

The Android Developer's Cookbook

Building Applications with the Android SDK

Android开发秘籍

[美] James Steele 著
Nelson To

李青 王瑜 赵丞兵 译



NLIC 2970683662

- Android开发专家倾情奉献
- 大量秘诀代码尽收囊中
- 秘籍在手，即刻开始移动开发



人民邮电出版社
POSTS & TELECOM PRESS

“本书堪称Android开发应用的一部杰作，不论是组织结构还是阐述方式都独具特色。强烈推荐给所有的人，不管你是否刚开始Android开发，都能从本书汲取全新的营养，运用到自己的研发中。”

——Kofi Addaquay，著名跨国交互设计公司Scripton CTO

“有关Android开发所需的知识一应俱全，还有各种Android应用的详尽代码示例！想要真正学习和提高Android开发技能，请从本书开始。赶紧去抢购吧，绝对物超所值！”

——Klaus S. Villaca，知名移动开发人士

“本书包括大量绝妙的代码示例，可以直接应用到自己的研发项目中，会为我们节省大量时间。对所有人来说，本书是学习Android开发应用的不二之选！”

——Alina，Android应用和游戏开发人士

The Android Developer's Cookbook

Building Applications with the Android SDK

Android开发秘籍

Android自2007年年末由开放手机联盟（Open Handset Alliance）发布以来，得到了全球众多厂商和移动运营商的支持，并迅速成为智能手机的主流操作系统，其潜在的巨大经济效益也展露无遗。如今，Android平台开发可谓如火如荼、炙手可热。

本书通过大量代码秘诀全面详尽地讲述了Android开发技术。从activity和intent基础知识开始，到基于位置的服务、Android安全开发和性能优化等高级应用，本书涵盖了Android开发的各类知识。通过本书的学习，开发人员能够：

- ◆ 从头开始编写Android应用；
- ◆ 编写跨多个Android版本的程序；
- ◆ 使用Android提供的各种API；
- ◆ 博览大量代码，迅速应用到自己的程序中；
- ◆ 学会在Android中运用多种方法完成同种任务；
- ◆ 理解Android编程的独特魅力。

Addison
Wesley

www.informit.com

图灵网站：www.turingbook.com 热线：(010)51095186转604

反馈/投稿/推荐信箱：contact@turingbook.com

有奖勘误：debug@turingbook.com

分类建议 计算机/移动开发

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-25718-5



9 787115 257185 >

ISBN 978-7-115-25718-5

定价：49.00元

TURING 图灵程序设计丛书 移动开发系列

The Android Developer's Cookbook

Building Applications with the Android SDK

Android开发秘籍

人民邮电出版社

北京

图书在版编目(CIP)数据

Android开发秘籍 / (美) 斯蒂尔 (Steele, J.),
(美) 图 (To, N.) 著; 李青, 王瑜, 赵丞兵译. -- 北京:
人民邮电出版社, 2011.8

(图灵程序设计丛书)

书名原文: The Android Developer's Cookbook:
Building Applications with the Android SDK
ISBN 978-7-115-25718-5

I. ①A… II. ①斯… ②图… ③李… ④王… ⑤赵…
III. ①移动终端—应用程序—程序设计 IV.
①TN929.53

中国版本图书馆CIP数据核字(2011)第129437号

内 容 提 要

作为 Google 开发的全新开源手机平台, Android 发展如火如荼。本书通过大量代码秘诀全面详尽地讲述了 Android 开发技术。从 activity 和 intent 基础知识开始, 到线程、服务、broadcast receiver 以及 alert 警告框, 再到用户界面布局、界面事件、多媒体技术、硬件接口、网络通信、数据存储方法、基于位置的服务、Android 高级开发技术和调试, 书中贯穿了经 Android 设备或者模拟器测试的可用范例, 将功能丰富、结构复杂的 Android 应用程序清晰完美地展现给开发人员。

对于那些有志于 Android 应用开发的人员来说, 本书是难得的参考读物。

图灵程序设计丛书 Android开发秘籍

- ◆ 著 [美] James Steele Nelson To
译 李 青 王 瑜 赵丞兵
责任编辑 朱 巍
执行编辑 刘美英
- ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
- ◆ 开本: 800×1000 1/16
印张: 17.25
字数: 408千字 2011年8月第1版
印数: 1-4 000册 2011年8月北京第1次印刷
著作权合同登记号 图字: 01-2010-8067号
ISBN 978-7-115-25718-5

定价: 49.00元

读者服务热线: (010)51095186转604 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

Authorized translation from the English language edition, entitled *The Android Developer's Cookbook: Building Applications with the Android SDK*, 978-0-321-74123-3 by James Steele and Nelson To, published by Pearson Education, Inc., publishing as Addison Wesley, Copyright © 2011 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD. and POSTS & TELECOM PRESS Copyright © 2011.

本书中文简体字版由 Pearson Education Asia Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。版权所有，侵权必究。



前言

Android 是发展最为迅速的移动操作系统 (OS)。以 Android 为核心的整个生态系统也在迅速成长, 仅去年一年就推出 30 多种智能手机, 而平均每月新增 1 万多个应用程序 (APP)。其设备功能多样, 可供选择的移动运营商众多, 足以让所有人动心。

上网本是安装 Android 的天然平台, 但 Android 的强劲发展势头使之进一步渗透到电视机甚至汽车工业。许多世界知名的大企业, 从银行到快餐连锁店和航空公司, 都拥有了自己的 Android 应用, 并提供兼容服务。Android 开发人员获得了更多机会, 因为其应用程序可以接触到比以往任何时候都要多的用户, 增加了开发程序的回报。

为什么要写本书^①

Android 操作系统简单易学, 而且谷歌提供了许多库, 方便大家实现功能丰富、结构复杂的应用程序。唯一美中不足的是缺少清晰详尽的文档, Android 开发者社区的许多人都提到了这一点。Android 的开源意味着任何人都可以深入研究并对一些文件做逆向工程。许多开发者论坛都使用这种方法推出了优良的范例。但我们仍然需要一本书通过一致的体例来讨论操作系统的各个方面。

一个清晰的可用范例比得上一万字的文档。开发人员在面对问题的时候, 通常喜欢采用类似极限编程的方法, 也就是说, 他们找到接近解决方案的可用代码范例, 经过修改或扩展来满足自己的需求。学习这些范例也是一种了解编码风格的有效方法, 开发者可按类似风格写出代码的其他部分。

本书提供了许多完整的秘诀, 以满足读者此种需要。介绍每个秘诀的同时, 我们也逐一讲解 Android 操作系统的主要概念。

目标读者

编写 Android 应用程序的开发人员将是本书的最大获益者。我假设读者了解基本的 Java 和 Eclipse 开发环境, 但对于本书大部分内容这点并非必要。Java 是一种模块化的语言, 大部分 (甚至全部) 的秘诀范例经过些许修改就可以整合到读者自己的 Android 工程中。我们编写每个主题时都考虑到可以将其作为 Android 课程的补充材料。

^① 秘籍, 此处的原文为 cookbook (食谱), 引申为秘籍的意思, 后文中的 recipe 是秘诀的意思。即这本秘籍由大量秘诀组成。——译者注

如何使用本书

总体上，本书中的代码秘诀内容丰富完整，包含了在 Android 设备上运行应用程序所有必要的信息。第 1 章和第 2 章从整体上介绍了 Android 的使用，当然，你可以跳过这一部分，从更重要的地方开始看起。

这本书是作为参考书来写的，书中主要通过范例讲述 Android 开发知识，这些范例实现了有意思的秘诀，会让你受益匪浅。秘诀中介绍的主要技术均在标题中体现。此外，每个秘诀在必要时还介绍了其他相关技术。

读完本书以后，开发人员应该能够：

- 从头开始编写 Android 应用程序；
- 编写适用于多个 Android 版本的程序；
- 使用 Android 提供的各种应用编程接口（API）；
- 博览大量代码，迅速运用到自己的程序中；
- 学会在 Android 中用多种方法完成同样任务，体会各种方法的优劣；
- 理解 Android 编程的独特魅力。

本书结构

第 1 章将介绍 Android 开发的方方面面，但不涉及代码层面。这是唯一不包括秘诀的一章，但提供了有用的背景知识。第 2 章概要介绍了 4 个 Android 组件，以及 Android 工程是如何组织的。本章着重介绍了作为应用程序主要构成的 activity。第 3 章介绍了线程、服务、接收器等后台服务，以及作为这些后台任务的通知方法使用的 alert 警告框。第 4 章涵盖了用户界面屏幕布局和视图。第 5 章涉及用户触发的事件，如触摸事件和手势。第 6 章讲的是多媒体操作、录制以及音频和视频播放。第 7 章介绍了 Android 设备的硬件 API 及其使用方法。第 8 章讨论了 Android 设备和外界应用的交互，包括 SMS、网页浏览和社交网络。第 9 章涵盖了 Android 中使用的各种数据存储技术，包括 SQLite。第 10 章着重介绍了通过 GPS 等不同的方法获取位置信息，以及使用诸如谷歌地图 API 的相关服务。第 11 章介绍一些 Android 的高级技巧，包括自定义视图、使用原生代码获得更快的处理速度，以及使用 Android 备份管理等。最后，第 12 章讲述了对整个开发周期都非常有用的测试和调试框架。

补充参考资料

网上有许多 Android 的在线参考信息。以下是一些必看的经典网站。

- Android 源代码：<http://source.android.com/>
- Android 开发者主页：<http://developer.android.com/>
- Android 开发者论坛：<http://www.svcAndroid.com/>
- 开放源码目录：<http://osdir.com/>
- “栈溢出”论坛（Stack Overflow Discussion Threads）：<http://stackoverflow.com/>
- Android 开发者讲坛（Talk Android Developer Forums）：<http://www.talkandroid.com/androidforums/>

目 录

第 1 章 Android 概述.....	1	1.8.3 脱颖而出.....	15
1.1 Android 演化史.....	1	1.8.4 为应用程序收费.....	15
1.2 Android 的两面性.....	2	1.8.5 管理评论和更新.....	16
1.3 运行 Android 的设备.....	2	1.8.6 Android Market 的候补之选.....	17
1.3.1 HTC 系列机型.....	4		
1.3.2 摩托罗拉系列机型.....	4	第 2 章 应用程序基础知识:	
1.3.3 三星系列机型.....	4	activity 和 intent.....	18
1.3.4 平板电脑.....	5	2.1 Android 应用程序预览.....	18
1.3.5 其他设备.....	5	2.1.1 秘诀 1: 创建工程并新建	
1.4 Android 设备的硬件差异.....	5	activity.....	19
1.4.1 屏幕.....	5	2.1.2 工程目录结构及自动生成内容.....	20
1.4.2 用户输入方式.....	6	2.1.3 Android 包和 Manifest 清单文件.....	22
1.4.3 传感器.....	6	2.1.4 重命名应用程序中的部分文件.....	23
1.5 Android 的特点.....	8	2.2 Activity 的生命周期.....	23
1.5.1 多进程和应用程序微件.....	8	2.2.1 秘诀 2: 使用其他的生命	
1.5.2 触摸、手势和多点触控.....	8	周期方法.....	24
1.5.3 硬键盘和软键盘.....	8	2.2.2 秘诀 3: 强制执行单任务模式.....	26
1.6 Android 开发.....	8	2.2.3 秘诀 4: 强制屏幕方向.....	26
1.6.1 如何使用本书中的秘诀.....	8	2.2.4 秘诀 5: 保存和恢复 activity	
1.6.2 好好设计应用程序.....	9	信息.....	27
1.6.3 保持向前兼容.....	9	2.3 多个 activity.....	28
1.6.4 健壮性.....	10	2.3.1 秘诀 6: 使用按钮和文本框.....	28
1.7 软件开发工具包.....	10	2.3.2 秘诀 7: 通过事件启动	
1.7.1 安装与更新.....	10	另外一个 activity.....	29
1.7.2 软件特性和 API 级别.....	11	2.3.3 秘诀 8: 将语音转换成文本	
1.7.3 利用模拟器或真机调试程序.....	12	并启动 activity 显示结果.....	32
1.7.4 使用 Android 调试桥.....	13	2.3.4 秘诀 9: 实现选择列表.....	34
1.7.5 签名和发布应用.....	14	2.3.5 秘诀 10: 使用隐式 intent 创建	
1.8 Android Market.....	14	activity.....	35
1.8.1 最终用户许可协议.....	14	2.3.6 秘诀 11: 在 activity 间传递	
1.8.2 提升应用程序的曝光率.....	15	基本数据类型.....	37

第3章 线程、服务、receiver 以及

alert 对话框..... 40

- 3.1 线程..... 40
 - 3.1.1 秘诀 12: 启动一个辅助线程..... 40
 - 3.1.2 秘诀 13: 创建实现 runnable
接口的 activity..... 44
 - 3.1.3 秘诀 14: 设置线程优先级..... 45
 - 3.1.4 秘诀 15: 取消线程..... 45
 - 3.1.5 秘诀 16: 在两个应用程序之间
共享线程..... 46
- 3.2 线程之间的消息机制: handler..... 46
 - 3.2.1 秘诀 17: 从主线程调度
runnable 任务..... 46
 - 3.2.2 秘诀 18: 使用倒数计时器..... 49
 - 3.2.3 秘诀 19: 处理耗时的初始化
工作..... 50
- 3.3 服务..... 51
- 3.4 添加 broadcast receiver..... 56
- 3.5 应用微件..... 58
- 3.6 alert 对话框..... 60
 - 3.6.1 秘诀 23: 使用 Toast 在屏幕上
显示简短消息..... 61
 - 3.6.2 秘诀 24: 使用 alert 对话框..... 61
 - 3.6.3 秘诀 25: 在状态栏中显示通知..... 62

第4章 用户界面布局..... 65

- 4.1 资源目录及其基本属性..... 65
- 4.2 view 和 ViewGroup..... 67
 - 4.2.1 秘诀 27: 利用 Eclipse 编辑器
生成布局..... 68
 - 4.2.2 秘诀 28: 控制 UI 元素的
宽度和高度..... 71
 - 4.2.3 秘诀 29: 设置相对布局和
布局 ID..... 73
 - 4.2.4 秘诀 30: 通过编程声明布局..... 74
 - 4.2.5 秘诀 31: 使用独立线程更新
布局..... 75
- 4.3 文本操作..... 78
 - 4.3.1 秘诀 32: 设置和更改文本属性..... 79
 - 4.3.2 秘诀 33: 提供文本输入..... 81

- 4.3.3 秘诀 34: 创建表单..... 82
- 4.4 其他控件: 从按钮到拖动条..... 83
 - 4.4.1 秘诀 35: 在表格布局中使用
图像按钮..... 83
 - 4.4.2 秘诀 36: 使用复选框和开关
按钮..... 86
 - 4.4.3 秘诀 37: 使用单选按钮..... 90
 - 4.4.4 秘诀 38: 创建下拉菜单..... 90
 - 4.4.5 秘诀 39: 使用进度条..... 92
 - 4.4.6 秘诀 40: 使用拖动条..... 94

第5章 用户界面事件..... 97

- 5.1 事件处理器和事件监听器..... 97
 - 5.1.1 秘诀 41: 截取物理按键事件..... 97
 - 5.1.2 秘诀 42: 创建菜单..... 100
 - 5.1.3 秘诀 43: 在 XML 文件中
定义菜单..... 104
 - 5.1.4 秘诀 44: 使用搜索键..... 105
 - 5.1.5 秘诀 45: 响应触摸事件..... 107
 - 5.1.6 秘诀 46: 监听滑动手势..... 109
 - 5.1.7 秘诀 47: 使用多点触控..... 110
- 5.2 高级用户界面库..... 113
 - 5.2.1 秘诀 48: 使用手势..... 114
 - 5.2.2 秘诀 49: 绘制 3D 图像..... 117

第6章 多媒体技术..... 122

- 6.1 图像..... 123
- 6.2 音频..... 128
 - 6.2.1 秘诀 51: 选取和播放音频文件..... 128
 - 6.2.2 秘诀 52: 录制音频文件..... 131
 - 6.2.3 秘诀 53: 处理原始音频..... 132
 - 6.2.4 秘诀 54: 有效使用音频资源..... 136
 - 6.2.5 秘诀 55: 添加媒体资源并
更新路径..... 137
- 6.3 视频..... 138

第7章 硬件接口..... 140

- 7.1 照相机..... 140
- 7.2 其他传感器..... 145
 - 7.2.1 秘诀 57: 获取设备旋转姿态..... 146

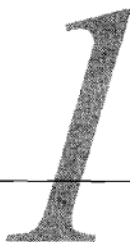
7.2.2 秘诀 58: 使用温度传感器和光传感器	149	9.2.3 秘诀 79: 创建个人日记	200
7.3 电话	150	9.3 内容提供者	204
7.3.1 秘诀 59: 使用电话管理器	150	9.4 保存和载入文件	209
7.3.2 秘诀 60: 监听电话状态	152		
7.3.3 秘诀 61: 拨打电话号码	154	第 10 章 基于位置的服务	210
7.4 蓝牙	154	10.1 位置服务入门	210
7.4.1 秘诀 62: 打开蓝牙	155	10.1.1 秘诀 81: 获取最新位置	212
7.4.2 秘诀 63: 搜索蓝牙设备	155	10.1.2 秘诀 82: 在位置改变时更新信息	212
7.4.3 秘诀 64: 与已绑定的蓝牙设备配对	156	10.1.3 秘诀 83: 列出所有可用的提供者	214
7.4.4 秘诀 65: 打开蓝牙套接字	156	10.1.4 秘诀 84: 将位置解析为地址 (反向地理编码)	216
7.4.5 秘诀 66: 使用设备振动功能	159	10.1.5 秘诀 85: 将地址解析为位置 (地理编码)	218
7.4.6 秘诀 67: 访问无线网络	159	10.2 使用谷歌地图	220
第 8 章 网络通信	161	10.2.1 秘诀 86: 在应用程序中添加谷歌地图	222
8.1 使用短信息	161	10.2.2 秘诀 87: 在地图上添加标记	224
8.2 使用 Web 内容	169	10.2.3 秘诀 88: 在地图上添加视图	228
8.2.1 秘诀 69: 定制 Web 浏览器	169	10.2.4 秘诀 89: 在地图上标记设备的当前位置	230
8.2.2 秘诀 70: 使用 HTTP GET 请求	170	10.2.5 秘诀 90: 设置临近警告	231
8.2.3 秘诀 71: 使用 HTTP POST 请求	174		
8.3 社交网络	174	第 11 章 Android 高级开发技术	232
第 9 章 数据存储方法	184	11.1 Android 的自定义视图	232
9.1 shared preferences	184	11.2 Android 的原生组件	238
9.1.1 秘诀 73: 创建和检索 shared preferences	184	11.3 Android 的安全机制	241
9.1.2 秘诀 74: 使用 preferences 框架	185	11.4 Android 的进程间通信	242
9.1.3 秘诀 75: 基于 Stored Data 改变用户界面	187	11.5 Android 的备份管理器	247
9.1.4 秘诀 76: 添加最终用户许可协议	190	11.5.1 秘诀 95: 备份运行时数据	247
9.2 SQLite 数据库	194	11.5.2 秘诀 96: 备份文件到云端	248
9.2.1 秘诀 77: 创建一个独立的数据库包	194	11.5.3 秘诀 97: 触发备份与还原操作	249
9.2.2 秘诀 78: 使用独立的数据库包	197	11.6 Android 的动画功能	250
		第 12 章 调试	255
		12.1 Eclipse 内置的调试工具	255
		12.1.1 秘诀 99: 设置运行配置	255
		12.1.2 秘诀 100: 使用 DDMS	256

12.1.3	秘诀 101: 断点调试	257	12.2.3	秘诀 104: 使用 Hierarchy Viewer 工具	261
12.2	Android SDK 中的调试工具	258	12.2.4	秘诀 105: 使用 TraceView 工具	262
12.2.1	秘诀 102: 使用 Android Debug Bridge 工具	258	12.3	Android 系统调试工具	264
12.2.2	秘诀 103: 使用 LogCat 工具	259			



第1章

Android概述



Android操作系统自2007年年末由开放手机联盟（Open Handset Alliance）发布以来，已经取得了长足的进步。为嵌入式系统设计开源操作系统这一想法由来已久，但谷歌的积极支持无疑推动了Android在短短数年时间就跻身业界前列。

各国众多移动运营商在不同的通信协议下都推出了Android手机。Android操作系统还被应用在其他嵌入式设备中，如平板电脑、上网本、电视机、机顶盒，甚至汽车也开始使用它了。

本章将介绍Android开发的各种一般性问题，了解这些问题对于开发者非常有益。它是创建Android应用程序的基础知识，并且为本书其余部分介绍的编程秘诀提供了背景知识。

1.1 Android 演化史

谷歌公司看到互联网使用和移动设备搜索的巨大增长潜力，于2005年收购了Android公司，并主攻移动设备平台的开发。苹果公司于2007年推出了iPhone手机，带来了多触点和开放的应用程序市场等一些具有开创性的想法。Android迅速跟进，将这些功能囊括其中，还提供一些特有的功能，例如开发者对系统有更多控制能力，实现多任务功能等。此外，Android集成了企业级的需求，如支持交流、远程擦除（wipe）和虚拟专用网（VPN），以谋求赢得企业市场。而目前在该领域中，RIM公司凭借其黑莓机型发展得很好。

对多种设备的支持和快速应变使得Android扩大了其用户规模，但这给开发者带来了挑战。应用程序需要支持五花八门的屏幕尺寸、分辨率、键盘、硬件传感器、OS版本、无线数据传输速率和系统配置。每项处理不当都可能导致不可预知的诡异表现，但又不可能在所有环境下都做应用程序测试。

因此，Android的设计力求在跨平台时可以获得一致的体验。通过将硬件差异抽象处理，Android操作系统试图将应用和具体设备差异隔离开，同时在需要时还可以灵活调整。应用程序要兼容未来的变化，以适应新的硬件平台和操作系统，这也是要事先考虑的。只有开发者意识到这种系统化的方法，才能做到这一点。Android提供的通用API和如何确保设备和操作系统的兼容性贯穿本书的讨论主题。

和任何嵌入式平台相同，我们必须进行全面的测试。谷歌通过Android开发工具（ADT，该工具是Eclipse的插件，也可作为独立的工具使用）向第三方开发者提供了各种形式的帮助，包括

实时日志功能、可以运行原生ARM代码的仿真模拟器，另外，用户可以向Android Market应用的开发者提交现场错误报告。

1.2 Android 的两面性

Android有一些有趣的两面性。提前知晓这些内容有助于了解Android是什么及不是什么。

Android是一个由Linux内核提供核心系统服务的嵌入式操作系统，但它不是嵌入式的Linux。例如，它不支持标准的Linux工具，如X-Windows和GNU C库。我们使用Java框架编写Android的应用程序，但它不是Java语言，并不支持Swing等标准Java库。其他Java库，如Timer等，也不支持，它们已被Android自己的库替代，这些库已针对资源有限的嵌入式环境进行了优化。

Android的操作系统开放源代码，这意味着开发人员可以查看和使用任何系统的源代码，包括射频协议栈（radio stack）。这些源代码是大家了解Android代码运行原理的第一手资料，在文档缺乏的情况下可以作为参考。这也意味着开发人员可以遵照系统核心程序那样的方式使用系统，可以用他们自己的组件替换系统组件。不过，Android设备也包含一些开发者无法访问的专有软件，如全球定位系统导航。

Android OS的最后一个两面性特点是谷歌还支持Chrome OS。Android OS专为嵌入式平台构建，而Chrome OS基于云计算平台构建。然而，哪个操作系统才是基于云端的嵌入式设备的最好选择？上网本填补了智能手机和笔记本电脑之间的空白，两者都有可能成为其发展方向（实际上已经呈现了这样的趋势）。Android已经开始更多地利用云服务。这是否意味着Chrome OS的末路指日可待？谷歌同时也支持基于Web的市场，因此Chrome OS目前享有和Android同等的开发者支持。这或许表明未来的融合早已在筹划之中。

1.3 运行 Android 的设备

市场上目前有十多个制造商提供四十余种Android手机。此外，其他硬件也运行Android，如平板电脑和电视机。软件可以通过android.os.Build类获得目标设备的信息，例如：

```
if(android.os.Build.MODEL.equals("NexusOne")) { ... }
```

由于其操作系统的特质，Android支持的硬件具有一些共同的特性。Android OS由以下image文件组成：

- Bootloader——在设备启动时开始加载Boot image；
- Boot image——Kernel和RAMdisk；
- System image——Android操作系统平台和应用程序；
- Data image——断电后保存的用户数据；
- Recovery image——重建或更新系统所用的文件；
- Radio image——射频协议栈文件。

这些image文件存储在非易失性闪存中，因此在设备断电时仍然不会丢失。闪存使用起来相当于只读存储器（因此有人称之为ROM），但可以在需要时将其重写（例如，Android操作系统无

线升级)。

启动时,微处理器执行Bootloader来加载内核和RAMdisk到RAM中,以快速存取。然后,微处理器执行所需的指令、系统页面和数据镜像到RAM。Radio image由基带处理器处理,后者直接连接到射频硬件。

表1-1比较了早期和近期推出的智能手机机型。结果表明,不同设备的运算处理部分的硬件结构非常相似,都包括微处理器单元(MPU)、同步动态随机存取记忆体(SDRAM或简称RAM)以及闪存(简称ROM)等。屏幕大小通过像素(pixel)来衡量,但每英寸点数(dpi)不同的物理屏幕是不相同的。例如,HTC Magic手机采用 320×480 像素的3.2英寸屏幕,这相当于每英寸180像素,在Android手机中处于中等水平(平均水平为160 dpi)。所有的智能手机都配备了CMOS图像传感器的摄像头、蓝牙(BT)和Wi-Fi(802.11),当然规格各异。

表1-1 一些有代表性的Android智能手机。数据来源于
http://en.wikipedia.org/wiki/List_of_Android_devices和<http://pdadb.net/>

型 号	MPU	RAM/ROM	屏 幕	其他特性
HTC Dream/G1 (2008年10月)	528-MHz QCOM MSM7201A	192 MB/256 MB	TFT LCD 320×480 mdpi	GSM/UMTS侧滑式键盘、轨迹球、AGPS、 蓝牙2.0、802.11b/g、310万像素摄像头
三星Moment (2009年11月)	800-MHz ARM 1176 JZF-S	288 MB/512 MB	AMOLED 320×480 mdpi	CDMA/1xEV-DO, 侧滑式键盘(带背 光)、DPAD, 蓝牙2.0、802.11b/g、310万 像素摄像头, AGPS
摩托罗拉Miles- tone/Droid(2009年 11月)	550-MHz TI OMAP3430	256 MB/512 MB	TFT LCD 480×854 hdpi	GSM/UMTS或CDMA/1xEV-DO, 侧滑 式键盘、DPAD, 蓝牙2.1、802.11b/g、500 万像素摄像头, AGPS
Nexus One/HTC Passion(2010年1 月)	1-GHz QCOM Snapdragon	512 MB/512 MB	AMOLED 480×800 hdpi	GSM/UMTS轨迹球、双麦克风, 蓝牙 2.0、802.11a/b/g/n、500万像素摄像头, AGPS、地理标记功能
HTC Droid Incred- ible(2010年4月)	1-GHz QCOM Snapdragon	512 MB/512 MB	AMOLED 480×800 hdpi	CDMA/1xEV-DO, 蓝牙2.1、802.11a/b/g/n、 800万像素摄像头, AGPS、地理标记功能
HTC EVO 4G (2010年6月)	1-GHz QCOM Snapdragon	512 MB/1 GB	AMOLED 480×800 hdpi	CDMA/1xEV-DO/802.16e-2005, 蓝牙 2.1、802.11b/g、800万像素摄像头, 130 万像素前置摄像头, AGPS
摩托罗拉Droid X(2010年6月)	1-GHz TI OMAP3630	512 MB/8 GB	TFT LCD 480×854 hdpi	CDMA/1xEV-DO、调频收音机, 蓝牙 2.1、802.11b/g/n、800万像素摄像头, AGPS、地理标记功能
索爱 Xperia X10a(2010年6月)	1-GHz QCOM Snapdragon	256 MB/1 GB	TFT LCD 480×854 hdpi	GSM/UMTS、调频收音机, 蓝牙2.1、 802.11b/g、800万像素摄像头, AGPS、地 理标记功能
三星 Galaxy S Pro(2010年8月)	1-GHz 三星 Hummingbird	512 MB/2 GB	AMOLED 480×800 hdpi	CDMA/1xEV-DO、802.16、调频收音机, 侧滑式键盘, 蓝牙3.0、802.11b/g/n、500 万像素摄像头, 30万像素前置摄像头、 AGPS
宏碁 Stream/ Liquid(2010年9 月)	1-GHz QCOM Snapdragon	512 MB/512 MB	AMOLED 480×800 hdpi	GSM/UMTS、调频收音机, 蓝牙2.1、 802.11b/g/n、500万像素摄像头, AGPS、 地理标记功能

除了在容量和性能上有所改善外,较新型号的另一个主要特色是附加功能。有些设备提供了4G网络,有的添加了调频收音机,有的具有侧滑式键盘,还有的配备了前置摄像头。了解设备的这些差异有助于开发人员做出优秀的应用。除了那些内置硬件外,每个Android设备都有安全数字(SD)卡插槽。SD卡可提供额外的存储空间,用来存储多媒体等应用程序数据。然而,在Andorid 2.2版以前,应用程序本身只能存储在内部ROM中。

1.3.1 HTC 系列机型

HTC(宏达)是一家成立于1997年的台湾公司。HTC Dream(也称为G1,G指代谷歌)是第一个运行Android的商用硬件。该机型于2008年10月发布。从那时开始到现在,HTC已推出了十多款运行Android的手机,包括谷歌公司的Nexus One。

Nexus One是第一批使用1 GHz微处理器的Android设备,这种微处理器是高通公司(Qualcomm)的Snapdragon平台。Snapdragon使用了高通自己的内核,而非ARM内核,它包含有720 p高清晰度视频解码电路。在此之后的大部分智能手机都采用了1 GHz的微处理器。Nexus One的特别之处还在于使用两个麦克风削减通话时的背景噪音,以及配备了背光轨迹球,可根据通知显示不同颜色的灯光。

HTC还于2010年4月发布了Droid Incredible机型。如表1-1所示,它和Nexus One类似,但基于CDMA而不是GSM射频硬件,并具有更高像素的摄像头。在2010年6月发布的HTC EVO 4G是第一款支持WiMAX(802.16e-2005标准)的商用手机。

1.3.2 摩托罗拉系列机型

20世纪80年代摩托罗拉制造了第一部手机,并在手机市场上取得多方面的成功。它的无线通信部门对于发展方向摇摆不定,直到最近才把重点放在Android上。摩托罗拉Droid的CDMA版本(它的GSM版本就是闻名全球的Milestone^①)于2009年11月发布,很多人也确实认为这是Android发展的一个重要的里程碑。Droid手机的影响力很明显,访问Android Market的相当多的手机都是Droid手机。

此外,摩托罗拉已经推出了近十款其他Android手机。摩托罗拉Droid X手机具有和HTC Droid Incredible相似的功能,包括高清视频拍摄功能。

1.3.3 三星系列机型

三星一直是移动市场的重要力量,目前,已经开始发展自己的Android设备。三星于2009年11月推出了三星 Moment手机,但不具备多点触摸硬件功能,不能升级到Android 2.1以上版本。在特定市场销售的配备了移动电视天线的定制版本可移动接收ATSC信号。

三星Galaxy S是三星对于iPhone的回应。众所周知,iPhone 3G和3GS使用了三星处理器,三星为Galaxy S开发了具有ARM Cortex-8核心的1GHz的Hummingbird(蜂鸟)处理器。这也是第一批兼容蓝牙3.0的手机。

① Milestone,该款手机的上市名称,中文意思为里程碑,有双关之意。——译者注

1.3.4 平板电脑

苹果公司推出iPad之后, Andriod制造商也非常期望推出他们自己的平板电脑。平板电脑一般具有4.8英寸或更大的屏幕, 带有Wi-Fi连接。由于很多平板电脑使用3G网络无线服务, 它们更像是大屏幕的智能手机。

爱可视公司(Archos)是最早在2009年年底向市场推出Android平板电脑的厂商之一。这款平板电脑具有4.8英寸的屏幕, 被称为Archos 5。爱可视在此之前已经推出了7英寸机型Archos 7。这些机型配备了硬盘驱动器, 可以存储更多的数据。戴尔公司推出了5英寸屏幕的Sreak平板电脑, 还计划推出配备7英寸屏幕和10英寸屏幕的机型。三星公司则推出了7英寸屏幕的Galaxy Tab平板电脑。这些平板电脑中的很多型号都有一个缺点, 就是无法访问Android Market, 但情况应该很快就会改变。表1-2比较了一些型号的平板电脑。

表1-2 一些有代表性的Android平板电脑

型 号	MPU	RAM/disk	屏 幕	其他特性
爱可视5 (2009年9月)	800-MHz TI OMAP 3440	256 MB/8 GB	TFT LCD 4.8英寸800×480	蓝牙2.0、802.11b/g/n、调频收音机
爱可视7 (2010年6月)	600-MHz Rockchip RK2808	128 MB/8 GB	TFT LCD 7英寸800×480	802.11b/g
戴尔Sreak (2010年6月)	1-GHz QCOM Snapdragon	256 MB/512 MB	TFT LCD 5英寸800×480	GSM/UMTS、蓝牙2.1、802.11b/g、500万像素摄像头、30万像素前置摄像头、AGPS、地理标记功能
三星 Galaxy Tablet GT-P1000 (2010年9月)	1-GHz三星Hummingbird	512 MB/16 GB	TFT LCD 7英寸1024×600	GSM/UMTS、蓝牙3.0、802.11b/g/n、310万像素摄像头

1.3.5 其他设备

Android是一个通用的嵌入式平台, 除智能手机和平板电脑以外, 也可以在许多其他行业使用。第一款带有Android设备的汽车是由上海汽车工业总公司制造的荣威350。Andorid主要用于GPS导航, 但也可以进行网页浏览。

第一款基于Android的电视是Google TV, 它是谷歌的软件、索尼的电视机、英特尔的处理器和罗技的机顶盒相结合的产物。它将互联网自然而然地引入了电视机, 但它也提供了从电视机访问Android Market的功能。

1.4 Android 设备的硬件差异

Android设备在硬件上会有一些差异, 如表1-1所示。一般来说, 大部分的差异对开发者都是透明的, 不在本书中进一步讨论。但是, 了解一些硬件差异有助于我们编写设备无关的代码。此处我们将讨论屏幕、用户输入方法和传感器等。

1.4.1 屏幕

液晶显示屏(LCD)和发光二极管(LED)是显示屏使用的两种技术。这两者具体体现在

Android手机上,就是使用薄膜晶体管(TFT)的LCD显示屏和使用有源矩阵有机发光二极管显示屏(AMOLED)。TFT显示屏的优势在于使用寿命较长,而AMOLED显示屏的优势在于没有背光,因此,显示的黑色更深且功耗较低。

总的来说,Android设备可按小、正常、大屏幕来分,也可按低、中、高像素密度来分。请注意,实际像素密度可能会有所不同,但肯定是其中之一。表1-3总结了目前已有设备的屏幕情况。请注意,表1-1列出了不同设备的屏幕密度类型。

表1-3 Android支持的设备屏幕汇总

屏幕类型	低密度(~120 ppi), ldpi	中密度(~160 ppi), mdpi	高密度(~240 ppi), hdpi
小屏幕	QVGA (240 × 320), 2.6英寸到3.0英寸对角线		
正常屏幕	WQVGA (240 × 400), 3.2英寸到3.5英寸对角线 FWQVGA (240 × 432), 3.5英寸到3.8英寸对角线	HVGA (320 × 480), 3.0英寸到3.5英寸对角线	WVGA (480 × 800), 3.3英寸到4.0英寸对角线 FWVGA (480 × 854), 3.5英寸到4.0英寸对角线
大屏幕	WVGA (480 × 800), 4.8英寸到5.5英寸对角线 FWVGA (480 × 854), 5.0英寸到5.8英寸对角线		

1.4.2 用户输入方式

触摸屏使用户能够和视觉显示互动。目前有下列三种触摸屏技术。

- 电阻屏——在玻璃屏幕的顶部覆盖了两层电阻材料层。当手指、手写笔或任何对象下压时,这两层接触在一起,触摸的位置就能被确定。电阻触摸屏的性价比高,但透光率只有75%,而且最近才实现多点触摸。
- 电容屏——在玻璃屏幕上覆盖有一个带电材料层。当手指或任何导电物体接触该层,会引起电量的变化,改变电容,就可以测量出接触的位置。电容式触摸屏的透光度高达90%,但是其精度要比电阻屏差。
- 表面声波——这里运用了一种更加先进的方法,发送和接收超声波来定位。当手指或任何物体碰触屏幕时,声波会被吸收。可以测量声波以确定碰触的位置。这是一种最经久耐用的解决方案,更适合于大屏幕,如银行的自动柜员机。

所有Android设备均使用电阻或电容触摸屏技术,并且除了一些早期的设备之外都支持多点触摸。

此外,Android的设备还配备了触摸屏的替代方法。可采用下列方法之一:

- D-pad十字键盘(方向键)——一个有上下左右方向的控制杆;
- 轨迹球——一种滚珠,类似于鼠标的指针设备;
- 触控板——一种特殊的长方形表面,用作指针设备。

1.4.3 传感器

在某种程度上,智能手机正在成为一个传感器中心,为用户提供了丰富的体验。麦克风之后,

在手机上出现的第一个附加传感器就是摄像头。不同手机的摄像头的性能差异很大，已经成为影响人们选择手机的一个重要因素。现在的附加传感器也存在这种多样性。

大部分的智能手机至少具有三种基本的传感器：一个三轴加速度计，用于测量重力加速度；一个三轴磁力计，用来测量周围的磁场；还有一个温度传感器，用来测量环境温度。例如，HTC Dream (G1) 手机包含下列传感器（可通过 `getSensorList()` 方法显示，将在第7章进一步介绍）：

- AK8976A 三轴加速度计
- AK8976A 三轴磁场传感器
- AK8976A 方向传感器
- AK8976A 温度传感器

AK8976A 是旭化成微系统公司 (AKM) 的元件，整合了压阻式加速度计、霍尔效应磁强计和温度传感器。所有传感器均提供8位精度的数据。方向传感器是一个虚拟传感器，结合使用加速度计和磁强计确定方向。

相比之下，摩托罗拉 Droid 手机中包含了以下传感器：

- LIS331DLH 三轴加速度计
- AK8973 三轴磁场传感器
- AK8973 温度传感器
- SFH7743 近距离传感器
- 方向传感器
- LM3530 光传感器

LIS331DLH 是意法半导体公司 (ST Microelectronics) 生产的12位电容式加速度计。它提供了更准确的数据，并且其采样频率可达1 kHz。AK8973 是 AKM 封装包，包含了8位霍尔效应磁强计和温度传感器。

此外，Droid 还包含两个其他传感器。SFH7743 是光电半导体的短距离近场探测器，用于在物体（如耳朵）贴近屏幕40毫米距离时关闭屏幕。LM3530 是美国国家半导体生产的可编程光传感器，可检测环境光并调节屏幕背光和LED闪光灯到适当亮度。

配备传感器的 Android 设备还有 HTC EVO 4G 手机，它具有以下传感器：

- BMA150 三轴加速度计
- AK8973 三轴磁场传感器
- AK8973 方向传感器
- CM3602 近距离传感器
- CM3602 光传感器

BMA150 是博世传感器公司生产的10位加速计，可提供高达1.5 kHz的采样频率。CM3602 是 Capella 公司生产的短距离近距离传感器和环境光感应器的组合产品。

总体而言，不同型号的 Android 设备具有不同的底层硬件。这些差异会导致性能和传感器的

精度上的差异。

1.5 Android 的特点

Android的详细特性和运用方法是贯穿全书的主题。从更宽泛的层面来看,Android的一些主要特点就是其主要卖点和差异化特点。我们最好要清楚认识到这些要点,并且尽可能利用它们。

1.5.1 多进程和应用程序微件

Android的操作系统不限制处理器在同一时刻只能执行一个应用程序。系统在单个应用程序中管理应用程序和线程的优先级。这样做的好处在于,当用户使用设备运行前台进程时,后台任务可以同时运行。例如,在用户玩游戏时,后台进程可以查询股票价格,在必要时触发警告框。

微件(Widget)是一类小型应用程序,可以嵌入到其他应用程序中(如主屏幕)。它们能够在其他应用程序正在运行时处理事件,例如启动一个音乐流媒体或显示外界温度。

多进程提供了丰富的用户体验。但是,必须小心避免耗电的应用程序耗尽电池。多进程的特点将在第3章进一步讨论。

1.5.2 触摸、手势和多点触控

触摸屏是手持设备上的一种直观用户界面。如果使用巧妙,许多操作可以无师自通。当手指触摸屏幕时,拖动和翻转是用户和图形交互的自然方式。多点触控可在同一时间跟踪按下的多个手指,通常是用来缩放或旋转视图。

一些触摸事件是对开发者透明的,不需要(编程)实现其具体的行为。可以根据需要自定义手势。重要的是触摸事件的用法要与其他应用程序尽量保持一致。这些触摸事件将在第5章进一步讨论。

1.5.3 硬键盘和软键盘

便携的口袋型设备(pocket device)使用户感兴趣的一个特点是,到底是用一个实体(也称为硬)键盘还是软件(也称为软)键盘。实体键盘有真实触感,各键位置明确,这会让一些人打字更快,而另一些人则喜欢软键盘输入简洁明快的设计和用起来方便。Android设备种类繁多,这两种类型都可以找到。对于开发者而言,副作用是两者都需要支持。软键盘的缺点是占用了一部分屏幕空间,专门用于输入,而开发者需要考虑到这一点,在多个用户界面(UI)布局中进行测试。

1.6 Android 开发

这本书关注的重点是编写Android代码,这是Android开发的主要方面。不过,我们也会适当地讲解其他方面的开发知识,包括设计和发布等方面的内容。

1.6.1 如何使用本书中的秘诀

一般来说,本书中的代码秘诀是完整的,包括在Android设备上运行可用应用程序必需的所

有信息。在第2章会讲到,运行应用程序还需要多个用户生成文件。甚至范例中只要有一个文件缺失就会对那些不熟悉Android设置的用户造成不便。因此,每个秘诀都包含必要的文件,使代码可以正常运行。每个代码文件列出时都以完整的文件名作为列表标题,这有助于找到该文件在Android工程中的位置。

同时,如果一下子给出太多文件,反而会让人分不清要实现什么功能。因此,本书示例采取了两种稍微不同于常规应用程序的编码风格。

- 代码中不包含太多注释。正文会比较详细地解释代码,而加粗的代码则表示特殊的技术要点。在日常编程实践中,程序员应在代码中多写些注释。
- 显式声明字符串,而不使用全局资源。把字符串定义为全局资源是个好办法,将在第4章中详细讨论,并配有多个范例。但是本书中,当某个秘诀只用到少量字符串时,则会显式声明字符串,而不是为定义它们去引入一个额外的文件。

初学Android的人非常适合使用安装了Android插件的Eclipse作为Android开发环境。正如第2章中所讨论的,这可确保Android工程设置和环境恰当配置,Eclipse甚至增加了图标占位符。该工具也有助于执行更高级的任务,如发布签名应用程序。

Android软件开发工具包(SDK)中的模拟器在实际应用过程中很有价值,如果能在真实的Android设备上运行应用程序就更好了。使用真实设备可以加快开发速度,提供更真实的测试。本书中所有代码示例均已经在运行Android 2.1的真实设备上测试过,并根据需要,有的还在Android 1.5或Android 2.2上进行了测试。在使用模拟器时,某些功能(例如,蓝牙配对或传感器的变化)难以模拟,对开发者不透明。因此,建议初步测试都在Android设备上完成。

1.6.2 好好设计应用程序

一个好的应用程序应该是“三好生”:好创意,好编码,好设计。通常情况下,最后一个要素是最容易被忽略的,因为大多数开发人员是独立工作的,他们不是平面设计师。谷歌肯定意识到了这一点,它推出了一系列设计指南,包括图标设计、微件设计、activity及任务设计、菜单设计。可通过如下网址访问:http://developer.android.com/guide/practices/ui_guidelines/。

好设计这一点应该要再三强调。好的设计可以让应用程序独树一帜,让用户更容易接受,并得到用户的认可。Android Market上一些最成功的应用程序都是平面设计师和开发者合作的成果。程序开发时应该专门安排一大块时间用于考虑如何做出最佳设计。

1.6.3 保持向前兼容

新的Android版本通常在API层面上让功能渐进增强,并向前兼容。事实上,只有当一个设备通过Android API兼容测试后才可被称为Android设备。然而,如果应用程序改变了底层系统,就无法保证其兼容性。为了确保未来Android更新安装到设备上时符合向前兼容性,应该遵循谷歌的下列建议。

- 不使用内部API或不受支持的API。
- 不在没有与用户交流的情况下直接操作设置。未来版本可能出于安全原因,限制某些设

置。例如，曾经应用程序可自己打开GPS或数据漫游，但是目前已经不允许了。

- 页面布局不走极端。虽然这种情况很罕见，但复杂布局（深度超过10或是总数超过30）就会导致程序崩溃。
- 不要对硬件作出错误的估计。并非所有Android设备都有全套支持的硬件。一定要检查所需的硬件，如果不存在，则需要例外处理。
- 要保证设备方向不会破坏应用程序或导致意外的行为。可以锁定屏幕方向，如第2章所述。

请注意，Android不保证向后兼容性。第2章讲到，最好是声明最低的SDK版本，使得设备可以载入适当的兼容性设置。如何在旧程序上使用新特性也将在本书多处进行讨论。

1.6.4 健壮性

和兼容性支持一样，应用程序的设计和测试也应该考虑到程序的健壮性。以下是一些可以帮助确保健壮性的提示。

- 优先使用Android库，然后才是Java库。Android库专为嵌入式设备打造，并涵盖了应用程序需要的多种需求。在其他情况下可以使用Java库。但是，在两者都可用的情况下，最好使用Android库。
- 注意内存分配。要初始化变量。尽量重用对象，而不是重新分配。这样可以加快应用程序执行速度，避免过度使用垃圾收集功能。可使用Dalvik调试监视器服务器（DDMS）工具跟踪内存分配，这将在第12章进一步讨论。
- 利用LogCat工具调试和检查警告或错误，这也将第12章讨论。
- 测试要彻底，如果可能的话在不同的环境和设备上测试。

1.7 软件开发工具包

Android SDK由开发Android应用程序需要的平台、工具、示例代码以及开发文档所组成。它作为Java开发包（JDK）的附加内容构建，而且有一个可以集成到Eclipse集成开发环境中的插件。

1.7.1 安装与更新

网上有很多文章详细介绍如何一步一步地安装Android SDK。例如，在谷歌站点<http://developer.android.com/sdk/>上可以找到所有安装过程的相关链接。因此，在此我们仅对最常见的安装步骤作一般概要介绍以供参考。这些步骤应该在作为开发环境的主机上完成。

(1) 安装Java开发包（例如安装JDK 6.0，它用于Android 2.1或者以上版本。JDK 5.0是Android开发要求的最低Java版本）。

(2) 安装Eclipse Classic（例如，Eclipse 3.5.2）。在Windows操作系统下，只需要解压下载包到某路径就可使用。

(3) 安装Android SDK（例如，r06版本）。在Windows操作系统下，只需要解压下载包到某路径就可使用。

(4) 启动Eclipse开发工具，单击Help，再选择Install New Software...，然后键入URL地址<https://dl-ssl.google.com/android/eclipse/>，安装Android DDMS和Android开发工具（ADT）。

(5) 在Eclipse中, 选择Window→Preferences... (在Mac系统上, 选择Eclipse→Preferences), 再选择Android, 然后点击Browse按钮选择Android SDK 的安装路径。

(6) 在Eclipse中, 依次选择Window→Android SDK and AVD Manager→Available Packages, 选择安装必要的API (例如Documentation for Android SDK, API 8; SDK Platform Android 2.2, API 8; Google APIs by Google Inc.以及Android API 8)。

(7) 同样从Android SDK and AVD Manager 菜单中, 创建一个Android 虚拟设备来运行模拟器或者安装USB驱动程序在真机中运行应用。

(8) 在Eclipse开发工具中, 依次选择Run→Run Configurations...来为每一个Android应用创建新的运行设置 (与Debug Configuration的设置类似)。Android JUnit测试也可以在此处配置。

至此, 开发环境基本配置完成, 我们就可以轻松地开发一个Android应用并且在模拟器或者真实Android设备中运行测试。我们可以通过在Eclipse开发环境中选择Help→Software Updates..., 再选择适当版本, 就可以更新SDK到新版本。

1.7.2 软件特性和 API 级别

Android操作系统会定期推出新功能和增强特性, 例如提高效率、修正bug等。操作系统更新的一个最主要推动力是新设备硬件能力的提升。事实上, 很多操作系统版本都是伴随着新硬件设备的推出而发布的 (例如Éclair随着设备Droid推出而发布)。

某些旧有的Android设备不符合新版本系统的硬件要求, 因此不能随着新操作系统的发布而更新系统。这就慢慢形成了使用不同Android系统版本的用户群。所以开发者需要检查用户硬件性能, 或者至少提示用户设备所需要的最低硬件特性。该任务可以通过检测一个数值, 即API级别来实现。

以下是从开发者的角度总结的不同操作系统版本的发布及主要特征。

代号Cupcake: Android OS 1.5, API level 3, 2009年4月30日发布。

- Linux内核2.6.27。
- 智能虚拟 (软) 键盘, 支持第三方键盘。
- AppWidget框架。
- Live文件夹。
- Raw格式的音频录制和播放。
- 交互式MIDI播放引擎。
- 视频录制API。
- 支持立体声蓝牙。
- 去除最终用户的根目录访问权限 (除非连接到电脑并使用SDK)。
- 通过RecognizerIntent (云服务) 进行语音识别。
- 快速GPS定位 (使用AGPS)。

代号Donut: Android OS 1.6, API Level 4, 2009年9月15日发布。

- Linux内核2.6.29。

- 支持多种屏幕尺寸。
- 手势API。
- Text-to-speech文本语音朗读功能引擎。
- 通过SearchManager整合快速搜索框。
- 虚拟专用网支持。

代号Eclair: Android OS 2.0, API Level 5, 2009年10月26日发布;

Android OS 2.0.1, API Level 6, 2009年12月3日发布;

Android OS 2.1, API Level 7, 2010年1月12日发布。

- 同步适配器API, 用于连接到任何后端。
- 在应用程序中嵌入快速联系人。
- 应用程序可以控制设备蓝牙连接。
- HTML 5支持。
- Microsoft Exchange支持。
- 通过MotionEvent类实现多点触控。
- 支持动态壁纸。

代号FroYo: Android OS 2.2, API Level 8, 2010年5月20日发布。

- Linux内核2.6.32。
- 支持即时编译 (JIT), 使代码执行速度更快。
- 通过蓝牙进行语音拨号。
- 支持车载和桌面底座模式。
- 更好定义的多点触摸事件。
- 云端到设备API。
- 应用可安装到设备的SD记忆卡中。
- 在选定设备支持Wi-Fi连接功能。
- 支持视频和图片的缩略图显示。
- 支持多语言的键盘输入。
- 为Market提供应用程序错误报告。

Android正逐渐走向成熟, 新版本发布的频率不再那么快了。虽然无线远程更新操作系统是有可能的, 但是逻辑上过于复杂, 运营商尽量避免这种方法。硬件制造商也更愿意有一个稳定的版本, 这样他们就不必对库存手机进行频繁刷新了。然而, 对于开发者来说, 当有新版本发布时, 它所带的新功能还是值得去探索和使用。

1.7.3 利用模拟器或真机调试程序

在用于开发的计算机上, 模拟器会启动一个看似Android手机的窗口, 执行真实的ARM指令。需要注意的是, 即使在高配置计算机上, 模拟器初始化的过程也会很慢。尽管有很多方法可配置模拟器模拟真机的多个方面, 例如电话呼入、有限的数据传输率、改变屏幕方向等, 但还是有很

多特性（例如感应器和音频/视频）是无法通过模拟器来实现的。对于没有设备的开发人员来说，可以把模拟器视为一种验证设备基本功能的有效方法。例如，可以用来测试平板电脑的屏幕尺寸而不用去购买一个真的平板电脑。

注意，我们必须创建一个目标虚拟设备，以正确启动模拟器。Eclipse开发工具提供了一个很好的方法来管理Android虚拟设备（AVD）。模拟器常用功能对应的快捷键如表1-4所示。

表1-4 Android OS模拟器控制

键	模拟器功能
Escape	后退
Home	主页
F2, PageUp	菜单
Shift-F2, PageDown	开始
F3	呼叫/拨号
F4	挂断/结束呼叫
F5	搜索
F7	电源按键
Ctrl-F3, Ctrl-KEYPAD_5	摄像头
Ctrl-F5, KEYPAD_PLUS	音量增
Ctrl-F6, KEYPAD_MINUS	音量减
KEYPAD_5	方向键中的确定键
KEYPAD_4, KEYPAD_6	方向键左，方向键右
KEYPAD_8, KEYPAD_2	方向键上，方向键下
F8	启用/关闭蜂窝网络
F9	切换代码分析（仅当-trace标记时有效）
Alt-ENTER	切换到全屏模式
Ctrl-T	切换到轨迹球模式
Ctrl-F11, KEYPAD_7	旋转屏幕方向到下一个布局或上一个布局
Ctrl-F12, KEYPAD_9	

一般来说，初次测试最好使用Android手机。这样可以确保功能的完整性，并获得模拟器不能完全模拟的真实感受。要把一部Android手机作为开发平台使用，只需将它连接到USB接口并确保USB驱动可以被检测（在MAC系统中这个过程可以自动完成，在Windows系统中驱动在SDK中，Linux系统请查看Google相应页面）。

为了能够支持开发调试，Android设备需要更改一些设置。从主页，依次选择MENU→Settings→Applications→Unknown sources 和 MENU→Settings→Applications→Development→USB debugging，确保应用可以通过USB线安装到设备上。更多的Android调试细节见第12章。

1.7.4 使用 Android 调试桥

通常我们可以方便地使用命令行访问Android设备。只要设备通过USB线连接到计算机上就

可以做到。SDK中自带了Android调试桥 (Debug Bridge)，能用于访问Android设备。例如像使用Linux计算机一样登录到Android设备，键入如下命令：

```
> adb shell
```

这时，许多UNIX命令都可用了。使用exit可以退出shell。只要加上一个命令就可以执行，而不需要先进入或者退出shell：

```
> adb shell mkdir /sdcard/app_bkup/
```

要从设备中复制文件，可使用pull命令，如果需要可以重命名：

```
> adb pull /system/app/VoiceSearchWithKeyboard.apk VSwthKeyboard.apk
```

要复制一份文件到设备中，可使用push：

```
> adb push VSwthKeyboard.apk /sdcard/app_bkup/
```

要删除应用，例如从设备中删除com.dummy.game，可键入如下命令：

```
> adb uninstall com.dummy.game
```

以上命令是最常用的，另外还有很多其他可用的命令，我们将在第12章介绍这些命令的用法。

1.7.5 签名和发布应用

要在Android Market上发布应用，必须对它进行签名。完成签名需要生成一个密钥并把它放在一个安全的地方，然后在发布模式下将应用程序打包，并用生成的密钥对其进行签名。当应用更新时，必须使用相同的密钥对其签名以确保升级过程对用户是透明的。

