TURING 图灵程序设计丛书 移动开发系列

Apress®

- 汇集Android社区智慧
- 全面、详尽、示例支持
- 移动开发人员必读必备

Beginning Android 2

Begin the journey toward your own successful Android 2 applications

Android开发入门教程

人民邮电出版社
POSTS & TELECOM PRESS

Beginning Android 2 Begin the journey toward your own successful Android 2 applications

Android开发入门教程

欢迎进入神秘的Android世界！自2008年相关设备进入市场以来，Android的发展日新月异，Android设备日益繁盛，而其背后开发应用潜藏的经济效益也展露无遗！

本书将引领大家开发引人入胜的Android 2.x应用，包括如何设计GUI、如何使用GPS和访问Web服务，以及如何将理念转换成实际应用！本书包含大量即时可用的简单示例，自此之后，构建实际、流行的应用将不再困难，只要你投入时间，发挥出创造力。

通读本书，你将了解以下内容：

- 应用Android为各种手机和设备构建基于Java的移动应用
- 同时使用Android部件框架和内置的WebKit驱动的Web浏览器组件创建UI
- 使用Android引擎特性，包括位置跟踪、地图、因特网访问
- 使用和创建整合了活动、服务、内容提供程序和广播接收器的Android应用
- 支持Android 1.5、1.6和2.0 设备，包括处理多个版本的Android OS、多种屏幕尺寸和其他特定于设备的特性

Mark L. Murphy 著名Android学习社区CommonsWare的创始人，畅销书*Busy Coder's Guide to Android Development*的作者。拥有25年软件开发经验，从20世纪70年代末的TRS-80到21世纪最新的移动设备，都留下了他开发实践的足迹。三次创业的经历，让Mark拥有丰富的社会阅历，并曾为财富500强公司提供应用程序开发方面的咨询服务。Mark除了是相关会议及培训方面的头面人物之外，还为著名Android博客AndroidGuys和专业IT杂志*NetworkWorld*的专栏写作。他也是文集*Rebooting America*（Personal Democracy Press，2008）的作者之一。

Apress®

图灵网站：www.turingbook.com 热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com
有奖勘误：debug@turingbook.com

分类建议 计算机/程序设计/移动开发

人民邮电出版社网址：www.ptpress.com.cn

人民邮电出版社 POSTS & TELECOM PRESS

ISBN 978-7-115-24116-0

9 787115 241160 >

ISBN 978-7-115-24116-0

定价：59.00元

TURING 图灵程序设计丛书 移动开发系列

Android开发入门教程

[美] Mark L. Murphy 著
李雪飞 吴明晖 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Android开发入门教程 / （美）墨菲（Murphy, M. L.）著 ; 李雪飞, 吴明晖译. -- 北京 : 人民邮电出版社, 2010.12
（图灵程序设计丛书）
书名原文: Beginning Android 2
ISBN 978-7-115-24116-0

Ⅰ. ①A… Ⅱ. ①墨… ②李… ③吴… Ⅲ. ①移动通信－携带电话机－应用程序－程序设计－教材 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字(2010)第201412号

内 容 提 要

本书是一部关于 Android 2 开发的基础教程。书中结合简单实例，讲解了创建用户界面、内置的 Web 浏览器组件、菜单开发、SDK 工具、多媒体、ContentProvider、位置服务、地图 API、多点触摸，以及 Android 2 新增特性。

本书适合对 Android 平台感兴趣的移动开发人员参考学习。

图灵程序设计丛书

Android开发入门教程

◆ 著 [美] Mark L. Murphy
译 李雪飞 吴明晖
责任编辑 朱 巍
执行编辑 毛倩倩

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：19.25
字数：466千字 2010年12月第1版
印数：1－3 000册 2010年12月北京第1次印刷

著作权合同登记号 图字：01-2010-2362号
ISBN 978-7-115-24116-0

定价：59.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

前　言

欢迎阅读本书

感谢你对开发 Android 应用程序怀有浓厚兴趣！毋庸置疑，越来越多的人在访问因特网服务时，都将使用所谓“非传统的”手段，如移动设备。现在我们在这个领域里做得越多，人们就会在这个领域投入更多的钱，从而使得将来构建更强大的移动应用程序会变得更容易。Android 还很新（基于 Android 的设备在 2008 年底才刚刚出现在市场上），但相信在兼具深度和广度优势的开放手机联盟（Open Handset Alliance）的支持下，它一定能够迅速地发展壮大。

在此，首先感谢你选择这本书，由衷地希望本书能够对你有所帮助，或者至少能让你不时地露出会心一笑。

预备知识

如果你想学习为 Android 编写应用程序，那么你至少要理解 Java 编程的基本概念。Android 编程使用的是 Java 语法和一个作为 Java SE 子集的类库（还有特定于 Android 的扩展）。如果此前你没有使用 Java 编写过程序，那在学习 Android 编程之前恐怕还得补上这一课。

本书不会详细介绍怎样下载或安装 Android 开发工具，包括基于 Eclipse IDE 的工具或者其他独立的工具。这些内容都可以在 Android 网站上查到。本书内容与你是否使用 IDE 应该没有什么关系[①]。不过，要是你想试验本书中给出的任何示例，那么恐怕就得下载、安装和测试 Android 网站上列出的 Android 开发工具了。

本书中的有些章节可能会引用前面章节的内容。而且，也不是书中的每个示例都会给出完整的源代码；否则本书就太厚了。如果读者想要编写示例，可以从 Apress 网站（www.apress.com）下载完整的源代码[②]。

① 原文似乎有误。——译者注

② 可以从图灵公司网站（www.turingbook.com）免费注册下载。——译者注

本书版本说明

本书是 Apress 和 CommonsWare 合作的产物。你现在看到的是 Apress 的版本，是纸质的，此外还有针对不同的数字图书服务的电子版，例如 Safari 版。

CommonsWare 会继续在最初文本内容的基础上不断保持更新，并随时提供给 Warescription 项目的成员，而书名则是 *The Busy Coder's Guide to Android Development*。

CommonsWare 的网站上有关于这一合作关系的 FAQ（问答），地址为 http://commonsware.com/apress。

源代码及许可

读者可以从 www.apress.com 下载本书的源代码。如果你想修改或将其中的示例用作其他用途，请注意源代码中的所有 Android 项目都必须遵循 Apaceh 2.0 许可，参见 www.apache.org/licenses/LICENSE-2.0.html。

致谢

首先要感谢 Android 开发团队，不仅谢谢他们奉献了如此优秀的产品，更因为他们在 Android Google Groups 中提供了无私帮助。特别是，我要感谢 Romain Guy、Justin Mattson、Dianne Hackborn、Jean-Baptiste Queru、Jeff Sharkey 和 Xavier Ducrohet。

本书示例用到的图标由 Nuvola 图标集[①]提供：www.icon-king.com/?p=15。

① Nuvola（名称来自意大利语的“云”，是 SKY 图标主题的改进版）是一套自由软件图标，并在 GNU LGPL 2.1 下发布。Nuvola 的作者是 David Vignoni。这套图标最初是为像 KDE 和 GNOME 之类的桌面环境所开发，现在也已经有 Windows 和 Macintosh 的版本。图标的最终版本 1.0 包含了将近 600 个图标。默认的图标为 PNG 格式，但也提供了 SVG 版本。这套图标用丰富的色彩来表现了大量常用并易于辨认的物体。大部分图标颜色为蓝色，但其他颜色也很好看。（摘自：http://zh.wikipedia.org/zh-cn/Nuvola。）——编者注

目　录

第 1 章 Android 开发概述

大体来说，Android 设备指的就是手机。目前，虽然大家都在讨论 Android 技术在其他领域（如车载“电脑”中）的应用，但谈论的焦点还是手机应用程序。对开发人员来说，为手机开发应用程序既能体验到新鲜刺激，但又要应对诸多挑战。

1.1 智能手机编程的挑战

从好的方面说，基于 Android 的智能手机确实非常之炫。通过移动设备提供因特网服务可以追溯到 20 世纪 90 年代中期的 HDML（Handheld Device Markup Language，手持设备标记语言）。但是直到最近几年，手机上网才真正成为现实。由于手机短消息的风行和苹果 iPhone 的出现，手机作为能上网的设备而迅速走红。因此，开发 Android 应用程序能让你在日新月异的细分市场（能够上网的手机）中，充分体验新技术（Android）带来的刺激。

可是，在你不得不为一些麻烦事而绞尽脑汁时，问题就来了。

任何有 PDA 或手机编程经验的人，都对手机的问题心知肚明——各方面都太小了，例如：

- 屏幕小（你不可能听到这样问题：“你的口袋里装了一块 24 英寸的 LCD，还是……？”）；
- 键盘小（如果有键盘的话）；
- 定位设备（如果有）不是给人制造麻烦（想想那些丢了手写笔的人），就是不够精确（粗大的手指跟“多点触摸”LCD 可不容易和谐）；
- 与桌面电脑或服务器相比，CPU 的速度和内存有限；
- 你可以使用任何编程语言和开发框架，前提是设备制造商必须选中并且将它写入了手机的固件里。

更令人难受的是，运行在手机上的应用程序无法回避一个事实：它们运行在手机上。

人们在打电话遇到问题时，通常很容易着急上火，这也是为什么 Verizon Wireless 的“你听得见我说话吗？”系列广告流行那么多年的原因所在。同样还是这些人，他们在看到你的程序

让他的手机“中断”时，同样也会气急败坏的；程序为什么会导致手机“中断”？以下是可能的原因。

- 耗尽CPU资源导致电话接不进来。
- 在手机来电或需要接听时不能安静地隐藏到后台，因为程序与手机操作系统不能很好地协调运作。
- 导致手机的操作系统宕机，例如让内存像过筛子一样泄漏。

事实上，开发手机程序与开发桌面程序、开发网站或开发服务器端程序有很大不同：所用的工具不同、框架的行为不同，开发程序受到的限制更多。

而Android的目的就是把你解放出来。

- 让你可以使用常用的编程语言（Java），常用的库（例如，某些Apache Commons API），甚至使用你熟悉的开发工具（Eclipse）。
- 让你有一个相当严格而且独立的框架，以便确保你的程序能够在手机里成为一个“好公民”，不会妨碍其他程序或手机本身的正常运行。

聪明的读者想必已经猜到了，本书很在程度上就是要讲述怎么使用这个框架，怎么编写在其权限范围内运行的程序，以及怎样挖掘它的潜力。

1.2　Android由哪些部分构成

在开发桌面应用程序时，你是“自己领域的主宰”。你可以启动主窗口以及任意多个子窗口，如对话框。从开发人员的角度看，一切都是你说了算，你可以想方设法地去利用操作系统提供的各种特性。而且很大程度上，你都不必考虑同一台计算机上还会同时运行其他应用程序。如果要与其他程序交互，一般都是通过API，如JDBC（Java Database Connectivity，Java数据库连接），或者构建于该API之上的构架，来与MySQL或者其他数据库通信。

开发Android程序的概念也类似，但是不同的封装和组织方式，确保了手机不会轻易崩溃。以下是Android应用程序中会用到的主要组件。

- `Activity`：用户界面是由`Activity`构建而成的。可以将Android程序中的`Activity`想象成桌面应用程序中的窗口或对话框。虽然`Activity`可以没有自己的用户界面，但多数情况下那些“没头没脑”的代码更适合以`ContentProvider`或`Service`的形式实现。
- `ContentProvider`：`ContentProvider`用于为设备中存储的数据提供了一个抽象层，以便不同的应用程序访问。Android开发模型鼓励你将自己的数据公开给自己的和其他的应用程序。通过创建`ContentProvider`来实现数据共享，可以对别人访问数据的方式拥有完全的控制权。

- Service：Activity 和 ContentProvider 都是短命的，随时可能被关掉。但 Service 则是可以独立于 Activity 而长期运行的（如果有必要的话）。因此，可以使用 Service 来检查更新 RSS 新闻源，或者在启动它的 Activity 退出之后仍然通过它来播放音乐。
- Intent：Intent 是系统消息，在设备内部运行，向应用程序发布各种事件，包括硬件状态变化（例如，插入了 SD 卡）、收到数据（例如，收到一条短信），以及应用程序事件（例如，用户通过设备的主菜单打开了 Activity）。除了响应 Intent 之外，你还可创建自定义 Intent 以打开其他 Activity，或者让它在特定的情形下给你发送通知（例如，当用户走到离某个地点 100 米范围内时，就触发某个 Activity）。

1.3 你能够控制什么

Android 提供了很多特性，用来辅助你开发应用程序。

- 存储：可以在应用程序中封装不会发生变化的数据，比如图标或帮助文件等。也可以占用设备中的一部分空间，用于保存包含用户输入或其他有用数据的文件或数据库。在用户提供大块存储空间的情况下（如 SD 卡），也可以从中读取或向其中写入文件。
- 网络：Android 设备通常随时都可以上网，而且接入网络的方式不止一种。你可以在任何级别上利用因特网访问，从使用原始的 Java 套接字，到在应用程序中嵌入内置的基于 WebKit 的浏览器的小部件。
- 多媒体：Android 设备具有播放和录制音频和视频的功能。虽然具体的实现方式因设备而异，但能够通过请求设备可以获知它的功能，然后再随心所欲地利用其多媒体功能。不管是播放音乐、拍照片，还是通过麦克风来录音。
- GPS（Global Positioning System，全球定位系统）：Android 设备会频繁访问位置服务（如 GPS），以便应用程序知道当前设备在地球上的什么位置。进而，可以显示该位置的地图，或者以其他方式来利用位置数据，例如在设备丢失的情况下跟踪设备的移动。
- 手机服务：当然，Android 设备通常都是手机。因此，你的程序可以拨打电话、发送和接收 SMS（Short Message Service，短信息服务）消息，以及实现最新的电话技术所能实现的一切功能。

第2章

项目和目标

2

下载并安装好最新的 Android SDK（Android Software Development Kit，Android 软件开发包）和基于 Eclipse 的 ADT（Android Developer Tools，Android 开发人员工具）插件[①]之后，就可以动手开发 Android 应用程序了。本章介绍与开发 Android 应用程序有关内容。

2.1 基本概念

在开发 Android 应用程序之前，必须创建一个相应的 Android 项目。如果你使用 Eclipse 开发的话，那就是创建一个 Eclipse 项目。这个项目用来保存所有源代码、资源（例如国际化字符串）、第三方 JAR 及相关内容。Android 构建工具（集成在 Eclipse 中的或独立的构建工具）会将项目的内容转换成 APK（Android Package，Android 包）文件，这也就是 Android 应用程序。这些工具还可以帮你把 APK 文件部署到 Android 模拟器，或者部署到真实的 Android 设备中，以便测试。

项目中的最关键的一个组成部分是描述文件（manifest），即 AndroidManifest.xml。这个文件中包含着应用程序的“目录”，列出了主要的应用程序组件、权限，等等。Android 在运行时会通过这个描述文件将应用程序与操作系统关联起来。此外，Android Market（或其他独立的“应用程序商店”）也会用到这些描述文件，以便使 Android 2.0 的应用程序不会被安装到运行 Android 1.5 的设备上。

要使用模拟器来测试应用程序，必须先创建一个 AVD（Android Virtual Device，Android 虚拟设备）。多数情况下，都需要创建很多 AVD，每个 AVD 分别模拟一种基于特硬件的 Android 设备。此外，还可以为不同屏幕尺寸、不同 Android 版本创建不同的 AVD。

在创建项目和 AVD 时，需要为 Android 指定要使用的 API 级别。API 级别是一个整数值，不

① Android SDK 和 ADT 都可以从 Android 开发人员站点（http://developer.android.com/）下载。如果官方站点打不开，可以访问 http://tinyurl.com/y66sfw7 或自行搜索下载；ADT 可以通过 Eclipse 更新管理器安装，安装说明参见 http://www.cn-cuckoo.com/2010/05/08/eclipse-adt-1618.html。——译者注

同的整数值映射到不同的 Android 版本（例如，API 3 对应 Android 1.5）。创建项目时，可以告诉 Android 你的应用程序支持的最小和最大 API 级别。而在创建 AVD 时，则是告诉 Android 新建的这个 AVD 要模拟哪个级别的 API。这样，就可以看到自己的应用程序在支持不同 Android 版本的不同（模拟）设备上运行的效果了。

上面提到的所有概念都会在本章中详细介绍。

2.2 创建项目

要想在命令行中创建项目（以便使用 ant 等构建工具），需要运行 `android create project` 命令。这个命令带有一些参数，用于指定应用程序代码所属的 Java 包、应用程序针对的 API 级别，等等。成功运行命令后，会得到一个文件夹，其中包含构建基本的 Android 应用程序所需的全部文件。

下面是一个 `android create project` 命令的示例：

```
android create project --target 2 --path ./FirstApp --activity FirstApp --package
apt.tutorial
```

如果你想使用 Eclipse 开发 Android 应用程序，而不是使用 `android create project` 方式，那么就需要使用 Eclipse 的新项目向导来创建一个新 Android 应用程序。

注意 本书的源代码是按照使用命令行构建工具构建应用程序而创建的。如果你要使用Eclipse，可以创建空Eclipse项目，再将代码导入到项目中。[①]

2.3 项目结构

每个 Android 项目都包含一套特定的目录树结构（非常类似 Java 项目），而 Android 构建系统就是基于该目录树结构组织的。不过，Android 的这套目录结构还是有点与众不同。下面我们就来简单介绍一下 Android 项目的目录结构，以便读者有个大概的印象，特别是让你能够轻松地看明白本书给出的示例代码。

2.3.1 根目录

在创建了新 Android 项目后（例如通过 `android create project` 命令），你会在项目的根目录下发现一些文件和子目录，以下是对它们的简要说明。

---

① 译者是这样做的：在 New Android Project 对话框中，选择 create project from existing source，通过 Browse...按钮找到源代码目录，打开具体示例项目的目录，Eclipse 即可自动填写有关新项目的信息。——译者注

- AndroidManifest.xml：是一个 XML 文件，用于描述将被构建的应用程序，以及应用程序中包含哪些组件（Activity、Service，等等）。
- build.xml：是一个 Ant 脚本，用于编译应用程序和在设备上安装应用程序。
- default.properties 和 local.properties：是属性文件，由 Ant 构建脚本使用。
- assets/：文件夹，用于存放需要打包到应用程序中的静态文件，以便部署到设备中。
- bin/：文件夹，用于存放编译后的应用程序。
- gen/：文件夹，Android 构建工具存放它们生成的源代码的地方。
- libs/：文件夹，用于存放应用程序用到的第三方 JAR。
- src/：文件夹，用于存放应用程序的 Java 源代码。
- res/：文件夹，用于存放应用程序的资源（如图标、GUI 布局等），将被打包到编译后 Java 中。
- tests/：文件夹，用于存放完全独立的 Android 项目，以便测试你所创建的项目。

2.3.2　主 Activity

在创建 Android 项目时（例如通过 android create project 命令），我们为应用程序的主活动指定了一个完整的类名（例如：com.commonsware.android.SomeDemo）。然后，你就会在项目的 src/目录下看到映射过来的命名空间目录树结构，以及一个表示主活动的 Activity 的待实现子类（例如，src/com/commonsware/ android/SomeDemo.java）。你可以修改这个文件，或者再向这个目录中添加实现应用程序所需要的其他文件。

首次编译项目时（例如通过 ant 命令），Android 在构建项目时会在主活动命名空间目录的外部创建 R.java。这个文件中包含很多常量，与你放在 res/文件夹中的各种资源一一对应。在阅读本书的过程中，你会发现很多示例中都会引用 R.java（例如，R.layout.main 就是引用布局文件的标识符）。

注意　你不能动手修改R.java，Android工具会为你自动创建和更新这个文件。

2.3.3　资源

如前所述，res/目录中保存着资源，即将会打包到应用程序中的静态文件（可能是原始格式，也可能是经过预处理的格式）。下面介绍 res/目录下的一些子目录，有些是自动创建的，有些则是需要你自己创建的。

- res/drawable/：存放图像（PNG、JPEG，等等）。
- res/layout/：存放基于 XML 的 UI 布局描述。

- res/menu/：存放基于 XML 的菜单描述。
- res/raw/：存放通用的文件（如，包含账号信息的 CSV 文件）。
- res/values/：存放字符串、尺寸值。
- res/xml/：存放通用的 XML 文件。

所有这些文件夹（以及其他资源）都会在本书中提到。

2.3.4 编译结果

在编译项目时（通过 ant 命令或 IDE），编译结果将被放到项目根目录下的 bin/文件夹中，这个文件夹中的情况如下。

- bin/classes/：存放编译后的 Java 类文件。
- bin/classes.dex：存放基于编译后的 Java 类创建的可执行文件。
- bin/*yourapp*.ap_：存放应用程序的资源，打包为 ZIP 文件（这里的 *yourapp* 是你应用程序的名称）。
- bin/ *yourapp*-debug.apk 或 bin/ *yourapp*-unsigned.apk：实际的 Android 应用程序（这里的 *yourapp* 是你应用程序的名称）。

其中，.apk 文件是一个 ZIP 压缩文件，包含.dex 文件、资源的编译版（resources.arsc）、其他未编译的资源（如放在 res/raw/中的资源）以及 AndroidManifest.xml 文件。.apk 文件也是经过数字签名的文件，文件名中的-debug 表示已经使用模拟器认可的调试密钥签名，-unsighed 表示待发布的应用程序（ant release），但还需要使用 jarsigner 及正式密钥进行签名。

2.4 AndroidManifest.xml 文件

描述文件是所有 Android 应用程序的基础，本节就来介绍这个位于项目根目录下的 AndroidManifest.xml 文件。除了在这个文件中声明应用程序中都包含哪些组件（如 Activity、Service，等等），还要指出它们怎么附加到整个 Android 系统中。例如，可以指定某个（某些）Activity 应该出现在设备的主菜单中（主菜单也称为“启动菜单”）。

在创建应用程序的时候，将会自动生成一个基本的 AndroidManifest.xml 文件。对简单的应用程序来说，这个文件中可能只包含一个 Activity。此时，使用自动生成的描述文件可能也没有问题，但有时候也需要对它进行一些小小的改动。另一方面，Android API 演示套件所使用的描述文件则非常庞大，有 1 000 多行。至于你自己开发的应用程序，其大小很可能会介于上述二者之间。

关于描述文件中那些最重要的部分，本书都会在讨论相关的 Android 特性时进行较为详细地

介绍。例如，第29章会详细介绍<service>元素，而那一章的主题是创建服务。就目前而言，只要理解描述文件的作用及其一般性结构就足够了。

2.4.1　一开始是根元素

所有描述文件的根元素都是——当然，没有什么好奇怪的——<manifest>元素：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.search">
...
</manifest>
```

注意命名空间声明。有点奇怪吧？在这个生成的描述文件中，只有特性（attribute）带有命名空间前缀，元素上反而没有（例如，是 manifest 而非 android:manifest）。在 Android 应用程序中，这种做法是有效的，除非 Android 以后会变，否则建议你还是遵守这一约定。

在<manifest>元素中，需要指定的最长的信息就是 package 特性（奇怪的是，也不需要在这个特性前面添加命名空间前缀）。在这里，可以提供作为应用程序“基准”的 Java 包的名字。此后，只要描述文件中其他地方需要类名，都可以用前置的点号来作为这个包的简写。例如，你需要在前面的描述文件中引用 com.commonsware.android.search.Snicklefritz，那么可以只使用.Snicklefritz，因为已经将 com.commonsware.android.search 定义为这个应用程序的包了。

2.4.2　权限、编排和应用程序

在<manifest>元素下面，可以发现下列元素。

- <uses-permission>元素：指出应用程序正常运行所需的权限。
- <permission>元素：为活动和服务声明权限，表示其他应用程序要使用当前应用程序的数据或逻辑必须具有的权限。
- <instrumentation>元素：表示需要在关键系统事件中（例如在为记录和监控而启动某个活动时）需要调用的代码。
- <uses-library>元素：引入可选的 Android 组件，例如地图服务。
- <uses-sdk>元素：表示当前应用程序基于哪个版本的 SDK 构建。
- <application>元素：定义与当前描述文件对应的应用程序的细节信息。

下面来看一个描述文件的例子：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android">
  <uses-permission android:name="android.permission.ACCESS_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_GPS" />
```

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

在这个例子中，描述文件使用<uses-permission>元素指出了当前应用程序需要的一些设备功能。具体来说，这里就是要允许应用程序来确定当前的位置。而<application>元素的内容将用于描述 Activity、Service 及其他各种构成应用程序的组件。

第 28 章将详尽讨论权限。

2.4.3 应用程序总要做点什么

描述文件的核心内容都是通过<application>元素的子元素来表达的。

默认情况下，在创建新的 Android 项目时，其中只包含一个<activity>元素：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.skeleton">
    <application>
        <activity android:name=".Now" android:label="Now">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

这个元素的 android:name 表示实现 Activity 的类，android:label 表示用于显示的 Activity 的名字，而（经常会有）<intent-filter>子元素用于描述显示当前 Activity 在什么情况下被调用。这个看似简单的<activity>元素设定了你的 Activity 将会出现在启动菜单中，因而用户可以选择运行这个 Activity。本书后面将会讨论到，一个项目中可以包含多个 Activity。

描述文件中可以有一或多个<provider>元素，表示 ContentProvider 组件，这个组件可以为你的活动以及经过你授权的当前设备中其他应用程序的活动提供数据。ContentProvider 将数据库或其他数据存储模式封装到一个 API 中，可以供任何应用程序使用。稍后，你会看到如何创建 ContentProvider，以及如何使用你和其他人创建的 ContentProvider。

最后，描述文件中还可以有一或多个<service>元素，表示 Service，也就是可以独立于 Activity 长时间运行的代码段。有关 Service 的最典型例子就是 MP3 播放器。通常，我们希望用户在启动其他 Activity，并且 MP3 播放器的用户界面“被挤掉”的情况下，仍然可以播放音乐。第 29 章和第 30 章将介绍如何创建和使用 Service。

2.4.4　确保最大兼容性

与大多数操作系统一样，Android 也需要打补丁、升级版本和改进。其中某些变化会影响到 SDK；换句话说，将会有一些新的类、方法或参数出现，而在之前的 SDK 中这些都是没有的。

如果要确保应用程序能够在具有某个（或更高）版本的 Android 环境的设备上运行，则需要向描述文件中添加<uses-sdk>元素，它是<manifest>元素的子元素。<uses-sdk>元素有一个名叫 minSdkVersion 的特性，表示你应用程序需要的 SDK 版本。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.search">
  <uses-sdk minSdkVersion="2" />
  ...
</manifest>
```

编写此书之时可能存在的 minSdkVersion 值有：

- 1：Android 1.0 SDK
- 2：Android 1.1 SDK
- 3：Android 1.5 SDK
- 4：Android 1.6 SDK
- 5：Android 2.0 SDK

在省略<uses-sdk>元素的情况下，应用程序的行为与将 minSdkVerion 设置 1 相同。

警告　Android Market似乎要求所有人都必须指定minSdkVersion。如果你想要通过该渠道销售自己的应用程序，别忘了指定适当的<uses-sdk>元素。

设置<uses-sdk>之后，应用程序将只能安装到兼容的设备上。不一定非要指定最新的 SDK；不过，如果你选择的是比较老一点的 SDK，那么你可得要小心一些，以确保应用程序能够在你宣称的所有 SDK 版本中运行。例如，在省略<uses-sdk>的情况下，实际上相当于宣称你的应用程序能够在每个 Android SDK 版本中运行。此时，就必须要通过大量测试来确保这一点。

另外还要注意，Android Market 有一个 bug；为此，你必须将<uses-sdk>作为<manifest>元素的第一个子元素。

2.4.5　版本＝控制

需要特别指出的是，在你通过 Android Market 或其他渠道发布应用程序时，可能还需要为<manifest>元素再添加两个特性：android:versionCode 和 android: versionName。这两个特性对升级应用程序有用。

其中，android:versionName 特性是一个人类容易看懂的标签，是应用程序的版本名或编号。因此，可以使用"3.0"、"System V"、"5000"、"3.1"等任何你觉得合适的值。

而 android:versionCode 特性则是一个整数，表示应用程序的版本号。系统可以使用这个特性来确定你的应用程序是不是比另一个版本更新。所谓更新，指的是“android:versionCode 的值更大”。在发布新版本时，是将实际的版本（即 android:versionName 的值）转换为数字，还是给这个特性的值加 1 完全取决于你。

2.5 模拟器和目标

本节我们花点时间来讨论一下 Android 中“目标”（target）的概念，这个概念不太好理解。在长周期应用程序开发中，特别是在使用 Android 模拟器测试应用程序时，目标是很重要的。

2.5.1 虚拟设备

在使用模拟器之前，需要创建一或多个 AVD。这些虚拟的设备可以用来模仿真实的 Android 设备，如 T-Mobile G1 或 HTC Magic。你可以告诉模拟器使用哪个 AVD，然后模拟器就会将自己装扮成该 AVD 描述的设备。

而在创建 AVD 时，就需要指定目标。目标表示的是 AVD 对应哪一类设备。在本书写作时，有 5 个目标可供选择。

- 1：Android 1.1 设备，例如未更新换代的 T-Mobile G1。
- 2：Android 1.5 设备，不支持 Google Maps。通常是那些移植了自定义 Android 的设备。
- 3：Android 1.5 设备，支持 Google Maps。
- 4：Android 1.6 设备，支持 Google Maps。
- 5：Android 2.0 设备，支持 Google Maps。

提示 使用android list targets命令可以查到可用的API目标。

如果你要开发一个使用 Google Maps 的应用程序，那么就必须将相应 AVD 的目标指定为 3 或更大的值。

可以按照需要创建任意多个 AVD，只要硬盘上有足够的空间就没问题。每个 AVD 都是一个完全不同的设备，因此在一个 AVD 上安装应用程序，不会影响其他 AVD。

要创建 AVD，可以使用 android create avd 命令，可以使用 Eclipse 或使用 AVD 管理器（Android 1.6 中新增的一个 GUI 工具）。只运行 android 命令，不带任何参数，即可启动 AVD 管理器，如图 2-1 所示。在这个界面中，你会看到以前创建的 AVD、用于创建和删除 AVD 的 New

（新建）和 Delete（删除）按钮，以及一个使用选中的 AVD 启动模拟器的 Start（开始）按钮等。

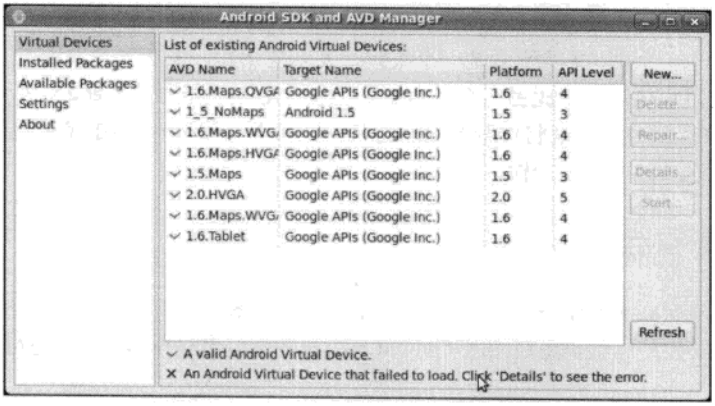


图 2-1　在 AVD 管理器的界面中，可以看到已经创建的 AVD

在通过这个图形用户界面添加 AVD 时（点击窗口中的 New 按钮），需要提供名称、目标 API/Google Maps、SD 卡映像文件以及你希望模拟的屏幕大小（名叫 Skin——皮肤）。图 2-2 是 Create new AVD 对话框。

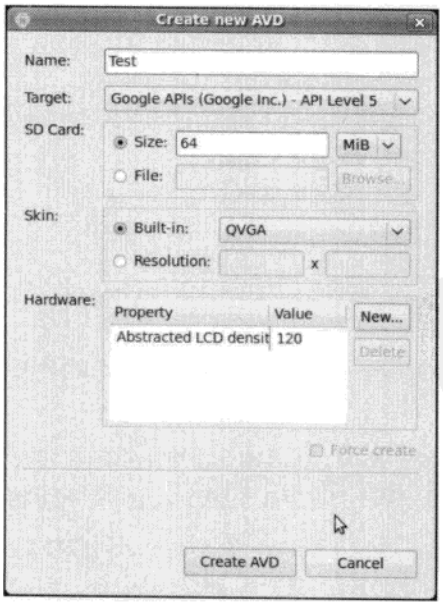


图 2-2　创建新 AVD 的界面

2.5.2 设定目标

在创建了一个新项目后（通过 android create project 或通过 Eclipse），需要指定这个项目针对的是哪一类设备。而这就要用到上一节给出的目标值了。例如，目标设定为 3 的项目，表示在支持 Android 1.5 的设备中运行。最终的应用程序将无法安装到与指定目标不匹配的设备上。

以下是几条与设定目标有关的经验规则。

- 只关注你真正需要的特性。如果你只考虑 Android 1.5 API，那么可能就需要基于 Android 1.5 API 构建应用程序，同时让能够运行该应用程序的设备号码尽量最大。
- 尽可能测试多个潜在的目标。例如，为了支持尽可能多的 Android 设备，你可能会不由自主地把目标设定为 1。虽然想法很好，但你还必须逐个测试目标为 1 的 AVD、目标为 2 的 AVD，以此类推。
- 关注每个 Android 版本发布时的新目标。每个小数点级的版本（如 2.0 或 1.6）都会伴随着一个新的目标值，甚至补丁级的 SDK 变化（如 1.5r1 与 1.5r2）都可能带来新目标值。因此，应该尽可能在新目标值下测试应用程序，有些人可是在新 Android 设备一发布就会抢先试用。
- 不考虑目标，而只在 AVD 上测试，不能取代在硬件上测试。AVD 是方便测试的一种简易环境，覆盖面很广，甚至包括一些还没有出现的硬件。但是，你确实应该至少要在一种 Android 设备上测试自己的应用程序。原因很简单，模拟器的速度与实际设备的速度很可能不一样；一般来说，根据系统环境不同，模拟器的速度可能会快一些，也可能会慢一些。

第3章 简单的应用程序

几乎每一本讲编程语言和环境的书，都会先从众所周知的“Hello World!”示例讲起。对于单纯地验证一下你确实能够编写代码，这个例子足够了。可是，通常的“Hello World!”程序都不具备交互能力（而仅仅是将一个字符串输出到控制台），因此实在是没有什么新意。

本章要演示的项目虽然简单，但却用到了“先进的按钮技术”以及当前的时间，看起来要比俗气的“Hello World!”强多了。

3.1 创建项目

上一章介绍过，要开发任何 Android 应用程序，都必须有一个项目。如果你的开发工具不支持 Android，可以使用 `android create project` 脚本，该脚本位于 SDK 安装目录的 tools 文件夹中。使用 `android create project` 命令时，需要给它传递 API 目标、存放项目的目录、默认 `Activity` 的名称以及程序最终所属的 Java 包：

```
android create project --target 2 \
  --path /path/to/my/project/dir --activity Now \
  --package com.commonsware.android.Now
```

读者也可以在 Apress 公司网站下载本书示例的项目文件（ZIP 格式）。这些项目解压后即可使用，不需要再对它们使用 `android create project` 命令。

在项目的 src/文件夹中，会包含一个与创建项目时指定的 Java 包对应的 Java 式的目录结构（例如，`com.commonsware.android` 对应的目录结构是 src/com/commonsware/android/）。在最内部的文件夹中，可以看到一个预先生成的 Now.java 文件，我们就要在这里创建第一个 `Activity`。这个 `Activity` 中包含一个按钮，用于显示最后一次按下它的时间（如果没有按这个按钮，则显示应用程序启动的时间）。

注意 如果从Apress网站下载源文件，可以直接使用其中的Skeleton/Now项目，这样就不必手工敲代码了。

在编辑器中打开 Now.java，加入下列代码：

```
package com.commonsware.android.skeleton;

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

public class Now extends Activity implements View.OnClickListener {
  Button btn;

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

    btn=new Button(this);
    btn.setOnClickListener(this);
    updateTime();
    setContentView(btn);
  }

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

  private void updateTime() {
    btn.setText(new Date().toString());
  }
}
```

接下来让我们细致剖析。

3.2 剖析 Activity

第一行代码中的包声明与创建项目时指定的包相同。然后，与其他 Java 项目一样，也需要导入代码中会引用到的类。Android 特有的类大多数都在 android 包中。

```
package com.commonsware.android.skeleton;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import java.util.Date;
```

需要注意的是，并不是所有 Java SE 的类都可以在 Android 程序中使用。要知道可以使用哪些类，不可以使用哪些类，请查询 Android 类参考。

```
public class Now extends Activity implements View.OnClickListener {
  Button btn;
```

Now 是公有类，它继承 android.app.Activity 基类。在这个例子中，Now 中保存了一个按钮(btn)。

注意　通过导入的包名可以看出，按钮是一个Android部件（widget），而部件就是你可以在应用程序中使用的用户界面元素。

此外，为了简单起见，我们希望通过 Now 本身来捕获按钮的单击事件，所以同时也让这个类实现了 OnClickListener：

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

  btn=new Button(this);
  btn.setOnClickListener(this);
  updateTime();
  setContentView(btn);
}
```

这个 onCreate()方法会在活动（Now）启动的时候被调用。在这个方法中，首先要做的是与超类联系起来，以便完成这个 Android 活动的初始化。

在我们的实现中，随后又创建了一个按钮的实例（new Button(this)），并告诉它把所有按钮单击事件都发送给 Now 的实例（通过 setOnClickListener()），接着调用 updateTime()方法（稍后介绍），最后再将按钮设置为 Now 的内容视图（通过 setContentView()）。

注意　所有部件都扩展了View基类。一般的界面都不止需要一层视图（view），但在这个例子中，我们只使用一个视图。

关于神奇的 Bundle icicle，我们将留到第 16 章再介绍。此时可以不必深究，只要知道创建任何 Activity 时都需要传入这个参数就好了。

```
public void onClick(View view) {
  updateTime();
}
```

在 Swing 中，单击 JButton 可以生成 ActionEvent 事件对象，这个事件对象会被传递到为该按钮配置的 ActionListener 中。而在 Android 中，单击按钮则会导致在为该按钮配置的 OnClickListenr 中的 onClick()方法方法被调用。而调用这个侦听器方法时，会给它传入触发单击的视图（在这个例子中，就是按钮）。在此，唯一要做的就是调用 updateTime()方法：

```
private void updateTime() {
  btn.setText(new Date().toString());
}
```

在创建活动时（onCreate()），或者在按钮被单击时（onClick()），则使用与 JButton 中功能相同的 setText()将按钮的标签（label）更新为当前时间。

3.3 构建和运行 Activity

要构建这个活动，可以使用内置有 Android 打包工具的集成开发环境（IDE），或者在这个项目的根目录中运行 ant 脚本。然后，按照下列步骤来运行这个活动。

(1) 运行 android 命令并在 AVD 管理器中选择一个 AVD，单击 Start 按钮来启动模拟器。接受 Lauch Options 中默认的选项即可。图 3-1 显示了 Android 的主屏幕。

注意 第一次使用AVD来启动模拟器的时间会比较长，后续的启动速度会有所加快。

(2) 安装程序包（例如，运行 ant install）。

(3) 在模拟器中查看已安装应用程序的列表，找到 Now 应用程序。在图 3-2 中，Now 应用程序位于底部。

(4) 打开 Now 应用程序。应该能够看到如图 3-3 所示的活动界面。

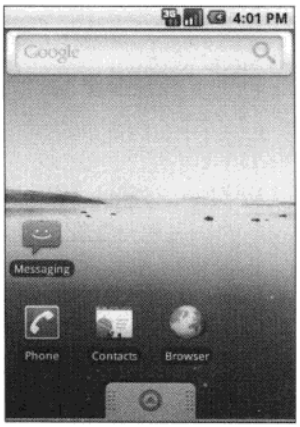


图 3-1 Android 的主屏幕

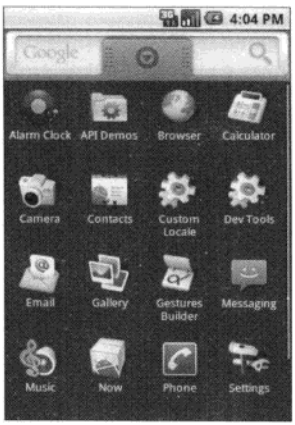


图 3-2 包含 Android 应用程序的启动菜单

图 3-3 通过 Now 应用程序演示 Activity

单击按钮（或者说，单击手机屏幕中的任何地方），按钮标签中的时间都会立即更新。

请注意该标签是水平和垂直方向都居中，这是因为给按钮标题应用了默认样式。当然，我们可以控制这些格式，详细内容将在第 5 章讨论。

在领略了“先进的按钮技术”之后，单击模拟器中的返回按钮，可以返回到启动程序中。

第 4 章

基于 XML 的布局

从技术角度讲，纯粹通过 Java 创建并给 Activity 添加部件是可行的，前一章也是这样做的。但是，更常见的方式则是使用 XML 格式的布局文件。部件的动态实例化一般只针对较为复杂的情况，即在编译时无法知道部件在哪里的情况（例如，需要基于从因特网取得的数据生成一列单选按钮）。

明白这一点之后，接下就该从 XML 入手，学习如何使用它布局 Android 活动了。

4.1 何谓基于 XML 的布局

顾名思义，基于 XML 的布局就是以 XML 格式对部件之间——以及部件与其容器（第 6 章讨论容器）之间——的相互关系进行说明。而且，Android 将基于 XML 的布局视为资源，因此布局文件会被保存在 Android 项目的 res/layout 文件夹中。

每个 XML 文件中包含着一组树形结构的元素，这些元素指定了构成视图层次的部件及其容器的布局。XML 元素的特性（attribute）都是属性（property），用于描述部件的外观或者容器的行为方式。例如，假设 Button 元素有一个特性值 android:textStyle = "bold"，则意味着按钮上的文本应该以粗体样式呈现。

Android 的 SDK 中有一个工具（aapt）会使用布局。这个工具会由 Android 工具链（例如 Eclipse 或 Ant 的 build.xml）自动调用。对于作为开发人员的你来说，必须知道 aapt 负责在项目中生成 R.java 源文件，以便 Java 代码能够直接访问该布局中的布局和部件；本章也将就此给出相应例子。

4.2 为什么使用基于 XML 的布局

可以使用 XML 布局文件实现的任务，大多数情况下也可以通过 Java 代码实现。例如，不在 XML 布局中使用属性，而是调用 setTypeface()方法也可以让按钮的文本呈现为粗体。考虑到使用 XML 布局会增加一个需要维护的文件，所以必须搞清楚为什么要使用它们。

最主要的一个原因，恐怕就是能够为定义视图的工具提供便利，例如 Eclipse IDE 中的 GUI Builder 或专门用于设计 Android GUI 的 DroidDraw。通常，这些 GUI Builder 可以生成 Java 代码而不是 XML。可是，重新读取 UI 定义以进行编辑并不是件轻而易举的事；如果相关数据是以 XML 这样的结构化格式（而非语言代码形式）保存的，这个过程就要容易得多了。此外，保持生成的 XML 定义与硬编码的 Java 代码分离，也可以避免重新生成定义时意外地破坏开发人员自己编写的源文件。使用 XML 格式在这个问题上可以取得某种平衡，既为工具编写者使用提供了方便，也使得开发人员手工编码变得更容易。

此外，将 XML 作为 GUI 定义的格式也是大势所趋。微软的 XAML（Extensible Application Markup Language，可扩展应用程序标记语言）、Adobe 的 Flex 和 Mozilla 的 XUL（XML User Interface Language，XML 用户界面语言）都采用了与 Android 类似的手段：将布局信息放在一个 XML 文件中，将逻辑代码放在源文件中（例如，XUL 与 JavaScript）。还有许多名气不大的 GUI 框架（如 ZK），也都使用 XML 来定义视图。虽然跟风不见得最好，但这样起码可以让熟悉其他 XML 视图描述语言的开发人员更容易掌握 Android 开发。

4.3 举个例子

以下是前一章的示例应用程序中用到的 Button，我们在此将它转换成了 XML 布局文件，可以在 Layouts/NowRedux 示例项目中找到这个文件：

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

在这里，部件的类名 Button 成了 XML 元素的名字。由于 Button 是 Android 提供的部件，因此仅使用类名就可以了。如果要是你自己创建了作为 android.view.View 子类的部件，那么就需要在此提供完整的包名（例如 com.commonsware.android.MyWidget）。

另外，还需要在根元素中声明 Android XML 的命名空间：

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

其余的所有元素都是根元素的子元素，因而都会继承这个命名空间。

因为我们需要在 Java 代码中引用这个按钮，所以还需要通过 android:id 特性为它指定一个标识符。下一节将详细说明 android:id 标识符的概念。

剩下的特性与相应 Button 实例的属性是一一对应的。

- android:text：表示按钮上初始显示的文本（在这个例子中是空字符串）。

- android:layout_width 和 android:layout_height：告诉 Android 按钮的宽度和高度与父元素的关系——在这个例子中，按钮会占据整个屏幕。

第 6 章还将详细介绍这些特性。

由于我们的 Activity 中只有这一个部件，因此 XML 文件中也只需要定义这一个元素。在更复杂的 UI 中，会有更多元素构成树形结构，用来表示部件以及控制它们位置的容器。本书后面各章都将尽可能使用 XML 布局格式，因而还会有很多例子供你学习。

4.4　什么时候加@符号

很多部件及其容器只要出现在 XML 布局文件就行了，不用在 Java 代码中引用。例如，布局文件中经常要使用静态标签（TextView）来表示它应该显示在哪里。像这种元素在 XML 中就不需要使用 android:id 特性为其指定名字。

一句话，对于确实需要在 Java 源代码中引用的元素，就需要为它指定 android:id 特性。

使用“@+id/...”作为 id 值是一种约定，其中“...”表示当前小部件在上下文中唯一的名字。在上一节给出的 XML 布局文件的例子中，Button 部件的标识符是@+id/button。

Android 提供了一些特殊的 android:id 值，格式为“@android:id/...”。在本书后面你会看到很多这样的例子。

4.5　怎样在 Java 中使用布局文件

在精心设计了部件和容器，将 XML 布局文件命名为 main.xml，然后保存到 res/layout 文件夹之后，只要在 Activity 的 onCreate()回调中加入一条语句，即可使用该布局：

```
setContentView(R.layout.main);
```

这同我们前面调用 setContentView()，给它传入一个 View 的子类（即 Button）的实例是一样的。Android（基于我们的布局）构建的 View 可以通过代码生成的 R 类来访问。布局中的所有资源都可以通过 R.layout 加上布局文件的名字来访问；例如，res/layout/main.xml 可以通过 R.layout.main 来访问。

要访问在布局文件中指定了标识符的部件，可以使用 findViewById()并传入部件的标识符。Android 在 R 类中会以 R.id.something（其中 something 是要找的特定部件）的形式生成相应的标识符。这些部件都是 View 的子类，与前一章中创建的 Button 一样。

4.6　把故事讲完

在最初的 Now 项目中，按钮上会显示当前时间，该时间反映了我们按下按钮的时间（如果

没有按下按钮，则是活动初次显示的时间）。在下面这个重写过的项目中（NowRedux），大多数逻辑仍然是相同的。只不过，这一次没有在 Activity 的 onCreate()回调方法中创建 Button 的实例，而是引用 XML 布局文件中的部件：

```
package com.commonsware.android.layouts;

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

public class NowRedux extends Activity
  implements View.OnClickListener {
  Button btn;

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

    setContentView(R.layout.main);

    btn=(Button)findViewById(R.id.button);
    btn.setOnClickListener(this);
    updateTime();
  }

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

  private void updateTime() {
    btn.setText(new Date().toString());
  }
}
```

第一个区别在设置内容视图的地方，这里没有设置在 Java 代码中创建的视图，而是将视图设置为对 XML 布局的引用（setContentView(R.layout.main)）。在重新构建这个包含对布局文件（项目中保存在 res/layout/文件夹内的 main.xml）引用的项目时，也会更新 R.java 源文件。

另一个区别是需要手工创建 Button 实例，因此调用了 findViewById()。由于我们将按钮标识为@+id/button，所以这里在引用按钮时就使用了标识符 R.id.button。这样，就获得了 Button 的实例，接下来就可以设置回调方法以及设置标签了。

这个例子的运行结果与原来的 Now 是一样的，如图 4-1 所示。

图 4-1 NowRedux 示例活动

第5章

使用基本的部件

任何 GUI 工具包都包含一些基本的部件：字段、标签、按钮，等等。Android 的工具包在此也没有什么不同；通过基本的部件可以让我们理解如何在 Android 的 Activity 中使用部件。

5.1 标签

最简单的部件就要数标签（label）了，也就是 Android 中的 TextView。与大多数 GUI 工具包一样，标签中包含的文本无法被用户直接编辑。通常，我们都是使用标签来标识邻近的部件（例如，“名字：”标签会放在要用户填写名字的字段旁边）。

在 Java 中，可以通过创建 TextView 的实例来创建一个标签。不过，更常见的方式则是在 XML 布局文件中创建标签，方法是向布局中添加一个 TextView 元素，并通过 android:text 特性来指定标签的值。如果你打算基于某种条件（例如，国际化语言转换）来切换标签内容，那么最好是在 XML 中使用资源引用，具体内容我们留到第 20 章再讨论。

TextView 也有很多其他与标签有关的属性，简介如下。

- android:typeface：设置标签中字体（例如 monospace，即等宽字体）。
- android:textStyle：设置标签中字体的样式是粗体（bold）、斜体（italic）还是粗斜体（bold_italic）。
- android:textColor：设置标签中文本的颜色，使用 RGB 十六进制格式（例如，红色是 #FF0000）。

例如，在 Basic/Label 项目中，可以找到以下布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="You were expecting something profound?"
  />
```

这个简单的布局，再加上由Android项目构建程序（如android create project）生成的初始 Java 代码，就可以得到如图 5-1 所示的结果。

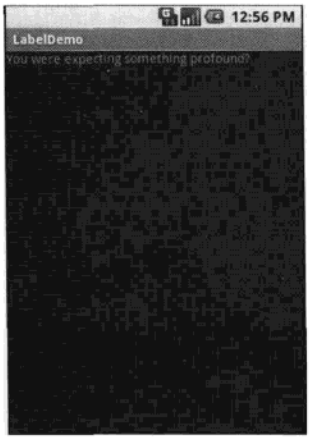


图 5-1 LabelDemo 示例应用程序

5.2 按钮

你在前两章已经看到 Button 小部件的应用了。事实上，Button 是 TextView 的子类，因此前一节讨论的所有属性也都适用于对按钮的文本进行格式化。

不过，Android 1.6 又新增了一个新特性，用于在 Button 上声明单击侦听器。除了定义某些对象（如 Activity）以实现 View.OnClickListener 接口之外，现在我们可以采取简单一些的方法了。

- 在 Activity 上定义某个方法，接收按钮作为 View 参数，返回 void，而且带 public 修饰符。
- 在 XML 布局文件内的 Button 元素中，添加 android:onClick 特性，这个特性的值就是上一步定义的方法名。

例如，如果在 Activity 中定义了下面这个方法：

```
public void someMethod(View theButton) {
 // 实现某些操作的代码
}
```

那么，就可以在 XML 中通过 android:onClick 特性来声明相应的 Button：

```
<Button
 android:onClick="someMethod"
 ...
/>
```

这样，Android 就完全可以把 Button 与单击处理程序关联起来了。

5.3 图像

为在 Activity 中嵌入图像，Android 提供了两个部件：ImageView 和 ImageButton。顾名思义，这两个小部件分别与 TextView 和 Button 在本质上一样，只不过都与图像有关。

这两个部件（在 XML 布局中）都有一个 android:src 特性，用于指定要使用的图像。图像在 Android 中一般被叫做可绘制（drawable）资源，详细讨论请参考第 20 章。此外，还可以使用 setImageURI()方法基于某个 ContentProvider 的 URI 来设置图像。

ImageButton 是 ImageView 的子类，混合了标准的 Button 行为，用于响应单击之类的操作。下面来看一看示例项目 Basic/ImageView 的 main.xml 布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/icon"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:adjustViewBounds="true"
    android:src="@drawable/molecule"
    />
```

而仅使用生成的 Activity 代码的结果如图 5-2 所示（显示了一幅图像）。

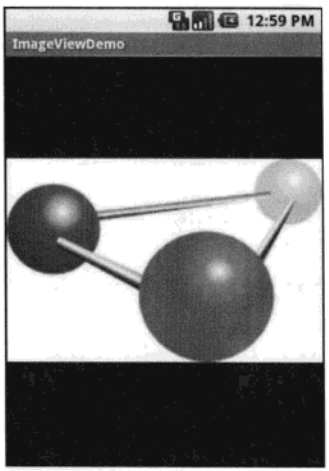


图 5-2　ImageViewDemo 示例应用程序

5.4　字段

除了按钮和标签之外，字段就是大多数 GUI 工具包中的必备元素了。在 Android 中，字段是通过 EditText 部件实现的，EditText 是用作标签的 TextView 的子类。

EditText 具有标准的 TextView 的特性（例如 android:textStyle），还有一些专用于构建字段的特性。

- android:autoText：控制字段是否应该提供自动拼写辅助功能。
- android:capitalize：控制字段是否应该自动大写输入文本的第一个字母（对于英文名字和市名比较有用）。
- android:digits：指定字段只接受某些数字。

- android:singleLine：控制字段是单行输入框还是多行输入框（换句话说，按回车键是将焦点移到下一个部件，还是换行？）。

其中几个特性同样可以在新增的 android:inputType 特性中使用，该特性是在 Android 1.5 新增的“软键盘”模块中添加的（具体内容将在第 10 章讨论）。

例如，Basic/Field 项目包含以下带有 EditText 部件的 XML 布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/field"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:singleLine="false"
  />
```

我们注意到，这里将 android:singleLine 设置为"false"，因此用户就可以键入多行文本了。

在这个项目中，FieldDemo.java 文件向输入字段中随便填充了一些文本内容：

```
package com.commonsware.android.field;

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

public class FieldDemo extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    EditText fld=(EditText)findViewById(R.id.field);
    fld.setText("Licensed under the Apache License, Version 2.0 " +
            "(the \"License\"); you may not use this file " +
            "except in compliance with the License. You may " +
            "obtain a copy of the License at " +
            "http://www.apache.org/licenses/LICENSE-2.0");
  }
}
```

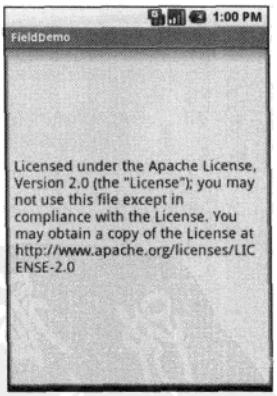


图 5-3 FieldDemo 示例应用程序

在构建并安装到模拟器之后，就可以看到如图 5-3 所示的结果。

另一种形式的字段能够提供自动完成支持，使用户不必输入完整的内容也可以完成字段。这种字段在 Android 中是通过 AutoCompleteTextView 部件来实现的，第 9 章将详细介绍这个部件。

5.5 复选框

经典的复选框有两种状态：选中和未选中。单击复选框可以在这两种状态间切换，以表示选还是不选（例如，“在订单中选定快速发货”）。

在 Android 中，CheckBox 部件就是复选框。这个小部件的基类是 TextView，因此也可以使用 android:textColor 之类的特性对它进行格式化。

而在 Java 代码中，可以调用下列方法。

- isChecked()：确定复选框是否被选中。
- setChecked()：强制选中或取消选中复选框。
- toggle()：像用户单击一样切换复选框的选中和未选中状态。

同样，也可以为复选框状态的变化注册一个侦听器对象（在这里，侦听器是 OnCheckedChangeListener 的实例）。

例如，下面就是 Basic/CheckBox 项目中的一个简单的复选框布局：

```
<?xml version="1.0" encoding="utf-8"?>
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/check"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This checkbox is: unchecked" />
```

对应的 CheckBoxDemo.java 负责获取和改变复选框的行为：

```
public class CheckBoxDemo extends Activity
  implements CompoundButton.OnCheckedChangeListener {
  CheckBox cb;

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

    cb=(CheckBox)findViewById(R.id.check);
    cb.setOnCheckedChangeListener(this);
  }

  public void onCheckedChanged(CompoundButton buttonView,
                               boolean isChecked) {
    if (isChecked) {
      cb.setText("This checkbox is: checked");
    }
    else {
      cb.setText("This checkbox is: unchecked");
    }
  }
}
```

注意，由于实现了 OnCheckedChangeListener 接口，因此这个活动 Activity 提供了自己的侦听器，用于处理复选框状态的变化（通过 cb.setOnCheckedChangeListener(this)）。这个侦听器的回调方法是 onCheckedChanged()，该方法接收状态已经改变的复选框和新状态作为参数。在这里，我们通过更新复选框的文本来反映复选框的变化。

结果如何？单击复选框会立即导致文本更新，如图 5-4 和图 5-5 所示。

图 5-4 CheckBoxDemo 示例应用程序，未选中复选框

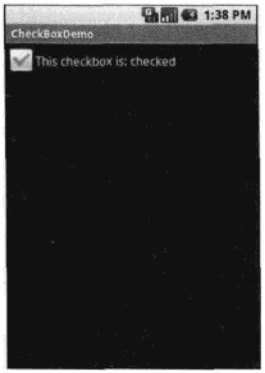


图 5-5 相同的应用程序，选中的了复选框

5.6 单选按钮

与其他工具包对单选按钮的实现一样，Android 的单选按钮也有两种状态（类似复选框），但可以将它们组织成一组，从而让这组单选按钮在任意时刻都只有一个处于选中状态。

RadioButton 与 CheckBox 一样，都继承自 CompoundButton，后者又继承自 TextView。因此，所有标准的 TextView 特性，诸如字体、样式、颜色特性等，都可以用于控制单选按钮的外观。类似地，你可以在 RadioButton 上调用 isChecked()，以检查它是否被选中，或者调用 toggle() 来选择它，等等——与使用 CheckBox 一样。

多数情况下，都需要把 RadioButton 放在一个 RadioGroup 中。RadioGroup 表示是的一组状态受约束的单选按钮，也就是任何时候该组中都只能有一个单选按钮被选中。如果在 XML 布局中为 RadioGroup 指定了 android:id 特性，那么就可以在 Java 代码中使用下列方法访问它。

- check()：通过 ID 选中某个单选按钮（例如 group.check(R.id.radio1)）。
- clearCheck()：取消选中所有单选按钮，结果是组中没有一个单选按钮被选中。
- getCheckedRadioButtonId()：取得当前被选中按钮的 ID（如果没有任何按钮被选中，则返回-1）。

示例应用程序 Basic/RadioButton 的 XML 布局文件中就包含了使用 RadioGroup 封装一组 RadioButton 的例子：

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

```
    <RadioButton android:id="@+id/radio1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Rock" />

    <RadioButton android:id="@+id/radio2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Scissors" />

    <RadioButton android:id="@+id/radio3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Paper" />
</RadioGroup>
```

在使用 Android 生成的基本 Java 代码的情况下，可以通过这个布局得到如图 5-6 所示的结果。

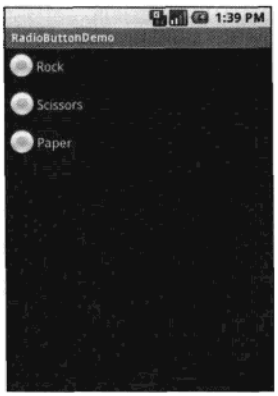


图 5-6　RadioButtonDemo 示例应用程序

在初始状态下，由于单选按钮组进行了初始设置，因此没有一个按钮被选中。如果想预先选中某个单选按钮，可以在 Activity 的 onCreate()回调中调用 RadioButton 的 setChecked()方法，或 RadioGroup 的 check()方法。

5.7　视图

前几节介绍的所有部件都扩展了 View，而这个类也给所有部件提供了很多前面未曾介绍的有用的特性和方法。

5.7.1　特性

View 中最常用的一些特性是下面这些用于控制焦点次序的特性：

- android:nextFocusDown
- android:nextFocusLeft
- android:nextFocusRight
- android:nextFocusUp

另外一个有用的特性是 android:visibility，用于控制部件在初始状态下是否可见。

5.7.2 方法

调用 setEnabled()方法可以启用或停用部件，而调用 isEnabled()方法则可以检测部件是否启用。常见的一种情况就是根据用户对 CheckBox 或 RadioButton 的选择，来禁用某些部件。

通过 requestFocus()方法可以使某个部件获得焦点，而通过 isFocused()可以检测部件是否拥有焦点。在禁用部件的情况下，应该通过这两个方法来确保被禁用的部件不会获得焦点。

为了方便对构成 Activity 视图的部件及其容器进行访问，可以使用下列方法。

- getParent()：查找父部件或容器。
- findViewById()：查找带有特定 ID 的子部件。
- getRootView()：取得视图树的根部件（也就是通过 setContentView()提供给 Activity 的部件）。

5.7.3 颜色

Android 的部件有两种颜色特性。有些类似于 android:background，接受一种颜色值（或者一幅作为背景的图像）。有的则类似于 TextView（或其子类）的 android:textColor，可以接受一个 ColorStateList，通过相应的 Java 访问器（在这里是 setTextColor()）来设置。

通过 ColorStateList 可以为不同条件指定不同颜色。例如，当 TextView 是当前列表中的被选择项时，可以显示一种颜色；而当它未被选择时，可以显示另一种颜色（第 7 章将讨论选择部件）。这种切换通过与 TextView 关联的默认 ColorStateList 来实现。

如果想在 Java 代码中改变 TextView 的颜色，有两种途径。

- 使用 ColorStateList.valueOf()方法，这个方法返回 ColorStateList（所有状态都被认为具有同一种颜色），并以之作为 valueOf()的参数。这个方法是 android:textColor 在 Java 中的等效手段，可以让 TextView 始终都是一种特定的颜色，与环境无关。
- 使用针对不同状态的值创建 ColorStateList，通过构造函数或通过 XML 文档。

第6章

使用容器

容器可以将一批部件（以及其他子容器）组织成特定的结构。如果你想要一个表单，左侧是标签，右侧是字段，那就得使用容器。如果你想让“确定”和“取消”按钮位于表单下方，紧靠在一起，而且与屏幕右侧对齐，那么也得使用容器。即使是从纯 XML 的角度来看，如果有多个部件（不算位于 RadioGroup 中的 RadioButton），也仍然需要一个作为根元素的容器来容纳这些部件。

在布局管理方面，大多数 GUI 工具包通常都会选择使用容器。例如，在 Java/Swing 中，有 BoxLayout 之类的布局管理器以及使用它们的容器（如 Box）。有些工具包，如 XUL 和 Flex，则严格遵循盒模型（box model），认为任何能够想象出来的布局都可以通过正确地组合嵌套的盒子来实现。Android 也通过 LinearLayout 支持盒模型。此外，Android 还提供了一系列容器，这些容器具有不同的布局规则。

本章，我们来讨论一些常用的容器：LinearLayout（盒模型）、RelativeLayout（规则模型）、TableLayout（网格模型）和 ScrollView（用来辅助实现滚动容器的容器）。

6.1 线性布局

LinearLayout 遵循盒模型，即部件或子容器会沿一列或一行对齐，一个接一个排放。这种机制与 Java/Swing 中的 FlowLayout，以及 Flex 和 XUL 中的 vbox 和 hbox 类似。

Flex 和 XUL 中的盒子是主要的布局单元。如果你愿意，当然也可以只使用 LinearLayout 而不使用其他容器。此时，实现你想要的视觉外观，很大程度上只是确定如何嵌套盒子，盒子应该带有哪些特性（例如与其他盒子的对齐关系等）的问题。

6.1.1 LinearLayout 的概念和特性

在配置 LinearLayout 时，可以控制它的 5 个方面：方向、填充模型、权重、对齐和内边距。

1. 方向

方向，设置 LinearLayout 是表示一行还是表示一列。在 XML 布局中，为 LinearLayout 元素添加 android:orientation 特性，将值设置为 horizontal 即表示行，设置为 vertical 即表示列。

在运行时，也可以动态改变方向，这需通过 LinearLayout 来调用 setOrientation()，并传入 HORIZONTAL 或 VERTICAL 参数。

2. 填充模型

我们来想象位于同一行中的部件，例如两个单选按钮。基于它们包含的文本，这两个部件都具有“原本”的大小。但把它们放在一起，它们的宽度之和很可能无法与 Android 设备屏幕的宽度保持匹配（这些屏幕有的大有的小）。你现在面临的问题就是如何处理剩余的空间。

为解决这个问题，位于 LinearLayout 中的所有部件都必须指定 android:layout_width 和 android:layout_height 特性。这两个特性的值可能是以下 3 种形式。

- 具体的大小，例如"125px"表示部件应该占据 125 像素。
- "wrap_content"（即包含内容），表示部件应该保持其本来大小。万一太大了，Android 会自动换行，以便部件适应屏幕空间。
- "fill_parent"（即填充父元素），表示在处理完所有其他部件之后，当前部件应该填满包含它的容器的所有可用空间。

其中，后两种形式最为常用，因为它们不依赖于屏幕尺寸，而且允许 Android 根据可用空间来调整视图布局。

3. 权重

如果有两个部件共享可用空间怎么办？例如，假设一列中有两个多行字段，你想在为其他部件分配完空间之后，让这两个多行字段占据剩余的空间。为此，除了要设置 android:layout_width（行宽）或 android:layout_height（列高）为"fill_parent"之外，还必须设置 android:layout_weight（权重）。

最后一个 android:layout_weight 特性，表示为相应部件分配的空间比例。例如，如果为这两个部件的 android:layout_weight 设置相同的非零值（如 1），那么剩余空间就会由它们平分。如果将一个部件设置为 1，另一个部件设置为 2，则第二个部件占据的空间就是第一个部件的两倍。部件默认的权重值为 0。

使用权重特性的另一种方式是以百分比为单位。以水平布局为例，使用百分比单位的步骤如下。

- 将布局中部件的 android:layout_width 值设置为 0。

- 将布局中部件的 android:layout_weight 值设置为想要的百分比。
- 保证所有这些部件的百分比之和为 100。

4. 重力

默认情况下，LinearLayout 中的所有部件都会沿左上角排列。因此，通过水平的 LinearLayout 创建一行部件时，这行部件会从屏幕左侧开始排列。如果你不想按照这种默认方式排列部件，可以指定一个重力（gravity）值[①]。通过在部件上设置 android:layout_gravity 特性（或在运行时通过部件的 Java 对象调用 setGravity()），可以告诉部件及其容器如何在屏幕中对齐部件。

对于一列部件，常见的重力值有：left、center_horizontal 和 right，分别表示左对齐、水平居中对齐和右对齐。

对于一行部件，默认的对齐方式是让它们包含的文本沿基线（每个字母都恰好位于其上的一条看不见的线）对齐。将重力值设置为 center_vertical 可以将部件与相应行的垂直中点对齐。

5. 内边距

默认情况下，部件相互之间是紧紧靠在一起的。如果想加大部件之间的距离，可以使用 android:padding 特性（或在运行时通过部件的 Java 对象调用 setPadding()）。内边距指的是部件“壳”的边界与部件内容的边界之间应该留出多大距离，如图 6-1 所示。

通过 android:padding 属性可以为部件的四边设置相同的内边距，结果就是部件的内容在空出的区域中居中。如果想让各边的内边距不同，可以使用 android:paddingLeft、android:paddingRight、android:paddingTop 和 android:paddingBottom。内边距的值是具体的尺寸，例如 5px 表示 5 像素的内边距。

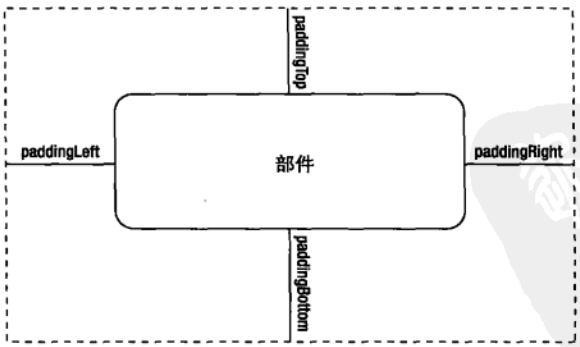


图 6-1　部件的“壳”之间的关系及内边距值

① 这里为保持中英文字面统一，将 gravity 译为了“重力值”；实际上，android:layout_gravity 值的作用就是控制当前部件相对于容器的对齐方式（android:gravity 控制的是当前容器中子元素的对齐方式）。——译者注

如果为某个部件应用了背景（例如，通过 android:background 特性），则该背景将位于部件和内边距后面。假如不想要这个效果，可以创建外边距；外边距会在不扩展部件固有大小的情况下在部件周围增加空间。可以使用 android:layout_marginTop 及相关特性来设置外边距。

6.1.2 LinearLayout 示例

下面我们来看一个示例（Containers/Linear），这个示例展示了如何在 XML 布局文件中和运行时设置 LinearLayout 的属性。以下是布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <RadioGroup android:id="@+id/orientation"
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="5px">
    <RadioButton
      android:id="@+id/horizontal"
      android:text="horizontal" />
    <RadioButton
      android:id="@+id/vertical"
      android:text="vertical" />
  </RadioGroup>
  <RadioGroup android:id="@+id/gravity"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="5px">
    <RadioButton
      android:id="@+id/left"
      android:text="left" />
    <RadioButton
      android:id="@+id/center"
      android:text="center" />
    <RadioButton
      android:id="@+id/right"
      android:text="right" />
  </RadioGroup>
</LinearLayout>
```

