

移动开发经典丛书

Android 开发范例 代码大全 (第 2 版)

[美] Dave Smith 著
Jeff Friesen
赵 凯 陶 冶 译

清华大学出版社

北 京

Dave Smith, Jeff Friesen

Android Recipes: A Problem-Solution Approach, Second Edition

EISBN: 978-1-4302-4614-5

Original English language edition published by Apress Media. Copyright © 2012 by Apress Media.

Simplified Chinese-Language edition copyright © 2014 by Tsinghua University Press. All rights reserved.

本书中文简体字版由 Apress 出版公司授权清华大学出版社出版。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。

北京市版权局著作权合同登记号 图字：01-2013-6037

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

图书在版编目(CIP)数据

Android 开发范例代码大全：第2版 / (美) 史密斯(Smith,D.) , (美) 弗里森 (Friesen,J.) 著；赵凯，陶冶译. —北京：清华大学出版社，2014

(移动开发经典丛书)

书名原文: Android Recipes: A Problem-Solution Approach, Second Edition

ISBN 978-7-302-35483-3

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

中国版本图书馆 CIP 数据核字(2014)第 031199 号

责任编辑：王 军 杨信明

装帧设计：牛静敏

责任校对：成凤进

责任印制：

出版发行：清华大学出版社

网 址：http://www.tup.com.cn, http://www.wqbook.com

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175 邮 购：010-62786544

投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈：010-62772015, zhiliang@tup.tsinghua.edu.cn

装 订 者：

经 销：全国新华书店

开 本：185mm×260mm 印 张：46 字 数：1119 千字

版 次：2014 年 4 月第 1 版 印 次：2014 年 4 月第 1 次印刷

印 数：1~3000

定 价：98.00 元

产品编号：

译者序

目前，Android 在移动互联网上的火爆程度大家已经有目共睹，学习 Android 开发的人也越来越多，众多的 Android 书籍也让初学者无从下手。本书则从一个个实际范例出发，逐个展示 Android 中的技术细节，一层层地揭开 Android 神秘的面纱。

如果你是一位刚刚开始学习 Android 的开发者，建议先简单地了解一下 Android 中的一些基本概念和常用控件，然后再利用本书中的范例进一步学习各个控件的特性。

如果你已经开发 Android 有段时间了，则可以将本书当成一本工具书放在电脑旁边，在项目有需要时适当地看一下书中范例，或许会获益良多。

本书涵盖的内容相当丰富，从基本的控件使用、数据持久化到相对高级的 NDK、Renderscript 开发都有比较详细的介绍，基本可以满足大多数开发者的需求。如果结合 Android API Demo 中的范例一起使用，效果则更好。如果你想成为真正的高手，建议多看看 Android 的源代码以及知名的开源项目代码，学习一下大师们的思路与技巧。

如果你能将每个范例都细看一下，基本上就可以了解到 Android 体系的大概，在实际进行 Android 编程工作的时候，对于应用程序功能的实现和优化可以快速和准确地定位到相应的技术，这样可以节省很多时间、少走很多弯路。

能翻译这本书确实是我的荣幸。虽然自己一直从事 Android 开发，但还是把每个范例都在自己的电脑上运行了一遍，而很多范例的细节都经过了再三的斟酌和推敲，在翻译过程中，我如履薄冰，生怕有违作者原意。

最后要感谢我的家人的支持，感谢朋友(陶老师)和同事(新卓、相亭、王旭、旭杰)的无私帮助，许多技术细节上的问题都是在他们的指导下完成的。由于时间比较紧张，加之译者水平有限，若译文有不当之处，敬请读者批评指正。

赵凯

序

Dave Smith 和 Jeff Friesen 为撰写本书付出了大量的心血。我很早就在移动开发社区里认识了 Dave，我知道这本书的每一章都凝聚着他的心血，每一个范例都经过了反复的讨论。为什么我知道这些？因为我有幸每天跟 Dave 在一起工作，在我们推出 Android 软件的过程中，当我们遇到各种困难时他为我们提供了系统化、标准化和严谨的解决方案。

随着短时间内 Android 设备的爆炸式发展，我们迎来了一个改变移动计算未来发展的难得机遇。Android 支持手机、平板电脑、工业设备以及未来我们不知道的一些设备。这一系列的设备都运行在同样的平台上，使得软件开发人员的“一次编写，到处运行”的梦想成为现实。本书中，Dave 和 Jeff 通过他们编写 Android 应用程序时遇到的一个个实例，引导读者开始 Android 开发的学习之旅。现在，把握住这个机会，为用户打造具有完美使用体验的应用程序。当你的应用程序上线时，这些移动设备就成了你的舞台。大量新的移动设备的涌现将会伴随着大量应用程序的出现，不过其中可能大部分都是垃圾。这时候就需要你站在用户的角度，解决他们遇到的问题，创造出引以为傲的佳作。关注细节才能得到用户的青睐——记住，“Real Artists Ship”（乔布斯的名言，即只有真正的艺术家才会让产品上线）。

—Ben Reubenstein (@benr75)
benr@xcellentcreations.com
Xcellent Creations, Inc.

作者简介



Dave Smith, 2006年毕业于科罗拉多矿业学院并获得电气工程和计算机科学学位，一直从事嵌入式平台软件和硬件的开发。目前，Dave全身心地投入到移动开发领域，现在是 Denver.CO 的顾问。从 2009 年开始，Dave 就从事 Android 平台各个版本上的开发，包括使用 SDK 编写用户应用程序以及构建和定制 Android 源代码。他本人比较喜欢的 Android 项目是那种可以在用户设备中集成定制硬件以及可以为定制的嵌入式平台包含构建 Android 的项目。此外，Dave 会定期更新开发博客 (blog.wiresareobsolete.com) 和 Twitter (@devunwired)。



Jeff Friesen, 自由职业者，主要从事 Java 软件开发。除了本书外，Jeff 还为 *JavaWorld*(www.javaworld.com)、*informIT*(www.informit.com)、*java.net*、*DevSource*(www.devsource.com)、*SitePoint*(www.sitepoint.com) 和 *BuildMobile*(www.buildmobile.com) 等网站撰写了很多 Java 和其他方面的技术文章。可以到 Jeff 的网站上联系他，网址是 tutortutor.ca。

技术评审人员简介



Chád Darby 是 Java 开发领域的作者、讲师和发言人。作为 Java 应用程序和架构方面公认的权威人士，他在全球软件发展会议上提出了要专注于技术领域。在他 15 年的专业软件架构师生涯中，他曾效力于 Blue Cross/BlueShield、Merck、Boeing、Northrop Grumman 以及一些刚成立的公司。

Chád 参与了很多 Java 书籍的编写，包括 *Professional Java E-Commerce* (Wrox 出版社)、*Beginning Java Networking* (Wrox 出版社) 和 *XML and Web Services Unleashed* (Sams 出版)。Chád 拥有 Sun Microsystems 和 IBM 的 Java 认证并拥有卡耐基梅隆大学计算机科学专业的学士学位。可以访问 Chád 的博客 www.luv2code.com 以及在他的 Twitter @darbyluvs2code 上关注他。

致 谢

首先，我要感谢我的妻子 Lorie，感谢她在我撰写和构建本书所涉及的各种素材时给予我的支持和耐心。其次，感谢本书的另一位作者 Jeff Friensen，他对 Android 开发的新想法和新思路给本书带来了许多新意，让本书精彩绝伦。还要感谢我的好友和同事 Ben Reubenstein 在百忙中抽出时间为本书撰写序言，并将我引荐给 Apress 的编辑团队。另外，我要诚挚地感谢 Apress 的编辑团队。最后，我还要诚挚地感谢 Apress 给我和 Jeff 安排的支持团队：Steve Anglin、Jill Balzano、Tom Welsh、Chád Darby 和其他所有的人。没有他们所投入的时间和精力，本书就不可能顺利付梓。

—Dave Smith

我要感谢 Steve Anglin 邀请我撰写此书，感谢 Corbin Collins 在本书的各个方面给予我的指导，感谢 Tom Welsh 在我负责撰写的几章中给予的帮助，还要感谢 Paul Connolly 细心地找出了我书稿中的各种不足。最后还要感谢本书的另一位作者 Dave Smith 对本书所做的卓越贡献。

—Jeff Friesen

前言

欢迎阅读《Android 开发范例代码大全(第2版)》!

如果你正在阅读本书,那么移动设备给软件开发人员和用户带来的无限机遇就不用我在此赘述了。近年来,Android 已经成为最主要的移动平台之一。对于开发人员而言,必须要了解 Android 才能确保自己跟得上市场的变化,从而把握各种潜在的机会。但是任何新平台在常见需求的开发和常见问题的解决方案上都会有不确定性。

我们撰写本书旨在帮助开发人员解决实际开发中的问题,通过直观的例子告诉读者如何编写 Android 平台上的应用程序。本书不会很深入地介绍 Android SDK、NDK 或是其他工具。我们不会让隐藏其中的各种琐碎细节和高深理论打击读者的积极性。但这并不意味着这些细节没意思或是不重要。读者应该研究这些细节,以避免在开发中犯下错误。但在解决迫在眉睫的问题时,这些东西通常只会让人分心。

本书不会讲解 Java 编程,也不会介绍如何构建 Android 应用程序。本书略去了很多基础知识(例如,如何使用 TextView 显示文本),因为我们觉得这些知识在学过之后就不会遗忘。相反,本书会帮助开发人员解决很多实际开发中经常要完成的任务,而这些复杂的任务不是寥寥几行代码就能完成的,自然也很难记住。

读者可以把本书当作一本可供随时查询的参考书、一本资源丰富的示例手册,随时都可以从中找到有助于高效完成工作的实用建议。

本书主要内容

尽管本书并不是针对新手的 Android 开发教程,但我们还是在第 1 章中概述了理解全书所需的 Android 基础知识。其中包括了 fragment 和资源的相关知识。第 1 章还介绍了一个很重要的应用程序 Univerter,展示了如何准备环境从而开发 Univerter 和其他 Android 应用程序。具体来说,就是如何安装 Android SDK、Eclipse、ADT 插件以及如何通过它们构建 Univerter。

随着 Android 开发经验的增长,为了节约时间,肯定要尽力避免重新发明轮子。开发人员应该创建和使用自己的可重用代码库,或者使用其他人开发的库。第 7 章会说明如何创建和使用自定义的 Jar 形式的代码库和 Android 库项目。除了创建自己的库,还介绍了两个 Android SDK 以外的 Java 库供应用程序使用。同样,将学习使用 Google 的支持库以及 GridLayout 类。

如果想开发成功的应用程序，性能问题是不可忽视的。大部分时候，这都不是问题，因为 Android(从 2.2 版开始)的 Dalvik 虚拟机有一个 Just-In-Time 的编译器，能将 Dalvik 字节码编译成设备的本地代码。如果这还不够，还可以利用 Android 的 NDK 进一步提升性能。第 8 章详述了 NDK，并用一个 OpenGL 示例演示了它的用途。

NDK 是一种比较复杂的技术，需要使用冗长的 Java Native Interface (JNI)，当应用程序过多地使用 JNI 调用时会影响到性能(以及应用程序本地部分的可移植性)。同样，当想要使用多个 CPU 内核时也需要做很多工作。幸运的是，Google 通过引入 Renderscript 已经消除了这种冗长编码并简化了多核执行任务，并实现了可移植性。第 8 章介绍了 Renderscript 并演示了如何使用它的计算引擎(并自动使用 CPU 的多核)来处理图片。

在其他几章中，我们会深入讲解如何用 Android SDK 解决各种实际问题。你将学习如何高效地创建能运行在各种设备上的用户界面。你将会成为整合各种硬件(收音机、传感器和摄像头)的专家，正是这些硬件让移动设备成为一个独具特色的平台。我们甚至还会讨论如何自行定制这个系统，集成 Google 提供的各种服务和应用程序，并兼容各个设备制造商的产品。以此为目标，我们还会推荐一些由 Google 和社区开发的工具，用于简化应用程序的开发和测试。

你对脚本语言(例如 Python 或 Ruby)感兴趣吗？如果感兴趣的话，你应该读一读附录 A，其中涵盖了 Scripting Layer for Android。这个特别的应用程序可以支持在 Android 上安装脚本语言解释器，在设备上编写脚本并运行，以提高开发速度。

为了快速了解 Android 众多工具的详细使用方法，附录 B 提供了各个支持工具的概述。其中，你会了解 Android 4.1 的 systrace 工具为什么不能运行在 Android 模拟器上。

在创建应用程序时，需要确保应用程序的性能好、响应速度快、且能与系统无缝衔接。低能耗、响应快、不会弹出 Application Not Responding (ANR，应用程序没有响应)窗口，且跟整个系统无缝衔接的应用程序才能让用户满意。此外，在将应用程序发布到 Google 的 Google Play 时，不能让不兼容的设备看到应用程序。应该要求 Google Play 过滤掉那些设备不兼容的用户，使之无法下载(甚至无法看到)你的应用程序。本书的附录 C 会指导你创建高性能、响应快而且与系统无缝衔接的应用程序，以及利用过滤功能只允许设备兼容的用户(从 Google Play)下载该应用程序。

第 1 章介绍了 Univerter 应用程序。本书最后的附录 D 会让你更加详细地了解 Univerter 的架构。

注意 API 级别

在本书中，读者会看到绝大部分的解决方案都有相应的最低 API 级别要求。本书中的大部分解决方案都只需要 API Level 1，换言之就是这些代码能在 Android 1.0 以上的任何设备上运行。但是，有些地方也用到了较新版本中引入的 API。注意各个范例的 API 级别，确保代码与应用程序要支持的 Android 版本相匹配。

目 录

第 1 章	Android 入门	1			
1.1	Android 简介	1			
1.2	Android 的发展史	2			
1.3	Android 架构	4			
1.4	应用程序架构	7			
1.4.1	组件	7			
1.4.2	资源	25			
1.4.3	Manifest 文件	33			
1.4.4	应用程序包	36			
1.4.5	安装 Android SDK	36			
1.4.6	安装 Android 平台	39			
1.4.7	创建 Android 虚拟设备	42			
1.4.8	启动 AVD	44			
1.4.9	Univerter 简介	48			
1.4.10	创建 Univerter	50			
1.4.11	安装和运行 Univerter	52			
1.4.12	准备 Univerter 在 Google Play 上发布	55			
1.4.13	移植到 Eclipse	60			
1.4.14	用 Eclipse 创建和运行 Univerter	63			
1.5	小结	66			
第 2 章	用户界面范例	67			
2.1	自定义窗口	67			
2.1.1	问题	67			
2.1.2	解决方案	67			
2.1.3	实现机制	67			
2.2	创建并显示视图	77			
2.2.1	问题	77			
2.2.2	解决方案	77			
2.2.3	实现机制	78			
2.3	监控单击动作	79			
2.3.1	问题	79			
2.3.2	解决方案	80			
2.3.3	实现机制	80			
2.4	适用于多种屏幕分辨率的 图形资源	81			
2.4.1	问题	81			
2.4.2	解决方案	81			
2.4.3	实现机制	82			
2.5	锁定 Activity 方向	83			
2.5.1	问题	83			
2.5.2	解决方案	83			
2.5.3	实现机制	83			
2.6	动态方向锁定	84			
2.6.1	问题	84			
2.6.2	解决方案	84			
2.6.3	实现机制	84			
2.7	手动处理旋转	86			
2.7.1	问题	86			
2.7.2	解决方案	86			
2.7.3	实现机制	87			
2.8	创建弹出菜单动作	88			
2.8.1	问题	88			
2.8.2	解决方案	88			
2.8.3	实现机制	88			
2.9	显示一个用户对话框	93			
2.9.1	问题	93			
2.9.2	解决方案	93			
2.9.3	实现机制	94			
2.10	自定义选项菜单	98			

2.10.1 问题	98	2.20 创建持久的对话框	134
2.10.2 解决方案	98	2.20.1 问题	134
2.10.3 实现机制	98	2.20.2 解决方案	134
2.11 自定义返回按键	101	2.20.3 实现机制	134
2.11.1 问题	101	2.21 实现针对具体场景的布局	136
2.11.2 解决方案	101	2.21.1 问题	136
2.11.3 实现机制	101	2.21.2 解决方案	136
2.12 模拟 Home 按键	104	2.21.3 实现机制	136
2.12.1 问题	104	2.22 自定义键盘动作	143
2.12.2 解决方案	104	2.22.1 问题	143
2.12.3 实现机制	104	2.22.2 解决方案	144
2.13 监控 TextView 的变动	105	2.22.3 实现机制	144
2.13.1 问题	105	2.23 隐藏软键盘	146
2.13.2 解决方案	105	2.23.1 问题	146
2.13.3 实现机制	105	2.23.2 解决方案	146
2.14 自动滚动的 TextView	107	2.23.3 实现机制	147
2.14.1 问题	107	2.24 自定义 AdapterView 的 空视图	147
2.14.2 解决方案	108	2.24.1 问题	147
2.14.3 实现机制	108	2.24.2 解决方案	147
2.15 动画视图	109	2.24.3 实现机制	147
2.15.1 问题	109	2.25 自定义 ListView 行	149
2.15.2 解决方案	109	2.25.1 问题	149
2.15.3 实现机制	109	2.25.2 解决方案	149
2.16 布局变化时的动画	119	2.25.3 实现机制	149
2.16.1 问题	119	2.26 制作 ListView 的节头部	153
2.16.2 解决方案	119	2.26.1 问题	153
2.16.3 实现机制	120	2.26.2 解决方案	153
2.17 用 Drawable 做背景	122	2.26.3 实现机制	153
2.17.1 问题	122	2.27 创建组合控件	156
2.17.2 解决方案	123	2.27.1 问题	156
2.17.3 实现机制	123	2.27.2 解决方案	156
2.18 创建自定义状态的 Drawable	128	2.27.3 实现机制	157
2.18.1 问题	128	2.28 处理复杂的单击事件	160
2.18.2 解决方案	128	2.28.1 问题	160
2.18.3 实现机制	128	2.28.2 解决方案	160
2.19 将遮罩应用到图片	130	2.28.3 实现机制	161
2.19.1 问题	130	2.29 转发触摸事件	177
2.19.2 解决方案	130	2.29.1 问题	177
2.19.3 实现机制	130	2.29.2 解决方案	177

2.29.3 实现机制	177
2.30 创建拖放视图	181
2.30.1 问题	181
2.30.2 解决方案	181
2.30.3 实现机制	182
2.31 自定义过渡动画	188
2.31.1 问题	188
2.31.2 解决方案	188
2.31.3 实现机制	189
2.32 创建视图变换	198
2.32.1 问题	198
2.32.2 解决方案	198
2.32.3 实现机制	198
2.33 视图之间滑动	204
2.33.1 问题	204
2.33.2 解决方案	204
2.33.3 实现机制	204
2.34 创建模块化接口	214
2.34.1 问题	214
2.34.2 解决方案	214
2.34.3 实现机制	214
2.35 高性能绘制	223
2.35.1 问题	223
2.35.2 解决方案	224
2.35.3 实现机制	224
2.36 实用工具推荐: Hierarchy Viewer 和 Lint	234
2.37 Hierarchy Viewer	234
2.38 浏览 View Hierarchy 窗口	236
2.39 Tree View 中的单个视图	238
2.40 使用 View Hierarchy 进行调试	238
2.41 浏览 Pixel Perfect 窗口	239
2.42 使用 Pixel Perfect Overlays	241
2.43 Lint	241
2.44 运行 Lint	242
2.45 小结	245

第 3 章 通信和联网	247
3.1 显示 Web 信息	247
3.1.1 问题	247
3.1.2 解决方案	247
3.1.3 实现机制	247
3.2 拦截 WebView 事件	251
3.2.1 问题	251
3.2.2 解决方案	251
3.2.3 实现机制	251
3.3 访问带 JavaScript 的 WebView	253
3.3.1 问题	253
3.3.2 解决方案	253
3.3.3 实现机制	253
3.4 下载一个图片文件	255
3.4.1 问题	255
3.4.2 解决方案	256
3.4.3 实现机制	256
3.5 完全在后台下载	259
3.5.1 问题	259
3.5.2 解决方案	259
3.5.3 实现机制	259
3.6 访问 REST API	262
3.6.1 问题	262
3.6.2 解决方案	262
3.6.3 实现机制	263
3.7 解析 JSON	286
3.7.1 问题	286
3.7.2 解决方案	286
3.7.3 实现机制	286
3.8 解析 XML	289
3.8.1 问题	289
3.8.2 解决方案	289
3.8.3 实现机制	289
3.9 接收短信	299
3.9.1 问题	299
3.9.2 解决方案	299
3.9.3 实现机制	299
3.10 发送短信	300
3.10.1 问题	300

3.10.2	解决方案	301	4.5.1	问题	349
3.10.3	实现机制	301	4.5.2	解决方案	349
3.11	蓝牙通信	303	4.5.3	实现机制	349
3.11.1	问题	303	4.6	录制音频	356
3.11.2	解决方案	303	4.6.1	问题	356
3.11.3	实现机制	303	4.6.2	解决方案	356
3.12	查询网络连接状态	312	4.6.3	实现机制	356
3.12.1	问题	312	4.7	自定义视频采集	358
3.12.2	解决方案	312	4.7.1	问题	358
3.12.3	实现机制	312	4.7.2	解决方案	358
3.13	使用 NFC 传输数据	314	4.7.3	实现机制	358
3.13.1	问题	314	输出格式方向	362	
3.13.2	解决方案	314	4.8	添加语音识别	362
3.13.3	实现机制	314	4.8.1	问题	362
3.14	USB 连接	321	4.8.2	解决方案	362
3.14.1	问题	321	4.8.3	实现机制	363
3.14.2	解决方案	321	4.9	播放音频/视频	365
3.14.3	实现机制	322	4.9.1	问题	365
3.15	小结	330	4.9.2	解决方案	365
			4.9.3	实现机制	365
第 4 章	实现设备硬件交互与媒体交互	331	4.10	播放音效	373
4.1	整合设备位置	331	4.10.1	问题	373
4.1.1	问题	331	4.10.2	解决方案	373
4.1.2	解决方案	331	4.10.3	实现机制	373
4.1.3	实现机制	332	4.11	创建倾斜监控器	376
4.2	地图位置	335	4.11.1	问题	376
4.2.1	问题	335	4.11.2	解决方案	376
4.2.2	解决方案	335	4.11.3	实现机制	376
4.2.3	实现机制	336	4.12	监控罗盘的方向	379
4.3	在地图上标记位置	339	4.12.1	问题	379
4.3.1	问题	339	4.12.2	解决方案	379
4.3.2	解决方案	339	4.12.3	实现机制	380
4.3.3	实现机制	339	4.13	在媒体内容中获取元数据	383
4.4	拍摄照片和视频	344	4.13.1	问题	383
4.4.1	问题	344	4.13.2	解决方案	383
4.4.2	解决方案	344	4.13.3	实现机制	383
4.4.3	实现机制	344	4.14	实用工具推荐:	
4.5	自定义摄像头覆盖层	349	Sensor Simulator	386	
			4.15	获得 Sensor Simulator	387

4.16 启动 Sensor Simulator Settings 和 Sensor Simulator	387	5.9 分享 SharedPreferences	430
4.17 在自己的应用程序中访问 Sensor Simulator	391	5.9.1 问题	430
4.18 小结	392	5.9.2 解决方案	430
第 5 章 数据持久化	393	5.9.3 实现机制	431
5.1 制作设置界面	393	5.10 分享其他数据	440
5.1.1 问题	393	5.10.1 问题	440
5.1.2 解决方案	393	5.10.2 解决方案	440
5.1.3 实现机制	393	5.10.3 实现机制	440
5.2 简单数据存储	398	5.11 实用工具推荐: SQLite3	446
5.2.1 问题	398	5.12 Univerter 和 SQLite3	448
5.2.2 解决方案	399	5.12.1 创建数据库	450
5.2.3 实现机制	399	5.12.2 扩展 Category 和 Conversion 类	451
5.3 读写文件	403	5.12.3 DBHelper 类简介	453
5.3.1 问题	403	5.12.4 扩展 Univerter 类	457
5.3.2 解决方案	403	5.12.5 运行改进后的 Univerter 应用程序	458
5.3.3 实现机制	404	5.13 小结	459
5.4 以资源的形式使用文件	409	第 6 章 与系统交互	461
5.4.1 问题	409	6.1 后台通知	461
5.4.2 解决方案	409	6.1.1 问题	461
5.4.3 实现机制	409	6.1.2 解决方案	461
5.5 管理数据库	412	6.1.3 实现机制	461
5.5.1 问题	412	6.2 创建定时和周期任务	469
5.5.2 解决方案	412	6.2.1 问题	469
5.5.3 实现机制	412	6.2.2 解决方案	469
5.6 查询数据库	417	6.2.3 实现机制	469
5.6.1 问题	417	6.3 定时执行周期任务	470
5.6.2 解决方案	417	6.3.1 问题	470
5.6.3 实现机制	418	6.3.2 解决方案	471
5.7 备份数据	419	6.3.3 实现机制	471
5.7.1 问题	419	6.4 创建粘性操作	474
5.7.2 解决方案	419	6.4.1 问题	474
5.7.3 实现机制	419	6.4.2 解决方案	474
5.8 分享数据库	423	6.4.3 实现机制	475
5.8.1 问题	423	6.5 长时间运行的后台操作	479
5.8.2 解决方案	424	6.5.1 问题	479
5.8.3 实现机制	424	6.5.2 解决方案	479

6.5.3 实现机制	480	6.15.1 问题	522
6.6 启动其他应用程序	485	6.15.2 解决方案	522
6.6.1 问题	485	6.15.3 实现机制	522
6.6.2 解决方案	485	6.16 实现 APPWidget	529
6.6.3 实现机制	486	6.16.1 问题	529
6.7 启动系统应用程序	489	6.16.2 解决方案	529
6.7.1 问题	489	6.16.3 实现机制	530
6.7.2 解决方案	489	6.17 小结	550
6.7.3 实现机制	489		
6.8 让其他应用程序启动你的 应用程序	493	第 7 章 使用库	551
6.8.1 问题	493	7.1 创建 Java 库 JAR	551
6.8.2 解决方案	494	7.1.1 问题	551
6.8.3 实现机制	494	7.1.2 解决方案	551
6.9 与联系人交互	496	7.1.3 实现机制	552
6.9.1 问题	496	7.2 使用 Java 库 JAR	554
6.9.2 解决方案	496	7.2.1 问题	554
6.9.3 实现机制	496	7.2.2 解决方案	554
6.10 设备媒体文件选择器	503	7.2.3 实现机制	554
6.10.1 问题	503	7.3 创建 Android 库项目	557
6.10.2 解决方案	503	7.3.1 问题	557
6.10.3 实现机制	503	7.3.2 解决方案	557
6.11 保存到 MediaStore	505	7.3.3 实现机制	557
6.11.1 问题	505	7.4 使用 Android 库项目	561
6.11.2 解决方案	505	7.4.1 问题	561
6.11.3 实现机制	505	7.4.2 解决方案	561
6.12 与日历的交互	508	7.4.3 实现机制	561
6.12.1 问题	508	7.5 绘图	565
6.12.2 解决方案	508	7.5.1 问题	565
6.12.3 实现机制	508	7.5.2 解决方案	565
6.13 执行日志代码	514	7.5.3 实现机制	565
6.13.1 问题	514	7.6 消息推送实战	577
6.13.2 解决方案	515	7.6.1 问题	577
6.13.3 实现机制	515	7.6.2 解决方案	577
6.14 创建后台工作线程	517	7.6.3 实现机制	578
6.14.1 问题	517	7.7 使用 Google 的支持包	585
6.14.2 解决方案	517	7.7.1 问题	585
6.14.3 实现机制	517	7.7.2 解决方案	585
6.15 自定义任务栈	522	7.7.3 实现机制	587
		7.8 小结	590

第 8 章 使用 Android NDK 和 Renderscript.....	591	附录 B Android 工具一览.....	647
8.1 Android NDK.....	591	B.1 SDK 工具.....	647
8.1.1 安装 NDK.....	592	B.1.1 android.....	647
8.1.2 浏览 NDK.....	595	B.1.2 apkbuilder.....	652
8.1.3 来自 NDK 的问候.....	596	B.1.3 ddms.....	652
8.1.4 NDK 示例.....	602	B.1.4 dmtracedump.....	652
8.2 发现本地 Activity.....	604	B.1.5 draw9patch.....	653
8.2.1 问题.....	604	B.1.6 emulator.....	653
8.2.2 解决方案.....	604	B.1.7 etc1tool.....	658
8.2.3 实现机制.....	605	B.1.8 hierarchyviewer.....	658
8.3 开发 Low-Level 本地 Activity.....	605	B.1.9 hprof-conv.....	659
8.3.1 问题.....	605	B.1.10 lint.....	659
8.3.2 解决方案.....	605	B.1.11 mkshcard.....	660
8.3.3 实现机制.....	607	B.1.12 monitor.....	661
8.4 开发 High-Level 的本地 Activity.....	615	B.1.13 monkeyrunner.....	661
8.4.1 问题.....	615	B.1.14 sqlite3.....	662
8.4.2 解决方案.....	615	B.1.15 systrace.....	663
8.4.3 实现机制.....	616	B.1.16 traceview.....	665
8.5 Renderscript.....	621	B.1.17 OpenGL ES 的 Tracer 工具.....	665
8.5.1 浏览 Renderscript 架构.....	622	B.1.18 zipalign.....	665
8.5.2 使用 Renderscript 对图像 进行灰度化处理.....	624	B.2 平台工具.....	666
8.6 了解更多关于 Renderscript 的 知识.....	631	B.2.1 aapt.....	666
8.6.1 问题.....	631	B.2.2 adb.....	667
8.6.2 解决方案.....	632	B.2.3 aidl.....	668
8.6.3 实现机制.....	632	B.2.4 dexdump.....	668
8.7 小结.....	635	B.2.5 dx.....	669
附录 A Android 的脚本层.....	637	B.2.6 fastboot.....	669
A.1 安装 SL4A.....	637	B.2.7 llvm-rs-cc.....	670
A.2 浏览 SL4A.....	638	附录 C 应用程序设计指南.....	673
A.2.1 添加 Shell 脚本.....	639	C.1 设计经过滤的应用程序.....	673
A.2.2 访问 Linux Shell.....	641	C.1.1 问题.....	673
A.3 安装 Python 解释器.....	641	C.1.2 解决方案.....	673
A.4 编写 Python 脚本.....	644	C.2 设计高性能的应用程序.....	675
		C.2.1 问题.....	675
		C.2.2 解决方案.....	675
		C.3 设计快速响应的应用程序.....	676
		C.3.1 问题.....	676

C.3.2 解决方案	677	D.2 浏览资源文件	701
C.4 设计无缝衔接的应用程序	677	D.2.1 应用程序启动器图标资源	702
C.4.1 问题	677	D.2.2 背景 Drawable 资源	702
C.4.2 解决方案	678	D.2.3 浏览主布局资源文件	703
C.5 设计安全的应用程序	679	D.2.4 浏览列表中每行的 布局资源	708
C.5.1 问题	679	D.2.5 浏览选项菜单的资源	709
C.5.2 解决方案	679	D.2.6 浏览 Help 对话框布局 资源	710
附录 D Univerter 的结构	681	D.2.7 浏览显示 Info 对话框的 布局资源	710
D.1 源代码	681	D.2.8 浏览颜色资源	712
D.1.1 Converter 接口	681	D.2.9 浏览字符串资源	712
D.1.2 Conversion 类	682	D.2.10 浏览样式资源	713
D.1.3 Category 类	683	D.3 浏览 Manifest 文件	714
D.1.4 Univerter 类	685		
D.1.5 Univerter 中的变量	686		
D.1.6 Univerter 的方法	688		

第 1 章

Android 入门

Android 炙手可热，许多人都在开发 Android 应用程序。或许你也希望能够开发相应的应用程序，但不知道如何着手。Google 网上发布的 *Android Developer's Guide*(<http://developer.android.com/index.html>)能够为用户提供所需的知识，但是面对该向导提供的海量信息，开发者很可能陷入无从下手的窘境。因此，本章提供了足够的理论来帮助你掌握 Android 的基础知识，这些理论是一些范例，它们可以指导你开发应用程序并将其发布到 Google Play(<https://play.google.com/store>)上。

1.1 Android 简介

Android Developer's Guide 最初将其定义为：Android 系统是一个软件栈，即能够为移动设备提供全功能解决方案的软件子系统集。该软件栈主要包括操作系统(修改版的 Linux 内核)、中间件和一些核心应用程序。中间件是用于连接底层操作系统和高层应用程序的软件，部分基于 Java 语言进行设计。而核心应用程序部分也采用 Java 语言来编写，例如 Web 浏览器(也称 Browser)以及联系人管理器(也称 Contacts)。

Android 系统具有如下特征：

- 可重用的应用程序框架和可替换程序组件(在本章的后面讨论)
- 蓝牙、EDGE、3G 和 WiFi 支持(依赖于硬件)
- 摄像头、GPS、罗盘和加速度计的支持(依赖于硬件)
- 为移动设备优化的 Dalvik 虚拟机
- GSM 电话支持(依赖于硬件)
- 集成基于开源 Webkit 引擎的浏览器
- 支持常见的音频、视频和静止图像的媒体格式(MPEG4、H.264、MP3、AAC、AMR、JPG、PNG 和 GIF)
- 使用自定义的 2D 图形库，基于 OpenGL ES 1.0、1.1 或 2.0 的 3D 图形规范(可选硬件加速)的图形优化

- 用于结构化数据存储的 SQLite

Android 丰富的开发环境(包括 Eclipse 集成开发环境中提供的设备模拟器和插件)虽然并不是 Android 设备软件栈的组成部分,但也被认为是 Android 系统的一个特征。

1.2 Android 的发展史

与你所想的恰恰相反,Android 并非源于 Google,Android 最初是由 Android 公司开发的,一家位于加利福尼亚帕罗奥图的小型创业公司。Google 在 2005 年夏天收购了该公司,并于 2007 年 11 月发布 Android SDK 的 beta 版。2008 年 9 月 23 日,Google 发布了 Android 1.0 版,其核心功能包括一个 Web 浏览器、支持摄像头、Google 搜索引擎等。表 1-1 给出了后续版本的特征(从 1.5 版开始,其命名规则改为以“甜点”作为系统版本代号进行命名)。

表 1-1 Android 各种版本

版本号	版本发布日期及变化情况
1.1	Google 在 2009 年 2 月 9 日发布了 SDK 1.1。其变化包括: 显示/隐藏免提拨号和保存消息中的附件
1.5(Cupcake, 杯式蛋糕) 基于 Linux 内核 2.6.27	Google 在 2009 年 4 月 30 日发布了 SDK 1.5。其变化包括: 拍摄及播放 MPEG-4 和 3GP 格式的视频,在主屏幕(其本身是一种特殊的应用程序,作为使用 Android 设备的起点)中植入了微型应用程序视图控件和动态的屏幕转换功能
1.6(Donut, 甜甜圈) 基于 Linux 内核 2.6.29	Google 在 2009 年 9 月 15 日发布了 SDK 1.6。其变化包括: 拓展了手势框架结构,增加了新的手势编译开发工具,集成了照相机/摄像机/Gallery 接口,支持 WVGA 屏幕分辨率并更新了搜索体验
2.0 / 2.1(Éclair, 泡芙) 基于 Linux 内核 2.6.29	Google 于 2009 年 10 月 26 日发布了 SDK 2.0。其变化包括: 动态壁纸,大量新的相机功能(包括 Flash 支持、数码变焦、场景模式、白平衡、色彩效果和宏焦点),提高了虚拟键盘打字的速度,具有从使用中的词汇和联系人的姓名中学习新词汇的字典,改进 Google Maps 3.1.2,支持蓝牙 2.1 Google 随后于 2009 年 12 月 3 日发布了 SDK update 2.0.1 版,并在 2010 年 1 月 12 日发布了 SDK update 2.1 版。2.0.1 版侧重于修改少量的应用程序接口、修复了漏洞和改变了部分的框架行为。2.1 版进一步修订了应用程序接口并进行了漏洞修复
2.2(Froyo, 冻酸奶) 基于 Linux 内核 2.6.32	Google 在 2009 年 5 月 20 日发布了 SDK 2.2。其变化包括: 将 Chrome V8 JavaScript 引擎集成到浏览器中,语音拨号和联系人共享蓝牙,支持 Adobe Flash,通过 JIT 编译进一步改进应用程序执行速度,并改进 USB 和 WiFi 热点功能 Google 随后于 2011 年 1 月 18 日发布了 SDK update 2.2.1 版,提供了错误修正、安全更新和性能改进。并于 2011 年 1 月 22 日发布了 SDK update 2.2.2 版,进行了少量的错误修正,包括 Nexus One 手机中的短信路由问题。Google 最终在 2011 年 11 月 21 日发布 SDK update 2.2.3 版,以及两个安全补丁

(续表)

版本号	版本发布日期及变化情况
2.3(Gingerbread, 姜饼) 基于 Linux 内核 2.6.35	<p>Google 在 2010 年 12 月 6 日发布了 SDK 2.3。其变化包括： 并行垃圾收集器，以提高应用程序的响应；支持陀螺仪和气压传感；支持 WebM / VP8 视频播放和 AAC 音频编码；支持近距离无线通信技术；支持增强的复制/粘贴功能，允许用户按键选词，并进行复制、粘贴操作</p> <p>Google 随后于 2011 年 2 月 9 日发布了 SDK update 2.3.3 版，提供改进和 API 修复。2011 年 4 月 28 日发布 SDK update 2.3.4 版，增加了通过 Google Talk 进行声音或视频聊天的功能</p> <p>2011 年 7 月 25 日发布了 SDK update 2.3.5 版，提供了系统增强功能，列表框滚动增加了阴影动画效果，提高电池的效率。2011 年 9 月 2 日发布了 SDK update 2.3.6 版，修复了语音搜索错误。2011 年 9 月 21 日发布了 SDK update 2.3.7 版，在 Nexus S 4G 手机中提供了对 Google Wallet 的支持</p>
3.0 (Honeycomb, 蜂巢) 基于 Linux 内核 2.6.36	<p>Google 于 2011 年 2 月 22 日发布 SDK 3.0，与以往不同，该版本侧重于对不同平板电脑的支持，例如摩托罗拉 Xoom 是第一款被支持的平板电脑产品(发布于 2011 年 2 月 24 日)。此外，除了改进用户界面，版本 3.0 还完善了多任务处理，支持多核处理器，支持硬件加速，提供了一个重新设计组件的 3D 桌面</p> <p>从 2011 年到 2012 年 2 月，Google 陆续发布了 SDK updates 3.1、3.2、3.2.1、3.2.2、3.2.4 和 3.2.6 版</p>
4.0(Ice Cream Sandwich, 冰淇淋三明治) 基于 Linux 内核 3.0.1	<p>Google 在 2011 年 10 月 19 日发布了 SDK 4.0.1。SDK 4.0.1 和 4.X 后续版本统一了 2.3.x 智能手机版和 3.x 平板电脑版，其特征包括 1080p 的视频记录和一个可定制的启动器</p> <p>Google 随后于 2011 年末和 2012 年 3 月发布了 SDK updates 4.0.2、4.0.3 版</p>
4.1(Jelly Bean, 果冻豆)	<p>Google 在 2012 年 6 月 27 日发布了 SDK 4.1 版。其功能包括垂直同步、定时、三重缓冲、自动调整大小应用程序组件、改进语音搜索、多声道音频和可扩展的通知。空中更新版(版本号 4.1.1)在 7 月发布。10 月初，Google 发布了 SDK 4.1.2 版，提供在 Nexus 7 中主屏幕旋转锁定支持，单指手势来展开/折叠的通知栏，错误修复和性能增强。后来在 10 月末，Google 发布 SDK 4.2 版，该版本支持球面全景照片，多用户账户(平板电脑)，设备闲置时激活“Daydream”屏保，通知电源管理，无线显示(Miracast 设备)的支持等</p>

1.3 Android 架构

Android 软件栈由顶部的应用程序层，位于中间层的中间件(包括应用程序框架、核心库、Android 运行库)，以及包括各种驱动程序的 Linux 核心层构成。图 1-1 显示了这个分层结构。



图 1-1 Android 框架层次的主要构成

鉴于用户关心的是应用程序层功能，Android 附带有各种有用的核心应用程序，其中包括 Browser、Contacts、Phone。所有的应用程序是用 Java 程序设计语言编写的，应用程序层位于 Android 框架结构的最顶层。

注意：

应用程序采用非标准 Java 语言来实现，将 Android 特定的 API 与 Java 5 API 和 Java 6 的一小部分(例如 java.io.file 类的方法：boolean setExecutable(boolean executable,boolean ownerOnly)进行了结合。因为 Android 不支持大多数 Java 6 和所有的 Java 7 API 方法，因此不能使用最新的 Java API 及其相应功能。例如，你不能使用 Java 7 资源释放 try-with-resources(带资源的 try 语句)语句声明，因为该功能需要使用 Java 7 的

java.lang.autocloseable 接口。

每一个 Android 版本(包括更新版)均被分配了一个 API Level(级别),该 Level 号是一个整型数值,用于唯一标识框架的 API 版本,该版本由相应 Android 平台来提供。例如,Android 4.1 版被分配的 API Level 为 16 而 Android 2.3.4 版的 API Level 为 10。具有较高 API Level 的 API 通常不能用于具有较低 API Level 的设备中(Google 支持库能够将更新的 API 应用到旧版本平台中,该部分将在第 7 章进行讨论)。例如,在仅支持 API Level 10(及更低)的设备中,不能使用 Level 16 的 API。API Level 常数在 android.os.build.version_codes 类中定义。在 *Android Developer's Guide* 中,查阅“Android API Levels”,可以了解更多关于 API Level 的相关知识(<http://developer.android.com/guide/topics/manifest/uses-sdkelement.html#ApiLevels>)。

在应用层下面是应用框架层,该层是一个高级组件块的集合,主要用于创建应用程序。应用框架预装在 Android 设备中,并由以下部分组成:

- **Activity 管理器:** 该组件提供了一个应用程序的生命周期并维护一个共享 Activity 栈,该栈可以在程序内部或不同应用程序之间进行导航。这两个主题将在后续章节进行讨论。
- **内容提供者:** 该类型组件用于封装数据(例如浏览器的书签),以确保该数据可以在不同应用程序之间实现共享。
- **定位管理器:** 该组件可以使一个 Android 设备知道它的物理位置。
- **通知管理器:** 该组件可以让应用程序在不打扰用户当前操作的情况下将重要的事件通知给用户(如消息事件等)。
- **包管理器:** 该组件可以让应用程序了解安装在当前设备上的其他软件包(应用程序包是在本章稍后讨论)。
- **资源管理器:** 该组件允许应用程序访问资源,该主题将在本章后面讨论。
- **电话管理器:** 该组件可以让应用程序了解设备上所具有的电话服务功能,此外还能处理打/接电话。
- **视图系统:** 该组件负责管理用户界面元素,并能生成面向用户界面的事件(该主题将在本章后面简述)。
- **窗口管理器:** 该组件将屏幕真实界面组织到窗口中,绘制窗口表面,并执行其他与窗口相关的工作。

应用框架组件需要借助 C/C++ 库来执行相关功能。开发者通过框架 API 可实现与如下库进行交互操作:

- **FreeType 库:** 本库支持位图和矢量字体渲染。
- **libc 函数库:** 该库是由源于 BSD 标准的 C 系统库实现,为针对基于 Linux 的嵌入式设备进行了修改。
- **网页核心库:** 该库提供了先进而快速的网络浏览器引擎,进一步促进了 Android 浏览器和嵌入式 Web 视图的发展。它基于 WebKit(<http://en.wikipedia.org/wiki/webkit>),也能用于 Google 浏览器和苹果公司的 Safari 浏览器。

- 媒体框架：基于 Packetvideo 推出的开源多媒体框架，支持回放和许多流行的音频和视频的记录格式，以及静态图像文件。支持的格式包括 MPEG4、H.264、MP3、AAC、AMR、JPEG 和 PNG。
- OpenGL/ES：这些 3D 图形库提供基于 OpenGL ES 1.0/1.1/2.0 API 的功能实现。使用硬件 3D 加速(如果可用)或包含(高度优化的)3D 软件光栅。
- SGL：该库提供了基本的 2D 图形引擎。
- SQLite 关系数据库：该库提供了一个强大的、轻量级的、面向所有程序的关系数据库引擎，Mozilla 的火狐浏览器和苹果公司的 iPhone 也采用该引擎进行持久化存储。
- SSL：该库提供了基于安全套接层的网络通信。
- Surface 管理器：该库管理显示子系统的访问和多个应用程序中无缝整合 2D 和 3D 的图形。

Android 运行时环境由核心库(Apache Harmony Java 第 5 版的子集)和 Dalvik 虚拟机组成。Dalvik 不是基于栈的，而是基于寄存器的非 Java 虚拟机。

注意：

Google 的 Dan Bornstein 创建并命名 Dalvik 虚拟机，该名字来源于他的祖先曾经居住过的冰岛上的一个小渔村。

每个 Android 应用程序默认在它自己的 Linux 进程中运行，该进程位于 Dalvik 虚拟机的一个实例中，设计 Dalvik 虚拟机的初衷是为了使设备可以有效地运行多个虚拟机。在效率方面，Dalvik 的优势主要是 Dalvik 执行的是基于 Dalvik 的可执行文件(简称 DEX)，该文件经过了优化，具备很小的内存占用率。

注意：

当应用程序的任何部分被执行时，Android 均会启动一个进程。当该应用程序不再需要或系统资源需要分配给其他应用程序时，Android 将关闭该进程。

也许你很困惑——Android 是如何使用非 Java 虚拟机来运行 Java 程序代码的。答案是，Dalvik 不运行 Java 代码，相反利用 DEX 格式工具，Android 可以将编译后的 Java 类文件转换为 DEX 格式，产生的代码则可以被 Dalvik 执行。

最后，Android 库和运行环境依赖于 Linux 内核(2.6.x 或 3.0.x)提供的底层核心服务，例如，线程、底层的内存管理、网络协议栈、进程管理和驱动程序模型。此外，内核还作为硬件和软件堆栈之间的抽象层。

Android 的安全模型

Android 的架构包括一个安全模型，该模型可以避免当前应用程序执行对其他应用程序、Linux 系统或用户有害的程序。安全模型基于标准 Linux 的特性(如用户和组 ID)，采用进程强制执行优先级策略，将进程放置于安全沙箱模型中。

在默认情况下，沙箱可防止应用程序读写用户的私有数据(如联系人或电子邮件)，读

写其他程序的文件、访问网络、保持设备唤醒、访问摄像机等。在访问网络或执行其他敏感的操作之前，应用程序必须首先获得许可。

Android 可采用多种方式来处理许可权限请求，一般通过证书方式或提示用户授予或撤销许可来自动允许或拒绝请求。应用程序所需的权限是在应用程序 manifest 文件中声明的(在本章的后面进行讨论)。对 Android 而言，当应用程序安装后，Android 会记录每个应用程序的许可权限，并且这些权限也是保持不变的。

1.4 应用程序架构

不同于桌面应用程序架构，Android 应用程序架构主要包括组件(采用 Intent 对象进行通信)、资源(通常在用户界面上下文中使用)、manifest(描述了应用程序组件等)和应用程序包(用于存储组件、资源和 manifest 内容)。

1.4.1 组件

应用程序由 Activity 组件、服务组件、广播接收器组件和内容提供器组件构成，上述组件运行在 Linux 进程中，并由 Android 负责管理：

- Activity 组件提供了当前用户界面的屏幕。
- 服务组件处理后台长时间执行的任务(例如播放音乐)，不负责提供用户界面。
- 广播接收器可从 Android 或其他组件中接收和响应广播。
- 内容提供器负责封装数据并确保数据的可用性。

每个组件都是在类中实现的，类也存储在 Java 包中，该包被称为应用程序包。从 Android SDK 的角度来看，每个类的源文件均存储在包目录层次结构中，该目录位于 src 目录下(在本章后面将介绍 Android SDK)。

应用程序并非使用上述所有组件。例如，一个应用程序可能仅包含 Activity 组件，而另一个应用程序要使用 Activity 组件和服务组件。

注意：

应用程序的 Activity、服务、广播接收器和/或内容提供器共用一套系统资源，如数据库、preference、文件系统和 Linux 进程。

Android 使用 Intent 对象与 Activity、服务和广播接收器进行通信。Intent 对象属于消息，一方面可用于描述动作的执行过程(如启动 Activity)，另一方面能够以广播方式通知外部事件的发生(例如，设备上的摄像机激活事件)。Activity、服务和广播接收器也可以使用 Intent 对象互相通信。

Intent 对象是 android.content.intent 类的实例。Intent 对象采用如下信息来描述消息：

- Action(动作)：动作使用一个字符串来命名将要执行的(例如广播 Intent)、已经发生的、正在报告的动作。动作使用 Intent 常数来描述其功能，例如 ACTION_CALL(初始化电话呼叫)、ACTION_EDIT(供用户编辑显示数据)和 ACTION_MAIN(作为第一个

Activity 启动)。用户也可自定义动作字符串,用于激活应用程序中的组件。这些字符串应该使用软件包作为前缀(例如, "com.example.project.GET_NEWSFEEDS")。

- **Category(类别):** 类别用字符串来标识,可对处理 Intent 对象组件的类别提供附加的信息。例如, CATEGORY_LAUNCHER 代表调用的 Activity 应该作为顶层的应用程序出现在设备的应用程序启动器中(应用程序启动器将在范例 1-4 中简述)。
- **Component Name (组件名称):** 采用完整的组件类名(即包名加类名)字符串来表示,用于 Intent 对象。组件名称是可选的。如果设置, Intent 对象被传递到指定类的实例中;如果没有设置, Android 使用 Intent 对象中的其他信息找到合适的目标。
- **Data(数据):** 操作数据的统一资源标识符(如联系人数据库中某个人的记录)。
- **附件信息(Extra):** 提供额外信息的键值对集合,应该传递给处理 Intent 对象的组件。例如,发送一封电子邮件动作,附加信息可能需要包含邮件的主题和正文等。
- **Flag(标识):** 二进制位值,用于指示 Android 如何启动 Activity(例如, Activity 应归属哪个 task, task 将在本章后面进行讨论)和在启动 Activity 后,如何处理该 Activity (例如,该 Activity 是否被认为是最近启动的 Activity)。在 Intent 类中,标识是由常量来表示的,例如, FLAG_ACTIVITY_NEW_TASK 用于指定当前 Activity 在该 Activity 的堆栈中,开始了一个全新的任务, Activity 堆栈的概念将本章后面讨论。
- **Type(类型):** Intent 数据的 MIME 类型。通常, Android 会根据数据推断其类型。如果指定了类型, Android 就不会自行推断了。

Intent 可以分为显式或隐式声明。显式 Intent 明确指定了组件的名称。但是,开发者一般不会知道其他应用的组件名称,所以显式 Intent 通常只用于在应用内部传递消息(例如一个 Activity 启动同一个应用中的另一个 Activity)。Android 将显式 Intent 传递给指定的目标类的实例。只有 Intent 对象中的组件名称指定的组件才能收到这种 Intent。

隐式 Intent 则不指定目标名称(组件名没有赋值)。隐式 Intent 通常用于启动其他应用中的组件。Android 会搜索最合适的组件(执行请求的单个 Activity 或 Service)或若干组件(响应广播内容的一组广播接收器)来处理隐式 Intent。在搜索过程中, Android 会将 Intent 对象中的内容跟 Intent 过滤器(Intent filter)和有可能接受 Intent 组件的 Manifest 信息进行比较。

过滤器将组件能处理的 Intent 类型公告出去,只确定该组件能处理的 Intent。过滤器让组件能收到其所公告的类型的隐式 Intent。如果组件没有 Intent 过滤器,那就只能接受显式 Intent。反之,有过滤器的组件则既能接收显式 Intent 也能接收隐式 Intent。Android 在过滤 Intent 时,会参考 Intent 对象的 Action、Category、Data 和 Type,而不会参考 Extra 和 Flag。

注意:

Android 大量使用 Intent 对象,这种方式允许用户很容易地使用自定义组件来取代现有组件。例如, Android 提供了发送电子邮件的 Intent。你的应用程序发送此 Intent 即可以激活标准邮件应用程序,或者也可以注册一个 Activity 来响应该 Intent 对象,从而使用该 Activity 取代标准的邮件应用程序。

这种面向组件的架构允许一个应用程序重用其他应用程序的组件,只要其他应用程序允许重用自身的组件。组件重用减少内存总占用率,这对于具有有限存储空间的设备尤其重要。

例如，你正在创建一个绘图程序，让用户从一个调色板中选择一种颜色，此时另一个应用程序中正好包含一个合适的颜色选取器并且该颜色选取器组件允许重用。在这种情况下，绘图应用程序可以调用其他应用程序的颜色选取器以实现让用户选择一种颜色，而不需要绘图程序自己设计颜色选择器。绘图应用程序不必包含其他应用程序的颜色选取器，甚至不需要与其他应用程序有什么链接操作。相反，在需要时直接启动该应用程序的颜色选择器组件就可以了。

注意：

当应用程序的任何组成部分(如前述的颜色选取器)需要执行时，Android 会启动一个进程，并实例化该组成部分的 Java 对象。这就是为什么 Android 的应用程序没有单一入口点(例如，没有 C 程序中的 `main()` 函数)的原因。相反，应用程序会根据需要使用实例化的组件并运行。

1. 剖析 Activity

Activity 本身是组件，代表能够与用户进行交互操作的界面屏幕。例如：Android 的 Contacts 应用程序包括一个用于录入新的联系人的 Activity；Phone 应用程序包含一个拨打电话号码的 Activity；而 Calculator 应用程序也包含一个用于执行基本的计算操作的 Activity(参见图 1-2)。

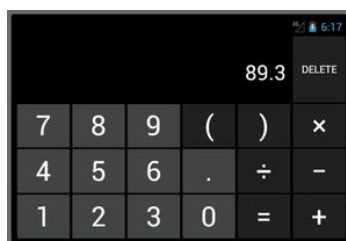


图 1-2 Android Calculator 应用程序的主 Activity 界面，用户可以进行基本的计算操作

尽管应用程序可以只包含一个 Activity 界面，但通常应用程序中会包含多个 Activity。例如，Calculator 应用程序还包含一个“advanced panel”的 Activity 界面，用于计算平方根、三角函数及其他复杂的数学运算。

Activity 是 `android.app.Activity` 类的子类，而 `android.app.Activity` 类是 `android.content.Context` 类的间接子类。

注意：

`Context` 是抽象类，它的方法让应用可以访问环境的全局信息(例如应用的资源和文件系统)，使得应用可以执行环境相关的操作，例如启动 Activity 和 Service、广播 Intent 以及打开私有文件等。

Activity 子类覆写了各种 Activity 生命周期回调方法，Android 会在 Activity 生命周期内调用这些方法。例如，程序清单 1-1 中的 `SimpleActivity` 类继承了 Activity，覆写了 `void onCreate(Bundle bundle)` 和 `void onDestroy()` 生命周期回调方法。

程序清单 1-1 Activity 框架

```

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

public class SimpleActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState); // 总是先调用父类的方法
        System.out.println("onCreate(Bundle) called");
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy(); //总是先调用父类的方法
        System.out.println("onDestroy() called");
    }
}

```

在程序清单 1-1 中, 覆写后的方法 `onCreate(Bundle)` 和 `onDestroy()` 必须首先调用父类中同名的方法, 其他生命周期回调方法, 例如 `void onStart()`、`void onRestart()`、`void onResume()`、`void onPause()` 和 `void onStop()` 方法也必须遵守这种模式。

- Activity 第一次创建时会调用 `onCreate(Bundle)`。该方法用于创建 Activity 的用户界面, 创建所需的后台线程, 并执行其他的全局初始化。如果能获得 Activity 以前的状态, 就可以将包含此状态的 `android.os.Bundle` 对象传给 `onCreate()`; 否则就传入一个空引用。在调用 `onCreate(Bundle)` 之后, Android 总会调用 `onStart()`。
- 在用户看到 Activity 之前会调用 `onStart()` 方法。当 Activity 进入前台时, Android 在调用 `onStart()` 之后就会调用 `onResume()`; 当 Activity 变成隐藏状态时, Android 就会在调用 `onStart()` 之后调用 `onStop()`。
- 当 Activity 停止后, 在其重新启动之前会调用 `onRestart()`。在调用 `onRestart()` 之后, Android 会调用 `onStart()`。
- 在 Activity 开始跟用户交互之前会调用 `onResume()`。此时, Activity 获得了焦点, 用户的输入会发送给该 Activity。当 Activity 必须暂停时, Android 就会在调用 `onResume()` 之后调用 `onPause()`。
- 当 Android 要恢复另一个 Activity 时会调用 `onPause()`。该方法一般用于保留未保存的修改, 停止可能会消耗处理器资源的动画等。它应该很快地完成工作, 因为只有等该方法返回时, 下一个 Activity 才能被重新激活。在调用了 `onPause()` 之后, 当 Activity 开始跟用户交互时, Android 就会调用 `onResume()`, 当 Activity 变成隐藏状态时调用 `onStop()`。许多 Activity 执行 `onPause()` 方法来提交数据的改变或者准备停止与用户的交互操作。
- 当 Activity 变为隐藏状态时会调用 `onStop()`。这种情况可能发生在 Activity 被销毁, 或者另一个 Activity(正在运行的或新启动的)被重新激活并将覆盖前一个 Activity

时。在调用 `onStop()` 之后, 如果 Activity 重新跟用户交互, Android 会调用 `onRestart()` 方法; 如果 Activity 退出了, 就会调用 `onDestroy()` 方法。

- 在 Activity 被销毁之前会调用 `onDestroy()`, 除非是内存不够, Android 强行终止了 Activity 的进程。在这种情况下就不会调用 `onDestroy()`。如果调用了 `onDestroy()`, 那它就是该 Activity 接收的最终调用。在 `onPause()`、`onStop()` 或 `onDestroy()` 返回之后, Android 可以终止托管 Activity 的进程。从 `onPause()` 返回后到调用 `onResume()` 之前, Activity 都处于可终止状态。在 `onPause()` 再次返回之前, Activity 都不会再处于可终止状态。

图 1-3 演示了包含 7 个方法的 Activity 生命周期。

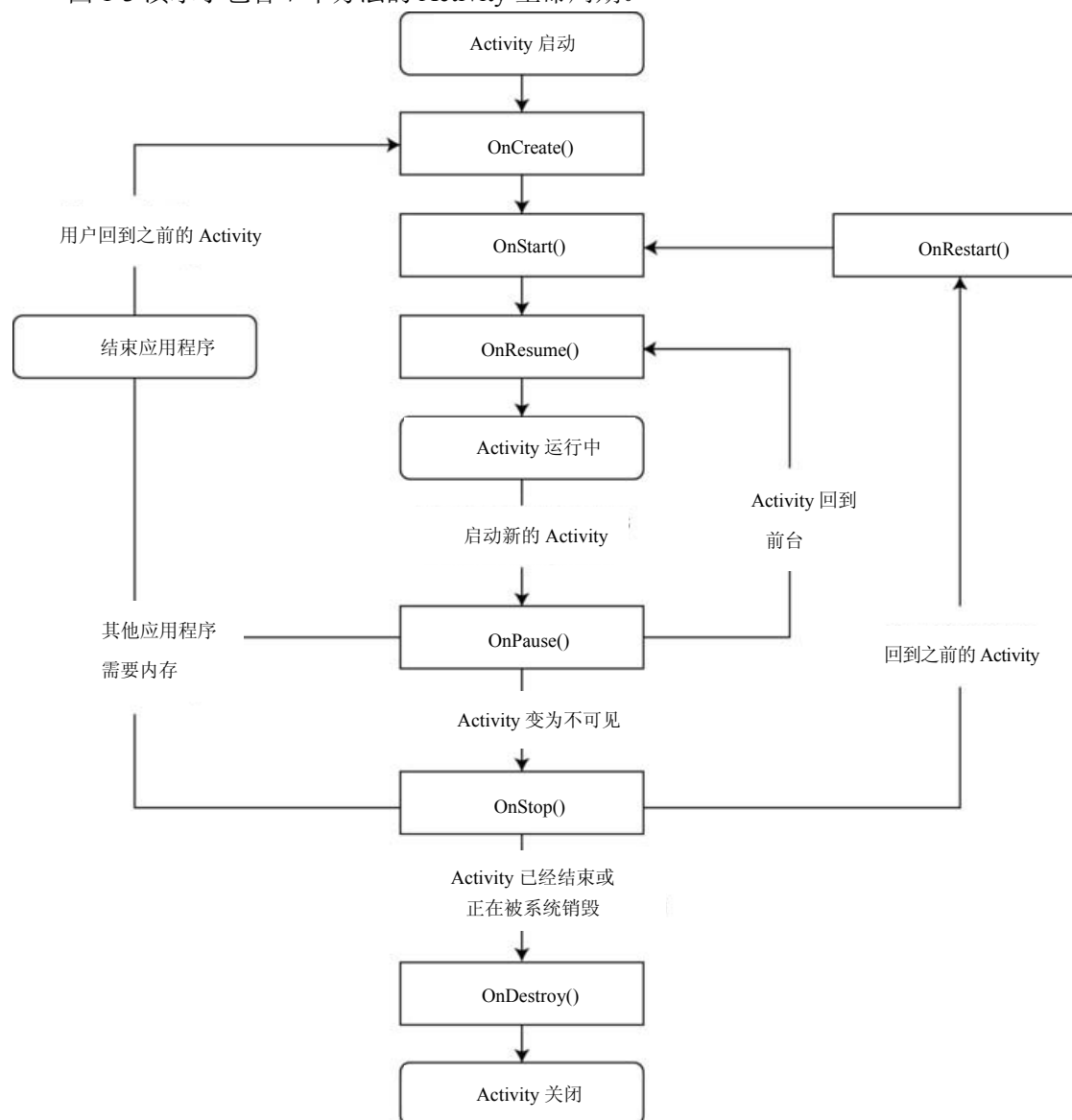


图 1-3 Activity 的生命周期表明并不一定会调用 `onDestroy()` 方法

图 1-3 显示 Activity 是通过调用 `startActivity()` 开始的。具体地说, 这个 Activity 是通过创建一个 `Intent` 对象(用于描述显式或隐式 `Intent`), 并把该对象传递给上下文类的 `void startActivity(Intent intent)` 方法(启动新的 Activity, 当它结束后不返回任何结果)启动的。

另外, 调用 Activity 类的 `void startActivityForResult(Intent intent, int requestCode)` 方法也能启动 Activity。然后, 特定的 `int` 结果会作为参数返回给 Activity 类的 `void onActivityResult(int requestCode, int resultCode, Intent data)` 回调方法。

注意:

Activity 响应后通过调用 Activity 类的 `Intent getIntent()` 方法能够查看启动 `Intent`。Android 系统调用 Activity 类的 `void onNewIntent(Intent intent)` 方法, 可将后续 `Intent` 传递到 Activity 中。

假设你已经创建了一个名为 `SimpleActivity` 的应用程序, 这个应用程序包含 `SimpleActivity`(见程序清单 1-1)和 `SimpleActivity2` 类。现假设你想要使用 `SimpleActivity` 的 `onCreate(Bundle)` 方法启动 `SimpleActivity2`, 示例如下:

```
Intent intent = new Intent(SimpleActivity.this, SimpleActivity2.class);
SimpleActivity.this.startActivity(intent);
```

第一行代码创建了一个描述显式 `Intent` 的 `Intent` 对象。将当前的 `SimpleActivity` 实例的引用和 `SimpleActivity2` 类的实例传递给 `Intent(Context packageContext, Class<?> cls)` 构造方法, 以初始化该对象。

第二行代码将这个 `Intent` 对象传递给 `startActivity(Intent)` 方法, 该方法负责加载 `SimpleActivity2.class` 描述的 Activity。如果 `startActivity(Intent)` 找不到指定的 Activity(这不该发生), 就会抛出 `android.content.ActivityNotFoundException` 异常。

从图 1-3 还可看出, 在应用程序终止之前, 方法 `onDestroy()` 可能不会被调用。因此, 不建议使用该方法来保存数据。例如, 如果一个 Activity 正在编辑内容提供器的数据, 这些数据通常应该在 `onPause()` 中进行处理。

注意:

`onDestroy()` 方法通常用来释放在 `onCreate(Bundle)` 中获得的系统资源(例如线程)。

这 7 个方法定义了 Activity 的整个生命周期, 组成了下面这 3 个嵌套的循环:

- Activity 的完整生命周期是第一次调用 `onCreate(Bundle)` 到最终调用 `onDestroy()` 的过程。Activity 会在 `onCreate(Bundle)` 中初始化所有的“全局”设置, 而在 `onDestroy()` 中释放所有资源。例如, 如果 Activity 在后台有一个从网络下载数据的线程, 它就可以在 `onCreate(Bundle)` 中创建该线程, 然后在 `onDestroy()` 中停止该线程。
- Activity 的可见生命周期就是调用 `onStart()` 到调用 `onStop()` 之间的时间段。在这段时间内, 用户可以在屏幕上看到这个 Activity(也有可能没在前台跟用户交互)。在这两个方法之间, Activity 可以获得将其展示给用户所需的资源。例如, 它可以在 `onStart()` 中注册一个广播接收器, 监控对用户界面有影响的变动; 当该 Activity 变为隐藏状态时, 则可以在 `onStop()` 中取消对该对象的注册。当 Activity 在用户可见状态和不可见状态之间切换时, 可以多次调用 `onStart()` 和 `onStop()` 方法。

- Activity 的前台生命周期就是调用 `onResume()` 到调用 `onPause()` 之间的时间段。在这段时间内, Activity 处于屏幕上的最前台, 正处于跟用户交互的状态。Activity 可以在重新激活和暂停状态之间频繁切换。例如, 当设备休眠或有新 Activity 启动时调用 `onPause()`, 当 Activity 有结果或收到新的 Intent 时调用 `onResume()`。这两个方法中的代码不宜过长。

Activity、任务和 Activity 栈

Android 将一系列相关的 Activity 称为任务, 用活动栈(也称为历史栈或回溯栈)保存这一序列。栈中的第一个 Activity 就是启动该任务的 Activity, 也称为根 Activity。该 Activity 通常是用户在设备的应用启动器中选择的。正在运行的 Activity 位于栈的顶部。

当前 Activity 启动另一个 Activity 时, 新的 Activity 就被压入栈并获得焦点(变成正在运行的 Activity)。而之前的 Activity 依然保存在栈中, 但已停止。当 Activity 停止时, 系统会保持其用户界面的当前状态。

当用户按下设备的返回键(BACK)时, 当前的 Activity 就被弹出栈(Activity 被销毁), 前一个 Activity 重新进入运行状态, 恢复操作(之前的用户界面状态也会还原)。

Activity 在栈中的顺序绝不会重排, 只有压栈和出栈。当 Activity 被压入栈时, 它就会成为当前运行的 Activity, 当用户按返回键离开 Activity 时, Activity 就从栈中弹出。这样, 栈就成了“后进先出”的对象结构。

每次用户按下返回键, 栈中的一个 Activity 就会被弹出并显示前一个 Activity。这个过程不断重复, 直到用户返回主界面或是回到任务中的第一个 Activity。当栈中的所有 Activity 都被移除时, 任务也就不存在了。

Google 的 Android 在线文档的“Tasks and Back Stack”一节更深入地讲解了关于 Activity 和任务的主体: <http://developer.android.com/guide/components/tasks-and-back-stack.html>。

2. 视图、ViewGroup 和事件监听器

Activity 的用户界面是基于视图(用户界面组件)、ViewGroup(由相关视图构成的集合)和事件监听器(用来监听视图或 ViewGroup 事件的对象)的。

注意:

Android 将视图看作 widget(控件)。不要将之与显示在 Android 主屏幕的 widget(小部件)混淆。虽然术语相同, 用户界面 widget(控件)和主屏幕 widget(小部件)的含义是不同的。用户界面 widget 是组件, 而主屏幕 widget(小部件)是正在运行的应用程序的微型视图。

View 是 `android.view.View` 的子类, 类似于 Java Swing 的组件。`android.widget` 包中包括各种 View 的子类, 如 Button、EditText 和 TextView(编辑框的父类)。

多个视图组成的集合是通过抽象类 `android.view.ViewGroup`(继承于视图类)的子类来描述的, 该类类似于 Java Swing 的容器类。`android.widget` 包中包括了不同的子类, 如 `LinearLayout`。

注意:

ViewGroup 类是 View 的子类, 即 ViewGroup 本身也是视图类。这样的安排允许在视图组中嵌入视图组, 这样就可以实现任意复杂度的屏幕设计。但不要过于复杂, 因为用户通常不想操作太复杂的屏幕。

事件监听器被定义在 View 和 ViewGroup(及其子类)类的内嵌接口成员变量中。例如 View.OnClickListener 监听器声明了 void onClick(View v)方法, 当单击视图(如按钮)对象时, 该方法会被调用。

下面的 onCreate(Bundle)方法使用按钮 Button、EditText 和 LinearLayout 类创建屏幕, 当用户输入文本并单击该按钮后, 可将该文本显示在一条弹出消息中:

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    LinearLayout layout = new LinearLayout(this);
    final EditText et = new EditText(this);
    et.setEms(10);
    layout.addView(et);
    Button btnOK = new Button(this);
    btnOK.setText("OK");
    layout.addView(btnOK);
    View.OnClickListener ocl;
    ocl = new View.OnClickListener()
    {
        @Override
        public void onClick(View v)
        {
            Toast.makeText(class.this, et.getText(),
                           Toast.LENGTH_SHORT).show();
        }
    };
    btnOK.setOnClickListener(ocl);
    setContentView(layout);
}
```

在调用父类的相同方法后, onCreate(Bundle)会实例化 LinearLayout 类。这个容器类以单一列或行布局其内部的子类对象, 其默认行为是将子类对象以行来布局。

this 关键字被传递给 LinearLayout 对象的构造方法, 其他控件的构造方法也采用该方式。通过引用当前的上下文对象 this, 可以在必要时允许控件加载和访问资源(在本章后面讨论)。

接下来, EditText 类被实例化后, 调用继承的 void setEms(int ems)方法(继承自 TextView 类)来将控件的宽度设置为 10 em(相对长度单位, 一个 em 等于大写字母“M”在默认字体时的高度)。

此时, 会调用 LinearLayout 继承的方法(从父类 ViewGroup)void addView(View child), 将 EditText 控件添加到 LinearLayout 实例对象中。

EditText 对象创建完成后, 由 `onCreate(Bundle)` 方法实例化 Button 对象, 调用方法 `void setText(CharSequence text)` (继承自 `TextView`) 来设置按钮显示 OK 文本, 并将其添加到 `LinearLayout` 对象中。

`onCreate(Bundle)` 接着会实例化一个匿名类, 它实现了 `View.OnClickListener` 接口, 覆写 `onClick(View)` 方法来显示 toast 提示框(能在窗口中短时间地显示一条弹出消息)。

警告:

上述代码片段是存在执行效率问题的, 该代码片段先创建单击监听器, 随后将其绑定到按钮对象。这种方法是低效的, 因为它需要一个新的对象(一个匿名类的实例, 以实现 `View.OnClickListener` 接口), 该对象在每次调用 `onCreate(Bundle)` 方法时均被创建(每当设备定位方向发生变化时, `onCreate` 方法都会被调用)。一种更有效的方法是使用 `ocl` 实例变量, 只有当 `ocl` 中不包含空引用时, 才实例化匿名类。然而, 上述问题有一个更好的解决方案, 本章后面探讨资源时会展示该方案。

toast 是通过调用 `android.widget.Toast` 类的 `makeText(Context context, CharSequence text, int duration)` 工厂方法创建的, 其中传递到 `duration` 参数的值为 `Toast.LENGTH_SHORT` 或 `Toast.LENGTH_LONG`。在 `Toast` 实例创建完成后, 会调用 `void show()` 方法, 从而在指定的一段时间内显示 toast 提示框(该方法被调用时消息逐渐淡入, 持续一段时间后逐渐淡出)。

创建完监听器后, `onCreate(Bundle)` 方法调用了 Button 的 `void setOnClickListener(View.OnClickListener l)` 方法(继承自 `View`), 将前面创建的监听器对象注册到按钮中。

最后, `onCreate(Bundle)` 调用 Activity 的 `void setContentView(View view)` 方法。这个方法用来将 `LinearLayout` 实例设置到 Activity 的视图结构中, 以便编辑框和按钮控件可以在 `LinearLayout` 实例中单行显示。

注意:

虽然通过实例化控件类可以创建用户界面, 但采用资源文件创建会更好。这会在本章后面进行讨论。

3. fragment

Android 3.0 引入了 `fragment` 的概念, 它们代表 Activity 的部分用户界面对象。`fragment` 代表 Activity 的界面区域, 每个 `fragment` 具有独立的生命周期, 能够接受自身的输入事件, 并且在 Activity 运行过程中, 可添加或删除 `fragment`。用户可将多个 `fragment` 组合成单一的 Activity, 从而建立多窗格的用户界面(通常在平板电脑中), 或在多个 Activity 中重用 `fragment` 对象。

注意:

必须总是在 Activity 中内嵌 `fragment` 对象。

Google 在 Honeycomb 中引入 `fragment` 主要是为了在平板电脑和其他的大屏幕设备中, 支持更加动态和灵活的用户界面。因为平板电脑的屏幕比手机要大得多, 存在更多的空间来组合和交换控件。使用 `fragment` 不必管理复杂的视图层次的变化, 通过将 Activity 的布

局放到 `fragment` 中进行组织，用户能够在运行时修改 `Activity` 的外观，并能够在 `Activity` 管理返回栈中保存其改变后的状态。

提示：

用户应该将每个 `fragment` 设计成模块化、可重用的 `Activity` 组件。因为每个 `fragment` 定义了自己的外观布局并拥有自身生命周期的回调行为方法，可在多个 `Activity` 中包含同一个 `fragment`，因此用户应该尽量采用重用设计，避免直接通过其他 `fragment` 操作一个 `fragment`。上述观点特别重要，因为模块化的 `fragment` 可通过修改 `fragment` 组合来适应不同屏幕的大小。

例如，新闻应用程序提供了文章标题列表和当前选择的文章的内容。平板电脑可以将标题列表和内容显示在同一个屏幕上。然而手机仅会将标题列表显示在一个屏幕上，而将内容显示在另一个屏幕上。你应该这样设计用户界面，使用一个 `fragment` 管理标题列表，其他的 `fragment` 管理内容。然后就可以在不同的布局配置中重用这些 `fragment`，并根据可用的屏幕空间来优化用户体验效果，如图 1-4 所示。

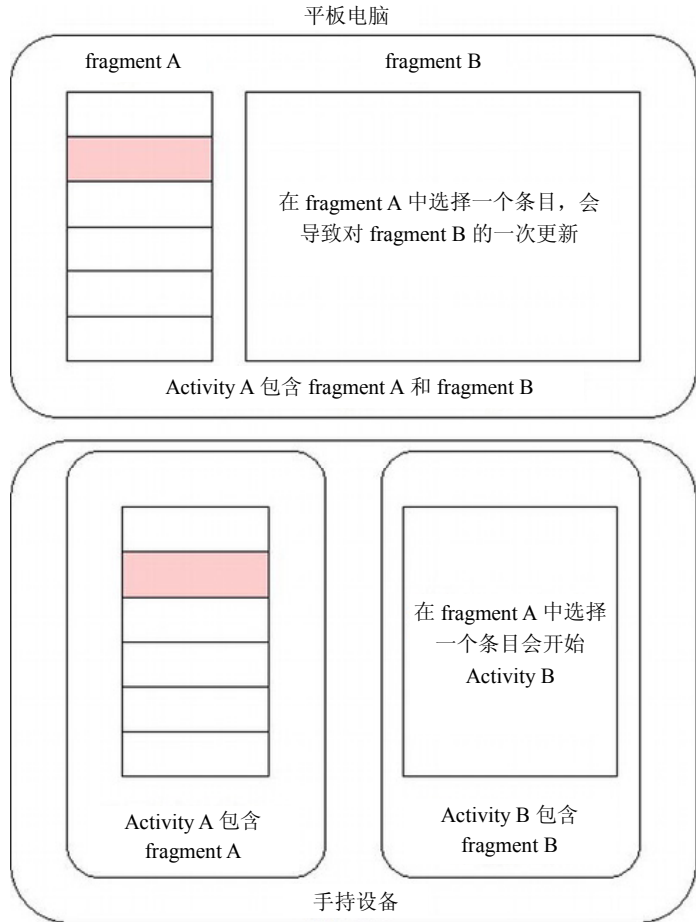


图 1-4 两个由 `fragment` 定义的用户界面。在平板电脑设计中，它们组合成一个 `Activity`，但在手机设计中，它们被拆分成两个 `Activity`

根据图 1-4, 当在平板设备中运行应用程序时, 应用程序将两个 `fragment` 嵌入在 `Activity` A 中。但在手机屏幕上, 由于缺乏足够的空间容纳两个 `fragment` 对象, 因此 `Activity` A 仅包括文章列表 `fragment` 对象。当用户选择一篇文章, `Activity` A 启动 `Activity` B, 该 `Activity` 包含了第二个 `fragment` 来显示这篇文章的内容。因此, 通过在不同的组合中重用 `fragment`, 应用程序可实现同时支持平板电脑和手机。

注意:

在 [Googlehttp://developer.android.com/guide/components/fragments.html](http://developer.android.com/guide/components/fragments.html) 中查阅 “Fragments” 相关文档, 可了解更多关于 `fragment` 的内容。

4. 剖析 Service

`Service`(服务)是运行在后台的组件, 其运行不限时间, 且不提供用户界面。与 `Activity` 类似, `Service` 运行在进程的主线程中, 它必须创建另一个线程以执行耗时操作。`Service` 分为本地 `Service` 或远程 `Service` 两类:

- 本地 `Service` 与应用程序其他部分运行在相同的进程中, 这种 `Service` 很容易执行后台任务。
- 远程 `Service` 运行在独立的进程中, 这种 `Service` 可以用来执行进程间的相互通信。

注意:

`Service` 并不是单独的进程, 尽管可以被指定在单独的进程中运行。此外 `Service` 也不是线程。相反 `Service` 允许应用程序告知 `Android` 系统该应用程序的某些功能希望运行在后台(甚至当用户不直接与应用程序进行交互时), 并允许应用程序将它的一些功能暴露给其他应用程序。

试想播放用户选择的音乐服务, 用户可利用 `Activity` 来选择播放歌曲, 并启动一个服务来响应用户的选择。服务会在另一个线程中进行音乐播放, 防止应用程序出现 `ANR`(`Application Not Responding`, 应用程序没有响应)对话框(详见附录 C)。

注意:

使用服务播放音乐的原则是即使初始化音乐的 `Activity` 界面从屏幕上消失, 用户也仍然希望音乐可以继续播放。

所有的服务都在抽象类 `android.app.Service` 的子类中, 而 `android.app.Service` 类则是 `Context` 的间接子类。

`Service` 子类需要覆写不同的 `Service` 生命周期回调方法(`Android` 在 `Service` 的生命周期中调用这些方法)。例如, 在程序清单 1-2 中的 `SimpleService` 类继承了 `Service`, 也覆写了 `void onCreate()`和 `void onDestroy()`生命周期回调方法。

程序清单 1-2 一个 Service 子类, 版本 1

```
import android.app.Service;

public class SimpleService extends Service
{
    @Override
    public void onCreate()
    {
        System.out.println("onCreate() called");
    }
    @Override
    public void onDestroy()
    {
        System.out.println("onDestroy() called");
    }
    @Override
    public IBinder onBind(Intent intent)
    {
        System.out.println("onBind(Intent) never called");
        return null;
    }
}
```

初始创建 Service 时会调用 onCreate(), 当服务被删除时则会调用 onDestroy()。因为 IBinder onBind(Intent intent)生命周期回调方法是抽象方法, 所以就算只是返回 null, 也必须重载这个方法。如果返回值是 null, 就表示要忽略这个方法。

注意:

Service 子类通常会重载 onCreate()方法和 onDestroy()方法, 以实现初始化并释放资源。跟 Activity 的 onCreate(Bundle)方法和 onDestroy()方法不同, Service 的 onCreate()方法不会被反复调用, 而 onDestroy()方法则是一定会被调用的。

Service 的生命周期就是调用 onCreate()方法和 onDestroy()方法返回之间的时间。Service 也是一种 Activity, Service 会在 onCreate()方法中初始化, 在 onDestroy()方法中释放所占据的资源。例如, 音乐播放 Service 会在 onCreate()方法中创建播放音乐的线程, 在 onDestroy()方法中停止该线程。

5. 本地 Service

本地 Service 通常是由 Context 的 ComponentName startService(Intent intent)方法启动的, 该方法会返回一个 android.content.ComponentName 实例, 用于标识所启动的 Service 组件; 若服务不存在, 则返回 null 引用。图 1-5 中进一步展示了 startService(Intent)在生命周期中的作用。

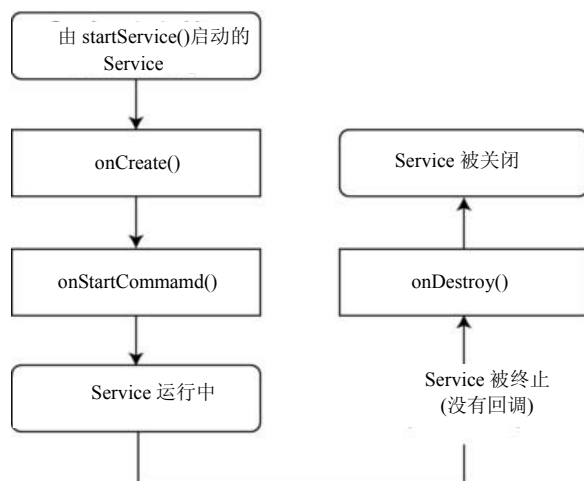


图 1-5 由 startService(Intent)调用 onStartCommand(Intent, int, int)启动的 Service 生命周期

调用 startService(Intent)方法会导致调用 onCreate()方法，然后会调用 int onStartCommand(Intent intent, int flags, int startId)。后一个生命周期回调方法会取代过时的 void onStart(Intent intent, int startId)方法，调用参数如下：

- intent 是传递给 startService(Intent)的 Intent 对象。
- flags 用于提供关于启动请求的其他信息，但通常设置为 0。
- startId 是用于描述该启动请求的唯一整数编号，可以将这个值传递给 Service 的 boolean stopSelfResult(int startId)来将自己停止。

onStartCommand(Intent, int, int)会处理 Intent 对象，通常会返回 Service.START_STICKY 常量，表示在终止该 Service 之前它会一直运行。此时，Service 正在运行，而且在发生下列某个事件之前会持续运行。

- 另一个组件调用 Context 的 boolean stopService(Intent intent)方法以终止该 Service。无论调用了多少次 startService(Intent)，都只需要调用一次 stopService(Intent)即可终止 Service。
- 该 Service 调用 Service 重载的 stopSelf()方法或是调用 Service 的 stopSelfResult(int)方法来将自己停止。

在调用 stopService(Intent)、stopSelf()或 stopSelfResult(int)方法之后，Android 会调用 onDestroy()，让 Service 执行清理任务。

注意：

调用 startService(Intent)方法启动服务时，不会调用 onBind(Intent)。

程序清单 1-3 展示了可以在 startService(Intent)方法中使用的 Service 子类的框架。

程序清单 1-3 一个 Service 子类，版本 2

```
import android.app.Service;
```

```

public class SimpleService extends Service
{
    @Override
    public void onCreate()
    {
        System.out.println("onCreate() called");
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        System.out.println("onStartCommand(Intent, int, int) called");
        return START_STICKY;
    }
    @Override
    public void onDestroy()
    {
        System.out.println("onDestroy() called");
    }
    @Override
    public IBinder onBind(Intent intent)
    {
        System.out.println("onBind(Intent) never called");
        return null;
    }
}

```

假如下面的代码片段位于程序清单 1-1 的 SimpleActivity 类的 onCreate()方法中，它通过传递一个显式 Intent，利用 startService(Intent)方法启动了程序清单 1-3 中的 SimpleService 类的一个实例：

```

Intent intent = new Intent(SimpleActivity.this, SimpleService.class);
SimpleActivity.this.startService(intent);

```

6. 远程 Service

远程 Service 是通过 Context 类的 boolean bindService(Intent service, ServiceConnection conn, int flags)方法来启动的，它会连接到一个正在运行的服务(如果需要，则创建该服务)，如果成功连接，则返回“true”。bindService(Intent, ServiceConnection, int)方法在服务的生命周期中的执行情况如图 1-6 所示。

调用 bindService(Intent, ServiceConnection, int)会导致调用 onCreate()方法，随后会调用 onBind(Intent)函数，该函数会返回一个用于跟 Service 交互的通信通道(实现 android.os.IBinder 接口的类的实例)。

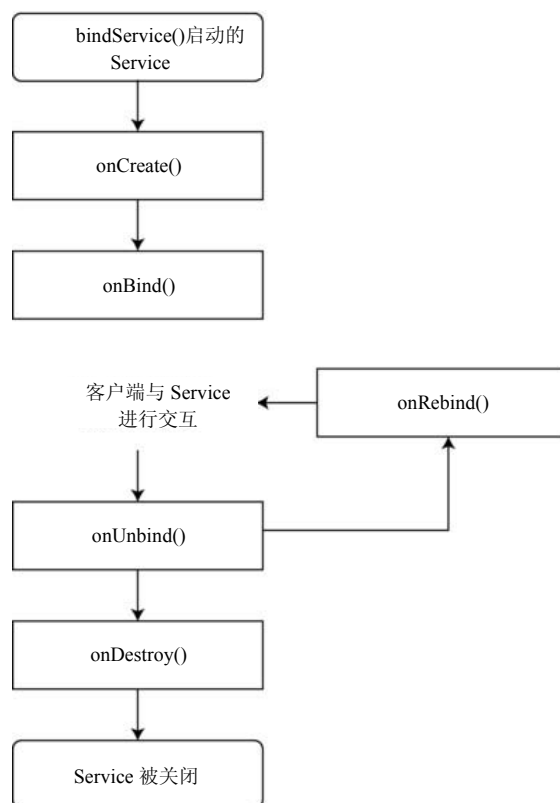


图 1-6 仅由 `bindService(Intent, ServiceConnection, int)` 启动的 Service 的生命周期，这里没有调用 `onStartCommand(Intent, int, int)`

客户端与 Service 交互的过程如下：

(1) 客户端子类化 `android.content.ServiceConnection` 类，并覆写该类的抽象方法 `void onServiceConnected(ComponentName className, IBinder service)` 和 `void onServiceDisconnected(ComponentName name)` 方法，以接收服务启动和停止的信息。当 `bindService(Intent, ServiceConnection, int)` 返回 `true` 时，且与服务的连接已经建立时，前一个方法会被调用，传递给该方法的 `IBinder` 参数的值与 `onBind(Intent)` 的返回值是相同的。当与服务的连接已经中断时，后一个方法会被调用。

连接中断通常在服务主进程崩溃或被销毁时发生，`ServiceConnection` 实例本身并没有被删除，与服务的绑定仍然处于活动状态，当该服务下次运行时，客户端将会调用 `onServiceConnected(ComponentName, IBinder)` 方法。

(2) 客户端将 `ServiceConnection` 子类对象传递给 `bindService(Intent, ServiceConnection, int)` 方法。

客户端通过调用 `Context` 的 `void unbindService(ServiceConnection conn)` 方法与 Service 断开连接。断开连接后，Service 重启时，该组件也不会收到调用。如果没有其他组件绑定到该 Service，那么该 Service 随时都可以停止。

在 Service 可以停止之前，Android 会调用 Service 的 `boolean onUnbind(Intent intent)` 生命周期回调方法，其中的 `Intent` 对象会传递给 `unbindService(ServiceConnection)`。假设

onUnbind(Intent)没有返回 true(如果返回 true, 就是让 Android 在之后每次客户端绑定到该 Service 时都调用 void onRebind(Intent intent)生命周期回调方法), Android 则调用 onDestroy() 销毁该 Service。

程序清单 1-4 展示了用于 bindService(Intent, ServiceConnection, int)方法的 Service 子类的结构。

程序清单 1-4 一个 Service 子类, 版本 3

```
import android.app.Service;

public class SimpleService extends Service
{
    public class SimpleBinder extends Binder
    {
        SimpleService getService()
        {
            return SimpleService.this;
        }
    }
    private final IBinder binder = new SimpleBinder();
    @Override
    public IBinder onBind(Intent intent)
    {
        return binder;
    }
    @Override
    public void onCreate()
    {
        System.out.println("onCreate() called");
    }
    @Override
    public void onDestroy()
    {
        System.out.println("onDestroy() called");
    }
}
```

程序清单 1-4 首先声明一个内部类 SimpleBinder, 该类继承于 android.os.Binder。SimpleBinder 类声明了 SimpleService 类的 getService()方法, 它返回 SimpleService 子类的实例。

注意:

Binder 和 IBinder 接口一起支持用于进程间通信的远程过程调用机制。尽管本例中 Service 和应用的其他部分运行在同一个进程中, 但依然需要 Binder 和 IBinder。

程序清单 1-4 接下来创建了 SimpleBinder 的实例, 并将其引用分配给私有的 binder 变量。该变量的值由后面覆写的方法 onBind(Intent)返回。

假设在程序清单 1-1 的 SimpleActivity 类声明了一个私有 SimpleService 类型的变量 ss (private SimpleService ss;)。接着假设以下示例包含在 SimpleActivity 类的 onCreate(Bundle) 方法中。

```
ServiceConnection sc = new ServiceConnection()
{
    @Override
    public void onServiceConnected(ComponentName className, IBinder
service)
    {
        ss = ((SimpleService.SimpleBinder) service).getService();
        System.out.println("Service connected");
    }
    @Override
    public void onServiceDisconnected(ComponentName className)
    {
        ss = null; System.out.println("Service disconnected");
    }
};
bindService(new Intent(SimpleActivity.this, SimpleService.class), sc,
Context.BIND_AUTO_CREATE);
```

这段代码首先实例化一个 ServiceConnection 子类。覆写 onServiceConnected(ComponentName, IBinder)方法并用它的 service 参数调用 SimpleBinder 的 getService()方法，并保存结果。

尽管 onServiceDisconnected(ComponentName)方法必须被覆写，但实际上它是不会被调用的。因为 SimpleService 和 SimpleActivity 运行在同一个进程中。

这段代码接下来将这个 ServiceConnection 子类对象，以及将 SimpleService 设为目标的 Intent 和 Context.BIND_AUTO_CREATE(创建永久连接)一起传递给 bindService(Intent, ServiceConnection, int)。

注意：

Service 启动(用 startService(Intent))后可以与多个连接绑定(用 bindService(Intent, ServiceConnection, int))。在这种情况下，Service 启动后或者至少有一个带 BIND_AUTO_CREATE 标识的连接绑定了该 Service，Android 就会保持其运行状态。如果这两个条件都不满足，就会调用 Service 的 onDestroy()方法将其停止。诸如停止线程、取消 Broadcast Receiver 的注册等清理工作应该在 onDestroy()方法返回之前完成。

7. 剖析 Broadcast Receiver 组件

Broadcast Receiver(广播接收器)是用来接收和响应广播的组件。许多广播源于系统代码，例如，时区改变或电池能量不足时系统都会发送广播。

应用程序也可以发起广播。例如，应用程序可能希望让其他应用程序知道某些数据已经从网络下载到设备上，当前已经可以使用这些数据。

Broadcast Receiver 是通过定义 android.content.BroadcastReceiver 抽象类的子类并覆写 Broadcast Receiver 父类的 void onReceive(Context context, Intent intent)抽象方法来定义的。

例如，程序清单 1-5 中的 SimpleBroadcastReceiver 类就继承了 BroadcastReceiver 父类并覆写了 onReceive()方法。

程序清单 1-5 Broadcast Receiver 的基本结构

```
public class SimpleBroadcastReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        System.out.println("onReceive(Context, Intent) called");
    }
}
```

创建一个 Intent 对象并将其传递给 Context 的任何一个广播方法(例如 Context 重载的 sendBroadcast()方法)就能启动 Broadcast Receiver, 将消息广播给所有对此感兴趣的 Broadcast Receiver。

假如将下面这段代码放入程序清单 1-1 中的 SimpleActivity 类的 onCreate()方法中, 就会创建程序清单 1-5 中的 SimpleBroadcastReceiver 类的如下实例:

```
Intent intent = new Intent(SimpleActivity.this, SimpleBroadcastReceiver.class);
intent.putExtra("message", "Hello, broadcast receiver!");
SimpleActivity.this.sendBroadcast(intent);
```

调用 Intent 的 Intent putExtra(String name, String value)方法是为了将消息保存为键/值对。跟 Intent 的其他 putExtra()方法一样, 该方法也会返回对 Intent 对象的引用, 以便连续调用其他方法。

8. 剖析 Content Provider

Content Provider(内容提供者)是可将应用程序特定的数据集提供给其他应用程序使用的组件。数据可以存储在 Android 的文件系统、SQLite 数据库中, 或使用其他方式进行存储。

Content Provider 非常适合直接访问原始数据, 因为 Content Provider 很好地将原始数据的格式与组件代码进行了解耦。这种解耦方式可以防止格式变化时再重新修改代码。

Content Provider 是通过定义 android.content.ContentProvider 抽象类的子类, 并重载 Content Provider 基类的抽象方法(例如 String getType(Uri uri))来定义的。例如, 程序清单 1-6 中的 SimpleContentProvider 类就继承了 ContentProvider 基类并重载了一些基类方法。

程序清单 1-6 Content Provider 的基本结构

```
public class SimpleContentProvider extends ContentProvider
{
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs)
    {
        System.out.println("delete(Uri, String, String[]) called");
    }
}
```

```

        return 0;
    }
    @Override
    public String getType(Uri uri)
    {
        System.out.println("getType(Uri) called");
        return null;
    }
    @Override
    public Uri insert(Uri uri, ContentValues values)
    {
        System.out.println("insert(Uri, ContentValues) called");
        return null;
    }
    @Override
    public boolean onCreate()
    {
        System.out.println("onCreate() called");
        return false;
    }
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder)
    {
        System.out.println("query(Uri, String[], String, String[], String)
called");
        return null;
    }
    @Override
    public int update(Uri uri, ContentValues values, String selection,
                     String[] selectionArgs)
    {
        System.out.println("update(Uri, ContentValues, String, String[])
called");
        return 0;
    }
}

```

客户端不会实例化 SimpleContentProvider 并直接调用这些方法。客户端会实例化 android.content.ContentResolver 抽象类的子类，然后再调用这些方法(例如 public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder))。

注意:

ContentResolver 实例可以与任何 Content Provider 交互。可以与 Content Provider 一起管理其所涉及的所有跨进程通信。

1.4.2 资源

资源是图像、字符串和组成应用程序的其他实体。开发者将它们存储在外部文件中来

维持代码的独立性。同时,将资源从代码中分离使得应用程序更适合运行在多种设备中,并支持多语言(地理、政治或文化区域)。

Android 支持以下类型的资源:

- **Animation(动画):** 通过设置属性动画(在该动画中,对象的属性值(例如背景颜色或透明值)在一段时间内进行修改)或视图动画(执行一系列变换操作(例如旋转或渐变)的动画,包括单个图像的补间动画和或多个图像连续显示而形成的帧动画)来模拟运动。
- **颜色状态列表:** 与控件状态(如按钮按下、获得焦点、没有按下和没有焦点)建立映射关系的颜色列表。
- **Drawable:** 绘制在屏幕上并通过合适的 API 获取的图形(例如位图)。
- **布局:** 屏幕上控件的布置方式(例如线性布局)。
- **菜单:** 应用程序菜单,如选项菜单(Activity 菜单项的主要组织方式)或上下文菜单(悬浮的菜单)。
- **字符串:** 文本、文本数组或具有可选样式和格式的多元文本。
- **样式:** 用户界面的外观格式,包括控件(如按钮)、Activity 或应用程序的外观格式。
- **其他资源:** 布尔值、色值、尺寸值、唯一标识符、整型值、整型数组、类型化数组和 raw/asset 文件夹。

注意:

样式是属性的集合,用于指定视图或窗口的外观和格式。Theme(主题)则是应用于整个应用程序或 activity 的样式,而不仅仅是单一的视图。

1. 资源的分类

Android 将资源分为默认资源、可选资源或平台相关的资源。前两种资源以文件的形式供开发者使用,存放在应用程序项目的 res 目录的子目录中。这些文件是不能直接放置在 res 目录中的,否则应用程序在编译时会出现错误。

默认资源

如果不存在可选资源来匹配当前的设备配置,会使用默认资源。例如,可使用相同的布局资源在小屏幕设备和大屏幕设备中布置控件。另一个示例是在使用英文文本字符串时,可不用考虑设备的地区设置问题。

默认资源存储在 res 目录的以下子目录中:

- **anim** 子目录保存了定义补间动画的 XML 文件。
- **animator** 子目录保存了定义属性动画的 XML 文件。属性动画的 XML 文件也可保存在 anim 目录中,然而属性动画应首选 animator 目录,以便更好地区分两类动画。
- **color** 子目录保存了定义颜色列表的 XML 文件。
- **drawable** 存储位图文件(.png、.png、.jpg、.gif)或可编译成位图文件、可变尺寸位图(nine-patches)、状态列表、形状、帧动画绘制或其他可绘制资源的 XML 文件。
- **layout** 子目录保存了用户界面布局的 XML 文件。
- **menu** 子目录保存了用于定义不同类型应用程序菜单(例如上下文菜单)的 XML 文件。

- **raw** 子目录用来存储原始文件，由于该目录中原始文件的名称已不再存在，因此为了能保存文件名称(及文件层次结构)，可以在 **assets** 目录中保存这些文件，**assets** 目录与 **res** 目录属于同级目录。
- **values** 子目录保存了定义简单类型数据的 XML 文件，例如字符串、整数或颜色值。在这个目录下的每个文件都能够定义多个资源，而在其他目录的 XML 文件仅能定义一种资源。由于每个资源定义了自己的 XML 元素，因此可任意为这些文件命名，并在同样的文件中使用不同的资源类型。但是，为了使意义更加明确，最好用不同的文件定义不同的资源类型并采用以下命名约定：**arrays.xml** 用来定义资源数组(即类型化数组)、**colors.xml** 用来定义颜色值、**dimens.xml** 用来定义设备尺寸值、**strings.xml** 用来定义字符串值、**styles.xml** 用来定义样式。
- **xml** 子目录保存了任意的 XML 文件，包括各种配置文件，例如可搜索的配置文件(一种基于 XML 的配置文件，该文件支持 Android 协助搜索，可在 Activity 中设置搜索查询，并提供搜索建议)。

保存在上述子目录中的资源定义了应用程序的默认设计和内容。除非被可选资源覆盖，否则它们会被应用于当前的 Android 设备。

可选资源

可选资源主要用于特定设备的配置。例如，当设备切换到横屏模式时，适用于横屏模式的最优布局将会代替默认的竖屏布局资源。另一个示例是当设备的地区设置更改为法国时，法语字符串的文本将替换默认的英语字符串。

与默认资源类似，可选资源存储在 **res** 目录的子目录中。每个子目录的名称以默认的资源子目录名称开始，然后用连字符(“-”)后跟配置限定名。例如，**layout-land** 标识了用于存储横屏布局文件的子目录。

下面的列表标识了一些配置限定名，它们可以被附加到默认的资源子目录名称后面：

- **语言和区域**：语言是由“ISO 639 - 1”(http://en.wikipedia.org/wiki/ISO_639-1)语言编码的两个字母表示，另外，后面还可以附加“ISO 3166 -1-alpha-2”(http://en.wikipedia.org/wiki/ISO_3166-1-alpha-2)地区编码的两个字母，并在前面加上小写字母 **r**。这些代码是不区分大小写的，前缀 **r** 用来区分地区编码，不能仅指定区域编码而缺少语言编码。语言和区域示例包括 **en**、**fr**、**en-rUS**、**en-rGB**、**fr-rFR** 和 **fr-rCA**。
- **平台版本**：设备支持的 API Level，以小写字母 **v** 开头的数字代码来表示。例如，**v1** 代表 API Level 1(支持 Android 1.0 或更高的设备)；**v4** 表示 API Level 4(支持 Android 1.6 或更高的设备)。
- **屏幕方向**：指定竖屏(纵向)或横屏(横向)方向。
- **屏幕像素分辨率**：**ldpi** 表示低分辨率屏幕(大约 120 dpi(每英寸点数))；**mdpi** 为中等分辨率(在传统 HVGA)屏幕(大约 160 dpi)；**hdpi** 为高分辨率屏幕(大约 240 dpi)；**xhdpi** 为超高分辨率屏幕(大约 320 dpi)；**nodpi** 主要用于位图资源，代表不能按比例缩放以匹配设备屏幕的像素分辨率；**tvdpi** 用于像素分辨率介于 **mdpi** 和 **hdpi** 之

间(约 213 dpi)的屏幕。xhdpi 限定符是在 API level 8 中引入的, 而 tvdpi 限定符是在 API level 13 中引入的。

- 屏幕尺寸: 小尺寸屏幕与低分辨率 QVGA 屏幕相似(最小布局尺寸为 320×426 dp(与密度无关)); 正常尺寸屏幕与中等分辨率 HVGA 屏幕相似(最小布局尺寸是大约 320×476 dp); 大尺寸屏幕与中等分辨率 VGA 屏幕相似(最小布局尺寸大约 480×640 dp); 超大尺寸屏幕远远大于传统的中等分辨率 HVGA 屏幕(最小布局尺寸大约是 720×960 dp)。超大屏幕设备很可能是面向平板电脑的设备。从 API level 4 开始支持屏幕大小配置的限定符, 而 xlarge 限定符则是从 API level 9 开始支持的。

支持多种类型的屏幕

Android 旨在支持各种屏幕方向、大小和分辨率, Google 的“Supporting Multiple Screens”文档(http://developer.android.com/guide/practices/screens_support.html)对上述功能进行了说明。同时, Google 还在尺寸资源文档中(<http://developer.android.com/guide/topics/resources/moreresources.html#Dimension>)讨论了 Android 支持的各种计量单位。考虑到本章和附录 D 采用的是密度无关的像素和比例无关的像素, 这两个计量单位的定义如下:

密度无关的像素(dp 或 dp)是一个抽象的单位(即虚拟像素), 基于屏幕的物理密度。该单位相对于 160 dpi 屏幕, 1dp 代表 160 dpi 屏幕上的一个像素。dp 与像素的比率随着屏幕分辨率的变化而变化, 但不一定成正比。当采用密度无关方式定义布局尺寸或位置时, 可使用该单位。例如, 在 XML 文件中声明 5 个密度无关像素(而不是 5 个密度相关的像素)来作为视图的 padding, 其语句为: `android:padding="5dp"` (或 `android:padding="5dp"`), 而不是 `android:padding="5px"`。

比例无关的像素(sip 或 sp)类似于 dp, 但与用户的字体大小成比例关系。当在资源中使用字体大小来表示尺寸时可使用这个单位, 字体能够自动调整大小以适应屏幕分辨率和用户的偏好设置。例如, 在 XML 文件可以指定 `android:textSize="15 sp"` 而不是 `android:textSize="15 px"` 来设置 `<TextView>` 元素中的字体大小。

在默认的资源子目录的名称中, 经常包含多个限定符, 使用连字符将限定符和它的前导字符连接即可。例如, `drawable-fr-port` 代表设备被设置为法语和竖屏显示时使用的 Drawable 资源。

Android 配置限定符名称的规则:

- 当指定多个配置限定符名称时, 它们必须按照 Google 的“Providing Resources”(<http://developer.android.com/guide/topics/resources/providing-resources.html>)文档中指定的顺序。否则, 相关的资源将会被忽略。例如, `drawable-land-mdpi` 是正确的, 而 `drawable-mdpi-land` 是错误的。
- 可选资源不能内嵌使用, 例如, 不能指定 `res/drawable/drawable-fr`, 相反, 指定 `res/drawable-fr` 是正确的。
- 不能给限定符类型指定多个值, 例如, 不能为加拿大和法国指定 `drawable-rCA-rFR` 目录来存储相同的绘图文件。相反, 必须为它们创建独立的 `drawable-rCA` 和

drawable-rFR 目录,以便在每个目录中包含合适的文件或别名(别名将在本章后面讨论)。

应用程序必须具有默认资源和可选资源,你也可以在应用程序中使用 Google 提供的各种平台资源。

平台资源

Google 已经标准化了一些资源(例如样式、主题和布局),它们可供用户自己使用。这些平台资源可以通过 Android 包的 R 类及其子类(例如, R.anim 和 R.layout)来访问。平台资源更多的内容将在本章后面进行讨论。

2. 资源的访问

在创建应用程序的默认资源和替代资源(文件名遵从上述约定,并存储在正确命名的 res 子目录中)后,你可通过代码和/或其他 XML 文件,访问上述资源。此外也能访问平台资源。

基于代码的访问方式

Android 的 Asset Packager Tool (aapt)工具在项目创建的目录中,在应用程序项目的包层次结构下,生成一个名为 R.java 的文件。这个 Java 源文件为所有 res 目录下的资源存储了资源的 ID (包括资源名称和整型数值)。资源 ID 名称包括以下两部分:

- 资源类型: 每个资源均属于一种类型,例如绘图、字符串和布局类型。该类型使用采用基于 XML 的 id 来描述,通过定义元素的“android:id 特性”,并利用语法“@ + id /资源名称”来标识。例如, <TextView android:id="@+id/msg" /> 定义了 XML 中 <TextView> 元素 msg。
- 资源名称: 每个资源都有一个名称,该名称是文件名(不包括扩展名)、XML 文件的 android:name 特性值(当资源是简单的值时,例如字符串)或资源名称(当资源是通过语法“@ + id /资源名称”进行定义时)。

具备上述信息,即可通过代码来访问资源,方法如下:

R.资源类型.资源名称

R 为 R.java 所描述的类,资源类型和资源名称代表资源的类型和名称,并以句点分隔每个组件。例如, R.string.cancel 是指在类 R 中,访问具有字符串类型且资源名称为 cancel 的成员。

很多 Android API 方法都需要资源 ID 作为调用参数。例如, android.content.res.Resources 类的实例(通过调用 Context 类的 Resources getResources()方法返回)提供了多种方法,可返回应用程序的资源,这些方法均需要特定的资源 ID 作为参数,如下:

```
Resources res = getContext();
Drawable flag = res.getDrawable(R.drawable.canada);
String country = res.getString(R.string.canada);
```

Drawable getDrawable(int id)方法返回一个 android.graphics.Drawable 对象。drawable 资

源的 Drawable 对象通过 `R.drawable.canada` 来表示(可能是存储在 `res/drawable` 目录中的位图文件)。String `getString(int id)` 方法返回字符串资源, `id` 参数为 `R.string.canada`, 通常位于 `res/values` 目录的 `strings.xml` 文件中。

假设你已经创建了英语和法语环境的 `strings.xml` 文件, 在 `res/values` 和 `res/values-fr` 目录下具有 `canada` 条目, 当设备的语言环境被设置为英语时, Android 能从 `res/values/strings.xml` 文件中获得 `R.string.canada` 值。当语言环境被设置为法语时, Android 从 `res/valuesfr/strings.xml` 文件获得相应的 `id` 值。如果 Android 在 `res/values-fr/strings.xml` 文件中无法找到 `canada`, 则默认采用 `res/values/strings.xml` 文件的值。

注意:

可以调用 `Resources` 类的 `openRawResource()` 方法, 并以 `R.raw.file name` 作为资源 ID 来访问原始资源, 其中 `file name` 对应于原始文件名。所有方法在读取资源后将返回 `java.io.InputStream` 对象。保存在 `assets` 目录中的资源文件均可使用 `android.content.res.AssetManager` 来访问它们, 存储在 `assets` 目录的文件没有资源 ID。

使用 XML 访问资源

可以用各种 XML 标签特性来引用现有资源。例如, 通常会引用字符串和图像(即绘图)资源为在布局文件中指定的各种控件提供文本和图像。当从 XML 上下文中引用另一个资源时, 通常采用如下方式:

```
@resource type/resource name
```

@表示引用现有的资源。斜杠分隔了资源类型和资源名称, 其含义同前述。例如, `@string/cancel` 引用了 `cancel` 资源成员名称, 其资源类型为 `string`, 该资源通常位于 `res/values` 目录的 `strings.xml` 文件中。

你可能想要在多个设备配置中使用相同的资源, 并且不想将该资源作为默认资源。不必在多个替代资源目录中存储资源, 你可以(在某些情况下)创建一个可选资源, 并将其作为保存在默认的资源目录中的资源的别名。

例如, 如果需要在不同地区使用特定的应用程序图标(存储在 `icon.png` 中), 但在英裔加拿大和法裔加拿大地区需要使用相同的图标版本, 不必将同样的图像文件复制到 `res/drawable-en-rCA` 和 `res/drawable-fr-rCA` 目录中, 使用下面的方法即可:

(1) 将两个地区共同使用的图像保存为 `icon_ca.png`(不使用 `icon.png`)并放置到 `res/drawable` 目录中。

(2) 创建一个 `icon.xml` 文件, 该文件的 `<bitmap>` 标签引用 `icon_ca.png`(例如 `<bitmap xmlns:android="http://schemas.android.com/apk/res/android" android:src="@drawable/icon_ca"/>`), 并将文件保存在 `res/drawable-en-rCA` 和 `res/drawable-fr-rCA` 目录中。当 `icon.xml` 被保存在可替换资源目录时, 例如 `res/drawable-en-rCA` 目录, Android 将它编译成资源, 该资源即可通过代码方式 `R.drawable.icon` 来引用, 也可通过 XML 文件使用 `@drawable/icon` 进行访问。然而, 它实际上是 `R.drawable.icon_ca`(保存在 `res/drawable`)或 `@drawable/icon_ca` 的别名。

访问平台资源

在代码中通过完整的限定包名来访问平台资源, 如 `android.R.layout.simple_list_item_1`

(它是 `android.widget.ListView` 列表视图条目的布局)。通过 `android` 包名访问 XML，如 `@android:color/white`(XML 相当于 `android.R.color.white`)。

3. 资源和用户界面

你在前述章节中学习了通过实例化控件来创建 `Activity` 的用户界面。然而，创建用户界面的更好方式是将其声明在一个或多个 XML 文件中，该方法维护简单且可以更容易地将用户界面应用到不同的设备和地区环境中。下面的 `onCreate(Bundle)` 方法使用资源方式创建了一个用户界面，该界面包含 `Edittext` 和 `Button` 控件：

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

在调用父类同名的 `onCreate` 方法后，`onCreate(Bundle)` 执行了 `setContentView(R.layout.main)` 方法，并将资源 ID (`R.layout.main`) 传递给 `Activity` 的 `setContentView(int layoutResID)` 方法。

`setContentView(int)` 将 `R.layout.main` 标识的布局资源填充(将 XML 转换到一个视图层次)为 `Activity` 用户界面的视图对象层次结构。该资源存储在名为 `main.xml` 的文件中，竖屏时该文件位于 `res/layout` 目录中，横屏时位于 `res/layout-land` 目录中。

`main.xml` 文件声明描述了 `Edittext` 和 `Button` 控件，以及它们的线性布局容器。下面的代码片段显示该文件的内容(省略了前面的 `<?xml version="1.0" encoding="utf-8"?>` XML)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:id="@+id/et"
        android:ems="10"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/btnOK"
        android:onClick="doClickOk"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ok" />
</LinearLayout>
```

`<LinearLayout>` 元素中加入了 `<EditText>` 和 `<Button>` 标签，它们是 `<LinearLayout>` 的子元素，会随同 `<LinearLayout>` 一起被填充。

每个 `<LnearLayout>` 的 `android:layout_width` 和 `android:layout_height` 特性均被设置为 `fill_parent`，以便该布局容器能够占据 `Activity` 的整个屏幕。

注意：

`fill_parent` 特性值代表视图与其容器(减去 `padding`)具有相同的大小，实际上，视图将在容器内部进行扩展以占用更多的空间。从 API Level 8 开始，推荐使用 `match_parent` 特性，

而 `fill_parent` 特性并不建议使用，二者具有相同的含义。

`<EditText>` 元素具有一个 `android:ems` 元素，其含义与本章前述的 `setEms(int)` 方法相类似，该元素被分配 10 ems 大小。`<EditText>` 的 `android:layout_width` 和 `android:layout_height` 特性被设置为 `wrap_content`，可以确保这个控件以理想的大小进行显示。

注意：

`wrap_content` 特性表示视图的大小会根据内容的大小进行匹配。也就是说视图会以其希望的(原始)大小被显示，而希望的大小是根据视图的设置来确定的(例如，`EditText` 控件尺寸为 10 ems)。

`<Button>` 标签提供了类似的 `android:layout_width` 和 `android:layout_height` 特性，确保该控件以希望的大小进行显示。`android:text` 特性引用了一个字符串资源来作为按钮的显示文本(`<string name="ok">OK</string>`)，该资源很可能是在 `strings.xml` 文件中声明的。

提示：

应当避免在代码和布局资源中使用硬编码字符串，应将它们作为独立的资源条目存储在 `strings.xml` 文件中，这样可使应用程序更容易实现本地化。

`<Button>` 还提供 `onClick` 特性来引用 `doClickOk` 方法，该方法的返回值为 `void` 类型，唯一的一个参数的类型为 `View`。当按钮被单击时，`doClickOk` 方法被调用(不必实例化侦听器类和注册按钮实例)，`void doClickOk(View view)` 方法的定义如下：

```
public void doClickOk(View view)
{
    EditText et = (EditText) findViewById(R.id.et);
    Toast.makeText(Test.this, et.getText(),
        Toast.LENGTH_SHORT).show();
}
```

`doClickOk(View)` 调用了 `findViewById(R.id.et)` 方法，并将 `EditText` 的资源 ID `R.id.et` 传递给 `Activity` 类的 `View findViewById(int id)` 方法。`findViewById(int)` 将资源转换为 `EditText` 类型对象，并将其赋给变量 `et`(必须进行 `EditText` 类型转换)。

注意：

当 `findViewById(int)` 无法找到资源时会返回 `null` 值。

最后，`doClickOk(View)` 通过 `toast` 显示 `EditText` 的内容。

警告：

在调用 `findViewById(int)` 方法之前，必须在某处调用 `setContentView(int)` 方法。否则，`Android` 将显示一条消息，提示应用程序已经被终止，其原因是由于没有添加布局资源，从而导致 `Android` 无法在 XML 文件中找到 `EditText` 和 `Button` 控件。

1.4.3 Manifest 文件

通过检查应用程序的 XML 结构化的清单文件 `AndroidManifest.xml`, Android 可知道应用程序包含的各种组件(或更多信息)。例如, 程序清单 1-7 显示了 manifest 文件如何声明 Activity 组件。

程序清单 1-7 声明了 Activity 组件的 manifest 文件

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

程序清单 1-7 以 `<?xml version = " 1.0 " encoding = " utf - 8 " ?>` 开始, 表示该文件是一个 XML 1.0 版本文件, 其内容是按照 utf - 8 编码标准进行编码的。

接着程序清单 1-7 显示了一个 `<manifest>` 标签, 它是 XML 文档的根元素: 其中 `android` 表示 Android 命名空间, `package` 表示应用程序使用的 Java 包, 而 `versionCode/versionName` 表示版本信息。

嵌套在 `<manifest>` 中的是 `<application>`, 它是应用程序组件标签的父标签。其中 `android:icon` 和 `android:label` 特性引用了图标和标签资源, 它们代表了 Android 设备中的某个应用程序。

注意:

当选择应用程序启动器界面的选项菜单中的“应用程序管理”后(在手机控制器界面单击 MENU, 稍后讨论), `android:label` 特性指定的应用程序名称就会显示在出现的列表中。该特性还为 `<activity>` 标签提供了默认的 `android:label` 特性。

嵌套在 `<application>` 中的是 `<activity>` 标签, 它描述了一个 Activity 组件。这个标记的 `name` 特性表示一个实现 Activity 的类(MyActivity)。该名称以“.”开头, 代表相对于 `com.example.project` 包。

注意:

当 `AndroidManifest.xml` 通过命令行方式被创建时, 没有“.”字符。但如果该文件是使用 Eclipse(详见范例 1-10)创建的, 则会存在“.”字符。不管怎样, MyActivity 都是相对于 `<manifest>` 标签中 `package` 值(`com.example.project`)的。

嵌套在<activity>内部的是<intent-filter>。这个标记通过自封闭标签声明了组件的功能。例如，通过<action>和<category>标签声明 Activity 组件的功能：

- <action>代表执行的动作，该动作名以字符串形式存放在 android:name 特性中。字符串“android.intent.action.MAIN”的含义为该 Activity 会作为第一个 Activity 启动，并将其初始化为没有数据输入和输出的对象。想要启动应用程序，Android 将查找具有<intent-filter>的<activity>标签，并且其<action>标签的 android:name 为"android.intent.action.MAIN"的 Activity。
- <category>为组件的类型提供了额外的信息，并根据分配给 android:name 特性字符串的内容来处理 Intent 对象。"android.intent.category.LAUNCHER"字符串代表 Activity 可以作为应用程序的初始 Activity 对象，并且将出现在应用程序启动器屏幕中，并按其标签名称排序。

其他组件声明方法类似：服务使用<service>标记、广播接收器使用<receiver>标记、内容提供者使用<provider>标记。Android 不会创建没有在 manifest 文件中声明的组件。

注意：

运行时才创建的广播接收器，不必在 manifest 文件中进行声明。

manifest 文件也可包含<uses-permission>标记，以标识应用程序需要的许可权限。例如，应用程序如果要使用摄像头，则需要指定以下标记：<uses-permission android:name="android.permission.CAMERA" />。

注意：

<uses-permission>内嵌在<manifest>标记中，与<application>标签是同级的。

在应用程序安装时，应用程序的许可权限(使用<uses-permission>标记)是由 Android 包安装器授予的，它会检查应用程序声明的权限来检查相应的数字签名，此过程可能需要与用户进行交互。

当应用程序运行时，不再与用户进行权限验证。应用程序安装后，即可获得所期望功能的许可权限，如果没有通过授权，任何试图使用该功能的请求均告失败，且不会给用户任何消息提示。

注意：

AndroidManifest.xml 文件还提供了其他的信息，例如，指定应用程序所链接的库(除了默认的 Android 库)，并将应用程序所有的执行权限(通过<permission>标签)授予其他应用程序，例如控制谁可以启动应用程序的 Activity 对象。

另一个 Manifest 文件示例

程序清单 1-8 给出了一个 AndroidManifest.xml 文件，该文件将程序清单 1-1 中的 SimpleActivity 类和随后提到的 SimpleActivity2 类，作为应用 SimpleActivity 的两个组件，省略号表示与当前讨论无关的内容。

程序清单 1-8 SimpleActivity 的 Manifest 文件

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.project" ...>
    <application ...>
        <activity android:name=".SimpleActivity" ...>
            <intent-filter ...>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SimpleActivity2" ...>
            <intent-filter ...>
                <action android:name="android.intent.action.VIEW" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        ...
    </application>
</manifest>

```

程序清单 1-8 显示了 SimpleActivity 和 SimpleActivity2 通过内嵌在<activity>中的<intent-filter>标签与 Intent 过滤器建立了联系。当 Intent 对象的值与下列标签值相匹配时，SimpleActivity2 的<intent-filter>标签可帮助 Android 确定如何启动 Activity：

- <action>的 android:name 特性被设置为“android.intent.action.VIEW”
- <data>的 android:mimeType 特性被设置为“image/jpeg”MIME 类型。通常还需要提供附加特性(如 android:path)以确定要查看的数据。
- <category>的 android:name 特性被设置为 android.intent.category.DEFAULT”。该特性允许你不必显式指定它的组件就可以启动 Activity。

根据上述信息，下面的示例显示了如何隐式启动 SimpleActivity2：

```

Intent intent = new Intent();
intent.setAction("android.intent.action.VIEW");
intent.setType("image/jpeg");
intent.addCategory("android.intent.category.DEFAULT");
SimpleActivity.this.startActivity(intent);

```

前 4 行创建一个隐式 Intent 对象。传递 Intent 对象的 Intent setAction(String action)、Intent setType(String type)和 Intent addCategory(String category)方法的值分别指定了 Intent 对象的 Action、MIME 类型和 category。Android 利用它们将 SimpleActivity2 确定为待启动的 Activity 对象。

程序清单 1-2 显示了 SimpleService 类。可使用以下内容来扩展程序清单 1-8，以便可在应用程序中访问该类：

```
<service android:name=".SimpleService">
</service>
```

程序清单 1-5 显示了 SimpleBroadcastReceiver 类。可使用以下内容来扩展程序清单 1-8, 除非广播接收器是在运行时被创建的:

```
<receiver android:name=".SimpleBroadcastReceiver">
</receiver>
```

最后, 程序清单 1-6 显示了 SimpleContentProvider 类。可使用以下内容来扩展程序清单 1-8, 以便可在应用程序中访问该类:

```
<provider android:name=".SimpleContentProvider">
</provider>
```

1.4.4 应用程序包

Android 应用程序是用 Java 语言编写的。应用程序组件的 Java 代码被编译后, 将会继续转换为 DEX 格式。生成的代码文件和其他必要的资源随后被合并到一个应用程序包(APK)中, 并保存为后缀名为.apk 的压缩文件。

APK 并不是应用程序, 但可用于发布并将其安装在移动设备中。APK 不是应用程序, 是因为 APK 的组件可能会复用其他 APK 的组件, 并且(这种情况下)并不是所有应用程序都会驻留在 APK 文件中, 因此 APK 本身并不是应用程序。然而, 通常情况下, APK 就是指应用程序。

APK 文件必须签署证书(用于标识应用程序的作者), 其私钥属于它的开发者所有。该证书不需要由证书授权机构来签署。相反, Android 允许 APK 文件使用自签名证书进行签名, 这种情况很常见(APK 签名将在范例 1-8 中讨论)。

APK 文件、用户 ID 和安全

每一个安装在 Android 设备上的 APK 将被赋予唯一的 Linux 用户 ID, 只要 APK 文件驻留在该设备中, 用户 ID 将保持不变。由于安全检测发生在进程中, 并且每个 APK 的代码需要运行给不同的 Linux 用户, 因此任意两个 APK 文件中的代码通常情况下不能运行在相同的进程中。然而, 你可以给 AndroidManifest.xml 文件中<manifest>标记的 sharedUserId 特性分配相同的用户 ID 名, 使两个 APK 文件中的代码运行在相同的进程中。分配完成后, Android 将会知道这两个包属于同一个应用程序, 并具有相同的用户 ID 和文件权限。此外, 为了保持安全性, 两个同样签名的 APK 文件(在它们的 Manifest 文件中具有相同的 sharedUserId 特性值)才会获得相同的用户 ID。

1.4.5 安装 Android SDK

1. 问题

你已经阅读了上述关于 Android 的介绍, 你希望开发自己的第一个 Android 应用程序。

但是,在你开发应用程序之前必须先安装 Android SDK 工具包。

2. 解决方案

Google 为 Windows 操作系统、基于英特尔处理器的 Mac OS X 操作系统、基于 i386 的 Linux 操作系统提供了最新版本的 Android SDK 发布文件。根据不同的使用平台下载并解压相应的文件,并把解压后的主目录移到一个相对方便的位置。你可能还需要更新 PATH 环境变量,以便可以在文件系统的任何地方访问 SDK 的命令行工具。

在下载和安装这个文件之前,你必须了解 SDK 需求情况。如果你的开发平台不满足一些要求,将无法使用 SDK。

Android SDK 支持以下操作系统:

- Windows XP(32 位)、Vista(32 位或 64 位)或 Windows 7(32 位或 64 位)
- Mac OS X 10.5.8 或更高版本(仅 x86)
- Linux(在 Ubuntu Linux、Lucid Lynx 上测试过的)。GNU C 库(glibc)2.7 或更高版本。Ubuntu Linux 8.04 或更新版本。64 位发布系统必须能够运行 32 位应用程序。在 Ubuntu Linux 安装笔记 <http://developer.android.com/sdk/installing/index.html#Troubleshooting> 中,可了解如何添加对 32 位应用程序的支持。

你很快会发现 Android SDK 是以各种可单独下载的组件来组织的,它们被称为软件包。你需要确保有足够的磁盘存储空间来容纳各种安装包。空闲存储空间需要 2GB,该数值综合考虑了 Android API 文档和多个 Android 平台支持的需要(也称为 Android 软件栈)。

最后,还需要确认一下已经安装了如下软件:

- JDK 6 或 JDK 7: 你需要安装 Java 开发工具(JDK)来编译 Java 代码。仅仅安装 Java 运行时环境(JRE)是不够的(在发布模式下创建应用程序时, JDK 7 会有点问题, 范例 1-8 展示了该问题和它的解决方案)。
- Apache Ant: 你需要安装 Ant 1.8 或更高版本来构建 Android 项目。

注意:

如果你的开发平台已经安装了 JDK,花点时间确认一下它是否满足前面列出的版本要求(6 或 7)。一些 Linux 发布系统可能会包含并不支持 Android 开发的 JDK 1.4。此外,GNU 的 Java 编译器也不支持 Android 开发。

3. 实现机制

打开浏览器,导航到 <http://developer.android.com/sdk/index.html>, 下载 android-sdk_r20-windows.zip(Windows)、androidsdk_r20-macosx.zip(Mac OS X)或 android-sdk_r20-linux.tgz (Linux)中一个压缩包,它们都是 Android SDK Release 20 的发布版本(Release 20 是撰写本书时的最新版本)。

注意:

Windows 开发者可以选择下载并运行安装程序 installer_r20-windows.exe。这个工具可自动完成大多数的安装过程。

例如, 如果使用 Windows 操作系统(本章默认平台), 你可以选择下载 `android-sdk_r20-windows.zip` 文件。解压该文件后, 可将解压后的 `android-sdk-windows` 主目录移到文件系统中相对方便的位置。例如, 你可将 `C:\unzipped\android-sdk_r20-windows\android-sdk-windows` 主目录移到 C 盘根目录, 即目录 `C:\android-sdk-windows`。

注意:

建议将 `android-sdk-windows` 重命名为 `android`, 以避免在 Eclipse 中运行应用程序时, 出现模拟器崩溃的可能性。虽然这个问题可能并不存在, 但它的确在过去出现过, 该问题最可能的原因是在 `android` 和 `sdk` 之间, 以及在 `sdk` 和 `windows` 之间使用了连字符“-”。

想要完成安装, 需要添加 `tool` 子目录到 `PATH` 环境变量中, 以便可在文件系统的任何地方访问 SDK 的命令行工具。

接下来, 查看 `android-sdk-windows`(或 `android`)主目录, 该目录包含以下的子目录和文件:

- **add-ons:** 初始为空目录, 该目录用来保存 Google 和其他厂商提供的插件(应用程序目标核心平台以外的其他 SDK)。例如, Google API 插件存储在该目录中。
- **platforms:** 初始为空目录, 该目录用来保存 Android 的平台信息。例如, Android 4.1 存储在 `platforms` 的一个子目录中, 而 Android 2.3.4 将存储在另一个 `platforms` 子目录中。
- **tools:** 该目录包含一组平台无关的开发工具, 例如模拟器等。目录中的工具(也称为基本工具), 可以在任何时间被更新, 并且和 Android 平台无关。
- **AVD Manager.exe:** 该工具用于管理 Android 虚拟设备(Android Virtual Device, AVD)(使用 Android 模拟器运行的设备配置)。
- **SDK Manager.exe:** 该工具是用来管理 SDK 软件包, 并运行 AVD Manager 来响应菜单选择。
- **SDK Readme.txt:** 该文本文件的内容用于显示欢迎使用 Android SDK, 并告诉用户想要开始开发应用程序, 需要使用 SDK Manager 安装平台工具, 并至少安装 Android 平台。

`tools` 目录包含各种有用的基本工具, 包括:

- **android:** 用于创建和更新 Android 项目; 使用新 Android 或其他平台更新 Android SDK; 创建、删除和查看 AVD。
- **emulator:** 用于在内核中运行完整的 Android 软件堆, 它包含很多预先安装且可访问的应用程序(如浏览器)。
- **hierarchyviewer:** 提供了可视化布局的视图层次结构(Layout View)和放大显示的视图(Pixel Perfect 视图), 以便可以调试和优化 Activity 屏幕。
- **sqlite3:** 管理由 Android 应用程序创建的 SQLite 数据库。
- **zipalign:** 用来对 APK 文件进行压缩对齐优化。

附录 B 描述了 SDK 的所有基本工具。

1.4.6 安装 Android 平台

1. 问题

安装 Android SDK 对于开发 Android 应用程序是不够的，你还必须至少安装 Android 平台。

2. 解决方案

使用 SDK Manager 工具安装 Android 平台。如果 SDK Manager 无法显示 Android SDK Manager 对话框，你可能需要创建 JAVA_HOME 环境变量来指向 JDK 的主目录(例如，设置 JAVA_HOME=C:\Program Files\Java\jdk1.7.0_04)。然后再重试一次。

此外，你还可使用 android 工具安装 Android 平台。如果 Android 系统显示“Failed to convert path to a short DOS path:C:\Windows\system32\java.exe”，需要找到 find_java.bat 文件(查看 C:\android\tools\lib\find_java.bat)，并从下面每行中删除“-s”。

```
for /f %a in ('%~dps0\find_java.exe -s') do set java_exe=%a
for /f %a in ('%~dps0\find_java.exe -s -w') do set javaw_exe=%a
```

3. 实现机制

运行 SDK Manager 或 android 工具，两种工具均弹出 Android SDK Manager 对话框，如图 1-7 所示。

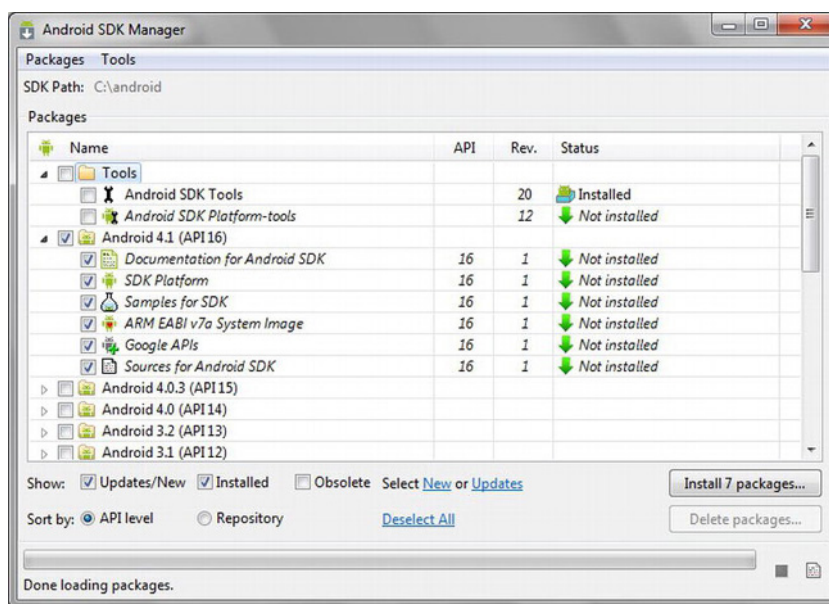


图 1-7 使用此对话框安装、更新和删除 Android 包，并访问 AVD 管理器

Android SDK Manager 提供了一个菜单栏和一块内容区域。菜单栏包括 Packages 和 Tools 菜单：

- **Packages:** 使用此菜单显示更新/新建包、安装包和废弃包；显示(或不显示)压缩详情信息；按照 API Level 或资源库对包进行排序；重新加载内容区域显示的包列表。
- **Tools:** 使用此菜单管理 AVD 和插件网站；指定代理服务器和其他选项，并显示 About 对话框。

内容区域可显示 SDK 的路径、包信息表、用来选择要显示包的复选框、将包按照 API Level 或资源库进行排序的单选按钮、安装或删除包的按钮、显示扫描包资源库进度信息的进度条。

Packages 表信息将包分为工具、特定的 Android 平台和 extras。每一类都带有一个复选框，当处于选中状态时，代表选择了该类所有的条目，而去掉复选框的选中即可取消条目的选定。

工具分为 SDK 工具和 SDK 平台工具：

- SDK 工具是基本工具，包含在 SDK 发布文件中，并存储在 tools 目录下，状态栏中关于 Android SDK 工具条目的 Installed 信息刚好可证实这一点。
- SDK 平台工具是与平台相关的应用程序开发工具。这些工具支持最新的 Android 平台特征，通常只有当新平台可用时，才进行更新。它们与旧平台保持向下兼容，但当安装新平台时，必须确保有这些工具的最新版本。如果不选中 Android SDK Platform 的工具条目(默认为不选中)，平台工具将会自动进行安装。

本书仅需要安装 Android 4.1 (Level 16)平台。该类别和它的所有条目均应选中，以便保留它们。除了这个平台外，你还需安装文档、示例、ARM 系统镜像(处理器架构模拟，x86 是另一个示例，但在撰写本文时它并不支持 Android 4.1)、Google API 和源代码。

最后，你可以安装 extras，它是构建应用程序时所使用的库或工具。例如，Google 的 USB 驱动程序项目在 extras 中已被选中。但是，你只有在 Windows 平台才需要安装该组件并且在实际的 Android 设备中测试你的应用程序。

单击 Install 7 packages 按钮(根据选择的安装包的多少，数值会有所不同)，你会看到 Choose Packages to Install 对话框，如图 1-8 所示。

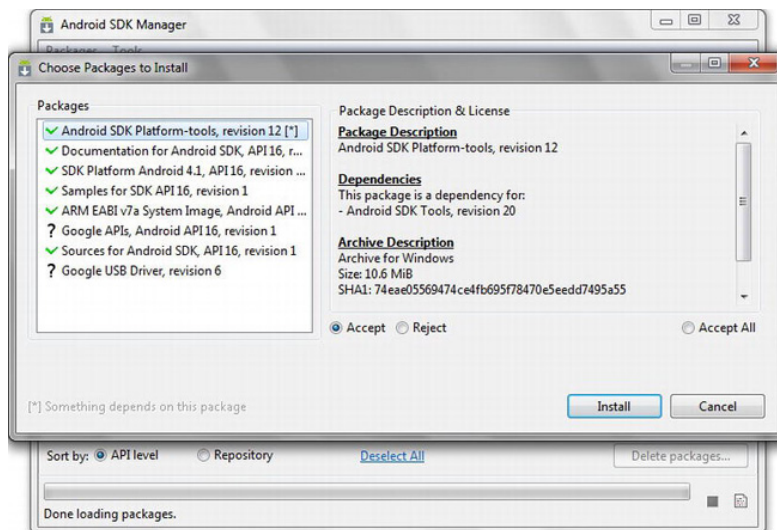


图 1-8 Packages 列表框中显示了可以安装的包

Choose Packages to Install 对话框显示了一个软件包列表，用于标识可安装的包。在包名的旁边显示绿色选中标记，代表可以安装的包，而没有被选择的包会在其包名的旁边显示问号标记。

注意：

尽管 Google API 和 Google 的 USB 驱动程序最初就被选中，但仍然显示没有选择它们(也许这是一个错误，信息没有被正确传递)。如果还想安装这些包，你需要高亮显示并选择它们。

对于高亮选择的包，Package Description & License 中显示了包的描述、依赖于当前安装包的其他包列表、存储包的归档信息和附加信息。单击 Accept 或 Reject 单选按钮来选择和拒绝安装包。

注意：

当你拒绝安装某个包时，在 Packages 列表框中该包名称的旁边会出现红色的 X。单击 Accept All 单选按钮可接受所有的包。

在某些情况下，一个 SDK 组件可能需要另一个组件或 SDK 工具的最低修订版。除了 Package Description & License 记录它们的相关性外，当需要处理依赖关系时，开发工具将会通过调试警告来通知你。

单击 Install 按钮开始安装，Android 系统将下载并安装已选择的包。你也将看到 Android SDK Manager Log 对话框，该对话框显示安装的状态。该对话框如图 1-9 所示。

注意图 1-9 中出现的“Stopping ADB server failed (code -1)”信息。ADB 代表 Android Debug Bridge，它是一种包括客户端和服务端程序的工具，以便控制和实现与 Android 设备的接口。出现这个信息是因为 ADB 服务器目前没在运行(此时它不必运行)。

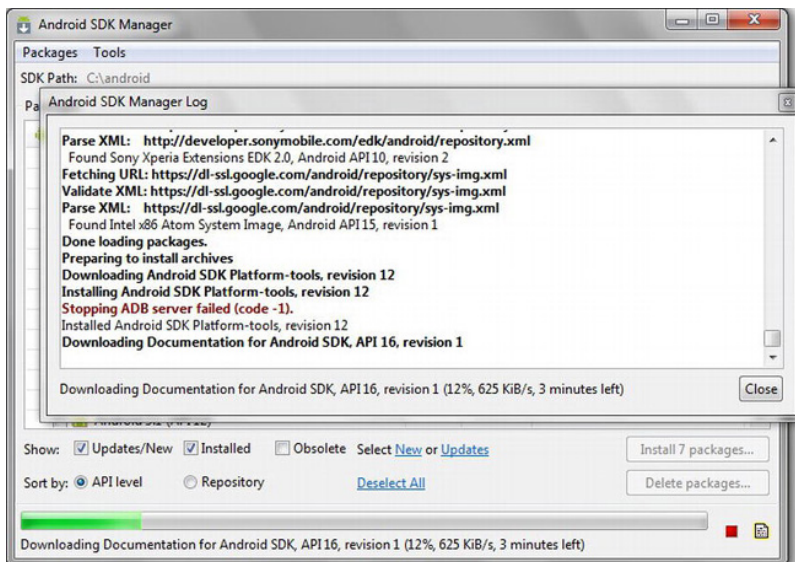


图 1-9 日志窗口显示了每个选定包下载和安装的进度

安装完成后，你应该在 Android SDK Manager Log 和 Android SDK Manager 对话框的

底部看到“Done loading packages”消息。在前面的对话框中单击 Close 按钮；而后面对话框中 Packages 表内的状态列将告诉你哪些包已经安装。

你还应该看到主目录下几个新的子目录，如下所示：

- platform-tools (位于 android 目录内)
- android-16 (位于 android/platforms 目录内)

platform-tools 子目录包含最新的平台工具——参见附录 B。android-16 子目录包含 Android 4.1 系统特定的文件。

提示：

可将平台工具添加到 PATH 环境变量中，以便在文件系统的任何地方都访问这些工具。

1.4.7 创建 Android 虚拟设备

1. 问题

安装 Android SDK 和 Android 平台后，你准备开始创建 Android 应用程序。然而，需要先创建一个 Android 虚拟设备(AVD)，否则无法通过模拟器工具来运行这些应用程序。AVD 是设备配置，用来代表 Android 设备。

2. 解决方案

使用 AVD Manager 或 android 工具创建 AVD。

3. 实现机制

运行 AVD Manager (或从 Android SDK Manager 对话框的 Tools 菜单中选择 Manage AVDs)。图 1-10 显示了 Android Virtual Device Manager 对话框。

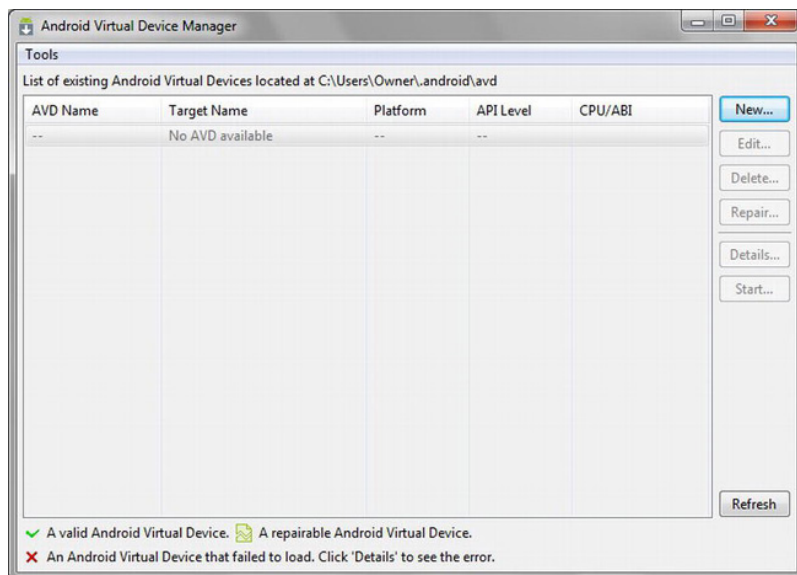


图 1-10 没有可用的 AVD 设备

单击 New 按钮，图 1-11 显示了 Create new Android Virtual Device (AVD)对话框。

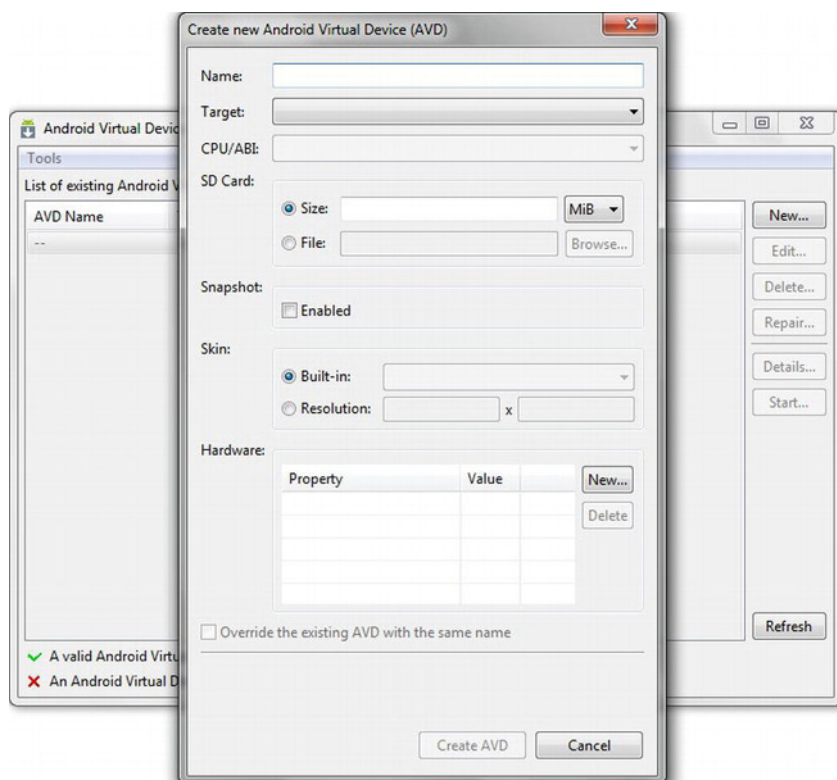


图 1-11 AVD 设备包含有名称、目标 Android 平台等

图 1-11 显示 AVD 设备具有名称 name、特定的 Android 目标平台、CPU/应用程序的二进制接口(如 ARM/armeabi-v7a)、可模拟 SD 卡、特定屏幕分辨率的外观皮肤以及各种不同的硬件属性。

输入名称 AVD1，目标平台选择为 Android 4.1 – API Level 16(这应该是唯一的选择)，然后将 SD 卡的 Size 大小设置为 100。

注意：

可以选中 Snapshot 下的复选框来保存模拟器在执行过程的工作状态，以便在第一次启动模拟器后能够快速重启。

选择 Android 4.1 – API Level 16 后，皮肤会被选择为 Default(WVGA800)。此外，硬件属性表显示的内容为：LCD 密度属性值为 240 点/英寸、最大 VM 应用程序堆大小属性为 48MB；设备内存大小为 512MB。

注意：

为了模拟平板电脑，外观皮肤应该选择 WXGA720、WXGA800 或 WXGA800-7in，而其他值用于模拟手机设备(本章的重点)。

属性表格右侧的 New 按钮可以显示更多的硬件属性。例如，当使用 Android 4.0 或更

高版本时,你可以选择模拟图形处理单元(GPU),它会增加颜色深度和减少图像伪影,设置方法如下:

- (1) 单击 New 按钮。
- (2) 从对话框选择 GPU 模拟,并单击 Ok 关闭该对话框。
- (3) 将 GPU 模拟默认值由 no 改为 yes。

输入名称 AVD1 后,选择 Android 4.1 – API Level 16,保持 WVGA800 作为手机默认皮肤(或选择另一个手机皮肤,例如 HVGA,它会出现一个更小的模拟器窗口,当屏幕分辨率是 1024×768、LCD 密度改为 160 时,该窗口与屏幕恰好适合),GPU 模拟属性值选 yes,单击 Create AVD 完成 AVD 设备的创建。图 1-10 中的 AVD 窗格中现在出现了一个 AVD1 条目。

注意:

当创建 AVD 设备以测试编译后的应用程序时,应确保目标平台的 API Level 要大于或等于应用程序所需的 API Level。换言之,如果打算在 AVD 设备中测试应用程序,应用程序通常无法访问某些平台 API,因为这些 API Level 比 AVD 所支持的 API Level 要新。

虽然使用 AVD Manager 创建 AVD 更加容易,但是也可利用 android 工具完成该任务,方法为: `android create avd -n name -t targetID [-option value]...`。在该语法中, name 标识了设备配置名(如 target_AVD); targetID 是整型 ID,标识 Android 的目标平台(执行 `android list targets` 可获得 ID); [-option value]...代表一系列的选项(如 SD 卡的大小)。

如果没有指定足够的选项, android 将提示创建自定义硬件配置文件。如果不想创建自定义硬件配置文件,而希望使用默认的硬件模拟选项时,请按回车键。例如, `android create avd -n AVD1 -t 1` 命令行将创建名为 AVD1 的 AVD 设备。该命令行假设 1 对应于 Android 4.1 平台并且提示创建一个自定义的硬件配置文件。

注意:

每个 AVD 作为一台独立设备,拥有自己的私有空间来存储用户数据、自己的 SD 卡等等。当使用 AVD 启动模拟器时,该工具会从 AVD 的目录中加载用户数据和 SD 卡数据。默认情况下,模拟器会保存用户数据、SD 卡数据和分配给 AVD 的目录中的缓存数据。

1.4.8 启动 AVD

1. 问题

你必须启动模拟器和 AVD,以便安装并运行应用程序。
你想知道如何完成这项任务。

2. 解决方案

使用 AVD Manager 启动 AVD,或使用 emulator 工具启动 AVD。

3. 实现机制

参见图 1-10, 你会看到一个灰色的 Start 按钮。该按钮只在创建了一个 AVD 条目(高亮显示)后, 才会处于可用状态。单击 Start 按钮运行模拟器工具, 并将高亮显示的 AVD 条目作为模拟器的设备配置。

此时会显示 Launch Options 对话框。该对话框指定了 AVD 的皮肤和屏幕密度。它还提供了未选中的复选框来调整模拟器的显示分辨率来匹配设备物理屏幕的大小; 是否擦除用户数据的复选框; 是否从先前保存快照中启动的复选框; 以及在设备退出时是否将设备状态保存到快照中的复选框。

注意:

当更新应用程序时, 你需要定期打包并将它们安装在模拟器中, 以便在用户数据磁盘分区中保留重启 AVD 所需的应用程序和它们的状态数据。为了确保更新后应用程序能正常运行, 你可能需要删除模拟器的用户数据磁盘分区, 可通过选中 Wipe user data 实现。

单击 Launch 按钮启动 AVD1 模拟器。AVD Manager 会短暂地显示 Starting Android Emulator 对话框, 然后会显示模拟器窗口。参见图 1-12。

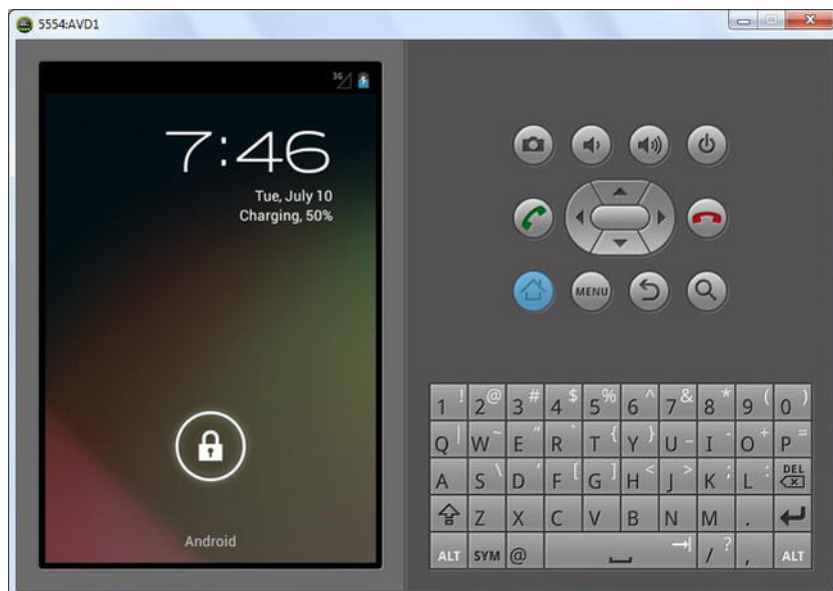


图 1-12 模拟器窗口(具有 HVGA 手机外观皮肤)左侧为主屏幕, 右侧为手机控制和键盘

图 1-12 显示的模拟器窗口分为两个窗格, 左窗格在黑色背景显示 Android 标志, 其后即为主屏幕。而右窗格用于显示手机控制和键盘。

状态栏出现主屏幕(每个应用程序屏幕)的顶部。状态栏显示当前时间、电池剩余的电量和其他信息, 此外它还能访问通知消息。

主屏幕初始化为锁屏模式。想要解锁屏幕, 向右拖动锁图标, 直到它触及解锁图标(或按下菜单按钮)。解锁后的主屏如图 1-13 所示。



图 1-13 主屏幕当前显示了应用程序启动器和其他内容

主屏幕显示内容如下：

- 墙纸背景：墙纸位于元素的后面，可以左右拖动。想要改变这个背景，在墙纸上按住鼠标左键，之后会弹出一个可以操作墙纸的菜单。
- Widget(小部件)：Google Search 小部件出现在顶部附近，Clock 小部件出现在中上部，Camera 小部件出现在左下角。Widget(小部件)是一个小型应用程序视图，能够嵌入到主屏幕或其他应用程序中，并接收定期更新。
- 应用程序启动器：应用程序启动器为常用的 Browser、Contacts、Messaging 和 Phone 应用程序提供了启动图标(屏幕底部)；它还有一个包含所有已安装应用程序的矩形网格，单击这些应用程序的图标即可启动它们。图 1-14 显示了一些图标。

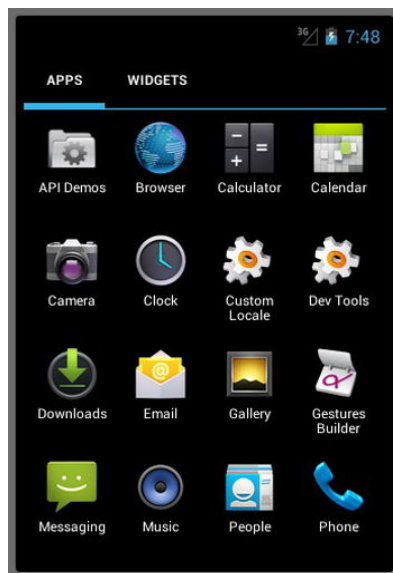


图 1-14 向左拖动此屏幕可以显示更多的图标

应用程序启动器根据屏幕左上角附近的选项卡来组织应用程序和部件。你可以在 APPS 选项卡中运行应用程序，或从 WIDGETS 选项卡中选择在主屏幕上显示的其他小部件(如果在主屏幕上需要更多空间才能显示小部件，可左右拖动墙纸)。

提示：

API 演示应用程序演示了丰富的 Android API 应用。如果你是 Android 应用程序开发的初学者，你应该运行每个演示程序，以便熟悉 Android 所能提供的内容。你可通过访问位于 `android/samples/android-16/ApiDemos` 文件夹中的源文件，来查看每个演示程序的源代码。

手机控制包括以下常用的按钮：

- 房子图标按钮可让你从任何地方直接返回到主屏幕。
- MENU 按钮会显示一个菜单，可在当前正在运行的应用程序进行特定的选择。
- 弧形箭头图标按钮可使你在 Activity 堆栈中返回到前一个 Activity 对象。

当 AVD 处于运行状态时，使用鼠标“触摸”触摸屏和使用键盘“按下”AVD 按键，可实现与 AVD 的交互操作。表 1-2 显示了 AVD 键和 键盘按键的映射关系。

表 1-2 AVD 键和键盘按键之间的映射

AVD 按键	键盘按键
Home	HOME
Menu (left softkey)	F2 或 Page Up
Star (right softkey)	Shift+F2 或 Page Down
Back	ESC
拨号按键	F3
挂机按键	F4
搜索	F5
电源按键	F7
增加音量按键	KEYPAD_PLUS, Ctrl+5
降低音量按键	KEYPAD_MINUS, Ctrl+F6
照相机按键	Ctrl+KEYPAD_5, Ctrl+F3
返回上一个布局定位 (例如，横向或纵向)	KEYPAD_7, Ctrl+F11
返回下一个布局定位	KEYPAD_9, Ctrl+F12
打开/关闭网络	F8
代码分析切换	F9 (仅使用 -trace 启动选项)
全屏切换	Alt+Enter
轨迹球方式切换	F6
临时进入轨迹球方式(按键被按下时)	Delete
导航左/上/右/下键	KEYPAD_4/8/6/2
导航中键	KEYPAD_5
Onion 皮肤的透明度	KEYPAD_MULTIPLY(*) /
增/减	KEYPAD_DIVIDE(/)

提示:

必须在开发计算机中先禁用 NumLock 键, 才能使用小键盘的按键。

表 1-2 中的 -trace 启动选项代表在上下文中触发代码分析功能。当使用 emulator 工具启动 AVD 时, 这个选项允许将分析结果存储在一个文件中。

例如, emulator -avd AVD1 -trace results.txt 启动 AVD1 配置模拟器, 当按 F9 时, 它会将分析结果存储在 results.txt 中, 再次按 F9 将停止代码分析。

图 1-12 在标题栏上显示 5554:AVD1, 数值 5554 标识了一个控制台端口, 你可以使用该端口进行动态查询, 也可以用于控制 AVD 环境。

注意:

Android 支持同时执行多达 16 个 AVD。每个 AVD 被分配一个偶数控制台端口号, 并从 5554 开始计数。

可使用 telnet localhostconsole-port 命令连接到 AVD 的控制台。例如, telnet localhost 5554 将连接到 AVD1 的控制台。图 1-15 显示了在 Windows 7 中生成的命令窗口。

```

C:\> Telnet localhost

Android Console: type 'help' for a list of commands
OK
help
Android console command help:

  help!h!?      print a list of commands
  event         simulate hardware events
  geo           Geo-location commands
  gsm           GSM related commands
  cdma          CDMA related commands
  kill          kill the emulator instance
  network       manage network settings
  power         power related commands
  quit!exit     quit control session
  redir         manage port redirections
  sms           SMS related commands
  avd           control virtual device execution
  window       manage emulator window
  gemu          QEMU-specific commands
  sensor        manage emulator sensors

try 'help <command>' for command-specific help
OK

```

图 1-15 输入一个命令名可获得该命令的帮助信息

提示:

telnet 命令在 Windows 7 中默认情况下被禁用(以确保操作系统更安全)。想要在 Windows 7 中使用 telnet, 需要启动控制面板, 单击 Programs and Features, 选择 Turn Windows features on or off(从 Windows Features 功能对话框中选择), 并选中 Telnet Client 复选框。

1.4.9 Univerter 简介

1. 问题

现在你已经安装了 Android SDK、一个 Android 平台, 还新建和启动了一个 AVD, 现在可以创建并在 AVD 上安装和运行应用了。当然, 你可以利用程序清单 1-1 中的 SimpleActivity

类创建一个应用, 不过本攻略中介绍的 Univerter 应用会更有趣(也更有用)。

2. 解决方案

Univerter(Unit Converter 的缩写)是一个用于单位换算的应用程序(支持 Android 2.3.3 和更高版本)。例如, 可以将摄氏温度换算成华氏温度, 将磅数换算成公斤数, 等等。

注意:

Univerter 支持 13 个种类之间的 200 种转换。

Univerter 只有一个 Activity, 它的用户界面包括显示框、16 个按钮、10 个数字和 1 个小数点, 如下:

- +/-: 单击该按钮可输入负值, 只有在转换方式输入负值有意义时(例如, 从摄氏或华氏温度转换), 该按钮才可用。
- CLR(清除): 该按钮用于清除显示框中的内容。
- CAT(种类): 单击该按钮可选择新的转换种类。在新种类中的第一种转换为默认的转换方式。角度 ANGLE 是默认转换种类。
- CON(换算方式): 单击该按钮可在当前种类中选择一种新的换算方式。
- CVT(转换): 单击此按钮, 可根据当前选择的换算方式, 转换显示框中显示的值。再次单击该按钮, 将已转换结果作为新值并按当前的换算方式继续转换。

单击 CAT 或 CON 按钮, 将弹出一个列表选择对话框, 选择后单击 Close 按钮以确认选择。

注意:

当 Univerter 的换算值的绝对值大于 $1.0e+18$ 时, 会出现溢出; 而当换算的绝对值小于 0.00000001 时, 会出现下溢问题, 但数值并不等于 0。

另外, 可单击设备的菜单按钮(如果存在, 当菜单按钮不存在时, 可使用 Android 3.0 或更高的设备的 action bar)来激活一个选项菜单, 在该菜单中可以获得关于 Univerter 和使用该软件的帮助信息。

3. 实现机制

Univerter 包含以下 4 个源代码文件:

- Category.java: 这个源文件声明了 Category 类, 用来描述换算的种类。
- Converter.java: 这个源文件声明了 Conversion 类, 用来描述换算方式。
- Converter.java: 这个源文件声明了转换器接口, 用来调用 solitary 方法以执行换算。
- Univerter.java: 这个源文件声明 Univerter 类, 用来描述 Activity 对象。

Univerter 还包括以下资源文件:

- res/drawable/gradientbg.xml: 这个 XML 文件描述了 Activity 的渐变背景。
- res/drawable-hdpi/ic_launcher.png: 这个 png 图像文件代表高密度屏幕的启动图标。
- res/drawable-ldpi/ic_launcher.png: 这个 png 图像文件代表低密度屏幕的启动图标。

- res/drawable-mdpi/ic_launcher.png: 这个 png 图像文件代表中等密度屏幕的启动图标。
- res/drawable-xhdpi/ic_launcher.png: 这个 png 图像文件代表超高密度的启动图标。
- res/layout/help.xml: 这个 XML 文件描述了帮助对话框竖屏或横屏的布局。
- res/layout/info.xml: 这个 XML 文件描述了信息对话框竖屏或横屏的布局。
- res/layout/list_row.xml: 这个 XML 文件描述了转换对话框中的列表项竖屏或横屏的布局。
- res/layout/main.xml: 这个 XML 文件描述了 Activity 对象的竖屏布局。
- res/layout-land/main.xml: 这个 XML 文件描述了 Activity 对象的横屏布局。
- res/menu/univerter.xml: 这个 XML 文件描述了 Activity 选项菜单的布局。
- res/values/colors.xml: 这个 XML 文件存储 Univerter 使用的颜色。
- res/values/strings.xml: 这个 XML 文件存储 Univerter 所使用的字符串。
- res/values/styles.xml: 这个 XML 文件存储自定义主题, 可缩小 Activity 对象屏幕顶部标题栏文本的大小。

此外, Univerter 还包含 AndroidManifest.xml 文件, 它描述了 Univerter Android 应用程序的信息。

注意:

你可以在附录 D 中查找这些文件, 可从与本书配套的代码中获得它们(见 www.apress.com/9781430246145)。

1.4.10 创建 Univerter

1. 问题

你想使用 Android SDK 创建 Univerter, 但不知道如何做(范例 1-10 显示了如何使用 Eclipse 创建 Univerter)。

2. 解决方案

使用 android 工具创建 Univerter, 然后使用 ant 构建该项目。

3. 实现机制

创建 Univerter 的第一步是使用 android 工具创建一个项目。需要遵守以下语法(为了便于阅读, 分多行显示):

```
android create project --target target_ID
                        --name your_project_name
                        --path /path/to/your/project/project_name
                        --activity your_activity_name
                        --package your_package_namespace
```

--name(或-n)用于指定项目名(如果提供该名称,当你创建应用程序时,它将用于生成的.apk 的文件名),除了--name (或-n)外,以下选项均需要进行设置:

- --target(或-t)选项用于指定应用程序构建的目标。这个目标_ID 值是一个整型值,用于标识 Android 平台,通过执行 `android list targets` 命令可以获得目标_ID 值。如果你只安装了 Android 4.1 平台,该命令将输出由整数 ID 1 标识的 Android 4.1 平台目标。
- --path(或-p)选项用于指定项目目录的位置。如果不存在,将创建该目录。
- --activity(或-a)选项用于指定默认 Activity 类的名称,生成的类文件位于 `/path/to/your/project/project_name/src/your_package_namespace/` 目录中。如果没有指定 --name(或-n),该类名将作为.apk 的文件名。
- --package(或-k)选项用于指定项目包的命名空间,它必须遵循由 Java 语言指定的包规则。

假设使用 Windows 7 平台,并且存在 `C:\prj\dev` 目录,而 Univerter 项目存储在 `C:\prj\dev\Univerter` 中。在文件系统的任何地方(除了根目录外),调用下面的命令(为了便于阅读,分两行显示)来创建 Univerter:

```
android create project -t 1 -p C:\prj\dev\Univerter -a Univerter
                        -k ca.tutortutor.univerter
```

该命令可创建多个目录并将文件添加到一些目录中。它在 `C:\prj\dev\Univerter` 中建立了如下文件和目录结构:

- `AndroidManifest.xml` 是正在创建的应用程序清单文件。该文件和前面通过--activity 或-a 指定的 Activity 保持一致。
- `ant.properties` 属性是 Ant 构建系统的自定义属性文件,可以编辑该文件来覆盖 Ant 默认的设置,在发布模式下创建应用程序时,你还可以通过它指定一个指向密钥库和密钥别名的指示器,以便 build 工具能够为应用程序签名(稍后讨论)。
- `bin` 是 Apache Ant 构建脚本的输出目录。
- `build.xml` 是项目的 Apache Ant 构建脚本。
- `libs` 目录包含私有库(当需要时)。
- `local.properties` 是生成的文件,其中包含 Android SDK 主目录的位置。
- `proguard-project.txt` 包含使用的 ProGuard 的信息。ProGuard 是一个 SDK 工具,可使开发者混淆加密它们的代码(增加反编译的难度),它是构建发布版本时的一个组成部分。
- `project.properties` 是生成的文件,用以确定项目的 Android 目标平台。
- `res` 目录包含项目资源。
- `src` 目录包含项目的源代码。

你需要用本书的代码压缩包中的 `AndroidManifest.xml` 文件替换当前的 `AndroidManifest.xml` 文件。

res 包含如下子目录:

- layout 目录包含布局文件。main.xml 文件存储在该目录中。
- values 目录包含值文件。strings.xml 文件存储在该目录中。

你需要用范例 1-5 所示的资源目录替换该目录。

src 包含 ca\tutortutor\univerter 目录结构, 最终的 univerter 子目录会包含 Univerter.java 源文件。你需要将范例 1-5 提及的 4 个源文件复制到 univerter 目录中。

假设当前目录为 C:\prj\dev\Univerter, 如果使用 Apache Ant 工具构建应用程序, 将默认执行该目录中的 build.xml 文件。在命令行上, 在 ant 后指定 debug 或 release 来设置 build 模式:

- Debug(调试)模式: 构建用于测试和调试的应用。构建工具会为产生的 APK 文件签署调试密钥, 并用 zipalign 对该 APK 进行优化。用 ant debug 命令构建调试模式。
- Release 模式: 构建发布给用户的应用。必须用你自己的私钥签署该 APK, 然后用 zipalign 对 APK 进行优化(将在范例 1-8 中讨论)。用 ant release 命令构建发布模式。

在 C:\prj\dev\Univerter 目录中执行 ant debug 就是以调试模式构建 Univerter。该命令会创建 gen 子目录, 其中有 ant 生成的 R.java 文件(在 ca\tutortutor\univerter 目录中), 并将创建的 Univerter-debug.apk 文件保存在 bin 子目录中。

1.4.11 安装和运行 Univerter

1. 问题

要将刚刚创建的 Univerter-debug.apk 包安装到前面启动的 AVD1 上, 运行该应用程序。

2. 解决方案

使用 adb 工具来安装 Univerter-debug.apk。转到应用程序启动器屏幕来运行 Univerter。

3. 实现机制

假设 AVD1 仍在运行, 执行下面的命令后, 可在 AVD1 中安装 Univerter-debug.apk:

```
adb install C:\prj\dev\Univerter\bin\Univerter-debug.apk
```

稍后, 你应当能看到与如下类似的信息:

```
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
269 KB/s (75946 bytes in 0.275s)
  pkg: /data/local/tmp/Univerter-debug.apk
Success
```

前两行“daemon”消息表明 ADB 守护进程没有运行, 但它已经被启动了。查看 <http://developer.android.com/tools/help/adb.html> 可了解更多关于 ADB 守护进程的内容。

注意:

如果在尝试安装应用程序时出现了一条失败的消息,很有可能是该应用程序已经安装了。

在主屏幕单击应用程序启动图标(在主屏幕底部居中的矩形网格图标),并左移 APPS 选项卡的内容。图 1-16 显示了 Univerter 应用程序的入口。

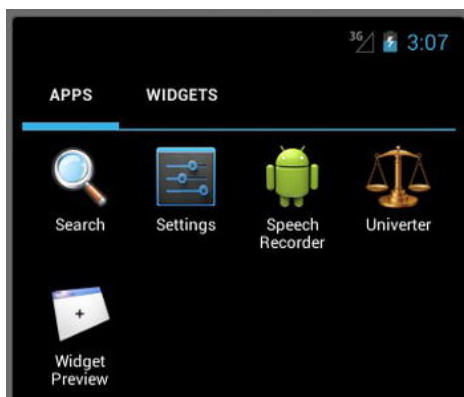


图 1-16 Univerter 应用程序入口为一个黄色的天平图标

单击 Univerter 图标,你应该看到图 1-17 所示的界面。



图 1-17 默认的种类是 ANGLE, 默认的换算是 CIRCLES > DEGREES

将 AVD1 的方向改为横向,会看到如图 1-18 所示的界面。

单击 CAT 按钮会显示种类列表,如图 1-19 所示。



图 1-18 换算名称显示在标题栏中



图 1-19 选择一个种类后单击 Close 按钮

图 1-20 显示了单击 CON 按钮后会显示当前(ANGLE)种类的转换关系列表。

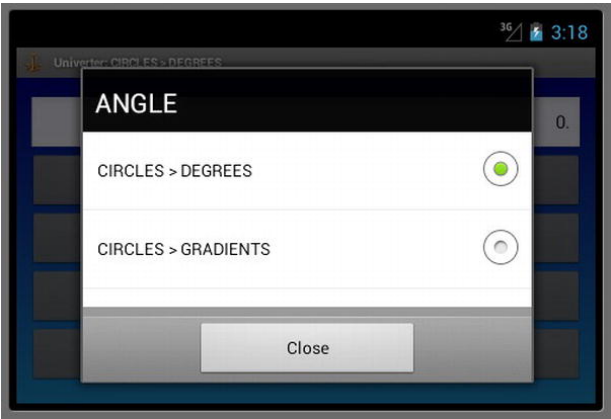


图 1-20 选择一个转换关系后，单击 Close 按钮

单击 MENU 按钮(或 action bar 上面的更多菜单(竖着显示 3 个点的地方))显示选项菜单。单击 help 菜单项会弹出帮助对话框，如图 1-21 所示。

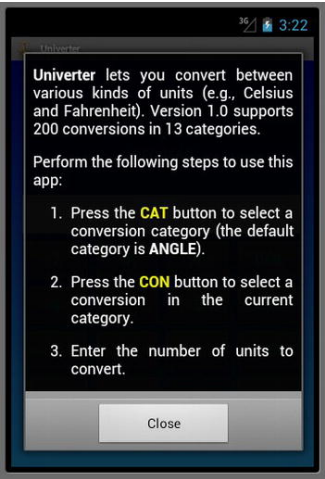


图 1-21 滚动帮助文本来查看 Univerter 的使用方法

最后单击 info 菜单项显示信息对话框，如图 1-22 所示。

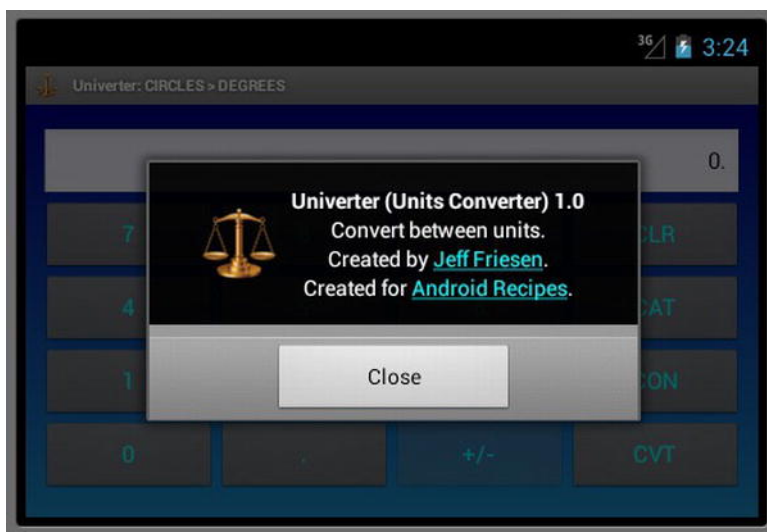


图 1-22 在 info 对话框中单击 Android Recipes 链接可访问本书的网页

继续体验 Univerter 吧。当体验完以后，可返回应用程序启动器界面尝试一下使用其他应用程序。

提示：

可以使用两种方式卸载 Univerter。从应用程序启动器界面的选项菜单中选择 Manageapps，向下滚动到 Univerter，选择该条目，单击 Uninstall 按钮。或者在命令行输入 `adbuninstall ca.tutortutor.univerter`。在程序开发中，第二种方法更快和更方便。

1.4.12 准备 Univerter 在 Google Play 上发布

1. 问题

Univerter 已经添加了很多的换算关系和其他功能，现在你希望在 Google Play(以前被称为 Android Market)上发布这个应用程序(<https://play.google.com/store>)，然而，你还不知道为应用程序发布做哪些准备工作。

2. 解决方案

在发布 Univerter 或另一个应用程序之前，你需要遵循如下 6 个步骤：

- (1) 全面地测试应用程序。
- (2) 在 manifest 文件中设置应用程序的版本。
- (3) 在 manifest 文件中请求必要的权限。
- (4) 在发布模式下构建应用程序。
- (5) 为应用程序包签名。
- (6) 对齐应用程序包。

完成上述步骤之后,在 Google Play 中注册(如果你还没有这么做的话),然后上传应用程序的 APK 文件。

3. 实现机制

以下 6 个部分分别详述了 6 个准备步骤。

全面地测试应用程序

Android 支持各种版本、设备类别(手机和平板电脑)和设备特征(如屏幕密度、是否具有摄像头),它们为开发应用程序提供充满挑战的环境。在各种环境下(版本/类别/特性的组合)全面测试应用程序非常重要。

Android 提供了工具和资源以帮助完成测试。例如,Android 采用基于 JUnit 的测试,并借助 junit.framework 和 junit.runner 包进行单元测试。在 Google 的 Android 系统文档中查看“Testing”章节(<http://developer.android.com/tools/testing/index.html>)可了解更多信息。

manifest 文件中的应用程序版本信息

Android 通过 AndroidManifest.xml 文件的<manifest>标签中的 versionCode 和 versionName 属性指定应用程序的版本信息。

versionCode 是一个整数值,表示应用代码的版本。该属性的值是一个整数,其他应用可以据此判断应用版本的升降。尽管可以将该属性的值设置为任意的整数,但应该确保每次发布时都递增这个值。Android 对此并没有强制要求,但每次发布时加大版本号才是正常的。

versionName 是一个表示应用程序版本的字符串,应该通过应用程序显示给用户。这个值是一个字符串,所以可以这样描述应用程序的版本<major>.<minor>.<point>,当然也可以使用其他类型的绝对或相对版本标识符。与 android:versionCode 不同,Android 不会将 versionName 用于内部操作。发布服务会将应用的 versionName 显示给用户。

Univerter 的 AndroidManifest.xml 文件的<manifest>标签中的 versionCode 属性初始化为“1”,versionName 属性则初始化为“1.0”。

在管理项目版本时,还应该为应用程序指定所支持的最小 SDK 版本。为了实现上述目的,可在 AndroidManifest.xml 文件中使用<uses-sdk>标签,并将其 minSdkVersion 属性设置为所期望的最低 API Level。例如,下面的标签将<uses-sdk>设置为 Level 10 (Gingerbread/2.3.3),这是被 Univerter 支持的最低 SDK 版本。

