疯狂Android讲义（第3版）

李 刚 编著

Publishing House of Electronics Industry
北京•BEIJING

疯狂Android讲义（第3版）

第1章 Android应用和开发环境

本章要点

Android手机平台的发展与现状

Android手机平台的架构与特性

搭建Android应用的开发环境

管理Android虚拟设备

使用Android模拟器

安装和使用Genymotion模拟器

调试工具Monitor的用法

使用ADB工具复制文件、安装APK等

使用Android Studio开发Android应用

手动开发Android应用

掌握Android应用的结构

自动生成Android应用的清单文件

Android应用的res目录

Android应用的程序权限

Android应用的四大组件

对Android应用程序进行签名

Android系统已经成为全球应用最广泛的手机操作系统，三星、华为、中兴等手机厂商早已通过Android阵营取得了巨大成功。目前国内对Android开发人才的需求也在迅速增长。而且搭载Android智能系统的手机越来越不像"手机"，更像是一台小型电脑。因此手机软件必将在未来IT行业中具有举足轻重的地位—你不可能带着一台电脑到处跑，而且时时开着机，但手机可以做到。从趋势上来看，对Android软件人才的需求会越来越大。

本书所介绍的平台是Android 5平台，该版本的Android平台经过几年的沉淀，不仅功能十分强大，而且十分高效、稳定。本书将会全面介绍Android平台的软件开发。但本章是全书的基础，将会简要介绍Android平台的历史、现状，重点向读者讲解如何搭建和使用Android应用开发环境，包括安装Android SDK、Android开发工具；也包括如何使用Android提供的ADB、Monitor、AAPT等工具，掌握这些工具是开发Android应用的基础技能。

1.1 Android的发展和历史

Android是由Andy Rubin创立的一个手机操作系统，后来被Google收购。Google希望与各方共同建立一个标准化、开放式的移动电话软件平台，从而在移动产业内形成一个开放式的操作平台。

1.1.1 Android的发展和简介

Android并不是Google创造的，而是由Android公司所创造的，该公司的创始人是Andy Rubin。该公司后来被Google收购，而Andy Rubin也成为Google公司的Android产品负责人。

Google于2007年11月5日发布了Android 1.0手机操作系统，这个版本的Android系统还没有赢得广泛的市场支持。

2009年5月，Google发布了Android 1.5，该版本的Android提供了一个非常"豪华"的用户界面，而且提供了蓝牙连接支持。这个版本的Android吸引了大量开发者的目光。接下来，Android的版本更新得较

快，目前最新的Android版本是5.0，这也是本书所介绍的Android版本。

目前Android已经成为一个重要的手机操作系统。当前市场上常见的手机操作系统有如下这些。

➢ **iOS**：Apple公司的手机、平板操作系统，市场占有率较高。

➢ **Windows Phone**：Microsoft公司的手机操作系统，2014年发布的最新版本为Windows Phone 8.1，但局势依然不够明朗，前途依然堪忧。

➢ **BlackBerry**：即将被淘汰。

➢ **Symbian**：已被彻底淘汰，彻底退出历史舞台。

目前Android系统的市场占有率已经远超iOS，而Windows Phone作为Microsoft公司最后的"赌注"，自然也是全力以赴，希望至少能与iOS、Android三足鼎立，但目前局势似乎并不乐观。无论从哪个角度来看，Android已经成为最主流的手机操作系统。

提示：

事实上，Android已经超出了手机操作系统的范畴，Android系统已经广泛应用于TV、手表以及各种可穿戴设备。最新发布的Android 5.0已经专门提供了TV、Wear等系统镜像。

就目前国内环境来说，已有大量手机厂商开始生产Android操作系统的手机，因为Android手机平台是一个真正开放式的平台，无须支付任何费用即可使用。出于节省研发费用的考虑，不管是对于知名的手机生产厂商，还是大量的山寨手机厂商，Android操作平台都是一个不错的选择。

从2008年9月22日，T-Mobile在纽约正式发布第一款Android手机—T-Mobile G1开始，Android系统不断地获得各个手机厂商的青睐。

2010年1月7日，Google在其美国总部正式向外界发布了旗下首款合作品牌手机Nexus One（HTC G5），同时开始对外发售。

目前，已发布搭载Android系统的手机厂商包括：三星、HTC、索尼爱立信、LG等；国内厂商如华为、联想、中兴、小米等也都开始发布搭载Android系统的手机。

1.1.2 Android 5.x平台架构及特性

Android系统的底层建立在Linux系统之上，该平台由操作系统、中间件、用户界面和应用软件4层组成，它采用一种被称为软件叠层（Software Stack）的方式进行构建。这种软件叠层结构使得层与层之间相互分离，明确各层的分工。这种分工保证了层与层之间的低耦合，当下层的层内或层下发生改变时，上层应用程序无须任何改变。

图1.1显示了Android系统的体系结构。

图1.1 Android系统的体系结构

备注：

图1.1以Android官方文档提供的附图为基础进行了修改。

从图1.1可以看出，Android系统主要由5部分组成，下面分别对这5部分进行简单介绍。

1.应用程序层

Android系统将会包含一系列的核心应用程序，包括电子邮件客户端、SMS程序、日历、地图、浏览器、联系人等。这些应用程序通常都是用Java编写的。这也是本书所介绍的主要内容：编写Android系统上的应用程序。

2.应用程序框架

前面已经提到，本书所要介绍的内容就是开发Android应用程序，当我们开发Android应用程序时，就是面向底层的应用程序框架进行的。从这个意义上来看，Android系统上的应用程序是完全平等的，不管是Android系统提供的程序，还是普通开发者提供的程序，都可以访问Android提供的API框架。

Android应用程序框架提供了大量API供开发者使用，关于这些API的具体功能和用法则是本书后面要详细介绍的内容，此处不再展开阐述。

应用程序框架除了可作为应用程序开发的基础之外，也是软件复用的重要手段，任何一个应用程序都可发布它的功能模块—只要发布时遵守了框架的约定，那么其他应用程序就可使用这个功能模块。

3.函数库

Android包含一套被不同组件所使用的C/C++库的集合。一般来说，Android应用开发者不能直接调用这套C/C++库集，但可以通过它上面的应用程序框架来调用这些库。

下面列出一些核心库。

➢ **系统 C 库：**一个从 BSD 系统派生出来的标准 C 系统库（libc），并且专门为嵌入式Linux设备调整过。

➢ **媒体库：**基于 PacketVideo 的 OpenCORE，这套媒体库支持播放和录制许多流行的音频和视频格式，以及查看静态图片。主要包括MPEG4、H.264、MP3、AAC、AMR、JPG、PNG等多媒体格式。

➢ **Surface Manager：**管理对显示子系统的访问，并可以对多个应用程序的2D和3D图层提供无缝整合。

➢ **LibWebCore：**一个全新的Web 浏览器引擎，该引擎为Android浏览器提供支持，也为WebView提供支持，WebView完全可以嵌入开发

者自己的应用程序中。本书后面会有关于WebView的介绍。

➢ **SGL**：底层的2D图形引擎。

➢ **3D libraries**：基于OpenGL ES API实现的3D系统，这套3D库既可使用硬件3D加速（如果硬件系统支持），也可使用高度优化的软件3D加速。

➢ **FreeType**：位图和向量字体显示。

➢ **SQLite**：供所有应用程序使用的功能强大的轻量级关系数据库。

4.Android运行时

Android运行时由两部分组成：Android核心库集和ART。其中核心库集提供了Java语言核心库所能使用的绝大部分功能，而虚拟机则负责运行Android应用程序。

Android 5.0以前的Android运行时由Dalvik虚拟机和Android核心库集组成，但由于Dalvik虚拟机采用了一种被称为JIT（Just-in-time）的解释器进行动态编译并执行，因此导致Android App运行时比较慢；而ART模式则是在用户安装App时进行预编译（Ahead-of-time，简称AOT）的，将原本在程序运行时的编译动作提前到应用安装时，这样使得程序在运行时可以减少动态编译的开销，从而提升Android App的运行效率。

反过来，由于ART需要在安装App时进行AOT处理，因此ART需要占用更多的存储空间，应用安装和系统启动时间会延长不少。

除此之外，ART还支持ARM、x86和MIPS架构，并且能完全兼容64位系统，Android 5.0必然会带来更好的用户体验。

5.Linux内核

Android系统建立在Linux 2.6之上。Linux内核提供了安全性、内存管理、进程管理、网络协议栈和驱动模型等核心系统服务。除此之外，Linux内核也是系统硬件和软件叠层之间的抽象层。

1.2 搭建Android开发环境

在开始搭建Android开发环境之前，笔者假定读者已经具有一定的Java编程基础，像JDK安装、环境设置、设置JAVA_HOME环境变量之类的入门知识不在本书介绍范围之内。如果读者暂时还不会这些，建议先学习Java入门知识。

下面将从Android SDK的安装开始讲起，详细说明Android开发、调试环境的安装和使用，这些内容是Android开发的基础。

1.2.1 安装Android Studio

Android Studio是Google为Android提供的官方IDE工具，Google建议广大Android开发者尽快从Eclipse+ADT的开发环境改为使用Android Studio。

提示：

如果读者仍然希望学习使用Eclipse+ADT作为开发工具的开发方式，则可以选择阅读光盘根目录下的Eclipse+ADT.pdf电子文档，那是本书第2版的关于Eclipse+ADT安装和开发的内容。总之，无论使用何种IDE工具，技术才是根本，只有真正掌握了技术本身，任何工具才能用得得心应手。

Android Studio不再基于Eclipse，而是基于IntelliJ IDEA的Android开发环境。实际上，IntelliJ IDEA一直都是一款非常优秀的Java IDE工具，只是因为IntelliJ IDEA是一款商业的IDE工具（虽然也有免费的社区交流版，但功能相当有限），因此影响了IntelliJ IDEA的广泛应用。现在，Google以IntelliJ IDEA为基础推出的Android Studio同样可以免费使用，因此具有非常大的吸引力。

下载和安装Android Studio请按如下步骤进行。

1 登录http：//developer.android.com/sdk/index.html页面，滚动到该页面的最下方，即可看到如图1.2所示的下载链接。

图1.2 下载Android Studio

2 单击“android-studio-ide-135.1641136-windows.zip”链接，即可下载得到一个android-studio-ide-135.1641136-windows.zip压缩包。

3 将android-studio-ide-135.1641136-windows.zip压缩包解压到任意盘符的根路径下，然后单击解压路径下bin目录下的studio.exe或studio64.exe文件，32位系统运行studio.exe，64位系统运行studio64.exe。运行该程序即可看到如图1.3所示的对话框。

图1.3所示的对话框询问用户是否需要导入Android Studio设置，如果以前用过Android Studio且保存了定制该IDE的设置信息，则可以选择第一个单选钮，并通过下面的文件浏览框选择Android Studio设置信息的存储位置；否则选择第二个单选钮。

4 此处选择第二个单选钮，然后单击“OK”按钮，系统显示Android Studio的加载界面，加载完成将会看到如图1.4所示的配置向导对话框。

图1.3 选择是否导入Android Studio设置

图1.4 配置向导对话框

5 图1.4所示的对话框用于引导用户从网络上下载Android SDK，选择“Custom”单选钮，然后单击“Finish”按钮，即可看到如图1.5所示的

对话框。

6 单击“Finish”按钮，即可看到开始下载Android SDK的进度对话框。单击“Show Details”按钮，即可看到如图1.6所示的对话框。

图1.5 选择Android SDK的安装路径

图1.6 下载Android SDK

7 下载完成后单击“Finish”按钮，即可完成Android Studio的安装。安装完成后将看到如图1.7所示的对话框。

由于Android Studio是基于IntelliJ IDEA的IDE工具，因此Android Studio中Project（项目）的概念与Eclipse的Project概念不同，Android Studio的项目相当于Eclipse的Workspace（工作空间），Android Studio的Module（模块）才相当于Eclipse的项目—由此可见，Android Studio的项目相当于一个工作空间，一个工作空间可包含多个模块，每个模块对应一个Android项目。因此要记住一句有些拗口的话：Andriod Studio的项目可以包含多个Android项目（模块）。

8 接下来即可单击图1.7所示对话框中的“Start a new Android Studio project”列表项来新建一个Android Studio项目。记住：Android Studio的所谓项目只是一个工作空间，与Android项目并不对应。单击“Start a new Android Studio project”会显示如图1.8所示的对话框。

图1.7 Android Studio的开始对话框

图1.8 新建项目

9 按图1.8所示方式输入新建项目所需的项目名、公司域名（Android Studio将会根据项目名、公司域名自动确定项目的包名，当然开发者也可通过右边的“Edit”链接手动编辑项目的包名）和项目存储路径，然后单击“Next”按钮，接下来即可看到如图1.9所示的选择Android SDK版本的对话框。

图1.9 选择Android SDK版本

10 根据业务需要选择SDK的最低版本要求—由于本书需要介绍Android 5.0的新功能，因此一般都会选择“API 21 Android 5.0”作为最低版本要求。选好最低的SDK版本要求之后，单击“Next”按钮，接下来即可看到如图1.10所示的添加Activity的对话框。

提示：

Activity是Android应用中最主要的应用组件，Activity在Android应用中是负责与用户交互的组件—大致上可以把它想象成传统界面编程中的窗口，读者此处无须过多掌握有关Activity的内容，本书后面会详细介绍它。

图1.10 选择添加Activity

11 选择“Add No Activity”（不添加Activity），然后单击“Finish”按钮，即可看到Android Studio正常打开的窗口。

1.2.2 下载和安装Android SDK

虽然安装Android Studio时已经附带安装了Android SDK，但最新版的Android Studio自动下载的Android SDK往往不是最新版的，正如从图1.6所看到的，Android Studio下载的Android SDK版本为r22.6.2，但

最新的Android SDK版本为r24.0.2，因此往往还需要重新下载最新的Android SDK。

Android的官方站点是http：//www.android.com，登录该站点即可下载Android SDK。下载和安装Android SDK请按如下步骤进行。

1 登录http：//developer.android.com/sdk/index.html页面，滚动到该页面下方，即可看到如图1.11所示的下载链接。

图1.11 Android SDK的下载链接

注意：

笔者使用电信ADSL无法正常访问上面的站点（该站点可能被电信封了，原因未知），如果读者也遇到这样的问题，建议设置代理服务器来访问该站点。

2 单击“android-sdk_r24.0.2-windows.zip”链接，通过该链接即可下载Android 5.0 SDK压缩包。

提示：

不同平台下载相应的SDK压缩包，不管是Windows平台还是其他平台，都只需将下载得到的压缩包解压，再配置一些环境变量即可。

3 下载完成后得到一个android-sdk_r24.0.2-windows.zip文件，将该文件解压缩到任意路径下，比如D：\盘的根路径下。解压缩后得到一个android-sdk-windows文件夹，该文件夹下包含如下文件结构。

➢ **add-ons**：该目录下存放第三方公司为 Android 平台开发的附加功能系统。刚解压缩时该目录为空。

➢ **platforms**：该目录下存放不同版本的Android系统。刚解压缩时该目录为空。

➢ **tools**：该目录下存放了大量Android开发、调试的工具。

➢ **AVD Manager.exe**：该程序是AVD（Android虚拟设备）管理器。通过该工具可以管理AVD。

➢ **SDK Manager.exe**：该程序就是Android SDK 管理器。通过该工具可以管理Android SDK。

4 启动SDK Manager.exe，即可看到如图1.12所示的窗口。

图1.12 Android SDK管理器

提示：

如果读者运行Android SDK管理器时无法看到如图1.12所示的列表，请单击该管理器上方的“Tools”菜单中的“Options”菜单项，然后在弹出的对话框中勾选以“Force”开头的复选框。并修改Windows系统目录中System32\drivers\etc目录下的hosts文件，在其中增加如下两行：

203.208.46.146 dl.google.com

203.208.46.146 dl-ssl.google.com

5 在图1.12所示窗口的列表中勾选需要安装的平台和工具，比如Android 5.0.1的工具和平台，其中Android文档、SDK Platform是必选的。如果想查看Android官方提供的示例程序，使用Android SDK的源代码，则可以勾选“Samples for SDK”和“Sources for Android SDK”两个列表项（最好将Android 5.0.1包含的所有工具都安装上，如果无须为Android T V、可穿戴设备开发应用，则可暂时不勾选以Android TV、Android Wear开头的选项）。至于是否需要安装Android早期版本的SDK，则取决于读者喜好。选中所需安装的工具之后，单击“Install××packages（其中××代表用户勾选的列表项的数量）”按钮，将看到出现如图1.13所示的窗口。

图1.13 列出将要安装的Android工具包

6 单击“Accept License”单选钮—确认需要安装所有的工具包，然后单击“Install”按钮，系统开始在线安装Android SDK及相关工具。取决于读者的网络状态及选中的工具包的数量，在线安装时间不会太短，甚至可能花费一两个小时，耐心等待即可。

7 安装完成后将可以看到在Android SDK目录下增加了如下几个文件夹。

➢ **docs**：该文件夹下存放了Android SDK开发文件和API文档等。

➢ **extras**：该文件夹下存放了Google提供的USB驱动、Intel提供的硬件加速等附加工具包。

➢ **platform-tools**：该文件夹下存放了Android平台相关工具。

➢ **samples**：该文件夹下存放了不同Android平台的示例程序。

➢ **sources**：该文件夹下存放了Android 5.0的源代码。

➢ **system-images**：该文件下存放了不同Android平台针对不同CPU架构提供的系统镜像。

8 为了在命令行窗口可以使用Android SDK的各种工具，建议将Android SDK目录下的tools子目录、platform-tools子目录添加到系统的PATH环境变量中。

成功安装了最新版本的Android SDK之后，还需要为Android Studio设置Android SDK的路径，单击“File”→“Other Settings”→“Default Project Structure”菜单，即可看到如图1.14所示的对话框。

图1.14 设置Android SDK的安装路径

在第一个文件浏览框中选择Android SDK的安装路径，然后单击“OK”按钮，为Android Studio设置Android SDK完成。如果读者还需要对Android Studio进行一些定制化设置，则可通过Android Studio的“File”→“Settings”菜单打开如图1.15所示的对话框。

图1.15 设置Android Studio

1.2.3 安装运行、调试环境

Android程序必须在Android手机上运行，因此Android开发时必须准备相关的运行、调试环境。准备Android程序的运行、调试环境有如下3种方式。

➢ 条件允许，优先考虑购买Android真机（真机调试的速度更快、效果更好）。

➢ 配置Android虚拟设备（即AVD）。

➢ 使用第三方提供的Genymotion模拟器。

1.使用真机作为运行、调试环境

使用真机作为运行、调试环境时，只要完成如下3步。

1 用USB连接线将Android手机连接到电脑上。

2 在电脑上为手机安装驱动，不同手机厂商的Android手机的驱动略有差异，请登录各手机厂商官网下载手机驱动。

注意：

通常都需要在电脑上为手机安装驱动。可能有读者会感到疑惑：Android手机连接电脑后，电脑即可识别到Android的存储卡，不需要安装驱动啊？需要提醒读者的是，电脑仅能识别Android手机的存储卡是不够的，安装驱动才能把Android手机整合成运行、调试环境。

3 打开手机的调试模式。打开手机，依次单击“Dev Tools”→“开发者选项”，进入如图1.16所示的设置界面，在该界面中“开启”开发者选项。

图1.16 打开调试模式

按图1.16所示，勾选“Always stay awake”“USB调试”“允许模拟位置”3个选项即可。如果开发者还有其他需要，则可以勾选其他的开发者选项。

2.使用AVD作为运行、调试环境

Android SDK为开发者提供了可以在电脑上运行的“虚拟手机”，Android把它称为Android Virtual Device（AVD）。如果开发者没有Android手机，则完全可以在AVD上运行我们编写的Android应用。

创建、删除和浏览AVD之前，通常应该先为Android SDK设置一个环境变量：ANDROID_SDK_HOME，该环境变量的值为磁盘上一个已有的路径。如果不设置该环境变量，开发者创建的虚拟设备默认保存在C：\Documents and Settings\<user_name>\.android目录（以Windows XP为例）下；如果设置了ANDROID_SDK_HOME环境变量，那么虚拟设备就会保存在%ANDROID_SDK_HOME%\.android路径下。

注意：

这里有一点非常容易混淆，此处的%ANDROID_SDK_HOME%环境变量并不是Android SDK的安装目录。而学习过Java EE的读者可能都记

得JAVA_HOME、ANT_HOME等环境变量，它们都是指向自身的安装目录，但Android的%ANDROID_SDK_HOME%不是。

在图形界面下管理AVD比较简单，因为可以借助于Android SDK和AVD管理器完成，完全可以在图形用户界面下操作，比较适合新上手的用户。

1 通过Android SDK安装目录下的AVD Manager.exe启动AVD管理器，如图1.17所示。单击该管理器左边的“Android Virtual Devices”选项卡，管理器列出当前已有的AVD设备，如图1.17所示。

2 单击右边的“Create...”按钮，AVD管理器弹出如图1.18所示的对话框。

图1.17 查看所有可用的AVD设备

图1.18 创建AVD设备

3 填写AVD设备的名称、Android平台的版本和虚拟SD卡的大小，然后单击“OK”按钮，管理器即将开始创建AVD设备，开发者只要稍作等待即可。

创建完成后，管理器返回如图1.17所示的窗口，该管理器将会列出当前所有可用的AVD设备。如果开发者想删除某个AVD设备，只要在图1.17所示窗口中选择指定的AVD设备，然后单击右边的“Delete...”按钮即可。

AVD设备创建成功之后，接下来就可以使用模拟器来运行该AVD了。在Android SDK和AVD管理器中运行AVD非常简单：①在图1.17所示窗口中选择需要运行的AVD设备；②单击“Start...”按钮即可。启动后的虚拟手机如图1.19所示。

图1.19所示的就是一个运行在电脑上的虚拟手机，用过手机的读者对这个界面应该不会陌生。现在开始使用该“虚拟手机”来模拟一些“手机操作”，读者可以花点时间来熟悉一下Android系统的操作习惯。

单击Android桌面上的“程序列表”按钮，Android进入如图1.20所示的界面。

图1.19 启动后的虚拟手机

图1.20 Android应用程序列表

从图1.20所示列表中可以看到Android系统默认提供的所有可用的程序，以后我们开发的Android程序也可以在这里找到。当包含的程序太多时，可以通过手指左右拖动来查看更多的程序。

对于国内用户来说，设置中文操作界面、设置中文输入法是两个常用的操作。设置中文操作界面可通过单击图1.20所示界面中的“Settings”项来进行，依次单击“Settings”→“Language&input”→“Language”，Android系统将出现如图1.21所示的列表，选中其中的“中文（简体）”列表项，然后单击虚拟手机上的确认键返回。

提示：

为Android模拟器设置了中文操作界面之后，在有些电脑上启动、运行模拟器特别慢，慢到令人难以忍受。如果遇到这种情况，请放弃使用中文操作界面。

图1.21 设置中文操作界面

开启中文输入法通过单击图 1.20 所示界面中的“Settings”项进行设置，依次单击“Settings”→“Language&input”，在出现的列表中勾选“谷歌拼音输入法”列表项，然后单击虚拟手机上的确认键返回即可。

提示：

有时开发者启动了Android模拟器，“虚拟手机”的显示屏右上方可能提示没有网络信号，通常是因为模拟器无法访问网络的缘故。一般来说，只要运行模拟器的电脑已经处于局域网内（已接入Internet也可以），并且没有防火墙阻止Android模拟器访问网络，Android模拟器都不应该提示没有网络信号。如果运行Android模拟器的机器既不在局域网内，也没有接入Internet，则可将电脑DNS服务器设为与本机相同。例如，设置本机IP地址为192.168.1.50，再将DNS服务器地址也设为192.168.1.50即可。

3.安装Genymotion模拟器

使用Android自带的模拟器虽然简单、方便，但最大的问题就是慢，慢到让大部分开发者难以忍受，这时可以选择使用第三方模拟器：Genymotion，这个模拟器最大的特点是速度快，使用该模拟器可模拟出与真机媲美的速度。

提示：如果读者的电脑性能很好，使用Android自带的模拟器的性能还可以承受，则完全可以跳过这部分内容。

下载和安装Genymotion模拟器按如下步骤进行。

1 登录https：//www.genymotion.com/＃！/download站点，看到如图1.22所示的页面。

图1.22 下载Genymotion模拟器

2 单击“Get Genymotion”链接，浏览器打开如图1.23所示的页面。

图1.23 登录Genymotion官网

提示：Genymotion 有两个版本：免费的个人使用版和收费的商业版本。免费版的功能比较少，只能个人使用，但商业版本要收费，因此此处我们使用免费版。

在图1.23所示页面中输入用户名、密码登录Genymotion官网，如果读者暂时没有该网站的账户，则可单击该页面上的“Create Account”按钮注册一个。注册账户时需要输入一个有效的邮箱地址—注册完成后必须登录该邮箱来激活账户。

3 登录成功之后就会开始下载，下载完成后得到一个genymotion-2.3.1-vbox.exe文件，该文件就是Genymotion的安装文件，双击该文件即可开始安装。安装Genymotion与安装其他Windows程序并没有什么区别，此处不再赘述。

提示：安装Genymotion时会自动安装Oracle的VM VirtualBox，读者只要不断地单击“Next”按钮即可完成安装。

安装完成后会自动重启电脑。

4 从“开始”菜单中启动Genymotion，即可看到如图1.24所示的窗口。

5 从图1.24可以看出，此时还没有任何Genymotion模拟器，读者可通过单击该窗口上方的“Add”按钮来添加模拟器。单击“Add”按钮会再次输入用户名、密码登录系统，输入前面在Genymotion官网注册的用户名、密码，即可看到如图1.25所示的列表。

图1.24 管理Genymotion模拟器

图1.25 选择模拟器

6 在图1.25所示列表中选择一个模拟器（此处的选择并不重要，后面还需要对此处的选择进行重新设置），此处选择第一项：Custom Phone-5.0.0-API21-768x1280，然后单击“Next”按钮。系统将会显示一个文本框让用户输入该模拟器的名称，按自己喜欢随意输入一个名称，然后单击“Next”按钮。Genymotion将会开始下载文件，下载完成后将会自动完成配置，配置完成后单击“Finish”按钮即可返回图1.24所示窗口，此时将可在该窗口中看到我们刚刚配置的Android模拟器。

7 在图1.24所示窗口中右击刚刚创建的Android模拟器，在弹出的右键菜单中单击“Settings”菜单项，即可看到如图1.26所示的设置窗口。

图1.26 设置Genymotion模拟器的模拟参数

注意：

图1.26所示的内容设置了模拟器有两个CPU，内置内存为1024MB，读者可根据自己的电脑的性能进行适当的调整：如果读者的电脑的性能很好，则可以将模拟器的CPU个数、内存容量适当调大；如果读者的电脑的性能较差，则可以将模拟器的CPU个数、内存容量适当调小。

8 在图1.26所示对话框中设置完成后，单击“OK”按钮即可返回图1.24所示窗口，在该窗口的模拟器列表中就可以看到刚刚配置的模拟器。在该窗口中选择刚刚配置的模拟器，然后单击窗口上方的“Start”按钮，即可启动Genymotion模拟器。

Genymotion模拟器的启动速度比Android自带的模拟器的启动速度快，因此很快即可看到如图1.27所示的模拟器界面。

注意：

如果读者在启动Android 5.0版本的Genymotion模拟器时一直卡在黑屏界面，或者卡在Android Logo界面，则可能是因为电脑没有开启CPU虚拟化支持需要重启电脑，并进入BIOS将CPU的虚拟化支持开启。

图1.27 使用Genymotion模拟器

读者可以操作Genymotion模拟器，感受它的运行效率：这个模拟器的运行速度真的很流畅，本书后面大部分示例都是基于这个模拟器的。

1.3 Android常用开发工具的用法

前面主要介绍了Android SDK的安装，运行、调试环境的搭建，以及Android开发环境Android Studio的安装，但这些内容只是最基本的知识，要真正掌握Android开发，还必须掌握Android开发的大量辅助工具。

1.3.1 在命令行创建、删除和浏览AVD

在命令行下管理AVD需要借助于android命令（位于Android SDK安装目录的tools子目录下），如果直接执行android命令将会启动Android SDK管理器。除此之外，该命令还支持如下子命令。

➢ **list**：列出机器上所有已经安装的Android版本和AVD设备。

➢ **list avd**：列出机器上所有已经安装的AVD设备。

➢ **list target**：列出机器上所有已经安装的Android版本。

➢ **create avd**：创建一个AVD设备。

➢ **move avd**：移动或重命名一个AVD设备。

➢ **delete avd**：删除一个AVD设备。

➢ **update avd**：升级一个AVD设备使之符合新的SDK环境。

➢ **create project**：创建一个新的Android项目。

➢ **update project**：更新一个已有的Android项目。

➢ **create test-project**：创建一个新的Android测试项目。

➢ **update test-project**：更新一个已有的Android测试项目。

如果希望查看当前系统上已经安装的Android版本及已经安装的AVD设备，则运行android list或者android list avd命令即可，如图1.28所示。

图1.28 列出已经安装的Android版本和AVD设备

如果要创建一个全新的AVD设备，可执行如下命令：

在上面的create avd子命令中，只有-n和-t选项是必需的，其余的-b选项、-p选项、-s选项都是可选的。如果不设置-p选项，创建的AVD设备默认保存在%ANDROID_SDK_HOME%/.android/avd路径下。

例如，需要创建一个名为crazyit的AVD设备，则可输入如下命令：

上面的命令中21是Android 5.0的代号，如图1.28所示。执行上面的命令，系统会提醒用户是否需要定制AVD的硬件，开发者可以选择yes或no，如果输入no，即可直接开始创建AVD设备；如果输入yes，即可开始定制AVD硬件的各种选项，定制完成后系统开始创建AVD设备。

为系统创建了两个AVD之后（前面通过图形界面创建了一个），即可在%ANDROID_SDK_HOME%/.android目录下看到一个avd子目录，该子目录下包含两个文件和两个文件夹。

➢ **fkjava.avd和fkjava.ini**：fkjava AVD的基本信息和AVD设备。其中fkjava.avd目录下有一个userdata.img文件，它是AVD中用户数据的镜像。还有一个sdcard.img文件，它是该AVD所使用的虚拟SD卡的镜像。

➢ **crazyit.avd和crazyit.ini**：crazyit AVD的基本信息和AVD设备。其中crazyit.avd目录下有一个userdata.img文件，它是AVD中用户数据的镜像。

1.3.2 使用Android模拟器（Emulator）

Android模拟器就是一个运行在电脑上的“虚拟手机”。实际上，前面我们已经使用过Android模拟器了，在Android SDK和AVD管理器中选择指定的AVD设备，然后单击“Start...”按钮就是启动模拟器来运行Android系统。

在Android SDK安装目录的tools子目录下有一个emulator.exe（另外还有emulator-arm.exe、emulator-mips.exe、mulator-x86.exe），它们都是Android模拟器。这个模拟器做得十分出色，几乎可以模拟真实手机的绝大部分功能（只是速度很慢），后面我们会陆续看到—当然，它只是模拟，不要指望用模拟器与你现实中的朋友“煲电话粥”。

使用emulator.exe启动模拟器有两种用法。

➢ **emulator-avd<AVD名称>**

➢ **emulator-data 镜像文件名称**

第一种用法是运行指定的AVD设备，例如如下命令：

第二种用法是直接使用指定的镜像文件来运行AVD，例如如下命令：

1.3.3 使用Monitor进行调试

当Android应用在模拟器上运行时，我们看不到程序运行的过程，在命令行控制台也看不到程序的输出，那如何调试Android应用呢?

不用担心，Android已经为我们考虑好了这个问题。Android提供了一个Monitor工具，该工具可用于监视Android设备的运行，它是一个功能非常强大的调试环境。运行如下命令：

即可看到系统启动了如图1.29所示的窗口。

图1.29 Monitor的调试窗口

在图1.29所示的窗口中有如下几个重要的面板。

➢ **设备面板：**Monitor窗口左上角的面板，该面板会列出当前所有运行的手机（包括真机和模拟器），并列出各手机内的所有进程信息。如果需要查看指定手机或指定进程信息，则应先在该面板内选中指定手机或进程。

➢ **信息输出面板：**该面板位于Monitor窗口的下方，相当于传统Java应用控制台，因此非常重要。

➢ **线程跟踪面板：**该面板可用于查看指定进程内所有正在执行的线程状态。如果需要让该面板显示指定进程内线程的状态，则应保证下面两步：①在设备面板上选中需要查看的进程；②在设备面板上单击“Update Threads”按钮。

➢ **Heap内存跟踪面板：**该面板可用于查看指定进程内堆内存的分配和回收信息。如果需要让该面板显示指定进程内Heap的回收和分配状态，则应保证下面两步：①在设备面板上选中需要查看的进程；②在设备面板上单击“Update Heap”按钮。

➢ **模拟器控制面板：**该面板用于让模拟器模拟拨打电话、发送短信等，还可以设置模拟器的虚拟位置信息等。图1.30显示了该面板的示意图。

➢ **文件管理器面板：**该面板可用于查看Android设备所包含的文件，也可用于将Android设备的文件导出到电脑上，也可将电脑中的文件导入Android设备，如图1.31所示。

图1.30 模拟器控制面板

图1.31 文件管理器面板

实际上，Android Studio已经将Monitor集成进来，在Android Studio下方可以看到如图1.32所示的面板。

图1.32 Android Studio继承的Monitor面板

1.3.4 Android Debug Bridge（ADB）的用法

Android Debug Bridge（ADB）是一个功能非常强大的工具，它位于Android SDK安装目录的platform-tools子目录下。ADB工具既可完成模拟器文件与电脑文件的相互复制，也可安装APK应用，甚至可以直接切换到Android系统中执行Linux命令。

ADB工具的功能很多，此处就几个常用的功能略做说明。

1.查看当前运行的模拟器

输入如下命令，即可查看当前运行的模拟器：

2.电脑与手机之间文件的相互复制

在默认情况下，ADB工具总是操作当前正在运行的模拟器。

如果需要将电脑文件复制到模拟器中，则可使用adb push命令：

上面的命令将电脑的D：\盘根目录下的abc.txt文件复制到手机的/sdcard/目录下。

如果需要将模拟器文件复制到电脑中，则可使用adb pull命令：

上面的命令将模拟器的/sdcard/目录下的xyz.txt文件复制到电脑的D：\盘根目录下。

3.启动模拟器的shell窗口

Android平台的内核是基于Linux的，有时开发者希望直接打开Android平台的shell窗口，这样就可以在该窗口内执行一些常用的Linux命令，如ls、mkdir、rm等。此时可考虑使用adb shell命令：

4.安装、卸载APK程序

APK程序就是Android程序的发布包。虽然我们使用Java开发了Android应用，但并不是直接将Java二进制文件复制到手机或模拟器上即可，而是需要将Android应用打包成APK包。

一旦将Android应用打包成APK包，接下来就可以通过ADB工具来安装、卸载APK程序了。

使用ADB安装APK程序的命令格式如下：

上面的命令格式指定安装<file>代表的APK包。其中-r表示重新安装该APK包；-s表示将APK包安装到SD卡上一默认是将APK包安装到内部存储器上。例如，运行如下命令即可安装test.apk包：

如果希望从Android系统中删除指定软件包，则可使用如下命令：

上面的命令格式指定删除<package>代表的APK包。其中-k表示只删除该应用程序，但保留该程序所用的数据和缓存目录。

1.3.5 使用mksdcard管理虚拟SD卡

正如前面在Android SDK和AVD管理器中所见到的，我们可以在创建AVD设备时创建一个虚拟SD卡。实际上还可以使用mksdcard命令来单独创建一个虚拟存储卡。

mksdcard命令的语法格式如下：

上面的命令格式中<size>指定虚拟SD卡的大小，<file>指定保存虚拟SD卡的文件镜像。

例如如下命令：

上面命令创建了一个大小为64MB的虚拟SD卡，该SD卡对应的镜像文件为D：\avds\.android\avd\leegang.avd\sdcard.img。

如果希望在启动模拟器时使用指定的虚拟SD卡，则在启动模拟器时增加-sdcard <file>选项，其中<file>代表虚拟SD卡的文件镜像。例如如下命令：

到此为止，我们已经成功地安装了Android SDK、配置了Android开发环境，并且对Android相关开发工具都有了一个大致的了解，接下来正式开始Android应用开发。

1.4 开始第一个Android应用

无须担心，Android应用的开发十分简单！Android应用程序建立在应用程序框架之上，所以Android编程就是面向应用程序框架API编程—这种开发方式与编写普通的Java SE应用程序并没有太大的区别，只是Android新增了一些API而已。

1.4.1 使用Android Studio开发第一个Android应用

使用Android Studio开发Android应用非常方便，因为Android Studio会为我们自动完成许多工作。使用Android Studio开发Android应用大致需要如下3步。

1 创建一个Android项目或Android模块。

2 在XML布局文件中定义应用程序的用户界面。

3 在Java代码中编写业务实现。

上面3个步骤只是最粗粒度的归纳。下面以开发一个HelloWorld应用为例来介绍Android开发。详细步骤如下。

1 通过Android Studio主菜单的“File”→“New Project...”菜单项，创建一个Android Studio项目。Android Studio弹出如图1.8所示的对话框。

2 在图1.8所示窗口中输入项目名、公司域名、包名和项目存储路径，然后单击“Next”按钮，接下来即可看到如图1.9所示的选择Android SDK版本的对话框。

提示：Android应用的包名非常重要，Android应用的包名可作为Android应用的唯一标识。

3 在图1.9所示对话框中根据业务需要选择SDK的最低版本要求，此处会选择“API 21 Android 5.0”作为最低的版本要求。选好最低的SDK版本要求之后，单击“Next”按钮，接下来即可看到如图1.10所示的添加Activity的对话框。

4 在图1.10所示对话框中选择“Blank Activity”（空Activity，这是Android应用中最简单的Activity），然后单击“Next”按钮，即可看到如图1.33所示的对话框。

图1.33 为创建Activity设置信息

5 单击“Finish”按钮，Android Studio即成功创建了一个项目，该项目包含一个Android应用。

提示：虽然Android Studio项目可以包含多个Android应用，但本书为了方便读者浏览光盘项目，将会为每个Android应用都创建一个Android Studio项目。

Android项目创建完成后将看到如图1.34所示的项目结构。

6 Android项目的layout目录下有一个activity_main.xml文件，该文件用于定义Android应用用户界面。在Android Studio工具中打开该文件，将看到如图1.35所示的界面。

图1.34 Android项目结构

图1.35 Android Studio提供的界面设计工具

图1.35所示的"所见即所得"的设计界面十分简单，有过网页编辑经验的读者可能对Dreamweaver之类的工具比较熟悉，那么可以把这个界面近似地当成Dreamweaver。

从图1.35所示界面的控件面板中向程序拖入一个Button控件（按钮），再切换到源代码编写界面，将activity_main.xml文件（该文件就是MainActivity配套的XML布局文件）修改为如下形式。

程序清单：
\codes\01\1.4\HelloWorld\app\src\main\res\layout\activity_main.xml

上面XML文档的根元素是RelativeLayout，它代表一个相对布局，在该界面布局里包含如下两个UI控件。

➢ **TextView**：代表一个文本框。

➢ **Button**：代表一个普通按钮。

在Android用户界面设计中，各种界面布局元素将会在后面进行详细介绍，不同UI组件也会在后面进行详细介绍。此处只想说明UI组件上的

几个通用属性。

➢ **android：id：**指定该控件的唯一标识，在Java程序中可通过findViewById（"id"）来获取指定的Android界面组件。

➢ **android：layout_width：**指定该界面组件的宽度。如果该属性值为 match_parent，则说明该组件与其父容器具有相同的宽度；如果该属性值为 wrap_content，则说明该组件的宽度取决于它的内容一能包裹它的内容即可。

➢ **android：layout_height：**指定该界面组件的高度。如果该属性值为match_parent，则说明该组件与其父容器具有相同的高度；如果该属性值为 wrap_content，则说明该组件的高度取决于它的内容一能包裹它的内容即可。

可能有读者感到奇怪：Android怎么采用XML文件来定义用户界面呢？或者有过Swing编程经验的读者感到不习惯：怎么不是在Java代码里定义用户界面，而是在XML文档里定义用户界面呢？

大家要接受Android的这种优秀设计。Android把用户界面放在XML文档中定义，就可以让XML文档专门负责用户UI设置，而Java程序则专门负责业务实现，这样可以降低程序的耦合性。对于不习惯这种方式的读者来说，其实可以近似地把activity_main.xml文件当成一个HTML页面一它们都通过标记语言来定义用户界面。区别在于HTML页面使用HTML标签，而activity_main.xml文件则使用Android标签。

7 Android Studio项目下的app\src目录是Android项目的源代码，该目录的main\java\org\crazyit\helloworld目录下有一个MainActivity.java文件，它就是Android项目的Java文件。打开该文件，将该文件编辑为如下形式。

程序清单：
codes\01\1.4\HelloWorld\app\src\main\java\org\crazyit\helloworld\MainActivity.java

对于一个有不错的Java基础的读者来说，上面这个程序十分简单，它只做了如下两件事情。

1 设置该Activity使用activity_main.xml文件定义的界面布局作为用户界面。

2 定义了一个clickHandler（）方法作为按钮的事件处理方法—在处理方法中改变ID为R.id.show的文本框的内容。

至此，这个HelloWorld级的Android应用已经开发完成了。

提示：

前面介绍hello_world.xml文件时说过，读者可以把该文件当成一份HTML代码，Java程序中通过findViewById（）方法即可获取指定ID的界面控件，实际上读者完全可以把findViewById（）类比成JavaScript代码中的getElementById（）。

1.4.2 通过Andorid Studio运行Android应用

通过Andorid Studio来运行Android应用非常简单，只要如下两步即可。

1 运行Genymotion模拟器（或通过Android提供的AVD管理器或直接使用emulator命令运行指定的AVD虚拟机，但Android自带的虚拟机比较慢）。如果打算用真机作为运行、调试环境，则使用USB线连接手机，并打开手机的调试模式。

2 在Android Studio的工具条中选中Android app，然后单击工具条中的“运行”按钮，如图1.36所示。

单击“运行”按钮之后，Android Studio将会弹出如图1.37所示的对话框，该对话框询问用户要将该Android应用部署到哪个设备上?