在这个布局中，LinearLayout 封装了两组 RadioGroup。由于 RadioGroup 是 LinearLayout 的子类，因此这个示例中嵌套的盒子就好像都是 LinearLayout 容器一样。

上方的 RadioGroup 设置了一行（android:orientation = "horizontal"）RadioButton 部件。而且，这个 RadioGroup 的每一边都有 5px 的内边距，从而将其与另一个 RadioGroup 隔开。它的高度和宽度特性都设置成了 wrap_content，结果单选按钮就会只占据各自所需的空间。

下方的 RadioGroup 中包含一列（android:orientation = "vertical"）3 个 RadioButton 部

件。同样，它的四周也各有5像素的内边距，并保持本来的高度（android:layout_height = "wrap_content"）。可是，由于我们将 android:layout_width 设置成了 fill_parent，结果这一列中的单选按钮就会占据屏幕的整个宽度。

要在运行时根据用户输入来调整这些设置，则需要下列 Java 代码：

```
package com.commonsware.android.linear;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.text.TextWatcher;
import android.widget.LinearLayout;
import android.widget.RadioGroup;
import android.widget.EditText;

public class LinearLayoutDemo extends Activity
  implements RadioGroup.OnCheckedChangeListener {
  RadioGroup orientation;
  RadioGroup gravity;

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

    orientation=(RadioGroup)findViewById(R.id.orientation);
    orientation.setOnCheckedChangeListener(this);
    gravity=(RadioGroup)findViewById(R.id.gravity);
    gravity.setOnCheckedChangeListener(this);
  }

  public void onCheckedChanged(RadioGroup group, int checkedId) {
    switch (checkedId) {
      case R.id.horizontal:
        orientation.setOrientation(LinearLayout.HORIZONTAL);
        break;

      case R.id.vertical:
        orientation.setOrientation(LinearLayout.VERTICAL);
        break;

      case R.id.left:
        gravity.setGravity(Gravity.LEFT);
        break;

      case R.id.center:
        gravity.setGravity(Gravity.CENTER_HORIZONTAL);
        break;

      case R.id.right:
        gravity.setGravity(Gravity.RIGHT);
        break;
    }
  }
}
```

在 onCreate()中，我们找到两个 RadioGroup 容器，并为每一个都注册了侦听器，以便在单

选按钮的状态改变时收到通知（setOnCheckedChangeListener(this)）。由于这个 Activity 实现了 OnCheckedChangeListener，因此 Activity 本身也就是个侦听器了。

在 onCheckedChanged() 方法（侦听器的回调）中，检查了哪个 RadioGroup 的状态发生了变化，继而又根据用户的选择来调整方向。如果是重力按钮组（gravity group），则基于用户的选择调整重力属性。

图 6-2 展示了在模拟器中初次启动这个布局示例的结果。

如果此时单击 vertical 单选按钮，则上方的 RadioGroup 会据以自动调整，如图 6-3 所示。

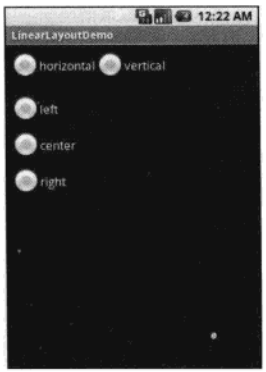


图 6-2 LinearLayoutDemo 示例应用程序（初始启动状态）

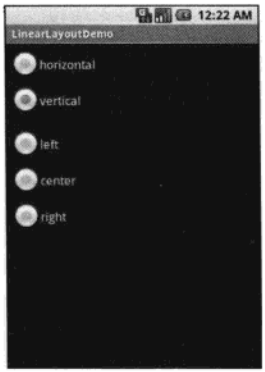


图 6-3 选择 vertical 单选按钮后的同一个示例应用程序

如果选择了 center 或 right 单选按钮，下方的 RadioGroup 则会据以调整，如图 6-4 和图 6-5 所示。

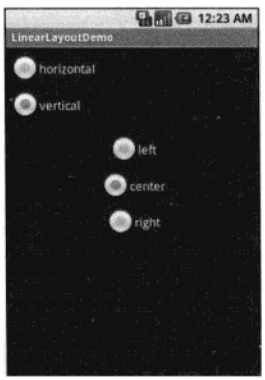


图 6-4 选择 center 单选按钮后的同一个示例应用程序

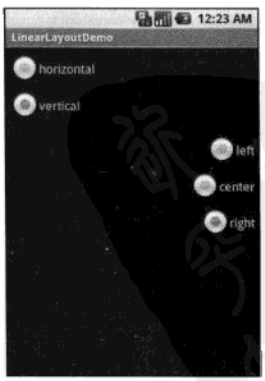


图 6-5 选择 right 单选按钮后的同一个示例应用程序

6.2　相对布局

顾名思义，RelativeLayout 就是相对于容器中的其他部件，或者相对于父容器进行布局。例如，可以将部件 X 放在下方，而且位于部件 Y 的左侧，让部件 Z 的底边与容器的底边对齐，等等。说到这，不禁让人联想到 James Elliot 对使用 Java/Swing 中 RelativeLayout 的讲解[①]。

6.2.1　RelativeLayout 的概念和属性

要使用 RelativeLayout，必须通过某些方式在 XML 布局文件中引用其他部件，还要通过另一些方式来表示这些部件的相对位置。

1. 相对于容器定位

在相对定位中，最简单的关系莫过于参照容器的位置了。

- android:layout_alignParentTop：将部件的顶部与容器的顶部对齐。
- android:layout_alignParentBottom：将部件的底部与容器的底部对齐。
- android:layout_alignParentLeft：将部件的左侧与容器的左侧对齐。
- android:layout_alignParentRight：将部件的右侧与容器的右侧对齐。
- android:layout_centerHorizontal：将部件在容器内水平居中。
- android:layout_centerVertical：将部件在容器内垂直居中。
- android:layout_centerInParent：将部件在容器内水平和垂直居中。

所有这些属性的值都是一个布尔值（true 或 false）。

注意　在按照不同方式对齐时，部件的内边距会计算在内。换句话说，对齐意味着与部件的整体空间对齐（既包括本来的空间也包括内边距）。

---

2. 特性中的关联记号

与 RelativeLayout 相关的另一个特性是一个标识符，用来识别容器中的某个部件。为此，需要：

- 以@+id/...的形式为所有元素添加这个标识符（即 android:id 特性的值）
- 以不带加号的形式（@id/...）来引用其他部件

例如，如果将部件 A 标识为@+id/widget_a，那么部件 B 就可以在它的某个特性中通过标识

---

① James Elliot 是 *Java Swing, Second Edition*（Oreilly,2002）一书的第一作者。——译者注

符@id/widget_a 来引用部件 A。

3. 相对于其他部件定位

有 4 个属性用于控制某个部件相对于其他部件定位。

- android:layout_above：表示当前部件应该位于特性中引用的部件上方。
- android:layout_below: 表示当前部件应该位于特性中引用的部件下方。
- android:layout_toLeftOf：表示当前部件应该位于特性中引用的部件左侧。
- android:layout_toRightOf：表示当前部件应该位于特性中引用的部件右侧。

除了这 4 个属性外，还有 5 个属性用于控制某个部件相对于其他部件对齐。

- android:layout_alignTop：表示当前部件的上沿应该与特性中引用的部件的上沿对齐。
- android:layout_alignBottom：表示当前部件的下沿应该与特性中引用的部件的下沿对齐。
- android:layout_alignLeft：表示当前部件的左边应该与特性中引用的部件的左边对齐。
- android:layout_alignRight：表示当前部件的右边应该与特性中引用的部件的右边对齐。
- android:layout_alignBaseline：表示两个部件应该沿基线对齐（所谓基线，就是文本恰好位于其上的一条看不见的线）。

其中，在对齐标签和字段时 android:layout_alignBaseline 特性很有用，它可以让文本显得更自然。由于字段周围有一圈边框，而标签没有，因此使用 android:layout_alignTop 会将字段盒子的上沿与标签的上沿对齐，而结果就会导致标签中的文本在屏幕上看起来要高于用户在字段中输入的文本。

举例来说，如果想将部件 B 定位在部件 A 的右侧，那么在部件 B 的 XML 元素中，需要包含 android:layout_toRightOf = "@id/widget_a"（假设@id/widget_a 是部件 A 的标识符）。

4. 求值次序

通常，Android 会以单次求值方式来处理 RelativeLayout 定义的布局规则。换句话说，如果 XML 中尚未声明某个部件，就不能引用该部件（例如，通过 android:layout_above）。而这就导致了在定义某些布局时会遇到麻烦。从 Android 1.6 开始，Android 采取二次求值方式来处理布局规则，这样就可以放心地引用那些尚未定义的部件了。

6.2.2 RelativeLayout 示例

现在，我们来看一个典型的“表单”，它包含一个字段、一个标签和一对按钮（分别是 OK 和 Cancel）。以下是 XML 布局，摘自 Containers/Relative 示例项目：

```
<?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:padding="5px">
  <TextView android:id="@+id/label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="URL:"
    android:paddingTop="15px"/>
  <EditText
    android:id="@+id/entry"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/label"
    android:layout_alignBaseline="@id/label"/>
  <Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/entry"
    android:layout_alignRight="@id/entry"
    android:text="OK" />
  <Button
    android:id="@+id/cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/ok"
    android:layout_alignTop="@id/ok"
    android:text="Cancel" />
</RelativeLayout>
```

首先，我们声明了一个 RelativeLayout。在这个例子中，我们希望布局占据整个屏幕的宽度（android:layout_width = "fill_parent"），但只占据必要的高度（android:layout_ height = "wrap_content"），而且要让容器与其内容之间保持 5 像素的内边距（android: padding = "5px"）。

然后，继续定义标签，除了拥有 15 像素的内边距之外（android:padding = "15px"），这个标签很简单（后面还会用到它）。

接着，添加一个字段。我们希望这个字段位于标签的右侧，并且让它们的文本沿基线对齐。同时，这个字段还应该占据布局中这一“行”的剩余空间。为此，需要使用下列 3 个特性：

- android:layout_toRightOf = "@id/label"
- android:layout_alignBaseline = "@id/label"
- android:layout_width = "fill_parent"

如果此时去掉标签上 15 像素的内边距，将会发现字段的上沿会被剪切掉一块。原因在于，容器本身只有 5 像素的内边距，而 android:layout_alignBaseline = "@id/label"只是简单地将标签与字段沿基线对齐。标签在默认情况下是与父容器的上沿对齐的。但是，由于字段四周还有包围它的盒子，所以标签就会比字段矮一些。既然字段要相对于标签的位置定位，而且标签的位置已经确定（因为它在 XML 中先出现），那么字段就会显得过高，而其盒子上沿就会被容器的

内边距剪掉一部分。

在使用 RelativeLayout 来实现预期布局的过程中，你会发现自己经常会碰到类似的问题。

上面 XML 布局中解决这个问题的办法，就是为标签上方设置 15 像素的内边距，这样就可以将标签向下推出足够远的距离，从而确保字段不会被剪掉。

OK 按钮被放在了字段下方（android:layout_below = "@id/entry"），而且其右侧与字段的右侧对齐（android:layout_alignRight = "@id/entry"）。而 Cancel 按钮则被放在了 OK 按钮的左侧（android:layout_toLeft = "@id/ok"），同时其上沿与 OK 按钮的上沿对齐（android:layout_alignTop = "@id/ok"）。

当然，那 15 像素的内边距终归有几分将就的味道。如果是在 Android 1.6 及更高版本中，则可以采取一种更好的解决方案，即在屏幕上方固定一个 EditText，然后再让 TextView 与 EditText 的基线对齐，如下面的例子所示。（在 Android 1.5 及更早版本中，这是不可能的，原因就是前面提到过的单次解析。）

```
<?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:padding="5px">
        <TextView android:id="@+id/label"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="URL:"
                android:layout_alignBaseline="@+id/entry"
                android:layout_alignParentLeft="true"/>
        <EditText
                android:id="@id/entry"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/label"
                android:layout_alignParentTop="true"/>
        <Button
                android:id="@+id/ok"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_below="@id/entry"
                android:layout_alignRight="@id/entry"
                android:text="OK" />
        <Button
                android:id="@+id/cancel"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_toLeftOf="@id/ok"
                android:layout_alignTop="@id/ok"
                android:text="Cancel" />
</RelativeLayout>
```

在不改动自动生成的 Java 代码的情况下，模拟器可以给出如图 6-6 所示的结果。

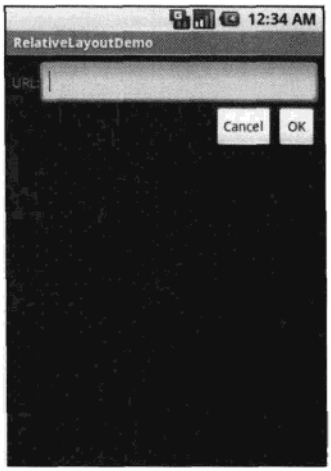


图 6-6　RelativeLayoutDemo 示例应用程序

6.3　表格布局

如果你喜欢 HTML 表格、Excel 式的网格，那么你也会喜欢 Android 的 TableLayout。使用 TableLayout 可以按照你的意图来对部件进行定位。可以控制行数和列数，哪一列可以收缩或伸展以容纳其中的内容，等等。

TableLayout 需要与 TableRow 一起配合使用。TableLayout 控制容器的整体行为，而部件则相继地放入网格中一行一个的 TableRow 容器中。

6.3.1　TableLayout 的概念和特性

要使用表格布局，首先必须理解部件与行和列的关系，以及如何处理位于行外的部件。

1. 在行中添加单元格

表格中的行由你（开发人员）来声明，方法就是在 TableLayout 的每个 TableRow 中放入部件。因此，你可以直接控制表格中有多少行。

列的数量由 Android 控制；你只能间接控制列数。首先，在最长的一行中，至少一个部件一列。因此如果有 3 行：第一行有 2 个部件、第二行有 3 个部件、第三行有 4 个部件，那么这个表格至少会有 4 列。不过，通过设置 android:layout_span 特性（表示部件跨过的列数），可以让一个部件占据多列空间。这个属性与 HTML 单元格的 colspan 特性相似。在下面这个 XML 布局片段中，字段占据了 3 列的空间：

```
<TableRow>
  <TextView android:text="URL:" />
  <EditText
    android:id="@+id/entry"
    android:layout_span="3"/>
</TableRow>
```

一般来说，Android 会从第一个可用的列开始放置部件。对于前面的片段而言，标签将被放到第一列（列 0，因为列从 0 开始计数），而字段会被放到合并后的 3 列（列 1 到列 3）。不过，使用 android:layout_column 特性并指定基于 0 的列索引，也可以把部件放到不同的列中：

```
<TableRow>
  <Button
    android:id="@+id/cancel"
    android:layout_column="2"
    android:text="Cancel" />
  <Button android:id="@+id/ok" android:text="OK" />
</TableRow>
```

在这个 XML 布局片段中，Cancel 按钮会被放到第三列（列 2）。而 OK 按钮则会相应被放入下一列，即第四列中。

2. TableLayout的其他子元素

通常，TableLayout 中只包含 TableRow 作为直接子元素。但实际上也可以在行间放置部件。对于这些放在行外（行间）的部件，TableLayout 会像沿垂直方向布局的 LinearLayout 一样来对齐它们。换句话说，这些部件的宽度会自动设置为 fill_parent，从而占据与最长的行相同的空间。

在这种情况下，一般可以使用一个简单的 View 来设置间距。例如，可以使用<View android:layout_height = "2px" android:background = "#0000FF" />来设置一个与表格同宽的 2 像素高的蓝色分隔条。

3. 扩展、收缩和折叠

默认情况下，每一列都会根据自己包含的部件的原本尺寸确定大小（也包括跨多列的列）。不过，有时候这种默认机制生成的结果不一定理想，因此还需要对列的行为进行更精细地控制。

例如，可以为 TableLayout 添加一个 android:stretchColumns 特性。这个特性的值应该是一个列索引值（当然，还是基于 0 的），或者一个逗号分隔的列索引值列表。然后，这些列就会扩展并占据行中的可用空间。在内容比可用空间窄的情况下，可以考虑设置这个属性。

与此相反，还可以在 TableLayout 中添加 android:shrinkColumns 特性。同样，这个属性的值应该是一个列索引值（当然，还是基于 0 的），或者一个逗号分隔的列索引值列表。而列表中指定的列就会对其包含的内容进行换行处理，以减少列的实际宽度。默认情况下，部件的内容是不换行的。在内容相对比较冗长，可能会将某些列挤到屏幕右侧时，可以考虑使用这个属性。

此外，还可以在 TableLayout 上设置 android:collapseColumns 特性。还是一样，这个属性的值应该是一个列索引值（当然，还是基于 0 的），或者一个逗号分隔的列索引值列表。然后，指定的列在一开始时将被折叠起来，也就是说，它们仍然是表格的一部分，只是不可见而已。在使用编程方式控制表格时，可以通过在 TableLayout 上调用 setColumnCollapsed()来折叠或取消折叠列。可以利用这个属性来让用户自己控制只显示那些他们认为重要的列，而隐藏那些他们认为不重要的列。

当然，在运行时，可以通过 setColumnStretchable()和 setColumnShrinkable()来控制列的扩展与收缩。

6.3.2　TableLayout 示例

下面的 XML 片段组合了前面给出的代码，它通过 TableLayout 实现了与前面使用 RelativeLayout 实现的相同的表单布局，而且在标签/字段与两个按钮之间，还添加了一个分隔带（可以 Containers/Table 项目中找到这个文件）：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="1">
  <TableRow>
    <TextView
        android:text="URL:" />
    <EditText android:id="@+id/entry"
      android:layout_span="3"/>
  </TableRow>
  <View
    android:layout_height="2px"
    android:background="#0000FF" />
  <TableRow>
    <Button android:id="@+id/cancel"
      android:layout_column="2"
      android:text="Cancel" />
    <Button android:id="@+id/ok"
      android:text="OK" />
  </TableRow>
</TableLayout>
```

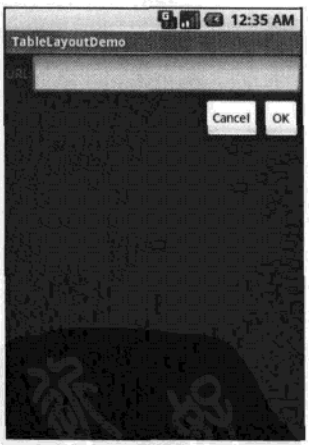


图 6-7　TableLayoutDemo 示例应用程序

在基于生成的 Java 代码编译并在模拟器中运行之后，可以看到如图 6-7 所示的结果。

6.4　滚动

手机屏幕相对比较小，这就要求开发人员要使用某些技巧，以便通过有限的空间展示较多的内容。滚动就是这样一个技巧，也就是说屏幕在某一时刻只显示部分信息，而通过向上或向下滚

动可以显示其余信息。

ScrollView是一个可为其内容提供滚动条的容器。这样，还是使用相同的布局技术，可以把一个需要几屏才能放下的大布局放在一个 ScrollView 中。然后，用户就只能一次看到布局的一部分了。

下面来看一个在 XML 布局文件中使用 ScrollView 的例子（见 Containers/Scroll 项目）：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content">
  <TableLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="0">
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#000000"/>
      <TextView android:text="#000000"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#440000" />
      <TextView android:text="#440000"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#884400" />
      <TextView android:text="#884400"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#aa8844" />
      <TextView android:text="#aa8844"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#ffaa88" />
      <TextView android:text="#ffaa88"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
```

```
        android:layout_height="80px"
        android:background="#ffffaa" />
      <TextView android:text="#ffffaa"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
    <TableRow>
      <View
        android:layout_height="80px"
        android:background="#ffffff" />
      <TextView android:text="#ffffff"
        android:paddingLeft="4px"
        android:layout_gravity="center_vertical" />
    </TableRow>
  </TableLayout>
</ScrollView>
```

要是没有 ScrollView 的话，这个例子中的表格至少要 560 像素（7 行，每行 80 像素）才能放得下。当然，有些设备的屏幕可能会大一些，也可能会小一些。有了 ScrollView 之后，就可以保持表格的原貌，只不过每次只显示一部分罢了。

在 Android 模拟器上第一次看到这个 Activity 的效果，将会如图 6-8 所示。

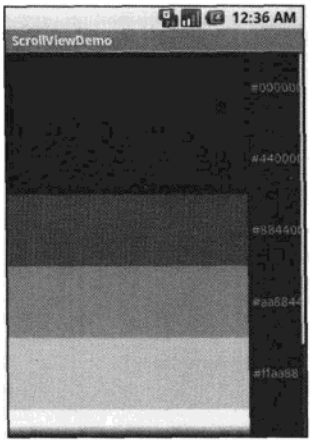


图 6-8　ScrollViewDemo 示例应用程序

注意，画面中只显示出来了前五行和第六行的部分。通过按 D-pad 上的上/下键，可以向上和向下滚动屏幕，从而看到剩余的行。此外，我们注意到内容右侧还被滚动条剪切掉一部分。要么添加一些内边距，要么就想其他办法不要让内容出现这种被剪切掉的情况。

Android 1.5 引入了 HorizontalScrollView，其用法与 ScrollView 类似，只不过是提供水平滚动条。显然，这个容器适用于那些过宽而不是过高的内容。但要注意的是，ScrollView 和 HorizontalScrollView 都不能同时提供两个方向的滚动条，要么垂直，要么水平。

第 7 章

使用选择部件

7

我们在第 5 章曾经介绍过，可以为字段添加一些约束，从而限制可以输入的值（如只允许输入数字）。这种约束可以帮助用户在填写信息时“回答正确”，在使用移动设备小键盘的时候这种作用尤为突出。

当然，说到终极约束，恐怕就是只允许用户选择给定的选项了；单选按钮组（也是在第 5 章介绍的）就是一个例子。经典的 UI 工具包中都会包含列表框、组合框、下拉列表，以及具有类似约束的其他部件。Android 也提供了很多类似的部件，而且还提供了特定于移动设备的部件（例如用于浏览照片的 `Gallery`）。

此外，Android 还提供了一个灵活的框架，用于确定这些部件中包含什么选项。特别地，Android 提供了一个数据适配器框架，为选择列表提供了一个公共接口，涵盖了静态的数组和动态的数据库内容。Selection 视图（展现选项列表的部件）会与一个适配器结合，以提供实际的选项。

7.1 适配器

抽象地讲，适配器可以为多个不相关的 API 提供一个公共的接口。具体到 Android 来说，适配器为基于选择的部件（例如列表框）背后的数据模型提供了一个公共的接口。Java 接口的这种用法非常普遍（例如，Java/Swing 中针对 `JTable` 的模型适配器），而且 Java 也远不是提供这种接口的唯一环境（例如，Flex 的 XML 数据绑定模型接受静态 XML 数据，也接受从 Internet 动态取得的数据）。

Android 的适配器负责为选择部件提供数据源，也负责将单独的数据元素转换为显示在选择部件中的特定视图。适配器的后一种角色似乎有点奇怪，但在实际应用当中，这种转换几乎与其他 GUI 工具包覆盖默认显示行为的机制没有什么区别。例如，在 Java/Swing 中，如果想让一个使用 `JList` 创建的列表框成为一个复选框列表（每一行都由一个复选框和一个标签组成，单击可以切换复选框的状态），就必须调用 `setCellRenderer()`以提供自定义的 `ListCellRenderer`，而这个自定义的 `ListCellRenderer` 又会将列表中的字符串转换成“JCheckBox+JLabel”形式的复合部件。

最简单的适配器是 ArrayAdapter。你所要做的就是为这个类的构造器传入一个 Java 数组或一个 java.util.List 实例，然后就可以得到一个功能齐备的适配器了：

```
String[] items={"this", "is", "a",
                "really", "silly", "list"};
new ArrayAdapter<String>(this,
  android.R.layout.simple_list_item_1, items);
```

ArrayAdapter 接收 3 个参数：

- 要使用的上下文（通常就是当前 Activity 的实例）。
- 要使用的视图的资源 ID（例如以前示例中展示的内置部件的资源 ID）。
- 要实际显示的选项数组或列表。

默认情况下，ArrayAdapter 会在列表中的每个对象上调用 toString()，然后再将得到的字符串分别封装在由提供的资源指定的视图中。在这里，android.R.layout.simple_ list_item_1 就是将作为选项的字符串封装到 TextView 对象中。然后，那些 TextView 对象就会被显示在使用这个适配器的列表、微调框（spinner）或其他部件中。第 8 章将介绍如何创建 Adapter 的实例并重写创建行的行为，以便对列表项的外观进行更全面的控制。

以下是 Android 提供的另外两个适配器。

- CursorAdapter：将光标（通常来自 ContentProvider）转换为可以在选择视图中显示的内容（第 22 章在讲解数据库时将详细介绍 CursorAdapter）。
- SimpleAdapter：转换在 XML 资源中找到的数据。

7.2　列表

Android 中经典的列表框部件叫做 ListView。在你的布局中包含一个 ListView，调用 setAdapter()以提供数据和子视图，然后通过 setOnItemSelectedListener()添加一个侦听器，以便在选项变化时作出反应——这样，你就拥有了一个功能齐备的列表框。

不过，如果你的 Activity 仅涉及一个列表，那么就应该考虑将 Activity 创建为 ListActivity 的子类，而不是仍然以 Activity 为基类。如果主视图中只包含这个列表，甚至都不必提供布局；ListActivity 就可以为你构建一个全屏的列表。如果你确实想自定义布局，只要将 ListView 标识为@android:id/list，ListActivity 就知道哪个部件是这个活动的主列表。

例如，下面是示例项目 Selection/List 中使用的布局文件，其中只包含一个上方带有标签的用以显示选项的列表：

```
<?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/selection"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"/>
    <ListView
      android:id="@android:id/list"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:drawSelectorOnTop="false"
      />
  </LinearLayout>
```

用来配置这个列表及联系列表与标签的 Java 代码如下所示：

```
public class ListViewDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1,
                        items));
    selection=(TextView)findViewById(R.id.selection);
  }

  public void onListItemClick(ListView parent, View v, int position,
                              long id) {
   selection.setText(items[position]);
  }
}
```

继承 ListActivity 之后，就可以通过 setListAdapter()来设置列表适配器——这里提供的 ArrayAdapter 中只封装了一个包含无意义数据的数组。为在列表选项变化时得到通知，重写了 onListItemClick()并根据提供的子视图和位置采取了下一步行动——这里是使用相应位置的文本来更新标签的内容。结果如图 7-1 所示。

ArrayAdapter 的第二个参数是 android.R.layout.simple_list_item_1，用于控制列表中每一行的外观。前面示例中用到的值表示使用标准的 Android 列表行：大字号、较大的内边距和白色文本。

默认情况下，ListView 只能响应单击列表选项的操作。如果你想让列表能够跟踪用户的选择，或者支持多项选择，ListView 也完全支持，只不过要麻烦一点。

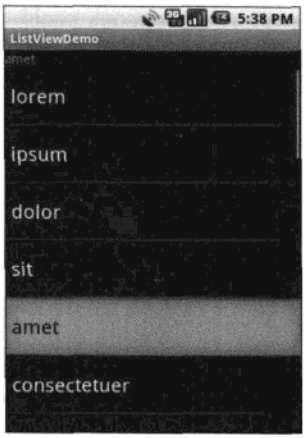


图 7-1　ListViewDemo 示例应用程序

- 在 Java 代码中，基于 ListView 调用 setChoiceMode()设置选择模式，传入 CHOICE_MODE_SINGLE 或 CHOICE_MODE_ MULTIPLE 值作为参数。调用 ListActivity 的 getListView()可以取得这个 ListView。
- 在设置列表行布局时，不再向 ArrayAdapter 构造器中传入 android.R.layout.simple_list_item_1,而是传入 android.R.layout.simple_list_item_single_choice 或 android.R.layout.simple_list_item_multiple_choice，这两个值分别表示单选（图 7-2）和多选（图 7-3）列表。
- 要确定用户选择了哪些选项，在 ListView 上调用 getCheckedItemPositions()。

图 7-2　单选模式

图 7-3　多选模式

7.3 微调控件

在 Android 中，微调控件 Spinner 与其他工具包中的下拉选择器（如 Java/Swing 中的 JComboBox）类似。按下 D-pad 中间的按钮，会弹出一个选择对话框，用户可以从用选择一项。使用微调控件既可以像使用列表一样提供选项，又可以不占用 ListView 那么大的空间，唯一让用户感觉麻烦一点的就是需要多一次点击，或者触摸一下屏幕。

与 ListView 一样，也是使用 setAdapter()来提供数据和视图的适配器，并通过 setOnItemSelectedListener()来为选项添加一个侦听器对象。

如果想控制显示的下拉视图的大小，需要配置适配器而不是 Spinner 部件。为此，要调用 setDropDownViewResource()方法并提供要使用视图的资源 ID。

以下是示例项目 Selection/Spinner 中的 XML 布局文件，这是一个包含 Spinner 的简单视图：

```
<?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/selection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
  <Spinner android:id="@+id/spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:drawSelectorOnTop="true"
  />
</LinearLayout>
```

这个视图与上一节的视图相同，只是使用了 Spinner 而不是 ListView。Spinner 的特性 android:drawSelectorOnTop 用于控制是否在 Spinner UI 右侧的选择器按钮上绘制箭头。

为了填充和使用这个 Spinner，需要下列 Java 代码：

```
public class SpinnerDemo extends Activity
  implements AdapterView.OnItemSelectedListener {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

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

```
    setContentView(R.layout.main);
    selection=(TextView)findViewById(R.id.selection);

    Spinner spin=(Spinner)findViewById(R.id.spinner);
    spin.setOnItemSelectedListener(this);
    ArrayAdapter<String> aa=new ArrayAdapter<String>(this,
                            android.R.layout.simple_spinner_item,
                            items);

    aa.setDropDownViewResource(
      android.R.layout.simple_spinner_dropdown_item);
    spin.setAdapter(aa);
  }

  public void onItemSelected(AdapterView<?> parent,
                             View v, int position, long id) {
    selection.setText(items[position]);
  }

  public void onNothingSelected(AdapterView<?> parent) {
    selection.setText("");
  }
}
```

在这个例子中，我们将 Activity 本身作为了选择侦听器（spin.setOnItemSelectedListener(this)）。之所以可以这样做，是因为当前 Activity 实现了 OnItemSelectedListener 接口。在配置适配器时，不光是使用了随意拼凑起来的数据，而且还为它指定了用于显示下拉列表的资源（通过调用 aa.setDropDownViewResource()）。另外，这里使用 android.R.layout.simple_spinner_item 作为内置的 View 来显示微调控件本身的选项。

最后，还实现了 OnItemSelectedListener 回调方法，该方法基于用户的选择更新标签。图 7-4 和图 7-5 给出了两个屏幕截图。

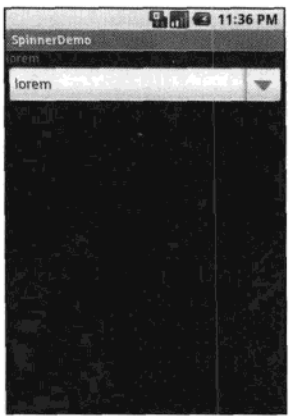


图 7-4　SpinnerDemo 示例应用程序，初始启动状态

图 7-5　同一个应用程序，显示微调控件下拉列表的状态

7.4 网格

顾名思义，GridView 可以为我们提供一个可供选择的二维选项网格。开发人员可以控制网格列的数量和宽度；行的数量是基于适配器提供的选项数，在确保有效显示的条件下动态确定的。

组合起来之后，可以通过下列特性控制列的数量和宽度。

- android:numColumns：指定网格中包含多少列；或者，也可以指定 auto_fit 值，这样 Android 就会根据可用空间大小和下面特性的设置来计算列数。
- android:verticalSpacing 和 android:horizontalSpacing：指定网络中选项之间的垂直和水平间距。
- android:columnWidth：指定每一列的像素宽度。
- android:stretchMode：指定如何处理未被列占用的剩余空间或空白（对于 android:numColumns 特性值为 auto_fit 的情况而言）。这个特性的值可以是 columnWidth，表示让所有列占据剩余空间；也可以是 spacingWidth，表示由列间的空白共享剩余空间。

例如，假设屏幕宽度为 320 像素，而我们把 android:columnWidth 设置为 100px，把 android:horizontalSpacing 设置为 5px，那么 3 列总共会占用 310 像素（3 × 100+2 × 5=310）。在将 android:stretchMode 设置为 columnWidth 的情况下，则 3 列会分别扩展 3-4 像素，以占据剩余的 10 像素空间。而在将 android:stretchMode 设置为 spacingWidth 的情况下，则 2 个空白区域会分别扩展 5 像素，以占据剩余的 10 像素空间。

除此之外，GridView 与其他选择部件的用法都类似：使用 setAdapter()提供数据和子元素的视图，调用 setOnItemSelectedListener()注册选择侦听器，等等。

下面我们来看 Selection/Grid 示例项目中的 XML 布局文件，其中包含着对 GridView 的配置：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TextView
    android:id="@+id/selection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
  <GridView
    android:id="@+id/grid"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:verticalSpacing="35px"
    android:horizontalSpacing="5px"
    android:numColumns="auto_fit"
```

```
    android:columnWidth="100px"
    android:stretchMode="columnWidth"
    android:gravity="center"
    />
</LinearLayout>
```

这个例子中的网格占据了除选择标签所需之外的整个屏幕空间。网格的列数由 Android 根据5像素的水平间距(android:horizontalSpacing = "5px")和100像素的列宽(android:columnWidth = "100px") 来计算确定 (android:numColumns="auto_fit")，同时所有列再共同分摊屏幕中剩余的空间 (android:stretchMode = "columnWidth")。

配置这个 GridView 的 Java 代码如下所示：

```
public class GridDemo extends Activity
  implements AdapterView.OnItemSelectedListener {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

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

    GridView g=(GridView) findViewById(R.id.grid);
    g.setAdapter(new FunnyLookingAdapter(this,
                      android.R.layout.simple_list_item_1,
                      items));
    g.setOnItemSelectedListener(this);
  }

  public void onItemSelected(AdapterView<?> parent, View v,
                             int position, long id) {
    selection.setText(items[position]);
  }

  public void onNothingSelected(AdapterView<?> parent) {
    selection.setText("");
  }

  private class FunnyLookingAdapter extends ArrayAdapter {
    Context ctxt;

    FunnyLookingAdapter(Context ctxt, int resource,
                        String[] items) {
      super(ctxt, resource, items);

      this.ctxt=ctxt;
    }

    public View getView(int position, View convertView,
                        ViewGroup parent) {
      TextView label=(TextView)convertView;
```

```
        if (convertView==null) {
          convertView=new TextView(ctxt);
          label=(TextView)convertView;
        }

        label.setText(items[position]);

        return(convertView);
      }
    }
  }
```

对于网格单元，我们没有像在前一节看到的示例一样继续使用自动生成的 TextView，而是通过子类化 ArrayAdapter 和重写 getView()创建了自己的视图。为了有所区别，这里在 TextView 部件中包含了看起来好玩的（funny-looking）字符串。如果 getView()取得了一个 TextView，则只是重置其文本，否则就创建一个新 TextView 实例并填充该实例。

通过 XML 布局设置 35 像素的垂直空间后（android:verticalSpacing = "35"），网格会溢出模拟器屏幕的边界，如图 7-6 和图 7-7 所示。

图 7-6 GridDemo 示例应用程序，初始启动状态

图 7-7 同一个应用程序，滚动到网格底部

7.5 自动完成字段（至少减少 35%的输入）

可以将 AutoCompleteTextView 看成是 EditText（字段）和 Spinner 的组合。使用自动完成功能时，用户已输入的文本作为前缀筛选条件与候选文本列表进行比较。匹配的结果将显示在一个选择列表中（就是一个 Spinner），这个列表位于字段下方。用户可以输入完整的内容（在没有出

现建议列表的情况下），也可以从出现的列表中选择一项作为字段的值。

AutoCompleteTextView 是 EditText 的子类，因此可以使用相同的特性来配置它的外观。此外，AutoCompleteTextView 还有一个 android:completionThreshold 特性，表示在触发列表筛选功能之前，用户必须输入的最少字符数目。

可以通过 setAdapter()为 AutoCompleteTextView 提供一个适配器，适配器中包含候选值的列表。不过，由于用户可以输入未包含在列表中的任意值，因此 AutoCompleteTextView 不支持选择侦听器。为此，我们可以注册一个 TextWatcher（就像针对 EditText 部件一样），从而在文本变化时也可以收到通知。不管是用户手工输入，还是从下拉列表中选择，都会触发这个事件。

下面是一个熟悉的 XML 布局文件，但其中包含了一个 AutoCompleteTextView 部件（参见 Selection/AutoComplete 示例项目）：

```
<?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/selection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
  <AutoCompleteTextView android:id="@+id/edit"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:completionThreshold="3"/>
</LinearLayout>
```

对应的 Java 代码如下：

```
public class AutoCompleteDemo extends Activity
  implements TextWatcher {
  TextView selection;
  AutoCompleteTextView edit;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    selection=(TextView)findViewById(R.id.selection);
    edit=(AutoCompleteTextView)findViewById(R.id.edit);
    edit.addTextChangedListener(this);
```

```
    edit.setAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_dropdown_item_1line,
                        items));
  }

  public void onTextChanged(CharSequence s, int start, int before,
                            int count) {
    selection.setText(edit.getText());
  }

  public void beforeTextChanged(CharSequence s, int start,
                                int count, int after) {
    // 实现接口需要实现这个方法，但在这里用不到
  }

  public void afterTextChanged(Editable s) {
    // 实现接口需要实现这个方法，但在这里用不到
  }
}
```

这一次，我们的 Activity 实现了 TextWatcher，意味着回调方法变成了 onTextChanged()和 beforeTextChanged()。在此，我们只使用了前一个回调方法，通过它来更新选择标签以匹配 AutoCompleteTextView 的当前内容。

图 7-8、图 7-9 和图 7-10 展示了结果。

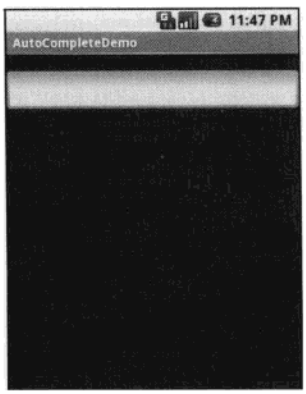


图 7-8 AutoCompleteDemo 示例应用程序，初始启动状态

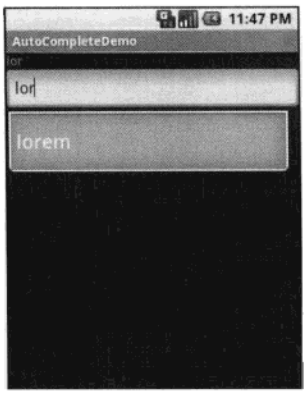


图 7-9 同一个应用程序，输入几个匹配的字符后显示了自动完成下拉列表

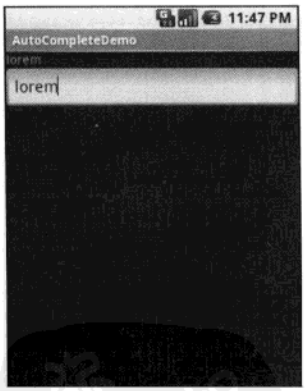


图 7-10 同一个应用程序，选择了自动完成功能提供的值之后

7.6 画廊

Gallery 在传统的 GUI 工具包中是没有的。但实际上，Gallery 不过就是一个水平排布的列表框而已。画廊（gallery）中的项目沿水平方向一个接一个排列，当前选中项会突出显示。在 Android 设备中，用户可以使用 D-pad 的左、右按钮在画廊中的选项间导航。

与 ListView 相比，Gallery 占用的空间更少，但却可以一次显示多个项（假设这些项都比较短）。与 Spinner 相比，Gallery 的优势同样在于可以一次显示多个项。

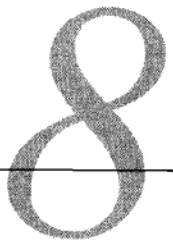
图像预览功能是使用 Gallery 的经典范例。在显示一组照片缩略图的时候，用户可以在 Gallery 中选择想要预览的照片。

从代码方面看，Gallery 与 Spinner 或 GridView 很相似。而在 XML 布局中，还有几个特性可以使用。

- android:spacing：指定列表项之间的间距（像素）。
- android:spinnerSelector：指定使用什么来表示选择，可以是对 Drawable 的引用（参见第 20 章），也可以是#RRGGBB 格式的 RGB 值或类似格式的颜色值。
- android:drawSelectorOnTop：指定是在绘制选中的子元素之前（false）还是之后（true），绘制选择条（或 Drawable）。如果设置为 true，则要保证选择器具有一定的透明度，以便显示出子元素；否则，用户可能会看不到选项。

第8章

使用列表



ListView虽然简单，但却是所有Android应用程序中最重要的一个部件，因为它最常用。无论是查找联系人的电话号码或电子邮件，甚至浏览电子书，相关Activity中都可以见到ListView的身影。当然，如果列表视图中不光是纯文本就更好了。没问题，Android列表可以随你所愿（不过，别忘了移动设备屏幕的限制）。可是，要想让列表看起来更漂亮，额外的工作是必不可少的，而这正是本章我们将要学习的内容。

8.1 初步改进

Android经典的ListView是纯文本形式的列表——实在但不够炫目。把包含一堆文本的数组交给ListView，然后告诉Android以简单的内置布局把文本以列表形式呈现出来，仅此而已。

除了文本之外，列表还可以包含图标、图标和文本、复选框和文本或其他部件。问题的核心就在于要给适配器提供足够的数据，让适配器能够用更丰富的View对象来填充列表的每一行。

例如，假设我们想让ListView的每一行包含图标及相关文本。那么就可以像下面这样构建每一行的布局文件（见FancyLists/Static示例项目）：

```
<?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:orientation="horizontal"
>
  <ImageView
    android:id="@+id/icon"
    android:layout_width="22px"
    android:paddingLeft="2px"
    android:paddingRight="2px"
    android:paddingTop="2px"
    android:layout_height="wrap_content"
    android:src="@drawable/ok"
  />
  <TextView
    android:id="@+id/label"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="44sp"
  />
</LinearLayout>
```

这个布局使用 LinearLayout 作为一行，其中图标位于左侧，文本（相当大的字号）位于右侧。

不过，默认情况下，Android 不会知道我们想把这个布局与 ListView 一块使用。为了将它们联系起来，需要为 Adapter 提供自定义布局的资源 ID：

```
public class StaticDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue",
          "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new ArrayAdapter<String>(this,
                        R.layout.row, R.id.label,
                        items));
    selection=(TextView)findViewById(R.id.selection);
  }

  public void onListItemClick(ListView parent, View v,
                              int position,  long id) {
   selection.setText(items[position]);
  }
}
```

以上代码遵循了前面 ListView 示例的整体结构。主要的区别在于，我们告诉了 ArrayAdapter 使用自定义的布局（R.layout.row），并且在这个自定义布局中使用 TextView 来显示文本内容。

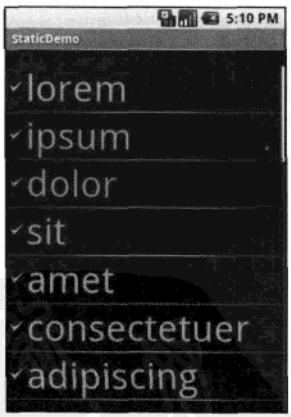


图 8-1　StaticDemo 示例应用程序

注意　在引用布局时（row.xml），要使用R.layout作为XML布局文件的前缀名（R.layout. row）。

结果，就是 ListView 的左侧会带有图标。在这个例子中，所有图标都是相同的，如图 8-1 所示。

8.2　动态列表

对于像前面的例子那样简单的情况，只要为行提供不同的布局就可以了。可是，如果每一行

的布局没有那么简单，就必须另当别论了，例如：

- 并不是每一行都使用相同的布局（例如，某些行包含一行文本，而其他行包含两行文本）。
- 需要在行里面使用部件（例如，不同情况下使用不同的图标）。

在这些情况下，更好的做法是创建相应 Adapter 的子类，重写 getView()方法，再构建不同的行。我们知道，getView()方法负责返回 View，也就是与适配器数据对应的相应位置的行。

下面来看一个例子，这次要在代码中重写 getView()，以便在行中显示不同的图标。具体来说，对于短文本的行，使用一种图标，对于长文本的行，使用另一种图标（见 FancyLists/Dynamic 示例项目）。

```
public class DynamicDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue",
          "purus"};
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new IconicAdapter());
    selection=(TextView)findViewById(R.id.selection);
  }

  public void onListItemClick(ListView parent, View v,
                              int position, long id) {
   selection.setText(items[position]);
  }

  class IconicAdapter extends ArrayAdapter {
    IconicAdapter() {
      super(DynamicDemo.this, R.layout.row, items);
    }

    public View getView(int position, View convertView,
                        ViewGroup parent) {
      LayoutInflater inflater=getLayoutInflater();
      View row=inflater.inflate(R.layout.row, parent, false);
      TextView label=(TextView)row.findViewById(R.id.label);

      label.setText(items[position]);

      ImageView icon=(ImageView)row.findViewById(R.id.icon);

      if (items[position].length()>4) {
        icon.setImageResource(R.drawable.delete);
      }
      else {
        icon.setImageResource(R.drawable.ok);
      }
```

```
            return(row);
        }
    }
}
```

在这个例子中，我们重写了 getView()，以便根据要显示的对象返回行；其中，对象是用 Adapter 中的位置索引来表示的。不过，仔细看一看前面的代码，你会发现引用了 LayoutInflater 类，下面我们简单解释一下为什么要使用这个类。

对于眼下这个例子来说，“inflation”（填装）意味着将 XML 布局说明转换为 XML 表示的实际的 View 结构。想一想看，这个过程还是有点棘手的：接收一个元素，创建特定 View 类的实例，查找特性，将它们转换成对属性设置方法（setter）的调用，然后迭代所有子元素，再如此这般地重复下去。先别急，Android 开发团队的那些家伙为我们想到了这一点，他们把上述所有操作都封装在了一个名为 LayoutInflater 的类中，我们只要使用这个类即可。具体到这个列表而言，我们需要为列表中的每一行都填装一个 View，以便通过简单的 XML 引用来描述行的外观。

于是，我们把创建的 R.layout.row 布局放入其中。这样就得到一个 View 对象，而这个对象实际上就是由 R.layout.row 规定的一个带有 ImageView 和一个 TextView 的 LinearLayout。但这些都不用我们动手，相应的 XML 布局文件和 LayoutInflater 会实现这一切。

结果，我们使用 LayoutInflater 得到了表现每一行的 View。但此时每一行里面“空空如也”，因为静态资源文件不知道要把哪些数据放到哪一行。这时候该我们出手了，必须在返回 View 之前完成对行的自定义和填充，步骤如下。

- 使用对应位置的文本向标签（label）部件中填充内容。
- 然后，看看文本长度是否大于 4 个字符，如果是，则找到 ImageView 图标部件并将其替换成不同的资源。

这样，ListView 中的每一行都会根据列表条目的长短，带有相应的图标，如图 8-2 所示。

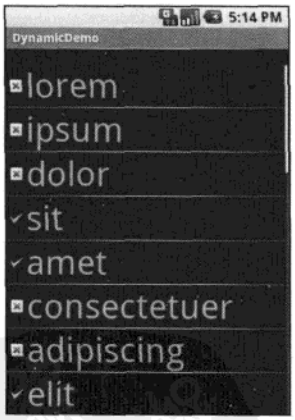


图 8-2　DynamicDemo 应用程序

显然，这个例子还不是那么实用，但同样的技术也可以用于根据任何条件来自定义行，例如根据返回 Cursor 中的其他列来自定义行。

8.3　更好，更快，更强

上一节中的 getView()虽然可以使用，但效率不高。用户每次滚动屏幕，都要创建一批新 View 对象，以填充新出现的行。这当然不好。

想象一下，向下滚动列表，列表反应却很迟钝，这绝不是最好的用户体验。不仅如此，这样

也会消耗更多电能——CPU 每运转一次都要吃掉几分之一格电。再加上垃圾收集程序还要不断把我们额外创建的对象都销毁，又是一笔开销。换句话说，代码效率越低，电池耗尽越快，用户心情越糟。我们都希望用户开心一点，对不？

下面，我们就来介绍一些让 ListView 部件更有效率的技巧。

8.3.1 使用 convertView

实际上，getView()方法还接收了一个参数，也是一个 View，叫做 convertView。有时候，convertView 的值为 null。此时，就需要新建一行的 View（例如，通过“装填”），与前面的例子一样。但是，如果 convertView 不是 null，那就说明之前已经创建了这个对象。创建这个对象是由用户滚动 ListView 引起的。随着新行的出现，Android 会循环使用从另一端滚动出列表的行的视图，以减少重新构建视图的需求。

假设列表中每一行的结构都相同，就可以使用 findViewById()来取得组成行的部件，修改它们的内容，而不是创建新行，然后再通过 getView()返回 convertView 即可。下面，我们来看一看使用 convertView 对前面的例子进行优化之后的 Java 代码（见 FancyLists/ Recycling 示例项目）：

```
public class RecyclingDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue",
          "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new IconicAdapter());
    selection=(TextView)findViewById(R.id.selection);
  }

  public void onListItemClick(ListView parent, View v,
                              int position, long id) {
   selection.setText(items[position]);
  }

  class IconicAdapter extends ArrayAdapter {
    IconicAdapter() {
      super(RecyclingDemo.this, R.layout.row, items);
    }

    public View getView(int position, View convertView,
                        ViewGroup parent) {
      View row=convertView;

      if (row==null) {
```

```
      LayoutInflater inflater=getLayoutInflater();

      row=inflater.inflate(R.layout.row, parent, false);
    }

    TextView label=(TextView)row.findViewById(R.id.label);

    label.setText(items[position]);

    ImageView icon=(ImageView)row.findViewById(R.id.icon);

    if (items[position].length()>4) {
      icon.setImageResource(R.drawable.delete);
    }
    else {
      icon.setImageResource(R.drawable.ok);
    }

    return(row);
  }
 }
}
```

这一次，首先检查 convertView 是否为 null。如果是，则“装填”一行；否则，就重用它。装填内容（图标和文本）的过程与前面一样。这样做的好处是可以避免多余的装填导致的开销。

8.3.2　使用持有者模式

与创建炫目视图如影随形的另一个代价较大的操作，就是调用 findViewById()。这个方法会深入到已装填的行，根据指定的标识符取出相应部件，以便你可以修改部件的内容（例如，修改 TextView 的文本，或修改 ImageView 中的图标）。由于 findViewById()可以从行所在根视图的所有子元素中找到部件，因此可能需要执行相当多的指令，而在重复取得相同部件的情况下则更是如此。

在某些 GUI 工具包中，可以通过在程序代码（这里就是 Java 代码）中整体性地声明复合的 View 对象（如行）来避免这个问题。因为再访问个别的部件，无非就是调用 getter，或者访问字段的问题了。当然，通过 Android 也可以做到这一点，只不过代码可能会繁琐一些。

我们认为理想的方案应该是仍然使用 XML 布局，但又可以缓存行中的关键子部件，也就是只需要查找它们一次即可。说到这里，也就意味着要使用持有者模式（holder pattern）了。所有 View 对象都有 getTag()和 setTag()方法。通过这两个方法可以在任何对象与部件之间建立联系。在持有者模式中，“标签”（tag）用来保存对象，而对象又用来保存要使用的子部件。在将持有者添加到行视图之中后，只要用到了行，就可轻而易举地访问其子部件，而不必再使用 findViewById()了。

好了，下面就来看一个持有者类（见 FancyLists/ViewWrapper 示例项目）：

```
class ViewWrapper {
  View base;
  TextView label=null;
  ImageView icon=null;

  ViewWrapper(View base) {
    this.base=base;
  }

  TextView getLabel() {
    if (label==null) {
      label=(TextView)base.findViewById(R.id.label);
    }

    return(label);
  }

  ImageView getIcon() {
    if (icon==null) {
      icon=(ImageView)base.findViewById(R.id.icon);
    }

    return(icon);
  }
}
```

ViewWrapper 不仅是持有子部件，而且还会被动查找（lazy find）子部件。如果你只是创建包装器，而不需要访问特定的子部件，那么根本就不会调用 findViewById()，因此就不会消耗 CPU 周期。

除此之外，持有者模式还有其他优点。

- 可以将所有基于以部件为单位的类型转换集合于一处，而不是每调用一次 findViewById() 就进行一次类型转换。
- 可以使用它来跟踪与行有关的其他信息，例如，不必探查底层模型就可以知道状态信息。

要使用 ViewWrapper，就要在装填一行时创建它的一个实例，并通过 setTag()将该实例添加到行视图中，如下面重写的 getView()方法所示：

```
public class ViewWrapperDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue",
          "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new IconicAdapter());
    selection=(TextView)findViewById(R.id.selection);
  }
```

```
private String getModel(int position) {
  return(((IconicAdapter)getListAdapter()).getItem(position));
}

public void onListItemClick(ListView parent, View v,
                            int position, long id) {
 selection.setText(getModel(position));
}

class IconicAdapter extends ArrayAdapter<String> {
  IconicAdapter() {
    super(ViewWrapperDemo.this, R.layout.row, items);
  }

  public View getView(int position, View convertView,
                      ViewGroup parent) {
    View row=convertView;
    ViewWrapper wrapper=null;

    if (row==null) {
      LayoutInflater inflater=getLayoutInflater();

      row=inflater.inflate(R.layout.row, parent, false);
      wrapper=new ViewWrapper(row);
      row.setTag(wrapper);
    }
    else {
      wrapper=(ViewWrapper)row.getTag();
    }

    wrapper.getLabel().setText(getModel(position));

    if (getModel(position).length()>4) {
      wrapper.getIcon().setImageResource(R.drawable.delete);
    }
    else {
      wrapper.getIcon().setImageResource(R.drawable.ok);
    }

    return(row);
  }
 }
}
```

与检查 convertView 是否为 null，然后现决定是否创建行视图对象一样，这里也要取得（或创建）相应行的 ViewWrapper。然后，访问行视图的子部件就只需简单地调用包装器的相应方法即可。

8.4　交互式列表

带有图标的列表确实已经相当不错了。但是，能不能让 ListView 部件包含一些具有交互性的子部件呢？换句话说，不光是包含一些被动的 TextView 和 ImageView 部件？举个例子，我想创建一个 RatingBar 部件，好让用户通过单击一组五角星图标来为列表项评级。那么，怎样组织

RatingBar 和文本，才能让用户既可以滚动列表（假设是歌曲列表），又能够直接在列表中对歌曲进行评级呢？

这样做可行，但也有问题。

说这样做可行，是因为交互性部件可以出现在列表的行中。至于问题嘛，就是需要一点小技巧，才能在交互性部件的状态变化（例如在字段中输入了值）时采取某些操作。为此，需要把相应的状态保存起来，因为 RatingBar 部件会在用户滚动 ListView 时循环使用。在循环使用 RatingBar 的时候，必须能够基于用户看到的歌曲来设置 RatingBar 的状态，还需要在状态改变时保存状态，以便特定的行再滚动进视图时恢复其状态。

这个示例的看点在于，默认情况下，RatingBar 绝对不会知道自己要在 ArrayAdapter 中查找什么数据。毕竟，RatingBar 只是 ListView 的行里使用的部件而已。作为开发人员，我们必须告诉列表行它们当前显示的是什么，以便在用户单击五角星来评级时，它们知道修改哪个模型的状态。

下面，我们就来分析一下示例项目 FancyLists/RateList 的代码，看一看上述过程是如何实现的。这个示例仍然使用了前面示例中的一些基本的类，同样，列表中显示的也是一些无意义的文本。此外，被评为最高级别的行文本会全部大写。

```
public class RateListDemo extends ListActivity {
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue",
          "purus"};
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  ArrayList<RowModel> list=new ArrayList<RowModel>();

  for (String s : items) {
    list.add(new RowModel(s));
  }

  setListAdapter(new RatingAdapter(list));
}

private RowModel getModel(int position) {
  return(((RatingAdapter)getListAdapter()).getItem(position));
}

class RatingAdapter extends ArrayAdapter<RowModel> {
  RatingAdapter(ArrayList<RowModel> list) {
    super(RateListDemo.this, R.layout.row, list);
  }

  public View getView(int position, View convertView,
                      ViewGroup parent) {
    View row=convertView;
    ViewWrapper wrapper;
```

```
    RatingBar rate;

    if (row==null) {
      LayoutInflater inflater=getLayoutInflater();

      row=inflater.inflate(R.layout.row, parent, false);
      wrapper=new ViewWrapper(row);
      row.setTag(wrapper);
      rate=wrapper.getRatingBar();

      RatingBar.OnRatingBarChangeListener l=
                  new RatingBar.OnRatingBarChangeListener() {
        public void onRatingChanged(RatingBar ratingBar,
                                    float rating,
                                    boolean fromTouch)  {
          Integer myPosition=(Integer)ratingBar.getTag();
          RowModel model=getModel(myPosition);

          model.rating=rating;

          LinearLayout parent=(LinearLayout)ratingBar.getParent();
          TextView label=(TextView)parent.findViewById(R.id.label);

          label.setText(model.toString());
        }

      };

      rate.setOnRatingBarChangeListener(l);
    }
    else {
      wrapper=(ViewWrapper)row.getTag();
        rate=wrapper.getRatingBar();
      }

      RowModel model=getModel(position);

      wrapper.getLabel().setText(model.toString());
      rate.setTag(new Integer(position));
      rate.setRating(model.rating);

      return(row);
    }
  }

  class RowModel {
    String label;
    float rating=2.0f;

    RowModel(String label) {
      this.label=label;
    }

    public String toString() {
      if (rating>=3.0) {
        return(label.toUpperCase());
      }

      return(label);
    }
  }
}
```

以下是这里的 getView()实现与前一个例子中实现的区别。

- 在仍然使用无意义的 String[]项填充列表行的同时，并没有直接将 String 数组直接放到 ArrayAdapter 中，而是放到了 RowModel 对象的一个列表中。RowModel 是一个可变模型，用于保存无意义的文本和当前选中状态。在真实的系统中，这些可能是通过 Cursor 生成的对象，而对象的属性也会具有更多的实际意义。
- 更新了 onListItemClick()方法，以反映从使用纯 String 模型到使用 RowModel 的变化。
- ArrayAdapter 的子类（RatingAdapter）在其 getView()方法中，检测 convertView 是否为 null。如果是，则通过装填简单的布局来创建新行，同时也会将新创建的行添加到一个 ViewWarpper 中。对于每行的 RatingBar，我们添加了匿名的 onRatingChanged()侦听器，以检查行的标签（getTag()）并将其转换成整数，也就是当前显示的行在 ArrayAdapter 中的位置。使用这个整数值，评级条就可以取得对应行的 RowModel，进而基于评级条中的状态更新模型。当然，在检测到与评级条状态匹配时，也会更新 RatingBar 旁边的文本。
- 还要确保 RatingBar 包含恰当的内容，并包含一个指向当前显示的行在适配器中位置的标签（通过 setTag()）。

布局文件就是 LinearLayout 包含 RatingBar 和 TextView：

```
<?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:orientation="horizontal"
>
  <RatingBar
    android:id="@+id/rate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="3"
    android:stepSize="1"
    android:rating="2" />
  <TextView
    android:id="@+id/label"
    android:paddingLeft="2px"
    android:paddingRight="2px"
    android:paddingTop="2px"
    android:textSize="40sp"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

ViewWrapper 从行的 View 中提取出了 RatingBar 和 TextView：

```
class ViewWrapper {
  View base;
  RatingBar rate=null;
  TextView label=null;

  ViewWrapper(View base) {
    this.base=base;
```

```
  }

  RatingBar getRatingBar() {
    if (rate==null) {
      rate=(RatingBar)base.findViewById(R.id.rate);
    }

    return(rate);
  }

  TextView getLabel() {
    if (label==null) {
      label=(TextView)base.findViewById(R.id.label);
    }

    return(label);
  }
}
```

最终的结果不难想见了，请看图 8-3。至于评定了最高级后将文本转换为全大写的效果，请参见图 8-4。

图 8-3　RateListDemo 应用程序，初始启动状态

图 8-4　相同的应用程序，最高级别的文本全部大写

8.5　可重用列表

上一节展示的评级列表可以运行，但实现起来却非常麻烦。更差劲的是，其中很多琐碎的操作都得不到重用，这个问题在受限的环境下显示尤其扎眼。我们还可以做得更好。

我们认为最佳的方案，应该是创建类似如下所示的布局：

```
<?xml version="1.0" encoding="utf-8"?>
<com.commonsware.android.fancylists.seven.RateListView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@android:id/list"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:drawSelectorOnTop="false"
/>
```

而在代码中，以前与 ListView 有关的几乎所有逻辑，都能够“很好地适用于”这个布局中的 RateListView：

```
String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
        "consectetuer", "adipiscing", "elit", "morbi", "vel",
        "ligula", "vitae", "arcu", "aliquet", "mollis",
        "etiam", "vel", "erat", "placerat", "ante",
        "porttitor", "sodales", "pellentesque", "augue",
        "purus"};

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

  setListAdapter(new ArrayAdapter<String>(this,
                    android.R.layout.simple_list_item_1,
                    items));
}
}
```

回想一下，从本章开始到现在，所有示例都没有修改过 ListView 本身。我们把所有精力都放在了使用适配器、重写 getView()和装填列表行上了。意识到这一点，让我们感觉到了几分挑战性。

如果想让 RateListView 不抛弃简单的 ListAdapter，而且还能正常工作——把评级条按需要放在相应的行上，那么就必须采取一些特殊的举措。其中最重要的，就是要把“原始”的 ListAdapter 封装到其他 ListAdapter 中，而后者知道怎样把评级条放在适当的行上，知道怎样跟踪这些评级条的状态。

首先，要建立一个 ListAdapter 增强另一个 ListAdapter 的模式。下面展示了 AdapterWrapper 的代码，这个类接收一个 ListAdapter 作为参数，并将该接口的所有方法委托给它（见 FancyLists/RateListView 示例项目）：

```
public class AdapterWrapper implements ListAdapter {
  ListAdapter delegate=null;

  public AdapterWrapper(ListAdapter delegate) {
    this.delegate=delegate;
  }

  public int getCount() {
    return(delegate.getCount());
  }
```

```
  public Object getItem(int position) {
    return(delegate.getItem(position));
  }

  public long getItemId(int position) {
    return(delegate.getItemId(position));
  }

  public View getView(int position, View convertView,
                      ViewGroup parent) {
    return(delegate.getView(position, convertView, parent));
  }

  public void registerDataSetObserver(DataSetObserver observer) {
    delegate.registerDataSetObserver(observer);
  }

  public boolean hasStableIds() {
    return(delegate.hasStableIds());
  }

  public boolean isEmpty() {
    return(delegate.isEmpty());
  }

  public int getViewTypeCount() {
    return(delegate.getViewTypeCount());
  }

  public int getItemViewType(int position) {
    return(delegate.getItemViewType(position));
  }

  public void unregisterDataSetObserver(DataSetObserver observer) {
    delegate.unregisterDataSetObserver(observer);
  }

  public boolean areAllItemsEnabled() {
    return(delegate.areAllItemsEnabled());
  }

  public boolean isEnabled(int position) {
    return(delegate.isEnabled(position));
  }
}
```

然后，就可以基于AdapterWrapper来创建名为RateableWrapper的子类，重写默认的getView()方法，但还是让受委托的ListAdapter承担实际的工作：

```
public class RateableWrapper extends AdapterWrapper {
  Context ctxt=null;
  float[] rates=null;
public RateableWrapper(Context ctxt, ListAdapter delegate) {
  super(delegate);

  this.ctxt=ctxt;
  this.rates=new float[delegate.getCount()];

  for (int i=0;i<delegate.getCount();i++) {
    this.rates[i]=2.0f;
```

```
  }
}

public View getView(int position, View convertView,
                    ViewGroup parent) {
  ViewWrapper wrap=null;
  View row=convertView;

  if (convertView==null) {
    LinearLayout layout=new LinearLayout(ctxt);
    RatingBar rate=new RatingBar(ctxt);

    rate.setNumStars(3);
    rate.setStepSize(1.0f);

    View guts=delegate.getView(position, null, parent);

    layout.setOrientation(LinearLayout.HORIZONTAL);

    rate.setLayoutParams(new LinearLayout.LayoutParams(
          LinearLayout.LayoutParams.WRAP_CONTENT,
          LinearLayout.LayoutParams.FILL_PARENT));
    guts.setLayoutParams(new LinearLayout.LayoutParams(
          LinearLayout.LayoutParams.FILL_PARENT,
          LinearLayout.LayoutParams.FILL_PARENT));

