忘录信息，并且需要维护这些备忘录信息，例如删除、修改、查询等功能。这里我们通过一个 ListView 来展示备忘录列表，通过选项菜单提供维护数据接口。

为了实现这一功能，我们定义一个 TaskListActivity 类，该类继承 ListActivity，通过 ListView 来显示备忘录列表。该类的主要功能有三个：一是通过前面定义的 TaskListProvider 查询备忘录列表，得到 Cursor 游标，通过 Cursor 构建 SimpleCursorAdapter，并通过 ListView 显示；二是响应 ListView 单击事件，当用户单击某条备忘录信息时，显示该条备忘录的详细信息；三是提供选项菜单添加和删除备忘录信息。

```
package com.amaker.ch17.app;

import android.app.ListActivity;
import android.content.ContentUris;
import android.content.Intent;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.Menu;
```

```
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.AdapterView.OnItemClickListener;

import com.amaker.ch17.app.TaskList.Tasks;
/**
 *
 * @author 郭宏志
 * 备忘录列表类，提供数据展示
 */
public class TaskListActivity extends ListActivity {

    // 菜单项常量
    private static final int NEW = 1;
    private static final int DEL = 2;

    // 查询列数组
    private static final String[] PROJECTION = new String[] {
            Tasks._ID,        // 0
            Tasks.CONTENT,    // 1
            Tasks.CREATED,    // 2
            Tasks.ALARM, // 3
            Tasks.DATE1,      // 4
            Tasks.TIME1,      // 5
            Tasks.ON_OFF // 6
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 获得 Intent
        final Intent intent = getIntent();
        // 设置 Uri
        if (intent.getData() == null) {
            intent.setData(Tasks.CONTENT_URI);
        }
        // 获得 ListView
        ListView listView = getListView();

        // 查询所有备忘录信息
        Cursor cursor = managedQuery(getIntent().getData(), PROJECTION, null,
                null, Tasks.DEFAULT_SORT_ORDER);
        // 创建 Adapter
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_2, cursor,
                new String[] {Tasks._ID, Tasks.CONTENT },
                new int[] { android.R.id.text1,android.R.id.text2 });
        // 将备忘录信息显示到 ListView
        setListAdapter(adapter);
```

```
// 为 ListView 添加单击事件监听器
listView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> av, View v, int position,
            long id) {
        // 通过 ID 查询备忘录信息
        Uri uri = ContentUris.withAppendedId(Tasks.CONTENT_URI, id);
        Cursor cursor = managedQuery(uri, PROJECTION, null,
                null, Tasks.DEFAULT_SORT_ORDER);

        if(cursor.moveToNext()){
            int id1 = cursor.getInt(0);
            String content = cursor.getString(1);
            String created = cursor.getString(2);
            int alarm = cursor.getInt(3);
            String date1 = cursor.getString(4);
            String time1 = cursor.getString(5);
            int on_off = cursor.getInt(6);
            Bundle b = new Bundle();
            b.putInt("id", id1);
            b.putString("content", content);
            b.putString("created", created);

            b.putInt("alarm", alarm);
            b.putString("date1", date1);
            b.putString("time1", time1);

            b.putInt("on_off", on_off);

            // 将备忘录信息添加到 Intent
            intent.putExtra("b", b);
            // 启动备忘录详细信息 Activity
            intent.setClass(TaskListActivity.this,
TaskDetailActivity.class);
            startActivity(intent);

        }
    }
});

}

// 创建选项菜单
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(0, NEW, 0, "新建");
    menu.add(0, DEL, 0, "删除");
    return true;
}
// 现象菜单项单击方法
public boolean onOptionsItemSelected(MenuItem item) {
```

```
        switch (item.getItemId()) {
        case NEW:
            // 启动备忘录详细信息 Activity
            Intent intent = new Intent();
            intent.setClass(this, TaskDetailActivity.class);
            startActivity(intent);
            return true;
        case DEL:
            return true;
        }
        return false;
    }
}
```

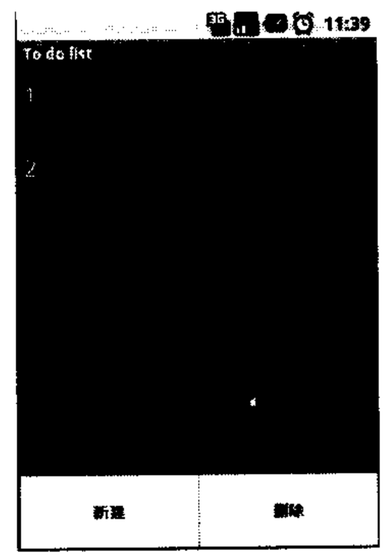


图 17.1　备忘录列表

程序运行结果如图 17.1 所示。

17.3 备忘录维护

备忘录维护是该应用程序的核心，提供了添加、修改备忘录信息的功能。备忘录信息也通过 ListView 来展示，ListView 的每一项通过自定义 View 来实现。通过单击 ListView 的每一项来设置备忘录信息。

```
package com.amaker.ch17.app;

import java.util.Calendar;

import android.app.AlarmManager;
import android.app.AlertDialog;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.app.ListActivity;
import android.app.PendingIntent;
import android.app.TimePickerDialog;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.BaseAdapter;
import android.widget.CheckedTextView;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.TimePicker;
```

```
import android.widget.AdapterView.OnItemClickListener;

import com.amaker.ch17.app.TaskList.Tasks;
/**
 *
 * @author 郭宏志
 * 备忘录详细信息类
 */
public class TaskDetailActivity extends ListActivity {
    // 备忘录信息列表
    private ListView listView = null;
    // 提醒日期
    private int mYear;
    private int mMonth;
    private int mDay;
    // 提醒时间
    private int mHour;
    private int mMinute;
    // 日期显示 TextView
    private TextView dateName, dateDesc;
    // 时间显示 TextView
    private TextView timeName, timeDesc;
    // 提醒内容 TextView
    private TextView contentName, contentDesc;
    // 是否开启提醒
    private int on_off = 0;
    // 是否声音警告
    private int alarm = 0;

    // 显示日期、时间对话框常量
    static final int DATE_DIALOG_ID = 0;
    static final int TIME_DIALOG_ID = 1;

    // 保存内容、日期、时间字符串
    private String content, date1, time1;
    // 备忘录 ID
    private int id1;
    // 多选框
    private CheckedTextView ctv1, ctv2;
    // 访问布局实例
    private LayoutInflater li;

    // 初始化方法
    private void init(Intent intent) {
        Bundle b = intent.getBundleExtra("b");
        if (b != null) {
            id1 = b.getInt("id");
            content = b.getString("content");
            date1 = b.getString("date1");
            time1 = b.getString("time1");
            on_off = b.getInt("on_off");
            alarm = b.getInt("alarm");
```

```
            if (date1 != null && date1.length() > 0) {
                String[] strs = date1.split("/");
                mYear = Integer.parseInt(strs[0]);
                mMonth = Integer.parseInt(strs[1]);
                mDay = Integer.parseInt(strs[2]);
            }

            if (time1 != null && time1.length() > 0) {
                String[] strs = time1.split(":");
                mHour = Integer.parseInt(strs[0]);
                mMinute = Integer.parseInt(strs[1]);
            }
        }

    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // 获得 ListView
        listView = getListView();
        // 实例化 LayoutInflater
        li = getLayoutInflater();
        // 设置 ListView Adapter
        listView.setAdapter(new ViewAdapter());
        // 可多选
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);

        // 获得 Calendar 实例
        final Calendar c = Calendar.getInstance();
        // 获得当前日期、时间
        mYear = c.get(Calendar.YEAR);
        mMonth = c.get(Calendar.MONTH);
        mDay = c.get(Calendar.DAY_OF_MONTH);
        mHour = c.get(Calendar.HOUR_OF_DAY);
        mMinute = c.get(Calendar.MINUTE);

        // 响应列表单击事件
        listView.setOnItemClickListener(new OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> av, View v, int position,
                    long id) {

                switch (position) {
                // 设置是否开启提醒
                case 0:
                    ctv1 = (CheckedTextView) v;
                    if (ctv1.isChecked()) {
                        on_off = 0;
                    } else {
```

```
                    on_off = 1;
                }
                break;
            // 设置提醒日期
            case 1:
                showDialog(DATE_DIALOG_ID);
                break;
            // 设置提醒时间
            case 2:
                showDialog(TIME_DIALOG_ID);
                break;
            // 设置提醒内容
            case 3:
                showDialog1("请输入内容: ");
                break;
            // 设置是否开启语音提醒
            case 4:
                ctv2 = (CheckedTextView) v;
                if (ctv2.isChecked()) {
                    alarm = 0;
                    setAlarm(false);
                } else {
                    alarm = 1;
                    setAlarm(true);
                }
                break;
            default:
                break;
            }
        }
    });
}

@Override
protected void onResume() {
    super.onResume();
    // 初始化列表
    init(getIntent());
}
// ListView Adatper，该类实现了列表的每一项通过自定义视图实现
class ViewAdapter extends BaseAdapter {
    // 列表显示内容
    String[] strs = { "是否开启", "日期", "时间", "内容", "开启闹钟" };
    // 获得列表列数
    @Override
    public int getCount() {
        return strs.length;
    }
    // 返回列表项
    @Override
    public Object getItem(int position) {
        return position;
```

```
    }
    // 返回列表 ID
    @Override
    public long getItemId(int position) {
        return position;
    }
    // 获得当前列表项视图
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = li.inflate(R.layout.item_row, null);
        switch (position) {
        // 是否开启该条备忘录
        case 0:
            ctv1 = (CheckedTextView) li
                    .inflate(
                            android.R.layout.simple_list_item_multiple_choice,
                            null);
            ctv1.setText(strs[position]);
            if (on_off == 0) {
                ctv1.setChecked(false);
            } else {
                ctv1.setChecked(true);
            }
            return ctv1;
        // 提醒日期
        case 1:
            dateName = (TextView) v.findViewById(R.id.name);
            dateDesc = (TextView) v.findViewById(R.id.desc);
            dateName.setText(strs[position]);
            dateDesc.setText(mYear + "/" + mMonth + "/" + mDay);
            return v;
        // 提醒时间
        case 2:
            timeName = (TextView) v.findViewById(R.id.name);
            timeDesc = (TextView) v.findViewById(R.id.desc);
            timeName.setText(strs[position]);
            timeDesc.setText(mHour + ":" + mMinute);
            return v;
        // 提醒内容
        case 3:
            contentName = (TextView) v.findViewById(R.id.name);
            contentDesc = (TextView) v.findViewById(R.id.desc);
            contentName.setText(strs[position]);
            contentDesc.setText(content);
            return v;
        // 是否声音提示
        case 4:
            ctv2 = (CheckedTextView) li
                    .inflate(
                            android.R.layout.simple_list_item_multiple_choice,
                            null);
            ctv2.setText(strs[position]);
```

```
            if (alarm == 0) {
                ctv2.setChecked(false);
            } else {
                ctv2.setChecked(true);
            }
            return ctv2;
        default:
            break;
        }

        return null;
    }
}

// 显示对话框
@Override
protected Dialog onCreateDialog(int id) {
    switch (id) {
    // 显示日期对话框
    case DATE_DIALOG_ID:
        return new DatePickerDialog(this, mDateSetListener, mYear, mMonth,
                mDay);
    // 显示时间对话框
    case TIME_DIALOG_ID:
        return new TimePickerDialog(this, mTimeSetListener, mHour, mMinute,
                false);
    }
    return null;
}

// 设置通知提示
private void setAlarm(boolean flag) {
    final String BC_ACTION = "com.amaker.ch17.TaskReceiver";
    // 获得 AlarmManager 实例
    final AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
    // 实例化 Intent
    Intent intent = new Intent();
    // 设置 Intent action 属性
    intent.setAction(BC_ACTION);
    intent.putExtra("msg", content);
    // 实例化 PendingIntent
    final PendingIntent pi = PendingIntent.getBroadcast(
            getApplicationContext(), 0, intent, 0);
    // 获得系统时间
    final long time1 = System.currentTimeMillis();
    Calendar c = Calendar.getInstance();
    c.set(mYear, mMonth, mDay, mHour, mMinute);
    long time2 = c.getTimeInMillis();
    if (flag&&(time2-time1)>0&&on_off==1){
        am.set(AlarmManager.RTC_WAKEUP, time2, pi);
    }else{
```

```
            am.cancel(pi);
        }
    }

    /*
     * 设置提示日期对话框
     */
    private void showDialog1(String msg) {
        View v = li.inflate(R.layout.item_content, null);
        final EditText contentET = (EditText) v.findViewById(R.id.content);
        contentET.setText(content);
        new AlertDialog.Builder(this).setView(v).setMessage(msg).setCancelable(
                false).setPositiveButton("确定",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        content = contentET.getText().toString();
                        contentDesc.setText(content);
                    }
                }).show();
    }
    // 时间选择对话框
    private TimePickerDialog.OnTimeSetListener mTimeSetListener =
        new TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            mHour = hourOfDay;
            mMinute = minute;
            timeDesc.setText(mHour + ":" + mMinute);
        }
    };
    // 日期选择对话框
    private DatePickerDialog.OnDateSetListener mDateSetListener =
        new DatePickerDialog.OnDateSetListener() {
        public void onDateSet(DatePicker view, int year, int monthOfYear,
                int dayOfMonth) {
            mYear = year;
            mMonth = monthOfYear;
            mDay = dayOfMonth;
            dateDesc.setText(mYear + "/" + mMonth + "/" + mDay);
        }
    };
    // 保存或修改备忘录信息
    protected void onPause() {
        super.onPause();
        saveOrUpdate();
    };

    // 保存或修改备忘录信息
    private void saveOrUpdate() {
        ContentValues values = new ContentValues();

        values.clear();
```

```
        values.put(Tasks.CONTENT, contentDesc.getText().toString());
        values.put(Tasks.DATE1, dateDesc.getText().toString());
        values.put(Tasks.TIME1, timeDesc.getText().toString());

        values.put(Tasks.ON_OFF, ctv1.isChecked() ? 1 : 0);
        values.put(Tasks.ALARM, ctv2.isChecked() ? 1 : 0);

        // 修改
        if (id1 != 0) {
            Uri uri = ContentUris.withAppendedId(Tasks.CONTENT_URI, id1);
            getContentResolver().update(uri, values, null, null);
        // 保存
        } else {
            Uri uri = TaskList.Tasks.CONTENT_URI;
            getContentResolver().insert(uri, values);
        }

    }
}
```

程序运行结果如图 17.2 和图 17.3 所示。

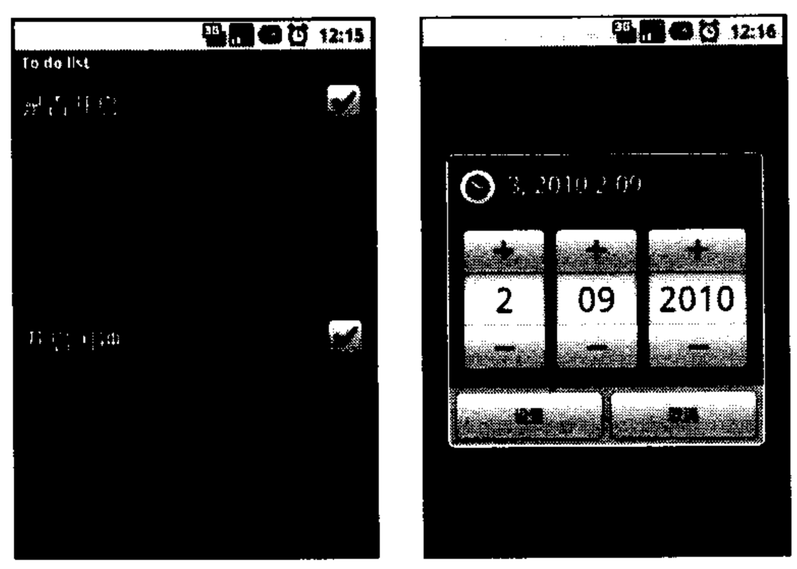


图 17.2　备忘录维护

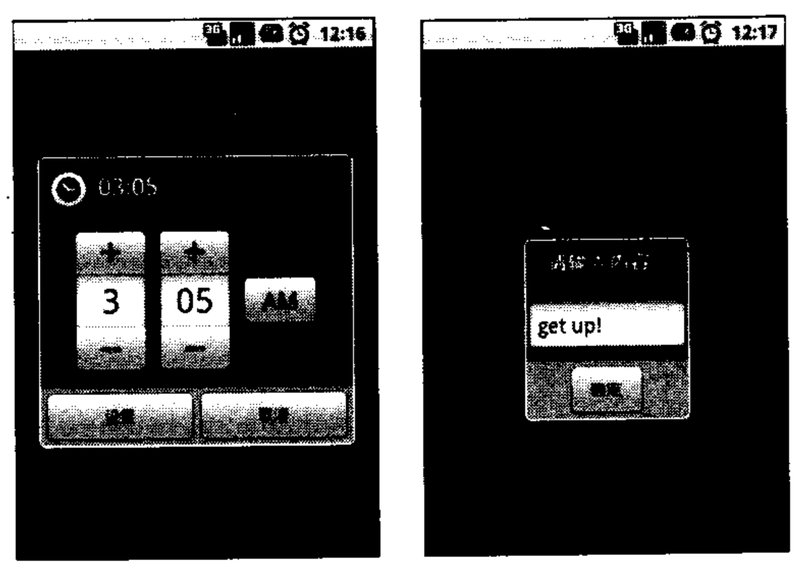


图 17.3　备忘录维护

17.4 备忘录定时提醒

为了更好地提醒用户在特定时间做某项工作，设置语音或振动提醒是必不可少的，本节将实现此功能。该功能通过定义 Activity 发出广播，BroadcastReceiver 接收广播，并通过 Notification 提醒用户。

17.4.1　定义提醒广播接收器

当用户选择了备忘录信息中的“开启闹钟”选项时，便通过广播的方式启动一个定时闹钟，之后定义一个广播接收器接收广播。

```
        // 设置通知提示
```

```
private void setAlarm(boolean flag) {
    final String BC_ACTION = "com.amaker.ch17.TaskReceiver";
    // 获得 AlarmManager 实例
    final AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
    // 实例化 Intent
    Intent intent = new Intent();
    // 设置 Intent action 属性
    intent.setAction(BC_ACTION);
    intent.putExtra("msg", content);
    // 实例化 PendingIntent
    final PendingIntent pi = PendingIntent.getBroadcast(
            getApplicationContext(), 0, intent, 0);
    // 获得系统时间
    final long time1 = System.currentTimeMillis();
    Calendar c = Calendar.getInstance();
    c.set(mYear, mMonth, mDay, mHour, mMinute);
    long time2 = c.getTimeInMillis();
    if (flag&&(time2-time1)>0&&on_off==1){
        am.set(AlarmManager.RTC_WAKEUP, time2, pi);
    }else{
        am.cancel(pi);
    }
}
```

广播接收器代码如下所示：

```
package com.amaker.ch17.app;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

/**
 * @author 郭宏志
 * 定时提醒广播
 */
public class TaskReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {

        intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

        intent.setClass(context, AlarmActivity.class);

        context.startActivity(intent);

    }
}
```