```
<uses-sdk android:minSdkVersion="10"/>
```

在 manifest 文件中请求必要的权限

应用程序需要获得许可才能执行某些任务。例如,如果应用程序使用了 android.webkit.WebView 类来查看互联网上的网页,你必须在 androidmanifest.xml 中添加如下<uses-permission>标签:

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

使用 `WebView` 类时，并不总是需要上述标签。例如，在 `Univerter` 应用程序中使用 `WebView` 就不需要该标签。因为 `Univerter` 可从字符串资源中获得 HTML 内容。

在发布模式下构建应用程序

在调试模式下构建的应用程序是不能发布的，必须将其在发布模式下重新构建，可执行下面的命令来完成该任务：

```
ant release
```

假设 `Univerter` 是在发布模式下构建的，那么 `bin` 目录应该包含 `Univerter-release-unsigned.apk` 文件。

为应用程序包签名

Android 系统要求所有已安装的应用程序具有数字签名的证书，其密钥由应用程序的开发者持有。它使用证书来识别应用程序的作者，并在应用程序之间建立信任的关系，证书并不用于控制哪些应用程序可以由用户来安装。

注意：

证书不需要由证书颁发机构签署，Android 应用程序允许(通常情况下)使用自签名证书。

Android 只有在安装期间才会检查签名证书的有效期。如果安装应用程序后，其签名证书才到期，该应用程序是可继续正常使用的。

在为应用程序包签名之前，必须具有一个合适的密钥。密钥应符合如下规范：

- 密钥中包含了能用于标识应用的个人、公司或组织信息。
- 密钥的有效期要比应用的期望生命周期长。Google 建议有效期应超过 25 年。如果要将应用发布到 Google Play，要注意有效期必须超过 2033 年 10 月 22 号。如果签署应用的密钥过期，就无法将应用上传到 Android Market。
- 密钥不是 Android SDK 工具生成的调试密钥。

JDK 的 `keytool` 工具用于创建一个合适密钥。下面的命令行(为方便阅读，分两行显示)，假设当前目录为 `C:\prj\dev\Univerter`，使用 `keytool` 生成密钥：

```
keytool -genkey -v -keystore univerter-release-key.keystore -alias
univerter_key -keyalg RSA -keysize 2048 -validity 10000
```

命令行使用的参数如下：

- `-genkey` 导致 `keytool` 生成一个公钥和一个密钥(一对)。
- `-v` 启用详细输出。
- `-keystore` 指定保存密钥的 `keystore`(密钥数据库和验证公钥的 X.509 证书链接)。在这个命令行中，文件名为 `univerter-release-key.keystore`。
- `-alias` 设置密钥的别名(在真正的签名操作中使用别名时，只会使用前 8 个字符)。本例的别名为 `univerter_key`。
- `-keyalg` 设置生成密钥的加密算法，支持 DSA 和 RSA，本例用的是 RSA。

- `-keysize` 设置生成的密钥大小。Google 推荐使用 2048 位或更大的密钥，本例设置为 2048 位(默认大小是 1024 位)。
- `-validity` 设置密钥的有效期(单位是天，Google 推荐的值是大于等于 10 000)，本例中设置为 10 000。

keytool 会提示你输入密码(用于保护保存密钥的文件)，重复输入一次密码。然后它会提示你输入姓名、部门名称、组织名称、城市或居住地名称、州或省的名称以及两个字母的国家代码。

keytool 接下来提示你确认上述信息是否正确(如果正确，输入 yes 再按 Enter 键；如果不正确，直接按 Enter 键)。如果输入 yes，keytool 会要你为钥匙输入一个不同的密码，当然也可以使用保护密钥文件的密码。

警告：

一定要确保密钥的安全，否则应用的版权就可能会受到威胁并降低用户的信任度。下面是一些确保私钥安全的提示：

- 用复杂的密码保护密钥文件和密钥。
- 在用 keytool 生成密钥时，不要在命令行中使用 `-storepass` 和 `-keypass` 选项。否则，任何使用你电脑的用户都可以在命令行解释器的历史记录中看到你的密码。
- 在用 jarsigner 签署应用时，不要在命令行中使用 `-storepass` 和 `-keypass` 选项(原因同上)。
- 不要将密钥交给其他人，也不要让未经授权的人获取你的密钥文件和密钥密码。

keytool 会在当前目录中创建 `univerter-release-key` 文件。执行下面的命令可以查看密钥文件的信息：

```
keytool -list -v -keystore univerter -release -key.keystore
```

在询问密钥文件密码后，keytool 会输出密钥文件中的条目数量(应该是 1)和认证信息。

JDK 的 jarsigner 工具用于签署 `Univerter-release-unsigned.apk`。假设当前目录是 `C:\prj\dev\Univerter`，目录中有 keytool 创建的 `Univerter-release-key.keystore` 文件，`Univerter-releaseunsigned.apk` 在该目录的 `bin` 子目录中，执行下面的命令对该文件进行签名(为了便于阅读，分两行显示)：

```
jarsigner -verbose -keystore univerter-release-key.keystore
        bin/Univerter-release-unsigned.apk univerter_key
```

命令行中使用的参数如下：

- `-verbose` 启用详细输出。
- `-keystore` 设置保存密钥 keystore，本例中设置的是 `Univerter-release-key.keystore`。
- `bin / Univerter-release-unsigned.apk` 表示被签名的 .apk 文件的位置和文件名。
- `univerter-key` 设置之前创建的密钥别名。

jarsigner 会提示你输入之前通过 keytool 设置的 keystore 文件密码。然后会输出如下信息:

```
adding: META-INF/MANIFEST.MF
adding: META-INF/UNIVERTE.SF
adding: META-INF/UNIVERTE.RSA
signing: res/drawable/gradientbg.xml
signing: res/layout/help.xml
signing: res/layout/info.xml
signing: res/layout/list_row.xml
signing: res/layout/main.xml
signing: res/menu/univerter.xml
signing: AndroidManifest.xml
signing: resources.arsc
signing: res/drawable-hdpi/ic_launcher.png
signing: res/drawable-ldpi/ic_launcher.png
signing: res/drawable-mdpi/ic_launcher.png
signing: res/drawable-xhdpi/ic_launcher.png
signing: res/layout-land/main.xml
signing: classes.dex
```

注意:

前面的 jarsigner 命令在 JDK 7 中是有问题的。当为 APK 文件的发布版本签名(并对齐 APK 文件,稍后讨论)后,APK 文件将无法安装到设备上。解决方法是在命令中添加 -digestalg SHA1 -sigalg MD5withRSA 选项,相关文档参见 <http://code.google.com/p/android/issues/detail?id=19567>。对于 JDK 7 用户而言,可用下面的命令行(分三行以便阅读)代替:

```
jarsigner -verbose -keystore univerter-release-key.keystore
        bin/Univerter-release-unsigned.apk -digestalg SHA1
        -sigalg MD5withRSA univerter_key
```

执行下列命令行(为了方便阅读,分两行显示)来验证 Univerter-release-unsigned.apk 是否已经签名:

```
jarsigner -verify -keystore univerter-release-key.keystore
        bin/Univerter-release-unsigned.apk
```

如果验证成功,将出现“jar verified”消息。

对齐应用程序包

出于性能优化方面的考虑,Android 要求签署过的 APK 中未经压缩的内容应该与文件的起点对齐,为此 Android SDK 中提供了 zipalign 工具来完成此项工作。根据 Google 的文档,APK 中所有未经压缩的数据,例如图片或原始文件,都应该对齐到 4 字节边界。

下面的命令用 zipalign 将输入的 APK 对齐成输出的 APK:

```
zipalign [-f] [-v] alignment infile.apk outfile.apk
```

命令行参数如下：

- 如果 outfile.apk 文件存在，-f 强行覆盖。
- -v 启用详细输出。
- alignment 设置 APK 内容对齐的位数。似乎 zipalign 能接受的值只有 4。
- infile.apk 是已签名的要对齐的 APK 文件。
- outfile.apk 是输出的已签名和对齐的 APK 文件。

假设当前目录为 C:\prj\dev\Univerter\bin，执行下面的命令将 Univerter-release-unsigned.apk 对齐为 Univerter.apk 文件：

```
zipalign -f -v 4 Univerter-release-unsigned.apk Univerter.apk
```

zipalign 用以下语法来验证现存的 APK 文件是否对齐：

```
zipalign -c -v alignment existing.apk
```

命令行参数如下：

- -c 核对 existing.apk 文件是否对齐。
- -v 启用详细输出。
- alignment 设置 APK 内容对齐的位数。似乎 zipalign 能接受的值只有 4。
- existing.apk 指定要对齐的 APK 文件(已签名)。

执行以下命令来验证 Univerter.apk 是否对齐：

```
zipalign -c -v 4 Univerter.apk
```

zipalign 会显示 APK 文件中的内容清单，标明其中哪些是压缩的，哪些是未经压缩的，接下来是验证是否成功的消息。

Univerter.apk 现在已经可以发布了。

1.4.13 移植到 Eclipse

1. 问题

你更喜欢使用 Eclipse IDE 开发应用程序。

2. 解决方案

为了使用 Eclipse 开发应用程序，你需要安装一个 IDE，例如 Eclipse Classic 4.2。此外，还需要安装 Android 开发工具(Android Development Tools, ADT) 插件。

3. 实现机制

在使用 Eclipse 开发 Android 应用程序之前，必须至少完成以下三个任务中的前两个：

(1) 安装 Android SDK 和至少一个 Android 平台(参见范例 1-1 和 1-2)。必须安装 JDK 6 或 JDK 7。

(2) 安装一个 Eclipse 版本，使 Eclipse IDE 能兼容 Android SDK 和 ADT 插件。

(3) 安装 ADT 插件。

你应该按次序完成上述任务。不能在安装 Eclipse 之前安装 ADT 插件，也不能在安装 Android SDK 和至少一个 Android 平台之前配置或使用 ADT 插件。

ADT 插件的优势

虽然在 Eclipse 中开发 Android 应用程序时，可不使用 ADT 插件，然而使用插件能更快、更容易地创建和调试程序，因此推荐使用插件来开发应用程序。

ADT 插件具有如下功能：

- 允许在 Eclipse IDE 内部访问其他 Android 开发工具。例如，ADT 允许使用 Dalvik 调试监控服务器(DDMS)工具的许多功能，允许截图、管理端口转发、设置断点、直接从 Eclipse 查看线程和进程信息。
- 它提供了新的项目向导，有助于快速创建新的 Android 应用程序所需的所有基本文件。
- 它能自动化执行并简化构建 Android 应用程序的流程。
- 它提供了 Android 代码编辑器，以辅助 Android 写出有效的 manifest 和资源 XML 文件。
- 它允许将项目输出为已签名 APK 文件，该签名的 APK 文件可以供用户使用。

在学习如何安装 Eclipse 之后，你将学习如何安装 ADT 插件。

在 Eclipse.org 网站上可以下载多种 IDE 包，以满足不同的需求。下载和安装哪一种 IDE 包的 Google 规定如下：

- 安装 Eclipse3.6.2 (Helios)或更高级的 IDE 包。
- 确保下载的 Eclipse 包中包括 Eclipse JDT(Java 开发工具)插件。大多数软件包都包括这个插件。

按照以下步骤安装 Eclipse Classic 4.2，该版本是撰写本书时 IDE 的最新版：

(1) 打开浏览器并导航到 Eclipse Classic 4.2 页面，其网址为：

<http://eclipse.org/downloads/packages/eclipse-classic-42/junior>。

(2) 在页面右侧的下载链接框中，单击任一下载链接选择适当的发布文件。例如，你可以单击 Windows 64 位平台。

(3) 单击下载链接，将发布文件保存到硬盘中。例如，你可以将 eclipse-SDK-4.2-win32-x86_64.zip 文件保存到硬盘。

(4) 解压发布文件，并将 eclipse 主目录移动到一个相对方便的位置。例如，在 64 位 Windows 7 中，你应该将 eclipse 移动到 C:\Program Files 目录中，该目录负责组织 64 位的应用程序。

(5) 需要为位于 eclipse 主目录的 eclipse 应用程序创建桌面快捷方式。

按照以下步骤安装最新修订的 ADT 插件：

(1) 启动 Eclipse。

(2) 第一次启动 Eclipse 时，在启动屏幕显示之后，将会出现工作区启动对话框。你可

以使用此对话框选择一个工作区文件夹来存储项目。也可设置在以后启动 Eclipse 时不显示该对话框。修改或保持默认文件夹设置，然后单击 OK 按钮。

(3) Eclipse 显示主窗口后，在 Help 菜单中选择 Install New Software。

(4) 在之后的 Install dialog box's Available Software 界面上单击 Add 按钮。

(5) 在之后的 Add Repository 对话框中的 Name 一栏输入远程站点的名称(例如，Android 插件)，并在 Location 一栏中输入 <https://dl-ssl.google.com/android/eclipse/>，单击 OK 按钮。

(6) 现在你应该在 Install 对话框的中间看到 Developer Tools 和 NDK Plugins 列表。

(7) 选中相应分类旁边的复选框，将自动选中该分类下面嵌套的项。单击 Next 按钮。

(8) 之后出现的 Install Details 界面中会列出 Android DDMS、Android Development Tools、Android Hierarchy Viewer、Android Native Development Tools、Android Traceview 和 Tracer for OpenGL ES。单击 Next 按钮阅读和接受许可协议，然后单击 Finish。

(9) 之后会出现 Installing Software 对话框，提示进行安装。如果遇到 Security Warning 对话框，单击 OK 按钮。

(10) 最后，Eclipse 会显示 Software Updates 对话框，提示你重新启动 IDE。单击 Yes 重启。

提示：

如果在步骤(5)无法获得插件，在 Location 一栏中尝试使用 http 而不是 https(使用 https 是由于安全的原因)。

想要完成安装 ADT 插件，还必须在 Eclipse 中修改 ADT 的设置，使其指向 Android SDK 的主目录。步骤如下：

(1) 从 Window 菜单中选择 Preferences，从而打开 Preferences 对话框。在 Mac OS X 中，从 Eclipse 菜单选择 Preferences 选项。

(2) 从左侧面板中选择 Android。

(3) 如果 SDK Location 一栏已经被设置为 SDK 的主目录(例如 C:\android)，直接关闭 Preferences 对话框即可，不必做任何操作。

(4) 如果 SDK Location 一栏中并没有设置为 SDK 的主目录，单击旁边的 Browse 按钮，在弹出的“Browse For Folder”对话框中找到下载的 SDK 的主目录，选择该目录，单击 OK 按钮关闭对话框，然后在 Preferences 对话框中单击 Apply 按钮，确认所选位置，这时在下面将出现 SDK Targets(例如 Android 4.1)列表。

注意：

为了获得更多安装 ADT 插件的帮助信息(包括遇到问题的帮助信息)，可以在 Google 的在线 Android 文档中查看“Installing the Eclipse Plugin”页面(<http://developer.android.com/sdk/installing/installingadt.html>)。

1.4.14 用 Eclipse 创建和运行 Univerter

1. 问题

现在你已经安装了 Eclipse Classic 4.2 和 ADT 插件，你想学习如何使用这个 IDE 和插件创建和运行 Univerter。

2. 解决方案

首先需要创建一个 Android Eclipse 项目，名称为 Univerter。然后复制 Univerter 的源文件和资源到该项目中。最后，从菜单栏中选择 Run 菜单以执行 Univerter。

3. 实现机制

使用 Eclipse 创建和运行 Univerter 的首要任务是创建一个新的 Android 项目，步骤如下：

(1) 如果 Eclipse 没有运行，则启动它。

(2) 选择 File 菜单中的 New，在之后的弹出菜单中选择 Project。

(3) 在之后出现的 New Project 对话框中，展开向导树中的 Android 节点(如果尚未展开)，选择该节点下的 Android Application Project(如果尚未选择)，单击 Next 按钮。

(4) 在之后出现的 New Android App 对话框中，在 Application Name 一栏输入 Univerter (这个名字同时会在 Project Name 一栏中显示，它表示 Univerter 项目所在的文件夹/目录)并且在 Package Name 一栏中输入 ca.tutortutor.univerter。同时，在 Minimum Required SDK 列表中选择 API 10: Android 2.3.3 (Gingerbread)并且不选中 Create customlauncher icon。保持其他设置项不动，单击 Next 按钮。

(5) 在之后出现的 Create Activity 窗格中，不选中 Create Activity，单击 Finish。

在 Eclipse 工作区目录中，Eclipse 会创建 Univerter 目录，该目录还包含如下子目录和文件：

- .settings: 此目录包含一个 org.eclipse.jdt.core.prefs 文件，用于记录项目的具体设置。
- assets: 这个目录用于存储非结构化层次文件。存储在该目录的任何对象均可通过原始字节流被应用程序检索。
- bin: APK 文件保存在该目录。
- gen: 生成的 R.java 文件存储在子目录结构中，反映出包的层次结构(如 ca\tutortutor\univerter)。
- res: 应用程序资源存储在该目录不同的子目录中。
- src: 应用程序源代码，按照包的层次结构存储。
- .classpath: 这个文件存储项目的类路径信息，用于定位项目所需的外部库。
- .project: 此文件包含项目的重要信息，如项目的名字和构建规范。
- AndroidManifest.xml: 此文件包含 Univerter 的清单。
- proguard-project.txt: 此文件包含使用的 ProGuard 的信息。
- project.properties: 该文件标识了该项目的目标 Android 平台。

选择 welcome 选项卡，Eclipse 然后显示如图 1-23 的用户界面。

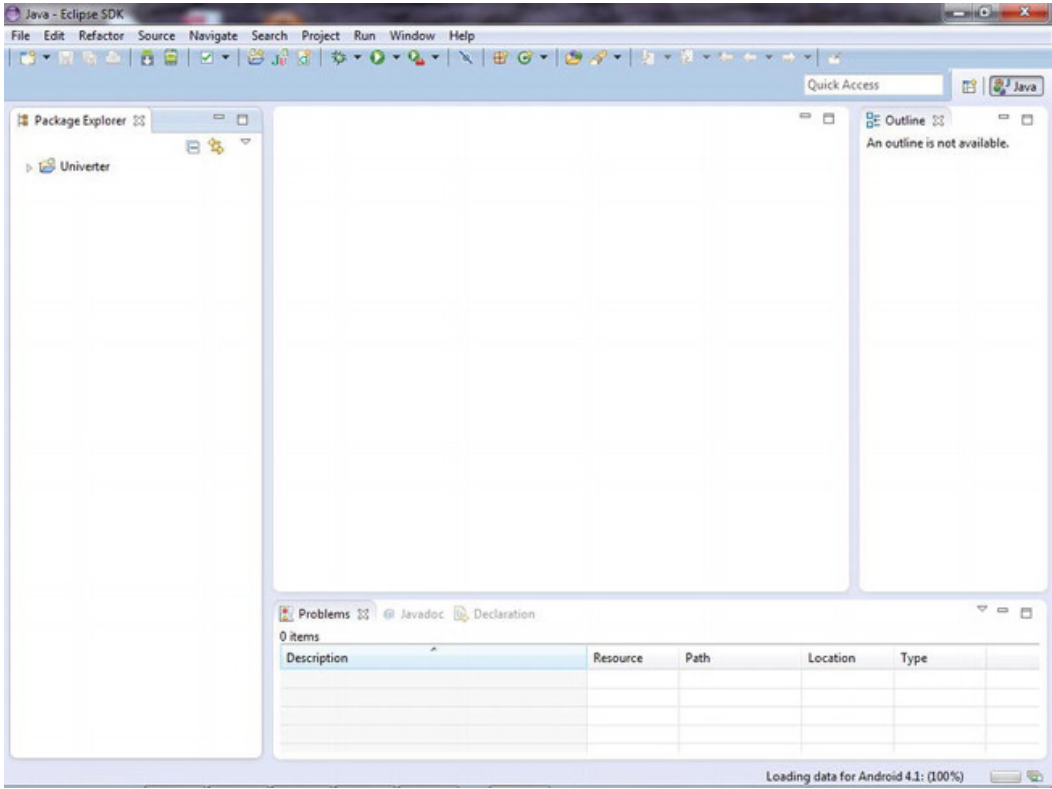


图 1-23 Eclipse 的用户界面由菜单栏、工具栏、若干个窗口(例如 Package Explorer 和 Outline)、状态栏以及保留给编辑器窗口的空白区域组成

这个用户界面就是 Eclipse 工作台。其左侧是 Package Explorer 窗口，其中显示了当前工作空间中的各个项目及其组件。图 1-23 中的工作空间中只有 Univerter 一个项目。

为了了解 Eclipse 如何组织 Univerter 项目，可单击左侧 Univerter 节点的三角图标。图 1-24 显示了扩展后的项目层次结构。

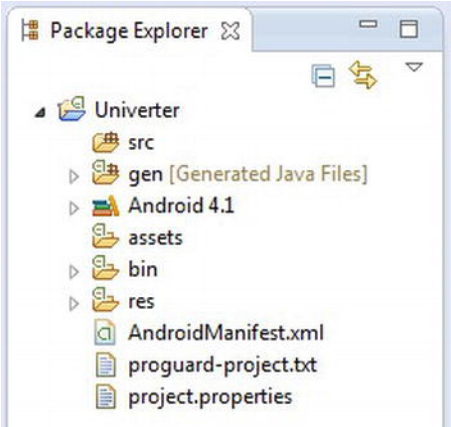


图 1-24 该层次结构显示了比较重要的 src、res 目录及 Androidmanifest.xml 清单文件

src 节点是空的, 按照以下步骤, 在 src 目录下创建 ca\tutortutor\univerter 目录结构:

(1) 右击 src 节点, 然后在弹出的菜单中选择 New | Folder。

(2) 在之后出现的 New Folder 对话框中, 在 Folder name 一栏中输入 ca/tutortutor/univerter, 单击 Finish 按钮。

接下来, 需要将 Category.java、Conversion.java、Converter.java 和 Univerter.java 源文件放到 ca\tutortutor\univerter 目录中, 其方法如下:

(1) 把这些文件复制到剪贴板。

(2) 右击 src 下面的 ca.tutortutor.univerter 节点, 并且从弹出菜单中选择 Paste。

现在可以看到 Category.java、Conversion.java、Converter.java 和 Univerter.java 文件已经在 ca.tutortutor.univerter 节点的下面了。

res 节点包含多个不需要的节点, 按照以下步骤用 Univerter 的资源结构填充该目录:

(1) 将 res 下的所有节点高亮选定、右击, 并从弹出菜单选择 Delete。

(2) 将 Univerter res 下的所有目录复制到剪贴板。

(3) 右击 res 节点, 并从弹出菜单中选择 Paste。

现在, 在 res 下能看到 drawable、drawable-hdpi、drawable-ldpi、drawablemdpi、drawable-xhdpi、layout、layout-land、menu 和 values 节点。

最后更新 AndroidManifest.xml 节点, 使其引用正确的内容, 方法如下:

(1) 将 Univerter 的 AndroidManifest.xml 文件复制到剪贴板。

(2) 右击 AndroidManifest.xml 节点, 并从弹出菜单中选择 Paste。

此时在工作区底部的 Problems 界面中, Eclipse 可能会报告 9 个错误。这些错误都是关于在覆写的接口方法(例如, public void convert(Context ctx, double value))中使用 @Override 注解的。在 Java 5(1.5)中无法使用该注解, 它从 Java 6 才开始被允许。

按照如下方法步骤, 可以很容易地解决这个问题:

(1) 右击 Univerter 节点, 从弹出菜单选择 Properties。

(2) 在 Univerter 的 Properties 对话框中, 选择 Java Compiler。

(3) 在 Java Compiler 面板中, 将 Compiler compliance 级别从 1.5 修改为 1.6, 然后关闭对话框。

错误报告应该消失了。

此时文件结构和兼容级别已经设置完成, 从菜单栏选择 Run 菜单, 在之后出现的下拉菜单中选择 Run。然后在出现的 Run As 对话框中, 选择 Android Application, 单击 OK 按钮。

如果一切顺利, Eclipse 会启动模拟器 AVD1 并安装 Univerter-debug.apk 文件, 然后应用程序开始运行(参见图 1-17)。当第一次启动 AVD1 时, 可能需要绕过主屏幕介绍和启动器界面才能看到应用程序。

Eclipse 和发布模式

在某些时候, 可能需要在 Eclipse 中创建 Univerter 的发布版本, 可采用如下方法:

(1) 从 File 菜单选择 Export。

(2) 在之后的 Export 对话框中, 选择 Android | Export Android Application under Android。

单击 Next 按钮。

(3) 在之后出现的 Export Android Application 对话框中, 在 Project 一栏中输入 Univerter, 单击 next。

(4) 在 Keystore selection 界面中, 将密钥的位置(例如, C:\prj\dev\Univerter\univerterrelease-key.keystore) 输入到 Location 一栏中, 并在 Password 一栏中输入密码(Univerter)。单击 Next 按钮。

(5) 在 Key alias selection 界面中, 选择密钥别名(univerter_key), 并输入密码(univerter), 单击 Next 按钮。

(6) 在 Destination and key/certificate checks 窗格中, 输入目标 APK 文件的位置(例如, C:\temp\Univerter.apk), 并单击 Finish。

几秒钟后, 一个已签名的 Univerter.apk 文件应该在目标目录中被成功创建。

1.5 小结

Android 让很多为这个平台开发(甚至销售)应用的人们激动不已。现在加入这场盛宴还不晚。本章带你快速地浏览了 Android 的关键概念和开发工具。

首先了解了 Android 是一个针对移动设备的软件栈, 这个栈由应用、中间件和 Linux 操作系统组成。然后学习了 Android 的历史, 包括现有的各种 SDK 的更新。

随后描述了 Android 的分层架构, 其中顶层是应用, 中间是由应用框架、C/C++库和 Dalvik 虚拟机构成的中间件, 底层则是修改过的 Linux 内核。

然后介绍应用程序架构, 该架构基于组件, 并通过 Intent 进行通信。资源常用于用户界面上下文环境中; manifest 文件描述了应用程序的组件(甚至更多); 应用程序包存储了组件、资源和 manifest 文件。

接下来, 本章结束了基本理论的介绍, 通过一系列范例介绍了一些实践操作。前几个范例集中介绍了如何安装 Android SDK 和 Android 平台, 以及如何创建 AVD 并用模拟器启动 AVD。

接下来的范例介绍了 Univerter——单位转换器应用程序。这些范例向你展示了如何创建应用程序、在模拟器中安装、在模拟器中运行以及如何创建发布版本, 并在 Google Play 上发布。

在命令行环境下使用命令行工具很乏味, 出于这个原因, 最后两个范例重点讨论将应用程序导出到 Eclipse IDE, 并介绍了在该图形环境中如何创建和运行 Univerter。

在对 Univerter 应用程序的探索中, 还提出各种用户界面的概念。第 2 章将继续使用这些概念, 并提出更多的范例, 演示如何创建基于界面的任务。

第 2 章

用户界面范例

Android 平台能运行在各种类型的设备上，这些设备会有各种各样的设备尺寸、屏幕尺寸和分辨率。为了帮助开发者应对这个挑战，Android 提供了大量的用户界面组件工具集，开发者可以根据具体的应用程序选择使用组件或者自定义组件。Android 还非常依赖于可扩展的 XML 框架和设置资源标识符以实现能够兼容各种环境变化的浮动布局。本章中，我们会学习如何使用这个框架以满足具体的开发需求。

2.1 自定义窗口

2.1.1 问题

想要让你的应用程序在所有用户可能运行的 Android 版本上创建一致的外观和体验。应用程序可能还需要切换系统元素来获得更多的屏幕空间。

2.1.2 解决方案

(API Level 1)

用主题和 WindowManager 自定义窗口的属性和功能。在没有任何自定义时，Android 应用程序的 Activity 会加载默认的系统主题。根据目标 Android 版本的不同，在 Android 2.x 上是标准的平面黑色，Android 3.x 和 Android 4.x 是著名的 Holo 主题，或者制造商自己定义的皮肤来取代 Android 设备默认的主题。为了确保你的应用程序在所有设备上如你所愿，你需要声明使用一个系统主题或自定义主题。

2.1.3 实现机制

1. 用主题自定义窗口属性

在 Android 中，主题(Theme)就是一种应用到整个应用程序或某个 Activity 的外观风格。

使用主题有两个选择,使用系统主题,或是使用自定义主题。无论是哪种方法,都要在 AndroidManifest.xml 文件中设置主题,如程序清单 2-1 所示。

程序清单 2-1 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...>
    <!--Apply to the application tag for a global theme -->
    <application android:theme="THEME_NAME"
        ...>
        <!--Apply to the activity tag for an individual theme -->
        <activity android:name=".Activity" android:theme="THEME_NAME"
            ...>
            <intent-filter>
                ...
            </intent-filter>
        </activity>
    </application>
</manifest>
```

系统主题

具有 Android 框架的 styles.xml 文件中包含了一些主题,其中是一些有用的自定义属性。完整的清单请查阅 SDK 文档中的 R.style,下面是几个常用的示例:

- Theme.Light: 标准主题的变种,该主题的背景和用户元素使用一个相反的颜色主题。它是 Android 3.0 以前版本的应用程序默认推荐使用的主题。
- Theme.NoTitleBar.Fullscreen: 移除标题栏和状态栏,全屏显示(去掉屏幕上所有的组件)。
- Theme.Dialog: 让 Activity 看着像对话框。
- Theme.Holo.Light: (API Level 11)使用逆配色方案的主题并默认拥有一个 ActionBar。这是 Android 3.0 上应用程序默认推荐的基本主题。
- Theme.Holo.Light.DarkActionBar: (API Level 14)使用逆配色方案的主题但 ActionBar 是黑色实线的。这是 Android 4.0 上应用程序默认推荐的基本主题。

程序清单 2-2 中的示例通过设置 AndroidManifest.xml 文件中的 android:theme 属性,将一个系统主题应用到整个应用程序。

程序清单 2-2 设置为应用主题的 manifest 文件

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...>
    <!--为应用程序标签设置全局主题 -->
    <application android:theme="Theme.NoTitleBar"
        ...>
        ...
    </application>
</manifest>
```

自定义主题

有时候系统提供的主题还不能满足需求。毕竟，系统提供的主题并不能自定义窗口中的所有元素。定义自定义主题能方便地解决这个问题。

找到项目目录 `res/values` 下的 `styles.xml` 文件，如果没有就创建一个。记住，主题就是应用范围更广的风格样式，所以二者是在同一个地方定义的。跟窗口自定义有关的主题元素可以在 SDK 的 `R.attr` 中找到，下面是常用的一些元素：

- `android:windowNoTitle`
 - 控制是否要移除默认的标题栏
 - 设为 `true` 以移除标题栏
- `android:windowFullscreen`
 - 控制是否移除系统状态栏
 - 设为 `true` 以移除状态栏并全屏显示
- `android:windowBackground`
 - 将某个颜色或可绘制资源设为背景
 - 设置颜色、可绘制的值或资源
- `android:windowContentOverlay`
 - 窗口内容的前景之上的可绘制资源。默认情况下，就是状态栏下的阴影
 - 可以用任何的资源代替默认的状态栏，或者设为 `null`(XML 中为 `@null`)以将其移除
- `android:windowTitleBackgroundStyle`
 - 应用到窗口的标题视图的样式
 - 可设为任何样式资源
- `android:windowTitleSize`
 - 窗口标题视图的高度
 - 可设为任何尺度或尺度资源
- `android:windowTitleStyle`
 - 应用到窗口标题文本的样式
 - 可设为任何样式资源
- `android:actionBarStyle`
 - 应用到窗口 `ActionBar` 的样式
 - 可设为任何样式资源

程序清单 2-3 就是一个 `styles.xml` 示例，其中创建了两个自定义主题。

- `MyTheme.One`: 没有标题栏，移除默认的状态栏阴影
- `MyTheme.Two`: 用自定义的背景图片全屏显示

程序清单 2-3 有两个自定义主题的 `res/values/styles.xml`

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

```

<style name="MyTheme.One" parent="@android:style/Theme">
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowContentOverlay">@null</item>
</style>
<style name="MyTheme.Two" parent="@android:style/Theme">
    <item name="android:windowBackground">@drawable/window_bg</item>
    <item name="android:windowFullscreen">true</item>
</style>
</resources>

```

注意，主题(或风格)也可以从父主题继承属性，所以并不需要从头创建整个主题，在这个示例中，我们选择了继承 Android 默认的系统主题，只自定义我们要修改的属性。所有平台主题都定义在 Android 包的 res/values/themes.xml 文件中。关于风格和主题的更多细节请查阅 SDK 文档。

程序清单 2-4 展示了如何在 AndroidManifest.xml 中对单个 Activity 实例应用这些主题。

程序清单 2-4 在 manifest 中为每个 Activity 设置主题

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    ...>
    <!--通过 application 标签来设置全局主题-->
    <application
        ...>
        <!--通过 activity 标签设置单独的主题 -->
        <activity android:name=".ActivityOne" android:theme="MyTheme.One"
            ...>
            <intent-filter>
                ...
            </intent-filter>
        </activity>
        <activity android:name=".ActivityTwo" android:theme="MyTheme.Two"
            ...>
            <intent-filter>
                ...
            </intent-filter>
        </activity>

    </application>
</manifest>

```

在代码中自定义窗口特性

除了使用样式 XML 文件，还可以在 Activity 的 Java 代码中设置窗口的特性。使用该方法时，开发者能自定义的特性集稍有不同，但大部分和 XML 文件是一样的。

通过代码自定义窗口，在将内容视图用于 Activity 之前，每个窗口特性的改动都需要调用 Activity.requestWindowFeature()方法来请求系统。

注意:

所有通过 `Activity.requestWindowFeature()` 方法修改窗口特性的请求都必须在调用 `Activity setContentView()` 之前完成。在此之后的所有改动都不会生效。

从窗口可以获得的特性及其含义如下:

- `FEATURE_CUSTOM_TITLE`: 将自定义的布局资源设为 `Activity` 的标题视图。
- `FEATURE_NO_TITLE`: 将该标题视图从 `Activity` 移除。
- `FEATURE_PROGRESS`: 在标题中使用一个确定式进度条(0%~100%)。
- `FEATURE_INDETERMINATE_PROGRESS`: 在标题视图中使用一个小的非确定式(圆形的)进度指示器。
- `FEATURE_LEFT_ICON`: 在标题视图的左侧放置一个小标题图标。
- `FEATURE_RIGHT_ICON`: 在标题视图的右侧放置一个小标题图标。

FEATURE_CUSTOM_TITLE

通过这个窗口特性, 可以用完全自定义的布局资源替换标准的标题(参见程序清单 2-5)。

程序清单 2-5 给 `Activity` 设置一个自定义的标题布局

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
    setContentView(R.layout.main);

    //设置自定义标题的布局资源
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
        R.layout.custom_title);
}
```

注意:

因为这个特性完全替换了默认的标题视图, 所以该特性不能跟其他的窗口特性标志一起使用。

FEATURE_NO_TITLE

该窗口特性用于移除标准的标题视图(参见程序清单 2-6)。

程序清单 2-6 移除 `Activity` 的标准标题视图

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.main);
}
```

注意:

因为该特性彻底移除了默认的标题视图, 所以该特性不能跟其他的特性标志一起使用。

FEATURE_PROGRESS

该窗口特性用于在窗口标题中设置一个确定式进度条。该进度条的值可以是 0(0%) 到 10000(100%) 之间的任何值(参见程序清单 2-7)。

程序清单 2-7 使用窗口进度条 Activity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_PROGRESS);
    setContentView(R.layout.main);

    //设置进度条可见
    setProgressBarVisibility(true);
    //用 setProgress 控制进度条
    setProgress(0);
    //进度到达 100%时, 进度栏消失
    setProgress(10000);
}
```

FEATURE_INDETERMINATE_PROGRESS

用这个窗口特性设置一个显示后台 Activity 的非确定式进度指示器(也叫转动进度条)。由于该指示器是非确定式的, 所以只有显示和隐藏两种状态(参见程序清单 2-8)。

程序清单 2-8 使用窗口的非确定式进度栏的 Activity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
    setContentView(R.layout.main);

    //显示进度指示器
    setProgressBarIndeterminateVisibility(true);

    //隐藏进度指示器
    setProgressBarIndeterminateVisibility(false);
}
```

FEATURE_LEFT_ICON 和 FEATURE_RIGHT_ICON (API Level 8)

该窗口特性用于在标题视图的左侧放置一个可绘制的小图标(参见程序清单 2-9)。

程序清单 2-9 使用图标的 Activity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_LEFT_ICON);
    requestWindowFeature(Window.FEATURE_RIGHT_ICON);

    setContentView(R.layout.main);

    //设置自定义图标的布局资源
    setFeatureDrawableResource(Window.FEATURE_LEFT_ICON, R.drawable.icon);
    setFeatureDrawableResource(Window.FEATURE_RIGHT_ICON, R.drawable.icon);
}
```

注意:

这些特性在 API Level 8 之前都是可用的, 但有一个 bug, 就是 FEATURE_RIGHT_ICON 并不会真正位于标题文本的右侧。

FEATURE_ACTION_BAR (API Level 11)

在 SDK 目标版本为 11 或更高的应用程序中, 这个窗口特性会作为默认样式的一部分默认启用。但是, 如果在用老的样式主题时希望在某些情况下能够启用 ActionBar, 这个特性也是可以在代码中进行请求的。参见程序清单 2-10。

程序清单 2-10 使用 ActionBar 的 Activity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //在 setContentView 之前请求窗口特性
    requestWindowFeature(Window.FEATURE_ACTION_BAR);
    setContentView(R.layout.main);

    //访问 ActionBar 并修改它
    ActionBar actionBar = getSupportActionBar();
}
```

FEATURE_ACTION_BAR_OVERLAY (API Level 11)

使用这个窗口特性后, ActionBar 元素会位于你的视图内容的上方, 而不是内容的下方。这在应用程序希望暂时隐藏和显示 ActionBar 而又不希望每次都调整整体布局的场景中非常有用(更多内容可以查看下一节)。参见程序清单 2-11。

程序清单 2-11 使用 ActionBar 叠层的 Activity

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

```
//在 setContentView 之前请求窗口特性
requestWindowFeature(Window.FEATURE_ACTION_BAR_OVERLAY);
setContentView(R.layout.main);
}
```

图 2-1 显示了一个同时启用了所有图标和进度特性的 Activity，而另外一个 Activity 则启用了 ActionBar 特性。

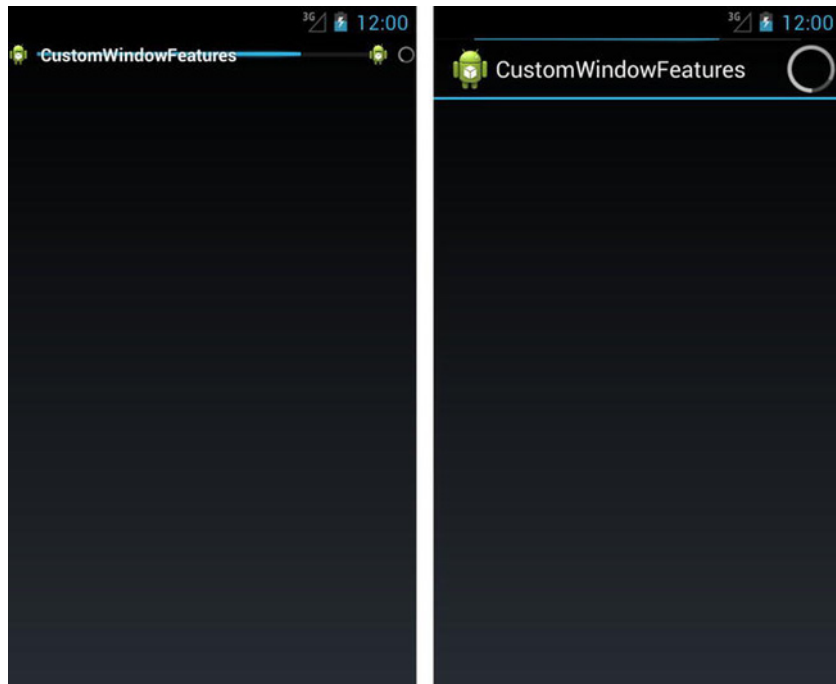


图 2-1 这里显示了在标题视图中启用窗口特性的效果(左边)和在 ActionBar 中启用特性的效果(右边)

2. 动态开关系统的 UI 控件

在应用程序的内容显示时，通过暂时隐藏系统 UI 控件，从而尽可能地提供更大的空间，可以让很多应用程序(例如阅读器或视频播放器)呈现更好的内容体验。从 Android 3.0 开始，开发者可以在运行时动态地调整这些属性而不必再静态地请求窗口特性以及主题声明的值。

夜间模式

(API Level 11)

通常也称为“熄灯模式”。指的是调暗屏幕导航控件(以及稍后发布版本中的系统状态栏)，而不是真正移除它们来减少屏幕上的系统元素(这些元素可能会将用户的注意力从当前视图中分散开去)。

想要启用这个模式，只需要简单地在视图结构中的任何视图中调用 `setSystemUiVisibility()` 即可。而想要恢复到默认模式，需要以同样的方式调用 `SYSTEM_UI_FLAG_VISIBLE`。通过调用 `getSystemUiVisibility()` 并检查标识的当前状态就可以知道我们现在所处的模式了。

注意:

这些标识名称都是在 API Level 14(Android 4.0)中引入的, 在之前的版本中叫做 STATUS_BAR_HIDDEN 和 STATUS_BAR_VISIBLE。它们的值都是一样的, 所以新的标识在 Android3.x 设备上也可以实现相同的功能。

程序清单 2-12 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:text="Toggle Mode"
        android:onClick="onToggleClick" />
</RelativeLayout>
```

程序清单 2-13 开关夜间模式的 Activity

```
public class DarkActivity extends Activity {

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

    public void onToggleClick(View v) {
        int currentVis = v.getSystemUiVisibility();
        int newVis;
        if((currentVis & View.SYSTEM_UI_FLAG_LOW_PROFILE)
            == View.SYSTEM_UI_FLAG_LOW_PROFILE) {
            newVis = View.SYSTEM_UI_FLAG_VISIBLE;
        } else {
            newVis = View.SYSTEM_UI_FLAG_LOW_PROFILE;
        }
        v.setSystemUiVisibility(newVis);
    }
}
```

要调节参数的窗口中所有可见的 View, 可以调用 setSystemUiVisibility()和 getSystemUiVisibility()方法。

隐藏导航控件

(API Level 14)

这个视图标记会移除没有物理按钮的设备屏幕上的 HOME 和 BACK 控件。Android 赋予开发者这个功能时是非常谨慎的, 因为这个功能对于用户来说太重要了。如果导航控制

器被手动地隐藏，屏幕上的任何单击都会回到上一界面。程序清单 2-14 显示了这个功能的示例。

程序清单 2-14 开关导航控制器的 Activity

```
public class HideActivity extends Activity {

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

    public void onToggleClick(View v) {
        //这里我们只需要单击屏幕即可隐藏控制器，因为控制器隐藏后，
        //只要再单击一下屏幕，系统就会让控制器自动重新出现
        v.setSystemUiVisibility(View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
    }
}
```

这个示例中还需要注意的是，因为我们是在根布局中进行设置，所以按钮会上下移动来适应内容区域的改变。如果你打算使用这个标识，需要注意在布局改变时所有位于屏幕底部的视图都会移动。

全屏 UI 模式

(API Level 11)

Android 4.1 之前是没有方法动态地隐藏系统状态栏的，只能通过设置静态主题来实现。在隐藏和显示 ActionBar 时，ActionBar.show()和 ActionBar.hide()会动态地显示和隐藏 ActionBar。如果请求的是 FEATURE_ACTION_BAR_OVERLAY，页面的变化将不会影响到 Activity 的内容；否则，视图的内容会上下移动来适应这种变化。

(API Level 16)

程序清单 2-15 演示了如何暂时隐藏所有系统 UI 的示例。

程序清单 2-15 开关所有系统 UI 的 Activity

```
public class FullActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //请求这个特性，这样 ActionBar 就会隐藏起来
        requestWindowFeature(Window.FEATURE_ACTION_BAR_OVERLAY);
        setContentView(R.layout.main);
    }

    public void onToggleClick(View v) {
        //这里我们只需要单击屏幕即可隐藏控制器，因为控制器隐藏后，
        //只要再单击一下屏幕，系统就会让控制器自动重新出现
    }
}
```

```

        v.setSystemUiVisibility(
            /* 这个标识会告诉 Android 在改变窗口大小来
             * 隐藏/显示系统元素时不要移动我们的布局
             */
            View.SYSTEM_UI_FLAG_LAYOUT_STABLE
            /* 这个标识会隐藏系统状态栏。如果请求 ACTION_BAR_OVERLAY,
             * 同时会隐藏 ActionBar
             */
            | View.SYSTEM_UI_FLAG_FULLSCREEN
            /* 这个标识会隐藏屏幕上的所有控制器
             */
            | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
    }
}

```

和只隐藏导航控制器的示例类似，我们不需要再次显示控制器，因为任何屏幕上的单击都会让它们再次显示出来。作为 Android 4.1 的一个便捷之处，当系统通过这种方式清除 `SYSTEM_UI_FLAG_HIDE_NAVIGATION` 后，同时会清除 `SYSTEM_UI_FLAG_FULLSCREEN` 标签，所以顶部和底部的元素会一起可见。如果我们请求了 `FEATURE_ACTION_BAR_OVERLAY`，Android 将会作为全屏标签的一部分隐藏 `ActionBar`；否则，只会影响到状态栏。

我们在这个示例中添加另一个有趣的标识：`SYSTEM_UI_LAYOUT_STABLE`。这个标识会通知 Android 在添加和移除系统 UI 时不要移动我们的内容视图。正因为如此，我们的按钮在开关 UI 时会一直位于中间。

2.2 创建并显示视图

2.2.1 问题

应用程序需要视图元素来显示信息并与用户交互。

2.2.2 解决方案

(API Level 1)

无论是使用 Android SDK 中的各种视图和工具，还是创建自定义的视图，所有的应用程序都需要使用视图来与用户进行交互。在 Android 中构建用户界面比较好的方法是，在 XML 中将其定义好，然后在运行时调用

Android 中的视图结构是树状的，根部通常是 `Activity` 或窗口的内容视图。`ViewGroup` 是一种特殊的视图，用于管理一个或多个子视图的显示方式。子视图可以是另一个 `ViewGroup`，整棵视图树就这样继续生长。所有的标准布局类都源自 `ViewGroup`，经常作为 XML 布局文件的根节点。

2.2.3 实现机制

下面定义一个有两个 Button 实例和一个 EditText 的布局来接收用户输入。我们可以在 res/layout 中定义一个名为 main.xml 的文件，参见程序清单 2-16。

程序清单 2-16 res/layout/main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <EditText
        android:id="@+id/editText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <Button
            android:id="@+id/save"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Save"
        />
        <Button
            android:id="@+id/cancel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cancel"
        />
    </LinearLayout>
</LinearLayout>
```

LinearLayout 是一个 ViewGroup，它将元素横向或纵向排列。在 main.xml 中，EditText 和其中的 LinearLayout 是按序纵向排列的。内部的 LinearLayout(里面是按钮)的内容是横向排列的。带有 android:id 值的视图元素可以在 Java 代码中引用，以备自定义或显示之用。

为了用这个布局显示 Activity 的内容，必须在运行时将其填充。经过覆写的 Activity.setContentView()方法可以很方便地完成这个工作，只需要提供布局的 ID 即可。在 Activity 中设置布局就是这样简单：

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //继续初始化 Activity
}
```

除了 ID(main.xml 有一个自动生成的 ID——R.layout.main)，其他东西都不需要。如果

在将布局附加到窗口时还需要进一步自定义，可以手动将其填充，在完成所需的自定义后再将其作为内容视图添加。程序清单 2-17 中填充了同一个布局，但在显示之前加上了第三个按钮。

程序清单 2-17 在显示之前修改布局

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //填充布局文件
    LinearLayout layout = (LinearLayout)getLayoutInflater().inflate
        (R.layout.main,null);
    //添加一个按钮
    Button reset = new Button(this);
    reset.setText("Reset Form");
    layout.addView(reset,
        new LinearLayout.LayoutParams(LayoutParams.FILL_PARENT,
            LayoutParams.WRAP_CONTENT));

    //将视图关联到窗口
    setContentView(layout);
}
```

在这个示例中，这个 XML 布局是在 Activity 的代码中用 `LayoutInflater` 填充的，它的 `inflate()` 方法会返回一个指向填充后的视图的句柄。因为 `LayoutInflater.inflate()` 返回的是视图，所以我们必须将其转换成 XML 中的某个子类，这样才能在将其关联到窗口之前进行修改。

注意：

XML 布局文件中的根元素是 `LayoutInflater.inflate()` 返回的 `View` 元素。

`inflate()` 的第二个参数代表父 `ViewGroup`，这个参数非常重要，因为它定义了该如何处理被填充布局中的 `LayoutParams`。可能的话，只要你知道被填充视图的父视图，都应该把它传进来；否则，XML 中根视图的 `LayoutParams` 会被忽略。当传入一个父视图后，还要注意 `inflate()` 的第三个参数，该参数决定了被填充的布局是否会自动关联到父视图上。在后面的范例中会看到这种机制对于自定义视图是非常有用的。但在本例中，我们填充的是 Activity 最顶层的视图，因此这里传递了 `null`。

2.3 监控单击动作

2.3.1 问题

当用户单击某个视图时，应用程序要做出反馈。

2.3.2 解决方案

(API Level 1)

确保视图对象是可单击的，然后添加 `View.OnClickListener` 来处理单击事件，默认情况下，SDK 中的很多控件都是可单击的，例如 `Button`、`ImageButton` 和 `CheckBox`。不过，通过在 XML 中设置 `android:clickable="true"` 或是在代码中调用 `View.setClickable(true)`，任何 `View` 都是可接收单击事件的。

2.3.3 实现机制

为了接收和处理单击事件，需要创建一个 `onClickListener` 并将其关联到视图对象上。在这个示例中，视图对象是一个定义在根布局中的按钮，如下所示：

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

在 `Activity` 代码中，这个按钮是通过它的 `android:id` 值获得的，然后给它加上监听器(参见程序清单 2-18)

程序清单 2-18 给按钮设置监听器

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // 获取按钮对象
    Button myButton = (Button)findViewById(R.id.myButton);
    // 添加监听器
    myButton.setOnClickListener(clickListener);
}

// 处理单击事件的监听对象
View.OnClickListener clickListener = new View.OnClickListener() {
    public void onClick(View v) {
        // 处理单击事件的代码
    }
};
```

提醒：

所有的视图控件都可以设置为可单击。因此，应用程序不一定要使用 `Button` 或 `ImageButton` 来进行交互。事实上，它们也只是 `TextView` 和 `ImageView` 的可单击、可聚焦版本。

(API Level 4)

从 API Level 4 开始，为视图控件添加基本的单击监听器有了更高效的方法。可以在

XML 中设置视图部件的 `android:onClick` 属性，运行时会用 Java 的反射机制在事件发生时调用所需的方法。如果用这种方法修改前面那个示例，按钮的 XML 变为如下内容：

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="My Button"
    android:onClick="onMyButtonClick"
/>
```

这个示例就不再需要设置 `android:id` 属性了，因为该属性的唯一用途就是在代码中添加监听器。这就将 Java 代码简化成程序清单 2-19 所示的样子。

程序清单 2-19 在 XML 中添加监听器

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //这里不需要添加监听器的代码
}

public void onMyButtonClick(View v) {
    //处理单击事件的代码
}
```

提示：

`android:onClick` 的工作原理是在底层创建一个新的 `OnClickListener` 并通过当前上下文反射调用 `onClick` 属性中传入的方法。因此，这些监听器确实可以只设置到 `Activity` 上。如果希望 `Fragment` 或者其他控件能够处理单击事件，需要像之前的方式一样，手动地关联所有监听器。

2.4 适用于多种屏幕分辨率的图形资源

2.4.1 问题

使用 Android 传统的机制在高分辨率的设备上放大图片，呈现的效果非常不理想。

2.4.2 解决方案

(API Level 4)

用资源标识符为每幅图像资源提供多个尺寸。Android SDK 中定义了 4 种屏幕分辨率 (或者叫屏幕密度)，如下所示：

- 低分辨率(ldpi): 120 dpi
- 中分辨率(mdpi): 160 dpi
- 高分辨率(hdpi): 240 dpi

- 超高分辨率(xhdpi): 320 dpi (API Level 8 新增)
- 超级高(xxhdpi): 480 dpi (API Level 16 新增)
 - 主要用于高分辨率、大屏幕设备的启动器图标

默认情况下, Android 项目只有 `res/drawable/` 目录, 其中存放了所有的图像资源。在这种情况下, Android 在中分辨率屏幕上就会按 1:1 的尺寸使用这些图片。当应用程序在高分辨率的屏幕上运行时, Android 会将图片放大到 150%(在 xhdpi 上是 200%), 这样会影响图片的质量。

2.4.3 实现机制

为了避免这个问题, 建议你为每张图片提供多个不同分辨率的副本, 将其放在各自的资源目录中。

- `res/drawable-ldpi/`
 - 尺寸是 mdpi 的 75%
- `res/drawable-mdpi/`
 - 原始的图片尺寸
- `res/drawable-hdpi/`
 - 尺寸是 mdpi 的 150%
- `res/drawable-xhdpi/`
 - 尺寸是 mdpi 的 200%
 - 只有支持 API Level 8 或更高的 API Level 的应用程序才支持
- `res/drawable-xxhdpi/`
 - 尺寸是 mdpi 的 300%
 - 只有支持 API Level 16 的应用程序才支持

同一张图片在各个目录中的名称必须是一样的。例如, 如果在 `AndroidManifest.xml` 中使用的是默认的启动器 icon 值(即 `android:icon="@drawable/icon"`), 那么在项目中就需要下面这些资源:

```
res/drawable-ldpi/icon.png (36×36 像素)
res/drawable-mdpi/icon.png (48×48 像素)
res/drawable-hdpi/icon.png (72×72 像素)
res/drawable-xhdpi/icon.png (96×96 像素, 需要支持)
res/drawable-xxhdpi/icon.png (144×144 像素, 需要支持)
```

Android 会根据设备屏幕的分辨率选择合适的资源, 将其作为应用程序的图标显示在 Launcher 屏幕中, 不需要缩放, 也不会损失图片的质量。在 Android 3.0 上, 系统会自动选择比屏幕配置高一级的分辨率的资源。例如, 在 xhdpi 设备上会使用 xxhdpi 的资源。

另一个示例, 某个 logo 图片可能会在应用程序中的多个地方显示, 在中分辨的设备上是 200×200 像素。应该用资源标识符提供图片的各种尺寸:

```
res/drawable-ldpi/logo.png (150×150 像素)
res/drawable-mdpi/logo.png (200×200 像素)
```

res/drawable-hdpi/logo.png (300×300 像素)

该应用程序不支持超高分辨率, 所以我们只需要提供 3 张图片即可。在引用该资源时, 只要用@drawable/logo(在 XML 中)或 R.drawable.logo(在 Java 代码中), Android 就会根据设备屏幕的分辨率选择合适的资源

2.5 锁定 Activity 方向

2.5.1 问题

应用程序中的某个 Activity 不能旋转, 或是旋转会影响应用程序的某些代码。

2.5.2 解决方案

(API Level 1)

在 AndroidManifest.xml 文件中可以用静态声明将 Activity 的方向锁定为横向或纵向。这个声明只能用于<Activity>标签, 所以不能一次性解决整个应用程序。而在<Activity>元素中加上 android:screenOrientation="portrait"或 android:screenOrientation="landscape", 无论设备处于什么位置, Activity 都会按指定的方向显示。

另一个可以在 XML 中使用的选项是"behind"。如果为 Activity 设置了 android:screenOrientation="behind", Activity 就会跟 Activity 栈中前一个 Activity 的方向保持一致。在 Activity 需要将锁定方向与之前的页面保持一致的场景中, 这是一种设置 Activity 方向的好方法。

2.5.3 实现机制

在程序清单 2-20 中, 示例 AndroidManifest.xml 文件中有 3 个 Activity。其中两个 Activity(MainActivity 和 ResultActivity)的方向被锁定为纵向, 而 UserEntryActivity 则是可旋转的, 因为用户可能需要旋转屏幕及使用物理键盘。

程序清单 2-20 有 Activity 被锁定为纵向的 manifest

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

```

        </intent-filter>
    </activity>
    <activity android:name=".ResultActivity"
        android:screenOrientation="portrait" />
    <activity android:name=".UserEntryActivity" />
</application>
</manifest>

```

2.6 动态方向锁定

2.6.1 问题

在某些特定的条件下，不能让屏幕旋转，但这个条件是临时的或是根据用户的意愿决定的。

2.6.2 解决方案

(API Level 1)

用 Android 的请求方向机制(requested orientation mechanism)，应用程序可以调整显示 Activity 的屏幕方向，将其固定为某个方向或是交由设备决定。这是通过调用 Activity.SetRequestedOrientation()方法实现的，该方法的参数是 ActivityInfo.screenOrientation 属性组中的整数常量。

默认情况下，屏幕的方向设为 SCREEN_ORIENTATION_UNSPECIFIED，也就是由设备决定屏幕的方向。这通常是根据设备的物理方向来确定的。当前的方向可以随时通过调用 Activity.getRequestedOrientation()方法获得。

2.6.3 实现机制

使用旋转锁定按钮

举个例子，让我们创建一个 ToggleButton 实例来控制是否锁定当前屏幕方向，这样就能让用户随时控制 Activity 的方向。

在 main.xml 布局中，可以这样定义一个 ToggleButton：

```

<ToggleButton
    android:id="@+id/toggleButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOff="Lock"
    android:textOn="LOCKED"
/>

```

在 Activity 代码中，我们给这个按钮的状态创建一个监听器，根据按钮的值决定锁定或解锁屏幕的方向(参加程序清单 2-21)。

程序清单 2-21 动态锁定/解锁屏幕方向的 Activity

```

public class LockActivity extends Activity {

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //获得按钮资源的句柄
        ToggleButton toggle = (ToggleButton)findViewById(R.id.toggleButton);
        //在添加监听器之前设置默认状态
        if( getRequestedOrientation() !=
            ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED ) {
            toggle.setChecked(true);
        } else {
            toggle.setChecked(false);
        }
        //将监听器关联到按钮
        toggle.setOnCheckedChangeListener(listener);
    }

    OnCheckedChangeListener listener = new OnCheckedChangeListener() {
        public void onCheckedChanged(CompoundButton buttonView,
            boolean isChecked) {
            int current = getResources().getConfiguration().orientation;
            if(isChecked) {
                switch(current) {
                    case Configuration.ORIENTATION_LANDSCAPE:
                        setRequestedOrientation(
                            ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                        break;
                    case Configuration.ORIENTATION_PORTRAIT:
                        setRequestedOrientation(
                            ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                        break;
                    default:
                        setRequestedOrientation(
                            ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
                }
            } else {
                setRequestedOrientation(
                    ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
            }
        }
    }
}

```

监听器中的这段代码是本攻略的关键。如果用户按下这个按钮，将其置为 ON 状态，应用程序就通过保存 `Resources.getConfiguration()` 的 `orientation` 参数读取当前的屏幕方向。`Configuration` 对象用来表示屏幕方向的常数与所请求的方向使用的常数不同，所以我们根

据当前的屏幕方向进行切换，然后再用合适的常数调用 `setRequestedOrientation()` 方法。

注意：

如果所请求的方向跟当前状态不一样，且 Activity 在前台，Activity 就会立即切换方向以满足请求。

如果用户按下了这个按钮，将其设为 OFF 状态，我们就不再锁定方向，因此用 `SCREEN_ORIENTATION_UNSPECIFIED` 常数再调用一次 `setRequestedOrientation()`，将控制权交还给设备。如果设备当前的方向跟应用程序锁定的方向不一致，也会导致屏幕切换方向。

注意：

改变屏幕的方向不会保持默认的 Activity 生命周期。如果设备配置发生变化(例如物理键盘弹出，或是设备方向改变)，Activity 依然会被销毁并重新创建，因此所有保持 Activity 状态的规则都适用。

2.7 手动处理旋转

2.7.1 问题

在旋转过程中，默认会将 Activity 销毁，然后再重新创建，这会严重影响应用程序的性能。

如果没有自行修改的话，在配置变化时，Android 会结束当前的 Activity 实例，然后重新创建一个适用于新配置的 Activity 实例，这会带来性能上的损失，因为这需要先保存 UI 状态，然后再完全重新构建 UI。

2.7.2 解决方案

(API Level 1)

利用 manifest 文件中的 `android:configChanges` 参数，告诉 Android 某个 Activity 在处理旋转事件时不需要运行时进行干预，这不仅能降低 Android 的工作量，即销毁和重建 Activity 实例，也会降低应用程序的工作量。保持 Activity 实例，应用程序就不必为了保持一致性而花费时间保存和还原应用程序的当前状态。

注册了一个或多个配置变动的 Activity 可以通过 `Activity.onConfigurationChanged()` 回调方法收到通知，在该方法中可以执行各种跟配置变动有关的手动操作。

要完全以手动方式处理旋转，Activity 至少要注册两个配置变动参数：`orientation` 和 `keyboardHidden`。`orientation` 参数注册了设备方向变动的事件。`keyboardHidden` 参数注册了用户滑开或关闭物理键盘的事件，尽管后者看上去跟屏幕旋转没有直接关系，但如果不注册这些事件的话，Android 就会在这些事件发生时重建 Activity，这会使前面手动处理旋转的努力付之东流。

2.7.3 实现机制

将这些参数添加到 AndroidManifest.xml 中的任意一个<Activity>元素，如下所示：

```
<activity android:name=".MyActivity"
android:configChanges="orientation|keyboardHidden" />
```

在一条赋值语句中可以注册多种变动，用“|”符号将它们分开即可。因为这些参数都不能应用于<application>元素，所以每个 Activity 都必须在 AndroidManifest.xml 中注册。

在 Activity 注册后，配置发生改变时就会调用 Activity 的 onConfigurationChanged() 方法。程序清单 2-22 是一个简洁的 Activity 定义，用于处理变动发生时产生的回调。

程序清单 2-22 手动管理旋转的 Activity

```
public class MyActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //必须调用 super
        super.onCreate(savedInstanceState);
        //加载视图资源
        loadView();
    }

    @Override
    public void onConfigurationChanged(Configuration newConfig) {
        //必须调用 super
        super.onConfigurationChanged(newConfig);
        //保存重要的 UI 状态
        saveState();
        //重新加载视图资源
        loadView();
    }

    private void saveState() {
        //实现持久化 UI 状态的代码
    }

    private void loadView() {
        setContentView(R.layout.main);

        //根据新的配置，处理其他必要的 UI 改动
        //包括恢复的状态和保存的状态
    }
}
```

注意：

Google 并不推荐这样处理旋转，除非应用程序的性能确实有这个要求。在响应各个变动事件时，所有跟配置有关的资源都必须手动加载。

Google 建议允许默认的 Activity 重建，除非应用程序的性能对此确实有要求。这主要是因为，阻止默认的 Activity 重建会导致 Android 无法加载在资源条件目录中存放的备选资源(例如 res/layout-land/目录中的横屏资源)。

在示例的 Activity 中，所有用于处理视图布局的代码都被抽象成在 onCreate()和 onConfigurationChanged()中调用的私有方法——loadView()。在该方法中，类似 setContentView()这样的代码可以确保加载与配置相匹配的布局。

调用 setContentView()会重新加载视图，所以一定要在没有诸如 onSaveInstanceState()和 onRestoreInstanceState()等生命周期回调函数的协助下，保存所有的 UI 状态。为了达到该目的，示例中实现了 saveState()方法。

2.8 创建弹出菜单动作

2.8.1 问题

为用户在用户界面中的选择提供多个可供执行的动作。

2.8.2 解决方案

显示 ContextMenu 或 ActionMode 来响应用户的动作。

2.8.3 实现机制

1. ContextMenu

(API Level 1)

使用 ContextMenu 是一个可行的解决方案，尤其适用于根据用户在 ListView 或其他 AdapterView 中单击内容来提供一系列的动作选项的情况。因为 ContextMenu.ContextMenuInfo 对象可以为每个选中的项提供有用信息(例如 id 和 position)，这有助于构建菜单。

首先，在 res/menu/中创建一个 XML 文件，以定义菜单，我们将其命名为 contextmenu.xml(参见程序清单 2-23)。

程序清单 2-23 res/menu/contextmenu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_delete"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="Delete Item"
    />
    <item
        android:id="@+id/menu_edit"
        android:icon="@android:drawable/ic_menu_edit"
```

```

        android:title="Edit Item"
    />
</menu>

```

然后在 Activity 中用 `onCreateContextMenu()` 和 `onContextItemSelected()` 方法填充菜单，处理用户的选择(参见程序清单 2-24)。

程序清单 2-24 使用自定义菜单的 Activity

```

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
    ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    getMenuInflater().inflate(R.menu.contextmenu, menu);
    menu.setHeaderTitle("Choose an Option");
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    //切换选项的 ID, 找到用户选择的动作
    switch(item.getItemId()) {
        case R.id.menu_delete:
            //执行删除动作
            break;
        case R.id.menu_edit:
            //执行编辑动作
            break;
        default:
            return super.onContextItemSelected(item);
    }
    return true;
}

```

为了激活这些回调方法，必须注册将启用该菜单的视图。实际上，就是将视图的 `View.OnCreateContextMenuListener` 注册到当前的 Activity:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //为上下文事件注册一个按钮
    ListView list = new ListView(this);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, ITEMS);
    list.setAdapter(adapter);
    registerForContextMenu(list);

    setContentView(list);
}

```

Android 中很多视图的默认用户行为是在长按屏幕时显示 `ContextMenu` 而不是在单击时显示，因此，在本例中长按 `ListView` 中的条目也会显示选项菜单。

提示:

也可以调用 `Activity.openContextMenu()` 来触发任意视图的 `ContextMenu`，传入之前注册的视图即可。

把上述东西都串在一起，就得到了一个 `Activity`，该 `Activity` 注册了一个显示菜单的按钮，单击该按钮，就会弹出菜单(参见程序清单 2-25)。

程序清单 2-25 使用上下文动作菜单的 `Activity`

```
public class ContextActivity extends Activity {

    private static final String[] ITEMS =
        {"Mom", "Dad", "Brother", "Sister", "Uncle", "Aunt"};

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //为上下文事件注册一个按钮
        ListView list = new ListView(this);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, ITEMS);
        list.setAdapter(adapter);
        registerForContextMenu(list);

        setContentView(list);
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        getMenuInflater().inflate(R.menu.contextmenu, menu);
        menu.setHeaderTitle("Choose an Option");
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        //可以从绑定的 ContextMenuInfo 对象获得单击的条目信息，
        //在 ListView 中它是一个 AdapterContextMenuInfo 实例
        AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
        int listPosition = info.position;

        //切换项目的 id，找到用户选择的动作
        switch(item.getItemId()) {
            case R.id.menu_delete:
                //执行删除动作
                break;
            case R.id.menu_edit:
                //执行编辑动作
                break;
        }
    }
}
```

```

        default:
            return super.onContextItemSelected(item);
        }
        return true;
    }
}

```

当用户做了选择后，可以通过检查传入的 `MenuItem` 来判断相应的动作。另外，这个 `MenuItem` 还会包含一个 `ContextMenuInfo` 对象，该对象中包含了原始列表中选中项的相关数据信息。这在针对条目数据执行真正的请求动作时非常有用。运行结果如图 2-2 所示。

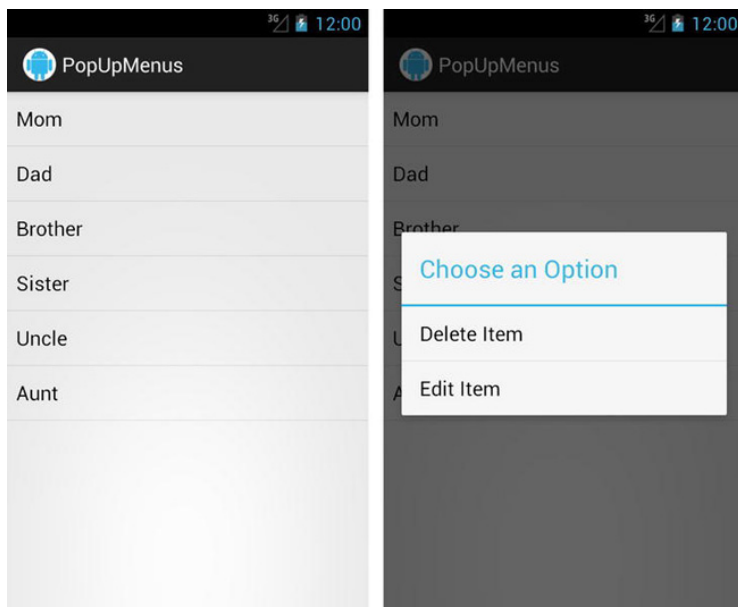


图 2-2 上下文动作菜单

2. ActionMode

(API Level 11)

`ActionMode` API 解决了和 `ContextMenu` 类似的问题，允许用户对 UI 中的某个条目执行某种动作，但它的实现方式稍微有点不同。激活 `ActionMode` 后会在系统的 `ActionBar` 上出现一个包含你提供的菜单选项的叠层，并出现一个额外的可以退出 `ActionMode` 的选项。还允许同时选择多个条目来执行同一个动作。程序清单 2-26 演示了这一功能。

程序清单 2-26 使用了 `Context` `ActionMode` 的 `Activity`

```

public class ActionActivity extends Activity implements
    AbsListView.MultiChoiceModeListener {

    private static final String[] ITEMS =
        {"Mom", "Dad", "Brother", "Sister", "Uncle", "Aunt"};

    private ListView mList;

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //为上下文事件注册一个按钮
    mList = new ListView(this);
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_activated_1, ITEMS);
    mList.setAdapter(adapter);
    //通过上下文 ActionMode 设置这个列表
    mList.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    mList.setMultiChoiceModeListener(this);

    setContentView(mList);
}

@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    //如果 ActionMode 一直是无效的, 可以在这里做些工作来更新菜单
    return true;
}

@Override
public void onDestroyActionMode(ActionMode mode) {
    //退出 ActionMode 时会调用这个方法
}

@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.contextmenu, menu);
    return true;
}

@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    //获得索引选中条目在列表中的位置, 然后进行操作
    SparseBooleanArray items = mList.getCheckedItemPositions();
    //通过条目的 ID 得到用户选择的动作
    switch(item.getItemId()) {
        case R.id.menu_delete:
            //执行删除动作
            break;
        case R.id.menu_edit:
            //执行编辑动作
            break;
        default:
            return false;
    }
    return true;
}

```

```

@Override
public void onItemCheckedStateChanged(ActionMode mode, int position,
    long id, boolean checked) {
    int count = mList.getCheckedItemCount();
    mode.setTitle(String.format("%d Selected", count));
}
}

```

要使用我们的 `ListView` 来激活一个多选择项的 `ActionMode`，需要将 `choiceMode` 属性设置为 `CHOICE_MODE_MULTIPLE_MODAL`。它和传统的 `CHOICE_MODE_MULTIPLE` 不同，`CHOICE_MODE_MULTIPLE` 是在每个条目上都提供一个可选择的控件。在 `ActionMode` 处于激活状态时，它的模式标识只设置了这个选择模式。

对于 `ActionMode`，有很多的回调方法需要去实现，因为它不像 `ContextMenu` 一样是直接内置在 `Activity` 中的。我们需要实现 `ActionMode.Callback` 接口来响应创建菜单和选择选项的事件。`ListView` 有一个名为 `MultiChoiceModeListener` 的特殊接口，它是 `ActionMode.Callback` 的子接口，本例中就实现了这个特殊的接口。

我们在 `onCreateActionMode()` 中的响应和 `onCreateContextMenu()` 类似，只是构建了要显示的菜单选项。这个 `Menu` 不需要包含图标，这时，`ActionMode` 会显示选项名。当每个条目选中后，我们会在 `onItemCheckedStateChanged()` 方法中得到通知。这里，我们会更新 `ActionMode` 的标题来显示当前选中条目的个数。

当用户结束选择并单击一个菜单选项时会调用 `onActionItemClicked()` 方法。因为会有多个条目同时选择，所以我们通过 `getCheckedItemPositions()` 得到所有选择的条目，这样就可以对所有的选择条目进行操作。图 2-3 显示了使用之前的列表时 `ActionMode` 的样子。

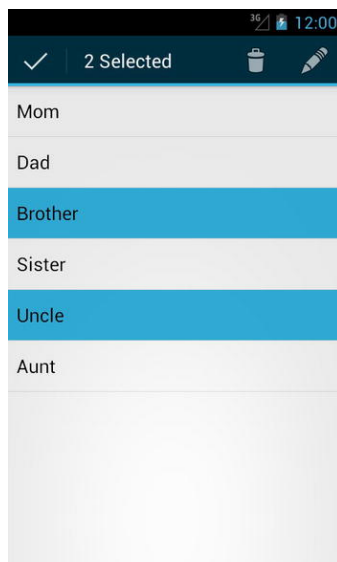


图 2-3 选择了两个选项的 `ActionMode`

2.9 显示一个用户对话框

2.9.1 问题

需要向用户显示一个简单的弹出式对话框来进行事件通知或展示一个选项列表。

2.9.2 解决方案

(API Level 1)

在向用户快速展示重要信息的场景中，`AlertDialog` 是最高效的解决方案。它展示的内

容可以很轻松地进行自定义，同时框架还提供了一个方便的 `AlertDialog.Builder` 类来快速构建弹出式对话框。

2.9.3 实现机制

通过 `AlertDialog.Builder`，可以构建类似的 `AlertDialog` 但包含不同的额外选项。`AlertDialog` 在创建简单弹出式对话框来获得用户反馈方法时是一个非常通用的类。通过 `AlertDialog.Builder`，可以很容易地在一个控件中添加单选或多选列表、按钮和消息字符串。

为了说明这一点，让我们用 `AlertDialog` 创建一个和以前一样的弹出式选择框。这一次，我们将在选项列表的底部增加一个 `Cancel` 按钮(参见程序清单 2-27)。

程序清单 2-27 使用 `AlertDialog` 的动作菜单

```
public class DialogActivity extends Activity
    implements DialogInterface.OnClickListener, View.OnClickListener {

    private static final String[] ZONES = {"Pacific Time", "Mountain Time",
        "Central Time", "Eastern Time", "Atlantic Time"};

    Button mButton;
    AlertDialog mActions;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("Activity");
        mButton = new Button(this);
        mButton.setText("Click for Time Zones");
        mButton.setOnClickListener(this);

        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Select Time Zone");
        builder.setItems(ZONES, this);
        //这里的取消动作只会让对话框消失掉，但在用户单击 Cancel 按钮时，也可以添加一个
        //监听器来处理一些其他的操作
        builder.setNegativeButton("Cancel", null);
        mActions = builder.create();

        setContentView(mButton);
    }

    //这里处理列表的选择事件
    @Override
    public void onClick(DialogInterface dialog, int which) {
        String selected = ZONES[which];
        mButton.setText(selected);
    }
}
```

```

//这里处理 Button 的单击事件(弹出对话框)
@Override
public void onClick(View v) {
    mActions.show();
}
}

```

本例中, 我们创建了一个新的 `AlertDialog.Builder` 实例并使用它的便捷方法添加了如下内容:

- 标题, 通过 `setTitle()` 设置
- 选项列表, 通过 `setItems()` 和一个字符串数组(也可以是数组资源)设置
- Cancel 按钮, 通过 `setNegativeButton()` 设置

当选择列表中的一个选项时, 我们关联到列表项的监听器会返回所选择的选项(索引是之前提供的数组的索引, 从 0 开始), 然后我们就可以根据用户的选择来更新按钮上的文本信息。对于 Cancel 按钮, 因为我们只想 Cancel 时关闭对话框, 所以我们为 Cancel 按钮传入的监听器为 `null`。如果在按下 Cancel 时还想处理一些重要的工作, 则可以在 `setNegativeButton()` 方法中传入其他的监听器。

另外, 还有其他的一些选项可以设置除了选项列表之外的对话框中的内容。

- `setMessage()` 可以为对话框的内容设置一条简单的文本信息。
- `setSingleChoiceItems()` 和 `setMultiChoiceItems()` 会创建一个和本例类似的列表, 但每个选项是以复选框的方式显示的。
- `setView()` 可以为对话框的内容使用任意自定义视图。

现在, 按钮按下后的效果如图 2-4 所示。

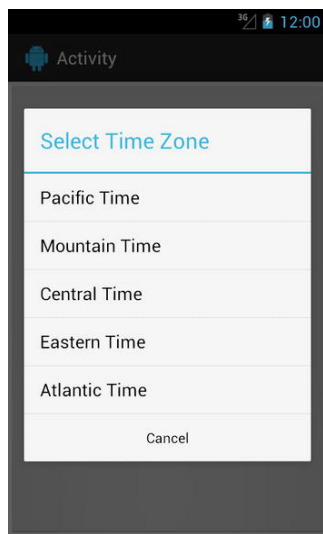


图 2-4 带有选项列表的 AlertDialog

自定义列表选项

`AlertDialog.Builder` 允许传入一个自定义的 `ListAdapter` 来作为对话框中显示列表的数据源, 这也就意味着我们可以自定义列表中每一行的布局来显示更加详细的信息。在程序清单 2-28 和 2-29 中我们改进了之前的示例, 为每一行使用一个自定义的行布局来显示其他信息。

程序清单 2-28 res/layout/list_item.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"

```

```

android:minHeight="?android:attr/listPreferredItemHeight">
<TextView
    android:id="@+id/text_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:textAppearance="?android:attr/textAppearanceMedium"/>
<TextView
    android:id="@+id/text_detail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"
    android:textAppearance="?android:attr/textAppearanceSmall"/>
</RelativeLayout>

```

程序清单 2-29 自定义布局的 AlertDialog

```

public class CustomItemActivity extends Activity
    implements DialogInterface.OnClickListener, View.OnClickListener {

    private static final String[] ZONES = {"Pacific Time", "Mountain Time",
        "Central Time", "Eastern Time", "Atlantic Time"};
    private static final String[] OFFSETS =
        {"GMT-08:00", "GMT-07:00", "GMT-06:00", "GMT-05:00", "GMT-04:00"};

    Button mButton;
    AlertDialog mActions;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("Activity");
        mButton = new Button(this);
        mButton.setText("Click for Time Zones");
        mButton.setOnClickListener(this);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            R.layout.list_item) {
            @Override
            public View getView(int position, View convertView, ViewGroup parent) {
                View row = convertView;
                if(row == null) {
                    row = getLayoutInflater().inflate(R.layout.list_item,
                        parent, false);
                }

                TextView name = (TextView) row.findViewById(R.id.text_name);
                TextView detail = (TextView) row.findViewById(R.id.text_detail);
                name.setText(ZONES[position]);
                detail.setText(OFFSETS[position]);
            }
        }
    }
}

```

```

        return row;
    }

    @Override
    public int getCount() {
        return ZONES.length;
    }
};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Select Time Zone");
builder.setAdapter(adapter, this);
//这里的取消动作只会让对话框消失,但在用户单击 Cancel 按钮时,也可以添加一个
//监听器来处理一些其他的操作
builder.setNegativeButton("Cancel", null);
mActions = builder.create();

setContentView(mButton);
}

//这里处理列表的选择事件
@Override
public void onClick(DialogInterface dialog, int which) {
    String selected = ZONES[which];
    mButton.setText(selected);
}

//这里处理 Button 的单击事件(弹出对话框)
@Override
public void onClick(View v) {
    mActions.show();
}
}

```

这里我们为 builder 提供了一个 ArrayAdapter 而不是简单地传入选项数组。ArrayAdapter 对 getView() 进行了自定义的实现,会返回一个 XML 中自定义的布局,其中显示了两个文本标签:一个靠左对齐,另一个靠右对齐。通过这个自定义布局,我们现在可以显示 GMT 偏移值和时区名称。本章的后面还会进一步讨论自定义 adapter 的特性。图 2-5 显示了新的、更加实用的弹出式对话框。

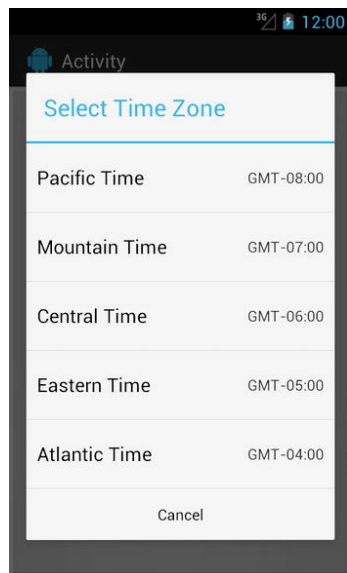


图 2-5 自定义选项的 AlertDialog

2.10 自定义选项菜单

2.10.1 问题

应用程序需要为用户提供一个动作集，但又不想占用视图结构的屏幕空间。

2.10.2 解决方案

(API Level 1)

使用框架中的选项菜单功能来提供一个弹出式动作菜单供用户选择。根据平台版本的不同，Android 的菜单功能也有所不同。在早期的版本中，所有的 Android 都有一个物理 MENU 键可以触发这个功能。从 Android 3.0 开始，出现了没有物理按钮的设备，菜单功能也变成了 ActionBar 的一部分。

尽管有所不同，但两个版本都使用相同的选项菜单 API(Activity 的一部分)，因此应用程序的代码不必再判断 Android 版本。

2.10.3 实现机制

程序清单 2-30 定义了将在 XML 中使用的选项菜单。

程序清单 2-30 res/menu/options.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_add"
        android:title="Add Item"
        android:icon="@android:drawable/ic_menu_add"
        android:showAsAction="always" />
    <item android:id="@+id/menu_remove"
        android:title="Remove Item"
        android:icon="@android:drawable/ic_menu_delete"
        android:showAsAction="always" />
    <item android:id="@+id/menu_edit"
        android:title="Edit Item"
        android:icon="@android:drawable/ic_menu_edit"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/menu_settings"
        android:title="Settings"
        android:icon="@android:drawable/ic_menu_preferences"
        android:showAsAction="never" />
</menu>
```

title 和 icon 属性定义了每个选项该如何显示，旧版本平台会显示这两个值，而新版本中会显示一个或根据位置显示另一个。只有 Android 3.0 及以后的设备可以识别 showAsAction 属性，该属性定义了选项是否应该变成 ActionBar 中的一个动作或者放到更多菜单中。这

个属性最常用的值如下：

- **always**: 总是作为一个动作显示其图标
 - **never**: 总是显示在更多菜单中并显示其名字
 - **ifRoom**: 如果 **ActionBar** 上有空间，就作为一个动作显示，否则显示在更多菜单中
- 程序清单 2-31 演示了如何将这个菜单关联到一个 **Activity** 上。

程序清单 2-31 覆写菜单动作的 **Activity**

```
public class OptionsActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        //在这个回调中创建菜单并做一些初始化设置
        getMenuInflater().inflate(R.menu.options, menu);
        return true;
    }

    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        //在这个回调中处理每次打开菜单时的相应设置
        return super.onPrepareOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        //通过 id 获得选中的选项
        switch (item.getItemId()) {
            case R.id.menu_add:
                //执行添加动作
                break;
            case R.id.menu_remove:
                //执行移除动作
                break;
            case R.id.menu_edit:
                //执行编辑动作
                break;
            case R.id.menu_settings:
                //执行设置动作
                break;
            default:
                break;
        }

        return true;
    }
}
```

```

    }
}

```

当用户按下设备上的 MENU 键后(或者含有 ActionBar 的 Activity),就会调用 `onCreateOptionsMenu()`方法来构建菜单。有一个名为 `MenuInflater` 的特殊 `LayoutInflater` 对象可以用来根据 XML 创建菜单。这里我们使用 `Activity` 中已有的 `getMenuInflater()` 方法获得 `MenuInflater` 的实例来创建 XML 菜单。

如果需要在用户每次打开菜单时传递一些动作,可以在 `onPrepareOptionsMenu()`中完成。这里有一个建议,就是所有变成 `ActionBar` 中的动作在用户选择它们时将不会触发这个回调方法,但在更多菜单中的动作还会触发该方法。

用户做了选择后, `onOptionsItemSelected()`回调方法将会触发同时传入选择的菜单选项。因为我们在 XML 中为每个选项都定义了唯一的 ID,所以可以通过 `switch` 语句判断用户的选项并进行相应的操作。

图 2-6 显示了不同版本和配置的设备上这个菜单的样子。Android 3.0 之前的设备上会在屏幕底部显示一个悬浮的全菜单。依然带有物理按键的新设备会在 `ActionBar` 上显示各个动作,但更多菜单还是通过 MENU 键触发的。最后,带有软键的设备会挨着 `ActionBar` 显示一个更多菜单的按钮。

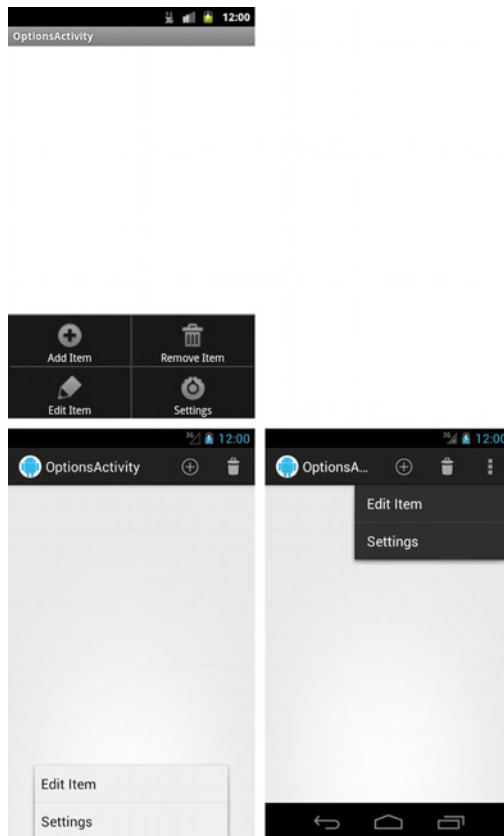


图 2-6 选项菜单分别在 Android 2.3 上(上边)、带有物理按键的 Android 4.1 设备上(左边)和带有软键的 Android 4.0 设备上

2.11 自定义返回按键

2.11.1 问题

应用程序要以自己的方式来处理用户按下 BACK 按键后的行为。

2.11.2 解决方案

(API Level 5)

可以在 Activity 中使用 `onBackPressed()` 回调方法或者在 Fragment 中操作回退栈。

2.11.3 实现机制

如果想要用户在你的 Activity 上按下 BACK 时可以得到相应的通知，可以覆写 `onBackPressed()` 方法，如下所示：

```
@Override
public void onBackPressed() {
    //实现自定义返回功能

    //调用 super 进行常规的处理(例如销毁 Activity)
    super.onBackPressed();
}
```

这个方法的默认实现会将当前回退栈中的 fragment 弹出并且销毁 Activity。如果你打算改变这个流程，只需要调用父类的实现来保持这种常规的处理方式。

警告：

重载物理按键事件时应保持谨慎。在这个 Android 系统中，所有的物理按键都有一致的功能，如果这些按键的功能变化太大，会让用户感到困惑和不满。

fragment 的 BACK 操作

当 UI 中包含 fragment 时，可以进一步自定义设备 BACK 键的行为。默认情况下，在 UI 中添加或替换 fragment 的操作并不会再在回退栈中添加相应的 fragment，因此当用户按下 BACK 键后，并不会回到之前的界面。但是，所有 `FragmentTransaction` 都可以作为实体通过简单地调用 `addToBackStack()` (在事务提交前)被添加到回退栈中。

默认情况下，当用户按下 BACK 后，Activity 会调用 `FragmentManager.popBackStackImmediate()`，这样每个通过这种方式添加的 `FragmentTransaction` 都会在每次单击时弹出，直到栈中一个不剩，然后 Activity 会被销毁。另外，这个方法还有一些变种，允许可以直接跳到栈中的某个位置。让我们看一下程序清单 2-32 和 2-33。

程序清单 2-32 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Go Home"
        android:onClick="onHomeClick" />
    <FrameLayout
        android:id="@+id/container_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>

```

程序清单 2-33 自定义 fragment 回退栈的 Activity

```

public class MyActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //构建 fragment 的回退栈
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.container_fragment, MyFragment.newInstance("First Fragment"));
        ft.commit();

        ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.container_fragment, MyFragment.newInstance("Second Fragment"));
        ft.addToBackStack("second");
        ft.commit();

        ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.container_fragment, MyFragment.newInstance("Third Fragment"));
        ft.addToBackStack("third");
        ft.commit();

        ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.container_fragment, MyFragment.newInstance("Fourth Fragment"));
        ft.addToBackStack("fourth");
        ft.commit();
    }

    public void onHomeClick(View v) {
        getSupportFragmentManager().popBackStack("second",
            FragmentManager.POP_BACK_STACK_INCLUSIVE);
    }
}

```

```

public static class MyFragment extends Fragment {
    private CharSequence mTitle;

    public static MyFragment newInstance(String title) {
        MyFragment fragment = new MyFragment();
        fragment.setTitle(title);

        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        TextView text = new TextView(getActivity());
        text.setText(mTitle);
        text.setBackgroundColor(Color.WHITE);

        return text;
    }

    public void setTitle(CharSequence title) {
        mTitle = title;
    }
}

```

注意:

这里我们使用了支持库, 允许 Android 3.0 之前的版本使用 fragment。如果应用程序的目标 API 为 11 或更高版本, 可以将 `FragmentManager` 用 `Activity` 替换, 将 `getSupportFragmentManager()` 用 `getFragmentManager()` 替换。

这个示例会向栈中放入 4 个自定义的 `Fragment` 实例, 所以在应用程序运行时会显示最后添加的那个 `Fragment` 实例。对于每个事务, 我们调用了 `addToBackStack()`, 同时传入一个标记名称来标识这个事务。如果你不想直接跳转到栈中某个位置, 可以不需要以上这步, 直接传入 `null` 即可。每次按下 `BACK` 按钮后, 会移除一个 `Fragment` 实例, 直到只剩下第一个 `Fragment` 实例, 这时再按, `Activity` 就会被正常销毁。

注意, 第一个事务并没有被添加到栈中, 这是因为我们希望第一个 `fragment` 作为根视图。如果将它也加入到回退栈中, 会导致它在 `Activity` 销毁前被弹出栈, 这对 UI 会有空白状态出现。

这个应用程序还有 “Go Home” 按钮, 它可以让用户无论在哪个界面都可以立即回到根 `Fragment`。这是通过调用 `FragmentManager` 的 `popBackStack()` 方法, 同时传入希望跳转的事务的标识名称实现的。我们还传入了 `POP_BACK_STACK_INCLUSIVE` 标识来告诉管理器在栈中移除我们标识的事务。如果没有这个标识, 这个示例就会跳转到第 2 个 `fragment`, 而不是根视图。

注意:

Android 会跳转到第一个匹配给定标识的事务中。如果同一个标识被使用多次,则会跳到第一个添加的事务中,而不是最新的事务。

我们不能使用这个方法直接跳到根视图,因为我们无法引用那个事务在回退栈中的标识。这个方法还有一个使用唯一事务 ID(ID 为 `FragmentManager` 的 `commit()` 方法的返回值)的版本。使用这个方法,无需标识也可以直接跳到根视图。

2.12 模拟 Home 按键

2.12.1 问题

应用程序需要实现与按下物理 HOME 按键一样的功能。

2.12.2 解决方案

(API Level 1)

用户按下 HOME 按键的行为会发送一个 `Intent` 给系统,要求系统加载 Home Activity。这与在应用程序中启动其他的 Activity 并没有什么区别,你要做的就是构建合适的 `Intent` 实现该效果。

2.12.3 实现机制

把下面这几行代码添加到 Activity 中要实现该功能的地方:

```
Intent intent = new Intent(Intent.ACTION_MAIN);
intent.addCategory(Intent.CATEGORY_HOME);
startActivity(intent);
```

该功能的一个常见用途就是重载返回键,让用户按下返回键时直接返回主屏幕而不是回到前一个 Activity。当要保护前台 Activity 之前的 Activity(例如登录界面)时,这是很有用的。如果执行返回键的默认行为,就有可能让用户在未授权的情况下访问系统。下面这个示例利用前两个范例中介绍的技术,实现了在 Activity 中按下 BACK 键时直接显示主屏幕的行为。

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    if(keyCode == KeyEvent.KEYCODE_BACK) {
        Intent intent = new Intent(Intent.ACTION_MAIN);
        intent.addCategory(Intent.CATEGORY_HOME);
        startActivity(intent);
        return true;
    }
    return super.onKeyDown(keyCode, event);
}
```

2.13 监控 TextView 的变动

2.13.1 问题

应用程序需要持续监控 TextView 控件(类似于 EditText)中文本内容的变动情况。

2.13.2 解决方案

(API Level 1)

实现 android.text.TextWatcher 接口。TextWatcher 提供了 3 个文本更新过程中的回调方法：

```
public void beforeTextChanged(CharSequence s, int start, int count, int after);
public void onTextChanged(CharSequence s, int start, int before, int count);
public void afterTextChanged(Editable s);
```

beforeTextChanged()和 onTextChanged()方法主要用于提供提示功能，因为这两个方法实际上都无法修改 CharSequence。如果要截获视图中输入的文本，当 afterTextChanged()方法被调用时文本就有可能发生了变化。

2.13.3 实现机制

调用 TextView.addTextChangedListener()方法将 TextWatcher 注册到 TextView。注意，根据语法，可以将多个 TextWatcher 注册到一个 TextView。

1. 字符计算示例

TextWatcher 的一个简单应用是创建 EditText 的字数计算器，能随着用户的输入和删除操作实时显示其中的字数。程序清单 2-34 中的示例就实现了这样一个 TextWatcher，注册了一个 EditText 控件并在 Activity 的标题上显示字符数。

程序清单 2-34 字符计算器 Activity

```
public class MyActivity extends Activity implements TextWatcher {

    EditText text;
    int textCount;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //创建 EditText 控件并添加监控器
        text = new EditText(this);
        text.addTextChangedListener(this);

        setContentView(text);
    }
}
```

```

    }

    /*实现 TextWatcher 的方法*/
    public void beforeTextChanged(CharSequence s, int start, int count,
        int after) { }

    public void onTextChanged(CharSequence s, int start, int before, int count) {
        textCount = text.getText().length();
        setTitle(String.valueOf(textCount));
    }

    public void afterTextChanged(Editable s) { }
}

```

因为不需要修改用户正在输入的文本，所以可以调用 `onTextChanged()` 方法获得字数，只要用户输入的文本发生变化就会回调该方法。其他的方法暂时没用，留空即可。

2. 货币符号格式化示例

SDK 中有一些很方便的预定义的用于格式化文本输入的 `TextWatcher` 实例。`PhoneNumberFormattingTextWatcher` 就是其中之一。这些实例的任务就是在用户输入时将其按标准格式化，减少输入规范化数据的击键次数。

在程序清单 2-35 中，我们创建了一个 `CurrencyTextWatcher`，在 `TextView` 中插入货币符号和分隔点。

程序清单 2-35 货币符号格式器

```

public class CurrencyTextWatcher implements TextWatcher {

    boolean mEditing;

    public CurrencyTextWatcher() {
        mEditing = false;
    }

    public synchronized void afterTextChanged(Editable s) {
        if(!mEditing) {
            mEditing = true;

            //带符号
            String digits = s.toString().replaceAll("\\D", "");
            NumberFormat nf = NumberFormat.getCurrencyInstance();
            try{
                String formatted = nf.format(Double.parseDouble(digits)/100);
                s.replace(0, s.length(), formatted);
            } catch (NumberFormatException nfe) {
                s.clear();
            }
        }
    }
}

```

```

        mEditing = false;
    }
}

public void beforeTextChanged(CharSequence s, int start, int count,
    int after) { }

public void onTextChanged(CharSequence s, int start, int before, int count) { }
}

```

注意:

在 `afterTextChanged()` 方法中修改 `Editable` 的值会导致 `TextWatcher` 方法被调用(因为你刚刚修改了文本)。考虑到这一点, 在实现 `TextWatcher` 时, 应该通过某个逻辑变量或是其他跟踪机制来判断文本修改的来源, 避免产生无限循环。

我们可以将这个自定义的文本格式化器应用在 `Activity` 中的 `EditText` 上(参见程序清单 2-36)。

程序清单 2-36 使用货币符号格式化器的 `Activity`

```

public class MyActivity extends Activity {

    EditText text;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        text = new EditText(this);
        text.addTextChangedListener(new CurrencyTextWatcher());
        setContentView(text);
    }
}

```

有了这个格式化器, 就可以很方便地在 XML 中定义 `EditText` 的 `android:inputType` 和 `android:digits` 属性来格式化用户的输入, 从而保护字段不受输入错误的影响。在这里, 为 `EditText` 加上 `android:digits="0123456789."`(注意末尾的小数点), 不仅能保护应用程序, 对用户也有好处。

2.14 自动滚动的 `TextView`

2.14.1 问题

要创建能在屏幕上滚动显示内容的“走马灯”。

2.14.2 解决方案

(API Level 1)

用 `TextView` 内置的 `marquee` 特性。当 `TextView` 中的内容太长超出边界时，默认会对文本进行裁剪。可以用 `android:ellipsize` 属性配置裁剪行为，可用的选项如下：

- `none`
 - 默认值
 - 从末尾裁剪文本，没有视觉标记
- `start`
 - 从开头裁剪文本，在视图开头标记一个省略号
- `middle`
 - 从中间裁剪文本，在视图中间标记一个省略号
- `end`
 - 从末尾裁剪文本，在视图末尾标记一个省略号
- `marquee`
 - 不添加省略号，当被选中时滚动显示文本

注意：

只有当 `TextView` 被选中时，`marquee` 才能滚动显示文本，仅设置 `android:ellipsize` 属性是不会让视图滚动显示的。

2.14.3 实现机制

为创建不断循环滚动显示文本的“走马灯”，要在 XML 布局中添加一个 `TextView`。如下所示：

```
<TextView
    android:id="@+id/ticker"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:scrollHorizontally="true"
    android:ellipsize="marquee"
    android:marqueeRepeatLimit="marquee_forever"
/>
```

最后 4 个属性是配置视图的关键。没有 `android:singleLine` 和 `android:scrollHorizontally`，`TextView` 就不能正确地显示超长文本(这是实现“走马灯”效果的关键)。设置 `android:ellipsize` 和 `android:marqueeRepeatLimit` 实现文字的不断滚动。可以将重复的次数设置为任何正数，在达到重复滚动指定的次数后文字停止滚动。

在 XML 中正确设置了 `TextView` 的属性后，Java 代码还必须将其设为选中状态，这样才能激活文字滚动动画：

```
TextView ticker = (TextView)findViewById(R.id.ticker);
ticker.setSelected(true);
```

如果要通过用户界面中的事件来控制动画的开始和停止，每次只需要用 `true` 或 `false` 调用 `setSelected()`方法即可。

2.15 动画视图

2.15.1 问题

要让视图对象运动起来，实现变化或其他特效。

2.15.2 解决方案

(API Level 1)

`Animation` 对象可以用于任何视图，通过 `View.startAnimation()`方法启动该对象，这会立即运行动画。还可以用 `View.setAnimation()`设计动画并将其赋给视图，但并不立即执行。使用 `Animation` 必须指定好启动时间参数。通过这个 API，会修改视图在屏幕上短暂显示的样子，而不会修改视图本身。

(API Level 12)

`ObjectAnimator` 的实例，例如 `ViewPropertyAnimator`，可以用来操作 `View` 对象的属性，例如视图的位置或角度。`ViewPropertyAnimator` 是通过 `View.animate()`获得的，然后根据动画的特征进行修改。通过这个 API 进行的修改会影响到视图本身的真实属性。

2.15.3 实现机制

1. 系统动画

为了方便开发人员，`Android SDK` 中提供了一些可以应用于视图的转换动画，可以在运行时用 `AnimationUtils` 类加载这些动画：

- 滑入和渐显
 - `AnimationUtils.makeInAnimation()`
 - 用布尔参数决定滑入的方向是左侧还是右侧
- 向上滑入和渐显
 - `AnimationUtils.makeInChildBottomAnimation()`
 - 视图总是从屏幕的底部向上滑入
- 滑出和渐隐
 - `AnimationUtils.makeOutAnimation()`
 - 用布尔参数决定滑入的方向是左侧还是右侧
- 渐隐

- AnimationUtils.loadAnimation()
- 将 int 参数设为 android.R.anim.fade_out
- 渐显
 - AnimationUtils.loadAnimation()
 - 将 int 参数设为 android.R.anim.fade_in

注意:

这些变换动画只会临时改变视图的显示方式。如果要永久添加或删除某个视图对象,还必须设置视图的 visibility 参数。

程序清单 2-37 通过按钮的单击事件以动画的形式实现了视图的显现和消失。

程序清单 2-37 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/toggleButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Click to Toggle"
    />
    <View
        android:id="@+id/theView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#AAA"
    />
</LinearLayout>
```

在程序清单 2-38 中,用户每次单击按钮,其下方的灰色视图就会以动画的形式出现或消失。

程序清单 2-38 实现视图过渡动画的 Activity

```
public class AnimateActivity extends Activity implements View.OnClickListener {

    View viewToAnimate;

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

        Button button = (Button)findViewById(R.id.toggleButton);
```

```

        button.setOnClickListener(this);

        viewToAnimate = findViewById(R.id.theView);
    }

    @Override
    public void onClick(View v) {
        if(viewToAnimate.getVisibility() == View.VISIBLE) {
            //如果视图已经是可见的,就从右侧将其滑出
            Animation out = AnimationUtils.makeOutAnimation(this, true);
            viewToAnimate.startAnimation(out);
            viewToAnimate.setVisibility(View.INVISIBLE);
        } else {
            //如果视图是隐藏的,立即渐显
            Animation in = AnimationUtils.loadAnimation(this,
                android.R.anim.fade_in);
            viewToAnimate.startAnimation(in);
            viewToAnimate.setVisibility(View.VISIBLE);
        }
    }
}

```

视图的消失方式是从屏幕右侧滑出,而出现的方式则是简单地渐显。我们在这里选择一个简单的视图来说明任何 UI 元素(因为它们都是 **View** 的子类)都可以像这样运动起来。

2. 自定义动画

通过缩放、旋转、变换创建自定义动画,给视图加上特效,为用户提供更炫的用户界面。在 Android 中能创建以下动画元素:

- **AlphaAnimation**
 - 以动画的定时改变视图的透明度。
- **RotateAnimation**
 - 以动画的形式改变视图的旋转角度。
 - 旋转的中心点是可配置的,默认是左上角。
- **ScaleAnimation**
 - 以动画的形式改变视图的缩放比例(大小)。
 - 缩放的中心点是可配置的,默认是左上角。
- **TranslateAnimation**
 - 以动画的形式改变视图的位置。

下面通过一个示例查看如何构建和添加自定义动画对象,在这个示例应用程序中会实现图片的“掷硬币”效果(参见程序清单 2-39 和程序清单 2-40)

程序清单 2-39 res/layout/main.xml

```

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

```

```

        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <ImageView
            android:id="@+id/flip_image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
        />
    </RelativeLayout>

```

程序清单 2-40 带自定义动画的 Activity

```

public class Flipper extends Activity {

    boolean isHeads;
    ScaleAnimation shrink, grow;
    ImageView flipImage;

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

        flipImage = (ImageView)findViewById(R.id.flip_image);
        flipImage.setImageResource(R.drawable.heads);
        isHeads = true;

        shrink = new ScaleAnimation(1.0f, 0.0f, 1.0f, 1.0f,
                                    ScaleAnimation.RELATIVE_TO_SELF, 0.5f,
                                    ScaleAnimation.RELATIVE_TO_SELF, 0.5f);
        shrink.setDuration(150);
        shrink.setAnimationListener(new Animation.AnimationListener() {
            @Override
            public void onAnimationStart(Animation animation) {}

            @Override
            public void onAnimationRepeat(Animation animation) {}

            @Override
            public void onAnimationEnd(Animation animation) {
                if(isHeads) {
                    isHeads = false;
                    flipImage.setImageResource(R.drawable.tails);
                } else {
                    isHeads = true;
                    flipImage.setImageResource(R.drawable.heads);
                }
                flipImage.startAnimation(grow);
            }
        });
        grow = new ScaleAnimation(0.0f, 1.0f, 1.0f, 1.0f,

```

```

        ScaleAnimation.RELATIVE_TO_SELF, 0.5f,
        ScaleAnimation.RELATIVE_TO_SELF, 0.5f);
    grow.setDuration(150);
}
@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_DOWN) {
        flipImage.startAnimation(shrink);
        return true;
    }
    return super.onTouchEvent(event);
}
}

```

这个示例由以下几个组件构成：

- 两张图片资源，硬币的正反两面(文件名是 heads.png 和 tails.png)。
 - 图片可以是 res/drawable 中的任何两张图片资源。ImageView 默认会显示硬币正面的图片。
- 两个 ScaleAnimation 对象。
 - Shrink：以图片中心为缩放中心点，将图片宽度从最大缩小到 0。
 - Grow：以图片中心为缩放中心点，将图片宽度逐渐放大到最大。
- 依序连续有两个动画的匿名监听器 AnimationListener。

可以在 XML 或代码中定义自定义动画对象。下一节会介绍如何在 XML 资源中创建动画，这里用下面的构造函数创建这两个 ScaleAnimation 对象：

```

ScaleAnimation(
    float fromX,
    float toX,
    float fromY,
    float toY,
    int pivotXType,
    float pivotXValue,
    int pivotYType,
    float pivotYValue
)

```

前 4 个参数是横向和纵向的缩放比例。注意示例中的 X 的缩放范围总是 100%~0%，而 Y 则总是保持 100%不变。

其他的参数设定了动画运行时视图的锚点(anchor point)。本例中，应用程序以视图的中点为锚点，在视图缩小时，整个视图从两侧向中间缩小。放大图片也是如此：中间保持不变，图片向其原来边缘扩大。

Android 本身并没有提供将多个动画对象按序链接的功能，因此要通过 Animation.AnimationListener 来实现这个目的。这个监听器中的方法会在动画开始、重播和结束时被调用。本例中会用到最后一个方法，在缩小动画结束时，可以自动启动放大的动画。

这个示例中的最后一个方法是 setDuration()，设置动画的时长。这里值的单位是毫秒，

所以我们的硬币翻转动画总时长是 300ms，每个 ScaleAnimation 的时长是 150ms。

3. AnimationSet

经常会遇到这种情况：所需的动画需要将前面介绍的多种基本动画组合使用，这就是 AnimationSet 大展身手的地方。AnimationSet 定义了一组按序播放的动画。默认情况下，所有的动画都会一起开始，然后根据各自的时长结束。

本节会介绍如何用 Android 偏好的 XML 资源来定义自定义动画。XML 动画应该在项目的 res/anim/文件夹中。其中支持下面的标签，这些标签可以用于动画的根节点或子节点。

- <alpha>: 一个 AlphaAnimation 对象
- <rotate>: 一个 RotateAnimation 对象
- <scale>: 一个 ScaleAnimation 对象
- <translate>: 一个 TranslateAnimation 对象
- <set>: 一个 AnimationSet 对象

不过，其中只有<set>标签可以包含其他的动画标签。

在这个示例中，我们继续使用前面制作的硬币翻转动画，为之设计新的动画效果。将两个 ScaleAnimation 对象跟一个 TranslateAnimation 对象组合成一个集合。要实现的效果是图片在屏幕上“翻转”。为了实现该目的，在两个 XML 文件中定义了这个动画，将它们放在 res/anim/目录中，参见程序清单 2-41 和程序清单 2-42。首先是 grow.xml。

程序清单 2-41 res/anim/grow.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:duration="150"
        android:fromXScale="0.0"
        android:toXScale="1.0"
        android:fromYScale="1.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
    />
    <translate
        android:duration="150"
        android:fromXDelta="0%"
        android:toXDelta="0%"
        android:fromYDelta="50%"
        android:toYDelta="0%"
    />
</set>
```

接下来是 shrink.xml。

程序清单 2-42 res/anim/shrink.xml

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

```

<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:duration="150"
        android:fromXScale="1.0"
        android:toXScale="0.0"
        android:fromYScale="1.0"
        android:toYScale="1.0"
        android:pivotX="50%"
        android:pivotY="50%"
    />
    <translate
        android:duration="150"
        android:fromXDelta="0%"
        android:toXDelta="0%"
        android:fromYDelta="0%"
        android:toYDelta="50%"
    />
</set>

```

在 XML 文件中定义缩放比例跟之前在构造函数中定义缩放比例没有区别。唯一要注意的就是 `pivot` 参数单位的定义方式。所有动画的维度都可以根据 XML 的语法定义成 `ABSOLUTE`、`RELATIVE_TO_SELF` 或 `RELATIVE_TO_PARENT`

- `ABSOLUTE`: 用浮点数表示真实的像素值(例如, “5.0”)。
- `RELATIVE_TO_SELF`: 用 0~100 的百分比表示(例如, “50%”)。
- `RELATIVE_TO_PARENT`: 用带后缀的百分比表示(例如, “25%p”)。

在定义了这些动画文件后,可以修改前面的示例来加载这些 `AnimationSet` 对象(参见程序清单 2-43 和程序清单 2-44)。

程序清单 2-43 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView
        android:id="@+id/flip_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
    />
</RelativeLayout>

```

程序清单 2-44 使用 `AnimationSet` 的 Activity

```

public class Flipper extends Activity {

    boolean isHeads;
    Animation shrink, grow;
    ImageView flipImage;

```

```

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

    flipImage = (ImageView)findViewById(R.id.flip_image);
    flipImage.setImageResource(R.drawable.heads);
    isHeads = true;

    shrink = AnimationUtils.loadAnimation(this, R.anim.shrink);
    shrink.setAnimationListener(new Animation.AnimationListener() {
        @Override
        public void onAnimationStart(Animation animation) {}

        @Override
        public void onAnimationRepeat(Animation animation) {}

        @Override
        public void onAnimationEnd(Animation animation) {
            if(isHeads) {
                isHeads = false;
                flipImage.setImageResource(R.drawable.tails);
            } else {
                isHeads = true;
                flipImage.setImageResource(R.drawable.heads);
            }
            flipImage.startAnimation(grow);
        }
    });
    grow = AnimationUtils.loadAnimation(this, R.anim.grow);
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_DOWN) {
        flipImage.startAnimation(shrink);
        return true;
    }
    return super.onTouchEvent(event);
}
}

```

得到的效果就是翻转的硬币，每次翻转时还沿屏幕的 Y 轴上下滑动。

4. ViewPropertyAnimator

(API Level 12)

从 Android 3.2 开始，引入了一个更加方便的动画视图 **ViewPropertyAnimator**。它就像构建器一样，所有对不同属性修改的调用都可以连接起来组成一个动画。在当前线程的

Looper 的相同迭代中，对 `ViewPropertyAnimator` 的所有调用都会汇集到一个动作中。程序清单 2-45 演示了同样的视图过渡动画示例，但使用新的 API 进行实现。

程序清单 2-45 使用 `ViewPropertyAnimator` 的 Activity

```
public class AnimateActivity extends Activity implements View.OnClickListener {

    View viewToAnimate;

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

        Button button = (Button)findViewById(R.id.toggleButton);
        button.setOnClickListener(this);

        viewToAnimate = findViewById(R.id.theView);
    }

    @Override
    public void onClick(View v) {
        if(viewToAnimate.getAlpha() > 0f) {
            //如果视图已经可见，则将其从右侧滑出
            viewToAnimate.animate().alpha(0f).translationX(1000f);
        } else {
            //如果视图是隐藏的，原地做一个渐显动画
            //Property Animation 会实际修改视图，因此必须首先恢复视图的位置
            viewToAnimate.setTranslationX(0f);
            viewToAnimate.animate().alpha(1f);
        }
    }
}
```

在这个示例中，滑动动画和渐显动画是通过修改 `alpha` 和 `translationX`(这个值需要足够大才能够让视图移出屏幕)属性一起实现的。我们不需要将这些方法调用链接到一起，从而组成一个动画。即使我们在不同的地方调用，它们也还是会一起执行，因为它们都是在主线程的 `Looper` 的相同迭代中设置的。

注意，这里我们首先恢复了视图的位置属性，然后运行了没有滑动效果的渐显动画。这是因为属性动画会修改视图本身，而不是只是暂时地绘制(之前的动画 API 的机制)。如果不恢复这个属性，依然会有渐显动画，但会在屏幕外右侧 1 000 像素处进行。

5. ObjectAnimator

(API Level 11)

虽然 `ViewPropertyAnimator` 可以很方便且快速地实现简单的属性动画，但对于一些更加复杂的工作，例如将多个动画链接到一起，这种方式会受到一定的限制。这时我们可以使用它的父类 `ObjectAnimator`。通过 `ObjectAnimator` 我们可以设置监听器，从而在动画开

始和结束时得到相应的通知；另外在动作做增量更新时也可以得到通知。程序清单 2-46 显示了如何使用 ObjectAnimator 来修改我们的 Flipper 动画代码。

程序清单 2-46 使用 ObjectAnimator 实现掷硬币动画

```
public class Flipper extends Activity {

    boolean isHeads;
    ObjectAnimator flipper;
    Bitmap headsImage, tailsImage;
    ImageView flipImage;

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

        headsImage = BitmapFactory.decodeResource(getResources(),
            R.drawable.heads);
        tailsImage = BitmapFactory.decodeResource(getResources(),
            R.drawable.tails);

        flipImage = (ImageView)findViewById(R.id.flip_image);
        flipImage.setImageBitmap(headsImage);
        isHeads = true;

        flipper = ObjectAnimator.ofFloat(flipImage, "rotationY", 0f, 360f);
        flipper.setDuration(500);
        flipper.addUpdateListener(new AnimatorUpdateListener() {
            @Override
            public void onAnimationUpdate(ValueAnimator animation) {
                if (animation.getAnimatedFraction() >= 0.25f && isHeads) {
                    flipImage.setImageBitmap(tailsImage);
                    isHeads = false;
                }
                if (animation.getAnimatedFraction() >= 0.75f && !isHeads) {
                    flipImage.setImageBitmap(headsImage);
                    isHeads = true;
                }
            }
        });
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if(event.getAction() == MotionEvent.ACTION_DOWN) {
            flipper.start();
            return true;
        }
        return super.onTouchEvent(event);
    }
}
```

```
    }
}
```

属性动画提供了一些之前旧动画系统没有的变换功能，例如 x 轴和 y 轴的旋转效果，从而实现三维变换的效果。本例中，我们不需要计算缩放比例来实现旋转，只需要告诉视图沿着 y 轴旋转即可。正因为如此，我们不再需要使用两个动画来操作硬币，整个旋转过程只需要操作 `rotationY` 属性即可。

另一个强大之处就是 `AnimationUpdateListener`，它提供了动画运行过程中的常规回调方法。`getAnimatedFraction()`方法会返回当前动画完成的百分比。还可以通过 `getAnimatedValue()`得到当前动画中某个属性的准确值。

本例中，我们使用第一个方法在动画运行到两个时刻(硬币可以换面，即 90° 和 270° 或者动画时长的 25%和 75%)时，更换正反面的图片。因为并不能保证从每个角度我们都可以得到通知，所以当达到阈值后，我们会立即更换图片。我们还设置了一个布尔标识来避免在旋转过程中对同一个值进行重复的图片设置(会产生不必要的性能损耗)。

如果需要链接多个动画的应用程序需要的话，`ObjectAnimator` 还可以支持更加传统的 `AnimationListener` 来响应动画的主要事件，例如开始、结束和重复。

2.16 布局变化时的动画

2.16.1 问题

应用程序动态地添加或移除布局中的视图，你希望这种变化能够以动画的形式展示出来。

2.16.2 解决方案

(API Level 11)

使用 `LayoutTransition` 对象自定义在给定布局中对视图结构修改后的动画效果。在 Android 3.0 及以后版本中，只需要简单地在 XML 中设置 `android:animateLayoutChanges` 标签或者在 Java 代码中添加一个 `LayoutTransition` 对象即可实现任何 `ViewGroup` 改变布局时的动画效果。

布局中的每个 `View` 对象在布局变换时有 5 种状态。应用程序可以为下面任何一种状态设置自定义动画：

- **APPEARING**：容器中出现一个视图
- **DISAPPEARING**：容器中消失一个视图
- **CHANGING**：布局改变导致某个视图随之改变，例如调整大小，但不包括添加或移除视图。
- **CHANGE_APPEARING**：其他视图的出现导致某个视图改变。
- **CHANGE_DISAPPEARING**：其他视图的消失导致某个视图改变。

2.16.3 实现机制

程序清单 2-47 和 2-48 演示了一个基本的 LinearLayout 改变时的动画。

程序清单 2-47 res/layout/main.xml

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

    <Button
        android:id="@+id/button_add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onAddClick"
        android:text="Click To Add Item" />

    <LinearLayout
        android:id="@+id/verticalContainer"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:animateLayoutChanges="true"
        android:orientation="vertical" />

</LinearLayout>
```

程序清单 2-48 添加和移除视图的 Activity

```
public class MainActivity extends Activity {

    LinearLayout mContainer;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mContainer = (LinearLayout) findViewById(R.id.verticalContainer);
    }

    //添加可以自己移除自己的按钮
    public void onAddClick(View v) {
        Button button = new Button(this);
        button.setText("Click To Remove");
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mContainer.removeView(v);
            }
        });
    }
}
```

```

    }
    });
    mContainer.addView(button, new LinearLayout.LayoutParams(
        LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
}
}

```

这个简单的示例在单击 Add Item 按钮时会在 LinearLayout 中添加 Button。每个新 Button 在单击时都具有将自己从布局中移除的功能。想要此过程动起来，我们只需要在 LinearLayout 上设置 `android:animateLayoutChanges="true"`，之后框架会进行接下来的工作。默认情况下，新 Button 会渐入到新的位置，而不会干扰其他视图；移除时 Button 会出现渐出动画，而周围的视图则会平滑地填充移除时产生的空隙。

我们可以为每个过程自定义过渡动画来实现自定义的动画效果。参见程序清单 2-49，会向之前的 Activity 中添加一些自定义过渡动画。

程序清单 2-49 使用自定义 LayoutTransition 的 Activity

```

public class MainActivity extends Activity {

    LinearLayout mContainer;

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

        //布局改变时的动画
        mContainer = (LinearLayout) findViewById(R.id.verticalContainer);
        LayoutTransition transition = new LayoutTransition();
        mContainer.setLayoutTransition(transition);

        //通过翻转进入的动画代替默认的出现动画
        Animator appearAnim = ObjectAnimator.ofFloat(null, "rotationY", 90f, 0f)
            .setDuration(transition.getDuration(LayoutTransition.APPEARING));
        transition.setAnimator(LayoutTransition.APPEARING, appearAnim);

        //通过翻转消失的动画代替默认消失动画
        Animator disappearAnim = ObjectAnimator.ofFloat(null, "rotationX", 0f, 90f)
            .setDuration(transition.getDuration(LayoutTransition.DISAPPEARING));
        transition.setAnimator(LayoutTransition.DISAPPEARING, disappearAnim);

        //通过滑动动画代替默认的布局改变时的动画
        //我们需要立即设置一些动画属性，所以创建了多个 PropertyValueHolder
        //这个动画会让视图滑动进入并短暂地缩小一半长度
        PropertyValuesHolder pvhSlide = PropertyValuesHolder.ofFloat("y", 0, 1);
        PropertyValuesHolder pvhScaleY =
            PropertyValuesHolder.ofFloat("scaleY", 1f, 0.5f, 1f);
        PropertyValuesHolder pvhScaleX =
            PropertyValuesHolder.ofFloat("scaleX", 1f, 0.5f, 1f);
    }
}

```

```

        Animator changingAppearingAnim = ObjectAnimator.ofPropertyValuesHolder(
            this, pvhSlide, pvhScaleY, pvhScaleX);
        changingAppearingAnim.setDuration(
            transition.getDuration(LayoutTransition.CHANGE_DISAPPEARING));
        transition.setAnimator(LayoutTransition.CHANGE_DISAPPEARING,
            changingAppearingAnim);
    }

    public void onAddClick(View v) {
        Button button = new Button(this);
        button.setText("Click To Remove");
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mContainer.removeView(v);
            }
        });

        mContainer.addView(button, new LinearLayout.LayoutParams(
            LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
    }
}

```

本例中，我们在 Button 布局中修改了 APPEARING、DISAPPEARING 和 CHANGE_DISAPPEARING 时的过渡动画。前两个过渡动画会影响到视图添加或移除时的效果。当单击 Add Item 按钮后，新增的按钮会水平旋转进入现有视图中。当单击 remove 按钮时，该按钮会在视图中垂直旋转地消失。这两个动画都是通过创建新的 ObjectAnimator 对象并设置自定义属性来实现的，而动画持续的时间则是每种过渡类型的默认时间(通过一个特定过渡类型的键值设置到我们的 LayoutTransition 实例上)。最后一种过渡动画稍微有点复杂，需要创建一个动画，让周围的视图可以平滑地运动到新位置上，滑动的同时会产生缩放效果。

注意：

在自定义视图改变时的过渡动画时，添加移动到视图新位置的动画非常重要，否则在创建视图或填充视图消失区域时可能会出现闪烁的现象。

为了实现这种效果，需要通过 PropertyValuesHolder 实例创建一个 ObjectAnimator 来设置一些属性。动画的每个属性都是单独的 PropertyValuesHolder，并且通过 ofPropertyValuesHolder() 工厂方法添加到 animator 对象中。最后这个过渡动画使移除的按钮下面的所有按钮向上滑动到刚刚空出的位置，同时稍微收缩一下。

2.17 用 Drawable 做背景

2.17.1 问题

应用程序需要创建自定义背景，要求背景是带渐变的圆角矩形，而你并不想为此耗费

大量的时间来处理各种图片文件。

2.17.2 解决方案

(API Level 1)

用 Android 最强力的 XML 资源系统实现：创建形状 Drawable。如果能做到这点，用 XML 资源创建这些视图就很有意义了，因为用 XML 创建的 Drawable 本身就是可缩放的，如果将其设为背景，它会自动填充视图的边界。

在 XML 中用<shape>标签定义 Drawable 时，实际上是创建了一个 GradientDrawable 对象。对象的形状可以是矩形、椭圆、线条或圆圈，最常见的背景形状是矩形。具体来说，在使用矩形时，可以用下面这些参数定义形状：

- 角半径
 - 定义 4 个角的半径，或是分别定义各个角的半径。
- 渐变
 - 线性、放射或 sweep 渐变。
 - 两个或三个颜色值。
 - 方向可以是 45° 的任何倍数(0 就是从左到右，90 就是从下到上，以此类推)。
- 固定颜色
 - 用一种颜色填充形状。
 - 如果同时定义了渐变的话，效果会受影响。
- 边线
 - 对象形状的边界。
 - 定义宽度和颜色。
- 大小和 padding

2.17.3 实现机制

创建视图的静态背景是件非常麻烦的事情，需要准备各种尺寸的图片才能确保背景能在所有的设备上正常显示。如果视图的大小还需要根据其中的内容动态变化，那事情会变得更复杂。

为了避免这个问题，在 res/drawable 中创建一个描述形状 of XML 文件，然后通过 android:background 属性将其应用为视图的背景。

1. 渐变的 ListView 行

第一个示例是创建一个渐变的矩形，可以将其设为 ListView 中各行的背景。程序清单 2-50 中是定义该形状的 XML。

程序清单 2-50 res/drawable/backgradient.xml

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

```

<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#E0E0E0"
        android:endColor="#808080"
        android:type="linear"
        android:angle="270"
    />
</shape>

```

这里我们选择了使用两种灰色之间的线性渐变，从上到下变化。如果要在渐变中加入第 3 种颜色，需要在<gradient>标签中添加 android:middleColor 属性。

现在，可以在构建自定义 ListView 的视图或布局(在 2.23 节中将详细介绍如何创建这些视图)中引用这个 Drawable 资源。将这个 Drawable 设为背景，可以在视图的 XML 文件中加入 android:background="@drawable/backgradient"，也可以在 Java 代码中调用 View.setBackgroundResource (R.drawable.backgradient)。

高级提示：

XML 中颜色的种类限制是 3 种，但是 GradientDrawable 的构造函数中的颜色参数是一个整型数组 int[]，传递多少颜色都可以。

将该 Drawable 设为 ListView 中各行的背景，结果如图 2-7 所示。

1. 圆角视图组

另一种流行的技术是用 XML Drawable 创建布局的背景，将若干个部件组织在一起。常用的风格是圆角加上细边框。在 XML 中定义该形状的代码如程序清单 2-51 所示。

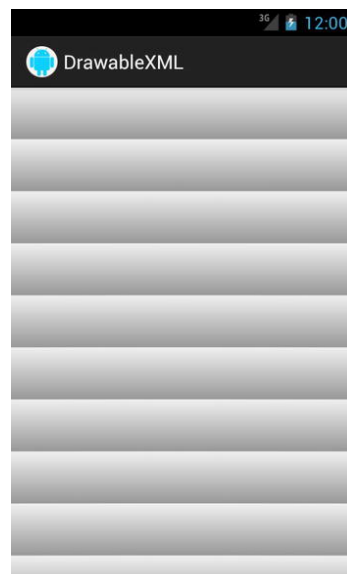


图 2-7 行背景是渐变的 Drawable

程序清单 2-51 res/drawable/roundback.xml

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid
        android:color="#FFF"
    />
    <corners
        android:radius="10dip"
    />
    <stroke
        android:width="5dip"
        android:color="#555"
    />
</shape>

```

```

/>
</shape>

```

这个示例中，填充色是白色，边框是灰色。如前一个示例所示，在任何视图和布局中都可以在 XML 中加入 `android:background="@drawable/roundback"` 或是在 Java 代码中调用 `View.setBackgroundResource(R.drawable.roundback)` 以引用这个 Drawable。

将该背景应用到视图，结果如图 2-8 所示。

2. Drawable 模式

我们要看的关于 Drawable 的下一个分类就是模式。通过 XML，我们可以定义一些规则来定义小图片的重复平铺模式。这对于处理全屏背景是非常有用的，因为这种方式不需要向内存中加载大的位图。

应用程序可以通过设置 `<bitmap>` 元素的 `tileMode` 属性来创建一种模式，属性值如下：

- `clamp`：复制源位图的边缘像素
- `repeat`：源位图会在横向和纵向重复地平铺
- `mirror`：源位图会重复地进行平铺，但按照源位图和镜像位图交替的方式平铺

图 2-9 显示了两张小的正方形图片，它们将成为 Drawable 模式的源图片。



图 2-9 Drawable 模式的源位图

程序清单 2-52 和 2-53 显示了如何将一个 XML 模式定义为背景。

程序清单 2-52 `res/drawable/pattern_checker.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/checkers"
    android:tileMode="repeat" />

```

程序清单 2-53 `res/drawable/pattern_strips.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<bitmap xmlns:android="http://schemas.android.com/apk/res/android"
    android:src="@drawable/strips"
    android:tileMode="mirror" />

```



图 2-8 带边框的圆角矩形视图背景

提示:

Drawable 模式只可以使用有固定边界的位图, 例如外部图片。而 XML 的 shape 不能作为 Drawable 模式的源位图。

图 2-10 显示了视图在应用每种模式后的背景效果。

你会看到棋盘图片会按原样重复, 而条纹图片会在水平和垂直方向进行镜像, 之后再平铺到整个屏幕上。

3. 9-patch 图片

NinePatchDrawable 是 Android 最大的优势之一, 使其可在所有设备上都灵活调整 UI。9-patch 是一种特殊的图片, 通过指定可缩放和不可缩放的区域实现某个特定区域的拉伸。事实上, 这种图片类型得名于 9 个拉伸区域, 这些区域会创建图像映射(在某一时刻)。

让我们看一个示例以更好地了解它的实现机制。图 2-11 显示了两张图片, 左边的是原始图片, 右边的是转换后生成的 9-patch 图片。

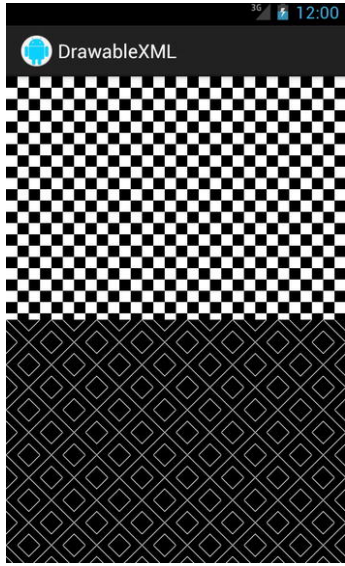


图 2-10 背景模式



图 2-11 语音气泡源图片 speech_background.png(左边)和 9-patch 文件 speech_background.9.png(右边)

注意图片上各条边上的黑色标记。一个有效的 9-patch 图片文件就是一个简单的 PNG 图片, 该图片最外边的 1 像素只会是黑色像素或透明像素。每条边上的黑色像素定义了图片该如何拉伸并保护图像的一部分内容不被拉伸。

- 左侧: 这里的黑色像素定义了图片垂直方向拉伸的区域。这个区域的内容会平铺以实现拉伸。图 2-10 的示例图片定义了该拉伸区域。
- 顶部: 这里的黑色像素定义了图片水平方向拉伸的区域。这个区域的内容会平铺以实现拉伸。图 2-10 的示例图片定义了两个这样的拉伸区域。
- 右侧: 这里的黑色像素定义了图片垂直方向的内容区域, 也就是视图内容的显示区域。事实上, 它决定了顶部和底部的 padding 值, 但保留了背景图像区域。
- 底部: 这里的黑色像素定义了图片水平方向的内容区域, 也就是视图内容的显示区域。事实上, 它决定了左侧和右侧的 padding 值, 但保留了背景图像区域。这里必须包含一行, 组成该行的实体像素定义了该区域。

这个图片是使用 Android SDK 提供的 draw9patch 工具创建的。为了更好地看清楚这些

标记如何影响图片的显示效果，让我们看一下该图片加载到工具中的效果。参见图 2-12。

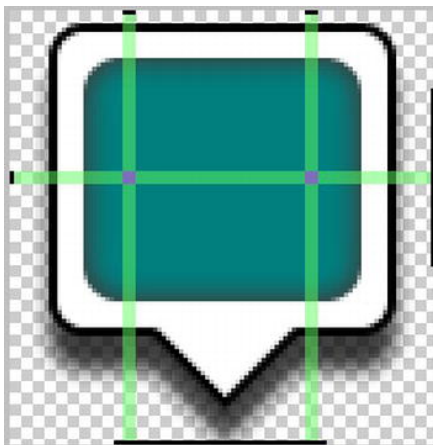


图 2-12 draw9patch 工具中的语音气泡图片

现在你已经知道 9-patch 名称的由来。图片的非高亮部分是不可以被拉伸的。每个图片的高亮区域将会在某个方向(水平方向或垂直方向，根据它们的方向)进行拉伸，交叉高亮的区域则可以在每个方向都进行拉伸。在各个方向至少拥有一个拉伸区域的图片，将会在图片中创建 9 个独立的映射区域：4 个角是不能修改的、4 个中间区域拉伸一次、一个中间区域拉伸两次。

创建 NinePatchDrawable 和使用它作为背景并不需要任何特殊的代码；图片文件只需要以特殊的 .9.png 命名，这样 Android 就可以正确地处理它。程序清单 2-54 显示了如何将这个图片设置为背景，而图 2-13 显示了将这个图片设置为 TextView 的背景后的效果。

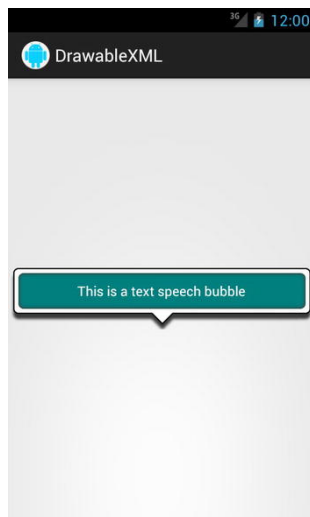


图 2-13 语音气泡作为 TextView 的背景

程序清单 2-54 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:gravity="center"
        android:text="This is a text speech bubble"
        android:background="@drawable/speech_background" />
    </RelativeLayout>
```

注意，两三个像素宽的横向拉伸区是如何围绕语音气泡的原点中心均匀分配它们之间多余的空间的。要在两个拉伸点之间创建偏移区域，可以通过改变它们到图像中心的距离或改变它们的大小来实现。如果有个区域是三个像素宽，而另一个只有一个像素宽，那么前者在拉伸时会占据三倍于后者的空间。

2.18 创建自定义状态的 Drawable

2.18.1 问题

要自定义诸如 Button 或 CheckBox 之类有多个状态(默认、按下、选中等)的元素。

2.18.2 解决方案

(API Level 1)

创建一个状态列表(state-list)的 Drawable，将其应用到要自定义的元素。无论是用 XML 定义 Drawable 还是使用图片，都可以通过 Android 提供的另一个 XML 元素<selector>创建一个引用，指向多个图片以及多个图片可见的条件。

2.18.3 实现机制

先看看下面这个状态列表 Drawable 的示例，然后再深入讨论它的各个部分：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="false" android:drawable="@drawable/disabled" />
    <item android:state_pressed="true" android:drawable="@drawable/selected" />
    <item android:state_focused="true" android:drawable="@drawable/selected" />
    <item android:drawable="@drawable/default" />
</selector>
```

注意：

<selector>是有序的。Android 在遍历列表时，会返回第一个状态完全匹配的 Drawable。在将状态属性应用到各条目时要注意这一点。

列表的每个 item 中所引用的 Drawable 都对应一种状态，如果某个 item 需要匹配多个状态的话，可以加入多个状态参数。Android 会遍历这个列表，选择其中第一个能匹配当前视图各项指标的状态，将其对应的可绘制资源应用到视图。从这个角度考虑，最好是将正常状态，即没有任何附加指标的默认状态放在列表的最底部。

下面列出了最常用的状态属性，它们都有布尔值：

- state_enabled
 - isEnabled()返回的视图的值。
- state_pressed

- 用户在触摸屏上单击了该视图。
- state_focused
 - 视图获得焦点。
- state_selected
 - 用户通过按键或 D-pad 选中了视图。
- state_checked
 - isChecked()返回的可选中视图的值。

现在来查看如何将把这些状态列表 Drawable 应用到各种视图。

1. Button 和其他可单击控件

当视图在上述状态间切换时，诸如 Button 之类的控件的背景 Drawable 会发生变化。所以，XML 中的 android:background 属性或 view.setBackgroundDrawable()方法都可用于添加状态列表。程序清单 2-55 中的示例是 res/drawable/中定义的 button_states.xml 文件。

程序清单 2-55 res/drawable/button_states.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="false" android:drawable="@drawable/disabled" />
    <item android:state_pressed="true" android:drawable="@drawable/selected" />
    <item android:drawable="@drawable/default" />
</selector>
```

这里的 3 个 @drawable 资源是项目中的图片，选择器就在这些图片间切换，如前所述，如果所有的项都不能与当前视图的状态匹配，那么最后一项就会作为默认值返回，所以最后一项不需要标明其所匹配的状态。下面将这个状态列表添加到 XML 中定义的一个视图：

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="My Button"
    android:background="@drawable/button_states" />
```

2. CheckBox 和其他可选择控件

很多控件都实现了可单击的界面，例如 CheckBox 和其他的 CompoundButton 的子类，它们之间状态切换的机制都大同小异。在这些控件中，控件背景跟状态没有关联，而是通过另一个名为 button 的属性实现用自定义的 Drawable 来表示“选中”状态。在 XML 中就是 android:button 属性，在代码中则是用 CompoundButton.setButtonDrawable()方法来实现的。

程序清单 2-56 中的示例是 res/drawable/中的 check_stater.xml 文件。再提醒一下，其中罗列的 @drawable 资源就是要切换的图片。

程序清单 2-56 res/drawable/check_states.xml

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

```

<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_enabled="false" android:drawable="@drawable/disabled" />
    <item android:state_checked="true" android:drawable="@drawable/checked" />
    <item android:drawable="@drawable/unchecked" />
</selector>

```

在 XML 中将其关联到 CheckBox 上:

```

<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:button="@drawable/check_states" />

```

2.19 将遮罩应用到图片

2.19.1 问题

要将裁剪遮罩应用到图片或形状，定义应用中另一张图片的可见边界。

2.19.2 解决方案

(API Level 1)

利用 2D 图形和 PorterDuffXferMode，可以将各种遮罩应用到某个 Bitmap 图像：本范例的基本步骤如下所示：

创建一个空白的 Bitmap，以及在其中绘图的 Canvas。

- (1) 首先在 Canvas 上画好遮罩模式。
- (2) 将 PorterDuffXferMode 应用到 Paint。
- (3) 用 transfer 模式将源图绘制到 Canvas 上。

其中的关键就是 PorterDuffXferMode，它在绘图操作中既会考虑到源图的状态，也会考虑到目标对象的状态。目标对象就是 Canvas 中已有的数据，源图就是被应用到当前操作的图片数据。

有很多模式参数可以应用到 PorterDuffXferMode，可以得到各种不同的效果。在这里只需要使用 PorterDuff.Mode.SRC_IN 模式即可。这个模式只会在源图和目标图重叠的地方绘制图形，内容来自源图；换句话说，就是会根据目标图对源图进行裁剪。

2.19.3 实现机制

1. 圆角 Bitmap

这种技术一个常见的应用就是在 ImageView 中显示 Bitmap 图形之前，为其添加圆角。为了演示这个示例，我们将给图 2-14 中的原图加上遮罩。



图 2-14 原有的源图

首先在 Canvas 中根据所需的圆角半径创建一个圆角矩形，这就是我们的“遮罩”。然后以 PorterDuff.Mode.SRC_IN 为画笔在同一个 Canvas 上绘制源图，得到的就是带圆角的源图。

SRC_IN 转换模式就是告诉 paint 对象，只在源图和目标图(已经画好的圆角矩形)重叠的地方绘制像素点，像素点则来自源图。程序清单 2-57 是 Activity 中的代码。

程序清单 2-57 将圆角遮罩应用到 Bitmap 的 Activity

```
public class MaskActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ImageView iv = new ImageView(this);

        //创建并加载图片(通常是不可修改的)
        Bitmap source = BitmapFactory.decodeResource(getResources(),
            R.drawable.dog);

        //创建一个供修改的图片，然后加上 Canvas
        Bitmap result = Bitmap.createBitmap(source.getWidth(), source.getHeight(),
            Config.ARGB_8888);
        Canvas canvas = new Canvas(result);
        Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);

        //首先创建并绘制圆角矩形“遮罩”
        RectF rect = new RectF(0,0,source.getWidth(),source.getHeight());
        float radius = 25.0f;
        paint.setColor(Color.BLACK);
        canvas.drawRoundRect(rect, radius, radius, paint);
        //用转换模式转化并绘制原图
        paint.setXfermode(new PorterDuffXfermode(Mode.SRC_IN));
```

```

        canvas.drawBitmap(source, 0, 0, paint);
        paint.setXfermode(null);

        iv.setImageBitmap(result);
        setContentView(iv);
    }
}

```

得到的结果如图 2-15 所示。

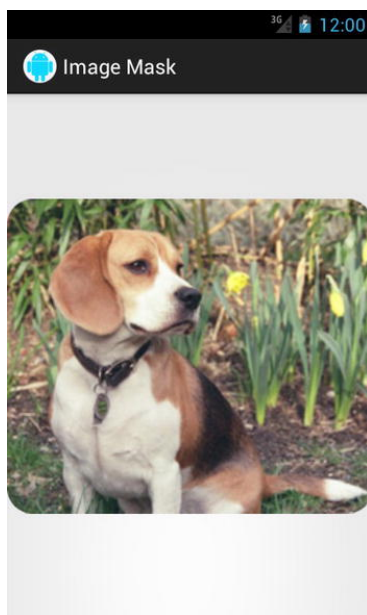


图 2-15 使用了圆角遮罩的图片

2. 任意遮罩图形

下面来看一个更有意思的示例。现在有两张图片：源图和用于表示遮罩的图(这里用的遮罩是一个倒三角，见图 2-16)。



图 2-16 原始的源图(左)和任意形状的遮罩图片(右)

所选择的遮罩图片并不非得是我们所选的那样，只要把要露出来的地方设为黑色，其他地方设为透明就可以。不过，最好是选择确保能在系统上正确显示的图片。程序清单 2-58 中的 Activity 将遮罩应用到该图片上并将其显示出来。

程序清单 2-58 将任意遮罩应用到 Bitmap 的 Activity

```
public class MaskActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ImageView iv = new ImageView(this);

        //创建并加载图片(通常是不可修改的)
        Bitmap source = BitmapFactory.decodeResource(getResources(),
            R.drawable.dog);
        Bitmap mask = BitmapFactory.decodeResource(getResources(),
            R.drawable.triangle);

        //创建一个供修改的图片，然后加上 Canvas
        Bitmap result = Bitmap.createBitmap(source.getWidth(), source.getHeight(),
            Config.ARGB_8888);
        Canvas canvas = new Canvas(result);
        Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);

        //先画上遮罩图片，然后用转换模式绘制源图
        canvas.drawBitmap(mask, 0, 0, paint);
        paint.setXfermode(new PorterDuffXfermode(Mode.SRC_IN));
        canvas.drawBitmap(source, 0, 0, paint);
        paint.setXfermode(null);

        iv.setImageBitmap(result);
        setContentView(iv);
    }
}
```

跟前面一样，先在 Canvas 上画好遮罩，然后用 PorterDuff.Mode.SRC_IN 模式将源图跟遮罩图片重叠的部分画出来，结果如图 2-17 所示。

3. 试试这个

用 PorterDuffXferMode 可以融合两张图片得到很多有意思的效果。把上面示例中 PorterDuffXfer.Mode 的参数改成其他值。每个参数混合两个 Bitmap 的方式都会略有不同。很有意思的，动手试试吧！

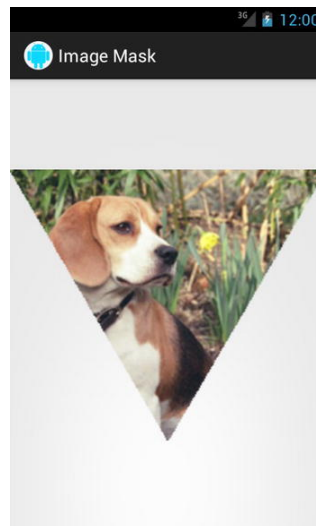


图 2-17 应用了遮罩的图片

2.20 创建持久的对话框

2.20.1 问题

创建一个有多个输入控件或显示多种信息的用户对话框，当设备旋转时对话框不能消失。

2.20.2 解决方案

(API Level 1)

绝对不要使用对话框控件！用 Dialog 主题创建一个 Activity。对话框控件是托管对象，如果在其可见时设备发生了旋转，必须要小心处理。否则就会导致窗口管理器中的引用丢失。通过用 Activity.showDialog()和 Activity.dismissDialog()来让 Activity 管理对话框能解决这个问题，但也仅仅是解决了这个问题。

Dialog 在设备旋转的过程中没有任何保存其自身状态的机制，这个工作自然就得交给显示这个 Dialog 的 Activity，显然，在 Dialog 消失之前，必须要把在其中输入的数据持久化或者是传递回去。

如果呈现给用户的界面需要在旋转过程中保持状态，始终处于最前端，更好的办法是使用 Activity。这样就可以通过各种生命周期回调方法来保存和读取状态。此外，Activity 在旋转过程中不需要处理消失和重现，这就避免了引用丢失的担忧。从用户的角度来看，使用 Theme.Dialog 系统主题可以将 Activity 弄得跟 Dialog 一样。

2.20.3 实现机制

程序清单 2-59 中是一个有标题和 TextView 的简单 Activity。

程序清单 2-59 使用 Dialog 主题的 Activity

```
public class DialogActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("Activity");
        TextView tv = new TextView(this);
        tv.setText("I'm Really An Activity!");
        //添加 padding, 对话框边框留白
        tv.setPadding(15, 15, 15, 15);
        setContentView(tv);
    }
}
```

我们可以在应用的 AndroidManifest.xml 文件中将 Dialog 主题应用到该 Activity(参见程序清单 2-60)。

程序清单 2-60 在 Manifest 文件中将 Dialog 主题应用到上述 Activity

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.dialogs"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".DialogActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Dialog">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

注意其中的 `android:theme="@android:style/Theme.Dialog"` 参数, 这个主题让 Activity 具有类似 Dialog 的外观, 但又具有 Activity 的各种优点。运行这个应用, 看到的效果如图 2-18 所示。

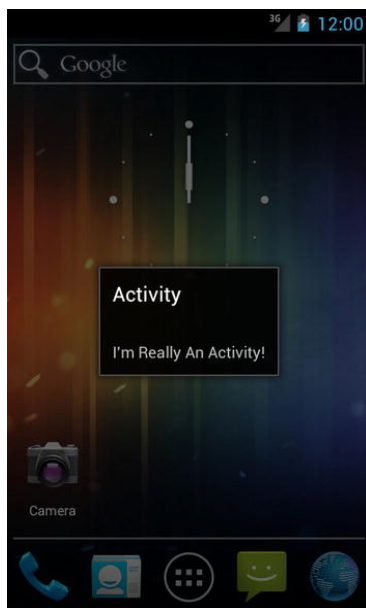


图 2-18 将 Dialog 主题应用到 Activity

尽管这是一个具有完整功能的 Activity, 但它在用户界面中和 Dialog 完全一样, 能局部覆盖其下面的 Activity(这里就是 Home 屏幕)。

2.21 实现针对具体场景的布局

2.21.1 问题

应用必须是通用的，能在各种尺寸和方向的屏幕上运行。你需要为各种不同的情况准备好相应的布局。

2.21.2 解决方案

(API Level 4)

构建多个布局文件，然后通过资源标识符让 Android 选择合适的布局。下面看看如何构建适用于不同大小和方向的屏幕的资源。还将学习在多个配置使用同一个布局时如何使用布局别名来减少重复代码。

2.21.3 实现机制

1. 针对不同方向

用下面的标识符可以为 Activity 的横屏和竖屏创建不同的资源：

- resource-land
- resource-port

这适用于所有类型的资源，但通常都是用于布局。因此，在项目中不只有一个 res/layout/ 文件夹，而是应该有 res/layout-port/ 和 res/layout-land/ 两个文件夹。

注意：

在实际开发中最好有一个不带标识符的资源文件夹。这样在 Android 设备不能匹配任何给定的设置时也能有可用的资源。

2. 针对不同尺寸

屏幕尺寸标识符(物理尺寸，不要与像素密度混淆)可以用于针对平板电脑等大屏幕设备的资源。大部分情况下，一个布局就能满足所有手机设备的屏幕尺寸，但如果在平板电脑上就可能需要一个专门的平板电脑布局来填充用户所面对的大屏幕。

在 Android 3.2(API Level 13)之前，下面的资源标识符用于标识屏幕的物理尺寸。

- resource-small
 - 屏幕尺寸最小为 426 dp×320 dp
- resource-medium
 - 屏幕尺寸最小为 470 dp×320 dp
- resource-large
 - 屏幕尺寸最小为 640 dp×480 dp
- resource-xlarge

- 屏幕尺寸最小为 960 dp×720 dp

随着手持设备和平板中大屏幕设备越来越普及，显然，只是 4 个广义的类型不能完全避免资源定义上的重叠。因此，在 Android 3.2 中，引入了一个新的基于屏幕真实尺寸(单位为 dp)的系统。通过这个新系统，可以对物理屏幕使用以下的资源标识符：

- 最小宽度：资源-sw_dp
 - 屏幕在最短方向上(即方向无关)至少具有指定的 dp(density-independent pixels)
 - 640dp×480dp 的屏幕通常最小宽度为 480dp
- 宽度：资源-w_dp
 - 屏幕在当前水平方向上(即方向无关)至少具有指定的 dp
 - 640dp×480dp 的屏幕横屏时宽度为 640dp 而竖屏时宽度为 480dp
- 高度：资源-h_dp
 - 屏幕在当前垂直方向上(即方向无关)至少具有指定的 dp
 - 640dp×480dp 的屏幕竖屏时高度为 640dp 而横屏时高度为 480dp

所以想要在整个应用程序中包含平板专用的布局，只需要为旧版本的平板添加 /res/layout-large/ 目录，而为新版本平板添加 res/layout-sw720dp/ 目录即可。

3. 布局别名

在创建通用的应用程序 UI 时，还有最后一个概念需要讨论，那就是布局的别名。通常是在同一个布局要用在多个不同的设备配置，但在同一个资源目录中使用多个资源限定符(如最小宽度限定符和传统的尺寸限定符)会产生问题的情况。这种情况常常会导致开发者要在不同的目录中创建同一个布局的多个副本，这维护起来非常困难。

我们可以通过别名解决这个问题。首先在默认资源目录中创建一个布局文件，然后就可以在每个使用该布局的配置并通过“资源-标识符”命名的目录下为这个文件创建多个别名。

下面的代码演示了 res/layout/main_tablet.xml 文件的别名。

```
<resources>
  <item name="main" type="layout">@layout/main_tablet</item>
</resources>
```

name 属性表示别名的名称，表示这个别名代表的资源会用在特定的配置上。当在代码中使用 R.layout.main 时，这个别名就会链接到 main_tablet.xml。这段代码可以放到 res/values-xlarge/layout.xml 和 res/values-sw720dp/layout.xml 中，这样这两个配置都会链接到同一个布局上。

4. 示例

接下来看一个使用这些技术的简单示例。定义一个 Activity，在其中用代码加载一个布局资源，不过，在资源中这个布局被定义了三次，分别是针对竖屏、横屏和平板电脑。先看看程序清单 2-61 中的 Activity。

程序清单 2-61 加载一个布局的简单 Activity

```
public class UniversalActivity extends Activity {

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

现在我们将为该 Activity 的不同配置定义三个单独的布局。程序清单 2-62 到 2-64 显示默认的布局、横屏时的布局和平板的 UI 布局。

程序清单 2-62 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!--默认布局-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is the default layout" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button One" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button Two" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button Three" />
</LinearLayout>
```

程序清单 2-63 res/layout-land/main.xml

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

```

        android:text="This is a horizontal layout for LANDSCAPE" />
<!--三个按钮会平均大小的填满整个屏幕 -->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button One" />
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button Two" />
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button Three" />
</LinearLayout>
</LinearLayout>

```

程序清单 2-64 res/layout/main_tablet.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!--平板的布局 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
<!--所有的用户按钮占用 25%的屏幕宽度-->
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:orientation="vertical">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="This is the layout for TABLETS" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button One" />
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button Two" />
        <Button

```

```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button Three" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button Four" />
</LinearLayout>

<!--显示详细内容的视图 -->
<TextView
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="3"
    android:text="Detail View"
    android:background="#CCC" />
</LinearLayout>

```

一种方式是创建具有相同文件名的三个文件并把它们放到各自的配置目录中，例如用于横屏的 `res/layout-land` 目录和用于平板的 `res/layout-large` 目录。如果每个文件只使用一次，这种方案是非常好的，但我们想要在多个配置下复用每个布局，因此在这个示例中采用了为这三个布局创建别名的方式。程序清单 2-65 到 2-68 显示了如何将每个布局链接到正确的配置上。

程序清单 2-65 `res/values-large-land/layout.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <item name="main" type="layout">@layout/main_tablet</item>
</resources>

```

程序清单 2-66 `res/value-sw600dp-land/layout.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <item name="main" type="layout">@layout/main_tablet</item>
</resources>

```

程序清单 2-67 `res/values-xlarge/layout.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <item name="main" type="layout">@layout/main_tablet</item>
</resources>

```

程序清单 2-68 `res/values-sw720dp/layout.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<resources xmlns:android="http://schemas.android.com/apk/res/android">
    <item name="main" type="layout">@layout/main_tablet</item>
</resources>

```

我们定义了一组配置来适应三种类型的设备：手机、7 英寸平板设备、10 英寸平板设备。手机设备会在竖屏模式时加载默认的布局，屏幕旋转时加载横屏布局。因为只有一种配置使用这些文件，所以文件会被直接分别放置到 `res/layout` 和 `res/layout-land` 目录下。

7 英寸的平板设备在之前的尺寸方案中通常会被定义为大屏幕设备，而在新的方案中它们是最小宽度为 600dp 的设备。竖屏时，我们决定让应用程序都使用默认的布局，而在横屏时会得到更大的空间，因此我们加载了平板的布局。想要实现该功能，我们为横屏模式创建了多个配置目录来匹配设备的类型。同时使用了最小宽度标识符和广义的尺寸标识符，这样就可以兼容旧平板和新平板。

10 英寸的平板设备在之前的尺寸方案中通常会被定义为大屏幕设备，而在新的方案中它们是最小宽度为 720dp 的设备。对于这些设备，屏幕大到可以在两个方向都使用平板布局，因此只通过屏幕尺寸定义了配置目录。

和小平板一样，同时使用最小宽度和广义尺寸标识符可以保证可以兼容所有的平板版本。

在所有引用平板布局的场景中，我们只需要创建布局文件即可管理，这要多亏了使用了别名。现在，运行应用程序后，会看到 Android 是如何选择合适的布局来匹配我们的配置的。图 2-19 显示了手机上默认的布局和横屏时的布局。

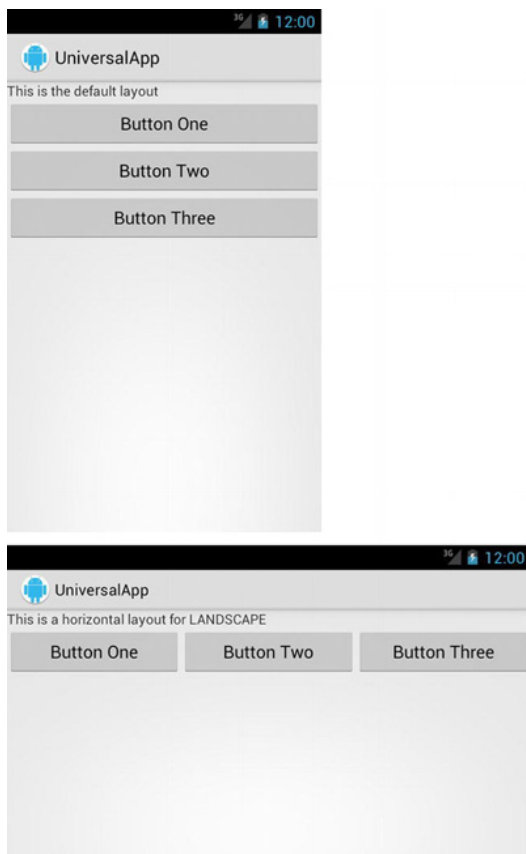


图 2-19 手机上竖屏和横屏的布局

同样的应用程序运行在 7 英寸的平板设备上会在竖屏时显示默认的布局，但在横屏上会显示全屏的平板布局(参见图 2-20)。

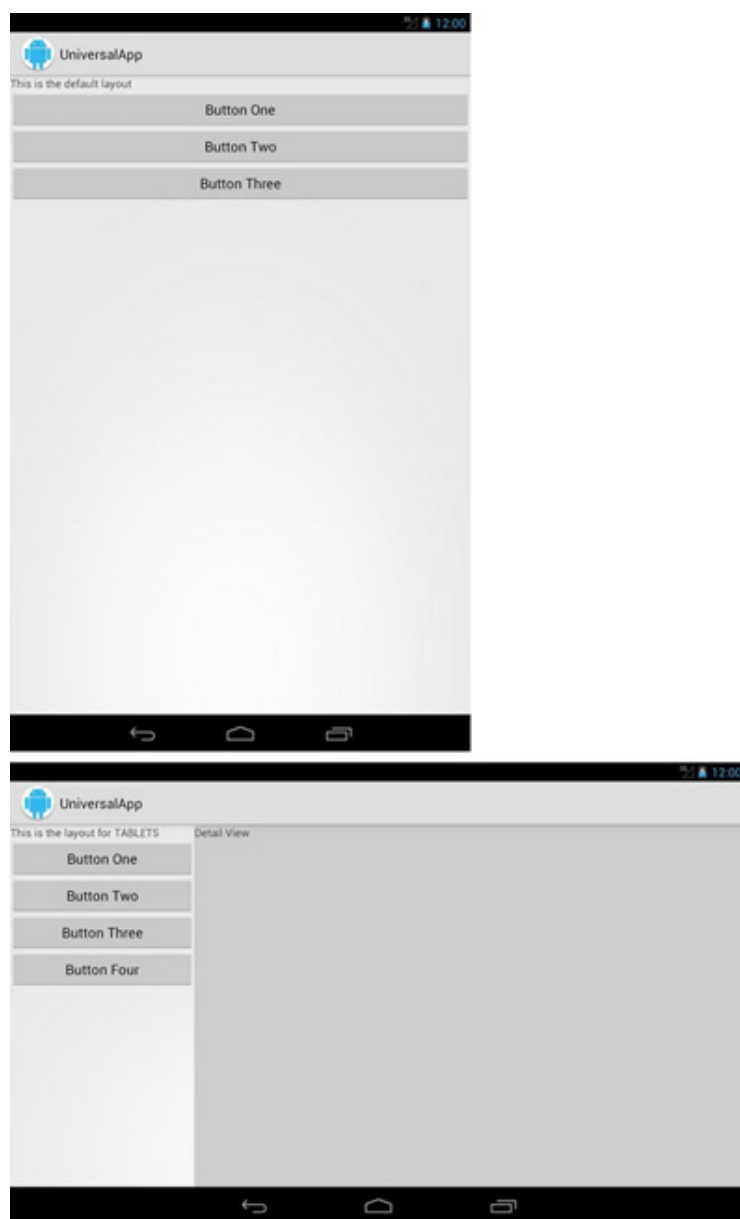


图 2-20 7 英寸平板：默认竖屏布局和平板横屏布局

最后，在图 2-21 中我们看到，在更大的 10 英寸平板上横屏和竖屏都会显示全屏的平板布局。

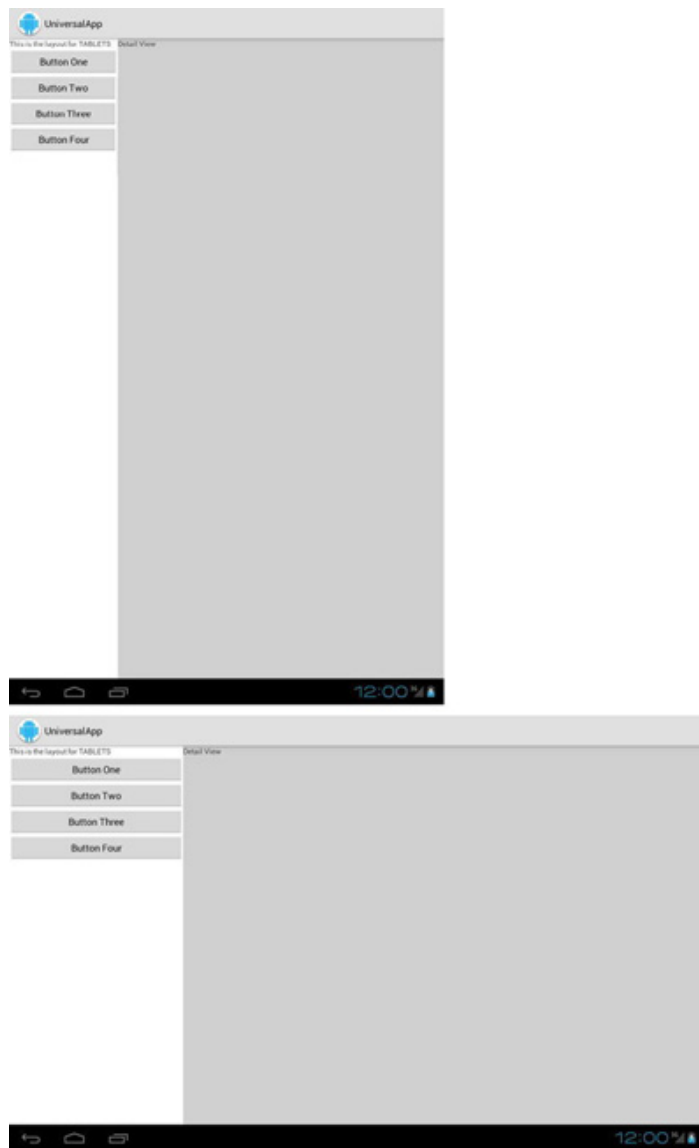


图 2-21 10 英寸平板：两个方向时都显示全屏平板布局

通过 Android 资源选择系统的强大能力，大大减少了为不同设备类型提供不同的优化 UI 布局的难度。

2.22 自定义键盘动作

2.22.1 问题

要自定义软键盘 Enter 键的外观，或者是改变用户按这个键所触发的动作，或者是两者都要实现。

2.22.2 解决方案

(API Level 3)

自定义键盘输入数据控件的输入方法(Input Method, IME)。

2.22.3 实现机制

1. 自定义 Enter 键

软键盘出现在屏幕上时, Enter 键上的文字通常显示的是根据当前聚焦的控件在视图中的顺序所执行的动作。在没有特别指定时, 如果视图中还有其他可聚焦的控件, 这个按键会显示 next; 如果当前聚焦的对象已经是最后一个可聚焦对象, 则会显示 done。通过视图的 XML 文件中的 android:imeOptions 可以自定义这个值。可用于自定义 Enter 键的值如下所示:

- actionUnspecified: 默认值, 根据设备的情况显示动作。
 - 动作事件是 IME_NULL。
- actionGo: 在 Enter 键上显示 Go。
 - 动作事件是 IME_ACTION_GO。
- ActionSearch: 在 Enter 键上显示 Search。
 - 动作事件是 IME_ACTION_SEARCH。
- actionSend: 在 Enter 键上显示 Send。
 - 动作事件是 IME_ACTION_SEND。
- actionNext: 在 Enter 键上显示 Next。
 - 动作事件是 IME_ACTION_NEXT。
- actionDone: 在 Enter 键上显示 Done。
 - 动作事件是 IME_ACTION_DONE。

下面看一个有两个可编辑文本框的布局, 如程序清单 2-69 所示。第一个文本框在 Enter 键上显示搜索图标, 第二个则显示 Go。

程序清单 2-69 带自定义输入选择的 EditText 控件的布局

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <EditText
        android:id="@+id/text1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:imeOptions="actionSearch"
    />
    <EditText
```

```

        android:id="@+id/text2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:imeOptions="actionGo"
    />
</LinearLayout>

```

最终显示的键盘可能会因为生产商自定义的用户界面而有些许差异，Google 原生用户界面的结果如图 2-22 所示。

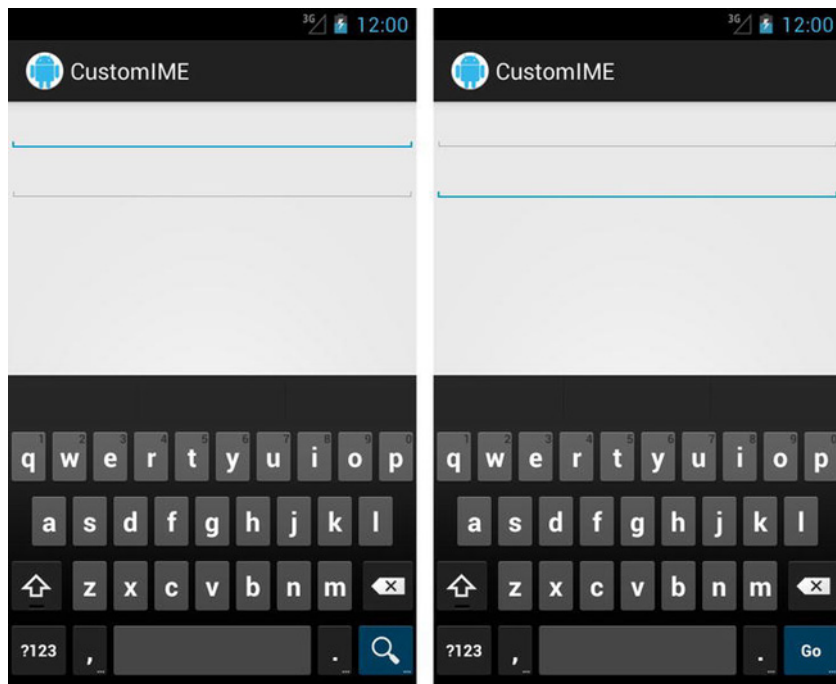


图 2-22 Enter 键上的自定义输入选项

注意：

自定义编辑器选项只会影响软键盘输入。改变这些选项的值不会影响到用户在物理键盘上输入时触发的事件。

2. 自定义动作

与自定义按键上显示的文字一样重要的是，自定义用户按下按键时所触发的动作。重载动作的默认行为需要给相应的视图加上 `TextView.OnEditorActionListener`。下面继续修改上面的布局示例，这次给两个视图都加上一个自定义的动作(参见程序清单 2-70)。

程序清单 2-70 实现了自定义按键动作的 Activity

```

public class MyActivity extends Activity implements OnEditorActionListener {

```

```

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

    //给视图添加监听器
    EditText text1 = (EditText)findViewById(R.id.text1);
    text1.setOnEditorActionListener(this);
    EditText text2 = (EditText)findViewById(R.id.text2);
    text2.setOnEditorActionListener(this);
}

@Override
public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
    if(actionId == IME_ACTION_SEARCH) {
        //处理搜索按键单击
        return true;
    }
    if(actionId == IME_ACTION_GO) {
        //处理 Go 按键单击
        return true;
    }
    return false;
}
}

```

onEditorAction()返回的布尔值会告诉系统应用是否处理了这个事件，或者是否应该将其传递给下一个可能的响应者。所以在实现该方法时一定要返回 true，这样系统就不会再对其他进行处理。当然，如果没有处理这个事件，就可以返回 false，这样系统的其他部分就能对其进行处理

2.23 隐藏软键盘

2.23.1 问题

需要通过用户界面上的某个事件隐藏软键盘。

2.23.2 解决方案

(API Level 3)

用 InputMethodManager.hideSoftInputFromWindow()方法可以让输入法管理器隐藏可见的输入法。

2.23.3 实现机制

下面这个示例演示了如何在 View.OnClickListener 中调用该方法：

```
public void onClick(View view) {
    InputMethodManager imm = (InputMethodManager) getSystemService(
        Context.INPUT_METHOD_SERVICE);
    imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
}
```

hideSoftInputFromWindow()方法的参数是一个 IBinder 窗口令牌。可以用 View.getWindowToken()从附加到窗口的 View 对象上获得。大部分情况下，每个事件的回调方法都会有一个引用指向正在编辑的 TextView。或是发起事件的 View(比如某个按键)。通过调用这些对象获得窗口令牌，再将其传递给 InputMethodManager 最方便的做法。

2.24 自定义 AdapterView 的空视图

2.24.1 问题

要在 AdapterView(ListView、GridView 等诸如此类的视图)没有数据时显示自定义的视图。

2.24.2 解决方案

(API Level 1)

把要显示的视图跟 AdapterView 放在同一个布局树中，然后调用 AdapterView.setEmptyView()自行处理。AdapterView 会根据其中 ListAdapter 的 isEmpty()方法的返回值选择显示其自身还是显示空视图。

重点：

一定要将 AdapterView 和空视图放入布局中，AdapterView 仅仅只是变换两者是否可见的参数，而绝对不会在布局树中插入或删除某个视图。

2.24.3 实现机制

下面将一个简单的 TextView 用作空视图。首先，在布局中放入这两个视图，参见程序清单 2-71。

程序清单 2-71 带有 AdapterView 和空视图的布局

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

```

<TextView
    android:id="@+id/myempty"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="No Items to Display"
/>
<ListView
    android:id="@+id/mylist"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
</FrameLayout>

```

然后在 Activity 中将空视图的引用提供给 ListView，让其进行管理(参见程序清单 2-72)。

程序清单 2-72 将空视图链接到列表的 Activity

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ListView list = (ListView)findViewById(R.id.mylist);
    TextView empty = (TextView)findViewById(R.id.myempty);
    //添加引用
    list.setEmptyView(empty);

    //继续给列表添加 Adapter 和数据
}

```

1. 让空视图更有趣

空视图不一定非得是简单无趣的 TextView。接下来做点对用户更有用的东西并且在列表为空时加上一个刷新按钮(参见程序清单 2-73)。

程序清单 2-73 交互式空布局

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:id="@+id/myempty"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="No Items to Display"
        />
        <Button
            android:layout_width="fill_parent"

```

```

        android:layout_height="wrap_content"
        android:text="Tap Here to Refresh"
    />
</LinearLayout>
<ListView
    android:id="@+id/mylist"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
</FrameLayout>

```

现在，一样是用前面的 Activity 代码，将一个完整的布局设为空视图，让用户可以对空数据进行一些操作。

2.25 自定义 ListView 行

2.25.1 问题

要自定义 ListView 中的各行外观。

2.25.2 解决方案

(API Level 1)

创建一个自定义的 XML 布局，将其传递给常见的适配器，或是扩展你自己的适配器，然后用自定义的状态 Drawable 覆盖背景和选中状态下的行。

2.25.3 实现机制

1. 简单的自定义

如果需要简单，那创建一个布局，链接到已有的 ListAdapter 即可。我们用 ArrayAdapter 为例加以介绍。ArrayAdapter 的参数是自定义的布局资源和用于填充该布局的 TextView 控件的 ID。让我们创建一个用作背景的自定义 Drawable 和一个满足这些要求的布局(参见程序清单 2-74 到程序清单 2-76)。

程序清单 2-74 res/drawable/row_background_default.xml

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#EFEFEF"
        android:endColor="#989898"
        android:type="linear"
        android:angle="270"
    />
</shape>

```

```

    />
</shape>

```

程序清单 2-75 res/drawable/row_background_pressed.xml

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#0B8CF2"
        android:endColor="#0661E5"
        android:type="linear"
        android:angle="270"
    />
</shape>

```

程序清单 2-76 res/drawable/row_background.xml

```

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

```

程序清单 2-77 是一个自定义布局，其中的文本是居中排列的，而不是左对齐的。

程序清单 2-77 res/layout/custom_row.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="10dip"
    android:background="@drawable/row_background">
    <TextView
        android:id="@+id/line1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
    />
</LinearLayout>

```

这个布局的背景是自定义的渐变状态列表，在这个列表中为每一行设置了默认状态和按下状态的外观。我们已经定义好了一个能与 `ArrayAdapter` 所期望的行为契合的布局，现在可以创建一个 `Activity`，调用这个列表，不必再进行其他任何自定义(参见程序清单 2-78)。

程序清单 2-78 使用了自定义行布局的 `Activity`

```

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

```

```

        ListView list = new ListView(this);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            R.layout.custom_row,
            R.id.line1,
            new String[] { "Bill", "Tom", "Sally", "Jenny" });
        list.setAdapter(adapter);

        setContentView(list);
    }

```

2. 更复杂的选项

有时自定义列表中的行意味着要扩展 `ListAdapter`。常见的情况是要在一行中放入多项数据或是有些数据并不是文本。在这个示例中，我们再次用自定义的 `Drawable` 作为背景，不过这里的布局会变得更有趣(参见程序清单 2-79)

程序清单 2-79 修改后的 `res/layout/custom_row.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dip">
    <ImageView
        android:id="@+id/leftimage"
        android:layout_width="32dip"
        android:layout_height="32dip"
    />
    <ImageView
        android:id="@+id/rightimage"
        android:layout_width="32dip"
        android:layout_height="32dip"
        android:layout_alignParentRight="true"
    />
    <TextView
        android:id="@+id/line1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/rightimage"
        android:layout_toRightOf="@id/leftimage"
        android:layout_centerVertical="true"
        android:gravity="center_horizontal"
    />
</RelativeLayout>

```

这个布局中同样有一个居中对齐的 `TextView`，但它的每条边都是一个 `ImageView`。要将这个布局应用到 `ListView`，我们需要扩展 SDK 中的 `ListAdapter`。具体扩展哪个 `ListAdapter` 取决于要在列表中呈现的数据来源。如果数据只是简单的字符串数组，扩展 `ArrayAdapter`

