



图灵程序设计丛书



- Amazon榜首畅销书全新升级
Android入门进阶不二之选
- 全面覆盖Android开发知识点
通过实战项目手把手教你逐步写Android应用

Android 编程权威指南

(第2版)

[美] Bill Phillips Chris Stewart Brian Hardy Kristin Marsicano 著
王明发 译

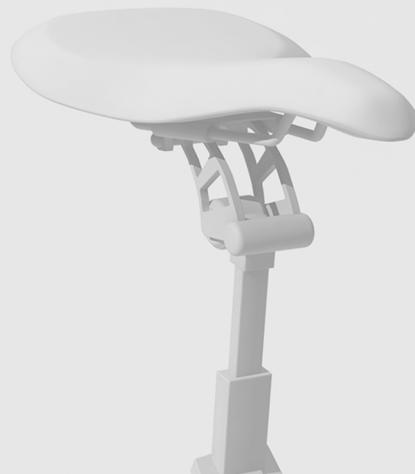
Android Programming
The Big Nerd Ranch Guide, Second Edition



人民邮电出版社
POSTS & TELECOM PRESS



图灵程序设计丛书



Android 编程权威指南 (第2版)

[美] Bill Phillips Chris Stewart Brian Hardy Kristin Marsicano 著
王明发 译



Android Programming
The Big Nerd Ranch Guide, Second Edition

人民邮电出版社
北京

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.) 等著 ; 王明发译. -- 2版. -- 北京 :
人民邮电出版社, 2016.5
(图灵程序设计丛书)
ISBN 978-7-115-42246-0

I. ①A… II. ①菲… ②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(2016)第083123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。本书主要以其 Android 训练营教学课程为基础，融合了几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 34 章，详细介绍了 8 个 Android 应用。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，或者了解面向对象编程，那就立刻开始 Android 编程之旅吧！

-
- ◆ 著 [美] Bill Phillips Chris Stewart Brian Hardy
Kristin Marsicano
 - 译 王明发
 - 责任编辑 朱 巍
 - 执行编辑 杨 琳
 - 责任印制 彭志环
 - ◆ 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：35.5
 - 字数：839千字 2016年5月第2版
 - 印数：24 001~28 000册 2016年5月北京第1次印刷
 - 著作权合同登记号 图字：01-2015-8300号
-

定价：109.00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：京东工商广字第 8052 号

献词

献给上帝，或者你的其他信仰所在。读者，希望本书内容能帮到你。请不要在意它们是怎么得来的。我曾以为这是我自己的功劳，但幸运的是我错了。

——B.P.

献给我的爸爸David，他教我懂得辛苦工作的意义。献给我的妈妈Lisa，她一直推动着我去做正确的事。

——C.S.

献给Donovan。希望他的人生充满activity，并且知道在何时使用fragment。

——B.H.

献给我的爸爸Dave Vadas，他激励并支持我投身计算机行业。献给我的妈妈Joan Vadas，在这么多年的浮浮沉沉里，她总能让我保持乐观。（她会给我支招：心情不好的时候就看集《黄金女郎》吧！）

——K.M.

致 谢

我们很不安，因为封面上只印了我们几个人的名字。事实上，本书能够出版发行，完全是整个团队的合作成果。我们满怀感激之情。

- 感谢我们Android开发团队的同事Andrew Lunsford、Bolot Kerimbaev、Brian Gardner、David Greenhalgh、Jason Atwood、Josh Skeen、Kurt Nelson、Matt Compton、Paul Turner和Sean Farrell老师。他们在使用持续更新的材料教学时充满耐心，并针对其中的内容给出了改进建议。即使我们有魔法变出分身来做这些事情，我们也不愿意这么做。我们更愿意将我们的想法汇集起来，和诸位同事分享，并肩作战。他们所做的一切我们都绝对信任。
- 特别感谢Sean Farrell。每次Android Studio升级，他就为本书同步更新一大批截图。还要感谢Matt Compton，我们在Google Play商店里的示例应用都是他发布的。
- 感谢Big Nerd Ranch设计团队的Kar Loong Wong和Zack Simon。Kar设计的BeatBox应用美观而又不失威慑力。他还为第33章提供了宝贵意见。Zack专门挤出时间为设计了MockWalker应用。在我们眼里，他俩的能力深不可测，简直就是超人。
- 感谢技术审校Frank Robles和Roy Kravitz，他们帮我们找出并修正了多处问题。
- 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Elizabeth Holaday。她带我们一次又一次走出困境。在她的指导下，我们才能有的放矢，写出清晰、有趣、简洁的教材。感谢你，Liz，你不仅做事有条理，而且非常有耐心。尽管住得很远，但我们总能随时得到你的帮助。
- 感谢Ellie Volckhausen为本书设计了封面。
- 感谢我们的排印编辑Simone Payment。他发现并修正了不少瑕疵。
- 感谢IntelligentEnglish.com网站的Chris Loper。他设计并制作了本书的纸质版、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。

最后感谢我们的学员。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，我们才有动力编写这本书，再次感谢。

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——Java语言。但仅掌握Java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，我们认为，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。以此为起点，你就能深入学习和开发，成长为一名合格的Android开发者。

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知识忘得差不多了，那么阅读本书应该不会有太大的问题。对于接口、匿名内部类等重要的Java语言点，我们会提供必要的简短回顾。建议在学习过程中手边备上一本Java参考书，方便查阅。

第 2 版有哪些新内容

本书第2版会教读者学习如何使用Android Studio集成环境开发各类面向Android 5.1 (Lollipop) 并向后兼容Android 4.1 (Jelly Bean) 的应用。除了Lollipop新引入的toolbar和material design，本书还更新介绍了一些Android编程基础知识。此外，本书还涵盖了支持库中的一些新工具，如RecyclerView和Google Play服务，以及一些标准库工具，如SoundPool、animation和assets。

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，建议不要跳读，以免学习效果大打折扣。显然，本书不适合作为参考书。本书旨在帮你跨越学习的初始障碍，进而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。

我们为学员提供了良好的培训环境：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。因此，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) 参与本书论坛的交流讨论 (forums.bignerdranch.com)；
- (4) 寻求Android开发高手的帮助。

本书内容

通过本书，我们会学习开发8个Android应用。有些应用很简单，一章即可讲完；有些则相对复杂。最复杂的一个应用跨越了11章。通过这些精心编排的应用，你能学到很多重要的理论知识和开发技巧，并获得最直接的开发经验。

GeoQuiz

本书的第一个应用，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intent等内容。

BeatBox

通过这个可以震慑敌人的应用，继续深入学习fragment、媒体文件的播放与控制、主题以及drawable。

□ NerdLauncher

通过个性化启动器的开发，深入学习intent以及任务的概念知识。

□ PhotoGallery

通过开发从Flickr网站下载并显示照片的客户端应用，学习Android服务、多线程、网络内容获取服务等知识。

□ DragAndDraw

一个简单的画图应用，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。

□ Sunset

一个漂亮的日落动画应用，通过它学习Android动画知识。

□ Locatr

查询当前位置的Flickr图片并显示在地图上的应用。借此应用学习如何使用定位服务和地图。

挑战练习

大部分章末都配备有练习题。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。

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。这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，请访问论坛<http://forums.bignerdranch.com>寻求帮助。

深入学习

部分章末还包含一块名为“深入学习”的内容。这些内容针对相应章内的知识点，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，但还是希望大家有兴趣阅读并有所收获。

代码风格

有别于其他Android开发学习社区的编码风格，我们有着自己的判断与选择，主要体现在以下两个方面。

□ 在监听器代码部分使用匿名内部类

这主要看个人倾向。我们认为，使用匿名内部类，代码可以更简练，监听器实现方法更一目了然。尽管在高性能要求的场景下，匿名内部类可能会有一些问题，但大多数情况下都很正常。

□ 自第7章引入fragment后，后续所有用户界面都使用它

对于这一点，我们有充足的理由坚持。相信我们，使用得当的话，fragment就是Android

开发人员手中的一大利器。一旦适应了它，也就没想象中的那么难用了。相比activity，fragment在创建和显示用户界面时明显具有更加灵活的优势，因此值得为此付出努力。

版式说明

为方便读者阅读，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);  
}
```

Android 版本

本书主要针对当前广泛在用的各个系统版本（Android 4.1至Android 5.1）进行开发教学。虽然更老的系统版本仍有人在用，但对大多数开发者来说，为这部分人开发应用就是个赔本的买卖。如果应用确实需要支持Android 4.1之前的系统版本（尤其是Android 2.2和Android 2.3），请参考本书第1版相关内容。

Google还会不断地发布新版本Android系统。请放心，Android能很好地向后兼容支持（详见第6章），即便有了新系统，本书所授技术和知识也不会过时。而且，通过forums.bignerdranch.com论坛，我们也会不断跟踪Android开发新动向，及时为读者提供开发指导和支持。

开发必备工具

准备开发前，你需要安装Android Studio。基于流行的IntelliJ IDEA创建，Android Studio是用于Android开发的一套集成开发工具。

Android Studio的安装包括：

- Android SDK

最新版本的Android SDK。

- Android SDK工具和平台工具

用来测试与调试应用的一套工具。

- Android模拟器系统镜像

用来在不同虚拟设备上开发测试应用。

本书撰写时，Google一直在积极开发和更新Android Studio版本。因此，请注意了解你当前在用版本和本书所用版本之间的差异。如需帮助，请访问forums.bignerdranch.com论坛。

Android Studio的下载与安装

可以从Android开发者网站下载Android Studio：<https://developer.android.com/sdk/>。

首次安装的话，你还需要从<http://www.oracle.com>下载并安装Java开发者套件（JDK7）。

如仍有安装问题，请访问网址<https://developer.android.com/sdk/>寻求帮助。

下载早期版本的SDK

Android Studio自带最新版本的SDK和系统模拟器镜像。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。在Android Studio中，选择Tools → Android → SDK Manager菜单项，如图0-1所示。

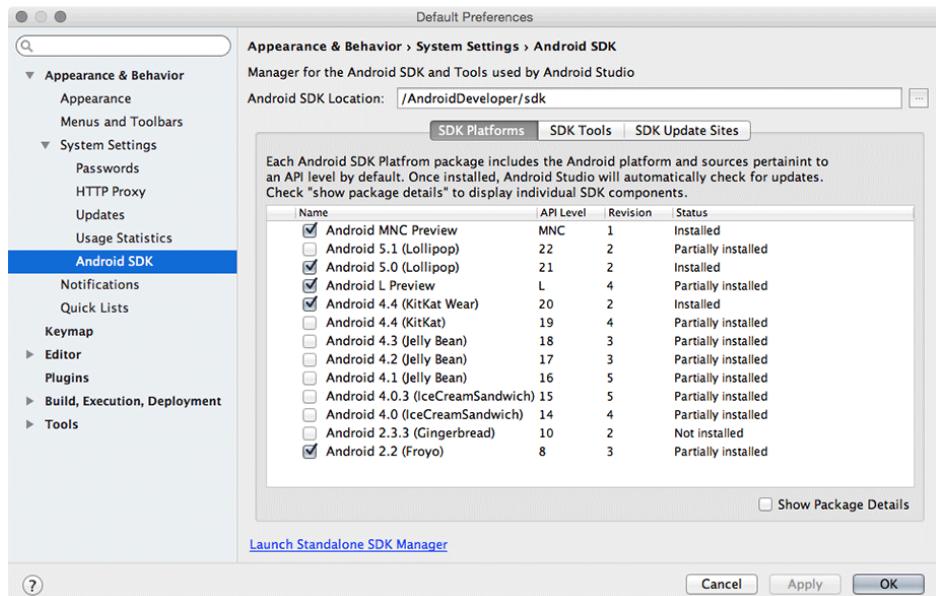


图0-1 Andriod SDK管理器

选择并安装需要的Android版本和工具。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最新发布内容，比如新系统平台或新版本工具等。

第三方模拟器

Google一直在设法提高Android模拟器的运行速度，并取得了显著效果。目前，它已能很好地运行本书中的应用。

如果仍不满意，可试试Genymotion这个较为流行第三方Android模拟器。本书偶尔会用到Genymotion模拟器。想了解有关它的更多信息，请访问网址<http://genymotion.com/>。

硬件设备

用模拟器和Genymotion来测试应用已足够了，但在测试应用性能时，就一定要用真实的Android设备。如果手头有物理设备，建议在需要时使用。

目 录

第 1 章 Android 开发初体验	1
1.1 应用开发基础	1
1.2 创建 Android 项目	2
1.3 Android Studio 使用导航	6
1.4 用户界面设计	7
1.4.1 视图层级结构	10
1.4.2 组件属性	11
1.4.3 创建字符串资源	12
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	17
1.6.1 引用组件	18
1.6.2 设置监听器	18
1.7 创建提示消息	20
1.8 使用模拟器运行应用	23
1.9 深入学习：Android 编译过程	25
第 2 章 Android 与 MVC 设计模式	29
2.1 创建新类	29
2.2 Android 与 MVC 设计模式	32
2.3 更新视图层	34
2.4 更新控制层	36
2.5 在设备上运行应用	40
2.5.1 连接设备	40
2.5.2 配置设备用于应用开发	41
2.6 添加图标资源	42
2.6.1 向项目中添加资源	43
2.6.2 在 XML 文件中引用资源	45
2.7 关于挑战练习	46
2.8 挑战练习：为 TextView 添加监听器	46
2.9 挑战练习：添加后退按钮	46
2.10 挑战练习：从按钮到图标按钮	47
第 3 章 Activity 的生命周期	49
3.1 日志跟踪理解 Activity 生命周期	50
3.1.1 输出日志信息	50
3.1.2 使用 LogCat	52
3.2 设备旋转与 Activity 生命周期	55
3.3 设备旋转前保存数据	59
3.4 再探 Activity 生命周期	61
3.5 深入学习：测试 onSaveInstanceState(Bundle) 方法	62
3.6 深入学习：日志记录的级别与方法	64
第 4 章 Android 应用的调试	65
4.1 异常与栈跟踪	66
4.1.1 诊断应用异常	67
4.1.2 记录栈跟踪日志	68
4.1.3 设置断点	69
4.1.4 使用异常断点	72
4.2 Android 特有的调试工具	73
4.2.1 使用 Android Lint	73
4.2.2 R 类的问题	75
第 5 章 第二个 activity	76
5.1 创建第二个 activity	77
5.1.1 创建新的 activity	78
5.1.2 创建新的 activity 子类	81
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	84

5.3 activity 间的数据传递	86
5.3.1 使用 intent extra	86
5.3.2 从子 activity 获取返回结果	89
5.4 activity 的使用与管理	94
5.5 挑战练习	97
第 6 章 Android SDK 版本与兼容	98
6.1 Android SDK 版本	98
6.2 Android 编程与兼容性问题	99
6.2.1 比较合理的版本	99
6.2.2 SDK 最低版本	101
6.2.3 SDK 目标版本	101
6.2.4 SDK 编译版本	101
6.2.5 安全添加新版本 API 中的 代码	101
6.3 使用 Android 开发者文档	104
6.4 挑战练习：报告编译版本	106
第 7 章 UI fragment 与 fragment 管理器	107
7.1 UI 设计的灵活性需求	108
7.2 fragment 的引入	108
7.3 着手开发 CriminalIntent	109
7.3.1 创建新项目	112
7.3.2 fragment 与支持库	113
7.3.3 在 Android Studio 中增加依赖 关系	114
7.3.4 创建 Crime 类	117
7.4 托管 UI fragment	118
7.4.1 fragment 的生命周期	118
7.4.2 托管的两种方式	119
7.4.3 定义容器视图	119
7.5 创建 UI fragment	120
7.5.1 定义 CrimeFragment 的布局	121
7.5.2 创建 CrimeFragment 类	122
7.6 添加 UI fragment 到 Fragment- Manager	125
7.6.1 fragment 事务	126
7.6.2 FragmentManager 与 fragment 生命周期	129
7.7 采用 fragment 的应用架构	130
7.8 深入学习：为什么应优先使用 支持库版 fragment	131
7.9 深入学习：使用操作系统内置版 fragment	131
第 8 章 使用布局与组件创建用户界面	132
8.1 升级 Crime 类	132
8.2 更新布局	133
8.3 生成并使用组件	135
8.4 深入探讨 XML 布局属性	136
8.4.1 样式、主题及主题属性	136
8.4.2 dp、sp 以及屏幕像素密度	137
8.4.3 Android 开发设计原则	138
8.4.4 布局参数	139
8.4.5 边距与内边距	139
8.5 使用图形布局工具	140
8.5.1 创建水平模式布局	141
8.5.2 添加新组件	142
8.5.3 在属性视图中编辑组件属性	143
8.5.4 在框架视图中重新组织组件	144
8.5.5 更新子组件的布局参数	145
8.5.6 android:layout_weight 属性 的工作原理	146
8.5.7 图形布局工具使用总结	147
8.5.8 组件 ID 与多种布局	148
8.6 挑战练习：日期格式化	148
第 9 章 使用 RecyclerView 显示列表	149
9.1 升级 CriminalIntent 应用的 模型层	150
9.2 使用抽象 activity 托管 fragment	153
9.2.1 通用的 fragment 托管布局	153
9.2.2 抽象 activity 类	154
9.3 RecyclerView、Adapter 和 ViewHolder	158
9.3.1 ViewHolder 和 Adapter	159
9.3.2 使用 RecyclerView	161
9.3.3 实现 Adapter 和 ViewHolder	163
9.4 定制列表项	166

9.4.1 创建列表项布局.....	166
9.4.2 使用定制列表项视图	168
9.5 响应点击	170
9.6 深入学习： ListView 和 GridView	171
9.7 深入学习：单例	171
第 10 章 使用 fragment argument.....	173
10.1 从 fragment 中启动 activity	173
10.1.1 附加 extra 信息.....	174
10.1.2 获得 extra 信息.....	175
10.1.3 使用 Crime 数据更新 CrimeFragment 视图	176
10.1.4 直接获得 extra 信息的缺点	177
10.2 fragment argument	177
10.2.1 附加 argument 给 fragment.....	178
10.2.2 获得 argument	179
10.3 刷新显示列表项	180
10.4 通过 fragment 获得返回结果.....	182
10.5 挑战练习：实现高效的 RecyclerView 刷新.....	183
10.6 深入学习：为何要用 fragment argument	183
第 11 章 使用 ViewPager.....	185
11.1 创建 CrimePagerActivity.....	186
11.1.1 ViewPager 与 PagerAdapter	187
11.1.2 整合并配置使用 CrimePagerActivity	188
11.2 FragmentStatePagerAdapter 与 FragmentPagerAdapter	190
11.3 深入学习：ViewPager 的工作原理	192
11.4 深入学习：以代码的方式创建布局	193
第 12 章 对话框.....	194
12.1 使用 AppCompat 兼容库	195
12.2 创建 DialogFragment	196
12.2.1 显示 DialogFragment	198
12.2.2 设置对话框的显示内容	199
12.3 fragment 间的数据传递	202
12.3.1 传递数据给 DatePicker- Fragment	203
12.3.2 返回数据给 Crime- Fragment	204
12.4 挑战练习：更多对话框	211
12.5 挑战练习：按设备类型展现 DialogFragment	212
第 13 章 工具栏.....	213
13.1 AppCompat	213
13.2 工具栏菜单	216
13.2.1 在 XML 文件中定义菜单	217
13.2.2 创建菜单	221
13.2.3 响应菜单项选择	223
13.3 实现层级式导航	225
13.4 可选菜单项	226
13.4.1 切换菜单项标题	227
13.4.2 “还有个问题”	229
13.5 深入学习：工具栏与操作栏	231
13.6 挑战练习：删除 crime 记录	231
13.7 挑战练习：优化字符串资源显示	231
13.8 挑战练习：用于 RecyclerView 的 空视图	232
第 14 章 SQLite 数据库	233
14.1 定义 Schema	233
14.2 创建初始数据库	234
14.3 修改 CrimeLab 类	238
14.4 写入数据库	239
14.4.1 使用 ContentValues	239
14.4.2 插入和更新记录	240
14.5 读取数据库	242
14.5.1 使用 CursorWrapper	243
14.5.2 创建模型层对象	244
14.6 深入学习：数据库高级主题介绍	247
14.7 深入学习：应用上下文	248
14.8 挑战练习：删除 Crime 记录	248
第 15 章 隐式 intent	249
15.1 添加按钮组件	250

15.2	添加嫌疑人信息至模型层	252
15.3	使用格式化字符串	254
15.4	使用隐式 intent	255
15.4.1	隐式 intent 的组成	255
15.4.2	发送消息	256
15.4.3	获取联系人信息	259
15.4.4	检查可响应任务的 activity	263
15.5	挑战练习：ShareCompat	265
15.6	挑战练习：又一个隐式 intent	265
第 16 章 使用 intent 拍照		266
16.1	布置照片	266
16.2	外部存储	269
16.3	使用相机 intent	272
16.3.1	外部存储使用权限	272
16.3.2	触发拍照	273
16.4	缩放和显示位图	274
16.5	功能声明	277
16.6	深入学习：使用 include 标签	278
16.7	挑战练习：优化照片显示	278
16.8	挑战练习：优化缩略图加载	279
第 17 章 Master-Detail 用户界面		280
17.1	增加布局灵活性	281
17.1.1	修改 SingleFragment-Activity	282
17.1.2	创建包含两个 fragment 容器的布局	283
17.1.3	使用别名资源	284
17.1.4	创建平板设备专用可选资源	285
17.2	Activity：fragment 的托管者	286
17.3	深入学习：设备屏幕尺寸的确定	295
第 18 章 Assets		297
18.1	为何使用 assets	297
18.2	创建 BeatBox 应用	298
18.3	导入 assets	301
18.4	处理 assets	303
18.5	使用 Assets	305
18.6	访问 Assets	308
18.7	深入学习：什么是 non-assets	308
第 19 章 使用 SoundPool 播放音频		309
19.1	创建 SoundPool	309
19.2	加载音频文件	310
19.3	播放音频	311
19.4	释放音频	313
19.5	设备旋转和对象保存	314
19.5.1	保留 fragment	315
19.5.2	旋转和已保留 fragment	316
19.6	深入学习：是否要保留	318
19.7	深入学习：设备旋转处理再探	318
第 20 章 样式与主题		321
20.1	颜色资源	321
20.2	样式	322
20.3	主题	324
20.4	添加主题颜色	327
20.5	覆盖主题属性	328
20.6	修改按钮属性	332
20.7	深入学习：样式继承拾遗	334
20.8	深入学习：引用主题属性	335
20.9	挑战练习：创建多版本主题	335
第 21 章 XML drawable		336
21.1	统一按钮样式	337
21.2	shape drawable	338
21.3	state list drawable	340
21.4	layer list drawable	341
21.5	深入学习：为什么要用 XML drawable	342
21.6	深入学习：使用 9-patch 图像	343
21.7	深入学习：使用 Mipmap 图像	347
第 22 章 深入学习 intent 和任务		348
22.1	创建 NerdLauncher 项目	348
22.2	解析隐式 intent	351
22.3	在运行时创建显式 intent	355
22.4	任务与后退栈	357
22.4.1	在任务间切换	357
22.4.2	启动新任务	358

22.5 使用 NerdLauncher 应用作为设备主屏幕.....	361	25.4 优化应用.....	422
22.6 挑战练习：应用图标.....	362	25.5 挑战练习：深度优化 PhotoGallery 应用	423
22.7 深入学习：进程与任务.....	362	第 26 章 后台服务	424
22.8 深入学习：并发文档.....	364	26.1 创建 IntentService	424
第 23 章 HTTP 与后台任务	367	26.2 服务的作用	427
23.1 创建 PhotoGallery 应用	368	26.3 查找最新返回结果	428
23.2 网络连接基本	371	26.4 使用 AlarmManager 延迟运行服务	430
23.3 使用 AsyncTask 在后台线程上运行代码	373	26.4.1 合理控制服务启动的频度	432
23.4 线程与主线程	374	26.4.2 PendingIntent	433
23.5 从 Flickr 获取 JSON 数据	376	26.4.3 使用 PendingIntent 管理定时器	434
23.6 从 AsyncTask 回到主线程	383	26.5 控制定时器	434
23.7 清理 AsyncTask	386	26.6 通知信息	437
23.8 深入学习：AsyncTask 再探	387	26.7 挑战练习：可穿戴设备上的通知	439
23.9 深入学习：AsyncTask 的替代方案	388	26.8 深入学习：服务细节内容	440
23.10 挑战练习：Gson	388	26.8.1 服务的能与不能	440
23.11 挑战练习：分页	388	26.8.2 服务的生命周期	440
23.12 挑战练习：动态调整网格列	389	26.8.3 non-sticky 服务	440
第 24 章 Looper、Handler 和 HandlerThread	390	26.8.4 sticky 服务	441
24.1 配置 RecyclerView 以显示图片	390	26.8.5 绑定服务	441
24.2 批量下载缩略图	393	26.9 深入学习：JobScheduler 和 JobService	442
24.3 与主线程通信	393	26.10 深入学习：Sync Adapter	445
24.4 创建并启动后台线程	394	26.11 挑战练习：在 Lollipop 设备上使用 JobService	446
24.5 Message 与 message handler	396	第 27 章 broadcast intent	447
24.5.1 消息的剖析	397	27.1 一般 intent 和 broadcast intent	447
24.5.2 Handler 的剖析	397	27.2 接收系统 broadcast：重启后唤醒	448
24.5.3 使用 handler	398	27.2.1 standalone receiver	448
24.5.4 传递 handler	402	27.2.2 使用 receiver	450
24.6 深入学习：AsyncTask 与线程	407	27.3 过滤前台通知消息	452
24.7 挑战练习：预加载以及缓存	407	27.3.1 发送 broadcast intent	452
24.8 深入学习：解决图片下载问题	408	27.3.2 动态 broadcast receiver	453
第 25 章 搜索	409	27.3.3 使用私有权限	455
25.1 搜索 Flickr 网站	410	27.3.4 使用有序 broadcast	458
25.2 使用 SearchView	414	27.4 receiver 与长时运行任务	462
25.3 使用 shared preferences 实现轻量级数据存储	419	27.5 深入学习：本地事件	462
		27.5.1 使用 EventBus	463

27.5.2 使用 Rxjava.....	463
27.6 深入学习：检测 fragment 的状态.....	464
第 28 章 网页浏览	466
28.1 最后一段 Flickr 数据.....	466
28.2 简单方式：隐式 intent.....	469
28.3 较难方式：使用 WebView.....	470
28.4 处理 WebView 的设备旋转问题.....	476
28.5 深入学习：注入 JavaScript 对象	477
28.6 深入学习：KitKat 的 WebView.....	478
28.7 挑战练习：使用后退键浏览历史 网页	478
28.8 挑战练习：非 HTTP 链接支持	479
第 29 章 定制视图与触摸事件	480
29.1 创建 DragAndDraw 项目.....	480
29.1.1 创建 DragAndDraw- Activity.....	481
29.1.2 创建 DragAndDraw- Fragment.....	481
29.2 创建定制视图	482
29.3 处理触摸事件.....	484
29.4 onDraw(...)方法内的图形绘制	488
29.5 挑战练习：设备旋转问题.....	490
29.6 挑战练习：旋转矩形框	490
第 30 章 属性动画	492
30.1 建立场景.....	492
30.2 简单属性动画	495
30.2.1 视图属性转换	498
30.2.2 使用不同的 interpolator	500
30.2.3 色彩渐变	500
30.3 播放多个动画	502
30.4 深入学习：其他动画 API	504
30.4.1 传统动画工具	504
30.4.2 转场	504
30.5 挑战练习.....	504
第 31 章 地理位置和 Play 服务	505
31.1 地理位置和定位类库	505
31.2 创建 Locatr 项目	506
31.3 Play 服务和模拟器	507
31.4 创建 Locatr 应用	510
31.5 配置 Google Play 服务	512
31.6 使用 Google Play 服务	514
31.7 基于地理位置的 Flickr 搜索	516
31.8 获取定位数据	517
31.9 寻找并显示图片	519
31.10 挑战练习：进度指示器	521
第 32 章 使用地图	522
32.1 导入 Play 地图服务库	522
32.2 Android 上的地图服务	522
32.3 地图 API 设置	523
32.4 创建地图	525
32.5 获取更多地理位置数据	526
32.6 使用地图	529
32.7 深入学习：团队开发和 API key	534
第 33 章 material design	536
33.1 material surface	536
33.1.1 elevation 和 Z 值	538
33.1.2 state list animator	539
33.2 动画工具	540
33.2.1 circular reveal	541
33.2.2 shared element transition	542
33.3 新的视图组件	545
33.3.1 card	545
33.3.2 floating action button	547
33.3.3 snackbar	548
33.4 深入学习 material design	549
第 34 章 编后语	550
34.1 终极挑战	550
34.2 关于我们	551
34.3 致谢	551

第1章

Android开发初体验

1

本章介绍编写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。后续章节还会涉及这些概念并有更加详细的讲解。

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。

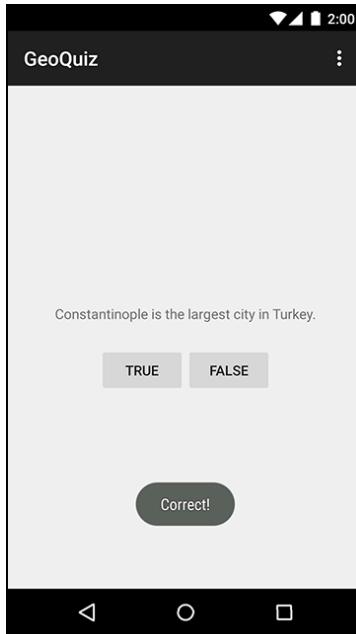


图1-1 正确答案应该是伊斯坦布尔（Istanbul），而不是君士坦丁堡

1.1 应用开发基础

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。
- 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。

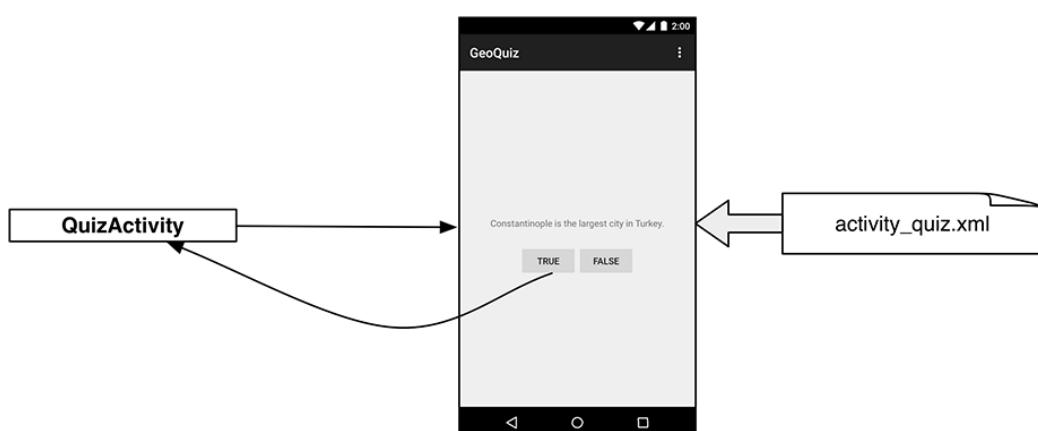


图1-2 QuizActivity管理着activity_quiz.xml文件定义的用户界面

学习了这些基本概念后，我们来创建本书第一个应用。

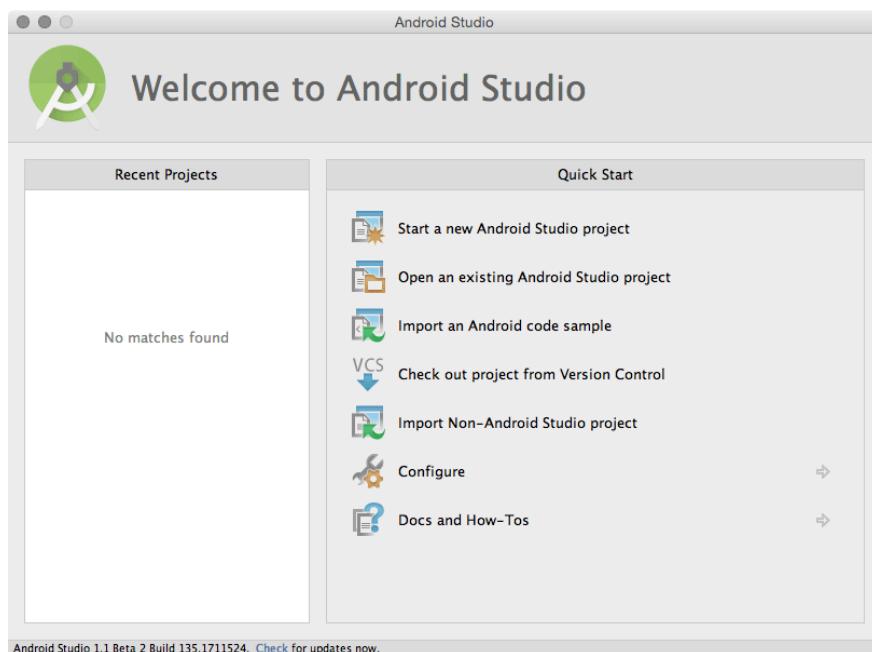
1.2 创建Android项目

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Android Studio程序，首次运行的话，会看到如图1-3所示的欢迎对话框。

在欢迎界面，选择创建Android Studio新项目选项（Start a new Android Studio project）；非首次运行的话，选择File→New Project...菜单项即可。

现在，你应该打开了新建项目向导界面。在此界面的应用名称（Application name）处输入GeoQuiz，如图1-4所示。在公司域名（Company Domain）处输入android.bignerdranch.com。此时自动产生的包名称（Package name）会变为com.bignerdranch.android.geoquiz。至于项目存储位置（Project location），这就看个人喜好了。



1

图1-3 欢迎来到Android Studio

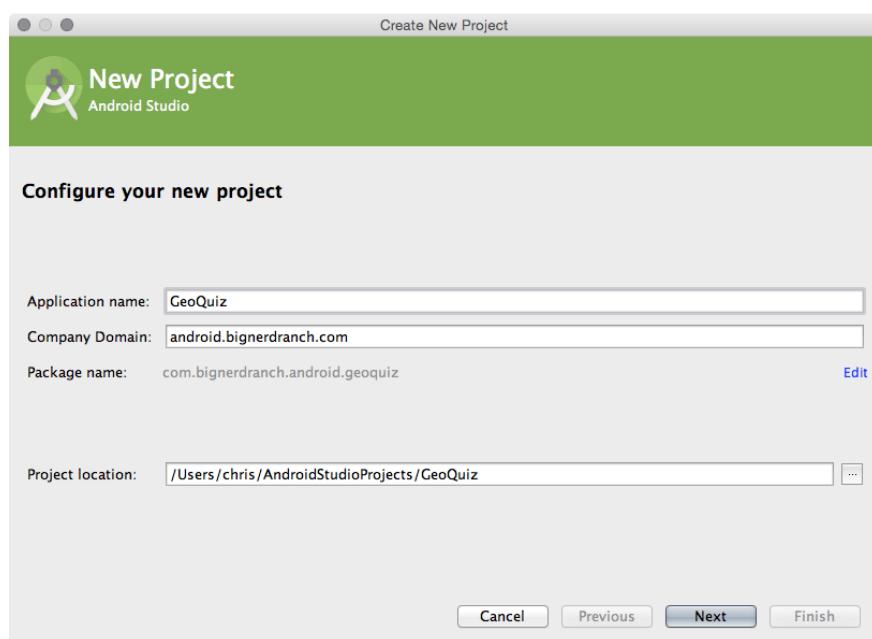


图1-4 创建新项目

注意，以上包名遵循了“DNS反转”约定，也就是将企业组织或公司的域名反转后，在尾部附加上应用名称。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。

单击Next按钮，接下来配置应用支持哪些版本的Android设备。GeoQuiz应用只能在手机上运行，所以这里勾选Phone and Tablet选项。SDK最低版本选择API 16: Android 4.1 (Jelly Bean)，如图1-5所示。第6章会介绍Android不同SDK版本的差异。

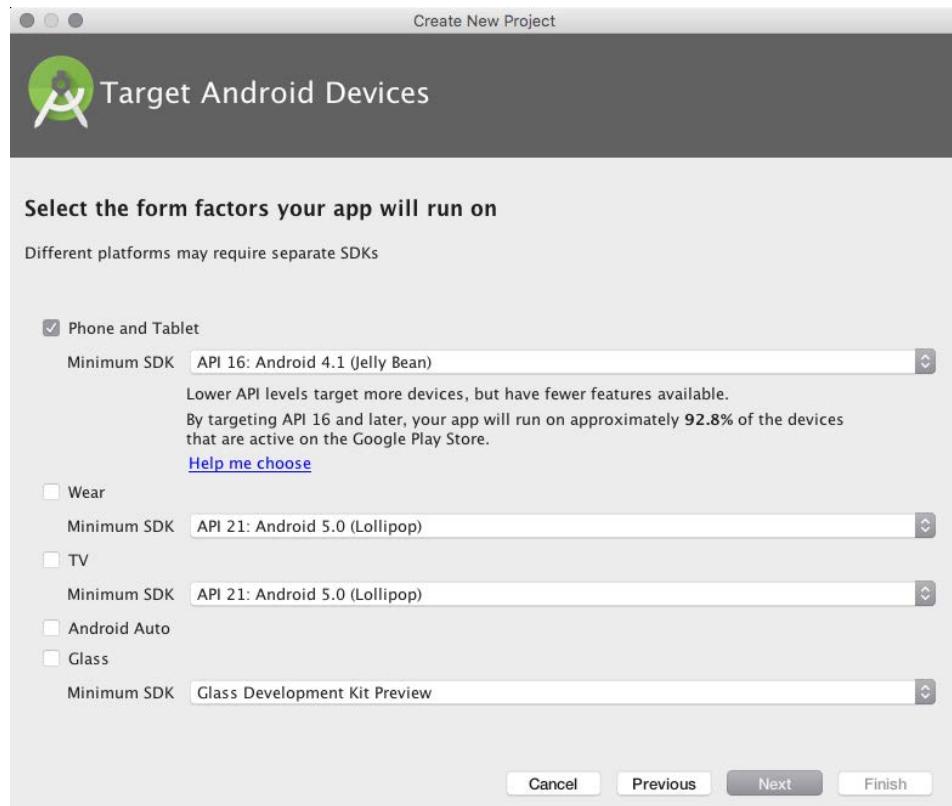


图1-5 设备支持配置

单击Next按钮继续。

在接下来的窗口中，需要为GeoQuiz应用的启动初始屏选择模板，如图1-6所示。选择Empty Activity后单击Next按钮继续。

(Android Studio更新频繁，因此新版本的向导画面看起来可能与本书所示略有不同。通常，这不是问题，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>，获取最新版本开发工具的使用信息。)

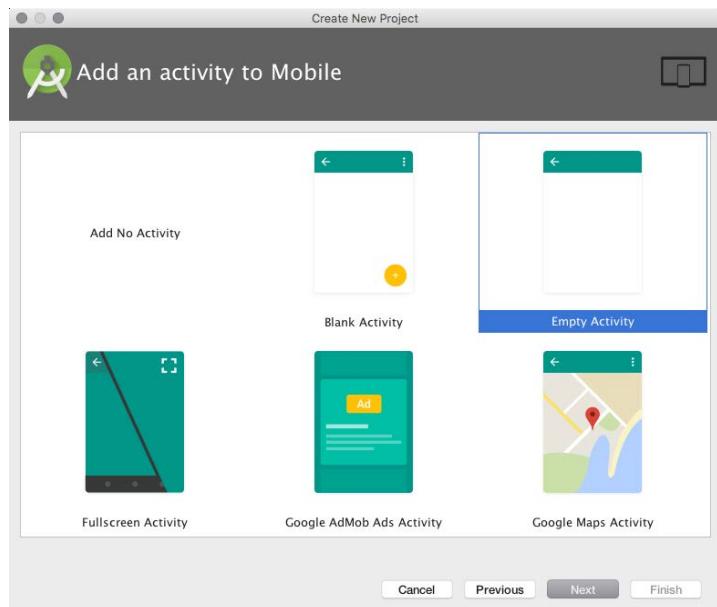


图1-6 选择activity种类（空activity）

在应用向导的最后一个窗口，命名activity子类为**QuizActivity**，如图1-7所示。注意子类名的Activity后缀。尽管不是必需的，但我们建议遵循这种规范的命名约定。

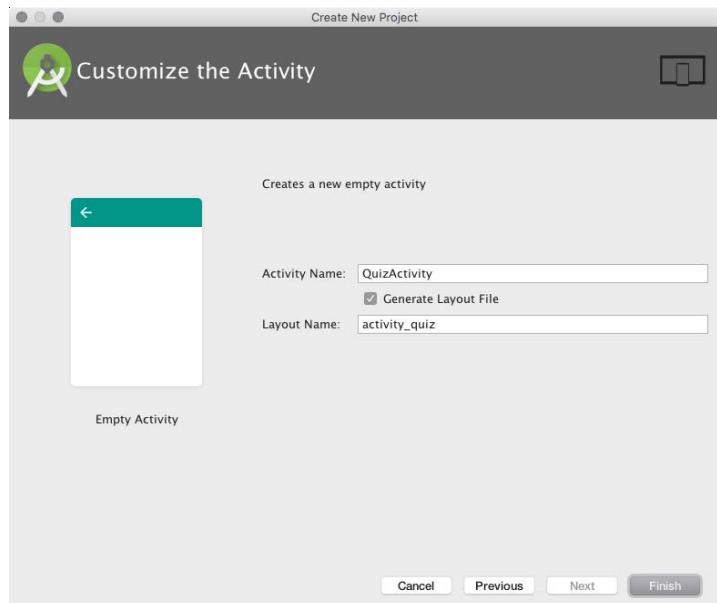


图1-7 配置新建的activity

保持Generate Layout File处于选中状态。为体现布局与activity间的对应关系，布局名（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。

如果在你的Android Studio版本中还有其他选项，保持默认选择不变。单击Finish按钮，Android Studio会完成创建并打开新的项目。

1.3 Android Studio 使用导航

如图1-8所示，Android Studio已在工作区窗口里打开新建项目。整个工作区窗口分为不同的区域，这里统称为工具窗口（Tool Window）。

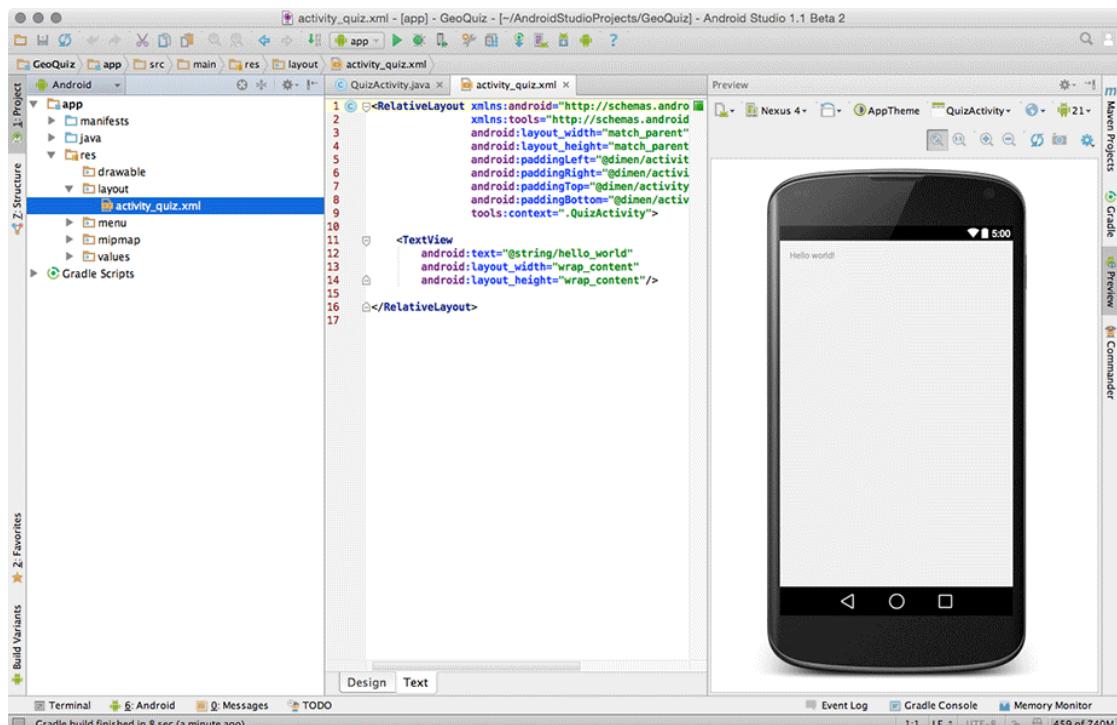


图1-8 新的项目窗口

左边是项目工具窗口（project tool window）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，Android Studio默认在代码编辑区打开了activity_quiz.xml文件。（如果你在代码区看到的是图片，可点击底部的Text页切换。）当然，你也可以设置在右边窗口预览当前编辑的文件。

点击工作区窗口左边、右边以及底部标有各种名称的工具按钮区域，可显示或隐藏各类工具窗口。当然，也可以直接使用它们对应的快捷键。假如看不到某个工具按钮的话，可以点击左下角的灰色方形区域或单击View → Tool Buttons菜单项找到它。

1.4 用户界面设计

当前，activity_quiz.xml文件定义了默认的activity布局。应用默认的XML布局文件内容经常改变，但总是与代码清单1-1所示文件相似。

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"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".QuizActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

应用activity的默认布局定义了两个组件（widget）：RelativeLayout和TextView。

组件是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。每一个组件都是View类或其子类（如TextView或Button）的一个具体实例。

图1-9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。

不过，图1-9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1-10展示了以上组件是如何构成QuizActivity用户界面的。

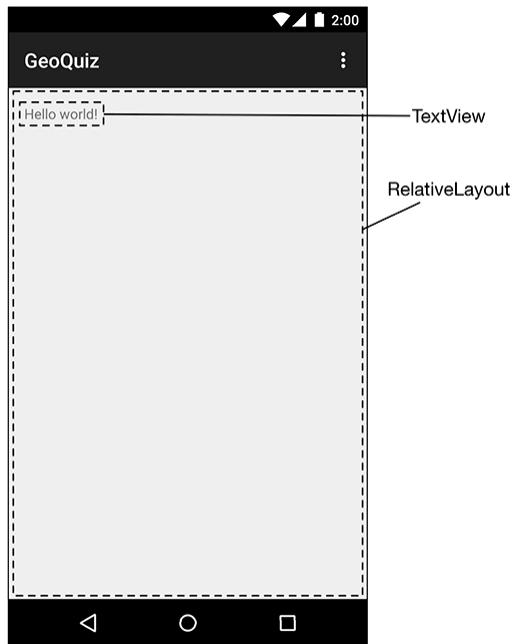


图1-9 显示在屏幕上的默认组件

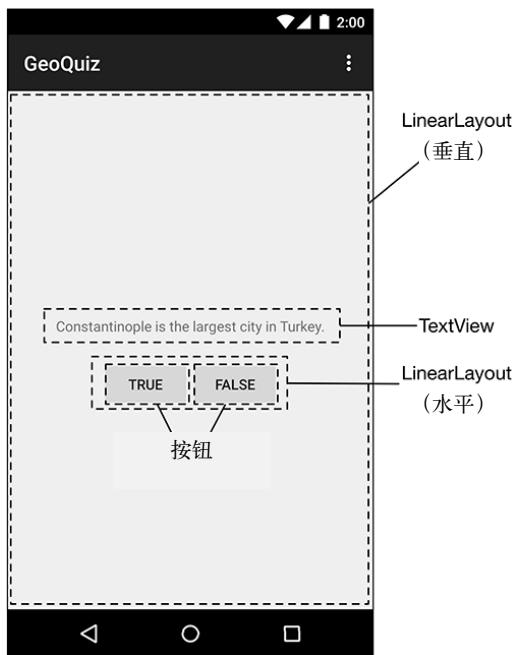


图1-10 布置并显示在屏幕上的组件

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"  
    android:paddingLeft="@dimen/activity_horizontal_margin"  
    android:paddingRight="@dimen/activity_horizontal_margin"  
    android:paddingTop="@dimen/activity_vertical_margin"  
    android:paddingBottom="@dimen/activity_vertical_margin"  
    tools:context=".QuizActivity">  
  
    <TextView  
        android:text="@string/hello_world"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content" />  
  
    </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，请尽量避免输入或拼写错误。

根据所使用的工具版本不同，有三行以`android:text`开头的代码可能会产生错误信息。暂时忽略它们，稍后会处理。

将XML文件与图1-10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。

为便于理解元素与属性的工作原理，接下来我们将以层次等级视角来研究布局。

1.4.1 视图层级结构

组件包含在视图对象的层级结构中，这种结构称作视图层级结构（view hierarchy）。图1-11展示了代码清单1-2所示XML布局对应的视图层级结构。

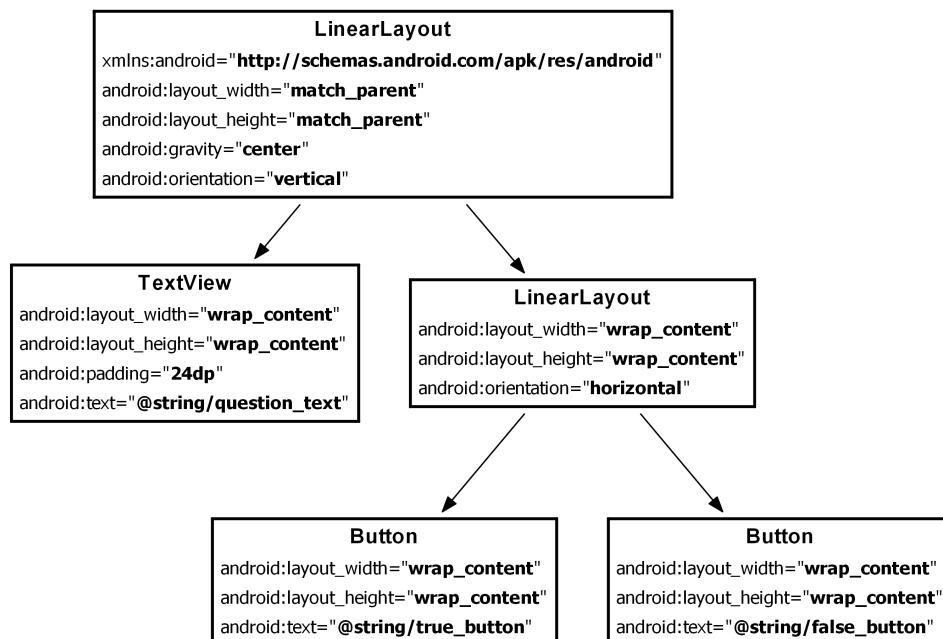


图1-11 布局中组件及属性的层级结构

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。以下是它们的两个常见属性值（二选一）。

□ match_parent：视图与其父视图大小相同。

□ wrap_content：视图将根据其展示的内容自动调整大小。

（以前还有个 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 1-10 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，指与密度无关像素，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，它决定两者的子组件是水平放置还是垂直放置。根 LinearLayout 是垂直的，子 LinearLayout 是水平的。

LinearLayout 子组件的定义顺序决定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。（如果设备文字从右至左显示，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。该属性指定组件要显示的文字内容。

请注意， android:text 属性值不是字符串值，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值，如 android:text="True"，但这通常不是个好主意。比较好的做法是：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。这样也便于对应用进行本地化。

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。

在项目工具窗口中，找到app/res/values目录，显示目录内容，然后打开strings.xml文件。

可以看到，项目模版已经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）

```
<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>
</resources>
```

（项目已默认配置好应用菜单，请勿删除action_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。）

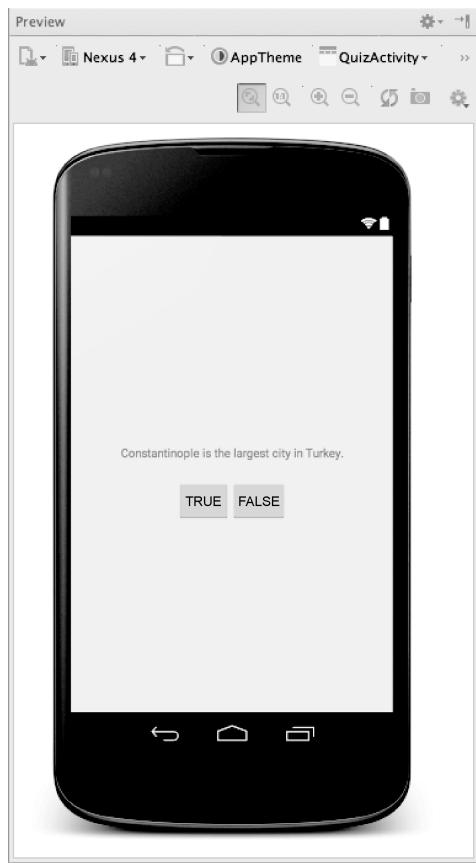
现在，在GeoQuiz项目的任何XML文件中，只要引用到@string/false_button，应用运行时，就会得到文本“FALSE”。

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。）

默认的字符串文件名为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，应用就能找到并正确使用它们。

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，点击编辑区右边的Tab页打开预览工具窗口（如果还没没打开的话），如图1-12所示。



1

图1-12 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，我们也创建了一个名为QuizActivity的Activity子类。QuizActivity类文件存放在项目的app/java目录下。java目录是项目全部Java源代码的存放处。

在项目工具窗口中，依次展开app/java目录与com.bignerdranch.android.geoquiz包，显示其中的内容。然后打开QuizActivity.java文件，查看其中的代码，如代码清单1-4所示。

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

```
package com.bignerdranch.android.geoquiz;  
  
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class QuizActivity extends AppCompatActivity {

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

}
```

(是不是不明白AppCompatActivity的作用？它实际就是一个Activity子类，主要是为Android旧版本系统提供兼容性支持。第13章会详细介绍AppCompatActivity。)

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⊕符号来显示它们。

该Java类文件包含一个Activity方法：onCreate(Bundle)。

(如果你的文件还包含onCreateOptionsMenu(Menu)和onOptionsItemSelected(MenuItem)方法，暂时不用理会。第13章会详细介绍它们。)

activity子类的实例创建后，onCreate(Bundle)方法会被调用。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)
```

根据传入的布局资源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app/res的子目录下。在项目工具窗口中可以看到，activity_quiz.xml布局资源文件存放在res/layout/目录下。包含字符串资源的strings文件存放在res/values/目录下。

可以使用资源ID在代码中获取相应的资源。activity_quiz.xml定义的布局的资源ID为R.layout.activity_quiz。

查看GeoQuiz应用的资源ID需要切换项目视图。Android Studio默认使用Android项目视图，如图1-13所示。为让开发者专注于最常用的文件和目录，默认视图隐藏了Android项目的真实文件目录结构。

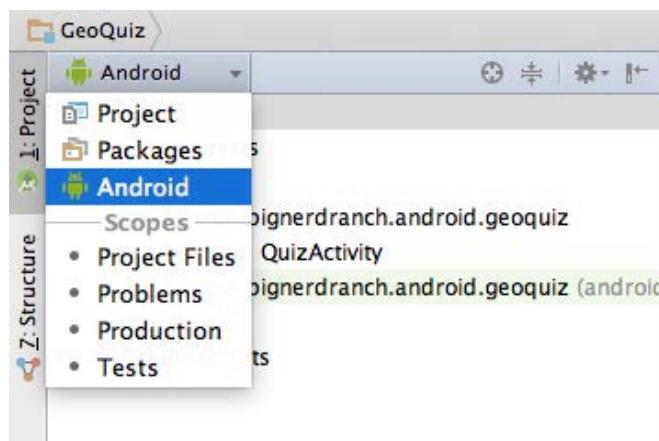


图1-13 切换项目视图

在项目工具窗口的最上部找到倒三角下拉菜单，从Android项目视图切换至Project视图。Project视图会显示出当前项目的所有文件和目录。

展开目录app/build/generated/source/r/debug，找到项目包名并打开其中的R.java文件，即可看到GeoQuiz应用当前所有的资源。R.java文件在Android项目编译过程中自动生成，如该文件头部的警示所述，请不要修改该文件的内容。

修改资源后，资源文件不会实时刷新。Android Studio另外还存有一份代码编译用的R.java隐藏文件。当前编辑区打开的R.java文件仅在应用安装至设备或模拟器前产生，因此只有在Android Studio中点击运行应用时，它才会得到更新。

R.java文件通常比较大，代码清单1-5仅展示了部分内容。

代码清单1-5 GeoQuiz应用当前的资源ID（R.java）

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

public final class R {
    public static final class anim {
        ...
    }
    ...

    public static final class id {
        ...
    }
}
```

```

public static final class layout {
    ...
    public static final int activity_quiz=0x7f030017;
}
public static final class mipmap {
    public static final int ic_launcher=0x7f030000;
}
public static final class string {
    ...
    public static final int app_name=0x7f0a0010;
    public static final int correct_toast=0x7f0a0011;
    public static final int false_button=0x7f0a0012;
    public static final int incorrect_toast=0x7f0a0013;
    public static final int question_text=0x7f0a0014;
    public static final int true_button=0x7f0a0015;
}
}

```

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。

应用需要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，如果需要，可以使用以下方法：

```
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。

本书主要使用Android项目视图，生成资源ID之前，记得检查当前视图模式。当然，如果你更喜欢Project视图，也不会有什么问题。

要为组件生成资源ID，请在定义组件时为其添加`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。这是因为我们在创建资源ID，而对字符串资源只是做引用。

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1-7所示代码。（请勿使用代码自动补全功能。）

代码清单1-7 添加成员变量（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

文件保存后，会看到两个错误提示。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编程应遵循的命名约定，本书将始终遵循该约定。

现在，在将鼠标移至代码左边的错误提示处时，会看到两条同样的错误信息：Cannot resolve symbol ‘Button’。

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;
```

或者使用Option+Return（或Alt+Enter）组合键，让Android Studio自动为你导入。代码有误时，也可以使用该组合键来修正。记得要常用。

类包导入完成后，刚才的错误提示应该就消失了。（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。

1.6.1 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)
```

该方法以组件的资源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，赋值给对应的成员变量，如代码清单1-8所示。注意，赋值前，必须先将返回的View转型（cast）为Button。

代码清单1-8 引用组件（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

    @Override
    protected 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);
    }
}
```

1.6.2 设置监听器

Android应用属于典型的事件驱动类型。不像命令行或脚本程序，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直观。）

应用等待某个特定事件的发生，也可以说应用正在“监听”特定事件。为响应某个事件而创建的对象叫作监听器（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onCreate(...)方法的变量赋值语句后输入下列代码，如代码清单1-9所示。

代码清单1-9 为TRUE按钮设置监听器（QuizActivity.java）

```
...
@Override
protected 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Option+Return或Alt+Enter快捷键导入View类。）

在代码清单1-9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。传入setOnClickListener(OnClickListener)方法的参数是一个监听器。该参数是一个实现了OnClickListener接口的对象。

使用匿名内部类

传入SetOnClickListener(OnClickListener)方法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，不过有个助记小技巧：最外层括号内的全部实现代码就是传入SetOnClickListener(OnClickListener)方法内的一个参数。该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以匿名内部类来实现。这样做有两大好处。第一，因为匿名内部类的使用，我们可在同一处实现监听器方法，代码更清晰可读；第二，事件监听器一般只在同一处使用，使用匿名内部类可避免不必要的命名类实现。

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。虽然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1-10为FALSE按钮设置类似的事件监听器。

代码清单1-10 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!
        }
    });
}
```

1.7 创建提示消息

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是用来通知用户的简短弹出消息，用户无需输入或进行任何操作。这里，我们要用toast来告知用户其答案正确与否，如图1-14所示。

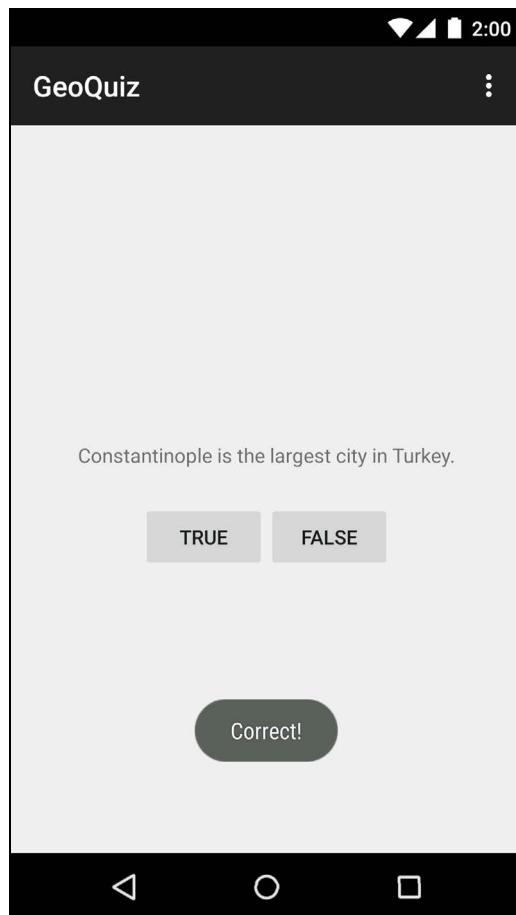


图1-14 toast反馈消息提示

首先回到strings.xml文件，如代码清单1-11所示，为toast添加消息显示用的字符串资源。

代码清单1-11 增加toast字符串（strings.xml）

```
<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>
</resources>
```

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要显示字符串消息的资源ID。Toast类必须借助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，可调用Toast.show()方法在屏幕上显示toast消息。

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1-12所示。在添加makeText(...)时，可利用Android Studio的代码自动补全功能，让代码输入更加轻松。

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1-12，依次输入代码。当输入到Toast类后的点号时，Android Studio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。

要选择需要的建议方法，使用上下方向键。（如果不想使用代码自动补全功能，请不要按Tab键、Return/Enter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 context, int resID, int duration)方法，代码自动补全功能会自动添加完整的方法调用。

完成makeText方法的全部参数设置，完成后的代码如代码清单1-12所示。

代码清单1-12 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();
        // Does nothing yet, but soon!
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                      R.string.correct_toast,
                      Toast.LENGTH_SHORT).show();
        // Does nothing yet, but soon!
    }
});
```

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QuizActivity.this，不要想当然地直接输入this。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，再也不用担心某个类忘记导入了，因为Android Studio会自动导入

所需的类。

现在保存，然后看看这个应用的运行情况。

1.8 使用模拟器运行应用

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。

要创建Android虚拟设备 (AVD)，在Android Studio中，选择Tools → Android → AVD Manager菜单项。AVD管理器窗口弹出时，点击窗口左边的Create Virtual Device...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Nexus 5设备，如图1-15所示。点击Next继续。

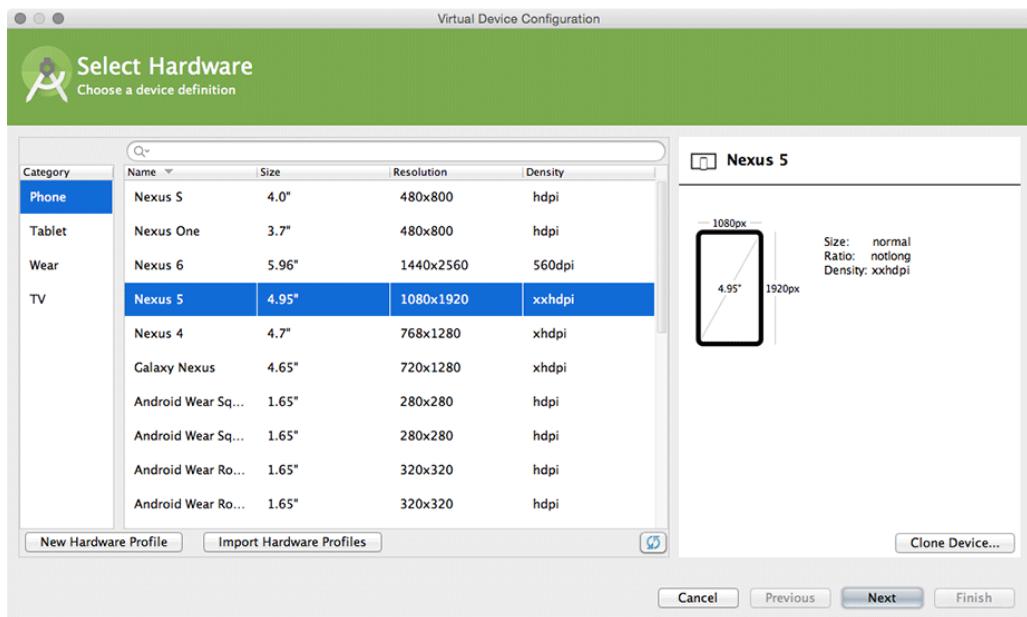


图1-15 创建新的AVD

如图1-16所示，接下来选择模拟器的系统镜像。选择x86 Lollipop模拟器后点击Next按钮继续。

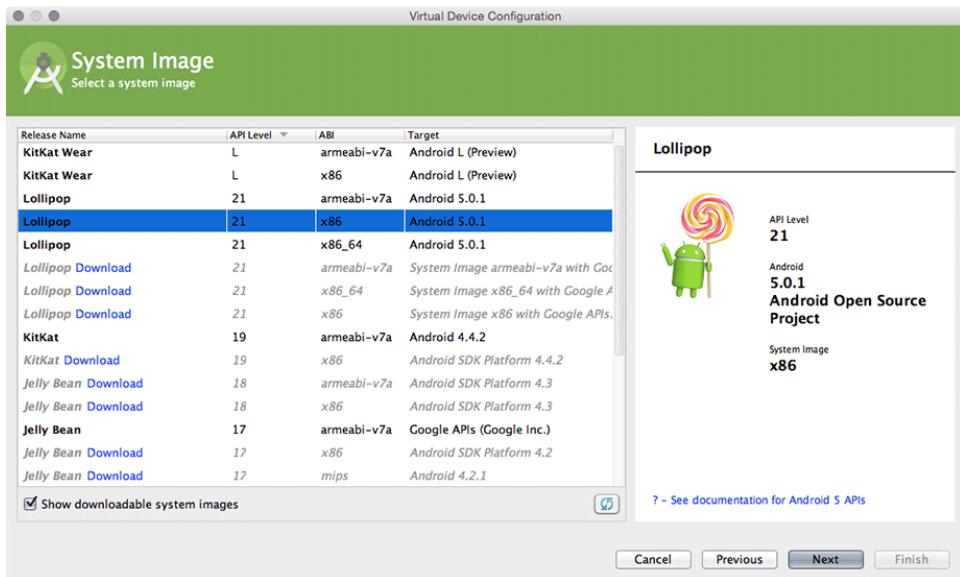


图1-16 选择系统镜像

最后，可以对模拟器的各项参数做最后修改并确认，如图1-17所示。当然，如有需要，也可以事后再编辑修改模拟器的各项参数。现在，为模拟器取个便于识别的名称，点击Finish按钮完成虚拟设备的创建。

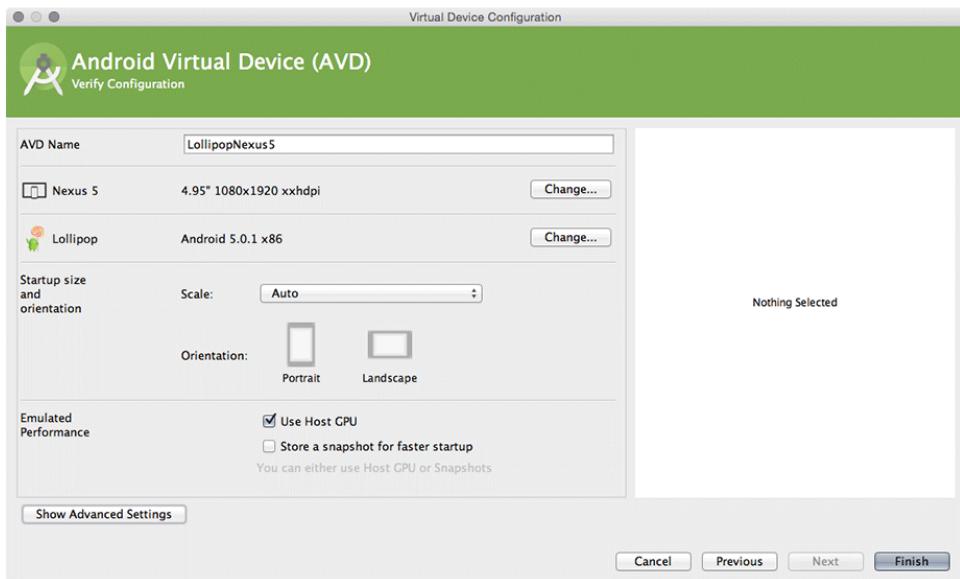
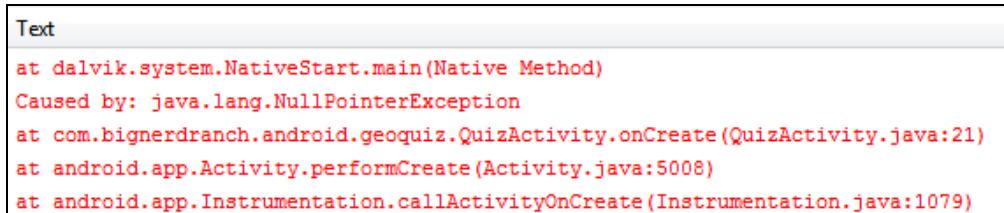


图1-17 模拟参数调整

AVD创建成功后，我们用它运行GeoQuiz应用。点击Android Studio工具栏上的Run按钮，或者使用Control+R快捷键。Android Studio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。

虚拟机启动过程非常缓慢，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。跟真实设备一样，AVD闲置一定时间会自动锁上。）

假如GeoQuiz应用启动时或在点击按钮时发生崩溃，可以在Android DDMS工具窗口的LogCat视图中看到有用的诊断信息。（如果LogCat没有自动打开，可点击Android Studio窗口底部的Android按钮打开它。）查看日志，可看到抢眼的红色异常信息，如图1-18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1-18 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，尝试重新运行应用（在接下来的两章，还会深入介绍LogCat和代码调整的知识）。

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AVD模拟器上的Android后退按钮可以停止应用。这个后退按钮的形状像一个指向左侧的三角形（在较早版本的Android中，它像一个U型箭头）。需要调试变更时，再通过Android Studio重新运行应用。

模拟器虽然有用，但在真实设备上测试应用能获得更准确的结果。在第2章中，我们会在实体设备上运行GeoQuiz应用，还会为GeoQuiz应用添加更多挑战用户的地理知识问题。

1.9 深入学习：Android 编译过程

学习到这里，你可能对Android编译的工作方式充满疑惑。你已经知道在项目文件发生变化时，无需使用命令行工具，Android Studio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。.apk应用要在模拟器上运行，.apk文件还需以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1-19展示了完整的编译过程。

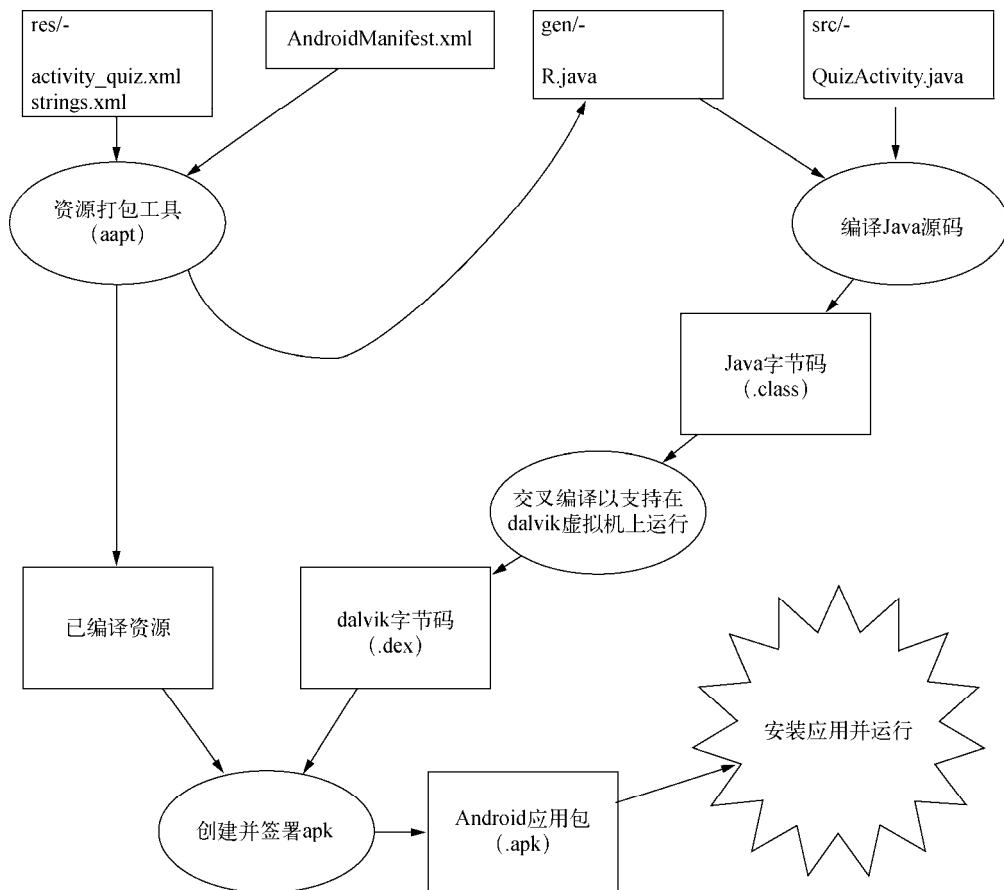
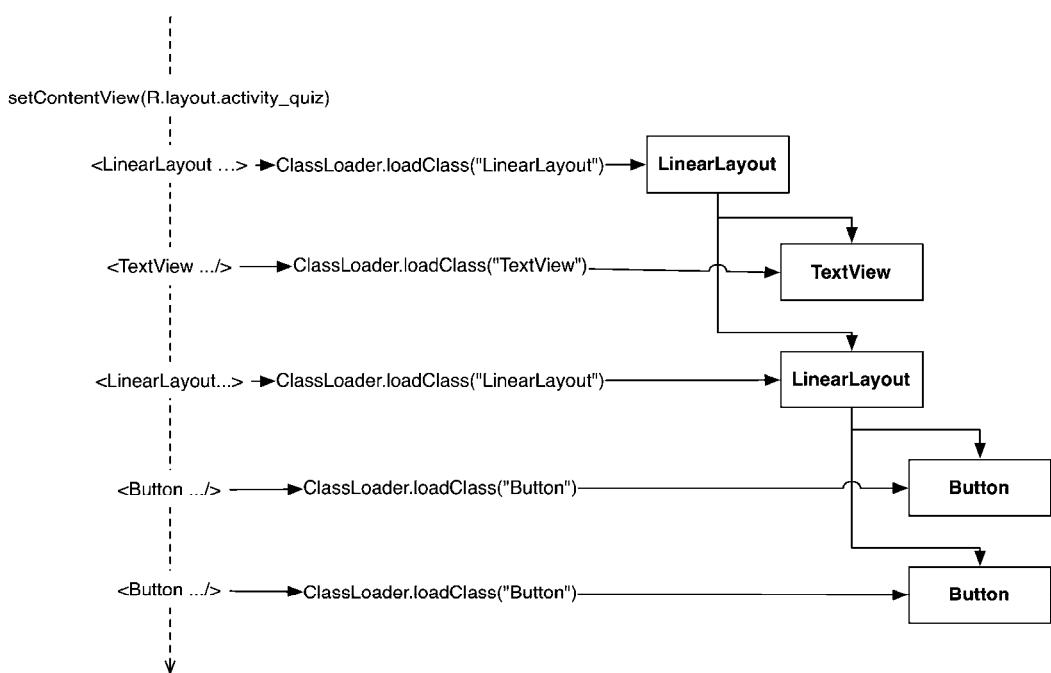


图1-19 编译GeoQuiz应用

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。然后，在QuizActivity类的onCreate(...)方法调用setContentView(...)方法时，QuizActivity使用LayoutInflater类实例化布局文件中定义的每一个View对象，如图1-20所示。

图1-20 `activity_quiz.xml`中的视图实例化

(除了在XML文件中定义视图的方式外，也可以在activity里使用代码来创建视图类。不过，从设计角度来看，应用展现层与逻辑层分离好处多多，其中最主要的一点是可以利用SDK内置的设备配置改变，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如何显示在屏幕上等更多信息，请参见第8章。

Android 编译工具

截至目前，我们所看到的项目编译都是在Android Studio里执行的。编译功能已整合到IDE中，IDE负责调用aapt等Android标准编译工具，但编译过程本身仍由Android Studio管理。

有时，出于种种原因，可能需要脱离Android Studio编译代码。最简单的方法是使用命令行编译工具。现代Android编译系统使用Gradle编译工具。

(注意，能读懂本小节内容并按步骤操作那是最好了。如果看不懂，甚至不知道为什么要手工编译代码，或者是无法正确使用命令行，也不必太在意，请继续学习下一章内容。如何使用命令工具以及为什么要使用命令行工具，不在本书的讨论范围之内。)

要使用Gradle，请切换到项目目录并执行以下命令：

```
$ ./gradlew tasks
```

如果是Windows系统，执行以下命令：

```
> gradlew.bat tasks
```

执行以上命令会显示一系列可用任务。你需要的是任务是installDebug，因此，再执行以下命令：

```
$ ./gradlew installDebug
```

如果是Windows系统，执行以下命令。

```
> gradlew.bat installDebug
```

以上命令将把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。

本章我们将升级GeoQuiz应用，提供更多的地理知识测试题目，如图2-1所示。

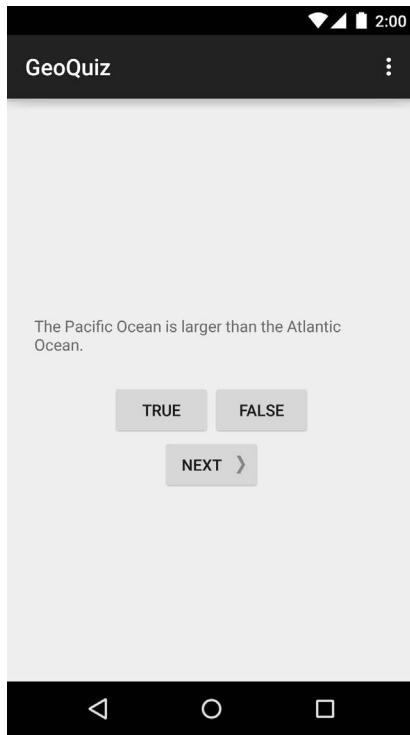


图2-1 更多测试题目

为实现目标，需要为GeoQuiz项目新增一个**Question**类。该类的一个实例代表一道题目。然后再创建一个**Question**数组对象交由**QuizActivity**管理。

2.1 创建新类

在项目工具窗口中，右键单击**com.bignerdranch.android.geoquiz**类包，选择New→Java

Class菜单项。如图2-2所示，类名处填入Question，然后单击OK按钮。

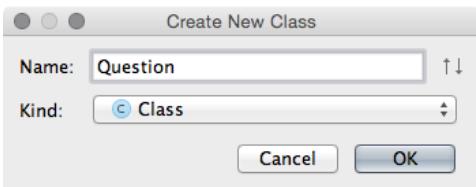


图2-2 创建Question类

在Question.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 Question类中的新增代码（Question.java）

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;

    public Question(int textResId, boolean answerTrue) {
        mTextResId = textResId;
        mAnswerTrue = answerTrue;
    }
}
```

Question类中封装的数据有两部分：问题文本和问题答案（true或false）。

mTextResId为什么是int类型，而不是String类型呢？变量mTextResId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。需要用到的问题字符串资源稍后会处理。

新增的两个变量需要获取方法与设置方法。为避免手工输入，可设置由Android Studio自动生成。

生成获取方法与设置方法

首先，配置Android Studio识别成员变量的m前缀。

打开Android Studio首选项对话框（Mac用户选择Android Studio菜单，Windows用户选择File→Settings菜单）。分别展开Editor和Code Style选项，在Java选项下选择Code Generation选项页。

在Naming表单中，选择Fields行，添加m作为fields的前缀，如图2-3所示。然后添加s作为Static Fields的前缀。（GeoQuiz项目不会用到s前缀，但之后的项目会用到。）

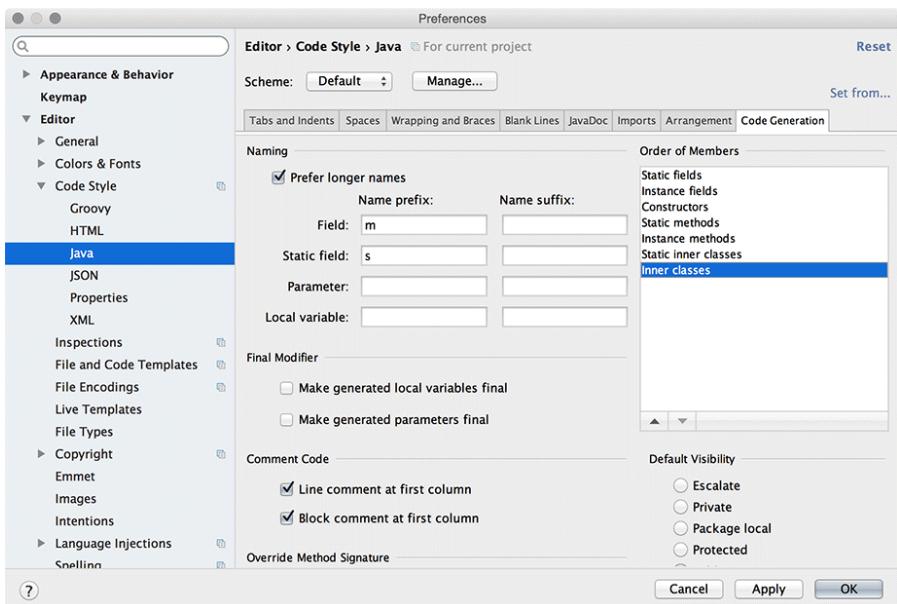


图2-3 设置Java代码风格首选项

单击OK按钮完成。

刚才设置的前缀有何作用？那就是，需要Android Studio为`mTextResId`生成获取方法时，它生成的是`getTextResId()`而不是`getMTextResId()`方法；而在为`mAnswerTrue`生成获取方法时，生成的是`isAnswerTrue()`而不是`isMAnswerTrue()`方法。

回到`Question.java`中，右击构造方法后方区域，选择`Generate... → Getter And Setter`菜单项。选择`mTextResId`和`mAnswerTrue`，为每个变量都生成获取方法与设置方法。

单击OK按钮，Android Studio随即生成了获取方法与设置方法共4个方法的代码，如代码清单2-2所示。

代码清单2-2 生成获取方法与设置方法（`Question.java`）

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;

    ...

    public int getTextResId() {
        return mTextResId;
    }

    public void setTextResId(int textResId) {
        mTextResId = textResId;
    }
}
```

```

public boolean isAnswerTrue() {
    return mAnswerTrue;
}

public void setAnswerTrue(boolean answerTrue) {
    mAnswerTrue = answerTrue;
}
}

```

这样Question类就完成了。稍后，我们会修改QuizActivity类来配合Question类使用。现在，先整体了解一下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Question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，并根据用户的回答作出反馈，如图2-4所示。

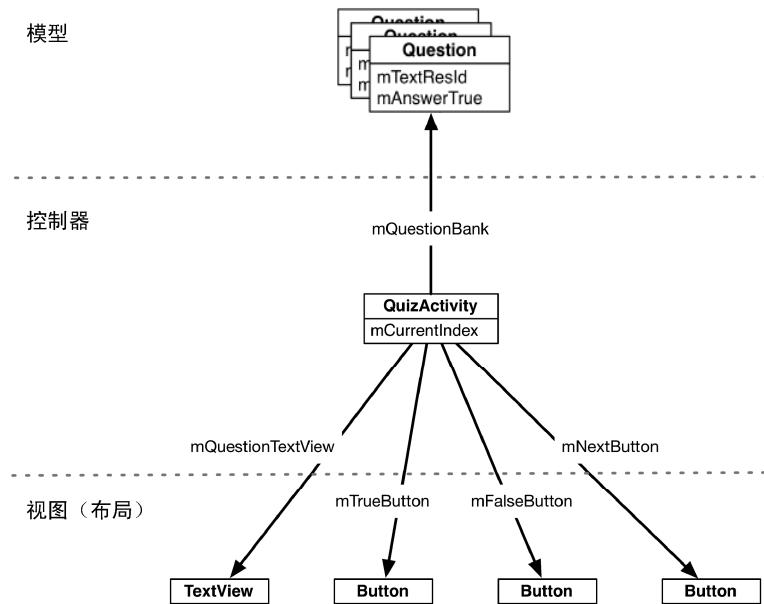


图2-4 GeoQuiz应用对象图解

2.2 Android与MVC设计模式

如图2-4所示，应用对象按模型、控制器和视图的类别分为三部分。Android应用基于模型—控制器—视图（Model-View-Controller，MVC）的架构模式进行设计。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**Question**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。

- 控制对象含有应用的逻辑单元，是视图与模型对象的联系纽带。控制对象响应视图对象触发的各类事件，此外还管理着模型对象与视图间的数据流动。

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26章将分别介绍**fragment**和**service**的概念）。

GeoQuiz应用的控制层仅由**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。控制器作为它们之间的联系纽带，接收对象发送的消息，然后向其他对象发送操作指令。

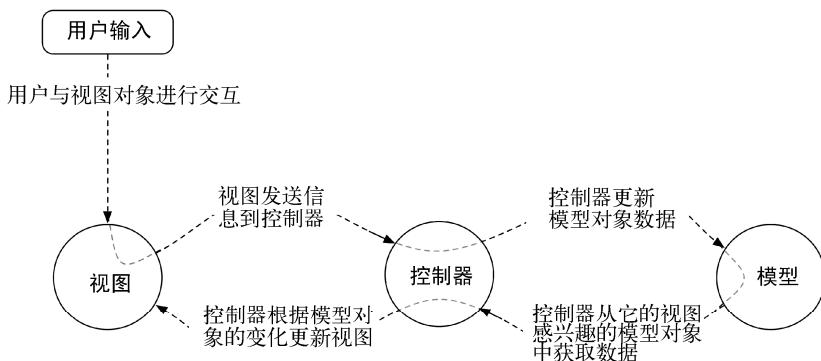


图2-5 MVC数据控制流与用户交互

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类组织代码有助于从整体视角设计和理解应用。这样，我们就可以按类而不是按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应用不复杂，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，为它添加一个NEXT按钮。我们会发现，在添加NEXT按钮的过程中，可以完全不用考虑刚才创建的Question类。

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，功能单一的专用类更加有利于代码复用。

举例来说，模型类Question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使用Question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Question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，不作改动的组件属性这里就不再列出了。）

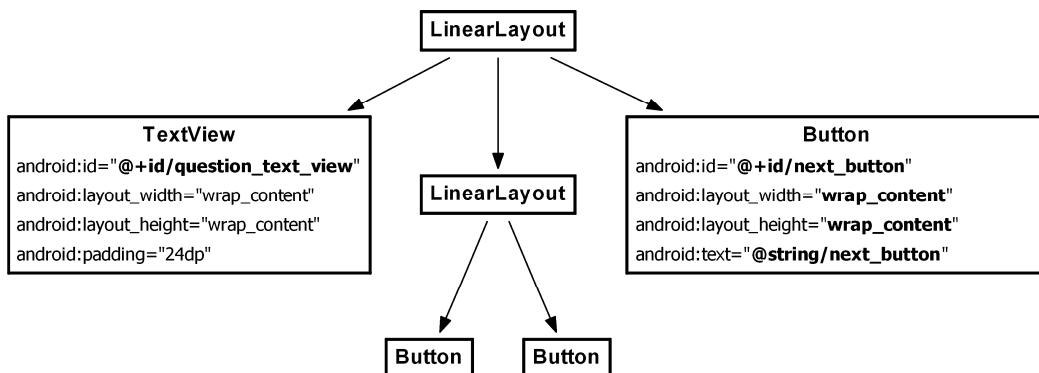


图2-6 新增的按钮

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地理知识问题。
- 为TextView新增android:id属性。TextView组件需要资源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
        ...
    <Button
        ...
    <Button
        ...
    
```

```

    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，可能会看到一个错误弹框提示，提醒我们缺少字符串资源。返回到res/values/strings.xml文件中。删除硬编码的地理知识问题字符串，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文件的打开状态，继续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="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，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，QuizActivity类就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Button变量。另外，再创建一个Question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Question对象数组（QuizActivity.java）

```

public class QuizActivity extends AppCompatActivity {

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

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

    private int mCurrentIndex = 0;
    ...
}

```

这里，我们通过多次调用Question类的构造方法，创建了Question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，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。)

要把一系列地理知识问题显示在屏幕上，可以使用mQuestionBank数组、mCurrentIndex

变量以及**Question**对象的存取方法。

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地理知识问题，如代码清单2-7所示。

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

```
public class QuizActivity extends AppCompatActivity {
    ...
    @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].getTextResId();
        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 AppCompatActivity {
    ...
    @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].getTextResId();
        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].getTextResId();
        mQuestionTextView.setText(question);
    }
});
...
}
}

```

可以看到，用来更新mQuestionTextView变量的相同代码分布在两个不同的地方。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，以初步设置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 AppCompatActivity {

    ...

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

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

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

        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getTextResId();
                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。当前，GeoQuiz应用认为所有问题的答案都是false，下面着手修正这个逻辑错误。同样，为避免代码重复，我们将解决方案封装在一个私有方法里。

要添加到QuizActivity类的方法如下：

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

该方法接受boolean类型的变量参数，判别用户单击了TRUE还是FALSE按钮。然后，将用户的答案同当前Question对象中的答案作比较。最后，判断答案正确与否，生成一个Toast消息反馈给用户。

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 AppCompatActivity {

    ...

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

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

        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 AppCompatActivity {  
    ...  
    @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应用已准备就绪，可以在设备上运行了。

2.5 在设备上运行应用

本节，我们学习如何设置系统、设备和应用，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，请到设备生产商的网站下载。

2.5.2 配置设备用于应用开发

要在设备上测试应用，首先应打开设备的USB调试模式。

- Android 4.2或之后版本的设备，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（Build Number）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。
- 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 4.0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。

从以上操作可以看出，不同版本设备的设置有较大差异。如在设置过程中遇到问题，请访问<http://developer.android.com/tools/device.html>寻求帮助。

可打开Devices视图来确认设备是否得到识别。选择Android Studio底部的Android工具窗口，可快速打开Devices视图。如图2-7所示，你会看到已连接设备的下拉列表。AVD以及硬件设备应该已经列在其中了。

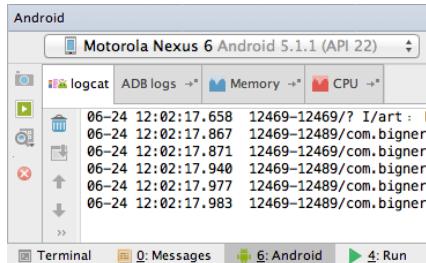


图2-7 查看已连接设备

如果遇到设备无法识别的问题，请首先确认是否已打开设备和开发者项。如果仍然无法解决，请访问Android开发网站<http://developer.android.com/tools/device.html>，或访问本书论坛<http://forums.bignerdranch.com>寻求帮助。

再次运行GeoQuiz应用，Android Studio会询问是在虚拟设备上还是在物理设备上运行应用。选择物理设备并继续。稍等片刻，GeoQuiz应用应该已经在设备上运行了。

如果Android Studio没有提供选择，应用依然在虚拟设备上运行，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。然后，再检查运行配置是否有问题。要修改运行配置，请选择Android Studio窗口靠近顶部的app下拉列表，如图2-8所示。



图2-8 打开运行配置

选择Edit Configurations打开运行配置编辑窗口，如图2-9所示。选择窗口左边区域的app，确认已打开Target Device区域的Show chooser dialog选项。点击OK按钮并重新运行该应用，现在你会看到可以运行应用的设备选项。

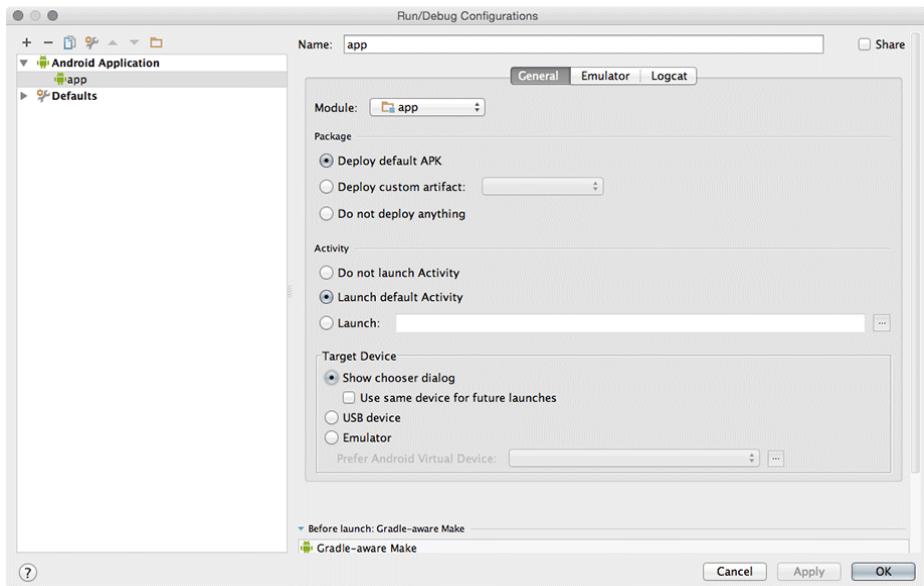


图2-9 运行配置界面

2.6 添加图标资源

GeoQuiz应用现在已经可用。如果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。

本书随书文件中提供了这样的箭头图标（<https://www.bignerdranch.com/solutions/Android Programming2e.zip>）。随书文件集合了Android Studio项目文件，每章对应一个项目文件。

通过以上链接下载文件后，找到并打开02_MVC/GeoQuiz/app/src/main/res目录。在该目录下，可以找到drawable-hdpi、drawable-mdpi、drawable-xhdpi和drawable-xxhdpi四个目录。

四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）。
- xxhdpi：超超高像素密度屏幕（约480dpi）。

（另外还有ldpi和xxxhdpi这两个本书用不到的类别，因此，代码文件省略了他们。）

每个目录下，可看到两个图片文件：arrow_right.png和arrow_left.png。这些图片文件都是按

照目录名对应的dpi进行定制的。

GeoQuiz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。注意，在为不同设备准备适配图片的同时，应用安装包需要的容量也随之增大。当然，对于GeoQuiz这样的小项目，问题并不明显。

如果应用不包含设备对应的屏幕像素密度文件，在运行时，Android系统会自动找到可用的图片资源，针对该设备进行适配。有了这种特性，就不必准备各种屏幕像素密度文件了。因此，为控制应用包的大小，我们可以只为主流设备准备分辨率较高的定制图片资源。至于那些不常见的低分辨率设备，让Android系统自动适配就好。

(在第21章，我们会介绍为屏幕像素密度定制图片的替代方案。另外，还会解释mipmap目录的用途。)

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。

首先，确认已准备好了需要的drawable目录。再确认打开了Android Studio的Project视图。展开GeoQuiz/app/src/main/res目录，找到匹配各类像素密度的子目录，如图2-10所示。(你还有可能看到其他的子目录，目前暂时忽略。)

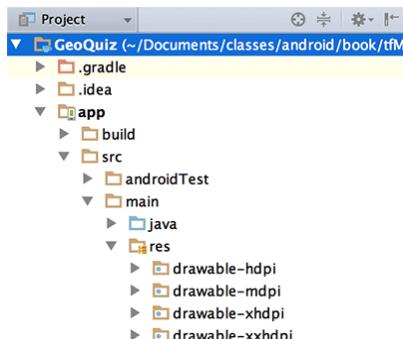


图2-10 确认准备好了各类drawable目录

如果要用到的drawable目录不存在，首先需要创建它们。右键单击res目录，选择New → Directory，会弹出如图2-11所示的对话框。输入需要的目录名，如drawable-mdpi，然后点击OK按钮确认。



图2-11 创建drawable目录

成功创建drawable-mdpi目录后，就应该能在Project视图中看到它。如果看不到，很可能你当前打开的还是Android视图，请自行切换至Project视图。

如有需要，重复上述步骤，完成创建drawable-hdpi、drawable-xhdpi和drawable-xxhdpi目录。

然后将已下载文件目录中对应的图片文件（arrow_left.png和arrow_right.png）复制到项目的对应drawable目录中。

完成复制后，在Android Studio的项目工具窗口就可以看到新的arrow_left.png和arrow_right.png文件了，如图2-12所示。

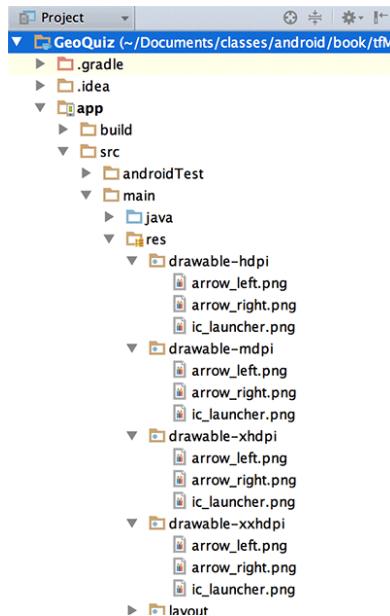


图2-12 drawable目录中的箭头图标

如果将项目工具窗口切换回Android视图，新增加的drawable图片资源会以图2-13所示的形式展示。

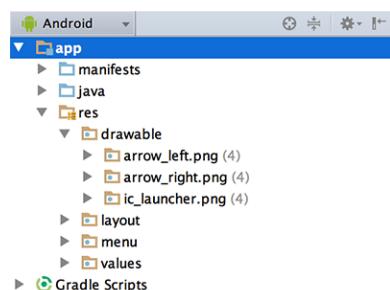


图2-13 drawable目录中的箭头图标汇总

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的文件都会自动获得资源ID。(注意，文件名必须是小写字母且不能有任何空格符号。)

这些资源ID并不按照屏幕密度匹配。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。

Android资源系统是如何运作的？从第3章起，我们会深入学习这方面的相关知识。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？

很简单，只是语法稍有不同而已。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_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GeoQuiz应用。新按钮很漂亮吧？测试一下，确认它仍然工作正常。

别高兴太早，GeoQuiz应用有个bug。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Fn+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章会进行专题介绍。

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，需要较强的问题解决能力。

希望你一定完成这些练习。攻克它们不仅可以巩固所学，树立信心，还可以让自己从被动学习快速成长为自主开发的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，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其他读者一起交流学习。

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

在你的电脑上，通过文件浏览器找到项目文件的根目录，复制一份GeoQuiz文件并重命名为GeoQuiz Challenge。回到Android Studio中，选择File→Import Project...菜单项，通过导入功能找到GeoQuiz Challenge并导入。这样，复制项目就在新窗口中打开了。开始挑战吧！

2.8 挑战练习：为 TextView 添加监听器

NEXT按钮不错，但如果用户单击应用的TextView文字区域（地理知识问题），也可以跳转到下一道题，用户体验应该会更好。你来试一试。

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

2.9 挑战练习：添加后退按钮

为GeoQuiz应用新增后退按钮，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2-14所示。

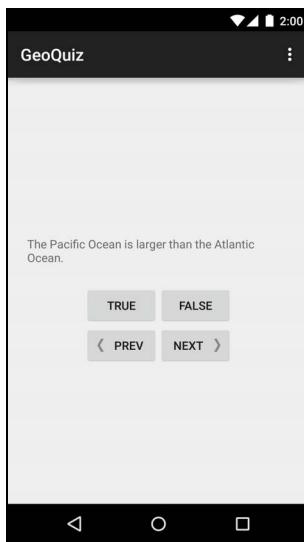


图2-14 添加了后退按钮的用户界面

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：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更加简洁美观。只显示图标按钮的用户界面如图2-15所示。

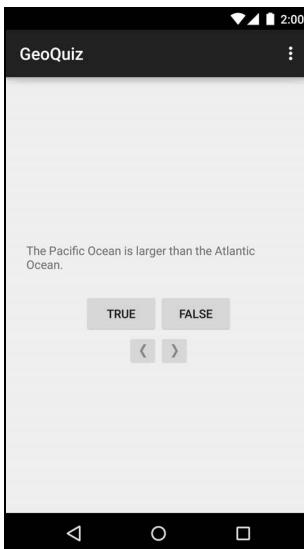


图2-15 只显示图标的按钮

完成此练习，需将普通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2-16所示。

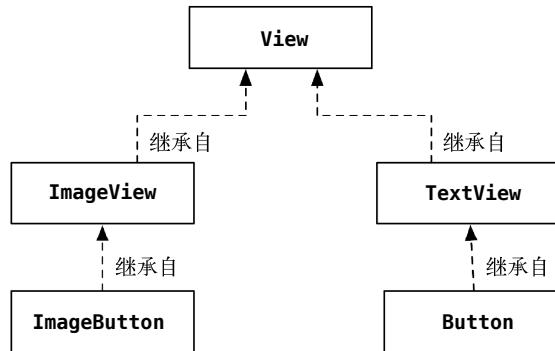


图2-16 ImageButton和Button与View间的继承关系

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

当然，为了使用ImageButton，记得调整QuizActivity类代码。

将按钮组件替换成ImageButton后，Android Studio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，那么在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，都有对应的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。

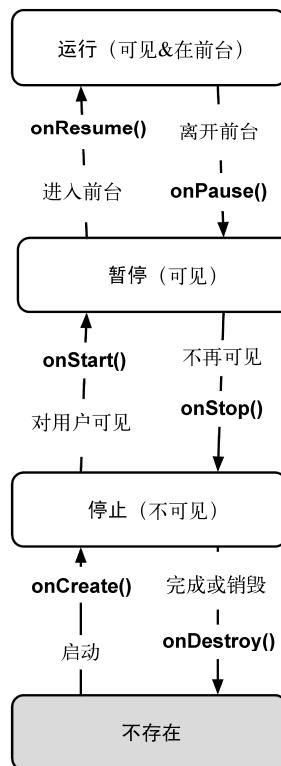


图3-1 Activity的状态图解

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之前，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相关的工作：

- 实例化组件并将组件放置在屏幕上（调用`setContentView(int)`方法）；
- 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。

千万不要自己去调用`onCreate(...)`方法或任何其他Activity生命周期方法，记住这一点很重要。我们只需在activity子类里覆盖这些方法，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。借助各个覆盖方法的日志输出，我们可知道操作系统何时调用了它们。

3.1.1 输出日志信息

Android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。（本章最后一节会详细讲解有关`Log`级别的内容。）第一个参数表示日志的来源，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 AppCompatActivity {
    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 AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```

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

...
}

}

```

接下来，在QuizActivity类中，覆盖其他五个生命周期方法，方法是将代码清单3-3所示代码添加至onCreate(Bundle)之后。

代码清单3-3 覆盖更多生命周期方法 (QuizActivity.java)

```

@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，然后再调用具体的日志记录方法。这些超类方法的调用不可或缺。另外，在onCreate(...)方法里，必须首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。而在其他几个方法中，是否首先调用超类方法就不那么重要了。

知道为什么要使用@Override注解吗？使用@Override注解，就是要求编译器保证当前类具有你要覆盖的方法。例如，对于如下拼写错误的方法，编译器将发出警告：

```

public class QuizActivity extends AppCompatActivity {

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

    ...
}

```

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工具中的一款日志查看器。

应用运行时，LogCat应该已经出现在Android Studio底部了，如图3-2所示。如果看不到，请切换至Android工具窗口模式，并确保已选中Devices | logcat选项页。

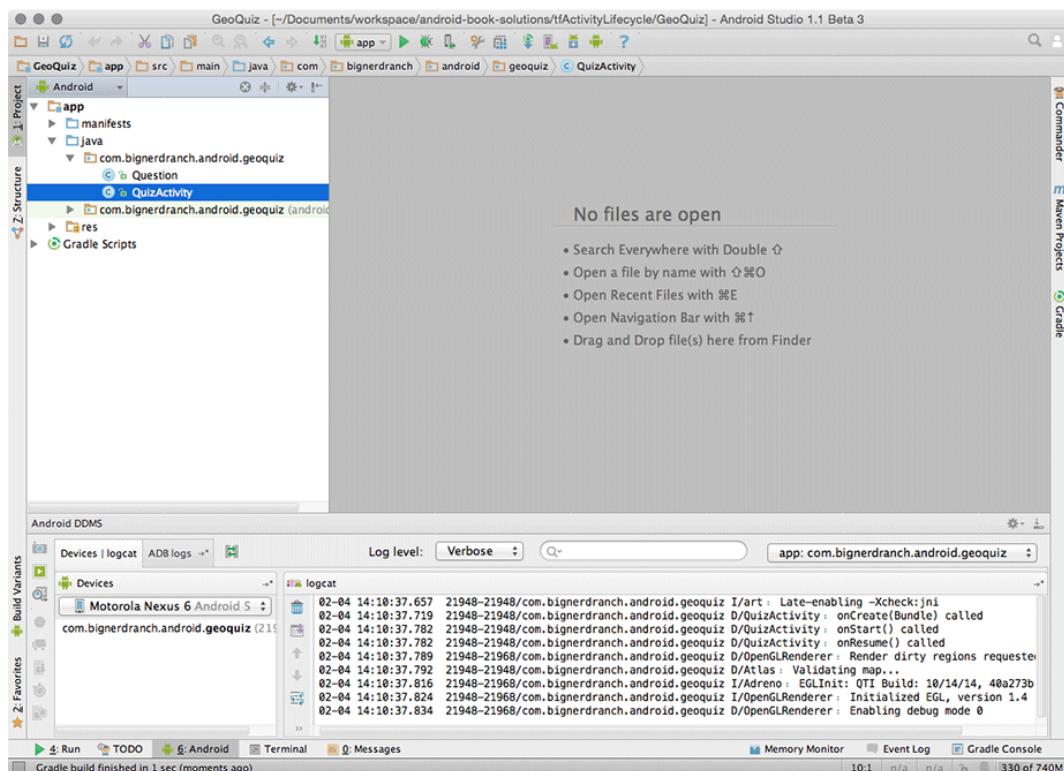


图3-2 Android Studio中的LogCat

运行GeoQuiz应用。立刻可看到出现在LogCat窗口中的各类混杂信息。这些日志中，有些是以应用包名为默认名的应用类信息，有些是系统输出信息。

为方便查找，可使用TAG常量过滤日志输出。单击LogCat面板右上角的过滤器下拉列表，可看到现有的过滤项。其中，No Filters选项控制只显示系统输出信息。

要创建过滤设置，选择Edit Filter Configuration选项。单击绿色+按钮，创建一个消息过滤器。在Name处输入QuizActivity，by Log Tag处同样输入QuizActivity，如图3-3所示。

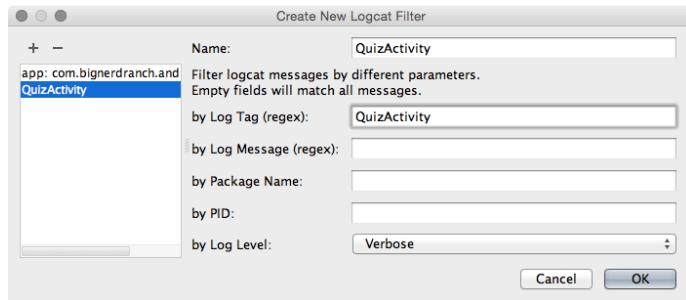


图3-3 在LogCat中创建过滤器

单击OK按钮。现在，LogCat窗口中仅显示了Tag为QuizActivity的日志信息，如图3-4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。

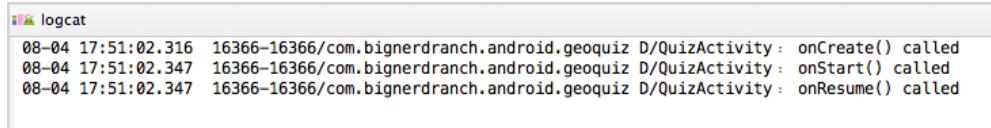


图3-4 应用启动后，被调用的三个生命周期方法

(如看不到过滤后的信息列表，请选择LogCat过滤器下拉窗口中的QuizActivity过滤项。)

下面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3-5所示。

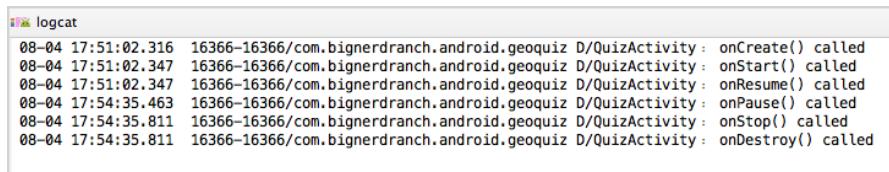
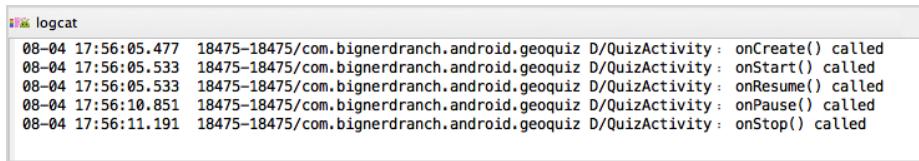


图3-5 单击后退键销毁activity

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。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3-6所示。



```
logcat
08-04 17:56:05.477 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 17:56:10.851 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 17:56:11.191 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
```

图3-6 单击主屏幕键停止activity

现在，我们调出设备的任务管理器。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3-7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3-7 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，为快速响应并返回应用，Android只是暂停当前activity而并没有销毁它。

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

另外，如果当前activity界面被完全或部分遮挡（如弹出窗口），那么它会被系统暂停，用户

无法同它交互。弹出窗口关闭后，它会继续运行。

在本书的后续学习过程中，为完成各种现实任务，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Fn+Control+F12/Ctrl+F12组合键。）

设备旋转后，GeoQuiz应用又回到了第一道问题。查看LogCat日志查找问题原因，如图3-8所示。

```

08-04 18:01:52.527 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 18:01:52.557 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:01:52.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 18:08:53.557 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onDestroy() called
08-04 18:08:53.585 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called

```

图3-8 QuizActivity已死，QuizActivity万岁

设备旋转时，系统会销毁当前QuizActivity实例，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`会初始化为0，因此用户看到的还是第一道题目。稍后会修正这个缺陷。现在先来深入分析该问题产生的原因。

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，下面为设备配置变更新建备选资源，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

在项目工具窗口中，右键单击res目录后选择New→Android resource directory菜单项。创建资

源目录界面列出了资源类型及其对应的资源特征，如图3-9所示。从资源类型（Resource type）列表中选择layout，保持Source Set的main选项不变。接下来选中待选资源特征列表中的Orientation，然后单击>>按钮将其移动至已选资源特征区域。