图1.36 运行Android应用

图1.37 选择要将Android应用部署到哪个设备上

在图1.37所示对话框中选择要部署Andriod应用的设备，然后单击“OK”按钮，Android Studio将会把该应用部署到指定设备上。此时打开“手机”，进入程序列表，即可看到刚刚开发的HelloWorld程序，如图1.38所示。

单击“HelloWorld”程序项，即可在“手机”上运行刚刚开发的HelloWorld程序，运行效果如图1.39所示。

图1.38 安装成功的Android应用

图1.39 成功运行Android应用

1.5 Android应用结构分析

使用Android Studio开发Android应用简单、方便，除了创建Android项目，开发者只需做两件事情：使用activity_main.xml文件定义用户界面；打开Java源代码编写业务实现。但对于一个喜欢“穷根究底”的学习者来说，这种开发方式不免让其迷惑：

➢ findViewById（R.id.show）；代码中的R.id.show是什么？从哪里来?

➢ 为何 setContentView（R.layout.activity_main）；代码设置使用 activity_main.xml 文件定义的界面布局?

……

实际上，喜欢"穷根究底"的学习者才是真正好的学习者。借用拿破仑的一句话：不想当将军的士兵不是好士兵。类似地，也可以说：不喜欢探究原理、机制的程序员不是好程序员。

每次看到那些局限于特定IDE工具，只能在特定IDE里做事，而且自我感觉良好的学习者、工作者，笔者都忍不住十分遗憾：程序员，又少了一个。

下面将带领大家"徒手"开发一个Android项目，这样所有的事情都是由开发者自己完成的一自然对Android开发的每个细节更加熟知，以后用IDE工具开发才能做到不仅知其然，也知其所以然。

1.5.1 创建一个Android应用

前面介绍android命令时已经提到，该命令有一个create project子命令，该子命令可用于"手动"创建一个Android应用，在命令行窗口输入如下命令：

提示：

在上面的命令中，-n选项指定创建项目的名称；-t选项指定项目针对的Android平台；-p选项指定该项目的保存路径；-k选项指定该项目的包名；-a选项指定Activity的名称。

运行上面的命令，可以在当前目录的HelloWorld子目录下创建一个Android项目。进入该项目所在的目录，可以看到如下两个必要的文件夹：

在上面的文件结构中，res目录、src目录、AndroidManifest.xml文件是Android项目必需的。其他目录、文件都是可选的。

➢ res目录存放Android项目的各种资源文件，比如layout存放界面布局文件，values目录下则存放各种XML格式的资源文件，例如字符串资源文件：strings.xml；颜色资源文件：colors.xml；尺寸资源文件：dimens.xml。drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi这4个子目录则分别用于存放低分辨率、中分辨率、高分辨率、超高分辨率的4种图片文件。

➢ src目录只是一个普通的、保存Java源文件的目录。

➢ AndroidManifest.xml文件是Android项目的系统清单文件，它用于控制Android应用的名称、图标、访问权限等整体属性。除此之外，Andriod 应用的 Activity、Service、ContentProvider、BroadcastReceiver这4大组件都需要在该文件中配置。

除此之外，还可以在HelloWorld目录下看到一个build.xml文件，这是Android为该项目提供的一个Ant生成文件。通过该生成文件，开发者可以通过Ant来生成、安装Android项目。

使用Android Studio开发的Android项目结构也与此类似，在Android Studio项目下包含一个app目录，该目录下包含如下3个子目录。

➢ **build**：Android Studio自动生成的各种源文件（包括R.java文件也放在该目录的子目录下）。

➢ **libs**：存储Android项目所需的第三方JAR包。

➢ **src：**存储Android项目开发的各种源文件，包括各种Java源文件（放在main\java子目录下）、各种资源文件（放在main\res子目录下）和AndroidManifest.xml文件。除此之外，src目录下还包含一个androidTest子目录，该目录下存放的是Android测试项目。

提示：

Ant生成的Android项目（包括早期ADT创建的Android项目）主要对应于Android Studio创建的Android项目中app/src/目录下的main子目录。app/src/目录下的androidTest子目录则用于编写单元测试的测试用例；app目录下的libs目录则与Ant生成的Android项目（包括早期ADT创建的Android项目）下的libs目录完全相同。Android Studio创建的Android项目根目录下的其他文件和文件夹，只是该项目相关的配置文件，相当于Eclipse工作空间下的配置文件，与具体项目无关（Android Studio的项目对应于Eclipse的工作空间，Android Studio的模块才对应于Eclipse的模块）。

对于Android开发者而言，重点关注的就是两部分。

➢ src目录下（对于Android Studio项目的app\src\main\java子目录）的各种Java文件。

➢ res目录下（对于Android Studio项目的app\src\main\res子目录）的各种资源文件。

与前面介绍的Android开发相似的是，此处的开发同样是先编辑XML格式的界面布局文件，再编辑相应的Java文件。编写文件的内容与前一个示例并没有太大的区别，此处不再赘述。

编辑完成后启动命令行窗口，并转入HelloWorld目录下，执行ant help命令将可以看到如图1.40所示的输出。

图1.40 Android项目的生成文件

从图1.40可以看出，Android项目提供的build.xml文件包含了如下常用的生成target。

➢ **clean**：清除项目生成的内容—也就是恢复原来的样子。

➢ **debug**：打包一个调试用的Android应用的APK包，使用debug key进行签名。

➢ **release**：打包一个发布用的Android应用的APK包。

➢ **test**：运行测试。要求该项目必须是一个测试项目。

➢ **install**：将生成的调试用的APK包安装到模拟器上。

➢ **uninstall**：从模拟器上卸载该应用程序。

提示：

Ant是一个非常简洁、易用的项目生成工具。对于绝大部分Java开发者来说，使用Ant应该是一项最基本的技能。考虑到有些读者对Ant用法不熟，此处简略介绍Ant的一些安装和使用方法。

1 登录http：//ant.apache.org/bindownload.cgi站点下载Ant最新版，笔者使用的是1.9.4，建议下载该版本。Windows平台下载*.zip压缩包，而Linux平台则下载.gz压缩包。

2 将下载到的压缩文件解压缩到任意路径，例如，本书解压缩到D：\根路径下。

3 Ant的运行需要两个环境变量。① JAVA_HOME：该环境变量应指向JDK的安装路径。如果已经成功安装了Android S tudio，则该环境变量应该已经是正确的。② ANT_HOME：该环境变量应指向Ant的安装路径。Ant的安装路径就是前面释放Ant压缩文件的路径。Ant安装路径下应该包含bin、docs、etc和lib四个文件夹。

4 Ant工具的关键命令就是%ANT_HOME%/bin路径下的ant.bat命令，如果读者希望操作系统可以识别该命令，还应该将%ANT_HOME%/bin路径添加到操作系统的PATH环境变量之中。

经过上面4个步骤，即可在命令行窗口使用ant.bat命令了，它就是Ant工具。至于Ant更详细的用法介绍，请参阅疯狂Java体系的《轻量级Java EE企业应用实战》。

先执行ant release命令来发布该项目，发布完成后看到HelloWorld目录下出现了两个子目录。

➢ **bin**：该目录用于存放生成的目标文件，如Java的二进制文件、资源打包文件（.ap_后缀）、Dalvik虚拟机的可执行性文件（.dex后缀）等。

提示：

早期的Android系统需要用Dalvik虚拟机来运行Android应用。

➢ **gen**：该目录用于保存自动生成的、位于Android项目包下的R.java文件。前面我们编写Android程序代码时多次使用了R.layout.main、R.id.show、R.id.ok……现在读者应该明白这里R是什么了，原来它是Android项目自动生成的一个Java类。接下来将会详细介绍R.java文件。

1.5.2 自动生成的R.java

打开gen\org\crazyit\helloworld目录下的R.java文件，看到如下代码。

程序清单：
codes\01\1.5\HelloWorld\gen\org\crazyit\helloworld\R.java

通过R.java类中的注释可以看出，R.java文件是由AAPT工具根据应用中的资源文件自动生成的，因此可以把R.java理解成Android应用的资源字典。

AAPT生成R.java文件的规则主要是如下两条。

➢ 每类资源都对应于R类的一个内部类。比如所有界面布局资源对应于layout内部类；所有字符串资源对应于string内部类；所有标识符资源对应于id内部类。

➢ 每个具体的资源项都对应于内部类的一个public static final int类型的Field。例如，前面在界面布局文件中用到了show标识符，因此R.id类里就包含了这两个Field；由于drawable-xxxx 文件夹里包含了ic_launcher.png 图片，因此 R.drawable 类里就包含了ic_launcher Field。

随着我们不断地向Android项目中添加资源，R.java文件的内容也会越来越多。后面还会详细介绍Android资源访问的相关内容，因此后面还会进一步说明不同资源在R.java文件中的表现形式。

1.5.3 res目录说明

Android应用的res目录是一个特殊的项目，该项目里存放了Android应用所用的全部资源，包括图片资源、字符串资源、颜色资源、尺寸资源等—后面还会进一步介绍Android应用中资源的用法，此处先对res目录的资源进行简单的归纳。

Android按照约定，将不同的资源放在不同的文件夹内，这样可以方便地让AAPT工具来扫描这些资源，并为它们生成对应的资源清单类：R.java。

以/res/value/strings.xml文件来说，该文件的内容十分简单，它只是定义了一个个的字符串常量，如以下代码所示。

程序清单：codes\01\1.5\HelloWorld\res\values\strings.xml

上面的资源文件中定义了一个字符串常量，常量的值为HelloWorld，该字符串常量的名称为app_name。一旦定义了这份资源文件之后，Android项目就允许分别在Java代码、XML文件中使用这份资源文件中的字符串资源。

1.在Java代码中使用资源

为了在Java代码中使用资源，AAPT会为Android项目自动生成一份R.java文件，R类里为每份资源分别定义了一个内部类，其中每个资源项对应于内部类里一个int类型的Field。例如，上面的字符串资源文件对应于R.java里的如下内容：

借助于AAPT自动生成R类的帮助，在Java代码中可通过R.string.app_name引用到"HelloWorld"字符串常量。

2.在XML文件中使用资源

在XML文件中使用资源更加简单，只要按如下格式来访问即可：

例如，我们要访问上面的字符串资源中定义的"HelloWorld"字符串常量，则使用如下形式来引用即可：

但有一种情况例外，当我们在XML文件中使用标识符时—这些标识符无须使用专门的资源进行定义，直接在XML文档中按如下格式分配标识符即可：

例如，使用如下代码为一个组件分配标识符：

上面的代码为该组件分配了一个标识符，接下来就可以在程序中引用该组件了。

如果希望在Java代码中获取该组件，通过调用Activity的findViewById（）方法即可实现。

如果希望在XML文件中获取该组件，则可通过资源引用的方式来引用它，语法如下：

1.5.4 Android应用的清单文件：AndroidManifest.xml

AndroidManifest.xml清单文件是每个Android项目所必需的，它是整个Android应用的全局描述文件。AndroidManifest.xml清单文件说明了该应用的名称、所使用的图标以及包含的组件等。

AndroidManifest.xml清单文件通常可以包含如下信息。

➢ 应用程序的包名，该包名将会作为该应用的唯一标识。

➢ 应用程序所包含的组件，如Activity、Service、BroadcastReceiver和ContentProvider等。

➢ 应用程序兼容的最低版本。

➢ 应用程序使用系统所需的权限声明。

➢ 其他程序访问该程序所需的权限声明。

不管是Android Studio工具还是android.bat命令，它们所创建的Android项目都有一个AndroidManifest.xml文件。但随着不断地进行开

发，可能需要对AndroidManifest.xml清单文件进行适当的修改。

下面是一份简单的AndroidManifest.xml清单文件。

程序清单：codes\01\1.5\HelloWorld\AndroidManifest.xml

上面这份AndroidManifest.xml清单文件中的注释已经大致说明了各元素的作用，故不再详细说明每个元素。上面的清单文件中有两处用到了资源。

➢ android：label="@string/app_name"，这说明该应用的标签（Label）为/res/value 目录下strings.xml文件中名为app_name的字符串值。

➢ android：icon="@drawable/ic_launcher"，这说明该应用的图标为/res/drawable-l/m/hdpi目录下主文件名为icon的图片。

1.5.5 应用程序权限说明

一个Android应用可能需要权限才能调用Android系统的功能；一个Android应用也可能被其他应用调用，因此它也需要声明调用自身所需要的权限。

1.声明运行该应用本身所需要的权限

通过为<manifest.../>元素添加<uses-permission.../>子元素即可为程序本身声明权限。

例如，在<manifest.../>元素里添加如下代码：

2.声明调用该应用所需的权限

通过为应用的各组件元素，如<activity.../>元素添加<uses-permission.../>子元素即可声明调用该程序所需的权限。

例如，在<activity.../>元素里添加如下代码：

通过上面的介绍可以看出，<uses-permission.../>元素的用法倒不难，但到底有多少权限呢？实际上Android提供了大量的权限，这些权限都位于Manifest.permission类中。一般来说，有如表1.1所示的常用权限。

表1.1 Android系统的常用权限

续表

1.6 Android应用的基本组件介绍

Android应用通常由一个或多个基本组件组成，前面我们看到Android应用中最常用的组件就是Activity。事实上Android应用还可能包括Service、BroadcastReceiver、ContentProvider等组件。本节先让读者对这些组件建立一个大致的认识，后面的章节还会对这些组件做更详细的介绍。

1.6.1 Activity和View

Activity是Android应用中负责与用户交互的组件—大致上可以把它想象成Swing编程中的JFrame控件。不过它与JFrame的区别在于：JFrame本身可以设置布局管理器，不断地向JFrame中添加组件，但Activity只能通过setContentView（View）来显示指定组件。

View组件是所有UI控件、容器控件的基类，View组件就是Android应用中用户实实在在看到的部分。但View组件需要放到容器组件中，或者使用Activity将它显示出来。如果需要通过某个Activity把指定View显示出来，调用Activity的setContentView（）方法即可。

setContentView（）方法可接受一个View对象作为参数，例如如下代码：

上面的程序通过代码创建了一个LinearLayout对象（它是ViewGroup的子类，ViewGroup又是View的子类），接着调用Activity的setContentView（layout）把这个布局管理器显示出来。

setContentView（）方法也可接受一个布局管理资源的ID作为参数，例如如下代码：

从这个角度来看，大致上可以把Activity理解成Swing中的JFrame组件。当然，Activity可以完成的功能比JFrame更多，此处只是简单地类比一下。

提示：

实际上Activity是Window的容器，Activity包含一个getWindow（）方法，该方法返回该Activity所包含的窗口。对于Activity而言，开发者一般不需要关心Window对象。如果应用程序不调用Activity的setContentView（）来设置该窗口显示的内容，那么该程序将显示一个空窗口。

Activity为Android应用提供了可视化用户界面，如果该Android应用需要多个用户界面，那么这个Android应用将会包含多个Activity，多个Activity组成Activity栈，当前活动的Activity位于栈顶。

Activity包含了一个setTheme（int resid）方法来设置其窗口的风格。例如，我们希望窗口不显示ActionBar、以对话框形式显示窗口，都可通过该方法来实现。

1.6.2 Service

Service与Activity的地位是并列的，它也代表一个单独的Android组件。Service与Activity的区别在于：Service通常位于后台运行，它一般不需要与用户交互，因此Service组件没有图形用户界面。

与Activity组件需要继承Activity基类相似，Service组件需要继承Service基类。一个Service组件被运行起来之后，它将拥有自己独立的生命周期，Service组件通常用于为其他组件提供后台服务或监控其他组件的运行状态。

1.6.3 BroadcastReceiver

BroadcastReceiver是Android应用中另一个重要的组件，顾名思义，BroadcastReceiver代表广播消息接收器。从代码实现角度来看，BroadcastReceiver非常类似于事件编程中的监听器。与普通事件监听器不同的是，普通事件监听器监听的事件源是程序中的对象；而BroadcastReceiver监听的事件源是Android应用中的其他组件。

使用BroadcastReceiver组件接收广播消息比较简单，开发者只要实现自己的BroadcastReceiver子类，并重写onReceive（Context context，Intent intent）方法即可。当其他组件通过sendBroadcast（）、sendStickyBroadcast（）或sendOrderedBroadcast（）方法发送广播消息时，如果该BroadcastReceiver也对该消息“感兴趣”（通过IntentFilter配置），BroadcastReceiver的onReceive（Context context，Intent intent）方法将会被触发。

开发者实现了自己的BroadcastReceiver之后，通常有两种方式来注册这个系统级的“事件监听器”。

➢ 在Java代码中通过Context.registReceiver（）方法注册BroadcastReceiver。

➢ 在AndroidManifest.xml文件中使用<receiver.../>元素完成注册。

读者此处只要对BroadcastReceiver有一个大致的印象即可，本书后面的章节还会详细介绍如何开发、使用BroadcastReceiver组件。

1.6.4 ContentProvider

对于 Android 应用而言，它们必须相互独立，各自运行在自己的进程中，如果这些 Android应用之间需要实现实时的数据交换—例如，我们开发了一个发送短信的程序，当发送短信时需要从联系人管理应用中读取指定联系人的数据—这就需要多个应用程序之间进行数据交换。

Android系统为这种跨应用的数据交换提供了一个标准：ContentProvider。当用户实现自己的ContentProvider时，需要实现如下抽象方法。

➢ **insert（Uri，ContentValues）**：向ContentProvider插入数据。

➢ **delete（Uri，ContentValues）**：删除ContentProvider中指定数据。

➢ **update（Uri，ContentValues，String，String[]）**：更新ContentProvider中指定数据。

➢ **query（Uri，String[]，String，String[]，String）**：从ContentProvider查询数据。

通常与ContentProvider结合使用的是ContentResolver，一个应用程序使用ContentProvider暴露自己的数据，而另一个应用程序则通过ContentResolver来访问数据。

1.6.5 Intent和IntentFilter

Intent并不是Android应用的组件，但它对于Android应用的作用非常大—它是Android应用内不同组件之间通信的载体。当Android运行时需要连接不同的组件时，通常就需要借助于Intent来实现。Intent可以启动应用中另一个Activity，也可以启动一个Service组件，还可以发送一条广播消息来触发系统中的BroadcastReceiver。也就是说，Activity、Service、BroadcastReceiver三种组件之间的通信都以Intent作为载体，只是不同组件使用Intent的机制略有区别而已。

➢ 当需要启动一个 Activity 时，可调用 Context 的 startActivity（Intent intent）或startActivityForResult（Intent intent，int requestCode）方法，这两个方法中的Intent参数封装了需要启动的目标Activity的信息。

➢ 当需要启动一个 Service 时，可调用 Context 的 startService（Intent in tent）方法或bindService（Intent service，ServiceConnection conn，int flags）方法，这两个方法中的Intent参数封装了需要启动的目标Service的信息。

➢ 当需要触发一个BroadcastReceiver时，可调用Context的sendBroadcast（Intent intent）、sendStickyBroadcast（Intent i ntent） 或 sendOrderedBroadcast（Intent in tent，String receiverPermission）方法来发送广播消息，这三个方法中的 Intent 参数封装了需要触发的目标BroadcastReceiver的信息。

通过上面的介绍不难看出，Intent封装了当前组件需要启动或触发的目标组件的信息，因此有些资料也将Intent翻译为“意图”。实际上Intent对象里封装了大量关于目标组件的信息，本书后面还会更详细地介绍Intent所封装的数据，此处不再深入讲解。

当一个组件通过Intent表示了启动或触发另一个组件的“意图”之后，这个意图可分为两类。

➢ **显式Intent**：显式Intent明确指定需要启动或者触发的组件的类名。

➢ **隐式Intent**：隐式Intent只是指定需要启动或者触发的组件应满足怎样的条件。

对于显式Intent而言，Android系统无须对该Intent做任何解析，系统直接找到指定的目标组件，启动或触发它即可。

对于隐式Intent而言，Android系统需要对该Intent进行解析，解析出它的条件，然后再去系统中查找与之匹配的目标组件。如果找到符合条件的组件，就启动或触发它们。

那么Android系统如何判断被调用组件是否符合隐式Intent呢？这就需要靠IntentFilter来实现了，被调用组件可通过IntentFilter来声明自己所满足的条件—也就是声明自己到底能处理哪些隐式Intent。关于Intent和IntentFilter本书后面还会有进一步阐述，此处不再深入讲解。

1.7 签名Android应用程序

前面已经介绍过，Android项目以它的包名作为唯一标识。如果在同一台手机上安装两个包名相同的应用，后面安装的应用就可以覆盖前面安装的应用。为了避免这种情况发生，Android要求对作为产品发布的应用进行签名。

签名主要有如下两个作用：

➢ 确定发布者的身份。由于应用开发者可以通过使用相同包名来替换已经安装的程序，因此使用签名可以避免发生这种情况。

➢ 确保应用的完整性。签名会对应用包中的每个文件进行处理，从而确保程序包中的文件不会被替换。

通过以上介绍不难看出，Android应用签名的作用类似于现实生活中的签名。当开发者对Android应用签名时，相当于告诉外界：该应用程序是由“我”开发的，“我”会对该应用负责—因为有签名（签名有密钥），别人无法冒名顶替“我”；与此同时，“我”也无法冒名顶替别人。

提示：

在应用的开发、调试阶段，Android Studio或Ant工具会自动生成调试证书对Android应用签名，因此部署、调试前面两个示例并没有经过签名。需要指出的是，如果要正式发布一个Android应用，必须使用合适的数字证书来给应用程序签名，不能使用Android Studio或Ant工具生成的调试证书来发布。

1.7.1 使用Android Studio对Android应用签名

大部分时候，开发者会直接使用Android Studio对Android应用签名。使用Android Studio对Android应用签名的步骤如下。

1 单击Android Studio主菜单中的“Build”→“Generate Signed APK...”菜单项，Android Studio弹出如图1.41所示的对话框。

2 如果系统中还没有数字证书，则可以在图1.41所示窗口中单击“Create new...”按钮，并按图1.42所示格式填写数字证书的存储路径和密码。

图1.41 选择已有的Key store或创建新的Key store

图1.42 创建数字证书

3 填写完成后单击“OK”按钮，Android Studio返回图1.41所示的对话框，并在该对话框中使用刚刚创建的数字证书，如图1.43所示。

4 单击“Next”按钮，Android S tudio将会显示如图1.44所示的对话框，该对话框用于指定生成签名后的APK安装包的存储路径。

图1.43 使用刚刚创建的数字证书

图1.44 指定签名后的APK安装包的存储路径

单击“Finish”按钮，签名完成。Android Studio将会在指定路径下生成一个签名后的APK安装包。

上面步骤的第2步用于制作新的数字证书，一旦数字证书制作完成，以后就可以直接使用该数字证书签名了。

利用已有的数字证书进行签名，请按如下步骤进行。

1 在图1.41所示窗口中单击“Choose existing...”按钮浏览已有的数字证书。

2 浏览到已有的数字证书之后，在该对话框下面的Key store password、Key alias、Key password文本框中输入已有的数字证书对应的信息，这样将会看到如图1.43所示的效果。剩下的事情同样是单击“Next”按钮，并选择签名APK的存储路径即可。

1.7.2 使用命令对APK包签名

如果不想借助于Android Studio对Android应用程序签名，或者在某些场合下，需要对一个“未签名”的APK包进行签名，则可通过“命令”对Android应用进行手动签名。

使用命令对Android应用签名的步骤如下。

1 创建Key store库。JDK安装目录下的bin子目录下提供了keytool.exe工具来生成数字证书。在命令行窗口输入如下命令：

上面命令中各选项说明如下。

➢**-genkeypair**：指定生成数字证书。

➢**-alias**：指定生成数字证书的别名。

➢**-keyalg**：指定生成数字证书的算法。使用RSA算法。

➢**-validity**：指定生成的数字证书的有效期。

➢**-keystore**：指定所生成的数字证书的存储路径。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书keystore的密码、作者、公司等详细信息，如图1.45所示。

图1.45 生成数字证书

提示：

第1步的作用是生成属于你们公司、你的数字证书，这个步骤只要做一次即可。一旦数字证书创建成功之后，只要在该证书有效期内，就可以一直重复使用该证书。

2 如果Android项目没有错误，在Android Studio的“Build”→“Make Project”即可生成未签名的APK安装包。在Android Studio项目的app\build\outputs\apk路径下即可找到一个app-release-unaligned.apk文件，该文件就是未签名的APK安装包。

提示：

第2步的作用是生成一个未签名的APK安装包，如果本来已有这个未签名的安装包，或者该安装包是你们委托第三方公司开发的、第三方公司负责提供该未签名的安装包，那么这个步骤是可以省略的。

3 使用jarsigner命令对未签名的APK安装包进行签名。JDK安装目录下的bin子目录下提供了jarsigner.exe工具进行签名。在命令行窗口输入如下命令：

上面命令中各选项说明如下。

➢**-verbose**：指定生成详细输出。

➢**-keystore**：指定数字证书的存储路径。

➢**-signedjar**：该选项的3个参数分别为签名后的APK包、未签名的APK包、数字证书的别名。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书keystore的密码，如图1.46所示。

图1.46 执行数字签名

1.8 本章小结

本章简要介绍了Android应用开发的背景知识，包括Android是什么，它是干什么的；简要介绍了Android的发展历史及现状。读者本章需要掌握的重点是搭建、使用Android开发平台，包括下载与安装Android Studio和Android SDK；除此之外，Android SDK提供的各种小工具，如ADB、Monitor也是需要掌握的，这些内容是开发Android应用的基础。本章还介绍了一个Android的HelloWorld应用，分别讲解了通过Android Studio工具开发和不使用工具开发两种方式；通过介绍不使用

任何工具来开发Android应用，可以让读者对Android应用的程序结构更加熟悉。本章详细介绍了Android应用的AndroidManifest.xml文件，以及如何在该文件中管理程序权限。

第2章 Android应用的界面编程

本章要点

Android的程序界面与View组件

View组件与ViewGroup组件

Android控制程序界面的3种方式

通过继承View开发自定义View

Android常见的布局管理器

文本框组件：TextView和EditText

按钮组件：Button

特殊的按钮组件：RadioButton、CheckBox、ToggleButton和Switch

时间显示组件：AnalogClock与TextClock

图片浏览组件：ImageView、ImageButton、ZoomButton与QuickContactBadge

AdapterView组件与Adapter

ListView和GridView的功能与用法

ExpandableListView组件的功能与用法

Spinner和Gallery的功能与用法

AutoCompleteTextView组件的功能与用法

AdapterViewFlipper和StackView的功能与用法

ProgressBar进度条的功能与用法

SeekBar的功能与用法

RatingBar的功能与用法

ViewAnimator 和 ViewSwitch 的功能与用法

ImageSwitch、TextSwitch和ViewFlipper的功能与用法

使用Toast创建简单提示

CalendarView的功能与用法

DatePicker、TimerPicker和NumberPicker的功能与用法

SearchView的功能与用法

TabHost的功能与用法

ScrollView的功能与用法

使用Notification发送全局通知

使用AlertDialog创建对话框

使用AlertDialog创建各种复杂的对话框

具有对话框风格的窗口

使用PopupWindow创建对话框

开发选项菜单和子菜单

为菜单项提供响应

使用ActionBar显示选项菜单

为ActionBar添加ActionView

ActionBar结合Fragment实现Tab导航

ActionBar结合Fragment实现下拉式导航

Android应用开发的一项内容就是用户界面的开发。不管应用实际包含的逻辑多么复杂、多么优秀，如果这个应用没有提供友好的图形用户界面（Graphics User Interface，GUI），也将很难吸引最终用户。相反，如果为应用程序提供了友好的图形用户界面，最终用户通过手指拖动、点击等动作就可以操作整个应用，这个应用程序就会受欢迎得多（实际上，Windows之所以广为人知，其最初的吸引力就是来自于它所提供的图形用户界面）。作为一个程序设计者，必须优先考虑用户的感受，一定要让用户感到"爽"，我们的程序才会被需要、被使用，这样的程序才有价值。

Android提供了大量功能丰富的UI组件，开发者只要按一定规律把这些UI组件组合起来—就像小朋友"搭积木"一样，把这些UI组件搭建在一起就可以开发出优秀的图形用户界面。为了让这些UI组件能响应用户的手指触摸、键盘动作，Android也提供了事件响应机制，这样保证图形界面应用可响应用户的交互操作。

通过学习本章，读者应该能开发出漂亮的图形用户界面，这些图形用户界面是Android应用开发的基础，也是非常重要的组成部分。

2.1 界面编程与视图（View）组件

Android应用是运行于手机系统上的程序，这种程序给用户的第一印象就是用户界面。从市场的角度来看，所有开发者都应充分重视Android应用的用户界面。Android提供了非常丰富的用户界面组件，借助于这些用户界面组件，开发者可以非常方便地进行用户界面开发，而且可以开发出非常优秀的用户界面。

2.1.1 视图组件与容器组件

Android应用的绝大部分UI组件都放在android.widget包及其子包、android.view包及其子包中，Android应用的所有UI组件都继承了View类，View组件非常类似于Swing编程的JPanel，它代表一个空白的矩形区域。

View类还有一个重要的子类：ViewGroup，但ViewGroup通常作为其他组件的容器使用。

Android的所有UI组件都是建立在View、ViewGroup基础之上的，Android采用了“组合器”设计模式来设计View和ViewGroup：ViewGroup是View的子类，因此ViewGroup也可被当成View使用。对于一个Android应用的图形用户界面来说，ViewGroup作为容器来盛装其他组件，而ViewGroup里除了可以包含普通View组件之外，还可以再次包含ViewGroup组件。

图2.1显示了Android图形用户界面的组件层次图。

图2.1来自Android文档。对于每个Android开发者而言，Android提供的官方文档是必看的。下面简单介绍读者应该如何查看Android文档—这实际上是一种学习方法。实际上，笔者常常觉得掌握学习方法比记住几个知识点更重要。

图2.1 图形用户界面的组件层次

在第1章在线安装Android SDK组件时，通过图1.12所示窗口选择Android工具时勾选“Documentation for Android SDK”项，就会将Android文档安装到本地磁盘。一旦我们将Android文档安装到本地磁盘，就可以在Android SDK安装目录下找到docs子目录，打开docs子目录下的index.html页面，并单击该页面上方的“Develop”→“API Guides”（开发指南）标签页，用户将看到如图2.2所示的页面。

图2.2 Android开发指南

图2.2所示就是Android官方提供的开发指南文档，这份文档也是笔者当初学习、开发Android应用的重要文档。

提示：

如果具有良好的英文阅读能力，而且Java基本功扎实，学习Android完全可以不用购买任何图书，直接阅读这份API Guides也是很好的学习方法。

单击图2.2所示页面中的Reference标签页，接下来所看到的就是Android的API文档，如图2.3所示。

图2.3 Android的API文档

图2.3所示的API文档与我们熟悉的API文档大致相同，最大的区别在于Android的API文档并未在类列表区直接列出所有类，只有当开发者选择指定包之后，类列表区才会列出该包下的所有类，这给开发者带来了一些不便。建议使用Chrome、Firefox等浏览器查看这份API文档，使用IE查看这份API文档会比较慢。这份API文档与所有API文档一样，都是开发者必须时常查阅的手册。

提示：

图2.3所示页面中API Guides、Reference两个标签页的内容是Android文档的最重要内容，本书内容将会基本覆盖API Guides的内容。不仅如此，Resource标签页中也包含了Android的一些简单的入门示例，也是初学者学习Android应用开发不错的资料。

前面介绍Android应用结构时已经指出，Android推荐使用XML布局文件来定义用户界面，而不是使用Java代码来开发用户界面，因此所有组件都提供了两种方式来控制组件的行为。

➢ 在XML布局文件中通过XML属性进行控制。

➢ 在Java程序代码中通过调用方法进行控制。

实际上不管使用哪种方式，它们控制Android用户界面行为的本质是完全一样的。大部分时候，控制UI组件的XML属性还有对应的方法。

对于View类而言，它是所有UI组件的基类，因此它包含的XML属性和方法是所有组件都可使用的。表2.1所示是View类常用的XML属性、相关方法及简要说明。

表2.1 View类的XML属性、相关方法及说明

续表

续表

提示：

Drawable是Android提供的一个抽象基类，它代表了“可以被绘制出来的某种东西”，Drawable包括了大量子类，比如BitmapDrawable代表位图Drawable、ColorDrawable代表颜色Drawable、ShapeDrawable代表几何形状Drawable。各种Drawable可用于定制UI组件的背景等外观。本书第7章会详细介绍各种Drawable的功能与用法。

注意：

上面表格中介绍View组件时列出的setElevation（float）和setTranslationZ（float）方法都是Android 5.0新增的Material Design中的功能，它们的作用也基本相似，它们都可用于设置该组件垂直屏幕“浮”起来的效果，通过该属性可让该组件呈现3D效果。当该组件垂直于屏幕“浮”起来之后，组件下的实时阴影就会自动显现。

ViewGroup继承了View类，当然也可以当成普通View来使用，但ViewGroup主要还是当成容器类使用。但由于ViewGroup是一个抽象类，因此实际使用中通常总是使用ViewGroup的子类来作为容器，例如各种布局管理器。

ViewGroup容器控制其子组件的分布依赖于ViewGroup.LayoutParams、ViewGroup.MarginLayoutParams两个内部类。这两个内部类中都提供了一些XML属性，ViewGroup容器中的子组件可以指定这些XML属性。

表2.2显示了ViewGroup.LayoutParams所支持的两个XML属性。

表2.2 ViewGroup.LayoutParams支持的XML属性

android：layout_height、android：layout_width两个属性支持如下3个属性值。

➢ **fill_parent**：指定子组件的高度、宽度与父容器组件的高度、宽度相同（实际上还要减去填充的空白距离）。

➢ **match_parent**：该属性值与fill_parent完全相同，而且从Android 2.2开始就推荐使用这个属性值来代替fill_parent。

➢ **wrap_content**：指定子组件的大小恰好能包裹它的内容即可。

读者可能对布局高度与布局宽度感到疑惑：为组件指定了高度与宽度不就够了吗？为何还要设置布局高度与布局宽度呢？这是由Android的

布局机制决定的，Android组件的大小不仅受它实际的宽度、高度控制，还受它的布局高度与布局宽度控制。比如设置一个组件的宽度为30pt，如果将它的布局宽度设为match_parent，那么该组件的宽度将会被“拉宽”到占满它所在的父容器；如果将它的布局宽度设为wrap_content，那么该组件的宽度才会是30pt。

ViewGroup.MarginLayoutParams用于控制子组件周围的页边距（Margin，也就是组件四周的留白），它支持的XML属性如表2.3所示。

表2.3 ViewGroup.MarginLayoutParams支持的XML属性

后面我们还会详细介绍ViewGroup各子类的用法，此处不再详述。

2.1.2 使用XML布局文件控制UI界面

Android推荐使用XML布局文件来控制视图，这样不仅简单、明了，而且可以将应用的视图控制逻辑从Java代码中分离出来，放入XML文件中控制，从而更好地体现MVC原则。

当我们在Android应用的app\src\main\res/layout目录下定义一个主文件名任意的XML布局文件之后（R.java会自动收录该布局资源），Java代码可通过如下方法在Activity中显示该视图：

当在布局文件中添加多个UI组件时，都可以为该UI组件指定android:id属性，该属性的属性值代表该组件的唯一标识。接下来如果希望在Java代码中访问指定UI组件，则可通过如下代码来访问它：

一旦在程序中获得指定UI组件之后，接下来就可以通过代码来控制各UI组件的外观行为了，包括为UI组件绑定事件监听器等。

2.1.3 在代码中控制UI界面

虽然Android推荐使用XML布局文件来控制UI界面，但如果开发者愿意，Android允许开发者像开发Swing应用一样，完全抛弃XML布局文件，完全在Java代码中控制UI界面。如果希望在代码中控制UI界面，那么所有的UI组件都将通过new关键字创建出来，然后以合适的方式“搭建”在一起即可。

实例：用编程的方式开发UI界面

下面将试图开发一个完全用代码控制UI界面的Android应用。由于该应用完全采用代码来控制UI界面，因此可以完全抛弃XML布局文件。下面是通过代码控制UI界面的代码。**程序清单：codes\02\2.1\CodeView\app\src\main\java\org\crazyit\ui\CodeViewActivity.java**

从上面程序的粗体字代码可以看出，该程序中所用到的UI组件都是通过new关键字创建出来的，然后程序使用LinearLayout容器来“盛装”这些UI组件，这样就组成了图形用户界面。

从上面的程序代码中可以看出，无论创建哪种UI组件，都需要传入一个this参数，这是由于创建UI组件时传入一个Context参数，Context代表访问Android应用环境的全局信息的API。让UI组件持有一个Context参数，可以让这些UI组件通过该Context参数来获取Android应用环境的全局信息。

Context本身是一个抽象类，Android应用的Activity、Service都继承了Context，因此Activity、Service都可直接作为Context使用。

在模拟器中运行上面的程序，将可以看到如图2.4所示的界面。

图2.4 在代码中控制UI界面

从上面的程序代码中不难看出，完全在代码中控制UI界面不仅不利于高层次的解耦，而且由于通过new关键字来创建UI组件，需要调用方法来设置UI组件的行为，因此代码也显得十分臃肿；相反，如果通过XML布局文件来控制UI界面，开发者只要在XML布局文件中使用标签即可创建UI组件，而且只要配置简单的属性即可控制UI组件的行为，因此要简单得多。