17.4.2　显示提醒 Activity

如上代码所示，接收器收到广播后启动一个 Activity 来显示提醒信息。在 Activity 中

通过 Notification 来提醒用户。在 Activity 中添加一个 TextView 来显示提醒内容，添加一个 Button 来取消提醒。

```
package com.amaker.ch17.app;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

/**
 * @author 郭宏志
 * 提醒用户
 */
public class AlarmActivity extends Activity {
   public static final int ID = 1;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.alarm);
      // 获得 Button、TextView 实例
      Button btn = (Button) findViewById(R.id.cancelButton01);
      TextView tv = (TextView)findViewById(R.id.msgTextView01);

      // 获得 NotificationManager 实例
      String service = Context.NOTIFICATION_SERVICE;
      final   NotificationManager   nm   =   (NotificationManager)getSystemService
(service);
      // 实例化 Notification
      Notification n = new Notification();
      // 设置显示提示信息，该信息也会在状态栏显示
      String msg = getIntent().getStringExtra("msg");
      // 显示时间
      n.tickerText = msg;
      tv.setText(msg);

      // 设置声音提示
      n.sound = Uri.parse("file:///sdcard/fallbackring.ogg");
      // 发出通知
      nm.notify(ID, n);
      // 取消通知
      btn.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View v) {
```

```
                nm.cancel(ID);
                finish();
            }
        });
    }
}
```

AndroidManifest.xml 清单文件代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.amaker.ch17.app"
     android:versionCode="1"
     android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">

     <provider android:name="TaskListProvider"
     android:authorities="com.amaker.provider.TaskList"/>

     <activity android:name=".TaskListActivity"
             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="TaskDetailActivity">
        <intent-filter>
           <action android:name="android.intent.action.NEW" />
           <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
     </activity>


     <receiver android:name="TaskReceiver">
     <intent-filter>
        <action android:name="com.amaker.ch17.TaskReceiver"/>
        <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
     </receiver>

     <activity android:name="AlarmActivity">
      <intent-filter>
        <action android:name="com.amaker.ch17.AlarmAction"/>
        <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>
     </activity>