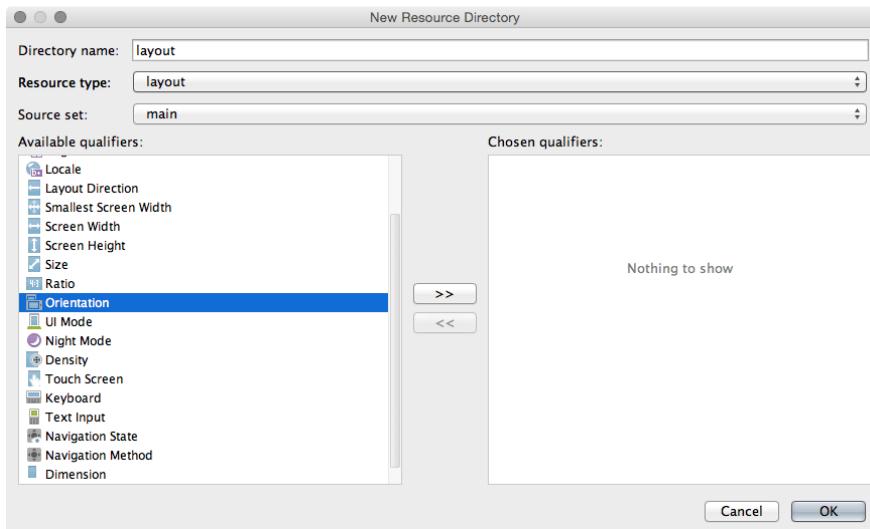


图3-9 创建新的资源目录

最后，确认选中Screen Orientation下拉列表中的Landscape选项，并确保目录名显示为layout-land，如图3-10所示。点击OK按钮让Android Studio创建res/layout-land。

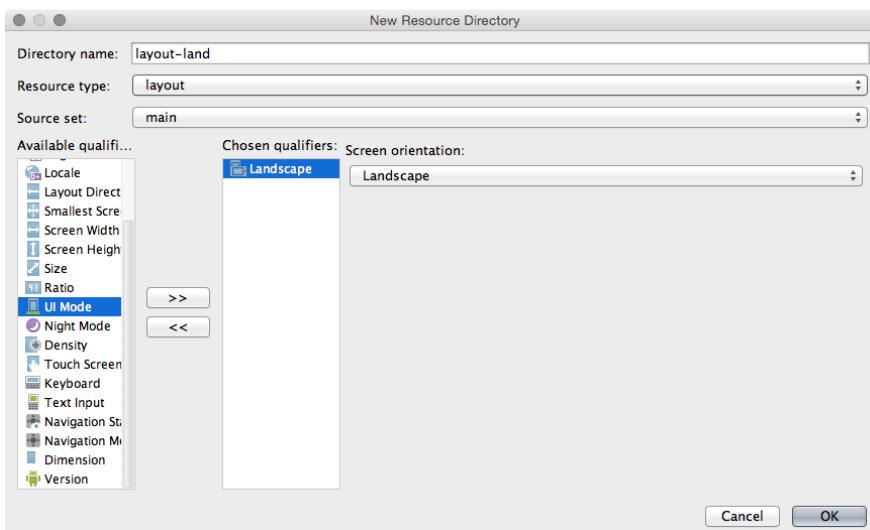


图3-10 创建res/layout-land

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。

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。然而, 目前在res/layout-land目录下并没有布局资源。让我们解决这个问题。

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。

为了与默认的布局文件相区别, 我们需要修改水平模式布局文件。请参照图3-11进行相应修改。

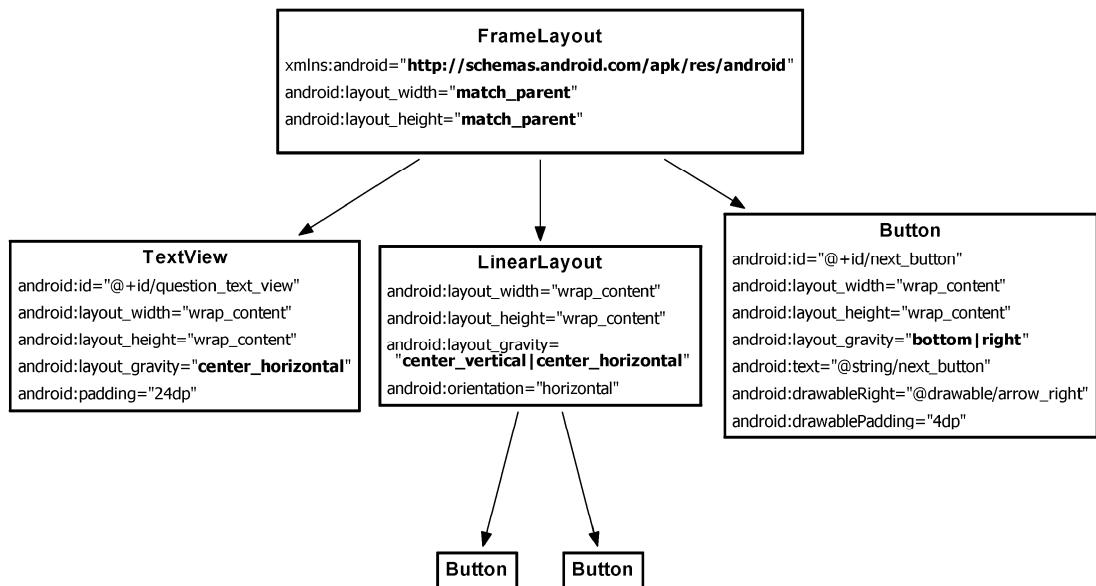


图3-11 备选的水平模式布局

用FrameLayout替换LinearLayout。FrameLayout是最简单的ViewGroup组件, 它不以特定方式安排其子视图的位置。FrameLayout子视图的位置排列取决于它们各自的android:layout_gravity属性。

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3-11, 打开layout-land/activity_quiz.xml文件进行相应的修改。完成后可同代码清单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GeoQuiz应用。旋转设备至水平方位，查看新的布局界面，如图3-12所示。当然，这不仅是一个新的布局界面，也是一个新的QuizActivity。

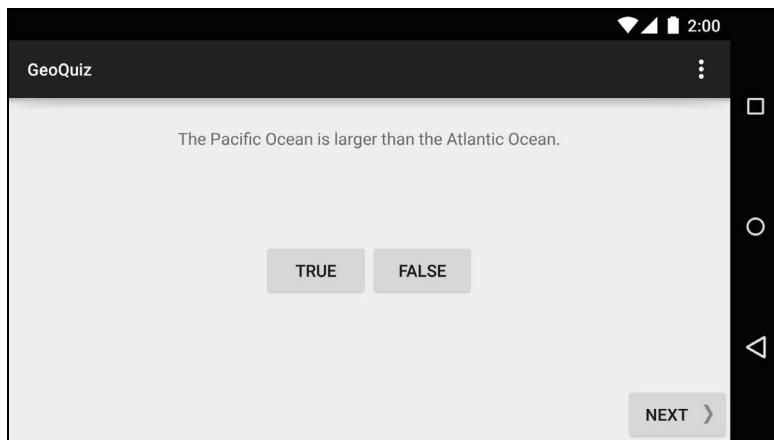


图3-12 处于水平方位的QuizActivity

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可自动完成最佳匹配资源的调用，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，需再次调用setContentView(R.layout.activity_quiz)方法。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配置新的设备配置。

请记住，在应用运行中，只要设备配置发生了改变，Android就会销毁当前activity，然后再创建新的activity。另外，虽然在应用运行中也会发生可用键盘或语言的改变，但设备屏幕方向的改变最为常见。

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(...)的默认实现要求所有activity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。处理设备旋转问题时，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 AppCompatActivity {

    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    private 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(...)`方法中确认是否成功获取该数值。如获取成功，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();  
}
```

3

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

注意，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`或`Parcelable`接口的对象。在`Bundle`中保存定制类对象不是个好主意，因为你取回的对象可能已经过时了。比较好的做法是，通过其他方式保存定制类对象，而在`Bundle`中保存对象对应的基本数据类型的标示符。

测试`onSaveInstanceState(...)`实现方法是个好习惯，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。

基于用户体验考虑，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3-13所示。

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。

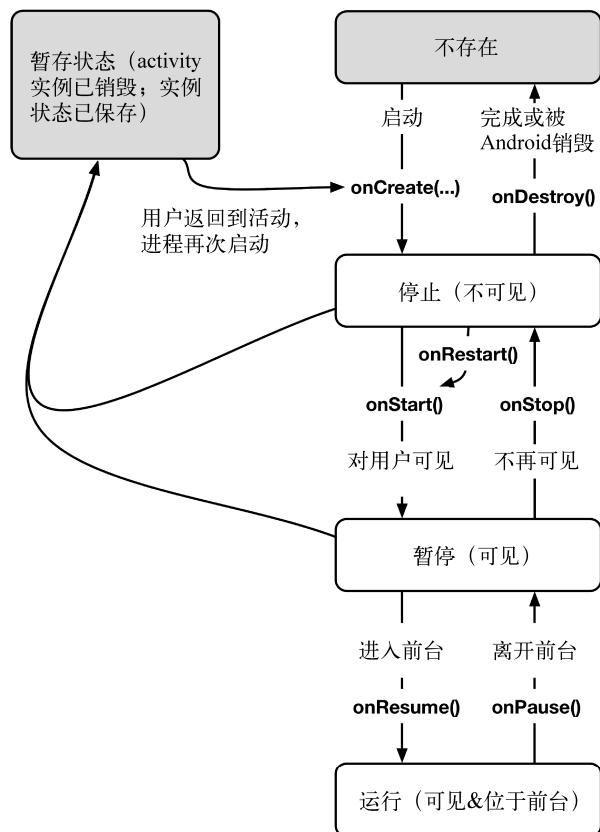


图3-13 完整的activity生命周期

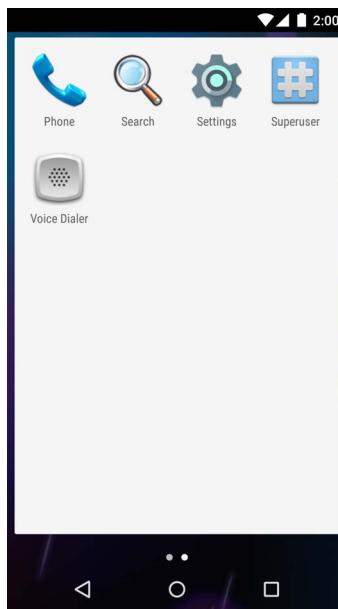
有时，Android不仅会销毁activity，还会彻底停止当前应用的进程。不过，只有在用户离开当前应用时才会发生这种情况。即使这种情况真的发生了，暂存的activity记录依然被系统保留着，以便于用户返回应用时activity的快速恢复。

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状态是否如预期般正确保存和恢复。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3-14所示。大部分模拟器包含的系统镜像应该都包含该应用。



3

图3-14 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3-15所示。

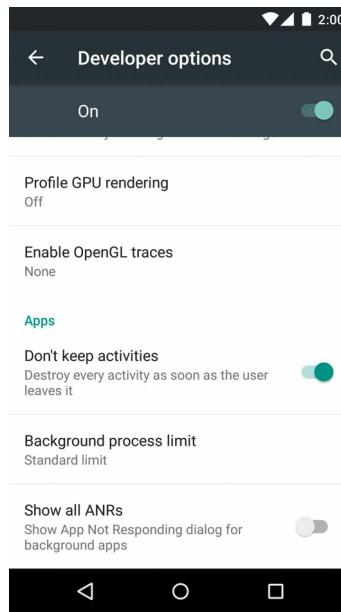


图3-15 启用Don't keep activities选项

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。测试完毕，记得关闭Don't keep activities选项，否则会产生系统和应用的性能问题。

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”。

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，不仅可以控制日志的内容，还可以控制用来区分信息重要程度的日志级别。Android支持如图3-16所示的五种日志级别。每一个级别对应着一个`Log`类方法。要输出什么级别的日志，调用对应的`Log`类方法就可以了。

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

图3-16 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);

Question 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);
}
```

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Android Studio内置的代码调试器。

为练习调试，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，看看会发什么。

图4-1是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。但假如不知道的话，从另一个角度来看或许有助于问题的排查。

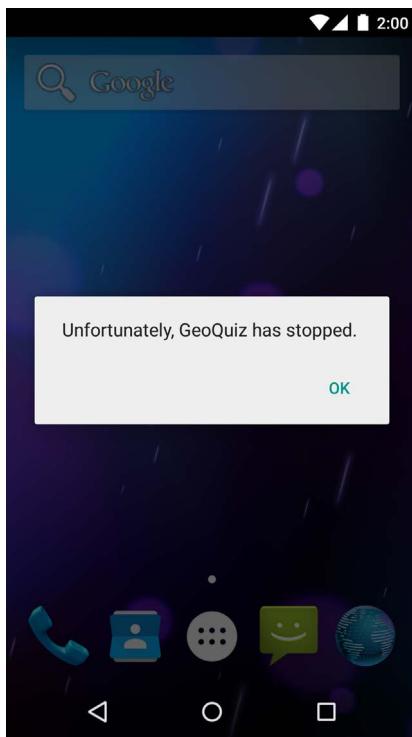


图4-1 GeoQuiz应用崩溃了

4.1 异常与栈跟踪

为方便查看异常或错误信息，展开Android DDMS工具窗口。上下滑动LogCat窗口滚动条，应该会看到如图4-2所示的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。如果看不到，可试着选择LogCat的No Filters过滤项。另外，还可以调整Log Level为Error，让系统只输出严重问题日志。

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。

在我们编写的大部分代码中，最后一个没有原因的异常往往是要关注的目标。这里，没有原因是`java.lang.NullPointerException`。紧接着该异常语句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。单击蓝色链接，Android Studio会自动跳转到源代码的对应行上。

Android Studio定位的这行代码是`mQuestionTextView`变量在`updateQuestion()`方法中的首次使用。名为`NullPointerException`的异常暗示了问题所在，即变量没有初始化。

```

09-03 12:44:08.523 5458-5458/? E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.bignerdranch.android.geoquiz, PID: 5458
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch.android.ge
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2184)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invoke(Native Method) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
    at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
    at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:35)
    at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
    at android.app.Activity.performCreate(Activity.java:5231)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1087)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2148)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invoke(Native Method)
    at java.lang.reflect.Method.invoke(Method.java:515) <3 more...>

```

图4-2 LogCat中的异常与栈追踪

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，记得在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，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内容。

4.1.1 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，注释掉`mcurrentIndex`变量递增的语句，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 = (mcurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});

...
}

```

运行GeoQuiz应用，点击NEXT按钮。可以看到，应用毫无响应。

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。

如确实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。

4.1.2 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 AppCompatActivity {
    ...

    private void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mcurrentIndex,
              new Exception());
        int question = mQuestionBank[mcurrentIndex].getTextResId();
        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。可以创建一个新的`Exception()`方法，把它作为不抛出的异常对象传入该方法。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，输出结果如图4-3所示。

```
09-04 12:47:37.733 30612-30612/com.bignerdranch.android.geoquiz D/QuizActivity: Updating question text
java.lang.Exception
    at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:34)
    at com.bignerdranch.android.geoquiz.QuizActivity.access$100(QuizActivity.java:12)
    at com.bignerdranch.android.geoquiz.QuizActivity$3.onClick(QuizActivity.java:83)
    at android.view.View.performClick(View.java:4438)
    at android.view.View$PerformClick.run(View.java:18422)
    at android.os.Handler.handleCallback(Handler.java:733)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
    at dalvik.system.NativeStart.main(Native Method)
```

图4-3 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。如果需要，我们可以直接从LogCat中复制并粘贴日志内容。

在继续学习之前，先删除日志记录代码，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 AppCompatActivity {

    ...

    private void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
              new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResId();
        mQuestionTextView.setText(question);
    }
}
```

4.1.3 设置断点

要使用Android Studio自带的代码调试器调试上一节中我们遇到的问题，首先要在`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红色圆圈。这就是我们设置的一处断点，如图4-4所示。

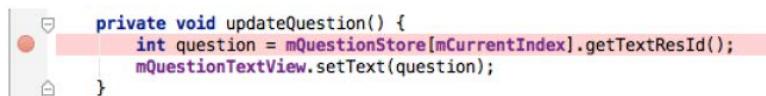


图4-4 已设置的一处断点

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，单击run按钮旁边的debug按钮，或选择Run→Debug ‘app’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，就会暂停。应用首先调用QuizActivity.onCreate(Bundle)方法，接着调用updateQuestion()方法，然后触发断点。

如图4-5所示，QuizActivity.java代码已经在代码编辑区打开了，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。

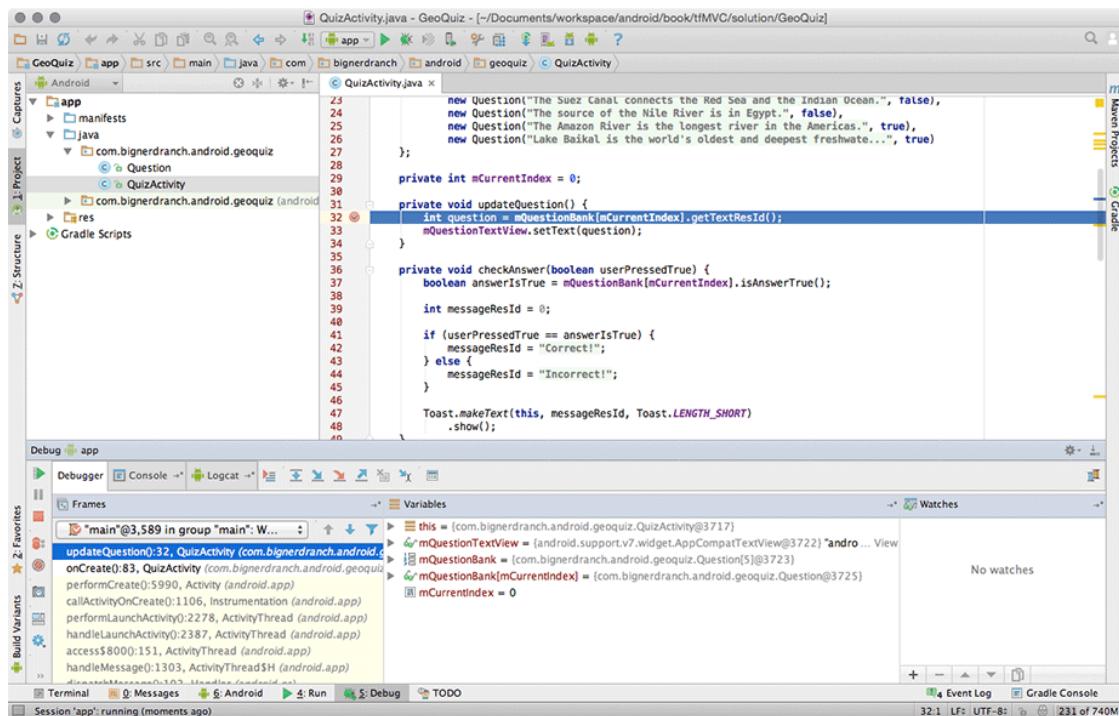


图4-5 代码在断点处停止执行

这时，由Frames和Variables视图组成的Debug工具窗口出现在了Android Studio的底部，如图4-6所示。

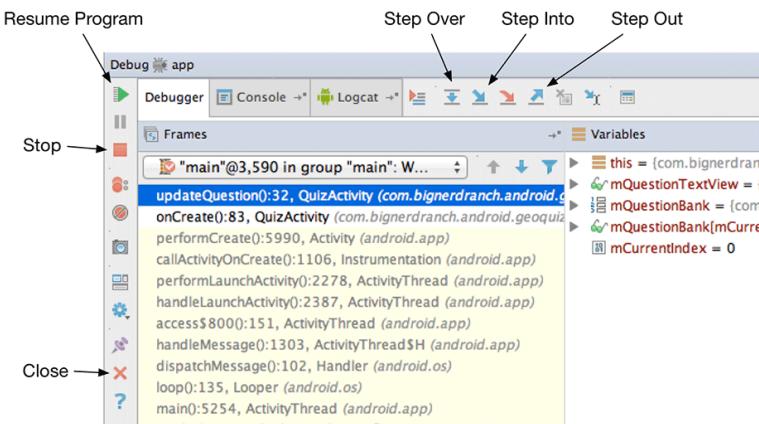


图4-6 代码调试视图

使用视图顶部的箭头按钮可单步执行应用代码。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 Program按钮让程序继续运行。然后，再次点击GeoQuiz中的NEXT按钮观察断点是否被激活（应该激活）。

既然程序执行停在了断点处，就可以趁机了解其他视图。变量视图（Variables）可以让我们观察到程序中各对象的值。应该可以看到在QuizActivity中创建的变量，另外还有一个特别的`this`变量值（QuizActivity本身）。

展开`this`变量后可看到很多变量。它们是QuizActivity类的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。

我们只需关心变量`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 Out按钮。

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。

接下来就是修复代码问题。不过，在修改代码前，必须先停止调试应用。停止调试有以下两种方式：

- 停止程序，单击图4-6所示的Stop按钮。
- 断开调试器，单击图4-6所示的Close按钮。

回到代码编辑区，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();
    }
});
...
}

```

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。大家实际体验后，也许各有各爱吧。

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、运行应用并跟踪排查应用问题。相对而言，代码调试的方法更为方便。应用以调试模式运行后，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。

4.1.4 使用异常断点

前面介绍的调试方法还不够用？那就试试使用调试器来捕捉异常吧。在QuizActivity.java中，注释掉一行会让应用崩溃的代码，如代码清单4-6所示。

代码清单4-6 使GeoQuiz再次崩溃（QuizActivity.java）

```

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
...
mNextButton = (Button) findViewById(R.id.next_button);
// mNextButton = (Button) findViewById(R.id.next_button);
mNextButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        mcurrentIndex = (mcurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});
...
}

```

选择Run → View Breakpoints...菜单项调出异常断点设置窗口，如图4-7所示。

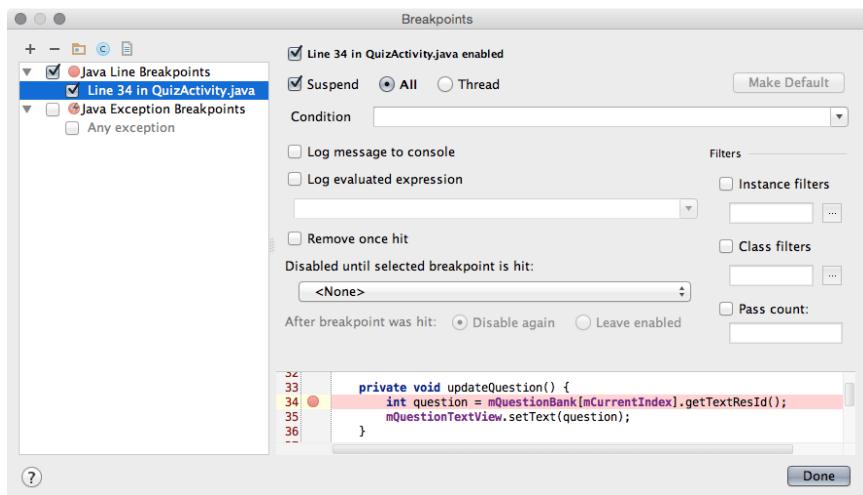


图4-7 设置异常断点

可以看到，当前设置的断点都显示在左边窗口，选中先前设置的断点，点击删除按钮（-）删除。

通过该对话窗口设置新的异常断点。这样，无论任何时候，只要应用抛出异常就可以触发该断点。如果需要，可限制断点仅针对未捕获的异常生效，也可以设置为对两种类型的异常都生效。

单击新增断点按钮（+）设置一个新断点。选择下拉列表中的Java Exception Breakpoints选项。接下来选择要捕捉的异常类型。输入 `RuntimeException`，按提示选择 `RuntimeException` (`java.lang`)。点击 Done 按钮完成断点设置。`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。

点击 Done 按钮调试 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，建议及时清除那些不需要的断点。否则，在调试的时候，如果一些系统框架代码或者我们无需关注的地方有异常发生，可能会触发先前设置的断点。继续学习之前，删除刚才设置的断点。

取消对 QuizActivity.java 的代码注释，使 GeoQuiz 保持良好的状态。

4.2 Android特有的调试工具

大多数Android应用调试和Java应用调试相似。然而，Android还是有其特有的应用调试场景，比如应用资源问题。显然，Java编译器并不擅长处理此类问题。

4.2.1 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检查代码错误的特殊程序。凭着对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。在大多数情况下,Android Lint检查出的问题都值得关注。

我们会在第6章看到Android Lint对设备兼容问题的警告。此外,Android Lint能够检查定义在XML文件中的对象类型。在QuizActivity.java中,人为制造一处错误,如代码清单4-7所示。

代码清单4-7 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却可以捕获到该错误。可以看到Lint立即高亮显示了一行代码,指示此处有问题。

假如想主动查看项目中的所有潜在问题,可以选择Analyze→Inspect Code...菜单项手动运行Lint。在被问及检查项目的哪部分时,选择Whole project。Android Studio会立即运行Lint和其他一些静态分析器开始分析代码。

检查完毕,所有的潜在问题会按类别列出。展开Android Lint类别可看到项目的具体Lint信息,如图4-8所示。

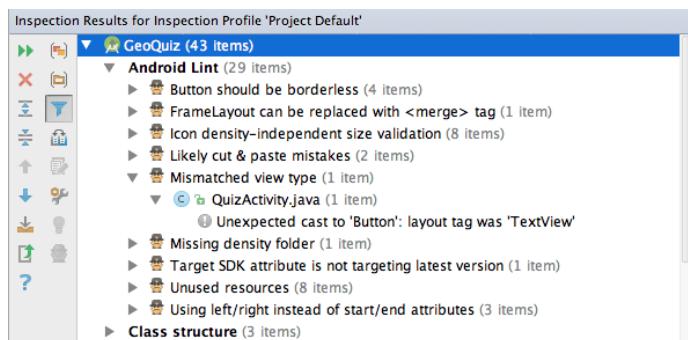


图4-8 Lint发出的警告

继续展开还可以看到更加详细的信息,包括问题发生的地方。

Mismatched view type错误是我们人为制造的。现在,对照代码清单4-8修正代码错误。

代码清单4-8 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);

    ...
}

```

4

最后，重新运行应用，确认问题已得到修正。

4.2.2 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重新保存文件，Android Studio会准确无误地重新编译项目。

不过，资源编译错误有时会一直或莫名其妙地出现。如遇这种情况，请尝试如下操作。

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，项目中资源引用的地方都会出错。通常，这是由某个布局XML文件中的拼写错误引起的。既然布局XML文件有时无法得到有效校验，拼写错误自然也就难以发现了。修正找到的错误并重新保存XML文件，Android Studio会生成新的R.java文件。

清理项目

选择Build → Clean Project菜单项。Android Studio会重新编译整个项目，消除错误。建议经常进行深度项目清理。

使用Gradle同步项目

如果修改了build.gradle配置文件，就需要同步更新项目的编译设置。选择Tools → Android → Sync Project with Gradle Files菜单项，Android Studio会使用正确的项目设置重新编译项目。这会解决Gradle配置变更带来的问题。

运行Android Lint

仔细查看Lint警告信息。你常常会在这个工具有新的发现。

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。休息冷静一下，再重新查看Android Lint报告的错误和警告，或许就能找出代码错误或拼写输入错误。

如果上述操作无法解决问题，或是遇到Android Studio的使用问题，还可以访问网站<http://stackoverflow.com>或本书论坛<http://forums.bignerdranch.com>寻求帮助。

第5章

第二个activity

5

本章，我们为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，方便用户查看当前问题的答案，如图5-1所示。

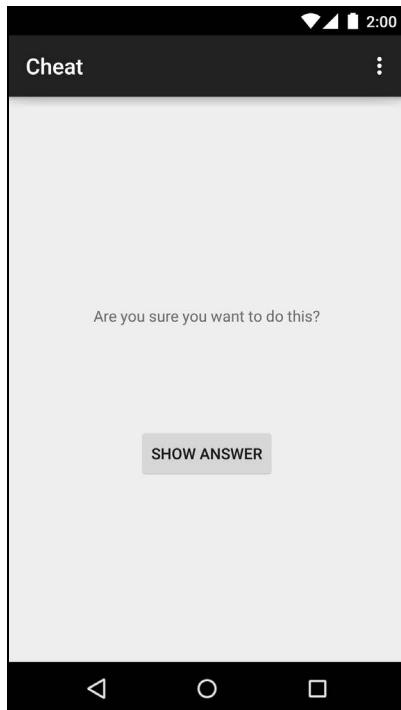


图5-1 CheatActivity提供了偷看答案的机会

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

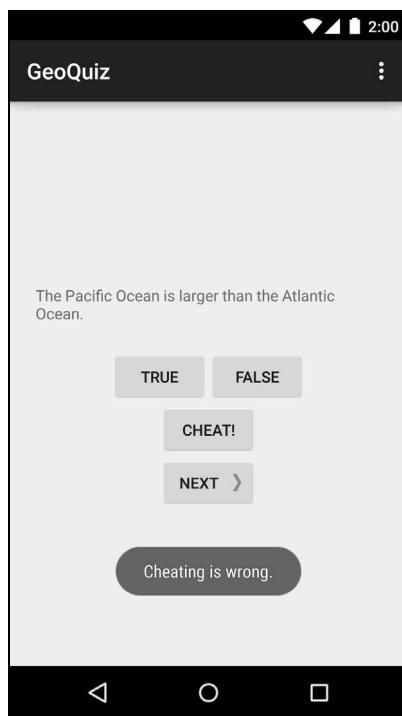


图5-2 有没有偷看答案，别想瞒过QuizActivity

通过本章GeoQuiz应用的升级开发，我们可以学到以下知识点。

- 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）间传递数据。

5.1 创建第二个 activity

要创建新的activity，接下来要做的事不少。好在Android Studio的新activity创建向导能帮我们省不少事。

不过，现在还是先打开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="warning_text">Are you sure you want to do this?</string>
<string name="show_answer_button">SHOW ANSWER</string>
<string name="cheat_button">CHEAT!</string>
<string name="judgment_toast">Cheating is wrong.</string>

</resources>

```

5.1.1 创建新的activity

创建新的activity至少涉及三个文件：Java类、XML布局和应用的manifest文件。这三个文件关联紧密、环环相扣，绝对不能搞错。因此，强烈建议使用Android Studio的新建activity向导功能。

在项目工具窗口中，右键单击com.bignerdranch.android.geoquiz包，选择New → Activity → Empty Activity菜单项启动新建activity向导，如图5-3所示。

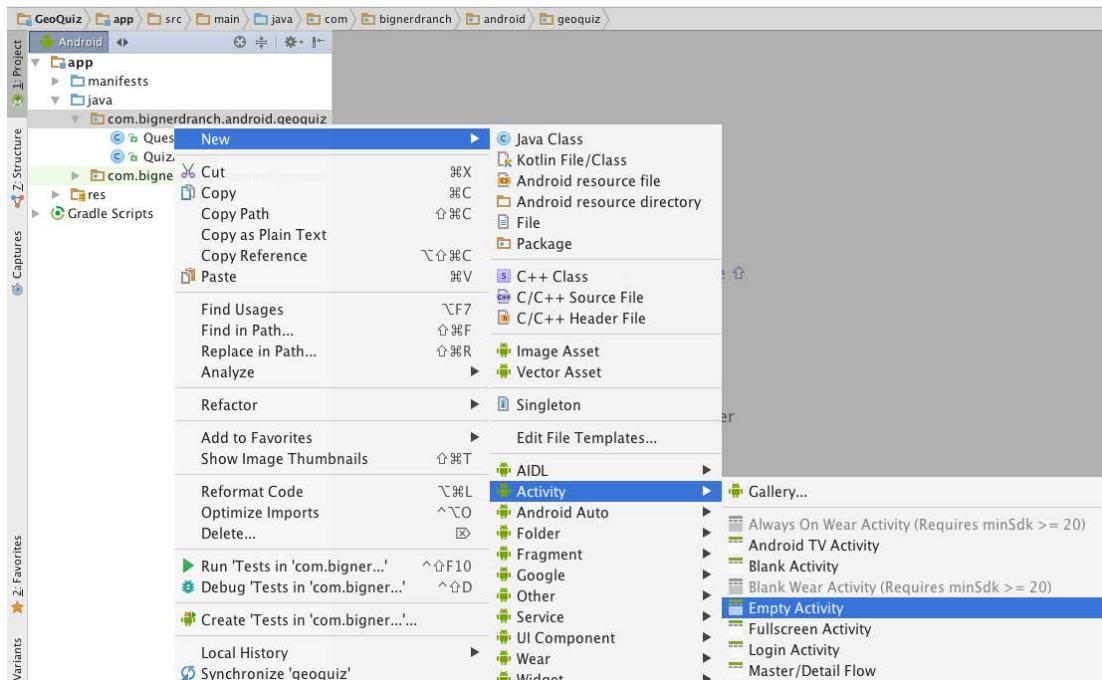
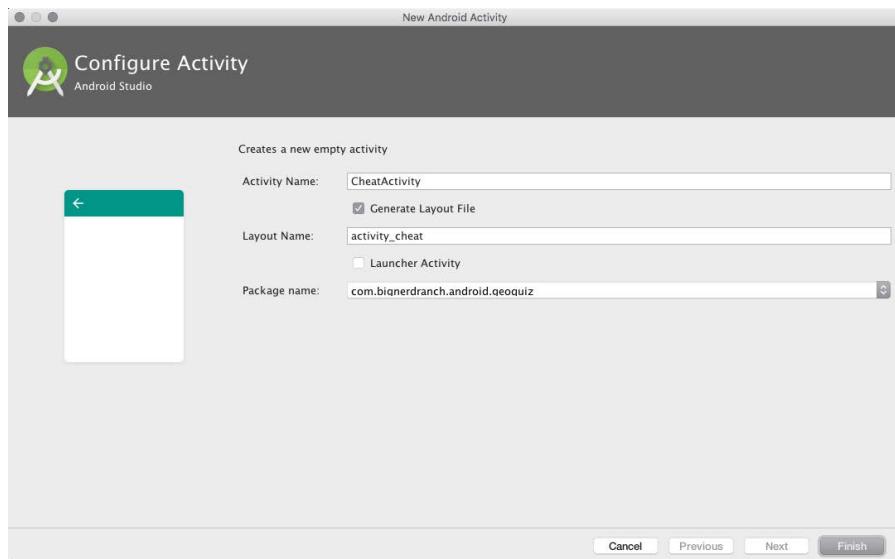


图5-3 新建activity向导菜单

在随后弹出的对话框中，将类命名为**CheatActivity**，如图5-4所示。这是Activity子类的名字。命名完类后，Layout Name自动赋值为**activity_cheat**。这是向导为布局文件创建的基本名称。



5

图5-4 新的空activity向导

由于包名决定CheatActivity.java文件存放的位置，再看看包名是否符合要求。最后，保持其他默认设置不变，点击Finish按钮一起来见证Android Studio的强大向导功能。

接下来的任务是设计美观的用户界面。本章开头的截图是CheatActivity视图完成后的样子。组成视图的组件定义如图5-5所示。

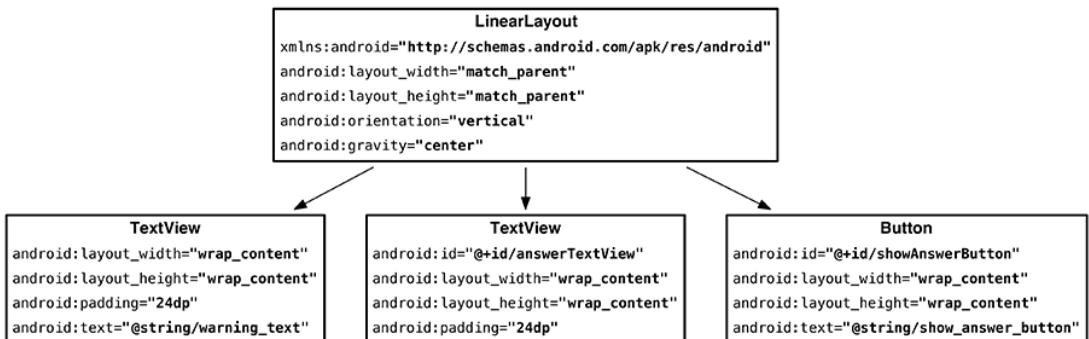


图5-5 组件定义示意图

新建activity向导完成后，Android Studio应该已经打开了layout目录中的activity_cheat.xml。如果没有，请将其打开并切换至文字视图模式。

参照图5-5创建布局XML文件。依次以LinearLayout组件替换样例布局。第8章以后，我们将不再展示大段的XML代码，而仅以图5-5的方式给出布局组件图示。最好现在就开始习惯参照图示创建布局XML文件。完成后，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）

```

<LinearLayout 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"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.bignerdranch.android.geoquiz.CheatActivity">

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

    <TextView
        android:id="@+id/answer_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        tools:text="Answer"/>

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

</LinearLayout>

```

注意用于显示答案的TextView组件，它的tools和tools:text属性的命名空间比较特别。该命名空间可以覆盖组件的任何属性，以便在Android Studio预览中进行不同的展示。既然TextView有text属性，我们可以为它提供初始值，在应用运行前就知道它大概的样子。不用担心，应用运行时，Answer文字不会显示出来。真的很方便！

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。

在预览工具窗口中，找到预览界面上方工具栏里一个横屏设备模样的按钮（上方带弧形蓝色箭头）。单击该按钮切换布局预览方位，如图5-6所示。

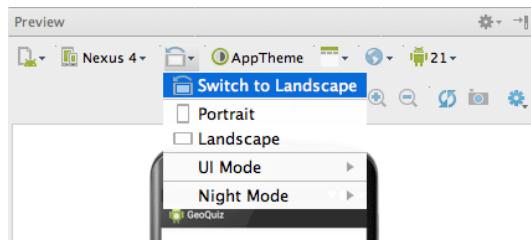


图5-6 水平方位预览布局（activity_cheat.xml）

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

在项目工具窗口中，找到com.bignerdranch.android.geoquiz类包，打开CheatActivity.java文件中的CheatActivity类。

当前，CheatActivity类已有onCreate(...)方法的默认实现，用来将activity_cheat.xml文件中的布局资源ID传递给setContentView(...)方法。

CheatActivity类的onCreate(...)方法还有更多的事情要做。现在，先一起来看看新建activity向导自动完成的另一件事：manifest配置文件中的CheatActivity声明。

5

5.1.3 在 manifest 配置文件中声明 activity

manifest配置文件是个包含元数据的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app/manifests目录中找到它。

在项目工具窗口中，找到并打开它。还可使用Android Studio的快速打开文件功能：使用Command+Shift+O（或Ctrl+Shift+N）快捷键，呼出快速打开对话框，利用提示功能或直接输入目标文件名，按Return（或Enter）键打开。

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。同样，新建activity向导也自动声明了CheatActivity，如代码清单5-3加亮部分所示。

代码清单5-3 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" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".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">
        </activity>
```

```
</application>
</manifest>
```

这里的`android:name`属性是必需的。属性值前面的`.`告诉操作系统：activity类文件位于manifest配置文件头部包属性值指定的包路径下。

`android:name`属性值也可以设置成完整的包路径，如`android:name="com.bignerdranch.android.geoquiz.CheatActivity"`。

manifest配置文件里还有很多有趣的东西。不过，现在还是先集中精力搞定`CheatActivity`的配置和运行。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，应用立即创建`CheatActivity`实例，并显示其用户界面。这就需要在`layout/activity_quiz.xml`和`layout-land/ activity_quiz.xml`布局文件中定义新按钮。

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，如代码清单5-4所示。

代码清单5-4 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"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"/>

</LinearLayout>
```

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

代码清单5-5 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。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5-6所示。

代码清单5-6 启用Cheat按钮（QuizActivity.java）

```
public class QuizActivity extends AppCompatActivity {

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

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**startActivity**方法：

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

你是否想当然地认为，**startActivity(...)**方法是个静态方法，启动activity就是调用Activity子类的该方法？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，调用请求发送给了操作系统的**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5-7所示。

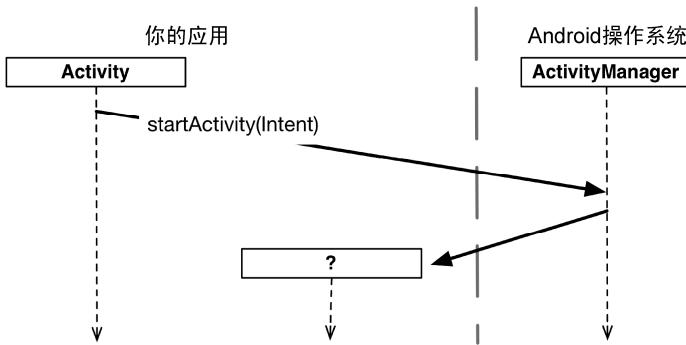


图5-7 启动activity

ActivityManager该启动哪个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是一种多用途通信工具。**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，intent用来告诉**ActivityManager**该启动哪个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**类型参数告诉**ActivityManager**应该启动哪个activity；**Context**参数告诉**ActivityManager**在哪里可以找到它，关系图如图5-8所示。

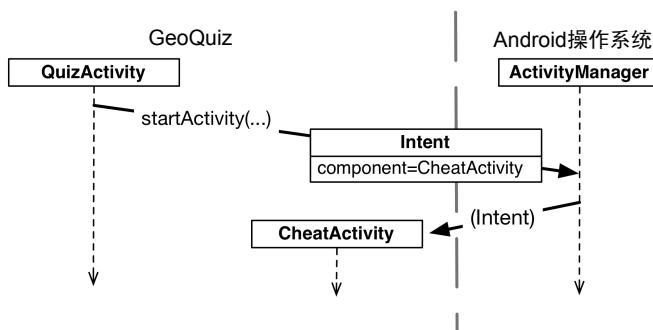


图5-8 intent: ActivityManager的信使

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5-7所示。

代码清单5-7 启动CheatActivity (QuizActivity.java)

```

...
mCheatButton = (Button)findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // Start CheatActivity
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        startActivityForResult(i);
    }
});

...

```

在启动activity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，可能会导致应用崩溃。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。

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。

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。在同一应用中，我们使用显式intent来启动activity。

同一应用里的两个activity，通信却要借助于应用外部的ActivityManager，这看起来可能有点奇怪。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章学习使用隐式intent。

5.3 activity 间的数据传递

既然QuizActivity和CheatActivity都已经就绪，接下来就可以考虑它们之间的数据传递了。图5-9展示了两个activity间传递的数据信息。

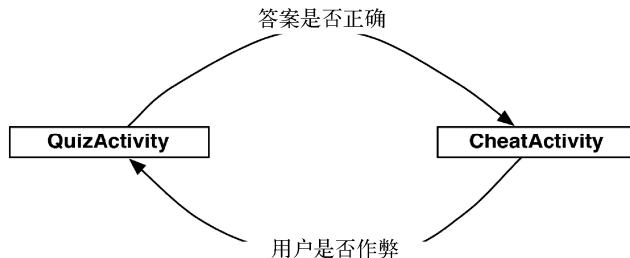


图5-9 QuizActivity与CheatActivity间的对话

CheatActivity启动后，QuizActivity会通知它当前问题的答案。

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

接下来，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

为通知CheatActivity当前问题的答案，需将以下语句的返回值传递给它：

```
mQuestionBank[mCurrentIndex].isAnswerTrue()
```

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，访问并获取其中的extra数据信息。关系图如图5-10所示。

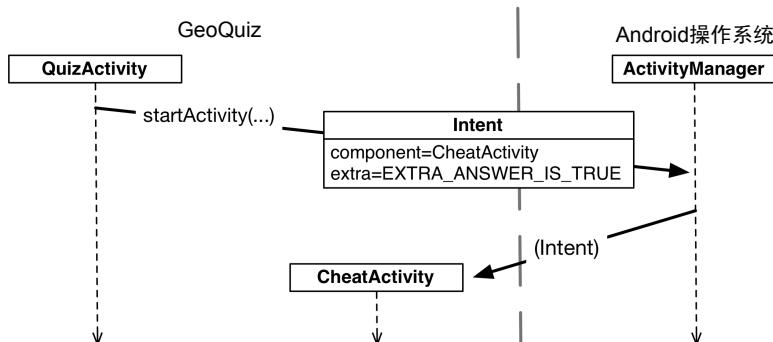


图5-10 intent extra: activity间的通信与数据传递

如同QuizActivity.onOptionsItemSelected(Bundle)方法中用来保存mCurrentIndex值

的键值结构，extra也是一种键值结构。

要将extra数据信息添加给intent，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(...)方法形式多变。不变的是，它总是有两个参数。一个参数是固定为String类型的键，另一个参数值可以是多种数据类型。该方法返回intent自身，因此，需要时可进行链式调用。

在CheatActivity.java中，为extra数据信息新增键值对中的键，如代码清单5-8所示。

代码清单5-8 添加extra常量（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private 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定义键。如代码清单5-8所示，使用包名修饰extra数据信息，可以避免来自不同应用的extra间发生命名冲突。

现在，可以返回到QuizActivity并将extra附加到intent上。不过我们有个更好的实现方法。对于CheatActivity处理extra信息的实现细节，QuizActivity和应用的其他代码无需知道。因而，我们可转而在newIntent(...)方法中封装这些逻辑。

在CheatActivity中，创建newIntent(...)方法，如代码清单5-9所示。

代码清单5-9 CheatActivity中的newIntent(...)方法（CheatActivity.java）

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";

    public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
        Intent i = new Intent(packageContext, CheatActivity.class);
        i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
        return i;
    }
    ...
}
```

使用新建的静态方法，可以正确创建Intent，它配置有CheatActivity需要的extra。answerIsTrue布尔值以EXTRA_ANSWER_IS_TRUE常量放入intent以供解析。利用这种方式，配置传递intent是不是容易多了？

现在就体验吧，在QuizActivity的按钮监听器中，应用newIntent(...)方法，如代码清单5-10所示。

代码清单5-10 用一个extra启动 (QuizActivity.java)

```

    ...
    mCheatButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Start CheatActivity
            Intent i = new Intent(QuizActivity.this, CheatActivity.class);
            boolean answerIsTrue = mQuestionBank[mcurrentIndex].isAnswerTrue();
            Intent i = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
            startActivity(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。如果附加多个extra，要在newIntent(...)方法相应添加多个参数。

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），它在无法获得有效键值时使用。

在CheatActivity代码中，编写代码实现从extra获取信息，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 AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";

    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，实现单击SHOW ANSWER按钮后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 AppCompatActivity {  
    ...  
    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.answer_text_view);  
  
        mShowAnswer = (Button) findViewById(R.id.show_answer_button);  
        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。TextView.setText(int)方法用来设置TextView要显示的文字。TextView.setText(...)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。下面我们就来解决这个问题。

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，通常会用到该请求代码。虽然QuizActivity只启动一种类型的子activity，但为应对未来的需求变化，现在就应设置请求代码常量。

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)

```
public class QuizActivity extends AppCompatActivity {
    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";
    private static final int REQUEST_CODE_CHEAT = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        ...

        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean answerIsTrue = mQuestionBank[mcurrentIndex].isAnswerTrue();
                Intent i = CheatActivity.newIntent(QuizActivity.this, answerIsTrue);
                startActivityForResult(i, REQUEST_CODE_CHEAT);
            }
        });
    ...
}
```

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)
```

一般来说，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按钮和一个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。

在CheatActivity代码中，为extra的键增加常量，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 AppCompatActivity {

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

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

        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);
            }
        });
    }

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

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5-11展示了应用内部的交互时序。

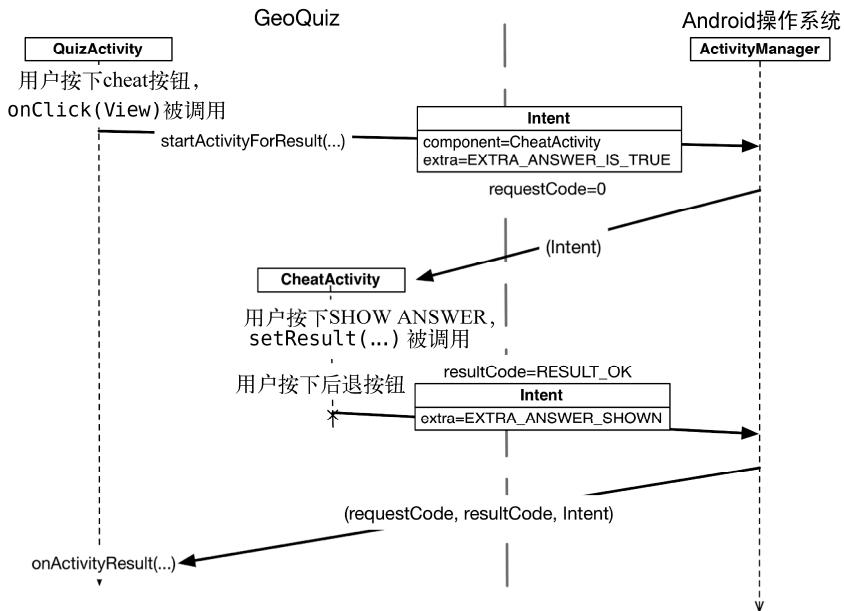


图5-11 GeoQuiz应用内部的交互时序图

最后覆盖QuizActivity的onActivityResult(int, int, Intent)方法来处理返回结果。然而，结果intent的内容也是CheatActivity的实现细节，因而还要添加另一个方法协助解析出QuizActivity能用的信息，如代码清单5-15所示。

代码清单5-15 解析结果intent (CheatActivity.java)

```

public static Intent newIntent(Context packageContext, boolean answerIsTrue) {
    Intent i = new Intent(packageContext, CheatActivity.class);
    i.putExtra(EXTRA_ANSWER_IS_TRUE, answerIsTrue);
    return i;
}

public static boolean wasAnswerShown(Intent result) {
    return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);
}

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

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。别忘了检查请求代码和返回代码是否符合预期。这是最佳代码实践，方便将来进行维护。onActivityResult(...) 方法的实现如代码清单 5-16 所示。

代码清单 5-16 on.onActivityResult(...) 方法的实现 (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsCheater;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }
        if (requestCode == REQUEST_CODE_CHEAT) {
            if (data == null) {
                return;
            }
            mIsCheater = CheatActivity.wasAnswerShown(data);
        }
    }
    ...
}
```

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 5-17 所示。

代码清单 5-17 基于 mIsCheater 变量值改变 toast 消息 (QuizActivity.java)

```
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
    int messageId = 0;
    if (mIsCheater) {
        messageId = 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5-18所示。

代码清单5-18 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" />
</application>

</manifest>

```

QuizActivity实例出现在屏幕上后，用户可单击CHEAT!按钮。CheatActivity实例随之在QuizActivity实例上被启动。此时，它们都处于activity栈中，如图5-12所示。

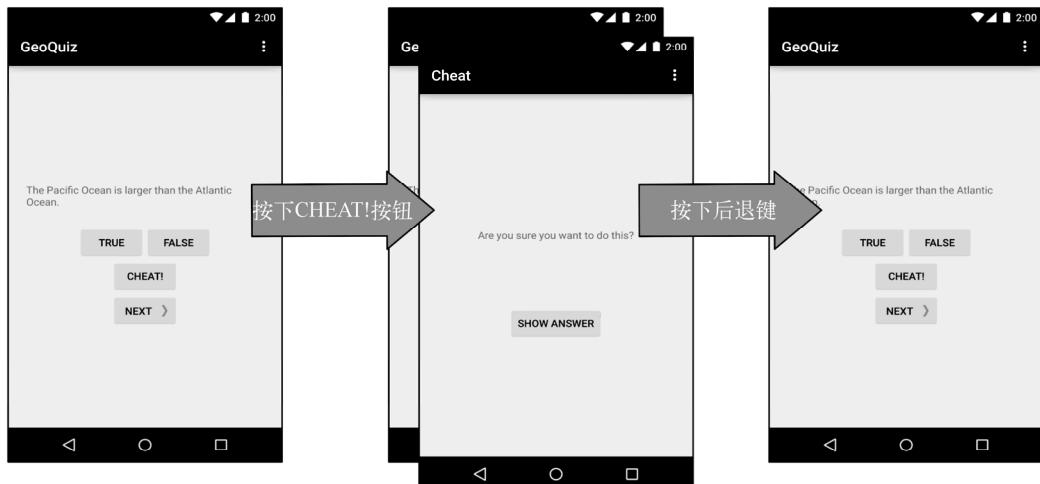


图5-12 GeoQuiz的回退栈

单击后退键，CheatActivity实例被弹出栈外，QuizActivity重新回到栈顶部，如图5-12所示。

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

如运行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5-13所示。

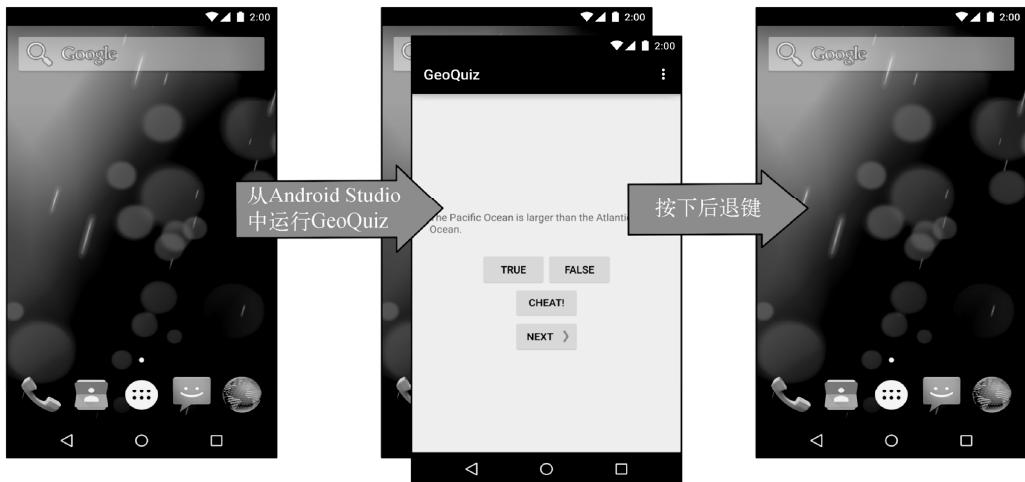


图5-13 后退返回至桌面

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5-14所示。

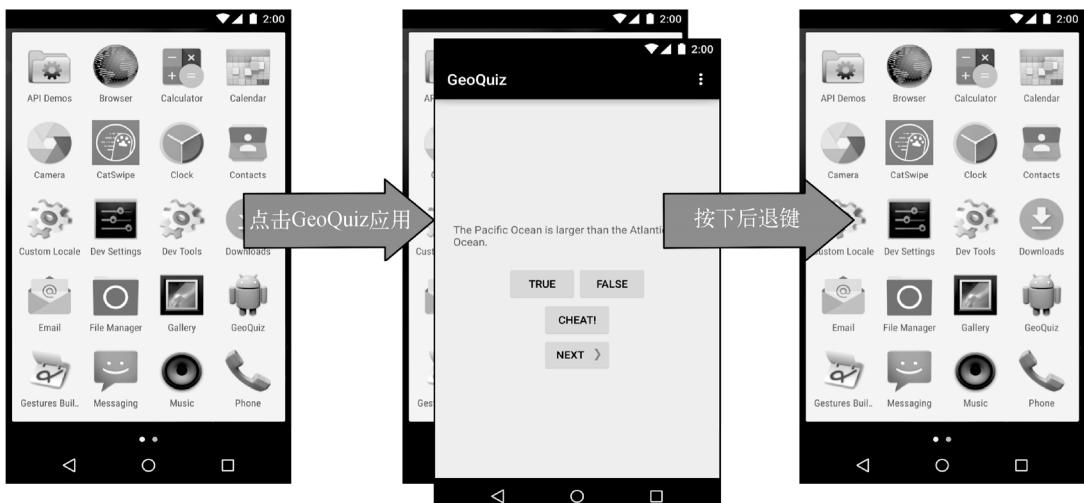


图5-14 从桌面启动器启动GeoQuiz应用

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，可以看到，**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“向上”按钮？我们将在第13章学习如何使用并配置它。）

5.5 挑战练习

作弊者是注定会失败的。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，可以旋转CheatActivity来清除作弊痕迹。
- 作弊返回后，用户可以旋转QuizActivity来清除mIsCheater变量值。
- 用户可以不断单击NEXT按钮，跳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。本章我们来学习Android系统版本相关的知识。在学习本书后续章节，以及应对未来实际的复杂应用开发时，你就会明白掌握本章内容有多么重要。

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2015年6月使用各版本的设备比例。

表6-1 Android API级别、固件版本以及在用设备比例

API级别	代号	设备固件版本	在用设备比例 (%)
22	Lollipop	5.1	0.8
21		5.0	11.6
19	KitKat	4.4	39.2
18	Jelly Bean	4.3	5.2
17		4.2	17.5
16		4.1	14.7
15	Ice Cream Sandwich (ICS)	4.0.3、4.0.4	5.1
10	Gingerbread	2.3.3 ~ 2.3.7	5.6
8	Froyo	2.2	0.3

注意，本表已忽略在用设备比例低于0.1%的版本。

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，Android 4.0.3及4.0.4 (API 15级) 的增量发行版本就取代了它。

当然，表6-1中的比例会不断变化，但从这些数字中已看出一种重要趋势，即新版本发布后，运行老版本的Android设备是不会立即进行升级或更换的。截至2015年6月，10%的设备仍然运行着代号为Ice Cream Sandwich或Gingerbread的SDK版本。Android 4.0.4 (ICS最后的升级版本) 发布于2012年3月。

(感兴趣的话，可去<http://developer.android.com/about/dashboards/index.html>查看表6-1数据的

动态更新。)

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，而他们又想在竞争中脱颖而出。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。

定制版Android设备不能运行Google发布的最新版Android系统。因此，用户只能寄望于定制版的兼容升级。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，有的厂商干脆就不提供升级。生产商往往更愿意投入资源推出新设备，而不是持续升级旧设备。

6.2 Android 编程与兼容性问题

6

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编程带来了严重的兼容性问题。为扩大市场份额，对于运行Jelly Bean、KitKat、Lollipop等最新版本的Android设备（还要考虑各种尺寸），Android开发人员必须保证应用兼容并运行良好。

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，但配置修饰符可用来处理屏幕适配（第17章会介绍相关知识）。不过，对于同样运行着Android系统的Google TV和Android穿戴设备，由于UI差异太大，应用的交互模式和设计通常需要重新考虑。

6.2.1 比较合理的版本

本书支持的最老版本是API 16级（Jelly Bean）。虽然还在支持，但我们更应该投入精力在较新系统版本上（API 16+级）。当前，Froyo、Gingerbread和Ice Cream Sandwich系统版本的市场份额正逐月下降，还在这些老设备上投入过多已得不偿失了。

对于增量版本，向下兼容通常问题不大。主要版本向下兼容才是真正的大麻烦。也就是说，仅支持4.x版本的工作量不大，但需要支持到2.x的话，考虑到这么多不同版本的差异，工作量就相当大了。（有关2.x版本开发支持的细节，请参读本书第1版。）针对Android 5.0（Lollipop）和4.x的支持，Google提供了一些兼容库，大大降低了开发困难。我们在后续章节会进行学习。

为何要如此耗费精力支持2.x版本的开发呢？Honeycomb版本的发布是Android世界的一个重大转变分支，该版本引入了全新的UI和构造组件。Honeycomb只面向平板设备开发，所以直到Ice Cream Sandwich的发布，这些新发展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。

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学习的复杂性。

新建GeoQuiz项目时，在新建应用向导界面，设置最低SDK版本，如图6-1所示。（注意，Android的“SDK版本”和“API级别”可以交替使用。）

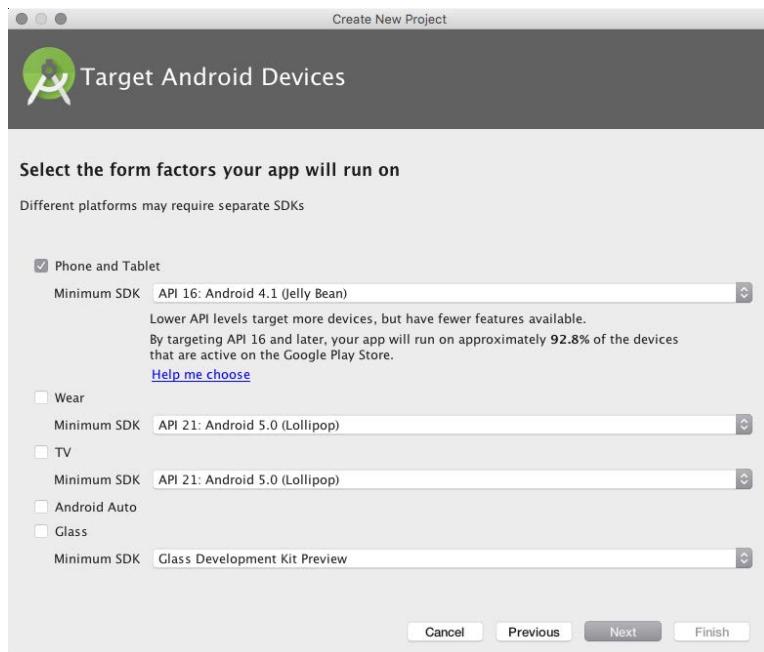


图6-1 创建新项目向导，还有印象吗

除了最低支持版本，还可以设置目标和编译版本。下面来看看都有哪些默认选项，新建项目时该如何选择。

所有的设置都保存在应用模块的build.gradle文件中。编译版本独占该文件。虽然最低SDK版本和目标SDK版本也设置在该文件中，但它们的作用是覆盖和设置AndroidManifest.xml配置文件。

打开应用模块下的build.gradle文件，查看compileSdkVersion、minSdkVersion和targetSdkVersion的属性值，如代码清单6-1所示。

代码清单6-1 查看编译配置（app/build.gradle）

```
...
compileSdkVersion 22
buildToolsVersion "23.0.0"

defaultConfig {
    applicationId "com.bignerdranch.android.geoquiz"
    minSdkVersion 16
    targetSdkVersion 22
}
...
```

6.2.2 SDK 最低版本

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 16级（Jelly Bean），便赋予了系统在运行Jelly Bean及以上版本的设备上安装GeoQuiz应用的权限。而在运行Froyo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将Jelly Bean作为SDK最低版本比较合适，因为有88%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

代码清单6-1中，最后一项标为compileSdkVersion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。Android Studio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21，代号为Lollipop）。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，从而使用新版本引入的方法和类。

可以修改build.gradle文件中的最低SDK版本、目标SDK版本以及编译SDK版本。修改完毕，项目和Gradle更改重新同步后才能生效。选择Tools→Android→Sync Project with Gradle Files菜单项，项目随即会重新完成编译。

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较大，由此带来的兼容问题需要处理。例如，在GeoQuiz应用中，如果调用了Jelly Bean（API 16级）以后的SDK版本中的代码会怎么样呢？

结果显示，在Jelly Bean设备上安装并运行应用时，应用会崩溃。

这个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，现在在老版本系统上调用新版本代码时，潜在问题在编译时就能被发现了。也就是说，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 16级或更早版本。现在，我们来增加API 21级（Lollipop）的代码，看看会发生什么。

打开CheatActivity.java文件，在SHOW ANSWER按钮的OnClickListener方法中，添加代码清单6-2所示代码，从而在隐藏按钮的同时，显示一段圆球特效动画。

代码清单6-2 添加动画特效代码（CheatActivity.java）

```
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);

        int cx = mShowAnswer.getWidth() / 2;
        int cy = mShowAnswer.getHeight() / 2;
        float radius = mShowAnswer.getWidth();
        Animator anim = ViewAnimationUtils
            .createCircularReveal(mShowAnswer, cx, cy, radius, 0);
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                mShowAnswer.setVisibility(View.INVISIBLE);
            }
        });
        anim.start();
    }
});
```

传入一些特定参数，`createCircularReveal`方法创建了一个`Animator`。首先，指定要显示或隐藏的`View`，然后是动画的中心位置、起始半径和结束半径。为隐藏按钮，这里将它的起始半径设置为按钮宽度，结束半径设置为0。

动画启动前，设置一个监听器，确定动画何时展示完毕。动画一结束，就显示答案并隐藏按钮。

最后播放动画，实现我们需要的效果。有关Android动画方面的知识将在第30章学习。

Android SDK API级别21才加入`ViewAnimationUtils`和它的`createCircularReveal`方法。因此，上述代码在低版本设备上运行时会崩溃。

加入代码清单6-2所示代码时，Android Lint会立即提示，这段代码对于最低版本SDK是不安

全的。如果没看到提示，请选择Analyze → Inspect Code....菜单项手动触发Lint。因为SDK编译版本为API 21级，编译器本身编译代码没有问题，而Android Lint知道项目SDK最低版本，所以及时指出了问题。

虽然Lint提示了类似Call requires API level 21 (Current min is 16)的警告信息，你可以忽略它。不过，出了问题可别怪Lint没有提醒你。

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21。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API 16级和更老版本设备上，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解决兼容性问题。

比较好的做法是将高API级别代码置于检查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

```

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);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            int cx = mShowAnswer.getWidth() / 2;
            int cy = mShowAnswer.getHeight() / 2;
            float radius = mShowAnswer.getWidth();
            Animator anim = ViewAnimationUtils
                .createCircularReveal(mShowAnswer, cx, cy, radius, 0);
            anim.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    super.onAnimationEnd(animation);
                    mShowAnswer.setVisibility(View.INVISIBLE);
                }
            });
            anim.start();
        } else {
            mShowAnswer.setVisibility(View.INVISIBLE);
        }
    }
});
```

`Build.VERSION.SDK_INT`常量代表了Android设备的版本号。可将该常量同代表Lollipop版本的常量进行比较。（版本号清单可参看网页http://developer.android.com/reference/android/os/Build.VERSION_CODES.html。）

现在动画特效代码只有在API 21级或更高版本的设备上运行应用才会被调用。应用代码在API

16级设备上终于安全了，Android Lint应该也满意了吧。

在Lollipop或更高版本的设备上运行GeoQuiz。启动CheatActivity时，确认看到了想要的动画特效。

也可以在Jelly Bean或KitKat设备（虚拟或实体）上运行GeoQuiz应用。当然，动画特效是看不到了，但可验证应用仍能正常运行。

6.3 使用Android开发者文档

从Android Lint错误信息中可看到不兼容代码所属的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。

越早熟悉使用开发者文档越有利于开发。没人能记住Android SDK中的海量信息，更不要说定期发布的新版本系统了。因此，学会查阅SDK文档，不断学习新的东西就尤显重要了。

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。发布部分讲述如何在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。

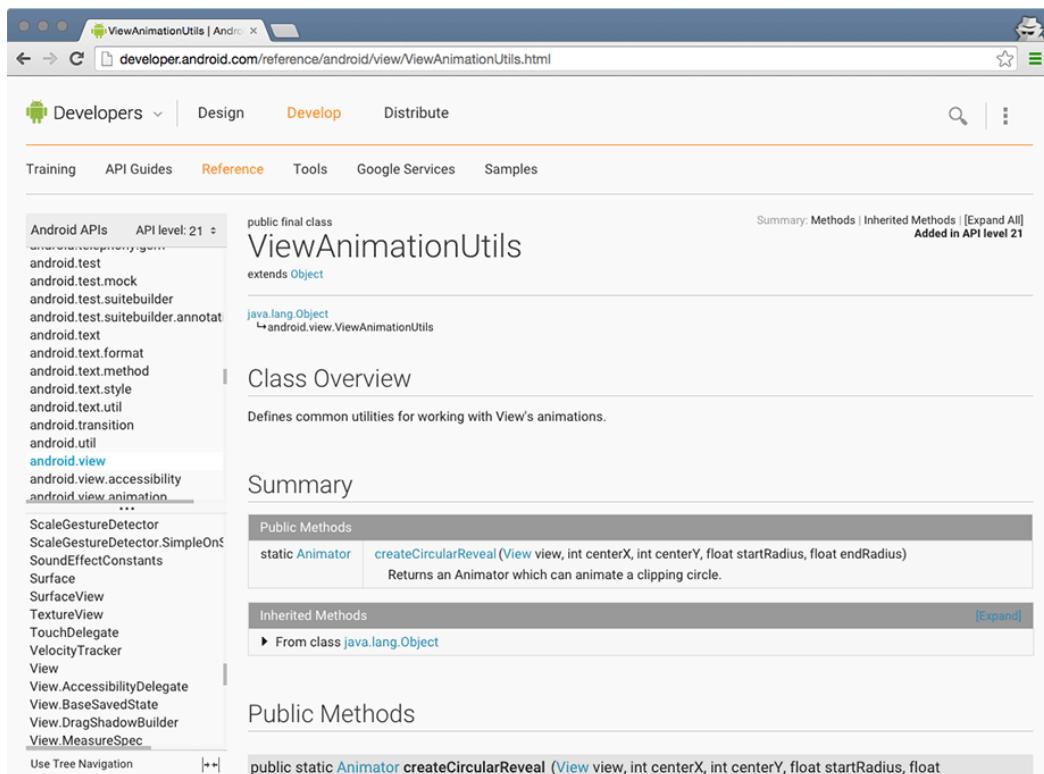
开发部分可细分为六大块内容。

- **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。
- **Google服务：**Google专属API的相关信息，包括Google地图和Google云消息。
- **示例代码：**如何使用各种API的示例代码。

Android文档可脱机查看。浏览已下载SDK的文件系统，找到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**ViewAnimationUtils**类所属的API级别，使用文档浏览器右上角的搜索框搜索它。搜索结果显示有多种类别的信息。我们想要的结果位于参考文档部分。（注意灵活使用左边的搜索过滤功能）。

选择第一条结果，进入**ViewAnimationUtils**类的参考文档页面，如图6-2所示。该页面顶部的链接可以链接到不同的部分。

图6-2 `ViewAnimationUtils`参考文档页面

向下滚动，找到并点击`createCircularReveal(...)`方法查看具体的方法描述。从该方法名的右边可以看到，`createCircularReveal(...)`方法最早被引入的API级别是API 21级。

如想查看`ViewAnimationUtils`类的哪些方法可用于API 16级，可按API级别过滤引用。在页面左边按包索引的类列表上方，找到API级别过滤框，目前它显示为API level: 21。展开下拉菜单，选择数字16。一般而言，所有API 16级以后引入的方法都会被过滤掉，自动变为灰色。`ViewAnimationUtils`类是在API 21级引入的，所以，我们会看到一条该类无法用于API 16级的警示信息。

API级别过滤非常有用，可以让我们知道应用要用到的类在哪个API级别可用。例如，在文档的参考页搜索`Activity`类，以API 16级过滤。结果显示，诸如`onEnterAnimationComplete`的很多方法是在API 16级才开始添加的。而在Lollipop SDK中，`onEnterAnimationComplete`属附加方法，为activity间的跳转提供了有趣的过场动画效果。

在后续章节的学习过程中，一定要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。Google还在不断地更新和改进Android文档，新知识、新概念也因此不断涌现，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6-3所示。

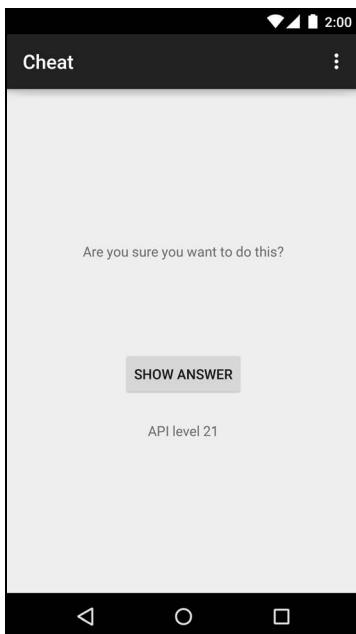


图6-3 完成后的用户界面

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属性来调整文本的尺寸或样式。

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。

CriminalIntent应用能记录陋习的标题、日期以及照片,也支持在联系人中查找当事人，通过E-mail、Twitter、Facebook或其他应用提出抗议。把陋习处理完后，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，需要13章的篇幅来完成它。应用的用户界面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。用户可新增记录或查看和编辑现有记录，如图7-1所示。

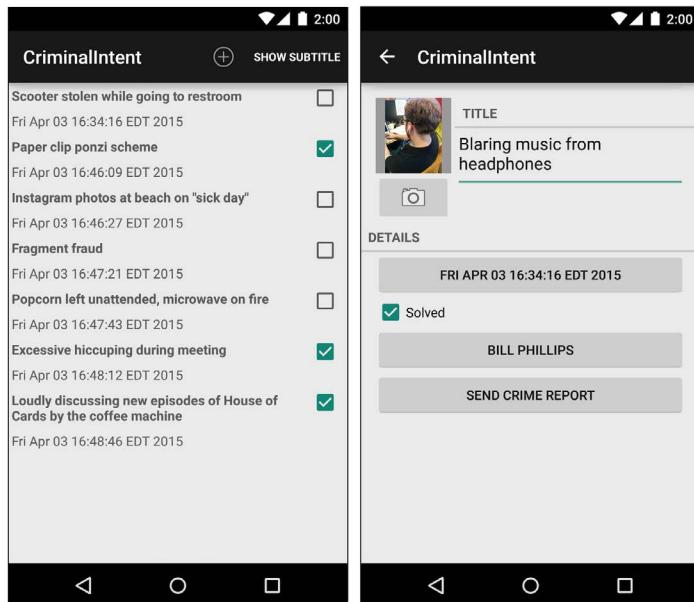


图7-1 CriminalIntent，一个列列表明细应用

7.1 UI设计的灵活性需求

你可能认为开发一个由两个activity组成的列表明细类应用就行了，其中一个activity负责管理记录列表界面，另一个负责管理记录明细界面。单击列表中某条记录会启动其明细activity实例，单击后退键会销毁明细activity并返回到记录列表activity界面，如此反复。

理论上这想法行得通，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的屏幕较大，能够同时显示列表和记录明细（最起码在横屏模式下是这样），如图7-2所示。

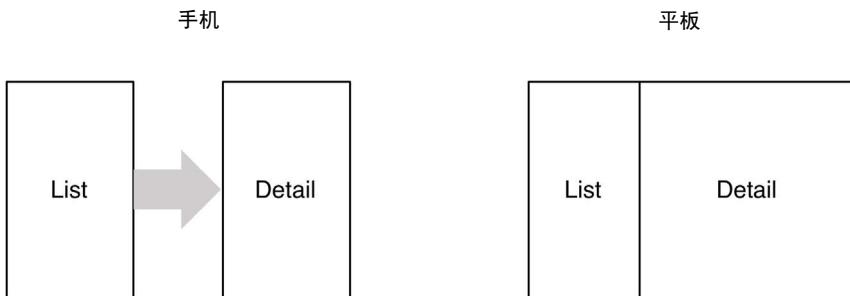


图7-2 手机和平板上理想的列表明细界面

- 假设用户正在手机上查看记录明细信息，并想查看列表中的下一条记录信息。如无需返回列表界面，滑动屏幕就能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。也就是说，适应用户或设备的需求，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来管理应用UI，可绕开Android系统activity使用规则的限制。

fragment是一种控制器对象，activity可委派它完成一些任务。这些任务通常就是管理用户界面。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。它自己也有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视图可预留供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，activity视图在技术上保持不变。因此不用担心会违反Android系统的activity使用规则。

下面来看看应用该如何支持同屏显示列表与明细内容。实际上，这类应用的activity视图是由列表fragment和明细fragment组装而成。明细视图负责显示列表项的明细内容。

利用fragment，可轻松实现选择不同的列表项就显示对应的明细视图Activity负责以一个明细fragment替换另一个明细fragment，如图7-3所示。这样，视图切换的过程中，就不用销毁activity了。

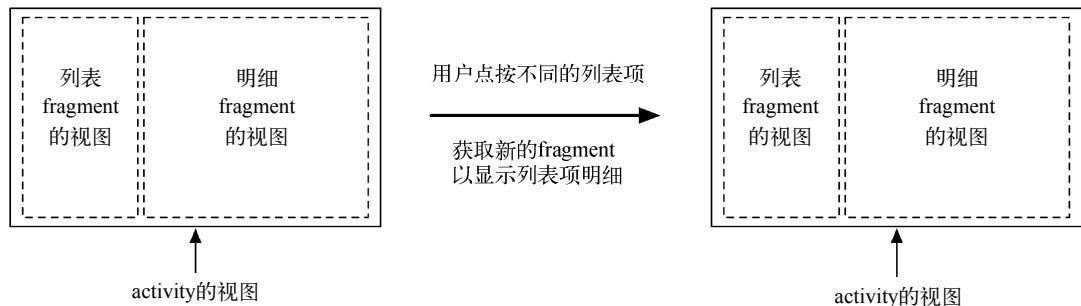


图7-3 明细fragment的切换

除列表明细应用外，使用UI fragment将应用的UI分解成构建块，还适用于其他类型的应用。例如，利用单个构建块，可以方便地构建分页界面、动画侧边栏界面等更多定制界面。

不过，UI设计具备灵活性也是要付出代价的，即更复杂的应用、更多的部件以及更多的实现代码。我们会在第11章和第17章中体会到使用fragment的好处。现在先来感受一下它的复杂性。

7.3 着手开发 CriminalIntent

CriminalIntent应用比较复杂，我们首先开发应用的记录明细部分。完成后的画面如图7-4所示。开发目标看上去是不是很平常？要记住，本章实际是在为后续开发奠定基础。

我们会设计一个名为CrimeFragment的UI fragment来管理图7-4所示的用户界面，再设计一个名为CrimeActivity的activity来托管CrimeFragment实例。

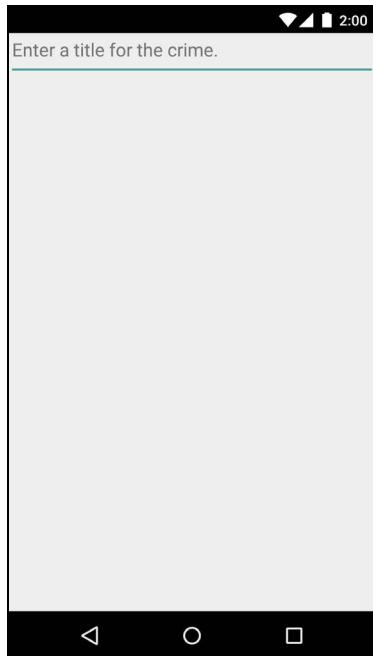


图7-4 本章结束时，CriminalIntent应用的界面

托管可以这样理解：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。

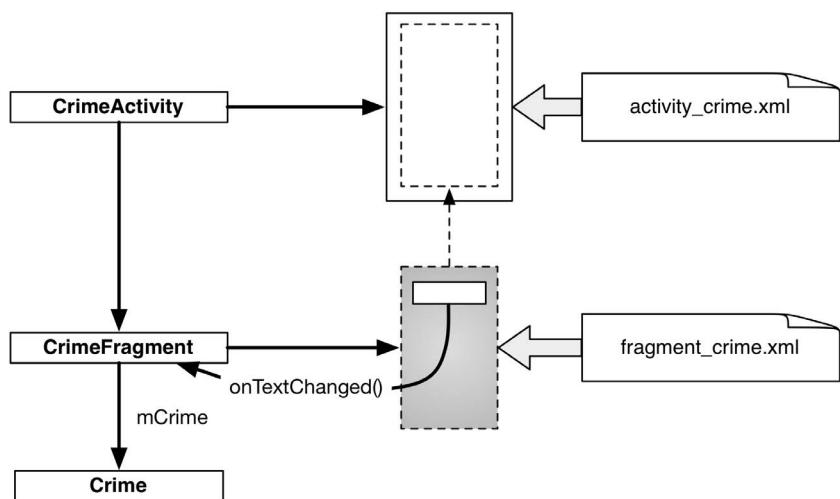
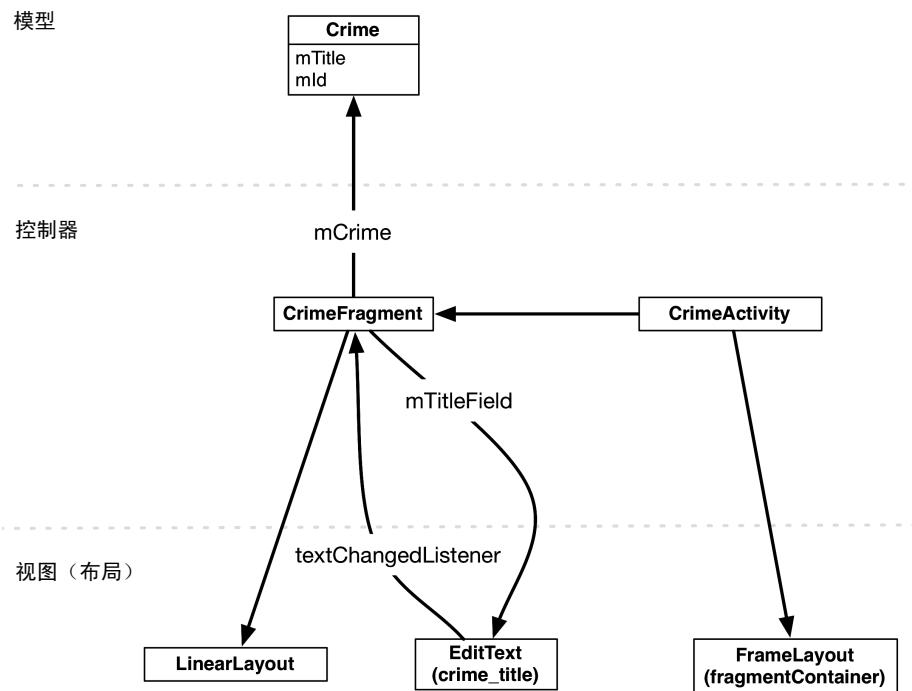


图7-5 CrimeActivity托管着CrimeFragment

CriminalIntent是个大型项目，可通过对象图解来更好地理解它。图7-6展示了CriminalIntent项目涉及的对象以及对象间的关系。可以不去记忆，但整体了解开发目标将非常有助于我们的开发。



7

图7-6 CriminalIntent应用的对象图解（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。

图7-6中的`Crime`、`CrimeFragment`以及`CrimeActivity`是我们要开发的类。

`Crime`实例代表某种办公室陋习。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“向水槽中倾倒有毒物”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位置。

`CrimeFragment`的视图由一个`LinearLayout`组件及一个`EditText`组件组成。`CrimeFragment`类中有个存储`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，就更新模型层的数据。

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File→New Project...菜单项创建新的Android应用。如图7-7所示，命名应用为CriminalIntent，命名包为com.bignerdranch.android.criminalintent。

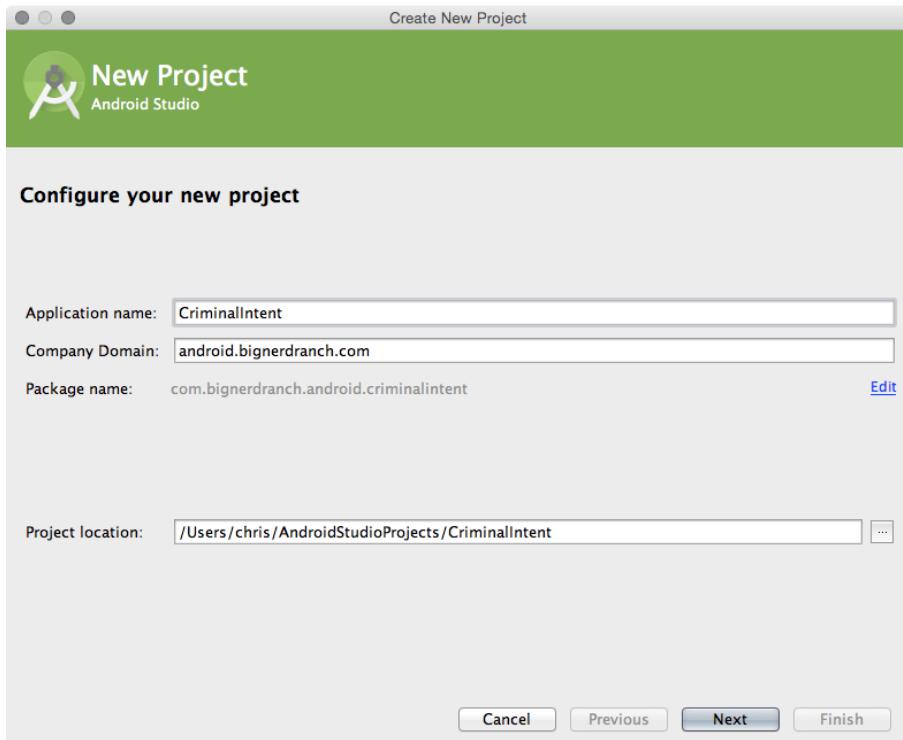
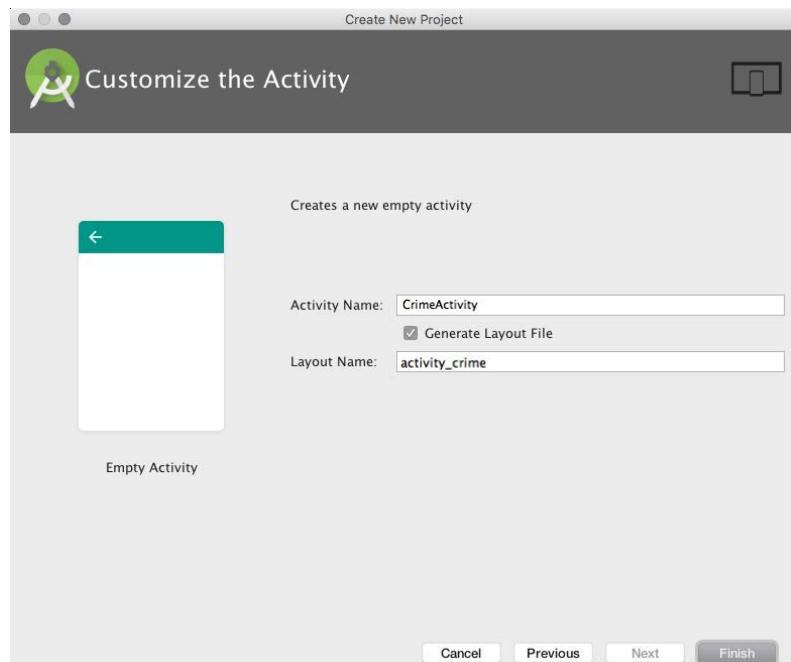


图7-7 创建CriminalIntent应用

单击Next按钮，指定最低SDK版本为API 16: Android 4.1，并确认仅勾选了Phone and Tablet设备类型。

单击Next按钮，选择新增Empty Activity，单击Next继续。

最后，命名activity为CrimeActivity，单击Finish按钮完成，如图7-8所示。



7

图7-8 配置CrimeActivity

7.3.2 fragment 与支持库

随着Android平板设备的发布，为满足平板设备的UI灵活性设计要求，Google在API 11级中引入了fragment。正如本书第1版所述，以前的开发者要考虑支持的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，Android提供了开发支持库，这些开发者仍然可以使用fragment。支持库提供了完整的fragment相关的类实现，最低能兼容支持API 4级。相较于Android操作系统内置的fragment实现，本书选择使用支持库中的fragment实现。考虑到fragment API不断引入新特性以及支持库不断更新的现状，本书这一选择还是比较明智的（进一步解释，请参见7.8节）。如果要使用新特性，升级项目的支持库就行了。注意，使用支持库中的类，不仅是在无原生类的旧版本设备上使用，而且还要代替原生类在新版本设备上使用。

我们要用到两个重要的支持库类，一个是Fragment类（`android.support.v4.app.Fragment`），另一个是FragmentActivity（`android.support.v4.app.FragmentActivity`）。使用fragment的前提是，activity知道如何管理fragment。FragmentActivity类知道如何管理支持版本的fragment。

图7-9显示了这些类的名称及所在位置。既然支持库（以及`android.support.v4.app.Fragment`）直接引入到应用中了，不管是什么版本的设备，安全运行应用自然就得到了保证。

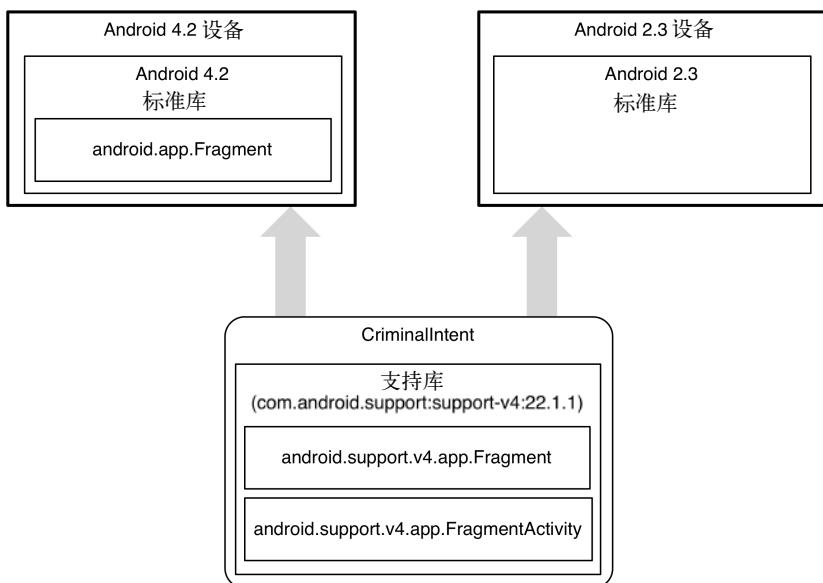


图7-9 新旧版本设备上的fragment支持类

7.3.3 在Android Studio中增加依赖关系

要使用支持库，项目必须将其列入依赖关系。打开应用模块下的build.gradle文件。每个项目都有两个build.gradle文件。一个用于整个项目，另一个用于应用模块。我们要编辑的是app/build.gradle文件。

build.gradle待编辑文件的当前依赖设置类似于代码清单7-1。也就是说，项目依赖于libs目录下的所有jar包。

代码清单7-1 Gradle依赖设置 (app/build.gradle)

```
apply plugin: 'com.android.application'

android {
    ...
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

Gradle允许设置未复制到项目中的依赖项。我们要做的就是预先写好指令，剩下的就交给Gradle了。应用编译时，Gradle会找到并下载依赖包，自动导入到项目中。

考虑到依赖项设置指令复杂难记，Android Studio维护了一份常用库列表。选择File→Project Structure...菜单项打开项目结构对话框。

选择左边的应用模板，然后在右边点击Dependencies选项页。可以看到，应用模板的依赖项都列在这了，如图7-10所示。

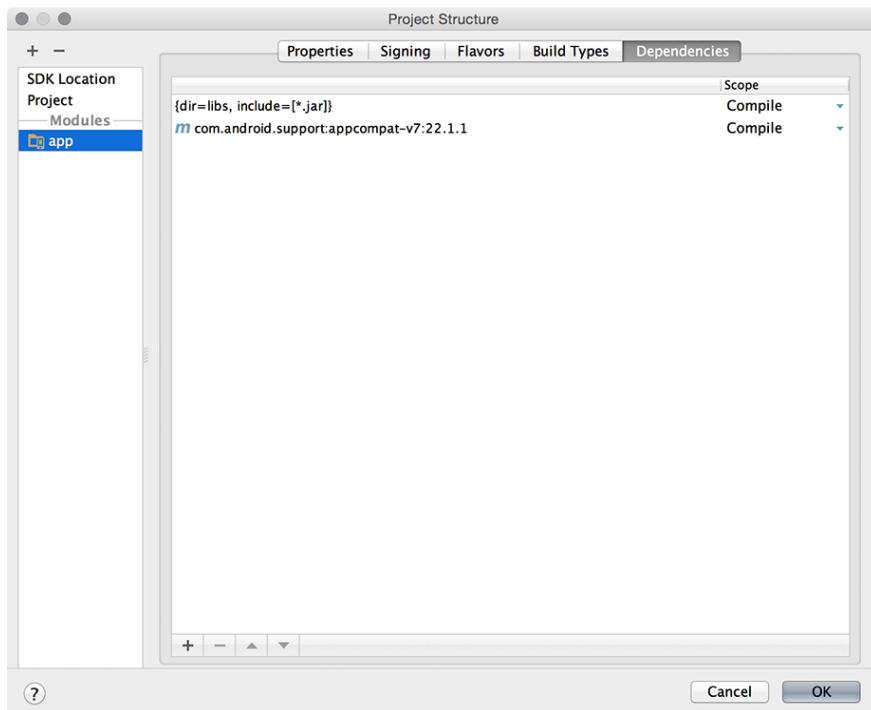


图7-10 app依赖项

你可能已经指定了AppCompat这样的依赖项。不用管它，AppCompat库会在第13章详细介绍。单击+号按钮，在选择Library dependency界面添加新的依赖项，如图7-11所示。从列表中选中support-v4库后单击OK按钮确认。

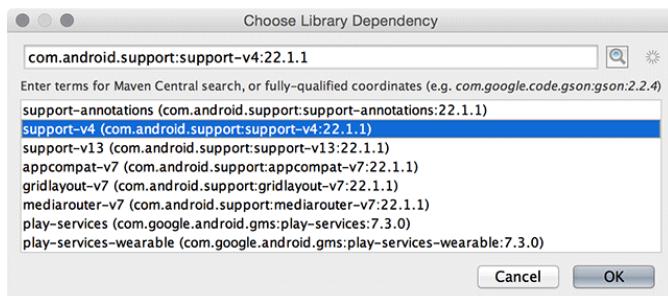


图7-11 可选依赖项

回到app/build.gradle文件的编辑窗口，新的依赖项已经添加完毕，如代码清单7-2所示。

代码清单7-2 Gradle依赖项已更新（app/build.gradle）

```

...
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:22.1.1'
}

```

（如果手工修改了app/build.gradle文件，就需要同步项目和Gradle文件以更新修改内容。同步就是要求Gradle通过下载或删除依赖项，基于修改更新编译。选择Tools→Android→Sync Project with Gradle Files菜单项可手工执行同步。）

代码清单7-2中，加亮部分的依赖项字符串使用了Maven坐标模式：**groupId:artifactId:version**。（Maven是个依赖包管理工具，详见其官方网站<https://maven.apache.org/>。）

groupId通常是类库的基础包名，唯一标识了Maven仓库中的依赖类库，如上例中的com.android.support。

artifactId是包中的特定库名，我们指定的是support-v4。com.android.support包中有很多不同的库，如support-v13、appcompat-v7和gridlayout-v7。Google使用basename-vX模式作为支持库的命名约定。**-vX**指所支持的最低API级别。因此，以appcompat-v7为例，这里的-v7就是说Google兼容库可以应用到Android API 7及以上级别的设备上。

最后一项**version**是指类库的版本号。CriminalIntent应用依赖于版本号为22.1.1的support-v4库。这是本书截稿时的最新版本。当然，我们的项目支持这之后的任一新版本。为了使用更新的API以及受益于bug修正，建议使用最新版本的支持库。假如Android Studio更新了新的支持库，就不要回退了，使用最新版本就好。

既然项目依赖的支持库已设置好了，现在就来用吧。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)和onOptionsItemSelected(MenuItem)实现代码，如代码清单7-3所示。（第13章将介绍如何从头创建CriminalIntent应用的选项菜单。）

代码清单7-3 修改模板代码（CrimeActivity.java）

```

public class CrimeActivity extends AppCompatActivity FragmentActivity {

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

```

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。

7.3.4 创建 Crime 类

在项目工具窗口中，右键单击 com.bignerdranch.android.criminalintent 包，选择 New → Java Class 菜单项。在新建类对话框中，命名类为 Crime，单击 OK 按钮完成。

在随后打开的 Crime.java 中，增加代码清单 7-4 所示的代码。

代码清单 7-4 Crime 类的新增代码 (Crime.java)

```
public class Crime {

    private UUID mId;
    private String mTitle;

    public Crime() {
        // Generate unique identifier
        mId = UUID.randomUUID();
    }
}
```

接下来，为只读成员变量 mId 生成一个获取方法，为成员变量 mTitle 生成获取方法和设置方法。右键单击构造方法下面的空白处，选择 Generate... → Getter 菜单项，然后选择 mId 变量。再选择 Generate... → Getter and Setter 为变量 mTitle 生成获取方法和设置方法。生成的获取方法与设置方法如代码清单 7-5 所示。

代码清单 7-5 已生成的获取方法与设置方法 (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 Crime 类所需的全部代码实现工作。

至此，除了模型层，我们还创建了能够托管支持库 fragment 的 activity。接下来，我们将继续

学习activity托管fragment的具体实现部分。

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7-12展示了fragment的生命周期。类似于activity的生命周期,它具有停止、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。

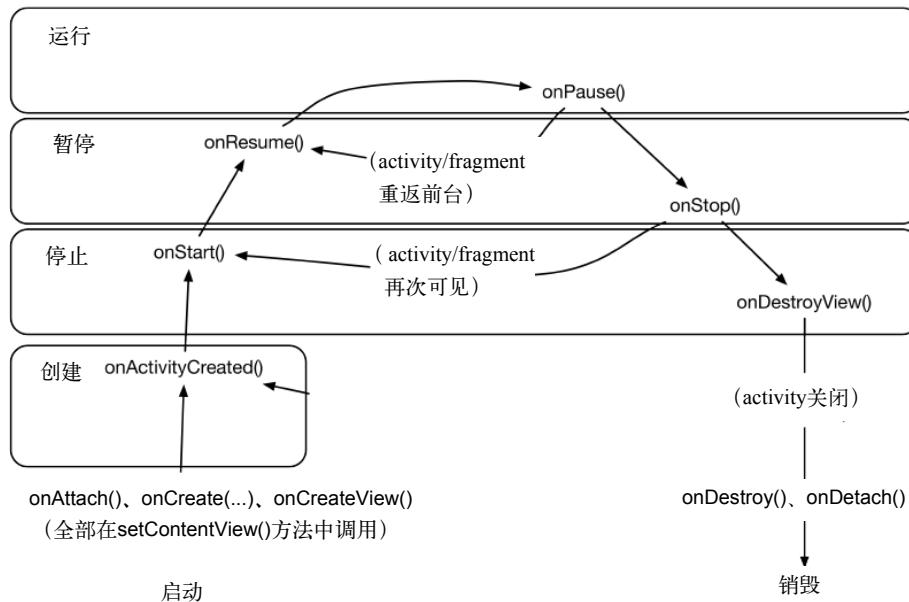


图7-12 fragment的生命周期图解

生命周期方法的对应非常重要。因为fragment代表activity工作,它的状态应该反映activity的状态。显然,fragment需要相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。操作系统不关心activity用来管理视图的fragment。fragment的使用是activity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

activity托管UI fragment有如下两种方式：

- 在activity布局中添加fragment；
- 在activity代码中添加fragment。

第一种方式就是使用布局fragment。这种方式简单但不够灵活。在activity布局中添加fragment，就等同于将fragment及其视图与activity的视图绑定在一起，并且在activity的生命周期过程中，无法切换fragment视图。

第二种方式比较复杂，但也是唯一可以在运行时控制fragment的方式。我们自行决定何时添加fragment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，然后重新添加已移除的fragment。

因而，为追求真正的灵活性UI设计，就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。本章稍后会介绍代码的实现细节。现在，先来学习定义CrimeActivity的布局。

7.4.3 定义容器视图

虽然我们选择在托管activity代码中添加UI fragment，但还是要在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7-13中的FrameLayout所示。

```
FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/fragment_container"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

图7-13 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7-13所示的FrameLayout替换默认布局。完成后的XML文件应如代码清单7-6所示。

代码清单7-6 创建fragment容器布局（activity_crime.xml）

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    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。

现在预览布局文件，或者运行CriminalIntent应用检查实现代码。不过，CrimeActivity还没有托管任何fragment，因此只能看到一个空的FrameLayout，如图7-14所示。

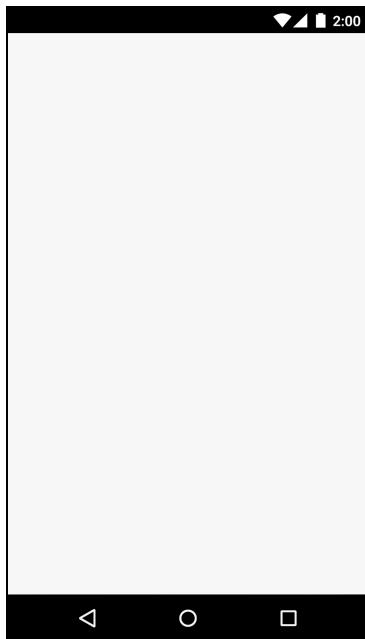


图7-14 一个空的FrameLayout

稍后，我们会编写代码，将fragment的视图放置到FrameLayout中。不过，首先需要创建一个fragment。

7.5 创建UI fragment

创建UI fragment的步骤与创建activity的步骤相同，具体如下：

- 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，组装在布局文件中实例化的组件。

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视图还将包含更多有趣的东西。但在本章，我们只需一个文本栏位来存放crime的标题。

图7-15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组件有一块区域，可供用户添加或编辑文字信息。

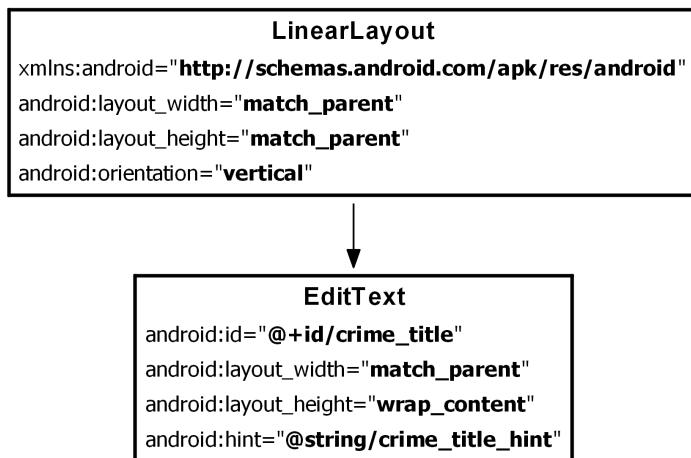


图7-15 CrimeFragment的初始布局

要创建布局文件，在项目工具窗口中，右键单击res/layout文件夹，选择New→Layout resource file菜单项。命名布局文件为fragment_crime.xml。输入LinearLayout作为根元素节点后，单击OK按钮完成创建。

新建文件打开后，查看XML，会发现向导已经添加了LinearLayout。按照图7-15所示，对fragment_crime.xml进行必要的调整。可对照代码清单7-7检查有无差错。

代码清单7-7 fragment视图的布局文件（fragment_crime.xml）

```

<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，结果如代码清单7-8所示。

代码清单7-8 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="action_settings">Settings</string>
<string name="crime_title_hint">Enter a title for the crime.</string>
</resources>

```

保存所有文件。回到文件编辑窗口，预览已完成的fragment_crime.xml布局。

7.5.2 创建 CrimeFragment 类

右键单击com.bignerdranch.android.criminalintent包，选择New→Java Class菜单项。在弹出的新建类对话框中，命名类为CrimeFragment，然后单击OK按钮完成类创建。

接着，让CrimeFragment类继承Fragment类，如代码清单7-9所示。

代码清单7-9 继承Fragment类 (CrimeFragment.java)

```

public class CrimeFragment extends Fragment {
}

```

修改代码继承Fragment类时，Android Studio会找到两个同名Fragment类：Fragment（android.app）和Fragment（android.support.v4.app）。前者是Android操作系统内置版Fragment，后者是支持库版Fragment。我们应选择后者，如图7-16所示。



图7-16 选择支持库中的Fragment类

完成后的代码应该和代码清单7-10一样。

代码清单7-10 导入支持库版Fragment (CrimeFragment.java)

```

package com.bignerdranch.android.criminalintent;

import android.support.v4.app.Fragment;

public class CrimeFragment extends Fragment {
}

```

如果看不到Android Studio的提示框，或者错误地导入了android.app.Fragment类，请先删除导入语句，使用Option+Return（或Alt+Enter）快捷键手工重新导入。千万不要搞错，我们需要的是支持库版Fragment。

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`的生命周期方法完成的。`fragment`的许多生命周期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7-11所示。

代码清单7-11 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();
    }

}
```

需注意以上实现代码中的以下几点。

首先，`Fragment.onCreate(Bundle)`是公共方法，而`Activity.onCreate(Bundle)`是保护方法。`Fragment.onCreate(...)`方法及其他`Fragment`生命周期方法必须是公共方法，因为托管`fragment`的`activity`要调用它们。

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。

另外，`fragment`的视图并没有在`Fragment.onCreate(...)`方法中生成。虽然我们在`Fragment.onCreate(...)`方法中配置了`fragment`实例，但创建和配置`fragment`视图是另一个`fragment`生命周期方法完成的：

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState)
```

该方法实例化`fragment`视图的布局，然后将实例化的`View`返回给托管`activity`。`LayoutInflater`及`ViewGroup`是实例化布局的必要参数。`Bundle`用来存储恢复数据，可供该方法从保存状态下重建视图。

在`CrimeFragment.java`中，添加`onCreateView(...)`方法的实现代码，从`fragment_crime.xml`布局中实例化并返回视图，如代码清单7-12所示。

代码清单7-12 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 container,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, container, 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，我们通常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，因为我们将以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7-13所示。

代码清单7-13 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 container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(
                CharSequence s, int start, int count, int after) {
                // This space intentionally left blank
            }

            @Override
            public void onTextChanged(
                CharSequence s, int start, int before, int count) {
                mCrime.setTitle(s.toString());
            }
        });
    }
}

```

```

    @Override
    public void afterTextChanged(Editable s) {
        // 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中完全一样。创建实现`TextWatcher`监听器接口的匿名内部类，如代码清单7-13所示。`TextWatcher`有三种方法，不过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，我们需要先把`CrimeFragment`添加到`CrimeActivity`。

7.6 添加 UI fragment 到 FragmentManager

在`Fragment`类引入Honeycomb的时候，为协同工作，`Activity`类中相应添加了`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事务回退栈（稍后会介绍）。

`FragmentManager`的图解如图7-17所示。

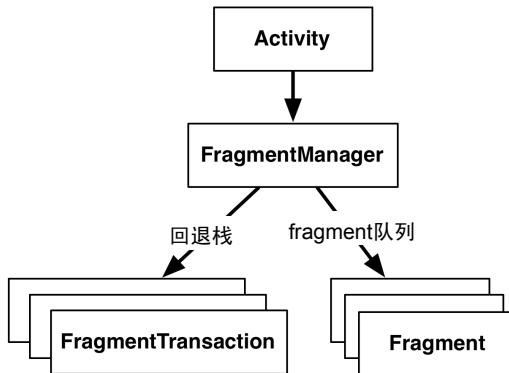


图7-17 FragmentManager图解

在CriminalIntent应用中，我们只需关心FragmentManager管理的fragment队列即可。

要以代码的方式将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7-14所示代码到onCreate(...)方法中。

代码清单7-14 获取FragmentManager (CrimeActivity.java)

```

public class CrimeActivity extends FragmentActivity {

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

        FragmentManager fm = getSupportFragmentManager();
    }
}
  
```

如果添加代码时遇到了错误，记得检查导入语句，看看导入的是不是支持库版本的FragmentManager类。

因为我们使用了支持库及FragmentActivity类，所以这里调用的方法是getSupportFragmentManager()。如果不考虑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FragmentManager之后，添加代码清单7-15所示代码，获取一个fragment并交由FragmentManager管理。（现在只需对照添加，稍后会逐行解读代码。）

代码清单7-15 添加一个CrimeFragment (CrimeActivity.java)

```

public class CrimeActivity extends FragmentActivity {
  
```

```

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

    FragmentManager fm = getSupportFragmentManager();
    Fragment fragment = fm.findFragmentById(R.id.fragment_container);

    if (fragment == null) {
        fragment = new CrimeFragment();
        fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
    }
}
}

```

在代码清单7-15中，理解新增代码的最佳位置并非首行。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：

```

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragment_container, 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关键。FragmentManager管理着fragment事务回退栈。

`FragmentManager.beginTransaction()`方法创建并返回`FragmentTransaction`实例。`FragmentTransaction`类使用了名为fluent interface的接口方法，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，上述阴影部分代码可解读为：“创建一个新的fragment事务，加入一个添加操作，然后提交该事务。”

`add(...)`方法是整个事务的核心，它含有两个参数：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的作用有：

- 告诉`FragmentManager`，`fragment`视图应该出现在`activity`视图的什么位置；
- 用作`FragmentManager`队列中`fragment`的唯一标识符。

如需从`FragmentManager`中获取`CrimeFragment`，使用容器视图资源ID就行了：

```

FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragment_container);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()

```

```
.add(R.id.fragment_container, 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是FragmentManager的内部实现机制。如果要向activity添加多个fragment，通常需要分别为每个fragment创建不同ID的容器。

现在我们从头至尾对代码清单7-15中的新增代码进行总结。

首先，使用R.id.fragment_container的容器视图资源ID，向FragmentManager请求并获取fragment。如果要获取的fragment已存在于队列中，FragmentManager就直接返回它。

为什么要获取的fragment会存在于队列中呢？前面说过，设备旋转或回收内存时，Android会销毁CrimeActivity，而后重建时，会调用CrimeActivity.onCreate(...)方法。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，如果指定容器视图资源ID的fragment不存在，则fragment变量为空值。这时应新建CrimeFragment，并启动一个新的fragment事务，将新建fragment添加到队列中。

CrimeActivity目前托管着CrimeFragment。运行CriminalIntent应用验证这一点，应该可以看到定义在fragment_crime.xml中的视图，如图7-18所示。

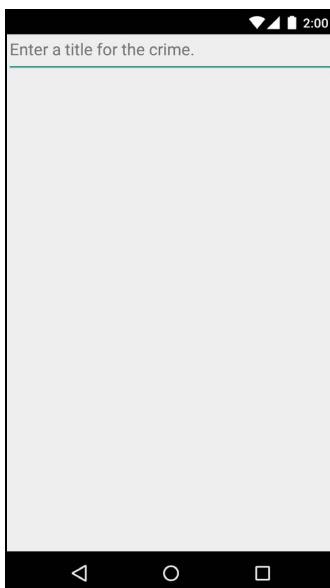


图7-18 CrimeActivity托管的CrimeFragment视图

我们在本章中付出了很多，但是看上去只收获了这么个视图组件。不要沮丧，事实上，本章所做的一切已经为CriminalIntent应用的后续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，有必要重新审视fragment的生命周期，如图7-19所示。

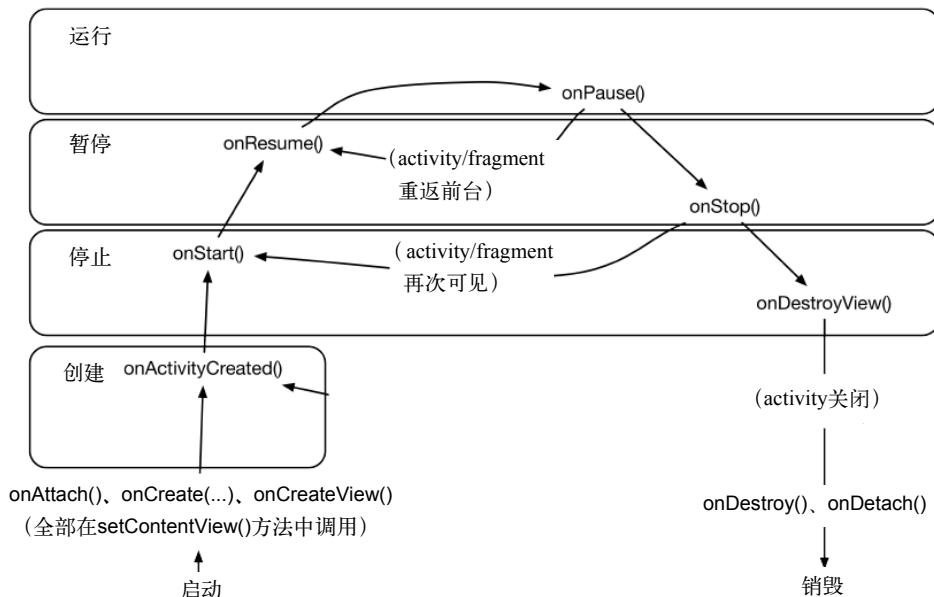


图7-19 再探fragment生命周期

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。

托管activity的`onCreate(...)`方法执行后，`onActivityCreated(...)`方法也会被调用。因为我们正在`CrimeActivity.onCreate(...)`方法中添加`CrimeFragment`，所以fragment被添加后，该方法会被调用。

在activity处于运行状态时，添加fragment会发生什么呢？此种情况下，FragmentManager立即驱使fragment行动，执行必要方法，快速跟上activity的步伐（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()`。

一旦fragment的状态与activity的状态保持了同步，托管activity的FragmentManager就会边接收操作系统的调用指令，边调用其他生命周期方法，以继续保持fragment与activity的状态一致。

7.7 采用fragment的应用架构

应用设计时，正确使用fragment非常重要。然而，许多开发者学习了fragment之后，每当应用涉及复用的组件，就直接采用fragment。这实际是在滥用fragment。

封装可复用组件是Google设计fragment的本意。这里所说的复用组件，是指应用单屏上的组件。如果在单屏使用了大量fragment，不仅应用代码充斥着fragment事务处理，模块的职责分工也会不够分明。遇到这种复用场景，比较好的架构设计是使用定制视图（使用View子类）。

总之，我们要合理使用fragment。一个良好的使用原则是：应用单屏至多使用2~3个fragment，如图7-20所示。



图7-20 少就是多的哲学

使用fragment的理由

自本章开始，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可以不用fragment。用户界面可以只使用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，因此大家应尽快适应它。

有人可能觉得在应用开发初期暂不使用fragment，等到需要时再添加它会好一些。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，YAGNI原则

在诱惑你不去使用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处理。你可能会想到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，但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，既不用担心返工会带来麻烦和痛苦，也不用再去记住应用每个部分使用哪种界面控制风格了。

因而，对于fragment，我们坚持AUF（Always Use Fragments）原则，即“总是使用fragment”。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：为什么应优先使用支持库版 fragment

本书没有使用Android操作系统内置版fragment，而是使用了支持库版fragment。这似乎没有走寻常路。毕竟，Google提供支持库版fragment的本意是方便开发人员在不支持该API的Android老版本上使用fragment；而目前，大多数开发人员都在使用支持fragment的Android版本。

不过我们仍然优先使用支持库版fragment。为什么？因为要升级支持库版fragment的话，我们只需要下载升级包，重新编译发布一个新版本应用就行了。Google每年会多次更新支持库，并借此引入新特性、修复bug。享受这些好处，我们只需要升级项目的支持库版本即可。

举个例子，Google自Android 4.2开始支持fragment嵌套使用（在fragment中托管fragment）。如果开发基于Android操作系统内置版fragment，并且面向Android 4.0及以上版本的设备，应用就无法使用这个新特性了。假如用了支持库版fragment，就能轻松升级应用的支持库版本，随意使用fragment嵌套新特性，只要设备内存足够大。

此外，使用支持库版fragment没有显著的缺点。它和系统内置版本的代码实现几乎完全一样。唯一的真正缺点就是导入支持库包会占用空间。不过考虑到上述诸多优点，牺牲几兆字节空间并不算什么。

本书强调实用工程学，基于我们的应用开发实践，Android支持库堪称无冕之王。

7.9 深入学习：使用操作系统内置版 fragment

如果你坚持不听取以上建议，可以使用Android操作系统内置版fragment。

要使用标准库里的fragment，需对项目进行以下三处调整。

- 放弃使用`FragmentActivity`类，改用标准库中的`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，改用`android.app.Fragment`类。
- 放弃使用`getSupportFragmentManager()`方法，改用`getFragmentManager()`方法获取`FragmentManager`。

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（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();  
    }  
  
    ...  
}
```