就足够了。如果数据更复杂些,就有可能需要扩展抽象的 `BaceAdapter`。需要扩展的方法只有 `getView()`, 这个方法负责控制列表中每一行的显示方式。

在这个示例中,数据只有简单的字符串数组,所以只需要简单扩展 `ArrayAdapter` 即可(参见程序清单 2-80)。

程序清单 2-80 显示新布局的 Activity 和自定义的 `ListAdapter`

```
public class MyActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ListView list = new ListView(this);
        setContentView(list);

        CustomAdapter adapter = new CustomAdapter(this,
            R.layout.custom_row,
            R.id.line1,
            new String[] { "Bill", "Tom", "Sally", "Jenny" });
        list.setAdapter(adapter);
    }

    private static class CustomAdapter extends ArrayAdapter<String> {

        public CustomAdapter(Context context, int layout, int resId,
            String[] items) {
            //调用 ArrayAdapter 的实现
            super(context, layout, resId, items);
        }

        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
            View row = convertView;
            //如果某一行没有被回收的话,则填充该行
            if(row == null) {
                row = LayoutInflater.from(getContext())
                    .inflate(R.layout.custom_row, parent, false);
            }
            String item = getItem(position);
            ImageView left = (ImageView)row.findViewById(R.id.leftimage);
            ImageView right = (ImageView)row.findViewById(R.id.rightimage);
            TextView text = (TextView)row.findViewById(R.id.line1);

            left.setImageResource(R.drawable.icon);
            right.setImageResource(R.drawable.icon);
            text.setText(item);

            return row;
        }
    }
}
```

```
    }
}
```

注意，这里用来创建适配器实例的构造函数与前面使用的构造函数相同，因为它们都继承自 `ArrayAdapter`。因为重载了适配器的视图显示机制，所以在这里将 `R.layout.custom_row` 和 `R.id.lin1` 传递给构造函数只是因为这是构造函数的必要参数。在这个示例中，这两个参数是不起作用的。

现在，当 `ListView` 要显示一行东西时，就会调用其适配器的 `getView()` 方法。我们已经对这个方法进行了自定义，以便能控制每一行内容的返回方式，`getView()` 方法有一个名为 `convertView` 的参数，这个参数对性能有重要影响。将 XML 文件转换为布局是一个代价很高的操作，为了减少其对系统的影响，在滚动时 `ListView` 会回收视图。如果回收的视图还能重用，就作为 `convertView` 传递给 `getView()` 方法。所以，为了确保列表的相应速度，要尽可能重用视图，而不是生成新的视图。

在这个示例中，调用 `getItem()` 得到列表(字符串数组)中位置的当前值，然后将 `TextView` 设为这个值并应用于该行。我们还可以给每行设置图片来凸显数据，不过这里值使用了应用程序的图标。

2.26 制作 `ListView` 的节头部

2.26.1 问题

需要创建一个有若干节内容的列表，其中每一节都有各自的头部。

2.26.2 解决方案

(API Level 1)

用这里定义的 `SimplerExpandableListAdapter` 和 `ExpandableListView` 就能实现。`Android` 本身并没有提供在列表中分解的扩展方式，不过它提供的 `ExpandableListView` 控件及其适配器可以处理分节列表中的二维数据结构。缺点就是 `SDK` 提供的适配器在处理简单数据结构时显得比较繁琐。

2.26.3 实现机制

看看 `SimplerExpandableListAdapter` 这个示例(参见程序清单 2-81)，这个类继承了 `BaseExpandableListAdapter`，可以处理字符串二维数组，另外还有一个单独的字符串数组用于保存每一节的标题。

程序清单 2-81 `SimplerExpandableListAdapter`

```
public class SimplerExpandableListAdapter extends BaseExpandableListAdapter {
    private Context mContext;
    private String[][] mContents;
```

```

private String[] mTitles;

public SimpleExpandableListAdapter(Context context, String[] titles,
    String[][] contents) {
    super();
    //检查参数
    if(titles.length != contents.length) {
        throw new IllegalArgumentException(
            "Titles and Contents must be the same size.");
    }

    mContext = context;
    mContents = contents;
    mTitles = titles;
}

//返回一个子条目
@Override
public String getChild(int groupPosition, int childPosition) {
    return mContents[groupPosition][childPosition];
}

//返回子条目的 id
@Override
public long getChildId(int groupPosition, int childPosition) {
    return 0;
}

//返回每行内容的视图
@Override
public View getChildView(int groupPosition, int childPosition,
    boolean isLastChild, View convertView, ViewGroup parent) {
    TextView row = (TextView)convertView;
    if(row == null) {
        row = new TextView(mContext);
    }
    row.setText(mContents[groupPosition][childPosition]);
    return row;
}

//返回每一节的条目数
@Override
public int getChildrenCount(int groupPosition) {
    return mContents[groupPosition].length;
}

//返回各节
@Override
public String[] getGroup(int groupPosition) {
    return mContents[groupPosition];
}

```

```

    }

    //返回节数
    @Override
    public int getGroupCount() {
        return mContents.length;
    }

    返回某节的 id
    @Override
    public long getGroupId(int groupPosition) {
        return 0;
    }

    //返回每一节标题的视图
    @Override
    public View getGroupView(int groupPosition, boolean isExpanded,
        View convertView, ViewGroup parent) {
        TextView row = (TextView)convertView;
        if(row == null) {
            row = new TextView(mContext);
        }
        row.setTypeface(Typeface.DEFAULT_BOLD);
        row.setText(mTitles[groupPosition]);
        return row;
    }

    @Override
    public boolean hasStableIds() {
        return false;
    }

    @Override
    public boolean isChildSelectable(int groupPosition, int childPosition) {
        return true;
    }
}

```

现在我们可以创建一个简单的数据结构，在示例的 `Activity` 中将其放入 `ExpandableListView`(参见程序清单 2-82)。

程序清单 2-82 使用 `SimplerExpandableListAdapter` 的 `Activity`

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //设置一个可扩展的列表
    ExpandableListView list = new ExpandableListView(this);
    list.setGroupIndicator(null);
    list.setChildIndicator(null);
}

```

```

//设置简单的数据和新的适配器
String[] titles = {"Fruits", "Vegetables", "Meats"};
String[] fruits = {"Apples", "Oranges"};
String[] veggies = {"Carrots", "Peas", "Broccoli"};
String[] meats = {"Pork", "Chicken"};
String[][] contents = {fruits, veggies, meats};
SimplerExpandableListAdapter adapter = new
SimplerExpandableListAdapter(this, titles, contents);

list.setAdapter(adapter);
setContentView(list);
}

```

1. 可打开和收起的清单

有这样一种使用 `ExpandableListView` 的方式：可打开和收起的。`ExpandableListView` 设计的初衷就是当用户单击每组数据的头部时，其中的内容会打开或收起。默认情况下，所有的内容都是收起的，只能看到头部的内容。

在某些情况下，这正是我们所需要的效果，但如果我们只想给每节内容加上一个头部的话，这种扩展和收起的功能就显得没必要了。为处理这种情况，再完成以下两个操作即可：

(1) 在 `Activity` 的代码中，打开所有的组。如下所示：

```

for(int i=0; i < adapter.getGroupCount(); i++) {
    list.expandGroup(i);
}

```

(2) 在适配器中，覆写 `onGroupCollapsed()`，强制重新打开。这需要添加一个指向该适配器的清单控件的引用。

```

@Override
public void onGroupCollapsed(int groupPosition) {
    list.expandGroup(groupPosition);
}

```

2.27 创建组合控件

2.27.1 问题

需要通过组合现有的元素来创建自定义的新控件。

2.27.2 解决方案

(API Level 1)

通过扩展通用的 `ViewGroup` 并添加所需的功能就能创建自定义的控件。创建自定义控件或是可重用的用户界面元素的最简单、最实用的方法就是利用 `Android SDK` 提供的控件

来创建组合控件。

2.27.3 实现机制

ViewGroup 及其子类 LinearLayout、RelativeLayout 等能帮助你摆放控件的位置，这样你就可以专注于添加所需的功能。

1. TextImageButton

下面将创建一个 Android SDK 中没有原生提供的控件：含有图片或文字的按钮。为了实现这个控件，我们创建一个 TextImageButton 类，这是对 FrameLayout 的扩展。其中包含一个用于放置文本的 TextView，以及一个用于放置图片的 ImageView(参见程序清单 2-83)。

程序清单 2-83 自定义 TextImageButton 控件

```
public class TextImageButton extends FrameLayout {

    private ImageView imageView;
    private TextView textView;
    /* 构造函数 */
    public TextImageButton(Context context) {
        this(context, null);
    }

    public TextImageButton(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public TextImageButton(Context context, AttributeSet attrs, int
defaultStyle) {
        //通过系统的按钮样式初始化父布局
        //这样会设置 clickable 属性和按钮背景来匹配当前的主题
        super(context, attrs, android.R.attr.buttonStyle);
        //创建子视图
        imageView = new ImageView(context, attrs, defaultStyle);
        textView = new TextView(context, attrs, defaultStyle);
        //创建子视图的 LayoutParams 为 WRAP_CONTENT 并居中显示
        FrameLayout.LayoutParams params = new FrameLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT,
            Gravity.CENTER);

        //添加视图
        this.addView(imageView, params);
        this.addView(textView, params);

        //如果有图片，切换到图片模式
        if(imageView.getDrawable() != null) {
            textView.setVisibility(View.GONE);
            imageView.setVisibility(View.VISIBLE);
        }
    }
}
```

```

        } else {
            textView.setVisibility(View.VISIBLE);
            imageView.setVisibility(View.GONE);
        }
    }

    /*访问器*/
    public void setText(CharSequence text) {
        //切换到文字模式
        textView.setVisibility(View.VISIBLE);
        imageView.setVisibility(View.GONE);
        //设置文字
        textView.setText(text);
    }

    public void setImageResource(int resId) {
        //切换到图片模式
        textView.setVisibility(View.GONE);
        imageView.setVisibility(View.VISIBLE);
        //设置图片
        imageView.setImageResource(resId);
    }

    public void setImageDrawable(Drawable drawable) {
        //切换到图片模式
        textView.setVisibility(View.GONE);
        imageView.setVisibility(View.VISIBLE);
        //设置图片
        imageView.setImageDrawable(drawable);
    }
}

```

SDK 中所有的控件都有 3 个构造函数。第一个构造函数的参数只有一个 Context，通常用于在代码中新建视图。另外两个构造函数用于将 XML 转换为视图，XML 文件中定义的属性会传递给 AttributeSet 参数。这里我们用 Java 的 this() 概念调用实际干活的函数，实现了这两个构造函数。这样就确保了我们能在 XML 布局中定义自定义的控件。如果不实现这两个属性化构造函数(attributed constructor)，就不能在 XML 布局中使用自定义的控件。

为了让 FrameLayout 看起来更像一个标准的按钮，我们在构造函数中传入了 android.R.attr.buttonStyle。这里定义了和当前主题匹配的样式并设置到了视图上。即设置了背景来匹配其他的按钮，还让视图变得可单击和可聚焦(因为这些样式就是系统样式的一部分)。只要有可能，都应该从当前主题中加载你自定义控件的外观，从而和应用程序的其他部分保持统一。

构造函数创建了一个 `TextView` 和一个 `ImageView`，将它们放入布局中。默认情况下，`FrameLayout` 不是一个可交互的视图，所以构造函数要将此控件设为可单击和可聚焦的，这样才能处理用户交互事件，而且我们还将视图的背景色设为系统默认的按钮背景，这样用户就能知道这个控件是可交互的。其他的代码是根据属性中传递的数据设置默认的显示模式(文字或图片)。

加入访问器函数是为了方便以后改变按钮的内容。这些函数还负责在内容变化时在文字和图片模式间进行切换。

因为这个自定义的控件并没有在 `android.view` 或 `android.widget` 包中，所以在 XML 布局中使用该控件必须要使用全名。程序清单 2-84 和程序清单 2-85 演示了含有该控件的 Activity。

程序清单 2-84 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <com.examples.customwidgets.TextImageButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Click Me!"
        android:textColor="#000" />
    <com.examples.customwidgets.TextImageButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />
</LinearLayout>
```

程序清单 2-85 使用了自定义控件的 Activity

```
public class MyActivity extends Activity {

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

注意，我们还是可以使用传统的属性来设置要显示的文字或图片。这是因为我们是用属性化构造函数来构造各个元素(`FrameLayout`、`TextView` 和 `ImageView`)，所以每个视图都会根据自己的需求设置参数，忽略其他参数。

如果我们定义使用该布局的 Activity，效果如图 2-23 所示。

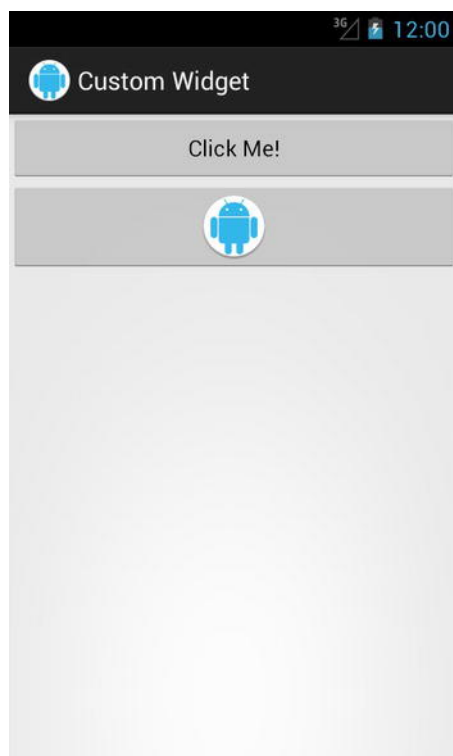


图 2-23 同时显示文字和图片的 TextImageButton

2.28 处理复杂的单击事件

2.28.1 问题

应用程序需要实现自定义的单点触摸或者多点触摸来与 UI 进行交互。

2.28.2 解决方案

(API Level 3)

可以使用框架中的 `GestureDetector` 和 `ScaleGestureDetector`, 或者干脆通过覆写 `onTouchEvent()` 和 `onInterceptTouchEvent()` 方法手动处理视图的所有触摸事件。前者可以很容易地在应用程序中添加复杂的手势控制。后者则非常强大, 但也有一些需要注意的地方。

Android 是通过一个自上而下的分发系统来处理 UI 上的触摸事件的, 这是框架在多层结构中发送消息的通用模式。触摸事件源于顶层窗口并首先发送给 `Activity`。然后, 这些事件被分发到视图结构中的根视图, 并从父视图依次传递到相应的子视图, 直到事件被处理或者整个视图链都已经传递过了。

父视图的工作就是要确认一个触摸事件应该发送给哪个子视图(通常通过检查视图的边界)以及以正确的顺序将事件分发出。如果可以分发给多个子视图(例如子视图是重叠

的), 父视图会按照子视图添加的顺序反向地将事件分发出去, 这样就可以保证 z-顺序中最高级别的视图(顶层视图)可以优先获得触摸事件。如果没有子视图处理事件, 父视图在该事件传回到视图层次之前则会获得处理该事件的机会。

任何视图都可以通过在 `onTouchEvent()` 方法中返回 `true` 来表明已经处理了某个特定的触摸事件, 这样该事件就不会再向其他地方分发了。所有的 `ViewGroup` 都可以通过 `onInterceptTouchEvent()` 回调方法拦截或窃取传递给其子视图的触摸事件。这在父视图需要控制某个特定动作的场景下非常有用, 例如 `ScrollView` 会在用户拖动手指时控制触摸事件。

在一个手指进行的过程中会有几种不同的触摸事件动作标识符:

- **ACTION_DOWN**: 当第一个手指单击屏幕时的第一个事件。这个事件通常是新手势的开始。
- **ACTION_MOVE**: 当一个手指在屏幕上改变位置时的事件。
- **ACTION_UP**: 最后一个手指离开屏幕时的事件。这个事件通常是一个手势的结束。
- **ACTION_CANCEL**: 这个事件是子视图收到的, 即在子视图接收事件时父视图拦截了手势事件。和 **ACTION_UP** 一样, 这标志着视图上的手势操作已经结束。
- **ACTION_POINTER_DOWN**: 当另一个手指单击屏幕时的事件。在转入多点触摸时很有用。
- **ACTION_POINTER_UP**: 当另一个手指离开屏幕时的事件。在转出多点触摸时很有用。

为了提高效率, 在一个视图没有处理 **ACTION_DOWN** 事件的情况下, **Android** 将不会向该视图传递后续的事件。因此, 如果你正在自定义处理触摸事件并希望处理后续的事件, 那么必须在 **ACTION_DOWN** 事件中返回 `true`。

如果在一个父 `ViewGroup` 的内部实现自定义触摸事件处理器, 你可能还需要在 `onInterceptTouchEvent()` 方法中编写代码。这个方法和 `onTouchEvent()` 类似, 如果返回 `true` 会接收手势后续所有的触摸事件(即 **ACTION_UP** 和 **ACTION_UP** 之前的所有事件)。这个操作是不可取消的, 在确定要截获所有事件之前不要轻易拦截这些事件。

最后, **Android** 提供了大量有用的阈值常量, 这些值可以根据设备屏幕的分辨率进行缩放, 可以用于构建自定义触摸交互。这些常数都保存在 `ViewConfiguration` 类中。本例中会使用到最小和最大急滑(**fling**)值以及触摸倾斜常量, 表示 **ACTION_MOVE** 事件变化到什么程度才表示是用户手指的真实移动动作。

2.28.3 实现机制

程序清单 2-86 演示了一个自定义的 `ViewGroup`, 该 `ViewGroup` 实现了平面滚动, 即在内容足够大的情况下, 允许用户在水平方向和垂直方向上进行滚动。该实现使用了 `GestureDetector` 来处理触摸事件。

程序清单 2-86 通过 `GestureDetector` 自定义 `ViewGroup`

```
public class PanGestureScrollView extends FrameLayout {

    private GestureDetector mDetector;
```

```

private Scroller mScroller;

/*最后位移事件的位置*/
private float mInitialX, mInitialY;
/*拖拽阈值*/
private int mTouchSlop;

public PanGestureScrollView(Context context) {
    super(context);
    init(context);
}

public PanGestureScrollView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init(context);
}

public PanGestureScrollView(Context context, AttributeSet attrs,
    int defStyle) {
    super(context, attrs, defStyle);
    init(context);
}

private void init(Context context) {
    mDetector = new GestureDetector(context, mListener);
    mScroller = new Scroller(context);
    //获得触摸阈值的系统常量
    mTouchSlop = ViewConfiguration.get(context).getScaledTouchSlop();
}

/*
 * 覆盖 measureChild... 的实现来保证子视图可以尽可能大。
 * 默认实现会强制一些子视图会和该视图一样大
 */
@Override
protected void measureChild(View child, int parentWidthMeasureSpec,
    int parentHeightMeasureSpec) {
    int childWidthMeasureSpec;
    int childHeightMeasureSpec;

    childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(0,
        MeasureSpec.UNSPECIFIED);
    childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(0,
        MeasureSpec.UNSPECIFIED);

    child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
}

@Override
protected void measureChildWithMargins(View child,

```

```

        int parentWidthMeasureSpec, int widthUsed,
        int parentHeightMeasureSpec, int heightUsed) {
    final MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();

    final int childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(
        lp.leftMargin + lp.rightMargin, MeasureSpec.UNSPECIFIED);
    final int childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(
        lp.topMargin + lp.bottomMargin, MeasureSpec.UNSPECIFIED);

    child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
}

//处理所有触摸事件的监听器
private SimpleOnGestureListener mListener = new SimpleOnGestureListener() {
    public boolean onDown(MotionEvent e) {
        //取消当前的急滑
        if (!mScroller.isFinished()) {
            mScroller.abortAnimation();
        }
        return true;
    }

    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
        float velocityY) {
        //调用一个辅助方法来启动滚动动画
        fling((int) -velocityX / 3, (int) -velocityY / 3);
        return true;
    }

    public boolean onScroll(MotionEvent e1, MotionEvent e2,
        float distanceX, float distanceY) {
        //任意视图都可以调用它的 scrollBy()进行滚动
        scrollBy((int) distanceX, (int) distanceY);
        return true;
    }
};

@Override
public void computeScroll() {
    if (mScroller.computeScrolloffset()) {
        //会在 ViewGroup 绘制时调用。
        //我们使用这个方法保证急滑动画的完成
        int oldX = getScrollX();
        int oldY = getScrollY();
        int x = mScroller.getCurrX();
        int y = mScroller.getCurrY();

        if (getChildCount() > 0) {
            View child = getChildAt(0);
            x = clamp(x, getWidth() - getPaddingRight() - getPaddingLeft(),

```

```

        child.getWidth());
        y = clamp(y,
            getHeight() - getPaddingBottom() - getPaddingTop(),
            child.getHeight());
        if (x != oldX || y != oldY) {
            scrollTo(x, y);
        }
    }
    //在动画完成前会一直绘制
    postInvalidate();
}

//覆写方法进行每个滚动请求的边界检查
@Override
public void scrollTo(int x, int y) {
    // 我们依赖 View.scrollTo 调用 scrollTo
    if (getChildCount() > 0) {
        View child = getChildAt(0);
        x = clamp(x, getWidth() - getPaddingRight() - getPaddingLeft(),
            child.getWidth());
        y = clamp(y, getHeight() - getPaddingBottom() - getPaddingTop(),
            child.getHeight());
        if (x != getScrollX() || y != getScrollY()) {
            super.scrollTo(x, y);
        }
    }
}

/*
 * 监控传递给子视图的触摸事件并且一旦拖曳就进行拦截
 */
@Override
public boolean onInterceptTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            mInitialX = event.getX();
            mInitialY = event.getY();
            //将按下事件传给手势检测器，这样当/如果拖曳开始就有了上下文
            mDetector.onTouchEvent(event);
            break;
        case MotionEvent.ACTION_MOVE:
            final float x = event.getX();
            final float y = event.getY();
            final int yDiff = (int) Math.abs(y - mInitialY);
            final int xDiff = (int) Math.abs(x - mInitialX);
            //检查 x 或者 y 上的距离是否适合拖曳
            if (yDiff > mTouchSlop || xDiff > mTouchSlop) {
                //开始捕捉事件
                return true;
            }
    }
}

```

```

        }
        break;
    }
    return super.onInterceptTouchEvent(event);
}

/*
 * 将我们接受的所有触摸事件传给检测器去处理
 */
@Override
public boolean onTouchEvent(MotionEvent event) {
    return mDetector.onTouchEvent(event);
}

/*
 * 初始化 Scroller 和开始重新绘制的工具方法
 */
public void fling(int velocityX, int velocityY) {
    if (getChildCount() > 0) {
        int height = getHeight() - getPaddingBottom() - getPaddingTop();
        int width = getWidth() - getPaddingLeft() - getPaddingRight();
        int bottom = getChildAt(0).getHeight();
        int right = getChildAt(0).getWidth();

        mScroller.fling(getScrollX(), getScrollY(), velocityX, velocityY,
            0, Math.max(0, right - width), 0,
            Math.max(0, bottom - height));
        invalidate();
    }
}

/*
 * 用来进行边界检查的辅助方法
 */
private int clamp(int n, int my, int child) {
    if (my >= child || n < 0) {
        //子视图超过了父视图的边界或者小于父视图，不能滚动
        return 0;
    }
    if ((my + n) > child) {
        //请求的滚动超出了子视图的右边界
        return child - my;
    }
    return n;
}
}

```

与 `ScrollView`、`HorizontalScrollView` 类似，这个示例有一个子视图并可以根据用户行为滚动它的内容。这个示例的多数代码与触摸事件的处理并没有直接关系，而是处理滚动并让滚动位置不要超过子视图的边界。

作为一个 `ViewGroup`，第一个可以看到所有触摸事件的地方就是 `onInterceptTouchEvent()`。在这个方法中我们必须分析用户的触摸行为从而确定是否是真正的拖动。这个方法中 `ACTION_DOWN` 和 `ACTION_MOVE` 的处理一起决定了用户的手指移动了多远，如果大于系统的触摸阈值常量，我们才认为是拖动事件并拦截后续触摸事件。这种做法允许子视图可以接收简单的单击事件，所以按钮和其他控件可以放心作为这个视图的子视图并且依然会得到单击事件。如果该视图没有可交互的子视图控件，事件将会被直接传递到我们的 `onTouchEvent()` 方法中，但因为我们允许这种情况发生，所以这里做了一个初始的检查。

这里的 `onTouchEvent()` 方法很简单，因为所有的事件都被转发到了 `GestureDetector` 中，它会追踪和计算用户正在做的特定动作。然后我们会通过 `SimpleOnGestureListener` 对那些事件进行响应，特别是 `onScroll()` 和 `onFling()` 事件。为了保证 `GestureDetector` 能够准确地设置手势的初始触点，我们还在 `onInterceptTouchEvent()` 向它转发了 `ACTION_DOWN` 事件。

`onScroll()` 在用户的手指移动一段距离时会被重复调用。所以，在手指拖动时，我们可以很方便地将这些值直接传递给视图的 `scrollBy()` 来移动视图的内容。

`onFling()` 中需要做稍微多一点的工作。急滑(fling)就是用户在屏幕上快速移动手指并抬起的动作。这个动作期望的结果就是一个惯性的滚动。同样，当用户手指抬起时会计算手指的速度，但必须依然保持滚动动画。这就是引入 `Scroller` 的原因。`Scroller` 是框架的一个控件用来通过用户的输入值和时间差值动画设置来让视图滚动起来。本例的动画四通过调用 `Scroller` 的 `fling()` 方法并刷新视图实现的。

注意：

如果目标版本为 API Level 9 或者更高，可以使用 `OverScroller` 代替 `Scroller`，它将为较新的设备提供更好的性能。还允许包含拉到底发光的动画(overscroll glow)。可以通过传入一个自定义的 `Interpolator` 加工急滑动画。

这会启动一个循环进程，在这个进程中框架会定期调用 `computeScroll()` 来绘制视图，我们刚好通过这个时机来检查 `Scroller` 当前的状态并且将视图向前滚动(如果动画未完成的话)。这也是开发者对 `Scroller` 感到困惑的地方。该控件是用来让视图动起来的，但实际上却没有制作任何的动画。它只是简单地提供了每个绘制帧移动的时机和距离。应用程序必须调用 `computeScrollOffset()` 来获得新位置然后再调用一个方法(本例中为 `scrollTo()` 方法)渐进地改变视图。

最后一个 `GestureDetector` 中使用的回调方法是 `onDown()`，它会在侦测器收到 `ACTION_DOWN` 事件时得到调用。如果用户手指单击屏幕，我们会通过这个回调方法终止所有当前的急滑动画。程序清单 2-87 显示了我们该如何在 `Activity` 中使用这个自定义视图。

程序清单 2-87 使用 `PanGestureScrollView` 的 `Activity`

```
public class PanScrollActivity extends Activity {

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

```

PanGestureScrollView scrollView = new PanGestureScrollView(this);

LinearLayout layout = new LinearLayout(this);
layout.setOrientation(LinearLayout.VERTICAL);
for(int i=0; i < 5; i++) {
    ImageView iv = new ImageButton(this);
    iv.setImageResource(R.drawable.ic_launcher);
    //让每个视图可以足够大来请求滚动
    layout.addView(iv, new LinearLayout.LayoutParams(1000, 500));
}

scrollView.addView(layout);
setContentView(scrollView);
}
}

```

我们使用大量的 `ImageButton` 实例来填充这个自定义 `PanGestureScrollView`，这就是为了演示这些按钮都是可以单击的并且可以接收单击事件，但是只要你拖动或急滑手指，视图就会开始滚动。想要了解 `GestureDetector` 为我们做了多少工作，查看程序清单 2-88，它实现了相同的功能但需要在 `onTouchEvent()` 中手动处理所有的触摸事件。

程序清单 2-88 使用自定义触摸处理的 `PanScrollView`

```

public class PanScrollView extends FrameLayout {

    //急滑控件
    private Scroller mScroller;
    private VelocityTracker mVelocityTracker;

    /* 上一个移动事件的位置 */
    private float mLastTouchX, mLastTouchY;
    /* 拖动阈值 */
    private int mTouchSlop;
    /* 急滑的速度 */
    private int mMaximumVelocity, mMinimumVelocity;
    /* 拖动锁 */
    private boolean mDragging = false;

    public PanScrollView(Context context) {
        super(context);
        init(context);
    }

    public PanScrollView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context);
    }

    public PanScrollView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }
}

```

```

        init(context);
    }

    private void init(Context context) {
        mScroller = new Scroller(context);
        mVelocityTracker = VelocityTracker.obtain();
        //获得触摸阈值的系统常量
        mTouchSlop = ViewConfiguration.get(context).getScaledTouchSlop();
        mMaximumVelocity = ViewConfiguration.get(context)
            .getScaledMaximumFlingVelocity();
        mMinimumVelocity = ViewConfiguration.get(context)
            .getScaledMinimumFlingVelocity();
    }

    /*
    * 覆写 measureChild...的实现来保证子视图可以尽可能地大。
    * 默认实现会强制一些子视图和该视图一样大
    */
    @Override
    protected void measureChild(View child, int parentWidthMeasureSpec,
        int parentHeightMeasureSpec) {
        int childWidthMeasureSpec;
        int childHeightMeasureSpec;

        childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(0,
            MeasureSpec.UNSPECIFIED);
        childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(0,
            MeasureSpec.UNSPECIFIED);

        child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
    }

    @Override
    protected void measureChildWithMargins(View child,
        int parentWidthMeasureSpec, int widthUsed,
        int parentHeightMeasureSpec, int heightUsed) {
        final MarginLayoutParams lp = (MarginLayoutParams) child
            .getLayoutParams();

        final int childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(
            lp.leftMargin + lp.rightMargin, MeasureSpec.UNSPECIFIED);
        final int childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(
            lp.topMargin + lp.bottomMargin, MeasureSpec.UNSPECIFIED);

        child.measure(childWidthMeasureSpec, childHeightMeasureSpec);
    }

    @Override
    public void computeScroll() {
        if (mScroller.computeScrollOffset()) {

```

```

        //会在 ViewGroup 绘制时调用。
        //我们使用这个方法保证急滑动画的完成
        int oldX = getScrollX();
        int oldY = getScrollY();
        int x = mScroller.getCurrX();
        int y = mScroller.getCurrY();

        if (getChildCount() > 0) {
            View child = getChildAt(0);
            x = clamp(x, getWidth() - getPaddingRight() - getPaddingLeft(),
                    child.getWidth());
            y = clamp(y,
                    getHeight() - getPaddingBottom() - getPaddingTop(),
                    child.getHeight());
            if (x != oldX || y != oldY) {
                scrollTo(x, y);
            }
        }
        //在动画完成前会一直绘制
        postInvalidate();
    }
}

//覆写方法进行每个滚动请求的边界检查
@Override
public void scrollTo(int x, int y) {
    //我们依赖 View.scrollTo 调用 scrollTo
    if (getChildCount() > 0) {
        View child = getChildAt(0);
        x = clamp(x, getWidth() - getPaddingRight() - getPaddingLeft(),
                child.getWidth());
        y = clamp(y, getHeight() - getPaddingBottom() - getPaddingTop(),
                child.getHeight());
        if (x != getScrollX() || y != getScrollY()) {
            super.scrollTo(x, y);
        }
    }
}

/*
 * 监控传递给子视图的触摸事件并且一旦拖曳就进行拦截
 * 如果子视图时可交互的(例如, 按钮), 会依然允许子视图接收触摸事件
 */
@Override
public boolean onInterceptTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            //终止所有正在进行的急滑动画
            if (!mScroller.isFinished()) {
                mScroller.abortAnimation();
            }
    }
}

```

```

    }
    //还原速度跟踪器
    mVelocityTracker.clear();
    mVelocityTracker.addMovement(event);
    //保存初始触点
    mLastTouchX = event.getX();
    mLastTouchY = event.getY();
    break;
case MotionEvent.ACTION_MOVE:
    final float x = event.getX();
    final float y = event.getY();
    final int yDiff = (int) Math.abs(y - mLastTouchY);
    final int xDiff = (int) Math.abs(x - mLastTouchX);
    //检查 x 或者 y 上的距离是否适合拖曳
    if (yDiff > mTouchSlop || xDiff > mTouchSlop) {
        mDragging = true;
        mVelocityTracker.addMovement(event);
        //我们自己开始捕捉事件
        return true;
    }
    break;
case MotionEvent.ACTION_CANCEL:
case MotionEvent.ACTION_UP:
    mDragging = false;
    mVelocityTracker.clear();
    break;
}

return super.onInterceptTouchEvent(event);
}

/*
 * 将我们接受到的所有触摸事件传给检测器去处理
 */
@Override
public boolean onTouchEvent(MotionEvent event) {
    mVelocityTracker.addMovement(event);

    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            //我们已经保存了初始触点，但如果这里发现有子视图没有捕捉事件，
            //还是需要返回 true 的。
            return true;
        case MotionEvent.ACTION_MOVE:
            final float x = event.getX();
            final float y = event.getY();
            float deltaY = mLastTouchY - y;
            float deltaX = mLastTouchX - x;
            // Check for slop on direct events
            //检查各个方向上的阈值

```

```

        if ((Math.abs(deltaY) > mTouchSlop || Math.abs(deltaX) > mTouchSlop)
            && !mDragging) {
            mDragging = true;
        }
        if (mDragging) {
            //滚动视图
            scrollBy((int) deltaX, (int) deltaY);
            //更新最后一个触摸事件
            mLastTouchX = x;
            mLastTouchY = y;
        }
        break;
    case MotionEvent.ACTION_CANCEL:
        mDragging = false;
        //终止所有进行的急滑动画
        if (!mScroller.isFinished()) {
            mScroller.abortAnimation();
        }
        break;
    case MotionEvent.ACTION_UP:
        mDragging = false;
        //计算当前的速度, 如果高于最小阈值则启动一个急滑
        mVelocityTracker.computeCurrentVelocity(1000, mMaximumVelocity);
        int velocityX = (int) mVelocityTracker.getXVelocity();
        int velocityY = (int) mVelocityTracker.getYVelocity();
        if (Math.abs(velocityX) > mMinimumVelocity
            || Math.abs(velocityY) > mMinimumVelocity) {
            fling(-velocityX, -velocityY);
        }
        break;
    }
    return super.onTouchEvent(event);
}

/*
 * 初始化 Scroller 和开始重新绘制的工具方法
 */
public void fling(int velocityX, int velocityY) {
    if (getChildCount() > 0) {
        int height = getHeight() - getPaddingBottom() - getPaddingTop();
        int width = getWidth() - getPaddingLeft() - getPaddingRight();
        int bottom = getChildAt(0).getHeight();
        int right = getChildAt(0).getWidth();

        mScroller.fling(getScrollX(), getScrollY(), velocityX, velocityY,
            0, Math.max(0, right - width), 0,
            Math.max(0, bottom - height));

        invalidate();
    }
}

```

```

    }

    /*
     * 用来进行边界检查的辅助方法
     */
    private int clamp(int n, int my, int child) {
        if (my >= child || n < 0) {
            //子视图超过了父视图的边界或者小于父视图，不能滚动
            return 0;
        }
        if ((my + n) > child) {
            //请求的滚动超出了子视图的右边界
            return child - my;
        }
        return n;
    }
}

```

本例中, `onInterceptTouchEvent()` 和 `onTouchEvent()` 中的工作会多一点。如果当前存在子视图处理初始的触摸事件, 那么在我们接管事件之前, `ACTION_DOWN` 和开始的一些移动事件都会通过 `onInterceptTouchEvent()` 进行传递; 但是, 如果并不存在可交互的子视图, 所有的初始触摸事件都会直接传递到 `onTouchEvent()` 中。在这两个方法中, 我们必须都要对初始拖动进行阈值检查, 如果确实开始了拖动事件, 会设置一个标识。一旦标识了用户正在拖动, 滚动视图的代码就和之前的一样了, 即调用 `scrollBy()`。

提示:

只要某个 `ViewGroup` 通过 `onTouchEvent()` 返回了 “true”, 即使没有显式地请求拦截, 也不会有事件被传递到 `onInterceptTouchEvent()`。

想要实现急滑效果, 我们必须手动使用 `VelocityTracker` 对象跟踪用户的滚动速度。该对象会将发生的事件通过 `addMovement()` 方法收集起来, 然后通过 `computeCurrentVelocity()` 计算相应的平均速度。我们的自定义视图会根据 `ViewConfiguration` 最小速度在每次用户抬起手指时计算这个速度值从而决定是否要开始一段急滑动画。

提示:

在不需要显示返回 `true` 来处理一个事件的情形下, 最好返回父类的实现而不是返回 `false`。通常父类会有很多关于 `View` 和 `ViewGroup` 的隐藏处理(你不想覆写的)。

程序清单 2-89 使用 `PanScrollView` 的 `Activity`

```

public class PanScrollActivity extends Activity {

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

        PanScrollView scrollView = new PanScrollView(this);
    }
}

```

```

        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        for(int i=0; i < 5; i++) {
            ImageView iv = new ImageView(this);
            iv.setImageResource(R.drawable.ic_launcher);
            layout.addView(iv, new LinearLayout.LayoutParams(1000, 500));
        }

        scrollView.addView(layout);
        setContentView(scrollView);
    }
}

```

我们将视图的内容设定为 `ImageView` 而非 `ImageButton`，从而演示子视图不能交互时的对比效果。

多点触摸处理

(API Level 8)

现在，让我们看一个处理多点触摸事件的示例。程序清单 2-90 是一个自定义的添加了多点触摸交互的 `ImageView`。

程序清单 2-90 处理多点触摸的 `ImageView`

```

public class RotateZoomImageView extends ImageView {

    private ScaleGestureDetector mScaleDetector;
    private Matrix mImageMatrix;
    /*上次的旋转角度*/
    private int mLastAngle = 0;
    /* 变换时的轴点 */
    private int mPivotX, mPivotY;

    public RotateZoomImageView(Context context) {
        super(context);
        init(context);
    }

    public RotateZoomImageView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context);
    }

    public RotateZoomImageView(Context context, AttributeSet attrs, int
defStyle) {
        super(context, attrs, defStyleAttr);
        init(context);
    }

    private void init(Context context) {

```

```

        mScaleDetector = new ScaleGestureDetector(context, mScaleListener);

        setScaleType(ScaleType.MATRIX);
        mImageMatrix = new Matrix();
    }

    /*
     * 在 onSizeChanged() 中根据视图的尺寸计算一些值。
     * 这个视图在 init() 时并没有尺寸，因此必须等到这个回调方法才能得到尺寸。
     */
    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        if (w != oldw || h != oldh) {
            // 将图片移到视图的中央
            int translateX = Math.abs(w - getDrawable().getIntrinsicWidth()) / 2;
            int translateY = Math.abs(h - getDrawable().getIntrinsicHeight()) / 2;
            mImageMatrix.setTranslate(translateX, translateY);
            setImageMatrix(mImageMatrix);
            // 得到未来缩放和旋转变换时的中轴点
            mPivotX = w / 2;
            mPivotY = h / 2;
        }
    }

    private SimpleOnScaleGestureListener mScaleListener =
        new SimpleOnScaleGestureListener() {
            @Override
            public boolean onScale(ScaleGestureDetector detector) {
                // ScaleGestureDetector 会根据手指的分开和合拢计算出一个缩放因子
                float scaleFactor = detector.getScaleFactor();
                // 将缩放因子传给图片进行缩放
                mImageMatrix.postScale(scaleFactor, scaleFactor, mPivotX,
                    mPivotY);
                setImageMatrix(mImageMatrix);

                return true;
            }
        };

    /*
     * 处理两个手指的事件来旋转图片。
     * 这个方法根据触点间的角度变化对图片进行相应的旋转。
     * 当用户旋转手指时，图片也会跟着旋转
     */
    private boolean doRotationEvent(MotionEvent event) {
        // 计算两个手指间的角度
        float deltaX = event.getX(0) - event.getX(1);
        float deltaY = event.getY(0) - event.getY(1);
        double radians = Math.atan(deltaY / deltaX);
        // 转化为角度

```

```

int degrees = (int)(radians * 180 / Math.PI);

switch (event.getAction()) {
case MotionEvent.ACTION_DOWN:
    //记住初始的角度
    mLastAngle = degrees;
    break;
case MotionEvent.ACTION_MOVE:
    //ATAN 返回了一个转化后的-90 度到+90 度的值,
    //这样在两个手指垂直触摸时可以得到一个翻转信号和相应的角度
    //这种情况下会将图片在我们侦测到方向上旋转一个很小的角度(5 度)

    if ((degrees - mLastAngle) > 45) {
        //逆时针旋转(可以超出边界)
        mImageMatrix.postRotate(-5, mPivotX, mPivotY);
    } else if ((degrees - mLastAngle) < -45) {
        //顺时针旋转(可以超出边界)
        mImageMatrix.postRotate(5, mPivotX, mPivotY);
    } else {
        //正常旋转, 旋转角度即为手指的旋转角度
        mImageMatrix.postRotate(degrees - mLastAngle, mPivotX,
mPivotY);
    }
    //将旋转矩阵发送给图片
    setImageMatrix(mImageMatrix);
    //保存当前的角度
    mLastAngle = degrees;
    break;
}

return true;
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_DOWN) {
        //我们并不直接关心这个事件, 但会声明要处理后续的多点触摸
        return true;
    }

    switch (event.getPointerCount()) {
    case 3:
        //3 个手指时, 使用 ScaleGestureDetector 缩放图片
        return mScaleDetector.onTouchEvent(event);
    case 2:
        //两个手指时, 根据手指操作旋转图片
        return doRotationEvent(event);
    default:
        //忽略这个事件
        return super.onTouchEvent(event);
    }
}

```

```

    }
}

```

这个示例创建了一个自定义的 `ImageView` 监听多点触摸事件并及时变换图像的内容。这个视图可以侦测到的两种事件就是 2 个手指的旋转操作和 3 个手机的缩放操作。旋转事件是通过每个 `MotionEvent` 来处理的，缩放事件则是通过 `ScaleGestureDetector` 来处理的。这个视图的 `ScaleType` 被设置为 `MATRIX`，这样就可以让我们通过不同的 `Matrix` 变换调整图片的外观。

当该视图构建完成后，就会触发 `onSizeChanged()` 回调方法。这个方法可以被多次调用，所以我们只会在上次值和本次值不同时才会去计算相应的值。这里，我们会根据视图的尺寸设置一些值，以便将图片放置到视图的中央并稍后进行正确的变换。同时我们执行了第一次变换，即将图片移动到视图的中央。

我们会分析 `onTouchEvent()` 接收的触摸事件来决定处理哪个事件。通过检查每个 `MotionEvent` 的 `getPointerCount()` 方法，可以判断触摸了几个手指并将事件传递给相应的处理器。正如之前说的一样，这里必须也要处理第一个 `ACTION_DOWN` 事件；否则用户其他手指的后续触摸事件将不会传递到这个视图。虽然我们不想对这个事件做任何操作，但仍然需要显式地返回 `true`。

`ScaleGestureDetector` 会分析应用程序反馈的每个触摸事件，当出现缩放事件时，就调用一系列的 `OnScaleGestureListener` 回调方法。最重要的回调方法就是 `onScale()`，它在用户手指移动时就会被经常调用，但开发者可以使用 `onScaleBegin()` 和 `onScaleEnd()` 在手势开始和结束时进行一些操作。

`ScaleGestureDetector` 提供了很多有用的计算值，应用程序可以使用这些值来修改 UI。

- `getCurrentSpan()`: 获得该手势中两个触点间的距离。
- `getFocusX()/getFocusY()`: 获得当前手势的焦点坐标。它是触点收缩时的平均位置。
- `getScaleFactor()`: 得到当前事件和之前事件之间的变化比例。多个手指分开时，这个值会稍微大于 1，收拢时会稍微小于 1。

这个示例从侦测器中得到缩放因子并使用它通过 `postScale()` 设置图像的 `Matrix` 从而缩放视图中的图片内容。

我们两个手指的旋转事件是手动处理的。对于每个传入的事件，会通过 `getX()` 和 `getY()` 计算两个手指间 x 和 y 方向的距离。`getX()` 和 `getY()` 方法使用的参数为触点的索引，0 表示第一个触点，1 表示第二个触点。

使用这些距离，我们可以使用一点三角函数的知识计算出两个手指形成的无形直线的角度。我们将使用这个角度来进行变换。`ACTION_DOWN` 时，不管角度值是什么都会把它作为初始值并简单地保存起来。在接下来的 `ACTION_MOVE` 事件中，会根据每次触摸事件的角度变化发送给图片相应的旋转矩阵。

有一种边界情况这个示例必须要处理，必须使用 `Math.atan()` 三角函数。这个方法会返回一个介于 -90° 和 $+90^\circ$ 的角度值，而有种翻转发生在一个手指垂直地位于另一个手指上方的情况。这个问题会导致触摸角度不再是逐渐改变的：在手指旋转时，角度值会从 $+90^\circ$ 立即变为 -90° ，从而导致图片跳动。为了解决这个问题，我们会检查之前角度和当前角度

超过这种边界值的情况，然后在相同的方向做 5° 的小旋转从而保证动画的流畅性。

注意，变换图片的所有操作都是使用 `postScale()` 和 `postRotate()` 的，而不是之前的这些方法的如 `setTranslation()` 的 `setXXX` 的版本。这是因为每个变换都只是一种新增的变换，这意味着只能适当地改变当前的状态而不是替换它。调用 `setScale()` 或者 `setRotate()` 将会清除当前的状态从而导致只剩下 `Matrix` 中的变换。

这些变换都是围绕我们在 `onSizeChanged()` 中计算出的轴点(视图的中央)进行的。这么做是因为默认情况下变换发生在 (0,0)，即视图的左上角。因为我们已经将图片移动到了视图中央，所以需要保证所有的变换也是发生在同样的中央轴点。

2.29 转发触摸事件

2.29.1 问题

你的应用程序中的一些视图或者触摸目标非常小导致手指很难准确地触摸到它。

2.29.2 解决方案

(API Level 1)

使用 `TouchDelegate` 指定一个任意的矩形区域来向小视图转发触摸事件。`TouchDelegate` 的设计宗旨就是为父 `ViewGroup` 关联一个特定的区域，该区域侦测到触摸事件后会将该事件转发给它的子视图。`TouchDelegate` 会发送每个事件到目标视图，就像触摸目标视图自己一样。

2.29.3 实现机制

程序清单 2-91 和 2-92 演示了如何在一个自定义父 `ViewGroup` 中使用 `TouchDelegate`。

程序清单 2-91 自定义父视图实现 `TouchDelegate`

```
public class TouchDelegateLayout extends FrameLayout {

    public TouchDelegateLayout(Context context) {
        super(context);
        init(context);
    }

    public TouchDelegateLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        init(context);
    }

    public TouchDelegateLayout(Context context, AttributeSet attrs, int
        defStyle) {
```

```

        super(context, attrs, defStyle);
        init(context);
    }

    private CheckBox mButton;
    private void init(Context context) {
        //创建一个很小的子视图, 我们会将触摸事件转发给它
        mButton = new CheckBox(context);
        mButton.setText("Tap Anywhere");

        LayoutParams lp = new FrameLayout.LayoutParams(FrameLayout.LayoutParams.WRAP_CONTENT,
            FrameLayout.LayoutParams.WRAP_CONTENT, Gravity.CENTER);
        addView(mButton, lp);
    }

    /*
    *TouchDelegate 会将该视图(父视图)的某个特定矩形区域作为代理区域将
    *所有触摸事件转发给 CheckBox(子视图)。这里, 矩形区域即为父视图的全部大小
    */
    *这个过程必须得在视图确定了大小以后进行, 这样才能知道矩形应该有多大,
    *所以我们选择在 onSizeChanged() 中添加代理区域
    */
    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        if (w != oldw || h != oldh) {
            //将该视图的整个区域作为代理区域
            Rect bounds = new Rect(0, 0, w, h);
            TouchDelegate delegate = new TouchDelegate(bounds, mButton);
            setTouchDelegate(delegate);
        }
    }
}

```

程序清单 2-92 示例 Activity

```

public class DelegateActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TouchDelegateLayout layout = new TouchDelegateLayout(this);

        setContentView(layout);
    }
}

```

这个示例中, 我们创建了一个父视图, 其中包含了一个居中显示的复选框。这个视图还包含一个 `TouchDelegate`, 它会将父视图区域内收到的触摸事件转发给复选框。因为我们想让父布局的整个区域转发触摸事件, 所以会等到 `onSizeChanged()` 被调用后再去构建和关联 `TouchDelegate` 实例。如果在构造函数中构建将不会生效, 因为在执行构造函数时, 视

图还有被测量并没有可以读取的尺寸大小。

Android 框架会将没有被处理的触摸事件自动从 `TouchDelegate` 分发到它的代理视图，因此无需额外代码即可转发这些事件。在图 2-24 中可以看到，应用程序在距离复选框很远的地方收到触摸事件后，复选框会做相应的响应，如同它自己直接被触摸了一样。

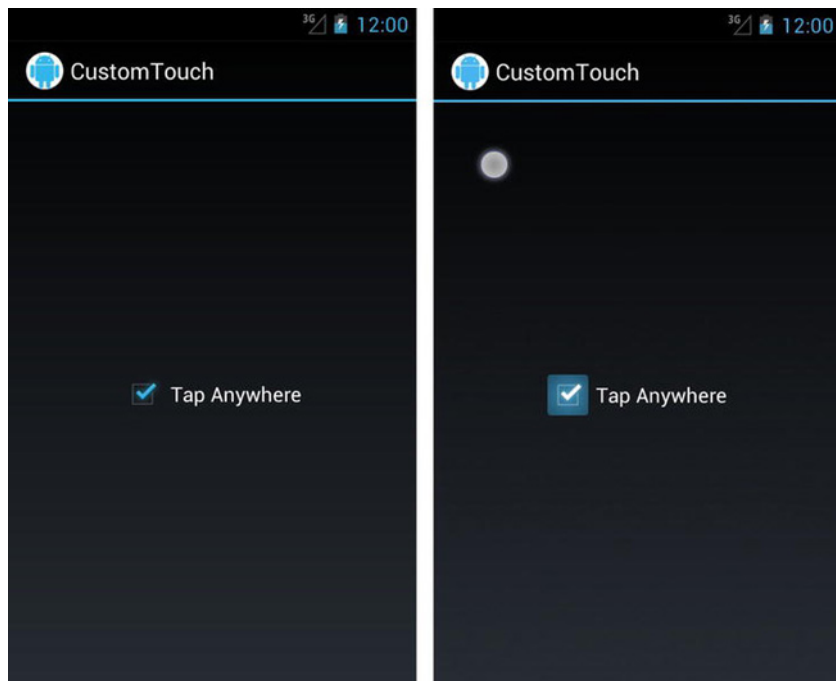


图 2-24 带有复选框的示例应用程序(左边)，复选框会接收被转发的触摸事件(右边)

1. 自定义触摸转发(远程滚动条)

`TouchDelegate` 非常适合于转发单击事件，但它有一个缺点，就是每个被转发的事件转发到代理视图后都会定位到代理视图的中间位置。这也就意味着如果想要通过 `TouchDelegate` 转发一系列 `ACTION_MOVE` 事件的话，结果将不会如你所愿，因为这时会显示手指并没有移动过(每次都定位到同一个点上)。

如果想要以一种更加精确的方式重新路由触摸事件，可以通过手动地调用目标视图的 `dispatchTouchEvent()` 方法来实现。参见程序清单 2-93 和 2-94 了解相应的实现机制。

程序清单 2-93 `res/layout/main.xml`

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

    <TextView
        android:id="@+id/text_touch"
        android:layout_width="match_parent"
```

```

        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center"
        android:text="Scroll Anywhere Here" />

<HorizontalScrollView
    android:id="@+id/scroll_view"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:background="#CCC">
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="horizontal" >
        <ImageView
            android:layout_width="250dp"
            android:layout_height="match_parent"
            android:scaleType="fitXY"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="250dp"
            android:layout_height="match_parent"
            android:scaleType="fitXY"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="250dp"
            android:layout_height="match_parent"
            android:scaleType="fitXY"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="250dp"
            android:layout_height="match_parent"
            android:scaleType="fitXY"
            android:src="@drawable/ic_launcher" />
    </LinearLayout>
</HorizontalScrollView>
</LinearLayout>

```

程序清单 2-94 转发触摸事件的 Activity

```

public class RemoteScrollActivity extends Activity implements
    View.OnTouchListener {

    private TextView mTouchText;
    private HorizontalScrollView mScrollView;

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

```

```

        mTouchText = (TextView) findViewById(R.id.text_touch);
        mScrollView = (HorizontalScrollView) findViewById(R.id.scroll_view);
        //为顶层视图关联一个触摸事件的监听器
        mTouchText.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        //如果需要的话可以修改事件位置。
        //这里我们将事件的垂直方向的位置设置为 HorizontalScrollView 的中间。

        //视图需要的事件位置都是相对于自己的坐标。
        event.setLocation(event.getX(), mScrollView.getHeight() / 2);

        //将 TextView 上的每个事件转发到 HorizontalScrollView 上
        mScrollView.dispatchTouchEvent(event);
        return true;
    }
}

```

这个示例将一个 Activity 一分为二。上半部分是一个 TextView，它会提示你触摸并滑动它，而下半部分是一个内部包含若干张图片的 HorizontalScrollView。Activity 为 TextView 设置了一个 OnTouchListener，这样就可以将它接收的所有触摸事件转发给 HorizontalScrollView。

我们希望触摸事件就像发生在(从 HorizontalScrollView 的角度)HorizontalScrollView 自己视图内部一样。所以在转发事件之前，会调用 setLocation()来修改 x/y 坐标。本例中，x 坐标就是原来的坐标，y 坐标则调整到了 HorizontalScrollView 的中间。这样，当用户手指向前或者向右滚动时，就好像在 HorizontalScrollView 的中间滚动一样。然后，调用 dispatchTouchEvent()将修改后的事件交予 HorizontalScrollView 处理。

注意：

避免直接使用 onTouchEvent()方法转发触摸事件。调用 dispatchTouchEvent()可以使其像常规触摸事件一样处理目标视图的触摸事件，包括必要时的事件拦截。

2.30 创建拖放视图

2.30.1 问题

应用程序的 UI 允许用户将一些视图在屏幕上进行拖动，而且可以将它们放置到其他视图的上面。

2.30.2 解决方案

(API Level 11)

使用 Android 3.0 框架中可用的拖放 API。View 类包含了对管理屏幕上的所有拖动事

件的改进，而 `OnDragListener` 接口则可以被关联到任何拖动事件发生时需要得到通知的 `View` 上。想要开始一个拖动事件，只需要在你希望用户开始拖动的视图上简单地调用 `startDrag()` 方法即可。这个方法需要一个 `DragShadowBuilder` 实例，它用来构建视图中拖动部分的外观，另外还有两个参数将会传递给放置时的目标和监听器。

在所有参数汇总首先是一个 `ClipData` 对象，用来传递一个文本或者 `Uri` 实例。它在传递文件路径或者查询 `ContentProvider` 的场景下非常有用。第二个参数是一个对象表示拖动事件的“本地状态”。这个参数可以为任意的对象，它是一个轻量级的实例用来对拖动进行一些应用程序相关的描述。`ClipData` 只会用于拖动视图放下事件的监听器，而本地状态对于所有的监听器都是可访问的(任何时刻调用 `DragEvent` 的 `getLocalState()` 方法即可)。

在拖放过程中发生的每个特定事件都会调用 `OnDragListener.onDrag()` 方法，同时传回一个描述每个事件特征的 `DragEvent`。每个 `DragEvent` 都具有以下动作中一个：

- **ACTION_DRAG_STARTED**：当调用 `startDrag()` 开始一个新的拖动事件时会向所有视图发送该动作。
 - 位置信息可以通过 `getX()` 和 `getY()` 获得。
- **ACTION_DRAG_ENTERED**：当拖动事件进入某个视图的边界框时会向该视图发送该动作。
- **ACTION_DRAG_EXITED**：当拖动事件离开某个视图的边界框时会向该视图发送该动作。
- **ACTION_DRAG_LOCATION**：当拖动事件介于 **ACTION_DRAG_ENTERED** 和 **ACTION_DRAG_EXITED** 之间时会向某个视图发送该动作，同时会传递该视图中拖动的当前位置。
 - 位置信息可以通过 `getX()` 和 `getY()` 获得。
- **ACTION_DROP**：当拖动终止并依然位于某个视图边界中时会向该视图发送该动作。
 - 位置信息可以通过 `getX()` 和 `getY()` 获得。
 - 随同事件传入的 `ClipData` 只能在该动作时通过 `getClipData()` 方法获得。
- **ACTION_DRAG_ENDED**：当前拖动事件完成时会向所有视图发送该动作。
 - 拖动的结果可以在这里通过 `getResult()` 获得。
 - 该方法的返回值取决于放置时的目标视图是否拥有一个可用的 `OnDragListener` 可以针对 **ACTION_DROP** 事件返回 `true`。

这个方法和自定义触摸事件的工作方式类似，即监听器中返回的值决定了后续的事件传递。如果某个特殊的 `OnDragListener` 并没有对 **ACTION_DRAG_STARTED** 动作返回 `true`，那么除了 **ACTION_DRAG_ENDED** 以外，它将不会收到拖动过程中任何后续的事件。

2.30.3 实现机制

让我们看一个拖放功能的示例，首先是程序清单 2-95。这里我们创建了一个自定义的 `ImageView`，它实现了 `OnDragListener` 接口。

程序清单 2-95 自定义 View 实现 OnDragListener

```

public class DropTargetView extends ImageView implements OnDragListener {

    private boolean mDropped;

    public DropTargetView(Context context) {
        super(context);
        init();
    }

    public DropTargetView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    public DropTargetView(Context context, AttributeSet attrs, int
defaultStyle) {
        super(context, attrs, defaultStyle);
        init();
    }

    private void init() {
        //我们必须设置一个有效的监听器来接收 DragEvent
        setOnDragListener(this);
    }

    @Override
    public boolean onDrag(android.view.View v, DragEvent event) {
        PropertyValuesHolder pvhX, pvhY;
        switch (event.getAction()) {
            case DragEvent.ACTION_DRAG_STARTED:
                //通过收缩视图响应新的拖动动作
                pvhX = PropertyValuesHolder.ofFloat("scaleX", 0.5f);
                pvhY = PropertyValuesHolder.ofFloat("scaleY", 0.5f);
                ObjectAnimator.ofPropertyValuesHolder(this, pvhX, pvhY).start();
                //新的拖动行为时会清空当前的放置图片
                setImageDrawable(null);
                mDropped = false;
                break;
            case DragEvent.ACTION_DRAG_ENDED:
                //拖动结束时, 如果没有找到放置目标, 视图则恢复到原来的大小
                if (!mDropped) {
                    pvhX = PropertyValuesHolder.ofFloat("scaleX", 1f);
                    pvhY = PropertyValuesHolder.ofFloat("scaleY", 1f);
                    ObjectAnimator.ofPropertyValuesHolder(this, pvhX, pvhY).start();
                    mDropped = false;
                }
                break;
            case DragEvent.ACTION_DRAG_ENTERED:

```

```

        //拖动进入到边界区域时,视图会稍微放大一下
        pvhX = PropertyValuesHolder.ofFloat("scaleX", 0.75f);
        pvhY = PropertyValuesHolder.ofFloat("scaleY", 0.75f);
        ObjectAnimator.ofPropertyValuesHolder(this, pvhX, pvhY).start();
        break;
    case DragEvent.ACTION_DRAG_EXITED:
        //拖动离开视图时,会恢复到之前的大小
        pvhX = PropertyValuesHolder.ofFloat("scaleX", 0.5f);
        pvhY = PropertyValuesHolder.ofFloat("scaleY", 0.5f);
        ObjectAnimator.ofPropertyValuesHolder(this, pvhX, pvhY).start();
        break;
    case DragEvent.ACTION_DROP:
        //拖动后放置时会有一段简短的帧动画并将视图的图片
        //设置为拖动事件传入的 drawable 图片

        //这个动画会收缩一下视图,然后再还原
        Keyframe frame0 = Keyframe.ofFloat(0f, 0.75f);
        Keyframe frame1 = Keyframe.ofFloat(0.5f, 0f);
        Keyframe frame2 = Keyframe.ofFloat(1f, 0.75f);
        pvhX = PropertyValuesHolder.ofKeyframe("scaleX", frame0, frame1,
            frame2);
        pvhY = PropertyValuesHolder.ofKeyframe("scaleY", frame0, frame1,
            frame2);
        ObjectAnimator.ofPropertyValuesHolder(this, pvhX, pvhY).start();
        //DragEvent 中传递的 Object 设置我们的图片
        setImageDrawable((Drawable) event.getLocalState());
        //我们设置放置标识让 ENDED 动画不再运行
        mDropped = true;
        break;
    default:
        //忽略我们不感兴趣的事件
        return false;
    }
    //处理所有感兴趣的事件
    return true;
}
}

```

这个 `ImageView` 用来监控新产生的拖动事件并自己运行相应的动画。每次新的拖动行为出现时, `ACTION_DRAG_STARTED` 事件就会被发送到这里, 这个 `ImageView` 就会自己缩小 50%。这对用户是一个非常好的引导, 可以告诉用户他刚刚选择的小图片可以拖动到哪里。这里我们还确保这个监听器会对这个事件返回 `true`, 这样就可以接收拖动过程中的其他事件了。

如果用户将小图片拖动到这个 `ImageView` 上, 就会触发 `ACTION_DRAG_ENTERED`, 这时 `ImageView` 会稍微放大一下, 表示该 `ImageView` 可以接收的小图片放置行为。当小图片离开时会触发 `ACTION_DRAG_EXITED` 事件, 这时 `ImageView` 会恢复到刚进入“拖放模式”时的大小。如果用户在该 `ImageView` 的上方松手, 会触发 `ACTION_DROP` 事件, 同时会运行一段特殊的动画表示放置动作已经收到。这时我们会读取事件中的本地状态变

量，如果是一个 Drawable，就把它设置为 ImageView 的内容。

ACTION_DRAG_ENDED 会通知该 ImageView 恢复到之前的大小，因为此时已经不再处于“拖动模式”了。但是，如果这个 ImageView 就是放置的目标，而希望保持它的大小，因此这种情况下会忽略掉这个事件。

程序清单 2-96 和 2-97 显示了一个示例 Activity，该 Activity 允许用户通过长按一张图片，然后可以拖动该图片到我们自定义的放置目标上。

程序清单 2-96 res/layout/main.xml

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

    <!--顶部行为可拖拽条目-->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <ImageView
            android:id="@+id/image1"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:src="@drawable/ic_send" />
        <ImageView
            android:id="@+id/image2"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:src="@drawable/ic_share" />
        <ImageView
            android:id="@+id/image3"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:src="@drawable/ic_favorite" />
    </LinearLayout>

    <!--底部一行为可放置的目标 -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:orientation="horizontal" >
        <com.examples.dragtouch.DropTargetView
            android:id="@+id/drag_target1"
            android:layout_width="0dp"
            android:layout_height="100dp"
```

```

        android:layout_weight="1"
        android:background="#A00" />
<com.examples.dragtouch.DropTargetView
    android:id="@+id/drag_target2"
    android:layout_width="0dp"
    android:layout_height="100dp"
    android:layout_weight="1"
    android:background="#0A0" />
<com.examples.dragtouch.DropTargetView
    android:id="@+id/drag_target3"
    android:layout_width="0dp"
    android:layout_height="100dp"
    android:layout_weight="1"
    android:background="#00A" />
</LinearLayout>

</RelativeLayout>

```

程序清单 2-97 转发触摸事件的 Activity

```

public class DragTouchActivity extends Activity implements OnLongClickListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //为每个 ImageView 关联长按监听器
        findViewById(R.id.image1).setOnLongClickListener(this);
        findViewById(R.id.image2).setOnLongClickListener(this);
        findViewById(R.id.image3).setOnLongClickListener(this);
    }

    @Override
    public boolean onLongClick(View v) {
        DragShadowBuilder shadowBuilder = new DragShadowBuilder(v);
        //开始一个拖动, 将 View 的图片作为本地状态传递出去
        v.startDrag(null, shadowBuilder, ((ImageView) v).getDrawable(), 0);

        return true;
    }
}

```

这个示例会在屏幕的顶部显示一行三张图片, 同时在屏幕的底部显示 3 个我们自定义的目标视图。每个图片都设置了一个长按监听器, 长按动作会通过 `startDrag()` 触发一个新的拖动事件。拖动事件初始化时传入的 `DragShadowBuilder` 是框架提供的默认实现。下一节会查看如何自定义 `DragShadowBuilder`, 但本节中视图在拖动时会创建一个透明的副本并显示在触摸点的下方。

我们还通过 `getDrawable()` 得到了用户选择的视图的图片内容并把它作为拖动的本地状态传递出去, 自定义的放置目标会使用它来设置图片。这样就会产生视图已经放置到了目

标上的效果。参见图 2-25 查看加载时的效果、拖动操作过程中的效果以及图片被放置到了某个放置目标后的效果。

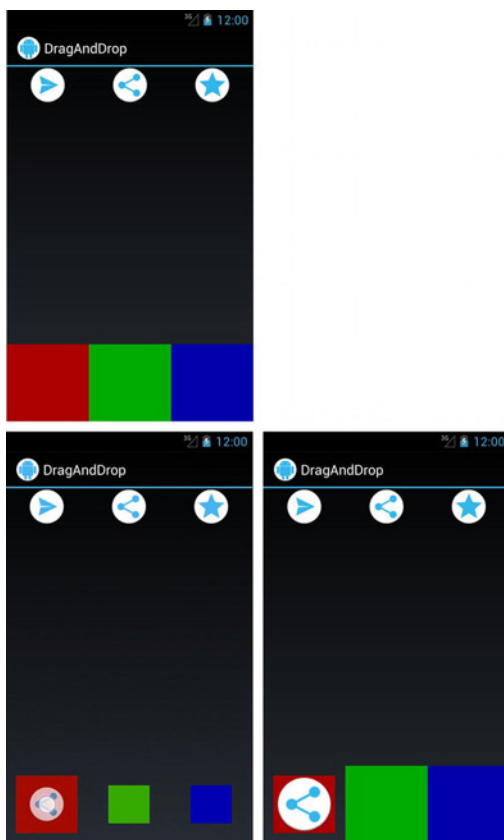


图 2-25 Drag 示例拖动前的效果(上边)，用户拖动并游走于放置目标上方时的效果(左边)以及视图放置后的效果(右边)

1. 自定义 DragShadowBuilder

DragShadowBuilder 的默认实现非常方便，但有可能并不是你的应用程序所需要的。让我们看一下程序清单 2-98，它实现了一个自定义的 DragShadowBuilder。

程序清单 2-98 自定义 DragShadowBuilder

```
public class DrawableDragShadowBuilder extends DragShadowBuilder {
    private Drawable mDrawable;

    public DrawableDragShadowBuilder(View view, Drawable drawable) {
        super(view);
        //设置 Drawable 并使用一个绿色的过滤器
        mDrawable = drawable;
        mDrawable.setColorFilter(
            new PorterDuffColorFilter(Color.GREEN, PorterDuff.Mode.MULTIPLY));
    }
}
```

```

@Override
public void onProvideShadowMetrics(Point shadowSize, Point touchPoint) {
    //填充大小
    shadowSize.x = mDrawable.getIntrinsicWidth();
    shadowSize.y = mDrawable.getIntrinsicHeight();
    //设置阴影相对于触摸点的位置
    //这里阴影位于手指下方的中心
    touchPoint.x = mDrawable.getIntrinsicWidth() / 2;
    touchPoint.y = mDrawable.getIntrinsicHeight() / 2;

    mDrawable.setBounds(new Rect(0, 0, shadowSize.x, shadowSize.y));
}

@Override
public void onDrawShadow(Canvas canvas) {
    //在提供的 canvas 上绘制阴影视图
    mDrawable.draw(canvas);
}
}

```

自定义的实现会使用一个单独的 `Drawable` 参数，阴影的显示会使用该图片而不是使用源视图的副本。另外，我们还对该图片使用了绿色的 `ColorFilter` 来增加一些效果。`DragShadowBuilder` 是一个非常容易扩展的类。只需要有效地覆写两个主要的方法。

第一个方法是 `onProvideShadowMetrics()`，它会在 `DragShadowBuilder` 初始化时调用一次并使用两个 `Point` 对象填充 `DragShadowBuilder` 的内容。首先会填充阴影使用的图片的大小，即会将期望的宽度设置为 `x` 值，高度设置为 `y` 值。本例中会将该大小设置为图片本来的宽度和高度。另外需要填充的是阴影期望触摸的位置。这里会定义阴影图片相对于用户手指的位置，例如将 `x` 和 `y` 都设置为 0 时，手指会位于图片的左上角。在我们的示例中，我们设置到了图片的中心点，因此手指会位于图片中心上方。

第二个方法是 `onDrawShadow()`，它会被重复调用来渲染阴影图片。这个方法中传入的 `Canvas` 是由框架根据 `onProvideShadowMetrics()` 中包含的信息创建的。这里你可以像其他自定义视图一样进行各种自定义绘制。我们的示例只是简单地告诉 `Drawable` 在 `Canvas` 中绘制它自己。

2.31 自定义过渡动画

2.31.1 问题

应用程序需要自定义 `Activity` 切换或者 `fragment` 切换时的过渡动画。

2.31.2 解决方案

(API Level 5)

想要修改 `Activity` 间的过渡动画，可以使用 `overridePendingTransition()` API 进行某次切

换时的动画，或者在应用程序的主题中声明自定义动画值来进行全局的设置。想要修改 Fragment 的过渡动画，可以使用 onCreateAnimation()或 onCreateAnimator() API 方法。

2.31.3 实现机制

1. Activity

自定义 Activity 切换时的过渡动画时，可以考虑四种动画：打开一个新 Activity 时的进入动画和退出动画，当前 Activity 关闭时的进入动画和退出动画。

每种动画都会应用到过渡动画中所涉及的两个 Activity 中的一个上。例如，当打开一个新 Activity 时，当前 Activity 将会运行“打开退出”动画而新 Activity 会运行“打开进入”动画。由于这些动画都是同时运行的，因此动画间应该是互补的，否则会看起来不太协调。程序清单 2-99 到 2-102 演示了这四种动画。

程序清单 2-99 res/anim/activity_open_enter.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="90" android:toDegrees="0"
        android:pivotX="0%" android:pivotY="0%"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
    <alpha
        android:fromAlpha="0.0" android:toAlpha="1.0"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
</set>
```

程序清单 2-100 res/anim/activity_open_exit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="0" android:toDegrees="-90"
        android:pivotX="0%" android:pivotY="0%"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
    <alpha
        android:fromAlpha="1.0" android:toAlpha="0.0"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
</set>
```

程序清单 2-101 res/anim/activity_close_enter.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:fromDegrees="-90" android:toDegrees="0"
        android:pivotX="0%p" android:pivotY="0%p"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
    <alpha
        android:fromAlpha="0.0" android:toAlpha="1.0"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
</set>
```

程序清单 2-102 res/anim/activity_close_exit.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <rotate
        android:fromDegrees="0" android:toDegrees="90"
        android:pivotX="0%p" android:pivotY="0%p"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
    <alpha
        android:fromAlpha="1.0" android:toAlpha="0.0"
        android:fillEnabled="true"
        android:fillBefore="true" android:fillAfter="true"
        android:duration="500" />
</set>
```

我们创建了 2 个“打开”动画，即旧的 Activity 顺时针旋转消失、新 Activity 顺时针旋转进入。补足的“关闭”动画会将当前 Activity 逆时针旋转退出、之前的 Activity 逆时针旋转进入。

每个动画还有一个渐出或渐入的效果，这样过渡动画看起来会更加流畅。想要在特定时刻应用这些自定义动画，可以在 `startActivity()` 或者 `finish()` 后立刻调用 `overridePendingTransition()` 方法，如下：

```
//使用自定义过渡动画启动一个新的 Activity
Intent intent = new Intent(...);
startActivity(intent);
overridePendingTransition(R.anim.activity_open_enter,
    R.anim.activity_open_exit);

//使用自定义过渡动画关闭当前 Activity
finish();
```

```
overridePendingTransition(R.anim.activity_close_enter,
    R.anim.activity_close_exit);
```

这种方式在只希望在某些场合使用自定义动画的情况下非常有用。但如果希望在应用程序中自定义每个 Activity 的过渡动画，到处调用这个方法可能会有点麻烦。反之最好在应用程序的主题中使用自定义的动画。程序清单 2-103 演示了一个自定义的主题可以全局使用这些过渡动画。

程序清单 2-103 res/values/styles.xml

```
<resources>
    <style name="AppTheme" parent="android:Theme.Holo.Light">
        <item name="android:windowAnimationStyle">@style/ActivityAnimation</item>
    </style>

    <style name="ActivityAnimation" parent="@android:style/Animation.Activity">
        <item
            name="android:activityOpenEnterAnimation">@anim/activity_open_enter
        </item>
        <item
            name="android:activityOpenExitAnimation">@anim/activity_open_exit
        </item>
        <item
            name="android:activityCloseEnterAnimation">@anim/activity_close_enter
        </item>
        <item
            name="android:activityCloseExitAnimation">@anim/activity_close_exit
        </item>
    </style>
</resources>
```

通过提供一个主题的自定义属性 `android:windowAnimationStyle` 值，我们可以自定义这些过渡动画。引用框架的父样式也很重要，这是因为这四种动画并不是该样式中的唯一内容，否则的话可能会无意中去除现有的一些窗口动画。

2. 支持 Fragment

自定义 Fragment 的过渡动画会有些不同，这取决于你是否使用了支持库。这是因为原生的 Fragment 使用了新的 Animator 对象，该对象在支持库的 Fragment 版本中是被不支持的。

使用支持库时，可以通过调用 `setCustomAnimations()` 覆写单个 `FragmentManager` 的过渡动画。该方法两个参数的版本可以设置添加/替换/移除动作时的动画效果，但在界面栈回退时不能设置相应的动画。

该方法四个参数的版本则可以为界面栈的回退添加自定义的动画。还是使用之前示例中一样的 `Animation` 对象，下面的代码显示了如何将这些动画添加到 `FragmentManager` 中。

```
FragmentManager ft = getSupportFragmentManager().beginTransaction();
```

```
//首先必须调用该方法
ft.setCustomAnimations(R.anim.activity_open_enter, R.anim.activity_open_exit,
    R.anim.activity_close_enter, R.anim.activity_close_exit);
ft.replace(R.id.container_fragment, fragment);
ft.addToBackStack(null);
ft.commit();
```

重点:

setCustomAnimations()必须在 add()、replace()和其他动作方法之前调用,否则动画将不会运行。最好是在每个事务代码块开始时就调用该方法。

如果希望对某个 Fragment 一直使用同样的动画,可能需要覆写 Fragment 中的 onCreateAnimation()方法。程序清单 2-104 显示了使用这种方式定义的 Fragment 动画。

程序清单 2-104 使用自定义动画的 Fragment

```
public class SupportFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        TextView tv = new TextView(getActivity());
        tv.setText("Fragment");
        tv.setBackgroundColor(Color.RED);
        return tv;
    }

    @Override
    public Animation onCreateAnimation(int transit, boolean enter, int
        nextAnim) {
        switch (transit) {
            case FragmentTransaction.TRANSIT_FRAGMENT_FADE:
                if (enter) {
                    return AnimationUtils.loadAnimation(getActivity(),
                        android.R.anim.fade_in);
                } else {
                    return AnimationUtils.loadAnimation(getActivity(),
                        android.R.anim.fade_out);
                }
            case FragmentTransaction.TRANSIT_FRAGMENT_CLOSE:
                if (enter) {
                    return AnimationUtils.loadAnimation(getActivity(),
                        R.anim.activity_close_enter);
                } else {
                    return AnimationUtils.loadAnimation(getActivity(),
                        R.anim.activity_close_exit);
                }
            case FragmentTransaction.TRANSIT_FRAGMENT_OPEN:
            default:
                if (enter) {
```

```

        return AnimationUtils.loadAnimation(getActivity(),
            R.anim.activity_open_enter);
    } else {
        return AnimationUtils.loadAnimation(getActivity(),
            R.anim.activity_open_exit);
    }
}
}
}
}

```

Fragment 的动画行为和 FragmentTransaction 的设置有很大的关系。有很多的事务值可以通过 setTransition() 方法关联到事务上。如果没有调用 setTransition(), Fragment 就无法知道打开动画和关闭动画的区别, 因此我们唯一知道的就是运行进入动画还是退出动画。

想要获得之前通过 setCustomAnimations() 实现的相同效果, 需要将事务的过渡值设为 TRANSIT_FRAGMENT_OPEN。这时会使用这个过渡值调用初始的事务, 但同时会通过 TRANSIT_FRAGMENT_CLOSE 调用界面栈回退动作, 这样就允许 Fragment 提供不同的动画。下面的代码片段演示了用这种方式构造一个事务。

```

FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
// 设置过渡值来触发相应的动画
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
ft.replace(R.id.container_fragment, fragment);
ft.addToBackStack(null);
ft.commit();

```

Fragment 还有第三种状态, 这在 Activity 中是没有的, 它是通过 TRANSIT_FRAGMENT_FADE 过渡值定义的。这个动画是在过渡行为不再是变化的一部分时出现的, 例如添加或替换, 但在 Fragment 隐藏或显示时不会出现。在这个示例中, 我们使用了标准的系统渐变动画来诠释这种情形。

3. 本地 Fragment

如果应用程序的目标版本是 API Level 11 或之后版本, 则不必使用支持库中的 fragment, 而且这种情况下的自定义动画代码会稍微有些不同。本地 Fragment 实现使用了相对较新的 Animator 对象来创建过渡动画而非旧的 Animation 对象。

需要对代码做一些修改, 首先, 需要使用 Animator 来定义所有的 XML 动画。参见程序清单 2-105 到程序清单 2-108。

程序清单 2-105 res/animator/fragment_exit.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <objectAnimator
        android:valueFrom="0" android:valueTo="-90"
        android:valueType="floatType"
        android:propertyName="rotation"
        android:duration="500"/>

```

```

        <objectAnimator
            android:valueFrom="1.0" android:valueTo="0.0"
            android:valueType="floatType"
            android:propertyName="alpha"
            android:duration="500"/>
    </set>

```

程序清单 2-106 res/animator/fragment_enter.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <objectAnimator
        android:valueFrom="90" android:valueTo="0"
        android:valueType="floatType"
        android:propertyName="rotation"
        android:duration="500"/>
    <objectAnimator
        android:valueFrom="0.0" android:valueTo="1.0"
        android:valueType="floatType"
        android:propertyName="alpha"
        android:duration="500"/>
</set>

```

程序清单 2-107 res/animator/fragment_pop_exit.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <objectAnimator
        android:valueFrom="0" android:valueTo="90"
        android:valueType="floatType"
        android:propertyName="rotation"
        android:duration="500"/>
    <objectAnimator
        android:valueFrom="1.0" android:valueTo="0.0"
        android:valueType="floatType"
        android:propertyName="alpha"
        android:duration="500"/>
</set>

```

程序清单 2-108 res/animator/fragment_pop_enter.xml

```

<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <objectAnimator
        android:valueFrom="-90" android:valueTo="0"
        android:valueType="floatType"
        android:propertyName="rotation"
        android:duration="500"/>
    <objectAnimator
        android:valueFrom="0.0" android:valueTo="1.0"
        android:valueType="floatType"

```

```

        android:propertyName="alpha"
        android:duration="500"/>
</set>

```

除了语法的细微区别,这些动画几乎和之前创建的动画一模一样。其他仅有的差别是,这些动画被设置为围绕在视图的中心(默认行为),而不是左上角。

和以前一样,可以通过 `setCustomAnimations()` 直接设置某个 `FragmentTransaction` 单独的过渡动画;但是,新的版本使用了我们的 `Animator` 实例。下面的代码片段使用新 API 实现了这一过程:

```

FragmentTransaction ft = getFragmentManager().beginTransaction();
// 首先必须调用该方法
ft.setCustomAnimations(R.animator.fragment_enter, R.animator.fragment_exit,
    R.animator.fragment_pop_enter, R.animator.fragment_pop_exit);
ft.replace(R.id.container_fragment, fragment);
ft.addToBackStack(null);
ft.commit();

```

如果想要对某一特定 `Fragment` 总是使用相同的过渡动画,可以像以前一样自定义 `Fragment`。但本地 `Fragment` 没有 `onCreateAnimation()` 方法而是使用了 `onCreateAnimator()`。查看程序清单 2-109,它使用新的 API 重新定义了我们创建的 `Fragment`。

程序清单 2-109 自定义过渡动画的本地 `Fragment`

```

public class NativeFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        TextView tv = new TextView(getActivity());
        tv.setText("Fragment");
        tv.setBackgroundColor(Color.BLUE);
        return tv;
    }

    @Override
    public Animator onCreateAnimator(int transit, boolean enter, int nextAnim) {
        switch (transit) {
            case FragmentTransaction.TRANSIT_FRAGMENT_FADE:
                if (enter) {
                    return AnimatorInflater.loadAnimator(getActivity(),
                        android.R.animator.fade_in);
                } else {
                    return AnimatorInflater.loadAnimator(getActivity(),
                        android.R.animator.fade_out);
                }
            case FragmentTransaction.TRANSIT_FRAGMENT_CLOSE:
                if (enter) {
                    return AnimatorInflater.loadAnimator(getActivity(),

```

```

        R.animator.fragment_pop_enter);
    } else {
        return AnimatorInflater.loadAnimator(getActivity(),
            R.animator.fragment_pop_exit);
    }
case FragmentTransaction.TRANSIT_FRAGMENT_OPEN:
default:
    if (enter) {
        return AnimatorInflater.loadAnimator(getActivity(),
            R.animator.fragment_enter);
    } else {
        return AnimatorInflater.loadAnimator(getActivity(),
            R.animator.fragment_exit);
    }
}
}
}
}

```

同样，会像支持库示例一样检查同样的过渡值，我们只是返回 **Animator** 实例。下面同样的代码片段可以使用过渡值开始一个事务：

```

FragmentTransaction ft = getFragmentManager().beginTransaction();
//设置过渡值来触发相应的动画
ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
ft.replace(R.id.container_fragment, fragment);
ft.addToBackStack(null);
ft.commit();

```

将这些自定义动画全局地应用到整个应用程序的最终方式就是将这些动画关联到应用程序的主题上。程序清单 2-110 显示了使用我们 **Fragment** 动画的自定义主题。

程序清单 2-110 res/values/styles.xml

```

<resources>
    <style name="AppTheme" parent="android:Theme.Holo.Light">
        <item name="android:windowAnimationStyle">@style/FragmentAnimation</item>
    </style>

    <style name="FragmentAnimation" parent="@android:style/Animation.Activity">
        <item
            name="android:fragmentOpenEnterAnimation">@animator/fragment_enter
        </item>
        <item
            name="android:fragmentOpenExitAnimation">@animator/fragment_exit
        </item>
        <item
            name="android:fragmentCloseEnterAnimation">@animator/fragment_pop_enter
        </item>
        <item
            name="android:fragmentCloseExitAnimation">@animator/fragment_pop_exit
        </item>
    </style>
</resources>

```

```

        </item>
        <item
            name="android:fragmentFadeEnterAnimation">@android:animator/fade_in
        </item>
        <item
            name="android:fragmentFadeExitAnimation">@android:animator/fade_out
        </item>
    </style>
</resources>

```

正如你所看到的，一个主题默认的 `Fragment` 动画属性就是相同 `windowAnimationStyle` 属性的一部分。而且，我们在自定义时要保证继承自同样的父样式，否则会移除其他的一些系统默认效果(例如 `Activity` 过渡动画)。必须依然要在 `FragmentTransaction` 请求相应的过渡类型从而触发相应的动画。

如果想要在主题中同时自定义 `Activity` 和 `Fragment` 的过渡动画，可以将它们一起放到一个相同的自定义样式中(参见程序清单 2-111)。

程序清单 2-111 res/values/styles.xml

```

<resources>
    <style name="AppTheme" parent="android:Theme.Holo.Light">
        <item name="android:windowAnimationStyle">@style/TransitionAnimation</item>
    </style>

    <style name="TransitionAnimation" parent="@android:style/Animation.Activity">
        <item
            name="android:activityOpenEnterAnimation">@anim/activity_open_enter
        </item>
        <item
            name="android:activityOpenExitAnimation">@anim/activity_open_exit
        </item>
        <item
            name="android:activityCloseEnterAnimation">@anim/activity_close_enter
        </item>
        <item
            name="android:activityCloseExitAnimation">@anim/activity_close_exit
        </item>
        <item
            name="android:fragmentOpenEnterAnimation">@animator/fragment_enter
        </item>
        <item
            name="android:fragmentOpenExitAnimation">@animator/fragment_exit
        </item>
        <item
            name="android:fragmentCloseEnterAnimation">@animator/fragment_pop_enter
        </item>
        <item
            name="android:fragmentCloseExitAnimation">@animator/fragment_pop_exit
        </item>
    </style>

```

```

        <item
            name="android:fragmentFadeEnterAnimation">@android:animator/fade_in
        </item>
        <item
            name="android:fragmentFadeExitAnimation">@android:animator/fade_out
        </item>
    </style>
</resources>

```

警告:

对主题添加的 Fragment 过渡动画只会作用于本地 Fragment。支持库中的 Fragment 因为早期的平台版本并没有这些属性所以也找不到这些属性。

2.32 创建视图变换

2.32.1 问题

应用程序需要动态变换视图的外观，从而为视图添加一些视觉效果，例如视角变换效果。

2.32.2 解决方案

(API Level 1)

ViewGroup 中的静态变换 API 提供了应用视觉效果的简单方法，例如旋转、缩放、透明度变化，而且不必依靠动画。使用它也很容易使用父视图的 context 来应用变换，比如根据位置的变化而缩放。

在初始化过程中调用 setStaticTranformationsEnabled(true)，可以启用任何 ViewGroup 的静态变换。启用此功能后，Framework 会定期对每个子视图的 getChildStaticTransformation() 从而允许应用程序设置变换。

2.32.3 实现机制

首先看一个示例，在该示例中变换被应用一次而且不会改变(参见程序清单 2-112)。

程序清单 2-112 使用静态变换自定义布局

```

public class PerspectiveLayout extends LinearLayout {

    public PerspectiveLayout(Context context) {
        super(context);
        init();
    }

    public PerspectiveLayout(Context context, AttributeSet attrs) {

```

```

        super(context, attrs);
        init();
    }

    public PerspectiveLayout(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        init();
    }

    private void init() {
        //启用静态变换, 这样对于每个子视图都会调用 getChildStaticTransformation()。
        setStaticTransformationsEnabled(true);
    }

    @Override
    protected boolean getChildStaticTransformation(View child, Transformation t) {
        // 清除所有现有的变换
        t.clear();

        if (getOrientation() == HORIZONTAL) {
            //根据到左边缘的距离对子视图进行缩放
            float delta = 1.0f - ((float) child.getLeft() / getWidth());

            t.getMatrix().setScale(delta, delta, child.getWidth() / 2,
                                   child.getHeight() / 2);
        } else {
            //根据到顶端边缘的距离对子视图进行缩放
            float delta = 1.0f - ((float) child.getTop() / getHeight());

            t.getMatrix().setScale(delta, delta, child.getWidth() / 2,
                                   child.getHeight() / 2);
            //同样也根据它的位置应用颜色淡出效果。
            t.setAlpha(delta);
        }
        return true;
    }
}

```

这个示例介绍了一个自定义的 `LinearLayout`, 它根据子视图到父视图起始边缘的距离, 对每个子视图做了缩放变换。`getChildStaticTransformation()` 中的代码通过子视图到父视图左边缘或顶端边缘的距离与父视图尺寸的比值计算得出应该使用的缩放比例。当变换设定好之后, 这个方法的返回值会通知 **Android** 框架。任何情况下, 只要应用程序中设置了一个自定义的变换, 这个方法就必须返回 “true”, 以确保它被关联到了视图上。

大多数的视觉效果比如旋转或缩放, 都实际上被应用于 `Transformation` 的 `Matrix` 上。在我们的示例中通过调用 `getMatrix().setScale()` 来调整每个子视图的缩放, 同时传入缩放比例和轴心点。轴心点就是缩放发生的位置, 我们将这个位置设置在视图的中心点, 这样缩放的结果就会居中显示。

如果布局是垂直方向的, 我们同样会根据相同的距离值为子视图应用透明渐变效果,

只需要直接使用 Transformation 的 `setAlpha()` 方法即可。程序清单 2-113 就是使用这个视图的示例布局。

程序清单 2-113 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <!-- 水平方向自定义布局 -->
    <com.examples.statictransforms.PerspectiveLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
    </com.examples.statictransforms.PerspectiveLayout>
    <!--垂直方向自定义布局 -->
    <com.examples.statictransforms.PerspectiveLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_launcher" />
    </com.examples.statictransforms.PerspectiveLayout>
```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_launcher" />
    </com.examples.statictransforms.PerspectiveLayout>
</LinearLayout>

```

图 2-26 显示了示例变换的结果。

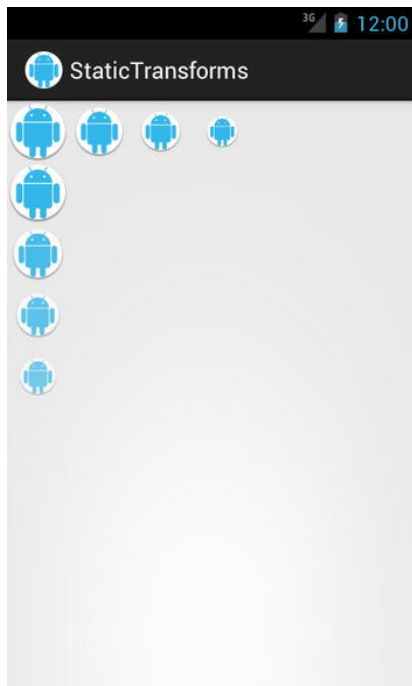


图 2-26 水平视角和垂直视角布局

在水平布局中，越往右的视图使用的缩放比例越小。同样，垂直方向视图的缩放比例也是越往下越小。另外，垂直方向的视图由于添加了透明度变化会出现逐渐淡出的效果。

现在让我们看一个提供了更为动态变化效果的示例。程序清单 2-114 展示了一个封装在 `HorizontalScrollView` 中的自定义布局。当子视图滚动时，这个布局使用静态变换来缩放子视图。在屏幕中心的视图大小总是正常的，越靠近边缘视图就越小。这样就会出现下面的效果：在滚动的过程中，视图首先会逐渐靠近然后会逐渐远离。

程序清单 2-114 自定义视角滚动内容

```

public class PerspectiveScrollContentView extends LinearLayout {

    /* 每个子视图的缩放比例都是可调节的*/
    private static final float SCALE_FACTOR = 0.7f;
    /*
     * 变换的轴心点。(0,0)是左上，(1,1)是右下。当前设置的是底部中间(0.5, 1)。
     */
    private static final float ANCHOR_X = 0.5f;

```

```

private static final float ANCHOR_Y = 1.0f;

public PerspectiveScrollContentView(Context context) {
    super(context);
    init();
}

public PerspectiveScrollContentView(Context context, AttributeSet
attrs) {
    super(context, attrs);
    init();
}

public PerspectiveScrollContentView(Context context, AttributeSet
attrs,
    int defStyle) {
    super(context, attrs, defStyle);
    init();
}

private void init() {
    //启动静态变换, 这样对于每个子视图 getChildStaticTransformation()
    //都会被调用。
    setStaticTransformationsEnabled(true);
}

/*
 * 工具方法, 用于计算屏幕坐标系内所有视图的当前位置。
 */
private int getViewCenter(View view) {
    int[] childCoords = new int[2];
    view.getLocationOnScreen(childCoords);
    int childCenter = childCoords[0] + (view.getWidth() / 2);

    return childCenter;
}

@Override
protected boolean getChildStaticTransformation(View child,
Transformation t) {
    HorizontalScrollView scrollView = null;
    if (getParent() instanceof HorizontalScrollView) {
        scrollView = (HorizontalScrollView) getParent();
    }
    if (scrollView == null) {
        return false;
    }

    int childCenter = getViewCenter(child);
    int viewCenter = getViewCenter(scrollView);

```

```

        //计算子视图和父容器中心之间的距离。这会决定设置的缩放比例
        float delta = Math.min(1.0f, Math.abs(childCenter - viewCenter)
            / (float) viewCenter);
        //设置最小缩放比例为 0.4
        float scale = Math.max(0.4f, 1.0f - (SCALE_FACTOR * delta));
        float xTrans = child.getWidth() * ANCHOR_X;
        float yTrans = child.getHeight() * ANCHOR_Y;

        //清除现有的所有变换
        t.clear();
        //为子视图设置变换
        t.getMatrix().setScale(scale, scale, xTrans, yTrans);

        return true;
    }
}

```

在这个示例中，自定义布局会根据每个子视图相对于父视图 `HorizontalScrollView` 中心位置的距离，为每个子视图计算变换。当用户滚动时每个子视图的变换需要重新计算从而实现视图移动时子视图的动态放大和缩小。这个示例将变换轴心点设置在每个子视图的底部中间位置，这样会创造出这样的效果：每个子视图会垂直放大，而且保持水平居中。程序清单 2-115 的 Activity 示例将这个布局付诸实践。

程序清单 2-115 使用 `PerspectiveScrollView` 的 Activity

```

public class ScrollActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        HorizontalScrollView parentView = new HorizontalScrollView(this);
        PerspectiveScrollView contentView =
            new PerspectiveScrollView(this);

        //对此视图禁用硬件加速，因为动态调整每个子视图的变换当前无法通过硬件实现。
        //也可以通过在 manifest 中添加 android:hardwareAccelerated="false"
        //禁用整个 Activity 或应用程序的硬件加速。但出于性能的考虑，最好尽可能少地
        //禁用硬件加速。
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            contentView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
        }

        //向滚动条中添加几张图片
        for (int i = 0; i < 20; i++) {
            ImageView iv = new ImageView(this);
            iv.setImageResource(R.drawable.ic_launcher);
            contentView.addView(iv);
        }
    }
}

```

```

        //添加要显示的视图
        parentView.addView(contentView);
        setContentView(parentView);
    }
}

```

在这个示例中创建了一个滚动视图并且关联了一个自定义的包含有若干个图片的 `PerspectiveScrollContentView`。这里的代码不需要太多关注，但有个非常重要的地方需要提一下。虽然一般情况下静态变换都会被支持，但视图刷新过程中动态更新变换效果在当前的 SDK 版本中是不能使用硬件加速的。因此，如果你的应用程序的目标 SDK 为 11 或以上版本，或者已经在某种程度上启用了硬件加速，这时需要对这个视图禁用硬件加速。

可以在 `manifest` 文件的 `<activity>` 或 `<application>` 标签中添加 `android:hardwareAccelerated="false"` 属性对硬件加速进行全局的设置，但是，我们也可以通过调用 `setLayerType()` 方法并设置 `LAYER_TYPE_SOFTWARE` 在 Java 代码中对这个自定义视图进行单独设置。如果你的应用程序的目标 SDK 版本低于此版本，即使是较新的设备，默认情况下硬件加速是关闭的，出于兼容性的问题考虑，这些代码也许是不必要的。

2.33 视图之间滑动

2.33.1 问题

需要在应用程序的 UI 中通过手势滑动来实现页面切换，例如视图之间或 `Fragment` 之间的切换。

2.33.2 解决方案

(API Level 4)

实现 `ViewPager` 控件以提供手势滑动时页面切换的功能。`ViewPager` 是当前只可以通过支持库使用的控件。无论在那个级别的 android 平台中，原生的 SDK 都不包含 `ViewPager`。不过，所有目标版本为 API Level 4 或以上的应用程序都可以通过支持库来使用它。

`ViewPager` 是 `AdapterView` 模式修改后的实现，`ListView` 和 `GridView` 也使用了框架的这种模式。`ViewPager` 需要一个继承自 `PagerAdapter` 的子类适配器实现，从概念上讲，该适配器与 `BaseAdapter` 和 `ListAdapter` 中使用的模式非常类似。`ViewPager` 本身并不能实现分页控件的回收，但它每时每刻都提供了回调方法来进行条目创建和销毁。所以在特定的时间内，内存中运行的内容视图的数量是固定的。

2.33.3 实现机制

使用 `ViewPager` 最大的工作就是 `PagerAdapter` 的实现。让我们开始一个简单的示例，参见程序清单 2-116，它实现了很多图像的分页显示。

程序清单 2-116 自定义图像 PagerAdapter

```

public class ImagePagerAdapter extends PagerAdapter {
    private Context mContext;

    private static final int[] IMAGES = {
        android.R.drawable.ic_menu_camera,
        android.R.drawable.ic_menu_add,
        android.R.drawable.ic_menu_delete,
        android.R.drawable.ic_menu_share,
        android.R.drawable.ic_menu_edit
    };

    private static final int[] COLORS = {
        Color.RED,
        Color.BLUE,
        Color.GREEN,
        Color.GRAY,
        Color.MAGENTA
    };

    public ImagePagerAdapter(Context context) {
        super();
        mContext = context;
    }

    /*
     * 页面的总数
     */
    @Override
    public int getCount() {
        return IMAGES.length;
    }

    /*
     * 如果要在 ViewPager 内一次显示超过一页的内容，那么需要重写该方法。
     */
    @Override
    public float getPageWidth(int position) {
        return 1f;
    }

    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        // 创建一个新的 ImageView 并把它添加到提供的容器中。
        ImageView iv = new ImageView(mContext);
        // 用 IMAGES 和 COLORS 中当前位置的内容设置新的 ImageView
        iv.setImageResource(IMAGES[position]);
        iv.setBackgroundColor(COLORS[position]);
    }
}

```

```

        //这里你必须自己添加视图, Android 框架是不会为你添加的。
        container.addView(iv);
        //将这个视图作为这个位置的键对象返回。
        return iv;
    }

    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
        //从容器中删除视图
        container.removeView((View) object);
    }

    @Override
    public boolean isViewFromObject(View view, Object object) {
        //检查从 instantiateItem() 返回的对象与添加到容器相应位置的视图是否是同一
        //个对象。我们的示例在这两个地方使用的是同一个对象。
        return (view == object);
    }
}

```

在这个示例中, 我们实现了一个 `PagerAdapter`, 它提供了很多的图片供用户翻看。和 `AdapterView` 的一样, `PagerAdapter` 第一个需要重写的就是 `getCount()` 方法, 它会返回要显示条目的总数。

`ViewPager` 是基于跟踪每个条目的键对象以及显示该对象的视图进行工作的, 这样会将适配器条目和它们的视图(开发者在使用 `AdapterView` 经常会用到)的分离开来。但是它们的实现方式略有不同。如果使用 `AdapterView`, 适配器的 `getView()` 方法会构建和返回条目上显示的视图。而使用 `ViewPager`, 当创建一个新视图或者某个视图滚动超出了页数限制的范围后需要删除该视图时就会分别调用 `instantiateItem()` 和 `destroyItem()` 回调方法, 通过 `setOffscreenPageLimit()` 方法来设置每个 `ViewPager` 可持有条目的数量。

注意:

屏幕以外的页数, 默认值为 3。这意味着 `ViewPager` 将会跟踪当前可见页面、当前界面左侧的界面、当前界面右侧的界面。跟踪页面的编号总是围绕当前可见的页面进行的。

在我们的示例中我们使用 `instantiateItem()` 来创建一个新的 `ImageView` 并设置该 `ImageView` 的相关属性。和 `AdapterView` 不同, `PagerAdapt` 除了通过返回一个唯一的键对来表示某个条目外, 还必须把要显示的视图关联到给定的 `ViewGroup` 上。这两个操作不需要一定相同, 但可以像本例中这样简单处理。需要重写 `PagerAdapter` 的 `isViewFromObject()` 回调方法, 这样应用程序就可以将键对象和视图关联起来。在我们的示例中, 将 `ImageView` 添加到给定的父视图上并将该 `ImageView` 作为 `instantiateItem()` 的 key 对象返回值。如此一来 `isViewFromObject()` 中的代码变得简单了, 如果两个参数的引用是相同的就返回 `true`。

和初始化过程类似, `PagerAdapter` 同样需要在 `destroyItem()` 方法中将指定的视图从父容器移除。如果页面上显示的是重量级的视图, 同时你想实现可以在适配器中循环利用的视图, 这个视图被删除后你可以保存它, 这样它就可以在 `instantiateItem()` 中附加在另一个键

对象上。程序清单 2-117 展示了一个 Activity 示例，在 ViewPager 中使用我们自定义的适配器，图 2-27 中展示了应用程序的结果。

程序清单 2-117 使用 ViewPager 和 ImagePagerAdapter 的 Activity

```
public class PagerActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ViewPager pager = new ViewPager(this);
        pager.setAdapter(new ImagePagerAdapter(this));

        setContentView(pager);
    }
}
```



图 2-27 可以在两个页面之间进行拖动的 ViewPager

运行这个应用程序后，用户可以水平滑动手指来分页浏览自定义适配器提供的所有图像，而且每个图像都是全屏显示。本例中有一个定义的方法我们没有提到：`getPageWidth()`。这个方法允许你在每个位置上设置图片页面大小相对于 ViewPager 页面大小的百分比。默认值为 1，前面的示例也没有改变该默认值。但如果要一次显示几个页面，可以通过调整这个方法的返回值来实现。

如果按照下面的代码修改 `getPageWidth()`，那么我们一次可以显示三个页面：

```
/*
 * 如果要在 ViewPager 内一次显示超过一页的内容，那么需要重写该方法。
 */
@Override
```

```
public float getPageWidth(int position) {
    //每个页面的宽应该是视图的 1/3
    return 0.333f;
}
```

图 2-28 展示了应用程序的修改结果。

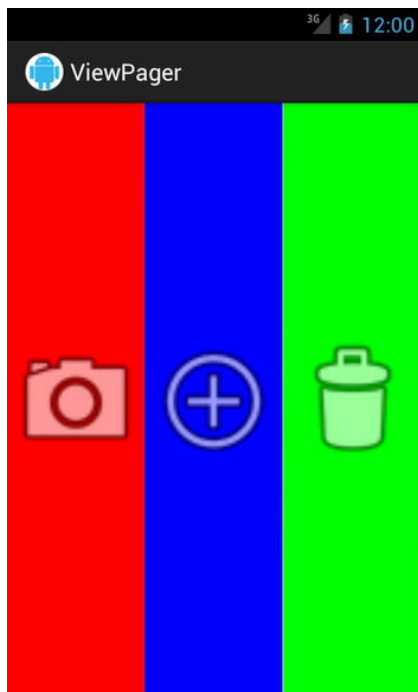


图 2-28 每次显示三个页面的 ViewPager

1. 添加和删除页面

程序清单 2-118 演示一个用于 ViewPager 的稍微复杂的适配器。它使用了框架中的 `FragmentPagerAdapter` 作为父类, `FragmentPagerAdapter` 的每个页面条目都是一个 `Fragment` 而不是简单的视图。

这个示例使用了一个很长的数据列表并将其分解成小段显示在每个页面上。这个适配器显示的 `Fragment` 是一个自定义内部实现, 它会接收一个条目的列表并将这些条目显示在 `ListView` 中。

程序清单 2-118 显示一个列表的 `FragmentPagerAdapter`

```
public class ListPagerAdapter extends FragmentPagerAdapter {

    private static final int ITEMS_PER_PAGE = 3;

    private List<String> mItems;

    public ListPagerAdapter(FragmentManager manager, List<String> items) {
        super(manager);
    }
}
```

```

        mItems = items;
    }

    /*
     * 这个位置首次需要一个 Fragment 时，该方法才会被调用
     */
    @Override
    public Fragment getItem(int position) {
        int start = position * ITEMS_PER_PAGE;
        return ArrayListFragment.newInstance(getPageList(position), start);
    }

    @Override
    public int getCount() {
        //得到分页的总数
        int pages = mItems.size() / ITEMS_PER_PAGE;
        //如果列表的大小不能整除页面的大小，就多添加一个页面来显示剩余的值。
        int excess = mItems.size() % ITEMS_PER_PAGE;
        if (excess > 0) {
            pages++;
        }

        return pages;
    }

    /*
     * 这个方法会在 getItem() 之后调用，而且在超出页数限制部分的 Fragment 再加回来时，
     * 也会调用该方法。我们要确保这些元素会被更新到列表中。
     */
    @Override
    public Object instantiateItem(ViewGroup container, int position) {
        ArrayListFragment fragment =
            (ArrayListFragment) super.instantiateItem(container,
                position);
        fragment.updateListItems(getPageList(position));
        return fragment;
    }

    /*
     * 当 notifyDataSetChanged() 被调用，该方法也会被框架所调用。我们必须决定如何为
     * 新的数据集更改 *Fragment。如果某个位置的 Fragment 不再需要，会返回 POSITION_NONE，
     * 这样适配器就可以将其删除。
     */
    @Override
    public int getItemPosition(Object object) {
        ArrayListFragment fragment = (ArrayListFragment) object;
        int position = fragment.getBaseIndex() / ITEMS_PER_PAGE;
        if (position >= getCount()) {
            //不再需要这个页面
            return POSITION_NONE;
        }
    }

```

```

        } else {
            //刷新 Fragment 数据显示
            fragment.updateListItems(getPageList(position));

            return position;
        }
    }

    /**
     * 辅助方法，用于获取整个列表的某一部分然后显示在给定的 Fragment 上。
     */
    private List<String> getPageList(int position) {
        int start = position * ITEMS_PER_PAGE;
        int end = Math.min(start + ITEMS_PER_PAGE, mItems.size());
        List<String> itemPage = mItems.subList(start, end);

        return itemPage;
    }

    /**
     * 内部自定义 Fragment，它会通过 ListView 中显示了列表的一个片段，并提供了外部方法
     来更新列表
     */
    public static class ArrayListFragment extends Fragment {
        private ArrayList<String> mItems;
        private ArrayAdapter<String> mAdapter;
        private int mBaseIndex;

        //按照惯例使用工厂模式创建 Fragment。
        static ArrayListFragment newInstance(List<String> page, int
        baseIndex) {
            ArrayListFragment fragment = new ArrayListFragment();
            fragment.updateListItems(page);
            fragment.setBaseIndex(baseIndex);
            return fragment;
        }

        public ArrayListFragment() {
            super();
            mItems = new ArrayList<String>();
        }

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //为列表条目创建一个新的适配器
            mAdapter = new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_1, mItems);
        }
    }

```

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    //构造并返回一个 ListView, 并为它关联我们的适配器。
    ListView list = new ListView(getActivity());
    list.setAdapter(mAdapter);
    return list;
}

//在全局列表中保存一个索引, 记录页面开始的地方。
public void setBaseIndex(int index) {
    mBaseIndex = index;
}

//在全局列表中检索索引, 可以找到页面开始的地方。
public int getBaseIndex() {
    return mBaseIndex;
}

public void updateListItems(List<String> items) {
    mItems.clear();
    for (String piece : items) {
        mItems.add(piece);
    }

    if (mAdapter != null) {
        mAdapter.notifyDataSetChanged();
    }
}
}
}

```

FragmentPagerAdapter 帮助我们实现了很多 PagerAdapter 底层的功能。不必再实现 instantiateItem()、destroyItem()和 isViewFromObject()方法, 只需要重写 getItem()来为每个页面位置提供相应的 Fragment。本例为每个页面应该显示的列表条目的数量定义了一个常量。在 getItem()内创建 Fragment 时, 会传入列表中的一部分数据, 而这些数据是根据索引偏移和之前定义的常量来计算的。分页的数量由 getCount()方法返回, 这个值是通过列表条目总量除以每页显示的条目常量计算得到的。

这个适配器还覆写了前面简单示例中未曾见到过的另一个方法: getItemPosition()。当应用程序从外部调用 notifyDataSetChanged()时, 这个方法会被调用。它主要的作用是页面发生变化时判断页面中的条目应该被移动还是被删除。如果条目的位置发生改变, 那么就应该返回新位置的值。如果条目不应该被移动, 就会返回一个常量值 PagerAdapter.POSITION_UNCHANGED。如果页面应该被删除, 应用程序应该返回 PagerAdapter.POSITION_NONE。

这个示例会比较检查当前页面的位置(我们需要从初始索引数据开始重新创建)和当前页面数量的大小。如果当前页面位置大于当前页面数量, 就需要从列表中删除足够的条目, 如此一来就不再需要该页面了, 然后返回 POSITION_NONE。而在其他情况下, 我们会更

新当前 Fragment 中显示的列表条目，并返回重新计算得到位置值。

每个 ViewPager 当前跟踪的页面都会调用 `getItemPosition()`，调用的次数即为 `getOffscreenPageLimit()` 返回的页面数量。然而，虽然 ViewPager 不会跟踪滚动出限定值之外的 Fragment，但 `FragmentManager` 会继续追踪。所以当之前的 Fragment 回滚时，`getItem()` 不会被再次调用，因为 Fragment 已经存在了。但是正因为如此，如果一个数据集在这期间发生改变，Fragment 列表数据不会跟着更新。这就是为什么要重写 `instantiateItem()`。虽然这个适配器不需要重写 `instantiateItem()`，当列表发生变化时，确实需要更新屏幕之外页数限制之外的 Fragment。因为 Fragment 回滚到页数限制以内后，每次都会调用 `instantiateItem()`，所以这是重置显示列表的好时机。让我们看一个使用该适配器的示例应用程序。参见程序清单 2-119 和 2-120。

程序清单 2-119 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add Item"
        android:onClick="onAddClick" />
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Remove Item"
        android:onClick="onRemoveClick" />

    <!--ViewPager 是支持库中的控件，所以需要完整的包名 -->
    <android.support.v4.view.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

程序清单 2-120 使用 ListPagerAdapter 的 Activity

```
public class FragmentPagerActivity extends FragmentActivity {

    private ArrayList<String> mListItems;
    private ListPagerAdapter mAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //创建初始数据集
    }
}
```

```

        mListItems = new ArrayList<String>();
        mListItems.add("Mom");
        mListItems.add("Dad");
        mListItems.add("Sister");
        mListItems.add("Brother");
        mListItems.add("Cousin");
        mListItems.add("Niece");
        mListItems.add("Nephew");
        //把数据关联到 ViewPager 上
        ViewPager pager = (ViewPager) findViewById(R.id.view_pager);
        mAdapter = new ListPagerAdapter(getSupportFragmentManager(),
        mListItems);

        pager.setAdapter(mAdapter);
    }

    public void onAddClick(View v) {
        //在列表的末尾添加一个新的唯一的条目
        mListItems.add("Crazy Uncle " + System.currentTimeMillis());
        mAdapter.notifyDataSetChanged();
    }

    public void onRemoveClick(View v) {
        //从列表顶部删除一个条目
        if (!mListItems.isEmpty()) {
            mListItems.remove(0);
        }
        mAdapter.notifyDataSetChanged();
    }
}

```

就像 ViewPager 的效果一样, 这个示例中有两个按钮用来添加和删除数据集中的条目。注意 ViewPager 必须在 XML 文件中定义并使用完整的包名, 因为它是支持库的类, 在 android.widget 或 android.view 包中并没有这个类。该 Activity 构建了一个默认的条目列表并把它传入我们自定义的适配器中, 然后再把该适配器关联到 ViewPager 上。

每次单击添加按钮都在列表末尾添加一个新的条目并通过调用 notifyDataSetChanged() 来触发 ListPagerAdapter 进行更新。每次单击删除按钮都会在列表顶部删除一个条目然后再次通知适配器。每次变化适配器都会调整当前可用页数并更新 ViewPager。如果当前可见页的所有条目都被删除, 那么该页也会被删除并显示上一页。

2. 其他有用的方法

ViewPager 中有几个其他的方法, 会对你的应用程序很有帮助:

- setPageMargin() and setPageMarginDrawable() 允许在页面之间设置额外的间隔并且使用一个 Drawable(可选的)来填充间隔的内容。
- setCurrentItem() 允许你以编程的方式设置要显示的页面, 并提供一个选项来禁用页面切换时的滚动动画。

- `OnPageChangeListener` 用于将滚动和变更动作通知给应用程序。
 - `onPageSelected()`会在显示一个新页面时被调用。
 - 当发生滚动操作时会连续调用 `onPageScrolled()`。
 - `onPageScrollStateChanged()`在 `ViewPager` 处于以下状态时会被调用：闲置时、用户主动滚动 `ViewPager` 时、自动滚动对齐到最近的页面时。

2.34 创建模块化接口

2.34.1 问题

希望应用程序的 UI 在存在多个设备配置时可以进行更多的复用。

2.34.2 解决方案

(API Level 4)

使用 `fragment` 来创建可复用的模块,它可以被插入到 `Activity` 代码中使得你的 UI 可以适应不同的设备配置,也可以为多个 `Activity` 设置一些通用的接口元素。`Fragment` 最早是从 `Android 3.0(API Level 11)`中引入的,之后成为支持库的一个重要的组成部分,从而允许在目标版本为 `Android 1.6(API Level 4)`以后版本的应用程序中使用它。

通过支持库使用 `fragment` 时,必须使用 `FragmentManager` 类代替默认的 `Activity`。这个类内置了较新平台才具备所有的 `Fragment` 功能,例如本地 `FragmentManager`。如果你的应用程序的目标平台为 `Android 3.0` 或以后版本,则不必再使用支持库并且直接使用 `Activity` 即可。

`Fragment` 的声明周期和 `Activity` 很像,因此 `Fragment` 中也有 `onCreate()`、`onResume()`、`onPause()`和 `onDestroy()`这样的回调方法。另外还有几个额外的生命周期回调方法,例如 `Fragment` 关联其父 `Activity` 时用到的 `onAttach()`和 `onDetach()`方法。作为 `setContentView()` 方法的替代品, `Android` 框架会调用 `onCreateView()`方法来获得要显示的内容。

`Fragment` 不必像 `Activity` 一样一定要拥有一个 UI 控件。如果不覆写 `onCreateView()`, `Fragment` 可以作为一个纯粹的数据源或者其他模块而存在。这种机制可以很好地将应用程序中的模型部分进行模块化处理,因为 `FragmentManager` 会使得 `Fragment` 之间的访问变得简单。另外, `FragmentManager` 还可以保存某个 `Fragment`,这样在设备配置变化时,可以避免持有数据(数据可能是从网上获得的)的 `Fragment` 被重新创建。

2.34.3 实现机制

这个示例用三个 `fragment` 演示了一个简单的概要-详情类的应用程序。宿主 `Fragment` 显示了一个用户可以访问的网站列表,而详情 `Fragment` 则通过一个 `WebView` 来显示选中网站列表条目后的详细 URL 信息。第三个 `Fragment` 是没有 UI 界面的,它只是用来为其他 `Fragment` 提供模型数据。根据设备显示方向的不同,为了更好地适应屏幕的实际空间大小,

这三个元素的显示也会有所不同。

首先，让我们看一下程序清单 2-121 所示的数据 Fragment。

程序清单 2-121 数据 Fragment

```
public class DataFragment extends Fragment {
    /*
     * 这是一个没有 UI 的 fragment 示例。
     * 它的作用是封装应用程序的数据逻辑，
     * 这样其他的 fragment 就可以很方便地访问该数据
     */

    public static final String TAG = "DataFragment";
    /*
     * 自定义的数据模型类，用来保存应用程序的数据
     */
    public static class DataItem {
        private String mName;
        private String mUrl;

        public DataItem(String name, String url) {
            mName = name;
            mUrl = url;
        }

        public String getName() {
            return mName;
        }

        public String getUrl() {
            return mUrl;
        }
    }

    /*
     * 工厂方法，用来创建新的实例
     */
    public static DataFragment newInstance() {
        return new DataFragment();
    }

    private ArrayList<DataItem> mDataSet;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //构建初始的数据集
        mDataSet = new ArrayList<DataFragment.DataItem>();
        mDataSet.add(new DataItem("Google", "http://www.google.com"));
        mDataSet.add(new DataItem("Yahoo", "http://www.yahoo.com"));
    }
}
```

```

        mDataSet.add(new DataItem("Bing", "http://www.bing.com"));
        mDataSet.add(new DataItem("Android", "http://www.android.com"));
    }

    //当前数据的访问器
    public ArrayList<DataItem> getLatestData() {
        return mDataSet;
    }
}

```

这个 `Fragment` 为我们的列表数据定义了一个自定义模型类, 并且构造了应用程序要使用的数据集。这个示例非常简单并且数据集是静态的, 但这里也可以放置从网络服务器获得下载数据的逻辑或者从 `ContentProvider` 获得数据库信息的逻辑(这两种方式会在后面章节做详细的介绍)。这个 `Fragment` 中没有视图控件, 但依然可以把它关联到 `FragmentManager` 上供应用程序的其他模块访问。

接下来, 在程序清单 2-122 中定义了宿主 `Fragment`。

程序清单 2-122 宿主视图 `Fragment`

```

public class MasterFragment extends DialogFragment implements
    AdapterView.OnItemClickListener {

    /*
     * 回调接口, 用来将选择的数据反馈给父 Activity
     */
    public interface OnItemSelectedListener {
        public void onDataItemSelected(DataItem selected);
    }

    /*
     * 工厂方法, 用来创建新的实例
     */
    public static MasterFragment newInstance() {
        return new MasterFragment();
    }

    private ArrayAdapter<DataItem> mAdapter;
    private OnItemSelectedListener mItemSelectedListener;

    /*
     * 在 onAttach 中去连接监听器接口, 并且确保我们要关联的 Activity 支持该接口
     */
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mItemSelectedListener = (OnItemSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new IllegalArgumentException(

```

```

        "Activity must implement OnItemSelectedListener");
    }
}

/*
 * 构建一个自定义适配器来显示数据模型中的 name 字段
 */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mAdapter = new ArrayAdapter<DataFragment.DataItem>(getActivity(),
        android.R.layout.simple_list_item_1) {
        @Override
        public View getView(int position, View convertView, ViewGroup
parent) {
            View row = convertView;
            if (row == null) {
                row = LayoutInflater.from(getContext())
                    .inflate(android.R.layout.simple_list_item_1,
                        parent, false);
            }

            DataItem item = getItem(position);
            TextView tv = (TextView) row.findViewById(android.R.id.text1);
            tv.setText(item.getName());

            return row;
        }
    };
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ListView list = new ListView(getActivity());
    list.setOnItemClickListener(this);
    list.setAdapter(mAdapter);
    return list;
}

/*
 * onCreateDialog 中可以直接访问要显示的对话框,
 * 我们在这里设置对话框的标题
 */
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Dialog dialog = super.onCreateDialog(savedInstanceState);
    dialog.setTitle("Select a Site");

    return dialog;
}

```

```

    }

    /**
     * resume 时, 从 DataFragment 中得到最新的数据信息
     */
    @Override
    public void onResume() {
        super.onResume();
        //获得最新的数据列表
        DataFragment fragment = (DataFragment) getFragmentManager()
            .findFragmentByTag(DataFragment.TAG);
        if (fragment != null) {
            mAdapter.clear();
            for (DataItem item : fragment.getLatestData()) {
                mAdapter.add(item);
            }
            mAdapter.notifyDataSetChanged();
        }
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position,
        long id) {
        //通知 Activity
        mItemSelectedListener.onDataItemSelected(mAdapter.getItem(position));

        //如果对话框是显示的, 则隐藏它。
        //在 fragment 嵌入到某个视图的情况下, 这个方法会返回 false。
        if (getShowsDialog()) {
            dismiss();
        }
    }
}

```

这个控件继承自 `DialogFragment`, `DialogFragment` 是 SDK 中一个特殊的控件, 拥有秘密的力量。`DialogFragment` 可以内嵌到一个 `Activity` 中并显示它的内容, 也可以在一个对话框中显示它的内容。这允许我们使用相同的代码来显示列表, 但会导致在有足够空间时, 它依然会被嵌入到 UI 中。在 `onCreate()` 中, 我们实现了一个自定义的 `ArrayAdapter`, 它可以在我们自定义模型类之外显示数据。在 `onCreateView()` 中, 我们创建了一个简单的 `ListView` 来显示模型数据。

在 `onResume()` 中, 我们看到了 `fragment` 间是如何通信的。这里会通过 `FragmentManager` 获得一个之前定义的 `DataFragment` 实例。如果存在的话, 会从 `Fragment` 中获得最新的数据模型列表。这个 `Fragment` 是通过它的 `tag` 得到的, 稍后我们会看到它的实现机制。

这个 `Fragment` 还定义了一个自定义的监听器接口, 用来与父 `Activity` 进行通信。在 `onAttach()` 回调方法中, 将我们关联的 `Activity` 设置为这个 `Fragment` 的监听器。这种方式只是众多与父 `Activity` 进行通信方法中的一种。如果在应用程序中, `Fragment` 总是会被关联到同一个 `Activity` 上, 可以简单地调用 `getActivity()`, 并将返回的 `Activity` 进行强转, 然后

直接访问 Activity 中已经写好的方法。我们可以用和访问 DataFragment 类似的方式让 MasterFragment 直接与 DetailsFragment 进行交互。

当选中列表中的一个条目后,就会通知监听器。DialogFragment 提供了 getShowsDialog() 方法来判断该 Fragment 当前是否被嵌入到一个 Activity 中或者正在作为一个 Dialog 显示。如果 Fragment 正在作为一个 Dialog 显示,选中后也会调用 dismiss()。

提示:

当 Fragment 作为 Dialog 显示时,从技术上讲 dismiss()方法就不起作用。它只是将视图从容器中移除而已。这种行为稍微有点尴尬,因此最好先检查一下当前的模式。

现在,查看最后一个 Fragment,程序清单 2-123 中的详情视图。

程序清单 2-123 详情视图 Fragment

```
public class DetailFragment extends Fragment {

    private WebView mWebView;

    /*
     * 自定义 web 客户端启用进度的显示。添加客户端会让 WebView 直接
     * 加载所有的请求而不是将请求交由系统浏览器处理
     */
    private WebViewClient mWebViewClient = new WebViewClient() {
        @Override
        public void onPageStarted(WebView view, String url, Bitmap favicon) {
            getActivity().setProgressBarIndeterminateVisibility(true);
        }

        public void onPageFinished(WebView view, String url) {
            getActivity().setProgressBarIndeterminateVisibility(false);
        }
    };

    /*
     * 创建并设置一个用来显示的 WebView
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        mWebView = new WebView(getActivity());
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.setWebViewClient(mWebViewClient);

        return mWebView;
    }

    /*
     * 外部方法,用来将一个新的网站内容加载到视图
     */
    public void loadUrl(String url) {
```

```

        mWebView.loadUrl(url);
    }

}

```

这个 Fragment 非常简单。这里只是创建了一个 WebView，它会加载传给它的 URL 中的内容。我们还关联了一个 WebViewClient 来监控加载的进度从而可以向用户显示进度条。关于 WebView 和 WebViewClient 的更多详细信息，可以查看第 3 章的范例。

重点:

因为这个应用程序使用了 WebView 来访问远程网站，所以你需要在 manifest 文件中声明 android.permission.INTERNET 属性。

最后，看一下为程序清单 2-124 到 2-126 定义的 Activity。

程序清单 2-124 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!--竖屏布局 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Show List"
        android:onClick="onShowClick" />
    <fragment android:name="com.examples.fragmentsample.DetailFragment"
        android:id="@+id/fragment_detail"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>

```

程序清单 2-125 res/layout-land/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<!--横屏布局 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <FrameLayout
        android:id="@+id/fragment_master"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />
    <fragment
        android:name="com.examples.fragmentsample.DetailFragment"
        android:id="@+id/fragment_detail"

```

```

        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />
</LinearLayout>

```

程序清单 2-126 宿主详情 Activity

```

public class MainActivity extends FragmentActivity implements
    MasterFragment.OnItemSelectedListener {

    private MasterFragment mMaster;
    private DetailFragment mDetail;

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

        //在窗口上启用一个进度条
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.main);
        setProgressBarIndeterminateVisibility(false);

        //加载数据 fragment。
        //如果 FragmentManager 不存在需要的 Fragment，就关联一个新的。
        DataFragment fragment = (DataFragment) getSupportFragmentManager()
            .findFragmentByTag(DataFragment.TAG);
        if (fragment == null) {
            fragment = DataFragment.newInstance();
            //我们希望保存这个实例，可以在配置变化时获得同一个实例
            fragment.setRetainInstance(true);
            //通过一个 tag 而不是容器 id 关联 fragment
            FragmentTransaction ft =
                getSupportFragmentManager().beginTransaction();
            ft.add(fragment, DataFragment.TAG);
            ft.commit();
        }

        //获得详情 fragment
        mDetail = (DetailFragment) getSupportFragmentManager()
            .findFragmentById(R.id.fragment_detail);

        //或者嵌入到宿主 fragment，或者直接作为 dialog 显示
        mMaster = MasterFragment.newInstance();
        //如果存在容器视图，嵌入 fragment
        View container = findViewById(R.id.fragment_master);
        if (container != null) {
            FragmentTransaction ft =
                getSupportFragmentManager().beginTransaction();
            ft.add(R.id.fragment_master, mMaster);
            ft.commit();
        }
    }
}

```

```

    }

    @Override
    public void onDataItemSelected(DataItem selected) {
        //将选中项通过详情视图来显示
        mDetail.loadUrl(selected.getUrl());
    }

    public void onShowClick(View v) {
        //如果存在按钮并且单击了按钮，就将 DetailFragment 作为一个对话框显示
        mMaster.show(getSupportFragmentManager(), null);
    }
}

```

我们为竖屏(默认的)和横屏创建了两个不同的布局。在竖屏布局中，使用<fragment>标签直接将详情 Fragment 嵌入到 UI 中。这时布局填充完成后会自动创建 Fragment 并关联它。竖屏情况下宿主列表无法显示，所以我们添加了一个按钮来让宿主列表通过 Dialog 的形式显示出来。在横屏布局中有足够的空间让它们并排显示。这里我们会再次嵌入详情 fragment，然后放置一个空的容器视图，该容器最终会关联宿主 fragment。

当首次创建 Activity 时，我们做的第一件事就是确保 DataFragment 已经被关联到 FragmentManager 上；如果没有关联，我们会创建一个新的实例并进行关联。这个 Fragment 中特意调用了 setRetainInstance()，它会告诉 FragmentManager，即使配置改变也要保存该 FragmentManager。这样就可以让该控件保存只会存在一次的数据模型并且在 UI 变化时数据也不会受到影响。

Fragment 是通过 FragmentTransaction 进行添加、移除和替换的。这是因为 Fragment 的操作是异步的。所有数据都会关联一个特定的操作，例如执行什么操作和操作是否应该加入到 BACK 按钮栈中，这些操作都会被设置为特定的 FragmentTransaction 并进行提交。

我们通过 FragmentManager 的 findFragmentById()方法获得 DetailsFragment。注意，这个 ID 就是每个布局中<fragment>标签的值。MasterFragment 是通过代码创建的，然后会根据布局的状态做相应的操作。如果我们空的容器存在的话，则会使用容器(想要显示内容视图的地方)的 ID 将该 fragment 关联到 FragmentManager 上。这样就可以快速将 MasterFragment 添加到视图体系中。如果容器不存在的话，什么也不做，因为 Fragment 稍后就会显示。

竖屏布局时，用户可以按下 Show List button，这时会调用 MasterFragment 的 show()方法从而让该 fragment 显示在一个 Dialog 中。同时 MasterFragment 会关联到 FragmentManager 上。记住，当用户选择列表上的一个选项后，监听器接口方法就会被调用。Activity 会将选择事件转发给 DetailsFragment，然后在 WebView 上显示相应的内容。

在图 2-29 和图 2-30 中可以看到应用程序在横屏和竖屏上是如何显示的。

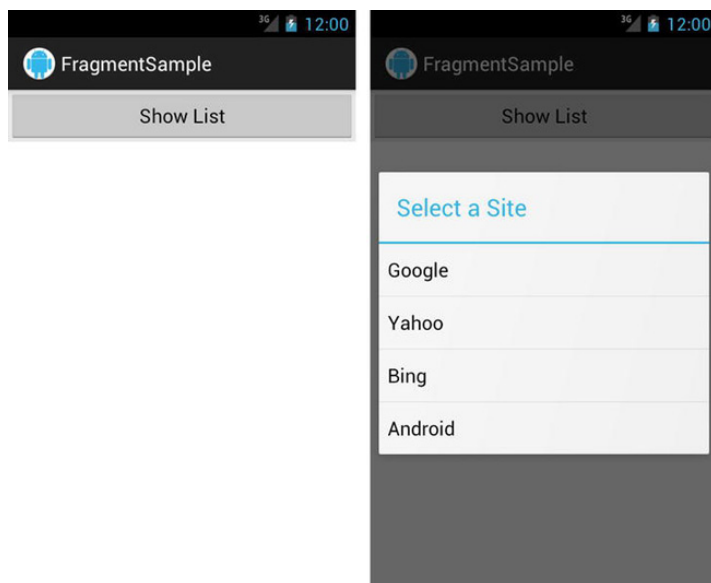


图 2-29 竖屏布局(左边)和显示对话框(右边)

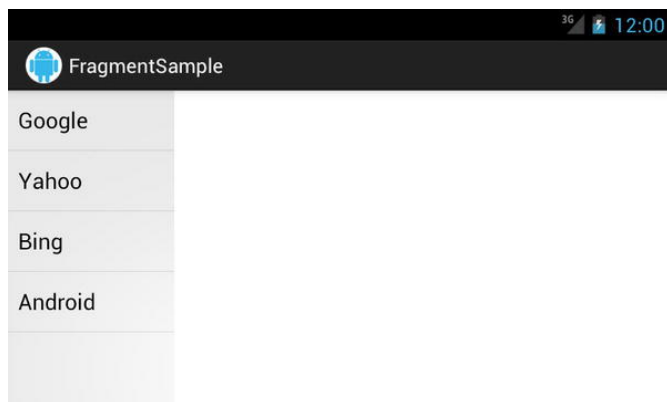


图 2-30 并排显示的横屏布局

Fragment 可以很完美地将你的代码模块化，这些模块可以被重组和复用，从而让应用程序在多个设备类型间可以很容易地进行缩放，而且非常易于维护。

2.35 高性能绘制

2.35.1 问题

应用程序需要在屏幕上渲染和绘制复杂的场景和动画，这些通常通过后台线程来完成。

2.35.2 解决方案

(API Level 1)

使用 `SurfaceView` 或 `TextureView` 将内容从后台线程渲染到屏幕上。在 Android UI 开发中一般遵循这样的规定：不要在主线程之外的线程中修改任何与 `View` 相关的属性。这两个则不遵循这个规定，这两个类专门设计用来在后台线程中执行绘制命令，并将绘制内容展现到屏幕上。在后面的章节中，你将看到 Android 框架使用这两个类来渲染相机的预览数据和视频输出。但是现在我们会关注如何实现我们的绘图。

`SurfaceView` 非常独特，与传统 `View` 的原理有很大差异。当实例化一个 `SurfaceView` 时，实际上会在 `View` 的位置创建另一个窗口，该窗口位于当前窗口的下方，然后 `View` 控件会在顶层窗口简单地“打一个洞”来透明地显示下面窗口的内容。这种方式的优势在于，可以在没有任何硬件加速支持的情况下实现高性能绘图。当然，这同时意味着 `SurfaceView` 是一种非常静态的视图，无法对动画和任何形式的变换做出很好的响应。

`TextureView` 适用于 Android 4.0 和之后的版本，在很多情况下可以顺便作为 `SurfaceView` 的替代品来使用。`TextureView` 的行为更像传统的 `View`，可以对绘制在它上面的内容实现动画和变形。但要求运行它的环境是硬件加速的，这可能会导致某些应用程序的兼容性问题。

2.35.3 实现机制

让我们看一个示例应用程序，这里一个后台线程将很多对象渲染到 `SurfaceView` 上。在这个示例中，我们会在屏幕上显示几个图标的连续动作动画。参见程序清单 2-127 和 2-128。

程序清单 2-127 res/layout/main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/button_erase"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Erase" />
    <SurfaceView
        android:id="@+id/surface"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_gravity="center" />

</FrameLayout>
```

程序清单 2-128 Surface 绘制的 Activity

```

public class SurfaceActivity extends Activity implements View.OnClickListener,
    View.OnTouchListener, SurfaceHolder.Callback {

    private SurfaceView mSurface;
    private DrawingThread mThread;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //为按钮关联监听器
        findViewById(R.id.button_erase).setOnClickListener(this);

        //通过一个触摸监听器和回调来设置 surface
        mSurface = (SurfaceView) findViewById(R.id.surface);
        mSurface.setOnTouchListener(this);
        mSurface.getHolder().addCallback(this);
    }

    @Override
    public void onClick(View v) {
        mThread.clearItems();
    }

    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            mThread.addItem((int) event.getX(), (int) event.getY());
        }
        return true;
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        mThread = new DrawingThread(holder,
            BitmapFactory.decodeResource(getResources(),
                R.drawable.ic_launcher));
        mThread.start();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
        int height) {
        mThread.updateSize(width, height);
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        mThread.quit();
    }

```

```

        mThread = null;
    }

    private static class DrawingThread extends HandlerThread implements
        Handler.Callback {
        private static final int MSG_ADD = 100;
        private static final int MSG_MOVE = 101;
        private static final int MSG_CLEAR = 102;

        private int mDrawingWidth, mDrawingHeight;

        private SurfaceHolder mDrawingSurface;
        private Paint mPaint;
        private Handler mReceiver;
        private Bitmap mIcon;
        private ArrayList<DrawingItem> mLocations;

        private class DrawingItem {
            //当前位置的标识
            int x, y;
            //动作方向的标识
            boolean horizontal, vertical;

            public DrawingItem(int x, int y, boolean horizontal, boolean
                vertical) {
                this.x = x;
                this.y = y;
                this.horizontal = horizontal;
                this.vertical = vertical;
            }
        }

        public DrawingThread(SurfaceHolder holder, Bitmap icon) {
            super("DrawingThread");
            mDrawingSurface = holder;
            mLocations = new ArrayList<DrawingItem>();
            mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
            mIcon = icon;
        }

        @Override
        protected void onLooperPrepared() {
            mReceiver = new Handler(getLooper(), this);
            //开始渲染
            mReceiver.sendMessage(MSG_MOVE);
        }

        @Override
        public boolean quit() {
            //退出之前清除所有的消息

```

```

        mReceiver.removeCallbacksAndMessages(null);
        return super.quit();
    }

    @Override
    public boolean handleMessage(Message msg) {
        switch (msg.what) {
            case MSG_ADD:
                //在触摸的位置创建一个新的条目, 该条目的开始方向是随机的
                DrawingItem newItem = new DrawingItem(msg.arg1, msg.arg2,
                    Math.round(Math.random()) == 0,
                    Math.round(Math.random()) == 0);
                mLocations.add(newItem);
                break;
            case MSG_CLEAR:
                //删除所有对象
                mLocations.clear();
                break;
            case MSG_MOVE:
                //渲染一帧
                Canvas c = mDrawingSurface.lockCanvas();
                if (c == null) {
                    break;
                }
                //首先清空 Canvas
                c.drawColor(Color.BLACK);
                //绘制每个条目
                for (DrawingItem item : mLocations) {
                    //更新位置
                    item.x += (item.horizontal ? 5 : -5);
                    if (item.x >= (mDrawingWidth - mIcon.getWidth())) {
                        item.horizontal = false;
                    } else if (item.x <= 0) {
                        item.horizontal = true;
                    }
                    item.y += (item.vertical ? 5 : -5);
                    if (item.y >= (mDrawingHeight - mIcon.getHeight())) {
                        item.vertical = false;
                    } else if (item.y <= 0) {
                        item.vertical = true;
                    }
                    //绘制到 Canvas
                    c.drawBitmap(mIcon, item.x, item.y, mPaint);
                }
                //显示到屏幕上
                mDrawingSurface.unlockCanvasAndPost(c);
                break;
        }
        //发送下一帧
        mReceiver.sendEmptyMessage(MSG_MOVE);
    }

```

```

        return true;
    }

    public void updateSize(int width, int height) {
        mDrawingWidth = width;
        mDrawingHeight = height;
    }

    public void addItem(int x, int y) {
        //通过 Message 参数将位置传给 Handler
        Message msg = Message.obtain(mReceiver, MSG_ADD, x, y);
        mReceiver.sendMessage(msg);
    }

    public void clearItems() {
        mReceiver.sendEmptyMessage(MSG_CLEAR);
    }
}

```

这个示例构造了一个简单的后台 `DrawingThread` 在 `SurfaceView` 上渲染和绘制内容。`DrawingThread` 是 `HandlerThread` 的子类, `HandlerThread` 是一个方便的框架辅助类, 用来生成后台线程来处理收到的消息。我们将在第 6 章详细探讨这种模式, 但现在只要说明我们的后台线程的响应操作是通过发送给 `Handler` 的 `handleMessage()` 的消息决定的。`SurfaceView` 实际上由两部分组成: 一个窗口下方的 `Surface`, 一个布局结构中的空白 `View`。要实现绘图功能, 实际上需要访问底层的 `Surface`, 它被封装在一个 `SurfaceHolder` 中。

直到 `view` 关联到当前的 `window` 后, `Surface` 才会开始构建。相反, 当创建、销毁或改变 `Surface` 时, `SurfaceHolder` 会有一个回调接口, 所以我们可以通过 `SurfaceHolder` 来管理依赖它的控件(本例中是 `DrawingThread`)的生命周期。本例中会在 `surfaceCreated()` 中创建一个新的 `DrawingThread` 并开始渲染, 在 `surfaceDestroyed()` 中停止渲染(因为此时 `Surface` 已经无效了)。只有最后的回调函数 `surfaceChanged()` 中提供 `Surface` 的尺寸值, 所以会在这里根据尺寸值(可用时)修改绘制代码。

我们为线程定义了三个不同的响应指令: `add`、`clear` 和 `move`。用户单击 `SurfaceView` 时会触发 `add` 相关的方法, 此时会在显示列表中添加一个绘制条目, 它的初始位置即为触点的位置。当按下按钮时, `clear` 方法会将显示列表中的所有条目都删除。

当线程向 `SurfaceView` 渲染每帧画面时, `move` 方法十分高效。每一个绘图操作都需要以 `lockCanvas()` 开始, `lockCanvas()` 可以提供一个 `Canvas` 来响应绘图调用。之后线程会迭代显示列表中的每一个条目, 并将每个条目更新到新的位置上, 然后每个更新后的位置绘制一个图标。同样会检查是否有条目超过了 `Surface` 边界, 如果有就将方向设置为相反的方向。在绘制每幅画面之前必须调用 `drawColor()` 方法清除之前画面的内容。如果没有这样的操作, 当图标移动时会在图标之前的位置出现拖尾的痕迹。某些应用程序可能需要这种拖尾效果(比如绘画应用程序, 会在一个事件上面添加另一个事件)但我们的示例并不需要这种效果。当所有的绘制调用完成之后, 应用程序需要调用 `unlockCanvasAndPost()` 方法将数据真正地渲染到屏幕上。

通过不断地给自己发送 MSG_MOVE 消息, DrawingThread 在整个处理过程中会一直运行直到被应用程序退出。通过 HandlerThread 来完成这个处理过程的好处是,可以随时调用 quit()方法来取消操作,而且线程可以销毁得很干净,这种方式比尝试中断线程的执行要好很多。

可以在图 2-31 中看到应用程序的运行结果。用户可以单击黑色区域(不限制次数),然后就可以看到飞行的图标叠加在一起。因为绘制代码只是用一个 bitmap 来绘制所有的图标,该视图能够支持很多的绘制条目,所以不用担心内存问题。

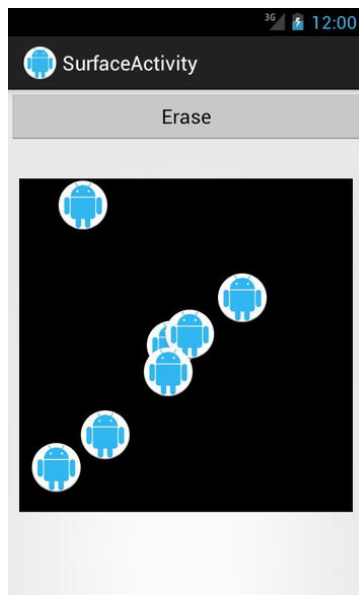


图 2-31 SurfaceView 绘图场景

1. TextureView

(API Level 14)

如果你的应用程序的目标版本为 Android 4.0 和之后的版本,也可以使用 TextureView,它有几个附加的属性可以使得你的应用更加完美。最有用的属性就是它可以进行变换。看一看程序清单 2-129 和 2-130,我们使用 TextureView 对前一个示例进行了修改。

程序清单 2-129 res/layout/main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/button_transform"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Rotate" />
    <TextureView
        android:id="@+id/surface"
        android:layout_width="300dp"
```

```

        android:layout_height="300dp"
        android:layout_gravity="center" />

</FrameLayout>

```

程序清单 2-130 Texture 绘制的 Activity

```

public class TextureActivity extends Activity implements View.OnClickListener,
    View.OnTouchListener, TextureView.SurfaceTextureListener {

    private TextureView mSurface;
    private DrawingThread mThread;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.texture);
        //为按钮关联监听器
        findViewById(R.id.button_transform).setOnClickListener(this);

        //通过一个触摸监听器和回调来设置 surface。
        mSurface = (TextureView) findViewById(R.id.surface);
        mSurface.setOnTouchListener(this);
        mSurface.setSurfaceTextureListener(this);
    }

    @Override
    public void onClick(View v) {
        mSurface.animate()
            .rotationBy(180.0f)
            .setDuration(750);
    }

    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            mThread.addItem((int) event.getX(), (int) event.getY());
        }
        return true;
    }

    @Override
    public void onSurfaceTextureAvailable(SurfaceTexture surface, int width,
        int height) {
        mThread = new DrawingThread(new Surface(surface),
            BitmapFactory.decodeResource(getResources(),
                R.drawable.ic_launcher));
        mThread.updateSize(width, height);
        mThread.start();
    }

    @Override

```

```

public void onSurfaceTextureSizeChanged(SurfaceTexture surface, int width,
    int height) {
    mThread.updateSize(width, height);
}

@Override
public void onSurfaceTextureUpdated(SurfaceTexture surface) {
    //对每一帧做相应处理。
}

@Override
public boolean onSurfaceTextureDestroyed(SurfaceTexture surface) {
    mThread.quit();
    mThread = null;

    //返回 true 并允许框架释放 surface。
    return true;
}

private static class DrawingThread extends HandlerThread implements
    Handler.Callback {
    private static final int MSG_ADD = 100;
    private static final int MSG_MOVE = 101;
    private static final int MSG_CLEAR = 102;

    private int mDrawingWidth, mDrawingHeight;

    private Surface mDrawingSurface;
    private Rect mSurfaceRect;
    private Paint mPaint;

    private Handler mReceiver;
    private Bitmap mIcon;
    private ArrayList<DrawingItem> mLocations;

    private class DrawingItem {
        //当前位置标识
        int x, y;
        //运动方向的标识
        boolean horizontal, vertical;

        public DrawingItem(int x, int y, boolean horizontal,
            boolean vertical) {
            this.x = x;
            this.y = y;
            this.horizontal = horizontal;
            this.vertical = vertical;
        }
    }
}

```

```

public DrawingThread(Surface surface, Bitmap icon) {
    super("DrawingThread");
    mDrawingSurface = surface;
    mSurfaceRect = new Rect();
    mLocations = new ArrayList<DrawingItem>();
    mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    mIcon = icon;
}

@Override
protected void onLooperPrepared() {
    mReceiver = new Handler(getLooper(), this);
    //开始渲染
    mReceiver.sendMessage(MSG_MOVE);
}

@Override
public boolean quit() {
    //退出之前清除所有消息。
    mReceiver.removeCallbacksAndMessages(null);
    return super.quit();
}

@Override
public boolean handleMessage(Message msg) {
    switch (msg.what) {
        case MSG_ADD:
            //在触摸的位置创建一个新的条目，该条目的开始方向是随机的
            DrawingItem newItem = new DrawingItem(msg.arg1, msg.arg2,
                Math.round(Math.random()) == 0,
                Math.round(Math.random()) == 0);
            mLocations.add(newItem);
            break;
        case MSG_CLEAR:
            //删除所有的对象
            mLocations.clear();
            break;
        case MSG_MOVE:
            //渲染一帧
            try {
                Canvas c = mDrawingSurface.lockCanvas(mSurfaceRect);
                if (c == null) {
                    break;
                }
            }
            //首先清空 Canvas
            c.drawColor(Color.BLACK);
            //绘制每个条目
            for (DrawingItem item : mLocations) {
                //更新位置
                item.x += (item.horizontal ? 5 : -5);
            }
        }
    }
}

```

```

        if (item.x >= (mDrawingWidth - mIcon.getWidth())) {
            item.horizontal = false;
        } else if (item.x <= 0) {
            item.horizontal = true;
        }
        item.y += (item.vertical ? 5 : -5);
        if (item.y >= (mDrawingHeight - mIcon.getHeight())) {
            item.vertical = false;
        } else if (item.y <= 0) {
            item.vertical = true;
        }
        //绘制到 Canvas 上
        c.drawBitmap(mIcon, item.x, item.y, mPaint);
    }
    //释放已经渲染的 surface
    mDrawingSurface.unlockCanvasAndPost(c);
} catch (Exception e) {
    e.printStackTrace();
}
break;
}
//发送下一帧
mReceiver.sendEmptyMessage(MSG_MOVE);
return true;
}

public void updateSize(int width, int height) {
    mDrawingWidth = width;
    mDrawingHeight = height;
    mSurfaceRect.set(0, 0, mDrawingWidth, mDrawingHeight);
}

public void addItem(int x, int y) {
    //通过 Message 参数将位置传给 Handler
    Message msg = Message.obtain(mReceiver, MSG_ADD, x, y);
    mReceiver.sendMessage(msg);
}

public void clearItems() {
    mReceiver.sendEmptyMessage(MSG_CLEAR);
}
}
}

```

在这个修改过的示例中，布局有一个 `TextureView` 的实例。和 `SurfaceView` 类似，直到 `view` 关联到当前的 `window` 上后，底层的 `Surface` 才会开始构建，所以我们在访问它之前需要依赖一个回调。对于 `TextureView` 来说，这个回调就是 `SurfaceTextureListener`。它的大多数方法都和 `SurfaceHolder.Callback` 中的方法很像，例如 `onSurfaceTextureAvailable()`、`onSurfaceTextureChanged()`和 `onSurfaceTextureDestroyed()`。但是有一个额外的回调方法

onSurfaceTextureUpdated()在本例中并没有用到。当 SurfaceTexture 渲染一个新的帧时会调用该方法。

TextureView 提供的界面绘制 surface 略有不同,它不用通过 SurfaceHolder 封装来访问。而是访问一个 SurfaceTexture 实例,该实例会封装一个新的 Surface 来进行绘图。因此,需要对我的 DrawingThread 做一点小的修改。SurfaceHolder 中有一个方便的无参版本的 lockCanvas()方法可以将整个 Surface 标记为“脏”的。但当直接使用 Surface 时,该方法并不存在。所以,我们需要在 lockCanvas()传入一个 Rect,从而确定 Surface 的那一部分在渲染时可以作为 Canvas 返回。因为我们仍然希望返回整个 surface,所以我们在 updateSize()中维护 Rect 的大小,这个方法在界面发生改变时会被监听事件调用。

为了生动地展示在 SurfaceTexture 渲染过程中对其进行变换,我们用 Rotate 按钮取代了 Erase 按钮。每次单击该按钮都会使 TextureView 做半周的旋转动画。当前动画运行时单击按钮会取消当前动画并从当前的点开始一个新的旋转,所以如果快速单击按钮视图会旋转出一个非常古怪的角度。整个过程中 SurfaceTexture 都会保持动态而且很流畅。在图 2-32 中你可以看到 TextureView 倒着旋转的应用程序。

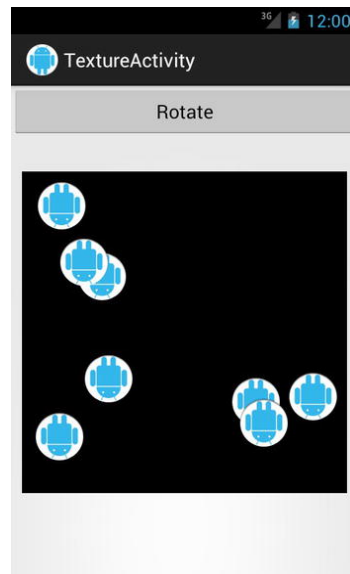


图 2-32 TextureView 绘制场景

2.36 实用工具推荐: Hierarchy Viewer 和 Lint

有时候布局会影响应用程序的速度。为了帮助调试布局中的问题,Android SDK 提供了 Hierarchy Viewer 和 Lint 工具。

2.37 Hierarchy Viewer

Hierarchy Viewer 是一个带界面的工具,用来调试和优化 UI。它提供了视图层次结构的可视化显示并且可以放大显示(Pixel Perfect View)

2.37.1 运行 Hierarchy Viewer

Hierarchy Viewer 可以通过命令行运行或者在 Eclipse 内运行。按照下面的步骤就可以通过命令行运行 Hierarchy Viewer:

(1) 连接你的设备或者开启一个模拟器。为了确保安全, Hierarchy Viewer 只会连接运行 Android 开发者系统版本的设备。

注意：

如果在设备上测试的话，可以使用 Google UI Framework 团队开发的开源项目 ViewServer 让 Hierarchy Viewer 授权访问应用程序中某个单独的窗口。关于在你的项目中引入该开源项目的更多信息，可以访问 <https://github.com/romainguy/ViewServer>。

(2) 首先安装一个你想要测试的应用程序。

(3) 运行这个应用程序并确保其用户界面是可见的。

(4) 在你的平台上运行 `hierarchyviewer`。这个工具位于 Android SDK 根目录下 `tools` 子目录中。

(5) 你看到的第一个界面显示了设备和模拟器的列表。想要展开指定模拟器或设备的活动对象列表，可以单击左边的箭头。这样就会显示设备或模拟器上用户界面当前可见的活动对象列表。列表中的对象则是通过 Android 的组件名称来标识的。该列表中同时包含了你的应用程序的 Activity 对象和系统的活动对象。图 2-33 显示了该界面的一个截图。

图 2-33 可以使用 Hierarchy Viewer 设备窗口查看 View Hierarchy 和 Pixel Perfect 窗口。

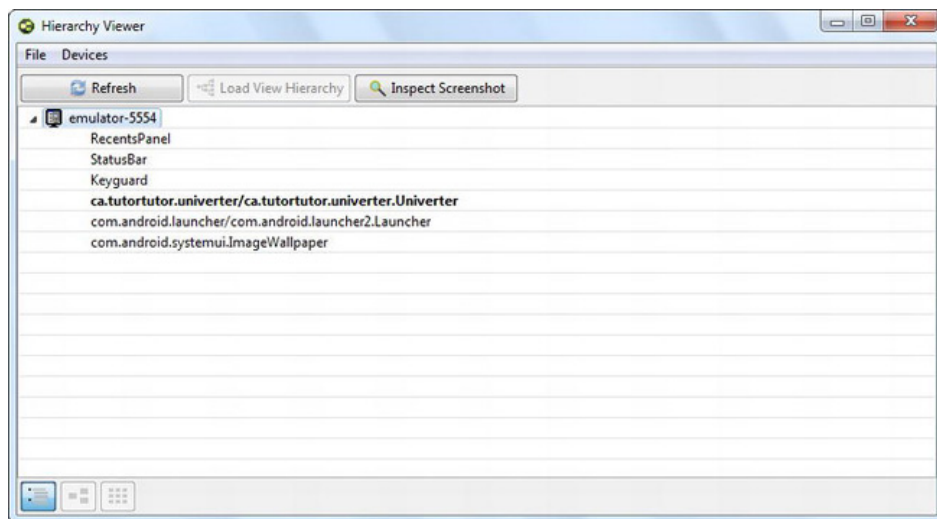


图 2-33 Hierarchy Viewer 的设备窗口包含 View 层次结构窗口和 Pixel Perfect 窗口

(6) 从列表中选择你的 Activity 的名称。然后就可以通过 View Hierarchy 窗口看到它的视图层次结构，或者可以通过 Pixel Perfect 窗口看到 UI 的放大效果。

图 2-33 显示了 Hierarchy Viewer 的用户界面。顶部的菜单栏提供了 File 和 Devices 菜单，File 菜单提供了 About 和 Exit 菜单项，而 Devices 菜单提供了 Refresh、Load View Hierarchy 和 Inspect Screenshot 菜单项，这几个选项同样会在菜单栏下方的工具栏上显示。

设备窗口位于工具栏下方，显示了所有已连接设备和模拟器的层次结构以及这些设备和模拟器下所有活动窗口的列表。当前可见的窗口条目用粗体显示，本例中显示的是 Univerter 应用程序的 Activity 窗口。

注意：

第 1 章介绍了 Univerter。附录 D 探讨了它的源代码、资源文件和 manifest。

最后，在设备窗口下方会有一个状态栏。左边的三个按钮用于在设备窗口、View Hierarchy 窗口和 Pixel Perfect 窗口之间进行切换。

想要在设备窗口访问 View Hierarchy 窗口中的一个活动窗口条目，可以选择在该条目后单击 Load View Hierarchy 按钮(或者双击该条目)。而单击 Inspect Screenshot 按钮则可以在 Pixel Perfect 窗口中访问粗体窗口。

2.38 浏览 View Hierarchy 窗口

View Hierarchy 窗口显示了设备或模拟器上运行的 Activity UI 中的视图对象。使用这个窗口可以查看整个 view 层次结构上下文中的单个视图对象。针对每个 View 对象，View Hierarchy 窗口同样会显示它的渲染性能数据。

图 2-34 显示了 `ca.tutortutor.univerter/ca.tutortutor.univerter.Univerter` 的 View Hierarchy 窗口(稍后要讨论的 Tree View 窗格已经被展开,并且选中了窗格左侧的 LinearLayout 节点)。



图 2-34 View Hierarchy 窗口分为 4 个窗格

View Hierarchy 窗口分成四个窗格：

- **Tree View:** 左手边的窗格显示了 Tree View，Tree View 提供了 Activity 对象的视图层次结构的图表。使用 Tree View 不仅可以查看用户界面上的每个视图对象而且可以看到各个视图对象之间的关系。
- 可以通过窗格底部的滑动条或者鼠标的滚轮缩放该窗格。要游览整个窗格或者

查看当前不可见的视图对象则需要单击并拖动窗格。

- 在“Filter byclass or id”文本框中输入字符串，Tree View 会高亮显示匹配该搜索字符串的类或 ID。匹配该搜索字符串的节点的背景会由灰色变为明亮的蓝色。单击 View Hierarchy 窗口顶部的“Save As PNG”会将 Tree View 的屏幕截图保存为一个 PNG 文件。此时会显示一个选择目录和文件名的对话框。
- 单击 View Hierarchy 窗口顶端的“Capture Layers”会将设备或模拟器的分层屏幕截图保存到一个 Adobe Photoshop (PSD)文件中，此时会显示一个选择目录和文件名的对话框。用户界面上的每个视图都会保存为一个单独的 Photoshop 图层。
- 在 Photoshop(或类似可以接受.psd 格式文件的程序)中，你可以单独隐藏、显示或编辑某个与其他图层无关的图层。当保存了一个分层的屏幕截图后，你可以单独地检查和修改某个视图的图像，这样可以帮助你试验设计效果。
- Tree Overview: 右上方的窗格显示的是 Tree Overview，它是 Tree View 窗格的缩略图。使用 Tree Overview 可以标识 Tree View 中显示的内容。
 - 也可以使用 Tree Overview 来浏览 Tree View 窗格。单击和拖曳某个阴影矩形区域，就会在 Tree View 中显示它。
- Properties View: 右手边中间的窗格是 Properties View，一个被选中视图对象的属性列表。有了 Properties View，无需 app 资源即可来查看所有的属性对象。
 - 这些属性是按照种类划分的。要查找某个属性，可以单击种类名称左侧的箭头展开该种类。这样就可以显示该类别的所有属性。
- Layout View: 右手边下方的窗格显示的是 Layout View，一个用户界面的缩略图。Layout View 是浏览用户界面的另一种方式。当单击 Tree View 中的某个视图对象，它在 UI 中相对应的位置就会高亮显示。反之，当单击 Layout View 中的某个位置时，Tree View 中与此位置相对应的视图对象也会高亮显示。
 - Layout View 中框的轮廓颜色会提供额外的信息。
 - 粗体红色：这个框代表 Tree View 中当前被选中的视图。
 - 淡红色：这个框代表粗体红色框的父类。
 - 白色：这个框代表一个可见的视图，该视图不是 Tree View 中当前被选中视图的父类或子类。
 - 检查 Properties View 中的“Show Extras”复选框或者单击“Load All Views”按钮，可以看到 ViewGroup(比如：线性布局)在 Layout View 中的实际内容。

当前的 Activity 发生改变时，View Hierarchy 窗口不会自动更新。可以单击窗口顶部的“Load View Hierarchy”来进行更新。

同样，当切换到新 Activity 时窗口也不会更新。如果要更新的话，单击窗口(设备窗口)底部最左边的图标就会返回到设备窗口，在这个窗口中单击新 Activity 的 Android 组件名，然后单击窗口顶部的“Load View Hierarchy”。

2.39 Tree View 中的单个视图

Tree View 中的每个节点代表一个单独的视图。有些信息总是可见的,选中某个节点后,你会看到(或看不到)以下信息:

- **View class:** 视图对象的类,在图 2-34 中是 `LinearLayout`。
- **View object address:** 指向视图对象的指针,在图 2-34 中是 `@4103c250`。
- **View object ID:** `android:id` 的属性值。在图 2-34 中没有 ID 是因为 `res/layout/main.xml` 文件(Univerter 运行在竖屏方向上)的 `<LinearLayout>` 元素没有 `android:id` 属性。
- **性能指标:** 一组着色的圆点表明了这个视图相对于 Tree View 中其他视图对象的渲染速度。三个圆点分别代表(从左到右)渲染时的测量、布局和绘制时间。
- 不同的颜色代表以下相关的性能:
 - 绿色: 对于这部分渲染时间,这个视图比 Tree View 中其他 50% 的视图都快。例如,测量时间的绿色圆点代表该视图的测量时间比 Tree View 中其他 50% 的视图对象都快。
 - 黄色: 对于这部分渲染时间,这个视图比 Tree View 中其他 50% 的视图都慢。例如,布局时间的黄色圆点代表该视图的布局时间比 Tree View 中其他 50% 的视图对象都慢。
 - 红色: 对于这部分渲染时间,这个视图是 Tree View 中最慢的。举个例子来说,一个红色圆点代表这个视图的绘制占用了所有视图对象绘制时间中的大部分时间。
 - 虽然被选中的 `LinearLayout` 节点能够很好地体现性能指标,但它并不是所有的圆点都会显示, `HierarchyViewer` 的 bug 可能会导致某个圆点无法显示。
- **视图索引:** 在父视图中的索引(从 0 开始)。如果该视图是唯一的子视图,那么它的索引就是 0。当选中一个节点时,会在节点上方出现一个小窗口显示该节点的其他信息。当单击一个节点后,你会看到下面的信息:
 - **图像:** 视图的实际图像,同模拟器中显示的图像是一样的。如果该视图有子视图,子视图也会显示出来。
 - **视图数量:** 该节点中的视图对象个数,该数量包括视图本身和它的子视图个数。比如:如果一个视图的数量是 4 的话,那么它有 3 个子视图。
 - **渲染时间:** 一个视图渲染时实际需要的测量、布局和绘制时间(以毫秒为单位)。这些值对应前面提到的各项性能指标。

2.40 使用 View Hierarchy 进行调试

View Hierarchy 窗口提供了一个静态显示的用户界面来帮助调试应用程序。首先会显示应用程序的开机画面,单步调试应用程序时,只有通过刷新和请求布局来进行视图的重

绘后，这个界面才会发生变化。

完成下面的步骤重绘一个视图：

(1) 选择 Tree View 中的一个视图。当向 Tree View 的根视图移动(向 Tree View 的左侧)时，会看到最高级别的视图对象。重绘高级别的视图对象一般也会强制重绘低级别的视图对象。

(2) 单击窗口顶部的“Invalidate”。这会将一个视图标记为无效，然后在下一次请求布局时进行重绘。

(3) 单击“Request Layout”来请求一个布局。这个视图和它的子视图会被重绘，其他需要重绘的视图对象也会被重绘。

手动重绘一个视图，可以让你通过代码断点一步一步地看到视图对象树和检查每个视图对象的属性。

使用 View Hierarchy 优化视图

View Hierarchy 可以帮助你识别渲染性能比较慢的地方。通过红色或黄色性能指示器(三个圆点)可以识别出在测量、布局和绘制时间上较慢的视图对象。请记住性能慢不一定会有问题。视图对象拥有太多的子视图和更为复杂的视图对象，会导致绘制性能更慢。

2.41 浏览 Pixel Perfect 窗口

Pixel Perfect 窗口显示设备或模拟器当前可见屏幕的放大图像。可以用来检查屏幕图像每个像素的属性。也是可以使用 Pixel Perfect 窗口来帮助展示基于 bitmap 的应用程序 UI。图 2-35 展示了 `ca.tutortutor.univerter/ca.tutortutor.univerter.Univerter` 的 Pixel Perfect 窗口。

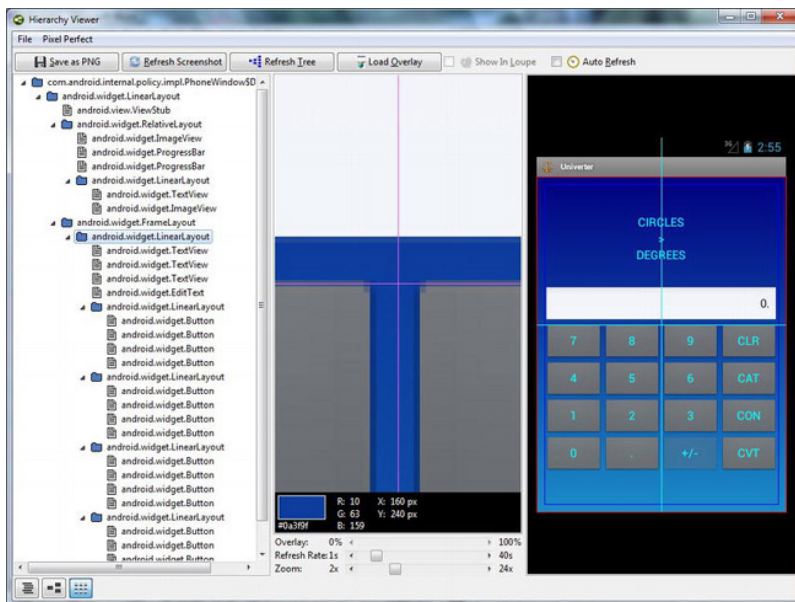


图 2-35 Pixel Perfect 窗口被分为三个窗格

Pixel Perfect 窗口被分为三个窗格：

- **View Object:** 它是一个设备和模拟器屏幕上当前可见视图对象的一个层次列表。这个列表包括应用中视图对象和系统生成的视图对象。这些对象根据它们的视图类被列出。要查看某个视图对象的子视图对象，可以单击它左边的箭头将其展开。当单击一个视图时，右边 Pixel Perfect 窗格相应的位置就会高亮显示。
- **Pixel Perfect Loupe:** 这是一个放大的图像。它叠加了一个网格，其中每一个格子代表一个像素。想要查看某个像素的信息，可以单击像素的区域。它的颜色和坐标会显示在窗格底部。
 - 红色十字线对应旁边窗格中的十字线位置。只有移动旁边窗格的十字线时它才会移动。
 - 可以通过窗格底部的 Zoom 滑块或者鼠标滚轮来放大或缩小图像。
 - 当选择了 Pixel Perfect Loupe 窗格中的一个像素后，会在窗格底部看到以下信息：
 - **Pixel swatch:** 一个和像素相同颜色的矩形。
 - **HTML color code:** 像素颜色对应的十六进制 RGB 码。
 - **RGB color values:** 像素颜色的红(R)、绿(G)和蓝(B)颜色值列表。每个值的范围是 0~255。
 - **X and Y coordinates:** 像素坐标，单位为 pixel。该值是从 0 开始的，X=0 位于屏幕的左边，Y=0 位于屏幕的顶端。
- **Pixel Perfect:** 该窗格显示当前可能在模拟器中显示的可见屏幕。
 - 使用青色的十字线进行粗略的定位。在图像中拖曳十字线，Loupe 十字线也会相应地移动。也可以单击 PixelPerfect 窗格中的某一点，此时十字线会移动到被单击的那个点。
 - 该图像对应于 View Object 窗格被选中的视图对象，它的轮廓是一个框，表示了该视图对象在屏幕中的位置。被选中的视图对象的轮廓框用深红色表示。它的兄弟视图和父视图用淡红色框表示。如果既不是其兄弟视图也不是其父视图用白色框表示。
 - 在布局框的内部或外部可能会有其他的矩形，每个矩形都代表视图的一部分。紫色或绿色的矩形代表视图的边界框。布局框内部中的白色或黑色框代表的是 padding，它定义了视图内容和其相邻边界框之间的距离。而外部的黑色或白色矩形代表的是 margin，它定义了边界框的边缘同相邻视图对象之间的距离。当布局的背景颜色为黑色时 padding 和 margin 框被着色为白色，而当布局的背景颜色为白色时它们被着色为黑色。
 - 可以将 Pixel Perfect 窗格当前显示的屏幕保存为一个 PNG 文件。这样做会创建一个当前屏幕截图。只需要单击窗口顶端的“Save as PNG”即可保存。此时显示一个对话框来选择目录和文件名。

当切换视图对象或启动其他 Activity 时窗格并不会自动刷新。需要单击窗口顶端的“Refresh Screenshot”来刷新 Pixel Perfect 和 Loupe 窗格。单击这个按钮会切换窗格呈现

当前屏幕的图像。你可能还需要刷新 View Object 窗格，单击窗口顶端的“Refresh Tree”就可以了。

想要在调试时自动刷新窗格，只需要选中窗口顶端的 Auto Refresh 复选框，然后使用 Loupe 窗格底部的 RefreshRate 滑块来设置刷新频率即可。

2.42 使用 Pixel Perfect Overlays

UI 的构建通常是基于位图的。为了帮助将视图布局匹配到位图上，Pixel Perfect 窗口允许将位图作为叠层加载到屏幕图像上。执行下面步骤可以将位图作为叠层使用：

(1) 在设备或模拟器中启动应用程序，并切换到要操作的 UI 所属的 Activity。

(2) 启动 Hierarchy Viewer 并切换到 Pixel Perfect 窗口。

(3) 单击窗口顶端的“Load Overlay”。会打开一个对话框提示加载图像文件。然后加载图像文件。

(4) Pixel Perfect 会在 Pixel Perfect 窗格中显示屏幕图像上的叠层。位图的左下角(X=0, Y=最大值)被锚定在屏幕左下角的像素上(X=0, Y=屏幕最大值)。

(5) 默认情况下，叠层的透明度是 50%，这样可以看到叠层下面的屏幕图像。可以通过 Loupe 窗格底部的 Overlay 滑块调整叠层的透明度。

(6) 同样，默认情况下叠层不会显示在 Loupe 窗格中。可以通过设置窗口顶端的“Show In Loupe”，将其显示在 Loupe 窗格中。

当将屏幕图像保存为一个 PNG 文件时，图层将不会作为屏幕截图的一部分被保存进去。

2.43 Lint

Lint 是一个静态扫描工具，用于帮助优化应用程序中的布局和布局层次结构，以及检测其他常见的编码问题。你可以针对你的布局文件或资源目录使用该工具，用来快速检查导致效率低下或其他影响应用程序性能的问题。

注意：

糟糕的代码结构会影响 Android 应用程序的可靠性和效率，并且使得代码难以维护。

比如，XML 资源文件中未使用的命名空间会占用空间和导致不必要的处理。其他结构性问题，如使用目标 API 版本不支持的已经废弃的元素或者 API，都可能会导致代码无法正常运行。

Lint 发现的每个问题报告中，都会有描述消息和严重级别，这样你就可以快速地把需要修改的问题按照优先级排序。也可以配置问题的严重级别来忽略与项目无关的问题，或者也可以提高严重级别。这个工具有命令行接口，因此可以很容易将其集成到自动化测试过程中。

图 2-36 展示了 Lint 如何处理应用程序的源文件。

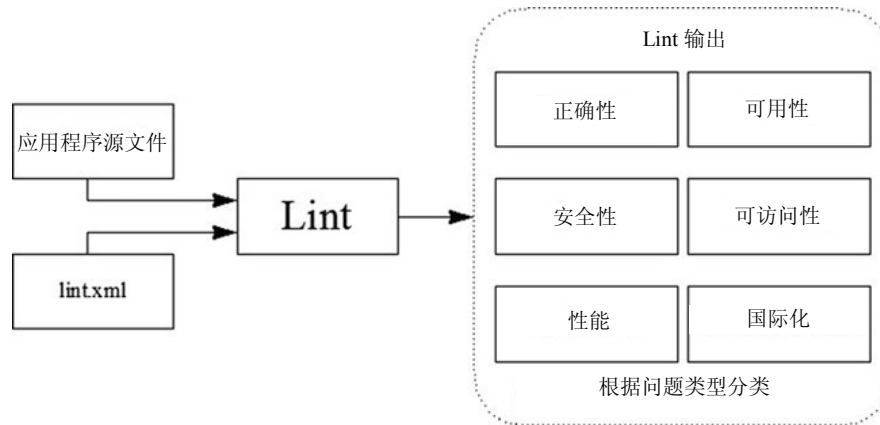


图 2-36 Lint 扫描代码的工作流

图 2-36 展示了以下内容：

- 应用程序源文件：源文件由组成 Android 项目的文件构成，包括 Java 和 XML 文件、图标和 ProGuard 配置文件。
- lint.xml：一个可以指定任何 Lint 检测项的配置文件，你可以排除特定检查项并可以自定义问题严重级别。
- Lint：一个静态代码扫描工具，可以使用命令行或者 Eclipse 运行在你的 Android 项目中。Lint 会检测影响 Android 应用程序质量和性能的代码结构问题。强烈建议在发布应用程序之前，请修正所有 Lint 检测到的错误。
- Lint Output：Lint 的结果可以在控制台或者在 Eclipse 中 Lint Warnings 视图中查看。每个问题都标识了在源文件中的位置和相应的问题描述。

2.44 运行 Lint

Lint 可以在命令行或者在 Eclipse 中运行。第一种方式需要在 lint(位于 Android SDK 根目录下的 tools 子目录中)后面指定一个项目目录。举个例子：假设当前目录包含 Univerter 项目目录，执行下面的命令：

```
lint Univerter
```

Lint 生成的输出信息的开头如下：

```
Scanning Univerter: .....
```

如果项目目录中没有包含 bin 子目录，你会看到下面的输出(为了方便阅读，分两行显示)：

```
Univerter: Error: No .class files were found in project "Univerter", so none
of the classfile based checks could be run. Does the project need to be
built first?
```

然后你会看到类似下面的输出内容(为了方便阅读,分两行显示):

```
res\layout-land\main.xml:9: Warning: This text field does not specify an
inputType or a hint [TextFields]
    <EditText android:id="@+id/display"
    ^

res\layout\main.xml:25: Warning: This text field does not specify an
inputType or a hint [TextFields]
    <EditText android:id="@+id/display"
    ^

res\layout\info.xml:7: Warning: [Accessibility] Missing contentDescription
attribute on image [ContentDescription]
    <ImageView android:id="@+id/image"
    ^

res\layout-land\main.xml:14: Warning: [I18N] Hardcoded string "0.", should
use @string resource [HardcodedText]
        android:text="0."
        ^

res\layout\main.xml:16: Warning: [I18N] Hardcoded string ">", should use
@string resource [HardcodedText]
        android:text=">"
        ^

res\layout\main.xml:30: Warning: [I18N] Hardcoded string "0.", should use
@string resource [HardcodedText]
        android:text="0."
        ^

0 errors, 6 warnings
```

这个输出显示了关于 Univerter 项目的六条警告。前两条是关于 `res/layout/main.xml` 和 `res/layout-land/main.xml` 文件中的 `<EditText>` 标签。这个标签缺少 `android:inputType` 或 `android:hint` 属性。