虽然Android应用完全允许开发者像开发Swing应用一样在代码中控制UI界面，但这种方式不仅编程烦琐，而且不利于高层次的解耦，因此不推荐开发者使用这种方式。

2.1.4 使用XML布局文件和Java代码混合控制UI界面

前面已经提到，完全使用Java代码来控制UI界面不仅烦琐，而且不利于解耦；而完全利用XML布局文件来控制UI界面虽然方便、便捷，但难免有失灵活。因此有些时候，可能需要混合使用XML布局文件和代码来控制UI界面。

当混合使用XML布局文件和代码来控制UI界面时，习惯上把变化小、行为比较固定的组件放在XML布局文件中管理，而那些变化较多、行为控制比较复杂的组件则交给Java代码来管理。

实例：简单图片浏览器

例如下面的应用，我们先在布局文件中定义一个简单的线性布局容器，该布局文件的代码如下。

程序清单：
codes\02\2.1\MixView\app\src\main\res\layout\main.xml

上面的布局文件只是定义了一个简单的线性布局容器。接下来我们会在程序中获取该线性布局容器，并往该容器中添加组件。下面是该实例的程序代码。

程序清单：
codes\02\2.1\MixView\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中第1行粗体字代码获取了该Activity所显示的LinearLayout（线性布局容器），第2行和第3行粗体字代码用于创建一个ImageView，并将该ImageView添加到LinearLayout容器中—其中LinearLayout布局容器通过XML布局文件管理，而ImageView组件则由Java代码管理。

除此之外，上面的程序还为ImageView组件添加了一个单击事件。当用户单击该组件时，ImageView显示下一张图片。运行上面的程序，可以看到如图2.5所示的界面。

2.1.5 开发自定义View

图2.5 XML布局文件和Java代码混合控制布局

前面已经提到，View组件的作用类似于Swing编程中的JPanel，它只是一个矩形的空白区域，View组件没有任何内容。对于Android应用的其他UI组件来说，它们都继承了View组件，然后在View组件提供的空白区域上绘制外观。

基于Android UI组件的实现原理，开发者完全可以开发出项目定制的组件—当Android系统提供的UI组件不足以满足项目需要时，开发者可以通过继承View来派生自定义组件。

当开发者打算派生自己的UI组件时，首先定义一个继承View基类的子类，然后重写View类的一个或多个方法。通常可以被用户重写的方法如下。

➢ **构造器：**重写构造器是定制View的最基本方式，当Java代码创建一个View实例，或根据XML布局文件加载并构建界面时将需要调用该构造器。

➢ **onFinishInflate（）：**这是一个回调方法，当应用从XML布局文件加载该组件并利用它来构建界面之后，该方法将会被回调。

➢ **onMeasure（int，int）：**调用该方法来检测View组件及其所包含的所有子组件的大小。

➢ **onLayout（boolean，int，int，int，int）：**当该组件需要分配其子组件的位置、大小时，该方法就会被回调。

➢ **onSizeChanged（int，int，int，int）：**当该组件的大小被改变时回调该方法。

➢ **onDraw（Canvas）：**当该组件将要绘制它的内容时回调该方法进行绘制。

➢ **onKeyDown（int，KeyEvent）：**当某个键被按下时触发该方法。

➢ **onKeyUp（int，KeyEvent）：**当松开某个键时触发该方法。

➢ **onTrackballEvent（MotionEvent）：**当发生轨迹球事件时触发该方法。

➢ **onTouchEvent（MotionEvent）：**当发生触摸屏事件时触发该方法。

➢ **onFocusChanged （boolean gainFocus，int direction，Rect previouslyFocusedRect）：** 当该组件焦点发生改变时触发该方法。

➢ **onWindowFocusChanged（boolean）：** 当包含该组件的窗口失去或得到焦点时触发该方法。➢ **onAttachedToWindow（）：** 当把该组件放入某个窗口时触发该方法。

➢ **onDetachedFromWindow（）：** 当把该组件从某个窗口上分离时触发该方法。

➢ **onWindowVisibilityChanged（int）：** 当包含该组件的窗口的可见性发生改变时触发该方法。

当需要开发自定义View时，开发者并不需要重写上面列出的所有方法，而是可以根据业务需要重写上面的部分方法。例如，下面的示例程序就只重写onDraw（Canvas）方法。

实例：跟随手指的小球

为了实现一个跟随手指的小球，我们考虑开发自定义的UI组件，这个UI组件将会在指定位置绘制一个小球，这个位置可以动态改变。当用户通过手指在屏幕上拖动时，程序监听到这个手指动作，把手指动作的位置传入自定义UI组件，并通知该组件重绘即可。

下面是自定义组件的代码。

程序清单：
codes\02\2.1\CustomView\app\src\main\java\org\crazyit\ui\DrawView.java

上面的DrawView组件继承了View基类，并重写了onDraw方法—该方法负责在该组件的指定位置绘制一个小球。除此之外，该组件还重写

了onTouchEvent（MotionEvent event）方法，该方法用于处理该组件的触碰事件，当用户手指触碰该组件时将会激发该方法。当手指在触摸屏上移动时，将会不断地触发触摸屏事件，事件监听器中负责触发事件的坐标将被传入DrawView组件，并通知该组件重绘—这样即可保证DrawView上小球跟随手指移动而移动。

有了这个自定义组件之后，接下来可以通过Java代码把该组件添加到指定容器中，这样就可以看到该组件的运行效果。下面是该应用的Activity类。

程序清单：
codes\02\2.1\CustomView\app\src\main\java\org\crazyit\ui\CustomViewActivity.java

上面的程序中先创建了自定义组件（DrawView）的实例，然后程序将该组件添加到LinearLayout容器中。

运行上面的程序，看到结果如图2.6所示。

图2.6 跟随手指的小球

该实例依然在Java代码中创建了DrawView组件的实例，并将它添加到LinearLayout容器中，实际上完全可以在XML布局文件中管理该组件，如果我们使用如下布局文件：

上面的布局文件已经添加了自定义组件，因此Java代码中只要加载该界面布局文件即可，无须通过Java代码来添加该自定义组件。因此，Activity的代码可以简化为如下形式：

2.2 第1组UI组件：布局管理器

Android 的界面组件比较多，如果不理顺它们内在的关系，孤立地学习、记忆这些 UI 组件，不仅学习起来事倍功半，而且不利于掌握它们内在的关系。为了帮助读者更好地掌握Android界面组件的关系，本书将会把这些界面组件按照它们的关联分析，分为几组进行介绍。本节介绍的是第1组UI组件：以ViewGroup为基类派生的布局管理器。

为了更好地管理Android应用的用户界面里的各组件，Android提供了布局管理器。通过使用布局管理器，Android应用的图形用户界面具有良好的平台无关性。通常来说，推荐使用布局管理器来管理组件的分布、大小，而不是直接设置组件位置和大小。例如，通过如下代码定义了一个文本框（TextView）：

为了让这个组件在不同的手机屏幕上都能运行良好—不同手机屏幕的分辨率、尺寸并不完全相同，如果让程序手动控制每个组件的大小、位置，则将给编程带来巨大的困难。为了解决这个问题，Android提供了布局管理器。布局管理器可以根据运行平台来调整组件的大小，程序员要做的，只是为容器选择合适的布局管理器。

与Swing界面编程不同的是，Android的布局管理器本身就是一个UI组件，所有的布局管理器都是ViewGroup的子类。图2.7显示了Android布局管理器的类图。

图2.7 Android布局管理器的类图

从图2.7可以看出，所有布局都可作为容器类使用，因此可以调用多个重载的addView（）向布局管理器中添加组件。实际上，我们完全可以用一个布局管理器嵌套到其他布局管理器中—因为布局管理器也继承了View，也可以作为普通UI组件使用。

2.2.1 线性布局

线性布局由LinearLayout类来代表，线性布局有点像Swing编程里的Box，它们都会将容器里的组件一个挨着一个地排列起来。LinearLayout可以控制各组件横向排列（通过设置android：orientation属性控制），也可控制各组件纵向排列。

Android的线性布局不会换行，当组件一个挨着一个地排列到头之后，剩下的组件将不会被显示出来。

表2.4显示了LinearLayout支持的常用XML属性及相关方法的说明。

表2.4 LinearLayout的常用XML属性及相关方法

续表

LinearLayout包含的所有子元素都受LinearLayout.LayoutParams控制，因此LinearLayout包含的子元素可以额外指定如表2.5所示的属性。

表2.5 LineLayout子元素支持的常用XML属性及相关方法

提示：

基本上很多布局管理器都提供了相应的LayoutParams内部类，该内部类用于控制它们的子元素支持指定android：layout_gravity属性，该属性设置该子元素在父容器中的对齐方式。与android：layout_gravity相似的属性还有android：gravity属性（一般容器才支持指定该属性），android：gravity属性用于控制它所包含的子元素的对齐方式。

例如，定义如下XML布局管理器。

程序清单：
codes\02\2.2\LinearLayoutTest\app\src\main\res\layout\main.xml

上面的界面布局非常简单，它只是定义了一个简单的线性布局，并在线性布局中定义了5个按钮。定义线性布局时使用粗体字代码指定了垂直排列所有组件，而且所有组件对齐到容器底部并且水平居中。运行上面的程序，使用Activity显示界面布局，将看到如图2.8所示的界面。

如果将上面布局文件中的android:gravity="bottom|center_horizontal"改为android:gravity="right|center_vertical"—也就是所有组件水平右对齐、垂直居中，再次使用Activity显示该界面布局，将看到如图2.9所示的界面。

图2.8 垂直的线性布局，底部、居中对齐

图2.9 垂直的线性布局，水平居右、垂直居中对齐

图2.10 水平的线性布局，顶端对齐

如果我们将上面的线性布局的方向改为水平，也就是设置android:orientation="horizontal"，并设置gravity="top"，再次使用Activity来显示该界面布局，将看到如图2.10所示的界面。

从图2.10所示的运行结果可以看出，当采用线性布局来管理5个按钮组件时，如果这5个按钮组件无法在一行中同时显示出来，但LinearLayout不会换行显示多余的组件，因此后面可能有些按钮会显示不出来。

2.2.2 表格布局

表格布局由TableLayout所代表，TableLayout继承了LinearLayout，因此它的本质依然是线性布局管理器。表格布局采用行、列的形式来管理UI组件，TableLayout并不需要明确地声明包含多少行、多少列，而是通过添加TableRow、其他组件来控制表格的行数和列数。

每次向TableLayout中添加一个TableRow，该TableRow就是一个表格行，TableRow也是容器，因此它也可以不断地添加其他组件，每添加一个子组件该表格就增加一列。

如果直接向TableLayout中添加组件，那么这个组件将直接占用一行。

在表格布局中，列的宽度由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度（默认总是占满父容器本身）。

在表格布局管理器中，可以为单元格设置如下3种行为方式。

➢ **Shrinkable**：如果某个列被设为 Shrinkable，那么该列的所有单元格的宽度可以被收缩，以保证该表格能适应父容器的宽度。

➢ **Stretchable**：如果某个列被设为Stretchable，那么该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间。

➢ **Collapsed**：如果某个列被设为Collapsed，那么该列的所有单元格会被隐藏。

TableLayout继承了LinearLayout，因此它完全可以支持LinearLayout所支持的全部XML属性。除此之外，TableLayout还支持如表2.6所示的XML属性。

表2.6 TableLayout的常用XML属性及相关方法

实例：丰富的表格布局

下面的程序示范了如何使用TableLayout来管理组件的布局。下面是界面布局所使用的布局文件。

程序清单：
codes\02\2.2\TableLayoutTest\app\src\main\res\layout\main.xml

上面页面中定义了3个TableLayout，3个TableLayout中粗体字代码指定了它们对各列的控制行为。

➢ 第1个TableLayout，指定第2列允许收缩，第3列允许拉伸。

➢ 第2个TableLayout，指定第2列被隐藏。

➢ 第3个TableLayout，指定第2列和第3列允许拉伸。

接下来为布局中第1个TableLayout添加两行，第1行不使用TableRow，直接添加一个Button，那么该Button自己将占用整行。第2行先添加一个TableRow，并为TableRow添加3个Button，那说明该表格将包含3列。第1个表格的界面布局代码如下。

程序清单：
codes\02\2.2\TableLayoutTest\app\src\main\res\layout\main.xml

注意：

在上面的界面布局文件中我们直接把按钮上的文本写在布局文件中，这不是一种好的做法，因为Android推荐将这些字符串集中放到XML文件中管理。但此处为了编程简单，所以直接在XML布局文件中给出了按钮文本的字符串。

接下来为布局中第2个TableLayout添加两行，第1行不使用TableRow，直接添加一个Button，那么该Button自己将占用整行。第2行先添加一个TableRow，并为TableRow添加3个Button，那说明该表格将包含3列。第2个表格内容与第1个表格内容基本相似，但由于我们为第2个表格指定了android：collapseColumns="1"，这意味着第2行中间的按钮将会被隐藏。

再接下来为布局中第3个TableLayout添加3行，第1行不使用TableRow，直接添加一个Button，那么该Button自己将占用整行。第2行先添加一个TableRow，并为TableRow添加3个Button，那说明该表格将包含3列。第3行也先添加一个TableRow，并为TableRow添加两个按钮，这意味着表格的第3行只有两个单元格，第3个单元格为空。

第3个表格布局管理器的内容如下。

程序清单：
codes\02\2.2\TableLayoutTest\app\src\main\res\layout\main.xml

使用Activity来显示上面的界面布局，将会看到如图2.11所示的界面。

2.2.3 帧布局

帧布局由FrameLayout所代表，FrameLayout直接继承了ViewGroup组件。

帧布局容器为每个加入其中的组件创建一个空白的区域（称为一帧），每个子组件占据一帧，这些帧都会根据gravity属性执行自动对齐。帧布局的效果有点类似于AWT编程的CardLayout，都是把组件一个个地叠加在一起。与CardLayout的区别在于，CardLayout可以将下面的Card移上来，但FrameLayout则没有提供相应的方法。

图2.11 表格布局

表2.7显示了FrameLayout常用的XML属性及相关方法说明。

表2.7 FrameLayout的常用XML属性及相关方法

FrameLayout包含的子元素也受FrameLayout.LayoutParams控制，因此它所包含的子元素也可指定android：layout_gravity属性，该属性控制该子元素在FrameLayout中的对齐方式。

下面示范了帧布局的用法，可以看到6个TextView叠加在一起，上面的TextView遮住下面的TextView。下面是使用帧布局的页面定义代码。

程序清单：
codes\02\2.2\FrameLayoutTest\app\src\main\res\layout\main.xml

上面的界面布局定义使用FrameLayout布局，并向该布局容器中添加了6个TextView，这6个TextView的高度、宽度则逐渐减少—这样可以保证最先添加的TextView不会被完全遮挡；而且我们设置了6个TextView的背景色渐变。

使用Activity显示上面的界面布局，将看到如图2.12所示的效果。

图2.12 帧布局

实例：霓虹灯效果

如果考虑轮换改变上面的帧布局中6个TextView的背景色，就会看到上面的颜色渐变条不断地变换，就像大街上的霓虹灯一样。下面的程序还是使用上面的FrameLayout布局管理器，只是程序启动了一个线程来控制周期性地改变这6个TextView的背景色。下面是该主程序的代码。

程序清单：
codes\02\2.2\FrameLayoutTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码定义了一个每0.2秒执行一次的任务，该任务仅仅向Handler发送一条消息，通知它更新6个TextView的背景色。

可能会有读者提出疑问：为何不直接在run（）方法里直接更新6个TextView的背景色呢？这是因为Android的View和UI组件不是线程安全的，所以Android不允许开发者启动线程访问用户界面的UI组件。因此，程序中额外定义了一个Handler来处理TextView背景色的更新。

注意：

上面的程序中直接使用了R.color.color1、R.color.color2、R.color.color3等整型常量来代表颜色，这也得益于Android的资源访问支持，本书后面会有关于颜色资源的详细介绍。

简单地说，上面的程序通过任务调度控制了每间隔0.2秒轮换更新一次6个TextView的背景色，这样看上去就像大街上的霓虹灯了。

2.2.4 相对布局

相对布局由RelativeLayout所代表，相对布局容器内子组件的位置总是相对兄弟组件、父容器来决定的，因此这种布局方式被称为相对布局。

如果A组件的位置是由B组件的位置来决定的，Android要求先定义B组件，再定义A组件。

RelativeLayout可支持如表2.8所示的两个XML属性。

表2.8 RelativeLayout的XML属性及相关方法说明

为了控制该布局容器中各子组件的布局分布，RelativeLayout提供了一个内部类：RelativeLayout.LayoutParams，该类提供了大量的XML属性来控制RelativeLayout布局容器中子组件的布局分布。

RelativeLayout.LayoutParams里只能设为true、false的XML属性如表2.9所示。

表2.9 RelativeLayout.LayoutParams里只能设为boolean值的属性

RelativeLayout.LayoutParams里属性值为其他UI组件ID的XML属性如表2.10所示。

表2.10 RelativeLayout.LayoutParams里只能设为其他UI组件ID的属性

除此之外，RelativeLayout.LayoutParams还继承了android.view.ViewGroup.MarginLayout Params，因此RelativeLayout

布局容器中每个子组件也可指定android.view.ViewGroup.MarginLayoutParams所支持的各XML属性。

下面以一个实例来介绍相对布局的用法。

实例：梅花布局效果

相对布局容器中的子组件总是相对其他组件来决定分布位置的，可以考虑先把一个组件放在相对布局容器的中间，然后以该组件为中心，将其他组件分布在该组件的四周，这样就可以形成“梅花”布局效果。

下面是“梅花”布局效果的界面布局文件。

程序清单：
codes\02\2.2\RelativeLayoutTest\app\src\main\res\layout\main.xml

上面程序中的第一行粗体字代码控制该组件位于父容器的中央，接下来定义的4个组件依次环绕在第一个组件四周。使用Activity来显示上面的布局文件，可以看到如图2.13所示的效果。

提示：

这里例子中使用了一张图片，这也是利用了Android的图片资源访问策略，本书后面还会集中介绍图片资源访问。

2.2.5 网格布局

图2.13 相对布局实现的“梅花”效果

网格布局由GridLayout所代表，它是Android 4.0新增的布局管理器，因此需要在Android 4.0之后的版本中才能使用该布局管理器。如果希望在更早的Android平台上使用该布局管理器，则需要导入相应的支撑库。

GridLayout的作用类似于HTML中的table标签，它把整个容器划分成rows×columns个网格，每个网格可以放置一个组件。除此之外，也可以设置一个组件横跨多少列、一个组件纵跨多少行。

GridLayout提供了setRowCount（int）和setColumnCount（int）方法来控制该网格的行数量和列数量。

表2.11显示了GridLayout常用的XML属性及相关方法。

表2.11 GridLayout的XML属性及相关方法

为了控制GridLayout布局容器中各子组件的布局分布，GridLayout提供了一个内部类：GridLayout.LayoutParams，该类提供了大量的XML属性来控制GridLayout布局容器中子组件的布局分布。

表2.12显示了GridLayout.LayoutParams常用的XML属性及相关方法。

表2.12 GridLayout.LayoutParams的XML属性及相关方法

下面将会通过一个实例来示范GridLayout的功能和用法。

实例：计算器界面

为了实现如图2.14所示的计算器界面，可以考虑将该界面分解成一个6×4的网格，其中第一个文本框横跨4列，第二个按钮横跨4列，后面每个按钮各占一格。

为了实现该界面，考虑按如下步骤来操作。

① 在布局管理器中定义一个GridLayout，并在该GridLayout中依次定义文本框、按钮，该文本框、按钮各横跨4列。

② 在Java代码中循环16次，依次添加16个按钮。

图2.14 计算器界面

下面先定义界面布局文件。

程序清单：
codes\02\2.2\GridLayoutTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了一个6×4的GridLayout，并在该布局管理器中添加了两个子组件，其中第一个子组件横跨4列，第二个子组件也横跨4列。

接下来使用如下Java代码采用循环控制添加16个按钮。

程序清单：
codes\02\2.2\GridLayoutTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的粗体字代码采用循环向GridLayout中添加了16个按钮，添加16个按钮时指定了每个按钮所在的行号、列号，并指定这些按钮将会自动填充单元格的所有空间—这样避免单元格中的大量空白。运行该程序，将可以看到如图2.14所示的界面。

2.2.6 绝对布局

绝对布局由AbsoluteLayout所代表。绝对布局就像Java AWT编程中的空布局，就是Android不提供任何布局控制，而是由开发人员自己通过X坐标、Y坐标来控制组件的位置。当使用AbsoluteLayout作为布局容器时，布局容器不再管理子组件的位置、大小—这些都需要开发人员自己控制。

注意：

大部分时候，使用绝对布局都不是一个好思路，因为运行Android应用的手机往往千差万别，因此屏幕大小、分辨率都可能存在较大差异，使用绝对布局会很难兼顾不同屏幕大小、分辨率的问题。因此，AbsoluteLayout布局管理器已经过时。

使用绝对布局时，每个子组件都可指定如下两个XML属性。

➢ layout_x：指定该子组件的X坐标。

➢ layout_y：指定该子组件的Y坐标。

实例：登录界面

下面介绍一个使用绝对布局开发的登录界面实例，这个登录界面中的所有组件都通过“绝对定位”的方式来指定位置。下面是该登录界面的界面布局文件。

程序清单：
codes\02\2.2\AbsoluteLayoutTest\app\src\main\res\layout\main.xml

上面的绝对布局容器中的每个子组件都指定了layout_x、layout_y两个定位属性，这样才控制了每个子组件在容器中的出现位置。使用

Activity显示上面的页面，将看到如图2.15所示的界面。

图2.15 绝对布局实现的登录界面

不要以为笔者一下子就通过绝对布局做出了图2.15所示的登录界面，实际上这个登录界面是不断调整各组件的位置，经过多次尝试之后得到的结果。当使用绝对布局来控制子组件布局时，编程要烦琐得多，而且在不同屏幕上的显示效果差异也很大。

上面的界面布局中指定各组件的android：layout_x、android：layout_y属性时指定了形如20dp这样的属性值，这是一个距离值。Android中一般支持如下常用的距离单位。

➢ **px（像素）**：每个px对应屏幕上的一个点。

➢ **dip或dp（device independent pixels，设备独立像素）**：一种基于屏幕密度的抽象单位。在每英寸160点的显示器上，1dip=1 px。但随着屏幕密度的改变，dip与px的换算会发生改变。

➢ **sp（scaled pixels，比例像素）**：主要处理字体的大小，可以根据用户的字体大小首选项进行缩放。

➢ **in（英寸）**：标准长度单位。

➢ **mm（毫米）**：标准长度单位。

➢ **pt（磅）**：标准长度单位，1/72英寸。

2.3 第2组UI组件：TextView及其子类

前面介绍了Android界面编程的一些基础知识，接下来将要介绍的是Android基本界面组件。“九层之台，起于垒土”—无论看上去多么美观的UI界面，开始都是先创建容器（ViewGroup的实例），然后不断地

向容器中添加界面组件，最后形成一个美观的UI界面。掌握这些基本用户界面组件是学好Android编程的基础。

2.3.1 文本框（TextView）与编辑框（EditText）的功能和用法

TextView直接继承了View，它还是EditText、Button两个UI组件类的父类。TextView的作用就是在界面上显示文本—从这个意义上来看，它有点类似于Swing编程中的JLabel，不过它比JLabel功能更强大。

从功能上来看，TextView其实就是一个文本编辑器，只是Android关闭了它的文字编辑功能。如果开发者想要定义一个可以编辑内容的文本框，则可以使用它的子类：EditText，EditText允许用户编辑文本框中的内容。

图2.16 T extView及其子类的类图

TextView还派生了一个CheckedTextView，CheckedTextView增加了一个checked状态，开发者可通过setChecked（boolean）和isChecked（）方法来改变、访问该组件的checked状态。除此之外，该组件还可通过setCheckMarkDrawable（）方法来设置它的勾选图标。

不仅如此，TextView还派生出了Button类，TextView及其子类的类图如图2.16所示。

TextView和EditText具有很多相似之处，它们之间的最大区别在于TextView不允许用户编辑文本内容，而EditText则允许用户编辑文本内容。

TextView提供了大量的XML属性，这些XML属性大部分不仅可适用于TextView，而且可适用于它的子类（EditText、Button等）。表2.13显示了TextView支持的XML属性及相关方法。

表2.13 TextView支持的XML属性及相关方法

续表

续表

下面通过一系列实例来介绍TextView和CheckedTextView的用法。

实例：不同颜色、字体、带链接的文本

由于TextView提供了大量XML属性，因此我们可以通过这些XML属性来控制TextView中文本的行为，例如如下界面布局文件。

程序清单：
codes\02\2.3\TextViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了6个TextView，它们指定的规则如下。

➢ 第1个TextView指定android：textSize="20pt"，这就指定了字号为20pt。而且指定了在文本框的结尾处绘制图片。

➢ 第2个TextView指定android：ellipsize="middle"，这就指定了当文本多于文本框的宽度时，从中间省略文本。而且指定了 android：textAllCaps="true"，表明该文本框的所有字母大写。

➢ 第3个TextView指定android：autoLink="email|phone"，这就指定了该文本框会自动为文本框内的E-mail地址、电话号码添加超链接。

➢ 第4个TextView指定一系列android：shadowXXX属性，这就为该文本框内的文本内容添加了阴影。

➢ 第5个TextView指定android：password="true"，这就指定了该文本框会用点来代替显示所有字符。

➢ 第6个CheckedTextView指定android：checkMark="@drawable/ok"，这就指定了该可勾选文本框的勾选图标。

图2.17 不同字体、颜色带链接的文本

使用Activity来显示上面的界面布局文件，将看到如图2.17所示的界面。

实例：圆角边框、渐变背景的TextView

在默认情况下，TextView是不带边框的，如果想为TextView添加边框，只能通过“曲线救国”的方式来实现—我们可以考虑为TextView设置一个背景Drawable，该Drawable只是一个边框，这样就实现了带边框的TextView。

由于可以为TextView设置背景Drawable对象，因此可以在定义Drawable时不仅指定边框，还可以指定渐变背景，这样即可为TextView添加渐变背景和边框。

下面的界面布局文件中定义了两个TextView，界面布局文件的代码如下。

程序清单：
codes\02\2.3\TextViewTest2\app\src\main\res\layout\ma

in.xml

上面的界面布局文件中定义了两个TextView，其中第一个指定了背景，第二个定义文本框时指定使用圆角边框、渐变背景。第一个文本框所指定的背景是由XML文件定义的，将该文件放在drawable文件夹内，该XML文件也可当成Drawable使用。下面是该XML文件的代码。

程序清单：
codes\02\2.3\TextViewTest2\app\src\main\res\drawable\bg_border.xml

第二个文本框所指定的背景是由XML文件定义的，将该文件放在drawable文件夹内，该XML文件也可当成Drawable使用。下面是该XML文件的代码。

程序清单：
codes\02\2.3\TextViewTest2\app\src\main\res\drawable\bg_border2.xml

使用Activity来显示上面定义的布局页面，可以看到如图2.18所示的界面。

从图2.18不难看出，通过为TextView的android：background赋值，可以为文本框增加大量自定义外观，这种控制方式非常灵活。

需要指出的是，表面上这里只是在介绍TextView，但由于TextView是EditText、Button等类的父类，因此此处介绍的对TextView控制的属性，同样适用于EditText与Button。

2.3.2 EditText的功能与用法

图2.18 圆角边框、渐变背景的文本

EditText与TextView非常相似，它甚至与TextView共用了绝大部分XML属性和方法。EditText与TextView的最大区别在于：EditText可以接受用户输入。表2.13中介绍的与输入相关的属性主要就是为EditText准备的。

EditText组件最重要的属性是inputType，该属性相当于HTML的<input.../>元素的type属性，用于将EditText设置为指定类型的输入组件。inputType能接受的属性值非常丰富，而且随着Android版本的升级，该属性能接受的类型还会增加。

EditText还派生了如下两个子类。

➢ **AutoCompleteTextView**：带有自动完成功能的 EditText，实际上该组件的命名不太恰当。笔者认为该类名应该叫AutoCompleteEditText比较合适。由于该类通常需要与Adapter结合使用，因此将会在讲解AdapteView组件时介绍该组件的用法。

➢ **ExtractEditText**：它并不是UI组件，而是EditText组件的底层服务类，负责提供全屏输入法支持。

下面通过一个实例来介绍EditText的用法。

实例：用户友好的输入界面

对于一个用户友好的输入界面而言，接受用户输入的文本框内默认会提示用户如何输入；当用户把焦点切换到输入框时，输入框自动选中其中已输入的内容，避免用户删除已有内容；当用户把焦点切换到只接受电话号码的输入框时，输入法自动切换到数字键盘。

下面程序的输入界面完成了以上功能，输入界面的界面布局文件如下。

程序清单：
codes\02\2.3\InputUI\app\src\main\res\layout\main.xml

上面界面布局中的第一个文本框通过android：hint指定了文本框的提示信息：请填写登录账号一这是该文本框默认的提示。当用户还没有输入时，该文本框内默认显示这段信息；第二个输入框通过android：inputType="numberPassword"设置这是一个密码框，而且只能接受数字密码，用户在该文本框输入的字符会以点号代替；第三个输入框通过android：inputType="number"设置为只能接受数值的输入框；第四个输入框通过android：inputType="date"指定它是一个日期输入框；第五个输入框通过android：inputType="phone"设置为一个电话号码输入框。

使用Activity显示上面的界面布局文件，将可看到如图2.19所示的界面。

从图2.19可以看出，当用户把焦点定位到数字密码输入框时，系统自动显示数字输入键盘，这就是设置android：inputType="numberPassword"的作用。第一个文本框默认显示了"请填写登录账号”，这是由android：hint属性指定的。

2.3.3 按钮（Button）组件的功能与用法

Button继承了TextView，它主要是在UI界面上生成一个按钮，该按钮可以供用户单击，当用户单击按钮时，按钮会触发一个onClick事件。关于onClick事件编程的简单示例，本书前面已经见到很多，后面还会详细介绍Android的事件编程。

按钮使用起来比较容易，可以通过指定android：background属性为按钮增加背景颜色或背景图片，如果将背景图片设为不规则的背景图片，则可以开发出各种不规则形状的按钮。

图2.19 友好的输入界面

如果只是使用普通的背景颜色或背景图片，那么这些背景是固定的，不会随着用户的动作而改变。如果需要让按钮的背景颜色、背景图片随用户动作动态改变，则可以考虑使用自定义Drawable对象来实现。

下面通过实例来开发出更强大的按钮。

实例：按钮、圆形按钮、带文字的图片按钮

为了定义图片随用户动作改变的按钮，可以考虑使用XML资源文件来定义Drawable对象，再将Drawable对象设为Button的android：background属性值，或设为ImageButton的android：src属性值。

例如，有如下界面布局文件。

程序清单：
codes\02\2.3\ButtonTest\app\src\main\res\layout\main.xml

上面界面布局中的第一个按钮是一个普通按钮，但为该按钮的文字指定了阴影—配置阴影的方式与为TextView配置阴影的方式完全相同，这是因为Button的本质还是TextView。第二个按钮通过background属性配置了背景图片，因此该按钮将会显示为背景图片形状的按钮。

第三个按钮有点特殊，它指定了android：background属性为@drawable/button_selector，该属性值引用一个Drawable资源，该资源对应的XML文件如下。

程序清单：
codes\02\2.3\ButtonTest\app\src\main\res\drawable\button_selector.xml

上面的资源文件使用<selector.../>元素定义了一个StateListDrawable对象一本书后面会详细介绍如何使用XML文件来定义Drawable的内容，此处不再深入讲解。

使用Activity显示上面的界面布局文件，可以看到如图2.20所示的界面。

在图2.20所示界面的三个按钮中，前两个按钮的背景色、图片都是固定的，用户单击这两个按钮不会看到任何改变；用户单击第三个按钮时，将会看到按钮的图片被切换为红色。

图2.20 各种按钮

以笔者的经验来看，Button生成的按钮功能很强大，就像第三个按钮就是Button生成的，而且它也可以通过背景色来设置图片，因此使用Button生成的按钮不仅可以是普通的文字按钮，也可以定制成任意形状，并可以随用户交互动作改变外观。

2.3.4 使用9Patch图片作为背景

从图2.20所示的第三个按钮来看，当按钮的内容太多时，Android会自动缩放整张图片，以保证背景图片能覆盖整个按钮。但这种缩放整张

图片的效果可能并不好。可能存在的情况是我们只想缩放图片中某个部分，这样才能保证按钮的视觉效果。

为了实现只缩放图片中某个部分的效果，我们需要借助于9Patch图片来实现。9Patch图片是一种特殊的PNG图片，这种图片以.9.png结尾，它在原始图片四周各添加一个宽度为1像素的线条，这4条线就决定了该图片的缩放规则、内容显示规则。

左侧和上侧的直线共同决定了图片的缩放区域：以左边直线为左边界绘制矩形，它覆盖的区域可以在纵向上缩放；以上面直线为上边界绘制矩形，它覆盖的区域可以水平缩放；它们二者的交集可以在两个方向上缩放。图2.21显示了定义图片缩放区域的示意图。

右侧和下侧的直线共同决定图片的内容显示区域：以右边直线为右边界绘制矩形，以下边直线为下边界绘制矩形，它们二者的交集就是图片的内容显示区域。图 2.22 显示了定义图片的内容显示区域的示意图。

图2.21 定义图片缩放区域

图2.22 定义图片的内容显示区域

Android为制作9Patch图片提供了draw9patch工具，该工具位于Android SDK安装路径的tools目录下，进入该目录双击draw9patch.bat文件，即可启动该工具。

提示：

如果启动该程序时提示java.lang.NoClassDefFoundError：org/jdesktop/swingworker/SwingWorker异常，则可能的原因是draw9patch.bat工具需要依赖于SwingWorker类，这个类在JDK 1.5以

前需要单独下载swing-worker的JAR包，从JDK 1.6以后，该类已经加入了javax.swing包下，但Android SDK还在使用早期的SwingWorker，而且Android SDK又删除了swing-worker-1.1.jar包，所以引发上面的错误。为了能正常启动draw9patch工具，手动将swing-worker-1.1.jar复制到Android SDK安装路径的tools/lib路径下即可。

启动draw9patch工具之后，通过该工具主菜单上的"File"→"Open 9"-"Patch"菜单项打开一张PNG图片，然后通过该工具定义图片的缩放区域、内容显示区域。图2.23显示了定义9Patch图片的示意图。

将上面生成的图片保存到应用的app\src\main\res\drawable路径下，主文件名任意，draw9patch工具自动将该文件的后缀保存为.9.png。

定义9Patch图片之后，接下来在应用中定义两个按钮，分别使用原始图片和9Patch图片作为Button的背景。页面定义文件比较简单，此处不再赘述。普通图片为背景的按钮及9Patch图片为背景的按钮的界面如图2.24所示。

图2.23 定义9Patch图片

图2.24 普通图片与9Patch图片作为背景的对比

从图2.24可以看出，普通图片作为背景时，整张图片都被缩放了；如果使用9Patch图片作为背景，则只有指定区域才会被缩放。

2.3.5 单选钮（RadioButton）和复选框（CheckBox）的功能与用法

单选钮（RadioButton）、复选框（CheckBox）、状态开关按钮（ToggleButton）和开关（Switch）是用户界面中最普通的UI组件，

它们都继承了Button类，因此都可直接使用Button支持的各种属性和方法。

RadioButton、CheckBox与普通按钮不同的是，它们多了一个可选中的功能，因此RadioButton、CheckBox都可额外指定一个android：checked属性，该属性用于指定RadioButton、CheckBox初始时是否被选中。

RadioButton与CheckBox的不同之处在于，一组RadioButton只能选中其中一个，因此RadioButton通常要与RadioGroup一起使用，用于定义一组单选钮。

下面通过实例来介绍RadioButton和CheckBox的用法。

实例：利用单选钮、复选框获取用户信息

在需要获取用户信息的界面中，有些信息不需要用户直接输入，可以考虑让用户进行选择，比如用户的性别、爱好等。下面的界面布局文件定义了一个让用户选择的输入界面。

程序清单：
codes\02\2.3\CheckButtonTest\app\src\main\res\layout\main.xml

上面的界面布局中定义了一组单选钮，并默认勾选了第一个单选钮，这组单选钮供用户选择性别；还定义了三个复选框，供用户选择喜欢的颜色。

注意：

如果在XML布局文件中默认勾选了某个单选钮，则必须为该组单选钮的每个按钮指定android：id属性值；否则这组单选钮不能正常工作。

为了监听单选钮、复选框的勾选状态的改变，可以为它们添加事件监听器。例如，下面Activity为RadioGroup添加了事件监听器，该监听器可以监听这组单选钮的勾选状态的改变。

程序清单：
codes\02\2.3\CheckButtonTest\app\src\main\java\org\crazyit\ui\MainActivity.java

提示：

上面代码添加事件监听器的方式采用了“委托式”事件处理机制，委托式事件处理机制的原理是，当事件源上发生事件时，该事件将会激发该事件源上的监听器的特定方法。本书第3章还会详细介绍这种事件处理机制。

运行上面的程序，并改变第一组单选钮的勾选状态，将看到如图2.25所示的界面。

图2.25 单选钮、复选框示意图

2.3.6 状态开关按钮（ToggleButton）和开关（Switch）的功能与用法

状态开关按钮（ToggleButton）和开关（Switch）也是由Button派生出来的，因此它们的本质也是按钮，Button支持的各种属性、方法也适用于ToggleButton和Switch。从功能上来看，ToggleButton、Switch与CheckBox复选框非常相似，它们都可以提供两种状态。不过ToggleButton、Switch与CheckBox的区别主要体现在功能上，ToggleButton、Switch通常用于切换程序中的某种状态。

表2.14显示了ToggleButton所支持的XML属性及相关方法。图2.15显示了Switch所支持的XML属性及相关方法。

表2.14 ToggleButton支持的XML属性及相关方法

表2.15 Switch支持的XML属性及相关方法

下面通过一个动态控制布局的实例来示范ToggleButton与Switch的用法。

实例：动态控制布局

该实例的思路是在页面中增加一个ToggleButton，随着该按钮状态的改变，界面布局中的LinearLayout布局的方向在水平布局、垂直布局之间切换。下面是该程序所使用的界面布局文件。

程序清单：
codes\02\2.3\ToggleButtonTest\app\src\main\res\layout\main.xml

上面LinearLayout中定义了三个按钮，该LinearLayout默认采用垂直方向的线性布局。接下来我们为ToggleButton按钮、Switch按钮绑定监听器，当它的选中状态发生改变时，程序通过代码来改变LinearLayout的布局方向。

程序清单：
codes\02\2.3\ToggleButtonTest\app\src\main\java\org\crazyit\ui\MainActivity.java

运行上面的程序，随着用户改变ToggleButton按钮的状态，界面布局的方向也在不断发生改变。图2.26显示了ToggleButton的界面。

2.3.7 时钟（AnalogClock和TextClock）的功能与用法

图2.26 ToggleButton与Switch的功能

时钟UI组件是两个非常简单的组件：TextClock本身就继承了TextView—也就是说，它本身就是文本框，只是它里面显示的内容总是当前时间。与TextView不同的是，为TextClock设置android：text属性没什么作用。

TextClock取代早期的DigitalClock组件，因此功能更加强大—TextClock能以24小时制或12小时制来显示时间，而且可以由程序员来指定时间格式。表2.17显示了TextClock所支持的XML属性及相关方法