Eclipse开发环境可以自动为我们完成以上工作。只需在要签名的工程上单击右键选择Export...→Export Android Application，即可开始打包工作。可以利用密码生成密钥，密钥需要保存起来以便将来的应用和应用更新使用，然后通过菜单继续完成APK文件的创建。它是一个发布模式的Android工程并用密钥签名的封包。现在我们就可以将应用程序上传到Android Market上了。

1.8 Android Market

完成设计、开发、测试和签名工作后，你就可以在Android Market发布应用程序。然而若想使用谷歌的Android Market，你还需要申请一个Google Checkout账号。该账号不仅用来支付25美元的注册费，也供开发人员收取付费应用的报酬。开发者的作品能够公之于众，在上传后的短短数小时内，让全世界的人们浏览、下载、评分和评价，是多么激动人心的事啊！下面列出几条应用程序发布建议供你参考。

1.8.1 最终用户许可协议

在全球大部分地区，任何形式的原创内容的发布均受《伯尔尼公约》^①的自动版权保护。但通常还是会为内容添加附有出版时间的版权声明，如©2010。我们会在第4章中讨论如何为Android

^① 《伯尔尼公约》(Berne Convention)，有关保护文字和艺术作品版权的国际公约。——译者注

应用程序添加版权声明。

最终用户许可协议（EULA）是开发者（或开发商）与顾客（或最终用户）签署的发布程序的软件合同，我们可以利用EULA进一步维护开发者（或开发商）的权益。通常EULA声明包括诸如授权许可证、版权和免责条款等具体权责内容。所以应用程序特别是付费应用最好还是附上EULA。我们将会在第9章讨论如何为Android应用添加EULA。

1.8.2 提升应用程序的曝光率

用户通常使用以下三种方式查找应用程序，充分利用这些渠道有助于打响应用程序知名度。

第一种方式，通过选择列出“最新应用”（“just in” apps）查看应用程序。首先要为你的应用程序选择恰当的描述性名称和所属类别，如游戏、通信软件等。为吸引人们的注意，用语一定要简洁明了。游戏大类下又分很多子类。如果你的应用程序非常好玩，但又没有明确的用途或者等级评分，不妨将其归入休闲游戏系列。即便如此，Android Market每月都会有10 000多款应用程序上架，上传的应用往往一两天后就被挤下了“最新应用”名单。

第二种方式，通过关键字检索查找应用程序。我们可以推测用户可能使用哪些主要的关键字，然后将其加到程序的标题或描述中。世界各地用户语言不同，所以应包含适当的国际关键字。

第三种方式，通过选择“热门程序排行”（Top apps）发现应用程序。这个排序由评分和下载次数决定。若想在此类目跻身前列，开发人员就要耗费时间和精力不断更新和完善应用。这就引出了提高程序知名度的最后一点，即健壮性原则。要确保你的程序不会含有致命性bug，耗电量小、卸载方式直观明了，否则，如果有“太过耗电”、“无法卸载”等类似评语就很容易吓跑潜在客户。

值得注意的是，开发者与用户之间的交互几乎全部发生在Android Market，人们通常不会通过发布者的联系方式和支持网站去找详细资料，因此这部分信息略显多余。

1.8.3 脱颖而出

有时开发者创建某个应用程序之后，突然发现Android Market已经发布了类似软件。我们应该把它看作是一个机遇而非挫折。优良的设计、界面和运行状况能快速赚取用户口碑，从而令我们的应用程序脱颖而出。通常来说，原创是最好的，但我们也没必要排斥学习和模仿，只要注意不侵犯版权即可。

1.8.4 为应用程序收费

应用程序在Android Market正式上架或者上传更新的时候，开发者需要决定是否对其收费，主要有以下5种方式。

- 完全免费。所有具有访问权限的人都可以在Android Market中查看和安装此应用。
- 载有广告的免费应用。有时开发者需要为应用程序拉赞助，更多的时候是和第三方集成商合作。广告商通过广告点击率付钱，而较少凭广告印象付费。图1-1即为一条AdMob的横幅广告。该类广告要求应用程序需具有访问Internet和设备所处地点的权限。建议使用低精度定位而不是精确定位，以免吓跑一些潜在客户，拒绝安装你的应用。

- 完全付费。Google帮助管理资费，并抽取三成收益。没有开通Google Checkout服务的国家或地区的用户无法查看和安装此类应用。因此，部分开发者随即转向第三方应用商店发售此类应用程序。
- 同时发布免费限制版和付费完整版两种应用。这样就可以试用该款应用程序，如果喜欢，用户购买完整版时就不会太犹豫。某些应用（如10级闯关游戏）非常适合采用此种模式推广，但并不是所有程序均适用。
- 售卖虚拟商品。这是Facebook的应用采用的重要模式，备受移动社区关注。

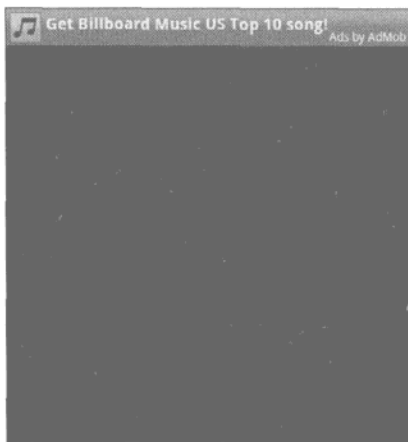


图1-1 AdMob的横幅广告案例

免费应用程序往往会有很多人浏览。再晦涩和古怪的新应用上市的时候，头一个月也至少会有1000人浏览和下载试用。有些开发者明确声明“本程序毫无用处”，结果应用的下载量也能达到10 000以上并获得四星级评价。有点儿用的一些免费应用下载量可以达到50 000，功能非常实用的应用程序下载量甚至可能超过100 000。对于大多数开发者来说，如此大的曝光率着实令人印象深刻。

目前移动广告仍然处于起步阶段，吸引力不足，所以广告点击率并不理想。现在Market中运营最好的还是付费应用，只要程序功能实用，描述清晰，有不少正面评论，人们就会购买它。如果应用程序非常成功，那么价格贵点也无可厚非。

1.8.5 管理评论和更新

很多应用程序都是开发者发布程序之后，再根据用户的反馈意见不断升级完善应用，从而大获成功的。用户喜欢看到开发者回应评论，这会引导越来越多的人下载该应用，随着下载量的攀升，应用程序也更加受认可。

一般来说，每200个用户中会有1人评分，发表评论的人就更少了。如果用户肯花时间评价你的程序，这些意见多半值得一听，尤其是一些建设性评论，比如“在HTC Hero型号手机上不能

正常运行”或者“非常不错的程序，若是怎么怎么做就更好了”之类。

开发者能够根据已有用户的反馈意见积极升级应用程序，会让大家看在眼里，从而吸引更多用户。任何情况下，开发者都要明确强调软件的升级原因。要知道很多用户一天能收到十几条应用程序的更新通知，如果没有一个很好的升级理由，他们很可能就不更新了。

1.8.6 Android Market 的候补之选

除Android Market之外还有一些独立的Android应用程序商店。虽然这些应用程序商店不像Google Market那样访问便捷，但有诸如曝光率高、收费条目多、不收取提成费用等优点。同时还有些Android设备制造商为其终端开发了定制的应用程序商店。例如，我们可以通过摩托罗拉应用商店使中国和拉丁美洲的摩托罗拉Android手机能看到我们的应用，该应用商店的网站为<http://developer.motorola.com/shop4apps>。



应用程序基础知识： activity和intent

2

每个Android应用程序在开发时都是一个独立的Android工程。本章介绍了Android工程目录结构，并简要概述程序的基本组成模块，这些内容为理解本书的秘诀提供了非常有用的背景知识。本章的后半部分将着重讲解activity以及触发这些activity的intent。

2.1 Android 应用程序预览

Android应用程序可以包含五花八门的功能，比如编辑文本、播放音乐、设定闹钟还有开启通讯录等。这些功能可以划分为以下四类Android组件（见表2-1），每个组件都归属于一个Java基础类。

表2-1 Android应用程序所包含的四种组件

功 能	Java基础类	范 例
关注用户操作	Activity	编辑文本，玩游戏
后台进程	Service	播放音乐，更新天气图标
接收消息	BroadcastReceiver	根据特定事件触发警报
存取数据	ContentProvider	开启手机通讯录

每个Android应用都由一个或多个组件组成。当要用到某个组件的时候，Android操作系统就会将它们实例化。在拥有特定权限的情况下，其他应用程序同样也可以使用它们。

Android操作系统集成了很多功能（某些功能甚至并非和某个应用程序直接相关，如呼入电话），每个组件都具有以下生命周期，包括创建（create）、获得焦点（focus）、失去焦点（defocus）和销毁（destroy）。我们可以通过改写程序默认的行为，使交互对用户更加友好，比如保存变量或者恢复用户界面元素。

除了ContentProvider组件，每个组件都需要一个叫做Intent的异步消息来激活。Intent可包含一组（Bundle）描述该组件的辅助数据。这也提供了一种在组件之间传递消息的方法。

本章最后将会使用最常见的组件Activity来演示前面提到的概念。由于activity总是和具体

的用户交互相关,所以每个activity在创建的时候会自动创建一个新窗口。当然还会提到一些关于UI的概要介绍。至于Service和BroadcastReceiver这两个组件我们将会在第3章讲解,而ContentProvider则会在第9章阐述。

2.1.1 秘诀 1: 创建工程并新建 activity

2

创建Android工程或者组件最简单的方法莫过于使用Eclipse提供的集成开发环境(IDE),该方法能够确保正确安装辅助文件。创建Android工程的具体过程如下所示。

(1) 在Eclipse中,选择File→New→Android Project。然后就会显示Android工程的创建画面。

(2) 填写工程名称(Project name),此处输入SimpleActivityExample。

(3) 在Build Target选项框中选择编译目标,这些选项与开发电脑的SDK版本有关。

(4) 填写程序名称(Application name),此处为Example of Basic Activity。

(5) 填写应用程序包名称(Package name),此处为com.cookbook.simple_activity。

(6) 根据同样的步骤创建主activity,注意一定要勾选Create Activity,并填写activity名称,此处为SimpleActivity。

所有的activity都继承于抽象类Activity或者其子类,并通过onCreate()方法创建。activity通常在初始化的时候都会重载该方法,比如设置UI、创建监听按钮、初始化参数或者开启线程等。

如果在创建工程时没有创建主activity,或者需要添加其他activity,可以通过以下步骤来创建activity。

(1) 首先创建一个新类扩展Activity类。(在Eclipse中,右键单击project,选择New→Class,然后指定android.app.Activity作为父类。)

(2) 重载onCreate()功能。(在Eclipse中,右键单击class文件,选择Source→Override/Implement Methods...,然后勾选onCreate()方法。)

(3) 作为最常被重载的方法之一,activity也必须激活父类方法,否则运行时可能会抛出异常。如清单2-1所示首先调用super.onCreate()方法,确保正确初始化activity。

清单2-1 src/com/cookbook/simple_activity/SimpleActivity.java

```
package com.cookbook.simple_activity;

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

public class SimpleActivity extends Activity {

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

(4) 如果使用UI,则要在res/layout/目录下的一个XML文件中指定页面布局。此处为main.xml,

如清单2-2所示。

清单2-2 res/layout/main.xml

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

(5) 通过setContentView()方法设置activity的布局,并将XML布局文件作为resource ID传递给它。此处为R.layout.main,见清单2-1。

(6) 在AndroidManifest XML文件中声明activity的属性,详细内容见清单2-5。

注意字符串类型的资源要在res/values/文件夹下的strings.xml文件中定义,如清单2-3所示。所有字符串都集中于此处定义,可以方便修改或重用。

清单2-3 res/values/strings.xml

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

现在我们进一步探讨该工程的目录结构和自动生成内容。

2.1.2 工程目录结构及自动生成内容

图2-1为Eclipse Package Explorer显示的一个工程结构示例。

除Android 2.0库文件以外,该工程的目录结构中的文件既有用户创建的也有系统自动生成的。

用户创建的文件如下所示。

- src/是开发者自己编写的或者导入的Java包。每个包可以包含多个不同的.java类文件。
- res/layout/用来存放说明每个界面布局的XML文件。
- res/values/用来存放被其他文件所引用的XML格式的資源文件。
- res/drawable-hdpi/、res/drawable-mdpi/和res/drawable-ldpi/是程序所使用图片的资源目录,分别存放高、中、低不

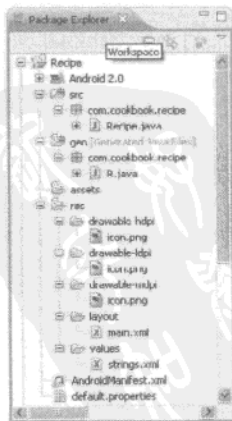


图2-1 Eclipse IDE显示的一个Android工程的目录结构

同dpi分辨率的图片。

- ❑ assets/存放程序使用的nonmedia文件。
- ❑ AndroidManifest.xml向Android操作系统说明该工程。

自动生成的文件如下所示。

- ❑ gen/存放系统自动生成代码，包括生成的R.java类。
- ❑ default.properties存放工程环境信息。尽管该文件由系统自动生成的，但开发人员也可以根据需要修改。

应用程序的资源包括描述布局的XML文件，描述字符串值、UI元素标签的XML文件，以及其他如图片、声音等辅助文件。编译时，对资源的引用都会添加到自动生成的包装类R.java中。该文件由AndroidAsset打包工具（aapt）自动生成。清单2-4为秘诀1使用的该文件。

清单2-4 gen/com/cookbook/simple_activity/R.java

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package com.cookbook.simple_activity;

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;
    }
}
```

此处的每个资源都被映射成一个唯一的整型值。通过这种方式，R.java类提供了一种在Java代码中引用外部资源的方法。例如想要在Java文件中引用main.xml布局文件，就需要使用整型值R.layout.main。如果是在XML文件中引用main.xml文件，就需要使用"@layout/main"字符串。

在Java或者XML文件中引用资源请参见表2-2。请注意，假若要定义一个ID为home_button的按钮，需要在引用字符串前添加“+”号，即：@+id/home_button。第4章再详细地探讨资源引用，此处内容对本章秘诀的学习已经足够。

表2-2 在Java和XML文件中引用不同的资源

资 源	在Java中引用	在XML中引用
<code>res/layout/main.xml</code>	<code>R.layout.main</code>	<code>@layout/main</code>
<code>res/drawable-hdpi/icon.png</code>	<code>R.drawable.icon</code>	<code>@drawable/icon</code>
<code>@+id/home_button</code>	<code>R.id.home_button</code>	<code>@id/home_button</code>
<code><string name="hello"></code>	<code>R.string.hello</code>	<code>@string/hello</code>

2.1.3 Android 包和 Manifest 清单文件

Android工程,有时也称为Android包,是Java包的集合。不同的Android包可以包含相同名称的Java包,但在安装到Android设备上时,各个Android包的名字必须是唯一的。

为了操作系统能够正确访问这些Android包,每个应用程序必须在名为AndroidManifest 的XML文件中注册声明它所使用的组件。此外该XML文件还包含运行该应用程序所需的权限及操作。清单2-5为秘诀1所用文件。

清单2-5 AndroidManifest.xml

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

Android包所有XML文件第一行都必须指定编码,该行代码为标准代码。manifest元素定义Android包的名称和版本号。versionCode可以根据你的程序情况定义,为确定版本高低关系的一个整数。versionName采用人可读懂的格式表示,可以声明主次修订版本号。

application元素定义用户从Android设备菜单可以看到的应用程序图标和名称。名称是一个字符串,为了确保在用户设备中将其显示在应用图标下方,应该尽量使其简短。一般来说,名称最多两个单词,每个单词最好在十个字符之内,中间不能含有空格。

activity元素定义程序启动时触发的主activity,以及该activity激活后标题栏中显示的名称。在这儿需要指定Java包名,本例为com.cookbook.simple_activity,相应activity名称为Simple-

Activity。由于Java包名称一般和Android包名称一致，所以常常会使用缩写SimpleActivity。不过最好记住Android包和Java包还是有区别的。

intent-filter元素向系统说明该组件功能。鉴于此作用，它可以包含多个action，category或者data元素。该点在不同的秘诀中都有所体现。

uses-sdk元素定义运行此程序所需的API级别。一般来说，API级别定义如下：

```
<uses-sdk android:minSdkVersion="integer"
          android:targetSdkVersion="integer"
          android:maxSdkVersion="integer" />
```

由于Android系统向前兼容，maxSdkVersion所定义的最高API支持版本会令人极度沮丧，因为它不支持Android 2.0.1及之后的版本。targetSdkVersion可要可不要，该项用于允许同一SDK版本的设备禁用加快操作速度的升级兼容性设置。但minSdkVersion必须定义，以确保应用程序在不支持该应用所需的功能的平台上运行时不会崩溃，定义时尽可能选择较低的API级别。

AndroidManifest存放运行该应用程序所需的权限。我们会在随后的章节中进一步详细阐述，但以上部分基本可以涵盖本章秘诀。

2.1.4 重命名应用程序中的部分文件

有时候我们需要重命名Android工程的部分文件，或许是从本书中手动复制一个文件放在工程中，或许是在开发过程修改了程序名称，需要在文件系统的目录树反映出来。Android提供了工具帮我们自动完成此项工作，并且可以同步更新交叉引用。例如在Eclipse IDE中，使用下列不同的方式来重命名应用程序的部分文件。

□ 重命名Android工程，步骤如下：

- (1) 右键单击该工程选择Refactor→Move移到文件系统中的一个新目录；
- (2) 右键单击该工程选择Refactor→Rename重命名工程。

□ 重命名Android包，步骤如下：

- (1) 右键单击该包选择Refactor→Rename重命名该包；
- (2) 更新AndroidManifest.xml包名称。

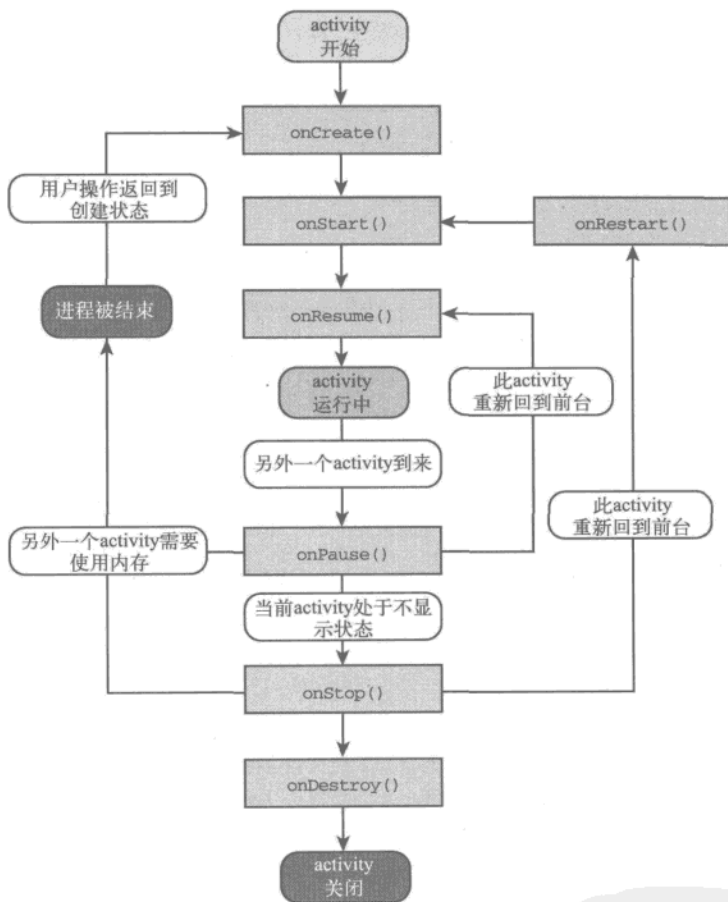
□ 重命名Android类（如Activity、Service、BroadcastReceiver、ContentProvider等主要组件），步骤如下：

- (1) 右键单击该Java文件选择Refactor→Rename重命名该类文件；
- (2) 更新AndroidManifest.xml确保android:name使用新组件名。

注意重命名XML文件等其他类型文件的时候，通常都要手动修改Java代码中的相应的引用。

2.2 Activity 的生命周期

程序中的每个activity都有自己的生命周期。通过调用onCreate()方法，activity能且仅能被创建一次。当onDestroy()方法执行时，该activity随即关闭。正如图2-2所阐述的那样，不同事件可以导致activity不同的运行状态。秘诀2将为我们一一呈现这些功能。

图2-2 activity的生命周期, 来源: <http://developer.android.com/>

2.2.1 秘诀 2: 使用其他的生命周期方法

下面的秘诀提供了一种查看活动中activity生命周期的简单方法。为便于演示, 每个被重载的方法都有明确说明, 我们通过加入Toast命令, 使得该方法在启动的时候, 在屏幕上显示。(关于Toast微件的更多内容请参见第3章)。在Android设备上运行以下代码 (如清单2-6所示), 并尝试各种情况, 特别是注意以下几种操作:

- 颠倒屏幕方向, 将结束并重新运行activity;
- 按下Home按钮将暂停activity, 但并不结束;
- 按下程序图标可能会开启新的activity实例, 即使先前的activity没有关闭;
- 屏幕处于休眠态时会暂停activity, 屏幕重新唤醒时会继续该activity (类似于呼入电话)。

清单2-6 src/com/cookbook/activity_lifecycle/ActivityLifecycle.java

```
package com.cookbook.activity_lifecycle;

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

public class ActivityLifecycle extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onResume() {
        super.onResume();
        Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
        super.onPause();
    }

    @Override
    protected void onStop() {
        Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
    }
}
```

```

        super.onDestroy();
    }
}

```

我们可以看到, 用户的很多常见操作都可能会导致activity暂停运行、结束甚至启动数个应用程序版本。在继续下一部分内容之前, 有必要给大家介绍两种方法来控制这种操作行为。

2.2.2 秘诀 3: 强制执行单任务模式

如果应用程序跳转走后再次启动的话, 可能会在设备上产生多个activity实例。最终为释放内存, 多余的activity实例会被系统杀死, 但与此同时, 也很可能会导致异常。为避免上述情况发生, 程序员可以在AndroidManifest中控制每个activity的这种行为。

为确保设备上只有一个activity实例在运行, 需要在activity元素中包含MAIN和LAUNCHER两个intent过滤器, 如下:

```
android:launchMode="singleInstance"
```

该行代码确保在任务中的任何时刻, 每个activity都只有唯一一个运行实例。此外, 该实例的所有子activity都作为自身任务启动。为进一步限制应用程序中的所有activity都只能运行一个实例, 不妨使用以下代码:

```
android:launchMode="singleTask"
```

这样使得所有activity作为同一个任务, 共享信息非常方便。

此外, 有时我们希望无论用户通过什么方式进入activity都能够保存任务的状态。例如, 如果用户离开了应用程序, 不久后又重新启动了该应用程序, 默认情况下系统会重设任务到初始化状态。为确保用户总是能返回到关闭之前的状态, 需要在任务的根activity的activity元素的属性中作如下定义:

```
android:alwaysRetainTaskState="true"
```

2.2.3 秘诀 4: 强制屏幕方向

每个带有加速度计的Android设备都可以判定方向。当设备由纵向模式切换到横向模式时, 默认动作是相应地旋转应用程序视图。然而秘诀2, 倒置屏幕会导致已经结束的activity重新启动。如果是这种情况, 那么就会丢掉当前的程序状态, 从而破坏用户体验。

解决屏幕倒置问题的一种方案是在发生改变之前保存用户的状态, 改变方向后重新启动activity时读取用户先前状态。还有一种更简单的办法, 就是强制设定屏幕的方向, 禁止旋转切换视图。AndroidManifest中列出的每个activity都可以定义屏幕方向。比如为了指定某个activity始终以纵向模式运行, 在activity元素中可以添加如下代码:

```
android:screenOrientation="portrait"
```

同样, 如果想设定为横向模式, 可以使用如下代码:

```
android:screenOrientation="landscape"
```

然而, 在硬键盘滑出时, 先前的情况还是会导致activity的关闭和重新启动。所以我们可以采

用第三种办法，即告知Android系统处理应用程序方向和键盘滑出事件。可以在activity元素的属性中添加如下代码：

```
android:configChanges="orientation|keyboardHidden"
```

该方法可以单独使用，也可以和screenOrientation属性结合在一起使用，视应用程序要求而定。

2.2.4 秘诀 5：保存和恢复 activity 信息

每当一个activity即将被杀死时，都会调用onSaveInstanceState()方法。重载该方法可以保存相关状态信息。当重新创建该activity时，则会调用onRestoreInstanceState()方法。重载该方法可以获取先前保存的状态信息。这样当应用程序经历生命周期变化时，就可以为用户带来无缝体验。值得注意的是，大部分UI控件状态都不需要我们亲自处理，系统会自动帮我们完成此项工作。

onPause()方法略有不同。如果另一个组件在activity中启动，就会调用onPause()方法暂停此activity活动。稍后系统如要回收内存等资源时，该activity仍然处于暂停状态，Android系统就会调用onSaveInstanceState()方法保存状态信息，然后将其杀死。

清单2-7为存取包含一个string数组和一个float数组的实例状态信息的示例。

清单2-7 onSaveInstanceState()和onRestoreInstanceState()示例

```
float[] localFloatArray = {3.14f, 2.718f, 0.577f};
String localUserName = "Euler";

@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    //save the relevant information
    outState.putString("name", localUserName);
    outState.putFloatArray("array", localFloatArray);
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    //restore the relevant information
    localUserName = savedInstanceState.getString("name");
    localFloatArray = savedInstanceState.getFloatArray("array");
}
```

请注意，onCreate()方法也包含Bundle savedInstanceState。当activity关闭之后重新初始化，之前onSaveInstanceState()方法中保存的bundle状态信息会传递给onCreate()方法。总之，所有保存的状态信息都会传递给onRestoreInstanceState()方法，所以自然会利用它来恢复之前状态。

2.3 多个 activity

即使最简单的应用程序也会拥有多个功能,所以经常需要使用多个activity。比如,一个游戏可能包含两个activity,一个用来显示高分排行榜,另一个则用来显示游戏画面。一个记事本程序可能包含三个activity:查看笔记列表、阅读某条笔记、编辑某条笔记或加新笔记。

当程序启动时,就会执行AndroidManifest XML文件中定义的主activity。通过事件触发,可以跳转到另外一个activity。当第二个activity被激活时,先前的主activity就处于暂停状态。当第二个activity运行结束后,主activity就会再次回到前台恢复运行。

若想激活程序中的某个组件,可以使用intent直接来指定该组件。但如果想通过intent过滤器指定,则可以使用隐式intent,再由系统决定最合适的组件,不管它是其他应用程序组件还是本机操作系统自带组件,都可以为其所用。要注意的是,其他应用程序中的隐式intent不需要在当前程序中的AndroidManifest文件注册声明。

Android主张尽可能利用隐式intent为用户提供强大的功能模块框架。当新开发的组件能满足隐式intent过滤器的需求,就可以用它来替代Android的内部intent。譬如,在Android设备上加载手机通讯录。当用户选择一个联系人时,Android系统会自动通过适当的intent过滤器查找联系人来发现所有可用的activity,并让用户自己选择所使用的activity。

2.3.1 秘诀 6: 使用按钮和文本框

我们使用触发事件充分演示多个activity的切换。为此,我们在示例中引入了按钮按下事件。下面将在某个页面布局中添加一个按钮,并指定按钮被按下时的动作,步骤如下。

- (1) 在XML页面布局文件中声明一个button控件:

```
<Button android:id="@+id/trigger"
        android:layout_width="100dip" android:layout_height="100dip"
        android:text="Press this button" />
```

- (2) 通过布局文件中的button ID声明button控件对象:

```
Button startButton = (Button) findViewById(R.id.trigger);
```

- (3) 添加点击按钮事件的OnClickListener监听器:

```
//setup button listener
startButton.setOnClickListener(new View.OnClickListener() {
    //insert onClick here
});
```

- (4) 重载监听器的onClick方法执行你想要的动作:

```
public void onClick(View view) {
    // do something here
}
```

我们可以通过改变屏幕上显示的文字向用户反馈交互结果。定义文本框并用编程手段来实现改动,步骤如下所示。

(1) 在XML布局文件中通过ID声明一个textview控件。同时也可以初始化, 设定为某个值。(此处将其初始化为strings.xml文件中名为“hello”的字符串值。)

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

(2) 在布局文件中声明一个TextView控件, 指向TextView ID:

```
private TextView tv = (TextView) findViewById(R.id.hello_text);
```

(3) 如果要修改文本内容, 可以使用setText方法:

```
tv.setText("new text string");
```

这两个UI技巧将会在本章后面的几个秘诀中用到。第4章将会系统讲解Android的UI控件。

2.3.2 秘诀 7: 通过事件启动另外一个 activity

在本秘诀中, MenuScreen是主activity, 如清单2-8所示, 这里将启动PlayGame activity。其触发事件为Button微件的单击事件。

当用户单击按钮时会运行startGame()方法, 该方法启动PlayGame activity。而当用户在PlayGame activity中点击按钮时, 则会调用finish()方法, 将控制权移交给调用它的activity。启动activity的步骤如下:

- (1) 声明一个intent, 指向即将被启动的activity;
- (2) 调用该intent的startActivity方法;
- (3) 在AndroidManifest中声明其他的activity。

清单2-8 src/com/cookbook/launch_activity/MenuScreen.java

```
package com.cookbook.launch_activity;

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

public class MenuScreen extends Activity {

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

        //setup button listener
        Button startButton = (Button) findViewById(R.id.play_game);
        startButton.setOnClickListener(new View.OnClickListener() {
```

```

        public void onClick(View view) {
            startGame();
        }
    });
}

private void startGame() {
    Intent launchGame = new Intent(this, PlayGame.class);
    startActivity(launchGame);
}
}

```

在匿名内部类中提供当前上下文环境

注意,在通过点击按钮启动activity时还需要作额外的考虑,如清单2-8所示。intent需要上下文环境。但this引用在onClick方法中不能正确解析。在匿名内部类中提供当前上下文环境的方法如下:

- 使用Context.this代替this;
- 使用getApplicationContext()来代替this;
- 显式地使用类名MenuScreen.this。

调用一个在适当的上下文环境级别中声明的方法,如清单2-8所使用的startGame()。

以上方法都可以互相转换,我们可以根据需要灵活运用。

清单2-9所示的PlayGame activity只有一个按钮,带有onClick监听器,点击该按钮调用finish()方法会将控制权返回给主activity。当然也可以根据需要为PlayGame activity添加更多的功能模块,每个分支模块的代码都可以调用finish()方法结束该activity的运行。

清单2-9 src/com/cookbook/launch_activity/PlayGame.java

```

package com.cookbook.launch_activity;

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

public class PlayGame extends Activity {

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

        //setup button listener
        Button startButton = (Button) findViewById(R.id.end_game);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                finish();
            }
        });
    }
}

```

```

    };
}
}

```

如清单2-10所示，该按钮必须添加到main.xml布局文件中，按钮的ID为play_game，必须与清单2-8中所声明的内容相匹配。此处使用和设备无关的像素（dip）定义按钮大小，我们将会在第4章深入讨论。

2

清单2-10 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        />
    <Button android:id="@+id/play_game"
        android:layout_width="100dip" android:layout_height="100dip"
        android:text="@string/play_game"
        />
</LinearLayout>

```

正如清单2-11所示，PlayGame activity引用了其自己的ID为end_game的按钮，该按钮的布局资源R.layout.game对应的是XML布局文件game.xml。

清单2-11 res/layout/game.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:id="@+id/end_game"
        android:layout_width="100dip" android:layout_height="100dip"
        android:text="@string/end_game" android:layout_centerInParent="true"
        />
</LinearLayout>

```

虽然在各种情况下，文本都可以显式地写在文件中，但为每个字符串定义变量是一种好的编程习惯。在本秘诀中，有play_game和end_game两个字符串值，它们被定义在一个字符串XML资源文件中，见清单2-12所示。

清单2-12 res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">This is the Main Menu</string>
    <string name="app_name">LaunchActivity</string>
    <string name="play_game">Play game?</string>
    <string name="end_game">Done?</string>
</resources>
```

最后需要在AndroidManifest XML文件中为PlayGame这个新类声明其默认的动作, 详见清单2-13。

清单2-13 AndroidManifest.xml

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

2.3.3 秘诀 8: 将语音转换成文本并启动 activity 显示结果

本秘诀将演示在启动activity时如何处理其返回值的问题。同时演示了如何利用Google的RecognizerIntent将语音转换输出成文本并在屏幕上显示的功能。此例的触发事件同样是按钮单击事件, 它将启动RecognizerIntent activity, 辨识麦克风输入的声音并将其转换成文本格式。运行结束后将文本传回给调用它的activity。

当返回时, 首先会用返回的数据调用onActivityResult()方法, 然后调用onResume()方法使activity正常运行。因为返回的activity可能存在问题, 导致不能正确传递值, 所以必须核查

resultCode确保是RESULT_OK才可以继续解析返回的数据。

请注意，启动回传数据的activity通常都会调用同一个onActivityResult()方法。所以需要
使用请求码（request code）判断哪个activity要回传数据。当activity启动完成后，就会把控制权重
新交回给调用它的activity，并用同样的请求码调用onActivityResult()方法。

启动带有返回值的activity的步骤如下。

(1) 通过intent调用startActivityForResult()，定义要启动的activity并标记requestCode。

(2) 重载onActivityResult()方法，检查返回结果的状态以检查期望的requestCode，并解
析返回数据。

使用RecognizerIntent的步骤如下。

(1) 声明一个intent，设置其动作为ACTION_RECOGNIZE_SPEECH。

(2) 向intent传递附加内容，至少EXTRA_LANGUAGE_MODEL是必需的。它可以被设置成
LANGUAGE_MODEL_FREE_FORM或者LANGUAGE_MODEL_WEB_SEARCH。

(3) 返回的数据包中包含可能和原文匹配一个字符串列表。通过data.getStringArray-
ListExtra可以获取这些数据，它将映射为ArrayList类型资源供稍后使用。

使用TextView将返回的文本显示到屏幕上。主activity的内容参见清单2-14。

此外还有main.xml和strings.xml这两个辅助文件，用于定义按钮和存放结果的TextView。具体
内容可以参考秘诀7中清单2-10以及清单2-12的内容。当前只需要在AndroidManifest中声明主
activity，这点和秘诀1中的步骤一致。RecognizerIntent activity是Android系统原生的activity，所以
在使用前就不需要显式声明。

清单2-14 src/com/cookbook/launch_for_result/RecognizerIntent Example.java

```
package com.cookbook.launch_for_result;

import java.util.ArrayList;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class RecognizerIntentExample extends Activity {
    private static final int RECOGNIZER_EXAMPLE = 1001;
    private TextView tv;

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

        tv = (TextView) findViewById(R.id.text_result);
```



```

//setup button listener
Button startButton = (Button) findViewById(R.id.trigger);
startButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        // RecognizerIntent prompts for speech and returns text
        Intent intent =
            new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);

        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
            "Say a word or phrase\nand it will show as text");
        startActivityForResult(intent, RECOGNIZER_EXAMPLE);
    }
});
}

@Override
protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {
    //use a switch statement for more than one request code check
    if (requestCode==RECOGNIZER_EXAMPLE && resultCode==RESULT_OK) {
        // returned data is a list of matches to the speech input
        ArrayList<String> result =
            data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);

        //display on screen
        tv.setText(result.toString());
    }

    super.onActivityResult(requestCode, resultCode, data);
}
}

```

2.3.4 秘诀 9: 实现选择列表

应用程序中常常需要提供给用户一个选择列表以供用户点击选择, 利用Activity的子类ListActivity就可以实现该功能, 并根据用户的选择触发事件。

创建选择列表的具体步骤如下。

(1) 首先创建一个类扩展ListActivity类, 而非Activity类:

```

public class ActivityExample extends ListActivity {
    //content here
}

```

(2) 创建一个字符串数组, 为每个选项指定标签:

```

static final String[] ACTIVITY_CHOICES = new String[] {
    "Action 1",
}

```

```

        "Action 2",
        "Action 3"
    };

    (3) 通过ArrayAdapter调用setListAdapter()方法, 并指明该选择列表和布局方式:
    setListAdapter(new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, ACTIVITY_CHOICES));
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    getListView().setTextFilterEnabled(true);

    (4) 运行OnItemClickListener监听确定用户选择了哪个选项, 并针对其作出反馈:
    getListView().setOnItemClickListener(new.OnItemClickListener()
    {
        @Override
        public void onItemClick(AdapterView<?> arg0, View arg1,
            int arg2, long arg3) {
            switch(arg2) { //extend switch to as many as needed
            case 0:
                //code for action 1
                break;
            case 1:
                //code for action 2
                break;
            case 2:
                //code for action 3
                break;
            default: break;
            }
        }
    });

```

上述技巧在下一个秘诀中也会用到。

2.3.5 秘诀 10: 使用隐式 intent 创建 activity

隐式intent不会确切指定需要使用哪个组件。相反, 它们通过过滤器确定所需要的功能, 再由Android系统选择最匹配该功能的组件。intent过滤器可以是动作、数据或者分类 (category)。

动作是最常用的intent过滤器, 而其中又数ACTION_VIEW最为常用。它需要声明一个统一资源标识符 (URI), 用来向用户显示数据。对于给定的URI选择最佳的处理方式。例如在下面的范例如中, 隐式intent在case 0、1、2中虽然句法格式相同, 但产生的结果却大大不同。

使用隐式intent启动activity的步骤如下:

- (1) 为intent声明恰当的intent过滤器 (ACTION_VIEW、ACTION_WEB_SEARCH等);
- (2) 向intent添加运行某个activity所需要的附加信息;
- (3) 将intent传递给startActivity()。

清单2-15将演示如何处理多个intent。

清单2-15 src/com/cookbook/implicit_intents/ListActivityExample.java

```

package com.cookbook.implicit_intents;

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

public class ListActivityExample extends ListActivity {
    static final String[] ACTIVITY_CHOICES = new String[] {
        "Open Website Example",
        "Open Contacts",
        "Open Phone Dialer Example",
        "Search Google Example",
        "Start Voice Command"
    };

    final String searchTerms = "superman";

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

        setListAdapter(new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, ACTIVITY_CHOICES));
        getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
        getListView().setTextFilterEnabled(true);
        getListView().setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1,
                int arg2, long arg3) {
                switch(arg2) {
                    case 0: //opens web browser and navigates to given website
                        startActivity(new Intent(Intent.ACTION_VIEW,
                            Uri.parse("http://www.android.com/")));
                        break;
                    case 1: //opens contacts application to browse contacts
                        startActivity(new Intent(Intent.ACTION_VIEW,
                            Uri.parse("content://contacts/people/")));
                        break;
                    case 2: //opens phone dialer and fills in the given number
                        startActivity(new Intent(Intent.ACTION_VIEW,
                            Uri.parse("tel:12125551212")));
                        break;
                }
            }
        });
    }
}

```

```

        case 3: //search Google for the string
            Intent intent= new Intent(Intent.ACTION_WEB_SEARCH );
            intent.putExtra(SearchManager.QUERY, searchTerms);
            startActivity(intent);
            break;
        case 4: //starts the voice command
            startActivity(new
                Intent(Intent.ACTION_VOICE_COMMAND));
            break;
        default: break;
    }
}
});
}
}

```

2.3.6 秘诀 11：在 activity 间传递基本数据类型

我们有时需要向被调用的activity传递数据，而有时被调用的activity反过来也需要向调用它的activity回传数据。比如，游戏最后得分就需要回传给高分排行榜界面。activity之间传递信息的方式有如下几种：

- 在发起调用的activity中声明相关变量（如public int finalScore），在被调用的activity中就可以为这些变量赋值（如CallingActivity.finalScore=score）；
- 通过Bundle包附加数据（本例演示）；
- 利用Preference属性存储数据，需要时再读取（将会在第5章中阐述）；
- 利用SQLite数据库存储数据，需要时再读取（将会在第9章中阐述）。

Bundle包是字符串值到各种parcelable类型的映射。它在向intent附加属性值时创建。下例将演示如何从主activity向它所启动的activity传递数据，并且在后者修改数据之后回传结果。

在StartScreen activity中声明了两个变量（本例中为一个整形变量和一个字符串类型变量）。当创建intent调用PlayGame类时，通过putExtra方法将这两个变量传给intent。当结果从被调用的activity返回后，可以使用getExtras方法读取变量值。调用程序如清单2-16所示。

清单2-16 src/com/cookbook/passing_data_activities/StartScreen.java

```

package com.cookbook.passing_data_activities;

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

public class StartScreen extends Activity {
    private static final int PLAY_GAME = 1010;

```

```
private TextView tv;
private int meaningOfLife = 42;
private String userName = "Douglas Adams";

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tv = (TextView) findViewById(R.id.startscreen_text);

    //display initial values
    tv.setText(userName + ":" + meaningOfLife);

    //setup button listener
    Button startButton = (Button) findViewById(R.id.play_game);
    startButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            startGame();
        }
    });
}

@Override
protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {
    if (requestCode == PLAY_GAME && resultCode == RESULT_OK) {
        meaningOfLife = data.getExtras().getInt("returnInt");
        userName = data.getExtras().getString("userName");
        //show it has changed
        tv.setText(userName + ":" + meaningOfLife);
    }
    super.onActivityResult(requestCode, resultCode, data);
}

private void startGame() {
    Intent launchGame = new Intent(this, PlayGame.class);

    //passing information to launched activity
    launchGame.putExtra("meaningOfLife", meaningOfLife);
    launchGame.putExtra("userName", userName);

    startActivityForResult(launchGame, PLAY_GAME);
}
}
```

传递给PlayGame activity的变量值可以使用getIntExtra和getStringExtra方法读取。当该activity结束后调用intent回传时,我们就可以使用putExtra方法返回数据到发出调用的activity。具体调用代码见清单2-17所示。

清单2-17 src/com/cookbook/passing_data_activities/PlayGame.java

```

package com.cookbook.passing_data_activities;

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

public class PlayGame extends Activity {
    private TextView tv2;
    int answer;
    String author;

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

        tv2 = (TextView) findViewById(R.id.game_text);

        //reading information passed to this activity
        //Get the intent that started this activity
        Intent i = getIntent();
        //returns -1 if not initialized by calling activity
        answer = i.getIntExtra("meaningOfLife", -1);
        //returns [] if not initialized by calling activity
        author = i.getStringExtra("userName");

        tv2.setText(author + ":" + answer);

        //change values for an example of return
        answer = answer - 41;
        author = author + " Jr.";

        //setup button listener
        Button startButton = (Button) findViewById(R.id.end_game);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                //return information to calling activity
                Intent i = getIntent();
                i.putExtra("returnInt", answer);
                i.putExtra("returnStr", author);
                setResult(RESULT_OK, i);
                finish();
            }
        });
    }
}

```

本章将继续介绍应用程序的基本组成模块。首先，我们从分离任务的角度讲述了线程的使用，然后再介绍服务（service）和broadcast receiver，它们有时需要用到线程，这在有些秘诀里会讲到。接着介绍使用receiver的应用微件。最后我们将讨论各种alert的用法。

3.1 线程

每一个应用程序在创建时都会默认运行单一的进程，它包含了所有的任务。为了避免阻塞用户界面，需要用一个驻留在后台的额外线程去处理相对耗时的任务，例如从网上下载资源或处理大量的计算。这些工作由开发者来实现，其后Android操作系统会确定它们执行的优先次序。

大部分应用程序都可以通过使用线程来提高性能。如果比较耗时的情况没有在应用程序的设计阶段被察觉，那么在测试应用程序时它们就会很快暴露出来，因为Android操作系统在用户界面被阻塞时会弹出一个alert对话框告知用户，如图3-1所示。

3.1.1 秘诀 12：启动一个辅助线程

在该秘诀中，当屏幕上的按钮被按下时就会播放手机铃声。下面是一个简单的例子，用来说明如何在耗时的操作中使用线程。在下例中，由于没有指定单独的线程，调用play_music()方法播放音乐时会挂起应用程序。

```
Button startButton = (Button) findViewById(R.id.trigger);
startButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view){
        // BAD USAGE: function call to time-consuming
        // function causes main thread to hang
        play_music();
    }
});
```

这意味着在这首歌播放完之前，任何用户请求（例如返回到主屏幕或多次按下屏幕上的按钮）都不会被系统响应。没有响应的用户界面甚至会导致Android系统弹出如图3-1所示的错误提示框。



图3-1 当一个线程阻塞时显示的消息示例

要想解决上述问题，可以启动一个辅助线程来调用`play_music()`方法。具体步骤如下。

(1) 创建一个新线程持有`Runnable`对象：

```
Thread initBkgdThread = new Thread(
    //insert runnable object here
);
```

(2) 创建一个`Runnable`对象，并重载`run()`方法调用耗时的任务：

```
new Runnable() {
    public void run() {
        play_music();
    }
}
```

(3) 启动新线程，然后由该线程去运行那个耗时的任务：

```
initBkgdThread.start();
```

主线程很快就能完成对包含耗时任务的辅助线程的设置工作，因此它可以继续为其他事件服务。

在显示完整的activity代码之前，我们先来讨论一下其他辅助文件。本书的第6章将会详细地讨论媒体播放的内容，此处要说明的是，本例中使用的铃声格式为一种称为铃声文本传输语言(RTTTL)格式的记序序列文件。例如，清单3-1中的RTTTL码描述了比中央C调低的A调(220 Hz)四分音符。把它放在`res/raw/`目录下的一个单行的文本文件`R.raw.a4`中，并在资源中注册。

清单3-1 RTTTL文件`res/raw/a4.rtttl`，它表示A仅仅低于中央C

```
<a4:d=4,o=5,b=250:a4;
```

然后，在Activity中调用媒体播放器播放该铃声音符：

```
m_mediaPlayer = MediaPlayer.create(this, R.raw.a4);
m_mediaPlayer.start();
```

本秘诀中使用了四种不同的音符，它们分别为四个单独的RTTTL文件：`g4.rtttl`、`a4.rtttl`、`b4.rtttl`和`c5.rtttl`。它们其实都是清单3-1中`a4`的翻版，只是为了在每个案例中使用新的音符而对文件作了

相应的修改，当然它也能被扩展为其他音符或者格式。

补充一点，MediaPlayer会启动它自己的后台线程播放媒体。所以，如果要单独播放一个较长的文件，有可能会像第6章描述的那样，要避免使用前台线程。但是如果有多文件需要立即播放的话，这种方法就不奏效了，此处需要注意的是，我们并不总需要线程。

我们通过单击按钮来触发播放音乐。该按钮微件需要在主布局文件（此处为main.xml）中将其ID命名为trigger，如清单3-2所示。

清单3-2 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <Button android:id="@+id/trigger"
        android:layout_width="100dip" android:layout_height="100dip"
        android:text="Press Me"
    />
</LinearLayout>
```

启动一个新线程带来的负面影响是：即使主线程被暂停，它也会一直运行。我们都知道音乐播放是通过后台线程实现的，所以即使返回到主屏幕，音乐也会继续播放到结束为止。如果你并不想要这样做，可以在主activity的onPaused()方法中设置一个flag标记，运行play_music()方法时检查这个标记（这里命名为paused），这样当主线程暂停时就会停止播放音乐。

将前面所有的代码组合起来就形成了完整的PressAndPlay activity，如清单3-3所示。

清单3-3 src/com/cookbook/launch_thread/PressAndPlay.java

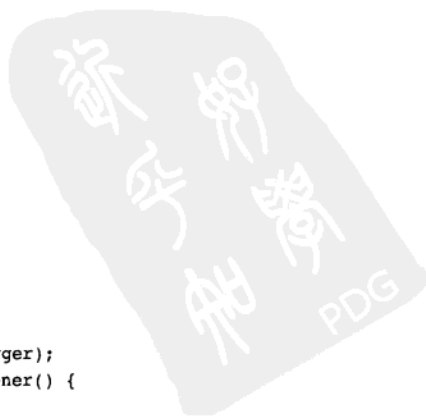
```
package com.cookbook.launch_thread;

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class PressAndPlay extends Activity {

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

        Button startButton = (Button) findViewById(R.id.trigger);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view){
```



```

        //standalone play_music() function call causes
        //main thread to hang. Instead, create
        //separate thread for time-consuming task
        Thread initBkgdThread = new Thread(new Runnable() {
            public void run() {
                play_music();
            }
        });
        initBkgdThread.start();
    }

    });
}

int[] notes = {R.raw.c5, R.raw.b4, R.raw.a4, R.raw.g4};
int NOTE_DURATION = 400; //millisec
MediaPlayer m_mediaPlayer;
private void play_music() {
    for(int ii=0; ii<12; ii++) {
        //check to ensure main activity not paused
        if(!paused) {
            if(m_mediaPlayer != null) {m_mediaPlayer.release();}
            m_mediaPlayer = MediaPlayer.create(this, notes[ii%4]);
            m_mediaPlayer.start();
            try {
                Thread.sleep(NOTE_DURATION);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

boolean paused = false;
@Override
protected void onPause() {
    paused = true;
    super.onPause();
}
@Override
protected void onResume() {
    super.onResume();
    paused = false;
}
}
}

```

注意，`Thread.sleep()`方法指定了线程要暂停的时长（单位为毫秒），在本例中休眠时长为变量`NOTE_DURATION`的值。

另外还要注意activity生命周期方法的使用惯例：添加的activity有关的逻辑应该在父方法中用

大括号括起。这是一种良好的编程习惯，以确保命令执行无误。因此在暂停activity之前，程序内部的pause标记要设定为true，在activity恢复之后将其设定为false。

3.1.2 秘诀 13：创建实现 runnable 接口的 activity

该秘诀中activity将会进行高负荷的计算，像图像边缘检查等。在此，我们虚构了一个名叫detectEdges()的方法来模拟实际的图像处理算法。

如果在onCreate()方法中自行调用detectEdges()方法，那么会挂起主线程，在运算结束前不能正常显示用户界面。因此，这里需要启动独立的线程来运行这个耗时的方法。但是考虑到本例中activity的主要目的就是执行这些耗时的操作，所以自然会想到干脆让这个activity自己去实现Runnable接口。如清单3-4所示，在onCreate()方法中声明一个后台执行的线程。后台线程启动后会调用activity的run()方法，该方法需要重载为我们预期的功能。

实例中的按钮跟秘诀12中的按钮一样，在后台任务detectEdges()运行时，通过点击按钮来证明用户界面仍然有响应。

清单3-4 src/com/cookbook/runnable_activity/EdgeDetection.java

```
package com.cookbook.runnable_activity;

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

public class EdgeDetection extends Activity implements Runnable {
    int numberOfTimesPressed=0;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final TextView tv = (TextView) findViewById(R.id.text);
        //in-place function call causes main thread to hang:
        /* detectEdges(); */
        //instead, create background thread for time-consuming task
        Thread thread = new Thread(EdgeDetection.this);
        thread.start();

        Button startButton = (Button) findViewById(R.id.trigger);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view){

                tv.setText("Pressed button " + ++numberOfTimesPressed
                    + " times\nAnd computation loop at "
                    + "(" + xi + ", " + yi + ") pixels");
            }
        })
    }
}
```

```

    });
}

@Override
public void run() {
    detectEdges();
}

//Edge Detection
int xi, yi;
private double detectEdges() {
    int x_pixels = 4000;
    int y_pixels = 3000;
    double image_transform=0;

    //double loop over pixels for image processing
    //meaningless hyperbolic cosine emulates time-consuming task
    for(xi=0; xi<x_pixels; xi++) {
        for(yi=0; yi<y_pixels; yi++) {
            image_transform = Math.cosh(xi*yi/x_pixels/y_pixels);
        }
    }
    return image_transform;
}
}
}

```

3.1.3 秘诀 14：设置线程优先级

Android系统管理线程的优先级。默认情况下，新线程myThread的优先级为5。开发者可以在调用myThread.start()前调用myThread.setPriority(priority)，为线程设定不同的优先级。这里的priority不能大于常量Thread.MAX_PRIORITY（该值为10），也不能小于常量Thread.MIN_PRIORITY（该值为1）。

3.1.4 秘诀 15：取消线程

有时当一个组件完成或被杀死后，开发者希望由它产生的线程同时也被杀死。例如，在某activity中定义一个线程：

```
private volatile Thread myThread;
```

方法myThread.stop()已经弃用，因为它可能会导致应用程序进入不可预知的状态。相反，必要时可以在父组件的onStop()方法中使用如下的方法：

```

//use to stop the thread myThread
if(myThread != null) {
    Thread dummy = myThread;
    myThread = null;
    dummy.interrupt();
}

```

在应用程序层面上，还有另外一种方法也可以达到同样的效果，即使用`setDaemon(true)`方法将所有生成的线程声明为守护线程（daemon thread）。这样可以确保如果应用程序的主线程被杀死，那么该应用程序的所有守护线程也都会被同时杀死。

```
//use when initially starting a thread
myThread.setDaemon(true);
myThread.start();
```

最后，我们总是可以通过在`run()`方法中使用`while(stillRunning)`循环，并且从循环外部设定`stillRunning=false`的方法来杀死线程。但是这种方法的缺点是不能有效地控制线程停止的时间。

3.1.5 秘诀 16：在两个应用程序之间共享线程

前面的秘诀教会了大家如何在同一个应用程序中使用多个线程。而反过来在多个应用程序之间共享一个线程有时也非常有用。例如，如果两个应用程序之间需要通信，那么它们可以使用绑定（binder）代替更复杂的进程间通信（IPC）协议。具体实现步骤如下。

(1) 出于安全考虑，确保每个应用程序在打包发布时都使用相同的密钥。

(2) 确保每个应用程序都运行在相同的用户ID下。在每个应用程序的`ActivityManifest.xml`文件中声明相同的`android:sharedUserID="my.shared.userid"`属性来保证。

(3) 声明所有相关activity或组件运行在同一进程中。这通过在`ActivityManifest.xml`文件中为每个组件声明相同的属性`Android:process="my.shared.processname"`来实现。

通过上述简单的步骤就可以确保两个组件运行在同一个线程中，并且可以彼此透明地共享一些信息。权限不共享情况下要相对复杂些，我们将在秘诀94中讲解。

3.2 线程之间的消息机制：handler

当多个线程（例如主线程和后台线程）同时运行时，它们之间就需要有一种通信方式。例如下面这些情况：

- 主线程主要负责处理时间紧迫的信息，而将耗时的处理发给后台线程处理；
- 当复杂耗时的计算执行完毕时，需要给调用线程发回运算结果。

这些都可以通过`handler`类来完成，`handler`类用于在两个线程之间传递信息。每一个`handler`都会和创建它的线程绑定，传递消息给该线程，并执行它的命令。

3.2.1 秘诀 17：从主线程调度 runnable 任务

该秘诀实现了应用程序中经常会用到的计时器功能。例如在一个游戏中，我们需要用计时器记录玩家花费多长时间过一关。此处提供了一个简单方法，用于在后台线程持续运行时处理用户交互。

计时器在应用程序的后台线程中运行，因此它不会阻塞UI线程，但是需要即时更新用户界面。如清单3-5所示，页面布局文件中有`TextView`和`Button`两个组件。`TextView`组件的ID为`text`，在开始时显示一条欢迎信息，`Button`组件的ID为`trigger`，设定其显示的文字为`Press Me`。