  </application>
```

```
    <uses-sdk android:minSdkVersion="3" />

<uses-permission android:name="android.permission.VIBRATE"></uses-permission>
</manifest> }
```

程序运行结果如图 17.4 所示。

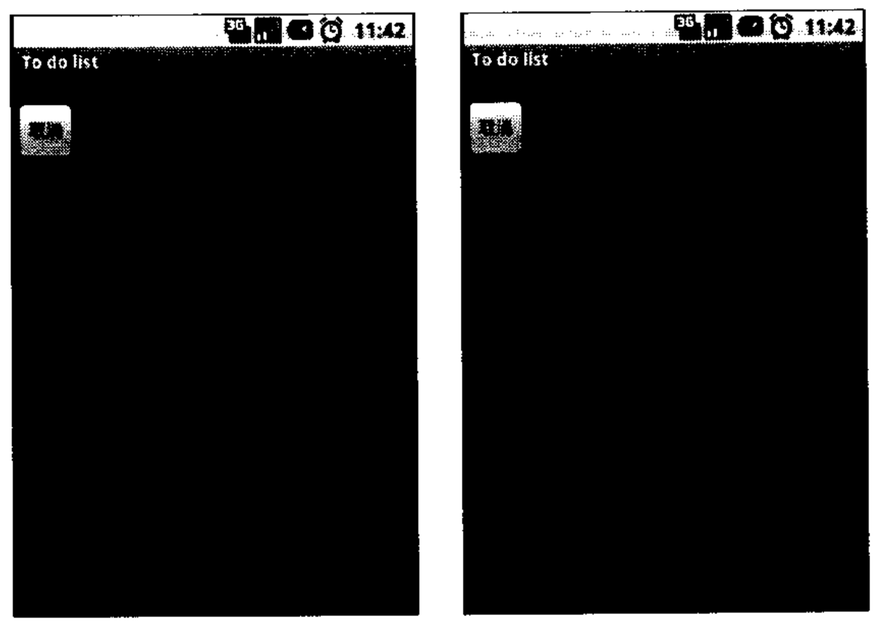
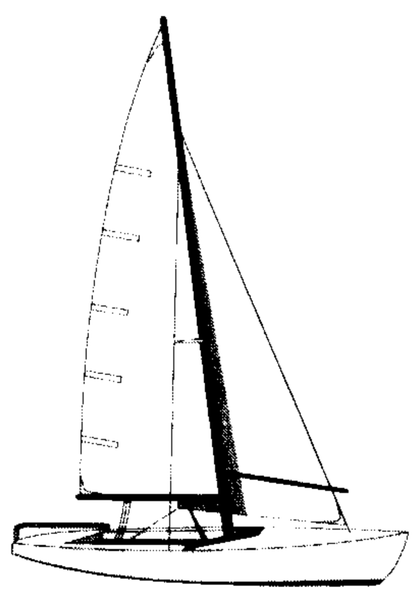


图 17.4　备忘录定时提醒



第 18 章　Android 应用案例——无线点餐系统

从 Android 的从业角度分析，Android 的开发概括为两类：一是系统开发，包括底层 Linux 内核的裁剪和扩展、硬件驱动的开发和系统移植等，从业人员主要工作在硬件厂商的公司里；二是应用开发，主要包括游戏开发、Android 互联网客户端开发和工具软件开发等。业内人士预言，Android 系统将成为手机操作系统领域的“Windows”。所有 Android 互联网客户端应用需求将非常大。本案例就是一个真实的移动互联网应用案例。

本系统将从需求分析、总体设计、详细设计和系统编码等方面，详细介绍无线点餐系统的应用、功能及具体实现。

18.1 无线点餐系统需求分析

传统的餐饮行业，点餐过程都是由餐厅服务员人工完成的。通常的过程是这样的，客人进入餐厅后，选择好桌位开始点餐，服务员等待客人点餐完成，将客人的点餐内容送到厨房，由厨师下厨做菜。这一过程对于一个很小的餐馆来说没有问题，但是如果餐馆比较大或客人比较多，客人的等待时间就会比较长。为了解决这个问题，今年出现的无线点餐系统，该系统一般由无线终端（一般是一个 PDA）、无线路由器和后台服务器组成。

这种技术架构很成熟，但是存在一个弊端就是 PDA 的价格较高，并且 PDA 只能作为点餐使用，这无形中增加了系统成本，也造成了资源的浪费。有了 Android 系统，我们就可以在其中开发一套客户端软件来完成点餐功能，Android 手机的其他功能（如打电话、发短信）照常使用。这样首先系统的成本会降低，其次，Android 系统功能强大，界面组件丰富，软件的功能自然会强大很多，用户操行更流畅，界面展现更完美。

18.2 无线点餐系统概要设计

本项目主要面向具有一定规模、需要提高餐厅点餐效率、体现出现代化的气氛、改进

客人就餐环境，并借此提高经济效益的餐厅和酒店。要求项目不会占用太大的预算，并能够尽量沿用原有的成熟技术，与原有系统进行良好的整合。

本方案主要包括后台数据库服务器、Web 服务器、无线网络、Android 点餐前端等部分组成。为了满足系统的需求，我们将从系统架构、技术选型、系统功能几个方面详细介绍该系统。

18.2.1 系统物理架构

系统的物理架构是这样的：客户端 Android 系统智能手机通过无线网络访问后台服务器，如果需要数据访问，则访问后台数据库。

系统物理结构如图 18.1 所示。

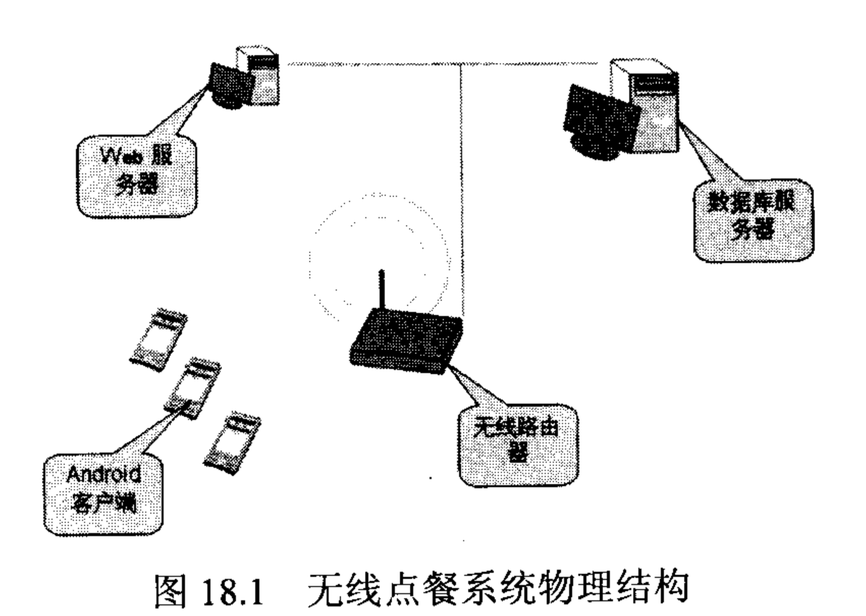


图 18.1 无线点餐系统物理结构

18.2.2 技术选型

Android 客户端应用使用 Java 技术，网络通信使用 Apache HTTP 协议客户端，中间 Web 服务器采用 Servlet 响应客户请求。

后台数据库采用 JDBC 访问数据库，客户端的一部分数据存储我们采用 SQLite 数据库，这里要涉及 Android 客户端的数据和服务器中的数据同步问题。

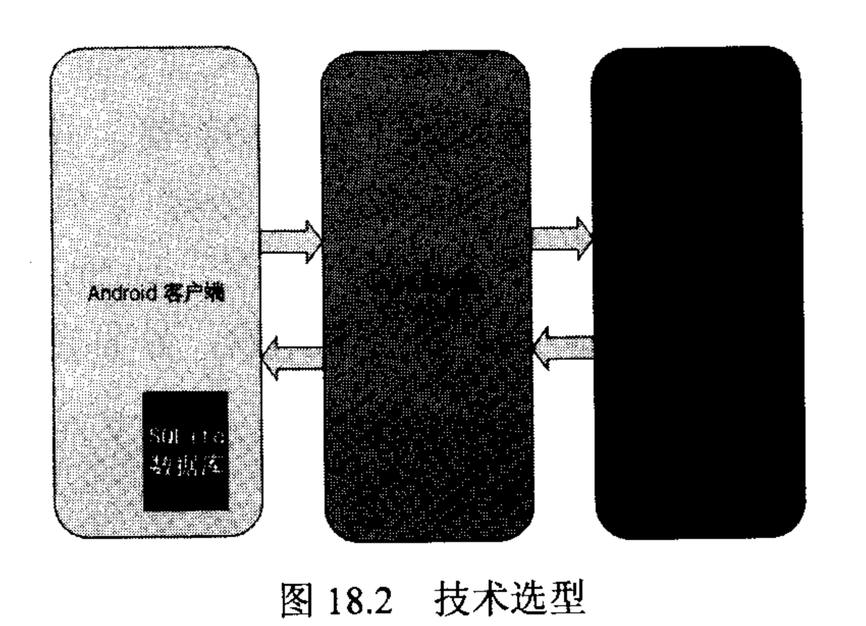


图 18.2 技术选型

18.2.3　系统功能

通过对系统进行需求分析，我们了解了系统的需求。通过系统物理架构和技术选择，我们确定了项目的可行性。接下来我们将对系统功能进行介绍。

本系统功能说明如下：

1．系统的登录功能

为了增强系统的使用安全，使用系统之前必须登录系统，登录过程是通过无线网络，在后台的数据库通过用户名称和密码进行查询。注册用户方可使用该系统。系统登录界面如图 18.3 所示。

2．系统主菜单

系统登录成功之后便进入了系统主菜单，主菜单通过一个 GridView 以图形菜单的方式来展现系统的功能，单击某个选项便进入该功能的操作界面。运行界面如图 18.4 所示。

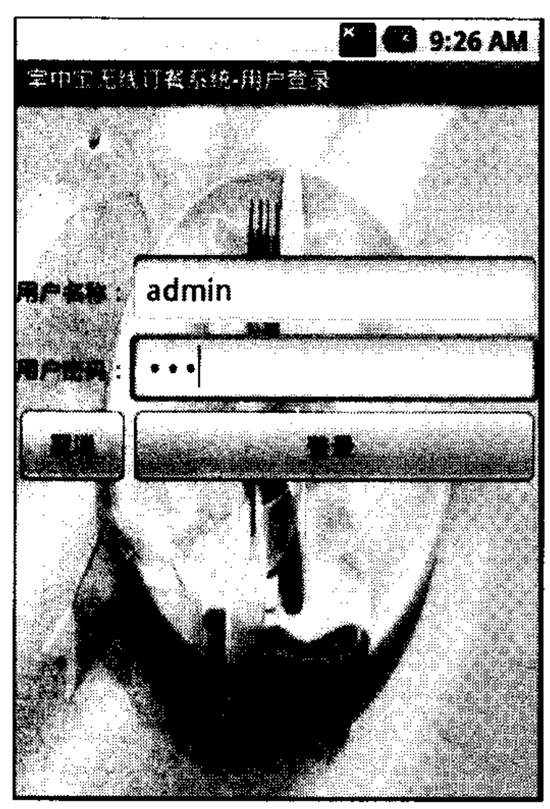


图 18.3　系统登录界面

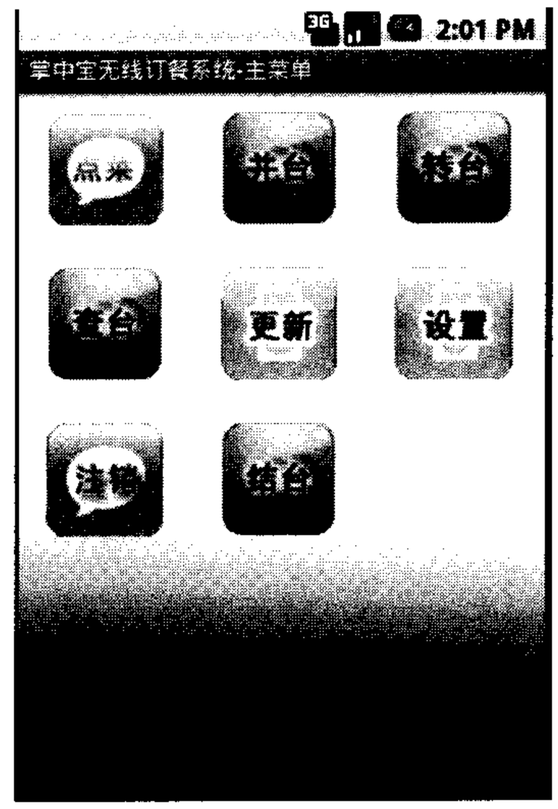


图 18.4　系统主菜单

3．点餐功能

点餐功能模块是本系统中一个重要的功能模块，该模块要完成的任务是操作员输入顾客的点餐信息，通过无线网络及时地将点餐信息传送到后台服务器，进而在厨房终端显示，有利于厨师尽快下厨做菜。

该功能包括了三部分：开桌、点菜和下单。运行界面如图 18.5 所示。

4．结算功能

顾客就餐结束需要结算，结算的过程是：操作员根据订单编号查询点餐订单信息和订单信息详细列表，顾客确认后单击结算按钮进行结算。运行界面如 18.6 所示。

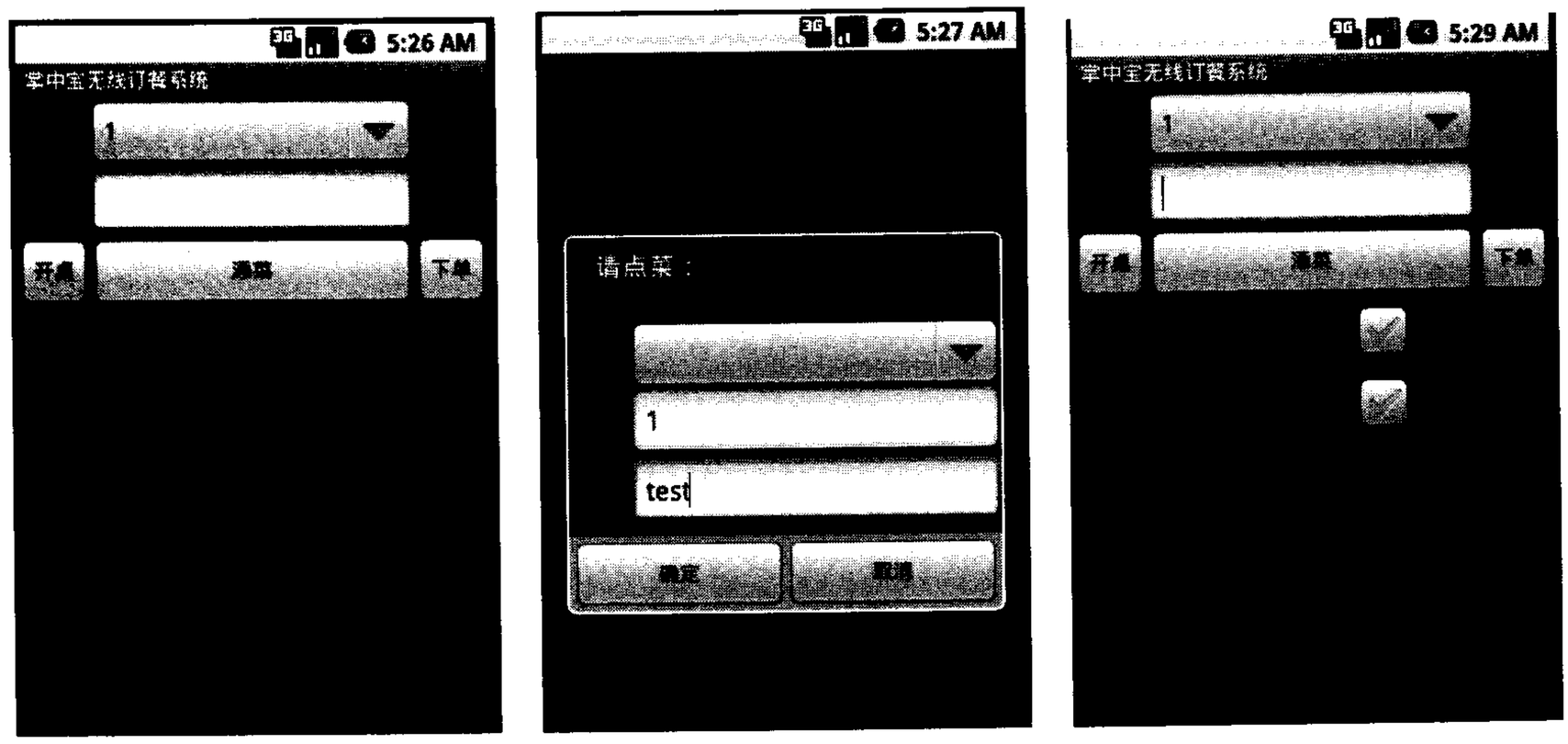


图 18.5　点餐运行界面

5. 查台功能

当有顾客进入餐馆时，需要服务员查询餐桌状态（有人或空位），来安排顾客就餐。这一过程是通过查询数据库中的餐桌数据得到当前餐桌状态列表，通过可视化的界面来进行显示。程序运行界面如图 18.7 所示。

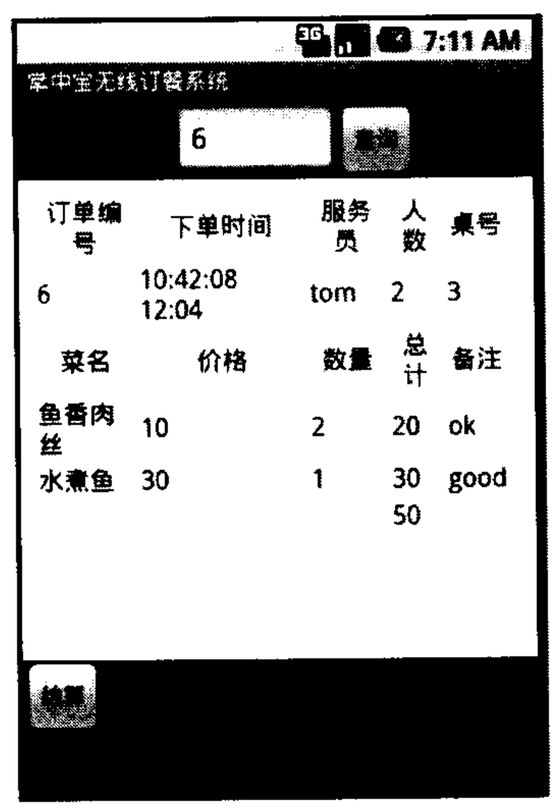


图 18.6　结算功能模块运行界面

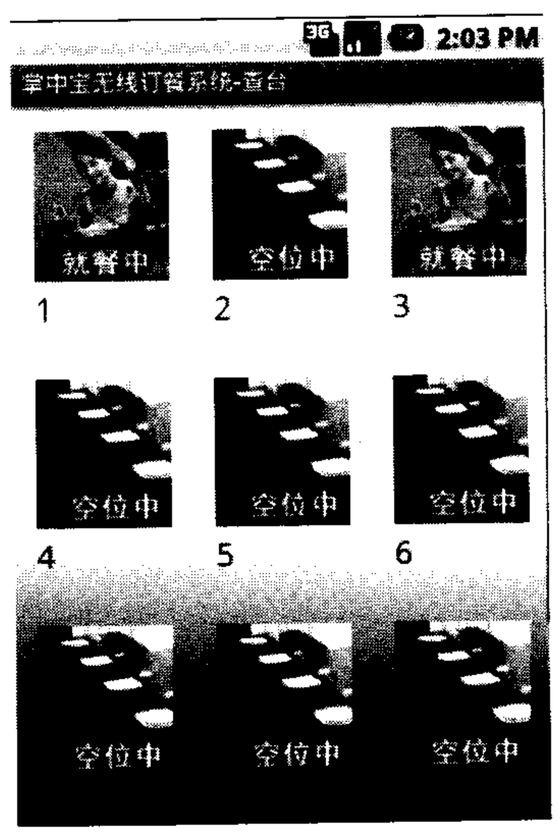


图 18.7　查台功能模块运行结果

6. 更新功能

为了提高程序的运行效率，将服务器中菜谱表（MenuTbl）和餐桌表（TableTbl）中的数据保存到客户端 SQLite 数据库中。因此，当服务器中的数据改变时需要及时将服务器中的数据动态地更新到客户端，更新功能就是为此而设计的。程序运行界面如图 18.8 所示。

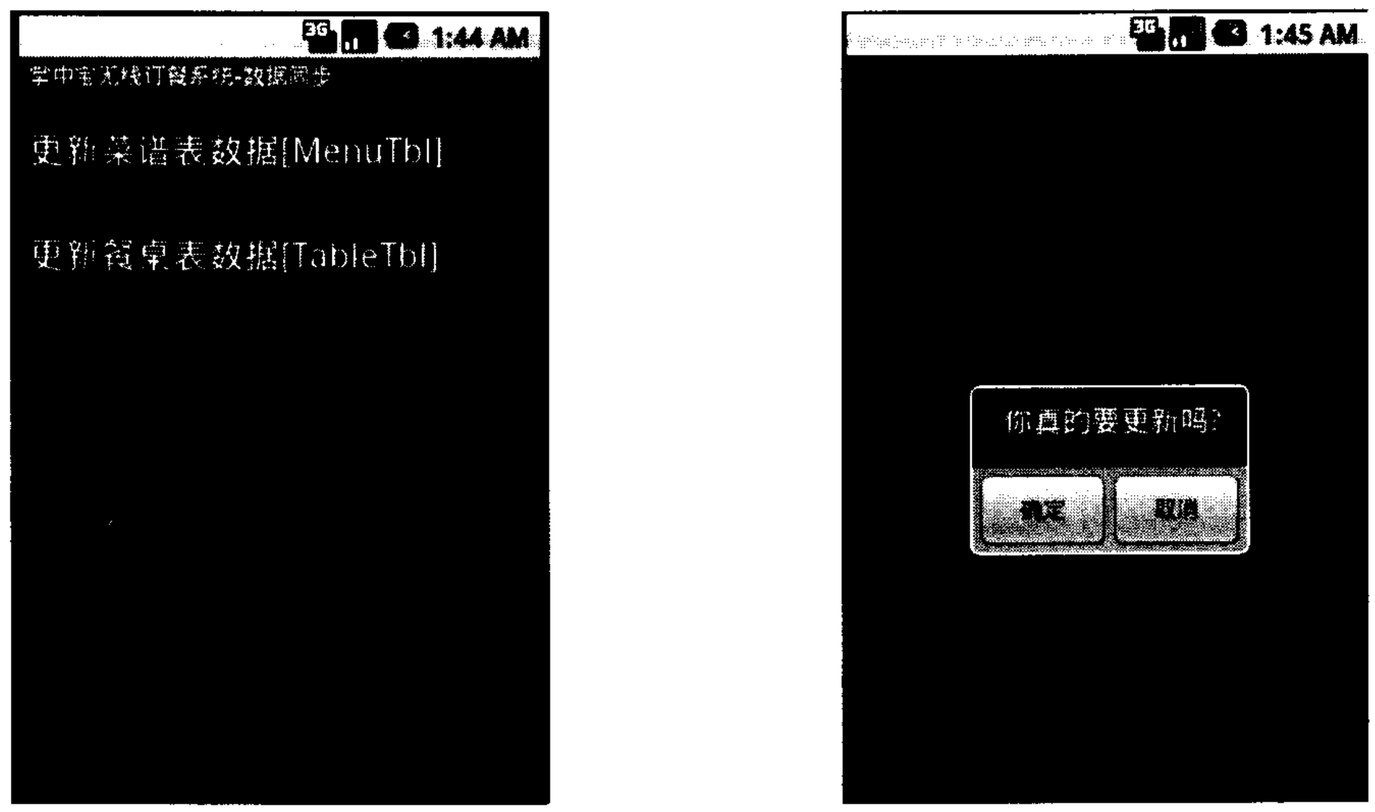


图 18.8　更新功能模块运行结果

7. 转台功能

转台功能是指顾客下单后，要求更换座位，这时需要更改已经提交的数据。更改的过程是根据订单号找到该订单对应的桌号，将该桌号更改为更换后的桌号。程序运行界面如图 18.9 所示。

8. 并台功能

并台功能是将已经下单的两组顾客合并到一张桌子的过程，该过程也需要更新已经提交到数据库中的数据。更新内容包括人数的合并、订单的更新、订单详细的更新和桌位状态的更新。程序运行界面如图 18.10 所示。

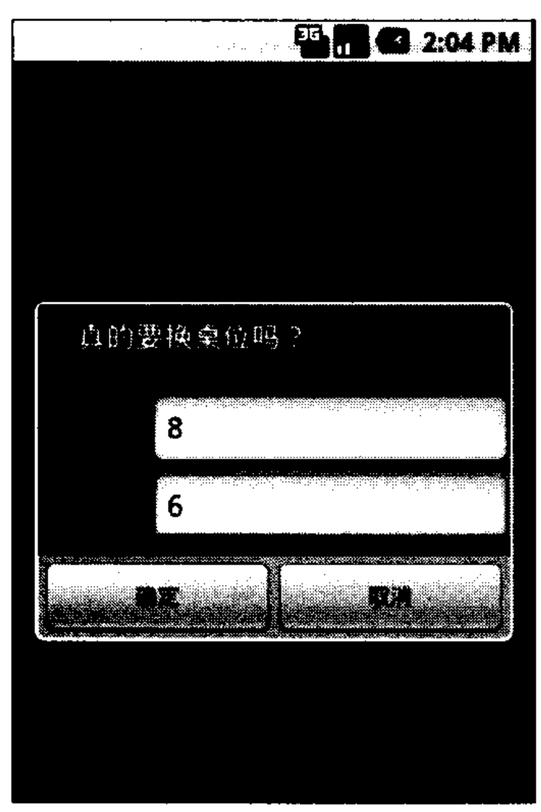


图 18.9　转台功能模块运行结果

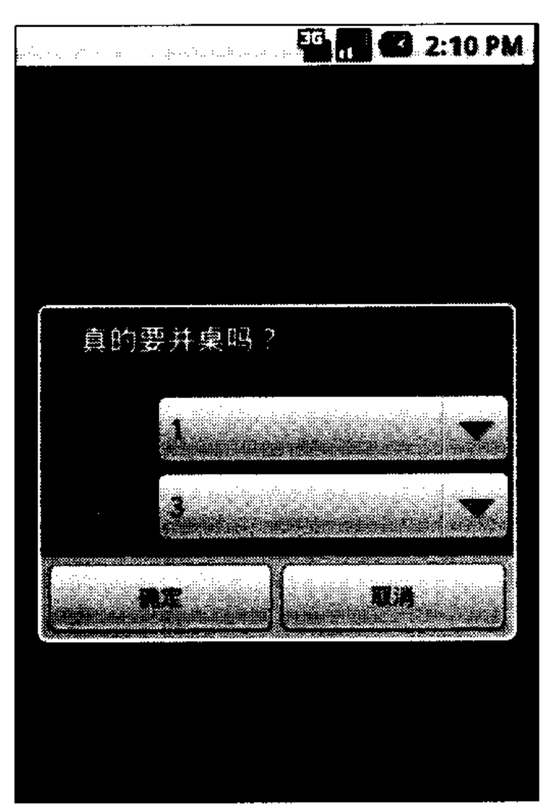


图 18.10　并台功能模块运行结果

18.3 无线点餐系统详细设计

确定了系统功能之后，我们就可以进行系统的详细设计了。详细设计里主要包括：系统包及其资源规划和系统数据库设计。

18.3.1 系统包及其资源规划

本系统的主要功能可以分为三大类：一是访问客户端 SQLite 数据库；二是 Activity 界面展示；三是通过网络访问服务器数据。客户端的包结构如表 18.1 所示。

表 18.1 系统包说明

包 名 称	包 描 述
com.amaker.provider	访问本地 SQLite 数据库的 Provider
com.amaker.util	工具类包
com.amaker.wlo	Activity 包

包及其资源结构如图 18.11 所示。

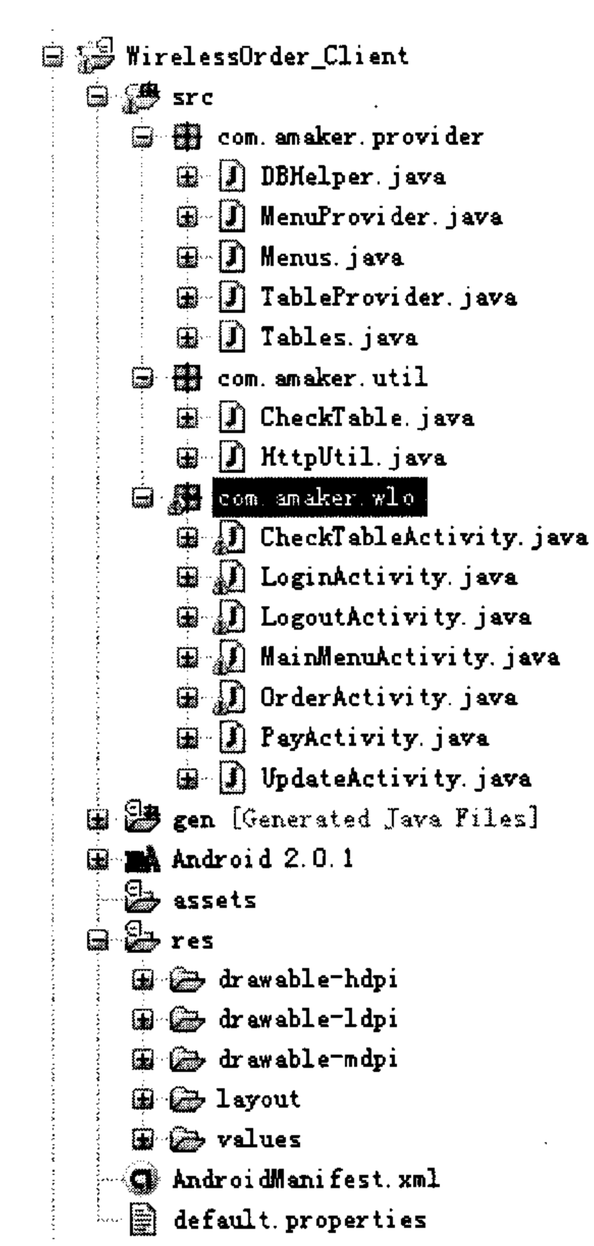


图 18.11 系统包及其资源结构

18.3.2　系统数据库设计

程序无非就是：输入、处理和输出。本系统也不例外，除了 Android 客户端应用程序和中间无线网络，还要有强大的数据库做支持。本系统 Android 客户端采用 SQLite 数据库，服务器端采用 MySQL 数据库，并提供了客户端数据和服务器端数据的同步功能，提高了程序的运行效率。本节将对系统使用到的表结构进行详细讲解。

1．系统用户表（UserTbl），登录系统使用。表结构如图 18.12 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
account	VARCHAR	20	0	☐	Null	登录账号
password	VARCHAR	20	0	☐	Null	登录密码
name	VARCHAR	20	0	☐	Null	姓名
gender	VARCHAR	20	0	☐	Null	性别
permission	INTEGER	11	0	☐	Null	权限 1：管理员 2：收银员 3：服务员。
remark	VARCHAR	200	0	☐	Null	备注

图 18.12　系统用户表结构

2．菜谱分类表（MenuTypeTbl），为了方便点餐，将菜单信息进行分类，如热菜、凉菜等。表结构如图 18.13 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
name	VARCHAR	20	0	☐	Null	类型名称

图 18.13　菜谱分类表结构

3．餐桌表（TableTbl），用来表示顾客在哪个桌位就坐。表结构如图 18.14 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
num	INTEGER	11	0	☐	Null	桌号
flag	INTEGER	11	0	☐	0	餐桌状态 0：空桌 1：有人就餐
description	VARCHAR	100	0	☐	Null	描述

图 18.14　餐桌表

4．菜谱表（MenuTbl），菜谱表用来保存菜信息，如分类、名称和价格等。表结构如图 18.15 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
typeID	INTEGER	11	0	☐	Null	外键，引用菜单类型表ID
name	VARCHAR	50	0	☐	Null	名称
price	INTEGER	11	0	☐	Null	价格
pic	VARCHAR	100	0	☐	Null	图片路径
remark	VARCHAR	200	0	☐	Null	备注

图 18.15　菜谱表结构

5．订单表（OrderTbl），当顾客开桌点菜时生成的信息，保存到该表。表结构如图 18.16 所示。

Field Name	Field Type	Size	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
orderTime	VARCHAR	30	0	☐	Null	下单时间
userID	INTEGER	11	0	☐	Null	下单用户
tableId	INTEGER	11	0	☐	Null	桌号，外键引用桌号Id
personNum	INTEGER	11	0	☐	Null	人数
isPay	INTEGER	11	0	☐	0	是否结算,"0"：未结算，"1"：结算
remark	VARCHAR	200	0	☐	Null	备注

图 18.16　订单表结构

6．订单明细表（OrderDetailTbl），订单明细表和上面的订单表是一对多的关系，即一桌客人点了多份菜。表结构如图 18.17 所示。

Field Name	Field Type	Size △	Precision	Not Null	Default	Comment
id	INTEGER	11	0	☑	Null	主键，自动增加
orderId	INTEGER	11	0	☐	Null	外键，参照订单Id
menuId	INTEGER	11	0	☐	Null	外键，参照菜谱Id
num	INTEGER	11	0	☐	Null	数量
remark	VARCHAR	200	0	☐	Null	备注

图 18.17　订单明细表结构

18.4 系统编码实现

有了以上的准备工作，接下来我们要开始系统编码实现了。本系统程序分为两个部分，前端 Android 实现和后端服务器实现。我们主要讲解前端 Android 实现，至于后端服务器实现只做简单讲解。对 Java 服务器端编程不是很熟悉的读者，请参阅 Java EE 的相关书籍。

18.4.1　登录模块 Android 客户端实现

首先，为了提高系统的安全性，要想使用该系统必须进行系统登录。其次，每个点餐信息中都要用到登录用户信息，这样可以做到责任分明、工作严谨。登录的基本流程是：通过用户输入的用户名称和用户密码，到后台数据库做查询，如果有该用户则进入系统，否则提示用户输入错误。

为了完成系统登录功能，需要 Android 客户端程序和后台服务器程序。本节讲述的是有关 Android 客户端的设计实现。

要完成登录模块的 Android 客户端实现，需要如下步骤。

① 创建一个 Android 工程，名称为“WirelessOrder_Client”。

② 在工程的 res\layout 目录下，创建一个名称为“login_system.xml”的布局文件，布局文件外层使用 LinearLayout 布局，内层嵌套 TableLayout 完成布局管理。该布局文件中引用一个图片资源来显示背景图片、两个 TextView 用来显示用户名称和密码、两个 EditText 用来输入用户名称和密码、两个 Button 用来登录系统和取消登录，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"
   android:background="@drawable/login"
   >

   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">

      <TableRow>
         <TextView
         android:text="用户名称："
         android:id="@+id/TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#0000ff"
         />

         <EditText
         android:text=""
         android:id="@+id/userEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>


      <TableRow>
         <TextView
         android:text="用户密码："
         android:id="@+id/TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:textColor="#0000ff"
         />

         <EditText
         android:text=""
         android:id="@+id/pwdEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:password="true"/>

      </TableRow>

      <TableRow
         android:gravity="right">

         <Button
```

```
            android:text="取消"
            android:id="@+id/cancelButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <Button
            android:text="登录"
            android:id="@+id/loginButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

    </TableRow>

  </TableLayout>
</LinearLayout>
```