`Android:inputType` 属性标识输入到文本框中的数据类型,而 `android:hint` 属性则是在文本框内容为空时用于指定显示的提示文本。缺少前一个属性将不允许用户通过键盘向文本框中输入数据。缺少后者是因为文本框从不为空。

虽然没有这些属性也不会出现问题,但是建议最好在每个 `main.xml` 文件中 `<EditText>` 标签中包含 `android:inputType="text"`。就把它当做一个练习,做一下这个修改吧。

第三个警告表示 `res/layout/info.xml` 文件中的 `<ImageView>` 缺少了 `android:contentDescription` 属性。该属性简要地描述了非文本视图的内容(比如图像视图)。这个属性主要应用于易读性。考虑将该属性添加到 `<ImageView>` 中,比如 `android:contentDescription="@string/desc"`, `strings.xml` 中包含了一个名为 `desc` 的 `<string>` 元素。

最后三个警告表示在 `res/layout/main.xml` 和 `res/layout-land/main.xml` 文件中存在文本硬编码。“0.”和“>”文本在这里是硬编码(“0.”连同“%,8f”都直接用到了 `Univerter.java` 中),会导致应用程序很难本地化。

“0.” 是否需要本地化？这取决于你想支持多少本地化。例如，如果打算支持阿拉伯数字的输入，则需要将这个文本本地化。

“%,8f” 是否需要本地化？答案依然是取决于你想支持多少本地化。西班牙语将逗号作为小数点和千位分隔符。在这个案例中，你应该浏览一下 `java.util.DateFormat` 类，学习如何针对当前地区格式化字符串。

作为一个练习，可以将“0.”和“>”作为字面量定义在字符串资源文件 `res/values/strings.xml` 中，并在 `main.xml` 文件中添加对这些资源文件的引用。然后返回 Lint 去查看是否还会得到这些警告消息。

默认情况下，Lint 会搜索所有可能的问题类型。但是可以通过指定 `--check` 选项(后面为用逗号分隔的问题类型和 ID 的列表)来缩小搜索范围(执行“`lint --list`”来获取这个列表)。

例如，执行下面的命令可以检查 `contentDescription` 的遗漏：

```
lint -check contentDescription Univerter
```

这个命令的结果只有一个警告消息：

```
Scanning Univerter: .....
res\layout\info.xml:7: Warning: [Accessibility] Missing contentDescription
attribute on image [ContentDescription]
    <ImageView android:id="@+id/image"
    ^

0 errors, 1 warnings
```

现在执行下面的命令检查 `contentDescription` 的遗漏和文本硬编码：

```
lint -check contentDescription,HardcodedText Univerter
```

这个命令的结果有四条警告消息：

```
Scanning Univerter: .....
res\layout\info.xml:7: Warning: [Accessibility] Missing contentDescription
attribute on image [ContentDescription]
    <ImageView android:id="@+id/image"
    ^

res\layout-land\main.xml:14: Warning: [I18N] Hardcoded string "0.", should use
@string resource [HardcodedText]
        android:text="0."
        ^

res\layout\main.xml:16: Warning: [I18N] Hardcoded string ">", should use
@string resource [HardcodedText]
        android:text=">"
        ^

res\layout\main.xml:30: Warning: [I18N] Hardcoded string "0.", should use
@string resource [HardcodedText]
        android:text="0."
        ^

0 errors, 4 warnings
```

注意:

关于 Lint 的更多的信息, 可以访问 Google 的 “Improving Your Code with lint” 页面 (<http://developer.android.com/tools/debugging/improving-w-lint.html>)。也可以访问 Android Tools Project 站点上的 “Android Lint” 页面 (<http://tools.android.com/tips/lint/>)。

2.45 小结

从本章的内容可以看出, Android 在其 SDK 中提供了非常灵活和可扩展的 UI 工具。合理地使用这些工具可以确保应用程序在各种 Android 设备上的观感一致。

本章中, 我们探讨了如何使用 Android 的资源框架为多种设备提供资源。介绍的技术包括处理静态的图片以及创建自定义的 Drawable 资源。接下来讨论如何覆写窗口装饰和系统输入法的默认行为, 分析了通过动画视图添加用户值的方法。最后, 我们扩展了默认的工具集, 创建了自定义的新控件, 还自定义了 AdapterView 以显示数据集合。

第 3 章会介绍如何使用 SDK 跟外部世界进行沟通, 访问网络资源并与其他设备交互。

第 3 章

通信和联网

很多手机应用程序成功的关键是它们拥有与远程数据源进行连接和交互的能力。当今世界中，Web 服务和 API 已经非常丰富，从天气预报到个人财务信息，一个应用程序可以和任何其他服务进行交互。移动平台最大的优势就是可以将这些数据发送到用户的手中并且可在任何地方访问。Android 是在 Google 的 Web 基础上发展起来的，而 Google 则为与外部世界进行通信提供了丰富的工具集。

3.1 显示 Web 信息

3.1.1 问题

在应用程序中，需要将 Web 上获取的 HTML 或者图像数据不加任何修改和处理地显示出来。

3.1.2 解决方案

(API Level 1)

在 WebView 中显示信息。WebView 是一个视图控件，在应用程序中，它可以嵌入到任何布局中来显示本地或者远程的网页内容。WebView 基于开源的 WebKit 技术，而 Android 浏览器也是基于这项技术。所以二者赋予 Web 应用程序相同的性能。

3.1.3 实现机制

除了最重要的二维滚动(横向和纵向同时滚动)和变焦控制，WebView 对于显示从网上下载的资源，还有很多值得称道的地方。WebView 非常适合处理大图片，如体育场的地图，用户在浏览时可能需要进行左右平移和缩放。在这里，我们将讨论如何实现本地和远程资源的显示。

1. 显示一个 URL

最简单的情况就是提供资源的 URL，然后在 WebView 中将该 URL 对应的 HTML 页面或者图像显示出来。以下是这项技术在应用程序中一些小的实际应用：

- 在应用程序中访问你的企业网站
- 通过一个 Web 服务器显示实时更新的页面，如 FAQ，这个页面的内容不必升级应用程序就可以动态更新。
- 显示一个很大的图像资源，用户可能需要通过缩放来与它交互。

让我们来看一个加载常见网页的简单示例，不过不是用浏览器，而是在 Activity 内部加载(参见程序清单 3-1 和 3-2)。

程序清单 3-1 包含一个 WebView 的 Activity

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

        WebView webview = new WebView(this);
        //启用 JavaScript 支持
        webview.getSettings().setJavaScriptEnabled(true);
        webview.loadUrl("http://www.google.com/");
        setContentView(webview);
    }
}
```

注意：

默认情况下，WebView 是禁用 JavaScript 支持的。如果显示的内容需要它的话，可以使用 WebView.WebSettings 对象来启用它。

程序清单 3-2 在 AndroidManifest.xml 中设置需要的权限

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

重点:

如果 WebView 加载的是远程内容, AndroidManifest.xml 必须声明使用 android.permission.INTERNET 权限。

加载结果是在 Activity 中显示一个 HTML 页面(如图 3-1 所示)。

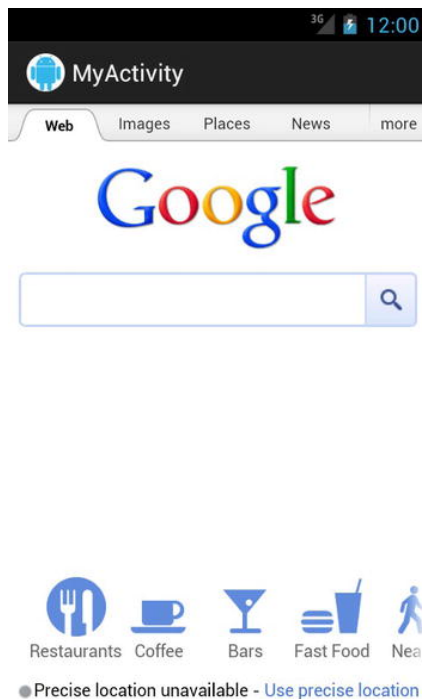


图 3-1 WebView 中的 HTML 页面

2. 本地资源

WebView 在显示本地内容时也非常有用, 它可以利用 HTML/CSS 来格式化内容或者为它的内容提供平移/缩放功能。你也许会使用 Android 项目的 assets 目录来存储你想在 WebView 中显示的资源, 如大型图片或者 HTML 文件。为了更好地组织资源, 也可以在 assets 下创建子目录来存储文件。

通过 `file:///android_asset/<resource path>` 这样的 URL 格式, `WebView.loadUrl()` 可以显示存储在 assets 中的文件。例如, 如果在 assets 目录中存放了文件 `android.jpg`, 使用 `file:///android_asset/android.jpg` 就可以将它加载到 WebView 中。

如果在 assets 下的 images 目录下存放了一个同样的文件, WebView 可以使用 `file:///android_asset/images/android.jpg` 来加载它。

另外, `WebView.loadData()` 可以将存储在 string 资源或者变量中的原始 HTML 代码加载到视图中。通过这项技术, 预先格式化的 HTML 文本可以保存在 `res/values/strings.xml` 中或使用远程 API 下载下来, 并显示在应用程序中。

下列程序清单 3-3 和 3-4 显示了一个示例 Activity, 它有两个 WebView, 其中一个垂直堆叠在另一个之上。上面的 view 显示了一张存储在 assets 目录中的大型图片, 下面的 view

则显示了存储在应用程序 `string` 资源中的一个 HTML 字符串。

程序清单 3-3 `res/layout/main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <WebView
        android:id="@+id/upperview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
    />
    <WebView
        android:id="@+id/lowerview"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1"
    />
</LinearLayout>
```

程序清单 3-4 显示本地 Web 内容的 Activity

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

        WebView upperView = (WebView)findViewById(R.id.upperview);
        //必须启用缩放功能
        upperView.getSettings().setBuiltInZoomControls(true);
        upperView.loadUrl("file:///android_asset/android.jpg");

        WebView lowerView = (WebView)findViewById(R.id.lowerview);
        String htmlString = "<h1>Header</h1><p>This is HTML text<br />"
            + "<i>Formatted in italics</i></p>";
        lowerView.loadData(htmlString, "text/html", "utf-8");
    }
}
```

当 Activity 显示时, 两个 `WebView` 将屏幕分为上下两个部分。HTML 字符串按预期的格式显示, 而大型图片则可以水平和垂直滚动, 用户甚至可以放大和缩小(如图 3-2 所示)。

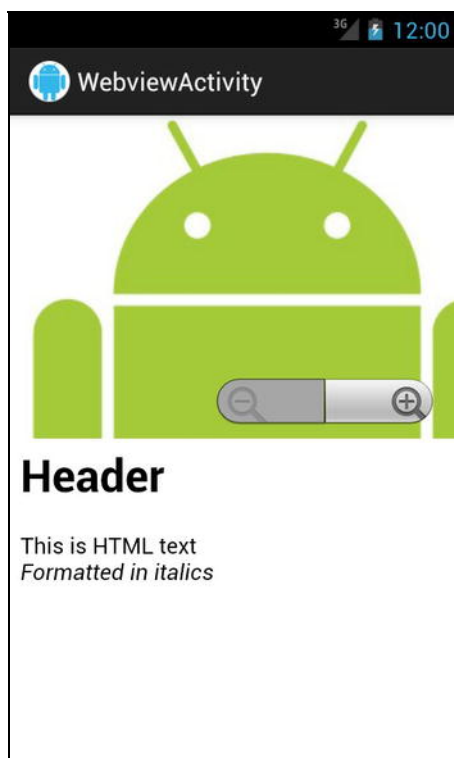


图 3-2 显示本地资源的两个 WebView

3.2 拦截 WebView 事件

3.2.1 问题

应用程序使用 WebView 显示内容，但在用户单击页面中的链接时还需要监听和响应。

3.2.2 解决方案

(API Level 1)

实现一个 `WebViewClient` 并把它关联到 `WebView` 上。`WebViewClient` 和 `WebChromeClient` 是两个 `WebKit` 类，它们可以让应用程序获得 `WebView` 的事件回调并且可以自定义 `WebView` 的行为。默认情况下，在没有指定 `WebViewClient` 时，`WebView` 会将一个 URL 传递给 `ActivityManager` 处理。而 `ActivityManager` 通常会在浏览器中打开用户单击的链接，而不是在当前的 `WebView` 中。

3.2.3 实现机制

在程序清单 3-5 中，我们创建了一个含有 `WebView` 的 `Activity`，该 `WebView` 将处理它自己的 URL 加载。

程序清单 3-5 带有一个 WebView 的 Activity, 该 WebView 会处理 URL

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

        WebView webview = new WebView(this);
        webview.getSettings().setJavaScriptEnabled(true);
        //添加一个客户端到视图上
        webview.setWebViewClient(new WebViewClient());
        webview.loadUrl("http://www.google.com");
        setContentView(webview);
    }
}
```

在本例中, 只是简单地为 **WebView** 提供了一个单纯功能的 **WebViewClient**, 它可以让 **WebView** 自己处理所有的 URL 请求, 而不是将这些请求传递给 **ActivityManager**, 因此单击一个链接会在原来的 **view** 中加载请求的页面, 这是因为 **shouldOverrideUrlLoading()** 的默认实现会简单地返回 **false**, 告诉客户端将 URL 传递给 **WebView** 而不是应用程序。

在下个示例中, 我们将利用 **WebViewClient.shouldOverrideUrlLoading()** 回调来拦截和监控用户的 **Activity**(见程序清单 3-6)。

程序清单 3-6 拦截 WebView URL 的 Activity

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

        WebView webview = new WebView(this);
        webview.getSettings().setJavaScriptEnabled(true);
        //添加一个客户端到视图上
        webview.setWebViewClient(mClient);
        webview.loadUrl("http://www.google.com");
        setContentView(webview);
    }

    private WebViewClient mClient = new WebViewClient() {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            Uri request = Uri.parse(url);

            if(TextUtils.equals(request.getAuthority(), "www.google.com")) {
                //允许加载
                return false;
            }

            Toast.makeText(MyActivity.this, "Sorry, buddy", Toast.LENGTH_SHORT)
        }
    }
}
```

```

        .show();
        return true;
    }
};
}

```

在本例中, `shouldOverrideUrlLoading()` 会根据传入的 URL 决定是否要在 `WebView` 中加载内容, 防止用户离开 Google 的网站。`Uri.getAuthority()` 会返回一个 URL 的主机域名部分, 我们会使用它检测用户单击的链接是否是 Google 的域名(`www.google.com`)。如果能够确认链接指向的是 Google 的其他页面, 会返回 `false`, 从而允许 `WebView` 加载内容。如果不是, 会返回 `true`, 并通知 `WebViewClient` 应用程序已经处理了这个 URL, 不允许 `WebView` 去加载它。

这项技术还能变得更加复杂, 应用程序可以对 URL 做各种处理。通过自定义的处理方式, 还可以在应用程序和 `WebView` 的内容之间打造一个完整的交互接口。

3.3 访问带 JavaScript 的 WebView

3.3.1 问题

应用程序需要访问 `WebView` 中的当前显示内容的 HTML 源代码, 读取或者修改其中的某个值。

3.3.2 解决方案

(API Level 1)

创建一个 JavaScript 接口来作为 `WebView` 和应用程序代码间的桥梁。

3.3.3 实现机制

`WebView.addJavascriptInterface()` 会为 JavaScript 绑定一个 Java 对象, 这样就可以在 `WebView` 中调用 Java 的方法。使用这个接口, 就可以使用 JavaScript 在应用程序代码和 `WebView` 的 HTML 间传递数据了。

警告:

允许 JavaScript 控制你的应用程序会存在安全威胁, 允许远程执行应用程序的代码。请在确实需要使用时再使用本接口。

让我们来实际看一个示例。程序清单 3-7 展示了 `WebView` 从本地 `assets` 目录中加载一个简单的 HTML 表单。程序清单 3-8 则是一个使用了两个 JavaScript 函数的 Activity, 这两个函数用来在 Activity preference 与 `WebView` 的内容间交换数据。

程序清单 3-7 assets/form.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<form name="input" action="form.html" method="get">
Enter Email: <input type="text" id="emailAddress" />
<input type="submit" value="Submit" />
</form>
</html>
```

程序清单 3-8 含有 JavaScript Bridge 接口的 Activity

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

        WebView webview = new WebView(this);
        webview.getSettings().setJavaScriptEnabled(true);
        webview.setWebViewClient(mClient);
        //将自定义接口关联到 view 上
        webview.addJavascriptInterface(new MyJavaScriptInterface(), "BRIDGE");

        setContentView(webview);
        //加载表单
        webview.loadUrl("file:///android_asset/form.html");
    }

    private static final String JS_SETELEMENT =
        "javascript:document.getElementById('%s').value='%s'";
    private static final String JS_GETELEMENT =
        "javascript:window.BRIDGE.storeElement('%s',document.getElementById('%s').value)";
    private static final String ELEMENTID = "emailAddress";

    private WebViewClient mClient = new WebViewClient() {
        @Override
        public boolean shouldOverrideUrlLoading(WebView view, String url) {
            //离开页面前, 尝试通过 JavaScript 获得 email
            view.loadUrl(String.format(JS_GETELEMENT, ELEMENTID, ELEMENTID));
            return false;
        }

        @Override
        public void onPageFinished(WebView view, String url) {
            //页面加载完成时, 使用 JavaScript 将地址注入页面中
            SharedPreferences prefs = getPreferences(Activity.MODE_PRIVATE);
            view.loadUrl(String.format(JS_SETELEMENT, ELEMENTID,
                prefs.getString(ELEMENTID, "")));
        }
    };
};
```

```

private class MyJavaScriptInterface {
    //将一个元素存储到 preference 中
    @SuppressWarnings("unused")
    public void storeElement(String id, String element) {
        SharedPreferences.Editor edit =
            getPreferences(Activity.MODE_PRIVATE).edit();
        edit.putString(id, element);
        edit.commit();
        //如果元素是有效的, 显示一个 Toast
        if(!TextUtils.isEmpty(element)) {
            Toast.makeText(MyActivity.this, element, Toast.LENGTH_SHORT)
                .show();
        }
    }
}

```

在这个稍微有点人为性质的示例中, HTML 中创建了一个单个元素的表单并显示一个 WebView。在 Activity 的代码中, 我们使用“emailAddress”这个 ID 在 WebView 中查找一个表单值, 并在页面中每次单击链接时(本例中, 就是单击表单的 submit 按钮), 通过 `shouldOverrideUrlLoading()` 把该值存储到 `SharedPreferences` 中。每次页面加载结束后(例如, 调用 `onPageFinished` 时), 我们会试图将当前的值从 `SharedPreferences` 再注入 Web 表单中。

我们创建了一个称作 `MyJavaScriptInterface` 的 Java 类, 它定义一个 `storeElement()` 方法。当创建了 View 后, 我们调用 `WebView.addJavascriptInterface()` 方法来把这个类的对象关联到 View 上, 并且给它起了个名字 `BRIDGE`。在 JavaScript 代码中, 可以使用这个名字来调用它的 `storeElement()` 方法。

在这里, 我们用字符串常量定义了两个 JavaScript 方法: `JS_GETELEMENT` 和 `JS_SETELEMENT`。通过把它们传递到 `loadUrl()` 中, 就可以在 WebView 中执行它们。注意, `JS_GETELEMENT` 是一个引用, 用来调用我们自定义的接口函数(如 `BRIDGE.storeElement`), 它将调用 `MyJavaScriptInterface` 的方法并把表单元素的值存储到 `preference` 中。如果检查到表单中的值不为空, 则会显示一个 Toast。

使用这种方式, 可以在 WebView 中执行任意的 JavaScript, 在自定义的接口中可以不需要包含任何方法。例如, `JS_SETELEMENT` 使用纯 JavaScript 来设置页面中表单元素的值。

经常使用这项技术的应用程序需要记住用户在应用程序中输入的表单数据, 但表单必须是基于 Web 的, 例如一个 Web 应用程序的预约表单或者支付表单, 而该 Web 应用程序需要较高 level 的 API 才可以访问。

3.4 下载一个图片文件

3.4.1 问题

你的应用程序需要在 Web 或者其他远程服务器上下载一张图片并显示。

3.4.2 解决方案

(API Level 3)

使用 `AsyncTask` 在后台线程中下载数据。`AsyncTask` 是个封装类，它可以很方便地让需要长时间运行操作的线程在后台运行；同样，它通过一个内部线程池管理线程的并发。除了管理后台线程外，在操作执行前、中、后都会提供回调方法，让你可以做任何需要在 UI 线程中进行的动作。

3.4.3 实现机制

在下载图片的内容中，我们会创建 `ImageView` 的一个子类，叫做 `WebImageView`，它会从远程资源中去延迟下载一张图片并且在该图片可用时就显示它。下载过程会在一个 `AsyncTask` 中执行(见程序清单 3-9)。

程序清单 3-9 `WebImageView`

```
public class WebImageView extends ImageView {

    private Drawable mPlaceholder, mImage;

    public WebImageView(Context context) {
        this(context, null);
    }

    public WebImageView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public WebImageView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
    }

    public void setPlaceholderImage(Drawable drawable) {
        mPlaceholder = drawable;
        if(mImage == null) {
            setImageDrawable(mPlaceholder);
        }
    }

    public void setPlaceholderImage(int resid) {
        mPlaceholder = getResources().getDrawable(resid);
        if(mImage == null) {
            setImageDrawable(mPlaceholder);
        }
    }

    public void setImageUrl(String url) {
```

```

        DownloadTask task = new DownloadTask();
        task.execute(url);
    }

    private class DownloadTask extends AsyncTask<String, Void, Bitmap> {
        @Override
        protected Bitmap doInBackground(String... params) {
            String url = params[0];
            try {
                URLConnection connection = (new URL(url)).openConnection();
                InputStream is = connection.getInputStream();
                BufferedInputStream bis = new BufferedInputStream(is);

                ByteBuffer baf = new ByteBuffer(50);
                int current = 0;
                while ((current = bis.read()) != -1) {
                    baf.append((byte)current);
                }
                byte[] imageData = baf.toByteArray();
                return BitmapFactory.decodeByteArray(imageData, 0,
                    imageData.length);
            } catch (Exception exc) {
                return null;
            }
        }

        @Override
        protected void onPostExecute(Bitmap result) {
            mImage = new BitmapDrawable(result);
            if(mImage != null) {
                setImageDrawable(mImage);
            }
        }
    };
}

```

正如你所看到的，WebImageView 是 Android 的 ImageView 控件的一个简单扩展。在远程内容下载完成之前，setPlaceholderImage()会为要显示的图片设置一个本地 drawable。大多数有趣的工作都是从 setImageUrl()传入一个给定远程 URL 开始的，该方法表示自定义的 AsyncTask 开始工作了。

请注意，AsyncTask 需要三个值：输入参数、进度值、结果值。在这种情况下，会传递给 task 的 execute 方法一个字符串，而后台操作应该返回一个 Bitmap。对于中间的进度值，我们在本例中并不会使用，因此它被设置为 Void。继承 AsyncTask 后，唯一需要实现的方法就是 doInBackground()，它定义了后台线程中需要大量运行的操作。在前面的示例中，这里是与提供的远程 URL 进行连接以及图片进行下载的地方。完成后，我们会试图用下载的数据创建一个 Bitmap。发生任何错误时，操作将中止并返回 null。

其他在 AsyncTask 中定义的回调方法，如 onPreExecute()、onPostExecute()和 onProgress-

Update(), 会在主线程中进行调用, 用来更新 UI。在之前的示例中, onPostExecute() 用来使用结果数据更新 View 中的图片。

重点:

Andorid UI 类是线程不安全的。确保在更新 UI 时使用运行在主线程上的回调方法。不要 doInBackground() 中更新 View。

程序清单 3-10 和 3-11 展示了在一个 Activity 中使用这个类的示例。因为这个类不是 android.widget 包或者 android.view 包的一部分, 所以在 XML 中使用它时不先指定它的全名。

程序清单 3-10 res/layout/main.xml

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

程序清单 3-11 示例 Activity

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

        WebImageView imageView = (WebImageView)findViewById(R.id.webImage);
        imageView.setPlaceholderImage(R.drawable.icon);
        imageView.setImageUrl(
            "http://apress.com/resource/weblogo/Apress_120x90.gif");
    }
}
```

本例中, 首先会设置一张本地图片(应用程序图标)作为 WebImageView 的占位图片, 这张图片会立刻显示给用户。然后我们会告诉 view 从 Web 上获取 Apress 的 logo 图片。正如前面所述, 这里会在后台下载图片, 下载完成后会替换掉 view 的占位图片。正是因为创建后台操作的简单性, Android 团队才会把 AsyncTask 叫做“无痛苦使用线程”。

3.5 完全在后台下载

3.5.1 问题

应用程序需要为设备下载一个大的资源，如电影文件，并且不要求用户让应用程序一直处于激活状态。

3.5.2 解决方案

(API Level 9)

使用 DownloadManager API。DownloadManager 是 API Level 9 中加入到 SDK 中的一个 service，它让系统完全去处理和管理需要长时间运行的下载操作。这个 service 最大的优点就是即使在下载失败、连接改变甚至设备重启时，DownloadManager 依然会继续尝试下载。

3.5.3 实现机制

程序清单 3-12 是一个示例 Activity，它使用 DownloadManager 来处理一个大图片文件的下载，这个图片会显示在一个 ImageView 上。在使用 DownloadManager 访问 Web 上的内容时，请确保在应用程序的 manifest 文件中声明了 android.permission.INTERNET 权限。

程序清单 3-12 DownloadManager 示例 Activity

```
public class DownloadActivity extends Activity {

    private static final String DL_ID = "downloadId";
    private SharedPreferences prefs;

    private DownloadManager dm;
    private ImageView imageView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        imageView = new ImageView(this);
        setContentView(imageView);

        prefs = PreferenceManager.getDefaultSharedPreferences(this);
        dm = (DownloadManager) getSystemService(DOWNLOAD_SERVICE);
    }

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

```

        if(!prefs.contains(DL_ID)) {
            //开始下载
            Uri resource = Uri.parse("http://www.bigfoto.com/dog-animal.jpg");
            DownloadManager.Request request =
                new DownloadManager.Request(resource);
            //设置允许的连接来处理下载
            request.setAllowedNetworkTypes(Request.NETWORK_MOBILE |
                Request.NETWORK_WIFI);
            request.setAllowedOverRoaming(false);
            //在通知栏上显示
            request.setTitle("Download Sample");
            long id = dm.enqueue(request);
            //保存唯一的 id
            prefs.edit().putLong(DL_ID, id).commit();
        } else {
            //下载已经开始, 检查下载状态
            queryDownloadStatus();
        }

        registerReceiver(receiver,
            new IntentFilter(DownloadManager.ACTION_DOWNLOAD_COMPLETE));
    }

    @Override
    public void onPause() {
        super.onPause();
        unregisterReceiver(receiver);
    }

    private BroadcastReceiver receiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            queryDownloadStatus();
        }
    };

    private void queryDownloadStatus() {
        DownloadManager.Query query = new DownloadManager.Query();
        query.setFilterById(prefs.getLong(DL_ID, 0));
        Cursor c = dm.query(query);
        if(c.moveToFirst()) {
            int status =
                c.getInt(c.getColumnIndex(DownloadManager.COLUMN_STATUS));
            switch(status) {
                case DownloadManager.STATUS_PAUSED:
                case DownloadManager.STATUS_PENDING:
                case DownloadManager.STATUS_RUNNING:
                    //什么也不做, 依然进行中
                    break;
                case DownloadManager.STATUS_SUCCESSFUL:

```

```

//下载完成，显示图片
try {
    ParcelFileDescriptor file =
        dm.openDownloadedFile(prefs.getLong(DL_ID, 0));
    FileInputStream fis =
        new ParcelFileDescriptor.AutoCloseInputStream(file);
    imageView.setImageBitmap(BitmapFactory.decodeStream(fis));
} catch (Exception e) {
    e.printStackTrace();
}
break;
case DownloadManager.STATUS_FAILED:
    //清除下载并稍后重试
    dm.remove(prefs.getLong(DL_ID, 0));
    prefs.edit().clear().commit();
    break;
}
}
}
}

```

重点:

本书出版时, SDK 中存在一个 bug, 就是在使用 `DownloadManager` 时抛出 `android.permission.ACCESS_ALL_DOWNLOADS` 的异常。实际上是在 `manifest` 中没有声明 `android.permission.INTERNET` 时才会抛出这个异常。

示例中所有有用的工作都是在 `Activity.onResume()` 中完成的, 这样每次用户返回到 `Activity` 时, 应用程序都可以检查下载的状态。每次下载会调用 `DownloadManager.enqueue()` 并返回一个 `long` 型的 ID 值, 该值可以用来引用本次下载。在本例中, 为了随时监控和检查下载的内容, 我们会把这个值保存到应用程序的 `preference` 中。

当示例应用程序第一次启动时, 会创建一个 `DownloadManager.Request` 请求对象, 它代表了下载的内容。这个请求至少要指定远程资源的 `Uri`。另外, 还可以为这个请求设置很多有用的属性来控制它的行为。这些有用的属性包括:

- `Request.setAllowedNetworkTypes()`: 指定下载所使用的网络类型
- `Request.setAllowedOverRoaming()`: 设定当设备处于漫游模式时是否要下载。
- `Request.setDescription()`: 设置下载在系统通知栏中显示的描述。
- `Request.setTitle()`: 设置下载在系统通知栏中显示的标题。

当获取 ID 后, 应用程序会使用那个值来检查下载的状态。通过注册 `BroadcastReceiver` 来监听 `ACTION_DOWNLOAD_COMPLETE` 广播, 在下载完成后, 会将图片文件设置到 `Activity` 的 `ImageView`。如果在下载完成时 `Activity` 处于暂停状态, 那么就会在下次恢复 `Activity` 时检查下载状态, 并设置 `ImageView` 的内容。

要注意 `ACTION_DOWNLOAD_COMPLETE` 是 `DownloadManager` 对其管理的所有下载任务发出的广播。正因为如此, 我们必须要根据下载 ID 才能判断我们的下载是否完成。

1. 目标

在程序清单 3-12 中，我们并没有告诉 `DownloadManager` 文件的下载位置。相反，当我们想要访问文件时，我们会使用 `DownloadManager.openDownloadedFile()` 方法和 `preference` 中储存的 ID 值来获得一个 `ParcelFileDescriptor`，它可以转换为应用程序可以读取的数据流。这是访问下载内容简单而直接的方式，但还是有一些事项需要注意。

如果没有指定的目标位置，文件都会下载到共享下载缓存中，系统随时有权删除这个缓存中的文件来回收空间。正因为如此，这种下载方式可以很方便地快速获取数据，但如果是下载需求是长期的，应该使用 `DownloadManager.Request` 的方法在外部存储器上指定一个固定的目标位置：

- `Request.setDestinationInExternalFilesDir()`：设置目标位置为外部存储器中的一个隐藏目录
- `Request.setDestinationInExternalPublicDir()`：设置目标位置为外部存储器中的一个公共目录
- `Request.setDestinationUri()`：设置目标位置为位于外部存储器中一个文件 Uri

注意：

所有在外部存储器中设置目标位置的方法都需要在 `manifest` 中声明 `android.permission.WRITE_EXTERNAL_STORAGE` 权限。

在调用 `DownloadManager.remove()` 来清理管理列表中的条目或者用户清理下载列表时，没有明确指定目标位置的文件通常会被移除；而外部存储器中下载的文件在这些条件下则不会被移除。

3.6 访问 REST API

3.6.1 问题

应用程序需要通过 HTTP 访问 RESTful API，实现与远程主机上的 Web 服务交互。

注意：

REST(Representational State Transfer, 表述性状态转移)，这是一种现在常见的 Web 服务架构风格。RESTful API 通常用于构建标准的 HTTP 动作，以创建对远程资源的请求。返回的通常是标准文档格式，例如 XML、JSON 或逗号分隔值(CSV)。

3.6.2 解决方案

基于 Android 网络连接来使用 HTTP 发送和接收数据时，有两种推荐的方式：第一种是 Apache 的 `HttpClient`，第二种是 Java 的 `HttpURLConnection`。具体在应用程序中选择哪种方式主要取决于你想支持的 Android 版本。

(API Level 3)

如果想支持稍早的 Android 版本,可以在 AsyncTask 中使用 Apache 的 HTTP 类。Android 已经包含了 Apache 的 HTTP 组件库,该组件库提供了一种与远程 API 进行连接的可靠方式。Apache 类库可以很容易地创建 GET、POST、PUT 和 DELETE 请求,另外还支持 SSL、cookie 存储、认证以及其他可能在 API 的 HttpClient 参数中会用到的 HTTP 需求。

这种方式的另一个主要的优点是 Apache 库的高度封装。很多情况下,应用程序只需要很少的代码就可以进行大多数的 HTTP 操作。很多底层传输的代码对开发者来说都是隐藏的。

最大的缺点就是 Android 所绑定的 Apache 组件版本并没有包含 MultipartEntity 类,该类对于二进制或者复合表单数据的 POST 传输非常重要。因此,如果需要这个功能还想使用 HttpClient 的话,必须引入新版本的组件库,作为外部 JAR 包。

(API Level 9)

可以在 AsyncTask 中使用 HttpURLConnection 类。这个类在 API Level 1 时就已经是 Android 框架的一部分,但在 Android 2.3 之前,它只推荐用于进行网络的 I/O 操作。主要的原因就是刚开始这个类的实现有很多的 bug,所以选择使用 HttpClient 会更加稳妥。后来,Android 团队对 HttpURLConnection 的性能和稳定性做了很大的提高,现在则成为一种推荐使用的方式。

HttpURLConnection 最大的优势就是它的性能。这些类都是轻量级的,在新版本的 Android 中响应速度更快并且内置很多增强功能。它的 API 更加开放所以也就更加普及,可以实现任何类型的 HTTP 传输。缺点是开发者需要编写更多的代码(但它不是你不买本书的理由?)。

3.6.3 实现机制

1. HttpClient

接下来,看一下通过 Apache HttpClient 使用 HTTP。程序清单 3-13 是一个 AsyncTask,它可以处理任何 HttpRequest 并返回反馈的字符串。

程序清单 3-13 处理 HttpRequest 的 AsyncTask

```
public class RestTask extends AsyncTask<HttpRequest, Void, Object> {
    private static final String TAG = "RestTask";

    public interface ResponseCallback {
        public void onRequestSuccess(String response);
        public void onRequestError(Exception error);
    }

    private AbstractHttpClient mClient;

    private WeakReference<ResponseCallback> mCallback;
```

```

public RestTask() {
    this(new DefaultHttpClient());
}

public RestTask(AbstractHttpClient client) {
    mClient = client;
}

public void setResponseCallback(ResponseCallback callback) {
    mCallback = new WeakReference<ResponseCallback>(callback);
}

@Override
protected Object doInBackground(HttpUriRequest... params) {
    try{
        HttpUriRequest request = params[0];
        HttpResponse serverResponse = mClient.execute(request);

        BasicResponseHandler handler = new BasicResponseHandler();
        String response = handler.handleResponse(serverResponse);
        return response;
    } catch (Exception e) {
        Log.w(TAG, e);
        return e;
    }
}

@Override
protected void onPostExecute(Object result) {
    if (mCallback != null && mCallback.get() != null) {
        if (result instanceof String) {
            mCallback.get().onRequestSuccess((String) result);
        } else if (result instanceof Exception) {
            mCallback.get().onRequestError((Exception) result);
        } else {
            mCallback.get().onRequestError(
                new IOException("Unknown Error Contacting Host"));
        }
    }
}
}

```

无论是否有 `HttpClient` 参数，我们都可以构建 `RestTask`。之所以这么做是因为多个请求可以共用一个客户端对象。如果你的 API 需要通过 `cookie` 维持回话。或是有一些必要的参数可以一次性完成设置(例如 `SSL` 存储)，这个功能就十分有用了。`Task` 需要处理 `HttpUriRequest` 参数(`HttpGet`、`HttpPost`、`HttpPut` 和 `HttpDelete` 都是它的子类)，然后执行该请求。

`BasicResponseHandler` 处理响应，这个类将处理响应的工作从检查各种错误的繁琐任务中抽象出来。如果响应代码是 `1XX` 或 `2XX`，`BasicResponseHandler` 会以字符串的形式返回 HTTP 响应；如果响应代码是 `300` 或者更高，`BasicResponseHandler` 就会抛出 `HttpResponse-`

Exception 异常。

这个类最后比较重要的部分在 `onPostExecute()` 方法中，即与 API 交互完成之后。`RestTask` 有一个可选的回调接口，当请求完成(响应数据为一个字符串)或者发生错误(会触发一个异常)时会通知这个接口。这个回调方法是以 `WeakReference` 的形式存储的，这样我们就可以放心地使用一个 `Activity` 或者系统组件作为回调，而不用担心当组件处于暂停或者停止状态时，运行着的 `task` 会阻止组件的回收。现在用这个强大的新工具创建几个基本的 API 请求。

GET 示例

在下面这个示例中我们会利用 Google 自定义搜索 REST API。这个 API 的请求需要以下几个参数：

- **key**: 唯一值用于标识发出请求的应用程序
- **cx**: 标识你想要访问的自定义搜索引擎
- **q**: 一个字符串，代表了你想要执行的搜索查询

关于这个 API 的更多信息可以访问 <https://developers.google.com/custom-search/>。

在很多公共 API 中，GET 请求是最简单，也是最常见的请求。参数必须与请求一起编码到 URL 字符串中发送，所以不需要提供其他的数据。下面将创建 GET 请求，搜索“Android”（参见程序清单 3-14）。

程序清单 3-14 执行 API GET 请求的 Activity

```
public class SearchActivity extends Activity implements ResponseCallback {

    private static final String SEARCH_URI =
        "https://www.googleapis.com/customsearch/v1?key=%s&cx=%s&q=%s";
    private static final String SEARCH_KEY =
        "AIzaSyBbW-WlSHCK4eW0kK74VGMLJj_b-byNzkI";
    private static final String SEARCH_CX =
        "008212991319514020231:lmkouq8yagw";
    private static final String SEARCH_QUERY = "Android";

    private TextView mResult;
    private ProgressDialog mProgress;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ScrollView scrollView = new ScrollView(this);
        mResult = new TextView(this);
        scrollView.addView(mResult,
            new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        setContentView(scrollView);
        try{
            //简单的 GET 请求
            String url = String.format(SEARCH_URI, SEARCH_KEY,
```

```

        SEARCH_CX, SEARCH_QUERY);
        HttpGet searchRequest = new HttpGet(url);

        RestTask task = new RestTask();
        task.setResponseCallback(this);
        task.execute(searchRequest);

        //向用户显示进度
        mProgress = ProgressDialog.show(this, "Searching",
            "Waiting For Results...", true);
    } catch (Exception e) {
        mResult.setText(e.getMessage());
    }
}

@Override
public void onRequestSuccess(String response) {
    //结束进度条
    if(mProgress != null) {
        mProgress.dismiss();
    }

    //处理返回的数据(这里只是把结果显示出来)
    mResult.setText(response);
}

@Override
public void onRequestError(Exception error) {
    //结束进度条
    if(mProgress != null) {
        mProgress.dismiss();
    }

    //处理返回的数据(这里只是把结果显示出来)
    mResult.setText(error.getMessage());
}
}

```

在这个示例中所创建的这种 HTTP 请求需要所需连接的 URL(这里,就是发送到 `googleapis.com` 的 GET 请求)。URL 保存为格式化字符串常量, Google API 所需的参数则在运行时创建请求之前加上。

Activity 中创建了一个 `RestTask` 并执行,并把该 Activity 作为 `RestTask` 的回调。当 task 完成后,会调用 `onRequestSuccess()`或者 `onRequestError()`,在成功的情况下,会解析 API 所返回的数据并处理它。我们会在 3.7 节和 3.8 节介绍如何解析结构化的 XML 或 JSON 反馈数据,目前只是将原始的反馈信息呈现到用户界面上。

POST 示例

很多时候, API 要求在请求中提供所需的数据,可能是认证令牌,也有可能是搜索查

询。API 会要求你通过 HTTP POST 发送请求，这样这些数据就会被编码到请求中，而不是被编码到 URL 中。为了演示 POST 是如何工作的，我们将向 httpbin.org 发送请求，它是一个开发网站，用来读取和验证请求的内容并将它们返回(参见程序清单 3-15)。

程序清单 3-15 执行 API POST 请求的 Activity

```
public class SearchActivity extends Activity implements ResponseCallback {

    private static final String POST_URI = "http://httpbin.org/post";

    private TextView mResult;
    private ProgressDialog mProgress;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ScrollView scrollView = new ScrollView(this);
        mResult = new TextView(this);
        scrollView.addView(mResult,
            new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        setContentView(scrollView);

        try{
            //简单的 POST 请求
            HttpPost postRequest = new HttpPost( new URI(POST_URI) );
            List<NameValuePair> parameters = new ArrayList<NameValuePair>();
            parameters.add(new BasicNameValuePair("title", "Android Recipes"));
            parameters.add(new BasicNameValuePair("summary",
                "Learn Android Quickly"));
            parameters.add(new BasicNameValuePair("authors", "Smith/Friesen"));
            postRequest.setEntity(new UrlEncodedFormEntity(parameters));

            RestTask task = new RestTask();
            task.setResponseCallback(this);
            task.execute(postRequest);
            //向用户显示进度
            mProgress = ProgressDialog.show(this, "Searching",
                "Waiting For Results...", true);
        } catch (Exception e) {
            mResult.setText(e.getMessage());
        }
    }

    @Override
    public void onRequestSuccess(String response) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }
    }
}
```

```

        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText(response);
    }

    @Override
    public void onRequestError(Exception error) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }

        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText(error.getMessage());
    }
}

```

注意,在这个示例中,传递给 API 执行搜索所必需的参数被编码到 `HttpEntity` 中,而不是直接传递给请求 URL。这里创建的请求是一个 `HttpPost` 实例,同时也是 `HttpRequest` 的子类(跟 `HttpGet` 一样),所以可以使用同样的 `RestTask` 来执行这个操作。和 GET 示例一样,我们会在 3.7 节和 3.8 节介绍如何解析结构化的 XML 或 JSON 反馈数据,目前只是将原始的反馈信息呈现到用户界面上。

提醒:

Android SDK 中捆绑的 Apache 库并没有提供对 `Multipart HTTP POST` 的支持。不过, `org.apache.http.mime` 库中的 `MultipartEntity` 是与 Android SDK 兼容的,可以作为外部源代码添加到项目中。

基本认证

使用 API 时,另外一个常见的需求就是要处理各种认证。`REST_API` 认证已经有了一些标准(例如 `Oauth 2.0`),不过最常见的认证方式还是 HTTP 上基本的用户名和密码认证。在程序清单 3-16 中,我们修改了 `RestTask`,在每个请求的 HTTP 头部中启用了基本认证。

程序清单 3-16 带有基本认证的 `RestTask`

```

public class RestAuthTask extends AsyncTask<HttpRequest, Void, Object> {
    private static final String TAG = "RestTask";

    private static final String AUTH_USER = "user@mydomain.com";
    private static final String AUTH_PASS = "password";

    public interface ResponseCallback {
        public void onRequestSuccess(String response);

        public void onRequestError(Exception error);
    }

    private AbstractHttpClient mClient;
}

```

```

private WeakReference<ResponseCallback> mCallback;

public RestAuthTask(boolean authenticate) {
    this(new DefaultHttpClient(), authenticate);
}

public RestAuthTask(AbstractHttpClient client, boolean authenticate) {
    mClient = client;
    if(authenticate) {
        UsernamePasswordCredentials creds =
            new UsernamePasswordCredentials(AUTH_USER, AUTH_PASS);
        mClient.getCredentialsProvider()
            .setCredentials(AuthScope.ANY, creds);
    }
}

@Override
protected Object doInBackground(HttpUriRequest... params) {
    try{
        HttpUriRequest request = params[0];
        HttpResponse serverResponse = mClient.execute(request);

        BasicResponseHandler handler = new BasicResponseHandler();
        String response = handler.handleResponse(serverResponse);
        return response;

    } catch (Exception e) {
        Log.w(TAG, e);
        return e;
    }
}

@Override
protected void onPostExecute(Object result) {
    if (mCallback != null && mCallback.get() != null) {
        if (result instanceof String) {
            mCallback.get().onRequestSuccess((String) result);
        } else if (result instanceof Exception) {
            mCallback.get().onRequestError((Exception) result);
        } else {
            mCallback.get().onRequestError(
                new IOException("Unknown Error Contacting Host"));
        }
    }
}
}

```

这样，基本认证就被添加到了 Apache 的 `HttpClient` 参数中了。因为示例允许传递特定的客户端对象，这其中已经包含了必要的认证信息，所以这里修改的是使用默认客户端的

示例。在这个示例中，我们用用户名和密码字符串创建了 UsernamePasswordCredentials 实例，然后将其设置为客户端的 CredentialsProvider。

2. HttpURLConnection

让我们看一下在相对较新的应用程序中所推荐使用的发送 HTTP 请求的方法 HttpURLConnection。首先，在程序清单 3-17 中定义一个相同的 RestTask 实现，并在程序清单 3-18 中定义了一个辅助类。

程序清单 3-17 使用 HttpURLConnection 的 RestTask

```
public class RestTask extends AsyncTask<Void, Integer, Object> {
    private static final String TAG = "RestTask";

    public interface ResponseCallback {
        public void onRequestSuccess(String response);

        public void onRequestError(Exception error);
    }

    public interface ProgressCallback {
        public void onProgressUpdate(int progress);
    }

    private HttpURLConnection mConnection;
    private String mFormBody;
    private File mUploadFile;
    private String mUploadFileName;

    //Activity 回调，使用 WeakReferences 来避免阻塞操作导致无法回收内存
    private WeakReference<ResponseCallback> mResponseCallback;
    private WeakReference<ProgressCallback> mProgressCallback;

    public RestTask(HttpURLConnection connection) {
        mConnection = connection;
    }

    public void setFormBody(List<NameValuePair> formData) {
        if (formData == null) {
            mFormBody = null;
            return;
        }

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < formData.size(); i++) {
            NameValuePair item = formData.get(i);
            sb.append( URLEncoder.encode(item.getName()) );
            sb.append("=");
            sb.append( URLEncoder.encode(item.getValue()) );
            if (i != (formData.size() - 1)) {
```

```

        sb.append("&");
    }
}

mFormBody = sb.toString();
}

public void setUploadFile(File file, String fileName) {
    mUploadFile = file;
    mUploadFileName = fileName;
}

public void setResponseCallback(ResponseCallback callback) {
    mResponseCallback = new WeakReference<ResponseCallback>(callback);
}

public void setProgressCallback(ProgressCallback callback) {
    mProgressCallback = new WeakReference<ProgressCallback>(callback);
}

private void writeMultipart(String boundary, String charset,
    OutputStream output, boolean writeContent) throws IOException {
    BufferedWriter writer = null;
    try {
        writer = new BufferedWriter(new OutputStreamWriter(output,
            Charset.forName(charset)), 8192);
        //发送表单数据组件
        if (mFormBody != null) {
            writer.write("--" + boundary);
            writer.write("\r\n");
            writer.write(
                "Content-Disposition: form-data; name=\""parameters\"");
            writer.write("\r\n");
            writer.write("Content-Type: text/plain; charset=" + charset);
            writer.write("\r\n");
            writer.write("\r\n");
            if (writeContent) {
                writer.write(mFormBody);
            }
            writer.write("\r\n");
            writer.flush();
        }
        //发送二进制文件
        writer.write("--" + boundary);
        writer.write("\r\n");
        writer.write("Content-Disposition: form-data; name=\""
            + mUploadFileName + "\"; filename=\""
            + mUploadFile.getName() + "\"");
        writer.write("\r\n");
        writer.write("Content-Type: "

```

```

        + URLConnection.guessContentTypeFromName(
        mUploadFile.getName()));
writer.write("\r\n");
writer.write("Content-Transfer-Encoding: binary");
writer.write("\r\n");
writer.write("\r\n");
writer.flush();
if (writeContent) {
    InputStream input = null;
    try {
        input = new FileInputStream(mUploadFile);
        byte[] buffer = new byte[1024];
        for (int length = 0; (length = input.read(buffer)) > 0;) {
            output.write(buffer, 0, length);
        }
        //不要关闭 OutputStream
        output.flush();
    } catch (IOException e) {
        Log.w(TAG, e);
    } finally {
        if (input != null) {
            try {
                input.close();
            } catch (IOException e) {
            }
        }
    }
}
// 这个回车换行标志着二进制数据块的结束
writer.write("\r\n");
writer.flush();

// multipart/form-data 的结束
writer.write("--" + boundary + "--");
writer.write("\r\n");
writer.flush();
} finally {
    if (writer != null) {
        writer.close();
    }
}
}

private void writeFormData(String charset, OutputStream output)
    throws IOException {
    try {
        output.write(mFormBody.getBytes(charset));
        output.flush();
    } finally {
        if (output != null) {

```

```

        output.close();
    }
}

@Override
protected Object doInBackground(Void... params) {
    // 生成用来标识界限的随机字符串
    String boundary = Long.toHexString(System.currentTimeMillis());
    String charset = Charset.defaultCharset().displayName();

    try {
        // 如果可以的话, 创建输出流
        if (mUploadFile != null) {
            // 我们必须做一个复合请求
            mConnection.setRequestProperty("Content-Type",
                "multipart/form-data; boundary=" + boundary);

            // 计算 extra 元数据的大小
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            writeMultipart(boundary, charset, bos, false);
            byte[] extra = bos.toByteArray();
            int contentLength = extra.length;
            // 将文件的大小加到 length 上
            contentLength += mUploadFile.length();
            // 如果存在表单体, 把它也加到 length 上
            if (mFormBody != null) {
                contentLength += mFormBody.length();
            }

            mConnection.setFixedLengthStreamingMode(contentLength);
        } else if (mFormBody != null) {
            // 这种情况下, 只是发送表单数据
            mConnection.setRequestProperty("Content-Type",
                "application/x-www-form-urlencoded; charset=" + charset);
            mConnection.setFixedLengthStreamingMode(mFormBody.length());
        }

        // 这是第一次调用 URLConnection, 它会真正执行网络 IO 操作。
        // openConnection() 执行的还是本地操作。
        mConnection.connect();

        // 如果可以的话 (对于一个 POST), 创建输出流
        if (mUploadFile != null) {
            OutputStream out = mConnection.getOutputStream();
            writeMultipart(boundary, charset, out, true);
        } else if (mFormBody != null) {
            OutputStream out = mConnection.getOutputStream();
            writeFormData(charset, out);
        }
    }
}

```

```

        // 获取响应数据
        int status = mConnection.getResponseCode();
        if (status >= 300) {
            String message = mConnection.getResponseMessage();
            return new HttpResponseException(status, message);
        }

        InputStream in = mConnection.getInputStream();
        String encoding = mConnection.getContentEncoding();
        int contentLength = mConnection.getContentLength();
        if (encoding == null) {
            encoding = "UTF-8";
        }
        BufferedReader reader = new BufferedReader(new InputStreamReader(
            in, encoding));
        char[] buffer = new char[4096];

        StringBuilder sb = new StringBuilder();
        int downloadedBytes = 0;
        int len1 = 0;
        while ((len1 = reader.read(buffer)) > 0) {
            downloadedBytes += len1;
            publishProgress((downloadedBytes * 100) / contentLength);
            sb.append(buffer);
        }

        return sb.toString();
    } catch (Exception e) {
        Log.w(TAG, e);
        return e;
    } finally {
        if (mConnection != null) {
            mConnection.disconnect();
        }
    }
}

@Override
protected void onProgressUpdate(Integer... values) {
    // 更新进度 UI
    if (mProgressCallback != null && mProgressCallback.get() != null) {
        mProgressCallback.get().onProgressUpdate(values[0]);
    }
}

@Override
protected void onPostExecute(Object result) {
    if (mResponseCallback != null && mResponseCallback.get() != null) {
        if (result instanceof String) {

```

```

        mResponseCallback.get().onRequestSuccess((String) result);
    } else if (result instanceof Exception) {
        mResponseCallback.get().onRequestError((Exception) result);
    } else {
        mResponseCallback.get().onRequestError(
            new IOException("Unknown Error Contacting Host"));
    }
}
}
}
}

```

程序清单 3-18 创建请求的工具类

```

public class RestUtil {

    public static final RestTask obtainGetTask(String url)
        throws MalformedURLException, IOException {
        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();
        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
        connection.setDoInput(true);

        RestTask task = new RestTask(connection);
        return task;
    }

    public static final RestTask obtainFormPostTask(String url,
        List<NameValuePair> formData) throws MalformedURLException,
        IOException {
        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();

        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
        connection.setDoOutput(true);

        RestTask task = new RestTask(connection);
        task.setFormBody(formData);

        return task;
    }

    public static final RestTask obtainMultipartPostTask(String url,
        List<NameValuePair> formPart, File file, String fileName)
        throws MalformedURLException, IOException {
        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();

        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
    }
}

```

```

        connection.setDoOutput(true);

        RestTask task = new RestTask(connection);
        task.setFormBody(formPart);
        task.setUploadFile(file, fileName);

        return task;
    }
}

```

第一件可能需要注意的事情就是这个示例在实现特定的请求时需要更多的代码，这主要是因为 API 更加底层。我们写的这个 `RestTask` 可以处理 GET、简单 POST 和复合 POST 请求，并根据添加到 `RestTask` 中的组件，动态地定义请求的参数。

与以前一样，当请求完成时，我们可以关联一个可选的通知回调。但除此之外，我们还关联了一个进度回调接口，它会在下载反馈内容时更新所有可见的进度 UI。使用这个 API 会更容易实现它，原因就是我们会直接与数据流进行交互。

本例中，应用程序将使用 `RestUtil` 辅助类创建一个 `RestTask` 的实例。该类细化了 `URLConnection` 的创建过程，它并不会进行任何的网络 I/O 操作，包括网络连接部分和与主机交互部分。辅助类创建了连接实例并且设置了超时值和 HTTP 请求类型。

注意：

默认情况下，所有 `URLConnection` 的请求类型会被设置为 GET。调用 `setDoOutput()` 则会隐式地把请求类型设置为 POST。如果需要设置为其他的 HTTP 类型，则需要使用 `setRequestMethod()`。

POST 情况下，如果有表单体内容，这些值会直接设置到我们自定义的 `task` 中，当 `task` 执行时，回去读取它们。

当 `RestTask` 执行后，它会检查是否有关联的 `body` 数据需要写入。如果我们已经关联了表单数据(以 `name-value` 对的方式)或者需要上传的文件，就会把它作为一个触发器来构造一个 POST `body` 并发送。使用 `URLConnection`，我们需要处理连接的各种事宜，包括告诉服务器收到数据的数量。`RestTask` 会花些时间计算即将发送多少数据以及通过调用 `setFixedLengthStreamingMode()` 构造一个头字段来告诉服务器我们内容的大小。对于简单表单 Post 请求的情况，这种计算则很简单，我们只需要传入 `body` 字符串的长度即可。

但是，复合 POST 包含的文件数据可能会复杂一些。复合 POST 在 `body` 中有很多的附加数据用于指定 POST 中各个部分之间的分界线，而且这些数据也会算到我们所设置的长度中。想要完成这个目标，`writeMultipart()` 在构造时可以传入一个本地 `OutputStream` (本例中为一个 `ByteArrayOutputStream`)，然后将所有的附加数据写入到这个 `OutputStream` 中，这样我们就可以衡量附加数据的大小了。当 `writeMultipart()` 通过这种方式调用时，会忽略真正的内容，例如文件和表单数据，这些内容稍后可以通过调用它们各自的 `length()` 方法添加，而且我们也不想浪费时间去将它们加载到内存。

注意：

如果不知道要 POST 的内容有多大，`URLConnection` 还通过 `setChunkedStreamingMode()`

方法提供了块上传的机制。本例中，你只需要传入即将发送的数据块的大小即可。

当 task 已经向主机中写入了任何的 POST 数据后，就可以读取响应内容了。如果初始请求是一个 GET 请求，由于并没有其他数据需要写入，task 可以直接忽略掉这一步。task 首先会检查响应代码的值保证没有服务器端的错误，之后将响应的内容下载到一个 StringBuilder 中。下载时每次大概会读取 4KB 的数据块，同时将下载数据与响应内容总长度的百分比通知给进度回调处理器。当所有的内容都下载完成后，task 会将所有结果响应数据以字符串的形式返回。

GET 示例

让我们看一下相同的 Google 自定义搜索示例，但这一次我们使用新的改进过的 RestTask(参见程序清单 3-19)。

程序清单 3-19 只需 API GET 请求的 Activity

```
public class SearchActivity extends Activity implements
    RestTask.ProgressCallback, RestTask.ResponseCallback {

    private static final String SEARCH_URI =
        "https://www.googleapis.com/customsearch/v1?key=%s&cx=%s&q=%s";
    private static final String SEARCH_KEY =
        "AIzaSyBbW-WlSHCK4eW0kK74VGMLJj_b-byNzkI";
    private static final String SEARCH_CX =
        "008212991319514020231:lmkouq8yagw";
    private static final String SEARCH_QUERY = "Android";

    private TextView mResult;
    private ProgressDialog mProgress;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ScrollView scrollView = new ScrollView(this);
        mResult = new TextView(this);
        scrollView.addView(mResult,
            new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        setContentView(scrollView);

        //创建请求
        try{
            //简单 GET 请求
            String url = String.format(SEARCH_URI, SEARCH_KEY,
                SEARCH_CX, SEARCH_QUERY);
            RestTask getTask = RestUtil.obtainGetTask(url);
            getTask.setResponseCallback(this);
            getTask.setProgressCallback(this);
```

```

        getTask.execute();

        //向用户显示进度
        mProgress = ProgressDialog.show(this, "Searching",
            "Waiting For Results...", true);
    } catch (Exception e) {
        mResult.setText(e.getMessage());
    }
}

@Override
public void onProgressUpdate(int progress) {
    if (progress >= 0) {
        if (mProgress != null) {
            mProgress.dismiss();
            mProgress = null;
        }
        //更新用户的进度
        mResult.setText(
            String.format("Download Progress: %d%%", progress));
    }
}

@Override
public void onRequestSuccess(String response) {
    //结束进度条
    if(mProgress != null) {
        mProgress.dismiss();
    }
    //处理返回的数据(这里只是把结果显示出来)
    mResult.setText(response);
}

@Override
public void onRequestError(Exception error) {
    //结束进度条
    if(mProgress != null) {
        mProgress.dismiss();
    }
    //处理返回的数据(这里只是把结果显示出来)
    mResult.setText("An Error Occurred: "+error.getMessage());
}
}

```

这个示例与之前的迭代几乎是一样的。还是通过一些必要的查询参数构造 URL 并且得到一个 RestTask 实例。然后将 Activity 作为请求和执行的回调。

但是可以看到，我们添加了 ProgressCallback，而 Activity 会实现它向右的接口，这些 Activity 就可以得到下载进度的通知。然而，并不是所有的网络服务器都会为请求返回一个有效的内容长度(而是返回-1)，这样会导致很难实现基于百分比的进度表示。这种情况

下，我们的回调函数在下载完成前会一直显示一个持续的进度对话框。而对于进度值可以确定的情况，该进度对话框会消失并在屏幕上显示进度的百分比。

下载完成后，Activity 会得到一个下载结果的 JSON 字符串回调。关于解析结构化 XML 和 JSON 数据的内容会在范例 3-7 和 3-8 中进行讨论，这里会先简单在用户界面上显示原始的响应数据。

POST 示例

程序清单 3-20 演示了使用新的 RestTask 来处理简单的表单数据 POST。访问的站点还是 httpbin.org，所以屏幕上显示的结果数据即为我们传入的表单参数的响应结果。

程序清单 3-20 执行 POST 请求 API 的 Activity

```
public class SearchActivity extends Activity implements
    RestTask.ProgressCallback, RestTask.ResponseCallback {

    private static final String POST_URI = "http://httpbin.org/post";

    private TextView mResult;
    private ProgressDialog mProgress;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ScrollView scrollView = new ScrollView(this);
        mResult = new TextView(this);
        scrollView.addView(mResult,
            new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        setContentView(scrollView);

        //创建请求
        try{
            //简单 POST 请求
            List<NameValuePair> parameters = new ArrayList<NameValuePair>();
            parameters.add(new BasicNameValuePair("title", "Android Recipes"));
            parameters.add(new BasicNameValuePair("summary",
                "Learn Android Quickly"));
            parameters.add(new BasicNameValuePair("authors", "Smith/Friesen"));
            RestTask postTask =
                RestUtil.obtainFormPostTask(POST_URI, parameters);
            postTask.setResponseCallback(this);
            postTask.setProgressCallback(this);

            postTask.execute();

            //向用户显示进度
            mProgress = ProgressDialog.show(this, "Searching",
                "Waiting For Results...", true);
        }
```

```

        } catch (Exception e) {
            mResult.setText(e.getMessage());
        }
    }

    @Override
    public void onProgressUpdate(int progress) {
        if (progress >= 0) {
            if (mProgress != null) {
                mProgress.dismiss();
                mProgress = null;
            }
            //更新用户的进度
            mResult.setText(
                String.format("Download Progress: %d%%", progress));
        }
    }

    @Override
    public void onRequestSuccess(String response) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }
        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText(response);
    }

    @Override
    public void onRequestError(Exception error) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }
        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText("An Error Occurred: "+error.getMessage());
    }
}

```

这里需要注意的是，进度回调只涉及了下载相关的响应，而没有涉及 POST 数据的上传，上传相关处理可能需要由开发者来实现。

上传示例

程序清单 3-21 演示了在 Android 框架中无法使用 Apache 原生组件实现的功能：复合 POST。

程序清单 3-21 执行复合 POST 请求 API 的 Activity

```

public class SearchActivity extends Activity implements
    RestTask.ProgressCallback,

```

```

        RestTask.ResponseCallback {

private static final String POST_URI = "http://httpbin.org/post";

private TextView mResult;
private ProgressDialog mProgress;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ScrollView scrollView = new ScrollView(this);
    mResult = new TextView(this);
    scrollView.addView(mResult,
        new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
    setContentView(scrollView);

    //创建请求
    try{
        //要 POST 的文件
        Bitmap image = BitmapFactory.decodeResource(getResources(),
            R.drawable.ic_launcher);
        File imageFile = new File(getExternalCacheDir(), "myImage.png");
        FileOutputStream out = new FileOutputStream(imageFile);
        image.compress(CompressFormat.PNG, 0, out);
        out.flush();
        out.close();
        List<NameValuePair> fileParameters =
            new ArrayList<NameValuePair>();
        fileParameters.add(new BasicNameValuePair("title",
            "Android Recipes"));
        fileParameters.add(new BasicNameValuePair("description",
            "Image File Upload"));
        RestTask uploadTask = RestUtil.obtainMultipartPostTask(POST_URI,
            fileParameters, imageFile, "avatarImage");
        uploadTask.setResponseCallback(this);
        uploadTask.setProgressCallback(this);

        uploadTask.execute();

        //向用户显示进度
        mProgress = ProgressDialog.show(this, "Searching",
            "Waiting For Results...", true);
    } catch (Exception e) {
        mResult.setText(e.getMessage());
    }
}

@Override
public void onProgressUpdate(int progress) {

```

```

        if (progress >= 0) {
            if (mProgress != null) {
                mProgress.dismiss();
                mProgress = null;
            }
            //更新用户的进度
            mResult.setText(
                String.format("Download Progress: %d%%", progress));
        }
    }

    @Override
    public void onRequestSuccess(String response) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }
        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText(response);
    }

    @Override
    public void onRequestError(Exception error) {
        //结束进度条
        if(mProgress != null) {
            mProgress.dismiss();
        }
        //处理返回的数据(这里只是把结果显示出来)
        mResult.setText("An Error Occurred: "+error.getMessage());
    }
}

```

本例中，我们构造的 POST 请求包含两个不同的部分：表单数据部分(由 name-value 对组成)和文件部分。由于只是为了演示这个示例，我们使用了应用程序的图标并将它快速地写入到外部存储器的一个 PNG 文件(用于上传)中。

本例中，HttpBin 的 JSON 响应数据将对应于表单数据元素和 Base64 编码表示的 PNG 图像。

基本授权

向新的 RestTask 中添加基本授权的过程相当简单。有两种方式可以实现它：在每个请求中直接添加或者全局使用名为 Authenticator 类。首先我们看一下在单个请求中添加基本授权。程序清单 3-22 修改了 RestUtil 的方法来关联适当格式的用户名和密码。

程序清单 3-22 实现了基本授权功能的 RestUtil

```

public class RestUtil {

    public static final RestTask obtainGetTask(String url)
        throws MalformedURLException, IOException {

```

```

        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();

        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
        connection.setDoInput(true);

        RestTask task = new RestTask(connection);
        return task;
    }

    public static final RestTask obtainAuthenticatedGetTask(String url,
        String username, String password) throws
        MalformedURLException, IOException {
        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();

        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
        connection.setDoInput(true);

        attachBasicAuthentication(connection, username, password);

        RestTask task = new RestTask(connection);
        return task;
    }

    public static final RestTask obtainAuthenticatedFormPostTask(String url,
        List<NameValuePair> formData, String username, String password)
        throws MalformedURLException, IOException {
        HttpURLConnection connection = (HttpURLConnection) (new URL(url))
            .openConnection();

        connection.setReadTimeout(10000);
        connection.setConnectTimeout(15000);
        connection.setDoOutput(true);

        attachBasicAuthentication(connection, username, password);

        RestTask task = new RestTask(connection);
        task.setFormBody(formData);

        return task;
    }

    private static void attachBasicAuthentication(URLConnection connection,
        String username, String password) {
        //添加基本授权头
        String userpassword = username + ":" + password;
        String encodedAuthorization =

```

```

        Base64.encodeToString(userpassword.getBytes(), Base64.NO_WRAP);
        connection.setRequestProperty("Authorization", "Basic " +
            encodedAuthorization);
    }
}

```

基本授权是作为头字段加入到一个 HTTP 请求中的，该头字段包括属性名称“Authorization”、“Basic”加上用户名密码的 Base64 编码字符串。attachBasicAuthentication() 辅助方法在 URLConnection 赋给 RestTask 之前会将上述属性名称和编码字符串设置到 URLConnection 上。

当不是所有请求都使用相同方式授权时，直接在请求中添加授权是一种非常好的方式。但是，有时候只需要设置一次认证信息即可让所有的请求使用它们。这时候就需要 Authenticator 类了。Authenticator 允许为应用程序进程的所有请求设置全局的用户名和密码认证信息。让我们看一下程序清单 3-23，它就实现了这个功能。

程序清单 3-23 使用 Authenticator 的 Activity

```

public class AuthActivity extends Activity implements ResponseCallback {

    private static final String URI =
        "http://httpbin.org/basic-auth/android/recipes";
    private static final String USERNAME = "android";
    private static final String PASSWORD = "recipes";

    private TextView mResult;
    private ProgressDialog mProgress;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mResult = new TextView(this);
        setContentView(mResult);

        Authenticator.setDefault(new Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(USERNAME,
                    PASSWORD.toCharArray());
            }
        });

        try {
            RestTask task = RestUtil.obtainGetTask(URI);
            task.setResponseCallback(this);
            task.execute();
        } catch (Exception e) {
            mResult.setText(e.getMessage());
        }
    }
}

```

```

@Override
public void onRequestSuccess(String response) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    mResult.setText(response);
}

@Override
public void onRequestError(Exception error) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    mResult.setText(error.getMessage());
}
}

```

该示例会再次连接 `HttpBin`，但本次连接的站点需要使用有效的认证信息。主机需要的用户名和命名是被编码到 URL 路径中的，如果没有提供相应的认证信息属性的话，主机的响应结果为 `UNAUTHORIZED`。

只需要调用 `Authenticator.setDefault()` 方法并传入一个新的 `Authenticator` 实例，后续所有的请求在认证时就都会使用上面提供的认证信息。所以我们创建了一个新的 `Password-Authentication` 实例并传入正确的用户名和密码，这样进程中的所有 `URLConnection` 实例都会使用它。注意，这个示例中，请求并没有关联认证信息，但执行请求后会得到一个已经授权的响应。

响应缓存

(API Level 13)

`HttpURLConnection` 最后一个可以利用的平台改进方案是通过 `HttpURLConnection` 设置响应缓存。加速应用程序响应速度的方法就是将从远程主机获取的响应缓存起来。这样对于频繁使用的请求就可以直接从缓存中加载而不必每次都访问网络。在应用程序中安装和移除缓存只需要简单的几行代码。

```

//安装一个响应缓存
try {
    File httpCacheDir = new File(context.getCacheDir(), "http");
    long httpCacheSize = 10 * 1024 * 1024; // 10 MiB
    HttpURLConnection.install(httpCacheDir, httpCacheSize);
} catch (IOException e) {
    Log.i(TAG, "HTTP response cache installation failed:" + e);
}

//清空响应缓存
HttpURLConnection cache = HttpURLConnection.getInstalled();

```

```

if (cache != null) {
    cache.flush();
}

```

注意:

HttpResponseCache 只能应用于 HttpURLConnection 的变种类。它并不适用于 Apache HttpClient。

3.7 解析 JSON

3.7.1 问题

应用程序需要解析从一个 API 或者其他资源所返回的 JSON 格式的响应结果。

3.7.2 解决方案

(API Level 1)

使用 Android 中的 org.json 解析类。SDK 在 org.json 包中自带了一个非常高效的类集用来解析 JSON 格式的字符串。只需要用已经格式化的字符串数据生成一个新的 JSONObject 或者 JSONArray，然后就可以使用一系列访问方法去获得这些对象中的原始数据或者内嵌的 JSONObject 和 JSONArray 数据。

3.7.3 实现机制

默认情况下，这个 JSON 解析器是非常严格的，也就意味着当遇到无效的 JSON 数据或者无效的 key 时会抛出一个异常。以“get”开头的访问方法在请求的值找不到时会抛出一个 JSONException。在某些情况下，这种机制可能不太好，因此就存在一套以“opt”为前缀的方法。这些方法在请求的 key 所对应的值找不到时会返回 null 而不是抛出一个异常。另外，它们中的很多方法都提供了重载版本，可以在失败时返回一个传入的参数而不是返回 null。

让我看一个示例，了解一下如何将一个 JSON 字符串解析成一些有用的小数据。

先看一下程序清单 3-24 中的 JSON 数据

程序清单 3-24 JSON 示例

```

{
  "person": {
    "name": "John",
    "age": 30,
    "children": [
      {
        "name": "Billy"
        "age": 5
      }
    ]
  }
}

```

```

        },
        {
            "name": "Sarah"
            "age": 7
        },
        {
            "name": "Tommy"
            "age": 9
        }
    ]
}
}

```

这里定义一个对象，它有 3 个值：**name** (字符串)、**age** (整型)和 **children**。名为“children”的参数是一个有三个对象组成的数组，每个对象都有自己的 **name** 和 **age**。程序清单 3-25 和 3-26 中的示例展示了使用 **org.json** 去解析这个数据并在 **TextView** 中显示它的一些元素。

程序清单 3-25 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/line1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <TextView
        android:id="@+id/line2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <TextView
        android:id="@+id/line3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>

```

程序清单 3-26 JSON 解析 Activity 示例

```

public class MyActivity extends Activity {
    private static final String JSON_STRING =
        "{ \"person\": { \"name\": \"John\", \"age\": 30, \"children\": [ \"
        + \" { \"name\": \"Billy\", \"age\": 5 }, \"
        + \" { \"name\": \"Sarah\", \"age\": 7 }, \"
        + \" { \"name\": \"Tommy\", \"age\": 9 } \"
        + \" ] } } \" ";
}

```

```

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

    TextView line1 = (TextView)findViewById(R.id.line1);
    TextView line2 = (TextView)findViewById(R.id.line2);
    TextView line3 = (TextView)findViewById(R.id.line3);
    try {
        JSONObject person =
            (new JSONObject(JSON_STRING)).getJSONObject("person");
        String name = person.getString("name");
        line1.setText("This person's name is " + name);
        line2.setText(name + " is " + person.getInt("age")
            + " years old.");
        line3.setText(name + " has "
            + person.getJSONArray("children").length()
            + " children.");
    } catch (JSONException e) {
        e.printStackTrace();
    }
}

```

在本例中，JSON 字符串作为一个常量采用了硬编码的方式。当创建了 Activity 后，这个字符串被转换为一个 JSONObject，这样它的数据就可以通过 key-value 对的方式进行访问，就像存储在 map 或者 dictionary 中一样。由于我们使用严格方法来访问数据，因此所有的业务逻辑都被封装在一个 try/catch 语句中。

JSONObject.getString() 和 JSONObject.getInt() 方法用来读取原始数据并把它放到 TextView 中；getJSONArray() 用来取出内嵌的“children”数组。在读取数据时，JSONArray 拥有和 JSONObject 一样的访问方法，但它的方法参数是数组的 index 而不是 key 的名称。另外，JSONArray 可以返回它的长度，我们会在示例中使用这个长度显示每个人孩子的数量。

如 3-3 显示了示例应用程序的结果。

1. 调试技巧

JSON 是一种高效的书写方式；然而，因为阅读原始 JSON 字符串很难，所以也很难调试解析问题。很多情况下，你解析的 JSON 来自于远程资源或者你根本不熟悉它，而且

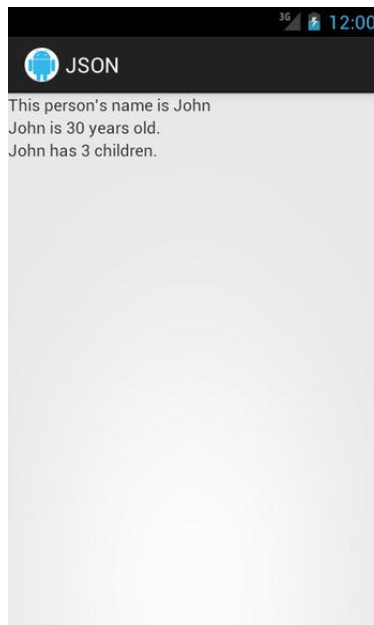


图 3-3 在 Activity 中显示解析过的 JSON 数据

在调试时还需要显示它。因此，JSONObject 和 JSONArray 都提供了重载的 toString()方法，参数为一个整型，它会返回一个用缩进的打印漂亮的字符串数据，使之更容易理解。通常在一些麻烦的地方添加诸如 myJsonObject.toString(2)这样的东西，可以省去很多时间和烦恼。

3.8 解析 XML

3.8.1 问题

应用程序需要解析从一个 API 或者其他资源所返回的 XML 格式的响应结果。

3.8.2 解决方案

(API Level 1)

可以实现一个 org.xml.sax.helpers.DefaultHandler 的子类来解析数据，它使用的是基于事件的 SAX 方式。Android 有三种主要用于解析 XML 数据的方式：DOM、SAX 和 Pull。这其中最容易实现的就是 SAX 解析器，它也是内存效率最高的。SAX 解析通过遍历 XML 数据来实现，并在每个元素的开头和结尾产生回调事件。

3.8.3 实现机制

为了进一步介绍如何解析 XML，先来看一下请求 RSS/ATOM 新闻源时返回的 XML 格式数据(参见程序清单 3-20)。

程序清单 3-27 RSS 基本结构

```
<rss version="2.0">
  <channel>
    <item>
      <title></title>
      <link></link>
      <description></description>
    </item>
    <item>
      <title></title>
      <link></link>
      <description></description>
    </item>
    <item>
      <title></title>
      <link></link>
      <description></description>
    </item>
    ...
  </channel>
</rss>
```

```

    </channel>
</rss>

```

在各个<title>、<link>和<description>标签之间就是每个项的值。我们可以使用 SAX 将这段数据解析成一个数组，应用程序可以很方便地在列表中将数据呈现给用户(参见程序清单 3-28)。

程序清单 3-28 自定义的 RSS 解析处理器

```

public class RSSHandler extends DefaultHandler {

    public class NewsItem {
        public String title;
        public String link;
        public String description;

        @Override
        public String toString() {
            return title;
        }
    }

    private StringBuffer buf;
    private ArrayList<NewsItem> feedItems;
    private NewsItem item;

    private boolean inItem = false;

    public ArrayList<NewsItem> getParsedItems() {
        return feedItems;
    }

    //在每个元素开始时调用
    @Override
    public void startElement(String uri, String name, String qName, Attributes atts) {
        if("channel".equals(name)) {
            feedItems = new ArrayList<NewsItem>();
        } else if("item".equals(name)) {
            item = new NewsItem();
            inItem = true;
        } else if("title".equals(name) && inItem) {
            buf = new StringBuffer();
        } else if("link".equals(name) && inItem) {
            buf = new StringBuffer();
        } else if("description".equals(name) && inItem) {
            buf = new StringBuffer();
        }
    }

    //在每个元素结束时调用

```

```

@Override
public void endElement(String uri, String name, String qName) {
    if("item".equals(name)) {
        feedItems.add(item);
        inItem = false;
    } else if("title".equals(name) && inItem) {
        item.title = buf.toString();
    } else if("link".equals(name) && inItem) {
        item.link = buf.toString();
    } else if("description".equals(name) && inItem) {
        item.description = buf.toString();
    }

    buf = null;
}

//调用元素中的字符数据
@Override
public void characters(char ch[], int start, int length) {
    //处理缓存已经初始化的情况
    if(buf != null) {
        for (int i=start; i<start+length; i++) {
            buf.append(ch[i]);
        }
    }
}
}

```

在每个元素开始和结束时都会通过 `startElement()` 和 `endElement()` 方法通知 `RSSHandler`。在这之间，元素值中的字符会传递给 `characters()` 回调方法。

- (1) 当解析器碰到第一个元素时，会初始化项目列表。
- (2) 对于遇到的每个项目元素，会初始化一个新的 `NewsItem`。
- (3) 在每个项目元素内部，数据元素被置入一个 `StringBuffer` 中，然后插入到 `NewsItem` 中。
- (4) 当到达每个项目的结尾时，会把 `NewsItem` 添加到列表中。
- (5) 解析完成后，`feedItems` 中包含了源数据中的所有项目。

接下来，使用在 3-6 节中介绍的一些技巧来下载最新的 RSS 格式的 Google 新闻内容(参见程序清单 3-29)。

程序清单 3-29 解析 XML 并显示各个项目内容的 Activity

```

public class FeedActivity extends Activity implements ResponseCallback {
    private static final String TAG = "FeedReader";
    private static final String FEED_URI =
        "http://news.google.com/?output=rss";

    private ListView mList;
    private ArrayAdapter<NewsItem> mAdapter;
    private ProgressDialog mProgress;
}

```

```

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

    mList = new ListView(this);
    mAdapter = new ArrayAdapter<NewsItem>(this,
        android.R.layout.simple_list_item_1,
        android.R.id.text1);
    mList.setAdapter(mAdapter);
    mList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
            NewsItem item = mAdapter.getItem(position);
            Intent intent = new Intent(Intent.ACTION_VIEW);
            intent.setData(Uri.parse(item.link));
            startActivity(intent);
        }
    });

    setContentView(mList);
}

@Override
public void onResume() {
    super.onResume();
    //获取 RSS 源数据
    try{
        HttpGet feedRequest = new HttpGet( new URI(FEED_URI) );
        RestTask task = new RestTask();
        task.setResponseCallback(this);
        task.execute(feedRequest);
        mProgress = ProgressDialog.show(this, "Searching",
            "Waiting For Results...", true);
    } catch (Exception e) {
        Log.w(TAG, e);
    }
}

@Override
public void onRequestSuccess(String response) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    //处理响应数据
    try {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser p = factory.newSAXParser();

```

```

        RSSHandler parser = new RSSHandler();
        p.parse(new InputSource(new StringReader(response)), parser);

        mAdapter.clear();
        for(NewsItem item : parser.getParsedItems()) {
            mAdapter.add(item);
        }
        mAdapter.notifyDataSetChanged();
    } catch (Exception e) {
        Log.w(TAG, e);
    }
}

@Override
public void onRequestError(Exception error) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    //显示错误
    mAdapter.clear();
    mAdapter.notifyDataSetChanged();
    Toast.makeText(this, error.getMessage(), Toast.LENGTH_SHORT).show();
}
}

```

这个示例修改之后会显示一个 `ListView`，其中的数据就是从 RSS 源解析出来的。在这个示例中，我们为列表添加了一个 `OnItemClickListener`，用户单击时会在浏览器中加载新闻链接。

当数据从 API 的回调方法返回时，Android 内置的 SAX parser 会遍历 XML 字符串。`SAXParser.parse()` 会使用 `RSSHandler` 的实例来处理 XML，从 XML 中解析的内容会用来填充 `RSSHandler` 的 `feedItems` 列表。然后再逐个处理解析出来的项目，将其添加到 `ArrayAdapter` 中，最终显示在 `ListView` 中。

XmlPullParser

由框架提供的 `XmlPullParser` 是另一种高效解析 XML 数据的方式。和 SAX 一样，解析过程也是基于流的，由于解析开始之前并不需要加载整个 XML 数据结构，因此在解析大数据源时也就不要太多的内存。下面，让我们看一下使用 `XmlPullParser` 解析 RSS 源数据的示例。与 SAX 不同，我们必须手动地干预每一步的解析过程，即使是我们不感兴趣的标签。

程序清单 3-30 包含了一个工厂类，它会迭代源数据并构建元素模型

程序清单 3-30 用来将 XML 解析成模型对象的工厂类

```

public class NewsItemFactory {

```

```

/*数据模型类*/
public static class NewsItem {
    public String title;
    public String link;
    public String description;

    @Override
    public String toString() {
        return title;
    }
}

/*
 * 将 RSS 源解析为一个 NewsItem 元素的列表
 */
public static List<NewsItem> parseFeed(XmlPullParser parser)
    throws XmlPullParserException, IOException {
    List<NewsItem> items = new ArrayList<NewsItem>();

    while (parser.next() != XmlPullParser.END_TAG) {
        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }

        if (parser.getName().equals("rss") ||
            parser.getName().equals("channel")) {
            //跳过这些元素, 但允许解析它们内部的元素
        } else if (parser.getName().equals("item")) {
            NewsItem newsItem = readItem(parser);
            items.add(newsItem);
        } else {
            //跳过其他元素以及它们的子元素
            skip(parser);
        }
    }

    //返回解析后的列表
    return items;
}

/*
 * 将每个<item>元素解析为一个 NewsItem
 */
private static NewsItem readItem(XmlPullParser parser) throws
    XmlPullParserException, IOException {
    NewsItem newsItem = new NewsItem();

    //开头必须是有效的<item>元素
    parser.require(XmlPullParser.START_TAG, null, "item");
    while (parser.next() != XmlPullParser.END_TAG) {

```

```

        if (parser.getEventType() != XmlPullParser.START_TAG) {
            continue;
        }

        String name = parser.getName();
        if (name.equals("title")) {
            parser.require(XmlPullParser.START_TAG, null, "title");
            newItem.title = readText(parser);
            parser.require(XmlPullParser.END_TAG, null, "title");
        } else if (name.equals("link")) {
            parser.require(XmlPullParser.START_TAG, null, "link");
            newItem.link = readText(parser);
            parser.require(XmlPullParser.END_TAG, null, "link");
        } else if (name.equals("description")) {
            parser.require(XmlPullParser.START_TAG, null, "description");
            newItem.description = readText(parser);
            parser.require(XmlPullParser.END_TAG, null, "description");
        } else {
            //跳过其他元素以及它们的子元素
            skip(parser);
        }
    }

    return newItem;
}

/*
 * 读取当前元素的文本内容，该内容是 start 和 end 标签之间的数据
 */
private static String readText(XmlPullParser parser) throws
    IOException, XmlPullParserException {
    String result = "";
    if (parser.next() == XmlPullParser.TEXT) {
        result = parser.getText();
        parser.nextTag();
    }
    return result;
}

/*
 * 辅助方法，用来跳过当前元素以及该元素的子元素
 */
private static void skip(XmlPullParser parser) throws
    XmlPullParserException, IOException {
    if (parser.getEventType() != XmlPullParser.START_TAG) {
        throw new IllegalStateException();
    }
}

/*
 * 对于每个新标签，会把一个 depth 计数器加 1。到达每个标签的结尾时会把计数器减 1

```

```

        * 并且在 end 标签与开始时标签匹配时会返回
        */
        int depth = 1;
        while (depth != 0) {
            switch (parser.next()) {
                case XmlPullParser.END_TAG:
                    depth--;
                    break;
                case XmlPullParser.START_TAG:
                    depth++;
                    break;
            }
        }
    }
}

```

Pull 解析过程的工作原理就是把数据流作为一系列的事件来处理。应用程序通过调用 `next()` 方法或者该方法的变种来告诉解析器处理下一个事件。以下是解析器会处理的事件类型：

- **START_DOCUMENT**：当解析器首次初始化时会返回这个事件。在首次调用 `next()`、`nextToken()` 或者 `nextTag()` 之前，都会是这个状态。
- **START_TAG**：解析器刚刚解析到标签元素的开始部分。标签的名称可以通过 `getName()` 获得，里面的任何属性也可以通过 `getAttributeValue` 和相关的方法获得。
- **TEXT**：读取标签元素内部的字符数据，可以通过 `getText()` 获取。
- **END_TAG**：解析器刚刚解析一个标签元素的结尾部分。和它相匹配的开始标签的名称可以通过 `getName()` 获得。
- **END_DOCUMENT**：到达了数据流的结尾。

由于我们必须自己操作解析器，故而我们创建了一个辅助方法 `skip()`，它可以帮助解析器跳过我们不感兴趣的标签。这个方法从当前位置开始遍历所有的内嵌子元素，直到找到匹配的结束标签，并把它们全部跳过。这里使用了一个 `depth` 计数器，碰到开始标签时会递增，碰到结束标签时会递减。当 `depth` 计数器为 0 时，我们就找到了与开始位置相匹配的结束标签了。

本例中，在调用 `parseFeed()` 方法时，解析器首先会迭代数据流来查找可以转换为 `NewsItem` 的 `<item>` 标签。除了 `<rss>` 和 `<channel>`，所有不是 `<item>` 的元素都可以跳过。这是因为所有的 `item` 都是内嵌在这两个标签之中的，因此即使我们对它们不感兴趣，也不能把它们交给 `skip()` 处理，否则的话所有的 `item` 都会被跳过。

解析每个 `<item>` 元素的工作是由 `readItem()` 方法完成的，它会构造一个新的 `NewsItem`，该 `NewsItem` 的内容来自于 `<item>` 内部的数据。`readItem()` 方法首先会调用 `require()`，它是一种安全性的检查，能够确保 XML 是我们所希望的格式。如果当前的解析器事件和传入的命名空间、标签名称相匹配的话，这个方法会静默地返回，否则，它会抛出一个异常。当我们遍历子元素时，我们主要查找 `title`、`link` 和 `description` 标签，这样就可以把它们的值读取到模型数据中。查找到所需的标签后，`readText()` 会操作解析器把相关数据取出。同样，在 `<item>` 内部有一些其他元素我们并没有解析，对于不需要的标签只需要调用 `skip()` 即可。

你已经看到了 XmlPullParser 非常灵活，原因是可控制整个过程的每一步，但这也要求写更多的代码来完成相同的结果。程序清单 3-31 展示了使用了新的解析器来完成源数据显示的 Activity。

程序清单 3-31 显示解析 XML 源的 Activity

```
public class PullFeedActivity extends Activity implements ResponseCallback {
    private static final String TAG = "FeedReader";
    private static final String FEED_URI =
        "http://news.google.com/?output=rss";

    private ListView mList;
    private ArrayAdapter<NewsItem> mAdapter;
    private ProgressDialog mProgress;

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

        mList = new ListView(this);
        mAdapter = new ArrayAdapter<NewsItem>(this,
            android.R.layout.simple_list_item_1,
            android.R.id.text1);
        mList.setAdapter(mAdapter);
        mList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View v,
                int position, long id) {
                NewsItem item = mAdapter.getItem(position);
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(Uri.parse(item.link));
                startActivity(intent);
            }
        });

        setContentView(mList);
    }

    @Override
    public void onResume() {
        super.onResume();
        //获取 RSS 源数据
        try{
            HttpGet feedRequest = new HttpGet( new URI(FEED_URI) );
            RestTask task = new RestTask();
            task.setResponseCallback(this);
            task.execute(feedRequest);
            mProgress = ProgressDialog.show(this, "Searching",
                "Waiting For Results...", true);
        } catch (Exception e) {
```

```

        Log.w(TAG, e);
    }
}

@Override
public void onRequestSuccess(String response) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    //处理响应数据
    try {
        XmlPullParser parser = Xml.newPullParser();
        parser.setInput(new StringReader(response));
        //跳到第一个标签
        parser.nextTag();

        mAdapter.clear();
        for(NewsItem item : NewsItemFactory.parseFeed(parser)) {
            mAdapter.add(item);
        }
        mAdapter.notifyDataSetChanged();
    } catch (Exception e) {
        Log.w(TAG, e);
    }
}

@Override
public void onRequestError(Exception error) {
    if (mProgress != null) {
        mProgress.dismiss();
        mProgress = null;
    }
    //显示错误
    mAdapter.clear();
    mAdapter.notifyDataSetChanged();
    Toast.makeText(this, error.getMessage(), Toast.LENGTH_SHORT).show();
}
}

```

使用 `Xml.newPullParser()` 可以实例化一个新的 `XmlPullParser`，通过 `setInput()` 可以将数据源的输入流作为一个 `Reader`。本例中，从 Web 服务中所返回的数据已经是字符串了，所以我们把它包装成一个 `StringReader` 来让解析器解析。我们可以把解析器传给 `NewsItemFactory`，之后会返回 `NewsItem` 元素的列表，我们把它添加到 `ListAdapter` 中然后像之前那样显示出来。

提示：

还可以使用 `XmlPullParser` 解析应用程序中绑定的本地 XML 数据。把你的原始 XML 放到资源文件中(如 `res/xml/`)，然后你就可以实例化一个 `XmlResourceParser`，它会使用

Resources.getXml()预加载你的本地数据。

3.9 接收短信

3.9.1 问题

应用程序需要响应接收到的短消息，也叫文本消息。

3.9.2 解决方案

(API Level 1)

注册一个 `BroadcastReceiver` 来监听收到的消息，并在 `onReceive()` 中处理它们。当收到一条短信时，操作系统会发送一个 `action` 值为 `android.provider.Telephony.SMS_RECEIVED` 的广播 `Intent`。应用程序则可以注册一个 `BroadcastReceiver` 过滤这个 `Intent` 并处理收到的数据。

注意：

接收这个广播同样并不会阻止系统其他的应用程序接收它。默认的消息处理应用程序还会接收这个消息并显示。

3.9.3 实现机制

上一个示例中，我们定义了一个 `BroadcastReceiver` 并作为 `Activity` 的内部私有成员变量。在本例中，最好单独定义接收器并在 `AndroidManifest.xml` 中使用 `<receiver>` 标签注册它。这样即使应用程序处于非激活状态，接收器也可以处理收到的事件。程序清单 3-31 和 3-32 显示了一个接收器的示例，它监控所有收到的短信并将感兴趣的短信显示在一个 `Toast` 上。

程序清单 3-31 接收短信的 `BroadcastReceiver`

```
public class SmsReceiver extends BroadcastReceiver {
    private static final String SHORTCODE = "55443";

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();

        Object[] messages = (Object[])bundle.get("pdu");
        SmsMessage[] sms = new SmsMessage[messages.length];
        //为每个收到的 PDU 创建短信
        for(int n=0; n < messages.length; n++) {
            sms[n] = SmsMessage.createFromPdu((byte[]) messages[n]);
        }
        for(SmsMessage msg : sms) {
```

```

        //检查短信是否来自我们已知的发送者
        if(TextUtils.equals(msg.getOriginatingAddress(), SHORTCODE)) {
            Toast.makeText(context,
                "Received message from the mothership: "
                + msg.getMessageBody(),
                Toast.LENGTH_SHORT).show();
        }
    }
}

```

程序清单 3-32 部分 AndroidManifest.xml 的内容

```

<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <application ...>
        <receiver android:name=".SmsReceiver">
            <intent-filter>
                <action android:name="android.provider.Telephony.SMS_RECEIVED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>

```

重点:

接收短信需要在 manifest 中声明 android.permission.RECEIVE_SMS 权限。

接收的短信作为一个 byte 数组(Object 数组), 通过广播 Intent 的 extra 传递, 每个 byte 数组代表一个短信 PDU(Packet Data Unit)。SmsMessage.createFromPdu()方法可以方便地从原始的 PDU 数据创建 SmsMessage 对象。完成设置之后, 可以检查每条短信, 判断是否要对其进行处理。在这个示例中, 我们会将每条短信的发送地址与一个已知的短码进行比较, 在收到符合要求的短信时通知用户。

在这个示例中显示 Toast 时, 你有可能希望在这个时候给用户提供更多有用的信息。短信中可能有应用程序的请求码, 这样就可以启动合适的 Activity 将该消息在应用程序中显示给用户。

3.10 发送短信

3.10.1 问题

应用程序需要发送短信。

3.10.2 解决方案

(API Level 4)

使用 `SmsManager` 发送文字或者数据短信。`SmsManager` 是一个系统服务用来处理短信发送并把操作的状态反馈给应用程序。`SmsManager` 提供了 `SmsManager.sendTextMessage()` 和 `SmsManager.sendMultipartTextMessage()` 来发送文字短信以及 `SmsManager.sendDataMessage()` 方法来发送数据短信。这些方法都含有用于传递发送操作状态和将消息传回请求目标的 `PendingIntent` 参数。

3.10.3 实现机制

下面看一个简单的示例，发送一个短信并监控其状态(参见程序清单 3-25)。

程序清单 3-33 发送短信的 Activity

```
public class SmsActivity extends Activity {
    private static final String SHORTCODE = "55443";
    private static final String ACTION_SENT = "com.examples.sms.SENT";
    private static final String ACTION_DELIVERED =
        "com.examples.sms.DELIVERED";

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

        Button sendButton = new Button(this);
        sendButton.setText("Hail the Mothership");
        sendButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                sendSMS("Beam us up!");
            }
        });

        setContentView(sendButton);
    }

    private void sendSMS(String message) {
        PendingIntent sIntent = PendingIntent.getBroadcast(this, 0,
            new Intent(ACTION_SENT), 0);
        PendingIntent dIntent = PendingIntent.getBroadcast(this, 0,
            new Intent(ACTION_DELIVERED), 0);
        //监控操作的状态
        registerReceiver(sent, new IntentFilter(ACTION_SENT));
        registerReceiver(delivered, new IntentFilter(ACTION_DELIVERED));
        //发送短信
        SmsManager manager = SmsManager.getDefault();
        manager.sendTextMessage(SHORTCODE, null, message, sIntent, dIntent);
    }
}
```

```

    }

    private BroadcastReceiver sent = new BroadcastReceiver(){
        @Override
        public void onReceive(Context context, Intent intent) {
            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    //发送成功
                    break;
                case SmsManager.RESULT_ERROR_GENERIC_FAILURE:
                case SmsManager.RESULT_ERROR_NO_SERVICE:
                case SmsManager.RESULT_ERROR_NULL_PDU:
                case SmsManager.RESULT_ERROR_RADIO_OFF:
                    //发送失败
                    break;
            }

            unregisterReceiver(this);
        }
    };

    private BroadcastReceiver delivered = new BroadcastReceiver(){
        @Override
        public void onReceive(Context context, Intent intent) {
            switch (getResultCode()) {
                case Activity.RESULT_OK:
                    //传递成功
                    break;
                case Activity.RESULT_CANCELED:
                    //传递失败
                    break;
            }

            unregisterReceiver(this);
        }
    };
}

```

重点:

发送短信需要在 manifest 中声明 android.permission.SEND_SMS 权限。

当用户单击按钮时就会通过 SMSManager 发送一条短信。由于 SMSManager 是一项系统服务，必须调用静态方法 SMSManager.getDefault()来获得它的一个引用。sendTextMessage()方法的参数为目标地址(号码)、服务中心地址以及短信内容。想要 SMSManager 使用系统默认的服务中心地址，那这个参数就应该设为 null。

这里注册了两个 BroadcastReceiver 用来接收要发出的回调 Intent：一个是发送操作的状态，另一个是传递的状态。这些接收器只有在操作处于待发送状态时才会被注册，在处理完 Intent 后就会立即取消注册。

3.11 蓝牙通信

3.11.1 问题

在应用程序中，想通过蓝牙通信实现不同设备之间的数据传输。

3.11.2 解决方案

(API Level 5)

可以使用 API 5 中引入的蓝牙 API 创建一个点对点的连接。蓝牙是一种非常流行的无线电技术，几乎现在所有的移动设备都支持该技术。很多用户认为蓝牙只能用来连接移动设备与无线耳机或者与车载音响系统整合。但实际上，对于开发者来说，在应用程序中蓝牙还是一种用来创建点对点连接的简单而高效的方式。

3.11.3 实现机制

重点：

Android 模拟器现在还不支持蓝牙。因此想要执行本例中的代码，必须把它运行在一个 Android 设备上。此外，想要很好地测试这个功能，需要在两个设备上同时运行这个应用程序。

1. 蓝牙点对点

程序清单 3-34 到程序清单 3-36 演示了使用蓝牙查找附近的其他用户并快速交换联系信息(本例中，只是交换电子邮件信息)。蓝牙上的连接是通过发现可用的“服务”，并通过一个 128 位的 UUID 连接到相应的服务。也就是说，在连接某个服务之前，必须首先发现或知道它的 UUID。

在本例中，连接两端设备运行的是相同的应用程序，两个应用都会有对应的 UUID，因此我们可以在代码中将 UUID 定义为常数。

注意：

为了确保你指定的 UUID 是唯一的，请使用网络上免费的 UUID 生成器。

程序清单 3-34 AndroidManifest.xml

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

```

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
<uses-sdk android:minSdkVersion="5" />

<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
</manifest>

```

重点:

记住, 想要使用这些 API, 需要在 manifest 中声明 android.permission.BLUETOOTH 权限。另外, 想要改变蓝牙的可发现性以及启用/禁用蓝牙适配器, 还要在 manifest 中声明 android.permission.BLUETOOTH_ADMIN 权限。

程序清单 3-35 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Enter Your Email:"
    />
    <EditText
        android:id="@+id/emailField"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"
        android:singleLine="true"
        android:inputType="textEmailAddress"
    />
    <Button
        android:id="@+id/scanButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Connect and Share"
    />
    <Button
        android:id="@+id/listenButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/scanButton"

```

```

        android:text="Listen for Sharers"
    />
</RelativeLayout>

```

这个示例的 UI 由一个用户输入他/她的电子邮件地址的 EditText 和两个进行通信的按钮组成。名为“Listen for Sharers”的按钮用来将设备设为监听模式。在这种模式下，设备会接受其他设备的连接，并与之进行通信。名为“Connect and Share”用来将设备设为搜索模式。在这种模式下，设备会搜索当前处于监听模式的设备，并与之进行连接(见程序清单 3-36)。

程序清单 3-36 蓝牙交换 Activity

```

public class ExchangeActivity extends Activity {

    //本应用程序唯一的 UUID(从网上生成的)
    private static final UUID MY_UUID =
        UUID.fromString("321cb8fa-9066-4f58-935e-ef55d1ae06ec");
    //发现时所有的一个更加友好的名字
    private static final String SEARCH_NAME = "bluetooth.recipe";

    BluetoothAdapter mBtAdapter;
    BluetoothSocket mBtSocket;
    Button listenButton, scanButton;
    EditText emailField;

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

        //检查系统状态
        mBtAdapter = BluetoothAdapter.getDefaultAdapter();
        if(mBtAdapter == null) {
            Toast.makeText(this, "Bluetooth is not supported.",
                Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        if (!mBtAdapter.isEnabled()) {
            Intent enableIntent =
                new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableIntent, REQUEST_ENABLE);
        }

        emailField = (EditText)findViewById(R.id.emailField);
        listenButton = (Button)findViewById(R.id.listenButton);
        listenButton.setOnClickListener(new View.OnClickListener() {

```

```

        @Override
        public void onClick(View v) {
            //首先要确保设备是可以被发现的
            if (mBtAdapter.getScanMode() !=
                BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
                Intent discoverableIntent =
                    new Intent(
                        BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
                discoverableIntent.putExtra(BluetoothAdapter.
                    EXTRA_DISCOVERABLE_DURATION, 300);
                startActivityForResult(discoverableIntent,
                    REQUEST_DISCOVERABLE);
                return;
            }
            startListening();
        }
    });
    scanButton = (Button)findViewById(R.id.scanButton);
    scanButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mBtAdapter.startDiscovery();
            setProgressBarIndeterminateVisibility(true);
        }
    });
}

@Override
public void onResume() {
    super.onResume();
    //为 activity 注册一个广播 intent
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(mReceiver, filter);
    filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
    registerReceiver(mReceiver, filter);
}

@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(mReceiver);
}

@Override
public void onDestroy() {
    super.onDestroy();
    try {
        if(mBtSocket != null) {
            mBtSocket.close();
        }
    }
}

```

```

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

    private static final int REQUEST_ENABLE = 1;
    private static final int REQUEST_DISCOVERABLE = 2;

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        switch(requestCode) {
            case REQUEST_ENABLE:
                if(resultCode != Activity.RESULT_OK) {
                    Toast.makeText(this, "Bluetooth Not Enabled.",
                        Toast.LENGTH_SHORT).show();
                    finish();
                }
                break;
            case REQUEST_DISCOVERABLE:
                if(resultCode == Activity.RESULT_CANCELED) {
                    Toast.makeText(this, "Must be discoverable.",
                        Toast.LENGTH_SHORT).show();
                } else {
                    startListening();
                }
                break;
            default:
                break;
        }
    }

    //启动服务器端口并监听
    private void startListening() {
        AcceptTask task = new AcceptTask();
        task.execute(MY_UUID);
        setProgressBarIndeterminateVisibility(true);
    }

    //AsyncTask to accept incoming connections
    private class AcceptTask extends AsyncTask<UUID,Void,BluetoothSocket> {
        @Override
        protected BluetoothSocket doInBackground(UUID... params) {
            String name = mBtAdapter.getName();
            try {
                //监听时，将发现名设置为一个指定的值
                mBtAdapter.setName(SEARCH_NAME);
                BluetoothServerSocket socket =
                    mBtAdapter.listenUsingRfcommWithServiceRecord(
                        "BluetoothRecipe", params[0]);
            }
        }
    }

```

```

        BluetoothSocket connected = socket.accept();
        //设置蓝牙适配器名称
        mBtAdapter.setName(name);
        return connected;
    } catch (IOException e) {
        e.printStackTrace();
        mBtAdapter.setName(name);
        return null;
    }
}

@Override
protected void onPostExecute(BluetoothSocket socket) {
    if(socket == null) {
        return;
    }
    mBtSocket = socket;
    ConnectedTask task = new ConnectedTask();
    task.execute(mBtSocket);
}

}

//AsyncTask 接收一行数据并发送
private class ConnectedTask extends
    AsyncTask<BluetoothSocket,Void,String> {

    @Override
    protected String doInBackground(BluetoothSocket... params) {
        InputStream in = null;
        OutputStream out = null;
        try {
            //发送数据
            out = params[0].getOutputStream();
            out.write(emailField.getText().toString().getBytes());
            //接收其他数据
            in = params[0].getInputStream();
            byte[] buffer = new byte[1024];
            in.read(buffer);
            //从结果创建一个空字符串
            String result = new String(buffer);
            //关闭连接
            mBtSocket.close();
            return result.trim();
        } catch (Exception exc) {
            return null;
        }
    }

    @Override

```

```

        protected void onPostExecute(String result) {
            Toast.makeText(ExchangeActivity.this, result, Toast.LENGTH_SHORT)
                .show();
            setProgressBarIndeterminateVisibility(false);
        }
    }

    //用来监听发现设备的 BroadcastReceiver
    private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            //当找到一个设备时
            if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                //从 Intent 中获得 BluetoothDevice 对象
                BluetoothDevice device =
                    intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (TextUtils.equals(device.getName(), SEARCH_NAME)) {
                    //匹配找到的设备, 连接
                    mBtAdapter.cancelDiscovery();
                    try {
                        mBtSocket =
                            device.createRfcommSocketToServiceRecord(MY_UUID);
                        mBtSocket.connect();
                        ConnectedTask task = new ConnectedTask();
                        task.execute(mBtSocket);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
            //发现完成
        } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED
            .equals(action)) {
            setProgressBarIndeterminateVisibility(false);
        }
    };
}

```

当应用程序启动后, 会首先对设备蓝牙的状态做一些基本的检查。如果 `BluetoothAdapter.getDefaultAdapter()` 返回 `null`, 表明设备不支持蓝牙, 应用程序无法继续使用。即使设备上有蓝牙, 它也必须是启用的, 这样应用程序才能使用它。如果蓝牙是禁用的, 推荐启用适配器的方法是向系统发送一个 `action` 为 `BluetoothAdapter.ACTION_REQUEST_ENABLE` 的 `Intent`。这样就会通知用户启用蓝牙。可以用 `enable()` 方法手动启用 `BluetoothAdapter`, 我们不推荐这种方法, 除非需要通过某个特别的方式来获得用户的许可。

验证过蓝牙可用后, 应用程序会等待用户输入。正如之前提到的, 这个示例可以将设

备设为两种状态：监听模式或者搜索模式。接下来看看这两种模式各自的工作方式。

监听模式

单击“Listen for Sharers”按钮开始对接入的连接进行监听。对于一个设备来说，如果想要接收未知设备的接入连接，那么该设备必须设置为可被发现的。应用程序会检查适配器的扫描模式是否等于 `SCAN_MODE_CONNECTABLE_DISCOVERABLE`，从而确定设备是否是可被发现的。如果适配器不满足这个要求，就会给系统发送一个 `Intent`，告诉用户应该让设备处于可发现状态，这和要求用户启用蓝牙的过程很相似。如果用户接受了该请求，`Activity` 就会返回设备处于可发现状态的时长，如果用户拒绝了该请求，`Activity` 会返回 `Activity.RESULT_CANCELED`。这个示例会在 `onActivityResult()` 中处理用户拒绝请求的行为，即终止应用。

如果用户启用了可发现状态或者设备已经处于可发现状态，就会创建并执行一个 `AcceptTask`。这个任务会为我们定义的服务的 `UUID` 创建监听器端口，通过这个端口等待连接请求。在收到有效的请求时，就会接受。然后应用会切换到 `Connected Mode`(已连接模式)。

在设备处于监听状态的过程中，蓝牙的名称会被设定为特定的值(`SEARCH_NAME`)来加速发现的过程(具体原因见后面的“搜索模式”一节)。连接建立后，适配器就会恢复到默认名称。

搜索模式

单击“Connect and Share”按钮通知应用程序开始搜索另一个想要连接的设备。这当中会首先启动一个蓝牙发现过程并且在一个 `BroadcastReceiver` 中处理搜索结果。当通过 `BluetoothAdapter.startDiscovery()` 开始一次发现后，以下两种情况下，`Android` 会通过广播进行异步回调：找到了另一个设备和发现过程完成。

私有的接收器 `mReceiver` 在 `Activity` 对用户可见时会随时进行注册，对于每一个新发现的设备，它都会收到一个广播。回忆一下，在讨论监听模式时，监听设备的设备名称被设置成一个唯一的值。在每次发现完成后，接收器会检测和当前名称所匹配的设备，并且在搜索到一个结果后会尝试进行连接。这对于发现过程的速度非常重要，因为验证一个设备是否可用的唯一途径就是将这个设备与一个特殊的服务 `UUID` 进行尝试性连接来看看操作是否成功。蓝牙连接过程属于重量级的并且很慢，应该在确保其他一切运行良好时才进行这个操作。

这种匹配设备的方式同样减轻了用户手动连接他想要连接的设备的设备的过程。应用程序会智能地寻找到同样运行这个应用程序并且处于监听模式的设备来完成传输。移除用户也意味着这个值是唯一且非常少见的，就是为了避免查找其他设备时，可能意外地具有相同的名称。

找到了匹配的设备后，就会停止发现过程(因为它同样是重量级并且会减缓连接过程)，然后连接到服务的 `UUID` 上。在连接成功后，应用程序就进入了已连接模式。

已连接模式

一旦连接成功，两种设备上的应用程序将创建一个 `ConnectedTask` 来发送和接收用户

联系信息。连接的 `BluetoothSocket` 会用一个 `InputStream` 和一个 `OutputStream` 进行数据传输。首先,电子邮件的文本字段的当前值包装后被写入到 `OutputStream`。然后,从 `InputStream` 读取接收远程设备的信息。最后,每个设备都需要将它接收的原始数据包装成一个单纯的字符串显示给用户。

`ConnectedTask.onPostExecute()` 方法的任务是向用户显示交流的结果,目前,是将接收的内容显示在一个 `Toast` 中。交流完成后,连接被关闭,两个设备都会进入相同的模式,准备进行下一次交流。

有关此主题的更多信息,可以查看 Android SDK 提供的 `BluetoothChat` 示例应用程序。这个应用程序很好地演示了如何使用一个长连接在设备之间发送聊天消息。

2. Android 之外的蓝牙

正如本节开始所描述的那样,除了手机和平板电脑,许多无线设备上也有蓝牙。在诸如蓝牙调制解调器和串行适配器这样的设备上同样有 `RFCOMM` 接口。Android 设备上创建点对点连接所使用的 API 同样可以用来连接其他嵌入式蓝牙设备从而实现对设备的监控和控制。

想要与这些嵌入式设备建立连接,关键是要获得它们所支持的 `RFCOMM` 服务的 UUID。和前面的示例一样,我们可以使用适当的 UUID 创建一个 `BluetoothSocket` 和传输数据。然而,就像在最后一个示例中那样,对于不知道的 UUID,我们必须有一种方法来发现并获得它。

SDK 就拥有这种能力,虽然在 Android 4.0.3 (API Level 15)之前它并不是 SDK 的开放部分。对于蓝牙设备来说,有两个方法能够提供这个信息:`fetchUidsWithSdp()`和`getUids()`。前者只会简单地返回在发现期间所找到设备的实例缓存,而后者则会异步连接设备并进行一个新的查询。正因为如此,在使用 `fetchUidsWithSdp()`时,必须注册一个 `BroadcastReceiver`,它将接收 action 值为 `BluetoothDevice.ACTION_UUID` 的 Intent 并发现 UUID 值。

发现一个 UUID

快速浏览一下 `BluetoothDevice` 的源代码(感谢 Android 的开放源码)会发现,已经存在一些方法能够返回一个远程设备的 UUID 信息,这些方法现在都是公共的 API 并且未来也不会变,但对于之前的 Android 版本,而且如果必要的话,还是可以使用反射使用它们的。最简单的就是一个名为 `getUids()`的同步(阻塞)方法,它返回一个 `ParcelUuid` 对象的数组,这些对象是指向每个服务的。下面这个示例就是使用反射的方法从远程设备的服务记录中读取 UUID:

```
public ParcelUuid servicesFromDevice(BluetoothDevice device) {
    try {
        Class cl = Class.forName("android.bluetooth.BluetoothDevice");
        Class[] par = {};
        Method method = cl.getMethod("getUids", par);
        Object[] args = {};
        ParcelUuid[] retval = (ParcelUuid[])method.invoke(device, args);
        return retval;
    } catch (Exception e) {
```

```

        e.printStackTrace();
        return null;
    }
}

```

你也许会以同样的方式调用 `fetchUuidsWithSdp()` 方法，对于之前的版本，它返回的 `Intent` 结构会有一些不同，因此对于之前的版本并不建议使用这个方法。

3.12 查询网络连接状态

3.12.1 问题

应用程序需要监控网络连接状态的变化。

3.12.2 解决方案

(API Level 1)

通过 `ConnectivityManager` 监控设备的网络连接状态。在移动应用程序设计时需要考虑的一个很重要的问题就是网络并不是随时都是连通的。随着人的移动，网络的速度和容量都在不断变化。正因为如此，使用网络资源的应用程序需要随时监测这些资源是否可访问，并在不能访问时通知用户。

除了连通性，`ConnectivityManager` 还能监测网络连接的类型。这样就能根据情况决定是否要下载大文件，如果用户处于漫游状态的话，下载大文件会使数据流量暴增。

3.12.3 实现机制

程序清单 3-37 封装了一个方法，可以把它放到你代码中来检查网络的连通性。

程序清单 3-37 `ConnectivityManager` Wrapper

```

public boolean isNetworkReachable() {
    ConnectivityManager mManager =
        (ConnectivityManager)context.getSystemService(
            Context.CONNECTIVITY_SERVICE);
    NetworkInfo current = mManager.getActiveNetworkInfo();
    if(current == null) {
        return false;
    }
    return (current.getState() == NetworkInfo.State.CONNECTED);
}

```

`ConnectivityManager` 可以检查网络的状态，这个封装方法可以方便地检查所有可能的网络通路。注意：如果没有可用的数据连接，`ConnectivityManager.getActiveNetworkInfo()` 会返回 `null`，所以首先要检查这种情况。如果有可用的网络，就可以检查其状态，可能的

返回值如下：

- DISCONNECTED
- CONNECTING
- CONNECTED
- DISCONNECTING

如果返回的状态是 **CONNECTED**，网络就是稳定的，可以用来访问远程资源。

当网络请求失败时最好检查网络的连通性，告知用户请求失败是网络问题。程序清单 3-38 中的示例演示了如何在网络访问失败时检查网络连通性。

程序清单 3-38 告知用户网络不通

```
try {
    //尝试访问网络资源时，如果失败可能会抛出 HttpResponseException 或
    //其他 IOException 异常
} catch (Exception e) {
    if( !isNetworkReachable() ) {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setTitle("No Network Connection");
        builder.setMessage("The Network is unavailable."
            + " Please try your request again later.");
        builder.setPositiveButton("OK",null);
        builder.create().show();
    }
}
```

判断连接类型

在某些情况下，应用程序还必须知道用户所连接的网络是否是按流量收费的，可以调用活动网络连接的 `NetworkInfo.getType()` 方法获取相关信息(参见程序清单 3-39)。

程序清单 3-39 ConnectivityManager 带宽检查

```
public boolean isWifiReachable() {
    ConnectivityManager mManager =
        (ConnectivityManager)context.getSystemService(
            Context.CONNECTIVITY_SERVICE);
    NetworkInfo current = mManager.getActiveNetworkInfo();
    if(current == null) {
        return false;
    }
    return (current.getType() == ConnectivityManager.TYPE_WIFI);
}
```

这个修改后的连通性检查能够判断用户是否连接到了 WiFi 网络，如果连接的是 WiFi 网络，通常就意味着网速比较快，而且流量是免费的。

3.13 使用 NFC 传输数据

3.13.1 问题

需要通过最少的设置实现两个设备间小数据包的快速传输。

3.13.2 解决方案

(API Level 16)

使用 NFC Beam API。NFC 通信起初是在 Android 2.3 上加入到 SDK 中的,在 4.0 中做了扩展,包括通过一个名为 Android Beam 的进程实现设备间短消息的无障碍传输。在 Android 4.1 中,又对 Beam API 做了完善,使之在两个设备间的数据传输方面更加成熟。

Android 4.1 中对它一个比较大的补充就是可以通过一些可选的连接实现大数据的传输。在发现设备和建立初始连接方面,NFC 表现非常优秀,但它的带宽较低,对于发送像全彩色图片这样的大数据包效率不是很高。以前,开发者可以使用 NFC 在两个设备间建立连接,但是在实际传输文件数据时需要手动选择第二种连接方式,如 WIFI 或者蓝牙。在 Android 4.1 中,框架层处理了整个过程,任何应用程序只需要调用一个 API 就可以通过可用的连接完成大文件的分享。

3.13.3 实现机制

根据想要推送内容的大小,有两种机制可以用来在两个设备间 Beam 数据。

1. 使用前台 Push 进行 Beam

如果使用 NFC 在设备间发送简单的内容,可以使用前台推送机制来创建一个 NfcMessage,它包含了一个或多个 NfcRecord 实例。程序清单 3-40 和 3-41 演示了创建一个简单的 NfcMessage 并推送到另一台设备上。

程序清单 3-40 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.nfcbeam"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="16" />

    <uses-permission android:name="android.permission.NFC" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="NfcBeam">
```

```

<activity
    android:name=".NfcActivity"
    android:label="NfcActivity"
    android:launchMode="singleTop">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.nfc.action.NDEF_DISCOVERED" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType=
            "application/com.example.androidrecipes.beamtext"/>
    </intent-filter>
</activity>
</application>
</manifest>

```

首先需要注意的是在使用 NFC 服务时需要 `android.permission.NFC` 权限。另外，我们的 Activity 中添加了一个自定义的 `<intent-filter>`。这样 Android 就可以知道哪个应用程序应该响应它所收到的内容。

程序清单 3-41 生成一个 NFC 前台推送的 Activity

```

public class NfcActivity extends Activity implements
    CreateNdefMessageCallback, OnNdefPushCompleteCallback {
    private static final String TAG = "NfcBeam";
    private NfcAdapter mNfcAdapter;
    private TextView mDisplay;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mDisplay = new TextView(this);
        setContentView(mDisplay);

        //检查 NFC 适配器是否可用
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            mDisplay.setText("NFC is not available on this device.");
        } else {
            //注册回调来设置 NDEF 消息。这样做可以使 Activity 处于前台时，
            //NFC 数据推送处于激活状态
            mNfcAdapter.setNdefPushMessageCallback(this, this);
            //注册回调来监听消息发送成功
            mNfcAdapter.setOnNdefPushCompleteCallback(this, this);
        }
    }

    @Override

```

```

public void onResume() {
    super.onResume();
    //检查是否是一个 Beam 启动了这个 Activity
    if (NfcAdapter.ACTION_NDEF_DISCOVERED
        .equals(getIntent().getAction())) {
        processIntent(getIntent());
    }
}

@Override
public void onNewIntent(Intent intent) {
    //这之后会调用 onResume 来处理这个 Intent
    setIntent(intent);
}

void processIntent(Intent intent) {
    Parcelable[] rawMsgs =
        intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
    //beam 期间只发送了一个消息
    NdefMessage msg = (NdefMessage) rawMsgs[0];
    //记录 0 包含了 MIME 类型
    mDisplay.setText(new String(msg.getRecords()[0].getPayload()));
}

@Override
public NdefMessage createNdefMessage(NfcEvent event) {
    String text =
        String.format("Sending A Message From Android Recipes at %s",
            DateFormat.getTimeFormat(this).format(new Date()));
    NdefMessage msg = new NdefMessage(NdefRecord.createMime(
        "application/com.example.androidrecipes.beamtext",
        text.getBytes()));
    return msg;
}

@Override
public void onNdefPushComplete(NfcEvent event) {
    //这个回调是在一个绑定线程上执行的，不要在这个方法中直接更新 UI
    Log.i(TAG, "Message Sent!");
}
}

```

这个示例针对的是 NFC 推送的发送和接收，因此两个设备上都应该安装相同的应用程序：一个负责发送数据，一个负责接收数据。Activity 通过 NfcAdapter 的 `setNdefPushMessageCallback()` 方法将自己注册为可进行前台推送。这其中同时做了两件事情。在传输开始时，它告诉 NFC 服务去调用这个 Activity 来接收它需要发送的信息，同时在 Activity 处于前台时，会激活 NFC 推送。另外，还有一个类似的方法叫做 `setNdefPushMessage()`，该方法只接收信息，但不会回调。

这个回调方法只创建了一个 NdefMessage，NdefMessage 则只包含了一个 NDEF MIME

record(通过 `NdefRecord.createMime()` 方法创建的)。MIME record 是一种传递应用程序所指定数据的简单方式。创建方法包含两个参数，一个是用来指定 MIME 类型的字符串，一个是原始数据的 byte 数组。传递的信息可以是任何数据，如一个字符串或者一个小图片；应用程序负责该数据的包装和解析。注意，这里的 MIME 类型和 manifest 中 `<intent-filter>` 定义的类型是一样的。

想要推送执行的话，负责发送设备的 Activity 必须处于前台激活状态，接收的设备也不能是锁屏状态。当用户同时触摸两个设备时，负责发送设备的屏幕上会显示“Touch to beam”，这时再单击一下屏幕就会把消息发送到另一台设备上。一旦接收到消息，负责接收设备上的应用程序就会启动，并且会触发发送设备的 `onNdefPushComplete()` 回调方法。

在负责接收的设备上，会使用 `ACTION_NDEF_DISCOVERED` 的 Intent 来启动 Activity，因此我们的示例会检查 `NdefMessage` 的 Intent 并且解析出其中的负载数据，即将 byte 数组转换为一个字符串。这种使用先匹配 Intent 然后发送 NFC 数据的方式非常灵活，但是有些时候你可能需要显式地调用应用程序。这时候就需要 Android Application Record 出马了。

2. Android Application Record

应用程序可以在一个 `NdefMessage` 中添加一个额外的 `NdefRecord`，它可以引导接收设备去调用一个指定的包名。想要在之前的示例中实现它，我们只需要这样来简单地修改一下 `CreateNdefMessageCallback` 方法。

```
@Override
public NdefMessage createNdefMessage(NfcEvent event) {
    String text = String.format("Sending A Message From Android Recipes at %s",
        DateFormat.getTimeFormat(this).format(new Date()));
    NdefMessage msg = new NdefMessage(NdefRecord.createMime(
        "application/com.example.androidrecipes.beamtext", text.getBytes()),
        NdefRecord.createApplicationRecord("com.examples.nfcbeam"));
    return msg;
}
```

加了 `NdefRecord.createApplicationRecord()` 这个额外的参数后，现在可以保证推送消息就只会启动我们的 `com.examples.nfcbeam` 包。消息的第一个记录还是原来的文本消息，所以我们对接收的消息的解析过程是不用变的。

3. Beam 大数据

在本节开头，我们提到了最好不要使用 NFC 发送大数据。尽管如此，Android Beam 还是有能力处理它的。程序清单 3-42 和 3-43 演示了使用 Beam 来发送大型图片文件。

程序清单 3-42 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.nfcbeam"
    android:versionCode="1"
    android:versionName="1.0" >
```

```

<uses-sdk
    android:minSdkVersion="16"
    android:targetSdkVersion="16" />

<uses-permission android:name="android.permission.NFC" />
<application
    android:icon="@drawable/ic_launcher"
    android:label="NfcBeam">
    <activity
        android:name=".BeamActivity"
        android:label="BeamActivity"
        android:launchMode="singleTop">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <data android:mimeType="image/*" />
        </intent-filter>
    </activity>
</application>

</manifest>

```

程序清单 3-43 Beam 一个图片文件的 Activity

```

public class BeamActivity extends Activity implements
    CreateBeamUriCallback, OnNdefPushCompleteCallback {
    private static final String TAG = "NfcBeam";
    private static final int PICK_IMAGE = 100;

    private NfcAdapter mNfcAdapter;
    private Uri mSelectedImage;

    private TextView mUriName;
    private ImageView mPreviewImage;

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

        mUriName = (TextView) findViewById(R.id.text_uri);
        mPreviewImage = (ImageView) findViewById(R.id.image_preview);

        //检查 NFC 适配器是否可用
        mNfcAdapter = NfcAdapter.getDefaultAdapter(this);
        if (mNfcAdapter == null) {
            mUriName.setText("NFC is not available on this device.");
        } else {

```

```

        //注册回调来设置 NDEF 消息
        mNfcAdapter.setBeamPushUriCallback(this, this);
        //注册回调来监听消息发送成功
        mNfcAdapter.setOnNdefPushCompleteCallback(this, this);
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if (requestCode == PICK_IMAGE && resultCode == RESULT_OK
        && data != null) {
        mUriName.setText( data.getData().toString() );
        mSelectedImage = data.getData();
    }
}

@Override
public void onResume() {
    super.onResume();
    //检查 Activity 是否由于 Android Beam 而启动的
    if (Intent.ACTION_VIEW.equals(getIntent().getAction())) {
        processIntent(getIntent());
    }
}

@Override
public void onNewIntent(Intent intent) {
    //这之后会调用 onResume 来处理这个 Intent
    setIntent(intent);
}

void processIntent(Intent intent) {
    Uri data = intent.getData();
    if(data != null) {
        mPreviewImage.setImageURI(data);
    } else {
        mUriName.setText("Received Invalid Image Uri");
    }
}

public void onSelectClick(View v) {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("image/*");
    startActivityForResult(intent, PICK_IMAGE);
}

@Override
public Uri[] createBeamUri(NfcEvent event) {
    if (mSelectedImage == null) {

```

```

        return null;
    }
    return new Uri[] {mSelectedImage};
}

@Override
public void onNdefPushComplete(NfcEvent event) {
    //这个回调是在一个绑定线程上执行的，不要在这个方法中直接更新 UI
    //这里最好告诉用户不需要再把手机放在一起了
    Log.i(TAG, "Push Complete!");
}
}

```

这个示例使用了 `CreateBeamUriCallback`，它允许应用程序构造一个 `Uri` 实例的数组，这些 `Uri` 指向了你要发送的内容。Android 首先会通过 NFC 建立一个初始连接，然后再寻找一种合适的连接方式如蓝牙或者 WiFi 来完成大文件的传输。

在本例中，接收设备上的数据是通过系统标准的 `Intent.ACTION_VIEW` action 启动的，因此没必要在两台设备上都加载应用程序。尽管如此，我们的应用程序还是对 `ACTION_VIEW` 进行了过滤，这样的话如果接收设备愿意，可以使用它来浏览接收的图片。

这里，会要求用户从他的设备中选择一张图片来 Beam，一旦选定后，图片的 `Uri` 会显示出来。一旦用户单击设备到另一个设备，屏幕同样会显示“Touch to beam”（参见图 3-4），当再次单击屏幕时传输就开始了。

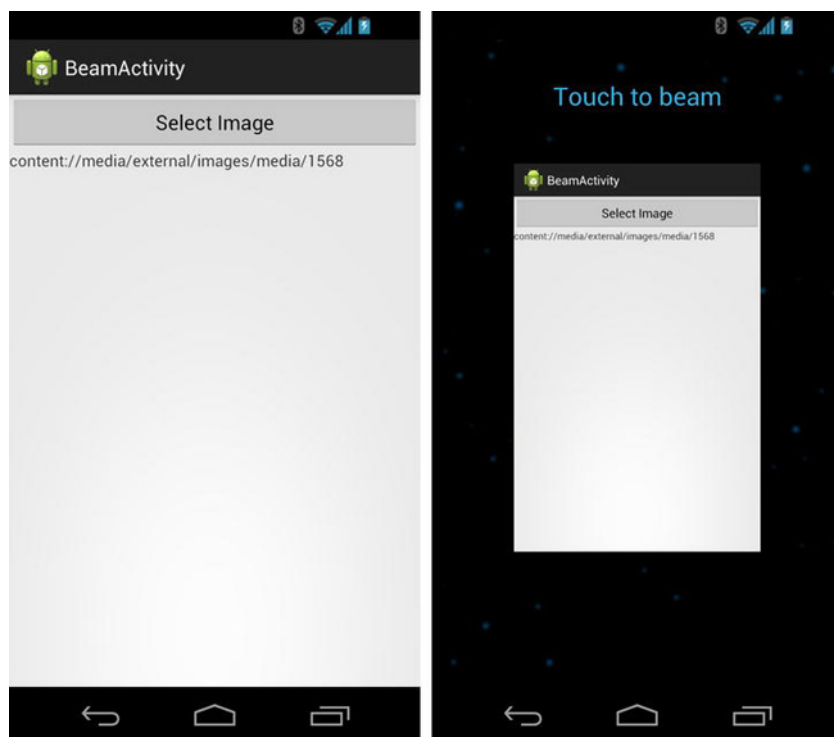


图 3-4 触摸 Activity 来激活 Beam

当传输过程中有关 NFC 的部分完成后，就会在发送设备上调用 `onNdefPushComplete()`

方法。这时，传输过程就转移到其他类型的连接上，因此用户也就不需要再把手机放在一起了。

在文件传输时，接收设备会在系统窗口的顶部显示一个进度通知，当传输完成时，用户可以单击该通知来浏览内容。如果选择了我们的应用程序作为浏览器，图片就会显示在应用程序的 `ImageView` 上。为你的应用程序注册这种通用的 `Intent` 可能有一个缺点，那就是设备上所有的应用程序就可以用你的应用程序浏览图片，所以定义过滤器时要谨慎。

3.14 USB 连接

3.14.1 问题

应用程序需要与 USB 设备进行通信来控制或者传输数据。

3.14.2 解决方案

(API Level 12)

对于拥有 USB 主机电路的设备，Android 已经内置了对它的支持，可以与已经连接的 USB 设备进行通信。`USBManager` 是一项系统服务，可以让应用程序访问任何通过 USB 连接的外部设备，接下来我们将看一下在应用程序中如何使用这个服务来建立一个连接。

```
<uses-feature android:name="android.hardware.usb.host" />
```

设备上的 USB 主机电路已经越来越普及，但是还是很稀少的。刚开始，只有平板电脑设备拥有这种能力，但随着科技的快速发展，在商用 Android 手机上它也有可能很快成为一个通用的接口。正因为如此，需要在你的应用程序的 `manifest` 中包含以下元素：

```
<uses-feature android:name="android.hardware.usb.host" />
```

这样只有真正拥有相应硬件的设备，才可以使用你的应用程序。

Android 提供的 API 和 USB 规范几乎一样，并没有更多更深入的知识。这就意味着如果想要使用这些 API，你至少需要了解一些 USB 的基础知识以及设备间是如何通信的。

USB 概述

在查看 Android 是如何与 USB 设备进行交互的示例之前，让我们花点时间定义一些 USB 术语。

- 端点：USB 设备的最小构件。应用程序最终就是通过连接这些端点发送和接收数据的。主要分为 4 种类型：
 - 控制传输：用于配置和状态命令。每个设备至少有一个控制端点，即“端点 0”，它不会关联任何接口。
 - 中断传输：用于小量的、高优先级的控制命令。
 - 批量传输：用于传输大数据。通常都是双向成对出现的(1 IN 和 1 OUT)。

- 同步传输：用于实时数据传输，如音频。撰写本书时最新的 Android SDK 还不支持这个功能。
- 接口：端点的集合，用来表示一个“逻辑”设备。
 - 多个物理 USB 设备对于主机来说可以呈现为多个逻辑设备，即通过暴露接口来标识。
- 配置：一个或多个接口的集合。USB 强制规定一个设备在某一特定时间只能有一个配置是激活的。事实上，多数设备也就只有一个配置，并把它作为设备的操作模式。

3.14.3 实现机制

程序清单 3-44 和 3-45 演示了使用 `UsbManager` 来检查通过 USB 连接的设备以及使用控制传输来进一步查询配置的示例。

程序清单 3-44 `res/layout/main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/button_connect"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Connect"
        android:onClick="onConnectClick" />
    <TextView
        android:id="@+id/text_status"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/text_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

程序清单 3-45 USB 主机上查询设备的 Activity

```
public class USBActivity extends Activity {
    private static final String TAG = "UsbHost";

    TextView mDeviceText, mDisplayText;
    Button mConnectButton;

    UsbManager mUsbManager;
    UsbDevice mDevice;
```

```

PendingIntent mPermissionIntent;

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

    mDeviceText = (TextView) findViewById(R.id.text_status);
    mDisplayText = (TextView) findViewById(R.id.text_data);
    mConnectButton = (Button) findViewById(R.id.button_connect);

    mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
}

@Override
protected void onResume() {
    super.onResume();
    mPermissionIntent =
        PendingIntent.getBroadcast(this, 0,
            new Intent(ACTION_USB_PERMISSION), 0);
    IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
    registerReceiver(mUsbReceiver, filter);

    //检查当前连接的设备
    updateDeviceList();
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(mUsbReceiver);
}

public void onConnectClick(View v) {
    if (mDevice == null) {
        return;
    }
    mDisplayText.setText("---");

    //这里如果用户已经授权会立即发送 ACTION_USB_PERMISSION 广播,
    //否则会向用户显示一个授权对话框
    mUsbManager.requestPermission(mDevice, mPermissionIntent);
}

/*
 * 捕捉用户权限响应的接收器, 在和已经连接的设备进行真正的交互时是需要这些权限的
 */
private static final String ACTION_USB_PERMISSION =
    "com.android.recipes.USB_PERMISSION";
private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {

```

```

        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            if (ACTION_USB_PERMISSION.equals(action)) {
                UsbDevice device = (UsbDevice) intent.getParcelableExtra(
                    UsbManager.EXTRA_DEVICE);

                if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED,
                    false) && device != null) {
                    //查询设备的描述符
                    getDeviceStatus(device);
                } else {
                    Log.d(TAG, "permission denied for device " + device);
                }
            }
        }
    };

    //类型: 表示读取还是写入
    //与 USB_ENDPOINT_DIR_MASK 进行匹配判断是 IN 还是 OUT
    private static final int REQUEST_TYPE = 0x80;
    //请求: GET_CONFIGURATION_DESCRIPTOR = 0x06
    private static final int REQUEST = 0x06;
    //值: 描述符类型(高)和索引值(低)
    // Configuration Descriptor = 0x2
    // Index = 0x0 (第一个配置)
    private static final int REQ_VALUE = 0x200;
    private static final int REQ_INDEX = 0x00;
    private static final int LENGTH = 64;

    /*
     * 初始化一个控制传输来请求设备的第一个配置描述符
     */
    private void getDeviceStatus(UsbDevice device) {
        UsbDeviceConnection connection = mUsbManager.openDevice(device);
        //为传入的数据创建一个足够大的缓冲区
        byte[] buffer = new byte[LENGTH];
        connection.controlTransfer(REQUEST_TYPE, REQUEST, REQ_VALUE, REQ_INDEX,
            buffer, LENGTH, 2000);
        //将接收到的数据解析为一个描述符
        String description = parseConfigDescriptor(buffer);

        mDisplayText.setText(description);
        connection.close();
    }

    /*
     * 按照 USB 规范解析 USB 配置描述符响应信息。返回一个可打印的连接设备的信息。
     */
    private static final int DESC_SIZE_CONFIG = 9;
    private String parseConfigDescriptor(byte[] buffer) {

```

```

StringBuilder sb = new StringBuilder();
//解析配置描述符头信息
int totalLength = (buffer[3] & 0xFF) << 8;
totalLength += (buffer[2] & 0xFF);
//接口数量
int numInterfaces = (buffer[5] & 0xFF);
//配置的属性
int attributes = (buffer[7] & 0xFF);
//电量递增 2mA
int maxPower = (buffer[8] & 0xFF) * 2;

sb.append("Configuration Descriptor:\n");
sb.append("Length: " + totalLength + " bytes\n");
sb.append(numInterfaces + " Interfaces\n");
sb.append(String.format("Attributes:%s%s%s\n",
    (attributes & 0x80) == 0x80 ? " BusPowered" : "",
    (attributes & 0x40) == 0x40 ? " SelfPowered" : "",
    (attributes & 0x20) == 0x20 ? " RemoteWakeup" : ""));
sb.append("Max Power: " + maxPower + "mA\n");

//描述符的剩余部分为接口和端点信息
int index = DESC_SIZE_CONFIG;
while (index < totalLength) {
    //读取长度和类型
    int len = (buffer[index] & 0xFF);
    int type = (buffer[index+1] & 0xFF);
    switch (type) {
        case 0x04: //接口描述符
            int intfNumber = (buffer[index+2] & 0xFF);
            int numEndpoints = (buffer[index+4] & 0xFF);
            int intfClass = (buffer[index+5] & 0xFF);

            sb.append(String.format("- Interface %d, %s, %d Endpoints\n",
                intfNumber, nameForClass(intfClass), numEndpoints));
            break;
        case 0x05: //端点描述符
            int endpointAddr = ((buffer[index+2] & 0xFF));
            //端点号为低 4 位
            int endpointNum = (endpointAddr & 0x0F);
            //方向为高位
            int direction = (endpointAddr & 0x80);

            int endpointAttrs = (buffer[index+3] & 0xFF);
            //类型为低两位
            int endpointType = (endpointAttrs & 0x3);

            sb.append(String.format("-Endpoint %d, %s %s\n",
                endpointNum,
                nameForEndpointType(endpointType),
                nameForDirection(direction) ));
    }
}

```

```

        break;
    }
    //继续下一个描述符
    index += len;
}

return sb.toString();
}

private void updateDeviceList() {
    HashMap<String, UsbDevice> connectedDevices = mUsbManager
        .getDeviceList();
    if (connectedDevices.isEmpty()) {
        mDevice = null;
        mDeviceText.setText("No Devices Currently Connected");
        mConnectButton.setEnabled(false);
    } else {
        StringBuilder builder = new StringBuilder();
        for (UsbDevice device : connectedDevices.values()) {
            //打开最后一个(如果有多个)检测到的设备
            mDevice = device;
            builder.append(readDevice(device));
            builder.append("\n\n");
        }
        mDeviceText.setText(builder.toString());
        mConnectButton.setEnabled(true);
    }
}

/*
 * 遍历所有已经连接的设备的端点和接口。
 * 这里不涉及权限，在尝试连接真实设备之前这些都是“publiclyavailable”的。
 */
private String readDevice(UsbDevice device) {
    StringBuilder sb = new StringBuilder();
    sb.append("Device Name: " + device.getDeviceName() + "\n");
    sb.append(String.format(
        "Device Class: %s -> Subclass: 0x%02x -> Protocol: 0x%02x\n",
        nameForClass(device.getDeviceClass()),
        device.getDeviceSubclass(), device.getDeviceProtocol()));

    for (int i = 0; i < device.getInterfaceCount(); i++) {
        UsbInterface intf = device.getInterface(i);
        sb.append(String.format("++-Interface %d Class: %s ->"
            + "Subclass: 0x%02x -> Protocol: 0x%02x\n",
            intf.getId(),
            nameForClass(intf.getInterfaceClass()),
            intf.getInterfaceSubclass(),
            intf.getInterfaceProtocol()));

        for (int j = 0; j < intf.getEndpointCount(); j++) {

```

```

        UsbEndpoint endpoint = intf.getEndpoint(j);
        sb.append(String.format(" ---Endpoint %d: %s %s\n",
                                endpoint.getEndpointNumber(),
                                nameForEndpointType(endpoint.getType()),
                                nameForDirection(endpoint.getDirection())));
    }
}

return sb.toString();
}

/* 辅助方法, 用来将 USB 常量转换为可读性更强的名字 */

private String nameForClass(int classType) {
    switch (classType) {
        case UsbConstants.USB_CLASS_APP_SPEC:
            return String.format("Application Specific 0x%02x", classType);
        case UsbConstants.USB_CLASS_AUDIO:
            return "Audio";
        case UsbConstants.USB_CLASS_CDC_DATA:
            return "CDC Control";
        case UsbConstants.USB_CLASS_COMM:
            return "Communications";
        case UsbConstants.USB_CLASS_CONTENT_SEC:
            return "Content Security";
        case UsbConstants.USB_CLASS_CSCID:
            return "Content Smart Card";
        case UsbConstants.USB_CLASS_HID:
            return "Human Interface Device";
        case UsbConstants.USB_CLASS_HUB:
            return "Hub";
        case UsbConstants.USB_CLASS_MASS_STORAGE:
            return "Mass Storage";
        case UsbConstants.USB_CLASS_MISC:
            return "Wireless Miscellaneous";
        case UsbConstants.USB_CLASS_PER_INTERFACE:
            return "(Defined Per Interface)";
        case UsbConstants.USB_CLASS_PHYSICA:
            return "Physical";
        case UsbConstants.USB_CLASS_PRINTER:
            return "Printer";
        case UsbConstants.USB_CLASS_STILL_IMAGE:
            return "Still Image";
        case UsbConstants.USB_CLASS_VENDOR_SPEC:
            return String.format("Vendor Specific 0x%02x", classType);
        case UsbConstants.USB_CLASS_VIDEO:
            return "Video";
        case UsbConstants.USB_CLASS_WIRELESS_CONTROLLER:
            return "Wireless Controller";
        default:
    }
}

```

```

        return String.format("0x%02x", classType);
    }
}

private String nameForEndpointType(int type) {
    switch (type) {
        case UsbConstants.USB_ENDPOINT_XFER_BULK:
            return "Bulk";
        case UsbConstants.USB_ENDPOINT_XFER_CONTROL:
            return "Control";
        case UsbConstants.USB_ENDPOINT_XFER_INT:
            return "Interrupt";
        case UsbConstants.USB_ENDPOINT_XFER_ISOC:
            return "Isochronous";
        default:
            return "Unknown Type";
    }
}

private String nameForDirection(int direction) {
    switch (direction) {
        case UsbConstants.USB_DIR_IN:
            return "IN";
        case UsbConstants.USB_DIR_OUT:
            return "OUT";
        default:
            return "Unknown Direction";
    }
}
}

```

当 Activity 首次进入前台时, 它注册一个自定义动作(稍后会详细探讨)的 `BroadcastReceiver`, 并且通过 `UsbManager.getDeviceList()` 方法来查询当前已连接设备的列表, 该方法会返回一个 `UsbDevice` 的 `HashMap`, 然后就可以遍历这个 `HashMap`。对于每个连接的设备, 我们会查询它的接口和端点, 并且会构建需要显示给用户的每个设备的描述信息。然后, 我们会在用户界面上打印这些信息。

注意:

就目前来说, 这个应用程序不需要在 `manifest` 中声明任何权限。对于只是简单地查询连接到主机上设备的信息, 并不需要声明权限。

如你所见, 对于你想与之通信的连接设备, `UsbManager` 提供的 API 可以获得你想要的所有信息。一些标准的定义如设备种类、端点类型和传输方向也都在 `UsbConstants` 做了定义, 所以不需要你自己定义就可以匹配你想要的类型。

那么, 为什么要注册一个 `BroadcastReceiver` 呢? 在用户按下屏幕上的 `Connect` 按钮后, 这个示例的剩余部分做了相应的响应。这时候我们想要与连接设备进行真正的交互, 这时候就需要用户权限。这里, 当用户单击按钮后, 会调用 `UsbManager.requestPermission()` 来询问用户是否可以连接。如果还没有授权相应的权限, 用户会看到一个询问授权连接的对

话框。

如果选项 yes, 传入方法的 PendingIntent 就会被触发。我们的示例中, 这个 Intent 是通过自定义动作字符串来广播的, 此时会触发 BroadcastReceiver 的 onReceive()方法; 接下来任何的 requestPermission()的调用都会立即触发这个接收器。在接收器内部, 我们会检查确保结果是一个授权响应并通过 UsbManager.openDevice()打开与设备的连接, 如果连接成功则会返回一个 UsbDeviceConnection 实例。

对于一个有效的连接, 我们会通过一个控制传输来请求设备的配置描述符从而得到设备更加详细的信息。控制传输一般都是通过设备的“端点 0”来请求的。配置描述符包含配置的信息以及每个接口和端点的信息, 因此它的长度是可变的。我们则分配了一个合适大小的缓冲区来保证可以得到所有的信息。

controlTransfer()返回后, 缓冲区中已经填好了响应数据。接下来应用程序会处理这些数据, 得到设备的一些详细信息, 例如设备的最大能耗以及设备是使用 USB 供电(总线供电)还是其他方式供电(自供电)。这个示例只是从这些标识符中解析出一小部分有用的信息。同样, 所有解析出的数据都会放到一个字符串中并显示在用户界面上。

第一节中从框架 API 中读取的信息和第二节中直接从设备中读取的信息都是一样的, 并且按照 1:1 的比例通过两个文本报告显示在用户屏幕上。需要注意的一点就是只有在设备连接上时该应用程序才会工作: 在应用程序运行时才连接的设备, 应用程序并不会得到通知。下节将讨论如何处理这种场景。

获取设备连接时的通知

想要 Android 在设备连接时可以通知你的应用程序, 需要在 manifest 中通过<intent-filter>注册要匹配的设备类型。程序清单 3-46 和 3-47 演示了这个过程。

程序清单 3-46 AndroidManifest.xml 中的部分代码

```
<activity
    android:name=".USBActivity"
    android:label="@string/title_activity_usb" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action
            android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
    </intent-filter>

    <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
        android:resource="@xml/device_filter" />
</activity>
```

程序清单 3-47 res/xml/device_filter.xml

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

```
<resources>
    <usb-device vendor-id="5432" product-id="9876" />
</resources>
```

能够处理设备连接的 Activity 添加了一个名为 USB_DEVICE_ATTACHED 动作字符串的过滤器和描述想要处理的设备的 XML 元数据信息。可以在<usb-device>中添加很多的设备属性字段从而过滤哪些连接事件可以通知到应用程序。

- vendor-id
- product-id
- class
- subclass
- protocol

必要时，可以定义以上的很多属性来适应你的应用程序。例如，如果只想和某一个特定设备进行通信，或许可以像示例代码一样设置 vendor-id 和 product-id。如果想匹配某一类型(例如，所有的大数据量存储设备)的设备，或许只需要定义 class 属性即可。甚至于可以不定义任何属性，这样应用程序就可以匹配所有连接的设备。

3.15 小结

在当前这个一切皆 Web 的时代，将 Android 应用程序与网络和 Web 服务连接起来是提升用户体验的好方法。Android 中连接网络和其他远程主机的框架使得添加这类功能变得很方便。本章研究了如何将 Web 标准加入到应用程序中，在原生环境中用 HTML 和 JavaScript 实现用户交互。本章还介绍了如何用 Android 从远程服务器下载内容并在应用程序中使用它们。本章还指出 Web 服务器并不是唯一可以连接的远程主机，还可以用蓝牙和短信实现设备间的直接通信。在第 4 章中，我们将探讨如何使用 Android 提供的工具与设备的硬件资源进行交互。

第 4 章

实现设备硬件交互与媒体交互

在应用程序中整合设备硬件的功能可以向用户提供只有移动平台才具备的独特用户体验。使用麦克风和摄像头采集媒体，用户就可以通过照片或录音实现个性化交流。整合传感器和位置数据可以帮助你开发能够回答“我是谁？”和“我看到了什么？”这种问题的应用程序。

本章将介绍 Android 提供的关于位置、媒体和传感器的 API，这些 API 可以使你的应用程序具有移动平台上特有的功能。

4.1 整合设备位置

4.1.1 问题

想要在应用程序中使用设备的定位功能报告当前的物理位置。

4.1.2 解决方案

(API Level 1)

可以使用 Android LocationManager 提供的后台服务。移动应用程序一个最大的好处就是可以根据用户当前的位置为用户提供各种信息。应用程序既可以定时查询 LocationManager 来更新设备的位置信息，也可以在发现设备移动了一段距离后再更新设备的位置信息。

使用 Android 位置服务时，需要注意设备的电量以及尊重用户的意愿。通过设备的 GPS 获取详细位置信息是非常耗电的，如果持续的话，很快就会耗光用户设备的电量。因此，Android 允许用户关闭特定位置信息的数据源，例如 GPS。在应用程序判断如何获取位置信息时，一定要注意检查这些设置。

每个位置信息源都有精度上的差异。GPS 返回的信息更加精确(误差为几米)，但需要的时间更长并且更耗电，而网络位置通常精确到几千米，但速度更快而且省电。可以根据应

用程序的需求选择使用哪种信息源；如果应用程序只想显示当前城市的信息，可能用 GPS 就没什么必要了。

重点：

使用位置服务时，记住需要在应用程序的 manifest 中声明 `android.permission.ACCESS_COARSE_LOCATION` 或 `android.permission.ACCESS_FINE_LOCATION` 权限。如果声明了 `android.permission.ACCESS_FINE_LOCATION` 权限，就不必声明另一个了，因为它已经包含了模糊位置服务的权限。

4.1.3 实现机制

当在 Activity 或 Service 中针对用户位置进行简单的监控时，我们需要考虑以下几点：

(1) 判断所需的位置信息源是否可用。如果不可用的话，决定是要求用户启用它还是尝试其他的信息源。

(2) 用适当的最小距离和更新时间间隔注册设备位置的变化情况。

(3) 当不需要时，及时取消对设备位置变化情况的注册，以节约电量。

程序清单 4-1 中，在 Activity 对用户可见时，我们注册该 Activity 来监听位置的变化并且将位置信息显示在屏幕上。

程序清单 4-1 监控位置变化的 Activity

```
public class MyActivity extends Activity {

    LocationManager manager;
    Location currentLocation;

    TextView locationView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        locationView = new TextView(this);
        setContentView(locationView);

        manager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    }

    @Override
    public void onResume() {
        super.onResume();
        if(!manager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            //询问用户启用 GPS
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Location Manager");
            builder.setMessage("We want to use your location, but GPS is
```

```

disabled.\n"+"Would you like to change these settings now?");
builder.setPositiveButton("Yes",
    new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            //打开设置, 允许用户进行修改
            Intent i =new
                Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            startActivity(i);
        }
    });
builder.setNegativeButton("No",
    new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {
            //没有位置服务则退出 Activity
            finish();
        }
    });
builder.create().show();
}

//如果有的话, 获得缓存的位置信息
currentLocation =
    manager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
updateDisplay();
//注册位置变化
int minTime = 5000;
float minDistance = 0;
manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    minTime, minDistance, listener);
}

@Override
public void onPause() {
    super.onPause();
    manager.removeUpdates(listener);
}

//更新 TextView
private void updateDisplay() {
    if(currentLocation == null) {
        locationView.setText("Determining Your Location...");
    } else {
        locationView.setText(String.format("Your Location:\n%.2f, %.2f",
            currentLocation.getLatitude(),
            currentLocation.getLongitude()));
    }
}
}

```

```

//处理位置回调事件
private LocationListener listener = new LocationListener() {

    @Override
    public void onLocationChanged(Location location) {
        currentLocation = location;
        updateDisplay();
    }

    @Override
    public void onProviderDisabled(String provider) { }

    @Override
    public void onProviderEnabled(String provider) { }

    @Override
    public void onStatusChanged(String provider, int status,
        Bundle extras) { }
};
}

```

这个示例通过设备的 GPS 来获得位置信息的更新。因为它是 Activity 的核心功能，所以会在每次 resume 时检查 LocationManager.GPS_PROVIDER 是否还是启用的。如果由于某种原因，用户禁用了这项功能，我们会询问用户是否需要启用该功能。应用程序不能替用户完成这项工作，如果用户同意的话，我们会通过动作作为 Settings.ACTION_LOCATION_SOURCE_SETTINGS 的 Intent 启动一个设置界面，在该界面中用户可以启用 GPS。

提示：

如果使用 Android 模拟器测试应用程序的话，应用程序将不能从任何的系统 provider 中接收真实的位置数据信息。通过 SDK 的 DDMS 工具，可以手动地为 GPS_PROVIDER 注入位置改变事件。

当 GPS 已经激活并且可用时，Activity 注册了一个 LocationListener 来接收位置更新时的通知。LocationManager.requestLocationUpdates()方法有 4 个参数，除了位置信息 provider 类型和目标监听器外，还有两个主要的参数：

- minTime
 - 两次更新时间的更小间隔，单位为毫秒。
 - 将它设置为非零值，可以让位置 provider 在两次更新间处于休息状态。
 - 这个参数可以减少耗电量，所以不应该把这个值设置为低于最小可接受更新频率的值。
- minDistance
 - 设备必须移动超过这个距离时，才会进行更新，单位为米。
 - 将它设置为非零值，在设备的移动距离大于该参数之前是不会更新位置信息的。

在本例中，要求更新的事件间隔不大于 5 秒，但没有限定设备的移动距离。当收到更新时，就会调用注册的监听器的 onLocationChanged()方法。注意当位置信息 provider 发生

变化时，也会通知 `LocationListener`，但这里并没有用到这些回调方法。

提示：

如果在 `Service` 或其他后台操作中接收位置更新，Google 建议最小时间间隔不应小于 60 000(60 秒)。

本例会时刻记录所接收的最新位置信息。起初，这个值存放于位置信息 `provider` 的缓存中，调用 `getLastKnownLocation()` 即可获得。如果位置信息 `provider` 没有缓存位置信息，该方法就会返回 `null`。在每次收到更新时，都会重置位置，用户界面也会显示更新后的位置信息。

4.2 地图位置

4.2.1 问题

需要在地图上显示一个或多个位置。

4.2.2 解决方案

(API Level 1)

向用户显示地图最简单的方式就是用位置数据创建一个 `Intent` 并把它传递给 Android 系统来启动一个地图应用程序。在后面的章节中你将深入了解这个方法以完成各种任务。另外，Google Maps API SDK 扩展组件中的 `MapView` 和 `MapActivity` 可以在你的应用程序中嵌入地图。

Maps API 是核心 SDK 的扩展模块，不过它们是捆绑在一起的。如果还没有 Google APIs SDK 的话，打开 SDK 管理器，在“Thirdparty Add-ons”下面可以找到针对各个 API level 的包。

想要在应用程序中使用 Maps API，必须先要在 Google 中得到一个 API 密钥。这个密钥是通过应用程序签署的私钥构建的。没有 API 密钥的话，地图或许可以使用，但应用程序无法获得地图标题。

注意：

关于 SDK 和获取 API 密钥的更多信息可以访问 <http://code.google.com/android/add-ons/google-apis/mapkey.html>。注意在调试模式(例如从 IDE 中运行应用程序)时，Android 对所有的应用程序使用的是一样的签名密钥，所以在测试阶段，你开发的所有应用程序使用一个密钥即可。

如果在模拟器上测试你的代码，模拟器的目标平台必须是包含 Google API 的 SDK，这样地图才能正常运行。如果是通过命令行创建模拟器，目标的名称就是“GoogleInc.:Google APIs:X”，其中 X 是 API 的版本。如果在 IDE(例如 Eclipse)中创建模拟器，目标的命名方式也是类似的：Google APIs (Google Inc.)——X，其中 X 是 API 的版本。

准备好 API 密钥和合适的测试平台后, 就开始编写代码了。

4.2.3 实现机制

想要显示一个地图, 只需要简单地在一个 `MapActivity` 中创建一个 `MapView` 实例。然后需要从 Google 那里获得一个 API 密钥, 并把这个密钥传给 XML 布局中的 `MapView` 的一个属性中。

程序清单 4-2 布局中典型的 `MapView`

```
<com.google.android.maps.MapView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="API_KEY_STRING_HERE"
/>
```

注意:

在 XML 中添加 `MapView` 时, 必须指定包的全名, 这是由于 `MapView` 并没有包含在 `android.view` 或 `android.widget` 中。

虽然同样可以用代码实例化一个 `MapView`, 但也需要在构造函数的参数中传入 API 密钥。

```
MapView map = new MapView(this, "API_KEY_STRING_HERE");
```

另外, 在应用程序的 `manifest` 中必须声明使用 `Maps` 库, 这样 Google Play 就不会将该应用程序显示在没有 `Maps` 库的设备上。

现在, 让我们看一个示例, 它会将用户上次最新的位置显示在地图上。参见程序清单 4-3。

程序清单 4-3 `AndroidManifest.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.mapper"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="3" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />

    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MyActivity"
            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>
```

```

        </activity>

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

    </application>
</manifest>

```

注意需要声明 INTERNET 和 ACCESS_FINE_LOCATION 权限。声明 ACCESS_FINE_LOCATION 权限是因为需要使用 LocationManager 获得缓存的位置信息。另一个关键点就是 manifest 中必须要有<uses-library>标签，该标签会引用 Google Maps API。Android 系统会在构建应用程序的时候通过这个标签链接相应的外部库，同时它还有另一个作用。Google Play 会根据这个库声明将应用程序显示给那些具有这个地图库的设备。

程序清单 4-4 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:text="Map Of Your Location"
    />
    <com.google.android.maps.MapView
        android:id="@+id/map"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="YOUR_API_KEY_HERE"
    />
</LinearLayout>

```

注意，必要时在需要输入 API 密钥的地方输入 API 密钥。同样，还要注意，尽管 MapView 必须在 MapActivity 中展开，但这个 Activity 布局中还可以包含其他东西，并不是只能有一个 MapView，参见程序清单 4-5。

程序清单 4-5 显示缓存位置的 MapActivity

```

public class MyActivity extends MapActivity {

    MapView map;
    MapController controller;

    @Override
    public void onCreate(Bundle savedInstanceState) {

```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);

map = (MapView)findViewById(R.id.map);
controller = map.getController();
LocationManager manager =
    (LocationManager) getSystemService(Context.LOCATION_SERVICE);
Location location =
    manager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
int lat, lng;
if(location != null) {
    //转换为微度
    lat = (int)(location.getLatitude() * 1000000);
    lng = (int)(location.getLongitude() * 1000000);
} else {
    //默认定位到 Google 总部
    lat = 37427222;
    lng = -122099167;
}
GeoPoint mapCenter = new GeoPoint(lat,lng);
controller.setCenter(mapCenter);
controller.setZoom(15);
}

//必须实现的抽象方法, 返回 false
@Override
protected boolean isRouteDisplayed() {
    return false;
}
}

```

这个 Activity 会获取用户最新的位置信息并将该位置设为地图的中心。关于地图的所有控制操作都是通过 MapController 实例来完成的, 而 MapController 实例则是通过调用 MapView.getController()得到的; 这个控制器可以对屏幕上的地图进行平移、缩放等调整。本例中, 我们使用了控制器的 setCenter()和 setZoom()方法对地图的显示做了调整。

MapController.setCenter()的参数为一个 GeoPoint, 它和我们从 Android 服务收到的 Location 略微有点不同。最大的不同就是 GeoPoint 是以微度(或者度数的 10⁻⁶)来表示精度和纬度的, 而不是使用十进制来表示。因此, 将 Location 应用到地图之前必须要对它进行转换。

MapController.setZoom()可以将地图设置为一个特定的缩放级别, 范围为 1~21。默认情况下, 地图的缩放级别为 1, SDK 文档将其定义为了一个全局的 View, 每放大一个等级, 地图的比例尺就会缩小一半, 如图 4-1 所示。

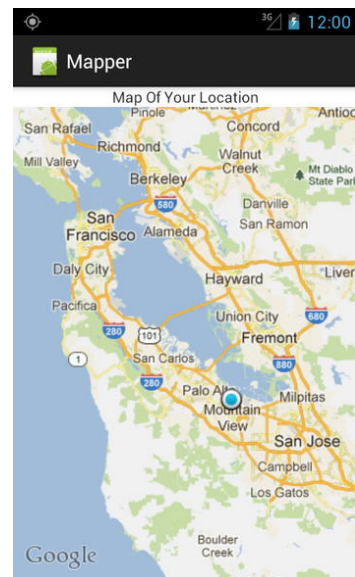


图 4-1 用户位置地图

你最先发现的问题可能是，地图上用户所在的位置并没有任何标识(例如大头针)。在4.3节中，我们会在地图上创建这些标记并对其进行自定义。

4.3 在地图上标记位置

4.3.1 问题

除了将指定的位置显示在地图的中心，应用程序还需要在该位置上加上标记，使其更加醒目。

4.3.2 解决方案

(API Level 1)

为地图创建一个自定义的 `ItemizedOverlay`，该 `ItemizedOverlay` 上包含了所有要标记的点。`ItemizedOverlay` 是一个用于在 `MapView` 上绘制各种对象的抽象基类。要绘制的对象是 `OverlayItem` 的实例，这个模型类定义了地图上标记点的名称、子标题和 `Drawable` 标记。

4.3.3 实现机制

让我们实现一个 `ItemizedOverlay`，它会接受一个 `GeoPoints` 的数组，然后将这些点用相同的 `Drawable` 标记绘制到地图上。参见程序清单 4-6。

程序清单 4-6 基本的 `ItemizedOverlay` 实现

```
public class LocationOverlay extends ItemizedOverlay<OverlayItem> {
    private List<GeoPoint> mItems;

    public LocationOverlay(Drawable marker) {
        super( boundCenterBottom(marker) );
    }

    public void setItems(ArrayList<GeoPoint> items) {
        mItems = items;
        populate();
    }

    @Override
    protected OverlayItem createItem(int i) {
        return new OverlayItem(mItems.get(i), null, null);
    }

    @Override
    public int size() {
        return mItems.size();
    }
}
```

```

@Override
protected boolean onTap(int i) {
    //处理单击事件
    return true;
}
}

```

在这个实现中，构造函数中传入了一个 `Drawable`，它用来标识地图上的每个标记点。在 `overlay` 上使用的 `Drawable` 必须要有合适的基准位置，而 `boundCenterBottom()` 正好可以帮助我们处理这个问题。具体地说，该方法就是将 `Drawable` 资源最底部一行像素的中点与地图上指定的位置对齐。

`ItemizedOverlay` 有两个必须要覆写的抽象方法：`createItem()` 返回声明类型的对象，`size()` 返回管理的项目数量。本例将若干个 `GeoPoints` 包装成 `OverlayItem`。在获得所有数据之后，要尽快调用 `Overlay` 的 `populate()` 方法，准备将其显示出来。在这个示例中，就是在 `setItems()` 结尾处。

下面将这个 `Overlay` 应用到地图上，用默认的应用程序图标标记 Google 总部附近的 3 个自定义位置，参见程序清单 4-7。

程序清单 4-7 使用自定义 Map Overlay 的 Activity

```

public class MyActivity extends MapActivity {

    MapView map;
    MapController controller;

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

        map = (MapView)findViewById(R.id.map);
        controller = map.getController();

        ArrayList<GeoPoint> locations = new ArrayList<GeoPoint>();
        //Google 总部的位置 37.427,-122.099
        locations.add(new GeoPoint(37427222,-122099167));
        //减去 0.01 度
        locations.add(new GeoPoint(37426222,-122089167));
        //增加 0.01 度
        locations.add(new GeoPoint(37428222,-122109167));

        LocationOverlay myOverlay =
            new LocationOverlay(getResources().getDrawable(R.drawable.icon));
        myOverlay.setItems(locations);
        map.getOverlays().add(myOverlay);
        controller.setCenter(locations.get(0));
        controller.setZoom(15);
    }
}

```

```

    }

    //需要覆写的方法，返回 false
    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
}

```

运行后，Activity 的效果如图 4-2 所示。

注意，其中图标阴影是由 MapView 和 ItemizedOverlay 生成的。

不过，该如何定义各个位置的图标，以显示不同的标记图片呢？实现方式就是设置各个位置的标记，为各个位置使用不同的自定义 Drawable 图片。在这种情况下，ItemizedOverlay 构造函数中的 Drawable 图片就只是在没有自定义图片存在情况下的默认标记。修改一下之前的实现，参见程序清单 4-8。

程序清单 4-8 使用自定义标记的 ItemizedOverlay

```

public class LocationOverlay extends ItemizedOverlay<OverlayItem> {
    private List<GeoPoint> mItems;
    private List<Drawable> mMarkers;

    public LocationOverlay(Drawable marker) {
        super( boundCenterBottom(marker) );
    }

    public void setItems(ArrayList<GeoPoint> items,
        ArrayList<Drawable> drawables) {
        mItems = items;
        mMarkers = drawables;
        populate();
    }

    @Override
    protected OverlayItem createItem(int i) {
        OverlayItem item = new OverlayItem(mItems.get(i), null, null);
        item.setMarker( boundCenterBottom(mMarkers.get(i)) );
        return item;
    }

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

```

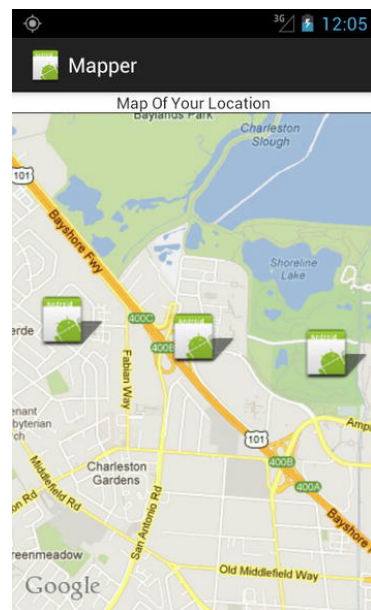


图 4-2 带有 ItemizedOverlay 的地图

```

@Override
protected boolean onTap(int i) {
    //处理单击事件
    return true;
}
}

```

这样修改后，`OverlayItem` 就会使用自定义的 `Drawable` 资源作为标记图片。如果所使用的 `Drawable` 是有状态的，当内容被选中或者被触摸时，就会显示按下和聚焦状态。下面的示例使用了这个新的实现，参见程序清单 4-9。

程序清单 4-9 使用自定义标记的示例 Activity

```

public class MyActivity extends MapActivity {

    MapView map;
    MapController controller;

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

        map = (MapView)findViewById(R.id.map);
        controller = map.getController();

        ArrayList<GeoPoint> locations = new ArrayList<GeoPoint>();
        ArrayList<Drawable> images = new ArrayList<Drawable>();

        //Google 总部的位置 37.427,-122.099
        locations.add(new GeoPoint(37427222,-122099167));
        images.add(getResources().getDrawable(R.drawable.logo));
        //减去 0.01 度
        locations.add(new GeoPoint(37426222,-122089167));
        images.add(getResources().getDrawable(R.drawable.icon));
        //增加 0.01 度
        locations.add(new GeoPoint(37428222,-122109167));
        images.add(getResources().getDrawable(R.drawable.icon));

        LocationOverlay myOverlay =
            new LocationOverlay(getResources().getDrawable(R.drawable.icon));
        myOverlay.setItems(locations, images);
        map.getOverlays().add(myOverlay);
        controller.setCenter(locations.get(0));
        controller.setZoom(15);
    }

    //需要覆写的抽象方法，返回 false
    @Override
    protected boolean isRouteDisplayed() {

```

```

        return false;
    }
}

```

现在我们的示例针对地图上显示的不同标记点使用了不同的图片。具体来说，我们在 Google 总部的的位置使用了 Google 的 logo，而其他两个地方使用的则是同样的标记，如图 4-3 所示。

1. 实现交互

也许你注意到 `LocationOverlay` 中定义的 `onTap()`，但我们并没有涉及它。基于 `ItemizedOverlay` 实现的另一个好处就是它能处理碰撞测试以及很方便地根据各个条目的索引处理触摸事件。在这个方法中，可以弹出一个 `toast`、显示一个对话框、启动一个新的 `Activity`、或者执行其他动作(例如，用户单击地图标记来获取详细信息动作)。

2. 我在哪里？

Android 的 Maps API 还提供了一个用于绘制用户当前位置的特殊 `Overlay`—`MyLocationOverlay`。这个 `overlay` 用起来很简单，在 `Activity` 可见时启用它即可。否则，不必要的资源消耗会降低系统的性能和电池的寿命。参见程序清单 4-10。

程序清单 4-10 添加 `MyLocationOverlay`

```

public class MyActivity extends MapActivity {

    MapView map;
    MyLocationOverlay myOverlay;

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

        map = (MapView)findViewById(R.id.map);
        myOverlay = new MyLocationOverlay(this, map);
        map.getOverlays().add(myOverlay);
    }

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

```

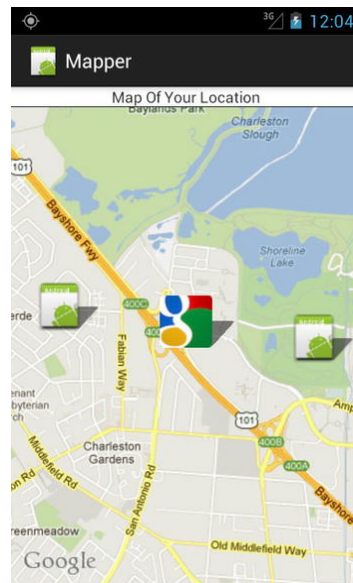


图 4-3 使用自定义标记图片的地图

```

@Override
public void onPause() {
    super.onResume();
    myOverlay.disableMyLocation();
}

//需要覆写的抽象方法, 返回 false
@Override
protected boolean isRouteDisplayed() {
    return false;
}
}

```

这样会在用户的当前位置上显示一个标准的点或者箭头标记(取决于设备是否使用了罗盘), 只要启用了这个 overlay, 它就会一直跟踪用户的移动位置。

使用 MyLocationOverlay 的关键是要在不用时(Activity 不可见时)禁用它而在需要时再重新启用它。这和 LocationManager 的用法是一样的, 都是为了避免不必要的电量消耗。

4.4 拍摄照片和视频

4.4.1 问题

应用程序需要使用设备的摄像头采集媒体信息, 可以是静态图片或者小的视频片段。

4.4.2 解决方案

(API Level 3)

向 Android 发送一个 Intent, 将控制权交给 Camera 应用程序, 并将用户拍摄的照片返回。Android 已经包含了可以直接访问摄像头硬件、预览和拍照或录制视频的 API。但是, 如果只是想简单地让用户在熟悉的界面中拍摄照片和视频, 最好的解决方案则是让 Camera 应用程序来处理。

4.4.3 实现机制

让我们看一下如何使用 Camera 应用程序拍摄静止图片和录制视频片段。

1. 拍摄照片

让我们看一个示例 Activity, 在按下“Take a Picture”时该 Activity 会激活一个 Camera 应用程序。在 Camera 应用程序中操作完成后, 会得到一个 Bitmap 结果。

程序清单 4-11 res/layout/main.xml

```

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

```

```

        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <Button
            android:id="@+id/capture"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Take a Picture"
        />
        <ImageView
            android:id="@+id/image"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scaleType="centerInside"
        />
    </LinearLayout>

```

程序清单 4-12 拍摄照片的 Activity

```

public class MyActivity extends Activity {

    private static final int REQUEST_IMAGE = 100;

    Button captureButton;
    ImageView imageView;

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

        captureButton = (Button)findViewById(R.id.capture);
        captureButton.setOnClickListener(listener);

        imageView = (ImageView)findViewById(R.id.image);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        if(requestCode == REQUEST_IMAGE && resultCode == Activity.RESULT_OK) {
            //处理并显示图片
            Bitmap userImage = (Bitmap)data.getExtras().get("data");
            imageView.setImageBitmap(userImage);
        }
    }

    private View.OnClickListener listener = new View.OnClickListener() {
        @Override

```

```

        public void onClick(View v) {
            Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            startActivityForResult(intent, REQUEST_IMAGE);
        }
    };
}

```

这个方法会拍摄照片并且在“data”附加信息中返回按比例缩小的 Bitmap。如果需要拍摄照片并且保存全尺寸的照片，则需要开始拍照前，在 Intent 的 MediaStore.EXTRA_OUTPUT 中放入照片保存位置的 URI。参见程序清单 4-13。

程序清单 4-13 将全尺寸照片保存到文件中

```

public class MyActivity extends Activity {

    private static final int REQUEST_IMAGE = 100;

    Button captureButton;
    ImageView imageView;
    File destination;

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

        captureButton = (Button)findViewById(R.id.capture);
        captureButton.setOnClickListener(listener);

        imageView = (ImageView)findViewById(R.id.image);

        destination =
            new File(Environment.getExternalStorageDirectory(), "image.jpg");
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        if(requestCode == REQUEST_IMAGE && resultCode == Activity.RESULT_OK) {
            try {
                FileInputStream in = new FileInputStream(destination);
                BitmapFactory.Options options = new BitmapFactory.Options();
                options.inSampleSize = 10; //Downsample by 10x

                Bitmap userImage =
                    BitmapFactory.decodeStream(in, null, options);
                imageView.setImageBitmap(userImage);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }
}

private View.OnClickListener listener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        //添加附件信息来保存全尺寸的图片
        intent.putExtra(MediaStore.EXTRA_OUTPUT,
            Uri.fromFile(destination));
        startActivityForResult(intent, REQUEST_IMAGE);
    }
};
}

```

这个方法会告诉 Camera 应用程序将图片保存到某个位置(本例为设备 SD 卡下的“image.jpg”),这样图片就不会缩放了。当 Camera 应用程序中的操作完成后,就可以直接到刚才指定的文件位置寻找图片。

为了避免将全尺寸的 Bitmap 加载到内存中,需要图片在屏幕上显示之前,通过 BitmapFactory.Options 对图片进行缩放。同样,你会发现这个示例会将保存图片的文件位置指定为设备的外部存储器,这就需要在 API Level 4 及以上版本上指定 android.permission.WRITE_EXTERNAL_STORAGE 权限。如果在其他地方存储的话,则不需要这个权限。

2. 拍摄视频

通过这种方法可以直接拍摄视频片段,只是结果稍有差异。真正的视频片段数据是不会通过 Intent 的附加信息返回的,通常都是保存到一个指定的文件中。以下两个参数可能会作为附加信息添加到 Intent 中:

(1) MediaStore.EXTRA_VIDEO_QUALITY

- a. 整型值,用来描述拍摄的视频的质量等级。
- b. 0 表示低质量,1 表示高质量

(2) MediaStore.EXTRA_OUTPUT

- c. 保存视频内容的 Uri
- d. 如果没有指定保存位置,视频则会保存到设备的标准位置中。

视频录制结束后,会在结果 Intent 的 data 中返回视频数据真正存储位置的 Uri。让我们看一个类似的示例,该示例允许用户录制和保存他们的视频并将视频的存储位置显示在屏幕上。参见程序清单 4-14 和 4-15。