清单3-5 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView android:id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
    <Button android:id="@+id/trigger"
        android:layout_width="100dip" android:layout_height="100dip"
        android:text="Press Me"
    />
</LinearLayout>
```

XML布局文件中的文本资源和Java程序中activity的TextView变量相关联,该activity的ID为BackgroundTimer,使用下面的语句初始化:

```
mTimeLabel = (TextView) findViewById(R.id.text);
mButtonLabel = (TextView) findViewById(R.id.trigger);
```

在Java程序中定义这些组件后,就可以在运行时修改其显示的文本。应用程序启动时,mUpdateTimeTask开始计时,并会持续更新文本组件mTimeLabel,以几分几秒的形式显示时间。当你点击按钮后,它的onClick()方法便会重写mButtonLabel的文本,显示按钮被点击的次数。

我们创建了名为mHandler的handler对象,并用它来排列runnable对象mUpdateTimeTask。mUpdateTimeTask方法第一次被调用是在onCreate()方法中,然后它每隔200毫秒递归调用自身一次以达到持续更新时间的效果。这样做足够确保时间每秒的变化平滑显示又不增加系统调度开销。完整的activity如清单3-6所示。

清单3-6 src/com/cookbook/background_timer/BackgroundTimer.java

```
package com.cookbook.background_timer;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.SystemClock;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class BackgroundTimer extends Activity {
    //keep track of button presses, a main thread task
```

```
private int buttonPress=0;
TextView mButtonLabel;

//counter of time since app started, a background task
private long mStartTime = 0L;
private TextView mTimeLabel;

//Handler to handle the message to the timer task
private Handler mHandler = new Handler();

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

    if (mStartTime == 0L) {
        mStartTime = SystemClock.uptimeMillis();
        mHandler.removeCallbacks(mUpdateTimeTask);
        mHandler.postDelayed(mUpdateTimeTask, 100);
    }

    mTimeLabel = (TextView) findViewById(R.id.text);
    mButtonLabel = (TextView) findViewById(R.id.trigger);

    Button startButton = (Button) findViewById(R.id.trigger);
    startButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view){
            mButtonLabel.setText("Pressed " + ++buttonPress
                                + " times");
        }
    });
}

private Runnable mUpdateTimeTask = new Runnable() {
    public void run() {
        final long start = mStartTime;
        long millis = SystemClock.uptimeMillis() - start;
        int seconds = (int) (millis / 1000);
        int minutes = seconds / 60;
        seconds = seconds % 60;

        mTimeLabel.setText("" + minutes + ":"
                           + String.format("%02d",seconds));
        mHandler.postDelayed(this, 200);
    }
};

@Override
protected void onPause() {
    mHandler.removeCallbacks(mUpdateTimeTask);
    super.onPause();
}
```



```

    }

    @Override
    protected void onResume() {
        super.onResume();
        mHandler.postDelayed(mUpdateTimeTask, 100);
    }
}

```

3.2.2 秘诀 18: 使用倒数计时器

前一个秘诀是教大家如何通过 handler 实现计时器功能。Android 系统的内建类 `CountDownTimer` 提供了另外一种计时器。它将后台线程的创建和 handler 队列封装成为一个方便的类调用。

`CountDownTimer` 有两个参数, 一个是倒计时时间量, 另一个是处理 `onTick()` 回调的时间间隔, 两者均以毫秒计数。 `onTick()` 方法用于更新倒计时文本显示。除此之外, 该实例的其他部分与前一个例子都相同。完整的 activity 如清单 3-7 所示。

清单 3-7 src/com/cookbook/countdown/CountDownTimerExample.java

```

package com.cookbook.countdown;

import android.app.Activity;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class CountDownTimerExample extends Activity {
    //keep track of button presses, a main thread task
    private int buttonPress=0;
    TextView mButtonLabel;

    //count down timer, a background task
    private TextView mTimeLabel;

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

        mTimeLabel = (TextView) findViewById(R.id.text);
        mButtonLabel = (TextView) findViewById(R.id.trigger);

        new CountDownTimer(30000, 1000) {
            public void onTick(long millisUntilFinished) {
                mTimeLabel.setText("seconds remaining: "

```

```

        + millisUntilFinished / 1000);
    }
    public void onFinish() {
        mTimeLabel.setText("done!");
    }
}.start();

Button startButton = (Button) findViewById(R.id.trigger);
startButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view){
        mButtonLabel.setText("Pressed " + ++buttonPress + " times");
    }
});
}
}
}

```

3.2.3 秘诀 19: 处理耗时的初始化工作

常常可以看到,应用程序启动时需要先进行耗时的初始化工作,本秘诀就教大家如何解决这个问题。应用程序启动之初会显示一个特定的Loading...启动画面,该启动画面在布局文件loading.xml中设置。本例中的启动画面仅显示一条简单的文字信息,如清单3-8所示,但在实际应用程序中可能会包含公司的标志或者起始页面的动画。

清单3-8 res/layout/loading.xml

```

<?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">
    <TextView android:id="@+id/loading"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Loading..."
    />
</LinearLayout>

```

在显示这一布局的同时,我们启动一个后台线程来执行耗时的initializeArrays()方法,以免用户界面被挂起。我们在初始化过程中使用了静态变量,确保画面切换或activity的其他实例不需要重新计算数据。

初始化工作完成以后,会发送一条消息给mHandler。由于发送消息的动作是我们需要的所有信息,在此将通过mHandler.sendMessage(0)发送一条空的消息。

一旦收到消息,UI线程就会运行handleMessage()方法。我们需要重载该方法,以在初始化以后能够继续执行activity,本例中我们通过布局文件main.xml设置了应用程序的主屏幕。完整的activity如清单3-9所示。

清单3-9 src/com/cookbook/handle_message/HandleMessage.java

```

package com.cookbook.handle_message;

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

public class HandleMessage extends Activity implements Runnable {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.loading);

        Thread thread = new Thread(this);
        thread.start();
    }

    private Handler mHandler = new Handler() {
        public void handleMessage(Message msg) {
            setContentView(R.layout.main);
        }
    };

    public void run(){
        initializeArrays();
        mHandler.sendMessage(0);
    }

    final static int NUM_SAMPS = 1000;
    static double[][] correlation;
    void initializeArrays() {
        if(correlation!=null) return;

        correlation = new double[NUM_SAMPS][NUM_SAMPS];
        //calculation
        for(int k=0; k<NUM_SAMPS; k++) {
            for(int m=0; m<NUM_SAMPS; m++) {
                correlation[k][m] = Math.cos(2*Math.PI*(k+m)/1000);
            }
        }
    }
}

```

3

3.3 服务

服务 (service) 是一种Android组件, 它主要在系统的后台运行, 并且没有用户交互。它可以

被其他任何组件启动或停止。service在运行时可以被其他组件绑定。当然service也可以自行停止。下面是一些service的使用情境。

- 某个activity允许用户选择一组音乐文件，然后它启动一个service去播放这些文件。在播放期间，一个新的activity启动，绑定该service，使得用户可以改变歌曲列表或者停止播放。
- 某个activity启动一个service向某个网站上传一组图片。一个新的activity启动并且绑定现有的service，确定哪个图片文件正被上传，并把这张图片显示在屏幕上。
- 某个broadcast receiver收到一条信息：用户拍摄了一张照片，随即启动service上传这张新照片到某个网站。其后broadcast receiver转为非激活状态，并被系统杀死并回收内存，但是service会继续运行，一旦上传完成后，service就会自动停止。

通常情况下，服务的生命周期如图3-2所示。

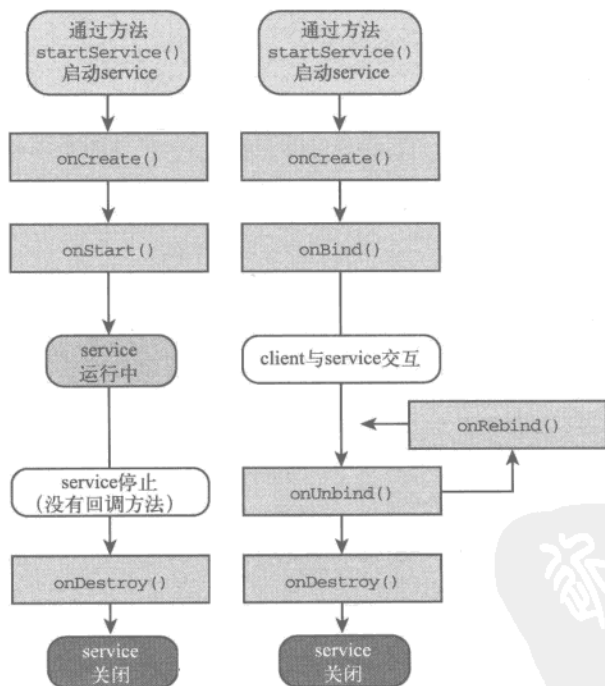


图3-2 service生命周期图，引自<http://developer.android.com/>

顺便再说一下第三种情境：杀死某个组件时，该组件的所有后台任务都会被杀死。因此，如果在组件停止后仍需要继续运行后台任务的话，就要启动一个service，以确保操作系统知道该进程还在活跃地运行任务。

所有的service都继承自抽象类Service或它的子类。和activity相似，每个service的入口点都是onCreate()方法。service中没有暂停的概念，不过可以通过调用onDestroy()方法停止service。

秘诀 20：创建一个完整的 service

创建一个与单个组件相关的完整service的步骤如下。

(1) 创建一个新类扩展Service类。(在Eclipse中, 右键点击你的工程, 选择New→Class, 并指定android.app.Service为父类。)

(2) 通过添加类似下面的语句, 在AndroidManifest.xml文件中声明该service (在Eclipse中应该会被上一步自动完成): `<service Android:name=".myService"></service>`。

(3) 重载onCreate()和onDestroy()方法。(在Eclipse中, 右键单击你的类文件, 选择Source→Override/Implement Methods..., 然后勾选onCreate()和onDestroy()方法。)这两个方法分别包含了service启动和停止时要执行的功能。

(4) 重载onBind()方法, 这样新组件创建后可以通过这个方法绑定该service。

(5) 通过外部触发器启动service。service不能自行启动, 必须通过外部组件或者以某种方式触发。例如某个组件可以创建一个intent来调用startService()和stopService()分别启动和停止service。

为了说明前面的内容, 清单3-10展示了一个简单的service范例, 它使用了本章秘诀12中的play_music()方法, 解释如下。

- 在service启动或停止时使用Toast显示提示信息。
- onBind()方法被重载, 但是没有被使用 (可以根据需要对程序进行扩展)。
- 为了防止UI线程被阻塞, 仍然需要创建一个用于播放音乐的新线程。
- activity被销毁 (例如, 改变屏幕方向) 或被暂停时 (例如, 按下home键), service都不会停止运行。因此可以看出, 尽管service由activity启动, 但是service自己就是运行主体。

清单3-10 src/com/cookbook/simple_service/SimpleService.java

```
package com.cookbook.simple_service;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.widget.Toast;

public class SimpleService extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    boolean paused = false;
    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this, "Service created ...",
```



```

        Toast.LENGTH_LONG).show();
        paused = false;
        Thread initBkgdThread = new Thread(new Runnable() {
            public void run() {
                play_music();
            }
        });
        initBkgdThread.start();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Service destroyed ...",
            Toast.LENGTH_LONG).show();
        paused = true;
    }

    int[] notes = {R.raw.c5, R.raw.b4, R.raw.a4, R.raw.g4};
    int NOTE_DURATION = 400; //millisec
    MediaPlayer m_mediaPlayer;
    private void play_music() {
        for(int ii=0; ii<12; ii++) {
            //check to ensure main activity not paused
            if(!paused) {
                if(m_mediaPlayer != null) {m_mediaPlayer.release();}
                m_mediaPlayer = MediaPlayer.create(this, notes[ii%4]);
                m_mediaPlayer.start();
                try {
                    Thread.sleep(NOTE_DURATION);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

现在AndroidManifest.xml文件中既声明了activity又声明了service，如清单3-11所示。

清单3-11 AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.simple_service"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SimpleActivity"

```

```

        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=".SimpleService"></service>
</application>
<uses-sdk android:minSdkVersion="3" />
</manifest>

```

范例中的activity设置了用户界面去启动和停止service，如清单3-12所示，两个按钮的布局文件如清单3-13所示。

清单3-12 src/com/cookbook/simple_service/SimpleActivity.java

```

package com.cookbook.simple_service;

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

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

        Button startButton = (Button) findViewById(R.id.Button01);
        startButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view){
                startService(new Intent(SimpleActivity.this,
                                        SimpleService.class));
            }
        });

        Button stopButton = (Button) findViewById(R.id.Button02);
        stopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v){
                stopService(new Intent(SimpleActivity.this,
                                      SimpleService.class));
            }
        });
    }
}

```

清单3-13 res/layout/main.xml

```

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

```

```

        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
/>
<Button android:text="Do it" android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>
<Button android:text="Stop it" android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>
</LinearLayout>

```

3.4 添加 broadcast receiver

broadcast receiver 监听广播消息并触发相应的事件。下面是一些 Android 操作系统发出的广播事件范例：

- 按下相机按钮；
- 电池快没电了；
- 安装新应用程序；

用户生成的组件也可以发送广播，例如：

- 完成某个计算；
- 启动某个特定线程。

所有的 broadcast receiver 都继承自 BroadcastReceiver 抽象类或其子类。broadcast receiver 的生命周期很简单，receiver 每收到一条消息时便会调用 onReceive() 方法，运行完该方法时 BroadcastReceiver 实例就即刻进入非活动状态。

broadcast receiver 通常会启动一个独立的组件或在它的 onReceive() 方法中发送一条通知给用户，这点在本章稍后介绍。如果 broadcast receiver 需要处理耗时的任务，就应该启动一个服务而不是线程，因为非活动的 broadcast receiver 会被系统杀死。

秘诀 21：在按下相机键时启动 service

本秘诀演示了如何启动基于广播事件的服务，例如当按下相机键时。broadcast receiver 需要监听特定的事件，然后启动服务。broadcast receiver 本身被另外的组件启动。（本例中，作为一个独立的 activity 执行，名为 SimpleActivity。）

清单 3-14 所示的 activity 创建了一个 broadcast receiver，并且设置了一个 intent，该 intent 过滤相机按钮事件。同时为了更好地演示该功能，还增加了“安装新包”消息（package-added messages）过滤器。然后，启动 broadcast receiver，使用 registerReceiver() 方法将该 intent 过滤器传递

给它。

清单3-14 src/com/cookbook/simple_receiver/SimpleActivity.java

```
package com.cookbook.simple_receiver;

import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

public class SimpleActivity extends Activity {
    SimpleBroadcastReceiver intentReceiver =
        new SimpleBroadcastReceiver();

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

        IntentFilter intentFilter =
            new IntentFilter(Intent.ACTION_CAMERA_BUTTON);
        intentFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
        registerReceiver(intentReceiver, intentFilter);
    }

    @Override
    protected void onDestroy() {
        unregisterReceiver(intentReceiver);
        super.onDestroy();
    }
}
```

此处需要注意的是，如果销毁activity，receiver也要被释放掉。虽然这不是必须的，但是这么做却很有用。BroadcastReceiver组件的代码如清单3-15所示。为了检查所有广播事件，仅需要重载onReceive()生命周期方法。如果它匹配了指定的事件（此处为ACTION_CAMERA_BUTTON事件），服务就会在activity原有的上下文中启动。

清单3-15 src/com/cookbook/simple_receiver/SimpleBroadcastReceiver.java

```
package com.cookbook.simple_receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class SimpleBroadcastReceiver extends BroadcastReceiver {
    @Override
```

```
public void onReceive(Context rcvContext, Intent rcvIntent) {  
    String action = rcvIntent.getAction();  
    if (action.equals(Intent.ACTION_CAMERA_BUTTON)) {  
        rcvContext.startService(new Intent(rcvContext,  
            SimpleService2.class));  
    }  
}  
}
```

清单3-15中的SimpleBroadcastReceiver启动的service显示在清单3-16中。该service仅在其启动或停止时通过Toast显示信息。

清单3-16 src/com/cookbook/simple_receiver/SimpleService2.java

```
package com.cookbook.simple_receiver;  
  
import android.app.Service;  
import android.content.Intent;  
import android.os.IBinder;  
import android.widget.Toast;  
  
public class SimpleService2 extends Service {  
    @Override  
    public IBinder onBind(Intent arg0) {  
        return null;  
    }  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Toast.makeText(this, "Service created ...",  
            Toast.LENGTH_LONG).show();  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        Toast.makeText(this, "Service destroyed ...",  
            Toast.LENGTH_LONG).show();  
    }  
}
```

3.5 应用微件

应用微件 (App Widget) 是嵌入到应用程序中具有图标外观的视图。为了周期性地更新视图, 它继承了broadcast receiver的子类。应用微件简称微件, 它们可以通过长按 (按下不放) 触摸屏空白区域嵌入到主屏幕等其他应用程序中。这会弹出菜单选择在哪个位置安装微件。长按微件并

拖动它到垃圾箱就可以删除它。总之，创建微件需要具备如下条件。

- 一个描述微件外观的视图。在XML布局资源文件中定义，包含文本、背景和其他布局参数。
- 一个应用微件提供程序，用于接收广播事件和连接微件并更新其视图。
- 应用微件的详细信息，如大小、更新频率等。注意，主屏幕被划分为 4×4 的单元格，所以通常一个微件会占用若干个单元格大小（通常在竖屏模式下为 80×100 dp，横屏模式下为 106×74 dp）。
- 这是一个可选项。你可以定义一个activity用于配置微件的各项参数。该activity在微件创建时加载。

秘诀 22：创建应用微件

本秘诀将创建一个简单的应用微件，用于在主屏幕上显示一些文本信息。设置文本信息每秒更新一次，但要注意的是，默认状态下Android系统会强制设定最短更新时间为每30分钟更新一次。这样可以避免一些有编程问题的微件耗尽电池。清单3-17实现了一个AppWidgetProvider，它是BroadcastReceiver的子类。程序重载了其主要的方法onUpdate()，在系统更新微件时调用该方法。

清单3-17 src/com/cookbook/widget/example/SimpleWidgetProvider.java

```
package com.cookbook.simple_widget;

import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.widget.RemoteViews;

public class SimpleWidgetProvider extends AppWidgetProvider {
    final static int APPWIDGET = 1001;
    @Override
    public void onUpdate(Context context,
        AppWidgetManager appWidgetManager, int[] appWidgetIds) {
        super.onUpdate(context, appWidgetManager, appWidgetIds);
        // Loop through all widgets to display an update
        final int N = appWidgetIds.length;
        for (int i=0; i<N; i++) {
            int appWidgetId = appWidgetIds[i];
            String titlePrefix = "Time since the widget was started:";
            updateAppWidget(context, appWidgetManager, appWidgetId,
                titlePrefix);
        }
    }

    static void updateAppWidget(Context context, AppWidgetManager
        appWidgetManager, int appWidgetId, String titlePrefix) {
        Long millis = System.currentTimeMillis();
        int seconds = (int) (millis / 1000);
```

```

        int minutes = seconds / 60;
        seconds     = seconds % 60;

        CharSequence text = titlePrefix;
        text += " " + minutes + ":" + String.format("%02d",seconds));

        // Construct the RemoteViews object.
        RemoteViews views = new RemoteViews(context.getPackageName(),
            R.layout.widget_layout);
        views.setTextViewText(R.id.widget_example_text, text);

        // Tell the widget manager
        appWidgetManager.updateAppWidget(appWidgetId, views);
    }
}

```

清单3-18是描述微件详细信息的XML文件。它描述了微件在主屏幕上显示的大小和单位为毫秒的更新周期（系统默认的最小值为30分钟）。

清单3-18 src/res/xml/widget_info.xml

```

<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dp"
    android:minHeight="72dp"
    android:updatePeriodMillis="1000"
    android:initialLayout="@layout/widget_layout">
</appwidget-provider>

```

描述微件外观的视图也被定义在一个XML文件中，如清单3-19所示。

清单3-19 src/res/layout/widget_layout.xml

```

<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget_example_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#ff000000"
    android:background="#ffffffff"
/>

```

3.6 alert 对话框

alert对话框是一种在应用程序用户界面之外快速为用户提供信息的方法。它可以显示在一个叠加窗口中，如Toast消息提示框或弹出对话框，也可以显示在屏幕上方的通知栏中。Toast使用一行代码就可以在屏幕上显示消息，而不需要定义布局文件。出于该原因，它也是一种便捷的调试工具，相当于C语言中printf语句。

3.6.1 秘诀 23：使用 Toast 在屏幕上显示简短消息

Toast方法在前面章节中已经简单介绍过：

```
Toast.makeText(this, "text", Toast.LENGTH_SHORT).show();
```

以上代码也可以写成多行命令：

```
Toast tst = Toast.makeText(this, "text", Toast.LENGTH_SHORT);
tst.show();
```

第二种方式在需要多次显示信息时会非常有用，因为第一行创建的实例可以被重复使用。

另外两种使用多行Toast命令的情况是：重新定位文本显示位置或为Toast添加图片。为了重新设置文本位置，或者使Toast在屏幕上居中显示，在调用show()方法之前使用setGravity方法：

```
tst.setGravity(Gravity.CENTER, tst.getXOffset() / 2,
tst.getYOffset() / 2);
```

在Toast中添加图片的代码如下：

```
Toast tst = Toast.makeText(this, "text", Toast.LENGTH_LONG);
ImageView view = new ImageView(this);
view.setImageResource(R.drawable.my_figure);
tst.setView(view);
tst.show();
```

3.6.2 秘诀 24：使用 alert 对话框

使用AlertDialog类可以向用户显示对话框，并且可最多使用三个不同动作的按钮。范例如下。

- “你的最终得分是80/100。你可以重玩此关，进入下一关，或是返回主菜单。”
- “图片文件已损坏，请另选一张或取消。”

本秘诀就以第一个例子说明应用程序执行什么动作取决于哪个按钮被点击。示例代码如清单3-20所示。

AlertDialog使用create()方法初始化；使用setMessage()方法指定显示的文本；在setButton()方法中定义三个按钮上的显示文字及点击它们执行的相应动作；最后，调用show()方法在屏幕上显示对话框。注意，此处的onClick()回调函数中的逻辑仅仅是作为例子来说明如何定义按钮动作。

清单3-20 AlertDialog示例

```
AlertDialog dialog = new AlertDialog.Builder(this).create();

dialog.setMessage("Your final score: " + mScore + "/" + PERFECT_SCORE);
dialog.setButton(DialogInterface.BUTTON_POSITIVE, "Try this level again",
    new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            mScore = 0;
            start_level();
        }
    })
```

```

    });
    dialog.setButton(DialogInterface.BUTTON_NEGATIVE, "Advance to next level",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                mLevel++;
                start_level();
            }
        });
    dialog.setButton(DialogInterface.BUTTON_NEUTRAL, "Back to the main menu",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                mLevel = 0;
                finish();
            }
        });
    dialog.show();

```

这样就创建了一个弹出对话框，如图3-3所示。注意，按钮显示的顺序是BUTTON_POSITIVE、BUTTON_NEUTRAL、BUTTON_NEGATIVE。如果对话框只需要两个选项或一个选项，不要指定全部三个按钮选项。

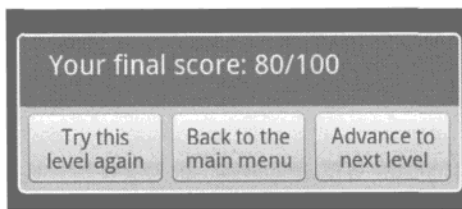


图3-3 一个对话框实例

3.6.3 秘诀 25：在状态栏中显示通知

状态栏横跨设备的屏幕顶部，其功能是向用户显示待阅读的通知，以使用户方便的时候阅读。一般来说，由于activity主要是用来和用户交互的，所以service利用此功能的时候要更多一些。通常，通知应该是言简意赅的，这样才会有更好的用户体验。

创建状态栏通知的一般步骤如下所示。

(1) 声明一个通知并指定其在状态栏的显示方式：

```

String ns = Context.NOTIFICATION_SERVICE;
mNManager = (NotificationManager) getSystemService(ns);
final Notification msg = new Notification(R.drawable.icon,
    "New event of importance",
    System.currentTimeMillis());

```

(2) 定义状态栏打开时的外观细节，并且声明单击时应执行的动作（通过PendingIntent类来声明该动作）：

```

Context context = getApplicationContext();
CharSequence contentTitle = "ShowNotification Example";
CharSequence contentText = "Browse Android Cookbook Site";
Intent msgIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://www.pearson.com"));
PendingIntent intent =
    PendingIntent.getActivity>ShowNotification.this,
        0, msgIntent,
        Intent.FLAG_ACTIVITY_NEW_TASK);

```

(3) 添加其他配置信息，例如是否闪烁LED，播放声音或在选中后自动清除。后两者的代码如下所示：

```

msg.defaults |= Notification.DEFAULT_SOUND;
msg.flags |= Notification.FLAG_AUTO_CANCEL;

```

(4) 将通知事件的信息添加到系统中：

```

msg.setLatestEventInfo(context,
    contentTitle, contentText, intent);

```

(5) 在相应事件发生时，通过唯一标识触发某个事件的通知：

```

mNManager.notify(NOTIFY_ID, msg);

```

(6) 通知完成后，如需要可以使用相同的标识清除该通知。

如果信息内容发生改变，应该更新通知而不是发送另外一个通知。可以通过更新步骤2中的相关信息来实现，然后再重新调用setLatestEventInfo方法。下面是一个在activity中显示通知的示例，代码如清单3-21所示。

清单3-21 src/com/cookbook/show_notification/ShowNotification.java

```

package com.cookbook.show_notification;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class ShowNotification extends Activity {

    private NotificationManager mNManager;
    private static final int NOTIFY_ID=1100;

    /** Called when the activity is first created. */
    @Override

```

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

    String ns = Context.NOTIFICATION_SERVICE;
    mNManager = (NotificationManager) getSystemService(ns);
    final Notification msg = new Notification(R.drawable.icon,
        "New event of importance",
        System.currentTimeMillis());

    Button start = (Button)findViewById(R.id.start);
    Button cancel = (Button)findViewById(R.id.cancel);

    start.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            Context context = getApplicationContext();
            CharSequence contentTitle = "ShowNotification Example";
            CharSequence contentText = "Browse Android Cookbook Site";
            Intent msgIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://www.pearson.com"));
            PendingIntent intent =
                PendingIntent.getActivity(context, 0, msgIntent,
                    Intent.FLAG_ACTIVITY_NEW_TASK);

            msg.defaults |= Notification.DEFAULT_SOUND;
            msg.flags |= Notification.FLAG_AUTO_CANCEL;

            msg.setLatestEventInfo(context,
                contentTitle, contentText, intent);
            mNManager.notify(NOTIFY_ID, msg);
        }
    });

    cancel.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            mNManager.cancel(NOTIFY_ID);
        }
    });
}
```


Android用户界面（以下简称UI）由屏幕视图组件、屏幕触摸事件和键盘事件（key press）组成。为了使用户界面适配各种不同的Android设备，系统提供了一个框架用于定义用户界面。本章着重讲解如何利用该框架初始化图形布局并做后续改动，而关于处理按键和手势事件的内容则放到第5章中讲解。

4.1 资源目录及其基本属性

用户界面会涉及Android工程目录结构中由开发者生成的资源文件，其中一些已经在第2章中讨论过。但为了更完整地阐述，我们将整个资源目录总结如下。

- res/anim/——存放逐帧动画或过门动画对象。
- res/drawable/——存放图片资源。注意这些图片会在编译过程中修改和优化。
- res/layout/——存放定义界面布局的扩展标记语言XML文件。
- res/values/——存放描述资源的XML文件。和其他资源目录一样，文件名可以任意定义，但通常为本书所使用的arrays.xml、colors.xml、dimens.xml、strings.xml和styles.xml。
- res/xml/——存放前面未提及的其他XML文件。
- res/raw/——存放前面未提及的其他资源，包括在编译过程中不可以被修改或优化的图片资源。

每个UI对象都包括三个自定义属性以定制UI的外观，它们是：对象的大小、对象中显示的文本和对象的颜色。表4-1是这三个通用UI属性的取值范围。注意，在设置尺寸的时候最好使用独立于设备的dp或sp作为单位。

表4-1 三大UI基本属性的可能取值

属 性	取 值
Dimension	任何数字，后面加上下面的尺寸单位： px——屏幕上的实际像素 dp（或者dip）——基于屏幕密度的抽象单位，与设备像素无关，在每英寸160点的显示屏上，1dp=1px sp——与设备无关的像素，可以根据用户字体大小偏好缩放 in——屏幕的物理尺寸，表示英寸 mm——屏幕的物理尺寸，表示毫米 pt——屏幕的物理尺寸，表示一个大小为1/72英寸的点

(续)

属 性	取 值
String	任何字符串, 单引号和双引号需要转义: Don't worry 任何正确引用的字符串: "Don't worry" 格式化的字符串, 比如: Population: %1\$d ^① 可包含HTML标签, 如、<i>或者<u> 可包含特殊字符, 如通过值©给出字符©
Color	可用值包括: 12位的#rgb格式颜色值, 带有alpha透明通道的16位#argb格式颜色值, 24位的#rrgbb格式颜色值, 或者是带有alpha透明通道的32位#aarrgbb颜色值。也可以使用Color类中预定义的颜色: 比如Color.CYAN

为了统一应用程序的视觉效果, 这些属性最好统一定义在一个全局文件中。也可以以后重新定义这些属性, 因为它们集中在以下三个文件中。

- ❑ 在XML资源文件res/values/dimens.xml中声明控件的尺寸大小。比如:
 - XML声明——<dimen name="large">48sp</dimen>
 - XML引用——@dimen/large
 - Java 引用——getResources().getDimension(R.dimen.large)
- ❑ 在XML资源文件res/values/strings.xml中声明控件的文本标签。比如:
 - XML声明——<string name="start_pt">I\'m here</string>
 - XML引用——@string/start_pt
 - Java引用——getBaseContext().getString(R.string.start_pt)
- ❑ 在XML资源文件res/values/colors.xml中声明控件的颜色。比如:
 - XML声明——<color name="red">#f00</color>
 - XML引用——@color/red
 - Java引用——getResources().getColor(R.color.red)

秘诀 26: 声明替代资源

上节描述的资源是Android系统使用的默认通用配置。开发者也可以通过修饰符区别不同的配置以便为其声明不同取值。

为了支持多种语言, 字符串值可以被翻译成其他语言并放在相应语言的values目录下。例如美式英语、英式英语、法语、简体中文(中国大陆)、繁体中文(中国台湾)和德语, 可以通过如下方式添加。

```
res/values-en-rUS/strings.xml
res/values-en-rGB/strings.xml
res/values-fr/strings.xml
res/values-zh-rCN/strings.xml
res/values-zh-rTW/strings.xml
res/values-de/strings.xml
```

① 表示十进制数字, 字符串为%2\$s。——译者注

并非所有的字符串都需要在这些文件中重新定义。如果所选语言文件中没有定义某字符串值，应用程序会返回到默认的res/values/strings.xml文件中查找，所以该文件必须包含应用程序所有相关字符串资源。如果任何drawable资源包含文本，并且需要一个特定的语言表单，类似的目录结构对其也适用（如res/drawables-zh-hdpi/）。

为兼容多种屏幕像素密度，drawable和raw资源（如果需要的话）可以缩放并被存放在不同dpi值的多个目录资源供使用。例如，某个图片文件可能在下面的每个文件目录中都存在：

```
res/drawable-ldpi/
res/drawable-mdpi/
res/drawable-hdpi/
res/drawable-nodpi/
```

低、中、高屏幕像素密度分别为120 dpi、160 dpi和240 dpi。并非所有的dpi选项都需要设定。Android系统会在运行时选择最接近的drawable，并适当调整其大小。对于位图图像，我们可用nodpi选项设定其不被缩放。如果语言和dpi都需要设置，文件夹可以同时添加这两种修饰符：drawable-cn-rUS-mdpi/。

Android设备具有的不同屏幕类型，我们已经在第1章讨论过。我们常常会针对不同屏幕类型定义不同的XML布局，常用修饰符如下。

- 纵向和横向屏幕方向：-port和-land
- 常规屏幕（QVGA、HVGA和VGA）和宽屏（WQVGA、FWVGA和WVGA）：-notlong和-long
- 小屏幕（屏幕尺寸不到3英寸）、正常屏幕（屏幕尺寸不到4.5英寸）和大屏幕（屏幕尺寸在4.5英寸以上）：-small，-normal以及-large

如果未定义屏幕方向和纵横比，Android系统会根据屏幕情况自动缩放用户界面（虽然并不总是能够做到恰到好处）。然而如果定义了不同屏幕类型的布局，则需要Android Manifest XML文件中的application中添加如下的特定元素，以确保系统对其的正常支持。

```
<supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:resizable="true"
    android:anyDensity="true" />
```

要注意如果android:minSdkVersion或者android:targetSdkVersion的值为3（Android 1.5），那么默认情况下只有android:normalScreens（G1手机的屏幕）被设置为true。因此为应用程序显式声明supports-screens元素是非常有必要的，以便新型手机可以正确有效地缩放UI。

4.2 view 和 viewGroup

view是图形界面的基本构成模块。每个view描述一个View对象，每个View对象占据一个矩形区域并负责处理该区域内的事件。View是那些与用户交互对象的基类，一般称这些对象为微件。微件包括按钮和复选框等。

ViewGroup对象是一个特殊的View，它是一个可以容纳多个View（或者ViewGroup）的容器。如图4-1，我们可以看出ViewGroup可以按需要将一组view和widget水平或垂直摆放。ViewGroup是设置屏幕布局的一个基类。

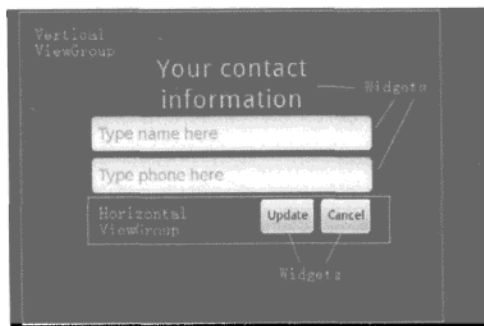


图4-1 包含ViewGroup和微件的view示例

布局定义了用户界面元素、它们的位置及其行为。它可以在XML中或是Java程序中定义。通常在XML文件中声明初始布局，运行时再由Java程序来调整。这样做可以享受使用XML定义View和ViewGroup对象的整体布局的简便性，又能获得在应用程序中利用Java修改组件的灵活性。

在Android程序中将XML布局和Java activity分开定义的另一个好处是，可以方便地根据屏幕方向、设备类型（手机或是平板电脑）、语言环境（英语或是中文）作出不同的响应。这些自定义项可以被抽象出来放置在不同的XML资源文件中，而不用去更改底层的activity。

4.2.1 秘诀 27：利用 Eclipse 编辑器生成布局

我们可以使用Eclipse中的图形化布局编辑器（graphical layout editor）快速生成布局。新建一个activity并打开它的XML布局资源文件，此处为main.xml。然后单击Layout选项卡，展示图形化布局效果。单击黑色屏幕，删除屏幕上所有东西以便重新生成一个布局，步骤如下。

(1) 从左侧的Layouts Selector中拖放一个布局控件到屏幕区域。例如，我们可以选择TableLayout，它可以以列表栏的样式容纳多个View或ViewGroup。

(2) 点击并拖动其他的布局控件使其嵌套放置到TableLayout中。例如，我们可以选择TableRow，它可以以行的方式排列容纳多个View或ViewGroup。在本例中需要添加三个TableRow。

(3) 在Outline大纲视图中分别右键单击每个TableRow，然后从Views Selector中为其添加view元素。比如，在第一个TableRow中添加一个Button和CheckBox，第二个TableRow中添加两个TextView，而在第三个TableRow中则添加了一个TimePicker。

(4) 在TableRow下面添加一个Spinner和VideoView视图控件。

此时的界面外观如图4-2所示，可以切换横向和纵向屏幕显示模式观察布局变化。点击main.xml选项卡则会显示清单4-1中的XML代码。该方法提供了一种创建Android用户界面外观的便捷方式。

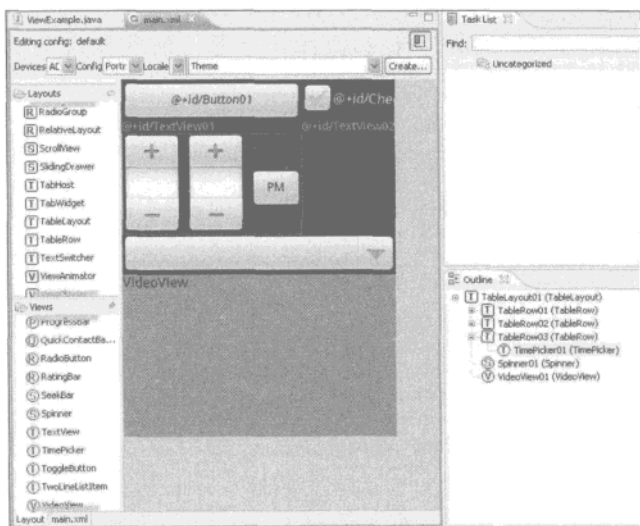


图4-2 Eclipse中显示的Android布局生成器示例

清单4-1 main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:id="@+id/TableLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TableRow android:id="@+id/TableRow01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <Button android:text="@+id/Button01"
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <CheckBox android:text="@+id/CheckBox01"
            android:id="@+id/CheckBox01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
    <TableRow android:id="@+id/TableRow02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView android:text="@+id/TextView01"
            android:id="@+id/TextView01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView android:text="@+id/TextView02"

```

```

        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</TableRow>
<TableRow android:id="@+id/TableRow03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TimePicker android:id="@+id/TimePicker01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</TableRow>
<Spinner android:id="@+id/Spinner01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<VideoView android:id="@+id/VideoView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</TableLayout>

```

另外一种查看界面布局的方法是使用Hierarchy Viewer工具。在模拟器中运行一个应用程序，然后从命令行窗口中运行hierarchyviewer命令。该工具存放在软件开发工具包安装目录下的tools/目录。考虑到安全因素，这种方法只能在模拟器上使用，因为在真机中运行hierarchyviewer可能会泄露安全设置。单击目标窗口，选择Load View Hierarchy。将会生成各个布局控件的层级关系视图。本例的显示结果如图4-3所示。



图4-3 清单4-1示例的Android hierarchy viewer视图

4.2.2 秘诀 28：控制 UI 元素的宽度和高度

本秘诀将介绍如何指定UI元素的宽度和高度，从而调整整体布局。每个View对象都必须指定其全宽`android:layout_width`和全高`android:layout_height`，可采用以下三种方式之一：

- `exact dimension`——精确控制，但对不同屏幕类型的兼容性不好；
- `wrap_content`——高或宽大小刚好可以包住元素内容加上padding；
- `fill_parent`——最大化元素尺寸可以占满父容器的空间，包括padding。

padding是元素周围的空白区域，如果没有明确指定，默认值为0。它是UI元素大小尺寸的一个部分，必须被声明为某个确切尺寸，可以使用下面两种属性指定：

- `padding`——设置元素四边的空白都等于某值；
- `paddingLeft`、`paddingRight`、`paddingTop`、`paddingBottom`——分别设置元素各边的空白宽度。

此外还有一个属性`android:layout_weight`，可以定义为某个数值。它可用来在Android系统中界定不同布局元素的相对重要性。

清单4-2中的`main.xml`布局文件为一个带有4个按钮的线性界面布局。4个按钮设置为屏幕上水平分布，如图4-4所示。

清单4-2 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:text="add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <Button android:text="subtract"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <Button android:text="multiply"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
    <Button android:text="divide"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```

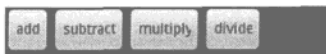


图4-4 清单4-2中代码生成了按线性布局水平排列的4个按钮

如果将add按钮的高度设置为fill_parent，按钮就会垂直填满父组件，同时这4个按钮的文本会水平对齐。如果将某个按钮的宽度设置为fill_parent，那么它后面的按钮就会被挤出视图，如图4-5所示。

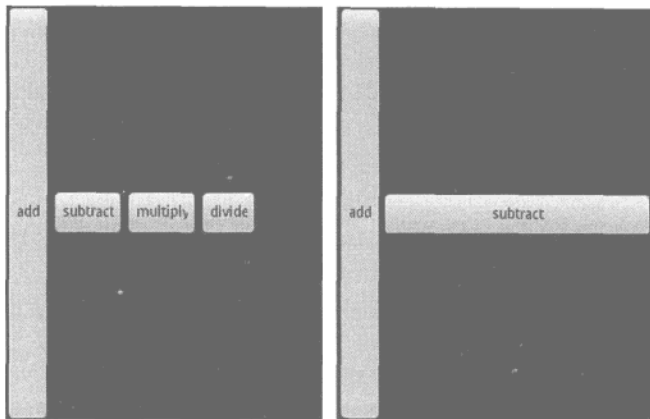


图4-5 将某个按钮高度设置为fill_parent还会保持水平布局，但如果将宽度设置为fill_parent将会挤掉剩余按钮

图4-4还有一个问题就是multiply和divide按钮的最后一个字符都没有显示全。这个问题可以通过在文本后面添加一个空格来解决。然而更常用的做法是通过布局来解决这个问题。请看图4-6所示的各种按钮格式。

图4-6所示的4行按钮布局的方式如下。

- 第1行按钮和清单4-2的布局相同，只是在每个单词的末尾追加一个空格。

- 第2行中的最后一个按钮的宽度设置为fill_parent，为按钮提供足够大的空间，但是这种方式不适用于前面的按钮，否则将会和图4-5显示得一样，将剩余按钮挤掉：

```
<Button android:text="divide"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
/>
```

- 第3行为multiply按钮添加了padding，从而使按钮尺寸更大，但是没有在文本结尾追加空格，因为文本的宽度和高度设置都为wrap_content：

```
<Button android:text="multiply"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingRight="20sp"
/>
```

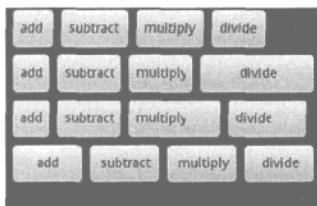


图4-6 使用多种方式调整4个按钮的显示布局

- 第4行所有按钮都设置为fill_parent，但同时都添加了layout_weight属性，并为4个按钮设定了同样的值。使之成为最令人满意的布局：

```
<Button android:text="add"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
/>
<Button android:text="subtract"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
/>
<Button android:text="multiply"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
/>
<Button android:text="divide"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
/>
```

4.2.3 秘诀 29：设置相对布局和布局 ID

有时对象的位置可以通过前面的起始对象或父对象的相对位置设定，这样做比使用绝对值更为方便。而且，若UI控件设定线性布局方式，使用相对布局也许更为简单。清单4-3所示的布局就可以用RelativeLayout视图实现。布局界面如图4-7所示。



图4-7 RelativeLayout布局案例的4个text view

清单4-3 RelativeLayout示例

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/mid" android:text="middle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"/>
    <TextView android:id="@+id/high" android:text="high"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/mid"/>
    <TextView android:id="@+id/low" android:text="low"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true">
```

```

        android:layout_below="@id/mid"/>
<TextView android:id="@+id/left" android:text="left"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@id/high"
        android:layout_toLeftOf="@id/low"/>
</RelativeLayout>

```

关于相对布局的各个属性的解释和使用规则见表4-2。由于各种布局方式既可以在XML文件中声明也可以在Java代码中声明，所以表中对这两种设置界面布局方法都作了介绍。表中前3行显示的属性需要指向某个视图的ID，而最后两行的属性值为布尔值。

表4-2 相对布局内子控件的使用规则

相对布局规则	XML属性（都以android:标签开始）	Java常量
将该视图的边缘相对于给定ID的视图的边缘对齐	layout_above	ABOVE
	layout_below	BELOW
	layout_toRightOf	RIGHT_OF
	layout_toLeftOf	LEFT_OF
将该视图的边缘和给定ID视图的边缘对齐	layout_alignTop	ALIGN_TOP
	layout_alignBottom	ALIGN_BOTTOM
	layout_alignRight	ALIGN_RIGHT
	layout_alignLeft	ALIGN_LEFT
将该视图的文本基线和给定ID视图的文本基线对齐	layout_alignBaseline	ALIGN_BASELINE
将该视图的边缘和父视图的边缘对齐	layout_alignParentTop	ALIGN_PARENT_TOP
	layout_alignParentBottom	ALIGN_PARENT_BOTTOM
	layout_alignParentRight	ALIGN_PARENT_RIGHT
	layout_alignParentLeft	ALIGN_PARENT_LEFT
将该视图定位于父视图的中心	layout_centerInParent	CENTER_IN_PARENT
	layout_centerHorizontal	CENTER_HORIZONTAL
	layout_centerVertical	CENTER_VERTICAL

4.2.4 秘诀 30：通过编程声明布局

通过Android系统的XML布局框架定义布局是我们优先选择的方法，这样不但能具有良好的设备兼容性还可以简化开发过程。但是有时候通过编程，即使用Java代码设置布局也非常有用。事实上，所有的布局都可以通过Java声明。为了便于说明，清单4-4向我们展示了如何使用Java代码实现前面秘诀中的布局。但需要强调的是，通过Java代码设置布局不但繁琐，而且这种方式并没有利用资源目录的模块化方式，不能在不更改Java代码的前提下简单地修改布局，如同秘诀26描述的那样。因此，我们通常不鼓励使用这种方式。

清单4-4 src/com/cookbook/programmaticlayout/ProgrammaticLayout.java

```
package com.cookbook.programmatic_layout;
```

```
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;
import android.widget.RelativeLayout;
import android.widget.TextView;

public class ProgrammaticLayout extends Activity {
    private int TEXTVIEW1_ID = 100011;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        //Here is an alternative to: setContentView(R.layout.main);
        final RelativeLayout relLayout = new RelativeLayout( this );
        relLayout.setLayoutParams( new RelativeLayout.LayoutParams(
            LayoutParams.FILL_PARENT,
            LayoutParams.FILL_PARENT ) );

        TextView textView1 = new TextView( this );
        textView1.setText("middle");
        textView1.setTag(TEXTVIEW1_ID);

        RelativeLayout.LayoutParams text1layout = new
            RelativeLayout.LayoutParams( LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT );
        text1layout.addRule( RelativeLayout.CENTER_IN_PARENT );
        relLayout.addView(textView1, text1layout);
        TextView textView2 = new TextView( this );
        textView2.setText("high");

        RelativeLayout.LayoutParams text2Layout = new
            RelativeLayout.LayoutParams( LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT );
        text2Layout.addRule(RelativeLayout.ABOVE, TEXTVIEW1_ID );
        relLayout.addView( textView2, text2Layout );

        setContentView( relLayout );
    }
}
```

4.2.5 秘诀 31：使用独立线程更新布局

前面的第3章中我们提到：当某个耗时的activity运行时，必须确保UI线程处于响应状态。我们可以创建一个新线程分担耗时的任务，并使UI线程始终保持在较高优先级。如果这个新线程需要更新UI，就可以使用handler向UI线程发送更新信息。

本秘诀使用按钮触发一个分为两段的耗时计算，每段计算完成后都会更新UI。应用程序使用的布局如同清单4-5中的main.xml，包括一个用于显示状态的文本框computation_status和一个

用于触发的按钮action。应用程序使用的字符串定义在strings.xml文件中，如清单4-6所示。

清单4-5 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/computation_status"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" android:textSize="36sp"
        android:textColor="#000" />
    <Button android:text="@string/action"
        android:id="@+id/action"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

清单4-6 res/layout/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HandlerUpdateUi!</string>
    <string name="app_name">HandlerUpdateUi</string>
    <string name="action">Press to Start</string>
    <string name="start">Starting...</string>
    <string name="first">First Done</string>
    <string name="second">Second Done</string>
</resources>
```

后台线程更新UI界面的步骤如下：

- (1) 初始化一个handle以通过后台线程更新UI对象（本例中为av）；
 - (2) 定义一个在需要时更新UI的runnable接口（本例中为mUpdateResults）；
 - (3) 声明一个handler对象处理线程间的消息（本例中为mHandler）；
 - (4) 在后台线程中设置适当的标志用来通告状态的变化（在本例中，text_string和background_color将会被改变）；
 - (5) 后台线程通过hander通知主线程更新UI。
- 使用这些步骤的activity如清单4-7所示。

清单4-7 src/com/cookbook/handler_ui/HandlerUpdateUi.java

```
package com.cookbook.handler_ui;

import android.app.Activity;
import android.graphics.Color;
```

```

import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class HandlerUpdateUi extends Activity {
    TextView av; //UI reference
    int text_string = R.string.start;
    int background_color = Color.DKGRAY;

    final Handler mHandler = new Handler();
    // Create runnable for posting results to the UI thread
    final Runnable mUpdateResults = new Runnable() {
        public void run() {
            av.setText(text_string);
            av.setBackgroundColor(background_color);
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        av = (TextView) findViewById(R.id.computation_status);

        Button actionButton = (Button) findViewById(R.id.action);
        actionButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                do_work();
            }
        });
    }

    //example of a computationally intensive action with UI updates
    private void do_work() {
        Thread thread = new Thread(new Runnable() {
            public void run() {
                text_string=R.string.start;
                background_color = Color.DKGRAY;
                mHandler.post(mUpdateResults);

                computation(1);
                text_string=R.string.first;
                background_color = Color.BLUE;
                mHandler.post(mUpdateResults);

                computation(2);
                text_string=R.string.second;
            }
        });
    }
}

```



```

        background_color = Color.GREEN;
        mHandler.post(mUpdateResults);
    }
});
thread.start();
}

final static int SIZE=1000; //large enough to take some time
double tmp;
private void computation(int val) {
    for(int ii=0; ii<SIZE; ii++)
        for(int jj=0; jj<SIZE; jj++)
            tmp=val*Math.log(ii+1)/Math.log1p(jj+1);
}
}
}

```

4.3 文本操作

在视图控件中（如TextView、EditText以及Button）添加文本，要添加到XML布局文件中的android:text元素中。正如本章开始所述，最好养成习惯将所有字符串都集中定义在string.xml文件中。向UI控件，如TextView，添加文本的代码如下所示。

```

<TextView android:text="@string/myTextString"
    android:id="@+id/my_text_label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```

默认字体取决于Android设备和用户偏好设置。如果要确切指定某种字体，可以使用表4-3中的元素。

表4-3 TextView属性表，最后一列中的默认属性值加粗显示

TextView属性	XML元素	Java方法	值域及默认值
显示字符串	android:text	setText(CharSequence)	Any string
字体大小	android:textSize	setTextSize(float)	Any dimension
字体颜色	android:textColor	setTextColor(int)	Any color
背景颜色	N/A	setBackgroundColor(int)	Any color
字体样式	android:textStyle	setTypeface(Typeface)	bold italic bold italic
字体类型	android:typeface	setTypeface(Typeface)	normal sans serif monospace
文本在显示区域中的位置	android:gravity	setGravity(int)	top bottom left right (更多)

4.3.1 秘诀 32：设置和更改文本属性

该秘诀向我们演示如何通过单击按钮改变文本颜色。它也可以容易地扩展为改变字体大小或样式，我们将会在秘诀的最后讨论这部分内容。

清单4-8显示的主页面布局中只有一个TextView和一个Button控件，采用垂直分布的LinearLayout线性布局方式。文本控件ID定义为mod_text，其显示内容为string.xml文件所定义的changed_text字符串，如清单4-9所示。而按钮的ID定义为change，其显示的文本也定义在清单4-9的字符串XML文件中，是ID为button_text的字符串。

清单4-8 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:text="@string/changed_text"
        android:textSize="48sp"
        android:id="@+id/mod_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:text="@string/button_text"
        android:textSize="48sp"
        android:id="@+id/change"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

清单4-9 res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ChangeFont</string>
    <string name="changed_text">Rainbow Connection</string>
    <string name="button_text">Press to change the font color</string>
</resources>
```

清单4-10中的activity使用main.xml中的布局，并标识TextView指向mod_text。然后重载按钮的OnClickListener方法，设置文本颜色如表4-3所示。所需的颜色资源定义在全局文件colors.xml中，如清单4-11所示。根据代码定义，颜色有红色（red）、绿色（green）和蓝色（blue），但我们根据功能相应命名为开始（start）、中间（mid）和结束（last）。采用这种处理方式无需更改控件视图名称就可以随意更换颜色，非常便捷。

清单4-10 src/com/cookbook/change_font/ChangeFont.java

```
package com.cookbook.change_font;

import android.app.Activity;
```

```

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class ChangeFont extends Activity {
    TextView tv;
    private int color_vals[]={R.color.start, R.color.mid, R.color.last};
    int idx=0;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.mod_text);

        Button changeFont = (Button) findViewById(R.id.change);
        changeFont.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                tv.setTextColor(getResources().getColor(color_vals[idx]));
                idx = (idx+1)%3;
            }
        });
    }
}

```

清单4-11 res/values/colors.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="start">#f00</color>
    <color name="mid">#0f0</color>
    <color name="last">#00f</color>
</resources>

```

我们可以修改该秘诀，用来轻松更改字体大小（或者样式）。只要将color_vals[]替换为size_vals[]并指向R.dimen资源即可：

```

private int size_vals[]={R.dimen.small, R.dimen.medium, R.dimen.large};
tv.setTextSize(getResources().getDimension(size_vals[idx]));

```

同样也要把colors.xml替换成dimens.xml，如清单4-12所示。

清单4-12 dimens.xml文件类似用法示例

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="small">12sp</dimen>
    <dimen name="medium">24sp</dimen>
    <dimen name="large">48sp</dimen>
</resources>

```


如果将本秘诀换做改变文本内容，可以将color_vals[]替换成text_vals[]，并指向R.string资源：

```
private int text_vals[]={R.string.first_text,
                        R.string.second_text, R.string.third_text};
tv.setText(getBaseContext().getString(text_vals[idx]));
```

所使用的strings.xml文件如清单4-13所示。

清单4-13 string.xml文件类似用法示例

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ChangeFont</string>
    <string name="changed_text">Rainbow Connection</string>
    <string name="button_text">Press To Change the Font Color</string>
    <string name="first_text">First</string>
    <string name="second_text">Second</string>
    <string name="third_text">Third</string>
</resources>
```

4.3.2 秘诀 33：提供文本输入

EditText类是一个供用户输入文本的简单视图控件。其声明方法类似于TextView，常见属性见表4-4。它的每个属性也都有对应的Java方法，此处就不再具体阐述了。

表4-4 EditText的常见属性，对表4-3中内容的补充。同样，最后一列的默认属性值加粗显示

EditText属性	XML元素	值域和默认值
最小显示行数	android:minLines	任意整数
最大显示行数	android:maxLines	任意整数
值为空时的提示文本	android:hint	任意字符串
输入类型	android:inputType	text textCapSentences textAutoCorrect textAutoComplete textEmailAddress textNoSuggestions textPassword number phone date time (更多)

例如，在布局文件使用下面的XML代码，将会在文本输入框中显示灰色的提示语Type text here。对于那些没有输入键盘或者屏幕键盘被隐藏的设备，当选中编辑框时就会弹出软键盘用于输入文本，效果如图4-8所示。

```
<EditText android:id="@+id/text_result"
    android:inputType="text"
    android:textSize="30sp"
```

```

android:hint="Type text here"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />

```

当用户选中输入框，使用`android:inputType="phone"`会调出输入电话号码的软键盘，而使用`android:inputType="textEmailAddress"`则会相应调出输入Email地址的软键盘。上述内容如图4-9所示，同时输入框中的提示文本也会相应变化。



图4-8 使用软键盘输入文字



图4-9 当

还有一点要注意，我们可以按表4-4所示的那样声明文本输入方法，如：输入英文语句时自动将句首字母转化为大写；自动纠正拼写错误的单词；输入文本时关闭文本提示。根据输入文本的情境控制这个选项非常有用。

4.3.3 秘诀 34：创建表单

表单是一种图形化布局，具有文本输入或者文本选择的区域。要想输入文本可以使用`EditText`控件。在声明该控件之后，需要用一段Java代码在程序运行时捕捉输入的文本。具体方法如清单4-14所示。注意，本例中输入的文本`textResult`不能修改。如果需要修改，就要生成文本内容的副本再作相应的修改。

清单4-14 从`EditText`对象捕获文本

```

CharSequence phoneNumber;
EditText textResult = (EditText) findViewById(R.id.text_result);
textResult.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        // register the text when "enter" is pressed
        if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
            (keyCode == KeyEvent.KEYCODE_ENTER)) {
            // grab the text for use in the activity
            phoneNumber = textResult.getText();
            return true;
        }
    }
});

```

```

        return false;
    }
});

```

若此处onKey()方法的返回值为true，则是向父方法表明按键事件已经完成，后面无需再做进一步的处理。

用户可在表单中使用多种控件，我们可以在此使用一些标准微件，如复选框、单选按钮和下拉选择菜单等，具体使用方法见下一节内容。

4.4 其他控件：从按钮到拖动条

Android系统提供了一些标准图形控件，开发人员可以使用它们营造完整的应用程序用户体验。常用控件包括以下几种。

- 按钮 (Button) —— 是一种矩形图形，点击其边界内区域即可点中，其可包含用户自定义的文本和图像。
- 复选框 (CheckBox) —— 是一种显示对号标记的图形按钮，配有说明文本，触摸可切换选中 and 取消状态。ToggleButton的使用与它相似，我们将放在一起讨论。
- 单选按钮 (RadioButton) —— 是一种圆点状按钮，触摸可以选中，选中后就不能被取消。利用RadioGroup可以将多个单选按钮分组，一个RadioGroup在同一时刻只能有一个按钮处于选中状态。
- 下拉列表 (Spinner) —— 是一种显示当前选中内容的按钮，按钮后方配有触发下拉菜单的箭头。当触摸选中下拉列表时会列出可选项。重新选择后，就会在下拉列表中显示新选项。
- 进度条 (ProgressBar) —— 是一种加亮的横条，可以直观地呈现运行过程中某项进程的百分比（也可选择使用辅助进度条）。进度条不具有交互性。如果进程不可量化，则可以设置为不确定模式，显示为不断旋转的圆圈。
- 拖动条 (SeekBar) —— 是一种具有交互功能的进度条，用户可以拖动改变进度。拖动条有助于显示媒体播放状态。它能显示已经播放了多少内容，用户还可以向前和向后拖动播放文件内容。

以下是这些控件的实际应用案例。

4.4.1 秘诀 35：在表格布局中使用图像按钮

前面在第2章中我们已经介绍过按钮。和其他视图一样，可以通过android:background属性为按钮添加背景图像。但使用专门的图像按钮ImageButton控件，布局会更灵活一些。该控件使用android:src属性指定图像。代码如下：

```

<ImageButton android:id="@+id/imagebutton0"
    android:src="@drawable/android_cupcake"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```

使用上述方式，图像会位于按钮控件的顶部。ImageButton继承自ImageView类，可根据android:scaleType的属性值决定图像位置。该属性的可选值和对图片的修饰如图4-10所示。

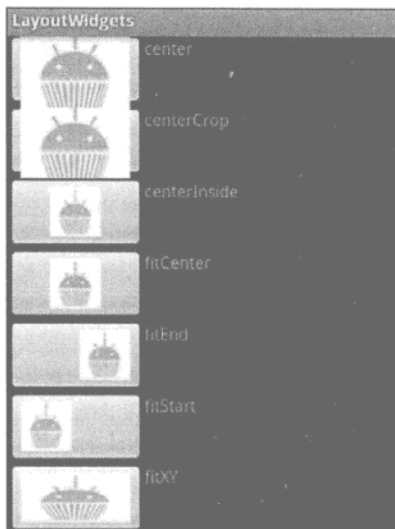


图4-10 图像视图使用android:scaleType属性的结果示例

除此以外，还可以对图像按钮作其他设置：

- 通过android:padding属性防止按钮交叉重叠，或增加按钮之间的空隙；
- 设定android:background属性值为null（在XML布局文件中为@null）以将按钮隐藏，只显示图像。

按钮被隐藏后，在默认情况下按下图像按钮不会有任何视觉反馈。这种情况可以通过创建仅包含一个选择元素的XML可绘制文件来纠正：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/myImage_pressed"
        android:state_pressed="true" />
    <item android:drawable="@drawable/myImage_focused"
        android:state_focused="true" />
    <item android:drawable="@drawable/myImage_normal" />
</selector>
```

以上代码定义了按钮的按下状态、获得焦点和正常状态，分别显示三个不同的图像。在案例中，这三个不同的图像应当放在可绘制资源目录下（如res/drawable-mdpi/）。其后通过图像按钮的android:src属性指定相应的图像文件。

当一个布局页面中包含多个图像按钮时，使用表格布局更方便，如本秘诀中显示的这样。TableLayout视图组有点像垂直方向的LinearLayout布局。可以通过每行使用一个TableRow视

图组定义多行。清单4-15中的示例布局在每一行都声明了一个ImageButton图像按钮和一个TextView，最后生成的布局界面如图4-11所示。

清单4-15 res/layout/ibutton.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TableRow>
        <ImageButton android:id="@+id/imagebutton0"
            android:src="@drawable/android_cupcake"
            android:scaleType="fitXY"
            android:background="@null"
            android:padding="5dip"
            android:layout_width="wrap_content"
            android:layout_height="90dip" />
        <TextView android:text="Cupcake"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
        <ImageButton android:id="@+id/imagebutton1"
            android:src="@drawable/android_donut"
            android:scaleType="fitXY"
            android:background="@null"
            android:padding="5dip"
            android:layout_width="wrap_content"
            android:layout_height="90dip" />
        <TextView android:text="Donut"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
        <ImageButton android:id="@+id/imagebutton2"
            android:src="@drawable/android_eclair"
            android:scaleType="fitXY"
            android:background="@null"
            android:padding="5dip"
            android:layout_width="wrap_content"
            android:layout_height="90dip" />
        <TextView android:text="Eclair"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
        <ImageButton android:id="@+id/imagebutton3"
            android:src="@drawable/android_froyo"
```

```

        android:scaleType="fitXY"
        android:background="@null"
        android:padding="5dip"
        android:layout_width="wrap_content"
        android:layout_height="90dip" />
<TextView android:text="FroYo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</TableRow>
<TableRow>
    <ImageButton android:id="@+id/imagebutton4"
        android:src="@drawable/android_gingerbread"
        android:scaleType="fitXY"
        android:background="@null"
        android:padding="5dip"
        android:layout_width="wrap_content"
        android:layout_height="90dip" />
    <TextView android:text="Gingerbread"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</TableRow>
</TableLayout>

```



图4-11 使用TableLayout布局定义ImageButton和TextView

4.4.2 秘诀 36：使用复选框和开关按钮

复选框控件预定义了对号图形、选中或未选中状态时对号图形的颜色以及按下时的颜色，为Android应用程序提供了统一的外观风格。然而，如果需要自定义图形来标识选项，可以通过 `setButtonDrawable()` 方法来实现。

述及此处的复选框范例，需要在布局文件中声明CheckBox控件，内容见清单4-16。代码中的android:text属性定义了复选框后面显示的文本标签。为了便于说明，布局文件中还添加了几个TextView控件。

清单4-16 res/layout/ckbox.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox0"
        android:text="Lettuce"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <CheckBox android:id="@+id/checkbox1"
        android:text="Tomato"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <CheckBox android:id="@+id/checkbox2"
        android:text="Cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:text="Lettuce, Tomato, Cheese choices:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:id="@+id/status"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

如清单4-17所示，布局文件中的视图控件可以与Java文件中的视图实例相关联。在此，我们通过一个私有的内部类登记三明治的馅料。三个复选框都设置了onClickListener，以跟踪记录馅料的变化，最终更新TextView控件中显示的文本。最终的输出结果如图4-12所示。

清单4-17 src/com/cookbook/layout_widgets/CheckBoxExample.java

```
package com.cookbook.layout_widgets;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.CheckBox;
import android.widget.TextView;

public class CheckBoxExample extends Activity {
    private TextView tv;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.ckbox);
    tv = (TextView) findViewById(R.id.status);

    class Toppings {private boolean LETTUCE, TOMATO, CHEESE;}
    final Toppings sandwichToppings = new Toppings();
    final CheckBox checkbox[] = {
        (CheckBox) findViewById(R.id.checkbox0),
        (CheckBox) findViewById(R.id.checkbox1),
        (CheckBox) findViewById(R.id.checkbox2)};

    checkbox[0].setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (((CheckBox) v).isChecked()) {
                sandwichToppings.LETTUCE = true;
            } else {
                sandwichToppings.LETTUCE = false;
            }
            tv.setText(""+sandwichToppings.LETTUCE + " "
                +sandwichToppings.TOMATO + " "
                +sandwichToppings.CHEESE + " ");
        }
    });
    checkbox[1].setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (((CheckBox) v).isChecked()) {
                sandwichToppings.TOMATO = true;
            } else {
                sandwichToppings.TOMATO = false;
            }
            tv.setText(""+sandwichToppings.LETTUCE + " "
                +sandwichToppings.TOMATO + " "
                +sandwichToppings.CHEESE + " ");
        }
    });
    checkbox[2].setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (((CheckBox) v).isChecked()) {
                sandwichToppings.CHEESE = true;
            } else {
                sandwichToppings.CHEESE = false;
            }
            tv.setText(""+sandwichToppings.LETTUCE + " "
```



```

        +sandwichToppings.TOMATO + " "
        +sandwichToppings.CHEESE + " "};
    }
}
}

```



图4-12 显示选中或未选中状态下的复选框示例

开关按钮 (toggle button) 类似于复选框，但其显示的图像不同。除此之外，开关按钮的文本集成到按钮图形中而不是旁边。我们可以修改清单4-16（同样包括清单4-17），使用开关按钮替代复选框，代码如下。

```

<ToggleButton android:id="@+id/ToggleButton0"
    android:textOff="No Lettuce"
    android:textOn="Lettuce"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```

请注意开关按钮使用android:textoff（如果没有声明，默认值为OFF）和android:texton（如果没有声明，默认值为ON）元素取代了复选框的android:text元素，根据开关按钮是否选中来决定显示内容。输出结果如图4-13所示。



图4-13 一组选中或者未选中状态的开关按钮范例

4.4.3 秘诀 37：使用单选按钮

单选按钮 (radio button) 类似于不可取消选择的复选框。选中一个单选按钮就会取消前面按钮的选中状态。利用RadioGroup视图组可以组合多个按钮，确保在一个RadioGroup中只能有一个按钮处于选中状态。布局文件如清单4-18所示。

清单4-18 res/layout/rbutton.xml

```
<?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">
    <RadioGroup android:id="@+id/RadioGroup01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <RadioButton android:text="Republican"
            android:id="@+id/RadioButton02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton android:text="Democrat"
            android:id="@+id/RadioButton03"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <RadioButton android:text="Independent"
            android:id="@+id/RadioButton01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </RadioGroup>
</LinearLayout>
```

activity页面与前面清单4-17示例类似，只不过是复选框替换成单选按钮。清单4-18中的布局显示效果如图4-14所示。

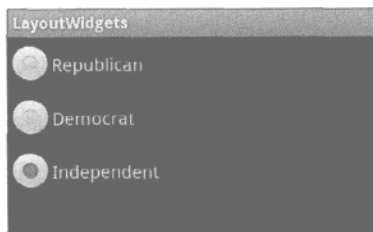


图4-14 三个单选按钮组成的RadioGroup示例

4.4.4 秘诀 38：创建下拉菜单

spinner是一种下拉菜单。该控件通常的布局定义方式如清单4-19所示。

清单4-19 res/layout/spinner.xml

```
<?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">
    <Spinner android:id="@+id/spinner"
        android:prompt="@string/ocean_prompt"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

下拉菜单的标题可以通过android:prompt属性定义。使用的字符串需要在strings.xml文件中定义，例如：

```
<string name="ocean_prompt">Choose your favorite ocean</string>
```

下拉菜单也需要独立的布局文件来定义下拉菜单的外观，本例的spinner_entry.xml内容如清单4-20所示。

清单4-20 res/layout/spinner_entry.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center"
    android:textColor="#000"
    android:textSize="40sp"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
</TextView>
```

注意下拉菜单的内容不局限于文本，还可以使用其他对象，包括图像或其他布局文件支持的对象。

调用下拉菜单的activity需要声明一个Adapter适配器，以便用布局文件定义的视图填充下拉菜单。清单4-21是一个activity页面的示例。

清单4-21 src/com/cookbook/layout_widgets/SpinnerExample.java

```
package com.cookbook.layout_widgets;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.Spinner;

public class SpinnerExample extends Activity {
    private static final String[] oceans = {
        "Pacific", "Atlantic", "Indian",
```

```

        "Arctic", "Southern" });

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.spinner);

    Spinner favoriteOcean = (Spinner) findViewById(R.id.spinner);

    ArrayAdapter<String> mAdapter = new
        ArrayAdapter<String>(this, R.layout.spinner_entry, oceans);
    mAdapter.setDropDownViewResource(R.layout.spinner_entry);
    favoriteOcean.setAdapter(mAdapter);
}
}

```

在上面例子中，下拉菜单的列表选项由字符串数组oceans[]定义，再传递给ArrayAdapter构造方法。这种方式假定下拉菜单的列表项在运行时不会更改。如要适应更一般的情况，添加和操作下拉菜单的内容，就需要使用mAdapter的add()方法。onCreate()方法中的加粗代码部分要改为如下所示。

```

Spinner favoriteOcean = (Spinner) findViewById(R.id.spinner);
ArrayAdapter<String> mAdapter = new
    ArrayAdapter<String>(this, R.layout.spinner_entry);
mAdapter.setDropDownViewResource(R.layout.spinner_entry);
for(int idx=0; idx<oceans.length; idx++){
    mAdapter.add(oceans[idx]);
}
favoriteOcean.setAdapter(mAdapter);

```

现在，该ArrayAdapter允许在运行时通过add()、remove()和clear()方法更改列表选项，还可以使用getView()方法为每个菜单项重用布局视图选项以提高程序运行的速度。

4.4.5 秘诀 39：使用进度条

本秘诀将演示如何使用进度条(progress bar)。前面清单4-7中通过更新文本来显示计算进度，而在此我们将使用更为形象的图形化方式来标识进度。我们可以在布局文件中添加进度条对象来实现。如下所示：

```

<ProgressBar android:id="@+id/ex_progress_bar"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="270px"
    android:layout_height="50px"
    android:progress="0"
    android:secondaryProgress="0" />

```

随着进度的变化，android:progress属性值也会随之变化，横跨屏幕的亮橙色进度条也会对应地不断向前延伸。可选属性android:secondaryProgress显示一个浅色的进度条，例如，可用来标识进度的阶段。

清单4-22是一个更新进度条的activity范例。和清单4-7类似，只是使用了进度条控件。在该例中通过Java代码的更新结果函数更新进度。

清单4-22 src/com/cookbook/handler_ui/HandlerUpdateUi.java

```
package com.cookbook.handler_ui;

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

public class HandlerUpdateUi extends Activity {
    private static ProgressBar m_progressBar; //UI reference
    int percent_done = 0;

    final Handler mHandler = new Handler();
    // Create runnable for posting results to the UI thread
    final Runnable mUpdateResults = new Runnable() {
        public void run() {
            m_progressBar.setProgress(percent_done);
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        m_progressBar = (ProgressBar) findViewById(R.id.ex_progress_bar);

        Button actionButton = (Button) findViewById(R.id.action);
        actionButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                do_work();
            }
        });
    }

    //example of a computationally intensive action with UI updates
    private void do_work() {
        Thread thread = new Thread(new Runnable() {
            public void run() {
                percent_done = 0;
                mHandler.post(mUpdateResults);

                computation(1);
                percent_done = 50;
                mHandler.post(mUpdateResults);
            }
        });
        thread.start();
    }
}
```

```

        computation(2);
        percent_done = 100;
        mHandler.post(mUpdateResults);
    }
    });
    thread.start();
}

final static int SIZE=1000; //large enough to take some time
double tmp;
private void computation(int val) {
    for(int ii=0; ii<SIZE; ii++)
        for(int jj=0; jj<SIZE; jj++)
            tmp=val*Math.log(ii+1)/Math.log1p(jj+1);
}
}

```

如果要让进度更新得更频繁一点，可以使用调用对象的`postDelayed`方法代替`post`方法，并在`Runnable`接口`mUpdateResults`的后面添加`postDelayed`方法（类似于前面第3章的秘诀17）。

4.4.6 秘诀 40：使用拖动条

拖动条（seek bar）类似于进度条，可以根据用户输入更新进度值。当前进度通过一个小滑块显示，我们称之为thumb。用户可以点击或拖拉小滑块，直观地设置进度。主activity的代码如清单4-23所示。

清单4-23 src/com/cookbook/seekbar/SeekBarEx.java

```

package com.cookbook.seekbar;

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

public class SeekBarEx extends Activity {
    private SeekBar m_seekBar;
    boolean advancing = true;

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

        m_seekBar = (SeekBar) findViewById(R.id.SeekBar01);
        m_seekBar.setOnSeekBarChangeListener(new
            SeekBar.OnSeekBarChangeListener() {
            public void onProgressChanged(SeekBar seekBar,
                int progress, boolean fromUser) {

```

```

        if(fromUser) count = progress;
    }

    public void onStartTrackingTouch(SeekBar seekBar) {}
    public void onStopTrackingTouch(SeekBar seekBar) {}
});
Thread initThread = new Thread(new Runnable() {
    public void run() {
        show_time();
    }
});
initThread.start();
}

int count;
private void show_time() {
    for(count=0; count<100; count++) {
        m_seekBar.setProgress(count);

        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
}
}
}

```

布局XML文件中声明了控件，如清单4-24所示。注意，本例没有使用默认的滑块，而是使用一个杯形蛋糕的图像代替，效果如图4-15所示。

清单4-24 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="24sp" android:text="Drag the cupcake"
        android:layout_alignParentTop="true" />
    <SeekBar android:id="@+id/SeekBar01"
        android:layout_centerInParent="true"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:thumb="@drawable/pink_cupcake_no_bg" />
</RelativeLayout>

```

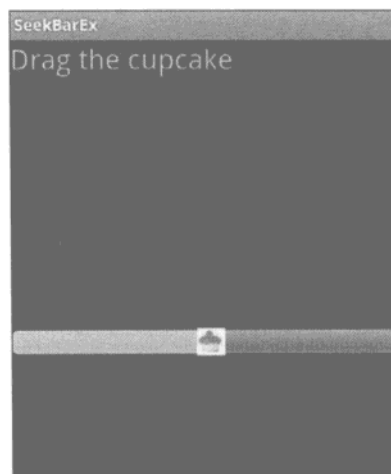


图4-15 使用自定义的杯形蛋糕图像代替拖动条上的小滑块



用户界面包含了屏幕布局（screen layout）和事件处理（event handling）两部分内容。先前在第4章中，我们已经探讨了如何利用文本框和按钮等View视图对象完成页面布局。本章将主要介绍如何处理用户交互事件，如按键事件（key press）、触摸事件（touch event）和菜单导航事件（menu navigation）等。此外，本章还会涉及某些高级用户界面库的使用方法，如手势（gesture）和3D图形等。

5.1 事件处理器和事件监听器

大多数与Android设备进行的用户交互都由系统捕获，然后再传递给相应的回调方法去处理。譬如，当我们按下物理按键“返回键”时就会调用onBackPressed()方法。这些事件可以通过继承事件处理器类event handler并重载相应方法来实现。

与View或ViewGroup对象相关的用户交互同样也支持事件监听器。监听方法会等待某个已注册的事件发生，然后触发系统向回调方法发送事件信息。例如，我们可以为某个按钮注册事件监听器setOnClickListener()，按下按钮时，就会调用它的onClick()方法。

应该尽可能的使用事件监听器，因为这样做可以避免过度继承。此外，实现事件监听接口的activity可以直接获取一个其包含的所有内置布局对象的回调方法，从而使代码更为简洁。本章将通过处理物理按键事件及屏幕触摸事件来讲解事件处理器和事件监听器。

5.1.1 秘诀 41：截取物理按键事件

标准的Android设备具有多个物理按键，它们都可以触发相应的事件，具体如表5-1所示。

表5-1 Android设备上可能包含的物理按键

物理按键	按键事件	功能描述
电源按钮	KEYCODE_POWER	开启设备或从休眠状态中唤醒设备；锁定屏幕
后退按钮	KEYCODE_BACK	导航至先前的屏幕
菜单按钮	KEYCODE_MENU	显示当前应用程序的菜单
主页按钮	KEYCODE_HOME	导航至首页面
搜索按钮	KEYCODE_SEARCH	在当前应用程序中启动搜索功能

(续)

物理按键	按键事件	功能描述
照相按钮	KEYCODE_CAMERA	启动相机
音量键	KEYCODE_VOLUME_UP KEYCODE_VOLUME_DOWN	根据上下文情境调节媒体音量（通话时的语音音量，播放音乐时的媒体回放音量，或者来电铃声音量）
方向键	KEYCODE_DPAD_CENTER KEYCODE_DPAD_UP KEYCODE_DPAD_DOWN KEYCODE_DPAD_LEFT KEYCODE_DPAD_RIGHT	设备的方向键（有些设备没有）
轨迹球	-	设备上的方向操纵杆（有些设备没有）
Keyboard	KEYCODE_0, ..., KEYCODE_9, KEYCODE_A, ..., KEYCODE_Z	设备上的下拉式键盘（有些设备没有）
媒体播放按钮	KEYCODE_HEADSETHOOK	耳机上的播放/暂停按键

系统首先会把KeyEvent事件发送给获得焦点的activity或者视图中的相应回调方法，这些回调方法包括：

- onKeyUp()、onKeyDown()、onKeyLongPress()——按下物理按键时调用的回调方法；
- onTrackballEvent()、onTouchEvent()——轨迹球移动和屏幕触摸事件发生时调用的回调方法；
- onFocusChanged()——当某个视图组件获得或者失去焦点时调用的回调方法。

这些方法都可以被应用程序重载，从而自定义不同的事件动作。比如，若要关闭相机按钮（以防不小心按到），只需要在activity的onKeyDown()回调方法中消耗^①相应的事件即可。通过如下代码可以截获KeyEvent.KEYCODE_CAMERA事件，并返回true值：

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_CAMERA) {
        return true; // consume event, hence do nothing on camera button
    }
    return super.onKeyDown(keyCode, event);
}
```

被程序消耗后，该事件就不会再传递给其他Android组件处理。但也存在一些特殊的情况。

- 电源按钮和HOME按键由系统捕获，普通应用程序无法自定义动作。
- BACK、MENU、HOME和SEARCH按键事件不能在KeyDown方法中截获，而应在KeyUp回调方法中处理。这点和Android 2.0的建议一致，因为某些平台上并没有这些按键。

清单5-1显示了截取物理按键事件的一组示例，如下：

- 相机按钮和DPAD向左按键由onKeyDown()方法截获并将消息显示在屏幕上，然后返回

① 此处的原文为consume，翻译为“消耗”，意思是按键事件在程序中被使用，然后消失。后文中的“消耗”均是此意。——译者注

true完成对事件的调用;

- 截获音量增加按键事件并在屏幕上显示相应消息, 由于事件并没有被消耗 (返回值为 false), 因此也真正地增加了音量;
- SEARCH键由onKeyDown()方法捕获, 并利用startTracking()方法追踪直到按键被松开, 这时显示消息到屏幕;
- BACK按键事件通过onBackPressed()方法捕获。

关于最后一项的说明: Android的可用性准则一般认为最好不要自定义BACK键。但假如由于某种原因需要在某个activity或者对话框中使用自定义的BACK键, 那么可以使用API level 5 (代号Eclair) 及之后版本声明的一个专门用于捕获BACK键事件的回调方法, 即onBackPressed()。

为了兼容早期的软件开发工具包版本, 可以像清单5-1示例的那样截获KeyEvent.KEYCODE_BACK事件, 然后调用onBackPressed()方法。请注意, 由于显式地调用了Eclair的功能, 该方法只能在Android 2.0及以上版本中编译, 但在所有设备上运行时都会兼容。在视图控件中截获BACK按键事件 (本例中没有涉及) 和清单5-1所示的捕获SEARCH按键事件类似, 都要使用startTracking()方法。

清单5-1 src/com/cookbook/PhysicalKeyPress.java

```
package com.cookbook.physkey;
import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.widget.Toast;
public class PhysicalKeyPress extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        switch (keyCode) {
            case KeyEvent.KEYCODE_CAMERA:
                Toast.makeText(this, "Pressed Camera Button",
                    Toast.LENGTH_LONG).show();
                return true;
            case KeyEvent.KEYCODE_DPAD_LEFT:
                Toast.makeText(this, "Pressed DPAD Left Button",
                    Toast.LENGTH_LONG).show();
                return true;
            case KeyEvent.KEYCODE_VOLUME_UP:
                Toast.makeText(this, "Pressed Volume Up Button",
                    Toast.LENGTH_LONG).show();
                return false;
            case KeyEvent.KEYCODE_SEARCH:
                //example of tracking through to the KeyUp
                if(event.getRepeatCount() == 0)
```

```

        event.startTracking();
        return true;
    case KeyEvent.KEYCODE_BACK:
        // Make new onBackPressed compatible with earlier SDK's
        if (android.os.Build.VERSION.SDK_INT
            < android.os.Build.VERSION_CODES.ECLAIR
            && event.getRepeatCount() == 0) {
            onBackPressed();
        }
        return super.onKeyDown(keyCode, event);
    }

    public void onBackPressed() {
        Toast.makeText(this, "Pressed BACK Key",
            Toast.LENGTH_LONG).show();
    }

    public boolean onKeyUp(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_SEARCH && event.isTracking()
            && !event.isCanceled()) {
            Toast.makeText(this, "Pressed SEARCH Key",
                Toast.LENGTH_LONG).show();
            return true;
        }
        return super.onKeyUp(keyCode, event);
    }
}

```

5.1.2 秘诀 42: 创建菜单

Android系统为开发人员提供了三种菜单，我们将在本秘诀中一一讲解。

- 选项菜单 (Options menu) ——它是activity的主菜单，在按下设备的菜单键时显示。选项菜单包含图标菜单 (Icon menu) 和扩展菜单 (Expanded Menu) 两类。扩展菜单是当按下“更多”菜单项时显示出的竖向工程列表。
- 上下文菜单 (Context Menu) ——它是一个浮动菜单列表，通常在长按某视图时出现。
- 子菜单 (Submenu) ——它是一个浮动菜单列表，通常在某个菜单项被选中时出现。

选项菜单在activity中按下MENU键时被初次创建，系统会调用onCreateOptionsMenu()回调方法，在此方法中通常包含菜单方法，如下代码所示。

```

menu.add(GROUP_DEFAULT, MENU_ADD, 0, "Add")
    .setIcon(R.drawable.icon);

```

add()方法的第1个参数是为菜单组工程设定标签。同一组的菜单项可以一起操作。第2个参数是一个整数值，用于标示菜单项的ID。通过该参数，回调方法可以确定哪个菜单项被选中。第3个参数说明菜单项出现的先后顺序。如果不设置该参数，则默认按照向Menu对象添加菜单项时

的顺序来排序。最后一个参数是该菜单项显示的文本。它可以是String对象，也可以是类似R.string.myLabel的字符串资源。选项菜单是唯一一种支持在菜单项中添加图标的菜单类型，可以通过setIcon()方法为菜单项添加图标。

onCreateOptionsMenu()方法仅被调用一次，在该activity的其余部分无需重新构建此菜单。如果要在运行时改变某个菜单选项的内容，可以调用onOptionsItemSelected()方法。

在选项菜单中的某个菜单项被点击时，系统会调用onOptionsItemSelected()方法。传递的参数为所选菜单项的ID，可以使用switch语句来判定哪个选项被选中。

在本秘诀中使用的菜单选项为：添加便条、删除便条和发送便条。在示例中这些功能被简化模拟为递增计数器（itemNum）、递减计数器和在屏幕用Toast弹出提示显示当前计数器的数值。为了演示程序在运行时动态更新菜单选项的情况，程序规定只有在便条已添加的情况下delete选项才可用。我们可以将delete选项组合成另外的一个菜单组，当itemNum为0时，该菜单组就会隐藏起来。该activity的代码如清单5-2所示。

清单5-2 src/com/cookbook/building_menus/BuildingMenus.java

```
package com.cookbook.building_menus;

import android.app.Activity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.Menu;
import android.view.MenuItem;
import android.view.SubMenu;
import android.view.View;
import android.view.ContextMenu.ContextMenuInfo;
import android.widget.TextView;
import android.widget.Toast;

public class BuildingMenus extends Activity {
    private final int MENU_ADD=1, MENU_SEND=2, MENU_DEL=3;
    private final int GROUP_DEFAULT=0, GROUP_DEL=1;
    private final int ID_DEFAULT=0;
    private final int ID_TEXT1=1, ID_TEXT2=2, ID_TEXT3=3;
    private String[] choices = {"Press Me", "Try Again", "Change Me"};

    private static int itemNum=0;
    private static TextView bv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bv = (TextView) findViewById(R.id.focus_text);

        registerForContextMenu((View) findViewById(R.id.focus_text));
```

```
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(GROUP_DEFAULT, MENU_ADD, 0, "Add")
        .setIcon(R.drawable.icon); //example of adding icon
    menu.add(GROUP_DEFAULT, MENU_SEND, 0, "Send");
    menu.add(GROUP_DEL, MENU_DEL, 0, "Delete");

    return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    if(itemNum>0) {
        menu.setGroupVisible(GROUP_DEL, true);
    } else {
        menu.setGroupVisible(GROUP_DEL, false);
    }
    return super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case MENU_ADD:
            create_note();
            return true;
        case MENU_SEND:
            send_note();
            return true;
        case MENU_DEL:
            delete_note();
            return true;
    }
    return super.onOptionsItemSelected(item);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    if(v.getId() == R.id.focus_text) {
        SubMenu textMenu = menu.addSubMenu("Change Text");
        textMenu.add(0, ID_TEXT1, 0, choices[0]);
        textMenu.add(0, ID_TEXT2, 0, choices[1]);
        textMenu.add(0, ID_TEXT3, 0, choices[2]);
        menu.add(0, ID_DEFAULT, 0, "Original Text");
    }
}
```

```

    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
            case ID_DEFAULT:
                bv.setText(R.string.hello);
                return true;
            case ID_TEXT1:
            case ID_TEXT2:
            case ID_TEXT3:
                bv.setText(choices[item.getItemId()-1]);
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

    void create_note() { // mock code to create note
        itemNum++;
    }

    void send_note() { // mock code to send note
        Toast.makeText(this, "Item: "+itemNum,
            Toast.LENGTH_SHORT).show();
    }

    void delete_note() { // mock code to delete note
        itemNum--;
    }
}

```

清单5-2中的activity也是一个带有上下文菜单和子菜单的示例。在清单5-3中，我们在页面布局上添加一个TextView控件focus_text，并在activity的onCreate()方法中使用registerForContextMenu()方法将focus_text注册为一个上下文菜单。

当长按该视图控件时，系统就会调用onCreateContextMenu()回调方法创建上下文菜单。此处的子菜单是通过Menu实例的addSubMenu()方法来实现的。声明主菜单选项时，其子菜单选项也都会一并声明，不论选中上述两种菜单中的哪个菜单项，系统都会调用onContextItemSelected()方法来确认所选菜单项。在此，本秘诀演示了如何根据菜单选项变换所显示的文本。

清单5-3 res/layout/main.xml

```

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

```

```

        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:text="@string/hello"
    />
</LinearLayout>

```

图5-1和图5-2显示了不同情况下的菜单外观差异。

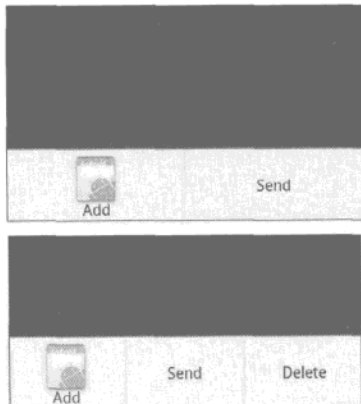


图5-1 选项菜单（上图）和在运行时添加选项（下图）

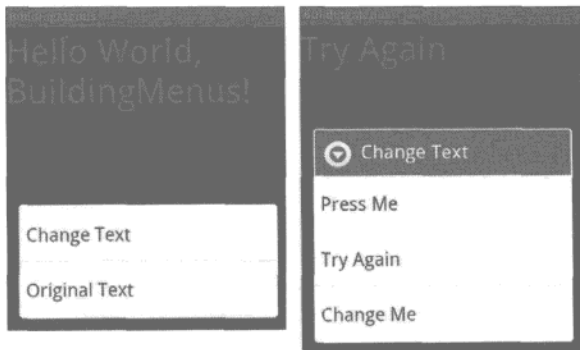


图5-2 长按文本时弹出的上下文菜单（左图）以及选择Change Text选项时弹出的三个备选文本子菜单选项（右图）

5.1.3 秘诀 43：在 XML 文件中定义菜单

当然我们也可以在XML文件中构建菜单，然后再配以前面秘诀中提到的相应回调方法。这种方式在构建大型菜单时非常有用。这样仍旧可以利用Java代码动态处理菜单选项。

菜单文件通常保存在res/menu/资源目录下。譬如，要实现第4章生成的上下文菜单，可以创建一个XML文件定义嵌套菜单，具体代码如清单5-4所示。

清单5-4 res/menu/context_menu.xml

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/submenu" android:title="Change Text">
        <menu xmlns:android="http://schemas.android.com/apk/res/android">
            <item android:id="@+id/text1" android:title="Press Me" />
            <item android:id="@+id/text2" android:title="Try Again" />
            <item android:id="@+id/text3" android:title="Change Me" />
        </menu>
    </item>
    <item android:id="@+id/orig" android:title="Original Text" />
</menu>

```


然后在创建菜单时使用该XML文件，并在菜单项选择方法中引用菜单项的ID。清单5-2中使用的这两个方法在清单5-5所示的代码中被替换。

清单5-5 修改了主activity中使用的方法

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch(item.getItemId()) {
        case R.id.orig:
            bv.setText(R.string.hello);
            return true;
        case R.id.text1:
            bv.setText(choices[0]);
            return true;
        case R.id.text2:
            bv.setText(choices[1]);
            return true;
        case R.id.text3:
            bv.setText(choices[2]);
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

5.1.4 秘诀 44：使用搜索键

如果当前应用程序的某个activity声明为支持搜索功能，那么通过SEARCH键就可以唤醒它。但为了兼容没有SEARCH键的Android设备调用支持搜索功能的activity，我们还要冗余地定义菜单或者其他唤醒方式。菜单方式只需要调用onSearchRequested()方法即可。

正如前面第2章所提到的，支持搜索功能的activity最好将启动模式（launch mode）定义为singleTop。该模式可以支持多个搜索任务同时发生，而不会由于创建多个activity实例导致堵塞堆栈。清单文件中需要包含下面几行代码：

```
<activity android:name=".SearchDialogExample"
    android:launchMode="singleTop" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    <meta-data android:name="android.app.searchable"
        android:resource="@xml/my_search"/>
</activity>
```

引用搜索的XML文件见清单5-6。

清单5-6 res/xml/my_search.xml

```
<?xml version="1.0" encoding="utf-8"?>
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name" android:hint="Search MyExample Here" >
</searchable>
```

本秘诀具有一个搜索界面。当程序启动时，就会显示一个非常简单的主activity页面，如清单5-7所示，页面布局使用默认的main.xml文件。

清单5-7 src/com/cookbook/search_diag/MainActivity.java

```
package com.cookbook.search_diag;

import android.app.Activity;
import android.os.Bundle;
public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

如果按下SEARCH键，就会激活负责搜索的activity。接着，onCreate()方法会核查intent是否为ACTION_SEARCH，如果是的话就继续运行。清单5-8是主activity的代码，仅使用Toast弹出提示框在屏幕上显示搜索内容。

清单5-8 src/com/cookbook/search_diag/SearchDialogExample.java

```
package com.cookbook.search_diag;

import android.app.Activity;
import android.app.SearchManager;
import android.content.Intent;
import android.os.Bundle;
import android.widget.Toast;

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

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUERY);
```

```

        Toast.makeText(this, "The QUERY: " + query,
                        Toast.LENGTH_LONG).show();
    }
}
}

```

5.1.5 秘诀 45：响应触摸事件

不论我们是通过触摸方式与屏幕交互，还是使用轨迹球来交互，任何和屏幕相关的交互都是与屏幕相应位置上的视图控件的交互。如第4章所述，由于界面布局是一个具有层级结构的视图树，系统从该层级结构的顶端开始逐级往下传递事件，直到某个视图组件处理该事件。如果在处理以后还有某些事件没有被消耗，则会继续在层级树中传递。

清单5-9向我们演示一个名为ex_button的按钮如何通过事件监听器处理单击事件和长按事件。当事件发生时，程序会调用相应的回调方法，通过Toast将事件所触发的方法显示在屏幕上。

清单5-9 src/com/cookbook/touch_examples/TouchExamples.java

```

package com.cookbook.touch_examples;

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

public class TouchExamples extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button ex = (Button) findViewById(R.id.ex_button);

        ex.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Toast.makeText(TouchExamples.this, "Click",
                               Toast.LENGTH_SHORT).show();
            }
        });
        ex.setOnLongClickListener(new OnLongClickListener() {
            public boolean onLongClick(View v) {
                Toast.makeText(TouchExamples.this, "LONG Click",
                               Toast.LENGTH_SHORT).show();
                return true;
            }
        });
    }
}

```

```

    });
}
}

```

清单5-10为按钮的布局文件代码。

清单5-10 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/ex_button"
        android:text="Press Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>

```

在清单5-9如此定义该回调方法是为了使代码紧凑，我们也可将它定义如下，以增加代码的易读性和可重用性：

```

View.OnClickListener myTouchMethod = new View.OnClickListener() {
    public void onClick(View v) {
        //insert relevant action here
    }
};
ex.setOnClickListener(myTouchMethod);

```

还有一种方法是在activity中直接实现OnClickListener接口。这样该方法就属于activity层级的方法，从而避免多余的加载过程：

```

public class TouchExamples extends Activity implements OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button ex = (Button) findViewById(R.id.ex_button);
        ex.setOnClickListener(this);
    }

    public void onClick(View v) {
        if(v.getId() == R.id.directory_button) {
            // insert relevant action here
        }
    }
}

```

上面的代码在activity级别实现了onClick()接口，可以帮助我们理解父控件如何处理多种子控件的触摸事件。

5.1.6 秘诀 46：监听滑动手势

正如本章开头提到的，每个视图控件都有与之关联的`onTouchEvent()`方法。在本秘诀中它被手势探测器重载，以设置一个手势监听器。`OnGestureListener`接口可响应的手势包括：

- `onDown()`——按下时立刻触发；
- `onFling()`——按下并滑动（fling）一段距离后，匹配事件时触发；
- `onLongPress()`——长按时触发；
- `onScroll()`——滚动时触发；
- `onShowPress()`——按下且无滑动或释放时触发；
- `onSingleTapUp()`——按下之后放开时触发。

如果仅需要使用少数手势，则可以通过继承`SimpleOnGestureListener`类来代替。如果前面提到的方法没有显式地实现则会返回`false`。

滑动手势包含两个动作事件：按下（第一个`MotionEvent`）和释放（第二个`MotionEvent`）。每个动作事件都通过屏幕上的 (x,y) 坐标位置指定，其中 x 表示横轴坐标， y 表示纵轴坐标， (x,y) 为事件具有的移动速率。

清单5-11展示了一个实现了`onFling()`方法接口的activity。当移动的幅度足够大时（本例定义为60像素），完成事件操作，并在屏幕上追加显示该事件的说明。

清单5-11 `src/com/cookbook/fling_ex/FlingExample.java`

```
package com.cookbook.fling_ex;

import android.app.Activity;
import android.os.Bundle;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.GestureDetector.SimpleOnGestureListener;
import android.widget.TextView;

public class FlingExample extends Activity {
    private static final int LARGE_MOVE = 60;
    private GestureDetector gestureDetector;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.text_result);

        gestureDetector = new GestureDetector(this,
            new SimpleOnGestureListener() {
                @Override
                public boolean onFling(MotionEvent e1, MotionEvent e2,
                    float velocityX, float velocityY) {
```



```

        if (e1.getY() - e2.getY() > LARGE_MOVE) {
            tv.append("\nFling Up with velocity " + velocityY);
            return true;

        } else if (e2.getY() - e1.getY() > LARGE_MOVE) {
            tv.append("\nFling Down with velocity " + velocityY);
            return true;

        } else if (e1.getX() - e2.getX() > LARGE_MOVE) {
            tv.append("\nFling Left with velocity " + velocityX);
            return true;

        } else if (e2.getX() - e1.getX() > LARGE_MOVE) {
            tv.append("\nFling Right with velocity " + velocityX);
            return true;
        }

        return false;
    } });
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    return gestureDetector.onTouchEvent(event);
}
}

```

前面activity中用来显示事件描述说明的TextView控件定义在清单5-12中的main.xml布局文件中。

清单5-12 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/text_result"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:textSize="16sp"
        android:text="Fling right, left, up, or down\n" />
</LinearLayout>

```

5.1.7 秘诀 47：使用多点触控

多点触控是指在同一时间屏幕上有多点（如手指）触摸的事件。多点触控通过OnTouchListener触摸监听器识别，该监听器能够捕获下列不同类型的动作事件：

- ❑ ACTION_DOWN——（手指）按下第一个点开始按压手势；
- ❑ ACTION_POINTER_DOWN——（手指）按下完成第二个点；
- ❑ ACTION_MOVE——在按压手势发生的时候变换了按压的位置；
- ❑ ACTION_POINTER_UP——释放第二个点；
- ❑ ACTION_UP——释放第一个点，完成此按压手势。

该秘诀将在屏幕上显示一幅图像，通过多点触控事件将图片放大或缩小。同时也支持单点触控事件，在屏幕上随意拖动图片。这个过程的activity代码见清单5-13。首先，该activity实现了OnTouchListener接口，并在onCreate()方法中设置了该接口。当发生触摸事件时，onTouch()方法就会判断动作事件类型，进行如下操作。

- ❑ 如果按下第一个触点，则声明touchState为拖动动作，并保存触点位置和Matrix。
- ❑ 如果在第一个触点还没有放开时又按下第二个触点，将会计算两个触点位置之间的距离。当两者距离超过一定值（此处设定为50像素），则将touchState声明为缩放动作，并保存触点之间的距离位置、中间点位置和Matrix矩阵。
- ❑ 移动发生时，单点触摸事件会移动图片，而多点触控事件则缩放图片。
- ❑ 松开触点时，将touchState声明为NONE，即没有动作。

清单5-13 src/com/cookbook/multitouch/MultiTouch.java

```
package com.cookbook.multitouch;

import android.app.Activity;
import android.graphics.Matrix;
import android.os.Bundle;
import android.util.FloatMath;

import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;

public class MultiTouch extends Activity implements OnTouchListener {
    // Matrix instances to move and zoom image
    Matrix matrix = new Matrix();
    Matrix eventMatrix = new Matrix();

    // possible touch states
    final static int NONE = 0;
    final static int DRAG = 1;
    final static int ZOOM = 2;
    int touchState = NONE;

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

```
        ImageView view = (ImageView) findViewById(R.id.imageView);
        view.setOnTouchListener(this);
    }

    final static float MIN_DIST = 50;
    static float eventDistance = 0;
    static float centerX = 0, centerY = 0;
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        ImageView view = (ImageView) v;

        switch (event.getAction() & MotionEvent.ACTION_MASK) {
            case MotionEvent.ACTION_DOWN:
                //primary touch event starts: remember touch down location
                touchState = DRAG;
                centerX = event.getX(0);
                centerY = event.getY(0);
                eventMatrix.set(matrix);
                break;

            case MotionEvent.ACTION_POINTER_DOWN:
                //secondary touch event starts: remember distance and center
                eventDistance = calcDistance(event);
                calcMidpoint(centerX, centerY, event);
                if (eventDistance > MIN_DIST) {
                    eventMatrix.set(matrix);

                    touchState = ZOOM;
                }
                break;

            case MotionEvent.ACTION_MOVE:
                if (touchState == DRAG) {
                    //single finger drag, translate accordingly
                    matrix.set(eventMatrix);
                    matrix.setTranslate(event.getX(0) - centerX,
                                       event.getY(0) - centerY);

                } else if (touchState == ZOOM) {
                    //multi-finger zoom, scale accordingly around center
                    float dist = calcDistance(event);

                    if (dist > MIN_DIST) {
                        matrix.set(eventMatrix);
                        float scale = dist / eventDistance;

                        matrix.postScale(scale, scale, centerX, centerY);
                    }
                }
            }
        }
    }
}
```



```

        // Perform the transformation
        view.setImageMatrix(matrix);
        break;

    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
        touchState = NONE;
        break;
    }

    return true;
}

private float calcDistance(MotionEvent event) {
    float x = event.getX(0) - event.getX(1);
    float y = event.getY(0) - event.getY(1);
    return FloatMath.sqrt(x * x + y * y);
}

private void calcMidpoint(float centerX, float centerY,
    MotionEvent event) {
    centerX = (event.getX(0) + event.getX(1))/2;
    centerY = (event.getY(0) + event.getY(1))/2;
}
}

```

清单5-14布局文件则声明了要缩放的图片。本秘诀中使用的图片是由Eclipse自动创建的icon.png，当然也可以替换为其他任何图片。

清单5-14 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <ImageView android:id="@+id/imageView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:src="@drawable/icon"
        android:scaleType="matrix" >
    </ImageView>
</FrameLayout>

```

5.2 高级用户界面库

某些用户界面功能需要使用复杂的算法进行计算。为嵌入式系统优化算法通常是既耗时又充满挑战的工作，因此开发者最好还是直接使用现成的UI库比较省力。下面的两个秘诀将演示一些

可参考的范例，权作抛砖引玉。

5.2.1 秘诀 48：使用手势

手势是在触摸屏上手绘的形状。android.gesture包提供了辨识和处理这些手势的简单方法。首先，每个SDK都有一个示例程序，可以用来构建手势集，在platforms/Android-2.0/samples/GestureBuilder/可以找到该程序的源代码。Gesture Builder工程可以导入并运行在Android设备上。该程序生成一个文件/sdcard/gestures，这个文件可以从设备上复制出来，作为本例的原始资源使用。

作为示例，该程序可以生成如图5-3所示的手写数字文件。由于多个手势可以具有相同的名称，因此为同一种手势提供多个例子有助于提高识别度。

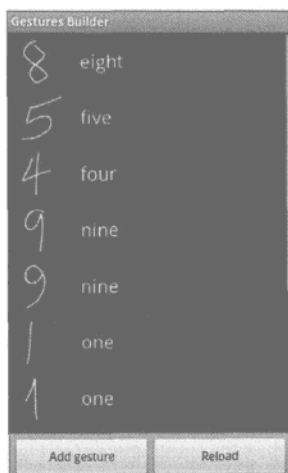


图5-3 随Android SDK发布的Gesture Builder示例程序，可以用来创建手势库

在该文件中，当从0到9的所有相关变体都创建成功后，可以将其复制到res/raw/numbers资源目录下。布局文件如清单5-15所示，主activity的代码见清单5-16。在该activity中，使用原始资源初始化GestureLibrary对象。

本秘诀将在屏幕顶层添加一个GestureOverlayView控件，并实现OnGesturePerformedListener接口。每当手势绘制完时，就会将其传递给onGesturePerformed()方法判断，将该手势与手势库中所有手势进行比较，然后按照相似度由高到低返回一个匹配结果列表。每个预测结果都包括名字和匹配度分值两项内容，其中名字为手势库中定义的手势名，而匹配度分值则反映了与预设输入的手势的匹配程度。通常来讲只要第一项的分值大于1，就算匹配成功。

清单5-15 res/layout/main.xml

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

```

        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal" android:textSize="20sp"
        android:text="Draw a number"
        android:layout_margin="10dip"/>
    <android.gesture.GestureOverlayView
        android:id="@+id/gestures"
        android:layout_width="fill_parent"
        android:layout_height="0dip"
        android:layout_weight="1.0" />

    <TextView android:id="@+id/prediction"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal" android:textSize="20sp"
        android:text=""
        android:layout_margin="10dip"/>
    </LinearLayout>

```

为便于说明，本秘诀将所有的预测结果都转换为字符串，并将它们显示在屏幕上。图5-4为其中一个输出示例。该秘诀表明，即便看上去没有写完的数字，数字笔画的部分内容也可以与库中的数字良好匹配。

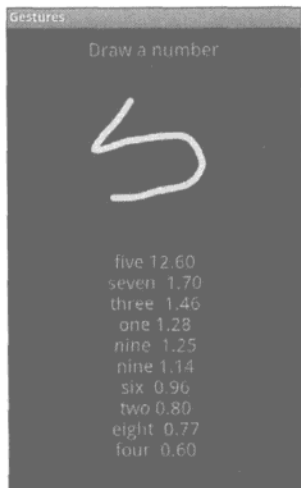


图5-4 显示匹配分值的手势辨识示例

清单5-16 src/com/cookbook/gestures/Gestures.java

```

package com.cookbook.gestures;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;

import android.app.Activity;
import android.gesture.Gesture;
import android.gesture.GestureLibraries;
import android.gesture.GestureLibrary;
import android.gesture.GestureOverlayView;
import android.gesture.Prediction;
import android.gesture.GestureOverlayView.OnGesturePerformedListener;
import android.os.Bundle;
import android.widget.TextView;

public class Gestures extends Activity
    implements OnGesturePerformedListener {

    private GestureLibrary mLibrary;
    private TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.prediction);

        mLibrary = GestureLibraries.fromRawResource(this, R.raw.numbers);
        if (!mLibrary.load()) finish();

        GestureOverlayView gestures =
            (GestureOverlayView) findViewById(R.id.gestures);
        gestures.addOnGesturePerformedListener(this);
    }

    public void onGesturePerformed(GestureOverlayView overlay,
        Gesture gesture) {
        ArrayList<Prediction> predictions = mLibrary.recognize(gesture);
        String predList = "";
        NumberFormat formatter = new DecimalFormat("#0.00");
        for(int i=0; i<predictions.size(); i++) {
            Prediction prediction = predictions.get(i);
            predList = predList + prediction.name + " "
                + formatter.format(prediction.score) + "\n";
        }
        tv.setText(predList);
    }
}

```

5.2.2 秘诀 49：绘制 3D 图像

Android支持针对嵌入式系统的开放图形库(OpenGL ES)。本秘诀基于一个Android API Demo演示程序,将向我们展示如何利用这个图形库创建三维金字塔形状,并让它在屏幕上四处弹跳同时旋转以显示其边缘形状。程序的主activity需要有两个辅助类,一个用于定义形状,如清单5-17所示;另外一个用来渲染金字塔,如清单5-18所示。

清单5-17 src/com/cookbook/open_gl/Pyramid.java

```
package com.cookbook.open_gl;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.opengles.GL10;

class Pyramid {
    public Pyramid() {
        int one = 0x10000;
        /* square base and point top to make a pyramid */
        int vertices[] = {
            -one, -one, -one,
            -one, one, -one,
            one, one, -one,
            one, -one, -one,
            0, 0, one
        };

        /* purple fading to white at the top */
        int colors[] = {
            one, 0, one, one,
            one, 0, one, one,
            one, 0, one, one,
            one, 0, one, one,
            one, one, one, one
        };

        /* triangles of the vertices above to build the shape */
        byte indices[] = {
            0, 1, 2, 0, 2, 3, //square base
            0, 3, 4, // side 1
            0, 4, 1, // side 2
            1, 4, 2, // side 3
            2, 4, 3, // side 4
        };

        // Buffers to be passed to gl*Pointer() functions
        ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length*4);
```

```

        vbb.order(ByteOrder.nativeOrder());
        mVertexBuffer = vbb.asIntBuffer();
        mVertexBuffer.put(vertices);
        mVertexBuffer.position(0);

        ByteBuffer cbb = ByteBuffer.allocateDirect(colors.length*4);
        cbb.order(ByteOrder.nativeOrder());
        mColorBuffer = cbb.asIntBuffer();
        mColorBuffer.put(colors);
        mColorBuffer.position(0);

        mIndexBuffer = ByteBuffer.allocateDirect(indices.length);
        mIndexBuffer.put(indices);
        mIndexBuffer.position(0);
    }

    public void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CW);
        gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);
        gl.glColorPointer(4, GL10.GL_FIXED, 0, mColorBuffer);
        gl.glDrawElements(GL10.GL_TRIANGLES, 18, GL10.GL_UNSIGNED_BYTE,
                           mIndexBuffer);
    }

    private IntBuffer    mVertexBuffer;
    private IntBuffer    mColorBuffer;
    private ByteBuffer   mIndexBuffer;
}

```

注意，该金字塔共有5个点：方形底座上有4个，尖顶上有1个。这5个点的位置排列不是简单的随机排列，而是有序的，它们可以围绕穿越图形的一条线旋转。图形的中心位于原点(0, 0, 0)。

5个RGBA格式的颜色对象color分别与5个顶点对应，基座的4个顶点定义为紫色，顶部顶点则定义为白色。图形库会以渐变方式填充该图形。这样的多层次色泽和阴影有助于实现3D外观效果。

程序中的主方法draw()用于绘制三角形元素。底座四边形由两个三角形组成，每个向上的面为一个三角形，因此整个金字塔共由6个三角形、18个标记点组成。图5-5从两个不同的视角显示了金字塔弹跳时的图形效果。

接着，创建一个新类实现GLSurfaceView.Renderer接口，此后就可以使用OpenGL ES库渲染该金字塔图形，具体如清单5-18所示。同时要实现GLSurfaceView.Renderer接口的3个方法：

- onSurfaceCreated()——一次性初始化OpenGL框架；
- onSurfaceChanged()——在(动画)开始或者调整视角时，设置投影；
- onDrawFrame()——逐帧绘制图形图像。