Android Studio会找到两个同名Date类。使用Option+Return（或Alt+Enter）快捷键手工导入类。在确认应导入哪个版本的Date类时，选择java.util.Date类。

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的默认发生时间。

接着，为新增变量生成获取方法与设置方法（右键单击文件，选择Generate... → Getter and Setter菜单项），如代码清单8-2所示。

代码清单8-2 已产生的获取方法与设置方法（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文件中实例化并使用这些组件。

8.2 更新布局

8

本章结束时，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：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。Android Studio应该会给出缺少字符串资源的错误提示。暂时忽略，稍后再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。该按钮用于显示Crime的发生日期，显示内容将以代码的方式设置。

为什么要在Button上显示日期呢？这是在为应用的后续开发作准备。现在，crime的发生日期默认为当前日期且不可更改。在第12章，我们将配置按钮组件，实现单击按钮弹出DatePicker

组件以供用户自定义日期。

布局定义中还有一些新的知识点需要探讨，如**style**和**margin**属性。不过我们还是先让添加了新组件的**CriminalIntent**运行起来吧。

打开**res/values/strings.xml**文件，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="crime_title_hint">Enter a title for the crime.</string>
    <string name="crime_title_label">Title</string>
    <string name="crime_details_label">Details</string>
    <string name="crime_solved_label">Solved</string>
</resources>
```

检查确认无拼写错误之后，保存修改过的文件。

8.3 生成并使用组件

接下来，要让**CheckBox**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。

当前，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) {
        ...
    }
```

接着，在**onCreateView(...)**方法中，引用新添加的按钮，设置它的文字属性值为**crime**日期，然后暂时禁用它，如代码清单8-6所示。

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;
}

```

禁用按钮可以确保它不响应用户的单击事件。禁用后，按钮的外观样式也会发生改变（变为灰色），表明它已处于禁用状态。等到第12章设置监听器时，我们会启用它。

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() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            // Set the crime's solved property
            mCrime.setSolved(isChecked);
        }
    });

    return v;
}

```

创建OnCheckedChangeListener时，Android Studio会提供两个导入选项。确认选择的是 android.widget.CompoundButton。

运行CriminalIntent应用。尝试勾选清除CheckBox状态，欣赏一下显示日期的禁用Button吧。

8.4 深入探讨 XML 布局属性

在本节中，我们来回顾fragment_crime.xml文件中添加的一些属性定义，同时解答可能令人困扰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，下列样式资源能够配置组件，让其显示的文字大小大于正常值。

```

<style name="BigTextStyle">
    <item name="android:textSize">20sp</item>

```

```
<item name="android:padding">3dp</item>
</style>
```

我们可以创建自己的样式文件（创建方法请参见第20章）。具体做法是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。

再来看看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，能让屏幕上的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平台主题。例如，在创建CriminalIntent应用时，向导就设置了默认主题（是在manifest文件的application标签下引用的）。

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。在fragment_crime.xml文件中，样式属性值?android:listSeparatorTextViewStyle的使用就是个很好的例子。

使用主题属性引用，就是告诉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，基于特定主题的整体风格，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显示风格。

我们还会在第20章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单位已在之前的布局文件中出现过了，下面我们来具体学习一下。

有时需为视图属性指定大小尺寸值（通常以像素为单位，有时也用点、毫米或英寸）。最常见的属性有：

- 文字大小（text size），指定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。

在2.6节，我们看到，Android使用密度修饰drawable目录（如drawable-xhdpi）下的图像文件自动适配不同像素密度的屏幕。那么问题来了，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 unit）。使用这种单位，可在不同屏幕密度的设备上获得同样的尺寸。无需进行麻烦的转换计算，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。

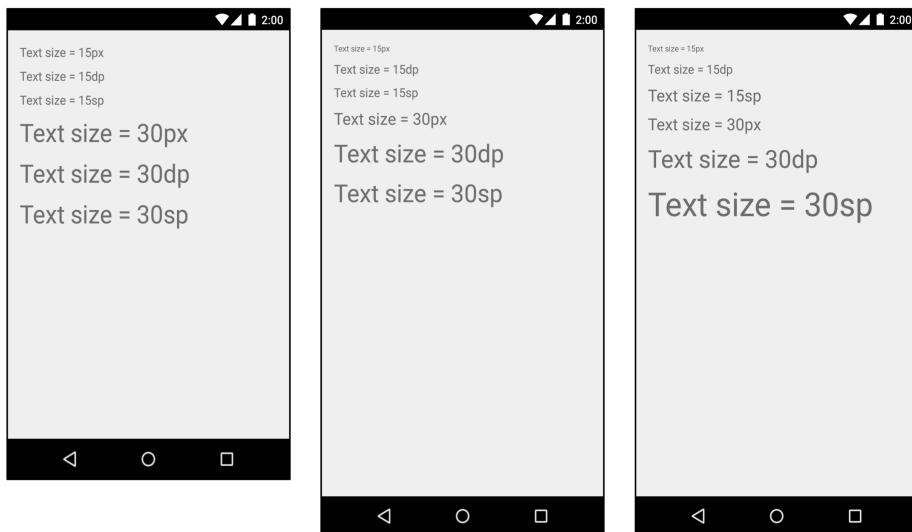


图8-2 应用在Textview上的密度无关尺寸单位（左：MDPI；中：HDPI；右：HDPI+大字体）

dp（或dip）

英文density-independent pixel的缩写，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。如果设备屏幕密度较高，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，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，我们往往只会用到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，我们用16dp单位值设定边距尺寸，如代码清单8-3所示。该单位值的设定遵循了Android的material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，旧版本设备往往无法获得或实现这些功能。不过有些设计原则可通过使用AppCompat库获得支持，第13章中我们会详细介绍它。

8.4.4 布局参数

你是否已经注意到，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`。

不以`layout_`开头的属性作用于组件。组件实例化时，会调用某个方法按照属性及属性值进行自我配置。

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。它们会告诉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：`android:layout_width`和`android:layout_height`。`LinearLayout`实例化时，它的父布局会使用这两个属性。也就是说，`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一无所知，边距必须对该组件的父组件负责。

内边距则并非布局参数。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（如图8-3所示），可将代码清单8-8所示属性添加给`Button`。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"
/>
```

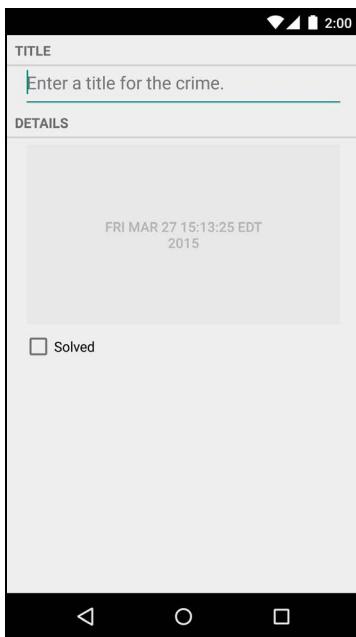


图8-3 实话实说，我喜欢大按钮

很可惜，大按钮虽方便，但在继续学习前，还是应该删除这个属性。

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。本节，我们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身及其子类。不过，默认的调整有时并不能合理利用用户界面空间。

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。

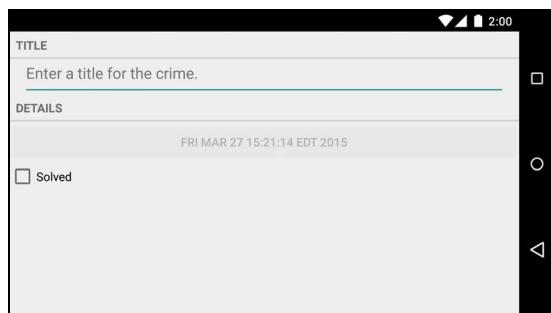


图8-4 水平模式下的CrimeFragment

可以看到，显示日期的按钮变成了一个长条。水平模式下，按钮如果能与单选框并排放置会更加美观。

下面我们切换至图形布局工具进行调整。打开fragment_crime.xml文件，然后选择文件底部的Design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，包含了所有我们可能用到的组件，按类别组织（如图8-5所示）。

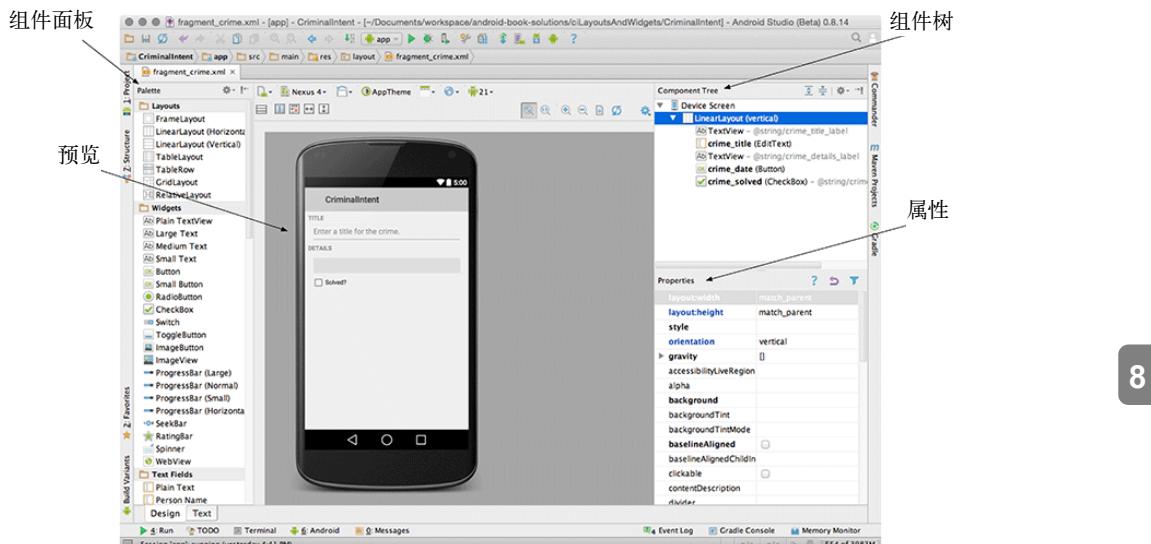


图8-5 图形布局工具中的视图

组件树位于预览视图的右边，表明组件是如何在布局中组织的。

组件树下面是属性视图。在此视图中，我们可以查看并编辑组件树中已选中的组件属性。

8.5.1 创建水平模式布局

可通过图形布局编辑器产生水平模式的布局文件。找到像纸张且右底部带Android图标按钮，点击它并选择Create Landscape Variation菜单项，如图8-6所示。

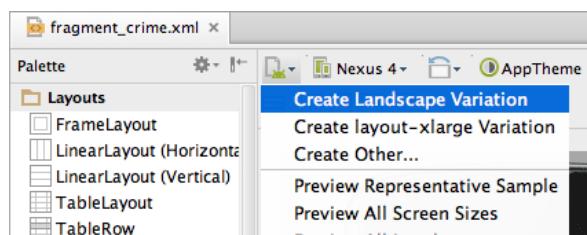


图8-6 创建水平模式布局

可以看到，新建布局出现了。另外，工具还会在后台创建res/layout-land目录，并将当前的fragment_crime.xml复制进去。

现在，参照图8-7，看看要对布局进行哪些调整。

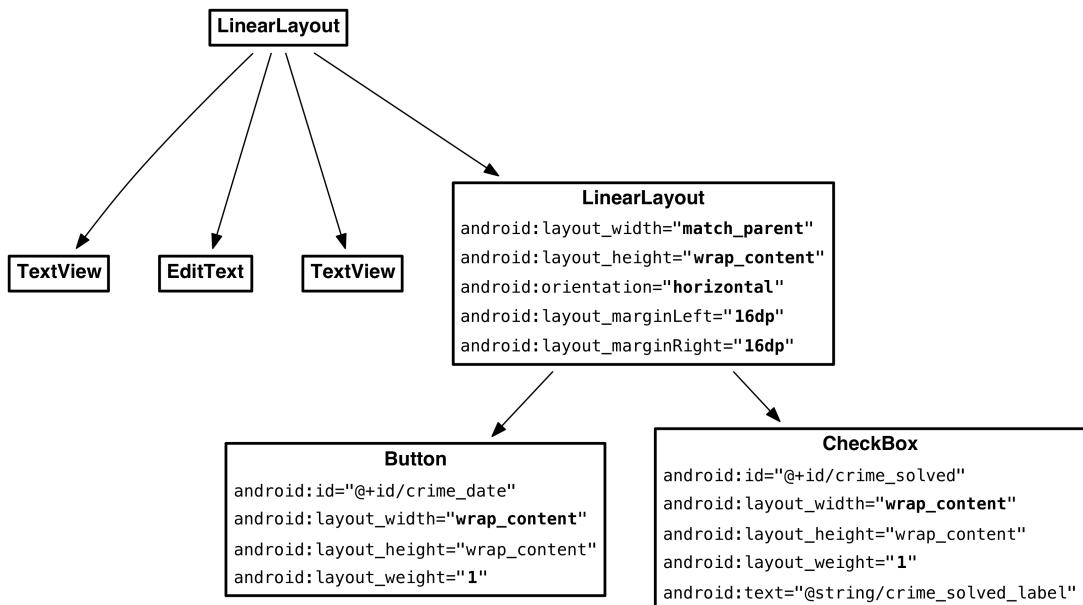


图8-7 CrimeFragment的水平模式布局

需要的调整具体如下：

- 新增一个LinearLayout组件；
- 编辑LinearLayout组件的属性；
- 把Button和CheckBox组件设置为LinearLayout的子组件；
- 更新Button和CheckBox组件的布局参数。

8.5.2 添加新组件

在组件面板中选中目标组件，然后将其拖曳到组件树中，即可完成组件的添加。单击组件面板的布局类别，选中水平的LinearLayout并将其拖曳到组件树中。LinearLayout应位于日期按钮之上，且为根LinearLayout的直接子组件，如图8-8所示。

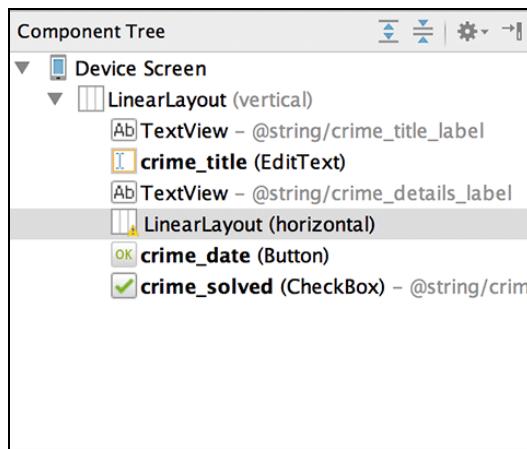


图8-8 添加到fragment_crime.xml中的LinearLayout

要获得想要的组件层级结构，也可以直接把组件从组件面板拖曳添加到预览界面。不过由于布局组件通常是空的或者可能被其他视图遮挡，我们很难判断到底该把组件放在预览视图的哪个部位。显然，拖曳组件到组件树中相对方便些。

8

8.5.3 在属性视图中编辑组件属性

选择组件树中新添加的LinearLayout后，属性视图中会显示出它的属性。查看layout:width和layout:height属性。

修改layout:width属性为match_parent，layout:height属性为wrap_content，如图8-9所示。现在，LinearLayout的宽度正常了，高度也恰到好处地适应了CheckBox和Button按钮的显示。



图8-9 修改layout:width和layout:height属性

我们还需要调整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8-10所示。

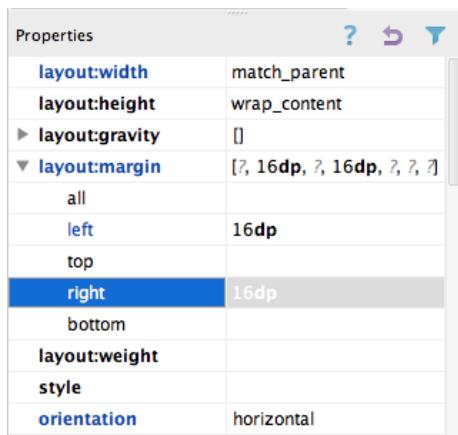


图8-10 在属性视图中设置边距属性

保存布局文件，选中预览界面底部的text标签切换到XML模式。应该可以看到修改了尺寸和边距属性的`LinearLayout`元素。

8.5.4 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，在组件树中，选中`Button`并将其拖曳至`LinearLayout`上。

从组件树可以看出，`Button`现在是新增`LinearLayout`的一个子组件。对`CheckBox`进行同样的操作，结果如图8-11所示。

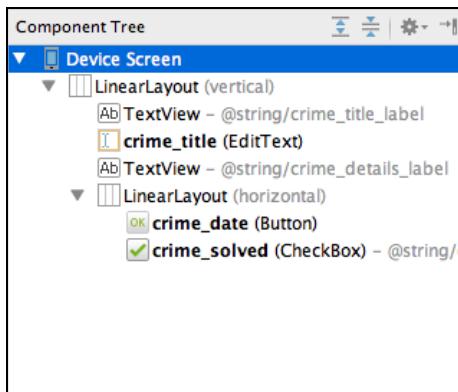


图8-11 Button及CheckBox现在是新增LinearLayout的子组件

如果子组件的摆放不合适，可在组件树中通过拖曳重新放置。当然，也可以在组件树中直接删除布局组件。要当心的是，删除组件会连带删除它的子组件。

回到预览界面，`CheckBox`似乎不见了。这是因为`Button`遮住了它。`LinearLayout`考虑到了

第一个子组件（Button）的宽度属性（`match_parent`），并赋予了它全部空间，导致`CheckBox`没有了立身之地，如图8-12所示。

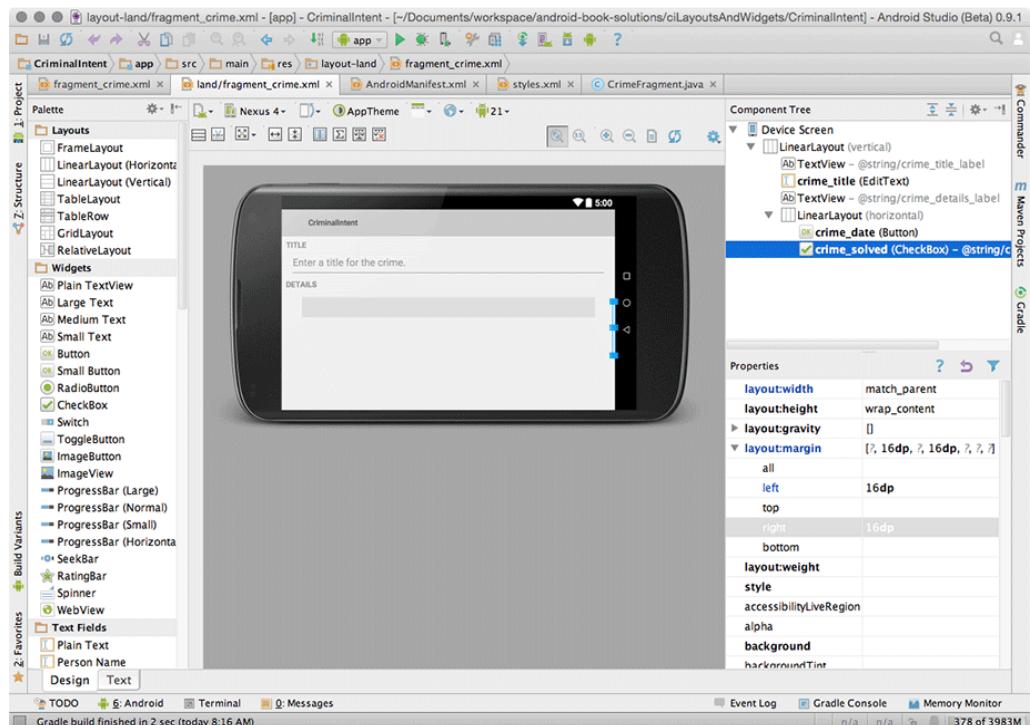


图8-12 Button子组件遮住了CheckBox组件

通过调整子组件的布局参数，可让其他子组件得到`LinearLayout`的平等对待。

8.5.5 更新子组件的布局参数

首先，在组件树中选中日期按钮。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`。

然后，删除按钮左右16dp的边距值。既然按钮已被放置在`LinearLayout`里面，也就不再需要边距值了。

最后，在布局参数区找到`Weight`栏位，设置其值为1。该栏位在XML文件中对应的属性是`android:layout_weight`，如图8-7所示。

在组件树中选中`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文件确认已作的修改。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>
```

运行CriminalIntent应用，旋转设备，好好欣赏水平模式下的布局显示效果吧。

8.5.6 android:layout_weight 属性的工作原理

android:layout_weight属性告诉LinearLayout如何布置安排子组件。我们已经为两个组件设置了同样的值，但这并不意味着它们在屏幕上占据同样的宽度。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8-13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8-13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。）

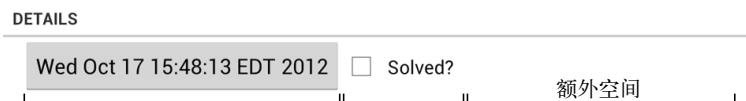


图8-13 第一步：基于layout_width属性值分配的空间大小

第二步，`LinearLayout`依据`layout_weight`属性值进行额外的空间分配，如图8-14所示。



图8-14 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。若将`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8-15所示。



图8-15 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，我们使用的是一种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。

如果想让`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，如图8-16所示。



图8-16 如果`layout_width="0dp"`，则只考虑`layout_weight`属性值

8.5.7 图形布局工具使用总结

图形布局工具非常有用。随着Android Studio各个版本的发布，Android一直在努力改进这一工具。然而，图形布局工具有时却存在着一些问题，因此很难应付复杂的布局界面。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。

对于本书中应用开发的布局创建，可随意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8-7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，请读者视情况自行决定。

8.5.8 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之后，在代码中引用它们。

如果一个组件只存在于一个布局上，则应先在代码中进行空值检查，确认当前方向的组件存在，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。尽管时间戳够用了，但在按钮上显示人们习惯看到的日期应该会更好，如“Jul 22, 2015”。使用`android.text.format.DateFormat`类实例可实现此目标。要用好它，建议先查阅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；也可以自己定制字符串格式。最后我们来挑战一个高级的任务：创建一个包含星期的字符串格式，如“Wednesday, Jul 22, 2015”。

使用RecyclerView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应用以支持显示crime列表。列表会显示每个Crime实例的标题、发生日期以及处理状态，如图9-1所示。



图9-1 crime列表

图9-2展示了CriminalIntent应用在本章的整体规划。

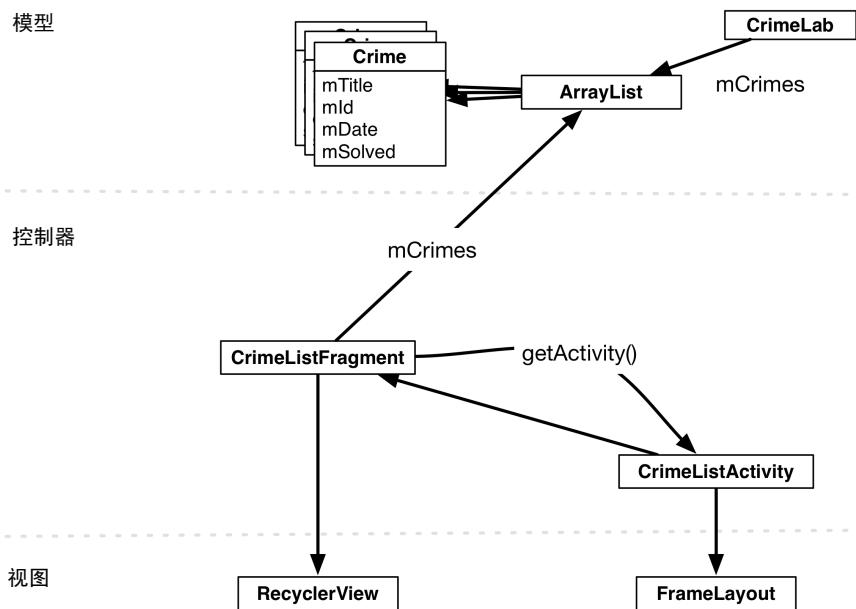


图9-2 CriminalIntent应用对象图

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：`CrimeListActivity`和`CrimeListFragment`。

(图9-2中怎么没有`CrimeActivity`和`CrimeFragment`呢？因为它们是明细视图相关的类，所以这里没有显示它们。第10章，我们将学习如何关联CriminalIntent应用的列表视图和明细视图。)

在图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`RecyclerView`组成。`RecyclerView`类的相关知识稍后会详细介绍。

9.1 升级 CriminalIntent 应用的模型层

首先，我们来升级应用的模型层，从容纳单个`Crime`对象变为可容纳一组`Crime`对象（`Crime`列表）。

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，一个单例类仅允许创建一个实例。

应用能在内存里存多久，单例就能存多久。因此将对象列表保存在单例里的话，就能随时获

取到crime数据，不管activity和fragment的生命周期怎么变化。使用单例还要注意一点：Android从内存里移除应用时，单例对象也就不复存在了。虽然CrimeLab单例不是数据持久保存的好方案，但它确实能保证仅拥有一份crime数据，并且方便了控制层类间的数据传递。

(要进一步了解单例类，请参见9.7节。)

要创建单例，需创建一个带有私有构造方法及get()方法的类。如实例已存在，get()方法就直接返回它；如实例还不存在，get()方法就会调用构造方法创建它。

右键单击com.bignerdranch.android.criminalintent类包，选择New → Java 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。

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()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;

    public static CrimeLab get(Context context) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(context);
        }
        return sCrimeLab;
    }

    private CrimeLab(Context context) {
    }
}
```

9

以上代码有几点需要注意。

首先，注意sCrimeLab变量的s前缀。这是Android开发的命名约定，一看到此前缀，我们就知道sCrimeLab是个静态变量。

其次，再来看CrimeLab的私有构造方法。显然，其他类无法创建CrimeLab对象，因为它们无法调用get()方法。

最后，在get()方法里，我们传入的是Context对象。不过我们在第14章才会用到它。下面，我们往CrimeLab中存储Crime对象。在CrimeLab的构造方法里，创建一个空的List用来保存Crime对象。此外，再添加两个方法：getCrimes()和getCrime(UUID)。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List（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
```

```

public static CrimeLab get(Context context) {
    ...
}

private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
}

public List<Crime> getCrimes() {
    return mCrimes;
}

public Crime getCrime(UUID id) {
    for (Crime crime : mCrimes) {
        if (crime.getId().equals(id)) {
            return crime;
        }
    }
    return null;
}
}

```

`List<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、新增和删除数组中元素的方法。常见的`List`实现是`ArrayList`（利用Java数组存储列表元素）。

既然`mCrimes`含有`ArrayList`，而`ArrayList`也是个`List`，那么对`mCrimes`来说，`ArrayList`和`List`都是有效的类型。对于类似的情况，我们推荐在声明变量的时候使用`List`接口类型。这样，如果有需要，就可以方便地使用其他`List`实现，如`LinkedList`。

`mCrimes`实例化语句使用了Java 7引入的`<>`符号。该符号告诉编译器，`List`中的元素类型可以基于变量声明传入的抽象参数来确定。这里，因为变量声明语句`private List<Crime> mCrimes;`中指定了`Crime`参数，编译器可据此推测出`ArrayList`里可放入`Crime`对象。（Java 7之前，我们必须这么写：`mCrimes = new ArrayList<Crime>();`。）

最后，新建`List`将包含用户自建的`Crime`，用户可自由存取它们。现在，我们在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 context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // Every other one
        mCrimes.add(crime);
    }
}

```

这样，我们就拥有了一个满是数据的模型层（100个`crime`）。

9.2 使用抽象 activity 托管 fragment

创建托管CrimeListFragment的CrimeListActivity类之前，首先为CrimeListActivity创建视图。

9.2.1 通用的 fragment 托管布局

对于CrimeListActivity，我们仍可以使用定义在activity_crime.xml文件中的布局（代码清单9-4）。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。

代码清单9-4 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/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

activity_crime.xml布局文件并没有特别指定fragment，因此对于任何activity托管fragment，都可以使用这个布局文件。下面，为了让该布局更加通用，我们把它重命名为activity_fragment.xml。

在项目工具窗口中，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，Android Studio会自动更新资源文件的所有引用。

Android Studio应该自动更新引用了activity_fragment.xml资源文件。如看到CrimeActivity.java代码有错，则需要在CrimeActivity文件中手工更新引用代码，如代码清单9-5所示。

代码清单9-5 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.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

```

        }
    }
}

```

9.2.2 抽象 activity 类

可以复用CrimeActivity的代码来创建CrimeListActivity类。回顾一下前面写的CrimeActivity类，实现代码简单且近乎通用，如代码清单9-5所示。事实上，CrimeActivity类代码唯一不通用的地方是CrimeFragment类在添加到FragmentManager之前的实例化部分，如代码清单9-6所示。

代码清单9-6 近乎通用的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.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}

```

本书中，几乎每个新建activity都需要同样一段代码。为避免重复，我们将这些重复代码封装为抽象类。

在CriminalIntent类包里创建一个名为SingleFragmentActivity的抽象类。设置超类为FragmentActivity类，如代码清单9-7所示。

代码清单9-7 创建一个Activity抽象类（SingleFragmentActivity.java）

```

public abstract class SingleFragmentActivity extends FragmentActivity {
}

```

然后，按照代码清单9-8添加相关代码。可以看到，除了加亮部分，其余代码和原来的CrimeActivity代码完全一样。

代码清单9-8 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.fragment_container);

    if (fragment == null) {
        fragment = createFragment();
        fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
    }
}
}

```

在以上代码里，我们设置从activity_fragment.xml布局里实例化activity视图。然后在容器中查找FragmentManager里的fragment。如果找不到，就新建fragment并将其添加到容器中。

代码清单9-8与CrimeActivity代码唯一的区别就是，为了实例化新的fragment，我们新增了名为createFragment()的抽象方法。SingleFragmentActivity的子类会实现该方法，来返回由activity托管的fragment实例。

1. 使用抽象类

下面我们来测试使用CrimeActivity抽象类。首先将它的超类改为SingleFragmentActivity。然后，删除onCreate(Bundle)方法，再添加代码清单9-9所示的createFragment()方法。

代码清单9-9 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.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}

```

2. 新建控制类

现在，新建两个控制类：CrimeListActivity和CrimeListFragment。

右键单击com.bignerdranch.android.criminalintent包，选择New→Java Class菜单项，在弹出对话框中，命名类为CrimeListActivity。

修改CrimeListActivity类的超类为SingleFragmentActivity类，并实现createFragment()方法，如代码清单9-10所示。

代码清单9-10 实现CrimeListActivity (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

如果新建类时工具默认加入了其他方法，如onCreate，请手工删除。让SingleFragmentActivity去做它们的工作，保持CrimeListActivity类尽量简单。

接下来是创建CrimeListFragment类。

再次右键单击com.bignerdranch.android.criminalintent包，选择New→Java Class菜单项，在弹出对话框中，命名类为CrimeListFragment，如代码清单9-11所示。

代码清单9-11 实现CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    // Nothing yet
}
```

如代码清单9-10所示，CrimeListFragment类现在有个空结构就可以了。稍后再来处理它。

随着后续章节的深入学习，相信你会发现，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。

3. 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，CriminalIntent应用启动后，用户看到的主界面应该是crime列表，因此还要配置CrimeListActivity为launcher activity。

如代码清单9-12所示，在manifest配置文件中，首先声明CrimeListActivity，然后删除CrimeActivity的launcher activity配置，改配CrimeListActivity为launcher activity。

代码清单9-12 声明CrimeListActivity为launcher activity (AndroidManifest.xml)

```
...
<application>
```

```
    android:allowBackup="true"
    android:icon="@mipmap/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是launcher activity。运行CriminalIntent应用，会看到CrimeListActivity的FrameLayout托管了一个无内容的CrimeListFragment，如图9-3所示。



图9-3 没有内容的CrimeListActivity用户界面

9.3 RecyclerView、Adapter 和 ViewHolder

我们需要CrimeListFragment向用户展示crime列表，这就要用到RecyclerView类。

RecyclerView是ViewGroup的子类，每一个列表项都是作为一个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。

首先来实现简易版的列表项显示，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9-4所示。

在图9-4中，我们可以看到12个TextView。稍后，升级版CriminalIntent应用能支持滑动屏幕查看所有crime项。这是不是意味着要准备100个TextView呢？大声说“不”吧。因为我们有RecyclerView。

为所有列表项创建TextView很容易搞垮应用。可以想象，真实应用要显示的列表项远不止100个，而且TextView显示内容更为复杂。另外，在屏幕上显示单个crime的话，单个View也就够了。因此，完全没必要准备100个视图，按需创建视图对象才是比较合理的解决方案。

RecyclerView就是这么做的。它只创建刚好充满屏幕的12个视图，而不是100个视图。用户滑动屏幕切换视图时，上一个视图会被回收利用。顾名思义，RecyclerView所做的就是回收再利用，循环往复。

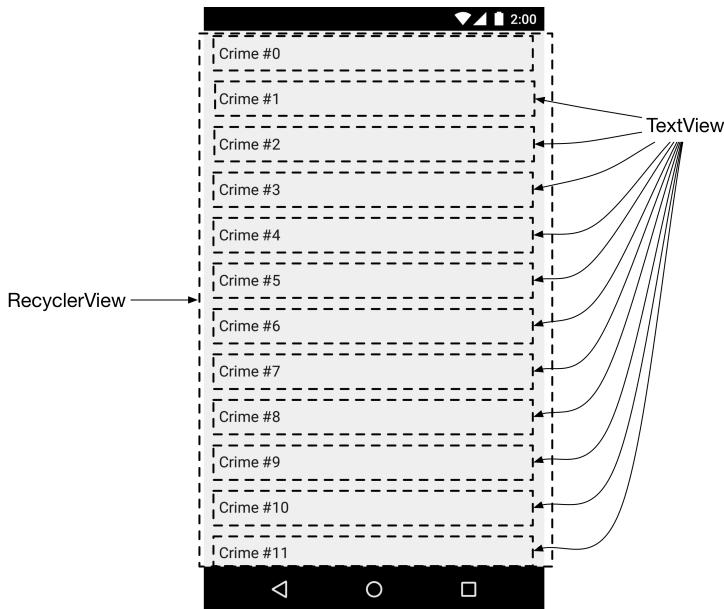


图9-4 带有子TextView的RecyclerView

9.3.1 ViewHolder 和 Adapter

RecyclerView的任务仅限于回收和定位屏幕上的TextView。TextView能够显示数据还离不开另外两个类的支持：Adapter子类和ViewHolder子类。ViewHolder要做的事很少，我们首先讨论它。顾名思义，ViewHolder只做一件事：容纳View视图（如图9-5所示）。

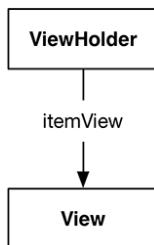


图9-5 没什么地位的ViewHolder

ViewHolder要做的事确实够少了。典型的ViewHolder子类如代码清单9-13所示。

代码清单9-13 典型的ViewHolder子类

```

public class ListRow extends RecyclerView.ViewHolder {
    public ImageView mThumbnail;

    public ListRow(View view) {
        super(view);

        mThumbnail = (ImageView) view.findViewById(R.id.thumbnail);
    }
}
  
```

9

我们可以创建ListRow来获取自定义的mThumbnail和RecyclerView.ViewHolder超类传入的itemView，如代码清单9-14所示。ViewHolder为itemView而生：它引用着我们传给super(view)的整个View视图。

代码清单9-14 ViewHolder的使用示例

```

ListRow row = new ListRow(inflater.inflate(R.layout.list_row, parent, false));
View view = row.itemView;
ImageView thumbnailView = row.mThumbnail;
  
```

RecyclerView自身不会创建视图，它创建的是ViewHolder，而ViewHolder引用着一个个itemView，如图9-6所示。

如果处理的是简单视图，ViewHolder的工作也会相对简单。对于复杂视图，ViewHolder就得处理不同部分的itemView，以简单高效地展示Crime项。稍后，在创建复杂视图时，我们就能一窥其中的奥秘。

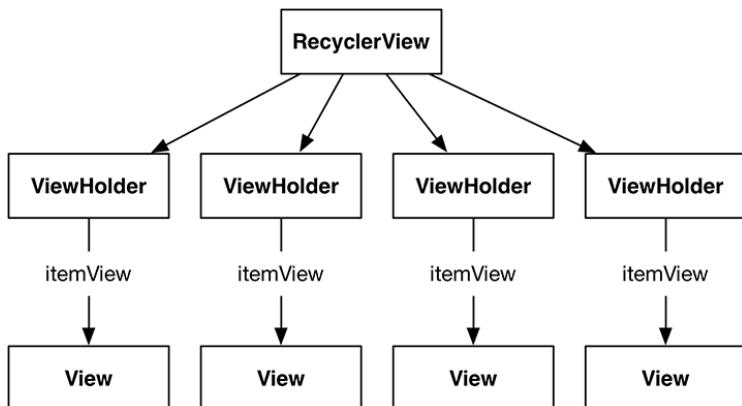


图9-6 ViewHolder配合RecyclerView使用

adapter

图9-6进行了简化，实际上隐藏了一些信息。RecyclerView自己不创建ViewHolder。这个任务实际是由adapter来完成的。adapter是个控制器对象，从模型层获取数据，然后提供给RecyclerView显示，起到了沟通的桥梁作用。

adapter负责：

- 创建必要的ViewHolder；
- 绑定ViewHolder至模型层数据。

要创建adapter，首先要定义RecyclerView.Adapter子类。然后由它封装从CrimeLab获取的crime。

RecyclerView需要显示视图对象时，就会去找它的adapter。图9-7展示了一个RecyclerView可能发起的会话。

首先，通过调用adapter的getItemCount()方法，RecyclerView询问数组列表中包含多少个对象。

接着，RecyclerView调用adapter的createViewHolder(ViewGroup, int)方法创建ViewHolder以及ViewHolder要显示的视图。

最后，RecyclerView会传入ViewHolder及其位置，调用onBindViewHolder(ViewHolder, int)方法。adapter会找到目标位置的数据并绑定到ViewHolder的视图上。所谓绑定，就是使用模型数据填充视图。

整个过程执行完毕，RecyclerView就能在屏幕上显示crime列表项了。需要注意的是，相对于onBindViewHolder(ViewHolder, int)方法，createViewHolder(ViewGroup, int)方法的调用并不频繁。一旦创建了够用的ViewHolder，RecyclerView就会停止调用createViewHolder(...)方法。然后，通过回收利用旧的ViewHolder节约时间和内存。

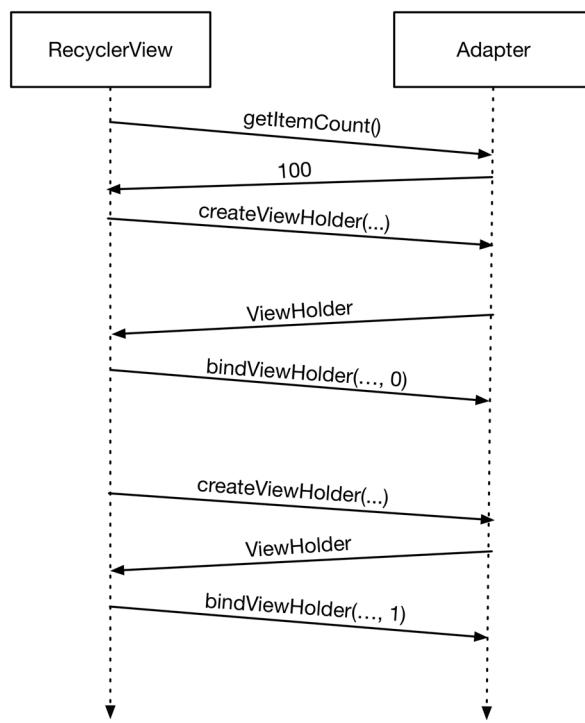


图9-7 生动有趣的RecyclerView-Adapter会话

9

9.3.2 使用 RecyclerView

原理说了这么多，是时候动手实施RecyclerView类了。RecyclerView类来自于Google支持库。要使用它，首先要添加RecyclerView依赖库。

单击File → Project Structure...菜单项切换至项目结构窗口，选择左边的app模块，然后单击Dependencies选项页。单击+按钮弹出依赖库添加窗口。

找到并选择recyclerview-v7支持库，单击OK按钮完成依赖库添加，如图9-8所示。

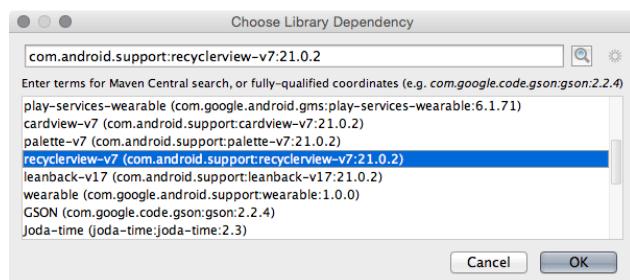


图9-8 添加RecyclerView依赖库

RecyclerView视图需在CrimeListFragment的布局文件中定义。我们首先要创建这个布局文件。右键单击res/layout目录，选择New → Layout resource file菜单项。命名布局文件为fragment_crime_list后单击OK按钮完成。

打开新建的fragment_crime_list布局文件，修改根视图为RecyclerView，并为其配置ID属性，如代码清单9-15所示。

代码清单9-15 在布局文件中添加RecyclerView视图（fragment_crime_list.xml）

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

配置完CrimeListFragment的视图，接下来的任务就是视图和fragment的关联。修改CrimeListFragment类文件，使用布局并找到布局中的RecyclerView视图，如代码清单9-16所示。

代码清单9-16 为CrimeListFragment配置视图（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

        mCrimeRecyclerView = (RecyclerView) view
            .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        return view;
    }
}
```

注意，没有LayoutManager的支持，不仅RecyclerView无法工作，还会导致应用崩溃。所以，RecyclerView视图创建完成后，就立即转交给了LayoutManager对象。

如前所述，RecyclerView类的任务就是回收再利用以及定位屏幕上的TextView视图。实际上，定位的任务被委托给了LayoutManager。除了在屏幕上定位列表项，LayoutManager还负责定义屏幕滚动行为。因此，没有LayoutManager，RecyclerView也就没法正常工作了。Android将来可能会改变这种工作模式，但现在就是如此。

除了一些Android操作系统内置版实现，LayoutManager还有很多第三方库实现版本。我们使用的是LinearLayoutManager类，它支持以竖直列表的形式展示列表项。本书后续章节中还会使用GridLayoutManager类，以网格形式展示列表项。

运行应用，应该还是看不到内容；我们看到的是一个RecyclerView空视图。要显示出crime列表项，还需要完成Adapter和ViewHolder的实现。

9.3.3 实现 Adapter 和 ViewHolder

首先在CrimeListFragment类中定义ViewHolder内部类，如代码清单9-17所示。

代码清单9-17 定义ViewHolder内部类 (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ...
    private class CrimeHolder extends RecyclerView.ViewHolder {
        public TextView mTitleTextView;
        public CrimeHolder(View itemView) {
            super(itemView);
            mTitleTextView = (TextView) itemView;
        }
    }
}
```

可以看到，ViewHolder类引用了用于显示crime标题的TextView视图。这就要求itemView必须是个TextView，否则代码会崩溃。稍后，我们还会让CrimeHolder承担更多责任。

定义完ViewHolder，接下来的任务是创建adapter，如代码清单9-18所示。

代码清单9-18 创建adapter内部类 (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ...
    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
        private List<Crime> mCrimes;
        public CrimeAdapter(List<Crime> crimes) {
            mCrimes = crimes;
        }
    }
}
```

(注意，代码清单9-18的当前代码无法编译通过。这个问题稍后会解决。)

ViewHolder需要被创建或与Crime对象关联时，RecyclerView会和它沟通。RecyclerView不关心也不了解具体的Crime对象，这是Adapter要做的事。

接下来，在CrimeAdapter中实施三个方法，如代码清单9-19所示。

代码清单9-19 武装CrimeAdapter (CrimeListFragment.java)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
```

```

public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
    View view = layoutInflater
        .inflate(android.R.layout.simple_list_item_1, parent, false);
    return new CrimeHolder(view);
}

@Override
public void onBindViewHolder(CrimeHolder holder, int position) {
    Crime crime = mCrimes.get(position);
    holder.mTitleTextView.setText(crime.getTitle());
}

@Override
public int getItemCount() {
    return mCrimes.size();
}
}

```

以上代码值得解读。首先来看onCreateViewHolder方法。

RecyclerView需要新的View视图来显示列表项时，会调用onCreateViewHolder方法。在这个方法内部，我们创建View视图，然后封装到ViewHolder中。此时，RecyclerView并不要求封装视图装载数据。

为得到View视图，我们实例化了Android标准库中名为simple_list_item_1的布局。该布局定义了美观的TextView视图，主要用于列表项的展示。不过，本章稍后会定制更加美观实用的View视图。

接下来要说的是onBindViewHolder方法。该方法会把ViewHolder的View视图和模型层数据绑定起来。收到ViewHolder和列表项在数据集中的索引位置后，我们通过索引位置找到要显示的数据进行绑定。绑定完毕，刷新显示View视图。

所谓索引位置，实际上就是数组中Crime的位置。取出目标数据后，通过发送crime标题给ViewHolder的TextView视图，我们就完成了Crime数据和View视图的绑定。

搞定了Adapter，最后要做的就是将它和RecyclerView关联起来。实现一个设置CrimeListFragment用户界面的updateUI方法，该方法创建CrimeAdapter，然后设置给RecyclerView，如代码清单9-20所示。

代码清单9-20 设置Adapter (CrimeListFragment.java)

```

public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

```

```
mCrimeRecyclerView = (RecyclerView) view
    .findViewById(R.id.crime_recycler_view);
mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

updateUI();

return view;
}

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    mAdapter = new CrimeAdapter(crimes);
    mCrimeRecyclerView.setAdapter(mAdapter);
}

...
}
```

在稍后的章节中，随着对用户界面的配置更为复杂，会向updateUI()中添加更多内容。运行CriminalIntent应用，滚动查看RecyclerView视图。应用运行界面如图9-9所示。

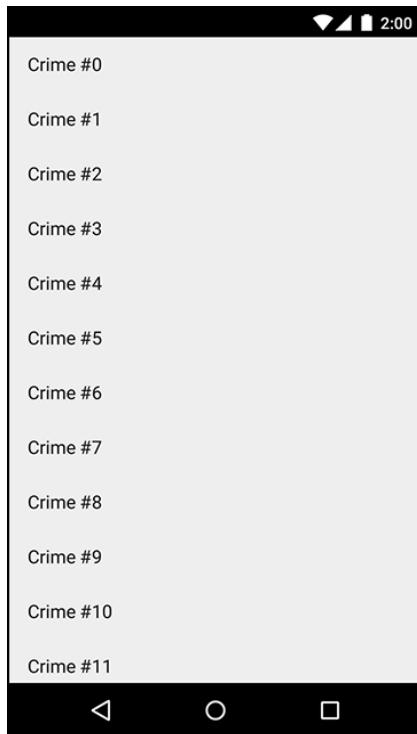


图9-9 看上去还不错的Crime列表项

9.4 定制列表项

当前，每个列表项只是显示了Crime标题（`TextView`简单视图）。

如果还想显示其他列表项信息，或者个性化定制各个列表项，该怎么做呢？创建独立的列表项布局文件就可以了。把列表项视图剥离为独立的布局文件不仅能满足我们的需求，还能保持代码的简洁。

9.4.1 创建列表项布局

按照设想，`CriminalIntent`应用中每个列表项的视图布局应包含crime的三项内容：标题、创建日期，以及是否解决的状态，如图9-10所示。这要求视图布局新增两个`TextView`和一个`CheckBox`。

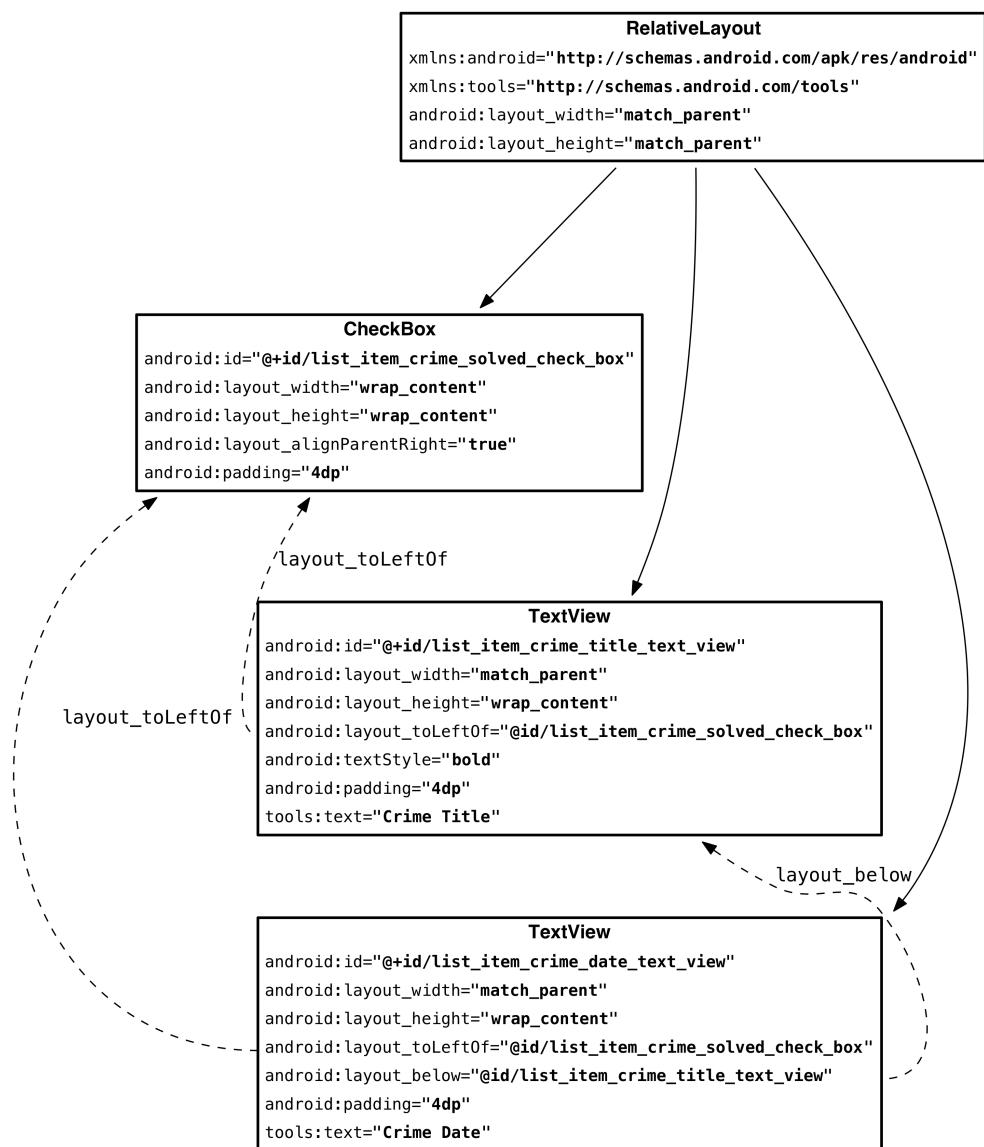


图9-10 漂亮的定制列表项

与创建activity或fragment视图的方法一样，为列表项创建新的布局视图。在项目工具窗口中，右键单击`res/layout`目录，选择`New → Layout resource file`菜单项。在随后出现的对话框中，命名布局文件为`list_item_crime`，设置其根元素为`RelativeLayout`，最后单击OK按钮完成。

在`RelativeLayout`里，需使用一些布局参数控制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。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9-11展示了定制列表项布局的全部组件。`CheckBox`子视图应该首先被定义，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。同样，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（使用`@+id`），其他组件才能在定义时使用它的资源ID（使用`@id`）。

图9-11 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。如果有必要，当然可以使用`+`在其他地方创建ID，但在组件的`android:id`属性中包括ID的话，布局文件更加清晰可读。

定制列表项布局创建就完成了，接下来是更新adapter。

9.4.2 使用定制列表项视图

现在，更新CrimeAdapter类，使用新的list_item_crime布局文件，如代码清单9-21所示。

代码清单9-21 实例化定制布局（CrimeListFragment.java）

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
        View view = layoutInflater
            .inflate(android.R.layout.simple_list_item_1
                    R.layout.list_item_crime, parent, false);
        return new CrimeHolder(view);
    }
    ...
}
```

接下来，该是赋予CrimeHolder更多责任的时候了。修改CrimeHolder，找到TextView标题视图、TextView日期视图以及表明问题处理状态的CheckBox视图，如代码清单9-22所示。

代码清单9-22 在CrimeHolder中寻找视图（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder {
    public TextView mTitleTextView;
    private TextView mTitleTextView;
    private TextView mDateTextView;
    private CheckBox mSolvedCheckBox;

    public CrimeHolder(View itemView) {
        super(itemView);

        mTitleTextView = (TextView) itemView;
        mTitleTextView = (TextView)
            itemView.findViewById(R.id.list_item_crime_title_text_view);
        mDateTextView = (TextView)
            itemView.findViewById(R.id.list_item_crime_date_text_view);
        mSolvedCheckBox = (CheckBox)
            itemView.findViewById(R.id.list_item_crime_solved_check_box);
    }
}
```

这就是ViewHolder闪亮登场的地方。调用findViewById(int)方法开销较大。调用过程中，该方法需要遍历整个itemView找寻目标视图：“嗨，是list_item_crime_title_text_view吗？不是？好吧，打扰了，我再找找看。”这个过程不仅耗时，而且耗费内存。

这个时候就需要ViewHolder的驰援了。通过保存findViewById(int)方法的成果，能够把

宝贵的时间花在`createViewHolder(...)`方法上。因此，等到调用`onBindViewHolder(...)`方法时，一切已经准备就绪。这很好地解决了问题，因为相比`onCreateViewHolder(...)`方法，调用更加频繁的是`onBindViewHolder(...)`方法。

然而，目前的绑定过程有点复杂。添加`bindCrime(Crime)`方法给`CrimeHolder`可以改善这种状况，如代码清单9-23所示。

代码清单9-23 在CrimeHolder中绑定视图 (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder {  
  
    private Crime mCrime;  
  
    ...  
  
    public void bindCrime(Crime crime) {  
        mCrime = crime;  
        mTitleTextView.setText(mCrime.getTitle());  
        mDateTextView.setText(mCrime.getDate().toString());  
        mSolvedCheckBox.setChecked(mCrime.isSolved());  
    }  
}
```

现在，获取到`Crime`，`CrimeHolder`就能刷新显示`TextView`标题视图、`TextView`日期视图和表明问题解决状态的`CheckBox`视图了。

万事俱备，就看`CrimeHolder`表现了。最后，修改`CrimeAdapter`类使用`bindCrime`新方法，如代码清单9-24所示。

代码清单9-24 关联CrimeAdapter和CrimeHolder (CrimeListFragment.java)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {  
  
    ...  
  
    @Override  
    public void onBindViewHolder(CrimeHolder holder, int position) {  
        Crime crime = mCrimes.get(position);  
        holder.mTitleTextView.setText(crime.getTitle());  
        holder.bindCrime(crime);  
    }  
  
    ...  
}
```

运行CriminalIntent应用，查看定制列表项，如图9-12所示。



图9-12 使用定制列表项的用户界面

9.5 响应点击

受益于RecyclerView的内置特性，列表项能够响应用户的点击。在第10章中，用户点击列表项时，应用支持弹出新界面显示crime明细信息。现在，先实现弹出一个Toast消息。

你应该已经注意到了，尽管RecyclerView功能强大，但它实际上只专注于做好本职工作。（这对我们而言也是个好榜样。）因此，处理触摸事件要我们自己动手做。当然，如果真的需要，RecyclerView也能帮你转发触摸事件；不过大多数时候没有必要这样做。

很自然，我们想到的常用解决方案就是：设置OnClickListener监听器。既然列表项视图都关联有ViewHolder，就可以让ViewHolder为它监听用户触摸事件。

修改CrimeHolder类来处理用户点击事件，如代码清单9-25所示。

代码清单9-25 检测用户点击事件（CrimeListFragment.java）

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ...
    public CrimeHolder(View itemView) {
        super(itemView);
        itemView.setOnClickListener(this);
    }
    ...
}
```

```

...
@Override
public void onClick(View v) {
    Toast.makeText(getActivity(),
        mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
        .show();
}
}

```

在以上代码中，`CrimeHolder`类实现了`OnClickListener`接口；而对于`itemView`来说，`CrimeHolder`承担了接收用户点击事件的任务。

运行CriminalIntent应用。点击某个列表项，可看到弹出的Toast响应消息。

9.6 深入学习：ListView 和 GridView

Android操作系统核心库包含`ListView`、`GridView`和`Adapter`类。Android 5.0之前，创建列表项或网格项都应该优先使用这些类。

这些类的API与`RecyclerView`非常相似。`ListView`和`GridView`不关心具体的展示项，只负责展示项的滚动。`Adapter`负责创建列表项的所有视图。不过，使用`ListView`和`GridView`不一定非要使用`ViewHolder`模式（虽然可以并且应该使用）。

过去传统的实现方式现已被`RecyclerView`的实现方式取代，因为我们不用再费力地去调整`ListView`和`GridView`的工作行为了。

举例来说，`ListView`API不支持创建水平滚动的`ListView`，我们需要许多额外的定制工作。使用`RecyclerView`时，虽然创建定制布局和滚动行为需要额外的工作，但`RecyclerView`天生支持拓展，所以使用体验还不错。

此外，`RecyclerView`还有支持列表项动画效果的优点。如果要让`ListView`和`GridView`支持添加和删除列表项的动画效果，实施任务既复杂又容易出错；而对于天生支持动画特效的`RecyclerView`来说，对付这些任务简直是小菜一碟。

口吐莲花，不如直接秀代码。例如，如果`crime`列表项要从位置0移动到位置5，下面这段代码就可以做到：

```
mRecyclerView.getAdapter().notifyItemMoved(0, 5);
```

9.7 深入学习：单例

Android开发实践中，经常会用到`CrimeLab`中使用过的单例模式。然而，单例使用不当的话，会导致应用难以维护，因此它也常遭人诟病。

Android开发常用到单例的一大原因是，它们比`fragment`或`activity`活得久。例如，在设备旋转或是在`fragment`和`activity`间跳转的场景下，单例不会受到影响，而旧的`fragment`或`activity`已经不复存在了。

单例能方便地存储控制模型层对象。假设有个比CriminalIntent更为复杂的CriminalIntent应用，它的许多个activity和fragment会修改crime数据。某个控制单元修改了crime数据之后，怎么保证发送给其他控制单元的是最新数据呢？如果CrimeLab掌控数据对象，所有的修改都由它来处理，是不是数据的一致性控制就容易多了？而且，在控制单元间流转时，我们还可以给每个crime添加ID标识，让控制单元使用ID标识从CrimeLab获取完整的crime数据。

再来谈谈单例的缺点。举个例子，虽然单例能存储数据，活得比控制单元长久，但这并不代表它能永存。在我们切换至其他应用，又逢Android回收内存时，单例连同那些实例变量也就不复存在了。结论很明显：单例无法做到持久存储。（将文件写入磁盘或是发送到Web服务器是不错的数据持久化存储方案。）

单例还不利于单元测试。例如，如果应用代码直接调用CrimeLab对象的静态方法，测试时以模拟版本的CrimeLab代替实际CrimeLab实例就不太现实。实践中，Android开发人员会使用依赖注入工具解决这个问题。这个工具允许以单例模式使用对象，对象也可以按需替换。

单例使用很方便，因而很容易被滥用。在想用就用，想存就存之前，希望你能深思熟虑：数据究竟用在哪里？用在哪里能真正解决问题？

假如不慎重对待这个问题，很可能后来人在查看你的单例代码时，就像打开了一个满是乱糟糟废品的抽屉：废电池、拉链扣、旧照片，等等。它们有什么存在的意义？再强调一次：请确保有充足的理由使用单例模式存储你的共享数据！

使用得当，单例就是拥有优秀架构的Android应用中的关键部件。

使用fragment argument

10

本章，我们将关联CriminalIntent应用的列表与明细部分。用户点击某个crime列表项时，会创建一个托管CrimeFragment的CrimeActivity，以展现Crime实例的明细信息。

在GeoQuiz应用里，我们已实现从activity (QuizActivity) 中启动activity (CheatActivity)。在CriminalIntent应用里，我们要实现从fragment中启动CrimeActivity。准确地说，是从CrimeListFragment中启动CrimeActivity实例，如图10-1所示。

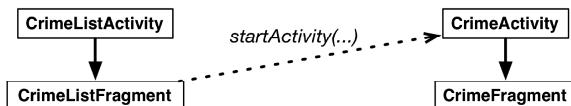


图10-1 从CrimeListActivity中启动CrimeActivity

10

10.1 从 fragment 中启动 activity

从fragment中启动activity类似于从activity中启动activity。我们调用`Fragment.startActivity(Intent)`方法，然后在后台触发调用对应的Activity方法。

在CrimeListFragment的CrimeHolder类里，用启动CrimeActivity实例的代码，替换Toast消息处理代码，如代码清单10-1所示。

代码清单10-1 启动CrimeActivity (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
        Toast.makeText(getActivity(),
            mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
            .show();
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        startActivity(intent);
    }
}
```

```
    }  
}
```

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，托管CrimeFragment的新CrimeActivity随即出现在屏幕上，如图10-2所示。

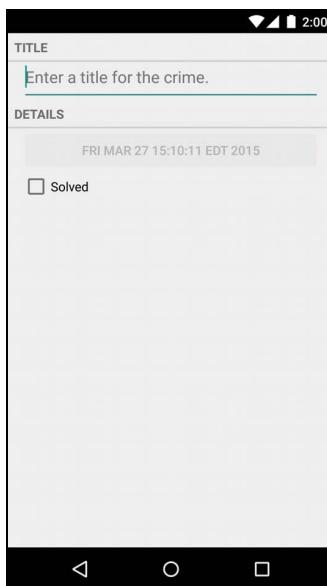


图10-2 空白的CrimeFragment

由于不知道该显示哪个Crime对象，CrimeFragment也就没有显示出具体Crime信息。

10.1.1 附加 extra 信息

启动CrimeActivity时，传递附加到Intent extra上的crime ID，CrimeFragment就能知道该显示哪个Crime。

这需要在CrimeActivity中新增newIntent方法，如代码清单10-2所示。

代码清单10-2 创建newIntent方法（CrimeActivity.java）

```
public class CrimeActivity extends SingleFragmentActivity {  
  
    public static final String EXTRA_CRIME_ID =  
        "com.bignerdranch.android.criminalintent.crime_id";  
  
    public static Intent newIntent(Context packageContext, UUID crimeId) {
```

```

Intent intent = new Intent(packageContext, CrimeActivity.class);
intent.putExtra(EXTRA_CRIME_ID, crimeId);
return intent;
}

...
}

```

创建了显式intent后，调用putExtra(...)方法，传入匹配CrimeId的字符串键与键值。这里，由于UUID是Serializable对象，我们需要调用putExtra(String, Serializable)方法。

更新CrimeHolder，使用newIntent新方法，如代码清单10-3所示。

代码清单10-3 传递crime实例（CrimeListFragment.java）

```

private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {

    ...
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}

```

10.1.2 获取 extra 信息

crime ID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intent中的数据：一种简单直接，另一种复杂但比较灵活（涉及fragment argument的概念）。

我们首先来看简单的方式：CrimeFragment直接使用getActivity()方法获取CrimeActivity的intent。返回至CrimeFragment类，得到CrimeActivity的intent内的extra信息后，再用它获取Crime对象，如代码清单10-4所示。

代码清单10-4 获取extra数据并取得Crime对象（CrimeFragment.java）

```

public class CrimeFragment extends Fragment {
    ...
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
        UUID crimeId = (UUID) getActivity().getIntent()
            .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
    }
}

```

```

        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }

    ...
}

```

在代码清单10-4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，再利用它从`CrimeLab`单例中调取`Crime`对象。

10.1.3 使用 Crime 数据更新 CrimeFragment 视图

既然获取到了`Crime`对象，`CrimeFragment`的视图便可显示该`Crime`对象的数据了。参照代码清单10-5，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10-5 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。

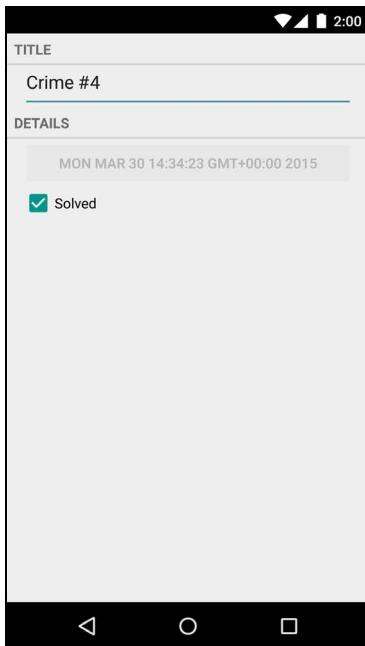


图10-3 Crime #4列表项的明细内容

10.1.4 直接获取 extra 信息的缺点

只需几行简单的代码，就可让fragment直接从托管activity的intent中获取信息。然而，这种方式破坏了fragment的封装性。CrimeFragment不再是可复用的构建单元，因为它现在由某个特定的activity托管着，该特定activity的Intent又定义了名为`com.bignerdranch.android.criminalintent.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crime ID存储在属于CrimeFragment的某个地方，而不是保存在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。属于fragment的“某个地方”实际就是它的argument bundle。

10.2 fragment argument

每个fragment实例都可附带一个Bundle对象。该bundle包含有键值对，我们可以像附加extra到Activity的intent中那样使用它们。一个键值对即一个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

要附加argument bundle给fragment，需调用Fragment.setArguments(Bundle)方法。而且，还必须在fragment创建后、添加给activity前完成。

为满足以上要求，Android开发者采取的习惯做法是：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，转而调用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。

在CrimeFragment类中，编写可以接受UUID参数的newInstance(UUID)方法，如代码清单10-6所示。通过该方法，创建argument bundle和fragment实例，然后附加argument给fragment。

代码清单10-6 编写newInstance(UUID)方法（CrimeFragment.java）

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckbox;

    public static CrimeFragment newInstance(UUID crimeId) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_CRIME_ID, crimeId);

        CrimeFragment fragment = new CrimeFragment();
        fragment.setArguments(args);
        return fragment;
    }

    ...
}
```

现在，需创建CrimeFragment时，CrimeActivity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，并传入CrimeFragment.newInstance(UUID)方法，如代码清单10-7所示。

既然其他类不会用到EXTRA_CRIME_ID，可以将其改为私有。(注意，为了方便，我们直接删除并替换了第一行，实际上只要改public为private就可以了。)

代码清单10-7 使用newInstance(UUID)方法 (CrimeActivity.java)

```
public class CrimeActivity extends SingleFragmentActivity {

    public static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    ...

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        return CrimeFragment.newInstance(crimeId);
    }

}
```

注意，activity和fragment不需要也无法同时相互保持独立性。CrimeActivity必须了解CrimeFragment的内部细节，比如知道它内部有个newInstance(UUID)方法。这很正常。托管activity应该知道这些细节，以便托管fragment；但fragment就不一定要知道其托管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

10.2.2 获取 argument

fragment需要获取它的argument时，会先调用Fragment类的getArguments()方法，再调用Bundle的限定类型“get”方法，如getSerializable(...)方法。

现在回到CrimeFragment.onCreate(...)方法中，改为从fragment的argument中获取UUID，如代码清单10-8所示。

代码清单10-8 从argument中获取crime ID (CrimeFragment.java)

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

    UUID crimeId = (UUID) getActivity().getIntent()
            .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);

    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);

}
```

运行CriminalIntent应用。虽然运行结果一样，但我们应该感到由衷地高兴。这是因为我们不仅保持了CrimeFragment类的通用性，还为下一章实现更为复杂的列表项导航打下了良好基础。

10.3 刷新显示列表项

运行CriminalIntent应用，点击某个列表项，然后修改对应的Crime明细信息。这些修改数据已保存至模型层，但返回列表后，RecyclerView视图并没有刷新。下面就来处理这个问题。

模型层保存的数据如有变化（或可能发生变化），应通知RecyclerView的Adapter，以便其及时获取最新数据并刷新显示列表项。在恰当的时机，与系统的ActivityManager回退栈协同运作，可实现列表项的刷新。

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随即弹出栈外并被销毁。此时，CrimeListActivity立即重新启动并恢复运行。应用的回退栈如图10-4所示。

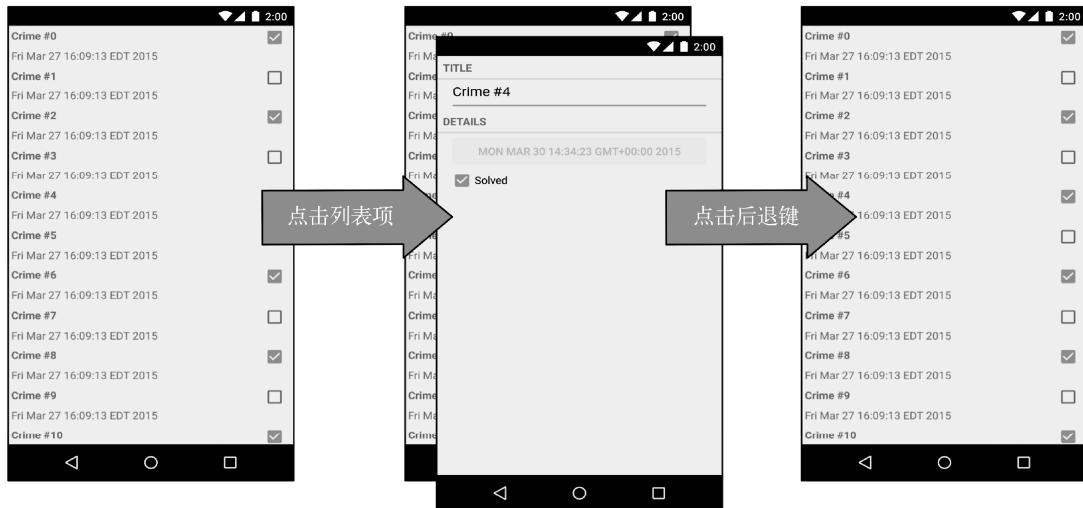


图10-4 CriminalIntent应用的回退栈

CrimeListActivity恢复运行后，操作系统会发出调用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。这里的fragment就是指CrimeListFragment。

在CrimeListFragment中，覆盖onResume()方法，触发调用updateUI()方法刷新显示列表项，如代码清单10-9所示。如果已配置好CrimeAdapter，就调用notifyDataSetChanged()方法来修改updateUI()方法。

代码清单10-9 在onResume()方法中刷新列表项 (CrimeListFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
}

@Override
public void onResume() {
    super.onResume();
    updateUI();
}

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }
}
```

为什么选择覆盖onResume()方法来刷新列表项显示，而不用onStart()方法呢？当有其他activity位于我们的activity之前时，我们无法确定自己的activity是否会被停止。如果前面的activity是透明的，我们的activity可能会被暂停。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，可以看到，列表项已经刷新。

经过前两章的开发，CriminalIntent应用已获得大幅更新。现在，我们来看看升级后的应用对象图解，如图10-5所示。

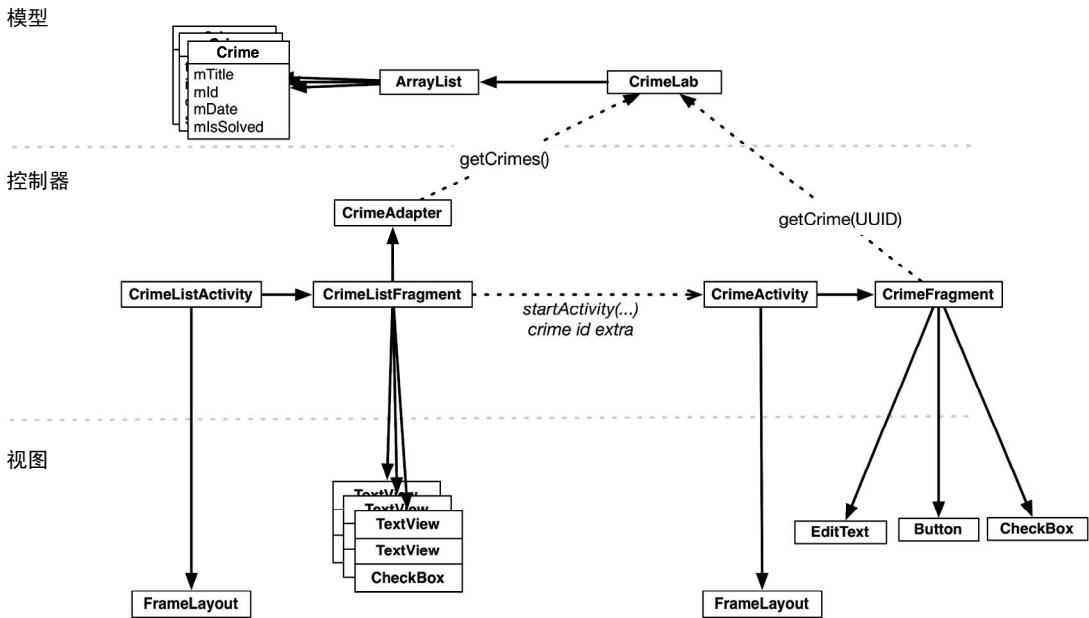


图10-5 应用对象图解更新版

10.4 通过 fragment 获得返回结果

如需从已启动的activity获取返回结果，可使用与GeoQuiz应用中类似的实现代码。具体的代码调整就是：不调用Activity的startActivityForResult(...)方法，转而调用Fragment.startActivityForResult(...)方法；不覆盖Activity.onActivityResult(...)方法，转而覆盖Fragment.onActivityResult(...)方法。

```
public class CrimeListFragment extends Fragment {  
  
    private static final int REQUEST_CRIME = 1;  
  
    ...  
  
    private class CrimeHolder extends RecyclerView.ViewHolder  
        implements View.OnClickListener {  
        ...  
  
        @Override  
        public void onClick(View v) {  
            Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());  
            startActivityForResult(intent, 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)`方法类似于Activity的同名方法。

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，但其自身无法持有返回结果。只有activity拥有返回结果。因此，尽管Fragment有自己的`startActivityForResult(...)`和`onActivityResult(...)`方法，但却没有`setResult(...)`方法。

相反，我们应让托管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);
    }
}

```

10.5 挑战练习：实现高效的 RecyclerView 刷新

10

`Adapter`的`notifyDataSetChanged`方法会通知`RecyclerView`刷新全部的可见列表项。

在CriminalIntent应用里，这个方法不够高效。我们知道，返回`CrimeListFragment`时，最多只有一个`Crime`实例会发生变化。

只需要刷新列表项中的单个`crime`项的话，应该使用`RecyclerView.Adapter`的`notifyItemChanged(int)`方法。修改代码调用这个方法很简单，但如何定位并刷新具体位置的列表项呢？这是个挑战！

10.6 深入学习：为何要用 fragment argument

fragment argument的使用还是有点复杂。为什么不直接在`CrimeFragment`里创建一个实例变量呢？

创建实例变量的方式并不可靠。因为，在操作系统重建fragment时，设备配置发生改变时，用户暂时离开当前应用时，甚至操作系统按需回收内存时，任何实例变量都不复存在了。尤其是内存不够，操作系统强制杀掉应用的情况，可以说是无人能挡。

因此，可以说，fragment argument就是为应对上述场景而生。

我们还有另一个方法应对上述场景，那就是使用实例状态保存机制。具体来说，就是将crime ID赋值给实例变量，然后在`onSaveInstanceState(Bundle)`方法保存下来。要用时，从`onCreate(Bundle)`方法中的`Bundle`中取回。

然而，这种解决方案维护成本高。举例来说，若干年后，你要修改fragment代码添加其他argument，很可能会忘记在`onSaveInstanceState(Bundle)`方法里保存新增argument。

Android开发者更喜欢fragment argument这个解决方案，因为这种方式很清楚直白。若干年后，再回头修改老代码时，只需一眼就能知道，crime ID是以argument保存和传递使用的。即使要新增argument，也会记得使用argument bundle保存它。

使用ViewPager

11

本章，我们将创建一个新的activity，用来托管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。

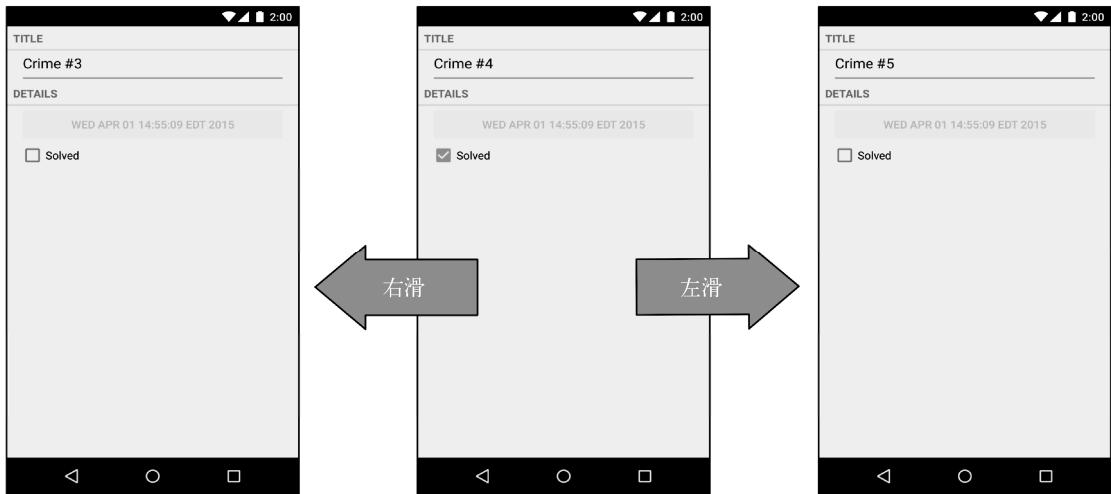


图11-1 划屏切换显示Crime明细内容

图11-2为升级后的CriminalIntent应用对象图。可以看到，名为CrimePagerAdapter的新建activity取代了CrimeActivity。其布局由一个ViewPager组成。

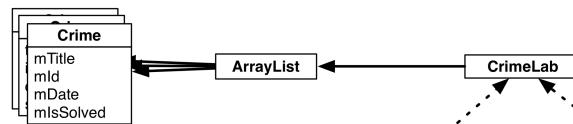
如图11-2所示，无需改变CriminalIntent应用的其他部分，我们只要创建虚线框中的对象就能实现实划屏切换Crime明细页面。特别要说的是，经过上一章的封装，CrimeFragment类既通用又独立，因此这里就不用调整它了。

接下来，我们需要完成以下任务：

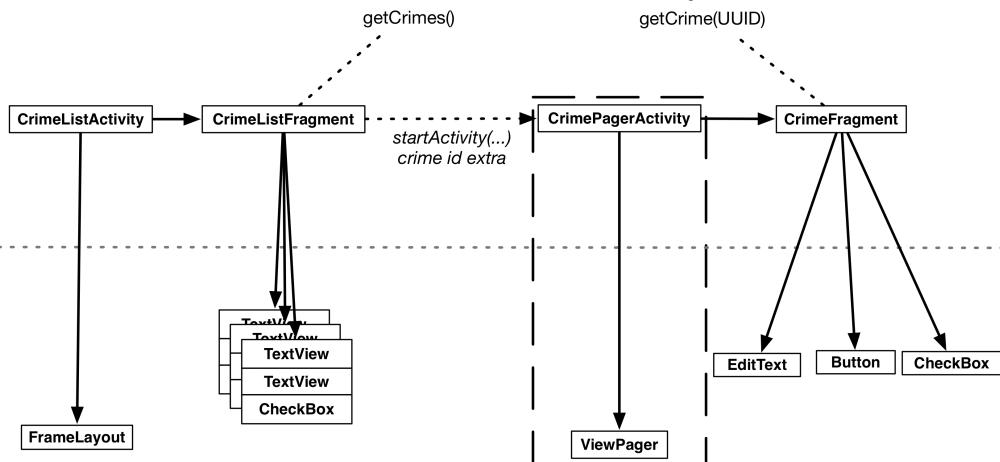
- 创建CrimePagerAdapter类；
- 定义包含ViewPager的视图层级结构；
- 在CrimePagerAdapter类中关联使用ViewPager及其adapter；

□ 修改CrimeHolder.onClick(...)方法，启动CrimePagerActivity。

模型



控制器



视图

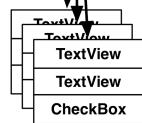


图11-2 CrimePagerActivity的布局示意图

11.1 创建CrimePagerActivity

CrimePagerActivity是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CrimePagerActivity新类并为其配置视图，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 {

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

```

参照图11-3，以ViewPager为根视图，在res/layout/目录下创建名为activity_crime_pager的布局文件。注意，必须使用ViewPager的包名全称（ android.support.v4.view.ViewPager ）。

```

    android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_crime_pager_view_pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

图11-3 CrimePagerAdapter的ViewPager布局 (activity_crime_pager.xml)

因为ViewPager类来自于支持库，所以添加到布局时包名要用全称。与Fragment类不同，ViewPager仅存在于支持库。而且，在SDK的后续版本中，Google也没有在标准库中实现ViewPager类。

11.1.1 ViewPager 与 PagerAdapter

ViewPager在某种程度上类似于RecyclerView。RecyclerView需借助于Adapter提供视图。同样地，ViewPager也需要PagerAdapter的支持。

不过，相较于RecyclerView与Adapter间的协同工作，ViewPager与PagerAdapter间的配合要复杂得多。好在Google提供了PagerAdapter的子类FragmentStatePagerAdapter，它能协助处理许多细节问题。

FragmentStatePagerAdapter化繁为简，提供了两个有用的方法：getCount()和getItem(int)。调用getItem(int)方法，获取并显示crime数组中指定位置的Crime时，它会返回配置过的CrimeFragment来完成显示任务。

在CrimePagerAdapter中，设置ViewPager的pager adapter，并实现它的getCount()和getItem(int)方法，如代码清单11-2所示。

代码清单11-2 设置pager adapter (CrimePagerAdapter.java)

```

public class CrimePagerAdapter extends FragmentActivity {

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

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

        mViewPager = (ViewPager) findViewById(R.id.activity_crime_pager_view_pager);

        mCrimes = CrimeLab.get(this).getCrimes();
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {

            @Override
            public Fragment getItem(int position) {
                Crime crime = mCrimes.get(position);
                return CrimeFragment.newInstance(crime.getId());
            }
        });
    }
}

```

```

        }

        @Override
        public int getCount() {
            return mCrimes.size();
        }
    });
}
}

```

下面来逐行解读新增代码。在activity视图中找到ViewPager后，我们从CrimeLab中（crime的List）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实例需要FragmentManager。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。

（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。细节请参见11.3节。）

pager adapter的两个方法简单直接。getCount()方法返回数组列表中包含的列表项数目。getItem(int)方法有着神奇的魔法。它首先获取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。

11.1.2 整合并配置使用CrimePagerAdapter

现在，弃用CrimeActivity，我们来配置使用CrimePagerAdapter。

首先新增newIntent方法和crime ID的extra常量，如代码清单11-3所示。

代码清单11-3 创建newIntent方法（CrimePagerAdapter.java）

```

public class CrimePagerAdapter extends FragmentActivity {
    private static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimePagerAdapter.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }

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

```

```

        UUID crimeId = (UUID) getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);
        ...
    }
}

```

然后需要修改CrimeListFragment，使得用户单击某个列表项时，CrimeListFragment启动的是CrimePagerActivity实例。

回到 CrimeListFragment.java 文件，修改 CrimeHolder.onClick(...) 方法来启动 CrimePagerActivity，如代码清单11-4所示。

代码清单11-4 配置启动CrimePagerActivity (CrimeListFragment.java)

```

private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {

    ...
    @Override
    public void onClick(View v) {
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}

```

最后，要让操作系统启动CrimePagerActivity，还要在manifest配置文件中添加它，如代码清单11-5所示。打开AndroidManifest.xml，添加CrimePagerActivity声明，同时删除不再使用的CrimeActivity声明；也可以直接重命名CrimeActivity为CrimePagerActivity。

代码清单11-5 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:name=".CrimePagerActivity"
            android:label="@string/app_name" >
        </activity>
        ...
    </application>
</manifest>

```

最后，为保持项目代码的整洁，从项目工具窗口中删除CrimeActivity.java文件。

运行CriminalIntent应用。点击Crime #0查看其明细，然后划屏浏览其他crime明细。可以看到，

页面切换流畅顺滑，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，因此响应迅速。如有需要，也可调用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，再点选其他crime列表项。可以看到屏幕上显示的仍是第一个crime列表项的明细，而非当前点选的列表项。

ViewPager默认只显示PagerAdapter中的第一个列表项。要显示选中的列表项，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。

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。如果Crime实例的mId与intent extra的crimeId相匹配，设置显示指定位置的列表项，如代码清单11-6所示。

代码清单11-6 设置初始分页显示项（CrimePagerActivity.java）

```
public class CrimePagerActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
            ...
            for (int i = 0; i < mCrimes.size(); i++) {
                if (mCrimes.get(i).getId().equals(crimeId)) {
                    mViewPager.setCurrentItem(i);
                    break;
                }
            }
        });
    }
}
```

运行CriminalIntent应用。选择任意列表项，其对应的Crime明细应该能够显示了。现在，ViewPager的使用配置已完成，可以投入使用了。

11.2 FragmentStatePagerAdapter 与 FragmentPagerAdapter

FragmentPagerAdapter是另外一种可用的PagerAdapter，其用法与FragmentStatePagerAdapter基本一致。唯一的区别在于：卸载不再需要的fragment时，各自采用的处理方法有所不同。

FragmentStatePagerAdapter会销毁不需要的fragment。事务提交后，activity的FragmentManager中的fragment会被彻底移除。FragmentStatePagerAdapter类名中的“state”表明：在销毁fragment时，可在onSaveInstanceState(Bundle)方法中保存fragment的Bundle信息。用户切换回来时，保存的实例状态可用来恢复生成新的fragment（如图11-4所示）。

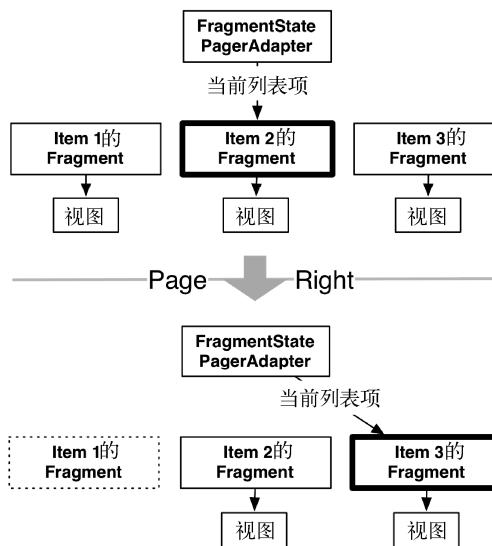


图11-4 FragmentStatePagerAdapter的fragment管理

相比之下，FragmentPagerAdapter有不同的做法。对于不再需要的fragment，FragmentPagerAdapter会选择调用事务的detach(Fragment)方法来处理它，而非remove(Fragment)方法。也就是说，FragmentPagerAdapter只是销毁了fragment的视图，fragment实例还保留在FragmentManager中。因此，FragmentPagerAdapter创建的fragment永远不会被销毁（如图11-5所示）。

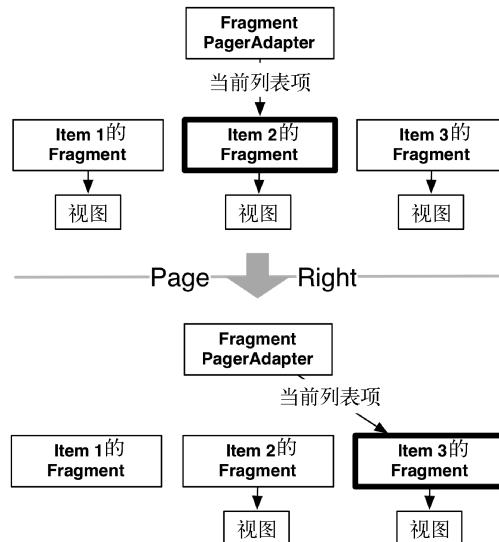


图11-5 FragmentPagerAdapter的fragment管理

选择哪种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、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，以多页面的形式展现。显然，为用户界面添加支持滑动切换的`ViewPager`，能增强应用的触摸体验。此外，将fragment保存在内存中，更易于管理控制层的代码。对于这种类型的用户界面，每个activity通常只有两三个fragment，基本不用担心有内存不足的风险。

11.3 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`ViewPager`的工作原理。

介绍之前，先提个醒：大多情况下，我们无需了解其内部实现细节。

不过，如果要自己实现`PagerAdapter`接口，则需了解`ViewPager-PagerAdapter`和`RecyclerView-Adapter`各自关系的异同。

什么时候需要自己实现`PagerAdapter`接口呢？需要`ViewPager`托管非fragment视图时，就需要实现原生`PagerAdapter`接口。例如，在`ViewPager`中托管图片这样的常见视图对象时。

为什么选择使用`ViewPager`而不是`RecyclerView`呢？

由于无法使用现有的`Fragment`，在`CriminalIntent`应用中使用`RecyclerView`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`。因此，当`RecyclerView`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，因为它要处理更多的视图管理工作。以下为它的基本内部实现。

`PagerAdapter`不使用可返回视图的`onBindViewHolder(...)`方法，而是使用下列方法：

```
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)`方法并不要求立即创建视图。因此，`PagerAdapter`可自行决定何时创建视图。

视图创建完成后，`ViewPager`会在某个时间点注意到它。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)方法就应返回true值，否则返回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。这样的要求显然很宽松，因而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Object (fragment)。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，都要覆盖实现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！

11.4 深入学习：以代码的方式创建布局

本书一直是通过布局文件创建视图的，其实也可以在代码里创建视图。

实际上，可以完全不用创建布局文件，使用以下代码定义ViewPager。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ViewPager viewPager = new ViewPager(this);
    setContentView(viewPager);

    ...
}
```

11

以代码的方式创建视图很简单：调用视图类的构造方法，并传入Context参数。不创建任何布局文件，用代码就能创建完整的视图层级机构。

但最好不要这样做，下面我们就来谈谈使用布局文件的好处。

布局文件能很好地分离控制层和视图层对象：视图定义在XML布局文件中，控制层对象定义在Java代码中。这样，假设控制层有代码修改的话，代码变更管理相对容易很多；反之亦然。

另外，使用布局文件，我们还能使用Android的资源修饰系统，实现按设备属性自动调用合适的布局文件（如第3章中横屏模式的布局文件）。

当然，布局也不是毫无缺点。如果应用只需一个视图，估计没人愿意麻烦地创建并实例化布局XML文件。

除此之外，创建布局文件的缺点都不值得一提。要知道，Android开发团队从没提倡过以代码的方式创建视图，即便是在因设备配置低、开发人员绞尽脑汁压榨机器来提高应用性能的年代。因此，使用布局文件吧，哪怕只是添加个小小的视图ID也会更加简单！

第 12 章

对话框

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，我们为应用添加对话框，以便用户修改crime记录日期。用户点击CrimeFragment中的日期按钮时，会弹出对话框，如图12-1所示。

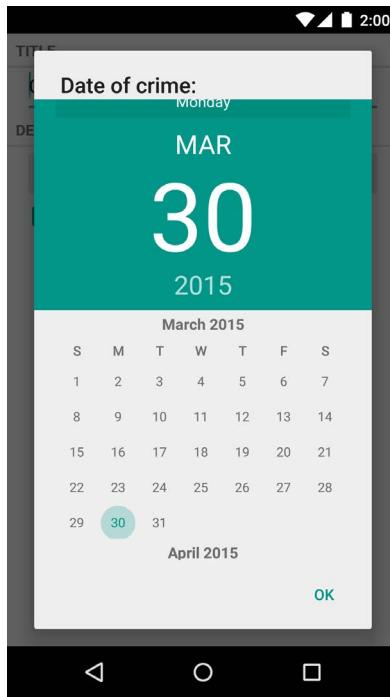


图12-1 可选择crime日期的对话框

对话框是AlertDialog类的一个实例。实际开发中，AlertDialog类是个常用的多用途Dialog子类。

Google重新设计了Lollipop系统的对话框。相比旧系统版本的对话框（参见图12-2左侧），新版对话框界面看起来漂亮多了。

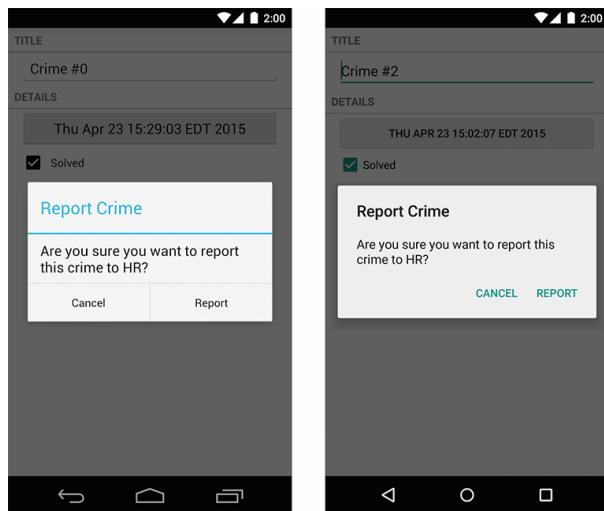


图12-2 新老系统对话框界面对比

如果不限定系统版本，所有Android用户都能体验到新特性该有多好啊！Google也是这么想的，因而推出了AppCompat库。

AppCompat兼容库能将部分最新系统的特色功能移植到Android旧版本系统中。本章，我们使用AppCompat让新旧系统展示风格一致的对话框。第13章和第20章还会使用AppCompat库提供一些其他的特色功能。

12.1 使用 AppCompat 兼容库

要使用AppCompat库，首先要添加它为依赖库。取决于项目创建的方式，很可能你的项目中已经引入了AppCompat依赖库。

打开项目结构窗口（File→Project Structure...），选择app模块并点击Dependencies选项页。如果看不到AppCompat，请点击+按钮，然后从依赖项列表中选择appcompat-v7完成添加（如图12-3所示）。

12

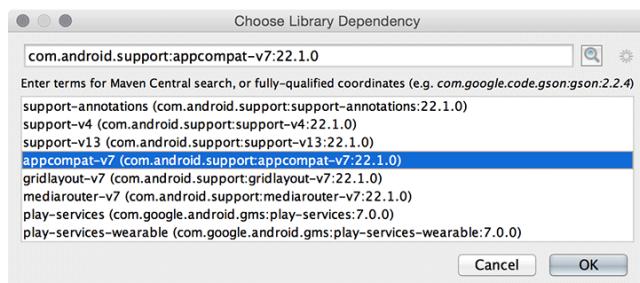


图12-3 添加AppCompat依赖项

AppCompat有自己的AlertDialog实现类。它和Android操作系统内置版AlertDialog类似。导入AlertDialog类时，请注意区分清楚。我们要使用的是`android.support.v7.app.AlertDialog`。

12.2 创建 DialogFragment

建议将AlertDialog封装在DialogFragment（Fragment的子类）实例中使用。当然，不使用DialogFragment也可显示AlertDialog视图，但不推荐这样做。使用FragmentManager管理对话框，可以更灵活地显示对话框。

另外，如果旋转设备，单独使用的AlertDialog会消失，而封装在fragment中的AlertDialog则不会有此问题（旋转后，对话框会被重建恢复）。

就CriminalIntent应用来说，我们首先会创建名为DatePickerFragment的DialogFragment子类。然后，在DatePickerFragment中，创建并配置显示DatePicker组件的AlertDialog实例。DatePickerFragment同样由CrimePagerActivity托管。

图12-4展示了以上各对象间的关系。

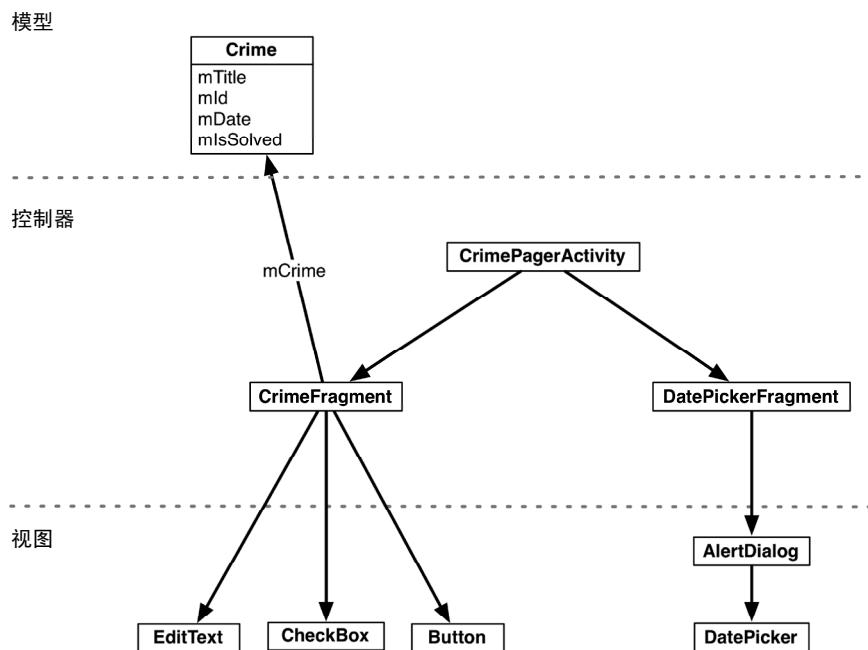


图12-4 由CrimePagerActivity托管的两个fragment对象

要显示对话框，首先应完成以下任务：

- 创建DatePickerFragment类；
- 创建AlertDialog；

□ 借助FragmentManager在屏幕上显示对话框。

稍后，我们还将配置使用DatePicker，并实现在CrimeFragment和DatePickerFragment之间传递数据。

继续学习之前，请参照代码清单12-1添加字符串资源备用。

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="date_picker_title">Date of crime:</string>
</resources>
```

创建DatePickerFragment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)
```

为在屏幕上显示DialogFragment，托管activity的FragmentManager会调用它。

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组件稍后添加。）

导入AlertDialog时，请确认选择的是AppCompat库中的版本：android.support.v7.app.AlertDialog。

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();
    }
}
```

以上代码中，使用AlertDialog.Builder类，我们以流接口（fluent interface）的方式创建了AlertDialog实例。下面来详细解读这段代码。

首先，将Context参数传入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：字符串资源和实现`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；至于监听器参数，暂时传入`null`值。监听器接口稍后实现。

(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。如果同一对话框上放置有多个按钮，按钮的类型与命名决定着它们在对话框上显示的位置。)

最后，调用`AlertDialog.Builder.create()`方法，返回配置完成的`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，请查阅开发文档详细了解。接下来，我们学习如何在屏幕上显示对话框。

12.2.1 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`FragmentManager`队列中的`DialogFragment`。两个方法任你选：如传入`FragmentTransaction`参数，你自己负责创建并提交事务；如传入`FragmentManager`参数，系统会自动创建并提交事务。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。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 {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...

        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
    }
}
```

```

mDateButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        FragmentManager manager = getFragmentManager();
        DatePickerFragment dialog = new DatePickerFragment();
        dialog.show(manager, DIALOG_DATE);
    }
});

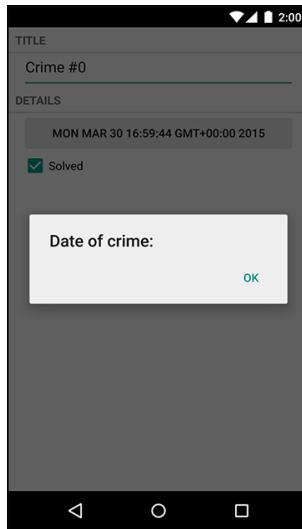
mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
...

return v;
}

...
}

```

运行CriminalIntent应用。点击日期按钮弹出对话框，如图12-5所示。



12

图12-5 带标题和OK按钮的AlertDialog

12.2.2 设置对话框的显示内容

接下来，使用`AlertDialog.Builder`的`setView(...)`方法，添加`DatePicker`组件给`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。

在项目工具窗口中，以`DatePicker`为根元素，创建名为`dialog_date.xml`的布局文件。新布局

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12-6，配置DatePicker布局。

```

DatePicker
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"

```

图12-6 DatePicker布局 (layout/dialog_date.xml)

在DatePickerFragment.onCreateDialog(...)方法中，实例化DatePicker视图并添加给对话框，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 = LayoutInflater.from(getActivity())
        .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，确认能在对话框上显示DatePicker视图。如果是Lollipop系统，还能看到日历选择界面，如图12-7所示。



图12-7 Lollipop系统中的DatePicker

图12-7所示的日历选择界面是随Material design引入的。这个版本的DatePicker组件会忽略布局中指定的`calendarViewShown`属性。如果使用旧版本系统，DatePicker组件会使用`calendarViewShown`属性，因而我们会看到图12-8所示的用户界面。

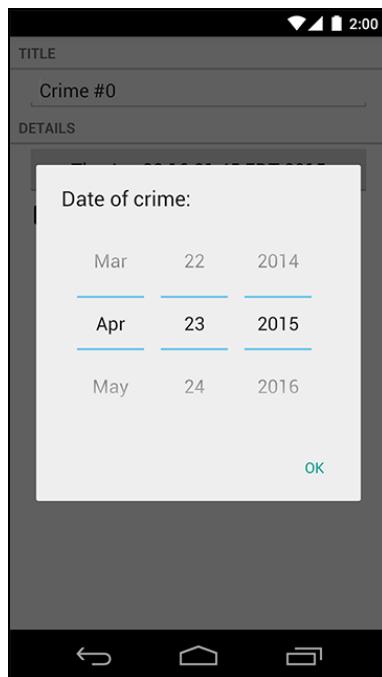


图12-8 显示DatePicker的AlertDialog

对话框完全兼容新旧系统，不过新系统版本的界面显然更美观。

采用下列代码也能创建DatePicker对象，为何还要费事地定义XML布局文件，再去实例化视图对象呢？

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker datePicker = new DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(datePicker)
        ...
        .create();
}
```

这是因为，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，如果想在对话框的DatePicker旁再添加一个TimePicker，只需更新布局文件就能显示新视图。

你可能已经注意到，即使设备旋转，用户所选日期也都会得到保留。（弹出对话框，选择任意日期，按Fn+Control+F12/Ctrl+F1组合键确认。）这是如何做到的呢？前面说过，设备配置改变

时，具有ID属性的视图可以保存运行状态；而我们以dialog_date.xml布局创建DatePicker时，编译工具已为DatePicker生成了唯一的ID。如果以代码的方式创建DatePicker，想看到同样的效果，记得为其设置ID属性。

至此，显示对话框的工作就完成了。下一节，我们将关联显示Crime日期，并支持用户对其进行修改。

12.3 fragment 间的数据传递

前面，我们实现了activity之间以及基于fragment的activity之间的数据传递。现在需实现同一activity托管的两个fragment（CrimeFragment和DatePickerFragment）之间的数据传递，如图12-9所示。

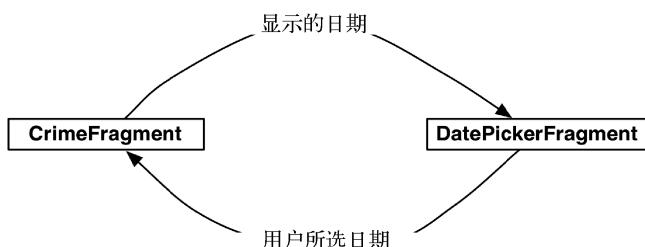


图12-9 CrimeFragment与DatePickerFragment间的对话

要传递crime的记录日期给DatePickerFragment，需新建newInstance(Date)方法，然后将Date作为argument附加给fragment。

为返回新日期给CrimeFragment，并更新模型层以及对应视图，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12-10所示。

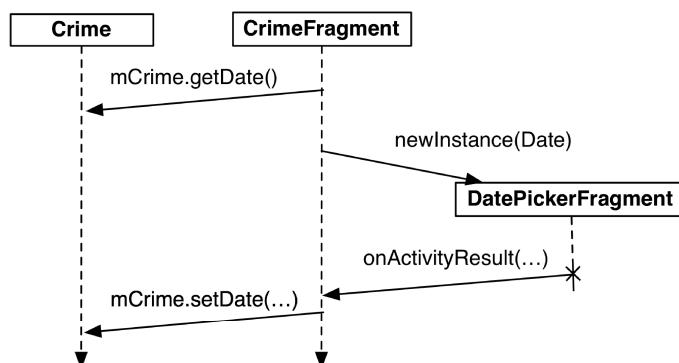


图12-10 CrimeFragment和DatePickerFragment间的事件流

在稍后的实施代码中可以看到，我们没有调用托管activity的Activity.onActivityResult(...)方法，而是调用了Fragment.onActivityResult(...)方法，这似乎令人费解。实际上，调用onActivityResult(...)方法实现fragment间的数据传递不仅行得通，而且可以更灵活地展现对话框fragment。

12.3.1 传递数据给 DatePickerFragment

要传递crime记录日期给DatePickerFragment，需将它保存在DatePickerFragment的argument bundle中。这样，DatePickerFragment便可直接获取到它。

创建和设置fragment argument通常是在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 {

    private static final String ARG_DATE = "date";

    private DatePicker mDatePicker;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_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() {
        @Override
        public void onClick(View v) {
            FragmentManager manager = getFragmentManager()
                DatePickerFragment dialog = new DatePickerFragment();
                DatePickerFragment dialog = DatePickerFragment
```

```

        .newInstance(mCrime.getDate());
        dialog.show(manager, DIALOG_DATE);
    }
});

return v;
}

```

DatePickerFragment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是个时间戳，无法直接提供整数形式的月、日、年。

要达到目的，必须首先创建一个Calendar对象，然后用Date对象配置它，再从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) {
    Date date = (Date) getArguments().getSerializable(ARG_DATE);

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = LayoutInflater.from(getActivity())
        .inflate(R.layout.dialog_date, null);

    mDatePicker = (DatePicker) v.findViewById(R.id.dialog_date_picker);
    mDatePicker.init(year, month, day);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}

```

现在，成功完成了CrimeFragment向DatePickerFragment传递日期。运行CriminalIntent应用，查看最终效果。

12.3.2 返回数据给 CrimeFragment

要让CrimeFragment接收DatePickerFragment返回的日期数据，首先需要清楚它们之间的关系。

实现activity的数据回传，我们调用startActivityForResult(...)方法，ActivityManager负责跟踪管理父activity与子activity间的关系。回传数据后，子activity被销毁，但ActivityManager知道接收数据的是哪个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。即使是在CrimeFragment和DatePickerFragment被销毁和重建后，操作系统也会重新关联它们。调用以下Fragment方法可建立这种关联：

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

该方法有两个参数：目标fragment以及类似于传入startActivityForResult(...)方法的请求代码。需要时，目标fragment使用请求代码确认是哪个fragment在回传数据。

目标fragment和请求代码由FragmentManager负责跟踪管理，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 {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    private static final int REQUEST_DATE = 0;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
                           Bundle savedInstanceState) {
        ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                    .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(manager, DIALOG_DATE);
            }
        });
        return v;
    }

    ...
}
```

2. 传递数据给目标fragment

建立CrimeFragment与DatePickerFragment间的联系后，需将数据回传给CrimeFragment。回传日期将作为extra附加给Intent。

要使用什么方法发送intent信息给目标fragment? 令人难以置信的是, 我们会让DatePickerFragment类调用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, ActivityManager会自动调用Activity.onActivityResult(...)方法。父activity接收到Activity.onActivityResult(...)方法调用后, 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, 就能实现数据的回传。该方法恰好有我们需要的如下信息。

- 请求代码: 与传入setTargetFragment(...)方法相匹配, 告诉目标fragment返回结果来自哪里。
- 结果代码: 决定下一步该采取什么行动。
- Intent: 包含extra数据。

在DatePickerFragment类中, 新建sendResult(...)私有方法, 创建intent并将日期数据作为extra附加到intent上。最后调用CrimeFragment.onActivityResult(...)方法, 如代码清单12-9所示。

代码清单12-9 回调目标fragment (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {

    public static final String EXTRA_DATE =
        "com.bignerdranch.android.criminalintent.date";

    private static final String ARG_DATE = "date";

    ...