    RatingBar.OnRatingBarChangeListener l=
                  new RatingBar.OnRatingBarChangeListener() {
      public void onRatingChanged(RatingBar ratingBar,
                                  float rating,
                                  boolean fromTouch)  {
        rates[(Integer)ratingBar.getTag()]=rating;
      }
    };

    rate.setOnRatingBarChangeListener(l);

    layout.addView(rate);
    layout.addView(guts);

    wrap=new ViewWrapper(layout);
    wrap.setGuts(guts);
    layout.setTag(wrap);

    rate.setTag(new Integer(position));
    rate.setRating(rates[position]);
      row=layout;
    }
    else {
      wrap=(ViewWrapper)convertView.getTag();
      wrap.setGuts(delegate.getView(position, wrap.getGuts(),
                                    parent));
      wrap.getRatingBar().setTag(new Integer(position));
      wrap.getRatingBar().setRating(rates[position]);
    }

    return(row);
  }
}
```

关键在于，我们让 RateableWrapper 包含了评级列表所需的大多数逻辑。这个类负责将评级条放在相应的行上，（在用户点击评级时）负责跟踪评级条的状态。为处理状态，使用了 float[] 与委托配合。

这里 RateableWrapper 对 getView()的实现，不禁让人联想到 RateListDemo。但这里没有使用 LayoutInflater，而是手工构建了 LinearLayout，以保存 RatingBar 和“guts”（或者说，委托创建的任何用来装饰评级条的视图）。LayoutInflater 的用途是基于原始部件构建 View。在这个例子中，除了知道有评级条之外，我们事先并不知道列表行到底是什么样子。不过，其他地方都与 RateListDemo 中的实现相似，如使用了 ViewWrapper，通过 onRatingBarChanged()让评级条能够更新状态，等等。

```
class ViewWrapper {
  ViewGroup base;
  View guts=null;
  RatingBar rate=null;

  ViewWrapper(ViewGroup base) {
    this.base=base;
  }

  RatingBar getRatingBar() {
    if (rate==null) {
      rate=(RatingBar)base.getChildAt(0);
    }

    return(rate);
  }

  void setRatingBar(RatingBar rate) {
    this.rate=rate;
  }

  View getGuts() {
    if (guts==null) {
      guts=base.getChildAt(1);
    }

    return(guts);
  }

  void setGuts(View guts) {
    this.guts=guts;
  }
}
```

有了以上定义和实现，RateListView 就比较简单了：

```
public class RateListView extends ListView {
  public RateListView(Context context) {
    super(context);
  }

  public RateListView(Context context, AttributeSet attrs) {
    super(context, attrs);
  }
```

```
  public RateListView(Context context, AttributeSet attrs,
                      int defStyle) {
    super(context, attrs, defStyle);
  }

  public void setAdapter(ListAdapter adapter) {
    super.setAdapter(new RateableWrapper(getContext(), adapter));
  }
}
```

这里只是创建了 ListView 的子类并重写了 setAdapter() ，以便在 RateableWrapper 中封装接收到的 ListAdapter。

从视觉上说，最终结果与 RateListDemo 非常相似，只不过被评为最高级的文本不会全部大写，如图 8-5 所示。

图 8-5 RateListViewDemo 示例应用程序

区别主要在于可重用性。我们可把 RateListView 打包成独立的 JAR 文件，然后用于任何 Android 项目。从这个角度说，虽然 RateListView 的编写过程比较复杂，但只需编写一次，以后的应用程序开发就异常简单了。

注意 当然，示例中的RateListView还可以用到更多特性，例如以编程方式改变状态（同时更新float[]和实际的RatingBar），在切换RatingBar的状态时，可以执行其他应用程序逻辑（通过某种回调），等等。我们还是把这样或那样可能的增强，留给读者朋友作为练习吧。

8.6　选用其他适配器

对于任何适配器，都可以遵循扩展 ArrayAdapter 并重写 getView()的模式来定义行。不过，CursorAdapter 及其子类对 getView()有默认的实现。

同样，getView()方法会检查接收到的 View 以便重用。如果值为 null，getView()调用 newView()，再调用 bindView()。如果值不是 null，getView()仅调用 bindView()。

如果想要扩展 CursorAdapter（通常是用于显示查询数据库或 ContentProvider 的结果），应该重写 newView()和 bindView()，而不是重写 getView()。结果，就是可以省去 getView()中的 if()测试，而将不同的分支代码组织在相互独立的方法中，类似如下所示：

```
public View newView(Context context, Cursor cursor,
                    ViewGroup parent) {
  LayoutInflater inflater=context.getLayoutInflater();
  View row=inflater.inflate(R.layout.row, null);
  ViewWrapper wrapper=new ViewWrapper(row);

  row.setTag(wrapper);

  return(row);
}

public void bindView(View row, Context context, Cursor cursor) {
  ViewWrapper wrapper=(ViewWrapper)row.getTag();

  // 基于 Cursor 填充行的逻辑代码
}
```

第 22 章将详细讨论 Cursor。

第9章 高级部件和容器

到目前为止，我们介绍的部件和容器不仅可以在多数（同类或同风格的）GUI工具包中找到，同时也是构建Web、桌面或手机GUI应用程序的常用组件。本章将要介绍的部件和容器使用率相对低一点，但用处却一点也不小。

9.1 选择日期和时间

在手机这种输入相对受限的设备中，一些预置输入内容的部件将会为用户提供极大方便。有了这些部件，不仅可以减少键盘输入或屏幕点击，更重要的是可以减少输入错误（例如，在要求输入数字的地方输入了字母）。

第5章曾经介绍过，EditText适合输入数字和文本。此外，Android还支持一些能帮助用户输入日期和时间的部件（DatePicker和TimePicker）和对话框（DatePickerDialog和TimePickerDialog）。

DatePicker和DatePickerDialog可以让用户以年、月、日的形式选择开始日期；其中，对于月份来说，0表示1月，11表示12月。可以给这两个部件提供一个回调对象（OnDateChangedListener或OnDateSetListener），以便获悉用户选择的新日期。必要的时候，需要将这些日期值保存起来——特别是在使用对话框的情况下，因为之后无法再取得用户选定的日期。

类似地，TimePicker和TimePickerDialog可以让用户设置初始时间，时间构成是小时（0～23）和分钟（0～59）。可以设置时间的格式是带有AM/PM的12小时制，还有24小时制（在美国这意味着“军用时间”，而在其他国家和地区则是“正常的24小时制”）。同样也可以为它们提供一个回调对象（OnTimeChangedListener或OnTimeSetListener），从而获悉用户选择的新时间——由小时和分钟组成。

下面我们来看一看Fancy/Chrono示例项目，项目的简单布局中包含一个标签和两个按钮，点击按钮可以打开对话框风格的日期和时间选择对话框：

```
<?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/dateAndTime"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      />
    <Button android:id="@+id/dateBtn"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Set the Date"
      />
    <Button android:id="@+id/timeBtn"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Set the Time"
      />
  </LinearLayout>
```

当然，最值得一看的还是 Java 源代码：

```
public class ChronoDemo extends Activity {
  DateFormat fmtDateAndTime=DateFormat.getDateTimeInstance();
  TextView dateAndTimeLabel;
  Calendar dateAndTime=Calendar.getInstance();
  DatePickerDialog.OnDateSetListener d=new DatePickerDialog.OnDateSetListener() {
    public void onDateSet(DatePicker view, int year, int monthOfYear,
                int dayOfMonth) {
      dateAndTime.set(Calendar.YEAR, year);
      dateAndTime.set(Calendar.MONTH, monthOfYear);
      dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth);
      updateLabel();
    }
  };
  TimePickerDialog.OnTimeSetListener t=new TimePickerDialog.OnTimeSetListener() {
    public void onTimeSet(TimePicker view, int hourOfDay,
                          int minute) {
      dateAndTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
      dateAndTime.set(Calendar.MINUTE, minute);
      updateLabel();
    }
  };

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

    Button btn=(Button)findViewById(R.id.dateBtn);
    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        new DatePickerDialog(ChronoDemo.this,
            d,
            dateAndTime.get(Calendar.YEAR),
            dateAndTime.get(Calendar.MONTH),
            dateAndTime.get(Calendar.DAY_OF_MONTH)).show();
      }
    });
```

```
    btn=(Button)findViewById(R.id.timeBtn);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        new TimePickerDialog(ChronoDemo.this,
                    t,
                    dateAndTime.get(Calendar.HOUR_OF_DAY),
                    dateAndTime.get(Calendar.MINUTE),
                    true).show();
      }
    });

    dateAndTimeLabel=(TextView)findViewById(R.id.dateAndTime);

    updateLabel();
  }

  private void updateLabel() {
    dateAndTimeLabel.setText(fmtDateAndTime
                             .format(dateAndTime.getTime()));
  }
}
```

这个 Activity 中的模型是一个 Calendar 的实例，初始状态下显示当前日期和时间。而 DateFormat 则用于格式化日期和时间，以便显示在视图当中。在 updateLabel()方法中，我们取得了当前的 Calendar，经过格式化又将其放到 TextView 中。

每个按钮都接收到一个 OnClickListener 回调对象。单击按钮后，会显示 DatePickerDialog 或 TimePickerDialog。以 DatePickerDialog 为例，我们为它传入了 OnDateSetListener 回调对象，在用户选择了新日期（年、月、日）后更新 Calendar。此前，我们会从 Calendar 中取得当前日期并显示在对话框中。对于 TimePickerDialog 来说，则是通过 OnTimeSetListener 回调对象将用户选择的时间作为 Calendar 的时间，而 true 表示使用 24 小时制。

将以上布局和代码综合起来得到的最终 Activity 如图 9-1、图 9-2 和图 9-3 所示。

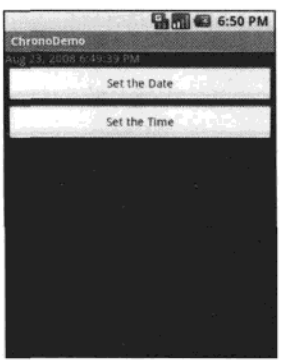


图 9-1 ChronoDemo 示例应用程序，初始启动状态

图 9-2 同一个应用程序，显示日期选择对话框

图 9-3 同一个应用程序，显示时间选择对话框

9.2　时钟

如果你只想显示时间，而不是让用户输入时间，那么可以使用 DigitalClock 或 AnalogClock 部件。这两个部件非常简单易用，而且它们的值会随时间自动更新。我们所要做的，就是把它们放到自己的布局当中。

例如，在示例应用程序 Fancy/Clocks 中，其 XML 布局文件就包含了 DigitalClock 和 AnalogClock 部件：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <AnalogClock android:id="@+id/analog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_alignParentTop="true"
    />
  <DigitalClock android:id="@+id/digital"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_below="@id/analog"
    />
</RelativeLayout>
```

除了一些自动生成的元素之外，无需编写其他 Java 代码，即可完成这个项目并得到如图 9-4 所示的 Activity。

要了解关于计时器的其他应用，建议参考 Chronometer。在 Chronometer 中，可以通过 start() 和 stop()来设置开始和停止计时，而且可以修改显示文本的格式化字符串。图 9-5 展示了这个示例。

图 9-4　ClocksDemo 示例应用程序

图 9-5　Android 2.0 SDK 中的 Views / Chronometer API 示例

9.3 进度条

如果要执行一次长时间的操作，那么应该为用户做两件事：

- 使用后台线程；
- 不断向用户报告进度，以免用户对程序是否正常运行产生怀疑。

让用户了解进度的最典型方式，就是向用户显示一个进度条，或者一个动态浏览图示（类似多数 Web 浏览器右上角的那个动画图像）。在 Android 中，可以通过 ProgressBar 部件来实现这一功能。

ProgressBar 部件用于跟踪进度，以整数定义，0 表示进度尚未开始。使用 setMax()可以定义进度最终完成时的最大值。默认情况下，ProgressBar 从 0 开始；不过，使用 setProgress()也可以设置其他位置作为起点。

如果想让进度条不确定地显示进度，可以通过 setIndeterminate()启用不确定模式（传入 true）。

在 Java 代码中，可以明确地设置已经完成的进度量（通过 setProgress()），也可以基于当前进度量设置进度增量（通过 incrementProgressBy()）。要想知道当前进度，可以使用 getProgress()。

由于 ProgressBar 与线程联系紧密（需要一个后台进程以新进度信息来更新 UI 线程），因此我们将在第 15 章再详细展示其应用。

9.4 滑动选择

SeekBar 是 ProgressBar 的子类。ProgressBar 是输出部件，告诉用户进度已经完成了多少。而 SeekBar 是输入部件，用户可以通过它选择一定范围内的值，如图 9-6 所示。

通过拖动滑块或单击滑块左右的滑动条，都可以重新定位滑块的位置。滑块的移动表示值的变化。值的变化范围可以从 0 到某个设定的最大值（默认为 100），这个最大值可以使用 setMax()来设置。要想了解当前位置，可以使用 getProgress()，或者由通过 setOnSeekBarChangeListener()注册的侦听器来告诉你用户改变了滑块的位置。

第 8 章中介绍的 RatingBar 的例子可以作为参考。

图 9-6 Android 2.0 SDK 中的 Views/SeekBar API 示例应用程序

9.5　选项卡

Android的开发理念就是保持Activity简单明了。如果要显示的信息过多，虽然有滚动条但仍然不方便，那么你可能就会想到通过Intent来启动另一个Activity了（第18章将详细介绍Intent）。可是，这样做毕竟有点兴师动众的味道。况且，某些情况下也确实仅需要通过简单的操作来显示较多的信息。

在传统的UI设计中，我们可以使用选项卡（例如Java/Swing中的JTabbedPane）来盛放较多信息。而在Android中，则可以使用TabHost，使用方法也类似。使用TabHost时，Activity界面的一部分是选项卡，单击选项卡就会切换到视图的另一部分并显示其他内容。举例来说，可以让用户在一个选项卡中输入位置，而在另一个选项卡中显示包含该位置的地图。

有的GUI工具包提到选项卡时，把它描述成用户单击可以切换视图的容器。另一些工具包则认为选项卡是类似按钮的元素与显示内容的元素的结合体，选择哪个按钮就显示哪些内容。Android中的“选项卡按钮”和“选项卡内容”是不同的部件，本节对它们就使用这两个称呼。

9.5.1　构建

要构建选项卡视图，需要用到下列几个容器和部件。

- TabHost用于容纳选项卡按钮和选项卡内容。
- TabWidget用于容纳选项卡按钮，每个按钮由文本标签及可选的图标组成。
- FrameLayout用于容纳选项卡内容，每块内容都是FrameLayout的一个子类。

说起来，这倒有点类似于Mozilla的XUL；例如，XUL中的tabbox元素对应着Android中的TabHost，同样地，tabs元素对应TabWidget，而tabpanels元素对应FrameLayout。

9.5.2　规则

要想让前述3个组件在一起好好工作，必须遵守几条规则——至少在Android的这个里程碑版本中是这样。

- 必须将TabWidget的android:id设置为@android:id/tabs。
- 如果你想使用TabActivity，则必须将TabHost的android:id设置为@android:id/tabhost。

与ListActivity一样，TabActivity将一种常见的UI模式（一个Activity包含全部选项卡）封装进了一个基于模式的Activity子类当中。没有必要非得使用TabActivity，即使是一个简单的Activity照样可以使用选项卡。

例如，以下就是Fancy/Tab中对应于选项卡式Activity的布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/tabhost"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TabWidget android:id="@android:id/tabs"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    />
    <FrameLayout android:id="@android:id/tabcontent"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
      <AnalogClock android:id="@+id/tab1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_centerHorizontal="true"
      />
      <Button android:id="@+id/tab2"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="A semi-random button"
      />
    </FrameLayout>
  </LinearLayout>
</TabHost>
```

要注意的是，TabWidget 和 FrameLayout 是 TabHost 的直接子类，而 FrameLayout 本身也有一些子类可以表示各种选项卡。在这个例子中，有两个选项卡：时钟和按钮。在更复杂一些的情况下，这些选项卡可能会是某种容器（例如 LinearLayout），容器中再包含相应内容。

9.5.3 使用

Java 代码需要告诉 TabHost 哪个视图表示选项卡内容，以及选项卡按钮的外观如何。这些信息都需要通过 TabSpec 对象来传达；而 TabSpec 对象要通过 newTabSpec()来创建，然后进行相应设置，最后再以适当的顺序添加到 TabHost 当中。

TabSpec 有两个重要的方法。

- setContent()：表示选项卡中包含什么内容，一般需要传入相应视图的 android:id。
- setIndicator()：设置选项卡按钮的标题；有意思的是，可以通过这个方法将一个 Drawable 设置为选项卡按钮的图标。

注意，所谓的选项卡“指示器”（indicator）实际上完全可以是一个视图——如果除了一个简单的标签和可选的图标之外，你还需要实现更多控制的话。

还要注意一点，就是在配置任何 TabSpec 对象前，必须在 TabHost 上调用 setup()方法。但在以 TabActivity 作为活动基类的情况下，则不需要调用 setup()方法。

好了，下面来看一看代码吧，这些代码为前面布局示例中的选项卡赋予了生命：

```
package com.commonsware.android.fancy;

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

public class TabDemo extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    TabHost tabs=(TabHost)findViewById(R.id.tabhost);

    tabs.setup();

    TabHost.TabSpec spec=tabs.newTabSpec("tag1");

    spec.setContent(R.id.tab1);
    spec.setIndicator("Clock");
    tabs.addTab(spec);

    spec=tabs.newTabSpec("tag2");
    spec.setContent(R.id.tab2);
    spec.setIndicator("Button");
    tabs.addTab(spec);
  }
}
```

我们先通过 findViewById()方法找到 TabHost，然后再通过它调用 setup()。然后，通过向 newTabSpec()传递一个假定的名称取得一个 TabSpec。有了 spec 之后，再调用 setContent()和 setIndicator()，接着再通过 TabHost 调用 addTab()，从而注册选项卡以供使用。最后，如果想设置默认显示的选项卡，则可以调用 setCurrentTab()，并传入基于 0 的选项卡索引。

图 9-7 和图 9-8 展示了目前的结果。

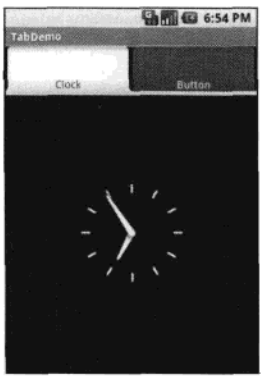


图 9-7　TabDemo 示例应用程序，显示第一个选项卡

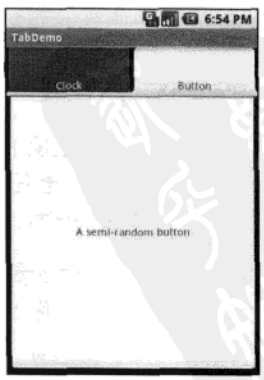


图 9-8　同一个应用程序，显示第二个选项卡

9.5.4 增强

TabWidget 部件用于在编译时定义选项卡。但有时候，我们需要在运行时向 Activity 中添加选项卡。例如在一个电子邮件客户端里，为了便于切换，需要将每封邮件都显示在各自的选项卡中。在这种情况下，我们不知道运行时总共需要多少个选项卡，也不知道每个选项卡中都会包含什么内容，因为打开哪封电子邮件完全取决于用户。好在，Android 也支持在运行时动态地添加选项卡。

在运行时动态地添加选项卡与上一节描述的在编译时添加选项卡非常相似，区别仅在于传递给 setContent()的是 TabHost.TabContentFactory 的实例。这个实例是一个以后会被调用的回调。你要提供 createTabContent()的实现，并通过它构建并返回相应的 View 作为选项卡的内容。

下面我们来看一个例子（Fancy/DynamicTab）。首先是活动的 XML 布局文件，它设置了选项卡视图，定义了一个选项卡和一个按钮：

```
<?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">
  <TabHost android:id="@+id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TabWidget android:id="@android:id/tabs"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
    />
    <FrameLayout android:id="@android:id/tabcontent"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:paddingTop="62px">
      <Button android:id="@+id/buttontab"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="A semi-random button"
      />
    </FrameLayout>
  </TabHost>
</LinearLayout>
```

我们打算让用户单击这个按钮时会添加一个新选项卡。以下就是实现这个功能的 Java 代码：

```
public class DynamicTabDemo extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    final TabHost tabs=(TabHost)findViewById(R.id.tabhost);

    tabs.setup();

    TabHost.TabSpec spec=tabs.newTabSpec("buttontab");
    spec.setContent(R.id.buttontab);
```

```
        spec.setIndicator("Button");
        tabs.addTab(spec);

        Button btn=(Button)tabs.getCurrentView().findViewById(R.id.buttontab);

        btn.setOnClickListener(new View.OnClickListener() {
          public void onClick(View view) {
            TabHost.TabSpec spec=tabs.newTabSpec("tag1");

            spec.setContent(new TabHost.TabContentFactory() {
              public View createTabContent(String tag) {
                return(new AnalogClock(DynamicTabDemo.this));
              }
            });
            spec.setIndicator("Clock");
            tabs.addTab(spec);
          }
        });
      }
    }
```

在按钮的 setOnClickListener()的回调中，我们创建了一个 TabHost.TabSpec 对象并给它添加了一个匿名的 TabHost.TabContentFactory。这个工厂（factory）继而又会返回选项卡中使用的 View——在这个例子中是一个 AnalogClock。实际构建选项卡中 View 的逻辑会相对复杂一些，例如会涉及使用 LayoutInflater 基于 XML 布局来构建视图。

开始的时候，Activity 刚一启动，只会看到一个选项卡，如图 9-9 所示。图 9-10 中则包含了 3 个选项卡。

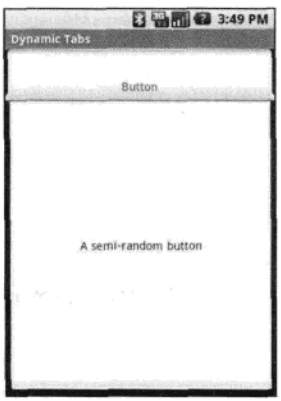


图 9-9　DynamicTab 示例应用程序，开始时只有一个选项卡

图 9-10　DynamicTab 示例应用程序，动态创建了 3 个选项卡

9.5.5　Intent 和 View

在前面的例子中，每个选项卡的内容都是一个 View（如 Button）。使用 View 来填充选项卡

当然简单，也很直观，但却不是唯一的选择。你还可以通过 Intent 来为应用程序添加另一个 Activity。

通过 Intent，可以指定你想要做什么，可以告诉 Android 去找哪些应用程序来执行你的指令。一般来说，结果就是 Activity 遍地开花。比如说，你只要在 Android 启动菜单中启动一个应用程序，控制台就会创建一个 Intent，而 Android 也会打开与该 Intent 关联的 Activity。关于 Intent 的完整介绍，以及怎么将 Activity 放到选项卡中，请参见第 18 章。

9.6 翻转

有时候，你想要的只是选项卡的整体效果（一次只显示某些 View），但并不想真地去实现选项卡的 UI 界面。或许是觉得选项卡占用的屏幕空间太多，或许是想基于手势操作或摇晃设备来切换视角，抑或只是想让人觉得有几分新意。

幸好，我们有 ViewFlipper 容器，可以实现基于选项卡的视图翻转，但使用方式与传统的选项卡又有不同。

ViewFlipper 继承自 FrameLayout，其使用方法与描述 TabWidget 内部结构的方法相同。但不同的是，ViewFlipper 开始的时候只会显示一个子视图。可以让用户通过交互操作来翻转视图，也可以通过定时器来翻转视图，翻转的次序由你来定。

下面来看一个例子，这是一个简单的 Activity（Fancy/Flipper1）的布局，其中包含一个 Button 和一个 ViewFlipper：

```
<?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/flip_me"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Flip Me!"
      />
  <ViewFlipper android:id="@+id/details"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textStyle="bold"
      android:textColor="#FF00FF00"
      android:text="This is the first panel"
    />
    <TextView
      android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textColor="#FFFF0000"
        android:text="This is the second panel"
      />
      <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textColor="#FFFFFF00"
        android:text="This is the third panel"
      />
    </ViewFlipper>
  </LinearLayout>
```

我们注意到，这个布局为 ViewFlipper 定义了 3 个子视图——TextView，每个子视图中只包含一条简单的消息。当然啦，如果你愿意，也可以在这里包含非常复杂的子视图。

9.6.1　手工翻转

要手工翻转视图，必须将视图与 Button 联系起来，并在按钮被点击时翻转它们：

```
public class FlipperDemo extends Activity {
  ViewFlipper flipper;

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

    flipper=(ViewFlipper)findViewById(R.id.details);

    Button btn=(Button)findViewById(R.id.flip_me);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        flipper.showNext();
      }
    });
  }
}
```

就如同使用任何 ViewAnimator 类一样，实现手工翻转无非就是在 ViewFlipper 上面调用 showNext()方法而已。

结果就是一个简单的 Activity：点击按钮，显示队列中的下一个 TextView；显示完最后一个，继续从头显示第一个。如图 9-11 和图 9-12 所示。

当然，更简单的办法是使用一个 TextView，在每次单击按钮时修改其中的文本和颜色。不过，你可以想象一下，ViewFlipper 里面其实可以放更复杂的内容，就如同放到 TabView 中的内容一样。

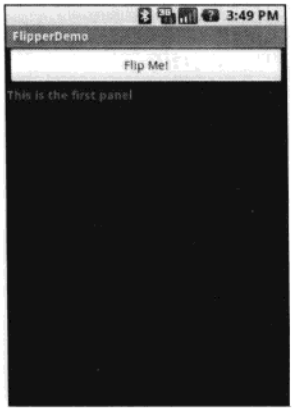


图 9-11 Flipper1 应用程序，显示第一个面板（panel）

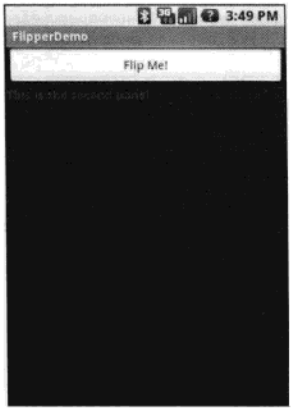


图 9-12 相同的应用程序，切换到了第二个面板

9.6.2 动态添加内容

与使用 TabWidget 一样，有时候，在编译时恐怕也无法知道 ViewFlipper 包含什么内容。当然，与使用 TabWidget 一样，这种情况下动态添加内容还是易如反掌。

下面，我们再看另外一个示例 Activity（Fancy/Flipper2），其布局文件如下：

```
<?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"
    >
  <ViewFlipper android:id="@+id/details"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  </ViewFlipper>
</LinearLayout>
```

看到了吗，ViewFlipper 在编译时不包含内容。而且，这里也没有供用户切换内容的 Button 元素——具体内容，我们到下一节再介绍。

为了给 ViewFlipper 添加内容，我们将创建一些超级大的 Button 部件，每个部件中包含一个随机的词汇（本书有很多章都用到了这些没有意义的词汇）。然后，我们要对 ViewFlipper 进行设置，让它能够自动循环显示这些 Button 部件，而且还要为切换过程应用动画效果。

```
public class FlipperDemo2 extends Activity {
  static String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
                         "consectetuer", "adipiscing", "elit",
                         "morbi", "vel", "ligula", "vitae",
```

```
                        "arcu", "aliquet", "mollis", "etiam",
                        "vel", "erat", "placerat", "ante",
                        "porttitor", "sodales", "pellentesque",
                        "augue", "purus"};
  ViewFlipper flipper;

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

    flipper=(ViewFlipper)findViewById(R.id.details);

    flipper.setInAnimation(AnimationUtils.loadAnimation(this,
                                        R.anim.push_left_in));
    flipper.setOutAnimation(AnimationUtils.loadAnimation(this,
                                        R.anim.push_left_out));

    for (String item : items) {
      Button btn=new Button(this);

      btn.setText(item);

      flipper.addView(btn,
                  new ViewGroup.LayoutParams(
                          ViewGroup.LayoutParams.FILL_PARENT,
                          ViewGroup.LayoutParams.FILL_PARENT));
    }

    flipper.setFlipInterval(2000);
    flipper.startFlipping();
  }
}
```

在从布局文件中取得了 ViewFlipper 部件之后，我们首先为它设置了“进入”（in）和“离开”（out）动画。用 Android 的话来说，动画就是部件离开（out）和进入（in）可视区域的方式。动画是资源，保存在项目的 res/anim 目录下。在这个例子中，我们（遵照 Apache 2.0 许可）使用了由 SDK 示例提供的一对动画。顾名思义，应用这两个动画后，部件无论进入还是离开可视区域，都是被“推”（push）向左侧。

注意　动画是个非常复杂的主题。我在The *Busy Coder's Guide to Advanced Android Development* (CommonsWare LLC, 2009)一书中介绍了动画。

9.6.3　自动翻转

在迭代这堆无意义的词汇期间，我们把每个词分别放到一个 Button 中，然后再将 Button 作为子元素添加到 ViewFlipper 中，最后将 flipper 设置为自动翻转子元素（flipper.setFlipInterval(2000);）并开始翻转（flipper.startFlipping();）。

结果就是一连串无休止切换的按钮。每个按钮出现后都会停留两秒钟，然后就会从左侧滑出，同时队列中的下一个按钮向左侧滑入。等最后一个按钮滑出后，第一个按钮又会滑入，如此往复。图 9-13 展示了一个界面的示例。

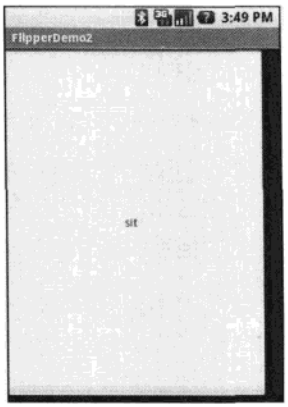


图 9-13 Flipper2 应用程序，正以动画方式切换按钮

这种自动翻转的 ViewFlipper 适合用作状态条，或者要显示的信息很多但空间又不足的情况。不过要注意，视图之间的自动切换会给用户操作单个视图带来不便，因为用户正操作的视图很可能在操作期间会移出屏幕。

9.7 滑动的抽屉

长期以来，Android 开发人员一直都很渴望实现“滑动抽屉”（sliding drawer）式的容器，就跟主屏幕中那个包含应用程序图标的启动菜单一样。在 Android 1.5 之前，官方的实现虽然是开源的，但一直没有包含在 SDK 中。自 Android 1.5 起，开发人员提供了内置的 SlidingDrawer。

SlidingDrawer 与 Android 中的其他大多数容器不一样，它的切换模式是由关闭到打开。正因为如此，并不是任何容器都可以包含 SlidingDrawer——这个容器必须支持多个部件的层叠。RelativeLayout 和 FrameLayout 满足这个条件；FrameLayout 本身就是为层叠部件而设计的。相反，LinearLayout 不允许部件层叠（只支持一个接一个地排列部件），因此 SlidingDrawer 不能作为 LinearLayout 的直接子元素。

下面我们看一个例子（Fancy/DrawerDemo），其中的 SlidingDrawer 包含在 FrameLayout 中：

```
<?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"
    android:background="#FF4444CC"
    >
  <SlidingDrawer
    android:id="@+id/drawer"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:handle="@+id/handle"
    android:content="@+id/content">
    <ImageView
      android:id="@id/handle"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/tray_handle_normal"
    />
    <Button
      android:id="@id/content"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
```

```
        android:text="I'm in here!"
    />
  </SlidingDrawer>
</FrameLayout>
```

SlidingDrawer 包含两个部件：

- “把手”，一般都使用 ImageView 或类似的视图，我们这里使用的图片取自 Android 开源项目；
- “抽屉”本身的内容，一般都是某种容器，但这里使用了一个 Button。

当然，SlidingDrawer 必须通过 android:handle 和 android:content 特性分别知悉“把手”和“内容”和 android:id 值。只有这样，“抽屉”才能以动画效果打开和关闭。

图 9-14 展示了关闭状态的 SlidingDrawer，备有“把手”；图 9-15 展示了它打开的状态。

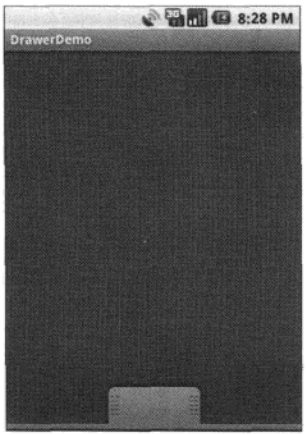


图 9-14　SlidingDrawer，关闭状态

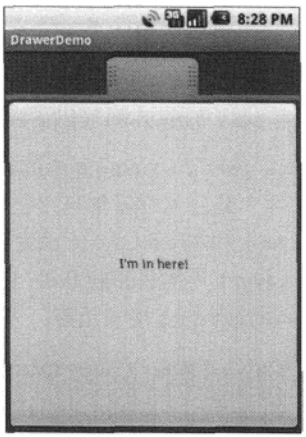


图 9-15　SlidingDrawer，打开状态

读者恐怕已经猜到了，在 Java 代码中通过用户触摸事件同样可以打开和关闭“抽屉”（触摸事件由部件负责处理，因此你不用担心）。此时，有两组方法可供选择：一次到位式（open()、close()和 toggle()）和动画式（animateOpen()、animateClose()和 animateToggle()）。

另外，还可调用 lock()和 unlock()来“上锁”或“解锁”；在“上锁”后，“抽屉”不再响应触摸事件。

可以根据需要注册下列回调：

- “抽屉”打开时调用的侦听器；
- “抽屉”关闭时调用的侦听器；
- “抽屉”滚动过程中（例如，用户拖拉或回送“把手”时）调用的侦听器。

例如，启动菜单的 SlidingDrawer 会在“抽屉”打开、关闭时，以及长按屏幕时，切换“把手”的图标（长按屏幕会导致“把手”变成“删除”图标）。而这些切换很大程度上就是通过上述回调实现的。

SlidingDrawer 可以是垂直的，也可以是水平的。但要知道，它的方向与屏幕的方向无关。换句话说，无论如何旋转运行 DrawerDemo 的 Android 设备或模拟器，“抽屉”始终都只能从底部打开——它不会始终附着在打开它的那一边。如果你想让自己的“抽屉”跟启动菜单一样，始终都从相同一边打开，就必须单独创建纵向和横向布局，具体内容将在第 20 章再作介绍。

9.8 其他容器

Android 还提供了 AbsoluteLayout，包含在其中的内容基于指定的坐标点定位。你只要告诉 AbsoluteLayout 将子部件精确地放在哪个坐标点上（x,y），Android 就会想方设法地把它放到那里。

从积极的一方面说，AbsoluteLayout 可以让你精确地定位。从消极的一方面说，这种绝对布局可能只在屏幕大小既定的设备中看着舒服，除非你再编写一大堆代码来基于屏幕大小调整坐标。考虑到 Android 设备的屏幕有大有小，新尺寸也层出不穷，使用 AbsoluteLayout 很可能会带来无尽的麻烦。

注意 官方已经不推荐使用AbsoluteLayout了；虽然还可以使用，但应该能不用就不用。

Android 还有一个 ExpandableListView，它提供了简单的树状结构，支持二级深度：元素组和子元素。元素组中包含子元素，子元素包含“叶”元素。但由于 ListAdapter 不会给列表中的项提供任何组信息，所以使用 ExpandableListView 必须要有一组新的适配器才行。

第 10 章 输入法框架 10

Android 1.5 引入了 IMF（Input Method Framwork，输入法框架），开发人员通常称它为软键盘。不过，软键盘这种称呼未必准确，因为 IMF 支持手写识别以及其他通过屏幕接收输入文本的方式。

本章介绍如何使用 IMF 控制软键盘，满足应用程序开发的需要。

10.1 键盘，硬还是软

有些 Android 设备（像 HTC Magic）没有硬键盘，而另一些 Android 设备（像 T-Mobile G1）带有硬键盘（滑开可见）。在可见的未来，将会有不少 Android 设备带有硬键盘（例如上网本和某些手机屏幕的下方，始终都会有一个 QWERTY 键盘）。IMF 就是用来处理所有这些情况的。简言之，如果没有硬键盘，那么只要触摸有效的 `EditText`，就会调出 IME（Input Method Editor，输入法编辑器）。

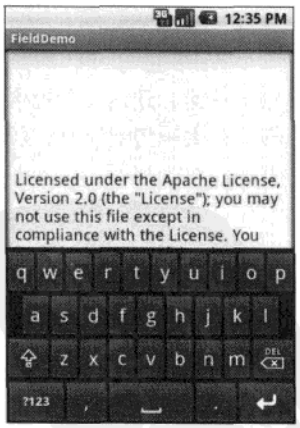


图 10-1 FieldDemo 示例应用程序中的输入法编辑器

只要 IME 的默认功能符合需求，你就不必修改应用程序。好在，Android 也非常聪明，它知道你想要什么，因此有时候只使用 IME 测试就可以了，不必修改代码。

然而，有时候软键盘却不能适合应用程序的需要。例如，在示例项目 Basic/Field 中，IMF 遮住了 `FieldDemo` 中的多行 `EditText`，如图 10-1 所示。这时候，我们希望能够控制软键盘的外观，同时指定 IME 的其他行为。没问题，IMF 为我们提供了许多选项，本章我们就来介绍这些选项。

10.2 按需定制

Android 1.1 及以前的版本为 `EditText` 部件提供了很多特性，用以控制这个输入控件的外观，

例如 android:password 表示密码输入字段(以掩码遮住密码,防止有心人偷窥)。到了 Android 1.5,由于有了 IMF,很多类似的上述特性都被组合到了一个 android:inputType 特性中。

android:inputType 特性的值是类别加修饰符的形式,各项之间以竖线(|)分隔。类别一般描述的是允许用户输入什么,因而也就决定了软键盘中会包含哪些键。可用的类别如下:

- text(默认类别)
- number
- phone
- datetime
- date
- time

这些类别多数都配有额外的修饰符,用以进一步限定用户可以输入的内容。为了理解修饰符的作用,下面我们来看看示例项目 InputMethod/IMEDemo1 的布局文件(res/layout/ main.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="1"
  >
  <TableRow>
    <TextView
      android:text="No special rules:"
    />
    <EditText
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="Email address:"
    />
    <EditText
      android:inputType="text|textEmailAddress"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="Signed decimal number:"
    />
    <EditText
      android:inputType="number|numberSigned|numberDecimal"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="Date:"
    />
    <EditText
      android:inputType="date"
    />
  </TableRow>
  <TableRow>
```

```
    <TextView
      android:text="Multi-line text:"
    />
    <EditText
      android:inputType="text|textMultiLine|textAutoCorrect"
      android:minLines="3"
      android:gravity="top"
    />
  </TableRow>
</TableLayout>
```

我们可以看到一个包含 5 行的 TableLayout，每行展示了一种类别的 EditText：

- 第一行的 EditText 没有指定任何特性，结果就是一个纯文本输入字段；
- 第二行指定了 android:inputType = "text|textEmailAddress"，虽然也是文本字段，但只接受电子邮件地址；
- 第三行指定的 android:inputType = "number|numberSigned|numberDecimal"表示允许输入有符号的十进制数值；
- 第四行则只允许输入日期（android:inputType = "date"）；
- 最后一行允许多行输入，同时能够自动校正拼写错误（android:inputType = "text|textMultiLine|textAutoCorrect"）。

以上类别和修饰符共同决定了键盘的配置。纯文本输入字段对应的就是普通的软键盘，如图 10-2 所示。

电子邮件地址输入字段对应的软键盘上带有@符号，因此空格键变短了，如图 10-3 所示。

数值和日期字段对应的键盘上只包含数字键，外加一组在给定字段中可能有效也可能无效的字符，如图 10-4 所示。

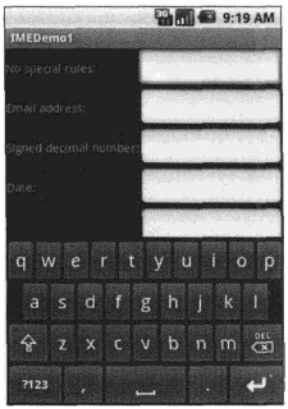


图10-2　标准的输入法编辑器（又称软键盘）

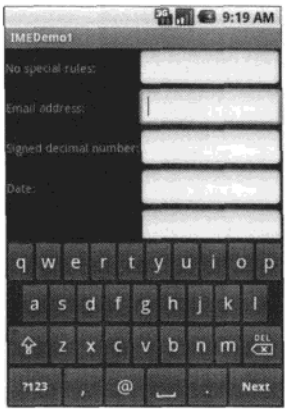


图10-3　对应于电子邮件地址的输入法编辑器

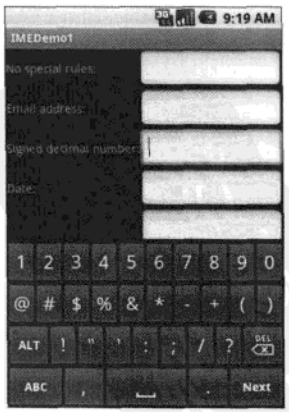


图10-4　与有符号十进制数值对应的输入法编辑器

以上只是几个例子而已。通过选择适当的 android:inputType，可以为用户提供与输入数据最匹配的软键盘。

10.3 修改附属键

注意观察图 10-2 和图 10-3，你会发现这两个 IME 除了@键不同之外，还有一处细微的差别。软键盘右下角的键在第一个字段时是 Next 键，而在第二个字段时是换行键盘。这反映出两点：

- 如果未指定 android:inputType，默认的 EditText 允许多行输入；
- 你可以控制右下角那个附属键的功能。

默认情况下，与指定了 android:inputType 的 EditText 部件对应的附属键是 Next（下一个），其功能是将输入焦点转移到下一个 EditText，如果已经到达了最后一个 EditText，那么附属键就是 Done（完成）。通过 android:imeOptions 特性可以手工指定附属键上显示的文本。例如，在示例项目 nputMethod/IMEDemo2 的 res/layout/main.xml 文件中，我们对前一个例子进行了修改，为两个输入字段指定了附属键的外观：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView 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="1"
    >
    <TableRow>
      <TextView
        android:text="No special rules:"
      />
      <EditText
      />
    </TableRow>
    <TableRow>
      <TextView
        android:text="Email address:"
      />
      <EditText
        android:inputType="text|textEmailAddress"
        android:imeOptions="actionSend"
      />
    </TableRow>
    <TableRow>
      <TextView
        android:text="Signed decimal number:"
      />
      <EditText
        android:inputType="number|numberSigned|numberDecimal"
        android:imeOptions="actionDone"
      />
    </TableRow>
```

```
    <TableRow>
      <TextView
        android:text="Date:"
      />
      <EditText
        android:inputType="date"
      />
    </TableRow>
    <TableRow>
      <TextView
        android:text="Multi-line text:"
      />
      <EditText
        android:inputType="text|textMultiLine|textAutoCorrect"
        android:minLines="3"
        android:gravity="top"
      />
    </TableRow>
  </TableLayout>
</ScrollView>
```

在此，我们为电子邮件地址的附属键指定了 Send（发送）动作（android:imeOptions = "actionSend"），为中间的字段指定了 Done（完成）动作（android:imeOptions = "actionDone"）。

默认情况下，Next 会将焦点移到下一个 EditText，而 Done 会关闭 IME。不过，无论是对这两个动作，还是对 Send 之类的其他动作，我们都可以通过 EditText（从技术上看，是通过超类 TextView）调用 setOnEditorActionListener()，控制用户单击附属键或按下回车键时执行的操作。除了可以为预期动作提供一个标志之外（例如 IME_ACTION_SEND），还可以编写相应代码来处理该请求（例如，向提供的电子邮件地址发送一封电子邮件）。

10.4　适应布局

细心的读者会发现，IMEDemo1 的布局与前面 IMEDemo2 的布局还有一处差别，即使用了 ScrollView 容器来包含 TableLayout。这样，就为 IME 引入了另一个层次的控制：软键盘出现时，你的 Activity 会有什么变化？根据环境不同，有 3 种可能。

- Android 可能会“推移”Activity，实际上是滑动整个布局以容纳 IME；或者，根据被编辑的 EditText 是在上方还是在下方，有可能让 IME 盖住布局，结果 UI 的某些部分就会被隐藏起来。
- Android 可能会调整 Activity 的大小，也就是把它缩小到较小的屏幕尺寸，从而让 IME 显示在 Activity 下方。这种情况适合容易收缩的布局（例如，由列表控制的布局或者不需要占据整个屏幕的多行输入字段）。
- 在横向模式下，Android 可能会全屏显示 IME，遮挡住整个 Activity。此时的键盘尺寸更大了，因此输入数据也更容易。

全局显示完全由 Android 内部控制。而在默认情况下，Android 会根据你的布局选择“推移”

或“调整大小”的模式。如果你想自己指定 “推移”或者“调整大小”模式，则需要修改 AndroidManifest.xml 文件的<activity>元素，为它指定 android:windowSoftInputMode 特性。例如，下面就是 IMEDemo2 的描述文件（manifest）：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.commonsware.android.imf.two"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:label="@string/app_name"
        android:icon="@drawable/cw">
        <activity android:name=".IMEDemo2"
                  android:label="@string/app_name"
                  android:windowSoftInputMode="adjustResize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

由于我们在这里指定了“调整大小”（android:windowSoftInputMode="adjustResize"），因此 Android 会缩小布局以适应 IME。在为布局添加 ScrollView 容器的情况下（也就是说，必要时会出现滚动条），带有 IME 的 UI 如图 10-5 所示。

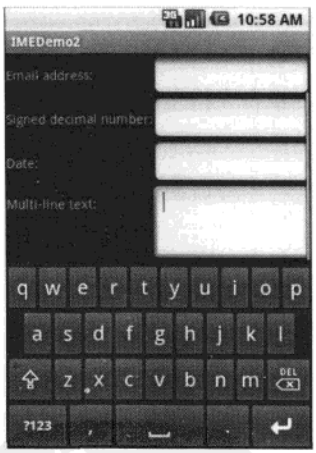


图 10-5 缩小后可滚动的布局

10.5 释放创造力

你还可以创建和发布自己的 IME。比如创建一个 Dvorak 软键盘、一个其他语言的键盘或者一个能够发音的键盘。

IME 包是作为服务的形式存在的，Service 组件将在第 29 章和第 30 章介绍。如果读者对创建这样一个编辑器有兴趣，那么最好看一看 Android 1.5 SDK 附带的 SoftKeyboard 示例应用程序，还有相应的 Android 源代码（搜索 LatinIME 类）。

第 11 章 使用菜单

与桌面应用程序和某些移动操作系统（如 Windows Mobile）一样，Android 的 Activity 中也支持应用程序菜单，叫做**选项菜单**。有的 Android 手机专门设计了一个键用于调出选项菜单，有的则提供了其他手段，例如 Android 平板电脑 Archos 5 使用的屏幕按钮。

Android 还与许多 GUI 工具包类似，也支持在应用程序中创建上下文菜单。在移动设备中，要调出上下文菜单通常需要触摸并按住某个部件。例如，某个 TextView 拥有上下文菜单，而相应设备支持手指触摸输入，那么通过手指按住这个 TextView 并保持一到两秒钟，就会弹出上下文菜单来。

本章就来介绍如何使用 Android 的选项和上下文菜单。

11.1 选项菜单

选项菜单通过按下设备上的物理 Menu 按钮来触发。

选项菜单有两种模式：图标模式和扩展模式。用户在按下 Menu 按钮后，屏幕下方会以图标模式将 6 个菜单项显示在网格中，每个菜单项都比较大，便于通过手指点按。如果菜单项的数量不止 6 个，网格中的第六个按钮会变成 More（更多）。选择 More 项会调出以扩展模式显示的更多菜单项，此时的菜单是可以滚动的，用户可以在此找到常规菜单中不包含的任何项。

11.1.1 创建选项菜单

与构建 UI 的其他部分时调用 onCreate()方法不同，创建选项菜单必须实现 onCreate-OptionMenu()方法，这个方法接收一个 Menu 的实例。

你要做的第一件事，就是与超类建立联系（super.onCreateOptionsMenu(menu)），以便 Andorid 框架能够添加它认为必要的菜单项。然后，你就可以添加自己的菜单项了，如何添加菜单项将在下一节介绍。

如果想在 Activity 运行期间调整菜单（例如，禁用当前无效的菜单项），可以设置传入 onCreateOptionsMenu()方法的 Menu 实例。或者，也可以实现 onPrepareOptionsMenu()，这个方法会在每次显示菜单之前被调用。

11.1.2 添加菜单项和子菜单

在通过 onCreateOptionsMenu()接收到 Menu 对象之后，调用它的 add()方法可以添加菜单项。调用 add()方法的方式有很多种，这取决于如何使用下列参数。

- 分组标识符（int），一般值为 NONE，除非是创建了一个在 setGroupCheckable()中使用的特定菜单项分组（稍后介绍）；
- 菜单项标识符（int），用于在菜单项被选中时在 onOptionsItemSelected()回调方法中识别该项；
- 顺序标识符（int），指定在菜单中同时包含 Android 提供的项及自定义项时，当前菜单项的位置；目前，可以使用 NONE；
- 菜单项的文本，可以是 String，也可以是资源 ID。

所有 add()方法都返回 MenuItem 实例，通过这个实例可以调整现有菜单项的设置（例如，修改菜单项的文本）。

还可以为菜单项设置快捷键，也就是在菜单项可见时用于选择该项的一个助记符。Android 支持字母（或 QWERTY）快捷键，也支持数字快捷键，分别通过调用 setAlphabeticShortcut()和 setNumericShortcut()来设置。在菜单上调用 setQwertyMode()并传入参数 true，可以将菜单设置为字母快捷键模式。

菜单项和分组标识符很重要，通过它们可以激活额外的菜单功能，例如：

- 如果菜单项由复选框和标题组成，那么调用 MenuItem#setCheckable()时传入相应的菜单项标识符，可以让用户在选择该项时切换复选框的值；
- 调用 Menu#setGroupCheckable()，将一组菜单项转换成一组互斥的单选按钮，以便任何时候这个组里都只能有一项处于被选中的状态。

最后，通过调用 addSubMenu()并传入与 addMenu()相同的参数，可以添加外挂的子菜单。Android 最终会调用 onCreatePanelMenu()并传入相应子菜单的标识符，以及另一个表示该子菜单的 Menu 的实例。与 onCreateOptionsMenu()一样，同样也需要先向上与超类建立联系，然后再向这个子菜单中添加菜单项。不过有一个限制——不能无限制地嵌套子菜单。一个菜单只能有一个子菜单，而且子菜单不能再包含子菜单。

如果用户选择了一个菜单项，那么 Activity 就会通过 onOptionsItemSelected()回调方法获

知用户选择了菜单项。在这个方法中，你会接收到一个与被选择的菜单项对应的 MenuItem 对象。此时的典型模式就是使用 switch()来判断菜单项的 ID（item.getItemId()），然后再执行适当的操作。注意，无论用户选择的是一级菜单还是子菜单中的项，Android 都是通过 onOptionsItemSelected()来通知你的。

11.2　上下文菜单

用手指按住某个带有上下文菜单的部件不放时，就可以调出上下文菜单。

总的来说，上下文菜单与选项菜单的构成类似。它们的主要区别在于如何填充菜单和如何获知菜单项被选择。

首先，需要说明 Activity 中的哪个（些）部件有上下文菜单。为此，要在 Activity 中调用 registerForContextMenu()，传入带有上下文菜单的 View（部件）。

其次，要实现 onCreateContextMenu()，并向其传入上述传给 registerForContextMenu()的 View 及其他一些参数。在 Activity 中有多个菜单的情况下，可以通过这个方法决定构建哪个菜单。

onCreateContextMenu()的参数有 ContextMenu 本身、上下文菜单所属的 View 以及 ContextMenu.ContextMenuInfo；最后一个参数保存的是用户在调出上下文菜单时按住的列表项，可以根据这个信息来自定义上下文菜单。例如，可以根据该项的当前状态来切换可选的菜单项。

另一个要注意的问题是，onCreateContextMenu()会在每次请求上下文菜单时被调用。这与选项菜单不同（选项菜单针对每个 Activity 只构建一次）；换句话说，在用户使用后或退出后，上下文菜单都会被丢弃。实际上，并不需要持久地保存 ContextMenu 对象，只要根据用户操作在必要时重新构建上下文菜单即可。

要想知道用户选择了哪个上下文菜单项，需要在活动中实现 onContextItemSelected()。但在这个回调方法中，只能接收到被选择的 MenuItem 实例。为此，如果 Activity 中包含两个或多个上下文菜单，就要注意为所有菜单项都赋予唯一的标识符，以便在这个回调方法中区分它们。而且，在 MenuItem 上调用 getMenuInfo()得到的信息与在 onCreateContentMenu()中得到的 ContextMenu.ContextMenuInfo 相同。除此之外，这个回调方法的行为与上一节讲到的 onOptionsItemSelected()相同。

11.3　简单的示例

示例项目 Menus/Menus 是第 7 章 ListView（列表）示例的修订版，这次则添加了选项和上下文菜单。由于这些菜单是通过 Java 代码定义的，因此 XML 布局跟第 7 章的布局没有什么不同。在 Java 代码中，有一些新的变化：

```
public class MenuDemo extends ListActivity {
  TextView selection;
  String[] items={"lorem", "ipsum", "dolor", "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};
  public static final int EIGHT_ID = Menu.FIRST+1;
  public static final int SIXTEEN_ID = Menu.FIRST+2;
  public static final int TWENTY_FOUR_ID = Menu.FIRST+3;
  public static final int TWO_ID = Menu.FIRST+4;
  public static final int THIRTY_TWO_ID = Menu.FIRST+5;
  public static final int FORTY_ID = Menu.FIRST+6;
  public static final int ONE_ID = Menu.FIRST+7;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new ArrayAdapter<String>(this,
              android.R.layout.simple_list_item_1, items));
    selection=(TextView)findViewById(R.id.selection);

    registerForContextMenu(getListView());
  }

  public void onListItemClick(ListView parent, View v,
                              int position, long id) {
   selection.setText(items[position]);
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
                                  ContextMenu.ContextMenuInfo menuInfo) {
    populateMenu(menu);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    populateMenu(menu);

    return(super.onCreateOptionsMenu(menu));
  }
  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    return(applyMenuChoice(item) ||
           super.onOptionsItemSelected(item));
  }

  @Override
  public boolean onContextItemSelected(MenuItem item) {
    return(applyMenuChoice(item) ||
           super.onContextItemSelected(item));
  }

  private void populateMenu(Menu menu) {
    menu.add(Menu.NONE, ONE_ID, Menu.NONE, "1 Pixel");
    menu.add(Menu.NONE, TWO_ID, Menu.NONE, "2 Pixels");
    menu.add(Menu.NONE, EIGHT_ID, Menu.NONE, "8 Pixels");
    menu.add(Menu.NONE, SIXTEEN_ID, Menu.NONE, "16 Pixels");
```

```
    menu.add(Menu.NONE, TWENTY_FOUR_ID, Menu.NONE, "24 Pixels");
    menu.add(Menu.NONE, THIRTY_TWO_ID, Menu.NONE, "32 Pixels");
    menu.add(Menu.NONE, FORTY_ID, Menu.NONE, "40 Pixels");
  }

  private boolean applyMenuChoice(MenuItem item) {
    switch (item.getItemId()) {
      case ONE_ID:
        getListView().setDividerHeight(1);
        return(true);

      case EIGHT_ID:
        getListView().setDividerHeight(8);
        return(true);

      case SIXTEEN_ID:
        getListView().setDividerHeight(16);
        return(true);

      case TWENTY_FOUR_ID:
        getListView().setDividerHeight(24);
        return(true);

      case TWO_ID:
        getListView().setDividerHeight(2);
        return(true);

      case THIRTY_TWO_ID:
        getListView().setDividerHeight(32);
        return(true);

      case FORTY_ID:
        getListView().setDividerHeight(40);
        return(true);
    }

    return(false);
  }
}
```

在 onCreate()方法中，我们为列表部件注册了一个上下文菜单，并在 onCreateContextMenu()中利用私有方法 populateMenu()填充了这个菜单。

我们也实现了 onCreateOptionsMenu()回调方法，这意味着 Activity 也包含一个选项菜单。同样，选项菜单的填充任务也委托给了 populateMenu()。

对 onOptionsItemSelected()（选项菜单项被选择）和 onContextItemSelected()（上下文菜单项被选择）的实现，都委托给了私有方法 applyMenuChoice()，在用户没有选中我们提供的菜单项的情况下，则通过超类来处理。

在 populateMenu()中，我们添加了 7 个菜单项，每个菜单项带有唯一的标识符。为了省事，这里没有使用图标。

在 applyMenuChoice()中，我们检查了用户是否选择了某个菜单项，如果是，则将列表的分隔条设置为用户所选的宽度。

开始的时候，通过模拟器看到的 Activity 与 ListDemo 中的相同，如图 11-1 所示。

在按下 Menu 按钮后，会看到图 11-2 中所示的选项菜单。

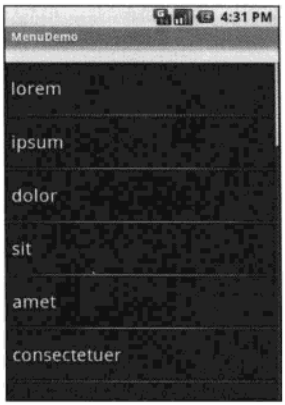


图 11-1 MenuDemo 示例应用程序，初始启动状态

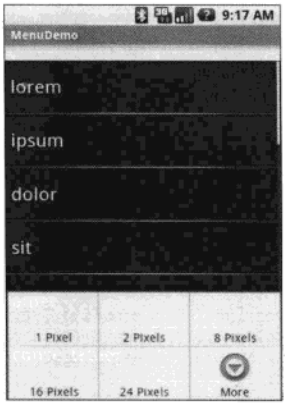


图 11-2 相同的示例应用程序，显示了选项菜单

选择 More 按钮，显示了另外两个菜单项，如图 11-3 所示。

从菜单项中选择相应的高度（如，16 Pixels），会导致列表分隔条的高度变得更刺眼，如图 11-4 所示。

通过按住列表中的一项不放，也可以触发上下文菜单，如图 11-5 所示。同样，选择一项会设置分隔条的高度。

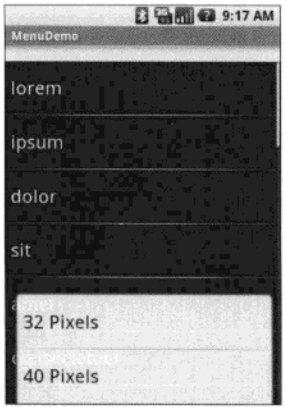


图11-3 相同的示例应用程序，显示了其他菜单项

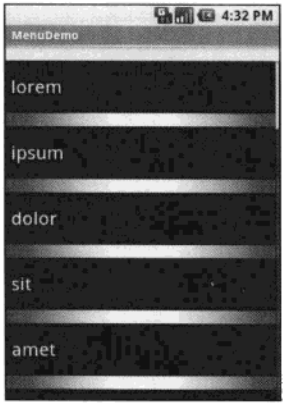


图11-4 相同的示例应用程序，很难看

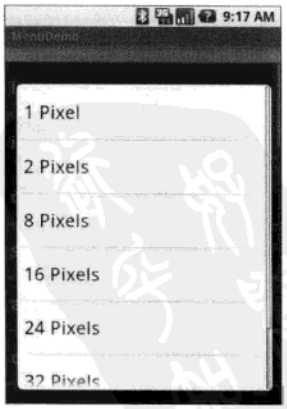


图11-5 相同的示例应用程序，显示上下文菜单

11.4　扩展的示例

第8章解释了如何通过XML文件来描述View，然后再在运行时将它们“装填”为实际的View对象。Android也允许通过XML文件来描述菜单，然后在需要时再装填菜单。这样可以保持菜单结构与菜单处理逻辑的分离，也为开发菜单创建工具提供了一种简便方式。

11.4.1　菜单的XML结构

菜单的XML文件位于项目的res/menu目录中，与项目可能用到的其他资源放在一块。与布局文件一样，一个项目也可以有多个菜单XML文件，每个文件的扩展名都是.xml，但以不同的文件名命名。

例如，在Menus/Inflation示例项目中，有一个名为sample.xml的菜单文件：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/close"
    android:title="Close"
    android:orderInCategory="3"
    android:icon="@drawable/eject" />
  <item android:id="@+id/no_icon"
    android:orderInCategory="2"
    android:title="Sans Icon" />
  <item android:id="@+id/disabled"
    android:orderInCategory="4"
    android:enabled="false"
    android:title="Disabled" />
  <group android:id="@+id/other_stuff"
    android:menuCategory="secondary"
    android:visible="false">
    <item android:id="@+id/later"
      android:orderInCategory="0"
      android:title="2nd-To-Last" />
    <item android:id="@+id/last"
      android:orderInCategory="1"
      android:title="Last" />
  </group>
  <item android:id="@+id/submenu"
    android:orderInCategory="3"
    android:title="A Submenu">
    <menu>
      <item android:id="@+id/non_ghost"
        android:title="Non-Ghost"
        android:visible="true"
        android:alphabeticShortcut="n" />
      <item android:id="@+id/ghost"
        android:title="A Ghost"
        android:visible="false"
        android:alphabeticShortcut="g" />
    </menu>
  </item>
</menu>
```

在构建XML菜单时要注意以下几点。

- 根元素必须是<menu>。

- <menu>元素中是<item>元素和<group>元素，后者表示可以作为一组项目来操作的菜单项的集合。
- 通过在<item>元素中添加<menu>元素来指定子菜单，然后再使用这个新的<menu>元素来描述子菜单的内容。
- 如果想在 Java 代码中检测用户选择了哪一项，或者引用某一个<item>或<group>，别忘了像在 View 的布局文件中那样，为它们指定 android:id 特性。

11.4.2 菜单项与 XML

在<item>和<group>元素内部，可以指定各种选项，对应着 Menu 或 MenuItem 中的相关方法，简述如下。

- 标题（*title*）：通过<item>元素的 android:title 特性指定菜单项的标题。可以是字符串字面量，也可以是对字符串资源的引用（例如，@string/foo）。
- 图标（*icon*）：菜单项的图标是可选的。指定图标时，要将一个指向可绘制资源的引用（例如，@drawable/eject）作为<item>元素 android:icon 特性的值。
- 次序（*order*）：默认情况下，菜单项的先后次序由它们在 XML 文件中的次序决定。要改变默认次序，可以在<item>元素中指定 android:orderInCategory 特性。这个特性的值是一个从 0 开始计数的索引，表示当前项在当前类别中的次序。有一个隐式的默认类别。要显式地指定一个不同的类别，可以在<group>元素中指定 android:menuCategory 特性，以便包含在该<group>中的<item>使用。一般来说，最简单的方法就是按照它们应该出现的先后次序在 XML 中将它们排列好。
- 启用（*enabled*）：可以启用或禁用菜单项、组，方法是在 XML 文件中通过<item>和<group>元素的 android:enabled 特性来控制。默认情况下，菜单项和组都是启用的。禁用后的菜单项和组也会出现在菜单上，只是不能选择了。在运行时，可以通过调用 MenuItem 的 setEnabled()方法，或者调用 Menu 的 setGroupEnabled()方法来改变菜单项和组的启用状态。
- 可见（*visible*）：在 XML 文件中通过<item>和<group>元素的 android:visible 特性可以控制菜单项、组是否可见。默认情况下，菜单项、组是可见的。设置为不可见时，它们不会出现在菜单中。在运行时，可以通过调用 MenuItem 的 setVisible()方法，或者调用 Menu 的 setGroupVisible()方法来改变菜单项和组的可见状态。在前一节的 XML 布局文件中，就将 other_stuff 菜单组设置成了一开始是不可见的。这样，在 Java 代码中，我们就可以调用相应的方法将它们设置为可见，达到“变魔术”的效果。
- 快捷键（*shortcut*）：可以为菜单项指定快捷键，如一个字母（android:alphabeticShortcut）或数字（android:numericShortcut），从而为用户提供一种除触摸屏幕、D-pad 或轨迹球之外的选择菜单项的方式。

11.4.3　创建菜单

在XML中定义完之后，到了真正使用菜单的时候，其实非常简单。只要创建MenuInflater并告诉它去填充菜单即可：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  theMenu=menu;

  new MenuInflater(getApplication()).inflate(R.menu.sample, menu);

  return(super.onCreateOptionsMenu(menu));
}
```

第 12 章 字 体 12

在开发任何类型的应用程序时，不可避免地都会出现这样一个问题："嘿，我们能改变一下这种字体吗？"答案取决于平台提供了哪些字体、你是否可以添加其他字体，以及如何将它们应用至部件或者诸如此类需要改变字体的地方。

Android 应用程序中的字体与其他应用程序中的没有什么差别。Android 带有一些字体，并且还可添加新字体。但是，与任何新环境一样，一些特性需小心处理，本章将就此简单介绍。

12.1 珍惜已有字体

Android 本身能识别 3 种字体，用速记符号表示为"sans"、"serif"和"monospace"。这些字体实际上是 Droid 系列的字体，由 Ascender 公司（http://www.ascendercorp.com/ oha.html）为开放手机联盟制作。要使用这些字体，你可以在布局 XML 中引用它们，如 Fonts/FontSampler 示例项目中的下列布局所示：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:stretchColumns="1">
  <TableRow>
    <TextView
      android:text="sans:"
      android:layout_marginRight="4px"
      android:textSize="20sp"
    />
    <TextView
      android:id="@+id/sans"
      android:text="Hello, world!"
      android:typeface="sans"
      android:textSize="20sp"
    />
  </TableRow>
  <TableRow>
```

```
    <TextView
      android:text="serif:"
      android:layout_marginRight="4px"
      android:textSize="20sp"
    />
    <TextView
      android:id="@+id/serif"
      android:text="Hello, world!"
      android:typeface="serif"
      android:textSize="20sp"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="monospace:"
      android:layout_marginRight="4px"
      android:textSize="20sp"
    />
    <TextView
      android:id="@+id/monospace"
      android:text="Hello, world!"
      android:typeface="monospace"
      android:textSize="20sp"
    />
  </TableRow>
  <TableRow>
    <TextView
      android:text="Custom:"
      android:layout_marginRight="4px"
      android:textSize="20sp"
    />
    <TextView
      android:id="@+id/custom"
      android:text="Hello, world!"
      android:textSize="20sp"
    />
  </TableRow>
</TableLayout>
```

此布局会构建一个表格，显示 4 种字体的简短示例。请注意前 3 个内容示例都有 android:typeface 特性，其值为 3 种内置字体之一（例如“sans”）。

12.2　更多字体

这 3 种内置字体非常实用。但是，设计师、管理人员或客户可能想使用不同的字体，或者因为专门用途想要使用某种其他字体，如用印刷装饰字体来代替一组 PNG 图。完成这一操作最简单的方法是将所需的字体打包到应用程序中。在项目根目录中创建一个 assets/文件夹，然后将你的 TrueType (TTF)字体放入该文件夹。例如，你可以创建 assets/fonts/文件夹并将 TTF 文件放入其中。

然后需要告知部件使用这些字体。遗憾的是，这样就不能再为这些部件使用布局 XML 了，因为 XML 无法识别作为应用程序资产加入项目的任何字体。此时，你需要在 Java 代码中进行修改：

```
public class FontSampler extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);

    TextView tv=(TextView)findViewById(R.id.custom);
    Typeface face=Typeface.createFromAsset(getAssets(),
                                           "fonts/HandmadeTypewriter.ttf");

    tv.setTypeface(face);
  }
}
```

此处我们取得用于 custom 示例的 TextView，然后通过静态的 createFromAsset()构建程序方法创建 Typeface 对象。这个方法接受应用程序的 AssetManager（由 getAssets()返回）及 assets/目录内指向所需字体的路径作为参数。

然后就是通过 TextView 调用 setTypeface()的问题了，传入你刚刚创建的 Typeface。在此处所示的情况下，我们会使用 Handmade Typewriter 字体（http://moorstation.org/typoasis/designers/klein07/text01/handmade.htm）。图 12-1 显示了结果。

图 12-1 FontSampler 应用程序

注意，Android 似乎没有对所有 TrueType 字体做到一视同仁。当 Android 不识别某个自定义字体时，它不会引发异常，而悄悄地将其换成 Droid Sans（“sans”）字体。因此，如果你试用不同的字体，却似乎无法奏效，可能是这种字体与 Android 不兼容。