③ 创建一个名称为“LoginActivity”的 Activity，在类的顶部声明用到的 EditText 和 Button 组件，在 onCreate()方法中调用 setContentView(R.layout.login_system)方法设置布局视图。通过 findViewById()方法实例化 EditText 和 Button 对象。程序代码如下所示。

```
package com.amaker.wlo;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.amaker.util.HttpUtil;

public class LoginActivity extends Activity {
   // 声明登录、取消按钮
   private Button cancelBtn,loginBtn;
   // 声明用户名、密码输入框
   private EditText userEditText,pwdEditText;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      // 设置标题
      setTitle("掌中宝无线订餐系统-用户登录");
      // 设置当前 Activity 界面布局
      setContentView(R.layout.login_system);
      // 通过 findViewById 方法实例化组件
      cancelBtn = (Button)findViewById(R.id.cancelButton);
      // 通过 findViewById 方法实例化组件
      loginBtn = (Button)findViewById(R.id.loginButton);
```

```
        // 通过 findViewById 方法实例化组件
        userEditText = (EditText)findViewById(R.id.userEditText);
        // 通过 findViewById 方法实例化组件
        pwdEditText = (EditText)findViewById(R.id.pwdEditText);
        }
}
```

④ 为了实现登录功能，Android 需要通过网络访问后台的数据库进行查询。Android 访问网络的实现方法有很多，如 Web Service、HTTP、直接使用 WebView 浏览器。Android 集成了 Apache 的 Http Client 模块，这样使得使用 HTTP 访问网络成为可能。在这里我们使用 HTTP 访问网络。HTTP 是请求（request）/响应（response）的一种机制，这里我们定义了一个工具类 HttpUtil 来获得 HttpRequest 对象和 HttpResponse 对象，以及发送 get 和 post 请求获得返回 response 信息的方法。

```
package com.amaker.util;

import java.io.IOException;

import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class HttpUtil {
   // 声明 Base URL 常量
   public static final String BASE_URL="http://192.168.1.101:8888/ WirelessOrder_
Server/";

   // 通过 url 获得 HttpGet 对象
   public static HttpGet getHttpGet(String url){
       // 实例化 HttpGet
       HttpGet request = new HttpGet(url);
       return request;
   }
   // 通过 URL 获得 HttpPost 对象
   public static HttpPost getHttpPost(String url){
        // 实例化 HttpPost
        HttpPost request = new HttpPost(url);
        return request;
   }
   // 通过 HttpGet 获得 HttpResponse 对象
   public static HttpResponse getHttpResponse(HttpGet request) throws ClientProtocol
Exception, IOException{
       // 实例化 HttpResponse
       HttpResponse response = new DefaultHttpClient().execute(request);
       return response;
   }
   // 通过 HttpPost 获得 HttpResponse 对象
```

```
    public static HttpResponse getHttpResponse(HttpPost request) throws ClientProtocol
Exception, IOException{
        // 实例化 HttpResponse
        HttpResponse response = new DefaultHttpClient().execute(request);
        return response;
    }


    // 通过 url 发送 post 请求，返回请求结果
    public static String queryStringForPost(String url){
        // 获得 HttpPost 实例
        HttpPost request = HttpUtil.getHttpPost(url);
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得返回结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        } catch (IOException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        }
        return null;
    }

    // 通过 HttpPost 发送 get 请求，返回请求结果
    public static  String queryStringForGet(String url){
        // 获得 HttpGet 实例
        HttpGet request = HttpUtil.getHttpGet(url);
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得返回结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
```

```
        } catch (IOException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        }
        return null;
    }

    // 通过 HttpPost 发送 post 请求，返回请求结果
    public static String queryStringForPost(HttpPost request){
        String result = null;
        try {
            // 获得 HttpResponse 实例
            HttpResponse response = HttpUtil.getHttpResponse(request);
            // 判断是否请求成功
            if(response.getStatusLine().getStatusCode()==200){
                // 获得请求结果
                result = EntityUtils.toString(response.getEntity());
                return result;
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        } catch (IOException e) {
            e.printStackTrace();
            result = "网络异常！";
            return result;
        }
        return null;
    }
}
```

⑤ 定义一个显示提示信息的对话框方法，用于显示提示信息。

```
// 定义一个显示提示信息的对话框
private void showDialog(String msg){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage(msg)
          .setCancelable(false)
          .setPositiveButton("确定", new DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int id) {
              }
          });
    AlertDialog alert = builder.create();
    alert.show();
}
```

⑥ 定义 validate()方法，对用户名称和密码进行非空验证。

```
// 对用户名称和密码非空进行验证
private boolean validate(){
    String username = userEditText.getText().toString();
```

```
        if(username.equals("")){
            showDialog("用户名称是必填项！");
            return false;
        }
        String pwd = pwdEditText.getText().toString();
        if(pwd.equals("")){
            showDialog("用户密码是必填项！");
            return false;
        }
        return true;
    }
```

⑦ 定义一个 query()方法，通过用户名称和密码进行查询，发送 Post 请求，获得响应结果。

```
    // 通过用户名称和密码进行查询，发送 Post 请求，获得相应结果。
    private String query(String username,String password){
        // 查询字符串
        String queryString = "username="+username+"&password="+password;
        // 查询 URL
        String url = HttpUtil.BASE_URL+"servlet/LoginServlet?"+queryString;
        // 查询并返回结果
        return HttpUtil.queryStringForPost(url);
    }
```

⑧ 定义 login()方法，该方法调用 query()方法，获得查询结果，并将查询结果通过 SharedPreferences 保存到 XML 配置文件，以便在后面程序的点餐中使用用户信息。login()方法调用 saveUserMsg()方法将用户信息保存到配置文件。

```
    // 将用户信息保存到配置文件
    private void saveUserMsg(String msg){
        // 用户编号
        String id = "";
        // 用户名称
        String name = "";
        // 获得信息数组
        String[] msgs = msg.split(";");
        int idx = msgs[0].indexOf("=");
        id = msgs[0].substring(idx+1);
        idx = msgs[1].indexOf("=");
        name = msgs[1].substring(idx+1);
        // 共享信息
        SharedPreferences pre = getSharedPreferences("user_msg", MODE_WORLD_WRITEABLE);
        SharedPreferences.Editor editor = pre.edit();
        // 保存用户 Id
        editor.putString("id", id);
        // 保存用户名称
        editor.putString("name", name);
        // 提交
        editor.commit();
    }
```

```
    // 登录方法
    private boolean login(){
        // 获得用户名称
        String username = userEditText.getText().toString();
        // 获得密码
        String pwd = pwdEditText.getText().toString();
        // 获得登录结果
        String result=query(username,pwd);

        if(result!=null&&result.equals("0")){
            return false;
        }else{
            saveUserMsg(result);
            return true;
        }
    }
```

⑨ 为登录按钮添加单击监听器，在 onClick()方法中，首先调用 validate()方法进行验证，如果验证通过再调用 login()方法，如果登录成功，系统进入主菜单界面，否则提示登录失败信息。如果验证失败则提示失败信息，请求用户重新输入。

```
        // 为按钮添加单击事件监听器
        loginBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                if(validate()){
                    if(login()){
                        Intent intent = new Intent(LoginActivity.this,MainMenuActivity.
class);
                        startActivity(intent);
                    }else{
                        showDialog("用户名称或者密码错误，请重新输入！");
                    }
                }
            }
        });
```

⑩ 为取消按钮添加单击监听器，在 onClick()方法中调用 finish()方法结束该 Activity。

```
        // 设置取消监听器
        cancelBtn.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // 结束当前 Activity
                finish();
            }
        });
```

程序运行结果如图 18.18 所示。

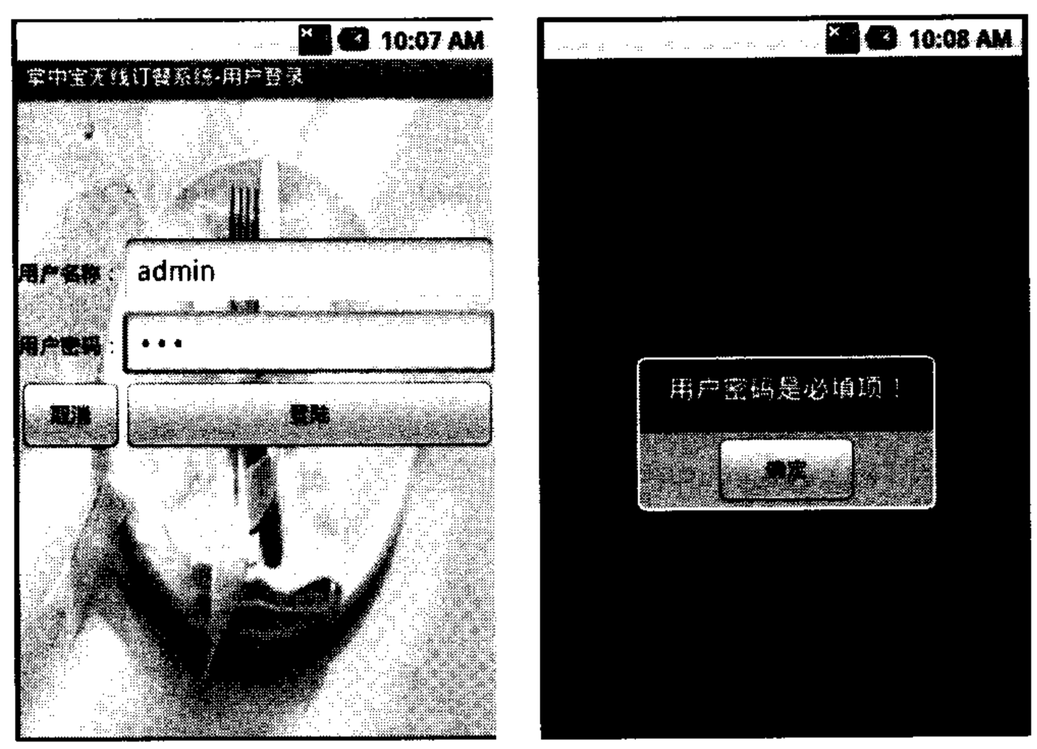


图 18.18　用户登录

18.4.2　登录模块服务器端实现

后台的服务器实现采用 JDBC 访问后台数据库，Servlet 响应 HttpRequest 请求，并返回响应结果。为了完成该功能我们需要安装 MySQL 数据库并配置 Tomcat 服务器，这部分内容将不再赘述。有不明白的读者请参阅 Java EE 的相关书籍。

为了程序更加清晰易懂，我们依然采用 MVC+DAO 的设计模式及分层开发思想。

要完成登录模块的服务器端实现需要如下步骤。

① 创建一个名称为“WirelessOrder_Server”的 Web 工程。

② 创建程序需要的包结构。dao 包用于放置采用 DAO 设计模式实现的访问数据库接口及其实现类；entity 包用于放置对应数据库表的实体类；servlet 包用于放置 Servlet 类；filter 包用于放置过滤器；util 包是工具类。其内容如图 18.19 所示。

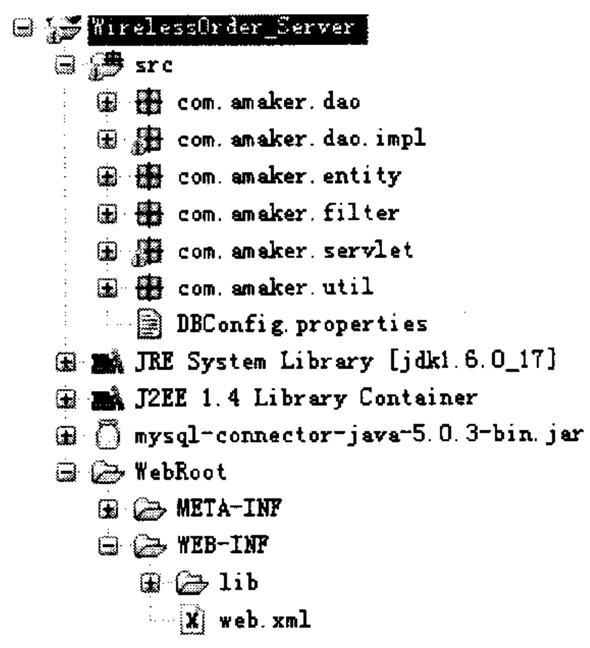


图 18.19　后台系统包结构

③ 因为程序要访问 MySQL 数据库，所以要添加 MySQL 驱动包“mysql-connector-java-5.0.3-bin.jar”。将该 jar 文件添加到工程的 classpath 下。本工程可以直接复制到 lib 目录下即可。

④ 创建访问数据库的工具类 DBUtil，该类的主要功能是打开和关闭数据库连接。为了程序更具可扩展性，我们将配置信息保存到配置文件当中，这样可以在不修改程序的前提下移植数据库。

配置文件 DBConfig.properties 的内容如下所示。

```
// 数据库驱动
driver=com.mysql.jdbc.Driver
// 数据库连接 url
url=jdbc:mysql://localhost:3306/wireless_db?useUnicode=true&characterEncoding=utf
-8
// 用户名称
username=root
// 密码
password=1
```

DBUtil 工具类内容如下所示。

```
package com.amaker.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
/**
 *
 * @author hz.guo
 * 数据库工具类
 */
public class DBUtil {
    // 关闭数据库连接
   public void closeConn(Connection conn){
       try {
          conn.close();
       } catch (SQLException e) {
          e.printStackTrace();
       }
   }

   // 打开数据库连接
   public Connection openConnection() {
       // 实例化 Properties
       Properties prop = new Properties();
       // 驱动
       String driver = null;
       // 数据库连接 url
       String url = null;
       // 用户名称
```

```
        String username = null;
        // 用户密码
        String password = null;

        try {
            // 加载配置文件
            prop.load(this.getClass().getClassLoader().getResourceAsStream(
                    "DBConfig.properties"));
            // 获得数据库驱动
            driver = prop.getProperty("driver");
            // 获得数据库 url
            url = prop.getProperty("url");
            // 获得用户名称
            username = prop.getProperty("username");
            // 获得用户密码
            password = prop.getProperty("password");
            // 注册驱动
            Class.forName(driver);
            // 返回数据库连接
            return DriverManager.getConnection(url, username, password);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

⑤ 创建用于封装 UserTbl 表信息的实体类 User 类。

```
package com.amaker.entity;
/**
 *
 * @author hz.guo
 * 用于封装 UserTbl 表的实体类
 */
public class User {
    // 编号
    private int id;
    // 账号
    private String account;
    // 密码
    private String password;
    // 用户名称
    private String name;
    // 性别
    private String gender;
    // 权限
    private int permission;
    // 备注
    private String remark;
    getters()…
    setters()…
```

⑥ 创建访问数据库的 DAO 接口类 UserDao。

```
package com.amaker.dao;
```

```
import com.amaker.entity.User;
public interface UserDao {
    // 登录方法，返回 User 实例
    public User login(String username,String password);
}
```

⑦ 创建 UserDao 接口实现类 UserDaoImpl。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import com.amaker.entity.User;

import com.amaker.dao.UserDao;
import com.amaker.util.DBUtil;
/**
 *
 * @author 郭宏志
 * 用户登录 DAO 实现类
 */
public class UserDaoImpl implements UserDao {

    // 通过用户名称和密码登录，登录成功返回 User 对象，登录失败返回 null
    public User login(String username, String password) {
        // SQL 查询语句
        String sql = " select id,username,password from UserTbl where username=? and
password=? ";
        // 实例化数据库工具类
        DBUtil util = new DBUtil();
        // 获得数据库连接
        Connection conn = util.openConnection();
        try {
            // 创建预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置查询参数
            pstmt.setString(1, username);
            // 设置查询参数
            pstmt.setString(2, password);
            // 结果集
            ResultSet rs = pstmt.executeQuery();
            // 判断该用户是否存在
            if (rs.next()) {
                // 获得编号
                int id = rs.getInt(1);
                // 实例化 User
                User u = new User();
                // 设置 User 属性
                u.setId(id);
                u.setPassword(password);
```

```
                u.setUsername(username);
                return u;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
        return null;
    }
}
```

⑧ 创建 LoginServlet 类响应请求。该类通过 request.getParameter()方法获得请求参数，调用 UserDaoImpl 的 login()方法，并根据登录结果返回响应结果信息。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.UserDao;
import com.amaker.dao.impl.UserDaoImpl;
import com.amaker.entity.User;
/**
 *
 * @author 郭宏志
 * 响应 Android 客户端发来的请求
 */
public class LoginServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        UserDao dao = new UserDaoImpl();
        // 获得客户端请求参数
        String username = request.getParameter("account");
        String password = request.getParameter("password");

        User u = dao.login(username, password);
        if(u!=null){
            // 响应客户端内容，登录成功
            out.print(build(u));
        }else{
            // 响应客户端内容，登录失败
            out.print("0");
        }
```

```
        out.flush();
        out.close();
    }

    // 将用户对象转换为字符串返回给客户端
    private String build(User u){
        String userMsg = "";
        userMsg+="id="+u.getId();
        userMsg+=";";
        userMsg+="name="+u.getName();
        return userMsg;
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // 调用 doGet 方法
        doGet(request,response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
    // 构造方法
    public LoginServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }

}
```

到此为止，系统登录模块的前后台实现已全部完成。

18.4.3　程序主菜单实现

用户成功登录之后便可以进入系统主菜单界面了。主菜单是通过一个 GridView 来展示的。在 GridView 中的子组件是 ImageView，每一个 ImageView 代表一个功能，当单击 ImageView 时跳转到相应功能界面。

程序主菜单的实现需要如下步骤。

① 在工程的 res\layout\目录下创建一个布局文件，该布局文件外层是 LinearLayout，使用一个图片资源充当背景色，内层是一个 GridView。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@drawable/menu_bg"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<GridView
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:columnWidth="90dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
</LinearLayout>
```

② 创建一个名称为“MainMenuActivity”的 Activity，在 onCreate()方法中设置 Activity 界面标题和布局，实例化 GridView 组件，为 GridView 绑定 Adapter。

```
package com.amaker.wlo;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.Toast;

public class MainMenuActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
       // 设置标题
        setTitle("掌中宝无线订餐系统-主菜单");
       // 设置界面布局
        setContentView(R.layout.main_menu);
       // 获得 GridView 实例
        GridView gridview = (GridView) findViewById(R.id.gridview);
       // 为 GridView 设置 Adapter
        gridview.setAdapter(new ImageAdapter(this));
    }
}
```