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...
    }

    private void sendResult(int resultCode, Date date) {
        if (getTargetFragment() == null) {
            return;
        }

        Intent intent = new Intent();
        intent.putExtra(EXTRA_DATE, date);

        getTargetFragment()
            .onActivityResult(getTargetRequestCode(), resultCode, intent);
    }
}
```

现在来使用sendResult(...)私有方法。用户点按对话框中的positive按钮时, 需要从

DatePicker中获取日期并回传给CrimeFragment。在onCreateDialog(...)方法中，替换掉setPositiveButton(...)的null参数，实现DialogInterface.OnClickListener监听器接口。在监听器接口的onClick(...)方法中，获取日期并调用sendResult(...)方法，如代码清单12-10所示。

代码清单12-10 一切是否都OK? (DatePickerFragment.java)

```
@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() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    int year = mDatePicker.getYear();
                    int month = mDatePicker.getMonth();
                    int day = mDatePicker.getDayOfMonth();
                    Date date = new GregorianCalendar(year, month, day).getTime();
                    sendResult(Activity.RESULT_OK, date);
                }
            })
        .create();
}
```

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12-11所示。

代码清单12-11 响应DatePicker对话框 (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
    ...
}

@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(...)`和`onActivityResult(...)`方法中，设置按钮显示文字的代码完全一样。为避免代码冗余，可以将其封装到`updateDate()`公共方法中，然后分别调用。

除手工封装公共代码的方式外，还可以使用Android Studio的内置工具。加亮选取设置 `mDateButton` 显示文字的代码，右键单击并选择 Refactor → Extract → Method... 菜单项，弹出图 12-11 所示的界面。

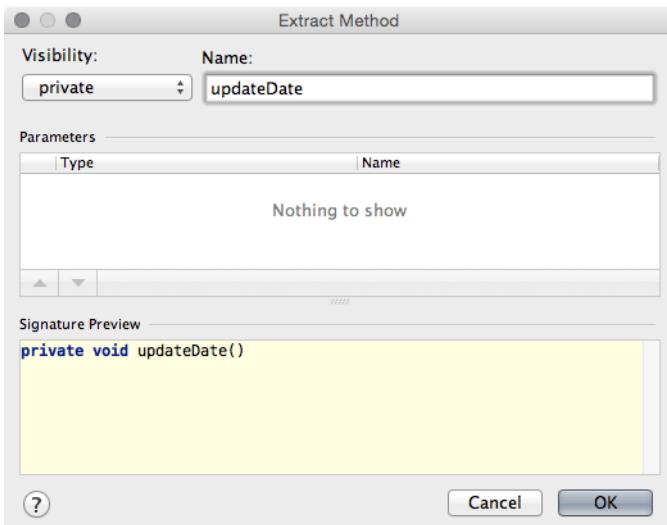


图12-11 使用Android Studio抽取方法

设置方法私有并命名为updateDate。点击OK按钮，Android Studio会提示还有其他地方使用了这段代码。点击Yes允许它自动处理。然后确认updateDate方法封装完成并实现了在相应地方的调用，如代码清单12-12所示。

代码清单12-12 使用updateDate()公共方法 (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {  
    ...  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        View v = inflater.inflate(R.layout.fragment_crime, container, false);  
        return v;  
    }  
}
```

```
...
    mDateButton = (Button) v.findViewById(R.id.crime_date);
    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);
        updateDate();
    }
}

private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}
}
```

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

编写需要用户大量输入以及要求更多空间显示输入的应用，并且要让应用同时支持手机和平板设备时，使用onActivityResult(...)方法返回数据给目标fragment是比较方便的。

手机屏幕空间有限，因此通常需要使用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12-12所示。

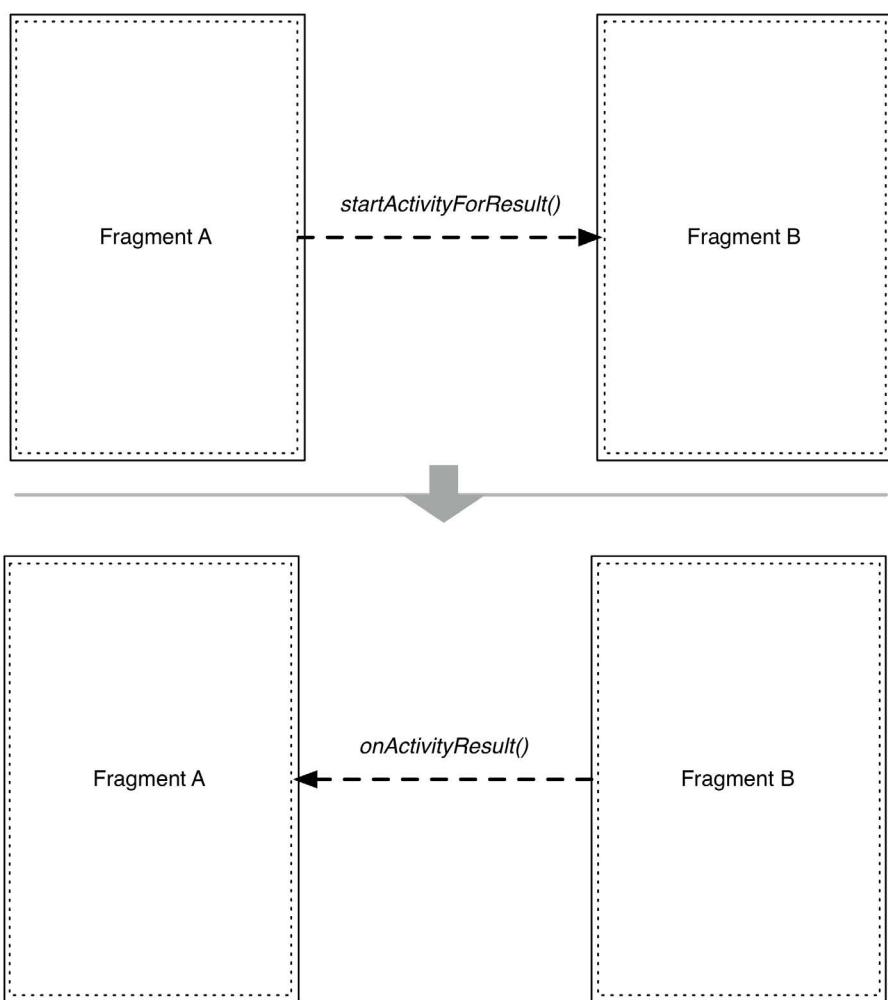


图12-12 手机设备上activity间的数据传递

平板设备的屏幕空间比较大，适合以弹出对话框的方式显示信息和接收用户输入。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12-13所示。

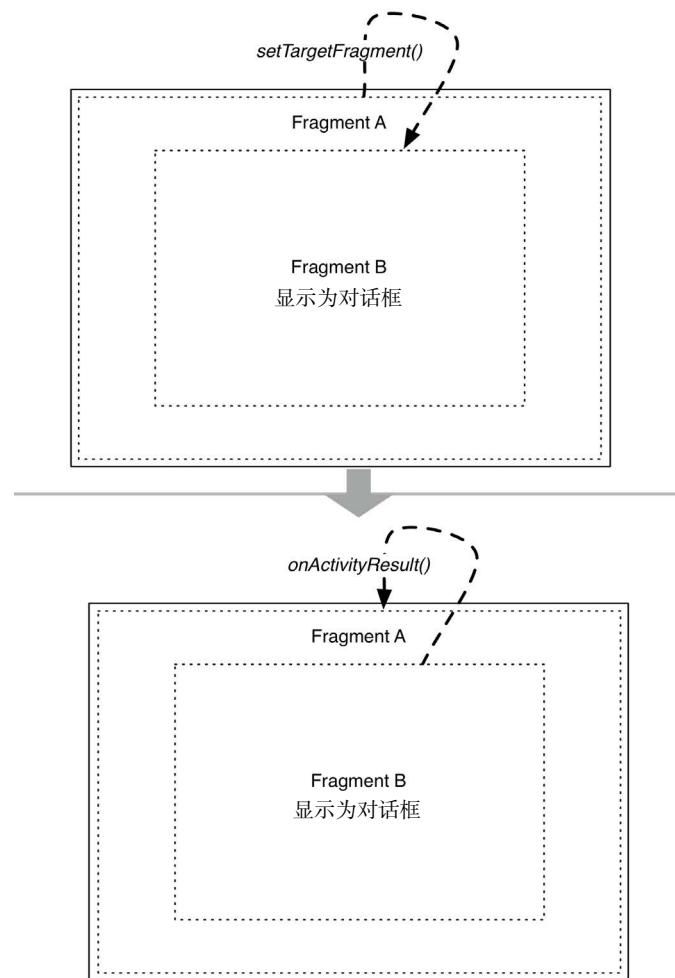


图12-13 平板设备上fragment间的数据传递

12

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。

12.4 挑战练习：更多对话框

首先看个简单的练习。写一个名为`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。在`CrimeFragment`用户界面上再添加一个按钮，以显示`TimePickerFragment`视图界面。

12.5 挑战练习：按设备类型展现 DialogFragment

再来看个有些难度的练习：优化DatePickerFragment的呈现方式。

完成这个挑战，初步分析需三大步骤。第一步，替换掉onCreateDialog方法，改用onCreateView方法来创建DatePickerFragment的视图。以这种方式创建DialogFragment的话，对话框界面上看不到title区域，同样没有放置按钮的空间。这需要我们自行在dialog_date.xml布局中创建OK按钮。

有了DatePickerFragment视图，接下来就能以对话框或以在activity中内嵌的方式展现。第二步，我们创建SingleFragmentActivity子类。它的任务就是托管DatePickerFragment。

选择这种方式展现DatePickerFragment，就要使用startActivityForResult机制回传日期给CrimeFragment。在DatePickerFragment中，如果目标fragment不存在，就调用托管activity的setResult(int, intent)方法回传日期给CrimeFragment。

最后，修改CriminalIntent应用：如果是手机设备，就以全屏activity的方式展现DatePickerFragment；如果是平板设备，就以对话框的方式展现DatePickerFragment。想知道如何按设备屏幕大小优化应用，请提前学习第17章的相关内容。

优秀的Android应用都注重设计工具栏。工具栏可安置菜单选项、提供应用导航，还能帮助统一设计风格、塑造品牌形象。

本章，我们为CriminalIntent应用创建工具栏菜单，提供新增crime记录的菜单项，并实现向上按钮的导航功能，如图13-1所示。

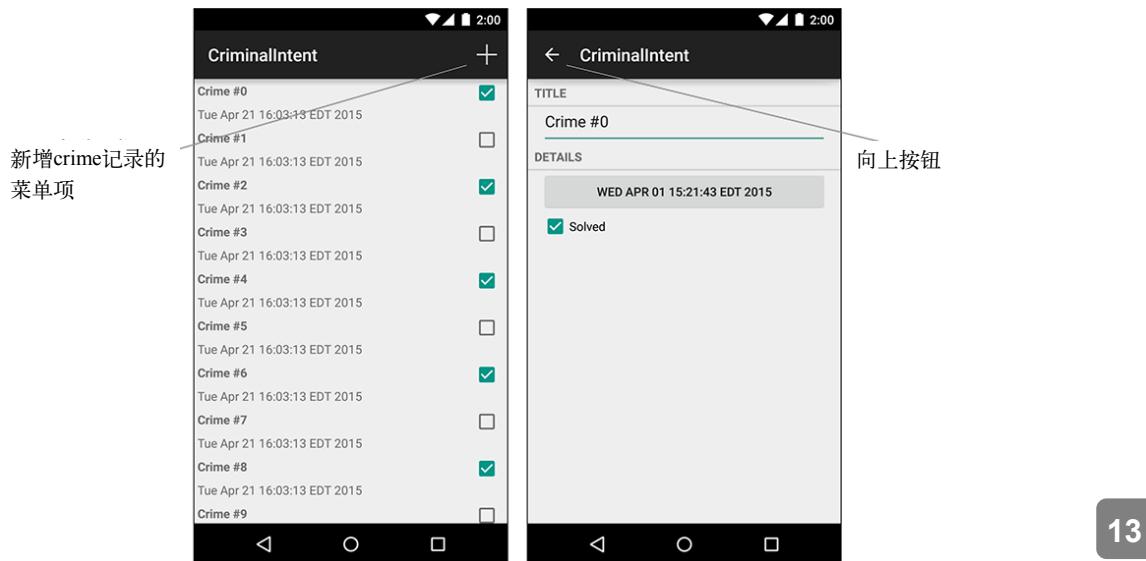


图13-1 CriminalIntent应用的工具栏

13.1 AppCompat

Android 5.0 (Lollipop) 引入了工具栏这个新增组件。Lollipop之前，应用中用于导航或提供菜单操作的是操作栏。

工具栏和操作栏有些类似，但它基于操作栏进化而来。因而，工具栏的界面更美观，使用更灵活。

CriminalIntent应用最低只支持API 16级，原生工作栏无法支持更老版本的系统。不过，Google已将它移植到了AppCompat库。这样一来，老版本系统（API 7级、Android 2.1以上）都能使用Lollipop上的工作栏了。

使用 AppCompat 库

第12章已经添加过AppCompat依赖项。本章我们要完全整合AppCompat库，所以还需几个步骤的操作。取决于项目的创建方式，有些步骤你可能已经做过。

完全整合AppCompat库，还需要：

- 添加AppCompat依赖项；
- 使用一种AppCompat主题；
- 确保所有的activity都是AppCompatActivity子类。

1. 更新主题

添加了依赖库，接下来至少要使用一种AppCompat主题。AppCompat库自带以下三种主题。

- Theme.AppCompat：黑色主题
- Theme.AppCompat.Light：浅色主题
- Theme.AppCompat.Light.DarkActionBar：带黑色工具栏的浅色主题

应用级别的主题设置在AndroidManifest.xml文件中。主题也可按activity配置。打开AndroidManifest.xml文件，查看application标签的android:theme属性，应该可以看到如代码清单13-1所示的主题配置。

代码清单13-1 各种manifest配置项（AndroidManifest.xml）

```
...
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
...

```

AppTheme定义在res/values/styles.xml文件中。取决于创建项目的方式，你的项目很可能有定义在多个styles.xml文件中的多个AppTheme版本。这些资源修饰文件用来适配不同版本的Android设备。不过，AppCompat库可以统一各系统版本的主题风格，从而省去了适配的麻烦。

如果有多个版本的styles.xml文件，现在就删掉多余的，如图13-2所示。保留唯一的res/values/styles.xml就可以了。

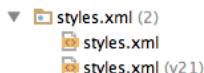


图13-2 多余的styles.xml文件

清理完多余的主题样式文件，打开res/values/styles.xml文件，参照代码清单13-2设置应用的主题。

代码清单13-2 使用AppCompat主题 (res/values/styles.xml)

```
<resources>  
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
    </style>  
</resources>
```

在第20章，我们还会进一步学习与样式和主题相关的知识。

2. 使用AppCompatActivity

完全使用AppCompat库的最后一步是让activity类继承AppCompatActivity类。当前，为使用支持库中的fragment实现，CriminalIntent应用中的所有activity都是FragmentActivity子类。

Google早已做好了基础工作。AppCompatActivity恰恰就是FragmentActivity的子类。事情因此简单多了，我们只要进行简单的代码修改就行了。

修改SingleFragmentActivity和CrimePagerActivity，让它们直接继承AppCompatActivity类，如代码清单13-3和代码清单13-4所示。（注意，CrimeListActivity是SingleFragmentActivity的子类，请不要修改。）

代码清单13-3 转换为AppCompatActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {  
    public abstract class SingleFragmentActivity extends AppCompatActivity {  
        ...  
    }
```

代码清单13-4 转换为AppCompatActivity (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity AppCompatActivity {  
    ...  
}
```

运行CriminalIntent应用。一切正常的话，应该可以看到如图13-3所示的画面。

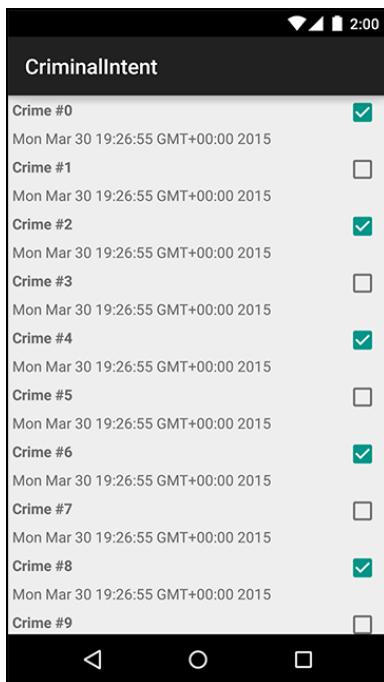


图13-3 新工具栏

现在，CriminalIntent应用用上了AppCompat工具栏，可以着手添加工具栏菜单了。

13.2 工具栏菜单

工具栏菜单由操作项（又称为菜单项）组成，它占据着工具栏的右上方区域。操作项的操作应用于当前屏幕，甚至整个应用。现在，我们来添加允许用户新增crime记录的菜单项。

菜单及菜单项需用到一些字符串资源。参照代码清单13-5，将它们添加到strings.xml文件中。尽管有些字符串资源现在还用不到，但方便起见，一并完成添加。

代码清单13-5 添加字符串资源（res/values/strings.xml）

```
<resources>
    ...
    <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_format">%1$s crimes</string>
</resources>
```

13.2.1 在 XML 文件中定义菜单

菜单是一种类似于布局的资源。创建菜单定义文件并放置在res/menu目录下，Android会自动生成相应的资源ID。随后，在代码中实例化菜单时，就可以直接使用。

在项目工具窗口中，右键单击res目录，选择New→Android resource file菜单项。在弹出的窗口界面，选择Menu资源类型，并命名资源文件为fragment_crime_list，点击OK按钮确认，如图13-4所示。Android Studio随后会创建res/menu/fragment_crime_list.xml文件。

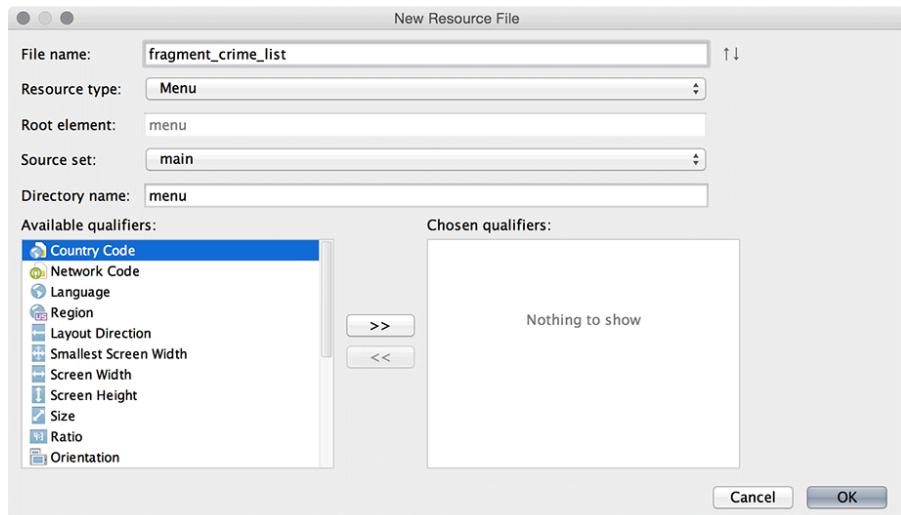


图13-4 创建菜单文件

打开新建的fragment_crime_list.xml文件。参照代码清单13-6，添加新的item元素。

代码清单13-6 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"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>
</menu>
```

showAsAction属性用于指定菜单选项是显示在工具栏上，还是隐藏于溢出菜单（overflow menu）。该属性当前设置为ifRoom和withText的组合值。因此，只要空间足够，菜单项图标及其文字描述都会显示在工具栏上。如空间仅够显示菜单项图标，文字描述就不会显示。如空间大小不够显示任何项，菜单项就会隐藏到溢出菜单中。

可通过工具栏最右端带有三个点的图标来访问溢出菜单，如图13-5所示。

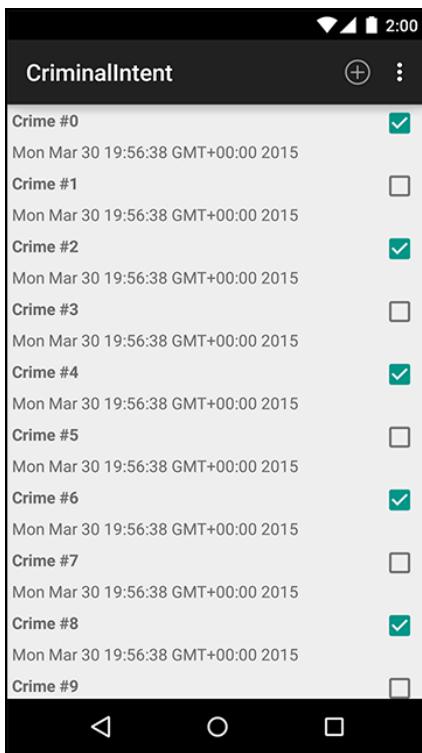


图13-5 工具栏中的溢出菜单

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`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，`never`属性值是个不错的选择。总之，为避免混乱的用户界面，工具栏上只应放置常用菜单项。

1. 应用命名空间

注意，不同于常见的`android`命名空间声明，`fragment_crime_list.xml`文件使用`xmlns`标签定义了全新的`app`命名空间。指定`showAsAction`属性时，就用了这个新定义的命名空间。

基于历史代码的原因，AppCompat库需要使用`app`命名空间。操作栏API随Android 3.0引入。为支持各种旧系统版本设备，早期创建的AppCompat库捆绑了兼容版操作栏。在运行Android 2.3或更早版本系统的设备上，菜单及其相应的XML文件是确实存在的，但是`android:showAsAction`属性是随着操作栏的发布才添加的。

AppCompat库不希望使用原生`showAsAction`属性，因此，它提供了定制版`showAsAction`属性（`app:showAsAction`）。

2. 使用Android Asset Studio

应用使用的图标有两种：系统图标和项目资源图标。系统图标（system icon）是Android操作系统内置的图标。`android:icon`属性值`@android:drawable/ic_menu_add`就引用了系统图标。

在应用原型设计阶段，使用系统图标不会有什么问题；而在应用发布时，无论用户运行什么设备，最好能统一应用的界面风格。要知道，不同设备或操作系统版本间，系统图标的显示风格往往具有很大差异。有些设备的系统图标甚至与应用的整体风格完全不搭。

一种解决方案是创建定制图标。这需要针对不同屏幕显示密度或各种可能的设备配置，准备不同版本的图标。访问<http://developer.android.com/design/style/iconography.html>，查看Android的图标设计指南，可了解更多相关信息。

另一种解决方案是找到适合应用的系统图标，将它们直接复制到项目的drawable资源目录中。

可在Android SDK的安装目录下找到系统图标。例如，在Mac电脑上，路径通常为/Users/user/Library/Android/sdk；在Windows电脑上，默认的路径是\Users\user\sdk。此外，还可以打开项目结构窗口，选择SDK Location来确认SDK的具体存放路径。

打开SDK目录，可找到包括ic_menu_add在内的Android系统资源。资源的具体目录是/platforms/android-21/data/res，路径中的数字21代表Android的API级别。

下面介绍第三个，也是最容易的解决方案：使用Android Studio内置的Android Asset Studio工具。可以用它为工具栏定制图片。

在项目工具窗口中，右键单击drawable目录，选择New→Image Asset菜单项，弹出如图13-6所示的Asset Studio窗口。

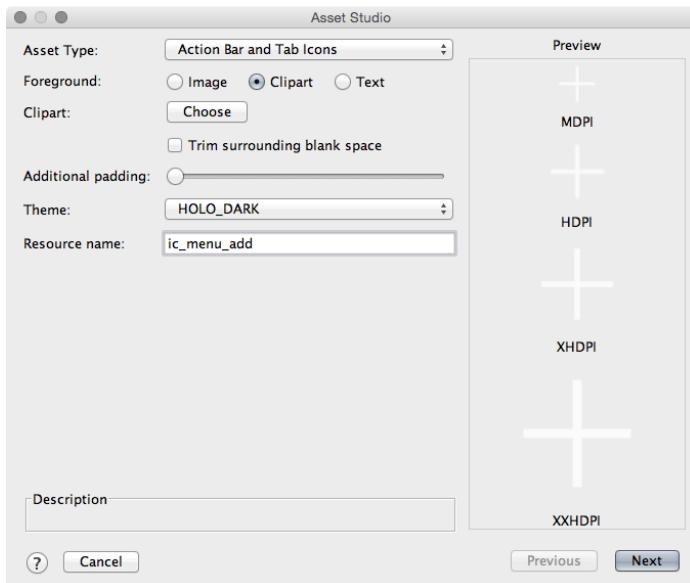


图13-6 Asset Studio

这里，我们可以生成各类图标。在Asset Type中选择ActionBar and Tab Icons，确定Foreground选项为Clipart，然后单击Choose按钮挑选图标。

在可选图标窗口，选择看上去像+号的图片，如图13-7所示。

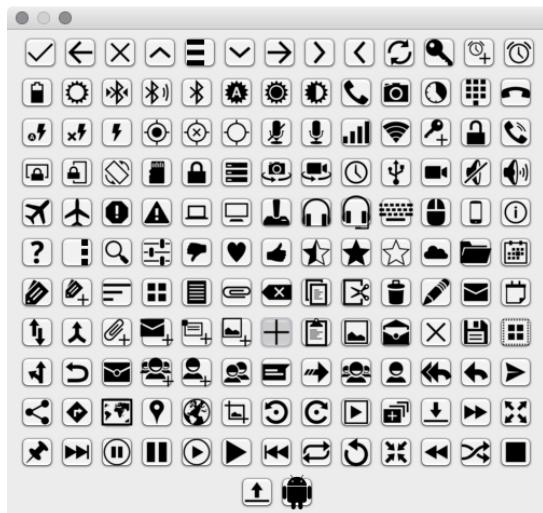


图13-7 可选图标

最后，命名资源为ic_menu_add，单击Next按钮进入图13-8所示的画面。

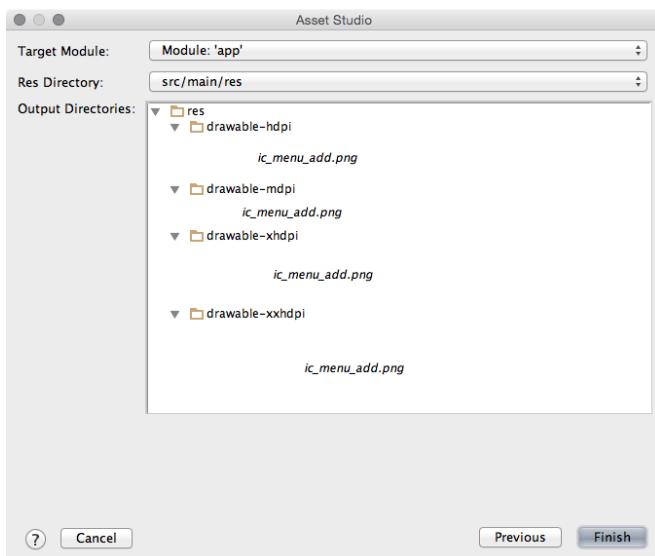


图13-8 Asset Studio生成的文件

Asset Studio会询问添加图片到哪个模块和目录。这里使用默认设置就可以了。Asset Studio还提供了待添加的mdpi、hdpi、xhdpi和xxhdpi类型图标的预览画面。好了，点击Finish按钮完成。真是太方便了！

最后，要在项目中使用这些图标，别忘了修改布局文件中的`android:icon`属性，如代码清

单13-7所示。

代码清单13-7 引用资源 (menu/fragment_crime_list.xml)

```
<item
    android:id="@+id/menu_item_new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/new_crime"
    app:showAsAction="ifRoom|withText"/>
```

13.2.2 创建菜单

在代码中，Activity类提供了管理菜单的回调函数。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有一套自己的选项菜单回调函数。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fragment_crime_list.xml中定义的菜单，如代码清单13-8所示。

代码清单13-8 实例化选项菜单 (CrimeListFragment.java)

```
@Override
public void onResume() {
    super.onResume();
    updateUI();
}

@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，将布局文件中定义的菜单项目填充到Menu实例中。

注意，我们也调用了超类的onCreateOptionsMenu(...)方法。当然，也可以不调用它，但作为一项开发规范，有理由推荐这么做。调用该超类方法，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，onCreateOptionsMenu(...)超类方法什么也没做，这里的调用仅仅是遵循约定而已。

Fragment.onCreateOptionsMenu(Menu, MenuInflater)方法是由FragmentManager负责调用的。因此，当activity接收到操作系统的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，让FragmentManager知道CrimeListFragment需接收选项菜单方法回调，如代码清单13-9所示。

代码清单13-9 调用setHasOptionsMenu方法 (CrimeListFragment.java)

```
...
private RecyclerView mCrimeRecyclerView;
private CrimeAdapter mAdapter;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
...

```

运行CriminalIntent应用，查看新创建的选项菜单，如图13-9所示。



图13-9 显示在工具栏上的菜单项图标

菜单项标题怎么没有显示？大多数手机设备在竖直模式下屏幕空间有限。因此，应用的操作栏上只够显示菜单项图标。长按工具栏上的菜单项图标，可弹出其标题，如图13-10所示。



图13-10 长按工具栏上的图标，显示菜单项标题

水平模式下，工具栏上会有足够的空间同时显示菜单项图标和标题，如图13-11所示。

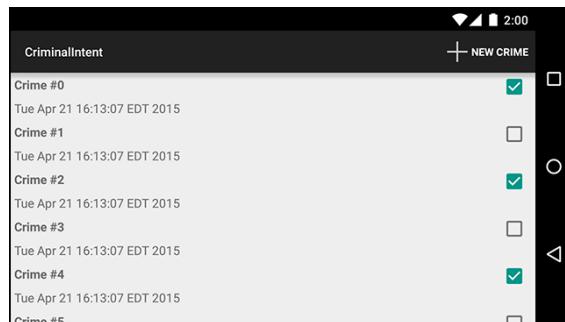


图13-11 同时显示的菜单项图标和标题

13

13.2.3 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列表。在CrimeLab.java中，新增addCrime()方法，实现添加Crime到列表，如代码清单13-10所示。

代码清单13-10 添加新的crime (CrimeLab.java)

```
...
```

```
public void addCrime(Crime c) {
```

```

    mCrimes.add(c);
}

public List<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13-11所示。

代码清单13-11 再见，随机crime记录！(CrimeLab.java)

```

private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0);
        mCrimes.add(crime);
    }
}

```

用户点击菜单中的菜单项时，fragment会收到onOptionsItemSelected(MenuItem)方法的回调请求。传入该方法的参数是一个描述用户选择的MenuItem实例。

当前菜单仅有一个菜单项，但菜单通常包含多个菜单项。通过检查菜单项ID，可确定被选中的是哪个菜单项，然后作出相应的响应。这个ID实际就是在菜单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，创建新的Crime实例，并将其添加到CrimeLab中，然后启动CrimePagerActivity实例，让用户可以编辑新创建的Crime记录，如代码清单13-12所示。

代码清单13-12 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 intent = CrimePagerActivity
                .newIntent(getActivity(), crime.getId());
            startActivity(intent);
            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`值以表明全部任务已完成。另外，默认`case`表达式中，如果菜单项ID不存在，超类版本方法会被调用。

运行CriminalIntent应用，尝试使用菜单，添加一些crime记录并进行编辑。（新增记录前，空空如也的列表看上去不够专业，但是不用担心，本章末的挑战练习就是为此而设。）

13.3 实现层级式导航

目前为止，CriminalIntent应用主要靠后退键在应用内导航。后退键导航又称为临时性导航，只能返回到上一次浏览过的用户界面；而层级式导航（hierarchical navigation，有时又称为ancestral navigation）可在应用内逐级向上导航。

有了层级式导航，用户可点击工作栏左边的向上按钮向上导航。在Jelly Bean（API 16级）设备上，可轻松实现层级式导航。但在这之前，开发者只能自己动手处理向上按钮的显示和点击事件。

打开AndroidManifest.xml，参照代码清单13-13添加`parentActivityName`属性，开启CriminalIntent应用的层级式导航。

代码清单13-13 启用向上按钮（AndroidManifest.xml）

```

...
<activity
    android:name=".CrimePagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".CrimeListActivity">
</activity>
...

```

运行应用并创建新的crime记录。在屏幕的上方，可看到如图13-12所示的向上按钮。点击按钮可向上一级导航至CrimeListActivity用户界面。



图13-12 CrimePagerActivity界面上的向上按钮

层级导航的工作原理

CriminalIntent应用中，后退按钮导航和向上按钮导航执行同样的操作。在CrimePagerActivity界面，无论按哪个按钮导航，都是回到CrimeListActivity界面。虽然结果一样，但它们各自的后台实现机制却大不相同。知道这一点很重要，因为取决于具体应用，向上导航很可能让用户

迷失在众多activity中（这里指回退栈内的众多activity）。

用户点击向上按钮自CrimePagerActivity界面向上导航时，如下的intent会被创建：

```
Intent intent = new Intent(this, 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的activity实例。如存在，则弹出栈内所有其他activity，让启动的目标activity出现在栈顶（显示在屏幕上），如图13-13所示。

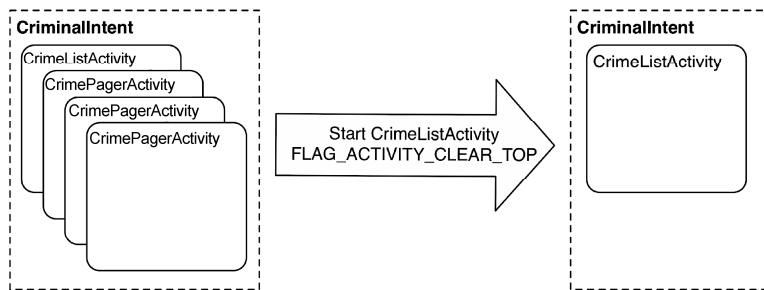


图13-13 工作中的FLAG_ACTIVITY_CLEAR_TOP

13.4 可选菜单项

在本节中，我们将利用前面学过的菜单资源相关知识，添加一个菜单项来实现显示或隐藏CrimeListActivity工具栏的子标题。

打开res/menu/fragment_crime_list.xml文件，参照代码清单13-14，新增一个名为Show Subtitle的菜单项。如显示空间足够，它将显示在工具栏上。

代码清单13-14 添加Show Subtitle菜单项（res/menu/fragment_crime_list.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>

    <item
        android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

子标题需显示crime记录条数，参照代码清单13-15，创建updateSubtitle()新方法实现这

个需求。

代码清单13-15 设置工具栏子标题 (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

getString(int resId, Object...formatArgs)方法接受字符串资源中占位符的替换值, updateSubtitle()用它产生子标题字符串。

接着, 托管CrimeListFragment的activity被强制类型转换为AppCompatActivity。既然CriminalIntent应用使用了AppCompat库, 所有activity就都是AppCompatActivity的子类, 自然也能访问工具栏。出于兼容历史代码的原因, 在AppCompat库中, 工具栏在很多地方仍被称为操作栏。

在onOptionsItemSelected(...)方法中, 调用updateSubtitle()方法响应新增菜单项的单击事件, 如代码清单13-16所示。

代码清单13-16 响应Show Subtitle菜单项单击事件 (CrimeListFragment.java)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
        case R.id.menu_item_show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行CriminalIntent应用, 点击Show Subtitle菜单项, 确认子标题显示出crime记录条数。

13.4.1 切换菜单项标题

工具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。

调用`onOptionsItemSelected(MenuItem)`方法时，传入的参数是用户点击的`MenuItem`。虽然可以在这个方法里更新`Show Subtitle`菜单项的文字，但设备旋转重建工具栏时，子标题的变化会丢失。

比较好的解决方法是在`onCreateOptionsMenu(...)`方法内更新`Show Subtitle`菜单项，并在用户点击子标题菜单项时重建工具栏。对于用户选择菜单项或重建工具栏的场景，都可以使用这段菜单项更新代码。

首先新增跟踪记录子标题状态的成员变量，如代码清单13-17所示。

代码清单13-17 记录子标题状态（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {  
  
    private RecyclerView mCrimeRecyclerView;  
    private CrimeAdapter mAdapter;  
    private boolean mSubtitleVisible;  
  
    ...
```

接着，用户点击`Show Subtitle`菜单项时，在`onCreateOptionsMenu(...)`方法内更新子标题，同时重建菜单项，如代码清单13-18所示。

代码清单13-18 更新菜单项（CrimeListFragment.java）

```
@Override  
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {  
    super.onCreateOptionsMenu(menu, inflater);  
    inflater.inflate(R.menu.fragment_crime_list, menu);  
  
    MenuItem subtitleItem = menu.findItem(R.id.menu_item_show_subtitle);  
    if (mSubtitleVisible) {  
        subtitleItem.setTitle(R.string.hide_subtitle);  
    } else {  
        subtitleItem.setTitle(R.string.show_subtitle);  
    }  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.menu_item_new_crime:  
            ...  
        case R.id.menu_item_show_subtitle:  
            mSubtitleVisible = !mSubtitleVisible;  
            getActivity().invalidateOptionsMenu();  
            updateSubtitle();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

最后，根据`mSubtitleVisible`变量值，联动菜单项标题与子标题，如代码清单13-19所示。

代码清单13-19 实现菜单项标题与子标题的联动（CrimeListFragment.java）

```
private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    if (!mSubtitleVisible) {
        subtitle = null;
    }

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

运行CriminalIntent应用，确认菜单项标题与子标题能够联动。

13.4.2 “还有个问题”

解决Android编程问题如同对付神探可伦坡^①的盘问。你以为你的解决方案无懈可击，可以高枕无忧了，但每次都会被Android堵在门口提醒道：“还有个问题没解决。”

准确地说，还有两个问题。首先，新建crime记录后，使用回退按钮回到`CrimeListActivity`界面，子标题显示的总记录数不会更新。其次，子标题显示后，旋转设备，显示的子标题会消失。

先看记录刷新问题。在返回`CrimeListActivity`界面时，再次刷新子标题显示就能解决这个问题。也就是说，在`onResume`方法里再次调用`updateSubtitle`方法。既然`onResume`和`onCreate`方法会调用`updateUI`方法，那就在`updateUI`方法里直接调用`updateSubtitle`方法（如代码清单13-20所示）。

代码清单13-20 显示最新状态（CrimeListFragment.java）

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

^① 美国经典电视电影系列《神探可伦坡》的主角。“还有个问题”（Just one more thing...）是他在破案过程中常说的一句话。——编者注

运行CriminalIntent应用。显示子标题，然后新建crime记录并按回退按钮返回到CrimeListActivity界面。可以看到，工具栏显示的总记录数没问题了。

现在，重复上述步骤，但这次改用向上按钮回退。注意看，子标题显示被重置了！这又是什么情况？

这是Android实现层级导航带来的问题：导航回退到的目标activity会被完全重建。既然父activity是全新的activity，实例变量值以及保存的实例状态显然会彻底丢失。

在向上导航时保证子标题的可见状态不容易。一种方案是覆盖向上导航的机制。在CriminalIntent应用中，调用CrimePagerActivity的finish方法直接回退到前一个activity界面。遗憾的是，这种方法只能回退一个层级，而实际开发的应用绝大多数都需要多层次导航。

另一种方案是启动CrimePagerActivity时，把子标题状态作为extra信息传给它。然后，在CrimePagerActivity中覆盖getParentActivityIntent()方法，用附带了extra信息的intent重建CrimeListActivity。这需要CrimePagerActivity类知道父类工作机制的细节。

上述两种方案都不够理想，但目前没有更好的方法。

无论如何，CriminalIntent应用的子标题显示算是解决了。现在解决设备旋转问题。只要利用实例状态保存机制，保存mSubtitleVisible实例变量值就能解决问题了，如代码清单13-21所示。

代码清单13-21 保存子标题状态值 (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private static final String SAVED_SUBTITLE_VISIBLE = "subtitle";

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        ...

        if (savedInstanceState != null) {
            mSubtitleVisible = savedInstanceState.getBoolean(SAVED_SUBTITLE_VISIBLE);
        }

        updateUI();

        return view;
    }

    @Override
    public void onResume() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
```

```

        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }
}

```

运行CriminalIntent应用。显示子标题并旋转设备，可以看到子标题在新建视图中依然能正确显示。

13.5 深入学习：工具栏与操作栏

工具栏和操作栏究竟有什么区别呢？

首先，两者给我们最直观的印象就是工具栏界面更美观。工具栏的左边不再放置图标，右边菜单项的间距也更小。另外就是向上的导航按钮。操作栏上的这个按钮不够醒目，只是旁边按钮的附属物。

除了感官上的差异，在使用上，工具栏比操作栏更灵活。操作栏的使用限制多多，比如，整个应用只能配置一个操作栏且位置及尺寸必须固定（在屏幕顶部）。工具栏就没有这些限制。

本章使用的工具栏应用了AppCompat主题。如有需要，也可以通过activity和fragment布局定制标准视图的工具栏。可以在屏幕的任何位置摆放工具栏，甚至可以在同一屏配置多个工具栏。应用设计的自由度由此大大提高了，例如，可以为每个fragment定制专用工具栏。可以想象，在同一个用户界面托管多个fragment时，每个fragment都由自己的工具栏控制，这比所有fragment共享一个位于屏幕顶部的工具栏方便多了。

此外，工具栏还能支持内嵌视图和调整高度，这极大丰富了应用的交互使用模式。毫不夸张地说，应用设计最大的局限就是我们的想象空间。

13.6 挑战练习：删除 crime 记录

CriminalIntent应用目前不支持删除现有crime记录。请为CrimeFragment添加菜单项，允许用户删除当前crime记录。用户点击删除菜单项后，记得调用CrimeFragment托管活动的finish方法回到前一个activity界面。

13.7 挑战练习：优化字符串资源显示

注意到没有，只有一条crime记录的时候，显示总记录数的子标题会显示：1 crimes。请改正这个粗心的语法错误。

实现思路上，你可以在代码中准备不同字符串资源分情况使用，但这会给应用本地化制造麻烦。比较好的做法是使用复数字符串资源（又称为量化字符串）。

首先，在strings.xml文件中定义复数字符串资源。

```

<plurals name="subtitle_plural">
    <item quantity="one">%1$s crime</item>
    <item quantity="other">%1$s crimes</item>
</plurals>

```

然后，使用`getQuantityString`方法正确处理单复数问题。

```
int crimeSize = crimeLab.getCrimes().size();
String subtitle = getResources()
    .getQuantityString(R.plurals.subtitle_plural, crimeSize, crimeSize);
```

13.8 挑战练习：用于 RecyclerView 的空视图

当前，CriminalIntent应用启动后，会显示一个空白列表。从用户体验上来讲，即使`crime`列表是空的，也应展示提示或解释类信息。

请设置空视图展示类似“没有`crime`记录可以显示”的信息。再添加一个按钮，方便用户直接创建新的`crime`记录。

判断`crime`列表是否包含数据，然后使用任何类都有的`setVisibility`方法控制占位视图的显示。

前面介绍的`savedInstanceState`无法满足应用持久化保存数据的需求。Android为此提供了长期存储地：手机或平板设备闪存上的本地文件系统。

Android设备上的应用都有一个沙盒目录。将文件保存在沙盒中，可阻止其他应用甚至是设备用户的访问和窥探。（当然，设备被root了的话，用户就可以随意访问各种目录和文件了。）

应用沙盒目录是`/data/data/[your package name]`的子目录。例如，`CriminalIntent`应用的沙盒目录是`/data/data/com.bignerdranch.android.criminalintent`。

需要保存大量数据时，大多数应用都不会使用类似`txt`这样的普通文件。原因很简单：假设将`crime`记录写入了这样的文件，在仅需要修改`crime`标题时，就得首先读取整个文件的内容，完成修改后再全部保存。数据量大的话，这将非常耗时。

怎么办呢？该是SQLite闪亮登场的时候了。SQLite是类似于MySQL和Postgresql的开源关系型数据库。不同于其他数据库的是，SQLite使用单个文件存储数据，使用SQLite库读取数据。Android标准库包含SQLite库以及配套的一些Java辅助类。

本章，我们仅学习如何使用SQLite基本辅助类，打开应用沙盒中的数据库，以及读取和写入数据。如需深入学习，请访问网站<http://www.sqlite.org>，阅读SQLite完全使用手册。

14.1 定义 Schema

创建数据库前，首先要清楚存储什么样的数据。`CriminalIntent`应用要保存的是一条条`crime`记录，这需要定义如图14-1所示的`crimes`数据表。

<code>_id</code>	<code>uuid</code>	<code>title</code>	<code>date</code>	<code>solved</code>
1	13090636733242	Stolen yogurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

图14-1 `crimes`数据表

定义Schema的方式众多，如何选择往往因人而异。处理类似的任务，开发人员都有个共同的目标：“不要重复造轮子。”实际上，这也是人人都应遵守的编程准则：多花时间思考复用代码的编写和调用，避免在应用中到处使用重复代码。

基于上述准则，我们可以使用能统一定义模型层对象（如Crime）的高级ORM（对象关系映射）工具。不过，对于CriminalIntent应用，本章打算直接在Java代码中定义描述表名和数据字段的数据库schema。

首先，我们来创建定义schema的Java类。创建时，命名类为CrimeDbSchema，同时在新建类对话框中输入包名database.CrimeDbSchema。这样，就可以将CrimeDbSchema.java文件放入专门的database包中，实现数据库操作相关代码的组织和归类。

在CrimeDbSchema类中，再定义一个描述数据表的CrimeTable内部类，如代码清单14-1所示。

代码清单14-1 定义CrimeTable内部类 (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";
    }
}
```

CrimeTable内部类唯一的用途就是定义描述数据表元素的String常量。首先要定义的是数据库表名 (CrimeTable.NAME)。

接下来定义数据表字段，如代码清单14-2所示。

代码清单14-2 定义数据表字段 (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

有了这些数据表元素，就可以在Java代码中安全地引用了。例如，CrimeTable.Cols.TITLE就是指crime记录的title字段。此外，这种定义方式还给修改字段名称或新增表元素带来了方便。

14.2 创建初始数据库

定义完数据库 schema，就可以创建数据库了。openOrCreateDatabase(...) 和 databaseList()方法是Android提供的Context底层方法，可以用来打开数据库文件并将其转换为SQLiteDatabase实例。

不过，实际开发时，建议总是遵循以下步骤。

- (1) 确认目标数据库是否存在。
- (2) 如果不存在，首先创建数据库，然后创建数据库表以及必需的初始化数据。

(3) 如果存在，打开并确认CrimeDbSchema是否是最新版本(后续章节可能会在CriminalIntent中有增删操作)。

(4) 如果是旧版本，就运行相关代码升级到最新版本。

令人高兴的是，Android提供的SQLiteOpenHelper类可以帮助我们处理这些。在数据库包中创建CrimeBaseHelper类，如代码清单14-3所示。

代码清单14-3 创建CrimeBaseHelper类 (CrimeBaseHelper.java)

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

有了SQLiteOpenHelper类，打开SQLiteDatabase的繁杂工作都可以交给它处理。在CrimeLab中用它创建crime数据库，如代码清单14-4所示。

代码清单14-4 打开SQLiteDatabase (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;

    ...

    private CrimeLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new CrimeBaseHelper(mContext)
            .getWritableDatabase();
        mCrimes = new ArrayList<>();
    }

    ...
}
```

(你可能会问，为什么要把context赋值给实例变量呢？不要奇怪，CrimeLab会在第16章用到

它。)

这里调用`getWritableDatabase()`方法时，`CrimeBaseHelper`要做如下工作。

(1) 打开`/data/data/com.bignerdranch.android.criminalintent/databases/crimeBase.db`数据库；如果不存在，就先创建`crimeBase.db`数据库文件。

(2) 如果是首次创建数据库，就调用`onCreate(SQLiteDatabase)`方法，然后保存最新的版本号。

(3) 如果已创建过数据库，首先检查它的版本号。如果`CrimeOpenHelper`中的版本号更高，就调用`onUpgrade(SQLiteDatabase, int, int)`方法升级。

最后，再做个总结：`onCreate(SQLiteDatabase)`方法负责创建初始数据库；`onUpgrade(SQLiteDatabase, int, int)`方法负责与升级相关的工作。

`CriminalIntent`当前只有一个版本，暂时可以不用操心`onUpgrade(...)`方法。我们在`onCreate(...)`方法中创建数据库表，这需要导入`CrimeDbSchema`类的`CrimeTable`内部类。

导入分两步完成。首先，编写SQL创建初始代码，如代码清单14-5所示。

代码清单14-5 编写SQL创建初始代码（`CrimeBaseHelper.java`）

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeDbSchema.CrimeType.NAME);
}
```

把光标定位到`CrimeType`一词上，按`Option+Return`（或`Alt+Enter`）组合键，然后选择提示里的`Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeType'`可选项，如图14-2所示。

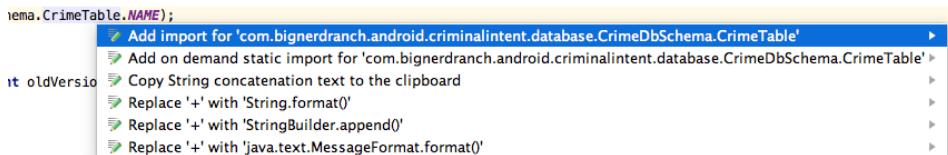


图14-2 导入`CrimeType`

Android Studio会自动生成如下导入语句：

...

```
import com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeType;
```

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
```

```
    ...
```

这样，我们就可以直接以`CrimeType.Cols.UUID`的形式使用`CrimeDbSchema.CrimeType`中的`String`常量了（不用费事地输入`CrimeDbSchema.CrimeType.Cols.UUID`）。完成其余表定义代

码，如代码清单14-6所示。

代码清单14-6 创建crime表（CrimeBaseHelper.java）

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
        "_id integer primary key autoincrement, " +
        CrimeTable.Cols.UUID + ", " +
        CrimeTable.Cols.TITLE + ", " +
        CrimeTable.Cols.DATE + ", " +
        CrimeTable.Cols.SOLVED +
    ")"
);
}
```

相比其他数据库，创建SQLite数据库表省事又简单：创建表字段时，不需要指定表字段类型。

运行CriminalIntent应用后，应该可以看到新建数据库文件了，如图14-3所示。在模拟器或是已root的设备上，可直接看到已创建的数据库文件。如果是模拟器，请选择Tools → Android → Android Device Monitor菜单项，然后查看/data/data/com.bignerdranch.android.criminalintent/databases/目录。

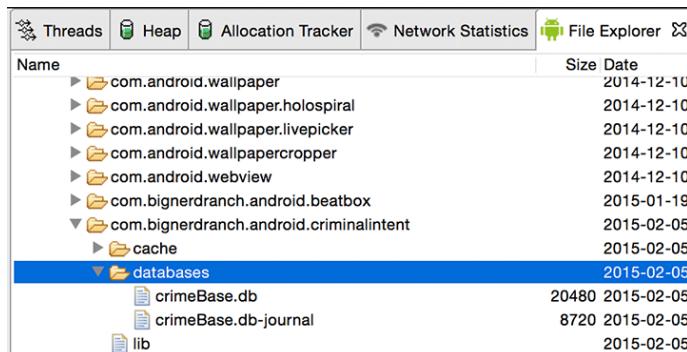


图14-3 数据库文件已生成

处理数据库相关问题

编写SQLite数据库操作代码时，经常会碰到调整数据库表结构的情况。例如，我们还要在后续章节中添加crime嫌疑人。这就需要为crime数据表新增字段。比较常规的做法是，在SQLiteOpenHelper记录版本号，然后在onUpgrade(...)方法中升级数据表。

这种常规方法涉及不少代码量。而且，编写和维护很少更新版本的代码也很伤脑筋。所以，在实际开发中，最好的做法是直接删除数据库文件，让SQLiteOpenHelper.onCreate(...)方法重新创建它。

直接从设备上删除应用是删除数据库文件最便捷的方法。要在模拟器上删除应用，可以切换

到应用浏览器界面，向上拖动CriminalIntent应用图标，直到屏幕顶部出现Uninstall字样（注意，不同Android版本的操作可能有所差别）。执行并确认删除应用，如图14-4所示。

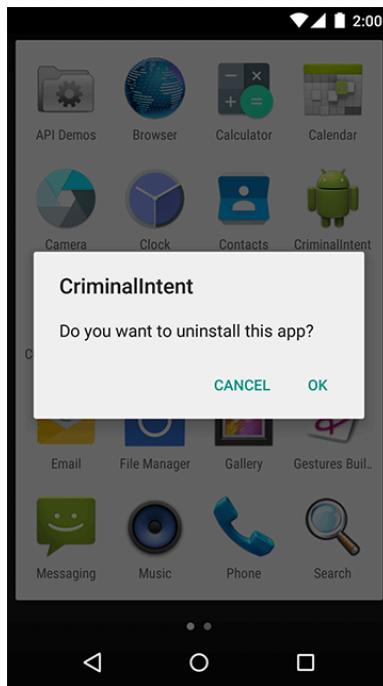


图14-4 删除应用

记住，在本章学习过程中，如遇数据库相关问题，可尝试直接删除应用，重头来过。

14.3 修改 CrimeLab 类

创建完数据库，接下来是调整CrimeLab类代码，改用mDatabase存储数据。

首先要删除CrimeLab中所有mCrimes相关的代码，如代码清单14-7所示。

代码清单14-7 删除mCrimes相关代码（CrimeLab.java）

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;

    public static CrimeLab get(Context context) {
        ...
    }
}
```

```

private CrimeLab(Context context) {
    mContext = context.getApplicationContext();
    mDatabase = new CrimeBaseHelper(mContext)
        .getWritableDatabase();
    mCrimes = new ArrayList<>();
}

public void addCrime(Crime c) {
    mCrimes.add(c);
}

public List<Crime> getCrimes() {
    return mCrimes;
    return new ArrayList<>();
}

public Crime getCrime(UUID id) {
    for (Crime crime : mCrimes) {
        if (crime.getId().equals(id)) {
            return crime;
        }
    }
    return null;
}
}

```

代码调整完毕，运行CriminalIntent应用只会看到空列表和空CrimePagerActivity。没关系，下面我们就来逐步完善它。

14.4 写入数据库

要使用`SQLiteDatabase`，数据库中首先要有数据。数据库写入操作有：向`crime`表中插入新记录以及在`Crime`变更时更新原始记录。

14.4.1 使用 ContentValues

负责处理数据库写入和更新操作的辅助类是`ContentValues`。它是个键值存储类，类似于Java的`HashMap`和前面用过的`Bundle`。不同的是，`ContentValues`只能用于处理`SQLite`数据。

将`Crime`记录转换为`ContentValues`实际就是在`CrimeLab`中创建`ContentValues`实例。我们需要新建一个私有方法，如代码清单14-8所示。（记住使用前述的两个步骤导入`CrimeTable`：把光标定位到`CrimeTable.Cols.UUID`上，按`Option+Return`（或`Alt+Enter`）组合键，然后选择提示里的`Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable'`可选项。）

代码清单14-8 创建ContentValues（CrimeLab.java）

```

public getCrime(UUID id) {
    return null;
}

```

```

    }

    private static ContentValues getContentValues(Crime crime) {
        ContentValues values = new ContentValues();
        values.put(CrimeTable.Cols.UUID, crime.getId().toString());
        values.put(CrimeTable.Cols.TITLE, crime.getTitle());
        values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
        values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);

        return values;
    }
}

```

`ContentValues`的键就是数据表字段。注意不要搞错，否则会导致数据插入和更新失败。除了`_id`是由数据库自动创建外，其他所有数据表字段都要编码指定。

14.4.2 插入和更新记录

准备好`ContentValues`，是时候向数据库写入数据了。参照代码清单14-9，在`addCrime(Crime)`方法中新增数据插入实现代码。

代码清单14-9 插入记录（CrimeLab.java）

```

public void addCrime(Crime c) {
    ContentValues values = getContentValues(c);

    mDatabase.insert(CrimeTable.NAME, null, values);
}

```

`insert(String, String, ContentValues)`方法有两个重要的参数，还有一个很少用到。传入的第一个参数是数据库表名，最后一个是要写入的数据。

第二个参数称为`nullColumnHack`。它有什么用途呢？

别急，举个例子你就明白了。假设你想调用`insert(...)`方法，但传入了`ContentValues`空值。这时，SQLite不干了，`insert(...)`方法调用只能以失败告终。

然而，如果能以`uuid`值作为`nullColumnHack`传入的话，SQLite就可以忽略`ContentValues`空值，而且还会自动传入一个带`uuid`且值为`null`的`ContentValues`。结果，`insert(...)`方法得以成功调用并插入了一条新记录。

听起来不错吧，也许某天会用得着；但肯定不是现在。因此，你目前只要了解就可以了。

完成了数据插入，下面继续使用`ContentValues`，新增数据库记录更新方法，如代码清单14-10所示。

代码清单14-10 更新记录（CrimeLab.java）

```

public Crime getCrime(UUID id) {
    return null;
}

public void updateCrime(Crime crime) {

```

```

String uuidString = crime.getId().toString();
ContentValues values = getContentValues(crime);

mDatabase.update(CrimeTable.NAME, values,
    CrimeTable.Cols.UUID + " = ?",
    new String[] { uuidString });
}

private static ContentValues getContentValues(Crime crime) {
    ContentValues values = new ContentValues();
    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
    ...
}

update(String, ContentValues, String, String[])更新方法类似于insert(...)方法，向其传入要更新的数据表名和为表记录准备的ContentValues。然而，与insert(...)方法不同的是，你要确定该更新哪些记录。具体的做法是：创建where子句（第三个参数），然后指定where子句中的参数值（String[]数组参数）。

```

问题来了，为什么不直接在where子句中放入uuidString呢？这可比使用?然后传入String[]简单多了！

事实上，很多时候，String本身会包含SQL代码。如果将它直接放入query语句中，这些代码可能会改变query语句的含义，甚至会修改数据库资料。这实际就是SQL脚本注入，危害相当严重。

使用?的话，就不用关心String包含什么，代码执行的效果肯定就是我们想要的。因此，建议你保持这种良好的代码编写习惯。

用户可能会在CrimeFragment中修改Crime实例。修改完成后，我们需要刷新CrimeLab中的Crime数据。这可以通过在CrimeFragment.java中覆盖CrimeFragment.onPause()方法完成，如代码清单14-11所示。

代码清单14-11 Crime数据刷新（CrimeFragment.java）

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}

@Override
public void onPause() {
    super.onPause();

    CrimeLab.get(getActivity())
        .updateCrime(mCrime);
}

```

数据库写入部分处理完了。不过，要等到数据库读取也处理完才能检验代码。继续学习之前，可以运行CriminalIntent应用看看代码能否正常编译。一切正常的话，你应该看到的是个Crime空列表。

14.5 读取数据库

读取SQLite数据库中数据需要用到query(...)方法。这个方法有好几个重载版本。我们要用的版本如下：

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

以前写过SQL代码的话，你应该已经熟悉这些select语句参数了。不熟悉也没关系，只关注我们马上要用的就好。

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

参数table是要查询的数据表。参数columns指定要依次获取哪些字段的值。参数where和whereArgs的作用与update(...)方法中的一样。

新增一个便利方法调用query(...)方法查询CrimeTable中的记录，如代码清单14-12所示。

代码清单14-12 查询crime记录 (CrimeLab.java)

```
...
values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);

return values;
}

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
    Cursor cursor = mDatabase.query(
        CrimeTable.NAME,
        null, // Columns - null selects all columns
        whereClause,
        whereArgs,
        null, // groupBy
        null, // having
        null // orderBy
    );
}
```

```
    return cursor;
}
```

14.5.1 使用 CursorWrapper

`Cursor`是个神奇的表数据处理工具，其任务就是封装数据表中的原始字段值。从`Cursor`获取数据的代码大致如下所示：

```
String uuidString = cursor.getString(
    cursor.getColumnIndex(CrimeType.Cols.UUID));
String title = cursor.getString(
    cursor.getColumnIndex(CrimeType.Cols.TITLE));
long date = cursor.getLong(
    cursor.getColumnIndex(CrimeType.Cols.DATE));
int isSolved = cursor.getInt(
    cursor.getColumnIndex(CrimeType.Cols.SOLVED));
```

每从`Cursor`中取出一条`crime`记录，以上代码都要重复写一次。（这还不包括按照这些字段值创建`Crime`实例的代码。）

显然，遇到这种情况，我们应考虑到前面说过的代码复用原则。与其机械地编写重复代码，不如创建可复用的专用`Cursor`子类。使用`CursorWrapper`可快速方便地创建`Cursor`子类。顾名思义，`CursorWrapper`能够封装一个个`Cursor`的对象，并允许在其上添加新的有用方法。

参照代码清单14-13，在数据库包中新建`CrimeCursorWrapper`类。

代码清单14-13 创建`CrimeCursorWrapper`类（`CrimeCursorWrapper.java`）

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }
}
```

可以看到，以上代码创建了一个`Cursor`封装类。该类继承了`Cursor`类的全部方法。注意，这样封装的目的就是为了定制新方法，以方便操作内部`Cursor`。

参照代码清单14-14，新增获取相关字段值的`getCrime()`方法。（别忘了使用前面介绍的两步导入`CrimeType`的技巧。）

代码清单14-14 新增`getCrime()`方法（`CrimeCursorWrapper.java`）

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }

    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeType.Cols.UUID));
        String title = getString(getColumnIndex(CrimeType.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeType.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeType.Cols.SOLVED));
```

```

        return null;
    }
}

```

我们需要返回具有UUID的Crime。在Crime.java中新增一个有此用途的构造方法，如代码清单14-15所示。

代码清单14-15 新增Crime构造方法（Crime.java）

```

public Crime() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}

public Crime(UUID id) {
    mId = id;
    mDate = new Date();
}

```

最后，完成getCrime()方法，如代码清单14-16所示。

代码清单14-16 新增getCrime()方法（CrimeCursorWrapper.java）

```

public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);

    return crime;
    return null;
}

```

（Android Studio会让你确定是选择java.util.Date还是java.sql.Date。不要搞错，即便我们现在是在编写数据库相关代码，也应该选java.util.Date。）

14.5.2 创建模型层对象

使用CrimeCursorWrapper类，可直接从CrimeLab中取得List<Crime>。大致思路无外乎将查询返回的cursor封装到CrimeCursorWrapper类中，然后调用getCrime()方法遍历取出Crime。

首先让queryCrimes(...)方法返回CrimeCursorWrapper对象，如代码清单14-17所示。

代码清单14-17 使用cursor封装方法 (CrimeLab.java)

```

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
    private CrimeCursorWrapper queryCrimes(String whereClause, String[] whereArgs) {
        Cursor cursor = mDatabase.query(
            CrimeTable.NAME,
            null, // Columns - null selects all columns
            whereClause,
            whereArgs,
            null, // groupBy
            null, // having
            null // orderBy
        );

        return cursor;
        return new CrimeCursorWrapper(cursor);
    }
}

```

然后，完善getCrime()方法：遍历查询取出所有的crime，返回Crime数组对象，如代码清单14-18所示。

代码清单14-18 返回crime列表 (CrimeLab.java)

```

public List<Crime> getCrimes() {
    return new ArrayList<>();
    List<Crime> crimes = new ArrayList<>();

    CrimeCursorWrapper cursor = queryCrimes(null, null);

    try {
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            crimes.add(cursor.getCrime());
            cursor.moveToNext();
        }
    } finally {
        cursor.close();
    }

    return crimes;
}

```

数据库cursor之所以被称为cursor，是因为它内部就像有根手指似的，总是指向查询的某个地方。因此，要从cursor中取出数据，首先要调用moveToFirst()方法移动虚拟手指指向第一个元素。读取行记录后，再调用moveToNext()方法，读取下一行记录，直到isAfterLast()告诉我们没有数据可取为止。

最后，别忘了调用Cursor的close()方法关闭它。否则，后果很严重：轻则看到应用报错，重则导致应用崩溃。

CrimeLab.getCrime(UUID)方法类似于getCrimes()方法，唯一区别就是它只需要取出已存在的首条记录，如代码清单14-19所示。

代码清单14-19 重写getCrime(UUID)方法(CrimeLab.java)

```

public Crime getCrime(UUID id) {
    return null;
    CrimeCursorWrapper cursor = queryCrimes(
        CrimeTable.Cols.UUID + " = ?",
        new String[] { id.toString() }
    );
    try {
        if (cursor.getCount() == 0) {
            return null;
        }

        cursor.moveToFirst();
        return cursor.getCrime();
    } finally {
        cursor.close();
    }
}

```

上述代码的作用如下。

- 现在可以插入crime记录了。也就是说，点击New Crime菜单项，实现将Crime添加到CrimeLab的代码可以正常工作了。
- 数据库查询没有问题了。CrimePagerActivity现在能够看见CrimeLab中的所有Crime了。
- CrimeLab.getCrime(UUID)方法也能正常工作了。CrimePagerActivity托管的CrimeFragment终于可以显示真正的Crime对象了。

现在，点击New Crime菜单项可以在CrimePagerActivity界面看到新增Crime了。运行CriminalIntent应用确认无问题发生。

刷新模型层数据

本章的工作还没有结束。虽然Crime记录已存入数据库，但数据读取还未完善。例如，编辑完新的crime后，尝试点击回退按钮，你会发现CrimePagerActivity并没有相应刷新。

这是因为CrimeLab的工作方式已经改变。以前，只有一个List<Crime>，而且每个Crime在List<Crime>中只存有一个对象。要获取哪个Crime只能去找mCrimes。

现在，mCrimes已废弃不用了。因此，getCrimes()方法返回的List<Crime>是Crime对象的快照。要刷新CrimeListActivity界面，首先要更新这个快照。

好在大部分基础工作已经就绪。CrimeListActivity目前调用updateUI()方法来刷新界面。剩下的事情就是刷新CrimeLab的视图了。

要刷新crime显示，我们首先添加一个setCrimes(List<Crime>)方法给CrimeAdapter，如代码清单14-20所示。

代码清单14-20 添加setCrimes(List<Crime>)方法(CrimeListFragment.java)

```

private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
}

```

```

@Override
public int getItemCount() {
    return mCrimes.size();
}

public void setCrimes(List<Crime> crimes) {
    mCrimes = crimes;
}
}

```

然后在updateUI()方法中调用setCrimes(List<Crime>)方法，如代码清单14-21所示。

代码清单14-21 调用setCrimes(List<Crime>)方法 (CrimeListFragment.java)

```

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.setCrimes(crimes);
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}

```

现在，可以验证我们的成果了。运行CriminalIntent应用，新增一项crime记录，然后按回退键，确认CrimeListActivity中会出现刚才新增的记录。

此外，还建议验证CrimeFragment中的updateCrime(Crime)方法调用是否正常。点击任意crime列表项，在CrimePagerActivity中编辑它的标题。然后按回退键，确认对应列表项的标题得到更新。

14.6 深入学习：数据库高级主题介绍

为了简化教学及控制篇幅，本章没有讨论数据库应用的方方面面。要知道，很多专业应用都需要高级数据库使用支持。为了提高开发效率，人们常常会求助于ORM这样专业又复杂的工具。

开发复杂的数据库应用时，难免会需要以下数据库元素。

- **字段类型。**从技术实现上讲，SQLite的字段不需要指定数据类型。没有它们完全不影响数据库使用；然而，如果能在合适的地方给出数据类型提示会更好。
- **索引。**查询数据库字段时，字段不加索引会严重影响查询性能和效率。
- **外键。**本章使用数据库时只用了一张表。如果涉及多张表，关联数据应该需要外键约束。

数据库性能优化需要考虑的因素众多。每次查询数据库时，CriminalIntent应用都会创建包含所有Crime对象的列表。要获得高性能表现，需要采取一些优化措施。比如，循环使用Crime实

例，或者把它们当作内存对象。显然，这涉及更多的代码量。当然，这也是ORM这样的专业工具要解决的问题。

14.7 深入学习：应用上下文

前面，我们在CrimeLab的构造方法中使用了应用上下文。

```
private CrimeLab(Context context) {  
    mContext = context.getApplicationContext();  
    ...  
}
```

应用上下文有什么特别呢？就上例来看，为什么要用应用上下文，而不直接用activity作为context呢？

要回答上述问题，关键就在于考虑它们的生命周期。只要有activity在，Android肯定也创建有application对象。用户在应用的不同界面间导航时，各个activity时而存在时而消亡，但application对象不会受任何影响。可以说，它的生命周期要比任何activity都要长。

CrimeLab是个单例。这表明，一旦创建，它就会一直存在直至整个应用进程被销毁。由代码可知，CrimeLab引用着mContext对象。显然，如果把activity作为mContext对象保存的话，这个由CrimeLab一直引用着的activity肯定会免遭垃圾回收器的清理，即便用户跳转离开这个activity时也是如此。

为了避免资源浪费，我们使用了应用程序上下文。这样，CrimeLab仍可以引用Context对象，而activity的存在和消亡也不用受它束缚了。

14.8 挑战练习：删除Crime记录

如果已为应用添加过Delete Crime菜单项的话，就可以直接调用CrimeLab的deleteCrime(Crime)方法，继而调用mDatabase.delete(...)方法来实现删除功能。

如果还没有，那就先给CrimeFragment的工具栏添加一个Delete Crime菜单项，然后调用CrimeLab.deleteCrime(Crime)方法实现删除功能。

在Android系统中，可利用隐式intent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，我们只要描述要完成的任务，操作系统就会找到合适的应用，并在其中启动相应的activity。

本章，我们将使用隐式intent发送短消息给Crime嫌疑人。用户首先从某个联系人应用中选取联系人，然后从短消息应用列表中选取目标应用发送消息，如图15-1所示。



图15-1 打开联系人应用和消息发送应用

对开发者来说，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。对用户来说，他们也乐意在你的应用中调用自己熟悉或喜爱的应用。

创建隐式intent之前，需完成以下准备工作：

- 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消息模板。

15.1 添加按钮组件

首先，我们要在CrimeFragment布局中添加两个投诉用按钮：一个嫌疑人选取按钮（suspect按钮）和一个消息发送按钮（report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15-1所示。

代码清单15-1 添加按钮用字符串（strings.xml）

```
...
<string name="subtitle_format">%1$s crimes</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15-2，添加两个按钮组件。注意，为突出重点，我们在布局定义示意图中隐藏了第一个线性布局及其全部子元素。

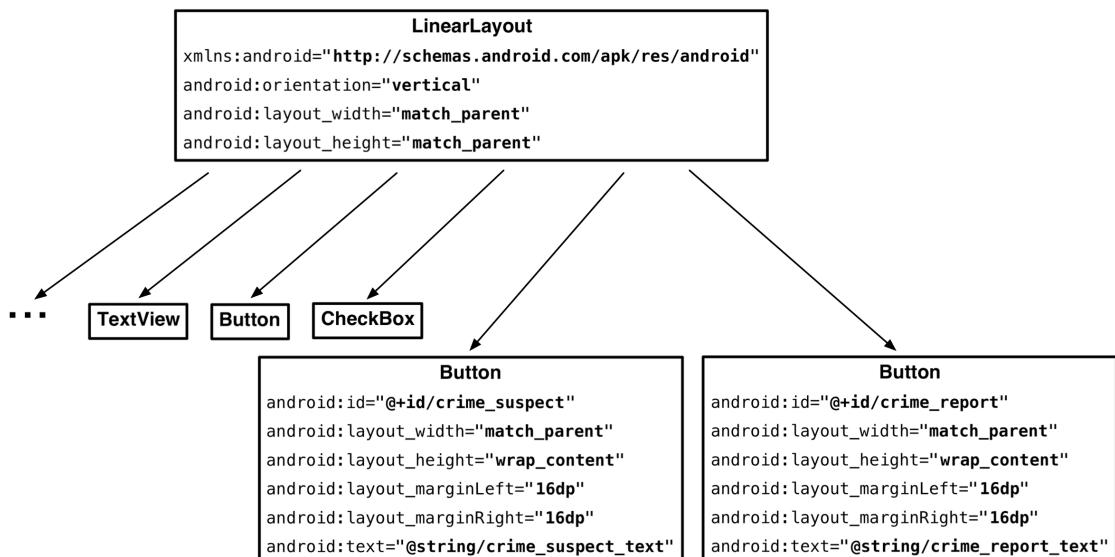


图15-2 添加嫌疑人选取和消息发送按钮（layout/fragment_crime.xml）

在水平模式布局中，需要将新增按钮作为子元素添加到新的水平线性布局中，并置于包含日期按钮和Solved单选框的线性布局下方。最后的效果如图15-3所示。

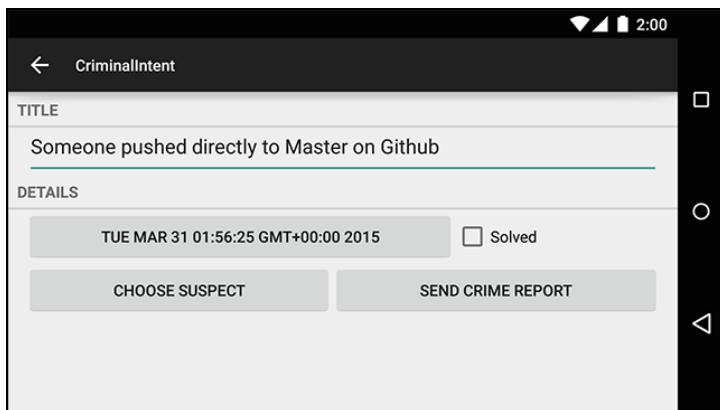


图15-3 水平模式布局

在layout-land/fragment_crime.xml布局中，参照图15-4，添加线性布局和两个按钮组件。

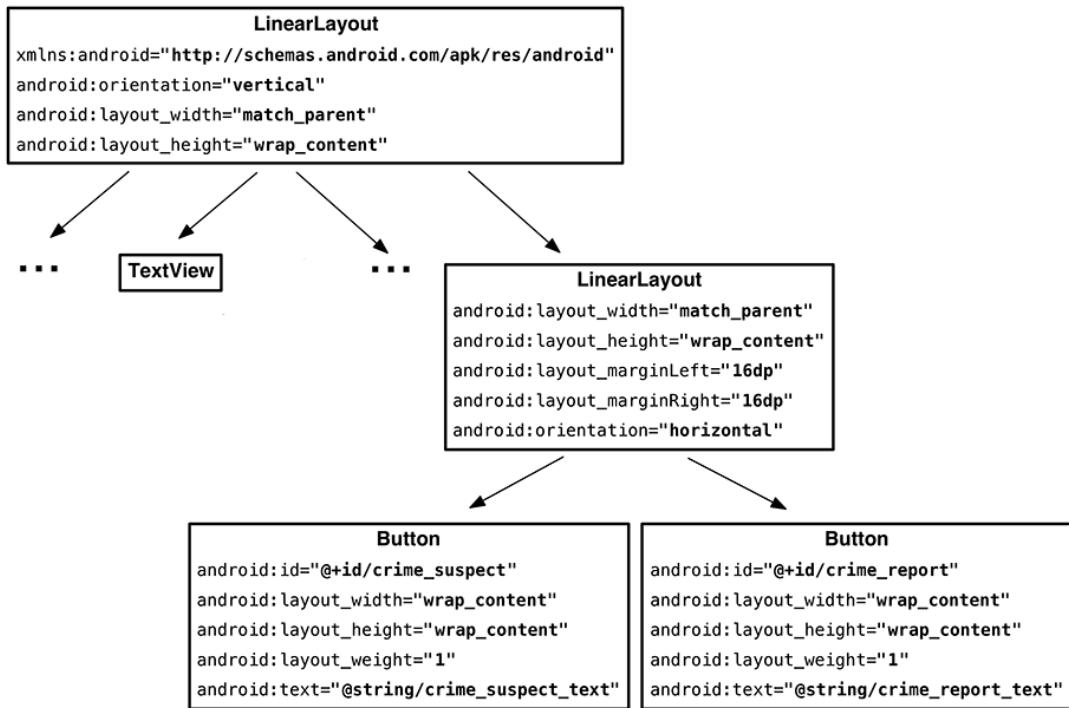


图15-4 添加嫌疑人选取和消息发送按钮（layout-land/fragment_crime.xml）

现在，可以预览新布局了。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。

15.2 添加嫌疑人信息至模型层

接下来，返回到Crime.java中，新增存储嫌疑人姓名的mSuspect成员变量，如代码清单15-2所示。

代码清单15-2 添加mSuspect成员变量（Crime.java）

```
public class Crime {

    ...
    private boolean mSolved;
private String mSuspect;

    public Crime() {
        this(UUID.randomUUID());
    }

    ...

    public void setSolved(boolean solved) {
        mSolved = solved;
    }

    public String getSuspect() {
        return mSuspect;
    }

    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

现在，需要新增crime数据库字段。首先，在数据库schema中定义嫌疑人字段，如代码清单15-3所示。

代码清单15-3 添加嫌疑人数据库字段（CrimeDbSchema.java）

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
            public static final String SUSPECT = "suspect";
        }
    }
}
```

同时，也要在CrimeBaseHelper中新增嫌疑人数据库字段，如代码清单15-4所示。（注意，别忘了CrimeTable.Cols.SOLVED后的逗号。）

代码清单15-4 添加嫌疑人数据库字段 (CrimeBaseHelper.java)

```

@Override
public void onCreate(SQLiteDatabase db) {

    db.execSQL("create table " + CrimeTable.NAME + "(" +
        "_id integer primary key autoincrement, " +
        CrimeTable.Cols.UUID + ", " +
        CrimeTable.Cols.TITLE + ", " +
        CrimeTable.Cols.DATE + ", " +
        CrimeTable.Cols.SOLVED + ", " +
        CrimeTable.Cols.SUSPECT +
        ")"
    );
}

```

接下来，更新CrimeLab.getContentValues(Crime)方法中的数据库写入代码，如代码清单15-5所示。

代码清单15-5 添加嫌疑人数据库字段 (CrimeLab.java)

```

...
private static ContentValues getContentValues(Crime crime) {
    ContentValues values = new ContentValues();
    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
    values.put(CrimeTable.Cols.TITLE, crime.getTitle());
    values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
    values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
    values.put(CrimeTable.Cols.SUSPECT, crime.getSuspect());

    return values;
}
...

```

最后，更新CrimeCursorWrapper中的数据库读取代码，如代码清单15-6所示。

代码清单15-6 读取嫌疑人信息 (CrimeCursorWrapper.java)

```

...
public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
    String suspect = getString(getColumnIndex(CrimeTable.Cols.SUSPECT));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);
    crime.setSuspect(suspect);

    return crime;
}

```

注意，如果设备上已安装CriminalIntent应用，当前应用中的数据库是不包含嫌疑人字段的，而且`onCreate(SQLiteDatabase)`方法也不会添加这个新字段。这时，最容易的解决办法就是删除旧数据库（这是开发过程中常有的事）。

如前所述，首先删除CriminalIntent应用。然后，从Android Studio中重新运行安装它。这样，带有新增字段的数据库就创建成功了。

15.3 使用格式化字符串

最后一项准备工作是创建消息模板。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以及另外四个字符串参数（与要替换的占位符顺序一致）。

首先，在`strings.xml`中，添加代码清单15-7所示的字符串资源。

代码清单15-7 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，并返回拼接完整的消息，如代码清单15-8所示。

代码清单15-8 新增getCrimeReport()方法（CrimeFragment.java）

```
...
private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}

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;
}

```

(注意, 有两个DateFormat类: android.text.format.DateFormat和java.text.DateFormat。我们要用的是android.text.format.DateFormat。)

至此, 准备工作全部完成了, 接下来学习如何使用隐式intent。

15.4 使用隐式 intent

Intent对象用来向操作系统说明需要处理的任务。使用显式intent, 我们需指定要操作系统启动哪个activity。下面是之前创建过的显式intent:

```

Intent intent = new Intent(getActivity(), CrimePagerActivity.class);
intent.putExtra(EXTRA_CRIME_ID, crimeId);
startActivity(intent);

```

使用隐式intent, 只需告诉操作系统我们想要做什么, 操作系统就会去启动能够胜任工作任务的activity。如果找到多个符合的activity, 用户会看到一个可选应用列表, 然后就看用户如何选择了。

15.4.1 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包含数据位置,

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、何地或者如何使用某个activity的。例如，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的activity。（如果有多个应用适用，用户可自由选择。）

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。例如，如果想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`类别必须在intent过滤器中明确地设置。`action`元素告诉操作系统，activity能够胜任指定任务，`DEFAULT`类别告诉操作系统，activity愿意处理某项任务。`DEFAULT`类别实际隐含于所有隐式intent中。（当然也有例外，详见第22章。）

和显式intent一样，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，不会使用附加在隐式intent上的任何extra。

注意，显式intent也可以使用隐式intent的操作和数据部分。这相当于要求特定activity去处理特定任务。

15.4.2 发送消息

在CriminalIntent应用中，我们通过创建发送消息的隐式intent，来看看它是如何工作的。消息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15-9所示。

代码清单15-9 发送消息（CrimeFragment.java）

```
private Crime mCrime;
private EditText mTitleField;
private Button mDateButton;
private CheckBox mSolvedCheckbox;
private Button mReportButton;

...
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mReportButton = (Button) v.findViewById(R.id.crime_report);
    mReportButton.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。关于其他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。

消息内容和主题是作为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，自然也知道该如何使用它们的关联值。

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，我们很可能看到长长候选activity列表，如图15-5所示。

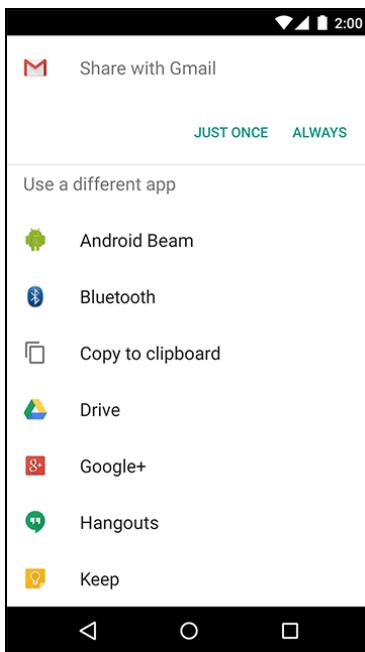


图15-5 支持发送消息的全部activity

从列表中作出选择后，可以看到消息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，我们有时可能看不到候选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，要么是设备上仅有一个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，也可以创建每次都显示的activity选择器。和以前一样创建隐式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15-10所示。

代码清单15-10 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，只要有两个activity可以处理隐式intent，我们就会得到一个候选activity列表，如图15-6所示。

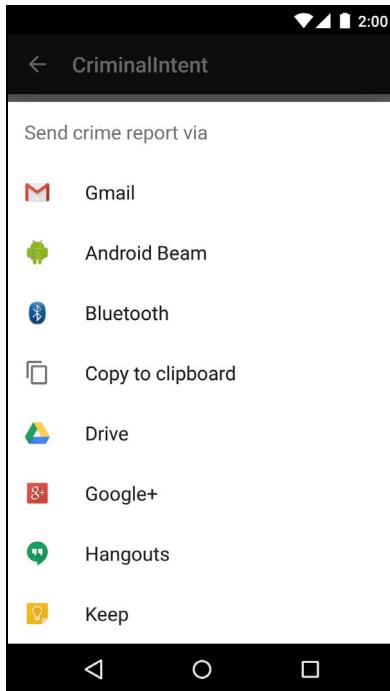


图15-6 通过选择器选择应用发送信息

15.4.3 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。简而言之，就是请Android帮忙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，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15-11所示。

代码清单15-11 添加suspect按钮成员变量（CrimeFragment.java）

```

...
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;

...
private CheckBox mSolvedCheckbox;
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15-12所示。

代码清单15-12 发送隐式intent（CrimeFragment.java）

```

public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    final Intent pickContact = new Intent(Intent.ACTION_PICK,
        ContactsContract.Contacts.CONTENT_URI);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivityForResult(pickContact, REQUEST_CONTACT);
        }
    });
    ...
    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }
    return v;
}

```

稍后还会使用pickContact，所以这里没有将它放在OnClickListener监听器代码中。运行CriminalIntent应用并点击CHOOSE SUSPECT按钮，我们会看到一个类似图15-7所示的联系人列表。

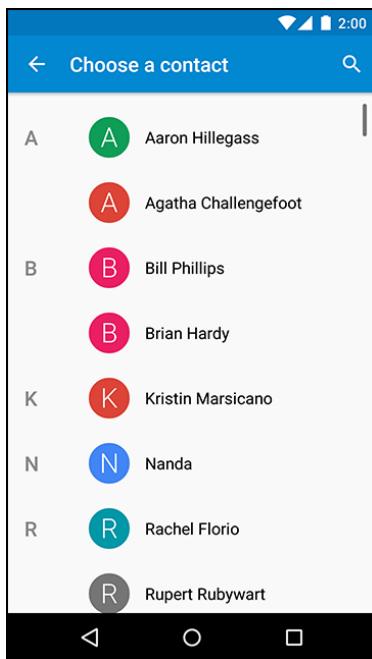


图15-7 包含嫌疑人的联系人列表

注意，如果设备上安装了其他联系人应用，应用界面可能会有所不同。另外可以看到，从当前应用中调用联系人应用时，我们完全不用知道应用的名字。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。隐式intent真好用！

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。很多应用都会共享联系人信息，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。

前面，我们以ACTION_PICK启动了activity并要求返回结果，因此调用onActivityResult(...)方法会接收到一个intent。该intent包括了数据URI。这个URI是个数据定位符，指向用户所选联系人。

在CrimeFragment.java中，将代码清单15-13所示代码添加到onActivityResult(...)实现方法中。

代码清单15-13 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) {
        ...
        updateDate();

    } else if (requestCode == REQUEST_CONTACT && data != null) {
        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, null);

        try {
            // Double-check that you actually got results
            if (c.getCount() == 0) {
                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);
        } finally {
            c.close();
        }
    }
}

```

代码清单15-13创建了一条查询语句，要求返回全部联系人的名字。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。

(联系人数据库是个比较复杂的主题，这里不会展开讨论。如需详细了解，可以阅读Contacts Provider API指南：[http://developer.android.com/guide/topics/providers/contacts-provider.html。\)](http://developer.android.com/guide/topics/providers/contacts-provider.html。)

现在可以运行应用测试了。有些设备可能没有联系人应用可用，如果是这样，请使用模拟器。

2. 联系人信息使用权限

我们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赋给了我们。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，会添加一个Intent.FLAG_GRANT_READ_URI_PERMISSION标志。该标志告诉Android，CriminalIntent应用中的父activity可以使用联系人数据一次。这很有用，因为不需要访

问整个联系人数据库，我们只需要访问其中的一条联系人信息。

15.4.4 检查可响应任务的 activity

本章创建的第一个隐式intent总是会以某种方式得到响应，因为就算没有可用的发送消息应用，至少还会出现一个应用选择器；但第二个就不一定了，因为有些设备上根本就没有联系人应用。如果操作系统找不到匹配的activity，应用就会崩溃。

解决办法是首先通过操作系统中的PackageManager类进行自检。在onCreateView(...)方法中实现检查，如代码清单15-14所示。

代码清单15-14 检查是否存在联系人应用 (CrimeFragment.java)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }
    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
            PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }
    return v;
}
```

Android设备上安装了哪些组件以及包括哪些activity，PackageManager类全都知道。（本书后续章节还会介绍更多组件。）调用resolveActivity(Intent, int)方法，我们可以找到匹配给定Intent任务的activity。flag标志MATCH_DEFAULT_ONLY限定只搜索带CATEGORY_DEFAULT标志的activity。这和startActivity(Intent)方法类似。

如果搜到目标，它会返回ResolveInfo告诉你找到了哪个activity。如果找不到的话，必须禁用嫌疑人按钮，否则应用就会崩溃。

如果想验证过滤器，但手头又没有不带联系人的设备，可临时添加额外的类别给intent。这个类别没有实际的作用，只是不让任何联系人应用和你的intent匹配。过滤器验证代码如代码清单15-15所示。

代码清单15-15 过滤器验证代码 (CrimeFragment.java)

```
...
final Intent pickContact = new Intent(Intent.ACTION_PICK,
    ContactsContract.Contacts.CONTENT_URI);
pickContact.addCategory(Intent.CATEGORY_HOME);
mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);
```

```
mSuspectButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        startActivityForResult(pickContact, REQUEST_CONTACT);  
    }  
});  
  
...  
}
```

现在，联系人按钮应该被禁用了，如图15-8所示。

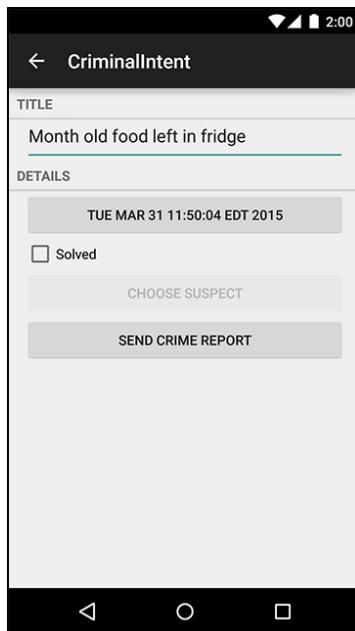


图15-8 嫌疑人选取按钮已禁用

验证完毕，记得删除相关代码，如代码清单15-16所示。

代码清单15-16 删除验证代码（CrimeFragment.java）

```
...  
  
final Intent pickContact = new Intent(Intent.ACTION_PICK,  
    ContactsContract.Contacts.CONTENT_URI);  
pickContact.addCategory(Intent.CATEGORY_HOME);  
mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);  
mSuspectButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        startActivityForResult(pickContact, REQUEST_CONTACT);  
    }  
});  
  
...
```

15.5 挑战练习：ShareCompat

第一个练习比较简单。Android支持库有个叫作ShareCompat的类，它有一个IntentBuilder. ShareCompat.IntentBuilder内部类。利用这个内部类创建用于发送消息按钮的Intent略微方便一些。

因此，你要接受的挑战就是：在mReportButton的监听器中，改用ShareCompat. IntentBuilder来创建你的Intent。

15.6 挑战练习：又一个隐式 intent

相较于发送消息，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，直接拨打陋习嫌疑人的电话。

完成这个挑战，首先需要联系人数据库中的手机号码。这需要查询ContactsContract数据库中的CommonDataKinds.Phone表。关于查询方法，建议参阅它们的参考文档。

小提示：你应该使用`android.permission.READ_CONTACTS`权限。利用这个权限，可以查询到ContactsContract.Contacts._ID。然后再使用联系人ID查询CommonDataKinds.Phone表。

搞定了电话号码，就可以使用电话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。这样的话，用户就有了冷静下来改变主意的机会。这种贴心的设计应该会受到欢迎的。

第 16 章

使用intent拍照

16

掌握了隐式intent之后，可以考虑利用它丰富crime记录信息。例如，给陋习现场拍张照片就是个不错的主意。

拍照需要用到一些包括隐式intent在内的新工具。隐式intent可以启动用户喜爱的相机应用并接收它拍摄的照片。

接收到照片后，该如何存储和展示这些照片呢？答案就在本章。

16.1 布置照片

首先要做的是在用户界面上布置照片。这需要新增两个View对象：显示照片缩略图的ImageView和拍照按钮。完成后的用户界面如图16-1所示。

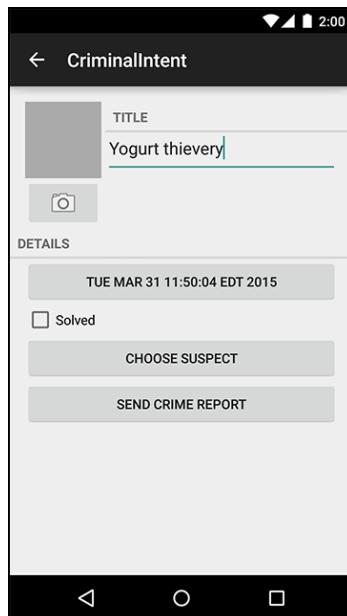


图16-1 重新布置的用户界面

在同一行放置照片缩略图和拍照按钮的话，应用界面就会显得拥挤，给人不专业的感觉。下面来学习如何进行合理的布置。

引入布局文件

为放置新组件，我们要在fragment_crime.xml布局（水平和竖直模式）上开辟一块较大的区域。同时修改res/layout/fragment_crime.xml和res/layout-land/fragment_crime.xml布局文件是种可行的方案。但这并不是唯一的选择，还可采用其他方式：使用include引入布局文件。

使用include可在在一个布局中引入另外的布局。要采用这种方式，首先要创建新开辟视图区域的公共布局文件，如图16-2所示。

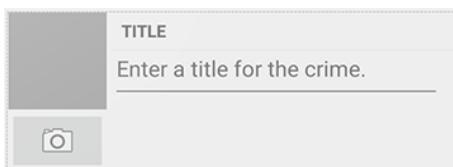


图16-2 相机和标题栏区域

新建这样的布局文件并取名为view_camera_and_title.xml。参照图16-3，先定义左边区域。

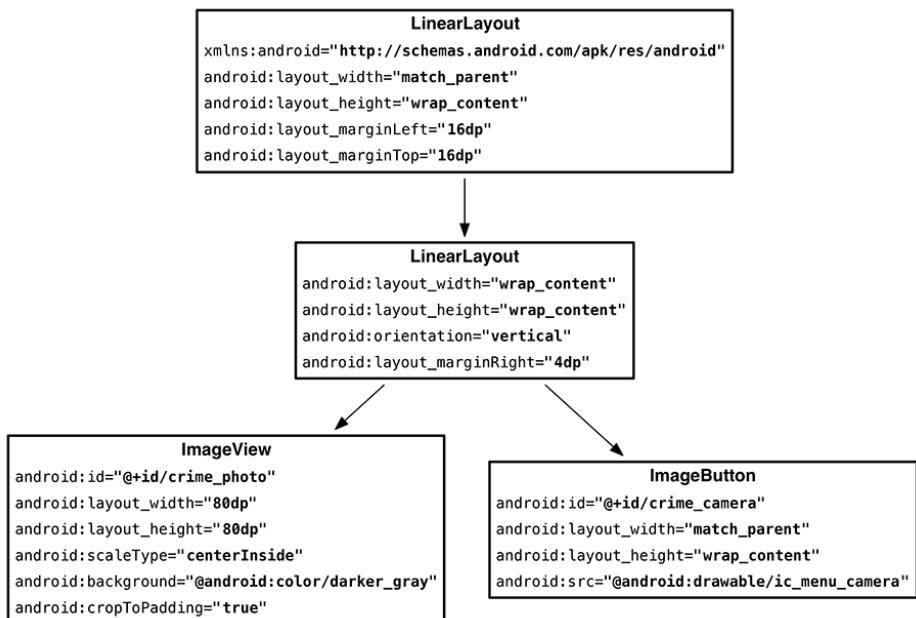


图16-3 相机视图布局（res/layout/view_camera_and_title.xml）

然后，参照图16-4定义右边区域。

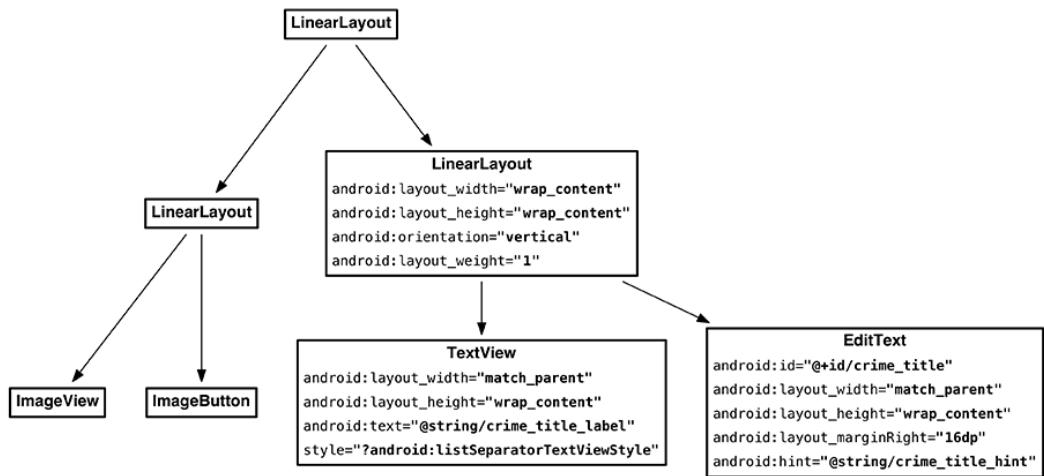


图16-4 标题视图布局 (res/layout/view_camera_and_title.xml)

使用Android Studio的设计视图确认完成后的布局和图16-2一样。

现在可以使用include标签引入我们刚创建的公共布局了。注意，使用include标签时，layout属性不应再使用android前缀了。

首先修改主布局文件res/layout/fragment_crime.xml，如图16-5所示。

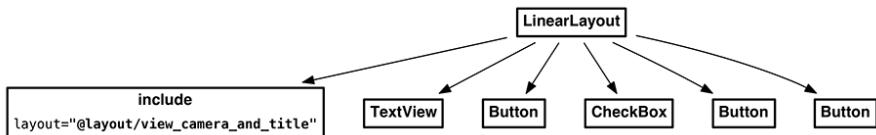


图16-5 引入公共布局（竖直模式）

然后修改水平模式的布局，如图16-6所示。

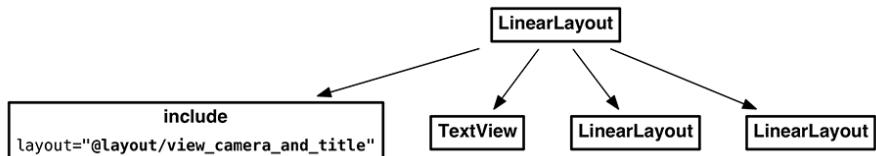


图16-6 引入公共布局（水平模式）

运行CriminalIntent应用，应该可以看到如图16-1所示的应用界面。

漂亮的用户界面完成了，但要响应ImageButton按钮点击和控制ImageView视图的内容展示，我们还要添加引用它们的实例变量。和从fragment_crime.xml布局中寻找视图一样，我们也调用findViewById(int)方法从view_camera_and_title.xml布局中找到相应视图。

代码清单16-1 添加实例变量（CrimeFragment.java）

```

...
private CheckBox mSolvedCheckbox;
private Button mSuspectButton;
private ImageButton mPhotoButton;
private ImageView mPhotoView;

...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    ...

    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
                                         PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}

...

```

用户界面相关的工作完成了。接下来的任务是编码实现拍照和显示照片功能。

16.2 外部存储

相机照片动辄几Mb大小，直接保存在数据库中肯定不现实。很自然，大家会想到直接使用设备的文件系统。

一般来讲，应用都应该使用私有存储空间保存各类文件。还记得吗？在前面章节中，我们在私有存储空间保存过SQLite数据文件。使用类似Context.getFileStreamPath(String)和Context.getFilesDir()这样的方法，我们也可以实现这样的存储目标，如表16-1所示。（结果就是照片文件保存在databases子目录相邻的某个子目录中。）

表16-1 Context类提供的基本文件和目录处理方法

方 法	使 用 目 的
File getFilesDir()	获取/data/data/<packagename>/files目录
FileInputStream openFileInput(String name)	打开现有文件进行读取
FileOutputStream openFileOutput(String name, int mode)	打开文件进行写入，如不存在就创建它
File getDir(String name, int mode)	获取/data/data/<packagename>/目录的子目录（如不存在就先创建它）

(续)

方 法	使 用 目 的
<code>String[] fileList()</code>	获取 <code>/data/data/<packagename>/files</code> 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 <code>/data/data/<packagename>/cache</code> 目录。应注意及时清理该目录，并节约使用空间

如果想存储的文件仅供应用内部使用，使用上表中的各类方法就可以了。

如果要允许其他应用读写你的文件，事情就没那么简单了：虽然有个`Context.MODE_WORLD_READABLE`可以传入`openFileOutput(String, int)`方法，但这个flag已经遭废弃了。即使强制使用，这种使用方式在新系统设备上也不是那么可靠。因而，想共享文件给其他应用或是接收其他应用的文件（如相机应用拍摄的照片）时，路只有一条：使用外部存储保存文件。

外部存储有两类：主外部存储和其他各类存储介质。所有的Android设备至少应有一个主外部存储地。使用`Environment.getExternalStorageDirectory()`可以返回这个外部存储目录。以前，这个存储地通常是指SD卡，但现在都已基本整合至了设备内部。即使现在还有设备使用扩展外部存储，也应算作其他各类存储介质这一类了。

`Context`也提供了一些访问外部存储空间要用到的方法，如表16-2所示。使用这些方法，我们可以方便地处理外部存储空间里文件和目录的读写。不过，这些方法存储的文件属公共文件，请谨慎使用。

表16-2 Context类提供的外部文件和目录处理方法

方 法	使 用 目 的
<code>File getExternalCacheDir()</code>	获取主外部存储上的缓存文件目录。用法类似 <code>getCacheDir()</code> 方法，但要注意，Android一般不会自动清理该目录
<code>File[] getExternalCacheDirs()</code>	获取多个外部存储上的缓存文件目录
<code>File getExternalFilesDir(String)</code>	获取主外部存储上存放常规文件的文件目录。通过 <code>String</code> 参数，可访问特定内容类型的子目录。内容类型常量以 <code>DIRECTORY_</code> 为前缀，定义在 <code>Environment</code> 中。例如，用于图像文件的 <code>Environment.DIRECTORY_PICTURES</code>
<code>File[] getExternalFilesDirs(String)</code>	类似 <code>getExternalFilesDir(String)</code> 方法，但该方法可获取指定类型的所有文件目录
<code>File[] getExternalMediaDirs()</code>	获取Android存储图片、视频和音乐文件的所有外部文件目录。和 <code>getExternalFilesDir(Environment.DIRECTORY_PICTURES)</code> 方法区别在于，调用该方法，多媒体扫描器会自动扫描目标目录，并将存放的多媒体文件暴露给能够播放音乐、浏览视频和图片的应用。也就是说， <code>getExternalMediaDirs()</code> 方法返回目录中存放的任何文件都会自动出现在多媒体应用中

从技术上讲，既然有些设备的外部存储使用的是可插拔的SD卡，上述外部存储目录很可能无法使用。不过，这种情况实际还是比较少见，因为几乎所有的现代设备都内置了供外部存储用

的不可插拔存储介质。

指定照片存放位置

现在要处理的是指定照片存放位置。首先，在Crime.java中添加获取文件名的方法，如代码清单16-2所示。

代码清单16-2 添加文件名获取方法（Crime.java）

```
...
public void setSuspect(String suspect) {
    mSuspect = suspect;
}

public String getPhotoFilename() {
    return "IMG_" + getId().toString() + ".jpg";
}
}
```

Crime.getPhotoFilename()方法不知道图片文件该存储在哪个目录。不过，既然文件名基于Crime ID编制，它也具有唯一性。

接下来，找到要保存文件的目录。CrimeLab负责CriminalIntent应用的数据持久工作。既然CrimeLab负责CriminalIntent应用的数据持久工作，那么在CrimeLab类里添加getPhotoFile(Crime)方法也就再合适不过了，如代码清单16-3所示。

代码清单16-3 定位图片文件（CrimeLab.java）

```
public class CrimeLab {
    ...

    public Crime getCrime(UUID id) {
        ...
    }

    public File getPhotoFile(Crime crime) {
        File externalFilesDir = mContext
            .getExternalFilesDir(Environment.DIRECTORY_PICTURES);

        if (externalFilesDir == null) {
            return null;
        }

        return new File(externalFilesDir, crime.getPhotoFilename());
    }

    ...
}
```

上述新增方法不会创建任何文件。它的作用就是返回指向某个具体位置的File对象。我们在这个方法里加了个校验：确认外部存储是否可用。如果不可用，getExternalFilesDir(String)方法会返回null值。

16.3 使用相机 intent

现在可以实现拍照功能了。这并不难，只要使用一个隐式intent就可以了。

首先是保存图片文件存储位置，如代码清单16-4所示。（接下来好几个地方会用到它，做好这步会省事不少。）

代码清单16-4 获取图片文件位置（CrimeFragment.java）

```
...
private Crime mCrime;
private File mPhotoFile;
private EditText mTitleField;
...

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    mPhotoFile = CrimeLab.get(getActivity()).getPhotoFile(mCrime);
}

...

```

然后是处理相机拍照按钮，实现点击触发拍照。相机intent定义在MediaStore里。这个类负责处理所有多媒体相关的任务。发送一个带MediaStore.ACTION_IMAGE_CAPTURE操作的intent，Android会启动相机activity拍照。

实现拍照功能的思路已经理清了，但还有些小细节不能忘了处理。

16.3.1 外部存储使用权限

读写外部存储需要获得权限。在manifest文件中，权限通常以<uses-permission>标签字符串值的形式存在。这些权限告诉Android：应用想要做的事需要获得Android授权。

为强化安全控管的责任，Android会要求应用申请各类权限。假设应用想使用外部存储，它就要向Android提出授权请求。于是，在应用安装时，Android就会通知用户该应用希望获得这样的授权。既然用户已经知情并给予放行，在SD卡上保存文件自然也就不足为奇了。

不过，Android新版系统放宽了权限管控的限制。既然Context.getExternalFilesDir(String)方法会返回应用专用的文件目录，那么可以直接读写这个目录应该是理所当然的事。因此，对Android 4.4（API 19）及其之后的新版系统来说，应用就不用申请使用这个目录的权限了。（使用其他外部存储仍然需要。）

在manifest文件中，添加读写外部存储的授权请求（适用的API级别最大为18级），如代码清单16-5所示。

代码清单16-5 申请外部存储读写权限 (AndroidManifest.xml)

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.criminalintent" >

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18"
    />
    ...

```

在上述代码中，`maxSdkVersion`属性规定：只有在API级别小于19的Android设备上，应用才需要这样的权限申请。

注意，我们只申请了外部存储的读权限。虽然还有`WRITE_EXTERNAL_STORAGE`写权限，但这不用我们操心：相机应用会替我们搞定它。

16.3.2 触发拍照

准备工作都已完成，可以使用相机intent了。对于intent的操作，我们需要定义在`MediaStore`类中的`ACTION_CAPTURE_IMAGE`。`MediaStore`类定义了一些公共接口，可用于处理图像、视频以及音乐这些常见的多媒体任务。当然，这也包括触发相机应用的拍照intent。

`ACTION_IMAGE_CAPTURE`打开相机应用，默认只能拍摄缩略图这样的低分辨率照片，而且照片会保存在`onActivityResult(...)`返回的Intent对象里。

要想获得全尺寸照片，就要让它使用文件系统存储照片。这可以通过传入保存在`MediaStore.EXTRA_OUTPUT`中的指向存储路径的Uri来完成。

编写用于拍照的隐式intent，如代码清单16-6所示。拍摄的照片应该保存在`mPhotoFile`指定的地方。同时，别忘了检查设备上是否安装有相机应用，以及是否有地方存储照片。（要确认是否有可用的相机应用，可以查询`PackageManager`查看是否有响应相机隐式intent的activity。关于查询`PackageManager`的详细内容，参见15.4.4节。）

代码清单16-6 使用相机intent (CrimeFragment.java)

```

...
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
private static final int REQUEST_PHOTO= 2;

...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    final Intent captureImage = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

```

```
boolean canTakePhoto = mPhotoFile != null &&
    captureImage.resolveActivity(packageManager) != null;
mPhotoButton.setEnabled(canTakePhoto);

if (canTakePhoto) {
    Uri uri = Uri.fromFile(mPhotoFile);
    captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);
}

mPhotoButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivityForResult(captureImage, REQUEST_PHOTO);
    }
});

mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

return v;
}
```

运行CriminalIntent应用，点击相机按钮启动相机应用，如图16-7所示。



图16-7 打开相机应用

16.4 缩放和显示位图

使用相机intent，我们就可以成功拍摄陋习现场照片并保存了。

有了照片，接下来就是找到并加载它，然后展示给用户看。在技术实现上，这需要加载照片到大小合适的Bitmap对象中。要从文件生成Bitmap对象，我们需要BitmapFactory类：

```
Bitmap bitmap = BitmapFactory.decodeFile(mPhotoFile.getPath());
```

看到这里，有没有感觉不对劲？肯定有的。否则依照本书代码风格，上述代码就会直接加粗印刷，你对照输入就行了。

不卖关子了，问题在于：介绍Bitmap时，我们提到“大小合适”。Bitmap是个简单对象，它只存储实际像素数据。也就是说，即使原始照片已压缩过，但存入Bitmap对象时，文件并不会同样压缩。因此，16万像素24位已压缩为5Mb大小的JPG照片文件，一旦载入Bitmap对象，就会立即膨胀至48Mb大小！

这个问题可以设法解决，但需要手工缩放位图照片。具体做法就是，首先确认文件到底有多大，然后考虑按照给定区域大小合理缩放文件。最后，重新读取缩放后的文件，创建Bitmap对象。

创建名为PictureUtils.java的新类，并在其中添加getScaledBitmap(String, int, int)缩放方法，如代码清单16-7所示。

代码清单16-7 创建getScaledBitmap(...)方法 (PictureUtils.java)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, int destWidth, int destHeight) {
        // 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;

        // Figure out how much to scale down by
        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;

        // Read in and create final bitmap
        return BitmapFactory.decodeFile(path, options);
    }
}
```

上述方法中，inSampleSize值很关键。它决定着缩略图像素的大小。假设这个值是1的话，

就表明缩略图和原始照片的水平像素大小一样。如果是2的话，它们的水平像素比就是1：2。因此，`inSampleSize`值为2时，缩略图的像素数就是原始文件的四分之一。

问题总是接踵而来。解决了缩放问题，又冒出了新问题：`PhotoView`究竟有多大无人知道。`onCreate(...)`、`onStart()`和`onResume()`方法启动后，才会有首个实例化布局出现。也就在此时，显示在屏幕上的视图才会有大小尺寸。这也是出现新问题的原因。

解决方案有两个：要么等布局实例化完成并显示，要么干脆使用保守估算值。特定条件下，尽管估算比较主观，但确实是唯一切实可行的办法。再添加一个`getScaledBitmap(String, Activity)`静态`Bitmap`估算方法，如代码清单16-8所示。

代码清单16-8 编写合理的缩放方法（PictureUtils.java）

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, Activity activity) {
        Point size = new Point();
        activity.getWindowManager().getDefaultDisplay()
            .getSize(size);

        return getScaledBitmap(path, size.x, size.y);
    }

    ...
}
```

该方法先确认屏幕的尺寸，然后按此缩放图像。这样，就能保证载入的`ImageView`永远不会过大。看到没有，无论如何，这是一个比较保守的估算有时就是能解决问题。

接下来，为把`Bitmap`载入`ImageView`。在`CrimeFragment.java`中，添加刷新`mPhotoView`的方法，如代码清单16-9所示。

代码清单16-9 更新mPhotoView（CrimeFragment.java）

```
...
private String getCrimeReport() {
    ...

    private void updatePhotoView() {
        if (mPhotoFile == null || !mPhotoFile.exists()) {
            mPhotoView.setImageDrawable(null);
        } else {
            Bitmap bitmap = PictureUtils.getScaledBitmap(
                mPhotoFile.getPath(), getActivity());
            mPhotoView.setImageBitmap(bitmap);
        }
    }
}
```

然后，分别在`onCreateView(...)`和`onActivityResult(...)`方法中调用`updatePhotoView()`方法，如代码清单16-10所示。

代码清单16-10 调用updatePhotoView()方法 (CrimeFragment.java)

```

mPhotoButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivityForResult(captureImage, REQUEST_PHOTO);
    }
});

mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);
updatePhotoView();

return v;
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_CONTACT && data != null) {
        ...