此外，将字体的文件名全部改为小写形式，以配合其他资源采用的命名规范，这样可能使用起来更顺手。

Android 1.6 可根据文件系统（如用户 SD 卡上的文件系统）中的 TrueType 文件，通过 Typeface 上的 createFromFile()静态方法创建 Typeface 对象。

12.3 字形介绍

TrueType 字体使用起来可能有些不方便，特别是它支持大量可用的 Unicode 字符子集的情况下。前面示例中使用的 Handmade Typewriter 字体大小超过 70KB。DejaVu 免费字体可超过 500KB。即使经过压缩，也会使应用程序增大太多，所以尽量不要过分使用自定义字体，以免应用程序占用太多的用户手机空间。

还有要切记，字体可能并不具备你需要的所有字形。接下来，让我们讨论一下省略号。

Android的TextView类具有的内置功能可“省略”文本，如果文本长于可用的空间，那么将截短文本并添加省略号。例如，你可以通过android:ellipsize特性使用该功能。这种工作方式很好，至少对于单行文本来说的确如此。

Android采用的省略号不是3个句点，而是实际的省略号字符，其中的3个点均包含在一个字形中。因此，如果使用省略功能，显示的任何字体都需要省略号字形。

除此之外，虽然Android可能会使屏幕上呈现的字符串内容增加，但是这样添加省略号前后的长度（以字符为单位）能够保持相等。为了完成这项操作，Android用省略号替换一个字符，然后用Unicode字符‘ZERO WIDTH NO-BREAK SPACE’(U+FEFF)替换其他所有被删除的字符。这样，省略号后面的多余字符便不会占用屏幕上的任何可见空间，但仍是字符串的一部分。不过，这意味着通过android:ellipsize特性用于TextView部件的任何自定义字体还必须支持这种特殊的Unicode字符。并非所有字体均必须支持这种特殊字符，如果字体缺少这种字符，那么缩短的字符串在屏幕上会显示奇怪的内容。（例如，多个讨厌的X会显示在行尾。）

当然，Android的国际部署意味着字体必须能够处理用户可能要输入的任何语言字符，也可以通过特定于语言的输入法编辑器进行处理。

因此，虽然在Android中使用自定义字体是完全可能的，但仍存在许多潜在问题。对于应用程序，应该仔细权衡引入自定义字体的利弊。

第 13 章 嵌入 WebKit 浏览器

有不少 GUI 工具包允许使用 HTML 来呈现信息，从功能有限的 HTML 渲染器（例如，Java/Swing 和 wxWidgets）到将 Internet Explorer 嵌入到.NET 应用程序中。Android 也几乎一样，因为可以将内置的 Web 浏览器以部件的形式嵌入到 Activity 中，用于显示 HTML 或实现全面的浏览功能。Android 浏览器基于 WebKit，苹果公司的 Safari Web 浏览器也基于该引擎。

Android 浏览器比较复杂，它拥有自己的 Java 包（android.webkit）。WebView 部件本身既可以实现简单的功能，也可以实现强大的功能，这取决于需求，本章中将简单介绍这一点。

13.1 小型浏览器

简单来讲，WebView 与 Android 中的其他部件并没有显著的不同。可通过 Java 代码将它插入到布局中，告诉它导航到哪个 URL，仅此而已。

例如，下面给出了一个使用 WebView 的简单布局（WebKit/Browser1）：

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

与其他部件一样，需要告诉它应该如何填充布局的空间。在本例中，它填充了所有剩余空间。

对应的 Java 代码同样也很简单：

```
package com.commonsware.android.browser1;

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

```
import android.webkit.WebView;

public class BrowserDemo1 extends Activity {
  WebView browser;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    browser=(WebView)findViewById(R.id.webkit);

    browser.loadUrl("http://commonsware.com");
  }
}
```

这个 onCreate()版本唯一不同的是，我们在 WebView 部件上调用了 loadUrl()，告诉它加载一个网页（在本例中为某个公司的主页）。

但是，我们还需要对 AndroidManifest.xml 进行一处更改，以请求互联网访问权限：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.browser1">
  <uses-permission android:name="android.permission.INTERNET" />
  <application android:icon="@drawable/cw">
    <activity android:name=".BrowserDemo1" android:label="BrowserDemo1">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

如果未能添加此权限，浏览器将拒绝加载页面。第 28 章将详细介绍权限。

得到的 Activity 看起来像一个 Web 浏览器，但却没有滚动条，如图 13-1 所示。

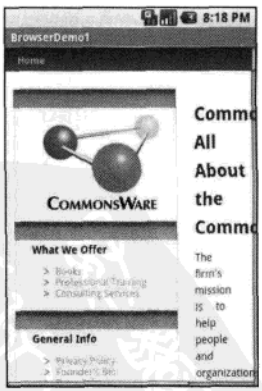


图 13-1　Browser1 示例应用程序

与常规的 Android 浏览器一样，可以通过拖动操作来平移页面。使用 D-pad 可在网页上所有可拥有焦点的元素间移动。

所缺少的是构成 Web 浏览器的额外元素，如导航工具栏。

现在，你可能想将这段源代码中的 URL 替换为大量使用 JavaScript 的网页的地址，例如谷歌公司的主页地址。默认情况下，JavaScript 在 WebView 部件中是被关闭的。如果希望启用 JavaScript，在 WebView 实例上调用 getSettings().setJavaScriptEnabled(true)即可。

13.2　加载内容

向 WebView 添加内容有两种主要方式。一种方式是向浏览器提供一个 URL，让浏览器通过

loadUrl()来显示页面，如上一节所述。浏览器将通过手机目前所支持的任何方式访问因特网，这些方式包括 Wi-Fi、蜂窝网络、蓝牙手机，等等。

另一种方式是调用 loadData()。此时你为浏览器提供要查看的 HTML。可以通过此方式执行以下任务：

- 显示以文件形式同应用程序包一同安装的手册；
- 显示在进行其他处理时获取的 HTML 片段，例如 Atom 源中某个条目的描述；
- 使用 HTML 而不是用 Android 部件生成整个 UI。

有两个版本的 loadData()。较简单的一种允许以字符串的形式提供内容、MIME 类型和编码。通常，普通 HTML 的 MIME 类型是 text/html，编码是 UTF-8。

例如，可以将上一个例子中的 loadUrl()调用替换为以下内容：

```
browser.loadData("<html><body>Hello, world!</body></html>",
                 "text/html", "UTF-8");
```

结果如图 13-2 所示。

这也是一个完全可构建的示例，名为 WebKit/Browser2。

图 13-2 Browser2 示例应用程序

13.3 导航内容

从前面可以看到，WebView 部件没有附带导航工具栏，因此适用于工具栏无用且浪费屏幕空间的情况。也就是说，如果希望提供导航功能也没有问题，但需要提供 UI。

- WebView 提供了多种方式来执行常规的浏览器导航，包括以下方法。
- reload()：刷新当前查看的网页。
- goBack()：后退到浏览器历史记录中的上一步。
- canGoBack()：确定是否有可供后退的浏览器历史记录。
- goForward()：前进到浏览器历史记录中的下一步。
- canGoForward()：确定是否有可供前进的浏览器历史记录。
- goBackOrForward()：在浏览器历史记录中后退或前进。如果参数为负数，则表示后退的步数。正数表示前进的步数。
- canGoBackOrForward()：确定浏览器是否可以后退或前进所规定的步数（正/负数约定与 goBackOrForward()的相同）。
- clearCache()：清除浏览器资源缓存。

- clearHistory()：清除浏览历史记录。

13.4　扩展应用程序

如果要将 WebView 作为本地 UI（与浏览网页相反），需要在关键时刻获得控制权，尤其是用户单击链接时。需要确保这些链接得到了合适的处理，例如将你自己的内容加载回 WebView，或向 Android 提交一个 Intent 以在完整的浏览器中打开 URL，或者采取其他方式（参见第 18 章）。

可通过 setWebViewClient()与 WebView 活动挂钩，setWebViewClient()接受一个 WebViewClient 实现的实例作为参数。所提供的回调对象将被告知各种活动。在加载了部分页面时（例如 onPageStarted()），以及主机应用程序需要处理某些由用户或环境发起的事件时（例如 onTooManyRedirects()或 onReceivedHttpAuthRequest()），回调对象都会得到通知。

shouldOverrideUrlLoading()是一个常见的挂钩，此时回调接收到一个 URL（以及 WebView 本身），如果希望处理请求，那么应返回 true，如果希望采用默认处理方式（例如，实际获取 URL 引用的网页），那么应返回 false。例如，在源阅读器应用程序中，可能没有在阅读器中内置具有导航功能的完整浏览器。对于这种情况，如果用户单击一个 URL，你可以使用 Intent 要求 Android 在一个完整的浏览器中加载该页面。但是如果向 HTML 中插入了一个“假”URL，表示由 Activity 提供的对某种内容的链接，你可以亲自更新 WebView。

例如，我们修改一下第一个浏览器演示程序，使其在被单击时显示当前时间。以下是 WebKit/Browser3 中修改后的 Java 代码：

```
public class BrowserDemo3 extends Activity {
  WebView browser;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    browser=(WebView)findViewById(R.id.webkit);
    browser.setWebViewClient(new Callback());

    loadTime();
  }

  void loadTime() {
    String page="<html><body><a href=\"clock\">" +new Date().toString() +"</a></body></html>";

          browser.loadDataWithBaseURL("x-data://base", page, "text/html", "UTF-8", null);
  }

  private class Callback extends WebViewClient {
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
      loadTime();
```

```
            return(true);
        }
    }
}
```

此处我们将一个包含当前时间的简单网页加载到浏览器中（loadTime()），将当前时间做成了 URL 为/clock 的超链接。这里还定义了 WebViewClient 子类的一个实例，提供了我们的 shouldOverrideUrlLoading()实现。在本例中，无论 URL 指向哪里，我们想要的都只是通过 loadTime()重新加载 WebView。

图 13-3 Browser3 示例应用程序

运行此 Activity 将得到如图 13-3 所示的结果。

选择链接并单击 D-pad 中央的按钮将选中该链接，结果将导致使用新时间重新构建该页面。

13.5 设置、首选项和选项

在你最喜爱的桌面 Web 浏览器中，有一些设置、首选项或选项窗口。可以在这些窗口和工具栏控件中调整浏览器的行为，从首选的字体到 JavaScript 的行为都可以。

类似地，可以通过对部件 getSettings()方法的调用所返回的 WebSettings 实例，适当调整 WebView 部件的设置。

WebSettings 拥有许多选项。大部分选项都不常用（如 setFantasyFontFamily()）。但是，下面这些选项可能很有用。

- 通过 setDefaultFontSize()（使用点大小）或 setTextSize()（使用可表示相对大小的常量，如 LARGER 和 SMALLEST）控制字体大小。
- 通过 setJavaScriptEnabled()(完全禁用)和 setJavaScriptCanOpenWindowsAutomatically()（仅阻止其打开弹出窗口）控制 JavaScript。
- 通过 setUserAgent()控制网站呈现。值 0 表示 WebView 为网站提供一个用户代理字符串，表示它是一个移动浏览器。值 1 将提供一个用户代理字符串，表示它是桌面浏览器。

对设置的更改不是持久的，所以如果允许用户确定设置，而不是将设置硬编码到应用程序中，那么应该将它们存储在某处。（例如通过 Android 首选项引擎存储，将在第 21 章讨论。）

第 14 章 显示弹出消息 14

有时，Activity（或其他 Android 代码片段）也需要表露自己的心声。并不是所有与 Android 用户的交互都可以有条不紊地包含在由多个 View 组成的 Activity 中。许多错误会突然冒出来。后台任务的执行时间可能比预期的要长。可能会发生一些异步操作，比如收到短信。对于诸如此类的情况，可能需要能够脱离传统 UI 与用户进行通信的功能。

当然，这并不是什么新话题。对话框形式的错误消息已经存在很长时间了。此外，也存在一些更微妙的指示器，比如任务托盘图标、弹跳图标、手机振动。

Android 提供了多个系统支持脱离基于 Activity 的 UI 用其他方式来提醒用户。一种是通知，它与 Intent 和 Service 联系紧密，因此将在第 31 章介绍。本章将介绍弹出消息的两种方式：Toast 和提醒框。

14.1 弹出 Toast

Toast 是一种短暂的消息，它会自行显示和消失，不需要用户干预。而且，它不会从当前活动的 Activity 那里获取焦点，所以如果用户正忙于编写一部优秀的编程指南，那么他的输入不会被该消息打断。

由于 Toast 是短暂的，所以无法知道用户是否已注意到它。你不会得到任何确认，消息也不会出现太长时间，以至于影响到用户。因此，Toast 通常用于建议性的消息，例如提示一个运行时间很长的后台任务已经完成，电池电量低（但不是太低），等等。

构建 Toast 非常简单。Toast 类提供了一个静态 makeText()方法，它接受一个 String（或字符串资源 ID）并返回一个 Toast 实例。makeText()方法还需要 Activity（或其他 Context）以及一个持续时间。持续时间表示为 LENGTH_SHORT 或 LENGTH_LONG 常量形式，以相对方式指示消息应该显示多久。

如果喜欢使用另一种 View 来构建 Toast，那么只需通过构造函数（它将接受一个 Context 参数）创建一个新 Toast 实例，然后调用 setView()向它提供要使用的视图，调用 setDuration()来设置持续时间。效果要比只使用文本好。

配置好 Toast 后，调用其 show()方法即可显示该消息。

14.2 提醒框

如果喜欢更加经典的对话框样式，可以使用 AlertDialog。与任何其他模态对话框一样，AlertDialog 将弹出并获取焦点，一直显示，直到被用户关闭。可以用这个提醒框来显示关键错误，如在基本 Activity UI 中显示不下的验证消息，或者你觉得必须让用户立即看到的消息。

构造 AlertDialog 的最简单方式是使用 Builder 类。遵循真正的构造器规则，Builder 提供了一系列方法来配置 AlertDialog，每个方法返回 Builder 以便连续调用。最后，在构造器上调用 show()来显示对话框。

Builder 上常用的配置方法如下所示。

- setMessage()：将对话框的“主体”设置为一个简单的文本消息，来自所提供的 String 或所提供的字符串资源 ID。
- setTitle()和 setIcon()：配置要在对话框标题栏中显示的文本和/或图标。
- setPositiveButton()、setNeutralButton()和 setNegativeButton()：指示哪些按钮应该出现在对话框底部，它们应该放在什么位置（分别放在左侧、中间或右侧），这些按钮的显示名称应该是什么，以及单击按钮时应该调用什么逻辑（除了关闭对话框）。

如果需要的 AlertDialog 配置超出了构造器的允许范围，那么不应调用 show()，而应调用 create()来构建 AlertDialog 实例的一部分，配置剩余部分，然后在 AlertDialog 上调用 show()。

调用 show()之后，对话框将显示并等待用户输入。

14.3 检查效果

要查看以上功能的实际应用，请看一下 Messages/Message，其中包含以下布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <Button
    android:id="@+id/alert"
    android:text="Raise an alert"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

```
  <Button
    android:id="@+id/toast"
    android:text="Make a toast"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
```

下面是 Java 代码：

```
public class MessageDemo extends Activity implements View.OnClickListener {
  Button alert;
  Button toast;

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

    setContentView(R.layout.main);

    alert=(Button)findViewById(R.id.alert);
    alert.setOnClickListener(this);
    toast=(Button)findViewById(R.id.toast);
    toast.setOnClickListener(this);
  }

  public void onClick(View view) {
    if (view==alert) {
      new AlertDialog.Builder(this)
        .setTitle("MessageDemo")
        .setMessage("eek!")
        .setNeutralButton("Close", new DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dlg, int sumthin) {
            // do nothing - it will close on its own
          }
        })
        .show();
    }
    else {
      Toast
        .makeText(this, "<clink, clink>", Toast.LENGTH_SHORT)
        .show();
    }
  }
}
```

布局没有什么特别的，就是一对用于触发提醒框和 Toast 的按钮。

单击 Raise an alert 按钮时，我们使用一个构造器（new Builder(this)）设置标题（setTitle("MessageDemo")）、消息（setMessage("eek!")）和暗色的按钮（setNeutralButton("Close", new OnClickListener() ...），然后显示对话框。当单击 Close 按钮时，OnClickListener 回调不会执行任何操作，按下这个按钮只会导致对话框被解除。但是，可以根据用户操作来更新 Activity 中的信息，尤其是在有多个按钮可供用户选择的情况中。得到的结果是一个典型的对话框，如图 14-1 所示。

当单击 Make a toast 按钮时，Toast 类为我们创建一个基于文本的 Toast（makeText(this, "<clink, clink>", LENGTH_SHORT)），然后我们使用 show()来显示它。显示的结果是短暂的、不碍事的消息，如图 14-2 所示。

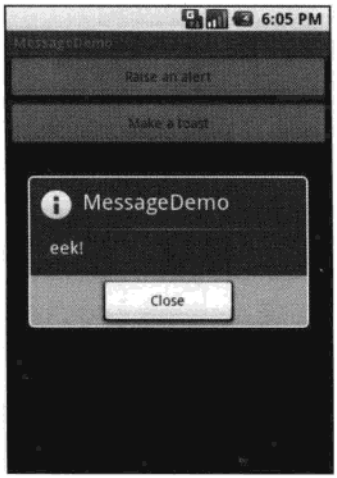


图 14-1 单击 Raise an alert 按钮之后的 MessageDemo 示例应用程序

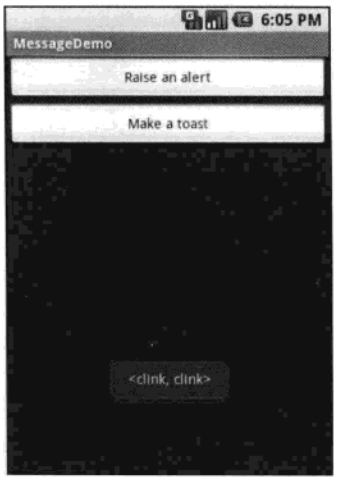


图 14-2 单击 Make a toast 按钮之后的 MessageDemo 应用程序

第 15 章 处理线程 15

在理想情况下，你会希望 Activity 反应迅速，让用户感觉不到应用程序的停顿。迅速响应用户输入（例如在 200 ms 内）是一个重要目标。但至少要确保在 5 s 内做出响应，以免 ActivityManager 认为该 Activity 停止响应而将其结束。

当然，Activity 可能真的有事情需要做，而这件事会占用大量时间。这种情况下就必须使用后台线程了。Android 提供了真正多样化的方式来设置后台线程，但同样允许它们安全地与 UI 线程上的 UI 进行交互。

“安全地交互”这个概念至关重要。不能从后台线程修改 UI 的任何部分；修改 UI 的工作必须在 UI 线程中完成。这通常意味着需要在执行工作的后台线程与显示该工作结果的 UI 线程之间进行某种协调。

本章将介绍如何在 Android 应用程序中使用后台线程和 UI 线程。

15.1 了解处理程序

创建 Android 友好的后台线程时，最灵活的方式就是创建 Handler 子类的一个实例。每个 Activity 仅需要一个 Handler 对象，而且不需要手动注册它。创建该实例就会向 Android 线程子系统注册它。

后台线程可以与 Handler 通信，后者将在 Activity 的 UI 线程上执行自己的所有工作。这一点非常重要，因为 UI 更改（例如更新部件）只能发生在 Activity 的 UI 线程上。

有两种与 Handler 通信的方式：消息和 Runnable 对象。

15.1.1 消息

要向 Handler 发送一个 Message，首先调用 obtainMessage()从池中获取 Message 对象。obtainMessage()对象有许多特点，允许创建空的 Message 对象或填充了消息标识符和参数的对

象。需要的 Handler 处理过程越复杂，就越需要将数据放在 Message 中，以帮助 Handler 区分不同的事件。

然后可通过消息队列将 Message 发送给 Handler，此时需使用 sendMessage...()系列方法中的一个方法，例如以下方法。

- sendMessage()：立即将消息放在队列中。
- sendMessageAtFrontOfQueue()：立即将消息放在队列中，并将其放在消息队列的最前面，这样该消息就会具有比所有其他消息更高的优先级。
- sendMessageAtTime()：在规定的时间将消息放在队列中，这个时间用 ms 表示，基于系统正常工作时间（SystemClock.uptimeMillis()）。
- sendMessageDelayed()：在一段延迟之后将消息放在队列中，延迟用 ms 表示。

要处理这些消息，Handler 需要实现 handleMessage()，将使用出现在消息队列中的每个消息来调用 handleMessage()。此处 Handler 可以根据需要更新 UI。但是，它仍然应该迅速完成此工作，因为在它完成工作之前，其他 UI 工作会暂停。

例如，让我们创建一个 ProgressBar 并通过 Handler 更新它。下面是 Threads/Handler 示例对象的布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <ProgressBar android:id="@+id/progress"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

除了像平常一样设置宽度和高度，ProgressBar 还会使用 style 属性。这种特定的样式表明 ProgressBar 应该绘制为传统的水平进度栏，显示已经完成的工作量。

下面给出了 Java 代码：

```
package com.commonsware.android.threads;

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

public class HandlerDemo extends Activity {
  ProgressBar bar;
  Handler handler=new Handler() {
    @Override
```

```
    public void handleMessage(Message msg) {
      bar.incrementProgressBy(5);
    }
  };
  boolean isRunning=false;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    bar=(ProgressBar)findViewById(R.id.progress);
  }

  public void onStart() {
    super.onStart();
    bar.setProgress(0);

    Thread background=new Thread(new Runnable() {
      public void run() {
        try {
          for (int i=0;i<20 && isRunning;i++) {
            Thread.sleep(1000);
            handler.sendMessage(handler.obtainMessage());
          }
        }
        catch (Throwable t) {
          // just end the background thread
        }
      }
    });

    isRunning=true;
    background.start();
  }

  public void onStop() {
    super.onStop();
    isRunning=false;
  }
}
```

作为构造 Activity 的一部分，我们使用 handleMessage()实现创建了 Handler 的一个实例。从基本上讲，对于收到的任何消息，我们都会将 ProgressBar 更新 5 个百分点，然后退出消息处理程序。

在 onStart()中，我们设置了一个后台线程。在实际系统中，此线程将做一些有意义的事情。在这里，我们只是休眠 1 s，然后将一个 Message 发送给 Handler，并重复该过程 20 次。这与 ProgressBar 位置每次增加 5 个百分点相结合，将在屏幕上清楚地标识进度栏的变化，ProgressBar 默认的最大值为 100。可以通过 setMax()调节该最大值。例如，可以将最大值设置为所处理的数据库的行数，并每处理完一行后更新一次。

请注意，我们然后退出 onStart()。这一步很关键。因为 onStart()方法是在 Activity UI 线程上被调用的，所以它可以更新部件。但是，这意味着我们需要退出 onStart()，让 Handler 完

成自己的工作，也让 Android 不会误认为我们的 Activity 停止了响应。

得到的活动就是一个简单的水平进度栏，如图 15-1 所示。

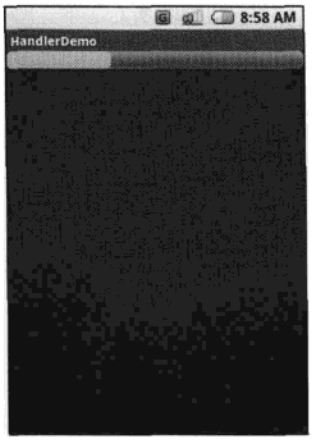


图 15-1 HandlerDemo 示例应用程序

注意，虽然这个 ProgressBar 示例显示了你要编写代码在 UI 线程上更新进度，但对于这个具体的部件，这并不是必需的。自 Android 1.5 开始，ProgressBar 就已经是 UI 线程安全的，因为可以从任何线程更新它，它将处理在 UI 线程上执行实际 UI 更新的各种细节问题。

15.1.2 Runnable

如果不愿意使用 Message 对象，也可以将 Runnable 对象传递给 Handler，Handler 将在 Activity UI 线程上运行这些 Runnable 对象。Handler 提供了一组 post...()方法来传入 Runnable 对象供最终处理。

15.2 就地运行

与 Handler 支持 post()和 postDelayed()以便将 Runnable 对象添加到事件队列一样，我们也可以在 View 上使用这两个方法。这可轻微地简化代码，因为你随后可以跳过 Handler 对象。但是，这会失去一定的灵活性。而且，Handler 已在 Android 工具包中存在很长时间了，可能已通过了更多的测试。

15.3 我的 UI 线程到哪去了

有时，你可能不知道当前是否正在应用程序的 UI 线程上执行操作。例如，如果将一些代码封装到一个 JAR 文件中供其他用户重用，你可能不会知道这些代码正在 UI 线程上执行，还是正在后台线程执行。

为了帮助解决此问题，Activity 提供了 runOnUiThread()。它的工作原理类似于 Handler 和 View 上的 post()方法，因为如果现在不在 UI 线程上，它会让 Runnable 排队等候在 UI 线程上运行。如果已经在 UI 线程上，它会立即调用 Runnable。这同时提供了两方面的优势：如果在 UI 线程上，那么不会有延迟，不在则可以保证安全性。

15.4 异步观感

Android 1.5 引入了一种新的后台操作思考方式 AsyncTask。在一个很方便的类中，Android

在UI线程上处理所有工作，而不是在后台线程上处理。而且，Android本身会分配和删除后台线程。它维护着一个小型工作队列，进一步加强了AsyncTask的“触发即忘”的感觉。

15:4.1　原理

在市场周期方面有一句非常流行的名言：“如果一个人在五金店购买了一个1/4英寸的钻头，他并不是想要一个1/4英寸的钻头，而是想要1/4英寸的钻孔。”五金店不能销售钻孔，所以他们销售替代的产品：能轻松钻出钻孔的装置（钻和钻头）。

类似地，Android开发人员在后台线程管理上付出了大量努力，但他们并不是真的想要后台线程。他们只是希望在不使用UI线程的情况下完成工作，以便用户不用等待，Activity也不会遇到ANR（application not responding，用程序无响应）错误。而且，尽管Android无法使工作不占用UI线程时间，但它可以提供一些方法来使这类后台操作更加简单和透明。AsyncTask就是其中一个例子。

要使用AsyncTask，必须：

- 创建AsyncTask的一个子类，通常作为某种使用该任务的私有内部类（例如一个Activity）；
- 覆写一个或多个AsyncTask方法来完成后台工作，以及与需要在UI上执行的任务相关的工作（例如更新进度）；
- 在需要时，创建AsyncTask子类的一个实例并调用execute()来执行其工作。

不需要完成以下操作：

- 创建后台线程；
- 在适当的时间终止后台线程；
- 调用各种各样的方法安排要在UI线程上完成的处理。

15.4.2　AsyncTask、泛型和Vararg

创建AsyncTask的子类并不像实现Runnable接口那么简单。AsyncTask使用泛型，所以需要指定3种数据类型：

- 处理任务所需的信息类型（例如，用于下载的URL）；
- 任务间传递的用于指示进度的信息类型；
- 任务完成时传递到任务后代码的信息类型。

让问题更复杂的是，前两种数据类型实际上都被用作Vararg，这意味着会在AsyncTask子类

中使用这些类型的一个数组。

看看一个示例应该能让我们理解这些类型。

15.4.3 AsyncTask 的各个阶段

在 AsyncTask 中可以覆写 4 种方法来完成工作。

对于这个很有用的任务类，一个必须覆写的方法是 doInBackground()。AsyncTask 会在后台线程上调用该方法。为了完成特定任务的必要工作，只要有必要就可以运行它。但是请注意，任务应该是有限的，所以不建议在无限循环中使用 AsyncTask。

doInBackground()方法将接受上一小节列出的第一种数据类型的 Vararg 数组作为参数，这个数组是处理任务所需的数据。所以，如果任务的目标是下载一个 URL 集合，doInBackground()将接收这些 URL 供处理。doInBackground()方法一定会返回上一节中第三种类型的值——后台工作的结果。

你可能希望覆写 onPreExecute()方法。在后台线程执行 doInBackground()之前，会从 UI 线程调用此方法。在这里，可以初始化一个 ProgressBar，或者指示后台工作已经开始。

另外，你可能希望覆写 onPostExecute()。在 doInBackground()完成之后会从 UI 线程调用此方法。它接受 doInBackground()返回的值作为参数（例如成功或失败标志）。此时可以解除 ProgressBar 并使用在后台已完成工作的结果，例如更新列表内容。

此外，你还可能希望覆写 onProgressUpdate()。如果 doInBackground()调用了任务的 publishProgress()方法，那么传递给该方法的对象将提供给 UI 线程中的 onProgressUpdate()。这样，onProgressUpdate()可以提醒用户后台工作所完成的进度，例如更新 ProgressBar 或继续播放动画。onProgressUpdate()方法将接受上一小节所列出的第二种数据类型的 vararg，也就是 doInBackground()通过 publishProgress()发布的数据。

15.4.4 示例任务

前面已经提到，实现 AsyncTask 并不像实现 Runnable 那么简单。但是，了解泛型和 vararg 之后，就会发现实现 AsyncTask 实际并不难。

例如，下面是一个使用 AsyncTask 的 ListActivity 实现，这段代码来自 Threads/Asyncer 示例项目：

```
package com.commonsware.android.async;

import android.app.ListActivity;
import android.os.AsyncTask;
```

```
import android.os.Bundle;
import android.os.SystemClock;
import android.widget.ArrayAdapter;
import android.widget.Toast;
import java.util.ArrayList;

public class AsyncDemo extends ListActivity {
  private static String[] items={"lorem", "ipsum", "dolor",
                                 "sit", "amet", "consectetuer",
                                 "adipiscing", "elit", "morbi",
                                 "vel", "ligula", "vitae",
                                 "arcu", "aliquet", "mollis",
                                 "etiam", "vel", "erat",
                                 "placerat", "ante",
                                 "porttitor", "sodales",
                                 "pellentesque", "augue",
                                 "purus"};
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    setListAdapter(new ArrayAdapter<String>(this,
                       android.R.layout.simple_list_item_1,
                       new ArrayList()));

    new AddStringTask().execute();
  }

  class AddStringTask extends AsyncTask<Void, String, Void> {
    @Override
    protected Void doInBackground(Void... unused) {
      for (String item : items) {
        publishProgress(item);
        SystemClock.sleep(200);
      }

      return(null);
    }

    @Override
    protected void onProgressUpdate(String... item) {
      ((ArrayAdapter)getListAdapter()).add(item[0]);
    }

    @Override
    protected void onPostExecute(Void unused) {
      Toast
        .makeText(AsyncDemo.this, "Done!", Toast.LENGTH_SHORT)
        .show();
    }
  }
}
```

这是 lorem ipsum 单词列表的另一个变体，本书中会经常使用。现在，我们并不是简单地将这个单词列表传递给 ArrayAdapter，而是假装需要使用 AsyncTask 的实现 AddStringTask 在后台创建这些单词。

如果构建、安装和运行此项目，数秒之后就会看到列表已被填充，最后由一个 Toast 表示完成，如图 15-2 所示。

图 15-2 正在加载单词列表的 AsyncDemo 应用程序

接下来逐段分析一下此项目的代码。

1. AddStringTask声明

首先，我们看一下 AddStringTask 声明：

```
class AddStringTask extends AsyncTask<Void, String, Void> {
```

这里，我们使用泛型来设置将在 AddStringTask 中利用的特定数据类型，如下所示：

- 本例中不需要任何配置信息，所以第一个类型是 Void；
- 我们希望将后台任务生成的每个字符串传递给 onProgressUpdate()，以便将它添加到列表中，所以第二个类型是 String；
- 严格来讲我们没有任何结果（除了更新），所以第三个类型是 Void。

2. doInBackground()方法

接下来是 doInBackground()方法：

```
@Override
protected Void doInBackground(Void... unused) {
  for (String item : items) {
    publishProgress(item);
    SystemClock.sleep(200);
  }

  return(null);
}
```

doInBackground()方法在后台线程中调用。因此，我们只要愿意，就可以调用它。在产品应用程序中，我们可能会对一个URL列表进行迭代并下载每个URL。在这里，我们迭代静态的lorem ipsum单词列表，为每个单词调用publishProgress()，然后休眠1/4 s以模拟真实的工作。

因为我们选择了不要任何配置信息，所以应该不需要向doInBackground()传递任何参数。但是，与AsyncTask之间的契约表明，我们需要接受第一种数据类型的vararg，因此我们的方法参数为Void...unused。

因为我们选择了没有任何结果，所以应该不需要返回任何内容。但是同样地，与AsyncTask之间的契约表明，我们必须返回一个具有第三种数据类型的对象。因为该数据类型为void，所以我们返回的对象是null。

3. onProgressUpdate()方法

onProgressUpdate()方法如下所示：

```
@Override
protected void onProgressUpdate(String... item) {
  ((ArrayAdapter)getListAdapter()).add(item[0]);
}
```

onProgressUpdate()方法在UI线程上调用，我们需要执行一些操作来让用户知道我们正在加载这些字符串。在本例中，我们简单地将字符串添加到ArrayAdapter，所以它们会出现在列表末尾。

onProgressUpdate()方法接受一个String...vararg，因为这是类声明中的第二种数据类型。由于我们为每次调用publishProgress()只传递一个字符串，我们只需要检查vararg数组中的第一项。

4. onPostExecute()方法

下面是onPostExecute()方法：

```
@Override
protected void onPostExecute(Void unused) {
  Toast
    .makeText(AsyncDemo.this, "Done!", Toast.LENGTH_SHORT)
    .show();
}
```

onPostExecute()方法在UI线程上调用，我们需要执行一些操作来表明后台工作已完成。在实际的系统中，可能会解除某个ProgressBar或停止某个动画。这里，我们只是显示一个Toast。

因为我们选择了没有任何结果，所以应该不需要任何参数。但与AsyncTask的契约表明，我们必须接受一个具有第三种数据类型的参数。因为该数据类型是Void，所以我们的方法参数为

Void unused。

5. Activity

最后看一下 Activity：

```
new AddStringTask().execute();
```

为了使用 AddStringsTask，我们简单地创建了一个实例并对它调用 execute()。这会开始一系列事件，最终导致后台线程执行自己的工作。

如果 AddStringsTask 需要配置参数，我们就不能使用 Void 作为首选数据类型，构造函数将接受零个或多个具有既定类型的参数。这些值最终将被传递到 doInBackground()。

15.5 附加说明

后台线程尽管很可能使用 Android 的 Handler 系统，但并不是百利而无一害。后台线程不但会增加任务的复杂性，而且在可用内存、CPU 和电池电量方面的确具有一定的开销。

因此，使用后台线程时，有许多情况需要引起注意，包括以下情况。

- 在后台线程繁忙地工作时，用户可能与 Activity 的 UI 交互。如果后台线程所执行的工作由于用户输入而改变或变得无效，那么就需要就此状况与后台线程通信。Android 在 java.util.concurrent 包中包含了许多类，可帮助你安全地与后台线程通信。
- 在后台工作进行时，Activity 可能会被结束。例如，在开始一个 Activity 之后，用户可能有电话呼入，然后收到一条文本消息，紧接着需要查看一个联系人，所有这些都足以将一个 Activity 挤出内存。第 16 章将介绍 Android 将使 Activity 经历的各种事件。挂钩到合适的事件，并确保有机会时尽可能完全关闭后台进程。
- 如果消耗了大量 CPU 时间和电池电量而没有提供任何回馈，用户可能会生气。从战术上讲，这意味着要使用 ProgressBar 或其他方式让用户知道正在发生什么事情。从战略上讲，这意味着仍然需要有效地执行操作，后台线程并不是缓慢或无用代码的灵丹妙药。
- 可能在进行后台处理时遇到错误。例如，在从 Internet 收集信息时，手机可能失去连接。通过通知（将在第 31 章介绍）提醒用户出现的问题并关闭后台线程，这可能是最好的选择了。

第16章

16 处理Activity生命周期事件

我们知道，从总体而言 Android 设备就是手机。因此，某些特定活动相对而言更重要。例如，对用户而言，接电话可能比玩游戏更重要。另外，由于它是一部手机，其 RAM 可能比目前的台式机或笔记本电脑的要小。

由于手机的 RAM 有限，所以你有可能会发现 Activity 被结束的情况，因为其他 Activity 正在运行且系统需要你的 Activity 所使用的内存。可以将此视为生命周期的 Android 版本：一个 Activity 结束了，其他 Activity 才能够运行，等等。

不能假设 Activity 将在你认为它完成之前或者甚至在用户认为它完成之前会持续运行。这是 Activity 生命周期影响应用程序逻辑的一个例子，也可能是最重要的例子。

本章将介绍组成 Activity 生命周期的各种状态和回调，以及如何适当地与它们挂钩。

16.1 Activity 的状态

通常而言，一个 Activity 在任何时间都处于以下 4 种状态之一。

- 活跃：Activity 由用户启动，正在运行，并且在前台。这是我们过去常常想到的 Activity 的操作。
- 暂停：Activity 由用户启动，正在运行并且可见，但是一个通知或某个事物覆盖了屏幕的一部分。在这段时间，用户可以看到 Activity，但不能与之交互。例如，如果有电话呼入，用户可以接听或忽略它。
- 停止：Activity 由用户启动，正在运行，但被已启动或被切换到的其他 Activity 所隐藏。应用程序将无法直接向用户呈现任何有意义的内容，但可以通过通知（将在第 31 章介绍）的形式与用户通信。
- 死亡：Avtivity 从未启动（例如，刚刚重置了手机之后），或者 Activity 被终止了，可能是由于可用内存不足。

16.2 Activity 的生命周期

一个 Activity 在上一节介绍的 4 种状态之间转变时，Android 将使用本节介绍的方法调用该 Activity。一些转变可能导致多次调用一个 Activity，有时 Android 也会在不调用该 Activity 的情况下结束应用程序。这部分的内容条理不太清晰而且随时可能改变，所以在决定哪些事件值得注意以及哪些事件可以安全地忽略时，请密切关注官方 Android 文档以及这里的信息。

注意，对于所有这些方法，应该调用超类的方法版本，否则 Android 可能抛出异常。

16.2.1 onCreate()和 onDestroy()

在所有示例中，我们在所有 Activity 子类中实现了 onCreate()方法。此方法将在以下 3 种情形下调用。

- 当 Activity 首次启动时(例如自系统重新启动后)，将使用一个 null 参数调用 onCreate()。
- 如果 Activity 已经运行，并在随后的某个时刻被结束，那么将使用来自 onSaveInstanceState()的 Bundle 作为参数调用 onCreate()。
- 如果 Activity 已经运行，并且已将 Activity 设置为根据不同的手机状态(例如横向和纵向)提供不同的资源，那么将重新创建 Activity 并调用 onCreate()。资源将在第 20 章介绍。

无论以何种方式使用 Activity，在这里初始化 UI 并设置任何需要一次性完成的操作。

在生命周期的另一端，当 Activity 关闭时可能会调用 onDestroy()，这可能是因为 Activity 调用了 finish()（“完成” Activity），也可能是因为 Android 需要 RAM 并过早地关闭了该 Activity。请注意，如果对 RAM 的需求很紧急（例如有一个电话呼入），可能不会调用 onDestroy()，但仍然会关闭该 Activity。因此，onDestroy()通常用于完全释放在 onCreate()中获得的资源（如果有）。

16.2.2 onStart()、onRestart()和 onStop()

Activity 在前台运行可能是因为它是首次启动，也可能是因为在被隐藏（例如被另一个 Activity 或一个呼入的电话隐藏）之后被调回前台。在这些情况下都会调用 onStart()方法。

停止后重新启动 Activity 时，将调用 onRestart()方法。

相反，要停止 Activity 时，将调用 onStop()。

16.2.3 onPause()和 onResume()

在 Activity 初始启动、从停止状态重新启动之后，或者清除了一个弹出对话框（如一个呼

入的电话）之后，在 Activity 回到前台之前将调用 onResume()方法。可以在此时根据用户上一次查看 Activity 之后可能发生的操作刷新 UI。例如，如果轮询一个服务来查看对某些信息的更改（例如来自一个源的新条目），那么可以调用 onResume()来刷新当前视图，也可以适当地启动后台线程以更新视图（例如通过 Handler）。

相反，使用户远离你的Activity的任何操作(通常是激活另一个Activity)都将导致onPause()被调用。在这里，应该撤销在 onResume()中所做的任何操作，例如停止后台线程，释放所获得的独占访问资源（例如照相机）等。

调用 onPause()之后，Android 有权在任何时候结束 Activity 的进程。因此，不应依赖于会接收到任何后续事件。

16.3　优美的状态

通常，上述方法都是用于在通用应用程序级别上处理事物。(例如，在 onCreate()中连接 UI 中的上一部分，或者在 onPause()中关闭后台线程。)

但是，Android 的一个重要目标是实现无缝性。Android 可能根据内存的需求启动或关闭 Activity，但是在理想情况下，用户不应该知道发生了这些操作。例如，如果用户正在使用一个计算器，并在离开片刻之后继续使用该计算器，那么他应该看到最初使用的数字，除非他确实采取了某种操作关闭了计算器。

要实现这一点，Activity 需要能够保存应用程序实例的状态，并且要能尽快且高效地完成这一操作。因为 Activity 可能在任何时候被关闭，所以可能需要比我们的预期更为频繁地保存 Activity 的状态。这样，当 Activity 重新启动时，它应该获得以前的状态，以便将 Activity 恢复到以前的模样。

实例状态的保存由 onSaveInstanceState()处理。它提供了一个 Bundle，Activity 可以向 Bundle 填充它所需要的任何数据（例如计算器显示的数字）。此方法实现需要非常快，因此不要尝试绚丽的效果，只需要将数据放入 Bundle 并退出该方法。

该实例状态会在两个位置再次提供：onCreate()和 onRestoreInstanceState()中。在需要重新获得 Activity 的状态数据时，两种回调都是不错的选择，具体使用哪个由你决定。

第 17 章 创建 Intent 过滤器

到目前为止，本书的重点一直放在用户直接从手机启动程序打开的活动上。准备好活动，让用户能够查看，大部分情况下都是如此，这也是用户开始使用应用程序的主要方式。

但是，Android 系统的基础是很多松散耦合的组件。通过对话框、子窗口等在桌面 GUI 中实现的操作很可能都是独立的活动。尽管一个活动是“特殊的”，因为它出现在启动程序中，但其他活动都必须以某种方式连接。

这里所说的“某种方式”就是通过 Intent。

Intent 基本上是一个传递给 Android 的消息：“你好，我想执行某个操作！”描述“操作”的详细程度取决于具体的情况。有时你知道确切要做什么（例如，打开一个其他活动），而有时候却不知道。

理论上，Android 的全部用途都与 Intent 以及这些 Intent 的接收器有关。那么，现在让我们深入这些 Intent，以便创建更加复杂的应用程序，同时成为“Android 良好市民”。

17.1 你有什么意图

Tim Berners-Lee 先生构思 HTTP（Hypertext Transfer Protocol，超文本传输协议）时，设置了一个动词系统，并附加了 URL 形式的地址。地址表示资源，比如网页、图片或者服务器端程序。动词表示要完成的操作：GET 表示获取，POST 表示发送表格数据以供处理等。

Intent 与此类似，它表示为动作和上下文。Android Intent 中的上下文动作和组件比 HTTP 动词和资源要多，但概念是相同的。就像 Web 浏览器知道如何处理动词-URL 对一样，Android 知道如何查找可以处理给定 Intent 的活动或其他应用程序逻辑。

17.1.1 Intent 组成

Intent 有两个最为重要的组成部分，一个是动作，另一个是 Android 所谓的数据。它们与 HTTP

的动词和 URL 是完全类似的：动作就是动词，数据就是 URI，比如 content://contacts/ people/1 表示联系人数据库中的某个联系人。动作是常量，比如 ACTION_VIEW（生成查看器以查看资源）、ACTION_EDIT（编辑资源）或者 ACTION_PICK（在给定表示集合的 URI 中选择一个可用的项，比如 content://contacts/people）。

如果你创建了一个 Intent，其中组合了 ACTION_VIEW 和 content://contacts/people/1 这种内容 URI，并将该 Intent 传递给 Android，那么 Android 将知道去查找并打开一个能查看该资源的活动。

除了动作和数据 URI 之外，你还可以在 Intent 中放置其他条件（表示为 Intent 对象），如下所示。

- 类别："主要的"活动属于 LAUNCHER 类别（category），表示它应该显示在启动菜单上。其他活动可以归入 DEFAULT 或 ALTERNATIVE 类别。
- MIME 类型：如果不知道集合 URI，这可以表示你想操作的资源的类型。
- 组件：这是一个可以接收此 Intent 的活动类。用这种方式使用组件（component）就消除了使用其他 Intent 属性的必要。但是，它会让 Intent 变得更加脆弱，因为它需要特定的实现。
- Extras：你希望随 Intent 一起传递给接收器的其他信息 Bundle，接收器也许要利用这些信息。接收器使用哪部分信息由它自己决定，而且该信息的记录非常清晰（希望如此）。

你可以在 Android SDK 文档的 Intent 类部分中找到标准动作和类别列表。

17.1.2　Intent 路由

正如上一部分所述，如果你指定 Intent 的目标组件，Android 不会质疑 Intent 路由的目的地，它将启动指定的活动。如果目标 Intent 位于你的应用程序中，就可以采用这种方式。不推荐向其他应用程序发送 Intent。

一般说来，组件名称被视为应用程序私有，且名称可能发生更改。标识你希望第三方代码提供的服务时，内容 URI 模板和 MIME 类型是首选方式。

如果没有指定目标组件，Android 必须指出哪些活动（或者其他 Intent 接收器）有资格接受 Intent。注意是"哪些活动"，因为编写较为宽泛的 Intent 可能解析为多个活动。这就是 Intent（意图，这里用了双关语），本章后文将介绍这一点。这种路由方法称为隐式路由（implicit routing）。

基本上有 3 个条件，有资格用于给定 Intent 的给定活动必须同时符合这 3 个条件：

- 活动必须支持指定的动作；

- 活动必须支持已声明的 MIME 类型（如有提供）；
- 活动必须支持 Intent 中指定的所有类别。

这样一来，你需将 Intent 的具体程度设置为足以找到正确的接收器，但不用更具体了。本章后文的一些示例可能会让你更清楚此内容。

17.2 叙述 Intent

所有希望通过 Intent 获得通知的 Android 组件都必须声明 Intent 过滤器，以便 Android 知道哪些 Intent 应该到达该组件。为此，你需要向 AndroidManifest.xml 文件添加 `intent- filter` 元素。

所有示例项目都有已定义的 Intent 过滤器，Android 应用程序构建脚本（`activityCreator` 或者 IDE 对应物）中免费提供了过滤器。它们类似以下代码：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.skeleton">
    <application>
        <activity android:name=".Now" android:label="Now">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

请注意 `activity` 元素下方的 `intent-filter` 元素。在此，我们声明此活动：

- 是此应用程序的主要活动；
- 属于 `LAUNCHER` 类别，表示它在 Android 主菜单中有一个图标。

因为此活动是应用程序的主要活动，在某人从主菜单中选择该应用程序时，Android 知道这是它应该启动的组件。

Intent 过滤器中可以有多个动作或多个类别。这表示相关组件（例如活动）可处理多个不同的 Intent。

实际上，你还可以有次要（non-main）活动，指定所处理数据的 MIME 类型。这样，如果 Intent 以该 MIME 类型为目标（要么直接指定，要么通过引用该类型内容的 URI 间接指定），Android 将知道该组件处理此类数据。

例如，你可以将活动声明如下：

```
<activity android:name=".TourViewActivity">
  <intent-filter>
```

```
      <action android:name="android.intent.action.VIEW" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:mimeType="vnd.android.cursor.item/vnd.commonsware.tour" />
    </intent-filter>
  </activity>
```

此活动将由 Intent 启动，请求查看一个表示内容 vnd.android.cursor.item/vnd.commonsware.tour 部分的 URI。该 Intent 可能来自同一个应用程序的另一个活动（例如，此应用程序的 MAIN 活动），也可能来自于另一个 Android 应用程序中的活动，前提是它知道此活动处理的 URI。

17.3　缩小接收器范围

在之前的例子中，Intent 过滤器设置在活动上。有时候，将 Intent 与活动绑定并不恰当，比如以下情况。

- 有些系统事件可能导致触发服务而不是活动中的操作。
- 有些事件可能需要根据不同环境启动不同的活动，其条件并不仅限于 Intent 本身，而且还受到其他状态的限制。（例如，如果你有 Intent X，数据库有 Y，然后启动活动 M；如果数据库没有 Y，则启动活动 N。）

为了处理这些情况，Android 提供了实现 BroadcastReceiver 接口的类作为 Intent 接收器。Intent 接收器是可以任意处理的对象，能接收 Intent（特别是广播 Intent）并采取行动。动作通常涉及启动其他 Intent 来触发活动、服务或者其他组件中的逻辑。

BroadcastReceiver 接口只有一个方法：onReceive()。Intent 接收器实现该方法，并对传入的 Intent 做任何想做的操作。要声明 Intent 接收器，可以向 AndroidManifest.xml 文件添加一个接收器元素：

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

Intent 接收器只有在处理 onReceive()时才会激活。只要该方法返回，接收器实例就被垃圾收集，不能重用。这意味着 Intent 接收器某种程度上将受到限制，通常是为避免涉及回调的任何操作。例如，它们无法绑定到服务，无法打开对话框。

如果 BroadcastReceiver 在某个生命周期较长的组件上实现，比如活动或服务，那么情况将有所不同。在这种情况下，Intent 接收器的生命周期与“宿主”一样长（例如，直到活动冻结为止）。但是在这种情况下，你不能通过 AndroidManifest.xml 声明 Intent 接收器。相反，需要在 Activity 的 onResume()回调上调用 registerReceiver()来声明对 Intent 的关注，不再需要这些 Intent 时，通过 Activity 的 onPause()调用 unregisterReceiver()。

17.4 暂停警告

使用 Intent 对象传递任意消息时有一个小问题，这种方式只在接收器活动时才能正常运行。引用 BroadcastReceiver 文档中的话就是：

> 如果在 Activity.onResume()实现中注册接收器，应该使用 Activity.onPause()注销。（在暂停时不会接收到 Intent，这将减少不必要的系统开销。）不要使用 Activity.onSaveInstanceState()注销，因为当用户在历史栈中回退时不会调用该方法。

因此，只有在以下情况下，你才可以使用 Intent 框架作为传递任意消息的总线：

- 接收器不关心它是否错过了消息，因为它没有激活；
- 提供某种方式让接收器“捕获”它未激活时错过的消息。

在第 29 章和第 30 章，你将学习一些有关第一种情况的示例，其中接收器（服务客户端）将使用基于 Intent 的消息（这些消息可用时），但是在客户端没有激活时不需要。

第18章 启动活动和子活动

我们知道，Android UI体系结构背后的理论是，开发人员应该将应用程序分解为不同的活动。每个活动都实现为一个Activity，每个活动都可以通过Intent访问，“主”活动应该由Android启动程序启动。例如，日历应用程序的活动包括查看日历、查看单个事件、编辑事件（包括添加新事件），等等。

这表明，活动可以在另一个活动上启动。例如，如果有人从查看日历活动中选择了某个事件，那么应该为该事件展示视图-事件活动。因此，你需要让视图-事件活动能够启动，并展示特定的事件（用户选择的事件）。本章将介绍如何做到这一点。

说明 本章假定你了解要启动哪个活动，可能是因为该活动是应用程序中的另一个活动，也可能是因为有一个用于执行某些操作的内容URI，你希望用户能够用它执行一些操作，但却无法预知具体做什么。这种情况下需要更多的高级处理，我在*The Busy Coders Guide to Advanced Android Development* (CommonsWare, 2009)一书中对此有详细介绍。

18.1 对等活动和子活动

决定启动活动时，你需要回答一个关键问题：你的活动需要了解已启动的活动何时终止吗？

例如，你希望启动一个活动来收集所连接的某个网页服务的验证信息[可能需要使用OpenID（http://openid.net/）进行验证，这样才能使用的OAuth服务（http://oauth.net/）]。在这种情况下，主活动需要了解何时完成了验证，以便开始使用网页服务。

另一方面，想象Android中的电子邮件应用程序。当用户选择查看附件时，你和用户都未必希望主活动知道用户何时查看完了附件。

在第一种情况中，被启动的活动很明显从属于启动它的活动。此时，可能需要将被启动的活动作为子活动启动，这也意味着，在子活动完成时，启动它的活动将获得通知。

在第二种情况中，被启动的活动更像是一个对等活动，所以应该将这个子活动作为常规活动启动。子活动完成之后活动将不会获得通知，而活动实际上也不需要知道这一点。

18.2 启动

启动一个活动包括两方面的内容，即 Intent 和启动方式。

18.2.1 制作 Intent

在上一章中我们讨论过，Intent 包含一个请求，用于 Android 的某些活动或其他 Intent 接收器执行某种操作。

如果你希望启动的活动是自己的，那么创建显式 Intent 是最简单的方法，只需要指定要启动的组件即可。例如，在活动中可以这样创建 Intent：

```
new Intent(this, HelpActivity.class);
```

这表示你要启动 HelpActivity。该活动需要在 AndroidManifest.xml 文件中指定，不一定需要 Intent 筛选器，因为此处是直接请求它。

你也可以同时使用 URI 和 Intent，请求特定的操作：

```
Uri uri=Uri.parse("geo:"+lat.toString()+","+lon.toString());
Intent i=new Intent(Intent.ACTION_VIEW, uri);
```

该代码中，你有某个位置的经纬度（分别为 lon 和 lat），类型为 Double，构建了一个 geo 模式 URI，并创建了一个请求查看此 URI 的 Intent（ACTION_VIEW）。

18.2.2 进行调用

有了 Intent 之后，需要将其传递到 Android 并启动子活动。此时有两个选择，如下所示。

- 最简单的选择是使用 Intent 参数调用 startActivity()。这会让 Android 查找最匹配的活动，并将 Intent 传递给该活动进行处理。子活动完成后，活动不会得到通知。
- 可以调用 startActivityForResult()，传入 Intent 和编号（对于调用活动是唯一的）。Android 将查找最匹配的活动，并将 Intent 传递给该活动。子活动完成后，将通过 onActivityResult()回调通知活动。

如前文所述，使用 startActivityForResult()时可以实现 onActivityResult()回调，在子活

动完成工作之后通知活动。该回调接受为 startActivityForResult()提供的唯一编号，这样就可以确定哪个子活动已经完成。你还将获得以下结果。

- 子活动调用 setResult()产生的结果代码。通常是 RESULT_OK 或 RESULT_CANCELLED，尽管你也可以创建自己的返回代码（选用从 RESULT_FIRST_USER 开始的值）。
- 可选的 String，包含某些结果数据，可能是某些内部或外部资源的 URL。例如，ACTION_PICK Intent 通常通过此数据字符串返回选定的内容位。
- 可选的 Bundle，包含除结果代码和数据字符串之外的其他信息。

为了演示如何启动对等活动，请查看 Activities/Launch 示例应用程序。XML 的布局非常简单明了：用于维度和经度的两个字段，以及一个按钮。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <TableLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:stretchColumns="1,2"
  >
    <TableRow>
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="2dip"
        android:paddingRight="4dip"
        android:text="Location:"
      />
      <EditText android:id="@+id/lat"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:cursorVisible="true"
        android:editable="true"
        android:singleLine="true"
        android:layout_weight="1"
      />
      <EditText android:id="@+id/lon"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:cursorVisible="true"
        android:editable="true"
        android:singleLine="true"
        android:layout_weight="1"
      />
    </TableRow>
  </TableLayout>
  <Button android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Show Me!"
  />
</LinearLayout>
```

按钮的 OnClickListener 只获取维度和经度，将其放入 geo 模式 URI，然后启动活动。

```
package com.commonsware.android.activities;

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

public class LaunchDemo extends Activity {
  private EditText lat;
  private EditText lon;

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

    Button btn=(Button)findViewById(R.id.map);
    lat=(EditText)findViewById(R.id.lat);
    lon=(EditText)findViewById(R.id.lon);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        String _lat=lat.getText().toString();
        String _lon=lon.getText().toString();
        Uri uri=Uri.parse("geo:"+_lat+","+_lon);

        startActivity(new Intent(Intent.ACTION_VIEW, uri));
      }
    });
  }
}
```

如图 18-1 所示，该活动很简单。

如果填入一个位置（例如，维度 38.889 1 和经度−77.049 2）并单击按钮，得到的地图将更加有趣，如图 18-2 所示。

注意，这是内置的 Android 地图活动，我们没有创建自己的活动来显示此地图。在第 33 章中，你将了解如何在自己的活动中创建地图，以便更好地控制地图的显示方式。

说明 在模拟器中，这个示例应用程序在Android 2.0 AVD上可能无法运行，因为AVD似乎缺少Maps应用程序。

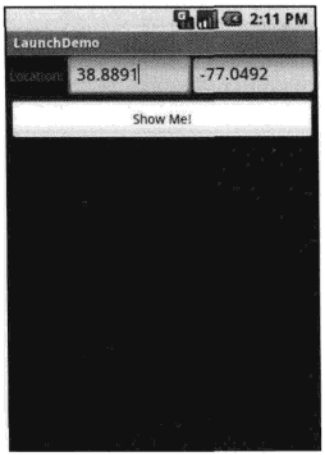


图 18-1　LaunchDemo 示例应用程序，其中填入了位置数据

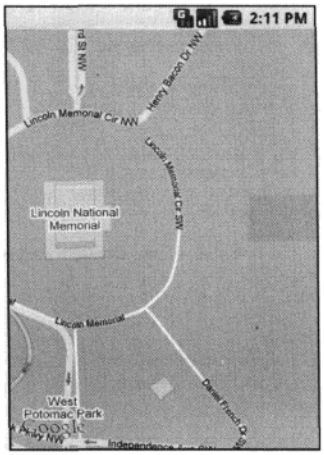


图 18-2　LaunchDemo 启动的地图，展示了华盛顿地区的林肯纪念碑

18.3　多标签浏览

现代桌面 Web 浏览器的一个主要特性就是多标签浏览，即一个浏览器窗口可以使用一系列标签展示多个页面。在手机中，这没有太大的意义，因为标签本身可能占据很大的屏幕空间。但是在本书中，这些小问题都无法阻止我们，我们来演示一个使用 TabActivity 和 Intent 对象的多标签浏览器。

在第 9 章我们说过，标签内容可以是 View，也可以是 Activity。如果希望使用 Activity 作为标签内容，则需要提供一个将启动所需 Activity 的 Intent；Activity 的标签管理框架将把该 Activity 的 UI 放入标签对应的窗口。

你可能本能地按照我们在上例中使用 geo: URI 的方式使用 http: URI：

```
Intent i=new Intent(Intent.ACTION_VIEW);
i.setData(Uri.parse("http://commonsware.com"));
```

通过这种方式，你可以使用内置的浏览器应用程序，并获得它提供的所有功能。

可惜这样做不行。你不能在标签中托管其他应用程序的活动；出于安全原因，只允许托管自己的活动。

因此，我们去掉第 13 章中的 WebView 演示内容，使用以下内容并作为 Activities/Intent Tab 重新打包。

以下是主活动（托管 TabView）的源代码：

```
public class IntentTabDemo extends TabActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    TabHost host=getTabHost();

    host.addTab(host.newTabSpec("one")
            .setIndicator("CW")
            .setContent(new Intent(this, CWBrowser.class)));
    host.addTab(host.newTabSpec("two")
            .setIndicator("Android")
            .setContent(new Intent(this, AndroidBrowser.class)));
  }
}
```

从中可以看出，我们使用 TabActivity 作为基类，因此不需要自己的布局 XML，原因在于 TabActivity 可以提供。我们要做的只是访问 TabHost 并添加两个标签，每个标签指定直接引用另一个类的 Intent。在本例中，我们的两个标签分别托管 CWBrowser 和 AndroidBrowser。

这些活动是对之前的浏览器演示内容进行了简单修改的结果：

```
public class CWBrowser extends Activity {
  WebView browser;

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

    browser=new WebView(this);
    setContentView(browser);
    browser.loadUrl("http://commonsware.com");
  }
}
public class AndroidBrowser extends Activity {
  WebView browser;

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

    browser=new WebView(this);
    setContentView(browser);
    browser.loadUrl("http://www.android.com/");
  }
}
```

它们只是将另一个 URL 加载到浏览器中：一个是 CommonsWare 主页，另一个是 Android 主页。

提示　对每个目标页面使用不同的子类很浪费资源。你可以打包URL并在Intent中作为“extra”打开，并使用该Intent生成一个一般用途的BrowserTab活动，该活动可以从Intent extra中读取URL并使用它。

得到的 UI 展示了 Android 中多标签浏览的情况，如图 18-3 和图 18-4 所示。

图 18-3　IntentTabDemo 示例应用程序，展示第一个标签内容

图 18-4　IntentTabDemo 示例应用程序，展示第二个标签内容

第 19 章 处理旋转 19

有些 Android 手机（如 T-Mobile G1）提供了一个滑盖键盘，可以触发将屏幕从纵向旋转为横向的事件。还有些手机可以使用加速计来感应屏幕旋转，像 iPhone 一样。因此，可以很合理地认为，用户有时需要在纵向与横向之间进行切换。

在本章中您将看到，Android 有很多方式可以帮助您处理屏幕旋转，让应用程序可以恰当地针对这两个方向进行处理。但是要认识到这些工具只是帮助检测和管理旋转过程，你仍需确保在每个方向上都有看起来不错的布局。

19.1 销毁问题

默认情况下，如果手机配置中出现了可能影响资源选择的更改，Android 将在下一次显示正在运行或暂停的 Activity 时将其销毁并重新创建。很多不同的配置更改都会出现这种情况（例如，语言选择的更改），而这很可能会对旋转产生不利的影响，因为方向的更改可能导致要加载不同的资源（例如布局）。

此处的关键在于：这是默认的行为。它对于一个或多个 Activity 也许是最好的行为。但你的确可以控制该问题，甚至可以自定义 Activity 如何响应方向更改或类似配置的切换。

19.2 异同

默认情况下，Android 会针对旋转销毁并重建 Activity，因此你可能只需挂钩到 onSaveInstanceState()上，在由于其他原因销毁 Activity 时（例如内存不足），你也需要挂钩到 onSaveInstanceState()上。在 Activity 中实现该方法，并在所提供的 Bundle 中填入足够的信息，就可以返回到当前状态。然后，在 onCreate()（或者 onRestoreInstanceState()，如果你更喜欢）中，从 Bundle 挑选数据来恢复 Activity。

为了演示该问题，让我们看看 Rotation/RotationOne 项目。该项目以及本章中的另一个项目

使用一对 main.xml 布局用于横向模式：一个位于 res/layout/，另一个位于 res/layout- land/。以下是纵布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <Button android:id="@+id/pick"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:text="Pick"
    android:enabled="true"
  />
  <Button android:id="@+id/view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:text="View"
    android:enabled="false"
  />
</LinearLayout>
```

以下是类似的横向布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <Button android:id="@+id/pick"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:text="Pick"
    android:enabled="true"
  />
  <Button android:id="@+id/view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:text="View"
    android:enabled="false"
  />
</LinearLayout>
```

基本上，布局中包含一对按钮，每个按钮都将占据半个屏幕。在纵向模式中，按钮是上下排列的；在横向模式中，它们是并排的。

如果创建一个项目，使用这两种布局并进行编译，应用程序将运行良好——旋转（在模拟器中按 Ctrl+F12 键）将导致布局更改。虽然按钮缺少状态，如果使用其他部件（例如 EditText），你仍可能会发现该 Android 带有某些部件状态（例如，在 EditText 中输入的文本）。

Android 无法自动处理部件之外保持的内容。

该应用程序允许你挑选一个联系人，然后通过不同的按钮查看联系人信息。View 按钮只有在选择了联系人之后才会被启用。

让我们看看如何使用 onSaveInstanceState()处理该问题：

```
public class RotationOneDemo extends Activity {
  static final int PICK_REQUEST=1337;
  Button viewButton=null;
  Uri contact=null;

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

    Button btn=(Button)findViewById(R.id.pick);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        Intent i=new Intent(Intent.ACTION_PICK,
                            Contacts.CONTENT_URI);

        startActivityForResult(i, PICK_REQUEST);
      }
    });

    viewButton=(Button)findViewById(R.id.view);

    viewButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        startActivity(new Intent(Intent.ACTION_VIEW, contact));
      }
    });

    restoreMe(savedInstanceState);

    viewButton.setEnabled(contact!=null);
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode,
                                  Intent data) {
    if (requestCode==PICK_REQUEST) {
      if (resultCode==RESULT_OK) {
        contact=data.getData();
        viewButton.setEnabled(true);
      }
    }
  }

  @Override
  protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    if (contact!=null) {
```

```
      outState.putString("contact", contact.toString());
    }
  }

  private void restoreMe(Bundle state) {
    contact=null;

    if (state!=null) {
      String contactUri=state.getString("contact");

      if (contactUri!=null) {
        contact=Uri.parse(contactUri);
      }
    }
  }
}
```

大体看来，它就像一个普通的 Activity。(它本来就是。)最初，“模型”（URI 指定的 contact）为 null。它被设为 ACTION_PICK 子活动的结果。其字符串表示形式使用 onSaveInstance State()保存，使用 restoreMe()恢复（由 onCreate()调用）。如果联系人不为 null，则启用 View 按钮用于查看所选的联系人。

它看起来就像你所期望的那样，如图 19-1 和图 19-2 所示。

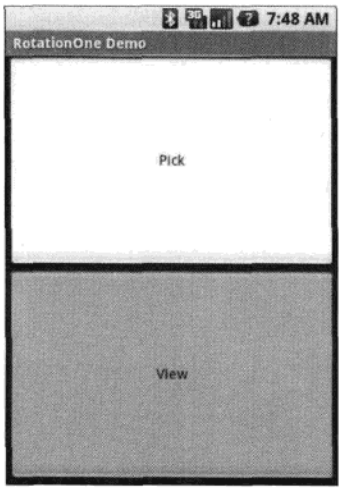


图 19-1　纵向模式下的 RotationOne 应用程序

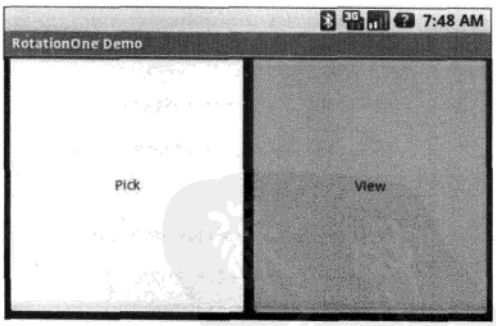


图 19-2　横向模式下的 RotationOne 应用程序

此实现的好处在于，它不仅可以处理旋转，还能处理很多系统事件，例如由于内存不足由 Android 关闭程序的问题。

可以尝试在 onCreate()中注释掉 restoreMe()调用并运行应用程序。在旋转模拟器或手机时，你将看到应用程序“忘记”了一个在一个方向中所选的联系人。

说明 本章的所有示例仅能用于Android 2.0及以上的系统，因为它们使用更新的方法从Contacts内容提供程序中挑选联系人。（将在第26章中讨论。）

19.3 更多保存

onSaveInstanceState()的问题在于，它受限于Bundle。这是因为在整个进程可能被终止时（如内存不足），才将使用此回调，因此要保存的数据必须可以序列化，并且与正在运行的进程没有依赖关系。

对于某些Activity，这个限制不是什么问题。但对其他Activity则存在问题。以在线聊天为例，你没有办法在Bundle中存储套接字，因此在默认情况下，需要断开与聊天服务器的连接然后再重新建立连接。这不仅对性能是一个重大影响，也会影响到聊天本身，例如出现在聊天记录中的断开连接和重新连接。