表2.16 TextClock支持的XML属性及相关方法

AnalogClock则继承了View组件，它重写了View的OnDraw方法，它会在View上绘制模拟时钟。

表2.17显示了AnalogClock所支持的XML属性。

表2.17 AnalogClock支持的XML属性

TextClock和AnalogClock都会显示当前时间。不同的是，TextClock显示数字时钟，可以显示当前的秒数；AnalogClock显示模拟时钟，不会

显示当前的秒数。

下面通过实例来示范AnalogClock和TextClock的用法。

实例：手机里的“劳力士”

由于我们可以通过图片定制AnalogClock模拟指针的表盘、时针、分针，因此只要使用合适的图片，就可以对AnalogClock进行任意定制。本实例将会使用“劳力士”图片来定义模拟时钟，从而开发手机里的“劳力士”。

下面是本实例的界面布局文件。

程序清单：
codes\02\2.3\ClockTest\app\src\main\res\layout\main.xml

使用Activity显示上面的界面布局，将看到如图2.27所示的界面。

正如从上面的粗体字代码中看到的，如果想控制模拟时钟显示时间的字号大小、字体颜色等，都可通过android：textSize、android：textColor等属性进行控制—因为TextClock的本质还是一个TextView，所以它可以使用TextView的XML属性和方法。

2.3.8 计时器（Chronometer）

图2.27 数字时钟和模拟时钟

Android 还提供了一个计时器组件：Chronometer，该组件与 TextClock 都继承自 TextView，因此它们都会显示一段文本。但

Chronometer并不显示当前时间，它显示的是从某个起始时间开始，一共过去了多长时间。

Chronometer的用法也很简单，它只提供了一个android：format属性，用于指定计时器的计时格式。除此之外，Chronometer还支持如下常用方法。

➢ **setBase（long base）：** 设置计时器的起始时间。

➢ **setFormat（String format）：** 设置显示时间的格式。

➢ **start（）：** 开始计时。

➢ **stop（）：** 停止计时。

➢ **setOnChronometerTickListener（Chronometer.OnChronometerTickListener listener）：** 为计时器绑定事件监听器，当计时器改变时触发该监听器。

下面的程序简单示范了Chronometer的用法。该程序界面中定义了一个Chronometer组件和一个Button组件。当用户单击Button时系统开始计时，当计时超过20秒时停止计时。

程序清单：
codes\02\2.3\ChronometerTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码用于设置Chronometer的起始时间，并启动Chronometer。启动计时器后可以看到该组件显示已流逝的时间，如图2.28所示。

图2.28 计时器

提示：

程序中用到的SystemClock类仅仅是一个获取系统时间、运行时间的工具类，用法很简单，读者自行查阅API文档即可。

2.4 第3组UI组件：ImageView及其子类

图2.29 I mageView及其子类的类图

ImageView继承自View组件，它的主要功能是用于显示图片一实际上这个说法不太严谨，因为它能显示的不仅仅是图片，任何Drawable对象都可使用ImageView来显示。

除此之外，ImageView还派生了ImageButton、ZoomButton等组件，图2.29显示了ImageView及其子类的类图。

从图2.29可以看出，ImageView派生了ImageButton、ZoomButton等组件，因此ImageView支持的XML属性、方法，基本上也可应用于ImageButton、ZoomButton等组件。

表2.18显示了ImageView支持的常用XML属性及相关方法。

表2.18 ImageView支持的XML属性及相关方法

ImageView所支持的android：scaleType属性可指定如下属性值。

➢ **matrix（ImageView.ScaleType.MATRIX）**：使用matrix方式进行缩放。

➢ **fitXY（ImageView.ScaleType.FIT_XY）**：对图片横向、纵向独立缩放，使得该图片完全适应于该ImageView，图片的纵横比可能会改变。

➢ **fitStart（ImageView.ScaleType.FIT_START）**：保持纵横比缩放图片，直到该图片能完全显示在ImageView中（图片较长的边长与ImageView相应的边长相等），缩放完成后将该图片放在ImageView的左上角。

➢ **fitCenter（ImageView.ScaleType.FIT_CENTER）**：保持纵横比缩放图片，直到该图片能完全显示在ImageView中（图片较长的边长与ImageView相应的边长相等），缩放完成后将该图片放在ImageView的中央。

➢ **fitEnd（ImageView.ScaleType.FIT_END）**：保持纵横比缩放图片，直到该图片能完全显示在ImageView中（图片较长的边长与ImageView相应的边长相等），缩放完成后将该图片放在ImageView的右下角。

➢ **center（ImageView.ScaleType.CENTER）**：把图片放在ImageView 的中间，但不进行任何缩放。

➢ **centerCrop（ImageView.ScaleType.CENTER_CROP）**：保持纵横比缩放图片，以使得图片能完全覆盖ImageView。只要图片的最短边能显示出来即可。

➢ **centerInside（ImageView.ScaleType.CENTER_INSIDE）**：保持纵横比缩放图片，以使得ImageView能完全显示该图片。

为了控制ImageView显示的图片，ImageView提供了如下方法。

➢ **setImageBitmap（Bitmap bm）**：使用Bitmap位图设置该ImageView显示的图片。

➢ **setImageDrawable（Drawable drawable）**：使用Drawable对象设置该ImageView显示的图片。

➢ **setImageResource（int resId）**：使用图片资源ID设置该ImageView显示的图片。

➢ **setImageURI（Uri uri）**：使用图片的URI设置该ImageView显示的图片。

ImageView的功能比较简单，下面结合一个图片浏览器的实例来示范ImageView的功能和用法。

实例：图片浏览器

本例的图片浏览器可以改变所查看图片的透明度，可通过调用ImageView的setImageAlpha（）方法来实现。不仅如此，这个图片浏览器还可通过一个小区域来查看图片的原始大小，因此本例会定义两个ImageView，一个用于查看图片整体，一个用于查看图片局部的细节。

下面是本例的界面布局文件。

程序清单：
codes\02\2.4\ImageViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了两个ImageView，其中第一个ImageView指定了android：scaleType="fitCenter"，这表明ImageView显示图片时会保持纵横比缩放，并将缩放后的图片放在该ImageView的中央。

为了能动态改变图片的透明度，接下来需要为按钮编写事件监听器，当用户单击按钮时动态改变图片的Alpha值。为了能动态显示图片的局

部细节，程序为第一个ImageView添加OnTouchListener监听器，用户在第一个ImageView上发生触摸事件时，程序从原始图片中读取相应部分的图片，并将其显示在第二个ImageView中。下面是主程序的代码。

程序清单：
codes\02\2.4\ImageViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的第一行粗体字代码调用了ImageView的setImageResoure（）方法动态修改该ImageView所显示的图片，当用户单击“下一张”按钮时，即可控制ImageView显示图片数组中的下一张图片。程序中第二行粗体字代码会动态设置第一个ImageView的Alpha值，这就是动态改变该图片的透明度了。

程序中的粗体字代码段将会用于从源位图的触碰点“截取”源位图，并将截取的部分显示在第二个ImageView中。此时用到了Bitmap类，它是一个代表位图的类，调用它的createBitmap（）静态方法即可截取位图的指定部分，该方法返回截取区域生成的新位图。

运行上面的程序，将看到如图2.30所示的界面。

图2.30 使用ImageView实现图片浏览器

ImageView派生了如下两个子类。

➢ **ImageButton**：图片按钮。

➢ **QuickContactBadge**：显示关联到特定联系人的图片。

Button与ImageButton的区别在于，Button生成的按钮上显示文字，而ImageButton上则显示图片。需要指出的是，为ImageButton按钮指定android：text属性没用（ImageButton的本质是ImageView），即使指定了该属性，图片按钮上也不会显示任何文字。

如果考虑使用ImageButton，图片按钮可以指定android：src属性，该属性既可使用静止的图片，也可使用自定义的Drawable对象，这样即可开发出随用户动作改变图片的按钮。

ImageButton派生了一个ZoomButton，ZoomButton可以代表“放大”、“缩小”两个按钮。ZoomButton的行为基本类似于ImageButton，只是Android默认提供了btn_minus、btn_plus两个Drawable资源，只要为ZoomButton的android：src属性分别指定btn_minus、btn_plus，即可实现“缩小”、“放大”按钮。

实际上Android还提供了一个ZoomControls组件，该组件相当于同时组合了“放大”、“缩小”两个按钮，并允许分别为两个按钮绑定不同的事件监听器。

实例：强大的图片按钮

本实例定义了多个图片按钮，并定义了两个ZoomButton。两个ZoomButton的android：src属性分别指定为@android：drawable/btn_minus、@android：drawable/btn_plus，这样即可定义“缩小”和“放大”两个按钮。

下面是该实例的界面布局文件。

程序清单：
codes\02\2.4\ImageButtonTest\app\src\main\res\layout\main.xml

上面界面布局文件的开头定义了两个ImageButton，第一个ImageButton的android：src指定为一张静态图片，这样无论用户有怎样的行为，该ImageButton总显示这张静态图片。第二个ImageButton的android：src指定为@drawable/button_selector，该Drawable组合了两张图片，可以保证用户单击该按钮时切换图片。

该布局文件中间定义了两个ZoomButton，并分别指定了放大、缩小Drawable。该布局文件的结尾处定义了一个ZoomControls组件。使用Activity显示该界面布局文件，可以看到如图2.31所示的效果。

图2.31 强大的图片按钮

对于第二个图片按钮，当用户单击第二个图片按钮时，将会看到按钮的图片被切换为红色图片。

实例：使用QuickContactBadge关联联系人

QuickContactBadge继承了ImageView，因此它的本质也是图片按钮，也可以通过android：src属性指定它显示的图片。QuickContactBadge额外增加的功能是：该图片可以关联到手机中指定联系人，当用户单击该图片时，系统将会打开相应联系人的联系方式界面。

为了让QuickContactBadge与特定联系人关联，可以调用如下方法。

➢ **assignContactFromEmail（String emailAddapp\src\main\ress，boolean lazyLookup）：**将该图片关联到指定E-mail地址对应的联系人。

➢ **assignContactFromPhone（String phoneNumber，boolean lazyLookup）：**将该图片关联到指定电话号码对应的联系人。

➢ **assignContactUri（Uri contactUri）**：将该图片关联到特定Uri对应的联系人。

本实例示范了如何使用QuickContactBadge关联特定联系人。

该实例的界面布局文件如下。

程序清单：
codes\02\2.4\QuickContactBadgeTest\app\src\main\res\layout\main.xml

上面的界面布局文件非常简单，它只是包含了一个QuickContactBadge组件与TextView组件。接下来在Activity代码中可以让QuickContactBadge与特定联系人建立关联。下面是该Activity的代码。

程序清单：
codes\02\2.4\QuickContactBadgeTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的粗体字代码将该QuickContactBadge组件与电话号码为02088888888的联系人建立了关联，当用户单击该图片时，系统将会打开该联系人对应的联系方式界面。

运行该实例，可以看到如图2.32所示的界面。

图2.32左边的图片就是QuickContactBadge组件，单击该组件，如果手机中存储有电话号码为02088888888的联系人，系统将会打开该联系人的联系方式界面，如图2.33所示。

图2.32 使用QuickContactBadge

图2.33 使用QuickContactBadge打开特定联系人

2.5 第4组UI组件：AdapterView及子类

AdapterView是一组重要的组件，AdapterView本身是一个抽象基类，它派生的子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

AdapterView具有如下特征。

➢ AdapterView继承了ViewGroup，它的本质是容器。

➢ AdapterView可以包括多个“列表项”，并将多个“列表项”以合适的形式显示出来。

➢ AdapterView 显示的多个“列表项”由 Adapter 提供。调用 AdapterView 的setAdapter（Adapter）方法设置Adapter即可。

AdapterView及其子类的继承关系图如图2.34所示。

图2.34 AdapterView及其子类的继承关系图

从图2.34不难看出，AdapterView派生了三个子类：AbsListView、AbsSpinner和AdapterViewAnimator，这三个子类依然是抽象的，实际使用时往往采用它们的子类。

注意：

由于Gallery是一个已经过时的API，Android推荐使用HorizontalScrollView来代替它，因此本书不再介绍Gallery的功能和用法。

2.5.1 列表视图（ListView）和ListActivity

ListView是手机系统中使用非常广泛的一种组件，它以垂直列表的形式显示所有列表项。

生成列表视图有如下两种方式。

➢ 直接使用ListView进行创建。

➢ 让Activity继承ListActivity（相当于该Activity显示的组件为ListView）。

一旦在程序中获得了ListView之后，接下来就需要为ListView设置它要显示的列表项了。在这一点上，ListView显示出AdapterView的特征：通过setAdapter（Adapter）方法为之提供Adapter，并由Adapter提供列表项即可。

提示：

ListView、GridView、Spinner、Gallery等AdapterView都只是容器，而Adapter负责提供每个“列表项”组件，AdapterView则负责采用合适的方式显示这些列表项。

AbsListView提供的常用XML属性及相关方法如表2.19所示。

表2.19 AbsListView的常用XML属性及相关方法

ListView提供的常用XML属性如表2.20所示。

表2.20 ListView的常用XML属性

下面通过几个实例来示范ListView的功能与用法。

实例：改变分隔条、基于数组的ListView

下面的界面布局文件中定义了两个ListView。

程序清单：
codes\02\2.5\SimpleListViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了一个ListView，并通过android：entries指定了列表项数组，该 ListView还通过android：divider改变了列表项之间的分隔条。

上面这个ListView指定了android：entries="@array/books"，该属性值用到了数组资源，因此还需要在应用中定义一个名为books的数组，定义数组的资源文件如下。

程序清单：
codes\02\2.5\SimpleListViewTest\app\src\main\res\values\arrays.xml

使用Activity显示上面的ListView，将可以看到如图2.35所示的效果。

使用数组创建ListView十分简单，但这种ListView能定制的内容很少，甚至连每个列表项的字号大小、颜色都不能改变。

如果想对ListView的外观、行为进行定制，就需要把ListView作为AdapterView使用，通过Adapter控制每个列表项的外观和行为。

图2.35 使用数组创建ListView

2.5.2 Adapter接口及实现类

Adapter本身只是一个接口，它派生了ListAdapter和SpinnerAdapter两个子接口，其中ListAdapter为AbsListView提供列表项，而SpinnerAdapter为AbsSpinner提供列表项。Adapter接口及其实现类的继承关系图如图2.36所示。

图2.36 Adapter及其实现类的继承关系图

图2.36所示继承关系类图中粗线框标出的是比较常用的Adapter。从图2.36中所示的继承关系可以看出，几乎所有的Adapter都继承了BaseAdapter，而BaseAdapter同时实现了ListAdapter、SpinnerAdapter两个接口，因此BaseAdapter及其子类可以同时为AbsListView、AbsSpinner提供列表项。

Adapter常用的实现类如下。

➢ **ArrayAdapter**：简单、易用的Adapter，通常用于将数组或List集合的多个值包装成多个列表项。

➢ **SimpleAdapter**：并不简单、功能强大的 Adapter，可用于将 List 集合的多个对象包装成多个列表项。

➢ **SimpleCursorAdapter**：与SimpleAdapter基本相似，只是用于包装Cursor提供的数据。

➢ **BaseAdapter**：通常用于被扩展。扩展BaseAdapter可以对各列表项进行最大限度的定制。

下面先通过ArrayAdapter来实现ListView。

实例：使用ArrayAdapter创建ListView

本实例在界面布局文件中定义了两个ListView。

程序清单：
codes\02\2.5\ArrayAdapterTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了两个ListView，但这两个ListView都没有指定android：entries属性，这意味着它们都需要通过Adapter来提供列表项。

接下来Activity为两个ListView提供Adapter，Adapter决定ListView所显示的列表项。程序如下。

程序清单：
codes\02\2.5\ArrayAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的两行粗体字代码创建了两个ArrayAdapter，创建ArrayAdapter时必须指定如下三个参数。

➢ **Context**：这个参数无须多说，它代表了访问整个 Android 应用的接口。几乎创建所有组件都需要传入Context对象。

➢ **textViewResourceId**：一个资源ID，该资源ID代表一个TextView，该TextView组件将作为ArrayAdapter的列表项组件。

➢ **数组或List**：该数组或List将负责为多个列表项提供数据。

从上面的介绍不难看出，创建ArrayAdapter时传入的第二个参数控制每个列表项的组件，第三个参数则负责为列表项提供数据。该数组或List 包含多少个元素，就将生成多少个列表项，每个列表项都是TextView组件，TextView组件显示的文本由数组或List的元素提供。

以上面代码中的第一个ArrayAdapter为例，该ArrayAdapter对应的数组为{"孙悟空"，"猪八戒"，"牛魔王"}，该数组将会负责生成一个包含三个列表项的ArrayAdapter，每个列表项的组件外观由R.layout.array_item布局文件（该布局文件只是一个TextView组件）控制，第一个TextView列表项显示的文本为“孙悟空”，第二个TextView列表项显示的文本为“猪八戒”……

上面程序中的R.layout.array_item布局文件如下。

程序清单：
codes\02\2.5\ArrayAdapterTest\app\src\main\res\layout\array_item.xml

上面程序中的R.layout.checked_item布局文件与上面的布局文件类似，具体可以参考本书光盘代码。

图2.37 使用ArrayAdatper创建ListView

运行上面的程序，将可以看到如图2.37所示的效果。

实例：基于ListActivity实现列表

如果程序的窗口仅仅需要显示一个列表，则可以直接让Activity继承ListActivity来实现，ListActivity的子类无须调用setContentView（）方法来显示某个界面，而是可以直接传入一个内容Adapter，ListActivity的子类就呈现出一个列表。

例如如下程序。

程序清单：
codes\02\2.5\ListActivityTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序的Activity继承了ListActivity，ListActivity无须界面布局文件—相当于它的布局文件中只有一个 ListView，因此只要为 ListActivity 设置 Adapter 即可。上面程序使用 Android 提供的R.layout.simple_list_item_multiple_choice布局文件作为列表项组件。

图2.38 基于ListActivity实现的列表

运行上面的程序，将看到如图2.38所示的界面。

ListActivity的默认布局是由一个位于屏幕中心的列表组成的。

实例：使用SimpleAdapter创建ListView

通过ArrayAdapter实现Adapter虽然简单、易用，但ArrayAdapter的功能比较有限，它的每个列表项只能是TextView。如果开发者需要实现更复杂的列表项，则可以考虑使用SimpleAdapter。

不要被SimpleAdapter的名字欺骗了，SimpleAdapter并不简单，而且它的功能非常强大。ListView的大部分应用场景，都可以通过SimpleAdapter来提供列表项。

下面先定义界面布局文件。

程序清单：
codes\02\2.5\SimpleAdapterTest\app\src\main\res\layout\main.xml

上面的界面布局文件中仅定义了一个ListView，该ListView将会显示由SimpleAdatper提供的列表项。

下面是Activity代码。

程序清单：
codes\02\2.5\SimpleAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序的关键在于粗体字代码，粗体字代码创建了一个SimpleAdapter。使用SimpleAdapter的最大难点在于创建SimpleAdapter对象，它需要5个参数，其中后面4个参数十分关键。

➢ **第2个参数：**该参数应该是一个List<？e xtends Ma p<String，？>>类型的集合对象，该集合中每个Map<String，？>对象生成一个列表项。

➢ **第3个参数：**该参数指定一个界面布局的ID。例如，此处指定了R.layout.simple_item，这意味着使用app\src\main\res\layout\simple_item.xml文件作为列表项组件。

➢ **第4个参数：**该参数应该是一个String[]类型的参数，该参数决定提取Map<String，？>对象中哪些key对应的value来生成列表项。

➢ **第5个参数：**该参数应该是一个int[]类型的参数，该参数决定填充哪些组件。

从上面的程序看，listItems是一个长度为4的集合，这意味着它生成的ListView将会包含4个列表项，每个列表项都是R.layout.simple_item对应的组件（也就是一个LinearLayout组件）。LinearLayout中包含了3

个组件：ID为R.id.header的ImageView组件、ID为R.id.name和R.id.desc的TextView组件，这些组件的内容由listItems集合提供。

R.layout.simple_item对应的布局文件代码如下。

程序清单：
codes\02\2.5\SimpleAdapterTest\app\src\main\res\layout\simple_item.xml

举例来说，上面SimpleAdapter将会生成4个列表项，其中第一个列表项组件是一个LinearLayout组件，第一个列表项的数据是{"personName"="虎头"，"header"=R.id.tiger，"desc"="可爱的小孩"}Map集合。创建SimpleAdapter时第5个参数、第4个参数指定使用ID为R.id.name组件显示personName对应的值，使用ID为R.id.header组件显示header对应的值，使用ID为R.id.desc的组件显示desc对应的值，这样第一个列表项组件所包含的三个组件都有了显示内容。后面的每个列表项依此类推。

图2.39 使用SimpleAdapter创建ListView

运行上面的程序，将看到如图2.39所示的界面。

SimpleAdapter同样可作为ListActivity的内容Adapter，这样可以让用户方便地定制ListActivity所显示的列表项。

如果需要监听用户单击、选中某个列表项的事件，则可以通过AdapterView的setOnItemClickListener（）方法为单击事件添加监听器，或者通过setOnItemSelectedListener（）方法为列表项的选中事件添加监听器。

例如，可以在上面的Activity中为ListView通过如下代码绑定事件监听器。

程序清单：
codes\02\2.5\SimpleAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

再次运行上面程序，如果单击列表项或选中列表项，将可以看到LogCat控制台有如图2.40所示的输出。

图2.40 ListView的事件监听

注意：

上面的程序为ListView的列表项单击事件、列表项选中事件绑定了事件监听器，但事件监听器只是简单地在LogCat控制台输出一行内容。在实际项目中我们可以在事件处理方法中做“任何”事情。不仅如此，上面绑定事件监听器的setOnItemClickListener、setOnItemSelectedListener方法都来自于AdapterView，因此这种事件处理机制完全适用于AdapterView的其他子类。

实例：扩展BaseAdapter实现不存储列表项的ListView

本实例将会通过扩展BaseAdapter来实现Adapter，扩展BaseAdapter可以取得对Adapter最大的控制权：程序要创建多少个列表项，每个列表项的组件都由开发者来决定。

本实例的布局文件非常简单，布局文件中只包含一个简单的ListView，布局文件代码如下。

程序清单：
codes\02\2.5\BaseAdapterTest\app\src\main\res\layout\main.xml

该实例的Activity将会扩展BaseAdapter来实现Adapter对象，Activity代码如下。

程序清单：
codes\02\2.5\BaseAdapterTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的关键在于粗体字代码部分，粗体字代码创建了一个BaseAdapter对象，扩展该对象需要重写如下4个方法。

➢ **getCount（）：** 该方法的返回值控制该Adapter将会包含多少个列表项。

➢ **getItem（int position）：** 该方法的返回值决定第position处的列表项的内容。

➢ **getItemId（int position）：** 该方法的返回值决定第position处的列表项的ID。

➢ **getView（int position，View convertView，ViewGroupparent）：** 该方法的返回值决定第 position 处的列表项组件。

上面4个方法中最重要的是第1个与第4个。

运行上面的程序，可以看到如图2.41所示的效果。

图2.41 不存储列表项的ListView

到目前为止，我们已经通过ListView介绍了4种实现Adapter的方式。从表面上看，此处只是在介绍ListView，但实际上这里介绍的知识完全适用于AdapterView的其他子类：GridView、Spinner、Gallery、AdapterViewFlipper等。因此，后面介绍这些组件的步骤依然是如下两步。

1 采用4种方式之一创建Adapter。

2 调用AdapterView的setAdapter（Adapter）方法设置Adapter即可。

2.5.3 自动完成文本框（AutoCompleteTextView）的功能与用法

自动完成文本框（AutoCompleteTextView）从EditText派生而出，实际上它也是一个文本编辑框，但它比普通编辑框多了一个功能：当用户输入一定字符之后，自动完成文本框会显示一个下拉菜单，供用户从中选择，当用户选择某个菜单项之后，AutoCompleteTextView 按用户选择自动填写该文本框。

AutoCompleteTextView除了可使用EditText提供的XML属性和方法之外，还支持如表2.20所示的常用XML属性及相关方法。

表2.21 AutoCompleteTextView支持的常用XML属性及相关方法

使用AutoCompleteTextView很简单，只要为它设置一个Adapter即可，该Adapter封装了AutoCompleteTextView预设的提示文本。

AutoCompleteTextView还派生了一个子类：MultiAutoCompleteTextView，该子类的功能与AutoCompleteTextView

基本相似，只是MultiAutoCompleteTextView允许输入多个提示项，多个提示项以分隔符分隔。MultiAutoCompleteTextView提供了setTokenizer（）方法来设置分隔符。

下面的界面布局文件中包含了AutoCompleteTextView和MultiAutoCompleteTextView。

程序清单：
codes\02\2.5\AutoCompleteTextViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了AutoCompleteTextView和MultiAutoCompleteTextView，接下来在程序中为它们绑定同一个Adapter，这意味着两个自动完成文本框的提示项完全相同，只是它们的表现行为略有差异。

Adapter负责为它提供提示文本。主程序如下。

程序清单：
codes\02\2.5\AutoCompleteTextViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码负责为AutoCompleteTextView、MultiAutoCompleteTextView设置同一个Adapter，并为MultiAutoCompleteTextView设置了分隔符。

运行上面的程序，将看到如图2.42所示的界面。

2.5.4 网格视图（GridView）的功能与用法

GridView用于在界面上按行、列分布的方式来显示多个组件。

图2.42 自动完成文本框

GridView和ListView有共同的父类：AbsListView，因此GridView和ListView具有很高的相似性，它们都是列表项。GridView与ListView的唯一区别在于：ListView只显示一列；而GridView可以显示多列。从这个角度来看，ListView相当于一种特殊的GridView，如果让GridView只显示一列，那么该GridView就变成了ListView。

与ListView类似的是，GridView也需要通过Adapter来提供显示的数据：开发者可以采用上面介绍的4种方式中的任意一种来创建Adapter。不管使用哪种方式，GridView与ListView的用法是基本一致的。

GridView提供的常用XML属性及相关方法如表2.22所示。

表2.22 GridView的常用XML属性及相关方法

注意：

使用GridView时一般都应该指定numColumns大于1；否则该属性的默认值为1。如果将该属性设为1，则意味着该GridView只有一列，那么GridView就变成了ListView。

表2.22所示的android：stretchMode属性支持如下几个属性值。

➢ **NO_STRETCH**：不拉伸。

➢ **STRETCH_SPACING**：仅拉伸元素之间的间距。

➢ **STRETCH_SPACING_UNIFORM**：表格元素本身、元素之间的间距一起拉伸。

➢ **STRETCH_COLUMN_WIDTH**：仅拉伸表格元素本身。

下面通过一个实例来介绍GridView的用法，本实例采用SimpleAdapter为GridView提供数据。

实例：带预览的图片浏览器

本实例将会采用一个GridView以行、列的形式来组织所有图片的预览视图。然后程序用一个ImageView来显示图片。

下面是本实例所使用的界面布局文件。

程序清单：
codes\02\2.5\GridViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中只是简单地定义了一个GridView、一个ImageView。定义GridView时指定了android：numColumns="4"，这意味着该网格包含4列。那么该GridView包含多少行呢？这是动态改变的一正如ListView到底包含多少行是由该ListView对应的Adapter所决定的，GridView到底包含多少行也是由Adapter决定的。

下面是主程序代码。

程序清单：
codes\02\2.5\GridViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的程序同样使用了SimpleAdapter对象作为GridView的Adapter，这个SimpleAdapter底层保证了一个长度为16的List集合一这意味着该

GridView一共需要显示16个组件，GridView总共有4列，因此该GridView一共包含4行。

上面的粗体字代码创建SimpleAdapter时指定使用R.layout.cell作为界面布局，因此还需要在app\src\main\res\layout目录下定义一个cell.xml界面布局文件。该文件中只包含一个简单的ImageView组件，此处不再给出该界面布局文件的代码。

运行上面的程序，将可以看到程序界面上方显示了16个图片预览视图（由GridView提供支持），单击任何一个图片预览视图，下面的ImageView将会显示对应的图片，如图2.43所示。

图2.43 G ridView效果

2.5.5 可展开的列表组件（ExpandableListView）

ExpandableListView是ListView的子类，它在普通ListView的基础上进行了扩展，它把应用中的列表项分为几组，每组里又可包含多个列表项。

ExpandableListView的用法与普通ListView的用法非常相似，只是ExpandableListView所显示的列表项应该由ExpandableListAdapter提供。ExpandableListAdapter也是一个接口，该接口下提供的类继承关系图如图2.44所示。

与Adapter类似的是，实现ExpandableListAdapter也有如下三种常用方式。

图2.44 ExpandableListAdapter及其子类的继承关系图

➢ 扩展BaseExpandableListAdapter实现ExpandableListAdapter。

➢ 使用SimpleExpandableListAdapter将两个List集合包装成ExpandableListAdapter。

➢ 使用SimpleCursorTreeAdapter将Cursor中的数据包装成SimpleCursorTreeAdapter。表2.23显示了ExpandableListView额外支持的常用XML属性。

表2.23 ExpandableListView额外支持的常用XML属性

下面的程序示范了如何通过自定义ExpandableListAdapter为ExpandableListView提供列表项。该程序的界面布局文件非常简单，只是在LinearLayout内定义了一个ExpandableListView，因此此处不再给出。

程序清单：
codes\02\2.5\ExpandableListViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序的关键代码就是扩展BaseExpandableListAdapter来实现ExpandableListAdapter，当扩展BaseExpandableListAdapter时，关键是实现如下4个方法。

➢ **getGroupCount（）**：该方法返回包含的组列表项的数量。

➢ **getGroupView（）**：该方法返回的View对象将作为组列表项。

➢ **getChildrenCount（）**：该方法返回特定组所包含的子列表项的数量。

➢ **getChildView（）：**该方法返回的 View 对象将作为特定组、特定位置的子列表项。

上面程序中的getChildView（）方法返回了一个普通TextView，因为每个子列表项都是一个普通文本框。而getGroupView（）方法则返回了一个LinearLayout对象，该LinearLayout对象里包含一个ImageView和一个TextView。因此，每个组列表项都由图片和文本组成。

图2.45 ExpandableListView效果

运行上面的程序，将看到如图2.45所示的界面。

2.5.6 Spinner的功能与用法

Spinner组件与Swing编程中的Spinner不同，此处的Spinner其实就是一个列表选择框。不过Android的列表选择框并不需要显示下拉列表，而是相当于弹出一个菜单供用户选择。

Spinner与Gallery都继承了AbsSpinner，AbsSpinner继承了AdapterView，因此它也表现出AdapterView的特征：只要为AdapterView提供Adapter即可。

Spinner支持的常用XML属性及相关方法如表2.24所示。

表2.24 Spinner支持的常用XML属性及相关方法

注意：

android：entries属性并不是Spinner定义的，而是在AbsSpinner中定义的，因此Gallery（继承了AbsSpinner）也支持该XML属性。

实例：让用户选择

如果开发者使用Spinner时已经可以确定列表选择框里的列表项，则完全不需要编写代码，只要为Spinner指定android：entries属性即可让Spinner正常工作；如果程序需要在运行时动态地决定Spinner的列表项，或者程序需要对Spinner的列表项进行定制，则可使用Adapter为Spinner提供列表项。

如下界面布局文件中定义了两个Spinner组件，其中第一个Spinner组件指定了android：entries属性，第二个Spinner组件没有指定android：entries属性，因此需要在Activity中为它设置Adapter。

程序清单：
codes\02\2.5\SpinnerTest\app\src\main\res\layout\main.xml

上面的界面布局文件中第一个Spinner使用@array/books指定数组资源，因此需要在app\src\main\res\value目录下使用XML文件来定义一份数组资源，该数组资源文件的内容如下。

程序清单：
codes\02\2.5\SpinnerTest\app\src\main\res\values\arrays.xml

上面的界面布局文件中第二个Spinner没有指定android：entries属性，Activity将会采用ArrayAdapter为该Spinner提供列表项。下面是Activity的代码。

程序清单：
codes\02\2.5\SpinnerTest\app\src\main\java\org\crazyit

\ui\MainActivity.java

上面的粗体字代码创建了一个简单的ArrayAdapter，并调用了Spinner的setAdapter（adapter）为之设置Adpater。运行该程序，将看到如图2.46所示的界面。

图2.46 S pinner的效果

Gallery与Spinner组件有共同的父类：AbsSpinner，表明Gallery和Spinner都是一个列表选择框。它们之间的区别在于，Spinner显示的是一个垂直的列表选择框，而Gallery显示的是一个水平的列表选择框。Gallery与Spinner还有一个区别：Spinner的作用是供用户选择，而Gallery则允许用户通过拖动来查看上一个、下一个列表项。

Gallery本身的用法非常简单一基本上与Spinner的用法相似，只要为它提供一个内容Adapter即可，该Adapter的getView（）方法所返回的View将作为Gallery列表的列表项。如果程序需要监控到Gallery选择项的改变，通过为Gallery添加OnItemSelectedListener监听器即可实现。

注意：

Android已经不再推荐使用Gallery组件，而是推荐使用其他水平滚动组件如HorizontalScrollView和ViewPager来代替Gallery组件。故本书不再详细介绍该界面组件的用法。

2.5.7 AdapterViewFlipper的功能与用法

AdapterViewFilpper继承了AdapterViewAnimator，它也会显示Adapter提供的多个View组件，但它每次只能显示一个View组件，程序可通过

showPrevious（）和showNext（）方法控制该组件显示上一个、下一个组件。

AdapterViewFilpper可以在多个View切换过程中使用渐隐渐显的动画效果。除此之外，还可以调用该组件的startFlipping（）控制它"自动播放"下一个View组件。

AdapterViewAnimator支持的XML属性如表2.25所示。

表2.25 AdapterViewAnimator支持的XML属性

AdapterViewFilpper额外支持的XML属性及相关方法如表2.26所示。

表2.26 AdapterViewFilpper支持的XML属性及相关方法

实例：自动播放的图片库

本实例示范了如何使用AdapterViewFlipper开发自动播放的图片库。该实例的界面上除了包含一个AdapterViewFilpper之外，还包含三个按钮，用于控制显示上一个、下一个图片和自动播放。为了控制AdapterViewFilpper要显示的多个列表项，程序为AdapterViewFilpper设置一个Adapter即可。

下面是该实例的XML布局文件。

程序清单：
codes\02\2.5\AdapterViewFlipperTest\app\src\main\res\layout\main.xml

上面的粗体字代码定义了一个AdapterViewFlipper组件，并为三个按钮指定了事件处理方法。该实例的Activity会采用扩展BaseAdapter的方式来实现自己的Adapter，并为AdapterViewFlipper组件设置Adapter。下面是该Activity的代码。

程序清单：
codes\02\2.5\AdapterViewFlipperTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码调用了AdapterViewFlipper的showPrevious（）、showNext（）方法来控制该组件显示上一个、下一个组件，并调用了startFlipping（）方法控制自动播放。

运行该程序，可以看到如图2.47所示的界面。

在图2.47中单击“自动播放”按钮，将可以看到AdapterViewFlipper每隔5秒更换一个图片，切换图片时会使用渐隐渐显效果。

图2.47 使用AdapterViewFlipper实现自动播放图片

2.5.8 StackView的功能与用法

StackView也是AdapterViewAnimator的子类，它也用于显示Adapter提供的一系列View。StackView将会以“堆叠（Stack）”的方式来显示多个列表项。

为了控制StackView显示的View组件，StackView提供了如下两种控制方式。

➢ 拖走StackView中处于顶端的View，下一个View将会显示出来。将上一个View拖进StackView，将使之显示出来。

➢ 通过调用StackView的showNext（）、showPrevious（）控制显示下一个、上一个组件。下面的实例示范了StackView的功能与用法。

实例：叠在一起的图片

该实例会使用StackView将照片叠在一起，并允许用户通过拖动或单击按钮来显示上一个、下一个图片。该实例的布局文件如下。

程序清单：
codes\02\2.5\StackViewTest\app\src\main\res\layout\main.xml

上面的布局文件中定义了一个StackView组件，该StackView将会以"叠"的方式显示多个View组件。与所有AdapterView类似的是，只要为StackView设置Adapter即可。

下面Activity将会创建一个SimpleAdapte作为StackView的Adapter，并为布局文件中的两个按钮的onClick事件提供处理方法。下面是该Activity的代码。

程序清单：
codes\02\2.5\StackViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

图2.48 StackView示例效果

上面的Activity代码中粗体字代码创建了一个SimpleAdapter，并将这个SimpleAdapter设置为该StackView的Adapter，这样该StackView将会显示该SimpleAdapter包含的一系列View组件。

运行该程序，并拖动ViewStack顶端的View组件，将可以看到如图2.48所示的效果。

图2.48中下方这个图片是正在被拖动的图片，这个图片显示的“动画”效果正是StackView支持的效果。

2.6 第5组UI组件：ProgressBar及其子类

ProgressBar也是一组重要的组件，ProgressBar本身代表了进度条组件，它还派生了两个常用的组件：SeekBar和RatingBar。ProgressBar及其子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

ProgressBar及其子类的继承关系图如图2.49所示。

图2.49 ProgressBar及其子类的继承关系图

2.6.1 进度条（ProgressBar）的功能与用法

进度条也是UI界面中一种非常实用的组件，通常用于向用户显示某个耗时操作完成的百分比。进度条可以动态地显示进度，因此避免长时间地执行某个耗时操作时，让用户感觉程序失去了响应，从而更好地提高用户界面的友好性。

Android支持多种风格的进度条，通过style属性可以为ProgressBar指定风格。该属性可支持如下几个属性值。

➢**@android：style/Widget.ProgressBar.Horizontal**：水平进度条。

➢**@android：style/Widget.ProgressBar.Inverse**：普通大小的环形进度条。

➢**@android：style/Widget.ProgressBar.Large**：大环形进度条。

➢**@android：style/Widget.ProgressBar.Large.Inverse**：大环形进度条。

➢**@android：style/Widget.ProgressBar.Small**：小环形进度条。

➢**@android：style/Widget.ProgressBar.Small.Inverse**：小环形进度条。

除此之外，ProgressBar还支持如表2.27所示的常用XML属性。

表2.27 ProgressBar支持的常用XML属性

表2.27中的android：progressDrawable用于指定进度条的轨道的绘制形式，该属性可指定为一个LayerDrawable对象（该对象可通过在XML文件中用<layer-list>元素进行配置）的引用。

ProgressBar提供了如下方法来操作进度。

➢ **setProgress（int）**：设置进度的完成百分比。

➢ **incrementProgressBy（int）**：设置进度条的进度增加或减少。当参数为正数时进度增加；当参数为负数时进度减少。

下面的程序简单示范了进度条的用法。该程序的界面布局文件只是定义了几个简单的进度条，并指定style属性为@android:style/Widget.ProgressBar.Horizontal，即水平进度条。界面布局文件如下。

程序清单：
codes\02\2.6\ProgressBarTest\app\src\main\res\layout\main.xml

上面的界面布局文件中先定义了三个环形进度条，这种环形进度条无法显示进度，它只是显示一个不断旋转的图片。布局文件的后面定义的两个进度条的最大值为100，第一个进度条的样式为水平进度条；第二个进度条的外观被定义为@drawable/my_bar，因此还需要在drawable中定义如下文件。

程序清单：
codes\02\2.6\ProgressBarTest\app\src\main\res\drawable\my_bar.xml

下面的主程序用一个填充数组的任务模拟了耗时操作，并以进度条来标识任务的完成百分比。

程序清单：
codes\02\2.6\ProgressBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码用于修改进度条的完成进度。运行上面的程序，将看到如图2.50所示的界面。

实例：显示在标题上的进度条

图2.50 进度条

还有一种进度条，可以直接在窗口标题上显示，这种进度条甚至不需要使用ProgressBar组件，它是直接由Activity的方法启用的。为了在窗口标题上显示进度条，需要经过如下两步。