③ 定义一个名称为“ImageAdapter”的类，该类继承 BaseAdapter，为 GridView 设置 Adapter。

```
      // 继承 BaseAdapter
    public class ImageAdapter extends BaseAdapter {
    // 上下文
        private Context mContext;
        // 构造方法
        public ImageAdapter(Context c) {
            mContext = c;
        }
        // 组件个数
        public int getCount() {
            return mThumbIds.length;
        }

}
```

```
        // 当前组件
        public Object getItem(int position) {
            return null;
        }
        // 当前组件 id
        public long getItemId(int position) {
            return 0;
        }
        // 获得当前视图
        public View getView(int position, View convertView, ViewGroup parent) {
        // 声明图片视图
            ImageView imageView;
            if (convertView == null) {
                // 实例化图片视图
                imageView = new ImageView(mContext);
                // 设置图片视图属性
                imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
                imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                imageView.setPadding(8, 8, 8, 8);
            } else {
                imageView = (ImageView) convertView;
            }
            // 设置图片视图图片资源
            imageView.setImageResource(mThumbIds[position]);
            // 为当前视图添加监听器
            switch (position) {
            case 0:
                // 添加点餐监听器
                imageView.setOnClickListener(orderLinstener);
                break;
            case 1:
                // 并台监听器
                imageView.setOnClickListener(unionTableLinstener);
                break;
            case 2:
                // 添加转台监听器
                imageView.setOnClickListener(changeTableLinstener);
                break;
            case 3:
                // 添加查台监听器
                imageView.setOnClickListener(checkTableLinstener);
                break;
            case 4:
                // 添加更新监听器
                imageView.setOnClickListener(updateLinstener);
                break;
            case 6:
                // 添加注销监听器
                imageView.setOnClickListener(exitLinstener);
                break;
            case 7:
                // 添加结算监听器
                imageView.setOnClickListener(payLinstener);
```

```
            break;
        default:
            break;
        }

        return imageView;
    }
    // 图片资源数组
    private Integer[] mThumbIds = {
            R.drawable.diancai, R.drawable.bingtai,
            R.drawable.zhuantai, R.drawable.chatai,
            R.drawable.gengxin, R.drawable.shezhi,
            R.drawable.zhuxiao, R.drawable.jietai
    };
    }
}
```

④ 定义一个单击监听器，打开点餐界面。

```
    // 订餐监听器
    OnClickListener orderLinstener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent();
            // 启动订餐Activity
            intent.setClass(MainMenuActivity.this,
OrderActivity.class);
            startActivity(intent);
        }
    };
```

程序运行结果如图 18.20 所示。

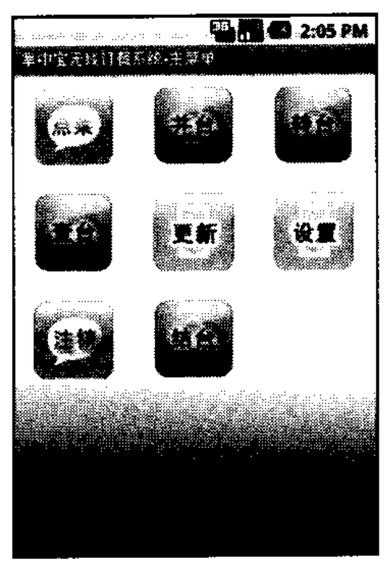


图 18.20　系统主菜单界面

18.4.4　点餐功能客户端实现

进入程序主菜单之后，单击“点菜”项便进入了点菜操作界面。点菜功能是本项目中最为复杂的一个功能，它的操作过程分为三个步骤：开桌、点菜和下单。

开桌的过程是要求操作员输入桌号和人数，将点餐时间、操作员编号、桌号和人数保存到服务器端的 OrderTbl 表中。点餐时间取服务器当前时间由程序自动完成，操作员编号取登录过程中保存在 SharedPreferences 中的登录用户信息，桌号引用 TableTbl 表中的数据，这里为了提高程序的运行效率，在客户端使用 SQLite 将服务器端数据保存下来，点餐过程所使用的桌号从客户端数据库取，程序提供了客户端数据和服务器端数据的同步功能，人数由操作员输入。

点菜，开桌后同一桌人可能要点好多道菜，这样在开桌的基础上进行点菜，点菜的过程是要求用户输入菜名称、数量和备注信息，将这些信息保存到服务器数据库 OrderDetailTbl 中。这里菜名称编号引用 MenuTbl 表中的数据，为了提高程序的运行效率，

将服务器数据库中的数据同步到客户端 SQLite 数据库中，程序提供了数据同步功能。数量由操作员输入，备注信息由操作员输入（如微辣、清淡等）。

下单的过程是将开桌信息和点菜信息保存到服务器数据库的 OrderTbl 表和 OrderDetailTbl 表中的过程。

点餐功能具体实现步骤如下所示。

① 在工程的 res\layout\目录下创建一个名称为“order.xml”的布局文件，该文件外层使用 LinearLayout 布局，内层嵌套 TableLayout。TableLayout 中有三行：第一行是一个 TextView 和一个 Spinner，提示操作员选择桌号；第二行是一个 TextView 和一个 EditText，要求用户输入人数；第三行是三个 Button，分别是开桌、点菜和下单。TableLayout 下面有一个 ListView 用来显示客户点菜列表。

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

<TableLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:stretchColumns="1">
      <TableRow>
         <TextView
         android:text="桌号:"
         android:id="@+id/TextView01"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>

         <Spinner
         android:id="@+id/tableNoSpinner"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>


      <TableRow>
         <TextView
         android:text="人数:"
         android:id="@+id/TextView02"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"></TextView>

         <EditText android:text=""
         android:id="@+id/personNumEditText02"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"></EditText>
      </TableRow>
```

```
        <TableRow android:gravity="right">

            <Button
            android:text="开桌"
            android:id="@+id/startButton02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

            <Button
            android:text="添菜"
            android:id="@+id/addMealButton01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

            <Button
            android:text="下单"
            android:id="@+id/orderButton02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        </TableRow>

    </TableLayout>

    <ListView
    android:id="@+id/orderDetailListView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

</LinearLayout>
```

② 创建一个名称为“OrderActivity”的 Activity，该 Activity 继承 Activity，在 OrderActivity 顶部声明使用到的实例，在 OnCreate 方法中初始化之。

```
package com.amaker.wlo;

import java.io.UnsupportedEncodingException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.http.NameValuePair;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;

import android.app.Activity;
```

```
import android.app.AlertDialog;
import android.content.ContentResolver;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.SimpleCursorAdapter;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import com.amaker.util.HttpUtil;

public class OrderActivity extends Activity {
    // 桌号下拉列表
    private Spinner tableNoSpinner;
    // 开桌、点菜和下单按钮
    private Button orderBtn, addBtn, startBtn;
    // 人数编辑框
    private EditText personNumEt;
    // 点菜列表
    private ListView lv;
    // 开桌生成的订单 Id
    private String orderId;
    // 点菜列表中绑定的数据
    private List data = new ArrayList();
    // 点菜列表中具体的数据项
    private Map map;
    // ListView 的 Adapter
    private SimpleAdapter sa;
    // ListView 中显示的数据项
    private String[] from = { "id", "name","num", "price", "remark" };
    // 引用的 TextView Drawable ID
    private int[] to = new int[5];

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 为 Activity 设置界面布局
        setContentView(R.layout.order);

        // 实例化 Spinner
        tableNoSpinner = (Spinner) findViewById(R.id.tableNoSpinner);
        // 为桌号下拉列表 Spinner 绑定数据
        setTableAdapter();

        // 实例化开桌按钮
```

```
        startBtn = (Button) findViewById(R.id.startButton02);
        // 为开桌按钮添加监听器
        startBtn.setOnClickListener(startListener);

        // 实例化点菜按钮
        addBtn = (Button) findViewById(R.id.addMealButton01);
        // 为点菜按钮添加监听器
        addBtn.setOnClickListener(addListener);

        // 实例化下单按钮
        orderBtn = (Button) findViewById(R.id.orderButton02);
        // 为下单按钮添加监听器
        orderBtn.setOnClickListener(orderListener);

        // 实例化人数编辑框
        personNumEt = (EditText) findViewById(R.id.personNumEditText02);

        // 实例化 ListView
        lv = (ListView) findViewById(R.id.orderDetailListView01);
        // 为点菜列表 ListView 绑定数据
        setMenusAdapter();
    }
}
```

③ 实现开桌功能，开桌功能是要求操作员选择桌号并输入人数，并将这些信息保存到服务器端 MySQL 数据库中的 OrderTbl 表中。这里需要为桌号下拉列表 Spinner 绑定数据，为开桌按钮添加监听器，实现具体操作。

3.1 绑定数据。代码段如下所示。

```
    // 为桌号下拉列表 Spinner 绑定数据
    private void setTableAdapter(){
        // 访问本地 SQLite 数据库中桌号表的 Uri
         Uri uri = Uri.parse("content://com.amaker.provider.TABLES/table");
        // 要选择桌号表中的列
        String[] projection = { "_id", "num", "description" };
        // 查询放回游标
        Cursor c = managedQuery(uri, projection, null, null, null);
        // 实例化桌号下拉列表 Spinner 的 Adapter
        SimpleCursorAdapter adapter2 = new SimpleCursorAdapter(this,
                android.R.layout.simple_spinner_item, c,
                new String[] { "_id" }, new int[] { android.R.id.text1 });

        // 为桌号 Spinner 绑定数据
        tableNoSpinner.setAdapter(adapter2);
    }
```

3.2 开桌监听器。代码段如下所示。

```
    // 开桌监听器
    private OnClickListener startListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            Date date = new Date();
            SimpleDateFormat sdf = new SimpleDateFormat("yy:mm:dd hh:MM");
```

```
            // 开桌时间
            String orderTime = sdf.format(date);
            // 开桌用户，从登录配置文件中获得
            SharedPreferences pres = getSharedPreferences("user_msg",
                    MODE_WORLD_READABLE);
            String userId = pres.getString("id", "");
            // 桌号
            TextView tv = (TextView) tableNoSpinner.getSelectedView();
            String tableId = tv.getText().toString();
            // 人数
            String personNum = personNumEt.getText().toString();
            // 请求参数列表
            List<NameValuePair> params = new ArrayList<NameValuePair>();
            // 添加请求参数
            params.add(new BasicNameValuePair("orderTime", orderTime));
            params.add(new BasicNameValuePair("userId", userId));
            params.add(new BasicNameValuePair("tableId", tableId));
            params.add(new BasicNameValuePair("personNum", personNum));
            UrlEncodedFormEntity entity1=null;
            try {
                 entity1 =  new UrlEncodedFormEntity(params,HTTP.UTF_8);
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
            // 请求服务器 url
            String url = HttpUtil.BASE_URL+"servlet/StartTableServlet";
            // 获得请求对象 HttpPost
            HttpPost request = HttpUtil.getHttpPost(url);
            // 设置查询参数
            request.setEntity(entity1);
            // 获得响应结果
            String result= HttpUtil.queryStringForPost(request);
            // 将开桌生成的订单 Id，赋值给全局 orderId
            orderId = result;
            Toast.
            makeText(OrderActivity.this, result, Toast.LENGTH_LONG).show();
        }
    };
```

④ 实现点菜功能。单击点菜按钮，弹出点菜对话框，要求操作员输入点菜信息。这里需要自定义一个 View 视图，添加到对话框，并且自定义视图中有一个 Spinner 引用菜单列表数据。

4.1 在工程的 res\layout\目录下创建一个名称为“order_detail.xml”的布局文件，该布局文件外层是 LinearLayout，内层嵌套 TableLayout，TableLayout 中有三行：第一行中有一个 TextView 和一个 Spinner，提示操作员选择菜名称；第二行有一个 TextView 和一个 EditText，提示操作员输入数量；第三行有一个 TextView 和一个 EditText，提示用户输入备注信息。

```
<?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">

   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">
      <TableRow>
         <TextView
         android:text="选菜："
         android:id="@+id/TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>

         <Spinner
         android:id="@+id/menuSpinner"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>

      <TableRow>
         <TextView
         android:text="数量："
         android:id="@+id/TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>

         <EditText
         android:text=""
         android:id="@+id/numEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>


      <TableRow>
         <TextView
         android:text="备注："
         android:id="@+id/TextView"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"/>

         <EditText
         android:text=""
         android:id="@+id/add_remarkEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"/>
      </TableRow>

   </TableLayout>

</LinearLayout>
```

4.2 为点菜列表 ListView 绑定数据。

```
// 为点菜列表 ListView 绑定数据
private void setMenusAdapter(){
    // 显示点菜项的 TextView 引用
    to[0] = R.id.id_ListView;
    to[1] = R.id.name_ListView;
    to[2] = R.id.num_ListView;
    to[3] = R.id.price_ListView;
    to[4] = R.id.remark_ListView;
    // 实例化点菜列表 ListView Adapter
    sa = new SimpleAdapter(this, data, R.layout.listview, from, to);
    // 为 ListView 绑定数据
    lv.setAdapter(sa);
}
```

4.3 为点菜按钮添加事件监听器。

```
// 添菜监听器
private OnClickListener addListener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        // 调用点菜方法
        addMeal();
    }
};

//添菜方法
private void addMeal() {
    // 获得 LayoutInflater 实例
    LayoutInflater inflater = LayoutInflater.from(this);
    // 实例化在弹出对话框中添加的视图
    final View v = inflater.inflate(R.layout.order_detail, null);
    // 获得视图中的 Spinner 对象，菜单下拉列表
    final Spinner menuSpinner = (Spinner) v.findViewById(R.id.menuSpinner);
    // 获得视图中的 EditText 对象,数量
    EditText numEt = (EditText) v.findViewById(R.id.numEditText);
    // 获得视图中的 EditText 实例，备注
    EditText remarkEt = (EditText) v.findViewById(R.id.add_remarkEditText);

    // 访问本地 SQLite 数据库中 MenuTbl 表的 Uri
    Uri uri = Uri.parse("content://com.amaker.provider.MENUS/menu1");
    // 查询列
    String[] projection = { "_id", "name", "price" };
    // 获得 ContentResolver 实例
    ContentResolver cr = getContentResolver();
    // 查询放回游标
    Cursor c = cr.query(uri, projection, null, null, null);
    // 实例化 SimpleCursorAdapter
    SimpleCursorAdapter adapter2 = new SimpleCursorAdapter(this,
            R.layout.spinner_lo, c,
            new String[] { "_id", "price", "name" }, new int[] {
                    R.id.id_TextView01, R.id.price_TextView02,
                    R.id.name_TextView03, });
    // 为点菜下拉列表 Spinner 绑定数据
    menuSpinner.setAdapter(adapter2);
```

```
// 获得AlertDialog.Builder实例
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder
// 设置标题
.setMessage("请点菜: ")
// 设置自定义视图
.setView(v)
// 设置确定按钮
.setPositiveButton("确定", new DialogInterface.OnClickListener() {
            // 确定按钮事件
            public void onClick(DialogInterface dialog, int id) {

                // 获得ListView中的自定义视图LinearLayout
                LinearLayout v1 = (LinearLayout) menuSpinner
                        .getSelectedView();

                // 获得TextView，菜编号
                TextView id_tv = (TextView) v1
                        .findViewById(R.id.id_TextView01);
                // 获得TextView，菜价格
                TextView price_tv = (TextView) v1
                        .findViewById(R.id.price_TextView02);
                // 获得TextView，菜名称
                TextView name_tv = (TextView) v1
                        .findViewById(R.id.name_TextView03);
                // 获得EditText，菜数量
                EditText num_et = (EditText) v
                        .findViewById(R.id.numEditText);
                // 获得EditText，菜备注
                EditText remark_et = (EditText) v
                        .findViewById(R.id.add_remarkEditText);
                // 菜编号值
                String idStr = id_tv.getText().toString();
                // 菜价格值
                String priceStr = price_tv.getText().toString();
                // 菜名称值
                String nameStr = name_tv.getText().toString();
                // 菜数量值
                String numStr = num_et.getText().toString();
                // 菜备注值
                String remarkStr = remark_et.getText().toString();

                // 封装到Map中
                map = new HashMap();

                map.put("id", idStr);
                map.put("name", nameStr);
                map.put("num", numStr);
                map.put("price", priceStr);
                map.put("remark", remarkStr);

                // 添加到ListView
                data.add(map);
```

```
                    // 关联的 TextView
                    to[0] = R.id.id_ListView;
                    to[1] = R.id.name_ListView;
                    to[2] = R.id.price_ListView;
                    to[3] = R.id.remark_ListView;

                    // 实例化 SimpleAdapter
                    sa = new SimpleAdapter(OrderActivity.this, data,
                            R.layout.listview, from, to);
                    // 为 ListView 绑定数据
                    lv.setAdapter(sa);

                }
            }).setNegativeButton("取消", null);
        AlertDialog alert = builder.create();
        alert.show();
    }
```

⑤ 下单，下单的过程是将顾客的点餐信息列表保存到后台数据库中，实现代码如下所示。

```
    // 下单监听器
    private OnClickListener orderListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 遍历点菜列表
            for (int i = 0; i < data.size(); i++) {
                // 获得其中点菜 map
                Map map = (Map)data.get(i);
                // 获得点菜项
                String menuId = (String) map.get("id");
                String num = (String)map.get("num");
                String remark = (String)map.get("remark");
                String myOrderId = orderId;

                // 封装到请求参数中
                List<NameValuePair> params = new ArrayList<NameValuePair>();
                // 添加到请求参数中
                params.add(new BasicNameValuePair("menuId", menuId));
                params.add(new BasicNameValuePair("orderId", myOrderId));
                params.add(new BasicNameValuePair("num", num));
                params.add(new BasicNameValuePair("remark", remark));
                UrlEncodedFormEntity entity1=null;
                try {
                    entity1 =  new UrlEncodedFormEntity(params,HTTP.UTF_8);
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
                // 请求服务器 Servlet 的 url
                String url = HttpUtil.BASE_URL+"servlet/OrderDetailServlet";
                // 获得 HttpPost 对象
                HttpPost request = HttpUtil.getHttpPost(url);
                // 为请求设置参数
                request.setEntity(entity1);
                // 获得返回结果
```

```
                String result= HttpUtil.queryStringForPost(request);

                Toast.
                makeText(OrderActivity.this, result, Toast.LENGTH_LONG).show();
            }
        }
    };
```

程序运行结果如图 18.21 所示。

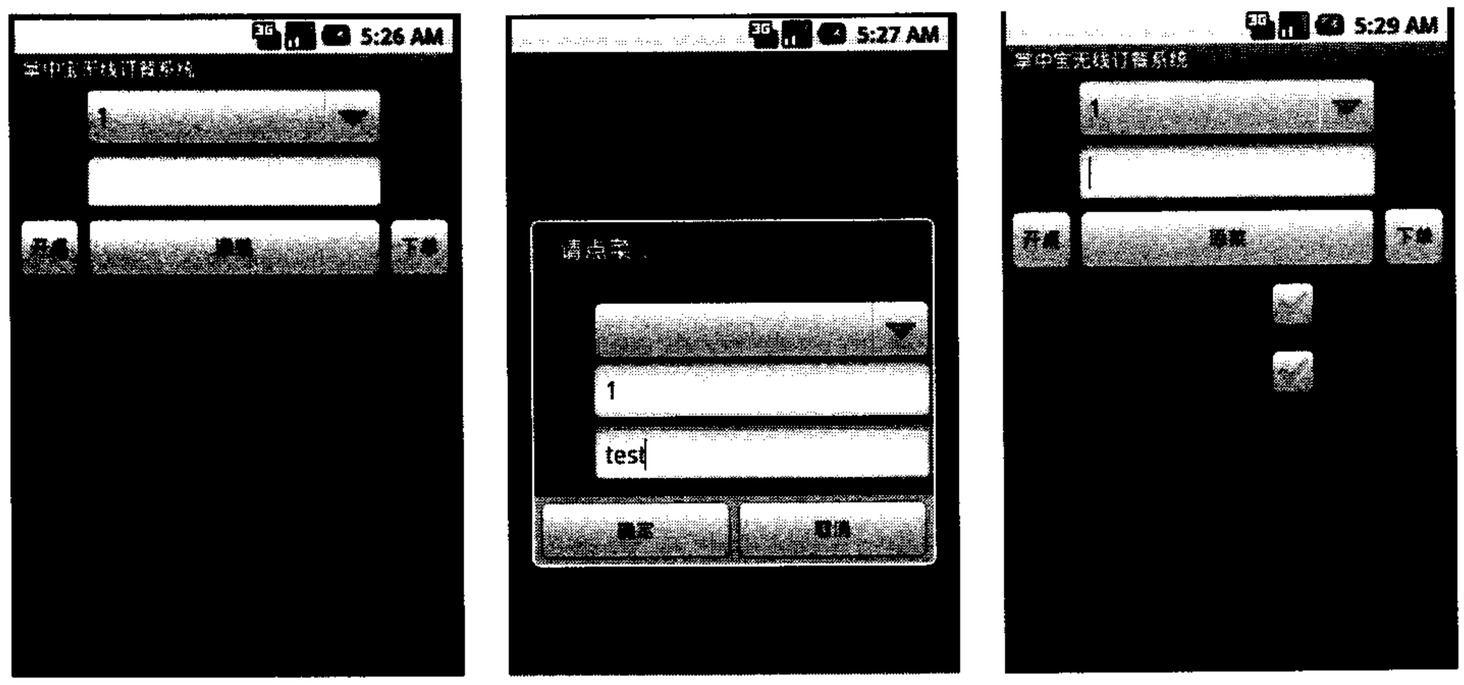


图 18.21　点餐模块运行结果

18.4.5　点餐功能服务器端实现

点餐功能的服务器实现主要有两个方面：一是创建 Servlet 响应客户端请求；二是创建 DAO 将数据保存到数据库。

该功能的具体实现步骤说明如下。

① 在“WirelessOrder_Server”工程的“com.amaker.entity”包中创建一个实体类 Order，该类封装了数据库中 OrderTbl 表的信息，以面向对象的方法来操作该表。