解决这个问题的方法之一是，对于旋转之类的“轻量级”改变，使用onRetainNonConfigurationInstance()取代onSaveInstanceState()。Activity的onRetainNonConfigurationInstance()回调可以返回一个Object，稍后可以通过getLastNonConfigurationInstance()检索它。Object可以是你需要的任何对象。通常，它是某种保存Activity状态的“上下文”对象，例如正在运行的线程、打开的套接字，等等。Activity的onCreate()可以调用getLastNonConfigurationInstance()。如果得到非null的响应，则拥有了套接字、线程等内容。最大的限制在于你不能将可能引用要交换的资源的内容放入所保存的上下文中，例如从资源加载的Drawable。

下面看看Rotation/RotationTwo示例项目，该项目使用此方法处理旋转。布局以及外观都与Rotation/RotationOne相同。稍有区别的是Java代码：

```
public class RotationTwoDemo extends Activity {
  static final int PICK_REQUEST=1337;
  Button viewButton=null;
  Uri contact=null;

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

    Button btn=(Button)findViewById(R.id.pick);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        Intent i=new Intent(Intent.ACTION_PICK,
                            Contacts.CONTENT_URI);

        startActivityForResult(i, PICK_REQUEST);
```

```
    }
  });

  viewButton=(Button)findViewById(R.id.view);

  viewButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      startActivity(new Intent(Intent.ACTION_VIEW, contact));
    }
  });

  restoreMe();

  viewButton.setEnabled(contact!=null);
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
                                Intent data) {
  if (requestCode==PICK_REQUEST) {
    if (resultCode==RESULT_OK) {
      contact=data.getData();
      viewButton.setEnabled(true);
    }
  }
}

@Override
public Object onRetainNonConfigurationInstance() {
  return(contact);
}

private void restoreMe() {
  contact=null;

  if (getLastNonConfigurationInstance()!=null) {
    contact=(Uri)getLastNonConfigurationInstance();
  }
}
}
```

在本例中，我们覆写了 onRetainNonConfigurationInstance()，返回联系人的实际 URI 而不是其字符串表示形式。restoreMe()调用 getLastNonConfigurationInstance()，如果后者返回值非 null，则我们将其返回对象用作联系人并启用 View 按钮。

此处的优势在于，我们围绕 URI 而不是字符串表示形式进行传递。在本例中，节省的开销并不多。但是实际情况可能要复杂得多，包括线程、套接字以及其他无法打包到 Bundle 的内容。

但是，此方法可能对应用程序存在很多干扰。假如你要开发一个实时游戏，如第一人称射击游戏。那么，在 Activity 被销毁并重新创建时用户体验到的“停顿”足以让他们被击毙，用户是绝对无法容忍这种情况的。虽然在 T-Mobile G1 上这应该不成问题，因为旋转需要滑动打开键盘，而在游戏中不太可能出现这种情况，但其他手机可能仅根据朝向（由加速计确定）进行旋转。在这种情况下，你应该将 Android 设为手动旋转，声明不需要框架的任何协助，如下文所述。

19.4 DIY 旋转

要自行处理旋转，请执行以下操作。

(1) 在 AndroidManifest.xml 文件中放入 android:configChanges 条目，列出要自行处理的配置更改，而不是要 Android 帮助你处理的配置更改。

(2) 在 Activity 中实现 onConfigurationChanged()，在出现 android:configChanges 中列出的某个配置更改时将调用它。

现在，对于任何配置更改，你可以绕过整个 Activity 销毁过程，一个简单的回调便可让你知道这个更改。

为了更加直观，请查看 Rotation/RotationThree 示例应用程序。布局仍然相同，因此应用程序的外观与前两个示例一样。但是，Java 代码明显不同，因为我们不再关心如何保存状态，只关注更新 UI 以处理布局。

首先，我们需要对配置文件做一点小改动：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.commonsware.android.rotation.three"
      android:versionCode="1"
      android:versionName="1.0.0">
  <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="6"
    />
    <application android:label="@string/app_name"
        android:icon="@drawable/cw">
        <activity android:name=".RotationThreeDemo"
                  android:label="@string/app_name"
                  android:configChanges="keyboardHidden|orientation">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

此处我们在文件中声明将自行处理 keyboardHidden 和 orientation 配置更改。这已经包括了所有导致旋转的原因，无论是滑开键盘还是机身旋转。注意这是对 Activity，而不是应用程序设置的。如果有多个 Activity，则需要使用本章介绍的内容确定每个活动的战略。

此项目的 Java 代码如下：

```
public class RotationThreeDemo extends Activity {
  static final int PICK_REQUEST=1337;
  Button viewButton=null;
  Uri contact=null;
```

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

  setupViews();
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
                                Intent data) {
  if (requestCode==PICK_REQUEST) {
    if (resultCode==RESULT_OK) {
      contact=data.getData();
      viewButton.setEnabled(true);
    }
  }
}

public void onConfigurationChanged(Configuration newConfig) {
  super.onConfigurationChanged(newConfig);

  setupViews();
}

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

  Button btn=(Button)findViewById(R.id.pick);

  btn.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      Intent i=new Intent(Intent.ACTION_PICK,
                          Contacts.CONTENT_URI);

      startActivityForResult(i, PICK_REQUEST);
    }
  });

  viewButton=(Button)findViewById(R.id.view);

  viewButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      startActivity(new Intent(Intent.ACTION_VIEW, contact));
    }
  });

  viewButton.setEnabled(contact!=null);
 }
}
```

onCreate()实现将大部分逻辑委托给 setupViews()方法，后者将加载布局并设置按钮。此逻辑可以分解出自己的方法，因为它也在 onConfigurationChanged()中被调用。

19.5　强制解决问题

在前3节中，我们介绍了处理旋转事件的方法。当然还有一种根本的替代方法：通知 Android

禁用 Activity 旋转功能。如果 Activity 旋转功能不可用，也就不必担心如何编写代码处理旋转问题了。

要禁用 Android 中 Activity 的旋转功能，向 AndroidManifest.xml 文件添加 android:screenOrientation = "portrait"（或者"landscape"，随你喜欢）即可，如下所示（摘自 Rotation/RotationFour 示例项目）：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.commonsware.android.rotation.four"
      android:versionCode="1"
      android:versionName="1.0.0">
  <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="6"
    />
    <application android:label="@string/app_name"
        android:icon="@drawable/cw">
        <activity android:name=".RotationFourDemo"
                  android:screenOrientation="portrait"
                  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>
```

这会应用到每个 Activity，因此必须决定对于哪些 Activity 需要打开旋转功能。

现在，你的 Activity 已经锁定为你指定的方向，无论你做什么都无法旋转。图 19-3 和图 19-4

图 19-3 纵向模式下的 RotationFour 应用程序

展示了前3节中展示过的活动，但是使用了上文的配置文件，并使用模拟器设置了纵向和横向。注意，UI没有移动，仍然是纵向模式。

注意，Android仍然销毁并重新创建Activity，即使你已经将方向设置为此处展示的特定值。如果你希望避免这一点，还需要在配置文件中设置android:configChanges，如本章前文所述。

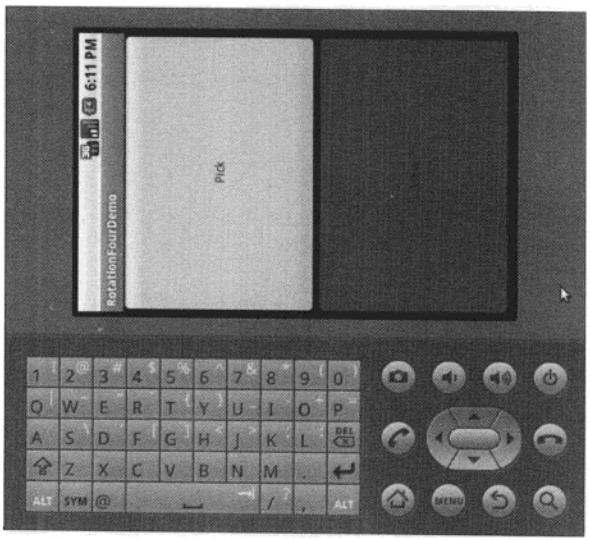


图19-4　横向模式下的RotationFour应用程序

19.6　综述

本章中展示的所有场景都假定你通过打开手机键盘(或在模拟器上按Ctrl+F12键)旋转屏幕。当然，这是Android应用程序的标准做法。但是，我们尚未考虑iPhone的情况。

你可能看过iPhone的商业广告，只需要转动手机就可以转动屏幕。有些Android手机也可以实现这一点，例如HTC Magic。还有一些设备则无法实现该行为，其屏幕只能根据键盘是否打开进行旋转。

但是，即使是对符合最后一种情况的手机也可以轻松地更改此行为，这样就可根据手机的朝向进行旋转。只需要在AndroidManifest.xml文件中添加android:screenOrientation = "sensor"即可。(摘自Rotation/RotationFive示例项目。)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.commonsware.android.rotation.five"
```

```
      android:versionCode="1"
      android:versionName="1.0.0">
    <uses-sdk
        android:minSdkVersion="5"
        android:targetSdkVersion="6"
    />
    <application android:label="@string/app_name"
        android:icon="@drawable/cw">
        <activity android:name=".RotationFiveDemo"
                  android:screenOrientation="sensor"
                  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>
```

在本例中，传感器通知 Android 你希望让加速计控制屏幕方向，以便机身方向的变化可以控制屏幕旋转。

至少在 T-Mobile G1 中，这似乎只有在从传统的纵向转变为传统的横向时有效——逆时针旋转 90 度。顺时针旋转 90 度屏幕没有变化。

还要注意，此设置禁用了键盘触发旋转事件。让手机处于纵向，如果滑出键盘，在普通的 Android 活动中屏幕将旋转；在 `android:screenOrientation = "sensor"`活动中，屏幕则不会旋转。

第20章 处理资源

资源是 Java 源代码外部保存的静态信息。在本书的示例中，你已经见过了一种资源——布局。在本章中，你将学习许多其他资源，例如图像和字符串，以及如何在 Android 应用程序中充分利用这些资源。

20.1 资源

在 Android 项目布局中，资源以文件的形式存储在 res/目录中。除了 raw 资源（res/raw/）之外，系统会为你解析其他所有类型的资源，这可能通过 Android 的打包系统，也可能通过手机或模拟器上的 Android 系统来完成。例如，通过布局资源（res/layout/）布局 Activity 的 UI 时，不需要自己解析布局 XML；Android 将为你处理。

除了布局资源（第 4 章中已介绍）和动画资源（第 9 章中已介绍）之外，还有一些其他类型的资源，包括：

- 图像（res/drawable/），用于在用户界面中放入静态图标或其他图片；
- Raw（res/raw/），用于任何对应用程序有用但对 Android 框架不一定有用的文件；
- 字符串、颜色、数组和维度（res/values/），用于指定常量符号名称，让它们与其他代码有区别（例如，用于国际化和本地化）；
- XML（res/xml/），包含数据和结构的静态 XML 文件。

20.2 字符串理论

一般来说，最好让标签和其他文本处于应用程序主源代码之外。这样做特别有助于进行国际化和本地化，20.6 节将介绍这一点。即使不会将字符串转换为其他语言的字符串，如果所有字符都位于一个位置，则很容易进行修正，而字符串分布在源代码的各处时情况就不是这样了。

Android 支持常规的不变字符串以及字符串格式，字符串有一些占位符用于动态插入的信

息。首先，Android 支持简单的文本格式化，称为样式文本，以便在常规文本中混杂粗体和斜体内容。

20.2.1 纯文本字符串

一般来说，使用纯文本字符串只需要 res/values 目录中的一个 XML 文件（通常名为 res/values/strings.xml），根元素为 resources，希望编码为资源的每个字符串都有一个 string 子元素。String 元素包含 name 特性，这是此字符串唯一的名称，还有一个文本元素，包含字符串的文本。

```
<resources>
  <string name="quick">The quick brown fox...</string>
  <string name="laughs">He who laughs last...</string>
</resources>
```

需要留意的地方在于字符串值是否包含引号（"）或者单引号（'）。如果包含，需要对这些值进行转义——前置一个反斜杠（例如，These are the times that try men\'s souls）。如果只有一个单引号，可以使用引号将值括起来（例如，"These are the times that try men's souls."）。

然后，可以从布局文件中引用此字符串（如@string/...，其中省略号表示唯一的名称，例如 @string/laughs）。此外，还可以使用 getString()和字符串资源的资源 ID，用 Java 代码来获取字符串，其中资源 ID 是前缀为 R.string 的唯一名称（例如，getString(R.string.quick)）。

20.2.2 字符串格式

与 Java 语言的其他实现一样，Android 的 Dalvik 虚拟机支持字符串格式。此处的字符串包含一些占位符，表示在运行时要使用可变信息替换的数据（例如，My name is %1$s）。存储为资源的纯文本字符串可以用作字符串格式：

```
String strFormat=getString(R.string.my_name);
String strResult=String.format(strFormat, "Tim");
((TextView)findViewByld(R.id.some_label)).setText(strResult);
```

20.2.3 样式文本

如果需要真正的富文本，应该有包含 HTML 的 raw 资源，然后将其放入 WebKit 部件。但是，对于使用<b>、<i>和<u>的轻量级 HTML 格式化，可以只使用字符串资源。问题在于必须转义 HTML 标记，而不是按一般情况处理：

```
<resources>
  <string name="b">This has &lt;b&gt;bold&lt;/b&gt; in it.</string>
  <string name="i">Whereas this has &lt;i&gt;italics&lt;/i&gt;!</string>
</resources>
```

访问的方式与纯文本字符串相同，但 getString()调用的结果确实是一个支持 android.text.Spanned 接口的对象：

```
((TextView)findViewById(R.id.another_label))
            .setText(getString(R.string.b));
```

20.2.4　样式字符串格式

处理样式字符串格式时样式文本有些棘手，因为 String.format()格式化命令对 String 对象有效，而对 Spanned 对象无效。如果希望有样式字符串格式，解决方法如下。

(1) 对字符串资源中的尖括号进行实体转义（例如，this is %1$s）。

(2) 按一般情况检索字符串资源，尽管此时它没有样式（例如，getString(R.string.funky_format)）。

(3) 生成格式结果，确保转义你替换的任何字符串值，以防它们包含尖括号或&符号。

```
String.format(getString(R.string.funky_format),TextUtils.htmlEncode(strName));
```

(4) 通过 Html.fromHtml()将实体转义的 HTML 转换为 Spanned 对象。

```
someTextView.setText(Html.fromHtml(resultFromStringFormat));
```

为了更加直观，请查看 Resources/Strings 演示内容。以下是布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    <Button android:id="@+id/format"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/btn_name"
      />
    <EditText android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      />
  </LinearLayout>
  <TextView android:id="@+id/result"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

从中可以看到，它只是一个按钮、一个字段和一个标签。此处的理念在于，用户在字段中输

入名称，然后单击按钮让标签使用其中包含其名称的格式化消息进行更新。

布局文件中的 Button 引用字符串资源（@string/btn_name），因此我们需要一个字符串资源文件（res/values/strings.xml）：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">StringsDemo</string>
  <string name="btn_name">Name:</string>
  <string name="funky_format">My name is &lt;b&gt;%1$s&lt;/b&gt;</string>
</resources>
```

通过 activityCreator 脚本自动创建 app_name 资源。btn_name 字符串是 Button 的标题，我们的样式字符串格式位于 funky_format 中。

最后，要将所有内容连接到一起，我们需要一部分 Java 代码：

```
package com.commonsware.android.strings;

import android.app.Activity;
import android.os.Bundle;
import android.text.TextUtils;
import android.text.Html;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class StringsDemo extends Activity {
  EditText name;
  TextView result;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    name=(EditText)findViewById(R.id.name);
    result=(TextView)findViewById(R.id.result);

    Button btn=(Button)findViewById(R.id.format);

    btn.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        applyFormat();
      }
    });
  }

  private void applyFormat() {
    String format=getString(R.string.funky_format);
    String simpleResult=String.format(format,
                        TextUtils.htmlEncode(name.getText().toString()));
    result.setText(Html.fromHtml(simpleResult));
  }
}
```

单击按钮时调用的 applyFormat()可以操作字符串资源。首先，我们通过 getString()获取格式（为了提高效率，我们在调用 onCreate()时就可以完成该操作）。其次，我们使用此格式对字段中的值进行格式化，重新获取 String，因为字符串资源是实体编码的 HTML。注意使用 TextUtils.htmlEncode()将输入的名称进行实体编码，以防有人决定使用&符号之类的内容。最后，我们通过 Html.fromHtml()将简单的 HTML 转换为样式文本对象并更新标签。

第一次启动 Activity 时，我们得到一个空标签，如图 20-1 所示。

填写名称并单击按钮之后，将得到图 20-2 所示的结果。

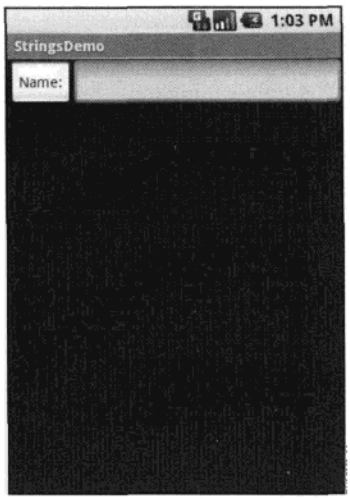


图 20-1　第一次启动时的 StringsDemo 示例应用程序

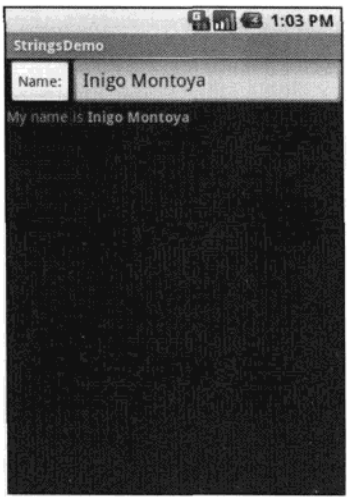


图 20-2　填写某位英雄人物名称后的同一个应用程序

20.3　获取图片

Android 支持 PNG、JPEG 和 GIF 格式的图片。但是，官方建议不要使用 GIF 格式。PNG 是首选格式。在需要 Drawable 的地方都可以使用图像，例如 ImageView 的图像和背景。

使用图像其实就是将图像文件放入 res/drawable/，然后作为资源引用。在布局文件中，图像作为@drawable/...引用，其中省略号是文件的基本名称（例如，对于 res/drawable/foo.png，资源名称为@drawable/foo）。在 Java 中，在需要图像资源 ID 的地方应使用 R.drawable.加上基本名称（例如 R.drawable.foo）的形式。

为了演示这一点，让我们更新上一个示例，为按钮使用一个图标而不是字符串资源。图标可

用 Resources/Images 获取。首先，我们稍微调整一下布局文件，使用 ImageButton 并引用名为 @drawable/icon 的 Drawable。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    <ImageButton android:id="@+id/format"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/icon"
      />
    <EditText android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      />
  </LinearLayout>
  <TextView android:id="@+id/result"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

接下来，我们需要使用图标的基本名称将图像文件放入 res/drawable。在这种情况下，我们使用取自 Nuvola 图标集（http://www.iconking.com/projects/nuvola/）的 32×32 的 PNG 文件。最后，我们修改 Java 源代码，使用 ImageButton 替换 Button：

```
package com.commonsware.android.images;

import android.app.Activity;
import android.os.Bundle;
import android.text.TextUtils;
import android.text.Html;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.EditText;
import android.widget.TextView;

public class ImagesDemo extends Activity {
  EditText name;
  TextView result;

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

    name=(EditText)findViewById(R.id.name);
    result=(TextView)findViewById(R.id.result);
```

```
    ImageButton btn=(ImageButton)findViewById(R.id.format);

    btn.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        applyFormat();
      }
    });
  }

  private void applyFormat() {
    String format=getString(R.string.funky_format);
    String simpleResult=String.format(format,
                        TextUtils.htmlEncode(name.getText().toString()));
    result.setText(Html.fromHtml(simpleResult));
  }
}
```

现在，我们的按钮拥有了所需的图标，如图 20-3 所示。

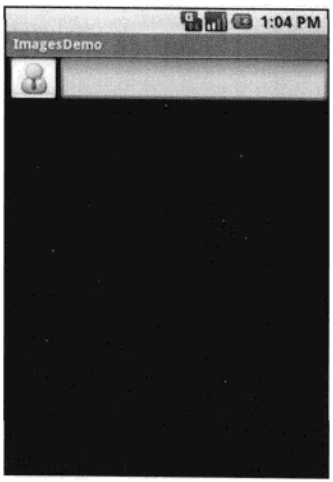


图 20-3　ImagesDemo 示例应用程序

20.4　XML：资源之路

如果希望在应用程序中打包静态 XML，可以使用 XML 资源。只需要将 XML 文件放入 res/xml/，然后就可以通过对 Resources 对象调用 getXml()访问该文件，但是要提供资源 ID——R.xml.加上 XML 文件的基本名称。例如，某个 Activity 有一个名为 words.xml 的 XML 文件，那么可以调用 getResources().getXml（R.xml.words）。

这将返回 Java 名称空间 org.xmlpull.v1 中的 XmlPull Parser 实例。XML pull 解析器是事件推动的：对解析器不断地调用 next()获取下一个事件，可能是 START_TAG、END_TAG、END_DOCUMENT，

等等。在 START_TAG 事件中，可以访问标记的名称和特性；一个 TEXT 事件表示一连串此元素所有直接子代的文本节点。通过循环、测试和调用每个元素的逻辑，就可以解析该文件。

为了更加直观，我们重新编写 Files/Static 示例项目的 Java 代码，以使用 XML 资源。新项目 Resources/XML 需要你使用 Static 的 words.xml 文件，但不是在 res/raw/中，而是在 res/xml/中。布局仍然是相同的，因此所需替换的只有 Java 源代码：

```
package com.commonsware.android.resources;

import android.app.Activity;
import android.os.Bundle;
import android.app.ListActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import java.io.InputStream;
import java.util.ArrayList;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;

public class XMLResourceDemo extends ListActivity {
  TextView selection;
  ArrayList<String> items=new ArrayList<String>();

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

    try {
      XmlPullParser xpp=getResources().getXml(R.xml.words);

      while (xpp.getEventType()!=XmlPullParser.END_DOCUMENT) {
        if (xpp.getEventType()==XmlPullParser.START_TAG) {
          if (xpp.getName().equals("word")) {
            items.add(xpp.getAttributeValue(0));
          }
        }

        xpp.next();
      }
    }
    catch (Throwable t) {
      Toast
        .makeText(this, "Request failed: "+t.toString(), 4000)
        .show();
    }

    setListAdapter(new ArrayAdapter<String>(this,
                           android.R.layout.simple_list_item_1,
                           items));
  }
```

```
    public void onListItemClick(ListView parent, View v, int position,
                long id) {
      selection.setText(items.get(position).toString());
    }
}
```

现在，在 try...catch 代码块中，我们获取 XmlPullParser，并执行一个循环到文档结束。如果当前事件是 START_TAG，元素名称是 word（xpp.getName().equals("word")），那么我们仅获取这个唯一的特性，并将其放入选定部件的条目列表中。由于我们可以完全控制 XML 文件，因此假定只有一个特性是足够安全的。如果不确定 XML 的定义是否恰当，可以考虑查看特性计数（getAttributeCount()）以及特性名称（getAttributeName()），确定是否可以假定索引为 0 的特性就是你所要的。

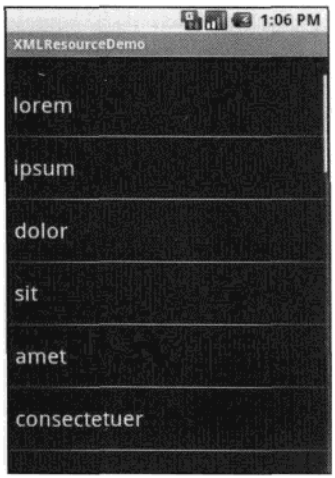


图 20-4　XMLResourceDemo 示例应用程序

得到的结果与之前相同，但标题栏的名称不同，如图 20-4 所示。

20.5　杂项

在 res/values/目录中，除了字符串资源之外，还可以放入一个（或多个）描述其他简单资源的 XML 文件，例如维度、颜色和数组。在之前的示例中，我们已经学习了如何使用维度和颜色，它们都以简单的字符串（例如，"10px"）形式作为参数传递给调用者。我们可以将它们设置为 Java static final 对象，并使用它们的符号名称，但是这只在 Java 源代码中有效，在布局 XML 文件中无效。若将这些值放入资源 XML 文件中，可以同时通过 Java 和布局引用这些值，而且集中放置也便于编辑。

资源 XML 文件有一个 resources 根元素，其他内容都是该根元素的子元素。

20.5.1　维度

在 Android 中，有些地方使用维度来描述距离，例如部件的填充。本书的大部分示例都使用像素（例如，10px 表示 10 像素），但也有一些不同的度量单位。

- in 和 mm 分别表示英寸和毫米。它们都基于屏幕实际大小。
- pt 表示磅。在出版术语中，一磅等于 1/72 英寸。（同样，它也基于屏幕实际大小。）
- dip 和 sp 分别表示与手机无关的像素和与比例无关的像素。对于分辨率为 160dpi 的屏幕，1 像素等于 1dip，比例则基于实际的屏幕像素密度。与比例无关的像素还要考虑用户的首选字体大小。

要将维度编码为资源，只需添加 dimen 元素、一个用于资源唯一名称的 name 特性，以及一个表示该值的子文本元素：

```
<resources>
  <dimen name="thin">10px</dimen>
  <dimen name="fat">1in</dimen>
</resources>
```

在布局中，可以用@dimen/...的形式引用维度，其中省略号是资源唯一名称的占位符（例如，上一个示例中的 thin 和 fat）。在 Java 中，可以通过带有前缀 R.dimen.的唯一名称引用维度资源。（例如，Resources.getDimen(R.dimen.thin)。）

20.5.2 颜色

Android 中的颜色是十六进制的 RGB 值，也可以指定 alpha 通道。我们可以选择使用一个字符的十六进制值还是两个字符的十六进制值，提供的样式有 4 种：

- #RGB
- #ARGB
- #RRGGBB
- #AARRGGBB

其工作原理类似于 CSS（Cascading Style Sheets，级联样式表）中的对应内容。

当然，你可以将这些 RGB 值作为字符串字面值放入 Java 源代码或布局资源。如果希望将其转变为资源，只需向资源文件中添加 color 元素、一个 name 特性（用作此颜色的唯一名称），以及一个包含 RGB 值的文本元素：

```
<resources>
  <color name="yellow_orange">#FFD555</color>
  <color name="forest_green">#005500</color>
  <color name="burnt_umber">#8A3324</color>
</resources>
```

在布局中，也可以用@color/...形式引用颜色，使用颜色的唯一名称（例如，burnt_umber）替换其中的省略号。在 Java 中，可以通过带有前缀 R.color.的唯一名称引用颜色资源（例如，Resources.getColor(R.color.forest_green)）。

20.5.3 数组

数组资源用于保存简单字符串列表，例如敬语列表（Mr.、Mrs.、Ms.、Dr.等）。

在资源文件中，每个数组都需要一个 string-array 元素，以及一个 name 特性，作为你为数组提供的唯一名称。然后，添加一个或多个子 item 元素，每个元素都有一个包含该项值的文本

元素：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="cities">
    <item>Philadelphia</item>
    <item>Pittsburgh</item>
    <item>Allentown/Bethlehem</item>
    <item>Erie</item>
    <item>Reading</item>
    <item>Scranton</item>
    <item>Lancaster</item>
    <item>Altoona</item>
    <item>Harrisburg</item>
  </string-array>
  <string-array name="airport_codes">
    <item>PHL</item>
    <item>PIT</item>
    <item>ABE</item>
    <item>ERI</item>
    <item>RDG</item>
    <item>AVP</item>
    <item>LNS</item>
    <item>AOO</item>
    <item>MDT</item>
  </string-array>
</resources>
```

在 Java 代码中，可以使用 Resources.getStringArray()获取列表中各项的 String[]。getStringArray()的参数是该数组的唯一名称，该名称带有前缀 R.array.（例如，Resources.getStringArray(R.array.honorifics)）。

20.5.4　因人而异

一个资源集也许无法满足可能使用应用程序的所有情况。一个明显的问题就是使用字符串资源以及处理国际化（I18N）和本地化（L10N）问题。对所有字符串都使用一种语言看起来不错，至少对开发人员来说是如此，但这没有涉及多种语言的情况。

资源需要有所变化的不止这一种情况。还包括以下情况。

- 屏幕方向：屏幕是纵向还是横向？还是说屏幕是正方形，因此不存在方向一说？
- 屏幕大小：屏幕有多少像素，以便相应地调整资源大小？（例如，大图标与小图标。）
- 触摸屏：手机有触摸屏吗？如果有，已经设为使用笔或手指操作触摸屏了吗？
- 键盘：用户使用哪种键盘（QWERTY 键盘、数字键盘还是其他），现在已经使用还是准备使用？
- 其他输入设备：手机有其他输入形式吗？例如 D-pad 或点击轮（click-wheel）？

Android 目前处理这些内容的方法是使用多个资源目录，并对嵌入名称的内容制定了标准。

假设要同时支持英语和西班牙语的字符串。一般来说，在一种语言的情况下，应该将字符串

放入名为 res/values/strings.xml 的文件中。为了同时支持英语和西班牙语，应该创建两个文件夹，名为 res/values-en/和 res/values-es/，其中连字符后的值是该语言的 ISO 639-1 双字母代码。英语字符串应该放入 res/values-en/strings.xml，西班牙语字符串应该放入 res/values-es/strings.xml。Android 将根据用户的手机设置选择恰当的文件。

还有一种更好的方法，就是将某种语言设为默认语言，将这些字符串放入 res/values/strings.xml。然后，创建其他资源目录用于翻译（例如，res/values-es/strings.xml 用于西班牙语）。Android 将尝试匹配特定语言的资源集；如果失败，它将使用默认的 res/values/strings.xml。

很简单，对不对？

如果需要对资源使用多个不同的标准，那么事情就开始复杂了。假如你想针对 T-Mobile G1 和两个当前的假想手机做开发。一个手机（Fictional One）是 VGA（“大”）屏幕，通常处于横向，总是打开 QWERTY 键盘，有 D-pad，但没有触摸屏。另一个手机（Fictional Two）有一个 G1 大小的屏幕（普通），有数字键盘但没有 QWERTY 键盘，有 D-pad，但没有触摸屏。

你可能想为两种手机使用不同的布局，以充分利用不同的屏幕资产和不同的输入选项，如：

- 每个分辨率与方向的组合；
- 触摸屏设备与没有触摸屏的手机；
- QWERTY 键盘与非 QWERTY 键盘设备；

如果涉及这些情况，将适用如下规则。

- 配置选项（例如，-en）有特定的优先级顺序，它们必须以该顺序出现在目录名称中。Android 文档概述了这些选项的具体出现顺序。对于此例，屏幕方向必须优先于触摸屏类型，后者又必须优先于屏幕大小。
- 每个目录的每个配置选项种类只能有一个值。
- 选项区分大小写。

因此，对于示例场景，理论上我们需要以下目录：

- res/layout-large-port-notouch-qwerty
- res/layout-normal-port-notouch-qwerty
- res/layout-large-port-notouch-12key
- res/layout-normal-port-notouch-12key
- res/layout-large-port-notouch-nokeys
- res/layout-normal-port-notouch-nokeys
- res/layout-large-port-stylus-qwerty

- res/layout-normal-port-stylus-qwerty
- res/layout-large-port-stylus-12key
- res/layout-normal-port-stylus-12key
- res/layout-large-port-stylus-nokeys
- res/layout-normal-port-stylus-nokeys
- res/layout-large-port-finger-qwerty
- res/layout-normal-port-finger-qwerty
- res/layout-large-port-finger-12key
- res/layout-normal-port-finger-12key
- res/layout-large-port-finger-nokeys
- res/layout-normal-port-finger-nokeys
- res/layout-large-land-notouch-qwerty
- res/layout-normal-land-notouch-qwerty
- res/layout-large-land-notouch-12key
- res/layout-normal-land-notouch-12key
- res/layout-large-land-notouch-nokeys
- res/layout-normal-land-notouch-nokeys
- res/layout-large-land-stylus-qwerty
- res/layout-normal-land-stylus-qwerty
- res/layout-large-land-stylus-12key
- res/layout-normal-land-stylus-12key
- res/layout-large-land-stylus-nokeys
- res/layout-normal-land-stylus-nokeys
- res/layout-large-land-finger-qwerty
- res/layout-normal-land-finger-qwerty
- res/layout-large-land-finger-12key
- res/layout-normal-land-finger-12key
- res/layout-large-land-finger-nokeys
- res/layout-normal-land-finger-nokeys

不用担心！一会儿我们就能缩短这个列表。

注意，对于大部分这些目录，实际的布局文件是相同的。例如，我们只关心触摸屏布局是否与其他两种布局不同，但是由于我们无法组合这两种布局，理论上需要对 `finger` 和 `stylus` 的相同内容使用不同的目录。

还要注意，当然可以使用另一个目录并舍弃基本名称（res/layout）。实际上，这也许是个好主意，以防将来的 Android 运行时版本引入你没有考虑到的其他配置选项。是否拥有默认布局对于应用程序在新手机上是否能够正常运行至关重要。

现在，我们可以使用一点技巧解码规则，Android 使用这些规则确定候选集合中哪个是要使用的正确资源目录。

- 首先，Android 将丢弃无效的目录。因此，如果手机的屏幕大小为普通的，-large 目录将不再是候选目录，因为它们要求其他屏幕大小。
- 接下来，Android 计算每个目录匹配的数量，并只关注匹配数最多的目录。
- 最后，Android 按照选项优先级顺序进行查看；换句话说，它按照目录名称从左到右执行检查。

因此，我们将只需要以下配置：

- res/layout-large-port-notouch-qwerty
- res/layout-port-notouch-qwerty
- res/layout-large-port-notouch
- res/layout-port-notouch
- res/layout-large-port-qwerty
- res/layout-port-qwerty
- res/layout-large-port
- res/layout-port
- res/layout-large-land-notouch-qwerty
- res/layout-land-notouch-qwerty
- res/layout-large-land-notouch
- res/layout-land-notouch
- res/layout-large-land-qwerty
- res/layout-land-qwerty
- res/layout-large-land
- res/layout-land

此处我们利用了具体匹配优先于未指定值这一事实。因此，带有 QWERTY 键盘的手机将选择目录中有 qwerty 的资源，而不是没有指定其键盘类型的资源。结合“最多匹配最佳”的规则，我们可知 res/layout-port 只匹配普通屏幕大小、无 QWERTY 键盘、有触摸屏且为纵向的手机。

我们还可以进一步细化，只涵盖特定的目标手机（T-Mobile G1、Fictional One 和 Fictional Two）

并充分利用 res/layout 作为总体默认内容的特点：

- res/layout-large-port-notouch
- res/layout-port-notouch
- res/layout-large-land-notouch
- res/layout-land-notouch
- res/layout-large-land
- res/layout

其中-large 可以将 Fictional One 与其他两种手机区分开，而 notouch 可以将 Fictional Two 与 T-Mobile G1 区分开。

你将在第 36 章看到这些资源集，这一章介绍了如何支持多种屏幕大小。

第21章 使用首选项 21

Android有很多方式可以存储活动长期使用的数据。最简单的方式是使用首选项系统，这也是本章的主题。

Android允许活动和应用程序以键/值对的形式（与Map类似）保存首选项，首选项与活动的调用密切相关。顾名思义，首选项主要用于存储用户指定的配置详情，比如用户在feed阅读器中上次查看的feed，默认对列表使用的排序，等等。当然，可以在首选项中存储任何内容，只要它有一个String键并且有一个基本类型的值（Boolean、String等）。

首选项可以仅用于一个活动，也可以在应用程序的所有活动之间共享。（最终，首选项应该能在应用程序之间共享，但是在本文编写之际还没有实现。）

21.1 获取想要的内容

要访问首选项，可以使用以下API：

- Activity中的getPreferences()，可访问特定于Activity的首选项；
- Activity中（或者其他应用程序Context）的getSharedPreferences()，可访问应用程序级别的首选项；
- PreferencesManager上的getDefaultSharedPreferences()，可访问与Android的整个首选项框架一致的共享首选项。

前两个API接受安全模式参数，目前可以传入0。getSharedPreferences()方法还可以接受一个首选项集合名称。使用活动的类名作为首选项集合名称，getPreferences()可以有效地调用getSharedPreferences()。getDefaultSharedPreferences()方法接受Context来获取首选项（例如，Activity）。

所有这些方法都返回SharedPreferences的一个实例，该实例提供一系列getter方法来访问

指定的首选项，返回适当类型的结果（例如，getBoolean()返回一个布尔值首选项）。getter 方法还可以接受默认值，在指定键下没有设定首选项时返回该默认值。

21.2　编辑首选项

给定合适的 SharedPreferences 对象，就可以使用 edit()获取首选项的编辑器。此对象有一个 setter 方法集合，对应于父 SharedPreferences 对象的 getter 方法。它还有以下方法。

- remove()：删除一个指定的首选项。
- clear()：删除所有首选项。
- commit()：让通过编辑器进行的更改持久保留。

commit()方法很重要。如果通过编辑器修改了首选项但无法 commit()更改，那么这些更改将在退出编辑器之后消失。

相反，由于首选项对象支持实时更改，如果应用程序的某个部分（比如活动）修改共享的首选项，其他部分（比如服务）将立即可访问更改后的值。

21.3　目前的框架

从 0.9 SDK 开始，Android 引入了一个框架，用于管理首选项。具有讽刺意味的是，该框架不能更改目前为止所显示的任何内容。相反，该框架只是为用户展示一个统一的首选项-设置选项集合，因此不同的应用程序不需要重新创建相关内容。

首选项框架的关键在于另一个 XML 数据结构。你可以在项目的 res/xml/目录中存储的 XML 文件内描述应用程序的首选项。这样一来，Android 可以展示一个不错的 UI 用于操作这些首选项，然后将这些首选项存储在 SharedPreferences 中，你可以通过 getDefaultSharedPreferences()获取这些内容。

以下是用于 Prefs/Simple 首选项示例项目的首选项 XML：

```
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <CheckBoxPreference
    android:key="checkbox"
    android:title="Checkbox Preference"
    android:summary="Check it on, check it off" />
  <RingtonePreference
    android:key="ringtone"
    android:title="Ringtone Preference"
    android:showDefault="true"
    android:showSilent="true"
    android:summary="Pick a tone, any tone" />
</PreferenceScreen>
```

首选项 XML 的根是 PreferenceScreen 元素。本章后文将介绍为什么要对它如此命名。现在，只需要把它看作一个合理的名称即可。

无疑，PreferenceScreen 元素中存放的一些内容是首选项定义。它们是 Preference 的子类，如 CheckBoxPreference 或 RingtonePreference，如之前的 XML 所示。你可能会想到，它们可以分别用于勾选复选框或选择手机铃声。在引入 RingtonePreference 的情况下，有一个选项允许用户选择系统默认手机铃声，或者选择“无声”。

21.4 让用户自己选择

假如设置了首选项 XML，那么可以使用几乎为内置的活动让用户设置其首选项。活动是“几乎内置的”，因为只需要将其子类化，并将其指向首选项 XML，将活动挂钩到应用程序的其余部分。

例如，以下是 Prefs/Simple 项目的 EditPreferences 活动。

```
package com.commonsware.android.simple;

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


public class EditPreferences extends PreferenceActivity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    addPreferencesFromResource(R.xml.preferences);
  }
}
```

此处没有什么可以看的。需要做的就是调用 addPreferencesFromResource()，并指定包含首选项的 XML 资源。

此处，需要将其作为活动添加到 AndroidManifest.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.simple">
    <application android:label="@string/app_name"
        android:icon="@drawable/cw">
        <activity
            android:name=".SimplePrefsDemo"
            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=".EditPreferences"
            android:label="@string/app_name">
        </activity>
    </application>
</manifest>
```

你需要重新安排以调用活动，比如通过菜单选项调用，下面从 SimplePrefsDemo 调用：

```
public boolean onCreateOptionsMenu(Menu menu) {
    menu.add(Menu.NONE, EDIT_ID, Menu.NONE, "Edit Prefs")
        .setIcon(R.drawable.misc)
        .setAlphabeticShortcut('e');

    return(super.onCreateOptionsMenu(menu));
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case EDIT_ID:
        startActivity(new Intent(this, EditPreferences.class));
        return(true);
    }

    return(super.onOptionsItemSelected(item));
  }
}
```

需要做的全在这里，首选项 XML 之外不需要太多代码。你努力的结果是 Android 提供的首选项 UI，如图 21-1 所示。

复选框可以直接勾选或取消勾选。要更改手机铃声首选项，只需要选择首选项列表中的这一项，以打开选择对话框，如图 21-2 所示。

注意，没有显式保存或者提交按钮或菜单。任何更改一旦完成，就是持久的。

SimplePrefsDemo 活动不仅可提供上文提到的菜单，而且可通过 TableLayout 显示当前的首选项：

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
>
  <TableRow>
    <TextView
        android:text="Checkbox:"
        android:paddingRight="5px"
    />
    <TextView android:id="@+id/checkbox"
    />
  </TableRow>
  <TableRow>
    <TextView
        android:text="Ringtone:"
```

```
        android:paddingRight="5px"
    />
    <TextView android:id="@+id/ringtone"
    />
  </TableRow>
</TableLayout>
```

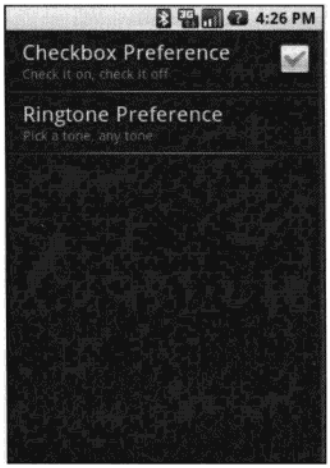


图 21-1 简单项目的首选项 UI

图 21-2 选择手机铃声首选项

该表格的字段通过 onCreate()获得：

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

  checkbox=(TextView)findViewById(R.id.checkbox);
  ringtone=(TextView)findViewById(R.id.ringtone);
}
```

在每个 resume()上更新这些字段：

```
public void onResume() {
  super.onResume();

  SharedPreferences prefs=PreferenceManager
                          .getDefaultSharedPreferences(this);

  checkbox.setText(new Boolean(prefs
                               .getBoolean("checkbox", false))
                   .toString());
  ringtone.setText(prefs.getString("ringtone", "<unset>"));
}
```

这意味着，在打开活动以及离开首选项活动（例如通过后退按钮）之后将更新字段，如图 21-3 所示。

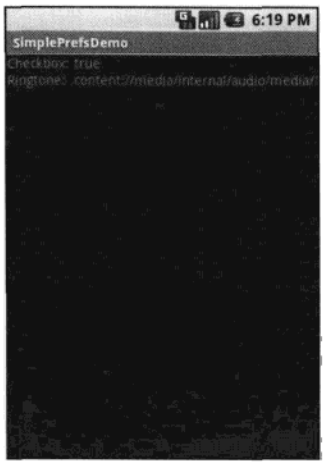


图 21-3　简单项目的已保存首选项列表

21.5　添加“分层”结构

如果要用户设置的首选项很多，将它们放在一个大列表中可能不是什么好主意。Android 的首选项框架提供了几种方法，可以实现首选项分层结构，包括种类和屏幕（screen）。

种类通过 PreferenceCategory 元素添加到首选项 XML 中，用于分组相关的首选项。不用再将所有首选项都作为根 PreferenceScreen 的子元素，可以在 PreferenceScreen 中放置一些 PreferenceCategory 元素，然后将首选项放入合适的种类中。从视觉上看，这将在首选项组之间添加带有种类标题的分隔栏。

如果拥有许多首选项，以至于用户无法方便地滚动查找，那么还可以引入 Preference Screen 元素，将其放在分隔的“屏幕”中。没错，就是那个 PreferenceScreen 元素。

PreferenceScreen 的任何子元素都出现在自己的屏幕中。如果嵌入 PreferenceScreen，父屏幕将把嵌入的屏幕作为占位符项显示，点击该项将打开子屏幕。

例如，在 Prefs/Structured 示例项目中，有一个包含 PreferenceCategory 和嵌套 PreferenceScreen 元素的首选项 XML 文件：

```
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory android:title="Simple Preferences">
    <CheckBoxPreference
      android:key="checkbox"
      android:title="Checkbox Preference"
      android:summary="Check it on, check it off"
```

```
    />
    <RingtonePreference
      android:key="ringtone"
      android:title="Ringtone Preference"
      android:showDefault="true"
      android:showSilent="true"
      android:summary="Pick a tone, any tone"
    />
  </PreferenceCategory>
  <PreferenceCategory android:title="Detail Screens">
    <PreferenceScreen
      android:key="detail"
      android:title="Detail Screen"
      android:summary="Additional preferences held in another page">
      <CheckBoxPreference
        android:key="checkbox2"
        android:title="Another Checkbox"
        android:summary="On. Off. It really doesn't matter."
      />
    </PreferenceScreen>
  </PreferenceCategory>
</PreferenceScreen>
```

使用此首选项 XML 与 PreferenceActivity 实现时，将得到一个分类的元素列表，如图 21-4 所示。

如果单击 Detail Screen 项，将出现子首选项屏幕，如图 21-5 所示。

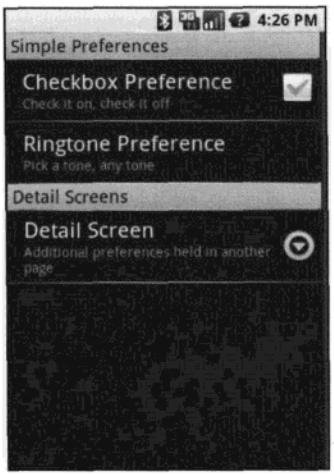


图 21-4 结构化项目的首选项 UI，展示了种类和屏幕占位符

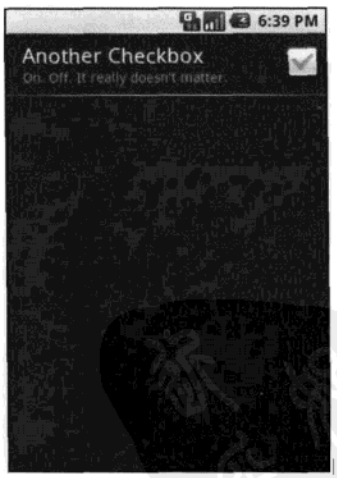


图 21-5 结构化项目中首选项 UI 的子首选项屏幕

21.6 弹出对话框

当然，并非所有首选项都是复选框和手机铃声。对于其他类别，如输入字段和列表，Android

使用弹出对话框。用户不是在首选项 UI 活动中直接输入首选项，而是单击首选项，填写值，然后单击 OK 提交更改。

从结构上看，在首选项 XML 中，字段和列表与其他首选项类型没有太多区别，如 Prefs/Dialogs 示例项目的首选项 XML 所示：

```
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
  <PreferenceCategory android:title="Simple Preferences">
    <CheckBoxPreference
      android:key="checkbox"
      android:title="Checkbox Preference"
      android:summary="Check it on, check it off"
    />
    <RingtonePreference
      android:key="ringtone"
      android:title="Ringtone Preference"
      android:showDefault="true"
      android:showSilent="true"
      android:summary="Pick a tone, any tone"
    />
  </PreferenceCategory>
  <PreferenceCategory android:title="Detail Screens">
    <PreferenceScreen
      android:key="detail"
      android:title="Detail Screen"
      android:summary="Additional preferences held in another page">
      <CheckBoxPreference
        android:key="checkbox2"
        android:title="Another Checkbox"
        android:summary="On. Off. It really doesn't matter."
      />
    </PreferenceScreen>
  </PreferenceCategory>
  <PreferenceCategory android:title="Simple Preferences">
    <EditTextPreference
      android:key="text"
      android:title="Text Entry Dialog"
      android:summary="Click to pop up a field for entry"
      android:dialogTitle="Enter something useful"
    />
    <ListPreference
      android:key="list"
      android:title="Selection Dialog"
      android:summary="Click to pop up a list to choose from"
      android:entries="@array/cities"
      android:entryValues="@array/airport_codes"
      android:dialogTitle="Choose a Pennsylvania city" />
  </PreferenceCategory>
</PreferenceScreen>
```

使用字段（EditTextPreference），再加上放入首选项本身的标题和摘要，还可以提供标题供对话框使用。

使用列表（ListPreference），可以提供对话框标题和两个字符串-数组资源：一个用于显示名称，另一个用于显示值。它们的顺序必须相同，因为所选显示名称的索引将确定哪些值作为首

选项存储在 SharedPreferences 中。例如，以下是上例所示的 ListPreference 使用的数组：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="cities">
    <item>Philadelphia</item>
    <item>Pittsburgh</item>
    <item>Allentown/Bethlehem</item>
    <item>Erie</item>
    <item>Reading</item>
    <item>Scranton</item>
    <item>Lancaster</item>
    <item>Altoona</item>
    <item>Harrisburg</item>
  </string-array>
  <string-array name="airport_codes">
    <item>PHL</item>
    <item>PIT</item>
    <item>ABE</item>
    <item>ERI</item>
    <item>RDG</item>
    <item>AVP</item>
    <item>LNS</item>
    <item>AOO</item>
    <item>MDT</item>
  </string-array>
</resources>
```

打开首选项 UI 时，可以看到另一个种类以及另一对首选项条目，如图 21-6 所示。

单击 Text Entry Dialog 条目打开文本输入对话框，以前的首选项条目已经填入，如图 21-7 所示。

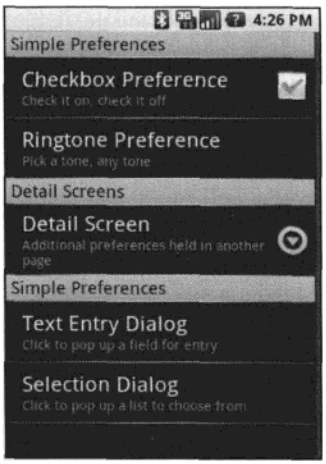


图 21-6 对话框项目首选项 UI 的首选项屏幕

图 21-7 编辑文本首选项

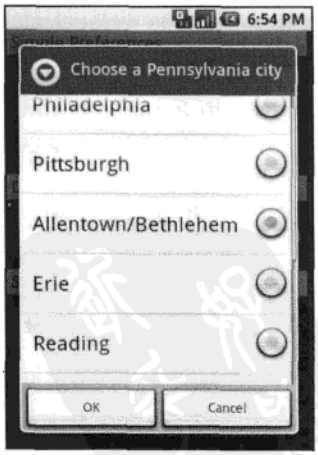


图 21-8 编辑列表首选项

单击 Selection Dialog 可以打开选择对话框，显示一个数组中的名称，如图 21-8 所示。

第 22 章 管理和访问本地数据库

SQLite 是一个非常流行的嵌入式数据库，它的 SQL 界面非常简洁，内存占用少，速度快。此外，它是公开的，所以每个人都可以使用。许多公司（如 Adobe、苹果公司、Google、Sun 和 Symbian）和开源项目（如 Mozilla、PHP 和 Python）都在产品中使用了 SQLite。

对于 Android，SQLite 已经“融入”到 Android 运行时，因此所有 Android 应用程序都可以创建 SQLite 数据库。由于 SQLite 使用 SQL 界面，对于有基于其他 SQL 数据库使用经验的用户而言，它的使用非常简单直接。但是，它的本机 API 不是 JDBC，而 JDBC 对于手机等内存有限的设备来说开销太大。因此，Android 程序员需要学习一个不同的 API。幸运的是它很容易使用。

本章将介绍在 Android 工作环境中 SQLite 的基本用法。这绝不是一个完整的 SQLite 介绍。如果想了解更多有关 SQLite 以及如何在非 Android 环境中使用的方法，可以参考 Michael Owens 撰写的图书 *The Definitive Guide to SQLite*（Apress，2006）。

22.1 数据库示例

本章中的大部分示例代码都来自 Database/Constants 应用程序。此应用程序展示了一列物理常量，包括从 Android 的 `SensorManager` 选取的名称和值，如图 22-1 所示。

可以弹出一个菜单来添加新的常量，这将产生一个对话框，用于填写常量的名称和值，如图 22-2 所示。

接下来常量便被加入到了列表中。长时间单击（按下不放）现有常量将产生一个带有 Delete 选项的上下文菜单，经过确认之后将删除该常量。

当然，所有这些都存储在 SQLite 数据库中。

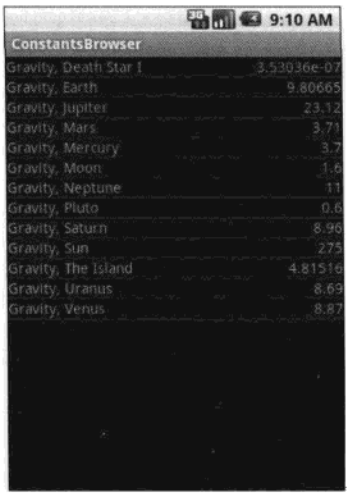


图 22-1 第一次启动的 Constants 示例应用程序

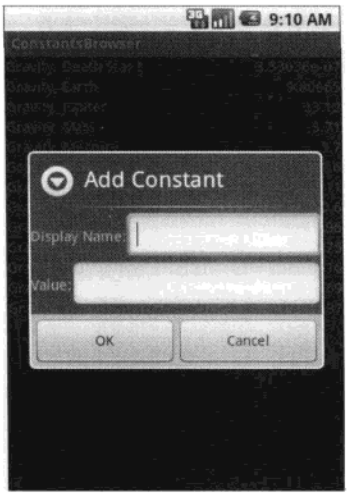


图 22-2 Constants 示例应用程序的 Add Constant 对话框

22.2 SQLite 快速入门

顾名思义，SQLite 使用 SQL 方言进行查询（SELECT）、数据操作（INSERT 之类）和数据定义（CREATE TABLE 之类）。SQLite 有些地方不遵从 SQL-92 标准，这种情况在大部分 SQL 数据库中都很常见。好消息是 SQLite 非常节省空间，因此 Android 运行时可以包含所有 SQLite，而不是缩减大小的任意功能子集。

SQLite 与其他 SQL 数据库之间最大的不同在于数据类型。就现状来说，可以使用 CREATE TABLE 语句指定列的数据类型，SQLite 也可以使用它们作为提示。可以将任何数据放入任何列中。可以在 INTEGER 列中放字符串吗？当然，没有问题！反过来呢？也没问题！如下列文档中所述，SQLite 将此称为清单类型（manifest typing）。

> 在清单类型中，数据类型是值本身的属性，而不是存储该值的列的属性。因此，SQLite 支持用户在任何列中存储任何数据类型的任何值，无论为该列声明的类型如何。

此外，SQLite 不支持一些标准的 SQL 功能，特别是 FOREIGN KEY 限制、嵌套事务、RIGHT OUTER JOIN、FULL OUTER JOIN 和一些版本的 ALTER TABLE。

除此之外，可以得到一个完整的 SQL 系统，包括触发器、事务等。SELECT 之类的 SQL 语句也能像预期那样正常工作。

说明　如果习惯于使用主流的数据库，比如Oracle，可能会将SQLite视为一个无足轻重的数据库。请在头脑中建立这样的观念，Oracle和SQLite要解决的问题不同，不久之后你不可能再在电话上看到完整的Oracle副本。

22.3　从头开始

Android不会自动提供任何数据库。如果希望使用SQLite，就需要创建数据库，然后使用表、索引和数据填充该数据库。

要创建和打开一个数据库，最好的选择是创建 SQLiteOpenHelper 的子类。该类可以根据你的规范和应用程序的需要包含在创建和升级数据库时所需的逻辑。SQLiteOpenHelper 的子类需要以下3个方法。

- 构造函数，向上链接 SQLiteOpenHelper 构造函数。此函数的参数包括 Context（例如，Activity）、数据库名称、可选的指针工厂（通常只需传递 null）以及一个表示所用数据库模式版本的整数。
- onCreate()，传递一个需使用合适表和初始数据填充的 SQLiteDatabase 对象。
- onUpgrade()，传递 SQLiteDatabase 对象和新旧版本号，以便确定将数据库从旧模式转变为新模式的最佳方式。最简单也是最不友好的方法是丢弃旧表，创建新表。

例如，以下是 Database/Constants 中的一个 DatabaseHelper 类，在 onCreate()中，创建表并添加一些行，在 onUpgrade()中则丢弃现有表并执行 onCreate()：

```
package com.commonsware.android.constants;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;
import android.hardware.SensorManager;

public class DatabaseHelper extends SQLiteOpenHelper {
  private static final String DATABASE_NAME="db";
  public static final String TITLE="title";
  public static final String VALUE="value";

  public DatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, 1);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE constants (_id INTEGER PRIMARY KEY AUTOINCREMENT,
 title TEXT, value REAL);");
```

```
ContentValues cv=new ContentValues();

cv.put(TITLE, "Gravity, Death Star I");
cv.put(VALUE, SensorManager.GRAVITY_DEATH_STAR_I);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Earth");
cv.put(VALUE, SensorManager.GRAVITY_EARTH);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Jupiter");
cv.put(VALUE, SensorManager.GRAVITY_JUPITER);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Mars");
cv.put(VALUE, SensorManager.GRAVITY_MARS);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Mercury");
cv.put(VALUE, SensorManager.GRAVITY_MERCURY);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Moon");
cv.put(VALUE, SensorManager.GRAVITY_MOON);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Neptune");
cv.put(VALUE, SensorManager.GRAVITY_NEPTUNE);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Pluto");
cv.put(VALUE, SensorManager.GRAVITY_PLUTO);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Saturn");
cv.put(VALUE, SensorManager.GRAVITY_SATURN);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Sun");
cv.put(VALUE, SensorManager.GRAVITY_SUN);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, The Island");
cv.put(VALUE, SensorManager.GRAVITY_THE_ISLAND);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Uranus");
cv.put(VALUE, SensorManager.GRAVITY_URANUS);
db.insert("constants", TITLE, cv);

cv.put(TITLE, "Gravity, Venus");
cv.put(VALUE, SensorManager.GRAVITY_VENUS);
db.insert("constants", TITLE, cv);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        android.util.Log.w("Constants", "Upgrading database, which will destroy all old
    data");
        db.execSQL("DROP TABLE IF EXISTS constants");
        onCreate(db);
      }
    }
```

要使用 SQLiteOpenHelper 子类，可以创建一个实例并调用 getReadableDatabase()或 getWriteableDatabase()，具体取决于是否要更改其内容。例如，我们的 ConstantsBrowser 活动使用 onCreate()打开数据库：

```
db=(new DatabaseHelper(this)).getWritableDatabase();
```

这将返回一个 SQLiteDatabase 实例，然后可以用它查询数据库或修改其数据。

完成数据库的使用（例如，活动关闭）之后，只需对 SQLiteDatabase 调用 close()释放连接。

22.4　设置表

为了创建表和索引，需要对 SQLiteDatabase 调用 execSQL()，使用希望对数据库应用的 DDL（Data Definition Language，数据定义语言）语句。除了数据库错误，此方法不返回任何内容。

例如，可以调用 execSQL()创建 constants 表，如 DatabaseHelper onCreate()方法所示：

```
db.execSQL("CREATE TABLE constants (_id INTEGER PRIMARY KEY AUTOINCREMENT, title
TEXT,
 value REAL);");
```

这将创建一个名为 constants 的表，带有名为_id 的主键列，它是一个自动增加的整数（插入行时 SQLite 将为你分配值），表中还包括两个数据列：title（文本）和 value（一个浮点值，SQLite 术语中称为 real）。SQLite 将自动为主键列创建索引。你在此处可以通过一些 CREATE INDEX 语句添加其他索引。

更有可能的是，在第一次创建数据库时创建表和索引，或者在数据库需要升级以适应新版本的应用程序时创建。如果不更改表模式，可能永远不会丢弃表或索引，但如果进行了更改，请使用 execSQL()根据需要调用 DROP INDEX 和 DROP TABLE 语句。

22.5　数据

假如有一个数据库，包括一个或多个表，你可能想在其中放些数据。有两种主要的方法可以实现这一点。

- 使用 execSQL()，就像创建表一样。execSQL()方法适用于不返回结果的任何 SQL，因此它可以很好地处理 INSERT、UPDATE、DELETE 等。

- 使用 SQLiteDatabase 对象上的 insert()、update()和 delete()方法。这些都是某些方法的“构造器”，它们将 SQL 语句分解为多个块，然后将这些块作为参数使用。

例如，下面我们使用 insert()在 constants 表中插入一个新行：

```
private void processAdd(DialogWrapper wrapper) {
  ContentValues values=new ContentValues(2);

  values.put("title", wrapper.getTitle());
  values.put("value", wrapper.getValue());

  db.insert("constants", "title", values);
  constantsCursor.requery();
}
```

这些方法使用了 ContentValues 对象，该对象实现一个 Map-esque 接口，当然也可有其他方法适用 SQLite 类型。例如，除了能够用 get()根据键检索值之外，还可以使用 getAsInteger()、getAsString()等。

insert()方法使用表名称、列名称（作为“null column hack”），以及一个 ContentValues（包含想要放入此行的初始值）。null column hack 适用于 ContentValues 实例为空的情况。指定为 null column hack 的列将通过 insert()生成的 SQL INSERT 语句被显式分配值 NULL。

update()方法使用表名称、一个 ContentValues（表示列和要使用的替代值），一个可选的 WHERE 子句，以及一个可选的参数列表（填充 WHERE 子句），用于替换任何嵌入的问号（?）。update()只更新具有固定值的列，而不是根据其他信息计算值的列，因此需要使用 execSQL()实现某些目的。WHERE 子句和参数列表的工作原理与其他 SQL API 的位置 SQL 参数类似。

delete()方法与 update()的工作原理类似，也使用表的名称、可选的 WHERE 子句，以及对应的参数（填充 WHERE 子句）。例如在以下代码中，我们将从 constants 表中删除一行，指定了该行的_ID：

```
private void processDelete(long rowId) {
  String[] args={String.valueOf(rowId)};

  db.delete("constants", "_ID=?", args);
  constantsCursor.requery();
}
```

22.6 有因必有果

与 INSERT、UPDATE 和 DELETE 一样，使用 SELECT 从 SQLite 数据库中检索数据有两种主要的方法：

- 使用 rawQuery()直接调用 SELECT 语句；
- 使用 query()通过各个组成部分生成查询。

SQLiteQueryBuilder 类以及指针和指针工厂的问题让事情变得更加复杂。但为了简便起见，我们分部分细细讨论。

22.6.1　Raw 查询

最简单的解决方案是 rawQuery()，至少在 API 中是如此。使用 SQL SELECT 语句就可以调用它。SELECT 语句可以使用位置参数，这些位置参数组成的数组形成了 rawQuery()的第二个参数。因此，我们这样做：

```
constantsCursor=db.rawQuery("SELECT _ID, title, value "+"FROM constants ORDER BY title", null);
```

返回值是一个指针，包含对结果进行迭代的方法。（将在 22.6.4 节中讨论。）

如果查询融入到应用程序中，则使用起来非常简单直接。但是，如果部分查询是动态的，事情会变得有点复杂，这已经超出了位置参数可以解决的范围。例如，如果需要检索的列集合在编译时不可知，将列名称连接形成的逗号分隔的列表非常繁琐，这时候就需要 query()了。

22.6.2　常规查询

query()方法使用 SELECT 语句的多个片段，并从中生成查询。这些片段在 query()参数中出现的顺序如下：

- 要查询的表的名称；
- 要检索的列的列表；
- WHERE 子句，包含位置参数（可选）；
- 要替换位置参数的值组成的列表；
- GROUP BY 子句（如果有）；
- ORDER BY 子句（如果有）；
- HAVING 子句（如果有）。

在不需要这些参数时，它们可以为 null。（当然，表名称除外。）

```
String[] columns={"ID", "inventory"};
String[] parms={"snicklefritz"};
Cursor result=db.query("widgets", columns, "name=?",parms, null, null, null);
```

22.6.3　使用构造器进行构建

另一个选择是使用 SQLiteQueryBuilder，它提供更加丰富的查询构建选项，对于涉及多个子查询结果的查询尤其有用。

SQLiteQueryBuilder 接口与 ContentProvider 接口紧密结合以执行查询。因此，内容提供程

序 query()实现的常用模式是创建 SQLiteQueryBuilder，填入一些默认值，然后让其构建（可选地执行）完整的查询——结合默认值以及查询请求时提供给内容提供程序的内容。

例如，以下代码片段就摘自一个使用 SQLiteQueryBuilder 的内容提供程序：

```
@Override
public Cursor query(Uri url, String[] projection, String selection,
                    String[] selectionArgs, String sort) {
  SQLiteQueryBuilder qb=new SQLiteQueryBuilder();

  qb.setTables(getTableName());

  if (isCollectionUri(url)) {
    qb.setProjectionMap(getDefaultProjection());
  }
  else {
    qb.appendWhere(getIdColumnName()+"="+url.getPathSegments().get(1));
  }

  String orderBy;

  if (TextUtils.isEmpty(sort)) {
    orderBy=getDefaultSortOrder();
  } else {
    orderBy=sort;
  }

  Cursor c=qb.query(db, projection, selection, selectionArgs,
                    null, null, orderBy);
  c.setNotificationUri(getContext().getContentResolver(), url);
  return c;
}
```

内容提供程序将在第 26 章和第 27 章详细介绍，在此之前请信任它们。此处你只需理解以下情况。

- 构造一个 SQLiteQueryBuilder。
- 获知用于查询的表（setTables(getTableName())）。
- 获知要返回的默认列集合（setProjectionMap()），或者提供一个 WHERE 子句，使用从提供给 query()调用的 URI 中提取的标识符标识表中的特定行（appendWhere()）。
- 最后，执行查询，混合预设置的值和调用 query()（qb.query(db, projection, selection, selectionArgs, null, null, orderBy)）时获得的值。

我们不会直接让 SQLiteQueryBuilder 执行查询，而是调用 buildQuery()生成它，并返回需要的 SQL SELECT 语句以便自己执行查询。

22.6.4 使用 Cursor

不管如何执行查询，都会返回 Cursor。这是 Android/SQLite 数据库版本中的指针，许多数据库系统都使用这一概念。使用指针可以执行以下操作：

- 通过 getCount()查看结果集中有多少行；
- 通过 moveToFirst()、moveToNext()和 isAfterLast()迭代行；
- 通过 getColumnNames()查找列名称，通过 getColumnIndex()将这些列名转换为列编号，通过 getString()、getInt()等方法从给定列的当前行获取值；
- 通过 requery()重新执行创建了指针的查询；
- 通过 close()释放指针的资源。

例如，我们在下面的代码中迭代 widgets 表的各项：

```
Cursor result=
  db.rawQuery("SELECT ID, name, inventory FROM widgets");

result.moveToFirst();

while (!result.isAfterLast()) {
 int id=result.getInt(0);
 String name=result.getString(1);
 int inventory=result.getInt(2);

 // do something useful with these
 result.moveToNext();
}

result.close();
```

你还可以在 SimpleCursorAdapter 或其他实现中包装 Cursor，然后将得到的适配器提交给 ListView 或其他选择部件。例如，在检索了已排序的常量列表之后，我们将其放入 ListView 供 ConstantsBrowser 活动使用，这只需几行代码：