1 调用Activity的requestWindowFeature（）方法，该方法根据传入的参数可启用特定的窗口特征。例如，传入Window.FEATURE_INDETERMINATE_PROGRESS在窗口标题上显示不带进度的进度条；传入Window.FEATURE_PROGRESS则显示带进度的进度条。

2 调用Activity的setProgressBarVisibility（boolean）或setProgressBarIndeterminate Visibility（boolean）方法即可控制进度条的显示和隐藏。

本实例的界面布局文件中仅仅定义了两个按钮，此处不再给出界面布局文件代码。该程序的Activity类代码如下。

程序清单：
codes\02\2.6\TitleProgressBar\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的①号代码控制窗口标题上显示带进度的进度条，而②号代码则控制窗口标题上显示不带进度的进度条。程序中两个按钮主要用于控制进度条的显示、隐藏。

如果程序启动窗口标题上显示带进度的进度条，则可看到如图2.51所示的界面。

如果程序启动窗口标题上显示不带进度的进度条，则可看到如图2.52所示的界面。

图2.51 显示在标题上的带进度的进度条

图2.52 显示在标题上的不带进度的进度条

提示：

可能读者已经发现，图2.51、图2.52所示的界面与前面实例的界面并不相同，这是因为前面实例都使用Android 5.0默认的主题：Material（Android 5.0的重要更新之一就是采用了Material Design），而这个默认主题将看不到标题上的进度条。因此，本实例在AndroidManifest.xml文件中将该应用的主题设为Holo。

2.6.2 拖动条（SeekBar）的功能与用法

拖动条和进度条非常相似，只是进度条采用颜色填充来表明进度完成的程度，而拖动条则通过滑块的位置来标识数值一而且拖动条允许用户拖动滑块来改变值，因此拖动条通常用于对系统的某种数值进行调节，比如调节音量等。

提示：

由于拖动条SeekBar继承了ProgressBar，因此ProgressBar所支持的XML属性和方法完全适用于SeekBar。

SeekBar允许用户改变拖动条的滑块外观，改变滑块外观通过如下属性来指定。

➢ **android：thumb**：指定一个Drawable对象，该对象将作为自定义滑块。

为了让程序能响应拖动条滑块位置的改变，程序可以考虑为它绑定一个OnSeekBarChangeListener监听器。

下面通过一个实例来示范SeekBar的功能和用法。

实例：通过拖动滑块来改变图片的透明度

该程序的界面布局中需要两个组件：一个ImageView用于显示图片，一个SeekBar用于动态改变图片的透明度。界面布局文件如下。

程序清单：
codes\02\2.6\SeekBarTest\app\src\main\res\layout\main.xml

上面程序中的粗体字代码定义了该拖动条的最大值、当前值都是255，并通过指定android：thumb属性来改变拖动条上滑块的外观。

该实例的主程序比较简单，程序只要为拖动条绑定一个监听器，当滑块位置发生改变时动态改变ImageView的透明度即可。主程序如下。

程序清单：
codes\02\2.6\SeekBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的粗体字代码就是监听拖动条上滑块位置发生改变的关键代码：当滑块位置发生改变时，ImageView的透明度将变为该拖动条的当前数值。运行上面的程序，将看到如图2.53所示的界面。

2.6.3 星级评分条（RatingBar）的功能与用法

图2.53 拖动滑块改变图片的透明度

星级评分条与拖动条有相同的父类：AbsSeekBar，因此它们十分相似。实际上星级评分条与拖动条的用法、功能都十分接近：它们都允许用户通过拖动来改变进度。RatingBar与SeekBar的最大区别在于：RatingBar通过星星来表示进度。

表2.28显示了RatingBar所支持的常见XML属性。

表2.28 RatingBar支持的常见XML属性

为了让程序能响应星级评分条评分的改变，可以考虑为它绑定一个OnRatingBarChangeListener监听器。

下面通过一个实例来示范RatingBar的功能和用法。

实例：通过星级改变图片的透明度

该程序其实就是前一个程序的简单改变，只是将前一个程序中的SeekBar组件改为使用RatingBar。下面是界面布局文件中关于RatingBar的代码片段。

程序清单：
codes\02\2.6\RatingBarTest\app\src\main\res\layout\main.xml

上面的界面布局文件中指定了该星级评分条的最大值为255，当前进度为255—其中两个属性都来自于ProgressBar组件，这没有任何问题，因为RatingBar本来就是一个特殊的ProgressBar。

主程序只要为RatingBar绑定事件监听器，即可监听星级评分条的星级改变。下面的主程序为星级评分条绑定监听器的代码。

程序清单：
codes\02\2.6\RatingBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

由于上面定义RatingBar时指定了android：stepSize="0.5"，因此该星级评分条中星级的最小变化为0.5，也就是最少要变化半个星级。运行上面的程序，将看到如图2.54所示的界面。

图2.54 星级评分条改变图片的透明度

2.7 第6组UI组件：ViewAnimator及其子类

图2.55 ViewAnimator及其子类的继承关系图

ViewAnimator是一个基类，它继承了FrameLayout，因此它表现出FrameLayout的特征，可以将多个View组件“叠”在一起。ViewAnimator额外增加的功能正如它的名字所暗示的，ViewAnimator可以在View切换时表现出动画效果。

ViewAnimator及其子类的继承关系图如图2.55所示。

ViewAnimator及其子类也是一组非常重要的UI组件，这种组件的主要功能是增加动画效果，从而使界面更加“炫”。使用ViewAnimator时可以指定如表2.29所示的常见XML属性。

表2.29 ViewAnimator支持的常见XML属性

在实际项目中往往会使用ViewAnimator的几个子类，下面一一详细介绍。

2.7.1 ViewSwitcher的功能与用法

ViewSwitcher代表了视图切换组件，它本身继承了FrameLayout，因此可以将多个View层叠在一起，每次只显示一个组件。当程序控制从一个View切换到另一个View时，ViewSwitcher支持指定动画效果。

为了给ViewSwitcher添加多个组件，一般通过调用ViewSwitcher的setFactory （ViewSwitcher.ViewFactory）方法为之设置ViewFactory，并由该ViewFactory为之创建View即可。

下面通过一个实例来介绍ViewFactory的用法。

实例：仿Android系统Launcher界面

Android早期版本的Launcher界面是上下滚动的，Android 5.0的Launcher界面已经实现了分屏、左右滚动（可能是模仿 iOS 的操作习惯吧），本实例就是通过 ViewSwitcher 来实现Android 5.0的分屏、左右滚动效果。

为了实现该效果，程序主界面考虑使用ViewSwitcher来组合多个GridView，每个GridView代表一个屏幕的应用程序，GridView中每个单元格显示一个应用程序的图标和程序名。

该程序的主界面布局文件如下。

程序清单：
codes\02\2.7\ViewSwitcherTest\app\src\main\res\layout\main.xml

上面的界面布局文件中只是定义了一个ViewSwitcher组件和两个按钮，这两个按钮分别用于控制该ViewSwitcher显示上一屏、下一屏的程序列表。

提示：

这个程序采用了按钮来控制滚动到上一屏、下一屏，这种操作方式显得不够“酷”，实际上完全可以实现通过手势来滚动屏幕。如果希望实现通过手势来滚动屏幕的效果，建议参考本书第8章关于手势的知识。

该实例的重点在于为该ViewSwitcher设置ViewFactory对象，并且当用户单击“<”和“>”两个按钮时控制ViewSwitcher显示“上一屏”和“下一屏”的应用程序。

该程序会考虑使用扩展BaseAdapter的方式为GridView提供Adapter，而本实例的关键就是根据用户单击的按钮来动态计算该BaseAdapter应该显示哪些程序列表。该程序的Activity代码如下。

程序清单：
codes\02\2.7\ViewSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的程序使用screenNo保存当前正在显示第几屏的程序列表。该程序的关键在于粗体字代码部分，该粗体字代码创建了一个

BaseAdapter对象，这个BaseAdapter会根据screenNo动态计算该Adapter总共包含多少个列表项（如getCount（）方法所示），会根据screenNo计算每个列表项的数据（如getItem（int position）方法所示）。

BaseAdapter的getView（）只是简单加载了R.layout.labelicon布局文件，并使用当前列表项的图片数据填充R.layout.labelicon布局文件中的ImageView，使用当前列表项的文本数据填充R.layout.labelicon布局文件中的TextView。下面是R.layout.labelicon布局文件的代码。

程序清单：
codes\02\2.7\ViewSwitcherTest\app\src\main\res\layout\labelicon.xml

当用户单击“>”按钮时，程序的事件处理方法将会控制ViewSwitcher调用showNext（）方法显示下一屏的程序列表一而且此时screenNo被加1，因而Adapter将会动态计算下一屏的程序列表，再将该Adapter传给ViewSwitcher接下来要显示的GridView。

为了实现ViewSwitcher切换View时的动画效果，程序的事件处理方法中调用了ViewSwitcher的setInAnimation（）、setOutAnimation（）方法来设置动画效果。本程序不仅利用了Android系统提供的两个动画资源，还自行提供了动画资源。

其中R.anim.slide_in_right动画资源对应的代码如下。

程序清单：
codes\02\2.7\ViewSwitcherTest\app\src\main\res\anim\slide_in_right.xml

R.anim.slide_out_left动画资源对应的代码如下。

程序清单：
codes\02\2.7\ViewSwitcherTest\app\src\main\res\anim\slide_out_left.xml

运行上面的程序，可以看到如图2.56所示的效果。

在图2.56所示界面中，当用户单击底端的“<”或“>”按钮时，将可以看到ViewSwitcher切换屏幕时的动画效果。

图2.56 仿Android系统的Launcher界面

2.7.2 图像切换器（ImageSwitcher）的功能与用法

ImageSwitcher继承了ViewSwitcher，因此它具有与ViewSwitcher相同的特征：可以在切换View组件时使用动画效果。ImageSwitcher继承了ViewSwitcher，并重写了ViewSwitcher的showNext（）、showPrevious（）方法，因此ImageSwitcher使用起来更加简单。使用ImageSwitcher只要如下两步即可。

1 为ImageSwitcher提供一个ViewFactory，该ViewFactory生成的View组件必须是ImageView。

2 需要切换图片时，只要调用ImageSwitcher的setImageDrawable（Drawable drawable）、setImageResource（int resid）和setImageURI（Uri uri）方法更换图片即可。

提示：

ImageSwitcher与ImageView的功能有点相似，它们都可用于显示图片，区别在于ImageSwitcher的效果更炫，它可以指定图片切换时的动画效果。

下面通过一个实例来介绍ImageSwitcher的用法。

实例：支持动画的图片浏览器

本实例是对前面的GridViewTest实例的修改，该实例使用ImageSwitcher代替了原有的ImageView。该实例的界面布局文件如下。

程序清单：
codes\02\2.7\ImageSwitcherTest\app\src\main\res\layout\main.xml

上面界面布局文件中的粗体字代码定义了一个ImageSwitcher，并通过android：inAnimation和android：outAnimation指定了图片切换时的动画效果。

接下来Activity代码需要为该ImageSwitcher设置ViewFactory，并让该ViewFactory的makeView（）方法返回ImageView。下面是该Activity的代码。

程序清单：
codes\02\2.7\ImageSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码重写了ViewFactory的makeView（）方法，该方法返回一个ImageView对象，这样该ImageSwitcher即可正常工作。该Activity还为GridView绑定了事件监听器，当用户单击GridView或选中GridView的指定单元格时，ImageSwitcher切换为显示对应的图片。

运行上面的实例，可以看到如图2.57所示的界面。

在图2.57所示界面中，用户单击或选中GridView中某个图标时，下面的ImageSwitcher将会切换为显示对应的图片，图片切换时会使用动画效果。

图2.57 ImageSwitcher测试

2.7.3 文本切换器（TextSwitcher）的功能与用法

TextSwitcher继承了ViewSwitcher，因此它具有与ViewSwitcher相同的特征：可以在切换View组件时使用动画效果。与ImageSwitcher相似的是，使用TextSwitcher也需要设置一个ViewFactory。与ImageSwitcher不同的是，TextSwitcher所需的ViewFactory的makeView（）方法必须返回一个TextView组件。

提示：

TextSwitcher与TextView的功能有点相似，它们都可用于显示文本内容，区别在于TextSwitcher的效果更炫，它可以指定文本切换时的动画效果。

下面的实例在界面上定义了一个TextSwitcher，界面布局文件如下。

程序清单：
codes\02\2.7\TextSwitcherTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了一个TextSwitcher，并指定了文本切换时的动画效果。接下来Activity只要为该TextSwitcher设置ViewFactory，该TextSwitcher即可正常工作。

下面是该Activity的代码。

程序清单：
codes\02\2.7\TextSwitcherTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的粗体字代码重写了ViewFactory的makeView（）方法，该方法返回了一个TextView，这样即可让TextSwitcher正常工作。当程序要切换TextSwitcher显示文本时，调用TextSwitcher的setText（）方法修改文本即可—如上面的①号代码所示。

图2.58 文本切换器

运行上面的程序，可以看到如图2.58所示的界面。

单击图2.58所示界面上的文本切换器，界面将会显示下一个文本，文本切换时将会使用动画效果。

2.7.4 ViewFlipper的功能与用法

ViewFlipper组件继承了ViewAnimator，它可调用addView（View v）添加多个组件，一旦向ViewFlipper中添加了多个组件之后，ViewFlipper就可使用动画控制多个组件之间的切换效果。

ViewFlipper与前面介绍的AdapterViewFlipper有较大的相似性，它们可以控制组件切换的动画效果。它们的区别是：ViewFlipper需要开发者通过addView（View v）添加多个View，而AdapterViewFlipper则只要传入一个Adapter，Adapter将会负责提供多个View。因此，ViewFlipper可以指定与AdapterViewFlipper相同的XML属性。

实例：自动播放的图片库

该实例与前面介绍的AdapterViewFlipper实例非常相似，区别只是该实例直接定义了该ViewFlipper所包含的View组件。下面是该实例的界面布局文件。

程序清单：
codes\02\2.7\ViewFlipperTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了一个ViewFlipper，并在该ViewFlipper中定义了三个ImageView，这意味着该ViewFlipper包含了三个子组件。接下来在Activity代码中即可调用ViewFlipper的showPrevious（）、showNext（）等方法控制ViewFlipper显示上一个、下一个子组件。为了控制组件切换时的动画效果，还需要调用ViewFlipper的setInAnimation（）、setOutAnimation（）方法设置动画效果。

下面是该Activity的代码。

程序清单：
codes\02\2.7\ViewFlipperTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码就是控制ViewFlipper切换组件的动画效果，以及控制ViewFlipper切换组件的关键代码。运行该程序，可以看到如图2.59所示的界面。

单击图2.59所示界面下方的按钮，即可看到图片上、下切换，而且组件切换时将可以看到动画切换效果。

2.8 各种杂项组件

除了前面介绍的6组UI组件之外，Android还有如下一些常用的杂项组件，掌握这些杂项组件也是开发Android应用必需的技能。

图2.59 ViewFlipper控制图片切换

2.8.1 使用Toast显示提示信息框

Toast是一种非常方便的提示消息框，它会在程序界面上显示一个简单的提示信息。这个提示信息框用于向用户生成简单的提示信息。它具有如下两个特点。

➢ Toast提示信息不会获得焦点。

➢ Toast提示信息过一段时间会自动消失。

使用Toast来生成提示消息也非常简单，只要如下几个步骤即可。

1 调用Toast的构造器或makeText（）静态方法创建一个Toast对象。

2 调用Toast的方法来设置该消息提示的对齐方式、页边距等。

3 调用Toast的show（）方法将它显示出来。

Toast的功能和用法都比较简单，大部分时候它只能显示简单的文本提示；如果应用需要显示诸如图片、列表之类的复杂提示，一般建议使用对话框来完成；如果开发者确实想通过Toast来完成，也是可以的，此时就需要调用Toast构造器创建实例，再调用setView（）方法设置该Toast显示的View组件。该方法允许开发者自己定义Toast显示的内容。

下面以一个实例程序来示范Toast的用法。

实例：带图片的消息提示

本实例程序非常简单，它在用户界面上显示了两个按钮，其中一个按钮用于激发普通的Toast提示；另一个按钮用于激发带图片的Toast提示。这意味着开发者必须调用该Toast对象的setView（）方法来改变该Toast对象的内容View。

本程序的用户界面很简单，只有两个普通按钮，故不再给出界面布局文件代码。本程序的Java代码如下。

程序清单：
codes\02\2.8\ToastTest\app\src\main\java\org\crazyit\ui\MainActivity.java

图2.60 带图片的消息提示

上面的程序比较简单：第一个按钮被单击时，程序只是简单地创建了一个Toast对象，并把它显示出来，因此单击第一个按钮只是看到一个简单的Toast提示；当第二个按钮被单击时，程序先创建了一个Toast对象，并调用该Toast对象的setView（）方法改变了该消息提示的内容。因此单击第二个按钮时将看到带图片的消息提示，如图2.60所示。

2.8.2 日历视图（CalendarView）组件的功能和用法

日历视图（CalendarView）可用于显示和选择日期，用户既可选择一个日期，也可通过触摸来滚动日历。如果希望监控该组件的日期改变，则可调用CalendarView的setOnDateChangeListener（）方法为此组件的点击事件添加事件监听器。

使用CalendarView时可指定如表2.30所示的常见XML属性及相关方法。

表2.30 CalendarView支持的常见XML属性及相关方法

下面通过实例来示范CalendarView组件的功能与用法。

实例：选择您的生日

本实例的重点是界面布局文件中添加了一个CalendarView组件。下面是该实例的界面布局文件。

程序清单：
codes\02\2.8\CalendarViewTest\app\src\main\res\layout\main.xml

上面的界面布局文件中粗体字代码定义了一个CalenarView组件，并设置该组件总共显示4周，以每周的星期二作为第一天。

为了监听用户选择日期的事件，本实例在Activity代码中调用该组件的setOnDateChangeListener（）方法来添加事件监听器。该实例的Activity代码如下。

程序清单：
codes\02\2.8\CalendarViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面Activity的粗体字代码实现了onSelectedDayChange（），当用户选择的日期发生改变时将会激发该方法。运行上面的程序，当用户选

择的日期发生改变时，将可以看到程序显示如图2.61所示的界面。

2.8.3 日期、时间选择器（DatePicker和TimePicker）的功能和用法

DatePicker和TimePicker是两个比较易用的组件，它们都从FrameLayout派生而来，其中DatePicker供用户选择日期；而TimePicker则供用户选择时间。

图2.61 通过CalendarView组件选择日期

DatePicker和TimePicker在FrameLayout的基础上提供了一些方法来获取当前用户所选择的日期、时间；如果程序需要获取用户选择的日期、时间，则可通过为 DatePicker 添加OnDateChangedListener进行监听、为TimePicker添加OnTimerChangedListener进行监听来实现。使用DatePicker时可指定如表2.31所示的常见XML属性。

表2.31 DatePicker支持的常见XML属性

下面以一个让用户选择日期、时间的实例来示范DatePicker和TimePicker的功能与用法。

实例：用户选择日期、时间

为了让用户能选择日期，本应用需要同时使用DatePicker和TimePicker两个组件，并为它们分别绑定监听器。下面是本应用的界面布局文件。

程序清单：
codes\02\2.8\ChooseDate\app\src\main\res\layout\main.xml

上面的界面布局中添加了一个DatePicker、一个TimePicker，这两个组件供用户选择日期、时间。除此之外，还包含了一个EditText，该组件用于显示用户选择的日期、时间。

本应用的主程序如下。

程序清单：
codes\02\2.8\ChooseDate\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的两行粗体字代码就是分别为DatePicker、TimePicker绑定事件监听器的代码，DatePicker和TimePicker绑定监听器的方式略有不同，但本质还是一样的。一旦为DatePicker、TimePicker绑定了监听器，当用户通过这两个组件来选择日期、时间时，监听器就会被触发—监听器负责使用EditText来显示用户选择的日期、时间。运行上面的程序，将看到如图2.62所示的界面。

提示：

对于DatePicker、TimePicker两个UI组件而言，使用Material风格时，虽然视觉效果非常好，但笔者测试时暂不能选择日期、时间，因此本实例同样改为使用Holo风格才可看到如图2.62所示的效果。

图2.62 选择日期、时间

2.8.4 数值选择器（NumberPicker）的功能与用法

数值选择器用于让用户输入数值，用户既可以通过键盘输入数值，也可以通过拖动来选择数值。使用该组件常用如下三个方法。

➢ **setMinValue（int minVal）**：设置该组件支持的最小值。

➢ **setMaxValue（int maxVal）**：设置该组件支持的最大值。

➢ **setValue（int value）**：设置该组件的当前值。

下面通过一个实例来介绍NumberPicker的功能与用法。

实例：选择您意向的价格范围

在该实例中，程序将会使用两个NumberPicker来让用户选择价格，其中第一个NumberPicker用于选择低价；第二个NumberPicker用于选择高价。下面是该实例的界面布局文件。

程序清单：
codes\02\2.8\NumberPickerTest\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了两个NumberPicker，接下来Activity代码需要为这两个NumberPicker设置最小值、最大值，并为它们绑定事件监听器。下面是该Activity的代码。

程序清单：
codes\02\2.8\NumberPickerTest\org\crazyit\ui\MainActivity.java

上面两段粗体字代码的控制逻辑基本是相似的，它们都调用了NumberPicker的setMinValue（）、setMaxValue（）、setValue（）来设置该数值选择器的最小值、最大值和当前值。除此之外，程序还

为两个数值选择器绑定了事件监听器：当它们的值发生改变时，将会激发相应的事件处理方法。

图2.63 使用NumberPicker选择数值

运行该程序，并通过NumberPicker选择数值，将可以看到如图2.63所示的界面。

2.8.5 搜索框（SearchView）的功能与用法

SearchView是搜索框组件，它可以让用户在文本框内输入文字，并允许通过监听器监控用户输入，当用户输入完成后提交搜索时，也可通过监听器执行实际的搜索。

使用SearchView时可使用如下常用方法。

➢ **setIconifiedByDefault （boolean iconified）**：设置该搜索框默认是否自动缩小为图标。

➢ **setSubmitButtonEnabled（boolean enabled）**：设置是否显示搜索按钮。

➢ **setQueryHint（CharSequence hint）**：设置搜索框内默认显示的提示文本。

➢ **setOnQueryTextListener （SearchView.OnQueryTextListener listener）**：为该搜索框设置事件监听器。

如果为SearchView增加一个配套的ListView，则可以为SearchView增加自动完成的功能。如下实例示范了SearchView的功能与用法。

实例：搜索

该实例的界面布局文件中定义了一个SearchView和ListView，其中ListView用于为SearchView显示自动补齐列表。界面布局文件如下。

程序清单：
codes\02\2.8\SearchViewTest\app\src\main\res\layout\main.xml

上面的布局文件中定义了一个SearchView组件，并为该SearchView组件定义了一个ListView组件，该ListView组件用于为SearchView组件显示自动完成列表。

下面是该实例对应的Activity代码。

程序清单：
codes\02\2.8\SearchViewTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码就是控制 SearchView 的关键代码，第二行粗体字代码为SeachView设置了事件监听器，并为该SearchView启用了搜索按钮。接下来程序重写了onQueryTextChange（）、onQueryTextSubmit（）两个方法，这两个方法用于为SearchView的事件提供响应。

运行该程序，将可以看到如图2.64所示的界面。

2.8.6 选项卡（TabHost）的功能和用法

TabHost是一种非常实用的组件，TabHost可以很方便地在窗口上放置多个标签页，每个标签页相当于获得了一个与外部容器相同大小的组件摆放区域。通过这种方式，就可以在一个容器里放置更多组件，例

如许多手机系统都会在同一个窗口中定义多个标签页来显示通话记录，包括“未接电话”、“已接电话”、“呼出电话”等。

图2.64 使用SearchView执行搜索

与TabHost结合使用的还有如下组件。

➢ **TabWidget**：代表选项卡的标题条。

➢ **TabSpec**：代表选项卡的一个Tab页面。

TabHost仅仅是一个简单的容器，它提供了如下两个方法来创建、添加标签页。

➢ **newTabSpec（String tag）**：创建选项卡。

➢ **addTab（TabHost.TabSpec tabSpec）**：添加选项卡。

使用TabHost的一般步骤如下。

1 在界面布局文件中定义TabHost组件，并为该组件定义该选项卡的内容。

2 Activity应该继承TabActivity。

3 调用TabActivity的getTabHost（）方法获取TabHost对象。

4 通过TabHost对象的方法来创建、添加选项卡。

除此之外，TabHost还提供了一些方法来获取当前选项卡。获取当前View的方法，具体可以参考API文档。如果程序需要监控TabHost里当前标签页的改变，则可以为它设置TabHost.OnTabChangeListener监听器。

下面通过一个简单的实例来示范选项卡的用法。

实例：通话记录界面

本实例程序使用TabHost定义一个标签容器，并使用三个LinearLayout来定义标签页（实际上可以使用任何View组件来定义标签页）。该程序的界面布局文件如下。

程序清单：
codes\02\2.8\TabHostTest\app\src\main\res\layout\main.xml

请注意上面的布局文件中粗体字代码部分，从上面的布局文件可以发现，TabHost容器内部需要组合两个组件：TabWidget和FrameLayout，其中TabWidget用于定义选项卡的标题条；FrameLayout则用于“层叠”组合多个选项页面。不仅如此，上面的布局文件中这三个组件的ID也有要求。

➢ TabHost的ID应该为@android：id/tabhost。

➢ TabWidget的ID应该为@android：id/tabs。

➢ FrameLayout的ID应该为@android：id/tabcontent。

上面这三个ID并不是开发者自己定义的，而是引用了Android系统已有的ID。

接下来主程序即可加载该布局资源，并将布局文件中的三个Tab页面添加到该TabHost容器中。该Activity代码如下。

程序清单：
codes\02\2.8\TabHostTest\app\src\main\java\org\crazyit

\ui\MainActivity.java

上面程序中的粗体字代码就是为TabHost创建并添加Tab页面的代码。上面的程序一共添加了三个标签页，因此类似粗体字的代码一共写了三次。其中第二个标签的标题上还添加了一个图标（该主题下不会显示图标）。

运行上面的程序，将看到如图2.65所示的界面。

上面的程序调用了TabHost.TabSpec对象的setContent（int viewId）方法来设置标签页内容。除此之外，还可调用setContent（Intent intent）方法来设置标签页内容，Intent还可用于启动其他Activity—这意味着TabHost.TabSpec可直接装载另一个Activity。本书后面有关于Intent的详细介绍。

图2.65 通话记录界面

注意：

最新版本的Android平台已经不再推荐使用TabActivity，而是推荐使用Fragment来代替TabActivity，本书第4章会详细介绍Fragment的相关知识。

2.8.7 滚动视图（ScrollView）的功能和用法

ScrollView由FrameLayout派生而出，它就是一个用于为普通组件添加滚动条的组件。ScrollView里最多只能包含一个组件，而ScrollView的作用就是为该组件添加垂直滚动条。

提示：

ScrollView的作用和Swing编程中的JScrollPane非常相似，它们甚至不能被称为真正的容器，它们只是为其他容器添加滚动条。

在默认情况下，ScrollView只是为其他组件添加垂直滚动条，如果应用需要添加水平滚动条，则可借助于另一个滚动视图—HorizontalScrollView来实现。ScrollView与HorizontalScrollView的功能基本相似，只是前者添加垂直滚动条，后者添加水平滚动条。

下面以一个例子来示范ScrollView、HorizontalScrollView的用法。

实例：可垂直和水平滚动的视图

本实例程序通过在ScrollView里嵌套HorizontalScrollView，来为应用的界面同时添加水平滚动条、垂直滚动条。下面是该应用的界面布局文件。

程序清单：
codes\02\2.8\ScrollViewTest\app\src\main\res\layout\main.xml

上面的界面布局实现了界面的垂直、水平同时滚动。使用Activity显示上面的界面布局，将看到如图2.66所示的界面。

2.8.8 Notification的功能与用法

图2.66 水平、垂直滚动

Notification是显示在手机状态栏的通知—手机状态栏位于手机屏幕的最上方，那里一般显示手机当前的网络状态、电池状态、时间等。Notification所代表的是一种具有全局效果的通知，程序一般通过NotificationManager服务来发送Notification。

提示：

NotificationManager是一个重要的系统服务，该API位于图1.1中的应用程序框架层，应用程序可通过NotificationManager向系统发送全局通知。

Android为Notification增加了Notification.Builder类，通过该类允许开发者更轻松地创建Notification对象。Notification.Builder提供了如下常用方法。

➢ **setDefaults（）**：设置通知LED灯、音乐、振动等。

➢ **setAutoCancel（）**：设置点击通知后，状态栏自动删除通知。

➢ **setContentTitle（）**：设置通知标题。

➢ **setContentText（）**：设置通知内容。

➢ **setSmallIcon（）**：为通知设置图标。

➢ **setLargeIcon（）**：为通知设置大图标。

➢ **setTick（）**：设置通知在状态栏的提示文本。

➢ **setContentIntent（）**：设置点击通知后将要启动的程序组件对应的PendingIntent。

发送Notification很简单，按如下步骤进行即可。

1 调用getSystemService（NOTIFICATION_SERVICE）方法获取系统的NotificationManager服务。

2 通过构造器创建一个Notification对象。

3 为Notification设置各种属性。

4 通过NotificationManager发送Notification。

下面通过实例来介绍Notification的功能和用法。

实例：加薪通知

本实例示范了如何通过NotificationManager来发送、取消Notification。本实例的界面很简单，只是包含两个普通按钮，分别用于发送Notification和取消Notification。本实例程序的Java代码如下。

程序清单：
codes\02\2.8\NotificationTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码用于为Notification设置各种属性，包括Notification的图标、标题、发送时间等。

除此之外，上面的程序还通过setDefaults（）方法为Notification设置了声音提示、振动提示、闪光灯等。该属性支持如下属性值。

- **DEFAULT_SOUND**：设置使用默认声音。
- **DEFAULT_VIBRATE**：设置使用默认振动。
- **DEFAULT_LIGHTS**：设置使用默认闪光灯。
- **ALL**：设置使用默认声音、振动、闪光灯。

如果不想使用默认设置，也可以使用如下代码：

接下来的①号代码用于为该Notification设置事件信息，设置事件信息时传入了一个PendingIntent对象，该对象里封装了一个Intent，这意味着单击该Notification时将会启动该Intent对应的程序。

图2.67 发送通知

运行上面的程序，单击“发送Notification”按钮，将可以看到手机屏幕上方出现了一个Notification。将状态栏向下拖动，将可以看到Notification的详情，如图2.67所示。

图2.67所示的Notification还关联了一个Activity：OtherActivity，因此当用户单击“普通通知”时即可启动OtherActivity—OtherActivity是一个十分简单的程序，故此处不再介绍。

由于上面的程序指定了该Notification要启动OtherActivity，因此一定不要忘记在AndroidManifest.xml文件中声明该Activity。而且上面的程序还需要访问系统闪光灯、振动器，这也需要在AndroidManifest.xml文件中声明权限。也就是增加如下代码片段：

2.9 第7组UI组件：对话框

Android提供了丰富的对话框支持，它提供了如下4种常用的对话框。

➢ **AlertDialog**：功能最丰富、实际应用最广的对话框。

➢ **ProgressDialog**：进度对话框，这个对话框只是对进度条的包装。

➢ **DatePickerDialog**：日期选择对话框，这个对话框只是对DatePicker的包装。

➢ **TimePickerDialog**：时间选择对话框，这个对话框只是对TimePicker的包装。

上面4种对话框中功能最强、用法最灵活的就是AlertDialog，因此应用也非常广泛，而其他三种对话框都是它的子类。图2.68显示了它们的继承关系图。

图2.68 AlertDialog及其子类的继承关系图

本节将会详细介绍各种对话框的功能和用法。

2.9.1 使用AlertDialog创建对话框

AlertDialog的功能很强大，它可以生成各种内容的对话框。但实际上AlertDialog生成的对话框总有如图2.69所示的结构。

图2.69 对话框的结构

从图2.69可以看出，AlertDialog生成的对话框可分为如下4个区域。

➢ 图标区

➢ 标题区

➢ 内容区

➢ 按钮区

从上面对话框的结构来看，创建一个对话框需要经过如下几步。

1 创建AlertDialog.Builder对象。

2 调用AlertDialog.Builder的setTitle（）或setCustomTitle（）方法设置标题。

3 调用AlertDialog.Builder的setIcon（）方法设置图标。

4 调用AlertDialog.Builder的相关设置方法设置对话框内容。

5 调用AlertDialog.Builder的setPositiveButton（）、setNegativeButton（）或setNeutralButton（）方法添加多个按钮。

6 调用AlertDialog.Builder的create（）方法创建AlertDialog对象，再调用AlertDialog对象的show（）方法将该对话框显示出来。

在上面的6个步骤中，第4步是最灵活的，AlertDialog允许创建各种内容的对话框。归纳起来，AlertDialog提供了如下6种方法来指定对话框的内容。

➢ **setMessage（）**：设置对话内容为简单文本。

➢ **setItems（）**：设置对话框内容为简单列表项。

➢ **setSingleChoiceItems（）**：设置对话框内容为单选列表项。

➢ **setMultiChoiceItems（）**：设置对话框内容为多选列表项。

➢ **setAdapter（）**：设置对话框内容为自定义列表项。

➢ **setView（）**：设置对话框内容为自定义View。

下面通过几个实例来介绍AlertDialog的用法。

实例：显示提示消息的对话框

本程序的界面非常简单，程序界面上定义了1个文本框和6个按钮，每次用户单击一个按钮时，将会显示不同类型的对话框。

本实例的界面只包含一个简单的文本框和几个按钮。当用户单击按钮时将会显示对话框。由于界面十分简单，故此处不再给出界面布局文件。

本程序将会通过上面介绍的6步来创建AlertDialog对话框。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码为该对话框设置了图标、标题等属性。上面程序中还调用了setPositiveButton（）和setNegativeButton（）方法添加按钮。这两个方法的代码如下。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

除此之外，AlertDialog.Builder还提供了如下方法来添加按钮。

➢ **setNeutralButton（CharSequence text，DialogInterface.OnClick Listener listener）**：添加一个装饰性按钮。

因此Android的对话框一共可以生成三个按钮。

运行上面的程序，单击“简单对话框”按钮，将看到如图2.70所示的界面。

图2.70 简单对话框

实例：简单列表项对话框

调用AlertDialog.Builder的setItems（）方法即可设置简单列表项对话框，调用该方法时需要传入一个数组或数组资源的资源ID。

下面的程序调用setItems（）来设置简单列表项对话框。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码调用AlertDialog.Builder的setItems（）方法为对话框设置了多个列表项，此处生成了4个普通列表项。

运行上面的程序，单击“简单列表项对话框”按钮，程序将显示如图2.71所示的界面。

实例：单选列表项对话框

只要调用AlertDialog.Builder的setSingleChoiceItems（）方法即可创建带单选列表项的对话框。调用setSingleChoiceItems（）方法时既可传入数组作为参数，也可传入Cursor（相当于数据库查询结果集）作为参数，还可传入ListAdapter作为参数。如果传入ListAdapter作为参数，则由ListAdapter来提供多个列表项组件。

图2.71 简单列表项对话框

下面的程序调用了setSingleChoiceItems（）方法来创建带单选列表项的对话框。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\craz

yit\ui\MainActivity.java

运行上面的程序，单击“单选列表项对话框”按钮，应用显示如图2.72所示的界面。

实例：多选列表项对话框

只要调用AlertDialog.Builder的setMultiChoiceItems（）方法即可创建一个多选列表项对话框。调用setMultiChoiceItems（）方法时既可传入数组作为参数，也可传入Cursor（相当于数据库查询结果集）作为参数。

下面的程序调用了setMultiChoiceItems（）方法来创建带多选列表项的对话框。

图2.72 单选列表项对话框

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

调用AlertDialog.Builder的setMultiChoiceItems（）方法添加多选列表项时，需要传入一个boolean[]参数，该参数有两个作用：①设置初始化时选中哪些列表项；②该boolean[]类型的参数还可用于动态地获取多选列表中列表项的选中状态。

运行上面的程序，然后单击“多选列表项对话框”按钮，程序将显示如图2.73所示的界面。

实例：自定义列表项对话框

AlertDialog.Builder提供了一个setAdapter（）方法来设置对话框的内容，该方法需要传入一个Adapter参数，这样即可由该Adapter负责提供多个列表项组件。

图2.73 多选列表项对话框

提示：

不仅setAdapter（）方法可以接受Adapter作为参数，setSingleChoice（）方法也可以接受Adapter作为参数，这意味着调用setSingleChoice（）方法也可实现自定义列表项对话框。

下面的程序调用了setAdapter（）方法来创建自定义列表项对话框。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码调用setAdapter（）方法时传入了一个ArrayAdapter，该ArrayAdapter将会负责提供多个自定义列表项。如果需要，完全可以用创建SimpleAdapter对象或扩展BaseAdapter的方式来实现Adapter，并作为参数传入setAdapter（）方法。

运行上面的程序，然后单击“自定义列表项对话框”按钮，程序将显示如图2.74所示的界面。

图2.74 自定义列表项对话框

实例：自定义View对话框

AlertDialog.Builder的setView（）方法可以接受一个View组件，该View组件将会作为对话框的内容，通过这种方式，开发者可以“随心所欲”地定制对话框的内容—因为在Android界面编程中，一切都是View。

本实例将会定义一个登录对话框，为实现该登录对话框，先定义登录界面布局，该界面布局文件如下。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\res\layout\login.xml

上面的界面布局文件定义了登录用的三个输入框：输入用户名的文本框、输入密码的密码框、输入电话号码的输入框。接下来在应用程序中调用AlertDialog.Builder的setView（View view）方法让对话框显示该输入界面。

该程序与前面介绍的列表对话框程序比较相似，只是将原来的调用setItems（）设置列表项，改为调用setView（）来设置自定义视图。下面给出该程序的关键代码。

程序清单：
codes\02\2.9\AlertDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的第一行粗体字代码显式加载了/app\src\main\res/layout/login.xml 文件，并返回该文件对应的

TableLayout作为View。接下来程序调用了AlertDialog.Builder的setView（）方法来显示上一行代码所获得的TableLayout。

运行上面的程序，然后单击“自定义View对话框”按钮，应用程序将显示如图2.75所示的界面。

2.9.2 对话框风格的窗口

还有一种自定义对话框的方式，这种对话框本质上依然是窗口，只是把显示窗口的Activity的风格设为对话框风格。

图2.75 自定义View对话框

下面的程序只需定义一个简单的界面布局，该界面布局里包含一个ImageView和一个Button。接下来程序使用Activity来显示该界面布局。

程序清单：
codes\02\2.9\DialogTheme\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的程序仅仅是为界面上的按钮绑定了一个监听器，当该按钮被单击时结束该Activity。

接下来在AndroidManifest.xml文件中指定该窗口以对话框风格显示，也就是在该清单文件中使用如下配置片段：

上面的粗体字代码指定DialogTheme使用对话框风格进行显示。运行上面的程序，将看到如图2.76所示的界面。

图2.76 对话框风格的窗口

2.9.3 使用PopupWindow

PopupWindow可以创建类似对话框风格的窗口，使用PopupWindow创建对话框风格的窗口只要如下两步即可。

1 调用PopupWindow的构造器创建PopupWindow对象。

2 调用PopupWindow的showAsDropDown（View v）将PopupWindow作为v组件的下拉组件显示出来；或调用PopupWindow的showAtLocation（）方法将PopupWindow在指定位置显示出来。

下面的程序示范了如何使用PopupWindow创建对话框风格的窗口。主程序中只有一个简单的按钮，用户单击该按钮时将会显示PopupWindow；其中PopupWindow负责加载并显示前一个示例的窗口界面。