程序清单 4-14 res/layout/main.xml

```

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

```

```

        android:id="@+id/capture"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Take a Video"
    />
    <TextView
        android:id="@+id/file"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>

```

程序清单 4-15 拍摄视频片段的 Activity

```

public class MyActivity extends Activity {

    private static final int REQUEST_VIDEO = 100;

    Button captureButton;
    TextView text;
    File destination;

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

        captureButton = (Button)findViewById(R.id.capture);
        captureButton.setOnClickListener(listener);

        text = (TextView)findViewById(R.id.file);

        destination =
            new File(Environment.getExternalStorageDirectory(), "myVideo");
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        if(requestCode == REQUEST_VIDEO && resultCode == Activity.RESULT_OK) {
            String location = data.getData().toString();
            text.setText(location);
        }
    }

    private View.OnClickListener listener = new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
            //添加(可选)附加信息从而将视频保存到指定文件
            intent.putExtra(MediaStore.EXTRA_OUTPUT,

```

```

        Uri.fromFile(destination));
//可选的附加信息用来设置视频质量
intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 0);
startActivityForResult(intent, REQUEST_VIDEO);
    }
};
}

```

本例和之前保存图片的示例一样，都是将录制的视频保存到设备的 SD 卡上(对于 API Level 4 及以上版本需要指定 `android.permission.WRITE_EXTERNAL_STORAGE` 权限)。首先，我们向系统发送了一个动作为 `MediaStore.ACTION_VIDEO_CAPTURE` 的 `Intent`。Android 则会启动默认的 Camera 应用程序来处理视频的录制并在录制完成时返回 OK。我们会在 `onActivityResult()` 回调方法中调用 `Intent.getData()` 来得到数据的存储位置，并把这个位置显示给用户。

这个示例显式地要求录制的视频使用低质量，但这个参数是可选的。如果没有在请求的 `Intent` 中指定 `MediaStore.EXTRA_VIDEO_QUALITY`，设备通常会选择高质量进行录制。

一旦指定了 `MediaStore.EXTRA_OUTPUT`，那么返回的 `Uri` 应该和指定的位置一致，除非应用程序在写入该位置时发生了错误。如果没有这个参数，返回的值就会是 `content://Uri`，可以从系统的 `MediaStore Content Provider` 中获取视频。

稍后在范例 4-8 中，我们将介绍如何在应用程序中播放媒体文件。

4.5 自定义摄像头覆盖层

4.5.1 问题

很多应用程序都希望能够直接访问摄像头，例如要使用自定义的摄像头控制 UI 界面或者根据位置和方向传感器的信息显示其他数据(增强真实感)。

4.5.2 解决方案

(API Level 5)

在自定义的 `Activity` 中直接关联一个摄像头。Android 提供了 API 来直接访问设备的摄像头(用于获得预览画面以及拍摄照片)。如果应用程序除了拍摄并显示照片外还要使用其他功能，也可以直接访问这些 API。

注意：

因为我们需要直接访问摄像头，所以必须在 `manifest` 中指定 `android.permission.CAMERA` 权限。

4.5.3 实现机制

我们首先会创建一个 `SurfaceView`，将使用这个专门的 `view` 实时地绘制摄像头的预览

数据流。这就可以将一个实时预览图内嵌到 Activity 的布局中。接下来,只要简单地根据应用程序的需求添加其他的 view 即可。让我们看一下代码(参见程序清单 4-16 和 4-17)。

注意:

这里使用的 Camera 类是 android.hardware.Camera, 不要和 android.graphics.Camera 弄混了。确保应用程序已经引入了正确的包。

程序清单 4-16 res/layout/main.xml

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

程序清单 4-17 显示活动的摄像头预览的 Activity

```
import android.hardware.Camera;

public class PreviewActivity extends Activity implements SurfaceHolder.Callback {

    Camera mCamera;
    SurfaceView mPreview;

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

        mPreview = (SurfaceView)findViewById(R.id.preview);
        mPreview.getHolder().addCallback(this);
        mPreview.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

        mCamera = Camera.open();
    }

    @Override
    public void onPause() {
        super.onPause();
        mCamera.stopPreview();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

```

        mCamera.release();
    }

    //Surface 回调方法
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format,
        int width, int height) {
        Camera.Parameters params = mCamera.getParameters();
        //得到设备支持的尺寸, 并选择第一个(最大)
        List<Camera.Size> sizes = params.getSupportedPreviewSizes();
        Camera.Size selected = sizes.get(0);
        params.setPreviewSize(selected.width,selected.height);
        mCamera.setParameters(params);

        mCamera.startPreview();
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        try {
            mCamera.setPreviewDisplay(mPreview.getHolder());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) { }
}

```

注意:

如果在模拟器上进行测试的话, 就不能预览摄像头。新版本的 SDK 已经可以在一些主机上使用内置的摄像头, 但并不是全部都可以使用的。对于摄像头不可用的情况, 模拟器会显示一个假的预览画面, 不同的模拟器版本效果稍微会有些不同。要验证这段代码是否正确, 打开模拟器上的 Camera 应用程序, 看看其中预览的内容是什么。所显示的效果应该和真实设备上的相同。通常最好是在运行有相应硬件的真实设备上进行代码的测试。

在本例中, 我们创建了一个 SurfaceView 来填满整个窗口并且指定在 SurfaceHolder 回调时通知我们的 Activity。在摄像头初始化工作彻底完成之前就不会在 surface 上显示预览信息, 所以我们在 surfaceCreated()回调时才将 view 的 SurfaceHolder 关联到 Camera 实例上。类似的, 当 surfaceChanged()回调时也就是 surface 确定后, 我们才能确定预览画面的尺寸并开始绘制。

这个应用程序的摄像头硬件资源是通过调用 Camera.open()来声明和打开的。Android 2.3 中提供了该方法的另一个版本, 在有多个摄像头的情况下, 该方法使用了一个整型(0 到 getNumberOfCameras()-1)参数来指定使用哪个摄像头。在多摄像头的设备中, 无参的版本通常会打开后置的摄像头。

重点:

很多新设备如 Google 的 Nexus7 平板电脑并没有后置摄像头, 所以旧的 `Camera.open()` 方法会返回 `null`。如果有一个支持旧版本 Android 的 Camera 应用程序, 需要修改代码使用新的 API 来判断设备是否提供了摄像头。

调用 `Parameters.getSupportedPreviewSizes()` 会返回设备支持的各种尺寸列表, 通常是从大到小进行排序的。本例中, 我们使用了第一个(最大的)预览分辨率并使用它设置预览大小。

注意:

在 2.0(API Level 5)之前的版本中, 是可以直接在 `Parameters.setPreviewSize()` 方法中传入高度和宽度参数的。但在 2.0 及以后版本中, 摄像头的预览尺寸只可以设置为设备支持的分辨率。否则会导致异常。

`Camera.startPreview()` 会开始在 `surface` 上实时绘制摄像头的的数据。注意, 预览画面同时是横屏显示的。在 Android 2.2 (API Level 8)之前, 并没有调整预览画面显示方向的官方方法。因此, 建议使用摄像头预览画面的 Activity 要在 `manifest` 中指定固定的方向 `android:screenOrientation="landscape"`, 这样才能兼容使用旧版本的设备。

Camera 服务在某一个时刻只能有一个应用程序使用。因此, 在摄像头不用时及时调用 `Camera.release()` 方法是很重要的。本例中, 在 Activity 销毁时, 我们不需要使用摄像头了, 因此在 `onDestroy()` 调用了该方法。

1. 改变拍摄方向**(API Level 8)**

从 Android 2.2 开始, 可以改变摄像头的预览方向。应用程序可以调用 `Camera.setDisplayOrientation()` 来旋转得到的数据从而匹配 Activity 的方向。有效的值为 0、90、180、270, 0 表示默认的横屏显示。这个方法主要会影响视频采集前预览数据该如何在 `surface` 上面进行绘制。

想要旋转摄像头输出的数据, 需要使用 `Camera.Parameters` 的 `setRotation()` 方法。这个方法的实现效果取决于设备, 它也许会旋转实际的输出图像、也许会通过选择参数修改 EXIF 数据, 或者二者皆有。

2. 照片覆盖层

现在我们可以前面示例中的照片预览界面上添加各种控件和 `view`。以下代码会添加一个 Cancel 按钮和 Snap Photo 按钮。参见程序清单 4-18 和 4-19。

程序清单 4-18 `res/layout/main.xml`

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

```

        android:layout_height="fill_parent">
        <SurfaceView
            android:id="@+id/preview"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
        />
        <RelativeLayout
            android:layout_width="fill_parent"
            android:layout_height="100dip"
            android:layout_alignParentBottom="true"
            android:gravity="center_vertical"
            android:background="#A000">
            <Button
                android:layout_width="100dip"
                android:layout_height="wrap_content"
                android:text="Cancel"
                android:onClick="onCancelClick"
            />
            <Button
                android:layout_width="100dip"
                android:layout_height="wrap_content"
                android:layout_alignParentRight="true"
                android:text="Snap Photo"
                android:onClick="onSnapClick"
            />
        </RelativeLayout>
    </RelativeLayout>

```

程序清单 4-19 添加了照片控制的 Activity

```

public class PreviewActivity extends Activity implements
    SurfaceHolder.Callback, Camera.ShutterCallback, Camera.PictureCallback {

    Camera mCamera;
    SurfaceView mPreview;

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

        mPreview = (SurfaceView)findViewById(R.id.preview);
        mPreview.getHolder().addCallback(this);
        mPreview.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

        mCamera = Camera.open();
    }

    @Override
    public void onPause() {
        super.onPause();
    }

```

```

        mCamera.stopPreview();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mCamera.release();
        Log.d("CAMERA", "Destroy");
    }

    public void onCancelClick(View v) {
        finish();
    }

    public void onSnapClick(View v) {
        //拍摄照片
        mCamera.takePicture(this, null, null, this);
    }

    //Camera 回调方法
    @Override
    public void onShutter() {
        Toast.makeText(this, "Click!", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        //可以将照片保存到某个位置, 这里保存到内部存储器中
        try {
            FileOutputStream out =
                openFileOutput("picture.jpg", Activity.MODE_PRIVATE);
            out.write(data);
            out.flush();
            out.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //必须重新启动预览
        camera.startPreview();
    }

    //Surface 回调方法
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format,
        int width, int height) {
        Camera.Parameters params = mCamera.getParameters();
    }

```

```

        List<Camera.Size> sizes = params.getSupportedPreviewSizes();
        Camera.Size selected = sizes.get(0);
        params.setPreviewSize(selected.width,selected.height);
        mCamera.setParameters(params);

        mCamera.setDisplayOrientation(90);
        mCamera.startPreview();
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        try {
            mCamera.setPreviewDisplay(mPreview.getHolder());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) { }
}

```

这里我们添加了一个简单的、部分区域透明的 overlay，它包含了两个摄像头的操作。Cancel 的作用没什么可说的，就是结束 Activity。但是，Snap Photo 在手动拍摄并返回照片的过程中则涉及了很多 Camera API。用户会初始化 Camera.takePicture()方法，然后则是该方法的一系列回调方法。

注意，本例中的 Activity 还额外实现了两个接口：Camera.ShutterCallback 和 Camera.PictureCallback。前者是在拍摄照片后(快门关闭后)立即被调用，而后者则在有不同形式的图片可用时在多个实例中被调用。

takePicture()的参数是一个 ShutterCallback 和不超过三个的 PictureCallback 实例。PictureCallback 会在以下时刻被调用(根据参数的顺序)：

- (1) 照片以 RAW 图片数据形式被拍摄
 - a. 在设备内存不足时，可能返回 null
- (2) 当图片被处理为缩放后的数据(也就是 POSTVIEW 图片)
 - b. 在设备内存不足时，可能返回 null
- (3) 当图片被压缩为 JPEG 图像数据

这个示例只用到了压缩为 JPEG 格式后的照片。所以，只用到了最后一个回调方法，也就是照片预览必须重新启动时。如果在照片拍摄后没有调用 startPreview()，屏幕上阅览画面就会一直停留在刚刚拍摄的照片上。

提示：

如果希望你的应用程序只能由拥有相应硬件功能的设备下载，可以在应用程序的 manifest 中添加市场过滤器，如下：<uses-feature android:name="android.hardware.camera" />。

4.6 录制音频

4.6.1 问题

应用程序需要使用设备的麦克风录音

4.6.2 解决方案

(API Level 1)

通过 `MediaRecorder` 采集音频并保存到文件中。

4.6.3 实现机制

`MediaRecorder` 用起来非常简单。你只需要提供一些文件编码格式的基本信息和保存音频数据的位置。程序清单 4-20 和 4-21 演示了如何录制音频文件并保存到设备的 SD 卡中,这期间会监听用户开始录音和结束录音的动作。

重点:

想要通过 `MediaRecorder` 录制音频输入,必须在应用程序的 `manifest` 中声明 `android.permission.RECORD_AUDIO` 权限。

程序清单 4-20 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/startButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Start Recording"
    />
    <Button
        android:id="@+id/stopButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Stop Recording"
        android:enabled="false"
    />
</LinearLayout>
```

程序清单 4-21 录制音频的 Activity

```
public class RecordActivity extends Activity {
```

```

private MediaRecorder recorder;
private Button start, stop;
File path;

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

    start = (Button)findViewById(R.id.startButton);
    start.setOnClickListener(startListener);
    stop = (Button)findViewById(R.id.stopButton);
    stop.setOnClickListener(stopListener);

    recorder = new MediaRecorder();
    path = new File(Environment.getExternalStorageDirectory(),
        "myRecording.3gp");

    resetRecorder();
}

@Override
public void onDestroy() {
    super.onDestroy();
    recorder.release();
}

private void resetRecorder() {
    recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
    recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
    recorder.setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
    recorder.setOutputFile(path.getAbsolutePath());
    try {
        recorder.prepare();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private View.OnClickListener startListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            recorder.start();

            start.setEnabled(false);
            stop.setEnabled(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

        }
    }
};

private View.OnClickListener stopListener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        recorder.stop();
        resetRecorder();

        start.setEnabled(true);
        stop.setEnabled(false);
    }
};
}

```

这个示例的 UI 非常简单。就是两个按钮，它们的使用是基于录制状态的。当用户按下 **Start** 按钮，就可以开始录音并且 **Stop** 按钮也会可用。当按下 **Stop** 时，将重置录音机以便进行下一次录音同时 **Start** 按钮也会重新可用。

MediaRecorder 的设置也很简单。我们在 SD 卡上创建了一个名为 “myRecording.3gp” 的文件，并将文件路径传给 **setOutputFile()** 方法。而其他的设置方法则告诉录音机将使用设备的麦克作为音频输入源(**AudioSource.MIC**)，并使用默认的编码器创建一个 3GP 格式的文件。

现在，可以使用设备的文件浏览器或者媒体播放器来播放这个音频文件。在后面的范例 4-8 中，还将展示如何使用应用程序播放音频。

4.7 自定义视频采集

4.7.1 问题

应用程序需要视频采集，同时需要能够比范例 4-4 更多地控制视频录制的过程。

4.7.2 解决方案

(API Level 8)

通过 **MediaRecorder** 和 **Camera** 彼此合作来创建自己的视频采集 **Activity**。和之前范例中使用 **MediaRecorder** 进行单纯地录制音频相比，这个示例稍微有点复杂。我们希望在没有录制音频时用户也可以看到摄像头的预览，想实现它必须要在录制与预览之间管理好摄像头的访问控制。

4.7.3 实现机制

程序清单 4-22 到 4-24 演示了录制视频并将结果保存到设备外部存储器的示例。

程序清单 4-22 部分 AndroidManifest.xml 代码

```

<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

...

<activity
    android:name=".VideoCaptureActivity"
    android:screenOrientation="portrait" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

```

在这个 manifest 中，核心的目的就是我们将我们 Activity 的方向设置为固定的竖屏。另外还指定了一些访问摄像头和录制音频音轨时需要的权限。

程序清单 4-23 res/layout/main.xml

```

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

    <Button
        android:id="@+id/button_record"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:onClick="onRecordClick" />

    <SurfaceView
        android:id="@+id/surface_video"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@id/button_record" />
</RelativeLayout>

```

程序清单 4-24 采集视频的 Activity

```

public class VideoCaptureActivity extends Activity implements SurfaceHolder.Callback {

    private Camera mCamera;
    private MediaRecorder mRecorder;

    private SurfaceView mPreview;
    private Button mRecordButton;

    private boolean mRecording = false;

```

```

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

    mRecordButton = (Button) findViewById(R.id.button_record);
    mRecordButton.setText("Start Recording");

    mPreview = (SurfaceView) findViewById(R.id.surface_video);
    mPreview.getHolder().addCallback(this);
    mPreview.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

    mCamera = Camera.open();
    //旋转预览画面为竖屏
    mCamera.setDisplayOrientation(90);

    mRecorder = new MediaRecorder();
}

@Override
protected void onDestroy() {
    mCamera.release();
    mCamera = null;
    super.onDestroy();
}

public void onRecordClick(View v) {
    updateRecordingState();
}

/*
 *初始化摄像头和摄像机
 *这些方法的顺序很重要，这是因为 MediaRecorder 的状态严格依赖于所调用的方法
 */
private void initializeRecorder() throws IllegalStateException,
    IOException {
    //解锁摄像头，允许 MediaRecorder 使用它
    mCamera.unlock();
    mRecorder.setCamera(mCamera);
    //设置 MediaRecorder 的数据源
    mRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
    mRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
    //修改输出设置
    File recordOutput = new File(Environment.getExternalStorageDirectory(),
        "recorded_video.mp4");
    if(recordOutput.exists()) {
        recordOutput.delete();
    }
    CamcorderProfile cpHigh =

```

```

        CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH);
mRecorder.setProfile(cpHigh);
mRecorder.setOutputFile(recordOutput.getAbsolutePath());
//为摄像机关联一个 surface, 从而实现在录制的同时可以预览
mRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());

//设置录制的一些限制值, 这个是可选的
mRecorder.setMaxDuration(50000); // 50 秒
mRecorder.setMaxFileSize(5000000); // 大约 5M

mRecorder.prepare();
}

private void updateRecordingState() {
    if(mRecording) {
        mRecording = false;
        //重置摄像机的状态以便进行下次录制
        mRecorder.stop();
        mRecorder.reset();
        //返回摄像机继续预览
        mCamera.lock();
        mRecordButton.setText("Start Recording");
    } else {
        try {
            //重置摄像机以便进行下次会话
            initializeRecorder();
            //开始录制
            mRecording = true;
            mRecorder.start();
            mRecordButton.setText("Stop Recording");
        } catch(Exception e) {
            //初始化摄像机时发生错误
            e.printStackTrace();
        }
    }
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
    //得到一个 surface 后, 立刻启动摄像头预览
    try {
        mCamera.setPreviewDisplay(holder);
        mCamera.startPreview();
    } catch(IOException e) {
        e.printStackTrace();
    }
}

@Override
public void surfaceChanged(SurfaceHolder holder, int format, int width,

```

```

        int height) { }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) { }
}

```

当 Activity 首次创建时，它得到了设备摄像头的一个实例，并且将摄像头的显示方向设置为我们在 manifest 中所指定的竖屏。调用这个方法只会影响到预览画面的显示，而不会影响录制输出，在后面会更多地讨论它。当 Activity 可见时，会收到 surfaceCreated() 的回调，这时 Camera 会开始发送预览数据。

当用户打算按下按钮开始录制时，Camera 会被解锁并交给 MediaRecorder 使用。然后 MediaRecorder 会设置进行视频采集所需的参数，如数据源和数据格式，以及时间和文件大小限制，这是为了防止用户的存储器负载过大。

注意：

使用 MediaRecorder 但不直接管理 Camera 也是可以录制视频的，但你将不能修改屏幕显示的方向并且应用程序在录制视频时只能显示预览画面。

录制完成后，会在用户的外部存储器中自动保存一个文件并且重置摄像机实例以便用户进行下一次的录制。我们也将重新获得 Camera 的控制权从而可以继续显示预览画面。

输出格式方向

(API Level 9)

本例中，使用 Camera.setDisplayOrientation() 使得预览画面的方向可以匹配我们的竖屏 Activity。但是，很多时候如果在电脑中播放这个视频，这个界面依然会是横屏的。想要解决这个问题，可以使用 MediaRecorder 的 setOrientationHint() 方法。这个方法的参数是一个与画面显示方向相匹配的角度值，通过这个值，视频文件(例如 3GP 和 MP4)的元数据会通知其他的视频播放应用程序，在播放该视频文件时需要把方向旋转一下。

这可能是不必要的，因为一些视频播放器在决定播放方向时是基于哪个方向上视频尺寸比较小的。正是出于这个原因，而且也为了能够和 API Level 8 相兼容，我们并没有在示例中加入这个方法。

4.8 添加语音识别

4.8.1 问题

应用程序需要使用语音识别技术来解析语音输入。

4.8.2 解决方案

(API Level 3)

通过 android.speech 包中的类可以使用 Android 设备中内置的语音识别技术。每个配备

了语音搜索(从 Android 开始)的 Android 设备都可以让应用程序使用设备内置的语音识别来处理语音输入。

想要使用这个功能,应用程序只需要向系统发送一个 `RecognizerIntent`,然后语音识别服务就会记录语音输入并且处理它;最后语音识别器就会将它听到的语音以字符串列表的形式返还给你。

4.8.3 实现机制

让我们用一个示例来了解这项技术。参见程序清单 4-25。

程序清单 4-25 启动和处理语音识别的 Activity

```
public class RecognizeActivity extends Activity {

    private static final int REQUEST_RECOGNIZE = 100;

    TextView tv;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        tv = new TextView(this);
        setContentView(tv);

        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
            RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "Tell Me Your
Name");
        try {
            startActivityForResult(intent, REQUEST_RECOGNIZE);
        } catch (ActivityNotFoundException e) {
            //如果不存在任何的识别器,就从 Google Play 下载一个
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Not Available");
            builder.setMessage("There is no recognition application
installed."
                + " Would you like to download one?");
            builder.setPositiveButton("Yes",
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        //例如下载 Google Voice Search
                        Intent marketIntent = new Intent(Intent.ACTION_VIEW);
                        marketIntent.setData(
                            Uri.parse(
                                "market://details?id=com.google.android.voicesearch"
                            ) );
                    }
                }
            );
        }
    }
}
```

```

        }
    });
    builder.setNegativeButton("No", null);
    builder.create().show();
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if(requestCode == REQUEST_RECOGNIZE &&
        resultCode == Activity.RESULT_OK) {
        ArrayList<String> matches =
            data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
        StringBuilder sb = new StringBuilder();
        for(String piece : matches) {
            sb.append(piece);
            sb.append('\n');
        }
        tv.setText(sb.toString());
    } else {
        Toast.makeText(this, "Operation Canceled",
            Toast.LENGTH_SHORT).show();
    }
}
}

```

注意:

如果是在模拟器上测试你的应用程序，注意 Google Play 和语音识别器都是未安装的。最好是在设备上进行测试。

本例在应用程序加载时会自动启动一个语音识别的 Activity，询问用户“Tell Me Your Name”。在接收用户的语音输入并处理了结果之后，Activity 会返回一个用户可能说的内容的列表。这个列表是按照概率排序的，所以多数情况下，只需要调用 `matches.get(0)` 返回最佳答案就可以了。但这个 Activity 会将所有的返回值显示在屏幕上——纯属娱乐。

在启动 `SpeechRecognizer` 时，可以通过传入 Intent 的附加信息自定义语音识别器的行为。本例使用了两个最常见的附加信息。

- `EXTRA_LANGUAGE_MODEL`
 - 用来优化语音识别器的处理结果。
 - 常规的语音到文本的查询应该使用 `LANGUAGE_MODEL_FREE_FORM` 选项。
 - 如果是较短的识别，`LANGUAGE_MODEL_WEB_SEARCH` 效果可能会更好。
- `EXTRA_PROMPT`
 - 提示用户开始语音识别的字符串。

除了这些选项，还可能用到其他的一些参数：

- `EXTRA_MAX_RESULTS`

- 这个整数设置了返回结果的最大数量。
- EXTRA_LANGUAGE
 - 要求用与系统当前的默认语言不同的语言返回结果。
 - 有效的 IETF 标签字符串。如 “en-US” 或者 “es”。

4.9 播放音频/视频

4.9.1 问题

应用程序需要在设备上播放本地或者远程的音频或者视频内容。

4.9.2 解决方案

(API Level 1)

可以使用 `MediaPlayer` 播放本地媒体或者流媒体。不管媒体文件是音频或者视频、本地或者远程，`MediaPlayer` 都可以高效地连接、准备、并且播放它们。在这个范例中，还会使用 `MediaController` 和 `VideoView` 在 `Activity` 的布局中实现简单的用户交互和视频播放功能。

4.9.3 实现机制

注意：

在播放某一特定格式的媒体片段或者媒体流之前，请先阅读开发者说明文档的“Android 支持的媒体格式”一节，看看是否支持该格式。

1. 音频播放

让我们看一个简单的示例，使用 `MediaPlayer` 来播放一段音频。参见程序清单 4-26。

程序清单 4-26 播放本地音频的 Activity

```
public class PlayActivity extends Activity implements
    MediaPlayer.OnCompletionListener {

    Button mPlay;
    MediaPlayer mPlayer;

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

        mPlay = new Button(this);
        mPlay.setText("Play Sound");
        mPlay.setOnClickListener(playListener);
```

```

        setContentView(mPlay);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if(mPlayer != null) {
            mPlayer.release();
        }
    }

    private View.OnClickListener playListener = new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            if(mPlayer == null) {
                try {
                    mPlayer = MediaPlayer.create(PlayActivity.this,
                                                    R.raw.sound);
                    mPlayer.start();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                mPlayer.stop();
                mPlayer.release();
                mPlayer = null;
            }
        }
    };

    //播放完成的监听方法
    @Override
    public void onCompletion(MediaPlayer mp) {
        mPlayer.release();
        mPlayer = null;
    }
}

```

这个示例通过一个按钮开始和停止播放一个本地音频文件，该文件存储在项目的 `res/raw` 目录下。`MediaPlayer.create()` 是一个简便的方法，它有很多种形式，旨在一步完成播放器对象的构造和准备工作。本例使用的形式为获取一个本地资源 ID 的引用，而通过 `MediaPlayer.create(Context context, Uri uri)` 这样的 `create()` 方法，还可以访问和播放远程资源。

创建完成后，示例会立刻播放音频。在播放时，用户可以再次按下按钮来停止播放。`Activity` 还实现了 `MediaPlayer.OnCompletionListener` 接口，因此会在播放操作正常完成之后收到一个回调。

在停止播放和播放完成这两种情况下，一旦播放停止，`MediaPlayer` 实例就会被释放。

这个方法可以确保只在需要时才会持有资源而且可以多次播放。为了确保不会无故持有资源，同样会在 Activity 销毁时释放依然存在的资源。

如果应用程序需要播放很多不同的声音，或许可以考虑在播放结束后调用 `reset()` 而不是 `release()`。但要记住，在不需要播放器时(或者 Activity 销毁时)还是需要调用 `release()`。

2. 音频播放器

除了简单地播放，如果应用程序还想为用户提供播放、停止、和拖曳播放进度的用户体验，该如何实现呢？MediaPlayer 中有很多在自定义 UI 中实现这些功能的方法，但 Android 同时还提供了 MediaController 来处理这样的需求。参见程序清单 4-27 和 4-28。

程序清单 4-27 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/root"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Now Playing..."
    />
    <ImageView
        android:id="@+id/coverImage"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="centerInside"
    />
</LinearLayout>
```

程序清单 4-28 使用 MediaController 播放音频的 Activity

```
public class PlayerActivity extends Activity implements
    MediaController.MediaPlayerControl,
    MediaPlayer.OnBufferingUpdateListener {
    MediaController mController;
    MediaPlayer mPlayer;
    ImageView coverImage;

    int bufferPercent = 0;

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

```

        coverImage = (ImageView)findViewById(R.id.coverImage);

        mController = new MediaController(this);
        mController.setAnchorView(findViewById(R.id.root));
    }

    @Override
    public void onResume() {
        super.onResume();
        mPlayer = new MediaPlayer();
        //设置音频数据源
        try {
            mPlayer.setDataSource(this, Uri.parse("URI_TO_REMOTE_AUDIO"));
            mPlayer.prepare();
        } catch (Exception e) {
            e.printStackTrace();
        }
        //设置专辑封面的图片
        coverImage.setImageResource(R.drawable.icon);

        mController.setMediaPlayer(this);
        mController.setEnabled(true);
    }

    @Override
    public void onPause() {
        super.onPause();
        mPlayer.release();
        mPlayer = null;
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        mController.show();
        return super.onTouchEvent(event);
    }

    // MediaPlayerControl 方法
    @Override
    public int getBufferPercentage() {
        return bufferPercent;
    }

    @Override
    public int getCurrentPosition() {
        return mPlayer.getCurrentPosition();
    }

    @Override
    public int getDuration() {

```

```

        return mPlayer.getDuration();
    }

    @Override
    public boolean isPlaying() {
        return mPlayer.isPlaying();
    }

    @Override
    public void pause() {
        mPlayer.pause();
    }

    @Override
    public void seekTo(int pos) {
        mPlayer.seekTo(pos);
    }

    @Override
    public void start() {
        mPlayer.start();
    }

    // BufferUpdateListener 方法
    @Override
    public void onBufferingUpdate(MediaPlayer mp, int percent) {
        bufferPercent = percent;
    }

    //Android2.0+ 目标回调
    public boolean canPause() {
        return true;
    }

    public boolean canSeekBackward() {
        return true;
    }

    public boolean canSeekForward() {
        return true;
    }
}

```

这个示例创建了一个简单的音频播放器，该播放器可以在播放音乐时显示艺术家的照片或者专辑封面(这里只显示应用程序图标)。本例依然使用的是 **MediaPlayer** 实例，但并不是通过 **create()**方法创建的。而是在创建实例后使用 **setDataSource()**设置它的内容。通过这种方式关联内容后，播放器并不会自动准备好，必须同时调用 **prepare()**让播放器做好播放的准备。

这时，音频已经做好播放的准备了。我们希望用 **MediaController** 处理所有的播放控制，

但 `MediaController` 只能用于实现 `MediaController.MediaPlayerControl` 接口的对象。奇怪的是, `MediaPlayer` 自己并没有实现这个接口, 因此我们指定了 `Activity` 去实现这个接口。这个接口中 7 个方法中的 6 个实际上都是由 `MediaPlayer` 实现的, 因此直接调用它们就可以了。

更新:

如果应用程序使用的是 API Level 5 或者更新的版本, 那么在 `MediaController.MediaPlayerControl` 接口中会新增三个方法:

```
canPause()  
canSeekBackward()  
canSeekForward()
```

这些方法只是简单地告诉系统是否允许这些操作, 因此我们的应用程序中的三个方法都返回了 `true`。如果使用的是低版本的 API, 这些方法不是必要的(这也是为什么没有在这些方法上面添加 `@Override` 标注), 但想要在新版本上获得更好的效果, 就需要实现这些方法了。

使用 `MediaController` 实现的最后方法为 `getBufferPercentage()`。想要获得这个数据, `Activity` 还需要实现 `MediaPlayer.OnBufferingUpdateListener`, 该接口会在缓冲百分比变化时更新缓冲百分比。

在实现 `MediaController` 时有一个技巧。这个控件是漂浮在它所在窗口中活动的 `view` 之上的, 而且一次只显示几秒钟。因此, 我们没有在 `content view` 的 XML 布局中初始化它, 而是使用代码进行初始化。`MediaController` 和 `contentview` 之间的纽带就是 `setAnchorView()`, 该方法还决定了 `MediaController` 在屏幕中的显示位置。本例中, 我们将它与根布局对齐, 所以会显示在屏幕的底部。如果 `MediaController` 对齐的是子 `view`, 它将会显示在子 `view` 的边上。

同样, 因为 `MediaController` 是单独的窗口, 所以不能在 `onCreate()` 中调用 `MediaController.show()`, 否则会导致一个致命的异常。`MediaController` 默认是隐藏的并且由用户激活。本例中, 我们覆写了 `Activity` 的 `onTouchEvent()` 方法, 当用户单击屏幕就会显示 `MediaController`。除非 `show()` 方法传入的参数是 0, 否则它都会在参数指定的时间之后消隐。如果调用 `show()` 时没有传入任何参数, 将使用默认的超时时间, 大概是 3 秒钟, 如图 4-4 所示。

现在音频播放的所有特性都是通过标准的 `MediaController` 控件实现的。本例中使用的 `setDataSource()` 版本使用了一个 `Uri` 作为参数, 这对于从 `ContentProvider` 或者远程地址加载音频非常合适。记住, 使用其他形式的 `setDataSource()` 也可以

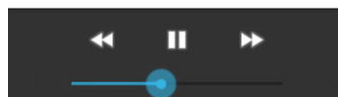
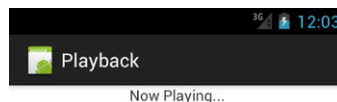


图 4-4 使用 `MediaController` 的 `Activity`

实现本地音频文件和资源的加载工作。

3. 视频播放

播放视频时，完整的播放控制通常包括播放、暂停和拖曳播放。此外，`MediaPlayer` 上必须有一个 `SurfaceHolder` 的引用来绘制视频的每一帧。正如前面示例中提到的一样，`Android` 提供了创建自定义视频播放体验的全部 API。但是，很多情况下最有效的方式还是使用 SDK 提供的类 `MediaController` 和 `VideoView` 去处理这些繁琐的工作。

让我们看一个在 `Activity` 中创建视频播放器的示例。参见程序清单 4-29。

程序清单 4-29 播放视频内容的 Activity

```
public class VideoActivity extends Activity {

    VideoView videoView;
    MediaController controller;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        videoView = new VideoView(this);

        videoView.setVideoURI( Uri.parse("URI_TO_REMOTE_VIDEO") );
        controller = new MediaController(this);
        videoView.setMediaController(controller);
        videoView.start();

        setContentView(videoView);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        videoView.stopPlayback();
    }
}
```

本例中，将一个远程视频地址的 URI 传给了 `VideoView`，让 `VideoView` 处理余下的事情。`VideoView` 同样可以被嵌入到较大的 XML 布局体系中，但通常都是单独使用并且全屏显示，所以通常在代码中将其设为布局中唯一的 `view`。

通过 `VideoView` 再与 `MediaController` 交互就非常简单了。`VideoView` 实现了 `MediaController.MediaPlayerControl` 接口，所以不再需要写其他任何代码就可以实现控制功能。`VideoView` 还可以在内部处理 `MediaController` 的对齐，使其可以显示在屏幕合适的位置上。

4. 处理重定向

最后看一下使用 `MediaPlayer` 类处理远程内容需要注意的事情。现在很多 Web 上的媒体内容服务器并不会公开暴露视频容器的 URL。出于防止追踪和安全考虑，公开的媒体

URL 通常需要一次或者多次的重定向才可以找到真正的媒体内容。**MediaPlayer** 并不能处理重定向的过程, 对于需要重定向的 URL, 它会返回错误。

如果找不到要播放内容的真正 URL, 那么在将 URL 传给 **MediaPlayer** 之前必须找到 URL 的重定向路径。程序清单 4-30 是一个简单的 **AsyncTask** 追踪器的示例, 它可以实现该功能。

程序清单 4-30 RedirectTracerTask

```
public class RedirectTracerTask extends AsyncTask<Uri, Void, Uri> {

    private VideoView mVideo;
    private Uri initialUri;

    public RedirectTracerTask(VideoView video) {
        super();
        mVideo = video;
    }

    @Override
    protected Uri doInBackground(Uri... params) {
        initialUri = params[0];
        String redirected = null;
        try {
            URL url = new URL(initialUri.toString());
            HttpURLConnection connection =
                (HttpURLConnection)url.openConnection();
            //连接后会追踪最终地址
            redirected = connection.getHeaderField("Location");

            return Uri.parse(redirected);
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    protected void onPostExecute(Uri result) {
        if(result != null) {
            mVideo.setVideoURI(result);
        } else {
            mVideo.setVideoURI(initialUri);
        }
    }
}
```

这个辅助类通过检查 HTTP 头来追踪 URL 重定向后的最终地址。如果目标 Uri 没有重定向, 后台操作将返回 null, 这种情况下原始的 Uri 会直接传给 **VideoView**。通过这个辅

助类，可以采用下面的方式把媒体的地址传给 `VideoView`：

```
VideoView videoView = new VideoView(this);
RedirectTracerTask task = new RedirectTracerTask(videoView);
Uri location = Uri.parse("URI_TO_REMOTE_VIDEO");

task.execute(location);
```

4.10 播放音效

4.10.1 问题

应用程序需要几个很短的低延迟的音效来响应与用户的交互。

4.10.2 解决方案

(API Level 1)

通过 `SoundPool` 将音频文件缓冲加载到内存中然后在响应用户操作时快速的播放。`Android` 框架提供了 `SoundPool` 来解码小音频文件并在内存中操作它们来进行音频的快速和重复播放。`SoundPool` 还有一些其他的特性，如可以在运行时控制音量和播放速度。声音文件可以放置在 `assets`、`resource` 或者设备的文件系统中。

4.10.3 实现机制

让我们看一下如何使用 `SoundPool` 加载一些音频并把它关联到 `Button` 的单击事件中。参见程序清单 4-31 和 4-32。

程序清单 4-31 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/button_beep1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Play Beep 1" />
    <Button
        android:id="@+id/button_beep2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Play Beep 2" />
    <Button
        android:id="@+id/button_beep3"
```

```

        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Play Beep 3" />
</LinearLayout>

```

程序清单 4-32 SoundPool 的 Activity

```

public class SoundPoolActivity extends Activity implements
    View.OnClickListener {

    private AudioManager mAudioManager;
    private SoundPool mSoundPool;
    private SparseIntArray mSoundMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //得到 AudioManager 系统服务
        mAudioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
        //设置声音池通过标准的扬声器每次只播放一个音频
        mSoundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);

        findViewById(R.id.button_beep1).setOnClickListener(this);
        findViewById(R.id.button_beep2).setOnClickListener(this);
        findViewById(R.id.button_beep3).setOnClickListener(this);

        //加载每个音频并把它们的 streamId 保存到一个 map 中
        mSoundMap = new SparseIntArray();
        AssetManager manager = getAssets();
        try {
            int streamId;
            streamId = mSoundPool.load(manager.openFd("Beep1.ogg"), 1);
            mSoundMap.put(R.id.button_beep1, streamId);

            streamId = mSoundPool.load(manager.openFd("Beep2.ogg"), 1);
            mSoundMap.put(R.id.button_beep2, streamId);

            streamId = mSoundPool.load(manager.openFd("Beep3.ogg"), 1);
            mSoundMap.put(R.id.button_beep3, streamId);
        } catch (IOException e) {
            Toast.makeText(this, "Error Loading Sound Effects",
                Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mSoundPool.release();
        mSoundPool = null;
    }
}

```

```

    }

    @Override
    public void onClick(View v) {
        //查找适合的音频的 ID
        int streamId = mSoundMap.get(v.getId());
        if(streamId > 0) {
            float streamVolumeCurrent =
                mAudioManager.getStreamVolume(AudioManager.STREAM_MUSIC);
            float streamVolumeMax =
                mAudioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
            float volume = streamVolumeCurrent / streamVolumeMax;
            //使用指定的音量播放音频，不循环播放并且使用标准的播放速度
            mSoundPool.play(streamId, volume, volume, 1, 0, 1.0f);
        }
    }
}

```

这个示例非常简单。Activity 首先将三个音频文件从应用程序的 `assets` 目录加载到 `SoundPool` 中。这个过程会将它们解码为原始 PCM 音频格式并缓存到内存中。每次使用 `load()` 将一个音频文件加载到声音池中时，都会返回一个 `stream` 标识符，后面则会使用这个标识符播放音频。我们通过 `SparseIntArray` 中保存 `key/value` 的方式将每个音频关联到一个特定的按钮上。

注意：

`SparseIntArray` (和它的同胞方法 `SparseBooleanArray`) 和 `Map` 类似，都是采用 `key/value` 的形式存储的。但是，在保存基本类型数据(如整型)时，它会更加高效，这是因为它避免了在自动装箱(`AutoBoxing`)时创建不必要的对象。如果有需要，应该选择这种方式来提高性能而不是使用 `Map`。

当用户按下一个按钮时，会查找到相应的 `stream` 标识符并使用 `SoundPool` 播放这个音频。因为 `SoundPool` 构造函数的 `maxStreams` 属性设置为 1，如果用户连续按下多个按钮，新声音将使得旧声音停止。如果这个属性值是递增的，则会同时播放多个声音。

`play()` 方法的参数允许每次访问时都可以配置声音。例如循环播放或者快速/慢速播放(相对于原始声音)这样的特性都可以在此处进行控制。

- 循环播放支持任何有限的播放次数，或者设置为 -1 代表无限播放。
- 播放速度支持任何介于 0.5 到 2.0(半速到两倍速度)之间的值。

如果想在某个特定时间内使用 `SoundPool` 动态改变加载到内存中的声音，不必重新创建声音池，可以使用 `unload()` 来移除声音池中的声音从而能够加载更多的声音。当 `SoundPool` 使用完成后，需要调用 `release()` 释放它的本地资源。

4.11 创建倾斜监控器

4.11.1 问题

应用程序不仅仅需要知道设备是横向和还是竖向的，还需要从设备的加速度计中得到反馈数据。

4.11.2 解决方案

(API Level 3)

使用 `SensorManager` 接收来自于加速度计传感器持续的反馈数据。`SensorManager` 为使用 Android 设备上的硬件传感器提供了一个通用的抽象接口。加速度计只是可能注册并获得定时更新的众多传感器中的一个。

4.11.3 实现机制

重点:

在模拟器中并没有加速度计这样的设备传感器。如果不能在 Android 设备上测试 `SensorManager` 代码的话，需要使用一个工具例如 `Sensor Simulator` 将传感器的事件植入到系统中。使用 `Sensor Simulator` 需要修改这个示例来使用一个不同的 `SensorManager` 进行测试。更多的信息可以参见本章最后的“实用工具推荐: `Sensor Simulator`”。

这个示例 `Activity` 使用 `SensorManager` 注册了一个加速度计，当该加速度计更新时会将数据显示在屏幕上。原始的 X/Y/Z 数据通过一个 `TextView` 显示在屏幕的底部，而设备的“倾斜”情况会通过 `TableLayout` 中的 4 个图形呈现出来。参见程序清单 4-33 和 4-34。

注意:

为了防止在移动和倾斜设备时 `Activity` 的屏幕也跟着旋转,推荐在应用程序的 `manifest` 中添加 `android:screenOrientation="portrait"` 或者 `android:screenOrientation="landscape"`。

程序清单 4-33 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TableLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="0,1,2">
        <TableRow
            android:layout_weight="1">
            <View
```

```

        android:id="@+id/top"
        android:layout_column="1"
    />
</TableRow>
<TableRow
    android:layout_weight="1">
    <View
        android:id="@+id/left"
        android:layout_column="0"
    />
    <View
        android:id="@+id/right"
        android:layout_column="2"
    />
</TableRow>
<TableRow
    android:layout_weight="1">
    <View
        android:id="@+id/bottom"
        android:layout_column="1"
    />
</TableRow>
</TableLayout>
<TextView
    android:id="@+id/values"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
/>
</RelativeLayout>

```

程序清单 4-34 监控倾斜的 Activity

```

public class TiltActivity extends Activity implements SensorEventListener {

    private SensorManager mSensorManager;
    private Sensor mAccelerometer;
    private TextView valueView;
    private View mTop, mBottom, mLeft, mRight;

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

        mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        mAccelerometer =
            mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        valueView = (TextView)findViewById(R.id.values);
        mTop = findViewById(R.id.top);
        mBottom = findViewById(R.id.bottom);
    }
}

```

```

        mLeft = findViewById(R.id.left);
        mRight = findViewById(R.id.right);
    }

    protected void onResume() {
        super.onResume();
        mSensorManager.registerListener(this, mAccelerometer,
            SensorManager.SENSOR_DELAY_UI);
    }

    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) { }

    public void onSensorChanged(SensorEvent event) {
        float[] values = event.values;
        float x = values[0] / 10;
        float y = values[1] / 10;
        int scaleFactor;

        if(x > 0) {
            scaleFactor = (int)Math.min(x * 255, 255);
            mRight.setBackgroundColor(Color.TRANSPARENT);
            mLeft.setBackgroundColor(Color.argb(scaleFactor, 255, 0, 0));
        } else {
            scaleFactor = (int)Math.min(Math.abs(x) * 255, 255);
            mRight.setBackgroundColor(Color.argb(scaleFactor, 255, 0, 0));
            mLeft.setBackgroundColor(Color.TRANSPARENT);
        }

        if(y > 0) {
            scaleFactor = (int)Math.min(y * 255, 255);
            mTop.setBackgroundColor(Color.TRANSPARENT);
            mBottom.setBackgroundColor(Color.argb(scaleFactor, 255, 0, 0));
        } else {
            scaleFactor = (int)Math.min(Math.abs(y) * 255, 255);
            mTop.setBackgroundColor(Color.argb(scaleFactor, 255, 0, 0));
            mBottom.setBackgroundColor(Color.TRANSPARENT);
        }
        //显示原始值
        valueView.setText(String.format("X: %1$1.2f, Y: %2$1.2f, Z: %3$1.2f",
            values[0], values[1], values[2]));
    }
}

```

从加速度计获取的设备在三个轴上的方向是这样定义的，看着设备的屏幕，从右上角算起：

- X: 横轴, 向右为正
- Y: 纵轴, 向上为正
- Z: 垂线, 远离你的方向为正

当 Activity 对用户可见时(在 `onResume()` 和 `onPause()` 之间), 会使用 `SensorManager` 注册加速度计从而接收来自加速度计的更新。注册时, `registerListener()` 的最后一个参数定义了更新频率。这里选择的是 `SENSOR_DELAY_UI`, 它是接收更新的最快频率, 每次更新都会直接修改 UI。

每当传感器有新值更新, 都会用 `SensorEvent` 值一起调用已注册监听器的 `onSensorChanged()` 方法。这个 `SensorEvent` 值包含了 X/Y/Z 轴上的加速度值。

科普知识:

加速度计是根据所受的力来计算加速度的。当设备静止时, 它所受的力就只有重力 ($\sim 9.8 \text{ m/s}^2$)。每个轴上的输出值就是这个力(指向地面)与各个方向向量的乘积。当二者平行时, 这个值就是其最大值 ($\sim 9.8-10$)。当二者垂直时, 这个值就是其最小值 (~ 0.0)。因此当把设备平放在桌子上时, X 轴和 Y 轴的读数大约是 0.0, 而 Z 轴大约为 9.8。

本示例应用程序使用 `TextView` 在屏幕的底部显示出了每个轴的原始加速度值。此外, 还有一个上/下/左/右显示的 4 个 `View` 的网格, 我们会根据设备的方向, 按比例调整这个网格的背景色。当设备完全处于水平状态时, X 轴和 Y 轴的值接近于 0, 整个屏幕会显示为白色。当设备倾斜是, 屏幕较低的那一端就开始变成红色, 在整个屏幕竖直后, 将完全变成红色。

提示:

试着修改本示例来使用其他的频率值, 如 `SENSOR_DELAY_NORMAL`。观察这些修改是如何影响更新频率的。

你还可以晃动设备, 看看随着设备的晃动, 每个方向的加速度是如何变化的。

4.12 监控罗盘的方向

4.12.1 问题

通过监控设备的罗盘传感器, 应用程序想要知道用户面对的大致方向。

4.12.2 解决方案

(API Level 3)

我们要再次求助 `SensorManager`。Android 并没有提供真正的“罗盘”传感器; 相反, 而是基于其他的传感器数据使用一些必要的方法推断出设备指向的方向。这种情况下, 会使用设备的磁场传感器和加速度计一起确定用户面向的方向。

然后可以使用 `SensorManager` 的 `getOrientation()` 得到用户在地球上的方向。

4.12.3 实现机制

重点:

在模拟器中并没有加速度计这样的设备传感器。如果不能在 Android 设备上测试 SensorManager 代码的话, 需要使用一个工具例如 Sensor Simulator 将传感器的事件植入到系统中。使用 Sensor Simulator 需要修改这个示例来使用一个不同的 SensorManager 进行测试, 更多的信息可以参见本章最后的“实用工具推荐: Sensor Simulator”。

同前面的加速度计示例一样, 我们使用 SensorManager 注册更新我们要使用的传感器(本例中, 要用到两个传感器), 然后在 onSensorChanged() 中处理结果。本例从设备的相机视角来计算和显示用户的方向, 这可以用来实现增强现实类的应用程序。

程序清单 4-35 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/direction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textSize="64dip"
        android:textStyle="bold"
    />
    <TextView
        android:id="@+id/values"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
    />
</RelativeLayout>
```

程序清单 4-36 监控用户方向的 Activity

```
public class CompassActivity extends Activity implements SensorEventListener {

    private SensorManager mSensorManager;
    private Sensor mAccelerometer, mField;
    private TextView valueView, directionView;

    private float[] mGravity;
    private float[] mMagnetic;

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

```

mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
mAccelerometer =
    mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
mField = mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

valueView = (TextView)findViewById(R.id.values);
directionView = (TextView)findViewById(R.id.direction);
}

protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mAccelerometer,
        SensorManager.SENSOR_DELAY_UI);
    mSensorManager.registerListener(this, mField,
        SensorManager.SENSOR_DELAY_UI);
}

protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
}

private void updateDirection() {
    float[] temp = new float[9];
    float[] R = new float[9];
    //将旋转矩阵加载到 R 中
    SensorManager.getRotationMatrix(temp, null, mGravity, mMagnetic);
    //映射为相机的视角
    SensorManager.remapCoordinateSystem(temp, SensorManager.AXIS_X,
        SensorManager.AXIS_Z, R);
    //返回方向值
    float[] values = new float[3];
    SensorManager.getOrientation(R, values);
    //转换为角度
    for(int i=0; i < values.length; i++) {
        Double degrees = (values[i] * 180) / Math.PI;
        values[i] = degrees.floatValue();
    }
    //显示罗盘方向
    directionView.setText( getDirectionFromDegrees(values[0]) );
    //显示原始值
    valueView.setText(
        String.format("Azimuth: %1$1.2f, Pitch: %2$1.2f, Roll: %3$1.2f",
            values[0], values[1], values[2]));
}

private String getDirectionFromDegrees(float degrees) {
    if(degrees >= -22.5 && degrees < 22.5) { return "N"; }
    if(degrees >= 22.5 && degrees < 67.5) { return "NE"; }
}

```

```

        if(degrees >= 67.5 && degrees < 112.5) { return "E"; }
        if(degrees >= 112.5 && degrees < 157.5) { return "SE"; }
        if(degrees >= 157.5 || degrees < -157.5) { return "S"; }
        if(degrees >= -157.5 && degrees < -112.5) { return "SW"; }
        if(degrees >= -112.5 && degrees < -67.5) { return "W"; }
        if(degrees >= -67.5 && degrees < -22.5) { return "NW"; }

        return null;
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) { }
    public void onSensorChanged(SensorEvent event) {
        switch(event.sensor.getType()) {
            case Sensor.TYPE_ACCELEROMETER:
                mGravity = event.values.clone();
                break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                mMagnetic = event.values.clone();
                break;
            default:
                return;
        }

        if(mGravity != null && mMagnetic != null) {
            updateDirection();
        }
    }
}

```

这个示例 Activity 在屏幕的底部实时地显示通过传感器计算返回的三个原始数值。此外，还会从这些原始数值中计算用户当前面向的罗盘方向并显示在屏幕中央。当收到传感器的更新指令时，各个传感器的本地拷贝都会更新。只要接收到了我们要使用的两个传感器中其中一个的信息，就会更新 UI。

`updateDirection()` 是进行所有核心工作的地方。`SensorManager.getOrientation()` 提供了显示方向所需的输出信息。这个方法不会返回数据，而是传入这个方法中一个空的浮点型数组来填充三个角度值，它们依次代表了：

- Azimuth
 - 围绕指向地轴的旋转角度
 - 这个值是本示例感兴趣的值
- Pitch
 - 围绕指向西方轴的旋转角度
- Roll
 - 围绕指向磁极北极的旋转角度

传入 `getOrientation()` 的一个参数代表旋转矩阵的浮点型数组。旋转矩阵表示了设备当前的坐标系是如何旋转的，这样就可以根据参考坐标提供合适的旋转角度。设备方向的旋

转矩阵是通过 `getRotationMatrix()` 方法获得的，它的输入是加速度计和磁场传感器的最新值。和 `getOrientation()` 一样，它也返回 `void`；同时会将一个长度为 9 或者 16(代表 3x3 或者 4x4 的矩阵)的数组作为参数传递给该方法，将方法会将结果填入这个数组。

最后，我们希望计算出来的方向能以相机视角为准。所以我们使用了 `remapCoordinateSystem()` 对得到的旋转数据进行进一步的转换。这个方法依次有四个参数：

- (1) 输入的数据代表了要转换的矩阵
- (2) 如何将设备的 x 轴转换为世界坐标
- (3) 如何将设备的 y 轴转换为世界坐标
- (4) 用户填充结果的空数组

本例中，我们不需要处理 x 轴，所以直接将 X 映射到 X。但是，需要将设备的 y 轴(纵轴)指向世界坐标系的 z 轴(指向地轴)。因为用户在拍照时是直握相机在屏幕上预览的，所以只有这样才能让获得的旋转矩阵与用户手握的方向相匹配。

计算出角度数据后，会对数据进行转换并将结果显示在屏幕上。`getOrientation()` 的输入单位为弧度，因此在显示之前首先需要把它转换为角度。此外，还需要将方向角的值转换为罗盘的方向；`getDirectionFromDegrees()` 辅助方法可以根据当前的角度返回设备的方向。顺时针转一圈，方向角会从 0° 变为 180° ，方向也会从北变到南。继续转一圈，方向角会从 -180° 变为 0° ，方向从南变到北。

4.13 在媒体内容中获取元数据

4.13.1 问题

应用程序需要从设备的媒体内容中得到截图的缩略图或者其他元数据

4.13.2 解决方案

(API Level 10)

使用 `MediaMetadataRetriever` 读取媒体文件并返回有用的信息。这个类可以读取和跟踪专辑或艺术家数据、或者内容数据本身。此外，对于视频文件，无论是某一特定时间的帧或者 Android 觉得具有代表性的任意一帧，都可以使用 `MediaMetadataRetriever` 抓取该帧的截图。

对于应用程序需要与设备上很多的媒体内容打交道以及需要显示媒体的附加数据来丰富 UI 来说，`MediaMetadataRetriever` 是非常好的选择。

4.13.3 实现机制

程序清单 4-37 和 4-38 展示了如何访问设备的附加元数据。

程序清单 4-37 res/layout/main.xml

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:id="@+id/button_select"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Pick Video"
        android:onClick="onSelectClick" />
    <TextView
        android:id="@+id/text_metadata"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button_select"
        android:layout_margin="15dp" />
    <ImageView
        android:id="@+id/image_frame"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_margin="10dp" />
</RelativeLayout>

```

程序清单 4-38 MediaMetadataRetriever 的 Activity

```

public class MetadataActivity extends Activity {
    private static final int PICK_VIDEO = 100;

    private ImageView mFrameView;
    private TextView mMetadataView;

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

        mFrameView = (ImageView) findViewById(R.id.image_frame);
        mMetadataView = (TextView) findViewById(R.id.text_metadata);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        Intent data) {
        if(requestCode == PICK_VIDEO && resultCode == RESULT_OK
            && data != null) {
            Uri video = data.getData();
            MetadataTask task = new MetadataTask(this, mFrameView,
                mMetadataView);

```

```

        task.execute(video);
    }
}

public void onSelectClick(View v) {
    Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
    intent.setType("video/*");
    startActivityForResult(intent, PICK_VIDEO);
}

public static class MetadataTask extends AsyncTask<Uri, Void, Bundle> {
    private Context mContext;
    private ImageView mFrame;
    private TextView mMetadata;
    private ProgressDialog mProgress;

    public MetadataTask(Context context, ImageView frame,
        TextView metadata) {
        mContext = context;
        mFrame = frame;
        mMetadata = metadata;
    }

    @Override
    protected void onPreExecute() {
        mProgress = ProgressDialog.show(mContext, "",
            "Analyzing Video File...", true);
    }

    @Override
    protected Bundle doInBackground(Uri... params) {
        Uri video = params[0];
        MediaMetadataRetriever retriever = new MediaMetadataRetriever();
        retriever.setDataSource(mContext, video);

        Bitmap frame = retriever.getFrameAtTime();

        String date = retriever.extractMetadata(
            MediaMetadataRetriever.METADATA_KEY_DATE);
        String duration = retriever.extractMetadata(
            MediaMetadataRetriever.METADATA_KEY_DURATION);
        String width = retriever.extractMetadata(
            MediaMetadataRetriever.METADATA_KEY_VIDEO_WIDTH);
        String height = retriever.extractMetadata(
            MediaMetadataRetriever.METADATA_KEY_VIDEO_HEIGHT);

        Bundle result = new Bundle();
        result.putParcelable("frame", frame);
        result.putString("date", date);
        result.putString("duration", duration);
    }
}

```

```

        result.putString("width", width);
        result.putString("height", height);
        return result;
    }

    @Override
    protected void onPostExecute(Bundle result) {
        if(mProgress != null) {
            mProgress.dismiss();
            mProgress = null;
        }

        Bitmap frame = result.getParcelable("frame");
        mFrame.setImageBitmap(frame);
        String metadata = String.format(
            "Video Date: %s\nVideo Duration: %s\nVideo Size: %s x %s",
            result.getString("date"),
            result.getString("duration"),
            result.getString("width"),
            result.getString("height") );
        mMetadata.setText(metadata);
    }
}

```

在这个示例中，用户可以从设备中选择一个要处理的视频文件。当收到一个有效的视频 Uri 后，Activity 开始用一个 AsyncTask 从视频中解析元数据。由于这个过程需要几秒钟或者更长的时间，而且我们不希望在此过程中阻塞 UI 线程，所以创建了一个 AsyncTask 去完成它。

后台的 Task 会创建 MediaMetadataRetriever 并将选择的视频作为它的数据源。然后调用 getFrameAtTime() 返回视频中一帧的 Bitmap 图像。这个方法在 UI 中创建一个视频的截图时非常有用。我们调用的这个方法是没有参数的版本，它返回的帧是半随机的。如果对某一特定帧更加感兴趣的话，可以使用另一个版本，该版本的参数为需要的视频帧的播放时间(单位为毫秒)。这种情况下，会返回视频中距离所指定的时间最近的关键帧。

除了帧的图像，我们还收集了一些视频的基本信息，包括视频创建的时间、视频时长和视频大小。所有的这些数据被打包到一个 bundle 中，并由后台线程传递出来。因为 task 的 onPostExecute() 方法是在主线程上调用的，所以可以通过它使用这些传递出来的数据更新 UI。

4.14 实用工具推荐：Sensor Simulator

由于大多数电脑上没有加速度计、罗盘传感器或者光感器，因此 Google 的 Android 模拟器并不能直接支持传感器的模拟。相反，SDK 工具允许开发者将 Android 设备和一个运行有特殊应用程序(SdkController)的机器相连接，从而可以通过 ADB 将数据转发到模拟器

中。这里会有一个矛盾的地方,就是如果开发者使用模拟器进行测试,很有可能没有 Android 设备来模拟传感器输入(除非他使用模拟器是想测试 Android 新版本的功能,而这些功能是他的设备所不支持的)。

注意:

更多关于 SdkController 和让设备支持传感器或多点触摸的信息,可以访问 <http://tools.android.com/tips/hardware-emulation>。

如果应用程序需要与传感器交互,但又只能在模拟器上测试的话,可以用 Sensor Simulator 解决这个问题。

Sensor Simulator(<http://code.google.com/p/openintents/wiki/SensorSimulator>)是一个开源工具,它可以模拟传感器数据,让这些数据可以用于应用程序的测试。它现在支持加速度计、罗盘/磁场、方向、温度、光线、近距离、压力、重力、线性加速、旋转向量、陀螺仪传感器。这些传感器都是可以配置的。

4.15 获得 Sensor Simulator

SensorSimulator 是以单独的 ZIP 压缩包发布的。用浏览器访问 <http://code.google.com/p/openintents/downloads/list?q=sensorsimulator> 并且单击 sensorsimulator-2.0-rc1.zip 下面的链接,在接下来的界面中单击 sensorsimulator-2.0-rc1.zip 下载这个 692KB 的文件。

解压完成后,你会得到一个 sensorsimulator-2.0-rc1 主目录,它有如下子目录:

- **bin:** 包含 SensorRecordFromDevice-2.0-rc1.apk(可以记录真实 Android 设备传感器数据的应用程序)、sensorsimulator-2.0-rc1.jar(Java 应用程序,用来选择和配置要模拟的传感器,并将测试数据发送到模拟器上)、SensorSimulatorSettings-2.0-rc1.apk(可以与桌面应用程序进行通信并启动一个测试的应用程序)三个可执行文件和相应的说明文件。
- **lib:** 包含 sensorsimulator-lib-2.0-rc1.jar 库。
- **release:** 包含用于编译发布文件(如 sensorsimulator-2.0-rc1.zip)的构建脚本。
- **samples:** 包含了如何在你的应用程序中引入 SensorSimulator 的相关内容。
- **SensorRecordFromDevice:** 包含了一个构建 SensorRecordFromDevice-2.0-rc1.apk 的 Eclipse 项目。
- **SensorSimulator:** 包含了一个构建 sensorsimulator-2.0-rc1.jar 的 Eclipse 项目。
- **SensorSimulatorSettings:** 包含了一个构建 SensorSimulatorSettings-2.0-rc1.apk 的 Eclipse 项目。

4.16 启动 Sensor Simulator Settings 和 Sensor Simulator

现在,你已经下载并解压了 Sensor Simulator 发布包,想要启动这个软件,可以执行以

下步骤:

(1) 如果还没运行 Android 模拟器的话,先启动 Android 模拟器。例如,在命令行执行 `emulator -avd AVD1`。这个命令的前提是你之前已经在第 1 章创建了 AVD1。

(2) 在模拟器上安装 `SensorSimulatorSettings-2.0-rc1.apk`, 执行 `adb install SensorSimulatorSettings-2.0-rc1.apk` 即可。这个命令的前提是 `adb` 工具已经加入到了 `PATH` 环境变量中并且当前目录是 `bin` 目录。当 APK 成功安装后会输出一个安装成功的消息(你也许还想安装 `SensorRecordFromDevice-2.0-rc1.apk`, 但没有必要这样做)。

(3) 单击应用程序启动器界面的 `Sensor Simulator` 图标启动应用程序。

(4) 启动 `bin` 目录下的 `Sensor Simulator` 的 Java 应用程序, 也就是 `sensorsimulator-2.0-rc1.jar`。例如, 在 Windows 环境下, 双击该文件名即可。图 4-5 突出显示了应用程序启动器界面中的 `Sensor Simulator` 图标。

单击该 icon。图 4-6 展示了 `Sensor Simulator Settings` 的 activity, 它分为两个界面: `Settings` 和 `Testing`。

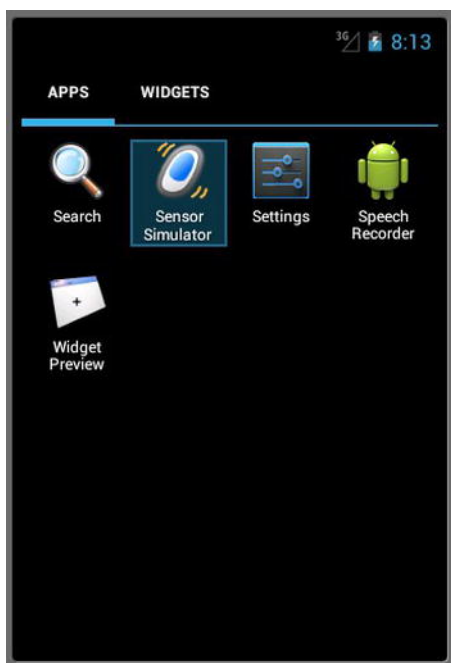


图 4-5 在应用程序启动器界面突出显示了 `Sensor Simulator` 图标

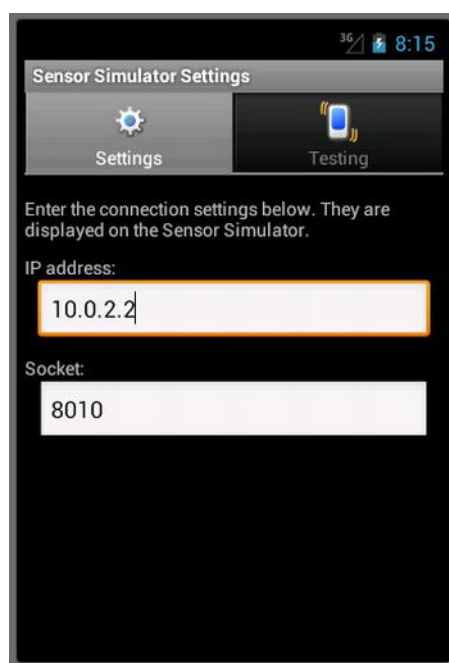


图 4-6 IP 地址默认为 10.0.2.2 而 socket 端口号默认为 8010

`Settings` 界面可以输入与 `Sensor Simulator` 应用程序通信的信息。图 4-7 展示了应用程序的 UI。

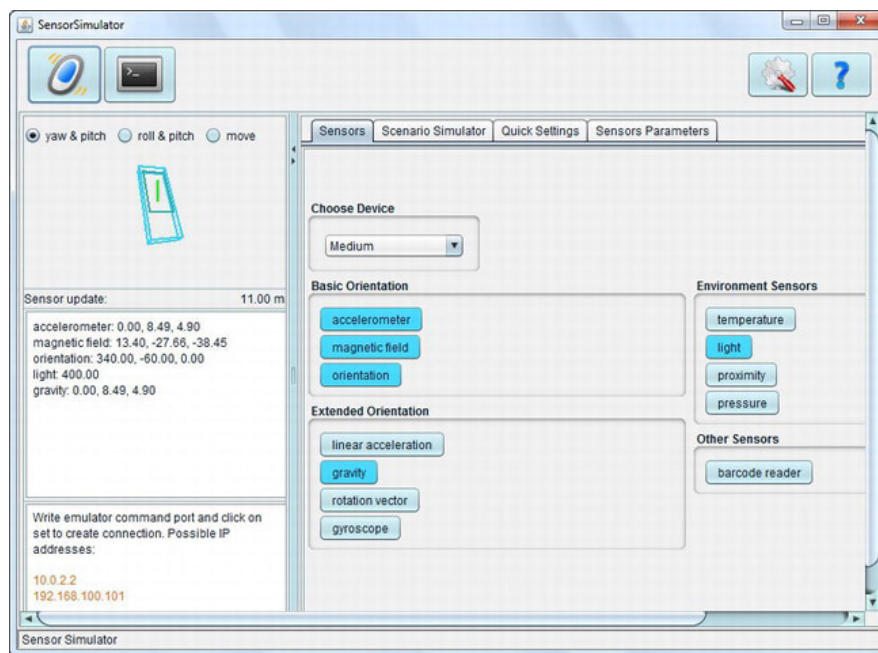


图 4-7 在 Sensor Simulator 应用程序的界面中配置传感器以及向模拟器发送传感器数据

在 Sensor Simulator 界面的顶部有一个按钮菜单，其中按钮是左右排列的，通过它们可以选择 Sensor Simulator、Telnet 和 Settings 界面。还可以通过你自己的浏览器来查看帮助信息。

- Sensor Simulator 显示了设备信息、可用的传感器以及可与模拟器进行通信的 IP 地址列表。它还显示了 Sensors、Scenario Simulator、Quick Settings、Sensors Parameters 共 4 个面板：
 - Sensors 用于选择启用哪个传感器。可以单独选择需要的传感器也可以选择某个设备支持的所有传感器。
 - Scenario Simulator 用于记录真实设备的模拟场景或者创建和编辑一个模拟场景，并且可以在设备或者模拟器上播放一个已记录的模拟场景(需要安装 SensorRecord-FromDevice-2.0-rc1.apk 应用程序)
 - Quick Settings 可用来设置设备的方向、温度、光线和压力
 - Sensors Parameters 用来进一步设置方向、温度、光线和压力的参数，还可以设置条形码阅读器的参数。
- Telnet 界面可以控制模拟器的 GPS 位置和电量级别。
- Settings 可以提供其他的配置、如传感器的更新间隔时长。
- Help 按钮会指向浏览器上的 <http://openintents.org/en/node/885> 页面。

Testing 界面可以连接到 Sensor Simulator 应用程序并且接收传感器数据。图 4-8 展示了这个界面。

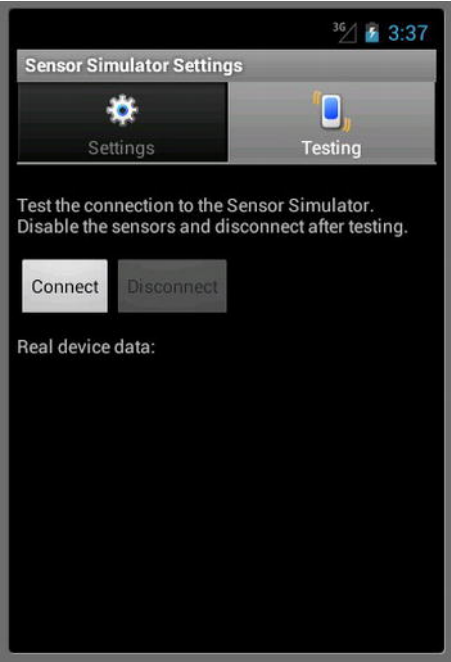


图 4-8 单击 Connect 连接到 Sensor Simulator 应用程序并且开始接收测试数据

从屏幕中可以看到，必须单击 Connect 按钮去连接 Sensor Simulator，Sensor Simulator 此刻必须是正在运行的(稍后可以单击 Disconnect 来关闭连接)。

单击 Connect 后，Testing 界面就会显示传感器和相关的值。这些信息在修改 Sensor Simulator 中的相应值时也会跟着更新。图 4-9 展示了这种情况。

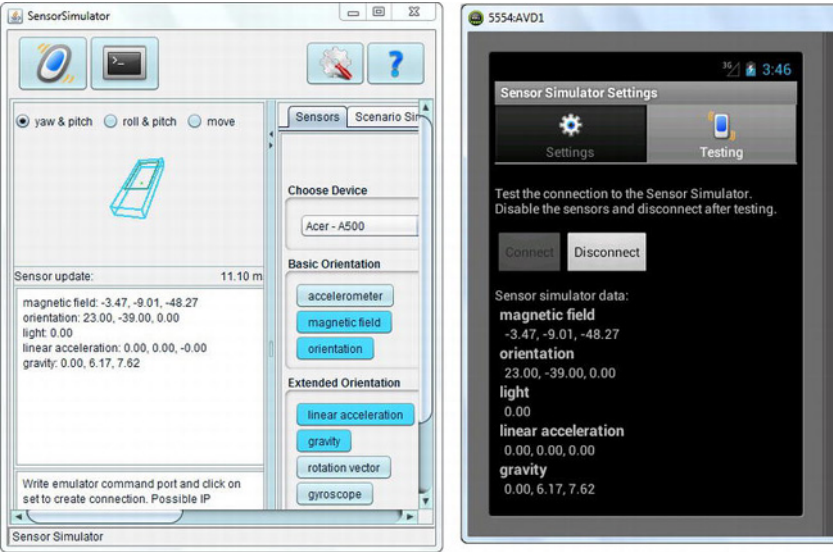


图 4-9 Sensor Simulator 应用程序正在向 Sensor Simulator 发送传感器数据

4.17 在自己的应用程序中访问 Sensor Simulator

Sensor Simulator 可以帮助你学习如何用 Sensor Simulator 将测试数据发送到应用程序,但这还不是你自己的应用程序。在某些情况下,需要将访问这些工具的代码整合到你自己的 activity 中。Google 提供了如下的指南可以让你的应用程序访问 Sensor Simulator:

- (1) 将 lib 目录下的 Jar 文件(例如, sensorsimulator-lib-2.0-rc1.jar)添加到你的项目中。
- (2) 在你的源代码中导入以下 Sensor Simulator 类型。

```
import org.openintents.sensorsimulator.hardware.Sensor;
import org.openintents.sensorsimulator.hardware.SensorEvent;
import org.openintents.sensorsimulator.hardware.SensorEventListener;
import org.openintents.sensorsimulator.hardware.SensorManagerSimulator;
```

(3) 将 activity 的 onCreate() 中的 SensorManager.getSystemService() 方法调用修改为 SensorManagerSimulator.getSystemService() 方法,例如,需要将 mSensorManager=(SensorManager) getSystemService(SENSOR_SERVICE); 替换为 mSensorManager=SensorManagerSimulator.getSystemService(this,SENSOR_SERVICE);。

(4) 用之前在 SensorSimulatorSettings 中设置的参数连接 Sensor SimulatorJava 应用程序: mSensorManager.connectSimulator();

(5) 其他代码不动。但记住要在 onResume() 中注册传感器并在 onStop() 中解除注册:

```
@Override
protected void onResume()
{
    super.onResume();
    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
        SensorManager.SENSOR_DELAY_FASTEST);
    mSensorManager.registerListener(this,
        mSensorManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE),
        SensorManager.SENSOR_DELAY_FASTEST);
}
@Override
protected void onStop()
{
    mSensorManager.unregisterListener(this);
    super.onStop();
}
```

(6) 最后,需要实现 SensorEventListener 接口:

```

class MySensorActivity extends Activity implements SensorEventListener
{
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        int sensor = event.type;
        float[] values = event.values;
        //使用传感器数据做一些事情
    }
}

```

注意:

SensorManagerSimulator 源于 Android 的 SensorManager 类并且实现了和 SensorManager 完全一样的方法。为了回调, 实现了新的 SensorEventListener 接口, 它和 Android 标准的 SensorEventListener 是类似的。

在没有连接 Sensor Simulator 应用程序时, 接收的都是真正的传感器数据: org.openintents.hardware.SensorManagerSimulator 会自动调用系统服务返回的 SensorManager 实例来接收真实的传感器数据。

4.18 小结

这些范例展示了如何在 Android 使用地图、用户位置和设备传感器数据, 将用户的周围环境信息整合到应用程序中。我们还讨论了如何使用设备的摄像头和麦克风, 让用户采集周边的信息, 并解析这些信息。最后, 通过使用媒体 API, 学习了如何获取用户本地拍摄或是从网络上下下载的多媒体资源, 以及如何在应用程序中播放这些多媒体。在第 5 章中, 我们将讨论使用 Android 的一些持久化技术在设备上存储非易失性的数据。

第 5 章

数据持久化

尽管现在流行将用户的数据转移到云上，但移动应用程序时间不稳定的本质决定了它必须要将部分用户数据持久地保存在本地设备上。这些数据可能是缓存的 Web Service 响应结果以确保能在离线状态下也能被访问，也可能是用户对特定应用程序行为的设置。Android 提供了一系列有用的框架，避免了用文件或数据库来实现信息的持久化。

5.1 制作设置界面

5.1.1 问题

在应用程序中，需要通过一种简单的方式存储、修改和显示用户设置和应用程序配置。

5.1.2 解决方案

(API Level 1)

通过 PreferenceActivity 和 XML Preference 文件就可以解决用户界面、键/值组合以及数据持久化的问题。通过这种方式可以创建和 Android 设备中的 Settings 应用程序一样的 UI，从而保持用户体验的一致性。

在 XML 内部，可以用 PreferenceScreen、PreferenceCategory 及相关 Preference 元素定义一个或者多个屏幕界面。而 Activity 只需要很少的代码就可以加载这个 XML 文件并呈现给用户。

5.1.3 实现机制

程序清单 5-1 和 5-2 展示了一个 Android 应用程序的基本设置。XML 中定义了两个屏幕界面，其中有 Android 框架支持的各种常用的设置类型。注意，其中一个屏幕是内嵌在另一个屏幕之中的，当用户在主屏幕上单击相关联的列表项时就会显示内部嵌套的屏幕。

程序清单 5-1 res/xml/settings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">
    <EditTextPreference
        android:key="namePref"
        android:title="Name"
        android:summary="Tell Us Your Name"
        android:defaultValue="Apress"
    />
    <CheckBoxPreference
        android:key="morePref"
        android:title="Enable More Settings"
        android:defaultValue="false"
    />
    <PreferenceScreen
        android:key="moreScreen"
        android:title="More Settings"
        android:dependency="morePref">
        <ListPreference
            android:key="colorPref"
            android:title="Favorite Color"
            android:summary="Choose your favorite color"
            android:entries="@array/color_names"
            android:entryValues="@array/color_values"
            android:defaultValue="GRN"
        />
        <PreferenceCategory
            android:title="Location Settings">
            <CheckBoxPreference
                android:key="gpsPref"
                android:title="Use GPS Location"
                android:summary="Use GPS to Find You"
                android:defaultValue="true"
            />
            <CheckBoxPreference
                android:key="networkPref"
                android:title="Use Network Location"
                android:summary="Use Network to Find You"
                android:defaultValue="true"
            />
        </PreferenceCategory>
    </PreferenceScreen>
</PreferenceScreen>

```

程序清单 5-2 res/values/arrays.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="color_names">
        <item>Black</item>
    </string-array>

```

```

        <item>Red</item>
        <item>Green</item>
    </string-array>
    <string-array name="color_values">
        <item>BLK</item>
        <item>RED</item>
        <item>GRN</item>
    </string-array>
</resources>

```

首先注意创建 XML 文件的方式。虽然这个资源文件可以放到任何的目录中(如 res/layout), 但通常都是把它们放到项目的 xml 通用目录中。

同样需要注意的是我们为每个 Preference 对象都提供了 android:key 属性, 而没有使用 android:id。当在应用程序的其他地方通过 SharedPreferences 对象引用这些设置值时, 就可以使用这些 key 了。另外, PreferenceActivity 有一个 findPreference()方法可以获得一个已经 inflate 的 Preference 的引用, 它比 findViewById()效率要高, findPreference()的参数也是 key。

Inflated 完成之后, PreferenceScreen 显示了带有以下三个条目(按顺序)的列表:

(1) Name 条目

- 它是一个 EditTextPreference 实例, 保存了一个字符串值。
- 单击这个条目会显示一个文本框, 用户可以在该文本框中输入新的设置值。

(2) Enable More Settings 条目, 旁边有一个复选框

- 它是一个 CheckBoxPreference 实例, 保存了一个布尔值。
- 单击这个条目将改变复选框的状态。

(3) More Settings 条目

- 单击这个条目将会加载另一个有更多条目的 PreferenceScreen。

当用户单击“More Settings”条目后, 会显示第二个设置屏幕, 它还有三个条目: 一个 ListPreference 条目和两个 CheckBoxPreferences, 这两个 CheckBoxPreferences 通过一个 PreferenceCategory 归类到了一起。PreferenceCategory 就是将真正的设置内容组合到一起并设置一个标题, 没有其他作用。

这个示例的最后一种设置类型为 ListPreference。这个条目需要两个数组参数(它们可以为同一个数组), 代表了供用户选择的选项集。android:entries 数组用于显示可读性很强的选项清单, 而 android:entryValues 数组则是实际保存的选项值。

所有的设置条目都可以设置默认的值(可选的), 但这个值不会被自动加载, 而是在显示 PreferenceActivity、XML 文件首次 inflate 或者调用 PreferenceManager.setDefaultValues()时才会加载。

现在让我们看一下 PreferenceActivity 是如何加载和管理这些设置的, 参见程序清单 5-3。

程序清单 5-3 PreferenceActivity 实战

```
public class SettingsActivity extends PreferenceActivity {
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //从 XML 中加载设置数据
    addPreferencesFromResource(R.xml.settings);
}
}

```

想要向用户显示设置界面并允许他们修改设置选项，只需要调用 `addPreferencesFromResource()` 即可。在 `PreferenceActivity` 不必调用 `setContentView()`，因为 `addPreferencesFromResource()` 就会 inflate XML 并显示它。不过也可以用一个含有 `ListView` 的布局，设置好 `ListView` 的 `android:id="@android:id/list"` 属性，这样 `PreferenceActivity` 就能够从中加载设置项。

为了单独控制访问，也可以将设置条目放到列表中。我们将“Enable More Settings”放到了列表中只是为了可以让用户启用或者禁用第二个 `PreferenceScreen`。为了实现这个目标，内嵌的 `PreferenceScreen` 包含了一个 `android:dependency` 属性，它根据另一个设置的状态决定是否要启用这个设置界面。当该属性指向的设置不存在或者为 `false` 时，这个设置界面就是禁用的。

当这个 `Activity` 加载完成，你会看到与图 5-1 类似的界面。

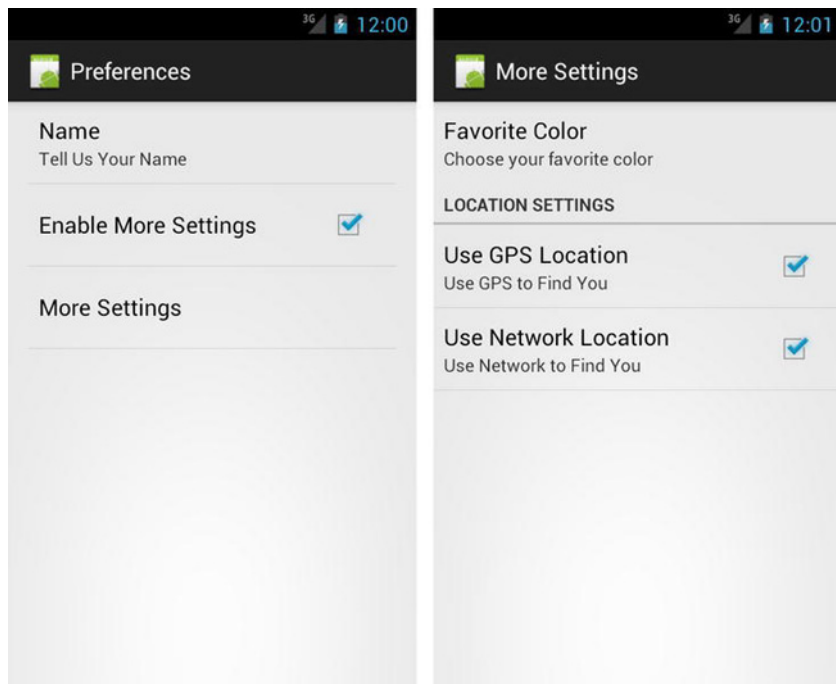


图 5-1 PreferenceScreen 实战

首先显示根 `PreferenceScreen`(左侧)。如果用户单击“MoreSettings”，就会显示第二个界面(右侧)。

1. 加载默认值和访问设置

通常情况下, `PreferenceActivity`(例如本例中的 `PreferenceActivity`)并不是一个应用程序的根界面。如果设置了默认值, 在用户访问 `Settings`(加载默认值的第一种情况)之前, 应用程序的其他部分就会经常访问这些默认值。所以, 如果要在应用程序中预先加载默认的设置值时, 可以调用以下方法:

```
PreferenceManager.setDefaultValues(Context context, int resId, boolean readAgain);
```

这个方法可能会被调用多次, 但默认值不会被反复加载。或许可以把它放到主 `Activity` 中, 这样就可以在首次启动时被调用, 或者放在所有要访问的 `shared preferences` 代码前面。

采用这种机制存储的 `Preferences` 会保存到默认的 `shared preferences` 对象中, 而 `shared preferences` 对象可以通过 `PreferenceManager.getDefaultSharedPreferences(Context context)` 来访问, 传入相应的 `Context` 即可。

下面的实例 `Activity` 会加载前面示例中的默认设置并访问其中的一些值, 参见程序清单 5-4。

程序清单 5-4 加载默认设置的 Activity

```
public class HomeActivity extends Activity {

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

        //加载默认设置
        PreferenceManager.setDefaultValues(this, R.xml.settings, false);
    }

    @Override
    public void onResume() {
        super.onResume();
        //访问当前的设置
        SharedPreferences settings =
            PreferenceManager.getDefaultSharedPreferences(this);

        String name = settings.getString("namePref", "");
        boolean isMoreEnabled = settings.getBoolean("morePref", false);
    }
}
```

调用 `setDefaultValues()` 会将 XML 文件中所有带 `android: defaultValue` 属性的设置项目的默认值保存到设置中。这样即使在用户尚未访问过设置界面的情况下, 应用程序也能够访问这些设置。

然后可以通过 `SharedPreferences` 对象的一系列类型化的访问器方法访问这些值。每个访问器方法都需要设置的 `key` 和一个默认值, 如果该 `key` 不存在的话就会返回这个默认值。

2. PreferenceFragment

(API Level 11)

从 Android 3.0 开始，以 PreferenceFragment 的形式引入了一种新的创建设置界面的方式。这个类并不是 android 支持库中的类，只有应用程序的目标版本最低为 API Level 11 时，才可以使用它来代替 PreferenceActivity。程序清单 5-5 和 5-6 修改了之前的示例，这里使用了 PreferenceFragment。

程序清单 5-5 包含 Fragment 的 Activity

```
public class MainActivity extends Activity {

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

        FragmentTransaction ft = getFragmentManager().beginTransaction();
        ft.add(android.R.id.content, new PreferenceFragment());
        ft.commit();
    }
}
```

程序清单 5-6 新的 PreferenceFragment

```
public class SettingsFragment extends PreferenceFragment {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //从 XML 中加载设置数据
        addPreferencesFromResource(R.xml.settings);
    }
}
```

现在，原来的设置都放到了 PreferenceFragment 中，PreferenceFragment 会像以前一样管理它们。另一个需要修改的地方就是 Fragment 不能自己单独存在，它必须包含在一个 Activity 中，因此我们创建了一个新的关联该 Fragment 的根 Activity。

5.2 简单数据存储

5.2.1 问题

应用程序需要以一种简单而快速的方式在持久存储器中保存一些基本数据，如数字或者字符串。

5.2.2 解决方案

(API Level 1)

通过 `SharedPreferences` 对象，应用程序可以快速创建一个或多个数据存储位置，稍后则可以在这些位置保存或者查询数据。实际上，这些对象是以 XML 文件的形式存储在应用程序的用户数据区。但是，与直接从文件中读取和写入数据不同，`SharedPreferences` 提供了一个非常高效的框架来持久保存基本数据类型。

最好是创建多个 `SharedPreferences` 对象，而不是将所有的数据都保存到默认的 `SharedPreferences` 对象中，尤其是在数据只需要保存一段时间的情况下更该如此。记住，所有的设置都是以 XML 的形式存储的，`PreferenceActivity` 也是保存在默认位置的，那么如何保存一组相关联的内容(如已经登录的用户)呢？当该用户退出时，将需要删除它所有保存的数据。如果把这些数据都存储到默认的设置里面，可能需要单独删除每个条目。而如果为这些设置单独创建一个设置对象的话，只需要在用户退出时简单地调用 `SharedPreferences.Editor.clear()` 即可。

5.2.3 实现机制

让我们看一个实际使用 `SharedPreferences` 保存简单数据的示例。程序清单 5-7 和 5-8 为用户创建了一个数据输入表单，可以用来向远程服务器发送简单消息。为了方便用户，在用户成功发送请求之前，我们会保存用户输入的所有各项数据。这样，用户就可以在暂时离开界面(或者被短信或电话打断)后不必再次输入所有的信息。

程序清单 5-7 res/layout/form.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Email:"
        android:padding="5dip"
    />
    <EditText
        android:id="@+id/email"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
    />
    <CheckBox
        android:id="@+id/age"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```

        android:text="Are You Over 18?"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Message:"
        android:padding="5dip"
    />
    <EditText
        android:id="@+id/message"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:minLines="3"
        android:maxLines="3"
    />
    <Button
        android:id="@+id/submit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Submit"
    />
</LinearLayout>

```

程序清单 5-8 具有持久化功能的输入表单

```

public class FormActivity extends Activity implements View.OnClickListener {

    EditText email, message;
    CheckBox age;
    Button submit;

    SharedPreferences formStore;

    boolean submitSuccess = false;

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

        email = (EditText)findViewById(R.id.email);
        message = (EditText)findViewById(R.id.message);
        age = (CheckBox)findViewById(R.id.age);

        submit = (Button)findViewById(R.id.submit);
        submit.setOnClickListener(this);

        //获取或创建设置对象
        formStore = getPreferences(Activity.MODE_PRIVATE);
    }
}

```

```

@Override
public void onResume() {
    super.onResume();
    //保存表单数据
    email.setText(formStore.getString("email", ""));
    message.setText(formStore.getString("message", ""));
    age.setChecked(formStore.getBoolean("age", false));
}

@Override
public void onPause() {
    super.onPause();
    if(submitSuccess) {
        //接下来可以调用 Editor
        formStore.edit().clear().commit();
    } else {
        //保存表单数据
        SharedPreferences.Editor editor = formStore.edit();
        editor.putString("email", email.getText().toString());
        editor.putString("message", message.getText().toString());
        editor.putBoolean("age", age.isChecked());
        editor.commit();
    }
}

@Override
public void onClick(View v) {

    //发送消息

    //标记操作成功
    submitSuccess = true;
    //关闭
    finish();
}
}

```

首先，我们创建了一个典型的用户表单，它包含有两个简单的 `EditText` 输入框和一个复选框。当创建 `Activity` 时，我们通过 `Activity.getPreferences()` 获得了一个 `SharedPreferences` 对象，所有的持久化对象都会保存到这里。在 `Activity` 被暂停，数据没有发送成功(通过一个布尔值控制)时，表单的当前数据就会保存到设置中。

注意：

在使用 `Editor` 向 `SharedPreferences` 发送数据时，一定要记得在完成修改后调用 `commit()` 或者 `apply()`。否则，你的修改将不会被保存。

相反的，当 `Activity` 可见时，`onResume()` 会将保存在设置对象中的信息加载到用户界面上。如果由于设置被清空或者根本没有被创建，表单就会被设置为空。

当用户按下 **Submit** 后, 信息就会被成功发送, 接下来再调用 `onPause()` 将会清空设置保存的所有表单数据。因为所有这些操作都是在一个私有的设置对象中进行的, 所以清空这些数据并不会影响以其他方式保存的设置。

注意:

`Editor` 中调用的方法, 返回的总是同一个 `Editor` 对象, 这样就能按序将这些操作串在一起, 从而提高代码的可读性。

创建共享 `SharedPreferences`

前面的示例演示了如何在一个单独的 `Activity` 中使用单个的 `SharedPreferences` 对象, 该对象是通过 `Activity.getPreferences()` 得到的。实际上, 这个方法只是 `Context.getSharedPreferences()` 一个为方便使用而实现的包装方法, 用 `Activity` 的名称作为设置保存的名称。如果保存的数据要在两个或两个一起的 `Activity` 中共享的话, 则需要调用 `getSharedPreferences()` 并传入一个通用的名称, 这样就可以在不同的地方访问该 `SharedPreferences` 对象。

程序清单 5-9 使用同一个 `Preferences` 的两个 `Activity`

```
public class ActivityOne extends Activity {
    public static final String PREF_NAME = "myPreferences";
    private SharedPreferences mPreferences;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = getSharedPreferences(PREF_NAME, Activity.MODE_PRIVATE);
    }
}

public class ActivityTwo extends Activity {

    private SharedPreferences mPreferences;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mPreferences = getSharedPreferences(ActivityOne.PREF_NAME,
            Activity.MODE_PRIVATE);
    }
}
```

在这个示例中, 两个 `Activity` 通过同一个名称(定义为字符串常量)得到了 `SharedPreferences` 对象: 这样它们就可以访问同一个数据集。此外, 两个设置甚至指向的是同一个设置对象实例, 这是因为 `Android` 框架为每组 `SharedPreferences`(用其名称所定义)创建的是单例的对象。这意味着一个 `Activity` 所做的修改会立即反应到另一个 `Activity` 中。

关于模式

`Context.getSharedPreferences()`同样会接收一个模式参数。传入 0 或者 `MODE_PRIVATE` 代表默认的行为,只允许创建这个 `SharedPreferences` 的应用程序才能对该 `SharedPreferences` 进行读写。这个方法还支持其他的模式参数: `MODE_WORLD_READABLE` 和 `MODE_WORLD_WRITEABLE`。通过在其创建的文件中设置用户权限,这两个模式允许其他应用程序访问设置。不过,外部应用程序也需要有一个指向创建该设置的包的有效 `Context` 来访问该文件。

例如,在应用程序中以全局可读权限创建一个 `SharedPreferences`,该应用程序的包名为 `com.examples.myfirstapplication`。想要在其他应用程序中访问这个设置,可以使用如下代码:

```
Context otherContext =
    createPackageContext("com.examples.myfirstapplication", 0);
SharedPreferences externalPreferences =
    otherContext.getSharedPreferences(PREF_NAME, 0);
```

警告:

如果选择使用模式参数来允许外部应用程序访问,就必须确保在每次调用 `getSharedPreferences()` 时都使用同样的模式参数。只有在创建设置文件时模式参数才会生效,所以在每次调用 `SharedPreferences` 时使用不同的模式参数只会让你徒增烦恼。

5.3 读写文件

5.3.1 问题

应用程序需要读取外部文件的数据或者保存更加复杂的数据。

5.3.2 解决方案

(API Level 1)

有些时候,必须要使用文件系统。使用文件允许应用程序读写那种不能通过键/值 `preferences` 和数据库方式保存的数据。`Android` 同样提供了文件用的缓存位置,这里可以放一些临时保存的文件数据。

`Android` 支持所有的 Java 文件 I/O API,可以很方便地创建、读取、更新和删除特定位置的文件。应用程序可以在三个主要位置操作文件:

- 内部存储
 - 受保护的用于读写文件数据的目录空间。
- 外部存储
 - 外部挂载的用于读写文件数据的空间。
 - API Level 4 以上需要 `WRITE_EXTERNAL_STORAGE` 权限。

- 通常都是设备的 SD 卡。
- Assets
 - APK 中只读的受保护空间。
 - 用于放置不能/不应该被编译的本地资源。

操作文件数据的基本原理都是一样的，下面看看各种具体情况之间的差异。

5.3.3 实现机制

正如前面所述，传统的 Java `FileInputStream` 和 `FileOutputStream` 类依然是访问文件数据的主要方法。实际上，随时都可以使用文件的绝对路径创建一个 `File` 实例，然后对数据进行读写。不过，不同设备有不同的根路径，有些目录还是应用程序无法访问的，所以建议使用更有效的方式来处理文件。

1. 内部存储

想要在内部存储中创建和修改文件的位置，可以使用 `Context.openFileInput()` 和 `Context.openFileOutput()` 方法。这两个方法只需要文件的名称作为参数，而不是文件的完整路径。它们会引用应用程序受保护目录空间中的文件，并不会关心设备的确切文件目录位置。

程序清单 5-10 在内部存储中 CRUD 文件

```
public class InternalActivity extends Activity {

    private static final String FILENAME = "data.txt";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        setContentView(tv);

        //创建一个新文件并写入一些数据
        try {
            FileOutputStream mOutput = openFileOutput(FILENAME,
                Activity.MODE_PRIVATE);
            String data = "THIS DATA WRITTEN TO A FILE";
            mOutput.write(data.getBytes());
            mOutput.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //读取已经创建的文件并显示在屏幕上
        try {
            FileInputStream mInput = openFileInput(FILENAME);
```

```

        byte[] data = new byte[128];
        mInput.read(data);
        mInput.close();

        String display = new String(data);
        tv.setText(display.trim());
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

//删除创建的文件
deleteFile(FILENAME);
}
}

```

这个示例通过 `Context.openFileOutput()` 向一个文件中写入一些简单的字符串数据。使用该方法时，如果文件不存在，首先会创建该文件。它有两个参数：文件名和操作模式。本例中，我们使用默认的操作模式 `MODE_PRIVATE`。使用这种模式，每次写入时都会覆盖原有的文件。如果想每次都写入文件的尾部，可以使用 `MODE_APPEND`。

写入完成后，本例调用了 `Context.openFileInput()`，该方法也只需要一个文件名称参数，它会打开一个输入流来读取文件数据。数据会被读取到一个 `byte` 数组中并通过 `TextView` 显示给用户。这个操作完成后，会调用 `Context.deleteFile()` 删除存储器中的文件。

注意：

数据是以 `byte` 的形式写入到文件流中的，因此高级的数据(甚至是字符串)必须转换为这种形式。

这个示例在执行完成之后并不会留下任何的文件痕迹，但我们希望可以尝试在结尾处不调用 `deleteFile()`，这样文件就会留在存储器中。使用 SDK 的 DDMS 工具和模拟器或者在已经解锁的设备上查看文件系统，你会找到这个应用程序在它自己的数据文件夹中所创建的文件。

由于它们都是 `Context` 的方法，并不是和 `Activity` 绑定的，因此只要需要的话，可以在应用程序的任何地方使用这种类型的文件访问方式，例如在 `BroadcastReceiver` 或者甚至自定义的类中。很多系统都会构建一个 `Context` 的子类，或是在其回调中传递一个指向某个 `Context` 子类的引用，因此在任何地方都可以执行同样的打开、关闭和删除操作。

2. 外部储存

内部存储器和外部存储器最大的区别就是外部存储器是可挂载的。这意味着用户可以将它们的设备连接到电脑上，然后将设备的外部存储器以移动硬盘的形式挂载到电脑上。通常情况下，外部存储器本身就是可以移除的(如 SD 卡)，但它并不是 Android 所必需的。

重点:

向设备的外部存储器中写入数据需要你在应用程序的 manifest 中添加 android.permission.WRITE_EXTERNAL_STORAGE 权限。

当设备的外部存储器被挂载到其他设备或者被移除时, 应用程序是无法访问它的。因此, 最好经常使用 Environment.getExternalStorageState() 先检查一下外部存储器是否可用。

让我们修改一下上一个文件示例, 使用设备的外部存储器实现相同的操作。

程序清单 5-11 在外部存储器中 CRUD 文件

```
public class ExternalActivity extends Activity {

    private static final String FILENAME = "data.txt";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        setContentView(tv);

        //创建文件的引用
        File dataFile = new File(Environment.getExternalStorageDirectory(),
            FILENAME);

        //检查外部存储器是否可用
        if(!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
            Toast.makeText(this, "Cannot use storage.", Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        //创建一个新文件并写入一些数据
        try {
            FileOutputStream mOutput = new FileOutputStream(dataFile, false);
            String data = "THIS DATA WRITTEN TO A FILE";
            mOutput.write(data.getBytes());
            mOutput.close();
        } catch(FileNotFoundException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }

        //读取已经创建的文件并显示在屏幕上
        try {
            FileInputStream mInput = new FileInputStream(dataFile);
            byte[] data = new byte[128];
            mInput.read(data);
```

```

        mInput.close();

        String display = new String(data);
        tv.setText(display.trim());
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    //删除创建的文件
    dataFile.delete();
}
}

```

对于外部存储器，我们更多地使用了传统的 Java 文件 I/O。使用外部存储器的关键就是需要调用 `Environment.getExternalStorageDirectory()` 来获得设备外部存储器的根目录。

在做任何操作之前，首先通过 `Environment.getExternalStorageState()` 检查设备外部存储器的状态。如果返回的值不是 `Environment.MEDIA_MOUNTED`，这意味着外部存储器不可写入，所以我们什么也不会做，`Activity` 会被关闭。否则，就会创建一个新的文件并执行其他相关操作。

输入流和输出流必须使用 Java 默认的构造函数，而没有使用 `Context` 中的便捷方法。输出流默认会覆盖现有的文件或者在文件不存在时创建一个。如果应用程序每次写入时需要在现有文件的尾部进行添加的话，可以将 `FileOutputStream` 构造函数的 `boolean` 参数设为 `true`。

通常，最好为你的应用程序在外部存储器中创建一个单独的目录。用 Java 的文件 API 就能实现它。参见程序清单 5-12。

程序清单 5-12 在新目录下 CRUD 文件

```

public class ExternalActivity extends Activity {

    private static final String FILENAME = "data.txt";
    private static final String DNAME = "myfiles";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        setContentView(tv);

        //在外部存储器中创建一个新的目录
        File rootPath = new File(Environment.getExternalStorageDirectory(),
            DNAME);
        if(!rootPath.exists()) {
            rootPath.mkdirs();
        }
    }
}

```

```

        //创建文件的引用
        File dataFile = new File(rootPath, FILENAME);

        //创建一个新文件并写入一些数据
        try {
            FileOutputStream mOutput = new FileOutputStream(dataFile, false);
            String data = "THIS DATA WRITTEN TO A FILE";
            mOutput.write(data.getBytes());
            mOutput.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //读取已经创建的文件并显示在屏幕上
        try {
            FileInputStream mInput = new FileInputStream(dataFile);
            byte[] data = new byte[128];
            mInput.read(data);
            mInput.close();

            String display = new String(data);
            tv.setText(display.trim());
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        //删除创建的文件
        dataFile.delete();
    }
}

```

本例中，我们在外部存储器目录中创建了一个新的目录并将这个目录作为数据文件的根目录。当通过新的目录创建好文件的引用后，剩余的代码都是一样的。

3. 外部系统目录

(API Level 8)

Environment 和 Context 的其他方法也能够访问外部存储器上其他一些标准的位置，这些位置可以写入一些特殊的文件。同样，它们会有一些额外的属性。

- Environment.getExternalStoragePublicDirectory(String type)
 - 返回一个所有应用程序保存媒体文件的通用目录。这个目录的内容对用户和其他应用程序都是可见的。特别的，对于 Gallery 这样的应用程序，这里的媒体文件有可能会被扫描并加入到设备的 MediaStore 中。
 - type 参数的类型可以为 DIRECTORY_PICTURES、DIRECTORY_MUSIC、

DIRECTORY_MOVIES 和 DIRECTORY_RINGTONES。

- Context.getExternalFilesDir(String type)
 - 返回外部存储器上媒体文件目录，但只适用于某一个应用程序。这里的媒体文件不是公开的，并不会出现在 MediaStore 中。
 - 由于该目录还是在外部存储器中，所以其他用户和应用程序还是可以看到它并可以直接编辑文件：并不是强制安全的。
 - 放在这里的文件在应用程序卸载时会被删除，因此对于应用程序需要使用大文件同时又不希望保存到内部存储器的场景，这里是绝佳的去处。
 - type 参数的类型可以为 DIRECTORY_PICTURES、DIRECTORY_MUSIC、DIRECTORY_MOVIES 和 DIRECTORY_RINGTONES。
- Context.getExternalCacheDir()
 - 返回一个外部存储器上供某一应用程序临时文件使用的目录。这个目录的内容对用户和其他应用程序都是可见的。
 - 放在这里的文件在应用程序卸载时会被删除，因此对于应用程序需要使用大文件同时又不希望保存到内部存储器的场景，这里是绝佳的去处。

5.4 以资源的形式使用文件

5.4.1 问题

应用程序需要使用不能被 Android 编译为资源 ID 的文件格式。

5.4.2 解决方案

(API Level 1)

应用程序可以在 Assets 目录中保存需要读取的文件，例如本地 HTML 文件、逗号分隔值(CSV)或者专有数据。Assets 目录是应用程序的一个受保护文件目录。这个目录下的文件会和最终的 APK 一起打包并且不会被处理或者编译。和其他的应用程序资源一样，Assets 中的文件都是只读的。

5.4.3 实现机制

我们在本书中已经看了几个关于使用 Assets 直接向 widget(如 WebView、MediaPlayer)中加载内容的示例。此外，在很多情况下，使用传统的 InputStream 也能够很好地访问 Assets。程序清单 5-13 和 5-14 展示了一个示例，它会读取 Assets 中的私有 CSV 文件并显示在屏幕上。

程序清单 5-13 assets/data.csv

```

John,38,Red
Sally,42,Blue
Rudy,31,Yellow

```

程序清单 5-14 读取 Asset 中的文件

```

public class AssetActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        setContentView(tv);

        try {
            //访问应用程序的 assets 目录
            AssetManager manager = getAssets();
            //打开数据文件
            InputStream mInput = manager.open("data.csv");
            //读取数据
            byte[] data = new byte[128];
            mInput.read(data);
            mInput.close();

            //解析 CSV 数据并显示
            String raw = new String(data);
            ArrayList<Person> cooked = parse(raw.trim());
            StringBuilder builder = new StringBuilder();
            for(Person piece : cooked) {
                builder.append(String.format("%s is %s years old, and likes
                    the color %s", piece.name, piece.age, piece.color));
                builder.append('\n');
            }
            tv.setText(builder.toString());

        } catch(FileNotFoundException e) {
            e.printStackTrace();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }

    /*简单 CSV 解析器*/
    private static final int COL_NAME = 0;
    private static final int COL_AGE = 1;
    private static final int COL_COLOR = 2;

    private ArrayList<Person> parse(String raw) {

```

```

        ArrayList<Person> results = new ArrayList<Person>();
        Person current = null;

        StringTokenizer st = new StringTokenizer(raw, ",\n");
        int state = COL_NAME;

        while(st.hasMoreTokens()) {
            switch(state) {
                case COL_NAME:
                    current = new Person();
                    current.name = st.nextToken();
                    state = COL_AGE;
                    break;
                case COL_AGE:
                    current.age = st.nextToken();
                    state = COL_COLOR;
                    break;
                case COL_COLOR:
                    current.color = st.nextToken();
                    results.add(current);
                    state = COL_NAME;
                    break;
            }
        }

        return results;
    }

    private class Person {
        public String name;
        public String age;
        public String color;

        public Person() { }
    }
}

```

访问 Assets 中文件的关键就是使用 `AssetManager`，它允许应用程序打开 Assets 目录中现有的任意资源。在 `AssetManager.open()` 中传入要打开的文件名称会得到一个可以用来读取文件数据的 `InputStream`。当数据流被读取内存后，会将这些原始数据传到一个解析例程中并将解析结果显示在用户界面上。

解析 CSV 数据

这里示例通过一种简单的方式读取一个 CSV 文件中的数据并将该数据解析为一个模型对象(本例中为 `Person`)。这种方式会读取整个文件到一个字节数组中，然后转换为一个单独的字符串进行处理。这种方式在需要读取的数据量很大时，并不是最省内存的，但对于本例中的这种小文件还是比较合适的。

原始字符串数据被传入到一个 `StringTokenizer` 实例中,并且指定了用作分隔符的字符:逗号和换行符。然后,文件中的每个数据块会被按序处理。通过一个基本的状态机方法,就可以将每行数据解析为新的 `Person` 实例并添加到结果列表中。

5.5 管理数据库

5.5.1 问题

应用程序需要查询或者修改保存数据的子集或单条记录。

5.5.2 解决方案

(API Level 1)

可以通过 `SQLiteOpenHelper` 帮助创建一个 `SQLiteDatabase` 并管理数据的存储。`SQLite` 是一个快速和轻量的数据库技术,它使用 `SQL` 语法进行数据的查询和管理。`Android SDK` 本身就支持 `SQLite`,因此在应用程序中设置和使用数据库也就变得非常简单了。

5.5.3 实现机制

通过自定义 `SQLiteOpenHelper` 可以管理数据库的创建和修改。这里还是数据库创建后设置初始和默认值的绝佳场所。程序清单 5-15 的示例展示了如何自定义辅助类来创建一个表单的数据库,该数据库中保存了人员的基本信息。

程序清单 5-15 自定义 `SQLiteOpenHelper`

```
public class MyDbHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "mydb";
    private static final int DB_VERSION = 1;

    public static final String TABLE_NAME = "people";
    public static final String COL_NAME = "pName";
    public static final String COL_DATE = "pDate";

    private static final String STRING_CREATE =
        "CREATE TABLE "+TABLE_NAME+" (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        +COL_NAME+" TEXT, "+COL_DATE+" DATE);";

    public MyDbHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //创建数据库表
    }
}
```

```

        db.execSQL(String.CREATE);

        //可能需要在数据库加载初始值
        ContentValues cv = new ContentValues(2);
        cv.put(COL_NAME, "John Doe");
        //格式化 SQL 日期
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        cv.put(COL_DATE, dateFormat.format(new Date())); //Insert 'now' as the date
        db.insert(TABLE_NAME, null, cv);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //清空并重新创建数据库
        db.execSQL("DROP TABLE IF EXISTS "+TABLE_NAME);
        onCreate(db);
    }
}

```

对于数据库最关键的信息就是数据的名称和版本号。SQLiteDatabase 的创建和更新需要了解一点 SQL 的知识，因此如果还不太熟悉 SQL 语法的话，建议先简单看一下 SQL 的参考资料。如果在数据库不存在时，通过 SQLiteOpenHelper.getReadableDatabase()或者 SQLiteOpenHelper.getWritableDatabase()访问该数据库，就会调用辅助类的 onCreate()方法。

本例中将表名和字段名都抽象为常量，这样就可以供外部使用了(很好的实践方法)。下面是 onCreate()中创建表的实际 SQL 语句：

```

CREATE TABLE people (_id INTEGER PRIMARY KEY AUTOINCREMENT, pName TEXT, pAge
INTEGER, pDate DATE);

```

在 Android 中使用 SQLite 时要遵循一些规范。其中最重要的一点就是，你创建的表中必须有一个 _id 字段。而该 SQL 语句还创建另外两个字段：

- 人员名称的文本字段
- 保存该记录日期的日期字段

数据是以 ContentValues 对象的形式插入到数据库中的。本例演示了如何在数据库创建后通过使用 ContentValues 在数据库中插入一些默认数据。SQLiteDatabase.insert()需要的参数为数据库的表名、空字段填充(nullcolumn hack)、和代表要插入数据的 ContentValues。

这里并没有用到空字段填充，但这个参数对某些应用程序是非常有用的。SQL 不能向数据库中插入全部为空的数据，这样做会导致出错。如果在编写代码时，传递给 insert()的 ContentValues 是空的，那么系统就会将空字段插入数据库，每个字段的内容都是 NULL。

1. 关于升级

SQLiteOpenHelper 还能在升级应用程序的过程中帮助迁移数据库架构。访问数据库时，如果设备上的数据库版本和当前版本(即构造函数中传入的版本号)不一致，onUpgrade()就会被调用。

在我们的示例中，我们采用了一种偷懒的方式只是简单地删除了现有的数据库并重新

创建它。实际上，数据库存放了用户输入的数据，这种方式就有点不合适了；用户可能并不希望他的数据会消失掉。所以，我们来看一个更加实用的 `onUpgrade()` 的示例。应用程序在其生命周期期间会使用下面三个版本的数据库：

- 版本 1：应用程序的第一版
- 版本 2：升级后的应用程序加入了电话号码字段
- 版本 3：升级后的应用程序加入了日期字段

我们可以使用 `onUpgrade()` 来修改现有的数据库，而不是将现有的数据库信息全部删除。参见程序清单 5-16。

程序清单 5-16 `onUpgrade()` 示例

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    //在 v1 的基础进行升级。添加电话号码字段
    if(oldVersion <= 1) {
        db.execSQL("ALTER TABLE "+TABLE_NAME+" ADD COLUMN phone_number INTEGER;");
    }
    //在 v2 的基础上进行升级。添加日期字段
    if(oldVersion <= 2) {
        db.execSQL("ALTER TABLE "+TABLE_NAME+" ADD COLUMN entry_date DATE;");
    }
}
```

在这个示例中，如果用户现在的数据库版本是 1，两个语句都会执行，数据库会增加两个字段。如果数据库版本已经是 2，则只会执行后一个语句，即添加日期字段。两种情况下，都会保留应用程序数据库中现有的数据。

2. 使用数据库

回到原来的示例，看一下 `Activity` 如何使用我们创建的数据库。参见程序清单 5-17 和 5-18。

程序清单 5-17 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
    <Button
        android:id="@+id/add"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```

        android:text="Add New Person"
    />
<ListView
    android:id="@+id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
</LinearLayout>

```

程序清单 5-18 查看和管理数据库的 Activity

```

public class DbActivity extends Activity implements View.OnClickListener,
    AdapterView.OnItemClickListener {

    EditText mText;
    Button mAdd;
    ListView mList;

    MyDbHelper mHelper;
    SQLiteDatabase mDb;
    Cursor mCursor;
    SimpleCursorAdapter mAdapter;

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

        mText = (EditText)findViewById(R.id.name);
        mAdd = (Button)findViewById(R.id.add);
        mAdd.setOnClickListener(this);
        mList = (ListView)findViewById(R.id.list);
        mList.setOnItemClickListener(this);

        mHelper = new MyDbHelper(this);
    }

    @Override
    public void onResume() {
        super.onResume();
        //建立和数据库的连接
        mDb = mHelper.getWritableDatabase();
        String[] columns = new String[] {"_id", MyDbHelper.COL_NAME,
            MyDbHelper.COL_DATE};
        mCursor = mDb.query(MyDbHelper.TABLE_NAME, columns, null, null,
            null, null, null);
        //更新列表
        String[] headers = new String[] {MyDbHelper.COL_NAME, MyDbHelper.COL_DATE};
        mAdapter = new SimpleCursorAdapter(this, android.R.layout.two_line_list_item,
            mCursor, headers, new int[]{android.R.id.text1,
            android.R.id.text2});
    }
}

```

```

        mList.setAdapter(mAdapter);
    }

    @Override
    public void onPause() {
        super.onPause();
        //关闭连接
        mDb.close();
        mCursor.close();
    }

    @Override
    public void onClick(View v) {
        //向数据库中添加新数据
        ContentValues cv = new ContentValues(2);
        cv.put(MyDbHelper.COL_NAME, mText.getText().toString());
        //格式化 SQL 日期
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
        //插入当前日期数据
        cv.put(MyDbHelper.COL_DATE, dateFormat.format(new Date()));
        mDb.insert(MyDbHelper.TABLE_NAME, null, cv);
        //更新列表
        mCursor.requery();
        mAdapter.notifyDataSetChanged();
        //清空编辑框
        mText.setText(null);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position,
long id) {
        //删除数据库中的条目
        mCursor.moveToPosition(position);
        //获得该行的 id
        String rowId = mCursor.getString(0); //Column 0 of the cursor is the id
        mDb.delete(MyDbHelper.TABLE_NAME, "_id = ?", new String[]{rowId});
        //更新列表
        mCursor.requery();
        mAdapter.notifyDataSetChanged();
    }
}

```

在这个示例中，我们通过自定义的 `SQLiteOpenHelper` 访问一个数据库，并将该数据库中的记录以列表的形式显示在用户界面上。数据库中的信息是以 `Cursor` 的形式返回的，而 `Cursor` 是一个可以用来读、写、遍历查询结果的接口。

当 `Activity` 可见时，会查询数据库中“people”表并返回该表中的所有记录。在此过程中，需要传入一个标识需要返回字段的数组。`query()` 的其他参数是用来缩小数据集的范围的，会在下一个范例中进行讨论。在数据库和游标连接不用时，一定要关闭它们。本例中，

则是在 onPause() 中进行关闭操作，这时候 Activity 已经不在前台了。

SimpleCursorAdapter 用来将数据库中的数据映射为标准的 Android 两行式的列表 view。映射时的参数为一个字符串数组和一个整型数组；系统会将字符串数组中的每一项数据和整型数组中相应的 id 值插入到 view 中。注意，这里传递的字段名称与传递给查询的数组是不太一样的。因为在其他操作中需要用到每条数据的 id，但在将数据映射到用户界面时是不需要这个 id 的。

用户可能会在文本框中输入名称并单击“Add New Person”按钮来创建一个新的 ContentValues 并插入到数据库中。如果是这样的话，为了让 UI 能够显示数据库的变化，我们调用了 Cursor.requery() 和 ListAdapter.notifyDataSetChanged() 方法。

相反，单击列表中的条目会删除数据库中相应的记录。要实现这个功能，我们构造了一个简单的 SQL 语句，告诉数据库删除与选择的 _id 相对应的记录。然后，游标和列表的 adapter 都会再次刷新。

将游标移动到选择的位置，调用 getString(0) 就能获取到 0 字段的值。因为所查询的第一个参数(索引是 0)就是 _id，所以这样得到的就是 _id。删除语句由两个参数组成：SQL 语句和参数。参数是以数组的形式传递的，其中的每个元素会按序插入到 SQL 语句字符串中各个问号所在的位置。

5.6 查询数据库

5.6.1 问题

应用程序使用了 SQLiteDatabase，需要从中获取所需的数据子集。

5.6.2 解决方案

(API Level 1)

利用完全结构化的 SQL 查询，可以很方便地创建各种数据过滤器，从数据库中获取所需的数据子集。SQLiteDatabase.query() 有好几种重载形式可以从数据库获取信息。下面看看其中最复杂的一种形式。

```
public Cursor query(String table, String[] columns, String selection, String[]
selectionArgs, String groupBy, String having, String orderBy, String limit)
```

前两个参数只是定义了查询数据的数据库表 and 要访问的列。其他参数则是定义了如何缩小结果的范围。

- selection
 - 查询的 SQL WHERE 子句。
- selectionArgs
 - 如果 selection 中有问号，就用这里的元素逐个填充那些问号位置。
- groupBy

- 查询的 SQL GROUP BY 子句。
- having
 - 查询的 SQL GROUP BY 子句。
- orderBy
 - 查询的 SQL ORDER BY 子句。
- limit
 - 查询返回结果的数量上限。

可以看到，所有的这些参数都是为了提供完整的 SQL 数据查询功能。

5.6.3 实现机制

让我们看一些能完成常见实际查询的示例。

- 返回能匹配给定参数的所有行。

```
String[] COLUMNS = new String[] {COL_NAME, COL_DATE};
String selection = COL_NAME+" = ?";
String[] args = new String[] {"NAME_TO_MATCH"};
Cursor result = db.query(TABLE_NAME, COLUMNS, selection, args, null, null,
null, null);
```

这个查询很直观。`selection` 语句告诉数据库根据参数所提供的字段名称(插入到 `selection` 字符串中间号的位置)返回数据。

- 返回最近插入数据库的 10 行记录。

```
String orderBy = "_id DESC";
String limit = "10";
Cursor result = db.query(TABLE_NAME, COLUMNS, null, null, null, null, orderBy,
limit);
```

这个查询没有指定 `selection` 指标，而是要数据库将结果以 `_id` 值按增序排列，首先返回的是最新的(`_id` 最大的)记录。`limit` 子句将返回结果的数量上限设为 10。

- 返回 `date` 字段在指定范围内的行(本例中是 2000 年)。

```
String[] COLUMNS = new String[] {COL_NAME, COL_DATE};
String selection = "datetime("+COL_DATE+") > datetime(?)"+
" AND datetime("+COL_DATE+") < datetime(?)";
String[] args = new String[] {"2000-1-1 00:00:00", "2000-12-31 23:59:59"};
Cursor result = db.query(TABLE_NAME, COLUMNS, selection, args, null, null,
null, null);
```

在 SQLite 中定义表时可以声明 `DATE` 类型，但实际上 SQLite 并没有专门的日期类型数据。可以用标准的 SQL 的日期和时间函数以 `TEXT`、`INTEGER` 或 `REAL` 的形式表示日期和时间。这里我们将数据库中的值和已经格式化好的时期起始时间字符串传递给 `datetime()`，然后比较返回的值。

- 返回整型字段在指定范围内的行(本例中是 7~10)。

```
String[] COLUMNS = new String[] {COL_NAME, COL_AGE};
String selection = COL_AGE+" > ? AND "+COL_AGE+" < ?";
String[] args = new String[] {"7","10"};
Cursor result = db.query(TABLE_NAME, COLUMNS, selection, args, null, null,
null, null);
```

这与前一个示例类似，但更简单。它只创建了一个返回值有上下限的 `selection` 语句。两个限制都是以参数的形式插入的，所以可以在应用程序中动态设置。

5.7 备份数据

5.7.1 问题

应用程序将数据保存到设备上，但在用户更换设备或者被迫重装应用程序时，需要为用户提供一种方式来备份和恢复这些数据。

5.7.2 解决方案

(API Level 1)

设备的外部存储器可以安全地保存数据库和其他文件。外部存储器一般都是可拆卸的，可以从一个设备移到另一个设备上进行数据恢复。即使不可拆卸，也可以将外部存储器挂载到计算机上，从而实现数据传输。

5.7.3 实现机制

程序清单 5-19 实现了一个 `AsyncTask`，它可以在设备的外部存储器和应用程序数据目录之间进行数据库文件的复制。它还定义了一个 `Activity` 可以实现的接口，用于在操作完成后通知 `Activity`。像复制这样的文件操作是需要一些时间的，因此通过 `AsyncTask` 可以让该操作在后台执行从而不会阻塞主线程。

程序清单 5-19 用来备份和恢复的 `AsyncTask`

```
public class BackupTask extends AsyncTask<String,Void,Integer> {

    public interface CompletionListener {
        void onBackupComplete();
        void onRestoreComplete();
        void onError(int errorCode);
    }

    public static final int BACKUP_SUCCESS = 1;
    public static final int RESTORE_SUCCESS = 2;
    public static final int BACKUP_ERROR = 3;
    public static final int RESTORE_NOFILEERROR = 4;
```

```

public static final String COMMAND_BACKUP = "backupDatabase";
public static final String COMMAND_RESTORE = "restoreDatabase";

private Context mContext;
private CompletionListener listener;

public BackupTask(Context context) {
    super();
    mContext = context;
}

public void setCompletionListener(CompletionListener aListener) {
    listener = aListener;
}

@Override
protected Integer doInBackground(String... params) {

    //获得数据库的引用
    File dbFile = mContext.getDatabasePath("mydb");
    //获得备份的目录位置
    File exportDir =new File
        (Environment.getExternalStorageDirectory(), "myAppBackups");
    if (!exportDir.exists()) {
        exportDir.mkdirs();
    }
    File backup = new File(exportDir, dbFile.getName());

    //检查需要的操作
    String command = params[0];
    if(command.equals(COMMAND_BACKUP)) {
        //尝试复制文件
        try {
            backup.createNewFile();
            fileCopy(dbFile, backup);

            return BACKUP_SUCCESS;
        } catch (IOException e) {
            return BACKUP_ERROR;
        }
    } else if(command.equals(COMMAND_RESTORE)) {
        //尝试复制文件
        try {
            if(!backup.exists()) {
                return RESTORE_NOFILEERROR;
            }
            dbFile.createNewFile();
            fileCopy(backup, dbFile);
            return RESTORE_SUCCESS;
        } catch (IOException e) {

```

```

        return BACKUP_ERROR;
    }
} else {
    return BACKUP_ERROR;
}
}

@Override
protected void onPostExecute(Integer result) {

    switch(result) {
    case BACKUP_SUCCESS:
        if(listener != null) {
            listener.onBackupComplete();
        }
        break;
    case RESTORE_SUCCESS:
        if(listener != null) {
            listener.onRestoreComplete();
        }
        break;
    case RESTORE_NOFILEERROR:
        if(listener != null) {
            listener.onError(RESTORE_NOFILEERROR);
        }
        break;
    default:
        if(listener != null) {
            listener.onError(BACKUP_ERROR);
        }
    }
}

private void fileCopy(File source, File dest) throws IOException {
    FileChannel inChannel = new FileInputStream(source).getChannel();
    FileChannel outChannel = new FileOutputStream(dest).getChannel();
    try {
        inChannel.transferTo(0, inChannel.size(), outChannel);
    } finally {
        if(inChannel != null)
            inChannel.close();
        if(outChannel != null)
            outChannel.close();
    }
}
}

```

从上面的代码可以看出，当在 BackupTask 的 execute() 中传入 COMMAND_BACKUP 时，会将特定名称的数据的当前版本复制到外部存储器上的特定目录下，而当传入 COMMAND_RESTORE 则执行相反的复制动作。

执行备份时, BackupTask 通过 Context.getDatabasePath()得到需要备份的数据库文件的引用。这行代码也可以修改为 Context.getFilesDir(), 这样就可以备份系统内部存储器中的文件。同样, 我们还获得了外部存储器上备份文件夹的引用。

文件的复制使用的是传统的 Java 文件 I/O, 复制成功会通知注册的监听器。在复制过程中, 如果出现异常, 也会通知监听器。现在, 让我看一下使用这个 task 来进行数据库备份的 Activity: 参见程序清单 5-20。

程序清单 5-30 使用 BackupTask 的 Activity

```
public class BackupActivity extends Activity implements
BackupTask.CompletionListener {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //伪示例数据库
        SQLiteDatabase db = openOrCreateDatabase("mydb",
            Activity.MODE_PRIVATE, null);
        db.close();
    }

    @Override
    public void onResume() {
        super.onResume();
        if( Environment.getExternalStorageState().equals(Environment.
            MEDIA_MOUNTED) ) {
            BackupTask task = new BackupTask(this);
            task.setCompletionListener(this);
            task.execute(BackupTask.COMMAND_RESTORE);
        }
    }

    @Override
    public void onPause() {
        super.onPause();
        if( Environment.getExternalStorageState().equals(Environment.
            MEDIA_MOUNTED) ) {
            BackupTask task = new BackupTask(this);
            task.execute(BackupTask.COMMAND_BACKUP);
        }
    }

    @Override
    public void onBackupComplete() {
        Toast.makeText(this, "Backup Successful", Toast.LENGTH_SHORT).show();
    }

    @Override
```

```

public void onError(int errorCode) {
    if(errorCode == BackupTask.RESTORE_NOFILEERROR) {
        Toast.makeText(this, "No Backup Found to Restore",
            Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Error During Operation: "+errorCode,
            Toast.LENGTH_SHORT).show();
    }
}

@Override
public void onRestoreComplete() {
    Toast.makeText(this, "Restore Successful", Toast.LENGTH_SHORT).show();
}
}

```

这个 Activity 实现了 BackupTask 定义的 CompletionListener, 因此在复制完成或者发生错误时都会得到通知。为了演示这个示例, 在应用程序的数据库目录中创建了一个伪数据库。调用 openOrCreateDatabase() 只是为了创建数据库文件, 所以在创建数据库之后会立即关闭这个连接。一般情况下, 这个数据库应该已经存在了, 这几行代码也可以省掉。

这个示例在每次 Activity resume 时, 会执行恢复操作, 同时会将自己注册到 task 中从而在操作状态发生变化时可以弹出 toast 来通知用户。注意, 对于外部存储器是否可用的检查也放到了 Activity 中, 在外部存储器不可用时, 不会执行任何 task。在 Activity 暂停时, 会执行备份操作, 这里并没有注册回调。这是因为一旦暂停了 Activity, 意味着用户对 Activity 已经不再感兴趣, 因此也就没有必要再将结果通知给用户了。

补充信息

这个后台任务可以扩展一下, 将数据保存到云端服务器上, 从而实现安全性的最大化和数据的可移动性。要实现这个功能有很多种方式, 包括 Google 自己的一些 Web API, 建议可以试一下。

Android在API Level 8中加入了将数据备份到云端服务器的API。这个API或许能满足你的要求, 这里就不再赘述了。Android框架并不能保证在所有的Android设备上这个服务都是可用的, 而且在编写本书时也没有API可以判断用户的设备是否能够支持Android备份, 因此不建议使用它进行重要数据的备份。

5.8 分享数据库

5.8.1 问题

应用程序需要将它维护的数据库内容提供给设备上的其他应用程序使用。

5.8.2 解决方案

(API Level 4)

创建一个 `ContentProvider` 来作为应用程序数据的外部接口。`ContentProvider` 可以通过和数据库接口类似的 `query()`、`insert()`、`update()` 和 `delete()` 方法将特定的数据集暴露给其他应用程序。接口和数据模型之间的映射关系可以灵活定制。通过 `ContentProvider` 对外提供 `SQLiteDatabase` 的数据是很简单的。多数情况下，开发者只需要将 `provider` 的调用传递给数据库即可。

要操作的数据集通常会编码为 `Uri`，然后传递给 `ContentProvider`。例如，发送 `content://com.examples.myprovider/friends` 这样的查询 `Uri` 就是告诉 `provider` 返回“friends”表中的数据，而 `content://com.examples.myprovider/friends/15` 则意味着在查询结果中返回 `id` 为 15 的记录。注意，这些是系统中其他应用程序获取数据的规范，而你必须确保你所创建的 `ContentProvider` 符合这个规范。`ContentProvider` 本身并没有提供这种功能。

5.8.3 实现机制

首先，想要创建一个能够和数据库进行交互的 `ContentProvider`，必须先要有一个数据库。程序清单 5-21 为一个示例 `SQLiteOpenHelper`，它可以创建一个数据库并访问该数据库。

程序清单 5-21 示例 `SQLiteOpenHelper`

```
public class ShareDbHelper extends SQLiteOpenHelper {

    private static final String DB_NAME = "friendddb";
    private static final int DB_VERSION = 1;

    public static final String TABLE_NAME = "friends";
    public static final String COL_FIRST = "firstName";
    public static final String COL_LAST = "lastName";
    public static final String COL_PHONE = "phoneNumber";

    private static final String STRING_CREATE =
        "CREATE TABLE "+TABLE_NAME+" (_id INTEGER PRIMARY KEY AUTOINCREMENT, "
        +COL_FIRST+" TEXT, "+COL_LAST+" TEXT, "+COL_PHONE+" TEXT);";

    public ShareDbHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //创建数据库表
        db.execSQL(STRING_CREATE);

        //向数据库中插入示例值
        ContentValues cv = new ContentValues(3);
```

```

        cv.put(COL_FIRST, "John");
        cv.put(COL_LAST, "Doe");
        cv.put(COL_PHONE, "8885551234");
        db.insert(TABLE_NAME, null, cv);
        cv = new ContentValues(3);
        cv.put(COL_FIRST, "Jane");
        cv.put(COL_LAST, "Doe");
        cv.put(COL_PHONE, "8885552345");
        db.insert(TABLE_NAME, null, cv);
        cv = new ContentValues(3);
        cv.put(COL_FIRST, "Jill");
        cv.put(COL_LAST, "Doe");
        cv.put(COL_PHONE, "8885553456");
        db.insert(TABLE_NAME, null, cv);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //清空并重新创建数据库
        db.execSQL("DROP TABLE IF EXISTS "+TABLE_NAME);
        onCreate(db);
    }
}

```

这个 helper 非常简单，它创建了一个表来保存好友信息列表，好友信息由三个字段组成，每个字段都是文本数据。为了演示这个示例，我们已经插入了三组数据。现在，让我看一下，ContentProvider 是如何将这个数据库暴露给其他应用程序的：程序清单 5-22 和 5-23。

程序清单 5-22 在 Manifest 中的声明 ContentProvider

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
    <application ...>
        <provider android:name=".FriendProvider"
            android:authorities="com.examples.sharedb.friendprovider">
        </provider>
    </application>
</manifest>

```

程序清单 5-23 数据库的 ContentProvider

```

public class FriendProvider extends ContentProvider {

    public static final Uri CONTENT_URI =
        Uri.parse("content://com.examples.sharedb.friendprovider/friends");

    public static final class Columns {
        public static final String _ID = "_id";
        public static final String FIRST = "firstName";
        public static final String LAST = "lastName";
        public static final String PHONE = "phoneNumber";
    }
}

```

```

    }

    /* 匹配 Uri */
    private static final int FRIEND = 1;
    private static final int FRIEND_ID = 2;

    private static final UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
    static {
        matcher.addURI(CONTENT_URI.getAuthority(), "friends", FRIEND);
        matcher.addURI(CONTENT_URI.getAuthority(), "friends/#", FRIEND_ID);
    }

    SQLiteDatabase db;

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int result = matcher.match(uri);
        switch(result) {
            case FRIEND:
                return db.delete(ShareDbHelper.TABLE_NAME, selection, selectionArgs);
            case FRIEND_ID:
                return db.delete(ShareDbHelper.TABLE_NAME, "_ID = ?",
                    new String[]{uri.getLastPathSegment()});
            default:
                return 0;
        }
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        long id = db.insert(ShareDbHelper.TABLE_NAME, null, values);
        if(id >= 0) {
            return Uri.withAppendedPath(uri, String.valueOf(id));
        } else {
            return null;
        }
    }

    @Override
    public boolean onCreate() {
        ShareDbHelper helper = new ShareDbHelper(getContext());
        db = helper.getWritableDatabase();
        return true;
    }

```

```

@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String sortOrder) {
    int result = matcher.match(uri);
    switch(result) {
        case FRIEND:
            return db.query(ShareDbHelper.TABLE_NAME, projection, selection,
                selectionArgs, null, null, sortOrder);
        case FRIEND_ID:
            return db.query(ShareDbHelper.TABLE_NAME, projection, "_ID = ?",
                new String[]{uri.getLastPathSegment()}, null, null, sortOrder);
        default:
            return null;
    }
}

@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
    int result = matcher.match(uri);
    switch(result) {
        case FRIEND:
            return db.update(ShareDbHelper.TABLE_NAME, values, selection,
                selectionArgs);
        case FRIEND_ID:
            return db.update(ShareDbHelper.TABLE_NAME, values, "_ID = ?",
                new String[]{uri.getLastPathSegment()});
        default:
            return 0;
    }
}
}

```

必须要在应用程序的 manifest 中声明 ContentProvider 及其 authority 字符串。这样外部应用程序才可以访问该 provider，而且即使在应用程序内部使用，也需要在 manifest 中声明。Android 是通过 authority 来匹配 Uri 请求和 provider 的，所以 authority 必须要与公开的 CONTENT_URI 的 authority 部分保持一致。

在继承 ContentProvider 后，需要覆写六个方法 query()、insert()、update()、delete()、getType()和 onCreate()。前 4 个方法在 SQLiteDatabase 中都有对应的方法，所以只需要传入相应参数调用数据库的方法即可。它们最大的区别就是 ContentProvider 的方法需要传入一个 Uri，provider 会根据这个 Uri 判断该操作数据库的哪个子集。

当 Activity 或者其他系统控件调用其内部的 ContentResolver 的相应方法时(参见程序清单 5-23)，就会调用这 4 个主要的 CRUD 方法。

为了符合本节开头提及的 Uri 规范，insert()会将新创建的记录 id 添加到路径的末尾，并将其作为 Uri 对象返回。调用 insert()的代码就可以通过这个 Uri 对象得到所创建的数据记录的直接引用。

其他的方法(query()、update()和 delete())也会检查收到的 Uri 指向的是某个记录还是整个表。这个工作是在 UriMatcher 这个便捷类的帮助下完成的。UriMatcher.match()会将 Uri 与设定好的模式进行比较,然后以整型的形式返回匹配的模式,如果没有找到能匹配的模式,会返回 UriMatcher.NO_MATCH。如果一个 Uri 包含了 id,对数据库的调用就只会影响到指定 id 的那一行。

UriMatcher 应该通过 UriMatcher.addURI()设置一个模式来进行初始化,Google 建议这个工作最好在 ContentProvider 内部的静态块中完成,这样在类刚加载到内存时 UriMatcher 就会被初始化。每个加入的模式都包含一个对应的常数编号,当该模式匹配时,就会返回这个编号。在模式中可以使用两个通配符:用来匹配数字的井号(#)和用来匹配任意文本的星号(*)。

我们的示例创建了两种匹配模式。第一种要匹配的模式就是 CONTENT_URI,指向这个数据表。第二种模式是在第一种模式的路径后面加上了一个 id,指向 id 所对应的记录。

访问数据库的引用是在 ShareDbHelper 的 onCreate()中获得的。在使用这种方式时,需要考虑数据的大小是否合适。我们的数据库创建时是比较小的,但大一些的数据库的创建过程可能会花费很长时间,这时候就不应该在主线程执行该操作了,最好将 getWritableDatabase()封装到一个 AsyncTask 中在后台执行。现在,让我看一下访问数据的示例 Activity: 参见程序清单 5-24 和 5-25。

程序清单 5-24 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.sharedb" android:versionCode="1" android:versionName="1.0">
    <uses-sdk android:minSdkVersion="4" />
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ShareActivity" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name=".FriendProvider"
            android:authorities="com.examples.sharedb.friendprovider">
        </provider>
    </application>
</manifest>
```

程序清单 5-25 访问 ContentProvider 的 Activity

```
public class ShareActivity extends FragmentActivity implements
    LoaderManager.LoaderCallbacks<Cursor>, AdapterView.OnItemClickListener {
    private static final int LOADER_LIST = 100;
    SimpleCursorAdapter mAdapter;

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

```

super.onCreate(savedInstanceState);
getSupportLoaderManager().initLoader(LOADER_LIST, null, this);

mAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1, null,
    new String[]{FriendProvider.Columns.FIRST},
    new int[]{android.R.id.text1}, 0);

ListView list = new ListView(this);
list.setOnItemClickListener(this);
list.setAdapter(mAdapter);

setContentView(list);
}

@Override
public void onItemClick(AdapterView<?> parent, View v, int position,
long id) {
    Cursor c = mAdapter.getCursor();
    c.moveToPosition(position);

    Uri uri = Uri.withAppendedPath(FriendProvider.CONTENT_URI,
        c.getString(0));
    String[] projection = new String[]{FriendProvider.Columns.FIRST,
        FriendProvider.Columns.LAST,
        FriendProvider.Columns.PHONE};
    //得到全部记录
    Cursor cursor = getContentResolver().query(uri, projection, null,
        null, null);
    cursor.moveToFirst();

    String message = String.format("%s %s, %s", cursor.getString(0),
        cursor.getString(1), cursor.getString(2));
    Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    cursor.close();
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String[] projection = new String[]{FriendProvider.Columns._ID,
        FriendProvider.Columns.FIRST};
    return new CursorLoader(this, FriendProvider.CONTENT_URI,
        projection, null, null, null);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    mAdapter.swapCursor(data);
}

```

```

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}
}

```

重点:

这个示例在 Android 1.6 及以上版本中需要使用支持库才能够访问 Loader。如果应用程序的目标平台为 Android 3.0+, 可以将 `FragmentActivity` 替换为 `Activity`, `getSupportLoaderManager()` 替换为 `getLoaderManager()`。

这个示例会查询 `FriendsProvider` 中的所有记录, 把它们放到一个列表中并只显示名称字段。为了让 `Cursor` 能够正确地匹配列表, 即使我们不显示 ID 字段, 在也必须在映射中包含 ID 字段。

如果用户单击了列表中的条目, 就会将记录的 ID 添加到路径的结尾从而构建出一个 `Uri` 并再次进行查询, 要求 `provider` 返回指定的记录。另外, 这里还有一个扩展后的映射可以获得指定好友的所有数据。

返回的数据会以 `Toast` 的形式显示给用户。通过游标可以返回每个字段的索引, 也就是传递给查询的映射中的索引, 并访问其中的数据。`Cursor.getColumnIndex()` 还可以通过游标查询指定字段名的索引。

正如我们在每次用户单击事件中所做的那样, `Cursor` 在不用时应该关闭。唯一的例外就是通过 `Loader` 创建和管理的 `Cursor`, 它不需要这样做。

图 5-2 显示了运行该示例后的结果, 会显示 `provider` 的内容。



图 5-2 来自 `ContentProvider` 的信息

5.9 分享 `SharedPreferences`

5.9.1 问题

需要将存储在 `SharedPreferences` 中的设置值提供给系统的其他应用程序使用, 在权限允许的情况下, 甚至允许其他应用程序修改设置值。

5.9.2 解决方案

(API Level 1)

创建一个 `ContentProvider` 来作为应用程序 `SharedPreferences` 与其他应用程序的接口。

设置数据将通过 `MatrixCursor` 传递，它可以用来处理非数据库中的数据。对 `ContentProvider` 中的数据读/写是需要单独权限的，只有拥有权限的应用程序才可以访问。

5.9.3 实现机制

为了更好地演示本范例中权限方面的问题，我们需要创建两个单独的应用程序：一个应用程序包含了我们的设置数据而另一个应用程序会通过 `ContentProvider` 读取并修改该数据。这是因为同一个 `Android` 应用程序中的操作是不需要权限的。让我首先看一下 `provider`，参见程序清单 5-26。

程序清单 5-26 应用程序设置的 `ContentProvider`

```
public class SettingsProvider extends ContentProvider {

    public static final Uri CONTENT_URI =
        Uri.parse("content://com.examples.sharepreferences.
            settingsprovider/settings");

    public static class Columns {
        public static final String _ID = Settings.NameValueTable._ID;
        public static final String NAME = Settings.NameValueTable.NAME;
        public static final String VALUE = Settings.NameValueTable.VALUE;
    }

    private static final String NAME_SELECTION = Columns.NAME + " = ?";

    private SharedPreferences mPreferences;

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        throw new UnsupportedOperationException(
            "This ContentProvider is does not support removing Preferences");
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        throw new UnsupportedOperationException(
            "This ContentProvider is does not support adding new Preferences");
    }

    @Override
    public boolean onCreate() {
        mPreferences = PreferenceManager.getDefaultSharedPreferences(getContext());
    }
}
```

```

        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
        MatrixCursor cursor = new MatrixCursor(projection);
        Map<String, ?> preferences = mPreferences.getAll();
        Set<String> preferenceKeys = preferences.keySet();

        if(TextUtils.isEmpty(selection)) {
            //获取所有条目
            for(String key : preferenceKeys) {
                //插入需要的字段
                MatrixCursor.RowBuilder builder = cursor.newRow();
                for(String column : projection) {
                    if(column.equals(Columns._ID)) {
                        //生成唯一的 id
                        builder.add(key.hashCode());
                    }
                    if(column.equals(Columns.NAME)) {
                        builder.add(key);
                    }
                    if(column.equals(Columns.VALUE)) {
                        builder.add(preferences.get(key));
                    }
                }
            }
        } else if (selection.equals(NAME_SELECTION)) {
            //解析 key 值并检查它是否存在
            String key = selectionArgs == null ? "" : selectionArgs[0];
            if(preferences.containsKey(key)) {
                //得到需要的条目
                MatrixCursor.RowBuilder builder = cursor.newRow();
                for(String column : projection) {
                    if(column.equals(Columns._ID)) {
                        //生成一个唯一 id
                        builder.add(key.hashCode());
                    }
                    if(column.equals(Columns.NAME)) {
                        builder.add(key);
                    }
                    if(column.equals(Columns.VALUE)) {
                        builder.add(preferences.get(key));
                    }
                }
            }
        }

        return cursor;
    }

```

```

    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
        String[] selectionArgs) {
        //检查key是否存在, 并更新它的值
        String key = values.getAsString(Columns.NAME);
        if (mPreferences.contains(key)) {
            Object value = values.get(Columns.VALUE);
            SharedPreferences.Editor editor = mPreferences.edit();
            if (value instanceof Boolean) {
                editor.putBoolean(key, (Boolean)value);
            } else if (value instanceof Number) {
                editor.putFloat(key, ((Number)value).floatValue());
            } else if (value instanceof String) {
                editor.putString(key, (String)value);
            } else {
                //无效的值, 不更新
                return 0;
            }
            editor.commit();
            //通知观察者
            getContext().getContentResolver().notifyChange(CONTENT_URI, null);
            return 1;
        }
        //key 不在设置中
        return 0;
    }
}

```

创建这个ContentProvider时, 我们得到了应用程序默认SharedPreferences的引用而不是像之前的示例一样打开一个数据库连接。这个provider只支持两个方法query()和update(), 其他方法都会抛出异常。这样就只允许对设置值进行读/写操作而不允许添加或者移除新的设置类型。

在 query()方法中, 我们会检查查询语句从而确定返回全部的设置值还是某个需要的值。每个设置都定义了三个字段: _id、name 和 value。_id 也许和设置本身关系不大, 但如果使用 provider 的用户想通过 CursorAdapter 显示结果列表, 则需要这个字段并且每个记录的值不能相同, 所以我们就定义了它。注意, 设置值是以对象的方式插入到游标中的, 这样就可以减少 provider 所包含的数据类型。

这个 provider 中所使用的游标为 MatrixCursor, 它是用来处理非数据库中的数据。这个示例会迭代需要的字段列表(映射)并根据所包含的字段构建每行数据。每行都是通过 MatrixCursor.newRow()方法创建的, 该方法同时会返回一个可以用来添加字段数据的 Builder 实例。注意, 添加的字段数据的顺序和映射的顺序应该是一致的。

update()方法只会检查传入 ContentValues 中需要更新的设置值。因为这已足以描述我们需要的选项, 所以我们并没有使用选择参数实现其他进一步的逻辑。如果设置的名字存在, 就会更新和保存它的值。不幸的是, 并没有方法可以简单地将一个对象插入到

SharedPreferences, 所以必须检查该值是否是 ContentValues 可以返回的有效类型, 然后调用相应的设置方法。最后, 我们调用了 notifyObservers() 通知所有注册的 ContentObserver 对象数据已经发生改变。

可能你会发现, 在 ContentProvider 中并没有我们之前承诺的关于管理读/写权限的代码! 实际上, 这是由 Android 来替我们管理的: 我们只需要适当修改 manifest 文件即可。参见程序清单 5-27。

程序清单 5-27 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.sharepreferences"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk ... />

    <permission
        android:name="com.examples.sharepreferences.permission.
            READ_PREFERENCES"
        android:label="Read Application Settings"
        android:protectionLevel="normal" />
    <permission
        android:name="com.examples.sharepreferences.permission.
            WRITE_PREFERENCES"
        android:label="Write Application Settings"
        android:protectionLevel="dangerous" />
    <uses-permission
        android:name="com.examples.sharepreferences.permission.
            READ_PREFERENCES" />
    <uses-permission
        android:name="com.examples.sharepreferences.permission.
            WRITE_PREFERENCES" />

    <application ... >
        <activity android:name=".SettingsActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
                    android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action
                    android:name="com.examples.sharepreferences.ACTION_SETTINGS"
                    />
                <category android:name="android.intent.category.DEFAULT"
                    />
            </intent-filter>
        </activity>
```

```

        <provider
            android:name=".SettingsProvider"
            android:authorities="com.examples.sharepreferences.settingsprovider"
            android:readPermission=
                "com.examples.sharepreferences.permission.READ_PREFERENCES"
            android:writePermission=
                "com.examples.sharepreferences.permission.WRITE_PREFERENCES" >
        </provider>
    </application>

</manifest>

```

这里，我们声明了两个自定义的<permission>元素并把它添加到了<provider>声明中。我们只需要添加这些代码，之后 Android 就会对 query()方法添加读取权限，对 insert()、update()和 delete()方法添加写入权限。我们在应用程序的 Activity 节点中还声明了一个自定义的<intent-filter>，在外部应用程序想要直接启动设置 UI 时，这个<intent-filter>就会派上用场。程序清单 5-28 到程序清单 5-30 显示了本例的剩余部分代码。

程序清单 5-28 res/xml/preferences.xml

```

<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android" >
    <CheckBoxPreference
        android:key="preferenceEnabled"
        android:title="Set Enabled"
        android:defaultValue="true"/>
    <EditTextPreference
        android:key="preferenceName"
        android:title="User Name"
        android:defaultValue="John Doe"/>
    <ListPreference
        android:key="preferenceSelection"
        android:title="Selection"
        android:entries="@array/selection_items"
        android:entryValues="@array/selection_items"
        android:defaultValue="Four"/>
</PreferenceScreen>

```

程序清单 5-29 res/values/arrays.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="selection_items">
        <item>One</item>
        <item>Two</item>
        <item>Three</item>
        <item>Four</item>
    </string-array>
</resources>

```

程序清单 5-30 Preferences Activity

```
//注意应用程序的包名
package com.examples.sharepreferences;

public class SettingsActivity extends PreferenceActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //刚启动就加载默认的设置
        PreferenceManager.setDefaultValues(this, R.xml.preferences, false);

        addPreferencesFromResource(R.xml.preferences);
    }
}
```

这个示例应用程序的设置值是通过一个简单的 PreferenceActivity 来管理的, Preference-Activity 的数据则定义在 preferences.xml 文件。

注意:

在 Android 3.0 中,已经不建议使用 PreferenceActivity,而建议使用 PreferenceFragment,但在本书出版时,PreferenceFragment 还没有被添加到支持库中。因此,为了支持稍早的 Android 版本,这里还是使用了 PreferenceActivity。

示例

接下来,让我们看一下程序清单 5-31 到 5-33,它们定义了另一个应用程序,该应用程序会通过这个 ContentProvider 接口去访问我们的设置数据。

程序清单 5-31 AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.accesspreferences"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk ... />

    <uses-permission
        android:name="com.examples.sharepreferences.permission.
            READ_PREFERENCES" />
    <uses-permission
        android:name="com.examples.sharepreferences.permission.
            WRITE_PREFERENCES" />

    <application ... >
        <activity android:name=".MainActivity" >
            <intent-filter>
```

```

        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>

</manifest>

```

这里的核心部分就是应用程序通过<uses-permission>元素声明使用了我们自定义的权限。这样，该应用程序就可以访问外部 provider 了。否则，ContentResolver 的请求就会导致一个 SecurityException 异常。

程序清单 5-32 res/layout/main.xml

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:id="@+id/button_settings"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Show Settings"
        android:onClick="onSettingsClick" />
    <CheckBox
        android:id="@+id/checkbox_enable"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button_settings"
        android:text="Set Enable Setting"/>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:orientation="vertical">
        <TextView
            android:id="@+id/value_enabled"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/value_name"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/value_selection"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
</RelativeLayout>

```

程序清单 5-33 与 Provider 进行交互的 Activity

//注意包名, 它表示另一个应用程序

```
package com.examples.accesspreferences;

public class MainActivity extends Activity implements OnCheckedChangeListener {

    public static final String SETTINGS_ACTION =
        "com.examples.sharepreferences.ACTION_SETTINGS";
    public static final Uri SETTINGS_CONTENT_URI =
        Uri.parse("content://com.examples.sharepreferences.
            settingsprovider/settings");
    public static class SettingsColumns {
        public static final String _ID = Settings.NameValueTable._ID;
        public static final String NAME = Settings.NameValueTable.NAME;
        public static final String VALUE = Settings.NameValueTable.VALUE;
    }

    TextView mEnabled, mName, mSelection;
    CheckBox mToggle;

    private ContentObserver mObserver = new ContentObserver(new Handler()) {
        public void onChange(boolean selfChange) {
            updatePreferences();
        }
    };

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

        mEnabled = (TextView) findViewById(R.id.value_enabled);
        mName = (TextView) findViewById(R.id.value_name);
        mSelection = (TextView) findViewById(R.id.value_selection);
        mToggle = (CheckBox) findViewById(R.id.checkbox_enable);
        mToggle.setOnCheckedChangeListener(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        //获得最新的 provider 数据
        updatePreferences();
        //当 Activity 可见时, 注册一个观察者来监听数据的变化
        getContentResolver().registerContentObserver(SETTINGS_CONTENT_URI,
            false, mObserver);
    }

    @Override
```

```

public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    ContentValues cv = new ContentValues(2);
    cv.put(SettingsColumns.NAME, "preferenceEnabled");
    cv.put(SettingsColumns.VALUE, isChecked);

    //更新 provider, 会触发我们的观察者
    getContentResolver().update(SETTINGS_CONTENT_URI, cv, null, null);
}

public void onSettingsClick(View v) {
    try {
        Intent intent = new Intent(SETTINGS_ACTION);
        startActivity(intent);
    } catch (ActivityNotFoundException e) {
        Toast.makeText(this,
            "You do not have the Android Recipes Settings App installed.",
            Toast.LENGTH_SHORT).show();
    }
}

private void updatePreferences() {
    Cursor c = getContentResolver().query(SETTINGS_CONTENT_URI,
        new String[] {SettingsColumns.NAME, SettingsColumns.VALUE},
        null, null, null);
    if (c == null) {
        return;
    }

    while (c.moveToNext()) {
        String key = c.getString(0);
        if ("preferenceEnabled".equals(key)) {
            mEnabled.setText( String.format("Enabled Setting = %s",
                c.getString(1)) );
            mToggle.setChecked( Boolean.parseBoolean(c.getString(1)) );
        } else if ("preferenceName".equals(key)) {
            mName.setText( String.format("User Name Setting = %s",
                c.getString(1)) );
        } else if ("preferenceSelection".equals(key)) {
            mSelection.setText( String.format("Selection Setting = %s",
                c.getString(1)) );
        }
    }

    c.close();
}
}

```

因为这是一个新的应用程序, 所以它不能访问第一个应用程序中定义的常量(除非通过库工程引用或者其他方式控制这两个应用程序), 我们必须重新再定义一遍。如果想创建一个带有外部 provider 且其他开发者可以使用该 provider 的应用程序, 最好提供一个 JAR 包,

这个 JAR 会包含一些访问 provider 的 Uri 和字段数据的必要常量，类似于 Android API 所提供的 ContactsContract 和 CalendarContract。

本例中，Activity 在每次进入前台时都会查询 provider 来得到设置的当前值，然后将其显示在一个 TextView 中。查询结果会通过一个 Cursor 返回，该 Cursor 每行有两个值：设置的名称和它的值。该 Activity 还注册了一个 ContentObserver，这样如果在 Activity 可见时数据发送了变化，显示的值可以跟着更新。当用户改变 CheckBox 的值时，会调用 provider 的 update() 来触发观察者更新显示。

最后，如果用户愿意的话，可以通过单击“Show Settings”按钮从外部应用程序直接启动 SettingsActivity。这个过程会调用 startActivity() 方法并传入自定义动作字符串的 Intent，该动作字符串正是 SettingsActivity 所需要的。

5.10 分享其他数据

5.10.1 问题

应用程序需要将它维护的文件或者其他数据提供给设备中其他应用程序使用。

5.10.2 解决方案

(API Level 3)

创建一个 ContentProvider 来作为应用程序数据对外的接口。ContentProvider 通过一些类似数据库的接口 query()、insert()、update() 和 delete()，将指定的数据集暴露给外部请求。但是具体的实现可以自由定义如何将这方法传来的数据传递给实际的模型。

ContentProvider 可以向外部请求暴露任何类型的应用程序数据，包括应用程序各种资源(包括 assets 下的资源)。

5.10.3 实现机制

让我们看一个暴露两个数据源的 ContentProvider 实现：一个数据源是内存中的字符串数组，另一个是 assets 目录中的一些图片文件。跟前面一样，我们必须先在应用程序的 manifest 文件中用 <provider> 标签声明我们的 provider，参见程序清单 5-34 和 5-35。

程序清单 5-34 ContentProvider 的 Manifest 声明

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>
    <application ...>
        <provider android:name=".ImageProvider"
            android:authorities="com.examples.share.imageprovider">
        </provider>
    </application>
</manifest>
```

程序清单 5-35 暴露 assets 中资源的自定义 ContentProvider

```

public class ImageProvider extends ContentProvider {

    public static final Uri CONTENT_URI =
        Uri.parse("content://com.examples.share.imageprovider");

    public static final String COLUMN_NAME = "nameString";
    public static final String COLUMN_IMAGE = "imageUri";

    private String[] mNames;

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        throw new UnsupportedOperationException("This ContentProvider is
            read-only");
    }

    @Override
    public String getType(Uri uri) {
        return null;
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        throw new UnsupportedOperationException("This ContentProvider is
            read-only");
    }

    @Override
    public boolean onCreate() {
        mNames = new String[] { "John Doe", "Jane Doe", "Jill Doe" };
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {
        MatrixCursor cursor = new MatrixCursor(projection);
        for(int i = 0; i < mNames.length; i++) {
            //只插入请求的字段
            MatrixCursor.RowBuilder builder = cursor.newRow();
            for(String column : projection) {
                if(column.equals("_id")) {
                    //使用数组索引作为唯一 id
                    builder.add(i);
                }
                if(column.equals(COLUMN_NAME)) {
                    builder.add(mNames[i]);
                }
            }
        }
    }
}

```

```

        if(column.equals(COLUMN_IMAGE)) {
            builder.add(Uri.withAppendedPath(CONTENT_URI,
                String.valueOf(i)));
        }
    }
    return cursor;
}

@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
    throw new UnsupportedOperationException("This ContentProvider is
read-only");
}

@Override
public AssetFileDescriptor openAssetFile(Uri uri, String mode) throws
FileNotFoundException {
    int requested = Integer.parseInt(uri.getLastPathSegment());
    AssetFileDescriptor afd;
    AssetManager manager = getContext().getAssets();
    //为请求项返回正确的 asset 资源
    try {
        switch(requested) {
            case 0:
                afd = manager.openFd("logo1.png");
                break;
            case 1:
                afd = manager.openFd("logo2.png");
                break;
            case 2:
                afd = manager.openFd("logo3.png");
                break;
            default:
                afd = manager.openFd("logo1.png");
        }
        return afd;
    } catch (IOException e) {
        e.printStackTrace();
        return null;
    }
}
}

```

你可能已经猜到了，这个示例对外暴露的是 3 个 logo 图片资源。本例中选用的图片如图 5-3 所示。



图 5-3 示例图片，保存在 assets 文件夹中的 ogo1.png (左)、logo2.png (中)和 logo3.png (右)

首先要注意，我们对外暴露的是 assets 目录下的只读内容，所以不必支持继承的 insert()、update()、或 delete()方法，对于这些方法只需要抛出 UnsupportedOperationException 即可。

在创建 provider 时，会创建一个存放人名的字符串数组，这之后 onCreate()返回 true，告诉系统 provider 创建成功。provider 将它自己的 URI 和可读取的字段名以常量的形式暴露。外部应用程序就可以通过这些值请求所需的数据了。

这个 provider 只支持查询其内部的数据。为了实现对特定记录或某个内容子集的条件查询，应用程序可以将查询条件传递给 query()的 selection 和 selectionArgs。在这个示例中，所有对 query()的调用都会构建一个包含其中所有 3 个元素的 cursor。

这个 provider 中使用的 cursor 实现是 MatrixCursor，这种 cursor 专用于非数据库中的数据。这个示例逐个字段处理列表，根据其中每个字段的信息构建行。每行都是通过调用 MatrixCursor.newRow()创建的，这个方法会返回一个用于添加列数据的 Builder 实例。一定要确保列数据添加的顺序和所要求映射的顺序保持一致。

name 字段中的值就是本地数组中相应的字符串。在 Android 的大部分 ListAdapter 中都需要 _id 字段就是以数组的索引返回的。每一行的 image 字段中的数据实际上是一个表示图片文件的 URI，它以 provider 的内容 URI 为基础，加上数组的索引构建而成。

当外部应用程序通过 ContentResolver.openInputStream()获取该内容时，会调用覆写后 openAssetFile()，返回指向 assets 目录中某个图片文件的 AssetFileDescriptor。这个实现通过解析内容 URI 结尾的索引来判断要返回的是哪一张图片。

使用实例

下面来看看如何实现该 provider，以及如何在 Android 应用程序中访问它，参见程序清单 5-36。

程序清单 5-36 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.share"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="3" />

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

```

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<provider android:name=".ImageProvider"
    android:authorities="com.examples.share.imageprovider">
</provider>
</application>
</manifest>

```

要实现该 provider, 拥有该内容的应用程序必须在其 manifest 文件中声明一个<provider> 标签, 确定 ContentProvider 的名称和相应的权限。Authority 值必须跟暴露的内容 URI 的基础部分保持一致。provider 必须在 manifest 中声明, 系统才能将其初始化并运行。即使拥有该 provider 的应用程序没有运行, 系统可以单独运行 provider。参见程序清单 5-37 和 5-38。

程序清单 5-37 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="20dip"
        android:layout_gravity="center_horizontal"
    />
    <ImageView
        android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="50dip"
        android:layout_gravity="center_horizontal"
    />
    <ListView
        android:id="@+id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
    />
</LinearLayout>

```

程序清单 5-38 从 ImageProvider 中读取内容的 Activity

```

public class ShareActivity extends FragmentActivity implements
    LoaderManager.LoaderCallbacks<Cursor>, AdapterView.OnItemClickListener {
    private static final int LOADER_LIST = 100;
    SimpleCursorAdapter mAdapter;

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

```

```

getSupportLoaderManager().initLoader(LOADER_LIST, null, this);
setContentView(R.layout.main);

mAdapter = new SimpleCursorAdapter(this, android.R.layout.simple_list_item_1,
    null, new String[]{ImageProvider.COLUMN_NAME},
    new int[]{android.R.id.text1}, 0);

ListView list = (ListView)findViewById(R.id.list);
list.setOnItemClickListener(this);
list.setAdapter(mAdapter);
}

@Override
public void onItemClick(AdapterView<?> parent, View v, int position,
    long id) {
    //得到 selection 的 cursor
    Cursor c = mAdapter.getCursor();
    c.moveToPosition(position);

    //加载 name 字段到 TextView
    TextView tv = (TextView)findViewById(R.id.name);
    tv.setText(c.getString(1));

    ImageView iv = (ImageView)findViewById(R.id.image);
    try {
        //从 image 字段中加载内容到 ImageView 中
        InputStream in =
            getContentResolver().openInputStream(Uri.parse(c.getString(2)));
        Bitmap image = BitmapFactory.decodeStream(in);
        iv.setImageBitmap(image);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    String[] projection = new String[]{"_id",
        ImageProvider.COLUMN_NAME,
        ImageProvider.COLUMN_IMAGE};
    return new CursorLoader(this, ImageProvider.CONTENT_URI,
        projection, null, null, null);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    mAdapter.swapCursor(data);
}

```

```

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}
}

```

重点:

这个示例需要支持库才可以在 Android 1.6 及以上版本中访问 Loader 类。如果应用程序的目标版本为 Android 3.0+，需要将 `FragmentActivity` 替换为 `Activity` 以及将 `getSupportLoaderManager()` 替换为 `getLoaderManager()`。

在这个示例中，从自定义的 `ContentProvider` 获得了一个托管 `cursor`，指向暴露的 URI 和数据的字段名称。然后用 `SimpleCursorAdapter` 将数据关联到 `ListView` 并显示 `name` 值。

当用户单击列表中的某项内容时，`cursor` 会移动到相应的位置并在上方显示相应的名称和图片。Activity 在这里调用 `ContentResolver.openInputStream()`，通过保存在字段中的 URI 来访问图片文件。

图 5-4 是应用程序运行并选择列表中最后一项(Jill Doe)的效果。

注意，这里并没有显示关闭 `Cursor` 的连接，因为这个 `Cursor` 是由 `Loader` 创建的，`Loader` 会替你管理它。

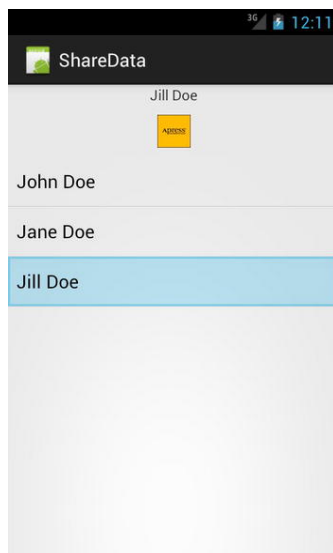


图 5-4 显示来自 `ContentProvider` 的资源的 Activity

5.11 实用工具推荐: SQLite3

Android 提供了 `sqlite3` 工具(位于 Android SDK 主目录的 `tools` 子目录下)，用来在开发平台或者(与 `adb`，即 `Android Debug Bridge` 工具一起)Android 设备上创建新的数据库以及管理现有的数据库。如果还不熟悉 `sqlite3`，可以打开浏览器访问 <http://sqlite.org/sqlite.html> 来学习这个命令行工具的简短教程。

可以在 `sqlite3` 后面加上一个文件名参数(如 `sqlite3employees`)，如果指定的数据库文件不存在，就会创建这个文件(至少要创建一个表)，否则就会直接打开该文件。然后会进入 `sqlite3` 的命令行解析器，在这里可以允许 `sqlite3` 以点开头的命令以及 SQL 语句。如图 5-5 所示，还可以不带参数运行 `sqlite3`，进入命令行解析器。

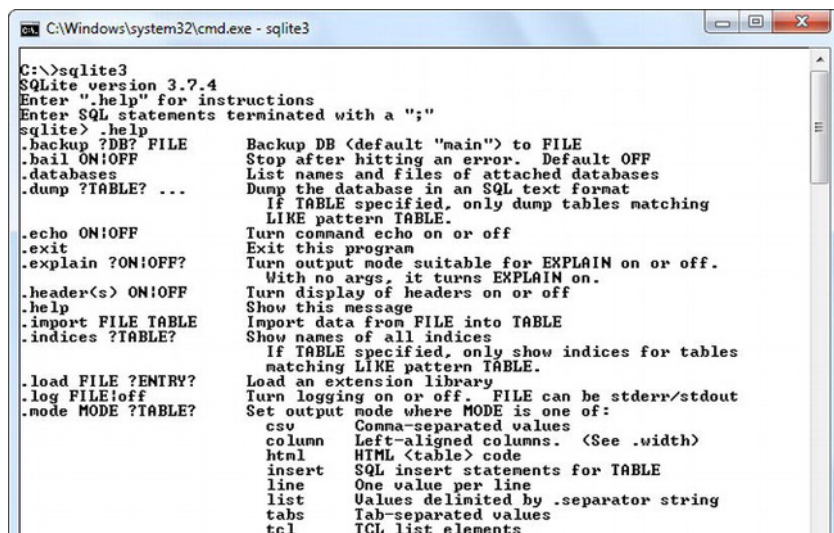


图 5-5 不在数据库文件名参数的情况下执行 `sqlite3`

图 5-5 显示了输入 `sqlite3` 后的欢迎信息，在 `sqlite>` 提示符后面可以输入命令。这里还包含一些帮助信息，可以输入 `sqlite3` 特定的“`.help`”获得更多的帮助信息。

提示：

在不带参数运行 `sqlite3` 之后，可以通过 SQL 语句创建数据库并填充数据(还可以创建索引)，然后在离开 `sqlite3` 之前用 `.backup filename`(其中 `filename` 表示保存数据的文件)命令将数据库保存到文件。

在你的开发机器上创建好数据库后，可以将它上传到 Android 设备上。这个过程需要调用 `adb` 工具的 `push` 命令，相应的命令行语法如下(为了便于阅读，分两行显示)：

```
adb [-s <serialNumber>] push local.db
/data/data/<application package>/databases/remote.db。
```

当你在开发平台上创建了一个数据库之后，可以把它上传到 Android 设备上。要实现这个功能，可以调用 `adb` 工具的 `push` 命令，命令如下：

```
adb [-s <serialNumber>] push local.db/data/data/<application
package>/databases/remote.db
```

注意：

`local` 和 `remote` 并不是真正的数据库名称，只是指代而已。通常，数据库文件的扩展名为 `.db`(但并不强制)。同样，`/data/data/<application package>` 代表的是应用程序私有的存储空间，`application package` 指的是应用程序唯一的包名。

当开发平台只连接了一个设备时,就不需要使用-s <serialNumber>了,本地数据库可以直接上传到这个设备上。而当连接了多个设备时,则可以使用-s <serialNumber>来区分特定的设备(如-semulator-5556)。

另外,你可能还需要将设备上的数据库下载到开发平台上,比如要在应用程序的桌面版本中使用该数据库。这个可以通过 adb 的 pull 命令来实现,语法如下(为了增加可读性,分两行显示):

```
adb [-s <serialNumber>] pull
/data/data/<applicationpackage>/databases/remote.dblocal.db
```

如果想通过 sqlite3 管理存储在设备上的数据库,需要在该设备的 adb 远程命令行解释器中运行 sqlite3。可以通过下面的语法启动 adb 和 sqlite3:

```
adb [-s <serialNumber>] shell# sqlite3 /data/data/<application
package>/databases/remote.db
```

adb 命令行解释器是以#提示符开头的。在 sqlite3 后面添加设备上已经存在的数据库文件路径就可以操作这个数据库文件,而在 sqlite3 后面添加新的数据库名称则会创建一个新的。当然,也可以不带参数启动 sqlite3。

输入 sqlite3 命令后会显示和图 5-5 一样的欢迎信息。输入 sqlite3 命令和 SQL 语句就可以管理 remote.db(或者创建一个新的数据库),然后退出 sqlite3(exit 或 quit),再退出 adb 命令行解释器(exit)。

注意:

运行了 adb 命令行解释器后再运行 sqlite3 的话,可能看到的版本号与直接运行 sqlite3 工具看到的版本号不同。

5.12 Univerter 和 SQLite3

第 1 章介绍了 Univerter。这个单位转换工具可以实现各种单位之间的换算(例如,华氏温度到摄氏温度)。

注意:

附录 D 详细描述了 Univerter 的完整结构,包括源代码、资源和 manifest。你如果对 Univerter 还没有了解的话,请先阅读一下该附录。

虽然很实用,但 Univerter 还是有缺陷的,这是因为它的转换是硬编码的。每次在转换列表中添加新的转换关系时,都需要重新构建这个应用程序。要想解决这个问题,我们可以将其他的转换关系保存到一个数据库中,然后在运行时将这些转换关系添加到硬编码列表中。

本节对 Univerter 进行了一下改进,会通过保存到数据库中的换算关系扩展应用程序原有的换算关系。首先,如果数据库存在的话,会将数据库中的转换关系添加到 Univerter 已有的换算关系中。如果数据库不存在,Univerter 还会使用原有的 200 个硬编码的换算关

系，不会影响到用户。

在改进之前，需要先确认几个问题：

- 现有的源代码需要做多大的改动，是否需要修改资源？
- 虽然对于简单的转换关系(将输入值直接乘以一个换算因子)可以很容易地保存到数据库中，但对于需要更多计算的复杂转换关系(例如，摄氏温度转换为华氏温度)该如何存储呢？
- Univerter 可以快速响应它的硬编码换算列表。但如果 Univerter 在每次 Activity 创建时都去访问数据库和创建换算列表的话，这种响应速度势必会变慢(并且会让用户很沮丧，尤其是换算的数量增加时)。如何才能提高响应速度呢？
- 每种换算关系会有一个字符串资源 ID(用来标识换算名称)，这样就可以将应用程序进行本地化从而支持多种语言(将来可能会用到的功能)。但由于字符串资源和它们的 ID 不能够在运行时动态创建，该如何处理新的换算名称和种类名称字符串呢？

这个问题的答案如下：

- 只需要对 Univerter 做两处改动，而且这两种改动影响不大。同样，需要向 Category 和 Conversion 中添加几个构造函数和普通方法以及一个私有变量：另外还要对这些类中的一小部分代码做一下改进。最后，需要一个新的 SQLiteOpenHelper 类。
- 第二个问题在本次改进中还没有得到解决。不过，可以通过创建一个简单的表达式语音，然后将代表计算和异常逻辑的字符串保存到数据库的方式来解决它。当访问数据库时，这些字符串可以被解析到动态创建的 Converter 中去。
- 每次 onCreate(Bundle)被调用时(当创建一个 Activity 时)不去数据库中获取换算关系就可以提供性能。而是通过一个静态的 Boolean 变量标识数据库之前是否已经访问过了，后续的执行逻辑会基于这个变量的值。
- 运行时并不能创建其他字符串资源 ID。但是，本地化文本可以保存到数据库中(可能在不同的表中或者不同的表的字段中)，然后通过判断当前的设备语言并访问相应的表/字段来获取正确的文本。

或许解决性能问题的最好方法就是生成 200 个换算关系的默认列表(启动时，通过一个后台线程生成)，并构建一个所有可能使用的换算关系的平行列表。下次用户单击 CAT 或 CON 按钮时，通过改变一个引用，可以将默认列表替换为平行列表。

提示：

使用 android.os.AsyncTask 类可以很方便在后台(通过一个工作线程)创建一个平行列表，然后在前台(通过 Activity 线程)将该列表的引用赋给一个 Category[] 变量。这个变量的引用将会被赋给单击 CAT 或 CON 按钮时所对应的换算种类。

还有另一个问题需要回答：带有新的换算关系的数据库应该如何分发给用户呢？

应用程序在每次运行时都不应该访问服务器并下载数据库，这样做通常也没什么必要(数据库的更新频率是多少才合适呢？)，只会浪费电量。也许在用户界面中加入一个“Upgrade”按钮是个不错的选择，这样用户就可以自己选择何时更新新数据库。

5.12.1 创建数据库

改进 Univerter 的第一步就是要设计一个数据库。数据库的名称和结构(根据表的特征和表结构)是什么样呢?

换算关系会被保存到一个名为 conversions.db 的数据库中。这个文件保存在/data/data/ca.tutortutor.univerter/databases/目录下。

我们会创建以下的表:

- **Categories:** 这个表会包含一个 id 字段(文本类型)用来保存种类表的非本地化标识符。这个表中还会包含一个 name_en 字段(文本类型), 它表示种类名称的英文名称。后面必要时可以添加更多的字段(例如 name_en_GB 和 name_fr)。
- **category name:**这个表是以种类表中id字段对应的项来命名的。它包含一个 name_en 字段(文本类型), 保存英文换算名称, 另外还有一个 multiplier 字段(real 类型), 保存乘数器。后面可以添加保存自定义转换字符串的字段(当乘数器包含一个 0 项时, 可以访问转换字符串字段)

sqlite3 工具可以让我们使用样本语句创建这个数据库, 当 SQL 命令保存在一个单独文件中时, 这种方式很方便(参见程序清单 5-39)。

程序清单 5-39 构建 conversions.db 的命令集合