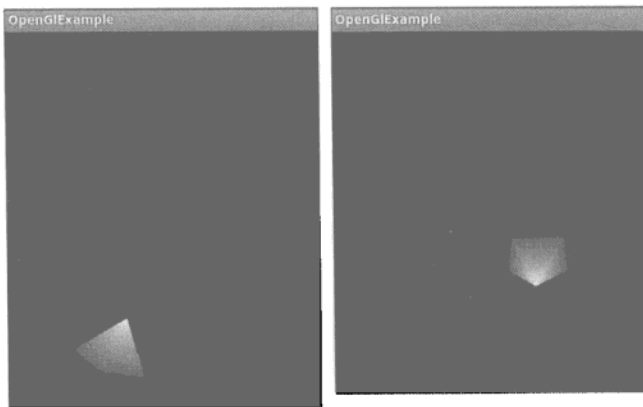


图5-5 用OpenGL ES创建的金字塔在翻转和弹跳

清单5-18 src/com/cookbook/open_gl/PyramidRenderer.java

```
package com.cookbook.open_gl;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.opengl.GLSurfaceView;

/**
 * Render a tumbling Pyramid
 */

class PyramidRenderer implements GLSurfaceView.Renderer {
    public PyramidRenderer(boolean useTranslucentBackground) {
        mTranslucentBackground = useTranslucentBackground;
        mPyramid = new Pyramid();
    }

    public void onDrawFrame(GL10 gl) {
        /* clear the screen */
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

        /* draw a pyramid rotating */
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(mCenter[0], mCenter[1], mCenter[2]);
        gl.glRotatef(mAngle, 0, 1, 0);
        gl.glRotatef(mAngle*0.25f, 1, 0, 0);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        mPyramid.draw(gl);
    }
}
```

```
mAngle += mAngleDelta;

/* draw it bouncing off the walls */
mCenter[0] += mVel[0];
mCenter[1] += mVel[1];

if(Math.abs(mCenter[0])>4.0f) {
    mVel[0] = -mVel[0];
    mAngleDelta=(float) (5*(0.5-Math.random()));
}
if(Math.abs(mCenter[1])>6.0f) {
    mVel[1] = -mVel[1];
    mAngleDelta=(float) (5*(0.5-Math.random()));
}
}

public void onSurfaceChanged(GL10 gl, int width, int height) {
    gl.glViewport(0, 0, width, height);

    /* Set a new projection when the viewport is resized */
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 20);
}

public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    gl.glDisable(GL10.GL_DITHER);

    /* one-time OpenGL initialization */
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
        GL10.GL_FASTEST);

    if (mTranslucentBackground) {
        gl.glClearColor(0,0,0,0);
    } else {
        gl.glClearColor(1,1,1,1);
    }
    gl.glEnable(GL10.GL_CULL_FACE);
    gl.glShadeModel(GL10.GL_SMOOTH);
    gl.glEnable(GL10.GL_DEPTH_TEST);
}

private boolean mTranslucentBackground;
private Pyramid mPyramid;
private float mAngle, mAngleDelta=0;
private float mCenter[]={0,0,-10};
private float mVel[]={0.025f, 0.03535227f, 0f};
}
```


程序通过onDrawFrame()方法捕获金字塔形状的弹跳轨迹。清除屏幕后绘制新图像,然后设置金字塔图形的中心点为mCenter[]。屏幕的中心被定义为原点,因此起始点应为(0,0,-10),使图形在开始时背对着屏幕(刚开始只能看到金字塔图形的底座四边形)。每次更新画面时,图形的旋转角度为mAngleDelta,位移为mVel[]。mVel对象在x轴和y轴方向的数字要有足够的差异,才能更丰富地展现出弹跳的视觉效果。当图形落到屏幕边缘时,就会切换速率方向,使之向屏幕中弹回。

最后主activity必须将其内容视图(content view)设置为OpenGL ES对象,代码如清单5-19所示。金字塔图形的运动会伴随activity的暂停而暂停、恢复而恢复。

清单5-19 src/com/cookbook/open_gl/OpenGLExample.java

```
package com.cookbook.open_gl;

import android.app.Activity;
import android.opengl.GLSurfaceView;
import android.os.Bundle;
/* Wrapper activity demonstrating the use of GLSurfaceView, a view
 * that uses OpenGL drawing into a dedicated surface. */
public class OpenGLExample extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Set our Preview view as the Activity content
        mGLSurfaceView = new GLSurfaceView(this);
        mGLSurfaceView.setRenderer(new PyramidRenderer(true));
        setContentView(mGLSurfaceView);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mGLSurfaceView.onResume();
    }

    @Override
    protected void onPause() {
        super.onPause();
        mGLSurfaceView.onPause();
    }

    private GLSurfaceView mGLSurfaceView;
}
```

Android平台提供了全面而强大的多媒体功能。本章主要介绍图像处理、音频的录制和播放，以及视频的录制和播放等技术。Android系统支持大部分的多媒体解码器读取多媒体文件，但只支持部分多媒体编码器创建多媒体文件。表6-1总结了Android 2.2支持的多媒体框架。特别要注意的是音频无损压缩格式目前尚不支持，在以后的版本将予以修正。

表6-1 Android 2.2支持读写多媒体类型

媒体类型	压缩格式 ^①	Android本地编/译码器 所支持的操作	格式类型
图像	None (raw)	查看	BMP
	Lossless	查看	GIF、PNG
	Lossy	保存/查看	JPEG
音频 (音乐)	None (raw)	录制/播放	PCM
	None (raw)	播放	WAVE
	Lossless	不支持	例如FLAC
	Lossy	播放	MP3、MP4、AAC、HE-AACv1、 HE-AACv2、Ogg Vorbis
	Midi	播放	MID、XMF、RTTTL、RTX、 OTA、IMY
音频 (语音)	Lossy	录制/播放	AMR-NB
	Lossy	播放	AMR-WB
视频	Nearly	播放	H.264
	Lossless		
	Lossy	录制/播放	H.263、MPEG-4 SP

应用程序中不管录制何种类型的媒体，都需要在AndroidManifest.xml文件中设置合适的权限许可（为下面两行中的一行，或两行都要设置）。

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

^① 表6-1中None (raw)为未经压缩的原始资源格式，Lossless为无损压缩，Lossy为有损压缩。——译者注

6.1 图像

正如前面第4章所讲到的,应用程序的本地图像文件通常放置在res/drawable/目录下,随应用程序一起打包发布。它们可以通过R.drawable.my_picture等恰当的资源标识符访问。Android设备文件系统中的图像文件则可以通过InputStream等通用Java类读取。然而,在Android中将图像读取到内容中进行操作的首选方法还是使用内置类BitmapFactory。

BitmapFactory类可以从文件、数据流和字节数组中创建Bitmap对象。下面是两个范例。

```
Bitmap myBitmap1 = BitmapFactory.decodeResource(getResources(),
                                         R.drawable.my_picture);

Bitmap myBitmap2 = BitmapFactory.decodeFile(filePath);
```

在图像读入到内存以后,我们就可以使用getPixel()和setPixel()等位图方法进行操作了。然而很多图片的原尺寸对于嵌入式设备都太大,不合适直接读入到内存。因此作为替代方法,我们可以考虑对图像二次采样(以缩小尺寸):

```
Bitmap bm = Bitmap.createScaledBitmap(myBitmap2, 480, 320, false);
```

这样做可以避免运行时的内存溢出错误。

秘诀 50: 图像的导入和处理

本秘诀演示了如何将一幅图像切成四块,打乱次序后显示在屏幕上方。同时也演示了如何创建可选择的图像列表。

设备拍摄的照片通常放在DCIM/Camera/目录下,本秘诀将使用该目录作为示例图像目录。将图片目录传递给名为ListFiles的activity后,它会列出所有的图像文件并返回用户选择的那幅。

然后将选中的图片加载到内存中供下一步操作。但如果文件太大,为节省内存,我们可以在加载时使用二次采样,只需将onActivityResult中的加粗语句替换为下面的代码即可。

```
BitmapFactory.Options options = new BitmapFactory.Options();
options.inSampleSize = 4;
Bitmap ImageToChange= BitmapFactory.decodeFile(tmp, options);
```

将inSampleSize的值设为4会把图像大小缩减为原图的1/16(每个像素只采样原图像素点的1/4)。该变量值可以根据原图的大小调整设定。

另一个节省内存的方法是在对图像执行任何操作之前先调整位图大小。如本秘诀所示,可以使用createScaledBitmap()方法实现此功能。主activity的代码如清单6-1所示。

清单6-1 src/com/cookbook/image_manip/ImageManipulation.java

```
package com.cookbook.image_manip;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.os.Environment;
```

```
import android.widget.ImageView;

public class ImageManipulation extends Activity {
    static final String CAMERA_PIC_DIR = "/DCIM/Camera/";
    ImageView iv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        iv = (ImageView) findViewById(R.id.my_image);

        String ImageDir = Environment.getExternalStorageDirectory()
            .getAbsolutePath() + CAMERA_PIC_DIR;

        Intent i = new Intent(this, ListFiles.class);
        i.putExtra("directory", ImageDir);
        startActivityForResult(i, 0);
    }

    @Override
    protected void onActivityResult(int requestCode,
        int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if(requestCode == 0 && resultCode==RESULT_OK) {
            String tmp = data.getExtras().getString("clickedFile");
            Bitmap ImageToChange= BitmapFactory.decodeFile(tmp);
            process_image(ImageToChange);
        }
    }

    void process_image(Bitmap image) {
        Bitmap bm = Bitmap.createScaledBitmap(image, 480, 320, false);
        int width = bm.getWidth();
        int height = bm.getHeight();
        int x= width>>1;
        int y= height>>1;
        int[] pixels1 = new int[(width*height)];
        int[] pixels2 = new int[(width*height)];
        int[] pixels3 = new int[(width*height)];
        int[] pixels4 = new int[(width*height)];
        bm.getPixels(pixels1, 0, width, 0, 0, width>>1, height>>1);
        bm.getPixels(pixels2, 0, width, x, 0, width>>1, height>>1);
        bm.getPixels(pixels3, 0, width, 0, y, width>>1, height>>1);
        bm.getPixels(pixels4, 0, width, x, y, width>>1, height>>1);
        if(bm.isMutable()) {
            bm.setPixels(pixels2, 0, width, 0, 0, width>>1, height>>1);
            bm.setPixels(pixels4, 0, width, x, 0, width>>1, height>>1);
            bm.setPixels(pixels1, 0, width, 0, y, width>>1, height>>1);
        }
    }
}
```

```

        bm.setPixels(pixels3, 0, width, x, y, width>>1, height>>1);
    }
    iv.setImageBitmap(bm);
}
}

```

activity对应的主布局文件如清单6-2所示。

清单6-2 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="Scrambled Picture" />
    <ImageView android:id="@+id/my_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>

```

第二个activity负责列出指定目录的所有图像文件，代码见清单6-3。它将根据传递给该activity的目录字符串创建一个File对象。如果该File对象是一个目录，那么就使用compare()方法根据文件的lastModified()标记将文件按逆时顺序进行排序。

清单6-3 src/com/cookbook/image_manip/ListFiles.java

```

package com.cookbook.image_manip;

import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;

public class ListFiles extends ListActivity {
    private List<String> directoryEntries = new ArrayList<String>();

    @Override

```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent i = getIntent();
    File directory = new File(i.getStringExtra("directory"));

    if (directory.isDirectory()){
        File[] files = directory.listFiles();

        //sort in descending date order
        Arrays.sort(files, new Comparator<File>(){
            public int compare(File f1, File f2) {
                return -Long.valueOf(f1.lastModified())
                    .compareTo(f2.lastModified());
            }
        });

        //fill list with files
        this.directoryEntries.clear();
        for (File file : files){
            this.directoryEntries.add(file.getPath());
        }
        ArrayAdapter<String> directoryList
            = new ArrayAdapter<String>(this,
                R.layout.file_row, this.directoryEntries);

        //alphabetize entries
        //directoryList.sort(null);
        this.setListAdapter(directoryList);
    }
}

@Override
protected void onItemClick(ListView l, View v, int pos, long id) {
    File clickedFile = new File(this.directoryEntries.get(pos));
    Intent i = getIntent();
    i.putExtra("clickedFile", clickedFile.toString());
    setResult(RESULT_OK, i);
    finish();
}
}
```

如果想按照字母顺序排列，可以使用`sort()`方法。（该方法也在`ListFiles` activity中，只不过被注释掉了而已）。然后，调用一个独立的布局文件`R.layout.file_row`在屏幕上搭建和显示该列表，代码如清单6-4所示。

`ListFiles` activity对应布局文件见清单6-4。两个activity都必须同时在`AndroidManifest.xml`文件中声明，详见清单6-5。输出的结果示例参见图6-1。

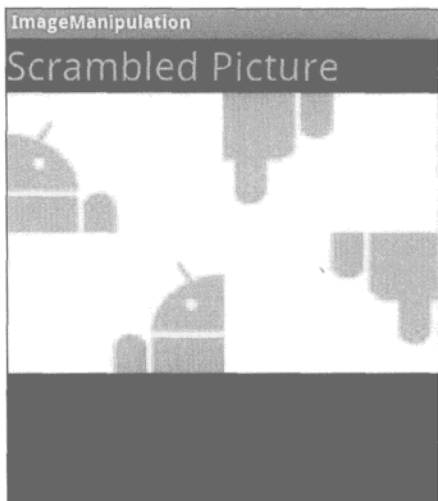


图6-1 打乱的Android图像

清单6-4 res/layout/file_row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:padding="3pt"
/>
```

清单6-5 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.image_manip"
    android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".ImageManipulation"
            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=".ListFiles"
            android:screenOrientation="portrait">
```

```
        android:label="Choose a File">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>
<uses-sdk android:minSdkVersion="5" />
</manifest>
```

6.2 音频

Android提供了两种不同的框架用于录制和播放音频。具体使用哪种需要根据应用程序进行适当的选择。

- MediaPlayer/MediaRecorder——是处理音频的标准方法，但必须是文件或基于流的数据。该类使用时需要创建自己的线程运行。SoundPool类就使用了该框架。
- AudioTrack/AudioRecorder——支持直接访问原始音频文件。用于在内存中处理音频文件，或者开始播放音频的同时写入缓冲区，或者在其他不需要文件和数据流的场合中使用。运行过程中不需要创建新线程。

这些方法将会在本节的多个秘诀中使用。

6.2.1 秘诀 51：选取和播放音频文件

MediaRecorder和MediaPlayer这两个类既可以用来录制和播放音频，也同样可以用来处理视频。本秘诀着重于音频，用法非常简单。若要播放音频，操作步骤如下。

(1) 创建一个MediaPlayer实例：

```
MediaPlayer m_mediaPlayer = new MediaPlayer();
```

(2) 指定媒体源。可以指定为raw资源目录下的某个音频文件：

```
m_mediaPlayer = MediaPlayer.create(this, R.raw.my_music);
```

也可以直接指定文件系统中某个要播放的音频文件（还需要调用prepare方法）：

```
m_mediaPlayer.setDataSource(path);
m_mediaPlayer.prepare();
```

不论使用哪种方法，语句都需要在try-catch语句块中调用，因为指定的音频资源可能不存在。

(3) 开始播放音频：

```
m_mediaPlayer.start();
```

(4) 当播放结束后，停止MediaPlayer并释放该实例，以释放相关资源：

```
m_mediaPlayer.stop();
m_mediaPlayer.release();
```

本秘诀使用了前面的清单6-3和清单6-4中定义的ListFiles activity来创建音频文件选择列表。我们假定音频文件位于Android设备的/sdcard/music/目录下，当然也可以指定为其他路径。

从ListFiles activity返回一个音频文件后，该文件就会被初始化为MediaPlayer的媒体源，然后调用startMP()方法。该方法启动MediaPlayer，同时将按钮文本设置为Pause。与此类似，pauseMP()方法会暂停MediaPlayer，同时将按钮文本设置为Play。这样用户就可以点击按钮暂停或播放音频文件了。

通常MediaPlayer会创建自己的后台进程，这样当主activity运行中止时，MediaPlayer还可以继续播放音乐。这种运行方式对于音乐播放器非常合理，但开发人员通常希望对音频播放行为进行人为控制。为了便于说明，本秘诀覆写了onPause()方法和onResume()方法，从而使音乐的暂停和播放状态与主activity生命周期相一致。详见清单6-6。

清单6-6 src/com/cookbook/audio_ex/AudioExamples.java

```
package com.cookbook.audio_ex;

import android.app.Activity;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.Button;

public class AudioExamples extends Activity {
    static final String MUSIC_DIR = "/music/";
    Button playPauseButton;

    private MediaPlayer m_mediaPlayer;

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

        setContentView(R.layout.main);
        playPauseButton = (Button) findViewById(R.id.play_pause);

        m_mediaPlayer = new MediaPlayer();

        String MusicDir = Environment.getExternalStorageDirectory()
            .getAbsolutePath() + MUSIC_DIR;

        //Show a list of Music files to choose
        Intent i = new Intent(this, ListFiles.class);
        i.putExtra("directory", MusicDir);
        startActivityForResult(i, 0);

        playPauseButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                if(m_mediaPlayer.isPlaying()) {
```

```
        //stop and give option to start again
        pauseMP();
    } else {
        startMP();
    }
}
});
}

@Override
protected void onActivityResult(int requestCode,
                                int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == 0 && resultCode==RESULT_OK) {
        String tmp = data.getExtras().getString("clickedFile");

        try {
            m_mediaPlayer.setDataSource(tmp);
            m_mediaPlayer.prepare();
        } catch (Exception e) {
            e.printStackTrace();
        }
        startMP();
    }
}

void pauseMP() {
    playPauseButton.setText("Play");
    m_mediaPlayer.pause();
}

void startMP() {
    m_mediaPlayer.start();
    playPauseButton.setText("Pause");
}

boolean needToResume = false;
@Override
protected void onPause() {
    if(m_mediaPlayer != null && m_mediaPlayer.isPlaying()) {
        needToResume = true;
        pauseMP();
    }
    super.onPause();
}

@Override
protected void onResume() {
    super.onResume();
    if(needToResume && m_mediaPlayer != null) {
```

```

        startMP();
    }
}
}

```

带有play/pause按钮的布局文件如清单6-7所示。

清单6-7 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/play_pause"
        android:text="Play"
        android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>

```

6.2.2 秘诀 52：录制音频文件

6

使用MediaRecorder录制音频和秘诀51中播放音频的用法相似，只是需要另外作一些声明。（注意也可以使用DEFAULT默认设置，下面选项列表的第一项为默认设置值。）

□ MediaRecorder.AudioSource:

- MIC——内置麦克风
- VOICE_UPLINK——语音通话时向外发送音频
- VOICE_DOWNLINK——语音通话时接收音频
- VOICE_CALL——语音通话时既包括上行发送（uplink）也包括下行传输（downlink）音频
- CAMCORDER——摄像状态时的麦克风（如果可用）
- VOICE_RECOGNITION——语音识别状态时的麦克风（如果可用）

□ MediaRecorder.OutputFormat:

- THREE_GPP——3GPP多媒体文件格式
- MPEG_4——MPEG4多媒体文件格式
- AMR_NB——自适应多速率窄带文件格式

□ MediaRecorder.AudioEncoder:

- AMR_NB——自适应多速率窄带编码器

录制音频的步骤如下。

(1) 创建一个MediaRecorder实例:

```
MediaRecorder m_Recorder = new MediaRecorder();
```

(2) 设置媒体源, 如麦克风:

```
m_Recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
```

(3) 设置输出文件的格式和编码方式, 例如:

```
m_Recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
```

```
m_Recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
```

(4) 设置文件保存的路径:

```
m_Recorder.setOutputFile(path);
```

(5) 准备并开始录制:

```
m_Recorder.prepare();
```

```
m_Recorder.start();
```

关于音频录制的步骤可以参考前一个秘诀。

6.2.3 秘诀 53: 处理原始音频

MediaRecorder/MediaPlayer框架适用于绝大多数音频, 但是要想使用从麦克风直接输入的原始音频, 在不保存文件的前提下直接处理, 或者直接播放原始音频, 就只能使用AudioRecord/AudioTrack框架了。首先需要在AndroidManifest.xml清单文件中设置权限许可:

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

录制音频的具体步骤如下所示。

(1) 创建一个AudioRecord实例, 向构造函数中声明如下内容。

- ❑ 音频源——使用前面秘诀提到的MediaRecorder.AudioSource选项, 如MediaRecorder.AudioSource.MIC。
- ❑ 采样频率, 以Hz为单位——使用44 100 Hz的CD品质采样率, 或者降低采样频率到22 050或者11 025都可以。(该采样频率对于声音来说已经足够了。)
- ❑ 声道设置——使用AudioFormat.CHANNEL_IN_STEREO录制立体声, 或者CHANNEL_IN_MONO录制单声道声音。
- ❑ 音频编码——8位编码使用AudioFormat.ENCODING_PCM_8BIT, 16位编码使用AudioFormat.ENCODING_PCM_16BIT。
- ❑ 以字节为单位的缓存大小——静态模式为整个分配的内存大小, 流媒体模式下为所分配块(chunk)的大小。但字节至少为getMinBufferSize()方法值。

(2) 用AudioRecord实例开始录制。

(3) 使用下列方法中的一个读入音频数据到内存中的audioData[]数组:

```
read(short[] audioData, int offsetInShorts, int sizeInShorts)
```

```
read(byte[] audioData, int offsetInBytes, int sizeInBytes)
```

```
read(ByteBuffer audioData, int sizeInBytes)
```

(4) 停止录制。

例如, 下面的代码就能够通过内置麦克风录制语言到内存缓冲区myRecordedAudio中, 内存缓冲区被声明为short[]类型(例如, 每个样本16位)。要注意的是, 每秒钟11 025个样本的采样

频率，对于大小只有10 000个样本的缓冲区只能录制不到一秒钟的音频。

```
short[] myRecordedAudio = new short[10000];
AudioRecord audioRecord = new AudioRecord(
    MediaRecorder.AudioSource.MIC, 11025,
    AudioFormat.CHANNEL_IN_MONO,
    AudioFormat.ENCODING_PCM_16BIT, 10000);
audioRecord.startRecording();
audioRecord.read(myRecordedAudio, 0, 10000);
audioRecord.stop();
```

播放音频的步骤如下。

(1) 创建一个AudioTrack实例，并在构造函数中定义如下内容。

- 流类型——使用AudioManager.STREAM_MUSIC从麦克风捕获或从扬声器回放。其他选项包括STREAM_VOICE_CALL、STREAM_SYSTEM、STREAM_RING和STREAM_ALARM。
- 以Hz为单位的采样频率——和录制时该项的意义相同。
- 通道配置——使用AudioFormat.CHANNEL_OUT_STEREO播放立体声，还有其他的选项，如CHANNEL_OUT_MONO和CHANNEL_OUT_5POINT1（用于环绕声）。
- 音频编码——和录制时该项的意义相同。
- 以字节为单位的缓冲区大小——每次播放时的数据块大小。
- 缓冲模式——对于完全能够转入内存的短声音文件，使用AudioTrack.MODE_STATIC模式，避免传输开销。否则使用AudioTrack.MODE_STREAM模式将数据写入硬件的缓冲块中。

(2) 通过AudioTrack实例开始播放。

(3) 使用下面方法中的一种将内存中的audioData[]写入硬件：

```
write(short[] audioData, int offsetInShorts, int sizeInShorts)
write(byte[] audioData, int offsetInBytes, int sizeInBytes)
```

(4) 停止播放（可选）。

例如，如下代码可用于播放前面录制音频范例中的语音数据：

```
AudioTrack audioTrack = new AudioTrack(
    AudioManager.STREAM_MUSIC, 11025,
    AudioFormat.CHANNEL_OUT_MONO,
    AudioFormat.ENCODING_PCM_16BIT, 4096,
    AudioTrack.MODE_STREAM);

audioTrack.play();
audioTrack.write(myRecordedAudio, 0, 10000);
audioTrack.stop();
```

本秘诀使用两种方法录制和播放音频。布局文件中定义了两个按钮以显示在屏幕上，一个用来录制音频，一个用来播放已录制的音频，如清单6-8所示。

清单6-8 res/layout/main.xml

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

```
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<TextView android:id="@+id/status"
    android:text="Ready" android:textSize="20sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<Button android:id="@+id/record"
    android:text="Record for 5 seconds"
    android:textSize="20sp" android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<Button android:id="@+id/play"
    android:text="Play" android:textSize="20sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

清单6-9中的主activity为录制和播放内存中音频缓冲的两个按钮分别设置了OnClick-Listener监听器。由于AudioTrack和AudioRecord这两个类都不应当在UI线程中运行，因此onClick()回调方法为之创建了对应的后台线程。为便于说明，本秘诀使用两种不同的线程创建方法：record_thread()启动了一个本地线程通过Handler更新UI，而播放线程则使用了主activity的run()方法。

程序将缓冲保存在内存中。为便于说明，我们设定录制时间为5秒。

清单6-9 src/com/cookbook/audio_ex/AudioExamplesRaw.java

```
package com.cookbook.audio_ex;

import android.app.Activity;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class AudioExamplesRaw extends Activity implements Runnable {
    private TextView statusText;

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

        statusText = (TextView) findViewById(R.id.status);
```

```

        Button actionButton = (Button) findViewById(R.id.record);
        actionButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                record_thread();
            }
        });

        Button replayButton = (Button) findViewById(R.id.play);
        replayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                Thread thread = new Thread(AudioExamplesRaw.this);
                thread.start();
            }
        });
    }

    String text_string;
    final Handler mHandler = new Handler();
    // Create runnable for posting
    final Runnable mUpdateResults = new Runnable() {
        public void run() {
            updateResultsInUi(text_string);
        }
    };

    private void updateResultsInUi(String update_txt) {
        statusText.setText(update_txt);
    }

    private void record_thread() {
        Thread thread = new Thread(new Runnable() {
            public void run() {
                text_string = "Starting";
                mHandler.post(mUpdateResults);

                record();

                text_string = "Done";
                mHandler.post(mUpdateResults);
            }
        });
        thread.start();
    }

    private int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
    private int frequency = 11025; //Hz
    private int bufferSize = 50*AudioTrack.getMinBufferSize(frequency,
        AudioFormat.CHANNEL_OUT_MONO, audioEncoding);
    // Create new AudioRecord object to record the audio.
    public AudioRecord audioRecord = new AudioRecord(

```



```
MediaRecorder.AudioSource.MIC,
frequency, AudioFormat.CHANNEL_IN_MONO,
audioEncoding, bufferSize);
// Create new AudioTrack object w/same parameters as AudioRecord obj
public AudioTrack audioTrack = new AudioTrack(
    AudioManager.STREAM_MUSIC, frequency,
    AudioFormat.CHANNEL_OUT_MONO,
    audioEncoding, 4096,
    AudioTrack.MODE_STREAM);
short[] buffer = new short[bufferSize];

public void record() {
    try {
        audioRecord.startRecording();
        audioRecord.read(buffer, 0, bufferSize);
        audioRecord.stop();
    } catch (Throwable t) {
        Log.e("AudioExamplesRaw", "Recording Failed");
    }
}

public void run() { //play audio using runnable Activity
    audioTrack.play();
    //this alone works: audioTrack.write(buffer, 0, bufferSize);
    //but for illustration showing another way to play using a loop:
    int i=0;
    while(i<bufferSize) {
        audioTrack.write(buffer, i++, 1);
    }
    return;
}

@Override
protected void onPause() {
    if(audioTrack!=null) {
        if(audioTrack.getPlayState()==AudioTrack.PLAYSTATE_PLAYING) {
            audioTrack.pause();
        }
    }
    super.onPause();
}
}
```

6.2.4 秘诀 54：有效使用音频资源

若要既像压缩音频文件那样使用较小的内存，又像原始音频文件那样具有回放的低时延性，我们可以使用SoundPool类。它在使用MediaPlayer服务进行音频解码的同时也提供了重复播放音频缓冲区的方法，并支持音频加速和减速播放。

该类在使用上和其他关于音频播放的秘诀类似：初始化、加载资源、播放音频，最后释放资源。不过要注意的是，SoundPool会启动一个后台线程，因此若在load()方法之后紧接着使用play()方法播放音频，有可能因为没有足够时间加载音频资源而播放不出声音。因此最好将SoundPool资源和activity的生命周期事件（如onCreate和onPause）绑定，或者将SoundPool资源的播放同用户发起的动作事件（如按下按钮或者游戏的过关）绑定也可以。

本秘诀的主activity代码如清单6-10所示，它使用的布局文件和清单6-7相同。按钮点击事件触发SoundPool重复播放鼓音8次（最初的1次加上重复7次）。同时多次点击按钮可以将音频的播放速率从半速到两倍速度之间进行切换。每次最多可以播放10个音频流，也就意味着快速按键10次可以同时播放10个打鼓声。

清单6-10 src/com/cookbook/audio_ex/AudioExamplesSP.java

```
package com.cookbook.audio_ex;

import android.app.Activity;
import android.media.AudioManager;
import android.media.SoundPool;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class AudioExamplesSP extends Activity {
    static float rate = 0.5f;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        Button playDrumButton = (Button) findViewById(R.id.play_pause);

        final SoundPool mySP = new
            SoundPool(10, AudioManager.STREAM_MUSIC, 0);
        final int soundId = mySP.load(this, R.raw.drum_beat, 1);

        playDrumButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                rate = 1/rate;
                mySP.play(soundId, 1f, 1f, 1, 7, rate);
            }
        });
    }
}
```

6.2.5 秘诀 55：添加媒体资源并更新路径

应用程序每新录制完一段音频文件后就可以在系统中将其注册为可用资源。这是通过

MediaStore类来实现的。例如,清单6-11为我们演示了在音频文件myFile被保存后如何将它注册为铃声、通知提示音或警告音,但同时设置其为MP3播放器不可用(因为IS_MUSIC被设置为false)。

清单6-11 向系统注册音频文件示例

```
//reload MediaScanner to search for media and update paths
sendBroadcast(new Intent(Intent.ACTION_MEDIA_MOUNTED,
    Uri.parse("file://"
        + Environment.getExternalStorageDirectory())));
ContentValues values = new ContentValues();
values.put(MediaStore.MediaColumns.DATA, myFile.getAbsolutePath());
values.put(MediaStore.MediaColumns.TITLE, myFile.getName());
values.put(MediaStore.MediaColumns.TIMESTAMP,
    System.currentTimeMillis());
values.put(MediaStore.MediaColumns.MIME_TYPE,
    recorder.getMimeContentType());
values.put(MediaStore.Audio.Media.ARTIST, SOME_ARTIST_HERE);
values.put(MediaStore.Audio.Media.IS_RINGTONE, true);
values.put(MediaStore.Audio.Media.IS_NOTIFICATION, true);
values.put(MediaStore.Audio.Media.IS_ALARM, true);
values.put(MediaStore.Audio.Media.IS_MUSIC, false);
ContentResolver contentResolver = new ContentResolver();
Uri base = MediaStore.Audio.INTERNAL_CONTENT_URI;
Uri newUri = contentResolver.insert(base, values);
String path = contentResolver.getDataFilePath(newUri);
```

此处的ContentValues用于声明文件的某些标准属性,如TITLE、TIMESTAMP和MIME_TYPE,ContentResolver用于在MediaStore内容数据库中创建一个条目,并自动添加该文件的路径。

6.3 视频

和前面秘诀中讨论的关于音频的录制和播放的例子类似,录制和播放视频文件也可使用MediaPlayer/MediaRecorder框架。为了内容的完整性,下面我们简要总结一下其基本步骤。首先,如果要录制视频,需要在AndroidManifest.xml文件中声明权限许可:

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

此处需要选择的标准和前述音频范例不同(注意:这里同样也可以使用DEFAULT,默认设置为下面每个选项列表的第一个选项)。

- ☐ MediaRecorder.VideoSource
 - CAMERA——内置摄像头
- ☐ MediaRecorder.OutputFormat
 - THREE_GPP——3GPP多媒体文件格式
 - MPEG_4——MPEG4多媒体文件格式
- ☐ MediaRecorder.VideoEncoder
 - H263——H.263视频编码

■ H264——H.264视频编码

■ MPEG_4_SP——MPEG4 Simple Profile视频编码

录制视频的步骤如下。

(1) 创建一个MediaRecorder实例:

```
MediaRecorder m_Recorder = new MediaRecorder();
```

(2) 声明媒体源, 当前指定为仅使用摄像头:

```
m_Recorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

(3) 设置输出文件的格式和编码方式:

```
m_Recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
```

```
m_Recorder.setAudioEncoder(MediaRecorder.AudioEncoder.H263);
```

(4) 设置文件保存的路径:

```
m_Recorder.setOutputFile(path);
```

(5) 准备并开始录制:

```
m_Recorder.prepare();
```

```
m_Recorder.start();
```

播放视频的步骤如下。

(1) 创建一个MediaPlayer实例:

```
MediaPlayer m_mediaPlayer = new MediaPlayer();
```

(2) 声明媒体源。可以指定raw资源目录下的某个视频文件:

```
m_mediaPlayer = MediaPlayer.create(this, R.raw.my_video);
```

也可以从文件系统中直接指定要播放的文件 (同样需要调用prepare()方法):

```
m_mediaPlayer.setDataSource(path);
```

```
m_mediaPlayer.prepare();
```

不论使用哪种方法, 这些语句都需要内嵌在try-catch语句块中, 因为指定的视频资源可能不存在。

(3) 开始播放视频:

```
m_mediaPlayer.start();
```

(4) 当播放结束后, 停止MediaPlayer并释放该实例, 以释放相关资源:

```
m_mediaPlayer.stop();
```

```
m_mediaPlayer.release();
```



Android设备内置了很多不同类型的硬件，开发者可以在程序中使用这些硬件。很多设备中都内置了多种传感器，如照相机、加速度计、磁力传感器、压力传感器、温度传感器和近距离（proximity）传感器。电话、蓝牙以及其他无线连接从某种程度来说也是开发者可用的。本章主要介绍如何利用这些硬件的API来丰富程序的用户体验。但需要注意的是，最好在真机中运行这些例子，因为模拟器可能无法真实、准确地反映硬件接口的行为。

7.1 照相机

照相机是Android设备中最常见和最常用的传感器。对于大多数消费者来说，这也是手机设备的一个卖点，并且随着设备的更新换代，其性能也变得越来越好。图像处理应用程序通常用于处理被拍的图像，但是扩增实境（augmented reality）等其他类型应用程序，则是利用摄像头将虚拟图像和真实环境实时叠加到同一个画面。

在应用程序中访问照相机的方式有两种。第一种方式通过声明一个隐式的intent来调用照相机，如第2章所述。隐式的intent将调用默认的照相机接口：

```
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
startActivity(intent);
```

第二种方式是使用Camera类，通过该类可以更灵活地设置照相机。这种方式会创建一个自定义的camera接口，下面的例子将着重介绍这部分内容。访问照相机需要在AndroidManifest.xml文件中添加相应权限。

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

下面的部分默认已经添加了该权限。

秘诀 56：自定义 Camera

在Android系统中，对照相机的控制可抽象为多个组件：

- Camera类——访问照相机硬件；
- Camera.Parameters类——设定照相机参数，如图片大小、图片质量、闪光灯模式和指定GPS位置的方法；

- 相机预览 (preview) 方法——设置相机输出显示, 开启和关闭视频流预览显示;
- SurfaceView类——该类是一个显示相机预览视图的替代容器, 专门用于绘制视图层级中最底层的视图。

在介绍如何将这部分联系在一起前, 首先要介绍程序的界面布局结构。清单7-1所示的main.xml布局文件中包含一个用于盛放照相机输出的SurfaceView控件。

清单7-1 res/layout/main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <SurfaceView android:id="@+id/surface"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    </SurfaceView>

</LinearLayout>
```

清单7-2中的代码通过一个额外的布局文件在视图的顶层添加一个控制接口。这个布局在底部放置了一个按钮控件, 并且在屏幕中间显示所拍的照片。

清单7-2 res/layout/cameraoverlay.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:gravity="bottom"
    android:layout_gravity="bottom">

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_horizontal">
        <Button
            android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="take picture"
        />
    </LinearLayout>
</LinearLayout>
```

主activity中包含多个功能。首先，界面布局设置如下。

(1) 修改窗口设置为半透明并全屏显示。(在这种情况下，系统会隐藏标题栏和通知栏。)

(2) 前面布局文件中定义的SurfaceView(R.id.surface)由相机预览画面填充。为了获取和控制显示界面，每个SurfaceView都绑定一个SurfaceHolder。该activity作为SurfaceHolder的回调页面添加到程序中，并将之设定为SURFACE_TYPE_PUSH_BUFFERS类型，表明它创建了一个“push”surface对象，该surface对象没有自己的缓冲区，所用到的数据由其他对象提供。这样会使图像预览更流畅。

(3) 声明一个LayoutInflater对象，在原有的(main.xml)布局上层填充另一个布局(cameraoverlay.xml)。

下一步是为activity设置一个开始拍照的触发器。

(1) 绑定监听器OnClickListener到cameraoverlay布局底部的按钮上，在按钮被点击时就会拍摄一张照片(mCamera.takePicture())。

(2) 为使用takePicture()方法，我们需要定义三个方法。

- ShutterCallback()方法用于定义拍摄照片后需要的任何效果，例如播放音频让用户知道该图像已经成功捕获。
- PictureCallback()方法用于在硬件有足够内存支持该特性时返回原始图像数据（否则返回数据可能为空）。
- 另外一个PictureCallback()方法用于处理压缩图像数据。它将调用局部方法done()来保存图像。

然后，通过activity保存所拍摄的图像。

(1) 压缩图像的字节数组保存于局部变量tempdata中，以便于后面对它的操作。BitmapFactory对象用于将ByteArray解码为Bitmap对象。

(2) 通过media内容提供者保存图像并返回一个URL地址。如果主activity被其他activity调用，那么主activity将会返回此URL以便调用它的activity获取图像。

(3) 上述过程完成之后，调用finish()方法杀死该activity。

最后，由该activity设置响应，以应对surface视图的变化。

(1) 实现SurfaceHolder.CallBack接口。这需要重载三个方法：

- surfaceCreated()——在surface对象首次创建时调用。这里用于初始化对象；
- surfaceChanged()——在surface对象改变（例如，格式或者大小的改变）时调用；
- surfaceDestroyed()——在surface对象从用户视图中移除之后并在销毁surface对象之前调用，用于清理内存。

(2) 当surface对象改变时，照相机的参数也要相应地改变（例如PreviewSize就是基于surface的大小）。

activity的所有功能如清单7-3所示。

清单7-3 src/com/cookbook/hardware/CameraApplication.java

```
package com.cookbook.hardware;
```

```

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.os.Bundle;
import android.provider.MediaStore.Images;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.View.OnClickListener;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.Toast;

public class CameraApplication extends Activity
    implements SurfaceHolder.Callback {
    private static final String TAG = "cookbook.hardware";
    private LayoutInflater mInflater = null;
    Camera mCamera;
    byte[] tempdata;
    boolean mPreviewRunning = false;
    private SurfaceHolder mSurfaceHolder;
    private SurfaceView mSurfaceView;
    Button takepicture;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        getWindow().setFormat(PixelFormat.TRANSLUCENT);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setContentView(R.layout.main);

        mSurfaceView = (SurfaceView)findViewById(R.id.surface);
        mSurfaceHolder = mSurfaceView.getHolder();
        mSurfaceHolder.addCallback(this);
        mSurfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

        mInflater = LayoutInflater.from(this);

```

```

View overView = mInflater.inflate(R.layout.cameraoverlay, null);
this.addContentView(overView,
    new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT));
takepicture = (Button) findViewById(R.id.button);
takepicture.setOnClickListener(new OnClickListener(){
    public void onClick(View view){
        mCamera.takePicture(mShutterCallback,
            mPictureCallback, mjpeg);
    }
});
}

ShutterCallback mShutterCallback = new ShutterCallback(){
    @Override
    public void onShutter() {}
};
PictureCallback mPictureCallback = new PictureCallback() {
    public void onPictureTaken(byte[] data, Camera c) {}
};
PictureCallback mjpeg = new PictureCallback() {
    public void onPictureTaken(byte[] data, Camera c) {
        if(data !=null) {
            tempdata=data;
            done();
        }
    }
};

void done() {
    Bitmap bm = BitmapFactory.decodeByteArray(tempdata,
                                                0, tempdata.length);
    String url = Images.Media.insertImage(getContentResolver(),
        bm, null, null);
    bm.recycle();
    Bundle bundle = new Bundle();
    if(url!=null) {
        bundle.putString("url", url);

        Intent mIntent = new Intent();
        mIntent.putExtras(bundle);
        setResult(RESULT_OK, mIntent);
    } else {
        Toast.makeText(this, "Picture can not be saved",
            Toast.LENGTH_SHORT).show();
    }
    finish();
}
@Override

```



```
public void surfaceChanged(SurfaceHolder holder, int format,
                           int w, int h) {
    Log.e(TAG, "surfaceChanged");
    try {
        if (mPreviewRunning) {
            mCamera.stopPreview();
            mPreviewRunning = false;
        }

        Camera.Parameters p = mCamera.getParameters();
        p.setPreviewSize(w, h);

        mCamera.setParameters(p);
        mCamera.setPreviewDisplay(holder);
        mCamera.startPreview();
        mPreviewRunning = true;
    } catch (Exception e) {
        Log.d("", e.toString());
    }
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
    Log.e(TAG, "surfaceCreated");
    mCamera = Camera.open();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    Log.e(TAG, "surfaceDestroyed");
    mCamera.stopPreview();
    mPreviewRunning = false;
    mCamera.release();
    mCamera=null;
}
}
```

需要注意的是，通过相机硬件看到的相机预览画面是不标准的，有些Android设备会显示相机预览的两侧背景黑框。这种情况下，可以通过在CameraPreview activity中的onCreate()方法中添加如下的一句代码轻松解决：

```
this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

7.2 其他传感器

小型低功耗的微机电系统 (MEMS) 变得越来越流行。智能手机俨然成为各种传感器的集合体，并且智能手机制造商对传感器精度提出了更高的要求，以满足大众对更高性能手机的需求。正如第1章中所提到的，每一款Android手机都集成了不同的传感器。传感器的两个标准配置

是：用于确定设备倾斜度的三轴加速度计和用于确定指南针方向的三轴磁力传感器。其他设备可能还集成了温度传感器、近距离传感器、光传感器以及陀螺仪等。当前版本的Android软件开发工具包所支持的传感器如表7-1所示。

表7-1 Android SDK可使用的传感器

传感器类型	描 述
TYPE_ACCELEROMETER (加速度计)	测量加速度, 以 m/s^2 (平方秒每米) 为单位
TYPE_ALL	描述传感器类型的常量
TYPE_GYROSCOPE (陀螺仪传感器)	根据角动量测量方向
TYPE_LIGHT (光传感器)	测量环境光, 以勒克司 (lux) 为单位
TYPE_MAGNETIC_FIELD (磁力传感器)	测量地磁场, 以微特斯拉 (micro-Tesla) 为单位
TYPE_PRESSURE (压力传感器)	测量空气压力
TYPE_PROXIMITY (近距离传感器)	测量目标对象的距离, 以厘米为单位
TYPE_TEMPERATURE (温度传感器)	测量温度, 以摄氏度为单位

我们可以通过`getSensorList()`方法列出设备中所有可用的传感器。`SensorManager`管理所有的传感器, 它通过`onSensorChanged()`和`onAccuracyChanged()`两个回调函数为我们提供了各种传感器监听器——用于监听传感器值和准确度的变化。

7.2.1 秘诀 57: 获取设备旋转姿态

在理想状态下, 加速度计测量到的地球引力场的重力加速度为 $G = 9.8 \text{ m/s}^2$, 磁力传感器测量的地球磁场根据地理位置的不同, 变化幅度为 $H = 30 \sim 60 \mu\text{T}$ 。通过这两个向量, 我们可以开发实现一个测量旋转的简单示例, 这两个向量将会在`getRotationMatrix()`方法中用到。本秘诀将展示如何使用这些信息。

设备的坐标系 (也被称为“机身坐标”) 定义如下:

- x轴为屏幕的较短边的方向 (具有菜单键的一侧);
- y轴为屏幕的较长边的方向;
- z轴为和屏幕垂直的方向。

通用坐标系 (也被称为“内部坐标”) 定义如下:

- x轴和y轴以及z轴交叉相对;
- y轴与地面相切, 并指向北极;
- z轴垂直于地面指向天空。

当设备水平放置于桌面, 屏幕朝上同时指向北方时, 这两个坐标系统是一致的。这时加速度计在x、y和z方向的测量值为(0,0,G)。在大多数地点, 即使设备朝向北方, 地球磁场也会轻微以角度 θ 向地面倾斜, 所以它的值一般为(0, $H \cos(\theta)$, $-H \sin(\theta)$)。

当设备倾斜和旋转时, `SensorManager.getRotationMatrix()`方法提供了一个 3×3 的旋转矩阵 $R[]$, 并通过计算设备坐标系与通用坐标系之间的差异为之赋值, 同时也提供了一个 3×3 的倾斜矩阵 $I[]$ (围绕x轴旋转) 并通过计算真实的磁场与理想态的磁场值(0, H, 0)之间的差异为之赋值。

注意，如果设备本身具有加速度或者靠近强磁场，测量值就不一定能够正确地反映地球参考系。

另外一种获得旋转数据的方法是使用 `SensorManager.getOrientation()` 方法。该方法提供了旋转矩阵 `R[]` 和姿态向量 `attitude[]`：

- `attitude[0]`——方位角（以弧度为单位）是围绕通用坐标系 `z` 轴的旋转角度，要求设备必须朝北。它的值介于 $-\pi$ 与 π 之间，0 代表北， $\pi/2$ 代表东；
 - `attitude[1]`——斜度（以弧度为单位）是围绕通用坐标系 `x` 轴的旋转角度，要求将设备沿着较长边屏幕朝上放置。它的值在 $-\pi$ 到 π 之间，0 代表设备朝上， $\pi/2$ 代表设备朝向地面；
 - `attitude[2]`——倾度（以弧度为单位）是围绕通用坐标系 `y` 轴的旋转角度，要求将设备沿着较短边屏幕朝上放置。它的值在 $-\pi$ 到 π 之间，0 代表设备朝上， $\pi/2$ 代表设备朝右。
- 本秘诀将显示屏幕的姿态信息。布局文件包含了一个 ID 为 `attitude` 的文本框，如清单 7-4 所示。

清单 7-4 `res/layout/main.xml`

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

主 activity 如清单 7-5 所示。注册加速度计和磁力传感器，使之向传感器的监听器返回数据。然后传感器事件侦听器（Sensor Event Listener）会判断是哪一个传感器触发了回调函数并进行赋值。程序通过旋转矩阵确定姿态信息，并将其从弧度转化为角度显示在屏幕上。另外还要注意，传感器可采用不同的刷新率值，如下所示：

- `SENSOR_DELAY_FASTEST`——可能的最快更新率（因设备不同，范围在 8 ms 到约 30 ms 不等）；
- `SENSOR_DELAY_GAME`——适合于游戏的更新率（大约为 40 ms）；
- `SENSOR_DELAY_NORMAL`——系统默认值，适合于屏幕方向的变化更新率（大约为 200 ms）；
- `SENSOR_DELAY_UI`——适合于用户界面的更新率（大约为 350 ms）。

清单 7-5 `src/com/cookbook/orientation/OrientationMeasurements.java`

```
package com.cookbook.orientation;

import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
```

```
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;

public class OrientationMeasurements extends Activity {
    private SensorManager myManager = null;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.attitude);
        // Set Sensor Manager
        myManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        myManager.registerListener(mySensorListener,
            myManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
            SensorManager.SENSOR_DELAY_GAME);
        myManager.registerListener(mySensorListener,
            myManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
            SensorManager.SENSOR_DELAY_GAME);
    }

    float[] mags = new float[3];
    float[] accels = new float[3];
    float[] RotationMat = new float[9];
    float[] InclinationMat = new float[9];
    float[] attitude = new float[3];
    final static double RAD2DEG = 180/Math.PI;
    private final SensorEventListener mySensorListener
        = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event)
        {
            int type = event.sensor.getType();

            if(type == Sensor.TYPE_MAGNETIC_FIELD) {
                mags = event.values;
            }
            if(type == Sensor.TYPE_ACCELEROMETER) {
                accels = event.values;
            }

            SensorManager.getRotationMatrix(RotationMat,
                InclinationMat, accels, mags);
            SensorManager.getOrientation(RotationMat, attitude);
            tv.setText("Azimuth, Pitch, Roll:\n"
                + attitude[0]*RAD2DEG + "\n"
```

```

        + attitude[1]*RAD2DEG + "\n"
        + attitude[2]*RAD2DEG);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};
}

```

为了获得一致的数据，一种比较好的做法是避免在onSensorChanged()方法中放置需要进行大量计算的代码。另外要注意，对于串联的传感器数据，重用了SensorEvent。因此对于高精度数据，最好使用clone()方法处理事件的值，例如：

```
accels = event.values.clone();
```

这样可确保数据accels无论在类中何处使用，都不会因为传感器的持续采样而不断变化。

7.2.2 秘诀 58：使用温度传感器和光传感器

温度传感器用于探测手机温度，以便校准内部硬件。光传感器用于测量环境光，从而自动调节屏幕亮度。

这些传感器并不是在所有的手机上都可用，但是如果它们可用，开发者也可以将它们用于其他用途。清单7-6代码向我们演示如何读取这些传感器的参数值。这段代码可以添加到秘诀57中的activity中，以显示读取的结果。

清单7-6 使用温度传感器和光传感器的示例代码

```

private final SensorEventListener mTListener
    = new SensorEventListener(){

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}

    @Override
    public void onSensorChanged(SensorEvent event) {
        Log.v("test Temperature",
            "onSensorChanged:"+event.sensor.getName());
        if(event.sensor.getType()==Sensor.TYPE_TEMPERATURE){
            tv2.setText("Temperature:"+event.values[0]);
        }
    }
};

private final SensorEventListener mLListener
    = new SensorEventListener(){

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}

    @Override
    public void onSensorChanged(SensorEvent event) {

```



```

        Log.v("test Light",
            "onSensorChanged:"+event.sensor.getName());
        if(event.sensor.getType()==Sensor.TYPE_LIGHT){
            tv3.setText("Light:"+event.values[0]);
        }
    }

};

sensorManager.registerListener(mTListener, sensorManager
    .getDefaultSensor(Sensor.TYPE_TEMPERATURE),
    SensorManager.SENSOR_DELAY_FASTEST);
sensorManager.registerListener(mLListener, sensorManager
    .getDefaultSensor(Sensor.TYPE_LIGHT),
    SensorManager.SENSOR_DELAY_FASTEST);

```

7.3 电话

Android中的电话API为我们提供了一种方法监测手机基本信息，包括网络类型、连接状态，并提供了对电话号码字符串操作的工具。

7.3.1 秘诀 59：使用电话管理器

电话 API中有一个名为TelephonyManager的类，它是一个Android系统服务，通过它可以访问设备中关于电话服务的信息。但是电话信息中有些是受权限保护的，因此必须在AndroidManifest.xml文件中进行权限声明方可访问：

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

主activity的代码如清单7-7所示。

清单7-7 src/com/cookbook/hardware/telephony/TelephonyApp.java

```

package com.cookbook.hardware.telephony;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.TelephonyManager;
import android.widget.TextView;

public class TelephonyApp extends Activity {
    TextView tv1;
    TelephonyManager telManager;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv1 =(TextView) findViewById(R.id.tv1);
        telManager = (TelephonyManager)

```

```

        getSystemService(TELEPHONY_SERVICE);

        StringBuilder sb = new StringBuilder();
        sb.append("deviceid:")
            .append(telManager.getDeviceId()).append("\n");
        sb.append("device Software Ver:")
            .append(telManager.getDeviceSoftwareVersion()).append("\n");
        sb.append("Line number:")
            .append(telManager.getLine1Number()).append("\n");
        sb.append("Network Country ISO:")
            .append(telManager.getNetworkCountryIso()).append("\n");
        sb.append("Network Operator:")
            .append(telManager.getNetworkOperator()).append("\n");
        sb.append("Network Operator Name:")
            .append(telManager.getNetworkOperatorName()).append("\n");
        sb.append("Sim Country ISO:")
            .append(telManager.getSimCountryIso()).append("\n");
        sb.append("Sim Operator:")
            .append(telManager.getSimOperator()).append("\n");
        sb.append("Sim Operator Name:")
            .append(telManager.getSimOperatorName()).append("\n");
        sb.append("Sim Serial Number:")
            .append(telManager.getSimSerialNumber()).append("\n");
        sb.append("Subscriber Id:")
            .append(telManager.getSubscriberId()).append("\n");
        sb.append("Voice Mail Alpha Tag:")
            .append(telManager.getVoiceMailAlphaTag()).append("\n");
        sb.append("Voice Mail Number:")
            .append(telManager.getVoiceMailNumber()).append("\n");
        tv1.setText(sb.toString());
    }
}

```

main.xml布局文件如清单7-8所示，运行结果如图7-1所示。

清单7-8 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>

```



图7-1 使用TelephonyManager类的输出结果

7.3.2 秘诀 60：监听电话状态

PhoneStateListener类为我们提供了不同设备的电话状态信息，包括网络服务状态、信号强度以及待收信息指示器（语音信箱）。有些信息需要明确声明许可，如表7-2所示。

表7-2 电话状态监听事件及其对应的权限

电话状态监听器	描 述	权 限
LISTEN_CALL_FORWARDING_INDICATOR	监听呼叫转移指示器的变化	READ_PHONE_STATE
LISTEN_CALL_STATE	监听通话状态变化	READ_PHONE_STATE
LISTEN_CELL_LOCATION	监听手机位置变化	ACCESS_COARSE_LOCATION
LISTEN_DATA_ACTIVITY	监听手机数据通信 上下行方向变化	READ_PHONE_STATE
LISTEN_DATA_CONNECTION_STATE	监听数据连接状态变化	None
LISTEN_MESSAGE_WAITING_INDICATOR	监听待收信息指示器变化	READ_PHONE_STATE
LISTEN_NONE	删除监听器	None
LISTEN_SERVICE_STATE	监听网络服务状态变化	None
LISTEN_SIGNAL_STRENGTHS	监听网络信号强度变化	None

例如，为了监听来电，TelephonyManager需要注册一个监听器以监听PhoneStateListener.LISTEN_CALL_STATE事件。呼叫状态有下面三种：

- CALL_STATE_IDLE——设备的电话服务处于闲置状态；
- CALL_STATE_RINGING——设备接到呼叫请求；

□ CALL_STATE_OFFHOOK——通话中。

本秘诀将在电话呼叫状态发生改变时显示其状态。我们可使用Logcat调试工具（在第12章介绍），在电话呼入或者呼出时，我们会看到这些状态信息。

主activity的代码如清单7-9所示。该段代码中创建了一个新的内部类继承PhoneStateListener类，程序重载了方法onCallStateChanged来捕获手机状态信息的变化。其他可以被重载的方法还有onCallForwardingIndicator()、onCellLocationChanged()和onData-Activity()。

清单7-9 src/com/cookbook/hardware.telephony/HardwareTelephony.java

```
package com.cookbook.hardware.telephony;

import android.app.Activity;
import android.os.Bundle;
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
import android.util.Log;
import android.widget.TextView;

public class HardwareTelephony extends Activity {
    TextView tv1;
    TelephonyManager telManager;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv1 = (TextView) findViewById(R.id.tv1);
        telManager = (TelephonyManager)
            getSystemService(TELEPHONY_SERVICE);

        telManager.listen(new TelListener(),
            PhoneStateListener.LISTEN_CALL_STATE);
    }

    private class TelListener extends PhoneStateListener {
        public void onCallStateChanged(int state, String incomingNumber) {
            super.onCallStateChanged(state, incomingNumber);
            Log.v("Phone State", "state:"+state);
            switch (state) {
                case TelephonyManager.CALL_STATE_IDLE:
                    Log.v("Phone State",
                        "incomingNumber:"+incomingNumber+" ended");
                    break;
                case TelephonyManager.CALL_STATE_OFFHOOK:
                    Log.v("Phone State",
                        "incomingNumber:"+incomingNumber+" picked up");
                    break;
            }
        }
    }
}
```