程序清单：
codes\02\2.9\PopupWindowTest\app\src\main\java\org\crazyit\ui\MainActivity.java

图2.77 PopupWindow示例效果

上面程序中的第一行粗体字代码用于创建PopupWindow对象，接下来的两行粗体字代码分别示范了两种显示PopupWindow的方式：以下拉方式显示和在指定位置显示，这就是在程序中创建并显示PopupWindow的关键代码。程序中①号粗体字代码负责销毁、隐藏

PopupWindow对象—当用户单击PopupWindow中的“关闭”按钮时，该PopupWindow将会关闭。

运行上面的程序，将可以看到如图2.78所示的结果。

从图2.77所示效果不难看出，PopupWindow非常适合显示一些需要浮动显示的内容。

2.9.4 使用DatePickerDialog、TimePickerDialog

DatePickerDialog与TimePickerDialog的功能比较简单，用法也简单，只要如下两步即可。

1 通过new关键字创建DatePickerDialog、TimePickerDialog实例，调用它们的show（）方法即可将日期选择对话框、时间选择对话框显示出来。

2 为DatePickerDialog、TimePickerDialog绑定监听器，这样可以保证用户通过DatePickerDialog、TimePickerDialog设置事件时触发监听器，从而通过监听器来获取用户设置的事件。

下面程序的界面上定义了两个按钮，其中一个按钮用于打开日期选择对话框；另一个按钮用于打开时间选择对话框。该程序的界面布局文件非常简单，这里不再给出。该程序的Java代码如下。

程序清单：
codes\02\2.9\DateDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的两段粗体字代码就是创建并显示DatePickerDialog、TimePickerDialog的关键代码。运行上面的程序，如果单击“设置时间”按钮，系统将会显示如图2.78所示的时间选择对话框。

如果用户单击“设置日期”按钮，系统将会显示日期选择对话框。需要指出的是，日期选择对话框、时间选择对话框只是供用户来选择日期、时间，对Android的系统日期、时间没有任何影响。

图2.78 时间选择对话框

提示：

Android暂时还没有公开设置系统日期、时间的API，如果需要在程序中设置Android系统日期、时间，目前所知的方式都需要重新编译Android系统源代码，比较烦琐。

2.9.5 使用ProgressDialog创建进度对话框

ProgressDialog代表了进度对话框，程序只要创建ProgressDialog实例，并将它显示出来就是一个进度对话框。使用ProgressDialog创建进度对话框有如下两种方式。

➢ 如果只是创建简单的进度对话框，那么调用ProgressDialog提供的静态show（）方法显示对话框即可。

➢ 创建ProgressDialog，然后调用方法对对话框里的进度条进行设置，设置完成后将对话框显示出来即可。

为了对进度对话框里的进度条进行设置，ProgressDialog包含了如下常用的方法。

➢ **setIndeterminate（boolean indeterminate）**：设置对话框里的进度条不显示进度值。

➢ **setMax（int max）**：设置对话框里进度条的最大值。

➢ **setMessage（CharSequence message）**：设置对话框里显示的消息。

➢ **setProgress（int value）**：设置对话框里进度条的进度值。

➢ **setProgressStyle（int style）**：设置对话框里进度条的风格。

下面的程序界面也很简单，界面上只有三个简单的按钮，当用户单击不同按钮时系统将会启动不同的进度对话框。其中第三个按钮激发的进度对话框比较复杂，该对话框使用填充数组来模拟耗时任务，随着任务进行不断更新进度对话框上进度的显示。

程序清单：
codes\02\2.9\ProgressDialogTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序为三个按钮定义了事件处理方法，其中一个按钮的事件处理方法调用了ProgressDialog的静态show（）方法创建并显示对话框，如上面①号粗体字代码所示，这也是使用对话框的最简单方式；第二个按钮的事件处理方法先创建了ProgressDialog对象，再调用该对话框的show（）方法将它显示出来，如上面②号粗体字代码所示；第三个按钮的事件处理方法也是先创建ProgressDialog对象，并设置其中对话框的相关属性，再调用show（）方法将它显示出来，如上面③号粗体字代码所示。

单击程序中的第一个按钮，将看到如图2.79所示的界面。

单击第三个按钮，将可以看到如图2.80所示的界面。

图2.79 环形风格的进度条

图2.80 水平风格的进度条

2.10 菜单

菜单在桌面应用中使用十分广泛，几乎所有的桌面应用都有菜单。菜单在手机应用中的使用减少了不少（主要受到手机屏幕大小制约），但依然有不少手机应用会添加菜单。

与桌面应用的菜单不同，Android应用中的菜单默认是看不见的，只有当用户按下手机上的“MENU”键（位于模拟器右边的物理键盘上）时，系统才会显示该应用关联的菜单，这种菜单叫选项菜单（Option Menu）。

注意：

从Android 3.0开始，Android并不要求手机设备上必须提供MENU按键，可能部分Android手机将不再提供MENU按键。在这样的情况下，Android推荐使用ActionBar来代替菜单。下一节会介绍Android的ActionBar支持。

Android应用同样支持上下文菜单（ContextMenu），当用户一直按住某个组件时，该组件所关联的上下文菜单就会显示出来。

2.10.1 选项菜单和子菜单（SubMenu）

为了让读者感受Android应用中菜单的外观和功能，先简单看一下Android系统自带的选项菜单，按如下两步进行。2 单击模拟器右边的“MENU”按键，将可以在手机屏幕下方看到系统默认的选项菜单，如图2.81所示。

从图2.81可以看出，Android的选项菜单默认是看不见的，当用户按下“MENU”按键时程序菜单将会出现在屏幕下方。

Android系统的菜单支持主要通过4个接口来体现，图2.82显示了Android菜单支持的4个接口。

图2.81 选项菜单

图2.82 Android系统的菜单支持接口

从图2.82可以看出，Menu接口只是一个父接口，该接口下有如下两个子接口。

➢ **SubMenu**：它代表一个子菜单。可以包含1～N个MenuItem（形成菜单项）。

➢ **ContextMenu**：它代表一个上下文菜单。可以包含1～N个MenuItem（形成菜单项）。

Android的不同菜单具有如下特征。

➢ **选项菜单**：选项菜单不支持勾选标记，并且只显示菜单的“浓缩（condensed）”标题。

➢ **子菜单（SubMenu）**：不支持菜单项图标，不支持嵌套子菜单。

➢ **上下文菜单（ContextMenu）**：不支持菜单快捷键和图标。

Menu接口定义了如下方法来添加子菜单或菜单项。

➢ **MenuItem add（int titleRes）**：添加一个新的菜单项。

➢ **MenuItem add（int groupId，int itemId，int order，CharSequence title）**：添加一个新的处于groupId组的菜单项。

➢ **MenuItem add（int groupId，int itemId，int order，int titleRes）**：添加一个新的处于groupId组的菜单项。

➢ **MenuItem add（CharSequence title）**：添加一个新的菜单项。

➢ **SubMenu addSubMenu（int titleRes）**：添加一个新的子菜单。

➢ **SubMenu addSubMenu（int groupId，int itemId，int order，int titleRes）**：添加一个新的处于groupId组的子菜单。

按键（返回桌面），系统返回桌面。

1 单击模拟器右边的

➢ **SubMenu addSubMenu（CharSequence title）**：添加一个新的子菜单。

➢ **SubMenu addSubMenu（int groupId，int itemId，int order，CharSequence title）**：添加一个新的处于groupId组的子菜单。

上面的方法归纳起来就是两个：add（）方法用于添加菜单项；addSubMenu（）用于添加子菜单。这些重载方法的区别是：是否将子菜单、菜单项添加到指定菜单组中，是否使用资源文件中的字符串资源来设置标题。

SubMenu继承了Menu，它就代表一个子菜单，额外提供了如下常用方法。

➢ **SubMenu setHeaderIcon（Drawable icon）**：设置菜单头的图标。

➢ **SubMenu setHeaderIcon（int iconRes）**：设置菜单头的图标。

➢ **SubMenu setHeaderTitle（int titleRes）**：设置菜单头的标题。

➢ **SubMenu setHeaderTitle（CharSequence title）**：设置菜单头的标题。

➢ **SubMenu setHeaderView（View view）**：使用View来设置菜单头。

掌握了上面Menu、SubMenu、MenuItem的用法之后，接下来就可通过它们来为Android应用添加菜单或子菜单了。添加菜单或子菜单的步骤如下。

1 重写Activity的onCreateOptionsMenu（Menu menu）方法，在该方法里调用Menu对象的方法来添加菜单项或子菜单。

2 如果希望应用程序能响应菜单项的单击事件，那么重写Activity的onOptionsItemSelected （MenuItem mi）方法即可。

下面的程序示范了如何为Android应用添加菜单和子菜单。该程序的界面布局很简单，故不再给出界面布局文件。该程序的Java代码如下。

程序清单：
codes\02\2.10\MenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码就是添加三个菜单的代码，三个菜单中有两个带有子菜单，而且程序还为子菜单设置了图标、标题等。运行上面的程序，单击“MENU”按键，将看到程序下方显示如图2.83所示的菜单。

图2.83所示菜单中“字体大小”包含子菜单，“普通菜单项”只是一个菜单项，“字体颜色”也包含子菜单，如果开发者单击“字体颜色”菜单，将看到屏幕上显示如图2.84所示的子菜单。

图2.83 选项菜单

图2.84 子菜单

提示：

Android 5.0采用了Material主题，这种主题不会显示子菜单的标题，因此图2.84中的子菜单看不到菜单标题。

由于程序重写了onOptionsItemSelected（MenuItem mi）方法，因此当用户单击指定菜单项时，程序可以为菜单项的单击事件提供响应。

前面已经介绍过，如果程序要监听菜单项的单击事件，可以通过重写onOptionsItemSelected（MenuItem item）方法来实现—每当用户单击任意菜单时，该方法都会被触发。如果开发者需要针对不同菜单提供响应，就需要在onOptionsItemSelected（MenuItem item）方法中进行判断了，如上面的程序使用switch 语句进行了判断。由于程序需要在onOptionsItemSelected（MenuItem item）方法中准确判断到底是哪个菜单项被单击了，因此添加菜单项时通常应为每个菜单项指定ID。

2.10.2 使用监听器来监听菜单事件

除了重写onOptionsItemSelected（MenuItem item）方法来为菜单单击事件编写响应之外，Android同样允许开发者为不同菜单项分别绑定监听器。为菜单项绑定监听器的方法为：

➢ **setOnMenuItemClickListener(MenuItem.OnMenuItemClickListener menuItemClickListener)**

在这种方式下，我们可以采用简单方法来添加菜单项，无须为每个菜单项指定ID。

一般来说，通过重写onOptionsItemSelected（MenuItem mi）方法来使处理菜单的单击事件更加简捷，因为所有的事件处理代码都控制在该方法内，只需要判断到底单击了哪个菜单项就行。通过为每个菜单项绑定事件监听器使得代码更加臃肿，因此，一般并不推荐为每个菜单项分别绑定监听器。

2.10.3 创建多选菜单项和单选菜单项

如果希望所创建的菜单项是单选菜单项或多选菜单项，则可以调用MenuItem的如下方法。

➢ **setCheckable（boolean checkable）：**设置该菜单项是否可以被勾选。

调用上面的方法后，菜单项默认是多选菜单项。

如果希望将一组菜单里的菜单项都设为可勾选的菜单项，则可调用如下方法。

➢ **setGroupCheckable（int group，boolean checkable，boolean exclusive）：**设置group组里的所有菜单项是否可勾选；如果将 exclusive 设为 true，那么它们将是一组单选菜单项。

除此之外，Android还为MenuItem提供了如下方法来设置快捷键。

➢ **setAlphabeticShortcut（char alphaChar）**：设置字母快捷键。

➢ **setNumericShortcut（char numericChar）**：设置数字快捷键。

➢ **setShortcut（char numericChar，char alphaChar）**：同时设置两种快捷键。

2.10.4 设置与菜单项关联的Activity

有些时候，应用程序需要单击某个菜单项时启动其他Activity（包括其他Service）。对于这种需求，Android甚至不需要开发者编写任何事件处理代码，只要调用MenuItem的setIntent（Intent intent）方法即可—该方法把该菜单项与指定Intent关联到一起，当用户单击该菜单项时，该Intent所代表的组件将会被启动。

如下程序示范了如何通过菜单项来启动指定Activity。该程序几乎不包含任何界面组件，因此不给出界面布局文件。该程序的Java文件如下。

程序清单：
codes\02\2.10\ActivityMenu\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的粗体字代码表示单击指定菜单项时启动OtherActivity，因此程序还应该定义OtherActivity，并在AndroidManifest.xml文件中添加如下代码来声明OtherActivity：

运行上面的程序，打开“启动程序”菜单，可以看到如图2.85所示的子菜单。

图2.85 启动Activity的菜单项

单击图2.85所示子菜单中的“查看Swift”，即可启动另一个Activity：OtherActivity。

2.10.5 上下文菜单

Android用ContextMenu来代表上下文菜单，为Android应用开发上下文菜单与开发选项菜单的方法基本相似，因为ContextMenu继承了Menu，因此程序可用相同的方法为它添加菜单项。

当然，开发上下文菜单与开发选项菜单的区别在于：开发上下文菜单不是重写onCreateOptionsMenu（Menu menu）方法，而是重写onCreateContextMenu （ContextMenu menu，View source，ContextMenu.ContextMenuInfo menuInfo）方法。其中source参数代表触发上下文菜单的组件。

开发上下文菜单的步骤如下。

1 重写Activity的onCreateContextMenu（ContextMenu menu，View source，ContextMenu.Context MenuInfo menuInfo）方法。

2 调用Activity的registerForContextMenu（View view）方法为view组件注册上下文菜单。

3 如果希望应用程序能为菜单项提供响应，则可以重写onContextItemSelected（MenuItem mi）方法，或为指定菜单项绑定事件监听器。

ContextMenu提供了如下方法，同样可以为上下文菜单设置图标、标题等。

➢ **ContextMenu setHeaderIcon（Drawable icon）**：为上下文菜单设置图标。

➢ **ContextMenu setHeaderIcon（int iconRes）**：为上下文菜单设置图标。

➢ **ContextMenu setHeaderTitle（int titleRes）**：为上下文菜单设置标题。

下面的程序示范了如何开发上下文菜单。该程序的用户界面同样很简单，故不再给出界面布局文件。下面是该程序的Java代码。

程序清单：
codes\02\2.10\ContextMenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面的程序重写了onCreateContextMenu（ContextMenu menu，View source，ContextMenu.Context MenuInfo menuInfo）方法，该方法的内部为程序创建了一个上下文菜单。

程序在①号代码处调用registerForContextMenu（txt）为txt组件（一个文本框组件）注册了上下文菜单，这意味着当用户长按该组件时显示上下文菜单。上下文菜单如图2.86所示。

2.10.6 使用XML文件定义菜单

图2.86 上下文菜单

Android提供了两种创建菜单的方式，一种是在Java代码中创建，一种是使用XML资源文件定义。上面的示例都是在Java代码中创建菜单，

在Java代码中定义菜单存在如下不足。

➢ 在Java代码中定义菜单、菜单项，必然导致程序代码臃肿。

➢ 需要程序员采用硬编码方式为每个菜单项分配ID，为每个菜单组分配ID，这种方式将导致应用可扩展性、可维护性降低。

一般推荐使用XML资源文件来定义菜单，这种方式可以提供更好的解耦。

提示：

使用Android Studio创建Android项目时，Android Studio默认就会在res目录下新建menu子目录，并在该子目录下提供menu_main.xml菜单资源文件。由此可见，Android推荐使用XML资源文件来定义菜单。

菜单资源文件通常应该放在app\src\main\res\menu目录下，菜单资源的根元素通常是<menu.../>，<menu.../>元素无须指定任何属性。<menu.../>元素内可包含如下子元素。

➢ **<item.../>元素**：定义菜单项。

➢ **<group.../>子元素**：将多个<item.../>定义的菜单项包装成一个菜单组。

<group.../>子元素用于控制整组菜单的行为，该元素可指定如下常用属性。

➢ **checkableBehavior**：指定该组菜单的选择行为。可指定为none（不可选）、all（多选）和single（单选）三个值。

➢ **menuCategory**：对菜单进行分类，指定菜单的优先级。有效值为container、system、secondary和alternative。

➢ **visible**：指定该组菜单是否可见。

➢ **enable**：指定该组菜单是否可用。

<item.../>元素用于定义菜单项，<item.../>元素又可包含<menu.../>元素，位于<item.../>元素内部的<menu.../>就代表了子菜单。

<item.../>元素可指定如下常用属性。

➢ **android**：**id**：为菜单项指定一个唯一标识。

➢ **android**：**title**：指定菜单项的标题。

➢ **android**：**icon**：指定菜单项的图标。

➢ **android**：**alphabeticShortcut**：为菜单项指定字符快捷键。

➢ **android**：**numericShortcut**：为菜单项指定数字快捷键。

➢ **android**：**checkable**：设置该菜单项是否可选。

➢ **android**：**checked**：设置该菜单项是否已选中。

➢ **android**：**visible**：设置该菜单项是否可见。

➢ **android**：**enable**：设置该菜单项是否可用。

在程序中定义了菜单资源后，接下来还是重写onCreateOptionsMenu（用于创建选项菜单）、onCreateContextMenu（用于创建上下文菜单）方法，在这些方法中调用MenuInflater对象的inflate方法加载指定资源对应的菜单即可。

接下来将会开发一个使用XML资源文件定义菜单的实例，本实例将会把前面开发的菜单示例程序改为使用XML资源文件定义菜单。

实例：使用XML资源文件定义菜单

本实例包含两种菜单：选项菜单和上下文菜单，其中选项菜单对应的XML资源文件如下。

程序清单：
codes\02\2.10\MenuResTest\app\src\main\res\menu\menu_main.xml

上面的菜单资源文件的<menu.../>元素里包含三个<item.../>子元素，这表明该菜单里包含三个菜单项。其中第一个、第三个菜单项都包含子菜单。

接下来为该应用定义上下文菜单的资源文件，代码如下。

程序清单：
codes\02\2.10\MenuResTest\app\src\main\res\menu\context.xml

定义了上面两份菜单资源之后，接下来即可在Activity的onCreateOptionsMenu、onCreateContextMenu方法中加载这两份菜单资源。下面是程序中加载并显示两份菜单资源的Java代码。

程序清单：
codes\02\2.10\MenuResTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的两行粗体字代码就是加载选项菜单资源、上下文菜单资源的关键代码。

需要指出的是，可勾选菜单的勾选状态必须由程序代码来控制，所以上面程序中的①、②号代码自己控制了菜单项勾选状态的切换。这一点是笔者感到很奇怪的地方：为何Android系统不为我们处理这些细节？这些细节给我们编程带来了不少烦琐的处理。

从上面的程序可以看出，如果使用XML资源文件来定义菜单，就像使用布局文件来定义应用程序界面一样，Android应用的Java代码就会简单很多，因此可维护性更好。

归纳起来，使用XML资源文件定义菜单有如下两个好处。

➢ XML资源文件不仅负责定义应用界面，也负责定义菜单，这样可把所有界面相关的内容交给XML文件管理，而Java代码的功能更集中。

➢ 后期更新、维护应用时，如果需要更新、维护菜单，打开、编辑XML文件即可，避免对Java文件的修改。

该应用程序的运行效果与前面介绍的菜单示例的效果大致相同，此处不再给出。

2.10.7 使用PopupMenu创建弹出式菜单

PopupMenu代表弹出式菜单，它会在指定组件上弹出PopupMenu，在默认情况下，PopupMenu会显示在该组件的下方或者上方。PopupMenu可增加多个菜单项，并可为菜单项增加子菜单。

使用PopupMenu创建菜单的步骤非常简单，只要如下步骤即可。

1 调用new PopupMenu（Context context，View anchor）创建下拉菜单，anchor代表要激发该弹出菜单的组件。

2 调用MenuInflater的inflate（）方法将菜单资源填充到PopupMenu中。

3 调用PopupMenu的show（）方法显示弹出式菜单。

下面的示例示范了使用PopupMenu的功能和用法。这个示例的界面布局文件中仅有一个普通按钮，界面布局文件非常简单，故此处不再给出代码。

该示例的Activity代码如下。

程序清单：
codes\02\2.10\PopupMenuTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的第一行粗体字代码创建了一个PopupMenu对象，第二行粗体字代码指定将该R.menu.popup_menu菜单资源填充到PopupMenu中，这样即可实现当用户单击界面上的按钮时弹出popup菜单。

运行该程序，单击程序界面上的按钮，将可以看到如图2.87所示的界面。

图2.87 使用PopupMenu开发弹出式菜单

2.11 使用活动条（ActionBar)

活动条（ActionBar）是Android 3.0的重要更新之一。ActionBar位于传统标题栏的位置，也就是显示在屏幕的顶部。ActionBar可显示应用的图标和Activity标题—也就是前面应用程序的顶部显示的内容。除此之外，ActionBar的右边还可以显示活动项（Action Item）。

归纳起来，ActionBar提供了如下功能。

➢ 显示选项菜单的菜单项（将菜单项显示成Action Item）。

➢ 使用程序图标作为返回Home主屏或向上的导航操作。

➢ 提供交互式View作为Action View。

➢ 提供基于Tab的导航方式，可用于切换多个Fragment。

➢ 提供基于下拉的导航方式。

2.11.1 启用ActionBar

最新的Android版本已经默认启用了ActionBar，因此只要在AndroidManifest.xml文件的SDK配置中指定该应用的目标版本高于11（Android 3.0的版本号），默认就会启用ActionBar。例如如下配置：

指定该应用程序可以部署在Android 4.2平台上，同时兼容Android 2.3.3及更高版本。如果Android版本高于3.0，该应用将会启用ActionBar。

如果希望关闭ActionBar，则可以设置该应用的主题为Xxx.NoActionBar。例如如下配置片段：

上面的粗体字代码指定该应用关闭ActionBar功能。一旦关闭了ActionBar，该Android应用将不能使用ActionBar。

在实际项目中，通常推荐使用代码来控制ActionBar的显示、隐藏。ActionBar提供了如下方法来控制显示、隐藏。

➢ **show（）：**显示ActionBar。

➢ **hide（）：**隐藏ActionBar。

如下示例示范了如何通过代码来控制ActionBar的显示、隐藏。

该示例的界面布局文件中只定义了两个按钮，界面布局文件非常简单，故此处不再给出界面布局文件代码。该示例的Activity代码如下。

程序清单：
codes\02\2.11\ActionBarTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的第一行粗体字代码调用了getActionBar（）方法获取该Activity关联的ActionBar。接下来就可以调用ActionBar的方法来控制它的显示、隐藏了。

运行该程序并单击“隐藏ActionBar”按钮，将可以看到如图2.88所示的界面。

图2.88 隐藏ActionBar

从图2.88可以看出，此处程序顶端的ActionBar已经“消失”了。

2.11.2 使用ActionBar显示选项菜单项

前面介绍菜单时已经指出，Android不再强制要求手机必须提供MENU按键，这样可能导致用户无法打开选项菜单。为了解决这个问题，Android已经提供了ActionBar作为解决方案，ActionBar可以将选项菜单显示成Action Item。

从Android 3.0开始，MenuItem新增了如下方法。

➢ **setShowAsAction（int actionEnum）**：该方法设置是否将该菜单项显示在ActionBar上，作为Action Item。

该方法支持如下参数值。

➢ **SHOW_AS_ACTION_ALWAYS**：总是将该MenuItem显示在ActionBar上。

➢ **SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW**：将该Action V iew折叠成普通菜单项。

➢ **SHOW_AS_ACTION_IF_ROOM**：当ActionBar位置足够时才显示MenuItem。

➢ **SHOW_AS_ACTION_NEVER**：不将该MenuItem显示在ActionBar上。

➢ **SHOW_AS_ACTION_WITH_TEXT**：将该MenuItem显示在ActionBar上，并显示该菜单项的文本。

正如前面看到的，实际项目推荐使用XML资源文件来定义菜单，因此Android允许在XML菜单资源文件中为<item.../>元素指定如下属性。

➢ **android：showAsAction**：该属性的作用类似于setShowAsAction（int actionEnum）方法。因此该属性也能支持类似于上面的属性值。

下面的程序对前面关于菜单的示例稍作修改，只是在定义菜单资源文件时为<item.../>元素增加了android：showAsAction属性。下面是本示例的菜单资源文件代码。

程序清单：
codes\02\2.11\ActionItemTest\app\src\main\res\menu\menu_main.xml

上面三行粗体字代码为选项菜单项增加了android：showAsAction属性，该属性可控制将这些菜单项显示在ActionBar上。再次运行该示例，将可以看到如图2.89所示的Action Item。

正如图2.89所显示的，手机顶部的ActionBar的空间是有限的，当选项菜单的菜单项很多时，ActionBar无法同时显示所有的选项菜单项，Android将会根据不同手机设备采取不同行为。

图2.89 使用ActionBar显示选项菜单项

➢ 对于有 MENU 按键的手机，用户单击MENU按键即可看到剩余的选项菜单项。

➢ 对于没有MENU 按键的手机，ActionBar 会在最后显示一个折叠图标，用户单击该折叠图标将会显示剩余的选项菜单项，类似于以前单击MENU按键后出现“More”按钮。

2.11.3 启用程序图标导航

为了将应用程序图标转变成可以点击的图标，可以调用ActionBar的如下方法。

➢ **setDisplayHomeAsUpEnabled（boolean showHomeAsUp）**：设置是否将应用程序图标转变成可点击的图标，并在图标上添加一个向左的箭头。

➢ **setDisplayOptions（int options）**：通过传入int类型常量来控制该ActionBar的显示选项。

➢ **setDisplayShowHomeEnabled（boolean showHome）**：设置是否显示应用程序图标。

➢ **setHomeButtonEnabled（boolean enabled）**：设置是否将应用程序图标转变成可点击的按钮。

下面的程序将该Activity的程序图标转变成可点击的图标，并控制单击该图标时直接返回程序的主Activity。该程序的界面布局文件、菜单资

源文件保持不变。下面是该Activity的代码。

程序清单：
codes\02\2.11\ActionHomeTest\app\src\main\java\org\crazyit\ui\MainActivity.java

上面程序中的前两行粗体字代码的任意一行都可将ActionBar上的应用程序图标转变成可点击的图标。接下来程序为点击事件绑定了事件监听器，当用户单击ID为android.R.id.home的Action Item（应用程序图标的ID）时，程序使用Intent返回了应用程序的FirstActivity，如上面程序中的第二段粗体字代码所示。

提示：

上面的示例中还定义了FirstActivity、SecondActivity两个Activity，关于开发、定义Activity的详细步骤，请参考本书第4章。

2.11.4 添加Action View

ActionBar上除了可以显示普通的Action Item之外，还可以显示普通的UI组件。为了在ActionBar上添加Action View，可以来用如下两种方式。

➢ 定义Action Item时使用android：actionViewClass属性指定Action View的实现类。

➢ 定义Action Item时使用android：actionLayout属性指定Action View对应的视图资源。

实例：“标题”上的时钟

本实例将会在菜单资源文件中定义两个Action Item，但这两个Action Item都是使用Action View，而不是普通的Action Item。资源文件代码如下。

程序清单：
codes\02\2.11\ActionViewTest\app\src\main\res\menu\menu_main.xml

上面资源文件中的两行粗体字代码分别采用两种方式定义了Action View，这样就可以在ActionBar上显示自定义View。其中第一行粗体字代码指定了Action View的实现类为SearchView；第二行粗体字代码指定该Action View对应的界面布局资源为@layout/clock。该布局文件代码如下。

程序清单：
codes\02\2.11\ActionViewTest\app\src\main\res\layout\clock.xml

该界面布局文件仅仅定义了一个AnalogClock，这表明该ActionBar的第二个Action View只是一个模拟时钟。

图2.90 添加Action View

运行该程序，可以看到如图2.90所示的界面。

2.11.5 使用ActionBar实现Tab导航

ActionBar还有常用的功能：实现Tab导航。ActionBar在顶端生成多个Tab标签，当用户点击某个Tab标签时，系统根据用户点击事件导航到

指定Tab页面。

为了使用ActionBar实现Tab导航，按如下步骤进行即可。

1 调用ActionBar的setNavigationMode（ActionBar.NAVIGATION_MODE_TABS）方法设置使用Tab导航方式。

2 调用ActionBar的addTab（）方法添加多个Tab标签，并为每个Tab标签添加事件监听器。

在实际项目中为了更好地展现Tab导航效果，ActionBar通常会与Fragment结合使用，因此这里先简单介绍Fragment的用法。

Fragment 是Android 3.0新增的重要API，Fragment 相当于Activity片段（Fragment本来就是片段的意思），我们通常使用单独的Activity组合多个Fragment，这样即可在一个Activity中创建多个用户界面。除此之外，也可让多个Activity复用同一个Fragment。总之，Fragment相当于Activity的模块化区域。

Fragment拥有自己的生命周期，它也可以接收、处理属于它自身的事件，并允许Activity运行期间动态地添加、删除Fragment。

Fragment 允许定义自己的布局，也可通过生命周期回调方法定义自己的行为，这一点Fragment非常像Activity。

与开发Activity类似的是，开发者自定义的Fragment也需要继承Fragment，并重写它的生命周期方法，通常会重写Fragment的onCreateView（）生命周期方法。

提示：

关于Fragment的生命周期及更详细的介绍，请参考本书第4章。

下面通过实例来介绍ActionBar结合Fragment实现Tab导航。

实例：ActionBar结合Fragment实现Tab导航

该实例的界面布局文件非常简单，该布局文件中只是定义了一个简单的容器，该容器甚至没有太多要求，既可使用LinearLayout，也可使用RelativeLayout……甚至是普通ViewGroup，该容器只是用于盛装Fragment。该实例的界面布局文件如下。

程序清单：
codes\02\2.11\ActionBar_TabNav\res\layout\main.xml

上面的界面布局文件只是定义了一个LinearLayout作为容器，接下来Activity将会使用该容器动态地盛装Fragment。下面是该Activity的代码。

程序清单：
codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\MainActivity.java

上面的第一段粗体字代码设置ActionBar使用Tab导航，为该ActionBar添加了三个Tab标签，并为每个Tab标签都设置了事件监听器。

注意：

Android 5.0已将ActionBar导航这种方式设为过时的方式，Android 5.0推荐使用更通用的导航方式，就是前面介绍的利用ActionBar上的图标导航，并未提供更有效的导航方式，因此ActionBar依然是一种非常重要的导航方式，所以本书依然会介绍这种方式。

当用户单击ActionBar上的指定Tab标签时，系统将会激发该监听器的onTabSelected（）方法，因此上面的第二段粗体字代码实现了

onTabSelected（）方法，并在该方法中根据用户选中的Tab标签替换新的Fragment。

上面的实例用到了一个DummyFragment，这是一个简单的Fragment，它只是显示一个简单的TextView组件。下面是该DummyFragment的代码。

程序清单：
codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\DummyFragment.java

运行该实例，可以看到如图2.91所示的效果。

需要指出的是，Android Studio为ActionBar的Tab导航提供了支持，当使用Android Studio创建Android项目时，Android Studio的向导工具在创建Tabbed Activity时会看到如图2.92所示的界面。

图2.91 ActionBar结合Fragment实现Tab导航

图2.92 为ActionBar选择导航方式

从图2.92可以看出，Android Studio为Activity提供了三种导航方式，其中Action Bar Tabs导航方式就是刚刚介绍的第一个实例。下面将会详细介绍另外两种导航方式。

实例：Android 3.0以前的Fragment支持

Fragment非常实用，Android也为3.0版本以前的平台增加了Fragment支持，只是该Fragment不是继承android.app.Fragment，而是继承android.support.v4.app.Fragment。

除此之外，Android还为该android.support.v4.app.Fragment提供了如下配套类。

➢ **FragmentActivity**：在早期版本上使用Fragment必须借助于FragmentActivity的支持，只有该支持类提供的getSupportFragmentManager（）方法才能获取FragmentManager管理器。

➢ **ViewPager**：它是Fragment容器，可以同时管理多个Fragment，并允许多个Fragment切换时提供动画效果。

➢ **FragmentPagerAdapter**：Adapter类，用于为ViewPager提供多个Fragment。通常用于被扩展。

提示：

FragmentPagerAdapter的作用有点类似于前面介绍的Adapter，只是Adapter用于为AdapterView提供多个列表项；而FragmentPagerAdapter则专门为ViewPager提供多个Fragment。

➢ **PagerTitleStrip**：与ViewPager结合使用，用于在ViewPager上显示"导航条"。

该实例的界面布局文件将会使用ViewPager容器，该容器可以盛装多个Fragment，并为多个Fragment切换时提供动画支持。该实例的布局文件如下。

程序清单：
codes\02\2.11\ActionBar_SwipeNav\res\layout\main.xml

上面的布局文件中定义了一个ViewPager组件，并为该ViewPager组件定义了配套的PagerTitleStrip组件—它是一个导航状态条组件。

接下来Activity中还是需要按上面介绍的两个步骤来启用ActionBar的Tab导航支持。除此之外，为了让ViewPager组件能正常工作，Activity需要为该ViewPager组件创建并设置FragmentPagerAdapter。

下面是该Activity的代码。

程序清单：
codes\02\2.11\ActionBar_SwipeNav\src\org\crazyit\ui\MainActivity.java

上面的程序采用匿名内部类的形式创建了一个FragmentPagerAdapter对象。接下来程序在①号代码处为ViewPager组件设置了该FragmentPagerAdapter对象，这样即可让该ViewPager正常工作。

注意：

上面的界面布局文件和MainActivity都使用了Android v4支持库，该支持库是位于extras\android\support\v4目录下的android-support-v4.jar文件，因此该实例需要将该JAR包复制到应用的app\libs目录下，并在Android Studio左上角的Project面板中选中该JAR，然后通过“Add as Library”右键菜单来添加该JAR包—这也是在Android Studio中为Androd应用添加第三方JAR包的方法。

为了启用ActionBar的Tab导航支持，上面的粗体字代码同样遵守前面介绍的两个步骤：先调用ActionBar的setNavigationMode（ActionBar.NAVIGATION_MODE_TABS）启用Tab导航支持；再调用ActionBar的addTab（）方法添加Tab标签，并为Tab标签绑定事件监听器。

由于此处使用了ViewPager来管理多个Fragment，程序代码处理Fragment的切换时更简单：只要调用ViewPager的setCurrentItem（）方法来显示指定Fragment即可，如以上程序中②号代码所示。

运行上面的程序，可以看到如图2.93所示的界面。

图2.93 A ndroid 3.0版本以前的Fragment支持

当用户单击图2.93所示界面上的Tab标签，或通过拖动来切换ViewPager显示的Fragment时，将可以看到Fragment切换时的动画效果。

2.11.6 使用ActionBar实现下拉式导航

ActionBar除了可以提供Tab导航支持之外，还提供了下拉式（DropDown）导航方式。下拉式导航的ActionBar在顶端生成下拉列表框，当用户单击某个列表项时，系统根据用户单击事件导航到指定Fragment。

为了使用ActionBar实现下拉式导航，按如下步骤进行即可。

1 调用ActionBar的actionBar.setNavigationMode（ActionBar.NAVIGATION_MODE_LIST）方法设置使用下拉列表的导航方式。

2 调用ActionBar的setListNavigationCallbacks（SpinnerAdapter adapter，ActionBar.OnNavigationListener callback）方法添加多个列表项，并为每个列表项设置事件监听器。其中第一个参数Adapter负责提供多个列表项，第二个参数为事件监听器。

实例：ActionBar结合Fragment实现下拉式导航

本实例的布局文件与前面介绍的Tab导航实例的布局文件完全相同：该布局文件只要有一个简单的容器即可，该容器用于盛装多个Fragment。此处不再给出该界面布局文件代码。

下面是该Activity的代码。

程序清单：
codes\02\2.11\ActionBar_DropDownNav\src\org\crazyit\ui\MainActivity java

上面程序中的第一段粗体字代码就是为ActionBar启用下拉导航支持的关键代码，这段代码做了上面介绍的两件事情：先调用ActionBar的setNavigationMode（ActionBar.NAVIGATION_MODE_LIST）启用下拉列表导航支持；然后为ActionBar传入ArrayAdapter（当然也可使用SimpleAdapter或扩展BaseAdapter对象）和监听器即可。

当用户选中指定的导航项时，将会激发该监听器的onNavigationItemSelected（），该方法的处理逻辑与前面Tab导航实例中onTabSelected（）处理方法的处理逻辑完全相同，该实例所使用的DummyFragment与前面的Fragment类的代码也完全相同，此处不再赘述。

图2.94 ActionBar结合Fragment实现下拉式导航

运行该实例，可以看到如图2.94所示的界面。

2.12 本章小结

本章主要介绍了Android应用界面开发的相关知识，对于一个手机应用来说，它面临的最终用户都是不太懂软件的普通人，这批用户第一眼看到的就是软件界面，因此为Android系统提供一个友好的用户界面十分重要。学习本章需要重点掌握View与ViewGroup的功能和用法；Android系统所有的基本UI组件、高级UI组件也都需要重点掌握。除此之外，用户界面少不了需要对话框与菜单，Android为对话框提供了AlertDialog类，还提供了PopupWindow、DatePickerDialog、

TimePickerDialog用于辅助开发对话框。另外，Toast是手机系统特有的一种“准对话框”，它可用于显示简单的提示信息，而且一段时间后会自动隐藏，非常方便。Android为菜单支持提供了SubMenu、ContextMenu、MenuItem等API，读者必须掌握这些API的用法，并能通过它们为Android应用添加菜单支持。

第3章 Android的事件处理

本章要点

事件处理概述与Android事件处理

基于监听的事件处理模型

事件与事件监听器接口

实现事件监听器的方式

基于回调的事件处理模型

基于回调的事件传播

常见的事件回调方法

响应系统设置事件

重写onConfigurationChanged方法响应系统设置更

Handler类的功能与用法

使用Handler更新程序界面

Handler、Looper、MessageQueue工作原理

异步任务的功能与用法

与界面编程紧密相关的知识就是事件处理了，当用户在程序界面上执行各种操作时，应用程序必须为用户动作提供响应动作，这种响应动作就需要通过事件处理来完成。因此本章知识与上一章的内容衔接得

非常紧密，实际上我们在介绍上一章示例时已经使用过Android的事件处理了。

Android提供了两种方式的事件处理：基于回调的事件处理和基于监听的事件处理。熟悉传统图形界面编程的读者对于基于回调的事件处理可能比较熟悉；熟悉AWT/Swing开发方式的读者对于基于监听的事件处理可能比较熟悉。Android系统充分利用了两种事件处理方式的优点，允许开发者采用自己熟悉的事件处理方式来为用户操作提供响应动作。

本章将会详细介绍Android事件处理的各种实现细节，学完本章内容之后，再结合上一章的内容，读者将可以开发出界面友好、人机交互良好的Android应用。

3.1 Android事件处理概述

不管是桌面应用还是手机应用程序，面对最多的就是用户，经常需要处理的就是用户动作—也就是需要为用户动作提供响应，这种为用户动作提供响应的机制就是事件处理。

Android提供了两套强大的事件处理机制：

➢ 基于监听的事件处理。

➢ 基于回调的事件处理。

对于Android基于监听的事件处理而言，主要做法就是为Android界面组件绑定特定的事件监听器，上一章我们已经见到大量这种事件处理的示例。

提示：

Android还允许在界面布局文件中为UI组件的android：onClick属性指定事件监听方法，通过这种方式指定事件监听方法时，开发者需要在

Activity中定义该事件监听方法（该方法必须有一个View类型的形参，该形参代表被单击的UI组件），当用户单击该UI组件时，系统将会激发android：onClick属性所指定的方法。

对于Android基于回调的事件处理而言，主要做法就是重写Android组件特定的回调方法，或者重写Activity的回调方法。Android为绝大部分界面组件都提供了事件响应的回调方法，开发者只要重写它们即可。