    } else if (requestCode == REQUEST_PHOTO) {
        updatePhotoView();
    }
}
}

```

再次运行应用，应该可以看到已拍照片的缩略图了。

16.5 功能声明

应用的拍照功能用起来不错，但还有件事情要做：告诉目标用户应用具有拍照功能。

假如应用要用到诸如相机、NFC，或者任何其他的随设备走的功能时，都应该要让Android系统知道。这样，假如设备缺少这样的功能，类似Google Play商店的安装程序就会拒绝安装应用。

为声明需要使用相机，在AndroidManifest.xml中加入<uses-feature>标签，如代码清单16-11所示。

代码清单16-11 添加<uses-feature>标签 (AndroidManifest.xml)

```

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

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18"

```

```

    />
<uses-feature android:name="android.hardware.camera"
              android:required="false"
/>
...

```

注意，我们在代码中使用了`android:required`属性。默认情况下，声明要使用某个设备功能后，应用就无法支持那些无此功能的设备了，但这不适用于CriminalIntent应用。这是因为，`resolveActivity(...)`方法可以判断设备是否支持拍照。如不支持，就直接禁用拍照按钮。

这里，设置了`android:required="false"`属性。Android系统就知道，尽管不带相机的设备会导致拍照功能缺失，但应用仍然可以正常安装和使用。

16.6 深入学习：使用include标签

本章，我们使用`include`标签在水平和竖直方向布局中引入了公共布局文件。这至少在两个方面给开发带来了便利：减少代码输入量以及促进代码复用。然而，这并不是说，只要水平和竖直布局有公共布局组件，就一定要用`include`标签。

既然这样，那问题来了：什么时候该用？什么时候要避免呢？回答以上问题前，首先谈谈`include`标签的工作原理。前面使用`include`标签时，我们没有添加任何其他`android`属性，因而引入视图会带入它在原始布局文件中的所有属性。

当然，我们也可以添加其他额外属性。这样，这些属性会直接添加给根视图并覆盖其原始属性值。也就是说，如果想修改`layout_width`属性值，真的可以做到。

另外，虽然本章使用`include`标签确实省时省力，但还是要提醒大家注意：`include`标签还称不上完美工具。

CriminalIntent应用视图上的一些按钮组件还是重复使用了。为什么不利用`include`标签进行复用呢？答案很简单：不推荐这么用。

经验表明，布局文件的优点是可靠又好用。例如，直接查看布局文件内容，就可以快速准确地知道应用视图是如何构建的。然而，一旦用了`include`标签，一切就不好说了。还想明白视图构成的话，就得仔细翻看布局主文件以及所有`include`的布局文件。这种非直观的感觉，极易让人失去耐心。

用户界面是应用改动相对频繁的部分。既然这样，不顾一切地追求复用原则很可能会适得其反。因此，在视图层开发时，我们一定要多多考量，尽量做到审慎、合理地使用`include`标签。

16.7 挑战练习：优化照片显示

现在虽然能够看到拍摄的照片，但没法看到它们的细节。

请创建能显示缩放版本照片的`DialogFragment`。只要点击缩略图，就会弹出这个`DialogFragment`，让用户查看缩放版本的陋习现场图片。

16.8 挑战练习：优化缩略图加载

本章，我们只能大致估算缩略图的目标尺寸。虽说可行且实施迅速，但还不够理想。

Android有个现成的API工具可用，叫作**ViewTreeObserver**。利用这个对象，我们可以从**Activity**层级结构中获取任何视图：

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

我们可以为**ViewTreeObserver**对象设置包括**OnGlobalLayoutListener**在内的各种监听器。使用**OnGlobalLayoutListener**监听器，可以监听任何布局的传递，控制事件的发生。

调整代码，使用有效的**mPhotoView**尺寸，等到有布局切换时再调用**updatePhotoView()**方法。

17

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17-1展示了这样的列表明细界面，也可称其为主从用户界面（master-detail interface）。

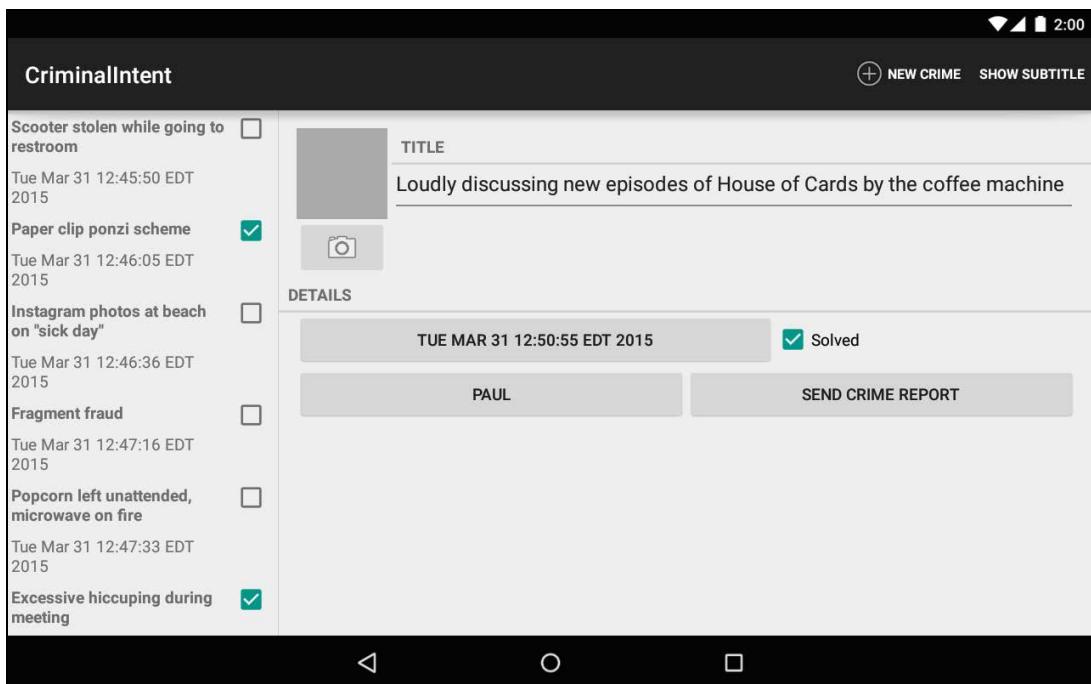


图17-1 同时显示列表和明细的用户界面

本章的代码验证需要平板设备或AVD。要创建平板AVD，首先选择Tools→Android→Android Virtual Device Manager菜单项，然后点击Create Virtual Device...按钮，在弹出的界面选择Tablet类别。选择目标虚拟硬件配置后，点击Next按钮继续，如图17-2所示。最后，确认API级别至少为21，点击Finish按钮完成。

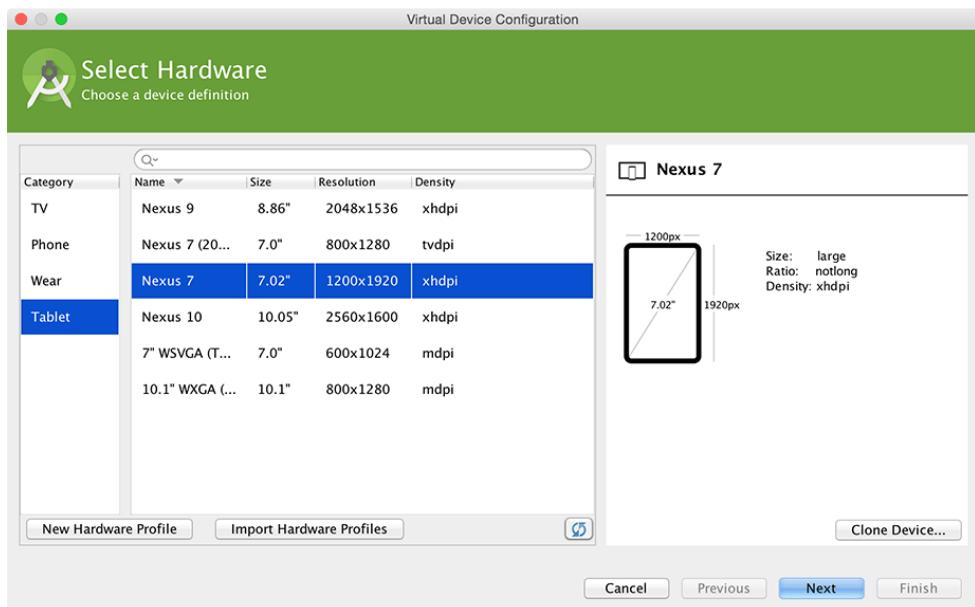


图17-2 AVD平板设备选择

17.1 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。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17-3所示。

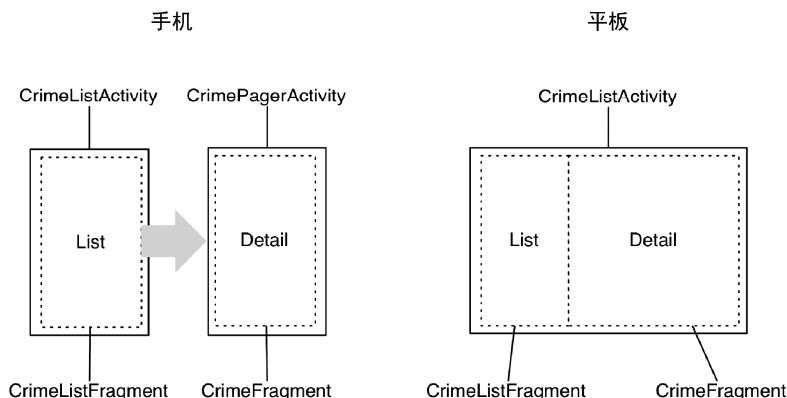


图17-3 不同类型的布局

要实现双版面布局，需完成如下任务：

- 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，在平板设备上实例化双版面布局。

17.1.1 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类更加灵活易用，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17-1所示。

代码清单17-1 增加SingleFragmentActivity类的灵活性（SingleFragmentActivity.java）

```
public abstract class SingleFragmentActivity extends AppCompatActivity {
    protected abstract Fragment createFragment();

    @LayoutRes
    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.fragment_container);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

现在，虽然SingleFragmentActivity抽象类的功能和以前一样，但是，如果不想再使用固定不变的activity_fragment.xml布局，它的子类可以选择覆盖getLayoutResId()方法返回所需布局。getLayoutResId()方法使用了@LayoutRes注解。这是告诉Android Studio，任何时候，该实现方法都应该返回有效的布局资源ID。

17.1.2 创建包含两个 fragment 容器的布局

在项目工具窗口中，右键单击res/layout/目录，新建一个XML文件。在弹出的新建XML文件界面，指定资源类型为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按钮完成。

参照图17-4，完成双版面布局的XML内容定义。

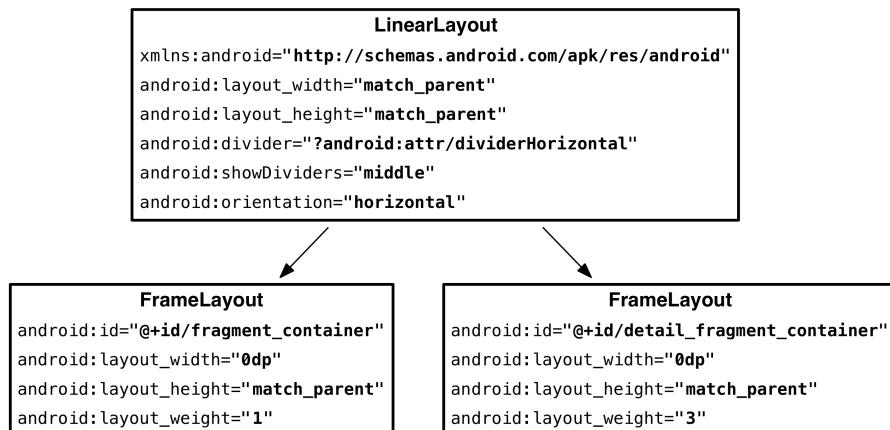


图17-4 包含两个fragment容器的布局 (layout/activity_twopane.xml)

注意，布局定义的第一个FrameLayout也有一个fragment_container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17-2所示。

代码清单17-2 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17-5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crime明细信息。本章稍后

会完成crime明细fragment容器的编码及设置工作。

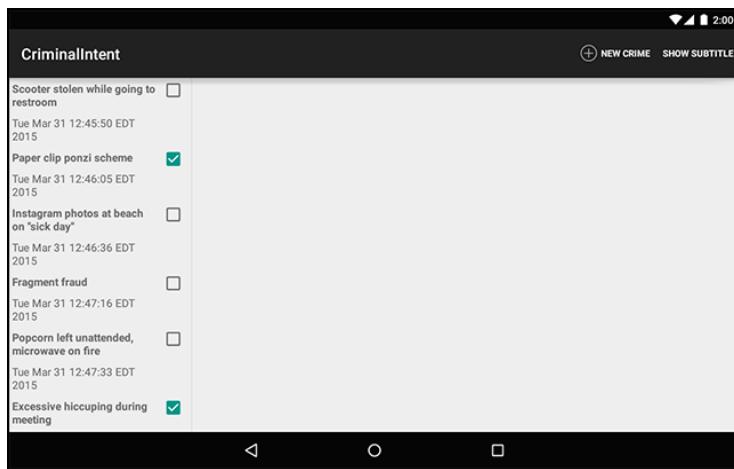


图17-5 平板设备上的双版面布局

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。

17.1.3 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。

接下来的任务就是让`CrimeListActivity`基于不同的设备使用不同的布局文件。这实际类似于前面章节对水平和竖直布局的选择和控制：使用资源修饰符。

让`res/layout/`目录中的文件使用资源修饰符虽然可行，但也有缺点。最明显的缺点就是数据冗余，因为每个布局文件都要复制一份。例如，如果想使用`activity_masterdetail.xml`布局文件，就需要将`activity_fragment.xml`复制到`res/layout/activity_masterdetail.xml`中，将`activity_twopane.xml`复制到`res/layout-sw600dp/activity_masterdetail.xml`中（`sw600dp`的作用稍后会提到）。

要解决上述问题，我们可以使用别名资源。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。

在项目工具窗口中，右键单击`res/values/`目录，新建`values`资源文件。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`。确认新建文件不带任何修饰符，最后单击`Finish`按钮。参照代码清单17-3，在新建的`refs.xml`中添加`item`节点定义。

代码清单17-3 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属性决定资源ID属于什么内部类。即使别名资源自身在res/values/目录中，它的资源ID依然属于R.layout内部类。

修改CrimeListActivity类，以R.layout.activity_masterdetail资源ID替换R.layout.activity_fragment，如代码清单17-4所示。

代码清单17-4 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应用验证别名资源的使用。一切正常的话，CrimeListActivity应该再次生成了单版面布局。

17.1.4 创建平板设备专用可选资源

因为res/values/目录中的别名资源是系统默认的别名资源，所以CrimeListActivity生成了单版面布局。

现在，创建一个大屏幕设备用可选别名资源，让activity_masterdetail别名资源指向activity_twopane.xml双版面布局资源。

在项目工具窗口中，右键单击res/values/目录，弹出如图17-6所示的新建资源文件窗口。资源文件名和目录名依然是refs.xml和values，但这次要用>>按钮把Available qualifiers窗口中的Smallest Screen Width选到右边窗口去。

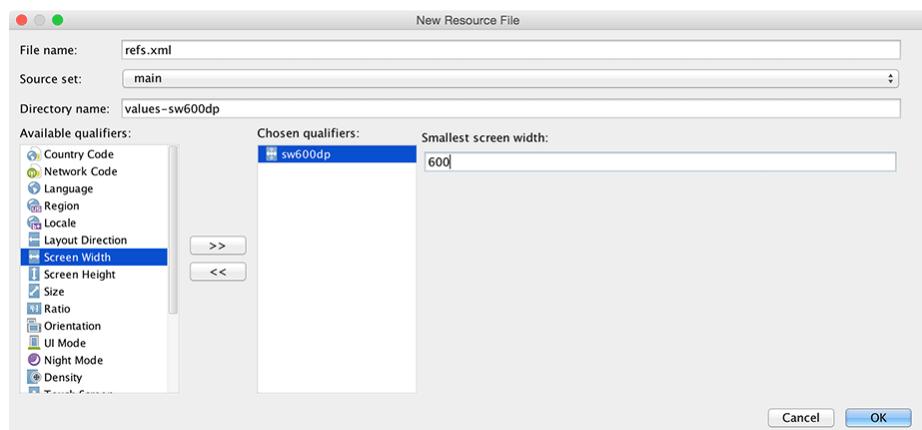


图17-6 添加资源修饰符

可以看到，资源修饰符处要求指定Smallest Screen Width的数值。输入600后点击OK按钮完成。在随后打开的新建资源文件中，添加activity_masterdetail别名资源指向activity_twopane.xml，如代码清单17-5所示。

代码清单17-5 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_twopane</item>
</resources>
```

对于上述新增别名资源，我们的目标是：

- 对于小于指定尺寸的设备，使用activity_fragment.xml资源文件；
- 对于大于指定尺寸的设备，使用activity_twopane.xml资源文件。

Android只提供一部分的资源适配机制。配置修饰符-sw600dp的作用是：如果设备尺寸大于某个指定值，就使用对应的资源文件。**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。

那另一部分的资源适配怎么处理呢？对于希望使用activity_fragment.xml的小尺寸设备要怎么做呢？这好办，Android是这样判断的：既然设备尺寸小于-sw600dp修饰符指定值，那就使用默认的activity_fragment.xml资源文件。

分别在手机和平板上运行CriminalIntent应用，确认单双版面布局的使用没有问题。

17.2 Activity: fragment 的托管者

处理完单双版面布局的显示，就可以着手添加CrimeFragment给crime明细fragment容器，让CrimeListActivity展示一个完整的双版面用户界面。

你可能会认为，只需再为平板设备实现一个CrimeHolder.onClick(View)监听器方法就行了。这样，无需启动新的CrimePagerActivity，onClick(View)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（CrimeListFragment.CrimeHolder）：

```
public void onClick(View v) {
    // Stick a new CrimeFragment in the activity's layout
    Fragment fragment = CrimeFragment.newInstance(mCrime.getId());
    FragmentManager fm = getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detail_fragment_container, fragment)
        .commit();
}
```

虽然行得通，但做法很老套。fragment天生是个独立的开发构件。如果要开发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。结果，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，并且认为CrimeListActivity的布局里包含有detail_fragment_container。但实际上，CrimeListFragment根本就不应关心这些，这都是其托管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

要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都必须实现它。

有了回调接口，就不用关心谁是托管者，fragment可以直接调用托管activity的方法。

1. 实现CrimeListFragment.Callbacks回调接口

为了实现Callbacks接口，首先要定义一个成员变量，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。

activity赋值是在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，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或指望它继续存在了。

在CrimeListFragment.java中，添加Callbacks接口。另外添加一个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17-6所示。

代码清单17-6 添加回调接口（CrimeListFragment.java）

```
public class CrimeListFragment extends Fragment {
    ...
    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 onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}

...

@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}

```

现在，`CrimeListFragment`有办法调用托管activity方法了。另外，它也不关心托管activity是谁。只要托管activity实现了`CrimeListFragment.Callbacks`接口，`CrimeListFragment`中的一切代码行为就可以都保持不变。

注意，未经类安全性检查，`CrimeListFragment`就将托管activity强制转换为了`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17-7所示。暂时不用理会`onCrimeSelected(Crime)`空方法，稍后再来处理。

代码清单17-7 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;
    }

    @Override

```

```

public void onCrimeSelected(Crime crime) {
}
}

```

最终，在用户创建新crime时，CrimeListFragment将在CrimeHolder.onClick(...)方法里调用onCrimeSelected(Crime)方法。现在，先思考如何实现CrimeListActivity.onCrimeSelected(Crime)方法。

onCrimeSelected(Crime)方法被调用时，CrimeListActivity需要完成以下任务其中之一：

- 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detail_fragment_container中。

是实例化手机还是平板界面布局，可以检查布局ID；但是推荐检查布局是否包含detail_fragment_container。这是因为布局文件名随时会变，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detail_fragment_container。

如果包含，那就创建一个fragment事务，将我们需要的CrimeFragment添加到detail_fragment_container中。如果之前就有CrimeFragment，应首先移除它。

在CrimeListActivity.java中，实现onCrimeSelected(Crime)方法，按照不同的布局界面分别处理，如代码清单17-8所示。

代码清单17-8 有条件的CrimeFragment启动 (CrimeListActivity.java)

```

@Override
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detail_fragment_container) == null) {
        Intent intent = CrimePagerActivity.newIntent(this, crime.getId());
        startActivity(intent);
    } else {
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());

        getSupportFragmentManager().beginTransaction()
            .replace(R.id.detail_fragment_container, newDetail)
            .commit();
    }
}

```

最后，在CrimeListFragment类中，在启动新的CrimePagerActivity处调用onCrimeSelected(Crime)方法。

在CrimeListFragment.java中，修改CrimeHolder.onClick(View)和onOptionsItemSelected(MenuItem)方法以调用Callbacks.onCrimeSelected(Crime)方法，如代码清单17-9所示。

代码清单17-9 调用全部回调方法 (CrimeListFragment.java)

```

@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 intent = CrimePagerActivity
                .newIntent(getActivity(), crime.getId());
        startActivity(intent);
        updateUI();
        mCallbacks.onCrimeSelected(crime);
        return true;
    case R.id.menu_item_show_subtitle:
        mSubtitleVisible = !mSubtitleVisible;
        getActivity().invalidateOptionsMenu();
        updateSubtitle();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}

...
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {

    ...
    @Override
    public void onClick(View v) {
        Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
        mCallbacks.onCrimeSelected(mCrime);
    }
}

```

在onOptionsItemSelected(...)方法中调用回调方法时，只要新增crime记录，crime列表就会立即重新加载。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。添加一条crime记录，可以看到，`detail_fragment_container`容器中立即显示了新的`CrimeFragment`视图。然后，尝试查看其他旧记录以观察`CrimeFragment`视图的切换，如图17-7所示。

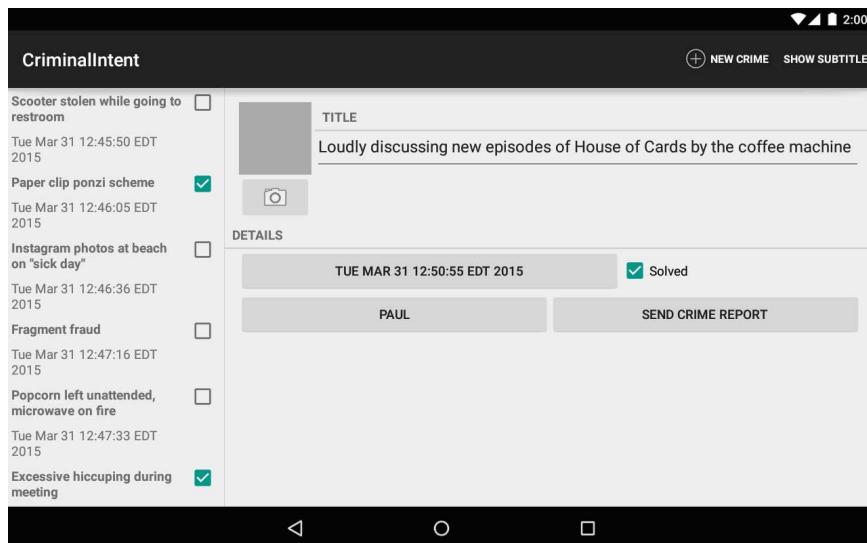


图17-7 已关联的主界面和明细界面

看上去真不错！不过，还不够完美：如果修改crime明细内容，列表项并不会显示最新内容。当前，在CrimeListFragment.onResume()方法中，只有新添加crime记录，我们才能立即刷新显示列表项界面。但是在平板设备上，CrimeListFragment和CrimeFragment同时可见。因此，当CrimeFragment出现时，CrimeListFragment不会暂停——没有暂停怎会有恢复？这就是crime列表项不能刷新的根本原因。

下面，我们在CrimeFragment中添加另一个回调接口来修正这个问题。

2. 实现CrimeFragment.Callbacks回调接口

CrimeFragment类中定义的接口如下：

```
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}
```

CrimeFragment如果要刷新数据，需要做两件事。首先，既然CriminalIntent应用的数据源是SQLite数据库，那么它需要把Crime保存到CrimeLab里。然后，CrimeFragment类会调用托管activity的onCrimeUpdated(Crime)方法。CrimeListActivity类会负责实现onCrimeUpdated(Crime)方法，从数据库获取并展示最新数据，重新加载CrimeListFragment的列表。

实现CrimeFragment的接口之前，先把CrimeListFragment.updateUI()方法修改为CrimeListActivity可以调用的公共方法，如代码清单17-10所示。

代码清单17-10 修改updateUI()方法的可见性 (CrimeListFragment.java)

```
private public void updateUI() {
    ...
}
```

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17-11所示。

代码清单17-11 新增CrimeFragment回调接口（CrimeFragment.java）

```
...
private ImageButton mPhotoButton;
private ImageView mPhotoView;
private Callbacks mCallbacks;

/**
 * Required interface for hosting activities.
 */
public interface Callbacks {
    void onCrimeUpdated(Crime crime);
}

public static CrimeFragment newInstance(UUID crimeId) {
    ...
}

@Override
public void onAttach(Activity activity) {
    super.onAttach(activity);
    mCallbacks = (Callbacks)activity;
}

@Override
public void onCreate(Bundle savedInstanceState) {
    ...
}

@Override
public void onPause() {
    ...
}

@Override
public void onDetach() {
    super.onDetach();
    mCallbacks = null;
}
```

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，如代码清单17-12所示。

代码清单17-12 刷新显示crime列表（CrimeListActivity.java）

```
public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
    ...
}
```

```

public void onCrimeUpdated(Crime crime) {
    CrimeListFragment listFragment = (CrimeListFragment)
        getSupportFragmentManager()
            .findFragmentById(R.id.fragment_container);
    listFragment.updateUI();
}
}

```

托管CrimeFragment的所有activity都必须实现CrimeFragment.Callbacks接口。因而在CrimePagerActivity中提供一个空接口实现，如代码清单17-13所示。

代码清单17-13 空接口实现 (CrimePagerActivity.java)

```

public class CrimePagerActivity extends AppCompatActivity
    implements CrimeFragment.Callbacks {
    ...
    @Override
    public void onCrimeUpdated(Crime crime) {
    }
}

```

CrimeFragment有两项重复性任务：在CrimeLab中保存mCrime，以及调用mCallbacks.onCrimeUpdated(Crime)。添加一个便利方法处理它们，如代码清单17-14所示。

代码清单17-14 新增updateCrime()方法 (CrimeFragment.java)

```

...
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
}

private void updateCrime() {
    CrimeLab.get(getActivity()).updateCrime(mCrime);
    mCallbacks.onCrimeUpdated(mCrime);
}

private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}

...

```

在CrimeFragment.java中，如果Crime对象的标题或问题处理状态有变动，触发调用updateCrime()方法，如代码清单17-15所示。

代码清单17-15 调用onCrimeUpdated(Crime)方法 (CrimeFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {

```

```

...
mTitleField.addTextChangedListener(new TextWatcher() {
    ...
    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        mCrime.setTitle(s.toString());
        updateCrime();
    }
});

...
mSolvedCheckbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        mCrime.setSolved(isChecked);
        updateCrime();
    }
});

...
}

```

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17-16所示。

代码清单17-16 再次调用onCrimeUpdated(Crime)方法 (CrimeFragment.java)

```

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...

    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        updateCrime();
        updateDate();
    } else if (requestCode == REQUEST_CONTACT && data != null) {
        ...

        try {
            ...
            String suspect = c.getString(0);
            mCrime.setSuspect(suspect);
            updateCrime();
            mSuspectButton.setText(suspect);
        } finally {
    }
}

```

```
        c.close();
    }
} else if (requestCode == REQUEST_PHOTO) {
    updateCrime();
    updatePhotoView();
}
}
```

在平板设备上运行CriminalIntent应用。确认CrimeFragment视图中发生的任何修改，RecyclerView视图都能够刷新显示，如图17-8所示。然后，在手机上运行CriminalIntent应用，确认应用运行一切正常。

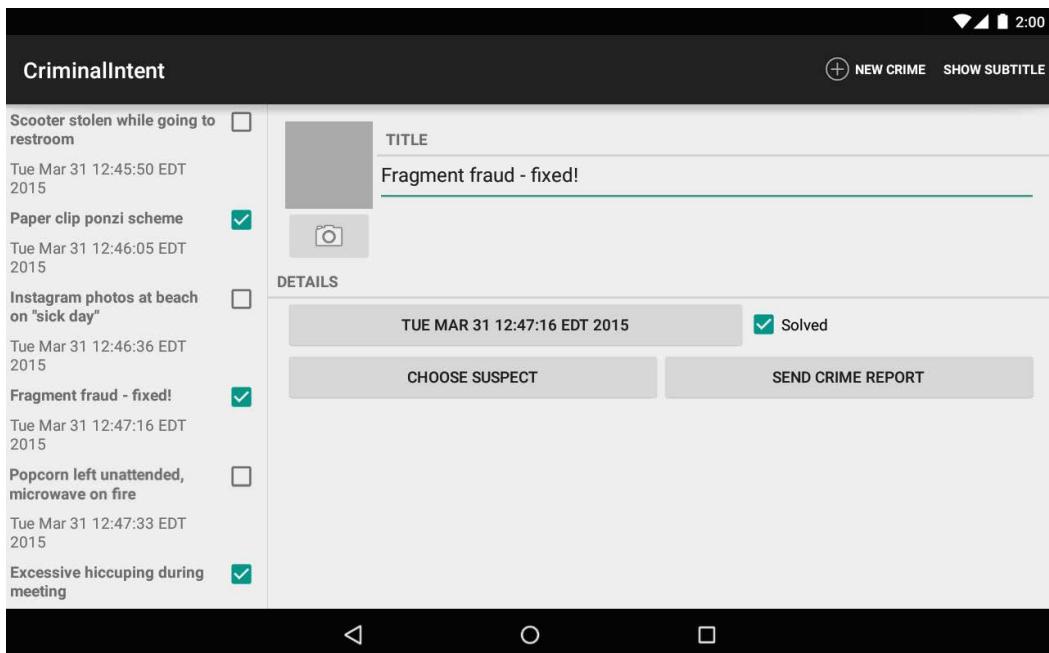


图17-8 列表刷新反映了明细界面的修改

至此，CriminalIntent应用的开发全部完成了。在这11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。

17.3 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17-1展示了每个类别修饰符的最低屏幕大小。

表17-1 屏幕大小修饰符

名称	最低屏幕大小	名称	最低屏幕大小
small	320x426dp	large	480x640dp
normal	320x470dp	xlarge	720x960dp

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17-2列出了新引入的修饰符。

表17-2 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XXX dp

假设想指定某个布局仅适用于屏幕宽度至少为300dp的设备，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）。

设备方向变换的话，设备的宽和高也会交换。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。设备的方向会变，因此sw可以是最小宽度，也可以是最小高度。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。

第 18 章

Assets

18

我们知道，Android资源系统（resources system）可以用来打包应用所需的图片、XML文件以及其他非Java资源。本章，我们要学习另一种资源打包方式：assets。

此外，还会学习开发一个叫作BeatBox的新应用（如图18-1所示）。BeatBox不是传统意义上的音乐节拍盒，而是一个能帮你打败对手的神奇盒子。不过，它不会让你做出狂舞胳膊，致人受伤的危险事。实际上，它的作用出人意料：发出一种特殊的喊声，直到对手求饶为止。



图18-1 本章要完成的BeatBox应用效果

18.1 为何使用 assets

resources资源可以存储声音文件。比如在res/raw目录保存79_long_scream.wav这样的文件，

就可以使用R.raw.79_long_scream这样的ID获取到它。将类似的声音文件存储为资源后，我们就可以按特定的方式使用它们了。例如，可以根据设备的不同方位、语言以及系统版本调用不同的声音资源。

不过BeatBox应用要用到很多不同的声效，涉及20多个不同声音文件的处理。可以想像，如果使用Android资源系统一个个去处理，效率会有多低。要是能把这些文件全放在一个目录里随应用打包就好了，可惜Android资源系统并不支持。

这正是assets大显其能的地方。assets可以被看作随应用打包的微型文件系统，支持任意层次的文件目录结构。因为这个优点，类似游戏这样需要加载大量图片和声音资源的应用通常都会使用它。

18.2 创建 BeatBox 应用

在Android Studio中，选择File → New Project...菜单项创建新项目。项目名称是BeatBox，公司域名是android.bignerdranch.com，最低SDK版本是API 16。新建一个名为BeatBoxActivity的Empty Activity，其余默认项保持不变，完成项目创建。

BeatBox应用会用到RecyclerView。因此，打开项目设置添加com.android.support:recyclerview-v7依赖项。

在界面设计方面，应用主屏幕会显示一排排可以播放声音的按钮。因此，我们需要两个布局文件，分别用于网格和按钮。

首先创建RecyclerView布局文件。我们不需要项目自动产生的res/layout/activity_beat_box.xml，因此直接将它改名为fragment_beat_box.xml，然后参照代码清单18-1完成布局定义。

代码清单18-1 创建主布局文件（res/layout/fragment_beat_box.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_beat_box_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

接下来，创建按钮布局文件res/layout/list_item_sound.xml，如代码清单18-2所示。

代码清单18-2 创建声音布局文件（res/layout/list_item_sound.xml）

```
<?xml version="1.0" encoding="utf-8"?>
<Button
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

在com.bignerdranch.android.beatbox包中，创建名为BeatBoxFragment的Fragment，如代码清单18-3所示。新建Fragment要实例化的布局就是刚才新建的两个布局文件。

代码清单18-3 创建BeatBoxFragment (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_beat_box, container, false);

        RecyclerView recyclerView = (RecyclerView)view
            .findViewById(R.id.fragment_beat_box_recycler_view);
        recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        return view;
    }
}
```

注意，与第9章对比可知，我们在上述代码中使用LayoutManager的方式有所不同。这里，LayoutManager会以网格的样式布局组件，即每行布局多个组件。数字3表示网格的每行有3列。创建一个使用list_item_sound.xml布局的ViewHolder，如代码清单18-4所示。

代码清单18-4 创建SoundHolder (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
    }

    private class SoundHolder extends RecyclerView.ViewHolder {
        private Button mButton;

        public SoundHolder(LayoutInflater inflater, ViewGroup container) {
            super(inflater.inflate(R.layout.list_item_sound, container, false));
            mButton = (Button)itemView.findViewById(R.id.list_item_sound_button);
        }
    }
}
```

接着，创建一个绑定了SoundHolder的Adapter，如代码清单18-5所示。(不要输入任何方法，直接把光标置于RecyclerView.Adapter上，然后按Option+Return(或Alt+Enter)组合键，Android

Studio会自动完成大部分代码。)

代码清单18-5 创建SoundAdapter (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ...
    private class SoundHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
        @Override
        public SoundHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            return new SoundHolder(inflater, parent);
        }

        @Override
        public void onBindViewHolder(SoundHolder soundHolder, int position) {

        }

        @Override
        public int getItemCount() {
            return 0;
        }
    }
}
```

现在，在onCreateView(...)方法中使用SoundAdapter，如代码清单18-6所示。

代码清单18-6 使用SoundAdapter (BeatBoxFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_beat_box, container, false);

    RecyclerView recyclerView = (RecyclerView)view
        .findViewById(R.id.fragment_beat_box_recycler_view);
    recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    recyclerView.setAdapter(new SoundAdapter());

    return view;
}
```

最后，在BeatBoxActivity中创建BeatBoxFragment。这里，BeatBoxActivity同样是直接继承CriminalIntent应用中用过的SingleFragmentActivity类。

首先，找到CriminalIntent项目的SingleFragmentActivity.java和activity_fragment.xml文件，将其分别复制到com.bignerdranch.android.beatbox包和app/src/res/目录。（既可以从此自己的CriminalIntent应用文件夹，也可以从随书文件中找到这些文件。关于如何获取随书文件，请参考

2.6节相关内容。)

然后，清空BeatBoxActivity类中的所有内容，改为继承SingleFragmentActivity类，并覆盖createFragment()方法，如代码清单18-7所示。

代码清单18-7 填充BeatBoxActivity (BeatBoxFragment.java)

```
public class BeatBoxActivity extends SingleFragmentActivity {  
    @Override  
    protected Fragment createFragment() {  
        return BeatBoxFragment.newInstance();  
    }  
}
```

BeatBox应用的基本框架搭建完毕，可以测试运行了。不过，BeatBoxFragment目前什么也显示不了，如图18-2所示。

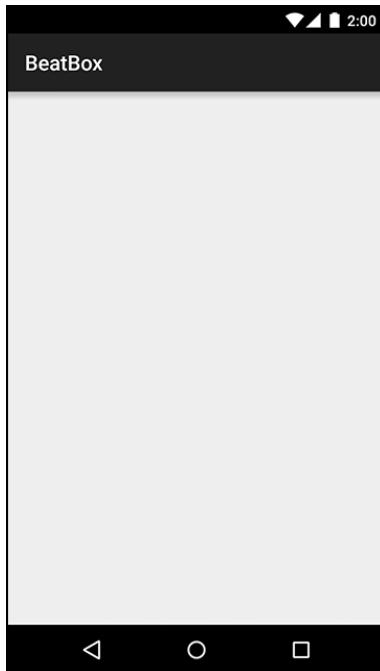


图18-2 BeatBox空空如也

18.3 导入 assets

现在我们来导入assets。首先是创建assets目录。右键单击app模块，选择New→Folder→Assets Folder菜单项，弹出如图18-3所示画面。清除Change Folder Location选项，保持Target Source Set的main选项不变，单击Finish按钮完成。

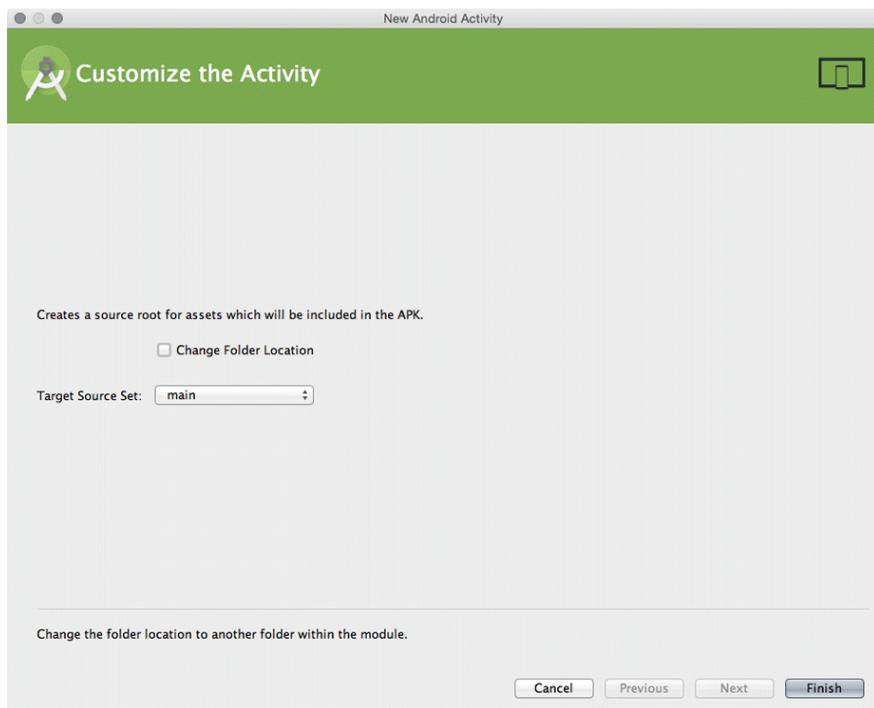


图18-3 创建assets目录

接着，右键单击assets目录，选择New → Directory菜单项，为声音资源创建sample_sounds子目录，如图18-4所示。

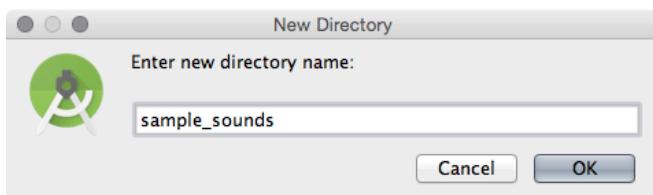


图18-4 创建sample_sounds子目录

assets目录中的所有文件都会随应用打包。为方便组织文件，我们创建了sample_sounds子目录。与资源不同，这并不是我们一定要做的。

要用到的声音资源文件在哪里找呢？在网站 www.freesound.org，我们会找到 [www.freesound.org](http://www.freesound.org/people/plagasul/packs/3/) 网站用户 plagasul 基于创作共用许可发布的一套声音文件（网址为 www.freesound.org/people/plagasul/packs/3/）。请访问以下地址获取这套文件的压缩包：

https://www.bignerdranch.com/solutions/sample_sounds.zip

下载并解压缩文件至assets/sample_sounds目录，如图18-5所示。

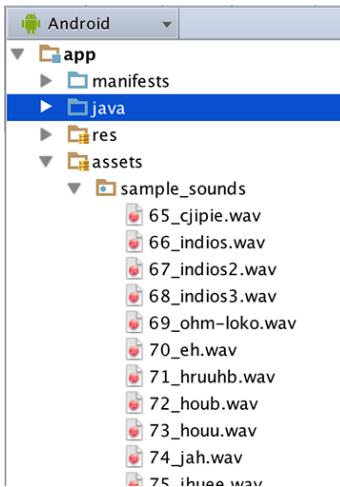


图18-5 已导入的assets

(确保这里只有.wav文件，而非尚未解压的.zip文件。)

重新编译应用，确保一切正常。接下来编写代码，列出所有这些资源并展示给用户。

18.4 处理 assets

assets导入后，我们还要能在应用中进行定位、管理记录以及播放。这需要新建一个名为BeatBox的资源管理类。如代码清单18-8所示，在com.bignerdranch.android.beatbox包中创建这个类，并添加两个常量：一个用于日志记录，另一个用于存储声音资源文件目录名。

代码清单18-8 创建BeatBox类（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
}
```

访问assets需要用到AssetManager类。我们可以从Context中获取它。既然BeatBox需要，不妨添加一个带Context参数的构造函数获取并留存它，如代码清单18-9所示。

代码清单18-9 获取AssetManager备用（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";

    private AssetManager mAssets;

    public BeatBox(Context context) {
```

```

        mAssets = context.getAssets();
    }
}

```

通常，在访问assets时，可以不用关心究竟使用哪个Context对象。而且，在实际开发的任何场景下，所有Context中的AssetManager管理的都是同一套assets资源。

要取得assets中的资源清单，可以使用list(String)方法。如代码清单18-10所示，实现一个loadSounds()方法，调用它给出声音文件清单。

代码清单18-10 查看assets资源（BeatBox.java）

```

public BeatBox(Context context) {
    mAssets = context.getAssets();
    loadSounds();
}

private void loadSounds() {
    String[] soundNames;
    try {
        soundNames = mAssets.list(SOUNDS_FOLDER);
        Log.i(TAG, "Found " + soundNames.length + " sounds");
    } catch (IOException ioe) {
        Log.e(TAG, "Could not list assets", ioe);
        return;
    }
}

```

AssetManager.list(String)方法能列出指定目录中的所有文件名。因此，只要传入声音资源所在的目录，就能看到其中的所有.wav文件。

为了验证代码逻辑，在BeatBoxFragment中创建BeatBox实例，如代码清单18-11所示。

代码清单18-11 创建BeatBox实例（BeatBoxFragment.java）

```

public class BeatBoxFragment extends Fragment {

    private BeatBox mBeatBox;

    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

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

        mBeatBox = new BeatBox(getActivity());
    }

    ...
}

```

运行BeatBox应用。查看日志输出，看看列出了多少个声音文件。随书文件中提供了22个.wav

文件，不出意外的话，可看到如下结果：

```
...1823-1823/com.bignerdranch.android.beatbox I/BeatBox: Found 22 sounds
```

18.5 使用 Assets

获取到资源文件名之后，要将其展示给用户，最终还需要播放这些声音文件。所以，我们得创建一个对象，让它管理资源文件名、用户应该看到的文件名以及其他一些相关信息。

任务明确了，我们来创建一个这样的Sound管理类，如代码清单18-12所示。（别忘了，Android Studio可自动产生获取方法。）

代码清单18-12 创建Sound对象（Sound.java）

```
public class Sound {
    private String mAssetPath;
    private String mName;

    public Sound(String assetPath) {
        mAssetPath = assetPath;
        String[] components = assetPath.split("/");
        String filename = components[components.length - 1];
        mName = filename.replace(".wav", "");
    }

    public String getAssetPath() {
        return mAssetPath;
    }

    public String getName() {
        return mName;
    }
}
```

为有效显示声音文件名，我们在构造方法中对其进行处理。首先使用String.split(String)方法分离出文件名，再使用String.replace(String, String)方法删除.wav后缀。

接下来，在BeatBox.loadSounds()方法中创建一个Sound列表，如代码清单18-13所示。

代码清单18-13 创建Sound列表（BeatBox.java）

```
public class BeatBox {
    ...

    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();

    public BeatBox(Context context) {
        ...
    }

    private void loadSounds() {
        String[] soundNames;
```

```

try {
    ...
} catch (IOException ioe) {
    ...
}

for (String filename : soundNames) {
    String assetPath = SOUNDS_FOLDER + "/" + filename;
    Sound sound = new Sound(assetPath);
    mSounds.add(sound);
}
}

public List<Sound> getSounds() {
    return mSounds;
}
}

```

然后，回到BeatBoxFragment中，在SoundHolder中添加代码绑定Sound，如代码清单18-14所示。

代码清单18-14 绑定Sound（BeatBoxFragment.java）

```

private class SoundHolder extends RecyclerView.ViewHolder {
    private Button mButton;
    private Sound mSound;

    public SoundHolder(LayoutInflater inflater, ViewGroup container) {
        ...
    }

    public void bindSound(Sound sound) {
        mSound = sound;
        mButton.setText(mSound.getName());
    }
}

```

再让SoundAdapter与Sound数组列表关联起来，如代码清单18-15所示。

代码清单18-15 绑定Sound列表（BeatBoxFragment.java）

```

private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
    private List<Sound> mSounds;

    public SoundAdapter(List<Sound> sounds) {
        mSounds = sounds;
    }

    ...

    @Override
    public void onBindViewHolder(SoundHolder soundHolder, int position) {
        Sound sound = mSounds.get(position);
        soundHolder.bindSound(sound);
    }
}

```

```
}

@Override
public int getItemCount() {
    return 0;
    return mSounds.size();
}
}
```

最后，在onCreateView(...)方法中传入BeatBox声音资源，如代码清单18-16所示。

代码清单18-16 设置adapter (BeatBoxFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_beat_box, container, false);

    RecyclerView recyclerView = (RecyclerView)view
        .findViewById(R.id.fragment_beat_box_recycler_view);
    recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    recyclerView.setAdapter(new SoundAdapter());
    recyclerView.setAdapter(new SoundAdapter(mBeatBox.getSounds()));

    return view;
}
```

现在，再次运行BeatBox应用，应该可以看到如图18-6所示的结果。



图18-6 已完成的BeatBox应用界面

18.6 访问Assets

本章的全部工作完成了。下一章还会继续开发BeatBox应用，播放这些声音资源。

在继续之前，我们要花点时间深入探讨一下assets的工作原理。

我们知道，Sound对象定义了assets文件路径。尝试使用File对象打开资源文件是行不通的；正确的方式是使用AssetManager：

```
String assetPath = sound.getAssetPath();
InputStream soundData = mAssets.open(assetPath);
```

这样才能得到标准的InputStream数据流。随后，和Java中的其他InputStream一样，该怎么用就怎么用。

不过，有些API可能还会需要FileDescriptor。（下一章的SoundPool类会用到。）这也好办，如下列代码所示，改调用AssetManager.openFd(String)方法就行了。

```
String assetPath = sound.getAssetPath();
// AssetFileDescriptors are different from FileDescriptors,
AssetFileDescriptor assetFd = mAssets.openFd(assetPath);
// but you get can a regular FileDescriptor easily if you need to.
FileDescriptor fd = assetFd.getFileDescriptor();
```

18.7 深入学习：什么是non-assets

AssetManager类还有openNonAssetFd(...)这样的方法。想不通吧，assets专属类为什么要关心non-assets？我们可以回答“这也不是你应该关心的事！”，让你假装自己从没听说过它。

就我们所知，还真找不到使用它的理由，所以不学习的理由也很充分。

不过，既然读者花钱买了这本书，下面就简单介绍，权当福利好了。

还记得吗？前面说过，Android有assets和resources两大资源系统。resources资源系统设计有良好的检索系机制，但它无法应付图形和声音文件这样的大文件。这些大资源实际是保存在assets系统里的。在后台，Android就是使用openNonAsset方法来打开这些资源。当然，这样的方法有不少没有对用户开放。

现在了解了吧！有机会用到它们吗？我们的答案是永远不会。

使用SoundPool播放音频

19

既然音频资源文件已准备就绪，现在就来学习如何播放这些.wav音频文件。Android的大部分音频API都比较低级，掌握它们不是那么容易。不过没关系，针对BeatBox应用，可以使用SoundPool这个特别定制的实用工具。

SoundPool能加载一批声音资源到内存中，并支持同时播放多个音频文件。因此所以，就算用户兴奋起来，狂按按钮播放全部音频，也不必担心会损毁坏应用或耗光手机电量。

准备好了吗？开始着手吧。

19.1 创建 SoundPool

首先创建一个SoundPool对象，如代码清单19-1所示。

代码清单19-1 创建SoundPool对象（BeatBox.java）

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
    private static final int MAX_SOUNDS = 5;

    private AssetManager mAssets;
    private List<Sound> mSounds = new ArrayList<>();
    private SoundPool mSoundPool;

    public BeatBox(Context context) {
        mAssets = context.getAssets();
        // This old constructor is deprecated, but we need it for
        // compatibility.
        mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
        loadSounds();
    }

    ...
}
```

Lollipop引入了新的方式创建SoundPool：使用SoundPool.Builder。不过，为了兼容API 16最低级别，只能选择使用SoundPool(int, int, int)这个老构造方法了。

第一个参数指定同时播放多少个音频。这里指定了5个。在播放5个音频时，如果尝试再播放第6个，SoundPool会停止播放原来的音频。

第二个参数确定音频流类型。Android有很多不同的音频流，它们都有各自独立的音量控制选项。这就是调低音乐音量，闹钟音量却不受影响的原因。打开文档，查看AudioManager类的AUDIO_*常量，还可以看到其他控制选项。STREAM_MUSIC使用的是同音乐和游戏一样的音量控制。

最后一个参数指定采样率转换品质。参考文档说这个参数不起作用，所以这里传入0值。

19.2 加载音频文件

接下来使用SoundPool加载音频文件。相比其他音频播放方法，SoundPool还有个快速响应的优势：指令刚一发出，它就会立即开始播放，一点都不拖沓。

不过反应快也是有代价的，那就是在播放前必须预先加载音频。SoundPool加载的音频文件都有自己的Integer类型ID。如代码清单19-2所示，在Sound类中添加mSoundId实例变量，并添加相应的获取方法和设置方法管理这些ID。

代码清单19-2 添加mSoundId实例变量（Sound.java）

```
public class Sound {
    private String mAssetPath;
    private String mName;
    private Integer mSoundId;

    ...

    public String getName() {
        return mName;
    }

    public Integer getSoundId() {
        return mSoundId;
    }

    public void setSoundId(Integer soundId) {
        mSoundId = soundId;
    }
}
```

注意，mSoundId用了Integer类型而不是int。这样，在Sound的mSoundId没有值时可以设置其为null值。

现在处理音频加载。在BeatBox中添加一个load(Sound)方法载入音频，如代码清单19-3所示。

代码清单19-3 加载音频（BeatBox.java）

```
private void loadSounds() {
    ...
}
```

```

private void load(Sound sound) throws IOException {
    AssetFileDescriptor afd = mAssets.openFd(sound.getAssetPath());
    int soundId = mSoundPool.load(afd, 1);
    sound.setSoundId(soundId);
}
}

```

调用`mSoundPool.load(AssetFileDescriptor, int)`方法可以把文件载入`SoundPool`待播。为方便管理、重播或卸载音频文件, `mSoundPool.load(...)`方法会返回一个`int`型ID。这实际就是存储在`mSoundId`中的ID。调用`openFd(String)`方法有可能抛出`IOException`, `load(Sound)`方法也是如此。

现在, 在`BeatBox.loadSounds()`方法中, 调用`load(Sound)`方法载入全部音频文件, 如代码清单19-4所示。

代码清单19-4 载入全部音频文件 (BeatBox.java)

```

private void loadSounds() {
    ...
    for (String filename : soundNames) {
        try {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
            load(sound);
            mSounds.add(sound);
        } catch (IOException ioe) {
            Log.e(TAG, "Could not load sound " + filename, ioe);
        }
    }
}

```

运行应用确认音频都已正确加载。否则, 会看到LogCat中的红色异常日志。

19.3 播放音频

最后一步是播放音频。在`BeatBox`中再添加一个`play(Sound)`方法。

代码清单19-5 播放音频 (BeatBox.java)

```

mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
loadSounds();
}

public void play(Sound sound) {
    Integer soundId = sound.getSoundId();
    if (soundId == null) {
        return;
    }
    mSoundPool.play(soundId, 1.0f, 1.0f, 1, 0, 1.0f);
}

```

```
public List<Sound> getSounds() {  
    return mSounds;  
}
```

播放前要检查并确保soundId不是null值。Sound加载失败会导致soundId出现null值。

检查通过以后，就可以调用SoundPool.play(int, float, float, int, int, float)方法播放音频了。这些参数依次是：音频ID、左音量、右音量、优先级（无效）、是否循环以及播放速率。我们需要最大音量和常速播放，所以传入值1.0。是否循环参数传入0值，代表不循环。（如果想无限循环，可以传入-1。我们觉得这会非常令人讨厌。）

最后，添加按钮监听器方法，实现点击按钮播放音频，如代码清单19-6所示。

代码清单19-6 点击按钮播放音频（BeatBoxFragment.java）

```
private class SoundHolder extends RecyclerView.ViewHolder  
    implements View.OnClickListener {  
    private Button mButton;  
    private Sound mSound;  
  
    public SoundHolder(LayoutInflater inflater, ViewGroup container) {  
        super(inflater.inflate(R.layout.list_item_sound, parent, false));  
  
        mButton = (Button)itemView.findViewById(R.id.list_item_sound_button);  
        mButton.setOnClickListener(this);  
    }  
  
    public void bindSound(Sound sound) {  
        mSound = sound;  
        mButton.setText(mSound.getName());  
    }  
  
    @Override  
    public void onClick(View v) {  
        mBeatBox.play(mSound);  
    }  
}
```

再次运行应用。点击任意按钮，应该可以听到播放的音频了，如图19-1所示。



图19-1 音频库可用了

19.4 释放音频

应用BeatBox可用了，但别忘了做善后工作。音频播放完毕，应调用**SoundPool.release()**方法释放**SoundPool**，如代码清单19-7所示。在BeatBox.java中，添加一个**BeatBox.release()**清理方法。

代码清单19-7 释放SoundPool (BeatBox.java)

```
public class BeatBox {  
    ...  
  
    public void play(Sound sound) {  
        ...  
    }  
  
    public void release() {  
        mSoundPool.release();  
    }  
  
    ...  
}
```

在BeatBoxFragment.java中，使用完毕后，就在**onDestroy()**中调用这个释放方法，如代码清单19-8所示。

代码清单19-8 释放BeatBox (BeatBoxFragment.java)

```

public class BeatBoxFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mBeatBox.release();
    }

    ...
}

```

再次运行应用，确认新添加的**release()**方法已生效。

19.5 设备旋转和对象保存

处理完资源释放，再来看一个经典的设备旋转问题。播放69_ohm-loko音频，然后旋转设备，音乐声会戛然而止。（如果没有，请确认已经实施了**onDestroy()**方法，并重新编译和运行应用。）

下面具体分析一下这个问题：设备旋转时，**BeatBoxActivity**随即被销毁。与此同时，**FragmentManager**也会销毁**BeatBoxFragment**。在销毁过程中，它会逐一调用**BeatBoxFragment**的**onPause()**、**onStop()**和**onDestroy()**生命周期方法。在**BeatBoxFragment.onDestroy()**方法中，**BeatBox.release()**方法会被调用。这会释放**SoundPool**，音频播放自然也就停止了。

前面，我们看到过**Activity**和**Fragment**因设备旋转而销毁的问题。当时使用**onSaveInstanceState(Bundle)**方法解决了问题。然而，这里用不上该方法，因为需要首先保存数据，然后再使用**Bundle**中的**Parcelable**恢复数据。

类似于**Serializable**，**Parcelable**是个把对象以字节流的方式保存的API。对于可保存对象，可以让它实现**Parcelable**接口。在Java世界，要保存对象，要么将其放入**Bundle**中，要么实现**Serializable**接口或者**Parcelable**接口。无论采用哪种方式，对象首先要是可保存对象。

怎么理解可保存呢？举个例子你就明白了。你和朋友在看电视节目。什么频道、音量大小、甚至是电视型号，你都可以记下来。如此一来，就算发生火灾警报、停电这样的事情，等一切恢复正常，你依然可以回去继续观看原来的电视节目。

由此可知，当前所看电视节目的配置是可以保存的。不过，我们观看节目的那段时间却无法保存：一旦发生火警或停电，其间那段时光就流逝掉了。就算恢复观看，流逝掉的那段再也找不回来了。所以，什么能保存，什么不能保存，再清楚不过了。

`BeatBox`的某一部分可以保存，例如，`Sound`类中的一切都可以保存；而`SoundPool`就无法保存了。虽然可以新建包含同样音频文件的`SoundPool`，或者从音频播放中断处继续，然而和电视播放例子一样，播放中断的那段时光是无论如何也找不回了。

不可保存性有向外传递的倾向。如果一个对象重度依赖另一个不可保存的对象，那么这个对象很可能也无法保存。`BeatBox`和`SoundPool`就是这样的一对。`SoundPool`要依靠`BeatBox`播放音频。基于这个事实，可以证明`BeatBox`也是无法保存的。（抱歉。）

`savedInstanceState`机制只适用于可保存的对象数据，但`BeatBox`不可保存。在`Activity`创建和销毁时，我们都需要`BeatBox`实例一直可用。

这个难题该怎么解决呢？

19.5.1 保留 fragment

告诉你个好消息，为应对设备配置变化，`fragment`有个特殊方法可确保`BeatBox`实例一直存在：`retainInstance`。覆盖`BeatBoxFragment.onCreate(...)`方法并设置`fragment`的属性值，如代码清单19-9所示。

代码清单19-9 调用`setRetainInstance(true)` (`BeatBoxFragment.java`)

```
...
public static BeatBoxFragment newInstance() {
    return new BeatBoxFragment();
}

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

    mBeatBox = new BeatBox(getActivity());
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    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，它会一直保留并在需要时原封不动地传递给新的`activity`。

对于已保留的`fragment`实例，其全部实例变量（如`mBeatBox`）值也会保持不变，因此可放心继续使用。

运行`BeatBox`应用。播放`69_ohm-loko`声音文件，然后旋转设备，确认音频播放不受影响。

19.5.2 旋转和已保留 fragment

我们来看看保留fragment的工作原理。保留的fragment利用了这样一个事实：可以销毁和重建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，这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使用时，则应重建视图。

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19-2所示。

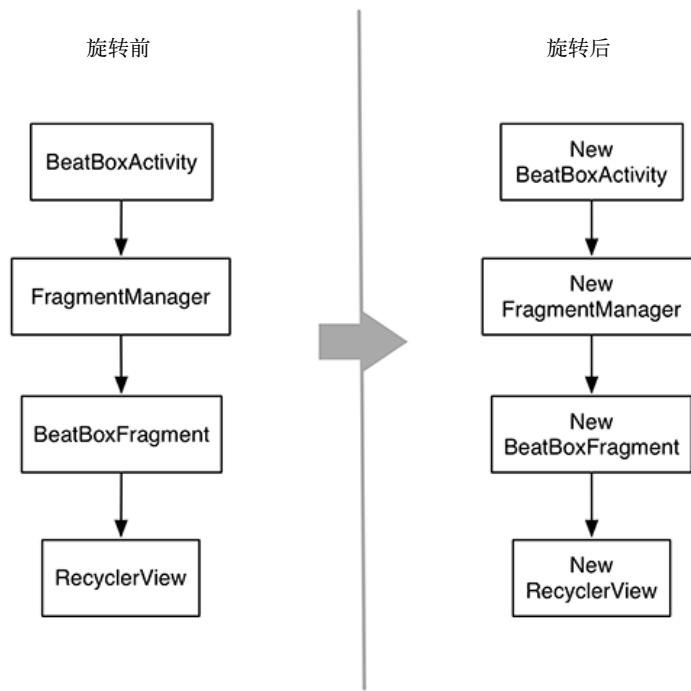


图19-2 设备旋转与默认不保留的UI fragment

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19-3所示。

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托管它，如图19-4所示。

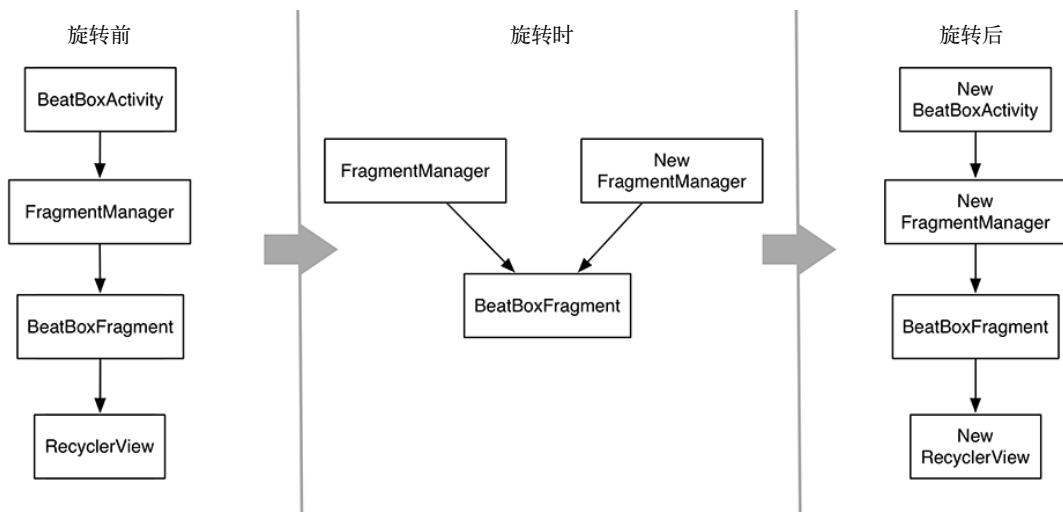


图19-3 设备旋转与保留的UI fragment

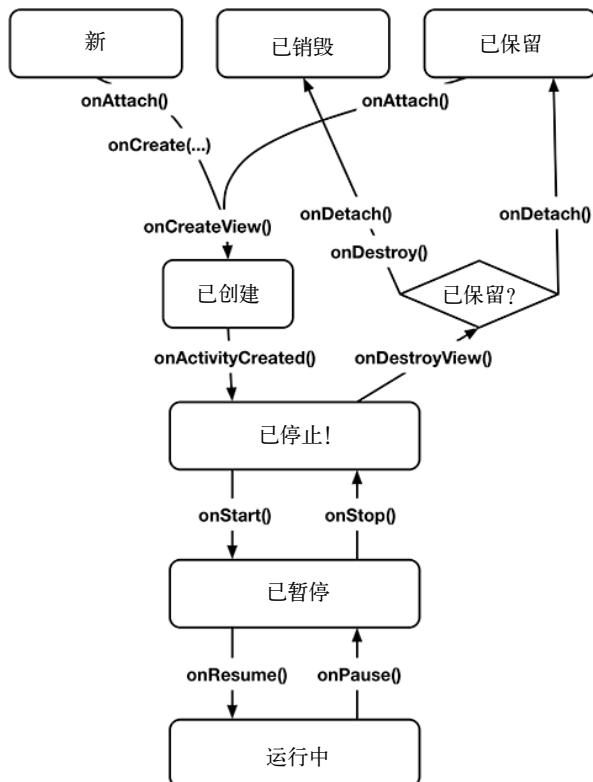


图19-4 fragment的生命周期

必须同时满足两个条件，fragment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只能短暂处于保留状态，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。

19.6 深入学习：是否要保留

保留fragment可以说是Android的巧妙设计，不是吗？没错！它确实极大地便利了应用开发，似乎解决了因设备旋转而销毁activity和fragment所导致的全部问题。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。

你可能会疑惑：为什么不保留所有fragment？为什么fragment的`retainInstance`默认属性值不是`true`？这是因为，除非万不得已，我们非常不建议使用这种机制。下面给出理由。

首先，相比非保留fragment，保留fragment的显示非常复杂。一旦出现问题，排查起来非常耗时。既然使用它会让程序复杂起来，能不用就不用吧。

其次，fragment在使用保存实例状态的方式处理设备旋转时，也能够应对所有生命周期场景；但保留的fragment只能处理activity因设备旋转而销毁的情况。如果activity是因操作系统需要回收内存而被销毁，则所有保留的fragment也会随之销毁，数据也就跟着丢失了。

19.7 深入学习：设备旋转处理再探

`onSaveInstanceState(Bundle)`方法是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，这就要归功于`onSaveInstanceState(...)`方法的默认工作行为。

CriminalIntent应用就是个极好的例子。`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。如只需短暂保留数据，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，则更能体现保留fragment的优势，因为再也无需操心要保存对象是否已实现`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也会随之销毁。

为了更清楚地看到两种数据保存方式的差异，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就被重置为零。无论用户在回答哪一道题目，设备旋转后，总是会回到第一道题目。因此，保存题目的索引值，然后在设备旋转后重新读取该值，

就能保证用户看到正确的题目。

GeoQuiz应用没有使用fragment。现在假设以QuizActivity托管一个QuizFragment的方式，重新设计GeoQuiz应用。那么，是覆盖**Fragment.onSaveInstanceState(...)**方法保存题目索引值，还是以保留QuizFragment的方式保存变量呢？

图19-5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。

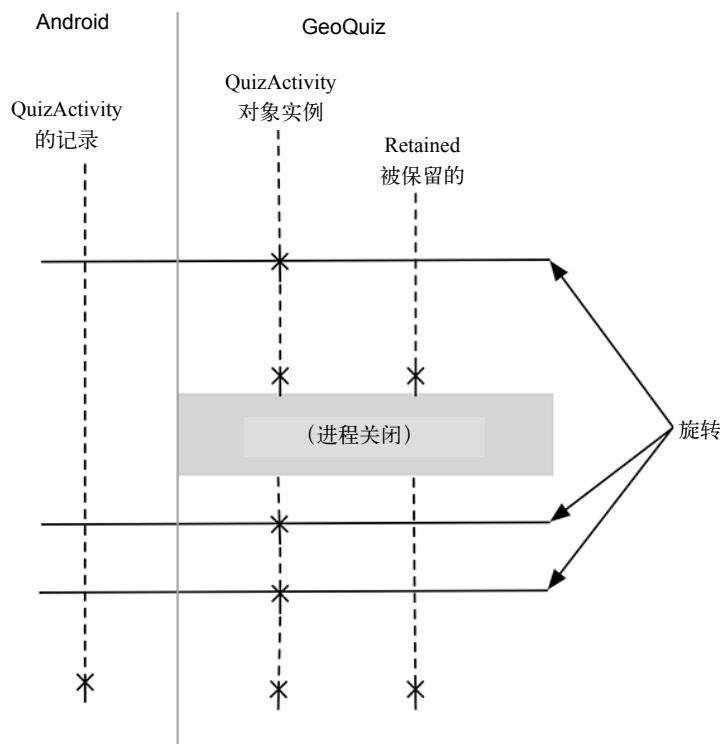


图19-5 三种不同的生命周期

activity对象的生命周期极短。这也是产生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，因此以保留QuizFragment的方式处理设备旋转问题相对容易一些，且涉及较少的代码量。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19-10所示。

代码清单19-10 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，进而解决了设备旋转导致的索引变量值重置问题。然而，如图19-5所示，进程关闭时，被保留QuizFragment中的索引变量值也会丢失。进程的关闭通常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。

应用当前只有五道题目，让用户从头来过勉强可行；但如果GeoQuiz应用有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，题目索引变量的存留时间与activity记录的生命周期应保持同步。因此，应设法在onSaveInstanceState(...)方法中保存题目索引变量值。这样，用户暂时离开应用再返回时，仍可接着答题。

因此，如果activity或fragment中有需要长久保存的东西，应覆盖onSaveInstanceState(Bundle)方法，保存其状态。这样，由于和activity记录的生命周期保持了同步，后续就可在需要时恢复。



既然BeatBox应用的音效能吓退人，它的用户界面也应透出一定的威慑力。

当前，BeatBox应用依然固守着默认的用户界面样式。按钮是最普通的那种，配色也平淡无奇。整个应用看上去毫不起眼，没有品牌特色。

不过我们有技术，可以使用样式和主题，定制出需要的用户界面风格。

定制界面的最终效果如图20-1所示。与之前相比，新界面更加美观、惹眼，独具风格。

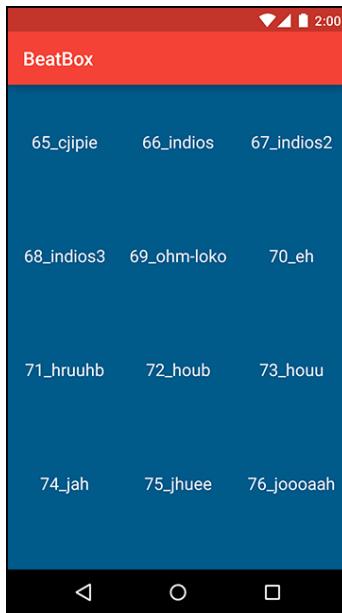


图20-1 应用了主题的BeatBox

20.1 颜色资源

首先，我们来定义本章要用到的颜色资源。参照代码清单20-1，在res/values中创建一个colors.xml文件。

代码清单20-1 定义一些颜色 (res/values/colors.xml)

```
<resources>
    <color name="red">#F44336</color>
    <color name="dark_red">#C3352B</color>
    <color name="gray">#607D8B</color>
    <color name="soothing_blue">#0083BF</color>
    <color name="dark_blue">#005A8A</color>
</resources>
```

使用颜色资源，可以方便地在一处定义各种颜色值供应用引用。

20.2 样式

现在，我们来给按钮添加样式。样式是一套能够应用于视图组件的属性。

打开res/values/styles.xml样式文件，添加BeatBoxButton新样式，如代码清单20-2所示。（创建BeatBox项目时，向导会创建默认的styles.xml文件。如果没有，请自行创建。）

代码清单20-2 添加样式 (res/values/styles.xml)

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>
```

新建样式名叫BeatBoxButton。该样式仅定义了android:background属性，属性值为深蓝色。样式可以为很多组件共用，更新修改属性时，只修改公共样式定义就行了。

定义好样式，把它添加给各个按钮，如代码清单20-3所示。

代码清单20-3 使用样式 (res/layout/list_item_sound.xml)

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    style="@style/BeatBoxButton"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

运行BeatBox应用。可以看到，所有按钮的背景都是深蓝色了，如图20-2所示。

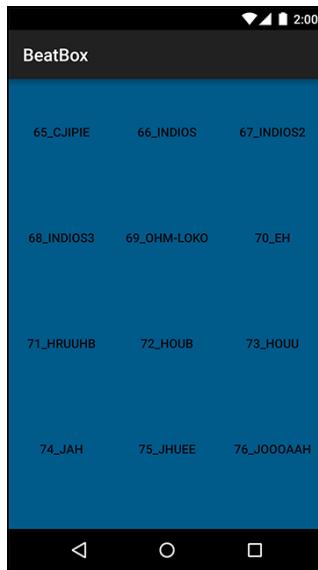


图20-2 添加了样式的按钮

如果需要，可以创建带多套属性的样式在应用里复用。这真是方便。

样式继承

样式支持继承。一个样式能继承并覆盖其他样式的属性。

创建一个名为BeatBoxButton.Strong的新样式。除了继承BeatBoxButton样式的按钮背景属性，再添加自己的`android:textStyle`属性，用粗体显示按钮文字，如代码清单20-4所示。

代码清单20-4 继承样式（res/layout/styles.xml）

```
...
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="BeatBoxButton.Strong">
    <item name="android:textStyle">bold</item>
</style>

...
```

（当然，可以直接对BeatBoxButton样式添加这个`android:textStyle`属性。创建BeatBoxButton.Strong样式只是为了演示样式继承。）

新样式的命名有点特别。BeatBoxButton.Strong的命名表明，这个新样式继承了BeatBoxButton样式的属性。

除了通过命名表示样式继承关系，也可以采用指定父样式的形式：

```
<style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
</style>

<style name="StrongBeatBoxButton" parent="@style/BeatBoxButton">
    <item name="android:textStyle">bold</item>
</style>
```

虽然有新方式用于继承样式，BeatBox应用还是继续使用特殊命名方式。

更新list_item_sound.xml布局，用上新的粗体文字样式，如代码清单20-5所示。

代码清单20-5 使用粗体文字样式（res/layout/list_item_sound.xml）

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    style="@style/BeatBoxButton.Strong"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

运行应用，确认按钮文字已显示为粗体，如图20-3所示。

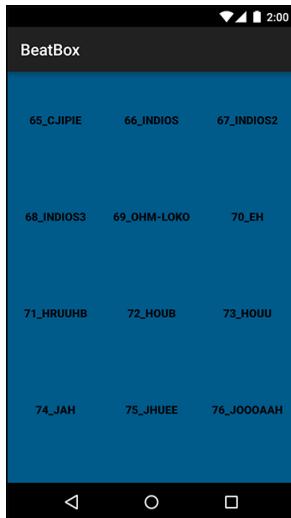


图20-3 使用了粗体文字样式的BeatBox

20.3 主题

样式很有用。在styles.xml公共文件中，可以为所有组件定义一套样式属性共用。可惜，定义公共样式属性虽方便，实际应用却很麻烦：需要逐个为所有组件添加它们要用到的样式。要是开

发一个复杂应用，涉及很多布局、无数按钮，仅仅添加样式就需要巨大的工作量。

该是主题闪亮登场的时候了！可以把主题看作样式的进化加强版。同样是定义一套公共主题属性，样式属性需要逐个添加，而主题属性则会自动应用于整个应用。主题属性能引用颜色这样的外部资源，也能引用其他样式。使用主题，可以简单地说：“所有按钮都使用这个样式。”再也不用找到每个按钮，告诉它们要用哪个主题了。

修改默认主题

BeatBox项目自带默认主题。找到并打开AndroidManifest.xml文件，可以看到application标签下的theme属性，如代码清单20-6所示。

代码清单20-6 BeatBox的默认主题（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.beatbox" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        ...
    </application>

</manifest>
```

theme属性指向的主题叫AppTheme。它也定义在styles.xml文件中。

你可能已猜到，主题实际就是一种样式。但是主题指定的属性有别于样式。（稍后就会看到。）既然是在manifest文件中声明它，主题自然就有了魔力。这也解释了为什么主题可以自动应用于整个应用。

要查看AppTheme主题定义，只要按住Command键（Windows系统是Ctrl键），点击@style/AppTheme，Android Studio就会自动打开res/values/styles.xml文件，如代码清单20-7所示。

代码清单20-7 BeatBox的AppTheme（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>

    <style name="BeatBoxButton">
        <item name="android:background">@color/dark_blue</item>
    </style>

    ...
</resources>
```

(本书编写时，在Android Studio中创建的项目都自带AppCompat主题。如果你的BeatBox项目没有，请参考第13章让项目使用AppCompat库。)

AppTheme现在继承Theme.AppCompat.Light.DarkActionBar的全部属性。如有需要，可以添加自己的属性值，或是覆盖父主题的某些属性值。

AppCompat库自带三大主题：

- Theme.AppCompat——深色主题
- Theme.AppCompat.Light——浅色主题
- Theme.AppCompat.Light.DarkActionBar——带深色工具栏的浅色主题

把AppTheme的父主题修改为Theme.AppCompat，如代码清单20-8所示。这样BeatBox项目就有了一个深色主题基板。

代码清单20-8 改用深色主题（res/values/styles.xml）

```
<resources>

    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>

</resources>
```

运行BeatBox应用，查看新的深色主题，如图20-4所示。

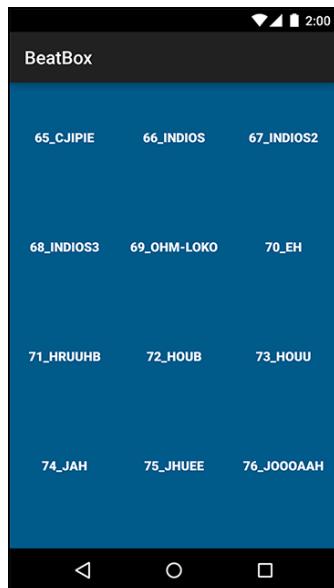


图20-4 应用了深色主题的BeatBox

20.4 添加主题颜色

现在，基于AppTheme主题模板来定制它的属性。

在styles.xml文件中，为AppTheme主题添加三个自定义属性，如代码清单20-9所示。

代码清单20-9 自定义主题属性（res/values/styles.xml）

20

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>
    </style>
    ...
</resources>
```

虽然这三个主题属性看上去和前面的样式属性差不多，但它们指定的属性名不一样。另外，它们的应用范围也不一样。样式属性仅适用于单个组件，如前面用粗体显示按钮文字的textStyle。主题属性则适用于所有使用同一主题的组件。例如，工具栏会以主题的colorPrimary属性设置自己的背景色。

使用这三个主题属性，应用界面能获得显著改观。colorPrimary属性主要用来设置工具栏背景色。由于应用名称是显示在工具栏上的，colorPrimary也可以称为应用品牌色。

colorPrimaryDark用于屏幕顶部的状态栏。从名字可以看出，它是深色版colorPrimary。注意，只有Lollipop以后的系统支持状态栏主题色。对于之前的系统，无论指定什么主题色，状态栏都是不变的黑底色。图20-5展示了这两种主题色的应用效果。

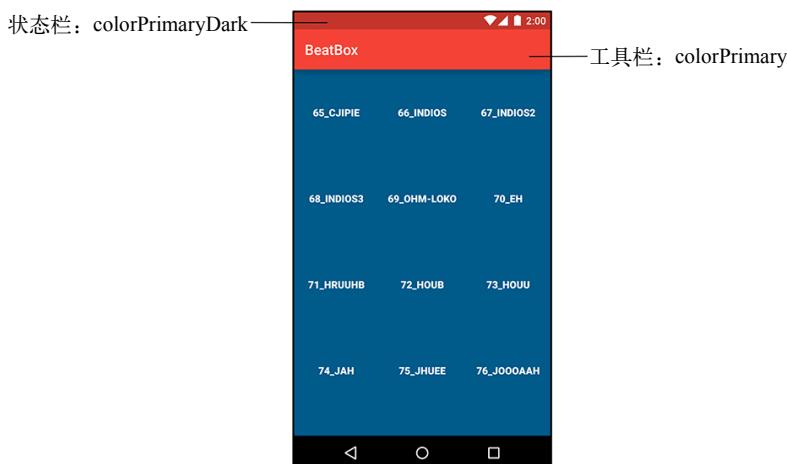


图20-5 带AppCompat颜色属性的BeatBox

最后，将`colorAccent`设置为灰色的。这个主题色应该和`colorPrimary`形成反差效果，主要用于给`EditText`这样的组件着色。

按钮组件不支持着色，所以`colorAccent`主题色在BeatBox项目中没有效果。不管有没有用，这里还是添加了`colorAccent`，因为最好能配套使用这三个主题属性。既然要搭配使用它们，继承自父主题的默认`colorAccent`可能会和你指定的其他两种主题色冲突，这一点要注意。至此，BeatBox应用的主题已像模像样，后面再调整也就方便多了。

运行应用查看主题效果。现在，应用界面看起来应该和图20-5一样。

20.5 覆盖主题属性

完成了主题配色，我们继续来点深入的，看看可以覆盖的主题属性都有哪些。给你提个醒，主题深挖之路坎坷崎岖，可不那么好走。研究诸如有哪些主题属性可用，哪些能覆盖，甚至是有些属性究竟有什么作用等这样的问题时，几乎没有官方参考文档可以参考还是小事，你很可能你还会完全迷失方向。所以，进行这方面的研究时最好是买本有此专题的书看看作指导，比如本书。

我们的首个目标是修改主题以改变BeatBox应用的背景色。当然，你可以打开`res/layout/fragment_beat_box.xml`文件，手工设置`RecyclerView`视图的`android:background`属性。如果还有其他`fragment`和`activity`要改，都照此处理。这简直是浪费：浪费时间，浪费系统资源。

主题已经设置了背景色，在此基础上再设置其他颜色，就是多出来在做额外的工作。而且，在应用里到处复制使用背景属性设置代码也不利于后期维护。

主题探秘

要解决上述问题，应设法覆盖主题背景色属性。为了找出可覆盖属性的名字，先来看看这个属性在其父主题里是怎么设置的：`Theme.AppCompat`。

你可能会想：“不知道名字，我怎么知道该覆盖哪个属性呢？”确实是这样。所以，首先查看目标属性的名字，凭直觉挑一个，覆盖它，然后运行应用验证你的猜想。

你需要找出主题继承的源头。主题继承树有多少深，谁也不知道，只能一层层向上找，直到找到`AppCompat`库的外面。

打开`styles.xml`文件，按住Command键（Windows系统是Ctrl键）点击`Theme.AppCompat`，来看看继承有多深。

（如果无法直接在Android Studio里追溯主题属性，或是想在工具之外查找，可以在`your-SDK-directory/platforms/android-21/data/res/values`目录找到主题源码。）

Android开发工具更新频繁，本书编写时，Android Studio会定位到一个大文件的这行：

```
<style name="Theme.AppCompat" parent="Base.Theme.AppCompat" />
```

`Theme.AppCompat`主题属性继承自`Base.Theme.AppCompat`。有趣的是，`Theme.AppCompat`本身没有覆盖任何属性，仅仅指向了其父主题。

按住Command键再点击Base.Theme.AppCompat，Android Studio会提示这个主题有多个版本。选择values-v14/values.xml版本，定位到如图20-6所示的主题定义。

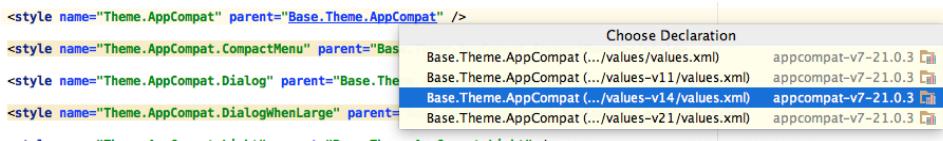


图20-6 选择v14父主题

(BeatBox支持16及以上API级别，所以这里选择了v14版本。如果选择v21版本，很可能还会看到API 21级里添加的新特性。想了解具体是什么特性，可阅读本章末的挑战练习部分。)

```
<style name="Base.Theme.AppCompat" parent="Base.V14.Theme.AppCompat">
    <item name="android:actionModeCutDrawable">?actionModeCutDrawable</item>
    <item name="android:actionModeCopyDrawable">?actionModeCopyDrawable</item>
    <item name="android:actionModePasteDrawable">?actionModePasteDrawable</item>
    <item name="android:actionModeSelectAllDrawable">?actionModeSelectAllDrawable</item>
    <item name="android:actionModeShareDrawable">?actionModeShareDrawable</item>
</style>
```

这个主题定义了不少属性，但找不到改变背景色的属性。继续定位到Base.V14.Theme.AppCompat主题。

```
<style name="Base.V14.Theme.AppCompat" parent="Base.V11.Theme.AppCompat" />
```

可以看到，这个主题也没有覆盖任何属性，那就再看看Base.V11.Theme.AppCompat主题。

```
<style name="Base.V11.Theme.AppCompat" parent="Base.V7.Theme.AppCompat" />
```

还是个空主题。继续看Base.V7.Theme.AppCompat。

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
    <item name="windowActionBar">true</item>
    <item name="windowActionBarOverlay">false</item>

    ...
</style>
```

距离目标越来越近了。Base.V7.Theme.AppCompat有许多属性，但还是没找到想要的。照此逐个解开这些AppCompat主题，就会发掘出越来越多的属性。继续定位到Platform.AppCompat。这个主题也有多个版本，选择values-v11/values.xml版本。

```
<style name="Platform.AppCompat" parent="android:Theme.Holo">
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowActionBar">false</item>
    <item name="buttonBarStyle">?android:attr/buttonBarStyle</item>
    <item name="buttonBarButtonStyle">?android:attr/buttonBarButtonStyle</item>
    <item name="selectableItemBackground">?android:attr/selectableItemBackground</item>
    ...
</style>
```

终于，在这里我们看到了Platform.AppCompat的android:Theme.Holo父主题。

注意，这里引用的不是Theme.Holo，而是android:Theme.Holo。前面的android命名空间不能丢。

AppCompat库可以看作BeatBox应用的一部分。编译项目时，工具会引入AppCompat库和它的一堆Java和XML文件。这些文件已包含在应用里，如同你自己编写的文件。如果想引用AppCompat库里的资源，像Theme.AppCompat这样，直接引用就可以了。

有些主题包含在Android操作系统里，引用时必须加上命名空间，如Theme.Holo。所以，在引用Theme.Holo主题时，AppCompat库就使用了android:Theme.Holo这样的形式。

现在，定位到android:Theme.Holo主题。

```
<style name="Theme.Holo">
    <item name="colorForeground">@color/bright_foreground_holo_dark</item>
    <item name="colorForegroundInverse">...</item>
    <item name="colorBackground">@color/background_holo_dark</item>
    <item name="colorBackgroundCacheHint">...</item>
    <item name="disabledAlpha">0.5</item>
    <item name="backgroundDimAmount">0.6</item>
    ...
</style>
```

总算找到了。在这里，终于可以看到所有可以覆盖的主题属性。当然，还可以继续定位到Theme主题，不过没这个必要。Holo主题已经覆盖了所有我们想要的属性。

查看代码，可以看到colorBackground这个属性。顾名思义，这就是用于主题背景色的属性。

```
<style name="Theme.Holo">
    <item name="colorForeground">@color/bright_foreground_holo_dark</item>
    <item name="colorForegroundInverse">...</item>
    <item name="colorBackground">@color/background_holo_dark</item>
    ...
</style>
```

这也是要在BeatBox应用中覆盖的属性。回到styles.xml文件中，覆盖colorBackground这个属性，如代码清单20-10所示。

代码清单20-10 设置窗口背景（res/values/styles.xml）

```
<style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/red</item>
    <item name="colorPrimaryDark">@color/dark_red</item>
    <item name="colorAccent">@color/gray</item>

    <item name="android:colorBackground">@color/soothing_blue</item>
</style>
```

注意，colorBackground这个属性来自Android操作系统，所以别忘了使用android命名空间。运行BeatBox应用。滚动到recycler视图底部查看背景，没有按钮覆盖的地方是浅蓝色，如图20-7所示。

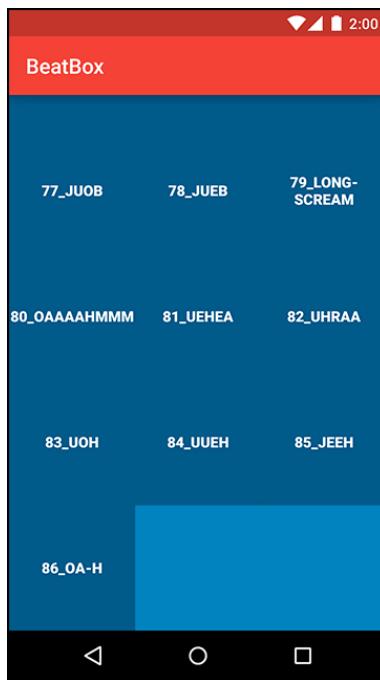


图20-7 设置了主题背景的BeatBox

如果想修改应用主题，每个开发人员差不多都要经历刚才查找colorBackground属性的过程。没办法，这些属性没有什么文档可参考，只能去看源代码了。

总结一下，刚才我们定位查看了以下主题：

- Theme.AppCompat
- Base.Theme.AppCompat
- Base.V14.Theme.AppCompat
- Base.V11.Theme.AppCompat
- Base.V7.Theme.AppCompat
- Platform.AppCompat
- android:Theme.Holo

刚才我们自下而上逐层定位，直到找到Android操作系统中的某个主题（AppCompat库之外）。将来，越来越熟练之后，你很可能会跳掉中间步骤而直达目标。不过，建议还是按部就班，这样可以清楚究竟哪个是根主题。

最后再提个醒，随着时间的推移，主题继承关系和层次可能有变，但上面介绍的方法不会变。想要知道该覆盖哪个属性，就沿着继承树找吧！

20.6 修改按钮属性

前面，我们通过在res/layout/list_item_sound.xml文件中手工设置样式属性，定制过BeatBox应用的按钮。如果一个复杂应用有很多带按钮的fragment，再去逐个fragment、逐个按钮地去设置style属性就很不应该了。在这种情况下，还是要靠主题。你可以在主题中定义一个用于所有按钮的样式。

在主题里添加按钮样式前，先打开res/layout/list_item_sound.xml文件，删掉原有样式属性，如代码清单20-11所示。

代码清单20-11 删掉！有更好的办法了（res/layout/list_item_sound.xml）

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    style="@style/BeatBoxButton.Strong"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

运行BeatBox应用。可以看到，按钮回到原来的模样了。

再次定位查看Theme.Holo主题定义，查找按钮属性。

```
<style name="Theme.Holo">
    ...
    <!-- Button styles -->
    <item name="buttonStyle">@style/Widget.Holo.Button</item>

    <item name="buttonStyleSmall">@style/Widget.Holo.Button.Small</item>
    <item name="buttonStyleInset">@style/Widget.Holo.Button.Inset</item>

    ...
</style>
```

注意到有个buttonStyle属性，这是应用中普通按钮的样式。

这个buttonStyle属性没有设置值，而是指向了一个样式资源。前面覆盖colorBackground属性时，直接传入了颜色值。这里，buttonStyle应该指向另一个样式。定位并查看Widget.Holo.Button样式。

```
<style name="Widget.Holo.Button" parent="Widget.Button">
    <item name="background">@drawable/btn_default_holo_dark</item>
    <item name="textAppearance">?attr/textAppearanceMedium</item>
    <item name="textColor">@color/primary_text_holo_dark</item>
    <item name="minHeight">48dip</item>
    <item name="minWidth">64dip</item>
</style>
```

BeatBox应用的所有按钮都使用了这些属性。

在BeatBox应用里，复用Android自身主题。修改BeatBoxButton样式的父样式为android:style/Widget.Holo.Button。另外，删除BeatBoxButton.Strong样式，如代码清单20-12所示。

代码清单20-12 创建按钮样式（res/values/styles.xml）

```

<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:colorBackground">@color/soothing_blue</item>
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

    <style name="BeatBoxButton.Strong">
        <item name="android:textStyle">bold</item>
    </style>

</resources>

```

继承`android:style/Widget.Holo.Button`样式，就是首先让所有按钮都继承常规按钮的属性。然后根据需要，有选择性地修改一些属性。

如果不指定`BeatBoxButton`样式的父样式，所有按钮会变得不再像个按钮，连按钮中间显示的文字都会丢失。

`BeatBoxButton`样式已重新定义完毕，可以使用了。经过前面主题深挖，我们知道要覆盖`buttonStyle`属性。下面覆盖`buttonStyle`属性，让它指向`BeatBoxButton`样式，如代码清单20-13所示。

代码清单20-13 使用BeatBoxButton样式（res/values/styles.xml）

```

<resources>

    <style name="AppTheme" parent="Theme.AppCompat">
        <item name="colorPrimary">@color/red</item>
        <item name="colorPrimaryDark">@color/dark_red</item>
        <item name="colorAccent">@color/gray</item>

        <item name="android:colorBackground">@color/soothing_blue</item>
        <item name="android:buttonStyle">@style/BeatBoxButton</item>
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@color/dark_blue</item>
    </style>

</resources>

```

运行BeatBox应用，所有的按钮都变成了深蓝色了，如图20-8所示。我们没有直接修改任何布局，就改变了普通按钮的样子。Android主题属性太强大了！

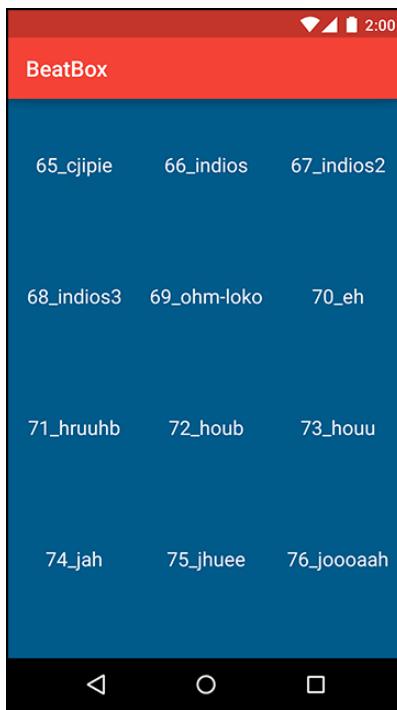


图20-8 带最终版主题的BeatBox

因为没有样式切换，点击按钮时，按钮没有任何变化，所以看不出究竟按了哪个按钮。下一章会修正这个问题，敬请期待！

20.7 深入学习：样式继承拾遗

本章前面对样式继承知识点的介绍还不够全面。在进行主题探秘时，你可能已经注意到了，样式继承的表示法时有切换。AppCompat主题都是使用主题名表示继承，直到碰到Platform.AppCompat这个主题。

```
<style name="Platform.AppCompat" parent="android:Theme.Holo">
    ...
</style>
```

在这里，继承是直接使用parent属性来表示的。为什么呢？

要以主题名的形式指定父主题，有继承关系的两个主题都应处于同一个包中。因此，对于Android操作系统内部主题间的继承，就可以直接使用主题名继承表示法。同理，AppCompat库内部也是这样。然而，一旦AppCompat库要跨库继承，就一定要明确使用parent属性。

在开发自己的应用时，应遵守同样的规则。如果是继承自己内部的主题，使用主题名指定父主题即可；如果是继承Android操作系统中的样式或主题，记得使用parent属性。

20.8 深入学习：引用主题属性

在主题中定义好属性后，可以在XML或代码中直接使用它们。

为了在XML中引用主题属性，我们使用第17章中`divider`属性用到的符号。在XML中引用具体值时（如颜色值），我们使用@符号。`@color/gray`指向某个特定资源。

在主题中引用资源时，我们使用?符号。

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    android:background="?attr/colorAccent"
    tools:text="Sound name"/>
```

上述XML中?符号的意思是使用`colorAccent`属性指向的资源。这里，是指定义在`colors.xml`文件中的灰色。

也可以在代码中使用主题属性，但是比较啰嗦。

```
Resources.Theme theme = getActivity().getTheme();
int[] attrsToFetch = { R.attr.colorAccent };
TypedArray a = theme.obtainStyledAttributes(R.style.AppTheme, attrsToFetch);
int accentColor = a.getInt(0, 0);
a.recycle();
```

先取得`Theme`对象，然后要求它找到定义在`AppTheme`（即`R.style.AppTheme`）中的`R.attr.colorAccent`属性。结果得到一个持有数据的`TypedArray`对象。接着，向`TypedArray`对象索要`int`值以取出颜色。取出颜色值之后就可以使用了，比如，用来更改按钮背景色。

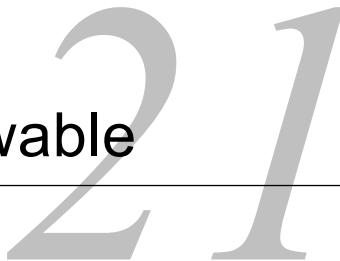
`BeatBox`应用中的工具栏和按钮就是采取上述方式使用主题属性美化自己的。

20.9 挑战练习：创建多版本主题

创建`BeatBoxButton`样式时，我们继承了`android:style/Widget.Holo.Button`中的一些属性。虽然可行，但没有用上最新的系统主题。

Google在Android 5.0（Lollipop）中提供了material主题。这个新主题修改了很多包括字体大小在内的按钮属性。如果设备支持material主题，为什么不用这个更美观的新主题呢？

挑战来了：请创建一个带资源修饰符的`styles.xml`文件：`values-v21/styles.xml`。然后，创建两个版本的`BeatBoxButton`样式，一个继承`Widget.Holo.Button`，另一个继承`Widget.Material.Button`。



BeatBox应用的主题非常漂亮，下面该是优化按钮表现的时候了。

当前，按钮就是个蓝方框，点击它也看不到任何反应。本章，我们将学习使用XML drawable，继续美化BeatBox应用，让它拥有如图21-1所示的用户界面。



图21-1 完全改观的用户界面

在Android世界里，凡是要在屏幕上绘制的东西都可以叫作drawable，比如抽象图形、Drawable类的子类、位图图像等。我们之前用来封装图片的BitmapDrawable就是一种drawable。本章，我们还会看到更多的drawable：state list drawable、shape drawable和layer list drawable。这三个drawable都定义在XML文件中，可以归为一类，统称为XML drawable。

21.1 统一按钮样式

定义XML drawable之前，先修改list_item_sound.xml文件隔开按钮，如代码清单21-1所示。

代码清单21-1 隔开按钮（res/layout/list_item_sound.xml）

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>

<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_margin="8dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/list_item_sound_button"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        tools:text="Sound name"/>
</FrameLayout>
```

现在，按钮的宽和高都是100dp。这样，稍后变为圆形时，这些按钮就不会歪斜了。

不论屏幕大小，recycler视图总是显示三列按钮。如果还有多余的空间，它会拉伸列格以适配屏幕。不过，按钮不应拉伸，所以把它们封装在frame布局里。

运行BeatBox应用。按钮的尺寸都完全统一了，并且彼此之间还留出了空间，如图21-2所示。

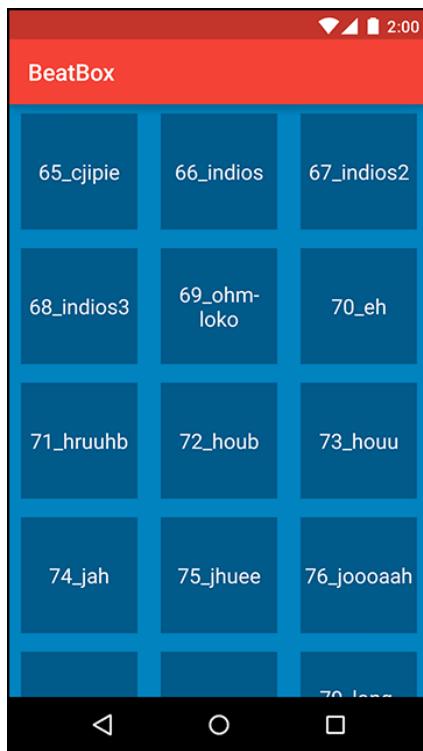


图21-2 隔开的按钮

21.2 shape drawable

使用ShapeDrawable，可以把按钮变成圆形。XML drawable和屏幕像素密度无关，所以无需考虑创建特定像素密度目录，直接把它放入默认的drawable文件夹就可以了。

打开项目工具窗口，在res/drawable目录下创建一个名为button_beat_box_normal.xml的文件，如代码清单21-2所示。（稍后还会创建一个“非正常”的文件，所以文件名里有normal（正常）字样。）

代码清单21-2 创建圆形drawable（res/drawable/button_beat_box_normal.xml）

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/dark_blue"/>

</shape>
```

该XML文件定义了一个背景为深蓝色的圆形。也可使用shape drawable定制其他各种图形，

如长方形、线条以及梯形等。欲详细了解shape drawable定制信息，可查看开发者文档：<http://developer.android.com/guide/topics/resources/drawable-resource.html>)。

在styles.xml中，使用新建的button_beat_box_normal作为按钮背景，如代码清单21-3所示。

代码清单21-3 修改按钮背景（res/values/styles.xml）

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat">
        ...
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@color/dark_blue</item>
        <item name="android:background">@drawable/button_beat_box_normal</item>
    </style>
</resources>
```

运行BeatBox应用。可以看到，圆形的按钮出现了，如图21-3所示。

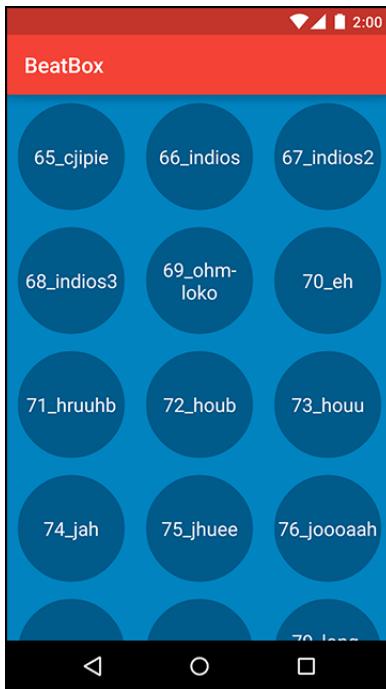


图21-3 圆形按钮

点击按钮之后会听到播放的声音，可按钮的样子却没有任何变化。按钮按下去时，如果能切换显示状态，用户体验应该会更好。

21.3 state list drawable

为解决这个问题，首先定义一个用于按钮按下状态的shape drawable。

在res/drawable目录下再创建一个名为button_beat_box_pressed.xml的文件，如代码清单21-4所示。除了背景颜色是红色外，这个shape drawable和前面的正常版本是一样的。

代码清单21-4 定义按钮按下时的shape drawable (res/drawable/button_beat_box_pressed.xml)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/red"/>

</shape>
```

接下来，要在按钮按下时使用这个新建的shape drawable。这需要用到state list drawable。

根据按钮的状态，state list drawable可以切换指向不同的drawable。按钮没有按下的时候指向button_beat_box_normal，按下的时候就指向button_beat_box_pressed。

在drawable目录中，定义一个state list drawable，如代码清单21-5所示。

代码清单21-5 创建一个state list drawable (res/drawable/button_beat_box.xml)

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
          android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal" />
</selector>
```

现在，在styles.xml中修改按钮样式，改用button_beat_box作为按钮背景，如代码清单21-6所示。

代码清单21-6 使用state list drawable (res/values/styles.xml)

```
<resources>

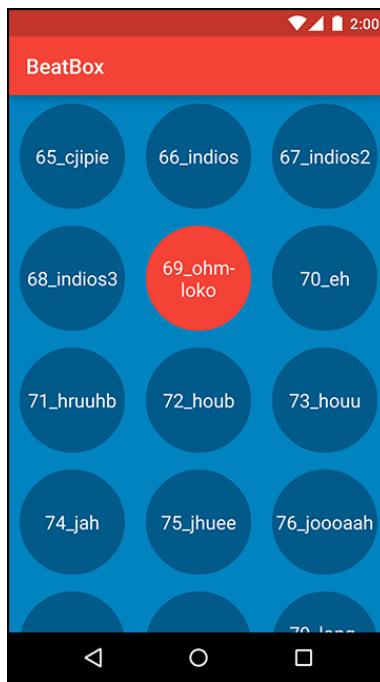
    <style name="AppTheme" parent="Theme.AppCompat">
        ...
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@drawable/button_beat_box_normal</item>
        <item name="android:background">@drawable/button_beat_box</item>
    </style>

</resources>
```

按钮没有按下的时候使用button_beat_box_normal作背景，按下时就使用button_beat_box_pressed作背景。

运行BeatBox应用。查看按下状态的按钮背景，如图21-4所示。



21

图21-4 按下状态的按钮

除了按下状态，state list drawable还支持禁用、聚焦以及激活等状态。若想详细了解，请访问网页：<http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList>。

21.4 layer list drawable

BeatBox应用看起来挺不错了。按钮圆圆的，按下时还有视觉反馈。不过，还要精益求精。

layer list drawable能让两个XML drawable合二为一。借助这个工具，可以为按下状态的按钮添加一个深色的圆环，如代码清单21-7所示。

代码清单21-7 使用layer list drawable (res/drawable/button_beat_box_pressed.xml)

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <shape
            android:shape="oval">

            <solid
                android:color="@color/red"/>
        </shape>
    </item>
    <item>
        <shape
            android:shape="oval">
```

```
<stroke  
    android:width="4dp"  
    android:color="@color/dark_red"/>  
  
</shape>  
</item>  
</layer-list>
```

现在，layer list drawable中指定了两个drawable。第一个是和以前一样的红圈。第二个则会绘制在第一个圈上，它定义了一个4dp粗的深红圈。

这两个drawable可以组成一个layer list drawable。多个当然也可以，会获得一些更复杂的效果。

运行BeatBox应用，随意点击几个按钮。可以看到，在按下状态，按钮有了漂亮的边圈，如图21-5所示。

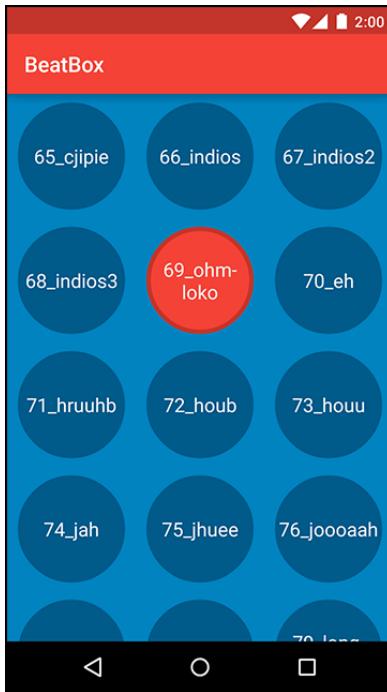


图21-5 最终版BeatBox

BeatBox应用真正完成了。还记得应用最初的样子吗？两相对比，犹如天壤之别。实践证明，精美的应用让人用起来舒心，容易获得用户的青睐。

21.5 深入学习：为什么要用 XML drawable

按钮状态切换总是需要的，所以state list drawable是Android开发不可或缺的工具。那shape

drawable和layer list drawable呢？它们有什么优势？

XML drawable用起来方便灵活，不仅用法多样，还易于更新维护。搭配使用shape drawable和layer list drawable可以做出复杂的背景图，连图像编辑器都省了。更改BeatBox应用的配色更是简单，直接修改XML drawable中的颜色就行了。

另外，XML drawable独立于屏幕像素密度，它们直接定义在drawable目录中，不需要加屏幕密度资源修饰符。如果是普通图像，就需要准备多个版本，以适配不同屏幕像素密度的设备；而XML drawable只要定义一次，就能在任何设备的屏幕上表现出色。

21.6 深入学习：使用 9-patch 图像

有时候（也可能经常），按钮背景图必须用到普通图片。那么，如果按钮需要以不同尺寸显示，背景图该如何变化呢？

如果按钮的宽度大于背景图的宽度，图片会被拉伸。拉伸的图片会有很好的效果吗？

朝一个方向拉伸背景图很可能让图片失去原样，所以得想个办法控制图片拉伸方式。

在本节中，BeatBox应用按钮要改用9-patch图片做背景。首先，修改list_item_sound.xml文件，允许按钮随屏幕大小动态调整，如代码清单21-8所示。

代码清单21-8 允许拉伸按钮 (res/layout/list_item_sound.xml)

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_margin="8dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <Button
        android:id="@+id/list_item_sound_button"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_gravity="center"
        tools:text="Sound name"/>
</FrameLayout>
```

调整后，按钮会使用多余空间，按钮的间隔还是8dp。新按钮背景图有个折角和阴影，如图21-6所示。



图21-6 新背景图 (res/drawable-xxhdpi/ic_button_beat_box_default.png)

在随书文件的21_XMLDrawables/BeatBox/app/src/main/res/drawable-xxhdpi目录下，找到包括按下状态在内的两个新背景图，复制到BeatBox项目的drawable-xxhdpi目录中。然后修改button_beat_box.xml文件使用它们，如代码清单21-9所示。