```
create table categories(id text, name_en text);
insert into categories(id, name_en) values('density', 'DENSITY');
insert into categories(id, name_en) values('energy', 'ENERGY AND WORK');

create table density(name_en text, multiplier real);
insert into density(name_en, multiplier)
  values("EARTH'S DENSITY (MEAN) > PSI/1000 FEET", 2392.204767079);
insert into density(name_en, multiplier)
  values("PSI/1000 FEET > EARTH'S DENSITY (MEAN)", 0.000418024);

create table energy(name_en text, multiplier real);
insert into energy(name_en, multiplier)
  values('WATT-HOURS > TONS (EXPLOSIVE)', 0.00000086);
insert into energy(name_en, multiplier)
  values('TONS (EXPLOSIVE) > WATT-HOURS', 1162222.222222);
```

程序清单 5-38 显示了 init.sql 文件(扩展名是可选的)的内容, 其中包含了创建表命令和插入命令(为了阅读方便, 每个插入语句都分两行显示)。下面的 Windows 命令会使用 sqlite3 和该文件创建并构建 conversions.db:

```
type init.sql | sqlite3 conversions.db
```

这个命令首先会使用 type init.sql 将 init.sql 文件的内容输出到标准输出中。管道标识(I)会将这个标准输入内容输出到 sqlite3 命令中, sqlite3 会在标准输入中执行每个 SQL 语句从而创建并构建 conversions.db。

提示:

在重新构建 conversions.db 之前, 要先执行删除 conversions.db。否则, 会得到表已经存在的错误。

5.12.2 扩展 Category 和 Conversion 类

改进 Univerter 的第二步就是重构 Conversion 和 Category 类, 以便可以被前面提及的 DBHelper 类所使用。

程序清单 5-40 展示了重构后的 Conversion 类。附录 D 中黑体部分表示了这个类的改动部分。

程序清单 5-40 重构后的 Conversion 类

```
package ca.tutortutor.univerter;

import android.content.Context;

class Conversion {
    private int nameID;
    private String name;
    private Converter converter;
    private boolean canBeNegative;

    Conversion(int nameID, final double multiplier) {
        this(nameID,
            new Converter() {
                @Override
                public double convert(Context ctx, double value) {
                    return value*multiplier;
                }
            },
            false);
    }

    Conversion(int nameID, Converter converter, boolean canBeNegative) {
        this.nameID = nameID;
        this.converter = converter;
        this.canBeNegative = canBeNegative;
    }

    Conversion(String name, final double multiplier) {
        this(name, new Converter() {
            @Override
            public double convert(Context ctx, double value) {
                return value*multiplier;
            }
        },
        false);
    }
}
```

```

    }

    Conversion(String name, Converter converter, boolean canBeNegative) {
        this.name = name;
        this.converter = converter;
        this.canBeNegative = canBeNegative;
    }

    boolean canBeNegative() {
        return canBeNegative;
    }

    Converter getConverter() {
        return converter;
    }

    String getName(Context ctx) {
        return (name == null) ? ctx.getString(nameID) : name;
    }
}

```

name 字段、新的构造函数以及改进后的 getName(Context)方法(name 不为 null 时会返回 name 的值)解决了换算关系没有字符串资源 ID 的问题(数据库中保存的换算关系没有字符串资源 ID)。

程序清单 5-41 展示了重构后的 Category 类。附录 D 中黑体部分表示了这个类的改动部分。

程序清单 5-41 重构后的 Category 类

```

package ca.tutortutor.univerter;

import android.content.Context;

class Category {
    private int nameID;
    private String name;
    private Conversion[] conversions;
    private String[] conversionNames;

    Category(int nameID, Conversion[] conversions) {
        this.nameID = nameID;
        this.conversions = conversions;
    }

    Category(String name, Conversion[] conversions) {
        this.name = name;
        this.conversions = conversions;
    }
}

```

```

Conversion getConversion(int index) {
    return conversions[index];
}

String[] getConversionNames(Context ctx) {
    if (conversionNames == null) {
        conversionNames = new String[conversions.length];
        for (int i = 0; i < conversionNames.length; i++) {
            conversionNames[i] = conversions[i].getName(ctx);
        }
    }
    return conversionNames;
}

String getName(Context ctx) {
    return (name == null) ? ctx.getString(nameID) : name;
}

int getNumConversions() {
    return conversions.length;
}

void setConversions(Conversion[] conversions) {
    this.conversions = conversions;
}
}

```

同样为了解决数据库中保存的新的种类名称没有字符串资源 ID 的问题，程序清单 5-41 介绍了如下的方法：

- **int getNumConversions():** 这个方法会返回保存在 Category 中的 Conversion 实例的数量。DBHelper 在合并新的 Conversion 实例(相同种类中新的换算关系)时会调用这个方法，这个新的 Conversion 实例是通过现有的 Conversion 实例从数据库中获得的。
- **void setConversions(Conversion[] conversions):** Category 的这个方法用 conversions 替代了之前的 Conversion 实例中的数组。DBHelper 在合并现有和新的 Conversion 实例到一个临时的 Conversion 实例数组后会调用这个方法。

5.12.3 DBHelper 类简介

改进 Univerter 的第三步就是引入了 DBHelper 类，该类封装了对数据库的访问，对 Univerter.java 的影响也很小。参见程序清单 5-42。

程序清单 5-42 DBHelper 类

```

package ca.tutortutor.univerter;

import android.content.Context;

```

```

import android.database.Cursor;
import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;

public class DBHelper extends SQLiteOpenHelper {
    private final static String DB_PATH =
        "data/data/ca.tutortutor.univerter/databases/";
    private final static String DB_NAME = "conversions.db";

    private final static int CATEGORIES_ID_COLUMN_ID = 0;
    private final static int CATEGORIES_NAME_EN_COLUMN_ID = 1;

    private final static int CATTABLE_NAME_EN_COLUMN_ID = 0;
    private final static int CATTABLE_MULTIPLIER_COLUMN_ID = 1;

    private Context ctx;
    private SQLiteDatabase db;

    public DBHelper(Context ctx) {
        super(ctx, DB_NAME, null, 1);
        this.ctx = ctx;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //什么也不做，我们不会创建一个新的数据库
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldver, int newver) {
        //什么也不做，我们不会升级数据库
    }

    public Category[] updateCategories(Category[] categories) {
        try {
            String path = DB_PATH+DB_NAME;
            db = SQLiteDatabase.openDatabase(path, null,
                SQLiteDatabase.OPEN_READONLY|
                SQLiteDatabase.NO_LOCALIZED_COLLATORS);
            Cursor cur = db.query("categories", null, null, null, null, null,
                null);
            if(cur.getCount() == 0) {

```

```

        return categories;
    }
    Comparator<Category> cmpCat;
    cmpCat = new Comparator<Category>() {
        @Override
        public int compare(Category c1, Category c2) {
            return c1.getName(ctx).compareTo(c2.getName(ctx));
        }
    };
    Set<Category> catSet = new TreeSet<Category>(cmpCat);
    Comparator<Conversion> cmpCon;
    cmpCon = new Comparator<Conversion>() {
        @Override
        public int compare(Conversion c1, Conversion c2) {
            return c1.getName(ctx).compareTo(c2.getName(ctx));
        }
    };
    Set<Conversion> conSet = new TreeSet<Conversion>(cmpCon);
    while (cur.moveToNext()) {
        String catID = cur.getString(CATEGORIES_ID_COLUMN_ID);
        String catEn = cur.getString(CATEGORIES_NAME_EN_COLUMN_ID);
        Conversion[] conversions = getConversions(catID);
        for (int i = 0; i < categories.length; i++) {
            Category cat = categories[i];
            catSet.add(cat);
            if (catEn.equals(cat.getName(ctx))) {
                int numCon = cat.getNumConversions();
                for (int j = 0; j < numCon; j++) {
                    conSet.add(cat.getConversion(j));
                }
                for (int j = 0; j < conversions.length; j++) {
                    conSet.add(conversions[j]);
                }
                cat.setConversions(conSet.toArray(new Conversion[0]));
                conSet.clear();
            }
            if (i == categories.length-1) {
                catSet.add(new Category(catEn, conversions));
            }
        }
    }
    return catSet.toArray(new Category[0]);
} catch (SQLException sqle) {
    //什么也不做
} finally {
    if (db != null)
        db.close();
}
return categories;
}

```

```

private Conversion[] getConversions(String catID) {
    try {
        Cursor cur = db.query(catID, null, null, null, null, null, null);
        if (cur.getCount() == 0) {
            return new Conversion[0];
        }
        List<Conversion> conList = new ArrayList<Conversion>();
        while (cur.moveToNext()) {
            String name_en = cur.getString(CATTABLE_NAME_EN_COLUMN_ID);
            double multiplier = cur.getDouble(CATTABLE_MULTIPLIER_COLUMN_ID);
            Conversion con = new Conversion(name_en, multiplier);
            conList.add(con);
        }
        return conList.toArray(new Conversion[0]);
    } catch (SQLException sqle) {
        //什么也不做
    }
    return null;
}
}

```

DBHelper 继承了 android.database.sqlite.SQLiteOpenHelper 类并覆写了它的 onCreate() 和 onUpgrade() 抽象方法。覆写方法什么也没做，最重要的信息就是要知道数据库是否可以打开。

数据库是在 Category[] updateCategories(Category[] categories) 中打开的，该方法会将其他 Category 和/或 Conversion 实例合入到 categories 数组参数里面。然后返回这个数组。

数据库成功打开之后，会调用 db.query("categories", null, null, null, null, null, null) 来返回一个 android.database.Cursor 对象，用来遍历 categories 表中的所有行。该对象就是一个 cursor(指向表的每一行)。

遍历时会连续调用 Cursor 的 boolean moveToNext() 方法，它会将 cursor 定位到每行的开头(最开始时，cursor 的位置是在第一行的前面)。每次遍历首先都会查找行的 id 以及 name_en 字段的值。

注意：

虽然这只是一个小小的练习，但从长远来看，如果有多种语言环境的话，直接访问 name_en 字段并不是一种明智的解决方案。更好的方案则是得到默认的语言值(调用 java.util.Locale class 类的 Locale getDefault() 类方法)，然后基于这个值再选择合适的字段。或许可以在找不到合适的特定语言的字段时，把 name_en 用作一个默认字段。

获得这些值后，会调用 private Conversion[] getConversions(String catID) 方法并传入 id 值(catID)来加载指定种类的表中的所有 conversion 行的内容并返回一个 Conversion 数组。

然后事情就简单了，只需要遍历当前所有的种类，然后判断换算关系是否属于某个已经存在的种类(并且必须添加到种类的换算关系中)或者是否需要创建一个新的种类来保存这些换算关系。

这里会创建两个 `java.util.TreeSet` 实例来保存 `Category` 和 `Conversion` 实例。使用 `TreeSet` 是因为 `TreeSet` 可以防止包含重复的对象并且可以将包含的对象进行排序。

这里创建了两个 `java.util.Comparator` 对象来比较 `Category` 对象或 `Conversion` 对象的名称。这些比较器会传入到 `TreeSet` 的构造函数中，从而保证 `TreeSet` 中的对象是按照名字升序进行排序的。

这里可能会抛出 `android.database.SQLException` 异常。如果发生这种异常，为了防止打扰用户并不会打印任何消息。而不管是否抛出异常都会关闭数据库。

警告：

`DBHelper` 提供了一种快速(但远不是最优)的解决方案来更新种类数组。另外，它还是有问题的。例如，当改变数据库结构后，这里的代码就不能用了。同样，种类数组和种类数组中的换算关系可能出现状态不一致的情况，这时应该抛出 `android.database.SQLException` 异常。

5.12.4 扩展 Univerter 类

改进 `Univerter` 的最后一步就是重构 `Univerter` 类来使用 `DBHelper`。首先，引入一个 `categoriesUpdated` 类变量，如下：

```
public class Univerter extends Activity {
    private static boolean categoriesUpdated;
    private static Category[] categories;
    static {
        categories = new Category[]
```

接下来，修改 `onCreate(Bundle)` 来初始化 `DBHelper` 并调用它的 `updateCategories(Category[])` 方法，如下：

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

    if (!categoriesUpdated) {
        DBHelper dbh = new DBHelper(this);
        categories = dbh.updateCategories(categories);
        categoriesUpdated = true;
    }

    catNames = new String[categories.length];
    for (int i = 0; i < catNames.length; i++) {
        catNames[i] = categories[i].getName(Univerter.this);
    }
}
```

这样，现在实现的 `Univerter` 就可以兼容其他换算关系了。

提示：

如果在数据库中保存了很多的换算关系，有可能会碰到可怕的 `Application Not Responding`

对话框。如果出现了这种情况,可以考虑在一个后台线程中创建一个平行的 `Category[]` 数组,然后在 Activity 线程中响应 CAT 或 CON 按钮单击事件时将该数组赋给 `categories`。

5.12.5 运行改进后的 Univerter 应用程序

在有无 `conversions.db` 的情况下都可以在设备上运行 Univerter。当 `conversions.db` 文件不存在时,Univerter 会基于 200 个换算关系进行正常运行。反之当该文件存在时,Univerter 会扩展为一个更加有用的应用程序。

下面的命令(为了便于阅读,分两行显示)会在设备上合适的位置保存 `conversions.db`:

```
adb push conversions.db/data/data/ca.tutortutor.univerter/  
databases/conversions.db
```

注意:

如果对 `conversions.db` 进行了改动,则在将改动后的数据库添加到设备上之前,必须先卸载 Univerter 并重新安装它。

启动 Univerter 并单击 CAT 按钮。应该会看到图 5-6 所示的新 DENSITY 种类。

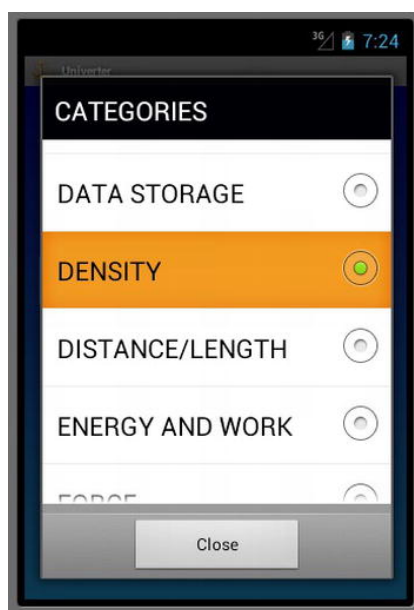


图 5-6 一个新的 DENSITY 种类被加入到了 CATEGORIES 列表中

关闭这个对话框并单击 CON 按钮。同样应该看到图 5-7 所示的 `density` 种类对应的换算关系列表。

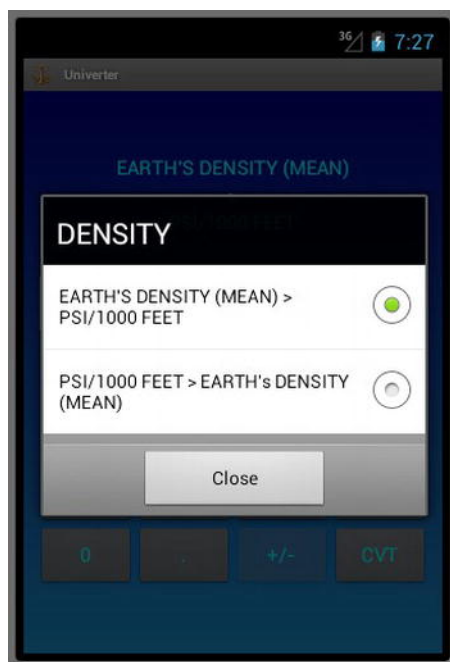


图 5-7 经过排序的换算关系列表

如果处在种类选择界面，如果选择种类“ENERGY AND WORK”，这时会看到新的“TONS (EXPLOSIVE) > WATT-HOURS”和“WATT-HOURS > TONS (EXPLOSIVE)”换算关系(按字母顺序排序)。

5.13 小结

本章介绍了一些在 Android 设备上实现数据持久化的方法。首先介绍了如何快速创建设置界面，如何使用 preferences 和简单的方法保存基本数据类型。然后学习了设置文件和存储文件的存放位置和方法，还学习了如何在应用程序间共享已经保存的数据。在第 6 章中，我们会研究如何利用操作系统的服务实现后台操作和应用程序间通信。

第 6 章

与系统交互

Android 操作提供提供了很多应用程序可以使用的服务。其中很多服务让应用程序不仅可以跟用户实现简单的互动，还可以跟系统互动。应用程序可以进行定时提醒、运行后台服务以及彼此发送消息。所有这些就可以让 Android 应用程序可以更好地跟移动设备融合在一起。此外，Android 还提供了一系列接口用来将 Android 核心应用程序采集到的数据共享给你的应用程序。通过这些接口，任何应用程序都可以与 Android 平台的核心功能整合在一起，为其添加新功能，改进旧功能，最终达到提升用户体验的目的。

6.1 后台通知

6.1.1 问题

应用程序正在后台运行，没有用户可见的界面，但还必须将发生的重要事情通知给用户。

6.1.2 解决方案

(API Level 4)

使用 `NotificationManager` 发送一个状态栏通知。通知是一种温和的方式，它可以引起用户的注意。对于接收一条新消息、有可用的更新或者完成了一个长时间运行的工作，通知都可以很完美地完成。

6.1.3 实现机制

通过所有的系统控件，如 `Service`、`BroadcastReceiver` 或者 `Activity`，都可以将一个通知发送到 `NotificationManager`。在程序清单 6-1 中会看到一个 `Activity`，它通过一个延迟操作模拟了长时间运行的任务，在该任务完成时会发送一个通知。

程序清单 6-1 发送通知的 Activity

```

public class NotificationActivity extends Activity implements
View.OnClickListener {

    private static final int NOTE_ID = 100;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Button button = new Button(this);
        button.setText("Post New Notification");
        button.setOnClickListener(this);
        setContentView(button);
    }

    @Override
    public void onClick(View v) {
        //单击后延迟 10 秒执行
        handler.postDelayed(task, 10000);
        Toast.makeText(this, "Notification will post in 10 seconds",
            Toast.LENGTH_SHORT).show();
    }

    private Handler handler = new Handler();
    private Runnable task = new Runnable() {
        @Override
        public void run() {
            NotificationManager manager =
                (NotificationManager) getSystemService(Context.NOTIFICATI-
                    ON_SERVICE);
            Intent launchIntent =
                new Intent(getApplicationContext(), NotificationActivity.class);
            PendingIntent contentIntent =
                PendingIntent.getActivity(getApplicationContext(), 0,
                    launchIntent, 0);

            //使用发送的时间创建通知
            NotificationCompat.Builder builder =
                new NotificationCompat.Builder(NotificationActivity.this);

            builder.setSmallIcon(R.drawable.icon)
                .setTicker("Something Happened")
                .setWhen(System.currentTimeMillis())
                .setAutoCancel(true)
                .setDefaults(Notification.DEFAULT_SOUND)
                .setContentTitle("We're Finished!")
                .setContentText("Click Here!")
                .setContentIntent(contentIntent);
            Notification note = builder.build();

```

```

        //发送通知
        manager.notify(NOTE_ID, note);
    }
};
}

```

这个示例使用了一个 `Handler` 来定时执行一个任务,即在按钮的监听器中调用 `Handler.postDelayed()`方法在按钮单击 10 秒后发送一个通知。不管 `Activity` 是否处于前台,这个任务都会执行,所以如果用户厌烦并关闭了应用程序,他还是可以得到通知的。

当定时任务执行时,会先使用 `Notification.Builder` 创建一个新的通知。这里需要一个图标资源和标题字符串,这些信息在显示通知时会呈现在状态栏上。此外,我们还传入了一个时间值(单位毫秒),它会作为事件时间显示在通知列表中。这里我们将这个值设为发送通知的时间,但在你自己的应用程序中这个时间可能会有不同的含义。

重点:

这个示例中我们使用了 `NotificationCompat.Builder`,它是支持库提供的,这个新的 API 是 Android 3.0 才引入的,通过支持库可以在 Android 1.6 中使用它。如果你的目标平台就是 Android 3.0+,则可以在代码中将 `NotificationCompat.Builder` 替换为 `Notification.Builder`。

在创建通知前,可以添加其他一些有用参数,如当用户在通知栏中展开通知时显示在通知上的详细信息。

我们传入了一个 `PendingIntent` 参数,它指向了我们的 `Activity`。这个 `Intent` 使得通知具有可交互功能,即用户在通知列表中单击通知时可以启动一个 `Activity`。

注意:

对于每个通知事件,这个 `Intent` 都会启动一个新的 `Activity`。如果更愿意用一个已经存在的 `Activity` 实例来响应用户的操作(好在栈中就有一个),则应用加上 `Intent` 标志及合适的 `manifest` 参数,例如 `Intent.FLAG_ACTIVITY_CLEAR_TOP` 和 `android:launchMode="singleTop"`。

为了让通知不仅仅以动画的形式出现在状态栏中,我们在 `Notification.defaults` 中加上系统默认的通知声音,当通知发出时就会播放这个声音。另外还可以加入 `Notification.DEFAULT_VIBRATION` 和 `Notification.DEFAULT_LIGHTS` 等各种参数。

提示:

如果想要通知播放一个自定义的声音,可以将 `Notification.sound` 参数设置为一个指向某个文件或者 `ContentProvider` 的 `Uri`。

给 `Notification` 加上各种标志可以进一步自定义通知。例如启用 `Notification.FLAG_AUTO_CANCEL`,意思是当用户在列表中单击该通知后,通知应尽快取消或从列表中移除。如果没有这个标志,通知就会一直在列表中,直到手动调用 `NotificationManager.cancel()`或 `NotificationManager.cancelAll()`才会移除。

- `FLAG_INSISTENT`

- 重复播放声音，直到用户进行响应
- FLAG_NO_CLEAR
 - 用户单击“Clear Notifications”不会清除该通知，只能通过调用 `cancel()` 移除通知

通知准备好后，通过 `NotificationManager.notify()` 方法就可以发送给用户，该方法也会使用一个 ID 参数。应用程序中每种类型的通知都应该拥有唯一的 ID。同一时刻，管理器只允许相同 ID 的一个通知显示在列表中，新的通知会覆盖旧的通知。此外，在手动取消某个特定通知时，也需要传入 ID。

运行这个示例，如图 6-1 所示的 Activity 会显示一个按钮。当用户按下按钮时，稍等一会就会看到通知，即使 Activity 已经不可见，仍然可以收到通知(见图 6-2)。



图 6-1 按下按钮会发送通知

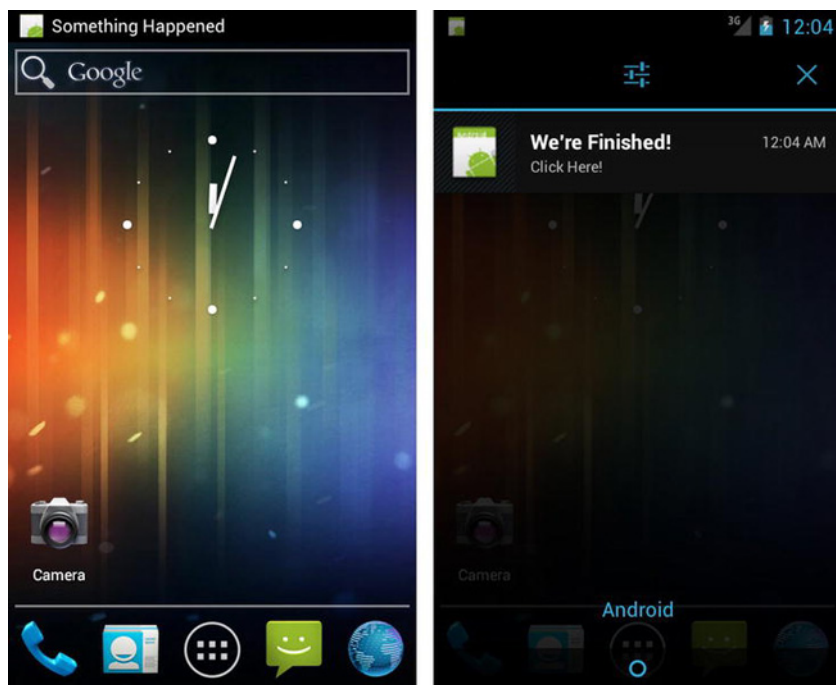


图 6-2 通知出现(左)和通知显示在列表中(右)

1. 扩展通知样式

(API Level 16)

从 Android 4.1 开始，可以在通知 view 上直接显示更加丰富的、具有交互性的信息，这就是通知的样式。当前处在屏幕顶端的 window shade 中的所有通知默认都是展开的，但用户可以使用两个手指的手势操作展开任意其他的通知。然而，扩展 view 并不会取代传统的 view，相反，它在某些情况下会提升用户体验。

Android 默认提供了三种样式(它们都实现了 `Notification.Style`):

- **BigTextStyle**: 显示更多的文本信息, 例如显示一条消息或者公告的全部内容。
- **BigPictureStyle**: 显示彩色的大图片
- **InboxStyle**: 提供了一个选项列表, 样子有点像 Gmail 中的收件箱

但是, 你不止限于使用这些样式。**Notification.Style** 就是一个接口, 应用程序可以实现它来显示任意更加适合你的需求的自定义扩展布局。

除了样式, Android 4.1 还为扩展通知添加了内联动作。也就是说可以在 **window shade view** 上为用户添加多个动作选项, 而不是只有在用户单击整个通知项时返回一个回调 **Intent**。这些选项会在扩展 **view** 的顶部到底部依次排列。程序清单 6-2 演示了如何修改之前的示例来添加一个 **BigTextStyle** 样式的扩展通知, 结果如图 6-3 所示。

程序清单 6-2 **BigTextStyle** 风格的通知

```
//使用发送的时间创建通知
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(NotificationActivity.this);

builder.setSmallIcon(R.drawable.icon)
    .setTicker("Something Happened")
    .setWhen(System.currentTimeMillis())
    .setAutoCancel(true)
    .setDefaults(Notification.DEFAULT_SOUND)
    .setContentTitle("We're Finished!")
    .setContentText("Click Here!")
    .setContentIntent(contentIntent);

//添加一些自定义动作
builder.addAction(android.R.id.drawable.ic_menu_call, "Call Back",
    contentIntent);
builder.addAction(android.R.id.drawable.ic_menu_recent_history,
    "Call History", contentIntent);

//应用一个扩展样式
NotificationCompat.BigTextStyle expandedStyle =
    new NotificationCompat.BigTextStyle(builder);
expandedStyle.bigText("Here is some additional text to be displayed when"
    + " the notification is in expanded mode. "
    + " I can fit so much more content into this giant view!");

Notification note = expandedStyle.build();

//发送通知
manager.notify(NOTE_ID, note);
```

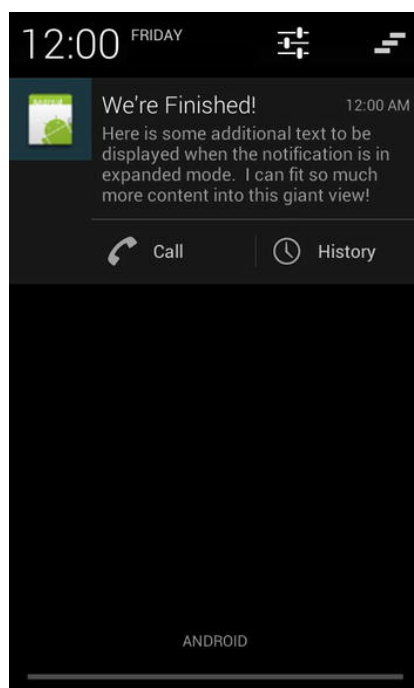


图 6-3 window shade 中的 BigTextStyle 风格

可以通过 builder 的 `addAction()` 方法关联自定义动作。这里演示了如何将动作布局加入到整个 view 中。本例中，每个动作的响应结果一样的，但也可以为每个动作都关联一个 `PendingIntent`，这样每个动作就可以有不同的响应结果了。

之前示例唯一修改的地方就是将已经创建的 `Builder` 对象封装到 `BigTextStyle` 中并做了自定义。本例中，另外还设置了 `bigText()` 以在扩展模式下显示文本。然后，通过 `style` 的 `build()` 而不是 `builder` 的 `build()` 方法创建通知。

下面查看一下程序清单 6-3 和图 6-4 所示的 `BigPictureStyle`。

程序清单 6-3 BigPictureStyle 样式的通知

```
//使用发送的时间创建通知
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(NotificationActivity.this);

builder.setSmallIcon(R.drawable.icon)
    .setTicker("Something Happened")
    .setWhen(System.currentTimeMillis())
    .setAutoCancel(true)
    .setDefaults(Notification.DEFAULT_SOUND)
    .setContentTitle("We're Finished!")
    .setContentText("Click Here!")
    .setContentIntent(contentIntent);

//添加一些自定义动作
builder.addAction(android.R.id.drawable.ic_menu_compass,
```

```

        "View Location", contentIntent);

//应用扩展样式
NotificationCompat.BigPictureStyle expandedStyle =
    new NotificationCompat.BigPictureStyle(builder);
expandedStyle.bigPicture(
    BitmapFactory.decodeResource(getResources(), R.drawable.icon) );

Notification note = expandedStyle.build();

//发送通知
manager.notify(NOTE_ID, note);

```

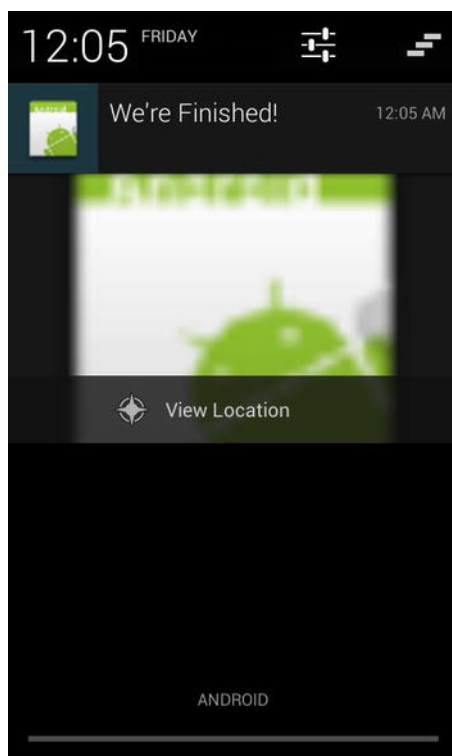


图 6-4 window shade 中的 BigPictureStyle

这段代码除了使用 `bigPicture()` 传递了一个要显示的全彩色 `Bitmap` 图片,剩下的几乎和 `BigTextStyle` 是一样的。最后,看一下程序清单 6-4 和图 6-5 所示的 `InboxStyle`。

程序清单 6-4 InboxStyle 样式的通知

```

//使用发送的时间创建通知
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(NotificationActivity.this);

builder.setSmallIcon(R.drawable.icon)
    .setTicker("Something Happened")
    .setWhen(System.currentTimeMillis())

```

```

        .setAutoCancel(true)
        .setDefaults(Notification.DEFAULT_SOUND)
        .setContentTitle("We're Finished!")
        .setContentText("Click Here!")
        .setContentIntent(contentIntent);

//应用扩展样式
NotificationCompat.InboxStyle expandedStyle =
    new NotificationCompat.InboxStyle(builder);
expandedStyle.setSummaryText("4 New Tasks");
expandedStyle.addLine("Make Dinner");
expandedStyle.addLine("Call Mom");
expandedStyle.addLine("Call Wife First");
expandedStyle.addLine("Pick up Kids");

Notification note = expandedStyle.build();

//发送通知
manager.notify(NOTE_ID, note);

```

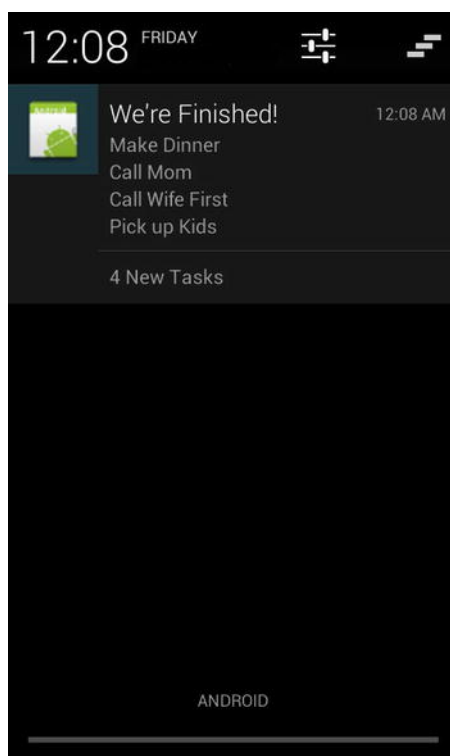


图 6-5 window shade 中的 InboxStyle

在 Notification.InboxStyle 样式下，可以通过 addLine()方法向列表中添加多个选项。我们还使用 setSummaryText()方法将所有的选项进行了分类总结，在之前的样式中该方法也是可以使用的。

与以前一样，我们使用了支持库中的 NotificationCompat 类，可以在运行 API Level 4

设备上的应用程序上调用这些方法。如果你的应用程序运行的最小平台是 Android 4.1，直接使用原生的 `Notification.Builder` 即可。

本例中，我们看到了支持库的强大之处。调用了 API Level 16 才有的方法，实际上是支持库在底层进行了版本检查，对于某一平台不支持的方法会忽略，则不需要建立判断分支来使用新 API。

因此，当在 Android 4.0 或更早版本的设备上运行这段同样的代码时，由于我们没有用到新特性，因此只会出现简单的传统通知。

注意：

支持库的一个强大之处就是可以在运行低版本的 Android 设备上使用高版本才有的 API，而且不必对你的代码使用判断分支。

6.2 创建定时和周期任务

6.2.1 问题

应用程序想要定时执行一个操作，例如定时更新 UI。

6.2.2 解决方案

(API Level 1)

使用 `Handler` 进行定时操作。`Handler` 可以有效地在某一确定时间点或者延迟特定时间后执行某个操作。

6.2.3 实现机制

让我们看一个示例 Activity，它通过一个 `TextView` 显示了当前的时间。参见程序清单 6-5。

程序清单 6-5 使用 `Handler` 进行更新的 Activity

```
public class TimingActivity extends Activity {

    TextView mClock;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mClock = new TextView(this);
        setContentView(mClock);
    }

    private Handler mHandler = new Handler();
```

```

private Runnable timerTask = new Runnable() {
    @Override
    public void run() {
        Calendar now = Calendar.getInstance();
        mClock.setText(String.format("%02d:%02d:%02d",
            now.get(Calendar.HOUR),
            now.get(Calendar.MINUTE),
            now.get(Calendar.SECOND)));
        //1 秒后进行下一次更新
        mHandler.postDelayed(timerTask,1000);
    }
};

@Override
public void onResume() {
    super.onResume();
    mHandler.post(timerTask);
}

@Override
public void onPause() {
    super.onPause();
    mHandler.removeCallbacks(timerTask);
}
}

```

这里，将读取当前时间并将更新 UI 的操作封装到一个名为 timerTask 的 Runnable 中，该 Runnable 会通过一个已经创建好了的 Handler 触发。当 Activity 可见时，Handler.post() 会立即执行 timerTask 中的操作。TextView 更新后，timerTask 最后会调用 Handler.postDelayed()，让 Handler 可以在 1 秒后进行再一次的更新操作。

只要 Activity 不被打扰，这个循环会一直进行，即每秒更新一次 UI。而当 Activity 处于暂停状态时(用户关闭了 Activity 或者其他事情改变 Activity 的状态)，Handler.removeCallbacks() 方法会移除所有待处理的操作，这样 timerTask 在 Activity 再次可见之前就不会被执行了。

提示：

在这个示例中，由于 Handler 是在主线程创建的，因此我们更新 UI 的操作是安全的。Handler 发出的操作都是在同一个线程中执行的。

6.3 定时执行周期任务

6.3.1 问题

应用程序需要注册执行一个周期任务，例如检查服务器更新或者提醒用户。

6.3.2 解决方案

(API Level 1)

使用 `AlarmManager` 管理和执行你的任务。`AlarmManager` 可用于计划未来的单次或重复操作,甚至在应用程序没有运行时也可以执行任务。`AlarmManager` 通过发出 `PendingIntent` 来发出警告。这个 `Intent` 可以在警告被触发时执行各种系统组件,例如 `Activity`、`BroadcastReceiver` 或者 `Service`。

注意,这种方式更加适用于应用程序没有运行但依然需要执行某些操作的场景。与应用程序运行时所使用的简单定时操作相比,`AlarmManager` 需要消耗更多的资源。后一种情况最好是使用 `Handler` 的 `postAtTime()`和 `postDelayed()`方法。

6.3.3 实现机制

让我们看一下在常规情况下,如何使用 `AlarmManager` 去触发一个广播。参见程序清单 6-6 到程序清单 6-8。

程序清单 6-6 要触发的广播

```
public class AlarmReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //可以执行一个有趣的操作,我们这里只是显示当前时间
        Calendar now = Calendar.getInstance();
        DateFormat formatter = SimpleDateFormat.getTimeInstance();
        Toast.makeText(context, formatter.format(now.getTime()),
            Toast.LENGTH_SHORT).show();
    }
}
```

提醒:

`BroadcastReceiver`(本例中为 `AlarmReceiver`)必须要在 `manifest` 中通过 `<receiver>` 标签声明,这样 `AlarmManager` 才可以被触发。请确保在 `<application>` 标签中包含如下类似的代码:

```
<application>
...
    <receiver android:name=".AlarmReceiver"></receiver>
</application>
```

程序清单 6-7 res/layout/main.xml

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

```

        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Start Alarm"
    />
    <Button
        android:id="@+id/stop"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Cancel Alarm"
    />
</LinearLayout>

```

程序清单 6-8 注册/解除注册 Alarm 的 Activity

```

public class AlarmActivity extends Activity implements View.OnClickListener {

    private PendingIntent mAlarmIntent;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //为两个按钮关联监听器
        findViewById(R.id.start).setOnClickListener(this);
        findViewById(R.id.stop).setOnClickListener(this);
        //创建提醒触发器
        Intent launchIntent = new Intent(this, AlarmReceiver.class);
        mAlarmIntent = PendingIntent.getBroadcast(this, 0, launchIntent, 0);
    }

    @Override
    public void onClick(View v) {
        AlarmManager manager =
            (AlarmManager) getSystemService(Context.ALARM_SERVICE);
        long interval = 5*1000; //5 seconds

        switch(v.getId()) {
            case R.id.start:
                Toast.makeText(this, "Scheduled", Toast.LENGTH_SHORT).show();
                manager.setRepeating(AlarmManager.ELAPSED_REALTIME,
                    SystemClock.elapsedRealtime()+interval,
                    interval,
                    mAlarmIntent);
                break;
            case R.id.stop:
                Toast.makeText(this, "Canceled", Toast.LENGTH_SHORT).show();
                manager.cancel(mAlarmIntent);
                break;
            default:

```

```

        break;
    }
}
}

```

这个示例中，我们提供了一个非常简单的 `BroadcastReceiver`，触发它时会通过 `Toast` 简单地显示当前的时间。这个 `receiver` 必须要在应用程序的 `manifest` 中通过 `<receiver>` 标签进行注册。否则，应用程序外部的 `AlarmManager` 将无法知道该如何触发它。这个示例的 `Activity` 有两个按钮：一个用于启动定时提醒，另一个则是取消所设置的提醒。

设置和取消提醒是通过 `PendingIntent` 实现的。我们创建一个直接指向应用程序的 `BroadcastReceiver` 的 `Intent`，然后用 `getBroadcast()` (因为我们创建了指向 `BroadcastReceiver` 的引用) 从中创建一个 `PendingIntent`。

提醒：

`PendingIntent` 也有创造器方法——`getActivity()` 和 `getService()`。但在创建 `PendingIntent` 时，要确保它能正确地指向要触发的应用程序组件。

按下启动按钮，`Activity` 会用 `AlarmManager.setRepeating()` 注册一个会反复发出的提醒。除了 `PendingIntent`，该方法还有一些参数用于设置触发提醒的时间。第一个参数定义了提醒的类型，包括时间单位和在设备休眠时是否发出提醒。在这个示例中，我们选择了 `ELAPSED_REALTIME`，表示从设备最近一次启动后算起的时间(毫秒)。此外，还有以下 3 种模式：

- `ELAPSED_REALTIME_WAKEUP`
 - 根据经过的时间触发提醒，如果设备处于休眠状态，会将设备激活。
- `RTC`
 - 根据 UTC 时间触发警告
- `RTC_WAKEUP`
 - 根据 UTC 时间触发警告，如果设备处于休眠状态，会将设备激活

接下来的两个参数分别是首次触发提醒的时间和重复发出提醒的时间间隔。因为选择的提醒类型是 `ELAPSED_REALTIME`，所以首次触发提醒的时间必须是相对某个时间经过的时间，`SystemClock.elapsedRealtime()` 可以设置这种格式的当前时间。

这个示例中的触发器会在按钮按下 5 秒钟后触发提醒，然后每 5 秒都会触发一次。每 5 秒钟，屏幕上都会通过 `Toast` 形式显示当前的时间，即使应用程序不在前台运行也是如此。当用户打开 `Activity`，按下停止按钮时，所有与 `PendingIntent` 匹配的提醒都会被取消，`Toast` 也会停止。

定时提醒示例

如果要在指定的时间发出提醒怎么办？(比如每天上午 9 时)还是要用 `AlarmManager` 再加上一些不同的参数即可，参见程序清单 6-9。

程序清单 6-9 定时提醒

```

long oneDay = 24*3600*1000; //24 hours
long firstTime;

//创建一个 Calendar(默认为当前日期)
//设置提醒时间为 09:00:00
Calendar startTime = Calendar.getInstance();
startTime.set(Calendar.HOUR_OF_DAY, 9);
startTime.set(Calendar.MINUTE, 0);
startTime.set(Calendar.SECOND, 0);

//获取当前时间
Calendar now = Calendar.getInstance();

if(now.before(startTime)) {
    //现在还没有到上午 9 时, 从今天算起
    firstTime = startTime.getTimeInMillis();
} else {
    //明天上午 9 时
    startTime.add(Calendar.DATE, 1);
    firstTime = startTime.getTimeInMillis();
}

//设置提醒
manager.setRepeating(AlarmManager.RTC_WAKEUP,
                    firstTime,
                    oneDay,
                    mAlarmIntent);

```

这个示例是根据真实时间触发提醒。首先是判断接下来的上午 9 时是今天还是要等到明天, 然后将返回的值作为第一次触发提醒的时间。然后将 24 小时换算成毫秒, 再将提醒触发的时间间隔传给 `manager.setRepeating()` 方法, 这样就能实现每天定时触发提醒。

重点:

设备重启后警告会消失。如果设备关闭, 然后再启动, 之前注册的提醒必须重新安排。

6.4 创建粘性操作

6.4.1 问题

应用程序即使在其终止时也可以执行一个或多个后台操作。

6.4.2 解决方案

(API Level 3)

创建一个 `IntentService` 来处理这项工作。`IntentService` 是 Android 标准 `Service` 实现的

封装类，是在无交互地执行后台任务的关键组件。`IntentService` 会将要执行的任务(用 `Intent` 表示)放到队列中，然后逐个处理每个请求，全部处理完成后会终止自己。

`IntentService` 会在后台创建一个执行任务所需的工作线程，所以就不需要使用 `AsyncTask` 或者 `Java Thread` 来确保后台操作的正确执行。

这个范例会演示使用 `IntentService` 创建一个后台操作的中央管理器。本例中，外部应用程序可以调用 `Context.startService()`来引用这个管理器。管理器会将所有收到的请求添加到队列中，然后通过调用 `onHandleIntent()`逐个处理。

6.4.3 实现机制

下面来看看如何构建一个简单的 `IntentService` 实现来处理一系列的后台操作。参见程序清单 6-10。

程序清单 6-10 用于处理后台操作的 `IntentService`

```
public class OperationsManager extends IntentService {

    public static final String ACTION_EVENT = "ACTION_EVENT";
    public static final String ACTION_WARNING = "ACTION_WARNING";
    public static final String ACTION_ERROR = "ACTION_ERROR";
    public static final String EXTRA_NAME = "eventName";

    private static final String LOGTAG = "EventLogger";

    private IntentFilter matcher;

    public OperationsManager() {
        super("OperationsManager");
        //创建一个过滤器来匹配收到的请求
        matcher = new IntentFilter();
        matcher.addAction(ACTION_EVENT);
        matcher.addAction(ACTION_WARNING);
        matcher.addAction(ACTION_ERROR);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //检查是否是有效的请求
        if(!matcher.matchAction(intent.getAction())) {
            Toast.makeText(this, "OperationsManager: Invalid Request",
                Toast.LENGTH_SHORT).show();
            return;
        }
        //直接在这个方法中处理每个请求。不需要创建其他的线程
        if(TextUtils.equals(intent.getAction(), ACTION_EVENT)) {
            logEvent(intent.getStringExtra(EXTRA_NAME));
        }
        if(TextUtils.equals(intent.getAction(), ACTION_WARNING)) {
```

```

        logWarning(intent.getStringExtra(EXTRA_NAME));
    }
    if(TextUtils.equals(intent.getAction(), ACTION_ERROR)) {
        logError(intent.getStringExtra(EXTRA_NAME));
    }
}

private void logEvent(String name) {
    try {
        //通过休眠来模拟一个长时间的网络操作
        Thread.sleep(5000);
        Log.i(LOGTAG, name);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

private void logWarning(String name) {
    try {
        //通过休眠来模拟一个长时间的网络操作
        Thread.sleep(5000);
        Log.w(LOGTAG, name);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

private void logError(String name) {
    try {
        //通过休眠来模拟一个长时间的网络操作
        Thread.sleep(5000);
        Log.e(LOGTAG, name);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}

```

IntentService 并没有默认的构造函数(除了无参的构造函数),因此在自定义 **IntentService** 实现时必须实现一个构造函数,该构造函数会调用父类的构造函数并传入一个服务名。这个名称在技术层面并不重要,只是用作调试而已,**Android** 会用这个名称命名创建新的工作线程。

所有的请求都是通过 **onHandleIntent()**方法处理的。这个方法是由工作线程调用的,所以可以在这里进行所有的工作,不必创建新线程或是执行其他操作。当 **onHandleIntent()**返回时,就是通知 **IntentService** 开始处理队列中的下一个请求了。

这个示例还提供了 3 个日志操作,可以在请求 **Intent** 中用不同的动作字符串调用。为了演示这些操作,每个操作都会用相应的日志级别(INFO、WARNING 或 ERROR)将消息写到设备上。注意,消息的内容是以请求 **Intent** 的附加信息的形式传递的。用 **Intent** 的 **Data**

和附加字段保存日志操作的参数，留下 Action 字段来定义操作的类型。

这个示例服务中还有一个 `IntentFilter`，用来判断请求是否有效。在创建服务时，所有有效请求都被添加到这个过滤器中，这样就能通过调用 `IntentFilter.matchAction()` 判断所收到的请求是否包含可处理的操作。

程序清单 6-11 和 6-12 是一个调用该服务的示例 Activity。

程序清单 6-11 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.sticky"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="3" />

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

提醒：

AndroidManifest.xml 中的 package 属性必须与应用程序的包名一致。com.examples.sticky 是我们示例中使用的包名称。

注意：

因为 `IntentService` 是作为 `Service` 被调用的，所以必须通过 `<service>` 标签在应用程序的 manifest 文件中进行声明。

程序清单 6-12 调用 IntentService 的 Activity

```
public class ReportActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        logEvent("CREATE");
    }

    @Override
    public void onStart() {
```

```

        super.onStart();
        logEvent("START");
    }

    @Override
    public void onResume() {
        super.onResume();
        logEvent("RESUME");
    }

    @Override
    public void onPause() {
        super.onPause();
        logWarning("PAUSE");
    }

    @Override
    public void onStop() {
        super.onStop();
        logWarning("STOP");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        logWarning("DESTROY");
    }

    private void logEvent(String event) {
        Intent intent = new Intent(this, OperationsManager.class);
        intent.setAction(OperationsManager.ACTION_EVENT);
        intent.putExtra(OperationsManager.EXTRA_NAME, event);

        startService(intent);
    }

    private void logWarning(String event) {
        Intent intent = new Intent(this, OperationsManager.class);
        intent.setAction(OperationsManager.ACTION_WARNING);
        intent.putExtra(OperationsManager.EXTRA_NAME, event);

        startService(intent);
    }
}

```

这个 Activity 中的东西不多，只是通过设备的日志发送事件而不是用户界面。但是，它帮助演示了之前所创建的服务中的队列机制。当 Activity 可见时，它会调用其所有的生命周期方法，对日志服务发出 3 个请求。每处理完一个请求，就会向日志输出一行内容，然后该服务转向处理下一个请求。

提示:

可以通过 SDK 的 logcat 工具看到这些日志语句。设备或者模拟器的 logcat 输出可以通过大多数的开发环境(包括 Eclipse)或者命令行(`adb logcat`)查看。

还要注意当服务处理完 3 个请求后,系统会向日志发出一个通知:服务已停止。只有在需要 `IntentService` 完成任务时,它才会处于内存中。这个服务是一个非常好的特性,使其成为整个系统中一个好“公民”。

按下 HOME 或者 BACK 键会触发更多的生命周期方法,向服务发送更多请求, `Pause/Stop/Destroy` 会调用服务中的一个单独操作, 它们的消息会作为警告记录到日志中。只要将请求 `Intent` 的动作字符串改成其他值就可以设置日志的类型。

注意,即使应用程序已经不可见(设置是打开了另一个应用),消息也会继续发送到日志中。这就是 Android 系统中 `Service` 组件的强大之处,这类操作受系统保护,无论用户做什么,这些操作都会执行到底。

潜在的不足

在每个操作方法中都有一个 5 秒的延迟,用来模拟实际请求访问远程 API 等操作所需的时间。在运行这个示例时,还演示了 `IntentService` 是如何在一个工作线程中逐个处理队列中的多个请求。这个示例中的请求来自应用的各个生命周期方法,但在日志中还是每 5 秒记录一条信息。这是因为在处理完当前的请求之前(也就是 `onHandleIntent()` 返回之前), `IntentService` 是不会开始处理下一个请求的。

如果应用程序实现并发操作,就需要用线程池来创建自定义的 `Service` 实现。Android 作为开源系统的一大魅力就是可以直接看到 `IntentService` 的源代码,然后以此为基础开发自己的实现,节约开发时间,减少代码量。

6.5 长时间运行的后台操作

6.5.1 问题

应用程序中需要有一个组件一直运行在后台,用于执行一些操作或者监控特定事件。

6.5.2 解决方案

(API Level 1)

可以将这个组件放到一个服务中。服务就是后台运行的组件,应用程序可以启动服务,然后让其在后台长期运行。跟后台的其他进程相比,服务的优先级更高,不会因为内存少而被终止。

可以为了某些操作而启动或停止服务,而并不一定要将其与其他的组件(如 `Activity`)连接起来,如果应用程序必须要与服务进行交互,服务也提供了相应的绑定接口来传递数据。在这里的示例中,服务的启动和终止都是由系统根据所请求的绑定来决定的。

实现服务时要注意的关键是确保它的用户友好性。无期限的操作应该由用户主动启动。整个应用程序中应该要有地方能使用户控制这类服务的启动和禁用。

6.5.3 实现机制

程序清单 6-12 是一个持久运行的示例服务，用来跟踪和输出用户一定时间段内的位置。

程序清单 6-13 持久运行的跟踪服务

```
public class TrackerService extends Service implements LocationListener {

    private static final String LOGTAG = "TrackerService";

    private LocationManager manager;
    private ArrayList<Location> storedLocations;

    private boolean isTracking = false;

    /* 服务的创建方法 */
    @Override
    public void onCreate() {
        manager = (LocationManager) getSystemService(LOCATION_SERVICE);
        storedLocations = new ArrayList<Location>();
        Log.i(LOGTAG, "Tracking Service Running...");
    }

    @Override
    public void onDestroy() {
        manager.removeUpdates(this);
        Log.i(LOGTAG, "Tracking Service Stopped...");
    }

    public void startTracking() {
        if(!manager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            return;
        }
        Toast.makeText(this, "Starting Tracker", Toast.LENGTH_SHORT).show();
        manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            30000, 0, this);

        isTracking = true;
    }

    public void stopTracking() {
        Toast.makeText(this, "Stopping Tracker", Toast.LENGTH_SHORT).show();
        manager.removeUpdates(this);
        isTracking = false;
    }
}
```

```

    public boolean isTracking() {
        return isTracking;
    }

    /* 服务访问方法 */
    public class TrackerBinder extends Binder {
        TrackerService getService() {
            return TrackerService.this;
        }
    }

    private final IBinder binder = new TrackerBinder();

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    public int getLocationsCount() {
        return storedLocations.size();
    }

    public ArrayList<Location> getLocations() {
        return storedLocations;
    }

    /* LocationListener 方法 */
    @Override
    public void onLocationChanged(Location location) {
        Log.i("TrackerService", "Adding new location");
        storedLocations.add(location);
    }

    @Override
    public void onProviderDisabled(String provider) { }

    @Override
    public void onProviderEnabled(String provider) { }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) { }
}

```

这个服务会监控和跟踪从 `LocationManager` 收到的更新信息。当服务被创建时，它创建了一个空的位置信息列表并开始等待跟踪。对于外部组件，如 `Activity`，则可以调用 `startTracking()` 和 `stopTracking()` 来启用和禁用位置更新的过程。此外，这个服务还提供了能访问该服务所记录的位置列表的方法。

因为这个服务需要直接与其他 `Activity` 或组件进行交互，所以需要提供一个 `Binder` 接口。在服务需要跨进程通信时，`Binder` 会变得很复杂。但在这个示例中，所有的东西都在

同一个进程中，所以所创建的 Binder 也很简单。这里的 Binder 是在 `getService` 方法中创建的，它向调用者返回服务实例本身。稍后我们在从 Activity 的角度仔细看看这个 Service。

当服务启动跟踪时，它会注册 `LocationManager` 的更新信息，并将其收到的每个更新都保存到位置列表中。注意，调用 `requestLocationUpdates()` 方法的时间间隔最少为 30 秒。因为该服务会长时间运行，较大的更新时间间隔可以让 GPS(和电池)“稍作休息”。

现在来看一个让用户访问该服务的 Activity，参见程序清单 6-14 到程序清单 6-16。

程序清单 6-14 AndroidManifest.xml

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

提醒:

服务必须要在应用程序中通过 `<service>` 标签进行声明，这样 Android 系统才能知道如何以及到哪里调用它。同样，因为需要用到 GPS，所以这个示例中还必须添加 `android.permission.ACCESS_FINE_LOCATION` 权限。

程序清单 6-15 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/enable"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Start Tracking"
    />
    <Button
        android:id="@+id/disable">
```

```

        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Stop Tracking"
    />
    <TextView
        android:id="@+id/status"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>

```

程序清单 6-16 与 Service 进行交互 Activity

```

public class ServiceActivity extends Activity implements View.OnClickListener {

    Button enableButton, disableButton;
    TextView statusView;
    TrackerService trackerService;
    Intent serviceIntent;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        enableButton = (Button)findViewById(R.id.enable);
        enableButton.setOnClickListener(this);
        disableButton = (Button)findViewById(R.id.disable);
        disableButton.setOnClickListener(this);
        statusView = (TextView)findViewById(R.id.status);

        serviceIntent = new Intent(this, TrackerService.class);
    }

    @Override
    public void onResume() {
        super.onResume();
        //启动服务使其持久化, 无视绑定
        startService(serviceIntent);
        //绑定服务
        bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
    }

    @Override
    public void onPause() {
        super.onPause();
        if(!trackerService.isTracking()) {
            //停止服务, 解除绑定后永久移除
            stopService(serviceIntent);
        }
        //解除服务绑定
        unbindService(serviceConnection);
    }
}

```

```

    }

    @Override
    public void onClick(View v) {
        switch(v.getId()) {
            case R.id.enable:
                trackerService.startTracking();
                break;
            case R.id.disable:
                trackerService.stopTracking();
                break;
            default:
                break;
        }
        updateStatus();
    }

    private void updateStatus() {
        if(trackerService.isTracking()) {
            statusView.setText(
                String.format("Tracking enabled. %d locations
                               logged.",trackerService.getLocationsCount()));
        } else {
            statusView.setText("Tracking not currently enabled.");
        }
    }

    private ServiceConnection serviceConnection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder
            service) {
            trackerService = ((TrackerService.TrackerBinder)service).getService();
            updateStatus();
        }

        public void onServiceDisconnected(ComponentName className) {
            trackerService = null;
        }
    };
}

```

图 6-6 显示了一个有两个按钮的 Activity，允许用户启用和禁用位置跟踪，并将当前服务的状态通过文本的形式显示出来。

当 Activity 可见时，会被绑定到 TrackerService 上。这是通过 ServiceConnection 接口实现的，该接口在绑定和解绑操作完成后都会提供相应的回调方法。当服务和 Activity 绑定后，就可以直接调用服务所提供的所有公共方法。

然而，只使用绑定还不能让服务长时间运行。仅仅通过 Binder 接口访问服务会使 Service 随着 Activity 的生命周期而自动创建或销毁。在这个示例中，我们希望即使 Activity 没有运行，Service 也能继续运行。为了实现这个目的，在将 Service 与 Activity 绑定之前，

先用 `startService` 启动 `Service`。给正在运行中的 `Service` 发送启动命令不会造成任何问题，所以在 `onResume()` 方法中这么做也完全可行。

现在即使 `Activity` 取消绑定后，`Service` 还会在内存中运行。这个示例中 `onPause()` 方法中会检查用户是否正在启动跟踪，如果没有就会停止 `Service`。这样在不需要跟踪时，`Service` 也会停止。在没有任务时，`Service` 就可从内存中永久移除。

运行这个示例，按下 `Start Tracking` 按钮就会启动持久运行的服务和 `LocationManager`。这时也许会离开应用程序但服务会一直运行，而且会记录从 GPS 收到的位置信息。当用户再次回到应用程序时，用户会看到一直运行着的服务和当前保存的位置数量。按下 `Stop Tracking` 按钮会终止该进程，然后在用户退出应用程序后立即结束 `Service`。

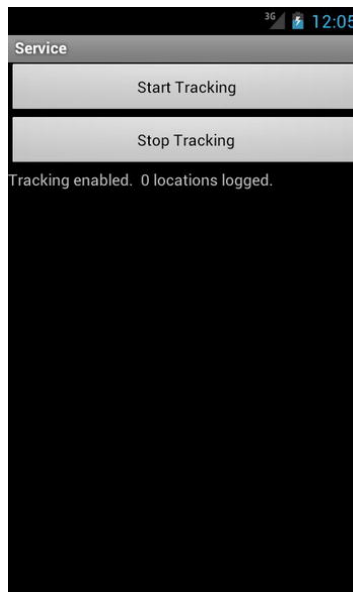


图 6-6 ServiceActivity 布局

6.6 启动其他应用程序

6.6.1 问题

你的应用程序需要的功能设备上其他应用程序已经具备了。为了避免功能的重复开发，需要能够启动其他应用程序来执行该功能。

6.6.2 解决方案

(API Level 1)

使用隐式的 `Intent` 告知系统你想要做的事情，系统会判断是否有满足需要的应用程序。通常，开发者会使用显式的 `Intent` 启动其他的 `Activity` 或 `Service`，如：

```
Intent intent = new Intent(this, NewActivity.class);
startActivity(intent);
```

声明了你要启动的特定组件后，`Intent` 的目的也就很明确了。同样，我们还可以使用 `action`、`category`、`data` 和 `type` 定义一个 `Intent` 来实现更加隐性的需求。

当你的应用程序通过这种方式启动一个外部应用程序时，通常它们会处在同一个 `Android` 任务栈中，因此当操作完成(或者用户退出外部应用程序)后，用户还会回到你的应用程序中。这样就确保了用户体验的无缝衔接，从用户的角度来看，这些应用程序是浑然一体的。

6.6.3 实现机制

定义 Intent 时需要设定哪些信息并不明确, 没有统一的标准, 有可能两个应用程序所能提供的服务(例如读取 PDF 文件)是一样的, 但监听所收到的 Intent 的过滤器却不尽相同。你需要向系统(或用户)提供足够的信息来选择最合适的应用程序完成所需的任务。

定义隐式 Intent 的核心是 action, 可以通过构造函数传递, 也可以用 Intent.setAction() 设置。这个值告诉 Android 你要做什么, 比如查看某些内容、发送信息、做选择或其他目的。这里要根据具体情况设置, 而且通常不同的组合可以得到相同的结果。让我们看几个实用的示例。

1. 读取 PDF 文件

几乎每台 Android 设备上都预装了 PDF 阅读器, 在 Android 电子市场里面也有无数的 PDF 阅读器, 但在核心 SDK 中并没有提供显示 PDF 文档的组件。所以, 在应用程序中内嵌显示 PDF 的功能是比较麻烦的, 而且没有必要。

相反, 程序清单 6-17 则演示了如果找到并启动其他应用程序来查看 PDF 文件。

程序清单 6-17 查看 PDF 的方法

```
private void viewPdf(Uri file) {
    Intent intent;
    intent = new Intent(Intent.ACTION_VIEW);
    intent.setDataAndType(file, "application/pdf");
    try {
        startActivity(intent);
    } catch (ActivityNotFoundException e) {
        //如果没有可用的应用程序, 就提示下载一个
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("No Application Found");
        builder.setMessage("We could not find an application to view PDFs."
            + " Would you like to download one from Android Market?");
        builder.setPositiveButton("Yes, Please",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    Intent marketIntent = new Intent(Intent.ACTION_VIEW);
                    marketIntent.setData(
                        Uri.parse("market://details?id=com.adobe.reader"));
                    startActivity(marketIntent);
                }
            });
        builder.setNegativeButton("No, Thanks", null);
        builder.create().show();
    }
}
```

这个示例会找到的最合适的应用程序并打开设备上任意的本地 PDF 文件(内部和外部

存储器)。如果没有找到可以查看 PDF 应用程序, 会提示用户到 Google Play 上下载一个。

创建并构造 `Intent` 时使用的是通用的 `Intent.ACTION_VIEW` 动作字符串, 它会告诉系统我们想要查看该 `Intent` 中所指定类型的数据文件。而数据文件本身及其 MIME 类型则可以让系统了解要查看的数据类型。

提示:

使用 `Intent.setData()` 和 `Intent.setType()` 会相互清除对方所设置的值。如果想要同时两个都设置, 就跟示例一样使用 `Intent.setDataAndType()` 即可。

如果 `startActivity()` 执行失败得到 `ActivityNotFoundException` 异常, 意味着在用户的设备上并没有安装可以查看 PDF 的应用程序。出现这种情况时, 为了可以让用户体验更好, 我们将问题告知用户并提示用户可以去 Market 上下载一个。如果用户按下 Yes, 就会发送另一个 `Intent` 请求直接打开 Google Play 上 Adobe Reader 的下载页, 用户可以免费下载。在下一个范例中我们会讨论这个 `Intent` 中所使用的 Uri 策略。

注意, 这个示例方法中传入了一个指向本地文件的 Uri 参数。下面的示例演示了如何获取内部存储器文件的 Uri。

```
String filename = NAME_OF_YOUR_FILE;
File internalFile = getFilePath(filename);
Uri internal = Uri.fromFile(internalFile);
```

`getFilePath()` 是通过 `Context` 调用的, 因此如果这段代码不是写在 `Activity` 的, 你还必须引用一个 `Context` 对象才能调用它。下面是如何构建指向外部存储器文件的 Uri。

```
String filename = NAME_OF_YOUR_FILE;
File externalFile = new File(Environment.getExternalStorageDirectory(),
    filename);
Uri external = Uri.fromFile(externalFile);
```

这个示例也适用于其他类型的文档, 只需要修改 `Intent` 的 MIME 类型即可。

与好友分享内容

在开发者中很流行的一个功能就是让用户可以通过电子邮件、短信或主流的社交网络与朋友分享应用程序中的内容。所有的 Android 设备都有电子邮件和短信应用程序, 大部分想通过社交网络(例如 Facebook 或 Twitter)分享内容的用户也会在自己的设备上安装相应的移动应用程序。

恰好, 这个任务也可以用隐式 `Intent` 完成, 因为大多数此类应用程序都会以某种方式响应 `Intent.ACTION_SEND` 动作。程序清单 6-18 演示了如何通过单个 `Intent` 请求发送各种内容。

程序清单 6-18 分享 Intent

```
private void shareContent(String update) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
```

```

        intent.putExtra(Intent.EXTRA_TEXT, update);
        startActivity(Intent.createChooser(intent, "Share..."));
    }

```

这里，我们告诉系统要以 `Intent` 的附加信息的形式发送一段文本。这是一种常见的请求，我们希望能有多个应用程序处理这种请求。默认情况下，`Android` 会弹出一个应用程序列表供用户选择。此外，部分设备还提供了一个选择框，让用户选择完成这种操作的默认应用程序，以后就不会再弹出这个列表了。

我们可以对这个过程施加更多的控制，但同时希望能给用户多种选择。要实现这个目的，就不要直接把 `Intent` 传递给 `startActivity()`，而是传递给 `Intent.createChooser()`，这样就能自定义标题，确保向用户显示供选择的列表。

用户做出选择后，指定的应用程序就会启动，加载 `EXTRA_TEXT` 中的信息，做好分享内容的准备！

2. ShareActionProvider

(API Level 14)

从 `Android 4.0` 开始，引入了一个新的 `ShareActionProvider` 控件，使用一种更加通用的机制来帮助应用程序分享内容。它被添加到选项菜单中从而在 `ActionBar` 或者更多菜单上显示。它还有一个附加的功能，默认情况下，它会根据用户的使用习惯将分享选项排序。也就是说，频繁使用的选项会排到列表的最上方。

在菜单中实现 `ShareActionProvider` 非常简单，与创建分享 `Intent` 比起来，只需要很少的几行代码即可。程序清单 6-19 显示了如何将 `ShareActionProvider` 关联到一个菜单项上。

程序清单 6-19 res/menu/options.xml

```

<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_share"
        android:showAsAction="ifRoom"
        android:title="Share"
        android:actionProviderClass="android.widget.ShareActionProvider"/>
</menu>

```

注意：

如果你的 `Menu` 不是在 `XML` 中定义的，可以在 `Java` 代码中调用 `setActionProvider()` 关联 `ShareActionProvider`。程序清单 6-20 显示了如何在 `Activity` 中将分享 `Intent` 关联到 `ShareActionProvider` 控件上。

程序清单 6-20 提供分享 `Intent`

```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //填充菜单
    getMenuInflater().inflate(R.menu.options, menu);

    //找到选项并设置分享 Intent
}

```

```
MenuItem item = menu.findItem(R.id.menu_share);
ShareActionProvider provider = (ShareActionProvider) item.getActionProvider();

Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, update);
provider.setShareIntent(intent);

return true;
}
```

好了！provider 处理了所有的用户交互，因此你的应用程序甚至不需要处理用户的 MenuItem 选择事件。

6.7 启动系统应用程序

6.7.1 问题

你的应用程序需要的功能设备上的系统应用程序已经具备了。为了避免功能的重复开发，需要能够启动系统应用程序来执行该功能。

6.7.2 解决方案

(API Level 1)

用隐式 Intent 告诉系统你所需要的应用程序。每个系统应用程序都有相应的 Uri。可以将其以数据的形式插入到隐式 Intent 中来启动你所需要的应用程序。

当你的应用程序通过这种方式启动启动一个系统应用程序时，通常它们会处在同一个 Android 任务栈中，因此当操作完成(或者用户退出系统应用程序)后，用户还会回到你的应用程序中。这样就确保了用户体验的无缝衔接，从用户的角度来看，这些应用程序是浑然一体的。

6.7.3 实现机制

下面几个示例都构建了在不同状态下启动系统应用程序的 Intent。在创建了 Intent 之后，就可以将 Intent 发送给 startActivity()来启动相应的应用程序了。

1. 浏览器

浏览器应用程序可以用来显示网页或执行网页搜索。

要显示网页，可以构建并加载下面的 Intent。

```
Intent pageIntent = new Intent();
pageIntent.setAction(Intent.ACTION_VIEW);
pageIntent.setData(Uri.parse("http://WEB_ADDRESS_TO_VIEW"));
```

```
startActivity(pageIntent);
```

用你自己想要显示的网页替换 **data** 字段中的 Uri。在浏览器中加载网页搜索则需要构建和加载下面的 Intent。

```
Intent searchIntent = new Intent();
searchIntent.setAction(Intent.ACTION_WEB_SEARCH);
searchIntent.putExtra(SearchManager.QUERY, STRING_TO_SEARCH);

startActivity(searchIntent);
```

用你自己的搜索词替换掉 Intent 中的附加信息数据。

2. 电话拨号器

拨号器应用程序用于拨打电话，加载拨号器的 Intent 如下：

```
Intent dialIntent = new Intent();
dialIntent.setAction(Intent.ACTION_DIAL);
dialIntent.setData(Uri.Parse( "tel:8885551234" ));

startActivity(dialIntent);
```

将数据中的 Uri 直接替换成你要拨打的电话号码。

注意：

这个动作只显示拨号器，并不会真正将这个号码拨出去。如果要直接拨打电话，需要用 Intent.ACTION_CALL。但在绝大多数情况下，Google 并不鼓励这么做。使用 Intent.ACTION_CALL 还需要在应用程序的 manifest 文件中声明 android.permission.CALL_PHONE 权限。

3. 地图

设备上的地图应用程序可以用于显示所处位置或是提供两点间的路径。如果知道要显示位置的经纬度，可以使用下面的 Intent：

```
Intent mapIntent = new Intent();
mapIntent.setAction(Intent.ACTION_VIEW);
mapIntent.setData(Uri.parse( "geo:latitude,longitude" ));

startActivity(mapIntent);
```

设置你当前位置的经纬度。例如，这个 Uri：“geo:37.422,122.084”，这会在地图上显示 Google 总部的地址。如果你知道要显示的位置地址，可以使用下面的 Intent：

```
Intent mapIntent = new Intent();
mapIntent.setAction(Intent.ACTION_VIEW);
mapIntent.setData(Uri.parse( "geo:0,0?q=ADDRESS" ));
```

```
startActivity(mapIntent);
```

这会在地图上加入要显示位置的地址。例如这个 Uri: `geo:0,0?q=1600 Amphitheatre Parkway, Mountain View, CA 94043`。

提示:

地图应用程序中的 Uri 可以用 “+” 号代替地址中的空格。如果在编码带空格的字符串时遇到问题, 可以试试用 “+” 代替。

如果要显示两个位置间的路径, 用下面的 Intent:

```
Intent mapIntent = new Intent();
mapIntent.setAction(Intent.ACTION_VIEW);
mapIntent.setData(Uri.parse("http://maps.google.com/maps?saddr=lat,lng&daddr=lat,lng"));

startActivity(mapIntent);
```

这会在地图上加入起点和终点的地址。

也可以仅用一个地址打开地图应用程序。例如, 这个 Uri: `http://maps.google.com/maps?&daddr=37.422,122.084`, 这会打开地图应用程序, 显示终点位置, 让用户自行输入他的起点地址。

4. 电子邮件

用下面的 Intent 可以撰写模式启动设备上的各种电子邮件应用程序:

```
Intent mailIntent = new Intent();
mailIntent.setAction(Intent.ACTION_SEND);
mailIntent.setType("message/rfc822");
mailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"recipient@gmail.com"});
mailIntent.putExtra(Intent.EXTRA_CC, new String[] {"carbon@gmail.com"});
mailIntent.putExtra(Intent.EXTRA_BCC, new String[] {"blind@gmail.com"});
mailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email Subject");
mailIntent.putExtra(Intent.EXTRA_TEXT, "Body Text");
mailIntent.putExtra(Intent.EXTRA_STREAM, URI_TO_FILE);

startActivity(mailIntent);
```

如果是空白邮件, 那就只需要设置 action 和 type。其他(信息就是电子邮件的各个组成部分了。注意, 就算是只有一个接收者, EXTRA_EMAIL(收件人)、EXTRA_CC(抄送)和 EXTRA_BCC(密件抄送)也都必须是字符串数组。邮件附件则是用 Intent 的 EXTRA_STREAM 设置。这个参数的值应该指向要发送的本地文件的 Uri。

如果需要给电子邮件添加多个附件, 就需要稍作改动, 如下所示:

```
Intent mailIntent = new Intent();
mailIntent.setAction(Intent.ACTION_SEND_MULTIPLE);
mailIntent.setType("message/rfc822");
mailIntent.putExtra(Intent.EXTRA_EMAIL, new String[] {"recipient@gmail.com"});
```

```

mailIntent.putExtra(Intent.EXTRA_CC, new String[] { "carbon@gmail.com" });
mailIntent.putExtra(Intent.EXTRA_BCC, new String[] { "blind@gmail.com" });
mailIntent.putExtra(Intent.EXTRA_SUBJECT, "Email Subject");
mailIntent.putExtra(Intent.EXTRA_TEXT, "Body Text");

ArrayList<Uri> files = new ArrayList<Uri>();
files.add(Uri.toFirstFile());
files.add(Uri.toSecondFile());
//.....根据具体情况复制 add() 以便添加所有需要的附件
mailIntent.putParcelableArrayListExtra(Intent.EXTRA_STREAM, files);

startActivity(mailIntent);

```

注意, 现在 Intent 的动作字符串是 ACTION_SEND_MULTIPLE。其中主要的字段跟前面的代码都是一样的, 只有附件是以 EXTRA_STREAM 添加的。这个示例会创建一份指向要添加文件的 Uri 列表, 然后用 putParcelableArrayListExtra() 将其添加到 Intent 中。

用户的设备上经常会有多个应用程序可以处理这种内容, 所以通常在将上述 Intent 传递给 startActivity() 之前会用 Intent.createChooser() 处理。

5. SMS(短消息)

应用程序可以调用消息应用程序编写新的短信, Intent 如下所示:

```

Intent smsIntent = new Intent();
smsIntent.setAction(Intent.ACTION_VIEW);
smsIntent.setType("vnd.android-dir/mms-sms");
smsIntent.putExtra("address", "8885551234");
smsIntent.putExtra("sms_body", "Body Text");

startActivity(smsIntent);

```

与编写电子邮件一样, 必须要设置 action 和 type 才能启动该应用程序。Address 和 sms_body 附加信息则用于设置短信的收件人(address)和消息正文(smsbody)。

注意, 在 Android 框架中, 这些关键字并没有完全确定。这意味着在将来的 Android 框架中, 这些关键字可能会发生变化。不过, 在撰写本书时, 这些关键字在所有版本的 Android 上都能正常使用。

6. 联系人选择器

应用程序可以启动默认的联系选择器让用户从联系人数据库中选择联系人, Intent 如下所示:

```

static final int REQUEST_PICK = 100;

Intent pickIntent = new Intent();
pickIntent.setAction(Intent.ACTION_PICK);
pickIntent.setData(Uri.toContactTable());

startActivityForResult(pickIntent, REQUEST_PICK);

```

这个 Intent 需要在数据字段中放入联系人表的 CONTENT_URI 字段。因为在 API Level 5(Android 2.0)之后联系人 API 有重大变化, 所以如果你的应用程序要同时支持 2.0 前后的各个版本, 就要注意这个 Uri 可能会有所不同。

举个例子, 在 2.0 之前的版本中从联系人列表中选择联系人的 Uri 是: `android.provider.Contacts.People.CONTENT_URI`。但在 2.0 及以后的版本中, 要获取同样的数据就应该使用: `android.provider.ContactsContract.Contacts.CONTENT_URI`。

注意查阅与你要访问的联系人数据有关的 API 文档。这个 Activity 还要返回用户选项所代表的 Uri, 因此需要使用 `startActivityForResult()`。

7. Google Play

可以在应用程序中启动 Google Play 显示某个应用程序的具体信息, 或是搜索某个关键字。加载某应用程序的 Android 市场页面的 Intent 如下所示:

```
Intent marketIntent = new Intent();
marketIntent.setAction(Intent.ACTION_VIEW);
marketIntent.setData(Uri.parse( "market://details?id=PACKAGE_NAME_HERE" ));

startActivity(marketIntent);
```

将要显示的应用程序的包名(例如 “com.adobe.reader”)插入即可。如果要打开市场, 进行搜索, 使用下面的 Intent:

```
Intent marketIntent = new Intent();
marketIntent.setAction(Intent.ACTION_VIEW);
marketIntent.setData(Uri.parse( "market://search?q=SEARCH_QUERY" ));

startActivity(marketIntent);
```

插入要搜索的查询字符串。搜索查询有以下 3 种形式。

- `q=<简单的文本字符串>`
 - 在市场中做关键字搜索
- `q=pname:<包名>`
 - 搜索包的名称, 只会返回完全匹配的结果
- `q=pub:<开发者名字>`
 - 搜索开发者的姓名, 只会返回完全匹配的结果

6.8 让其他应用程序启动你的应用程序

6.8.1 问题

你的应用程序能够很好地完成某项工作, 所以想提供一种方式让设备上的其他应用程序也可以运行你的应用程序。

6.8.2 解决方案

(API Level 1)

在应用程序的 Activity 或 Service 上创建一个 IntentFilter，然后在文档中说明访问该 Activity 或 Service 所需的 Intent 中的 action、category、data/type 和附加信息。前面说过，Intent 的 action、category、data/type 用来判断是否能匹配应用程序的需求。其他信息或是可选参数都应该作为 Intent 的附加信息传递。

6.8.3 实现机制

假设你已经创建了一个应用程序，该应用程序包含有一个 Activity 在播放视频的同时在屏幕的顶端显示视频标题的跑马灯。你想让其他的应用程序通过传递必要的参数调用这个应用程序来播放视频，这就需要定义一个合理的 Intent 结构，然后在应用程序的 manifest 文件中给这个 Activity 加上一个 IntentFilter 来判断是否匹配。

这个假想的 Activity 需要两个数据：

- (1) 视频文件的 Uri，可以是本地视频或者远程视频
- (2) 视频标题的字符串

如果应用程序只能播放某种视频类型。我们可以定义一个通用的 Intent，然后在 Data 中限定为应用程序所能处理的视频类型。程序清单 6-21 演示了如何在 manifest 文件中定义 Activity，以便过滤 Intent 的方法。

程序清单 6-21 AndroidManifest.xml 中带数据类型过滤器的<activity>元素

```
<activity android:name=".PlayerActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="video/h264" />
    </intent-filter>
</activity>
```

这个过滤器会匹配所有 H.264 格式或根据 Uri 判断为 H.264 格式的视频片段。外部应用程序调用该 Activity 播放视频的代码如下：

```
Uri videoFile = A_URI_OF_VIDEO_CONTENT;
Intent playIntent = new Intent(Intent.ACTION_VIEW);
playIntent.setDataAndType(videoFile, "video/h264");
playIntent.putExtra(Intent.EXTRA_TITLE, "My Video");
startActivity(playIntent);
```

在某些情况下，无论其视频是什么格式，外部应用程序都可以直接调用该播放器。那么用起来会更方便些。我们可以给 Intent 定义一个自己的动作字符串。让 manifest 文件中关联 Activity 的过滤器只接受这个自定义的动作字符串。参见程序清单 6-22。

程序清单 6-22 AndroidManifest.xml 中带自定义动作过滤器的<activity>元素

```

<activity android:name=".PlayerActivity">
    <intent-filter>
        <action android:name="com.examples.myplayer.PLAY" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>

```

外部应用程序可以通过下面的代码调用这个 Activity 来播放视频:

```

Uri videoFile = A_URI_OF_VIDEO_CONTENT;
Intent playIntent = new Intent( "com.examples.myplayer.PLAY" );
playIntent.setData(videoFile);
playIntent.putExtra(Intent.EXTRA_TITLE, "My Video" );
startActivity(playIntent);

```

处理一次成功的调用

无论 Intent 是如何匹配 Activity 的, 只要 Activity 启动后, 都需要检查所收到的 Intent 中 Activity 完成工作所需的两份数据, 参见程序清单 6-18。

程序清单 6-23 Activity 检查 Intent

```

public class PlayerActivity extends Activity {

    public static final String ACTION_PLAY = "com.examples.myplayer.PLAY";

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

        //检查启动该 Activity 的 Intent
        Intent incoming = getIntent();
        //从 data 字段中获取 URI
        Uri videoUri = incoming.getData();
        //获取可选的视频标题字符串
        String title;
        if(incoming.hasExtra(Intent.EXTRA_TITLE)) {
            title = incoming.getStringExtra(Intent.EXTRA_TITLE);
        } else {
            title = "";
        }

        /* 显示视频和标题 */

    }

    /* 其他 Activity 代码 */

}

```

Activity 启动后，调用 `Activity getIntent()` 可以得到传入的 `Intent`。视频内容的 `Uri` 是通过 `Intent` 的 `data` 字段传递的，可以通过 `Intent.getData()` 获得该 `Uri`。`Intent` 中的视频标题是个可选字段，我们会检查附加信息的 `bundle`，看看是否传入了这个字段；如果传了，就从 `Intent` 中得到这个值。

注意，这个示例中的 `PlayerActivity` 自定义了一个动作字符串常量，但我们上面使用 `Intent` 启动 Activity 的过程中并没有引用这个常量。这是因为外部应用程序无法访问我们在应用程序中定义的公共常量。

正因为如此，要尽量重用 SDK 中已有的 `Intent` key 而不要自定义新的常量。在这个示例中，我们选择了标准的 `Intent.EXTRA_TITLE` 来定义可选的附加信息，而非自己为这个值新建一个 key。

6.9 与联系人交互

6.9.1 问题

应用程序需要使用 Android 提供的 `ContentProvider` 对数据库中的用户联系人信息直接进行添加、查看、修改或删除。

6.9.2 解决方案

(API Level 5)

用 `ContactsContract` 提供的接口来访问数据。`ContactsContract` 包含一系列的 `ContentProvider` API，用于将系统中多个用户账户的联系人信息整合到一个数据库中，生成大量可以访问和修改的 `Uri`、表和字段。

`Contact` 是一个分层结构，分为三层：`Contacts`、`RawContacts` 和 `Data`：

- 一个 `Contact` 代表一个人，是 Android 认定为同一个人的所有 `RawContacts` 的集合。
- `RawContacts` 表示某个类型的账号数据集。这个账号可以是用户的电子邮件地址、Facebook 账号或其他账号。
- `Data` 是 `RawContacts` 中的一条信息，例如电子邮件地址、电话号码、邮寄地址等。

完整的 API 有太多的组合和选项，受篇幅限制，这里不可能一一详述，请参阅 SDK 文档。接下来会介绍如何实现联系人数据的基本查询和简单的修改。

6.9.3 实现机制

Android `Contacts` API 面对的是一个由多个表和多种链接(join)组成的复杂数据库，但在应用程序中访问这些数据的方法与访问其他 `SQLite` 数据库的方法并没有本质的区别。

1. 列举/查看联系人信息

让我们看一个示例，其中的 Activity 会列出通讯录数据库中所有的联系人，在选中某

个联系人后则会显示他的详细信息，参见程序清单 6-24。

重点：

要在你的应用程序中显示来自 Contacts API 的信息，需要在应用程序的 manifest 文件中声明 android.permission.READ_CONTACTS 权限。

程序清单 6-24 显示联系人信息的 Activity

```
public class ContactsActivity extends ListActivity implements
    AdapterView.OnItemClickListener {

    Cursor mContacts;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //返回所有联系人，通过名字排序
        String[] projection = new String[] { ContactsContract.Contacts._ID,
            ContactsContract.Contacts.DISPLAY_NAME };
        mContacts = managedQuery(ContactsContract.Contacts.CONTENT_URI,
            projection, null, null, ContactsContract.Contacts.DISPLAY_NAME);

        //在 ListView 中显示所有联系人
        SimpleCursorAdapter mAdapter = new SimpleCursorAdapter(this,
            android.R.layout.simple_list_item_1, mContacts,
            new String[] { ContactsContract.Contacts.DISPLAY_NAME },
            new int[] { android.R.id.text1 });
        setListAdapter(mAdapter);
        //监听条目的选择
        getListView().setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position,
        long id) {
        if (mContacts.moveToPosition(position)) {
            int selectedId = mContacts.getInt(0); // _ID column
            //从 email 表中获取电子邮件数据
            Cursor email = getContentResolver().query(
                CommonDataKinds.Email.CONTENT_URI,
                new String[] { CommonDataKinds.Email.DATA },
                ContactsContract.Data.CONTACT_ID + " = " + selectedId, null,
                null);
            //从 phone 表中获取手机号数据
            Cursor phone = getContentResolver().query(
                CommonDataKinds.Phone.CONTENT_URI,
                new String[] { CommonDataKinds.Phone.NUMBER },
                ContactsContract.Data.CONTACT_ID + " = " + selectedId, null,
                null);
        }
    }
}
```

```

//从 address 表中获取邮寄地址
Cursor address = getContentResolver().query(
    CommonDataKinds.StructuredPostal.CONTENT_URI,
    new String[]
{ CommonDataKinds.StructuredPostal.FORMATTED_ADDRESS },
    ContactsContract.Data.CONTACT_ID + " = " + selectedId, null,
    null);

//构建对话框信息
StringBuilder sb = new StringBuilder();
sb.append(email.getCount() + " Emails\n");
if (email.moveToFirst()) {
    do {
        sb.append("Email: " + email.getString(0));
        sb.append('\n');
    } while (email.moveToNext());
    sb.append('\n');
}
sb.append(phone.getCount() + " Phone Numbers\n");
if (phone.moveToFirst()) {
    do {
        sb.append("Phone: " + phone.getString(0));
        sb.append('\n');
    } while (phone.moveToNext());
    sb.append('\n');
}
sb.append(address.getCount() + " Addresses\n");
if (address.moveToFirst()) {
    do {
        sb.append("Address:\n" + address.getString(0));
    } while (address.moveToNext());
    sb.append('\n');
}

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle(mContacts.getString(1)); // Display name
builder.setMessage(sb.toString());
builder.setPositiveButton("OK", null);
builder.create().show();

//关闭临时光标
email.close();
phone.close();
address.close();
    }
}
}

```

从以上代码可以看出, 在这些 API 中引用表和字段的代码非常繁琐。在这个示例中, 所有的 Uri、表和字段的引用都是 ContactsContract 的内部类。在使用 Contacts API 时, 一

定要注意检查是否使用了正确的类,所有不是来自 `ContactsContract` 的 `Contacts` 类都不宜使用且不兼容。

在创建 `Activity` 时,我们会用参数 `Contacts.CONTENT_URI` 调用 `Activity.managedQuery()` 查询核心的 `Contacts` 表,只请求将 `Cursor` 封装到 `ListAdapter` 中所需的列。得到的 `Cursor` 以列表的形式显示在用户界面上。这个示例利用 `ListActivity` 的特性,用 `ListView` 作为内容视图,这样我们就不必管理这些组件了。

此刻用户可能会上下滑动设备上的联系人列表,然后单击其中某个联系人查看他的详细信息。在用户选中某个联系人后,该联系人的 `_ID` 值会被记录下来,应用程序会根据这个值到 `ContactsContract.Data` 表中获取更详细的信息。要注意的是,一个联系人的数据是分布在各个表中的(电子邮件在 `email` 表中,电话号码在 `phone` 表中等),所以需要多次查询才能获得完整信息。

每个 `CommonDataKinds` 都有一个唯一的 `CONTENT_URI` 用于查询时引用该表,同时还有一组字段别名用于请求数据。这些数据表中的每一行数据都通过 `Data.CONTACT_ID` 与一个联系人关联,所以每个 `Cursor` 只返回与这个值匹配的行。

在收集了用户选中的联系人的所有数据后,我们将结果显示在对话框中供用户查看。因为这些表中的数据是多个数据来源的集合,所以多个请求返回多个结果的情况并不少见。对于每个 `cursor`,会显示返回结果的数量,然后附上其所包含的每个值。在所有数据都组合好之后,就会创建一个对话框将这些信息显示给用户。

最后及时关闭所有的临时 `cursor` 和非托管 `cursor`。

2. 运行应用程序

在有多个账户的设备上运行这个应用程序时,你会发现所显示的列表过长,其长度远超过设备上预装的 `Contacts` 应用程序中联系人列表的长度。`Contacts` API 可以保存可能被用户隐藏或是内部使用的分组内容。例如, `Gmail` 经常会将所收到的电子邮件的地址保存下来,即使这个电子邮件地址跟任何联系人没有关联也会保存,这是为了以后使用方便。

在下一个示例中,我们将演示如何过滤这个列表,但目前的 `Contacts` 表中存放的数据实在太多了。

3. 修改/添加联系人

下面查看一个示例 `Activity`,操作某个具体的联系人,参见程序清单 6-25。

重点:

要在应用程序中使用 `Contacts` API,需要在应用程序的 `manifest` 文件中声明 `android.permission.READ_CONTACTS` 和 `android.permission.WRITE_CONTACTS` 权限。

程序清单 6-25 写入 `Contacts` API 的 `Activity`

```
public class ContactsEditActivity extends ListActivity implements
    AdapterView.OnItemClickListener, DialogInterface.OnClickListener {

    private static final String TEST_EMAIL = "test@email.com";
```

```

private Cursor mContacts, mEmail;
private int selectedContactId;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //返回所有的联系人, 按照名字排序
    String[] projection = new String[] { ContactsContract.Contacts._ID,
        ContactsContract.Contacts.DISPLAY_NAME };
    //只列出用户可见的联系人
    mContacts = managedQuery(ContactsContract.Contacts.CONTENT_URI,
        projection,
        ContactsContract.Contacts.IN_VISIBLE_GROUP+" = 1",
        null, ContactsContract.Contacts.DISPLAY_NAME);
    //在 ListView 中显示所有联系人
    SimpleCursorAdapter mAdapter = new SimpleCursorAdapter(this,
        android.R.layout.simple_list_item_1, mContacts,
        new String[] { ContactsContract.Contacts.DISPLAY_NAME },
        new int[] { android.R.id.text1 });

    setListAdapter(mAdapter);
    //监听条目的选择
    getListView().setOnItemClickListener(this);
}

@Override
public void onItemClick(AdapterView<?> parent, View v, int position,
    long id) {
    if (mContacts.moveToPosition(position)) {
        selectedContactId = mContacts.getInt(0); // _ID column
        //从 email 表中获取电子邮件数据
        String[] projection = new String[] { ContactsContract.Data._ID,
            ContactsContract.CommonDataKinds.Email.DATA };
        mEmail = getContentResolver().query(
            ContactsContract.CommonDataKinds.Email.CONTENT_URI,
            projection,
            ContactsContract.Data.CONTACT_ID+" = "+selectedContactId,
            null, null);
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Email Addresses");
        builder.setCursor(mEmail, this,
            ContactsContract.CommonDataKinds.Email.DATA);
        builder.setPositiveButton("Add", this);
        builder.setNegativeButton("Cancel", null);
        builder.create().show();
    }
}

@Override

```

```

public void onClick(DialogInterface dialog, int which) {
    //数据只能与一个 RAW contact 关联, 获取第一个匹配的行 id
    Cursor raw = getContentResolver().query(
        ContactsContract.RawContacts.CONTENT_URI,
        new String[] { ContactsContract.Contacts._ID },
        ContactsContract.Data.CONTACT_ID+" = "+selectedContactId,
        null, null);
    if(!raw.moveToFirst()) {
        return;
    }

    int rawContactId = raw.getInt(0);
    ContentValues values = new ContentValues();
    switch(which) {
    case DialogInterface.BUTTON_POSITIVE:
        //用户想要添加一个新的电子邮件地址
        values.put(ContactsContract.CommonDataKinds.Email.
            RAW_CONTACT_ID,
            rawContactId);
        values.put(ContactsContract.Data.MIMETYPE,
            ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE);
        values.put(ContactsContract.CommonDataKinds.Email.DATA, TEST_EMAIL);
        values.put(ContactsContract.CommonDataKinds.Email.TYPE,
            ContactsContract.CommonDataKinds.Email.TYPE_OTHER);
        getContentResolver().insert(ContactsContract.Data.
            CONTENT_URI, values);
        break;
    default:
        //用户想要编辑选中信息
        values.put(ContactsContract.CommonDataKinds.Email.DATA,
            TEST_EMAIL);
        values.put(ContactsContract.CommonDataKinds.Email.TYPE,
            ContactsContract.CommonDataKinds.Email.TYPE_OTHER);
        getContentResolver().update(ContactsContract.Data.
            CONTENT_URI, values,
            ContactsContract.Data._ID+" = "+mEmail.getInt(0), null);
        break;
    }

    //不再需要 email cursor
    mEmail.close();
}
}

```

这个示例中, 我们还是和以前一样, 会查询 `Contacts` 数据库中的所有数据。这次, 只有一个选择条件:

```
ContactsContract.Contacts.IN_VISIBLE_GROUP+" = 1"
```

这个限制条件是，返回用户在联系人用户界面中可以看到的内容。这会缩短(在个别情况下，会大大缩短)在 `Activity` 中显示的联系人列表的长度，使其中所显示的内容更接近 `Contacts` 应用程序中显示的联系人列表。

当用户从列表中选中某个联系人，就会显示一个对话框列出该联系人的所有信息。如果选中列表中的某个地址，就可以进行编辑；如果单击 `add` 按钮，就可以添加新的电子邮件地址。为了保持示例的简洁性，我们没有提供输入电子邮件地址的界面，而只是插入一个常量作为新记录或是更新选中的记录。

诸如电子邮件地址一类的数据元素只能与一个 `RawContact` 关联。因此，在添加新的电子邮件地址时，必须获取用户所选择的高层 `Contacts` 的一个 `RawContact` 的 ID。在这个示例中，我们并不关心到底是哪个 `RawContact`，所以我们就取了第一个匹配上的 `RawContact` 的 ID。只有在插入操作时才需要这个值，因为如果是更新已有的电子邮件记录，那么所需的行 ID 就已经保存在表中了。

还要注意，`CommonDataKinds` 中提供的作为别名以读取数据的 `Uri` 不能用来修改和更新数据。插入和更新必须要使用 `ContactsContract.Data` `Uri`。这意味着还必须设定一个元数据，即 `MIMETYPE`。如果没有设置所插入数据的 `MIMETYPE` 字段，接下来的查询就不会将其识别为联系人的电子邮件地址。

4. 聚合实例

因为这个示例是用同样的数据添加或编辑电子邮件地址，所以从这个示例中可以实时地看到 `Android` 的聚合操作。在运行这个示例应用程序时，你可能会注意到，在添加同样的电子邮件地址或将已有的记录修改为与其他电子邮件相同时，`Android` 会思考现在这个联系人和之前的联系人是不是同一个。在这个示例应用程序中，随着对核心联系人表的更新，你会发现有些联系人消失了，因为这些联系人会与其他的相关联系人聚合在一起。

注意：

在 `Android` 模拟器中还没有完全实现联系人聚合。要看到上述的效果，需要在真机上运行这段代码。

5. 维护引用

`Android` 的 `Contacts` API 引入了一个对某些应用程序很重要的新概念。因为这个聚合过程，表示一个联系人的行 ID 的含义就显得不那么明确了；当一个联系人跟另一个联系人聚合在一起之后，他会获得一个新的 ID。

如果应用程序需要一个指向特定联系人的长时间不变的引用，建议你使用 `ContactsContract.Contacts.LOOKUP_KEY`，而不要使用行 ID。在使用这个可以查询联系人时，还需要一个 `ContactsContract.Contacts.CONTENT_LOOKUP_URI` 这个形式特殊的 `Uri`。用这些值查询联系人数据库，可以使应用程序避免自动聚合过程所带来的麻烦。

6.10 设备媒体文件选择器

6.10.1 问题

应用程序需要导入一个用户可以选择的媒体选择器(视频、视频或图片)来显示和播放。

6.10.2 解决方案

(API Level 1)

使用 `Intent.ACTION_GET_CONTENT` 隐式 `Intent` 唤起一个系统媒体选择器界面。发送这个 `Intent` 时加入感兴趣媒体的内容类型就会唤起一个选择器界面，用户可以通过选择器选择其中的一个文件，这个 `Intent` 的结果会包含一个指向所选媒体文件的 `Uri`。

6.10.3 实现机制

让我们看一下在一个示例 `Activity` 中使用这项技术的示例。参见程序清单 6-26 和 6-27。

程序清单 6-26 `res/layout/main.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/imageButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Images"
    />
    <Button
        android:id="@+id/videoButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Video"
    />
    <Button
        android:id="@+id/audioButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Audio"
    />
</LinearLayout>
```

程序清单 6-27 使用媒体选择器的 Activity

```

public class MediaActivity extends Activity implements View.OnClickListener {

    private static final int REQUEST_AUDIO = 1;
    private static final int REQUEST_VIDEO = 2;
    private static final int REQUEST_IMAGE = 3;

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

        Button images = (Button)findViewById(R.id.imageButton);
        images.setOnClickListener(this);
        Button videos = (Button)findViewById(R.id.videoButton);
        videos.setOnClickListener(this);
        Button audio = (Button)findViewById(R.id.audioButton);
        audio.setOnClickListener(this);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
        data) {

        if(resultCode == Activity.RESULT_OK) {
            //返回的 Intent 中包含用户所选择的文件的 Uri
            Uri selectedContent = data.getData();

            if(requestCode == REQUEST_IMAGE) {
                //显示图片
            }
            if(requestCode == REQUEST_VIDEO) {
                //播放视频
            }
            if(requestCode == REQUEST_AUDIO) {
                //播放音频
            }
        }
    }

    @Override
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setAction(Intent.ACTION_GET_CONTENT);
        switch(v.getId()) {
            case R.id.imageButton:
                intent.setType("image/*");
                startActivityForResult(intent, REQUEST_IMAGE);
                return;
        }
    }
}

```

```

        case R.id.videoButton:
            intent.setType("video/*");
            startActivityForResult(intent, REQUEST_VIDEO);
            return;
        case R.id.audioButton:
            intent.setType("audio/*");
            startActivityForResult(intent, REQUEST_AUDIO);
            return;
        default:
            return;
    }
}
}

```

这个示例中有 3 个用户可以按下的按钮，每个按钮对应于一种特定媒体类型。当用户按下任意一个按钮时，就会向系统发送一个动作为 `Intent.ACTION_GET_CONTENT` 的 `Intent`，并启动相应的媒体选择器 `Activity`。如果用户通过选择器选择了一个有效的文件，这时会在返回的结果 `Intent` 中包含一个所选择的文件的 `Uri`，`Intent` 的状态为 `RESULT_OK`。如果用户取消或退出媒体选择器，结果 `Intent` 的状态将为 `RESULT_CANCELED`，而相应的 `data` 字段则为 `null`。

接收媒体文件的 `Uri` 后，应用程序可以自由地选择播放或者不播放所选择媒体内容。`MediaPlayer` 和 `VideoView` 这样的类可以直接通过 `Uri` 播放媒体内容，而对于图片则可以通过 `Uri.getPath()` 方法得到文件路径，然后传给 `BitmapFactory.decodeFile()`。

6.11 保存到 MediaStore

6.11.1 问题

应用程序想要保存媒体内容并把它加入到设备全局的 `MediaStore` 中，这样所有的应用程序都可以看到它。

6.11.2 解决方案

(API Level 1)

使用 `MediaStore` 的 `ContentProvider` 接口可以插入新的媒体文件。除了媒体文件本身，通过这个接口还可以插入用来标记媒体文件的各种元数据，例如标题、描述和创建时间等。`ContentProvider` 插入操作完成后会返回一个 `URI`，应用程序可以通过这个 `URI` 访问新的媒体文件。

6.11.3 实现机制

让我们看一个向 `MediaStore` 中插入图片或视频片段的示例。参见程序清单 6-28 和 6-29。

程序清单 6-28 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/imageButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Images"
    />
    <Button
        android:id="@+id/videoButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Video"
    />
</LinearLayout>

```

程序清单 6-29 保存数据到 MediaStore 的 Activity

```

public class StoreActivity extends Activity implements View.OnClickListener {

    private static final int REQUEST_CAPTURE = 100;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button images = (Button)findViewById(R.id.imageButton);
        images.setOnClickListener(this);
        Button videos = (Button)findViewById(R.id.videoButton);
        videos.setOnClickListener(this);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if(requestCode == REQUEST_CAPTURE && resultCode == Activity.RESULT_OK) {
            Toast.makeText(this, "All Done!", Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onClick(View v) {
        ContentValues values;
        Intent intent;
        Uri storeLocation;
    }
}

```

```

switch(v.getId()) {
case R.id.imageButton:
    //创建图片文件的元数据
    values = new ContentValues(2);
    values.put(MediaStore.Images.ImageColumns.DATE_TAKEN,
        System.currentTimeMillis());
    values.put(MediaStore.Images.ImageColumns.DESCRPTION,
        "Sample Image");
    //插入元数据并返回指向文件位置的 Uri
    storeLocation = getContentResolver().insert(
        MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
    //用获取的 Uri 作为媒体的存放目标
    intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, storeLocation);
    startActivityForResult(intent, REQUEST_CAPTURE);
    return;
case R.id.videoButton:
    //创建视频的元数据
    values = new ContentValues(2);
    values.put(MediaStore.Video.VideoColumns.ARTIST, "Yours
Truly");
    values.put(MediaStore.Video.VideoColumns.DESCRPTION,
        "Sample Video Clip");
    //插入元数据并返回指向文件位置的 Uri
    storeLocation = getContentResolver().insert(
        MediaStore.Video.Media.EXTERNAL_CONTENT_URI, values);
    //用获取的 Uri 作为媒体的存放目标
    intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, storeLocation);
    startActivityForResult(intent, REQUEST_CAPTURE);
    return;
default:
    return;
}
}
}

```

注意:

因为本例用到了摄像头，所以应该在真实设备上运行这个示例，这样才能看到效果。实际上，在 Android 2.2 及以后版本的模拟器上访问摄像头时存在一个 bug，运行这个示例会导致崩溃。早期的模拟器是没有问题的，但如果没有硬件支持的话，这个示例就失去意义了。

在这个示例中，当用户单击任何一个按钮时，媒体文件相关的元数据都会被插入到一个 ContentValues 实例中。有些元数据字段是图片文件和视频文件所共有的，如下：

- TITLE: 表示内容标题的字符串
- DESCRIPTION: 表示内容描述的字符串

- **DATE_TAKEN**: 描述媒体文件创建时间的整数值。`System.currentTimeMillis()`就表示当前的时间

接下来,通过相应的 `CONTENT_URI` 就可以将 `ContentValues` 插入到 `MediaStore` 中。注意,要在真正采集媒体文件之前插入元数据。插入成功后会返回一个完整的 `URI`,然后应用程序就可以将此 `URI` 作为媒体内容的存放目标。

在前面的示例中,我们使用了第 4 章录音和摄像中的简单方法,即要求系统应用程序来处理这个过程。回忆一下第 4 章的内容,录音和录像的 `Intent` 都可以传入一个标识文件保存位置的附加信息。这里我们传入的就是插入成功后所返回的 `Uri`。

当录音和摄像的 `Activity` 成功返回后,这个应用程序就结束了。外部应用程序将采集到的图片和视频保存到了 `MediaStore` 中。现在所有的应用程序都可以看到这些数据,包括系统的 `Gallery` 应用程序。

6.12 与日历的交互

6.12.1 问题

应用程序需要通过 `Android` 框架提供的 `ContentProvider` 在设备上添加、浏览、更改或者删除日历事件。

6.12.2 解决方案

(API Level 14)

使用 `CalendarContract` 读取/写入系统 `ContentProvider` 的事件数据。`CalendarContract` 提供了可以访问设备日历、事件、出席者、提醒等信息的 `API`。和 `ContactsContract` 非常类似,这个接口定义很多可以执行查询的数据。方法的使用方式和其他 `ContentProvider` 的使用方式是一样的。

6.12.3 实现机制

使用 `CalendarContract` 和使用 `ContactsContract` 非常相似,它们都提供了相应的 `Uri` 标识和在使用 `ContentResolver` 构建查询时所需的字段名称。程序清单 6-30 显示了获取和显示设备上日历事件列表的 `Activity`。

程序清单 6-30 显示设备上日历事件的 `Activity`

```
public class CalendarListActivity extends ListActivity implements
    LoaderManager.LoaderCallbacks<Cursor>, AdapterView.OnItemClickListener {
    private static final int LOADER_LIST = 100;

    SimpleCursorAdapter mAdapter;

    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    getLoaderManager().initLoader(LOADER_LIST, null, this);

    //在 ListView 中显示所有的日历事件
    mAdapter = new SimpleCursorAdapter(this,
        android.R.layout.simple_list_item_2, null,
        new String[] {
            CalendarContract.Calendars.CALENDAR_DISPLAY_NAME,
            CalendarContract.Calendars.ACCOUNT_NAME },
        new int[] {
            android.R.id.text1, android.R.id.text2 }, 0);
    setListAdapter(mAdapter);
    //监听条目的选择事件
    getListView().setOnItemClickListener(this);
}

@Override
public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
    Cursor c = mAdapter.getCursor();
    if (c != null && c.moveToPosition(position)) {
        Intent intent = new Intent(this, CalendarDetailActivity.class);
        //将选择的日历事件的_ID 和 TITLE 信息传递给下一个 Activity
        intent.putExtra(Intent.EXTRA_UID, c.getInt(0));
        intent.putExtra(Intent.EXTRA_TITLE, c.getString(1));
        startActivity(intent);
    }
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    //返回所有的日历事件，通过名字排序
    String[] projection = new String[] { CalendarContract.Calendars._ID,
        CalendarContract.Calendars.CALENDAR_DISPLAY_NAME,
        CalendarContract.Calendars.ACCOUNT_NAME };

    return new CursorLoader(this, CalendarContract.Calendars.CONTENT_URI,
        projection, null, null,
        CalendarContract.Calendars.CALENDAR_DISPLAY_NAME);
}

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    mAdapter.swapCursor(data);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}

```

```
    }
}
```

与之前联系人的示例相比，这里我们使用 Android 的 Loader 来查询数据并把得到的结果 Cursor 加载到列表中。这种方式相对于 `managedCursor()` 有很多优点，最大的好处就是所有的查询操作都是自动在后台线程中执行的，这样就可以保证 UI 线程可以及时响应。另外，Loader 这种方式是可以内部重用的，即请求相同数据的多个客户端实际上访问的是同一个由 LoaderManager 管理的 Loader。

使用 Loader 时，如果有新的日历数据可用，我们的 Activity 会接收很多回调方法。实际上，CursorLoader 也是像 ContentObserver 一样注册，在它所对应的数据集发生变化时甚至不需要重新加载就可以得到一个回调，该回调中包含有数据集新的 Cursor。但是对于 Calendar...

想要得到设备的日历事件列表，需要先使用 `Calendars.CONTENT_URI` 以及相应的字段(这里是记录的 ID、日历事件名称和账户名称)构建一个查询。查询完成后，会调用 `onLoadFinished()` 方法，这时会返回一个指向查询结果数据的新 Cursor，然后我们会把这个 Cursor 传递给 list adapter。当用户按下下一个日历事件条目后，就会初始化一个新的 Activity 来浏览这个日历事件。下一节会查看关于这个示例的更多细节信息。

1. 浏览/修改日历事件

程序清单展示了本例中另一个 Activity 的内容，该 Activity 中选中日历的所有事件列表。

程序清单 6-31 显示并修改日历事件的 Activity

```
public class CalendarDetailActivity extends ListActivity implements
    LoaderManager.LoaderCallbacks<Cursor>, AdapterView.OnItemClickListener,
    AdapterView.OnItemLongClickListener {
    private static final int LOADER_DETAIL = 101;

    SimpleCursorAdapter mAdapter;

    int mCalendarId;

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

        mCalendarId = getIntent().getIntExtra(Intent.EXTRA_UID, -1);

        String title = getIntent().getStringExtra(Intent.EXTRA_TITLE);
        setTitle(title);

        getLoaderManager().initLoader(LOADER_DETAIL, null, this);

        //在 ListView 中显示所有事件
        mAdapter = new SimpleCursorAdapter(this,
```

```

        android.R.layout.simple_list_item_2, null,
        new String[] {
            CalendarContract.Events.TITLE,
            CalendarContract.Events.EVENT_LOCATION },
        new int[] {
            android.R.id.text1, android.R.id.text2 }, 0);
setListAdapter(mAdapter);
//监听条目的选择事件
getListView().setOnItemClickListener(this);
getListView().setOnItemLongClickListener(this);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add("Add Event")
        .setIcon(android.R.drawable.ic_menu_add)
        .setShowAsAction(MenuItem.SHOW_AS_ACTION_ALWAYS);

    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    showAddEventDialog();
    return true;
}

//显示一个对话框来添加新事件
private void showAddEventDialog() {
    final EditText nameText = new EditText(this);
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("New Event");
    builder.setView(nameText);
    builder.setNegativeButton("Cancel", null);
    builder.setPositiveButton("Add Event",
        new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                addEvent(nameText.getText().toString());
            }
        });
    builder.show();
}

//使用一个特定的名称和当前时间(作为事件开始日期)向日历中添加一个事件
private void addEvent(String eventName) {
    long start = System.currentTimeMillis();
    //当前时间1小时后结束
    long end = start + (3600 * 1000);

```

```

        ContentValues cv = new ContentValues(5);
        cv.put(CalendarContract.Events.CALENDAR_ID, mCalendarId);
        cv.put(CalendarContract.Events.TITLE, eventName);
        cv.put(CalendarContract.Events.DESRIPTION,
                "Event created by Android Recipes");
        cv.put(CalendarContract.Events.EVENT_TIMEZONE,
                Time.getCurrentTimezone());
        cv.put(CalendarContract.Events.DTSTART, start);
        cv.put(CalendarContract.Events.DTEND, end);

        getContentResolver().insert(CalendarContract.Events.CONTENT_URI, cv);
    }

    //在日历中将选中的事件删除
    private void deleteEvent(int eventId) {
        String selection = CalendarContract.Events._ID + " = ?";
        String[] selectionArgs = { String.valueOf(eventId) };
        getContentResolver().delete(CalendarContract.Events.CONTENT_URI,
                selection, selectionArgs);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position,
        long id) {
        Cursor c = mAdapter.getCursor();
        if (c != null && c.moveToPosition(position)) {
            //选中后, 会显示一个包含日历事件详细信息的对话框
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
            HH:mm:ss");
            StringBuilder sb = new StringBuilder();

            sb.append("Location: " + c.getString(c.getColumnIndex(
                CalendarContract.Events.EVENT_LOCATION))
                + "\n\n");
            int startDateIndex = c.getColumnIndex(CalendarContract.Events.DTSTART);
            Date startDate = c.isNull(startDateIndex) ? null
                : new Date( Long.parseLong(c.getString(startDateIndex)) );
            if (startDate != null) {
                sb.append("Starts At: " + sdf.format(startDate) + "\n\n");
            }
            int endDateIndex = c.getColumnIndex(CalendarContract.Events.DTEND);
            Date endDate = c.isNull(endDateIndex) ? null
                : new Date( Long.parseLong(c.getString(endDateIndex)) );
            if (endDate != null) {
                sb.append("Ends At: " + sdf.format(endDate) + "\n\n");
            }
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle(
                c.getString(c.getColumnIndex(CalendarContract.
                    Events.TITLE)) );

```

```

        builder.setMessage(sb.toString());
        builder.setPositiveButton("OK", null);
        builder.show();
    }
}

@Override
public boolean onItemClick(AdapterView<?> parent, View view,
    int position, long id) {
    Cursor c = mAdapter.getCursor();
    if (c != null && c.moveToPosition(position)) {
        //用户长按时，会删除选中的日历事件
        final int eventId = c.getInt(
            c.getColumnIndex(CalendarContract.Events._ID));
        String eventName = c.getString(
            c.getColumnIndex(CalendarContract.Events.TITLE));
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("Delete Event");
        builder.setMessage(String.format(
            "Are you sure you want to delete %s?",
            TextUtils.isEmpty(eventName) ? "this event" : eventName));
        builder.setNegativeButton("Cancel", null);
        builder.setPositiveButton("Delete Event",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    deleteEvent(eventId);
                }
            });
        builder.show();
    }

    return true;
}

@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    //返回所有的日历，按照名称排序
    String[] projection = new String[] { CalendarContract.Events._ID,
        CalendarContract.Events.TITLE,
        CalendarContract.Events.DTSTART,
        CalendarContract.Events.DTEND,
        CalendarContract.Events.EVENT_LOCATION };
    String selection = CalendarContract.Events.CALENDAR_ID + " = ?";
    String[] selectionArgs = { String.valueOf(mCalendarId) };
    return new CursorLoader(this, CalendarContract.Events.CONTENT_URI,
        projection, selection, selectionArgs,
        CalendarContract.Events.DTSTART + " DESC");
}

```

```

@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    mAdapter.swapCursor(data);
}

@Override
public void onLoaderReset(Loader<Cursor> loader) {
    mAdapter.swapCursor(null);
}
}

```

可以看到，查询日历事件列表的代码和显示这些事件的代码非常相似，本例中是通过选择的日历 ID 作为 `selection` 参数来查询 `Events.CONTENT_URI` 的。这里，当用户单击一个日历事件后，就会向用户显示一个简单的对话框，而对话框中显示的则是日历事件的详细信息。此外，这个 `Activity` 还包含一些创建和删除日历事件的方法。

想要添加新的日历事件，可以在选项菜单中添加一个新的选项，如果设备只能看到一个选项，该选项会显示在 `ActionBar` 的最前面。按下该选项后，会弹出一个对话框让用户输入事件的名称。如果选择继续，就会创建一个含有日历事件必要信息的 `ContentValues` 对象。因为这个事件是一次性的，所以必须指定开始和结束时间以及有效的时区。我们还必须要指定日历的 ID，这样事件才可以和日历有效地关联起来。然后，通过 `ContentResolver` 将数据插入到 `Event` 表中。

想要删除一个事件，用户只需要在列表特定的条目上长按，然后在之后的对话框中确认删除即可。这种情况下，只需要选中事件的唯一记录 ID 并把它传入 `ContentResolver` 的 `selection` 字符串中。

你是否发现在这两个示例中，在添加/删除事件之后，我们并没有编写任何代码来更新 `Cursor` 或者 `CursorAdapter`？这就是 `Loader` 的强大之处！`CursorLoader` 会监控数据集，当数据集变化时，它会自动更新并赋予 `adapter` 一个新的 `Cursor`，该 `adapter` 则会更新界面的显示。

注意：

很多 `Loader` 都是从 Android 3.0(API Level 11)开始引入的，不过在 Android 支持库中也包含它们。因此，可以在 Android 1.6 中通过支持库使用它们。

6.13 执行日志代码

6.13.1 问题

为了调试或者测试，需要在代码中加入日志相关代码，并且需要在代码发布前移除这些日志代码。

6.13.2 解决方案

(API Level 1)

在 Log 类中使用 BuildConfig.DEBUG 标识来确保一些语句只在应用程序的调试阶段才会打印。出于对将来测试和开发考虑，即使在应用程序已经发布的情况下，在代码中保留一些日志语句还是极为方便的。但如果这些语句没有得到相应的保护，可能会在用户设备的控制台上打印出一些应用程序的隐私信息。通过对 Log 的简单封装来监控 BuildConfig.DEBUG，就可以放心地保留日志语句而不必担心它们会被错误显示。

6.13.3 实现机制

程序清单 6-32 显示了一个对于默认 Android 日志功能的简单封装类。

程序清单 6-32 日志封装类

```
public class Logger {
    private static final String LOGTAG = "AndroidRecipes";

    private static String getLogString(String format, Object... args) {
        //小优化，如果需要的话，只调用 String.format
        if(args.length == 0) {
            return format;
        }

        return String.format(format, args);
    }

    /*打印常用的 INFO、WARNING、ERROR 级别的日志*/

    public static void e(String format, Object... args) {
        Log.e(LOGTAG, getLogString(format, args));
    }

    public static void w(String format, Object... args) {
        Log.w(LOGTAG, getLogString(format, args));
    }

    public static void w(Throwable throwable) {
        Log.w(LOGTAG, throwable);
    }

    public static void i(String format, Object... args) {
        Log.i(LOGTAG, getLogString(format, args));
    }

    /*用 DEBUG 标识来限制 DEBUG 和 VERBOSE 日志级别*/

    public static void d(String format, Object... args) {
```

```

        if(!BuildConfig.DEBUG) return;

        Log.d(LOGTAG, getLogString(format, args));
    }

    public static void v(String format, Object... args) {
        if(!BuildConfig.DEBUG) return;

        Log.v(LOGTAG, getLogString(format, args));
    }
}

```

这个类对 Android 框架提供的日志做了一些简单的优化，使之更加实用。第一，它整合了整个应用程序的日志标记，这样 logcat 中的日志就会有统一的标题。其次，它的输入是一个格式化的字符串，这样变量不必分割日志字符串就可以打印。另外一个优化就是，String.format() 的执行可能会很慢，所以我们只在有参数特意指定时才会调用它。否则，直接返回原始字符串。

最后，它通过 BuildConfig.DEBUG 标识保护了五个主要日志级别中的两个，这两个级别的日志语句只有在应用程序调试时才会打印。在应用程序发布后也有一些情况(如应用程序出现错误)需要打印日志语句，所以最好不要隐藏 debug 后面的日志级别的日志。程序清单 6-33 展示了这个封装类是如何取代传统的日志功能的。

程序清单 6-33 使用 Logger 的 Activity

```

public class LoggerActivity extends Activity {

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

        //只有在调试时才会打印这条语句
        Logger.d("Activity Created");
    }

    @Override
    protected void onResume() {
        super.onResume();

        //只有在调试时才会打印这条语句
        Logger.d("Activity Resume at %d", System.currentTimeMillis());
        //总会打印这条语句
        Logger.i("It is now %d", System.currentTimeMillis());
    }

    @Override
    protected void onPause() {
        super.onPause();
    }
}

```

```

        //只有在调试时才会打印这条语句
        Logger.d("Activity Pause at %d", System.currentTimeMillis());
        //总会打印这条语句
        Logger.w("No, don't leave!");
    }
}

```

6.14 创建后台工作线程

6.14.1 问题

需要创建一个需要长时间运行的后台线程来执行某项任务，并且该线程在不用时可以很容易地终止。

6.14.2 解决方案

(API Level 1)

HandlerThread 可以帮助创建一个拥有 Looper 的后台线程，该 Looper 会关联一个 Handler，而 Handler 中的 MessageQueue 会处理所有的任务。Android 中最常用的后台技术就是 AsyncTask，这个类非常好用，可以在应用程序开发中使用它。但是，它也有一些缺点，即在某些情况下会使其他实现变得更加高效。一个缺点就是 AsyncTask 只能执行一次和受限的，如果想要在 Activity 或者 Service 这样的组件的生命周期中执行重复或者无限期的任务，AsyncTask 则显示有些重了。通常，需要创建多个 AsyncTask 实例才能完成这些任务。

在这种情况下，HandlerThread 的优势就是只需要创建一个工作对象，它可以在后台接收多个任务、并且通过 Looper 所维护的内部队列逐个处理每个任务。

6.14.3 实现机制

程序清单 6-34 展示了一个 HandlerThread 的子类，用来操作一些简单的图像数据。操作图像会需要一些时间，所以我们希望在一个后台线程中执行，这样就可以保证应用程序的界面不会被阻塞。

程序清单 6-34 后台工作线程

```

public class ImageProcessor extends HandlerThread implements Handler.Callback {
    public static final int MSG_SCALE = 100;
    public static final int MSG_CROP = 101;

    private Context mContext;
    private Handler mReceiver, mCallback;
}

```

```

public ImageProcessor(Context context) {
    this(context, null);
}

public ImageProcessor(Context context, Handler callback) {
    super("AndroidRecipesWorker");
    mCallback = callback;
    mContext = context;
}

@Override
protected void onLooperPrepared() {
    mReceiver = new Handler(getLooper(), this);
}

@Override
public boolean handleMessage(Message msg) {
    Bitmap source, result;
    //从传入的消息中解析参数
    int scale = msg.arg1;
    switch (msg.what) {
        case MSG_SCALE:
            source = BitmapFactory.decodeResource(mContext.getResources(),
                R.drawable.ic_launcher);
            //创建一个新的、缩放后的图片
            result = Bitmap.createScaledBitmap(source,
                source.getWidth() * scale, source.getHeight() * scale, true);
            break;
        case MSG_CROP:
            source = BitmapFactory.decodeResource(mContext.getResources(),
                R.drawable.ic_launcher);
            int newWidth = source.getWidth() / scale;
            //创建一个新的、横向裁剪的图片
            result = Bitmap.createBitmap(source,
                (source.getWidth() - newWidth) / 2, 0,
                newWidth, source.getHeight());
            break;
        default:
            throw new IllegalArgumentException("Unknown Worker Request");
    }

    //将图片返回给主线程
    if (mCallback != null) {
        mCallback.sendMessage(Message.obtain(null, 0, result));
    }
    return true;
}

//添加/删除回调 handler
public void setCallback(Handler callback) {

```

```

        mCallback = callback;
    }

    /*队列相关方法*/

    //缩放图标为特定的值
    public void scaleIcon(int scale) {
        Message msg = Message.obtain(null, MSG_SCALE, scale, 0, null);
        mReceiver.sendMessage(msg);
    }

    //居中裁剪图标，然后缩放为特定的值
    public void cropIcon(int scale) {
        Message msg = Message.obtain(null, MSG_CROP, scale, 0, null);
        mReceiver.sendMessage(msg);
    }
}

```

HandlerThread 这个名字可能不是特别恰当，这是因为它实际上并没有可以用来处理输入的 **Handler**。相反，它是一个线程，即和外部 **Handler** 一起创建的一个后台线程。正因为如此，我们必须自己实现一个 **Handler** 来真正处理我们想要执行的工作。本例中，我们自定义的处理器实现了 **Handler.Callback** 接口并传入了一个新的线程的 **Handler**。这样做也可以，而且可以避免使用 **Handler** 的子类。在 **onLooperPrepared()** 回调之后 **receiver Handler** 才会被创建，这是因为我们需要用 **HandlerThread** 所创建的 **Looper** 对象将要执行的任务发送到后台线程上。

我们创建的允许其他对象进入队列工作的外部 API 都创建了一个 **Message** 并将它发送到接收器 **Handler** 的 **handleMessage()** 方法中进行处理，该方法会检查 **Message** 的内容并创建适当修改过的图片。所有 **handleMessage()** 中的代码都会运行在我们的后台线程中。

当工作完成后，就需要另外使用一个主线程的 **Handler**，从而发送相关处理结果并更新 UI。

提醒：

任何和 UI 操作相关的代码必须在主线程中执行。

该回调 **Handler** 会接收一个 **Message**，该 **Message** 包含有图片数据的 **Bitmap**。这也是使用 **Message** 接口在不同的线程之间传递数据的好处之一；每个实例都可以带两个整型参数以及一个任意对象，所以在传递参数或者访问结果也不需要额外的代码。在这个示例中，传入的一个整型参数是图片的缩放值，对象参数则用来返回一个 **Bitmap** 结果。想要了解这个示例的实际结果，请参见程序清单 6-35 和 6-36。

程序清单 6-35 res/layout/main.xml

```

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

```

```

<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Scale Icon"
    android:onClick="onScaleClick" />
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Crop Icon"
    android:onClick="onCropClick" />

<ImageView
    android:id="@+id/image_result"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="center" />
</LinearLayout>

```

程序清单 6-36 和后台线程进行交互的 Activity

```

public class WorkerActivity extends Activity implements Handler.Callback {

    private ImageProcessor mWorker;
    private Handler mResponseHandler;

    private ImageView mResultView;

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

        mResultView = (ImageView) findViewById(R.id.image_result);
        //该 Activity 关联的后台回调 Handler
        mResponseHandler = new Handler(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        //启动一个新线程
        mWorker = new ImageProcessor(this, mResponseHandler);
        mWorker.start();
    }

    @Override
    protected void onPause() {
        super.onPause();
        //终止线程
        mWorker.setCallback(null);
    }
}

```

```

        mWorker.quit();
        mWorker = null;
    }

    /*
    *后台执行结果的回调方法
    *运行在 UI 线程上
    */
    @Override
    public boolean handleMessage(Message msg) {
        Bitmap result = (Bitmap) msg.obj;
        mResultView.setImageBitmap(result);
        return true;
    }

    /*将任务发送到后台线程 */

    public void onScaleClick(View v) {
        for(int i=1; i < 10; i++) {
            mWorker.scaleIcon(i);
        }
    }

    public void onCropClick(View v) {
        for(int i=1; i < 10; i++) {
            mWorker.cropIcon(i);
        }
    }
}

```

这个示例会在 Activity 处于前台时创建一个单独的后台线程，并且在用户单击按钮时将图片操作的请求发送给该线程。为了进一步演示缩放效果，在每次按钮单击时都会发送很多的操作请求。这个 Activity 也实现了 Handler.Callback，用一个简单的 Handler(运行在主线程上)接收后台线程发送过来的操作结果。

想要启动后台线程，只需要调用 HandlerThread 的 start()方法，这时会设置 Looper 和 Handler，然后等待任务的输入。终止后台线程就更简单了，只需要调用 quit()就可以停止 Looper 并立即移除队列中未处理的消息。我们将 callback 设为 null，这样此时可能正在运行的任务就不会再通知 Activity 了。

运行这个应用程序后，你会发现不管按钮按下的速度和频率多快，后台线程都不会拖慢 UI 线程。所有的请求都会被添加到消息队列中，在用户关闭 Activity 之前，这些请求会被尽可能处理。程序运行的结果就是在每个请求处理完成后在按钮的下方显示刚刚创建的图片。

6.15 自定义任务栈

6.15.1 问题

应用程序允许外部应用程序直接启动它的某些 Activity，你需要实现适当的 BACK 和 UP 导航行为。

6.15.2 解决方案

(API Level 4)

Android 支持库中的 NavUtils 和 TaskStackBuilder 可以很容易在应用程序中构建和生成合适的导航栈。实际上，这两个类的功能是 Android 4.1 以及以后版本原生的功能，但如果要在稍早版本的应用程序也支持该功能，可以使用 Android 支持库所提供的兼容 API，实际调用的也是本地方法。

BACK vs UP

Android 的界面导航有两个用户动作行为。一个就是用户按下 BACK 按钮后的行为。另一个则是在 ActionBar 上按下 Home 图标(如 UP 动作)后的行为。对于那些对平台还不太熟悉的开发者来说，经常会混淆它们，特别是很多情况下这两个动作都会执行相同的功能。

从概念上讲，BACK 应该让用户返回到上一个(相对于当前界面)浏览的界面。而 UP 动作应该返回当前界面的父界面。对于大多数应用程序来说，用户都是从主界面进入到带有特定内容的子界面，因此，BACK 和 UP 都会回到相同的地方，这也会让开发者对它们的用法产生质疑。

然而，考虑到对于一个应用程序的一个或多个 Activity 可以由外部应用程序启动的情况。例如，某个 Activity 的作用是用来浏览图像文件的。或者发送通知消息的应用程序在事件发生时允许用户直接进入到底层的 Activity。在这些情况下，BACK 动作会将用户返回到调用你的应用程序之前的应用程序中去。而对于 UP 动作，如果用户希望继续使用你的应用程序而不是回到之前的应用程序，UP 动作可以让用户返回到你的应用程序的界面栈中。这种情况下，你的应用程序的 Activity 栈通常还没有构建，这时就需要 TaskStackBuilder 和应用程序 manifest 中一些关键属性的帮助了。

6.15.3 实现机制

让我们定义两个应用程序来演示这个范例是如何工作的。首先查看一下程序清单 6-37，它显示了 manifest 中的<application>元素中的内容。

程序清单 6-37 AndroidManifest.xml 中的 Application 标签

```
<application
    android:icon="@drawable/ic_launcher"
```

```

    android:label="TaskStack"
    android:theme="@style/AppTheme" >
    <activity
        android:name=".RootActivity"
        android:label="@string/title_activity_root" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".ItemsListActivity"
        android:parentActivityName=".RootActivity">
        <!--为支持库定义的父亲界面-->
        <meta-data android:name="android.support.PARENT_ACTIVITY"
            android:value=".RootActivity" />
    </activity>
    <activity android:name=".DetailsActivity"
        android:parentActivityName=".ItemsListActivity">
        <!--为支持库定义的父亲界面-->
        <meta-data android:name="android.support.PARENT_ACTIVITY"
            android:value=".ItemsListActivity" />
        <!--提供一个过滤器，允许外部应用程序启动-->
        <intent-filter>
            <action android:name="com.examples.taskstack.ACTION_NEW_ARRIVAL" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
</application>

```

定义导航的第一步就是确定每个 Activity 之间的亲子关系。在 Android 4.1 中，引入了 `android:parentActivityName` 属性来创建这种关系。想要在老版本中也实现相同功能，需要使用支持库定义的 `<meta-data>` 值来为每个 Activity 定义父界面。我们的示例则采用了两种方式为每个底层 Activity 定义了亲子关系，即可以运行在本地 API 上也可以运行在支持库上。

在 `DetailsActivity` 中还有一个自定义的 `<intent-filter>`，它允许外部应用程序直接启动 `DetailsActivity`。

注意：

如果你的应用程序只支持 Android 4.1 及以后的版本，可以到此为止了。因为构建栈和导航的其余功能在这些版本中已经内置到了 Activity 的默认行为中，不必代码额外实现了。这种情况下，如果想在一些特殊的情形下自定义任务栈，只需要实现 `TaskStackBuilder` 即可。

定义好亲子关系后，开始编写每个 Activity 的代码。参见程序清单 6-38 到 6-40。

程序清单 6-38 RootActivity

```

public class RootActivity extends Activity implements View.OnClickListener {

```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Button listButton = new Button(this);
    listButton.setText("Show Family Members");
    listButton.setOnClickListener(this);

    setContentView(listButton,
        new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT));
}

public void onClick(View v) {
    //启动另一个 Activity
    Intent intent = new Intent(this, ItemsListActivity.class);
    startActivity(intent);
}
}

```

程序清单 6-39 第二级 Activity

```

public class ItemsListActivity extends Activity implements OnItemClickListener {

    private static final String[] ITEMS = {"Mom", "Dad", "Sister", "Brother",
        "Cousin"};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //启用 up 箭头的 ActionBar home 按钮
        getActionBar().setDisplayHomeAsUpEnabled(true);
        //创建并显示家庭成员的列表
        ListView list = new ListView(this);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1, ITEMS);
        list.setAdapter(adapter);
        list.setOnItemClickListener(this);

        setContentView(list);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                //创建父 Activity 的 Intent
                Intent upIntent = NavUtils.getParentActivityIntent(this);
                //检查是否需要创建整个栈
                if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
                    //若该栈不存在, 必须生成一个
                    TaskStackBuilder.create(this)

```

```

        .addParentStack(this)
        .startActivities();
    } else {
        //如果栈已经存在, 执行 up 的导航动作
        NavUtils.navigateUpFromSameTask(this);
    }
    return true;
default:
    return super.onOptionsItemSelected(item);
}
}

@Override
public void onItemClick(AdapterView<?> parent, View v, int position,
long id) {
    //启动最终的 Activity, 传入选择的条目的名字
    Intent intent = new Intent(this, DetailsActivity.class);
    intent.putExtra(Intent.EXTRA_TEXT, ITEMS[position]);
    startActivity(intent);
}
}

```

程序清单 6-40 第三级 Activity

```

public class DetailsActivity extends Activity {
    //自定义 Action 字符串, 用于外部 Activity 启动它
    public static final String ACTION_NEW_ARRIVAL =
        "com.examples.taskstack.ACTION_NEW_ARRIVAL";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //启用 up 箭头的 ActionBar home 按钮
        getActionBar().setDisplayHomeAsUpEnabled(true);

        TextView text = new TextView(this);
        text.setGravity(Gravity.CENTER);
        String item = getIntent().getStringExtra(Intent.EXTRA_TEXT);
        text.setText(item);

        setContentView(text);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                //创建父 Activity 的 Intent
                Intent upIntent = NavUtils.getParentActivityIntent(this);
                //检查是否需要创建整个栈
                if (NavUtils.shouldUpRecreateTask(this, upIntent)) {

```

```

        //若该栈不存在,必须生成一个
        TaskStackBuilder.create(this)
            .addParentStack(this)
            .startActivities();
    } else {
        //如果栈已经存在,执行 up 的导航动作
        NavUtils.navigateUpFromSameTask(this);
    }
    return true;
default:
    return super.onOptionsItemSelected(item);
}
}
}

```

这个示例应用程序由三个界面组成,根界面上只有一个按钮,可以启动第二级界面。第二级 Activity 中是一个含有一些选项的 ListView。选中一个选项后,就会启动第三级界面,该界面会在屏幕中间显示刚刚选择的选项。和想象中一样,用户可以使用 BACK 在这个界面栈中进行回退动作。然而,本例中还启用了 UP 动作来实现相同的界面导航。

在两个底层 Activity 启用 UP 导航时会有一些相同的代码。首先会调用 ActionBar 的 `setDisplayHomeAsUpEnabled()`。此时 bar 上的 home 图标可以被单击并显示默认的 back 箭头,也就是 UP 动作是可执行的。当用户单击这个箭头时,就会触发 `onOptionsItemSelected()`, ID 为 `android.R.id.home`,我们正式通过这个 ID 判断用户是否执行了 UP 导航请求。

在处理 UP 导航请求时,则要判断我们需要的 Activity 栈是否已经存在,是否需要创建, `shouldUpRecreateTask()` 方法就是这个作用。在 Android 4.1 之前的平台版本上,是判断目标 Intent 中是否包含非 `Intent.ACTION_MAIN` 的有效 action 字符串。而在 Android 4.1 及以后版本中,则是检查目标 Intent 的 `taskAffinity` 属性,从而得知与应用程序的其他界面是否存在亲子关系。

如果 Activity 栈不存在,主要是因为 Activity 是被直接启动的而不是在其应用程序中通过界面跳转而启动的,必须得重新创建。TaskStackBuilder 中包含了很多方法来创建界面栈,从而满足应用程序不同的需求。这里使用了便捷的 `addParentStack()` 方法,它会遍历所有的 `parentActivityName` 属性(或 `PARENT_ACTIVITY`,如果平台支持的话)和必要的 Intent 来重新创建这个 Activity 到根 Activity 之间的界面栈。之后,只需要调用 `startActivities()` 去构建栈并跳转到下一级界面。

如果栈已经存在,调用 NavUtils 的 `navigateUpFromSameTask()` 方法就可以回到上级界面。这种方法非常方便,不必像 `navigateUpTo()` 一样需要调用 `getParentActivityIntent()` 来构建一个目标 Intent。

现在,应用程序已经可以合理地响应 BACK/UP 导航模式,那么该如何测试呢?直接运行这个应用程序会发现 BACK 和 UP 动作执行的是相同的动作。让我们创建另一个简单的应用程序去启动 DetailsActivity,这样就可以很好地演示 BACK 和 UP 的导航效果。参见程序清单 6-41 和 6-42。

程序清单 6-41 res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/button_nephew"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add a New Nephew" />
    <Button
        android:id="@+id/button_niece"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add a New Niece" />
    <Button
        android:id="@+id/button_twins"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Add Twin Nieces!" />
</LinearLayout>

```

程序清单 6-42 Task 栈中启动 Activity

```

public class MainActivity extends Activity implements View.OnClickListener {
    // 自定义 Action 字符串，用于外部程序启动该 Activity
    public static final String ACTION_NEW_ARRIVAL =
        "com.examples.taskstack.ACTION_NEW_ARRIVAL";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 添加按钮监听器
        findViewById(R.id.button_nephew).setOnClickListener(this);
        findViewById(R.id.button_niece).setOnClickListener(this);
        findViewById(R.id.button_twins).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        String newArrival;
        switch(v.getId()) {
            case R.id.button_nephew:
                newArrival = "Baby Nephew";
                break;
            case R.id.button_niece:
                newArrival = "Baby Niece";
                break;
        }
    }
}

```

```

        case R.id.button_twins:
            newArrival = "Twin Nieces!";
            break;
        default:
            return;
    }

    Intent intent = new Intent(ACTION_NEW_ARRIVAL);
    intent.putExtra(Intent.EXTRA_TEXT, newArrival);
    startActivity(intent);
}
}

```

这个应用程序提供了一些选项供传入名字,然后会直接启动我们之前的 `DetailActivity`。本例中,我们看到了 `BACK` 和 `UP` 表现出来的不同行为。按下 `BACK` 键会将用户带回到选项选择界面,这是因为正是这个 `Activity` 启动了 `DetailActivity`。而按下 `UP` 动作按钮则会让用户返回原始应用程序的界面栈中,因此就显示 `ListView`。从这以后,用户的界面栈就发生了变化,此时按下 `BACK` 按钮同样会在原始应用程序的界面栈中进行导航,从而匹配后续的 `UP` 动作。参见图 6-7。

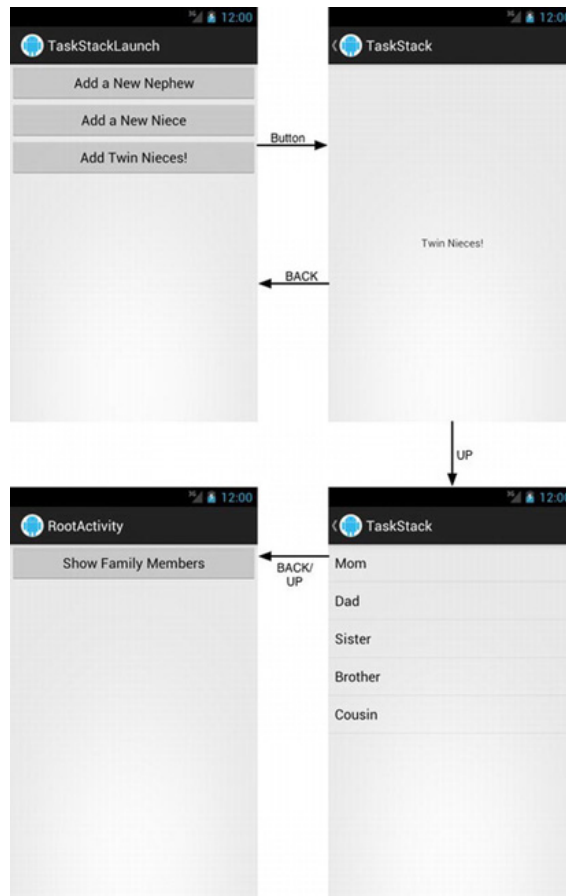


图 6-7 BACK vs UP 导航

6.16 实现 APPWidget

6.16.1 问题

应用程序提供的信息需要用户快速频繁地访问。你需要在用户的主界面中添加一个应用程序的可交互组件。

6.16.2 解决方案

(API Level 3)

用户可以选择将应用程序的部分安装到主界面上来构建 AppWidget。AppWidget 也是 Android 能够胜过其他操作系统的核心功能。AppWidget 最吸引人之处就用户可以在手机的主页面上定制对应用程序的快速访问。

AppWidget 即是一个 view，运行在手机启动器应用程序的进程中可以在你的应用程序进程中控制它。正因为如此，必须使用一种特殊的可以支持远程进程连接的框架。特别地，需要将 widget 的 view 结构封装到一个 RemoteViews 对象中，该对象可以通过 ID 更新 view 元素而不必直接访问它。RemoteViews 只支持框架中的一部分布局 and widget。下面的列表显示当前 RemoteViews 所支持的布局 and widget：

- Layouts
 - FrameLayout
 - GridLayout
 - LinearLayout
 - RelativeLayout
- Widgets
 - AdapterViewFlipper
 - AnalogClock
 - Button
 - Chronometer
 - GridView
 - ImageButton
 - ImageView
 - ListView
 - ProgressBar
 - StackView
 - TextView
 - ViewFlipper

你的 AppWidget 中的 view 只能由以上的这些元素组成，否则可能无法正确显示。

在一个远程进程中工作意味着很多的用户交互都要通过 `PendingIntent` 实例来传递，而不是传统的监听器接口。`PendingIntent` 可以让你的应用程序拦截 `Intent` 动作以及有权限执行该 `Intent` 的 `Context`，这样这个动作就可以自由地传递给其他进程并在某个特定的时间执行，整个过程就好像这个动作是从原始应用程序的 `Context` 传过来的一样。

大小

手机上的 `Android` 启动器界面通常都是 4×4 的网格界面空间，在这里可以调整 `AppWidget` 的大小。而平板电脑的空间会更大一些，在确定 `widget` 最小高度和宽度时需要考虑一下这种度量情况。`Android 3.1` 在 `AppWidget` 放置好以后还允许用户改变它的大小，但之前的平台版本中的 `AppWidget` 的大小是固定的。摘自 `Android` 文档，表 6-1 定义了一个比较好的规则，即一个给定的最小尺寸会占用多少网格。

表 6-1 主界面网格的大小

网格的大小	可用的空间
1	40dp
2	110dp
3	180dp
4	250dp
n	$70 * n - 30$

例如，`widget` 最小为 $200\text{dp} \times 48\text{dp}$ ，那么它在 `launcher` 上显示时需要三列一行。

6.16.3 实现机制

首先，我们看一下如何构建一个简单的 `AppWidget`，该 `AppWidget` 既可以通过 `widget` 本身更新也可以使用它所关联的 `Activity` 更新。这个示例会构造一个随机数字生成器(可以肯定大家都希望把它放置在启动器界面的) `AppWidget`。首先看一下程序清单 6-43 的应用程序的 `manifest` 文件。

程序清单 6-43 `AndroidManifest.xml`

```
<application android:label="@string/app_name"
    android:icon="@drawable/ic_launcher">
    <!--简单的 AppWidget 组件 -->
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name=".SimpleAppWidget">
```

```

        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>
        <!--配置 AppWidget 所需的数据 -->
        <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/simple_appwidget" />
    </receiver>

    <service android:name=".RandomService" />
</application>

```

这里创建 AppWidget 唯一需要做的就是添加名为 SimpleAppWidget 的<receiver>元素。这个元素必须指向 AppWidgetProvider 的子类，可能你已经想到了，该子类是一个自定义的 BroadcastReceiver。它必须在 manifest 中注册 APPWIDGET_UPDATE 广播动作。它还会处理很多其他的广播，但 APPWIDGET_UPDATE 广播动作必须要在 manifest 中声明。然后必须关联一个指向<appwidget-provider>的<meta-data>元素，它最终会被添加到 AppWidgetProviderInfo 中。程序清单 6-44 展示了 simple_appwidget.xml。

程序清单 6-44 res/xml/simple_appwidget.xml

```

<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="180dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/simple_widget_layout"/>

```

这些属性定义了 AppWidget 的配置。除了定义了大小外，updatePeriodMillis 还定义了 Android 自动更新该 widget 的时间间隔。这个值最好不要设置得比你需要的事件大。很多情况下，使用其他的 Service 或者观察者进行 AppWidget 的更新通知会更加高效。事实上，如果更新频率低于 30 秒，Android 系统是不会进行更新的。我们的 AppWidget 是一天一更新。这个示例还定义了一个 initialLayout 属性，它指定了 AppWidget 所使用的布局。另外这里还可以指定其他很多有用的属性：

- android:configure 指定了一个配置 AppWidget 的 Activity，该 Activity 是在 AppWidget 添加到启动器之前启动的。
- android:icon 该属性通过引用一个资源指定了系统选择界面中 AppWidget 的 icon 的样子。
- android:previewImage 该属性通过引用一个资源指定了系统选择界面(API Level)中 AppWidget 的全尺寸预览图源。
- android:resizeMode 该属性定义了在不同平台上如何调整大小：水平、垂直或者二者兼有(API Level 12)。

程序清单 6-45 和 6-46 展示了 AppWidget 的布局。

程序清单 6-45 res/layout/simple_widget_layout.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/widget_background"
    android:orientation="horizontal"
    android:padding="10dp" >
    <LinearLayout
        android:id="@+id/container"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="center_vertical"
        android:orientation="vertical">
        <TextView
            android:id="@+id/text_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:text="Random Number" />
        <TextView
            android:id="@+id/text_number"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:textStyle="bold"
            android:textAppearance="?android:attr/textAppearanceLarge" />
    </LinearLayout>

    <ImageButton
        android:id="@+id/button_refresh"
        android:layout_width="55dp"
        android:layout_height="55dp"
        android:layout_gravity="center_vertical"
        android:background="@null"
        android:src="@android:drawable/ic_menu_rotate" />

</LinearLayout>

```

程序清单 6-46 res/drawable/widget_background.xml

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners
        android:radius="10dp" />
    <solid
        android:color="#A333" />

```

```

        <stroke
            android:width="2dp"
            android:color="#333" />
    </shape>

```

通常最好为 AppWidget 创建一个在变化尺寸的容器中可以很容易拉伸和适配的布局 (特别是在 AppWidget 可以调整大小的平台)。这个示例中, 我们通过 XML 为 widget 定义了一个半透明的圆角矩形背景, 该背景可以适应任何尺寸。布局的子 view 也使用了 weight 属性, 这样就可以填充多余的空间。这个布局中有两个 TextView 和一个 ImageButton。这些 view 需要设置相应的 android:id 属性, 这是因为它们后面一旦被封装为 RemoteViews 实例, 将无法使用其他方式访问它们。程序清单 6-47 显示了之前提到的 AppWidgetProvider。

程序清单 6-47 AppWidgetProvider 实例

```

public class SimpleAppWidget extends AppWidgetProvider {
    /*
     * 更新这个 provider 所创建的 widget 时, 这个方法会被调用
     * 通常情况下, 以下情况会调用该方法:
     * 1. widget 开始被创建
     * 2. 到达了 AppWidgetProviderInfo 中定义的 updatePeriodMillis 时间间隔
     * 3. AppWidgetManager 中的 updateAppWidget() 方法被手动调用
     */
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
        //启动一个后台 Service 来更新 widget
        context.startService(new Intent(context, RandomService.class));
    }
}

```

这里只需要实现 onUpdate() 即可, 该方法在用户添加 widget 时会被调用, 另外之后 Android 框架或你的应用程序需要更新 widget 时也会调用该方法。很多情况下, 可以直接在该方法中创建 view 和更新 AppWidget。但由于 AppWidgetProvider 是一个 BroadcastReceiver, 最好不要在它里面执行时间过长的操作。如果构建 AppWidget 需要很多工作, 应该启动一个 Service 或者一个后台线程来完成, 我们这里就是这样做的。

为了方便, 这个方法会传入一个 AppWidgetManager 实例, 如果想要在这个方法中更新 AppWidget, AppWidgetManager 是必须要用到的。在同一个启动界面中是可以加载多个相同的 AppWidget 的。ID 数组就是每个 AppWidget 的引用, 因此可以全部更新。下面看一下程序清单 6-48 所示的 Service 的代码。

程序清单 6-48 AppWidget Service

```

public class RandomService extends Service {
    /*更新完成后的广播动作 */
    public static final String ACTION_RANDOM_NUMBER =
        "com.examples.appwidget.ACTION_RANDOM_NUMBER";
}

```

```

    /* 当前数据，通过静态变量保存*/
    private static int sRandomNumber;
    public static int getRandomNumber() {
        return sRandomNumber;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //更新随机数字
        sRandomNumber = (int)(Math.random() * 100);

        //创建 AppWidget
        RemoteViews views = new RemoteViews(getPackageName(),
            R.layout.simple_widget_layout);
        views.setTextViewText(R.id.text_number, String.valueOf(sRandomNumber));

        //为刷新按钮设置一个 Intent，该 Intent 会再次启动这个 service
        PendingIntent refreshIntent = PendingIntent.getService(this, 0,
            new Intent(this, RandomService.class), 0);
        views.setOnClickPendingIntent(R.id.button_refresh, refreshIntent);

        //设置一个 Intent，在单击 widget 文本时会打开一个 Activity
        PendingIntent appIntent = PendingIntent.getActivity(this, 0,
            new Intent(this, MainActivity.class), 0);
        views.setOnClickPendingIntent(R.id.container, appIntent);

        //更新 widget
        AppWidgetManager manager = AppWidgetManager.getInstance(this);
        ComponentName widget = new ComponentName(this, SimpleAppWidget.class);
        manager.updateAppWidget(widget, views);

        //发送一个广播，通知所有的监听者
        Intent broadcast = new Intent(ACTION_RANDOM_NUMBER);
        sendBroadcast(broadcast);

        //这个服务没必要继续运行下去
        stopSelf();
        return START_NOT_STICKY;
    }

    /*
     * 这里我们并没有绑定这个服务，所以这个方法返回 null 即可
     */
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}

```

这个 RandomService 在启动时会做两件事。首先，重新生成一个随机数字并保存到一

个静态变量中。第二，它为我们的 AppWidget 构建了一个新的 view。通过这种方式，我们就可以在需要时使用该服务更新 AppWidget 了。首先必须要创建一个 RemoteViews 实例，传入我们的 widget 布局，然后通过 setTextViewText()方法更新布局中 TextView 中的数字，通过 setOnClickPendingIntent()方法关联控件单击时的监听器。第一个 PendingIntent 关联的是 AppWidget 的刷新按钮，该 Intent 会重新启动这个 Service。第二个 PendingIntent 关联的是 widget 的主布局，单击主布局的任何地方都会启动应用程序的主 Activity。

RemoteViews 初始化的最后一步就是更新 AppWidget。这里是通过获得 AppWidgetManager 的实例然后调用 updateAppWidget()实现的。这里我们并没有 provider 关联的每个 AppWidget 的 ID，如果有的话，也可以使用 ID 更新每个 AppWidget。相反，我们可以传入一个引用 AppWidgetProvider 的 ComponentName，这时所有 AppWidgetProvider 关联的 AppWidget 都会被更新。

最后，我们发送了一个广播通知所有的监听器已经重新生成了一个新的随机数字，然后停止这个 Service。至此，我们完成了 AppWidget 的所有代码。最后我们需要额外添加一些布局代码和一个用于交互的 Activity。参见程序清单 6-49 和 6-50。

程序清单 6-49 res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Generate New Number"
        android:onClick="onRandomClick" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Current Random Number" />
    <TextView
        android:id="@+id/text_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="55dp"
        android:textStyle="bold" />

</LinearLayout>
```

程序清单 6-50 应用程序主 Activity

```
public class MainActivity extends Activity {
```

```

private TextView mCurrentNumber;

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

    mCurrentNumber = (TextView) findViewById(R.id.text_number);
}

@Override
protected void onResume() {
    super.onResume();
    updateNumberView();
    //注册一个 receiver 来在服务结束时接收更新
    IntentFilter filter = new
IntentFilter(RandomService.ACTION_RANDOM_NUMBER);
    registerReceiver(mReceiver, filter);
}

@Override
protected void onPause() {
    super.onPause();
    //解除 receiver 的注册
    unregisterReceiver(mReceiver);
}

public void onRandomClick(View v) {
    //调用 service 更新随机数字
    startService(new Intent(this, RandomService.class));
}

private void updateNumberView() {
    //用最新的数字更新 view
    mCurrentNumber.setText(String.valueOf(RandomService.getRandomNum-
ber()));
}

private BroadcastReceiver mReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        //使用新的数字更新 view
        updateNumberView();
    }
};
}

```

这个 Activity 显示了 RandomService 所提供的当前随机数字的值。单击按钮时还会启动 service 生成一个新的数字。它最大的好处就是会更新我们的 AppWidget 从而保证二者的同步。我们还注册了一个 BroadcastReceiver 监听 Service 生成新数字的结束事件，这样就

可以及时更新当前的界面。图 6-8 展示了应用程序的 Activity 以及添加到手机主界面上 AppWidget。

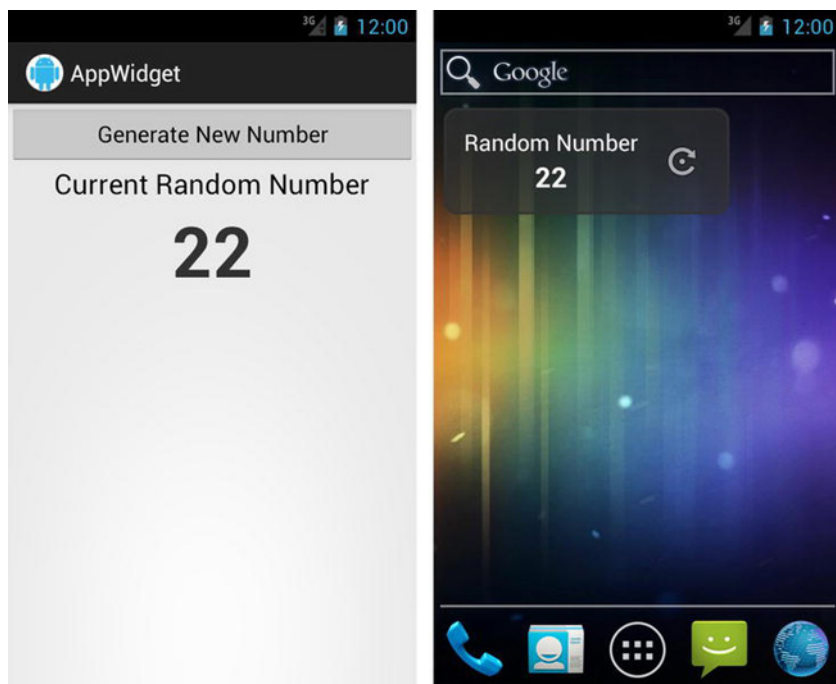


图 6-8 应用程序的随机数字 Activity(左边)和 AppWidget(右边)

基于集合的 AppWidget

(API Level 12)

从 Android 3.0 开始, 在 AppWidget 框架中加入了集合 view 后, AppWidget 上可以显示的东西也更多了。允许应用程序通过 list、grid 或者 stack 显示信息。在 Android 3.1 中, AppWidget 在放置好后还可以调整大小。让我们看一个 AppWidget 的示例, 该 AppWidget 允许用户查看自己的媒体集合信息。同样, 首先看一下程序清单 6-51 所示的 manifest。

程序清单 6-51 AndroidManifest.xml

```
<application android:label="@string/app_name"
    android:icon="@drawable/ic_launcher">
    <!--集合 AppWidget 组件 -->
    <activity android:name=".ListWidgetConfigureActivity">
        <intent-filter>
            <action
                android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
        </intent-filter>
    </activity>

    <receiver android:name=".ListAppWidget">
        <intent-filter>
            <action
```

```

        android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/list_appwidget" />
</receiver>

<service android:name=".ListWidgetService"
    android:permission="android.permission.BIND_REMOTEVIEWS" />
<service android:name=".MediaService" />
</application>

```

这个示例中的 AppWidgetProvider 和之前类似，名字变为 ListAppWidget。我们还定义了一个特殊权限(BIND_REMOTEVIEWS)的 Service。稍后会看到它实际上是一个 RemoteViewsService，Android 框架会使用这个 Service 为 AppWidget 提供数据，这个和 ListAdapter 与 ListView 的工作原理相似。最后，定义了一个 Activity，它会在用户添加 AppWidget 前对 AppWidget 进行设置。想要实现这个功能，这个 Activity 必须包含一个名为 APPWIDGET_CONFIGURE 的<intentfilter>。AppWidget 相关的 AppWidgetProviderInfo 定义可以参见程序清单 6-52。

程序清单 6-52 res/xml/list_appwidget.xml

```

<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="110dp"
    android:minHeight="110dp"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/list_widget_layout"
    android:configure="com.examples.appwidget.ListWidgetConfigureActivity"
    android:resizeMode="horizontal|vertical"/>

```

除了之前示例中介绍的一些标准属性，这里还添加 android:configure 属性(指向我们的配置 Activity)和 android:resizeMode 属性(允许 AppWidget 在水平垂直两个方向调整大小)。程序清单 6-53 到 6-55 显示了 AppWidget 和 ListView 条目的布局文件代码。

程序清单 6-53 res/layout/list_widget_layout.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@drawable/list_widget_background">
    <TextView
        android:id="@+id/text_title"
        android:layout_width="match_parent"
        android:layout_height="45dp"
        android:gravity="center"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <FrameLayout

```

```

        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <ListView
            android:id="@+id/list"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
        <TextView
            android:id="@+id/list_empty"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:text="No Items Available" />
    </FrameLayout>
</LinearLayout>

```

程序清单 6-54 res/drawable/list_widget_background.xml

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid
        android:color="#A333" />
</shape>

```

程序清单 6-55 res/layout/list_widget_item.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list_widget_item"
    android:layout_width="match_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:paddingLeft="10dp"
    android:gravity="center_vertical"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/line1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/line2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>

```

AppWidget 的布局就是一个简单的 ListView，ListView 上面是一个 TextView 标题。我们将 ListView 封装到一个 FrameLayout 中，这样就可以提供一个完全空白的 View。

提示:

尝试在 AppWidget 中使用一些 Android 标准的 ListView 行布局(例如 android.R.id.simple_list_item_1)将不会成功。这是因为这些行布局中通常会含有 CheckedTextView 这种 RemoteViews 所不支持的 view。必须要自己创建每行的布局。

在查看 AppWidgetProvider 之前,首先让我们看一下配置 Activity。这个界面是用户把 AppWidget 添加到主屏幕后(安装之前)看到的第一个界面。这个 Activity 的结果实际会检查 AppWidgetProvider 是否已经被调用。参见程序清单 6-56 和 6-57。

程序清单 6-56 res/layout/configure.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/text_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Select Media Type:" />
    <RadioGroup
        android:id="@+id/group_mode"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/text_title"
        android:orientation="vertical">
        <RadioButton
            android:id="@+id/mode_image"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Images" />
        <RadioButton
            android:id="@+id/mode_video"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Videos" />
    </RadioGroup>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Add Widget"
        android:onClick="onAddClick" />

</RelativeLayout>
```

程序清单 6-57 配置 Activity

```

public class ListWidgetConfigureActivity extends Activity {

    private int mAppWidgetId;
    private RadioGroup mModeGroup;

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

        mModeGroup = (RadioGroup) findViewById(R.id.group_mode);

        mAppWidgetId = getIntent()
            .getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                AppWidgetManager.INVALID_APPWIDGET_ID);

        setResult(RESULT_CANCELED);
    }

    public void onAddClick(View v) {
        SharedPreferences.Editor prefs =
            getSharedPreferences(String.valueOf(mAppWidgetId),
                MODE_PRIVATE)
            .edit();
        RemoteViews views = new RemoteViews(getPackageName(),
            R.layout.list_widget_layout);
        switch (mModeGroup.getCheckedRadioButtonId()) {
            case R.id.mode_image:
                prefs.putString(ListWidgetService.KEY_MODE,
                    ListWidgetService.MODE_IMAGE).commit();
                views.setTextViewText(R.id.text_title, "Image Collection");
                break;
            case R.id.mode_video:
                prefs.putString(ListWidgetService.KEY_MODE,
                    ListWidgetService.MODE_VIDEO).commit();
                views.setTextViewText(R.id.text_title, "Video Collection");
                break;
            default:
                Toast.makeText(this, "Please Select a Media Type.",
                    Toast.LENGTH_SHORT).show();
                return;
        }

        Intent intent = new Intent(this, ListWidgetService.class);
        intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
        intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHEME)));

        //通过指向 RemoteViewService 的 Intent 为列表关联一个 adapter, 从而填充数据
    }

```

```

        views.setRemoteAdapter(mAppWidgetId, R.id.list, intent);
        //设置列表为空白 view
        views.setEmptyView(R.id.list, R.id.list_empty);

        Intent viewIntent = new Intent(Intent.ACTION_VIEW);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
        viewIntent, 0);
        views.setPendingIntentTemplate(R.id.list, pendingIntent);
        AppWidgetManager manager = AppWidgetManager.getInstance(this);
        manager.updateAppWidget(mAppWidgetId, views);

        Intent data = new Intent();
        data.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
        setResult(RESULT_OK, data);
        finish();
    }
}

```

这个布局中有一个简单的用来选择图片和视频的 `RadioGroup`，之后会在 `AppWidget` 的列表和添加按钮上显示选择的媒体类型。按照惯例，在我们进入 `Activity` 后会立即将结果设为 `RESULT_CANCELED`。这是因为如果用户进入 `Activity` 不单击 `Add` 就离开，这时我们并不希望 `AppWidget` 显示在屏幕上。Android 框架会检测 `Activity` 的 `result` 从而决定是否添加 `AppWidget`。这里我们还得到了框架传过来的 `AppWidget` 的 ID，先保存起来。

当用户做了选择并单击了 `Add`，用户的选择就会被分别保存到名为 `AppWidget` 的 ID 的 `SharedPreferences` 实例中。这是因为我们希望应用程序可以处理多个 `widget`，而这些 `widget` 的配置是独立的，所以我们没有使用默认的 `SharedPreferences` 去保存数据。

注意：

在 Android 4.1 中，可以使用一个 `Bundle` 向 `AppWidget` 中传递配置数据。但为了兼容之前的版本，可以选择使用 `SharedPreferences` 传递数据。

然后开始构建 `AppWidget` 的 `RemoteViews`，根据用户选择的媒体类型设置 `RemoteView` 的标题。对于一个基于集合的 `AppWidget`，必须构建一个 `Intent` 来启动一个 `RemoteViews-Service`，从而作为集合中数据的 `adapter`，类似于 `ListAdapter`。然后通过 `setRemoteAdapter()` 将该 `adapter` 关联到 `RemoteViews`，这里需要传入 `adapter` 想要关联的 `ListView` 的 ID。在列表为空时，我们会使用 `setEmptyView()` 只显示一个 `TextView`。

每个列表条目都会关联一个 `PendingIntent`，用户单击列表的条目时会为触发这个 `PendingIntent`。框架需要知道你为每个条目设置的特定信息，因此这里为每个条目都使用一个 `PendingIntent` 模板。我们为每个条目都创建了一个简单 `ACTION_VIEW` 的 `Intent`，然后通过 `setPendingIntentTemplate()` 把它与列表关联起来，数据和额外信息会在稍后填充。

准备工作做好后，我们调用了 `AppWidgetManager` 的 `updateAppWidget()`。本例中，由于我们只想更新这个特定的 `AppWidget`，所以调用了只有一个 ID 参数而不是 `ComponentName` 作为参数的 `updateAppWidget()` 版本。然后，设置 `result` 的值为 `RESULT_OK` 并 `finish`，这时框架会将 `AppWidget` 添加到主屏幕上。让我们简单地看一下程序清单 6-58 所示的

AppWidgetProvider。

程序清单 6-58 List AppWidgetProvider

```
public class ListAppWidget extends AppWidgetProvider {

    /*
     * 更新这个 provider 所创建的 widget 时，这个方法会被调用
     * 通常情况下，以下情况会调用该方法：
     * 由于添加了配置 Activity，因此在第一次添加 widget 并不会调用该方法，但在下面这
    种还会被调用：
     * 1.到达了 AppWidgetProviderInfo 中定义的 updatePeriodMillis 时间间隔
     */
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds) {
        //更新这个 provider 创建的所有 widget
        for (int i=0; i < appWidgetIds.length; i++) {
            Intent intent = new Intent(context, ListWidgetService.class);
            intent.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                appWidgetIds[i]);
            intent.setData(Uri.parse(intent.toUri(Intent.URI_INTENT_SCHE
                -ME)));

            RemoteViews views = new RemoteViews(context.getPackageName(),
                R.layout.list_widget_layout);
            //根据 widget 的配置设置标题
            SharedPreferences prefs =
                context.getSharedPreferences(String.valueOf
                    (appWidgetIds[i]), Context.MODE_PRIVATE);
            String mode = prefs.getString(ListWidgetService.KEY_MODE,
                ListWidgetService.MODE_IMAGE);
            if (ListWidgetService.MODE_VIDEO.equals(mode)) {
                views.setTextViewText(R.id.text_title, "Video Collection");
            } else {
                views.setTextViewText(R.id.text_title, "Image Collection");
            }

            //通过指向 RemoteViewService 的 Intent 为列表关联一个 adapter，从而填
            充数据
            views.setRemoteAdapter(appWidgetIds[i], R.id.list, intent);

            //为列表设置空白 view
            views.setEmptyView(R.id.list, R.id.list_empty);

            //为每个条目设置单击时的模板 Intent
            Intent viewIntent = new Intent(Intent.ACTION_VIEW);
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
                viewIntent, 0);
            views.setPendingIntentTemplate(R.id.list, pendingIntent);
        }
    }
}
```

```

        appWidgetManager.updateAppWidget(appWidgetIds[i], views);
    }
}

/*
 * 当第一个 widget 添加到 providerr 会调用
 */
@Override
public void onEnabled(Context context) {
    //启动一个 service 来监控 MediaStore
    context.startService(new Intent(context, MediaService.class));
}

/*
 * 当所有的 widget 从 provider 中移除时会调用
 */
@Override
public void onDisabled(Context context) {
    //停止监控 MediaStore 的 service
    context.stopService(new Intent(context, MediaService.class));
}

/*
 * 当一个或者多个 widget 从这个 provider 中移除时会调用
 */
@Override
public void onDeleted(Context context, int[] appWidgetIds) {
    //每个 widget 移除时, 同时移除为每个 widget 创建的 SharedPreferences
    for (int i=0; i < appWidgetIds.length; i++) {
        context.getSharedPreferences(String.valueOf(appWidgetIds[i]),
            Context.MODE_PRIVATE)
            .edit()
            .clear()
            .commit();
    }
}
}

```

除了将之前的更新动作换成了现在读取用户配置信息动作, 该 provider 中 `onUpdate()` 方法中的代码和配置 Activity 中的代码几乎是一样的。这是因为我们希望后续的更新会有相同的 AppWidget 结果。

这个 provider 还覆写了 `onEnabled()` 和 `onDisabled()`。这两个方法分别会在第一个 widget 被添加到 provider 以及最后一个 widget 移除时调用。这时会使用它们启动和停止一个需要长时间运行的 Service, 稍后会详细了解这个 Service。最后, 每个 AppWidget 在移除时会回调 `onDeleted()` 方法。在这个示例中, 会在这个方法中清除在 AppWidget 被添加时创建的 `SharedPreferences`。

现在查看一下程序清单 6-59，它定义了 RemoteViewsService 来为 AppWidget 列表提供数据。

程序清单 6-59 RemoteViewsAdapter

```
public class ListWidgetService extends RemoteViewsService {

    public static final String KEY_MODE = "mode";
    public static final String MODE_IMAGE = "image";
    public static final String MODE_VIDEO = "video";

    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent) {
        return new ListRemoteViewsFactory(this, intent);
    }

    private class ListRemoteViewsFactory implements
        RemoteViewsService.RemoteViewsFactory {
        private Context mContext;
        private int mAppWidgetId;

        private Cursor mDataCursor;

        public ListRemoteViewsFactory(Context context, Intent intent) {
            mContext = context.getApplicationContext();
            mAppWidgetId =
                intent.getIntExtra(AppWidgetManager.EXTRA_APPWIDGET_ID,
                    AppWidgetManager.INVALID_APPWIDGET_ID);
        }

        @Override
        public void onCreate() {
            //得到用户在添加 widget 时设置的信息
            SharedPreferences prefs =
                mContext.getSharedPreferences(String.valueOf(
                    mAppWidgetId), MODE_PRIVATE);
            //得到用户的配置信息，默认为图片模式
            String mode = prefs.getString(KEY_MODE, MODE_IMAGE);
            //根据用户的配置信息查询相应的媒体类型数据
            if (MODE_VIDEO.equals(mode)) {
                //在 MediaStore 中查询视频数据
                String[] projection = {MediaStore.Video.Media.TITLE,
                    MediaStore.Video.Media.DATE_TAKEN,
                    MediaStore.Video.Media.DATA};
                mDataCursor = MediaStore.Images.Media.query(getContentResolver(),
                    MediaStore.Video.Media.EXTERNAL_CONTENT_URI, projection);
            } else {
                //在 MediaStore 中查询图片数据
                String[] projection = {MediaStore.Images.Media.TITLE,
                    MediaStore.Images.Media.DATE_TAKEN,
```

```

        MediaStore.Images.Media.DATA};
        mDataCursor = MediaStore.Images.Media.query(getContentResolver(),
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
            projection);
    }
}

/*
 * 这个方法会在 onCreate()后被调用，但外部调用
 * AppWidgetManager.notifyAppWidgetViewDataChanged()来刷新 widget
数据时也会调用这个方法
 */
@Override
public void onDataSetChanged() {
    //刷新 Cursor 数据
    mDataCursor.requery();
}

@Override
public void onDestroy() {
    //不需要时关闭 cursor
    mDataCursor.close();
    mDataCursor = null;
}

@Override
public int getCount() {
    return mDataCursor.getCount();
}

/*
 * 如果数据来自于网络或者其他可能需要一些时间来下载，这里可以显示一个 loading
view
 * 这个 loading view 在 getViewAt()阻塞时会一直显示，直到 getViewAt()返回
时才消失
 */
@Override
public RemoteViews getLoadingView() {
    return null;
}

/*
 * 返回列表中的每个条目的 view。这个方法里可以放心的执行长时间的操作
 * 该方法返回之前，会一直显示一个 loading view
 */
@Override
public RemoteViews getViewAt(int position) {
    mDataCursor.moveToPosition(position);

    RemoteViews views = new RemoteViews(getPackageName(),

```

```

        R.layout.list_widget_item);
views.setTextViewText(R.id.line1, mDataCursor.getString(0));
views.setTextViewText(R.id.line2, DateFormat.format("MM/dd/yyyy",
        mDataCursor.getLong(1)));

SharedPreferences prefs = mContext
        .getSharedPreferences(String.valueOf(mAppWidgetId),
MODE_PRIVATE);
String mode = prefs.getString(KEY_MODE, MODE_IMAGE);
String type;
if (MODE_VIDEO.equals(mode)) {
    type = "video/*";
} else {
    type = "image/*";
}

Uri data = Uri.fromFile(new File(mDataCursor.getString(2)));

Intent intent = new Intent();
intent.setDataAndType(data, type);
views.setOnClickListenerFillInIntent(R.id.list_widget_item, intent);

return views;
}

@Override
public int getViewTypeCount() {
    return 1;
}

@Override
public boolean hasStableIds() {
    return false;
}

@Override
public long getItemId(int position) {
    return position;
}
}
}

```

RemoteViewsService 必须要返回一个 RemoteViewsFactory 的实现，这里和 ListAdapter 非常相似。很多方法如 getCount()、getViewTypeCount() 作用于在本地列表中的功能是一样的。当 RemoteViewsFactory 开始创建时，我们会检查用户在配置阶段选择的设置值，然后会在系统的 MediaStore Content Provider 中查询相应的 Cursor 来显示图片或者视频数据。当不需要 factory 而要销毁它时，会关闭刚才查询到的 Cursor。当由于外部原因通知 AppWidgetManager 需要更新数据时，就会调用 notifyDataSetChanged() 方法。要更新数据，我们只需要重新查询 Cursor。

`getViewAt()`用来获得列表中每个条目的 `view`。这个方法可以放心地执行长时间的操作(例如网络 I/O),这时 Android 框架在 `getViewAt()`方法返回之前会一直显示 `getLoadingView()`方法中指定的内容。这个示例中,我们需要更新 `RemoteViews` 中每行布局中的标题和文本(指定条目创建的时间)。我们设置了图片或者视频数据的文件路径以及将 `data` 字段设置为相应的 MIME 类型。再加上 `ACTION_VIEW`,就可以在设备的 Gallery 应用程序(或者条目被单击时,打开其他可以处理媒体数据的应用程序)中打开相应的文件了。

你可能已经注意到了,这个示例在查询 `Cursor` 数据时并没有使用明确的字段名。这主要是因为两种媒体类型的 `projections` 中相应字段的名称是不同的,所以直接使用索引访问会更加高效些。最后,让我们看一下程序清单 6-60,它展示了由 `AppWidgetProvider` 进行启动和停止的后台 `service` 代码。