```

        case TelephonyManager.CALL_STATE_RINGING:
            Log.v("Phone State",
                "incomingNumber: "+incomingNumber+" received");
            break;
        default:
            break;
    }
}
}
}
}

```

7.3.3 秘诀 61：拨打电话号码

若要在应用程序中拨打电话，必须在AndroidManifest.xml文件中添加如下权限：

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

拨打电话的活动既可以使用ACTION_CALL也可使用ACTION_DIALER隐式intent。如果使用ACTION_DIALER intent，将会显示电话拨号器用户界面和准备拨打的电话号码。这通过如下代码实现：

```
startActivity(new Intent(Intent.ACTION_CALL,
    Uri.parse("tel:15102345678")));
```

如果要使用ACTION_CALL intent，则会不显示电话拨号器界面而直接拨打设定的电话号码。可以通过如下代码实现：

```
startActivity(new Intent(Intent.ACTION_DIAL,
    Uri.parse("tel:15102345678")));
```

7.4 蓝牙

蓝牙来自于IEEE802.15.1标准协议，它是一种开放的无线协议，用于支持设备短距离通信。最常见的例子是手机与耳机间使用蓝牙进行通信，但是在其他应用中还可以使用蓝牙进行近距离追踪^①。要在设备之间使用蓝牙进行通信，需要完成以下四个步骤：

- (1) 在设备上开启蓝牙；
- (2) 找到有效范围内的已配对设备或可用设备；
- (3) 连接到设备；
- (4) 在设备之间传递数据。

若要使用蓝牙服务，需要为应用程序设置BLUETOOTH许可权限以接受和发送数据，设置BLUETOOTH_ADMIN许可权限设定蓝牙或者启动蓝牙设备搜索。这些操作需要在AndroidManifest.xml文件中加入如下几行代码：

^① 近距离追踪 (proximity tracking)，是一种利用蓝牙设备进行距离跟踪的应用，例如可以使用手机和配有蓝牙的台式机配合，一旦手机离开台式机一段距离，就自动锁定台式机。——译者注

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

所有的蓝牙API功能都包含在android.bluetooth包中。主要通过以下五个类来实现这些功能：

- ❑ BluetoothAdapter——代表蓝牙无线接口，用于搜索设备和实例化蓝牙连接；
- ❑ BluetoothClass——描述蓝牙设备的一般特征；
- ❑ BluetoothDevice——代表远程蓝牙设备；
- ❑ BluetoothSocket——代表用于和其他蓝牙设备进行数据交换的套接字或连接点；
- ❑ BluetoothServerSocket——代表接入请求的开放套接字监听端口。

我们将会在下面的秘诀中讨论它们的具体使用细节。

7.4.1 秘诀 62：打开蓝牙

我们使用BluetoothAdapter类初始化蓝牙。getDefaultAdapter()方法可获取蓝牙无线接口的信息。如果返回值为null则意味着该设备不支持蓝牙。

```
BluetoothAdapter myBluetooth = BluetoothAdapter.getDefaultAdapter();
```

若要激活蓝牙，则使用BluetoothAdapter实例去查询蓝牙适配器的当前状态。如果蓝牙未启动，可以使用Android内置的ACTION_REQUEST_ENABLE activity请求用户打开蓝牙：

```
if(!myBluetooth.isEnabled()) {
    Intent enableIntent = new Intent(BluetoothAdapter
        .ACTION_REQUEST_ENABLE);
    startActivity(enableIntent);
}
```

7.4.2 秘诀 63：搜索蓝牙设备

在蓝牙被激活后，为了找到已配对蓝牙设备或可用蓝牙设备，需要异步调用BluetoothAdapter实例的startDiscovery()方法。这需要注册一个BroadcastReceiver去监听ACTION_FOUND事件，该事件会在搜索到有新的远程蓝牙设备时通知应用程序。示例代码如清单7-10所示。

清单7-10 搜索蓝牙设备的示例代码

```
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(
                BluetoothDevice.EXTRA_DEVICE);
            Log.v("BlueTooth Testing", device.getName() + "\n"
                + device.getAddress());
        }
    }
}
```

```
};

IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
myBluetooth.startDiscovery();
```

broadcast receiver 也能监听 ACTION_DISCOVERY_STARTED 事件和 ACTION_DISCOVERY_FINISHED事件, 并在搜索开始和结束时通知应用程序。

对于其他蓝牙设备搜索当前设备的情况, 应用程序可通过ACTION_REQUEST_DISCOVERABLE intent使其对搜索可见。该activity会在应用的顶部显示一个对话框, 请求用户是否允许当前设备为可见:

```
Intent discoverableIntent
    = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivity(discoverableIntent);
```

7.4.3 秘诀 64: 与已绑定的蓝牙设备配对

已绑定蓝牙设备是指曾与当前设备配对过的设备。当配对两个蓝牙设备时, 一个设备使用BluetoothServerSocket类作为服务器, 另一个使用BluetoothSocket类作为客户端。为了取得已绑定蓝牙设备列表, 我们可以使用BluetoothAdapter实例的getBondedDevices()方法:

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
```

7.4.4 秘诀 65: 打开蓝牙套接字

为了和其他设备建立蓝牙连接, 应用程序要么实现客户端套接字, 要么实现服务器端套接字。在服务器端和客户端连接之后, 每个设备都会有一个基于相同RFCOMM (一种蓝牙传输协议) 的已连接蓝牙套接字。然而, 客户端和服务器端却是以不同方式获得蓝牙套接字的。当接入请求被接受时, 服务器端接收蓝牙套接字服务实例。客户端则是在其开启连接服务器端的RFCOMM通道时接受蓝牙套接字服务实例。

服务器端的初始化使用通用的客户-服务器端编程模式, 应用程序需要一个开放的套接字服务用于监听接入请求 (与TCP类似)。BluetoothServerSocket接口用于创建一个服务器端监听端口。在连接请求被接受后, 就会返回一个BluetoothSocket以管理连接。

BluetoothServerSocket接口可以通过BluetoothAdapter实例的listenUsingRfcommWithServiceRecord()方法中获得。在获得套接字之后, 通过accept()方法开始监听请求, 等到接受了接入请求或者异常发生时再返回。在accept()方法返回一个有效的连接时, BluetoothSocket会返回。最后, 应调用close()方法释放服务器端口及其资源, 因为RFCOMM协议只允许在一个通道中一次建立一个客户端连接。这样做不会关闭已连接的BluetoothSocket。下面的代码片段演示了如何完成以上操作:

```
BluetoothServerSocket myServerSocket
    = myBluetoothAdapter.listenUsingRfcommWithServiceRecord(name, uuid);
myServerSocket.accept();
myServerSocket.close();
```

需要注意的是，此处对accept()方法的调用是阻塞调用，因此它不应该在主线程中执行。更好的办法是启动一个额外的线程来运行该方法，如清单7-11所示。

清单7-11 创建蓝牙套接字示例

```
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned
        // to mmServerSocket, because mmServerSocket is final
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                break;
            }
            // If a connection was accepted
            if (socket != null) {
                // Do work to manage the connection (in a separate thread)
                manageConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    /** Will cancel the listening socket, and cause thread to finish */
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) { }
    }
}
```

要实现客户端设备的相关功能，需要从远端设备中获得BluetoothDevice实例，然后获取套接字服务以建立连接。为了获得BluetoothSocket，我们需要使用BluetoothDevice方法的createRfcommSocketToServiceRecord(UUID)，其中的UUID来自于listenUsingRfcommWith-

ServiceRecord。在获取套接字后，使用connect()方法初始化连接。该方法也是一个阻塞方法，因此需要在独立的线程中执行，如清单7-12所示。

清单7-12 示例连接到蓝牙套接字

```
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        // Use a temporary object that is later assigned to mmSocket,
        // because mmSocket is final
        BluetoothSocket tmp = null;
        mmDevice = device;

        // Get a BluetoothSocket to connect with the given BluetoothDevice
        try {
            // MY_UUID is the app's UUID string, also used by the server code
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) { }
        mmSocket = tmp;
    }

    public void run() {
        // Cancel discovery because it will slow down the connection
        mAdapter.cancelDiscovery();

        try {
            // Connect the device through the socket. This will block
            // until it succeeds or throws an exception
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and get out
            try {
                mmSocket.close();
            } catch (IOException closeException) { }
            return;
        }

        // Do work to manage the connection (in a separate thread)
        manageConnectedSocket(mmSocket);
    }

    /** Will cancel an in-progress connection, and close the socket */
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

连接建立以后，蓝牙设备之间就可以通过常用的InputStream和OutputStream方法读取和发送数据了。

7.4.5 秘诀 66：使用设备振动功能

振动是所有手机共有的功能。若要控制Android设备的振动，必须在AndroidManifest.xml文件中声明权限如下：

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

此后就可以使用设备振动功能了，它是系统框架提供的另一项Android系统服务。通过Vibrator类可以获取振动服务。

```
Vibrator myVib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
```

程序实例化了一个Vibrator对象后，只需调用其vibrate()方法就可以开启设备振动功能。

```
myVib.vibrate(3000); // 振动3秒钟
```

如果需要在振动完成之前结束振动，可以调用cancel()方法在振动结束之前停止设备振动。

```
myVib.cancel(); // 取消振动
```

如果想要实现有节奏的振动。可以定义一个“振动-暂停”序列来实现该功能。例如：

```
long[] pattern = {2000,1000,5000};
myVib.vibrate(pattern,1);
```

这样做可以使设备等待2秒，然后以“振动1秒，再暂停5秒”这种方式无限循环地振动。方法vibrate()的第二个参数是上一行中pattern的重复模式。也可以将它设为-1，表示不循环执行pattern序列。

7.4.6 秘诀 67：访问无线网络

许多应用程序都会使用Android设备的网络连接功能。为了更好地理解如何根据网络变化处理应用程序的行为，Android系统提供了获取底层网络状态的方法。其原理是通过广播intent来通知应用程序组件网络连接发生变化，并提供了对网络设置和连接的控制。

Android系统通过ConnectivityManager类提供了一个系统服务，使得开发者能够监视网络连接状态、设置首选网络连接、管理连接故障切换等。初始化该服务的语句如下所示。

```
ConnectivityManager myNetworkManager
    = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
```

若要使用连接管理器（connectivity manager），则需要在AndroidManifest.xml文件中添加相应权限：

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

连接管理器提供了getNetworkInfo()和getActiveNetworkInfo()两个方法，以通过NetworkInfo类获得当前网络的详细情况。然而，更好的监测网络变化的方法是创建一个广播接收器，如下面的例子所示。

```

private BroadcastReceiver mNetworkReceiver = new BroadcastReceiver(){
    public void onReceive(Context c, Intent i){
        Bundle b = i.getExtras();
        NetworkInfo ni = (NetworkInfo)
            b.get(ConnectivityManager.EXTRA_NETWORK_INFO);
        if(ni.isConnected()){
            //do the operation
        }else{
            //announce the user the network problem
        }
    }
};

```

定义广播接收器后，就可以注册监听ConnectivityManager.CONNECTIVITY_ACTION intent了。

```

this.registerReceiver(mNetworkReceiver,
    new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));

```

前面定义的广播接收器实例mNetworkReceiver只接收来自于ConnectivityManager.EXTRA_NETWORK_INFO的NetworkInfo对象。然而，通过连接管理器还可以获得其具有的更多信息。这些不同类型的信息如表7-3所示。

表7-3 从连接管理器中可能获得的信息

信息类型	描 述
EXTRA_EXTRA_INFO	包含关于网络状态的额外信息
EXTRA_IS_FAILOVER	如果当前连接是故障切换网络的结果，则返回布尔值
EXTRA_NETWORK_INFO	返回NetworkInfo对象
EXTRA_NO_CONNECTIVITY	如果无网络连接，则返回布尔值
EXTRA_OTHER_NETWORK_INFO	返回一个指向网络断开时用于故障切换的可用网络的NetworkInfo对象
EXTRA_REASON	返回一个描述连接失败原因的字符串

ConnectivityManager还具有控制网络硬件和故障切换偏好设置功能。可以使用setNetworkPreference()方法选择网络类型。若要更改网络，应用程序还需要在AndroidManifest.xml文件添加额外的权限：

```

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

```


由于具有动态的内容和更强的交互性，基于网络的应用程序更具使用价值。网络使得从社交网络到云计算等各种功能成为现实。

本章重点介绍短消息服务（SMS）、基于互联网资源的应用程序以及社交网络应用程序。短消息是一种通信服务组件，移动电话设备之间使用它传递短文本信息。基于互联网资源的应用依赖网站内容（Web Content），如HTML（超文本标记语言）、XML（可扩展标记语言）、JSON（JavaScript 对象标记）。而社交网络应用程序，如Twitter，是人与人之间相互交流的一种重要手段。

8.1 使用短信息

通过SmsManager类，Android系统提供了完整的短消息服务。早期的Android版本将SmsManager类放在android.telephony.gsm包中。自Android 1.5版本以后，SmsManager同时支持GSM和CDMA移动电话标准，于是现在SmsManager类被放在了android.telephony包中。

使用SmsManager类发送短消息非常简单。具体步骤如下所示。

(1) 在AndroidManifest.xml文件中声明发送短消息的权限许可：

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

(2) 通过SmsManager.getDefault()静态方法获得短消息管理器（SMS manager）实例：

```
SmsManager mySMS = SmsManager.getDefault();
```

(3) 定义消息发送的目标电话号码，以及要发送的短消息内容。并使用sendTextMessage()方法发送短消息到另一个设备：

```
String destination = "16501234567";  
String msg = "Sending my first message";  
mySMS.sendTextMessage(destination, null, msg, null, null);
```

以上这些对于发送短消息已经足够了。不过，我们还可以看一下上述调用方法中被设置为null的三个其他参数的使用方法。

- 第2个参数指定要使用的SMS服务中心。设置为null代表使用默认服务中心。
- 第4个参数是一个PendingIntent对象，用于跟踪短消息是否已被发送。

□ 第5个参数是一个PendingIntent对象，用于跟踪短消息是否已被接收。

若要使用第4个参数和第5个参数，则需要声明一个发送消息的intent和一个消息已接收的intent，代码如下。

```
String SENT_SMS_FLAG = "SENT_SMS";
String DELIVER_SMS_FLAG = "DELIVER_SMS";

Intent sentIn = new Intent(SENT_SMS_FLAG);
PendingIntent sentPin = PendingIntent.getBroadcast(this,0,sentIn,0);

Intent deliverIn = new Intent(SENT_SMS_FLAG);
PendingIntent deliverPin
    = PendingIntent.getBroadcast(this,0,deliverIn,0);
```

然后为每一个PendingIntent都注册一个BroadcastReceiver，用来获取结果。

```
BroadcastReceiver sentReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
        switch(getResultCode()){
            case Activity.RESULT_OK:
                //sent SMS message successfully;
                break;
            default:
                //sent SMS message failed
                break;
        }
    }
};

BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
    @Override public void onReceive(Context c, Intent in) {
        //SMS delivered actions
    }
};

registerReceiver(sentReceiver, new IntentFilter(SENT_SMS_FLAG));
registerReceiver(deliverReceiver, new IntentFilter(DELIVER_SMS_FLAG));
```

大多数短消息服务限定每条文本信息不得超过140个字符。为了确保信息符合该限制条件，调用divideMessage()方法将消息文本按以字符数上限拆分为若干片段。然后再调用sendMultipartTextMessage()方法而不是sendTextMessage()方法发送信息。这两个方法的区别在于后者使用了ArrayList以及PendingIntent。

```
ArrayList<String> multiSMS = mySMS.divideMessage(msg);
ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliverIns = new ArrayList<PendingIntent>();

for(int i=0; i< multiSMS.size(); i++){
    sentIns.add(sentIn);
    deliverIns.add(deliverIn);
}
```

```
mySMS.sendMultipartTextMessage(destination, null,
                                multiSMS, sentIns, deliverIns);
```

秘诀 68：收到短消息后自动回复

由于很多短消息可能要等到数小时之后才被接收者查阅，所以本秘诀将介绍如何在收到短信息时自动向对方回复信息。我们可以创建一个Android后台服务接收传入的短消息，还有一种方法是在AndroidManifest.xml文件中注册一个broadcast receiver实现。

应用程序必须在AndroidManifest.xml文件中声明发送和接收短信息的权限许可，如清单8-1所示。此外该段代码中还声明了一个用于创建自动回复的主activity，即SMSResponder，以及一个在收到短消息时发送响应的服务，即ResponderService。

清单8-1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.SMSResponder"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".SMSResponder"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:enabled="true" android:name=".ResponderService">
        </service>
    </application>
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>
```

清单8-2是main.xml布局文件，包含了三个使用线性布局方式的视图控件：TextView控件用于显示自动回复的内容，按钮用于提交对自动回复消息的更改，EditText控件则可以让用户输入回复消息的内容。

清单8-2 res/layout/main.xml

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

```
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:textSize="18dp"
    />
    <Button android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Change my response"
    />
    <EditText android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>
```

主activity页面如清单8-3所示。它启动了监听和自动回复短消息的服务，同时它也允许用户修改回复内容并将其保存在一个SharedPreferences对象中，以备将来使用。

清单8-3 src/com/cookbook/SMSResponder/SMSResponder.java

```
package com.cookbook.SMSResponder;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class SMSResponder extends Activity {
    TextView tv1;
    EditText ed1;
    Button btl;
    SharedPreferences myprefs;
    Editor updater;
    String reply=null;

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

        myprefs = PreferenceManager.getDefaultSharedPreferences(this);
```

```

    tv1 = (TextView) this.findViewById(R.id.display);
    ed1 = (EditText) this.findViewById(R.id.editText);
    bt1 = (Button) this.findViewById(R.id.submit);

    reply = myprefs.getString("reply",
        "Thank you for your message. I am busy now. "
        + "I will call you later");
    tv1.setText(reply);

    updater = myprefs.edit();
    ed1.setHint(reply);
    bt1.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            updater.putString("reply", ed1.getText().toString());
            updater.commit();
            SMSResponder.this.finish();
        }
    });

    try {
        // start Service
        Intent svc = new Intent(this, ResponderService.class);
        startService(svc);
    }
    catch (Exception e) {
        Log.e("onCreate", "service creation problem", e);
    }
}
}

```

大段代码都包含在清单8-4所示的service中。代码首先检索该应用程序的Shared-Preferences存储对象。然后注册一个broadcast receiver用于监听呼入和呼出的短消息。其实本秘诀中并没有使用监听呼出短消息的broadcast receiver，只是为了程序的完整性而写了出来。

监听呼入短消息事件的broadcast receiver使用一个bundle对象获取协议描述单元（Protocol Description Unit, PDU），其中包括了短消息文本内容和附加的短消息元数据，并将它们解析到一个对象数组中。又通过createFromPdu()方法将该对象数组转换为SmsMessage对象数组。然后使用getOriginatingAddress()方法获取发送者的电话号码，并同时使用getMessageBody()方法获取短消息内容文本。

本秘诀中，在获取发送者电话号码之后，程序会执行respond()方法，试着获取存储在SharedPreferences对象中用于自动回复的数据内容。如果没有相关存储数据，应用程序就会使用默认值。接着，应用程序分别为跟踪短消息发送状态和短消息送达状态各创建了一个PendingIntent对象，并使用divideMessage()方法确保发送的短消息不会超过最大字数限制。当所有准备工作完成后，最后调用sendMultitextMessage()方法发送短消息。

清单8-4 src/com/cookbook/SMSresponder/ResponderService.java

```
package com.cookbook.SMSresponder;

import java.util.ArrayList;

import android.app.Activity;
import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class ResponderService extends Service {
    //The Action fired by the Android-System when a SMS was received.
    private static final String RECEIVED_ACTION =
        "android.provider.Telephony.SMS_RECEIVED";
    private static final String SENT_ACTION="SENT_SMS";
    private static final String DELIVERED_ACTION="DELIVERED_SMS";

    String requester;
    String reply="";
    SharedPreferences myprefs;

    @Override
    public void onCreate() {
        super.onCreate();
        myprefs = PreferenceManager.getDefaultSharedPreferences(this);

        registerReceiver(sentReceiver, new IntentFilter(SENT_ACTION));
        registerReceiver(deliverReceiver,
            new IntentFilter(DELIVERED_ACTION));

        IntentFilter filter = new IntentFilter(RECEIVED_ACTION);
        registerReceiver(receiver, filter);

        IntentFilter attemptedfilter = new IntentFilter(SENT_ACTION);
        registerReceiver(sender,attemptedfilter);
    }

    private BroadcastReceiver sender = new BroadcastReceiver(){
```

```

@Override
public void onReceive(Context c, Intent i) {
    if(i.getAction().equals(SENT_ACTION)) {
        if(getResultCode() != Activity.RESULT_OK) {
            String recipient = i.getStringExtra("recipient");
            requestReceived(recipient);
        }
    }
}

BroadcastReceiver sentReceiver = new BroadcastReceiver() {
    @Override public void onReceive(Context c, Intent in) {
        switch(getResultCode()) {
            case Activity.RESULT_OK:
                //sent SMS message successfully;
                smsSent();
                break;
            default:
                //sent SMS message failed
                smsFailed();
                break;
        }
    }
};

public void smsSent() {
    Toast.makeText(this, "SMS sent", Toast.LENGTH_SHORT);
}

public void smsFailed() {
    Toast.makeText(this, "SMS sent failed", Toast.LENGTH_SHORT);
}

public void smsDelivered() {
    Toast.makeText(this, "SMS delivered", Toast.LENGTH_SHORT);
}

BroadcastReceiver deliverReceiver = new BroadcastReceiver() {
    @Override public void onReceive(Context c, Intent in) {
        //SMS delivered actions
        smsDelivered();
    }
};

public void requestReceived(String f) {
    Log.v("ResponderService", "In requestReceived");
    requester=f;
}

BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override

```



```

public void onReceive(Context c, Intent in) {
    Log.v("ResponderService", "On Receive");
    reply="";
    if(in.getAction().equals(RECEIVED_ACTION)) {
        Log.v("ResponderService", "On SMS RECEIVE");

        Bundle bundle = in.getExtras();
        if(bundle!=null) {
            Object[] pdus = (Object[])bundle.get("pdus");
            SmsMessage[] messages = new SmsMessage[pdus.length];
            for(int i = 0; i<pdus.length; i++) {
                Log.v("ResponderService", "FOUND MESSAGE");
                messages[i] =
                    SmsMessage.createFromPdu((byte[])pdus[i]);
            }
            for(SmsMessage message: messages) {
                requestReceived(message.getOriginatingAddress());
            }
            respond();
        }
    }
}

@Override
public void onStart(Intent intent, int startId) {
    super.onStart(intent, startId);
}

public void respond() {
    Log.v("ResponderService", "Responding to " + requester);
    reply = myprefs.getString("reply",
        "Thank you for your message. I am busy now. "
        + "I will call you later");
    SmsManager sms = SmsManager.getDefault();
    Intent sentIn = new Intent(SENT_ACTION);
    PendingIntent sentPIn = PendingIntent.getBroadcast(this,
        0, sentIn, 0);
    Intent deliverIn = new Intent(DELIVERED_ACTION);
    PendingIntent deliverPIn = PendingIntent.getBroadcast(this,
        0, deliverIn, 0);
    ArrayList<String> Msgs = sms.divideMessage(reply);
    ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
    ArrayList<PendingIntent> deliverIns =
        new ArrayList<PendingIntent>();

    for(int i=0; i< Msgs.size(); i++) {
        sentIns.add(sentPIn);
        deliverIns.add(deliverPIn);
    }
}

```



```

    }

    sms.sendMultipartTextMessage(requester, null,
                                Msgs, sentIns, deliverIns);
}

@Override
public void onDestroy() {
    super.onDestroy();
    unregisterReceiver(receiver);
    unregisterReceiver(sender);
}

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

```

8.2 使用 Web 内容

我们可以使用第2章提到的隐式intent ACTION_VIEW打开互联网浏览器显示Web内容，例如：

```

Intent i = new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://www.google.com"));
startActivity(i);

```

开发者还可以使用WebView视图控件开发自己的Web浏览器，WebView是一种用来呈现网页内容的视图控件。和其他视图控件一样，它可以占据全屏也可以只占用activity页面的一部分。WebView使用WebKit渲染网页，它是在苹果公司Safari浏览器中使用的开源浏览器引擎。

8.2.1 秘诀 69：定制 Web 浏览器

获得WebView对象的方法有两种。我们可以通过构造函数实例化一个WebView对象：

```
WebView webview = new WebView(this);
```

也可以在布局文件中定义一个WebView控件，并在activity中声明如下：

```
WebView webView = (WebView) findViewById(R.id.webview);
```

在得到WebView对象后，就可以使用loadURL()方法显示网页了：

```
webview.loadUrl("http://www.google.com/");
```

我们可以通过WebSettings类来定义浏览器功能。例如，使用setBlockNetworkImage()方法可以使浏览器屏蔽网络图片以减少数据流量，使用setDefaultFontSize()方法可以设置网页字体大小。其他一些常用设置如下面的例子所示：

```

WebSettings webSettings = webView.getSettings();
webSettings.setSaveFormData(false);
webSettings.setJavaScriptEnabled(true);

```

```

webSettings.setSavePassword(false);
webSettings.setSaveFormData(false);
webSettings.setJavaScriptEnabled(true);
webSettings.setSupportZoom(true);

```

8.2.2 秘诀 70：使用 HTTP GET 请求

除了打开浏览器或者在应用程序中通过WebView微件定义基于Webkit的浏览器外，开发者或许还想开发原生的基于互联网的应用程序。也就是说应用程序必须依赖互联网上的原始数据，例如图片、媒体文件、XML数据等。所有相关数据都可以被载入，这对创建一个社交网络应用程序来说是非常重要的。在Android库中，主要由两个包来处理网络通信：`java.net`和`android.net`。

本秘诀使用HTTP GET获取XML或JSON（参见 <http://www.json.org/>）数据。特别要指出的是，本秘诀使用了Google搜索提供的REST（Representational State Transfer，表述性状态转移）API作为演示，查询请求语句如下所示：

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=
```

更多关于Google AJAX（Asynchronous JavaScript And XML，异步JavaScript和XML）搜索的信息可参考：<http://code.google.com/apis/ajaxsearch/>。

不管搜索何种主题，我们只需将要查询的主题添加到查询语句后面即可。例如，若要搜索关于美国职业篮球联赛（NBA）的相关信息，可使用下面的查询语句，并会返回JSON数据：

```
http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=NBA
```

该activity需要具有访问Internet的许可权限才能运行。因此我们应该在AndroidManifest.xml文件中添加下面的语句：

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

main.xml布局文件如清单8-5所示。其中定义了三个视图控件：EditText控件允许用户输入搜索主题，Button控件用于触发搜索动作，TextView控件则用来显示搜索结果。

清单8-5 res/layout/main.xml

```

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

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

    <Button
        android:id="@+id/submit"

```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search"
    />
    <TextView
        android:id="@+id/display"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="@string/hello"
        android:textSize="18dp"
    />
</LinearLayout>

```

主activity的代码如清单8-6所示。在它的onCreate()方法中对这三个布局元素进行了初始化。在按钮的OnClickListener中调用了SearchRequest()方法。此处使用了Google REST API URL组成检索工程，并构造了一个URL对象。再使用URL类实例获取一个URLConnection实例。

URLConnection实例对象可以获取网络连接状态。当URLConnection返回的结果代码为HTTP_OK时，代表整个HTTP事务完成。然后本次HTTP事务中返回的JSON数据可以完整地保存到一个字符串中。我们使用InputStreamReader对象传递到BufferedReader来读取数据，并创建一个字符串实例。在得到HTTP的结果后，activity将调用ProcessResponse()方法解析JSON数据。我们必须理解传入的JSON数据的结构才能解析它。在本例中，Google REST API以JSONArray对象的形式返回数据结果。图8-1为以NBA为主题进行搜索所获结果的截图。



图8-1 Google REST API查询搜索结果

清单8-6 src/com/cookbook/internet/search/GoogleSearch.java

```
package com.cookbook.internet.search;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.security.NoSuchAlgorithmException;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

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;
import android.widget.EditText;
import android.widget.TextView;

public class GoogleSearch extends Activity {
    /** Called when the activity is first created. */
    TextView tv1;
    EditText ed1;
    Button bt1;
    static String url =
"http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv1 = (TextView) this.findViewById(R.id.display);
        ed1 = (EditText) this.findViewById(R.id.editText);
        bt1 = (Button) this.findViewById(R.id.submit);

        bt1.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                if(ed1.getText().toString()!=null) {
                    try{
                        ProcessResponse(
                            SearchRequest(ed1.getText().toString()));
                    } catch(Exception e) {
                        Log.v("Exception google search",
                            "Exception:"+e.getMessage());
                    }
                }
            }
        });
    }
}
```

```

        }
        ed1.setText("");
    }
    });
}

public String SearchRequest(String searchString)
    throws MalformedURLException, IOException {
    String newFeed=url+searchString;
    StringBuilder response = new StringBuilder();
    Log.v("gsearch", "gsearch url:"+newFeed);
    URL url = new URL(newFeed);

    HttpURLConnection httpconn
        = (HttpURLConnection) url.openConnection();

    if(httpconn.getResponseCode()==HttpURLConnection.HTTP_OK) {
        BufferedReader input = new BufferedReader(
            new InputStreamReader(httpconn.getInputStream()),
            8192);
        String strLine = null;
        while ((strLine = input.readLine()) != null) {
            response.append(strLine);
        }
        input.close();
    }
    return response.toString();
}

public void ProcessResponse(String resp) throws IllegalStateException,
    IOException, JSONException, NoSuchAlgorithmException {
    StringBuilder sb = new StringBuilder();
    Log.v("gsearch", "gsearch result:"+resp);
    JSONObject mResponseObject = new JSONObject(resp);
    JSONObject responseObject
        = mResponseObject.getJSONObject("responseData");
    JSONArray array = responseObject.getJSONArray("results");
    Log.v("gsearch", "number of resultst:"+array.length());
    for(int i = 0; i<array.length(); i++) {
        Log.v("result", i+" "+array.get(i).toString());
        String title = array.getJSONObject(i).getString("title");
        String urllink = array.getJSONObject(i)
            .getString("visibleUrl");

        sb.append(title);
        sb.append("\n");
        sb.append(urllink);
        sb.append("\n");
    }
    tv1.setText(sb.toString());
}
}
}

```

8.2.3 秘诀 71：使用 HTTP POST 请求

有时我们需要从因特网上获取一些纯二进制数据，例如图片、视频、音频文件等，这可以通过 `setRequestMethod()` 方法使用 HTTP POST 协议来完成，例如：

```
httpconn.setRequestMethod(POST);
```

通过互联网访问数据可能会花费很多时间，并造成不可预知的结果。因此，只要是获取网络数据，都应该开启一个独立的线程来执行。除了第3章介绍的方法外，另一个 Android 系统内置类 `AsyncTask` 也可以执行后台操作，并能在 UI 线程中更新执行结果，而不需要操作线程或者使用 handler。因此，POST 方法可以通过如下代码实现异步操作。

```
private class mygoogleSearch extends AsyncTask<String, Integer, String> {

    protected String doInBackground(String... searchKey) {

        String key = searchKey[0];

        try {
            return SearchRequest(key);
        } catch (Exception e) {
            Log.v("Exception google search",
                "Exception:"+e.getMessage());
            return "";
        }
    }

    protected void onPostExecute(String result) {
        try {
            ProcessResponse(result);
        } catch (Exception e) {
            Log.v("Exception google search",
                "Exception:"+e.getMessage());
        }
    }
}
```

该代码片段可以添加在清单 8-6 中的 `GoogleSearch.java` 末尾。然后在按钮的 `OnClickListener` 方法中添加一处更改，实现同样的结果，如下：

```
new mygoogleSearch().execute(ed1.getText().toString());
```

8.3 社交网络

Twitter 是一种社交网络和微博服务，使得用户以 tweet 的形式发布和阅读消息。Twitter 被描述为“互联网上的短消息服务”。的确，每条 tweet 的内容不能超过 140 个字符。Twitter 用户可以接收其他人的 tweet，也可以将自己的 tweet 发送给他人。

秘诀 72：与 Twitter 整合

目前有一些第三方库文件可以帮助我们将Twitter整合到Android应用程序之中（出自<http://dev.twitter.com/pages/libraries#java>）：

- Twitter4J（由Yusuke Yamamoto开发）——在BSD许可证下发布，是一个针对TwitterAPI的、开源的、使用maven编译，并可安全使用Google App Engine的Java 类库；
- java-twitter（由DeWitt Clinton开发）——该类库为操作TwitterAPI提供了一个纯Java接口；
- jtwtter（由Daniel Winterstein开发）——连接到Twitter的开源纯Java接口；
- Twitter Client（开发者 Gist, Inc.）——连接到流式传输API的Java客户端。

在本秘诀中我们将使用由Yusuke Yamamoto开发的Twitter4J类库，该类库的用户文档见<http://twitter4j.org/en/javadoc/overview-summary.html>。本秘诀实现了用户登录Twitter和发布tweet的功能。同时它也会检索tweet的更新并将它们显示到屏幕上。

程序中有用户登录界面和状态更新界面（见图8-2）两个界面。状态更新界面中的EditText框的下方显示该登录用户的最新状态。这两个activity（分别对应一个界面）和访问Internet的权限许可需要在AndroidManifest.xml文件中声明，如清单8-7所示。

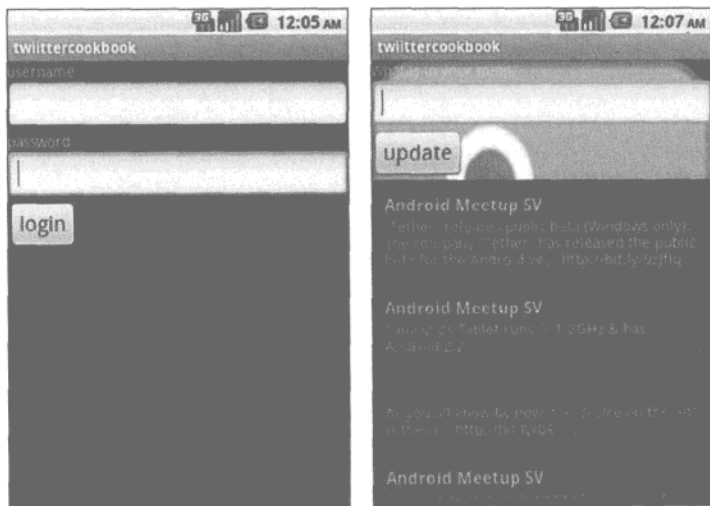


图8-2 Twitter秘诀中登录界面（左）和tweet界面（右）

清单8-7 AndroidManifest.xml

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

```

<application android:icon="@drawable/icon"
    android:label="@string/app_name">
    <activity android:name=".TwitterCookBook"
        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=".UpdateAndList" />
</application>
<uses-permission android:name="android.permission.INTERNET"/>
</manifest>

```

秘诀中需要的布局文件如下：

- login.xml——登录界面，如清单8-8所示；
- main.xml——更新状态及状态显示界面，如清单8-9所示；
- usertimelinerow.xml——显示每个状态的时间线具体的视图，如清单8-10所示。

清单8-8 res/layout/login.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="username"
    />
    <EditText
        android:id="@+id/userText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="password"
    />
    <EditText
        android:id="@+id/passwordText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:password="true"
        android:singleLine="true"
    />

```



```

/>
<Button
    android:id="@+id/loginButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="login"
    android:textSize="20dp"
/>
</LinearLayout>

```

清单8-9 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/twitter">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="What is in your mind?"
    />
    <EditText
        android:id="@+id/userStatus"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <Button
        android:id="@+id/updateButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="update"
        android:textSize="20dp"
    />
    <ListView
        android:layout_width="fill_parent"
        android:dividerHeight="1px"
        android:layout_height="fill_parent"
        android:id="list"
    />
</LinearLayout>

```

清单8-10 res/layout/usertimelinerow.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"

```

```

        android:layout_alignLeft="@+id/name"
        android:layout_below="@+id/name"
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:padding="12dip">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/name"
            android:layout_marginRight="4dp" android:text="Diary Title "
            android:textStyle="bold" android:textSize="16dip" />
        <TextView android:id="@+id/msg"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Date Recorded"
            android:textSize="14dip" />
    </LinearLayout>

```

程序需要两个activity，一个用于登录到Twitter，另一个用于更新tweet列表。其中用于登录的activity如清单8-11所示，它包含用于输入用户名和密码的EditText对象，一个提交登录数据的Button对象，还有一个SharedPreferences存储对象用于在首次登录成功后保存登录信息，以及twitter4j类库中的Twitter对象。

应用程序启动时，首先会检查SharedPreferences对象中是否存在登录信息。如果是的话，就会将登录信息预填充至EditText文本框中。当用户按下Button按钮时，应用程序便会使用文本框中的用户名和密码数据初始化Twitter对象。当Twitter对象被初始化后，就会调用getFollowersIDs()方法验证此次登录是否有效。如果登录无效则会抛出异常，在Toast弹出提示框中呈现登录失败信息。

清单8-11 src/com/cookbook/twitter/TwitterCookBook.java

```

package com.cookbook.twitter;

import twitter4j.Twitter;
import twitter4j.TwitterFactory;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class TwitterCookBook extends Activity {
    SharedPreferences myprefs;
    EditText userET, passwordET;

```



```

Button loginBT;
static Twitter twitter;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    myprefs = PreferenceManager.getDefaultSharedPreferences(this);
    final String username = myprefs.getString("username", null);
    final String password = myprefs.getString("password", null);
    setContentView(R.layout.login);
    userET = (EditText)findViewById(R.id.userText);
    passwordET = (EditText)findViewById(R.id.passwordText);
    loginBT = (Button)findViewById(R.id.loginButton);
    userET.setText(username);
    passwordET.setText(password);
    loginBT.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            try {

                twitter = new TwitterFactory()
                    .getInstance(userET.getText().toString(),
                                passwordET.getText().toString());
                twitter.getFollowersIDs();
                Intent i = new Intent(TwitterCookBook.this,
                                    UpdateAndList.class);
                startActivity(i);

                Editor ed = myprefs.edit();
                ed.putString("username", userET.getText().toString());
                ed.putString("password",
                            passwordET.getText().toString());
                ed.commit();
                finish();

            } catch (Exception e) {
                e.printStackTrace();
                Toast.makeText(TwitterCookBook.this, "login failed!!",
                              Toast.LENGTH_SHORT).show();
            }
        }
    });
}

```

登录成功后，程序随即启动UpdateAndList activity。如清单8-12所示，它包含一个供用户输入tweet消息的EditText对象，一个用于提交用户tweet状态消息到Twitter服务器的Button对象，一个来自于twitter4j类库的Twitter对象，和用于保存从Twitter对象返回的数据状态的

ResponseList^①，以及一个管理状态数据的自定义适配器。

该activity调用getHomeTimeline()方法获取基于时间顺序的状态信息，这些信息在用户登录后将显示在Twitter的首页上。需要注意的是，这里将所有需要访问网络资源的方法调用都放在了AsyncTask类中，以避免UI线程挂起。用户每次提交tweet状态信息时，都会调用getHomeTimeline()方法，并更新数据适配器。

为了以ListView的格式显示状态信息，该activity继承了ListActivity类。ListActivity中定义了一个名为UserTimeLineAdapter的自定义BaseAdapter。该适配器会将ResponseList<Status> userTimeLine中的用户状态信息映射到ListView列表中显示。

本例中的ListActivity包含有两个AsyncTask类：setup和loadstatus。它们都调用了相同的方法getHomeTimeLine()。唯一的区别在于，setup类中实例化了适配器类，并指定了UserTimeLineAdapter作为该ListActivity的适配器，而loadstatus类仅通知适配器UserTimeLineAdapter用户数据已更新。

清单8-12 src/com/cookbook/twitter/UpdateAndList.java

```
package com.cookbook.twitter;

import twitter4j.ResponseList;
import twitter4j.Status;
import twitter4j.Twitter;
import android.app.ListActivity;
import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.BaseAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class UpdateAndList extends ListActivity {
    EditText userET;
    Button updateBT;
    Twitter twitter;
    ResponseList<Status> userTimeline;
    UserTimeLineAdapter myAdapter;

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

① ResponseList，为twitter4j类库中定义的数据类型，用于替代List。——译者注

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);
userET = (EditText)findViewById(R.id.userStatus);

updateBT = (Button)findViewById(R.id.updateButton);

twitter = TwitterCookBook.twitter;
setup stup = new setup();
stup.execute();

updateBT.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        try {
            twitter.updateStatus(userET.getText().toString());
            loadstatus ldstatus = new loadstatus();
            ldstatus.execute();
            userET.setText("");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
}

private class UserTimeLineAdapter extends BaseAdapter{
    private LayoutInflater mInflater;

    public UserTimeLineAdapter(Context context) {
        mInflater = LayoutInflater.from(context);
    }

    @Override
    public int getCount() {
        return userTimeline.size();
    }

    @Override
    public Status getItem(int i) {
        return userTimeline.get(i);
    }

    @Override
    public long getItemId(int i) {
        return i;
    }

    @Override
    public View getView(int arg0, View arg1, ViewGroup arg2) {
        final ViewHolder holder;
        View v = arg1;
        if ((v == null) || (v.getTag() == null)) {

```



```
        v = mInflater.inflate(R.layout.usertimelinerow, null);
        holder = new ViewHolder();
        holder.mName = (TextView)v.findViewById(R.id.name);
        holder.mStatus = (TextView)v.findViewById(R.id.msg);
        v.setTag(holder);
    } else {
        holder = (ViewHolder) v.getTag();
    }

    holder.status= getItem(arg0);
    holder.mName.setText(holder.status.getUser().getName());
    holder.mStatus.setText(holder.status.getText());

    v.setTag(holder);

    return v;
}
public class ViewHolder {
    Status status;
    TextView mName;
    TextView mStatus;
}

private class setup extends AsyncTask<String, Integer, String> {

    protected String doInBackground(String... searchKey) {

        try{
            userTimeline = twitter.getHomeTimeline();
            return "";
        }catch(Exception e){
            Log.v("Exception Twitter query",
                "Exception:"+e.getMessage());
            return "";
        }
    }

    protected void onPostExecute(String result) {
        try {
            myAdapter = new UserTimeLineAdapter(UpdateAndList.this);
            UpdateAndList.this.setListAdapter(myAdapter);
        } catch(Exception e) {
            Log.v("Exception Twitter query",
                "Exception:"+e.getMessage());
        }
    }
}
```

```
private class loadstatus extends AsyncTask<String, Integer, String> {

    protected String doInBackground(String... searchKey) {

        try {
            userTimeline = twitter.getHomeTimeline();
            return "";
        } catch(Exception e) {
            Log.v("Exception Twitter query",
                "Exception:"+e.getMessage());
            return "";
        }
    }

    protected void onPostExecute(String result) {
        try {
            myAdapter.notifyDataSetChanged();
        } catch(Exception e) {
            Log.v("Exception twitter query",
                "Exception:"+e.getMessage());
        }
    }
}
}
```



复 杂健壮的Android应用程序经常需要使用一些数据。开发者可以根据情况的不同使用下列数据存储方法：

- 对于轻量级使用，例如保存应用程序设置和用户界面状态，使用shared preferences；
- 对于较复杂的应用，如存储应用程序记录，使用内置的SQLite数据库；
- 使用标准的Java平面文件存储方法：InputStream和OutputStream方法。

以上这些内容都将在本章中讨论。此外，我们还将讨论用于应用程序之间共享数据的Android组件，即内容提供者。需要注意的是，Android系统提供的另一种成对使用的基本数据存储方法，onSaveInstanceState()和onRestoreInstanceState()，已经在第2章中讨论过。我们需要根据实际情况选择使用最佳的方法，在后面的每个案例中都会对此作出讨论。

9.1 shared preferences

SharedPreferences是一种程序接口，它可以使应用程序快速、有效地以“名-值”对保存数据，和Bundle类似。信息存储在Android设备的一个XML文件中。举例来说，如果应用程序com.cookbook.datastorage创建一个shared preference，Android系统将在/data/data/com.cookbook.datastorage/shared_prefs目录下创建一个新的XML文件。shared preferences通常用于保存应用程序设置信息，如用户设置、主题和其他通用应用程序属性等。它也可以用于保存用户名、密码、“自动登录”标志、“记住用户”标志等登录信息。shared preferences数据可被创建它的应用程序的所有组件访问。

9.1.1 秘诀 73：创建和检索 shared preferences

activity相关的shared preferences可以使用getPreferences()方法访问，它指定了默认的preferences文件操作模式。相反，如果需要使用多个preferences文件，每个文件都可以使用getSharedPreferences()方法指定。如果shared preferences XML文件已经存在于数据目录中就打开它，否则就创建该文件。对preferences的不同类型访问权限由操作模式决定，具体如下：

- MODE_PRIVATE——只有调用程序具有访问该XML文件的权限；
- MODE_WORLD_READABLE——所有程序都可以读取该XML文件；

□ `MODE_WORLD_WRITEABLE`——所有程序都可以写入该XML文件。

取回SharedPreferences对象后, 需要使用Editor对象的put()方法将名-值对写入到XML文件中。目前支持的基本数据类型有5种: int、long、float、String和boolean。以下程序代码展示了如何创建和存储shared preferences数据:

```
SharedPreferences prefs = getSharedPreferences("myDataStorage",
                                                MODE_PRIVATE);

Editor mEditor = prefs.edit();
mEditor.putString("username", "datastorageuser1");
mEditor.putString("password", "password1234");
mEditor.commit();
```

下面的代码展示了如何取回 shared preferences数据:

```
SharedPreferences prefs = getSharedPreferences("myDataStorage",
                                                MODE_PRIVATE);

String username = prefs.getString("username", "");
String password = prefs.getString("password", "");
```

9.1.2 秘诀 74: 使用 preferences 框架

Android提供了一个标准框架, 使得设定的preferences可以被所有的应用程序使用。该框架使用分类偏好设置和屏幕将相关设置分组。PreferenceCategory用于声明将一组preferences归到一类, 而PreferenceScreen则将一组preferences显示到新屏幕上。

本秘诀使用的preferences在清单9-1所示的XML文件中定义, PreferenceScreen是根元素, 包含有两个EditTextPreference元素用于存储username和password。其他可选元素包括CheckBoxPreference、RingtonePreference和DialogPreference。Android系统会生成一个UI界面来操作这些preferences, 参见图9-1。这些preferences存储在shared preferences中, 这就意味着它们可以通过调用getPreferences()方法取回。

清单9-1 res/xml/preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <EditTextPreference android:title="User Name"
        android:key="username"
        android:summary="Please provide user
name"></EditTextPreference>
    <EditTextPreference android:title="Password"
        android:password="true"
        android:key="password"
        android:summary="Please enter your
password"></EditTextPreference>
</PreferenceScreen>
```

其后, 程序中的一个activity扩展了PreferenceActivity类, 并调用addPreferencesFromResource()方法将这些 preferences 导入到该activity中, 如清单9-2所示。

清单9-2 src/com/cookbook/datastorage/MyPreferences.java

```

package com.cookbook.datastorage;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class MyPreferences extends PreferenceActivity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        addPreferencesFromResource(R.xml.preferences);
    }
}

```

主 activity 只需在需要时 (例如, 按下 Menu 按钮时) 打开 PreferenceActivity。清单9-3展示了一个简单例子, 在 activity 开始时就显示 preferences。

清单9-3 src/com/cookbook/datastorage/DataStorage.java

```

package com.cookbook.datastorage;

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

public class DataStorage extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Intent i = new Intent(this, MyPreferences.class);
        startActivity(i);
    }
}

```

AndroidManifest XML 文件必须包含所有的 activity, 包括新建的 PreferenceActivity, 如清单9-4所示。

清单9-4 AndroidManifest.xml

```

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

```

```

<activity android:name=".DataStorage"
    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=".MyPreferences" />
</application>
<uses-sdk android:minSdkVersion="7" />
</manifest>

```

以上代码创建的 preferences 显示界面如图9-1所示。

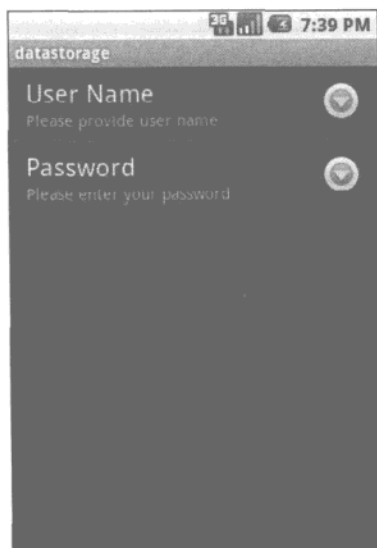


图9-1 Android系统根据 preferences XML文件生成的preferences用户界面

9.1.3 秘诀 75: 基于 Stored Data 改变用户界面

我们可以扩展前面秘诀中提到的DataStorage activity, 在载入时检查 shared preferences, 并对一些行为做出相应的提醒。在本秘诀中, 如果username和password 已经被保存在SharedPreferences文件中, 那么就会显示一个登录页面。成功登录后, 会跳过该activity继续显示。如果该文件中没有登录信息, 那么就显示该activity。

main.xml 页面布局文件可修改为如清单9-5所示的登录页面。该页面使用两个EditText对象来输入用户名和密码, 这部分内容已经在第4章中介绍过了。

清单9-5 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="username"
    />
    <EditText
        android:id="@+id/userText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="password"
    />
    <EditText
        android:id="@+id/passwordText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:password="true"
    />
    <Button
        android:id="@+id/loginButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="login"
        android:textSize="20dp"
    />
</LinearLayout>
```

我们修改了清单9-6所示的主activity的DataStorage，使之首先从SharedPreferences实例中读取username和password数据。如果这些数据未设置，程序将直接打开MyPreferences activity（见清单9-2），设置preferences。如果这些数据已设置，那么应用程序会显示图9-2中所示的登录页面，该页面在main.xml中定义。登录按钮的onClickListener校验登录信息是否和SharedPreferences文件中记录的username和password一致。登录成功的话就会继续运行程序，在本例中即打开MyPreferences activity。任何登录尝试都会显示一个弹出提示信息告知用户登录成功还是失败。

清单9-6 src/com/cookbook/datastorage/DataStorage.java

```

package com.cookbook.datastorage;

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

public class DataStorage extends Activity {
    SharedPreferences myprefs;
    EditText userET, passwordET;
    Button loginBT;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        myprefs = PreferenceManager.getDefaultSharedPreferences(this);
        final String username = myprefs.getString("username", null);
        final String password = myprefs.getString("password", null);
        if (username != null && password != null){
            setContentView(R.layout.main);
            userET = (EditText)findViewById(R.id.userText);
            passwordET = (EditText)findViewById(R.id.passwordText);
            loginBT = (Button)findViewById(R.id.loginButton);
            loginBT.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    try {
                        if(username.equals(userET.getText().toString())
                            && password.equals(
                                passwordET.getText().toString())) {
                            Toast.makeText(DataStorage.this,
                                "login passed!!",
                                Toast.LENGTH_SHORT).show();
                            Intent i = new Intent(DataStorage.this,
                                myPreferences.class);
                            startActivity(i);
                        } else {
                            Toast.makeText(DataStorage.this,
                                "login failed!!",
                                Toast.LENGTH_SHORT).show();
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            });
        }
    }
}

```

```

    }
    }
    });
} else {
    Intent i = new Intent(this, MyPreferences.class);
    startActivity(i);
}
}
}

```

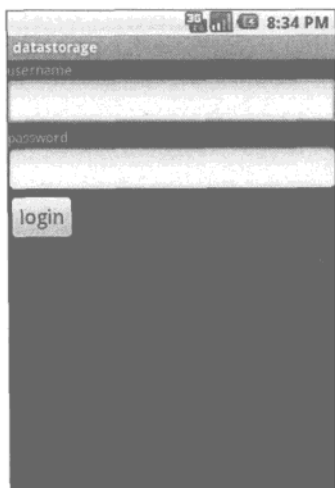


图9-2 清单9-5表示的登录屏幕

9.1.4 秘诀 76：添加最终用户许可协议

正如第1章中讨论的，在用户第一次安装和运行应用程序的时候有必要显示最终用户许可协议（EULA）。如果用户不接受该协议，系统就不会运行已下载的程序；如果用户接受了协议，该最终用户协议就不会再次显示。

清单9-7中的Eula类中实现了EULA功能，并可在遵守Apache许可证的前提下公开使用。该类使用了SharedPreferences，通过一个布尔值PREFERENCE_EULA_ACCEPTED确定EULA是已经接受还是未被接受。

清单9-7 src/com/cookbook/eula_example/Eula.java

```

/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.

```

```

* You may obtain a copy of the License at
*
*      http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/

package com.cookbook.eula_example;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.content.SharedPreferences;

import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.Closeable;

/**
 * Displays an EULA ("End User License Agreement") that the user has to accept
 * before
 * using the application.
 */
class Eula {
    private static final String ASSET_EULA = "EULA";
    private static final String PREFERENCE_EULA_ACCEPTED = "eula.accepted";
    private static final String PREFERENCES_EULA = "eula";

    /**
     * callback to let the activity know when the user accepts the EULA.
     */
    static interface OnEulaAgreedTo {
        void onEulaAgreedTo();
    }

    /**
     * Displays the EULA if necessary.
     */
    static boolean show(final Activity activity) {

        final SharedPreferences preferences =
            activity.getSharedPreferences(
                PREFERENCES_EULA, Activity.MODE_PRIVATE);

        //to test:

```

```
// preferences.edit()
// .putBoolean(PREFERENCE_EULA_ACCEPTED, false).commit();
if (!preferences.getBoolean(PREFERENCE_EULA_ACCEPTED, false)) {
    final AlertDialog.Builder builder =
        new AlertDialog.Builder(activity);
    builder.setTitle(R.string.eula_title);
    builder.setCancelable(true);
    builder.setPositiveButton(R.string.eula_accept,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                accept(preferences);
                if (activity instanceof OnEulaAgreedTo) {
                    ((OnEulaAgreedTo) activity).onEulaAgreedTo();
                }
            }
        });
    builder.setNegativeButton(R.string.eula_refuse,
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int which) {
                refuse(activity);
            }
        });
    builder.setOnCancelListener(
        new DialogInterface.OnCancelListener() {
            public void onCancel(DialogInterface dialog) {
                refuse(activity);
            }
        });
    builder.setMessage(readEula(activity));
    builder.create().show();
    return false;
}
return true;
}

private static void accept(SharedPreferences preferences) {
    preferences.edit().putBoolean(PREFERENCE_EULA_ACCEPTED,
        true).commit();
}

private static void refuse(Activity activity) {
    activity.finish();
}

private static CharSequence readEula(Activity activity) {
    BufferedReader in = null;
    try {
        in = new BufferedReader(new
            InputStreamReader(activity.getAssets().open(ASSET_EULA)));
    }
```



```

        String line;
        StringBuilder buffer = new StringBuilder();
        while ((line = in.readLine()) != null)
            buffer.append(line).append('\n');
        return buffer;
    } catch (IOException e) {
        return "";
    } finally {
        closeStream(in);
    }
}