代码清单21-9 使用新背景图（res/drawable/button_beat_box.xml）

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/ic_button_beat_box_pressed"
          android:state_pressed="true"/>
    <item android:drawable="@drawable/ic_button_beat_box_default" />
</selector>
```

运行应用，查看按钮显示效果，如图21-7所示。



图21-7 难看的背景图

呃，这也太丑了吧！

为什么这么丑？原来，Android向四面拉伸了ic_beat_box_button.png，包括折边和圆角。要是能控制该拉伸的部分拉伸，不该拉伸的不拉伸就好了。

我们可以使用9-patch图像解决问题。9-patch图像是一种特别处理过的文件，让Android知道

图像的哪些部分可以拉伸，哪些不可以。只要处理得当，就能确保背景图的边角与原始图像保持一致。

为什么要叫作9-patch呢？9-patch图像分成 3×3 的网格，即由9部分或9 patch组成的网格。网格角落部分不会被缩放，边缘部分的4个patch只按一个维度缩放，而中间部分则按两个维度缩放，如图21-8所示。

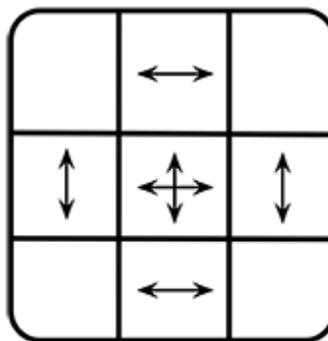


图21-8 9-patch原理

9-patch图像和普通PNG图像十分相似，只有两处不同：9-patch图像文件名以.**.png**结尾，图像边缘具有1像素宽度的边框。这个边框用以指定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。

任意图形编辑器都可用来创建9-patch图像，但Android SDK自带的**draw9patch**工具用起来更方便。（本书编写时，Android Studio的9-patch编辑器还不太好用。）该工具位于SDK安装目录下的**tools**目录内。

可以把两张新背景图转换为9-patch图像。在项目工具窗口中，右键单击**ic_button_beat_box_default.png**，选择**Refactor → Rename...**菜单项将其改名为：**ic_button_beat_box_default.9.png**。接着用相同的步骤得到另一个文件：**ic_button_beat_box_pressed.9.png**。

然后，双击默认图片在Android Studio内置的9-patch工具中打开，如图21-9所示。（如果Android Studio没能顺利打开9-patch编辑器，请先关闭图片文件，并在项目工具窗口中展开**drawable**目录，再尝试重新打开它。）

在9-patch工具中，把图像顶部和左边框填充为黑色，以标记图像的可伸缩区域。

标记的两条黑线告诉Android，不要拉伸图像的右上区域和四个角。重复上述步骤处理好另一个版本的图像。

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该如何处理呢？它们定义了9-patch图像的可选**drawable**区域。**drawable**区域是绘制内容（通常是文字）的地方。如果不标记**drawable**区域，则其默认与可拉伸区域保持一致。要的就是这个效果，因此无需定义**drawable**区域。

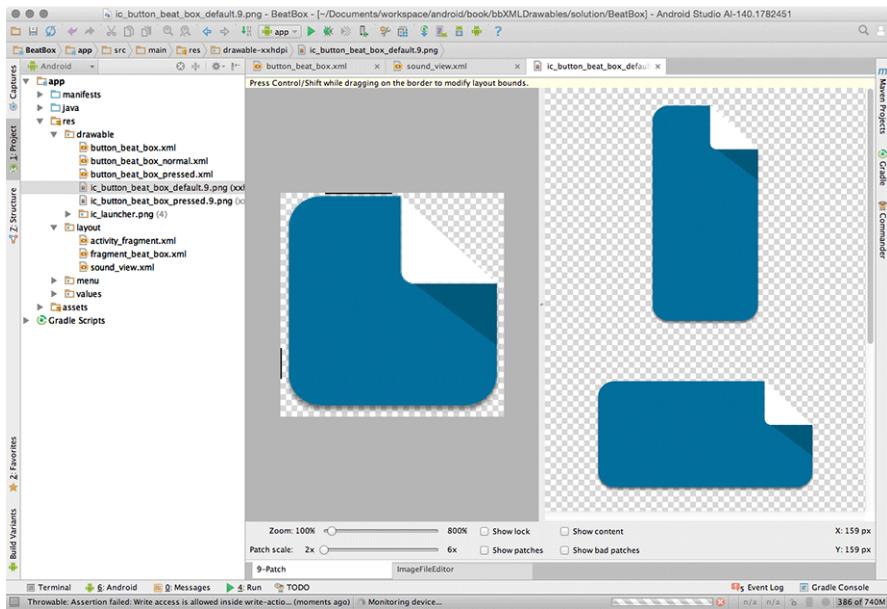


图21-9 创建9-patch图像

运行BeatBox应用。查看9-patch图像的使用效果，如图21-10所示。



图21-10 优化过的BeatBox应用

试着横屏查看应用。可以看到，图像拉伸得更厉害了，不过按钮背景图的效果依然不错。

21.7 深入学习：使用 Mipmap 图像

资源修饰符和drawable用起来都很方便。应用要用到图像，就针对不同的设备尺寸准备不同尺寸的图片，再分别放入drawable-mdpi和drawable-hdpi这样的文件夹。然后，按名字引用它们。剩下的就交给Android了，它会根据当前设备的屏幕密度调用相应的图片。

但是，有个问题不得不提。发布到Google应用商店的APK文件包含了项目drawable目录里的所有图片，哪怕是从来不会用到的图片。这是个负担。

为减轻负担，有人想到针对设备定制APK，比如mdpi APK一个，hdpi APK一个，等等。（有关APK分包的详细信息，可参阅工具文档网页：<http://tools.android.com/tech-docs/new-build-system/user-guide/apk-splits>。）

但问题解决得不够彻底。假如想保留各个屏幕像素密度的启动图标呢？

Android启动器是个常驻主屏幕的应用（详见第22章）。按下设备的主屏幕键，会回到启动器应用界面。

有些新版启动器会显示大尺寸应用图标。想让大图标清晰好看，启动器就需要使用更高分辨率的图标。对于hdpi设备，要显示大图标，启动器就会使用xhdpi图标。

找不到的话，就只能使用低分辨率的图标。可想而知，放大拉伸后的图标肯定很糟。

Android的终极解决之道是使用mipmap目录。本书编写时，Android Studio中的新项目已经可以使用mipmap资源作为应用的启动图标了，如图21-11所示。

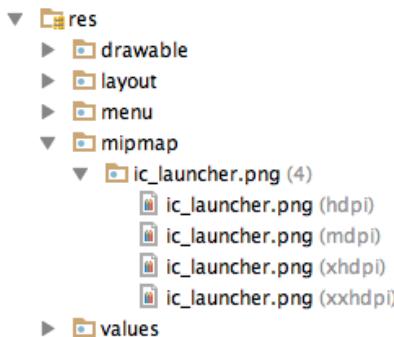


图21-11 mipmap图标

APK分包时，mipmap资源会全部包含在APK文件中。要一劳永逸，推荐的做法就是，把应用启动器图标放在mipmap目录中，其他图片都放在drawable目录中。

第 22 章

深入学习intent和任务

22

本章，我们会使用隐式intent创建一个替换Android默认启动器的应用。这个新建应用名为NerdLauncher，运行界面如图22-1所示。

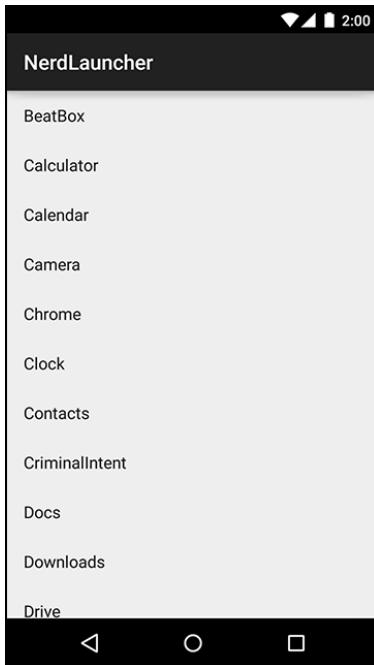


图22-1 NerdLauncher应用最终效果

NerdLauncher应用能列出设备上的其他应用。点选任意列表项会启动相应应用。

完成该应用可帮我们深入理解intent、intent过滤器以及Android应用间的交互。

22.1 创建 NerdLauncher 项目

创建一个新的Android应用项目，名为NerdLauncher。选择Phone and Tablet作为目标设备，最

低SDK版本为API 16: Android 4.1 (Jelly Bean)。新建名为NerdLauncherActivity的空activity。

NerdLauncherActivity需要托管fragment，因而它也应继承SingleFragmentActivity类。找到CriminalIntent项目中的SingleFragmentActivity.java和activity_fragment.xml文件，并把它们复制到NerdLauncher应用项目中备用。

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22-1所示。（createFragment()方法会报错。暂时忽略，稍后会创建NerdLauncherFragment类解决。）

代码清单22-1 另一个SingleFragmentActivity (NerdLauncherActivity.java)

```
public class NerdLauncherActivity extends SingleFragmentActivityAppCompatActivity {

    @Override
    protected Fragment createFragment() {
        return NerdLauncherFragment.newInstance();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Auto-generated template code... */
    }
}
```

按照第9章所述方法，添加RecyclerView依赖项，因为NerdLauncherFragment需要使用它显示应用列表。

为创建fragment布局，重命名layout/activity_nerd_launcher.xml为layout/fragment_nerd_launcher.xml，然后用图22-2中的RecyclerView替换原布局内容。

```
    android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_nerd_launcher_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

图22-2 创建NerdLauncherFragment布局 (layout/fragment_nerd_launcher.xml)

最后，以`android.support.v4.app.Fragment`为父类，创建一个名为NerdLauncherFragment的新类。在新建类中，新增`newInstance()`方法，覆盖`onCreateView(...)`方法。在覆盖方法中，将`RecyclerView`对象存放在`mRecyclerView`成员变量中，如代码清单22-2所示。（稍后会处理`RecyclerView`的数据绑定。）

代码清单22-2 NerdLauncherFragment初始类 (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
```

```
private RecyclerView mRecyclerView;

public static NerdLauncherFragment newInstance() {
    return new NerdLauncherFragment();
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_nerd_launcher, container, false);
    mRecyclerView = (RecyclerView) v
        .findViewById(R.id.fragment_nerd_launcher_recycler_view);
    mRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

    return v;
}
}
```

运行应用。一切正常的话，可看到如图22-3所示的用户界面。RecyclerView尚未绑定数据，现在还无法看到应用列表。

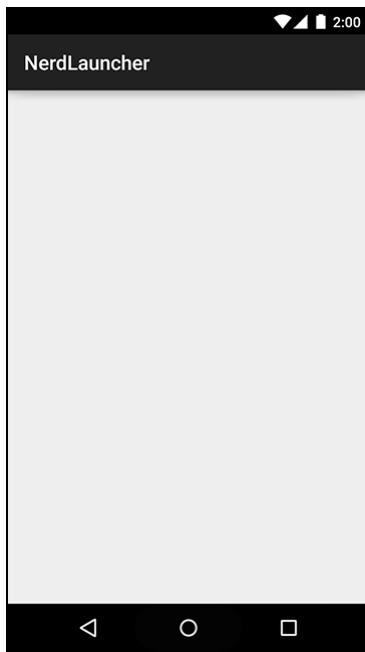


图22-3 NerdLauncher应用初始界面

22.2 解析隐式 intent

NerdLauncher应用能以列表的形式展示设备上的可启动应用。(可启动应用是指用户点击主屏幕或启动器界面上的图标就能打开的应用。)要实现该功能，它会使用PackageManager获取所有可启动主activity。可启动主activity都带有包含MAIN操作和LAUNCHER类别的intent过滤器。在之前项目的AndroidManifest.xml文件中，我们已见过这种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，新增setupAdapter()方法，然后在onCreateView(...)方法中调用它。(该方法最终还会创建RecyclerView.Adapter实例并设置给RecyclerView对象。) setupAdapter()方法负责创建隐式intent并从PackageManager那里获取匹配它的所有activity。最后，日志记录PackageManager返回的activity总数，如代码清单22-3所示。

代码清单22-3 向PackageManager查询activity总数 (NerdLauncherFragment.java)

```
public class NerdLauncherFragment extends Fragment {
    private static final String TAG = "NerdLauncherFragment";

    private RecyclerView mRecyclerView;

    public static NerdLauncherFragment newInstance() {
        return new NerdLauncherFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
        setupAdapter();
        return v;
    }

    private void setupAdapter() {
        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, "Found " + activities.size() + " activities.");
    }
}
```

运行NerdLauncher应用，在LogCat日志窗口查看PackageManager返回的activity数目。(我们第一次尝试时得到了42个。)

在CriminalIntent应用中，为使用隐式intent发送crime报告，我们先创建隐式intent，再将其封装在选择器intent中，最后调用startActivity(Intent)方法发送给操作系统：

```
Intent i = new Intent(Intent.ACTION_SEND);
... // Create and put intent extras
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
```

这里没有使用类似的处理方式，是不是很费解？原因很简单：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。

事实上，startActivity(Intent)方法意味着“启动匹配隐式intent的默认activity”，而不是想当然的“启动匹配隐式intent的activity”。调用startActivity(Intent)方法（或startActivityForResult(...)方法）发送隐式intent时，操作系统会悄悄为目标intent添加Intent.CATEGORY_DEFAULT类别。

因此，如果希望intent过滤器匹配startActivity(...)方法发送的隐式intent，就必须在对应的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。它只负责做好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为默认的主要入口点，所以可以不包含CATEGORY_DEFAULT类别。

前面说过，MAIN/LAUNCHER intent过滤器并不一定包含CATEGORY_DEFAULT类别，因此不能保证可以与startActivity(...)方法发送的隐式intent匹配。所以，我们转而使用intent直接向PackageManager查询带有MAIN/LAUNCHER intent过滤器的activity。

接下来，需要在NerdLauncherFragment的RecyclerView视图中显示查询到的activity标签。activity标签是用户可以识别的展示名称。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标签和其他一些元数据。

首先，使用ResolveInfo.loadLabel(...)方法，对ResolveInfo对象中的activity标签按首字母排序，如代码清单22-4所示。

代码清单22-4 对activity标签排序（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private void setupAdapter() {
        ...
        List<ResolveInfo> activities = pm.queryIntentActivities(startupIntent, 0);
        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());
            }
        });
    }
}
```

```

    });
    Log.i(TAG, "Found " + activities.size() + " activities.");
}
}

```

然后，定义一个ViewHolder用来显示activity标签名。另外，ResolveInfo信息需经常使用，这里使用成员变量存储它，如代码清单22-5所示。

代码清单22-5 实现ViewHolder（NerdLauncherFragment.java）

```

public class NerdLauncherFragment extends Fragment {
    ...

    private void setupAdapter() {
        ...

        private class ActivityHolder extends RecyclerView.ViewHolder {
            private ResolveInfo mResolveInfo;
            private TextView mNameTextView;

            public ActivityHolder(View itemView) {
                super(itemView);
                mNameTextView = (TextView) itemView;
            }

            public void bindActivity(ResolveInfo resolveInfo) {
                mResolveInfo = resolveInfo;
                PackageManager pm = getActivity().getPackageManager();
                String appName = mResolveInfo.loadLabel(pm).toString();
                mNameTextView.setText(appName);
            }
        }
    }
}

```

22

接下来参照代码清单22-6实现RecyclerView.Adapter。

代码清单22-6 实现RecyclerView.Adapter（NerdLauncherFragment.java）

```

public class NerdLauncherFragment extends Fragment {
    ...

    private class ActivityHolder extends RecyclerView.ViewHolder {
        ...
    }

    private class ActivityAdapter extends RecyclerView.Adapter<ActivityHolder> {
        private final List<ResolveInfo> mActivities;

        public ActivityAdapter(List<ResolveInfo> activities) {
            mActivities = activities;
        }

        @Override

```

```

public ActivityHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    LayoutInflater layoutInflater = LayoutInflater.from(getActivity());
    View view = layoutInflater
        .inflate(android.R.layout.simple_list_item_1, parent, false);
    return new ActivityHolder(view);
}

@Override
public void onBindViewHolder(ActivityHolder activityHolder, int position) {
    ResolveInfo resolveInfo = mActivities.get(position);
    activityHolder.bindActivity(resolveInfo);
}

@Override
public int getItemCount() {
    return mActivities.size();
}
}
}
}

```

最后，更新setupAdapter()方法创建ActivityAdapter实例并为RecyclerView设置adapter，如代码清单22-7所示。

代码清单22-7 为RecyclerView设置adapter (NerdLauncherFragment.java)

```

public class NerdLauncherFragment extends Fragment {
    ...
    ...

    private void setupAdapter() {
        ...
        Log.i(TAG, "Found " + activities.size() + " activities.");
        mRecyclerView.setAdapter(new ActivityAdapter(activities));
    }

    ...
}

```

运行NerdLauncher应用。现在应该可以看到显示了activity标签的RecyclerView视图，如图22-4所示。

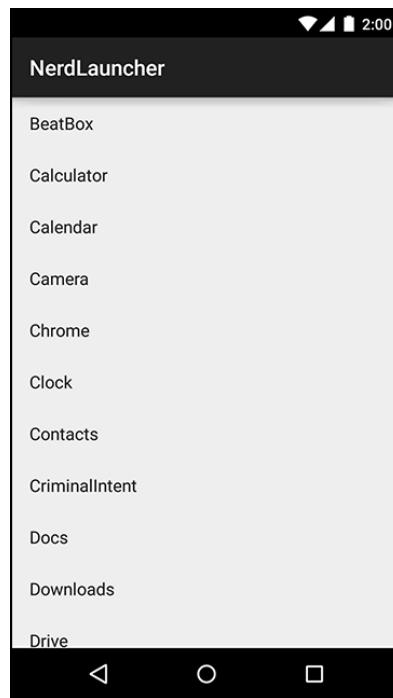


图22-4 设备上的全部activity

22.3 在运行时创建显式 intent

上一节，我们使用隐式intent获取目标activity并以列表的形式展示。接下来要实现用户点击任一列表项时，启动对应的activity。我们需使用显式intent来启动activity。

要创建启动activity的显式intent，需要从ResolveInfo对象中获取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：<http://developer.android.com/reference/android/content/pm/ResolveInfo.html>。）

更新ActivityHolder类实施一个点击监听器。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22-8所示。

代码清单22-8 启动目标activity（NerdLauncherFragment.java）

```
...
private class ActivityHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    private ResolveInfo mResolveInfo;
    private TextView mNameTextView;
```

```

public ActivityHolder(View itemView) {
    super(itemView);
    mNameTextView = (TextView) itemView;
    mNameTextView.setOnClickListener(this);
}

public void bindActivity(ResolveInfo resolveInfo) {
    ...
}

@Override
public void onClick(View v) {
    ActivityInfo activityInfo = mResolveInfo.activityInfo;

    Intent i = new Intent(Intent.ACTION_MAIN)
        .setClassName(activityInfo.applicationInfo.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22-8中，使用包名和类名创建显式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。之前，我们使用的是接受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ComponentName，然后使用下面的Intent方法创建显式intent：

```
public Intent setComponent(ComponentName component)
```

不过，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较少。

运行NerdLauncher应用并尝试启动一些应用。

22.4 任务与后退栈

在所有运行的应用中，Android都使用任务来跟踪用户的状态。通过Android默认启动器应用打开的应用都有自己的任务。然而，这并不适用于NerdLaucher应用。在NerdLaucher应用中启动的应用怎样获得这样的行为呢？我们首先要搞清楚究竟什么是任务，它是如何工作的。

任务是用户比较关心的activity栈。栈底部的activity通常称为基activity。用户可以看到栈顶的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，系统会退回主屏幕。

默认情况下，新activity都在当前任务中启动。在CriminalIntent应用中，无论何时启动新activity，它都会被添加到当前任务中，如图22-5所示。即使要启动的activity不属于CriminalIntent应用，它同样也在当前任务中启动。启动activity发送crime报告就是这样的一个例子。

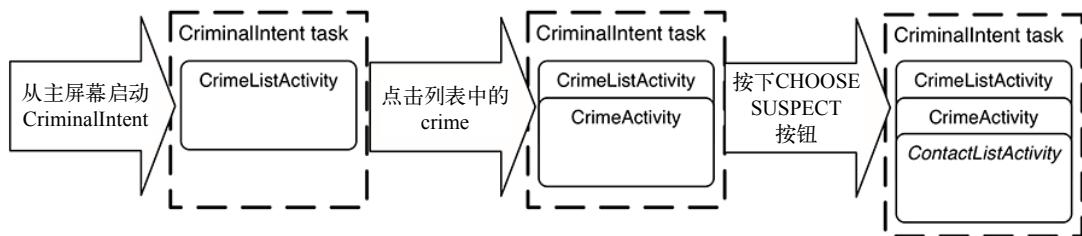


图22-5 CriminalIntent中的任务

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，如图22-6所示。

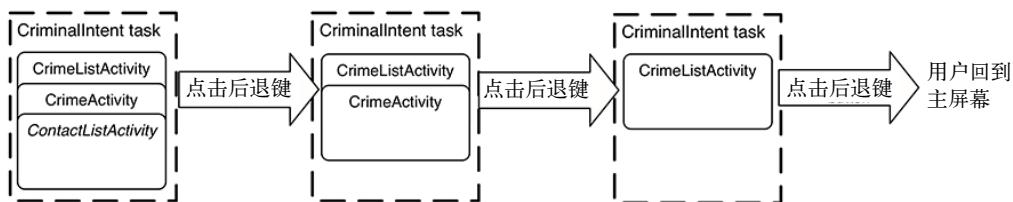


图22-6 点击后退键

22.4.1 在任务间切换

在不影响各个任务状态的情况下，overview screen可以让我们在任务间切换。例如，如果进入联系人应用，然后切换到Twitter应用查看信息，这时我们就启动了两个任务。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都会被保存下来。

(overview screen还有别的叫法，常听到的有任务管理器、最近使用屏、最近使用应用屏，以

及最近任务列表。)

我们可以在设备或模拟器上测试overview screen。首先，从主屏幕或应用启动器中启动CriminalIntent应用。(如果你的设备或模拟器已经卸载了CriminalIntent应用，可以打开Android Studio中的CriminalIntent项目并运行。)从crime列表中选择任意列表项，然后，点击主屏幕键回到主屏幕。接着，从主屏幕或应用启动器中启动BeatBox应用。

现在打开overview screen。具体打开方式因设备而异。如果设备有Recents按钮，就直接点击这个按钮。(Recents按钮看上去像个正方形，或是两个重叠的长方形，通常出现在导航栏的最右边，如图22-7所示。)如果没有的话，就长按或双击主屏幕键。

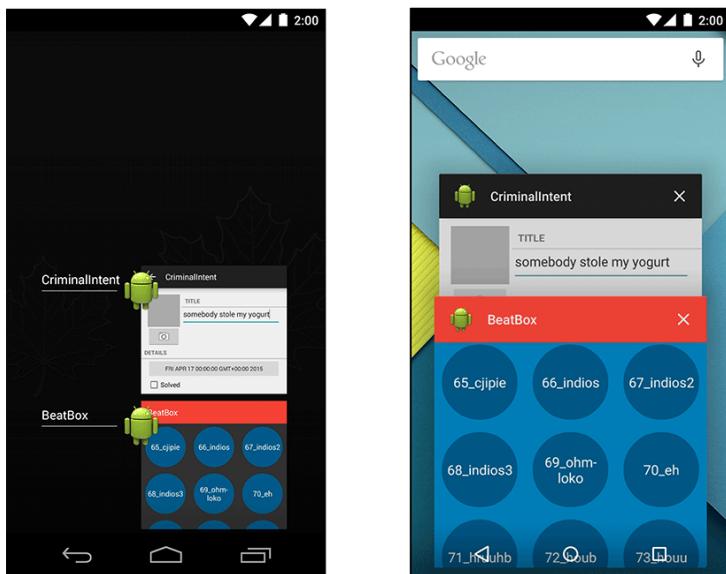


图22-7 overview screen的不同版本

如果是KitKat设备，用户会看到图22-7左边的overview screen；如果是Lollipop，会看到右边的overview screen。不管怎样，图中的每个应用显示项（就是Lollipop系统所说的卡片）就代表着一个应用任务。当前任务显示的是处于回退栈顶部activity快照。用户可以点击任意显示项切换至对应应用的当前activity。

要清除应用任务，用户只需滑动移除卡片即可。清除任务就是从应用回退栈中清除所有activity。

试着清除CriminalIntent应用任务并重启应用。重启后，我们看到的是crime列表界面，而不应再是清除前的crime编辑界面了。

22.4.2 启动新任务

我们有时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。

当前，从NerdLauncher启动的任何activity都会添加到NerdLauncher任务中，如图22-8所示。

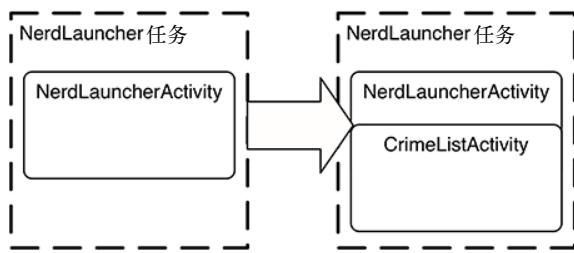


图22-8 NerdLauncher任务中包含CriminalIntent应用activity

要想验证的话，可先清除overview screen显示的所有任务。然后，启动NerdLauncher并点击CriminalIntent任务项启动CriminalIntent应用。再次打开overview screen界面时，应该看不到CriminalIntent任务了。CrimeListActivity启动后，它随即就添加到NerdLauncher任务中了，如图22-9所示。只要点击NerdLauncher任务，你就会回到启动overview screen界面之前所在的CriminalIntent用户界面。

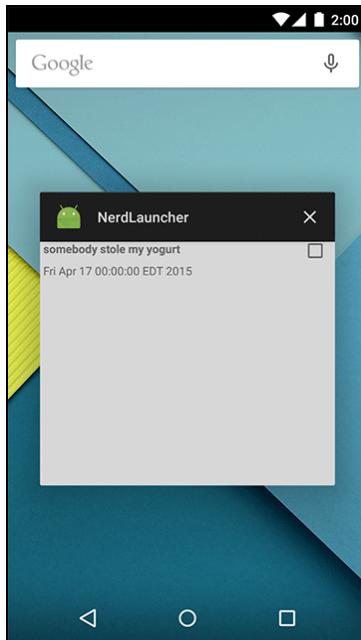


图22-9 CriminalIntent应用已不在自己的任务中

我们需要NerdLauncher在新任务中启动activity，如图22-10所示。这样，点击NerdLauncher启动器中的应用列表项可以让应用拥有自己的任务，用户因此可以在运行的应用间自由切换。

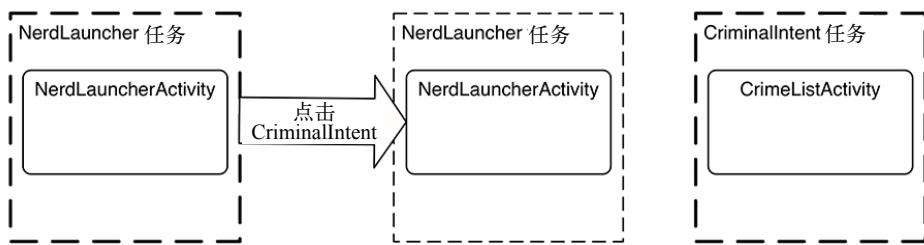


图22-10 让CriminalIntent在自身任务里启动

为了在启动新activity时启动新任务，需要为intent添加一个标志，如代码清单22-9所示。

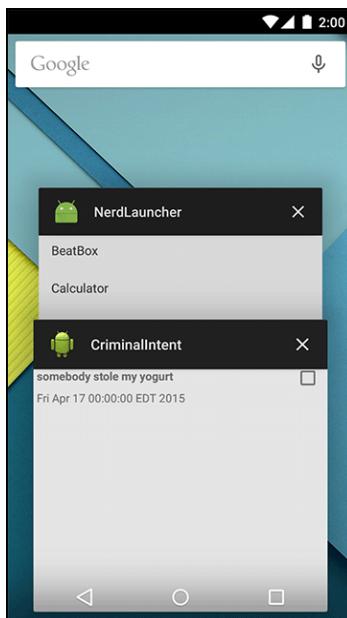
代码清单22-9 为intent添加新任务标志（NerdLauncherFragment.java）

```
public class NerdLauncherFragment extends Fragment {
    ...
    private class ActivityHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
        ...
        @Override
        public void onClick(View v) {
            ...
            Intent i = new Intent(Intent.ACTION_MAIN)
                .setClassName(activityInfo.applicationInfo.packageName,
                    activityInfo.name)
                .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(i);
        }
    }
    ...
}
```

先清除overview screen显示的所有任务，再运行NerdLauncher应用并启动CriminalIntent。这次，如果启动overview screen，就会看到CriminalIntent应用处于一个单独的任务中，如图22-11所示。

如果从NerdLauncher应用中再次启动CriminalIntent应用，不会创建第二个CriminalIntent任务。`FLAG_ACTIVITY_NEW_TASK`标志控制每个activity仅创建一个任务。`CrimeListActivity`已经有了一个运行的任务，因此Android会自动切换到原来的任务，而不是创建全新的任务。

眼见为实。在CriminalIntent应用中，打开任意crime的明细界面。然后，使用overview screen切换至NerdLauncher。点击应用列表中的CriminalIntent。可以看到，我们又回到了CriminalIntent应用中打开的crime明细界面。



22

图22-11 CriminalIntent应用处于独立的任务中

22.5 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22-10向intent主过滤器添加以下节点定义。

代码清单22-10 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。点击主屏幕键可以看到，在弹出的界面选择对话框中，NerdLauncher变成了主界面可选项，如图22-12所示。

（如果已设置NerdLauncher应用为主界面，恢复系统默认设置也很容易。首先，从NerdLauncher启动Settings应用。如果是Lollipop系统，可选择Settings → Apps菜单项，然后从应用列表中选择NerdLauncher。如果是Lollipop之前的系统，可以选择Settings → Applications → Manage Applications菜单项，找到NerdLauncher应用。选择了NerdLauncher后，在应用的信息屏，滚动到Launch by

default并点击CLEAR DEFAULTS按钮。完成后，下次再按主屏幕键时，就可以选择自己想要的默认主界面了。)

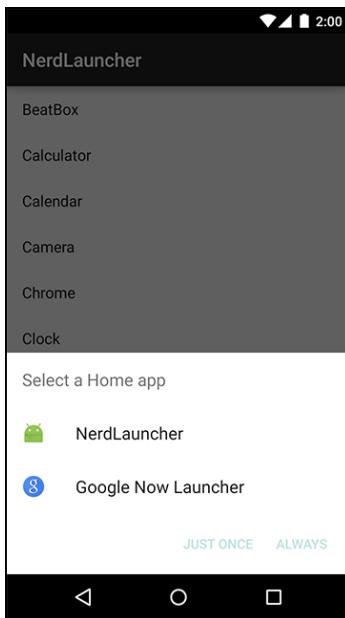


图22-12 选择主屏幕应用

22.6 挑战练习：应用图标

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。

22.7 深入学习：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虚拟机。

尽管存在未知的异常情况，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相似之处。任务只包含activity，这些activity通常来自于不同的应用；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，而且通常都是以其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overview screen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。以CriminalIntent应用和联系人应用为例，看看以下具体场景就会明白了。

打开CriminalIntent应用，选择任何crime项，然后点击CHOOSE SUSPECT按钮。这会打开联系人应用让我们选择目标联系人。随即，联系人activity会被加入CriminalIntent应用任务。如果此时点击后退键在不同activity间切换的话，用户可能意识不到他们正在进程间切换。

然而，联系人activity实例确实在联系人应用进程的内存空间创建的，而且也是在该应用进程里的虚拟机上运行的。这可以从图22-13中看出。

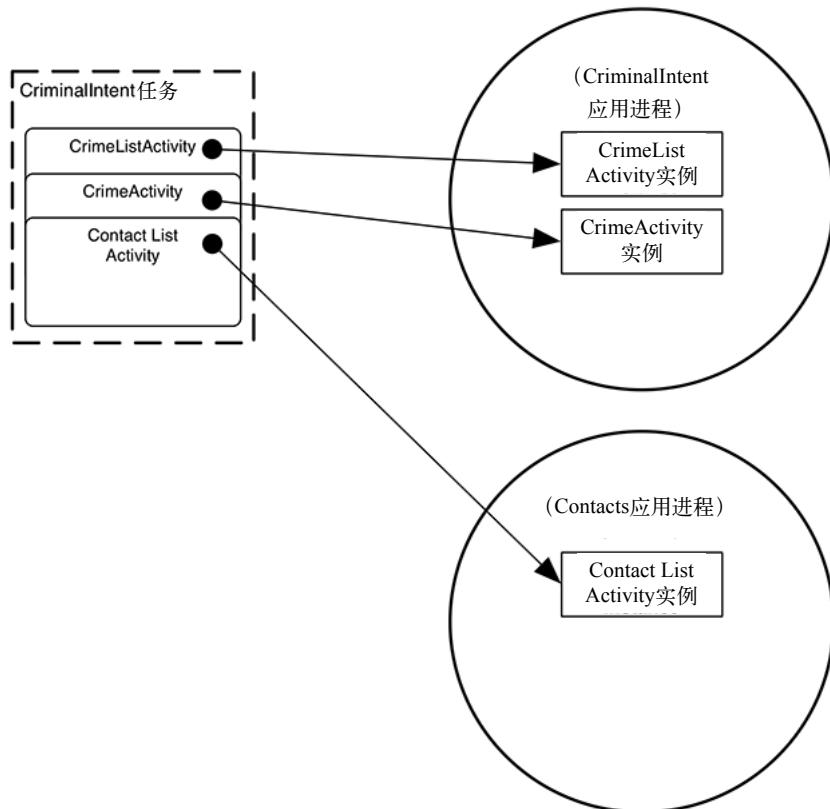


图22-13 任务与进程

为进一步了解进程和任务的概念，让CriminalIntent应用处于运行状态，并打开联系人列表界面。（继续之前，请确保联系人应用没有在overview screen出现。）点击主屏幕键回到主屏幕，从中启动联系人应用。然后从联系人列表选取任意联系人，或添加新的联系人。

在这个操作过程中，会在联系人应用进程中创建新的联系人列表activity和联系人明细界面实例。联系人应用新任务也会完成创建。这个新任务会引用联系人列表activity和联系人明细界面实例，如图22-14所示。

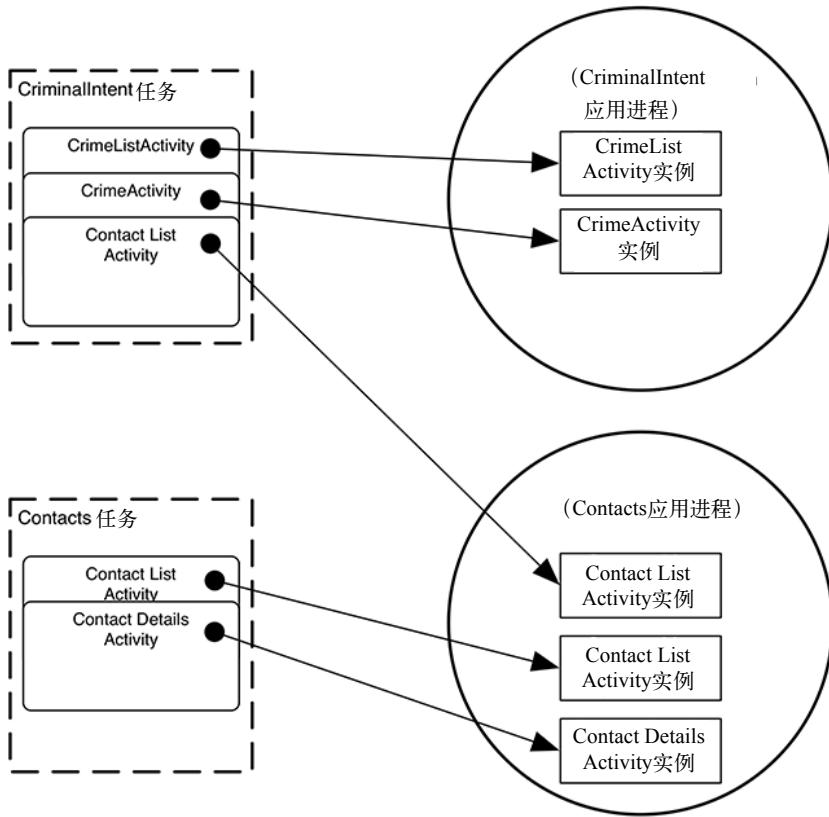


图22-14 任务与进程

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overview screen呢？很遗憾，Android没有提供任何处理它们的方法。虽然长按主屏幕键可以硬链接到默认的overview screen，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。

22.8 深入学习：并发文档

如果是在Lollipop设备上运行CriminalIntent应用的话，打开overview screen查看任务时，你会

发现一些有趣的现象。例如，在发送crime消息时，你所选择发送消息应用的activity不会添加到CriminalIntent应用任务中，而是添加到它自己的独立任务中，如图22-15所示。

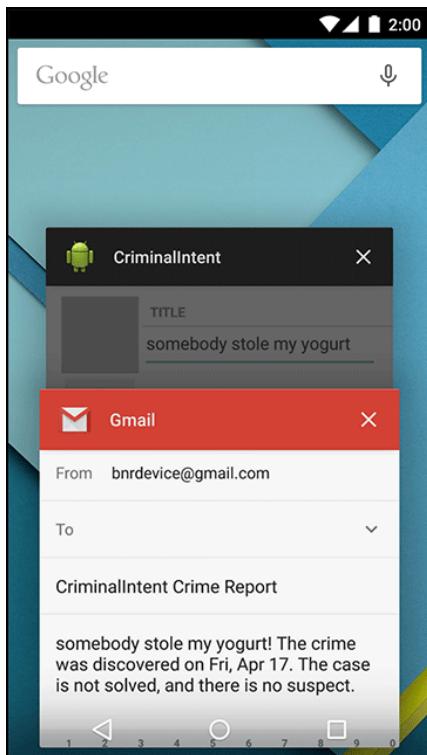


图22-15 Gmail处于独立的任务中

在Lollipop设备上，对以`android.intent.action.SEND`或`action.intent.action.ACTION_SEND_MULTIPLE`操作启动的activity，隐式intent选择器会创建独立的新任务。（在旧设备上，Gmail的activity是直接添加给CriminalIntent应用任务的。）

这种现象要归因于Lollipop中叫作并发文档（concurrent documents）的新概念。有了并发文档，我们就可以在应用运行时动态创建任意数目的任务。在Lollipop之前，应用任务只能预先定义好，而且还要在manifest文件中明确指定。

实践中，Google Drive应用就是并发文档的最好实例。用户可以用其打开并编辑多份文档。这些文档编辑activity都处在独立的任务中，如图22-16所示。在Lollipop之前的设备上查看overview screen的话，你只能看到孤零零的一个任务。前面已说过，Lollipop之前的系统需要在manifest中提前定义应用任务，所以系统无法为单个应用动态创建多个任务。

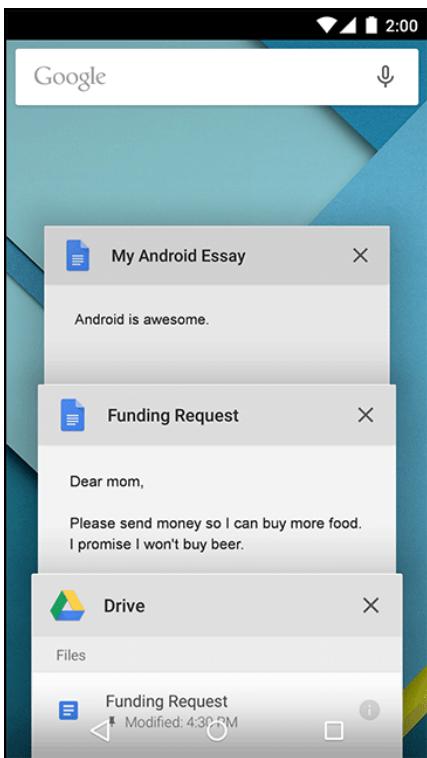


图22-16 Lollipop设备上的多个Google Drive任务

在Lollipop设备上，如果需要应用启动多个任务，可采用两种方式：给intent打上Intent.FLAG_ACTIVITY_NEW_DOCUMENT标签，再调用startActivity(...)方法；或者在manifest文件中，为activity设置如下documentLaunchMode：

```
<activity
    android:name=".CrimePagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".CrimeListActivity"
    android:documentLaunchMode="intoExisting" />
```

使用上述方法，一份文档只会对应一个任务。（如果发送带有和已存在任务相同数据的intent，系统就不会再创建新任务。）如果无论如何都想创建新任务，那就给intent同时打上Intent.FLAG_ACTIVITY_NEW_DOCUMENT和Intent.FLAG_ACTIVITY_MULTIPLE_TASK标签，或者把manifest文件中的documentLaunchMode属性值修改为always。

想要深入了解overview screen或者想知道Lollipop系统相较旧系统有哪些变化，请访问网页<https://developer.android.com/guide/components/recents.html>。

第 23 章

HTTP与后台任务

23

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，或是玩网络游戏。

为着手学习Android网络应用的开发，我们来创建一个名为PhotoGallery的应用。PhotoGallery是图片共享网站Flickr的客户端应用，它能获取并展示Flickr网站的最新公共图片。应用的最终运行效果如图23-1所示。

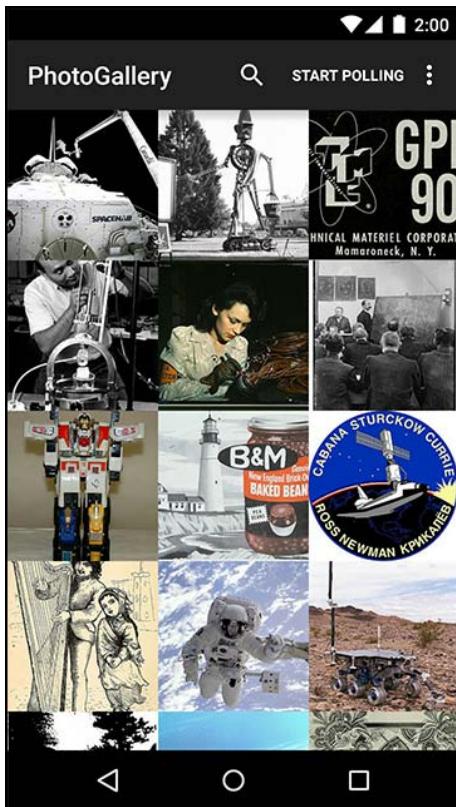


图23-1 PhotoGallery应用最终效果图

(PhotoGallery应用有过滤功能，只能展示无版权限制图片。可访问网址`https://www.flickr.com/commons/usage/`，进一步了解非限制使用图片。Flickr网站上有很多图片归上传者私有，使用它们需遵守使用许可限制条款。可访问网址`https://www.flickr.comcreativecommons/`，了解更多有关第三方内容的使用权限问题。)

接下来的六章都会用来学习开发PhotoGallery应用。前两章介绍网络下载、JSON文件解析、图像显示等基本知识。随后的几章里，会为应用添加一些特色功能，并借此介绍搜索、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。本章结束时，我们应完成的任务是：获取、解析以及显示Flickr图片的标题，如图23-2所示。(第24章会介绍图片获取与显示的相关内容。)

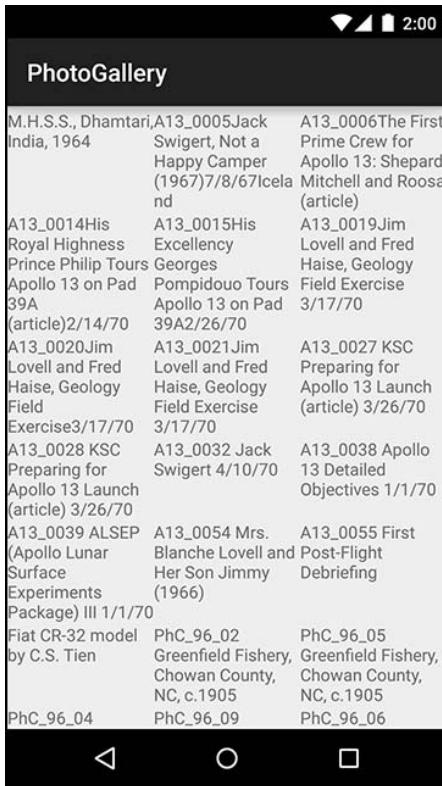
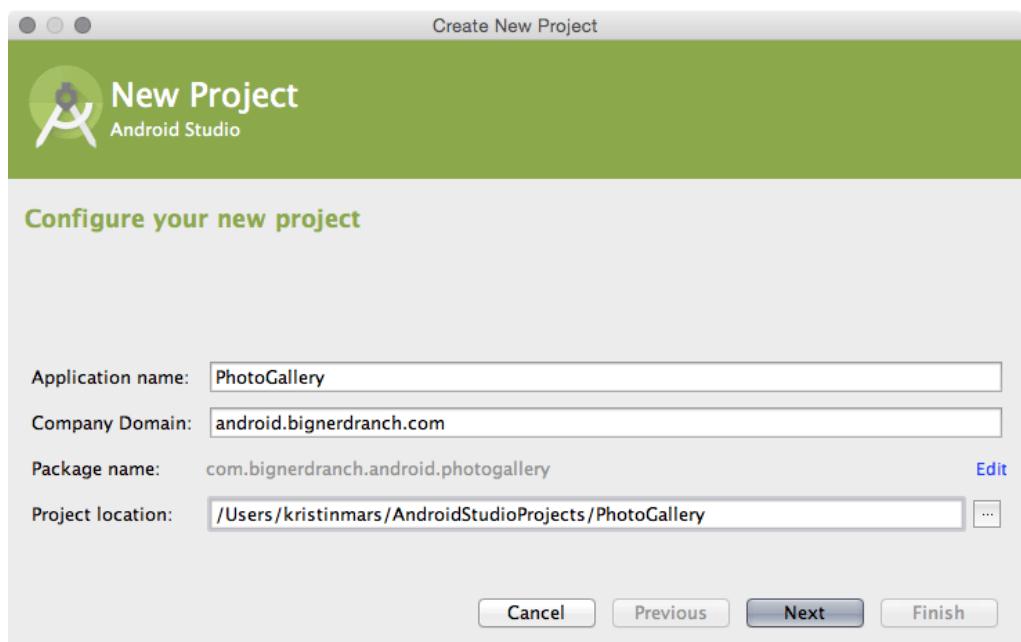


图23-2 本章结束时的应用效果图

23.1 创建 PhotoGallery 应用

按照图23-3所示的配置，创建一个全新的Android应用项目（目标设备选Phone and Tablet，最

低SDK版本选API 16)。



23

图23-3 创建PhotoGallery应用

单击Next按钮，通过应用向导创建一个名为PhotoGalleryActivity的空activity。

PhotoGallery应用继续沿用前面一直使用的设计架构。PhotoGalleryActivity依然继承SingleFragmentActivity，其视图为activity_fragment.xml中定义的容器视图。PhotoGalleryActivity负责托管稍后会创建的PhotoGalleryFragment实例。

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，删除工具自动产生的模板代码。然后，让PhotoGalleryActivity继承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23-1所示。(不用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23-1 activity的调整 (PhotoGalleryActivity.java)

```
public class PhotoGalleryActivity extends Activity SingleFragmentActivity {

    @Override
    public Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }

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

```

    /* Auto-generated template code... */
}

}

```

PhotoGallery应用将在RecyclerView视图（借助内置的GridLayoutManager）中显示内容。

首先是添加RecyclerView依赖库。打开项目结构窗口，选择左边的app模块。再选择Dependencies选项页，单击+按钮。在随后出现的下拉菜单中选择Library dependency。最后，找到并选择recyclerview-v7后点击OK按钮完成。

为创建fragment布局，重命名layout/activity_photo_gallery.xml为layout/fragment_photo_gallery.xml。然后以图23-4所示的RecyclerView替换原有内容。

```

        android.support.v7.widget.RecyclerView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/fragment_photo_gallery_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

```

图23-4 RecyclerView视图（layout/fragment_photo_gallery.xml）

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RecyclerView视图，如代码清单23-2所示。

代码清单23-2 一些代码片断（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {

    private RecyclerView mPhotoRecyclerView;

    public static PhotoGalleryFragment newInstance() {
        return new PhotoGalleryFragment();
    }

    @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);

        mPhotoRecyclerView = (RecyclerView) v
            .findViewById(R.id.fragment_photo_gallery_recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        return v;
    }
}

```

```

    }
}

```

(知道为什么要保留fragment吗？自己先思考思考，答案会在23.7节揭晓。)

继续之前，试着运行PhotoGallery应用。一切正常的话，可以看到一个空白视图。

23.2 网络连接基本

PhotoGallery应用中，我们需要一个网络连接专用类。应用要访问的是Flickr网站，因此新建一个名为FlickrFetchr的新Java类。

FlickrFetchr类一开始只有getUrlBytes(String)和getUrlString(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。getUrlString(String)方法则将getUrlBytes(String)方法返回的结果转换为String。

在FlickrFetchr.java中，参照代码清单23-3，实现getUrlBytes(String)和getUrlString(String)方法。

23

代码清单23-3 基本网络连接代码 (FlickrFetchr.java)

```

public class FlickrFetchr {
    public 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) {
                throw new IOException(connection.getMessage() +
                    ": with " +
                    urlSpec);
            }

            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 getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}

```

在getUrlBytes(String)方法中，首先根据传入的字符串参数，如https://www.bignerdranch.com，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，但要连接的是http URL，因此需将其强制类型转换为HttpURLConnection对象。这让我们得以调用它的getInputStream()、getResponseCode()等方法。

虽然HttpURLConnection对象提供了一个连接，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。

创建了URL并打开网络连接之后，我们便可循环调用read()方法读取网络数据，直到取完为止。只要还有数据，InputStream类就会不断地输出字节流数据。数据全部取回后，关闭网络连接，并将读取的数据写入ByteArrayOutputStream字节数组中。

虽然最重要的数据获取任务要靠getUrlBytes(String)方法完成，但getUrlString(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。看到这里，可能有人会问，为什么不在一个方法中完成全部任务？当然可以，但是在下一章处理图像数据下载时，你就能体会两个独立方法的好处了。

获取网络使用权限

要连接网络，还需完成一件事：取得使用网络的权限。正如用户怕被偷拍一样，他们也不想应用偷偷下载图片。

要取得网络使用权限，参照代码清单23-4，在AndroidManifest.xml文件中添加下列权限。

代码清单23-4 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" >

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

    <application
        ...
    </application>

</manifest>
```

用户下载应用时（比如PhotoGallery），会看到一个注明需要连接网络权限的对话框，用户可以选择接受或拒绝。

Android权限管理还不够完善。为了使某个功能正常工作，应用可能确实需要某个权限，但用户安装这个应用时可能会觉得暂时用不着这个功能。而且，如果用户不接受某一特定权限，只能不安装或删除整个应用。

即将发布的Android M会解决上述问题。在M中，应用可以在需要时才申请权限，而不是安装时就申请。此外，用户还可以随时取消权限。

不过，实际开发时，也要学会灵活处理。处理不可或缺的权限（如PhotoGallery应用中的网络连接），还是使用原有方式：应用安装时就向用户申请。处理那些很少使用或不怎么重要的权限时，采用新的权限管理方式会更好。

23.3 使用 AsyncTask 在后台线程上运行代码

接下来调用并测试新添加的网络连接代码。注意，不要直接在PhotoGalleryFragment类中调用FlickrFetchr.getString(String)方法。正确的做法是，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日志，如代码清单23-5所示。

代码清单23-5 实现AsyncTask工具类方法，第一部分（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;

    ...

    private class FetchItemsTask extends AsyncTask<Void,Void(Void) {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr()
                    .getUrlString("https://www.bignerdranch.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23-6所示。

代码清单23-6 实现AsyncTask工具类方法，第二部分（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";
```

```

private RecyclerView mPhotoRecyclerView;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    new FetchItemsTask().execute();
}

...
}

```

调用`execute()`方法会启动`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，可以看到一大堆Big Nerd Ranch网站主页HTML代码，如图23-5所示。

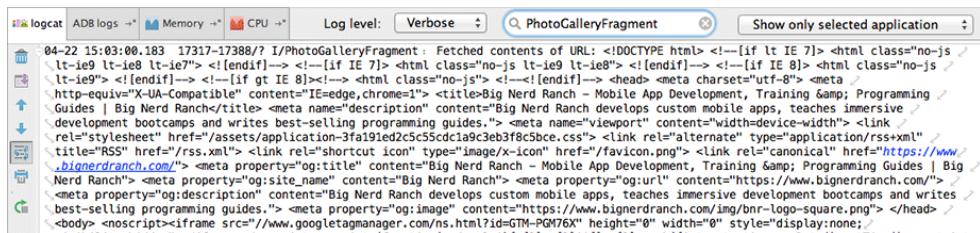


图23-5 LogCat中的HTML代码

LogCat窗口中的日志可不好找。要养成使用关键词搜索的好习惯。图23-5就是在LogCat搜索框中输入PhotoGalleryFragment后的查找结果。

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。

23.4 线程与主线程

网络连接需要时间。Web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。考虑到这个因素，Android禁止任何主线程网络连接行为。即使强行为之，Android也会抛出`NetworkOnMainThreadException`异常。

这是为什么呢？要想知道，首先要了解什么是线程，什么是主线程以及主线程的用途是什么。

线程是个单一执行序列。单个线程中的代码会逐步执行。所有Android应用的运行都是从主线程开始的。然而，主线程不是线程那样的预定执行序列。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。

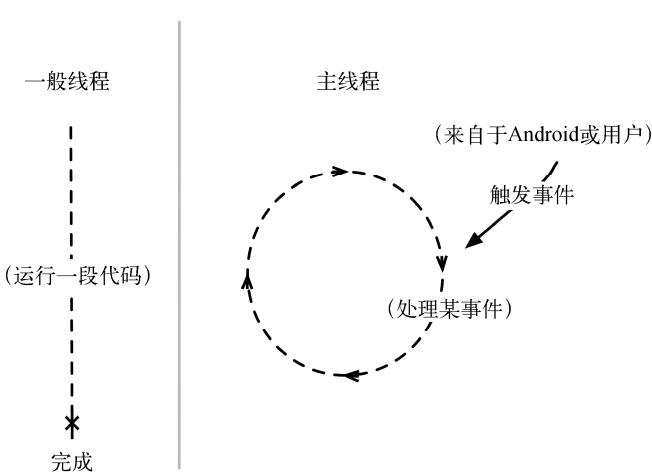


图23-6 一般线程与主线程

把应用想象成一家大型鞋店，闪电侠是这家店唯一的员工。（是不是人人梦寐以求的场景？）要让客户满意，他需要做大量的工作，如布置商品，为顾客取鞋，用量脚器为顾客量尺寸，等等。闪电侠并非浪得虚名，所以即便所有工作都由他一人完成，客户也能得到及时响应，感到满意。

为及时完成任务，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，必须有人花时间打电话调查此事。假设让闪电侠去做，他在忙于联络查找货物时，店里等候的顾客可就不耐烦了。

闪电侠就像应用里的主线程。它运行着所有更新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，主线程有时也叫作UI线程。）

事件处理循环让UI代码得以按顺序执行。这可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。

超越主线程

网络连接如同致电分销商寻找丢失的货物：相比其他任务，它更耗时。等待响应期间，用户界面毫无反应，这可能会导致应用无响应（Application Not Responding, ANR）现象发生。

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。用户就会看到如图23-7所示的画面。

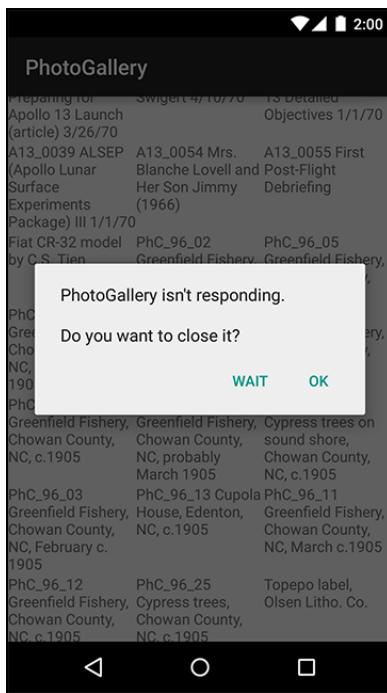


图23-7 应用无响应

回到假想的鞋店中，要想解决问题，需要再雇一名闪电侠专门负责联络供销商。Android系统中所做的操作与之类似，即创建一个后台线程，然后从该线程访问网络。

怎样使用后台线程最容易？答案是：使用**AsyncTask**工具类。

稍后，还会看到**AsyncTask**类的其他用处。现在，还是先利用网络连接代码做点实事吧。

23.5 从 Flickr 获取 JSON 数据

JSON（JavaScript Object Notation）是近年流行开来的一种数据格式，尤其适用于Web服务。Android提供了标准的org.json包，可以利用包里的一些类创建和解析JSON数据。Android开发者文档有其详细信息。要详细了解JSON数据格式，请访问其官方网站：<http://json.org>。

Flickr提供了方便而强大的JSON 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。我们打算使用最简单的REST服务。查看文档得知，它的API端点（endpoint）是<https://api.flickr.com/services/rest/>。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。文档对该方法的描述为：“返回最近上传到flickr的公共图片清单。”这恰好就是PhotoGallery应用需要的方法。

`getRecent`方法唯一需要的参数是一个API key。为获得它，返回<http://www.flickr.com/services/api/>网页，找到并点击API keys链接进行申请。申请需使用Yahoo ID登录。登录成功后，可申请一个全新的非商业用途API key。申请成功后，可获得类似4f721bgafa75bf6d2cb9af54f937bb70这样的API key。

获取API key后，可直接向Flickr网络服务发起一个和下面类似的GET请求：

```
https://api.flickr.com/services/rest/?  
method=flickr.photos.getRecent&api_key=xxx&format=json&nojsoncallback=1.
```

Flickr默认返回XML格式的数据。要获得有效的JSON数据，就需要同时指定`format`和`nojsoncallback`参数。设置`nojsoncallback`为1就是告诉Flickr，返回的数据不应包括封闭方法名和括号。这样才会方便Java代码解析数据。

复制上述链接到浏览器，使用刚获取的API key替换xxx字符后回车。很快，就能看到如图23-8所示的JSON返回数据。

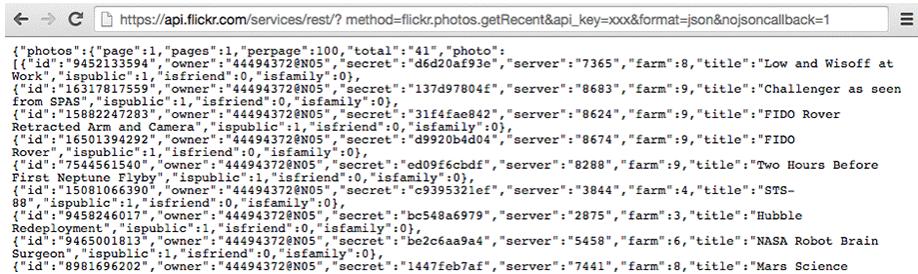


图23-8 JSON数据示例

接下来开始编码。首先，在`FlickrFetchr`类中添加一些常量，如代码清单23-7所示。

代码清单23-7 添加一些常量（FlickrFetchr.java）

```
public class FlickrFetchr {  
  
    private static final String TAG = "FlickrFetchr";  
  
    private static final String API_KEY = "yourApiKeyHere";  
    ...  
}
```

记得把`yourApiKeyHere`替换成刚获取的API key。

使用刚才定义的常量编写一个方法，构建请求URL并获取内容，如代码清单23-8所示。

代码清单23-8 添加`fetchItems()`方法（FlickrFetchr.java）

```
public class FlickrFetchr {  
  
    ...  
  
    String getUrlString(String urlSpec) throws IOException {
```

```

        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                .buildUpon()
                .appendQueryParameter("method", "flickr.photos.getRecent")
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter("format", "json")
                .appendQueryParameter("nojsoncallback", "1")
                .appendQueryParameter("extras", "url_s")
                .build().toString();
            String jsonString = getUrlString(url);
            Log.i(TAG, "Received JSON: " + jsonString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}

```

这里使用Uri.Builder构建了完整的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。

注意，我们还添加了method、api_key、format和nojsoncallback参数值。另外还指定了一个值为url_s的extras参数。这个参数值告诉Flickr：如有小尺寸图片，也一并返回其URL。

最后，修改PhotoGalleryFragment类中的AsyncTask内部类，调用新的fetchItems()方法，如代码清单23-9所示。

代码清单23-9 调用fetchItems()方法（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {
    ...
    private class FetchItemsTask extends AsyncTask<Void,Void(Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr()
                    .getUrlString("https://www.bignerdranch.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窗口中的Flickr JSON数据，如图23-9所示。（在LogCat搜索框中输入FlickrFetchr可以帮助查找。）



图23-9 Flickr JSON数据

本书撰写时，Android Studio的LogCat窗口还无法很好地换行显示。想查看返回的完整JSON字符串，需向右滚动窗口。（LogCat有时不好伺候。假如看不到类似图23-9的结果，也不用担心。模拟器连接有时不够稳定，可能无法及时显示日志内容。通常，它会自己恢复正常。实在不行，请重新运行应用或重启模拟器。）

成功获取Flickr JSON返回结果后，该如何使用呢？和处理其他数据一样，将其存入一个或多个模型对象中。稍后会为 PhotoGallery 应用创建的模型类名为 `GalleryItem`。图 23-10 为 PhotoGallery 应用的对象图解。

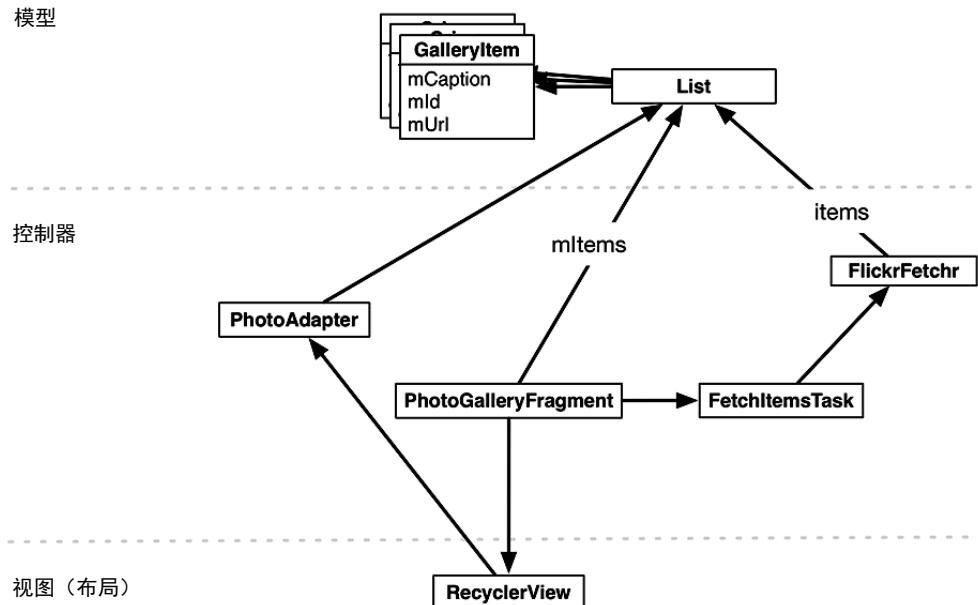


图23-10 PhotoGallery应用的对象图解

注意，为突出fragment和网络连接代码，图23-10并没有显示托管activity。
创建`GalleryItem`类并添加有关代码，如代码清单23-10所示。

代码清单23-10 创建模型对象类 (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    @Override
    public String toString() {
        return mCaption;
    }
}
```

利用Android Studio自动为mCaption、mId和mUrl变量生成获取方法与设置方法。

完成模型层对象的创建后，接下来的任务就是使用从Flickr JSON中解析的数据来填充它们。

解析JSON数据

浏览器和LogCat中显示的JSON数据难以阅读。如果用white space格式化后再打印出来，结果大致如图23-11所示。

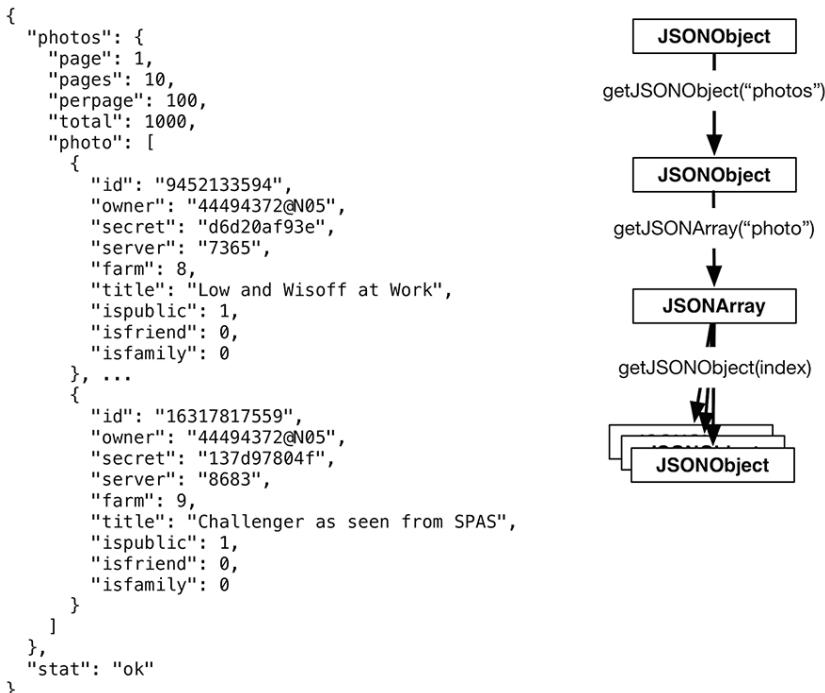


图23-11 格式化后的JSON数据

JSON对象是一系列包含在{}中的名值对。JSON数组是包含在[]中用逗号隔开的JSON对

象列表。对象彼此嵌套形成层级关系。

`json.org` API 提供有对应 JSON 数据的 Java 对象，如 `JSONObject` 和 `JSONArray`。使用 `JSONObject(String)` 构造函数，可以很方便地把 JSON 数据解析进相应的 Java 对象。更新 `fetchItems()` 方法执行解析任务，如代码清单 23-11 所示。

代码清单 23-11 解析 JSON 数据 (FlickrFetchr.java)

```
public class FlickrFetchr {
    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        try {
            ...
            Log.i(TAG, "Received JSON: " + jsonString);
            JSONObject jsonBody = new JSONObject(jsonString);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

`JSONObject` 构造方法解析传入的 Flickr JSON 数据后，会生成与原始 JSON 数据对应的对象树，如图 23-11 所示。

比较对象树和原始数据可知，顶层 `JSONObject` 对应着原始数据最外层的 {}。它包含了一个叫作 `photos` 的嵌套 `JSONObject`。层层往下，这个嵌套对象又包含了一个叫作 `photo` 的 `JSONArray`。这个嵌套数组中又包含了一组 `JSONObject`，而数组中的一个个 `JSONObject` 就是要获取的以 `metadata` 格式表示的一张张图片。

写一个 `parseItems(...)` 方法，取出每张图片的信息，生成一个个 `GalleryItem` 对象，再将它们添加到 `List` 中，如代码清单 23-12 所示。

代码清单 23-12 解析 Flickr 图片 (FlickrFetchr.java)

```
public class FlickrFetchr {
    private static final String TAG = "FlickrFetchr";
    ...
    public void fetchItems() {
        ...
    }
    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
```

```

    throws IOException, JSONException {
    JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
    JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

    for (int i = 0; i < photoJsonArray.length(); i++) {
        JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

        GalleryItem item = new GalleryItem();
        item.setId(photoJsonObject.getString("id"));
        item.setCaption(photoJsonObject.getString("title"));

        if (!photoJsonObject.has("url_s")) {
            continue;
        }

        item.setUrl(photoJsonObject.getString("url_s"));
        items.add(item);
    }
}
}

```

解析JSONObject层级结构时，这段代码用了getJSONObject(String name)和getJSONArray(String name)这两个便利方法（已标注在图23-11中）。

并不是每张图片都有对应的url_s链接，所以需要添加一个检查。

parseItems(...)方法需要List和JSONObject参数。因此，还要更新fetchItems()方法，让它返回一个包含GalleryItem的List，如代码清单23-13所示。

代码清单23-13 调用parseItems(...)方法 (FlickrFetchr.java)

```

public void List<GalleryItem> fetchItems() {
    List<GalleryItem> items = new ArrayList<>();

    try {
        String url = ...;
        String jsonString = getUrlString(url);
        Log.i(TAG, "Received JSON: " + jsonString);
        JSONObject jsonBody = new JSONObject(jsonString);
        parseItems(items, jsonBody);
    } catch (JSONException je) {
        Log.e(TAG, "Failed to parse JSON", je);
    } catch (IOException ioe) {
        Log.e(TAG, "Failed to fetch items", ioe);
    }

    return items;
}

```

运行PhotoGallery应用，测试JSON解析代码。现在，PhotoGallery应用还无法展示List中的内容。因此，要确认代码是否正确，需设置合适的断点，并使用调试器来检查代码逻辑。

23.6 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 RecyclerView 中显示图片标题。

首先定义一个 ViewHolder 内部类，如代码清单 23-14 所示。

代码清单 23-14 添加 ViewHolder 实现 (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder {
        private TextView mTitleTextView;
        public PhotoHolder(View itemView) {
            super(itemView);
            mTitleTextView = (TextView) itemView;
        }
        public void bindGalleryItem(GalleryItem item) {
            mTitleTextView.setText(item.toString());
        }
    }
    private class FetchItemsTask extends AsyncTask<Void,Void(Void> {
        ...
    }
}
```

接下来，添加一个 RecyclerView.Adapter 实现，提供基于 GalleryItem 对象 List 的 PhotoHolder，如代码清单 23-15 所示。

代码清单 23-15 添加 RecyclerView.Adapter 实现 (PhotoGalleryFragment.java)

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...
    private class PhotoHolder extends RecyclerView.ViewHolder {
        ...
    }
    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        private List<GalleryItem> mGalleryItems;
        public PhotoAdapter(List<GalleryItem> galleryItems) {

```

```

        mGalleryItems = galleryItems;
    }

    @Override
    public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        TextView textView = new TextView(getActivity());
        return new PhotoHolder(textView);
    }

    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        photoHolder.bindGalleryItem(galleryItem);
    }

    @Override
    public int getItemCount() {
        return mGalleryItems.size();
    }
}

...
}

```

既然RecyclerView要显示的数据已准备就绪，那么接下来编码完成adapter的配置和关联，如代码清单23-16所示。

代码清单23-16 实现setupAdapter()方法（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);
        mPhotoRecyclerView = (RecyclerView) v
            .findViewById(R.id.fragment_photo_gallery_recycler_view);
        mPhotoRecyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));

        setupAdapter();

        return v;
    }

    private void setupAdapter() {
        if (isAdded()) {
            mPhotoRecyclerView.setAdapter(new PhotoAdapter(mItems));
        }
    }
}

```

```

    }
}

...
}

```

根据当前模型数据（`GalleryItem`对象List）的状态，刚才添加的`setupAdapter()`方法会对应配置`RecyclerView`的adapter。应在`onCreateView(...)`方法中调用该方法，这样每次因设备旋转重新生成`RecyclerView`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，也应及时调用该方法。

注意，配置adapter前，应检查`isAdded()`的返回值是否为`true`。该检查确认fragment已与目标activity相关联，进而保证`getActivity()`方法返回结果不为空。

还记得吗？`fragment`可脱离任何activity而独立存在。在这之前，所有的方法调用都是由系统框架的回调方法驱动的，所以不会出现这种情况。如果`fragment`在接回调指令，则它必然关联着某个activity；如它单独存在，就会收不到回调指令。

既然在用`AsyncTask`，我们就正在从后台进程触发回调指令。因而不能确定`fragment`是否关联着activity。必须检查确认`fragment`是否仍与activity关联。如果没有关联，依赖于activity的操作（如创建`PhotoAdapter`，进而还会使用托管activity作为context来创建`TextView`）就会失败。

现在，从Flickr成功获取数据后，就需要调用`setupAdapter()`方法。我们的第一反应可能是在`FetchItemsTask`的`doInBackground(...)`方法尾部进行调用。这不是个好主意。还记得闪电侠与鞋店吗？现在，店里有两个闪电侠，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。

在计算机里，内存对象间步调不一致的冲突会让应用崩溃。因此，为避免安全隐患，不推荐也不允许从后台线程更新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。更为重要的是，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()`方法更新`RecyclerView`的数据源，如代码清单23-17所示。

代码清单23-17 添加adapter更新代码（`PhotoGalleryFragment.java`）

```

private class FetchItemsTask extends AsyncTask<Void,Void,Void List<GalleryItem>> {
    @Override
    protected Void List<GalleryItem> doInBackground(Void... params) {
        return new FlickrFetchr().fetchItems();
        return null;
    }

    @Override
    protected void onPostExecute(List<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返回的结果数据类型。它设置了doInBackground(...)方法返回结果的数据类型，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对象List。这样既修正了代码编译错误，还将GalleryItem对象List传递给onPostExecute(...)方法使用。

最后，我们添加了onPostExecute(...)方法实现代码。该方法接收doInBackground(...)方法返回的GalleryItem数据，并放入mItems变量，然后调用setupAdapter()方法更新RecyclerView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全部已下载GalleryItem的标题（类似图23-2）。

23.7 清理 AsyncTask

本章，AsyncTask的应用谨慎又合理，因此无需跟踪管理AsyncTask实例。例如，我们保留了fragment（调用setRetainInstance(true)方法），这样即使设备旋转，也不会重复创建新的AsyncTask去获取JSON数据。然而，在有些情况下，必须对其进行掌控；在需要的时候，甚至要能够撤销或重新运行AsyncTask。

在一些复杂的使用场景下，需将AsyncTask赋值给实例变量。一旦掌控了它，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。如果调用cancel(false)方法，它只是温和地设置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，如果调用cancel(true)方法，它会粗暴地终止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，应尽量避免使用它。

应该在什么时候、什么地方撤销AsyncTask呢？这要看情况了。先问问自己，如果fragment或activity已销毁了或是看不到了，AsyncTask当前的工作可以停止吗？如果可以，就在onStop(...)方法里（看不到视图），或者在onDestroy(...)方法里（fragment/activity实例已销毁）撤销AsyncTask实例。

即使fragment/activity已销毁了（或者视图已看不到了），也可以不撤销AsyncTask，让它运行至结束。不过，这可能会引发潜在的内存泄漏，也可能会出现UI更新问题（因为UI已失效）。如果不管用户怎么操作，一定要保证重要工作的完成，可以考虑其他解决方案，比如使用Service（详见第26章）。

23.8 深入学习：AsyncTask 再探

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>() {
    public Void doInBackground(String... params) {
        for (String parameter : params) {
            Log.i(TAG, "Received parameter: " + parameter);
        }
        return null;
    }
};
```

输入参数传入`execute(...)`方法（可接受一个或多个参数）：

```
task.execute("First parameter", "Second parameter", "Etc.");
```

然后，再把这些变量参数传递给`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 gestationProgressBar = /* A determinate progress bar */;
gestationProgressBar.setMax(42); /* max allowed gestation period */

AsyncTask<Void,Integer,Void> haveABaby = new AsyncTask<Void,Integer>() {
    public Void doInBackground(Void... params) {
        while (!babyIsBorn()) {
            Integer weeksPassed = getNumberOfWeeksPassed();
            publishProgress(weeksPassed);
            patientlyWaitForBaby();
        }
    }

    public void onProgressUpdate(Integer... params) {
        int progress = params[0];
        gestationProgressBar.setProgress(progress);
    }
};

/* call when you want to execute the AsyncTask */
haveABaby.execute();
```

进度更新通常发生在执行的后台进程中。问题是，在后台进程中无法完成必要的UI更新。因此`AsyncTask`提供了`publishProgress(...)`和`onProgressUpdate(...)`方法。

其工作方式是这样的：在后台线程中，从`doInBackground(...)`方法中调用`publishProgress(...)`方法。这样`onProgressUpdate(...)`方法便能够在UI线程上调用。因此，在`onProgressUpdate(...)`方法中执行UI更新就可行了，但必须在`doInBackground(...)`方法中使用`publishProgress(...)`方法对它们进行管控。

23.9 深入学习：AsyncTask的替代方案

在使用**AsyncTask**加载数据时，如果遇到设备配置改变，比如设备旋转，你得负责管理它的生命周期，同时还要保存好数据，不让其因旋转丢失。虽然调用**Fragment**的**setRetainInstance(true)**方法来保存数据可以解决问题，但它不是万能的。很多时候，你还得介入，编写特殊场景应对代码，让应用无懈可击。这些特殊场景有：用户在**AsyncTask**运行时点击后退键，以及启动**AsyncTask**的**fragment**因内存紧张而被销毁。

使用**Loader**是另一种可行的解决方案。它可以代劳很多（并非全部）棘手的事情。**Loader**用来从某些数据源加载数据（对象）。数据源可以是磁盘、数据库、**ContentProvider**、网络，甚至可以是另一进程。

AsyncTaskLoader是个抽象**Loader**。它可以使用**AsyncTask**把数据加载工作转移到其他线程上。我们创建的**loader**类几乎都是**AsyncTaskLoader**的子类。**AsyncTaskLoader**能在不阻塞主线程的前提下获取到数据，并把结果发送给目标对象。

相比**AsyncTask**，为什么要推荐使用**loader**呢？最重要的原因是，遇到类似设备旋转这样的场景时，**LoaderManager**会帮我们妥善管理**loader**及其加载的数据。而且，**LoaderManager**还负责启动和停止**loader**，以及管理**loader**的生命周期。怎么样？理由充足吧！

设备配置发生改变后，如果初始化一个已经加载完数据的**loader**，它能立即提交数据，而不是再次尝试获取数据。无论**fragment**是否得到保留，它都会这样做。这让我们轻松多了，从此再也不用考虑因保留**fragment**而产生的生命周期问题了。

23.10 挑战练习：Gson

无论什么平台，把JSON数据转化为Java对象都是应用开发的常见任务，如代码清单23-12所做的那样。于是，聪明的开发者就创建了一些工具库，希望能简化JSON数据和Java对象的互转。

Gson就是这样的一个工具库（<https://github.com/google/gson>）。不用写任何解析代码，Gson就能自动把JSON数据映射为Java对象。因为这个特性，Gson现在是开发者最喜爱的JSON解析库。

挑战自己，在应用中整合Gson库，简化**FlickrFetchr**中的JSON解析代码。

23.11 挑战练习：分页

getRecent方法默认返回一页包含100个结果的数据。不过，该方法还有个叫作**page**的参数，可以用它返回第二页、第三页等更多页数据。

请实现一个**RecyclerView.OnScrollListener**方法，只要用户看完当前页，就使用下页返回结果替换当前页。想更有挑战的话，可以尝试把后续结果页添加到当前结果页后面。

23.12 挑战练习：动态调整网格列

当前，显示图片标题的网格固定有3列。编写代码动态调整网格列数，实现在横屏或大屏幕设备上显示更多的标题列。

实现这个目标有个简单方法：分别为不同的设备配置或屏幕尺寸提供整数修饰资源。这实际和第17章中为不同尺寸屏幕提供不同布局的方式差不多。整数修饰资源应放置在res/values目录中。具体实施细节可参阅Android开发者文档。

提供整数修饰资源的方式不太好确定网格列细分粒度（只能凭经验预先定义列数）。下面再介绍一个颇具挑战的方法：在fragment的视图创建时就计算并设置好网格列数。显然，这种方式更加灵活实用。基于RecyclerView的当前宽度和预定义网格列宽，就可以计算出列数。

实施前还有个问题要解决：我们不能在onCreateView()方法中计算网格列数，因为这个时候RecyclerView还没有改变。不过，可以实现ViewTreeObserver.OnGlobalLayoutListener监听器方法和计算列数的onGlobalLayout()方法，然后使用addOnGlobalLayoutListener()把监听器添加给RecyclerView视图。

第 24 章

Looper、Handler和HandlerThread

从Flickr下载并解析JSON数据后，接下来的任务就是下载并显示图片。本章，为完成该任务，我们来学习如何使用Looper、Handler和HandlerThread。

24.1 配置 RecyclerView 以显示图片

在PhotoGalleryFragment中，当前PhotoHolder准备了TextView供RecyclerView的GridLayoutManager显示。每个TextView显示一个GalleryItem的标题。

要显示图片，就要让PhotoHolder提供ImageView。最终，每个ImageView都应显示一张从GalleryItem的mUrl地址下载的图片。

首先，为GalleryItem创建一个名为gallery_item.xml的布局文件。该布局包含一个ImageView组件，如图24-1所示。

```
ImageView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/fragment_photo_gallery_image_view"
android:layout_width="match_parent"
android:layout_height="120dp"
android:layout_gravity="center"
android:scaleType="centerCrop"
```

图24-1 Gallery图片项布局 (res/layout/gallery_item.xml)

ImageView由RecyclerView的GridLayoutManager负责管理，这意味着其宽度会变，而高度保持固定不变。为最大化利用ImageView的空间，应设置它的scaleType属性值为centerCrop。这个属性值的作用是先居中放置图片，然后放大较小图片，裁剪较大图片（裁两头）以匹配视图。

接下来更新PhotoHolder类，替换掉TextView，让其保存ImageView。同时，用一个新方法替换掉bindGalleryItem()方法，来设置ImageView的Drawable，如代码清单24-1所示。

代码清单24-1 更新PhotoHolder (PhotoGalleryFragment.java)

```

...
private class PhotoHolder extends RecyclerView.ViewHolder {
    private TextView mTitleTextView ImageView mItemImage;
    public PhotoHolder(View itemView) {
        super(itemView);
        mTitleTextView = (TextView) itemView;
        mItemImage = (ImageView) itemView
            .findViewById(R.id.fragment_photo_gallery_image_view);
    }
    public void bindGalleryItem(GalleryItem item) {
        mTitleTextView.setText(item.toString());
    }
    public void bindDrawable(Drawable drawable) {
        mItemImage.setImageDrawable(drawable);
    }
}
...

```

24

之前，传入PhotoHolder构造方法是TextView。现在，新版本PhotoHolder构造方法需要的是一个资源ID为R.id.fragment_photo_gallery_image_view的ImageView。

更新PhotoAdapter的onCreateViewHolder()方法，实例化gallery_item布局。然后将结果返回给PhotoAdapter的构造方法，如代码清单24-2所示。

代码清单24-2 更新PhotoAdapter的onCreateViewHolder()方法 (PhotoGalleryFragment.java)

```

public class PhotoGalleryFragment extends Fragment {
    ...
    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public PhotoHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
            TextView textView = new TextView(getActivity());
            return new PhotoHolder(textView);
        }
        LayoutInflator inflater = LayoutInflator.from(getActivity());
        View view = inflater.inflate(R.layout.gallery_item, viewGroup, false);
        return new PhotoHolder(view);
    }
    ...
}

```

现在，需要为每个ImageView设置占位图，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bill_up_close.png，并复制到项目的res/drawable目录中。

更新PhotoAdapter的onBindViewHolder()方法，使用占位图设置ImageView的Drawable，如代码清单24-3所示。

代码清单24-3 绑定默认图片（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {  
    ...  
  
    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {  
        ...  
  
        @Override  
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {  
            GalleryItem galleryItem = mGalleryItems.get(position);  
            photoHolder.bindGalleryItem(galleryItem);  
            Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);  
            photoHolder.bindDrawable(placeholder);  
        }  
  
        ...  
    }  
    ...  
}
```

运行PhotoGallery应用，欣赏Bill的一组大头照，如图24-2所示。



图24-2 满屏的Bill大头照

24.2 批量下载缩略图

当前，PhotoGallery应用联网代码的工作方式如下：PhotoGalleryFragment执行一个AsyncTask，该AsyncTask在后台线程上从Flickr获取JSON数据，然后解析JSON并将解析结果存入GalleryItem数组。最终每个GalleryItem都得到一个指向某张缩略图的URL。

接下来是下载这些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代码就行了？GalleryItem数组含有100个URL下载链接。我们每次下载一张，直到完成全部100张的下载。最后，执行onPostExecute(...)方法，让所有下载的图片全部显示在RecyclerView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，用户就只能对着Bill的照片看好久。

其次，保存缩略图也是个问题。100张缩略图保存在内存中固然轻松，但是1000张呢？如果还需要实现无限滚动来显示图片呢？显然，内存会耗尽。

考虑到这类问题，很多现实应用通常会选择仅在需要显示图片时才去下载。显然，RecyclerView及其adapter应负责实现按需下载。adapter触发图片下载就放在onBindViewHolder(...)方法中实现。

AsyncTask是执行后台线程的最简单方式，但它不适用于那些重复且长时间运行的任务。（关于具体原因，请阅读章末深入学习部分的内容。）

接下来会创建一个专用的后台线程，替换AsyncTask。这是实现按需下载的最常用方式。

24.3 与主线程通信

虽然我们打算让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RecyclerView的adapter来实现图片显示的呢？

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。他需要将库存已找回的消息通知给前台闪电侠。如果前台闪电侠非常忙碌，后台闪电侠就一时无法联系他。于是，他选择登记预约，等到前台闪电侠空闲下来再联系。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。后台闪电侠写下鞋已入库的信息，并将其放置在前台闪电侠的收件箱顶部。前台闪电侠如需告诉后台闪电侠库存已空的信息，也可以这样做。

实践证明，收件箱的办法非常好用。有时，闪电侠可能需要尽快完成一项任务，但不方便立即去做。这种情况下，他也可以在自己的收件箱放上一条提醒消息，等有空了就赶紧去完成它。

Android系统中，线程使用的收件箱叫作消息队列（message queue）。使用消息队列的线程叫作消息循环（message loop）。消息循环会循环检查队列上是否有新消息，如图24-3所示。

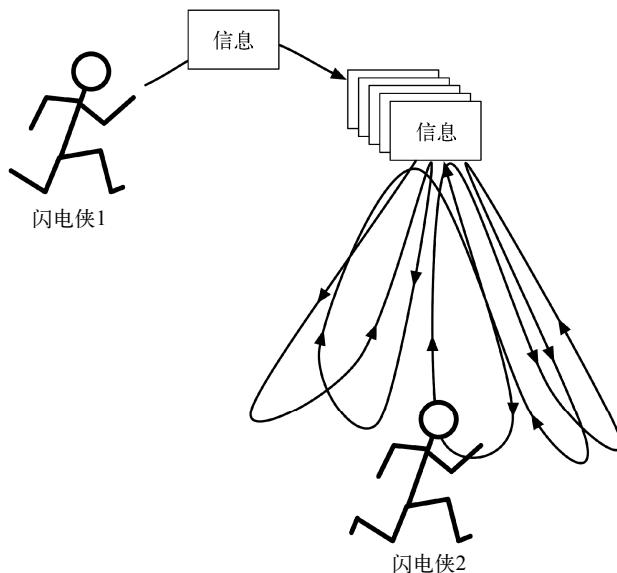


图24-3 闪电之舞

消息循环由线程和looper组成。Looper对象管理着线程的消息队列。

主线程就是个消息循环，因此也拥有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，我们会使用名为HandlerThread的类。

24.4 创建并启动后台线程

继承HandlerThread类，创建一个名为ThumbnailDownloader的新类。然后，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24-4所示。

代码清单24-4 初始线程代码（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";

    private Boolean mHasQuit = false;

    public ThumbnailDownloader() {
        super(TAG);
    }

    @Override
    public boolean quit() {
        mHasQuit = true;
        return super.quit();
    }
}
```

```

    public void queueThumbnail(T target, String url) {
        Log.i(TAG, "Got a URL: " + url);
    }
}

```

注意，ThumbnailDownloader类使用了<T>泛型参数。ThumbnailDownloader类的使用者（这里指PhotoGalleryFragment），需要使用某些对象来识别每次下载，并确定该使用下载图片更新哪个UI元素。有了泛型参数，实施起来方便了很多。

queueThumbnail()方法需要一个T类型对象（标识具体那次下载）和一个String参数（URL下载链接）。同时，它也是PhotoAdapter在其onBindViewHolder(...)实现方法中要调用的方法。

打开PhotoGalleryFragment.java文件，为PhotoGalleryFragment添加一个ThumbnailDownloader类型的成员变量。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24-5所示。

代码清单24-5 创建ThumbnailDownloader（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {

    private static final String TAG = "PhotoGalleryFragment";

    private RecyclerView mPhotoRecyclerView;
    private List<GalleryItem> mItems = new ArrayList<>();
    private ThumbnailDownloader<PhotoHolder> mThumbnailDownloader;

    ...

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        new FetchItemsTask().execute();

        mThumbnailDownloader = new ThumbnailDownloader<>();
        mThumbnailDownloader.start();
        mThumbnailDownloader.getLooper();
        Log.i(TAG, "Background thread started");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mThumbnailDownloader.quit();
        Log.i(TAG, "Background thread destroyed");
    }

    ...
}

```

`ThumbnailDownloader`的泛型参数支持任何对象，但在这里，`PhotoHolder`最合适，因为该视图是最终显示下载图片的地方。

下面是两点安全注意事项。

- 在`ThumbnailDownloader`线程上，在`start()`方法之后调用`getLooper()`方法。（稍后，我们会学习到更多有关`Looper`的知识。）这是一种保证线程就绪的处理方式，可以避免潜在竞争（尽管极少发生）。调用`getLooper()`方法之前，没办法保证`onLooperPrepared()`方法已得到调用。所以，`Handler`为空的话，调用`queueThumbnail()`方法很可能会失败。
- 在`onDestroy()`方法内调用`quit()`方法结束线程。这非常关键。如不终止`HandlerThread`，它会一直运行下去。

最后，在`PhotoAdapter.onBindViewHolder(...)`方法中，调用线程的`queueThumbnail()`方法，并传入放置图片的`PhotoHolder`和`GalleryItem`的URL，如代码清单24-6所示。

代码清单24-6 关联使用`ThumbnailDownloader` (`PhotoGalleryFragment.java`)

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
        ...
        @Override
        public void onBindViewHolder(PhotoHolder photoHolder, int position) {
            GalleryItem galleryItem = mGalleryItems.get(position);
            Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
            photoHolder.bindDrawable(placeholder);
            mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
        }
        ...
    }
    ...
}
```

运行`PhotoGallery`应用并查看LogCat窗口。在`RecyclerView`视图中滚动时，可看到`ThumbnailDownloader`正在处理各个下载请求。

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`ThubnailDownloader`的消息队列中。

24.5 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。

24.5.1 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“你跑得真快，闪电侠”，而是需要处理的各种任务。

消息是Message类的一个实例，它有好几个实例变量，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message handler的简称。创建Message时，它会自动与一个Handler相关联。Message待处理时，Handler对象负责触发消息处理事件。

24.5.2 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，如图24-4所示。

24

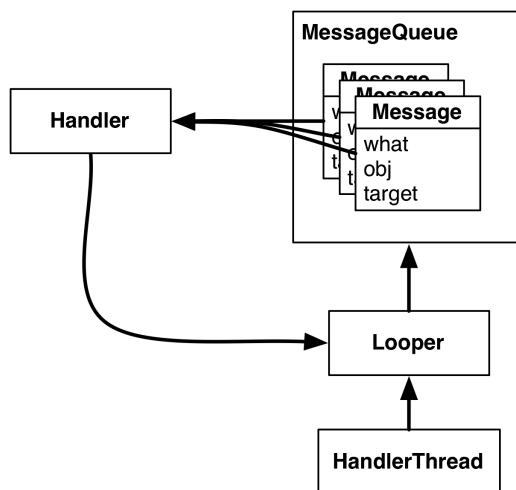


图24-4 Looper、Handler、HandlerThread和Message

Looper拥有Message收件箱，所以Message必须在Looper上发布或处理。既然有这层关系，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，如图24-5所示。Looper拥有整个Message队列。多个Message可以引用同一目标Handler。

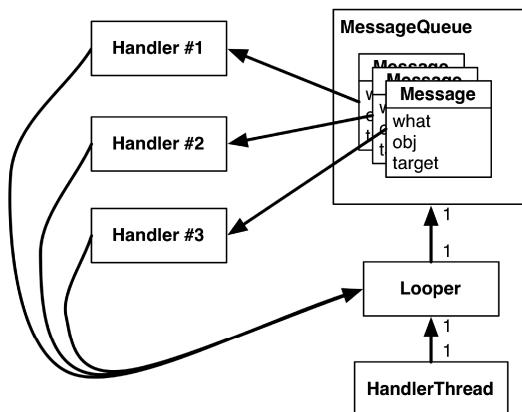


图24-5 多个Handler对应一个Looper

多个Handler也可与一个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。

24.5.3 使用 handler

通常不需要手动设置消息的目标Handler。创建信息时，最好调用`Handler.obtainMessage(...)`方法。当传入其他消息字段给它时，该方法会自动设置目标给Handler对象。

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。相比创建新实例，这样显然更有效率。

一旦取得Message，就可以调用`sendToTarget()`方法将其发送给它的Handler。然后，Handler会将这个Message放置在Looper消息队列的尾部。

对于PhotoGallery应用，我们会在`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T类型对象，这里指由adapter传入`queueThumbnail()`方法的`PhotoHolder`，它用于标识下载。

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24-6展示了其中的对象关系。

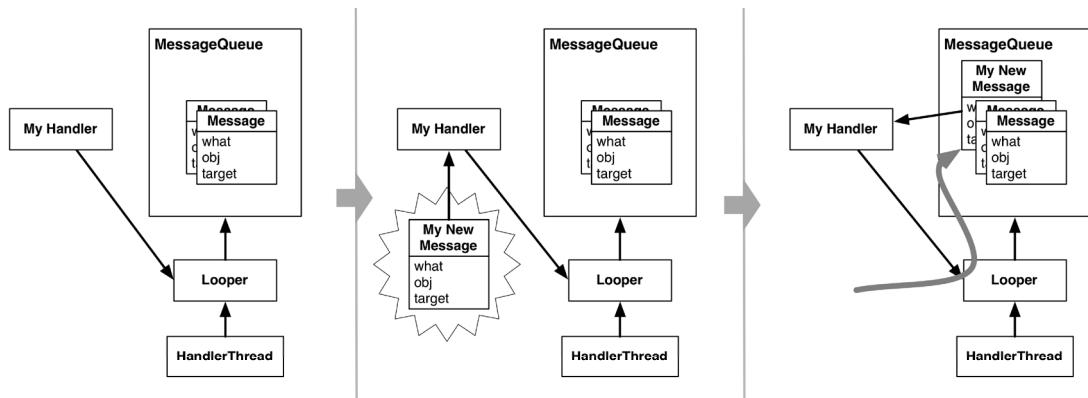


图24-6 创建并发送Message

这里，`handleMessage(...)`实现方法将使用FlickrFetchr从URL下载图片字节数据，然后再转换为位图。

首先，在ThumbnailDownloader.java中添加一些常量和成员变量，如代码清单24-7所示。

24

代码清单24-7 添加一些常量和成员变量（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentHashMap<T, String> mRequestMap = new ConcurrentHashMap<>();

    ...
}
```

`MESSAGE_DOWNLOAD`用来标识下载请求消息。（`ThumbnailDownloader`会把它设为任何新创建下载消息的`what`属性。）

新添加的`mRequestHandler`用来存储对`Handler`的引用。这个`Handler`负责在`ThumbnailDownloader`后台线程上管理下载请求消息队列。这个`Handler`也负责从消息队列里取出并处理下载请求消息。

新添加的`mRequestMap`是个`ConcurrentHashMap`。这是一种线程安全的`HashMap`。这里，使用一个标记下载请求的`T`类型对象作为key，我们可以存取和请求关联的URL下载链接。（这个标记对象是`PhotoHolder`，下载结果该发送给哪个显示图片的UI元素再明白不过了。）

接下来，在`queueThumbnail(...)`方法中添加代码，更新`mRequestMap`并把下载消息放到后台线程的消息队列中去，如代码清单24-8所示。

代码清单24-8 获取和发送消息（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
```

```

private static final int MESSAGE_DOWNLOAD = 0;

private boolean mHasQuit = false;
private Handler mRequestHandler;
private ConcurrentHashMap<T, String> mRequestMap = new ConcurrentHashMap<>();

public ThumbnailDownloader() {
    super(TAG);
}

@Override
public boolean quit() {
    mHasQuit = true;
    return super.quit();
}

public void queueThumbnail(T target, String url) {
    Log.i(TAG, "Got a URL: " + url);

    if (url == null) {
        mRequestMap.remove(target);
    } else {
        mRequestMap.put(target, url);
        mRequestHandler.obtainMessage(MESSAGE_DOWNLOAD, target)
            .sendToTarget();
    }
}
}

```

从mRequestHandler直接获取到消息后，mRequestHandler也就自动成为了这个新Message对象的target。这表明mRequestHandler会负责处理刚从消息队列中取出的这个消息。消息的what属性是MESSAGE_DOWNLOAD。它的obj属性是传递给queueThumbnail(...)方法的T target值（这里指PhotoHolder）。

新消息就代表指定为T target（RecyclerView中的PhotoHolder）的下载请求。还记得吗？PhotoGalleryFragment中，RecyclerView的adapter实现就是从onBindViewHolder(...)方法里调用queueThumbnail(...），把待下载图片及其URL传给PhotoHolder。

注意，消息自身是不带URL信息的。我们的做法是使用PhotoHolder和URL的对应关系更新mRequestMap。随后，我们会从mRequestMap中取出图片URL，以保证总是使用了匹配PhotoHolder实例的最新下载请求URL。（这很重要，因为RecyclerView中的ViewHolder是会不断回收重用的。）

最后，初始化mRequestHandler并定义该Handler在得到消息队列中的下载消息后应执行的任务，如代码清单24-9所示。

代码清单24-9 处理消息（ThumbnailDownloader.java）

```

public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;
}

```

```

private boolean mHasQuit = false;
private Handler mRequestHandler;
private ConcurrentMap<T, String> mRequestMap = new ConcurrentHashMap<>();

public ThumbnailDownloader() {
    super(TAG);
}

@Override
protected void onLooperPrepared() {
    mRequestHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            if (msg.what == MESSAGE_DOWNLOAD) {
                T target = (T) msg.obj;
                Log.i(TAG, "Got a request for URL: " + mRequestMap.get(target));
                handleRequest(target);
            }
        }
    };
}

@Override
Public boolean quit(){
mHasQuit = true;
return super.quit();
}

public void queueThumbnail(T target, String url) {
    ...
}

private void handleRequest(final T target) {
    try {
        final String url = mRequestMap.get(target);

        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);
    }
}
}

```

24

在onLooperPrepared()方法内，我们在Handler子类中实现了Handler.handleMessage(...)方法。HandlerThread.onLooperPrepared()是在Looper首次检查消息队列之前调用的，所以

该方法成了创建Handler实现的好地方。

在Handler.handleMessage(...)方法中，首先检查消息类型，再获取obj值（T类型下载请求），然后将其传递给handleRequest(...)方法处理。（队列中的下载消息取出并可以处理时，就会触发调用Handler.handleMessage(...)方法。）

handleRequest()方法是执行下载动作的地方。在这里，确认URL有效后，就将它传递给FlickrFetchr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把getUrlBytes(...)返回的字节数组转换为位图。

运行PhotoGallery应用，通过LogCat窗口的日志确认代码工作正常。

当然，在将位图设置给PhotoHolder视图（来自于PhotoAdapter）之前，请求并不会得到完全处理。不过这是UI的工作。因此，必须回到主线程上完成它。

目前为止，所有的工作就是在线程上使用handler和消息——ThumbnailDownloader把消息放入自己的收件箱。下一小节要学习内容是：ThumbnailDownloader如何使用Handler访问主线程。

24.5.4 传递 handler

当前，使用ThumbnailDownloader的mRequestHandler，可以从主线程安排后台线程任务，如图24-7所示。

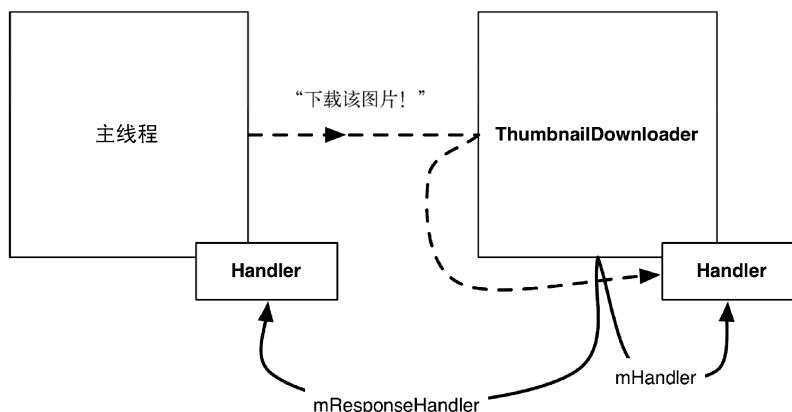


图24-7 从主线程安排ThumbnailDownloader上的任务

反过来，也可以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24-8所示。

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。主线程上创建的这个Handler也可以传递给另一线程。传递出去的Handler与创建它的线程Looper始终保持着联系。因此，已传出Handler负责处理的所有消息都将在主线程的

消息队列中处理。

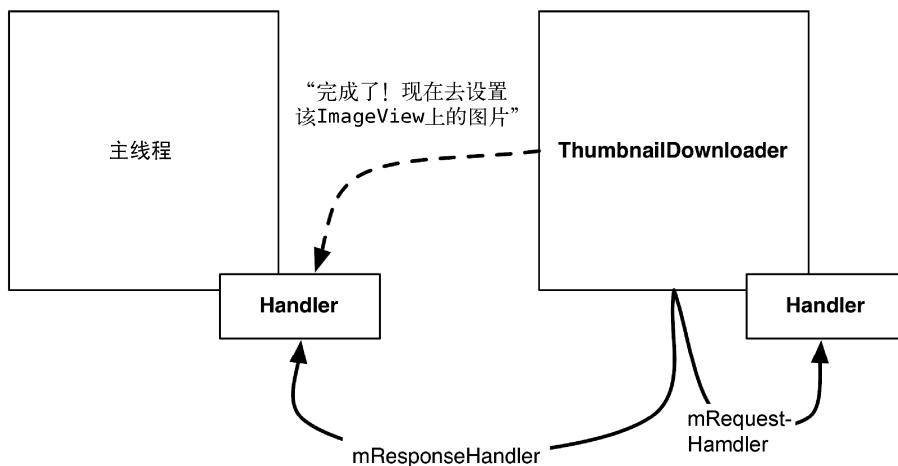


图24-8 从ThumbnailDownloader线程上规划主线程上执行的任务

24

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（在请求者和结果间）通信的监听器接口，如代码清单24-10所示。

代码清单24-10 处理消息（ThumbnailDownloader.java）

```

public class ThumbnailDownloader<T> extends HandlerThread {
    private static final String TAG = "ThumbnailDownloader";
    private static final int MESSAGE_DOWNLOAD = 0;

    private boolean mHasQuit = false;
    private Handler mRequestHandler;
    private ConcurrentHashMap<T, String> mRequestMap = new ConcurrentHashMap<>();
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;

    public interface ThumbnailDownloadListener<T> {
        void onThumbnailDownloaded(T target, Bitmap thumbnail);
    }

    public void setThumbnailDownloadListener(ThumbnailDownloadListener<T> listener) {
        mThumbnailDownloadListener = listener;
    }

    public ThumbnailDownloader(Handler responseHandler) {
        super(TAG);
        mResponseHandler = responseHandler;
    }

    ...
}

```

在图片下载完成，可以交给UI去显示时，定义在ThumbnailDownloadListener新接口中的onThumbnailDownloaded(...)方法就会被调用。稍后，为了把下载任务和UI更新任务（把图片放入ImageView）分开，我们会使用这个监听器方法把处理已下载图片的任务代理给另一个类（这里指PhotoGalleryFragment），而不是ThumbnailDownloader。这样，ThumbnailDownloader就可以把下载结果传给其他视图对象。

接下来，修改PhotoGalleryFragment类，将关联主线程的Handler传递给ThumbnailDownloader。另外，再设置一个ThumbnailDownloadListener处理已下载图片，如代码清单24-11所示。

代码清单24-11 关联使用反馈Handler（PhotoGalleryFragment.java）

```

...
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);
    new FetchItemsTask().execute();

    Handler responseHandler = new Handler();
    mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
    mThumbnailDownloader.setThumbnailDownloadListener(
        new ThumbnailDownloader.ThumbnailDownloadListener<PhotoHolder>() {
            @Override
            public void onThumbnailDownloaded(PhotoHolder photoHolder, Bitmap bitmap) {
                Drawable drawable = new BitmapDrawable(getResources(), bitmap);
                photoHolder.bindDrawable(drawable);
            }
        }
    );
    mThumbnailDownloader.start();
    mThumbnailDownloader.getLooper();
    Log.i(TAG, "Background thread started");
}
...

```

前面说过，Handler默认与当前线程的Looper相关联。这个Handler是在onCreate(...)方法中创建的，因此它会与主线程的Looper相关联。

现在，通过mResponseHandler，ThumbnailDownloader能够访问与主线程Looper绑定的Handler。同时，还有ThumbnailDownloadListener使用返回的Bitmap执行UI更新操作。具体来说，就是通过onThumbnailDownloaded实现，使用新下载的Bitmap来设置PhotoHolder的Drawable。

和在后台线程上把下载图片的请求放入消息队列类似，我们也可以返回定制Message给主线程，要求显示已下载图片。不过，这需要另一个Handler子类，以及一个handleMessage(...)覆盖方法。方便起见，我们转而使用另一个方便的Handler方法——post(Runnable)。

Handler.post(Runnable)是一个发布Message的便利方法。具体如下：

```

Runnable myRunnable = new Runnable() {
    @Override
    public void run() {
        /* Your code here */
    }
};
Message m = mHandler.obtainMessage();
m.callback = myRunnable;

```

Message设有回调方法时，它从消息队列取出后，是不会发给target Handler的。相反，存储在回调方法中的Runnable的run()方法会直接执行。

在ThumbnailDownloader.handleRequest()方法中，添加代码清单24-12所示代码。

代码清单24-12 图片下载与显示 (ThumbnailDownloader.java)

```

public class ThumbnailDownloader<T> extends HandlerThread {

    ...
    private Handler mResponseHandler;
    private ThumbnailDownloadListener<T> mThumbnailDownloadListener;
    ...

    private void handleRequest(final T target) {
        try {
            final String url = mRequestMap.get(target);

            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 (mRequestMap.get(target) != url ||
                        mHasQuit) {
                        return;
                    }

                    mRequestMap.remove(target);
                    mThumbnailDownloadListener.onThumbnailDownloaded(target, bitmap);
                }
            });
        } catch (IOException ioe) {
            Log.e(TAG, "Error downloading image", ioe);
        }
    }
}

```

24

因为mResponseHandler与主线程的Looper相关联，所以UI更新代码会在主线程中完成。

那么这段代码有什么作用呢？首先，它再次检查`requestMap`。这很有必要，因为`RecyclerView`会循环使用其视图。在`ThumbnailDownloader`下载完成`Bitmap`之后，`RecyclerView`可能循环使用了`PhotoHolder`并相应请求一个不同的URL。该检查可保证每个`PhotoHolder`都能获取到正确的图片，即使中间发生了其他请求也无妨。

接下来，选中`mHasQuit`。如果`ThumbnailDownloader`已经退出，运行任何回调方法可能都不太安全。

最后，从`requestMap`中删除配对的`PhotoHolder-URL`，然后将位图设置到目标`PhotoHolder`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`PhotoHolder`视图的失效，`ThumbnailDownloader`可能会挂起。如果点击这些`ImageView`，就会发生异常。

新增下列方法清除队列中的所有请求，如代码清单24-13所示。

代码清单24-13 添加清理方法（ThumbnailDownloader.java）

```
public class ThumbnailDownloader<T> extends HandlerThread {
    ...
    public void queueThumbnail(T target, String url) {
        ...
    }

    public void clearQueue() {
        mRequestHandler.removeMessages(MESSAGE_DOWNLOAD);
    }

    private void handleRequest(final T target) {
        ...
    }
}
```

既然视图已销毁，别忘了在`PhotoGalleryFragment.java`中清空下载器，如代码清单24-14所示。

代码清单24-14 调用清理方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        mThumbnailDownloader.clearQueue();
    }

    @Override
```

```

public void onDestroy() {
    ...
}

...
}

```

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应用已完成了从Flickr下载并显示图片的基本目标。接下来的几章还会为应用添加更多功能，如搜索图片、在Web视图中打开图片所在的Flickr网页。

24.6 深入学习：AsyncTask 与线程

理解了Handler和Looper之后，`AsyncTask`也就没有当初看上去那么神奇了。不过就本章所做的线程相关工作来看，使用`AsyncTask`确实省了不少事。那么为什么要用`HandlerThread`，而不使用`AsyncTask`呢？

原因有很多。最基本的一个原因是`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`AsyncTask`，那么很可能就是错用了它。

有一个技术层面的理由更让人信服：在Android 3.2系统版本中，`AsyncTask`的内部实现有了重大变化。自Android 3.2版本起，`AsyncTask`不再为每一个`AsyncTask`实例单独创建线程。相反，它使用一个`Executor`在单一的后台线程上运行所有`AsyncTask`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。

24.7 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。

为了让应用反应更快，大多数现实应用都通过以下两种方式增强自己的代码：

- 增加缓存层
- 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作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，缓存会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，每次下载完`Bitmap`时，将其存

入缓存中。随后，准备下载新图片时，应首先查看缓存，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，就预先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，为前十个和接下来十个GalleryItem预加载Bitmap。

24.8 深入学习：解决图片下载问题

本书教学使用的都是Android标准库中的工具。只要愿意，还可以使用各种第三方库。这些库专用于一些特定场景，可以节约大量开发时间，其中就包括PhotoGallery中的图片下载。

必须承认，本章PhotoGallery应用的图片下载解决方案远不够完美。如果还想优化性能，实现棘手的缓存功能，很自然就会想到是否有人已开发了更好的解决方案。答案是肯定的。有好几个高性能图片下载库可供使用。在开发生产应用时，我们选用了Picasso库 (<http://square.github.io/picasso/>)。

使用Picasso库，只需一条语句就能实现本章的图片下载功能：

```
private class PhotoHolder extends RecyclerView.ViewHolder {
    ...
    public void bindGalleryItem(GalleryItem galleryItem) {
        Picasso.with(getActivity())
            .load(galleryItem.getUrl())
            .placeholder(R.drawable.bill_up_close)
            .into(mItemImage);
    }
    ...
}
```

上述代码中，流接口需要使用with(Context)指定一个context。load(String)用于指定要下载图片的URL。into(ImageView)用于指定加载下载结果的ImageView对象。当然，还有一些其他配置选项可用，比如指定要显示的已完全下载的图片（使用placeholder(int)和placeholder(drawable)）。

在PhotoGallery应用中，只要引入Picasso依赖库，并在PhotoAdapter.onBindViewHolder(...)方法中替换原有代码（直到bindGalleryItem(...)方法前），就用上了Picasso库的强大下载功能。

Picasso包办了ThumbnailDownloader（还有ThumbnailDownloader.ThumbnailDownloadListener<T>回调方法）的所有工作以及FlickrFetchr中的图片处理相关工作，所以可以直接删除ThumbnailDownloader实现（FlickrFetchr中的JSON数据下载还是需要的）。使用Picasso，除了能大大简化代码，还可以轻松使用它的图片动画、磁盘缓存等高级功能。

可以在项目结构窗口中将Picasso作为库依赖项添加在项目中，就像RecyclerView等其他依赖项一样。

本章的任务是为PhotoGallery应用添加Flickr图片搜索功能。我们会学习如何以Android特有的方式，在应用中整合搜索功能。实际上，搜索功能一开始就整合进了Android系统，但随着Android各个新版本的推出，该功能也一直在变。

本章，我们会使用SearchView来实现搜索。

使用SearchView，用户可以输入查询关键字，提交查询请求搜索Flickr。最后，返回结果会显示在RecyclerView中，如图25-1所示。此外，用户提交过的查询关键字会保存在系统文件中。这样，即便应用甚至设备重启，用户上次输入的查询信息依然可以找回。

25

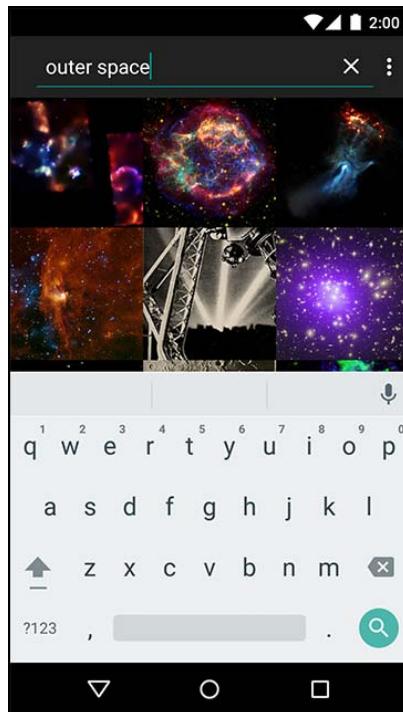


图25-1 搜索界面

25.1 搜索 Flickr 网站

搜索Flickr网站需要调用`flickr.photos.search`方法。以下为搜索cat文本的GET请求示例：

```
https://api.flickr.com/services/rest/?method=flickr.photos.search
&api_key=xxx&format=json&nojsoncallback=1&text=cat
```

可以看到，搜索方法指定为`flickr.photos.search`。一个`text`新参数附加在请求后面，它的内容就是cat这样的搜索字符串。

尽管搜索URL和图片下载请求URL不同，但网站返回的JSON数据格式是一样的。这方便了开发，因为不管是搜索还是下载图片，我们都可以使用同样的JSON数据解析逻辑。

首先，重构`FlickrFetchr`代码以复用JSON数据解析逻辑。先添加一些URL复用相关的常量，如代码清单25-1所示。从`fetchItems`方法中剪切URI创建代码，复制作为`ENDPOINT`的值。注意，只应使用加灰部分的代码。`ENDPOINT`常量不应包括查询方法参数，`build`语句不应使用`toString()`方法。

代码清单25-1 添加URL常量（`FlickrFetchr.java`）

```
public class FlickrFetchr {
    private static final String TAG = "FlickrFetchr";

    private static final String API_KEY = "yourApiKeyHere";
    private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
    private static final String SEARCH_METHOD = "flickr.photos.search";
    private static final Uri ENDPOINT = Uri
        .parse("https://api.flickr.com/services/rest/")
        .buildUpon()
        .appendQueryParameter("api_key", API_KEY)
        .appendQueryParameter("format", "json")
        .appendQueryParameter("nojsoncallback", "1")
        .appendQueryParameter("extras", "url_s")
        .build();

    ...

    public List<GalleryItem> fetchItems() {
        List<GalleryItem> items = new ArrayList<>();
        try {
            String url = Uri.parse("https://api.flickr.com/services/rest/")
                .buildUpon()
                .appendQueryParameter("method", "flickr.photos.getRecent")
                .appendQueryParameter("api_key", API_KEY)
                .appendQueryParameter("format", "json")
                .appendQueryParameter("nojsoncallback", "1")
                .appendQueryParameter("extras", "url_s")
                .build().toString();
            String jsonString = getUrlString(url);
            ...
        } catch (IOException e) {
            Log.e(TAG, "Exception in fetchItems()", e);
        }
    }
}
```

```

        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        } catch (JSONException je) {
            Log.e(TAG, "Failed to parse JSON", je);
        }

        return items;
    }

    ...
}

```

(刚才的代码调整会导致`fetchItems()`方法出错。没关系，现在可以直接忽略，这个方法稍后就会删除。)

为了通用，重命名`fetchItems()`方法为`downloadGalleryItems(String url)`。新方法也不应是公共的了，所以把它改成私有方法，如代码清单25-2所示。

代码清单25-2 重构Flickr代码（FlickrFetchr.java）

```

public class FlickrFetchr {

    ...
    public List<GalleryItem> fetchItems() {
        private List<GalleryItem> downloadGalleryItems(String url) {
            List<GalleryItem> items = new ArrayList<>();

            try {
                String jsonString = getUrlString(url);
                Log.i(TAG, "Received JSON: " + jsonString);
                JSONObject jsonBody = new JSONObject(jsonString);
                parseItems(items, jsonBody);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch items", ioe);
            } catch (JSONException je) {
                Log.e(TAG, "Failed to parse JSON", je);
            }

            return items;
        }

        ...
    }
}

```

`downloadGalleryItems(String)`新方法使用URL参数，不用再重头创建URL了。所以，在其内部添加一个新方法基于方法（搜索和下载）和查询值创建URL，如代码清单25-3所示。

代码清单25-3 添加创建URL的辅助方法（FlickrFetchr.java）

```

public class FlickrFetchr {

    ...

    private List<GalleryItem> downloadGalleryItems(String url) {

```

```

    ...

    private String buildUrl(String method, String query) {
        Uri.Builder uriBuilder = ENDPOINT.buildUpon()
            .appendQueryParameter("method", method);

        if (method.equals(SEARCH_METHOD)) {
            uriBuilder.appendQueryParameter("text", query);
        }

        return uriBuilder.build().toString();
    }

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
        throws IOException, JSONException {
        ...
    }
}

```

如同已删除的`fetchItems()`方法，`buildUrl(...)`方法会自动拼接必要的参数。不过，参数值是动态确定的。如果判断出是搜索，它就会附加一个`text`参数值。如果判断出是搜索，它还会附加一个`text`参数值。

现在为下载和搜索添加相应的方法，如代码清单25-4所示。

代码清单25-4 添加方法用于下载和搜索（FlickrFetchr.java）

```

public class FlickrFetchr {
    ...

    public String getUrlString(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public List<GalleryItem> fetchRecentPhotos() {
        String url = buildUrl(FETCH_RECENTS_METHOD, null);
        return downloadGalleryItems(url);
    }

    public List<GalleryItem> searchPhotos(String query) {
        String url = buildUrl(SEARCH_METHOD, query);
        return downloadGalleryItems(url);
    }

    private List<GalleryItem> downloadGalleryItems(String url) {
        List<GalleryItem> items = new ArrayList<>();
        ...
        return items;
    }

    ...
}

```

FlickrFetchr方法现在可以处理搜索和下载图片了。`fetchRecentPhotos()` 和 `searchPhotos(String)`方法各自担任从Flickr获取`GalleryItem`的公共接口。

FlickrFetchr方法中的重构应体现在fragment代码中。在PhotoGalleryFragment.java中，更新`FetchItemsTask`类代码。

代码清单25-5 硬编码的搜索字符串（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {  
    ...  
  
    private class FetchItemsTask extends AsyncTask<Void,Void,List<GalleryItem>> {  
  
        @Override  
        protected List<GalleryItem> doInBackground(Void... params) {  
            return new FlickrFetchr().fetchItems();  
            String query = "robot"; // Just for testing  
  
            if (query == null) {  
                return new FlickrFetchr().fetchRecentPhotos();  
            } else {  
                return new FlickrFetchr().searchPhotos(query);  
            }  
        }  
  
        @Override  
        protected void onPostExecute(List<GalleryItem> items) {  
            mItems = items;  
            setupAdapter();  
        }  
    }  
}
```

如果搜索查询字符串非空（现在肯定非空），`FetchItemsTask`就会执行Flickr搜索任务。否则，就和以前一样，默认下载最新公共图片。

尽管还没有为用户提供输入查询的用户界面，但我们可以使用硬编码搜索字符串来测试搜索代码。

运行PhotoGallery并查看返回结果。应该可以看到一两张机器人图片，如图25-2所示。



图25-2 搜索结果

25.2 使用 SearchView

既然FlickrFetchr已支持搜索，现在就来利用SearchView创建搜索界面，让用户输入查询关键字并触发搜索。

SearchView是个操作视图。所谓操作视图，就是可以内置在工具栏中的视图。SearchView可以让整个搜索界面完全内置在应用的工具栏中。

首先，确认应用顶部有工具栏（包含应用名称）。如果没有，请参照第13章添加。

接下来，在res/menu/fragment_photo_gallery.xml文件中，为PhotoGalleryFragment创建一个新的菜单XML文件，如代码清单25-6所示。可以通过这个文件指定工具栏上要显示什么。

代码清单25-6 添加菜单XML文件（res/menu/fragment_photo_gallery.xml）

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

    <item android:id="@+id/menu_item_search"
          android:title="@string/search"
          app:actionViewClass="android.support.v7.widget.SearchView"
          app:showAsAction="ifRoom" />

    <item android:id="@+id/menu_item_clear"
```

```

    android:title="@string/clear_search"
    app:showAsAction="never" />
</menu>

```

新XML文件会出现一些错误，因为暂时还没有为`android:title`属性定义字符串。忽略这些，稍后就会修正。

通过为`app:actionViewClass`属性指定`android.support.v7.widget.SearchView`值，代码清单25-6中的第一个定义项告诉工具栏要显示`SearchView`。（注意，`showAsAction`和`actionViewClass`属性都需要`app`命名空间。不清楚为什么这样用的话，建议再复习一下第13章的相关内容。）

`SearchView (android.widget.SearchView)`最早是在API 11中（Honeycomb 3.0）引入的。不过，现在它已放入支持库中（`android.support.v7.widget.SearchView`）。那么到底该用哪个版本的`SearchView`呢？相信你已在代码中看到答案：使用支持库版本。是不是有点奇怪？毕竟`PhotoGallery`应用最低SDK版本已经是16了。

基于与第7章一样的理由，我们推荐使用支持库版本。随着Android新版本的发表，新功能也在不断添加。这些新功能通常会加入支持库中。主题就是这样的一个例子。Lollipop 5.0（API 21）发布后，原生`SearchView`有许多选项可以定制`SearchView`界面风格。要想在早期版本Android系统（最低到API 7）上使用这些新特性，就只能选择支持库版`SearchView`了。

代码清单25-6中的第二个定义项会添加一个Clear Search选项。由于`app:showAsAction`属性值设置为`never`，这个选项就只能出现在溢出菜单中。后面，我们会配置它，实现通过点击该选项来删除用户保存下来的搜索字符串。所以，暂时先忽略它。

现在来解决菜单XML中的未定义字符串错误。打开`strings.xml`文件，添加缺失的字符串，如代码清单25-7所示。

代码清单25-7 添加搜索字符串（res/values/strings.xml）

```

<resources>
    ...
    <string name="search">Search</string>
    <string name="clear_search">Clear Search</string>
</resources>

```

最后，在`PhotoGalleryFragment.java`文件中，在`onCreate(...)`方法中调用`setHasOptionsMenu(true)`方法让`fragment`接收菜单回调方法。然后，覆盖`onCreateOptionsMenu(...)`方法并实例化菜单XML文件，如代码清单25-8所示。这样，工具栏就能显示定义在菜单XML中的选项了。

代码清单25-8 覆盖`onCreateOptionsMenu(...)`方法（`PhotoGalleryFragment.java`）

```

public class PhotoGalleryFragment extends Fragment {
    ...
    @Override

```

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setRetainInstance(true);  
    setHasOptionsMenu(true);  
    new FetchItemsTask().execute();  
  
    ...  
}  
  
...  
  