```
<?xml version="1.0" encoding="utf-8"?>
package com.amaker.entity;

public class Order {
    // 编号
    private int id;
    // 下单时间
    private String orderTime;
    // 下单用户
    private int userId;
    // 桌号
    private int tableId;
    // 人数
    private int personNum;
    // 是否结算
```

```
    private int isPay;
    // 备注
    private String remark;
    getters();
    setters();
}
```

② 在“WirelessOrder_Server”工程的“com.amaker.entity”包中创建一个实体类 OrderDetail，该类封装了数据库中 OrderDetailTbl 表的信息，以面向对象的方法来操作该表。

```
package com.amaker.entity;
public class OrderDetail {
    // 编号
    private int id;
    // 订单 Id
    private int orderId;
    // 菜谱 Id
    private int menuId;
    // 数量
    private int num;
    // 备注
    private String remark;
    getters();
    setters();
}
```

③ 在“com.amaker.dao”包中创建一个 OrderDao 接口，该接口中有四个方法，分别用来保存订单信息 Order、保存订单详细信息 OrderDetail、更新桌号状态为有人和更新桌号状态为空位。

```
package com.amaker.dao;
import com.amaker.entity.Order;
import com.amaker.entity.OrderDetail;
public interface OrderDao {
    // 保存开桌信息
    public int saveOrder(Order o );
    // 保存点菜列表信息
    public void saveOrderDetail(OrderDetail od);
    // 更新桌号状态，有人
    public void updateTableStatus(int tableId);
    // 更新桌号状态，空位
    public void updateTableStatus2(int orderId);
}
```

④ 在“com.amaker.dao.impl”包中创建一个 OrderDaoImpl 的实现类实现 OrderDao 接口里定义的方法。

```
package com.amaker.dao.impl;

import java.sql.Connection;
```

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.amaker.dao.OrderDao;
import com.amaker.entity.Order;
import com.amaker.entity.OrderDetail;
import com.amaker.util.DBUtil;
/**
 * @author 郭宏志
 * 点餐功能 DAO 实现类
 */
public class OrderDaoImpl implements OrderDao {

   // 保存点餐信息，放回订单 ID
   public int saveOrder(Order o) {
       // 添加 SQL 语句
       String sql = " insert into ordertbl(orderTime,userId,tableId,personNum)values
(?,?,?,?) ";
       // 数据库连接工具类
       DBUtil util = new DBUtil();
       // 获得连接
       Connection conn = util.openConnection();
       try {
          // 获得预定义语句
          PreparedStatement pstmt = conn.prepareStatement(sql);
          // 设置参数
          pstmt.setString(1, o.getOrderTime());
          pstmt.setInt(2, o.getUserId());
          pstmt.setInt(3, o.getTableId());
          pstmt.setInt(4, o.getPersonNum());
          // 执行更新
          pstmt.executeUpdate();
          // 返回订单编号
          String sql2 = " select max(id) as id  from orderTbl ";
          Statement stmt = conn.createStatement();
          ResultSet rs = stmt.executeQuery(sql2);
          if(rs.next()){
              int id = rs.getInt(1);
              return id;
          }
       } catch (SQLException e) {
          e.printStackTrace();
       } finally {
          util.closeConn(conn);
       }
       return 0;
   }
   // 保存点餐列表
   public void saveOrderDetail(OrderDetail od) {
       // 添加 SQL 语句
```

```
        String sql = " insert into orderdetailtbl(orderId,menuId,num,remark)values
(?,?,?,?) ";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置参数
            pstmt.setInt(1, od.getOrderId());
            pstmt.setInt(2, od.getMenuId());
            pstmt.setInt(3, od.getNum());
            pstmt.setString(4, od.getRemark());
            // 执行更新
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
    }

    // 更新桌号状态，有人
    public void updateTableStatus(int tableId) {
        // 更新 SQL 语句
        String sql = " update tableTbl set flag=1 where id = ? ";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置参数
            pstmt.setInt(1, tableId);
            // 执行更新
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
    }
    // 更新桌号状态，空桌
    public void updateTableStatus2(int orderId) {
        // 更新 SQL 语句
        String sql = " update TableTbl set flag = 0 where id = "+
                            " ( select tableId from OrderTbl where id = ?) ";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
```

```
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置参数
            pstmt.setInt(1, orderId);
            // 执行更新
            pstmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
    }
}
```

⑤ 在“com.amaker.servlet”包中创建一个名称为“StartTableServlet”的 Servlet，响应客户端请求，调用后台数据，保存数据，更新餐桌状态。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.OrderDao;
import com.amaker.dao.impl.OrderDaoImpl;
import com.amaker.entity.Order;

/**
 * @author 郭宏志
 * 开桌 Servlet，将操作员输入的信息保存到 OrderTbl 表中
 */
public class StartTableServlet extends HttpServlet {
    // 构造方法
    public StartTableServlet() {
        super();
    }

    // 销毁方法
    public void destroy() {
        super.destroy();
    }

    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
```

```
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // 点餐时间
        String orderTime = request.getParameter("orderTime");
        // 操作员编号
        String userId = request.getParameter("userId");
        // 桌号
        String tableId = request.getParameter("tableId");
        // 人数
        String personNum = request.getParameter("personNum");
        // 获得 DAO 接口
        OrderDao dao = new OrderDaoImpl();
        // 实例化订单类
        Order o = new Order();
        // 设置订单属性
        o.setOrderTime(orderTime);
        o.setPersonNum(Integer.parseInt(personNum));
        o.setUserId(Integer.parseInt(userId));
        o.setTableId(Integer.parseInt(tableId));
        // 返回订单 ID
        int id = dao.saveOrder(o);

        // 更新餐桌状态为有人
        dao.updateTableStatus(Integer.parseInt(tableId));
        // 返回 ID
        out.print(id);
        out.flush();
        out.close();
    }
    // 响应 Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    // 初始化
    public void init() throws ServletException {
    }
}
```

18.4.6　结算模块 Android 客户端实现

结算模块的流程是，根据订单编号查询订单详细信息，将详细信息通过一个 WebView 来显示，这是一个查询的过程，服务器返回 html 格式信息，通过 WebView 来显示。

当客户确认点餐信息后，操作员单击结算按钮，将数据库服务器中 OrderTbl 表中的 isPay 列数据更新为“1”，表示已结算（默认值为“0”，表示未结算）。

具体实现步骤说明如下。

① 在工程的 res\layout\目录下创建一个布局文件 pay.xml，该布局文件外层是 LinearLayout，内层又嵌套了一个 LinearLayout。内层 LinearLayout 中有 TextView、EditText

和 Button 三个组件，分别用来提示输入订单编号、输入订单编号和查询点餐信息。外层 LinearLayout 里面还有一个 WebView 和一个 Button，分别用来显示点餐信息和结算。

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

    <LinearLayout
    android:id="@+id/mainlayout"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:text="请输入订单号："
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:text=""
        android:id="@+id/pay_order_number_EditText01"
        android:layout_width="100px"
        android:layout_height="wrap_content"/>
    <Button
        android:text="查询"
        android:id="@+id/pay_query_Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
    <WebView
        android:id="@+id/pay_webview"
        android:layout_width="fill_parent"
        android:layout_height="300px"/>

    <Button
        android:text="结算"
        android:id="@+id/pay_Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</LinearLayout>
```

② 创建一个名称为“PayActivity”的 Activity。在该类的顶部声明使用到的实例对象，在 onCreate()方法中进行初始化。

```
package com.amaker.wlo;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.Toast;
import com.amaker.util.HttpUtil;

public class PayActivity extends Activity{
    // 显示点餐信息 WebView
    private WebView wv;
    // 查询点餐信息按钮和结算按钮
    private Button queryBtn,payBtn;
    // 订单编号
    private EditText orderIdEt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.pay);
        // 获得 WebView 实例
        wv = (WebView)findViewById(R.id.pay_webview);
        // 实例化查询按钮
        queryBtn = (Button)findViewById(R.id.pay_query_Button01);
        // 实例化结算按钮
        payBtn = (Button)findViewById(R.id.pay_Button01);
        // 实例化订单编号编辑框
        orderIdEt = (EditText)findViewById(R.id.pay_order_number_EditText01);

        // 添加查询点餐信息监听器
        queryBtn.setOnClickListener(queryListener);
        // 添加结算息监听器
        payBtn.setOnClickListener(payListener);
    }
}
```

③ 为查询按钮添加单击事件监听器，响应事件的内容时，查询服务器返回 html 格式点餐信息，并在 WebView 中显示。

```
    // 查询点餐信息监听器
    OnClickListener queryListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 获得订单编号
            String orderId = orderIdEt.getText().toString();
            // 请求服务器 url
            String url = HttpUtil.BASE_URL+"servlet/PayServlet?id="+orderId;
            // 将返回信息在 WebView 中显示
            wv.loadUrl(url);
        }
    };
```

④ 为结算按钮添加单击事件监听器，响应事件的内容时，请求服务器根据订单编号，将该订单信息的 isPay 数据更新为“1”，表示已结算。

```
    // 结算监听器
    OnClickListener payListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            // 获得订单编号
            String orderId = orderIdEt.getText().toString();
            // 请求服务器 url
```

```
            String url = HttpUtil.BASE_URL+"servlet/PayMoneyServlet?id="+orderId;
            // 获得查询结果
            String result = HttpUtil.queryStringForPost(url);
            // 显示结算结果
            Toast.makeText(PayActivity.this, result, Toast.LENGTH_LONG).show();
            // 使结算按钮失效
            payBtn.setEnabled(false);
        }
    };
```

运行程序结果如图 18.22 所示。

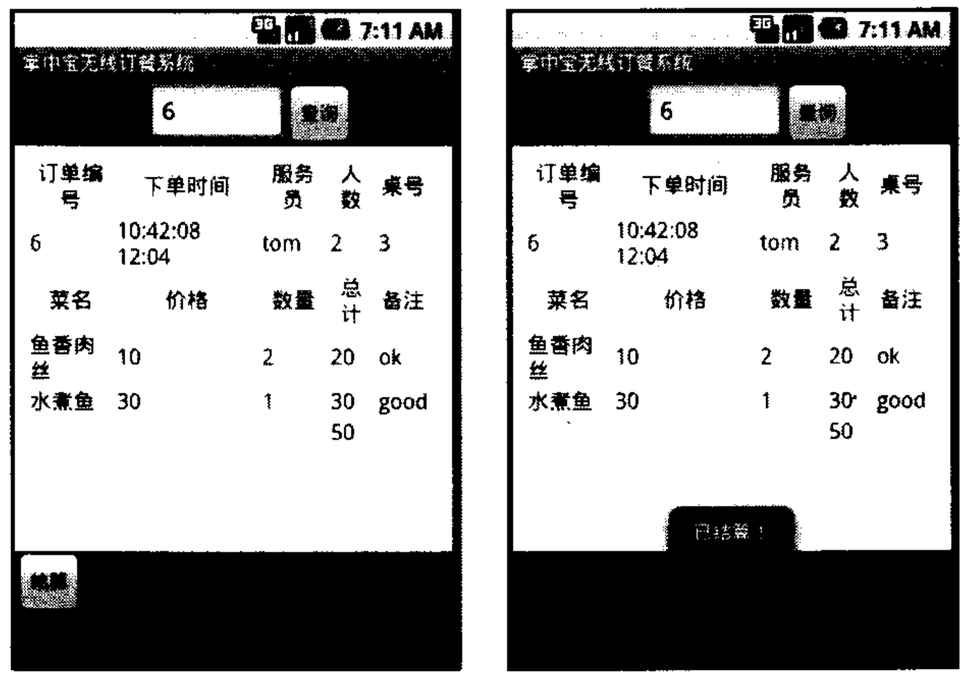


图 18.22　结算模块运行结果

18.4.7　结算模块服务器端实现

结算模块的服务器端实现，主要是定义 Servlet 响应客户端的请求，定义 DAO 接口访问数据库。这里需要根据订单编号查询订单信息和订单详细信息，以及更新订单的结算状态和餐桌的状态。

具体实现步骤说明如下。

① 在工程的“com.amaker.entity”包中创建一个 QueryOrder 类，封装查询订单信息。

```
package com.amaker.entity;
public class QueryOrder {
    // 下单时间
    private String orderTime;
    // 服务员名称
    private String name;
    // 顾客人数
    private int personNum;
    // 桌号
    private int tableId;

    getters();
    setters();
}
```

② 在工程的“com.amaker.entity”包中创建一个 QueryOrderDetail 类，封装查询订单详细信息。

```
        @Override
package com.amaker.entity;

public class QueryOrderDetail {
    // 菜名称
    private String name;
    // 价格
    private int price;
    // 数量
    private int num;
    // 总计
    private int total;
    // 备注
    private String remark;

    getters();
    setters();

}
```

③ 在“com.amaker.dao”包中创建一个 PayDao 接口，在该接口中定义三个方法，分别是查询订单信息、查询订单详细信息和结算。

```
package com.amaker.dao;
import java.util.List;
import com.amaker.entity.QueryOrder;

public interface PayDao {
    // 查询订单信息
    public QueryOrder getOrderById(int id);
    // 查询订单详细
    public List getOrderDetailList(int id);
    // 结算
    public void pay(int id);
}
```

④ 在“com.amaker.dao.impl”包中创建一个 PayDaoImpl 类，该类实现了 PayDao 接口中的方法。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
```

```
import com.amaker.dao.PayDao;
import com.amaker.entity.QueryOrder;
import com.amaker.entity.QueryOrderDetail;
import com.amaker.util.DBUtil;
/**
 * @author 郭宏志
 * 结算 DAO 实现类
 */
public class PayDaoImpl implements PayDao {

    // 根据订单编号，查询订单信息
    public QueryOrder getOrderById(int id) {
        // 查询 SQL 语句
        String sql ="   select ot.`orderTime`, "+
                    "   ut.`name`, "+
                    "   ot.`personNum`, "+
                    "   ot.`tableId` "+
                    "   from orderTbl as ot "+
                    "   left join userTbl as ut on ot.`userID` = ut.id "+
                    "   where ot.`id`=? ";

        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 设置查询参数
            pstmt.setInt(1, id);
            // 执行查询
            ResultSet rs = pstmt.executeQuery();
            // 判断订单是否存在
            if (rs.next()) {
                // 获得订单信息
                String orderTime = rs.getString(1);
                String userName = rs.getString(2);
                int personNum =rs.getInt(3);
                int tableId = rs.getInt(4);
                QueryOrder qo = new QueryOrder();
                qo.setName(userName);
                qo.setOrderTime(orderTime);
                qo.setPersonNum(personNum);
                qo.setTableId(tableId);
                return qo;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
```

```
    return null;
}
// 根据订单编号，查询订单详细列表
public List getOrderDetailList(int id) {
    // 查询 SQL 语句
    String sql ="   select mt.`name`, "+
                "   mt.`price`, "+
                "   odt.`num`, "+
                "   mt.price*odt.num as total, "+
                "   odt.`remark` "+
                "   from orderdetailTbl as odt "+
                "   left join menuTbl as mt on odt.`menuId` = mt.id "+
                "   where odt.`orderId`= ?"+

                "   union "+

                "   select        '',"+
                "   '',"+
                "   '',"+
                "   sum(mt.price*odt.num) as total1,"+
                "   '' "+
                "   from orderdetailTbl as odt"+
                "   left join menuTbl as mt on odt.`menuId` = mt.id"+
                "   where odt.`orderId`= ? ";

    // 数据库连接工具类
    DBUtil util = new DBUtil();
    // 获得连接
    Connection conn = util.openConnection();
    try {
        // 获得预定义语句
        PreparedStatement pstmt = conn.prepareStatement(sql);
        // 设置查询参数
        pstmt.setInt(1, id);
        pstmt.setInt(2, id);
        // 执行查询
        ResultSet rs = pstmt.executeQuery();
        // 判断订单详细

        List list = new ArrayList();

        while (rs.next()) {
            // 获得订单详细信息
            String name = rs.getString(1);
            int price = rs.getInt(2);
            int num = rs.getInt(3);
            int total = rs.getInt(4);
            String remark = rs.getString(5);

            QueryOrderDetail qod = new QueryOrderDetail();

            qod.setName(name);
```

```
            qod.setNum(num);
            qod.setPrice(price);
            qod.setTotal(total);
            qod.setRemark(remark);

            list.add(qod);
        }
        return list;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        util.closeConn(conn);
    }
    return null;
}