```
ListAdapter adapter=new SimpleCursorAdapter(this,
                    R.layout.row, constantsCursor,
                    new String[] {"title", "value"},
                    new int[] {R.id.title, R.id.value});

setListAdapter(adapter);
```

提示　有些情况可能需要使用自己的Cursor子类，而不是Android提供的实现。在这些情况下，可以使用 queryWithFactory() 和 rawQueryWithFactory()，它们将 SQLiteDatabase.CursorFactory实例作为参数。该工厂负责通过其newCursor()实现创建新指针。

22.7　无所不在的数据

如果习惯于使用其他数据库进行开发工作，除了数据库的 API 之外，你可能还习惯使用一些检测和操作数据库内容的工具。使用 Android 模拟器时相应地有两种选择。

首先，模拟器应该绑定在 sqlite3 控制台程序中，可以通过 adb shell 命令使用。一旦进入

模拟器 shell，只需要执行 sqlite3，为其提供数据库文件所在的路径即可。数据库文件可以在以下位置查找：

```
/data/data/your.app.package/databases/your-db-name
```

此处的.app.package 是应用程序的 Java 包（例如，com.commonsware.android），-db-name 是数据库的名称，曾提供给 createDatabase()。

sqlite3 程序可以运行，如果习惯于使用控制台接口处理表，会发现该程序很有用。如果更喜欢一些界面友好的工具，可以将 SQLite 数据库从手机复制到开发机器上，然后使用可感知 SQLite 的客户端程序处理。但要注意，此时处理的是数据库副本；如果想将更改提交回手机，则需要将数据库传输回手机。

要从手机复制数据库，可以使用 adb pull 命令（或者 IDE 上的等效命令，在 Dalvik Debug Monitor Service 上则可以使用 File Manager，将在第 35 章讨论），该命令将指向手机上数据库位置的路径和本地目标作为参数使用。要在手机上存储修改后的数据库，可使用 adb push，该命令将指向数据库的本地路径和手机目标作为参数使用。

最便于访问的 SQLite 客户端之一是 Firefox 的 SQLite Manager 扩展，如图 22-3 所示，它可以在多种平台上工作。

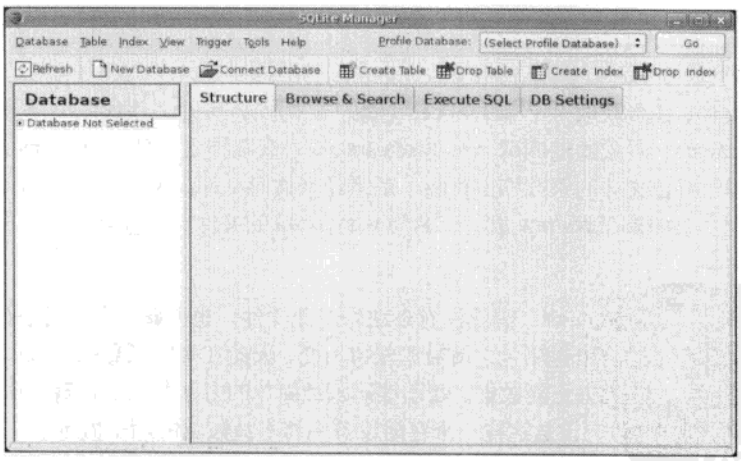


图 22-3 SQLite Manager Firefox 扩展

我们还可以在 SQLite 网站找到其他客户端工具。

第 23 章 访问文件 23

尽管 Android 通过首选项和数据库提供了结构化的存储方式，但有时简单的文件就足够了。Android 提供了两种模式来访问文件：一种用于访问与应用程序打包在一起的文件，一种用于应用程序在手机上创建的文件。这两种模式都将在本章介绍。

23.1 使用的数据

假设想随应用程序一起发布一些静态数据，比如用于拼写检查器的单词表。最简单的部署方式是将文件放入 res/raw 目录，这样它即可放入 Android 应用程序 APK 文件，以 raw 资源的形式进行打包。

要访问此文件，需要获取一个 Resources 对象。在 Activity 中，这就像调用 getResources() 一样简单。Resources 对象通过调用 openRawResource()获取指定文件的 InputStream。openRawResource()使用已打包文件的整数标识符而不是路径来访问文件。这就像通过 findViewById()访问部件一样。例如，如果在 res/raw 放一个名为 words.xml 的文件，在 Java 中可以用 R.raw.words 标识符访问文件。

由于只能获取 InputStream，因此没有办法修改此文件，也因此它只对静态引用数据有用。此外，在用户安装新版的应用程序包之前数据保持不变，因此在可以预见的将来引用数据必须有效，否则需要提供某种方式去更新数据。处理该问题最简单的方法是用引用数据引导某些可修改的存储形式（例如数据库），但是这样一来存储设备中将有该数据的两个副本。

还有一种替代方法，即保留引用数据不变，而在文件或数据库中保留修改版本，并在需要完整的信息时进行合并。例如，如果应用程序发布 URL 文件，可以用第二个文件跟踪用户添加的 URL，或者引用用户删除的 URL。

在 Files/Static 示例项目中，你会发现我们修改了第 7 章中列表框示例，这一次使用静态 XML 文件替代了 Java 中硬连接的数组。布局不变：

```
<?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/selection"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
  />
  <ListView
    android:id="@android:id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:drawSelectorOnTop="false"
  />
</LinearLayout>
```

除了该 XML 文件之外，还需要一个带有单词（要展示在列表中）的 XML 文件：

```
<words>
  <word value="lorem" />
  <word value="ipsum" />
  <word value="dolor" />
  <word value="sit" />
  <word value="amet" />
  <word value="consectetuer" />
  <word value="adipiscing" />
  <word value="elit" />
  <word value="morbi" />
  <word value="vel" />
  <word value="ligula" />
  <word value="vitae" />
  <word value="arcu" />
  <word value="aliquet" />
  <word value="mollis" />
  <word value="etiam" />
  <word value="vel" />
  <word value="erat" />
  <word value="placerat" />
  <word value="ante" />
  <word value="porttitor" />
  <word value="sodales" />
  <word value="pellentesque" />
  <word value="augue" />
  <word value="purus" />
</words>
```

虽然此 XML 结构实际上不是一个节省空间的模式，但它对于演示来说足够了。

Java 代码现在必须读入该 XML 文件，解析单词，并将其放入某个地方供列表获取：

```
public class StaticFileDemo extends ListActivity {
  TextView selection;
  ArrayList<String> items=new ArrayList<String>();

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

```
    selection=(TextView)findViewById(R.id.selection);

    try {
      InputStream in=getResources().openRawResource(R.raw.words);
      DocumentBuilder builder=DocumentBuilderFactory
                              .newInstance()
                              .newDocumentBuilder();
      Document doc=builder.parse(in, null);
      NodeList words=doc.getElementsByTagName("word");

      for (int i=0;i<words.getLength();i++) {
        items.add(((Element)words.item(i)).getAttribute("value"));
      }

      in.close();
    }
    catch (Throwable t) {
      Toast
        .makeText(this, "Exception: "+t.toString(), 2000)
        .show();
    }

    setListAdapter(new ArrayAdapter<String>(this,
                                android.R.layout.simple_list_item_1,
                                items));
  }

  public void onListItemClick(ListView parent, View v, int position,
                              long id) {
    selection.setText(items.get(position).toString());
  }
}
```

主要的不同之处在于 onCreate()。我们得到了 XML 文件的 InputStream（getResources().openRawResource(R.raw.words)），然后使用内置的 XML 解析逻辑将文件解析为 DOM Document，挑出单词元素，然后将值特性放入 ArrayList 供 ArrayAdapter 使用。

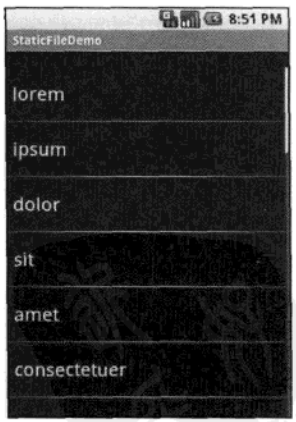


图 23-1　StaticFileDemo 示例应用程序

所得的 Activity 看起来与之前的相同，如图 23-1 所示，因为单词列表是一样的，只是改变了它们的位置。

当然，还有一种更简单的方法来以打包文件的形式使用 XML 文件，比如使用 XML 资源（第 20 章介绍）。但是，虽然此示例使用 XML，该文件却可以简单地看成一个每行一个单词的列表，或者 Android 资源系统本机无法处理的其他格式。

23.2　读取与写入

读取与写入自己特定于应用程序的数据文件与在桌面 Java 应用程序中的做法几乎是一样的。

关键在于对 Activity 或其他 Context 使用 openFileInput()或 openFileOutput()，以便分别获取 InputStream 或 OutputStream。此外，这就与常规 Java I/O 逻辑没有太多区别了：

- 根据需要包装这些流，如使用 InputStreamReader 或 OutputStreamWriter 处理基于文本的 I/O；
- 读取或写入数据；
- 使用 close()在完成后释放流。

如果两个应用程序都尝试通过 openFileInput()读取 notes.txt 文件，那么每个应用程序都可以访问自己版本的文件。如果需要让一个文件能够从多个位置访问，那么应该创建一个内容提供程序，详见第 27 章。

注意，openFileInput()和 openFileOutput()不接受文件路径（例如，path/to/file.txt)），只接受简单的文件名。

以下布局摘自 Files/ReadWrite 示例应用程序，可用于大部分简单的文本编辑器：

```
<?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/close"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Close" />
  <EditText
    android:id="@+id/editor"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:singleLine="false"
    android:gravity="top"
    />
</LinearLayout>
```

我们得到的将是一个大的文本编辑部件，其上有一个 Close 按钮。

Java 实现要稍微复杂一点：

```
package com.commonsware.android.readwrite;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import java.io.BufferedReader;
import java.io.File;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
```

```
public class ReadWriteFileDemo extends Activity {
  private final static String NOTES="notes.txt";
  private EditText editor;

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    editor=(EditText)findViewById(R.id.editor);

    Button btn=(Button)findViewById(R.id.close);

    btn.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        finish();
      }
    });
  }

  public void onResume() {
    super.onResume();

    try {
      InputStream in=openFileInput(NOTES);

      if (in!=null) {
        InputStreamReader tmp=new InputStreamReader(in);
        BufferedReader reader=new BufferedReader(tmp);
        String str;
        StringBuffer buf=new StringBuffer();

        while ((str = reader.readLine()) != null) {
          buf.append(str+"\n");
        }

        in.close();
        editor.setText(buf.toString());
      }
    }
    catch (java.io.FileNotFoundException e) {
      // that's OK, we probably haven't created it yet
    }
    catch (Throwable t) {
      Toast
        .makeText(this, "Exception: "+t.toString(), 2000)
        .show();
    }
  }

  public void onPause() {
    super.onPause();

    try {
      OutputStreamWriter out=
          new OutputStreamWriter(openFileOutput(NOTES, 0));

      out.write(editor.getText().toString());
      out.close();
    }
```

```
      catch (Throwable t) {
        Toast
          .makeText(this, "Exception: "+t.toString(), 2000)
          .show();
      }
    }
  }
```

首先，我们连接按钮，以便在用户单击该按钮时，使用 setOnClickListener()对 Activity 调用 finish()关闭 Activity。

接下来，我们挂钩 onResume()，以便在编辑器启动或者从冻结状态恢复时进行控制。我们使用 openFileInput()读取 notes.txt，并将其内容放入文本编辑器。如果没有找到文件，则假设 Activity 是第一次运行（或者使用其他方法删除了文件），我们只需要让编辑器留空即可。

最后，我们挂钩 onPause()，以便在其他 Activity 隐藏我们的 Activity 或者通过 Close 按钮关闭 Activity 时对其进行控制。此处我们使用 openFileOutput()打开 notes.txt，将在其中放入文本编辑器的内容。

我们将得到一个持久性的记事本，如图 23-2 和图 23-3 所示。键入的内容都将保留（除非进行了删除），即使在 Activity 关闭、电话关机以及其他类似的情况下也是如此。

欢迎在外部存储空间(如SD卡)进行读取和写入。使用 Environment.getExternalStorageDirectory()获取 SD 卡根目录下的 File 对象。从 Android 1.6 开始，还需要相应的权限才能处理外部存储空间。（例如，WRITE_EXTERNAL_STORAGE。）权限问题详见第 28 章。

记住，所有应用程序都可以访问外部存储空间，而 openFileInput()和 openFileOutput()则只能作用于应用程序私有空间。

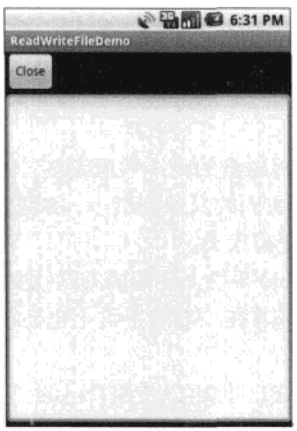


图 23-2 第一次启动的 ReadWriteFileDemo 示例应用程序

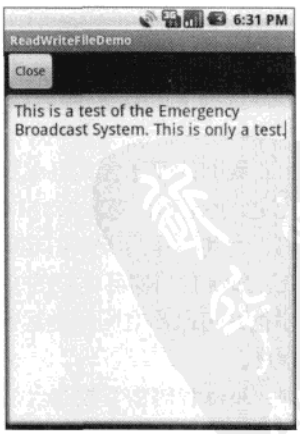


图 23-3 输入某些文本之后的 ReadWriteFileDemo 应用程序

第24章 充分利用Java库

与其他现代编程语言相比，Java有很多第三方库。我这里所说的第三方库指的是可以在服务器或桌面Java应用程序中包含的众多JAR文件——Java SDK本身不提供的文件。

在Android的情况下，DalvikVM（Virtual Machine，虚拟机）本质上不是真正的Java，它的SDK提供的内容与传统的Java SDK有所不同。也就是说，许多Java第三方库仍然提供Android本机缺少的功能，因此对于项目也许会有用。（如果你能够让它们处理Android样式的Java。）

本章将介绍如何充分利用这些库，以及Android对第三方代码的支持限制。

24.1 外部限制

并非所有可用的Java代码都能够很好地兼容Android。需要考虑的因素有很多，包括以下方面。

- 预计的平台API：代码采用的JVM（Java Virtual Machine，Java虚拟机）比Android的要新吗？代码采用的Java API包含在Java SE中，但没有包含在Android中（比如Swing）吗？
- 大小：现有的用于台式机或服务器的Java代码无需对磁盘的大小考虑太多，或者在某种程度上，甚至不会考虑RAM的大小。Android的这两方面的尺寸都很小。使用第三方Java代码，尤其在作为JAR打包时，可能会让应用程序变得太大。
- 性能：Java代码实际需要的CPU功能比大部分Android手机上使用的CPU更强吗？桌面程序也许可以没有问题地运行某个程序，但并不意味着在一般的手机上也能很好地运行该程序。
- 接口：Java代码需要控制台接口吗？还是说它是纯API，你可以使用自己的接口？

解决这些问题的一个技巧是使用开源Java代码，并实际处理该代码让它更适用于Android。例如，如果只用到10%的第三方库，那么值得将项目的子集重新编译为你需要的内容，或者至少

从 JAR 删除不需要的类。前一种方法更加安全，因为编译器可以帮助避免丢弃某些必要的代码片段，虽然真正做起来有些乏味。

24.2 Ant 和 JAR

将第三方代码集成到项目有两种选择：使用源代码或使用预先打包的 JAR。

如果选择使用源代码，则只需将其复制到源代码树（项目的 src/下），这样它就存在于现有的项目中了，然后让编译器完成其他工作。

如果选择使用现有 JAR（可能没有其源代码），那么只需要告诉构建链如何使用 JAR。首先，将 JAR 放入 Android 项目的 libs/目录。然后，如果使用 IDE，可能需要将 JAR 添加到构建路径。（Ant 将自动挑选 libs/中找到的所有 JAR。）

24.3 参照脚本

与其他移动设备操作系统不同，Android 对于可以运行的内容没有限制，只要能够使用 Dalvik VM 在 Java 中运行即可。因此，可以在应用程序中合并你自己的脚本语言，而这在其他一些设备上是明确禁止的。

一种可能的 Java 脚本语言是 BeanShell（http://beanshell.org）。BeanShell 提供可兼容 Java 的语法，使用隐式类型，无需编译。

要添加 BeanShell 脚本，需要将 BeanShell 解释器的 JAR 文件放入 libs/目录。遗憾的是，可以从 BeanShell 站点下载的 2.0b4 JAR 不能在 Android 0.9 和更新的 SDK 上实现开箱即用（可能是用于构建它的编译器造成的）。替代方法查看 Subversion 的源代码，并执行 `ant jarcore` 进行构建，然后将得到的 JAR（位于 BeanShell 的 dist/目录）复制到自己项目的 libs/。此外，也可以使用本书源代码随附的 BeanShell JAR（在 Java/AndShell 项目中）。

接下来，在 Android 上使用 BeanShell 的方式与在任何其他 Java 环境中使用 BeanShell 的方式没有什么不同：

(1) 创建 BeanShell `Interpreter` 类的实例；
(2) 通过 `Interpreter#set()`为脚本的使用设置任何全局变量；
(3) 调用 `Interpreter#eval()`运行脚本，还可以选择获得脚本最后一个语句的结果。

例如，下面是最小型的 BeanShell IDE 的 XML 布局：

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

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
    android:id="@+id/eval"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Go!"
    />
<EditText
    android:id="@+id/script"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:singleLine="false"
    android:gravity="top"
    />
</LinearLayout>
```

同时包含以下 Activity 实现：

```
package com.commonsware.android.andshell;

import android.app.Activity;
import android.app.AlertDialog;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import bsh.Interpreter;

public class MainActivity extends Activity {
  private Interpreter i=new Interpreter();

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

    Button btn=(Button)findViewById(R.id.eval);
    final EditText script=(EditText)findViewById(R.id.script);
    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        String src=script.getText().toString();

        try {
          i.set("context", MainActivity.this);
          i.eval(src);
        }
        catch (bsh.EvalError e) {
          AlertDialog.Builder builder=
                    new AlertDialog.Builder(MainActivity.this);

          builder
            .setTitle("Exception!")
            .setMessage(e.toString())
            .setPositiveButton("OK", null)
            .show();
        }
```

```
        }
      });
    }
}
```

编译并运行（包括合并上文提到的 BeanShell JAR），然后在模拟器上安装它。将其激活，你将得到一个简单的 IDE，其中有一个很大的文本区域用于脚本，还有一个大大的 Go!按钮用来执行脚本，如图 24-1 所示。

```
import android.widget.Toast;

Toast.makeText(context, "Hello, world!", 5000).show();
```

注意，在制作 Toast 时使用 context 引用 Activity，即让 Activity 引用本身的是一个全局设置。此全局变量可任意命名，只要 set()调用和脚本代码使用的名称与此相同。

单击 Go!按钮，就会得到如图 24-2 所示的结果。

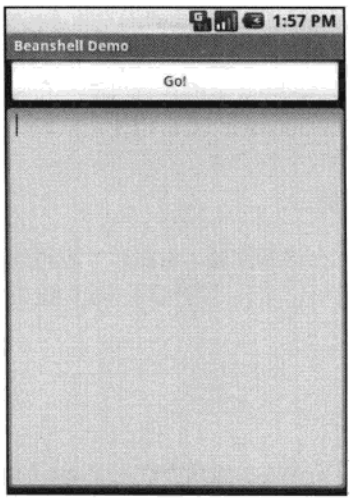


图 24-1 AndShell BeanShell IDE

图 24-2 执行某些代码的 AndShell BeanShell IDE

下面是一些需要注意的内容。

- 并非所有脚本语言都能正常使用。例如，实现自己的 JIT（Just-In-Time，即时）编译形式、动态生成 Java 字节码的脚本语言，可能需要添加参数才能生成 Dalvik VM 字节码。通过解析的脚本执行，调用 Java 反射 API 回调编译的类的脚本语言也许能更好地运行。尽管如此，如果该语言依赖于 Dalvik 中不存在的传统 Java API 的功能，那么它有些功能也无法正常运行。例如，BeanShell 或者 JAR 追加项中可能存在某些内容无法在如今的 Android 上工作。
- 未 JIT 的脚本语言必然要比经过编译的 Dalvik 应用程序慢。速度更慢意味着用户体验不

佳，也意味着同样的工作量需要消耗的电量更多。因此，如果只是因为感觉在 BeanShell 构建整个 Android 应用程序要容易而这样做，可能会让用户不高兴。

- 公开整个 Java API 的脚本语言，比如 BeanShell，可以执行底层 Android 安全模型允许的任何操作。因此，如果应用程序有 READ_CONTACTS 权限，那么该应用程序运行的任何 BeanShell 脚本也将具有相同的权限（权限问题详见第 28 章）。
- 最后，但同样重要的是，语言解释器 JAR 通常比较大。本例中使用的 BeanShell JAR 有 200KB。考虑到它的能耐，这是可以理解的，但是太大会让使用 BeanShell 的应用程序更难下载，占用更多的手机空间，等等。

24.4　滴酒不沾

正如上文所述，并非所有 Java 代码都能在 Android 和 Dalvik 上运行。以下是一些示例。

- 如果 Java 代码预期在 Java SE、Java ME 或 Java EE 上运行，它可能会缺少一些 Android 没有提供的 API。例如，有些制图库需要 Swing 或 AWT 基类，而这些在 Android 上一般不提供。
- Java 代码可能依赖于其他 Java 代码，而这些代码在 Android 上的运行可能存在问题。例如，你可能不想使用 Android 绑定的 Apache HttpComponents，而希望使用依赖较旧（或较新）Apache HttpComponents 版本的 JAR。
- Java 代码可能使用了 Dalvik 引擎无法使用的语言功能。

在这些情况下，如果有一个编译过的 JAR，在编译时不会遇到问题，但在运行应用程序时可能会出现问题。因此，如果可能，最好使用 Android 开源代码，同自己的代码一起构建第三方代码，以便快速发现问题。

24.5　评审脚本

由于本章涉及了 Android 中的脚本，你可能想知道，除了直接在项目中嵌入 BeanShell 之外，是否还有其他选择。

目前对其他基于 JVM 的编程语言进行了一些试验，这些语言包括 JRuby 和 Jython。它们对 Android 的支持还不完整，但已经在完善之中。

此外，Android Market 已经开始提供 ASE（Android Scripting Environment），你可以使用 Python 和 Lua 与 BeanShell 配合编写脚本。这些脚本不是功能全面的应用程序，在本书编写之际，还没有正式向其他人分发。还要注意，ASE 并非用于扩展其他程序，尽管可以用这种方式使用它。但是，如果想进行手机上的编程，ASE 可能就是最佳解决方案。更多有关 ASE 的信息，请参见其项目网页 http://code.google.com/p/android-scripting/。

第 25 章 通过 Internet 进行通信

大部分 Android 手机都有内置的 Internet 访问工具，这可能是 Wi-Fi、蜂窝数据服务（EDGE、3G 等），或者其他完全不同的技术。不管使用哪种技术，大部分（至少那些有数据计划或使用 Wi-Fi 访问的）人都能够通过 Android 手机访问 Internet。

无疑，Android 平台提供了很多方法帮助开发人员利用这种 Internet 访问。有些 Android 平台提供高级访问支持，比如集成的 WebKit 浏览器组件。如果需要，还可以使用最底层的 raw 套接字。在此之间，可以利用能访问特定协议（HTTP、XMPP、SMTP 等）的 API，包括手机上的 API 和第三方 JAR。

本书重点介绍较高级别的访问：WebKit 组件（已在第 13 章中介绍）和 Internet 访问 API（将在本章介绍）。在这个快节奏的世界，我们应该尝试重用现有的组件，而不是使用自己的在线协议。

25.1 REST 和 Relaxation

Android 没有内置的 SOAP 或 XML-RPC 客户端 API。但是，它的确融入了 Apache HttpComponents 库。你可以基于该库放置 SOAP/XML-RPC 层，也可以“直接”使用它访问 REST 式网页服务。本书将 REST 式网页服务视为简单的 HTTP 请求，即在完整的 HTTP 动词上加以普通的 URL，并将格式化的有效载荷（XML、JSON 等）视作响应。

可以访问 HttpClient 网站（http://hc.apache.org/）查找更多详细教程、常见问题解答、入门文档。这里我们以如何查看天气为例，只介绍基础知识。

25.2 通过 Apache HttpClient 操作 HTTP

HttpComponents 的 HttpClient 组件可以处理所有 HTTP 请求。使用 HttpClient 的第一步当然是创建一个 `HttpClient` 对象。由于 `HttpClient` 是一个接口，需要实际实例化该接口的某个实现，

如 DefaultHttpClient。

这些请求将绑定到 HttpRequest 实例，对于不同的 HTTP 动词使用不同的 HttpRequest 实现（例如，HttpGet 用于 HTTP GET 请求）。你创建一个 HttpRequest 实现实例，填写要检索的 URL 以及其他配置数据（例如，如果通过 HttpPost 执行 HTTP POST，则是表单值），然后将方法传递到客户端，通过 execute()进行实际的 HTTP 请求。

这时发生的事情可以简单，也可以复杂。你可以获取 HttpResponse 对象，以及响应代码（例如，200 表示 OK）、HTTP 头等。你也可以使用某种 execute()，将 ResponseHandler<String>作为参数，execute()将返回响应正文的字符串表示形式。在实际操作中，不推荐使用这个方法，因为你应该检查 HTTP 响应代码是否有错误。但是对于简单的应用程序，如本书的示例，使用 ResponseHandler<String>方法是可以的。

让我们看看 Internet/Weather 示例项目。该项目实现一个 Activity，针对你当前的位置从 National Weather Service（国家气象局）检索天气数据。（注意，这可能只在美国地区有效。）然后，将该数据转换为 HTML 页面，后者将放入 WebKit 部件中以便显示。作为练习，读者可以使用 ListView 重新构造此演示内容。同样，由于此示例相对较长，我们只展示与本章内容有关的 Java 代码片段，你可以从 Apress 网站下载完整的源代码。

为了增加趣味性，我们使用 Android 位置服务确定我们的位置。有关此方面的详细内容将在第 32 章介绍。

在 onResume()方法中，我们打开位置更新，现在可以得到位置信息了，在我们移动足够长的距离时（10 km）将得到通知。位置信息可用时（无论是以起点为基础还是以移动距离为基础），我们就可以通过 updateForecast()方法检索 National Weather Service 数据：

```
private void updateForecast(Location loc) {
  String url=String.format(format, loc.getLatitude(),
                           loc.getLongitude());
  HttpGet getMethod=new HttpGet(url);

  try {
    ResponseHandler<String> responseHandler=new BasicResponseHandler();
    String responseBody=client.execute(getMethod,
                                       responseHandler);

    buildForecasts(responseBody);
    String page=generatePage();

    browser.loadDataWithBaseURL(null, page, "text/html",
                                "UTF-8", null);
  }
  catch (Throwable t) {
    Toast
      .makeText(this, "Request failed: "+t.toString(), 4000)
```

```
        .show();
    }
  }
```

updateForecast()方法采用 Location 作为参数，Location 通过位置更新进程获取。现在，你需要知道的就是 Location 支持 getLatitude()和 getLongitude()方法，这两个方法将分别返回手机位置的纬度和经度。

我们以字符串资源的形式保存 National Weather Service XML 的 URL，并在运行时放入纬度和经度信息。由于在 onCreate()中创建了 HttpClient 对象，我们可以使用自定义的 URL 填充 HttpGet，然后执行该方法。给定从 REST 服务得到的 XML，我们就可以构建预测 HTML 页面（将在下一节介绍），并将其放入 WebKit 部件。如果 HttpClient 遇到异常，我们可以将错误作为 Toast 提供。

25.3 解析响应

使用一些系统可以格式化所得到的响应，这些系统如 HTML、XML、JSON 等。当然，你可以挑选所需的信息，并执行某些有用的操作。在 WeatherDemo 示例中，我们需要提取预测时间、温度和图标（指示天空情况和降水量），并据此生成一个 HTML 页面。

Android 包括 3 个 XML 解析器：传统的 W3C DOM 解析器（org.w3c.dom）、SAX 解析器（org.xml.sax），以及 XML pull 解析器（已在第 20 章介绍）。还有一个 JSON 解析器（org.json）。

如有可能，欢迎使用第三方 Java 代码处理其他格式，如用于 feed 阅读器的专用 RSS/Atom 解析器。使用第三方 Java 代码已在第 24 章中介绍。

对于 WeatherDemo，我们在 buildForecasts()方法中使用 W3C DOM 解析器：

```
void buildForecasts(String raw) throws Exception {
  DocumentBuilder builder=DocumentBuilderFactory.newInstance().newDocumentBuilder();
  Document doc=builder.parse(new InputSource(new StringReader(raw)));
  NodeList times=doc.getElementsByTagName("start-valid-time");

  for (int i=0;i<times.getLength();i++) {
    Element time=(Element)times.item(i);
    Forecast forecast=new Forecast();
    forecasts.add(forecast);
    forecast.setTime(time.getFirstChild().getNodeValue());
  }

  NodeList temps=doc.getElementsByTagName("value");

  for (int i=0;i<temps.getLength();i++) {
    Element temp=(Element)temps.item(i);
```

```
      Forecast forecast=forecasts.get(i);

      forecast.setTemp(new Integer(temp.getFirstChild().getNodeValue()));
    }

    NodeList icons=doc.getElementsByTagName("icon-link");

    for (int i=0;i<icons.getLength();i++) {
      Element icon=(Element)icons.item(i);
      Forecast forecast=forecasts.get(i);

      forecast.setIcon(icon.getFirstChild().getNodeValue());
    }
  }
```

National Weather Service XML格式的结构很奇怪，严重依赖于列表中的连续位置，而在RSS或Atom之类的格式中你会发现更多面向对象的样式。也就是说，利用我们需要的元素（start-valid-time用于预测时间，value用于温度，icon-link用于图标URL）在文档中唯一这个事实，我们可以有一定的自由，简化内容的解析。

HTML以InputStream的形式出现，并进入DOM解析器。此时，我们可以扫描start-valid-time元素，并使用这些起始时间填充Forecast模型。然后，我们可以查找温度的value元素和icon-link URL，并将其填入Forecast对象。

接下来，generatePage()方法将创建一个基本的预测HTML表格：

```
String generatePage() {
  StringBuffer bufResult=new StringBuffer("<html><body><table>");

  bufResult.append("<tr><th width=\"50%\">Time</th>"+
                   "<th>Temperature</th><th>Forecast</th></tr>");

  for (Forecast forecast : forecasts) {
    bufResult.append("<tr><td align=\"center\">");
    bufResult.append(forecast.getTime());
    bufResult.append("</td><td align=\"center\">");
    bufResult.append(forecast.getTemp());
    bufResult.append("</td><td><img src=\"");
    bufResult.append(forecast.getIcon());
    bufResult.append("\"></td></tr>");
  }
  bufResult.append("</table></body></html>");

  return(bufResult.toString());
}
```

所得结果如图25-1所示。

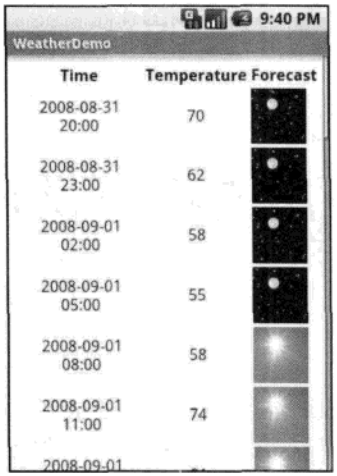


图 25-1 WeatherDemo 示例应用程序

25.4 要考虑的问题

如果需要使用 SSL（Secure Sockets Layer，安全套接字层）协议，记住默认的 HttpClient 设置不包括 SSL 支持。这主要是因为你需要决定如何处理 SSL 证书展示。你是无条件接受所有证书，包括自签名或过期的证书吗？还是希望询问用户是否真的要使用某个陌生的证书？

类似地，HttpClient 组件默认情况下只用于单线程。如果要在可能涉及多线程的某个服务或其他地方使用 HttpClient，也完全可以设置 HttpClient 以支持多线程。

对于这些主题，可以访问 HttpClient 网站查找相关的文档和支持。

第 26 章 使用内容提供程序

Android 中的任何以 content://模式开始的 URI 表示由某个内容提供程序提供的资源。内容提供程序使用 URI 实例作为句柄提供数据封装。你无需知道也不用关心 URI 表示的数据来自何处，只要在需要时能够获取就没有问题。这些数据可能存储在 SQLite 数据库、纯文本文件，或者通过 Internet 访问的某个远程服务器上。

给定了 URI，就可以使用内容提供程序执行基本的 CRUD（创建、读取、更新、删除）操作。URI 实例可以表示集合，也可以表示单独的内容片段。给定集合 URI 后，可以通过插入操作创建新的内容片段。给定实例 URI 后，可以读取 URI 表示的数据，更新数据或直接删除实例。

Android 允许使用现有的内容提供程序，也允许创建内容提供程序。本章将介绍如何使用内容提供程序。第 27 章将介绍如何将内容提供程序框架服务于数据。

26.1 数据片段

内容 URI 有一个简化的结构模型，即模式、数据的名称空间，以及可选的实例标识符——所有这些都使用 URL 样式的记号以斜杠分隔。内容 URI 的模式始终为 content://。

因此，内容 URI content://constants/5 表示标识符为 5 的 `constants` 实例。

模式和名称空间的组合称为内容提供程序的基 URI，或者说是内容提供程序支持的数据集合。在上一个示例中，content://constants 是一个内容提供程序的基 URI，提供有关“constants”（本例中是物理常量）的信息。

基 URI 也可能更加复杂。例如，如果联系人的基 URI 是 content://contacts/people，那么 Contacts 内容提供程序可能需要使用其他基 URI 值提供其他数据。

基 URI 表示实例集合。基 URI 与实例标识符（如 5）组合后可表示一个实例。

大部分 Android API 都将其视为 URI 对象，但是在一般的讨论中，基 URI 被简单地视为字符串。Uri.parse()静态方法可以通过字符串表示形式创建一个 URI。

26.2 获得句柄

那么，这些 URI 实例从何而来呢？

如果知道需要处理的数据类型，通常是用代码从内容提供程序本身获取基 URI。例如，CONTENT_URI 是表示为 people 的联系人的基 URI；这将映射到 content://contacts/people。如果需要的是集合，则此 URI 按原样处理。如果需要实例并知道其标识符，可以对 URI 调用 addId()进行注入操作，以便拥有该实例的 URI。

还可能从其他源获取 URI 实例，比如通过对应于 ACTION_PICK intent 的子活动获取联系人的 URI 句柄。在这种情况下，URI 实际上是个不透明句柄，除非你决定使用 URI 类上的各种 getter 将其挑选出来。

还可以硬连接字面值 String 对象（例如"content://contacts/people"）并通过 Uri.parse()将其转换为 URI 实例。这不是一个理想的解决方案，因为基 URI 值可能会随着时间的推移而更改。例如，由于子系统的全面修订，Contacts 内容提供程序的基 URI 不再是 content://contacts/people。

26.3 查询

给定基 URI 后，就可以运行查询，从与该 URI 有关的内容提供程序返回数据。这很像 SQL——你指定要返回的“列”，利用约束确定要返回的“行”，排序等。不同之处在于，此请求是对内容提供程序做出的，而不是直接请求某些数据库（例如 SQLite）。

连接它们的是活动可用的 managedQuery()方法。此方法带有 5 个参数：

- 要查询的内容提供程序的基 URI，或者要查询的特定对象的实例 URI；
- 实例属性数组，取自你希望查询返回内容的内容提供程序；
- 约束语句，功能类似于 SQL WHERE 子句；
- 绑定到约束子句的可选参数集，替换其中出现任何?字符；
- 可选排序语句，功能类似于 SQL ORDER BY 子句。

此方法将返回一个 Cursor 对象，你可以用它检索查询所返回的数据。

内容提供程序的属性就像是数据库的列。换句话说，查询返回的每个实例（行）都包括一组属性（列），每个列都表示一段数据。

用一个示例来说明可能更直观。

我们的内容提供程序示例取自 ContentProvider/ConstantsPlus 示例应用程序，明确地说是 ConstantsBrowser 类：

```
constantsCursor=managedQuery(Provider.Constants.CONTENT_URI,
                             PROJECTION, null, null, null);
```

调用 managedQuery()时，我们提供以下内容。

- 调用方传入活动的 URI（CONTENT_URI）；在此例中，表示内容提供程序管理的物理常量集合。
- 要检索的属性列表。
- 3 个 null 值，指示我们不需要约束子句（URI 表示我们需要的实例），约束子句不需要参数，不需要排序。（我们只能获取一个项。）

```
private static final String[] PROJECTION = new String[] {
    Provider.Constants._ID, Provider.Constants.TITLE,
    Provider.Constants.VALUE};
```

此处最“神奇”的地方是属性列表。内容提供程序本身的文档（或源代码）应该为给定内容提供程序提供可能的属性列。在本例中，我们在 Provider 内容提供程序实现类上定义了各种逻辑值，表示各种属性（即唯一标识符、显示名称或标题以及常量值）。

26.4　适应环境

现在，我们通过 managedQuery()获得了 Cursor，因而可以访问查询结果，并使用其执行任何想要的操作。例如，我们可以手动从 Cursor 中提取数据，用于填充部件或其他对象。

但是，如果查询的目标是返回用户可以从中选择项的列表，那么应该考虑使用 SimpleCursorAdapter。此类连接 Cursor 和选择部件，如 ListView 或 Spinner。将 Cursor 提交给 SimpleCursorAdapter，将适配器提交给部件，部件将展示可用的选项。

例如，以下是 ConstantsBrowser 的 onCreate()方法，将为用户提供一列物理常量：

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

  constantsCursor=managedQuery(Provider.Constants.CONTENT_URI,
                               PROJECTION, null, null, null);

  ListAdapter adapter=new SimpleCursorAdapter(this,
                      R.layout.row, constantsCursor,
                      new String[] {Provider.Constants.TITLE,
                                    Provider.Constants.VALUE},
                      new int[] {R.id.title, R.id.value});
```

```
    setListAdapter(adapter);
    registerForContextMenu(getListView());
}
```

执行 managedQuery()并获取 Cursor 之后，ConstantsBrowser 将使用以下参数创建一个 SimpleCursorAdapter。

- 创建适配器的活动（或其他 Context）；本例中是 ConstantsBrowser 本身。
- 用于呈现列表项的布局标识符（R.layout.row）。
- 指针（constantsCursor）。
- 从指针中获取并用于配置列表项 View 实例的属性（TITLE 和 VALUE）。
- TextView 部件在列表项布局中对应的标识符，属性将用于该布局（R.id.title 和 R.id.value）。

这之后，我们将适配器放入 ListView，然后将得到如图 26-1 所示的结果。

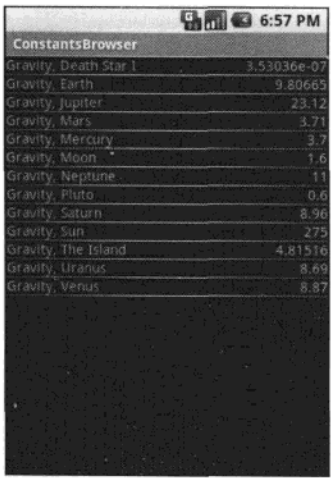


图 26-1 展示物理常量列表的 ConstantsBrowser

使用常用视图构造逻辑可以轻松控制视图，但是如果需要更高水平的控制，则可以创建 SimpleCursorAdapter 的子类，并覆写 getView()来创建自己的部件以访问列表，如本书前文所示。

当然，你也可以手动操作 Cursor（例如，moveToFirst()、getString()），这已经在第 22 章中介绍过。

26.5 舍与得

当然，如果不能从中添加或删除数据，不能进行更新，那么内容提供程序将没有太大作用。幸运的是，内容提供程序提供了这些功能。

要将数据插入内容提供程序，在 ContentProvider 接口上有两个选择（对活动调用 getContentProvider()来获取）：

- 调用 insert()并提供集合 URI 和 ContentValues 结构(它描述了要放入行的初始数据集)；
- 调用 bulkInsert()，并提供集合 URI，以及要一次填充几个行的 ContentValues 结构数组。

insert()方法返回一个 URI，可用于在新对象上执行以后的操作。bulkInsert()方法返回创建行的数目，需要执行查询才能检索刚刚插入的数据。

例如，以下 ConstantsBrowser 代码片段将向内容提供程序插入一个新的常量，给定可以访

问标题和常量值的 DialogWrapper：

```
private void processAdd(DialogWrapper wrapper) {
  ContentValues values=new ContentValues(2);

  values.put(Provider.Constants.TITLE, wrapper.getTitle());
  values.put(Provider.Constants.VALUE, wrapper.getValue());
  getContentResolver().insert(Provider.Constants.CONTENT_URI,
                              values);
  constantsCursor.requery();
}
```

由于已经有出色的 Cursor 指向内容提供程序的内容，所以我们可以调用 requery()更新 Cursor 的内容。这又可以更新包装 Cursor 的任何 SimpleCursorAdapter，并且将更新任何使用适配器的选择部件（例如 ListView）。

要从内容提供程序中删除一行或多行，可以使用 ContentResolver 上的 delete()方法。它的工作原理类似于 SQL DELETE 语句，带有 3 个参数：

- 表示要更新的集合（或实例）的 URI；
- 约束语句，功能类似于 SQL WHERE 子句，确定要更新哪些行；
- 绑定到约束子句的可选参数集，用于替换所出现的任何?字符。

26.6　感知 BLOB

许多数据库都支持 BLOB（Binary Large Objects，二进制大对象），包括 SQLite。但是，Android 模型的目标更多的集中在通过其自己的独立 URI 值支持此类数据块。因此，内容提供程序不能通过 Cursor 直接访问二进制数据，比如照片，而是要通过它的一个属性来提供特定于 BLOB 的内容 URI。你可以使用 ContentProvider 上的 getInputStream()和 getOutputStream()读取和写入二进制数据。

基本原理是最大限度减少不必要的数据复制。例如，在 Android 中照片的主要用途是向用户显示内容。ImageView 部件可以通过 JPEG 文件的内容 URI 做到这一点。存储照片并让它拥有自己的 URI，就不需要将内容提供程序的数据复制到某些临时存储区域（以便显示它），使用 URI 就足够了。可以预见，Android 应用程序一般不仅会上传二进制数据，还使用部件或内置活动显示该数据。

第 27 章 构建内容提供程序

构建内容提供程序可能是所有 Android 开发工作中最复杂、最乏味的任务了。就方法实现和要提供的公共数据成员而言，内容提供程序有很多要求。而且，只有实际使用内容提供程序之后，才能知道它是否配置正确。（不能构建活动并从资源编译器获取验证错误。）

可以说，如果应用程序希望向其他应用程序提供数据，那么构建内容提供程序尤为重要。如果应用程序只是将数据留给自己使用，那么可以不用创建内容提供程序，只需通过 Activity 直接访问数据。但是，如果希望数据能够供他人使用，例如，构建一个 feed 阅读器并希望其他程序能够访问你下载和缓存的 feed，那么内容提供程序正是你所需要的。

本章将展示 ContentProvider/ConstantsPlus 应用程序的一些示例代码。这个基本应用程序第一次出现在第 22 章，但在这里重新进行了编写，将数据库逻辑引入内容提供程序，然后供 Activity 使用。

27.1 剖析

在上一章中我们讨论过，内容 URI 是使用内容提供程序访问内部数据背后的关键所在。使用内容提供程序时，你只需要了解提供程序的基 URI 即可。然后，就可以根据需要运行查询，如果知道实例标识符，还可以构造特定实例的 URI。

但是，构建内容提供程序时，你需要更多有关内容 URI 内部结构的知识。

内容 URI 有 2 到 4 个片段，视情况而定。

- 它总是有一个模式（content://），表示它是一个内容 URI 而不是指向网络资源的 URI（http://）。
- 它总是有一个颁发机构，即模式之后的第一个路径段。颁发机构是一个唯一的字符串，标识了处理与此 URI 关联内容的内容提供程序。

- 它可能有一个数据类型路径，即颁发机构之后和实例标识符（如果有）之前的路径段列表。如果内容提供程序只处理一种类型的内容，那么数据类型路径可以为空。它可以是单路径段（foo），也可以是一个路径段链（foo/bar/goo），可以根据需要处理内容提供程序所需的数据访问场景。
- 它可能有一个实例标识符，这是一个标识特定内容片段的整数。不带实例标识符的内容 URI 引用颁发机构表示的内容集合（以及提供的数据路径）。

例如，内容 URI 可以是 content://sekrits 这种简单的形式，表示联系 `sekrits` 颁发机构（例如，`SecretsProvider`）的内容提供程序所保存的内容集合。它也可以是 content://sekrits/card/pin/17 这种复杂的形式，表示 `Sekrits` 内容提供程序管理的、数据类型为 `card/pin` 的内容片段（标识为 17）。

27.2　类型

接下来，你需要了解与内容提供程序提供的内容相对应的 MIME 类型。

Android 使用内容 URI 和 MIME 类型作为标识手机上内容的方式。内容 URI 集合，或者准确地说，颁发机构和数据类型路径的组合应该映射到一对 MIME 类型。一个 MIME 类型表示集合；另一个表示实例。这些 URI 模式的映射分别在非标识符和标识符之前的部分列出。正如本书前面几章所述，可以将 MIME 类型填入 `Intent`，以便将 `Intent` 路由到相应的活动。（例如，集合 MIME 类型上的 `ACTION_PICK` 可以调用选择活动，以从该集合中挑选一个实例。）

集合 MIME 类型的形式应该是 vnd.X.cursor.dir/Y，其中 *X* 是公司、组织或者项目的名称，*Y* 是句点分隔的类型名称。例如，你可以使用 vnd.tlagency.cursor.dir/sekrits.card.pin 作为机密内容集合的 MIME 类型。

实例 MIME 类型的形式应该是 vnd.X.cursor.item/Y，通常 *X* 和 *Y* 值与用于集合 MIME 类型的 *X* 和 *Y* 值相同（但这不是严格规定）。

27.3　创建内容提供程序

创建内容提供程序涉及 4 个基本步骤：创建提供程序类，提供 URI，声明属性，更新清单文件（Manifest）。

27.3.1　第一步：创建提供程序类

就像 `Activity` 和 `Intent` 接收器都是 Java 类一样，内容提供程序也是如此。因此，创建内容提供程序的首要步骤就是以 `ContentProvider` 为基类，创建 Java 子类。

在 ContentProvider 的子类中，需负责实现 6 个方法，结合这些方法可以执行内容提供程序希望提供给 Activity 的服务，用于创建、读取、更新或删除内容。

1. onCreate()

与 Activity 一样，内容提供程序主要的入口点是 onCreate()。在此可以做任何想要的初始化。具体来说，应该在这里延迟初始化数据存储库。例如，如果计划在 SD 卡的某个目录中存储数据，并计划使用 XML 文件存储数据，那么应该检查是否存在该目录和 XML 文件；如果不存在，应该创建它们，并让内容提供程序的其他部分知道它们已经可用。

类似地，如果已经充分地重写了内容提供程序，让数据存储库改变了结构，那么应该检查你现在拥有的结构，如果已经过期，则应该进行调整。你不用编写自己的“安装”程序。这意味着在调用 onCreate()时，你没有太好的办法确定对于内容提供程序来说这是第一次启动还是常规启动，也无法确定对于新发布的内容提供程序（就地升级）来说这是第一次启动还是常规启动。

例如，下面是一个 Provider 的 onCreate()方法，摘自 ContentProvider/ConstantsPlus 示例应用程序：

```
@Override
public boolean onCreate() {
  db=(new DatabaseHelper(getContext())).getWritableDatabase();

  return (db == null) ? false : true;
}
```

这段代码无法看到所有特殊之处，因为其神奇之处在于私有 DatabaseHelper 对象，详见第 22 章。

2. query()

可以预计，query()方法可以让内容提供程序获取某个 Activity 希望执行的查询详情。由你决定如何处理查询。

query()方法使用以下内容作为参数：

- 表示要查询的集合或实例的 URI；
- 表示应该返回的属性列表的 String[]；
- 表示等同于 SQL WHERE 子句的 String，限制了应该视为查询结果的实例；
- 表示要用于 WHERE 子句中的 String[]，用于替换其中的?字符；
- 表示等同于 SQL ORDER BY 子句的 String。

你负责解释这些参数的意义，并返回可以迭代和访问数据的 Cursor。

可以想象，这些参数针对的是使用 SQLite 数据库进行存储的人。你可以忽略其中某些参数（例如，你可以选择不推出自己的 SQL WHERE 子句解析器），但是需要记录下来，以便 Activity 仅根据实例 URI 进行查询，不使用你选择不进行处理的参数。

但是，对于以 SQLite 为基础的存储提供程序，query()方法实现几乎可以作为样板。使用 SQLiteQueryBuilder 可以将各种参数转换为一个 SQL 语句，对构造器使用 query()实际上将调用查询并返回一个 Cursor。Cursor 即是 query()方法返回的内容。

例如，以下是来自 Provider 的 query()：

```
@Override
public Cursor query(Uri url, String[] projection, String selection,
                    String[] selectionArgs, String sort) {
  SQLiteQueryBuilder qb=new SQLiteQueryBuilder();

  qb.setTables(getTableName());

  if (isCollectionUri(url)) {
    qb.setProjectionMap(getDefaultProjection());
  }
  else {
    qb.appendWhere(getIdColumnName()+"="+url.getPathSegments().get(1));
  }

  String orderBy;

  if (TextUtils.isEmpty(sort)) {
    orderBy=getDefaultSortOrder();
  } else {
    orderBy=sort;
  }

  Cursor c=qb.query(db, projection, selection, selectionArgs,
                    null, null, orderBy);
  c.setNotificationUri(getContext().getContentResolver(), url);
  return c;
}
```

我们创建 SQLiteQueryBuilder 并将查询详情放入构造器。注意，查询可能基于集合或实例 URI。如果是后一种情况，我们需要向查询添加实例 ID。添加之后，我们对构造器调用 query()方法获取 Cursor 作为结果。

3. insert()

insert()方法将接受表示集合和 ContentValues 结构的 URI，ContentValues 结构中为新实例的初始数据。你负责创建新实例，填入所提供的数据，并返回新实例的 URI。

如果这是基于 SQLite 的内容提供程序，其实现几乎可以作为样板。你只需要验证 Activity 提供了所有必需值，合并自己的默认值与所提供的数据，然后对数据库调用 insert()就可以实际

创建实例。

例如，以下是来自 Provider 的 insert()：

```
@Override
public Uri insert(Uri url, ContentValues initialValues) {
  long rowID;
  ContentValues values;

  if (initialValues!=null) {
    values=new ContentValues(initialValues);
  } else {
    values=new ContentValues();
  }

  if (!isCollectionUri(url)) {
    throw new IllegalArgumentException("Unknown URL " + url);
  }

  for (String colName : getRequiredColumns()) {
    if (values.containsKey(colName) == false) {
      throw new IllegalArgumentException("Missing column: "+colName);
    }
  }

  populateDefaultValues(values);

  rowID=db.insert(getTableName(), getNullColumnHack(), values);
  if (rowID > 0) {
    Uri uri=ContentUris.withAppendedId(getContentUri(), rowID);
    getContext().getContentResolver().notifyChange(uri, null);
    return uri;
  }
  throw new SQLException("Failed to insert row into " + url);
}
```

该模式与之前的相同：使用特定提供程序以及要插入的数据执行实际的插入操作。该示例要注意以下内容：

- 你只可以在集合 URI 中插入，因此我们调用 isCollectionUri()进行验证；
- 提供程序还了解需要的列（getRequiredColumns()），因此我们迭代这些列并确认我们提供的值符合要求；
- 提供程序还负责填入任何默认值（populateDefaultValues()），用于 insert()调用没有提供的列，以及 SQLite 表定义没有自动处理的列。

4. update()

update()方法可以接受实例的 URI 或集合，一个 ContentValues 结构（带有要应用的新值），一个用于 SQL WHERE 子句的 String，一个 String[]（带有替换 WHERE 子句中?字符的参数）。你的职责是标识要修改的实例（基于 URI 和 WHERE 子句），然后使用提供的值替换这些实例的当前属性值。

如果存储使用的不是 SQLite，这会非常繁琐。使用 SQLite，你可以将收到的所有参数传递给数据库的 update()调用，根据更新一个实例还是多个实例，update()调用可能稍微有所不同。

例如，以下是来自 Provider 的 update()：

```
@Override
public int update(Uri url, ContentValues values, String where, String[] whereArgs) {
  int count;

  if (isCollectionUri(url)) {
    count=db.update(getTableName(), values, where, whereArgs);
  }
  else {
    String segment=url.getPathSegments().get(1);
    count=db
        .update(getTableName(), values, getIdColumnName()+"="
            + segment
            + (!TextUtils.isEmpty(where) ? " AND (" + where
                + ')' : ""), whereArgs);
  }

  getContext().getContentResolver().notifyChange(url, null);
  return count;
}
```

在本例中，更新的可以是特定的实例，也可以针对整个集合，因此我们查看 URI (isCollectionUri())，如果更新的是集合，则执行更新即可。如果将更新单个实例，则需要向 WHERE 子句添加一个约束，以便只更新请求的行。

5. delete()

与 update()相同，delete()的参数包括：一个表示要处理的实例或集合的 URI，以及一个 WHERE 子句。如果 Activity 将删除一个实例，URI 应该表示该实例，WHERE 子句可以为 null。但是，Activity 可能请求删除一个开放式实例集，那么应该使用 WHERE 子句限制要删除的内容。

与 update()一样，如果使用 SQLite 进行数据库存储，则这非常简单。(是不是有些熟悉？)你可以让它处理各种解析并应用 WHERE 子句。只需对数据库调用 delete()即可。

例如，以下是来自 Provider 的 delete()：

```
@Override
public int delete(Uri url, String where, String[] whereArgs) {
  int count;
  long rowId=0;

  if (isCollectionUri(url)) {
    count=db.delete(getTableName(), where, whereArgs);
  }
  else {
```

```
    String segment=url.getPathSegments().get(1);
    rowId=Long.parseLong(segment);
    count=db
        .delete(getTableName(), getIdColumnName()+"="
            + segment
            + (!TextUtils.isEmpty(where) ? " AND (" + where
                + ')' : ""), whereArgs);
  }

  getContext().getContentResolver().notifyChange(url, null);
  return count;
}
```

这与上一节中介绍的 update()实现基本相同，它要么删除整个集合的一个子集，要么删除一个实例（如果满足提供的 WHERE 子句）。

6. getType()

最后需要实现的方法是 getType()。该方法接受一个 URI 参数，并返回与该 URI 关联的 MIME 类型。URI 可以是集合，也可以是实例 URI。你需要确定提供的 URI 表示的是集合还是单一实例，并返回对应的 MIME 类型。

例如，以下是来自 Provider 的 getType()：

```
@Override
public String getType(Uri url) {
  if (isCollectionUri(url)) {
    return(getCollectionType());
  }

  return(getSingleType());
}
```

你可以看到，大部分逻辑都委托给私有 getCollectionType()和 getSingleType()方法：

```
private String getCollectionType() {
  return("vnd.android.cursor.dir/vnd.commonsware.constant");
}

private String getSingleType() {
  return("vnd.android.cursor.item/vnd.commonsware.constant");
}
```

27.3.2 第二步：提供 URI

你还需要在某处添加一个公有静态成员，为内容提供程序支持的每个集合包含一个对应的 URI。一般情况下，这是一个公有静态 final URI，位于内容提供程序类本身：

```
public static final Uri CONTENT_URI
    =Uri.parse("content://com.commonsware.android.constants.Provider/constants");
```

你对内容 URI 使用的名称空间可以与 Java 类相同，以减少发生冲突的可能性。

27.3.3　第三步：声明属性

还记得在上一章中使用内容提供程序时引用的属性吗？现在，你自己的内容提供程序也需要这些了。

具体来说，需要一个实现 BaseColumns 的公有静态类，其中包含属性名称，比如 Provider 中的此示例：

```
public static final class Constants implements BaseColumns {
  public static final Uri CONTENT_URI
      =Uri.parse("content://com.commonsware.android.constants.Provider/constants");
  public static final String DEFAULT_SORT_ORDER="title";
  public static final String TITLE="title";
  public static final String VALUE="value";
}
```

如果使用 SQLite 存储数据，属性名称常量的值应该是表中对应列的名称，因此可以将投影（属性数组）利用 query()传递到 SQLite，也可以利用 insert()或 update()传递 ContentValues。

注意，这里的内容都不规定属性类型。它们可以是字符串、整数及其他。最大的限制是 Cursor 可以通过其属性 getter 方法提供访问的内容。代码中没有任何强制执行类型安全的内容，这意味着你应该记录属性类型，以便使用该内容提供程序的用户知道会出现什么。

27.3.4　第四步：更新清单文件

将内容提供程序实现与应用程序其他部分结合的内容位于 AndroidManifest.xml 文件中。只需要添加一个<provider>作为<application>元素的子元素即可：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.constants">
  <application android:label="@string/app_name"
    android:icon="@drawable/cw">
    <provider android:name=".Provider"
            android:authorities="com.commonsware.android.constants.Provider" />
    <activity android:name=".ConstantsBrowser" 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>
```

android:name 属性是内容提供程序类的名称，前端的点表示它位于此应用程序类的常用名称空间中。（就像对 Activity 使用的一样。）

android:authorities 属性应该是内容提供程序支持的颁发机构值的分号分隔列表。本章前文已经讨论过，每个内容 URI 都由模式、颁发机构、数据类型路径和实例标识符组成。每个

CONTENT_URI 值的每个颁发机构都应该包含在 android:authorities 列表中。

现在，每当 Android 遇到内容 URI 时，它可以通过清单文件审查注册的提供程序，查找匹配的颁发机构。这将告知 Android 哪些应用程序和类实现了内容提供程序， Android 便可在调用 Activity 和被调用的内容提供程序之间建立起桥梁。

27.4 更改通知支持

内容提供程序的一个可选功能是为客户端提供更改通知支持。这意味着内容提供程序能够让客户端知道用于给定内容 URI 的数据是否发生了更改。

假设你创建了一个基于用户的 feed 订阅（可能通过 OPML）从 Internet 获取 RSS 和 Atom feed 的内容提供程序。该内容提供程序对 feed 内容提供只读访问，同时关注电话上使用这些 feed 的几个应用程序，而不是让每个人都实现自己的 feed 轮询、获取和缓存系统。你还可以实现了一个服务，异步获取这些 feed 的更新，并更新底层数据存储库。内容提供程序可以在 feed 更新时向使用这类 feed 的应用程序发出提醒，于是使用特定 feed 的应用程序便可以刷新并获取最新的数据。

在内容提供程序一方，实现这一点的方式是对 ContentResolver（通过 getContext().get ContentResolver()可从内容提供程序获取）实例调用 notifyChange()。该方法有两个参数：更改的内容片段的 URI，发起更改的 ContentObserver。在很多情况下，后者为 null；非 null 值表示发起更改的观察程序不会得到本身更改的通知。

在内容使用者一方，Activity 对其 ContentResolver（通过 getContentResolver()）调用 registerContentObserver()。这将把 ContentObserver 实例与提供的 URI 连接起来，只要对这个特定的 URI 调用 notifyChange()，观察程序就将得到通知。当使用者使用完该 URI 之后，使用 unregisterContentObserver()释放连接。

第 28 章

请求和要求许可

20 世纪 90 年代末，病毒席卷了整个 Internet。它们从 Microsoft Outlook 中筛选联系人信息，借助电子邮件传播。病毒只是将自身的副本发送给 Outlook 中每一个有电子邮件地址的联系人。这是因为，当时的 Outlook 没有采取任何措施保护数据不受那些使用 Outlook API 的程序的损害，因为该 API 是面向普通开发人员设计的，而并不针对病毒作者。

时至今日，许多保存联系人数据的应用程序，都要求其他程序必须获得用户明确授权才能访问联系人信息，从而确保该数据的安全。该权限可以逐个授予，也可以在安装时一次性授予。

Android 也是如此，也要求应用程序在读取或写入联系人数据时获得许可（permission）。Android 的许可系统不仅保护联系人数据，而且为内容提供程序和那些非 Android 框架提供的服务提供了保护。

作为一名 Android 开发人员，经常要确保应用程序拥有合适的许可，以便对其他应用程序的数据执行操作。那么你也可以选择要求其他应用程序如果你想让其他 Android 组件使用你的数据或服务，必须拥有许可。本章将讨论如何完成这两个任务。

28.1 请求许可

请求使用其他应用程序的数据或服务，需要将 `uses-permission` 元素添加到 AndroidManifest.xml 文件。你的清单文件可以有零个或多个 `uses-permission` 元素，它们都是根 `manifest` 元素的直接子元素。

`uses-permission` 元素有一个特性 `android:name`，它是应用程序所要求的许可名称。

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

所有常用系统许可都以 `android.permission` 开始，并且可以在 `Manifest.permission` 的 Android SDK 文档中查到。第三方的应用程序可以有自己的许可，在这种情况下它们会通过文档

给出说明。下面是一些比较重要的内置许可：

- INTERNET，允许应用程序以任何方式访问 Internet，包括使用原始的 Java 套接字或 WebView 部件；
- READ_CALENDAR、READ_CONTACTS 等，用于从内置内容提供程序读取数据；
- WRITE_CALENDAR、WRITE_CONTACTS 等，用于修改内置内容提供程序中的数据。

在应用程序安装时，用户将会看到提示，如果确认即表明接受应用程序执行许可所要求的操作。因此重要的是，你需要请求尽可能少的许可并尽量使你的请求合理，这样用户才不会因为请求过多且不必要而放弃安装应用程序。注意，该提示在当前的模拟器上不可用。

如果你没有所需的许可，并且试图执行需要许可才能执行的任务，可能收到一个 SecurityException，通知缺少许可，但这并不能保证总会发生。失败可能以其他形式出现，取决于其他程序是否正在捕捉和试图处理异常。注意，只有在你忘记要求许可时，许可检查才会失败；应用程序正在运行并且没有获得所要求的许可，这是完全不可能的。

28.2 声明许可

另一方面就是保护应用程序的安全。如果应用程序仅仅是 Activity 和 Intent 接收器，安全性可能仅仅是一个“出站”相关的事情，意味着请求权限以使用其他应用程序的资源。相反，如果将内容提供程序或服务放入应用程序内，就要实现“入站”的安全性，以控制哪个应用程序可以对数据执行何种操作。

注意，这里的问题并不是其他应用程序是否会弄乱数据，而是用户信息的隐私性或是使用可能产生费用的服务。这也就是内置 Android 应用程序常用许可的重点：是否可以读取或修改联系人信息、发送消息，等等，如果应用程序没有存储可视为隐私的信息，安全性就不是什么问题。相反，如果应用程序存储了隐私数据，例如医疗信息，那么安全性就显得比较重要了。

使用许可保护应用程序安全的第一步，是在 AndroidManifest.xml 文件中再次声明所陈述的许可。在本例中，添加 permission 元素而不是 uses-permission 元素。同样，可以声明零个或多个 permission 元素，所有 permission 元素都是根 manifest 元素的直接子元素。

声明许可比使用许可要复杂一些。需要提供以下 3 方面的信息。

- 许可的符号名称：确保许可与其他应用程序的许可名称不冲突，所以应使用应用程序的 Java 命名空间作为前缀。
- 许可的标签：用户可以理解的、简练的表达。
- 许可的描述：用户可以理解的、稍详细的描述。

```
<permission
  android:name="vnd.tlagency.sekrits.SEE_SEKRITS"
  android:label="@string/see_sekrits_label"
  android:description="@string/see_sekrits_description" />
```

这里没有强制使用许可，相反说明了它可能是一个许可，应用程序必须在违反安全性时对其做出标记。

有两种方式可以让应用程序强制实施许可，声明在何处以及在什么环境下需要许可。可以在代码中强制实施许可，但更容易的方法是需要许可时，在清单文件中说明。

28.2.1　通过清单文件强制实施许可

Activity、服务和 Intent 接收器可以声明一个名为 android:permission 的特性，它的值是访问以下项所必需的许可名称：

```
<activity
  android:name=".SekritApp"
  android:label="Top Sekrit"
  android:permission="vnd.tlagency.sekrits.SEE_SEKRITS">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category
      android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

只有应用程序请求了你指定的许可才能访问受保护的组件。在这种情况下，“访问”意味着：

- 没有许可便无法开始 Activity；
- 没有许可便无法开始、停止服务，也无法将服务绑定到 Activity；
- 如果发送方没有许可，Intent 接收器忽略通过 sendBroadcast()发送的消息。

内容提供程序提供了两个不同的特性：readPermission 和 writePermission。

```
<provider
  android:name=".SekritProvider"
  android:authorities="vnd.tla.sekrits.SekritProvider"
  android:readPermission="vnd.tla.sekrits.SEE_SEKRITS"
  android:writePermission="vnd.tla.sekrits.MOD_SEKRITS" />
```

在本例中，readPermission 控制对内容提供程序的访问查询，writePermission 控制插入、更新或删除内容提供程序中的数据。

28.2.2　在其他地方强制实施许可

在代码中，有两种方式可强制实施许可。

- 服务可以通过 checkCallingPermission() 在每次调用时检查许可。这会返回 PERMISSION_GRANTED 或 PERMISSION_DENIED，取决于调用方是否拥有你指定的许可。例如，如果服务实现单独的读、写方法，你可以在代码中获得 readPermission 和 writePermission 的效果，办法是检查这些方法是否具有你需要的来自 Java 的许可。
- 调用 sendBroadcast()时可以包括一个许可。这意味着合法的接收器必须持有这个许可；没有该许可的接收器是不合法的，不能接收广播。例如，Android 子系统可能在广播它已收到一条短信时包含 RECEIVE_SMS 许可。这样，只有被授权可以接收该短信的 Intent 的接收器才合法。

28.3 别忘了文档

在编译时不能自动发现许可，所有许可失败都在运行时发生。因此，通过文档说明公共 API 要求的许可很重要，包括要从其他 Activity 启动的内容提供程序、服务和 Activity。否则，试图连接应用程序的程序员需要反复试验才能找出许可规则。

此外，应预料到应用程序的用户会收到提示，要求确认应用程序所需要的任何许可。因此，你需要通过文档向用户解释清楚他们会收到什么样的提示，以免他们对手机显示的问题感到迷惑，而选择放弃安装你的应用程序。

第 29 章 创建服务 29

如前所述，Android 服务针对即使脱离其他 Activity 也需要长时间保持运行的进程。例如播放音乐，即便是播放器 Activity 被垃圾回收了也会继续播放；在 Internet 上轮询 RSS/Atom 源更新和维护在线聊天连接，即便聊天客户端因来电话而失去焦点，连接也不会断开。

服务可以手工启动（通过 API 调用），也可在某个 Activity 试图通过 IPC（Inter Process Communication，进程间通信）连接到相应服务时创建。创建之后，服务将一直运行，除非不再需要该服务以及需要回收 RAM，或者由于其自身决定或因为不再有人使用而关闭。长时间运行不是没有代价的，所以要注意服务不应占用太多的 CPU，也不要保留收音机活跃太长时间，以免耗用过多电量。

本章介绍如何创建自己的服务。下一章将介绍如何从 Activity 或其他上下文中使用这样的服务。这两章都将分析 Service/WeatherPlus 示例应用程序。本章主要介绍 WeatherPlusService 实现。WeatherPlusService 扩展了最初 Internet/Weather 示例的天气信息获取逻辑，通过将其绑定到一个监视位置变化的服务，实现随着模拟器的“移动”来更新天气信息。