一般来说，基于回调的事件处理可用于处理一些具有通用性的事件，基于回调的事件处理代码会显得比较简洁。但对于某些特定的事件，无法使用基于回调的事件处理，只能采用基于监听的事件处理。

3.2 基于监听的事件处理

基于监听的事件处理是一种更"面向对象"的事件处理，这种处理方式与Java的AWT、Swing的处理方式几乎完全相同。如果开发者有AWT、Swing事件处理的编程经验，基本上可以直接上手编程，甚至不需要学习。如果以前没有任何事件处理的编程经验，就需要花点时间先去理解事件监听的处理模型。

3.2.1 监听的处理模型

在事件监听的处理模型中，主要涉及如下三类对象。

➢ **Event Source（事件源）：** 事件发生的场所，通常就是各个组件，例如按钮、窗口、菜单等。

➢ **Event（事件）：** 事件封装了界面组件上发生的特定事情（通常就是一次用户操作）。如果程序需要获得界面组件上所发生事件的相关信息，一般通过Event对象来取得。

➢ **Event Listener（事件监听器）：** 负责监听事件源所发生的事件，并对各种事件做出相应的响应。

提示：

有过JavaScript、Visual Basic等编程经验的读者都知道，事件响应的动作实际上就是一系列程序语句，通常以方法的形式组织起来。但Java是面向对象的编程语言，方法不能独立存在，所以必须以类的形式来组织这些方法，所以事件监听器的核心就是它所包含的方法—这些方法也被称为事件处理器（Event Handler）。

当用户按下一个按钮或者单击某个菜单项时，这些动作就会激发一个相应的事件，该事件就会触发事件源上注册的事件监听器（特殊的Java对象），事件监听器调用对应的事件处理器（事件监听器里的实例方法）来做出相应的响应。

Android的事件处理机制是一种委派式（Delegation）事件处理方式：普通组件（事件源）将整个事件处理委托给特定的对象（事件监听器）；当该事件源发生指定的事件时，就通知所委托的事件监听器，由事件监听器来处理这个事件。

每个组件均可以针对特定的事件指定一个事件监听器，每个事件监听器也可监听一个或多个事件源。因为同一个事件源上可能发生多种事件，委派式事件处理方式可以把事件源上所有可能发生的事件分别授权给不同的事件监听器来处理；同时也可以让一类事件都使用同一个事件监听器来处理。

提示：

委派式事件处理方式明显“抄袭”了人类社会的分工协作，例如某个单位发生了火灾，该单位通常不会自己处理该事件，而是将该事件委派给消防局（事件监听器）处理；如果发生了打架斗殴事件，则委派给公安局（事件监听器）处理；而消防局、公安局也会同时监听多个单位的火灾、打架斗殴事件。这种委派式的处理方式将事件源和事件监听器分离，从而提供更好的程序模型，有利于提高程序的可维护性。

如图3.1所示为事件处理流程示意图。

图3.1 事件处理流程示意图

下面以一个简单的入门程序来示范基于监听的事件处理模型。

先看本程序的界面布局代码，该界面布局中只是定义了两个组件：一个文本框和一个按钮，界面布局代码如下。

程序清单：
codes\03\3.2\EventQs\app\src\main\res\layout\main.xml

上面程序中定义的按钮将会作为事件源，接下来程序将会为该按钮绑定一个事件监听器—监听器类必须由开发者来实现。下面是Java程序代码。

上面程序中的粗体字代码定义了一个View.OnClickListener实现类，这个实现类将会作为事件监听器使用。程序中①号代码用于为bn按钮注册事件监听器。当程序中的bn按钮被单击时，该处理器被触发，将看到程序中文本框内变为“bn按钮被单击了”。

从上面的程序中可以看出，基于监听的事件处理模型的编程步骤如下。

1 获取普通界面组件（事件源），也就是被监听的对象。

2 实现事件监听器类，该监听器类是一个特殊的Java类，必须实现一个XxxListener接口。

3 调用事件源的setXxxListener方法将事件监听器对象注册给普通组件（事件源）。

当事件源上发生指定事件时，Android会触发事件监听器，由事件监听器调用相应的方法（事件处理器）来处理事件。

把上面的程序与图3.1结合起来看，可以发现基于监听的事件处理有如下规则。

➢ **事件源：**就是程序中的bn按钮，其实开发者不需要太多的额外处理，应用程序中任何组件都可作为事件源。

➢ **事件监听器：**就是程序中的MyClickListener类。监听器类必须由程序员负责实现，实现监听器类的关键就是实现处理器方法。

➢ **注册监听器：**只要调用事件源的setXxxListener（XxxListener）方法即可。

对于上面三件事情，事件源可以是任何界面组件，不太需要开发者参与；注册监听器也只要一行代码即可，因此事件编程的关键就是实现事件监听器类。

3.2.2 事件和事件监听器

从图3.1中可以看出，当外部动作在Android组件上执行操作时，系统会自动生成事件对象，这个事件对象会作为参数传给事件源上注册的事件监听器。

基于监听的事件处理模型涉及三个成员：事件源、事件和事件监听器，其中事件源最容易创建，任意界面组件都可作为事件源；事件的产生无须程序员关心，它是由系统自动产生的；所以，实现事件监听器是整个事件处理的核心。

但在上面的程序中，我们并未发现事件的踪迹，这是什么原因呢？这是因为Android对事件监听模型做了进一步简化：如果事件源触发的事件足够简单，事件里封装的信息比较有限，那就无须封装事件对象，将事件对象传入事件监听器。

但对于键盘事件、触摸屏事件等，此时程序需要获取事件发生的详细信息。例如，键盘事件需要获取是哪个键触发的事件；触摸屏事件需要获取事件发生的位置等，对于这种包含更多信息的事件，Android同样会将事件信息封装成XxxEvent对象，并把该对象作为参数传入事件处理器。

实例：控制飞机移动

下面以一个简单的飞机游戏为例来介绍键盘事件的监听。游戏中的飞机会随用户单击键的动作而移动：单击不同的键，飞机向不同的方向移动。

为了实现该程序，先开发一个自定义View，该View负责绘制游戏的飞机。该View类的代码如下。

程序清单：
codes\03\3.2\Plane\app\src\main\java\org\crazyit\event\PlainView.java

上面的PlaneView足够简单，因为这个程序只需要绘制玩家自己控制的飞机，没有增加“敌机”。如果游戏中要增加“敌机”，那么还需要增加数据来控制敌机的坐标，并会在View上绘制敌机。

该游戏几乎不需要界面布局，该游戏直接使用PlaneView作为Activity显示的内容，并为该PlaneView增加键盘事件监听器即可。下面是该程序的Activity代码。

程序清单：
codes\03\3.2\plane\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的粗体字代码就是控制飞机移动的关键代码—由于程序需要根据用户按下的键来确定飞机的移动方向，所以上面的程序先调用了KeyEvent（事件对象）的getKeyCode（）来获取触发事件的键，然后针对不同的键来改变游戏中飞机的坐标。

正如前面提到的，如果事件发生时有比较多的信息需要传给事件监听器，那么就需要将事件信息封装成Event对象，该Event对象将作为参数传入事件处理函数。

图3.2 控制飞机的移动

运行上面的程序，将看到如图3.2所示的界面。

对于图3.2所示的“游戏”，当用户按下模拟器右边的4个方向键时，将可以看到“游戏”中的飞机可以上、下、左、右自由移动。

在基于监听的事件处理模型中，事件监听器必须实现事件监听器接口，Android为不同的界面组件提供了不同的监听器接口，这些接口通常以内部类的形式存在。以View类为例，它包含了如下几个内部接口。

➢ **View.OnClickListener**：单击事件的事件监听器必须实现的接口。

➢ **View.OnCreateContextMenuListener**：创建上下文菜单事件的事件监听器必须实现的接口。

➢ **View.onFocusChangeListener**：焦点改变事件的事件监听器必须实现的接口。

➢ **View.OnKeyListener**：按键事件的事件监听器必须实现的接口。

提示：

上面这个“游戏”还只是一个“雏型”，为了增加这个游戏的可玩性，可以考虑为游戏随机地增加“敌机”，并让“敌机”在屏幕上移动。如果想增加对战效果，还可以考虑增加一个键盘监听器：监听特定按键，特定按键激发飞机射出子弹。

➢ **View.OnLongClickListener**：长按事件的事件监听器必须实现的接口。

➢ **View.OnTouchListener**：触摸事件的事件监听器必须实现的接口。

提示：

实际上可以把事件处理模型简化成如下理解。当事件源组件上发生事件时，系统将会执行该事件源组件上监听器的对应处理方法。与普通Java方法调用不同的是，普通Java程序里的方法是由程序主动调用的，事件处理中的事件处理器方法是由系统负责调用的。

通过上面的介绍不难看出，所谓事件监听器，其实就是实现了特定接口的Java类的实例。在程序中实现事件监听器，通常有如下几种形式。

➢ **内部类形式**：将事件监听器类定义成当前类的内部类。

➢ **外部类形式**：将事件监听器类定义成一个外部类。

➢ **Activity 本身作为事件监听器类**：让Activity 本身实现监听器接口，并实现事件处理方法。

➢ **匿名内部类形式**：使用匿名内部类创建事件监听器对象。

3.2.3 内部类作为事件监听器类

前面两个程序中所使用的事件监听器类都是内部类形式，使用内部类可以在当前类中复用该监听器类；因为监听器类是外部类的内部类，所以可以自由访问外部类的所有界面组件。这也是内部类的两个优势。

使用内部类来定义事件监听器类的例子可以参看前面的例子程序，此处不再赘述。

3.2.4 外部类作为事件监听器类

使用外部类定义事件监听器类的形式比较少见，主要因为如下两个原因。

➢ 事件监听器通常属于特定的GUI界面，定义成外部类不利于提高程序的内聚性。

➢ 外部类形式的事件监听器不能自由访问创建GUI界面的类中的组件，编程不够简洁。

但如果某个事件监听器确实需要被多个GUI界面所共享，而且主要是完成某种业务逻辑的实现，则可以考虑使用外部类形式来定义事件监听器类。下面的程序定义了一个外部类作为OnLongClickListener类，该事件监听器实现了发送短信的功能。

程序清单：
codes\03\3.2\SendSms\app\src\main\java\org\crazyit\event\SendSmsListener.java

上面的事件监听器类没有与任何GUI界面耦合，创建该监听器对象时需要传入两个EditText对象和一个Activity对象，其中一个EditText用于作为收信人号码，另一个EditText用于作为短信内容。

提示：

上面程序中的三行粗体字代码调用了SmsManager、PendingIntent来发送短信，关于SmsManager、Intent的用法可参看本书后面的内容。

该程序的界面布局比较简单，故不给出界面布局文件。该程序的Java代码如下。

程序清单：
codes\03\3.2\SendSms\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的粗体字代码用于为指定按钮的长单击事件绑定监听器，当用户长单击界面中的bn按钮时，程序将会触发SendSmsListener监听器，该监听器里包含的事件处理方法将会向指定手机发送短信。

运行上面的程序，将看到如图3.3所示的界面。

此处使用模拟器来发送短信—不要指望能给你的女朋友发送短信，否则中国移动要恼火了。该程序只能向另一台模拟器发送短信，比如执行如下命令来启动另一台模拟器（直接使用名为fkjava的AVD虚拟机）：

图3.3 向指定手机发送短信

启动完成后可以在模拟器窗口的标题上看到该模拟器的号码，如图3.4所示。

启动了如图3.4所示的模拟器之后，就可以通过图3.3所示的程序向指定模拟器发送短信了。

图3.4 模拟器的号码

实际上不推荐将业务逻辑实现写在事件监听器中，包含业务逻辑的事件监听器将导致程序的显示逻辑和业务逻辑耦合，从而增加程序后期的维护难度。如果确实有多个事件监听器需要实现相同的业务逻辑功能，则可以考虑使用业务逻辑组件来定义业务逻辑功能，再让事件监听器来调用业务逻辑组件的业务逻辑方法。

3.2.5 Activity本身作为事件监听器类

这种形式使用Activity本身作为监听器类，可以直接在Activity类中定义事件处理器方法。这种形式非常简洁，但这种做法有两个缺点。

➢ 这种形式可能造成程序结构混乱，Activity的主要职责应该是完成界面初始化工作，但此时还需包含事件处理器方法，从而引起混乱。

➢ 如果Activity界面类需要实现监听器接口，让人感觉比较怪异。

下面的程序使用Activity对象作为事件监听器。

程序清单：
codes\03\3.2\ActivityListener\app\src\main\java\org\crazyit\event\MainActivity.java

上面的程序让Activity类实现了OnClickListener事件监听器接口，从而可以在该Activity类中直接定义事件处理器方法：onClick（View v）（如上面的粗体字代码所示）。当为某个组件添加该事件监听器对象时，直接使用this作为事件监听器对象即可。

3.2.6 匿名内部类作为事件监听器类

大部分时候，事件处理器都没有什么复用价值（可复用代码通常都被抽象成了业务逻辑方法），因此大部分事件监听器只是临时使用一次，所以使用匿名内部类形式的事件监听器更合适。实际上，这种形式是目前使用最广泛的事件监听器形式。下面的程序使用匿名内部类来创建事件监听器。

程序清单：
codes\03\3.2\AnonymousListener\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的粗体字部分使用匿名内部类创建了一个事件监听器对象，“new 监听器接口”或“new 事件适配器”的形式就是用于创建匿名内部类形式的事件监听器。

对于使用匿名内部类作为事件监听器类的形式来说，唯一的缺点就是匿名内部类的语法有点不易掌握，如果读者的Java基础扎实，匿名内部类的语法掌握较好，通常建议使用匿名内部类作为监听器类。

3.2.7 直接绑定到标签

Android还有一种更简单的绑定事件监听器的方式，那就是直接在界面布局文件中为指定标签绑定事件处理方法。

对于很多Android界面组件标签而言，它们都支持onClick属性，该属性的属性值就是一个形如xxx（View source）方法的方法名。例如如下界面布局文件。

程序清单：
codes\03\3.2\BindingTag\app\src\main\res\layout\main.xml

上面程序中的粗体字代码用于在界面布局文件中为Button按钮绑定一个事件处理方法：clickHanlder，这就意味着开发者需要在该界面布局对应的Activity中定义一个void clickHandler（View source）方法，该方法将会负责处理该按钮上的单击事件。下面是该界面布局对应的Java代码。

程序清单：
codes\03\3.2\BindingTag\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的粗体字代码定义了一个clickHandler（View source）方法，当程序中的bn按钮被单击时，该方法将会被激发并处理bn按钮上的单击事件。

3.3 基于回调的事件处理

除了前面介绍的基于监听的事件处理模型之外，Android还提供了一种基于回调的事件处理模型。从代码实现的角度来看，基于回调的事件处理模型更加简单。

3.3.1 回调机制与监听机制

如果说事件监听机制是一种委托式的事件处理，那么回调机制则恰好与之相反：对于基于回调的事件处理模型来说，事件源与事件监听器是统一的，或者说事件监听器完全消失了。当用户在GUI组件上激发某个事件时，组件自己特定的方法将会负责处理该事件。

为了使用回调机制类处理GUI组件上所发生的事件，我们需要为该组件提供对应的事件处理方法—而Java又是一种静态语言，我们无法为某个对象动态地添加方法，因此只能继承GUI组件类，并重写该类的事件处理方法来实现。

为了实现回调机制的事件处理，Android为所有GUI组件都提供了一些事件处理的回调方法，以View为例，该类包含如下方法。

➢ **boolean onKeyDown（int keyCode，KeyEvent event）**：当用户在该组件上按下某个按键时触发该方法。

➢ **boolean onKeyLongPress（int keyCode，KeyEvent event）**：当用户在该组件上长按某个按键时触发该方法。

➢ **boolean onKeyShortcut（int keyCode，KeyEvent event）**：当一个键盘快捷键事件发生时触发该方法。

➢ **boolean onKeyUp（int keyCode，KeyEvent event）**：当用户在该组件上松开某个按键时触发该方法。

➢ **boolean onTouchEvent（MotionEvent event）**：当用户在该组件上触发触摸屏事件时触发该方法。

➢ **boolean onTrackballEvent（MotionEvent event）**：当用户在该组件上触发轨迹球事件时触发该方法。

下面的程序示范了基于回调的事件处理机制。正如前面所提到的，基于回调的事件处理机制可通过自定义View来实现，自定义View时重写该View的事件处理方法即可。下面是一个自定义按钮的实现类。

程序清单：
codes\03\3.3\CallbackHandler\app\src\main\java\org\crazyit\event\MyButton.java

在上面自定义的MyButton类中，我们重写了Button类的onKeyDown（int keyCode，KeyEvent event）方法，该方法将会负责处理按钮上的键盘事件。

接下来在界面布局文件中使用这个自定义View，界面布局文件如下。

程序清单：
codes\03\3.3\CallbackHandler\app\src\main\res\layout\main.xml

上面程序中的粗体字代码在XML界面布局文件中使用MyButton组件，接下来Java程序无须为该按钮绑定事件监听器—因为该按钮自己重写了onKeyDown（int keyCode，KeyEvent event）方法，这意味着该按钮将会自己处理相应的事件。

运行上面的程序，先把焦点定位到该按钮上，如图3.5所示。

接着单击模拟器上的任意按键，将可以看到Android Studio的LogCat中有如图3.6所示的输出。

图3.5 按钮获得焦点

图3.6 基于回调的事件处理

通过上面的介绍不难发现，对于基于监听的事件处理模型来说，事件源和事件监听器是分离的，当事件源上发生特定事件时，该事件交给事件监听器负责处理；对于基于回调的事件处理模型来说，事件源和事件监听器是统一的，当事件源发生特定事件时，该事件还是由事件源本身负责处理。

3.3.2 基于回调的事件传播

几乎所有基于回调的事件处理方法都有一个boolean类型的返回值，该返回值用于标识该处理方法是否能完全处理该事件。

➢ 如果处理事件的回调方法返回true，表明该处理方法已完全处理该事件，该事件不会传播出去。

➢ 如果处理事件的回调方法返回 false，表明该处理方法并未完全处理该事件，该事件会传播出去。

对于基于回调的事件传播而言，某组件上所发生的事件不仅会激发该组件上的回调方法，也会触发该组件所在Activity的回调方法—只要事件能传播到该Activity。

下面的程序示范了Android系统中的事件传播，该程序重写了Button类的onKeyDown（int keyCode，KeyEvent event）方法，而且重写了该Button所在Activity的onKeyDown（int keyCode，KeyEvent event）方法—程序没有阻止事件传播，因此程序可以看到事件从Button传播到Activity的情形。

下面是从Button派生而出的MyButton子类代码。

程序清单：
codes\03\3.3\Propagation\app\src\main\java\org\crazyit\event\MyButton.java

上面的MyButton子类重写了onKeyDown（int keyCode，KeyEvent event）方法，当用户在该按钮上按下某个键时将会触发该方法。但由于该方法返回了false，这意味着该事件还会继续向外传播。

该程序也按前一个示例的方式使用自定义组件，并在Activity中重写public boolean onKeyDown（int keyCode，KeyEvent event）方法，该方法也会在某个按键被按下时被回调。

看如下Activity类代码。

程序清单：
codes\03\3.3\Propagation\app\src\main\java\org\crazyit\event\MainActivity.java

从上面的程序可以看出，粗体字代码重写了Activity的onKeyDown（int keyCode，KeyEvent event）方法，当在该Activity包含的所有组件上按下某个键时，该方法都可能被触发—只要该组件没有完全处理该事件。

不仅如此，上面的程序还采用监听模式来处理该按钮上的按键被按下的事件。

运行上面的程序，先把焦点移动到程序界面的按钮上，然后按下模拟器右边的按键，将可以在Android Studio的LogCat中看到如图3.7所示的输出。

从图3.7不难看出，当该组件上发生某个按键被按下的事件时，Android系统最先触发的应该是该按键上绑定的事件监听器，然后才触发该组件提供的事件回调方法，最后还会传播到该组件所在的Activity—但如果我们让任何一个事件处理方法返回了true，那么该事件将不会继续向外传播。例如，我们改写上面的Activity代码，将程序中①号代码改为return true，然后再运行该程序，把焦点定位到该按钮上之后单击某个按键，在Android Studio的LogCat中将看到如图3.8所示的输出。

图3.7 基于回调的事件传播

图3.8 监听器阻止事件传播

3.3.3 重写onTouchEvent方法响应触摸屏事件

对比Android提供的两种事件处理模型，不难发现基于监听的事件处理模型具有更大的优势。

➢ 基于监听的事件处理模型分工更明确，事件源、事件监听器由两个类分开实现，因此具有更好的可维护性。

➢ Android的事件处理机制保证基于监听的事件监听器会被优先触发。

实例：通过回调实现跟随手指的小球

在某些特定情况下，基于回调的事件处理机制会更好地提高程序的内聚性，例如上一章的"实例：跟随手指的小球"，如果改为基于回调的实现，则可以更好地提高程序的内聚性。

将该实例中的DrawView类改为如下形式。

程序清单：
codes\03\3.3\CustomView\app\src\main\java\org\crazyit\event\DrawView.java

上面程序中的粗体字代码重写了View组件的onTouchEvent（MotionEvent event）方法，这表示该组件自己就可以处理触摸屏事件，当用户手指在屏幕上移动时，该View上绘制的小球将会跟随用户手指。也就是说，这个自定义View本身就可以很好地处理屏幕事件。

接下来在程序中使用这个DrawView几乎不需要增加任何处理，直接将View放到界面布局中即可。如下面的XML界面布局文件所示。

程序清单：
codes\03\3.3\CustomView\app\src\main\res\layout\main.

xml

接下来在Activity类中不需要为该View绑定事件监听器—因为该View自己就可以处理它的触摸屏事件。

通过为View提供事件处理的回调方法，可以很好地把事件处理方法封装在该View内部，从而提高程序的内聚性—基于回调的事件处理更适合于应付那种事件处理逻辑比较固定的View，比如上面介绍的这个跟随用户手指的View。

3.4 响应系统设置的事件

在开发Android应用时，有时候可能需要让应用程序随系统设置而进行调整，比如判断系统的屏幕方向、判断系统方向的方向导航设备等。除此之外，有时候可能还需要让应用程序监听系统设置的更改，对系统设置的更改做出响应。

3.4.1 Configuration类简介

Configuration类专门用于描述手机设备上的配置信息，这些配置信息既包括用户特定的配置项，也包括系统的动态设备配置。

程序可调用Activity的如下方法来获取系统的Configuration对象：

一旦获得了系统的Configuration对象，就可以使用该对象提供的如下常用属性来获取系统的配置信息。

➢ **public float fontScale**：获取当前用户设置的字体的缩放因子。

➢ **public int keyboard**：获取当前设备所关联的键盘类型。该属性可能返回KEYBOARD_NOKEYS、KEYBOARD_QWERTY（普通电脑键

盘）、KEYBOARD_12KEY（只有12个键的小键盘）等属性值。

➢ **public int keyboardHidden**：该属性返回一个boolean值用于标识当前键盘是否可用。该属性不仅会判断系统的硬件键盘，也会判断系统的软键盘（位于屏幕上）。如果该系统的硬件键盘不可用，但软键盘可用，该属性也会返回KEYBOARDHIDDEN_NO；只有当两个键盘都不可用时才返回KEYBOARDHIDDEN_YES。

➢ **public Locale locale**：获取用户当前的Locale。

➢ **public int mcc**：获取移动信号的国家码。

➢ **public int mnc**：获取移动信号的网络码。

➢ **public int navigation**：判断系统上方向导航设备的类型。该属性可能返回NAVIGATION_NONAV （无导航）、NAVIGATION_DPAD （DPAD 导航）、NAVIGATION_TRACKBALL（轨迹球导航）、NAVIGATION_WHEEL（滚轮导航）等属性值。

➢ **public int orientation**：获取系统屏幕的方向，该属性可能返回ORIENTATION_LANDSCAPE （横向屏幕）、ORIENTATION_PORTRAIT （竖向屏幕）、ORIENTATION_SQUARE （方形屏幕）等属性值。

➢ **public int touchscreen**：获取系统触摸屏的触摸方式。该属性可能返回 TOUCHSC-REEN_NOTOUCH（无触摸屏）、TOUCHSCREEN_STYLUS（触摸笔式的触摸屏）、TOUCHSCREEN_FINGER（接受手指的触摸屏）等属性值。

下面以一个实例来介绍Configuration的用法，该程序可以获取系统的屏幕方向、触摸屏方式等。

实例：获取系统设备状态

该程序的界面布局比较简单，程序只是提供了4个文本框来显示系统的屏幕方向、触摸屏方式等状态，故此处不再给出界面布局文件。

该程序的Java代码主要可分为两步。

1 获取系统的Configuration对象。

2 调用Configuration对象的属性来获取设备状态。

下面是该程序的Java代码。

程序清单：
codes\03\3.4\ConfigurationTest\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的粗体字代码用于获取系统的 Configuration 对象，一旦获得了系统的Configuration对象之后，程序就可以通过它来了解系统的设备状态了。运行上面的程序，将显示如图3.9所示的界面。

单击图3.9所示界面中的“获取手机信息”按钮，系统显示如图3.10所示的界面。

图3.9 未获取设备状态

图3.10 获取设备状态

3.4.2 重写onConfigurationChanged方法响应系统设置更改

如果程序需要监听系统设置的更改，则可以考虑重写Activity的onConfigurationChanged （Configuration newConfig）方法，该方法是一个基于回调的事件处理方法：当系统设置发生更改时，该方法会被自动触发。

为了在程序中动态地更改系统设置，我们可调用Activity的setRequestedOrientation（int）方法来修改屏幕的方向。

实例：监听屏幕方向的改变

该实例的界面布局很简单，该界面中仅包含一个普通按钮，该按钮用于动态修改系统屏幕的方向，此处不再给出系统界面布局代码。

该程序的Java代码主要会调用Activity的setRequestedOrientation（int）方法来动态更改屏幕方向。除此之外，我们还重写了Activity的onConfigurationChanged（Configuration newConfig）方法，该方法可用于监听系统设置的更改。程序代码如下。

程序清单：
codes\03\3.4\ChangeCfg\app\src\main\java\org\crazyit\event\MainActivity.java

上面程序中的前两行粗体字代码用于动态地修改手机屏幕的方向，接下来的粗体字代码重写了Activity的onConfigurationChanged（Configuration newConfig）方法，当系统设置发生更改时，该方法将会被自动回调。

除此之外，为了让该Activity能监听屏幕方向更改的事件，需要在配置该Activity时指定android：configChanges属性，该属性可以支持mcc、mnc、locale、touchscreen、keyboard、keyboardHidden、navigation、orientation、screenLayout、uiMode、screenSize、

smallestScreenSize、fontScale属性值，其中orientation|screenSize属性值指定该Activity可以监听屏幕方向改变的事件。

因此，将应用的AndroidManifest.xml文件改为如下形式。

程序清单：
codes\03\3.4\ChangeCfg\app\main\src\AndroidManifest.xml

上面的粗体字配置代码指定了该Activity可以监听屏幕方向改变的事件，这样当程序改变手机屏幕方向时，Activity的onConfigurationChanged（）方法就会被回调。

提供上面的程序和设置之后，运行该程序，单击应用程序中的"更改屏幕方向"按钮，将可以看到如图3.11所示的界面。

图3.11 设置横向屏幕并响应系统设置的更改

3.5 Handler消息传递机制

出于性能优化考虑，Android的UI操作并不是线程安全的，这意味着如果有多个线程并发操作UI组件，则可能导致线程安全问题。为了解决这个问题，Android制定了一条简单的规则：只允许UI线程修改Activity里的UI组件。

当一个程序第一次启动时，Android会同时启动一条主线程（Main Thread），主线程主要负责处理与UI相关的事件，如用户的按键事件、用户接触屏幕的事件及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以，主线程通常又被叫作UI线程。

Android的消息传递机制是另一种形式的“事件处理”，这种机制主要是为了解决Android应用的多线程问题—Android平台只允许UI线程修改Activity里的UI组件，这样就会导致新启动的线程无法动态改变界面组件的属性值。但在实际Android应用开发中，尤其是涉及动画的游戏开发中，需要让新启动的线程周期性地改变界面组件的属性值，这就需要借助于Handler的消息传递机制来实现了。

3.5.1 Handler类简介

Handler类的主要作用有两个。

➢ 在新启动的线程中发送消息。

➢ 在主线程中获取、处理消息。

上面的说法看上去很简单，似乎只要分成两步即可：在新启动的线程中发送消息；然后在主线程中获取并处理消息。但这个过程涉及两个问题：新启动的线程何时发送消息呢？主线程何时去获取并处理消息呢？这个时机显然不好控制。

为了让主线程能“适时”地处理新启动的线程所发送的消息，显然只能通过回调的方式来实现—开发者只要重写Handler类中处理消息的方法，当新启动的线程发送消息时，消息会发送到与之关联的MessageQueue，而Handler会不断地从MessageQueue中获取并处理消息—这将导致Handler类中处理消息的方法被回调。

Handler类包含如下方法用于发送、处理消息。

➢ **void handleMessage（Message msg）**：处理消息的方法。该方法通常用于被重写。

➢ **final boolean hasMessages（int what）**：检查消息队列中是否包含what属性为指定值的消息。

➢ **final boolean hasMessages（int what，Object object）：**检查消息队列中是否包含what属性为指定值且object属性为指定对象的消息。

➢ **多个重载的Message obtainMessage（）：**获取消息。

➢ **sendEmptyMessage（int what）：**发送空消息。

➢ **final boolean sendEmptyMessageDelayed（int what，long delayMillis）：**指定多少毫秒之后发送空消息。

➢ **final boolean sendMessage（Message msg）：**立即发送消息。

➢ **final boolean sendMessageDelayed（Message msg，long delayMillis）：**指定多少毫秒之后发送消息。

借助于上面这些方法，程序可以方便地利用Handler来进行消息传递。

实例：自动播放动画

本实例通过一个新线程来周期性地修改ImageView所显示的图片，通过这种方式来开发一个动画效果。该程序的界面布局代码非常简单，程序只是在界面布局中定义了ImageView组件，此处不再给出界面布局代码。

接下来主程序使用java.util.Timer来周期性地执行指定任务，程序代码如下。

程序清单：
codes\03\3.5\HandlerTest\app\src\main\java\org\crazyit\handler\MainActivity.java

上面程序中的第二段粗体字代码通过 Timer 周期性地执行指定任务，Timer 对象可调度TimerTask对象，TimerTask对象的本质就是启动一条新线程，由于Android不允许在新线程中访问Activity里的界面组件，因此程序只能在新线程里发送一条消息，通知系统更新ImageView组件。

上面程序中的第一段粗体字代码重写了Handler的handleMessage（Message msg）方法，该方法用于处理消息—当新线程发送消息时，该方法会被自动回调，handleMessage（Message msg）方法依然位于主线程中，所以可以动态地修改ImageView组件的属性。这就实现了本程序所要达到的效果：由新线程来周期性地修改ImageView的属性，从而实现动画效果。运行上面的程序，可以看到应用程序中5张图片交替显示的动态效果。

3.5.2 Handler、Loop、MessageQueue的工作原理

为了更好地理解Handler的工作原理，下面先介绍一下与Handler一起工作的几个组件。

➢ Message：Handler接收和处理的消息对象。

➢ Looper：每个线程只能拥有一个Looper。它的loop方法负责读取MessageQueue中的消息，读到信息之后就把消息交给发送该消息的Handler进行处理。

➢ MessageQueue：消息队列，它采用先进先出的方式来管理Message。程序创建Looper对象时，会在它的构造器中创建MessageQueue对象。Looper的构造器源代码如下：

该构造器使用了private修饰，表明程序员无法通过构造器创建Looper对象。从上面的代码不难看出，程序在初始化Looper时会创建一个与之关联的MessageQueue，这个MessageQueue就负责管理消息。

➢ **Handler：**它的作用有两个—发送消息和处理消息，程序使用Handler 发送消息，由Handler发送的消息必须被送到指定的MessageQueue。也就是说，如果希望Handler正常工作，必须在当前线程中有一个MessageQueue；否则消息就没有MessageQueue进行保存了。不过MessageQueue是由Looper负责管理的，也就是说，如果希望Handler正常工作，必须在当前线程中有一个Looper对象。为了保证当前线程中有Looper对象，可以分如下两种情况处理。

➢ 在主UI线程中，系统已经初始化了一个Looper对象，因此程序直接创建Handler即可，然后就可通过Handler来发送消息、处理消息了。

➢ 程序员自己启动的子线程，必须自己创建一个Looper对象，并启动它。创建Looper对象调用它的prepare（）方法即可。

prepare（）方法保证每个线程最多只有一个Looper对象。prepare（）方法的源代码如下：

接下来调用Looper的静态loop（）方法来启动它。loop（）方法使用一个死循环不断取出MessageQueue中的消息，并将取出的消息分给该消息对应的Handler进行处理。下面是Looper类的loop（）方法的源代码。

归纳起来，Looper、MessageQueue、Handler各自的作用如下。

➢ **Looper：**每个线程只有一个 Looper，它负责管理MessageQueue，会不断地从MessageQueue中取出消息，并将消息分给对应的Handler处理。

➢ **MessageQueue：**由Looper负责管理。它采用先进先出的方式来管理Message。

➢ **Handler**：它能把消息发送给Looper管理的MessageQueue，并负责处理Looper分给它的消息。

在线程中使用Handler的步骤如下。

1 调用Looper的prepare（）方法为当前线程创建Looper对象，创建Looper对象时，它的构造器会创建与之配套的MessageQueue。

2 有了Looper之后，创建Handler子类的实例，重写handleMessage（）方法，该方法负责处理来自于其他线程的消息。

3 调用Looper的loop（）方法启动Looper。

下面通过一个实例来介绍Looper与Handler的用法。

实例：使用新线程计算质数

该实例允许用户输入一个数值上限，当用户单击“计算”按钮时，该应用会将该上限数值发送到新启动的线程中，让该线程来计算该范围内的所有质数。

之所以不直接在UI线程中计算该范围内的所有质数，是因为UI线程需要响应用户动作，如果在UI线程中执行一个“耗时”操作，将会导致UI线程被阻塞，从而让应用程序失去响应。比如在该实例中，如果用户输入的数值太大，系统可能需要较长时间才能计算出所有质数，这就可能导致UI线程失去响应。

提示：

尽量避免在UI线程中执行耗时操作，因为这样可能导致一个“著名”的异常：ANR异常。只要在UI线程中执行需要消耗大量时间的操作，都会引发ANR，因为这会导致Android应用程序无法响应输入事件和Broadcast。

为了将用户在UI界面输入的数值上限动态地传给新启动的线程，本实例将会在线程中创建一个Handler对象，然后UI线程的事件处理方法就可以通过该Handler向新线程发送消息了。

该实例的界面布局文件比较简单，只有一个文本框和一个按钮，故此处不再给出界面布局文件。

该实例的Activity代码如下。

程序清单：
codes\03\3.5\CalPrime\app\src\main\java\org\crazyit\handler\MainActivity.java

上面的粗体字代码是实例的关键代码，这些粗体字代码在新线程内创建了一个Handler。由于在新线程中创建Handler时必须先创建Looper，因此程序先调用Looper的prepare（）方法为当前线程创建了一个Looper实例，并创建了配套的MessageQueue。新线程有了Looper对象之后，接下来程序创建了一个Handler对象，该Handler可以处理其他线程发送过来的消息。程序最后还调用了Looper的loop（）方法。

运行该程序，无论用户输入多大的数值，计算该范围内的质数都将会交给新线程完成，而前台UI线程不会受到影响。该程序的运行效果如图3.12所示。

图3.12 使用新线程计算质数

3.6 异步任务（AsyncTask）

前面已经介绍过，Android的UI线程主要负责处理用户的按键事件、用户触屏事件及屏幕绘图事件等，因此开发者的其他操作不应该、也不能阻塞UI线程；否则UI界面将会变得停止响应—用户感觉非常糟糕。

提示：

Android默认约定当UI线程阻塞超过20秒时将会引发ANR（Application Not Responding）异常。但实际上，不要说20秒，即使是5秒甚至2秒，用户都会感觉十分不爽（用户是很急燥的！）。因此笔者认为，其实没必要去记这个20秒的时间限度。总之，开发者需要牢记：不要在UI线程中执行一些耗时的操作。

为了避免 UI 线程失去响应的问题，Android 建议将耗时操作放在新线程中完成，但新线程也可能需要动态更新UI组件，比如需要从网上获取一个网页，然后在TextView中将其源代码显示出来，此时就应该将连接网络、获取网络数据的操作放在新线程中完成。问题是：获取网络数据之后，新线程不允许直接更新UI组件。

为了解决新线程不能更新UI组件的问题，Android提供了如下几种解决方案。

- **使用HanIder实现线程之间的通信。**
- **Activity.runOnUiThread(Runnable).**
- **View.post(Runnable).**
- **View.postDelayed(Runnable,long).**

上一节已经见到了使用Handler的实例，后面的三种方式可能导致编程略显烦琐，而异步任务（AsyncTask）则可进一步简化这种操作。相对来说AsyncTask更轻量级一些，适用于简单的异步处理，不需要借助线程和Handler即可实现。

AsyncTask<Params，Progress，Result>是一个抽象类，通常用于被继承，继承AsyncTask时需要指定如下三个泛型参数。

➢ **Params**：启动任务执行的输入参数的类型。

➢ **Progress**：后台任务完成的进度值的类型。

➢ **Result**：后台执行任务完成后返回结果的类型。

使用AsyncTask只要如下三步即可。

1 创建AsyncTask的子类，并为三个泛型参数指定类型。如果某个泛型参数不需要指定类型，则可将它指定为Void。

2 根据需要，实现AsyncTask的如下方法。

➢ **doInBackground（Params...）**：重写该方法就是后台线程将要完成的任务。该方法可以调用 publishProgress（Progress...values）方法更新任务的执行进度。

➢ **onProgressUpdate（Progress...values）**： 在doInBackground（） 方 法 中 调 用publishProgress（）方法更新任务的执行进度后，将会触发该方法。

➢ **onPreExecute（）**：该方法将在执行后台耗时操作前被调用。通常该方法用于完成一些初始化的准备工作，比如在界面上显示进度条等。

➢ **onPostExecute（Result result）**：当 doInBackground（）完成后，系统会自动调用onPostExecute（）方法，并将doInBackground（）方法的返回值传给该方法。

3 调用AsyncTask子类的实例的execute（Params...params）开始执行耗时任务。

使用AsyncTask时必须遵守如下规则。

➢ 必须在UI线程中创建AsyncTask的实例。

➢ 必须在UI线程中调用AsyncTask的execute（）方法。

➢ AsyncTask的onPreExecute（）、onPostExecute（Result result）、doInBackground （Params...params）、onProgressUpdate（Progress...v alues）方法，不应该由程序员代码调用，而是由Android系统负责调用。

➢ 每个AsyncTask只能被执行一次，多次调用将会引发异常。

实例：使用异步任务执行下载

本实例示范如何使用异步任务下载网络资源。该实例的界面布局很简单，只包含两个组件：一个文本框用于显示从网络下载的页面代码；一个按钮用于激发下载任务。此处不再给出界面布局文件。

该程序的Activity代码如下。

程序清单：
coder\03\3.6\AsyncTaskTest\app\src\main\java\org\crazyit\handler\MainActivity.java