/**
 * Closes the specified stream.
 */
private static void closeStream(Closeable stream) {
    if (stream != null) {
        try {
            stream.close();
        } catch (IOException e) {
            // Ignore
        }
    }
}
}
}

```

该Eula类需要按照下面的要求自定义。

(1) 最终用户许可协议的实际文本需要存放在一个名为 EULA 的文本文件中（在清单9-7中，通过ASSET_EULA变量定义），该文件应该放置在 Android 工程的assets/ 目录下。该文件通过Eula类的readEula()方法载入。

(2) “接受协议”对话框中的一些字符串需要定义。这些字符串从一个字符串资源文件中获得，如清单9-8中的范例所示。

清单9-8 res/values/strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Welcome to MyApp</string>
    <string name="app_name">MyApp</string>
    <string name="eula_title">License Agreement</string>
    <string name="eula_accept">Accept</string>
    <string name="eula_refuse">Don't Accept</string>
</resources>

```

这样，任何应用程序都可以自动拥有 EULA功能，你所要做的仅仅是将下面一行代码添加到应用程序主activity的onCreate()方法中：

```
Eula.show(this);
```

9.2 SQLite 数据库

对于更为复杂的数据结构，使用数据库可以比平面文件或shared preferences更快更灵活地访问数据。Android 提供了一个内置数据库，名为SQLite，该数据库可使用SQL命令，是一种完善的关系型数据库。使用SQLite的每个应用程序都有自己的数据库实例，默认情况下只能被自己访问。数据库存放在Android 设备的 /data/data/<package_name>/databases文件夹下。可以使用内容提供器在应用程序间共享数据库信息。使用SQLite数据库可分为如下步骤：

- (1) 创建数据库；
- (2) 打开数据库；
- (3) 创建数据库表；
- (4) 创建数据集的插入接口；
- (5) 创建数据集的查询接口；
- (6) 关闭数据库。

下面的秘诀提供了一个完成以上步骤的通用方法。

9.2.1 秘诀 77：创建一个独立的数据库包

良好的类模块结构对于结构复杂的Andorid工程很重要。此处，database类被放置在它自己的com.cookbook.data 包中，因此很容易重用。该包中包含了三个类：MyDB、MyDBhelper和Constants。

MyDB类的代码如清单9-9所示。它包含了一个SQLiteDatabase实例和一个MyDBhelper类(见下文描述)，后者包含了如下方法。

- ❑ MyDB()——初始化MyDBhelper实例（构造函数）。
- ❑ open()——初始化使用MyDBhelper的SQLiteDatabase实例。这样将打开一个可写的数据库连接。如果SQLite抛出异常，将返回只读数据库作为替代。
- ❑ close()——关闭数据库连接。
- ❑ insertdiary()——保存diary条目到数据库中，格式为ContentValues实例中的名-值对，然后将数据传递到SQLite数据库实例，并作插入操作。
- ❑ getdiaries()——从数据库中读取diary条目，将其保存在Cursor类中，然后从该方法中返回给调用对象。

清单9-9 src/com/cookbook/data/MyDB.java

```
package com.cookbook.data;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.util.Log;
```

```

public class MyDB {
    private SQLiteDatabase db;
    private final Context context;
    private final MyDBHelper dbHelper;
    public MyDB(Context c){
        context = c;
        dbHelper = new MyDBHelper(context, Constants.DATABASE_NAME, null,
                                   Constants.DATABASE_VERSION);
    }
    public void close()
    {
        db.close();
    }
    public void open() throws SQLiteException
    {
        try {
            db = dbHelper.getWritableDatabase();
        } catch(SQLiteException ex) {
            Log.v("Open database exception caught", ex.getMessage());
            db = dbHelper.getReadableDatabase();
        }
    }
    public long insertdiary(String title, String content)
    {
        try{
            ContentValues newTaskValue = new ContentValues();
            newTaskValue.put(Constants.TITLE_NAME, title);
            newTaskValue.put(Constants.CONTENT_NAME, content);
            newTaskValue.put(Constants.DATE_NAME,
                             java.lang.System.currentTimeMillis());
            return db.insert(Constants.TABLE_NAME, null, newTaskValue);
        } catch(SQLiteException ex) {
            Log.v("Insert into database exception caught",
                  ex.getMessage());
            return -1;
        }
    }
    public Cursor getdiaries()
    {
        Cursor c = db.query(Constants.TABLE_NAME, null, null,
                             null, null, null, null);
        return c;
    }
}

```

如清单9-10所示，MyDBHelper类扩展了SQLiteOpenHelper。SQLiteOpenHelper框架提供了方法管理数据库创建和升级。数据库在该类的构造函数MyDBHelper()中初始化。这就需要确

定环境配置和数据库名,用于在/data/data/com.cookbook.datastorage/databases目录下创建数据库文件,并需要确定数据库模式的版本来决定是调用onCreate()方法还是onUpgrade()方法。

数据库表可以在onCreate()方法中使用自定义的SQL命令添加,例如:

```
create table MyTable (key_id integer primary key autoincrement,
                      title text not null, content text not null,
                      recorddate long);
```

一旦数据库需要升级(例如用户下载了应用程序的新版本),由于数据库版本改变了,程序将会调用onUpgrade()方法。该方法可用于改变或删除所需的数据库表,将它们更新到新模式。

清单9-10 src/com/cookbook/data/MyDBhelper.java

```
package com.cookbook.data;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.util.Log;

public class MyDBhelper extends SQLiteOpenHelper{
    private static final String CREATE_TABLE="create table "+
        Constants.TABLE_NAME+" ("+
        Constants.KEY_ID+" integer primary key autoincrement, "+
        Constants.TITLE_NAME+" text not null, "+
        Constants.CONTENT_NAME+" text not null, "+
        Constants.DATE_NAME+" long);";
    public MyDBhelper(Context context, String name, CursorFactory factory,
                      int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.v("MyDBhelper onCreate","Creating all the tables");
        try {
            db.execSQL(CREATE_TABLE);
        } catch (SQLiteException ex) {
            Log.v("Create table exception", ex.getMessage());
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
                          int newVersion) {
        Log.w("TaskDBAdapter", "Upgrading from version "+oldVersion
            +" to "+newVersion
            +", which will destroy all old data");
    }
}
```

```

        db.execSQL("drop table if exists "+Constants.TABLE_NAME);
        onCreate(db);
    }
}

```

com.cookbook.data 包中的第三个文件是 Constants 类，如清单 9-11 所示。该类用于保存所有的字符串常量，因为 MyDB 和 MyDBhelper 需要使用这些字符串常量。

清单 9-11 src/com/cookbook/data/Constants.java

```

package com.cookbook.data;

public class Constants {
    public static final String DATABASE_NAME="datastorage";
    public static final int DATABASE_VERSION=1;
    public static final String TABLE_NAME="diaries";
    public static final String TITLE_NAME="title";
    public static final String CONTENT_NAME="content";
    public static final String DATE_NAME="recorddate";
    public static final String KEY_ID="_id";
}

```

9.2.2 秘诀 78：使用独立的数据库包

该秘诀演示了 SQLite 数据存储如何使用前面秘诀中的数据库包。它还使用了秘诀 75 中的登录屏幕，创建并列出了个人日记 (personal diary) 条目。首先是定义一个用于创建日记条目的 XML 布局文件 diary.xml，如清单 9-12 所示，该界面显示的结果见图 9-3。

清单 9-12 res/layout/diary.xml

```

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

```



```
        android:layout_height="wrap_content"
        android:text="Content"
    />
    <EditText
        android:id="@+id/diarycontentText"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
    />
    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="submit"
        android:textSize="20dp"
    />
</LinearLayout>
```

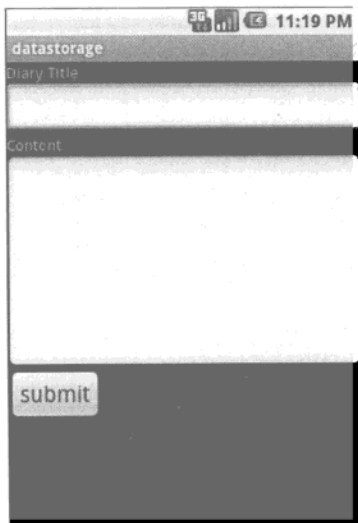


图9-3 日记条目创建界面

主activity为Diary.java, 代码见清单9-13。此处需要导入com.cookbook.data包, 声明、初始化并打开MyDB对象供程序使用。在此还将显示diary.xml描述的屏幕布局, 并处理submit按钮按下事件, 将数据存储到数据库中。

清单9-13 src/com/cookbook/datastorage/Diary.java

```
package com.cookbook.datastorage;

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;

import com.cookbook.data.MyDB;
public class Diary extends Activity {
    EditText titleET, contentET;
    Button submitBT;
    MyDB dba;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.diary);
        dba = new MyDB(this);
        dba.open();
        titleET = (EditText)findViewById(R.id.diarydescriptionText);
        contentET = (EditText)findViewById(R.id.diarycontentText);
        submitBT = (Button)findViewById(R.id.submitButton);
        submitBT.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try {
                    saveItToDB();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    public void saveItToDB() {
        dba.insertdiary(titleET.getText().toString(),
            contentET.getText().toString());

        dba.close();
        titleET.setText("");
        contentET.setText("");
        Intent i = new Intent(Diary.this, DisplayDiaries.class);
        startActivity(i);
    }
}

```

DataStorage.java类的代码和清单9-6所示的代码一致，只是要将登录成功时打开MyPreferences.class改为打开Diary.class:

```

Toast.makeText(DataStorage.this, "login passed!!",
    Toast.LENGTH_SHORT).show();

```

```
Intent i = new Intent(DataStorage.this, Diary.class);
startActivity(i);
```

最后，一定要记住更新 AndroidManifest XML 文件，加入新的 activity，如清单9-14所示。

清单9-14 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.datastorage"
    android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".DataStorage"
            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=".MyPreferences" />
        <activity android:name=".Diary"/>
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>
```

至此，我们在程序中加入了一个独立的数据库，条目列表的布局界面将在下一个秘诀中讨论，最后完成整个日记程序。

9.2.3 秘诀 79：创建个人日记

该秘诀利用ListView对象，以显示从SQLite数据库表中取回的多行记录。程序在一个纵向滚动列表中显示这些工程。当后台数据发生改变时，ListView控件通过一个数据适配器通知视图。我们需要创建两个XML文件：diaries.xml文件用于加入清单9-15中所示的ListView，diaryrow.xml用于在ListView控件中加入清单9-16中所示的内容行。

清单9-15 res/layout/diaries.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
        android:layout_width="fill_parent" android:dividerHeight="1px"
        android:layout_height="fill_parent"
        android:id="list">
    </ListView>
</LinearLayout>
```


清单9-16 res/layout/diaryrow.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:orientation="vertical"
    android:layout_alignLeft="@+id/name" android:layout_below="@+id/name"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="12dip">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/name"
        android:layout_marginRight="4dp" android:text="Diary Title "
        android:textStyle="bold" android:textSize="16dip" />
    <TextView android:id="@+id/datetime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Date Recorded"
        android:textSize="14dip" />
</RelativeLayout>

```

程序中的 DisplayDiaries.java 继承自 ListActivity，它可以显示一个 ListView。在该类中定义了两个内部类：MyDiary 类是一个数据类，用于保存日记条目的内容（标题、内容和日期）；DiaryAdapter 类是一个 BaseAdapter 类，用于从数据库取回数据（使用 getData() 方法）。以下是从 BaseAdapter 类派生并被 ListView 调用的方法：

- getCount() —— 返回适配器中的条目数量；
- getItem() —— 返回具体的条目；
- getItemID() —— 返回条目的 ID（在本例中，没有条目 ID）；
- getView() —— 返回每个条目的视图。

注意，ListView 调用 getView() 方法为每个条目绘制视图。为了改进渲染 UI 的性能，由 getView() 返回的视图应该尽可能回收重用。我们可以通过创建 ViewHolder 类来保留这些视图。

调用 getView() 时，显示给用户的当前视图也被传递进来，此时该视图会被保存到 ViewHolder 中并附加标签。后面如果要通过 getView() 调用同一个视图，该标签会辨认出该视图已经保存在 ViewHolder 中。在这个例子中，可以在已有的视图上改变内容，而不用创建新视图。

程序的主 activity 代码见清单 9-17。ListView 中日记条目的显示结果见图 9-4。

清单9-17 src/com/cookbook/datastorage/DisplayDiaries.java

```

package com.cookbook.datastorage;

import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Date;

import android.app.ListActivity;
import android.content.Context;
import android.database.Cursor;

```

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;

import com.cookbook.data.Constants;
import com.cookbook.data.MyDB;

public class DisplayDiaries extends ListActivity {
    MyDB dba;
    DiaryAdapter myAdapter;
    private class MyDiary{
        public MyDiary(String t, String c, String r){
            title=t;
            content=c;
            recorddate=r;
        }
        public String title;
        public String content;
        public String recorddate;
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        dba = new MyDB(this);
        dba.open();
        setContentView(R.layout.diaries);

        super.onCreate(savedInstanceState);
        myAdapter = new DiaryAdapter(this);
        this.setAdapter(myAdapter);
    }

    private class DiaryAdapter extends BaseAdapter {
        private LayoutInflater mInflater;
        private ArrayList<MyDiary> diaries;
        public DiaryAdapter(Context context) {
            mInflater = LayoutInflater.from(context);
            diaries = new ArrayList<MyDiary>();
            getdata();
        }
        public void getdata(){
            Cursor c = dba.getdiaries();
            startManagingCursor(c);
            if(c.moveToFirst()){
                do{
                    String title =
                        c.getString(c.getColumnIndex(Constants.TITLE_NAME));
```

```
String content =
    c.getString(c.getColumnIndex(Constants.CONTENT_NAME));
DateFormat dateFormat =
    DateFormat.getDateInstance();
String datedata = dateFormat.format(new
    Date(c.getLong(c.getColumnIndex(
        Constants.DATE_NAME))).getTime());
MyDiary temp = new MyDiary(title,content,datedata);
diaries.add(temp);
} while(c.moveToNext());
}
}

@Override
public int getCount() {return diaries.size();}
public MyDiary getItem(int i) {return diaries.get(i);}
public long getItemId(int i) {return i;}
public View getView(int arg0, View arg1, ViewGroup arg2) {
    final ViewHolder holder;
    View v = arg1;
    if ((v == null) || (v.getTag() == null)) {
        v = inflater.inflate(R.layout.diaryrow, null);
        holder = new ViewHolder();
        holder.mTitle = (TextView)v.findViewById(R.id.name);
        holder.mDate = (TextView)v.findViewById(R.id.datetext);
        v.setTag(holder);
    } else {
        holder = (ViewHolder) v.getTag();
    }

    holder.mdiary = getItem(arg0);
    holder.mTitle.setText(holder.mdiary.title);
    holder.mDate.setText(holder.mdiary.recorddate);

    v.setTag(holder);

    return v;
}

public class ViewHolder {
    MyDiary mdiary;
    TextView mTitle;
    TextView mDate;
}
}
```

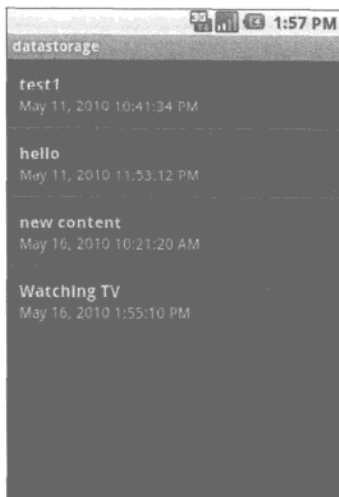


图9-4 显示日记条目的 ListView

9.3 内容提供者

每个应用程序都有自己的沙箱，不能访问其他应用程序的数据。如果需要访问自身沙箱没有提供的功能，应用程序必须在安装前显式地声明该许可。Android系统提供了一个名为ContentProvider的接口充当应用程序之间的桥梁，使之能够分享和改变对方的数据。内容提供者使应用层和数据层可以清楚分离。这需要在AndroidManifest XML文件中设置权限许可，并且可以通过一个简单的URI模型访问。

下面是一些在Android系统中使用了内容提供器的原生数据库。

- Browser——读取或修改书签，浏览器历史，或网页搜索记录。
- CallLog——查看或更新通话记录。
- Contacts——检索、修改或存储个人通讯录。通讯录信息存储在ContactsContract对象的一个三层数据表模型中。
 - ContactsContract.Data——包含各类个人数据。包括预定义的通用数据集，如电话号码、电子邮件地址，但是该表的格式与具体程序有关。
 - ContactsContract.RawContacts——包含一个数据对象集合，和单个账号或个人相联系。
 - ContactsContract.Contacts——包含一个或多个RawContacts的聚集，有可能是描述同一个人。
- LiveFolders——一个特殊的文件夹，其中的内容由ContentProvider提供。
- MediaStore——访问音频、视频和图像。
- Setting——查看和检索蓝牙设置、铃声和其他设备偏好设置。

- SearchRecentSuggestions——可配置为使用一个搜索建议提供者（search suggestions provider）。
- SyncStateContract——是通过数据数组账号连接数据的ContentProvider协议。使用标准方法存储数据的提供者可以使用它。
- UserDictionary——支持在输入法中由用户定义词组，用于在输入文字时预测输入。应用程序和输入法可以向字典中添加词组。词组可以和使用频率以及地点信息相关联。

应用程序需要通过ContentResolver实例访问内容提供者，以查询、插入、删除和更新内容提供器的数据，如下面的例子所示。

```
ContentResolver crInstance = getContentResolver(); //get a content Resolver instance
crInstance.query(People.CONTENT_URI, null, null, null, null); //query contacts
ContentValues new_Values= new ContentValues();
crInstance.insert(People.CONTENT_URI, new_Values); // insert new values
crInstance.delete(People.CONTENT_URI, null, null); //delete all contacts

ContentValues update_Values= new ContentValues();
crInstance.update(People.CONTENT_URI, update_Values, null, null); //update values
```

每个内容提供者都有一个统一资源标识符（URI），用来注册资源和标识许可权限。URI必须在所有的提供者中唯一，建议采用如下的通用格式。

```
content://<package name>.provider.<custom ContentProvider name>/<DataPath>
```

为简化该URI，也可以写为content://com.cookbook.datastorage/diaries，在下个秘诀中将会用到该URI。ContentProvider中使用UriMatcher确保合适的URI被传递。

秘诀 80：创建自定义内容提供者

在了解了如何使用内容提供者之后，现在我们接着讨论如何在前面秘诀的日记工程中使用它。本秘诀将演示如何将日记工程提供给其他选定的应用程序使用。自定义内容提供者扩展了Android的ContentProvider类，我们可根据需要重载它包含的6个方法中的数个：

- query()——允许第三方应用程序检索内容；
- insert()——允许第三方应用程序插入内容；
- update()——允许第三方应用程序更新内容；
- delete()——允许第三方应用程序删除内容；
- getType()——允许第三方应用程序读取系统支持的每个URI结构体；
- onCreate()——创建一个数据库实例以帮助获取内容。

例如，如果其他应用程序获得的许可是只能从提供者中读取内容，那么只需要重载onCreate()和query()方法。

自定义的ContentProvider如清单9-18所示，它在com.cookbook.datastorage包的UriMatcher类中添加了一个URI，其数据库表名为diaries。onCreate()方法生成了一个MyDB对象，程序代码见清单9-9。该类负责数据库的访问。query()方法从数据库中取回所有的日记记录并作为uri参数传递。如果需要选择某条具体的记录，还需要使用该方法的其他参数。

清单9-18 src/com/cookbook/datastorage/DiaryContentProvider.java

```
package com.cookbook.datastorage;

import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;

import com.cookbook.data.Constants;
import com.cookbook.data.MyDB;

public class DiaryContentProvider extends ContentProvider {

    private MyDB dba;
    private static final UriMatcher sUriMatcher;
    //the code returned for URI match to components
    private static final int DIARIES=1;
    public static final String AUTHORITY = "com.cookbook.datastorage";
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITY, Constants.TABLE_NAME,
                           DIARIES);
    }
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        return 0;
    }
    public String getType(Uri uri) {return null;}
    public Uri insert(Uri uri, ContentValues values) {return null;}
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {return 0;}

    @Override
    public boolean onCreate() {
        dba = new MyDB(this.getContext());
        dba.open();
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                       String[] selectionArgs, String sortOrder) {
        Cursor c=null;
        switch (sUriMatcher.match(uri)) {
            case DIARIES:
                c = dba.getdiaries();
                break;
        }
    }
}
```

```

        default:
            throw new IllegalArgumentException(
                "Unknown URI " + uri);
    }
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
}
}

```

我们需要在AndroidManifest XML文件中定义该提供者可访问，如清单9-19所示。

清单9-19 AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.datastorage"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".DataStorage"
            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=".MyPreferences" />
        <activity android:name=".Diary" />
        <activity android:name=".DisplayDiaries" />
        <provider android:name="DiaryContentProvider"
            android:authorities="com.cookbook.datastorage" />
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>

```

现在其他应用程序可以使用该内容提供者了。为了测试该内容提供者是否可用，我们创建了一个新的Android工程，名为DataStorageTester。该程序的主activity为DataStorageTester，程序代码见清单9-20。程序创建了一个ContentResult实例从DataStorage内容提供者查询数据。然后返回一个Cursor对象，用于测试的方法解析每个数据条目的第二列，将这些内容连接起来放到一个String对象中，最后通过一个StringBuilder对象显示在屏幕上。

清单9-20 src/com/cookbook/datastorage_tester/DataStorageTester.java

```

package com.cookbook.datastorage_tester;

import android.app.Activity;
import android.content.ContentResolver;
import android.database.Cursor;

```

```

import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

public class DataStorageTester extends Activity {
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.output);
        String myUri = "content://com.cookbook.datastorage/diaries";
        Uri CONTENT_URI = Uri.parse(myUri);
        //get ContentResolver instance
        ContentResolver crInstance = getContentResolver();
        Cursor c = crInstance.query(CONTENT_URI, null, null, null, null);
        startManagingCursor(c);
        StringBuilder sb = new StringBuilder();
        if(c.moveToFirst()){
            do{
                sb.append(c.getString(1)).append("\n");

            }while(c.moveToNext());
        }
        tv.setText(sb.toString());
    }
}

```

在main.xml布局文件中，需要在TextView输出内容添加一个ID，代码见清单9-21。

清单9-21 res/layout/main.xml

```

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

```

运行测试方法会显示日记条目的标题，显示结果见图9-5。



图9-5 另一个日记程序从内容提供者中获得的查询结果

9.4 保存和载入文件

除了前面提到的 Android 专用数据存储方法之外，我们也可以使用标准的 Java 包 `java.io.File`。这种方法支持对平面文件的操作，例如 `FileInputStream`、`FileOutputStream`、`InputStream` 和 `OutputStream`。下面是一个从文件中读取内容和将内容写入文件的范例：

```
FileInputStream fis = openFileInput("myfile.txt");
FileOutputStream fos = openFileOutput("myfile.txt",
    Context.MODE_WORLD_WRITEABLE);
```

另一个例子是将摄像头捕捉的 `bitmap` 图片保存为 `PNG` 格式，代码如下：

```
Bitmap takenPicture;
FileOutputStream out = openFileOutput("mypic.png",
    Context.MODE_WORLD_WRITEABLE);
takenPicture.compress(CompressFormat.PNG, 100, out);
out.flush();
out.close();
```

资源目录下的文件也可以打开，例如，我们可以使用下面的代码打开 `res/raw` 文件夹中的 `myrawfile.txt` 文件：

```
InputStream is = this.getResource()
    .openRawResource(R.raw.myrawfile.txt);
```

基于位置的服务 (LBS: Location-Based Service) 使一些最流行的移动应用程序获得成功。位置可以和许多功能整合, 如网络搜索、拍摄照片、游戏和社交网络。开发人员可以利用现有的定位技术提升他们的应用程序, 使之更具实用性。

本章将首先介绍获取设备位置的方法, 然后是跟踪、地理编码和绘制地图。此外, 还将介绍在地图上叠加标记和视图的秘诀。

10.1 位置服务入门

应用程序需要通过下面的组件访问Android系统的位置服务:

- ❑ `LocationManager`——该类提供了对Android系统位置服务的访问;
- ❑ `LocationListener`——该接口用于接收当位置发生变化时从`LocationManager`发出的通知;
- ❑ `Location`——该类代表某一特定时间的地理位置。

`LocationManager`需要调用Android系统服务`LOCATION_SERVICE`进行初始化。它可以为应用程序提供设备的当前位置、运动状态, 还可以在设备进入或离开某个特定地点时发出提醒。下面是一个初始化的例子:

```
LocationManager locationManager;  
locationManager = (LocationManager)  
    getSystemService(Context.LOCATION_SERVICE);
```

`LocationManager`实例启动后, 需要选择一个位置提供器。一个设备可以使用多种不同的定位技术, 如辅助全球定位系统 (AGPS)、无线网络等, 位置提供器的选择依据主要是定位准确性和电力消耗。我们可以通过使用`android.location.Criteria`中定义的`Criteria`类来筛选, 这使得Android系统能够根据特定的需求选择最优的定位技术。下面是通过`criteria`选择位置提供器的例子:

```
Criteria criteria = new Criteria();  
criteria.setAccuracy(Criteria.ACCURACY_FINE);  
criteria.setPowerRequirement(Criteria.POWER_LOW);  
String locationProvider =  
    locationManager.getBestProvider(criteria, true);
```

我们也可以通过位置管理器的`getProvider()`方法指定程序使用的定位技术。最常见的两个提供者是：基于卫星的全球定位系统（通过`LocationManager.GPS_PROVIDER`指定）和蜂窝发射塔的识别号（通过`LocationManager.NETWORK_PROVIDER`指定）。前者较准确，但后者在天空被遮盖的情况下（如室内）非常有用。

除非另有说明，本章所有的秘诀都会利用以下两个辅助文件。首先是主布局文件，通过一个`TextView`控件显示位置数据，如清单10-1所示。

清单10-1 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
</LinearLayout>
```

接着，允许在`AndroidManifest.xml`文件中的授权程序使用位置信息，如清单10-2所示（每个秘诀中只需要修改包名）。若要更准确地定位，如使用GPS，则需要添加`ACCESS_FINE_LOCATION`权限。其他情况则添加`ACCESS_COARSE_LOCATION`权限。

清单10-2 `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.mylocationpackage"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MyLocation"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="4" />

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

10.1.1 秘诀 81：获取最新位置

由于计算位置需要一定的时间，我们可以通过调用`getLastKnownLocation()`方法获取为指定提供器保存的最新位置。位置信息包括纬度、经度和世界标准时间（UTC）时间戳。有的提供器还会包括高度、速度和方位信息（使用`Location`对象的`getAltitude()`、`getSpeed()`和`getBearing()`方法获取这些信息，并且使用`getExtras()`方法检索卫星信息）。本秘诀将在屏幕上显示经度和纬度，主activity的代码如清单10-3所示。

清单10-3 `src/com/cookbook/lastlocation/MyLocation.java`

```
package com.cookbook.lastlocation;

import android.app.Activity;
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MyLocation extends Activity {
    LocationManager mLocationManager;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv1);
        mLocationManager = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String locationprovider =
            mLocationManager.getBestProvider(criteria,true);
        Location mLocation =
            mLocationManager.getLastKnownLocation(locationprovider);

        tv.setText("Last location lat:" + mLocation.getLatitude()
            + " long:" + mLocation.getLongitude());
    }
}
```

10.1.2 秘诀 82：在位置改变时更新信息

`LocationListener`接口用于在位置发生改变时接收通知。在位置提供器对象完成初始化以

后，我们需要调用位置管理器的`requestLocationUpdates()`方法确定何时通知当前的activity位置发生了变更。可使用以下参数：

- ❑ `provider`——应用程序使用的位置提供器；
- ❑ `minTime`——最小更新时间，以毫秒为单位（但是系统可能会加大这个时间参数，以节省电能）；
- ❑ `minDistance`——最小更新距离，以米为单位；
- ❑ `listener`——接收更新信息的位置侦听器（location listener）。

我们可以重载位置侦听器的`onLocationChanged()`方法，以指定在新位置要进行的操作。清单10-4的代码显示的更新条件为：时间变化5秒并且位置变化超过2米。在真实的应用程序中应该设定更大的变化值以节省电池电量。同时要注意的是，在`onLocationChanged()`方法中不应进行大量的运算处理。相反，应该复制数据并把它传给一个新线程。

清单10-4 `src/com/cookbook/update_location/MyLocation.java`

```
package com.cookbook.update_location;

import android.app.Activity;
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MyLocation extends Activity implements LocationListener {
    LocationManager mLocationManager;
    TextView tv;
    Location mLocation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv1);

        mLocationManager = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String locationprovider =
            mLocationManager.getBestProvider(criteria,true);

        mLocation =
```

```

        mLocationManager.getLastKnownLocation(locationprovider);
        mLocationManager.requestLocationUpdates(
            locationprovider, 5000, 2.0, this);
    }

    @Override
    public void onLocationChanged(Location location) {
        mLocation = location;
        showupdate();
    }
    // these methods are required
    public void onProviderDisabled(String arg0) {}
    public void onProviderEnabled(String provider) {}
    public void onStatusChanged(String a, int b, Bundle c) {}

    public void showupdate(){
        tv.setText("Last location lat:"+mLocation.getLatitude()
            + " long:" + mLocation.getLongitude());
    }
}

```

请注意，不要在activity级别实现LocationListener，可以像下面的代码中那样把它声明为一个单独的内部类。这个技巧可以很容易地应用于后面所有的秘诀中，它提供了一种位置信息的更新机制：

```

        mLocationManager.requestLocationUpdates(
            locationprovider, 5000, 2.0, myLocL);
    }

    private final LocationListener myLocL = new LocationListener(){
        @Override
        public void onLocationChanged(Location location){
            mLocation = location;
            showupdate();
        }

        // these methods are required
        public void onProviderDisabled(String arg0) {}
        public void onProviderEnabled(String provider) {}
        public void onStatusChanged(String a, int b, Bundle c) {}
    };
}

```

10.1.3 秘诀 83：列出所有可用的提供者

本秘诀将列出某个Android设备所有可用的位置提供者。某个示例输出的内容如图10-1所示，根据设备的不同，显示的内容可能会有所不同。其主activity的代码见清单10-5。我们可以使用getProviders(true)方法列出一个可用提供程序的清单，与前面的秘诀相反，在此我们将

LocationListener定义为一个匿名内部类以保留其完整的功能。

清单10-5 src/com/cookbook/show_providers/MyLocation.java

```
package com.cookbook.show_providers;

import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MyLocation extends Activity {
    LocationManager locationManager;
    TextView tv;
    Location mLocation;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv1);
        locationManager = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String locationprovider =
            locationManager.getBestProvider(criteria,true);

        List<String> providers = locationManager.getProviders(true);
        StringBuilder mSB = new StringBuilder("Providers:\n");
        for(int i = 0; i<providers.size(); i++) {
            locationManager.requestLocationUpdates(
                providers.get(i), 5000, 2.0f, new LocationListener(){

                // these methods are required
                public void onLocationChanged(Location location) {}
                public void onProviderDisabled(String arg0) {}
                public void onProviderEnabled(String provider) {}
                public void onStatusChanged(String a, int b, Bundle c) {}

            });
            mSB.append(providers.get(i)).append(": \n");
            mLocation =

```

```

        locationManager.getLastKnownLocation(providers.get(i));
        if(mLocation != null) {
            mSB.append(mLocation.getLatitude()).append(" ");
            mSB.append(mLocation.getLongitude()).append("\n");
        } else {
            mSB.append("Location can not be found");
        }
    }
    tv.setText(mSB.toString());
}
}

```



图10-1 在一个真实的Android设备中显示的所有可用位置提供者以及它们的lastKnownLocation

10.1.4 秘诀 84：将位置解析为地址（反向地理编码）

Geocoder类提供了一个方法，可以将地址解析为经纬度坐标（即地理编码），或是将经纬度坐标转换为地址（即反向地理编码）。反向地理编码可能只生成部分地址，例如城市和邮政编码，取决于位置提供器的信息的详细程度。

本秘诀使用反向地理编码，根据设备的位置获得一个地址，并显示到屏幕上，如图10-2所示。Geocoder实例需要根据上下文信息初始化，如果和系统位置不同的话则还要提供地区信息。在此例中，地区被显式地设置为Locale.ENGLISH，其后通过getFromLocation()方法可获得与所提供的位置相关的地址列表。此处设定返回结果的最大值为1（即最为可能的地址）。

地理编码器返回一个android.location.Address对象列表。对于地址的解析依赖于一个后台服务，其并没有包含在Android的核心框架中。谷歌地图API提供了一个基于客户端的Geocoder

服务。如果目标设备上没有这样的服务，则会返回一个空地址列表。包含地址的字符串列表将分行导入到一个字符串中，并显示在屏幕上。清单10-6是主activity的代码。



图10-2 反向地理编码示例，将经纬度坐标转换为地址

清单10-6 src/com/cookbook/rev_geocoding/MyLocation.java

```
package com.cookbook.rev_geocoding;

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.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class MyLocation extends Activity {
    LocationManager mLocationManager;
    Location mLocation;
    TextView tv;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);
    tv = (TextView) findViewById(R.id.tv1);

    mLocationManager = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);

    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    String locationprovider =
        mLocationManager.getBestProvider(criteria, true);

    mLocation =
        mLocationManager.getLastKnownLocation(locationprovider);

    List<Address> addresses;
    try {
        Geocoder mGC = new Geocoder(this, Locale.ENGLISH);
        addresses = mGC.getFromLocation(mLocation.getLatitude(),
                                         mLocation.getLongitude(), 1);

        if(addresses != null) {
            Address currentAddr = addresses.get(0);
            StringBuilder mSB = new StringBuilder("Address:\n");
            for(int i=0; i<currentAddr.getMaxAddressLineIndex(); i++) {
                mSB.append(currentAddr.getAddressLine(i)).append("\n");
            }

            tv.setText(mSB.toString());
        }
    } catch (IOException e) {
        tv.setText(e.getMessage());
    }
}
```

10.1.5 秘诀 85：将地址解析为位置（地理编码）

本秘诀演示了如何将一个地址解析为经纬度坐标，我们称之为地理编码。除了使用 `getFromLocationName()` 方法代替 `getFromLocation()` 方法以外，该秘诀与前面的秘诀非常相似。根据清单10-7列出的代码，此处需要在字符串 `myAddress` 中定义具体的地址，将其转换为位置，然后显示到屏幕上，如图10-3所示。

清单10-7 src/com/cookbook/geocoding/MyLocation.java

```

package com.cookbook.geocoding;

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.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MyLocation extends Activity {
    LocationManager locationManager;
    Location mLocation;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.tv1);

        locationManager = (LocationManager)
            getSystemService(Context.LOCATION_SERVICE);

        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String locationprovider =
            locationManager.getBestProvider(criteria, true);
        mLocation =
            locationManager.getLastKnownLocation(locationprovider);

        List<Address> addresses;

        String myAddress="Seattle,WA";
        Geocoder gc = new Geocoder(this);
        try {
            addresses = gc.getFromLocationName(myAddress, 1);
            if(addresses != null) {
                Address x = addresses.get(0);
            }
        }
    }
}

```

```
        StringBuilder mSB = new StringBuilder("Address:\n");

        mSB.append("latitude: ").append(x.getLatitude());
        mSB.append("\nlongitude: ").append(x.getLongitude());
        tv.setText(mSB.toString());
    }
} catch(IOException e) {
    tv.setText(e.getMessage());
}
}
}
```



图10-3 地理编码示例，将包含地址的字符串解析为经纬度坐标

10.2 使用谷歌地图

在Android系统中使用谷歌地图的方法有两种：用户可以通过浏览器或应用程序访问谷歌地图API。MapView类封装了谷歌地图API。为了使用MapView，我们需要完成如下的设置工作。

(1) 下载并安装谷歌API软件开发工具包：

- ① 在Eclipse中使用Android SDK和Android虚拟设备管理器下载谷歌API；
- ② 在使用该API的工程中单击鼠标右键，然后选择Properties；
- ③ 选择Android，然后选择Google API，在工程中启用该API。

(2) 为了使用谷歌地图服务，需要获得有效的地图API密钥（见<http://code.google.com/android/add-ons/google-apis/mapkey.html>）。

- ① 使用keytool命令，根据密钥的alias_name生成一个MD5证书指纹：

```
> keytool -list -alias alias_name -keystore my.keystore
> result:(Certificate fingerprint (MD5):
94:1E:43:49:87:73:BB:E6:A6:88:D7:20:F1:8E:B5)
```

② 使用MD5密钥库申请谷歌地图服务，网址为<http://code.google.com/android/maps-api-signup.html>。

③ 申请获得地图API密钥，并在MapView中使用这个密钥。

(3) 在 AndroidManifest.xml 文件中添加 `<uses-library android:name="com.google.android.maps"/>`，告知Android系统应用程序使用了来源于谷歌API的SDK中的com.google.android.maps库。

(4) 将android.permission.INTERNET权限添加到 AndroidManifest.xml文件，使该应用程序获得使用从互联网接收的谷歌地图服务数据的许可。

(5) 在XML布局文件中添加MapView。

具体来说，使用谷歌地图的activity需要下面两个辅助文件。首先，需要在AndroidManifest XML文件中添加一个适当的地图类库和许可权限，如清单10-8所示。

清单10-8 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.using_gmaps"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MyLocation"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <uses-library android:name="com.google.android.maps" />
    </application>
    <uses-sdk android:minSdkVersion="4" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>
```

其次，需要在XML布局文件中正确地声明MapView以显示谷歌地图，如清单10-9所示。也可以在此通过clickable元素声明用户是否可以与地图进行交互，默认值为false。

清单10-9 res/layout/main.xml

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

```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    >
    <TextView
        android:id="@+id/tv1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"
    />
    <com.google.android.maps.MapView
        android:id="@+id/map1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey="0ZDUMMY13442HjX49lCODE44MSsJzfDVlIQ"
    />
</LinearLayout>
```

该文件将在后面的秘诀中使用。

10.2.1 秘诀 86：在应用程序中添加谷歌地图

为了显示谷歌地图，主activity应扩展MapActivity类，如清单10-10所示。还必须指向主布局XML文件中地图的布局ID，此处为map1。注意，程序中还需实现isRouteDisplayed()方法，显示结果如图10-4所示。

清单10-10 src/com/cookbook/using_gmaps/MyLocation.java

```
package com.cookbook.using_gmaps;

import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

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

public class MyLocation extends MapActivity {
    LocationManager locationManager;
    Location mLocation;
    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
setContentView(R.layout.main);
MapView mapView = (MapView) findViewById(R.id.map1);
tv = (TextView) findViewById(R.id.tv1);

mLocationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);

Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setPowerRequirement(Criteria.POWER_LOW);
String locationprovider =
    mLocationManager.getBestProvider(criteria,true);
mLocation =
    mLocationManager.getLastKnownLocation(locationprovider);

tv.setText("Last location lat:" + mLocation.getLatitude()
    + " long:" + mLocation.getLongitude());
}

@Override
protected boolean isRouteDisplayed() {
    // this method is required
    return false;
}
}
```



图10-4 在程序中使用谷歌地图

10.2.2 秘诀 87：在地图上添加标记

ItemizedOverlay类提供了一种在MapView顶层绘制标记和图层的方法。该类用于以列表的形式管理图像等OverlayItem元素（如图像）的集合，并处理绘制、放置、点击、焦点控制和元素布局的优化。我们可创建一个类扩展ItemizedOverlay类，并重载以下方法。

- addOverlay()——添加一个OverlayItem到ArrayList。调用populate()方法读取该工程，并为绘制它作准备。
- createItem()——被populate()方法调用，以取回指定的OverlayItem。
- size()——返回ArrayList中OverlayItem元素的数量。
- onTap()——点击标记时的回调方法。

新创建的类参见清单10-11，显示结果如图10-5。

清单10-11 src/com/cookbook/adding_markers/MyMarkerLayer.java

```
package com.cookbook.adding_markers;

import java.util.ArrayList;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;

import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;

public class MyMarkerLayer extends ItemizedOverlay {

    private ArrayList<OverlayItem> mOverlays =
        new ArrayList<OverlayItem>();

    public MyMarkerLayer(Drawable defaultMarker) {
        super(boundCenterBottom(defaultMarker));
        populate();
    }

    public void addOverlayItem(OverlayItem overlay) {
        mOverlays.add(overlay);
        populate();
    }

    @Override
    protected OverlayItem createItem(int i) {
        return mOverlays.get(i);
    }

    @Override
    public int size() {
        return mOverlays.size();
    }
}
```




```

@Override
protected boolean onTap(int index) {
    AlertDialog.Builder dialog =
        new AlertDialog.Builder(MyLocation.mContext);
    dialog.setTitle(mOverlays.get(index).getTitle());
    dialog.setMessage(mOverlays.get(index).getSnippet());
    dialog.setPositiveButton("Ok",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                dialog.cancel();
            }
        });
    dialog.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
                dialog.cancel();
            }
        });
    dialog.show();
    return super.onTap(index);
}
}

```



图10-5 在地图上添加一个可点击的标记

以下是对清单10-11中粗体表示的MyMarkerLayer类的几点说明。

- 声明一个OverlayItem容器，并命名为mOverlays，以保存所有传递到图层的工程。
- 在绘制任何图层元素之前，必须定义一个绑定（binding point），所有图层工程都在此叠

加到地图之上。我们可以在类构造函数中添加boundCenterBottom, 指定地图底部的中心为该点。

- ❑ 重载所需的方法: addOverlay()、createItem()、size()和onTap()。在此, onTap()方法会在单击工程被点击时显示一个对话框。
- ❑ 在MyMarkerLayer构造函数末尾和addOverlay()末尾添加populate()方法, 告知MyMarkerLayer类准备所有的OverlayItem元素, 并在地图上绘制它们。

现在, 我们可以将ItemizedOverlay添加到前面秘诀中创建的MapActivity了。代码如下清单10-12所示, 该activity的功能如下。

(1) 使用getOverlays()方法从MapView检索现有的地图图层工程。在该方法的结尾部分添加标记层到这个容器中。

(2) 定义一个MyMarkerLayer实例处理图层工程。

(3) 检索地址对应的经纬度 (以度为计量单位)。该步骤使用GeoPoint类定义相关的点。由于GeoPoint中输入值的单位为microdegree, 因此经度和纬度值都需要乘以一百万 (1E6)。

(4) 使用地图控制器绘制指向GeoPoint的动画以及缩放视图。此外, 使用setBuiltInZoomControls()方法打开用户控制缩放功能。

(5) 定义OverlayItem, 显示GeoPoint相关消息。

(6) 使用addOverlayItem()方法将工程添加至MyMarkerLayer。然后把刚定义的MyMarkerLayer添加到步骤1中取回的图层列表中。

清单10-12 src/com/cookbook/adding_markers/MyLocation.java

```
package com.cookbook.adding_markers;

import java.io.IOException;
import java.util.List;

import android.content.Context;
import android.graphics.drawable.Drawable;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.widget.TextView;
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;

public class MyLocation extends MapActivity {
    TextView tv;
    List<Overlay> mapOverlays;
    MyMarkerLayer markerlayer;
    private MapController mc;
    public static Context mContext;
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mContext = this;
    setContentView(R.layout.main);
    MapView mapView = (MapView) findViewById(R.id.map1);
    tv = (TextView) findViewById(R.id.tv1);

    mapOverlays = mapView.getOverlays();
    Drawable drawable =
        this.getResources().getDrawable(R.drawable.icon);
    markerlayer = new MyMarkerLayer(drawable);

    List<Address> addresses;
    String myAddress="1600 Amphitheatre Parkway, Mountain View, CA";

    int geolat = 0;
    int geolon = 0;

    Geocoder gc = new Geocoder(this);
    try {
        addresses = gc.getFromLocationName(myAddress, 1);
        if(addresses != null) {
            Address x = addresses.get(0);

            geolat = (int)(x.getLatitude()*1E6);
            geolon = (int)(x.getLongitude()*1E6);
        }
    } catch(IOException e) {
        tv.setText(e.getMessage());
    }

    mapView.setBuiltInZoomControls(true);
    GeoPoint point = new GeoPoint(geolat,geolon);
    mc = mapView.getController();
    mc.animateTo(point);
    mc.setZoom(3);

    OverlayItem overlayitem =
        new OverlayItem(point, "Google Campus", "I am at Google");
    markerlayer.addOverlayItem(overlayitem);
    mapOverlays.add(markerlayer);
}

@Override
protected boolean isRouteDisplayed() { return false; }
}

```

10.2.3 秘诀 88：在地图上添加视图

开发人员可以在MapView中添加任何的View或ViewGroup。本秘诀演示了如何在地图上增加两个简单元素：TextView控件和Button控件。点击按钮，TextView中显示的文本将发生改变。调用addView()方法在MapView中加入这两个视图，并使用LayoutParams参数定义界面设置。在此，元素的位置通过(x, y)屏幕坐标指定，但开发者也可以将一个GeoPoint对象提供给LayoutParams来代替。清单10-13是主activity的代码，需要用到上个秘诀中定义的MyMarkerLayer类（见清单10-11，修改了第一行以显示正确的包名）。图10-6是地图视图的显示结果。

清单10-13 src/com/cookbook/mylocation/MyLocation.java

```
package com.cookbook.mylocation;

import java.io.IOException;
import java.util.List;

import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.graphics.drawable.Drawable;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
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;

public class MyLocation extends MapActivity {
    TextView tv;
    List<Overlay> mapOverlays;
    MyMarkerLayer markerlayer;
    private MapController mc;
    MapView.LayoutParams mScreenLayoutParams;
    public static Context mContext;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mContext = this;
        setContentView(R.layout.main);

        MapView mapView = (MapView) findViewById(R.id.map1);
```



```

mc = mapView.getController();
tv = (TextView) findViewById(R.id.tv1);
mapOverlays = mapView.getOverlays();
Drawable drawable =
    this.getResources().getDrawable(R.drawable.icon);
markerlayer = new MyMarkerLayer(drawable);

List<Address> addresses;
String myAddress="1600 Amphitheatre Parkway, Mountain View, CA";

int geolat = 0;
int geolon = 0;

Geocoder gc = new Geocoder(this);
try {
    addresses = gc.getFromLocationName(myAddress, 1);
    if(addresses != null) {
        Address x = addresses.get(0);

        StringBuilder mSB = new StringBuilder("Address:\n");
        geolat =(int)(x.getLatitude()*1E6);
        geolon = (int)(x.getLongitude()*1E6);
        mSB.append("latitude: ").append(geolat).append("\n");
        mSB.append("longitude: ").append(geolon);
        tv.setText(mSB.toString());
    }
} catch(IOException e) {
    tv.setText(e.getMessage());
}

int x = 50;
int y = 50;
mScreenLayoutParams =
    new MapView.LayoutParams(MapView.LayoutParams.WRAP_CONTENT,
        MapView.LayoutParams.WRAP_CONTENT,
        x,y,MapView.LayoutParams.LEFT);

final TextView tv = new TextView(this);
tv.setText("Adding View to Google Map");
tv.setTextColor(Color.BLUE);
tv.setTextSize(20);
mapView.addView(tv, mScreenLayoutParams);

x = 250;
y = 250;
mScreenLayoutParams =
    new MapView.LayoutParams(MapView.LayoutParams.WRAP_CONTENT,
        MapView.LayoutParams.WRAP_CONTENT,
        x,y,

```

```

        MapView.LayoutParams.BOTTOM_CENTER);

    Button clickMe = new Button(this);
    clickMe.setText("Click Me");
    clickMe.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            tv.setTextColor(Color.RED);
            tv.setText("Let's play");
        }
    });

    mapView.addView(clickMe, mScreenLayoutParams);
}

@Override
protected boolean isRouteDisplayed() { return false; }
}

```

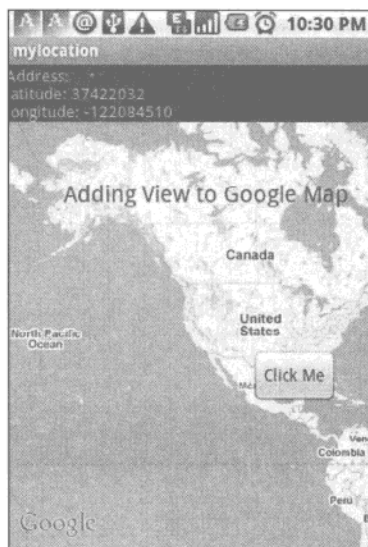


图10-6 在地图上添加视图

10.2.4 秘诀 89：在地图上标记设备的当前位置

`MyLocationOverlay`是一个使用方便的内置图层，它会自动在地图上绘制用户的当前位置，用一个蓝点来标记。它也可以显示精度和用户正在指向的方向（方位）。我们经常使用的方法有下面4个：

- `enableCompass()`——在地图上显示指南针标记；

- ❑ `enableMyLocation()`——注册最准确的位置修正更新信息，并绘制一个由蓝色圆盘包围的闪烁蓝点，代表精度；
- ❑ `getMyLocation()`——返回包含当前位置数据的GeoPoint对象；
- ❑ `getOrientation()`——返回最新设定的罗盘方位。

以上方法为开发人员提供了在地图上使用罗盘的简单方法。

10.2.5 秘诀 90：设置临近警告

LocationManager提供了一个方法来设置临近警告（Proximity Alert），当用户进入或离开一个预先定义好的区域就会触发。该区域由经纬度坐标和半径距离（单位为米）构成。当用户进出特定区域时，警告会由一个PendingIntent对象启动。警告的失效时间也是可设定的。清单10-14是一个如何实现该功能的例子。

清单10-14 设置临近警告的例子

```
double mlatitude=35.41;
double mlongitude=139.46;

float mRadius=500f; // in meters

long expiration=-1; //-1 never expires or use milliseconds

Intent mIntent = new Intent("You entered the defined area");
PendingIntent mFireIntent
    = PendingIntent.getBroadcast(this, -1, mIntent, 0);

mLocationManager.addProximityAlert(mlatitude, mlongitude,
                                    mRadius, expiration, mFireIntent);
```



本章将介绍一些高级开发技术，这些开发技术可以使应用程序更加健壮、运行速度更快，有时还能优化用户界面。本章首先通过一个例子介绍如何自定义Android标准视图控件。接着介绍原生程序开发包（Native Development Kit, NDK），利用该技术可以减少系统开销、提高复杂计算的执行效率。然后讨论了Android系统的安全问题。在此之后，是关于两个不同进程之间的跨进程通信的方法。紧接着介绍Android 2.2版本推出的新功能——备份数据到云端。最后是关于用户界面动画的相关技术的内容。

11.1 Android 的自定义视图

正如第4章所述，Android系统提供了视图（View）对象和视图组（ViewGroup）对象两种视图控件。自定义视图要么一切都从头开始，要么就继承现有的视图控件结构。某些标准控件是在Android框架下基于View或ViewGroup类定义的，如果可能的话，自定义控件可以继承以下现有的视图控件：

- View——Button、EditText、TextView、ImageView等；
- ViewGroup——LinearLayout、ListView、RelativeLayout、RadioGroup等。

秘诀 91：自定义按钮

本秘诀通过一个myButton类实现了自定义按钮。该类扩展了按钮控件，继承了按钮的大部分功能。要想自定义某个控件，最重要的是要重写onMeasure()和onDraw()这两个方法。

onMeasure()方法定义控件尺寸的大小。它有widthMeasureSpec和heightMeasureSpec两个参数。自定义控件需要根据控件内的内容确定其宽度和高度，然后通过这些属性值调用setMeasuredDimension()方法。如果不这么做，则measure()方法将抛出IllegalStateException（非法状态错误）异常。

onDraw()方法允许在控件上绘制自定义的图像。绘制过程是从View布局树的根节点开始向下逐个渲染每个视图节点。绘制完成父控件后才能绘制子控件。如果控件有背景图像，则要先绘制这个背景图像，然后才回调其onDraw()方法。

本秘诀的myButton类声明了八个成员方法以及两个构造方法。这些成员方法如下：

- ❑ setText()——设置按钮上要绘制的文本；
- ❑ setTextSize()——设置文本的大小；
- ❑ setTextColor()——设置文本的颜色；
- ❑ measureWidth()——计算按钮控件的宽度；
- ❑ measureHeight()——计算按钮控件的高度；
- ❑ drawArcs()——绘制圆弧；
- ❑ onDraw()——在按钮控件上绘制图形；
- ❑ onMeasure()——计算和设置按钮控件的边界。

其中setText()、setTextSize()和setTextColor()方法可用来改变文本属性。每次改变文本时，都需要调用invalidate()方法来强制视图重新绘制按钮，以显示更新后的按钮。setText()和setTextSize()方法会调用requestLayout()方法，而setTextColor()则不会调用该方法。这是因为只有控件边界改变时才需要重设布局，而只是文本颜色改变则不需要重设布局。

在onMeasure()方法中，要调用参数为measureWidth()和measureHeight()的方法setMeasuredDimension()。这是自定义控件的一个重要步骤。

方法measureWidth()和measureHeight()传入的参数是父视图的尺寸，根据调用请求中的度量模式返回自定义视图控件的适当尺寸。如果将度量模式设置为EXACTLY，那么该方法会从父视图返回一个精确的子视图尺寸值。如果模式为AT_MOST，则返回当前内容尺寸及父视图尺寸两者间的较小的值，从而确保内容尺寸设置恰当。其他情况下，方法会根据控件内的内容计算出自定义视图控件的高度和宽度。本秘诀中，内容尺寸是由文本的尺寸大小决定的。

drawArcs()方法是一个功能简单明确的方法，用来在按钮上绘制一个圆弧。onDraw()方法会在绘制文本时调用该方法。在此，还可以看到绘制圆弧的动画。每次绘制圆弧时，随着长度的递增会逐渐改变其旋转度数，从而构成优美的动画。

自定义按钮类如清单11-1所示。该类还需要定义一个构造方法，在本秘诀中，我们声明了两个带有不同参数列表的MyButton()构造方法。这两种方法都会使用自定义属性初始化标签视图控件。在图形操作方面，类库android.graphics.*与Matrix和Paint等Java库的格式相似。

清单11-1 src/com/cookbook/advance/MyButton.java

```
package com.cookbook.advance.customComponent;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.graphics.Shader;
```