@Override  
public void onDestroy() {  
    ...  
}  
  
@Override  
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {  
    super.onCreateOptionsMenu(menu, inflater);  
    inflater.inflate(R.menu.fragment_photo_gallery, menu);  
}  
  
private void setupAdapter() {  
    ...  
}  
  
...  
}
```

运行PhotoGallery看看SearchView的界面是什么样的。点击Search按钮，就能看到一个供用户输入的文本框，如图25-3所示。



图25-3 搜索界面

SearchView展开后，一个x按钮会出现在右边。点按它会删除用户输入文字。再次点按它，SearchView就会回到只有一个搜索按钮的界面。

现在尝试提交搜索不会有任何结果。不要担心，稍后SearchView就会有用了。

响应用户搜索

用户提交查询后，应用立即开始搜索Flickr网站，然后刷新显示搜索结果。查阅开发文档可知，SearchView.OnQueryTextListener接口已提供了接收回调的方式，可以响应查询指令。

更新onCreateOptionsMenu(...)方法，实现一个SearchView.OnQueryTextListener监听方法，如代码清单25-9所示。

代码清单25-9 日志记录SearchView.OnQueryTextListener事件（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {  
    ...  
  
    @Override  
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {  
        super.onCreateOptionsMenu(menu, menuInflater);  
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);  
  
        MenuItem searchItem = menu.findItem(R.id.menu_item_search);  
        final SearchView searchView = (SearchView) searchItem.getActionView();  
  
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {  
            @Override  
            public boolean onQueryTextSubmit(String s) {  
                Log.d(TAG, "QueryTextSubmit: " + s);  
                updateItems();  
                return true;  
            }  
  
            @Override  
            public boolean onQueryTextChange(String s) {  
                Log.d(TAG, "QueryTextChange: " + s);  
                return false;  
            }  
        });  
    }  
  
    private void updateItems() {  
        new FetchItemsTask().execute();  
    }  
  
    ...  
}
```

在onCreateOptionsMenu(...)方法中，我们首先从菜单中取出MenuItem并把它保存在

`searchItem`变量中。然后，使用`getActionView()`方法从这个变量中取出`SearchView`对象。

(提醒：API 11中就已经有了`MenuItem.getActionView()`方法。PhotoGallery应用最低SDK支持是API 16。然而，如果还想支持更早版本的系统，就应想其他办法获取`SearchView`对象。)

取到`SearchView`对象，就可以使用`setOnQueryTextListener(...)`方法设置`SearchView.OnQueryTextListener`了。另外，我们还必须覆盖监听器接口实现里的`onQueryTextSubmit(String)`和`onQueryTextChange(String)`方法。

只要`SearchView`文本框里的文字有变化（甚至是每个字符的改变），`onQueryTextChange(String)`回调方法就会执行。在PhotoGallery应用中，这个回调方法除了记日志以外不会干其他任何事。

用户提交搜索查询时，`onQueryTextSubmit(String)`回调方法就会执行。用户提交的搜索字符串会传给它。搜索请求受理后，该方法会返回`true`。这个方法也是启动`FetchItemsTask`搜索结果的地方。（现在`FetchItemsTask`里仍是一个硬编码的查询，我们稍后会更新这个方法以使用用户提交的查询请求。）

`updateItems()`方法现在还没多大用。稍后，会有好几个地方要执行`FetchItemsTask`。`updateItems()`就是一个调用`FetchItemsTask`的封装方法。

最后，在`onCreate(...)`方法中，删除创建和执行`FetchItemsTask`的那行代码，改用`updateItems()`封装方法，如代码清单25-10所示。

代码清单25-10 使用`updateItems()`封装方法（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        setHasOptionsMenu(true);
        new FetchItemsTask().execute();
        updateItems();

        ...
        Log.i(TAG, "Background thread started");
    }
    ...
}
```

运行应用并提交搜索查询。可以看到，虽然图片重新加载了，搜索结果仍然是基于代码清单25-5中的硬编码搜索字符串。另外，在日志中可以看出`SearchView.OnQueryTextListener`回调方法已成功执行。

注意：如果在模拟器上使用物理键盘提交查询（比如，笔记本的键盘），搜索会连续执行两次。从用户角度看，就是先看到下载的搜索结果，然后这些图片又全部重新加载了一次。这是

SearchView的一个bug。这个问题只会出现在模拟器上，所以可以不用管它。

25.3 使用 shared preferences 实现轻量级数据存储

现在，取代硬编码搜索字符串，我们来实现用户在SearchView中输入并提交的查询指令。

在PhotoGallery应用中，一次只有一个激活的查询。应用应该保存这个查询，即使应用或设备重启也不会丢失。要实现这个目标，可以把查询字符串写入shared preferences。只要用户提交查询，就把它写入shared preferences，覆盖掉之前保持的字符串。实际搜索Flickr时，就从shared preferences中取出查询字符串，把它作为text参数值。

shared preferences本质上就是文件系统中的文件，可使用SharedPreferences类读写它。SharedPreferences实例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。shared preferences文件保存在应用沙盒中，所以，类似密码这样的敏感信息不应该用它来保存。

要获得定制的SharedPreferences实例，可使用Context.getSharedPreferences(String,int)方法。然而，在实际开发中，我们并不关心具体是什么样的SharedPreferences实例，只要它能共享于整个应用就可以了。这种情况下，最好使用PreferenceManager.getDefaultSharedPreferences(Context)方法，该方法会返回具有私有权限和默认名称的实例（仅在当前应用内可用）。

如代码清单25-11所示，添加一个名为QueryPreferences的新类，用于读取和写入查询字符串。

代码清单25-11 管理保存的查询字符串 (QueryPreferences.java)

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";

    public static String getStoredQuery(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
            .getString(PREF_SEARCH_QUERY, null);
    }

    public static void setStoredQuery(Context context, String query) {
        PreferenceManager.getDefaultSharedPreferences(context)
            .edit()
            .putString(PREF_SEARCH_QUERY, query)
            .apply();
    }
}
```

PREF_SEARCH_QUERY用作查询字符串的存储key，读取和写入时都要用到它。

getStoredQuery(Context)方法返回shared preferences中保存的查询字符串值。不过，它首先要找到指定context中的默认SharedPreferences。（QueryPreferences类没有自己的

Context，所以该方法的调用者必须传入一个。)

取出查询字符串值非常简单，调用SharedPreferences.getString(...)就可以了。如果是其他类型数据，就调用对应的取值方法，比如getInt(...).SharedPreferences.getString(PREF_SEARCH_QUERY, null)方法的第二个参数指定默认返回值；如果找不到PREF_SEARCH_QUERY对应的值，就返回null值。

setStoredQuery(Context)方法向指定context的默认shared preferences写入查询输入值。在以上代码中，调用SharedPreferences.edit()方法，可获取一个SharedPreferences.Editor实例。它就是在SharedPreferences中保存查询信息要用到的类。与FragmentTransaction的使用类似，利用SharedPreferences.Editor，可将一组数据操作放入一个事务中。如有一批数据要更新，在一个事务中进行批量数据存储写入操作就可以了。

完成所有数据的变更准备后，调用SharedPreferences.Editor的apply()异步方法写入数据。这样，该SharedPreferences文件的其他用户就能看到写入的数据了。apply()方法首先在内存中执行数据变更，然后在后台线程上真正把数据写入文件。

QueryPreferences是PhotoGallery应用的数据存储引擎。既然已经搞定了查询信息的读取和写入方法，现在就来在PhotoGalleryFragment中应用它们。

首先是保存用户提交的查询信息，如代码清单25-12所示。

代码清单25-12 存储用户提交的查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {  
    ...  
    @Override  
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {  
        ...  
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {  
            @Override  
            public boolean onQueryTextSubmit(String s) {  
                Log.d(TAG, "QueryTextSubmit: " + s);  
                QueryPreferences.setStoredQuery(getActivity(), s);  
                updateItems();  
                return true;  
            }  
            @Override  
            public boolean onQueryTextChange(String s) {  
                Log.d(TAG, "QueryTextChange: " + s);  
                return false;  
            }  
        });  
    }  
    ...  
}
```

接下来，在用户从溢出菜单选择Clear Search选项时清除存储的查询信息（设置为null），如代码清单25-13所示。

代码清单25-13 清除查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_item_clear:
                QueryPreferences.setStoredQuery(getActivity(), null);
                updateItems();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
    ...
}
```

25

注意到了没有？和代码清单25-12中的做法一样，更新完查询信息，`updateItems()`方法会被调用。这很有必要，可以确保`RecyclerView`中显示最新的搜索结果。

最后，别忘了更新`FetchItemsTask`，来使用保存的查询字符串（终于可以不用硬编码字符串了）。在`FetchItemsTask`中添加一个定制版构造方法，用于接收查询信息并保存在一个成员变量中备用。更新`updateItems()`方法，从`shared preferences`中取出保存的查询信息，用它创建一个`FetchItemsTask`新实例，如代码清单25-14所示。

代码清单25-14 在`FetchItemsTask`中使用保存的查询信息（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    ...
    private void updateItems() {
        String query = QueryPreferences.getStoredQuery(getActivity());
        new FetchItemsTask(query).execute();
    }
    ...
    private class FetchItemsTask extends AsyncTask<Void, Void, List<GalleryItem>> {
        private String mQuery;
```

```

public FetchItemsTask(String query) {
    mQuery = query;
}

@Override
protected List<GalleryItem> doInBackground(Void... params) {
    String query = "robot"; // Just for testing

    if (query == null) {
        return new FlickrFetchr().fetchRecentPhotos();
    } else {
        return new FlickrFetchr().searchPhotos(query);
    }
}

@Override
protected void onPostExecute(List<GalleryItem> items) {
    mItems = items;
    setupAdapter();
}
}
}

```

搜索功能现在应该可以正常使用了。运行PhotoGallery应用，尝试进行一些搜索并查看返回结果。

25.4 优化应用

本章的任务完成了，可以考虑做点应用优化了。如果用户点击搜索按钮展开SearchView时，搜索文本框能显示已保存的查询字符串该多好。用户点击搜索按钮时，SearchView的View.OnClickListener.onClick()方法会被调用。利用这个回调方法设置搜索文本框的值，如代码清单25-15所示。

代码清单25-15 让SearchView文本框默认显示已保存查询信息（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {

    ...

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        ...

        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
            ...
        });

        searchView.setOnSearchClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

```

```
        String query = QueryPreferences.getStoredQuery(getActivity());
        searchView.setQuery(query, false);
    }
});

}

...
}
```

运行应用，尝试一些搜索。然后，检验一下刚才优化的成果。应用优化无法一蹴而就，关键是要做个有心人。

25.5 挑战练习：深度优化 PhotoGallery 应用

你可能已经注意到了，提交搜索时，`RecyclerView`要等好一会才能刷新显示搜索结果。请接受挑战，让搜索过程更流畅一些。用户一提交搜索，就隐藏软键盘，收起`SearchView`视图（回到只显示搜索按钮的初始状态）。

再来个挑战。用户一提交搜索，就初始化`RecyclerView`，显示一个搜索结果加载状态界面（使用状态指示器）。下载到JSON数据之后，就删除状态指示器。也就是说，一旦开始下载图片，就不应显示加载状态了。

第 26 章

后台服务

26

目前为止，本书所有的应用都离不开activity，意味着它们都有着一个或多个看得见的用户界面。

如果不给应用提供用户界面，应该怎样做呢？如果不用看、不用操作，只要任务在后台运行就行了，如播放音乐或在RSS feed上检查新博文推送，又该如何做呢？好办，使用服务（service）吧。

本章，我们将为PhotoGallery应用添加一项新功能，允许在后台下载新的搜索结果。一旦有了新的搜索结果，用户就能在状态栏接收到通知消息。

26.1 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，但可能是最常用的。创建一个名为PollService的IntentService子类，它就是用来轮询搜索结果的服务。

代码清单26-1 创建PollService（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    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，它和activity有点像。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)。为遵守良好的编程规范,我们添加了一个`newIntent(Context)`方法。无论谁想启动这个服务,都应使用`newIntent(...)`方法。

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。服务的种类不同,其执行命令的方式也不尽相同。

`IntentService`逐个执行命令队列里的命令,如图26-1所示。

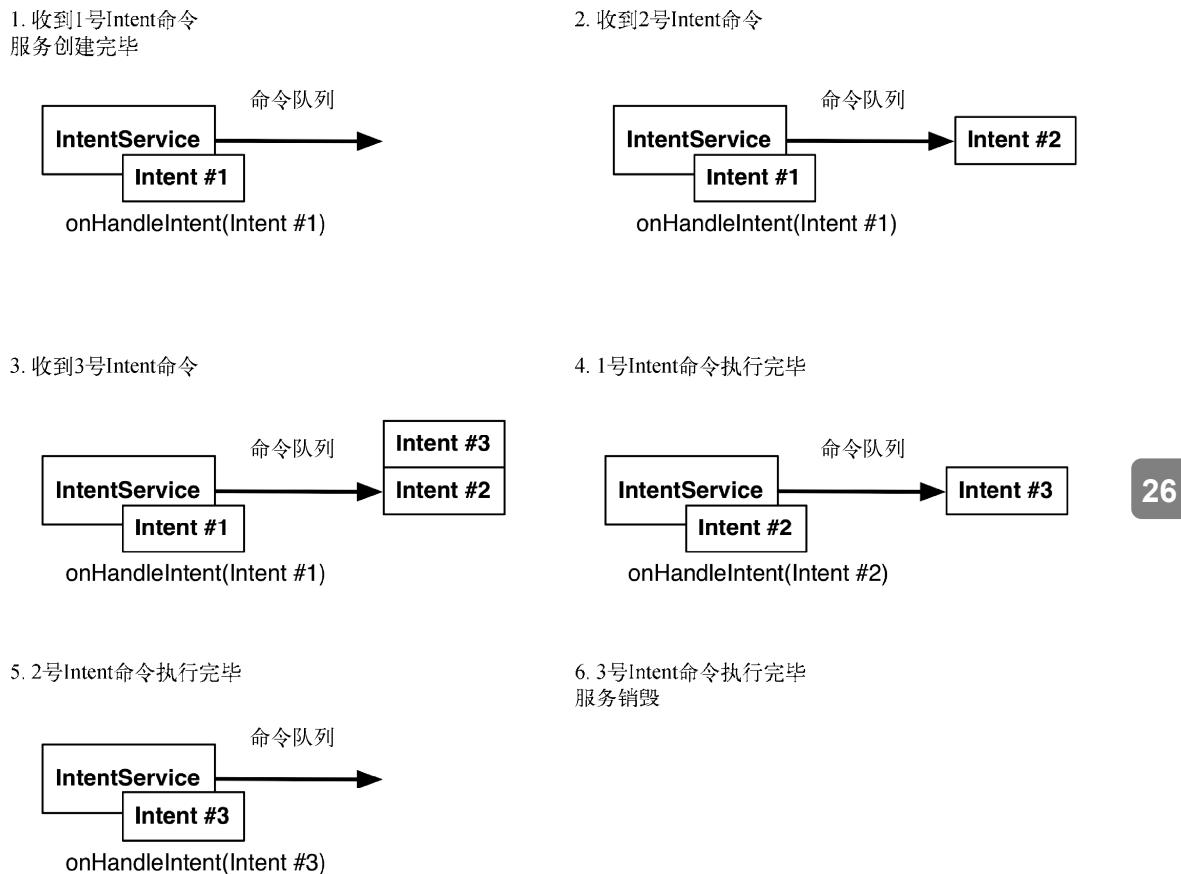


图26-1 `IntentService`执行命令的方式

接收到首个命令时,`IntentService`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,并针对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,执行完队列中的全部命令后,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IntentService的工作方式，你应该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，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26-2所示。

代码清单26-2 在manifest配置文件中添加服务（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.photogallery" >

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

    <application
        ...
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>
        <service android:name=".PollService" />
    </application>

</manifest>
```

然后，在PhotoGalleryFragment中，添加启动服务的代码，如代码清单26-3所示。

代码清单26-3 添加服务启动代码（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";
    ...

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

        Intent i = PollService.newIntent(getActivity());
        getActivity().startService(i);

        Handler responseHandler = new Handler();
        mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
        ...
    }

    ...
}
```

运行应用，查看LogCat窗口，可看到类似以下的结果。

```
02-23 14:25:32.450 2692-2717/com.bignerdranch.android.photogallery I/PollService:
Received an intent: Intent { cmp=com.bignerdranch.android.photogallery/.PollService }
```

26.2 服务的作用

查看LogCat日志是不是很乏味？确实是！但刚添加的代码着实令人兴奋！为什么？利用它可以完成什么？

再次回到之前的假想之地，我们也不再是应用开发者，而是与闪电侠一起工作的鞋店工作人员。

鞋店内有两个地方可以工作：与客户打交道的前台，以及不与客户接触的后台。取决于零售店的规模，工作后台可大可小。

目前为止，所有应用代码都在activity中运行。activity就是Android应用的前台。所有应用代码都专注于提供良好的用户视觉体验。

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消失好久了，后台服务依然可以持续不断地工作。

好了，鞋店的假想可以告一段落了。有服务可以做到，但activity却做不到的事情吗？有！用户离开当前应用后（打开其他应用或退回主屏幕），服务依然可以在后台运行。

安全的后台网络连接

服务将在后台轮询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。

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。参照代码清单26-4添加相应的检查代码。

代码清单26-4 检查后台网络的可用性（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    ...
    @Override
    protected void onHandleIntent(Intent intent) {
        if (!isNetworkAvailableAndConnected()) {
            return;
        }
        Log.i(TAG, "Received an intent: " + intent);
    }

    private boolean isNetworkAvailableAndConnected() {
        ConnectivityManager cm =
            (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);

        boolean isNetworkAvailable = cm.getActiveNetworkInfo() != null;
        boolean isNetworkConnected = isNetworkAvailable &&
            cm.getActiveNetworkInfo().isConnected();
```

```

        return isNetworkConnected;
    }

}

```

检查网络是否可用的逻辑在`isNetworkAvailableAndConnected()`方法中。使用后台数据设置选项关闭后台数据下载后，后台服务也就无法联网了。这样，`ConnectivityManager.getActiveNetworkInfo()`就会返回null值；这就相当于告诉后台服务网络不可用，即使设备实际是可以联网的。

如果后台服务能够使用网络，它会得到一个代表当前网络连接的`android.net.NetworkInfo`实例。然后还要调用`NetworkInfo.isConnected()`方法检查当前网络是否已完全连接。

如果应用找不到可用网络，或者设备没有完全连上网，`onHandleIntent(...)`方法就会直接返回（不会尝试去下载数据了，如果这个方法添加了数据下载代码的话）。这个做法是个好习惯：网都连不上，还谈什么数据下载。

不要忘了，要使用`getActiveNetworkInfo()`方法，还要在manifest配置文件中获取`ACCESS_NETWORK_STATE`权限，如代码清单26-5所示。

代码清单26-5 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" >

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

    <application
        ...
        ...
    </application>

</manifest>

```

26.3 查找最新返回结果

后台服务会一直查看最新返回结果，因此它要知道最近一次的获取结果。使用`SharedPreferences`保存结果值再合适不过了。

更新`QueryPreferences`以存储最近一次获取图片的ID，如代码清单26-6所示。

代码清单26-6 添加存储图片ID的preference常量（QueryPreferences.java）

```

public class QueryPreferences {

    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";

```

```

public static String getStoredQuery(Context context) {
    ...
}

public static void setStoredQuery(Context context, String query) {
    ...
}

public static String getLastResultId(Context context) {
    return PreferenceManager.getDefaultSharedPreferences(context)
        .getString(PREF_LAST_RESULT_ID, null);
}

public static void setLastResultId(Context context, String lastResultId) {
    PreferenceManager.getDefaultSharedPreferences(context)
        .edit()
        .putString(PREF_LAST_RESULT_ID, lastResultId)
        .apply();
}
}

```

接下来就是完善服务代码了。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，抓取第一条结果；
- 确认是否不同于上一次结果ID；
- 将第一条结果存入SharedPreferences。

回到PollService.java中，添加以上任务的实现代码。代码清单26-7中的代码虽长，但相信你已经很熟悉了，这里不再赘述。

代码清单26-7 检查最新返回结果（PollService.java）

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";

    ...

    @Override
    protected void onHandleIntent(Intent intent) {
        ...

        Log.i(TAG, "Received an intent: " + intent);
        String query = QueryPreferences.getStoredQuery(this);
        String lastResultId = QueryPreferences.getLastResultId(this);
        List<GalleryItem> items;

        if (query == null) {
            items = new FlickrFetchr().fetchRecentPhotos();
        } else {
            items = new FlickrFetchr().searchPhotos(query);
        }
    }
}

```

```

        if (items.size() == 0) {
            return;
        }

        String resultId = items.get(0).getId();
        if (resultId.equals(lastResultId)) {
            Log.i(TAG, "Got an old result: " + resultId);
        } else {
            Log.i(TAG, "Got a new result: " + resultId);
        }

        QueryPreferences.setLastResultId(this, resultId);
    }

    ...
}

```

看到前面任务清单的对应实现代码了吗？干的不错。

运行PhotoGallery应用，可看到应用首先获取了最新结果。如选择上一次的搜索查询，则提交搜索后，很可能会看到和上次同样的结果。

26.4 使用 AlarmManager 延迟运行服务

在没有activity运行的情况下，为在后台运行服务，得想个办法启动它。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。

Handler不行，我们还可以用AlarmManager。AlarmManager是可以发送Intent的系统服务。

如何将要发送的intent告诉AlarmManager呢？使用PendingIntent。使用PendingIntent打包一个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26-8所示。该方法是个静态方法。这样，定时器代码和与之相关的代码就可以放在一起了，同时，其他系统部件还可以调用到它。要知道，定时器通常是从前端的fragment或其他控制层代码中启停的。

代码清单26-8 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 * 60; // 60 seconds

    public static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }
}

```

```

    }

    public static void setServiceAlarm(Context context, boolean isOn) {
        Intent i = PollService.newIntent(context);
        PendingIntent pi = PendingIntent.getService(context, 0, i, 0);

        AlarmManager alarmManager = (AlarmManager)
            context.getSystemService(Context.ALARM_SERVICE);

        if (isOn) {
            alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
                SystemClock.elapsedRealtime(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }

    ...
}

```

以上代码中，首先是调用 `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，一个待发送的 `Intent` 对象以及一组用来决定如何创建 `PendingIntent` 的标志符。（稍后会使用其中的一个。）

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），定时器启动的时间，定时器循环的时间间隔以及一个到时要发送的 `PendingIntent`。

`AlarmManager.ELAPSED_REALTIME` 是基准时间值，这表明我们是以 `SystemClock.elapsedRealtime()` 走过的时间来确定何时启动时间的。也就是说，经过一段指定的时间，就启动定时器。假如使用 `AlarmManager.RTC`，启动基准时间就是当前时刻（例如，`System.currentTimeMillis()`）。也就是说，一旦到了某个固定时刻，就启动定时器。

取消定时器可调用 `AlarmManager.cancel(PendingIntent)` 方法。通常，也需同步取消 `PendingIntent`。稍后，你会看到取消 `PendingIntent` 是如何有助于跟踪定时器状态的。

添加一些快速测试代码，从 `PhotoGalleryFragment` 中启动 `PollService` 服务，如代码清单 26-9 所示。

代码清单26-9 添加定时器启动代码（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    ...

    @Override

```

```

public void onCreate(Bundle savedInstanceState) {
    ...
    updateItems();

    Intent i = PollService.newIntent(getActivity());
    getActivity().startService(i);
    PollService.setServiceAlarm(getActivity(), true);

    Handler responseHandler = new Handler();
    mThumbnailDownloader = new ThumbnailDownloader<>(responseHandler);
    ...
}

...
}

```

完成以上代码添加后，运行PhotoGallery应用。然后，立即点击后退键退出应用。

注意观察LogCat日志窗口。可看到PollService服务运行了一次，随后以60秒为间隔再次运行。这正是AlarmManager擅长的事情。即使进程停止了，AlarmManager依然会不断发送intent，反复启动PollService服务。（这种后台服务行为有时非常恼人。要彻底清除它，可能需要卸载应用。）

（如果嫌60秒太长，也可以设置较短的时间间隔。不过，本书编写时，Android 5.1设备支持的最小时间单位就是60秒。）

26.4.1 合理控制服务启动的频度

后台重复性工作的频度需要十分精确吗？后台服务重复性的工作会消耗电池电量和数据流量。而且，启动服务还要唤醒设备，这也是个开销极大的操作。幸运的是，这些都可以控制。我们可以采取措施合理地配置定时器的启停。比如，确定合理的时间间隔，设置唤醒条件等。

1. 精准和非精准重复

设置定时器的重复有两个方法可用：`AlarmManager.setRepeating(...)`和`AlarmManager.setInexactRepeating(...)`。

如果和PhotoGallery应用一样，对重复启动服务没有精确时间间隔要求，就可以把我们和其他应用的定时器放在一起管理。也就是说，后台搜索定时器不会按指定的时间间隔精准地启动。任务重复的真正时间间隔会有变化。既然系统可以批量处理定时器，唤醒设备的次数也就没那么频繁了。

直到KitKat 4.4 (API 19)，`setRepeating(...)`方法才实现以精准的时间间隔设置定时器。从名字可以看出，`setInexactRepeating(...)`方法是以不那么精准的时间间隔设置定时器的。如果指定一个预定义的时间间隔常量 (`INTERVAL_FIFTEEN_MINUTES`、`INTERVAL_HALF_HOUR`、`INTERVAL_HOUR`、`INTERVAL_HALF_DAY`或`INTERVAL_DAY`)，和预期一样，定时器就会不那么精准地运行。但是，如果指定一个定制的时间间隔，`setInexactRepeating(...)`方法的行为反而

就精准了。

自KitKat 4.4 (API 19) 开始，这两个方法的运行行为又有了变化。它们的运行效果变得一样了：都是以不那么精准的方法设置定时器。此外，无论是使用预定义常量时间间隔还是定制时间间隔，都没有区别了：两个方法都表现出了不精准的行为。

事实上，对于KitKat 4.4 (API 19) 及更高版本的系统，精准行为的概念也不复存在了。不过，我们可以改用**AlarmManager.setWindow(...)**和**AlarmManager.setExact(...)**这两个方法。利用它们也只能设置定时器精准地启动一次。

那么问题来了，如果一个应用不关心定时器的精确问题，开发人员该怎么做最好呢？如果应用仅面向KitKat 4.4 (API 19) 及更高版本的系统，请使用**setRepeating(...)**方法，时间间隔可以为任意时长。如果应用支持KitKat之前的设备，请使用**setInexactRepeating(...)**方法。而且，只要有可能，请尽量使用内置的时间间隔常量，以保证在所有的设备上都能获得不那么精准的定时器行为。

2. 时间基准选择

另一个要考虑的因素是时间基准值。Android提供了两个选项：**AlarmManager.ELAPSED_REALTIME**和**AlarmManager.RTC**。

AlarmManager.ELAPSED_REALTIME使用自最近一次设备重启（包括睡眠时间）开始走过的时间量作为间隔计算基准。既然是基于流逝的相对时间，并且不依赖于时钟时间，**ELAPSED_REALTIME**就成了PhotoGallery应用定时器的最佳选择。（查阅开发者文档可知，Android也首推使用**ELAPSED_REALTIME**。）

AlarmManager.RTC使用UTC时间。然而，UTC没有考虑本地时间的概念，而用户认为自己使用的时钟肯定已考虑了本地时间因素。所以，使用RTC作为定时器时间基准同样也要考虑本地时间因素。如果你想设置一个自然时间的定时器，就得自己处理本地时间和RTC基准时间的换算。嫌麻烦，那就干脆使用**ELAPSED_REALTIME**。

无论使用哪个时间基准值，如果设备处于睡眠模式（屏幕关掉状态），即使时间已过，定时器也不会触发。如果想避免这个问题，可使用与已选基准时间对应的**AlarmManager.ELAPSED_REALTIME_WAKEUP**和**AlarmManager.RTC_WAKEUP**常量，让定时器唤醒设备。然而，出于节能需要，应避免使用唤醒选项，除非需要绝对可靠的定时器任务。

26.4.2 PendingIntent

现在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()方法什么也做不了。

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。

26.4.3 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是否存在来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，而不是创建它。

添加一个名为isServiceAlarmOn(Context)的新方法，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26-10所示。

代码清单26-10 添加isServiceAlarmOn()方法（PollService.java）

```
public class PollService extends IntentService {
    ...
    public static void setServiceAlarm(Context context, boolean isOn) {
        ...
    }

    public static boolean isServiceAlarmOn(Context context) {
        Intent i = PollService.newIntent(context);
        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。

26.5 控制定时器

既然可以开关定时器（也能判定其启停状态），接下来就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26-11所示。

代码清单26-11 添加服务开关（menu/fragment_photo_gallery.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
```

```

    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/menu_item_search"
        ... />

    <item android:id="@+id/menu_item_clear"
        ... />

    <item android:id="@+id/menu_item_toggle_polling"
        android:title="@string/start_polling"
        app: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26-12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26-12 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">Start polling</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，添加菜单项实现代码，如代码清单26-13所示。

代码清单26-13 菜单项切换实现（PhotoGalleryFragment.java）

```

private static final String TAG = "PhotoGalleryFragment";

    ...

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

    PollService.setServiceAlarm(getActivity(), true);

    Handler responseHandler = new Handler();
    ...
}

    ...

@Override

```

```

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            QueryPreferences.setStoredQuery(getActivity(), null);
            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，就应该可以启停定时器了。然而，你可能已经注意到，即使定时器已经启动了，polling选项菜单也总是显示着Start polling。我们应该更新选项菜单，以准确反映定时器启停状态。

在onCreateOptionsMenu(...)方法中，检查定时器的开关状态，然后相应地更新menu_item_toggle_polling的标题文字，反馈正确的信息给用户，如代码清单26-14所示。

代码清单26-14 菜单项切换（PhotoGalleryFragment.java）

```

public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    ...

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater menuInflater) {
        super.onCreateOptionsMenu(menu, menuInflater);
        menuInflater.inflate(R.menu.fragment_photo_gallery, menu);

        MenuItem searchItem = menu.findItem(R.id.menu_item_search);
        final SearchView searchView = (SearchView) searchItem.getActionView();

        searchView.setOnQueryTextListener(...);

        searchView.setOnSearchClickListener(...);

        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);
        }
    }

    ...
}

```

接着，在`onOptionsItemSelected(MenuItem)`方法中，让`PhotoGalleryActivity`刷新工具栏选项菜单，如代码清单26-15所示。

代码清单26-15 让选项菜单失效（`PhotoGalleryFragment.java`）

```

...
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_clear:
            ...
        case R.id.menu_item_toggle_polling:
            boolean shouldStartAlarm = !PollService.isServiceAlarmOn(getActivity());
            PollService.setServiceAlarm(getActivity(), shouldStartAlarm);
            getActivity().invalidateOptionsMenu();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

...

```

现在，选项菜单切换应该可用了。不过，要使后台服务真正有用，还有一处需要完善。

26.6 通知信息

26

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，因此作用有限。

如果服务需要与用户沟通，通知信息（notification）总是一个不错的选择。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。

想要发送通知信息，首先要创建`Notification`对象。同第12章的`AlertDialog`类似，`Notification`需使用构造对象来创建。完整的`Notification`至少应包括：

- 在Lollipop之前的设备上，首次显示通知信息时，在状态栏上显示的ticker text（Lollipop之后，ticker text不再显示在状态栏上，但仍与可访问性服务相关）；
- 在状态栏上显示的图标（在Lollipop之前的设备上，图标在ticker text消失后出现）；
- 代表通知信息自身，在通知抽屉中显示的视图；
- 待触发的`PendingIntent`，用户点击抽屉中的通知信息时触发。

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。

首先是基础代码准备。在`PhotoGalleryActivity.java`中，添加一个`newIntent(...)`静态方法，如代码清单26-16所示。该静态方法会返回一个可用来启动`PhotoGalleryActivity`的`Intent`实例。（最后，`PollService`会调用这个方法，把返回结果封装在一个`PendingIntent`中，然后设置给通知消息。）

代码清单26-16 添加newIntent(...)静态方法（PhotoGalleryActivity.java）

```
public class PhotoGalleryActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context) {
        return new Intent(context, PhotoGalleryActivity.class);
    }

    @Override
    protected Fragment createFragment() {
        return PhotoGalleryFragment.newInstance();
    }
}
```

一旦有了新结果，就让PollService通知用户。也就是说，创建一个Notification对象，然后调用NotificationManager.notify(int, Notification)方法，如代码清单26-17所示。

代码清单26-17 添加通知信息（PollService.java）

```
...
@Override
protected void onHandleIntent(Intent intent) {
    ...

    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);

        Resources resources = getResources();
        Intent i = PhotoGalleryActivity.newIntent(this);
        PendingIntent pi = PendingIntent.getActivity(this, 0, i, 0);

        Notification notification = new NotificationCompat.Builder(this)
            .setTicker(resources.getString(R.string.new_pictures_title))
            .setSmallIcon(android.R.drawable.ic_menu_report_image)
            .setContentTitle(resources.getString(R.string.new_pictures_title))
            .setContentText(resources.getString(R.string.new_pictures_text))
            .setContentIntent(pi)
            .setAutoCancel(true)
            .build();

        NotificationManagerCompat notificationManager =
            NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);
    }

    QueryPreferences.setLastResultId(this, resultId);
}
```

我们来从上至下解读新增代码。首先，调用`setTicker(CharSequence)`和`setSmallIcon(int)`方法，配置ticker text和小图标。（注意，以`android.R.drawable.some_drawable_resource_name`包名形式引用的图标资源已内置于Android framework中，所以就没必要再单独放入资源文件夹了。）

然后配置Notification在下拉抽屉中的外观。虽然可以定制Notification视图的外观和样式，但使用带有图标、标题以及文字显示区域的标准视图会更容易些。图标的值来自于`setSmallIcon(int)`方法，而设置标题和显示文字则需分别调用`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，这里使用的是PendingIntent。用户在下拉抽屉中点击Notification消息时，传入`setContentIntent(PendingIntent)`方法的PendingIntent会被触发。调用`setAutoCancel(true)`方法可调整上述行为。一旦执行了`setAutoCancel(true)`设置方法，用户点击Notification消息时，该消息就会从消息抽屉中删除。

最后，从当前context中取出一个NotificationManagerCompat实例，然后调用`NotificationManagerCompat.notify(...)`方法贴出消息。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。如果使用同一ID发送两条消息，则第二条消息会替换掉第一条消息。在实际开发中，这也是进度条或其他动态视觉效果的实现方式。

本章任务到此结束了。运行应用并打开polling服务。不一会儿，应该就会看到状态栏的通知图标。拉开通知抽屉，就会看到后台服务发送的新结果消息。

既然后台服务已能正常工作，那就改用一个更为合理的定时器常量，如代码清单26-18所示。（使用AlarmManager的预定义时间间隔常量，可以保证在KitKat之前的设备上获得不那么精准的重复定时器行为。）

代码清单26-18 使用更为合理的定时器常量（PollService.java）

```
public class PollService extends IntentService {
    private static final String TAG = "PollService";

    public static final int POLL_INTERVAL = 1000 * 60; // 60 seconds
    private static final long POLL_INTERVAL = AlarmManager.INTERVAL_FIFTEEN_MINUTES;

    ...
}
```

26.7 挑战练习：可穿戴设备上的通知

创建和管理通知消息时，如果使用的是`NotificationCompat`和`NotificationManagerCompat`这两个类，通知信息会自动出现在已和手持设备配对的可穿戴设备上。用户接收到消息后，向左滑动，就能看到在手持设备上打开应用的选项。点击它，即可在手持设备上触发通知的pending intent。

创建一个Android可穿戴设备模拟器并与手持设备配对，测试一下这种通知行为。如果不知道怎么创建可穿戴设备模拟器，请访问<http://developer.android.com>寻求帮助。

26.8 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，因此有必要进一步了解并掌握服务，以便自己实现相关服务。作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。

26.8.1 服务的能与不能

与activity一样，服务是一个有生命周期回调方法的应用组件。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的最主要原因是大多数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。

下面来看看服务有哪些生命周期回调方法。

26.8.2 服务的生命周期

如果是startService(Intent)方法启动的服务，其生命周期很简单，并具有三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。它有两个整数参数，一个是标识符集，一个是启动ID。标识符集用来表示当前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。

服务停止时会调用onDestroy()方法。服务停止的方式取决于服务的类型。服务的类型由onStartCommand(...)方法的返回值确定，可能的服务类型有Service.START_NOT_STICKY、START_REDELIVER_INTENT和START_STICKY。

26.8.3 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。

`stopSelf(int)`是个有条件的方法。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`的后台工作原理。)

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`型服务说消亡就消亡了；而`START_REDELIVER_INTENT`型服务则会在资源不再吃紧时，尝试再次启动服务。

选择`START_NOT_STICKY`还是`START_REDELIVER_INTENT`，这要看服务对应用有多重要了。如果不重要，就选择`START_NOT_STICKY`。在`PhotoGallery`应用中，服务根据定时器的设定重复运行。即使发生问题，也不会有严重后果，因此应选择`START_NOT_STICKY`，同时，它也是`IntentService`的默认行为。如有需要，我们也可调用`IntentService.setIntentRedelivery(true)`方法，改用`START_REDELIVER_INTENT`。

26.8.4 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。为获得`sticky`服务，应返回`START_STICKY`。

`sticky`服务启动后会持续运行，除非某个组件调用`Context.stopService(Intent)`方法停止它。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，因为很难判断服务是否已启动。

26.8.5 绑定服务

除以上各类服务外，也可使用`bindService(Intent, ServiceConnection, int)`方法绑定一个服务，以此获得直接调用绑定服务方法的能力。`ServiceConnection`是代表服务绑定的一个对象。它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(getActivity(), MyService.class);
    getActivity().bindService(i, mServiceConnection, 0);
}

@Override
public void onDestroy() {
    super.onDestroy();
    getActivity().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：每次绑定服务时调用，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。

1. 本地服务绑定

MyBinder是怎样一种对象呢？如果服务是个本地服务，MyBinde很可能就是本地进程中一个简单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。服务是种高效的单例，与仅使用一个单例相比，使用此种模式显现不出优势。

2. 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中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Android文档中的AIDL或Messenger类，了解更多相关内容。

26.9 深入学习：JobScheduler 和 JobService

本章，我们已知道如何让AlarmManager、IntentService和PendingIntent相互配合，创建周期性的后台服务。实现一个完全可用的后台服务还需要手动执行以下操作：

- 计划一个周期性任务；

- 检查周期性任务的运行状态；
- 检查网络是否可用；

不过，在实际应用场景下，还有更多想法要实现。例如，如果请求失败，是否还需要稍后重试机制；或者为了省钱，只允许应用使用不限流量的网络连接；甚至是只在设备充电的时候，才允许联网检查是否有新图片。想法虽然可以成真，但具体实现起来却没那么容易。

在系统控制方面，本章实现的后台服务也存在一些问题。例如，服务启动后，即使发现没事情可做，它也得继续运行。因为没法给它下达指令：“在这种情况下停下来！”此外，为了保证在设备重启后，服务依然能按设定运行，开发人员还要做一些额外工作。（下一章，在接收到`BOOT_COMPLETED broadcast intent`时，你就会看到这些额外的工作。）

尽管存在上述问题，我们还是选择了本章的解决方案，因为`PollService`中用到的API也支持旧版本系统。而在Lollipop（API 21）系统中，为更好地实现后台服务，Android引入了一个叫作`JobScheduler`的全新API。除了实现常规后台服务之外，`JobScheduler`还支持按场景、按条件运行后台服务。

下面就来看看它的工作原理。首先，我们创建一个处理任务的服务（使用`JobService`子类）。`JobService`有两个可覆盖方法：`onStartJob(JobParameters)`和`onStopJob(JobParameters)`。（注意，以下代码仅供讨论之用，请勿直接在应用中使用。）

```
public class PollService extends JobService {
    @Override
    public boolean onStartJob(JobParameters params) {
        return false;
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        return false;
    }
}
```

Android准备好执行任务时，服务就会启动，此时会在主线程上收到`onStartJob(...)`方法调用。该方法返回`false`结果表示：“交代的任务我已全力去做，现在做完了。”返回`true`结果则表示：“任务收到，正在做，但是还没有做完。”

与`IntentService`不同，`JobService`需要单开新线程，这点比较麻烦。可使用`AsyncTask`按如下方式创建新线程：

```
private PollTask mCurrentTask;

@Override
public boolean onStartJob(JobParameters params) {
    mCurrentTask = new PollTask();
    mCurrentTask.execute(params);
    return true;
}

private class PollTask extends AsyncTask<JobParameters,Void,Void> {
```

```

@Override
protected Void doInBackground(JobParameters... params) {
    JobParameters jobParams = params[0];

    // Poll Flickr for new images

    jobFinished(jobParams, false);
    return null;
}
}

```

任务完成后，就可以调用`jobFinished(JobParameters, boolean)`方法通知结果。不过，如果该方法的第二个参数传入`true`的话，就等于说：“事情这次做不完了，请计划在下次某个时间继续吧。”

`onStopJob(JobParameters)`方法适合在中断任务时调用。用户通常需要服务在有Wi-Fi连接时才运行。如果在调用`JobFinished(...)`之前（任务完成之前），手机就离开了Wi-Fi覆盖区，`onStopJob(...)`方法就会被调用，也就是说，一切任务就立即停止了。

```

@Override
public boolean onStopJob(JobParameters params) {
    if (mCurrentTask != null) {
        mCurrentTask.cancel(true);
    }
    return true;
}

```

调用`onStopJob(...)`方法就是表明，服务马上就要停掉了。不要抱有幻想，请立即停止手上的一切事情。这里，返回`true`表示：“任务应该计划在下次继续。”返回`false`表示：“不管怎样，事情就到此结束吧，不要计划下次了。”

在manifest配置文件中登记服务时，必须导出它并为它添加权限：

```

<service
    android:name=".PollService"
    android:permission="android.permission.BIND_JOB_SERVICE"
    android:exported="true"/>

```

所谓导出服务，就是把服务暴露出来，但添加的权限控制只有`JobScheduler`才能运行它。

一旦创建了`JobService`，启动它就非常迅速了。我们可以使用`JobScheduler`检查是否已计划好了任务。

```

final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);

boolean hasBeenScheduled = false;
for (JobInfo jobInfo : scheduler.getAllPendingJobs()) {
    if (jobInfo.getId() == JOB_ID) {
        hasBeenScheduled = true;
    }
}

```

如果还没有，可以创建一个新的JobInfo说明我们期望的任务运行时间。嗯，PollService应该在什么时候运行呢？这样如何：

```
final int JOB_ID = 1;

JobScheduler scheduler = (JobScheduler)
    context.getSystemService(Context.JOB_SCHEDULER_SERVICE);

JobInfo jobInfo = new JobInfo.Builder(
    JOB_ID, new ComponentName(context, PollService.class))
    .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
    .setPeriodic(1000 * 60 * 15)
    .setPersisted(true)
    .build();
scheduler.schedule(jobInfo);
```

上述代码计划任务每15分钟运行一次，但前提条件是有Wi-Fi或有可用的不限流量网络。调用setPersisted(true)方法可保证服务在设备重启后也能按计划运行。还可采用其他方式配置JobInfo，具体做法请参阅Android开发者文档。

26.10 深入学习：Sync Adapter

我们还可以使用sync adapter创建常规的polling网络服务。和前面看到过的adapter不一样，sync adapter主要用于从某个数据源同步数据（上传、下载或既上传又下载）。不像JobScheduler，sync adapter早就存在了，所以不用担心系统版本的新旧问题。

和JobScheduler一样，sync adapter可代替PhotoGallery应用中的AlarmManger。不同应用中的同步功能都是默认一起执行的。而且即使设备重启，也不用重置同步定时器，因为sync adapter会自动完成。

sync adapter还能和操作系统完美整合。我们可以设置一个可同步账户，对外暴露应用。然后，用户通过Settings → Accounts菜单来管理应用的同步。当然，这个菜单还可以用来管理其他使用sync adapter的应用账户，如Google自己的一些应用套件，如图26-2所示。

虽然使用sync adapter既能让周期性的网络任务变得容易可靠，又能让开发者免于编写定时器管理和pending intent的相关代码，但开发者仍然免不了写另外一些代码。首先，需要与网络请求相关的代码（如FlickFetchr）。其次，要有一个content provider实现来封装数据、账户和授权类，用以代表远程服务器的某个账户（即使远程服务器不需要授权），以及一个sync adapter和sync service的实现。另外，还要懂得运用绑定服务。

所以，如果应用已经使用content provider作为数据层，并且需要账号授权，那使用sync adapter是最理想的。此外，相较于JobScheduler，sync adapter还有同操作系统提供的用户界面自然整合的优势。考虑到这些因素，即使要写一大堆代码，某些场景下，仍然值得使用sync adapter。

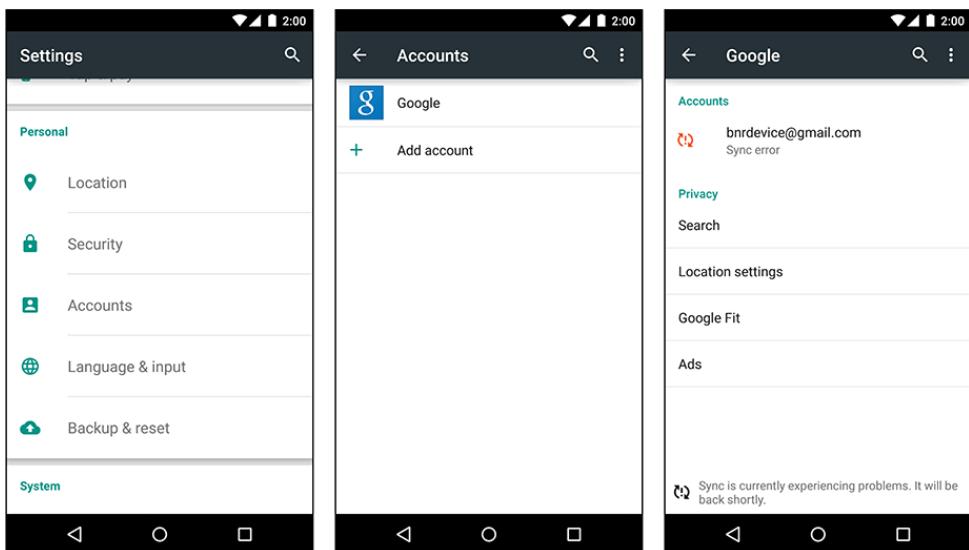


图26-2 账户设置

在线开发者文档提供了sync adapter使用教程，请访问以下网页进行学习：<https://developer.android.com/training/sync-adapters/index.html>。

26.11 挑战练习：在 Lollipop 设备上使用 JobService

请创建另一个PollService实现版本。新的PollService应该继承JobService并使用JobScheduler来运行。在PollService启动代码中，检查系统版本是否为Lollipop：如果是，就使用JobScheduler计划运行JobService；如果不是，依然使用AlarmManager实现。



本章，我们继续从两个方面完善PhotoGallery应用。首先，让应用轮询新结果并在有所发现时及时通知用户，即使用户重启设备后还没有打开过应用。其次，保证用户在使用应用时不出现新结果通知。（用户打开应用，看到了应用界面刷新显示的新的搜索图片，同时还收到了新结果通知，你说是不是既多余又烦人？）

在进行这些优化的过程中，我们将学习如何监听系统发送的broadcast intent，以及如何使用broadcast receiver处理它们。此外，我们会在运行的应用中动态地发送与接收broadcast intent。最后，还会使用有序broadcast判断应用是否正在前台运行。

27.1 一般 intent 和 broadcast intent

Android设备中，各种事件一直在频繁地发生。Wi-Fi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，等等。

许多系统组件需要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的工作原理类似于之前学过的intent，唯一不同的是broadcast intent可同时被多个叫作broadcast receiver的组件接收（如图27-1所示）。

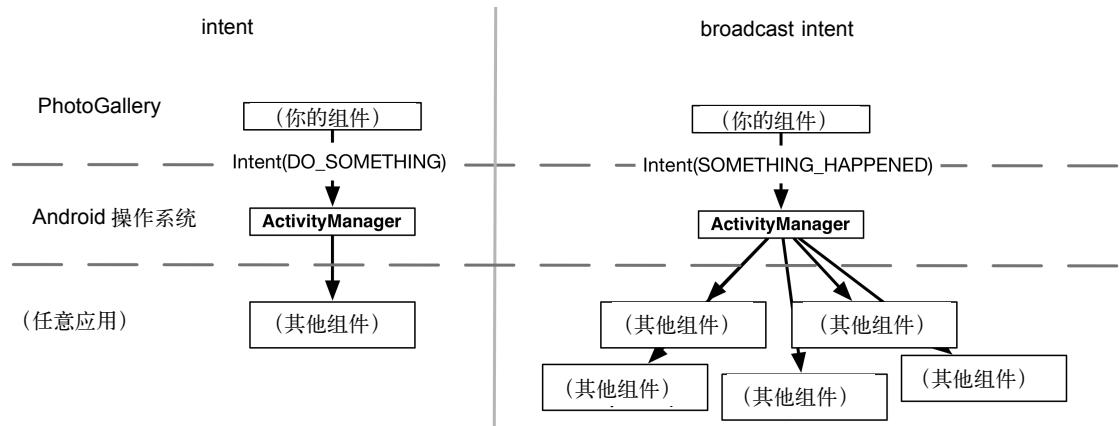


图27-1 一般intent与broadcast intent

作为公共API的一部分，无论什么时候，activity和服务都应该响应隐式intent。但大多数场景下，使用显式intent就够了。然而，broadcast intent之所以存在，最大的理由是它可以发送给多个接收者。这样看来，虽然broadcast receiver能响应显式intent，但几乎不会这么使用，因为显式intent只有一个接收者。

27.2 接收系统 broadcast：重启后唤醒

PhotoGallery应用的后台定时器虽然可以正常工作，但还不够完美。如果用户重启了设备，定时器就会失效。

设备重启后，那些持续运行的应用通常也需要重启。通过监听带有BOOT_COMPLETED操作的broadcast intent，可获知设备是否已完成启动。只要打开设备，系统就会发送一个BOOT_COMPLETED broadcast intent。要想监听它，可以创建并登记一个standalone broadcast receiver。

27.2.1 standalone receiver

standalone receiver是一个在manifest配置文件中声明的broadcast receiver。即便应用进程已消亡，standalone receiver也可以被激活。（稍后还会学习到可以同fragment或activity的生命周期绑定的dynamic receiver。）

与服务和activity一样，broadcast receiver必须在系统中登记后才能发挥作用。如果不登记，系统就不知道该向哪里发送intent。自然，broadcast receiver的onReceive(...)方法也就得不到预定的调用了。

要登记broadcast receiver，首先要创建它。创建一个StartupReceiver新类，继承android.content.BroadcastReceiver类，如代码清单27-1所示。

代码清单27-1 第一个broadcast receiver（StartupReceiver.java）

```
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());
    }
}
```

与服务和activity一样，broadcast receiver是接收intent的组件。当有intent发送给StartupReceiver时，它的onReceive(...)方法会被调用。

打开AndroidManifest.xml配置文件，参照代码清单27-2登记上StartupReceiver。

代码清单27-2 在manifest文件中添加receiver（AndroidManifest.xml）

```
<manifest ...>

<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
        android:name=".PhotoGalleryActivity"
        android:label="@string/app_name">
        ...
    </activity>

    <service android:name=".PollService"/>

    <receiver android:name=".StartupReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
        </intent-filter>
    </receiver>
</application>

</manifest>

```

登记响应隐式intent的standalone receiver和登记服务或activity差不多。我们使用receiver标签并在其中包含相应的intent-filter。StartupReceiver会监听BOOT_COMPLETED操作，而该操作也需要配置使用权限。因此，还需要添加一个相应的uses-permission标签。

在配置文件中完成声明后，即使应用并未运行，只要有匹配的broadcast intent发来，broadcast receiver就会醒来接收。一收到intent，broadcast receiver的onReceive(Context, Intent)方法即开始运行，随后会被销毁（如图27-2所示）。

27

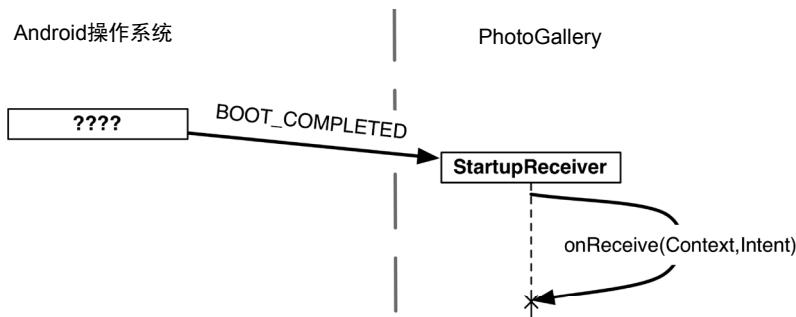


图27-2 接收BOOT_COMPLETED

设备重启后，StartupReceiver的onReceive(...)方法会被调用吗？现在就来验证。首先，运行更新版PhotoGallery应用。

然后，关闭设备。如果是物理设备，直接按电源键关机。如果是模拟器，最简单的方法是直接退出模拟器应用。

打开设备。如果是物理设备，直接按电源键开机。如果是模拟器，要么重新运行应用，要么

使用AVD Manager启动应用，但要保证使用的是刚关掉的那个模拟器。

现在，选择Tools → Android → Android Device Monitor菜单项打开Android Device Monitor。

(KitKat之前，Android Device Monitor常被称为Dalvik Debug Monitor Server或DDMS。Dalvik是Android上的运行时系统。自KitKat开始，Google又引入了ART(Android Runtime)。到了Lollipop，就只剩下ART可用了。虽然现在已改名为Android Device Monitor，但旧的名字仍时有耳闻。)

点击Android Device Monitor的Devices选项页中的设备。(如果看不到设备列表，请尝试插拔USB设备或重启模拟器。)

在Android Device Monitor窗口中，以Received broadcast intent关键字搜索LogCat输出(如图27-3所示)。

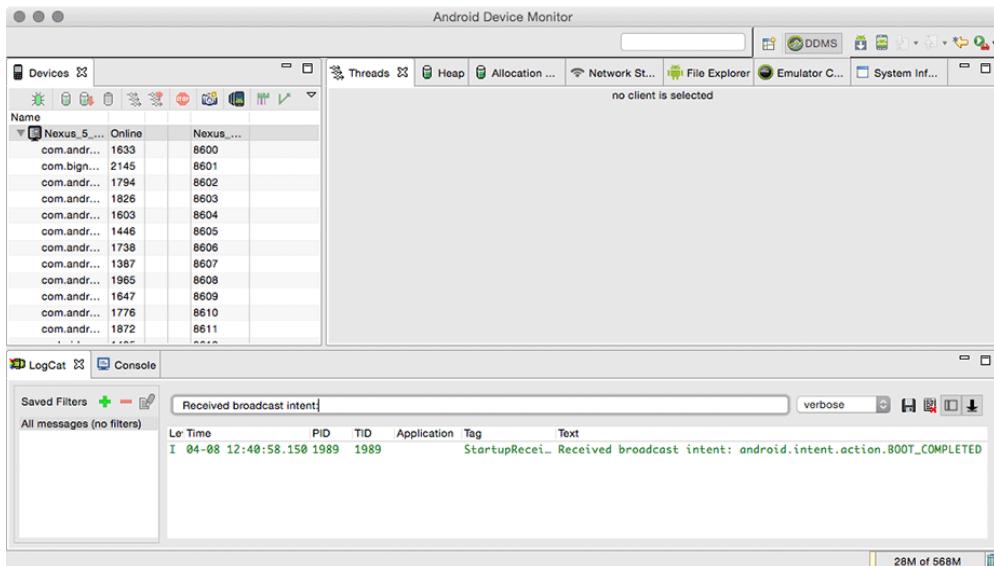


图27-3 搜索LogCat输出

可以在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进程。这是因为进程在运行broadcast receiver之后，就随即消亡了。

(使用Logcat输出测试receiver的运行不一定总是成功，尤其是在模拟器上。按照上述步骤操作，如果看不到log日志，建议多试几次。还是不行的话，那就暂时放弃，等学习到优化通知消息时，就能以其他更方便的办法来验证receiver的运行了。)

27.2.2 使用 receiver

broadcast receiver的生命非常短暂，因而难以有所作为。例如，我们无法使用任何异步API或登记任何监听器，因为一旦onReceive(Context, Intent)方法运行完，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费时费力

的任务，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。一些便利型任务就非常适合它，比如启动activity或服务（不需要返回结果），以及系统重启后重置定时运行的定时器。

receiver需要知道定时器的启停状态。为存储它的状态，在QueryPreferences类中添加一个preference常量和两个便利方法，如代码清单27-3所示。

代码清单27-3 添加定时器状态preference (QueryPreferences.java)

```
public class QueryPreferences {
    private static final String PREF_SEARCH_QUERY = "searchQuery";
    private static final String PREF_LAST_RESULT_ID = "lastResultId";
    private static final String PREF_IS_ALARM_ON = "isAlarmOn";

    ...

    public static void setLastResultId(Context context, String lastResultId) {
        ...
    }

    public static boolean isAlarmOn(Context context) {
        return PreferenceManager.getDefaultSharedPreferences(context)
            .getBoolean(PREF_IS_ALARM_ON, false);
    }

    public static void setAlarmOn(Context context, boolean isOn) {
        PreferenceManager.getDefaultSharedPreferences(context)
            .edit()
            .putBoolean(PREF_IS_ALARM_ON, isOn)
            .apply();
    }
}
```

接下来，更新PollService.setServiceAlarm(...)方法，在设置定时器后存下它的状态，如代码清单27-4所示。

代码清单27-4 存储定时器状态 (PollService.java)

```
public class PollService extends IntentService {
    ...

    public static void setServiceAlarm(Context context, boolean isOn) {
        ...

        if (isOn) {
            alarmManager.setInexactRepeating(AlarmManager.ELAPSED_REALTIME,
                SystemClock.elapsedRealtime(), POLL_INTERVAL, pi);
        } else {
            alarmManager.cancel(pi);
            pi.cancel();
        }
    }
}
```

```

        QueryPreferences.setAlarmOn(context, isOn);
    }

    ...
}

```

设备重启后，`StartupReceiver`就能用它打开定时器，如代码清单27-5所示。

代码清单27-5 设备重启后启动定时器（`StartupReceiver.java`）

```

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());

        boolean isOn = QueryPreferences.isAlarmOn(context);
        PollService.setServiceAlarm(context, isOn);
    }
}

```

再次运行PhotoGallery应用（为方便测试，可设置一个较短的时间间隔，如60秒）。点击工具栏的Start polling打开服务。重启设备。这次，后台polling服务应该重启了。

27.3 过滤前台通知消息

解决了设备重启后的服务唤醒问题，再来看PhotoGallery应用的另一缺陷。通知消息虽然很有用，但应用开着的时候不应该收到通知消息。

同样，我们使用broadcast intent来解决这个问题，但用法和以前完全不同。

首先，我们发送（或接收）定制版broadcast intent（最后会锁定它，只允许PhotoGallery应用部件接收它）。其次，不再使用manifest文件，改用代码为broadcast intent动态登记receiver。最后，发送一个有序broadcast在一组receiver中传递数据，借此保证最后才运行某个receiver。（不太理解这段话？别担心，跟着做，很快就会明白了。）

27.3.1 发送 broadcast intent

首先处理最容易的部分：发送自己定制的broadcast intent。具体来讲，就是发送broadcast通知目标部件有新的搜索结果消息了。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，输入代码清单27-6所示代码。

代码清单27-6 发送broadcast intent（`PollService.java`）

```

public class PollService extends IntentService {
    private static final String TAG = "PollService";

```

```

private static final long POLL_INTERVAL = AlarmManager.INTERVAL_FIFTEEN_MINUTES;

public static final String ACTION_SHOW_NOTIFICATION =
        "com.bignerdranch.android.photogallery.SHOW_NOTIFICATION";
...

@Override
protected void onHandleIntent(Intent intent) {

    ...

    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        ...

        NotificationManagerCompat notificationManager =
            NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION));
    }

    QueryPreferences.setLastResultId(this, resultId);
}

...
}

```

现在，只要有新结果，应用就会对外广播。

27

27.3.2 动态 broadcast receiver

完成intent的发送后，接下来的任务是使用receiver接收ACTION_SHOW_NOTIFICATION broadcast intent。

可以编写一个类似于StartupReceiver的standalone broadcast receiver来接收intent，并在manifest文件中登记；但这里行不通。我们需要在PhotoGalleryFragment存在的时候接收发过来的intent。在配置文件中声明的standalone receiver会不断地接收intent，而且它还要设法知道PhotoGalleryFragment的状态（这是个难点）。

使用动态broadcast receiver能解决问题。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，可调用registerReceiver(BroadcastReceiver, IntentFilter)方法；取消登记时，则调用unregisterReceiver(BroadcastReceiver)方法。receiver自身通常被定义为一个内部类实例，如同一个按钮点击监听器。然而，在registerReceiver(...)和unregisterReceiver(...)方法中，我们要的是同一个实例，因此需要将receiver赋值给一个实例变量。

新建一个VisibleFragment抽象类，继承Fragment类，如代码清单27-7所示。该类是一个

隐藏前台通知的通用型fragment。(在第28章，我们还会编写一个类似的fragment。)

代码清单27-7 VisibleFragment自己的receiver (VisibleFragment.java)

```
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";

    @Override
    public void onStart() {
        super.onStart();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter);
    }

    @Override
    public void onStop() {
        super.onStop();
        getActivity().unregisterReceiver(mOnShowNotification);
    }

    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();
        }
    };
}
```

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均能以代码的方式定义。要在代码中配置IntentFilter，可以直接调用addCategory(String)、addAction(String)和addDataPath(String)等方法。

使用动态登记的broadcast receiver时，要记得事后清理。通常，如果在启动生命周期方法中登记了一个receiver，就应在相应的停止方法中调用Context.unregisterReceiver(BroadcastReceiver)方法。这里，我们在onResume()方法里登记，在onPause()方法里撤销登记。同样，如果在onActivityCreated(...)方法里登记，就应在onActivityDestroyed()里撤销登记。

(顺便要说的是，我们应注意在保留fragment中的onCreate(...)和onDestroy()方法的使用。设备旋转时，onCreate(...)和onDestroy()方法中的getActivity()方法会返回不同的值。因此，

如果想在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。)

接下来，修改PhotoGalleryFragment类，转而继承新的VisibleFragment，如代码清单27-8所示。

代码清单27-8 设置fragment为可见（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends FragmentVisibleFragment {
    ...
}
```

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图标，如图27-4所示。

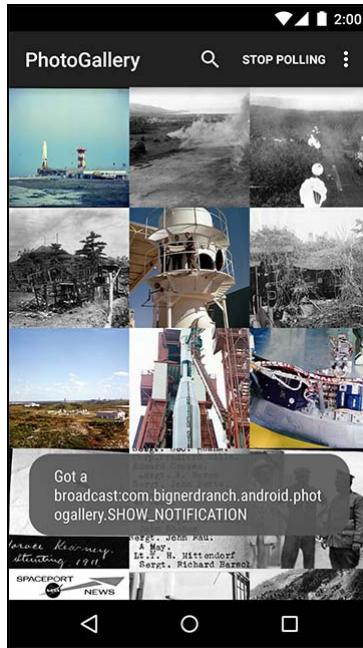


图27-4 验证broadcast的存在

27.3.3 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，有多种方式可以阻止未授权应用闯入我们的私人领域。一种办法是在manifest配置文件里给receiver标签添加一个`android:exported="false"`属性，声明它仅限应用内部使用。这样，系统中的其他应用就再也无法接触到该receiver了。

另外，也可创建自己的使用权限。这可以通过在AndroidManifest.xml中添加一个**permission**标签来完成。这就是PhotoGallery应用要用到的办法。

在AndroidManifest.xml配置文件中，声明并获取自己的使用权限，如代码清单27-9所示。

代码清单27-9 添加私有权限（AndroidManifest.xml）

```
<manifest ...>

    <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，我们使用**protection level**签名定义了自己的定制权限。稍后，还会学习到更多有关**protection level**的知识。如同前面用过的**intent**操作、类别和系统权限，权限本身只是一行简单的字符串。即使是自定义的权限，也必须在使用这个权限前获取它，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，并且要保证完全一致。因此，最好使用复制粘贴功能，而不是手动输入。

现在，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27-10所示。

代码清单27-10 发送带有权限的broadcast（PollService.java）

```
public class PollService extends IntentService {
    ...

    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 static Intent newIntent(Context context) {
        return new Intent(context, PollService.class);
    }

    ...

    @Override
    protected void onHandleIntent(Intent intent) {
        ...
    }
}
```

```

String resultId = items.get(0).getId();
if (resultId.equals(lastResultId)) {
    Log.i(TAG, "Got an old result: " + resultId);
} else {
    ...
    notificationManager.notify(0, notification);
    sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
}

QueryPreferences.setLastResultId(this, resultId);
}

...
}

```

要使用权限，须将其作为参数传入`sendBroadcast(...)`方法。有了这个权限，所有应用都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来触发它。同样，在`registerReceiver(...)`方法中传入自定义权限就能解决该问题，如代码清单27-11所示。

代码清单27-11 broadcast receiver的使用权限（VisibleFragment.java）

```

public abstract class VisibleFragment extends Fragment {
    ...

    @Override
    public void onStart() {
        super.onStart();
        IntentFilter filter = new IntentFilter(PollService.ACTION_SHOW_NOTIFICATION);
        getActivity().registerReceiver(mOnShowNotification, filter,
            PollService.PERM_PRIVATE, null);
    }

    ...
}

```

现在，只有photoGallery应用才能够触发目标receiver了。

深入学习安全级别

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。在PhotoGallery应用中，我们使用的`protectionLevel`是`signature`。

`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，选择`signature`安全级别比较合适。既然其他开发者没有相同的key，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。

表27-1 protectionLevel的可选值

可选值	用法描述
normal	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，用户可以看到相应的安全级别，但用户不会被明确要求给予授权。 <code>android.permission.RECEIVE_BOOT_COMPLETED</code> 使用该安全级别。同样，应用让手机振动时，也使用该安全级别
dangerous	用于normal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，包括一切可能会给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dangerous权限级别时，Android会明确要求用户确认是否授权
signature	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，系统会拒绝授权。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，因此开发者可自由控制权限的使用。前例中，它用来阻止其他应用监听PhotoGallery应用发出的broadcast。不过，如有需要，可开发能够监听它们的其他应用
signatureOrSystem	类似signature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。权限授予时，系统不会通知用户。开发人员一般不会用到它

27.3.4 使用有序 broadcast

最后，我们的任务是保证动态登记的receiver总是先于其他receiver接收到PollService.`ACTION_SHOW_NOTIFICATION` broadcast。然后，还要修改这个broadcast，制止通知消息的发布。

现在，虽然可以发送个人私有的broadcast了，但目前还只是只发不收的单向通信，如图27-5所示。

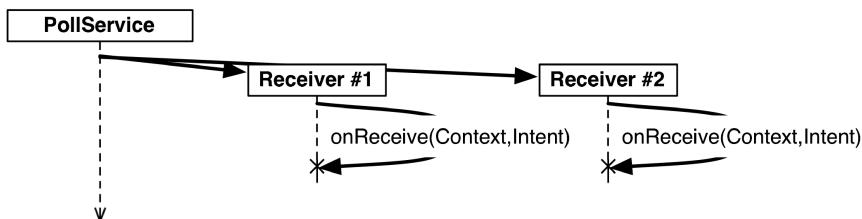


图27-5 一般broadcast intent

这是因为，从概念上讲，一般broadcast intent可同时被所有人接收。事实上，`onReceive(...)`方法是在主线程上调用的，所以receiver并没有同步并发运行。因而，不可能指望它们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。结果就是，无论是broadcast receiver之间要通信，还是intent发送者要从receiver接收信息，都会很麻烦。

为解决问题，可使用有序broadcast intent实现双向通信（如图27-6所示）。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还支持让broadcast发送者接收broadcast接收者的返回结果。

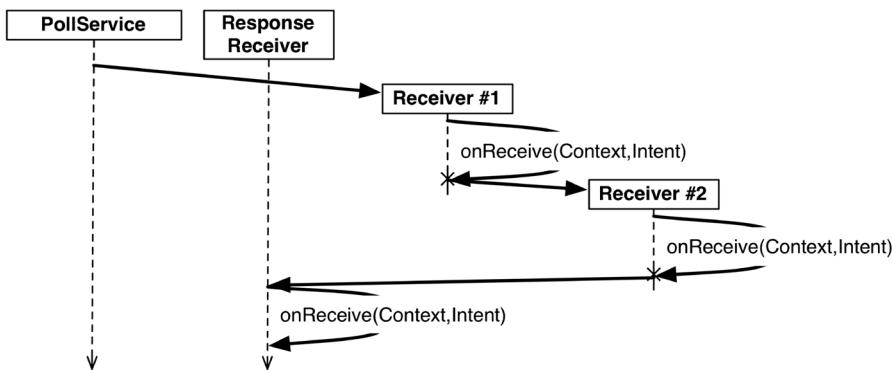


图27-6 有序broadcast intent

从接收方来看，这看上去与一般broadcast没什么不同。然而，我们却因此获得了特别的工具：一套改变receiver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型结果码，将此要求告诉信息发送者。稍后，我们会使用 setResultCode(int)方法，设置一个Activity.RESULT_CANCELED结果码。

修改VisibleFragment类，告诉SHOW_NOTIFICATION的发送者应该如何处置通知消息，如代码清单27-12所示。这个信息也会发送给接收链中的所有broadcast receiver。

代码清单27-12 返回一个简单结果码（VisibleFragment.java）

```

27
public abstract class VisibleFragment extends Fragment {
    private static final String TAG = "VisibleFragment";

    private BroadcastReceiver mOnShowNotification = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getActivity(),
                "Get 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);
        }
    };
    ...
}
  
```

此处要的只是YES或NO指示，因此使用int结果码即可。如需返回更多复杂数据，可调用 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它们。

为了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27-13所示。该方法打包一个Notification调用，然后作为一个broadcast发出。在onHandleIntent(...)方法中，删除原来直接发布通知给NotificationManager的代码，调用这个新方法发出一个有序broadcast。

代码清单27-13 发送有序broadcast (PollService.java)

```

...
public static final String PERM_PRIVATE =
    "com.bignerdranch.android.photogallery.PRIVATE";
public static final String REQUEST_CODE = "REQUEST_CODE";
public static final String NOTIFICATION = "NOTIFICATION";
...

@Override
protected void onHandleIntent(Intent intent) {
    ...

    String resultId = items.get(0).getId();
    if (resultId.equals(lastResultId)) {
        Log.i(TAG, "Got an old result: " + resultId);
    } else {
        Log.i(TAG, "Got a new result: " + resultId);
        ...

        Notification notification = ...;

        NotificationManagerCompat notificationManager =
            NotificationManagerCompat.from(this);
        notificationManager.notify(0, notification);

        sendBroadcast(new Intent(ACTION_SHOW_NOTIFICATION), PERM_PRIVATE);
        showBackgroundNotification(0, notification);
    }

    QueryPreferences.setLastResultId(this, resultId);
}

private 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接收者结束运行后，它才开始运行。虽然有时能使用result receiver接收broadcast和发布通知对象，但此处行不通。目标broadcast intent通常是在PollService对象消亡之前发出的，这意味着broadcast receiver很可能也不存在了。

因此，最终的broadcast receiver应该是个standalone receiver。而且，无论如何都要保证它在其他动态登记的receiver之后运行。

首先，新建一个NotificationReceiver类，继承BroadcastReceiver类，如代码清单27-14所示。

代码清单27-14 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(PollService.REQUEST_CODE, 0);
        Notification notification = (Notification)
            i.getParcelableExtra(PollService.NOTIFICATION);

        NotificationManagerCompat notificationManager =
            NotificationManagerCompat.from(c);
        notificationManager.notify(requestCode, notification);
    }
}
```

然后，登记这个新建的receiver并赋予优先级。为保证NotificationReceiver最后一个接收目标broadcast（这样，它就知道该不该向NotificationManager发出通知），需要为它设置一个低优先级。要让它最后一个运行，设置其优先级值为-999，这是用户能定义的最低优先级（-1000及以下值是系统保留值）。

另外，既然NotificationReceiver仅限PhotoGallery应用内部使用，还需设置一个`android:exported="false"`属性值，以确证外部应用看不到它，如代码清单27-15所示。

代码清单27-15 登记notification receiver (AndroidManifest.xml)

```
<manifest ...>
    ...
    <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>

```

运行PhotoGallery应用，多次切换后台polling状态。可以看到，应用开着的时候，通知信息不会出现了。(为避免傻等15分钟，可再次将PollService.POLL_INTERVAL的时间间隔设置为60秒。)

27.4 receiver 与长时运行任务

如不想受限于主线程的时间限制，希望broadcast intent触发一个长时运行任务，该怎么做呢？有两种方法可以选择。

- 将任务交给服务去处理，然后通过broadcast receiver启动服务。这是我们首推的方式。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请求。
- 使用BroadcastReceiver.goAsync()方法。该方法返回一个BroadcastReceiver.PendingResult对象，随后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。

goAsync()方法的弊端是不够灵活。我们仍需快速响应broadcast(10秒内)，并且与使用服务相比，没什么架构模式好选择。

当然，goAsync()方法也有明显的优势：可调用该方法设置有序broadcast的结果。如果真的要使用，应注意控制它的运行时长。

27.5 深入学习：本地事件

broadcast intent可实现系统内全局性的消息传递。如果仅需要应用内的消息事件广播，该怎么做呢？答案是使用事件总线(event bus)。

事件总线的设计思路就是，提供一个应用内的部件可以订阅的共享总线或数据流。事件一旦发布到总线上，各订阅部件就会被激活并执行相应的回调代码。

由greenrobot出品的EventBus是目前广为人知的一个第三方事件总线库。其他常用的事件总线库还有Square的Otto。也可以使用RxJava Subject和Observable来模拟事件总线。

为实现在应用内发送broadcast intent，Android自己也提供了一个叫作LocalBroadcastManager的广播管理类；但上述第三方类库用起来更为灵活和方便。

27.5.1 使用 EventBus

要在应用中使用EventBus，首先需要在项目中添加依赖库。然后，就可以定义事件类了（如果需要传送数据，可以向事件里添加数据字段）：

```
public class NewFriendAddedEvent { }
```

在应用的任何地方，都可以把消息事件发布到总线上：

```
EventBus eventBus = EventBus.getDefault();
eventBus.post(new NewFriendAddedEvent());
```

在总线上登记监听，应用的其他部分也可以订阅接收事件消息。通常，activity或fragment的登记和撤销登记都是在相应的生命周期方法中处理的，如onStart(...)和onStop(...)

```
// In some fragment or activity...
private EventBus mEventBus;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mEventBus = EventBus.getDefault();
}

@Override
public void onStart() {
    super.onStart();
    mEventBus.register(this);
}

@Override
public void onStop() {
    super.onStop();
    mEventBus.unregister(this);
}
```

有订阅的事件消息发布时，可实施onEvent(...)或者onEventMainThread(...)方法并传入合适的事件类型，让订阅者作出响应。如果使用的是onEvent(...)方法，事件消息来自哪个线程，就在哪个线程上处理。（可以实施onEventMainThread(...)方法确保事件在主线程上处理，哪怕它碰巧来自后台线程。）

```
// In some registered component, like a fragment or activity...
public void onEventMainThread(NewFriendAddedEvent event){
    Friend newFriend = event.getFriend();
    // Update the UI or do something in response to event...
}
```

27.5.2 使用 Rxjava

Rxjava也能用来实现事件广播机制。Rxjava库可用来开发reactive风格的Java代码。上述reactive概念有深广的含义，不在本书讨论之列。简而言之，就是可以发布和订阅各类事件，并且

还有很多通用工具用来管理这些事件。

所以，我们可以创建一个叫**Subject**的对象，然后发布事件给它以及在其上订阅事件。

```
Subject<Object, Object> eventBus = new SerializedSubject<>(PublishSubject.create());
```

可以像下面这样发布事件给它：

```
Friend someNewFriend = ...;
NewFriendAddedEvent event = new NewFriendAddedEvent(someNewFriend);
eventBus.onNext(event);
```

并且在其上订阅事件：

```
eventBus.subscribe(new Action1<Object>() {
    @Override
    public void call(Object event) {
        if (event instanceof NewFriendAddedEvent) {
            Friend newFriend = ((NewFriendAddedEvent)event).getFriend();
            // Update the UI
        }
    }
})
```

RxJava解决方案的优势在于，`eventBus`现在也是个**Observable**对象（代表RxJava的事件流）了。这就意味着RxJava的各种事件管理工具都可以为我们所用了。是不是愈发感兴趣了？那就看看RxJava的项目wiki主页吧：<https://github.com/ReactiveX/RxJava/wiki>。

27.6 深入学习：检测 fragment 的状态

本章，在实现PhotoGallery应用的通知功能时，我们使用了全局性的broadcast机制。`broadcast`虽然是全局的，但我们利用定制权限，限定**broadcast intent**只能在应用内接收。这就不免让人疑惑：“既然要限制，为什么还要使用全局机制？使用本地**broadcast**机制不是更好吗？”

这是因为有个难题要解决：如何判断**PhotoGalleryFragment**的存在状态。最终，利用有序**broadcast**、**standalone receiver**以及动态登记的**receiver**，问题总算解决了。虽然没那么干净利落，但这就是Android目前能提供的最好解决方案。

更具体来讲，就是**LocalBroadcastManager**既无法处理PhotoGallery应用里的这种**broadcast**通知，也无法知晓**fragment**的状态。如果要进一步分析，原因不外乎两点。

首先，**LocalBroadcastManager**不支持有序**broadcast**（虽然它有个**sendBroadcastSync(Intent intent)**方法，但依然不灵），而在PhotoGallery应用中，不使用有序**broadcast**，就无法控制**NotificationReceiver**最后一个运行。

其次，**sendBroadcastSync(Intent intent)**方法不支持在独立线程上发送和接收**broadcast**。而在PhotoGallery应用中，需要在后台线程上发送**broadcast**（使用**PollService.onHandleIntent(...)**方法），在主线程上接收**intent**（在主线程上的**onStart(...)**方法中，使用由**PhotoGalleryFragment**登记的动态**receiver**）。

本书撰写时，关于**LocalBroadcastManager**究竟是如何处理线程上的**broadcast**投递的，

Android还没有确切的说明；但经验告诉我们，它是有规律可循的。例如，如果从后台线程调用 `sendBroadcastSync(...)`，所有的pending broadcast都会在后台线程上涌出，不管是不是来自主线程。

当然，`LocalBroadcastManager`也不是一无是处，只不过不适合解决本章的问题而已。

第 28 章

网页浏览

28

从Flickr下载的图片都有对应的关联网页。本章继续升级PhotoGallery应用，让用户点击图片就能看到它的Flickr网页。我们会以两种不同的方式整合网页内容，完成后的效果如图28-1所示：左边是使用浏览器应用，右边是使用WebView在应用中显示网页内容。

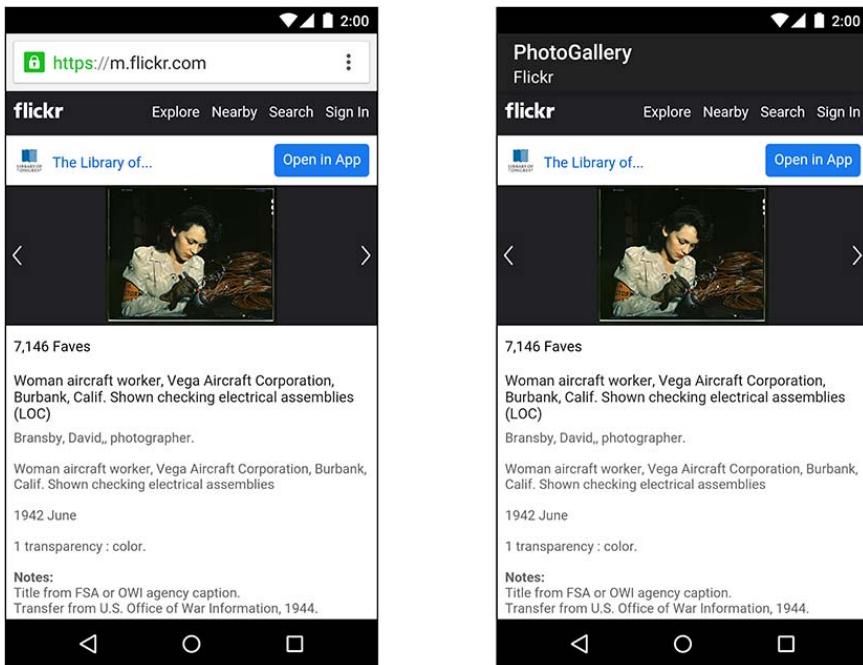


图28-1 以两种方式呈现Web内容

28.1 最后一段 Flickr 数据

无论哪种方式，都需要取得图片的Flickr网页URL。如果查看下载图片的JSON文件，可看到图片的网页地址并不包含在内。

```
{
  "photos": {
    ...
    "photo": [
      {
        "id": "9452133594",
        "owner": "44494372@N05",
        "secret": "d6d20af93e",
        "server": "7365",
        "farm": 8,
        "title": "Low and Wisoff at Work",
        "ispublic": 1,
        "isfriend": 0,
        "isfamily": 0,
        "url_s": "https://farm8.staticflickr.com/7365/9452133594_d6d20af93e_m.jpg"
      }, ...
    ],
    "stat": "ok"
  }
}
```

因此，我们想当然地认为需要编码获取更多JSON内容才行。实际上，并不是这样的。访问 <http://www.flickr.com/services/api/misc.urls.html> 查看 Flickr 官方文档的 Web Page URLs 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 JSON 数据里的 id 属性值。该值已保存在 GalleryItem 类的 mId 属性中。那么 user-id 呢？继续查阅 Flickr 文档可知，JSON 文件的 owner 属性值就是用户 ID。因此，只需从 JSON 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 28-1 所示代码，创建图片 URL。

28

代码清单 28-1 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 Uri getPhotoPageUri() {
    return Uri.parse("http://www.flickr.com/photos/")
        .buildUpon()
        .appendPath(mOwner)
        .appendPath(mId)
        .build();
}

@Override
public String toString() {
    return mCaption;
}
}

```

以上代码新建了一个mOwner属性，以及一个产生图片URL的getPhotoPageUri()方法。现在，修改parseItems(...)方法，从JSON数据中获取owner属性，如代码清单28-2所示。

代码清单28-2 从JSON数据中获取owner属性 (FlickrFetchr.java)

```

public class FlickrFetchr {
    ...

    private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
        throws IOException, JSONException {

        JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
        JSONArray photoJSONArray = photosJsonObject.getJSONArray("photo");

        for (int i = 0; i < photoJSONArray.length(); i++) {
            JSONObject photoJsonObject = photoJSONArray.getJSONObject(i);

            GalleryItem item = new GalleryItem();
            item.setId(photoJsonObject.getString("id"));
            item.setCaption(photoJsonObject.getString("title"));

            if (!photoJsonObject.has("url_s")) {
                continue;
            }

            item.setUrl(photoJsonObject.getString("url_s"));
            item.setOwner(photoJsonObject.getString("owner"));
            items.add(item);
        }
    }
}

```

非常简单，获取图片网页URL的任务就完成了。

28.2 简单方式：隐式 intent

我们首先使用隐式intent这个老朋友来访问图片URL。隐式intent可启动浏览器，并在其中打开图片URL指向的网页。

首先，监听 RecyclerView 显示项的点击事件。更新 PhotoGalleryFragment 类的 PhotoHolder，实现一个可以发送隐式intent的事件监听方法：

代码清单28-3 通过隐式intent实现网页浏览（PhotoGalleryFragment.java）

```
public class PhotoGalleryFragment extends VisibleFragment {
    ...

    private class PhotoHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {
        private ImageView mItemImageView;
        private GalleryItem mGalleryItem;

        public PhotoHolder(View itemView) {
            super(itemView);

            mItemImageView = (ImageView) itemView
                .findViewById(R.id.fragment_photo_gallery_image_view);
            itemView.setOnClickListener(this);
        }

        public void bindDrawable(Drawable drawable) {
            mItemImageView.setImageDrawable(drawable);
        }

        public void bindGalleryItem(GalleryItem galleryItem) {
            mGalleryItem = galleryItem;
        }

        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.getPhotoPageUri());
            startActivity(i);
        }
    }

    ...
}
```

28

然后，在 PhotoAdapter.onBindViewHolder(...) 方法中绑定 PhotoHolder 和 GalleryItem。

代码清单28-4 绑定GalleryItem（PhotoGalleryFragment.java）

```
...
private class PhotoAdapter extends RecyclerView.Adapter<PhotoHolder> {
```

```

    ...
    @Override
    public void onBindViewHolder(PhotoHolder photoHolder, int position) {
        GalleryItem galleryItem = mGalleryItems.get(position);
        photoHolder.bindGalleryItem(galleryItem);
        Drawable placeholder = getResources().getDrawable(R.drawable.bill_up_close);
        photoHolder.bindDrawable(placeholder);
        mThumbnailDownloader.queueThumbnail(photoHolder, galleryItem.getUrl());
    }
    ...
}
...

```

搞定了。启动PhotoGallery应用并点击任意图片。浏览器应用应该会弹出并加载显示对应的图片网页（类似图28-1的左边）。

28.3 较难方式：使用 WebView

使用隐式intent打开图片网页简单又高效。但是，如果不想打开独立的浏览器怎么办？

通常，我们只想在activity中显示网页内容，而不是打开浏览器：或许是想显示自己生成的HTML，或许是想以某种方式限制用户使用浏览器。对于大多数需要帮助文档的应用，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。打开浏览器查看帮助文档，既不专业，又妨碍应用行为的定制，同时也无法将网页整合进自己的用户界面。

下面就介绍我们的第二个方式：使用WebView类。虽然这里把使用WebView类归为一种较难的实现方式，但实际使用并不困难。（相对隐式intent来说，要困难一些。）

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28-2所示。

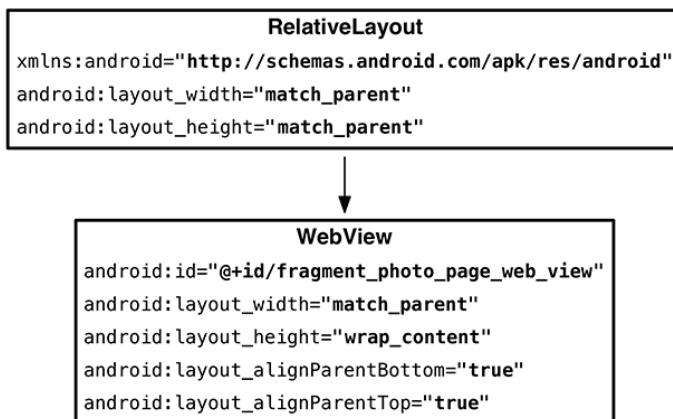


图28-2 初始布局（res/layout/fragment_photo_page.xml）

是不是认为这里的RelativeLayout起不了什么作用？确实如此，稍后，我们会添加更多的组件来完善它。

接下来创建fragment。新建PhotoPageFragment类，继承上一章的VisibleFragment类。然后，在这个新类中，实例化布局文件，从中引用WebView，并转发从intent数据中获取的URL，如代码清单28-5所示。

代码清单28-5 创建网页浏览fragment (PhotoPageFragment.java)

```

public class PhotoPageFragment extends VisibleFragment {
    private static final String ARG_URI = "photo_page_url";

    private Uri mUri;
    private WebView mWebView;

    public static PhotoPageFragment newInstance(Uri uri) {
        Bundle args = new Bundle();
        args.putParcelable(ARG_URI, uri);

        PhotoPageFragment fragment = new PhotoPageFragment();
        fragment.setArguments(args);
        return fragment;
    }

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

        mUri = getArguments().getParcelable(ARG_URI);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mWebView = (WebView) v.findViewById(R.id.fragment_photo_page_web_view);

        return v;
    }
}

```

28

当前，PhotoPageFragment类还未完成，稍后再来完成它。接下来，新建PhotoPageActivity托管类，继承SingleFragmentActivity类，如代码清单28-6所示。

代码清单28-6 创建显示网页的activity (PhotoPageActivity.java)

```

public class PhotoPageActivity extends SingleFragmentActivity {

    public static Intent newIntent(Context context, Uri photoPageUri) {
        Intent i = new Intent(context, PhotoPageActivity.class);
        i.setData(photoPageUri);
        return i;
    }
}

```

```

    }

    @Override
    protected Fragment createFragment() {
        return PhotoPageFragment.newInstance(getIntent().getData());
    }
}

```

回到PhotoGalleryFragment类中，弃用隐式intent，启动新建的activity，如代码清单28-7所示。

代码清单28-7 启动新建的activity (PhotoGalleryFragment.java)

```

public class PhotoGalleryFragment extends VisibleFragment {
    ...

    private class PhotoHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener{
        ...

        @Override
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_VIEW, mGalleryItem.getPhotoPageUri());
            Intent i = PhotoPageActivity
                .newIntent(getActivity(), mGalleryItem.getPhotoPageUri());
            startActivity(i);
        }
    }

    ...
}

```

最后，在配置文件中声明新建的activity，如代码清单28-8所示。

代码清单28-8 在配置文件中声明activity (AndroidManifest.xml)

```

<manifest ... >
    ...

    <application
        ...>
        <activity
            android:name=".PhotoGalleryActivity"
            android:label="@string/app_name" >
            ...
        </activity>

        <activity android:name=".PhotoPageActivity" />

        <service android:name=".PollService" />

        ...
    </application>

</manifest>

```

运行PhotoGallery应用，点击任意图片，可看到一个新的空activity弹出。

好了，现在来处理关键部分，让fragment发挥其作用。WebView要成功显示Flickr图片网页，需做三件事。

首先是告诉WebView要打开的URL。

其次是启用JavaScript。JavaScript默认是禁用的。虽然并不总是需要启用它，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），可以使用`@SuppressLint("SetJavaScriptEnabled")`注解`onCreateView(...)`方法以禁止Lint的警告。）

最后，需要覆盖`WebViewClient`类的`shouldOverrideUrlLoading(WebView, String)`方法，并返回`false`值。添加代码清单28-9所示代码。然后，我们来详细解读`PhotoPageFragment`类。

代码清单28-9 加载URL（PhotoPageFragment.java）

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

        mWebView = (WebView) v.findViewById(R.id.fragment_photo_page_web_view);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebViewClient(new WebViewClient() {
            public boolean shouldOverrideUrlLoading(WebView view, String url) {
                return false;
            }
        });
        mWebView.loadUrl(mUri.toString());

        return v;
    }
}
```

加载URL必须等`WebView`配置完成后进行，因此最后再执行这一操作。在此之前，首先调用`getSettings()`方法获得`WebSettings`实例，再调用`WebSettings.setJavaScriptEnabled(true)`方法，从而启用JavaScript。`WebSettings`是修改`WebView`配置的三种途径之一。另外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，配置`WebViewClient`。`WebViewClient`是一个事件接口。可自己实现`WebViewClient`来响应各种渲染事件。例如，可检测渲染器何时开始从指定URL加载图片，或决定是否需要向服务器重新提交POST请求。

`WebViewClient`有多个方法可覆盖，其中大多数都用不到。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`时，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，就表示：“不要处理这个URL，我自己来。”如返回`false`值，就是说：“`WebView`，去加载这个URL，我什么也不会做的。”

正如本章前面的做法，默认的实现会发送附有URL数据的隐式intent。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，就意味着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。

运行应用，点击任意图片，应该可以看到显示对应图片的WebView（类似图28-1的右边）。

使用WebChromeClient 优化 WebView 显示

既然花时间实现了自己的WebView，接下来开始优化，为它添加一个标题视图和一个进度条。打开fragment_photo_page.xml，添加如代码清单28-10所示的更新代码。

代码清单28-10 添加标题和进度条（fragment_photo_page.xml）

```
<RelativeLayout ...>

    <ProgressBar
        android:id="@+id/fragment_photo_page_progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:visibility="gone"
        style="?android:attr/progressBarStyleHorizontal"
        android:background="?attr/colorPrimary"/>

    <WebView
        android:id="@+id/fragment_photo_page_web_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true" />
        android:layout_alignParentBottom="true"
        android:layout_below="@+id/fragment_photo_page_progress_bar" />

</RelativeLayout>
```

为关联使用ProgressBar，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就是一个事件接口，用来响应那些改变浏览器中装饰元素的事件。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，编码实现WebChromeClient的关联使用，如代码清单28-11所示。

代码清单28-11 使用WebChromeClient（PhotoPageFragment.java）

```
public class PhotoPageFragment extends VisibleFragment {
    ...
    private WebView mWebView;
    private ProgressBar mProgressBar;
```

```
...
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_photo_page, container, false);

    mProgressBar =
        (ProgressBar)v.findViewById(R.id.fragment_photo_page_progress_bar);
    mProgressBar.setMax(100); // WebChromeClient reports in range 0-100

    mWebView = (WebView) v.findViewById(R.id.fragment_photo_web_view);
    mWebView.getSettings().setJavaScriptEnabled(true);
    mWebView.setWebChromeClient(new WebChromeClient() {
        public void onProgressChanged(WebView webView, int newProgress) {
            if (newProgress == 100) {
                mProgressBar.setVisibility(View.GONE);
            } else {
                mProgressBar.setVisibility(View.VISIBLE);
                mProgressBar.setProgress(newProgress);
            }
        }
        public void onReceivedTitle(WebView webView, String title) {
            AppCompatActivity activity = (AppCompatActivity) getActivity();
            activity.getSupportActionBar().setSubtitle(title);
        }
    });
    mWebView.setWebViewClient(new WebViewClient() {
        ...
    });
    mWebView.loadUrl(mUri.toString());
}

return v;
}
}
```

进度条和标题栏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.GONE`，将`ProgressBar`视图隐藏起来。

运行PhotoGallery应用，测试刚才更新的代码。应看到类似图28-3的应用画面。

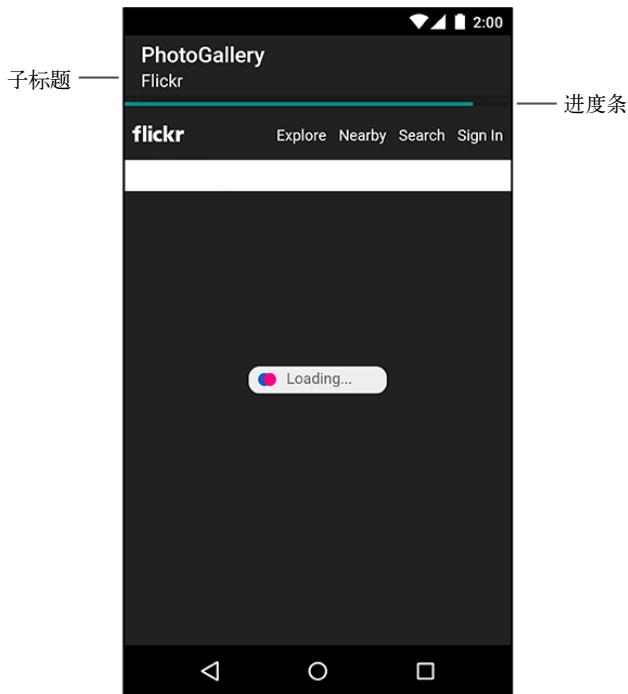


图28-3 漂亮的WebView

点击任意图片，`PhotoPageActivity`会弹出。网页加载时，会出现进度条，工具栏会出现来自`onReceivedTitle(...)`方法的子标题。页面加载完毕，进度条就会随即消失。

28.4 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`包含太多的数据，无法在`onSaveInstanceState(...)`方法内全部保存。所以每次设备旋转，它都必须从头开始加载网页数据。

是不是想到了`PhotoPageFragment`保留？不好意思，这里行不通。因为`WebView`是视图层级结构的一部分，所以旋转后它肯定会销毁并重建。

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`，就能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。

为了让`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28-12所示。

代码清单28-12 自己处理设备配置更改（AndroidManifest.xml）

```
<manifest ... >
...
<activity
    android:name=".PhotoPageActivity"
    android:configChanges="keyboardHidden|orientation|screenSize" />
...
</manifest>
```

`android:configChanges`属性表明，如果因键盘开或关、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，一切都完美了。

自己处理配置更改的风险

自己处理设备配置更改，我们轻松搞定了WebView的设备旋转问题。既然这么简单，为什么不全面推广使用这个方法呢？实际上，事情没那么简单，自己处理配置变更也是有风险的。

首先，资源修饰符无法自动工作了。开发人员必须手工重载视图。这实际是非常棘手的。

其次，也是更重要的一点，既然activity自己处理配置更改了，你很可能不会去覆盖Activity.`onSavedInstanceState(...)`方法存储UI状态。然而，这依然是必须的，即使自己处理设备配置更改也是一样。因为低内存情况下的生死还是要考虑的。（还记得吗？activity不运行的时候，系统可能会销毁并暂存它的状态，如第3章图3-13中看到的那样。）

28.5 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，还可以做到更多。查阅文档网页<http://developer.android.com/reference/android/webkit/WebView.html>，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() {
    @JavascriptInterface
    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>
```

自Jelly Bean 4.2 (API 17) 开始，只有以@JavascriptInterface注解的公共方法才会暴露给JavaScritp。在这之前，所有对象树中的公共方法都是开放访问的。

这可能有风险，因为一些可能的问题网页能够直接接触到应用。安全起见，要么自己掌控局面，要么严格控制不要暴露自己的接口。

28.6 深入学习：KitKat 的 WebView

基于Chromium开源项目，随KitKat 4.4(API 19)发布的WebView经历了一次大修。新WebView使用了和Android版Chrome一样的渲染引擎。现在，两者的页面外观和浏览器行为越来越趋于统一了。（然而，WebView并不具有Android Chrome的全部特性。查看网页：<https://developer.chrome.com/multidevice/webview/overview>，可看到它们的对照表。）

转向Chromium给WebView带来了一系列激动人心的改进，比如支持HTML5和CSS3新网页标准，一个全新的JavaScript引擎以及增强的网页展示性能。从开发者的角度看，一个最令人兴奋的新特性就是，WebView终于支持使用Chrome DevTools进行远程调试了（调用WebView.setWebContentsDebuggingEnabled()方法开启）。

如果应用需要支持KitKat之前的设备呢？那就尤其要关注新旧版WebView的一些较大差异。例如，如果是非本地网页内容（指远程服务器而非本机提供的网页），那么就不允许和内容提供者交互。另外，定制URL scheme的处理也会更严格。

如果应用的目标SDK版本低于API 19，WebView在尽量为用户带来性能改进和网页标准支持的同时，会避免使用KitKat中引入的新特性（这种行为常被称为怪异模式或兼容模式）。然而，问题还远不止这些。例如，默认的放大级别在API 19甚至更高的设备上就完全不支持了。

总之，如果应用打算支持KitKat之前的设备并且依赖WebView，就要好好研究KitKat之前和之后版本的差异。Android开发者网站上有一份很好的WebView使用迁移指南：<http://developer.android.com/guide/webapps/migrating.html>。另外，还应保证在新旧系统上的充分测试，以及关注那些会影响网页内容显示的变化。

28.7 挑战练习：使用后退键浏览历史网页

注意到了没有，启动了PhotoPageActivity之后，还可以在WebView中点击跳转到其他链接。然而，不管如何跳转，访问了多少个网页，只要按后退键，就会立即回到PhotoPageActivity。如果想使用后退键在WebView里层层退回到已浏览的历史网页呢？

覆盖后退键方法Activity.onBackPressed()就能实现这个行为。在该方法内，再搭配使用WebView的历史记录浏览方法（WebView.canGoBack()和WebView.goBack()）实现我们的浏览逻辑。如果WebView里有历史浏览记录，就回到前一个历史网页，否则调用super.onBackPressed()方法回到PhotoPageActivity。

28.8 挑战练习：非 HTTP 链接支持

在探索PhotoPageFragment的WebView时，可能会遇到非HTTP链接。例如，本书写作时，Flickr图片明细页会显示一个Open in App按钮。如果点击它，应该会启动已安装的Flickr应用；没安装的话，会启动Google Play应用商店让用户选择安装它。

然而，实际点击Open in App按钮，WebView却显示了如图28-4所示的一段错误文字。

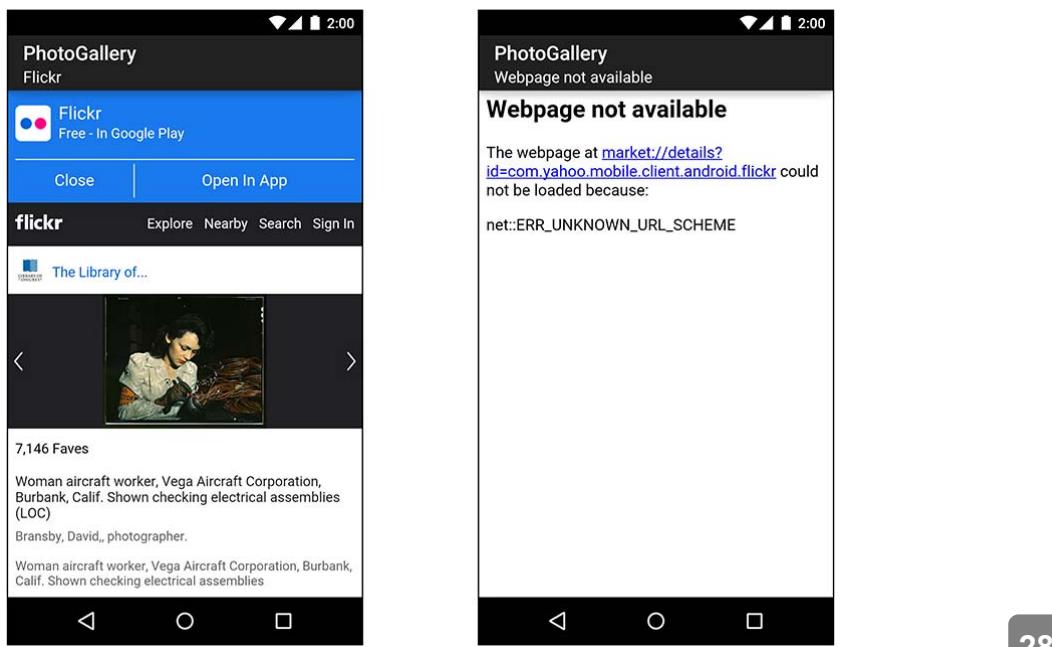


图28-4 Open in App错误

这是因为我们覆盖的WebViewClient.shouldOverrideUrlLoading(...)方法返回的是false值。这样，WebView总是会尝试自己加载URI，即使是它不支持的URI scheme。

要解决这个问题，非HTTP URI就要交给最合适的应用去处理。因此，加载URI前，先检查它的scheme，如果不是HTTP或HTTPS，就发送一个针对目标URI的Intent.ACTION_VIEW。

29

本章，我们通过开发一个名为BoxDrawingView的定制View子类，来学习如何处理触摸事件。在新项目DragAndDraw中，定制View会响应用户的触摸与拖动，在屏幕上绘制出矩形框，如图29-1所示。

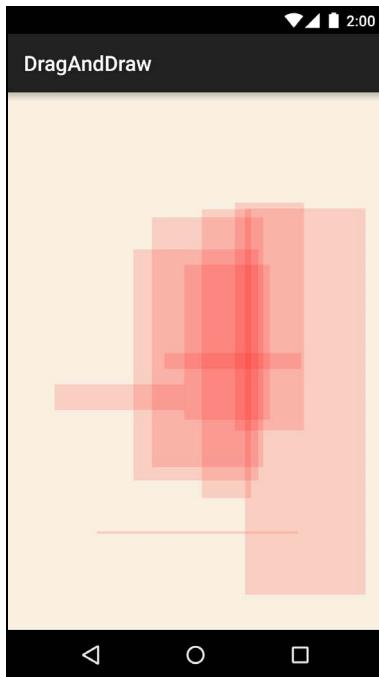


图29-1 各种形状大小的绘制框

29.1 创建 DragAndDraw 项目

创建DragAndDraw新项目。最低SDK版本选择API 16，新建空activity并命名为DragAndDrawActivity。

29.1.1 创建 DragAndDrawActivity

既然 SingleFragmentActivity 可实例化仅包含单个 fragment 的布局，我们让 DragAndDrawActivity 继承它。在 Android Studio 中，复制之前项目的 SingleFragmentActivity.java 和 activity_fragment.xml 文件到 DragAndDraw 项目的对应目录中。

在 DragAndDrawActivity.java 中，修改代码继承 SingleFragmentActivity 类，并创建返回一个 DragAndDrawFragment 对象（稍后会创建该类），如代码清单 29-1 所示。

代码清单 29-1 修改 activity (DragAndDrawActivity.java)

```
public class DragAndDrawActivity extends AppCompatActivity SingleFragmentActivity {

    @Override
    public Fragment createFragment() {
        return DragAndDrawFragment.newInstance();
    }

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

29.1.2 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 的布局最终由 BoxDrawingView 定制视图组成。稍后我们会创建这个定制视图。它会处理所有的图形绘制和触摸事件。

以 android.support.v4.app.Fragment 为超类，新建 DragAndDrawFragment 类。然后覆盖 onCreateView(...) 方法，并在其中实例化 fragment_drag_and_draw.xml 布局，如代码清单 29-2 所示。

代码清单 29-2 创建 DragAndDrawFragment (DragAndDrawFragment.java)

```
public class DragAndDrawFragment extends Fragment {

    public static DragAndDrawFragment newInstance() {
        return new DragAndDrawFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_drag_and_draw, container, false);
        return v;
    }
}
```

运行DragAndDraw应用，确认应用已正确创建，如图29-2所示。

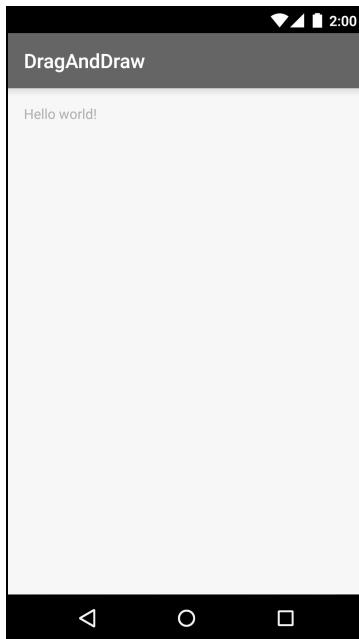


图29-2 带默认布局的DragAndDraw应用

29.2 创建定制视图

Android自带众多优秀标准视图与组件，但有时为追求独特的应用视觉效果，我们仍需创建定制视图。

尽管定制视图种类繁多，但无外乎分为以下两大类别。

- 简单视图。简单视图内部也可以很复杂；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。图形绘制任务都委托给了各个子视图。

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`是个空白画布，因此它作为超类最常见。对于聚合定制视图，我们应选择合适的超类布局，比如`FrameLayout`。
- 继承选定的超类，并至少覆盖一个超类构造方法。
- 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29-3所示。

代码清单29-3 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29-4所示。

代码清单29-4 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29-3所示。

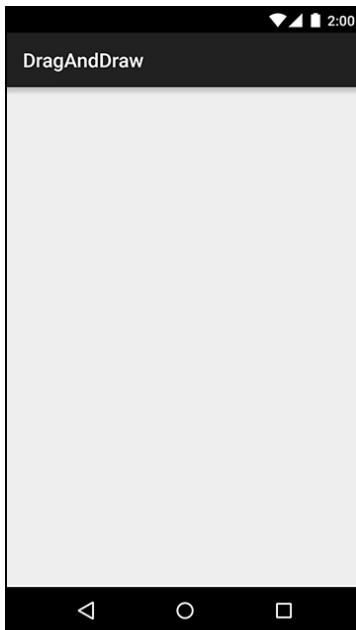


图29-3 未绘制的BoxDrawingView

接下来，让**BoxDrawingView**监听触摸事件，并实现在屏幕上绘制矩形框。

29.3 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。

不过，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)
```

该方法接收一个MotionEvent类实例, MotionEvent类可用来描述包括位置和动作的触摸事件。动作用于描述事件所处的阶段。

动作常量	动作描述
ACTION_DOWN	手指触摸到屏幕
ACTION_MOVE	手指在屏幕上移动
ACTION_UP	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, 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29-5所示。

代码清单29-5 实现BoxDrawingView视图类 (BoxDrawingView.java)

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";

    ...