程序清单 6-60 更新监控 Service

```
public class MediaService extends Service {

    private ContentObserver mMediaStoreObserver;

    @Override
    public void onCreate() {
        super.onCreate();
        //Service 启动后,创建一个 MediaStore 的观察者
        mMediaStoreObserver = new ContentObserver(new Handler()) {
            @Override
            public void onChange(boolean selfChange) {
                //更新 AppWidgetProvider 当前所关联的所有的 widget
                AppWidgetManager manager =
                    AppWidgetManager.getInstance(MediaService.this);
                ComponentName provider = new ComponentName(MediaService.this,
                    ListAppWidget.class);
                int[] appWidgetIds = manager.getAppWidgetIds(provider);
                //这个方法会触发 RemoteViewsService 中的 onDataSetChanged()方法
                manager.notifyAppWidgetViewDataChanged(appWidgetIds, R.id.list);
            }
        };
        //注册 Image 和 Video 的观察者
        getContentResolver().registerContentObserver(
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI, true,
            mMediaStoreObserver);
        getContentResolver().registerContentObserver(
            MediaStore.Video.Media.EXTERNAL_CONTENT_URI, true,
            mMediaStoreObserver);
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        //当 service 停止后,解除观察者的注册
        getContentResolver().unregisterContentObserver(mMediaStoreObserver);
    }
}
```

```

    }

    /*
     * 我们不需要绑定这个 Service, 返回 null 即可
     */
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}

```

这个 Service 的作用就是在有 AppWidget 可用时为 MediaStore 注册一个 ContentObserver。这样, 在添加或者删除了一个图片或者视频时, 就可以及时更新我们的 widget 的列表。每当触发 ContentObserver 时候, 我们都会对当前关联的每个 widget 调用 AppWidgetManager 的 notifyAppWidgetViewDataChanged() 方法。这时就会触发 onDataSetChanged() 中的 onDataSetChanged() 回调方法来更新列表。在图 6-9 和 6-10 中可以看到运行的结果。

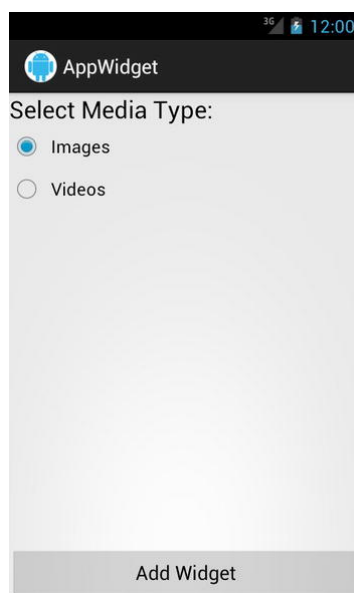


图 6-9 添加 AppWidget 前出现的配置 Activity

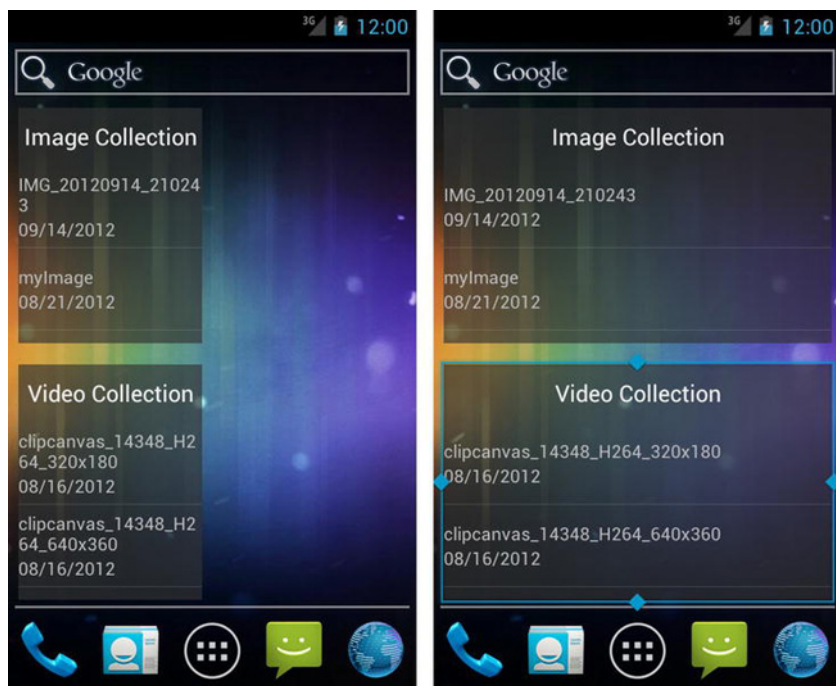


图 6-10 两种媒体类型的 AppWidget(左边)和调整大小后的效果(右边)

可以看到, 只需要在 AppWidgetProviderInfo 中添加一个简单的 resize 属性, 用户就可以调整 AppWidget 的大小了。每个列表都是可滚动的, 单击列表中的条目会打开默认的应用程序来浏览图片或者播放视频。

6.17 小结

本章中，你学会了如何让应用程序直接与 Android 操作系统互动。讨论了几种将操作提交到后台运行的方法，这些操作所需的运行时间长短不一。了解应用程序之间如何分工合作，相互调用，从而更好地完成任务。最后介绍了系统如何将其核心应用程序所收集的数据提供给其他应用程序使用。在第 7 章，将学习如何利用各种 Java 库来开发更加强大的应用程序。

第 7 章

使 用 库

聪明的 Android 开发者会利用各种各样的第三方库来将他们的应用程序快速发布到应用市场上，第三方库提供了相对稳定的代码，可以减少开发时间。开发者既可以使用他们自己的库，也可以使用他人开发的库，或者二者一起使用。

本章开始的范例介绍了如何创建并使用自己的库。接下来的范例会介绍 Kidroid 用于绘制柱状图、折线图、饼状图的 kiChart 图形库；用 IBM 的消息队列推送库(MQTT)在应用程序中实现轻量级的消息推送；使用 Google 支持包，它提供了各种各样的库，在一些旧的 Android 平台上，应用程序可以使用这些库访问旧平台不支持 Android 的新特性。

提示：

OpenIntents.org 发布了一个库列表，你或许可以找到对你的应用程序开发有帮助的库 (<http://openintents.org/en/libraries>)。其中包括了 AdWhirl 库，它可以在你的应用程序中嵌入来自各种广告网络的广告或者自己定制的广告，前面提到的 kiChart 库也在其中。

7.1 创建 Java 库 JAR

7.1.1 问题

需要创建一个库来存放可能会用于 Android 项目的代码，这个库也可以用在非 Android 项目中。

7.1.2 解决方案

创建一个基于 JAR 的库，用 JDK 命令行工具或 Eclipse 限制只使用 Java 5 (或更早版本) 的 API。

7.1.3 实现机制

假设你打算创建一个简单的数学计算工具库。这个库中有一个 `MathUtils` 类，类中有不同的静态方法。程序清单 7-1 展示了这个类的初始版本。

程序清单 7-1 通过静态方法实现了数学计算工具的 `MathUtils`

```
ca.tutortutor.mathutils;

public class MathUtils
{
    public static long factorial(long n)
    {
        if (n <= 0)
            return 1;
        else
            return n*factorial(n-1);
    }
}
```

现在的 `MathUtils` 只有一个 `long factorial(long n)` 类方法，它会计算并返回阶乘结果(可以用在排列组合的计算中)。你还可以扩展这个类来支持快速的傅里叶变化和其他 `java.lang.Math` 类所不支持的数学操作。

警告：

当创建可能在 Android 项目中使用的库时，要保证使用的是 Android 支持的标准 Java API(例如集合相关的 API)。不要使用 Android 不支持的 Java API(例如 Swing)或者 Android 专属的 API(例如 `Android widget`)。

1. 使用 JDK 创建 `MathUtils`

使用 JDK 开发基于 JAR 的库非常容易。执行以下步骤来创建一个包含 `MathUtils` 类的 `mathutils.jar` 文件。

(1) 在当前目录创建一个 `ca` 子目录，然后在 `ca` 子目录下创建一个 `tutortutor`，再在 `tutortutor` 子目录下创建一个 `mathutils` 子目录。

(2) 复制程序清单 7-1 的 `MathUtils.java` 源代码到 `mathutils` 目录下的 `MathUtils.java` 文件中。

(3) 假设当前目录中有 `ca` 子目录，执行 `javacca/tutortutor/mathutils/MathUtils.java` 编译 `MathUtils.java`。然后会在 `ca/tutortutor/mathutils` 生成一个 `MathUtils.class` 文件。

(4) 执行 `jar cfv mathutils.jarca/tutortutor/mathutils/*.class` 来生成 `mathutils.jar`。生成的 `mathutils.jar` 中会包含 `ca/tutortutor/mathutils/MathUtils.class`。

注意：

如果使用的是 JDK 7，可以使用以下一种命令行来编译 `MathUtils.java`：

```
javac -source 1.5 -target 1.5
```

```
ca/tutortutor/mathutils/MathUtils.java
```

```
javac -source 1.6 -target 1.6
ca/tutortutor/mathutils/MathUtils.java
```

每个命令行都会导致一个无关紧要的“bootclasspath”警告消息, https://blogs.oracle.com/darcy/entry/bootclasspath_older_source 对该信息做了解释。

如果不这样做的话, 在使用 ant debug 引用这个 JAR 库构建 APK 时会出现以下的警告消息:

```
[dx] trouble processing:

[dx] bad class file magic (cafebabe) or version (0033.0000)

[dx] ...while parsing ca/tutortutor/mathutils/MathUtils.class

[dx] ...while processing
ca/tutortutor/mathutils/MathUtils.class

[dx] 1 warning
```

此外, 在安装了 APK 并运行后, UseMathUtils 会导致出现一个 “Unfortunately, UseMathUtils has stopped.” 的对话框。虽然使用 JDK 环境开发 Android 应用程序和库存在一些问题(第 1 章就已经发现), 但还是可用的, 正如你所看到的第 1 章和上面的示例。

2. 使用 Eclipse 创建 MathUtils

使用 Eclipse 开发基于 JAR 的库稍微有点麻烦。执行以下步骤来创建一个包含 MathUtils 类的 mathutils.jar 文件:

- (1) 按照第 1 章介绍的方法安装好 Eclipse 后, 首先启动 Eclipse。
- (2) 选择 File 菜单中的 New, 在之后的弹出菜单中选择 Java Project。
- (3) 在弹出的 New Java Project 对话框中, 在 Project name 一栏中输入 mathutils。如果执行环境 JRE(详见 JRE 一节)被设置为 JavaSE-1.7, 请改为 JavaSE-1.6。之后, 单击 Finish 按钮。
- (4) 展开 Package Explorer 中的 mathutils 节点。在 src(位于 mathutils 节点下)节点上右击, 然后选择 New, 在弹出的菜单中选择 Package。
- (5) 在之后出现的 New Java Package 对话框中, 在 Name 一栏输入 ca.tutortutor.mathutils, 单击 Finish 按钮。
- (6) 然后右击 ca.tutortutor.mathutils 节点, 选择 New, 在弹出的菜单中选择 Class。
- (7) 在之后出现的 New Java Class 对话框中, 在 Name 一栏中输入 MathUtils, 单击 Finish 按钮。
- (8) 用程序清单 7-1 中的代码替换 MathUtils.java 中的框架代码。
- (9) 右击 mathutils 项目节点, 在弹出的菜单中选择 Build Project(首先可能需要在 project 菜单中取消 Build Automatically 的选择)。忽略 “Build path specifies execution environment

JavaSE-1.6. There are no JREs installed in the workspace that are strictly compatible with this environment” 警告消息。

(10) 右击 mathutils 项目节点，在弹出的菜单中选择 Export。

(11) 在之后出现的 Export 对话框中，选择 Java 节点下的 JAR file(如果未选择的话)，然后单击 Next 按钮。

(12) 在之后出现的 JAR Export 对话框中，在 JAR file 一栏中输入 mathutils.jar，其他保持不动。单击 Finish 按钮(这时，如果还没有保存第 8 步复制的代码，就会出现 Save Modified Resources 对话框)。最后生成的 mathutils.jar 会出现 Eclipse 工作区的根目录中。

7.2 使用 Java 库 JAR

7.2.1 问题

已经成功构建了 mathutils.jar，想要了解如何将这个 JAR 文件整合到基于命令行或者 Eclipse 的 Android 项目中。

7.2.2 解决方案

需要创建一个基于命令行或者 Eclipse 的 Android 项目，在 Android 项目中创建一个 libs 目录并把 mathutils.jar 复制到这个目录。

注意：

通常都是把库文件(.jar 文件和 Linux 共享目标库.so 文件)保存到 Android 项目目录中的 libs 子目录下。Android 构建系统会自动到 libs 中查找库文件并把它们集成到 APK 中。如果是共享目标库，它就会保存到.apk 文件的 lib 目录下(而不是 libs)。

7.2.3 实现机制

现在已经创建了 mathutils.jar，将需要一个 Android 应用程序来试验一下这个库。程序清单 7-2 展示单 activity 应用程序的 UseMathUtils 的源代码，它会计算 5 的阶乘，然后显示计算结果。

程序清单 7-2 调用 MathUtil 的 factorial()方法计算 5 的阶乘的 UseMathUtils

```
package ca.tutortutor.usemathutils;

import android.app.Activity;

import android.os.Bundle;

import android.widget.TextView;

import ca.tutortutor.mathutils.MathUtils;
```

```

public class UseMathUtils extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("5! = "+MathUtils.factorial(5));
        setContentView(tv);
    }
}

```

1. 使用 Android SDK 创建并运行 UseMathUtils

执行以下命令(为了便于阅读,分两行显示)创建一个 UseMathUtils 项目:

```

android create project -t 1 -p C:\prj\dev\UseMathUtils -a UseMathUtils
                        -k ca.tutortutor.usemathutils

```

这个命令的前提是 Android 4.1 平台、Window 系统、项目保存在 C:\prj\dev 目录下。

现在,用程序清单 7-2 的内容替换 src/ca/tutortutor/usemathutils/UseMathUtils.java 源文件中的内容。

接下来将 mathutils.jar 复制到项目的 libs 子目录下。然后,执行以下命令在调试模式下构建这个项目:

```
ant debug
```

如果构建成功,在项目的 bin 子目录下执行以下命令把 UseMathUtils-debug.apk 安装到 AVD1 上,如下:

```
adb install UseMathUtils-debug.apk
```

最后,启动应用程序。应该会看到如图 7-1 所示的输出信息。



图 7-1 UseMathUtils 的界面可以扩展为让用户输入任意的数字

2. 使用 Eclipse 创建和运行 UseMathUtils

执行以下步骤使用 Eclipse 创建 UseMathUtils 项目:

- (1) 按照第 1 章介绍的方法安装好 Eclipse 后,首先启动 Eclipse。
- (2) 选择 File 菜单中的 New,在之后的弹出菜单中选择 Project。
- (3) 在之后出现的 New Project 对话框中,展开向导树中的 Android 节点(如果尚未展开),选择该节点下的 Android Application Project(如果尚未选择),单击 Next 按钮。
- (4) 在之后出现的 New Android App 对话框中,在 Application Name 一栏输入 UseMathUtils。

这个名字同时会在 Project Name 一栏中显示,它表示了 UseMathUtils 项目所在的文件夹/目录。

(5) 在 Package Name 一栏输入 `ca.tutortutor.usemathutils`。

(6) 通过 Build Target 选择适合的 Android SDK 目标版本。这也是你即将构建的应用程序可以运行的 Android 平台。如果只安装了 Android 4.1 平台,就会只显示这一种平台并选中。

(7) 通过 Minimum SDK,可以设置运行应用程序所需的最低 Android SDK 版本,或者保持默认值。

(8) 如果需要创建一个自定义的启动图标,就保持“Create custom launcher icon”复选框的选中状态。否则,设置它为未选中状态,这时,会使用系统默认的启动图标。

(9) 因为并不是创建一个库,所以保持“Mark this project as a library”复选框的未选中状态即可。

(10) 保持“Create Project in Workspace”复选框的选中状态,单击 Next 按钮。

(11) 在之后出现的 Configure Launcher Icon 界面中,适当调整自定义的启动图标;单击 Next 按钮。

(12) 在之后出现的 Create Activity 界面中,保持 Create Activity 的选中状态,并确保选中了 BlankActivity,单击 Next 按钮。

(13) 在之后出现的 Blank Activity 界面,在 Activity Name 一栏输入 UseMathUtils。其他设置项保持不动,单击 Next 按钮(如果 Next 可用)。否则,单击 Finish 按钮。

(14) 如果 Next 可以单击的话,会看到一个 InstallDependencies 界面,它是提示你安装 Google 的支持库(本章的最后一个范例会讲到)。单击 Install/Upgrade 按钮来安装这个库,然后按照安装之后出现的对话框的提示操作即可。单击 Finish 按钮,就完成了项目的创建。

现在,Eclipse 会在 Package Explorer 窗口中创建一个 UseMathUtils 节点。执行以下步骤设置所需的文件。

(1) 展开 UseMathUtils 节点(未展开的话),然后展开 src 节点,最后展开 `ca.tutortutor.usemathutils` 节点。

(2) 双击打开 UseMathUtils.java 文件并用程序清单 7-2 的代码替换 UseMathUtils.java 中的代码。忽略所有的错误信息,一会它们就会消失。

(3) 使用系统的文件管理器选中之前创建的 mathutils.jar 文件并拖曳到 libs 节点下。如果显示 File Operation 对话框,保持选中 Copy files radio 按钮,并单击 OK 按钮。

(4) 展开 libs 节点,右击 mathutils.jar,选择 Build Path,然后在之后出现的弹出菜单 Configure Build Path。

(5) 在之后出现的 Properties for UseMathUtils 对话框中,选择 Libraries 选项卡,单击 Add Jars 按钮。

(6) 在之后出现的 JAR Selection 对话框中,展开 UseMathUtils 节点以及 libs 节点。选择 mathutils.jar,然后单击 OK 按钮关闭 JAR Selection 对话框。再次单击 OK 按钮关闭 Properties for UseMathUtils 对话框。

想要构建这个项目,右击 UseMathUtils 节点并在弹出的菜单中选择 Build Project。然后,再次选中 UseMathUtils 节点,选择菜单栏中的 Run,并之后出现的下拉菜单中选择

Run(如果出现了保存 UseMathUtils.java 的提示,在 Save Resource 对话框中单击 Yes 即可)。如果接下来出现 Run As 对话框,选择 Android Application 并单击 OK 按钮。这时 Eclipse 会启动模拟器,安装项目的 APK 并运行应用程序。图 7-2 显示了输出信息。

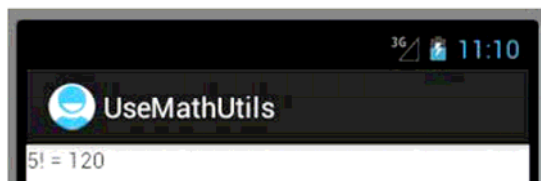


图 7-2 因为是 Eclipse 生成的自定义主题,所以 UseMathUtils 的用户界面看起来会有点不同

注意:

如果查看这个应用程序的 UseMathUtils.apk 文件(jar tvfUseMathUtils.apk),并不会找到 mathutils.jar。相反,你会看到一个 classes.dex 文件,它包含了应用程序的 Dalvik 可执行字节码。这个 classes.dex 中会包含 MathUtils 的字节码,这是因为 Android 构建系统会将 JAR 文件解压并将它的 Java 字节码内容用 dx 工具转换为 Dalvik 字节码,并将这些字节码添加到 classes.dex 文件中(UseMathUtilsdebug.apk 也是同样的道理)。

7.3 创建 Android 库项目

7.3.1 问题

想要创建一个专门用于 Android 的库,例如自定义的部件或者带有资源(也可以不带)的 Activity。

7.3.2 解决方案

可以创建一些 Android 库项目,这些项目中包含一些 Android 共享代码和资源,可以在其他 Android 项目中引用它们。这对于代码的复用非常有用。库项目是不可以安装到设备上的。它们在构建.apk 时会被添加到.apk 文件中。

注意:

Android 4.0 SDK(r14)对 Android 库项目的特性进行了调整。在之前的版本中,在编译资源和应用程序的代码时所引用的库项目有自己单独的资源 and 源代码目录。因为开发者希望将编译好的代码和资源只通过一个 JAR 文件发布出去,而且库项目的实现都非常脆弱,所以 r14 引用 Android 库项目是基于编译好的代码的。更多信息可以查阅“Changes to Library Projects in Android SDK Tools, r14”博客文章(<http://android-developers.blogspot.ca/2011/10/changes-to-library-projects-in-android.html>)。

7.3.3 实现机制

假设你要创建的库中包含一个可重用的自定义 View——一个游戏棋盘(用于玩国际象

棋、跳棋或者井字棋)。

程序清单 7-3 是实现这个 view 的 GameBoard 类。

程序清单 7-3 GameBoard 描述了一个可重用的自定义 View，用于绘制各种游戏棋盘

```
package ca.tutortutor.gameboard;

import android.content.Context;

import android.graphics.Canvas;
import android.graphics.Paint;

import android.view.View;

public class GameBoard extends View
{
    private int nSquares, colorA, colorB;

    private Paint paint;
    private int squareDim;

    public GameBoard(Context context, int nSquares, int colorA, int colorB)
    {
        super(context);
        this.nSquares = nSquares;
        this.colorA = colorA;
        this.colorB = colorB;
        paint = new Paint();
    }

    @Override
    protected void onDraw(Canvas canvas)
    {
        for (int row = 0; row < nSquares; row++)
        {
            paint.setColor(((row & 1) == 0) ? colorA : colorB);
            for (int col = 0; col < nSquares; col++)
            {
                int a = col*squareDim;
                int b = row*squareDim;
                canvas.drawRect(a, b, a+squareDim, b+squareDim, paint);
                paint.setColor((paint.getColor() == colorA) ? colorB : colorA);
            }
        }
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
    {
        //让 view 保持为正方形
    }
}
```

```

        int width = MeasureSpec.getSize(widthMeasureSpec);
        int height = MeasureSpec.getSize(heightMeasureSpec);
        int d = (width == 0) ? height : (height == 0) ? width :
            (width < height) ? width : height;
        setMeasuredDimension(d, d);
        squareDim = width/nSquares;
    }
}

```

Android 自定义 view 或者是 `android.view.View` 的子类，亦或者是 `android.view.View` 子类(如 `android.widget.TextView`)的子类。`GameBoard` 直接继承了 `View` 类，这是因为它不需要任何 `View` 子类的功能。`GameBoard` 声明了以下字段：

- `nSquares` 是棋盘每条边的格子数。常见的值为 3(即 3×3 的棋盘)和 8(即 8×8 的棋盘)。
- `colorA` 是偶数行的偶数格子和奇数行的奇数格子的颜色。行和列都是从 0 开始编号的。
- `colorB` 是偶数行的奇数格子和奇数行的偶数格子的颜色。
- `paint` 是一个 `android.graphics.Paint` 对象的引用，用来设置绘制棋盘上的格子颜色(`colorA` 或 `colorB`)。
- `squareDim` 是格子的大小，即每条边的像素数量。

`GameBoard` 的构造函数会将 `nSquares`、`colorA` 和 `colorB` 赋值给同名的字段，实例化这个部件，然后实例化 `Paint` 类。在此之前，需要将 `context` 传递给它的 `View` 父类。

注意：

`View` 的子类都需要给 `View` 传递一个 `android.content.Context` 实例。这样做是为了标识自定义 view 是运行在哪个上下文中。自定义的 `View` 子类随后可以通过 `View` 的 `Context` `getContext()` 方法得到这个 `Context` 对象，这样就可以调用 `Context` 的方法访问当前的主题和资源等。

Android 通过调用自定义 view 所覆写的 `protected void onDraw(Canvas canvas)` 方法来绘制该自定义 view。`GameBoard` 的 `onDraw(Canvas)` 会调用 `android.graphics.Canvas` 的 `void drawRect(float left, float top, float right, float bottom, Paint paint)` 方法绘制各行各列的格子。最后的 `paint` 参数则用于设置格子的颜色。

在 Android 调用 `onDraw(Canvas)` 之前，需要知道自定义 view 的宽和高。它是通过覆写 `View` 的 `protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)` 来实现的，该方法传入的参数为其父 `View` 所决定的宽度和高度。自定义 view 通常将这两个参数传递给嵌套在这个类的 `static int getSize(int measureSpec)` 方法中的 `View.MeasureSpec`，从而根据 `measureSpec` 参数得到自定义 view 的宽度和高度。返回的值(或稍作改动后)必须传递给 `View` 的 `setMeasuredDimension(int measuredWidth, int measuredHeight)` 方法来保存测得的宽度和高度。如果在运行时调用该方法失败，会抛出异常。因为游戏棋盘必须是方的，所以 `GameBoard` 的 `onMeasure(int, int)` 会将最小的宽度和高度传递给 `setMeasuredDimension(int, int)`，以确保游戏棋盘是正方形的。

1. 通过 Android SDK 创建 GameBoard

创建 Android 库项目与创建标准的应用程序项目非常类似。区别就是命令行将以“android create lib-project”开头而不是“android create project”，语法如下：

```
android create lib-project --target target_ID
                           --name your_project_name
                           --path /path/to/your/project/project_name
                           --package your_library_package_namespace
```

这个命令行创建了一个标准的项目结构，想要表明该项目是一个库项目，需要将以下的代码添加到项目的 project.properties 文件中：

```
android.library=true
```

命令执行后，就会创建库项目，可以将源代码和资源添加进去。

提示：

如果想要将现有的应用程序项目转换为一个库项目以便其他应用程序使用的话，可以在应用程序项目的 project.properties 文件中添加 android.library=true 属性。

执行以下命令(为了阅读方便，分两行显示)来创建一个 GameBoard 库项目：

```
android create lib-project -t 1 -p C:\prj\dev\GameBoard
                           -k ca.tutortutor.gameboard
```

继续在 src 目录下创建 ca/tutortutor/gameboard 这样的文件夹结构，将 GameBoard.java 的内容使用程序清单 7-3 的内容替换并保存到该文件目录下。

虽然可以通过执行 ant debug 或者 ant release 命令(不管用哪个命令，都会在 bin 目录下生成相同的 classes.jar 文件)构建库，但没必要这么做，因为当有其他项目引用这个库时，这个库会自动构建。

2. 使用 Eclipse 创建 GameBoard

使用 Eclipse 执行以下步骤来创建 GameBoard 项目：

- (1) 按照第 1 章介绍的方法安装好 Eclipse 后，首先启动 Eclipse。
- (2) 选择 File 菜单中的 New，在之后的弹出菜单中选择 Project。
- (3) 在之后出现的 New Project 对话框中，展开向导树中的 Android 节点(如果尚未展开)，选择该节点下的 Android Application Project(如果尚未选择)，单击 Next 按钮。
- (4) 在之后出现的 New Android App 对话框中，在 Application Name 一栏输入 GameBoard。这个名字同时会在 Project Name 一栏中显示，它表示了 GameBoard 项目所在的文件夹/目录。
- (5) 在 Package Name 一栏输入 ca.tutortutor.gameboard。
- (6) 通过 Build Target 选择合适的 Android SDK 目标版本。这也是你即将构建的应用程序可以运行的 Android 平台。如果只安装了 Android 4.1 平台，就会只显示这一种平台并被选中。

(7) 通过 Minimum SDK，可以设置运行应用程序所需的最低 Android SDK 版本，或者保持默认值。

(8) 因为库中并没有使用自定义的启动图标，所以不要选中 Create custom launcher icon。

(9) 选中 Mark this project as a library 复选框。

(10) 保持 Create Project in Workspace 复选框为选中状态，单击 Next 按钮。

(11) 在之后出现的 Create Activity 界面中，保持 Create Activity 的选中状态，单击 Finish 按钮。

GameBoard 项目现在就是一个 Android 库项目。但还缺少一个包含了程序清单 7-3 的内容的 GameBoard.java 源文件。

在 Package Explorer 的 GameBoard/src 节点下新建一个 ca.tutortutor.gameboard 节点(右击 src，选择 New，然后在弹出的菜单中选择 Package，在之后出现的 New Java Package 对话框中的 Name 一栏输入 ca.tutortutor.gameboard，单击 Finish 按钮)，在 ca.tutortutor.gameboard 节点下新建 GameBoard.java 文件(右击 ca.tutortutor.gameboard，选择 New，然后在弹出的菜单中选择 Class，在之后出现的 New Java Class 对话框的 Name 一栏输入 GameBoard，单击 Finish 按钮)，双击 GameBoard.java 文件，使用程序清单 7-3 的内容替换该内容。

虽然可以通过右击 GameBoard 文件并在弹出的菜单中选择 Build Project 的方式构建库(这时会在 bin 目录下生成一个 gameboard.jar 文件)，但没必要这么做，因为当有其他项目引用这个库时(下一范例会演示)，这个库会自动构建。

7.4 使用 Android 库项目

7.4.1 问题

成功构建了 GameBoard 库文件后，希望将这个库整合到基于命令行或者 Eclipse 的 Android 项目中。

7.4.2 解决方案

需要在要构建的应用程序项目的属性中设置 GameBoard 库，然后构建应用程序。

7.4.3 实现机制

现在已经创建了 GameBoard 库，需要用一个 Android 应用程序试用一下这个库。程序清单 7-4 显示了一个单一 activity 的应用程序 UseGameBoard 的源代码，它实例化了库中的 GameBoard 类并将其添加到了 activity 的 view 中。

程序清单 7-4 UseGameBoard 将 GameBoar 组件添加到了 Activity 的 View 中

```
package ca.tutortutor.usegameboard;
```

```

import android.app.Activity;

import android.graphics.Color;

import android.os.Bundle;

import ca.tutortutor.gameboard.GameBoard;

public class UseGameBoard extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        GameBoard gb = new GameBoard(this, 8, Color.BLUE, Color.WHITE);
        setContentView(gb);
    }
}

```

1. 使用 Android SDK 创建和运行 UseGameBoard

执行以下命令行(为了阅读方便, 分两行显示)来创建一个 UseGameBoard 项目:

```

android create project -t 1 -p C:\prj\dev\UseGameBoard -a UseGameBoard
                        -k ca.tutortutor.usegameboard

```

现在, 使用程序清单 7-4 中的内容替换 src/ca/tutortutor/usegameboard/UseGameBoard.java 中的内容。

然后执行以下命令行(为了阅读方便, 分两行显示)引用 GameBoard 库项目:

```

android update project -t 1 -p C:\prj\dev\UseGameBoard
                      -l ../GameBoard

```

可以通过“android update project”命令的-l(--library)选项让 UseGameBoard 引用 GameBoard 库项目(这里必须指定库项目的相对路径, 否则可能会构建失败, 提示信息会包含“Failed to resolve library path”)。对 GameBoard 库的引用信息会保存在 project.properties 文件中:

```

android.library.reference.1=../GameBoard

```

注意:

引用库项目的形式为 android.library.reference.n, n 为整型, 从 1 开始计数。

可以重复执行“android update project”命令实现对多个库项目的引用, 每条命令要指向不同的库。每次命令成功执行后都会在 project.properties 文件中将计数递增 1(2、3 等)。

计数不允许出现空档(如 android.library.reference.1=... and android.library.reference.3=... 中间缺少了 android.library.reference.2=...)。空档后面的计数所指向的库会被忽略掉(android.library.reference.3=... 会被忽略掉)。

在应用程序构建时, 库中的内容会按照优先级从低(计数最小)到高(计数最大)的顺序一

个个合并到应用程序中。注意一个库不能引用另一个库，并且在应用程序构建时，在库合并到应用程序之前，库与库之间是不能互相合并的。

接下来，执行以下命令行在调试模式下构建这个项目：

```
ant debug
```

如果构建成功，在项目的 bin 子目录下执行以下命令把 UseGameBoard-debug.apk 安装到 AVD1 上，如下：

```
adb install UseGameBoard-debug.apk
```

最后，启动应用程序。应该会看到如图 7-3 所示的输出信息。

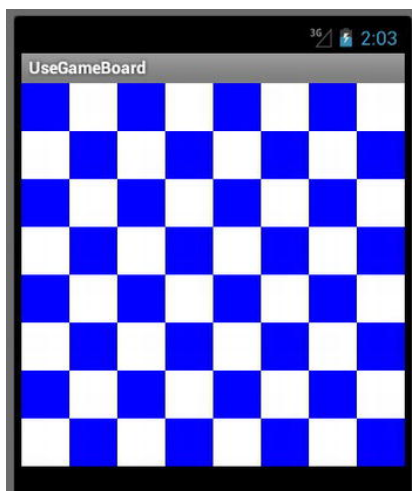


图 7-3 UseGameBoard 绘制出蓝白相间的游戏棋盘，可以用作跳棋或者国际象棋的背景

2. 使用 Eclipse 创建和运行 UseGameBoard

执行以下步骤在 Eclipse 中创建一个 UseGameBoard 项目：

- (1) 按照第 1 章介绍的方法安装好 Eclipse 后，首先启动 IDE。
- (2) 选择 File 菜单中的 New，在之后的弹出菜单中选择 Project。
- (3) 在之后出现的 New Android App 对话框中，在 Application Name 一栏输入 UseGameBoard。这个名字同时会在 Project Name 一栏中显示，它表示了 UseGameBoard 项目所在的文件夹/目录。
- (4) 在 Package Name 一栏输入 ca.tutortutor.usegameboard。
- (5) 通过 Build Target 选择适合的 Android SDK 目标版本。这也是你即将构建的应用程序可以运行的 Android 平台。如果只安装了 Android 4.1 平台，就会只显示这一种平台并选中。
- (6) 通过 Minimum SDK，可以设置运行应用程序所需的最低 Android SDK 版本，或者保持默认值。
- (7) 如果需要创建一个自定义的启动图标，就保持 Create custom launcher icon 复选框的选中状态。否则，设置它为未选中状态，这时，会使用系统默认的启动图标。

(8) 因为并不是创建一个库，所以保持“Mark this project as a library”复选框的未选中状态即可。

(9) 保持“Create Project in Workspace”复选框的选中状态，单击 Next 按钮。

(10) 在之后出现的 Configure Launcher Icon 界面中，适当地调整自定义的启动图标，单击 Next 按钮。

(11) 在之后出现的 Create Activity 界面中，保持 Create Activity 的选中状态，并确保选中了 BlankActivity，单击 Next 按钮。

(12) 在之后出现的 Blank Activity 界面，在 Activity Name 一栏输入 UseGameBoard。其他设置项保持不动，单击 Finish 按钮。

现在，Eclipse 会在 Package Explorer 窗口中创建一个 UseGameBoard 节点。执行以下步骤设置所需的文件。

(1) 展开 UseGameBoard 节点(未展开的话)，然后展开 src 节点，最后展开 ca.tutortutor.usegameboard 节点。

(2) 双击打开 UseGameBoard.java 文件(位于 underneathca.tutortutor.usegameboard 下)并用程序清单 7-4 的代码替换 UseGameBoard.java 中的代码。

(3) 右击 UseGameBoard 节点并在弹出的菜单中选择 Properties。

(4) 在之后出现的 Properties for UseGameBoard 对话框中，选择 Android 分类并单击 Add 按钮。

(5) 在之后出现的 Project Selection 对话框中，选择 GameBoard，单击 OK 按钮。

(6) 单击 Apply，然后单击 OK 按钮关闭 Properties for UseGameBoard。

想要构建和运行这个项目，选择菜单栏中的 Run，并之后出现的下拉菜单中选择 Run(如果出现了 Save Resources 的提示，单击 OK 按钮即可)。如果接下来出现 Run As 对话框，选择 Android Application 并单击 OK 按钮。这时 Eclipse 会启动模拟器，安装项目的 APK 并运行应用程序。图 7-4 显示了输出信息。

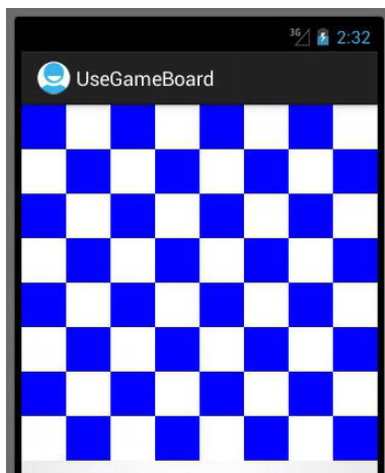


图 7-4 因为是 Eclipse 生成的自定义主题，所以 UseGameBoard 的用户界面看起来会有点不同

注意：

如果对在基于 Android 库项目的库中加入 Activity 感兴趣的话，请查看 Google 提供的

“Setting Up a Library Project” (<http://developer.android.com/tools/projects/projectseclipse.html#SettingUpLibraryProject>)和“Referencing a Library Project” (<http://developer.android.com/tools/projects/projectseclipse.html#ReferencingLibraryProject>)项目文档。

7.5 绘图

7.5.1 问题

需要一个简单的库来绘制柱状图、折线图或饼状图。

7.5.2 解决方案

用于绘图的 Android 库已经有好几个了，而 Kidroid.com 的 kiChart(www.kidroid.com/kichart/)以其简单易用吸引了很多人。0.3 版本支持柱状图、折线图或饼状图，Kidroid 承诺随后会加入新的图形类型。

在上面链接所对应的 kiChart 主页上可以下载 kiChart-0.3.jar(库)和 kiChart-manual-0.3.pdf(库的描述文档)。

7.5.3 实现机制

kiChart 的文档说它的绘图支持多系列数据，而且它所绘制的图形可以自定义参数(例如，字体颜色、字体大小、边距等等)并输出到图片文件中。

文档随后展示了一些由 demo 应用程序生成的折线图、柱状图和饼状图的截图。这些截图都有相应的代码，实际上就是 LineChart Activity。

LineChart 的源代码展示了绘制一个图形的基本步骤，如下所述：

(1) 创建一个从 `com.kidroid.kichart.ChartActivity` 继承的 Activity。这个 Activity 将绘制柱状图、折线图或饼状图。

(2) 在 Activity 的 `onCreate(Bundle)`方法中，创建一个 `String` 数组表示横轴标签，创建一个表示要绘制的每个圆柱或者线条数据的浮点数数组。

(3) 创建一个 `com.kidroid.kichart.model.Aitem`(数轴)数组，将保存在数据数组中的 `Aitem` 对象填入这个数组。调用 `Aitem` 的构造函数需要提供一个 `android.graphics.Color` 值表示数据数组的颜色(即显示数值和圆柱或线条的颜色)，一个 `String` 值表示颜色和数据数组的标签，以及数据数组本身(对于饼状图，需要使用 `com.kidroid.kichart.model.Bitem` 类)。

(4) 如果想要显示柱状图就实例化 `com.kidroid.kichart.view.BarView`；如果要显示折线图，就实例化 `com.kidroid.kichart.view.LineView`；如果要显示饼状图，就实例化 `com.kidroid.kichart.view.PieView`。

(5) 调用这个类的 `public void setTitle(String title)`方法设置图形的标题。

(6) 调用 `BarView` 或 `LineView` 的 `public void setAxisValueX(String[] labels)`方法设置柱状图或者折线图的横轴标签。

(7) 调用 `BarView` 或 `LineView` 的 `public void setItems(Aitem[] items)` 方法设置图形的数据项数组, 或者调用 `PieView` 的 `public void setItems(Bitem[])` 方法设置图形的数据项。

(8) 用这个类的实例作为参数调用 `setContentView()`, 显示图形。

(9) 不必担心纵轴的数值范围, `kiChart` 会自行选择合适的数值范围。

根据源代码可以看出 `kiChart` 中的各个类及其之间的关系。例如, `com.kidroid.kichart.view.ChartView` 是 `com.kidroid.kichart.view.AxisView` 的父类, 而 `com.kidroid.kichart.view.AxisView` 又是 `BarView` 和 `LineView` 的父类。虽然没有显示, `ChartView` 还是 `PieView` 的父类。

每个类的属性和 `ChartView` 的 `public boolean exportImage(String filename)` 方法都有相应的文档。这个方法用于将图形输出为一个 PNG 文件, 成功则返回 `true`, 否则返回 `false`。

提示:

要改变纵轴上显示的值的范围, 需要用到 `AxisView` 的 `intervalCount`、`intervalValue` 和 `valueGenerate` 属性。

在实际使用中, 你会发现 `kiChart` 用起来很简单。看一个示例, `ChartDemo` 应用程序的主 `Activity` (名字也为 `ChartDemo`) 是让用户在界面的 8 个文本框中分别输入 2010 和 2011 年的各个季度的销售额。主 `Activity` 上还有两个按钮, 让用户在 `BarChart`、`LineChart` 和 `PieChart Activity` 中以柱状图、折线图或者饼状图的形式查看这些数据。

程序清单 7-5 是 `ChartDemo` 的源代码。

程序清单 7-5 `ChartDemo` 演示了让用户输入数据后绘制柱状图、折线图或饼状图的 `Activity`

```
package ca.tutortutor.chartdemo;

import android.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.view.View;

import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;

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

        Button btnViewBC = (Button) findViewById(R.id.viewbc);
```

```

AdapterView.OnClickListener ocl;
ocl = new AdapterView.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        final float[] data2010 = new float[4];
        int[] ids = { R.id.data2010_1, R.id.data2010_2,
R.id.data2010_3,
                    R.id.data2010_4 };
        for(int i = 0; i < ids.length; i++)
        {
            EditText et = (EditText) findViewById(ids[i]);
            String s = et.getText().toString();
            try
            {
                float input = Float.parseFloat(s);
                data2010[i] = input;
            }
            catch(NumberFormatException nfe)
            {
                data2010[i] = 0;
            }
        }
        final float[] data2011 = new float[4];
        ids = new int[] { R.id.data2011_1, R.id.data2011_2,
                        R.id.data2011_3, R.id.data2011_4 };
        for(int i = 0; i < ids.length; i++)
        {
            EditText et = (EditText) findViewById(ids[i]);
            String s = et.getText().toString();
            try
            {
                float input = Float.parseFloat(s);
                data2011[i] = input;
            }
            catch (NumberFormatException nfe)
            {
                data2011[i] = 0;
            }
        }
        Intent intent = new Intent(ChatDemo.this, BarChart.class);
        intent.putExtra("2010", data2010);
        intent.putExtra("2011", data2011);
        startActivity(intent);
    }
};
btnViewBC.setOnClickListener(ocl);

Button btnViewLC = (Button) findViewById(R.id.viewlc);

```

```

ocl = new AdapterView.OnItemClickListener()
{
    @Override
    public void onClick(View v)
    {
        final float[] data2010 = new float[4];
        int[] ids = { R.id.data2010_1, R.id.data2010_2,
            R.id.data2010_3,
                R.id.data2010_4 };
        for(int i = 0; i < ids.length; i++)
        {
            EditText et = (EditText) findViewById(ids[i]);
            String s = et.getText().toString();
            try
            {
                float input = Float.parseFloat(s);
                data2010[i] = input;
            }
            catch (NumberFormatException nfe)
            {
                data2010[i] = 0;
            }
        }
        final float[] data2011 = new float[4];
        ids = new int[] { R.id.data2011_1, R.id.data2011_2,
            R.id.data2011_3, R.id.data2011_4 };
        for(int i = 0; i < ids.length; i++)
        {
            EditText et = (EditText) findViewById(ids[i]);
            String s = et.getText().toString();
            try
            {
                float input = Float.parseFloat(s);
                data2011[i] = input;
            }
            catch (NumberFormatException nfe)
            {
                data2011[i] = 0;
            }
        }
        Intent intent = new Intent(ChartDemo.this, LineChart.class);
        intent.putExtra("2010", data2010);
        intent.putExtra("2011", data2011);
        startActivity(intent);
    }
};
btnViewLC.setOnClickListener(ocl);

Button btnViewPC = (Button) findViewById(R.id.viewpc);
ocl = new AdapterView.OnItemClickListener()

```

```

    {
        @Override
        public void onClick(View v)
        {
            final float[] data2010 = new float[4];
            int[] ids = { R.id.data2010_1, R.id.data2010_2,
                R.id.data2010_3,
                    R.id.data2010_4 };
            for(int i = 0; i < ids.length; i++)
            {
                EditText et = (EditText) findViewById(ids[i]);
                String s = et.getText().toString();
                try
                {
                    float input = Float.parseFloat(s);
                    data2010[i] = input;
                }
                catch(NumberFormatException nfe)
                {
                    data2010[i] = 0;
                }
            }
            final float[] data2011 = new float[4];
            ids = new int[] { R.id.data2011_1, R.id.data2011_2,
                R.id.data2011_3, R.id.data2011_4 };
            for(int i = 0; i < ids.length; i++)
            {
                EditText et = (EditText) findViewById(ids[i]);
                String s = et.getText().toString();
                try
                {
                    float input = Float.parseFloat(s);
                    data2011[i] = input;
                }
                catch(NumberFormatException nfe)
                {
                    data2011[i] = 0;
                }
            }
            Intent intent = new Intent(ChatDemo.this, PieChart.class);
            intent.putExtra("2010", data2010);
            intent.putExtra("2011", data2011);
            startActivity(intent);
        }
    };
    btnViewPC.setOnClickListener(ocl);
}
}

```

ChartDemo 在它的 onCreate(Bundle)方法中实现了所有的逻辑。这个方法的主要功能就

是设置 content view，并给 view 中的两个按钮加上单击监听器。

因为这两个监听器的功能基本上相同，所以我们只给出 viewbc(查看柱状图)按钮的监听器代码(饼状图监听器的代码会有很多的不同，稍后会给出)。当按钮被按下时，就会调用监听器的 onClick(View)方法来完成以下工作：

- (1) 将 2010 年的 4 个文本框中的值填入 data2010 浮点数数组。
- (2) 将 2011 年的 4 个文本框中的值填入 data2011 浮点数数组。
- (3) 创建一个 Intent 对象，将 BarChart.class 设为要启动的 Activity。
- (4) 将 data2010 和 data2011 数组保存到这个对象中，让 BarChart Activity 可以访问这些数据。
- (5) 启动 BarChart Activity。

程序清单 7-6 是 BarChart 的源代码。

程序清单 7-6 BarChart 是显示柱状图的 Activity

```
package ca.tutortutor.chartdemo;

import com.kidroid.kichart.ChartActivity;

import com.kidroid.kichart.model.Aitem;

import com.kidroid.kichart.view.BarView;

import android.graphics.Color;

import android.os.Bundle;

public class BarChart extends ChartActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Bundle bundle = getIntent().getExtras();
        float[] data2010 = bundle.getFloatArray("2010");
        float[] data2011 = bundle.getFloatArray("2011");
        String[] arrX = new String[4];
        arrX[0] = "2010.1";
        arrX[1] = "2010.2";
        arrX[2] = "2010.3";
        arrX[3] = "2010.4";
        Aitem[] items = new Aitem[2];
        items[0] = new Aitem(Color.RED, "2010", data2010);
        items[1] = new Aitem(Color.GREEN, "2011", data2011);
        BarView bv = new BarView(this);
        bv.setTitle("Quarterly Sales (Billions)");
        bv.setAxisValueX(arrX);
        bv.setItems(items);
    }
}
```

```

        setContentView(bv);
    }
}

```

BarChart 首先用 Intent getIntent()方法获得它所收到的 Intent 对象的引用, 然后它用这个方法得到 Intent 对象的 Bundle 对象的引用, 其中以浮点数组的形式保存了数据。调用 Bundle 的 float[]getFloatArray(String key)方法即可得到数组。

接下来, BarChart 构建一个表示图形 X 轴标签的 String 数组, 创建一个 Aitem 数组并往其中填入两个 Aitem 对象。第一个对象中是 2010 年的数据, 这些数据的颜色是红色, 图例是 2010; 另一个对象中是 2011 年的数据, 颜色是绿色, 图例是 2011 年。

在初始化好了 BarView 后, BarChart 会调用该对象的 setTitle(String)方法设置图形的标题, 调用 setAxisValueX(String[])方法和 setItems(Aitem[])方法分别将表示 X 轴标签的数组和 Aitem 数组传递给该对象。然后将 BarView 对象传递给 setContentView(), 以显示图形。

注意:

因为 LineChart 与 BarChart 几乎一模一样, 所以这里就不展示其源代码了。把 BarView bv=new BarView(this); 这行代码改成 LineView bv = new LineView(this); 就可以创建 LineChart, 这里最好再把变量 bv 改名为 lv。另外, 不要忘记把 import com.kidroid.kichart.view.BarView; 改成 import com.kidroid.kichart.view.LineView;。

程序清单 7-7 是 PieChart 的源代码。

程序清单 7-7 Piechart 是显示饼状图的 Activity

```

package ca.tutortutor.chartdemo;

import com.kidroid.kichart.ChartActivity;

import com.kidroid.kichart.model.Bitem;

import com.kidroid.kichart.view.PieView;

import android.graphics.Color;

import android.os.Bundle;

public class PieChart extends ChartActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Bundle bundle = getIntent().getExtras();
        float[] data2010 = bundle.getFloatArray("2010");
        float[] data2011 = bundle.getFloatArray("2011");
        Bitem[] items = new Bitem[data2010.length];
        items[0] = new Bitem(Color.RED, "2010.1", data2010[0]);
    }
}

```

```

        items[1] = new Bitem(Color.GREEN, "2010.2", data2010[1]);
        items[2] = new Bitem(Color.BLUE, "2010.3", data2010[2]);
        items[3] = new Bitem(Color.MAGENTA, "2010.4", data2010[3]);
        PieView pv = new PieView(this);
        pv.setTitle("Quarterly Sales (Billions)");
        pv.setItems(items);
        setContentView(pv);
    }
}

```

PieChart 和 BarChart 在从 intent 对象中获得信息方面是类似的。虽然它不使用 data2010 数组, 但该数组可以在饼状图进行升级时使用。

与 BarChart 使用 Aitem 类保存数轴信息不同, PieChart 使用一个 Bitem 保存它的饼状信息。这两个类的最大不同就是传给 Aitem 构造函数的最后一个参数是一个浮点数组, 而传给 Bitem 的是一个浮点值。

程序清单 7-8 是 main.xml 的代码, 这里的布局和部件构成了 ChartDemo 的用户界面。

程序清单 7-8 main.xml 描述了 ChartDemo Activity 的布局

```

<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width = "match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="*">
    <TableRow>
        <TextView android:text="" />
        <TextView android:text="2010"
            android:layout_gravity="center" />
        <TextView android:text="2011"
            android:layout_gravity="center" />
    </TableRow>

    <TableRow>
        <TextView android:text="1st Quarter" />
        <EditText android:id="@+id/data2010_1"
            android:inputType="numberDecimal"
            android:maxLength="1" />
        <EditText android:id="@+id/data2011_1"
            android:inputType="numberDecimal"
            android:maxLength="1" />
    </TableRow>

    <TableRow>
        <TextView android:text="2nd Quarter" />
        <EditText android:id="@+id/data2010_2"
            android:inputType="numberDecimal"
            android:maxLength="1" />
        <EditText android:id="@+id/data2011_2"
            android:inputType="numberDecimal"

```

```

        android:maxLines="1"/>
</TableRow>

<TableRow>
    <TextView android:text="3rd Quarter"/>
    <EditText android:id="@+id/data2010_3"
        android:inputType="numberDecimal"
        android:maxLines="1"/>
    <EditText android:id="@+id/data2011_3"
        android:inputType="numberDecimal"
        android:maxLines="1"/>
</TableRow>

<TableRow>
    <TextView android:text="4th Quarter"/>
    <EditText android:id="@+id/data2010_4"
        android:inputType="numberDecimal"
        android:maxLines="1"/>
    <EditText android:id="@+id/data2011_4"
        android:inputType="numberDecimal"
        android:maxLines="1"/>
</TableRow>

<TableRow>
    <Button android:id="@+id/viewbc"
        android:text="View Barchart"
        android:layout_weight="1"/>
    <Button android:id="@+id/viewlc"
        android:text="View Linechart"
        android:layout_weight="1"/>
    <Button android:id="@+id/viewpc"
        android:text="View Piechart"
        android:layout_weight="1"/>
</TableRow>
</TableLayout>

```

Main.xml 用<TableLayout>标签定义了表格布局，用户界面是按 6 行 3 列布置的。这个标签的 layout_width 和 layout_height 两个属性都是“match_parent”，表示 Activity 会填满整个屏幕。该标签的 stretchColumns 属性的值是“*”，表示布局中每一列的宽度都是一样的。

注意：

弹性列(stretchable column)就是可以扩展宽度以填充空白的列。要指定弹性列，可以将以逗号分隔的从 0 开始编号的整数列表赋给 stretchColumns 属性。例如，“0,1”表示 0 列(最左边的列)和 1 列是可扩展的。“*”表示所有的列都是可扩展的，这会使所有的列的宽度相同。

嵌套在<TableLayout>和</TableLayout>标签中的是一系列<TableRow>标签。每个<TableRow>标签都表示表格布局中的一行，其中的内容可以是若干个 View(例如 TextView 和 EditText)，每个 View 占一列。

注意：

考虑代码的简洁性，这里的字符串是直接保存在 main.xml 文件中的，而没有放入单独的 strings.xml 文件中。可以将这作为一个练习，引入 strings.xml，然后用 strings.xml 中的字符串的引用替换字符串常量。

程序清单 7-9 是应用程序的 AndroidManifest.xml 文件，描述了应用程序及其中的 Activity。

程序清单 7-9 AndroidManifest.xml 将 ChartDemo 应用程序的所有东西组织在一起

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ca.tutortutor.chartdemo"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10"/>
    <application android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
        <activity android:name="ChartDemo"
            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="BarChart"/>
        <activity android:name="LineChart"/>
        <activity android:name="PieChart"/>
    </application>
</manifest>
```

一定要在 manifest 中分别给 BarChart、LineChart 和 PieChart 加入各自的<activity>标签。如果不这样做，会导致运行时出现一个错误对话框，显示应用程序不会再继续工作的信息。

创建一个 ChartDemo 项目(android create project -t 1 -pC:\prj\dev\ChartDemo -a ChartDemo -k ca.tutortutor.chartdemo); 将之前的代码文件(包括 LineChart.java 和 BartChart.java)、main.xml 资源文件和 AndroidManifest.xml 文件都复制到 src/ca/tutortutor/chartdemo 目录下; 将 kiChart-03.jar 复制到 libs 目录; 构建该项目(ant debug); 将它安装到 AVD1 上(adb install-ChartDemo-debug.apk); 启动安装的应用程序。

图 7-5 是 ChartDemo 的主 Activity，每个季度都输入了示例值。

	2010	2011
1st Quarter	100	95
2nd Quarter	200	260
3rd Quarter	300	300
4th Quarter	400	425

View Barchart View Linechart View Piechart

图 7-5 在 ChartDemo 中可以输入 8 个数值，然后选择用柱状图、折线图显示这些数据或者用饼状图显示 2010 列的数据

在输入这些数值后，单击 View Barchart 按钮，启动 BarChart Activity，会显示如图 7-6 所示的柱状图。



图 7-6 BarChart 通过彩色的柱形显示每个数组的数据值

除了显示柱状图，图 7-6 还表明所使用的 kiChart 是一个试用版。要去除这个水印，需要联系 Kidroid.com 获取授权。

单击 View Linechart 按钮启动 LineChart Activity，会显示如图 7-7 所示的折线图。



图 7-7 LineChart 通过带有颜色的线条显示每个数组的数据

最后，单击 View Piechart 按钮启动 PieChartActivity，会显示如图 7-7 所示的饼状图。

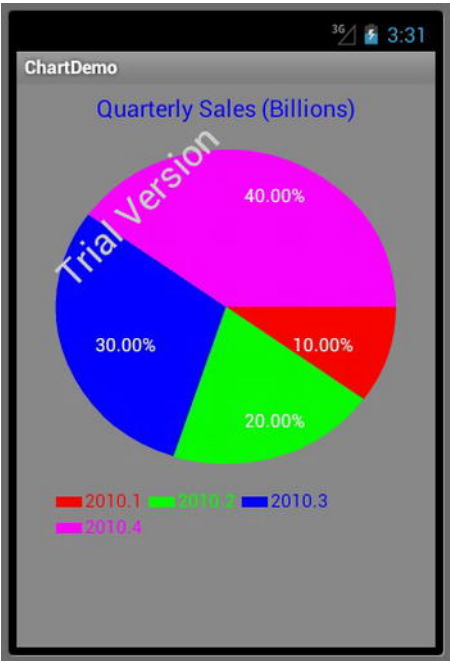


图 7-8 PieChart 通过带有颜色的饼块只显示 2010 数组的数据

7.6 消息推送实战

7.6.1 问题

Google 的 Cloud Messaging for Android (GCM) 框架(<http://developer.android.com/guide/google/gcm/index.html>)用于实现向设备推送消息, 而它自身的一些缺陷使其在实际的消息推送中难有作为。应用程序需要一个更加通用的消息推送解决方案。

Google GCM 的局限性

GCM 是 Google 为 Android 设备开发的基于 XMPP(Extensible Messaging and Presence Protocol)的框架。XMPP 是一种聊天客户端的常用协议。不过仔细研究后会发现, GCM 的一些特点限制了它在应用程序中的使用。

- 至少需要 API Level 8: 随着时间的推移, 这个限制的影响会越来越小, 但无论如何, 运行 Android 2.2 之前版本的设备是无法使用 GCM。
- 设备上需要有 Google 账号和 Google API: GCM 运行在 GTalk 聊天服务创建的 XMPP 信道上。如果用户所使用的 Android 设备没有包含 Google API(也就是没有 GTalk 应用程序), 或者是没有在设备上输入有效的 Google 账号, 应用程序就无法在该设备上注册 GCM 消息推送。
- 应用程序和 GCM 服务器之间的通信是用 HTTP POST 实现的: 从应用程序的服务器端看, GCM 用 HTTP 的 POST 请求逐个处理发送给设备的消息, 随着所发送的消息的数量增加, 这种机制会导致应用程序的速度越来越慢, 因此 GCM 不适用于对时效性要求较高的应用程序。

7.6.2 解决方案

利用 IBM 的 MQTT 库可以在应用程序中实现轻量级的消息推送。IBM 提供了纯 Java 实现的 MQTT 客户端库, 这意味着它完全可以用于 Android 设备, 而且对 API 级别没有任何要求。

MQTT 系统由以下 3 个主要部分组成:

- 客户端应用程序(Client app): 运行在设备上, 在消息代理商注册了某些“话题”消息。
- 消息代理(Message broker): 处理客户端的注册, 根据“话题”将服务器收到的消息分发给各个客户端。
- 服务器端应用程序(Server application): 负责给消息代理发送消息。

消息是根据话题过滤的。话题是树形结构定义的, 用路径字符串表示。客户端可以用合适的路径订阅某个话题或者一组子话题。例如, 我们给应用程序定义了两个话题:

```
examples/one
examples/two
```

客户端可以用完整的路径字符串订阅其中的某个话题。如果客户端要同时订阅这两个话题(或是后来添加到这组中的其他话题), 可以使用下面的路径:

```
examples/#
```

“#”通配符表示应用程序对这组中的所有话题都感兴趣。

在这个范例中, 我们关注如何用 MQTT 库在 Android 设备上实现客户端应用。IBM 为其他组件的开发和测试提供了很好的工具, 我们也会一一介绍。

7.6.3 实现机制

MQTT Java 库可以从 IBM 免费下载, 地址是: www-01.ibm.com/support/docview.wss?uid=swg24006006。下载的压缩包除了库 JAR 还包含示例代码、API Javadoc 和使用文档。

在下载的压缩包中找到 `wmqtt.jar` 文件。这是必须要添加到 Android 项目中的库文件。通常, 需要在项目目录中创建一个 `/libs` 目录来存放 JAR 文件。

为了测试客户端实现, IBM 提供了 RSMB(Really Small Message Broker)。RSMB 可以从这里下载: www.alphaworks.ibm.com/tech/rsmb。

RSMB 是一个跨平台的程序, 其中包括了消息代理和发送消息的命令行工具。IBM 为这个工具提供的授权禁止将其用于生产环境, 所以你得自己写一个, 或者是选用一个开源的实现。不过, 作为开发移动客户端的工具, RSMB 可以完全胜任。

1. 客户端示例

监控所收到的消息是一个长时间的操作, 下面我们看看如何在 `service` 中实现这个功能。

注意:

必须要确保在项目目录中包含 `libs/wmqtt.jar`, 并将其添加到了项目的构建目录中。

程序清单 7-10 展示了 MQTT `service` 示例的源代码。

程序清单 7-10 MQTT Service 示例

```
import com.ibm.mqtt.IMqttClient;
import com.ibm.mqtt.MqttClient;
import com.ibm.mqtt.MqttException;
import com.ibm.mqtt.MqttPersistenceException;
import com.ibm.mqtt.MqttSimpleCallback;

public class ClientService extends Service implements MqttSimpleCallback {

    //定位代理的运行地点
    private static final String HOST = HOSTNAME_STRING_HERE;
    private static final String PORT = "1883";
    //30 分钟不中断 ping
    private static final short KEEP_ALIVE = 60 * 30;
```

```

        //设备的唯一标识
        private static final String CLIENT_ID =
"apress/"+System.currentTimeMillis();
        //我们想关注的话题
        private static final String TOPIC = "apress/examples";

        private static final String ACTION_KEEPLIVE =
"com.examples.pushclient.ACTION_KEEPLIVE";

        private IMqttClient mClient;
        private AlarmManager mManager;
        private PendingIntent alarmIntent;

        @Override
        public void onCreate() {
            super.onCreate();
            mManager = (AlarmManager) getSystemService(Context.ALARM_SERVICE);

            Intent intent = new Intent(ACTION_KEEPLIVE);
            alarmIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

            registerReceiver(mReceiver, new IntentFilter(ACTION_KEEPLIVE));

            try {
                //格式: tcp://hostname@port
                String connectionString = String.format("%s%s@%s",
                    MqttClient.TCP_ID, HOST, PORT);
                mClient = MqttClient.createMqttClient(connectionString, null);
            } catch (MqttException e) {
                e.printStackTrace();
                //无客户端无法继续
                stopSelf();
            }
        }

        @Override
        public void onStart(Intent intent, int startId) {
            //Android2.0 之前设备的回调方法
            handleCommand(intent);
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
            //Android2.0 及以上设备的回调方法
            handleCommand(intent);
            //如果 Android 终止了该服务, 使该服务返回
            return START_STICKY;
        }

        private void handleCommand(Intent intent) {

```

```

        try {
            //创建连接
            mClient.connect(CLIENT_ID, true, KEEP_ALIVE);
            //在这个追踪 MQTT 回调方法
            mClient.registerSimpleHandler(this);
            //订阅一个话题
            String[] topics = new String[] { TOPIC };
            //Qos 为 0, 表示仅激活一次
            int[] qos = new int[] { 0 };
            mClient.subscribe(topics, qos);

            //定时 ping
            scheduleKeepAlive();
        } catch (MqttException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
        unscheduleKeepAlive();

        if(mClient != null) {
            try {
                mClient.disconnect();
                mClient.terminate();
            } catch (MqttPersistenceException e) {
                e.printStackTrace();
            }
            mClient = null;
        }
    }

    //处理收到的远程消息
    private Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            String incoming = (String)msg.obj;
            Toast.makeText(ClientService.this, incoming,
                Toast.LENGTH_SHORT).show();
        }
    };

    //处理 ping 警告以保持连接处于活动状态
    private BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if(mClient == null) {

```

```

        return;
    }
    //ping MQTT service
    try {
        mClient.ping();
    } catch (MqttException e) {
        e.printStackTrace();
    }
    //定时下一次警告
    scheduleKeepAlive();
}
};

private void scheduleKeepAlive() {
    long nextWakeup = System.currentTimeMillis() + (KEEP_ALIVE * 1000);
    mManager.set(AlarmManager.RTC_WAKEUP, nextWakeup, alarmIntent);
}

private void unscheduleKeepAlive() {
    mManager.cancel(alarmIntent);
}

/* MqttSimpleCallback 方法 */

@Override
public void connectionLost() throws Exception {
    mClient.terminate();
    mClient = null;
    stopSelf();
}

@Override
public void publishArrived(String topicName, byte[] payload, int qos,
boolean retained) throws Exception {
    //注意这里的 UI 代码!
    //最好使用 Handler 处理 UI 或者 Context 操作

    StringBuilder builder = new StringBuilder();
    builder.append(topicName);
    builder.append('\n');
    builder.append(new String(payload));
    //将消息传递给 handler
    Message receipt = Message.obtain(mHandler, 0, builder.toString());
    receipt.sendToTarget();
}

/*未使用的方法*/
//没有任何方法绑定到这个 Service
//这个 service 需要显式的指定来启动和停止
@Override

```

```

        public IBinder onBind(Intent intent) { return null; }
    }

```

警告:

因为 Service 需要与远程服务器通信,所以要在 manifest 文件中声明 android.permission.INTERNET 权限,并用<service>标签声明 Service。

继承自 Service 的子类,必须要实现 onBind()方法。本例中,我们并不需要 Binder 接口,因为任何方法都不会直接绑定到该 Service。因此,这个方法只是简单地返回 null。这个 Service 需要通过显式的指令来启动和停止,运行的时间长短是不确定的。

在创建了 Service 之后,用 createMqttClient()方法实例化一个 MqttClient 对象,客户端以字符串的形式获得消息代理主机的地址。连接字符串的格式是 tcp://hostname@port。在这个示例中,选用的端口号是 1883,这是 MQTT 通信的默认端口号。如果选择使用其他的端口号,必须要确认服务器端的实现也是运行在相应的端口上。

接下来,Service 在收到启动命令前会一直处于空闲状态。在收到启动命令后(调用 Context.startService())、onStart()或 onStartCommand()(具体取决于设备上运行的 Android 版本)就会被调用。如果调用的是 onStartCommand(),Service 返回 START_STICKY,这个常量是告诉系统让这个 Service 持续运行,如果因为内存原因关闭的话要自动重启 Service。

在 Service 启动之后,Service 就会注册到 MQTT 消息代理,将客户端 ID 和活跃(keep-alive)时间传递给消息代理。为了让示例便于理解,我们将客户端 ID 定义为服务创建的时间。在实际生产环境中,应该使用 WIFI MAC 地址或 TelephonyManager.getDeviceId()一类的标识符。注意:并不是每台设备都会有这些 ID。

活跃时间这个参数是代理连接客户端的超时时间(单位是秒)。为了避免超时,客户端应该发送消息,或是定时 ping 一下代理,后面会详细讨论。

在启动时,客户端还会订阅一个话题。注意:subscribe()方法的参数是一个数组,客户端仅调用一次该方法就可以订阅多个话题。订阅各个话题的另一个参数是 Qos(quality of service)值。对于移动设备来说,最合适的值是 0。意思是告诉代理,发送消息时不必确认。这样做可以减少代理和设备之间的握手次数。

在连接激活并注册成功之后,所有从远程代理接收的消息都会用其接收的数据调用 publishArrived()。可以在 MqttClient 创建并维护的任何一个后台线程中调用该方法,所以一定不要在这个方法中直接操作主线程。在这个示例中,所有接收到的消息都会传递给一个本地 Handler,确保生成的 Toast 会被发送给主线程并显示出来。

在实现 MQTT 客户端时,还有一个必须要完成的工作是定时 ping 一下代理,以保持连接处于活动状态。Service 注册了 AlarmManager 定时发送广播,以保持连接处于活动状态。即使设备处于休眠状态,也必须执行该任务,所有每次都是用 AlarmManager.RTC_WAKEUP 设置这个警告。每次触发警告时,Service 都只是调用 MqttClient.ping(),然后安排好下一次激活操作。

因为这种操作要不断进行,所以频率不应该太高,在这个示例中我们将其设置为 30 分钟。这个时间要尽可能降低更新频率(降低能耗,节约带宽),同时也要兼顾远程代理发现远程设备不存在并判断超时的延时。

在不需要推送服务时，调用 `Context.stopService()` 会调用 `onDestroy()` 方法。在这里，Service 会关闭 MQTT 连接，删除所有待发送的警告，释放所有资源。`MqttSimpleCallback` 接口的第二个回调方法是 `onConnectionLost()`，表示发生了意外，连接断开。在这种情况下，Service 会自动停止，停止方法与收到手动停止请求的停止方法是一样的。

2. 测试客户端

为了测试消息发送功能，需要在电脑上启动 RSMB。在命令行中切换到 RSMB 所在的文件夹，然后进入操作系统(Windows、Linux、Mac OS X)的子目录。在这里，执行 Broker 命令就可以在电脑上启动代理服务，端口是 localhost: 1883

```
CWNAN9999I Really Small Message Broker
CWNAN9997I Licensed Materials - Property of IBM
CWNAN9996I Copyright IBM Corp. 2007, 2010 All Rights Reserved
...
CWNAN0014I MQTT protocol starting, listening on port 1883
```

此时就可以连接 Service 了，然后发送消息或是注册要接收的消息。为了测试 Service，我们会创建一个简单的 Activity 来启动和停止 Service。

程序清单 7-11 res/menu/home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_start"
        android:title="Start Service" />
    <item
        android:id="@+id/menu_stop"
        android:title="Stop Service" />
</menu>
```

程序清单 7-12 控制 MQTT Service 的 Activity

```
//ClientActivity.java
package com.apress.pushclient;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class ClientActivity extends Activity {
    private Intent serviceIntent;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        serviceIntent = new Intent(this, ClientService.class);
    }
}
```

```

    }

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()) {
            case R.id.menu_start:
                startService(serviceIntent);
                return true;
            case R.id.menu_stop:
                stopService(serviceIntent);
                return true;
        }

        return super.onOptionsItemSelected(item);
    }
}

```

程序清单 7-12 创建了一个用于启动和停止 Service 的 Intent(见图 7-9)。按下 MENU 按钮，选择 Start Service，就会开始 MQTT 连接，设备会注册“apress/examples”主题的消息。

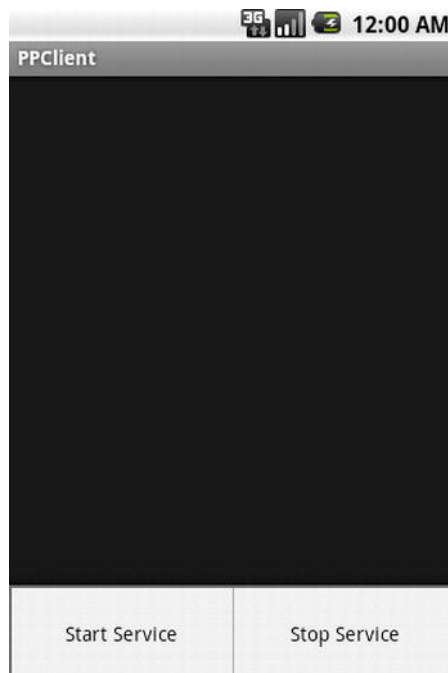


图 7-9 这个 Activity 会控制 Service

注意:

示例 Service 中的 HOST 值需要指向运行 RSMB 的电脑,即使是用模拟器在同一台电脑上测试,这个值也不能设为 localhost! 必须要将这个值设为运行代理电脑的 IP 地址。

在 Android 设备成功地注册了来自代理的推送消息后,打开另一个命令行窗口,切换到执行代理的目录。用另一个命令 `stdinput`,可以连接到正在运行的代理,向设备发送消息。在命令行中输入下面的命令:

```
stdinput aadress/examples
```

这个命令会注册一个客户端,使用与示例相匹配的话题发送消息。下面是执行结果:

```
Using topic aadress/examples
Connecting
```

现在可以输入任何消息,然后按回车,消息就会发送给代理,然后推送到已注册的设备上。多尝试几次,然后用 `Ctrl+C` 退出程序,或结束代理 Service。

提示:

RSMB 还有第三个命令——`stdoutsub`,从本地代理 Service 订阅一系列话题。这个命令可以完全关闭循环,测试问题到底是发生在测试工具上,还是你的 Android 应用程序中。

7.7 使用 Google 的支持包

7.7.1 问题

Google 一直通过升级 SDK 的方法提供新的功能(如 `fragment`)。此外,Google 还可以让你在原本不支持这些功能的老版本 Android 平台上使用这些功能。你希望通过 Google 的这种解决方案让你的应用程序使用它原本不支持的 `fragment` 和/或其他功能。

7.7.2 解决方案

Google 已经想到可以通过引入支持库的方式让老版本的 Android 平台使用新版本的功能。这些静态支持库添加到应用程序中后,应用程序就可以使用老 Android 版本不支持的 API 或者使用一些工具 API(它们并不是框架 API 的组成部分)。

支持库引入了很多不同的新功能,如下:

- `Fragment`(第 1 章介绍的)。
- 推荐的 Android 界面导航模式。
- 支持能够简化 Android Dreams 实现的类并能够向后兼容。Android Dreams(也被称为 Rocket Launcher)最早是在 Android 4.0(Ice Cream Sandwich)中引入的,它提供了一种新的屏保功能。

Google 会在它的“Support Library”页面 <http://developer.android.com/tools/extras/support->

library.html 讨论支持库相关的信息。从这个页面可以看到，每个静态支持库都有指定的最小 API 级别(在低于这个 API 级别的 Android 平台上该支持库将不起作用)。

目前主要有 3 个库：

- Level 4: 对应于 Android 1.6(Donut)。引入这个库的应用程序可以访问除了 Level 7 和 Level 13 之外的所有的功能。
- Level 7: 对应于 Android 2.1(Eclair)。引入这个库的应用程序可以使用一个与 `android.widget.GridLayout` 类似的类，`android.widget.GridLayout` 类是 Level 14 才引入的。
- Level 13: 对应于 Android 3.2(Honeycomb)。引入这个库的应用程序可以使用 Level 13 后才引入的 `fragment` 和 `Android Dreams`。

需要运行 SDK Manager 工具下载和安装支持库。可以通过命令行(如第 1 章所述)或者 Eclipse(在 Window 菜单下选择 Android SDK Manager)运行这个工具。图 7-10 显示了 Android 支持库在 Extras 选项中的入口。

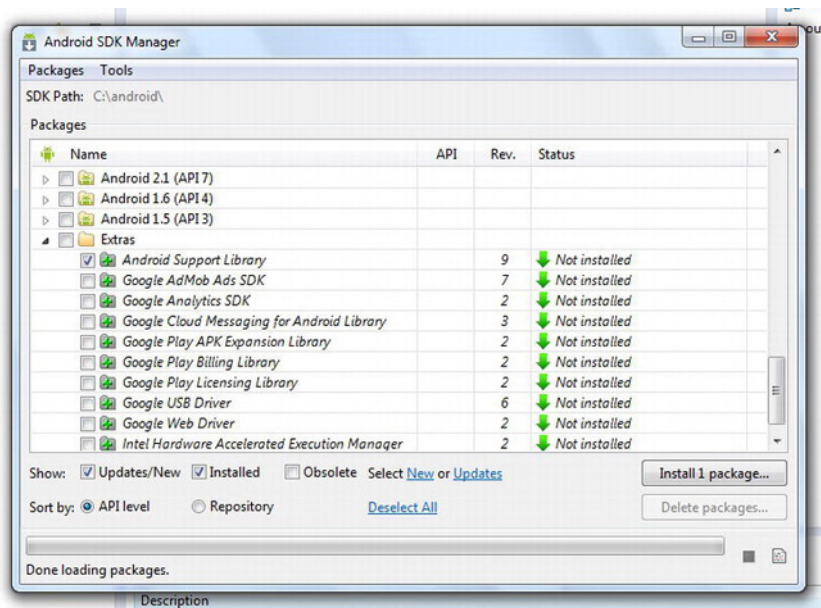


图 7-10 Android 支持库(即支持包)

单击 Install 1 package 按钮，在接下来的 Choose Packages to Install 界面选择 Install 按钮。Support Package Revision 9(编写本书时的版本)就会被安装到<Android_home_directory>/extras/android/support 目录下，该目录下会增加一些文本文件以及 samples、v4、v7 和 v13 目录。

V4 目录中包含一个 `android-support-v4.jar` 文件。同样，v13 目录中会包含一个 `android-support-v13.jar` 文件。而 v7 目录下则会包含一个库工程，该工程的 `libs` 子目录下会包含一个 `androidsupport-v7-gridlayout.jar` 文件，`res` 子目录下也会包含相应的资源文件。

注意：

`android-support-v7-gridlayout.jar` 库文件中会包含 `android.support.v7.widget.GridLayout` 类及相关的 `Space` 和 `ViewGroup` 类。`GridLayout` 是一个 `viewgroup`，它在一个矩形网格中组

织它的子 view。

v7 支持库中引入的 GridLayout 是 android.widget.GridLayout 的兼容替代类, 而 android.widget.GridLayout 是在 API Level 14 中才引入的。和指定 import android.widget.GridLayout; 不同, 在 Level 14 之前的应用程序的代码中只需要指定 import android.support.v7.widget.GridLayout; 即可访问 v7 支持库所提供的等价 GridLayout 类。

7.7.3 实现机制

想要使用 v4 或 v13 库, 将相应的 JAR 文件复制到项目的 libs 目录下即可。在 Eclipse, 还必须得把 JAR 文件添加到项目的构建路径中。这个过程需要右击 JAR 文件, 然后选择 Build Path, 再在弹出的菜单中选择 Add to Build Path。

使用 v7 库项目稍微有点麻烦。正因为如此, 这个范例会通过命令行和 Eclipse 两种方式将 v7 中的项目引入到我们的 UseGridLayout 项目中。程序清单 7-13 是本项目的 UseGridLayout.java 文件的源代码(简洁起见, 除了 AndroidManifest.xml 再没有其他文件了)。

程序清单 7-13 UseGridLayout 中显示了很多按钮组成的网格

```
package ca.tutortutor.usegridlayout;

import android.app.Activity;

import android.os.Bundle;

import android.support.v7.widget.GridLayout;

import android.widget.Button;

public class UseGridLayout extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        GridLayout gl = new GridLayout(this);
        gl.setRowCount(2);
        gl.setColumnCount(2);
        Button btn = new Button(this);
        btn.setText("1");
        gl.addView(btn);
        btn = new Button(this);
        btn.setText("2");
        gl.addView(btn);
        btn = new Button(this);
        btn.setText("3");
        gl.addView(btn);
        btn = new Button(this);
        btn.setText("4");
```

```

        gl.addView(btn);
        setContentView(gl);
    }
}

```

同样为了简洁，程序清单 7-13 对布局和文本资源做了硬编码。实例化 `GridLayout` 后，调用了该类的 `setRowCount(int rowCount)` 和 `void setColumnCount(int columnCount)` 方法建立一个网格布局。最后将网格布局及其子 `view` 设置给了 `Activity`。

1. 使用 Android SDK 创建和运行 UseGridLayout

执行以下命令(为了阅读方便，分两行显示)来创建一个 `UseGridLayout` 项目：

```

android create project -t 2 -p C:\prj\dev\UseGridLayout -a UseGridLayout
                        -k ca.tutortutor.usegridlayout

```

这个命令假设目标平台是 `Android 2.3.3` 并用 `ID 2` 标识。另外还假设你已经为 `Android 2.3.3` 这个目标平台创建一个 `AVD2` 设备。

现在，使用程序清单 7-13 中的内容替换 `src/ca/tutortutor/usegridlayout/UseGridLayout.java` 中的内容。

然后执行以下命令行(为了阅读方便，分两行显示)引用 `GridLayout` 库项目：

```

android update project -t 2 -p C:\prj\dev\UseGridLayout -l
                        ../../..\android\extras\android\support\v7\gridlayout

```

这个命令假设该命令执行时的当前目录是 `C:\prj\dev`。另外还假设 `C:\android\extras\android\support\v7\gridlayout` 就是库项目的位置。

这时，执行以下命令在调试模式下构建这个项目：

此时会输出以下的错误消息：

```

Invalid file: C:\android\extras\android\support\v7\gridlayout\build.xml

```

出现这个错误信息是因为在 `C:\android\extras\android\support\v7\gridlayout` 目录下缺少了 `build.xml` 文件。

想要创建这个文件，切换到这个目录并执行以下命令：

```

android update lib-project -t 2 -p .

```

可以用这个示例的项目替换掉 `lib-project`。如果 `build.xml` 已经创建好了，再次执行 `ant debug`。

如果成功，在项目的 `bin` 子目录下执行以下命令将 `UseGridLayout-debug.apk` 安装到 `AVD2` 上：

```

adb install UseGridLayout-debug.apk

```

最后，启动应用程序。应该会看到如图 7-11 所示的输出信息。

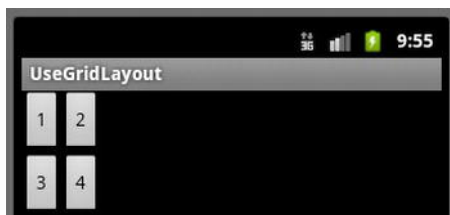


图 7-11 因为 gridlayout 的默认宽度和高度为 WRAP_CONTENT，所以按钮看起来都有点小

2. 使用 Eclipse 创建和运行 UseGridLayout

执行以下步骤使用 Eclipse 创建 UseGridLayout 项目并引入 GridLayout 库项目：

(1) 按照第 1 章介绍的方法安装好 Eclipse 后，首先启动 Eclipse。

(2) 在 File 菜单中选择 Import，在之后出现的 Import 对话框中选择“Existing Android Code Into Workspace”。单击 Next 按钮。

(3) 在之后出现的对话框中，单击 Browse 按钮，然后定位到 C:\android\extras\android\support\v7\gridlayout(或者类似的目录)。单击 OK 按钮退出所有的对话框，单击 Finish 按钮。

(4) 选择 File 菜单中的 New，在之后的弹出菜单中选择 Project。

(5) 在之后出现的 New Project 对话框中，展开向导树中的 Android 节点(如果尚未展开)，选择该节点下的 Android Application Project(如果尚未选择)，单击 Next 按钮。

(6) 在之后出现的 New Android App 对话框中，在 Application Name 一栏输入 UseGridLayout。这个名字同时会在 Project Name 一栏中显示，它表示了 UseGridLayout 项目所在的文件夹/目录。

(7) 在 Package Name 一栏输入 ca.tutortutor.usegridlayout。

(8) 通过 Build Target 选择适合的 Android SDK 目标版本。这也是你即将构建的应用程序可以运行的 Android 平台。如果只安装了 Android 2.3.3 平台，就会只显示这一种平台并选中。

(9) 通过 Minimum SDK，可以设置运行应用程序所需的最低 Android SDK 版本，或者保持默认值(不要选择低于 Level 10 的 SDK 版本)。

(10) 如果需要创建一个自定义的启动图标，就保持“Create custom launcher icon”复选框的选中状态。否则，设置它为未选中状态，这时，会使用系统默认的启动图标。

(11) 因为并不是创建一个库，所以保持 Mark this project as a library 选框的未选中状态即可。

(12) 保持“Create Project in Workspace”复选框的选中状态，单击 Next 按钮。

(13) 在之后出现的 Configure Launcher Icon 界面中，适当地调整自定义的启动图标，单击 Next 按钮。

(14) 在之后出现的 Create Activity 界面中，保持 Create Activity 的选中状态，并确保选中了 BlankActivity，单击 Next 按钮。

(15) 在之后出现的 Blank Activity 界面，在 Activity Name 一栏输入 UseGridLayout。其他设置项保持不动，单击 Finish 按钮。

现在，Eclipse 会在 Package Explorer 窗口中创建一个 UseGridLayout 节点。执行以下步骤设置所需的文件。

(1) 展开 UseMathUtils 节点(未展开的话), 然后展开 src 节点, 最后展开 ca.tutortutor.usegridlayout 节点。

(2) 双击打开 UseGridLayout.java 文件(位于 ca.tutortutor.usegridlayout 下)并用程序清单 7-13 的代码替换 UseGridLayout.java 中的代码。忽略所有的错误信息, 一会它们就会消失。

(3) 右击 UseGridLayout 节点并在弹出的菜单中选择 Properties。

(4) 在之后出现的 Properties for UseGridLayout 对话框中, 选择 Android 分类并单击 Add 按钮。

(5) 在之后出现的 Project Selection 对话框中, 选择 gridlayout, 单击 OK 按钮。

(6) 单击 Apply, 然后单击 OK 按钮关闭 Properties for UseGridLayout。

想要构建和运行这个项目, 选择菜单栏中的 Run, 并在之后出现的下拉菜单中选择 Run(如果出现了 Save Resources 的提示, 单击 OK 按钮即可)。如果接下来出现 Run As 对话框, 选择 Android Application 并单击 OK 按钮。这时 Eclipse 会启动模拟器, 安装项目的 APK 并运行应用程序。图 7-12 显示了输出信息。

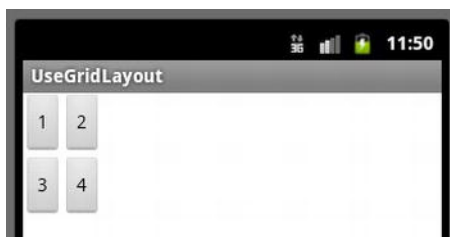


图 7-12 因为是 Eclipse 生成的自定义主题, 所以 UseGridLayout 的用户界面看起来会有点不同

7.8 小结

聪明的 Android 开发者会利用各种各样的第三方库来将他们的应用程序快速发布到应用市场上, 第三方库提供了相对稳定的代码, 可以减少开发时间。

本章的前几个范例介绍了如何创建和使用自定义的库。具体来说, 就好似如何创建和使用 Java 5(或更早版本)API 的库 JAR, 以及 Android 库项目的 JAR。

有时你可能会用到别人开发的库, 以避免重新发明轮子。例如, 如果需要一个简单的绘图库, 那就可以试试 kiChart, 它可以显示柱状图、折线图和饼状图。

如果使用了云计算技术, 可能会使用 Google 的 GCM 框架。不过, 这个框架也有一些缺点(例如, 至少需要 API Level 8), 所以也可以考虑用 IBM 的 MQTT 库在应用中实现轻量级的消息推送。

可以使用 Google 的支持库(也叫做支持包)使用旧 Android 平台所不支持的新 Android API(例如, 具有与 android.widget.GridLayout 相同功能的布局)。

除了附录, 本书的最后一章(第 8 章)将介绍 Android NDK(本地开发工具)和 Renderscript。

第 8 章

使用 Android NDK 和 Renderscript

开发者通常是完全使用 Java 来编写 Android 应用程序。但是，也会出现部分代码(或者是必须)使用其他的语言(尤其 C 或 C++)来编写比较好的情况。Google 提供了 Android 本地开发包(NDK)和 Renderscript 来处理这种情况。

8.1 Android NDK

Android NDK 是 Android SDK 的补充，它提供了一个工具集让用户可以使用 C 和 C++ 这样的本地代码语言实现部分应用程序。对于构建本地 Activity、处理用户输入、使用硬件传感器等操作，NDK 都提供了相应的头文件和库。

很多开发者认为用了 NDK 就可以提升应用程序的性能。但性能提升的同时，由于 Dalvik 虚拟机(VM)编译的 Java 代码与通过 Java 本地接口(JNI)调用的本地代码之间的转换会产生额外开销，这同样会影响到性能，可能导致性能更糟。

注意：

由于 Android 2.2 中，Dalvik 中集成了 Just-In-Time 编译器，致使运行在 Dalvik 上的代码性能有了显著的提高。

NDK 应用在以下场景：

- 应用程序包含有频繁使用 CPU 的代码导致无法分配足够的内存。这种代码的示例包括物理模拟、信号处理、巨大的阶乘计算以及检查巨大的整型质数。Renderscript (本章后面会讨论)可能更合适处理这些示例中一部分。
- 需要更容易地将基于 C/C++ 的源代码移植到你的应用程序中。使用 NDK 可以帮助加快应用程序的开发，并且可以保持应用程序中大多数或全部 C/C++ 代码不变。此外，使用 NDK 可以保持 Android 项目和非 Android 项目之间的代码修改同步。

警告:

在应用程序中加入本地代码时要考虑清楚。因为即使应用程序只是部分使用了本地代码也会增加应用程序的复杂度，增加了调试的难度。

8.1.1 安装 NDK

如果觉得只要部分地使用本地代码就可以提升应用程序的性能，这时候就需要安装 NDK 了。在安装之前，需要注意以下软件和系统要求：

- 需要安装完整 Android SDK(包括相关的组件)。NDK 支持 SDK1.5 和之后的版本。
- 支持以下的操作系统: Windows XP(32 位)、Windows Vista(32 位或者 64 位)、Windows 7(32 或者 64 位)、Mac OS X 10.4.8 或者以后的版本(必须是 x86 处理器)和 Linux(32 位或者 64 位); Ubuntu 8.04 或者其他使用 glibc2.7 或者以后版本的 Linux 发布版本。
- 对于所有的平台，都需要 GNU Make 3.81 或者以后的版本。以前的 GNU Make 版本或许也可以使用但并未测试过。同样，还需要 GNU Awk 或者 Nawk。
- 对于 Windows 平台，需要 Cygwin(1.7 或者高于 1.7)来支持调试。在 NDK Revision 7 之前，也需要使用 Cygwin 的 make 和 awk 工具来构建项目。
- Android NDK 所创建的本地库只能用于那些运行特定最小 Android 平台版本的设备。最小平台版本取决于目标设备的 CPU 架构。表 8-1 详细介绍了 Android 平台版本和特定 CPU 架构的本地代码之间的兼容情况。

表 8-1 本地代码 CPU 架构和与它兼容的 Android 平台之间的对应关系

本地代码使用的 CPU 架构	兼容的 Android 平台版本
ARM, ARM-NEON	Android 1.5 (API Level 3) 以及更高版本
x86	Android 2.3 (API Level 9) 以及更高版本
MIPS	Android 2.3 (API Level 9) 以及更高版本

这些要求意味着可以将通过 NDK 所创建的本地库用在那些运行在 Android 1.5 或者以后的版本的 ARM 设备上的应用程序中。如果要把本地库部署到基于 x86 和 MIPS 的设备上，你的应用程序的目标版本必须是 Android 2.3 或者以后的版本。

- 为了确保兼容性，使用 NDK 所创建的本地库的应用程序必须在 manifest 文件中声明一个<uses-sdk>元素，它的 android:minSdkVersion 属性值要设置为“3”或者更高。例如：

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

- 如果通过 NDK 创建了一个使用了 OpenGL ES API 的本地库，使用该库的应用程序只能部署在表 8-2 所示的最小平台版本的设备上。为了确保兼容性，请确保你的应用程序已经声明了正确的 android:minSdkVersion 属性值。

表 8-2 OpenGL ES 版本、兼容的 Android 平台和 Uses-SDK 之间的对应关系

使用的 OpenGL ES 版本	兼容的 Android 平台	需要的 uses-sdk 属性
OpenGL ES 1.1	Android 1.6 (API Level 4)以及更高版本	android:minSdkVersion="4"
OpenGL ES 2.0	Android 2.0 (API Level 5)以及更高版本	android:minSdkVersion="5"

- 此外，使用了 OpenGL ES API 的应用程序应该在它的 manifest 中声明一个<uses-feature>元素，其中的 android:glEsVersion 属性指定了应用程序需要的最小 OpenGL ES 版本。这样就可以确保 Google Play 只会向那些可以支持你的应用程序设备的用户推荐你的应用程序。

```
<manifest>
  <uses-feature android:glEsVersion="0x00020000" />
  ...
</manifest>
```

- 如果使用 NDK 创建了一个本地库来通过 AndroidAPI 访问 android.graphics.Bitmap 像素缓冲区或者使用本地 Activity，那么使用该库的应用程序只能部署在运行 Android 2.2(API level 8)或者更高版本的设备上。为了确保兼容性，请确保你的应用程序在它的 manifest 中声明了<uses-sdk android:minSdkVersion="8" />元素。

打开浏览器，输入 <http://developer.android.com/tools/sdk/ndk/index.html>，根据你使用操作系统下载以下相应的 NDK 包——撰写本书时，最新版本为 Revision 8b。

- android-ndk-r8b-windows.zip (Windows)
- android-ndk-r8b-darwin-x86.tar.bz2 (Mac OS X: Intel)
- android-ndk-r8b-linux-x86.tar.bz2 (Linux 32/64 位: x86)

下载了你所选择的包后，然后解压，把解压出来的 androidndk-r8b 主目录放到一个合适的位置，例如可以放在和 Android SDK 主目录相同的目录下。

安装 CYGWIN

Cygwin 是一个工具集，它在 Windows 上提供了 Linux 外观和体验环境。使用 Window 系统时，执行以下步骤来安装 Cygwin 1.7 或者更高版本：

- (1) 在浏览器中打开 <http://cygwin.com/>。
- (2) 单击 setup.exe 链接并将它保存到硬盘上。
- (3) 在 Window 系统上运行这个程序开始安装 Cygwi 1.7.16-1 版本(撰写本书时的最新版本)。如果选择了不同的安装路径(默认是 C:\cygwin)，要确保安装路径中没有空格。
- (4) 在 Select Packages 界面上，选择 Devel 分类并找到 Package 列为 make: The GNU version of the “make” utility 的这一项。在该项的 New 列中，单击 Skip；然后这个单词就会变成 3.82.90-1。同时，Bin?列的复选框也会被选中——如图 8-1。

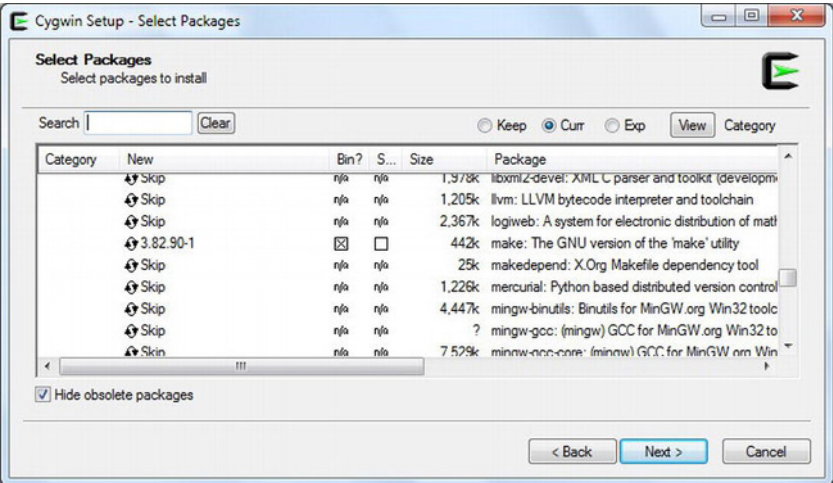


图 8-1 单击 Next 按钮之前，确保 New 列为 3.82.90-1，Bin? 列的复选框是选中的

(5) 单击 Next 按钮继续安装。

安装完成后，可以选择是否在桌面上创建一个默认图标以及在开始菜单中也添加一个图标。选择完成后，单击 Finish 按钮。

如果使用了默认的设置，单击桌面的图标。你会看到图 8-2 所示的 Cygwin 控制台(它是基于 Bash 解释器的)。

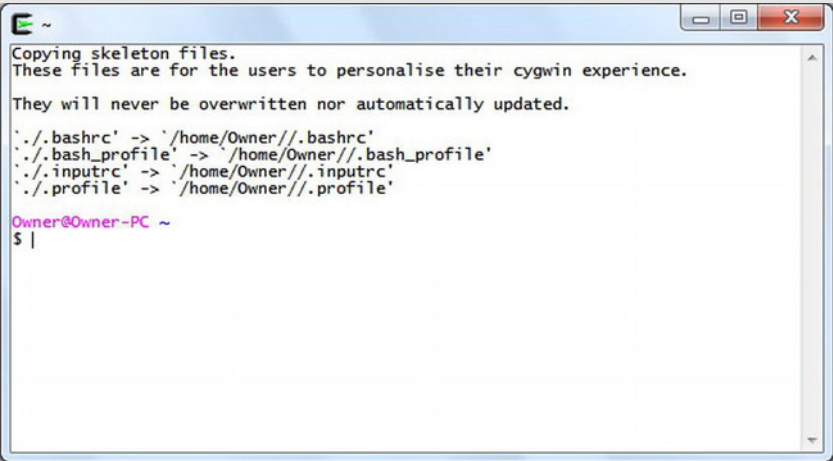
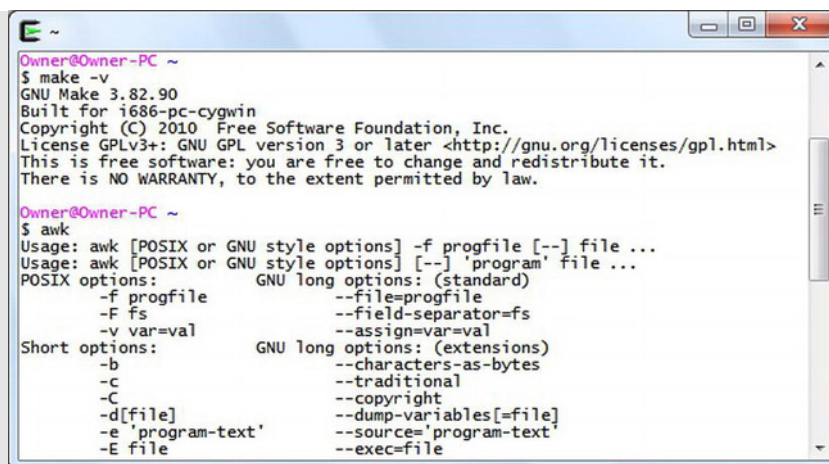


图 8-2 第一次运行时，Cygwin 控制台会显示安装消息

如果想确认一下在 Cygwin 中是否可以使用 GNU Make 3.81 及以后版本和 GNU Awk，输入图 8-3 所示的命令。



```

Owner@Owner-PC ~
$ make -v
GNU Make 3.82.90
Built for i686-pc-cygwin
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Owner@Owner-PC ~
$ awk
Usage: awk [POSIX or GNU style options] -f progfile [--] file ...
Usage: awk [POSIX or GNU style options] [--] 'program' file ...
POSIX options:          GNU long options: (standard)
  -f progfile           --file=progfile
  -F fs                 --field-separator=fs
  -v var=val            --assign=var=val
Short options:          GNU long options: (extensions)
  -b                   --characters-as-bytes
  -c                   --traditional
  -C                   --copyright
  -d[file]             --dump-variables[=file]
  -e 'program-text'    --source='program-text'
  -E file              --exec=file

```

图 8-3 Awk 工具没有显示版本号

关于 Cygwin 的更多信息可以访问 <http://cygwin.com> 以及维基百科的 Cygwin 条目 (<http://en.wikipedia.org/wiki/Cygwin>)。

8.1.2 浏览 NDK

现在已经在你的系统中安装好了 NDK，或许你想浏览一下它的主目录看看 NDK 都提供了哪些东西。下面的列表展示了 Window 系统 NDK 主目录中的目录和文件情况：

- build 中是 NDK 构建系统的相关文件。
- docs 中是 HTML 格式的 NDK 说明文档。
- platforms 中是一些子目录，这些子目录是 Android SDK 已安装的各个 Android 平台的头文件和共享库。
- prebuilt 中是一些工具(尤其是 make.exe 和 awk.exe)，让你在没有 Cygwin 的情况下也可以编译 NDK 源代码。
- samples 中包含了用于演示 NDK 各个方面功能的示例应用。
- sources 中是源代码和各种已经预编译好的共享库，如 cpufeatures(用于检测目标设备的 CPU 类型及其所支持的特性)和 stlport(多平台的 C++标准库)。在 Android NDK 1.5 上，要求开发者将本地代码库项目放在这个目录中。从 Android NDK1.6 开始，本地代码库则存储在 Android 项目目录的子目录 jni 中。
- tests 中包含了用于 NDK 自动测试的脚本和源代码。在测试自行构建的 NDK 时很有用。
- Toolchains 包含了编译器、连接器、和其他的一些工具，这些工具用于生成 Linux、OS X 和 Windows(带 Cygwin)系统上的 ARM(Advanced RISC Machine,它是 Android 所使用的 CPU，详见 http://en.wikipedia.org/wiki/ARM_architecture)二进制文件。
- documentation.html 是 NDK 文档的入口。
- GNUMakefile 是 GNU Make 使用的默认 makefile。
- ndk-build 是用于简化生成机器码的 shell 脚本。

- `ndk-build.cmd` 是一个 Windows `cmd.exe` 脚本, 它调用了 `prebuilt\windows\bin\make.exe` 这个可执行文件。
- `ndk-gdb` 是一个 shell 脚本, 它可以很容易为 NDK 生成的机器码启动一个本地调试会话(在 Window 系统中, 需要使用 Cygwin 来运行这个脚本)。
- 当使用 `adb logcat` 生成一些栈的追踪信息后, 使用 `ndk-stack.exe` 可以过滤这些信息。它还可以用相应的值替换一个共享库中的任意地址。实际上, 它可以让你得到更多可读的 `crash` 信息。
- `README.TXT` 是 NDK 的欢迎信息, 它列出了各种文档文件来描述当前版本(和更多版本)的变动情况。
- `RELEASE.TXT` 包含了 NDK 的发布编号(r8b)。

Platforms 目录中的每个子目录中都包含了稳定的本地 API 的头文件。Google 会保证以后每个平台版本中都会支持以下的 API(详见 <http://developer.android.com/tools/sdk/ndk/overview.html#tools>):

- Android 日志(liblog)
- Android 本地应用程序 API
- C 库(libc)
- C++基本库(stlport)
- JNI 接口 API
- 数学库(libm)
- OpenGL ES 1.1 和 OpenGL ES 2.0 (3D 图形库)API
- OpenSL ES 本地音频库 API
- 为 Android2.2 及以后版本提供像素缓冲区的访问(libjngraphics)。
- Zlib 压缩(libz)

警告:

这个列表中没有列出的本地系统库都是不稳定的, 在以后的 Android 平台中可能会发生变化。请勿使用。

8.1.3 来自 NDK 的问候

或许熟悉 NDK 编程最简单的方式就是创建一个小应用程序, 调用一个返回 Java String 的本地函数。例如, 程序清单 8-1 的单独 Activity 的应用程序 `NDKGreetings` 就调用了本地 `getGreetingMessage()`方法来返回一个问候消息, 并通过一个对话框显示该信息。

程序清单 8-1 接收 NDK 的问候

```
package ca.tutortutor.ndkgreetings;

import android.app.Activity;
import android.app.AlertDialog;

import android.os.Bundle;
```

```

public class NDKGreetings extends Activity
{
    static
    {
        System.loadLibrary("NDKGreetings");
    }

    private native String getGreetingMessage();

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        String greeting = getGreetingMessage();
        new AlertDialog.Builder(this).setMessage(greeting).show();
    }
}

```

程序清单 8-1 的 NDKGreetings 类展示了使用了本地代码的应用程序的三个重要特点:

- 本地代码存储在一个外部库中, 在调用该库的代码之前, 必须先加载它。通常是在类加载时, 调用 `System.loadLibrary()` 来加载库。这个方法只有一个字符串参数, 就是去掉 `lib` 前缀和 `.so` 后缀之后的库文件名。本例中, 真正的库文件名是 `libNDKGreetings.so` (如果无法加载库, 会抛出一个 `java.lang.UnsatisfiedLinkError` 类的实例的异常, 它会导致 Android 终止应用程序)。
- 需要声明一个或者多个与库中函数对应的本地方法。本地方法的前缀是它的返回类型, 之后是 `native` 关键字。
- 本地方法的调用方式和其他 Java 方法的调用方式是一样的。实际上, Dalvik 会去确保库中相应的原生函数可以被正确地调用。

程序清单 8-2 是原生代码库中的 C 代码, 它通过 JNI 实现了 `getGreetingMessage()` 方法。

程序清单 8-2 实现一个对 Dalvid 的问候反馈

```

#include <jni.h>

jstring
Java_ca_tutortutor_ndkgreetings_NDKGreetings_getGreetingMessage(JNIEnv
    nv* env,
                                                                    jobject this)
{
    return(*env)->NewStringUTF(env, "Greetings from the NDK!");
}

```

程序清单 8-2 首先指定了一个 `#include` 预处理指令, 在编译源代码时首先会包含 `jni.h` 头文件的内容。这个文件指定了各种各样的 JNI 常量、类型、和函数原型。

程序清单 8-2 然后声明了与程序清单 8-1 的 `getGreetingMessage()` 方法对应的本地函数。该本地函数头揭示了以下重点:

- 本地函数的返回值为 `jstring`。这种类型是在 `jni.h` 中定义的，表示本地代码层中的 `Java.lang.String` 对象。
- 函数的名字必须以 Java 包和类名开头，它标识了声明相关本地方法的位置。
- 函数的第一个参数 `env` 的类型为 `JNIEnv` 指针。`JNIEnv` 是 `jni.h` 中定义的一个 C 结构体，标识该 JNI 函数可以在 Java 中被调用并与之交互。
- 函数的第二个参数 `this` 的类型为 `jobject`。这个类型是在 `jni.h` 中定义的，它表示了本地代码层任意的一个 Java 对象。传递给该参数的值是一个隐式的 `this` 实例，是 Java VM 传递给所有 Java 实例方法的值。

这个函数会将 `env` 参数解引用用来调用 `NewStringUTF()` JNI 函数。`NewStringUTF()` 将它的第二个参数(一个 C 字符串)转换成相应的 `jstring`(这个字符串是用 Unicode UTF 编码标准编码的)，并将这个 Java 字符串返回，最后返回给 Java 层。

注意：

在 C 语言中使用 JNI 时，想要调用 JNI 函数就得解引用 `JNIEnv` 参数(例如 `*env`)。同样，必须把 `JNIEnv` 作为第一个参数传递给 JNI 函数。相反，C++ 中则不需要这么麻烦：没必要非得解引用 `JNIEnv` 参数，也不需要把它作为第一个参数传递给 JNI 函数。例如，程序清单 8-2 中用 C 编写的函数调用 `(*env)->NewStringUTF(env, "Greetings from the NDK!")` 在 C++ 中就可以写成 `env->NewStringUTF("Greetings from the NDK!")`。

1. 通过 Android SDK 构建并运行 NDKGreetings

想要使用 Android SDK 构建 NDKGreetings，首先使用 SDK 的 Android 工具创建一个 NDKGreetings 项目。假如使用的是 Windows 系统，NDKGreetings 项目会存储在 `C:\prj\dev` 这样的目录下(即 `C:\prj\dev\NDKGreetings`)，假设目标平台为 Android 4.1，对应的整数 ID 是 1(执行 `android list target` 可获得相应的 ID)，执行以下命令来创建 NDKGreetings：

```
android create project -t 1 -p C:\prj\dev\NDKGreetings -a NDKGreetings
-k ca.tutortutor.ndkgreetings
```

这个命令会在 `C:\prj\dev\NDKGreetings` 下创建各种目录和文件。例如，`src` 目录的结构为 `ca\tutortutor\ndkgreetings`，最终的 `ndkgreetings` 目录包含了 `NDKGreetings.java` 的框架代码文件。使用程序清单 8-1 的内容替换这个文件的内容。

然后，在 `C:\prj\dev\NDKGreetings` 中创建一个 `jni` 目录，将程序清单 8-2 的内容复制到 `C:\prj\dev\NDKGreetings\jni\NDKGreetings.c` 中。同样，将程序清单 8-3 的内容复制到 `C:\prj\dev\NDKGreetings\jni\Android.mk` 中，这是一个用于构建 `libNDKGreetings.so` 库的 GNU 生成文件(在 NDK 文档中有详细介绍)。

程序清单 8-3 NDKGreetings 的 Make 文件

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := NDKGreetings
```

```
LOCAL_SRC_FILES := NDKGreetings.c
include $(BUILD_SHARED_LIBRARY)
```

在 C:\prj\dev\NDKGreetings 目录下执行以下命令：

```
\android-ndk-r8b\ndk-build
```

这个命令会执行(在 Windows 上)ndk-build.cmd 脚本来构建库。构建成功后，会输出下面的消息：

```
Compile thumb : NDKGreetings <= NDKGreetings.c
SharedLibrary : libNDKGreetings.so
Install : libNDKGreetings.so => libs/armeabi/libNDKGreetings.so
```

这段输出表示 libNDKGreetings.so 位于 NDKGreetings 项目的 libs 子目录的 armeabi 子目录中。

注意：

如果看到的是 “No rule to make target”，而不是以上的输出，原因很有可能是在 Android.mk 中有多余的空格。

此外，也可以使用 Cygwin(假设如之前讨论的那样，已经安装好了 Cygwin)来完成这个工作。运行 Cygwin(如果还未运行的话)，在 Cygwin 的命令行窗口，将当前目录切换到 C:\prj\dev\NDKGreetings。如图 8-4 所示。

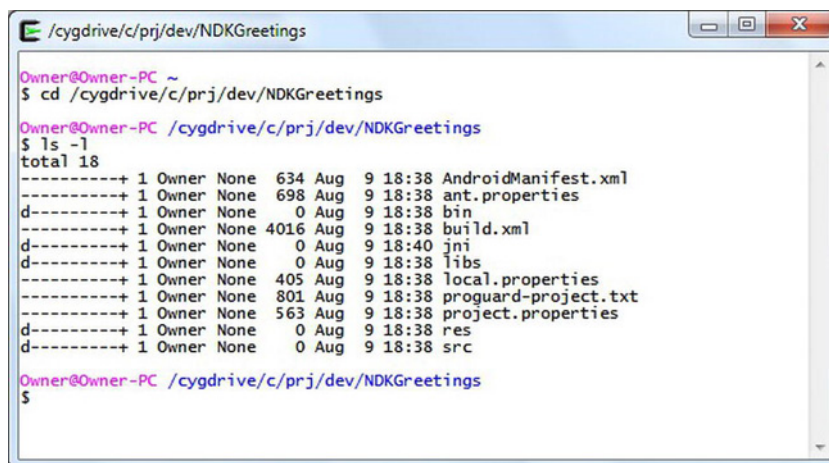


图 8-4 指向 C:\prj\dev\NDKGreetings 的路径以/cygdrive/c 为前缀

假设 NDK 的主目录为 android-ndk-r8b 并且位于 C 盘的根目录下，执行以下命令(在 Cygwin 中)来构建库：

```
../../../../android-ndk-r8b/ndk-build
```

你应该看到的是和之前显示一样的输出消息。

假设当前目录为 C:\prj\dev\NDKGreetings，执行以下命令(通过 Cygwin 的 shell 或者正常的 Windows 命令窗口)来创建 NDKGreetings-debug.apk：

```
ant debug
```

这个 APK 文件会放在 NDKGreetings 项目目录的 bin 子目录下。想要确认 libNDKGreetings.so 在这个 APK 中，在 bin 下执行以下命令：

```
jar tvf NDKGreetings-debug.apk
```

在 jar 命令的输出中应该可以看到 lib/armeabi/libNDKGreetings.so。

想要确认应用程序是否能够正常运行，启动模拟器，这一步可以通过在命令行执行以下命令来完成。

```
emulator -avd AVD1
```

这个命令的前提是你已经按照第 1 章的内容创建 AVD1 设备配置。

通过以下命令在模拟器上安装 NDKGreetings-debug.apk：

```
adb install NDKGreetings-debug.apk
```

这个命令的前提是 adb 已经加入到 path 环境变量中，而且当前目录是 bin。

当 adb 显示 NDKGreetings-debug.apk 已经成功安装后，回到应用启动器界面，单击 NDKGreetings 图标。结果如图 8-5 所示。

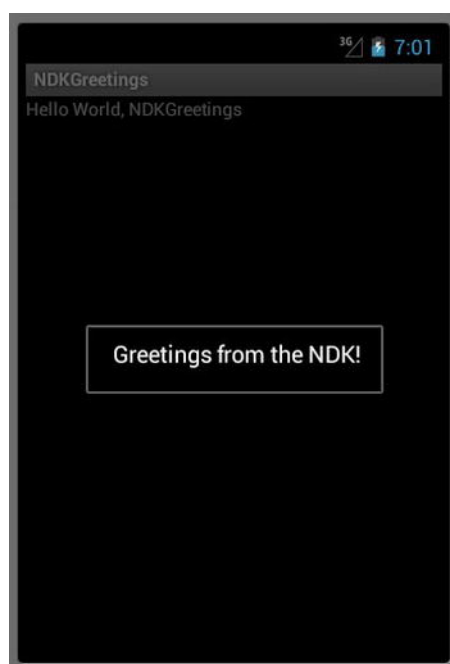


图 8-5 按下 Esc 键(在 Windows 上)结束这个对话框

对话框会显示“Greetings from the NDK!”信息，该信息通过调用本地代码库中的本地函数获得的。在屏幕上方，还有一个若隐若现的“Hello World, NDKGreetings”信息。这个信息来自 android 工具创建的项目默认文件 main.xml。

2. 在 Eclipse 中构建和运行 NDKGreetings

想要使用 Eclipse 构建 NDKGreetings, 首先要想在第 1 章范例 1-10 中所介绍的那样创建一个新的 Android 项目。为了方便起见, 可以使用下面的步骤创建项目:

(1) 如果还未启动 Eclipse 的话, 先启动 Eclipse

(2) 在 File 菜单中选择 New, 在弹出的菜单中选择 Project

(3) 在 New Project 对话框中, 展开向导树中的 Android 子节点(如果必要的话), 选择该节点下(如果必要的话)的 Android Application Project 分支, 然后单击 Next 按钮。

(4) 在 New Android App 对话框的 Application Name 一栏输入 NDKGreetings。这个名字同样会显示在 Project Name 中, 它标识了 NDKGreetings 项目所在的文件夹/目录。

(5) 在 Package Name 一栏输入 ca.tutortutor.ndkgreetings。

(6) 通过 Build SDK, 选择合适的目标 Android SDK 版本。这个选择标识了你想要构建应用程序的 Android 平台版本。假设你只安装了 Android 4.1, 那这里只会出现这一个目标并且已经被选中。

(7) 通过 Minimum SDK, 可以选择应用程序所需的最小 Android SDK 版本, 也可以使用默认设置。

(8) 如果想要自定义启动器图标的话, 请选中 Create custom launcher icon 复选框。否则, 会使用默认的启动器图标。

(9) 保持 Mark this project as a library 复选框的未选中状态, 因为你并没有打算创建一个库。

(10) 保持 Create Project in Workspace 复选框的未选中状态, 然后单击 Next 按钮。

(11) 在接下来的 Configure Launcher Icon 界面中, 对自定义启动图标做相应的调整, 然后单击 Next 按钮。

(12) 在随后的 Create Activity 界面中, 保持 Create Activity 复选框的选中状态, 选中 BlankActivity, 单击 Next 按钮。

(13) 在随后的 New Blank Activity 界面中, 在 Activity Name 一栏输入 NDKGreetings, 并在 Layout Name 一栏填写 main, 保持其他设置不变, 然后单击 Finish 按钮。使用 Eclipse 的 Package Explorer 找到 NDKGreetings.java 的源文件。双击该文件, 在弹出的编辑窗口中, 使用程序清单 8-1 中的内容替换该文件中的框架代码。

打开 Package Explorer, 在 NDKGreetings 项目下创建一个 jni 文件夹, 在该文件夹下创建一个 NDKGreetings.c 文件, 将它的内容使用程序清单 8-2 的内容填充, 再向 jni 文件夹下添加一个新的 Android.mk 文件, 加入程序清单 8-3 的内容。

这时, 可以使用 Cygwin 创建库文件, 或者创建一个构建器来帮助你完成这个工作。想要使用 Cygwin, 如果还没启动的话, 先启动 Cygwin, 使用 cd 命令切换到项目的目录(例如, cd /cygdrive/c/users/owner/workspace/NDKGreetings)。顺利的话, NDKGreetings 项目目录的 libs 子目录下应该多出一个 armeabi 子目录, 它会包含一个 libNDKGreetings.so 库文件。

执行以下步骤来创建一个构建器:

(1) 右击 NDKGreetings 节点, 在弹出的菜单中选择 Properties。

(2) 在弹出的 Properties for NDKGreetings 对话框中选择 Builders。

(3) 在 Builders 面板中, 单击 New 按钮。

(4) 在弹出的 Choose configuration type dialog 的对话框中, 选择 Program 并单击 OK 按钮。

(5) 在弹出的 Edit Configuration 对话框中, 随意填写一个构建器的名称(或者使用默认名称), 在 Location 一栏输入 C:\android-ndk-r8b\ndk-build.cmd(或者你自己定义的路径), 在 Working Directory 一栏输入 \${workspace_loc:/NDKGreetings}, 单击 OK 按钮关闭该对话框。

(6) 单击 OK 按钮关闭 Properties for NDKGreetings 对话框。

想要在 Eclipse 中运行 NDKGreetings, 在菜单栏中选择 Run, 然后在下拉菜单中选择 Run。如果出现了 Run As 对话框, 选择 Android Application 并单击 OK 按钮。Eclipse 会启动 AVD1 设备模拟器, 安装 NDKGreetings.apk, 并运行该应用程序, 输出结果如图 8-6 所示。



图 8-6 因为是 Eclipse 生成的自定义主题, 所以 NDKGreetings 的界面看起来会有点不一样

8.1.4 NDK 示例

NDK 安装目录中的 samples 子目录中有几个示例应用程序, 它们演示了 NDK 不同方面的功能。

- **bitmap-plasma:** 演示了如果在本地代码中访问 Android 的 `android.graphics.Bitmap` 对象的像素缓冲区, 并利用该功能生成了经典的电浆(plasma)特效。
- **hello-gl2:** 演示如何使用 OpenGL ES 2.0 的定点和片段着色器渲染一个三角形(如果使用 Android 模拟器运行这个应用程序, 你会得到一个应用程序意外终止的错误信息, 这是因为模拟器不能模拟支持 OpenGL ES 2.0 硬件的设备)。

- **hello-jni**: 演示了使用在共享库中实现的本地方法加载一个字符串, 并显示在用户界面上。这个应用程序和 `NDKGreetings` 很相似。
- **hello-neon**: 演示了如何使用 `cpufeatures` 库在运行时检查 CPU 的特性。如果 CPU 支持 NEON(这是 ARM 架构的 SIMD 指令集的商业名称), 还会使用 NEON 指令。具体来说, 这个应用程序实现了两个版本的 FIR 过滤循环的基准测试, 一个是 C 版本的, 另一个是针对支持 NEON 的设备做过优化的。
- **native-activity**: 演示了如何使用 `native-app-glue` 静态库来创建一个本地 activity(一个完全用本地代码实现的 activity)。
- **native-audio**: 演示了如何使用本地方法通过 OpenSL ES 播放音乐。
- **native-plasma**: 用本地 activity 实现的 `bitmap-plasma` 版本。
- **san-angeles**: 通过本地 OpenGL ES API 渲染 3D 图形, 同时用 `android.opengl.GLSurfaceView` 对象管理 activity 的生命周期。
- **two-libs**: 动态加载共享库, 并调用库提供的本地方法。在这个示例中, 调用的方法是在一个静态共享库中实现的, 该静态库由共享库导入。

可以使用 Eclipse 来构建这些应用程序。例如, 执行以下步骤来构建 `san-angeles`:

- (1) 如果还未启动 Eclipse 的话, 先启动 Eclipse。
- (2) 在 File 菜单中选择 New, 在弹出的菜单中选择 Project。
- (3) 在弹出的 New Project 对话框中, 在向导树中展开 Android 节点(如果需要的话)。

在该节点下选择 Android Project from Existing Code, 然后单击 Next 按钮。

- (4) 在弹出的 Import Projects 界面中, 单击 Browse 按钮。

(5) 在弹出的 Browse for Folder 对话框中, 选择 NDK 的 `san-angeles` 目录, 它位于 `samples` 目录下。单击 OK 按钮关闭对话框。

(6) 这时回到了 Import Projects 界面, 选中 Copy projects into workspace 复选框并单击 Finish 按钮。这时会在 Package Explorer 出现一个 `com.example.SanAngeles.DemoActivity` 节点。另外, 在 workspace 中还会出现 `com.example.SanAngeles.DemoActivity` 项目目录。这个目录包含了 NDK 的 `san-angeles` 项目的一个单独的副本。

- (7) 右击 `com.example.SanAngeles.DemoActivity` 节点, 在弹出菜单中选择 Properties。

(8) 在弹出的 Properties for `com.example.SanAngeles.DemoActivity` 对话框中, 选择 Builders。

- (9) 在弹出的 Builders 界面中, 选择 New 按钮。

- (10) 在 Choose configuration type 对话框中, 选择 Program 并单击 OK 按钮。

(11) 在弹出的 Edit Configuration 对话框中, 随意填写一个构建器的名称(或者使用默认名称), 在 Location 一栏输入 `C:\android-ndk-r8b\ndk-build.cmd`(或者你自己定义的路径), 在 Working Directory 一栏输入 `${workspace_loc:/com.example.SanAngeles.DemoActivity}`, 单击 OK 按钮关闭该对话框。

- (12) 单击 OK 按钮关闭 `com.example.SanAngeles.DemoActivity` 对话框。

在 Package Explorer 中选择 `com.example.SanAngeles.DemoActivity` 节点, 在菜单栏中选择 Run, 然后在下拉菜单中选择 Run。如果出现了 Run As 对话框, 选择 Android Application 并单击 OK 按钮。如果碰到了对话框说你的项目有错误的话, 就关闭这个对话框并再次选

择 Run。

这次, Eclipse 应该会启动一个第 1 章所创建的 AVD1 设备模拟器。它会在该模拟器上安装 DemoActivity.apk 并运行它。解锁屏幕后, 你应该会看到一个连续移动的屏幕, 屏幕的内容与图 8-7 所示的内容相似(出现这个画面会花费一点时间)。

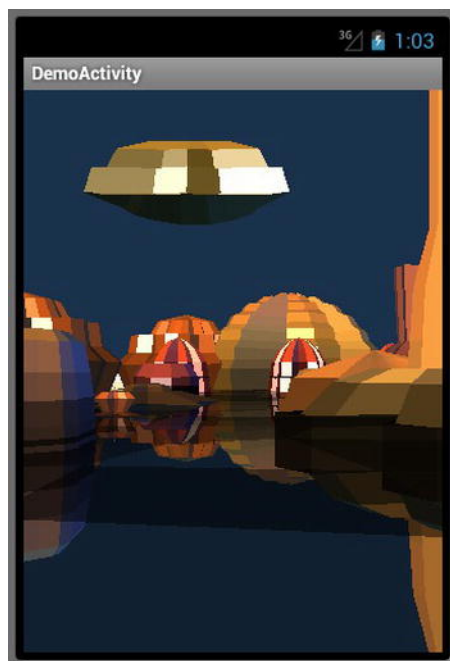


图 8-7 DemoActivity 将你带入三维世界

8.2 发现本地 Activity

8.2.1 问题

Android 支持本地 Activity, 你想进一步了解它。

8.2.2 解决方案

本地 activity 就是完全用本地代码实现的 activity。最早出现在 Android 2.3(API Level 9) 中的 `android.app.NativeActivity` 类, NDK 的版本 Revision 5 提供了开发支持, 本地 activity 让你在没有任何 Java 代码的情况下, 使用 C/C++ 来开发应用程序。

注意:

`NativeActivity` 实例等价于使用 JNI 调用本地代码的 `android.app.Activity`。

8.2.3 实现机制

`NativeActivity` 是一个辅助类，让你可以编写一个纯粹的本地 `activity` 进而开发纯粹的本地应用程序。它会处理 Android 框架层与你的本地代码之间的通信。你不需要创建它的子类或者调用它的方法。相反，只需要创建你的本地应用程序并在 `AndroidManifest.xml` 中声明它为本地即可。

本地 `activity` 并没有改变什么，Android 应用程序还是运行在它们自己 VM 中，与其他应用程序是隔离开来的。正因为如此，依然可以通过 JNI 方法 Android 框架层 API。此外，还可以使用一些本地接口来访问传感器、输入事件和资源等等。

注意：

想要了解本地 `activity` 可以访问的 API，请查看本章之前所示的稳定本地 API 的列表。

NDK 提供了两种方式来开发本地 `activity`：

- **Low-level:** `native_activity.h` 头文件(位于 NDK 目录的子目录 `platforms/android-9/arch-arm/usr/include/android` 下)定义了 `NativeActivity` 类的本地版本。它包含了在创建本地 `activity` 时所需的回调接口和数据结构。因为应用程序是在主线程中处理回调，所以需要回调的实现不能阻塞。如果阻塞的话，你会得到“`Application Not Responding`”的错误，这是因为在回调返回之前，主线程是没有反应的。查看 `native_activity.h` 中的注释可以获得更多相关的信息。
- **High-level:** `android_native_app_glue.h` 头文件(位于 NDK 主目录的 `sources/android/native_app_glue` 子目录中)定义了一个静态辅助库，它会在 `native_activity.h` 之上构建。它使用另一个线程去处理回调和输入事件。这个线程可以防止回调阻塞主线程，使回调的实现添加了一定的灵活性，所以你会发现这种编程模式很容易实现。对于 `android_native_app_glue.c` 源文件(位于同一目录中)，如果需要改变它的功能的话，可以直接修改它。查看 `android_native_app_glue.h` 中的注释可以获得更多相关的信息。

在下面两个范例中，你会了解更多关于本地 `activity` 的知识，这两个范例会演示如何在 `low-level` 和 `high-level` 环境下开发相似的本地 `activity`，它们都是使用 Android SDK 和 Eclipse 开发的。

8.3 开发 Low-Level 本地 Activity

8.3.1 问题

想要学习如何开发基于 `native_activity.h` 头文件的 `low-level` 本地 `activity`。

8.3.2 解决方案

如同常规 Android 应用程序项目一样创建一个 `low-level` 本地 `activity` 项目。然后适当

修改它的 AndroidManifest.xml 文件，并在项目目录中添加 jni 子目录，该子目录中包含了本地 activity 的 C/C++代码以及 Android.mk make 文件。

修改后的 AndroidManifest.xml 文件与常规的 AndroidManifest.xml 有点不同，体现在以下几方面：

- 在<application>元素之前是<uses-sdk android:minSdkVersion="9"/>，本地 activity 要求至少是 API Level 9。
- 因为本地 activity 不需要源代码，所以会在<application>标签中添加 android:hasCode="false"属性。
- <activity>元素的 android:name 属性的值为“android.app.NativeActivity”。当 Android 看到这个值后，就会在使用本地 activity 库中访问入口。
- 在<intent-filter>元素之前是一个<meta-data>元素。<meta-data>中指定了一个 android:name="android.app.lib_name"属性和值为本地 activity 的库名(没有 lib 前缀和.so 后缀)的 android:value 属性。

本地 activity 的 C/C++源代码中必须定义以下入口方法：

```
void ANativeActivity_onCreate(ANativeActivity* activity, void* savedState,
                             size_t savedStateSize)
```

这个方法有以下参数：

- activity：它是 ANativeActivity 结构体的地址。ANativeActivity 是在 NDK 的 native_activity.h 头文件中定义的，它声明了不同的成员变量，包括 callbacks(回调函数的指针数组；可以设置这些指针指向你自己的回调)、internalDataPath(应用程序内部数据目录的路径)、externalDataPath(应用程序外部【可拆卸/安装】数据目录的路径)、sdkVersion(平台的 SDK 版本号)和 assetManager(一个与应用程序的 android.content.res.AssetManager 类似的本地实例的指针，用于访问应用程序 APK 文件中的二进制资源)。
- savedState：这个是你的 activity 之前保存的状态。如果 activity 是通过之前所保存的实例来实例化的，那么 savedState 值为非 null 并且指向所保存的数据。如果需要后面访问这个数据，必须创建一个该数据的副本，因为函数返回时，分配给 savedState 的内存会被回收。
- savedStateSize：这个是 savedState 所指定的数据大小(单位为 byte)。

注意：

当启动一个基于本地 activity 的应用程序时，就会创建一个 android.app.NativeActivity 类的实例。该实例的 onCreate(Bundle)方法会通过 JNI 调用 void ANativeActivity_onCreate(ANativeActivity*,void*, size_t)。

void ANativeActivity_onCreate(ANativeActivity*, void*, size_t)应该覆写所有需要的回调。同时，需要马上创建一个线程来处理对输入事件的响应，也可以防止发生“Application Not Responding”错误。

注意:

`void ANativeActivity_onCreate(ANativeActivity*, void*,size_t)`和你的回调方法不可以阻塞; 否则, 会发生 “Application Not Responding” 错误。

最后, `Android.mk` 文件几乎和之前看到的是一样的。此外, 这个文件很有可能会包含一个 `LOCAL_LDLIBS`, 它标识了所有需要的库(毫无疑问, 这些库都会包含标准的 `libandroid.so` 库)。

8.3.3 实现机制

使用 `LLNADemo` 项目来演示 low-level 本地 activity。

程序清单 8-4 展示了这个项目中唯一的 `llnademc.c` 源文件的内容。

程序清单 8-4 从底层的视角体验一个本地 Activity

```
#include <android/log.h>
#include <android/native_activity.h>
#include <pthread.h>

#define LOGI(...) ((void)__android_log_print(ANDROID_LOG_INFO, \
                                              "llnademc", \
                                              __VA_ARGS__))

AInputQueue* _queue;
pthread_t thread;
pthread_cond_t cond;
pthread_mutex_t mutex;
static void onConfigurationChanged(ANativeActivity* activity)
{
    LOGI("ConfigurationChanged: %p\n", activity);
}

static void onDestroy(ANativeActivity* activity)
{
    LOGI("Destroy: %p\n", activity);
}

static void onInputQueueCreated(ANativeActivity* activity, AInputQueue*
queue)
{
    LOGI("InputQueueCreated: %p -- %p\n", activity, queue);
    pthread_mutex_lock(&mutex);
    _queue = queue;
    pthread_cond_broadcast(&cond);
    pthread_mutex_unlock(&mutex);
}

static void onInputQueueDestroyed(ANativeActivity* activity, AInputQueue*
```

```

queue)
{
    LOGI("InputQueueDestroyed: %p -- %p\n", activity, queue);
    pthread_mutex_lock(&mutex);
    _queue = NULL;
    pthread_mutex_unlock(&mutex);
}

static void onLowMemory(ANativeActivity* activity)
{
    LOGI("LowMemory: %p\n", activity);
}

static void onNativeWindowCreated(ANativeActivity* activity,
                                   ANativeWindow* window)
{
    LOGI("NativeWindowCreated: %p -- %p\n", activity, window);
}

static void onNativeWindowDestroyed(ANativeActivity* activity,
                                     ANativeWindow* window)
{
    LOGI("NativeWindowDestroyed: %p -- %p\n", activity, window);
}

static void onPause(ANativeActivity* activity)
{
    LOGI("Pause: %p\n", activity);
}

static void onResume(ANativeActivity* activity)
{
    LOGI("Resume: %p\n", activity);
}

static void* onSaveInstanceState(ANativeActivity* activity, size_t*
                                  outLen)
{
    LOGI("SaveInstanceState: %p\n", activity);
    return NULL;
}

static void onStart(ANativeActivity* activity)
{
    LOGI("Start: %p\n", activity);
}

static void onStop(ANativeActivity* activity)
{
    LOGI("Stop: %p\n", activity);
}

```

```

}

static void onWindowFocusChanged(ANativeActivity* activity, int focused)
{
    LOGI("WindowFocusChanged: %p -- %d\n", activity, focused);
}

static void* process_input(void* param)
{
    while(1)
    {
        pthread_mutex_lock(&mutex);
        if(_queue == NULL)
            pthread_cond_wait(&cond, &mutex);
        AInputEvent* event = NULL;
        while (AInputQueue_getEvent(_queue, &event) >= 0)
        {
            if(AInputQueue_preDispatchEvent(_queue, event))
                break;
            AInputQueue_finishEvent(_queue, event, 0);
        }
        pthread_mutex_unlock(&mutex);
    }
}

void ANativeActivity_onCreate(ANativeActivity* activity,
                             void* savedState,
                             size_t savedStateSize)
{
    LOGI("Creating: %p\n", activity);
    LOGI("Internal data path: %s\n", activity->internalDataPath);
    LOGI("External data path: %s\n", activity->externalDataPath);
    LOGI("SDK version code: %d\n", activity->sdkVersion);
    LOGI("Asset Manager: %p\n", activity->assetManager);

    activity->callbacks->onConfigurationChanged = onConfigurationChanged;
    activity->callbacks->onDestroy = onDestroy;
    activity->callbacks->onInputQueueCreated = onInputQueueCreated;
    activity->callbacks->onInputQueueDestroyed = onInputQueueDestroyed;
    activity->callbacks->onLowMemory = onLowMemory;
    activity->callbacks->onNativeWindowCreated = onNativeWindowCreated;
    activity->callbacks->onNativeWindowDestroyed = onNativeWindowDestroyed;
    activity->callbacks->onPause = onPause;
    activity->callbacks->onResume = onResume;
    activity->callbacks->onSaveInstanceState = onSaveInstanceState;
    activity->callbacks->onStart = onStart;
    activity->callbacks->onStop = onStop;
    activity->callbacks->onWindowFocusChanged = onWindowFocusChanged;

    pthread_mutex_init(&mutex, NULL);
}

```

```

    pthread_cond_init(&cond, NULL);
    pthread_create(&thread, NULL, process_input, NULL);
}

```

程序清单 8-4 开始是三个 `#include` 指令, 表明会包含三个 NDK 头文件的内容用来使用日志、本地 activity 和可移植操作系统接口(POSIX) 线程。

注意:

如果不了解 POSIX 的话, 可以查看维基百科的“POSIX”条目(<http://en.wikipedia.org/wiki/POSIX>)。

程序清单 8-4 然后声明了一个 LOGI 宏, 用于向 Android 设备的日志系统(可以执行 adb logcat 查看日志)中写入信息。这个宏引用了 `int __android_log_print(int prio, const char* tag, const char* fmt, ...)` 函数(函数原型在 `log.h` 头文件中)进行实际的写入动作。每个日志消息都会有一个 priority(如 `ANDROID_LOG_INFO`)、一个 tag(如 `lnademo`)和一个代表信息的格式化字符串。当格式化的字符串包含格式限定符(如 `%d`)时, 还会需要其他的参数。

程序清单 8-4 然后声明了一个 `AInputQueue*` 类型的 `_queue` 变量(`AInputQueue` 是在 `input.h` 头文件中定义的, 该头文件是在 `native_activity.h` 头文件中包含进来的)。当输入队列创建完成后, 该变量会指向这个队列的引用, 当队列销毁后, 该变量被置为 `NULL`。本地 activity 必须处理这个队列终端所有输入事件从而避免“ApplicationNot Responding”错误。

然后创建了三个全局变量: `thread`、`cond` 和 `mutex`。变量 `thread` 表示后面清单中所创建的线程, 变量 `cond` 和 `mutex` 分别用来防止阻塞等待和保证对 `_queue` 变量的同步访问。

后面是一系列以“on”开头的回调函数。每个函数都是静态的, 并且对外部模块隐藏(这里没必要使用 `static`, 但这种形式很好)。

每个以“on”开头的回调函数都会在主线程中调用并会打印一些信息, 这些信息可以在设备的日志系统中查看。而对于 `void onInputQueueCreated(ANativeActivity* activity, AInputQueue* queue)` 和 `void onInputQueueDestroyed(ANativeActivity* activity, AInputQueue* queue)` 两个函数还会做些其他的事情:

- `onInputQueueCreated(ANativeActivity*, AInputQueue*)` 必须将它的 `queue` 参数的地址赋给 `_queue` 变量。由于 `_queue` 也会在主线程之外的线程中访问, 因此需要同步来保证这些线程间不会冲突。同步是通过在 `pthread_mutex_lock(&mutex)` 和 `pthread_mutex_unlock(&mutex)` 之间访问 `_queue` 来实现的。前者会锁定一个 `mutex` (一个程序对象用来防止多个线程同时访问一个共享变量); 后者会将 `mutex` 解锁。由于非主线程在 `_queue` 的值为非 `NULL` 时会一直等待, 因此, `pthread_cond_broadcast(&cond)` 的调用同样会唤醒这个等待线程。
- `onInputQueueDestroyed(ANativeActivity*, AInputQueue*)` 比较简单, 就是在输入队列销毁时将 `_queue` (在一个锁定区域执行) 的值置为 `NULL`。

非主线程会执行 `void* process_input(void* param)` 函数。这个函数会不断执行 `int32_t AInputQueue_getEvent(AInputQueue* queue, AInputEvent** outEvent)` 来返回下一个输入事件。当没有需要处理的事件或者发生错误时, 会返回一个负数整型。当返回一个事件后,

该事件会使用 `outEvent` 来标识。

假设返回一个事件，就会调用 `int32_t AInputQueue_preDispatchEvent(AInputQueue* queue, AInputEvent* event)` 在 `app` 之前把这个事件(如果是键盘相关的事件)发送给当前的输入方法编辑器。如果该事件没有被预分发，这个函数就会返回 0，这也就意味着此时可以处理该事件了。当返回的是非 0 值，就不允许你再处理当前的事件了，这样该事件就可以再次出现在事件队列中(假设在预分发阶段并没有处理)。

此时，可以对该事件做些处理(当它没有被预分发)。尽管如此，你最后会调用 `void AInputQueue_finishEvent(AInputQueue* queue, AInputEvent* event, inthandled)` 来结束给定事件的分发工作。如果传入 `handled` 的值为 0 表示在你的代码中并没有处理这个事件。

最后，程序清单 8-4 声明了 `void ANativeActivity_onCreate(ANativeActivity*, void*, size_t)` 函数，它会打印日志消息，覆写大多数的默认回调函数(需要的话，也可以覆写其他回调函数)，初始化 `mutex` 和 `cond` 变量，并在最后创建和启动了执行 `void* process_input(void*)` 的线程。

程序清单 8-5 展示了这个项目的 `Android.mk` 文件。

程序清单 8-5 LLNADemo 的 Make 文件

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := llndemo
LOCAL_SRC_FILES := llndemo.c
LOCAL_LDLIBS := -llog -landroid
include $(BUILD_SHARED_LIBRARY)
```

这个 `make` 文件有一个 `LOCAL_LDLIBS`，它表明需要链接 `liblog.so` 和 `libandroid.so` 两个标准库。

1. 通过 Android SDK 构建和运行 LLNADemo

想要通过 Android SDK 构建 LLNADemo，首先需要使用 SDK 的 `android` 工具创建 LLNADemo 项目。假如使用的是 Windows 系统，LLNADemo 项目会存储在 `C:\prj\dev` 这样的目录下(即 `C:\prj\dev\ LLNADemo`)，假设目标平台为 Android 4.1，对应的整数 ID 是 1(执行 `android list target` 可获得相应的 ID)，执行以下命令(为了可读性更强，我们使用了两行)来创建 LLNADemo：

```
android create project -t 1 -p C:\prj\dev\LLNADemo -a LLNADemo
-k ca.tutortutor.llndemo
```

这个命令会在 `C:\prj\dev\ LLNADemo` 下创建各种目录和文件。由于不需要 `src` 目录和它的内容，因此想要减少 API 文件大小的话，可以删除该目录。同理，也可以删除 `res` 目录下除了 `values` 目录的所有目录和内容。

在 LLNADemo 项目下创建 `jni` 目录，然后在该目录下创建 `llndemo.c` 和 `Android.mk` 文件，并把程序清单 8-4 和 8-5 的内容分别拷贝到这两个文件中。最后，将 `AndroidManifest.xml` 文件的内容使用程序清单 8-6 的内容替换。

程序清单 8-6 LLNADemo 的 Manifest 文件

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ca.tutortutor.llnademodemo"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9"/>
    <application android:label="@string/app_name" android:hasCode="false">
        <activity android:name="android.app.NativeActivity"
            android:label="@string/app_name"
            android:configChanges="orientation">
            <meta-data android:name="android.app.lib_name"
                android:value="llnademodemo"/>
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

在<activity>标签中加入了 android:configChanges="orientation"属性,这样当设备屏幕方向改变(例如竖屏变横屏)时就会调用 void onConfigurationChanged(ANativeActivity*activity)方法。作为一个练习,可以删除这个属性然后观察一下日志消息的变化。

切换当前目录到 C:\prj\dev\LLNADemo(或者你相应的目录),执行以下与构建库类似的命令:

```
\android-ndk-r8b\ndk-build
```

一切顺利的话,应该看到如下消息:

```

Compile thumb : llnademodemo <= llnademodemo.c
SharedLibrary : libllnademodemo.so
Install : libllnademodemo.so => libs/armeabi/libllnademodemo.so

```

你同样会在 libs 下看到一个 armeabi 目录,在该目录下会有一个 libllnademodemo.so 文件。现在,执行以下命令来构建项目:

```
ant debug
```

如果构建成功,执行以下命令在当前设备上安装 LLNADemodebug.apk 文件。

```
adb install bin\LLNADemo-debug.apk
```

运行该应用程序前,首先启动 AVD1(第 1 章创建的)。然后在一个单独的窗口执行以下命令,这样就可以看到输出的日志:

```
adb logcat
```

启动 LLNADemo, 你会看到一个空白的屏幕。按下 Esc 键, 会返回到应用启动器界面。
图 8-8 展示了一部分和事件相关的日志。

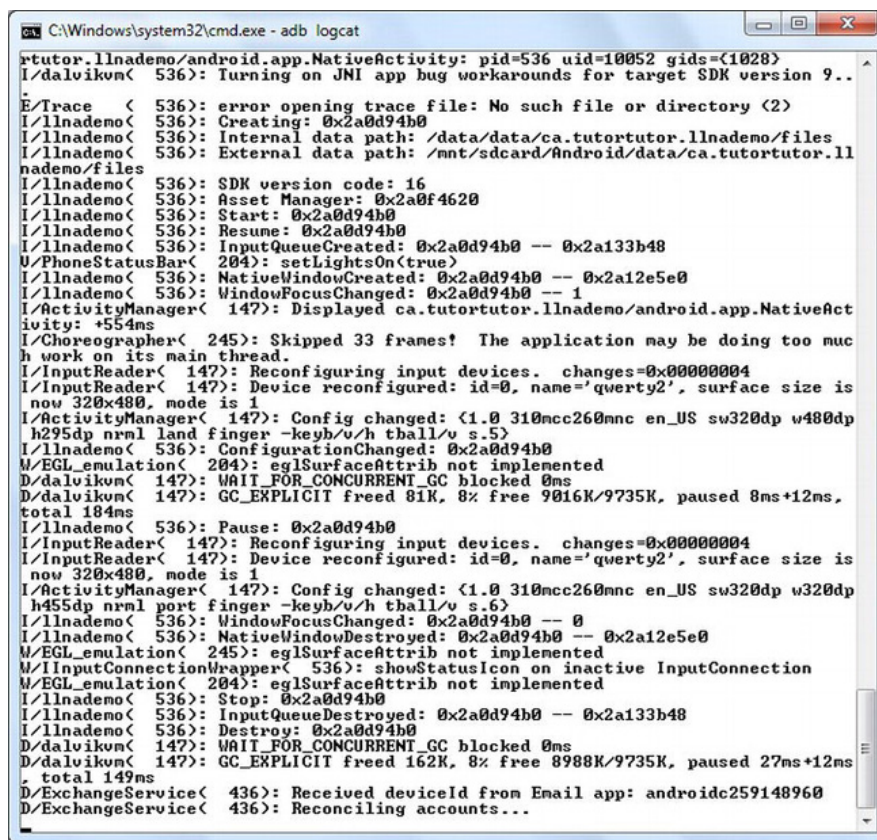


图 8-8 LLNADemo 运行期间各种日志消息

2. 在 Eclipse 中构建和运行 LLNADemo

想要使用 Eclipse 构建 LLNADemo, 首先要像在第 1 章范例 1-10 中所介绍的那样创建一个新的 Android 项目。为了方便起见, 可以使用下面的步骤创建项目:

- (1) 如果还未启动 Eclipse 的话, 先启动 Eclipse。
- (2) 在 File 菜单中选择 New, 在弹出的菜单中选择 Project。
- (3) 在 New Project 对话框中, 展开向导树中的 Android 子节点(如果必要的话), 选择该节点下(如果必要的话)的 Android Application Project 分支, 然后单击 Next 按钮。
- (4) 在 New Android App 对话框的 Application Name 一栏输入 LLNADemo。这个名称同样会显示在 Project Name 中, 它标识了 LLNADemo 项目所在的文件夹/目录。
- (5) 在 Package Name 一栏输入 ca.tutortutor.llnademodemo。
- (6) 通过 Build SDK, 选择合适的目标 Android SDK 版本。这个选择标识了你想要构建应用程序的 Android 平台版本。假设你只安装了 Android 4.1, 那这里只会出现这一个目标并且已经被选中。

(7) 通过 Minimum SDK, 可以选择应用程序所需的最小 Android SDK 版本, 也可以使用默认设置(不要低于 API Level 9)。

(8) 如果想要自定义启动图标的话, 请选中 Create custom launcher icon 复选框。否则, 会使用默认的启动器图标。

(9) 保持 Mark this project as a library 复选框的未选中状态, 因为你并没有打算创建库。

(10) 保持 Create Project in Workspace 复选框的未选中状态, 然后单击 Next 按钮。

(11) 在接下来的 Configure Launcher Icon 界面中, 对自定义启动图标做相应的调整, 然后单击 Next 按钮。

(12) 在随后的 Create Activity 界面中, 保持 Create Activity 复选框的选中状态, 选中 BlankActivity, 单击 Next 按钮。

(13) 在随后的 New Blank Activity 界面中, 在 Activity Name 一栏输入 NDKGreetings, 并在 Layout Name 一栏填写 main, 保持其他设置不变, 然后单击 Finish 按钮。

打开 Package Explorer, 在 LLNADemo 项目下创建一个 jni 文件夹, 在该文件夹下创建 LLNADemo.c 文件, 使用程序清单 8-4 的内容替换原来空白的内容, 再在 jni 下创建一个新的 Android.mk 文件, 使用程序清单 8-5 的内容替换原来空白的内容。

接下来, 我们创建一个构建器来创建库文件。执行以下步骤:

(1) 右击 LLNADemo 节点, 在弹出的菜单中选择 Properties。

(2) 在弹出的 Properties for LLNADemo 对话框中选择 Builders。

(3) 在 Builders 面板中, 单击 New 按钮。

(4) 在弹出的 Choose configuration type dialog 的对话框中, 选择 Program 并单击 OK 按钮。

(5) 在弹出的 Edit Configuration 对话框中, 随意填写一个构建器的名称(或者使用默认名称), 在 Location 一栏输入 C:\android-ndk-r8b\ndk-build.cmd(或者你自己定义的路径), 在 Working Directory 一栏输入 \${workspace_loc:/NDKGreetings}, 单击 OK 按钮, 关闭该对话框。

(6) 单击 OK 按钮, 关闭 Properties for LLNADemo 对话框。

最后, 使用 Eclipse 内置的 manifest 编辑器, 同本范例前面一样, 对 AndroidManifest.xml 做相应的修改。

在要在 Eclipse 中运行 NDKGreetings, 在菜单栏中选择 Run, 然后在下拉菜单中选择 Run。如果出现了 Run As 对话框, 选择 Android Application 并单击 OK 按钮。Eclipse 会启动 AVD1 设备模拟器, 安装 LLNADemo.apk, 并运行该应用程序, 输出结果如图 8-6 所示。

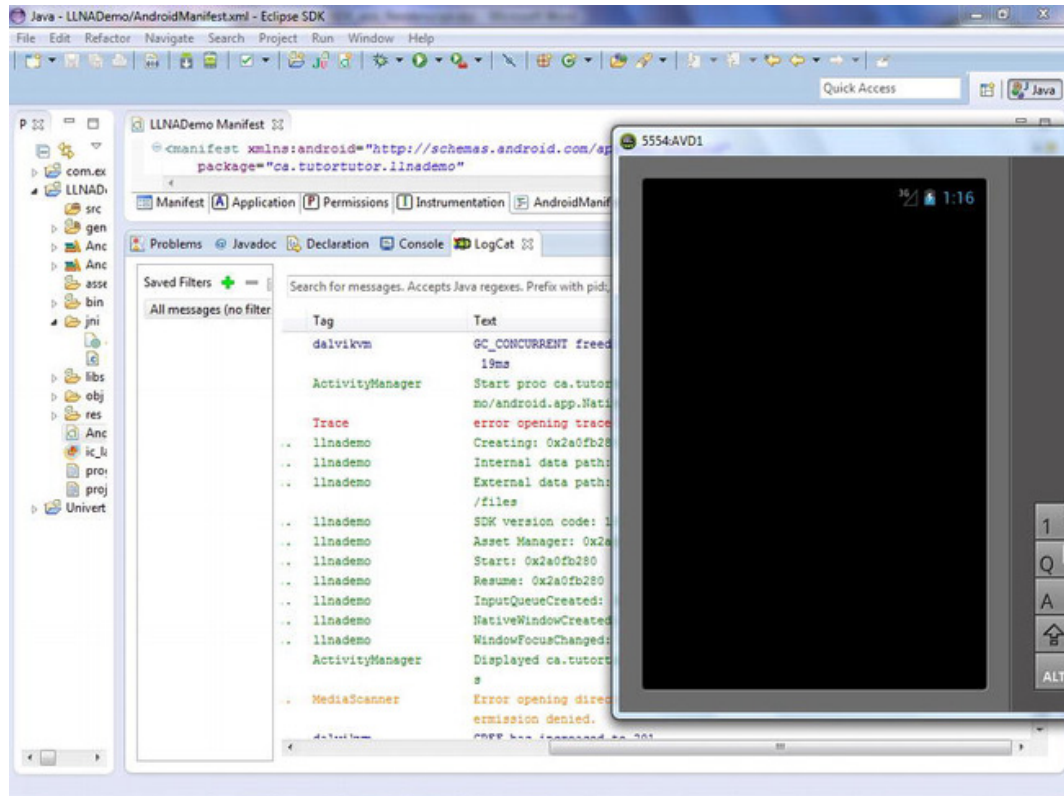


图 8-9 LLNADemo 运行期间各种日志消息

注意:

当目标 SDK 设置为 API Level 13 或者更高(Eclipse 默认目标 SDK 为 15)并且如果没有将<activity>的 configChanges 属性值设置为 screenSize 和 orientation("orientation | screenSize"), 这时改变屏幕的方向将不会看到 “ConfigurationChanged” 消息。

8.4 开发 High-Level 的本地 Activity

8.4.1 问题

需要学习如何开发基于 android_native_app_glue.h 头文件的 high-level 本地 activity。

8.4.2 解决方案

开发 high-level 的本地 activity 与开发 low-level 的本地 activity 非常相似。然而，还是需要一个新的源文件和 Android.mk 文件。

8.4.3 实现机制

使用 HLNADemo 项目来演示 high-level 本地 activity。

程序清单 8-7 展示了这个项目中唯一的 hlnademo.c 源文件的内容。

程序清单 8-7 从高层的视角体验一个本地 Activity

```
#include <android/log.h>
#include <android_native_app_glue.h>

#define LOGI(...) ((void)__android_log_print(ANDROID_LOG_INFO, \
                                              "hlnademo", \
                                              __VA_ARGS__))

static void handle_cmd(struct android_app* app, int32_t cmd)
{
    switch(cmd)
    {
        case APP_CMD_SAVE_STATE:
            LOGI("Save state");
            break;

        case APP_CMD_INIT_WINDOW:
            LOGI("Init window");
            break;

        case APP_CMD_TERM_WINDOW:
            LOGI("Terminate window");
            break;

        case APP_CMD_PAUSE:
            LOGI("Pausing");
            break;

        case APP_CMD_RESUME:
            LOGI("Resuming");
            break;

        case APP_CMD_STOP:
            LOGI("Stopping");
            break;

        case APP_CMD_DESTROY:
            LOGI("Destroying");
            break;

        case APP_CMD_LOST_FOCUS:
            LOGI("Lost focus");
            break;
    }
}
```

```

        case APP_CMD_GAINED_FOCUS:
            LOGI("Gained focus");
        }
    }

static int32_t handle_input(struct android_app* app, AInputEvent* event)
{
    if (AInputEvent_getType(event) == AINPUT_EVENT_TYPE_MOTION)
    {
        size_t pointerCount = AMotionEvent_getPointerCount(event);
        size_t i;
        for(i = 0; i < pointerCount; ++i)
        {
            LOGI("Received motion event from %zu: (%.2f, %.2f)", i,
                AMotionEvent_getX(event, i), AMotionEvent_getY(event, i));
        }
        return 1;
    }
    else if(AInputEvent_getType(event) == AINPUT_EVENT_TYPE_KEY)
    {
        LOGI("Received key event: %d", AKeyEvent_getKeyCode(event));
        if(AKeyEvent_getKeyCode(event) == AKEYCODE_BACK)
            ANativeActivity_finish(app->activity);
        return 1;
    }
    return 0;
}

void android_main(struct android_app* state)
{
    app_dummy(); // prevent glue from being stripped

    state->onAppCmd = &handle_cmd;
    state->onInputEvent = &handle_input;

    while(1)
    {
        int ident;
        int fdesc;
        int events;
        struct android_poll_source* source;

        while ((ident = ALooper_pollAll(0, &fdesc, &events, (void**)&source)) >=
0)
        {
            if (source)
                source->process(state, source);

            if (state->destroyRequested)
                return;
        }
    }
}

```

```

    }
}

```

程序清单 8-7 刚开始和程序清单 8-4 的风格几乎是一样的。只是之前的 `native_activity.h` 头文件已经被 `android_native_app_glue.h` 所替代,它包含了 `native_activity.h`(以及 `pthread.h`)。它同样提供了一个类似的 LOGI 宏。

`void handle_cmd(struct android_app* app, int32_t cmd)`函数(在非主线程中调用)用来响应一个 activity 命令。`app` 参数指向了一个 `android_app` 结构体(在 `android_native_app_glue.h` 中定义的),该结构体可以访问应用程序相关的数据,`cmd` 参数则表示一个命令。

注意:

命令都是与前面所述的 low-level 本地 activity 函数(如 `void onDestroy(ANativeActivity* activity)`)所对应的整型值。这些命令(如 `APP_CMD_DESTROY`)对应的整型常量会在 `android_native_app_glue.h` 头文件中定义。

`int32_t handle_input(struct android_app* app, AInputEvent* event)`函数(在非主线程中调用)用来响应一个输入事件。`event` 参数指向了一个 `AInputEvent` 结构体(在 `input.h` 中定义),该结构体可以访问不同类型与事件相关的信息。

`Input.h` 头文件中定义了一些很有用的输入函数,第一个函数是 `AInputEvent_getType(const AInputEvent* event)`,它会返回事件的类型。该类型的值为代表按键事件的 `AINPUT_EVENT_TYPE_KEY` 或者代码动作事件的 `AINPUT_EVENT_TYPE_MOTION`。

对于动作事件,调用 `size_t AMotionEvent_getPointerCount(const AInputEvent* motion_event)` 函数会返回这个事件信息中的触点(有效触点)的个数(该值大于等于 1)。在每次获得并记录触点坐标时,这个触点计数会重复。

注意:

有效触点和 `AMotionEvent_getPointerCount(const AInputEvent*)` 都涉及多点触摸的。想要了解更多这种 Android 特性,可以查看“Making Sense of Multitouch”(http://androiddevelopers.blogspot.ca/2010/06/making-sense-of-multitouch.html)。

对于按键事件,`int32_t AKeyEvent_getKeyCode(const AInputEvent*key_event)`函数会返回所按下的物理按键的键值。物理按键的键值是在 `keycodes.h` 中定义的。例如 `AKEYCODE_BACK` 对应设备的返回键。

键值会被记录下来,当用户想要终止 activity(或者终止只有一个 activity 的应用程序)时,会将这个记录的键值与 `AKEYCODE_BACK` 进行比较。如果二者相等,会调用 `void ANativeActivity_finish(ANativeActivity* activity)`,该函数传入 `app->activity` 即想要终止的 activity。

当处理完鼠标和按键事件后,`handle_input(struct android_app*,AInputEvent*)`函数会返回 1,这表示该函数已经处理了事件。如果事件未被处理的话(这时应该在后台使用默认的处理程序处理),这个函数会返回 0。

注意:

可以为 `handle_input(struct android_app*, AInputEvent*)` 的 `if (AKeyEvent_getKeyCode(event) == AKEYCODE_BACK)` 添加注释, 然后是 `ANativeActivity_finish(app->activity)`; 然后是 `return 1;` 语句, 或者返回 0 (即按下 back 键后的默认处理, 销毁 Activity)。

`void android_main(struct android_app* state)` 函数是程序的入口。它首先会调用一个叫做 `app_dummy()` 的本地固有函数, 该函数不会做任何事情。但是, 还必须要调用它, 这是确保 Android 构建系统在库中包含了 `android_native_app_glue.o` 模块。

注意:

想要了解这种奇怪现象的更多信息可以查看 http://blog.beuc.net/posts/Make_sure_glue_isn_39_t_stripped。

`android_app` 结构体中有一个 `void(*onAppCmd)(struct android_app* app, int32_t cmd)` 类型的 `onAppCmd` 的变量和一个 `int32_t (*onInputEvent)(struct android_app* app, AInputEvent* event)` 类型的 `onInputEvent` 变量。上述函数的地址会赋给这些变量。

然后进入到两个内嵌循环函数中。内部的循环会不断地调用 `intALooper_pollAll(int timeoutMillis, int* outFd, int* outEvents, void** outData)` 函数 (在 `looper.h` 中定义) 来返回下一个事件。当可以处理事件时, 这个函数的返回值会大于等于 0。

事件会记录在一个 `android_poll_source` 结构体中, 该结构体的地址存储在 `outData` 中。假设 `outData` 包含了一个非 NULL 的地址, 那么就会调用 `android_poll_source` 的 `void(*process)(struct android_app* app, struct android_poll_source* source)` 函数来处理该事件。在幕后, 或者调用 `handle_cmd(struct android_app*, int32_t)` 或者 `handle_input(struct android_app*, AInputEvent*)`; 取决于哪个函数更适合处理该事件。

最后, `android_app` 结构体的成员变量 `destroyRequested` 会设置为非 0 的值, 作为调用 `ANativeActivity_finish(ANativeActivity*)` 的结果。在每次循环迭代时都会检查这个变量, 这是为了确保在应用程序结束时, 可以从内嵌的循环和 `android_main(struct android_app*)` 中快速退出。如果无法及时从 `android_main(struct android_app*)` 函数中退出可能会导致 “Application Not Responding” 错误。

程序清单 8-8 展示了这个项目的 `Android.mk` 文件。

程序清单 8-8 HLNADemo 的 Make 文件

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := hlnademo
LOCAL_SRC_FILES := hlnademo.c
LOCAL_LDLIBS := -landroid
LOCAL_STATIC_LIBRARIES := android_native_app_glue
include $(BUILD_SHARED_LIBRARY)
$(call import-module, android/native_app_glue)
```

这个 make 文件和程序清单 8-5 的 make 文件非常相似。但也有一些不同的地方:

- LOCAL_LDLIBS 中不再包含 -llog, 这是因为当 android_native_app_glue 库构建时, 它链接的日志库也会被构建。
- LOCAL_STATIC_LIBRARIES 表明 android_native_app_glue 会作为一个库链接到 hlnademo 模块。
- \$(call import-module,android/native_app_glue)包含了 Android.mk 文件并关联了 android_native_app_glue 模块, 这样这个库才能被构建。

1. 使用 Android SDK 构建和运行 HLNADemo

可以像构建 LLNADemo 那样构建 HLNADemo(在 manifest 中把每个 llnademo 的引用改为 hlnademo 并且使用修改了的程序清单 8-8 所示的 Android.mk 文件)。然后启动 HLNADemo, 你应该会看到一个空白屏幕。图 8-10 展示了一部分和程序清单 8-7 相关的日志消息。

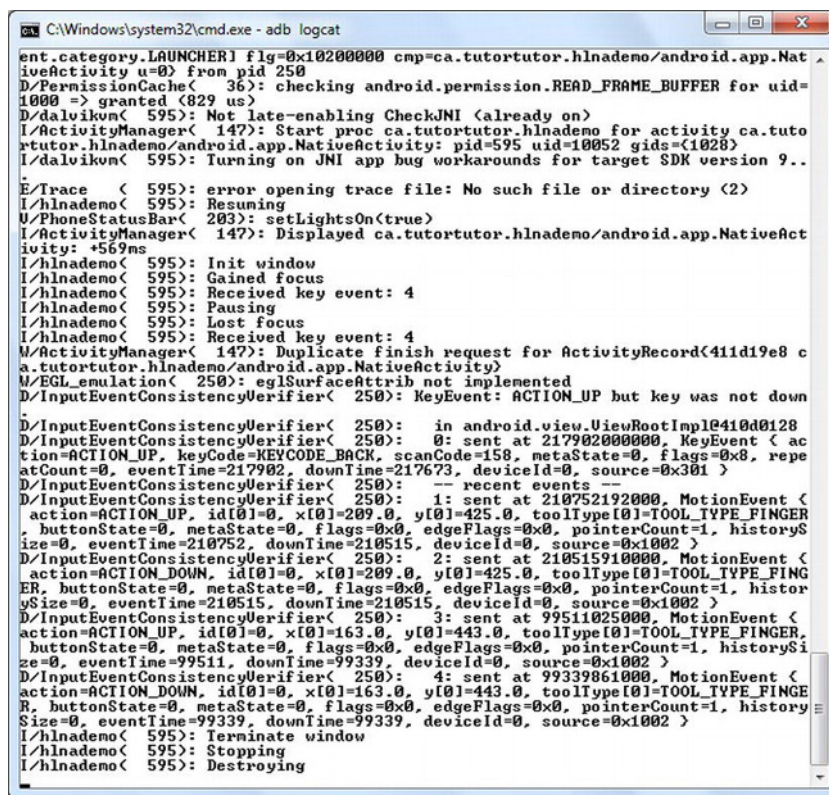


图 8-10 HLNADemo 运行期间各种日志消息

2. 在 Eclipse 中构建和运行 HLNADemo

可以像构建 LLNADemo 那样构建 HLNADemo(在 manifest 中把每个 llnademo 的引用改为 hlnademo 并且使用修改了的程序清单 8-8 所示的 Android.mk 文件)。然后启动 HLNADemo, 你应该会看到一个空白屏幕。图 8-11 展示了一部分和程序清单 8-7 相关的日志消息。

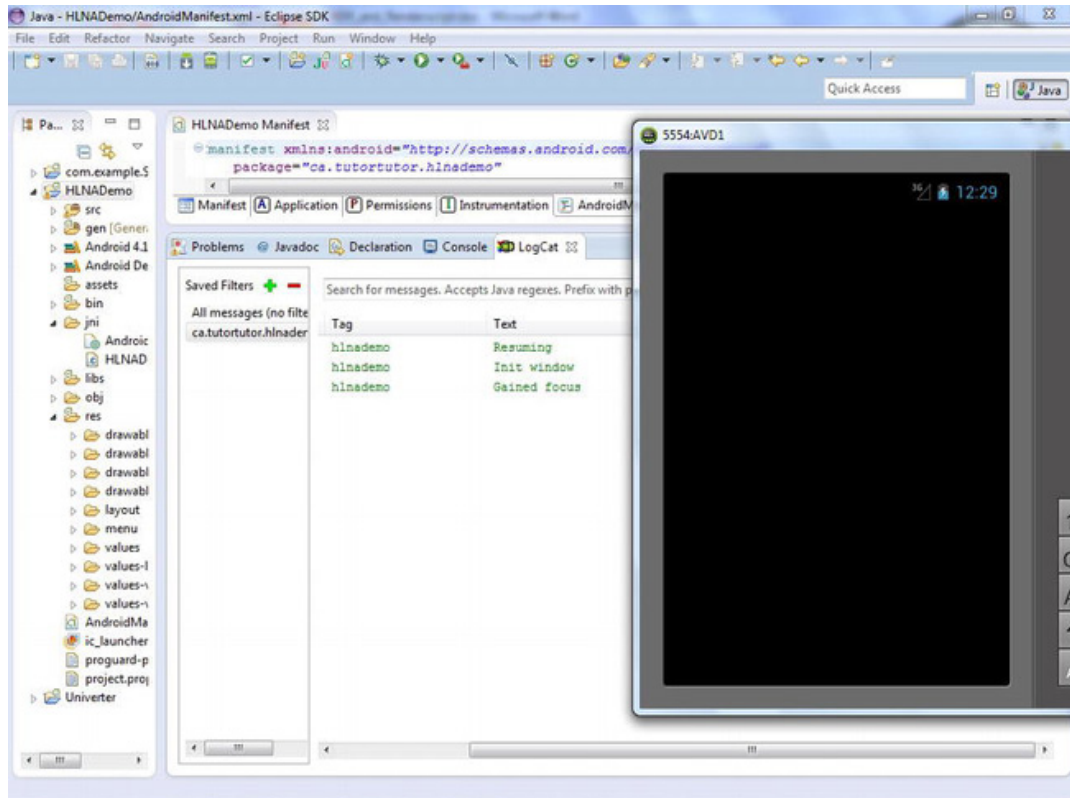


图 8-11 HLNADemo 运行期间各种日志消息

8.5 Renderscript

可以使用 Android NDK 快速地执行渲染操作和数据处理。但是，这种方式会有三个主要问题：

- 可移植性差：应用程序只能运行于本地代码所限定的设备上。例如运行在基于 ARM 设备上的本地库将不能用在基于 x86 的设备上。
- 性能差：通常情况下，你的代码会运行在多个内核上，它们可能是多个 CPU、GPU 或者 DSP 内核。但是，识别内核、为它们分配工作，以及多核同步问题都不是那么容易处理的。
- 可用性差：开发本地代码要比开发 Java 代码难。例如需要经常创建 JNI 固有代码，这是一个乏味的过程而且可能引起很多的 bug。

Google 的 Android 开发团队创建了 Renderscript 来解决这些问题，首先是可移植性问题，然后是性能问题，最后是可用性问题。Renderscript 是由基于 C99 的语言(C 语言的现代版本)、两个编译器和一个运行时组成的，这些元素共同帮助实现高性能和好看的可视化图形(本地代码实现，但可移植性强)。这样不必使用 JNI 就能获得本地应用程序的速度和 SDK 应用程序的可移植性。

注意:

虽然在 Android 2.0 中就已经引入了 Renderscript，但直到 Android 3.0 中用它实现活动壁纸及其他更多功能时，Renderscript 才开始公开。

Renderscript 由一个图形引擎和一个计算引擎组成。图形引擎用来实现 2D/3D 图形的快速渲染，而计算引擎则用来快速处理数据。通过在多核 CPU、GPU 和 DSP 上运行多个线程来提高性能。

提示:

计算引擎并不只局限于处理图形数据。例如：它还可以用来模拟天气数据。

8.5.1 浏览 Renderscript 架构

Renderscript 采用的架构为底层 Renderscript 运行时，它由上层的 Android 框架层控制。图 8-12 展示了这种架构。

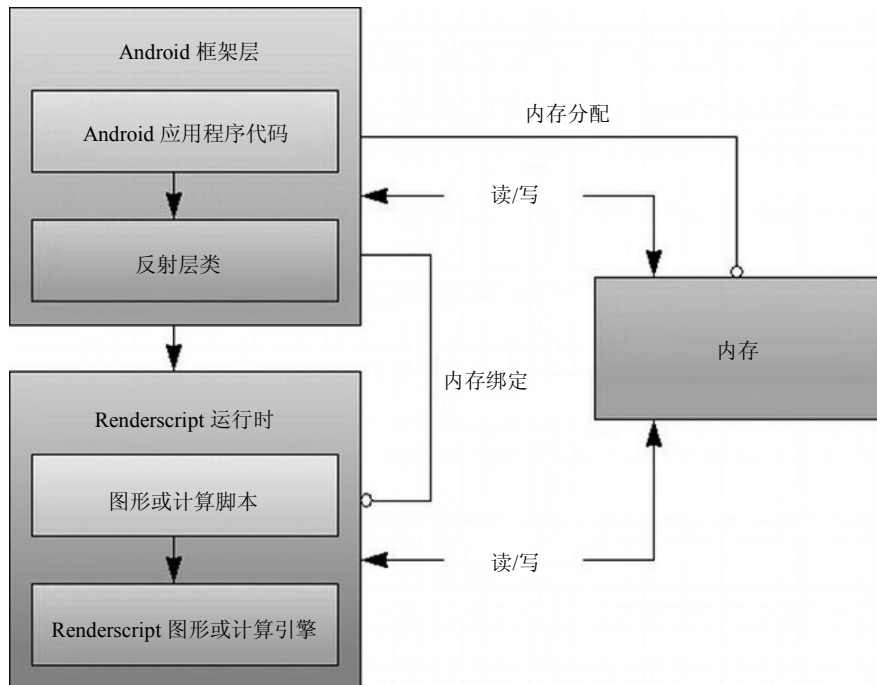


图 8-12 基于 Android 框架层和 Renderscript 运行时的 Renderscript 架构

Android 框架层由很多 Android 应用程序组成，这些应用程序运行在 Dalvik 虚拟机上并且通过反射层类的实例与运行在 Renderscript 运行时之上的图形或计算脚本进行通信。这些反射层类对脚本进行了封装，从而使这种通信成为可能。Android 构建工具在构建过程中会自动生成反射层的类。这些类则不必编写通常使用 NDK 时所需的固有代码。

内存管理是在 VM 层进行控制的。应用程序负责分配内存并把它绑定到 Renderscript 运行时，这样脚本就可以访问这块内存(脚本可以定义简单(非数组)的字段供自己使用，但只能自己使用)。

应用程序会异步调用 **Renderscript** 运行时(通过 **reflected layer** 类)来使用分配的内存并开始执行它们的脚本。随后它们就可以得到脚本的执行结果而不必担心脚本是否还在运行。

在构建 APK 时, **LLVM (Low-Level Virtual Machine)** 前端编译器(参见附录 B 的 **llvm-rs-cc** 工具)会将脚本编译为一个与设备无关的二进制码并保存到 APK 中。当应用程序启动时, 设备上的 **LLVM** 后端编译器会将二进制码编译为设备特定的代码并缓存到设备上, 这样每次运行应用程序时就不必重新编译了。这就是它的可移植性所在。

注意:

Android 4.1 中, 图形引擎已经不建议使用了。应用程序开发者告诉 **Android** 开发团队说由于他们更熟悉 **OpenGL** 故而更倾向于使用 **OpenGL**。虽然还可以使用图形引擎, 但在将来的版本中它可能会被去掉。因此, 本章下面部分只会着重于计算引擎。

1. 浏览基于计算引擎的应用程序的架构

基于计算引擎的应用程序由 **Java** 代码和一个定义计算脚本的 **.rs** 文件组成。**Java** 代码通过 **android.renderscript** 包中定义的 **API** 与脚本进行交互。该包中比较关键的类是 **RenderScript** 和 **Allocation**。

- **RenderScript** 定义了一个上下文环境用来与 **Renderscript** **API**(以及计算脚本的反射层类)进行进一步的交互。调用这个类的 **static RenderScript.create(Context ctx)** 工厂方法会返回一个 **RenderScript** 实例。
- **Allocation** 定义了从计算脚本中移入和移出数据时所使用的方法。这个类的实例称为 **allocations**, 它表示了一个 **android.renderscript.Type** 实例和用户数据和对象所需的内存。

通过初始化一个反射层类, **Java** 代码同样可以与计算脚本进行交互。类名的前缀为 **ScriptC_**, 接着是包含计算脚本的 **.rs** 文件名。例如, 有一个名为 **gray.rs** 的文件, 那么这个类名就为 **ScriptC_gray**。

基于 **C99** 的 **.rs** 文件首先是两个 **#pragmas** 来表示 **Renderscript** 的版本号以及应用程序的 **Java** 包名。以下为其他的一些选项:

- **rs_allocation** 指令表示应用程序创建的并且绑定到 **Renderscript** 代码的输入和输出 **allocations**。
- **rs_script** 指令为应用程序的 **ScriptC_script** 实例提供了一个链接, 以便计算的结果可以返回给这个实例。
- 可选的简单的变量声明, 变量的值由应用程序提供。
- **root()** 函数, 由每个 **cpu** 核进行调用来执行部分的整体计算。
- 返回值为 **void** 的无参初始化函数, 多个 **CPU** 核时会被 **Java** 代码间接调用来执行 **root()** 函数。

在运行时, 基于 **Java** 的 **Activity** 会创建一个 **Renderscript** 上下文、创建输入和输出 **allocations**、实例化前缀为 **ScriptC_** 的类, 然后使用这个对象将 **allocations** 和 **ScriptC_** 实例进行绑定, 调用计算脚本, 它的结果会在脚本的初始化函数中被调用。

初始化函数会执行其他的初始化动作(必要时)并使用 **rs_script** 和 **rs_allocation** 输入/输

出 allocations 来执行 rsForEach()函数。rsForEach()会让设备的可用 CPU 核执行 root()函数。然后执行结果会通过输出 allocation 返回给应用程序。图 8-13 演示了这一场景。

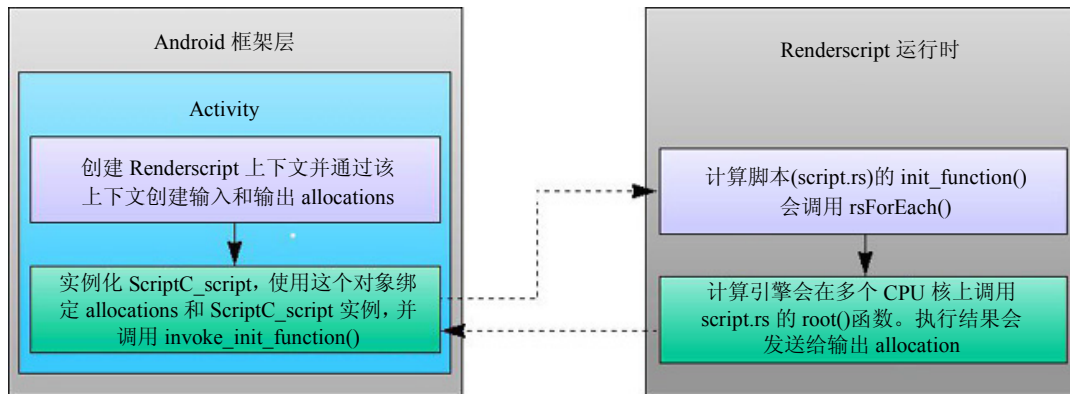


图 8-13 基于计算引擎的应用程序架构可以拆分为 4 个主要部分

8.5.2 使用 Renderscript 对图像进行灰度化处理

也许熟悉 Renderscript 计算方面知识的最简单方式就是创建一个小的应用程序，该应用程序执行了一个简单的图像处理操作，例如对图像进行灰度处理。程序清单 8-9 展示了 GrayScale 应用程序的源代码，它可以浏览 Sun 的图片，将图片进行灰度化处理，并且浏览结果。

程序清单 8-9 浏览原始和经过灰度化处理的 Sun 图片