```
import android.graphics.SweepGradient;
import android.util.AttributeSet;
import android.util.Log;
import android.widget.Button;

public class MyButton extends Button {
    private Paint mTextPaint, mPaint;
    private String mText;
    private int mAscent;
    private Shader mShader;
    private Matrix mMatrix = new Matrix();
    private float mStart;
    private float mSweep;
    private float mRotate;
    private static final float SWEEP_INC = 2;
    private static final float START_INC = 15;

    public MyButton(Context context) {
        super(context);
        initLabelView();
    }

    public MyButton(Context context, AttributeSet attrs) {
        super(context, attrs);
        initLabelView();
    }

    private final void initLabelView() {
        mTextPaint = new Paint();
        mTextPaint.setAntiAlias(true);
        mTextPaint.setTextSize(16);
        mTextPaint.setColor(0xFF000000);
        setPadding(15, 15, 15, 15);
        mPaint = new Paint();
        mPaint.setAntiAlias(true);
        mPaint.setStrokeWidth(4);
        mPaint.setAntiAlias(true);
        mPaint.setStyle(Paint.Style.STROKE);
        mShader = new SweepGradient(this.getMeasuredWidth()/2,
                                   this.getMeasuredHeight()/2,
                                   new int[] { Color.GREEN,
                                                Color.RED,
                                                Color.CYAN, Color.DKGRAY },
                                   null);
        mPaint.setShader(mShader);
    }

    public void setText(String text) {
        mText = text;
    }
}
```

```
        requestLayout();
        invalidate();
    }

    public void setTextColor(int color) {
        mTextPaint.setColor(color);
        requestLayout();
        invalidate();
    }

    public void setTextColor(int color) {
        mTextPaint.setColor(color);
        invalidate();
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec){
        setMeasuredDimension(measureWidth(widthMeasureSpec),
            measureHeight(heightMeasureSpec));
    }

    private int measureWidth(int measureSpec) {
        int result = 0;
        int specMode = MeasureSpec.getMode(measureSpec);
        int specSize = MeasureSpec.getSize(measureSpec);

        if (specMode == MeasureSpec.EXACTLY) {
            // We were told how big to be
            result = specSize;
        } else {
            // Measure the text
            result = (int) mTextPaint.measureText(mText)
                + getPaddingLeft()
                + getPaddingRight();
            if (specMode == MeasureSpec.AT_MOST) {
                result = Math.min(result, specSize);
            }
        }

        return result;
    }

    private int measureHeight(int measureSpec) {
        int result = 0;
        int specMode = MeasureSpec.getMode(measureSpec);
        int specSize = MeasureSpec.getSize(measureSpec);

        mAscent = (int) mTextPaint.ascent();
        if (specMode == MeasureSpec.EXACTLY) {
```

```
// We were told how big to be
result = specSize;
} else {
    // Measure the text (beware: ascent is a negative number)
    result = (int) (-mAscent + mTextPaint.descent())
        + getPaddingTop() + getPaddingBottom();
    if (specMode == MeasureSpec.AT_MOST) {
        Log.v("Measure Height", "At most Height:" + specSize);
        result = Math.min(result, specSize);
    }
}
return result;
}

private void drawArcs(Canvas canvas, RectF oval, boolean useCenter,
    Paint paint) {
    canvas.drawArc(oval, mStart, mSweep, useCenter, paint);
}

@Override protected void onDraw(Canvas canvas) {
    mMatrix.setRotate(mRotate, this.getMeasuredWidth()/2,
        this.getMeasuredHeight()/2);
    mShader.setLocalMatrix(mMatrix);
    mRotate += 3;
    if (mRotate >= 360) {
        mRotate = 0;
    }
    RectF drawRect = new RectF();
    drawRect.set(this.getWidth() - mTextPaint.measureText(mText),
        (this.getHeight() - mTextPaint.getTextSize())/2,
        mTextPaint.measureText(mText),
        this.getHeight() - (this.getHeight() - mTextPaint.getTextSize())/2);
    drawArcs(canvas, drawRect, false, mPaint);
    mSweep += SWEEP_INC;
    if (mSweep > 360) {
        mSweep -= 360;
        mStart += START_INC;
        if (mStart >= 360) {
            mStart -= 360;
        }
    }
    if (mSweep > 180) {
        canvas.drawText(mText, getPaddingLeft(),
            getPaddingTop() - mAscent, mTextPaint);
    }
    invalidate();
}
}
```

该自定义按钮控件可在清单11-2所示的布局中使用。

清单11-2 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_vertical"
    >
    <com.cookbook.advance.customComponent.MyButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/mybutton1"
    />
</LinearLayout>
```

布局XML文件中只有一个ViewGroup、一个LinearLayout和一个View视图控件，通过其所在的路径 com.cookbook.advance.customComponent.myButton 调用。该布局文件可以在 activity 中使用，如清单11-3所示。

清单11-3 src/com/cookbook/advance/ShowMyButton.java

```
package com.cookbook.advance.customComponent;

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

public class ShowMyButton extends Activity{

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

        setContentView(R.layout.main);
        MyButton myb = (MyButton)findViewById(R.id.mybutton1);
        myb.setText("Hello Students");
        myb.setTextSize(40);
    }
}
```

从上面代码中可以看出，自定义按钮在使用方式上和普通按钮是一致的。自定义按钮的显示效果如图11-1所示。



图11-1 自定义按钮示例

11.2 Android的原生组件

如果应用程序中计算密集的功能至关重要，也许就要使用C或C++原生代码开发该方法，以提高程序执行效率。Android NDK可以用来帮助开发原生应用组件。NDK是对Android SDK的补充，它提供了一系列库文件，可用于构建C/C++库。构建一个Android原生组件的步骤如下：

- (1) 下载 Android NDK (<http://developer.android.com/sdk/ndk/>)，其中包括了详细的使用文档；
- (2) 在NDK目录中，通过一般的方法创建Android工程；
- (3) 在第2步创建的工程中，创建jni/文件夹；
- (4) 在jni/文件夹下创建必要的C/C++程序文件；
- (5) 创建一个Android.mk的make文件；
- (6) 从工程目录中运行程序构造脚本（NDK-r4的ndk-build）；
- (7) 在Android Java工程中导入该库，并调用库中的原生方法。

使用Eclipse集成开发环境编译应用程序时，本地的库文件会妥善地自动打包到应用程序中。

秘诀 92：创建原生组件

在本秘诀中，使用了一个由C语言编写的数字阶乘方法，然后在由Java语言编写的activity中调用该C库函数，并将运行结果显示在屏幕上。首先，C语言程序如清单11-4所示。

清单11-4 jni/cookbook.c

```
#include <string.h>
#include <jni.h>
```

```

jint factorial(jint n){
    if(n == 1){
        return 1;
    }
    return factorial(n-1)*n;
}

jint Java_com_cookbook_advance_ndk_ndk_factorial( JNIEnv* env,
                                                    jobject thiz, jint n ) {
    return factorial(n);
}

```

在该C程序中，有一个特殊的数据类型jint，它是在C/C++中定义的Java数据类型。这将提供一种将原生类型转换为Java类型的方法。如果需要将数据从Java返回给C，就需要进行类型转换。表11-1总结了Java与C/C++之间的类型映射。

表11-1 Java 和C/C++之间的类型映射

C/C++中的Java类型	native类型	描 述
jboolean	unsigned char	无符号，8位
jbyte	signed char	有符号，8位
jchar	unsigned short	无符号，16位
jshort	short	有符号，16位
jint	long	有符号，32位
jfloat	float	32位
jlong	long long _int64	有符号，64位
jdouble	double	64位

C程序中有两个函数。第一个函数factorial是用来做实数运算的。Java程序调用第二个函数。这类函数的命名应该遵循JAVA_CLASSNAME_METHOD这样的接口格式。

第二个函数中有三个参数：一个JNIEnv指针，一个jobject对象指针，以及一个Java方法中要用的Java参数。JNIEnv是一个JNI（Java Native Interface，Java原生程序接口）指针，它负责向原生函数传递参数。这些函数要被映射成一个Java方法，它是一个包含了Java虚拟机（JVM）调用接口的结构体。它包含与JVM进行交互以及与Java对象协作的必要功能。在本例中没有调用任何Java方法，该程序使用的唯一参数是一个Java参数jint n。

编译器所需的makefile文件如清单11-5所示，它应该和C程序放在同一目录下。在该文件中为编译器定义了变量LOCAL_PATH，并且调用了CLEAR_VARS，保证在每次编译之前都先清理所有的LOCAL_*变量。然后将LOCAL_MODULE标识为名为ndkcookbook自定义库，此外还标识了待编译的源代码文件。在所有这些声明完成之后，还引用了BUILD_SHARED_LIBRARY。这是一个编译简单程序所需的通用makefile。关于makefile文件格式的详细信息可参考NDK的docs/目录下的ANDROID-MK.TXT文件。

清单11-5 jni/Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE     := ndkcookbook
LOCAL_SRC_FILES  := cookbook.c

include $(BUILD_SHARED_LIBRARY)
```

接下来是编译原生库文件。在NDK-r4中,可以在NDK根目录调用其提供的ndk-build构建脚本,并使用关联的makefile文件编译库文件。而对以前的版本,需要使用命令: `make APP=NAME_OF_APPLICATION`。其被编译后,会生成一个lib/目录,并且目录中包含了原生库文件libndkcookbook.so。在NDK-r4中,还生成了两个gdb文件用来调试。

在使用该库文件的Android activity中可以调用 `System.loadLibrary()` 方法加载ndkcookbook库。此外还需要声明原生方法,如清单11-6所示。运行结果如图11-2所示。

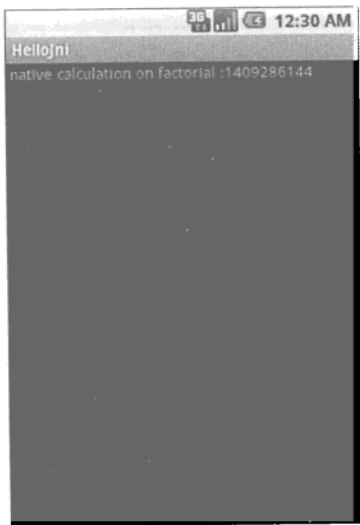


图11-2 NDK应用的运行结果

清单11-6 src/com/cookbook/advance/ndk/ndk.java

```
package com.cookbook.advance.ndk;

import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;
public class ndk extends Activity {
```



```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView tv = new TextView(this);
    tv.setText(" native calculation on factorial :"+factorial(30));
    setContentView(tv);
}

public static native int factorial(int n);
static {
    System.loadLibrary("ndkcookbook");
}
}

```

11.3 Android 的安全机制

Android是一个多进程系统。每个应用都运行在AndroidDalvik虚拟机中，每个Dalvik虚拟机又直接在某个Linux进程中运行，而每个进程都在它自己的沙箱里运行，这就意味着它只能访问自己创建的资源。

默认情况下，每个应用程序都被分配一个唯一的Linux用户ID。也可以设置多个应用程序共享一个用户ID。这样，这些应用程序在访问资源时就具有相同的权限。

应用程序需要从Android系统获得权限才能够访问应用程序沙箱外的资源。大部分Android原生组件都有权限限制。应用程序所需要的权限记录在应用程序的manifest文件中，并在安装过程中显示给用户。如果用户允许其安装，那么也就意味着同意将这些权限授予给应用程序。权限不能在应用程序安装后再追加。这些权限定义在android.Manifest.permission中。

正如第1章所述，每个应用程序都需要一个包含证书的自签名私有密钥。密钥用来识别应用程序的作者，而不是用来管理应用程序权限的。应用程序可以在AndroidManifest文件的permission标签中设置，将权限授权给相关权限组。

秘诀 93：声明和实施权限

权限可以分配给activity, broadcast receiver, 内容提供器和service。如果要给某组件分配权限，可以在应用程序XML文件AndroidManifest的permission元素中声明。例如：

```

<permission android:name="com.myapp"
    android:label="my app"
    android:description="using my app"
    android:permissionGroup="android.permission-group.COST_MONEY"
    android:protectionLevel="dangerous" />

```

这种做法不仅指定了所需的权限，同时也通过protectionLevel属性定义了访问级别。访问级别共分为4级：normal、dangerous、signature以及signatureOrSystem。permissionGroup属性仅用于帮助系统向用户显示权限，它是一个可选项。可用的权限组如下所示：

```

permission group:android.permission-group.DEVELOPMENT_TOOLS
permission group:android.permission-group.PERSONAL_INFO

```

```

permission group:android.permission-group.COST_MONEY
permission group:android.permission-group.LOCATION
permission group:android.permission-group.MESSAGES
permission group:android.permission-group.NETWORK
permission group:android.permission-group.ACCOUNTS
permission group:android.permission-group.STORAGE
permission group:android.permission-group.PHONE_CALLS
permission group:android.permission-group.HARDWARE_CONTROLS
permission group:android.permission-group.SYSTEM_TOOLS

```

使用label、description以及name这些属性使得权限更具有描述性。

11.4 Android 的进程间通信

如果两个应用程序之间需要共享资源，但又不能获取相关权限，这时就可以定义进程间通信（Inter-Process Communication，IPC）消息实现资源共享。为了支持IPC，应用程序需要实现一个接口作为应用程序之间的桥梁。该功能通过Android接口定义语言AIDL（Android Interface Definition Language）提供。

定义AIDL接口与Java接口定义类似。事实上，可以在Eclipse中创建一个Java接口，然后将文件的后缀.java改为.aidl，这样做更简单一些。

目前AIDL支持的数据类型有：

- Java基本数据类型，包括int、boolean、float；
- String；
- CharSequence；
- List；
- Map；
- 其他AIDL生成的接口；
- 实现Parcelable协议并通过值传递的自定义类。

秘诀 94：实现远程过程调用

本秘诀演示了两个activity之间的远程过程调用（RPC）。首先，我们要定义一个AIDL接口，如清单11-7所示。

清单11-7 com.cookbook.advance.rpc 的IAdditionalService.aidl

```

package com.cookbook.advance.rpc;

// Declare the interface.
interface IAdditionalService {
    int factorial(in int value);
}

```

创建AIDL文件以后，Eclipse编译项目时会在gen/目录中自动生成一个IAdditionalService.java文件。该文件的内容不可修改，其中包括了一个实现远程服务所需要的stub类。

在名为rpcService的第一个activity中声明了一个成员变量mBinder，作为IAdditional-Service的stub。它也可以被转化为一个IBinder对象。在onCreate()方法中，将mBinder对象初始化，并设定其调用factorial()方法。onBind()方法中会返回mBinder对象到调用者。在方法onBind()完成后，其他进程中的activity就可以连接到该服务了。该activity的代码如清单11-8所示。

清单11-8 src/com/cookbook/advance/rpc/rpcService.java

```
package com.cookbook.advance.rpc;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;

public class RPCService extends Service {

    IAdditionService.Stub mBinder;
    @Override
    public void onCreate() {
        super.onCreate();
        mBinder = new IAdditionService.Stub() {
            public int factorial(int value1) throws RemoteException {
                int result=1;

                for(int i=1; i<=value1; i++){
                    result*=i;
                }
                return result;
            }
        };
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

现在要定义运行在另一个进程中的第二个activity。相关的布局文件如清单11-9所示。在界面布局中，起主要作用的视图控件有3个：EditText控件用来获取用户输入内容，Button控件用来触发对factorial()方法的调用，ID为result的TextView则用来显示factorial运算的结果。

清单11-9 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Android CookBook RPC Demo"
        android:textSize="22dp" />
    <LinearLayout
        android:orientation="horizontal" android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <EditText android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/value1"
            android:hint="0-30"></EditText>
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/buttonCalc"
            android:text="GET"></Button>
    </LinearLayout>
    <TextView android:layout_width="wrap_content"

        android:layout_height="wrap_content" android:text="result"
        android:textSize="36dp" android:id="@+id/result"></TextView>
</LinearLayout>

```

清单11-10是AndroidManifest文件的代码。在service标签内，添加了一个额外的属性android:process=".remoteService"。它请求系统创建一个名为remoteService的新进程来运行第二个activity。

清单11-10 AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.advance.rpc"
    android:versionCode="1" android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".rpc" 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=".rpcService" android:process=".remoteService"/>
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>

```

清单11-11是第二个activity的代码。它需要调用bindService()方法,通过rpcService来获得factorial()方法。bindService()方法需要一个服务连接实例作为接口,以监视应用服务状态。因此,该activity包含了一个名为myServiceConnection的内部类来实现服务连接。

myServiceConnection类和IAdditionService类在名为rpc的activity中实例化。myServiceConnection负责监听onServiceConnected和onServiceDisconnected这两个回调方法。onServiceConnected回调方法用于将IBinder实例传递给IAdditionService。而onServiceDisconnected回调方法则将IadditionService实例置为null。

在rpc activity中还定义了initService()和releaseService()这两个方法。initService()方法会尝试实例化一个新的myServiceConnction对象。然后它会创建一个指向特定包名和类名的intent对象,并将该对象连同myServiceConnection实例和BIND_AUTO_CREATE标志一并传递给bindService。服务绑定后,就会触发onServiceConnected回调方法,将IBinder传递给IAdditionService实例,这样rpc activity就可以调用factorial方法了。最后运行的效果如图11-3所示。

清单11-11 src/com/cookbook/advance/rpc/rpc.java

```
package com.cookbook.advance.rpc;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class rpc extends Activity {
    IAdditionService service;
    myServiceConnection connection;

    class myServiceConnection implements ServiceConnection {

        public void onServiceConnected(ComponentName name,
                                      IBinder boundService) {
            service = IAdditionService.Stub.asInterface((IBinder) boundService);
            Toast.makeText(rpc.this, "Service connected", Toast.LENGTH_SHORT)
                .show();
        }
    }
}
```

```
public void onServiceDisconnected(ComponentName name) {
    service = null;
    Toast.makeText(rpc.this, "Service disconnected", Toast.LENGTH_SHORT)
        .show();
}

private void initService() {
    connection = new myServiceConnection();
    Intent i = new Intent();
    i.setClassName("com.cookbook.advance.rpc",
        com.cookbook.advance.rpc.rpcService.class.getName());
    if(!bindService(i, connection, Context.BIND_AUTO_CREATE)) {
        Toast.makeText(rpc.this, "Bind Service Failed", Toast.LENGTH_LONG)
            .show();
    }
}

private void releaseService() {
    unbindService(connection);
    connection = null;
}

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

    initService();

    Button buttonCalc = (Button) findViewById(R.id.buttonCalc);

    buttonCalc.setOnClickListener(new OnClickListener() {
        TextView result = (TextView) findViewById(R.id.result);
        EditText value1 = (EditText) findViewById(R.id.value1);

        public void onClick(View v) {
            int v1, res = -1;
            try {
                v1 = Integer.parseInt(value1.getText().toString());
                res = service.factorial(v1);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
            result.setText(new Integer(res).toString());
        }
    });
}

@Override
protected void onDestroy() {
```

```
        releaseService();  
    }  
}
```

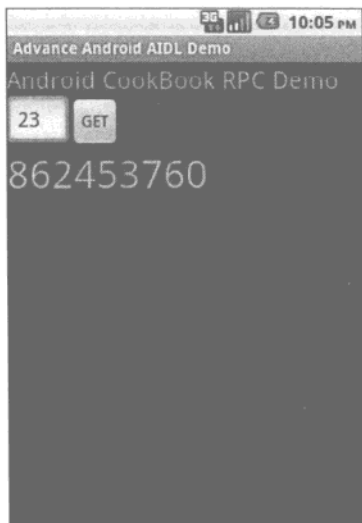


图11-3 AIDL应用程序的运行效果

11.5 Android 的备份管理器

在Android设备中，用户需要保存不同应用程序的各种数据，如笔记、游戏存档、应用程序设置信息、通讯录条目等。以上这些数据在删除后均不可恢复。过去开发者想要解决这个问题的话，需要将这些应用程序数据保存到远端的服务器上。但是随着Android 2.2的发布，谷歌推出了其托管的Android备份服务。所有的应用数据都可以通过备份服务将其保存到云端。

11.5.1 秘诀 95：备份运行时数据

Android为开发者提供了BackupManager类用于通知Backup服务进行备份和恢复操作。在接到通知以后，备份管理器（backup manager）将会向应用程序请求备份数据，并在备份过程中将其传递给云存储服务器。同样，在还原过程中它也可以从备份中获取备份数据，并将之返还给应用程序。

备份代理（backup agent）是BackupManager和应用程序的通信接口。开发者可以在他们的程序中通过继承BackupAgent类来为应用程序创建备份代理。任何继承BackupAgent的类都需要重载onBackup()和onRestore()这两个方法。onBackup()方法在每次调用dataChanged()方法时被触发。而方法onRestore()在每次调用requestRestore()方法时被触发。

```

public class MyBackupAgent extends BackupAgent {

    @Override
    public void onCreate() {
        ...
    }

    @Override
    public void onBackup(ParcelFileDescriptor oldState,
                        BackupDataOutput data,
                        ParcelFileDescriptor newState){
        ...
    }

    @Override
    public void onRestore(BackupDataInput data, int appVersionCode,
                        ParcelFileDescriptor newState){
        ...
    }
}

```

onBackup()方法传递给BackupManager 3个参数:

- oldState——返回上次备份的状态;
- data——需要备份的数据;
- newState——记录当前备份的状态,在下次备份时将会变成oldState。

在实现onBackup()方法时,需要将BackupManager传入的oldState与当前的状态数据进行核对,如果相同则不需要备份。如果不相同,则写入所传的data数据,并更新newState以完成备份。

onRestore()方法同样也传递给BackupManager 3个参数:

- data——最新备份的数据;
- appVersionCode——备份操作时应用程序的版本代码,版本代码定义在AndroidManifest XML文件中的android:versionCode属性中;
- newState——记录当前状态作为还原点。

任何由版本变化引起的数据转换都应该在onRestore()方法中完成。这也是向BackupManager传递appVersionCode应用程序版本代码的目的所在。在数据恢复到应用程序中以后,应用程序的状态会发生改变。从这一点来看,就需要记录newState。

11.5.2 秘诀 96: 备份文件到云端

BackupAgent类主要用来保存应用程序运行时数据。若要保存文件,就需要使用另外一个代理类BackupAgentHelper。该类是备份代理类的封装,支持两种不同的备份助手(helper):

- SharedPreferencesBackupHelper——用于备份SharedPreferences文件;
- FileBackupHelper——用于备份文件。

详细内容如清单11-12所示。

清单11-12 继承BackupAgentHelper类示例

```

public class MyFileBackupAgentHelper extends BackupAgentHelper {
    @Override
    public void onCreate() {
        FileBackupHelper filehelper = new FileBackupHelper(this,
                                                                DATA_FILE_NAME);

        addHelper(FILE_HELPER_KEY, helper);
        SharedPreferencesBackupHelper xmlhelper
            = new SharedPreferencesBackupHelper(this, PREFS);
        addHelper(PREFS_BACKUP_KEY, helper);
    }
}

```

所有的备份代理助手类都需要重载其onCreate()方法。BackupAgent可能拥有一个以上的备份助手。但在继承BackupAgentHelper类时，不需要再重载onBackup和onRestore方法，因为BackupAgent会帮我们妥善处理这些事情。

11.5.3 秘诀 97：触发备份与还原操作

若要触发备份或还原操作，需要为应用程序定义一个备份代理。这可以通过在application标签内添加android:backupAgent属性来完成，如清单11-13所示。

清单11-13 AndroidManifest.xml

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cookbook.databackuprestore"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8"/>
    <application android:label="Backup/Restore"
        android:backupAgent="myBackupAgent">
        <activity android:name="MyBandRActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

应用程序在任何时候向BackupManager触发备份或恢复操作时，它都会实例化指定的备份代理。例如，某个主activity的片段可如下所示：

```

public class MyBandRActivity extends Activity {

    BackupManager mBackupManager;

    @Override

```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
    mBackupManager = new BackupManager(this);
}

void dataUpdate() {
    ...
    // We also need to perform an initial backup; ask for one
    mBackupManager.dataChanged();
}
}

```

在MyBandRActivity activity中，onCreate()方法创建了一个BackupManager实例。如果要备份，就得调用BackupManager的dataChanged()方法。然后BackupManager会找到AndroidManifest文件中定义的BackupAgent，并调用它的onBackup()方法。

Android系统支持两种触发还原操作的方法。第一种方法是使用BackupManager的requestRestore()方法。该方法会触发备份代理的onRestore()方法。还有一种方法是只要用户启动设备上的“恢复到出厂设置”操作或重装应用程序时，Android系统会自动触发应用程序的还原操作。

Android系统除了能在应用程序中触发备份或还原操作外，同时也支持使用命令行脚本bmgr完成同样的工作。比如要触发备份操作，请键入：

```
> adb shell bmgr backup <package>
```

要触发还原操作，则要键入：

```
> adb shell bmgr restore <package>
```

其实，无论何时向BackupManager发送备份请求，它都不会立即执行备份操作，直到时机成熟才会触发。若想强制BackupManager立即执行备份操作，可以键入：

```
> adb shell bmgr run
```

11.6 Android 的动画功能

Android支持两种动画类型：逐帧动画和渐变动画（Tween animation）。逐帧动画按顺序显示图片序列。这样开发者可以定义要显示的图片，然后像播幻灯片一样显示它们。

逐帧动画首先需要在布局文件中声明一个包含一系列item元素的animation-list元素，以定义不同图片的显示顺序。oneshot属性声明动画是仅播放一次还是重复播放。动画列表的XML文件如清单11-14所示。

清单11-14 res/anim/animated.xml

```

<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/anddev1" android:duration="200" />

```

```

<item android:drawable="@drawable/anddev2" android:duration="200" />
<item android:drawable="@drawable/anddev3" android:duration="200" />
</animation-list>

```

为了显示逐帧动画，还需要将动画设置成某个视图控件的背景：

```

ImageView im = (ImageView) this.findViewById(R.id.myanimated);
im.setBackgroundResource(R.anim.animated);
AnimationDrawable ad = (AnimationDrawable)im.getBackground();
ad.start();

```

设置完视图控件背景以后，通过调用getBackground()方法可以获得一个drawable类型的资源并强制其转化为AnimationDrawable类型，然后调用start()方法播放动画。

创建渐变动画则要使用不同的方法，它主要是对单个图像执行的一系列变形。在Android中，系统提供了下面的类，这些类是制作所有动画的基础：

- AlphaAnimation——控制透明度变化；
- RotateAnimation——控制旋转角度；
- ScaleAnimation——控制放大和缩小；
- TranslateAnimation——控制位置变化。

这四个动画类可以在activity、布局、视图控件等之间作成过渡动画。它们都可以通过XML布局文件定义，使用<alpha>、<rotate>、<scale>和<translate>标签。它们必须在<set>标签内中包含动画集（AnimationSet）。

- <alpha>属性

android:fromAlpha, android:toAlpha

alpha值用于定义不透明度，它的变化范围是从0.0（透明）到1.0（不透明）。

- <rotate>属性

android:fromDegrees, android:toDegrees,

android:pivotX, android:pivotY

rotate定义了动画围绕给定轴心点旋转的角度。

- <scale>属性

android:fromXScale, android:toXScale,

android:fromYScale, android:toYScale,

android:pivotX, android:pivotY

scale定义了如何沿着x轴方向（或y轴方向）改变视图的大小。视图大小发生变化时枢轴（pivot）位置是保持不变的，但这个枢轴位置是可以被定义的。

- <translate>属性

android:fromXDelta, android:toXDelta,

android:fromYDelta, android:toYDelta

translate定义了视图平移的距离。

秘诀 98：创建动画

本秘诀创建了一个在收到新邮件时使用的“新邮件”动画示例。主布局文件如清单11-15所示，最后的运行效果如图11-4所示。

清单11-15 res/layout/main.xml

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

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

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

    <Button
        android:id="@+id/startAnimated"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="you've got mail"
    />
</LinearLayout>
```



图11-4 页面布局中的动画

要在视图中实现动画效果，需要定义一个动画集。在Eclipse中，用鼠标右键点击res/文件夹，选择New→Android XML File，然后将其命名为animated.xml，并选择文件类型为Animation。这样该文件就可以被编辑并用于创建内容。代码如清单11-16所示。

清单11-16 res/anim/animated.xml

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

<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator">
    <translate android:fromXDelta="100%p" android:toXDelta="0"
        android:duration="5000" />
    <alpha android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="3000" />
    <rotate
        android:fromDegrees="0"
        android:toDegrees="-45"
        android:toYScale="0.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="3000" />

    <scale
        android:fromXScale="0.0"
        android:toXScale="1.4"
        android:fromYScale="0.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="700"
        android:duration="3000"
        android:fillBefore="false" />

</set>
```

主activity如清单11-17所示。这个activity非常简单，通过AnimationUtils类生成一个Animation对象，并使用该对象来加载animated.xml文件所定义的动画集。这样每次用户点击按钮时，程序都会使用image view对象，并调用startAnimation()方法来运行Animation对象先前加载的动画。

清单11-17 src/com/cookbook/advance/myanimation.java

```
package com.cookbook.advance;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.animation.Animation;
```

```
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageView;

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

        final ImageView im
            = (ImageView) this.findViewById(R.id.myanimated);
        final Animation an
            = AnimationUtils.loadAnimation(this, R.anim.animated);

        im.setVisibility(View.INVISIBLE);
        Button bt = (Button) this.findViewById(R.id.startAnimated);
        bt.setOnClickListener(new OnClickListener(){
            public void onClick(View view){
                im.setVisibility(View.VISIBLE);
                im.startAnimation(an);
            }
        });
    }
}
```



调试程序很耗费时间，可能和开发工作费时相当甚至更多。那么了解如何调试常见问题的各种方法有助于节省大量的时间和精力。本章将会介绍调试Android应用程序的基本方法，并检视现有的多种调试工具。首先要讨论的是最常见的Eclipse集成开发环境中的调试工具，然后是Android软件开发工具包提供的Android工具，最后我们将探讨Android系统中的一些调试工具。每个应用程序都有差别，因此应当根据程序自身的特点来选择适当的调试方法。

12.1 Eclipse 内置的调试工具

安装有ADT插件的Eclipse集成开发环境是一个非常易用的开发环境。它包含了一个所见即所得的用户界面，而且内置了将资源布局文件转换成构建Android可执行程序所需内容的相关转换工具。下面将按步骤介绍如何设定配置。此处假定其版本为Eclipse 3.4 (Ganymede)，其他版本的Eclipse的配置过程大同小异。

12.1.1 秘诀 99：设置运行配置

每个应用程序的运行配置 (run configuration) 都各不相同。它负责告诉Eclipse如何运行工程，如何启动activity，以及将应用程序部署到模拟器上还是连机设备上。每当我们创建一个新应用程序时，ADT都会自动设定其运行配置，但我们也可以按照此处描述的方法自定义配置。

如果想新建一个运行配置或者编辑现有的运行配置，请在Eclipse中选择Run→Run Configurations... (或者Debug Configurations...)，就会弹出如图12-1所示的运行配置菜单。在运行配置窗口中包含了三个与应用程序测试相关的选项卡。

- Android——指定要启动的工程及activity。
- Target——选择应用程序将要运行的虚拟设备。对于模拟器环境，这里可以设定启动参数，如网络速度和时延。这样使得开发者可以模拟更为真实的无线网络连接状态，从而测试应用程序的行为。开发者也可以选择每次启动时清除模拟器中的用户数据。
- Common——指定运行配置的保存路径，以及是否将其显示在Eclipse的收藏夹菜单中。

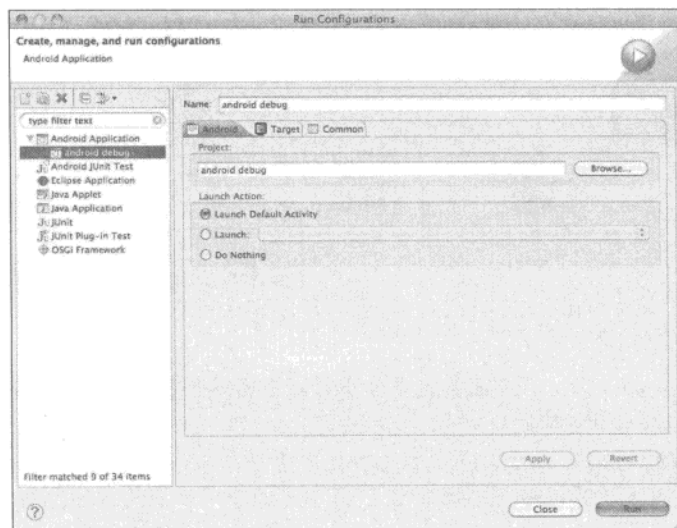


图12-1 Eclipse的运行配置菜单

正确完成上述设置后,只要单击运行按钮就可以在目标设备上运行应用程序了。如果Android真机设备没有连接到主机或者选择的设备是虚拟的,就会启动模拟器来运行应用程序。

12.1.2 秘诀 100: 使用 DDMS

当应用程序在目标设备上运行之后,可以打开Dalvik 调试监控服务 (DDMS) 检查设备状态,如图12-2所示。DDMS可以通过命令行启动或者在Eclipse中选择Window→Open Perspective→DDMS启动。

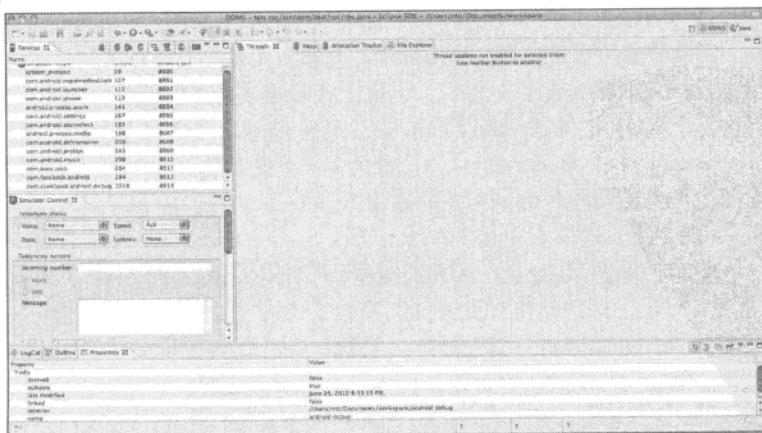


图12-2 DDMS控制面板

DDMS内置4个不同的面板，分别用来显示不同类型的调试数据。

- **Devices**——显示已连接的Android设备，包括模拟器和真实Android设备。
- **Emulator Control**——提供多种控制以向模拟器注入事件和数据，包括Telephony Status、Telephony Action以及Location Control。
 - The Telephony Status设定语音、数据格式、网络速度和时延。
 - The Telephony Actions提供了一种给模拟器拨出仿真语音电话或发送短消息的方法。如果指定为SMS，则可自定义消息内容。
 - The Location Control提供了一种向模拟器中GPS提供程序（GPS provider）发送模拟GPS信号的方法。
- **底部面板**——包括LogCat、Outline和Properties 3个选项卡。LogCat选项卡实时显示设备的所有数据记录，包括系统日志信息和用户日志信息。在应用程序中使用Log类可以捕获上述信息。
- **设备状态面板**——右上方的面板包含4个选项卡：Thread、Heap、Allocation Tracker以及File Explorer。它们通常用来分析处理的过程。如果在Devices选项卡中单击某个设备，这四个选项卡就会反映当前选择的设备或者模拟器的运行参数，具体界面如图12-3所示。

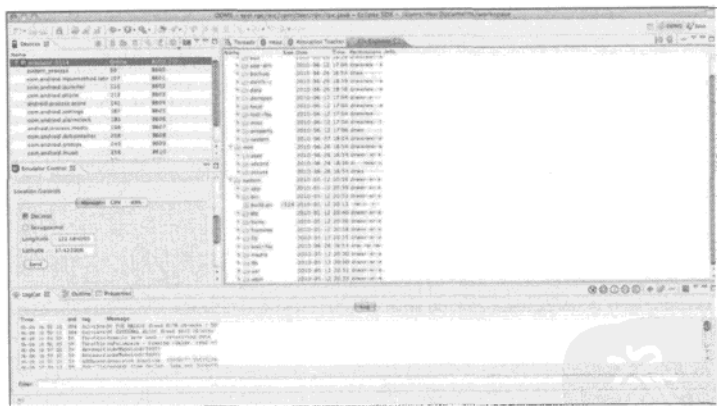


图12-3 DDMS控制面板的设备状态面板界面

12.1.3 秘诀 101：断点调试

开发者还可以在调试模式下运行应用程序并在应用程序中插入断点，以便在运行时“冻结”程序。首先，要在debug模式下启动应用程序，弹出如图12-4所示的确认对话框。如果选择Yes，则会切换到图12-5所示的Debug模式界面。

Debug模式界面包括一个显示源文件窗口，此外还有变量窗口（variables）、断点窗口（breakpoints）、大纲窗口（outline）等一些相关窗口。双击Eclipse中显示行数的左部区域（紧挨着代码）可以设置断点。当代码行前出现一个蓝色小圆圈时表示断点设置成功。

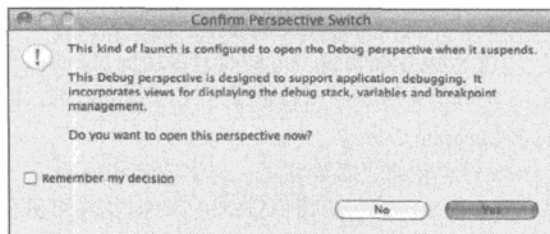


图12-4 切换为debug模式的确认对话框

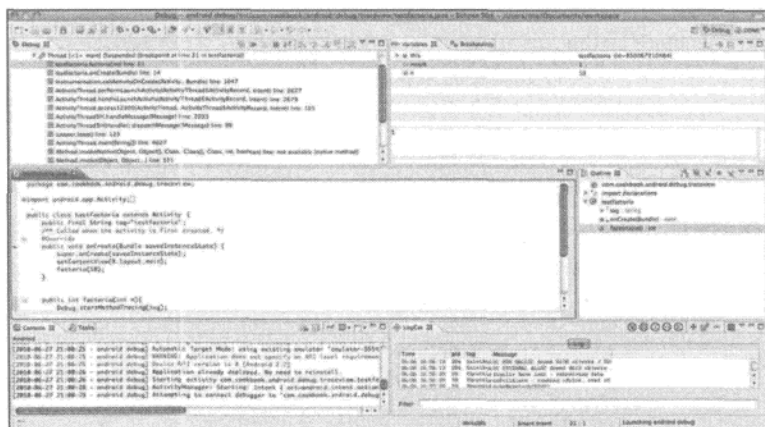


图12-5 Eclipse中的Debug窗口

对于嵌入式程序开发者而言，设置断点是一种标准的程序调试方法。它可以使用应用运行在断点处停止、逐步跟踪程序、查看内存中的变量值的变化，以及在运行时修改变量值。上述这些功能为程序员追踪复杂的bug和异常行为提供了强有力的方法。

12.2 Android SDK 中的调试工具

Android SDK提供了多种可单独使用的调试工具。下面我们将探讨Android Debug Bridge、LogCat、Hierarchy Viewer以及TraceView这4个调试工具。它们全部位于Android SDK安装路径下的tools/目录下。

12.2.1 秘诀 102：使用 Android Debug Bridge 工具

使用Android Debug Bridge (ADB) 工具可以管理模拟器实例状态或者通过USB连接的Android设备状态。ADB包括三个组成部分：一个客户端、一个服务器以及一个守护进程。客户端组件可从开发机上通过ADB shell脚本启动。服务器组件在开发机上作为后台进程运行。通过下面命令可以开启和终止adb服务进程：

```
> adb start-server
> adb kill-server
```

守护进程以后台进程的形式运行于模拟器或Android设备上。

12.2.2 秘诀 103: 使用 LogCat 工具

Logcat是Android提供的实时日志工具。它可以收集循环缓冲区中所有的系统日志和程序日志,程序员可以查看和过滤这些日志信息。它既可以作为一个独立工具使用,也可以作为DDMS工具一个子模块使用。

执行adb shell登录到设备或者在adb中使用logcat命令就可以在该设备上使用Logcat工具了:

```
> [adb] logcat [<option>] ... [<filter-spec>] ...
```

每一条利用android.util.Log类输出的调试信息与一个标签和优先级关联。标签应当是有意义的词组,并且和activity的动作相关。通过综合使用标签和优先级可使调试信息易于阅读和过滤。可用标签如下:

- ☐ V——Verbose (优先级最低)
- ☐ D——Debug
- ☐ I——Info
- ☐ W——Warning
- ☐ E——Error
- ☐ F——Fatal
- ☐ S——Silent (优先级最高,不打印任何调试信息)

LogCat数据中包含了太多的信息,为了避免信息过载,我们应该为logcat命令指定tag:priority参数来设置过滤器。比如:

```
> adb logcat ActivityManager:V *:S
```

该语句规定只显示关于ActivityManager的verbose (V) 数据,同时通过silence (S) 参数静默其他日志命令。

Android记录日志使用循环缓冲区系统。默认情况下,所有的日志信息都记录在主日志缓冲区。Android 2.2 SDK以后又添加了另外两个日志缓冲区:一个用来存放radio和telephony的相关信息,另外一个则存放和事件相关的缓冲区信息。不同的缓冲区都可以通过-b开关开启。例如:

```
> adb logcat -b events
```

该缓冲区同时也显示事件相关的信息:

```
I/menu_opened( 135): 0
I/notification_cancel( 74): [com.android.phone,1,0]
I/am_finish_activity( 74):
[1128378040,38,com.android.contacts/.DialtactsActivity,app-request]
I/am_pause_activity( 74):
[1128378040,com.android.contacts/.DialtactsActivity]
I/am_on_paused_called( 135): com.android.contacts.RecentCallsListActivity
I/am_on_paused_called( 135): com.android.contacts.DialtactsActivity
```

```

I/am_resume_activity( 74): [1127710848,2,com.android.launcher/.Launcher]
I/am_on_resume_called( 135): com.android.launcher.Launcher
I/am_destroy_activity( 74):
[1128378040,38,com.android.contacts/.DialtactsActivity]
I/power_sleep_requested( 74): 0
I/power_screen_state( 74): [0,1,468,1]
I/power_screen_broadcast_send( 74): 1
I/screen_toggled( 74): 0
I/am_pause_activity( 74): [1127710848,com.android.launcher/.Launcher]

```

另外一个例子是：

```
> adb logcat -b radio
```

下面是与radio和telephony相关的信息：

```

D/RILJ ( 132): [2981]< GPRS_REGISTRATION_STATE {1, null, null, 2}
D/RILJ ( 132): [2982]< REGISTRATION_STATE {1, null, null, 2, null, null,
null, null, null, null, null, null, null, null}
D/RILJ ( 132): [2983]< QUERY_NETWORK_SELECTION_MODE {0}
D/GSM ( 132): Poll ServiceState done: oldSS=[0 home T - Mobile T - Mo-
bile 31026 Unknown CSS not supported -1 -lRoamInd: -lDefRoamInd: -1]
newSS=[0 home T - Mobile T - Mobile 31026 Unknown CSS not supported -1 -
lRoamInd: -lDefRoamInd: -1] oldGprs=0 newGprs=0 oldType=EDGE newType=EDGE
D/RILJ ( 132): [UNSL]< UNSOL_NITZ_TIME_RECEIVED 10/06/26,21:49:56-28,1
I/GSM ( 132): NITZ: 10/06/26,21:49:56-28,1,237945599 start=237945602
delay=3
D/RILJ ( 132): [UNSL]< UNSOL_RESPONSE_NETWORK_STATE_CHANGED
D/RILJ ( 132): [2984]> OPERATOR
D/RILJ ( 132): [2985]> GPRS_REGISTRATION_STATE
D/RILJ ( 132): [2984]< OPERATOR {T - Mobile, T - Mobile, 31026}
D/RILJ ( 132): [2986]> REGISTRATION_STATE
D/RILJ ( 132): [2987]> QUERY_NETWORK_SELECTION_MODE
D/RILJ ( 132): [2985]< GPRS_REGISTRATION_STATE {1, null, null, 2}
D/RILJ ( 132): [2986]< REGISTRATION_STATE {1, null, null, 2, null, null,
null, null, null, null, null, null, null, null}
D/RILJ ( 132): [2987]< QUERY_NETWORK_SELECTION_MODE {0}

```

Logcat在调试基于Java的Android应用程序时非常有用。然而，当应用程序应用到原生组件时，就很难追踪错误了。在这种情况下，原生组件应当将日志记录到system.out或者system.err。默认状态下，Android系统会将标准输出和标准错误（system.out和system.err）输出到/dev/null。使用下面的ADB命令可以将它们转发到某日志文件：

```

> adb shell stop
> adb shell setprop log.redirect-stdio true
> adb shell start

```

上面的代码将停止正在运行的模拟器或者设备实例，并使用shell命令setprop重定向输出，然后重启模拟器或设备实例。

12.2.3 秘诀 104：使用 Hierarchy Viewer 工具

Hierarchy Viewer工具可以帮助我们调试和理解用户界面。该工具提供了两个查看用户界面的选项卡，一个选项卡直观地显示了视图控件布局层次结构（即Layout View），另外一个选项卡是界面的放大查看工具（即Pixel Perfect View）。

启动hierarchyviewer工具即可进入Hierarchy Viewer界面。执行该文件会看到如图12-6所示的界面。页面左边显示了连接到当前计算机的Android设备列表。选中某个设备时，页面右边的窗口就会显示该设备目前正在运行的程序列表；然后，我们就可以选择需要调试或者优化用户界面的程序了。

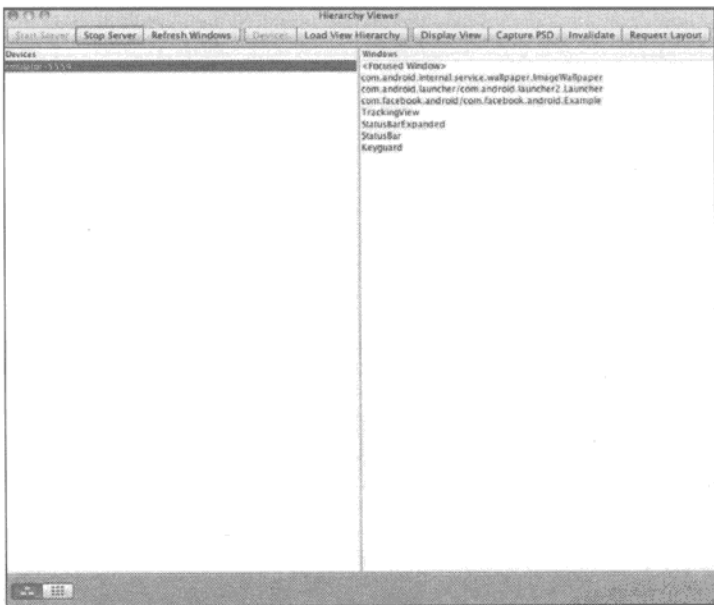


图12-6 Hierarchy Viewer工具

选定程序后，即可点击Load View Hierarchy按钮查看Hierarchy/Viewer构建的视图树了。该界面也被称为Layout View，其中包含以下三个视图：

- 树视图（Tree View）——页面左侧的视图控件层次图；
- 属性视图（Properties View）——页面右上方显示的所选择视图控件的属性列表；
- 线框视图（Wire-frame View）——页面右下方显示的布局线框图。

工具界面如图12-7所示。

三个视图之间彼此关联。选中视图控件树结构中的某个节点，控件所对应的属性列表视图及线框视图也会立刻更新。Android系统对每个应用程序可生成的视图树有限制，树的深度不可超过10，广度不可超过50。在Android 1.5及其之前的版本中，如果视图树超过限制系统会抛出堆栈

溢出异常。即便在这些限制以内，浅层次布局树层还是会使得程序运行得更快更流畅。我们可以通过在布局文件中合并视图或者使用RelativeLayout替代LinearLayout以实现对视图树的优化。

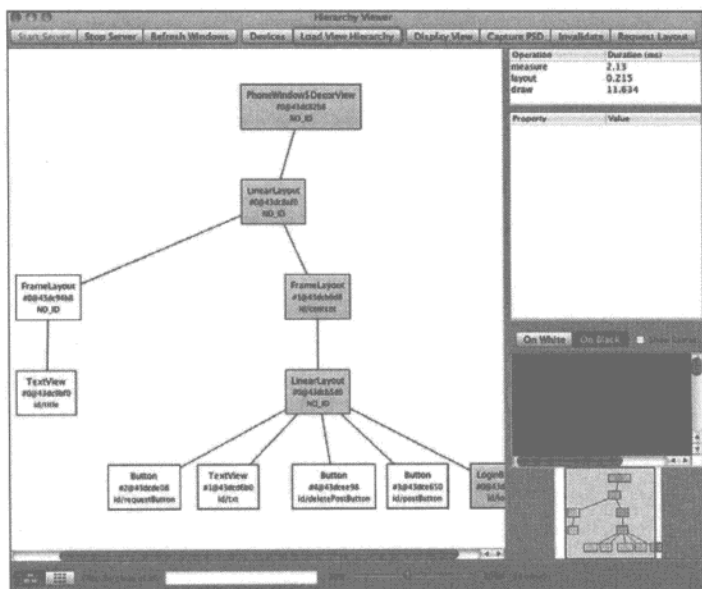


图12-7 Hierarchy View工具的Layout View选项卡视图

12.2.4 秘诀 105：使用 TraceView 工具

TraceView是一个用来优化应用程序性能的工具。使用该工具需要在应用程序中实现Debug类。它会创建包含跟踪信息的日志文件，以便进行分析。在此，本秘诀将演示如何使用TraceView工具。秘诀中声明了一个阶乘方法，以及一个调用该阶乘方法的方法。主activity的代码如清单12-1所示。

清单12-1 src/com/cookbook/Android/debug/traceview/testfactorial.java

```
package com.cookbook.android.debug.traceview;

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

public class testfactorial extends Activity {
    public final String tag="testfactorial";
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

```

        factorial(10);
    }

    public int factorial(int n) {
        Debug.startMethodTracing(tag);
        int result=1;
        for(int i=1; i<=n; i++) {
            result*=i;
        }
        Debug.stopMethodTracing();
        return result;
    }
}

```

factorial()方法调用了Debug类两次；调用startMethodTracing()方法开始跟踪，并生成testfactorial.trace^①文件。调用stopMethodTracing()方法，系统会继续缓冲生成的跟踪数据。当factorial(10)方法返回时，生成跟踪文件并保存到/sdcard/目录中。在文件生成后，从开发机可以通过下面的命令取回该文件：

```
> adb pull /sdcard/testfactorial.trace
```

TraceView工具位于Android SDK的tool文件夹中，可以帮助我们分析跟踪文件：

```
> traceview testfactorial.trace
```

运行脚本命令后，将会生成如图12-8所示的分析界面。

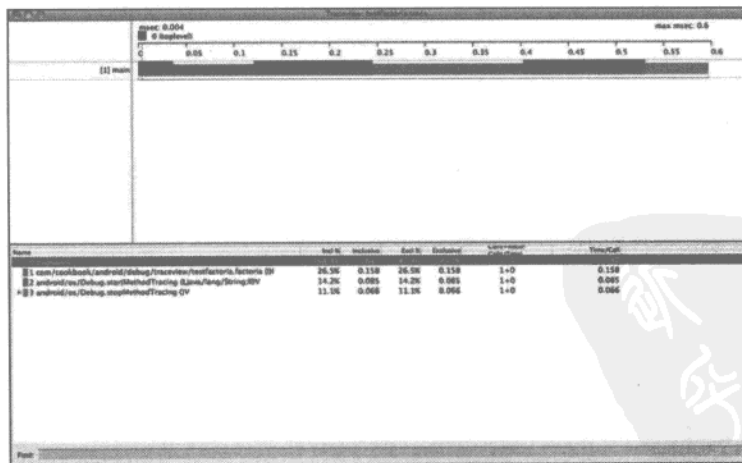


图12-8 TraceView工具的分析界面

界面会显示一个时间轴面板和一个Profile面板。屏幕上半部分的时间轴面板描述了每个线程

① 如果没有在Debug语句中设置名字则默认为dmtrace.trace。——译者注

> adb shell ps

该命令的输出结果如图12-10所示。

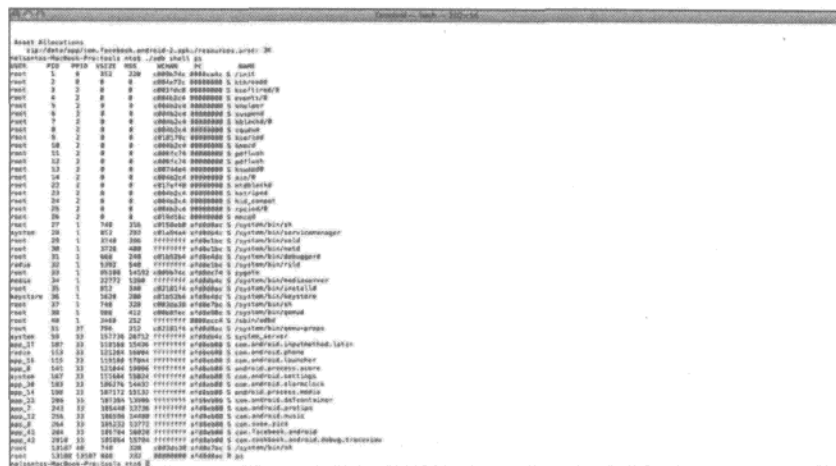


图12-10 ps命令的输出结果

该命令还显示了每个正在运行进程的进程ID (PID) 和用户ID。使用dumpsys命令可以查看内存分配情况：

> adb shell dumpsys meminfo <package name>

该命令的输出结果如图12-11所示。

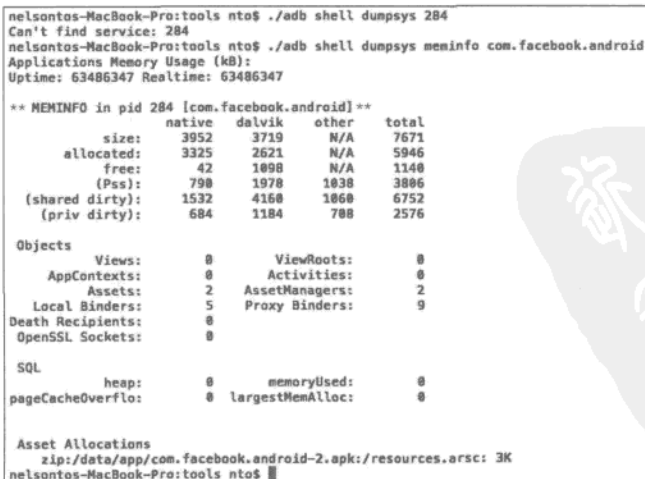


图12-11 dumpsys 命令的输出结果

上述命令可显示Java组件和原生组件的相关信息。这些信息对于优化和分析用原生程序开发包（NDK）开发的应用程序很有帮助。除内存信息之外，还可以查看进程中使用了多少个视图控件，使用了多少activity，以及使用了多少应用程序资源等。

秘诀 106：设置 GDB 调试

GNU工程DeBugger（GDB）是Linux系统中调试应用程序的常用工具。在Android系统中，可以使用GDB工具调试原生库文件。在NDK r4中，每当生成原生库文件时，同时也会产生gdbserver和gdb.setup。我们可以使用下列命令安装gdb：

```
> adb shell
> adb /data/
> mkdir myfolder
> exit
> adb push gdbserver /data/myfolder
```

若要运行gdb，可使用如下命令：

```
> adb shell /data/myfolder/gdbserver host:port <native program>
```

例如，对于一个当前运行于Android设备的应用程序myprogram，其IP地址为10.0.0.1，端口号为1234，可以运行下面的命令开启服务器：

```
> adb shell /data/myfolder/gdbserver 10.0.0.1:1234 myprogram
```

然后打开另一个终端窗口，在该程序上运行gdb。

```
> gdb myprogram
(gdb) set sysroot ../
(gdb) set solib-search-path ../system/lib
(gdb) target remote localhost:1234
```

在gdb命令提示符后，第一条命令设置目标镜像的根目录，第二条命令用来设置共享库文件的搜索路径，最后一个命令设置目标设备。target remote localhost:1234命令运行后，就可以使用gdb环境调试应用程序了。



[General Information]

书名=Android开发秘籍

作者=(美)斯蒂尔,(美)图著

页数=266

出版社=北京市:人民邮电出版社

出版日期=2011.07

SS号=12811965

DX号=000008138395

URL=<http://book1.duxiu.com/bookDetail.jsp?dxNumber=000008138395&d=D23460CFA7BFD6AB5B8A9FEBECA3C970>