29.1 通过类创建服务

创建服务与构建 Activity 有许多相同的特征：继承 Android 提供的基类，重写某些生命周期方法，并通过清单文件将服务与系统建立联系。

所以，创建服务的第一步就是扩展 Service 类。在我们的示例中，使用 WeatherPlusService 子类。

正如 Activity 拥有 onCreate()、onResume()、onPause()等方法，Service 的实现也有它们自己的生命周期方法，这些方法如下。

- onCreate()：与 Activity 一样，在以任何方式创建服务进程时调用。
- onStart()：每当通过 startService()启动服务时调用。

- onDestroy()：服务关闭时调用。

例如，下面是 WeatherPlusService 的 onCreate()方法：

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

  client=new DefaultHttpClient();
  format=getString(R.string.url);

  mgr=(LocationManager)getSystemService(Context.LOCATION_SERVICE);
  mgr.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                             10000, 10000.0f, onLocationChange);
}
```

首先，我们向上链接到超类，以便 Android 可以完成任何所需的设置工作。然后我们初始化 HttpClient 组件和格式字符串，与我们在 Weather 示例中所做的一样。然后为应用程序取得 LocationManager 实例，并请求在位置发生变化时通过 GPS LocationProvider 获得更新，第 32 章将详细介绍 LocationProvider。

onDestroy()方法比较简单：

```
@Override
public void onDestroy() {
  super.onDestroy();

  mgr.removeUpdates(onLocationChange);
}
```

此处我们仅关闭了位置监视逻辑，并向上链接到相应的超类，以便 Android 在内部进行必要的跟踪记录。

除了这些生命周期方法，服务还需要实现 onBind()，方法返回一个 IBinder，是 IPC 机制的关键。我们将在下一节详细介绍 onBind()。

29.2 单例

在默认情况下，服务与应用程序的其他组件（如其 Activity）在同一进程中运行。因此，可以在服务对象上调用 API 方法——如果能够取得服务对象的话。理想情况下，应该可以通过某些方式（甚至是类型安全的方法）要求 Android 提供本地服务对象。遗憾的是，在本书编写之时，还没有这样的 API。因此，我们被迫要耍些小伎俩。

任何给定的服务都至多有一个副本在内存中运行。如果服务没有启动，在内存中可能没有副本。但即使多个 Activity 正在尝试使用某个服务，实际上只有一个服务会运行。这是单例模式的完美实现，我们只需暴露单例本身，以便其他组件可以访问该对象。

可以通过一个公有静态数据成员或者一个公有静态 getter 方法暴露这个单例。不过这样会面临一些内存管理的风险。因为从静态上下文引用的所有对象都不受垃圾回收的控制，因此需要非常小心，确保在服务的 onDestroy()中将静态引用设置为 null。否则，在断开与 Android 的连接后，服务将会一直保存在内存中，直到 Android 决定关闭进程。

幸运的是，还有一个替代方法，即使用 onBind()。

绑定允许服务将 API 暴露给绑定到其上的 Activity（或者其他服务）。这种基础设施主要是为了支持远程服务，绑定到的 API 可通过 IPC 访问，所以一个服务可以将其 API 暴露给其他应用程序。然而，这个简单的绑定操作本身在服务和它的客户端都在同一个应用程序中的情况下比较有用，即本地服务的情况。

要通过本地绑定将服务本身暴露给 Activity，首先必须创建一个公有的内部类，它要扩展 android.os.Binder 类：

```
public class LocalBinder extends Binder {
  WeatherPlusService getService() {
    return(WeatherPlusService.this);
  }
}
```

此处的 binder 暴露了一个方法 getService()，它返回服务本身。这个方法对于远程服务没有意义，因为 IPC 不允许在进程之间传递服务。但是，对于本地服务来说，这是一个极为完美的 binder。

接下来，需要在 onBind()方法中返回该 binder 对象：

```
@Override
public IBinder onBind(Intent intent) {
  return(binder);
}
```

此时，任何绑定到服务的客户端都将能够访问服务对象，并且能够调用它的方法。我们将在下一章详细介绍这一内容。

29.3　清单文件的作用

最后，需要将服务添加到 AndroidManifest.xml 文件，以便将其作为一个有效的服务。为此，需要将 service 元素作为 application 元素的子元素添加到文件中，并通过 android:name 来引用服务类。

例如，下面是 WeatherPlus 的 AndroidManifest.xml 文件：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.commonsware.android.service">
  <uses-permission android:name="android.permission.INTERNET" />
```

```
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
  <application android:label="@string/app_name"
    android:icon="@drawable/cw">
    <activity android:name=".WeatherPlus" 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=".WeatherPlusService" />
  </application>
</manifest>
```


既然服务类与应用程序的其他组件位于同一 Java 命名空间中，我们可以使用简化的点符号来引用类（".WeatherPlusService"）。

如果需要启动或绑定到服务的人持有某种许可，可添加 android:permission 特性来指定许可。参见第 28 章了解详细信息。

29.4 事件提醒

有时候，服务需要以异步方式向 Activity 提醒所发生的事件。

例如，WeatherPlusService 实现背后的思路就是在手机（或模拟器）位置改变时服务能获得“反馈”。获得反馈后，服务调用 Web 服务并生成新的天气预报页面供 Activity 显示。然后服务必须让 Activity 得知新的天气预报数据已经准备好了，以便 Activity 可以加载和显示它。

要按上述方式实现各组件的互操作，有两个主要的方法：回调和广播 Intent。

注意，如果服务只需提醒用户某一事件，可以考虑使用通知（参阅第 31 章），因为这是处理该需求的常规方法。

29.4.1 回调

因为 Activity 可以直接使用本地服务，所以 Activity 能为服务提供某种侦听器对象，以便服务在需要时调用。要实现这一点，需要：

(1) 为该侦听器对象定义一个 Java 接口；
(2) 为服务提供一个公共的 API，以注册和撤销侦听器；
(3) 让服务在适当时候使用这些侦听器，通知已注册的侦听器某些事件的发生；
(4) 根据需要让 Activity 注册和撤销侦听器；
(5) 让 Activity 以适当方式响应基于侦听器的事件。

最重要的是确保在完成时让 Activity 撤销侦听器。侦听器对象通常都知道自己的 Activity，

有时候是明确的（通过数据成员），有时候是不明确的（作为内部类实现）。如果服务紧紧抓住无效的侦听器对象不放，相关 Activity 就会始终停留在内存中，尽管 Android 不再使用该 Activity。这会造成很严重的内存泄漏。可以使用 WeakReference、SoftReference 或类似结构确定 Activity 是否被销毁，任何在服务中注册的侦听器都不会将该 Activity 保留在内存中。

29.4.2　广播 Intent

第 17 章中首次提及了一个替代方法，即让服务发送一个可以被 Activity 接收的广播 Intent——假定 Activity 仍在运行并且没有被暂停。我们将在第 30 章从客户端角度讨论这一交换。在这里我们只介绍服务如何发送广播。

数据流的高级实现被封装在 FetchForecastTask 中，这是一个 AsyncTask 实现，让我们能够将 Internet 访问移到后台线程中：

```
class FetchForecastTask extends AsyncTask<Location, Void, Void> {
  @Override
  protected Void doInBackground(Location... locs) {
    Location loc=locs[0];
    String url=String.format(format, loc.getLatitude(),
                             loc.getLongitude());
    HttpGet getMethod=new HttpGet(url);

    try {
      ResponseHandler<String> responseHandler=new BasicResponseHandler();
      String responseBody=client.execute(getMethod, responseHandler);
      String page=generatePage(buildForecasts(responseBody));
      synchronized(this) {
        forecast=page;
      }

      sendBroadcast(broadcast);
    }
    catch (Throwable t) {
      android.util.Log.e("WeatherPlus",
                         "Exception in updateForecast()", t);
    }

    return(null);
  }

  @Override
  protected void onProgressUpdate(Void... unused) {
    // not needed here
  }

  @Override
  protected void onPostExecute(Void unused) {
    // not needed here
  }
}
```

这与最初 Weather 演示程序中相应的代码片段大致相同。它执行了 HTTP 请求，将其转换成

一组 Forcast 对象，并将这些对象放到一个网页中。除了引入 AsyncTask 之外，第一个差别是网页仅缓存在服务内，因为服务不能直接将页面放入 Activity 的 WebView 中。第二个差别就是调用 sendBroadcast()，由它接收 Intent 并将其发送给所有感兴趣的相关方。该 Intent 在类的前面首先声明。

```
private Intent broadcast=new Intent(BROADCAST_ACTION);
```

此处的 BROADCAST_ACTION 只是一个静态的 String，它拥有一个将此 Intent 与其他 Intent 区分开的值：

```
public static final String BROADCAST_ACTION=
          "com.commonsware.android.service.ForecastUpdateEvent";
```

29.5 远程服务与其他代码

在 Android 中，服务可以是本地的，也可以是远程的。本地服务与启动服务的 Activity 在同一个进程中运行。远程服务在自己的进程中运行。*The Busy Coder's Guide to Advanced Android Development*（CommonsWare，2009）中对远程服务进行了详细介绍。

我们在第 32 章会再次讨论这一服务，届时将讲述如何跟踪位置（这一章只讲了个开头而已）。

第 30 章

调用服务 30

服务可以被任何能够运行一段时间的应用程序组件使用。这包括 Activity、内容提供程序以及其他服务。显然，这不包括纯广播接收器（例如不是 Activity 一部分的 Intent 接收器），因为在每个实例处理一个进入的 Intent 之后，纯广播接收器都会被立即回收。

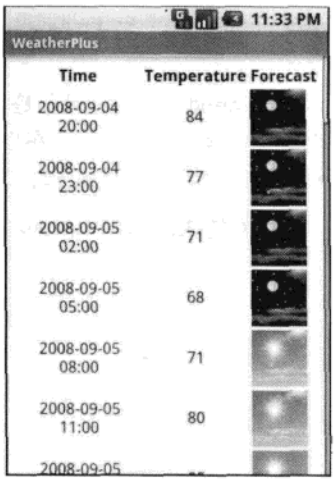


图 30-1 WeatherPlus 服务客户端

要使用本地服务，需要启动该服务，访问服务的对象，然后在该服务上调用方法。使用完成后，可以停止该服务，或者让服务自行停止。在本章中，我们将从客户角度了解 Service/WeatherPlus 示例应用程序。WeatherPlus 活动非常像最初的 Weather 应用程序。它就是一个显示天气预报信息的 Web 页面，如图 30-1 所示。

30.1 联系的纽带

启动服务的一个方法是调用 startService()，传入说明启动哪个服务的 Intent（同样，最简单的方式可能是指定服务类，前提是它是你自己的服务）。反之，要停止一个通过 startService()启动的服务，请使用在相关 startService()调用中所用的 Intent 来调用 stopService()。

服务启动后，需要与之通信。所有必要的通信都可以借助 Intent 内封装的额外功能进行。或者，如果它是一个提供单例的本地服务，可以引用那个单例。

然而，如果如上一章所述实现了 onBind()，还可以通过一种不同的方法获得服务：bindService()。

当 Activity 绑定到服务时，它主要是请求通过服务连接器（binder）访问由服务所公开的公

共 API，服务的 onBind()方法会返回这种连接器。与此同时，Activity 也可以通过 BIND_AUTO_CREATE 标志说明，如果该服务没有运行，就让 Android 自动启动该服务。

要使用这一技巧处理 WeatherPlus 和 WeatherPlusService 类，首先需要从 onCreate()调用 bindService()：

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

  browser=(WebView)findViewById(R.id.webkit);
  bindService(new Intent(this, WeatherPlusService.class),
              onService, Context.BIND_AUTO_CREATE);
}
```

此 bindService()调用引用一个 onService 回调对象，此对象是 ServiceConnection 的一个实例：

```
private ServiceConnection onService=new ServiceConnection() {
  public void onServiceConnected(ComponentName className,
                                 IBinder rawBinder) {
    appService=((WeatherPlusService.LocalBinder)rawBinder).getService();
  }

  public void onServiceDisconnected(ComponentName className) {
    appService=null;
  }
};
```

只要 WeatherPlusService 启动并开始运行，就会使用 onServiceConnected()调用 onService 对象。我们获得了 IBinder 对象，它是一个代表服务的不透明句柄。我们可以使用它获取由 WeatherPlusService 公开的 LocalBinder，进而获取实际的 WeatherPlusService 对象（它是被作为一个私有数据成员持有的）：

```
private WeatherPlusService appService=null;
```

然后，可以调用 WeatherPlusService 上的方法，例如在需要时通过调用获得天气预报信息页面：

```
private void updateForecast() {
  try {
    String page=appService.getForecastPage();

    browser.loadDataWithBaseURL(null, page, "text/html",
                                "UTF-8", null);
  }
  catch (final Throwable t) {
    goBlooey(t);

  }
}
```

还需要从 onDestroy()调用 unbindService()，以释放与 WeatherPlusService 的绑定：

```
@Override
public void onDestroy() {
  super.onDestroy();

  unbindService(onService);
}
```

如果没有其他客户端绑定到该服务，Android 将关闭服务，释放它所占用的内存。因此，无需亲自调用 stopService()，因为作为解除绑定的副作用，在需要时 Android 会处理它。

与仅仅使用服务对象的公共静态单例相比，此方法的代码比较长。但是，这个方法不太可能导致内存泄漏。

总之：

- 要让一个服务开始运行，请通过 BIND_AUTO_CREATE 使用 bindService()（如果希望通过绑定机制进行通信），或者使用 startService()。
- 要让一个服务停止运行，执行与启动服务相反的操作：unbindService()或 stopService()。

另一个停止服务的可能方法是，让服务对其自身调用 stopSelf()，如果使用 startService()让服务开始运行并在后台线程上执行一些操作，可以实现这一点，这样服务将会在后台工作完成时自行停止。

30.2　接收广播内容

上一章介绍了服务如何通过发送广播，让 WeatherPlus 活动知道系统根据移动距离对天气预报内容进行了变更。现在，我们介绍一下 Activity 如何接收并使用该广播内容。

下面是 WeatherPlus 的 onResume()和 onPause()实现：

```
@Override
public void onResume() {
  super.onResume();

  registerReceiver(receiver,
        new IntentFilter(WeatherPlusService.BROADCAST_ACTION));
}

@Override
public void onPause() {
  super.onPause();

  unregisterReceiver(receiver);
}
```

在 onResume()中我们注册了一个静态的 BroadcastReceiver，负责接收与服务所声明的操作相匹配的 Intent。在 onPause()中，我们禁用了 BroadcastReceiver，因为暂停的时候将不会接收任何这样的 Intent。

然后，BroadcastReceiver 只是安排对天气预报内容进行更新：

```
private BroadcastReceiver receiver=new BroadcastReceiver() {
  public void onReceive(Context context, Intent intent) {
    updateForecast();
  }
};
```

第31章 利用通知提醒用户

弹出式消息、带有相关消息提示框的托盘图标、回弹停靠图标等——毫无疑问，你已经习惯了这些有时出于正当的理由想方设法来引起你注意的程序。手机不只在收到来电的时候发出声音，在电池电量不足、闹钟、约会通知、收到文本消息时都可以发出声音。

毫无疑问的是，Android 以一个完整的框架来处理这种提醒，将其统称为通知，本章将详细介绍这个主题。

31.1 发布通知的类型

在后台运行的服务，需要用一种方式让用户知道他所感兴趣的事情已经发生，比如电子邮件已经收到。此外，服务需要以某种方式将用户引导至可以对这些事件进行处理的 Activity，如阅读已收到的消息。在这方面，Android 提供了状态栏图标、闪灯以及其他指示器，这些统称为“通知”。

目前的手机上可能已经有这样的图标，这些图标能够显示电池的剩余电量、信号强度、蓝牙是否已启用等。有了 Android，应用程序可添加自己的状态栏图标，且仅在需要时显示（例如，收到一条短信）。

在 Android 中，可以通过 NotificationManager 发出通知。NotificationManager 是一个系统服务。要使用它，需要通过 getSystemService(NOTIFICATION_SERVICE)从你的 Activity 中获得这个服务对象。

NotificationManager 提供了 3 个方法：一个用于发布通知（notify()），另外两个用于停止通知的发布（cancel()和 cancelAll()）。

notify()方法接受 Notification，Notification 是一种数据结构，清楚地说明了应该采用的通知发布形式。以下各节说明了 Notification 对象的功能。

31.1.1 硬件通知

可以通过将 lights 设为 true，让手机上的 LED 闪烁，还可同时指定颜色（例如，设置 ledARGB 中的#ARGB 值）和灯闪烁的模式（使用 ledOnMS 和 ledOffMS 设置 LED 关/开持续时间的毫秒数）。但是请注意，Android 手机会尽最大努力满足颜色需要，也就是说不同的手机将给出不同的颜色，或者根本不能控制颜色。例如，据报道摩托罗拉 CLIQ 仅有一个白色的 LED，所以你可以请求使用任何颜色，但得到的仍然是白色。

可通过 ContentManager(sound) 使用一个指向保存的内容片段的 URI 来播放声音。可将它作为应用程序的“铃声”。

还可以振动手机，通过 long[]说明振动的开/关模式（以 ms 为单位）来进行控制。可以将它设置成默认值，或者将其作为可选项，以便在环境需要比铃声更微妙的通知时，供用户选择。要使用这一方法，需要请求 VIBRATE 许可（第 28 章介绍了有关许可的更多信息）。

31.1.2 图标

闪烁的 LED、声音和振动的目的是让用户查看手机，图标的设计目的是指导他们进行下一步操作，告诉他们重要的事情是什么。

要建立一个用于 Notification 的图标，需要设置两个公共字段：icon，提供了代表图标的 Drawable 资源的标识符；contentIntent，在单击图标时提供可供发出的 PendingIntent。你应该确保 PendingIntent 会被某些方面所捕获（可以是你自己的应用程序代码），以便采取合适的步骤让用户处理正在触发通知的事件。也可以在图标出现在状态栏中时显示文本简介(tickerText)。

如果这 3 个都想要，较简单的方法就是调用 setLatestEventInfo()，它将图标、contentIntent 和 tickerText 包装在一个调用中。

31.2 查看运行中的通知发布

现在让我们看一下 Notifications/Notify1 示例项目，特别是 NotifyDemo 类：

```
package com.commonsware.android.notify;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import java.util.Timer;
import java.util.TimerTask;

public class NotifyDemo extends Activity {
  private static final int NOTIFY_ME_ID=1337;
  private Timer timer=new Timer();
  private int count=0;

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

    Button btn=(Button)findViewById(R.id.notify);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        TimerTask task=new TimerTask() {
          public void run() {
            notifyMe();
          }
        };

        timer.schedule(task, 5000);
      }
    });

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

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        NotificationManager mgr=
          (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        mgr.cancel(NOTIFY_ME_ID);
      }
    });
  }

  private void notifyMe() {
    final NotificationManager mgr=
      (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    Notification note=new Notification(R.drawable.red_ball,
                                       "Status message!",
                                       System.currentTimeMillis());
    PendingIntent i=PendingIntent.getActivity(this, 0,
                            new Intent(this, NotifyMessage.class),
                                               0);
     note.setLatestEventInfo(this, "Notification Title",
                             "This is the notification message", i);
     note.number=++count;

     mgr.notify(NOTIFY_ME_ID, note);
   }
}
```

如图31-1所示，该Activity使用了两个大按钮：一个是在5s之后发出通知，另一个是取消该通知（如果激活它）。

在 notifyMe()中创建通知需要 6 个步骤：

(1) 访问 NotificationManager 实例；

(2) 使用图标（红球）创建 Notification 对象，在发出通知时以及与此事件相关的时间内，在状态栏上将闪动一条消息；

(3) 创建一个 PendingIntent，触发对另一个 Activity（NotifyMessage）的显示；

(4) 使用 setLatestEventInfo()指定在单击通知时，我们将显示一个特定的标题和消息，如果单击标题和消息，将启动 PendingIntent；

(5) 更新与通知有关的数字；

(6) 告知 NotificationManager 显示通知。

因此，如果我们单击顶部的按钮，5s 之后，红球图标将出现在状态栏中，同时出现的还有简短的状态信息，如图 31-2 所示。红球将会让我们的数字（初始值是 1）叠加在右下角（例如，可以用它来表示未读消息的数量）。

如果单击红球，状态栏下面将会出现一个抽屉。拉开抽屉直至屏幕底部，查看醒目的通知，包括我们自己的通知，如图 31-3 所示。

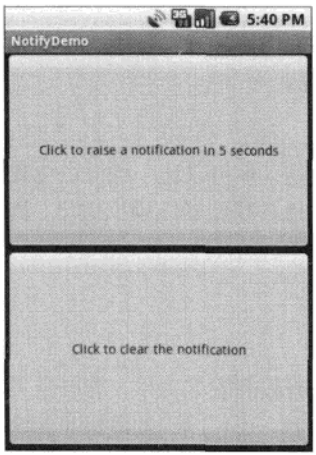


图 31-1 NotifyDemo 活动主视图

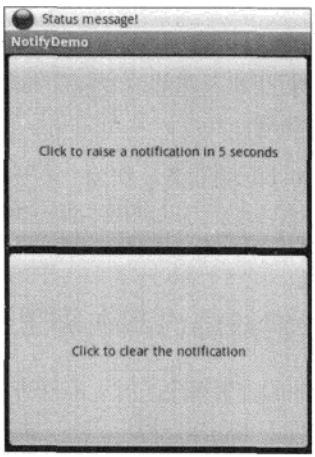


图 31-2 显示在状态栏中的通知，带有状态消息

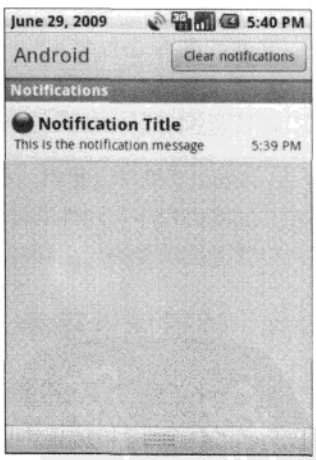


图 31-3 完全展开的通知抽屉，带有我们的通知

如果单击抽屉中的通知条目，会被引导至一个无关紧要的的 Activity，它显示了一条消息。在实际应用程序中，这个 Activity 将根据所发生的事件做些有用的事情（例如，让用户注意有新到的邮件）。

单击按钮清除通知，红球将从状态栏消失。

第32章 访问基于位置的服务

当前手机上一个比较流行的功能是GPS功能，有了它，无论你在何处，手机都可以及时告知你所处的位置。虽然GPS服务最常见的用途是提供地图和确定方向，但是如果知道所处的位置，还可以实现其他功能。例如，可以根据物理位置建立一个动态聊天应用程序，与距离最近的人员聊天，还可以向Twitter或者类似服务自动发送带地理位置标签的帖子。

GPS不是唯一的手机定位方式，还有其他替代方式：

- 在欧洲与GPS等同的产品叫做Galileo，在本书出版之时，它仍处在开发阶段；
- 手机基站三角测量，根据附近手机基站的信号强度确定位置；
- 靠近拥有已知位置的Wi-Fi热点。

Android手机可以提供一个或多个此类服务。作为一名开发人员，你可以让手机提供位置信息和有关提供程序可用性的详细信息，甚至可以利用本章提供的方法在模拟器上模拟定位，测试支持位置提供功能的应用程序。

32.1 位置提供程序：它们知道你藏在哪里

Android手机可以使用多个不同的位置确定方式。有些方式具有较高的准确性，有些是免费的，有些则收取相关费用。有些定位方式不仅仅可以告知当前的位置，还可以告知海拔高度或者当前速度等信息。

Android将这些方式全部抽象到一组LocationProvider对象中。Android环境通常拥有零个或多个LocationProvider实例：手机上可用的每个定位服务都有一个。提供程序不仅知道你的位置，而且知道其自身准确性、费用等特点。

开发人员将使用LocationManager，它保存了LocationProvider集，能够指出哪个LocationProvider适合你的特定情况。应用程序的许可也是必需的，否则各种定位API将会因为违反安全性而不能使用。根据要使用的位置提供程序，可能需要ACCESS_COARSE_LOCATION、ACCESS_FINE_

LOCATION，或者两者都需要。（许可在第 28 章中介绍。）

32.2 自我定位

显然，位置服务最主要的功能就是找到自己目前所处的位置。

要确定目前所处的位置，首先要获得 LocationManager，这需从 Activity 或服务中调用 getSystemService(LOCATION_SERVICE)，将其类型强制转换为 LocationManager。

下一步是获得要使用的 LocationProvider 的名称。此处你主要有两个选择：

- 让用户挑选一个提供程序；
- 根据一套标准确定最合适的提供程序。

如果希望由用户挑选提供程序，在 LocationManager 上调用 getProviders()将获得一个提供程序 List，可以将它提交给用户以供选择。

另一种方式是，创建并填充一个 Criteria 对象，说明 LocationProvider 要提供哪些特殊内容。例如，可以指定下列标准：

- setAltitudeRequired()：说明是否需要当前的高度信息；
- setAccuracy()：说明位置的最低精确度（以 m 为单位）；
- setCostAllowed()：控制提供程序是否必须免费，还是可以由手机使用者支付费用。

假设填好了一个 Critieria 对象，在 LocationManager 上调用 getBestProvider()。Android 将会根据标准进行筛选，并提供最佳答案。注意，并不是所有的标准都要满足。如果任何情况都不匹配，那么除了成本标准以外，其他标准都可放松。

欢迎硬连线一个 LocationProvider 名称（例如 GPS_PROVIDER），或许仅出于测试目的而这样做。

知道 LocationProvider 的名称以后，可调用 getLastKnownPosition()找出近期所处的位置，注意“近期”也许已经过期（例如，电话关掉）或者甚至是 null（如果没有为提供程序提供位置记录）。调用 getLastKnownPosition()不会有费用或电量方面的开销，因为不需要激活提供程序就能获得这个值。

这个方法返回一个 Location 对象，该对象以 Java 的 double 型数据提供手机的纬度和经度。如果特定的位置提供程序还提供其他数据，也可以获得这些数据：

- 高度，hasAltitude()将会告知是否有高度值，getAltitude()将以 m 为单位返回高度值；
- 方位（即指南针风格的方向），hasBearing()将会告知是否有可用的方位，而 getBearing 将会返回以正北稍偏东为准的度数；

- 速度，hasSpeed()将会告知速度是否已知，getSpeed()将会以 m/s 为单位返回速度值。

从一个 LocationProvider 获取 Location 的更合理方法是注册更新（如 32.3 节所述）。

32.3　移动

并非所有位置提供程序都必须是立即响应的。例如，在获得位置信息之前，GPS 要求激活无线电并获得卫星定位。这就是 Android 不提供 getMeMyCurrentLocationNow()方法的原因。此外，用户可能希望在应用程序中反映他们的移动，而你很可能非常需要关闭位置更新注册，并用它来获取当前的位置信息。

Weather 和 WeatherPlus 示例应用程序展示了如何注册更新：在 LocationManager 实例上调用 requestLocationUpdates()。这个方法有 4 个参数：

- 要使用的位置提供程序的名称；
- 获得位置更新所需的毫秒数；
- 获得位置更新前手机必须移动的米数；
- 可以获知关键位置相关事件的 LocationListener。

下面是一个 LocationListener 示例：

```
LocationListener onLocationChange=new LocationListener() {
  public void onLocationChanged(Location location) {
    updateForecast(location);
  }
  public void onProviderDisabled(String provider) {
    // required for interface, not used
  }

  public void onProviderEnabled(String provider) {
    // required for interface, not used
  }

  public void onStatusChanged(String provider, int status,
                              Bundle extras) {
    // required for interface, not used
  }
};
```

这里我们只需利用提供给 onLocationChanged()回调方法的 Location 来调用 updateForecast()。如第 29 章所示，updateForecast()实现利用当前位置处的天气预报信息构建网页，并发送广播让 Activity 知道更新可用。

当不再需要更新时，使用已注册的 LocationListener 调用 removeUpdates()。如果没有这样做，应用程序将会继续接收位置更新，甚至在所有 Activity 关闭之后仍然如此，这样会让 Android 无法收回应用程序所使用的内存。

32.4 我们到了吗

有时，你对现在所处的位置并不关心，或者在移动时也是如此，但是，你想知道何时能够到达目标地点。它可能是终点，也可能是一系列方向中的下一个地点，为用户提供下一步指导的起点。

为了完成这一任务，LocationManager 提供了 addProximityAlert()。这就注册了一个 PendingIntent，将在手机进入距离特定位置一定范围之内时被激活。addProximityAlert()方法的参数如下：

- 目标地点的经度和纬度；
- 一个半径值，指出距离目标地点多近后才会发出 Intent；
- 注册的有效期（单位ms，超出这段时间后注册自动失效）；值为-1 意味着注册有效期将一直持续，直到你用removeProximityAlert()手动删除它。
- 在手机进入目标范围内（以位置和半径来表示）时发出的 PendingIntent。

注意，这并不能保证你能实际收到 Intent。这可能因为位置服务出现中断，也可能是因为在接近提醒处于活动状态的时间内手机不在目标范围内。例如，如果目标地点远了一点儿，半径又比较严格的话，手机可能就在目标区域的周围，或者通过目标区域的速度太快，以至于在那段时间内，没有对手机位置进行取样。

由你安排 Activity 或者 Intent 接收器来响应你注册接近提醒的 Intent。Intent 到来的时候，能做的事情取决于你。例如，可以设置一个通知（例如，振动手机），将信息记录到内容提供程序，或者向 Web 站点发布一条消息。

注意，每当对位置进行取样且你处于目标范围之内（不是正在进入目标范围时），都会收到 Intent。因此，你会多次收到 Intent——或许次数还不少，这取决于目标范围的大小和手机移动的速度。

32.5 测试

Android 模拟器没有能力获得 GPS 定位信息，也不能通过手机基站三角测量来获得位置，更不能通过附近的 Wi-Fi 信号确定位置。因此，如果想要模拟移动中的手机，需使用一些方法向模拟器提供模拟的位置数据。

不管是出于什么原因，这个特定的区域随着 Android 自身的发展变化也发生了显著的变化。过去常常是可以在应用程序内部提供模拟的位置数据，非常便于展示，但这些选项在 Android 1.0 中删除了。

一个可能实现模拟的位置数据供应的选项是 DDMS（Dalvik Debug Monitor Service）。这是一个与模拟器分开的外部程序，它可以用几种不同的格式向模拟器提供单一位置点，或者提供要行驶的整个路线。DDMS 将在第 35 章中详细介绍。

第33章

使用MapView和MapActivity显示地图

除搜索服务外，Google 最受欢迎的一个服务当然是 Google 地图，使用它可以查找所有地点信息，包括距离最近的比萨饼店、从纽约市去旧金山的路线（有 2 905 英里!），应有尽有，还提供街景视图和卫星影像。

毫无疑问，大多数 Android 手机都整合了 Google 视图。对于这些手机，用户可以直接从主 Android 启动程序获得可用的地图 Activity。作为一名开发人员，更加相关的是 MapView 和 MapActivity，它们支持将地图整合到应用程序中。它们不仅可以显示地图、控制缩放级别、允许人们平移地图，还可以连接 Android 基于位置的服务（参见第 32 章），以显示手机的位置和行驶方向。

幸运的是，将基本地图功能整合到 Android 项目中非常容易。如果有其他的想法，开发人员还能实现一些更好的功能。

33.1 条款无情

将 Google 地图整合到应用程序前，需要同意大量的法律条款。这些条款中有些可能令人不快。

如果正在考虑整合 Google 地图，请认真核对这些条款，确定其使用不会与某些条款相抵触。强烈建议就是否有条款抵触问题咨询法律专业人士。

同时，多关注基于其他地图数据源的地图选项，如 OpenStreetMap（http://www.openstreetmap.org/）。

33.2 添加项问题

到 Android 1.5 为止，Google 地图不再是 Android SDK 不可分割的一部分。相反，它是 Google

API 添加项的一部分，是常用 SDK 的扩展。Android 添加项系统为其他子系统提供挂钩，这些子系统可以是某些特定设备的一部分。

> **说明** Google地图不是Android开源项目的组成部分，毫无疑问，某些手机会因为许可问题而缺少Google地图。例如，在本书撰写之时，Archos 5 Android平板电脑上就没有Google地图。

总之，Google 地图属于添加项这一事实不会影响日常开发。然而，请牢记以下几点：

- 创建项目的目标要合适，以确保 Google 地图 API 可用；
- 要测试 Google 地图整合，还需要一个支持 Google 地图 API 的 AVD。

33.3 基本要素

将地图整合到应用程序中时，最简便的方式是创建自己的 MapActivity 子类。与 ListActivity 类似（它将某些智能包装在后端，使 Activity 由 ListView 主导），MapActivity 处理设置由 MapView 主导的活动的微妙之处。

在 MapActivity 子类的布局中，需要添加某个元素，在本书撰写时，这个元素是 com.google.android.maps.MapView。这是一种“速记”方式，通过包含完整的包名和类名得出的部件类名称。这是必要的，因为在 com.google.android.widget 命名空间中没有 MapView。可以给予 MapView 部件任何 android:id 特性值，还可以处理所有布局细节，以便 MapView 与其他部件一起恰当呈现。

但是，必须要有这两项：

- android:apiKey，在产品中需要用作 Google 地图 API 密钥；
- android:clickable = "true"，让用户能够单击并在地图中平移。

例如，Maps/NooYawk 示例应用程序的主要布局如下：

```
<?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">
  <com.google.android.maps.MapView android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="<YOUR_API_KEY>"
    android:clickable="true" />
</RelativeLayout>
```

这个神秘的 apiKey 将在 33.8 节中介绍。

此外，在 AndroidManifest.xml 文件中还需要添加：

- INTERNET 和 ACCESS_COARSE_LOCATION 许可（后者与 MyLocationOverlay 类一同使用，本章

后面将有介绍）；

- 在<application>内，带有 android:name = "com.google.android.maps"的<uses-library>元素，说明正在使用一个可选的 Android API。

NooYawk 的 AndroidManifest.xml 文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.maps">
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

  <application android:label="@string/app_name"
    android:icon="@drawable/cw">
    <uses-library android:name="com.google.android.maps" />
    <activity android:name=".NooYawk" 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>
```

所有初学者所需的几乎全在这里，此外就是从 MapActivity 将 Activity 子类化。无需再进行其他操作，构建此项目之后将其放入模拟器，就可以获得精美的世界地图。但是注意，MapActivity 是抽象的。需要实现 isRouteDisplayed()来说明是否提供某种行驶路线。

理论上，用户可以使用 D-pad 在地图中平移。但是当整个世界都在他们手中时，这不太有用。

由于世界地图本身并不完美，我们接下来添加一些额外功能。

33.4　练习控制

与任何其他部件一样，可通过 findViewById()查找 MapView 部件。部件本身提供了 getMapController()方法。在 MapView 和 MapController 之间，可以自由决定地图显示什么、如何显示。下面介绍缩放和居中这两个最常用的功能。

33.4.1　缩放

一开始显现的世界地图非常宽泛。通常，人们在手机上查看地图，是想看到更小的范围，比如几个街区。

可以通过 MapController 上的 setZoom()方法直接控制缩放级别。它使用整数参数代表缩放级别，1 是世界视图，21 是所能提供的最小级别。每个级别之间的有效分辨率加倍：级别 1 中赤道宽度为 256 像素，级别 21 中赤道宽度为 268 435 456 像素。因为手机显示屏在任何一个维度上

都不可能达到 268 435 456 像素，所以用户只能查看地球上某个角落的地图。级别 16 在各维度上可显示几个街区，因此可能是实际试验合理的起点。

如果希望用户能改变缩放级别，请调用 setBuiltInZoomControls(true)，这样用户可以通过地图底部中央的缩放控件缩小和放大地图。

33.4.2 居中

通常情况下，除了地图缩放，还需要控制地图显示的内容，例如用户当前位置，或在 Activity 中保留了某些数据的位置。要想改变地图中的位置，可在 MapController 上调用 setCenter()。

setCenter()方法以 GeoPoint 为参数。GeoPoint 用经度和纬度表示地点。需要注意的是，GeoPoint 以整数型数据存储经度和纬度，表示实际经度和纬度乘以 1E6。与保存单精度浮点型或双精度浮点数据相比，这节省了一定的内存，并且大大加速了某些必要的 Android 内部计算，以将 GeoPoint 转换为地图上的位置。但是，这意味着你必须记得将实际世界的经度和纬度乘以 1E6。

33.5 地形起伏

在全尺寸计算机上使用的 Google 地图服务可以显示卫星影像，Android 地图也可以。

MapView 提供了 toggleSatellite()，顾名思义，它用来启动或关闭卫星视图。可以让用户通过一个选项菜单触发它，或者在 NooYawk 中通过按下某些键来实现。

```
@Override
 public boolean onKeyDown(int keyCode, KeyEvent event) {
   if (keyCode == KeyEvent.KEYCODE_S) {
     map.setSatellite(!map.isSatellite());
    return(true);
  }
  else if (keyCode == KeyEvent.KEYCODE_Z) {
    map.displayZoomControls(true);
    return(true);
  }

  return(super.onKeyDown(keyCode, event));
}
```

33.6 层上加层

如果曾使用过全尺寸版本的 Google 地图，可能习惯于看到地图上覆盖其他内容，比如指示搜索地点附近商铺的图钉。在地图用语中（以及在许多严谨的图形编辑器中），图钉位于地图上方的单独一层中，你看到的是图钉层与其下的地图层两者的综合视图。

Android 的地图绘制功能也允许创建层，因此可以基于用户输入和应用程序的目的在地图上

做标记。例如，NooYawk 就利用层来显示所选建筑物在曼哈顿岛上的位置。

33.6.1　Overlay 类

要向地图添加任何叠加内容，需要将其作为 Overlay 的子类实现。如果要添加图钉等内容层，可以使用 ItemizedOverlay 子类，ItemizedOverlay 可将此过程简化。

要向地图添加 overlay 类，请在 MapView 上调用 getOverlays()并使用 add()将 Overlay 实例添加到 MapView 上，如下面使用的自定义 SitesOverlay 所示：

```
marker.setBounds(0, 0, marker.getIntrinsicWidth(),
                       marker.getIntrinsicHeight());

map.getOverlays().add(new SitesOverlay(marker));
```

我们将在下一节详细介绍 marker。

33.6.2　绘制 ItemizedOverlay

顾名思义，ItemizedOverlay 允许你提供有意在地图上显示的点列表，具体地说是 OverlayItem 实例。此叠加层处理大部分绘图逻辑。实现此操作的最少步骤如下：

(1) 覆写 ItemizedOverlay<OverlayItem>作为自己的子类（此示例中为 SitesOverlay）；

(2) 在构造函数中构建 OverlayItem 实例的名单，并在它们准备好可用于叠加时调用 populate()；

(3) 实现 size()以返回待叠加层处理的条目数目；

(4) 覆写 createItem()以返回给定索引的 OverlayItem 实例；

(5) 在实例化 ItemizedOverlay 子类时，向其提供一个代表默认图标（如图钉）的 Drawable，以显示每一项。

来自于 NooYawk 构造函数的 marker 是第(5)步使用的 Drawable。它显示一个图钉。

你可能还希望覆写 draw()，以更好地处理 marker 的阴影。虽然地图能够处理投射阴影，但似乎你要为它提供一些帮助，告诉它图标底部的位置，这样才能画出合适的阴影。

例如，SitesOverlay 如下：

```
private class SitesOverlay extends ItemizedOverlay<OverlayItem> {
  private List<OverlayItem> items=new ArrayList<OverlayItem>();
  private Drawable marker=null;

  public SitesOverlay(Drawable marker) {
    super(marker);
    this.marker=marker;
```

```
    items.add(new OverlayItem(getPoint(40.748963847316034,
                                       -73.96807193756104),
                              "UN", "United Nations"));
    items.add(new OverlayItem(getPoint(40.76866299974387,
                                       -73.98268461227417),
                              "Lincoln Center",
                              "Home of Jazz at Lincoln Center"));
    items.add(new OverlayItem(getPoint(40.765136435316755,
                                       -73.97989511489868),
                              "Carnegie Hall",
            "Where you go with practice, practice, practice"));
    items.add(new OverlayItem(getPoint(40.70686417491799,
                                       -74.01572942733765),
                              "The Downtown Club",
                    "Original home of the Heisman Trophy"));

    populate();
  }

  @Override
  protected OverlayItem createItem(int i) {
    return(items.get(i));
  }

  @Override
  public void draw(Canvas canvas, MapView mapView,
                   boolean shadow) {
    super.draw(canvas, mapView, shadow);
    boundCenterBottom(marker);
  }

  @Override
  protected boolean onTap(int i) {
    Toast.makeText(NooYawk.this,
                   items.get(i).getSnippet(),
                   Toast.LENGTH_SHORT).show();

    return(true);
  }

  @Override
  public int size() {
    return(items.size());
  }
}
```

33.6.3 处理屏幕单击

Overlay 的子类还可以实现 onTap()，从而在用户单击地图时获得信息，以便调整所画的内容。例如，在全尺寸 Google 地图中，单击图钉会弹出一个标记地点周围商铺信息的泡泡。使用 onTap() 在 Android 中也可以完成这个工作。

用于 ItemizedOverlay 的 onTap()方法接收所单击的 OverlayItem 的索引。由你决定如何处理该事件。

如上一节所示，在 SitesOverlay 事件中，onTap()如下面所示：

```
@Override
protected boolean onTap(int i) {
  Toast.makeText(NooYawk.this,
                 items.get(i).getSnippet(),
                 Toast.LENGTH_SHORT).show();

  return(true);
}
```

此处我们用 OverlayItem 中的片段发出短整型 Toast，返回 true 表明我们处理了单击操作。

33.7　MyLocationOverlay

Android 拥有内置的 overlay，可处理两个常见的场景：

- 基于 GPS 或其他位置提供逻辑，在地图上显示位置；
- 基于内置指南针传感器（如果有），显示你指向的地点。

只需创建一个 MyLocationOverlay 实例，将其添加到 MapView 的 overlay 列表中，并在合适的时间启用和禁用所需功能即可。

“在合适的时间”目的是最大限度增加电池使用时间。Activity 停止后，更新位置或方向就没有意义了，因此建议在 onResume()中启用这些功能，并在 onPause()中禁用它们。

例如，NooYawk 将使用 MyLocationOverlay 显示指南针。为了实现这一操作，首先需要创建 overlay 并将其添加到 overlay 列表中：

```
me=new MyLocationOverlay(this, map);
map.getOverlays().add(me);
```

然后我们适时启用或禁用指南针：

```
@Override
public void onResume() {
  super.onResume();

  me.enableCompass();
}

@Override
public void onPause() {
  super.onPause();

  me.disableCompass();
}
```

33.8 关键所在

实际上，如果下载本书的源代码，编译 NooYawk 项目，并将其安装到模拟器上运行，可能会看到带有网格的绿屏和几个图钉，但是并非实际的地图。

那是因为源代码中的 API 密钥在你的开发计算机上无效。因此，你需要生成自己的 API 密钥，供应用程序使用。

可在 Android 网站找到为开发和产品应用生成 API 密钥的完整说明（http://code.google.com/android/add-ons/googleapis/mapkey.html）。为了简明起见，让我们集中精力研究在模拟器上运行 NooYawk。这需要下列步骤：

(1) 访问 API 密钥签署页面并检查服务条款；
(2) 重新阅读服务条款，确定你同意这些条款；
(3) 查找证书的 MD5 摘要，用于签署调试模式应用程序；
(4) 在 API 密钥签署页面上，粘贴 MD5 签名并提交表单；
(5) 在结果页面中，复制 API 密钥并在使用 MapView 的布局中将其粘贴为 apiKey 值。

最需要技巧的部分是查找用于签署调试模式应用程序的证书 MD5 签名。实际上，复杂性大多仅限于概念的解释。

所有 Android 应用程序都使用由证书生成的数字签名进行签署。当你设置 SDK 时，系统自动给出调试证书，系统有独立的过程创建自签署证书，供产品应用程序使用。签名过程包括使用 Java keytool 和 jarsigner 工具。要获得 API 密钥，你只需关注 keytool。

要获得调试证书的 MD5 摘要，如果运行在 Mac OS X 或 Linux 上，可使用下面的命令：

```
keytool -list -alias androiddebugkey -keystore ~/.android/debug.keystore -storepass↩
 android -keypass android
```

在其他开发平台上，需要使用平台和用户账户的位置替换-keystore 开关的值：

- 在 Windows XP 上，使用 C:\Documents and Settings\<user>\.android\debug. keystore；
- 在 Windows Vista/Windows7 上，使用 C:\Users\<user>\.android\debug.keystore（其中<user>是账户名）。

第二行输出包含 MD5 摘要，是一系列由冒号分隔的十六进制数字对。

第34章 呼叫处理

在大部分情况下，Android设备都是指手机。这样一来，不仅可使用户使用Android拨打和接听电话，而且如果你愿意，还有机会为他们拨打电话尽可能地提供帮助。

这样做可能的原因如下所示。

- 或许你正在编写一个销售管理应用程序（模仿Salesforce.com）的Android界面，希望用户单击一个按钮即可进行呼叫，而无需在应用程序或手机联系人应用程序中保存相关联系人。
- 或许你正在编写社交网络应用程序，由于可使用的电话号码列表经常发生变化，你希望用户直接从应用程序进行呼叫，而不必将手机的联系人数据库与社交网络联系人同步。
- 或许你正在创建现有联系人系统的替代界面，针对的可能是行动不便的用户（如老年人），使用较大的按钮等方法让他们也可轻松拨打电话。

无论原因是什么，你都可以像使用其他Android系统一样，灵活操作Android手机。

34.1 向管理者报告

为了充分发挥手机API的作用，可以使用`TelephonyManager`，该类允许你：

- 通过`getCallState()`确定手机是否正在使用，其返回值有`CALL_STATE_IDLE`（手机未使用）、`CALL_STATE_RINGING`（已请求呼叫，但仍在连接中）和`CALL_STATE_OFFHOOK`（接听中）；
- 通过`getSubscriberId()`获取SIM卡的ID（IMSI）；
- 通过`getPhoneType()`获取手机类型（如GSM），或通过`getNetworkType()`获取数据连接类型（如GPRS或EDGE）。

34.2 亲自进行呼叫

你也可以从应用程序（如从自己的Web服务中获取的电话号码）发起呼叫。为此，只需制

定具有 tel:NNNNN（其中 NNNNN 代表要拨打的电话号码）形式 URI 的 ACTION_DIAL Intent，并借助 startActivity()使用该 Intent。这实际上并不会拨打电话，而只是激活拨号程序 Activity，这样用户按一个按钮即可进行呼叫。

例如，我们以 Phone/Dialer 示例应用程序为例。下面是一个简单但有效的布局：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Number to dial:"
      />
    <EditText android:id="@+id/number"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:cursorVisible="true"
      android:editable="true"
      android:singleLine="true"
    />
  </LinearLayout>
  <Button android:id="@+id/dial"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Dial It!"
  />
</LinearLayout>
```

其中有一个标记字段用于键入电话号码，以及一个可拨打该号码的按钮。

Java 代码可以使用字段中的电话号码轻松开始拨号：

```
package com.commonsware.android.dialer;

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

public class DialerDemo extends Activity {
  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
```

```
    final EditText number=(EditText)findViewById(R.id.number);
    Button dial=(Button)findViewById(R.id.dial);

    dial.setOnClickListener(new Button.OnClickListener() {
      public void onClick(View v) {
        String toDial="tel:"+number.getText().toString();

        startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse(toDial)));
      }
    });
  }
}
```

活动本身的 UI 没有什么吸引人的，如图 34-1 所示。

然而，单击拨号按钮所出现的拨号程序要更好一些，可以显示要拨打的号码，如图 34-2 所示。

图 34-1　最初启动的 DialerDemo 示例应用程序

图 34-2　从 DialerDemo 启动的 Android Dialer 活动

第 35 章 开发工具

Android SDK 不仅仅是一个 Java 类库和 API 调用库，它还包含许多可协助应用程序开发的工具。

目前的关注点主要集中在 Eclipse 插件，目的是将 Android 开发与该 IDE 集成。其次便是用于其他 IDE 或者没有 IDE 的插件等同物，例如用于与运行中的模拟器通信的 adb。

本章将介绍其他工具。

35.1 层次结构管理

Android 提供了一个 Hierarchy Viewer（层级观察器）工具，其设计宗旨是提供在模拟器中运行的 Activity 的可视化布局。例如，可以确定特定的部件占用多少空间，或者尝试寻找在屏幕上未出现的部件隐藏在何处。

要使用 Hierarchy Viewer，首先需要启动模拟器，安装应用程序，启动 Activity，并通过导航找出你想要检查的对象。注意，不可以通过产品 Android 手机（例如 T-Mobile G1）使用 Hierarchy Viewer。此处为了说明，我们使用在第 23 章介绍的 ReadWrite 演示应用程序，如图 35-1 所示。

可以通过 hierarchyviewer 程序启动 Hierarchy Viewer，可在 Android SDK 安装目录中的 tools/ 目录下找到它。这将打开主 Hierarchy Viewer 窗口，如图 35-2 所示。

左侧的列表显示已经打开的各种模拟器。连字符后面的数字应该与模拟器标题栏中圆括号中的数字匹配。

单击模拟器时，右侧显示可用于检查的窗口列表，如图 35-3 所示。

注意，在我们打开的 Activity 旁边有许多其他窗口，包括 Launcher 窗口（即主屏幕）、Keyguard 窗口（即在第一次启动模拟器时出现的“Press Menu to Unlock”黑色屏幕）等。活动将通过应用程序包和类识别（例如，com.commonsware.android.files/...）。

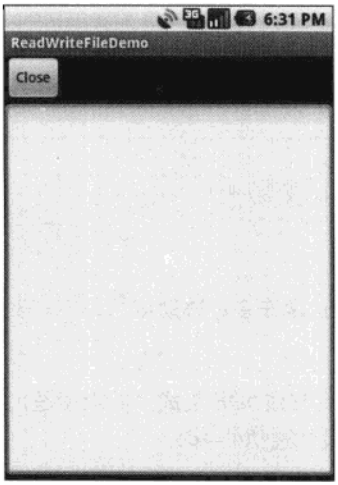


图 35-1　ReadWrite 演示应用程序

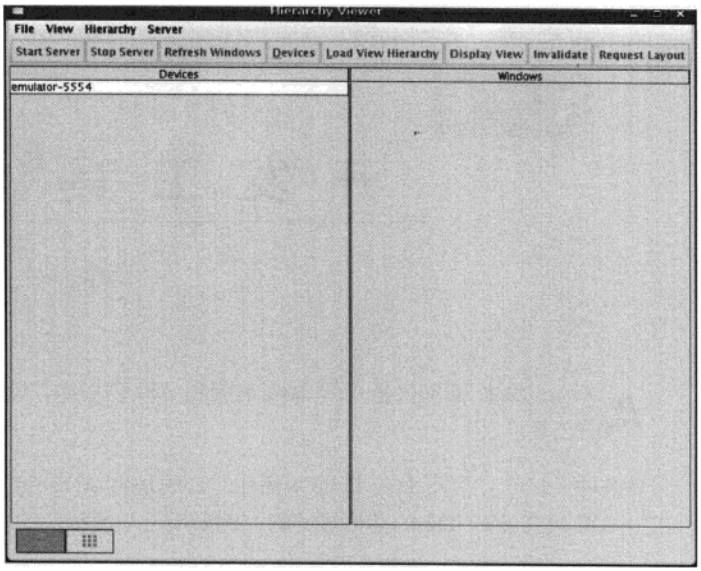


图 35-2　Hierarchy Viewer 的主窗口

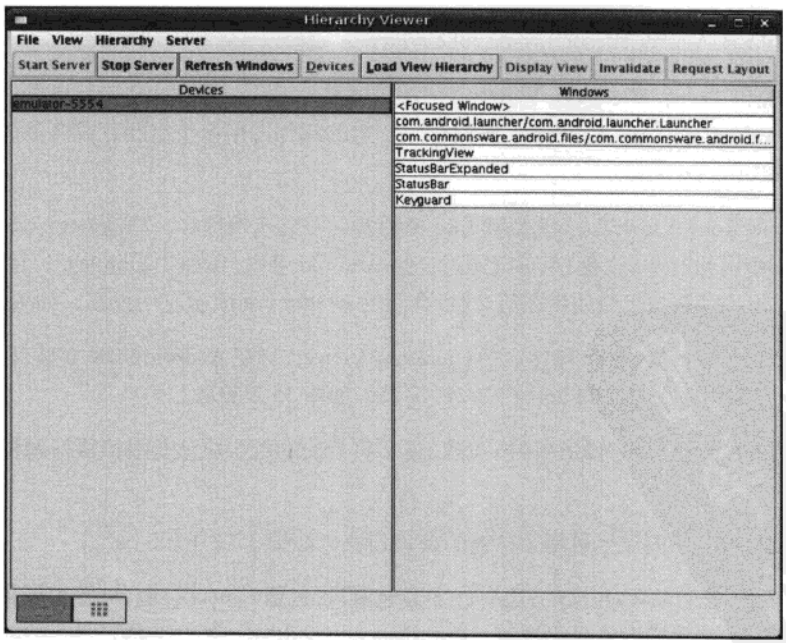


图 35-3　Hierarchy Viewer 的可用窗口列表

选择一个窗口并单击 Load View Hierarchy 时，事情会变得更有趣。几秒钟之后，视图中会显示细节，这个透视图叫做 Layout 视图，如图 35-4 所示。

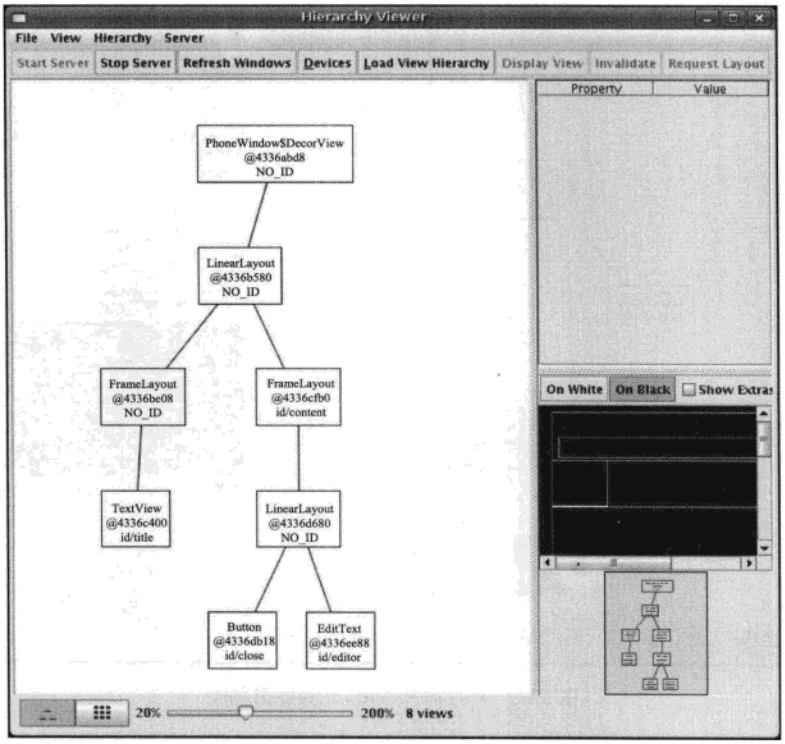


图 35-4 Hierarchy Viewer 的 Layout 视图

Layout 视图的主要区域显示一个组成 Activity 的各种视图的树，包括整体的系统窗口和单个 UI 部件。在右下角的分支，你将看到在前面的代码清单中出现过的 LinearLayout、Button 和 EditText。其余的视图都由系统提供，包括标题栏。

单击这些视图中的一个，这个透视图将显示更多的信息，如图 35-5 所示。

现在，在查看器的右上角区域，可看到选定部件（在本例中是 Button）的属性。这些属性显示不可编辑。

同时，在属性下面，部件在 Activity 的线框图中以红色高亮显示（默认的情况下，视图显示为黑色的背景，白色的轮廓）。如果你拥有多个按钮并且不能很好地从树上区分各项，这可以帮助你确保选择了正确的部件。

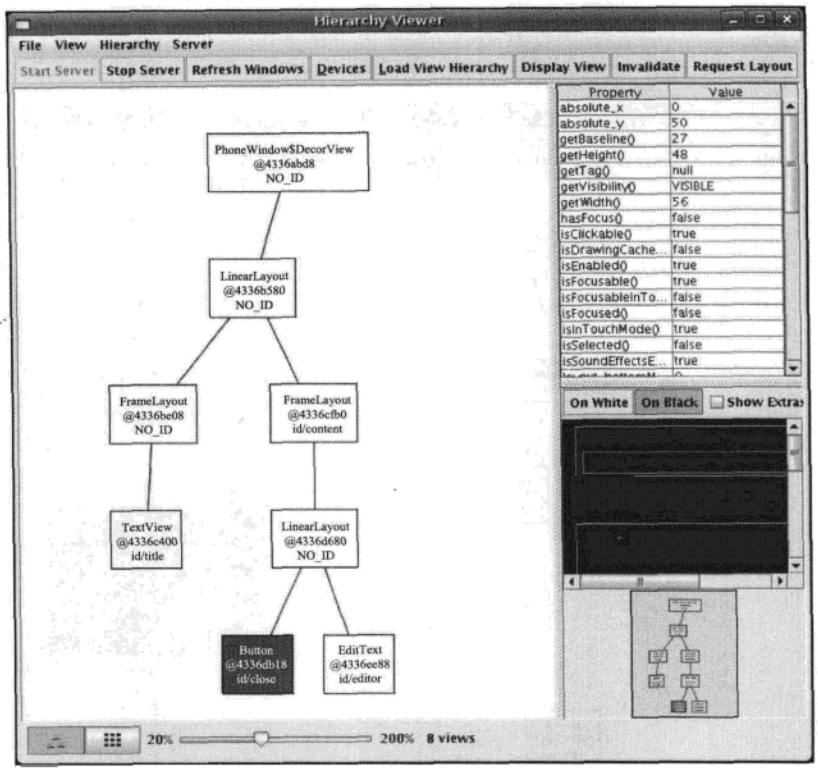


图 35-5　Hierarchy Viewer 的视图属性

如果双击树上的一个视图，将看到一个弹出窗格，只显示那个视图（及其子视图），与 Activity 的其他部分分开。

在左下角，你将看到两个切换按钮，以及开始选定的树按钮。单击网格按钮会让查看器进入一个全新的透视图，即 Pixel Perfect 视图，如图 35-6 所示。

在左侧，显示的是一个代表 Activity 中部件和其他视图的树。在中间显示 Activity（Normal 视图），在右侧显示 Activity 缩放的版本（Loupe 视图）。

起初可能不明显的是，这个图像是实时的。Activity 不时被轮询，轮询频率由 Refresh Rate 滑块控制。你在 Activity 中做的任何事情随后都会在 Pixel Perfect 视图的 Normal 和 Loupe 视图中反映出来。

覆盖在 Activity 上的细线（青色）显示被缩放的位置。单击一个新区域即可改变 Loupe 视图正在查看的位置。当然，还有另一个滑块来调整 Loupe 视图的缩放程度。

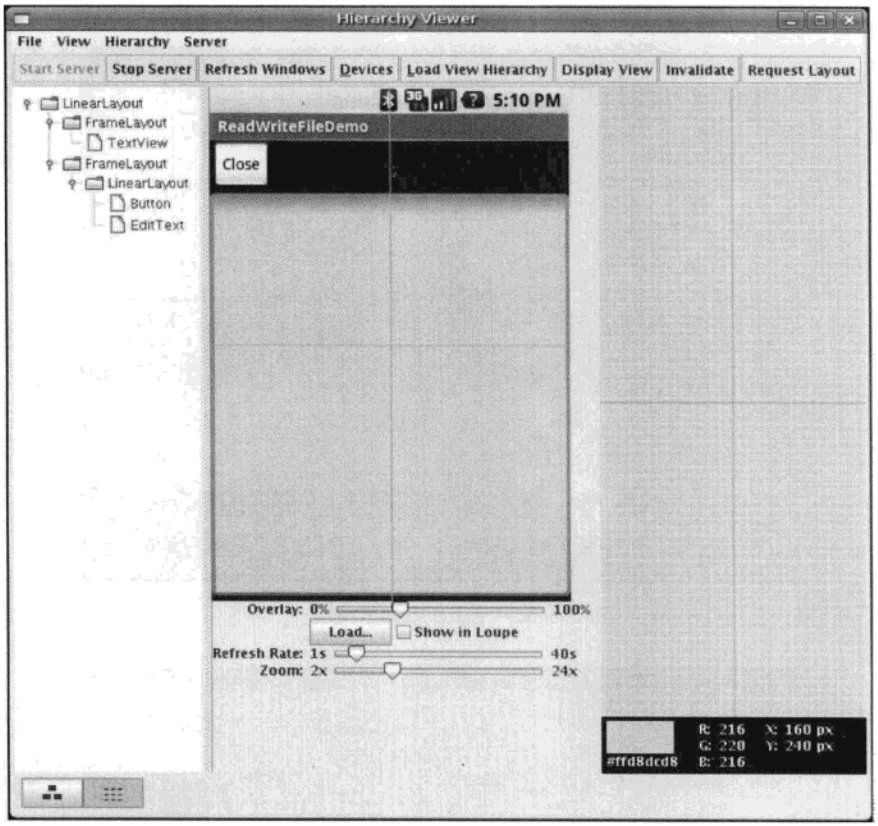


图 35-6 Hierarchy Viewer 的 Pixel Perfect 视图

35

35.2 令人愉快的 Dalvik 调试详细演示

Android 开发人员工具库中的另一件法宝是 DDMS（Dalvik Debug Monitor Service）。它就像一把瑞士军刀，能够做任何事情，包括浏览日志文件、更新模拟器提供的 GPS 位置、模拟接入呼叫和短信、浏览模拟器上的存储设备、推拉文件。

DDMS 有很多用途。此处我将介绍一些最有用的特性。

要启动 DDMS，运行 Android SDK 发行版的 tools/目录中的 ddms 程序。最初只显示一个模拟器树，在左侧显示正在运行的程序，如图 35-7 所示。

单击模拟器可以浏览底部的事件日志，并通过右侧的选项卡操纵模拟器，如图 35-8 所示。

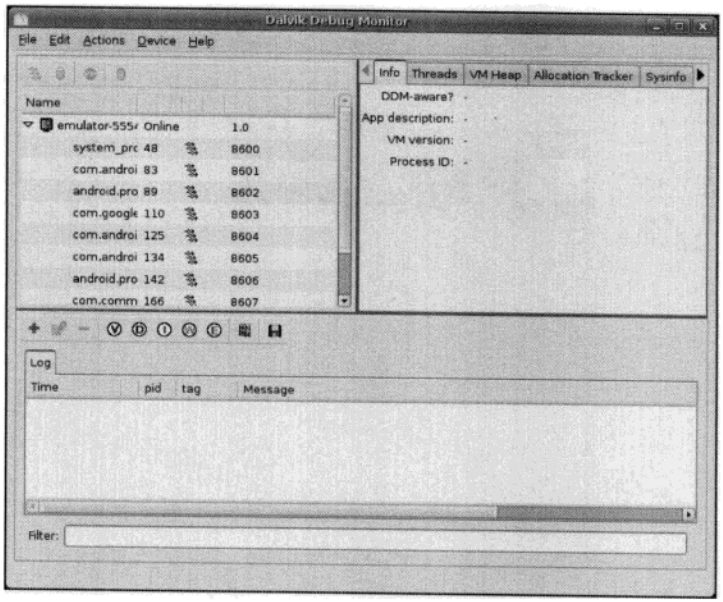


图 35-7　DDMS 初始视图

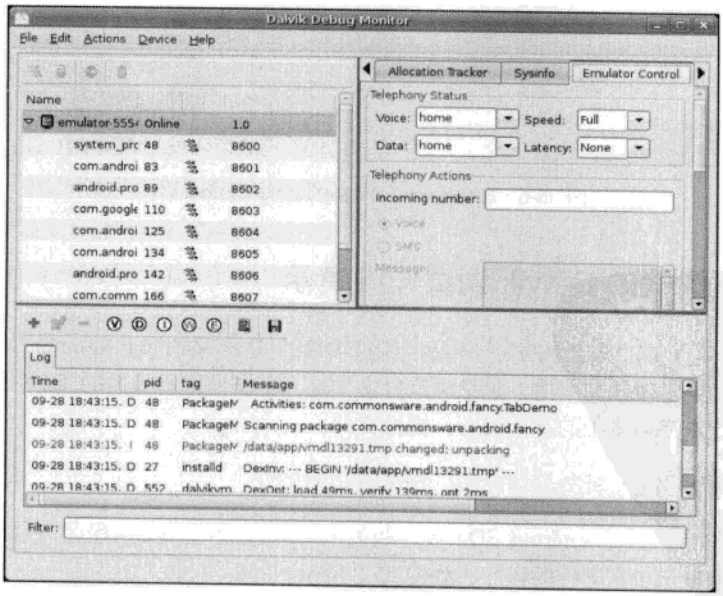


图 35-8　选定模拟器后的 DDMS

35.2.1 日志记录

不使用 adb logcat，DDMS 让你在一个可滚动的界面上查看日志信息。只需高亮显示你希望监控的模拟器或手机，屏幕的下半部分将显示日志。

此外，可以执行以下操作。

- 按照 5 个日志记录级别过滤 Log 选项卡内容，在 E 工具栏按钮上显示为 V。
- 创建自定义过滤器，以便通过单击+工具栏按钮仅查看匹配应用程序标签的条目并完成表单（参见图 35-9）。你在表单中输入的名称将被用作 DDMS 主窗口下半部分另一个日志输出选项卡的名称。
- 将日志信息保存到文本文件，以便以后细读或者搜索。

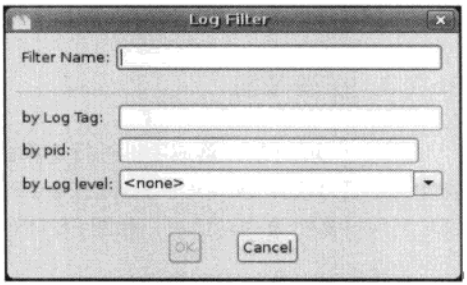


图 35-9 DDMS 日志过滤器

35.2.2 文件推拉

虽然可以使用 adb pull 和 adb push 从模拟器或手机上存放和获取文件，但 DDMS 可以让这个过程可视化。只需高亮显示要使用的模拟器或手机，然后从主菜单选择 Device → File Explorer。这会弹出典型的目录浏览器，如图 35-10 所示。

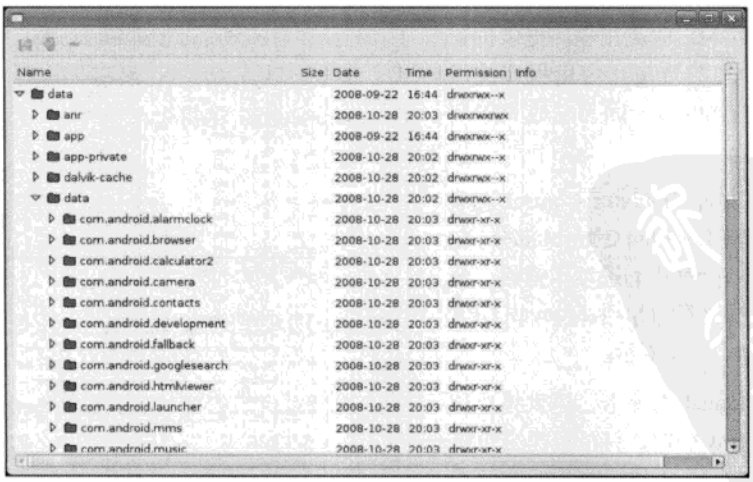


图 35-10 DDMS 文件浏览器

浏览至目标文件并单击 pull（最左侧）或 push（中央）工具栏按钮，与开发机器来回传输文件。要删除一个文件，单击 delete（最右侧）工具栏按钮。

在使用文件浏览器时有几点注意事项：

- 不能通过这个工具创建目录。要么使用 adb shell，要么在应用程序内创建它们；
- 虽然可以在模拟器上浏览大多数文件，但是因为 Android 的安全限制，在实际手机上可访问/sdcard 路径以外内容的机会极少。

35.2.3　屏幕截图

要对 Android 模拟器或手机进行屏幕截图，只需按 Ctrl+S 或者从主菜单选择 Device → Screen Capture。这时将弹出一个包含当前屏幕图像的对话框，如图 35-11 所示。

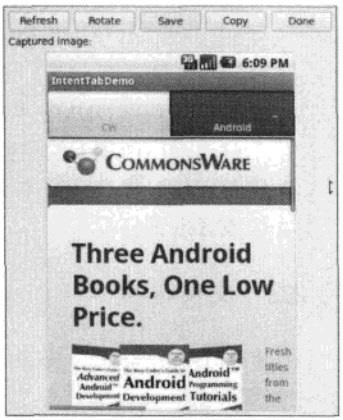


图 35-11　DDMS 屏幕截图

在此处可以执行下列操作：

- 单击 Save，在开发机器上的某个位置将图像保存为 PNG 文件；
- 单击 Refresh，基于当前的模拟器或手机状态更新图像；
- 单击 Done，关闭对话框。

35.2.4　位置更新

要使用 DDMS 向应用程序提供位置更新，首先要做的就是让应用程序使用 gps LocationProvider，因为它是 DDMS 设定更新的程序。

下一步，单击 Emulator Control 选项卡并向下滚动到 Location Control 部分。在这里，将出现一个更小的选项卡窗格，含有 3 个用于指定位置的选项：Manual、GPX 和 KML，如图 35-12 所示。

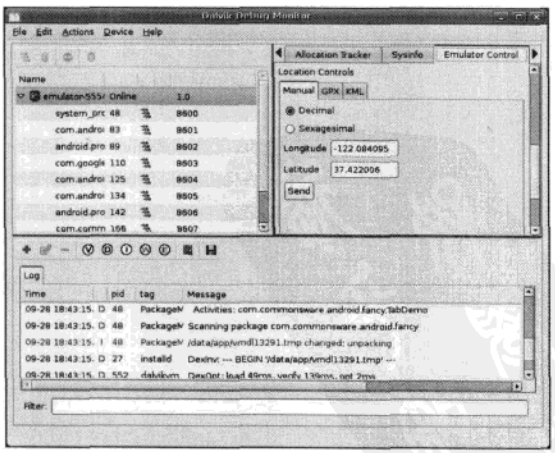


图 35-12　DDMS 位置控制

要使用 Manual 选项卡，提供经度和纬度并单击 Send 按钮，将位置提交给模拟器。模拟器会将新位置通知给所有位置侦听器。

另两个选项卡允许使用 GPX（GPS Exchange）格式或 KML（Keyhole Markup Language, Keyhole 标记语言）格式指定位置。

35.2.5 接入呼叫和消息

如果想模拟接入 Android 模拟器的呼叫或短信，DDMS 也可以处理此事。

在 Location Controls 组上面的 Emulator Control 选项卡上，有 Telephony Actions 组，如图 35-13 所示。

要模拟一个接入呼叫，请输入电话号码，选择 Voice 单选按钮，单击 Call。然后，模拟器将显示接入的呼叫，允许你接受（通过绿色电话按钮）或拒绝（通过红色电话按钮）此次呼叫，如图 35-14 所示。

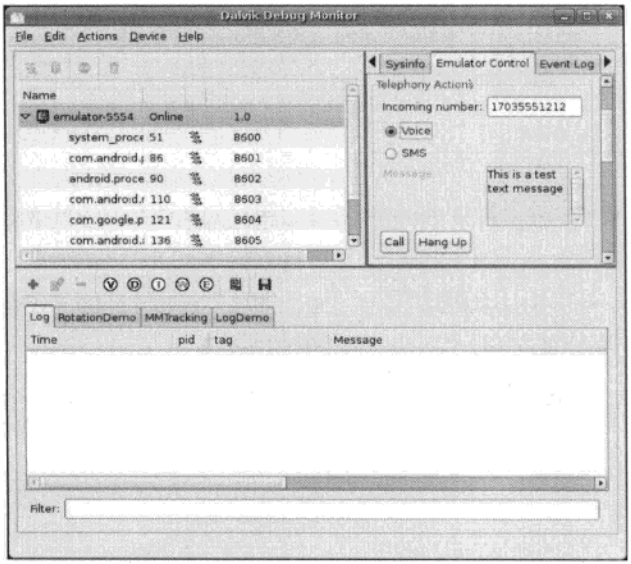


图 35-13 DDMS 电话控制

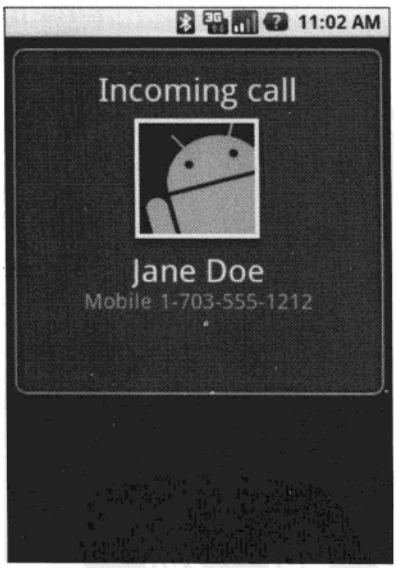


图 35-14 模拟接入呼叫

要模拟一个接入的文本消息，请输入一个电话号码，选择 SMS 单选按钮，在提供的文本区域输入一条消息，然后单击 Send。随后文本消息以通知的形式出现，如图 35-15 所示。

当然，你可以单击通知，在功能全面的消息传递应用程序中查看此消息，如图 35-16 所示。

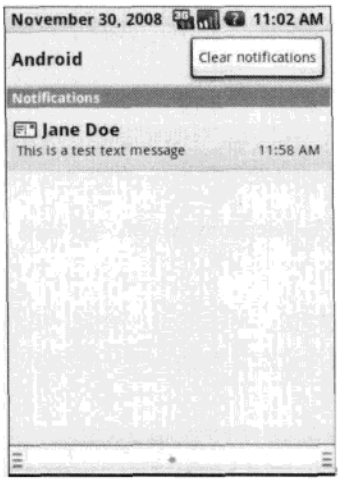


图 35-15　模拟文本消息

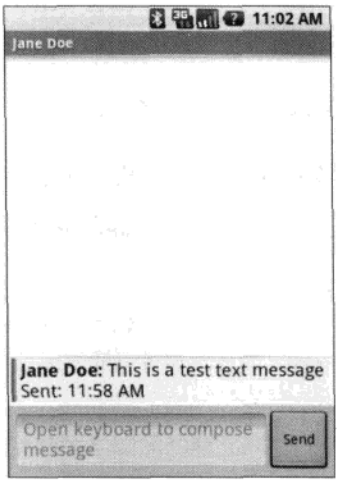


图 35-16　在消息传递应用程序中模拟显示的文本消息

35.3　存储卡

T-Mobile G1 拥有一个 microSD 卡插槽。许多其他 Android 手机也很可能拥有相似的可移动存储形式，Android 平台将其统称为 SD 卡。

强烈建议开发人员使用 SD 卡，用来存储大型数据集：图像、电影剪辑、音频文件等。特别是，T-Mobile G1 的机载闪存相对较少，所以能在 SD 卡上存储的内容越多越好。

当然，挑战是 G1 默认有一个 SD 卡，但模拟器没有。要使模拟器像 G1 一样工作，你需要创建并在模拟器中“插入”一个 SD 卡。

35.3.1　创建卡的映像

无需让模拟器以某种方式访问实际的读卡器并使用实际 SD 卡，可设置 Android 使用卡映像。一个卡的映像就是一个文件，模拟器将其视为一个 SD 卡卷。如果你习惯于通过虚拟化工具（例如 VirtualBox）使用磁盘映像，其概念是相同的。Android 使用磁盘映像代表 SD 卡内容。

使用 SDK 安装目录中 tools/目录内的 `mksdcard` 工具创建卡映像。它有两个参数，如下所述。

- 映像的大小，也就是得到的“卡”的大小。如果仅提供一个数字，它被解释为以 B 为单位的大小。另外，你可以将 K 或 M 添加到数字的末尾，分别表示以 KB 或 MB 为单位。
- 存储映像的文件名。

例如，要创建一个 1GB 的 SD 卡映像，在模拟器中模拟 G1 的 SD 卡，可运行下面的命令：