// 结算
public void pay(int id) {
    // 查询SQL语句
    String sql =" update OrderTbl set isPay=1 where id = ?";
    // 数据库连接工具类
    DBUtil util = new DBUtil();
    // 获得连接
    Connection conn = util.openConnection();
    try {
        // 获得预定义语句
        PreparedStatement pstmt = conn.prepareStatement(sql);
        // 设置查询参数
        pstmt.setInt(1, id);
        // 执行更新
        pstmt.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        util.closeConn(conn);
    }
  }
}
```

⑤ 在“com.amaker.servlet”包中定义一个名称为“PayServlet”的 Servlet 类，该类根据订单编号查询订单信息和订单详细信息，拼成 HTML 页面发送给客户端。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import com.amaker.dao.PayDao;
import com.amaker.dao.impl.PayDaoImpl;
import com.amaker.entity.QueryOrder;
import com.amaker.entity.QueryOrderDetail;

/**
 * @author 郭宏志
 * 根据订单编号，查询订单详细信息
 */
public class PayServlet extends HttpServlet {
    // 构造方法
    public PayServlet() {
        super();
    }

    // 销毁方法
    public void destroy() {
        super.destroy();
    }

    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        // 实例化 DAO
        PayDao dao = new PayDaoImpl();
        // 获得订单 ID
        String id = request.getParameter("id");
        // 查询订单信息
        QueryOrder qo = dao.getOrderById(Integer.parseInt(id));
        // 查询订单详细列表
        List list = dao.getOrderDetailList(Integer.parseInt(id));

        // 拼成 HTML 页面展示
        out
                .println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional
//EN\">");
        out.println("<HTML>");
        out.println("  <HEAD></HEAD>");
        out.println("  <BODY>");
        out.print("<table>");
            out.print("<tr>");

                out.print("<th>");
                    out.print("订单编号");
                out.print("</th>");

                out.print("<th>");
                    out.print("下单时间");
                out.print("</th>");

                out.print("<th>");
                    out.print("服务员");
                out.print("</th>");

                out.print("<th>");
                    out.print("人数");
                out.print("</th>");
```

```
            out.print("<th>");
                out.print("桌号");
            out.print("</th>");

        out.print("</tr>");

        out.print("<tr>");

            out.print("<td>");
                out.print(id);
            out.print("</td>");

            out.print("<td>");
                out.print(qo.getOrderTime());
            out.print("</td>");

            out.print("<td>");
                out.print(qo.getName());
            out.print("</td>");

            out.print("<td>");
                out.print(qo.getPersonNum());
            out.print("</td>");

            out.print("<td>");
                out.print(qo.getTableId());
            out.print("</td>");

        out.print("</tr>");


        out.print("<tr>");

        out.print("<th>");
            out.print("菜名");
        out.print("</th>");

        out.print("<th>");
            out.print("价格");
        out.print("</th>");

        out.print("<th>");
            out.print("数量");
        out.print("</th>");

        out.print("<th>");
            out.print("总计");
        out.print("</th>");

        out.print("<th>");
            out.print("备注");
        out.print("</th>");

    out.print("</tr>");

        for (int i = 0; i < list.size(); i++) {
            QueryOrderDetail qod = (QueryOrderDetail) list.get(i);
            String name = qod.getName();
```

```
                int price = qod.getPrice();
                int num = qod.getNum();
                int total = qod.getTotal();
                String remark = qod.getRemark();

                out.print("<tr>");

                    out.print("<td>");
                        out.print(name);
                    out.print("</td>");

                    out.print("<td>");
                        out.print(price==0? "" :price+"");
                    out.print("</td>");

                    out.print("<td>");
                        out.print(num==0? "" :num+"");
                    out.print("</td>");

                    out.print("<td>");
                        out.print(total);
                    out.print("</td>");

                    out.print("<td>");
                        out.print(remark);
                    out.print("</td>");

                out.print("</tr>");
            }
        out.print("</table>");
        out.println("  </BODY>");
        out.println("</HTML>");
        out.flush();
        out.close();
    }
    // 响应 Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
}
```

⑥ 在“com.amaker.servlet”包中定义一个名称为“PayMoneyServlet”的 Servlet 类，该类实现了结算功能。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import com.amaker.dao.OrderDao;
import com.amaker.dao.PayDao;
import com.amaker.dao.impl.OrderDaoImpl;
import com.amaker.dao.impl.PayDaoImpl;
/**
 * @author 郭宏志
 * 实现结算功能
 */
public class PayMoneyServlet extends HttpServlet {
    // 构造方法
    public PayMoneyServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        // 实例化 PayDao
        PayDao dao = new PayDaoImpl();
        // 获得订单编号
        String id = request.getParameter("id");
        // 结算
        dao.pay(Integer.parseInt(id));
        // 实例化 OrderDao
        OrderDao dao2 = new OrderDaoImpl();
        // 将餐桌状态更新为空位
        dao2.updateTableStatus2(Integer.parseInt(id));
        // 向客户端发送信息
        out.print("已结算！");
        out.flush();
        out.close();
    }
    // 响应 Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
}
```

18.4.8 查台模块 Android 客户端实现

查台的目的是根据当前餐桌的状态（有人或空位），来安排顾客就餐。程序的实现过

程是：查询数据库餐桌表信息，根据查询结果将餐桌状态以图形化的方式显示出来。例如，某号桌位有人还是空位。

程序的实现步骤说明如下。

① 在工程的 res\layout\目录下创建一个名称为“check_table.xml”的布局文件。该布局文件采用 LinearLayout 来布局，内部嵌套了一个 GridView 以图形化的方式来展现餐桌的当前状态。

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

② 创建一个名称为“CheckTableActivity”的 Activity，在该类的顶部声明使用到的对象，在 onCreate()方法中实例化之。

```
<?xml version="1.0" encoding="utf-8"?>
package com.amaker.wlo;

import java.util.ArrayList;
import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;
import android.widget.TextView;

import com.amaker.util.CheckTable;
import com.amaker.util.HttpUtil;
/**
 * 查台
 * @author 郭宏志
 */
```

```
public class CheckTableActivity extends Activity{
    // 显示餐桌状态的 GridView
    private GridView gv;
    // 餐桌数量
    private int count;
    // 保存餐桌信息的列表
    private List list = new ArrayList();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置标题
        setTitle("掌中宝无线订餐系统-查台");
        // 设置当前 Activity 的界面布局
        setContentView(R.layout.check_table);
        // 实例化
        gv = (GridView) findViewById(R.id.check_table_gridview);
        //获得餐桌列表
        getTableList();
        // 为 GridView 绑定数据
        gv.setAdapter(new ImageAdapter(this));
    }

}
```

③ 在该类中定义一个 getTableList()，并在 onCreate()方法中调用之，getTableList()方法获得当前餐桌信息列表，信息包括桌号和状态。

```
    // 获得餐桌信息列表，信息包括桌号和状态
    private void getTableList(){
        // 访问服务器 url
        String url = HttpUtil.BASE_URL+"servlet/CheckTableServlet";
        // 查询返回结果
        String result = HttpUtil.queryStringForPost(url);
        // 拆分字符串，转换成对象，添加到列表
        String[] strs = result.split(";");
        for (int i = 0; i < strs.length; i++) {
            int idx = strs[i].indexOf(",");
            int num = Integer.parseInt(strs[i].substring(0, idx));
            int flag = Integer.parseInt(strs[i].substring(idx+1));
            CheckTable ct = new CheckTable();
            ct.setFlag(flag);
            ct.setNum(num);
            list.add(ct);
        }
    }
```

④ 定义一个内部类 ImageAdapter，该类继承 BaseAdapter 类，为 GridView 绑定数据。该类的 getView()方法返回的 View 类是自定义的一个 LinearLayout 类，内容如下所示：

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

```
    android:orientation="vertical"
    >

<ImageView
android:id="@+id/check_table_ImageView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>

<TextView
android:text=""
android:id="@+id/check_tableTextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textSize="20dp"
android:gravity="center"
android:textColor="#ff0000"
/>

</LinearLayout>
```

ImageAdapter 的代码如下所示。

```
// 继承 BaseAdapter
public class ImageAdapter extends BaseAdapter {
    // 上下文
    private Context mContext;
    // 构造方法
    public ImageAdapter(Context c) {
        mContext = c;
    }
    // 组件个数
    public int getCount() {
        return list.size();
    }
    // 当前组件
    public Object getItem(int position) {
        return null;
    }
    // 当前组件 id
    public long getItemId(int position) {
        return 0;
    }
    // 获得当前视图
    public View getView(int position, View convertView, ViewGroup parent) {
    // 声明图片视图
    LayoutInflater inflater =
        LayoutInflater.from(CheckTableActivity.this);
    View v = null;
    ImageView imageView =null;
    TextView tv =null;
        if (convertView == null) {
            // 实例化图片视图
```

```
            v = inflater.inflate(R.layout.check_table_view,null);
            // 设置图片视图属性
            v.setPadding(8, 8, 8, 8);
        } else {
            v = (View) convertView;
        }
        // 获得 ImageView 对象
        imageView = (ImageView) v.findViewById(R.id.check_table_ImageView01);
        // 获得 TextView 对象
        tv = (TextView) v.findViewById(R.id.check_tableTextView01);
        // 获得 CheckTable 对象
        CheckTable ct = (CheckTable) list.get(position);
        if(ct.getFlag()==1){
            // 设置 ImageView 图片为 有人
            imageView.setImageResource(R.drawable.youren);
        }else{
            // 设置 ImageView 图片为 空位
            imageView.setImageResource(R.drawable.kongwei);
        }
        // 为 TextView 设置桌号
        tv.setText(ct.getNum()+"");
        return v;
    }
}
```

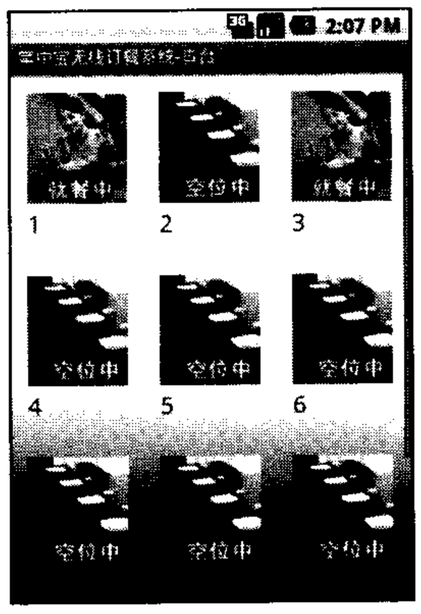


图 18.23　查台模块运行结果

程序运行结果如图 18.23 所示。

18.4.9　查台模块服务器端实现

查台的服务器端实现主要是定义 Servlet 响应客户端请求和定义 DAO 接口操作数据库两个部分。这里主要是查询数据库返回餐桌列表信息。

具体实现步骤说明如下。

① 在工程的“com.amaker.entity”包中定义一个 CheckTable 类，该类封装了餐桌的状态信息，包括桌号和状态两个属性。

```
package com.amaker.entity;
public class CheckTable {
    // 桌号
    private int num;
    // 状态
    private int flag;
    public int getFlag() {
        return flag;
    }
    public void setFlag(int flag) {
        this.flag = flag;
    }
    public int getNum() {
        return num;
```

```
    }
    public void setNum(int num) {
        this.num = num;
    }
}
```

② 在工程的“com.amaker.dao”包中创建一个 CheckTableDao 接口，该接口定义一个返回餐桌信息列表的方法。

```
package com.amaker.dao;
import java.util.List;
public interface CheckTableDao {
    // 获得餐桌列表
    public List getTableList();
}
}
```

③ 在工程的“com.amaker.dao.impl”包中创建一个 CheckTableDaoImpl 类，实现 CheckTableDao 接口。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import com.amaker.dao.CheckTableDao;
import com.amaker.entity.CheckTable;
import com.amaker.util.DBUtil;
/**
 * @author 郭宏志
 * 查台 DAO 实现类
 */
public class CheckTableDaoImpl implements CheckTableDao {

    // 获得餐桌列表
    public List getTableList() {
        // 查询 SQL 语句
        String sql =" select num,flag from tableTbl";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            Statement pstmt = conn.createStatement();
            ResultSet rs = pstmt.executeQuery(sql);
            List list = new ArrayList();
            while (rs.next()) {
```

```
            int num = rs.getInt(1);
            int flag = rs.getInt(2);
            CheckTable ct = new CheckTable();
            ct.setFlag(flag);
            ct.setNum(num);
            list.add(ct);
        }
        return list;
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        util.closeConn(conn);
    }
    return null;
  }
}
```

④ 在工程的“com.amaker.servlet”包中创建一个 CheckTableServlet 类，该类响应客户端请求，返回餐桌信息列表字符串。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.CheckTableDao;
import com.amaker.dao.impl.CheckTableDaoImpl;
import com.amaker.entity.CheckTable;
/**
 * @author 郭宏志
 * 实现查台功能
 */
public class CheckTableServlet extends HttpServlet {
    // 构造方法
    public CheckTableServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

```
        // 实例化 CheckTableDao
        CheckTableDao dao = new CheckTableDaoImpl();
        // 获得餐桌信息列表
        List list = dao.getTableList();
        // 转换为字符串
        String msg = build(list);
        // 返回给客户端
        out.print(msg);
        out.flush();
        out.close();
    }
    // 将 List 列表转化为字符串
    private String build(List list){
        String msg = "";
        for (int i = 0; i < list.size(); i++) {
            CheckTable ct = (CheckTable) list.get(i);
            int num = ct.getNum();
            int flag = ct.getFlag();
            msg+=num+","+flag;
            if(i<(list.size()-1))msg+=";";
        }
        return msg;
    }
    // 响应 Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    // 初始化
    public void init() throws ServletException {
    }
}
```

18.4.10　更新模块 Android 客户端实现

为了提高程序的运行效率，将一些频繁使用的数据库表保存在客户端 SQLite 数据库中。当服务器数据库中的数据发生变化时，将数据及时同步到客户端。这就是本模块要完成的功能。

该功能的实现过程是：将程序数据库服务器表中的数据封装成 XML 格式的数据发送到客户端，客户端通过 Java DOM 的方式解析 XML 文件，将其内容保存到客户端 SQLite 数据库中。

具体实现步骤说明如下。

① 在“com.amaker.wlo”包中创建一个名称为“UpdateActivity”的 Activity，在该类继承 ListActivity，界面通过 ListView 来展示。在 onCreate()方法中获得 ListView 实例，并为 ListView 绑定数据。

```
package com.amaker.wlo;
```

```
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

import android.app.AlertDialog;
import android.app.ListActivity;
import android.content.ContentResolver;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListAdapter;
import android.widget.ListView;

import com.amaker.provider.Menus;
import com.amaker.util.HttpUtil;
/**
 *
 * @author 郭宏志
 * 实现数据同步功能
 */
public class UpdateActivity extends ListActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // 设置标题
        setTitle("掌中宝无线订餐系统-数据同步");
        // 获得 ListView 实例
        ListView listView = getListView();
        // 声明 ListView 要绑定的数据
        String[] items = {"更新菜谱表数据[MenuTbl]", "更新餐桌表数据[TableTbl]" };
        // 实例化 adapter
        ListAdapter adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, items);
        // 为 ListView 设置 adapter
        listView.setAdapter(adapter);
    }
}
```

② 在该类中定义一个更新菜谱表的方法 updateMenu()，该方法调用服务器 Servlet 获得 XML 格式的输入流，通过 Java DOM 解析该 XML 获得菜谱表数据，将数据保存到本地 SQLite 数据库。

```
// 更新菜谱表
private void updateMenu() {
    // 访问服务器 url
    String urlStr = HttpUtil.BASE_URL + "servlet/UpdateServlet";
```

```
try {
    // 实例化 URL 对象
    URL url = new URL(urlStr);
    // 打开连接
    URLConnection conn = url.openConnection();
    // 获得输入流
    InputStream in = conn.getInputStream();
    // 实例化 DocumentBuilderFactory
    DocumentBuilderFactory factory = DocumentBuilderFactory
            .newInstance();
    // 实例化 DocumentBuilder
    DocumentBuilder builder = factory.newDocumentBuilder();
    // 获得 Document
    Document doc = builder.parse(in);
    // 获得节点列表
    NodeList nl = doc.getElementsByTagName("menu");
    // 获得访问数据接口 ContentResolver
    ContentResolver cr = getContentResolver();
    // 访问数据的 Uri
    Uri uri1 = Menus.CONTENT_URI;
    // 删除本地 SQLite 数据库中菜谱表中的数据
    cr.delete(uri1, null, null);

    // 循环将数据保存到菜谱表
    for (int i = 0; i < nl.getLength(); i++) {
        // 实例化 ContentValues
        ContentValues values = new ContentValues();
        // 解析 XML 文件获得菜单 id
        int id = Integer.parseInt(doc.getElementsByTagName("id")
                .item(i).getFirstChild().getNodeValue());
        // 名称
        String name = doc.getElementsByTagName("name").item(i)
                .getFirstChild().getNodeValue();
        // 图片路径
        String pic = doc.getElementsByTagName("pic").item(i)
                .getFirstChild().getNodeValue();
        // 价格
        int price = Integer.parseInt(doc.getElementsByTagName("price")
                .item(i).getFirstChild().getNodeValue());
        // 分类编号
        int typeId = Integer.parseInt(doc
                .getElementsByTagName("typeId").item(i).getFirstChild()
                .getNodeValue());
        // 备注
        String remark = doc.getElementsByTagName("remark").item(i)
                .getFirstChild().getNodeValue();

        // 添加到 ContenValues 对象
        values.put("_id", id);
        values.put("name", name);
        values.put("price", price);
        values.put("pic", pic);
```

```
                values.put("typeId", typeId);
                values.put("remark", remark);
                // 插入到数据库
                cr.insert(uriI, values);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

③ 定义一个 confirm()方法，提示用户确认是否真的要同步数据，防止用户误操作造成不必要的错误。

```
// 确认对话框
private void confirm(final int item) {
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage("你真的要更新吗?").setCancelable(false).setPositiveButton(
            "确定", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    if (item == 1) {
                        // 更新菜谱表数据
                        updateMenu();
                    } else {
                        // 更新桌位表数据
                        updateTable();
                    }
                }
            }).setNegativeButton("取消",
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });
    AlertDialog alert = builder.create();
    alert.show();
}
```

④ 覆盖 ListActivity 的 onListItemClick()方法，响应 ListView 列表项的单击事件。

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    switch (position) {
    // 更新菜谱表数据
    case 0:
        confirm(1);
        break;
    // 更新桌位表数据
    case 1:
        confirm(2);
        break;
    default:
        break;
```

```
        }
    }
```

程序运行结果如图 18.24 所示。

图 18.24　更新模块运行结果

18.4.11　更新模块服务器端实现

更新模块服务器端的实现主要是：创建 DAO 查询数据列表；创建 Servlet，拼成 XML 格式数据输出，响应客户端请求。具体步骤说明如下。

① 在工程的“com.amaker.dao”包中定义一个 UpdateDao 接口，在该接口中声明两个方法，分别用来获得菜谱数据列表和桌位数据列表。

```
package com.amaker.dao;
import java.util.List;
public interface UpdateDao {
    // 获得菜单列表
    public List getMenuList();
    // 获得餐桌列表
    public List getTableList();
}
```

② 在工程的“com.amaker.dao.impl”包中定义一个 UpdateDaoImpl 类实现 UpdateDao 接口中的方法。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import com.amaker.dao.UpdateDao;
import com.amaker.entity.Menu;
/**
 * @author 郭宏志
 * 完成更新数据功能
 */
public class UpdateDaoImpl implements UpdateDao {
    // 获得菜单列表
    public List getMenuList() {
        // 查询 SQL 语句
        String sql =" select id,typeId,price,name,pic,remark from MenuTbl ";
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
```

```
        Connection conn = util.openConnection();
        try {
            // 获得预定义语句
            Statement pstmt = conn.createStatement();
            // 执行查询
            ResultSet rs = pstmt.executeQuery(sql);
            // 判断订单详细
            List list = new ArrayList();
            while (rs.next()) {
                // 获得菜单信息

                int id = rs.getInt(1);
                int typeId = rs.getInt(2);
                int price = rs.getInt(3);
                String name = rs.getString(4);
                String pic = rs.getString(5);
                String remark = rs.getString(6);