```
package ca.tutortutor.grayscale;

import android.app.Activity;

import android.os.Bundle;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

import android.renderscript.Allocation;
import android.renderscript.RenderScript;

import android.view.View;

import android.widget.ImageView;

public class GrayScale extends Activity
{
    boolean original = true;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {

```

```

super.onCreate(savedInstanceState);
final ImageView iv = new ImageView(this);
iv.setScaleType(ImageView.ScaleType.CENTER_CROP);
iv.setImageResource(R.drawable.sol);
setContentView(iv);
iv.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        if(original)
            drawGS(iv, R.drawable.sol);
        else
            iv.setImageResource(R.drawable.sol);
        original = !original;
    }
});
}

private void drawGS(ImageView iv, int imID)
{
    Bitmap bmIn = BitmapFactory.decodeResource(getResources(), imID);
    Bitmap bmOut = Bitmap.createBitmap(bmIn.getWidth(), bmIn.getHeight(),
        bmIn.getConfig());

    RenderScript rs = RenderScript.create(this);
    Allocation allocIn;
    allocIn = Allocation.createFromBitmap(rs, bmIn,
        Allocation.MipmapControl.MIPM
        AP_NONE,
        Allocation.USAGE_SCRIPT);
    Allocation allocOut = Allocation.createTyped(rs, allocIn.getType());
    ScriptC_grayscale script = new ScriptC_grayscale(rs, getResources(),
        R.raw.grayscale);

    script.set_in(allocIn);
    script.set_out(allocOut);
    script.set_script(script);
    script.invoke_filter();
    allocOut.copyTo(bmOut);
    iv.setImageBitmap(bmOut);
}
}

```

程序清单 8-9 声明了一个名为 GrayScale 的 activity 类。这个类覆写了 onCreate(Bundle) 方法，声明了一个 void drawGS(ImageView iv, int imID)方法来灰度化处理资源 ID 为 imID 的图片，并把生成的结果显示在 iv 所代表的 android.widget.ImageView 实例上。

onCreate(Bundle)中创建了 activity 的 UI，其中包含了一个 ImageView 实例，该 ImageView 的内容为 R.drawable.sol 所代表的 drawable 资源。图片会进行统一缩放(保持图片的比例不变)，这样就可以适应每种分辨率屏幕的大小(padding 会小一点)。

Imageview 注册了一个 click listener。第一次单击它会调用 drawGS(ImageView, int)将

图片做灰度化处理并且用灰度化过的图片更新 `imageView` 的图片。第二次单击则会显示原始的图片。接下来再单击则会重复这种交替的行为模式。

`drawGS(ImageView, int)` 首先会调用 `android.graphics.BitmapFactory` 类的静态方法 `Bitmap decodeResource(Resources res, int id)` 方法。传入一个 `android.content.res.Resources` 实例(通过 `android.content.Context` 的 `Resources getResources()` 方法获得)和想要修改的 `drawable` 的资源 ID(`R.drawable.sol`)，这个方法会返回一个包含了该资源内容的 `android.graphics.Bitmap` 实例。

`drawGS(ImageView, int)` 然后调用了 `Bitmap` 的静态 `BitmapcreateBitmap(int width, int height, Bitmap.Config config)` 方法来创建并返回一个和上面那个 `bitmap`(它包含了 `Sun` 图片的内容)一样分辨率和配置的空 `bitmap`。

创建一个 `RenderScript` 上下文对象。之后这个对象会作为 `Allocation` 的静态方法 `createFromBitmap(RenderScript rs, Bitmap b, Allocation.MipmapControl mips, int usage)` 的第一个参数传入，该静态方法会创建一块内存来存储 `drawable` 的 `bitmap`。另外还需要传入其他三个参数：

- `bmIn` 表示 `allocation` 的源 `bitmap`。
- `Allocation.MipmapControl.MIPMAP_NONE` 表示不会生成 `mipmap`(预先计算的、优化过的图像集合，伴随着主纹理，旨在提高渲染速度和减少锯齿效果)，通过输入 `bitmap` 所生成的类型将不会包含其他级别的详细信息。
- `Allocation.USAGE_SCRIPT` 指定了计算脚本绑定和访问的 `allocation`。

创建的 `Allocation` 对象会作为输入 `allocation`，计算脚本会得到它的输入数据并进行处理。创建的第二个 `Allocation` 用来保存计算的输出，即调用 `Allocation` 的 static `Allocation createTyped(RenderScript rs, Type type)` 方法，参数如下：

- `rs` 表示 `RenderScript` 上下文。
- `allocIn.getType()` 返回输入 `allocation` 的类型。这个类型描述了输入 `allocation` 的 `bitmap` 的布局。

然后，反射的 `ScriptC_grayscale` 类会通过下面的参数进行初始化，它实现了应用程序与计算脚本之间的通信：

- `rs` 表示 `RenderScript` 上下文。
- `getResources()` 返回要访问资源的 `Resources` 实例。
- `R.raw.grayscale` 表示保存在 APK 的 `res/raw` 目录下的一个 `grayscale.bc` 二进制资源文件。

你会发现，计算脚本包括了 `in`、`out` 和脚本字段。应用程序(用于初始化)可以通过 `ScriptC_grayscale` 的 `set_in()`、`set_out()` 和 `set_script()` 访问这些字段。这些方法可以用来实现输入/输出 `allocations` 和 `ScriptC_grayscale` 实例与计算脚本之间的通信。

随同 `set_in()`、`set_out()` 和 `set_script()` 方法，LLVM 前端编译器会创建一个 `invoke_filter()` 方法，在这里会调用该方法执行计算脚本(这个名字的过滤器部分在计算脚本中会匹配无参的初始化函数)。

`RenderScript` 一个比较好的地方就是应用程序可以立刻请求脚本结果，而不必等到脚本执行完成。这种情况下，应用程序调用 `Allocation` 的 `void copyTo(Bitmap b)` 方法将输出 `allocation` 中的结果复制到传入的 `Bitmap` 实例中，这里只会处理空的 `bitmap`。

最后, 通过 `ImageView` 的 `void setImageBitmap(Bitmap bm)` 方法, 将前面空的 `bitmap` 指定给 `imageView`, 然后就可以看到灰度处理后的图片。

现在已经浏览了该应用程序的 Java 方面的东西, 而程序清单 8-10 则介绍了 C99 计算脚本方面的东西。

程序清单 8-10 对图片进行灰度处理

```
#pragma version(1)
#pragma rs java_package_name(ca.tutortutor.grayscale)

rs_allocation in;
rs_allocation out;
rs_script script;

const static float3 gsVector = {0.3f, 0.6f, 0.1f};

void root(const uchar4* v_in, uchar4* v_out)
{
    float4 f4 = rsUnpackColor8888(*v_in);
    *v_out = rsPackColorTo8888((float3) dot(f4.rgb, gsVector));
}

void filter()
{
    rsDebug("RS_VERSION = ", RS_VERSION);
    #if !defined(RS_VERSION) || (RS_VERSION < 14)
        rsForEach(script, in, out, 0);
    #else
        rsForEach(script, in, out);
    #endif
}
```

程序清单 8-10 首先显示了两个 `#pragmas`, 分别标识了 `Renderscript` 的版本号和该计算脚本所涉及的 Java 包名 `ca.tutortutor.grayscale`。

跟着是两个 `rs_allocation` 指令和一个 `rs_script` 指令。这些指令定义了指向 `input/output allocation` 和 `ScriptC_grayscale` 实例的变量。

接下来, 定义了 `gsVector`, 它是一个包含三个浮点型值(用 `float3` 类型表示)的向量。`Gsvector` 的初始值为 0.3f (red)、0.6f (green)和 0.1f(blue), 它表示整体灰度值中像素颜色的百分比。

接下来, 在每个可用的 CPU 上调用了每个像素的 `void root(const uchar4* v_in, uchar4* v_out)` 函数。每个像素的处理都独立于其他像素。

`v_in` 参数是一个指向 `uchar4` 结构体的指针, 该结构体有四个 8 位的值(red、green、blue 和 alpha 值)来表示输入的像素。而 `v_out` 也是一个指针, 它指向类似的结构体, 而这个结构体是用来存储像素的处理结果的。

`root(const uchar4*, uchar4*)` 首先将 `*v_in` 解析为一个 `float4` 值(`float4` 代表一个拥有 4 个浮点型值的向量)。这个工作是通过调用 `Renderscript` 的 `float4 rsUnpackColor8888(uchar4`

color)函数完成的。

调用 Renderscript 的 float dot(float lhs, float rhs)(点积)函数将 f4.rgb 与 gsVector 的三个颜色值相乘,这样就可以访问这个 vector 的 RGB 值。

注意:

dot()遵循一种模式,它可以接受 1、2、3 或者 4 个值作为参数。这种模式可以让你指定标量值(如 float result = dot(1.0f, 2.0f);)或者不超过 4 个值的向量(如 float3 result = dot(f4.rgb, gsVector);)。

最后的工作就是通过调用 Renderscript 的 uchar4 rsPackColorTo8888(float3 color)函数将 dot()的返回值解析为一个 uchar4 值,然后把这个值赋给*v_out。强转为(float3)类型有点多余但是必要的。

最后,应用程序执行 script.invoke_filter()将会导致 filter()函数被调用,注意这种会在函数名称前面附加 invoke 的形式。

filter()首先会执行 rsDebug("RS_VERSION = ", RS_VERSION); 会将 RS_VERSION 常量的值输出到日志中。可以调用一个 Renderscript 的重载函数 rsDebug()来输出调试信息。

RS_VERSION 是一个特殊的常量,它被设置为 SDK 的版本号。基于这个常量是否存在以及它的值,filter()使用了 #if 和 #else 指令来帮助编译器选择调用不同版本的 rsForEach()。

假设 RS_VERSION 存在并且值小于 14,就会调用 void rsForEach(rs_script script, rs_allocation input, rs_allocation output, const void* usrData)这个最简单的 rsForEach()变种函数。

注意:

usrData 可以让你传递一个附加 script-specific 数据的指针给 root()函数。在范例 8-4 中,你将了解如何在 root()中获得这个指针。

如果 RS_VERSION 的值大于等于 14,就会调用 void rsForEach(rs_script script, rs_allocation input, rs_allocation output)这个最简单的 rsForEach()变种函数。

注意:

在网上可能会遇到不检查 RS_VERSION 的 rsForEach()调用示例。但是,不通过 #if 和 #else 检查这个常数意味着在某个 Android SDK 或者 Eclipse 下脚本可能编译 ok,但在其他的开发平台会编译失败,它的输出信息类似于以下消息:

```
note: candidate function not viable: requires 4 arguments, but
3 were provided
```

```
note: candidate function not viable: requires 5 arguments, but
3 were provided
```

无论调用的是哪个 rsForEach()函数,在 Java 中它的第一个参数都是一个脚本对象的引用;第二个参数 in 对应 v_in;第三个参数 out 对应 v_out。

1. 通过 Android SDK 构建和运行 GrayScale

想要通过 Android SDK 构建 GrayScale, 首先使用 SDK 的 android 工具创建一个 GrayScale 项目。假如使用的是 Windows 系统, GrayScale 项目会存储在 C:\prj\dev 这样的目录下(即 C:\prj\dev\ GrayScale), 假设目标平台为 Android 4.1, 对应的整数 ID 是 1(执行 android list target 可获得相应的 ID), 执行以下命令(为了使可读性更强, 我们使用了两行)来创建 GrayScale:

```
android create project -t 1 -p C:\prj\dev\GrayScale -a GrayScale
                        -k ca.tutortutor.grayscale
```

将 src\ca\tutortutor\grayscale\GrayScale.java 文件的内容使用程序清单 8-9 的内容替换。同样, 使用程序清单 8-10 的内容创建一个 grayscale.rs 文件, 并将该文件存储在 src 目录下(Renderscript 源文件应该使用 .rs 扩展名并存储在项目的 src 目录)。最后, 创建一个 drawable-nodpi 目录, 将名为 sol.jpg(假设该图片包含了 Sun 的图像)的文件拷贝到该目录(Android 不会缩放存储在 drawable-nodpi 中的图片, 而是由应用程序负责缩放)

执行以下命令构建项目:

```
ant debug
```

你将看到如下的警告消息(为了使可读性更强, 采用多行显示)和构建失败的信息:

```
WARNING: RenderScript include directory
        'C:\prj\dev\GrayScale\${android.renderscript.include.path}' does
        not exist!
[llvm-rs-cc.exe] <built-in>:2:10: fatal: 'rs_core.rsh' file not found
```

Google 的 Android 问题数据库中问题 34569(<http://code.google.com/p/android/issues/detail?id=34569>)提供了一种变通方案: 在位于 Android SDK 主目录的 tools\ant 子目录的 build.xml 文件中, 添加如下属性(为了使可读性更强, 采用多行显示):

```
<property name="android.renderscript.include.path"
          location="${android.platform.tools.dir}/renderscript/include:
                  ${android.platform.tools.dir}/renderscript/clang-inc
          lude"/>
```

将这个<property>放到下面的<path>元素后面:

```
<!--Renderscript 包含路径 -->
<path id="android.renderscript.include.path">
  <pathelement
location="${android.platform.tools.dir}/renderscript/include" />
  <pathelement
location="${android.platform.tools.dir}/renderscript/clangCHAPTER
include" />
</path>
```

再次执行 debug, 这次构建应该会成功。最后, 在 AVD1(见第 1 章)上安装 grayscale-debug.apk 并运行该应用程序。结果如图 8-14 所示。

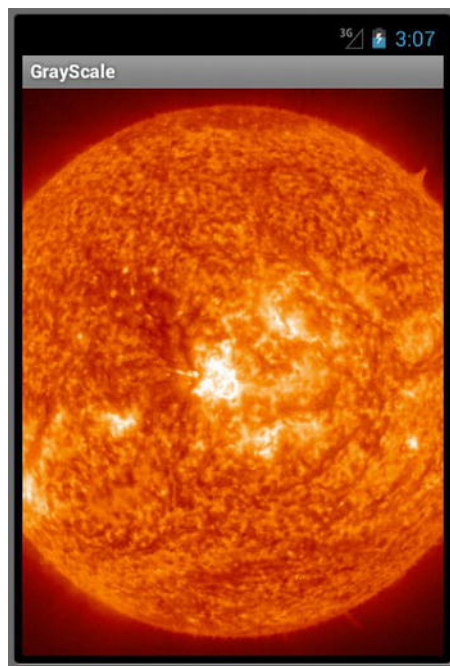


图 8-14 单击这张橙色的 Sun 图片，查看经过灰度处理后的图片

2. 使用 Eclipse 构建和运行 GrayScale

想要使用 Eclipse 构建 NDKScale，首先要想在第 1 章范例 1-10 中所介绍的那样创建一个新的 Android 项目。为了方便起见，可以使用下面的步骤创建项目：

- (1) 如果还未启动 Eclipse 的话，先启动 Eclipse。
- (2) 在 File 菜单中选择 New，在弹出的菜单中选择 Project。
- (3) 在 New Project 对话框中，展开向导树中的 Android 子节点(如果必要的话)，选择该节点下(如果必要的话)的 Android Application Project 分支，然后单击 Next 按钮。
- (4) 在 New Android App 对话框的 Application Name 一栏输入 GrayScale。这个名字同样会显示在 Project Name 中，它标识了 GrayScale 项目所在的文件夹/目录。
- (5) 在 Package Name 一栏输入 ca.tutortutor.grayscale。
- (6) 通过 Build SDK，选择合适的目标 Android SDK 版本。这个选择标识了你想要构建应用程序的 Android 平台版本。假设你只安装了 Android 4.1，那这里只会出现这一个目标并且已经被选中。
- (7) 通过 Minimum SDK，可以选择应用程序所需的最小 Android SDK 版本，也可以使用默认设置。
- (8) 如果想要自定义启动器图标的话，请选中 Create custom launcher icon 复选框。否则，会使用默认的启动器图标。
- (9) 保持 Mark this project as a library 复选框的未选中状态，因为你并没有打算创建一个库。
- (10) 保持 Create Project in Workspace 复选框的未选中状态，然后单击 Next 按钮。
- (11) 在接下来的 Configure Launcher Icon 界面中，对自定义启动图标做相应的调整，

然后单击 Next 按钮。

(12) 在随后的 Create Activity 界面中, 保持 Create Activity 复选框的选中状态, 选中 BlankActivity, 单击 Next 按钮。

(13) 在随后的 New Blank Activity 界面中, 在 Activity Name 一栏输入 GrayScale, 并在 Layout Name 一栏填写 main, 保持其他设置不变, 然后单击 Finish 按钮。

使用 Eclipse 的 Package Explorer 找到 GrayScale.java 源文件。双击该文件, 在弹出的编辑窗口中, 使用程序清单 8-9 中的内容替换该文件中的框架代码。

接下来, 在 src 节点下使用程序清单 8-10 的内容创建一个 grayscale.rs 文件节点, res 目录下创建一个 drawable-nodpi 目录, 并在该目录下添加 sol.jpg 文件。

要在 Eclipse 中运行 GrayScale, 在菜单栏中选择 Run, 然后在下拉菜单中选择 Run。如果出现了 Run As 对话框, 选择 Android Application 并单击 OK 按钮。Eclipse 会启动 AVD1 设备模拟器, 安装 GrayScale.apk, 并运行该应用程序, 输出结果如图 8-15 所示。

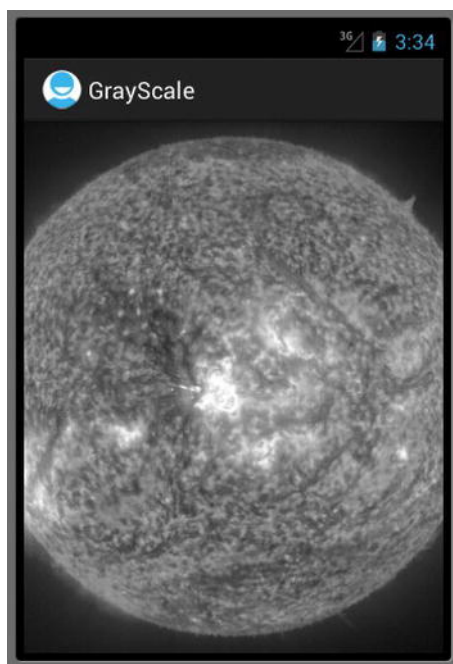


图 8-15 单击橙色的 Sun 图片后, 会出现经过灰度处理的图片

8.6 了解更多关于 Renderscript 的知识

8.6.1 问题

你已经迷上了 Renderscript 并想进一步了解它。例如你想知道如何在 root() 函数中接收 rsForEach() 的 usrData。

8.6.2 解决方案

下面的资源可以帮助你更多地了解 Renderscript:

- Romain Guy 和 Chet Haase 的“Learn about Renderscript”视频(<http://youtube.com/watch?v=5jz0kSuR2j4>)。这个一个半小时的视频涵盖了 Renderscript 图形和计算方面的知识, 很值得学习一下。
- Android 文档的 Renderscript 页面可以使你了解计算方面的重要信息。它还可以获得 Renderscript 方面的博客帖子的信息。
- android.renderscript 包的说明文档(<http://developer.android.com/reference/android/renderscript/package-summary.html>)可以帮助你重点了解 RenderScript 和 Allocation 类的不同类型。
- Renderscript 的参考页面(<http://developer.android.com/reference/renderscript/index.html>)提供了计算脚本中所需的 Renderscript 相关的函数的说明文档。

对于 root()来说, 这个函数至少需要两个参数来代表输入/输出的 allocation, 如 void root(const uchar4* v_in, uchar4* v_out)。此外, 也可以指定多于三个的参数来得到一个 usrData 值和输入 allocation 传入的 v_in 值的 x/y 坐标, 如下:

```
void root(const uchar4* v_in, uchar4* v_out, const void* usrData, uint32_t
x, uint32_t y)
```

8.6.3 实现机制

虽然 void root(const uchar4*, uchar4*, const void*, uint32_t, uint32_t)函数可能看起来有点吓人, 但使用起来并不难。例如, 程序清单 8-11 展示一个计算脚本的源代码, 该脚本就使用这个扩展的函数实现了图像的波浪效果, 就像水中看到的波纹一样。

程序清单 8-11 实现图像的波浪效果

```
#pragma version(1)
#pragma rs java_package_name(ca.tutortutor.wavyimage)

rs_allocation in;
rs_allocation out;
rs_script script;

int height;

void root(const uchar4* v_in, uchar4* v_out, const void* usrData, uint32_t x,
          uint32_t y)
{
    float scaledy = y/(float) height;
    *v_out = *(uchar4*) rsGetElementAt(in, x, (uint32_t) ((scaledy+
                                                             sin(scaledy*100)*0.03)*height));
}
```

```

void filter()
{
    rsDebug("RS_VERSION = ", RS_VERSION);
    #if !defined(RS_VERSION) || (RS_VERSION < 14)
        rsForEach(script, in, out, 0);
    #else
        rsForEach(script, in, out);
    #endif
}

```

程序清单 8-11 的 `root()` 函数忽略了 `usrData` (这里并不需要它), 但使用了传入的 `x` 和 `y` 的值。它还使用了传入的 `height` 的值, 该值代表了图像的高度。

函数首先使用 `height` 将传入的 `y` 的值缩小为一个 0 到 1 之间的浮点型小数。然后调用 Renderscript 的 `const void*rsGetElementAt(rs_allocation, uint32_t x, uint32_t y)` 函数来返回位于 `x` 和 `y` 位置的输入 `allocation` 元素, 然后赋值给 `*v_out`。

传入的 `x` 的值, 很显然刚好是 `root()` 函数的 `x` 参数的值。但是, 传入的 `y` 的值可能有点难以理解。它的意思就是将参数通过正弦函数处理一下, 这样从原始图像返回的像素就能产生波浪的效果。

程序清单 8-12 展示了 `WavyImage` 应用程序的源代码, 该应用程序会与储存在 `wavy.rs` 中的计算脚本进行通信。

程序清单 8-12 查看 Sun 的原始图片和波浪效果的图片

```

package ca.tutortutor.wavyimage;

import android.app.Activity;

import android.os.Bundle;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;

import android.renderscript.Allocation;
import android.renderscript.RenderScript;

import android.view.View;

import android.widget.ImageView;

public class WavyImage extends Activity
{
    boolean original = true;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        final ImageView iv = new ImageView(this);
    }
}

```

```

        iv.setScaleType(ImageView.ScaleType.CENTER_CROP);
        iv.setImageResource(R.drawable.sol);
        setContentView(iv);
        iv.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                if(original)
                    drawWavy(iv, R.drawable.sol);
                else
                    iv.setImageResource(R.drawable.sol);
                original = !original;
            }
        });
    }

    private void drawWavy(ImageView iv, int imID)
    {
        Bitmap bmIn = BitmapFactory.decodeResource(getResources(), imID);
        Bitmap bmOut = Bitmap.createBitmap(bmIn.getWidth(), bmIn.getHeight(),
                                           bmIn.getConfig());

        RenderScript rs = RenderScript.create(this);
        Allocation allocIn;
        allocIn = Allocation.createFromBitmap(rs, bmIn,

Allocation.MipmapControl.MIPMAP_NONE,

                                           Allocation.USAGE_SCRIPT);

        Allocation allocOut = Allocation.createTyped(rs, allocIn.getType());
        ScriptC_wavy script = new ScriptC_wavy(rs, getResources(), R.raw.wavy);
        script.set_in(allocIn);
        script.set_out(allocOut);
        script.set_script(script);
script.set_height(bmIn.getHeight());
        script.invoke_filter();
        allocOut.copyTo(bmOut);
        iv.setImageBitmap(bmOut);
    }
}

```

程序清单 8-12 和程序清单 8-9 最大的不同就是 `script.set_height(bmIn.getHeight())`，它将 bitmap 的高度传给了脚本的 `height` 变量，这样脚本就可以缩放 `y` 的值了。

如果构建并运行这个应用程序(和 `GrayScale` 使用一样的方式)，并且单击了 `Sun` 图片，那么你将看到如图 8-16 所示的结果。

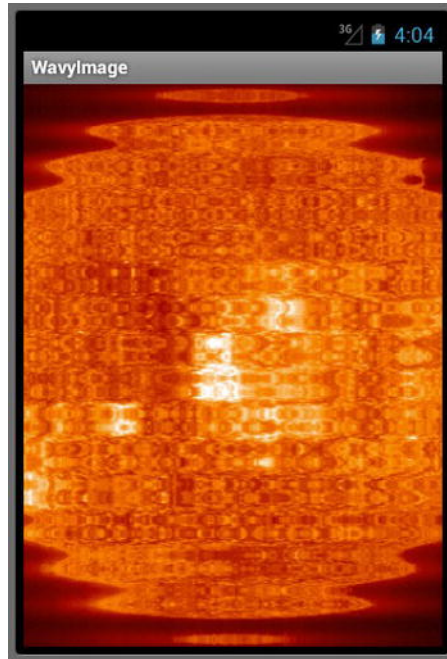


图 8-16 Sun 的波浪效果(或者可能是水纹效果)

8.7 小结

Android NDK 是 Android SDK 的补充，它提供了一个工具集让你可以使用 C 和 C++ 这样的本地代码语言实现你的部分应用程序。对于构建本地 Activity、处理用户输入、使用硬件传感器等操作，NDK 都提供了相应的头文件和库。

Renderscript 是由基于 C99 的语言(C 语言的现代版本)、两个编译器和一个运行时组成的，这些元素共同帮助实现高性能和好看的可视化图形(本地代码实现，但可移植性强)。这样不必使用 JNI 就获得了本地应用程序的速度和 SDK 应用程序的可移植性。

附录 A

Android 的脚本层

SL4A (Scripting Layer for Android, Android 脚本层)是一个平台, 之前称为 Android 脚本环境(AndroidScripting Environment), 用于在 Android 设备上安装脚本语言解释器, 通过解释器运行脚本。脚本可以访问很多 Android 应用程序可访问的 API, 但具体实现起来要简单得多。

注意:

SL4A 目前支持 Python、Perl、JRuby、Lua、BeanShell、RhinoJavaScript、Tcl 和 shell 脚本语言。可以在终端窗口(命令窗口)交互运行脚本, 或者可以通过 Locale(<http://www.twofortyfouram.com/>)在后台运行。Locale 是一个在预定时间或满足给定条件时(例如, 当你进入会场或剧院时运行一个脚本将手机的铃声模式设为震动)运行脚本的 Android 应用程序。

A.1 安装 SL4A

在使用 SL4A 之前, 首先要安装它。可以从托管在 Google 的项目网站(<http://code.google.com/p/android-scripting>)上下载最新版本的 APK 文件(撰写本书时最新版本为 sl4a_r6.apk)。

如果使用的是 Android 模拟器, 单击条形码图片下载 sl4a_r6.apk 文件。执行 `adb install sl4a_r6.apk` 命令在运行的模拟器上安装应用程序(如果看到消息 `device offline`, 可能需要多试几次)。图 A-1 是该应用程序在应用程序启动器中的图标。

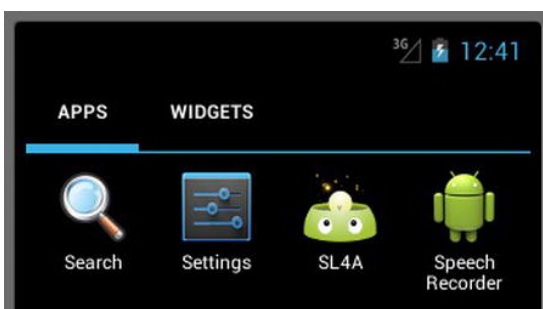


图 A-1 单击 SL4A 图标开始浏览 SL4A 应用程序的内容

A.2 浏览 SL4A

装好了 SL4A，接下来学习如何使用这个应用程序。单击 SL4A 的应用程序图标，就会看到如图 A-2 所示的 Scripts 界面。



图 A-2 同意或者拒绝使用习惯的跟踪

第一次运行 SL4A 时会出现使用习惯跟踪的对话框。你可以选择同意或者拒绝使用习惯信息的匿名收集行为。Scripts 界面开始会显示一个空的已安装的脚本列表。单击 MENU 按钮，会显示如图 A-3 所示的选项菜单。

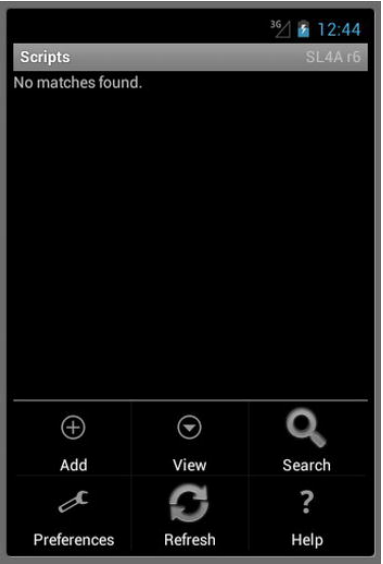


图 A-3 选项菜单可以在 Scripts 界面添加脚本和其他选项并执行其他一些任务

这个选项菜单共有 6 个选项：

- **Add:** 将文件夹(用于组织脚本和其他项)、内嵌 JavaScript 代码的 HTML 页面、shell 脚本和通过扫描条形码图片得到的脚本添加到 Scripts 列表中。文件夹和其他项都保存在设备的/sdcard/sl4a/scripts 目录中。
- **View:** 查看已安装的解释器(例如 Python 解释器)、trigger(判断在设备休眠时是否重复运行脚本的 intent,或是根据铃声模式运行脚本)和 logcat(查看系统调试输出的工具)。SL4A 自身只安装了 shell 解释器、HTML 和 JavaScript。
- **Search:** 根据输入的文本搜索脚本或其他内容。如果没有找到能匹配的结果,就会显示 “No matches found”。
- **Preferences:** 配置全部设置、脚本管理器、脚本编辑器和终端选项。
- **Refresh:** 刷新 Scripts 界面,显示变动,有可能在后台运行的脚本会更新列表。
- **Help:** 从维基百科的 SL4A 条目(<http://code.google.com/p/androidscripting/wiki/TableOfContents?tm=6>)获取 SL4A 帮助信息、YouTube 教学视频、终端帮助文档和 API 参考信息。

A.2.1 添加 Shell 脚本

下面向 Scripts 界面中添加一个简单的脚本,操作步骤如下:

- (1) 按下手机上的 MENU 键。
- (2) 在屏幕底部显示的菜单中单击 Add。
- (3) 从弹出的 Add 菜单中选择 Shell。
- (4) 在弹出的脚本编辑界面顶部的单行文本栏中输入 hw.sh,它是 shell 脚本的文件名。
- (5) 在多行文本框中输入 `#!/system/bin/sh`,换行后再输入 `echo "hello, world"` 和 `"sleep 3"`。第一行代码告诉 Android 到哪里去找 sh(命令行解释器程序),但这并不是必要的;第二行代码告诉 Android 在标准输出设备上输出一些文本。第三行代码会让脚本线程休眠 3 秒钟,从而保证输出文本可以显示出来(根据 <http://bit.ly/MRFrlU> 提供的信息,在显示终端窗口和运行脚本时会有两个线程存在。运行脚本的线程会在显示终端窗口的线程之前结束;如果不用 sleep 命令的话,将会看不到输出信息)。
- (6) 按下手机上的 MENU 键。
- (7) 在弹出的菜单中选择 Save & Exit 或者 Save & Run。

图 A-4 显示了单击 Save & Exit 之前编辑界面的样子。

现在 Scripts 界面会显示一个 hw.sh 项。单击这个脚本,然后会看到如图 A-5 中的图标菜单。

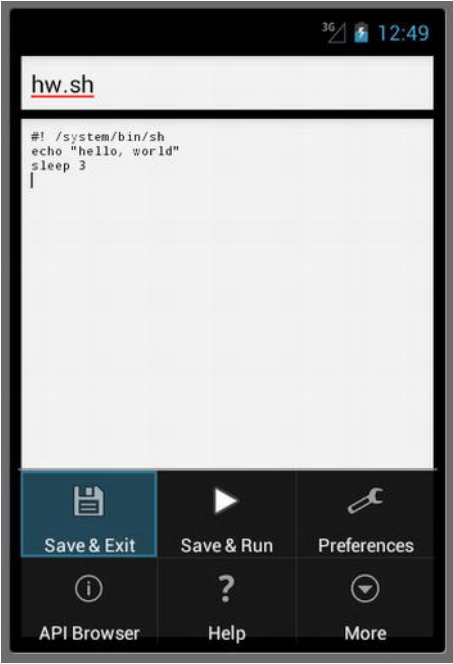


图 A-4 SL4A 的脚本编辑器界面
输入文件名和脚本内容

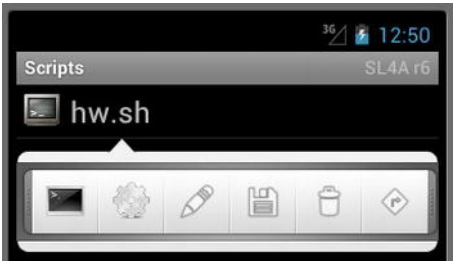


图 A-5 在这个图标菜单中,可以选择在终端窗口或后台
运行脚本、编辑脚本、重命名脚本以及删除脚本

这里,可以选择在终端窗口(最左边的图标)或后台(左起第二个图标)运行脚本。这两个图标用于运行 Shell 脚本。这时,虽然看不到脚本后台运行时的输出信息,但在终端窗口运行时应该会看到如图 A-6 所示的输出。

而执行 Sleep 命令会出现 “Process has exited. Close terminal?” 消息和 Yes、No 按钮。单击 Yes 按钮就会关闭终端窗口。

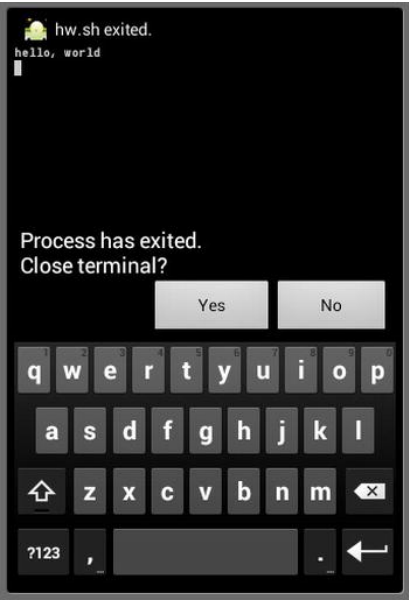


图 A-6

A.2.2 访问 Linux Shell

如果用前面的方法运行 `hw.sh` 脚本时看不到它的输出，那么可以在 Linux 的命令行解释器中查看，操作步骤如下：

- (1) 从 Scripts 界面的菜单中选择 View。
- (2) 从弹出的列表中选择 Interpreters。
- (3) 从 Interpreters 界面中选择 Shell，打开一个终端窗口。
- (4) 在终端窗口的 `$` 提示符后执行 `cd /sdcard/sl4a/scripts`，切换到 `hw.sh` 所在的目录。
- (5) 在 `$` 提示符后输入 `sh hw.sh` 运行 `hw.sh`。

图 A-7 是从命令行解释器中运行 `hw.sh` 的效果。还有执行 `exit` 命令(或按下手机的 BACK 键)后的效果。

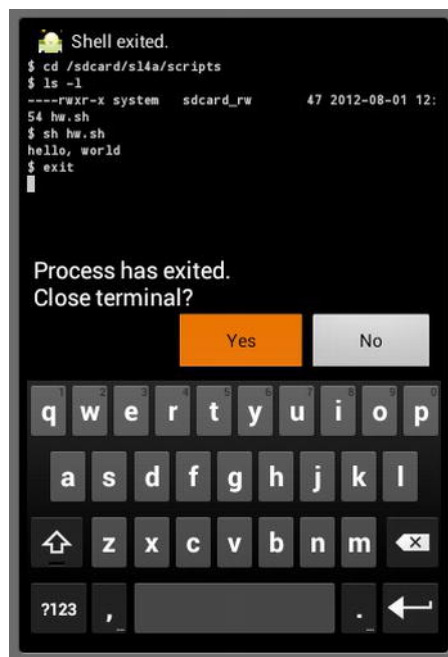


图 A-7 执行 `exit` 命令或者单击 BACK 按钮，会弹出 “Process has exited. Close terminal?”，单击 Yes 按钮退出命令行解释器

注意：

大多数的 shell 命令(如 `cpio` 和 `fdisk`)在没有 root 的模拟器上是无法使用的。想要了解完成这个示例需要的信息，可以在 <http://allencch.wordpress.com/2012/02/29/learn-to-root-android-using-emulator/> 查看 “Learn to root Android using emulator” 相关知识。

A.3 安装 Python 解释器

在 SL4A 中并不能做太多的事情，但可以通过这个特殊的应用程序安装 Python 或其他脚本语言。安装 Python 的操作步骤如下所示：

- (1) 在 Scripts 选项菜单中选择 View。
- (2) 在弹出的菜单中选择 Interpreters。
- (3) 按下电话的 MENU 键。
- (4) 选择菜单中的 Add。图 A-6 是添加解释器的菜单。

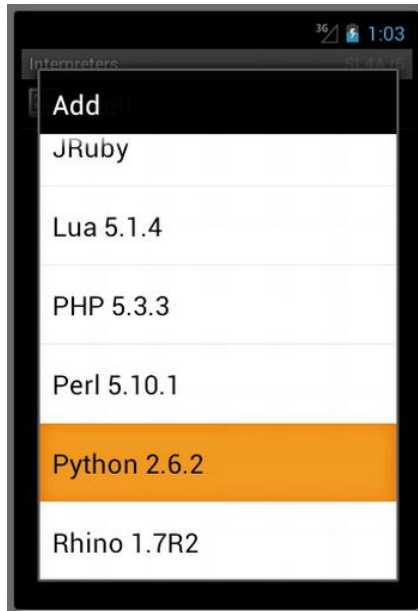


图 A-8 在 Add 菜单中可以选择要安装的解释器

(5) 单击 Python 2.6.2。SL4A 会在屏幕顶部显示一条浅蓝色的细线充当进度条。下载完成后，这个界面就会消失并返回到 Interpreters 界面。这时，在屏幕的左上角会看到通知图标。展开通知会看到图 A-9 所示的通知。

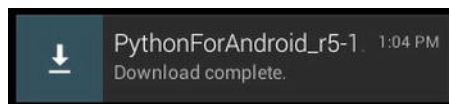


图 A-9 Python 的 APK 文件下载完成后的通知

- (6) 启动 shell 解释器并执行图 A-10 所示的命令。

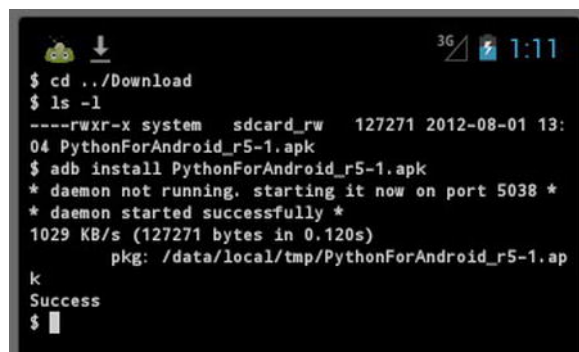


图 A-10 Python 的 APK 文件保存在/SDCard/Download 目录中

Python 的 APK 文件保存在/SDCard/Download 目录中。切换到这个目录后，执行 `adb install PythonForAndroid_r5-1.apk` 安装这个 APK。

(7) 退出 shell 和 SL4A。这时会在应用程序启动器中看到 Python 的图标。



图 A-11 SL4A 在 Android 应用程序启动器中显示了一个 Python 图标

(8) 单击 Python for Android 会出现图 A-12 所示的界面。



图 A-12 单击安装按钮下载和安装最新版本的 Python

(9) 单击 Install 按钮，会显示图 A-13 所示的下载界面。

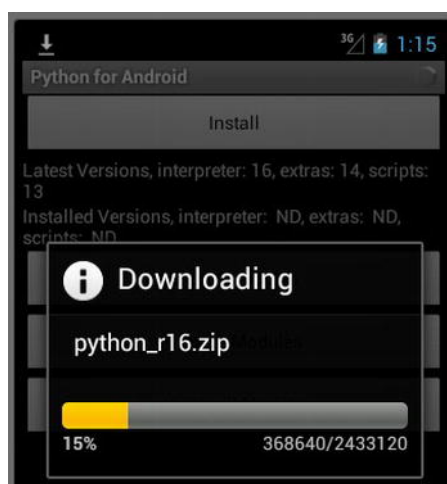


图 A-13 在 Android 模拟器上需要几分钟的时间下载并解压所有相关文件

(10) 下载几个压缩包并解压其他的内容, 这个过程结束后, Install 按钮会变成 Uninstall。关闭 Python for Android 并返回到应用程序启动器界面。

A.4 编写 Python 脚本

现在已经安装了 Python 2.6.2, 可以试试一个解释器了。单击 SL4A 图标会在屏幕上看到很多扩展名为.py 的脚本文件, 如图 A-14 所示。

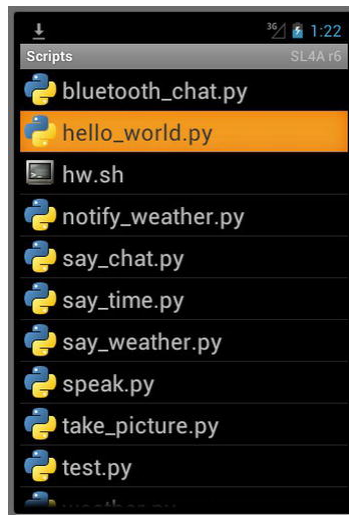


图 A-14 选择一个合适的脚本运行

选择一个文件(例如 hello_world.py)并运行。会看到和之前图 A-5 一样的选择列表。单击最左边(终端窗口)的图标会看到图 A-15 所示的内容。

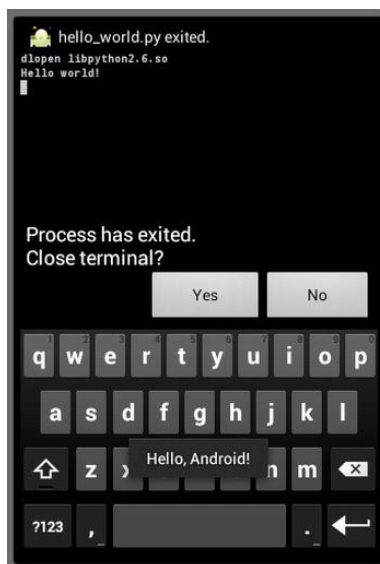


图 A-15 脚本会在键盘上显示一条 toast 消息

脚本会显示一条 toast 消息和其他输出。再来一次，保持脚本文件名高亮，同时选择铅笔图标来编辑脚本。图 A-16 显示了脚本的内容。



图 A-16 Python 需要导入一些其他模块

Python 需要导入一些其他模块。Import 语句导入了 Android 模块，Android 模块会提供一个 Android 类，在调用 makeToast()和其他方法前必须先实例化这个类。

注意：

Android 类的方法会返回 Result 对象。每个对象包含 id、result 和 error 字段：id 是对象的唯一标识，result 中包含方法的返回值(方法无返回值则返回 None)，error 表示错误信息(无错误则返回 None)。

访问 Python 解释器的方式基本和访问 Linux shell 的方式一样：

- (1) 选择 Scripts 界面选项菜单中的 View。
- (2) 在 View 上下文菜单中选择 Interpreters。
- (3) 在 Interpreters 界面选择 Python 2.6.2。

图 A-17 是一个 Python 示例对话，首先是显示版本号(来自 sys 模块的 version 变量)，然后显示了 math 模块的 pi 常量，最后执行 exit()函数退出 Python 解释器。

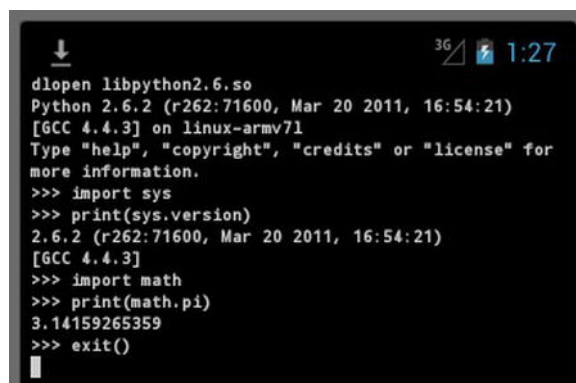


图 A-17 退出 Python 解释器的一种方法是执行 Python 的 exit()函数

注意：

同其他示例脚本一样，你应该查看 Google 的“android-scripting Tutorials”页面来了解 Python 解释器的特点和 SL4A 的常用知识，网址为 <http://code.google.com/p/androidscripting/wiki/Tutorials>。

附录 B

Android 工具一览

Apache Ant、带 Google ADT(Android Development Tools)插件的 Eclipse IDE、Google 的 SDK Manager 和 AVD Manager, 以及它的 SDK 工具和平台工具一起构成了 Android 应用程序开发生态系统。本附录会一一介绍每个 SDK 工具和平台工具。

注意:

这个附录在很多地方会引用 Android 系统启动方面的知识。想要了解 Android 系统的启动过程(以及在 Android Developer Phone1 方面该过程是如何执行的), 可以查看“Android Booting” (http://elinux.org/Android_Booting)。

B.1 SDK 工具

SDK 工具是 SDK 发布包中的基本工具, 保存在 tools 目录下。本节会介绍 Android SDK Revision 20 中的所有基本工具。

B.1.1 android

android(一个命令行工具)可以用来创建、删除、查看 AVD(Android 虚拟设备); 可以创建和更新 Android 项目; 还可以更新 Android SDK 到新平台、新插件和文档。

注意:

Android 的特性都被整合到了 ADT(Android Development Tool)插件中。

android 命令用法如下:

```
android [global options] action [action options]
```

可以整体为 android 命令指定各种全局选项。

选项包括：

- **-h 或 --help**：通过命令获得帮助信息。例如：`android -h create`, `android --help create avd`, `android -h createidentity`。
- **-s 或--silent**：运行静默模式(只显示错误信息)。
- **-v 或--verbose**：输出错误信息、警告信息和信息消息。

全局选项之后就是 **action**，**action** 会通知 **android** 执行一些任务。很多 **action** 都拥有相应的选项来进一步限制 **action**。表 B-1 展示了所有的 **action**。

表 B-1 支持的 action

Action	描 述	选 项
Avd	启动 AVD 管理器	无
create avd	创建新的 AVD	-a --snapshot 在 AVD 中放置一个快照文件，实现持久化 -b --abi AVD 使用的 ABI。如果该平台只有一个系统映像的 ABI，默认会自动选择它 -f --force 强制创建 AVD，若已有同名的 AVD，则直接覆盖 -n --name AVD 的名称(必需)。 -p --path 新创建的 AVD 的路径 -s --skin 新创建的 AVD 的皮肤 -t --target 新创建的 AVD 的 Target ID(必需)
createidentity	创建标识	-a --account 发布者账号(必需) -k --alias Key 的别名(必需) -p --storepass Keystore 的密码，将提示默认值 -s --keystore Keystore 的路径(必需) -w --keypass 别名密码，将提示默认值

(续表)

Action	描 述	选 项
create libproject	创建库项目	-k --package Android 库的包名(必需) -n --name 新项目的名称 -p --path 新项目的目录(必需) -t --target 新项目的 Target ID(必需)
Createproject	创建应用程序项目	-a --activity 创建的默认 Activity 的名称(必需) -k --package Android 应用程序的包名(必需) -n --name 新项目的名称 -p --path 新项目的目录(必需) -t --target 新项目的 Target ID(必需)
create test-project	创建 Android 项目 测试包	-m --main 要测试的应用程序的路径目录, 相对于测试项目的目录(必需) -n --name 新项目的名称 -p --path 新项目的目录(必需)
delete avd	删除现有 ADV	-n --name 要删除的 AVD 的名称(必需)
list	列出现有的 target 和 AVD	无
list avd	列出现有 AVD	-0 --null 行结束时使用\0 而不是\n。只有--compact 使用 -c --compact 精简输出(适用于脚本)

(续表)

Action	描 述	选 项
list sdk	列出远程 SDK 版本库	-a --all 列出所有可用的包(包括废弃不用的和已经安装的) -e --extended 显示每个包的详情 -o --obsolete 不建议使用。可以使用—all 替代 -s --no-https 下载时使用 HTTP 代替 HTTPS(默认) -u --no-ui 在控制台显示结果(没有界面)。默认为 true
list target	列出现有的 target	-0 --null 行结束时使用\0 而不是\n。只有—compact 使用 -c --compact 精简输出(适用于脚本)
move avd	Move or rename an AVD	-n --name 要移动或者重命名的 AVD 的名称(必需) -p --path AVD 新目录的路径 -r --rename AVD 的新名称
Sdk	启动 SDK 管理器	无
update adb	更新 ADB 来支持 SDK 插件中的声明的 USB 设备	无
update avd	更新 AVD 来匹配一个新的 SDK 目录	-n --name 要更新的 AVD 的名称(必需)
update libproject	更新 Android 库项目。必须要有 Android-Manifest.xml 文件	-p --path 项目目录(必需) -t --target 项目设置的 Target ID

(续表)

Action	描 述	选 项
Updateproject	更新 Android 项目。必须要有 Android-Manifest.xml 文件	-l --library 要添加的 Android 库的目录，相对于项目目录 -n --name 新项目的名称 -p --path 新项目的目录(必需) -s --subprojects 同时更新子目录中的项目，如测试项目 -t --target 项目设置的 Target ID
update sdk	当有新平台可用时更新 SDK	--proxy-host HTTP/HTTPS 代理主机(会覆盖已有的设置) --proxy-port HTTP/HTTPS 代理端口(会覆盖已有的设置) -a --all 所有的包，包括废弃不用的和没有依赖关系的 -f --force 强制替换包或包的一部分，即使有改动 -n --dry-mode 模拟更新但不下载或安装 -p --obsolete Deprecated. Use --all instead 不建议使用。可以使用—all 替代 -s --no-https 下载时使用 HTTP 代替 HTTPS(默认) -t --filter 通过逗号分隔的形式[platform, system-image, tool, platform-tool, doc, sample, source]来限制要更新的包的类型。这个选项还能使用 list sdk --extended 返回的标识符
update testproject	更新 Android 测试项目。必须要有 Android-Manifest.xml 文件	-m --main 要测试的应用程序的路径目录，相对于测试项目的目录(必需) -p --path 项目目录(必需)

B.1.2 apkbuilder

Apkbuilder 是一个不建议使用的工具，之前一直用来构建 APK 文件。想了解更多关于该工具的信息和它的新 Java 包，可以在命令行上运行 apkbuilder。

注意：

查看“如何在命令行中使用 SDK 工具+持续集成工具 CruiseControl 构建 Android 应用程序包(.apk)”来了解 apkbuilder 是如何构建 APIK 的，网址为 <http://asantoso.wordpress.com/2009/09/15/how-to-build-androidapplication-package-apk-from-the-command-line-using-the-sdktools-continuously-integrated-using-cruisecontrol/>。

B.1.3 ddms

ddms(Dalvik Debug Monitor Server)提供了端口转发服务、设备的屏幕捕捉、设备的线程和堆栈信息、logcat、进程、无线电状态信息、来电和短信模拟、位置数据模拟等更多功能。

ddms 的用法如下：

ddms

注意：

查看 Google 的“Using DDMS”页面(<http://developer.android.com/tools/debugging/ddms.html>)深入了解如何使用这个工具。

B.1.4 dmtracedump

dmtracedump 可以通过日志跟踪文件(而不使用 traceview)生成一个栈的使用情况的图表。

dmtracedump 的用法如下(为了便于阅读，分两行显示)：

```
dmtracedump [-ho] [-s sortable] [-d trace-file-name] [-g outfile]
             trace-file-name
```

这个工具会从 trace-file-name.data 和 trace-filename.key 文件中加载日志跟踪数据。表 B-2 是它的各种选项：

表 B-2 支持的选项

选 项	描 述
<i>-d trace-file-name</i>	使用这个跟踪文件实行一个差分操作
<i>-g outfile</i>	向 <i>outfile</i> 中写入一个图表
<i>-h</i>	启用 HTML 输出
<i>-k</i>	在写入图表时保留中间生成的 DOT 文件
<i>-o</i>	生成 dmtrace 文件而不是性能分析文件
<i>-s sortable</i>	提供相对于 <i>sortable javascript</i> 文件路径的 URL 基本路径
<i>-t threshold</i>	提供图表中包含节点的阈值百分比

B.1.5 draw9patch

draw9patch 是一个有界面的工具，可以很轻松地创建 9-patch(可改变大小的图片)。

Google 的“9-patch”文档(<http://developer.android.com/guide/topics/graphics/2dgraphics.html#nine-patch>)介绍了 9-patch 的相关信息。

draw9patch 的用法如下：

```
draw9patch
```

注意：

查看 Google 的“Draw 9-patch”页面(<http://developer.android.com/tools/help/draw9patch.html>)深入了解如何使用这个工具。

B.1.6 emulator

emulator 会启动一个用 AVD 表示的模拟设备。可以在该设备上安装和运行应用程序。

emulator 的用法如下：

```
emulator -avd avd_name [-option [value]] ... [-qemu args]
```

表 B-3 展示了各种选项。

表 B-3 支持的选项

选 项	描 述
-audio backend	使用指定的音频端
-avd name	使用指定的 AVD，这个必需的选项指定了一个 AVD 来加载该模拟器
-bootchart timeout	启用 bootcharting。想要了解更多信息，请查看 http://elinux.org/Using_Bootchart_on_Android
-cache filepath	使用 filepath 作为工作时的缓存区。filepath 的值是一个绝对路径或者是当前工作目录的相对路径。如果没有指定缓存文件，模拟器会默认使用一个临时文件代替
-cache-sizesize	指定缓存分区的大小(单位为 Mb)
-camera-backmode	模拟后置摄像头模式
-camera-front mode	模拟前置摄像头模式
-charmap file	选择 file 中的关键字符映射表
-cpu-delay cpudelay	通过设置 delay 来模拟减慢 CPU 的运行速度。delay 是一个 0~1000 的整型值。delay 与时钟频率或其他绝对指标也没有什么关系，它只是表示一种抽象的、相对的模拟行为。性能的提升也经常与 delay 的值没有什么直接关系

(续表)

选 项	描 述
-data <i>filepath</i>	使用 <i>filepath</i> 作为工作时的用户数据磁盘镜像。另外,你可以指定一个当前工作目录的相对路径。如果不使用-data 的话,模拟器会在 AVD 的存储位置中查找名为 <i>userdata-qemu.img</i> 的文件
-datadir <i>dir</i>	指定当前用户数据的保存位置
-debug <i>tags</i>	启用/禁用 <i>tags</i> 所指定的调试消息类型, <i>tags</i> 是一个用空格/逗号/字段名分隔的调试组件名称的列表。使用-help-debug-tags 可以打印所有你可以使用的调试组件列表。例如: -debug init
-debug-no-tag	禁用 <i>tag</i> 类型的调试信息
-debug-tag	启用 <i>tag</i> 类型的调试消息。使用-help-debug-tags 可以打印所有你可以在 <i>tag</i> 中使用的调试组件列表
-dns-server <i>servers</i>	在模拟系统里使用指定的 DNS 服务器。服务器的名字必须是一个以逗号分隔的 4 个 DNS 服务器名称或 IP 地址的列表
-dpi-device <i>dpi</i>	调整模拟器的分辨率从而匹配物理设备的屏幕大小。默认值为 165。参见-scale
-force-32bit	总是使用 32 位的模拟器
-gps <i>device</i>	重定向到改变字符设备的 NMEA GPS。这个命令可以模拟连接到外部字符设备或者 socket 上的一个 NMEA 兼容的 GPS 单元。 <i>device</i> 的格式必须符合特定 QEMU(Quick Emulator)串行设备的规范
-gpu on	开启模拟器的图形加速功能。这个选项只适合于使用 API15、revision3 及以上版本的模拟器
-help	打印一个所有 emulator 选项的列表
-help-option	打印特定启动 <i>option</i> 的帮助信息
-help-all	打印所有启动选项的帮助信息
-help-build-images	打印构建 Android 时用到的磁盘镜像的相关帮助信息
-help-char-devices	打印关联到一个模拟设备或者通信信道上的字符设备的帮助信息。示例 <i>baokuostdio</i> 和 <i>pipe:filename</i>
-help-debug-tags	打印-debug tags 选项相关的帮助信息
-help-disk-images	打印使用模拟器磁盘镜像相关的帮助信息
-help-environment	打印模拟器环境变量相关的帮助信息
-help-keys	打印当前密钥映射关系
-help-keyset-file	打印定义自定义密钥映射文件相关的帮助信息
-help-sdk-images	打印使用 SDK 时的磁盘镜像相关的帮助信息
-help-virtual-device	打印虚拟设备管理相关的帮助信息

(续表)

选 项	描 述
-http-proxy <i>proxy</i>	通过一个特定的 HTTP/HTTPS 代理(proxy)来建立所有的 TCP 连接。 <i>proxy</i> 的值可以是 <code>http://server:port</code> 或者 <code>http://username:password@server:port</code> 。 <code>http://</code> 前缀是可以省略的。如果没有指定 <code>-http-proxy proxy</code> 选项, 模拟器会查找 <code>HTTP_PROXY</code> 环境变量并且自动使用任意上面介绍的格式相匹配的 <i>proxy</i> 格式
-image <i>file</i>	已经废弃, 而使用 <code>-system file</code> 选项代替
-initdata <i>filepath</i>	指定一个文件, 它的内容在使用 <code>-wipe-data option</code> 选项时会被复制到一个新的 <code>userdata</code> 磁盘镜像中。默认情况下, 模拟器会从 <code>system/userdata.img</code> 中复制数据。另外, 你可以指定一个当前工作目录的相对路径。同样参见 <code>-wipe-data</code> 。注意, <code>-initdata</code> 等同于 <code>-init-data</code>
-kernel <i>filepath</i>	使用位于 <i>filepath</i> 中的内核
-keyset <i>file</i>	使用指定的 <code>keyset</code> 文件(<i>file</i>)代替默认的文件。 <code>keyset</code> 文件定义了模拟器和主机键盘键的按键关系
-logcat <i>logtags</i>	启用 <code>logcat</code> 输出 <i>logtags</i> 指定的信息。如果定义了 <code>ANDROID_LOG_TAGS</code> 环境变量并且不为空, 默认就会使用这个环境变量的值来启用 <code>logcat</code> 输出
-memcheck <i>flags</i>	启用内存访问检查
-memory <i>size</i>	指定物理 RAM 的大小(Mb)
-netdelay <i>delay</i>	通过设置 <i>delay</i> 的值来模拟器网络延迟。默认值为 <code>none</code> 。其他的值可以参见 http://developer.android.com/tools/devices/emulator.html#netspeed
-netfast	<code>-netspeed full -netdelay none</code> 选项的简化版
-netspeed <i>speed</i>	通过设置 <i>speed</i> 的值来模拟网络速度。默认值为 <code>full</code> 。其他的值可以参见 http://developer.android.com/tools/devices/emulator.html#netspeed
-no-audio	禁用当前模拟器的音频
-no-boot-anim	禁用模拟器的启动动画。禁用启动动画可以加快模拟器的启动速度
-no-cache	启动模拟器但没有缓存分区
-no-jni	Dalvik 运行时禁用 JNI 的检查
-no-skin	禁用所有的模拟器皮肤
-no-window	禁用模拟器的图形窗口显示
-noaudio	同 <code>-no-audio</code>
-nocache	同 <code>as -no-cache</code>
-nojni	同 <code>-no-jni</code>
-noskin	同 <code>-no-skin</code>

(续表)

选 项	描 述
-no-snapshot	执行一个全启动，加载时不会自动保存到快照。但是，QEMUvmload 和 vmsave 指令会对保存的快照(snapstorage)起作用
-no-snapshot-load	不会从快照自动启动，而是执行一个全启动
-no-snapshot-save	不管状态是否变化，在退出时都不会自动保存到快照
-no-snapshot-update-time	在恢复时不会试图更新快照的时间
-no-snapstorage	不挂载快照文件(所有的快照功能都会被禁用)
-onion image	在屏幕上使用 PNG 图片
-onion-alpha percent	指定 onion 皮肤的半透明值(百分比)。默认为 50%
-onion-rotation position	指定 onion 皮肤的方向。position 的值必须为 0、1、2、3
-partition-size size	指定系统/数据分区的大小(Mb)
-port port	通过 port 的值设置模拟器的控制台端口号。控制台端口号的值必须是 5554~5584 其中的一个。port+1 是不能用的，它是留给 adb 用的
-ports consoleport,adbport	指定控制台和 adb 使用的 TCP 端口号
-prop name=value	启动时，设置指定的系统属性
-qemu arguments...	传递参数给 QEMU 软件。重点：使用这个选项时，得确保它是最后的一个选项，这是因为所有它后面的选项都会认为是 QEMU 的选项
-qemu -h	显示 QEMU 的帮助信息
-radio device	改变特定字符设备的无线电模式。device 的格式必须符合特定 QEMU 串行设备的规范
-ramdisk filepath	使用 filepath 作为 ramdisk 的镜像文件。默认值为 system/ramdisk.img。另外，你可以指定一个当前工作目录的相对路径。更多磁盘镜像的信息可以查看-help-disk-images
-raw-keys	禁用 Unicode 键盘反向映射
-report-console socket	启动模拟器时，将模拟器的控制台端口报告给远程的第三方。socket 的格式必须为 tcp:port [, server] [,max=seconds] 或者 unix:port [, server] [, max=seconds]。更多信息可以查看-help-report-console
-scale scale	缩放模拟器窗口。scale 就是缩放比例，取值范围为 0.1 到 3。可以为 scale 指定一个 DPI 值，只需要在 scale 后面添加 dpi 后缀即可。如果设为 auto 就是告诉模拟器自动选择最佳的窗口大小
-screen mode	设置模拟的窗口模式
-sdcard filepath	使用 filepath 作为 SD 卡的镜像。默认值为 system/sdcard.img。另外，你可以指定一个当前工作目录的相对路径。更多磁盘镜像的信息可以查看-help-disk-images

(续表)

选 项	描 述
-shared-net-id <i>number</i>	加入到共享网络中, 使用的 IP 地址为 10.1.2. <i>number</i>
-shell	在当前终端创建一个 root shell 控制台。即使模拟系统中 ADB 守护进程坏掉了也可以使用这个命令。在 shell 中按下 Ctrl-c 键会终止模拟器而不是 shell
-shell-serial <i>device</i>	启用 root shell(这里和-shell 一样)并且指定 QEMU 字符设备与 shell 进行交互。 <i>device</i> 必须是一个 QEMU 设备类型。-shell-serial stdio 就等同于-shell, 而-shell-serial tcp::4444,server,nowait 可以让你通过 TCP 端口 4444 与 shell 进行通信
-show-kernel <i>name</i>	显示内核信息
-skin <i>skinID</i>	不建议使用。建议使用 AVD 的设置皮肤选项而非这个选项。这个选项有时候会产生意想不到和令人误解的结果, 因为使用密度(density)来渲染皮肤可能是不支持的。而 AVD 的设置选项则可以使用每个皮肤的默认密度(density), 需要时也可以设置这个密度
-skindir <i>dir</i>	不建议使用。参见-skin 中所述的原因
-snapshot <i>name</i>	指定存储区中的一个快照的名称(默认名称为 default-boot)用来自动启动和自动保存模拟器信息
-snapshot-list	显示所有可用快照的列表
-snapshotstorage <i>file</i>	指定一个文件, 该文件中(默认为 <i>datadir/snapshots.img</i>)包含所有状态的快照
-sysdir <i>dir</i>	在 <i>dir</i> 中搜索系统磁盘镜像文件
-system <i>filepath</i>	读取 <i>filepath</i> 中的初始系统镜像文件
-tcpdump <i>filepath</i>	抓取网络包到 <i>filepath</i> 中
-trace <i>name</i>	启用代码性能分析(按 F9 开始), 写入到指定的文件
-timezone <i>timezone</i>	通过 <i>timezone</i> 设置模拟器的时区, 而不用主机的时区。 <i>timezone</i> 必须按照时区信息格式来指定。例如, America/Los_Angeles 和 Europe/Paris
-verbose	启用输出详细信息, 等同于-debug-init。可以通过 Android 环境变量 ANDROID_VERBOSE 设置模拟器的默认详细信息输出选项。可以在一个逗号隔开的列表中定义要使用的选项(只需要定义每个选项的关键部分, 参见-debug-tags)。例如, setANDROID_VERBOSE=init,modem 即定义了 ANDROID_VERBOSE 使用-debug-init 和-debug-modem 选项。更多关于调试标识的信息, 可以参见-help-debug-tags
-version	显示模拟器版本号
-webcam-list	列出模拟器可以使用的所有网络摄像头

(续表)

选 项	描 述
-wipe-data	还原当前用户数据的镜像文件(即通过-datadir 和-data, 指定的文件或者默认的文件)。模拟器会将镜像文件中的所有用户数据删除, 然后在启动前将通过-initdata 所指定的文件中的内容复制到镜像文件中。参见-initdata。更多磁盘镜像的信息, 可以查看-help-disk-images

B.1.7 etc1tool

etc1tool 可以将 PNG 图片编码为 ETC1(Ericsson Texture Compression)压缩标准的图片, 同样也可以将使用 ETC1 压缩的图片解码为 PNG 图片。

etc1tool 的用法如下(为了便于阅读, 分两行显示):

```
etc1tool infile [--help | --encode | --encodeNoHeader | --decode]
               [--showDifference diff-file] [-o outfile]
```

这种用法展示了如下内容:

- *infile* 表示输入的文件, 可以对它进行压缩或者它本身就包含了已经压缩过的数据。
- --help 打印用法信息。
- --encode 将 PNG 文件转换为 ETC1 文件。在 etc1tool 没有指定任何选项时, 这个就是默认的模式。
- --encodeNoHeader 将 PNG 文件转换为一个原始 ETC1 数据文件(没有文件头)。
- --decode 将 ETC1 文件转换为一个 PNG 文件。
- --showDifference diff-file 会将原始文件与编码后的图片的差异部分写入到一个文件(diff-file)中(只有在编码时才有效)。
- -o *outfile* 表示输出文件的名称。当未指定 *outfile* 文件时, 输出文件即为输入文件的名称加上适当的后缀(.pkm 或者.png)。

B.1.8 hierarchyviewer

hierarchyviewer 是一个有界面的工具, 可以用来调试和优化你的应用程序用户界面。它提供了一个可视化的布局结构(Layout View)并且可以将布局进行放大显示(Pixel PerfectView)。

Hierarchyviewer 的用法如下:

```
hierarchyviewer
```

注意:

可以查看 Google 的“Optimizing Your UI”页面(<http://developer.android.com/tools/debugging/debugging-ui.html>)或者本书其他章节来进一步了解如何使用这个工具。

B.1.9 hprof-conv

hprof-conv 可以将 SDK 生成的 HPROF 文件转换为一种标准的格式，这样你就可以使用相应的性能分析工具查看这个文件了。

hprof-conv 的用法如下：

```
hprof-conv infile outfile
```

可以对 `infile` 或 `outfile` 是用 “-” 来指定标准输入和标准输出。

B.1.10 lint

Lint 是一个静态检查工具，它可以分析 Android 项目中存在的一些问题，例如正确性、安全性、性能问题、可用性和可访问性，检查 XML 资源、图片、ProGuard 配置文件、源文件、甚至是编译后的字节码。

lint 的用法如下：

```
lint [flags] project directories
```

表 B-4 描述了各种选项。

表 B-4 支持的选项

选 项	描 述
<code>--check list</code>	只检查 <i>list</i> 中指定的问题项。这个过程会先禁用所有问题项然后再重新启动给定的 <i>list</i> 中的问题项。 <i>list</i> 中的内容就是要检查的问题项 ID 或分类，以逗号分隔
<code>--config filename</code>	使用给定的配置文件来决定问题项是否应该启用。如果项目中已经存在了 <code>lint.xml</code> 文件，这个配置文件会作为备用
<code>--disable list</code>	禁用 <i>list</i> 中指定的分类和问题项。 <i>list</i> 中的内容就是要检查的问题项 ID 或分类，以逗号分隔
<code>--enable list</code>	启用 <i>list</i> 中指定的问题项。这个过程会检查所有的默认问题项以及 <i>list</i> 中的问题项。 <i>list</i> 中的内容就是要检查的问题项 ID 或分类，以逗号分隔
<code>--exitcode</code>	发现错误时将退出码设为 1
<code>--fullpath</code>	在错误的输出信息中使用全路径
<code>--help</code>	打印详细的帮助信息
<code>--help topic</code>	打印特定主题的帮助信息
<code>--html filename</code>	创建一个 HTML 报告。如果 <i>filename</i> 是个目录(或一个新的没有扩展名的文件名)，lint 会为每个扫描的项目创建一个单独的报告
<code>--list</code>	列出所有可用的问题项 ID 并且退出

(续表)

选 项	描 述
--nolines	在输出信息里不包含错误行的源文件。默认情况下，输出的错误信息中会包含出错行的源代码片段，但这个选项会关闭此功能
--quiet	不显示进度
--show	列出所有问题项，并伴有详细说明
--show ids	针对 ids 列表中的问题项做详细的说明
--showall	不截断长的信息、备用位置列表等
--simplehtml <i>filename</i>	创建一个简单的 HTML 报告
--urlfilepath= <i>url</i>	在 HTML 报告中添加链接，将本地路径前缀替换为 <i>url</i> 前缀。这种映射关系可以是一个以逗号分隔的路径前缀与 URL 前缀对应关系的列表，例如 C:\temp\Proj1=http://buildserver/sources/temp/Proj1。想要关闭文件的链接关系，可以使用--url none
--version	打印版本信息并退出
-w --nowarn	只检查错误(忽略警告)
-Wall	检查所有的警告，包括默认关闭的那些警告
-Werror	将所有的警告当作错误对待
--xml <i>filename</i>	创建一个 XML 报告

lint 还会返回下面退出状态码中的一个：

- 0：成功
- 1：Lint 检测到错误
- 2：Lint 用法
- 3：不能破坏现有文件
- 4：Lint 帮助
- 5：无效的命令行参数

B.1.11 mkSDcard

mkSDcard 可以快速创建一个 FAT32 格式的磁盘镜像，然后你可以将这个镜像加载到模拟器上，从而实现模拟设备上的 SD 卡。因为在使用 AVD Manager 创建 AVD 时可以指定 SD 卡，所以通常情况下都是在这里创建 SD 卡。而这个工具创建的 SD 卡不会绑定到某个 AVD，因此，在需要多个模拟器共享一个虚拟 SD 卡时，这个工具会非常有用。

mkSDcard 的用法如下：

```
mkSDcard -l label size file
```

这种用法展示了如下内容：

- -l *label* 指定了要创建的磁盘镜像的卷标。

- *size* 通过一个整型值指定了要创建的磁盘镜像的大小(单位为字节)。也可以通过 Kb、Mb、Gb 来指定 size, 即在 size 的后面加上 K、M 或者 G。例如: 1048576K、1024M、1000G。
- *file* 指定了创建的磁盘镜像的路径/文件名。

创建好磁盘镜像文件后, 可以在模拟器启动时通过模拟器的 `-sdcard` 选项将镜像文件加载到模拟器上。`-sdcard` 选项的用法如下:

```
emulator -sdcard file
```

B.1.12 monitor

monitor(Android Debug monitor, Android 调试监视器)是一个有界面的工具, 界面中提供了很多 Android 应用程序调试和分析工具。使用 monitor 不需要安装任何 IDE(integrated development environment, 集成开发环境), 例如 Eclipse, 它封装了如下一些工具(本附录的其他章节曾介绍过):

- ddms
- hierarchyviewer
- traceview
- Tracer for OpenGL ES

monitor 的用法如下:

```
monitor
```

然后启动 Android 模拟器或者通过 USB 线连接一个 Android 设备, 再在 Device 窗口中选择相应的设备即可监控设备。

B.1.13 monkeyrunner

monkeyrunner(非常适合于功能测试)提供了一个 API, 可以在 Android 代码以外的地方通过编程的方式控制 Android 设备或者模拟器。

注意:

monkeyrunner 可以用于功能测试。想要测试一个单一的 Activity, 还可以使用 `android.test.ActivityInstrumentationTestCase2` 类。

使用 monkeyrunner, 你可以编写一个 Python 程序来安装 Android 应用程序或者测试一个应用程序包、然后运行、向应用程序发送键盘指令、截屏以及将截屏文件保存到某个工作站上。

注意:

可以查看 Google 的“monkeyrunner”页面(http://developer.android.com/tools/help/monkeyrunner_concepts.html)来进一步了解如何使用这个工具。

B.1.14 sqlite3

sqlite3 可以让你管理 Android 应用程序所创建的 SQLite 数据库，既可以通过远程 shell 管理设备也可以通过你自己的主机机器进行管理。它有很多有用的命令，例如.dump 会打印一个数据表的内容，.schema 会打印现有的某个表的 SQL CREATE 语句。sqlite3 还能让你执行 SQLite 命令。

sqlite3 的用法如下：

```
sqlite3 [OPTIONS] [DATABASENAME]
```

sqlite3 可以单独运行，也可以只指定选项运行，还可以指定选项和数据库文件名运行。如果数据库文件不存在，则会创建一个新的数据库。

表 B-5 描述了各种选项。

表 B-5 支持的选项

选 项	描 述
-bail	遇到错误时停止
-batch	强制批量输入/输出
-column	设置输出模式为按字段显示
-csv	设置输出模式为按 csv(逗号分隔)的格式显示
-echo	执行前打印输入的命令
-help	打印帮助信息
-html	设置输出模式为通过 HTML 格式显示
-initfilename	读取并处理已经命名的文件
-interactive	强制进行 I/O 交互
-line	设置输出模式为按行显示
-list	设置输出模式为按列表显示
-[no]header	启用或者禁用头文件
-nullvalue'text'	设置 text 字符串为 NULL 值
-separator'x'	设置输出文件以 x 分隔
-stats	每次结束前打印内存信息
-version	显示 SQLite 版本号

注意：

可以查看 Google 的“sqlite3”页面(<http://developer.android.com/tools/help/sqlite3.html>)或者本书其他章节来进一步了解如何使用这个工具。

B.1.15 systrace

Systrace 是一个 Python 脚本，它会调用一个名为 `atrace` 的 Linux 程序从 Linux 内核中收集系统和用户行为的跟踪数据。`systrace` 然后会生成一个 HTML 文件将这个数据(通过 Google Chrome 浏览器)显示为一组垂直堆叠的时间序列图。

SDK 的 `tools` 目录下的 `systrace` 子目录中包含有 `systrace.py` 和一些其他用于在 HTML 中生成图表的文件。你需要安装一个 Python 解释器来运行 `systrace.py`(除非你已经安装过了)。

运行这个脚本前，还需要启用设备上的各种跟踪设置。步骤如下：

- (1) 在设备的主界面单击 MENU 键。
- (2) 在弹出的菜单中选择 System setting。
- (3) 在之后出现的 Settings 界面的 SYSTEM 一栏下面选择 Developer options。
- (4) 在之后出现的 Developer options 界面的 MONITORING 一栏下面选择 Enable traces。
- (5) 在弹出的 Select enabled traces 菜单中选择希望启用的跟踪选项。例如，可以选择 Graphics 和 View。
- (6) 单击 OK 按钮即可启用这个跟踪项。

图 B-1 显示了启用跟踪项的界面。

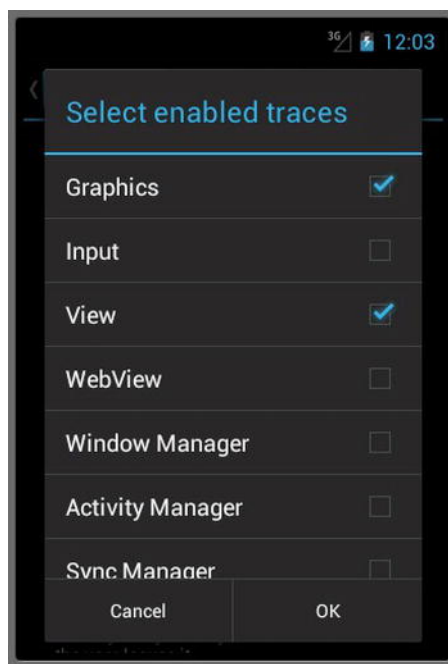


图 B-1 启用了 Graphics 和 View 的跟踪

切换到 `systrace` 目录并执行下面的命令：

```
python systrace.py
```

这个工具大概会运行 5 秒(这个是默认值，你也可以通过向 `systrace.py` 传入 `-t` 或者 `-time`

选项来设置这个值)。在这段时间内,你可以启动要跟踪的应用程序并且操作一下应用程序。

systrace 生成了如下消息:

```
capturing trace... done
downloading trace... done
```

另外还生成了一个关于将 trace.html 文件写入到 systrace 目录的消息(可以通过向 systrace.py 传入 -o 选项来更改 trace.html 的名字)。

现在,你得到了 trace.html 文件,你可以查看结果了。图 B-2 显示了结果图表的样子。



图 B-2 跟踪结果显示在了 Google Chrome 浏览器中。双击一个带颜色的区域可以放大显示

可以在真实设备上运行 systrace,但却不可以在 Android 模拟器上面运行它。如果你试图这样做, systrace 会生成如下的输出(为了便于阅读,最后一行进行了格式化):

```
Traceback (most recent call last):
  File "systrace.py", line 212, in <module>
    main()
  File "systrace.py", line 124, in main
    ready = select.select([adb.stdout, adb.stderr], [], [adb.stdout,
adb.stderr])
select.error: (10093, 'Either the application has not called WSASStartup,
or WSASStartup failed')
```

这个错误信息是 atrace 生成的,即在试图创建或者打开/sys/kernel/debug 目录的子目录

下的文件时会发送错误消息。这表明在 Android 模拟器中没有 debug 子目录。

你不能只是简单地执行 `mkdir /sys/kernel/debug` 命令来创建 debug 目录。相反，你必须启用内核扩展调试，而这个操作只能在重新构建内核时才可以实施(可以执行 `emulator -help-buildimages` 来了解这一过程的更多信息)。

B.1.16 traceview

Traceview 是一个带界面的工具，用来查看在代码中使用 `android.os.Debug` 类创建的日志追踪信息。traceview 能够帮助调试你的应用程序并了解它的性能。

traceview 的用法如下：

```
traceview
```

注意：

可以查看 Google 的“Profiling with Traceview and dmtracedump”页面(<http://developer.android.com/tools/debugging/debuggingtracing.html>)来进一步了解如何使用这个工具。

B.1.17 OpenGL ES 的 Tracer 工具

Tracer 是一种工具，用于在你的 Android 应用程序中分析 OpenGL 的嵌入式系统(ES)代码。该工具允许你捕获 OpenGL ES 命令和图像帧，以帮助你了解你的图形命令是如何执行的。

Tracer 并不是单独的工具，它是 ADT 插件的组成部分，同时还是 Android Device Monitor (monitor)的一部分。

注意：

可以查看 Google 的“Tracer for OpenGL ES”页面(<http://developer.android.com/tools/help/gltracer.html>)来进一步了解如何使用这个工具。

B.1.18 zipalign

zipalign 是一个压缩包对齐工具，可以很好地优化 Android APK。对齐的目的就是确保所有未压缩数据都从一个特定的对齐值开始(相对于 file 的起始位置)。

Zipalign 可以让 APK 中所有未压缩的数据(例如图片和原始文件)进行 4 字节对齐。这样，所有的对齐部分都可以直接通过 `mmap()` 函数访问，`mmap()` 函数会将文件或者设备映射到内容中，即使它们中包含对齐限制的二进制数据。这样做的好处就是减少了运行应用程序时的内存消耗。

警告：

应该在 APK 使用你的私钥签名之后再使用 zipalign。如果在签名前运行 zipalign，之后的签名过程会抵销掉之前的对齐效果。

同样，不要对已经对齐好的包做任何改动。对包的改动(例如，重命名或者删除一些文

件)可能会打乱之前的对齐关系以及后面文件的对齐关系。此外,向已经对齐好的压缩包中添加任何文件都会导致整个压缩包的对齐关系被破坏掉。

zipalign 的用法如下:

```
zipalign [-f] [-v] alignment infile.apk outfile.apk
```

这种用法会将 `infile.apk` 对齐后保存为 `outfile.apk`。

zipalign 还有下面这种用法:

```
zipalign -c -v alignment existing.apk
```

这种用法会将 `existing.apk` 进行对齐。

`alignment` 是一个整型值,它定义了字节对齐边界。这个值必须一直为 4(即 32 位对齐),其他值则不会有任何效果。

表 B-6 描述了各种选项。

表 B-6 支持的选项

选 项	描 述
-c	确认对齐一个给定的文件
-f	覆盖已经存在的 <code>outfile.apk</code> 文件
-v	打印详细输出信息

B.2 平台工具

SDK 平台工具是开发应用程序时要用到的一些依赖于平台的工具。这些工具支持最新 Android 平台的特性,通常只有在有新平台可用时才会更新。这些工具总是会向后兼容,前提是必须确保在安装新平台时这些工具的版本也随之更新了。

B.2.1 aapt

aapt (Android Asset Packaging Tool, Android 资源打包工具)可以用来查看、创建和更新与 Zip 格式相兼容的压缩包(ZIP、JAR、APK)。它还可以将资源编译为二进制文件。在编译资源时,会创建一个 `R.java` 文件,这样你就可以在 Java 代码中引用资源了。

虽然你可能没有经常直接用到 aapt,但构建脚本和 IDE 插件会通过这个工具将 Android 应用程序的资源打包为 APK 文件。

aapt 有很多的使用方法,下面是用法是最简单的:

```
aapt l[list] [-v] [-a] file.{zip,jar,apk}
```

这种用法会列出与 Zip 格式相兼容的压缩包中所有内容。选项 `-v` 表示详细的输出,选项 `-a` 会在列举压缩包内容时打印 Android 特有的数据(资源, manifest)。

B.2.2 adb

adb 可以用来与模拟器或者 Android 连接设备进行通信。它的结构是客户端-服务器端的模式，包含 3 个组件：

- 客户端，运行在你的开发机器上。可以通过在命令行解释器中输入 adb 命令启动一个客户端。其他 Android 工具(例如，ADT 插件和 ddms)也会创建 adb 客户端。
- 服务器端，会在你的开发机器上运行一个后台进程。服务器端会管理客户端与 ADB 守护进程(运行在模拟器或者设备上)之间的通信。
- 守护进程，运行在模拟器或设备上的一个后台进程。

adb 的用法如下：

```
adb [-d|-e|-s serialNumber|-p product_name_or_path] command
```

以上用法表明 adb 是用来提交命令的。表 B-7 描述了处理一个命令时的各种选项。

表 B-7 支持的选项

选 项	描 述
-d	只向唯一连接的 USB 设备发送命令。如果存在多个 USB 连接设备会返回错误
-e	只向运行着的模拟器发送命令。如果存在多个模拟器会返回错误
-p	指定一个简单的产品名称(例如，sooner)或者产品输出目录的相对/绝对路径(例如，out/target/product/sooner)。如果 -p 没有指定的话，就会使用 ANDROID_PRODUCT_OUT 环境变量,它必须是一个绝对路径
-s	向一个指定的模拟器/设备发送命令，模拟器/设备是通过 adb 分配的序列号标识的(例如，emulator-5556)

下面的列表中列出了众多可用命令中的一部分：

- adb devices 会打印所有已连接的模拟器/设备的列表。
- adb help 会打印所有支持命令的列表。
- adb install path-to-APK 会将一个 Android 应用程序(这里需要 APK 文件的全路径)安装到模拟器/设备的数据文件中。
- adb logcat 用来查看设备的日志。
- adb pull remote local 会将模拟器/设备上的特定文件复制到你的开发机器上。
- adb push local remote 会将你的开发机器上的特定文件复制到模拟器/设备上。
- adb shell 会启动一个目标模拟器/设备的远程命令行解释器。
- adb shell shellcommand 会在目标模拟器/设备上执行一个 shellcommand，然后退出远程命令解释器。
- adb uninstall package 会将 APK 从设备中卸载掉。package 即为应用程序的全路径 Java 包名。
- adb version 会打印 adb 的版本号。

注意:

可以查看 Google 的“Android Debug Bridge”页面(<http://developer.android.com/tools/help/adb.html>)来进一步了解如何使用这个工具。

B.2.3 aidl

aidl (Android Interface Definition Language, Android 接口定义语言)和你可能用过的其他 IDL 语言类似。它可以让你定义客户端和服务端端的编程接口,从而通过 IPC(interprocess communication, 进程间通信)实现彼此之间的通信。在 Android 中,一个进程通常是不能访问另一个进程的内存的。所以,需要将对象解析为操作系统可以理解的原始数据,以达到可以越界访问的目的。而对象解析的过程非常枯燥晦涩,所以 Android 使用了 aidl 处理这个工作。

aidl 的用法如下:

```
aidl OPTIONS INPUT [OUTPUT]
aidl --preprocess OUTPUT INPUT...
```

OPTIONS 是以下选项的组合:

- -a: 根据输入文件的名字在输出文件后面生成一个依赖文件。
- -b: 尝试编译 `parcelable` 类型的文件时会失败。
- -d FILE: 生成依赖清单文件。
- -I DIR: 在该 DIR 中搜索相应的 aidl 文件。
- -o DIRECTORY: 指定生成文件的输出根目录。
- -p FILE: 编译 aidl 时以预处理文件(FILE, 它通过 `--preprocess` 创建)为参数去生成.java。

INPUT 是一个 aidl 接口文件(输入)。OUTPUT 表示生成的接口文件。如果没有 OUTPUT 参数 `bing` 没有使用 `-o` 选项,生成的文件会使用输入文件名,即将输入文件的.aidl 扩展名换成.java 扩展名。如果使用了 `-o` 选项,生成的文件就会放在指定的根目录下的包名层命名的目录下。

注意:

可以查看 Google 的“Android Interface Definition Language (AIDL)”页面(<http://developer.android.com/guide/components/aidl.html>)来进一步了解如何使用这个工具。

B.2.4 dexdump

Dexdump 用来输出 Dalvik 可执行文件的内容(通常 class.dex 文件会保存在 APK 文件中)。

dexdump 的用法如下:

```
dexdump [-c] [-d] [-f] [-h] [-i] [-l layout] [-m] [-t tempfile] dexfile...
```

表 B-8 描述了各种选项:

表 B-8 支持的选项

选 项	描 述
-c	检查校验码并退出
-d	反编译代码片段
-f	显示文件头中的概要信息
-h	显示文件头的详细信息
-i	忽略校验码失败
-l <i>layout</i>	指定输出的布局, “plain” 或者 “xml”
-m	反编译出寄存器映射表(没有别的东西)
-t	指定临时文件的名称(默认为/sdcard/dex temp-*)

注意:

可以查看 <http://mylifewithandroid.blogspot.ca/2009/01/disassembling-dexfiles.html> 页面中的“Disassembling DEX files”来了解如何使用 dexdump 来反编译 classes.dex 中的内容。

B.2.5 dx

dx 会将编译好的.class 文件转换为可执行的.dex 文件。它有很多种用法, 最简单的一种如下:

```
dx --help
```

得到使用 dx 的详细帮助信息。

B.2.6 fastboot

fastboot 用来通过 USB 从一个主机那里更新 Android 设备的 flash 文件系统。

fastboot 的用法如下:

```
fastboot [option] command
```

这种用法建议在执行命令前首先指定各种选项。表 B-9 描述了各种选项:

表 B-9 支持的选项

Option	Description
-b <i>baseAddr</i>	指定一个自定义内核基址(<i>baseAddr</i>)
-c <i>commandline</i>	覆盖内核命令行(<i>commandline</i>)
-i <i>vendorID</i>	指定一个自定义 USB 厂商 ID(<i>vendorID</i>)
-n <i>pageSize</i>	指定闪存页的大小(<i>pageSize</i>), 默认为 2048 字节
-p <i>product</i>	指定产品名称(<i>product</i>)

(续表)

Option	Description
<i>-s serialNumber</i>	指定设备序列号(serialNumber)
<i>-l layout</i>	指定输出布局, “plain” 或者 “xml”
<i>-w</i>	擦除(删除)用户数据和缓存

表B-10描述了各种命令。

表 B-10 支持的命令

命 令	描 述
<code>boot kernel [ramdisk]</code>	下载并且开始内核引导
<code>continue</code>	继续自动引导
<code>devices</code>	列出所有连接的设备
<code>erase partition</code>	删除闪存分区
<code>flash partition [filename]</code>	将一个文件写入到闪存分区
<code>flash:raw boot kernel[ramdisk]</code>	创建引导镜像并加载到闪存
<code>flashall</code>	将引导文件+恢复文件+系统文件加载到闪存
<code>format partition</code>	格式化一个闪存分区
<code>getvar variable</code>	显示一个引导加载器 <i>variable</i>
<code>help</code>	显示帮助信息
<code>reboot</code>	正常情况下会重启设备
<code>reboot-bootloader</code>	重启设备到引导加载器
<code>update filename</code>	通过 <code>update.zip</code> 刷新设备

B.2.7 llvm-rs-cc

llvm-rs-cc 是一个 Renderscript(参见第 8 章)源代码编译器。

Renderscript 的用法如下:

```
llvm-rs-cc [options] inputs
```

这种用法建议在指定 *inputs* 之前首先指定各种选项。

表 B-11 描述了各种选项。

表 B-11 支持的选项

选 项	描 述
<code>-additional-dep-target value</code>	在依赖输出中显示其他目标
<code>-allow-rs-prefix</code>	允许用户自定义的函数使用 <code>rs</code> 前缀

(续表)

选 项	描 述
-bitcode-storage <i>value</i>	<i>value</i> 的值应该为 ar 或者 jc
-emit-asm	生成目标汇编文件
-emit-bc	构建抽象语法树(ASTs)然后转换为 LLVM, 最后生成.bc 文件
-emit-llvm	构建 ASTs, 然后转换为 LLVM, 最后生成.ll 文件
-emit-nothing	构建 ASTs, 然后转换为 LLVM, 最后什么也不生成
-g	发送 LLVM 调试元数据
-help	打印帮助信息
-I <i>directory</i>	添加目录来包含搜索路径
-java-reflection-package-name <i>value</i>	指定反射的 Java 文件的包名
-java-reflection-path-based <i>directory</i>	指定输出反射的 Java 文件的根目录
-O <i>optimization-level</i>	<i>optimization-level</i> 的值可以为 0 到 3(默认值)
-o <i>directory</i>	指定输出目录
-output-dep-dir <i>directory</i>	为依赖的输出指定输出目录
-reflect-c++	反射 C++类
-target-api <i>value</i>	指定目标 API Level(例如 14)
-version	打印汇编器的版本
-w	禁止所有警告

附录 C

应用程序设计指南

本书的重点是如何利用各种 Android 技术开发应用程序。不过，要成为成功的 Android 开发者，仅仅把应用程序开发出来是绝对不够的，还必须知道如何让应用程序只呈现给使用兼容设备的用户，如何让应用程序运行得更好，如何快速响应用户以及如何与其他应用程序正确地交互。本附录会介绍一些必要的设计知识，让你的应用程序变得更美好。

C.1 设计经过滤的应用程序

C.1.1 问题

在将应用程序发布到 Google Play 时，只想让应用程序兼容的设备看到应用程序。这需要 Google Play 过滤应用程序，确保使用不兼容设备的用户无法下载。

C.1.2 解决方案

Android 可以在各种设备上运行，这给开发者带来的巨大的潜在市场。不过，并不是所有的设备都具备同样的功能(例如，有些设备有摄像头，有些则没有)，所以一些应用程序在某些设备上可能是无法运行的。

为了解决这个问题，用户在 Android 设备上访问 Google Play 时可以使用 Google 提供的各种市场过滤器。如果应用程序不满足某个过滤器的要求，就不会显示给该用户。表 C-1 中列出了可以通过应用的 manifest 文件中特定元素启用的 3 个市场过滤器。

表 C-1 基于 manifest 元素的市场过滤器

过滤器名称	Manifest 元素	过滤器的工作原理
最低框架版本 (minSdkVersion)	<uses-sdk>	<p>应用程序所需的最低的 API level。不支持该 API 的设备无法运行该应用程序。</p> <p>APILevel 是用整数表示的。例如，整数 9 表示 Android 2.3(API Level 9)。查看 http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels 可以获得完整的 API Level 列表和相关平台版本号。</p> <p>例如：<uses-sdkandroid:minSdkVersion="9"/>是告诉 Google Play，应用程序只支持 Android 2.3 以上的版本。</p> <p>如果没有声明该属性，Google Play 就会取其默认值 1，表示应用程序能兼容所有的 Android 版本。</p>
设备功能 (name)	<uses-feature>	<p>应用程序所需的设备功能。这个元素在 Android 2.0(API Level 5)中引入。</p> <p>例如：<uses-featureandroid:name="android.hardware.sensor.compass"/>告诉 Google Play，设备必须要有罗盘才能运行该应用程序。</p> <p>抽象类 android.content.pm.PackageManager 定义了"android.hardware.sensor.compass"和其他功能 ID。</p>
Screen Size(屏幕尺寸)	<supports-screens>	<p>通过<supports-screens>元素(在 API Level 3 及以前版本并没有定义该元素)设置应用程序所支持的屏幕尺寸，在应用程序发布到 Google Play 之后，Google Play 会根据用户设备的屏幕尺寸判断是否向其显示该应用程序。</p> <p>例如：<supports-screensandroid:smallScreens="false"/>告诉 Google Play 应用程序不能在 QVGA(240X320 像素)屏幕上运行。</p> <p>Google Play 通常假设设备可以将较小的布局匹配到大屏幕上，但不能将较大的布局匹配到小屏幕上。因此，如果应用程序宣称能支持正常(normal)尺寸的屏幕，那么政策尺寸和大尺寸屏幕的设备都能看到该应用程序，但对于小尺寸屏幕的设备，该应用就会被过滤。</p>

Google Play 还可以基于高级 manifest 元素来过滤应用程序。例如，出现<supports-gl-texture>元素时，除非应用程序支持的一个或者多个 GL 纹理压缩格式可以被某个设备支持，否则 Google Play 会阻止该应用程序下载到该设备上。

最后，Google Play 还能通过应用程序的其他特征(例如设备用户所在的国家)来判断是否要显示或隐藏某个应用程序。表 C-2 列出了 3 个市场过滤器，可以根据各自特征启用。

表 C-2 基于应用程序特征的市场过滤器

过滤器名称	过滤器的工作原理
发布状态	只有已发布的应用程序才会显示在 Google Play 的搜索结果中。即使应用程序没有发布，只要用户能够在他们的已购买、已安装或最近删除的应用程序的 Downloads 区看到该应用程序，那就能安装该应用程序。如果应用程序被下架，那用户就算看到它，也不能重装或更新。
价格状态	并不是所有的用户都能看到收费应用程序。要显示收费应用程序，设备必须要有 SIM 卡，并运行 Android 1.1 或更高版本，而且必须位于已开通收费应用程序的国家(根据 SIM 卡的运营商来判断)。
国家或运营商	<p>在上传应用程序到 Google Play 时，可以选择所面向的国家。应用程序只会显示在你所选择国家(运营商)的用户设备上。</p> <ul style="list-style-type: none"> 设备的运营商决定了它所属的国家。如果不能判断运营商，Google Play 会根据 IP 地址来判断国家。 运营商是根据设备的 SIM 卡(GSM 设备)判断，而不是当前漫游的运营商。

注意：

想要了解更多过滤器的信息，可以查看 Google 的“Filters on Google Play”文档(<http://developer.android.com/guide/google/play/filters.html>)。

C.2 设计高性能的应用程序

C.2.1 问题

应用程序应该能流畅运行，尤其是在内存有限的设备上。此外，良好的性能还能降低耗电量。你需要知道如何设计高性能的应用程序。

C.2.2 解决方案

Android 设备是多种多样的。有些设备的处理器比较快，有些设备的内存比较大，而有些设备有 Just-In-Time (JIT)编译器，而有些设备则没有这项技术，不能将字节码指令在运行时转换成原生代码来提高代码执行速度。下面列出了在编写应用程序时应该考虑的与应用程序性能有关的注意事项：

- 仔细优化代码：尽可能优化应用程序的架构，使其在代码优化之前也能确保应用程序的基本性能。在应用程序能正确运行后，可以在各种设备上评测其性能，找到拖慢应用程序速度的瓶颈。记住，模拟器可能会让你对应用程序的性能有错误的印象。例如，模拟器的网速就是开发平台的网速，通常这要比真正 Android 设备的网速快不少。

- 尽可能少创建对象：创建对象会影响性能，尤其是在执行垃圾收集时。应该尽可能重用已有的对象，减少垃圾收集对应用程序性能带来的影响。例如，使用 `java.lang.StringBuilder`(或在可能有多个线程访问对象时使用 `java.lang.StringBuffer`) 对象构建字符串，而不要在循环中使用字符串连接操作，后者会创建大量冗余的 `String` 对象。
- 减少浮点运算：在 Android 设备上，浮点运算要比整数运算慢两倍。例如，没有浮点计算单元和 JIT 的第一代设备。此外，有些设备没有整数除法的硬件指令，这意味着整数除法在这些设备上是通过软件实现的，这尤其会给散列表的性能带来负面影响。
- 使用 `System.arraycopy()` 复制：在带 JIT 的 Nexus One 上，`java.lang.System` 的 `static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)` 方法要比你自己写的循环快 9 倍。
- 使用扩充版的循环语法：通常，在没有 JIT 的设备上，扩充版的循环(比如 `for (String s: strings) {}`)要比普通版的循环(比如 `for (int i = 0; i < strings.length; i++)`)快；在有 JIT 的设备上，前者并不会比后者慢。但在处理 `java.util.ArrayList` 时，扩充版的循环会比普通的循环慢一些，所以处理 `java.util.ArrayList` 时应该用普通的循环。

注意：

由于 Google 的性能文档中说枚举是依赖设备的，故本书的上一版建议避免使用枚举。枚举会增加 `.dex` 文件的大小并影响性能。例如，`public enum Directions { UP, DOWN, LEFT, RIGHT }` 与一个类中定义 4 个 `public static final` 的整型相比，会使 `.dex` 增大几百个字节。现在，由于枚举已经不是什么大问题了，故 Google 收回了“避免使用枚举”的建议。想要知道原因，可以查看 <http://stackoverflow.com/questions/5143256/why-was-avoid-enums-where-you-only-need-ints-removed-from-androids-performan>。

在选择算法和数据结构时也要小心。例如，线性搜索算法(从头到尾逐个比较各个元素与要搜索的值)平均要核对一半的元素，而二分查找算法迭代分支技术寻找要搜索的值，比较的次数会很少。例如，搜索 40 亿个元素，线性搜索平均要做 20 亿比较，而二分查找最多只需要比较 32 次。

C.3 设计快速响应的应用程序

C.3.1 问题

应用程序不能及时响应用户的操作，或者是有停顿、有时会出现 ANR(Application Not Responding)对话框(如图 C-1)。遇到这种情况，用户要么选择终止应用程序(很有可能会卸载)，要么耐着性子等待应用程序作出响应。

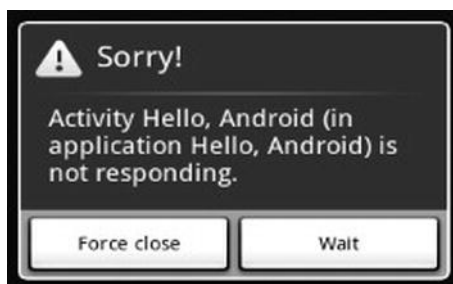


图 C-1 可怕的 ANR 对话框可能会导致用户卸载应用程序

你想知道如何设计出能快速作出响应的应用程序，以便避免出现这种对话框(这会给应用程序的口碑带来很糟糕的负面影响)。

C.3.2 解决方案

当应用程序无法响应用户的输入时，Android 就会显示 ANR 对话框。例如，应用程序因为 I/O 操作(通常是网络访问)阻塞，导致主应用程序无法处理收到用户的输入事件。在一定时间后，Android 就会认为应用程序卡死，然后就显示该对话框，让用户选择是否关闭该应用程序。

注意：

Activity 管理器和窗口管理器(见第 1 章，图 1-1)会监控应用程序的响应。当它们发现应用程序无法在 5 秒内响应输入事件(例如按键或者触摸屏幕)，或是 Broadcast Receiver 在 10 秒内没有完成，它们就会认为应用程序已卡死，并显示 ANR 对话框。

同理，当应用程序花费了大量时间在内存中构建数据结构，或是执行大量的计算时(例如在象棋或其他游戏中计算下一步操作)，Android 会认为应用程序已经挂起。所以，一定要尽可能利用范例 C-2 中介绍的各种技术来提高应用程序中各种计算的效率。

遇到这种情况，应用程序应该另外创建一个线程来完成这些工作。对于 Activity，更是如此。要尽可能在诸如 onCreate(Bundle) 和 onResume()之类的关键生命周期回调方法中处理这类工作。这样，主线程(驱动用户界面事件循环的线程)可以保持运行，Android 也就不会认为应用程序已经卡死了。

提示：

可以使用进度条让用户知道漫长操作的进度。

C.4 设计无缝衔接的应用程序

C.4.1 问题

要设计出能与其他应用程序正确互动的应用程序。具体来说，就是要知道应用程序应该避免哪些有可能给用户带来麻烦的问题(以及如何面对被卸载的可能)。

C.4.2 解决方案

应用程序必须与其他应用程序和平共处，这样其他应用程序才不会在用户与某个 Activity 交互时忽然弹出一个讨厌的对话框。而且，你肯定不希望应用程序的 Activity 在暂停时丢失状态信息，让用户回到该 Activity 时郁闷地发现之前输入的数据已经无影无踪。换言之，应用程序需要与其他应用程序配合，才能为用户提供良好的用户体验。

- 不要丢失数据：因为 Android 是一个移动平台，其他 Activity 随时都可能打断你的应用程序(比如接到电话就会启动 Phone 应用)。在遇到这种情况时，就会调用你的 Activity 的 `void onSaveInstanceState(Bundle outState)` 和 `onPause()` 回调方法，然后应用程序就有可能被关闭。如果用户此时正在编辑数据，而你没用 `onSaveInstanceState(Bundle)` 保存的数据可以在 `onCreate(Bundle)` 或 `void onRestoreInstanceState(Bundle savedInstanceState)` 方法中恢复。
- 不要提供原始数据：不要提供原始数据，因为其他应用程序不一定能理解你的数据格式。另外，如果你修改了格式，其他应用程序也会出问题，需要更新才能适应新的变化。你应该创建一个 `ContentProvider` 实例，用精心设计的 API 将数据提供给其他应用程序。
- 不要打断用户：如果用户与 Activity 的互动被一个突然弹出的对话框(可能是来自后台服务或是某个 `startActivity(Intent)` 方法的调用)打断，肯定会不高兴。更好的方式应该用 `android.app.NotificationManager` 类发送一个消息通知用户。消息会出现在状态栏中，用户可以在方便时再去查看消息的内容。
- 用线程执行长时间操作：应该把需要执行长时间运算，或是包含耗时的 Activity 的组件放到另一个线程中。这可以避免出现 ANR 对话框，降低用户卸载应用的概率。
- 不要在一个用户界面堆砌太多东西：如果应用程序的界面太复杂了，就应该分解成多个 Activity。这样，用户才不会因为页面过于复杂而感到无从下手。此外，这还能提高代码的可维护性，在 Android 的 Activity 栈模型中也会有更好的表现。
- 扩展系统主题：涉及用户界面的外观时，最好能够完美地统一起来。当用户界面与用户所期待的界面差异很大时，用户会很敏感。设计 UI 时，应该尽量避免与常规的 UI 差异太大。相反，可以使用主题，参见 <http://developer.android.com/guide/topics/ui/themes.html>。你可以覆写或扩展这些主题，但至少和其他应用程序的 UI 基本是一致的。
- 设计使用多种分辨率的用户界面：不同的 Android 设备的屏幕通常分辨率也不同。有些设备甚至可以随时改变分辨率，例如切换到横屏状态。所以，确保应用程序的布局和可绘制资源能在各种不同的屏幕上正确显示时很重要的。为常见的分辨率制作不同版本的图片，设计能适应不同尺寸的布局来解决这个问题(例如，避免使用硬编码的位置，尽量用相对布局)。尽可能这么做，而系统会解决其他问题；最终应用程序在任何设备上看上去都会很棒。
- 假设网络很慢：Android 设备可以连接各种网络，有些网络快，有些则要慢一些。当然，常见的最慢的网络是 GPRS(没有 3G 数据服务的 GSM 网络)。设置 3G 设备大部分时候也是运行在非 3G 网络上，所以网速在一定时间内依然是个很现实的问题。

题。因此，在编写应用程序时一定要减少网络访问，节约带宽。不要认为网络很快，要准备适应缓慢的网络。这样当用户在告诉网络上使用应用程序时，就会获得更好的用户体验。

- 不要假设有触摸屏或键盘：Android 支持各种各样的输入设备：有些 Android 有 QWERTY 全键盘，而有些设备则是 40-键、12-键或是其他类型的键盘。同样，有些设备上有触摸屏，有些则没有。在设计应用程序时要考虑到这些差异。如果不想把应用程序限定在某一种设备上，就不要假设具体的键盘布局。
- 节约用电：移动设备是电池供电的，因此减少电量的消耗很重要。电池的两个消耗大户分别是处理器和无线电，所以应用程序应该要尽量少用处理器，少访问网络。减少应用程序使用处理器的时间实际上就是要编写高效的代码。降低无线电的耗电量，实际上就是要小心处理错误情况，只下载必要的的数据。例如，在访问网络失败时，不要持续不断地反复尝试。马上重试肯定会失败，因为用户还没有做出任何反馈，这么做只会浪费电量。记住，用户会发现耗电的应用程序，然后把它删掉。

C.5 设计安全的应用程序

C.5.1 问题

你要熟悉 Android 安全方面的最佳实践，以确保你的应用程序可以利用 Android 的安全特性，还能减少无意中引入安全问题的可能性，这些安全问题可能会影响你的应用程序。

C.5.2 解决方案

Google 的“Designing for Security”文档(<http://developer.android.com/guide/practices/security.html>)展示了很多的安全特性，可以帮助开发者构建安全的应用程序。例如，Android 提供了一个加密的文件系统，可以在手机丢失或者被盗时保护数据的安全。

这个文档展示了大量的最佳实践，致力于应用程序和用户各种安全方面的引导。例如，虽然可以使用 Linux 网络 socket 或共享文件来执行进程间通信(IPC)，但最好使用 Android 的 IPC 机制(Intent、binder、service 和 receiver)，它们会验证要连接的应用程序的身份并设置相应的 IPC 安全策略。

附录 D

Univerter 的结构

第 1 章介绍了 Univerter，它是一个执行单位换算的应用程序。由于第 1 章篇幅有限，并没有详细了解该应用程序的结构，所以附录 D 会对 Univerter 的源代码、资源文件和 manifest 文件做进一步的探究。

D.1 源代码

Univerter 的源代码由 4 个文件组成：Category.java、Conversion.java、Converter.java 和 Univerter.java。每个文件开始都有一个包信息语句，表示每个文件的类都位于 ca.tutortutor.univerter 包中。应用程序的每个引用类型必须属于某一个包。

注意：

除了来自于其他应用程序的可复用的引用类型外，所有引用类型都必须属于同一个包。其他应用程序的引用类型位于另一个包中。

D.1.1 Converter 接口

程序清单 D-1 展示了 Converter.java 的内容。

程序清单 D-1 Converter 接口中定义了一个执行换算的方法

```
package ca.tutortutor.univerter;

import android.content.Context;

interface Converter
{
    double convert(Context ctx, double value);
}
```

程序清单 D-1 声明了一个 `Converter` 接口，该接口中声明了一个 `doubleconvert(Context ctx, double value)` 方法头来处理单位换算。传入了 `ctx` 参数可以在需要时访问一些字符串资源来提示出错信息。传入的 `value` 即为需要换算的值。该方法的返回值为换算后的结果。

注意：

可以通过 `android.content.Context` 类中声明的 `String getString(int resId)` 方法访问字符串资源。

不同的换算种类传入的参数可能是无效的。例如，摄氏度在换算为华氏温度时，-1000 就是无效的，因为它小于了绝对零度的值(-273.15 摄氏度)。本例中，`convert(Context, double)` 方法应该抛出一个 `java.lang.IllegalArgumentException` 异常，该异常由该方法的调用者捕获并处理。

D.1.2 Conversion 类

程序清单 D-2 展示了 `Conversion.java` 的内容。

程序清单 D-2 `Conversion` 类描述了一个简单的换算关系

```
package ca.tutortutor.univerter;

import android.content.Context;

class Conversion
{
    private int nameID;
    private Converter converter;
    private boolean canBeNegative;
    Conversion(int nameID, final double multiplier)
    {
        this(nameID,
            new Converter()
            {
                @Override
                public double convert(Context ctx, double value)
                {
                    return value*multiplier;
                }
            },
            false);
    }

    Conversion(int nameID, Converter converter, boolean canBeNegative)
    {
        this.nameID = nameID;
        this.converter = converter;
        this.canBeNegative = canBeNegative;
    }
}
```

```

    boolean canBeNegative()
    {
        return canBeNegative;
    }

    Converter getConverter()
    {
        return converter;
    }

    String getName(Context ctx)
    {
        return ctx.getString(nameID);
    }
}

```

程序清单 D-2 声明了一个 `Conversion` 类，该类有三个变量：`name`、`converter` 和 `canBeNegative`，`canBeNegative` 用来标识传入 `convert(Context, double)` 方法的值是否可以为负值(负值的摄氏度有意义，但负值的公顷也有意义？)，`Univerter` 用这个 `canBeNegative` 标志启用或者禁用+/-按钮。

`Conversion` 声明了两个构造方法：

- `Conversion(int nameID, double multiplier)`
- `Conversion(int nameID, Converter converter, Boolean canBeNegative)`

每个构造方法都声明了一个 `nameID` 参数，表示命名换算所需的字符串的整型 ID。

第一个构造方法还声明了一个 `multiplier` 参数，将收到的 `multiplier` 参数进行 `value * multiplier` 形式的转换。该构造方法还实现了 `Converter` 接口来执行换算，并将接口的实例传递给第二个构造方法。最后将第二个构造方法中的 `canBeNegative` 参数置为 `false`。

注意：

第一个构造方法传递给 `canBeNegative` 的值为 `false`，这是因为非温度换算(第一个构造方法创建的)中负值是没有意义的。

第二个构造方法也声明了一个 `converter` 参数使用一个非默认的 `Converter` 实现处理更加复杂的换算，例如摄氏度换算为华氏温度。同样，它声明了一个 `canBeNegative` 参数来指定换算是否可以使用负值。

最后，`Conversion` 类声明了 `String getName(Context)`、`Converter getConverter()` 和 `boolean canBeNegative()` 方法来返回相应的 `name`、`converter` 和 `canBeNegative`(标识要换算的值是否可以为负值)。`getName(Context)` 方法中的参数为 `Context` 类型，用它可以查找 `nameID`(使用 `getString(int)`)相应的字符串资源并返回该字符串。

D.1.3 Category 类

程序清单 D-3 展示了 `Category.java` 的内容。

程序清单 D-3 Category 类描述了一个 Conversions 数组

```

package ca.tutortutor.univerter;

import android.content.Context;

class Category
{
    private int nameID;
    private Conversion[] conversions;
    private String[] conversionNames;
    Category(int nameID, Conversion[] conversions)
    {
        this.nameID = nameID;
        this.conversions = conversions;
    }

    Conversion getConversion(int index)
    {
        return conversions[index];
    }

    String[] getConversionNames(Context ctx)
    {
        if (conversionNames == null)
        {
            conversionNames = new String[conversions.length];
            for (int i = 0; i < conversionNames.length; i++)
                conversionNames[i] = conversions[i].getName(ctx);
        }
        return conversionNames;
    }

    String getName(Context ctx)
    {
        return ctx.getString(nameID);
    }
}

```

程序清单 D-3 声明了 Category 类，它通过一个 name 和 Conversion 类型的数组描述了一种换算种类。它的 Category(int nameID, Conversion[] conversions)构造方法会通过一个整型字符串资源 nameID 和一个 Conversion 数组参数初始化一个 Category 实例。两个参数都会被保存在类的同名变量中。

Category 类还声明了 String getName(Context)和 Conversion getConversion(int index)方法来返回相应的 name 和 Conversion 数组中指定索引值的实例。getConversion(int)方法不用检查参数的有效性，这是因为 Univerter 不会传一个无效的索引值给它(如果传入无效的索引值，会抛出 java.lang.ArrayIndexOutOfBoundsException)。

最后，Category 声明了 String[] getConversionNames(Context)方法，Univerter 可以调用

它获得当前换算种类(显示在一个对话框中)中的所有换算名称的数组。这个方法经过了优化,可以避免不必要的对象创建,只有第一次调用时才会创建对象(之后的调用都不会创建)。

D.1.4 Univerter 类

程序清单 D-4 展示了 Univerter.java 的主要内容。

程序清单 D-4 Univerter 描述了应用程序唯一的 Activity

```
package ca.tutortutor.univerter;

import android.app.Activity;
import android.app.AlertDialog;

import android.content.Context;
import android.content.DialogInterface;

import android.graphics.Color;
import android.graphics.PorterDuff.Mode;

import android.os.Bundle;

import android.text.Html;

import android.text.method.LinkMovementMethod;

import android.view.Gravity;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;

import android.webkit.WebView;

import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.ListAdapter;
import android.widget.TextView;
import android.widget.Toast;

public class Univerter extends Activity
{
    //在这声明变量和方法
}
```

程序清单 D-4 声明了 Univerter 类，它继承自 android.app.Activity。Univerter 中声明了很多的变量和方法。

D.1.5 Univerter 中的变量

Univerter 首先声明了一个 categories 数组，然后是一个静态的初始化过程，即把这个变量初始化为一个 Category 实例数组。

```
private static Category[] categories;
static
{
    categories = new Category[]
    {
        new Category
            (R.string.cat_angle,
            new Conversion[]
            {
                new Conversion
                    (R.string.cat_angle_circles_to_deg,
                    360),
                // ...
                new Conversion
                    (R.string.cat_angle_rad_to_grad,
                    63.661977237)
            }
        ),
        // ...
        new Category
            (R.string.cat_temp,
            new Conversion[]
            {
                new Conversion
                    (R.string.cat_temp_celsius_to_fahrenheit,
                    new Converter()
                    {
                        @Override
                        public double convert(Context ctx,
                                             double value)
                        {
                            {
                                if (value < -273.15)
                                {
                                    String s;
                                    s = ctx.
                                        getString(R.string.
                                            error_less_than_abs0);
                                    thrownew
                                        IllegalArgumentException(s);
                                }
                                return value*9.0/5.0+32;
                            }
                        }
                    }
                )
            }
        )
    }
}
```

```

        },
        true),
    // ...
    new Conversion
    (R.string.cat_temp_kelvin_to_celsius,
    new Converter()
    {
        @Override
        public double convert(Context ctx,
                             double value)
        {
            return value-273.15;
        }
    },
    false)
    )),
    // ...
    new Category
    (R.string.cat_weightmass,
    new Conversion[]
    {
        new Conversion
        (R.string.cat_weightmass_ct_to_lb,
        0.000440925),
        // ...
        new Conversion
        (R.string.cat_weightmass_t_to_lb,
        2204.622621849)
    })
    );
};
}

```

每个 Category 都由种类名称字符串资源 ID 和一个 Conversion 实例的数组组成。每个 Conversion 实例又由换算名称字符串资源 ID、multiplier 或一个 Converter 实例组成，Converter 实例会覆写 convert(Context, double)方法来实现换算关系的转换。

当 Univerter 加载到内存后就会执行静态初始化块。由于要创建很多的对象，最好只在类加载时创建一次而不是在每个 Univerter 实例化时都去创建，不过这样做会有一个问题，就是 categories 可能不是静态的。

在静态初始化块后面是其他变量的声明：

```

private String[] catNames;
private int curCat, curCon;

private StringBuilder buffer;
private int state;
private int nDigits;
private boolean isDecimal;
private boolean btnCvtClicked;
private Button btnPm;

```

```
private EditText etDisplay;
private DialogInterface.OnClickListener oclCat, oclCatClose;
private DialogInterface.OnClickListener oclCon, oclConClose;
private int choice;
private String helpText;
```

这些变量有以下作用：

- catNames 指向一个种类名称的数组，当用户按下 Univerter 的 CAT 按钮时会显示该数组的内容。
- curCat 表示当前种类的索引值。
- curCon 表示当前种类中当前换算的索引值。
- Buffer 保存了输入的数字。
- State 表示当前的状态，它有两个可选的值，用来控制数字的输入。
- nDigits 表示输入数字的位数。
- isDecimal 表示输入的是否为小数点。
- btnCvtClicked 表示 CVT 是否被按下。
- btnPm 表示 +/- 按钮，该按钮可以被启用或者禁用。
- etDisplay 用于显示输入数字或者换算结果的显示条。
- oclCat 是 CAT 按钮的单击监听器。按下时会显示一个所有种类名称的对话框列表。
- oclCatClose 是种类名称对话框中 Close 按钮的单击监听器。
- oclCon 是 CON 按钮的单击监听器。按下时会显示一个当前种类的所有换算名称的对话框列表。
- oclConClose 是换算名称对话框中 Close 按钮的单击监听器。
- choice 记录了种类名称或换算名称对话框中最近单击的选项。
- helpText 中保存了需要在 help 对话框上显示的 HTML 文本信息。

D.1.6 Univerter 的方法

Univerter 中声明了 15 个方法，分为 3 类：单击监听器方法、回调方法、辅助方法。

1. Univerter 的单击监听器方法

Univerter 变量声明之后就是 7 个以“do”开头的单击监听器方法。这些方法对应于 main.xml 布局中的各个 Button 元素的 onClick 属性(第 1 章讨论过)关联的单击事件。

下面的 void doCatClicked(View view) 对应 CAT 按钮的单击事件：

```
public void doCatClicked(View view)
{
    choice = curCat;
    if (oclCat == null)
        oclCat = new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
```

```

        {
            choice = which;
        }
    };

    if (oclCatClose == null)
        oclCatClose = new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                curCat = choice;
                curCon = 0;
                updateConversionTitle();
                reset();
                btnPm.setEnabled(categories[curCat].
                    getConversion(curCon).
                    canBeNegative());
            }
        };

    new AlertDialog.Builder(Univerter.this).
        setSingleChoiceItems(catNames, curCat, oclCat).
        setTitle(R.string.categories).
        setNeutralButton(R.string.btnClose, oclCatClose).
        show();
}

```

doCatClicked(View)首先会用 curCat 的值初始化 choice，这是因为 choice 表示的是当前选中的类型而种类名称对话框在关闭时把 choice 的值赋给了 curCat。

注意：

doCatClicked(View)方法中的 android.view.View 实例就是所单击的按钮。

接下来，会初始化 oclCat 和 oclCatClose 监听器，不过为了避免创建不必要的对象，只有在它们还没有初始化的时才会去实例化。前面的监听器负责跟踪种类名称的选择；后面的监听器负责对话框关闭的响应动作。

当选择了某个种类时，就会调用第一个监听器的 void onClick(DialogInterface dialog, int which) method 方法，这时会传入对话框的引用 dialog 和所选择项的索引值 which。监听器会将 which 的值赋给 choice，这样就可以得到当前的种类。

当对话框关闭时，会调用第二个监听器的 onClick(DialogInterface,int)方法。然后将 choice 的值赋给 curCat、将 curCon 的值置为 0(定位到新种类的第一个换算上)、通过 updateConversionTitle()方法显示新的换算界面、将显示条设置为“0.”、启用或者禁用+/-按钮。

doCatClicked(View)方法最后会创建并显示种类名称对话框。这里用到了 android.app.AlertDialog 和它内部定义的 Builder 类型，使用了一些 Builder 的方法，如下：

- AlertDialog.Builder setSingleChoiceItems(CharSequence[]items, int checkedItem, DialogInterface.OnClickListenerlistener)方法会在对话框中显示一个列表(catNames)，

列表的条目包括种类名称以及种类名称右侧的单选按钮风格的选中标志，单击条目时，选中标志会选中。此外，还会将选中的条目的索引值(*curCat*)传给 *checkedItem*，并注册一个监听器(*oclCat*)来响应条目的选择。

- `AlertDialog.Builder setTitle(int titleId)`会通过传给 *titleId* 的字符串资源 ID 设置对话框的标题。
- `AlertDialog.Builder setNeutralButton(int textId,DialogInterface.OnClickListener listener)`会初始化对话框中间的按钮(Close 按钮)。按钮上的文字是通过传给 *textId* 的字符串资源 ID(`R.string.btnClose`)获得的，该按钮按下时所使用的监听器是 *oclClose*。
- `AlertDialog` 的 `show()`方法会实例化 `AlertDialog` 并显示相应的对话框。

下面的 `void doClrClicked (View view)`对应 CLR 按钮的单击事件：

```
public void doClrClicked(View view)
{
    reset();
}
```

`doClrClicked(View)`会将显示条的信息置为“0”。

下面的 `void doConClicked(View view)`对应 CON 按钮的单击事件：

```
public void doConClicked(View view)
{
    choice = curCon;
    if (oclCon == null)
        oclCon = new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                choice = which;
            }
        };
    if (oclConClose == null)
        oclConClose = new DialogInterface.OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                curCon = choice;
                updateConversionTitle();
                reset();
                btnPm.setEnabled(categories[curCat].
                    getConversion(curCon).
                    canBeNegative());
            }
        };
    ListAdapter adapter;
    adapter = new ArrayAdapter<String>(Univerter.this,
                                     R.layout.list_row,
```

```

categories[curCat].
    getConversionNames(Univerter.this));
new AlertDialog.Builder(Univerter.this).
    setSingleChoiceItems(adapter, curCon, oclCon).
    setTitle(categories[curCat].getName(Univerter.this)).
    setNeutralButton(R.string.btnClose, oclConClose).
    show();
}

```

doConClicked(View)方法的内容和 doCatClicked(View)方法的内容非常类似，区别就是 doConClicked(View)用于选择一种新的当前换算名称。这个方法比较有趣的地方就是和 android.widget.ListAdapter 相关的代码。这个接口和其 android.widget.ArrayAdapter<T>实现类会用在 AlertDialog.Builder's AlertDialog.Builder.setSingleChoiceItems(ListAdapter adapter, int checkedItem, DialogInterface.OnClickListener listener)方法中来显示一个自定义的 View(这里用到了 R.layout.list_row 和 res/menu/univerter.xml)，这样换算名称会以小字号显示，看起来会好看一些。

ArrayAdapter(Context context, int textViewResourceId, T[] objects)构造方法需要当前的 Context(这里为 Univerter.this)、换算名称列表中每一行条目的布局资源 ID(R.layout.list_row)和要显示换算名称的数组(categories[curCat].getConversionNames(Univerter.this))。生成的 ListAdapter 会被传入到 setSingleChoiceItems(ListAdapter, int, DialogInterface.OnClickListener)方法中，从而将对话框与换算名称数组以及相应的布局关联起来。

下面的 void doCvtClicked(View view)对应 CVT 按钮的单击事件：

```

public void doCvtClicked(View view)
{
    try
    {
        double value = Double.parseDouble(buffer.length() == 0 ? "0" :
                                           buffer.toString());
        value = categories[curCat].getConversion(curCon).getConverter().
            convert(Univerter.this, value);
        if (Math.abs(value) > 1.0e+18)
            throw new NumberFormatException(getString(R.string.overflow));
        else
            if (value != 0.0 && Math.abs(value) < 1.0e-8)
                throw new NumberFormatException(getString(R.string.underflow));
        buffer.setLength(0);
        buffer.append(" "+value);
        etDisplay.setText(String.format("%.8f", value));
    }
    catch (IllegalArgumentException iae)
    {
        Toast t = Toast.makeText(Univerter.this, iae.getMessage(),
                                Toast.LENGTH_SHORT);
        t.setGravity(Gravity.CENTER_HORIZONTAL |
                    Gravity.CENTER_VERTICAL, 0, 0);
        t.show();
    }
}

```

```

    }
    btnCvtClicked = true;
}

```

doCvtClicked(View)首先会解析输入的值(该值以字符的形式保存在 buffer 中),得到当前种类中当前换算所对应的换算器(converter),最后使用换算器进行换算。

如果换算成功,doCvtClicked(View)会检测结果值是否溢出或下溢。在出现溢出或下溢的情况时会实例化 java.lang.NumberFormatException 异常类并抛出该异常。假设一切运行正常,buffer 的内容会被替换为结果值。这个值在格式化之后会显示在显示条上。

对于温度换算这种输入值可以为负值的情况,输入小于绝对 0 度的值会抛出一个 IllegalArgumentException 异常。由于 NumberFormatException 是 IllegalArgumentException 的子类,因此可以使用 IllegalArgumentException 处理这两种异常。发生任意一种异常时,会在 Activity 屏幕的中央短暂显示一个描述异常的 toast。

最后,btnCvtClicked 会被置为 true,这样在下次单击数字键时 buffer 会被置空,单击的数字就不会被添加到 buffer 当前内容的后面了。

下面的 doDigitClicked(View view)会响应数字按钮的单击操作:

```

public void doDigitClicked(View view)
{
    if (btnCvtClicked)
    {
        reset();
        btnCvtClicked = false;
    }
    buildNumber(((String) view.getTag()).charAt(0));
    if (buffer.length() == 0)
        etDisplay.setText("0.");
    else
        etDisplay.setText(buffer.toString()+
                           (buffer.indexOf(".") == -1 ? "." : ""));
}

```

doDigitClicked(View)首先会检查 btnCvtClicked 的值从而判断之前是否单击了 CVT 按钮。如果是的话,会通过 reset()方法清空 buffer 和显示条。

之后的 buildNumber(((String) view.getTag()).charAt(0))方法调用会在 buffer 的后面会添加刚刚单击按钮上的数字,该数字是通过按钮的 view.getTag()方法标识的,view.getTag()方法会返回 Button 元素中的 tag 属性(稍后显示)的值。

因为第一个字符不能为 0,所以如果用户开始就单击了 0,它是不会添加到 buffer 中的。这时,buffer 的长度就为 0 了,则会在显示条上显示“0.”。如果用户单击的不是 0,显示条上会显示 buffer 中的内容。

输入时,小数点通常都是出现在数字的右侧。然而,当输入的数值中包含小数点,这个小数点右侧的数字应该不会显示;此时,会检测到 buffer 中已经有小数点了。

下面的 void doDotClicked (View view)对应小数点按钮的单击事件:

```

public void doDotClicked(View view)
{
    if (btnCvtClicked)
    {
        reset();
        btnCvtClicked = false;
    }
    buildNumber('.');
}

```

doDotClicked(View)和 doDigitClicked(View)类似，如果之前单击了 CVT 按钮，也会清空 buffer。然后，除非小数点已经存在，否则会把小数点添加到 buffer 的内容中。

最后，下面的 void doPmClicked (View view)对应+/-按钮的单击事件：

```

public void doPmClicked(View view)
{
    buildNumber('-');
    if (state == 1)
        etDisplay.setText(buffer.toString()+
                           (buffer.indexOf(".")!=-1?"":"."));
}

```

doPmClicked(View)会在 buffer 中添加一个符号或者将符号从 buffer 中移除，然后输出 buffer 的内容，对于负号只有在 state 为 1 时才会去处理它(state 为 0 时，buffer 是空的)。

2. Univerter 的回调方法

Univerter 单击监听器方法后是 4 个以“one”开头的回调方法。这些方法对应于 Activity 的生命周期或者用户界面事件。

下面的 onCreate(Bundle savedInstanceState)方法在 Activity 创建时会被调用，从启动器中启动一个应用程序或者设备进行横竖屏切换时会创建 Activity。

```

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

    catNames = new String[categories.length];
    for (int i = 0; i < catNames.length; i++)
        catNames[i] = categories[i].getName(Univerter.this);

    if (savedInstanceState == null)
    {
        curCat = 0;
        curCon = 0;
        buffer = new StringBuilder();
        state = 0;
        nDigits = 0;
        isDecimal = false;
        btnCvtClicked = false;
    }
}

```

```

    }
    else
    {
        curCat = savedInstanceState.getInt("curCat");
        curCon = savedInstanceState.getInt("curCon");
        buffer = new StringBuilder(savedInstanceState.getString("buffer"));
        state = savedInstanceState.getInt("state");
        nDigits = savedInstanceState.getInt("nDigits");
        isDecimal = savedInstanceState.getBoolean("isDecimal");
        btnCvtClicked = savedInstanceState.getBoolean("btnCvtClicked");
    }

    boolean isLeftIconSupported =
        requestWindowFeature(Window.FEATURE_LEFT_ICON);
    setContentView(R.layout.main);
    if (isLeftIconSupported)
        setFeatureDrawableResource(Window.FEATURE_LEFT_ICON,
                                   R.drawable.ic_launcher);

    updateConversionTitle();

    etDisplay = (EditText) findViewById(R.id.display);

    int[] btnDigitIds =
    {
        R.id.btn7,
        R.id.btn8,
        R.id.btn9,
        R.id.btnClr,
        R.id.btn4,
        R.id.btn5,
        R.id.btn6,
        R.id.btnCat,
        R.id.btn1,
        R.id.btn2,
        R.id.btn3,
        R.id.btnCon,
        R.id.btn0,
        R.id.btnDot,
        R.id.btnPm,
        R.id.btnCvt
    };
    for (int i = 0; i < btnDigitIds.length; i++)
    {
        Button btn = (Button) findViewById(btnDigitIds[i]);
        if (btnDigitIds[i] == R.id.btnPm)
        {
            btnPm = btn;
            btnPm.setEnabled(categories[curCat].getConversion(curCon).
                             canBeNegative());
        }
    }

```

```

    }
    btn.setBackground().
        setColorFilter(Color.GRAY, Mode.MULTIPLY);
}

helpText = getString(R.string.help);
int colorHelpHiliteText = getResources().
    getColor(R.color.helpHiliteText)&0x00ffffff;
helpText = helpText.replaceAll("#helpHiliteText",
    "#" + toHexString(colorHelpHiliteText, 6));
int colorHelpText = getResources().getColor(R.color.helpText)&0x00ffffff;
helpText = helpText.replaceAll("#helpText",
    "#" + toHexString(colorHelpText, 6));
int colorLink = getResources().getColor(R.color.link)&0x00ffffff;
helpText = helpText.replaceAll("#link",
    "#" + toHexString(colorLink, 6));
}

```

`onCreate(Bundle)`中首先调用父类的 `onCreate(Bundle)`方法。然后创建了一个种类名称的数组(`catNames`)，如之前介绍过的，`doCatClicked(View)`方法在创建种类名称对话框时会用到这个数组。

接下来，`onCreate(Bundle)`会检查它的 `android.os.Bundle` 参数来判断该方法的调用时机，即是在启动器中启动 `Univerter` 还是在设备屏幕方向切换时启动的。在启动器中启动时，`onCreate(Bundle)`中的 `Bundle` 参数为 `null`，`onCreate(Bundle)`会将变量的值设为默认值。但是，在屏幕方向切换时，这个方法中的 `Bundle` 参数为非 `null` 值，此时，所有变量的值会从传入的这个 `Bundle` 对象中获得。

为了让 `Univerter` 看起来更专业，会在标题栏的左侧添加一个图标，如下：

(1) 可以通过在 `Activity` 的 `boolean requestWindowFeature(int featureId)`方法中传入相应的 `featureId` 参数来启用扩展窗口特性，这里传入的值为 `Window.FEATURE_LEFT_ICON`。如果支持这个特性并可以启用，`requestWindowFeature(int)`方法就会返回 `true`。

(2) 执行 `setContentView(R.layout.main)`填充完 `Activity` 的布局内容后(必须在此时执行该方法)，`onCreate(Bundle)`中才会有条件(`requestWindowFeature(int)`方法返回 `true`)的执行 `Activity` 的 `void setFeatureDrawableResource(int featureId, int resId)`方法来初始化图标，参数为 `Window.FEATURE_LEFT_ICON` 和 `R.drawable.ic_launcher`(`Univerter` 的应用程序启动器图标的资源 ID)。

`setContentView(R.layout.main)`方法会根据当前屏幕方向选择相应的 `main.xml` 布局文件，然后将文件中内容填充到 `Activity` 的用户界面中。竖屏时使用的 `main.xml` 文件保存在 `res/layout` 目录下；横屏时使用的 `main.xml` 则保存在 `res/layout-land`。

`onCreate(Bundle)`方法然后调用 `updateConversionTitle()`方法，该方法会更新 `View`(竖屏模式)以及通过换算名称来更新标题栏(横屏模式)。再然后是初始化 `<EditText>`控件(定义在 `main.xml` 布局文件中)，即显示条，以便后续可以访问它。

接下来，`onCreate(Bundle)`会初始化每个按钮，并将“+/-”按钮的引用保存起来，以便可以在需要时动态地启用和禁用这个按钮，每个按钮的颜色被设置为深灰色，这样可以和

按钮文本的蓝绿色形成鲜明的对比。

为了使按钮的颜色变暗, onCreate(Bundle)中首先调用了 Button 父类(View)的 Drawable getBackground()方法获得一个 android.graphics.drawable.Drawable 实例。然后调用该实例的 void setColorFilter(int color, PorterDuff.Mode mode)方法, 参数为 Color.GRAY 和 Mode.MULTIPLY, 这时按钮的背景颜色即为按钮默认颜色乘以 Color.GRAY。

最后, onCreate(Bundle)得到了显示在 help 对话框中的一个基于 HTML 的帮助文本, 以及表示高亮文本、普通文本、链接文本所需的颜色资源。因为文本中包含这些颜色的占位符, 如#helpHiliteText、#helpText 和#link, 可以使用 java.lang.String 类的 String replaceAll (String regularExpression,String replacement)方法和 private String toHexString(inti, int numNibbles)方法(稍后会讨论)将这些占位符替换为相应的颜色资源。

当用户选择打开 Activity 的选项菜单(即用户选择 MENU 键【如果有的话】或者 Android 3.0 或以上版本中才有的 action bar 上面的更多菜单【竖着 3 个点】, Android 3.0 已经不建议使用 MENU 按钮了)就会调用下面的 boolean onCreateOptionsMenu(Menu menu)。

```
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.univerter, menu);
    return true;
}
```

onCreateOptionsMenu(Menu)会通过 android.view.MenuInflater 类的 void inflate(int menuRes, Menu menu)方法填充相应的菜单资源, 参数即为菜单的资源 ID(R.menu.univerter)和传给 onCreateOptionsMenu(Menu)方法的 android.view.Menu(填充好的菜单项会放在这里)参数, 返回 true 就会显示菜单。

当选择菜单菜单中的菜单项时会调用下面的 boolean onOptionsItemSelected(MenuItem item)方法。

```
public boolean onOptionsItemSelected(MenuItem item)
{
    LayoutInflater inflater;

    switch (item.getItemId())
    {
        case R.id.menu_help:
            inflater = (LayoutInflater) this.
                getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            WebView wv = (WebView) inflater.inflate(R.layout.help, null);
            wv.setBackgroundColor(Color.TRANSPARENT);
            wv.loadData(helpText, "text/html", "utf-8");
            new AlertDialog.Builder(Univerter.this).
                setView(wv).
                setNeutralButton(R.string.btnClose, null).
                show();
            return true;
    }
}
```

```

case R.id.menu_info:
    inflater = (LayoutInflater) this.
        getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    View view = inflater.inflate(R.layout.info, null);
    TextView tv;
    tv = (TextView) ((ViewGroup) view).findViewById(R.id.text1);
    tv.setText(Html.fromHtml(getString(R.string.info1)));
    tv = (TextView) ((ViewGroup) view).findViewById(R.id.text2);
    tv.setText(R.string.info2);
    tv = (TextView) ((ViewGroup) view).findViewById(R.id.text3);
    tv.setText(Html.fromHtml(getString(R.string.info3)));
    tv.setMovementMethod(LinkMovementMethod.getInstance());
    tv = (TextView) ((ViewGroup) view).findViewById(R.id.text4);
    tv.setText(Html.fromHtml(getString(R.string.info4)));
    tv.setMovementMethod(LinkMovementMethod.getInstance());
    ImageView iv;
    iv = (ImageView) ((ViewGroup) view).findViewById(R.id.image);
    iv.setImageResource(R.drawable.ic_launcher);
    new AlertDialog.Builder(Univerter.this).
        setView(view).
        setNeutralButton(R.string.btnClose, null).
        show();
    return true;

default:
    return super.onOptionsItemSelected(item);
}
}

```

`onOptionsItemSelected(MenuItem)`会调用的它的 `android.view.MenuItem` 参数的 `int getItemId()`方法得到所选择的菜单项。`int getItemId()`方法会返回菜单项(菜单项是在 `res/menu/univerter.xml` 文件中声明的)的资源 ID: `R.id.menu_help` 或者 `R.id.menu_info`。之后 `switch` 语句会使用这个 ID 执行相应的分支。在 `default` 分支中, 最好将 `MenuItem` 参数传给父类的 `onOptionsItemSelected(MenuItem)`方法。

每个菜单项都对应一个描述了对话框内容的布局资源。`help` 菜单项的布局资源保存在 `res/layout/help.xml` 中, 而 `info` 菜单项的布局资源保存在 `res/layout/info.xml` 文件中。布局资源使用之前, 首先要进行填充(`inflate`)。

填充过程是通过 Android 的布局服务处理的, 该服务可以将 XML 文件的内容解析为相应的 View 结构。该服务是通过调用 Context 的 `ObjectgetSystemService(String name)`方法获得的, 参数为 `Context.LAYOUT_INFLATER_SERVICE`。这个方法会返回一个通用的 `java.lang.Object` 类型的对象, 我们只需要将其强转为 `android.view.LayoutInflater` 类型即可。

`LayoutInflater` 中声明了一个 `View inflate(int resource, ViewGroup root)`方法来处理填充过程。其中的 `resource` 参数即为要填充的 XML 文件的资源 ID(`R.layout.help` 或 `R.layout.info`)。而 `root` 参数是个可选参数, 代表要填充的 View 结构的父容器。这个值传入 `null` 表示要填充的 XML 的根元素就是父容器。

`inflate(int, ViewGroup)`方法会返回 xml 文件中的根 View。对于 `help` 菜单来说, 返回的

根 View 即为可以用来显示网页的 `android.webkit.WebView` 实例。

`WebView` 中声明了一个 `void setBackgroundColor(int color)` 方法将它的背景颜色设为 `Color.TRANSPARENT`。设置这个值是为了确保对话框下面的背景颜色可以显示出来。

`WebView` 还声明了一个 `void loadData(String data, String mimeType, String encoding)` 方法来向 `WebView` 中加载特定 `mimeType` (`text/html`) 和 `encoding` (`UTF-8`) 类型的数据。这个方法加载之前 `onCreate(Bundle)` 方法中保存在 `helpText` 中的 HTML 数据。

对于 `info` 菜单, 返回的 View 是一个 `android.widget.RelativeLayout` 实例, 它内部的 View 会根据彼此的位置关系进行排列。

`RelativeLayout` 实例中包含了 4 个 `TextView` 实例来显示 4 行文本, 同时还包含有一个 `android.widget.ImageView` 实例来显示一张图片。第 3 个和第 4 个 `TextView` 实例还显示了一个链接, 该链接是通过字符串资源中的 HTML 标记元素来指定的。

标记元素必须转换为字符串显示。可以通过 `android.text.Html` 类的 `Spanned fromHtml (String source)` 方法完成这种转换, 该方法会返回一个 `android.text.Spanned` 实例(可以显示的文本样式), 然后这个实例被传入到 `TextView` 的 `void setText(CharSequence text)` 方法(`Spanned extends java.lang.CharSequence`)中。

需要为链接文本关联一个动作方法(movement method)来识别文本中的链接, 否则即使链接下面显示了下划线, 也是不能单击的。关联动作方法是调用 `TextView` 的 `void setMovementMethod(MovementMethod movement)` 并传入一个 `android.text.method.LinkMovementMethod` 实例(调用 `MovementMethod` 的 `getInstance()` 方法获得)的参数实现的。

`ImageView` 类使用 `void setImageResource(int resId)` 方法并传入一个资源 ID(`resId`)来设置该控件的内容。本例中, 图片资源就是应用程序启动器界面中的应用程序图标。

内容初始化完成之后, `help` 和 `info` 的 `switch` 语句分支还分别创建了一个对话框 builder, 同时调用 `AlertDialog.Builder` 的 `AlertDialog.Builder setView(View view)` 方法将对话框的内容设置为已经初始化好的 View。然后创建对话框并显示它的内容。

最后, 当 Activity 被终止时可以调用下面的 `void onSaveInstanceState(Bundle outState)` 方法保存 Activity 的状态。

```
public void onSaveInstanceState(Bundle outState)
{
    super.onSaveInstanceState(outState);
    outState.putInt("curCat", curCat);
    outState.putInt("curCon", curCon);
    outState.putString("buffer", buffer.toString());
    outState.putInt("state", state);
    outState.putInt("nDigits", nDigits);
    outState.putBoolean("isDecimal", isDecimal);
    outState.putBoolean("btnCvtClicked", btnCvtClicked);
}
```

`onSaveInstanceState(Bundle outState)` 方法首先会调用父类的同名方法保存控件的状态(例如 `etDisplay` 显示条上显示的文本)。然后会调用各种 `Bundle` 方法保存 `Univerter` 应用程序的状态信息。Activity 重新创建时通过 `onCreate(Bundle)` 方法恢复这些状态信息。

3. Univerter 的辅助方法

Univerter 回调方法之后是 4 个辅助方法：

单击数字按钮、小数点按钮或者+/-按钮时会调用下面的 void buildNumber(char ch) 方法。

```
private void buildNumber(char ch)
{
    switch (state)
    {
        case 0: if (ch >= '1' && ch <= '9')
            {
                buffer.append(ch);
                nDigits = 1;
                state = 1;
            }
        else
            if (ch == '.')
            {
                isDecimal = true;
                buffer.append("0.");
                nDigits = 1;
                state = 1;
            }
        break;

        case 1: if (ch >= '0' && ch <= '9')
            {
                if (nDigits != 10)
                {
                    buffer.append(ch);
                    nDigits++;
                }
            }
        else
            if (ch == '.')
            {
                if (isDecimal)
                    break;
                isDecimal = true;
                buffer.append('.');
            }
        else
            if (categories[curCat].getConversion(curCon).canBeNegative() &&
                ch == '-')
            {
                if (buffer.charAt(0) == '-')
                    buffer.deleteCharAt(0);
                else
                    buffer.insert(0, '-');
            }
    }
}
```

```

    }
}

```

`buildNumber(char)`会检查 `state` 变量的值并根据这个值(0 或者 1)处理响应的逻辑。

状态值 0 是最初始的值。这种状态下只处理 1~9 的数字或小数点。按下下一个数字键(保存在参数 `ch` 中)时就会在 `buffer` 中保存这个数字;按下小数点按钮会在 `buffer` 中保存“0.”。

状态值 1 即是最终的状态。这种状态下会处理 0 到 9 的数字、小数点和+/-按钮。

- 输入的数字位数超过小于等于 10 时, 每个单击的数字都会保存到 `buffer` 中。
- 单击小数点后如果在 `buffer` 中没有“.”的话会在 `buffer` 中保存一个“.”。
- 单击+/-按钮后, 首先会检查当前换算是否支持负值。如果支持, `buffer` 中的第一个字符会被设置为“-”。但如果“-”已经存在, 就会移除它。

当需要将输入的数字清空(即 `buffer` 为空)并显示“0.”时, 会调用下面的 `reset()`方法:

```

private void reset()
{
    buffer.setLength(0);
    state = 0;
    nDigits = 0;
    isDecimal = false;
    etDisplay.setText("0.");
}

```

下面的 `String toHexString(int i, int numNibbles)`方法是在 `onCreate(Bundle)`中调用的, 即将 `helpText` 文本内容中的占位符 ID 替换为 6 位十六进制的值:

```

private String toHexString(int i, int numNibbles)
{
    StringBuilder sb = new StringBuilder(Integer.toHexString(i));
    if (sb.length() > numNibbles)
        return null; //无法将结果转换为长度为 numNibbles 的十六进制字符串

    int numLeadingZeros = numNibbles-sb.length();
    for (int j = 0; j < numLeadingZeros; j++)
        sb.insert(0, '0');
    return sb.toString();
}

```

`toString(int, int)`会将要转换的数字转换为固定位数(4 位, 也可以用十六进制表示)的十六进制字符串。

首先会调用 `java.lang.Integer` 类的 `String toHexString(int i)`静态方法执行真正的转换, 然后将结果保存在一个 `java.lang.StringBuilder`(使用该类是为了避免创建不必要的 `String` 对象)对象中。

因为 `toHexString(int)`不会保存前面的 0, 所以这里会计算 0 的个数并把它加到 `String Builder` 的开头, 但也有一个条件, 就是保存在 `String Builder` 中的十六进制的位数不能超过传入参数要求的位数(否则, 会返回 `null`)。

接下来, `StringBuilder` 中的内容会转换为一个字符串返回。

最后, 会调用下面的 `updateConversionTitle()` 方法来更新(刚启动或者单击 CAT、CON 按钮并进行选择时)换算名称。

```
private void updateConversionTitle()
{
    TextView tv = (TextView) findViewById(R.id.conversion1);
    if (tv != null)
    {
        String s = categories[curCat].getConversion(curCon).
            getName(Univerter.this);
        tv.setText(s.substring(0, s.indexOf(">")-1));
        tv = (TextView) findViewById(R.id.conversion2);
        tv.setText(s.substring(s.indexOf(">")+2));
    }
    else
        setTitle(getString(R.string.app_name)+" : "+
            categories[curCat].getConversion(curCon).
            getName(Univerter.this));
}
```

`updateConversionTitle()` 会判断当前是竖屏还是横屏。如果是竖屏, `main.xml` 中会包含两个 `<TextView>` 元素, ID 分别为 `R.id.conversion1` 和 `R.id.conversion2`。这两个元素上的文本分别为换算时的源和目标名称。

如果是横屏, 由于垂直方向上的空间不足, `main.xml` 中并不会显示 `<TextView>` 元素; 否则 `TextView` 的内容很有可能会被截断。相反, 我们会将换算信息和应用程序名称显示在 `Univerter` 的 Activity 标题栏下, 即调用 Activity 的 `void setTitle(CharSequence title)` 方法。

D.2 浏览资源文件

`Univerter` 的资源分布在 14 个文件中:

```
res/drawable/gradientbg.xml
res/drawable-hdpi/ic_launcher.png
res/drawable-ldpi/ic_launcher.png
res/drawable-mdpi/ic_launcher.png
res/drawable-xhdpi/ic_launcher.png
res/layout/help.xml
res/layout/info.xml
res/layout/list_row.xml
res/layout/main.xml
```

```

res/layout-land/main.xml

res/menu/univerter.xml

res/values/colors.xml

res/values/strings.xml

res/values/styles.xml

```

D.2.1 应用程序启动器图标资源

Univerter 项目创建完成后, android 会在项目的 `res/drawable-hdpi`、`res/drawable-ldpi`、`res/drawable-mdpi` 和 `res/drawable-xhdpi` 目录下分别放置一个默认的 `ic_launcher.png` 文件(带有一个绿色机器人的图片)。每个图片的内容相同但分辨率不同。

- `drawable-hdpi` 中的文件为 72×72 像素的图片,用于高像素密度(每英寸 240 个像素点【dpi】)的屏幕。
- `drawable-ldpi` 中的文件为 36×36 像素的图片,用于低像素密度(120dpi)的屏幕。
- `drawable-mdpi` 中的文件为 48×48 像素的图片,用于中等像素密度(160dpi)的屏幕。
- `drawable-xhdpi` 中的文件为 96×96 像素的图片,用于超高像素密度(320dpi)的屏幕。

虽然可以保留默认的图标,但更专业的做法能够展示该应用程序的平台要求。除了通过 Android Asset Studio (<http://android-ui-utils.googlecode.com/hg/assetstudio/dist/index.html>)创建这个图标,也可以在其他地方寻找合适的图标。

Univerter 的图标来自于 Icon Archive(www.iconarchive.com/show/or-icons-by-iconleak/justice-balance-icon.html),并且得益于 IconLeak (<http://iconleak.com/>)。这个图标的内容为一个黄金天平,很适合单位换算类的应用程序,如图 D-1 所示。



图 D-1 4 种尺寸的 Univerter 应用程序启动器图标

想要了解更多关于启动器和其他图标的知识,可以查看 Google 的“LauncherIcons”(http://developer.android.com/guide/practices/ui_guidelines/icon_design_launcher.html)和“Iconography”(<http://developer.android.com/design/style/iconography.html>)页面。

D.2.2 背景 Drawable 资源

Univerter 使用了一个渐变颜色的背景(从顶部深蓝色到底部的淡蓝色),让 Activity 看起来更有意思。该背景是通过定义一个形状 drawable 资源实现的,形状 drawable 则是在 XML 中描述的。参见程序清单 D-5。

程序清单 D-5 Univerter Activity 的渐变颜色的背景

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient android:angle="270"
        android:endColor="@color/bgEnd"
        android:startColor="@color/bgStart" />
</shape>
```

程序清单 D-5 展示了 `res/drawable/gradientbg.xml` 的内容。首先是标准的 XML 文件头，然后是一个 `<shape>` 元素，你可以使用这个元素中的 `shape` 属性指定一种形状。如果没有设置 `shape` 属性，默认的形状为矩形。

`<shape>` 的内部嵌入了一个 `<gradient>` 元素，它描述了形状的颜色渐变情况。渐变是通过给 `startColor` 和 `endColor` 属性(这里恰好引用了 `res/values/colors.xml` 文件中的 `bgStart` 和 `bgEnd` 资源)设置相应的值来定义的。`angle` 属性指定矩形的渐变方向。如果不设置这个值，默认为 0。

想要了解更多关于形状 drawable 资源的信息，可以查看 <http://developer.android.com/guide/topics/resources/drawableresource.html#Shape>。

D.2.3 浏览主布局资源文件

Univerter 的界面是通过 `res/layout/main.xml`(竖屏)或 `res/layout-land/main.xml`(横屏)中定义的布局资源来描述的。每个文件都定义了界面中的部件和部件之间的关系。程序清单 D-6 显示了 `res/layout/main.xml` 中的内容。

程序清单 D-6 竖屏情况下 Univerter Activity 的布局情况

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="@drawable/gradientbg"
    android:gravity="center"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:padding="10dp">
    <TextView android:id="@+id/conversion1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:textColor="@color/conversionText"
        android:textSize="15sp" />
    <TextView android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text=">"
        android:textColor="@color/conversionText"
        android:textSize="15sp" />
    <TextView android:id="@+id/conversion2"
        android:layout_height="wrap_content"
```

```

        android:layout_width="wrap_content "
        android:paddingBottom="30dp"
        android:textColor="@color/conversionText "
        android:textSize="15sp"/>
<EditText android:id="@+id/display"
        android:focusable="false"
        android:gravity="right|center_vertical"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="0."
        android:textColor="@color/displayText"
        android:textSize = "15sp"/>
<LinearLayout android:layout_height="wrap_content"
        android:layout_width="match_parent">
    <Button android:id="@+id/btn7"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="7"
        android:text="@string/btn7"
        android:textColor="@color/keyText"
        android:textSize = "15sp"/>
    <Button android:id="@+id/btn8"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="8"
        android:text="@string/btn8"
        android:textColor="@color/keyText"
        android:textSize = "15sp"/>
    <Button android:id="@+id/btn9"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="9"
        android:text="@string/btn9"
        android:textColor="@color/keyText"
        android:textSize = "15sp"/>
    <Button android:id="@+id/btnClr"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doClrClicked"
        android:text="@string/btnClr"
        android:textColor="@color/keyText"
        android:textSize = "15sp"/>
</LinearLayout>

```

```

<LinearLayout android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <Button android:id="@+id/btn4"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="4"
        android:text="@string/btn4"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btn5"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="5"
        android:text="@string/btn5"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btn6"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="6"
        android:text="@string/btn6"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btnCat"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doCatClicked"
        android:text="@string/btnCat"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
</LinearLayout>
<LinearLayout android:layout_height="wrap_content"
    android:layout_width="match_parent">
    <Button android:id="@+id/btn1"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="1"
        android:text="@string/btn1"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btn2"

```

```

        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="2"
        android:text="@string/btn2"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
<Button android:id="@+id/btn3"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="3"
        android:text="@string/btn3"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
<Button android:id="@+id/btnCon"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doConClicked"
        android:text="@string/btnCon"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
</LinearLayout>
<LinearLayout android:layout_height="wrap_content"
        android:layout_width="match_parent">
    <Button android:id="@+id/btn0"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDigitClicked"
        android:tag="0"
        android:text="@string/btn0"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btnDot"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doDotClicked"
        android:text="@string/btnDot"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btnPm"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doPmClicked"

```

```

        android:text="@string/btnPm"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
    <Button android:id="@+id/btnCvt"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:onClick="doCvtClicked"
        android:text="@string/btnCvt"
        android:textColor="@color/keyText"
        android:textSize = "15sp" />
</LinearLayout>
</LinearLayout>

```

程序清单 D-6 中首先出现的是一个<LinearLayout>元素，它定义了 Activity 的布局。这个元素通过以下属性设置布局：

- `android:background="@drawable/gradientbg"`表示线性布局的背景为 `gradientbg.xml` 文件所定义的矩形渐变。
- `android:gravity="center"`表示线性布局的子 View 水平和垂直居中。
- `android:layout_height="match_parent"`表示线性布局会匹配其父容器(Activity)的宽度。
- `android:layout_width="match_parent"`表示线性布局会匹配其父容器的高度，即 Activity 界面窗口的宽度。
- `android:orientation="vertical"`表示线性布局的子 View 会垂直排列。这个属性的默认值为“horizontal”。
- `android:padding="10dp"`表示线性布局的子 View 的周围会有 10dp 的空白区域。

<LinearLayout>里面首先是 3 个<TextView>元素，然后是一个<EditText>元素，最后是 4 个<LinearLayout>元素。

3 个<TextView>分别表示换算源(例如，摄氏度)、>(即“换算为”)、换算目标(例如，华氏温度)。第 1 个和第 3 个<TextView>还分别指定了一个 `android:id` 属性，即 `"@+id/conversion1"`(代码中可以通过 `R.id.conversion1` 引用到它)和 `"@+id/conversion2"`(代码中可以通过 `R.id.conversion2` 引用到它)。

每个元素还指定了以下属性：

- `android:layout_height="wrap_content"`表示<TextView>高度就应该是其内容的高度。如果设置为 `fill_parent` 或 `match_parent`，<TextView>则会占满所有垂直方向上的剩余的空间。
- `android:layout_width="wrap_content"`表示<TextView>的宽度就应该是其内容的宽度。如果设置为 `fill_parent` 或 `match_parent`，<TextView>则会占满所有水平方向的剩余的空间，而且不会水平居中了。
- `android:textColor="@color/conversionText"`使用了 `res/values/colors.xml` 中定义的 `conversionText` 颜色资源作文本的颜色。
- `android:textSize="15sp"`表示了文本的字号为 15sp。相关的术语定义可以查阅第 1 章。

<EditText>控件用于显示输入和换算结果。它是用 `android:id="@+id/display"` 属性(在代码中可以使用 `R.id.display` 引用它)进行标识的。此外,除了和<TextView>一样的属性外,还指定了一些其他属性,如下:

- `android:focusable="false"`表示这个控件不能获得焦点。如果可以获得焦点,用户就可以在这个控件中输入任意字符,这是我们不希望看到的。可以防止输入的另一相关属性就是 `android:editable="false"`。但是,在控件无法获得焦点的情况下,也没必要设置这个属性了。
- `android:gravity="right|center_vertical"`表示这个控件的文本应该在靠右和垂直居中显示。如果不设置这个属性,控件的文本会靠左并稍微垂直居中显示。
- `android:layout_width="match_parent"`表示这个控件的宽度应该匹配其父容器(线性布局)的宽度(保留少许的 `padding`)。如果设置为“`wrap_content`”,这个控件的宽度就会是其文本的宽度,看起来不太像计算器的界面。
- `android:text="0."`表示控件的初始值。或许你想通过引用字符串资源的方式再设置这个值(例如, `android:text="@string/zerodisplay"`)。然而,这种做法必要性不大。

<EditText>之后是 4 个<LinearLayout>元素。每个<LinearLayout>元素的内容都是水平排列并且设置了 `android:layout_height="wrap_content"`和 `android:layout_width="match_parent"`属性。前面的属性表示<LinearLayout>中内容的高度就是其内容的高度,不会占用垂直方向剩余的空间。后面的属性标识<LinearLayout>的内容会占满水平方向的空间(保留少许的边距)。

每个<LinearLayout>元素内部有 4 个<Button>元素。每个<Button>所指定的属性和之前介绍差不多,除了下面 3 个属性:

- `android:layout_weight="1"`表示会尽可能多地占用水平方向的剩余空间。由于每个线性布局中的 4 个按钮都设置这个属性,所以每个按钮的水平宽度都是一样的。
- `android:onClick="doDigitClicked"`表示按钮被单击时所调用的单击事件处理方法。本例中,会调用 `doDigitClicked(View)`方法。
- `android:tag="7"`意味着按钮是通过一个名字值来标识的,和地域无关。无论给按钮赋予什么样的文本,这个值都不会受到影响,如前面所述,通过传入 `doDigitClicked(View)`方法中的 `View` 实例的 `getTag()`方法即可得到这个名字值。

除了 3 个<TextView>意外,位于 `res/layout-land` 中的 `main.xml` 文件中内容和程序清单 D-6 的内容都是一样的。

D.2.4 浏览列表中每行的布局资源

第 1 章的图 1-19 和 1-20 展示了 `category name` 和 `conversionname` 对话框。这两个对话框看起来都差不多,但 `conversionname` 字号要小一些,这是为了在 `conversionname` 比较长时会好看一点。程序清单 D-7 展示了如何实现这种小字号文本。

程序清单 Conversion Name 对话框中每行的布局

```
<?xml version="1.0" encoding="utf-8"?>
<CheckedTextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:checkMark="@android:drawable/btn_radio"
    android:gravity="center_vertical"
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:paddingLeft="12dp"
    android:paddingRight="7dp"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:textColor="?android:attr/textColorPrimaryInverseDisableOnly"/>
```

程序清单 D-7 展示了之前在 `doConClicked(View)` 方法中 `res/layout/list_row.xml` 的内容, 这个文件之前在 `doConClicked(View)` 方法中提到过。这些内容都有基于 `<CheckedTextView>` 元素的, `<CheckedTextView>` 源自 `<TextView>` 元素但会在 `text` 后面显示一个选中标记。

`android:checkMark="@android:drawable/btn_radio"` 属性指定了文本右侧的选中标记的样式。这里的样式用的是 Android 平台的单选按钮(见第 1 章)的样式资源, 单选按钮资源在代码中可以通过 `android.R.drawable.btn_radio` 访问。

`android:gravity="center_vertical"` 会让文本和单选按钮样式的选中标记垂直居中显示。

`android:minHeight="?android:attr/listPreferredItemHeight"` 属性指定了列表中每行的最小高度。它引用了 `android.R.attr.listPreferredItemHeight` 所对应的资源来设置当前的主题。

注意:

与 `@` 符号(可以引用其他项目资源文件中的资源, 如 `res/values/strings.xml`)不同, `?` 符号会引用当前主题(应用于整个应用程序或者 Activity 的主题, 而不是应用于某个 view 的主题)中的资源。

`android:paddingLeft="12dp"` 和 `android:paddingRight="7dp"` 分别指定了文本左侧(12dp)或者圆形选中标记右侧(7dp)的 `padding` 大小。`android:textAppearance="?android:attr/textAppearanceSmall"` 属性则指定了文本的字号大小。它引用了 `android.R.attr.textAppearanceSmall` 所对应的资源设置当前主题。

最后, `android:textColor="?android:attr/textColorPrimaryInverseDisableOnly"` 属性指定了文本的颜色。它引用了 `android.R.attr.textColorPrimaryInverseDisableOnly` 资源来设置当前的主题。

注意:

引用主题资源可以在 Univerter 的主题更换时, 让 `conversion name` 列表和 `category name` 的界面风格保持一致。

D.2.5 浏览选项菜单的资源

Univerter 中定义了一个包含 `help` 和 `info` 菜单项的选项菜单。菜单的布局如程序清单

D-8 所示:

程序清单 D-8 选项菜单的布局

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_help"
        android:icon="@android:drawable/ic_menu_help"
        android:title="@string/menu_help"/>
    <item android:id="@+id/menu_info"
        android:icon="@android:drawable/ic_menu_info_details"
        android:title="@string/menu_info"/>
</menu>
```

程序清单 D-8 显示了 res/menu/univerter.xml 文件的内容, 该文件描述了菜单用到的资源。在<menu>中内嵌了两个<item>元素, 分别描述了 help 和 info 菜单项。

每个<item>都通过 android:id 标识自己, 这样在 Univerter.java 就可以通过这个 id(R.id.menu_help 或 R.id.menu_info)访问它们。此外, 它还通过 android:icon 来访问 Android 平台的图标资源(如 android.R.drawable.ic_menu_help 和 android.R.drawable.ic_menu_info_details)并通过 android:title 设置相应的标题。

D.2.6 浏览 Help 对话框布局资源

单击 help 菜单项会显示一个对话框, 布局内容如程序清单 D-9。

程序清单 D-9 Help 对话框的布局

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/helpText"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
```

程序清单 D-9 展示了 res/layout/help.xml 文件的内容, 会让一个<WebView>元素充满它的父对话框(将 layout_height 和 layout_width 的值设置为"wrap_content", 也可以实现相同的效果)。

D.2.7 浏览显示 Info 对话框的布局资源

单击 info 菜单项会显示一个对话框, 布局内容如程序清单 D-10。

程序清单 D-10 Info 对话框的布局

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

```

        android:padding="10dp">
<ImageView android:id="@+id/image"
    android:layout_centerVertical="true"
    android:layout_height="wrap_content"
    android:layout_marginRight="10dp"
    android:layout_width="wrap_content"/>
<LinearLayout android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_toRightOf="@+id/image"
    android:orientation="vertical">
    <TextView android:id="@+id/text1"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textColor="@color/infoText"/>
    <TextView android:id="@+id/text2"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textColor="@color/infoText"/>
    <TextView android:id="@+id/text3"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textColor="@color/infoText"
        android:textColorLink="@color/link"/>
    <TextView android:id="@+id/text4"
        android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:textColor="@color/infoText"
        android:textColorLink="@color/link"/>
</LinearLayout>
</RelativeLayout>

```

程序清单 D-10 展示了 res/layout/info.xml 文件的内容。和 main.xml 将布局内容嵌入到 <LinearLayout> 元素不同，info.xml 会将布局内容放入一个 <RelativeLayout> 元素中。

<RelativeLayout> 可以根据相对位置组织子 View。每个 View 的位置可以根据相邻元素的相对位置(例如，在另一个 View 的左边或者下面)或者相对于父 <RelativeLayout> 的位置(例如，在父布局的底部、左侧中间)来指定。

<RelativeLayout> 中指定了一个 android:gravity="center_horizontal" 属性，确保它的内容可以水平居中。还指定了一个 android:padding="10dp" 属性，可以在内容的四周创建 10dp(设备无关的像素)的空白区域。

整个布局内容由一个 <ImageView> 和一个有 4 个 <TextView> 元素的 <LinearLayout> 组成。

<ImageView> 元素表示一张图片，图片内容会在代码中进行指定，它指定了一个 android:layout_centerVertical="true" 元素让图片可以垂直居中显示，还指定了属性 android:layout_marginRight="10dp" 让图片和它右侧的 View 保持 10dp 的距离(这样，图片和文本会

区分开来)。

<LinearLayout>元素指定了 `android:layout_toRightOf="@+id/image"` 属性, 确保它内部的<TextView>可以显示图片的右侧。

每个<TextView>元素都指定了 `android:gravity="center"` 属性, 让文本内容在自己的空间中居中显示(父<LinearLayout>提供了每个元素的空间)。

D.2.8 浏览颜色资源

Univerter.java 和其他资源文件使用的颜色资源都是在 `res/values/colors.xml` 中声明的。程序清单 D-11 展示了这些颜色资源。

程序清单 D-11 Univerter 的颜色

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="bgEnd">#168fc7</color>
    <color name="bgStart">#000080</color>
    <color name="conversionText">#00ffff</color>
    <color name="displayText">#000000</color>
    <color name="helpHiliteText">#eeee00</color>
    <color name="helpText">#ffffff</color>
    <color name="infoText">#ffffff</color>
    <color name="keyText">#00ffff</color>
    <color name="link">#00ffff</color>
</resources>
```

颜色资源是通过<resources>中的<color>元素来描述的。每个<color>标签会通过 `name` 属性标识一种颜色。夹在<color>和</color>之间的值就是颜色的值。

D.2.9 浏览字符串资源

Univerter.java 和其他资源文件使用的字符串资源都是在 `res/values/strings.xml` 中声明的。程序清单 D-12 展示了这些字符串资源。

程序清单 D-12 Univerter 的字符串资源

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Univerter</string>

    <string name="btn0">0</string>
    <string name="btn1">1</string>
    <string name="btn2">2</string>
    <string name="btn3">3</string>
    <string name="btn4">4</string>
    <string name="btn5">5</string>
    <string name="btn6">6</string>
```

```

<string name="btn7">7</string>
<string name="btn8">8</string>
<string name="btn9">9</string>
<string name="btnCat">CAT</string>
<string name="btnClose">Close</string>
<string name="btnClr">CLR</string>
<string name="btnCon">CON</string>
<string name="btnCvt">CVT</string>
<string name="btnDot">.</string>
<string name="btnPm">+/-</string>

<string name="cat_angle">ANGLE</string>
<string name="cat_angle_circles_to_deg">CIRCLES > DEGREES</string>
...
<string name="menu_help">Help</string>
<string name="menu_info">Info</string>
<string name="overflow">Overflow</string>
<string name="underflow">Underflow</string>
</resources>

```

字符串资源是通过<resources>中的<string>元素来描述的。每个<string>标签会通过name 属性标识一种颜色。夹在<color>和</color>之间的值就是字符串的值。

这其中的一些资源里面会嵌入 HTML 文本。需要将要嵌入的 HTML 文本放到标准的前缀(XML ![CDATA[]和相应的后缀()])之间，如下所示(为了便于阅读，分两行显示)：

```

<string name="info1"><![CDATA[<html><strong>Univerter (Units Converter)
1.0</strong></html>]]></string>

```

D.2.10 浏览样式资源

本章的前面，学习到当 Activity 处于横屏时，在标题栏上显示一个 conversion name。一些 conversion name 会很长，可能会在右侧截断，通过主题标题栏上的文本长度已经缩减了。

程序清单 D-13 展示了这个主题的内容。

程序清单 D-13 Univerter 的主题

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="CustomTheme" parent="android:Theme.Black">
    <item name="android:windowTitleStyle">@style/CustomWindowTitle</item>
  </style>

  <style name="CustomWindowTitle">
    <item name="android:shadowColor">#BB000000</item>
    <item name="android:shadowRadius">2.75</item>
    <item name="android:singleLine">true</item>
    <item name="android:textAppearance">@style/CustomTAWindowTitle</item>
  </style>

```

```

<style name="CustomTAWindowTitle" parent="android:TextAppearance.WindowTitle">
    <item name="android:textSize">10sp</item>
</style>
</resources>

```

程序清单 D-13 展示了 `res/values/styles.xml` 的内容。这个资源文件的结构与 `colors.xml` 和 `strings.xml` 很相似,也是在`<resources>`和`</resources>`标签之间嵌入一系列的资源。不同的是,每个资源是通过`<style>`元素描述的。

第一个`<style>`引入了一个名为 `CustomTheme` 的样式。它从标准的 `android:Theme.Black` 主题(它定义了 `Activity` 和应用程序的一系列样式)中继承了相应的样式属性,并且覆写了这个主题的 `android:windowTitleStyle` 属性(通过内部的`<item>`元素)来引用 `CustomWindowTitle`。

注意:

选择从 `android:Theme.Black` 而不是从 `android:Theme` 继承属性是为了确保对话框的背景颜色是黑色的。这是因为显示在帮助对话框中的 HTML 内容都是透明背景和白色文本(对于白色背景和白色文本则是不可取的)。

`CustomWindowTitle` 是一个`<style>`元素,它通过`<item>`元素定义了 `android:shadowColor`、`android:shadowRadius`、`android:singleLine` 和 `android:textAppearance` 属性。前 3 个属性的值都可以在 `android:Theme.Black` 中找到。最后一个属性则引用了 `CustomTAWindowTitle`。

`CustomTAWindowTitle` 也是一个`<style>`元素,它从 `android:TextAppearance.WindowTitle` 父样式那里继承了相应的样式属性,并且通过`<item>`元素覆盖了父样式的 `android:textSize` 属性,将标题栏上的文本字号缩减为 10sp(scale-independent pixel)。

如果你想知道这种`<style>`元素结构的源代码,可以查看 Android 的 `themes.xml` 和 `styles.xml` 文件的内容,链接如下:

- https://github.com/android/platform_frameworks_base/blob/master/core/res/res/values/themes.xml
- https://github.com/android/platform_frameworks_base/blob/master/core/res/res/values/styles.xml

看过这些文件后,你可能想知道为什么没有在第二个`<style>`中指定 `parent="android:WindowTitle"`以及 therefore the `android:shadowColor`、`android:shadowRadius` 和 `android:singleLine` 属性是重复的。这些都和 Android API 有关系。

Android 包会包含一个名为 `R` 的类,该类中又会内嵌一个 `style` 类。这个类中会声明 `android:Theme.Black(android.R.style.Theme_Black)`公共常量和 `android:TextAppearance.WindowTitle(android.R.style.TextAppearance_WindowTitle)`公共常量。但不会声明 `android:WindowTitle` 的公共常量。

D.3 浏览 Manifest 文件

Univerter 应用程序是通过程序列表 D-13 所示的 `AndroidManifest.xml` 文件来描述的。

程序清单 D-13 描述 Univerter 应用程序的 Manifest 文件

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ca.tutortutor.univerter"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="10"/>
    <application android:label="@string/app_name"
        android:icon="@drawable/ic_launcher"
        android:theme="@style/CustomTheme">
        <activity android:name="Univerter"
            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>

```

这个文件和第 1 章的文件基本一样，但有两个地方需要注意：

- 在<manifest>和<application>之间出现了<uses-sdk android:minSdkVersion="10"/>元素。这个元素会防止 Univerter 在低于 API Level 10(Gingerbread 2.3.3)的 Android 版本上运行。
- <application>标签下面包含了一个 android:theme="@style/CustomTheme"属性。这个属性会让应用程序使用之前介绍的 CustomTheme 样式，即标题栏会显示小号文本(这个应用程序只有一个 Activity，所以这个属性也可以直接应用到<activity>标签上)。