```
mksdcard 1024M sdcard.img
```

35.3.2 插入卡

要让模拟器使用这个 SD 卡映像，应使用-sdcard 开关启动模拟器，其中要包含在前面使用 mkscard 创建的映像文件的完全限定路径。虽然看不到什么影响，因为你不会在 Android 中看到图标或任何其他的东西表示你挂载了一个卡，但是现在可以对/sdcard 路径进行读取和写入。

要将文件放入/sdcard，可使用 DDMS 中的 File Explorer 或者从控制台使用 adb push 和 adb pull 命令。

第 36 章

处理多种屏幕尺寸

在Android 1.0发布后的一年内，生产的所有Android手机都有同样的屏幕分辨率（HVGA，320×480）和屏幕尺寸（约3.5英寸或9cm）。但是从2009年秋季起，开始出现了各种屏幕尺寸和分辨率的手机，从娇小的QVGA（240×320）屏幕到较大的WVGA（480×800）屏幕，各不相同。

当然，用户期待应用程序在各种屏幕上都能正常运行，并且可以充分利用更大的屏幕来实现更高的价值。为此，Android 1.6添加了新功能，目的是更好地支持不同的屏幕尺寸和分辨率。

Android 文档重点描述了处理多种屏幕尺寸的机制（http://d.android.com/guide/practices/screens_support.html）。建议你在学习本章的同时阅读这些信息，以便更好地理解如何处理多种屏幕尺寸，并且对其善加利用。

使用几段文字介绍屏幕尺寸选项和相关理论之后，本章还会深入介绍如何编写能出色处理多种屏幕尺寸，但又非常简单的应用程序。

36.1 默认设置

让我们假设在完全不考虑屏幕尺寸和分辨率的情况下开始编程，此时会怎样呢？

如果应用程序针对Android 1.5或更低版本进行编译，Android将假定应用程序的设计目的是在经典的屏幕尺寸和分辨率下效果良好。如果将应用程序安装在一个带有较大屏幕的手机上，Android会自动在兼容模式下运行该应用程序，根据实际屏幕大小对所有内容进行缩放。

例如，假设有一个24×24像素的PNG文件，Android在手机上安装并运行应用程序，该手机具有标准物理尺寸，但却具有WVGA分辨率（所谓的高密度屏幕）。Android可能将该PNG文件放大到36像素，所以它会在屏幕上占据相同的可视空间。从好处来说，Android可以自动处理这些过程。从坏处来说，位图缩放算法会导致图片有些模糊。

此外，Android将会阻止应用程序在小尺寸屏幕的手机上运行。因此，在像HTC Tattoo这样的QVGA手机上将无法运行该应用程序，尽管Android Market提供了该程序。

如果应用程序是针对Android 1.6及以上版本编译的，Android假定可正确处理所有屏幕尺寸问题，因此不会以兼容模式运行应用程序。在下一节中，你将会看到如何做到这一点。

36.2 多合一

在Android中，处理多种屏幕尺寸最简单的办法是设计UI，使其能够自动根据屏幕的大小进行缩放，无需任何特定于尺寸的编码或资源。换句话说："它就是能正常运行。"

这意味着在UI中使用的所有内容都可以被Android巧妙地缩放，也因此所有内容的大小都会很合适，即便在QVGA屏幕上也是如此。

下面几节在一个解决方案中包含了实现这些目标的一些技巧。

36.2.1 考虑规则，而不是位置

有些开发人员，或许通过拖放操作来完成UI开发，他们首要思考的是部件的位置。他们认为，需要保持特定的部件在固定位置具有固定的尺寸。他们不喜欢Android布局管理器（容器），而是倾向于将已弃用的`AbsoluteLayout`作为设计UI的一种常见方式。

这种方法很少能奏效，即便是在台式机上也是如此，就像应用程序中看到的那样，它们在重新调整窗口大小的问题上效果不佳。类似地，它在具有多种屏幕尺寸和分辨率的手机（尤其是Android）上，也不起作用。

不要考虑位置，应考虑规则。你需要为Android设定业务规则，即应该在何处调整部件的大小，以及应在何处安置部件，然后Android会根据手机屏幕实际支持的分辨率执行这些规则。

最简单的规则就是`android:layout_width`和`android:layout_height`的`fill_parent`和`wrap_content`值。它们没有指定具体的尺寸，但是可以根据可用空间进行调整。

能轻松指定规则的最好环境就是用`RelativeLayout`（第6章介绍）。虽然表面上看比较复杂，但是在让你控制布局的同时仍能根据屏幕尺寸进行调整这一方面，`RelativeLayout`表现出色。例如，你可以执行如下操作：

- 明确地将部件固定到屏幕的底部或右侧，而不是让它们由于其他布局而改变位置；
- 控制相互连接的部件（例如，字段的标签应位于字段的左侧）之间的距离，无需依靠填充内容或空白。

指定规则时最好的控制方法是创建自己的布局类。例如，假设你正在创建一系列实现纸牌游戏的应用程序。你也许希望拥有一个布局类，它知道如何玩纸牌——如何重叠、面朝上还是面朝下、要多大才能处理各种数量的纸牌等。虽然能够通过 RelativeLayout 实现想要的外观，但实现 PlayingCardLayout 或更明确地针对应用程序自定义的类可能效果会更好。遗憾的是，目前如何创建自定义布局类在文档中没有明确说明。

36.2.2　考虑物理尺寸

Android 提供了许多尺寸测量单位。最受欢迎的就是像素（px），因为它很容易让你将精力全部集中在概念上。毕竟，每一台 Android 手机都有一个屏幕，在屏幕的每个方向上都有一定数量的像素。

但是，随着屏幕密度的变化，像素开始变得令人烦恼。在给定尺寸的屏幕中，随着像素数目的增加，像素实际上在缩小。在传统的 Android 设备上，32 像素的图标也许可以用手指单击使用，但是在高密度的设备上（如 WVGA），要用手指使用该图标，32 像素可能有点小。

如果对象本质上是可缩放的（例如 Button），并且一直以像素为单位指定它的大小，可以考虑使用毫米（mm）或英寸（in）作为度量单位，而 10mm 就是 10mm，不受屏幕分辨率或屏幕尺寸的影响。这样，就可以确保部件的大小适于用手指使用，不必考虑需要多少像素。

36.2.3　避免使用实际像素

在有些环境中，使用毫米指定尺寸没有意义。所以应考虑使用其他度量单位，同时仍旧避免实际像素。

Android 提供了与密度无关的像素（dip）度量尺寸。对于 160-dpi 的屏幕（例如经典的 HVGA Android 设备），dip 与像素之间呈 1:1 比例，并由此开始进行缩放。例如，在 240-dpi 的设备上（例如，手机大小的 WVGA 设备），这一比例是 2:3，所以在 160-dpi 的屏幕中，50dip = 50px，在 240-dpi 的屏幕中，50dip=75px。用户使用 dip 的好处就是实际尺寸保持不变，所以从视觉上，160-dpi 的屏幕中的 50 dip 与 240-dpi 的屏幕中的 50 dpi 没有区别。

Android 还支持以缩放的像素（sp）作为度量尺寸。缩放的像素在理论上是基于用户选择的字体大小（System.Settings 中的 FONT_SCALE 值）进行缩放。

36.2.4　选择可缩放的 Drawable

经典的位图（PNG、JPG 和 GIF）从本质上讲都不可缩放。如果没有在兼容模式下运行程序，Android 不会试图根据屏幕分辨率和尺寸对它们进行缩放。不论你提供的位图是什么尺寸，它都

将是原来那个尺寸，尽管这可能让图像在某些屏幕上显得太大或太小。

解决这个问题的一个方式是尝试避免使用静态位图，使用 Nine-Patch（可拉伸）位图和 XML 定义的 Drawable（例如 GradientDrawable）作为替代方法。Nine-Patch（可拉伸）位图是一个 PNG 文件，进行了特殊编码，以通过规则说明如何拉伸该图像以占据更多的空间。XML 定义的 Drawable 使用一个类似 SVG XML 的语言来定义形状、笔触和填充等。

36.3 量身定制

有时，希望根据屏幕尺寸或密度的不同拥有不同的外观或行为。Android 提供了根据应用程序运行环境切换资源或代码块的方式。正确结合上一节中的技巧，实现与屏幕尺寸和密度无关的程序是完全可能的，至少对于运行 Android 1.6 及更高版本的手机来说是这样。

36.3.1 添加<supports-screens>

主动支持屏幕大小的第一步是将<supports-screens>元素添加到 AndroidManifest.xml 文件。这指定了要明确支持和不支持的屏幕尺寸。不明确支持的屏幕尺寸将由自动兼容模式处理，如前面所述。

下面是一个包含<supports-screens>元素的清单文件：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.eu4you"
  android:versionCode="1"
  android:versionName="1.0">
  <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
    android:smallScreens="true"
    android:anyDensity="true"
  />
  <application android:label="@string/app_name"
    android:icon="@drawable/cw">
    <activity android:name=".EU4You"
              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>
```

android:smallScreens、android:normalScreens 和 android:largeScreens 这 3 个特性都接受布尔值，指定应用程序明确支持那些屏幕（true），还是需要兼容模式的帮助（fasle）。

android:anyDensity 特性指定是否正在应用程序中将密度考虑在内（true 或 false）。如果为

fasle，Android将认为屏幕的所有尺寸（例如4px）都针对正常密度（160-dpi）屏幕。如果应用程序正运行于一个低密度或高密度的屏幕上，Android将相应地缩放尺寸。如果指定android:anyDensity = "true"，就等于告诉Android不要这样做，让你自己来使用与密度无关的单位，例如dip、mm或in。

36.3.2　资源和资源集

基于屏幕的大小或密度切换不同内容的主要方法是创建资源集。通过针对不同设备特点创建资源集，就能告诉Android如何渲染这些资源，而且Android能在这些资源集之间自动切换。

1. 默认缩放

默认情况下，Android将会缩放所有Drawable资源。那些本质上可缩放的资源将很好地缩放。Android将使用正常的缩放算法缩放普通的位图，这可能会产生很好的结果，也可能不会。这种缩放还可能会稍微减慢操作。如果不想出现这种情况，需要建立其中包含不可缩放位图的独立资源集。

2. 基于密度的设置

如果想根据不同的屏幕密度拥有不同的布局、尺寸等，可以使用-ldpi、-mdpi和-hdpi资源集标签。例如，res/values-hdpi/dimens.xml将包含在高密度设备中使用的尺寸。

3. 基于尺寸的设置

同样，如果想根据屏幕尺寸拥有不同的资源集，可使用Android提供的-small、-normal和-large资源集标签。创建res/layout-large-land/可指定用于横向大屏幕（如WVGA）上的布局。

4. 基于版本的设置

有时，Android的较早版本对于新的资源集标签会感到困惑。为了解决这一问题，可以向资源集加入一个-vN形式的版本标签，N表示API的级别。因此，res/drawable-large-v4/说明这些Drawable资源应该在API级别4（Android 1.6）以及更高版本上的较大屏幕中使用。

Android能够从很早开始对版本进行过滤，并且这种技术可以追溯到Android 1.5（可能更早）。

所以，如果发现Android 1.5模拟器或手机正在获取错误的资源集，应考虑向它们的资源集名中添加-v4，将它们过滤掉。

36.3.3　查找尺寸

如果需要基于屏幕尺寸或密度在Java代码中采用不同的操作，这里提供几种选择。

如果资源集中有独特的内容，你可以找到它并在代码中相应地使用分支语句。例如，正如本章后面的代码示例所示，你可以在一些布局中添加额外的部件（例如 res/layout-large/main.xml）；只需查看额外的部件是否存在，便能知道是否将在一个大屏幕上运行程序。

还可以通过 Configuration 对象查找屏幕尺寸类，这个对象通常由一个 Activity 通过 getResources().getConfiguration()获得。Configuration 对象拥有一个称作 screenLayout 的公共字段，这是一个说明正在运行应用程序的屏幕类型的位掩码。通过测试可以查看屏幕是小的、正常的还是大的，或者是否为长型的（长型是指纵横比为 16:9 或类似纵横比，相对于 4:3 而言）。例如，我们现在测试一下，看看是否正在大屏幕上运行应用程序：

```
if (getResources().getConfiguration().screenLayout
      & Configuration.SCREENLAYOUT_SIZE_LARGE)
    ==Configuration.SCREENLAYOUT_SIZE_LARGE) {
 // yes, we are large
}
else {
 // no, we are not
}
```

目前没有轻松的类似方式可以查看屏幕密度。如果你必须知道屏幕密度，可以在你的项目中创建 res/values-ldpi/、res/values-mdpi/和 res/values-hdpi/目录，并分别为它们添加一个 strings.xml 文件。将一个字符串资源放入 strings.xml，这个文件在全部 3 个资源集中拥有相同的名称，但是值不同（例如名称为 density，值分别是 ldpi、mdpi 和 hdpi）。然后，在运行时测试字符串资源的值。这种方法不是太好，但的确能起作用。

36.4 一切都是模拟的

Android 模拟器将会帮助你在不同尺寸的屏幕上测试应用程序。但是，它只能提供这么多，因为手机 LCD 的特点与台式机或笔记本电脑的不同，例如：

- 手机 LCD 的密度可能比开发机器的密度高很多；
- 与实际的指尖相比，鼠标能够支持更精确的触摸屏输入。

在可能的情况下，你要么需要以新奇的方式使用模拟器，要么需尝试用不同的屏幕分辨率在实际的手机上操作。

36.4.1 密度不同

Motorola DROID 拥有一个 240-dpi、3.7 英寸、480 × 854 像素的屏幕（EWVGA 显示屏）。要根据像素数模拟 DROID 屏幕，需要占用 19 英寸、1280 × 1024 LCD 显示器的三分之一，因为 LCD 显示器的密度比 DROID 的要低得多——只有大约 96dpi。所以，将用于（像 DROID 这样的）FWVGA 显示屏的 Android 模拟器启动时，将获得大得多的模拟器窗口。

这对于在 FWVGA 环境中确定整体应用程序外观来说再好不过。不论密度如何，部件都将保持一致对齐，尺寸的相互关系保持一致（例如，部件 A 的高度可能是部件 B 的两倍，无论密度如何，这一关系将保持不变），等等。

但是，可能会出现下列问题：

- 在 19 英寸的 LCD 上看似尺寸很合适的对象，在具有同样分辨率的手机屏幕上，可能显得太小；
- 在模拟器上可使用鼠标轻松单击的对象，在一个物理上更小、密度更高的屏幕上使用手指单击时，可能显得太小。

36.4.2　调整密度

在默认情况下，模拟器将以密度为代价，保持像素数目的准确性，这就是为什么你会获得实际上很大的模拟器窗口。你也可以选择以像素数目为代价，保持密度准确。

为此，最简单的方式是用在 Android 1.6 中引入的 Android AVD Manager。此工具的 Android 2.0 版本拥有一个 Launch Options 对话框，在通过 Start 按钮启动一个模拟器实例时，此对话框弹出，如图 36-1 所示。

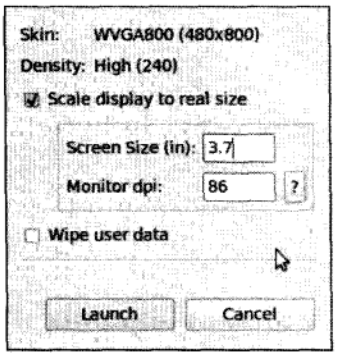


图 36-1　Launch Options 对话框

默认情况下，“Scale display to real size”复选框是未选中的，Android 将正常打开模拟器窗口。可以选中该复选框，然后提供两条缩放信息：

- 要模拟的手机屏幕的尺寸，以英寸为单位（例如，Motorola DROID 是 3.7 英寸）；
- 监视器的分辨率，以每英寸点数为单位（单击？按钮可调出计算器，帮助你确定这个值）。

这可获得一个模拟器窗口，该窗口将更准确地描绘 UI 在手机上的外观，至少在尺寸上是这样的。但是，因为模拟器使用的像素点数远远小于手机，字体可能难以阅读，图像可能浓淡不匀，等等。

36.4.3　访问实际设备

当然，查看应用程序在不同设备上的外观时，最佳方法是在不同的手机上进行实际测试。你不必获得所有已生产出来的 Android 手机，但是需要访问那些带有独特硬件的设备，它们可能对应用程序产生影响，并且屏幕大小几乎会影响到每一个人。下面是一些建议。

- 使用 DeviceAnywhere（http://www.deviceanywhere.com/）等服务虚拟地测试手机。这是对模拟器的改进，但是不是免费的，也肯定不能测试所有功能（例如位置变化）。
- 购买手机，可通过 eBay 这样的支持渠道。在需要测试手机操作，或者在无 SIM 的情况下，解锁的 GSM 手机可以用于共享 SIM。
- 如果住在一个城市内或城市附近，你可以建立某种形式的用户群，并利于这个群在集体硬件集上测试应用程序。
- 按照用户测试路线进行，将应用程序作为免费 beta 版发布，然后根据用户的反馈进行调整。你可以将它分发到 Android Market 外部，以免 beta 测试反馈影响应用程序的市场评分。

36.5 充分利用形势

目前我们已经讨论了如何确保布局外观在其他尺寸的屏幕上依然美观。而且，对于小于正常情况的屏幕（例如 QVGA），那也许是能实现的极限。

一旦使用更大的屏幕，另一些可能性就出现了：使用不同布局来利用额外的屏幕空间。这在实际屏幕更大的情况下更为有用（例如在 Archos 5 Android 平板电脑上的 5 英寸的 LCD），而不是简单地在相同物理空间内增加像素的数目。

以下各节介绍了利用额外空间的一些方式。

36.5.1 用按钮代替菜单

选项菜单的选择需要两个实际操作：单击 Menu 按钮，然后单击合适的菜单选项。上下文菜单选择也要求两个实际操作：长时单击部件，然后单击菜单选项。上下文菜单还有可能看不到的问题，例如用户可能意识不到 `ListView` 有一个上下文菜单。

你可以考虑增强 UI，在屏幕上提供直接的方式来完成操作，否则该操作将会隐藏在菜单中。这不仅降低了用户执行任务需要完成的步骤数，而且使这些选项更明显。

例如，假设你正在创建一个媒体播放器应用程序，并且希望提供手动的播放列表管理。你拥有一个 `Activity`，可将歌曲显示在 `ListView` 内的播放列表中。在选项菜单上有一个 Add 选项，用于从手机上向播放列表中添加新歌曲。在 `ListView` 上的上下文菜单上有一个 Remove 选项，以及 Move UP 和 Move Down 选项，用来重新排序列表中的歌曲。在更大的屏幕中，你可以考虑为这 4 个选项在 UI 中添加 4 个 `ImageButton` 部件，其中原来在上下文菜单中的 3 个只有在用 D-pad 或跟踪球选择了一行后才启用。在正常或小屏幕上，你应坚持使用菜单。

36.5.2 使用简单的 Activity 代替选项卡

你可能已经将 `TabHost` 引入到 UI，以便在可用的屏幕空间内显示更多的部件。只要将部件

转移到独立的选项卡所节省的空间比选项卡本身占据的空间更大，你就赢了。但是，拥有很多选项卡意味着，用户在UI中导航需要更多步骤，特别是当用户需要在选项卡之间频繁来回跳转时。

如果仅有两个选项卡，可考虑改变UI来提供一个大屏幕布局，即移除选项卡并将所有的部件放在一个屏幕上。将一切呈现在用户面前，无需在选项卡之间切换。

如果拥有3个及以上的选项卡，可能缺少屏幕空间，不能将所有选项卡内容放入一个Activity。但是可以考虑一半一半地来：将受欢迎的部件放在Activity中，让TabHost在一半屏幕中处理剩余的部件。

36.5.3　整合多个Activity

最强大的技术是用更大的屏幕彻底摆脱Activity的过渡。例如，如果拥有ListActivity，其中单击一个条目将在另一个独立Activity中弹出该条目的详细信息，请考虑支持一个大屏幕布局，让详细信息与ListView显示在同一Activity中（例如，在水平的布局中，ListView在左侧，详细信息在右侧）。这消除了用户必须不断按下Back按钮才能离开一组详细信息，然后再查看另一组的必要。

你将在下节的示例代码中看到这个技巧的应用方式。

36.6　示例：EU4You

要想知道如何使用目前所讨论的技术，让我们看一看ScreenSizes/EU4You示例应用程序。该应用程序有一个Activity（EU4You），其中包含EU（European Union, 欧盟）成员名单及其各自国旗的ListView（http://www.wpclipart.com/flags/Countries/index.html）。单击某个国家会弹出该国的移动维基页面。

在本书的源代码中，你将看到这个应用程序的4个版本。我们从忽略屏幕大小的那个应用程序开始，慢慢添加更多屏幕相关的功能。

36.6.1　第一个版本

首先，下面是AndroidManifest.xml文件，看上去与我们在本章前面所示的很像：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.commonsware.android.eu4you"
  android:versionCode="1"
  android:versionName="1.0">
  <supports-screens
    android:largeScreens="true"
    android:normalScreens="true"
```

```
    android:smallScreens="true"
    android:anyDensity="true"
  />
  <application android:label="@string/app_name"
    android:icon="@drawable/cw">
    <activity android:name=".EU4You"
              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>
```

注意，我们拥有<supports-screens>元素，说明我们确实支持所有的屏幕尺寸。如果说我们不支持特定的屏幕尺寸，是因为这个元素阻止了 Android 执行自动缩放功能。

主布局是与尺寸无关的，因为它是一个全屏 ListView：

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

但是我们的行最终需要一些调整：

```
<?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="2dip"
  android:minHeight="?android:attr/listPreferredItemHeight"
>
  <ImageView android:id="@+id/flag"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|left"
    android:paddingRight="4px"
  />
  <TextView android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|right"
    android:textSize="20px"
  />
</LinearLayout>
```

例如，现在字体大小设为 20px，并且不会随着屏幕尺寸或密度的变化而变化。

EU4You 这个 Activity 有一点冗长，主要是因为 EU 成员数目众多，并且我们需要运用技巧在同一行中显示国旗和文本。

```
package com.commonsware.android.eu4you;

import android.app.ListActivity;
```

```
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;
import java.util.ArrayList;

public class EU4You extends ListActivity {
  static private ArrayList<Country> EU=new ArrayList<Country>();

  static {
    EU.add(new Country(R.string.austria, R.drawable.austria,
                       R.string.austria_url));
    EU.add(new Country(R.string.belgium, R.drawable.belgium,
                       R.string.belgium_url));
    EU.add(new Country(R.string.bulgaria, R.drawable.bulgaria,
                       R.string.bulgaria_url));
    EU.add(new Country(R.string.cyprus, R.drawable.cyprus,
                       R.string.cyprus_url));
    EU.add(new Country(R.string.czech_republic,
                       R.drawable.czech_republic,
                       R.string.czech_republic_url));
    EU.add(new Country(R.string.denmark, R.drawable.denmark,
                       R.string.denmark_url));
    EU.add(new Country(R.string.estonia, R.drawable.estonia,
                       R.string.estonia_url));
    EU.add(new Country(R.string.finland, R.drawable.finland,
                       R.string.finland_url));
    EU.add(new Country(R.string.france, R.drawable.france,
                       R.string.france_url));
    EU.add(new Country(R.string.germany, R.drawable.germany,
                       R.string.germany_url));
    EU.add(new Country(R.string.greece, R.drawable.greece,
                       R.string.greece_url));
    EU.add(new Country(R.string.hungary, R.drawable.hungary,
                       R.string.hungary_url));
    EU.add(new Country(R.string.ireland, R.drawable.ireland,
                       R.string.ireland_url));
    EU.add(new Country(R.string.italy, R.drawable.italy,
                       R.string.italy_url));
    EU.add(new Country(R.string.latvia, R.drawable.latvia,
                       R.string.latvia_url));
    EU.add(new Country(R.string.lithuania, R.drawable.lithuania,
                       R.string.lithuania_url));
    EU.add(new Country(R.string.luxembourg, R.drawable.luxembourg,
                       R.string.luxembourg_url));
    EU.add(new Country(R.string.malta, R.drawable.malta,
                       R.string.malta_url));
    EU.add(new Country(R.string.netherlands, R.drawable.netherlands,
                       R.string.netherlands_url));
    EU.add(new Country(R.string.poland, R.drawable.poland,
                       R.string.poland_url));
    EU.add(new Country(R.string.portugal, R.drawable.portugal,
                       R.string.portugal_url));
    EU.add(new Country(R.string.romania, R.drawable.romania,
```

```
                R.string.romania_url));
  EU.add(new Country(R.string.slovakia, R.drawable.slovakia,
                R.string.slovakia_url));
  EU.add(new Country(R.string.slovenia, R.drawable.slovenia,
                R.string.slovenia_url));
  EU.add(new Country(R.string.spain, R.drawable.spain,
                R.string.spain_url));
  EU.add(new Country(R.string.sweden, R.drawable.sweden,
                R.string.sweden_url));
  EU.add(new Country(R.string.united_kingdom,
                R.drawable.united_kingdom,
                R.string.united_kingdom_url));
}

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

@Override
protected void onListItemClick(ListView l, View v,
                               int position, long id) {
  startActivity(new Intent(Intent.ACTION_VIEW,
                           Uri.parse(getString(EU.get(position).url))));
}

static class Country {
  int name;
  int flag;
  int url;

  Country(int name, int flag, int url) {
    this.name=name;
    this.flag=flag;
    this.url=url;
  }
}

class CountryAdapter extends ArrayAdapter<Country> {
  CountryAdapter() {
      super(EU4You.this, R.layout.row, R.id.name, EU);
    }

    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent) {
      CountryWrapper wrapper=null;

      if (convertView==null) {
        convertView=getLayoutInflater().inflate(R.layout.row, null);
        wrapper=new CountryWrapper(convertView);
        convertView.setTag(wrapper);
      }
      else {
        wrapper=(CountryWrapper)convertView.getTag();
      }
```

```
      wrapper.populateFrom(getItem(position));

      return(convertView);
    }
  }

  class CountryWrapper {
    private TextView name=null;
    private ImageView flag=null;
    private View row=null;

    CountryWrapper(View row) {
      this.row=row;
    }

    TextView getName() {
      if (name==null) {
        name=(TextView)row.findViewById(R.id.name);
      }

      return(name);
    }

    ImageView getFlag() {
      if (flag==null) {
        flag=(ImageView)row.findViewById(R.id.flag);
      }

      return(flag);
    }

    void populateFrom(Country nation) {
      getName().setText(nation.name);
      getFlag().setImageResource(nation.flag);
    }
  }
}
```

图 36-2、图 36-3 和图 36-4 分别显示了在普通的 HVGA 模拟器、WVGA 模拟器以及 QVGA 屏幕中 Activity 的外观。

图 36-2　EU4You，原始版本，HVGA

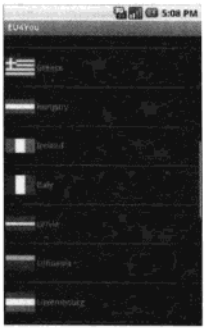


图 36-3　EU4You，原始版本，WVGA（800 × 480 像素）

图 36-4　EU4You，原始版本，QVGA

36.6.2 固定字体大小

第一个应该固定的问题是字体大小。正如你看到的，固定为 20 像素大小的字体显示尺寸大小不一，这取决于屏幕尺寸和密度。在 WVGA 屏幕上，该字体可能很难阅读。

我们可以将该大小作为一个资源（res/values/dimens.xml），并且根据屏幕尺寸或密度拥有该资源的不同版本。但是，指定一个与密度无关的大小则更简单一些，例如 5mm，如在 ScreenSizes/EU4You_2 项目中所示：

```
<?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="2dip"
  android:minHeight="?android:attr/listPreferredItemHeight"
>
  <ImageView android:id="@+id/flag"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|left"
    android:paddingRight="4px"
  />
  <TextView android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_vertical|right"
    android:textSize="5mm"
  />
</LinearLayout>
```

图 36-5、图 36-6 和图 36-7 分别显示了在 HVGA、WVGA 和 QVGA 屏幕上的程序运行结果。

图 36-5 EU4You，5mm 字体版本，HVGA

图 36-6 EU4You，5mm 字体版本，WVGA（800×480 像素）

图 36-7 EU4You，5mm 字体版本，QVGA

现在我们的字体大小一致，并且足够大，可与国旗相匹配。

36.6.3　固定大小的图标

那么，图标呢？它们也应该有不同的大小，因为它们在 3 个模拟器上应该是相同的。

但是，Android 会自动缩放位图资源，即使有<supports-screens>并且其特性设为 true。从好处来说，这意味着你可能不需要对这些位图做任何事情。但是，你依靠手机进行缩放，这毫无疑问地需要占用 CPU 时间（并且因此会降低电池使用时间）。而且，与在开发机器上使用图形工具能够实现的效果相比，手机使用的缩放算法可能不是最佳的。

ScreenSizes/EU4You_3 项目创建 res/drawable-ldpi 和 res/drawable-hdpi，分别放入较小的和较大的国旗。这个项目也将 res/drawable 重命名为 res/drawable-mdpi。Android 将按照手机或模拟器的需要，针对合适的屏幕密度使用国旗。

36.6.4　使用空间

在 WVGA 上，Activity 在纵向模式中看起来不错，在横向模式中却会浪费许多空间，如图 36-8 所示。

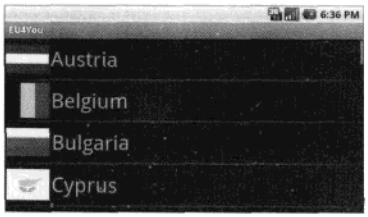


图 36-8　EU4You，横向 WVGA（800×480 像素）

在大屏幕横向模式中，我们可以通过让维基内容直接显示在主 Activity 上来更好地使用屏幕空间，无需创建独立的浏览器 Activity。

要实现这一点，我们首先必须将 main.xml 布局复制到一个 res/layout-large-land 呈现中，后者整合了一个 WebView 部件，如 ScreenSizes/EU4You_4 所示：

```
<?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"
>
  <ListView
    android:id="@android:id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
  />
  <WebView
    android:id="@+id/browser"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
  />
</LinearLayout>
```

然后，我们需要调整 Activity 来寻找那个 WebView，并且一旦找到就使用它；否则会默认启动一个浏览器 Activity：

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

  browser=(WebView)findViewById(R.id.browser);

  setListAdapter(new CountryAdapter());
}

@Override
protected void onListItemClick(ListView l, View v,
                               int position, long id) {
  String url=getString(EU.get(position).url);

  if (browser==null) {
    startActivity(new Intent(Intent.ACTION_VIEW,
                               Uri.parse(url)));
  }
  else {
    browser.loadUrl(url);
  }
}
```

这就得到了一个空间使用更加高效的 Activity 版本，如图 36-9 所示。

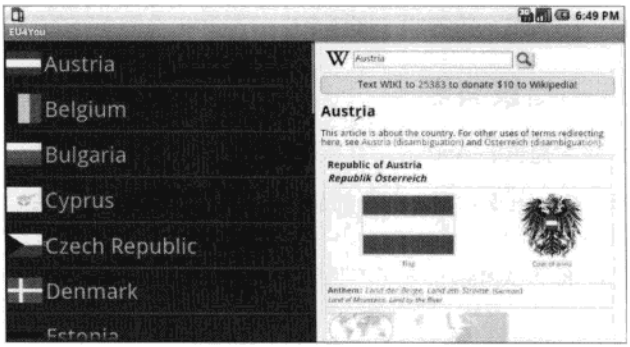


图 36-9 EU4You，横向 WVGA（800×480 像素），适用于正常密度的显示屏并且显示了嵌入的 WebView

当用户单击维基页面中的一个链接时，将打开一个完整的浏览器，让用户实现轻松冲浪。

注意，要测试这个 Activity 版本，并查看这个行为，需要一些额外的模拟器工作。默认情况下，Android 将 WVGA 设备设置为高密度，意味着 WVGA 的资源集不是大型的，而是 normal。你需要创建一个不同的模拟器 AVD，设置为正常（medium）密度，结果会得到 large 屏幕尺寸。

36.6.5 不是浏览器会怎样

当然，EU4You 也用了一点欺骗方法。次要 Activity 是浏览器（或嵌入式的 WebView），不是

你自己创建的 Activity。如果次要 Activity 是你自己的某个 Activity，那么事情就变得有些复杂。通过布局中的许多部件，你希望既可以将它用作 Activity（对于小屏幕），又要将它嵌入主 Activity 的 UI 中（对大屏幕）。

下面的模式可以处理这种情况。

(1) 首先将次要 Activity 作为一个 Activity 开发并测试。

(2) 将次要 Activity 的所有生命周期方法的逻辑委托给一个内部类。将那个内部类所需的 Activity 的所有数据成员移到那个内部类中，并且确保其能正常工作。

(3) 将内部类拉入一个单独的公共类，并确保它正常工作。

(4) 对于第一个（主）Activity，创建一个用于大屏幕的独立布局，并使用<include>指令将次要 Activity 的布局内容混合到第一个大屏幕 Activity 的布局中正确的地方。

(5) 在第一个 Activity 中，如果发现次要 Activity 的布局已经作为自身的一部分扩充（例如，通过 findViewById()检查某些部件的存在与否），可创建一个你在步骤(3)中创建的公共类的实例，并让其处理所有部件。修改代码以直接引用那个类，而不是如上一节所示启动次要 Activity。

总之，使用一个公共类和可重用的布局来保持代码和资源在一处，从独立的 Activity 使用它们，或作为主 Activity 大屏幕版本的一部分来使用。

36.7　合作伙伴的错误有哪些

Motorola DROID 随附提供的 Android 2.0 在屏幕尺寸方面有两个错误。

- 屏幕密度值不正确，水平和垂直方向都是如此。这意味着它没有基于物理大小 pt、mm 和 in 正确缩放尺寸。
- 它将 Android 2.0 作为 API 级别 6 而不是级别 5，因此特定于版本的资源目录需要使用-v6 后缀而不是-v5 后缀。

这两个错误都在 Android 2.0.1 以及更高版本中修复了，并且其他设备不提供 Android 2.0 或不受这些错误的影响。

第 37 章 手机的处理 37

Android 对于手机制造商来说是完全免费的，因为它是一个开源项目。因此，手机制造商有权随意使用 Android。这意味着手机使用者有诸多选择，他们能够使用任意形状、大小和颜色的 Android 手机，这也意味着开发人员要将一些手机的不同和特质考虑在内。

本章将介绍一些诀窍和建议来处理这些手机特定的问题，以支持第 36 章中关于屏幕大小的论述。

37.1 该应用程序包含显式指令

最初，唯一的 Android 手机是 T-Mobile G1。因此，如果正在编写一个 Android 应用程序，就要假定手机有硬件 QWERTY 键盘，一个导航轨迹球，等等。现在，有带不同硬件功能（例如，无键盘）的其他手机（例如，HTC Magic）。

理想情况下，不管是否存在各种类型的硬件，应用程序都可运行。不过，有些应用程序在没有某些硬件特性的情况下是不可用的。例如，一个全屏游戏可能依赖于硬件键盘或轨迹球来指明玩家的动作——仅有软键盘和触摸屏可能就不够了。

幸运的是，从 Android 1.5 开始，可以通过添加显式指令告诉 Android 应用程序的需求，因此不会将应用程序安装在缺少这种硬件的手机上。

除了使用目标 ID 系统指出项目所针对的设备级别之外，可以使用一个新的 AndroidManifest.xml 元素指定应用程序正常运行所需的硬件。可以在<manifest>元素内添加一个或多个<uses-configuration>元素。每个<uses-configuration>元素为应用程序运行所需的硬件指定一个有效配置。目前，通过这种方式可以指定 5 个可行的硬件需求：

- android:reqFiveWayNav：表示需要一个某种形式的五向导航定点手机（例如，android:reqFiveWayNav = "true"）；
- android:reqNavigation：将五向导航定点手机限制为某个类型（例如，android:reqNavigation = "trackball"）；

- android:reqHardKeyboard：指定是否需要一个硬件（物理）键盘（例如，android:reqHardKeyboard = "true"）；
- android:reqKeyboardType：与android:reqHardKeyboard联合使用，表示需要特定类型的硬件键盘（例如，android:reqKeyboardType = "qwerty"）；
- android:reqTouchScreen：表示如果有触摸屏的话，需要什么类型的触摸屏（例如，android:reqTouchScreen = "finger"）。

从Android 1.6开始，有一个类似的清单文件元素<uses-feature>，它旨在记录一个应用程序对Android手机上其他可选特性的需求。具体地说，可将以下特性放在一个<uses-feature>元素中：

- android:glEsVersion：表示应用程序需要OpenGL，其中特性的值表示OpenGL支持级别（例如，对于OpenGL 1.2或更高版本是0x00010002）；
- android:name = "android.hardware.camera"：表示应用程序需要一个照相机。
- android:name = "android.hardware.camera.autofocus"：表示应用程序明确需要一个自动对焦照相机。

37.2　按钮

与屏幕上的软键以及给定Android手机上按钮不可用相比，关于提供什么物理按钮，甚少对手机制造商有要求。

例如，HTC Dream（又名T-Mobile G1）有呼叫、结束呼叫、主目录、后退、菜单和照相机按钮，且在QWERTY键盘上有一个音量控制和专用的搜索按钮。HTC Magic（又名T-Mobile myTouch 3G）缺少照相机按钮，只有搜索按钮。Archos 5 Android Internet Tablet除了音量控制之外根本无硬件按钮，但有供主目录、后退和菜单使用的软键。

因此，在假定硬件按钮是否存在或按钮的位置时要谨慎。要为执行绑定到按钮的操作提供备选方式。例如，如果要覆写音量控制，将其用作向上翻页/向下翻页键，要确保还有其他备用方式供用户移页面。

37.3　有保障的市场

正如本章简介部分所述，Android是开源的。具体地说，它在Apache Software Lisence 2.0下，通常都可用。该许可对手机制造商的限制很少。因此坦白地讲，一个手机制造商很有可能会开发一个不能很好地运行Android的手机。它可能对于手机上自带的标准应用程序来说效果很好，但是不善于处理第三方应用程序，比如你自己编写的应用程序。

为解决该问题，Google有一些应用程序没有作为开源项目发布，比如Android Market。虽然

手机制造商可使用这些应用程序，但首先要对运行 Android Market 的手机进行测试，确保能提供不错的用户体验。

一位 Google 工程师引用了一个案例，一家手机制造商正在准备推出一部拥有 QVGA 屏幕的手机，在 Android 1.6 发布之前，QVGA 支持被正式添加到了平台上。虽然制造商安排了内置应用程序在较低分辨率的屏幕上兼容运行，但是第三方应用程序却是一团糟。Google 显然拒绝向该手机的制造商提供 Android Market。

因此，在一个手机上安装 Android Market，除了为应用程序提供一个发布方法，还意味着认同手机支持编写良好的第三方应用程序。

37.4 细枝末节

遗憾的是，Android Market 不保证在支持 Market 的手机上进行部署时不出现问题，也不阻止制造商在不检查 Market 的情况下交付 Android 手机。手机不可避免地会有一些对应用程序有负面影响的怪癖或特性。下面介绍一些 Android 手机（按照其公众可及性排列）和它们与更标准手机的区别。

37.4.1 Archos 5 Android Internet Tablet

Archos 5 Android Internet Tablet 是首个纯粹基于 Android 开源项目的主流手机。不同于 HTC、Motorola 和其他手机，Archos 5 不是一款 Google Experience 手机，且没有 Android Market、Google Maps 或其他专有 Google 应用程序。

Archos 5 是一种 WVGA 设备，但附带了 Android 1.5。因此，原始的 Archos 5 不会支持第 36 章中描述的新的 `-large` 资源集定义。鉴于该手机没有批量销售，在 Archos 5 支持 Android 1.6 之前，你可能仅有一个未优化的 UI。

Archos 5 的触摸屏是电阻式的，而非电容式的。这意味着用户要使用手指甲或光标笔操作屏幕，而非指尖。在设计手指友好的 UI 时要记住这一点。

Archos 5 与固件 1.1.01 一样，为 `ANDROID_ID`（赋给每个 Android 手机的一个唯一 ID）返回一个无效的值。`ANDROID_ID` 在模拟器中为 `null`，且在手机中应该是一个十六进制字符串。在 Archos 5 上，`ANDROID_ID` 是非 `null`、非十六进制字符串。如果你只在乎它是 `null` 还是非 `null`，那么 Archos 5 一切正常；如果需要一个 `ANDROID_ID` 的十六进制值，就会遇到一些问题。

由于 Archos 5 不是一种手机，所有与手机相关的特性，比如通过 `ACTION_DIAL` 拨号，都是不可用的。类似地，由于 Archos 5 没有照相机，所有与照相机相关的特性都不可用。如前所述，Archos 5 没有 Google Maps、Android Market 和其他专有 Google 应用程序。

另外，Archos IMEI 值是虚假的，因为它根本不是一个手机。

37.4.2 Motorola CLIQ/DEXT

Motorola CLIQ（或美国以外所称的 DEXT）是一种 HVGA 设备，最初装载了 Android 1.5。

CLIQ 有一个用于非触摸屏导航的 D-pad。但是，D-pad 在一个侧滑盖的 QWERTY 键盘上，就这一点而论，在手机处于纵向模式时 D-pad 不可用，除非你通过清单文件为活动强制使用纵向模式，或强制用户使用键盘滑出的 CLIQ。不要编写假定 D-pad 总是可用的应用程序。

CLIQ 还装有 MOTOBLUR，即 Motorola 的一个社交媒体表示层。这意味着，主应用程序、联系人和 Android 通常附带的其他功能已被 MOTOBLUR 的特定功能所取代。如果你坚持使用 SDK，这应该不会导致太多问题。有趣的一点是，并非所有 MOTOBLUR 联系人都能通过 Android Contacts 内容提供程序使用。例如，Facebook 联系人对于 MOTOBLUR 可用，但对第三方应用程序不可用，可能是许可造成的。使用 Android 2.0.1 和更高版本将 CLIQ 更新为新的 `ContactsContract` 系统时，这种情况可能会改变。

37.4.3 Motorola DROID/Milestone

Motorola DROID（或美国以外所称的 Milestone）是一种 WVGA854 设备，最初装载了 Android 2.0，不过现在这些手机大部分都运行 Android 2.0.1。

DROID 与 CLIQ 一样，在侧滑盖键盘上有一个 D-pad，因此该设备处于纵向模式时，D-pad 不能供用户随意使用。

因为 DROID 在正常手机大小的设备上有一个 WVGA854 屏幕，Android 会考虑让 DROID 有一个高密度屏幕，因此将用到 `-hdpi` 资源集。

37.4.4 Google/HTC Nexus One

由 HTC 制造并由 Google 销售的 Nexus One 是一种 WVGA800 设备，最初装载了 Android 2.1。

与 DROID 一样，Nexus One 将是一个高密度（`-hdpi`）手机。

37.4.5 Motorola BACKFLIP

Motorola BACKFLIP 是采用其他方法的定位手机。BACKFLIP 不使用轨迹球或 D-pad，而是有两个非触摸屏导航选项：

- QWERTY 键盘含有 PC 样式的箭头键，能生成标准 DPAD 键事件；
- 位于触摸屏背面的 BACKFLIP 触摸板将生成轨迹球事件（或者如果不使用轨迹球事件，将生成 DPAD 键事件）。

第 38 章 处理平台变更 38

Android 在接下来几年中将继续快速发展，可能改变速度会适当地减慢一些。不过此时此刻，你应该假定每隔 6 到 12 个月就会有重大的 Android 版本发布，而且可能的 Android 硬件阵容也会不断变化。因此，虽然目前 Android 关注的是手机，不久你就会看到 Android 上网本、Android 平板电脑、Android 媒体播放器，等等。

大部分变化对现有代码的影响不会很大。不过，有些变化必然会驱使新一轮的应用程序测试，可能还要基于测试结果改变这些应用程序。

本章介绍随着 Android 的发展，将来可能会带来问题的许多领域，并提供一些相关的处理建议。

38.1 品牌管理

在编写本书时，已发布的 Android 手机是 Google Experience 手机。这意味着它既有可在模拟器中找到的标准 Android 界面，又有像 Google Maps 和 Gmail 这样的附加应用程序。因此，制造商获准将“with Google”品牌标示在手机上。但是并非所有手机都会这样。

有些制造商会以 Android 为基础，更改其中包含的内容，添加他们自己的一些应用程序，甚至更改外观（菜单图标、主屏幕结构等）。

其他人可能使用完全来自开源库的 Android，因此他们可能使用标准的外观，进而使其设备缺少商业附加应用程序。

甚至在今天，有些手机根据其发布地域都有不同的应用程序组合。使用 T-Mobile G1 的美国人会发现手机上有一个 Amazon MP3 存储应用程序，但并非所有国家用户都会得到该程序。

如果应用程序与以上内容无关，那么它应该可以在任何地方运行。但是，如果应用程序代码或文档以存在 Google Maps、Gmail、Amazon MP3 存储为前提，那么可能会遇到问题。一定要在这些应用程序不可用的环境中对其进行全面的测试。

38.2　让人头疼的更多问题

上一节提到的大部分条目将重点放在硬件变更上。现在，让我们研究一些方法，解决当操作系统本身发生变化时 Android 面临的一些困难。

38.2.1　视图层次结构

Android 的设计宗旨不是处理任意复杂的视图层次结构。此处的视图层次结构是指容器与部件的层级关系，即容器中包含容器，后者包含容器，最后的容器中包含部件。

第 35 章介绍的 Hierarchy Viewer 程序很好地描述了这种视图层次结构，如图 38-1 所示。在该例中，你会看到一个 5 层深的层次结构，因为最长的容器和部件链是 5（从 `PhoneWindow$DecorView` 一直到 `Button`）。

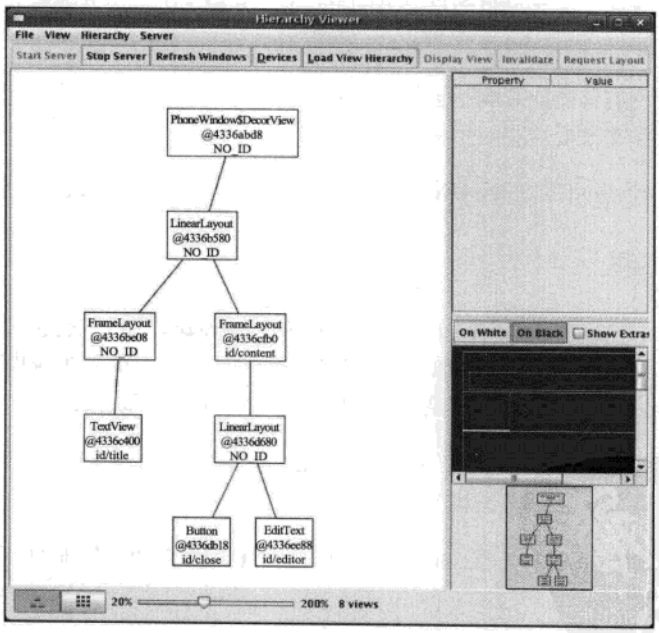


图 38-1　Hierarchy Viewer 布局视图

Android 对于视图层次结构的深度一直都有限制。不过在 Android 1.5 中，这个限制数减小了，因此在 Android 1.1 中运行良好的某些应用程序，在新版 Android 中会出现 `StackOverflowException` 并造成崩溃。当然，对于从来没有意识到视图层次结构深度会有问题，因而受到该变化困扰的开发人员来说，这让人感到很沮丧。

这里要吸取的教训如下所示。

- 保持较小深度的视图层次结构。一旦使用两位数深度，可能很快就会耗尽栈空间。
- 如果遇到 StackOverflowException，且栈跟踪看起来就在绘制部件中间的某个位置，那么你的视图层次结构可能太复杂了。

38.2.2 变更资源

核心 Android 团队可能会随 Android 的升级而变更资源，而这可能会在应用程序中产生意想不到的效果。例如在 Android 1.5 中，他们改变了常用 Button 的背景，以容纳较小的按钮。不过，依赖于之前较大最小尺寸按钮的应用程序最终会中断，且需要一些 UI 调整。

类似地，应用程序可以重用 Android 内部完全可用的公共资源，如图标。虽然这么做可以节省一些存储空间，但很多这些资源必须是公共的，且不被看作是 SDK 的一部分。例如，硬件制造商可能会更改图标来适应一些备用的 UI 外观。依赖于现有资源总是看起来一样有点危险。最好将这些资源从 Android 开源项目（http://source.android.com/）中复制到你自己的代码库中。

38.3 处理 API 变更

核心 Android 团队在保持 API 稳定不变这个方面一直做得很好，且在变更 API 时会支持一个过时模型。在 Android 中，过时并不意味着不再使用，而是不推荐继续使用。当然，每次 Android 更新都有新的 API 发布。对 API 的变更会随每次发布通过一个 API 差异报告记录在案。

遗憾的是，Android Market（Android 应用程序的基本发布渠道）仅允许你为每个应用程序上传一个 APK 文件。因此，你需要这个 APK 文件能处理尽可能多的 Android 版本。大部分时候，代码“运行良好”，不需要变更。不过有时候，还是需要做一些调整，特别是想支持新版本上的新 API 而不中断旧版本的使用时。让我们研究一下处理这些情况的方法。

38.3.1 检测版本

如果只是想基于版本在代码中采用不同的分支语句，最简单的方法就是检查 android.os.VERSION.SDK_INT。创建 AVD 并在清单文件中指定 API 级别时，这个公共静态整型值将反映同一 API 级别。因此，你可以比较该值与其他值，如 android.os.VERSION_CODES.DONUT，确定是否在 Android 1.6 或更高版本上运行。

38.3.2 包装 API

只要你尝试使用的 API 在支持的所有 Android 版本上都存在，设置代码分支可能就足够了。当 API 发生变更时就会变得很麻烦，比如有新的方法参数、新方法，甚至新类。不管是什么版本

的 Android，你都需要让代码运行，但同时要尽量利用新 API 的优势。

有一种推荐技巧可处理这种情况：反射，加上一点缓存。

例如在第 8 章中，我们使用 getTag()和 setTag()将一个任意对象与 View 关联起来。具体来说，我们使用了它们来关联一个会延迟查找所有必需部件的包装对象。你还学习了编入索引的 getTag()和 setTag()的新版本，它们以一个资源 ID 作为参数。

不过，这些编入索引的新方法在 Android 1.5 上不存在。如果你想使用这个新技术，就需要等待，直到你愿意支持 Android 1.6 或更高版本，否则就需要使用反射。在 Android 1.5 上，你可以关联一个 ArrayList<Object>作为标记，并拥有接受索引的自己的 getTag()/setTag()对。

这看起来相当直观，因此让我们看一下 APIVersions/Tagger。我们的 Activity 有一个简单布局，仅带一个 TextView：

```
<?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/test"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

Tagger 活动的源代码着眼于我们正在运行的 API 版本，且将 getTag()和 setTag()操作路由到本机编入索引的新版本（针对 Android 1.6 和更高版本），或路由到原始的未编入索引的 getTag()和 setTag()，其中我们使用 HashMap 来追踪所有单个索引对象：

```
package com.commonsware.android.api.tag;

import android.app.Activity;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import java.util.HashMap;
import java.util.Date;

public class Tagger extends Activity {
  private static final String LOG_KEY="Tagger";

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

    TextView view=(TextView)findViewById(R.id.test);
```

```
    setTag(view, R.id.test, new Date());

    view.setText(getTag(view, R.id.test).toString());
  }

  public void setTag(View v, int key, Object value) {
    if (Build.VERSION.SDK_INT>=Build.VERSION_CODES.DONUT) {
      v.setTag(key, value);
    }
    else {
      HashMap<Integer, Object> meta=(HashMap<Integer, Object>)v.getTag();

      if (meta==null) {
        meta=new HashMap<Integer, Object>();
      }

      meta.put(key, value);
    }
  }
  public Object getTag(View v, int key) {
    Object result=null;

    if (Build.VERSION.SDK_INT>=Build.VERSION_CODES.DONUT) {
      result=v.getTag(key);
    }
    else {
      HashMap<Integer, Object> meta=(HashMap<Integer, Object>)v.getTag();

      if (meta==null) {
        meta=new HashMap<Integer, Object>();
      }

      result=meta.get(key);
    }

    return(result);
  }
}
```

这看起来不错，而且如果我们构建并将其部署到 Android 1.6 或更高版本的模拟器或手机上，它运行起来很快，显示 Activity 中的当前时间。

如果我们构建并将其部署到 Android 1.5 模拟器或手机上并试图运行它，它会出现 VerifyError 错误。VerifyError 在本例中基本上是指我们引用了 Android 版本中不存在的对象。

- 我们引用了 Android 1.6 之前未引入的 SDK_INT。
- 我们引用了 getTag()和 setTag()的编入索引版本。尽管我们不会执行该代码，但类加载器仍然会尝试解析这些方法，但会以错误而告终。

因此，我们需要使用一些反射。

看一下 APIVersions/Tagger2。这是含有相同布局的同一项目，不过 Java 源代码更详尽：

```
package com.commonsware.android.api.tag;

import android.app.Activity;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Date;

public class Tagger extends Activity {
  private static final String LOG_KEY="Tagger";
  private static Method _setTag=null;
  private static Method _getTag=null;
static {
  int sdk=new Integer(Build.VERSION.SDK).intValue();

  if (sdk>=4) {
    try {
      _setTag=View.class.getMethod("setTag",
                                   new Class[] {Integer.TYPE,
                                                Object.class});
      _getTag=View.class.getMethod("getTag",
                                   new Class[] {Integer.TYPE});
    }
    catch (Throwable t) {
      Log.e(LOG_KEY, "Could not initialize 1.6 accessors", t);
    }
  }
};

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

  TextView view=(TextView)findViewById(R.id.test);

  setTag(view, R.id.test, new Date());

  view.setText(getTag(view, R.id.test).toString());
}

public void setTag(View v, int key, Object value) {
  if (_setTag!=null) {
    try {
      _setTag.invoke(v, key, value);
    }
    catch (Throwable t) {
      Log.e(LOG_KEY, "Could not use 1.6 setTag()", t);
    }
  }
  else {
    HashMap<Integer, Object> meta=(HashMap<Integer, Object>)v.getTag();

    if (meta==null) {
      meta=new HashMap<Integer, Object>();
      v.setTag(meta);
```

```
    }

    meta.put(key, value);
  }
}

public Object getTag(View v, int key) {
  Object result=null;

  if (_getTag!=null) {
    try {
        result=_getTag.invoke(v, key);
      }
      catch (Throwable t) {
        Log.e(LOG_KEY, "Could not use 1.6 getTag()", t);
      }
    }
    else {
      HashMap<Integer, Object> meta=(HashMap<Integer, Object>)v.getTag();

      if (meta==null) {
        meta=new HashMap<Integer, Object>();
        v.setTag(meta);
      }

      result=meta.get(key);
    }

    return(result);
  }
}
```

首先，最初加载类时，静态初始化例程运行。这里使用旧的 SDK String，而非新的 SDK_INT 整数，据此可以看出我们在运行哪个版本的 Android。如果使用 Android 1.6 或更高版本，我们使用反射来尝试查找编入索引的 getTag()和 setTag()方法，并且将这些结果进行缓存。由于这些方法在应用程序的生命周期内不改变，用静态变量缓存它们是安全的。

其次，真正要使用 getTag()和 setTag()时，我们查看缓存的 Method 对象是否存在（是否为 null）。如果它们为 null，那么我们需要使用这些方法的旧版本。如果 Method 对象存在，我们就使用它们，以充分利用本机索引版本的优势。

此版本的应用程序在 Android 1.5 及其更高版本上运行良好。Android 1.6 及其更高版本使用内置的索引方法，而 Android 1.5 使用虚假版的索引方法。

使用基于 Method 的反射会稍微增加一点开销，但在某些情况下还是值得的，比如要访问存在于新版 Android 中的 API，而不仅仅局限于使用旧的 API 时。甚至在所有类都是新版 Android 的新增内容时也有办法使用这个技巧（参见 http://android-developers.blogspot.com/2009/04/backward-compatibility-for-android.html）。

第 39 章 未来何去何从

很明显，本书无法涵盖所有内容。你的主要资源（除本书之外）是 Android SDK 文档，不过可能还需要更多信息。

在线搜索“android”和一个类名是找到给定 Android 类相关教程的一种好方式。但是要记住，2008 年 8 月末之前编写的教程可能是针对 M5 SDK 编写的，这在当前 SDK 中使用时需要做大量调整工作才行。

除了随机搜索教程之外，你可以使用本章概括的一些资源。

39.1 问题——部分答案

获得 Android 官方援助的地方是 Android Google Groups。至于 SDK，有 3 点需要考虑：

- 对于 Android 初学者，这是一个询问入门级问题的好地方；
- 对于 Android 开发人员，最适合询问较复杂的问题或钻研 SDK 较少使用的部分；
- Android 讨论，可进行与 Android 相关的任何讨论，不一定是编程问题和解答。

你可能还应考虑以下资源：

- http://anddev.org 上的 Android 教程和编程论坛；
- 针对 Android 的 Open Mob wiki（http://wiki.andmob.org/）；
- Freenode 上的#android IRC 渠道；
- StackOverflow 的 android 和 android-sdk 标记；
- 有关 JavaRanch 的 Android 管理委员会。

39.2 源代码

目前还提供 Android 的源代码。这主要针对想增强、改进或探究 Android 操作系统内部细节

的人们。但是你有可能在代码中找到寻求的答案，特别是在你想了解内置 Android 组件的工作方式时。

源代码和相关资源可在 http://source.android.com 上找到，你可以在该网站上执行以下操作：

- 下载或浏览源代码；
- 对操作系统本身提供 bug 报告；
- 提交补丁并了解评估和核准该补丁的流程；
- 加入一个独立的有关 Android 平台开发的 Google Groups 组。

你可能不想下载数千兆字节的 Android 源代码快照，而是想使用 Google Code Search (http://www.google.com/codesearch)。只需在搜索查询中添加 `android:package` 约束，它就会只在 Android 中搜索相关项目。

39.3 获得最新的信息

Ed Burnette 是一个自己编写 Android 书籍的好人，同时也是 Planet Android (http://www.planetandroid.com/)的经理，Planet Android 是大量 Android 相关博客的一个提要聚合器。订阅 planet 的提要就可以监控 Android 相关博客的发布，但内容不仅仅与编程有关。

要关注更多与编程相关的 Android 博客发布，你可以在 DZone 上搜索“android”并基于该搜索结果订阅一个提要。

[General Information]
书名=ANDROID开发入门教程
作者=（美）墨菲著
页数=293
出版社=北京市：人民邮电出版社
出版日期=2010.12
SS号=12719599
DX号=000006990007
URL=http://book1.duxiu.com/bookDetail.jsp?dxNumber=000006990007&d=066EC658202BE334DEF9141BE8CAFEFD