                Menu m = new Menu();
                m.setId(id);
                m.setName(name);
                m.setPic(pic);
                m.setPrice(price);
                m.setRemark(remark);
                m.setTypeId(typeId);

                list.add(m);
            }
            return list;
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            util.closeConn(conn);
        }
        return null;
    }
    // 获得餐桌列表
    public List getTableList() {
        return null;
    }
}
```

③ 在工程的“com.amaker.servlet”包中定义一个名称为“UpdateServlet”的 Servlet 类响应客户端的请求，将数据列表拼成 XML 格式数据，发送到客户端。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.UpdateDao;
import com.amaker.dao.impl.UpdateDaoImpl;
import com.amaker.entity.Menu;

/**
 * @author 郭宏志
 * 完成数据更新功能
 */
public class UpdateServlet extends HttpServlet {
    // 构造方法
    public UpdateServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/xml");
        PrintWriter out = response.getWriter();
        // 实例化 dao
        UpdateDao dao = new UpdateDaoImpl();
        // 获得菜谱列表
        List list = dao.getMenuList();

        // 拼成 XML 格式数据
        out.println("<?xml version='1.0' encoding='UTF-8'?>");
        // 根节点
        out.println("<menulist>");
            for (int i = 0; i <list.size(); i++) {
                Menu m = (Menu)list.get(i);
                out.println("<menu>");
                    // 菜谱编号
                    out.print("<id>");
                        out.print(m.getId());
                    out.println("</id>");
                    // 分类
                    out.print("<typeId>");
                        out.print(m.getTypeId());
                    out.println("</typeId>");
                    // 名称
                    out.print("<name>");
                        out.print(m.getName());
                    out.println("</name>");
                    // 图片路径
                    out.print("<pic>");
```

```
                    out.print(m.getPic());
                out.println("</pic>");
                // 价格
                out.print("<price>");
                    out.print(m.getPrice());
                out.println("</price>");
                // 备注
                out.print("<remark>");
                    out.print(m.getRemark());
                out.println("</remark>");

            out.println("</menu>");
        }
    out.println("</menulist>");
    out.flush();
    out.close();
  }
  // 响应 Post 请求
  public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    doGet(request,response);
  }
  // 初始化方法
  public void init() throws ServletException {
  }
}
```

18.4.12 转台模块 Android 客户端实现

转台模块要实现的功能是：当顾客开桌或下单后要求更换桌位的过程。这一过程需要更新保存在服务器数据库中的数据，具体更新的是 OrderTbl 表中该订单对应的桌号以及更换桌位的状态。具体实现步骤说明如下。

① 在工程的 res\layout\目录下定义一个名称为“change_table.xml”的布局文件，该布局文件外层使用 LinearLayout 布局，内层嵌套 TableLayout，TableLayout 中有两行要求用户输入订单编号和更换桌号。该布局作为自定义视图添加到弹出对话框中提示用户输入信息。

```
<?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">
   <TableLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:stretchColumns="1">
      <TableRow>
         <TextView
```

```
        android:text="订单编号："
        android:id="@+id/TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

        <EditText
        android:text=""
        android:id="@+id/change_table_order_number_EditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    </TableRow>

    <TableRow>
        <TextView
        android:text="更换桌号："
        android:id="@+id/TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
        <EditText
        android:text=""
        android:id="@+id/change_table_no_EditText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    </TableRow>
  </TableLayout>
</LinearLayout>
```

② 在工程的“MainMenuActivity”类中定义一个 changeTable()方法，该方法首先实例化定义的布局文件，将其添加到 AlertDialog 中，在对话框的确定方法中获得订单编号和更换桌号，调用服务器 Servlet，更新数据，并显示更新结果。

```
    // 换台系统
    private void changeTable(){
        // 获得 LayoutInflater 实例
        LayoutInflater inflater = LayoutInflater.from(this);
        // 获得 LinearLayout 视图实例
        View v =inflater.inflate(R.layout.change_table, null);
        // 从 LinearLayout 中获得 EditText 实例
        final EditText et1 = (EditText) v.findViewById(R.id.change_table_order_number_
EditText);
        final EditText et2 = (EditText) v.findViewById(R.id.change_table_no_EditText);

        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setMessage(" 真的要换桌位吗？")
              .setCancelable(false)
              .setView(v)
              .setPositiveButton("确定", new DialogInterface.OnClickListener() {
                 public void onClick(DialogInterface dialog, int id) {
               // 获得订单号
               String orderId = et1.getText().toString();
```

```
                // 获得桌号
                    String tableId = et2.getText().toString();
                    // 查询参数
                    String queryString = "orderId="+orderId+"&tableId="+tableId;
                    // url
                    String  url  =  HttpUtil.BASE_URL+"servlet/ChangeTableServlet?"+
queryString;
                    // 查询返回结果
                    String result = HttpUtil.queryStringForPost(url);
                    // 显示结果
                    Toast.makeText(MainMenuActivity.this,result,Toast.LENGTH_LONG).
show();
                }
            })
            .setNegativeButton("取消", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });
        AlertDialog alert = builder.create();
        alert.show();
    }
```

③ 定义转台监听器，调用 changeTable 方法。

```
// 转台监听器
OnClickListener changeTableLinstener = new OnClickListener(){
    @Override
    public void onClick(View v) {
        changeTable();
    }
};
```

④ 主菜单 GridView 添加监听器。

```
case 2:
// 添加转台监听器
imageView.setOnClickListener(changeTableLinstener);
break;
```

程序运行结果如图 18.25 所示。

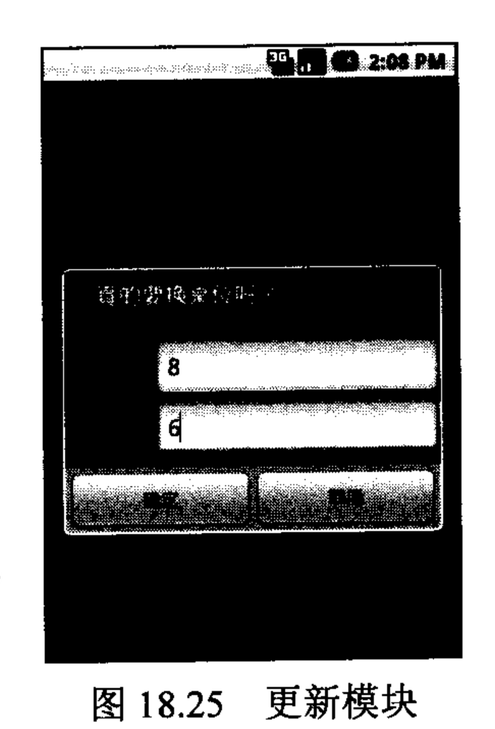


图 18.25　更新模块运行结果

18.4.13　转台模块服务器端实现

为了响应 Android 客户端的请求需要定义 Servlet，为了更新数据库需要定义 DAO 接口来操作数据库。具体步骤说明如下。

① 在工程的“com.amaker.dao”包中定义一个 ChangeTableDao 接口，在该接口中定义一个转台方法 changeTable()。

```
package com.amaker.dao;
// 转台 DAO 接口
```

```
public interface ChangeTableDao {
    // 转台方法
    public void changeTable(int orderId,int tableId);
}
```

② 在工程的“com.amaker.dao”包中定义一个 ChangeTableDaoImpl 类实现上述接口。该类需要更新订单表以及更换桌位的状态，为此本代码的实现使用了事务控制。

```
package com.amaker.dao.impl;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import com.amaker.dao.ChangeTableDao;
import com.amaker.util.DBUtil;
/**
 * @author 郭宏志
 * 完成转台功能
 */
public class ChangeTableDaoImpl implements ChangeTableDao {

    public void changeTable(int orderId, int tableId) {
        // 更新 SQL 语句

        String sql = " update TableTbl set flag = 0 where id = "+
          " (select tableId from OrderTbl  as ot where ot.id = ?)";
        String sql2 = " update OrderTbl set tableId = ? where id = ? ";
        String sql3 = " update TableTbl set flag = 1 where id = ?";

        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();

        try {
            conn.setAutoCommit(false);
            // 获得预定义语句
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, orderId);
            // 更新桌位状态
            pstmt.executeUpdate();

            pstmt = conn.prepareStatement(sql2);

            // 设置参数
            pstmt.setInt(1, tableId);
            pstmt.setInt(2, orderId);
            // 更新订单表
            pstmt.executeUpdate();

            pstmt = conn.prepareStatement(sql3);
```

```
            pstmt.setInt(1, tableId);
            // 更新桌位状态
            pstmt.executeUpdate();

            conn.commit();
        } catch (SQLException e) {
            e.printStackTrace();
            try {
                conn.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        } finally {
            util.closeConn(conn);
        }
    }
}
```

③ 在工程的“com.amaker.servlet”包中定义一个名称为“ChangeTableServlet”的 Servlet 类，该类响应客户端请求，调用 DAO 更新数据库数据，返回信息给客户端。

```
package com.amaker.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.amaker.dao.ChangeTableDao;
import com.amaker.dao.impl.ChangeTableDaoImpl;
/**
 * @author 郭宏志
 * 完成转台功能
 */
public class ChangeTableServlet extends HttpServlet {
    // 构造方法
    public ChangeTableServlet() {
        super();
    }
    // 销毁方法
    public void destroy() {
        super.destroy();
    }
    // 响应 Get 请求
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
```

```
        // 获得请求参数
        String orderId = request.getParameter("orderId");
        String tableId = request.getParameter("tableId");
        // 实例化 ChangeTableDao
        ChangeTableDao dao = new ChangeTableDaoImpl();
        // 调用转台方法
        dao.changeTable(Integer.parseInt(orderId), Integer.parseInt(tableId));
        // 返回客户端信息
        out.println("更换成功！");
        out.flush();
        out.close();
    }
    // 响应 Post 请求
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request,response);
    }
    // 初始化方法
    public void init() throws ServletException {
    }
}
```

18.4.14　并台模块 Android 客户端实现

并台模块要完成的任务是：当两桌顾客下单之后，在就餐过程中提出一起就餐。这一过程需要将保存在数据库中的数据做统一更新。具体更新的内容有：删除被合并桌位的订单、将被合并桌位客人数加到合并桌位、更新订单详细列表信息、更新桌位状态等。具体实现步骤说明如下。

① 并台是通过弹出对话框的方式来提示用户的，为此在对话框中添加自定义视图，提示用户选择要更换的桌位。在工程的 res\layout\目录下创建一个名称为 union_table.xml 的布局文件，该布局文件外层使用 LinearLayout 布局，内层嵌套 TableLayout，TableLayout 中有两行，每行通过一个 Spinner 提示用户选择桌号。

```
<?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"
    >

    <TableLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">

        <TableRow>
            <TextView
            android:text="被并桌位："
```

```
        android:id="@+id/TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <Spinner
        android:id="@+id/union_table_Spinner1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    </TableRow>


    <TableRow>
        <TextView
        android:text="目标桌位："
        android:id="@+id/TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <Spinner
        android:id="@+id/union_table_Spinner2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>

    </TableRow>

  </TableLayout>
</LinearLayout>
```

② 在 MainMenuActivity 类中创建一个名称为 unionTable()的方法，该方法定义了一个对话框提示用户选择合并桌号，在对话框中添加了自定义视图，访问服务器获得桌位信息并绑定到 Spinner，然后在确定方法中提交合并请求。

```
// 并台系统
private void unionTable(){
    // 实例化 LayoutInflater
    LayoutInflater inflater = LayoutInflater.from(this);
    // 获得自定义视图
    View v =inflater.inflate(R.layout.union_table, null);
    // 获得 Spinner
    final Spinner spinner1 = (Spinner) v.findViewById(R.id.union_table_Spinner1);
    final Spinner spinner2 = (Spinner) v.findViewById(R.id.union_table_Spinner2);
    // 访问服务器的 URL
    String urlStr = HttpUtil.BASE_URL + "servlet/UnionTableServlet";
    try {
        // 实例化 URL
        URL url = new URL(urlStr);
        // URLConnection 实例
        URLConnection conn = url.openConnection();
        // 获得输入流
        InputStream in = conn.getInputStream();
        // 获得 DocumentBuilderFactory 对象
        DocumentBuilderFactory factory = DocumentBuilderFactory
```

```
            .newInstance();
        // 获得 DocumentBuilder 对象
        DocumentBuilder builder = factory.newDocumentBuilder();
        // 获得 Document 对象
        Document doc = builder.parse(in);
        // 获得节点列表
        NodeList nl = doc.getElementsByTagName("table");
        // Spinner 数据
        List items = new ArrayList();

        // 获得 XML 数据
        for (int i = 0; i < nl.getLength(); i++) {
            // 桌位编号
            String id = doc.getElementsByTagName("id")
                    .item(i).getFirstChild().getNodeValue();
            // 桌号
            int num = Integer.parseInt(doc.getElementsByTagName("num")
                    .item(i).getFirstChild().getNodeValue());
            Map data = new HashMap();
            data.put("id", id);
            items.add(data);
        }

        // 获得 SpinnerAdapter
        SpinnerAdapter as = new
        SimpleAdapter(this, items,
                android.R.layout.simple_spinner_item,
                new String[] { "id" }, new int[] { android.R.id.text1 });

        // 绑定数据
        spinner1.setAdapter(as);
        spinner2.setAdapter(as);
    } catch (Exception e) {
        e.printStackTrace();
    }
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setMessage(" 真的要并桌吗？ ")
          .setCancelable(false)
          .setView(v)
          .setPositiveButton("确定", new DialogInterface.OnClickListener() {
             public void onClick(DialogInterface dialog, int id) {
              TextView tv1 = (TextView) spinner1.getSelectedView();
              TextView tv2 = (TextView) spinner2.getSelectedView();

              String tableId1 = tv1.getText().toString();
              String tableId2 = tv2.getText().toString();
                // 查询参数
                    String queryString = "tableId1="+tableId1+"&tableId2="+tableId2;
                    // url
                    String url = HttpUtil.BASE_URL+"servlet/UnionTableServlet2?"+
queryString;
                    // 查询返回结果
```

```
                    String result =  HttpUtil.queryStringForPost(url);
                }
            })
            .setNegativeButton("取消", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.cancel();
                }
            });
        AlertDialog alert = builder.create();
        alert.show();
    }
```

③ 创建一个并台监听器，调用 unionTable 方法。

```
// 并台监听器
OnClickListener unionTableLinstener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        unionTable();
    }
};
```

④ 为 GridView 添加单击监听器。

```
case 1:
// 并台监听器
imageView.setOnClickListener(unionTableLinstener);
break;
```

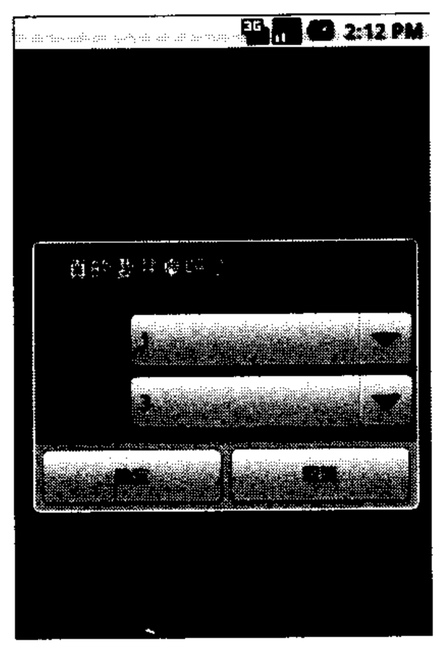


图 18.26　并台模块运行结果

程序运行结果如图 18.26 所示。

18.4.15　并台模块服务器端实现

并台模块的服务器端实现主要是定义 Servlet 响应 Android 客户端请求，定义 DAO 更新合并后的数据。具体步骤说明如下。

① 在工程的“com.amaker.dao”包中创建一个 UnionTableDao 接口。该接口中有两个方法，分别用来获得桌位列表和更新合并后的数据。

```
package com.amaker.dao;
import java.util.List;
// 并台功能接口
public interface UnionTableDao {
    // 获得桌位列表
    public  List getTableList();
    // 更新合并后的数据
    public void updateStatus(int tableId1,int tableId2);
}
```

② 在工程的“com.amaker.dao.impl”包中定义一个 UnionTableDaoImpl 类实现 UnionTableDao。

```
package com.amaker.dao.impl;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import com.amaker.dao.UnionTableDao;
import com.amaker.entity.UnionTable;
import com.amaker.util.DBUtil;

/**
 * @author 郭宏志
 * 实现并台功能
 */
public class UnionTableDaoImpl implements UnionTableDao {

   public List getTableList() {
       // 查询SQL语句
       String sql =" select id,num from TableTbl where flag = 1 ";
       // 数据库连接工具类
       DBUtil util = new DBUtil();
       // 获得连接
       Connection conn = util.openConnection();
       try {
           // 获得预定义语句
           Statement pstmt = conn.createStatement();
           // 执行查询
           ResultSet rs = pstmt.executeQuery(sql);
           // 判断订单详细
           List list = new ArrayList();
           while (rs.next()) {
               // 获得菜单信息

               int id = rs.getInt(1);
               int num = rs.getInt(2);

               UnionTable ut = new UnionTable();
               ut.setId(id);
               ut.setNum(num);

               list.add(ut);
           }
           return list;
       } catch (SQLException e) {
           e.printStackTrace();
       } finally {
           util.closeConn(conn);
       }
       return null;
   }
```

```
    public void updateStatus(int tableId1,int tableId2) {
        // 数据库连接工具类
        DBUtil util = new DBUtil();
        // 获得连接
        Connection conn = util.openConnection();
        CallableStatement cstmt;
        try {
            cstmt = conn.prepareCall("{call new_proc(?,?)}");
            cstmt.setInt(1, tableId1);
            cstmt.setInt(2, tableId2);
            cstmt.execute();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

③ ② 中的第二个方法是通过调用存储过程来实现的，存储过程的代码如下所示。

```
CREATE PROCEDURE `new_proc`(IN tableId1 INTEGER(11), IN tableId2 INTEGER(11))
    NOT DETERMINISTIC
    SQL SECURITY DEFINER
    COMMENT ''
BEGIN
     declare perNum,oId1,oId2 integer;

     select personNum  into perNum from OrderTbl as ot where
     ot.tableId =tableId2
     and ot.isPay = 0;

     select id into oId1 from OrderTbl as ot where
     ot.tableId = tableId1
     and ot.isPay = 0;

     select id into oId2 from OrderTbl as ot where
     ot.tableId = tableId2
     and ot.isPay = 0;

     update OrderTbl
     set personNum = personNum + perNum
     where id = oId2;

     update OrderDetailTbl set orderId = oId2 where orderId = oId1;

     update TableTbl set flag = 0 where id = tableId1;

     delete from OrderTbl where id = oId1;
END;
```

④ 定义 Servlet 响应 Android 客户端请求，调用 DAO 更新合并数据。这个过程比较简单，且限于篇幅，这里将不再给出代码实现。详细内容请参考光盘代码实现。

至此，Android 无线点餐系统已全部讲解完毕。