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                break;
        }

        Log.i(TAG, action + " at x=" + current.x +
              ", y=" + current.y);

        return true;
    }
}
```

注意, X和Y坐标已经封装到PointF对象中。稍后, 我们需要同时传递这两个坐标值。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

除了记录坐标，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，要知道定义矩形框的两个坐标点：

- 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29-6所示。

代码清单29-6 添加Box类（Box.java）

```
public class Box {  
    private PointF mOrigin;  
    private PointF mCurrent;  
  
    public Box(PointF origin) {  
        mOrigin = origin;  
        mCurrent = origin;  
    }  
  
    public PointF getCurrent() {  
        return mCurrent;  
    }  
  
    public void setCurrent(PointF current) {  
        mCurrent = current;  
    }  
  
    public PointF getOrigin() {  
        return mOrigin;  
    }  
}
```

用户触摸BoxDrawingView视图界面时，新的Box对象会创建并添加到现有矩形框数组中，如图29-4所示。

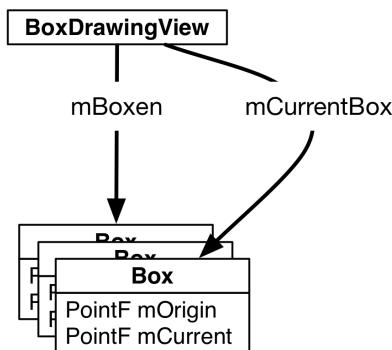


图29-4 DragAndDraw应用中的对象

回到BoxDrawingView类中，添加代码清单29-7所示代码，使用新的Box对象跟踪绘制状态。

代码清单29-7 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 List<Box> mBoxen = new ArrayList<>();

    ...

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        PointF current = new PointF(event.getX(), event.getY());
        String action = "";

        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                action = "ACTION_DOWN";
                // Reset drawing state
                mCurrentBox = new Box(current);
                mBoxen.add(mCurrentBox);
                break;
            case MotionEvent.ACTION_MOVE:
                action = "ACTION_MOVE";
                if (mCurrentBox != null) {
                    mCurrentBox.setCurrent(current);
                    invalidate();
                }
                break;
            case MotionEvent.ACTION_UP:
                action = "ACTION_UP";
                mCurrentBox = null;
                break;
            case MotionEvent.ACTION_CANCEL:
                action = "ACTION_CANCEL";
                mCurrentBox = null;
        }

        return true;
    }
}
  
```

```

        break;
    }

    Log.i(TAG, action + " at x=" + current.x +
        ", y=" + current.y);

    return true;
}
}

```

只要接收到ACTION_DOWN动作事件，就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会在屏幕上绘制数组中的全部Box。）

用户手指在屏幕上移动时，mCurrentBox mCurrent会得到更新。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。这同时也引出了接下来的任务：在屏幕上绘制矩形框。

29.4 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。这会引起自上而下的链式调用反应。首先，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视图也就生效了。

为加入这种绘制，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类拥有我们需要的所有绘制操作。其方法可决定绘在哪里以及绘什么，比如线条、圆形、字词、矩形等。
- Paint类决定如何绘制。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29-8所示。

代码清单29-8 创建Paint（BoxDrawingView.java）

```
public class BoxDrawingView extends View {
    private static final String TAG = "BoxDrawingView";
```

```

private Box mCurrentBox;
private List<Box> mBoxen = new ArrayList<>();
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，现在能在屏幕上绘制矩形框了，如代码清单29-9所示。

代码清单29-9 覆盖onDraw(Canvas)方法 (BoxDrawingView.java)

```

public BoxDrawingView(Context context, AttributeSet attrs) {
    ...
}

@Override
protected void onDraw(Canvas canvas) {
    // Fill the background
    canvas.drawPaint(mBackgroundPaint);

    for (Box box : mBoxen) {
        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);
    }
}

```

29

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，据其两点坐标，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，在屏幕上绘制红色矩形框。

运行DragAndDraw应用，尝试绘制一些红色矩形框，如图29-5所示。

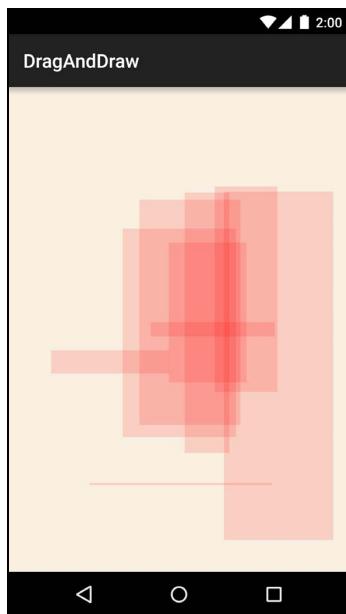


图29-5 程序员式的情绪表达

这样就完成了。我们创建了一个捕捉其触摸事件并执行绘制的视图，

29.5 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。首先，View视图有ID时，才可以调用它们。其次，相较于Bundle参数，这些方法返回并处理的是实现Parcelable接口的对象。推荐使用Bundle，这样我们就不用自己实现Parcelable接口了。（Parcelable接口的实现很复杂，如有可能，应尽量避免。）

最后，我们还需要保存BoxDrawingView的View父视图的状态。在Bundle中保存super.onSaveInstanceState()方法结果，然后调用super.onRestoreInstanceState(Parcelable)方法把结果发送给超类。

29.6 挑战练习：旋转矩形框

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，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。

只要代码没有致命性的错误，开发一个能跑起来的应用还是很简单的。然而，应用可用只是个开始，人们想要的是好用、爱用的应用。如果手机或平板设备上的应用能模拟物理世界，让人有身临其境之感，用户体验该有多棒啊！

现实世界灵动多变。应用用户界面也可以动起来，从技术上讲，就是让用户界面元素从一个位置移动到另一个位置。

本章，我们来开发一个落日景象的模拟应用。按住屏幕，太阳会慢慢落下海平面，天空的颜色随之不断变换，犹如真的日落。

30.1 建立场景

首先是创建动画场景。创建一个名为Sunset的新项目，确保`minSdkVersion`设置为API 16，并且使用空activity模板。将主activity命名为SunsetActivity。然后把之前项目的`SingleFragmentActivity.java`和`activity_fragment.xml`文件复制到当前项目备用。

海边落日光影变换，色彩繁多。因此，我们需要准备一些色彩资源。在`res/values`目录中新建`colors.xml`文件。该资源文件内容定义如代码清单30-1所示。

代码清单30-1 添加落日色彩（`res/values/colors.xml`）

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="bright_sun">#fcfcfb7</color>
    <color name="blue_sky">#1e7ac7</color>
    <color name="sunset_sky">#ec8100</color>
    <color name="night_sky">#05192e</color>
    <color name="sea">#224869</color>
</resources>
```

虽然矩形视图模拟天空和大海的效果还可以，但没人见过方方长长的太阳吧？技术实现简单可不是理由。所以，在`res/drawable`目录新建一个`sun.xml`椭圆形drawable资源，如代码清单30-2所示。

代码清单30-2 添加模拟太阳的XML drawable (res/drawable/sun.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid android:color="@color/bright_sun" />
</shape>
```

在矩形视图上显示这个椭圆形drawable，就会看到一个圆。现在，相信大家一定会点头赞许，仿佛看到真正的太阳挂在天空。

接下来，我们使用一个完整的布局文件构建整个场景。该布局会在稍后创建的SunsetFragment中使用，因此可直接命名为fragment_sunset.xml。布局定义如代码清单30-3所示。

代码清单30-3 创建落日场景布局 (res/layout/fragment_sunset.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <FrameLayout
        android:id="@+id/sky"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.61"
        android:background="@color/blue_sky">
        <ImageView
            android:id="@+id/sun"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_gravity="center"
            android:src="@drawable/sun"
        />
    </FrameLayout>
    <View
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="0.39"
        android:background="@color/sea"
    />
</LinearLayout>
```

现在可以预览布局了。怎么样，大海蔚蓝，天蓝日落，多么动人的画面啊！这不禁让人念起那漫步海滩、坐船出海的过往旅程。

布局搞定了，下面要编写代码让应用跑起来。创建一个SunsetFragment类，并在其中新增一个newInstance(...)方法，如代码清单30-4所示。然后，在onCreateView(...)方法中，实例化fragment_sunset布局并返回结果视图。

代码清单30-4 创建SunsetFragment类 (SunsetFragment.java)

```
public class SunsetFragment extends Fragment {  
  
    public static SunsetFragment newInstance() {  
        return new SunsetFragment();  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
        Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.fragment_sunset, container, false);  
  
        return view;  
    }  
}
```

为显示SunsetFragment，让SunsetActivity类继承SingleFragmentActivity类，如代码清单30-5所示。

代码清单30-5 创建SunsetFragment类 (SunsetActivity.java)

```
public class SunsetActivity extends SingleFragmentActivity {  
  
    @Override  
    protected Fragment createFragment() {  
        return SunsetFragment.newInstance();  
    }  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        ...  
    }  
}
```

运行Sunset应用。一切正常的话，可看到如图30-1所示的画面。

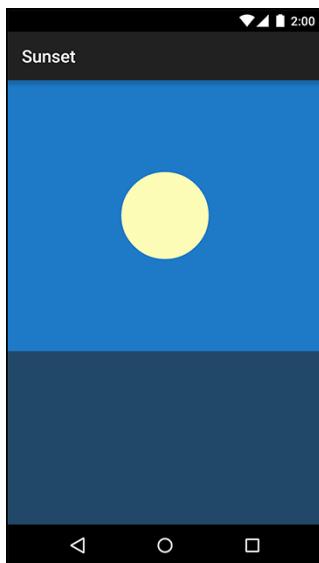


图30-1 落日徐徐

30.2 简单属性动画

创建完落日场景，现在要实现太阳慢慢落下海平面的动画效果。

首先，我们需要在fragment中获取一些必要的信息。在`onCreateView(...)`方法中，获取我们要控制的视图并存入相应变量中备用，如代码清单30-6所示。

代码清单30-6 获取视图引用（SunsetFragment.java）

```
public class SunsetFragment extends Fragment {  
  
    private View mSceneView;  
    private View mSunView;  
    private View mSkyView;  
  
    public static SunsetFragment newInstance() {  
        return new SunsetFragment();  
    }  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                            Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.fragment_sunset, container, false);  
  
        mSceneView = view;  
        mSunView = view.findViewById(R.id.sun);  
        mSkyView = view.findViewById(R.id.sky);  
    }  
}
```

```

        return view;
    }
}

```

做完了准备工作，接下来就是编码实现了。从技术上讲，所谓太阳落下海平面，实际就是平滑地移动mSunView视图，直到它的顶部刚好与海平面的顶部边缘重合。这可以通过调整mSunView视图顶部的坐标位置来实现。

显然，我们先要清楚动画的开始和结束点。这个任务就交给startAnimation()方法处理，如代码清单30-7所示。

代码清单30-7 获取视图的顶部坐标位置（SunsetFragment.java）

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    ...
}

private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();
}

```

和View视图类的getBottom()、getRight()和getLeft()方法一样，getTop()可以返回自己的local layout rect。视图的local layout rect是其相对父视图的位置和其尺寸大小的描述。视图一旦实例化，这些值都是相对固定的。虽然可以修改这些值，从而改变视图的位置，但不推荐大家这么做。要知道，每次布局切换时，这些值都会被重置，所以才会有相对固定一说。

无论怎么变化，动画的开始点都是mSunView视图的顶部位置。结束点是其父视图mSkyView的底部位置。移动距离是调用getHeight()方法返回的mSkyView高度。除了getHeight()方法，使用getBottom()和getTop()方法并取差值也能获得相同结果。

知道了动画的开始和结束点，我们创建一个ObjectAnimator对象执行动画，如代码清单30-8所示。

代码清单30-8 创建模拟太阳的animator对象（SunsetFragment.java）

```

private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
        .ofFloat(mSunView, "y", sunYStart, sunYEnd)
        .setDuration(3000);

    heightAnimator.start();
}

```

在onCreateView()方法中，为mSceneView视图设置监听器。只要用户点击它，就调用startAnimation()方法执行动画，如代码清单30-9所示。

代码清单30-9 响应触摸，执行动画（SunsetFragment.java）

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_sunset, container, false);

    mSceneView = view;
    mSunView = view.findViewById(R.id.sun);
    mSkyView = view.findViewById(R.id.sky);

    mSceneView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startAnimation();
        }
    });
}

return view;
}
```

运行Sunset应用。点击应用界面，欣赏一段美丽的落日动画，如图30-2所示。

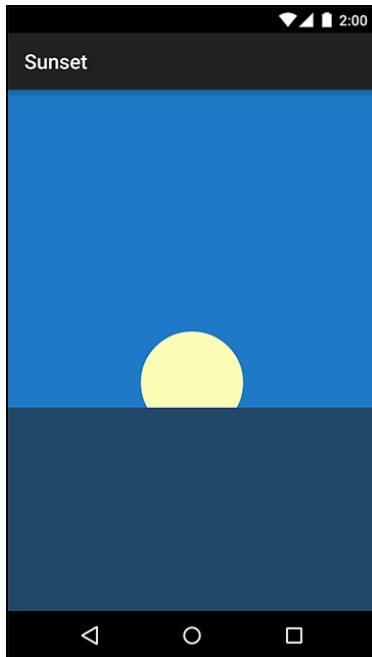


图30-2 落日

最后，我们来看看这段动画的实现原理：`ObjectAnimator`是个属性动画制作对象。要获得某种动画效果，传统方式是设法在屏幕上移动视图，而属性动画制作对象却另辟蹊径：以一组不同的参数值反复调用属性设置方法。

调用以下方法可以创建ObjectAnimator对象：

```
ObjectAnimator.ofFloat(mSunView, "y", 0, 1)
```

上例中，新建ObjectAnimator一旦启动，就会以从0开始递增的参数值反复调用mSunView.setY(float)方法：

```
mSunView.setY(0);
mSunView.setY(0.02);
mSunView.setY(0.04);
mSunView.setY(0.06);
mSunView.setY(0.08);
...
...
```

直到调用mSunView.setY(1)为止。这个0~1区间参数值的确定过程又称为interpolation。可以想象到，在这个interpolation过程中，即便很短暂，确定相邻参数值也是要耗费时间的；由于人眼的视觉暂留现象，动画效果就形成了。

30.2.1 视图属性转换

想让视图动起来的话，仅仅靠属性动画制作对象是不切实际的，尽管它确实很有用。因此，它找到了属性转换（transformation properties）这个合作伙伴。

前面说过，视图都有local layout rect（视图实例化时被赋予的位置及大小尺寸参数值）。知道了视图属性值（local layout rect），就可以改变这些属性值，从而实现四处移动视图。这种做法就叫作属性转换。例如，利用rotation、pivotX和pivotY这三个参数可以旋转视图；利用scaleX和scaleY可以缩放视图；而利用translationX和translationY可以四处移动视图，如图30-3、图30-4和图30-5所示。

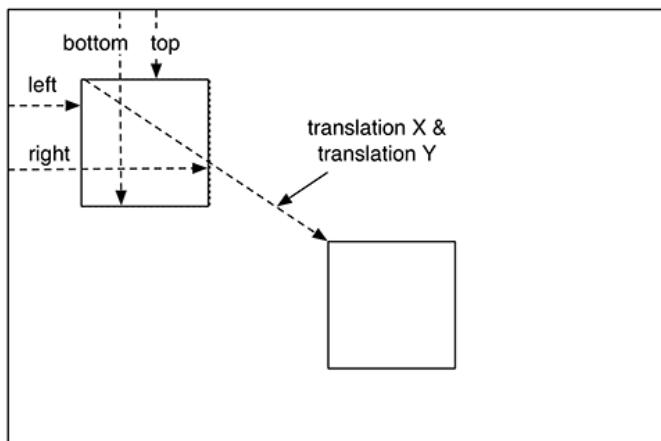


图30-3 视图移动

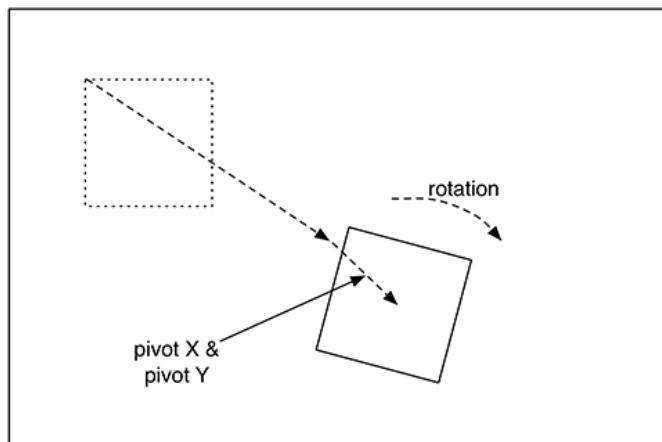


图30-4 视图旋转

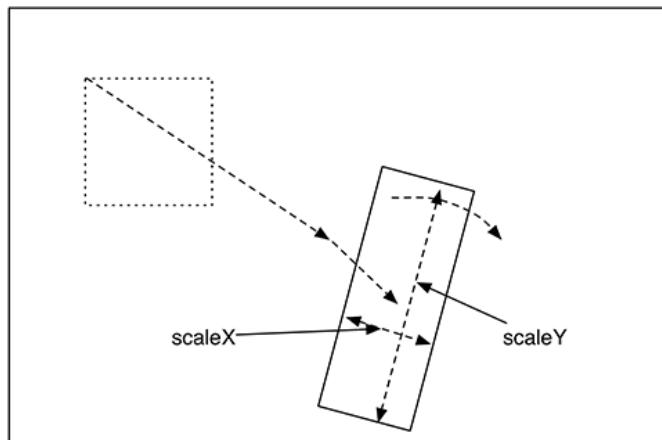


图30-5 视图缩放

视图的所有这些属性值都有获取方法和设置方法。例如，调用`getTranslationX()`方法就能得到`translationX`值；调用`setTranslationX(float)`方法就能设置`translationX`值。

那么`y`属性有什么作用呢？实际上，`x`和`y`属性是以布局坐标为参考值设立的一种便利开发的属性值。例如，简单写几行代码，就可以把视图置于某个`X`和`Y`坐标确定的位置。分析其背后原理可知，这就是通过修改`translationX`和`translationY`属性值来实现的。所以，调用`mSunView.setY(50)`方法就等同于：

```
mSunView.setTranslationY(50 - mSunView.getTop())
```

30.2.2 使用不同的 interpolator

目前，Sunset应用的动画效果还不够完美。假设太阳一开始静止于天空，在进入落下的动画时，应该有个加速过程。这也好办，使用TimeInterpolator就可以了。这个TimeInterpolator的作用就是：改变A点到B点的动画效果。

在startAnimation()方法中，添加代码清单30-10所示代码，使用一个AccelerateInterpolator对象实现太阳加速落下的特效。

代码清单30-10 添加加速特效（SunsetFragment.java）

```
private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
        .ofFloat(mSunView, "y", sunYStart, sunYEnd)
        .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    heightAnimator.start();
}
```

重新运行Sunset应用。点击屏幕观察动画效果。这次，太阳先是慢慢落下，然后朝着海平面方向加速坠落。

使用不同的TimeInterpolator对象可实现不同的动画特效。想要了解Android自带的TimeInterpolator还有哪些，请参阅TimeInterpolator参考文档的其他间接子类介绍。

30.2.3 色彩渐变

优化完落日的动画效果，接着处理天空随日落所呈现的色彩变换效果。在onCreateView(...)方法中，获取colors.xml文件定义的色彩资源并存入相应的实例变量，如代码清单30-11所示。

代码清单30-11 取出日落色彩资源（SunsetFragment.java）

```
public class SunsetFragment extends Fragment {

    ...
    private View mSkyView;

    private int mBlueSkyColor;
    private int mSunsetSkyColor;
    private int mNightSkyColor;

    ...
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
        mSkyView = view.findViewById(R.id.sky);
    }
}
```

```

Resources resources = getResources();
mBlueSkyColor = resources.getColor(R.color.blue_sky);
mSunsetSkyColor = resources.getColor(R.color.sunset_sky);
mNightSkyColor = resources.getColor(R.color.night_sky);

mSceneView.setOnClickListener(new View.OnClickListener() {
    ...
});

return view;
}

```

在startAnimation()方法中，再参照代码清单30-12添加一个ObjectAnimator，实现天空色彩从mBlueSkyColor到mSunsetSkyColor变换的动画效果。

代码清单30-12 实现天空的色彩变换（SunsetFragment.java）

```

private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

    ObjectAnimator heightAnimator = ObjectAnimator
        .ofFloat(mSunView, "y", sunYStart, sunYEnd)
        .setDuration(3000);
    heightAnimator.setInterpolator(new AccelerateInterpolator());

    ObjectAnimator sunsetSkyAnimator = ObjectAnimator
        .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
        .setDuration(3000);

    heightAnimator.start();
    sunsetSkyAnimator.start();
}

```

天空的动画效果似乎就这么完成了。运行SunSet应用看看吧。怎么，似乎不大对劲啊！从蓝色到橘黄色，天空的色彩变化太夸张了，一点都不自然。

仔细分析就知道，颜色int数值并不是个简单的数字。它实际是由四个较小数字转换而来。因此，只有知道颜色的组成奥秘，ObjectAnimator对象才能合理地确定蓝色和橘黄色之间的中间值。

不过，知道如何确定颜色中间值还不够，我们还需要一个TypeEvaluator子类的协助。TypeEvaluator能帮助ObjectAnimator对象精确地计算开始到结束间的递增值。Android提供的这个TypeEvaluator子类叫作ArgbEvaluator，如代码清单30-13所示。

代码清单30-13 使用ArgbEvaluator（SunsetFragment.java）

```

private void startAnimation() {
    float sunYStart = mSunView.getTop();
    float sunYEnd = mSkyView.getHeight();

```

```
ObjectAnimator heightAnimator = ObjectAnimator
    .ofFloat(mSunView, "y", sunYStart, sunYEnd)
    .setDuration(3000);
heightAnimator.setInterpolator(new AccelerateInterpolator());

ObjectAnimator sunsetSkyAnimator = ObjectAnimator
    .ofInt(mSkyView, "backgroundColor", mBlueSkyColor, mSunsetSkyColor)
    .setDuration(3000);
sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

heightAnimator.start();
sunsetSkyAnimator.start();
}
```

再次运行Sunset应用。夕阳西下，从蓝色到橘黄色，色彩的过渡终于自然了。

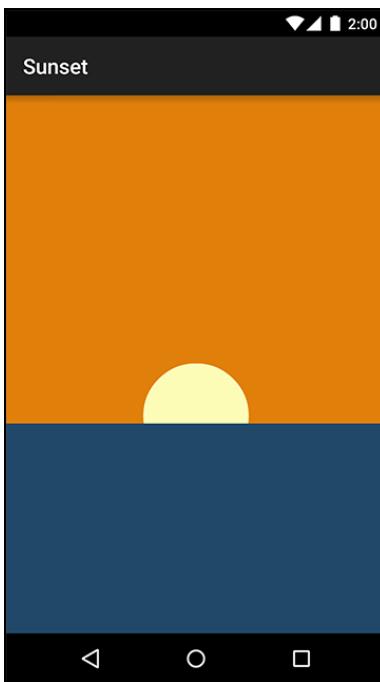


图30-6 天空的色彩随日落变换

30.3 播放多个动画

有时，我们需要同时执行一些动画。这很简单，同时调用**start()**方法就行了。

但是，假如要像编排舞步那样编排多个动画的执行，事情就没那么简单了。例如，为实现完整的日落景象，太阳落下去之后，天空应该从橘黄色再转为午夜蓝。

办法总是有的，我们可以使用**AnimatorListener**。**AnimatorListener**会让你知道动画什

么时候结束。这样，执行完第一个动画，我们就可以接力执行第二个夜空变化的动画。然而，理论分析很简单，实际去做的话，少不了要准备多个监听器，这也很麻烦。好在Android还设计了方便又简单的**AnimatorSet**。下面就来学习使用它。

首先，如代码清单30-14所示，删除掉原来的动画启动代码，并添加夜空变化的动画代码。

代码清单30-14 创建夜空动画（SunsetFragment.java）

```
private void startAnimation() {
    ...
    sunsetSkyAnimator.setEvaluator(new ArgbEvaluator());

    ObjectAnimator nightSkyAnimator = ObjectAnimator
        .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
        .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    heightAnimator.start();
    sunsetSkyAnimator.start();

}
```

然后，创建并执行一个动画集，如代码清单30-15所示。

代码清单30-15 创建动画集（SunsetFragment.java）

```
private void startAnimation() {
    ...

    ObjectAnimator nightSkyAnimator = ObjectAnimator
        .ofInt(mSkyView, "backgroundColor", mSunsetSkyColor, mNightSkyColor)
        .setDuration(1500);
    nightSkyAnimator.setEvaluator(new ArgbEvaluator());

    AnimatorSet animatorSet = new AnimatorSet();
    animatorSet
        .play(heightAnimator)
        .with(sunsetSkyAnimator)
        .before(nightSkyAnimator);
    animatorSet.start();
}
```

说白了，**AnimatorSet**就是可以放在一起执行的动画集。可以用好几种方式创建动画集，但使用上述代码中的**play(Animator)**方法最容易。

调用**play(Animator)**方法之前，要先创建一个**AnimatorSet.Builder**对象，然后利用它创建链式方法调用。传入**play(Animator)**方法的**Animator**是链首。所以，以上代码中的链式调用就可以这样解读：协同执行**heightAnimator**和**sunsetSkyAnimator**动画；在**nightSkyAnimator**之前执行**heightAnimator**动画。在实际开发中，可能会用到更复杂的动画集。这也没问题，需要的话，可以多次调用**play(Animator)**方法。

再次运行Sunset应用。用心感受下这幅动人祥和的画面，真是太棒了！

30.4 深入学习：其他动画API

除了广受欢迎的属性动画，Android动画工具箱里还有一些其他动画工具。不管用不用，花点时间了解一下总没错。

30.4.1 传统动画工具

Android有个叫作`android.view.animation`的动画工具类包。Honeycomb发布时，又引入了一个更新的`android.animation`包。这是两个不同的包，请注意区分。

它们都是传统的动画工具包，简单了解就可以了。注意到了吗？本章使用的动画工具类的类名都为`animator`。如果遇到`animation`这样的类名，就能断定它来自传统动画工具包，不看也罢！

30.4.2 转场

Android 4.4引入了新的视图转场框架。从一个activity小视图动态弹出另一个放大版activity视图就可以使用转场框架实现。

实际上，转场框架的工作原理很简单：我们定义一些场景，它们代表各个时点的视图状态，然后按照一定的逻辑切换场景。场景在XML布局文件中定义，转场在XML动画文件中定义。

在本章日落例子中，activity已经运行了，这种情况就不太适合使用转场框架，所以我们用到了强大的属性动画框架。再以CriminalIntent应用中处理crime图片为例，如果想实现以弹窗展示放大版图片这样的动画效果，首先要知道照片放在哪里，其次是如何在对话框里布置新图片。显然，对于这类布局动态转场任务，转场框架比`ObjectAnimator`更能胜任。

30.5 挑战练习

首先，让日落可逆。也就是说，点击屏幕，等太阳落下后，再次点击屏幕，让太阳升起来。动画集不能逆向执行，因此，你需要新建一个`AnimatorSet`。

第二个挑战是添加太阳动画特效，让它有规律地放大缩小或是加一圈旋转的光线。（这实际是反复执行一段动画特效，可考虑使用`ObjectAnimator`的`setRepeatCount(int)`方法。）

另外，海面上要是有太阳的倒影就更真实了。

最后，再来看个颇具挑战的练习。在日落过程中实现动画反转。在太阳慢慢下落时点击屏幕，让太阳无缝回升至原来所在的位置。或者，在太阳落下进入夜晚时点击屏幕，让太阳重新升回天空，如日出那样。

31 地理位置和Play服务

本章，我们来编写一个叫作Locatr的应用。Locatr支持基于地理位置的Flickr图片搜索。它首先会定位用户的当前位置，然后搜索附近的图片（如图31-1所示）。下一章，应用升级后，图片还会显示在地图上。

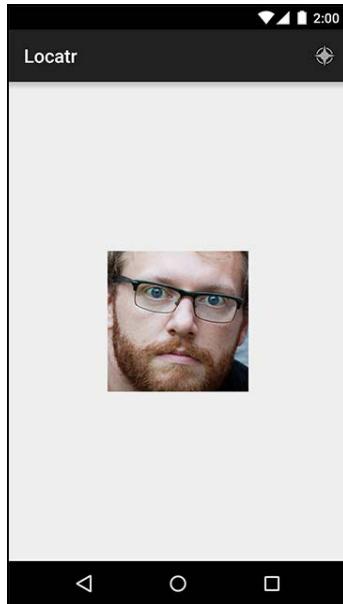


图31-1 Locatr应用效果图

31

寻找用户的当前位置简单又有趣。只要把标准库之外一个叫作Google Play服务的库整合进应用就可以了。

31.1 地理位置和定位类库

我们一起来看看普通Android设备能看到什么。另外，再看看Android提供了哪些工具，使用这些工具又能看到什么。

Android提供了一些基本地理位置API，开箱即用。使用这些API，应用可以从各种信息源接收地理位置数据。对大多数手机设备来说，这些信息源是指来自GPS定位仪的比较精准的地理位置数据以及来自手机基站或Wi-Fi连接的不太精准的地理位置数据。Android推出市场时就已经内置了这些API，可以在`android.location`库包中找到。

虽然它们早已存在，但使用起来仍不够完美。真实世界里的应用往往有这样的要求，“请尽可能给我最精确的定位数据，费不费电没关系”，或者“我需要定位数据，最好是不要太费电”；而像“请启动GPS定位仪，然后告诉我它显示的定位数据”这样的比较具体的要求却很少见。

设备一旦四处移动起来，问题就来了。如果在户外，GPS最合适。如果GPS没信号，手机基站定位也不错。如果两者都不行，凑合使用加速感应器和陀螺仪也总比完全没有方向要强。

过去，为获得定位数据，高质量应用必须求助于上述各种数据源，并且要根据不同场景随时切换。

Google Play服务

考虑到上述情况，就迫切需要更好的定位API。然而，如果要把这样的API加入标准库，开发人员不知道几年之后才能用得到。现实是，操作系统拥有定位所需的一切硬件：GPS、手机基站定位等。

还好，标准库并不是Google发布API的唯一途径。除了标准库，Google还提供了Play服务。这是一套随Google Play商店应用安装的常用服务。为解决定位疑难问题，Google在Play服务中提供了叫作Fused Location Provider的全新定位服务。

既然这些库包含在其他应用里，那么首先要安装这些应用。同时，这也意味着只有安装了Play商店应用且保持更新的设备才能用上你的应用。而且，你的应用也要通过Play商店发布。如果做不到，不好意思，请使用其他定位API吧。

为了进行练习，如果使用物理设备，请确保Play商店应用已更新到最新。要是使用模拟器呢？别担心，稍后就会提供解决办法。

31.2 创建Locatr项目

理论知识已足够，可以创建应用了。在Android Studio中，创建一个叫作Locatr的新项目。创建一个空activity并将其命名为`LocatrActivity`，最低SDK版本为API 16。然后将`SingleFragmentActivity.java`和`activity_fragment.xml`文件分别复制到Locatr项目的对应目录。

Locatr应用也需要搜索Flickr，所以为了方便，它需要PhotoGallery应用中的一些查询代码。从PhotoGallery的随书项目文件中找到`FlickrFetchr.java`和`GalleryItem.java`（建议使用第25章的文件），将它们复制到Locatr项目的对应目录。

稍后，我们会着手为Locatr应用创建用户界面。如果你使用的是模拟器，请继续阅读下一节配置测试环境。如果不是，请直接跳到31.4节。

31.3 Play 服务和模拟器

如果使用AVD模拟器，请确保模拟器镜像已更新到最新版本。

为更新镜像文件，打开SDK管理器（Tools→Android→SDK Manager）。找到你要使用的用于模拟器的Androd版本，并确认Google APIs System Images已安装并更新至最新。如果提示有更新，请按照提示完成更新（如图31-2所示）。

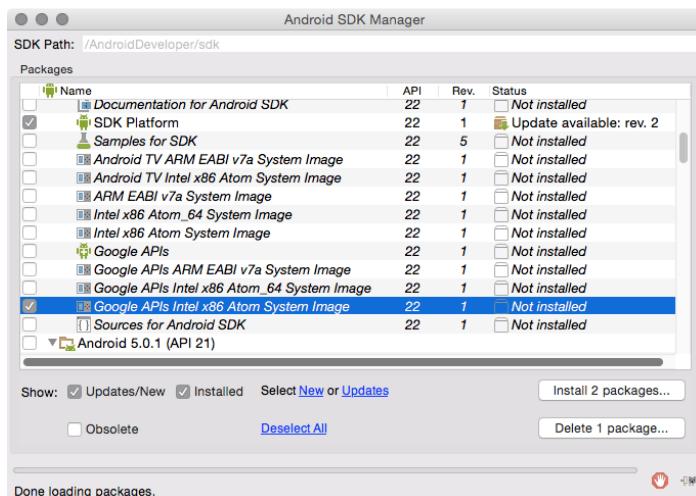


图31-2 确保模拟器已更新

AVD模拟器也应有个支持指定版本Google API的目标操作系统。创建模拟器时，选择目标操作系統版本时会看到API级别。对于Locatr项目，选择API 21级或更高即可（如图31-3所示）。

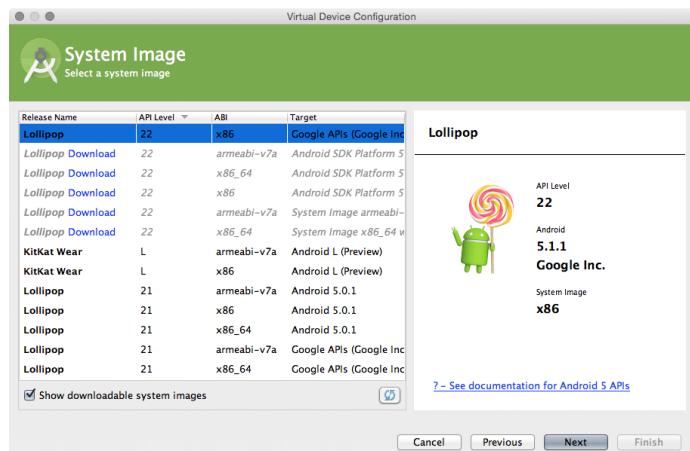


图31-3 选择API级别

如果先前已搭建了合适的模拟器，同样需要更新至最新版本。完成更新后，记得重启模拟器让其生效。

在本章和随一章，我们都会使用模拟器运行Locatr应用。推荐使用Android Studio内置的AVD模拟器，而非Genymotion模拟器。虽然两者都能用，但配置Genymotion模拟器用于Locatr应用稍显复杂，而且超出了本书讨论的范畴。如果确实想用Genymotion模拟器，请访问Genymotion网站寻求指导。

模拟定位数据

在模拟器上，我们也需要不断更新的地理位置信息用于测试。Android Studio提供有模拟器控制面板，可以发送地理位置坐标给模拟器；但它只适用于以前的定位服务，不能用于新的Fused Location Provider。看来，我们只能以代码的方式发布地理位置信息了。

在Big Nerd Ranch培训基地，只要你感兴趣，任何主题我们都愿意细细讲给你听。不过，在遭遇了狩猎教学大失败之后，我们终于明白不能这样教了。现在，我们更实际，更乐意传授一些学生最急需、更有用的知识。所以，我们放弃了教授如何编写模拟数据发送代码，而是为你写了一个叫作MockWalker的独立应用。要使用它，请从以下地址下载并安装其APK：

<https://www.bignerdranch.com/solutions/MockWalker.apk>

最容易安装的方式是打开模拟器上的浏览器应用，输入地址，如图31-4所示。

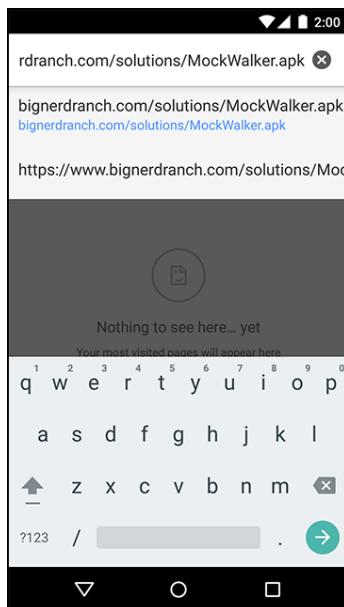


图31-4 输入下载地址

下载完成后，点击工具栏的下载通知项打开这个APK，如图31-5所示。

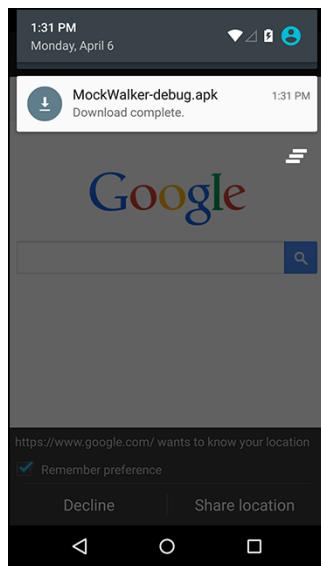


图31-5 打开下载文件

MockWalker会触发一个模拟步行行为，通过服务把模拟地理位置数据发送给Fused Location Provider。应用运行后，它会假装在亚特兰大的柯克伍德附近转悠。

服务运行时，只要Locatr应用向Fused Location Provider请求定位数据，它就会收到MockWalker发布的地理位置数据（如图31-6所示）。

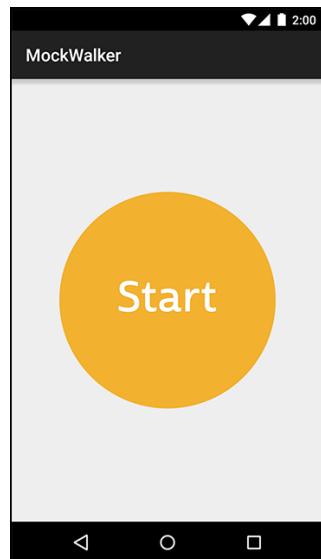


图31-6 运行MockWalker

运行MockWalker应用并点击Start按钮。退出应用后，服务会一直在后台运行。（千万不要退出模拟器。）使用完毕或不再需要的话，就重新打开MockWalker应用，点击Stop按钮关闭服务。

如果想了解MockWalker应用的工作原理，可以查看它的源代码（见本章随书代码文件）。为管理持续的地理位置更新，该应用使用了RxJava和sticky前台服务技术。

31.4 创建Locatr应用

接下来，为Locatr应用创建用户界面。首先，为搜索按钮添加字符串资源，如代码清单31-1所示。

代码清单31-1 添加搜索按钮文字（res/values/strings.xml）

```
<resources>
    <string name="app_name">Locatr</string>

    <string name="search">Find an image near you</string>
</resources>
```

老规矩，我们需要一个fragment，所以重命名activity_locatr.xml布局文件为fragment_locatr.xml。修改RelativeLayout控件，使用一个ImageView控件显示搜索到的图片，如图31-7所示。（padding属性值不重要，如果模板自带，可直接删除。）

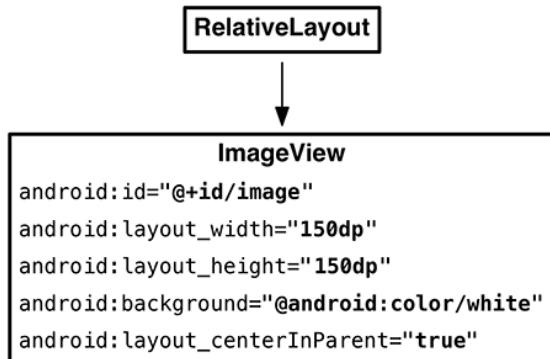


图31-7 Locatr应用的布局（res/layout/fragment_locatr.xml）

Locatr应用还需要一个按钮触发搜索。这里，可以利用工具栏来实现。重命名res/menu/menu_locatr.xml菜单文件为res/menu/fragment_locatr.xml。然后修改按钮以显示一个搜索按钮，如代码清单31-2所示。（没错，菜单文件名和res/layout/fragment_locatr.xml一样。这没有问题，菜单资源和布局资源的命名空间是不同的。）

代码清单31-2 配置菜单（res/menu/fragment_locatr.xml）

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    <item android:id="@+id/action_locate"
        android:icon="@android:drawable/ic_menu_compass"
```

```

        android:title="@string/search"
        android:enabled="false"
        app:showAsAction="ifRoom"/>
    
```

按钮默认是禁用的。后面，连上Play服务之后，我们再启用它。

现在，创建一个名为LocatrFragment的子类，继承Fragment，并在其中实例化布局，显示ImageView视图，如代码清单31-3所示。

代码清单31-3 创建LocatrFragment (LocatrFragment.java)

```

public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_locatr, container, false);

        mImageView = (ImageView) v.findViewById(R.id.image);

        return v;
    }
}

```

接着，完成菜单项的创建，如代码清单31-4所示。

代码清单31-4 添加菜单 (LocatrFragment.java)

```

public class LocatrFragment extends Fragment {
    private ImageView mImageView;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {

```

```

        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.fragment_locatr, menu);
    }
}

```

最后，把LocatrFragment交给LocatrActivity托管，如代码清单31-5所示。

代码清单31-5 托管Locatr fragment (LocatrActivity.java)

```

public class LocatrActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }
}

```

至此，Locatr应用的用户界面全部搞定了。接下来该是配置使用Play服务的时候了。

31.5 配置 Google Play 服务

要使用Fused Location Provider获取地理位置，就需要使用Google Play服务。要让这些服务运行起来，还需要一些基础准备工作。

首先是添加Google Play服务依赖库。服务本身是在Play应用里运行的，但Google Play服务库包含所有和它们交互的代码。

打开app模块设置（File→Project Structure）。在app模块的dependencies选项页，点击+按钮添加依赖库。添加的时候，要完整输入com.google.android.gms:play-services-location:7.3.0依赖库名。这是Play服务的定位类库。

随着代码的迭代，这个库的版本号一直在变。如果想知道最新的版本号，可使用play-services关键字搜索依赖库。如果想使用最新版本，可使用具体的版本号指定要使用的依赖库（play-services-location:x.x.x）。

如果有多个版本，到底该用哪个呢？我们的实践表明，尽量使用最新的。不过，我们无法保证本章代码一定能与未来版本的依赖库兼容，所以，学习本章时，请使用7.3.0版本。

既然依赖的服务在设备上的其他应用中运行，就不敢保证Play服务库一定可用。接下来，需要在代码中检测是否有可用的Play服务。更新主activity执行必要的检查，如代码清单31-6所示。

代码清单31-6 检测Play服务 (LocatrActivity.java)

```

public class LocatrActivity extends SingleFragmentActivity {
    private static final int REQUEST_ERROR = 0;

    @Override
    protected Fragment createFragment() {
        return LocatrFragment.newInstance();
    }

    @Override
    protected void onResume() {

```

```
super.onResume();

int errorCode = GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);

if (errorCode != ConnectionResult.SUCCESS) {
    Dialog errorDialog = GooglePlayServicesUtil
        .getErrorDialog(errorCode, this, REQUEST_ERROR,
            new DialogInterface.OnCancelListener() {

                @Override
                public void onCancel(DialogInterface dialog) {
                    // Leave if services are unavailable.
                    finish();
                }
            });
    errorDialog.show();
}
}
```

通常，我们不会像这样使用Dialog。不过，这里不用担心设备旋转问题。无论设备是否旋转，`errorCode`值都一样，所以Dialog会再次显示。

地理位置定位权限

Locatr应用还需要地理位置定位权限才能工作。定位相关的权限有两个：`android.permission.ACCESS_FINE_LOCATION`和`android.permission.ACCESS_COARSE_LOCATION`。精准定位来自GPS定位仪；非精准定位来自手机基站或Wi-Fi接入点。

Locatr应用需要精准定位数据，所以肯定要添加ACCESS_FINE_LOCATION权限。但最好也一并把ACCESS_COARSE_LOCATION加上，因为万一GPS没信号，至少还可以尝试使用手机基站或Wi-Fi定位嘛。

Locatr应用需要搜索Flickr，因此，添加完定位权限，再顺手把网络使用权限添加上，如代码清单31-7所示。

代码清单31-7 添加权限（AndroidManifest.xml）

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.bignerdranch.android.locatr" >  
  
    <uses-permission  
        android:name="android.permission.ACCESS_FINE_LOCATION" />  
    <uses-permission  
        android:name="android.permission.ACCESS_COARSE_LOCATION" />  
    <uses-permission  
        android:name="android.permission.INTERNET" />  
  
    ...  
  
</manifest>
```

31.6 使用Google Play服务

要使用Play服务，还需要创建一个客户端类。这个客户端是个`GoogleApiClient`类实例。在Play服务参考文档区（<http://developer.android.com/reference/gms-packages.html>），可以找到这个类（和其他所有Play服务类）的文档资料。

为创建客户端，我们创建一个`GoogleApiClient.Builder`。然后，使用我们要使用的API配置它。最后，调用`build()`方法创建实例。

在`onCreate(Bundle)`方法中，创建一个`GoogleApiClient.Builder`实例，如代码清单31-8所示。然后，把定位服务API添加给它。

代码清单31-8 创建GoogleApiClient (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private ImageView mImageView;
    private GoogleApiClient mClient;

    public static LocatrFragment newInstance() {
        return new LocatrFragment();
    }

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

        mClient = new GoogleApiClient.Builder(getActivity())
            .addApi(LocationServices.API)
            .build();
    }
}
```

创建了`mClient`客户端，我们之后需要连接它。Google推荐在`onStart()`方法里连接客户端，在`onStop()`方法里断开连接。调用客户端的`connect()`方法会改变菜单按钮的行为，所以要调用`invalidateOptionsMenu()`方法更新它的状态，如代码清单31-9所示。（稍后还会再次调用它：被告知已连接的时候。）

代码清单31-9 连接和断开连接 (LocatrFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
}

@Override
public void onStart() {
    super.onStart();

    getActivity().invalidateOptionsMenu();
    mClient.connect();
}
```

```

    }

    @Override
    public void onStop() {
        super.onStop();

        mClient.disconnect();
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        ...
    }
}

```

如果连不上客户端，应用就什么也做不了。所以，按钮的启用和禁用状态应作相应切换，如代码清单31-10所示。

代码清单31-10 更新菜单按钮状态 (LocatrFragment.java)

```

@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_locatr, menu);

    MenuItem searchItem = menu.findItem(R.id.action_locate);
    searchItem.setEnabled(mClient.isConnected());
}

```

发现连上客户端后，再添加另一个`getActivity().invalidateOptionsMenu()`调用更新菜单项。连接状态信息要通过两个回调接口传递：`ConnectionCallbacks`和`OnConnectionFailedListener`。在`onCreate(Bundle)`中，实现一个这样的`ConnectionCallbacks`监听器，如代码清单31-11所示。

代码清单31-11 监听连接事件 (LocatrFragment.java)

```

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

    mClient = new GoogleApiClient.Builder(getActivity())
        .addApi(LocationServices.API)
        .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
            @Override
            public void onConnected(Bundle bundle) {
                getActivity().invalidateOptionsMenu();
            }
        })
        @Override
        public void onConnectionSuspended(int i) {
            ...
        }
    .build();
}

```

好奇的话，还可以实现一个`OnConnectionFailedListener`监听器，看看会得到什么结果。当然，这不是必须的。

现在，Google Play服务可用。

31.7 基于地理位置的 Flickr 搜索

接下来是实现基于地理位置的Flickr搜索。要完成这个任务，除了常规搜索功能外，还要附加经纬度地理位置信息。

在Android中，定位API是在`Location`中封装这些地理位置信息的。所以，新写一个`buildUrl(...)`覆盖方法，从传入的`Location`对象中取出地理位置信息，创建我们需要的`searchUrl`，如代码清单31-12所示。

代码清单31-12 创建`buildUrl(Location)`方法（`FlickrFetchr.java`）

```
private String buildUrl(String method, String query) {
    ...

    private String buildUrl(Location location) {
        return ENDPOINT.buildUpon()
            .appendQueryParameter("method", SEARCH_METHOD)
            .appendQueryParameter("lat", "" + location.getLatitude())
            .appendQueryParameter("lon", "" + location.getLongitude())
            .build().toString();
    }
}
```

然后，再写一个匹配的`searchPhotos(Location)`方法，如代码清单31-13所示。

代码清单31-13 创建`searchPhotos(Location)`方法（`FlickrFetchr.java`）

```
public List<GalleryItem> searchPhotos(String query) {
    ...

    public List<GalleryItem> searchPhotos(Location location) {
        String url = buildUrl(location);
        return downloadGalleryItems(url);
    }
}
```

31.8 获取定位数据

现在，万事俱备，只缺地理位置信息了。有一个类名叫`FusedLocationProviderApi`，顾名思义，要使用Fused Location Provider API就全靠它了。这个类有一个名为`FusedLocationApi`的实例，它是`LocationServices`对象中的一个单例对象。

要通过这个API获取地理位置信息，首先要使用`LocationRequest`对象创建一个定位请求。编写一个`findImage()`方法创建并配置我们需要的定位请求，如代码清单31-14所示。（有两个

LocationRequest类可用,这里要用com.google.android.gms.location.LocationRequest类。)

代码清单31-14 创建定位请求 (LocatrFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}

private void findImage() {
    LocationRequest request = LocationRequest.create();
    request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    request.setNumUpdates(1);
    request.setInterval(0);
}
}
```

有好几个参数可以配置LocationRequest定位请求对象。

- 时间间隔 (interval): 地理位置更新的频繁度。
- 更新次数 (number of updates): 地理位置应该更新多少次。
- 优先级 (priority): 是省电优先, 还是定位精准度优先。
- 失效 (expiration): 定位请求是否会失效。如果会失效, 在什么时候失效。

□ 最小位移 (smallest displacement): 触发位置更新, 设备需移动的最小距离 (以米单位)。

默认创建的LocationRequest定位请求的精度是街区级的, 更新也很慢。所以, 通过修改优先级、更新次数和时间间隔, 我们创建了既精准又快速的定位请求。

接下来就是发出定位请求并从Location那里监听反馈。这需要添加一个LocationListener。有两个版本的LocationListener可用, 这里选择的是com.google.android.gms.location.LocationListener, 如代码清单31-15所示。

代码清单31-15 发送定位请求 (LocatrFragment.java)

```
public class LocatrFragment extends Fragment {
    private static final String TAG = "LocatrFragment";
    ...

    private void findImage() {
        LocationRequest request = LocationRequest.create();
        request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
        request.setNumUpdates(1);
        request.setInterval(0);
        LocationServices.FusedLocationApi
            .requestLocationUpdates(mClient, request, new LocationListener() {
                @Override
                public void onLocationChanged(Location location) {
                    Log.i(TAG, "Got a fix: " + location);
                }
            });
    }
}
```

如果是要持续一段时间的定位请求,那就要不断监听,并在合适的时候调用removeLocationUpdates(...)方法取消定位请求。不过,既然我们使用了setNumUpdates(1),请求发出后就不用管它了。

最后,编码让搜索按钮发出定位请求。覆盖onOptionsItemSelected(...)方法并在其中调用findImage()方法,如代码清单31-16所示。

代码清单31-16 关联使用搜索按钮 (LocatrFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    ...
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_locate:
            findImage();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

运行应用并点击搜索按钮。如果使用的是模拟器,不要忘了先运行MockWalker应用。(遇到菜单方面的问题的话,请参见第13章导入AppCompat库解决。)查看日志,应该看到以下类似数据:

```
...D/libEGL: loaded /system/lib/egl/libGLESv2_MRVL.so
...D/GC: <tid=12423> OES20 ===> GC Version : GC Ver rls_pxa988_KK44_GC13.24
...D/OpenGLRenderer: Enabling debug mode 0
...I/LocatrFragment: Got a fix: Location[fused 33.758998,-84.331796 acc=38 et=...]
```

在日志中,可以看到定位的数据有经纬度、精确度以及定位时间。在Google地图里输入经纬度数据,就可以看到当前所处的具体位置了(如图31-8所示)。

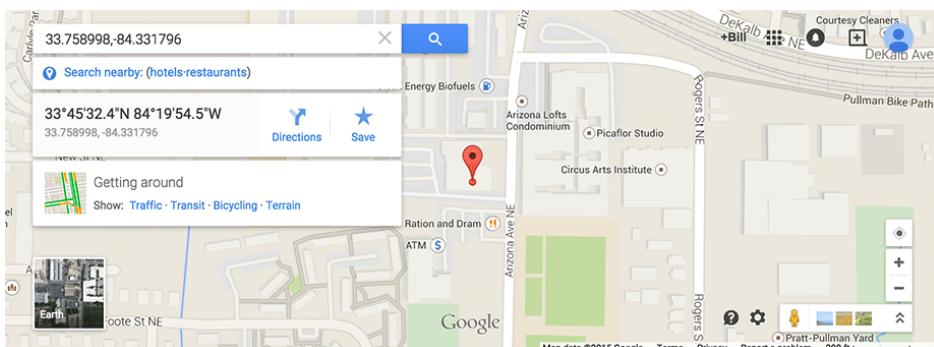


图31-8 当前所处位置

31.9 寻找并显示图片

有了定位数据，我们下面来使用它。编写一个异步任务找到当前地理位置的`GalleryItem`，下载它关联的图片并显示出来。

如代码清单31-17所示，编写一个`SearchTask`内部类。执行搜索，如果有返回数据，取出第一个`GalleryItem`。

代码清单31-17 编写`SearchTask`内部类（LocatrFragment.java）

```
private void findImage() {
    ...
    LocationServices.FusedLocationApi
        .requestLocationUpdates(mClient, request, new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                Log.i(TAG, "Got a fix: " + location);
                new SearchTask().execute(location);
            }
        });
}

private class SearchTask extends AsyncTask<Location,Void(Void> {
    private GalleryItem mGalleryItem;

    @Override
    protected Void doInBackground(Location... params) {
        FlickrFetchr fetchr = new FlickrFetchr();
        List<GalleryItem> items = fetchr.searchPhotos(params[0]);

        if (items.size() == 0) {
            return null;
        }

        mGalleryItem = items.get(0);

        return null;
    }
}
```

现在，保存`GalleryItem`并没有什么用处，但下一章会用到它。

接下来，下载`GalleryItem`关联的图片数据并解析出图片。然后，在`onPostExecute(Void)`方法中，使用`mImageView`显示图片，如代码清单31-18所示。

代码清单31-18 下载并显示图片（LocatrFragment.java）

```
private class SearchTask extends AsyncTask<Location,Void(Void> {
    private GalleryItem mGalleryItem;
    private Bitmap mBitmap;

    @Override
    protected Void doInBackground(Location... params) {
```

```
...
mGalleryItem = items.get(0);

try {
    byte[] bytes = fetchr.getUrlBytes(mGalleryItem.getUrl());
    mBitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
} catch (IOException ioe) {
    Log.i(TAG, "Unable to download bitmap", ioe);
}
return null;
}

@Override
protected void onPostExecute(Void result) {
    mImageView.setImageBitmap(mBitmap);
}
}
```

有了这些，就应该能在Flickr上找到附近的图片了，如图31-9所示。启动Locatr应用并点击你的位置按钮吧。



图31-9 应用完成了

31.10 挑战练习：进度指示器

点击搜索按钮后，对于Locatr应用接下来干了什么，用户一无所知。一个好的应用最好能在用户操作时立即给予有效的反馈。

请优化Locatr应用，在用户点击搜索按钮后，立即显示一个搜索进度条。可使用Progress-Dialog类来展示一个旋转的进度指示。另外，还应跟踪SearchTask的运行状态，在有搜索结果返回时，立即清除状态指示。

本章，我们将继续完善LocatrFragment。除了搜索并显示附近的图片外，还要找到图片的经纬度数据并在地图上标出。

32.1 导入 Play 地图服务库

继续学习之前，先导入地图库。这是另一个Play服务库。新添加的依赖库名为`com.google.android.gms:play-services-maps:7.0.0`，输入时不要搞错。上一章已提到过，由于Google的持续升级，Play服务库的版本号一直在变。开发时，请尽量使用最新版本的类库。

32.2 Android 上的地图服务

能获取定位数据确定手机所处位置已足以让用户开心了；如果数据还能以图形化的方式直观显示，用户肯定会更加高兴。

地图应用应该是智能手机上的首个杀手级应用。这也是为什么Android从一开始就有地图服务的原因。

地图服务庞大繁杂，需要大型支持服务器系统来提供基础地图数据。大多数Android服务或应用都能独立为Android开源项目，唯独地图不太可能。

因此，虽然Android一直内置地图服务，但地图API却一直是以独立的Android API存在的。当前版本的Maps v2 API和Fused Location Provider一起，都包含在Google Play服务里。所以，要使用它，要么有台安装了Play商店应用的设备，要么使用带有Google APIs的模拟器。

如果你在开发地图应用，碰巧翻到本章想看看有什么可以参考的，那么在继续之前，首先要确保已做到了这些（已在上一章完成）：

- 确保设备支持Play服务；
- 导入了合适的Play服务库；
- 适时使用`GooglePlayServicesUtil`，确保安装上最新版本的Play商店应用。

32.3 地图 API 设置

接下来我们继续。除了上一章的定位权限，地图API也需要其他一些权限设置。

地图API需要的权限有：

- 从网上下载地图数据（`android.permission.INTERNET`）
- 查询网络状态（`android.permission.ACCESS_NETWORK_STATE`）
- 把临时地图数据写入外部存储（`android.permission.WRITE_EXTERNAL_STORAGE`）

上一章已添加过`INTERNET`权限。所以，这里只需添加另外两个就可以了。

代码清单32-1 为地图服务添加权限（AndroidManifest.xml）

```
<?XML version="1.0"encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.locatr" >

    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission
        android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission
        android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    ...

```

获取 Maps API key

使用Maps API还需要在manifest文件中声明你自己的API key。这个API key可以授权你的应用使用Google地图服务。

要得到自己的专属API Key，首先要获取你签名key的散列值，然后在Google开发者终端上用它登记使用Google Maps v2 API。下一小节，我们会学习如何使用Android工具查看你自己的签名key。如何使用Google开发者终端不属于本书讨论范畴。不过，我们稍后会告诉你该查看哪些Web文档。

签名key是一个在数学概念上不好理解的数字串，可以唯一标识你自己。为确定应用拥有者的身份，Android要求所有安装到设备上的应用都要以唯一的签名key做签名。

目前为止，还轮不到我们操心如何签名。这是因为Android Studio已自动为我们创建了默认的签名key。通常，我们把这个签名key叫作调试key。每次Android Studio编译应用时，都先使用默认的调试key给APK签名，然后才会在设备上部署应用。

1. 查看签名key

执行一些命令行命令，我们就能使用Gradle方便地查看到签名key。

首先打开操作系统自带的命令行工具，使用`cd`命令进入项目文件所在的目录。在OS X上，命令应该像代码清单32-2这样（请自己替换用户名）。

代码清单32-2 进入随时文件目录（命令控制台）

```
$ cd /Users/bphillips/src/android/Locatr
```

然后使用一个gradle命令行工具获得一个签名报告。如果是Linux或OS X，就运行如代码清单32-3所示的这条命令。

代码清单32-3 Linux或OS X上的签名报告（命令控制台）

```
$ cd /Users/bphillips/src/android/Locatr
$ ./gradlew signingReport
```

如果是Windows系统，则应使用Windows目录结构并执行和代码清单32-4类似的命令。

代码清单32-4 Windows上的签名报告（命令控制台）

```
> cd c:\users\bphillips\Documents\android\Locatr
> gradlew.bat signingReport

$ ./gradlew signingReport
:app:signingReport
Variant: debug
Config: debug
Store: /Users/bphillips/.android/debug.keystore
Alias: AndroidDebugKey
MD5: XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
SHA1: XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
Valid until: Friday, May 16, 2042
-----
Variant: release
Config: none
-----
Variant: debugTest
Config: debug
Store: /Users/bphillips/.android/debug.keystore
Alias: AndroidDebugKey
MD5: XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
SHA1: XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX:XX
Valid until: Friday, May 16, 2042
-----

BUILD SUCCESSFUL

Total time: 4.354 secs
```

在实际输出报告中，你会看到以16进制数字表示的MD5和SHA1值（上述报告中的XX仅作示例使用）。注意，稍后，我们会使用上述报告中加亮部分的debug SHA1值来获取API key。

2. 获取API key

有了debug SHA1值，就可以获取API key了。具体如何操作，请参阅Google文档：

<https://developers.google.com/maps/documentation/android/start>

按照Google官方指令操作完成后，你应该会得到对应调试签名key的API key。打开AndroidManifest.xml文件，参照代码清单32-5添加刚才获得的API key。

代码清单32-5 添加API key (AndroidManifest.xml)

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"/>
    ...
</application>
```

地图API设置做完了，下面开始学习创建和使用地图。

32.4 创建地图

首先来创建地图。地图显示在MapView中。MapView的使用和其他视图类差不多，但有一点例外：你必须像下面这样转发所有的生命周期方法。

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

    mMapview.onCreate(savedInstanceState);
}
```

这简直太痛苦了。不过，我们可以转而使用MapFragment，或者使用支持库版SupportMapFragment。痛苦的事就交给SDK去做吧。MapFragment会为我们创建和托管MapView，当然还包括前面说过的生命周期方法回调。

要使用SupportMapFragment，首先要废除原来的用户界面。尽管这听上去像个大工程，实际做起来却很简单。只要改为继承SupportMapFragment类，删除onCreateView(...)方法，再删除所有使用ImageView的代码就行了，如代码清单32-6所示。

代码清单32-6 改用SupportMapFragment (LocatrFragment.java)

```
public class LocatrFragment extends SupportMapFragment {
    private static final String TAG = "LocatrFragment";

    private ImageView mImageView;
    private GoogleApiClient mClient;

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
```

```
View v = inflater.inflate(R.layout.fragment_locatr, container, false);
mImageView = (ImageView) v.findViewById(R.id.image);

return v;
}

...
private class SearchTask extends AsyncTask<Location,Void(Void> {
    ...

    @Override
    protected void onPostExecute(Void result) {
        mImageView.setImageBitmap(mBitmap);
    }
}
}
```

SupportMapFragment自己会覆盖onCreateView(...)方法，改造任务也就完成了。运行Locatr应用，可以看到如图32-1所示的地图。



图32-1 一幅地图

32.5 获取更多地理位置数据

为在地图上标注图片，需要知道图片的地理位置。再添加一个extra参数给Flickr API查询串，为GalleryItem取回经纬度值，如代码清单32-7所示。

代码清单32-7 添加经纬度查询参数 (FlickrFetchr.java)

```
private static final String API_KEY = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
private static final String FETCH_RECENTS_METHOD = "flickr.photos.getRecent";
private static final String SEARCH_METHOD = "flickr.photos.search";
private static final Uri ENDPOINT = Uri.parse("https://api.flickr.com/services/rest/")
    .buildUpon()
    .appendQueryParameter("api_key", API_KEY)
    .appendQueryParameter("format", "json")
    .appendQueryParameter("nojsoncallback", "1")
    .appendQueryParameter("extras", "url_s,geo")
    .build();
```

现在，为GalleryItem添加经纬度属性，如代码清单32-8所示。

代码清单32-8 添加经纬度属性 (GalleryItem.java)

```
public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;
    private double mLat;
    private double mLon;

    ...

    public void setId(String id) {
        mId = id;
    }

    public double getLat() {
        return mLat;
    }

    public void setLat(double lat) {
        mLat = lat;
    }

    public double getLon() {
        return mLon;
    }

    public void setLon(double lon) {
        mLon = lon;
    }

    @Override
    public String toString() {
        return mCaption;
    }
}
```

然后，从返回的Flickr JSON数据取出经纬度值，如代码清单32-9所示。

代码清单32-9 取出经纬度值 (FlickrFetchr.java)

```

private void parseItems(List<GalleryItem> items, JSONObject jsonBody)
    throws IOException, JSONException {

    JSONObject photosJsonObject = jsonBody.getJSONObject("photos");
    JSONArray photoJsonArray = photosJsonObject.getJSONArray("photo");

    for (int i = 0; i < photoJsonArray.length(); i++) {
        JSONObject photoJsonObject = photoJsonArray.getJSONObject(i);

        GalleryItem item = new GalleryItem();
        item.setId(photoJsonObject.getString("id"));
        item.setCaption(photoJsonObject.getString("title"));

        if (!photoJsonObject.has("url_s")) {
            continue;
        }

        item.setUrl(photoJsonObject.getString("url_s"));
        item.setLat(photoJsonObject.getDouble("latitude"));
        item.setLon(photoJsonObject.getDouble("longitude"));

        items.add(item);
    }
}

```

搞定了图片地理位置信息。, 接下来在主fragment中添加一些实例变量用来保存搜索结果的。如代码清单32-10所示，一个用于保存待显示的Bitmap，一个用于GalleryItem，还有一个用于Location地理位置。

代码清单32-10 添加地图数据 (LocatrFragment.java)

```

public class LocatrFragment extends SupportMapFragment {
    private static final String TAG = "LocatrFragment";

    private GoogleApiClient mClient;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;

    ...

```

然后，在SearchTask类中，保存这些地图数据，如代码清单32-11所示。

代码清单32-11 保存查询结果 (LocatrFragment.java)

```

private class SearchTask extends AsyncTask<Location,Void,Void> {
    private Bitmap mBitmap;
    private GalleryItem mGalleryItem;
    private Location mLocation;

    @Override

```

```

protected Void doInBackground(Location... params) {
    mLocation = params[0];
    FlickrFetchr fetchr = new FlickrFetchr();
    ...
}

@Override
protected void onPostExecute(Void result) {
    mMapImage = mBitmap;
    mMapItem = mGalleryItem;
    mCurrentLocation = mLocation;
}
}

```

至此，所有要用到的地图数据都有了。接下来的任务就是在地图上显示出来。

32.6 使用地图

`SupportMapFragment`会创建`MapView`，而`MapView`又会去托管真正做事的`GoogleMap`。所以，我们需要引用到`GoogleMap`对象。调用`getMapAsync(OnMapReadyCallback)`方法可以获取到`GoogleMap`对象，如代码清单32-12所示。

代码清单32-12 获取`GoogleMap` (`LocatrFragment.java`)

```

public class LocatrFragment extends SupportMapFragment {
    private static final String TAG = "LocatrFragment";

    private GoogleApiClient mClient;
    private GoogleMap mMap;
    private Bitmap mMapImage;
    private GalleryItem mMapItem;
    private Location mCurrentLocation;

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

        mClient = new GoogleApiClient.Builder(getActivity())
            ...
            .build();

        getMapAsync(new OnMapReadyCallback() {
            @Override
            public void onMapReady(GoogleMap googleMap) {
                mMap = googleMap;
            }
        });
    }
}

```

顾名思义，`SupportMapFragment.getMapAsync(...)`方法可以异步获取到地图对象。如果在`onCreate(Bundle)`方法中调用这个方法，地图一旦完成创建和初始化，我们就会取到它。

既然已获取到了`GoogleMap`对象，根据`LocatrFragment`的当前状态，我们可以更新地图的展示。首先是放大显示某个目标区域。另外，我们还想在目标区域周围留出一定间距，所以再添加一个间距尺寸值，如代码清单32-13所示。

代码清单32-13 添加间距 (res/values/dimens.xml)

```
<resources>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="map_inset_margin">100dp</dimen>
</resources>
```

然后，添加一个`updateUI()`方法执行放大动作，如代码清单32-14所示。

代码清单32-14 放大 (LocatrFragment.java)

```
private void findImage() {
    ...
}

private void updateUI() {
    if (mMap == null || mMapImage == null) {
        return;
    }

    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
        mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());

    LatLngBounds bounds = new LatLngBounds.Builder()
        .include(itemPoint)
        .include(myPoint)
        .build();

    int margin = getResources().getDimensionPixelSize(R.dimen.map_inset_margin);
    CameraUpdate update = CameraUpdateFactory.newLatLngBounds(bounds, margin);
    mMap.animateCamera(update);
}

private class SearchTask extends AsyncTask<Location,Void(Void> {
```

...

下面来详细解读上述代码。为四处移动`GoogleMap`，我们创建了一个`CameraUpdate`对象。`CameraUpdateFactory`有好几个静态方法可用。通过改变位置、放大级别以及一些其他属性，可以使用这些静态方法创建不同的`CameraUpdate`对象。

这里，我们创建了一个指向特定`LatLngBounds`的`CameraUpdate`对象。可以把`LatLngBounds`看作一个包围某个坐标的矩形框。指定西南角和东北角的坐标，就能显式地创建一个`LatLngBounds`。

通常来讲，提供矩形框包围的一系列坐标会更容易些。`LatLngBounds.Builder`可以轻松做到这一点：只要创建一个`LatLngBounds.Builder`，然后针对`LatLngBounds`要包围的每一个坐标调用`.include(LatLng)`方法。最后，再调用`build()`方法，就能得到一个配置好的`LatLngBounds`。

搞定了包围框，就看如何缩放地图了。有两种方式待选：`moveCamera(CameraUpdate)`或`animateCamera(CameraUpdate)`。以动画的方式更有趣些，所以自然是选择`animateCamera(CameraUpdate)`方法。

接下来，参照代码清单32-15安排在两个地方调用`updateUI()`方法：首次获得地图时，以及搜索完成时。

代码清单32-15 调用`updateUI()`方法（LocatrFragment.java）

```

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

    getMapAsync(new OnMapReadyCallback() {
        @Override
        public void onMapReady(GoogleMap googleMap) {
            mMap = googleMap;
            updateUI();
        }
    });
}

...

private class SearchTask extends AsyncTask<Location,Void(Void> {
    ...

    @Override
    protected void onPostExecute(Void result) {
        mMapImage = mBitmap;
        mMapItem = mGalleryItem;
        mCurrentLocation = mLocation;

        updateUI();
    }
}

```

运行Locatr应用并点击搜索按钮。地图应该呈现并放大了一块包含你当前所在位置的区域(如图32-2所示)。(注意，模拟器用户需要先让MockWalker运行起来。)

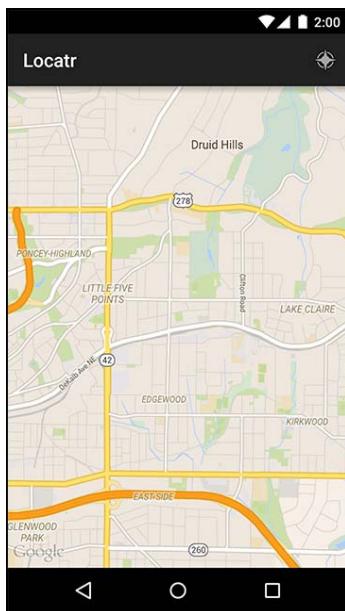


图32-2 放大版地图

在地图上绘制

地图看上去真不错，但也仅仅是副地图而已。当前，你肯定正处在这幅地图的某个位置，Flickr图片也是。可是，到底在哪呢？好吧，为了能真切地看到，我们在地图上标注出来。

在地图上绘制和在普通视图上绘制是两个概念，前者相对更容易。在普通视图上绘制是绘制像素到屏幕上，而地图绘制则是在某个地理位置区域添加对象。使用“绘制”这个字眼，我们的解读是：“首先创建一些对象，然后添加到GoogleMap上，GoogleMap随后会自动绘制出它们。”

其实上面的说法也不够准确。事实上，添加到GoogleMap上的对象也是由GoogleMap创建的。我们创建的对象是用来告诉GoogleMap我们想要创建什么的。这是种描述性对象，又称options objects。

创建两个MarkerOptions对象。然后，调用mMap.addMarker(MarkerOptions)方法在地图上添加它们，如代码清单32-16所示。

代码清单32-16 地图标注（LocatrFragment.java）

```
private void updateUI() {
    ...
    LatLng itemPoint = new LatLng(mMapItem.getLat(), mMapItem.getLon());
    LatLng myPoint = new LatLng(
        mCurrentLocation.getLatitude(), mCurrentLocation.getLongitude());
```

```

BitmapDescriptor itemBitmap = BitmapDescriptorFactory.fromBitmap(mMapImage);
MarkerOptions itemMarker = new MarkerOptions()
    .position(itemPoint)
    .icon(itemBitmap);
MarkerOptions myMarker = new MarkerOptions()
    .position(myPoint);

mMap.clear();
mMap.addMarker(itemMarker);
mMap.addMarker(myMarker);

LatLngBounds bounds = new LatLngBounds.Builder()
...
}

```

调用`addMarker(MarkerOptions)`方法时，`GoogleMap`会创建`Marker`实例并添加到地图上。如果需要删除或修改`Marker`，应该保存这个`Marker`实例。每次调`updateUI()`方法时，我们会清除地图，所以这里就不需要保存它了。

运行Locatr应用并点击搜索按钮。可以看到地图上的两个标注出现了，如图32-3所示。

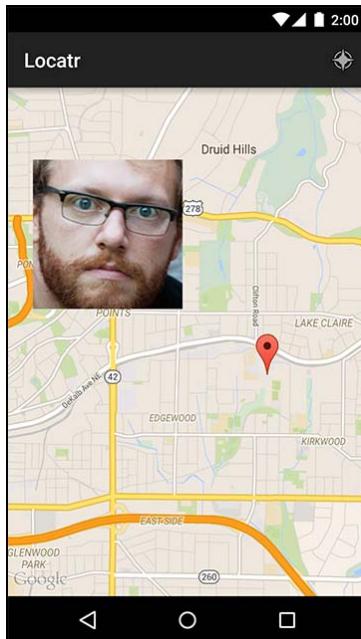


图32-3 带标注的地图

终于，Locatr应用完工了。借助这个应用，你知道该如何使用定位和地图这两个Play服务了：你定位过手机位置，登记使用过Google的Web services API，还学会了地图标注。辛苦了，可以休息一会儿了。

32.7 深入学习：团队开发和API key

团队合作开发使用API key的应用时，管理API key就会成为一件麻烦事。你的签名凭证会保留在你唯一拥有密钥文件中。团队成员也同样拥有他们自己的密钥文件。每当有新人加入，都得找他们要SHA1，然后更新你自己的API key凭证。

虽然麻烦，但这起码是个可用的API key管理办法：在项目中管理所有的签名散列值。如果你想借此明确地控制团队成员的开发行为，这未尝不是个合适的方法。

但是，这里还有另外一个方法推荐：创建项目专用调试密钥文件。这个方法首先需要使用Java的keytool创建一个新的调试密钥文件，如代码清单32-17所示。

代码清单32-17 创建新的调试密钥文件（终端）

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
-storepass android -keypass android -keyalg RSA -validity 14600
```

命令执行后，keytool会抛出一堆问题。据实回答即可。（既然是调试key，除了名字外，其他都可以使用默认值。）

```
$ keytool -genkey -v -keystore debug.keystore -alias androiddebugkey \
-storepass android -keypass android -keyalg RSA -validity 14600
What is your first and last name?
[Unknown]: Bill Phillips
...
```

得到了debug.keystore文件后，把它移到应用模块目录。然后，打开项目结构界面，选择app模块，再选择Signing选项页。点击+按钮添加签名配置。Name栏位输入debug，Store File栏位输入debug.keystore（如图32-4所示）。

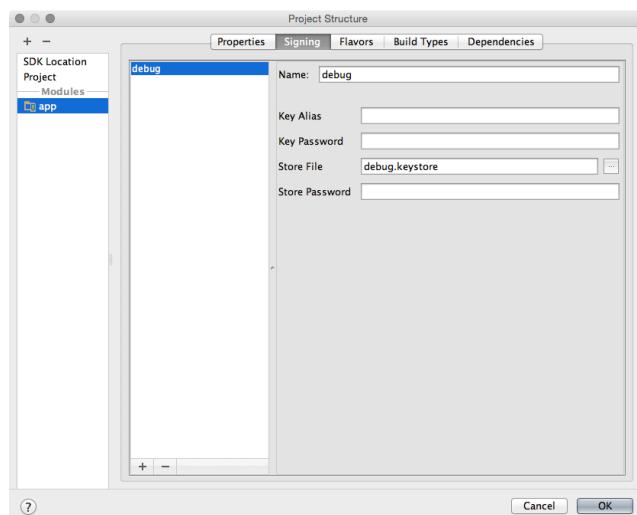


图32-4 配置debug签名key

经过这样的配置，任何人只要使用同样的keystore文件，就能使用同样的API key了。是不是很方便呢？

注意，如果采用这种方式管理API key，在分发debug.keystore时就要当心了。如果仅在团队内部分享，问题不大。但千万不要把keystore发布到公共仓库：既然任何人都可以访问，那他们自然也就可以使用你的API key了。

material design 33

Android 5.0（代号Lollipop）最大的变化是引入了全新的material design设计风格。这种新的视觉设计语言一经推出，立即引爆了设计界。为方便学习和开发，Google还提供了详尽的设计指南。

当然，开发人员一般不用操心设计问题。设计素材是什么并不重要，只要在应用里用好它们就行了。然而，随着material design的推出，老观念或许要改一改了。除了全新的用户交互概念，material design还要求使用者具备一定的设计敏感性。一旦熟悉了这种新的设计语言，设计的应用实施会更加得心应手。

和前面不同，本章完全可看作本书最大的深入学习专题。所以，本章不涉及实例开发，而且稍后要介绍的大多数内容是可以选读的。

对设计者来说，material design强调以下三大设计原则。

- 实体隐喻（拟物化）：应用部件的运作应体现实物感。
- 醒目、形象、有目的性：如装帧美观、设计精良的杂志或图书带来的体验一样，应用设计元素也应有跃然纸上的观感。
- 动画就是表现力：动态响应用户的操作。

醒目、形象及有目的性这条原则适用于设计人员，本书不作讨论。如果你是全能型人才，打算自己设计应用，请仔细阅读material design指导手册把握该原则。

就实体隐喻原则来讲，有了开发人员的参与，设计人员才能创造出更好的拟物化界面。作为开发者，我们需要知道如何使用Z轴属性布置三维界面，以及如何使用floating action bar和snackbar这两个新的material组件。

最后是动画表现力这条原则。为更好地把握该原则，我们还要学习一些新的动画工具：state list animator、animated state list drawable（是的，你没看错，state list animator和animated state list drawable是不同的工具）、circular reveal以及shared element transition。这些工具能给应用增添许多视觉动画特效。相信我，平面设计者再怎么努力也只有羡慕的份了。

33.1 material surface

作为开发人员，material design中的material surface是要重点把握的关键概念。在设计者眼中，material surface如同1dp厚的纸板。这些纸板如同真的纸墨那样，可以变大，也可以显示动画和动态文字，如图33-1所示。



图33-1 有两个material surface的界面

然而，和真实纸张一样，虚拟的material surface再怎么神奇，它的行为也不应违背物理世界的准则。例如，一张纸绝不可能穿过另一张纸。同理，创造动画特效时，material surface也要遵守这一原则。

因此，在三维空间里，surface的位置摆放以及它们之间的互动要注意处理好。以使用者手指为参照对象，surface可以上下移动或是移走，如图33-2所示。

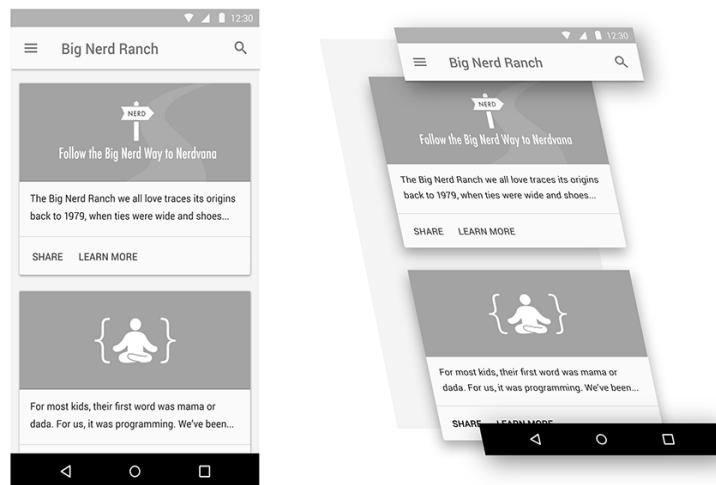


图33-2 三维空间里的material design

要制作一个surface越过另一个的动画特效，可直接向上移动它，然后越过另一个surface，如图33-3所示。

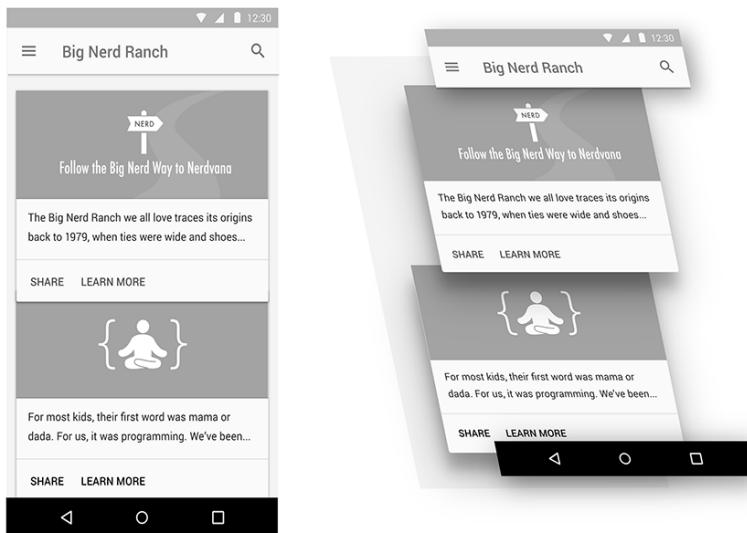


图33-3 surface越过动画

33.1.1 elevation 和 Z 值

应用界面元素间的投影最能让用户清晰地感知用户界面的深度。投影该如何实现呢？有些人第一反应是：这是设计人员的事，开发人员直接调用就行了。他们这样想或许有他们的道理，无论如何，有了分歧就要具体问题具体分析。

稍加分析可知，哪怕是简单的应用，也涉及大量的surface动画特效，处理这样千变万化的投影简直是巨大的工作量。显然，交给设计人员去绘制肯定不现实。实际上，只要给每个视图设置elevation，Android就可以帮我们实现阴影绘制。

随着Lollipop系统的发布，Android为布局系统引入了Z轴概念。这允许我们在三维空间里布置视图。如图33-4所示，elevation类似赋予布局视图的坐标：视图可以动态远离其原始坐标，但其原始位置是不变的。



图33-4 Z平面上的elevation

可以使用View.setElevation(float)方法或在布局XML文件中设置elevation值，如代码清单33-1所示。

代码清单33-1 在布局文件中设置elevation值

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:elevation="2dp"/>
```

因为elevation值要作为Z基准值使用，所以我们最好采用设置XML属性值的方式。而且，相比setElevation(float)方法，这种方式使用灵活，Lollipop以前版本的系统会默认忽略android:elevation属性，因此，比较难以对付的兼容性问题也就不用考虑了。

要修改View视图的elevation，我们可以使用translationZ和Z属性。它们的用法和第30章介绍的translationX、translationY、X和Y一样。如图33-5所示，Z值总是等于elevation加上translationZ。如果给Z一个值，那么系统会自动计算得出translationZ值。

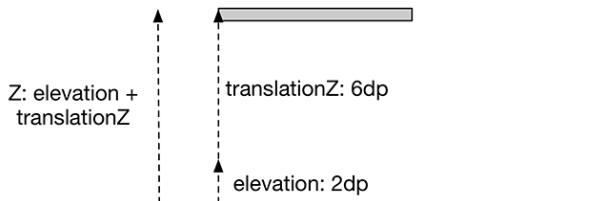


图33-5 Z和translationZ

33.1.2 state list animator

使用material design的应用包含许多用户交互动画特效。例如，在Lollipop设备上，按住按钮时，按钮会基于Z轴向用户手指移动，松开时则会后退。

为方便应用这样的动画特效，Lollipop引入了state list animator。和state list drawable交替出现的动画效果不同，它只是改变视图的状态。只要在res/animator中定义一个state list animator，就能实现按钮浮出的动画效果，如代码清单33-2所示。

代码清单33-2 state list animator 使用示例

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true">
        <objectAnimator android:propertyName="translationZ"
            android:duration="100"
            android:valueTo="6dp"
            android:valueType="floatType"
            />
    </item>
    <item android:state_pressed="false">
        <objectAnimator android:propertyName="translationZ"
            android:duration="100"
            android:valueTo="2dp"
            android:valueType="floatType"
            />
    </item>
</selector>
```

```

        android:duration="100"
        android:valueTo="0dp"
        android:valueType="floatType"
    />
</item>
</selector>
```

对属性动画来说，这非常有用。假設想实施帧动画，我们还要使用animated state list drawable这个动画工具。

这个工具的名字很容易让人迷惑。听起来和state list animator差不多，但实际用法大不相同。类似于常规的state list drawable，使用animated state list drawable可以为视图的不同状态定义不同的图片，甚至是定义状态间的帧动画转场。

在第21章，BeatBox应用的声音按钮是使用state list drawable定义的。如果你是个设计狂，没准还想实现每次按钮按下时的多帧动画。没问题，参考代码清单33-3修改XML文件就可以了。不过，修改版XML文件必须放在res/drawable-21目录中，因为Lollipop之前的系统不支持这个功能。

代码清单33-3 animated state list drawable使用示例

```

<?xml version="1.0" encoding="utf-8"?>
<animated-selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/pressed"
          android:drawable="@drawable/button_beat_box_pressed"
          android:state_pressed="true"/>
    <item android:id="@+id/released"
          android:drawable="@drawable/button_beat_box_normal" />

    <transition
        android:fromId="@+id/released"
        android:toId="@+id/pressed">
        <animation-list>
            <item android:duration="10" android:drawable="@drawable/button_frame_1" />
            <item android:duration="10" android:drawable="@drawable/button_frame_2" />
            <item android:duration="10" android:drawable="@drawable/button_frame_3" />
            ...
        </animation-list>
    </transition>
</animated-selector>
```

注意，在上述代码中，animated-selector元素里的每个item都有ID。在不同的ID间定义转场就可以实现多帧动画。如果还想实现按钮释放动画，那就再添加一个transition标签。

33.2 动画工具

material design引入了很多漂亮的动画特效。实现时，有些很容易，有些不容易，需要花点力气。不过，如果能善用Android提供的便利工具，事情就好办得多。

33.2.1 circular reveal

circular reveal动画看起来就像墨滴在一张纸上向外快速扩散。从一个交互点出发（通常是用户的按压点），视图或是一段文字向外扩散式显现。模拟效果如图33-6所示。

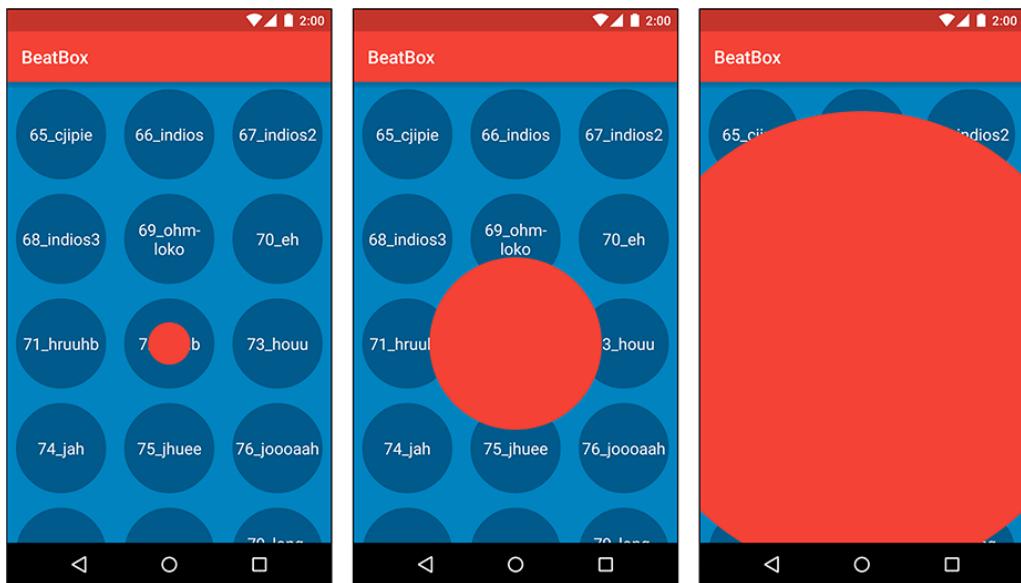


图33-6 在BeatBox应用中模拟circular reveal动画特效

要创建circular reveal动画特效，可调用ViewAnimationUtils的createCircularReveal(...)方法。该方法有5个参数：

```
static Animator createCircularReveal(View view, int centerX, int centerY,
                                    float startRadius, float endRadius)
```

第一个View参数就是要向外扩散显现的视图。在图33-6中，这个视图就是和BeatBoxFragment宽高一致的红色实心视图。如果动画从startRadius（值为0）圆点开始到endRadius结束，这个红点视图会先变为透明状态，并随着一个不断放大的圆慢慢显现。centerX和centerY是这个圆的圆点坐标（也就是View的坐标）。该方法会返回一个Animator（和第30章用过的Animator一样）。

material design指南指出，circular reveal动画应该开始于用户手指在屏幕上的触点。所以，首先要找到用户点击视图的坐标，如代码清单33-4所示。

代码清单33-4 找到点击视图坐标

```
@Override
public void onClick(View clickSource) {
    int[] clickCoords = new int[2];
```

```

// Find the location of clickSource on the screen
clickSource.getLocationOnScreen(clickCoords);

// Tweak that location so that it points at the center of the view,
// not the corner
clickCoords[0] += clickSource.getWidth() / 2;
clickCoords[1] += clickSource.getHeight() / 2;

performRevealAnimation(mViewToReveal, clickCoords[0], clickCoords[1]);
}

```

然后开始执行circular reveal动画，如代码清单33-5所示。

代码清单33-5 执行circular reveal动画

```

private void performRevealAnimation(View view, int screenCenterX, int screenCenterY) {
    // Find the center relative to the view that will be animated
    int[] animatingViewCoords = new int[2];
    view.getLocationOnScreen(animatingViewCoords);
    int centerX = screenCenterX - animatingViewCoords[0];
    int centerY = screenCenterY - animatingViewCoords[1];

    // Find the maximum radius
    Point size = new Point();
    getActivity().get WindowManager().getDefaultDisplay().getSize(size);
    int maxRadius = size.y;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        ViewAnimationUtils.createCircularReveal(view, centerX, centerY, 0, maxRadius)
            .start();
    }
}

```

注意，成功调用`createCircularReveal(...)`方法的前提条件是，布局中已有目标视图。

33.2.2 shared element transition

shared element transition（又称为hero transition）是material design中引入的另一新动画特效。它适用于一种特殊场景：两个待切换视图显示同样一些元素。

前面，在CriminalIntent应用中，crime记录的明细界面会显示一张缩略图。当时的一个挑战练习要求弹出一个新视图展示全尺寸照片。大家完成后的效果图很可能和图33-7一样。

这是一种常见的交互模式：点击一个元素，弹出新的视图显示元素明细。

对于两个展示相同元素的视图，其间的任何动画切换场景都可以使用shared element transition来实现。图33-7中，右边的大图片和左边的小图片是同一张图。这张图就是个shared element。

在Lollipop中，Android有办法实现activity或fragment间的动画切换。图33-8是动画过程中的一副截图，可大致看出实现效果。现在一起来看看如何把这种动画应用于activity。

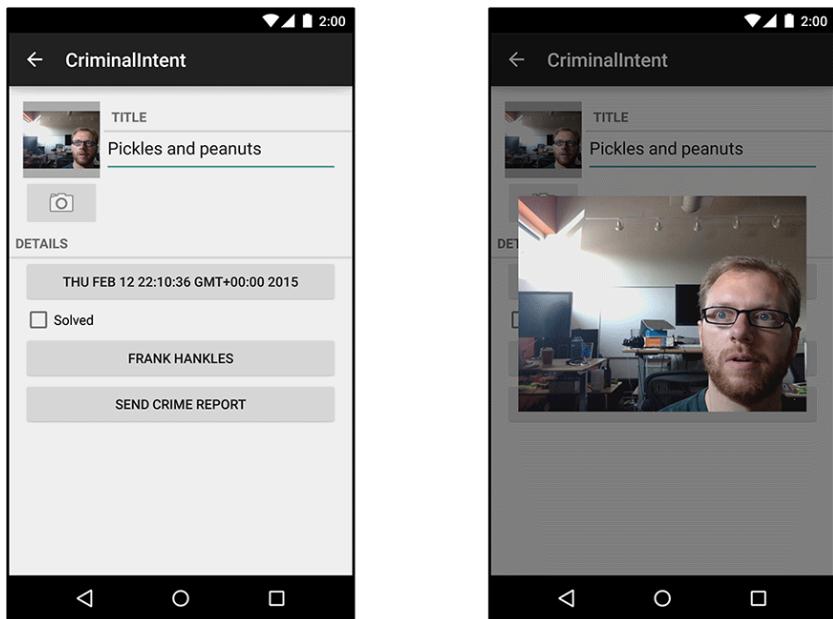


图33-7 放大版照片视图

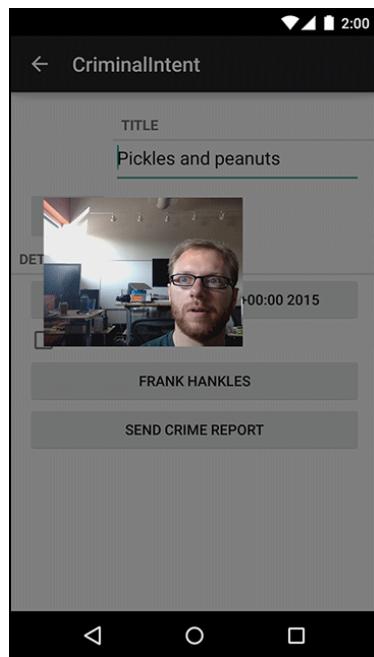


图33-8 shared element的变换

实现activity间的动画切换涉及以下三个步骤：

- 打开activity transition；
- 为每个shared element视图设置transition名值；
- 启动带ActivityOptions（触发动画）的activity。

首先是打开activity transition。如果你的activity使用了AppCompat主题，这个步骤可以直接跳过。（AppCompat继承Material主题，会自动为你打开activity transition。）

在CriminalIntent的例子中，为了让目标activity拥有透明背景，我们使用了@android:style/Theme.Translucent.NoTitleBar主题样式。这个主题没有继承Material主题，所以需要手工打开activity transition。有两种方式可以打开activity transition，先来看如何用代码打开它，如代码清单33-6所示。

代码清单33-6 以代码的方式打开activity transition

```
@Override
public void onCreate(Bundle savedInstanceState) {
    getWindow().requestFeature(Window.FEATURE_ACTIVITY_TRANSITIONS);
    super.onCreate(savedInstanceState);

    ...
}
```

另外一种方式是修改activity的样式，设置android:windowActivityTransitions属性值为true，如代码清单33-7所示。

代码清单33-7 在样式里打开activity transition

```
<resources>
    <style name="TransparentTheme"
        parent="@android:style/Theme.Translucent.NoTitleBar">
        <item name="android:windowActivityTransitions">true</item>
    </style>

</resources>
```

再来看如何为shared element视图设置transition名值。Android在API 21中为View引入了transitionName属性。所以，可以在布局或代码中设置这个属性值。两种方式各有其适用的场景，具体问题具体分析。本例中，我们在布局XML文件里，将android:transitionName属性设置为image，如图33-9所示。

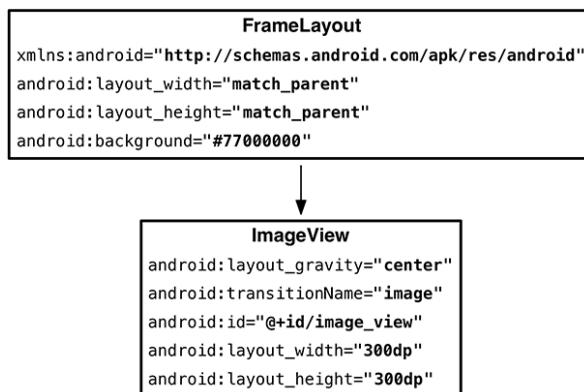


图33-9 设置`android:transitionName`属性值为`image`

然后，再定义一个`startWithTransition(...)`静态方法，为视图设置`transitionName`名称，如代码清单33-8所示。

代码清单33-8 定义`startWithTransition(...)`方法

```

public static void startWithTransition(Activity activity, Intent intent,
    View sourceView) {
    ViewCompat.setTransitionName(sourceView, "image");
    ActivityOptionsCompat options = ActivityOptionsCompat
        .makeSceneTransitionAnimation(activity, sourceView, "image");

    activity.startActivity(intent, options.toBundle());
}
  
```

Android旧版本系统中，`View`视图没有`setTransitionName(String)`属性方法。所以，需要使用`ViewCompat.setTransitionName(View, String)`方法设置`TransitionName`。

在代码清单33-8中，作为三个步骤的最后一步，我们使用`ActivityOptions`对象，让操作系统知道`shared element`是什么，以及使用哪个`transitionName`值。

`transition`和`shared element transition`能做的远不止这些。例如，它们还可以用于`fragment`间的动画切换。欲了解更多，请阅读Google的`transition`框架文档：<https://developer.android.com/training/transitions/overview.html>。

33.3 新的视图组件

Lollipop的material design指南还设计了一些全新视图组件。Android设计团队已实现了一些。下面就一起来学习一下，没准哪天就会用上了。

33.3.1 card

首先学习的是`card`这个新组件，如图33-10所示。它是个`frame`容器组件。

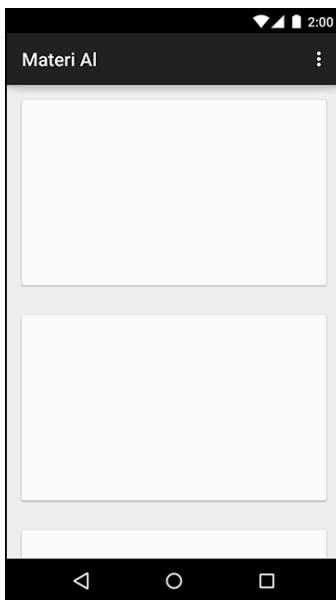


图33-10 Card组件

card容器组件用来装载其他内容。从上图可以看出，它有着圆圆的边角，微微浮出底平面，在身后投下了一圈阴影。

设计问题已超出本书讨论范畴，所以，该在什么时候、什么地方使用card，我们给不了建议。（想知道的话，可以查阅Google的material design文档：<http://www.google.com/design/spec>。）不过，我们会告诉你如何创建它：使用CardView。

类似RecyclerView，CardView是com.android.support:cardview-v7支持库中的一个视图类。使用前，首先要在项目中添加这个依赖库。

引入以后，就可以像使用布局中的ViewGroup一样使用CardView，如代码清单33-9所示。CardView是个FrameLayout子类，所以FrameLayout的任何布局参数它都可以用。

代码清单33-9 在布局中使用CardView

```
<LinearLayout 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"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <android.support.v7.widget.CardView
        android:id="@+id/item"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_margin="16dp"
    >
```

```

    ...
</android.support.v7.widget.CardView>

</LinearLayout>

```

既然是支持库类，在旧设备上，它自然有良好的兼容性。和其他组件不一样，CardView总带投影效果。（不过，在旧设备上，它只绘制自己，投影效果不明显。）感兴趣的话，可以查阅CardView文档了解它在新旧设备上还有哪些视觉差异。

33.3.2 floating action button

另一个常见组件是floating action button（FAB），如图33-11所示。

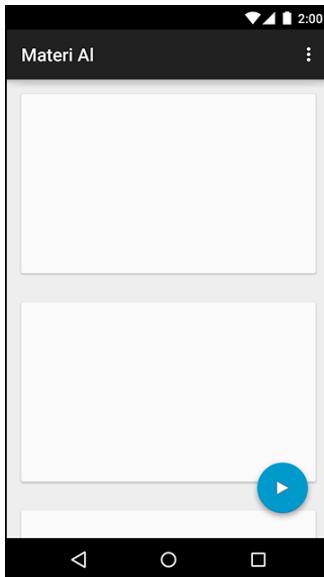


图33-11 一个floating action button

floating action button是在Google设计支持库中实现的。同样，使用前，首先要在项目中引入com.android.support:design:22.2.0这个依赖库。

floating action button可以看作是由一个实心圆和一个OutlineProvider提供的圆形投影所组成。作为ImageView子类，FloatingActionButton类负责生成实心圆和阴影。所以，创建floating action button很简单：在布局文件中放入FloatingActionButton，并设置它的src属性指向要显示在按钮上的图片。

如果把floating action button放在FrameLayout中，设计支持库还会引入一个比较智能的CoordinatorLayout。这个FrameLayout子类布局会随其他控件的移动智能地改变floating action button的位置。这样，如果要展示一个Snackbar，FAB会自动上移以避免被Snackbar挡住。具体实施代码如代码清单33-10所示。

代码清单33-10 布局floating action button

```
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    [... main content here ...]
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/floating_action_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:layout_margin="16dp"
        android:src="@drawable/play"/>
</android.support.design.widget.CoordinatorLayout>
```

上面这段代码会把按钮放在右下角位置的其他内容之上，互不干扰。

33.3.3 snackbar

Snackbar比floating action button复杂一些。它位于屏幕底部，是一种不怎么需要用户交互的控件（如图33-12所示）。

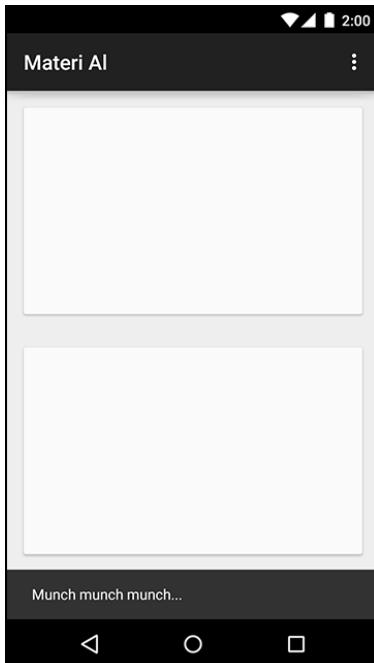


图33-12 一个Snackbar控件

Snackbar控件的动画效果是从屏幕底部向上弹出。短暂停留一会，或是屏幕上有了其他交互操作，它就会自动退回。Snackbar的作用和Toast类似。不同的是，Snackbar本身就是应用界面的一部分；而Toast会出现在应用界面之上，一动不动，即使用户已导航至其他页面。此外，为方便用户响应和操作，Snackbar上还允许放置按钮。

和floating action button一样，Snackbar也是在Android设计支持库中实现的。

与Toast类似，创建和显示Snackbar的方式如代码清单33-11所示。

代码清单33-11 创建并显示Snackbar

```
Snackbar.make(container, R.string.munch, Snackbar.LENGTH_SHORT).show();
```

创建Snackbar需要传入放置它的视图、要显示的文字以及它暂留的时间。最后，调用show()方法展示它。

为方便用户执行某些操作，可在Snackbar控件的右端配置操作选项。比如，配置一个撤销删除按钮。这样，就算用户不小心点了crime记录删除按钮，也可以立即补救。

33.4 深入学习 material design

本章，我们介绍了一大堆新工具。工具不是摆设，首先要用起来，否则不论有再好的特效，用户体验不到也是白搭。所以，开发人员平时要多多留心，积极研究如何使用这些全新的动画工具和特效。

Android提供有material design规格说明书，可访问以下网页阅读：<http://www.google.com/design/spec/material-design/introduction.html>。它里面满是精彩的点子，可以帮助激发设计灵感。当然，也可以从Google Play下载精品应用，看看别人是怎么应用material design的，并问问自己：怎么将其用到我自己的应用里呢？经过这一番努力，你可能就会找到突破口，做出超越精品的好应用。

编后语 34

恭喜各位完成本书的学习！这很了不起，不是人人都能做到的。现在就去犒赏一下自己吧！
总之，千辛万苦终于有了回报：你已经是一名合格的Android开发者了。

34.1 终极挑战

最后，请再接受一个挑战：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。各位应该找寻最适合自己的路。

前进的方向在哪儿？这里，我们有一些建议。

□ 编写代码

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，利用一切机会编写代码。

□ 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。开发时，如遇到问题，记得经常查阅文档，或阅读有关更高级开发主题的书籍。另外，也可收看YouTube的Android开发者频道，或收听Google的Android开发者播客（Android Developers Backstage podcast）。

□ 参与技术交流

参与本地技术交流大会，结识那些乐于助人的开发者。参与Android开发者大会，与其他Android开发人员（包括我们）会面和交流。另外，还可以关注一些活跃在Twitter和Google Plus上的开发高手。

□ 探索开源社区

从<http://www.github.com>可以看出，Android开发已呈爆发趋势。找到很酷的共享库后，也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。另外，也可以订阅Android每周邮件列表，及时跟踪了解Android开发社区新动向（<http://androidweekly.net/>）。

34.2 关于我们

来Twitter找我们吧！Bill、Chris、Brian和Kristin的帐号分别是@billjings、@cstew、@lyricsboy和@kristinmars。

喜欢本书的话，也可以访问<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，我们也可以提供合作开发。欲详细了解，请访问网站<http://www.bignerdranch.com>。

34.3 致谢

正是有大家这样的读者，我们的工作才显得格外有意义。感谢所有购买并阅读本书的读者！

版 权 声 明

Authorized translation from the English language edition, entitled *Android Programming: The Big Nerd Ranch Guide, Second Edition* by Bill Phillips, Chris Stewart, Brian Hardy, Kristin Marsicano, published by The Big Nerd Ranch (Aaron Hillegass). Copyright © 2015 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 © 2016 by 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。

权威——源自大名鼎鼎的Big Nerd Ranch训练营培训讲义，该训练营已经为微软、谷歌、Facebook等行业巨头培养了众多专业人才。

全面——涵盖Android开发所有必备理论概念和技术知识点。

实用——8个Android应用开发实战项目，传授最直接的开发经验。

易懂——以循序渐进的方式精心编排章节，一步一步写出Android应用。



对第1版的赞誉

“对我们来说，这是一本非常全面的培训教材，它已使我们公司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

“不管你是刚刚迈进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。其完整的内容体系、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

“整本书的内容编排非常人性化！每个例程都从一个简单的activity开始，一步一步地往里面添加新的功能，每一步都讲解得细致入微，然后在读者的面前，慢慢变得强大起来。可以说，每个例程都是从开发者的角度开始，遵循一套科学的开发流程，最后变成一个功能强大的应用程序。与此同时，需要掌握的开发技巧也就融合进去了。”

——亚马逊读者评论

“知识点讲解得很全面，通过实际示例练习逐步上手。章节末尾的挑战练习和深入学习非常到位，遇到自己暂时无法解决的问题也可以去官网论坛上同其他读者交流讨论。读完此书后，我的Android应用开发技能有了很大提高。”

——亚马逊读者评论

“每一章都是实打实的例子，由小到大、由浅入深，顺序安排得很贴心，让人学得很舒服，是迄今为止看到过的最好的教程。难能可贵的一点是，每个例子的代码规范都很棒。”

——亚马逊读者评论



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 978-7-115-42246-0



ISBN 978-7-115-42246-0

定价: 109.00元