上面程序的download（）方法很简单，它只是创建了DownTask（AsyncTask的子类）实例，并调用它的execute（）方法开始执行异步任务。

该程序的重点是实现AsyncTask的子类，实现该子类时实现了如下4个方法。

➢ **doInBackground（）：** 该方法的代码完成实际的下载任务。

➢ **onPreExecute（）：** 该方法的代码负责在下载开始的时候显示一个进度条。

➢ **onProgressUpdate（）：** 该方法的代码负责随着下载进度的改变更新进度条的进度值。

➢ **onPostExecute（）：** 该方法的代码负责当下载完成后，将下载的代码显示出来。

提示：

该程序使用了网络编程从网络下载数据。关于Android网络编程的知识，请参考本书第13章的内容。除此之外，本程序需要访问网络，因此还需要在AndroidManifest.xml文件中声明如下权限：<uses-permission android：name="android.permission.INTERNET"/>。

运行该程序并单击“下载”按钮，将可以看到如图3.13所示的界面。

图3.13 使用异步任务下载网络资源

3.7 本章小结

本章是对上一章内容的补充：图形界面编程肯定需要与事件处理结合，当我们开发了一个界面友好的应用之后，用户在程序界面上执行操作时，程序必须为这种用户操作提供响应动作，这种响应动作就是由事件处理来完成的。学习本章的重点是掌握Android的两种事件处理机制：基于回调的事件处理和基于监听的事件处理；对于基于监听的事件处理来说，开发者需要掌握事件监听的处理模式，以及不同事件对应的监听器接口；对于基于回调的事件处理来说，开发者需要掌握不同事件对应的回调方法。除此之外，本章还介绍了重写

onConfigurationChanged方法响应系统设置更改。需要指出的是，由于Android不允许在子线程中更新界面组件，如果想在子线程中更新界面组件，开发者需要借助于Handler对象来实现。本章详细介绍了Handler、Looper与MessageQueue之间的关系及工作原理。

第4章 深入理解Activity与Fragment

本章要点

理解Activity的功能与作用

开发普通Activity类

在AndroidManifest.xml中配置Activity

特殊Activity的功能和用法

在程序中启动Activity

关闭Activity

使用Bundle在不同Activity之间交换数据

启动其他Activity并返回结果

Activity的回调机制

Activity的生命周期

Fragment概述及Fragment的设计哲学

开发Fragment

使用ListFragment

管理Fragment，并让Fragment与Activity通信

Fragment的生命周期

Activity是Android应用的重要组成单元之一（另外三个是Service、BroadcastReceiver和ContentProvider），而Activity又是Android应用最常见的组件之一。前面看到的示例通常都只包含一个Activity，但在实际应用中这是不大可能的，往往包括多个Activity，不同的Activity向用户呈现不同的操作界面。Android应用的多个Activity组成Activity栈，当前活动的Activity位于栈顶。

有Web开发经验的读者对Servlet的概念应该比较熟悉。实际上Activity对于Android应用的作用有点类似于Servlet对于Web应用的作用——一个Web应用通常都需要N个Servlet组成（JSP的本质依然是Servlet）；那么一个Android应用通常也需要N个Activity组成。对于Web应用而言，Servlet（把JSP也统一成Servlet）主要负责与用户交互，并向用户呈现应用状态；对于Android应用而言，Activity大致也具有相同的功能。

当Activity处于Android应用中运行时，同样受系统控制，有其自身的生命周期，本章深入介绍Activity的相关知识。

4.1 建立、配置和使用Activity

Activity是Android应用中最重要、最常见的应用组件（此处的组件是粗粒度的系统组成部分，并非指界面控件：widget）。Android应用开发的一个重要组成部分就是开发Activity，下面将会详细介绍Activity开发、配置的相关知识。

4.1.1 Activity

与开发Web应用时建立Servlet类相似，建立自己的Activity也需要继承Activity基类。当然，在不同应用场景下，有时也要求继承Activity的子类。例如，如果应用程序界面只包括列表，则可以让应用程序继承ListActivity；如果应用程序界面需要实现标签页效果，则可以让应用程序继承TabActivity。

图4.1显示了Android提供的Activity类。

图4.1 各种Activity基类

如图4.1所示，Activity类间接或直接地继承了Context、ContextWrapper、ContextThemeWrapper等基类，因此Activity可以直接调用它们的方法。

与Servlet类似，当一个Activity类定义出来之后，这个Activity类何时被实例化、它所包含的方法何时被调用，这些都不是由开发者决定的，都应该由Android系统来决定。

为了让Servlet能响应用户请求，开发者需要重写HttpServlet的doRequest（..）、doResponse（..）方法，或重写service（..）方法。Activity与此类似，创建一个Activity也需要实现一个或多个方法，其中最常见的就是实现onCreate（Bundle status）方法，该方法将会在Activity创建时被回调，该方法调用Activity的setContentView（View view）方法来显示要展示的View。为了管理应用程序界面中的各组件，调用Activity的findViewById（int id）方法来获取程序界面中的组件，接下来修改各组件的属性和方法即可。

实例：用LauncherActivity开发启动Activity的列表

通过前几章的实例已经介绍了Activity、ListActivity、TabActivity等基类的用法，接下来开发一个继承LauncherActivity的应用。

LauncherActivity继承了ListActivity，因此它本质上也是一个开发列表界面的Activity，但它开发出来的列表界面与普通列表界面有所不同。它开发出来的列表界面中的每个列表项都对应于一个Intent，因此当用户单击不同的列表项时，应用程序会自动启动对应的Activity。

使用LauncherActivity的方法并不难，由于依然是一个ListActivity，因此同样需要为它设置Adapter—既可使用简单的ArrayAdapter，也可使

用SimpleAdapter，当然还可以扩展BaseAdapter来实现自己的Adapter。与使用普通ListActivity不同的是，继承LauncherActivity时通常应该重写Intent intentForPosition（int position）方法，该方法根据不同列表项返回不同的Intent（用于启动不同的Activity）。

下面是LauncherActivity的一个子类的代码。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\MainActivity.java

上面程序中的第一行粗体字代码为该ListActivity设置了所需的内容Adapter，第二段粗体字代码则根据用户单击的列表项去启动对应的Activity。

上面的程序还用到了如下两个Activity。

➢ **ExpandableListActivityTest**：它是ExpandableListActivity的子类，用于显示一个可展开的列表窗口。

➢ **PreferenceActivityTest**：它是 PreferenceActivity 的子类，用于显示一个显示设置选项参数并进行保存的窗口。

实例：使用ExpandableListActivity实现可展开的Activity

先看ExpandableListActivityTest，它继承了ExpandableListActivity基类，ExpandableListActivity的用法与前面介绍的ExpandableListView的用法基本相似，只要为该Activity传入一个ExpandableListAdapter对象即可，接下来ExpandableListActivity将会生成一个显示可展开列表的窗口。

下面是ExpandableListActivityTest的代码。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\ExpandableListActivityTest.java

上面程序中的粗体字代码为ExpandableListActivity设置了一个ExpandableListAdapter对象，即可使得该Activity显示可展开列表的窗口。

实例：PreferenceActivity结合PreferenceFragment实现参数设置界面

PreferenceActivity是一个非常有用的基类，当我们开发一个Android应用程序时，不可避免地需要进行选项设置，这些选项设置会以参数的形式保存，习惯上我们会用Preferences进行保存。

提示：

关于Preferences的介绍，请参看本书第8章关于SharedPreferences的内容。

需要指出的是，如果Android应用程序中包含的某个Activity专门用于设置选项参数，那么Android为这种Activity提供了便捷的基类：PreferenceActivity。

一旦Activity继承了PreferenceActivity，那么该Activity完全不需要自己控制Preferences的读写，PreferenceActivity会为我们处理一切。

PreferenceActivity与普通Activity不同，它不再使用普通的界面布局文件，而是使用选项设置的布局文件。选项设置的布局文件以PreferenceScreen作为根元素—它表明定义一个参数设置的界面布局。

为了创建一个PreferenceActivity，需要先创建一个对应的界面布局文件。从Android 3.0开始，Android不再推荐直接让PreferenceActivity加载选项设置的布局文件，而是建议将PreferenceActivity与PreferenceFragment结合使用，其中PreferenceActivity只负责加载选项设置列表的布局文件，PreferenceFragment才负责加载选项设置的布局文件。

本实例中PreferenceActivity加载的选项设置列表的布局文件如下。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\res\xml\preference_headers.xml

上面的布局文件中定义了三个列表项，其中前两个列表项通过android：fragment选项指定启动相应的PreferenceFragment；第三个列表项通过<intent.../>子元素启动指定的Activity。

提示：

关于Intent与<intent.../>的介绍，请参考本书下一章的内容。

上面的布局文件中指定使用Prefs1Fragment、Prefs2Fragment两个内部类，为此我们将会在PreferenceActivityTest类中定义这两个内部类。下面是PreferenceActivityTest的代码。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\java\org\crazyit\app\PreferenceActivityTest.java

上面的Activity重写了PreferenceActivity的public void onBuildHeaders（List<Header> target）方法，重写该方法指定加载前面定义的preference_headers.xml界面布局文件。

上面的Activity中定义了两个PreferenceFragment，它们需要分别加载preferences.xml、display_prefs两个选项设置的布局文件。

建立选项设置的布局文件按如下步骤进行。

1 用鼠标右击Android Studio项目管理面板上的app节点，然后在弹出的右键菜单中单击“New”→“Android resource file”菜单，如图4.2所示。

2 Android Studio弹出如图4.3所示的窗口，选择创建“XML”类型的资源文件，该文件默认保存在/res/xml路径下。并选择该XML文件的根元素为PreferenceScreen，然后单击“Finish”按钮完成创建。

图4.2 新建Android资源文件

图4.3 创建Android XML文件

提示：

上面介绍的两步就是创建Android资源文件的通用步骤，包括界面布局文件、菜单资源文件、字符串资源等，都可通过这两步来进行创建。

接下来在Android Studio中打开preferences.xml进行编辑，Android Studio为编辑该文件提供了良好的提示信息，该XML文件能接收哪些子元素、各子元素能包含哪些属性，Android Studio中都有优秀的提示，如图4.4所示。

图4.4 preferences.xml文件中可添加的元素

3 其中PreferenceCategory用于对参数选项进行分组，其他元素都用于设置相应的参数。编辑preferences.xml文件完成后，可以看到如下所示的界面布局文件。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\res\xml\preferences.xml

上面的界面布局文件中定义了一个参数设置界面，其中包括两个参数设置组，而且该参数设置界面全面应用了各种元素，这样方便读者以后查询。

定义了参数设置的界面布局文件之后，接下来在PreferenceFragment程序中使用该界面布局文件进行参数设置、保存十分简单，只要如下两步即可。

1 让Fragment继承PreferenceFragment。

2 在onCreate（Bundle savedInstanceState）方法中调用addPreferencesFromResource（..）方法加载指定的界面布局文件。

上面的实例中还用到了选项设置布局文件display_prefs.xml，该布局文件的创建步骤与preferences.xml文件的创建步骤相同。该文件的代码如下。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\res\xml\display_prefs.xml

至此，我们为该应用程序开发了三个Activity类，但这三个Activity还不能使用，必须在AndroidManifest.xml清单文件中配置Activity才行。

4.1.2 配置Activity

Android应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加<activity.../>子元素即可配置Activity。例如如下的配置片段：

从上面的配置片段可以看出，配置Activity时通常指定如下几个属性。

➢ **name**：指定该Activity的实现类的类名。

➢ **icon**：指定该Activity对应的图标。

➢ **label**：指定该Activity的标签。

➢ **exported**：指定该Activity是否允许被其他应用调用。如果将该属性设为true，那么该Activity将可以被其他应用调用。

➢ **launchMode**：指定该Activity的加载模式，该属性支持standard、singleTop、singleTask和singleInstance这4种加载模式。本章后面会详细介绍这4种加载模式。

除此之外，配置Activity时通常还需要指定一个或多个<intent-filter.../>元素，该元素用于指定该Activity可响应的Intent。

提示：

关于Intent和IntentFilter的介绍，请参看本书下一章的介绍。

为了在AndroidManifest.xml文件中配置、管理上面的三个Activity，可以在清单文件的<application.../>元素中增加如下三个<activity.../>子元素。

程序清单：
codes\04\4.1\OtherActivity\app\src\main\AndroidManifest.xml

上面的配置片段配置了三个Activity，其中第一个Activity还配置了一个<intent-filter.../>元素，该元素指定该Activity作为应用程序的入口。

运行上面的应用程序，将看到如图4.5所示的界面。

在图4.5所示的程序界面中，用户单击任意列表项即可启动对应的Activity。例如，单击“设置程序参数”将会启动PreferenceActivityTest，单击“查看星际兵种”将会启动ExpandableListActivityTest。单击图4.5所示列表的第一个列表项，将看到如图4.6所示的界面。

图4.5 启动Activity的列表

图4.6 选项设置列表界面

图4.6所示界面就是利用PreferenceActivity生成的选项设置列表界面。这个界面只是包含三个列表项，其中前两个列表项用于启动PreferenceFragment，最后一个列表项将会根据Intent启动其他Activity。

单击图4.6所示界面中的第一个列表项，将可以看到如图4.7所示的界面。

图4.7所示界面就是利用PreferenceFragment生成的选项设置界面，这个界面非常漂亮，而且系统会自动将设置的参数永久地保存到系统中—这都得益于PreferenceActivity。例如，我们单击“填写用户名”列表项，系统将会显示如图4.8所示的输入框。

在图4.8所示对话框中输入用户名，单击“OK”按钮，程序将会自动保存设置的选项参数。程序所设置的参数将会保存在/data/data/<应用程序包名>/shared_prefs路径下，文件名为<应用程序包名>_preferences.xml。

运行本程序系统将会在/data/data/org.crazyit.app/shared_prefs/路径下生成一个org.crazyit.app_preferences.xml文件。打开DDMS的File Explorer面板，进入该参数的参数文件的保存路径，将可以看到如图4.9所示的窗口。

图4.7 PreferenceFragment生成的选项设置界面

图4.8 使用EditTextPreference生成的输入框

通过图4.9所示窗口把其中的“org.crazyit.app_preferences.xml”文件导出来，该文件的内容为：

上面的文件就是程序运行设置的参数。

如果单击图4.5所示列表的第二个列表项，将可以看到如图4.10所示的界面。

图4.9 查看系统生成的参数文件

图4.10 ExpandableListActivity生成的可展开的列表界面

4.1.3 启动、关闭Activity

正如前面所介绍的，一个Android应用通常都会包含多个Activity，但只有一个Activity会作为程序的入口一当该Android应用运行时将会自动启动并执行该Activity。至于应用中的其他Activity，通常都由入口Activity启动，或由入口Activity启动的Activity启动。

Activity启动其他Activity有如下两个方法。

➢ **startActivity（Intent intent）**：启动其他Activity。

➢ **startActivityForResult（Intent intent，int requestCode）**：以指定的请求码（requestCode）启动 Activity，而且程序将会获取新启动的 Activity 返回的结果（通过重写onActivityResult（..）方法来获取）。

启动Activity时可指定一个requestCode参数，该参数代表了启动Activity的请求码。这个请求码的值由开发者根据业务自行设置，用于标识请求来源。

上面两个方法都用到了Intent参数，Intent是Android应用里各组件之间通信的重要方式，一个Activity通过Intent来表达自己"意图"一想要启动哪个组件，被启动的组件既可是Activity组件，也可是Service组件。

Android为关闭Activity准备了如下两个方法。

➢ **finish（）**：结束当前Activity。

➢ **finishActivity（int requestCode）**：结束以startActivityForResult（Intent in tent，int requestCode）方法启动的Activity。

下面的示例程序示范了如何启动Activity，并允许程序在两个Activity之间切换。

程序的第一个Activity的界面布局很简单，该界面只包含一个按钮，该按钮用于进入第二个Activity。此处不给出界面布局文件，Java代码如下。

程序清单：
codes\04\4.1\StartActivity\app\src\main\java\org\crazyit\app\MainActivity.java

上面程序中的粗体字代码就是在Activity中启动其他Activity的关键代码。

程序中第二个Activity的界面同样简单，它只包含两个按钮，其中一个按钮用于简单地返回上一个Activity（并不关闭自己）；另一个按钮用于结束自己并返回上一个Activity。此处不给出第二个Activity的界面布局文件。

下面是第二个Activity的Java代码。

程序清单：
codes\04\4.1\StartActivity\app\src\main\java\org\crazyit\app\SecondActivity.java

上面程序中两个按钮的监听器里的处理代码只有一行区别：finish（），如果有这行代码，则表明该Activity会结束自己。

4.1.4 使用Bundle在Activity之间交换数据

当一个Activity启动另一个Activity时，常常会有一些数据需要传过去—这就像Web应用从一个Servlet跳到另一个Servlet时，习惯把需要交换的数据放入requestScope、sessionScope中。对于Activity而言，在Activity之间进行数据交换更简单，因为两个Activity之间本来就有一个“信使”：Intent，因此我们主要将需要交换的数据放入Intent中即可。

Intent提供了多个重载的方法来“携带”额外的数据，如下所示。

➢ **putExtras（Bundle data）**：向Intent中放入需要“携带”的数据包。

➢ **Bundle getExtras（）**：取出Intent中所“携带”的数据包。

➢ **putExtra（String name，Xxx value）**：向Intent中按key-value对的形式存入数据。

➢ **getXxxExtra（String name）**：从Intent中按key取出指定类型的数据。

上面方法中的Bundle就是一个简单的数据携带包，该Bundle对象包含了多个方法来存入数据。

➢ **putXxx（String key，Xxx data）**：向Bundle中放入Int、Long等各种类型的数据。

➢ **putSerializable（String key，Serializable data）**：向Bundle中放入一个可序列化的对象。

为了取出Bundle数据携带包里的数据，Bundle提供了如下方法。

➢ **getXxx（String key）：**从Bundle中取出Int、Long等各种类型的数据。

➢ **getSerializable（String key，Serializable data）：**从Bundle中取出一个可序列化的对象。

从上面的介绍不难看出，Intent主要通过Bundle对象来携带数据，因此Intent提供了putExtras（）和getExtras（）两个方法。除此之外，Intent也提供了多个重载的putExtra（String name，Xxx value）、getXxxExtra（String name），那么这些方法存取的数据在哪里呢？其实Intent提供的putExtra（String name，Xxx value）、getXxxExtra（String name）方法，只是两个便捷的方法，这些方法依然是存取Intent所携带的Bundle中的数据。

提示：

Intent的putExtra（String name，Xxx value）方法是"智能"的，当程序调用Intent的putExtra（String name，Xxx value）方法向Intent中存入数据时，如果该Intent中已经携带了Bundle对象，则该方法直接向Intent所携带的Bundle中存入数据；如果Intent还没有携带Bundle对象，putExtra（String name，Xxx value）方法会先为Intent创建一个Bundle，再向Bundle中存入数据。

下面通过一个实例应用来介绍两个Activity之间如何通过Bundle交换数据。

实例：用第二个Activity处理注册信息

本实例程序包含两个Activity，其中第一个Activity用于收集用户的输入信息，当用户单击该Activity的"注册"按钮时，应用进入第二个Activity，第二个Activity将会获取第一个Activity中的数据。

下面是第一个Activity的界面布局文件。

程序清单：
codes\04\4.1\BundleTest\app\src\main\res\layout\main.xml

该界面布局对应的Java类代码如下。

程序清单：
codes\04\4.1\BundleTest\app\src\main\java\org\crazyit\app\MainActivity.java

图4.11 注册界面

上面程序中的粗体字代码根据用户输入创建了一个Person对象，Person类只是一个简单的DTO对象，该Person类实现了java.io.Serializable接口，因此Person对象是可序列化的。

上面的程序创建了一个Bundle对象，并调用putSerializable（"person"，p）将Person对象放入该Bundle中，然后再使用Intent来“携带”这个Bundle，这样即可将Person对象传入第二个Activity。

运行该程序，第一个Activity显示的界面如图4.11所示。

当用户单击“注册”按钮时，程序将会启动ResultActivity，并将用户输入的数据传入该Activity。下面是ResultActivity的界面布局文件。

程序清单：codes\04\4.1\BundleTest\res\layout\result.xml

这个Activity的程序将会从Bundle中取出前一个Activity传过来的数据，并将它们显示出来。该Activity的Java代码如下。

程序清单：
codes\04\4.1\BundleTest\app\src\main\java\org\crazyit\app\ResultActivity.java

上面程序中的粗体字代码用于获取前一个Activity所传过来的数据，至于该Activity获取数据之后如何处理它们，完全由开发者自己决定。本应用程序只是将获取的数据显示出来，用户在图4.11所示界面中输入注册信息之后，单击“注册”按钮，将看到如图4.12所示的界面。

图4.12 注册成功

4.1.5 启动其他Activity并返回结果

前面已经提到，Activity还提供了一个startActivityForResult（Intent intent，int requestCode）方法来启动其他Activity。该方法用于启动指定Activity，而且期望获取指定Activity返回的结果。这种请求对于实际应用也是很常见的，例如应用程序第一个界面需要用户进行选择一但需要选择的列表数据比较复杂，必须启动另一个Activity让用户选择。当用户在第二个Activity中选择完成后，程序返回第一个Activity，第一个Activity必须能获取并显示用户在第二个Activity中选择的结果。在这种应用场景下，也是通过Bundle进行数据交换的。

为了获取被启动的Activity所返回的结果，需要从两方面着手。

➢ 当前Activity需要重写onActivityResult（int requestCode，int resultCode，Intent intent），当被启动的Activity返回结果时，该方法

将会被触发，其中requestCode代表请求码，而resultCode代表Activity返回的结果码，这个结果码也是由开发者根据业务自行设定的。

➢ 被启动的Activity需要调用setResult（）方法设置处理结果。

一个Activity中可能包含多个按钮，并调用多个startActivityForResult（）方法来打开多个不同的Activity处理不同的业务，当这些新Activity关闭后，系统都将回调前面Activity的onActivityResult（int requestCode，int resultCode，Intent data）方法。为了知道该方法是由哪个请求的结果所触发的，可利用requestCode请求码；为了知道返回的数据来自于哪个新的Activity，可利用resultCode结果码。

下面通过一个实例来介绍如何启动Activity并获取被启动Activity的结果。

实例：用第二个Activity让用户选择信息

本实例程序也包含两个Activity，第一个Activity的界面布局比较简单，它只包含一个按钮和一个文本框，故此处不再给出界面布局文件。第一个Activity对应的Java代码如下。

程序清单：
codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\app\MainActivity.java

上面程序中的粗体字代码用于启动ActivityForResult，并等待该Activity返回的结果一但问题是，该Activity如何获取ActivityForResult返回的结果呢？当前Activity启动SelectCityActivity之后，SelectCityActivity何时返回结果是不确定的，因此当前Activity无法去获取SelectCityActivity返回的结果。

为了让当前Activity获取SelectCityActivity所返回的结果，开发者应该重写onActivityResult（）方法—当被启动的SelectCityActivity返回结果时，onActivityResult（）方法将会被回调。

因此还需要为上面的ActivityForResult添加如下方法。

程序清单：
codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\activity\MainActivity.java

运行该程序，将看到如图4.13所示的界面。

单击图4.13所示界面中的"选择您所在城市"按钮，系统将会启动SelectCityActivity，该SelectCityActivity将会显示一个可展开的列表。该SelectCityActivity无须界面布局文件。该SelectCityActivity的Java代码如下。

图4.13 让用户选择所在城市

程序清单：
codes\04\4.1\ActivityForResult\app\src\main\java\org\crazyit\app\SelectCityActivity.java

上面的Activity只是一个普通的显示可展开列表的Activity，程序还为该Activity的各子列表项绑定了事件监听器，当用户单击子列表项时，该Activity将会把用户选择城市返回给上一个Activity。

当上一个Activity获取SelectCityActivity选择城市之后，将会把该结果显示在图4.13所示界面右边的文本框内。

4.2 Activity的回调机制

有Web开发经验的读者都知道：当一个Servlet开发出来之后，该Servlet运行于Web服务器中。服务器何时创建Servlet的实例，何时回调Servlet的方法向用户生成响应，程序员无法控制，这种回调由服务器自行决定。

前面已经提到，Android应用中的Activity与Web应用中的Servlet有点相似，也就是说，Activity被开发出来之后，开发者只要在AndroidManifest.xml文件中配置该Activity即可。至于该Activity何时被实例化，它的方法何时被调用，对开发者来说是完全透明的。

当开发者开发一个Servlet时，根据不同的需求场景，可能需要选择性地实现如下方法。

- **init(ServletConfig config)**
- **destroy()**
- **doGet(HttpServletRequest req,HttpServletResponse resp)**
- **doPost(HttpServletRequest req,HttpServletResponse resp)**
- **service(HttpServletRequest req,HttpServletResponse resp)**

当把这个Servlet部署在Web应用中之后，Web服务器将会在特定的时刻，调用该Servlet上面的各种方法—这种调用就被称为回调。

所谓回调，在实现具有通用性质的应用架构时非常常见：对于一个具有通用性质的程序架构来说，程序架构完成整个应用的通用功能、流程，但在某个特定的点上，需要一段业务相关的代码—通用的程序架构无法实现这段代码，那么程序架构会在这个点上留一个“空”。

对于Java程序来说，程序架构在某个点上留的“空”，可以以如下两种方式存在。

➢ **以接口形式存在：**该接口由开发者实现，实现该接口时将会实现该接口的方法，那么通用的程序架构就会回调该方法来完成业务相关的处理。

➢ **以抽象方法（也可以是非抽象方法）的形式存在：**这就是Activity的实现形式。在这些特定的点上方法已经被定义了，如onCreate、onActivityResult等方法，开发者可以有选择性地重写这些方法，通用的程序架构就会回调该方法来完成业务相关的处理。

提示：

回调机制的第一种实现方式就是典型的命令者模式，关于命令者模式请参考疯狂Java体系的《轻量级Java EE企业应用实战》第9章。

前面介绍的事件处理也用到了回调机制：当开发者开发一个组件时，如果开发者需要该组件能响应特定的事件，则可以有选择性地实现该组件的特定方法—当用户在该组件上激发某个事件时，该组件上特定的方法就会回调。

Activity的回调机制也与此类似，当Activity被部署在Android应用中之后，随着应用程序的运行，Activity会不断地在不同的状态之间切换，该Activity中特定的方法就会被回调—开发者就可以有选择性地重写这些方法来加入业务相关的处理。

Activity运行过程所处的不同状态也被称为生命周期，下面将详细介绍Activity的生命周期。

4.3 Activity的生命周期与加载模式

当Activity处于Android应用中运行时，它的活动状态由Android以Activity栈的形式管理，当前活动的Activity位于栈顶。随着不同应用的

运行，每个Activity都有可能从活动状态转入非活动状态，也可能从非活动状态转入活动状态。

4.3.1 Activity的生命周期演示

归纳起来，Activity大致会经过如下4种状态。

➢ **运行状态**：当前Activity位于前台，用户可见，可以获得焦点。

➢ **暂停状态**：其他Activity位于前台，该Activity依然可见，只是不能获得焦点。

➢ **停止状态**：该Activity不可见，失去焦点。

➢ **销毁状态**：该Activity结束，或Activity所在的进程被结束。

图4.14 A ctivity的生命周期及相关回调方法

图4.14（该图参照了Android官方文档，但细节方面比官方文档更准确）显示了Activity的生命周期及相关回调方法。

从图4.14可以看出，在Activity的生命周期中，如下方法会被系统回调。

➢ **onCreate（Bundle savedStatus）**：创建Activity时被回调。该方法只会被调用一次。

➢ **onStart（）**：启动Activity时被回调。

➢ **onRestart（）**：重新启动Activity时被回调。

➢ **onResume（）**：恢复 Activity 时被回调。在onStart（）方法后一定会回调onResume（）方法。

➢ **onPause（）：** 暂停Activity时被回调。

➢ **onStop（）：** 停止Activity时被回调。

➢ **onDestroy（）：** 销毁Activity时被回调。该方法只会被调用一次。

正如开发Servlet时可以根据需要有选择性地覆盖指定方法一样，开发Activity时也可根据需要有选择性地覆盖指定方法。其中最常见的就是覆盖onCreate（Bundle saveStatus）方法—前面所有示例都覆盖了Activity的onCreate（Bundle saveStatus）方法，该方法用于对该Activity执行初始化。除此之外，覆盖onPause（）方法也很常见，比如用户正在玩一个游戏，此时有电话进来，那么我们需要将当前游戏暂停，并保存该游戏的进行状态，这就可以通过覆盖onPause（）方法来实现。接下来当用户再次切换到游戏状态时，onResume（）方法已经被回调，因此可以通过重写onResume（）方法来恢复游戏状态。

下面的Activity覆盖了上面的7个生命周期方法，并在每个方法中增加了一行记录日志代码。该 Activity 的界面布局很简单，包含了两个按钮，其中一个用于启动一个对话框风格的Activity；另一个用于退出该应用。此处不给出界面布局代码。该Activity的Java代码如下。

程序清单：
codes\04\4.3\Lifecycle\app\src\main\java\org\crazyit\app\MainActivity.java

将该Activity设置成程序的入口Activity，当程序启动时将会自动启动并执行该Activity，此时将可以在Android Studio的LogCat窗口看到如图4.15所示的输出。

图4.15 启动Activity时回调的方法

单击该程序界面上的“启动对话框风格的Activity”按钮，对话框风格的Activity进入前台，虽然MainActivity不能获得焦点，但依然“部分可见”，此时该Activity进入“暂停”状态。此时将可以在Android Studio的LogCat窗口看到如图4.16所示的输出。

图4.16 暂停Activity时回调的方法

在当前状态下，按下模拟器右边的键，返回Lifecycle Activity，该Activity再次进入“运行”状态，此时看到LogCat窗口有如图4.17所示的输出。

图4.17 恢复Activity时回调的方法

在当前程序运行状态下，按下模拟器右边的键，返回系统桌面，当前该Activity将失去焦点且不可见。但该Activity并未被销毁，它进入“停止”状态，此时看到LogCat窗口有如图4.18所示的输出。

图4.18 停止Activity时回调的方法

在模拟器程序列表处再次找到该应用程序并启动它，将可以看到LogCat窗口有如图4.19所示的输出。

图4.19 重新启动Activity时回调的方法

如果用户单击该程序界面上的“退出”按钮，该Activity将会结束自己，并且可以在LogCat窗口看到如图4.20所示的输出。

图4.20 结束Activity时回调的方法

通过上面的运行过程，相信读者对Activity的生命周期状态及在不同状态之间切换时所回调的方法有了很清晰的认识。

4.3.2 Activity与Servlet的相似性和区别

虽然很少有人会把Activity和Servlet放在一起对比，但就笔者的经验来看，Activity与Servlet之间确实存在不少相似之处，如果读者已经具备一定的Web开发经验，通过这种“触类旁通”式的学习，相信可以更好地理解Activity的设计思想。

笔者一直坚信“温故而知新”是一种好的学习方式：当我们学习一门新的知识时，最好能找到新知识和已掌握知识之间的类比关系，这样既可迅速获得对新知识的直观把握，又可巩固已掌握的旧知识，避免“知识越学越多”的困扰。你的知识越积累越多，你会发现眼界越来越高，视野越来越广，看问题更容易简明、扼要地把握本质。

当然，两门知识之间除了存在相似的地方之外，也少不了差异，这部分差异正是需互相参考、互相对照的部分，以求深入理解各自的设计思想、原理。当我们学习知识时，除了掌握怎么用它之外，最好能站在设计者的角度来看：他为何要设计这个类？这个类为何要包含这些方法？方法为何要有这些形参？理解这些设计，剩下的也就简单了。

Activity与Servlet的相似之处大致如下。

➢ Activity、Servlet的职责都是向用户呈现界面。

➢ 开发者开发Activity、Servlet都继承系统的基类。

➢ Activity、Servlet开发出来之后都需要进行配置。

➢ Activity运行于Android应用中，Servlet运行于Web应用中。

➢ 开发者无须创建 Activity、Servlet 的实例，无须调用它们的方法。Activity、Servlet的方法都由系统以回调的方式来调用。

➢ Activity、Servlet都有各自的生命周期，它们的生命周期都由外部负责管理。

➢ Activity、Servlet都不会直接相互调用，因此都不能直接进行数据交换。Servlet之间的数据交换需要借助于Web应用提供的requestScope、sessionScope等；Activity之间的数据交换要借助于Bundle。

当然，Activity与Servlet之间的差别很大，因为它们本身所在场景是完全不同的，它们之间的区别也很明显。

➢ Activity是Android窗口的容器，因此Activity最终以窗口的形式显示出来。而Servlet并不会生成应用界面，只是向浏览者生成文本响应。

➢ Activity运行于Android应用中，因此Activity的本质还是通过各种界面组件来搭建界面；而Servlet则主要以IO流向浏览者生成文本响应，浏览者看到的界面其实是由浏览器负责生成的。

➢ Activity之间的跳转主要通过Intent对象来控制；而Servlet之间的跳转则主要由用户请求来控制。

4.3.3 Activity的4种加载模式

正如前面介绍Activity配置时提到的，配置Activity时可指定android:launchMode属性，该属性用于配置该Activity的加载模式。该属性支持如下4个属性值。

➢ **standard**：标准模式，这是默认的加载模式。

➢ **singleTop**：Task栈顶单例模式。

➢ **singleTask**：Task内单例模式。

➢ **singleInstance**：全局单例模式。

可能有读者会问：为什么要为Activity指定加载模式？加载模式有什么用？在讲解Activity的加载模式之前，先介绍Android对Activity的管理。Android采用Task来管理多个Activity，当我们启动一个应用时，Android就会为之创建一个Task，然后启动这个应用的入口Activity（即<intent-filter.../>中配置为MAIN和LAUNCHER的Activity）。

Android的Task是一个有点麻烦的概念—因为Android并没有为Task提供API，因此开发者无法真正去访问Task，只能调用Activity的getTaskId（）方法来获取它所在的Task的ID。事实上，我们可以把Task理解成Activity栈，Task以栈的形式来管理Activity：先启动的Activity被放在Task栈底，后启动的Activity被放在Task栈顶。

那么Activity的加载模式，就负责管理实例化、加载Activiyt的方式，并可以控制Activity与Task之间的加载关系。

下面详细介绍这4种加载模式。

1.standard模式

每次通过这种模式来启动目标Activity时，Android总会为目标Activity创建一个新的实例，并将该Activity添加到当前Task栈中—这种模式不会启动新的Task，新Activity将被添加到原有的Task中。

下面的示例使用了standard模式来不断启动自身。

程序清单：
codes\04\4.3\StandardTest\app\src\main\java\org\crazyit\app\MainActivity.java

正如上面的粗体字代码所示，每次单击按钮，程序都会再次启动MainActivity，程序配置该Activity时无须指定launchMode属性，该Activity默认采用standard加载模式。

运行该程序，多次单击程序界面上的“启动MainActivity”按钮，程序将会不断启动新的MainActivity实例（不同Activity实例的hashCode值有差异），但它们所在的Task ID总是相同的—这表明这种加载模式不会使用全新的Task。

standard加载模式的示意图如图4.21所示。

图4.21 s tandard加载模式的示意图

正如图4.21所示，当用户单击手机的“返回”键时，系统将会“逐一”从Activity栈顶删除Activity实例。

2.singleTop模式

这种模式与standard模式基本相似，但有一点不同：当将要启动的目标Activity已经位于Task栈顶时，系统不会重新创建目标Activity的实例，而是直接复用已有的Activity实例。

如果将上面实例中MainActivity的加载模式改为singleTop，那么无论用户单击多少次按钮，界面上的程序都不会有任何变化。

如果将要启动的目标Activity没有位于Task栈顶，此时系统会重新创建目标Activity的实例，并将它加载到Task栈顶—此时与standard模式完全相同。

3.singleTask模式

采用这种加载模式的Activity在同一个Task内只有一个实例，当系统采用singleTask模式启动目标Activity时，可分为如下三种情况。

➢ 如果将要启动的目标Activity不存在，系统将会创建目标Activity的实例，并将它加入Task栈顶。

➢ 如果将要启动的目标Activity已经位于Task栈顶，此时与singleTop模式的行为相同。

➢ 如果将要启动的目标Activity已经存在、但没有位于Task栈顶，系统将会把位于该Activity上面的所有Activity移出Task栈，从而使得目标Activity转入栈顶。

下面的示例示范了上面第三种情况。该实例包含两个Activity，其中第一个Activity上显示文本框和按钮，该按钮用于启动第二个Activity；第二个Activity上显示文本框和按钮，该按钮用于启动第一个Activity。

第一个Activity的代码如下。

程序清单：
codes\04\4.3\SingleTaskTest\app\src\main\java\org\crazyit\app\SingleTaskTest.java

正如上面的粗体字代码所示，当用户单击该Activity上的按钮时，系统将会启动SecondActivity。下面是SecondActivity的代码。

程序清单：
codes\04\4.3\SingleTaskTest\app\src\main\java\org\crazyit\app\SecondActivity.java

正如上面的粗体字代码所示，当用户单击该Activity上的按钮时，系统将会启动SingleTaskTest。程序将该SecondActivity的加载模式配置成singleTask。运行该示例，系统默认启动SingleTaskTest Activity，单击

该界面上的按钮，系统将以singleTask模式打开SecondActivity，如图4.22所示。

图4.22 singleTask模式

图4.22所示界面的Task栈中目前有两个Activity（从底向上）：SingleTaskTest→SecondActivity。

单击图4.22所示界面上的"启动SingleTaskTest"按钮，系统以标准模式再次加载一个新的SingleTaskTest Activity。此时Task栈中有三个Activity（从底向上）：SingleTaskTest→SecondActivity→SingleTaskTest。

在SingleTaskTest的界面上再次单击按钮，系统将会以singleTask模式再次打开SecondActivity，系统会将位于SecondActivity上面的所有Activity移出，使得SecondActivity转入栈顶。此时Task栈中只有两个Activity（从底向上）：SingleTaskTest→SecondActivity，也就是再次恢复到图4.22所示的状态。

4.singleInstance模式

在这种加载模式下，系统保证无论从哪个Task中启动目标Activity，只会创建一个目标Activity实例，并会使用一个全新的Task栈来加载该Activity实例。

当系统采用singleInstance模式启动目标Activity时，可分为如下两种情况。

➢ 如果将要启动的目标 Activity 不存在，系统会先创建一个全新的Task，再创建目标Activity的实例，并将它加入新的Task栈顶。

➢ 如果将要启动的目标Activity已经存在，无论它位于哪个应用程序中、位于哪个Task中，系统都会把该Activity所在的Task转到前台，从

而使该Activity显示出来。

需要指出的是，采用singleInstance模式加载Activity总是位于Task栈顶，且采用singleInstance模式加载的Activity所在Task将只包含该Activity。

下面示例的SingleInstanceTest中包含一个按钮，当用户单击该按钮时，系统启动SecondActivity。程序清单如下。

程序清单：
codes\04\4.3\SingleInstanceTest\app\src\main\java\org\crazyit\app\SingleInstanceTest.java

上面的粗体字代码指定单击按钮时将会启动SecondActivity，将该SecondActivity配置成singleInstance加载模式，并且将该Activity的exported属性配置成true—表明该Activity可被其他应用启动。

配置该Activity的配置片段如下：

配置该Activity时，将它的exported属性设为true，表明允许通过其他程序来启动该Activity；配置该Activity时还配置了<intent-filter.../>子元素，这表明该Activity可通过隐式Intent启动—关于隐式Intent的介绍，请参考下一章的内容。

图4.23 singleInstance加载模式

运行该示例，系统默认显示SingleInstanceTest，当用户单击该Activity界面上的按钮时，系统将会采用singleInstance模式加载SecondActivity：系统启动新的Task，并用新的Task加载新创建的

SecondActivity实例，SecondActivity总是位于该新Task栈顶。此时可看到如图4.23所示的界面。

另一个示例将采用隐式Intent再次启动该SecondActivity。下面是采用隐式Intent启动SecondActivity的示例代码。

程序清单：
codes\04\4.3\OtherTest\app\src\main\java\org\crazyit\other\MainActivity.java

运行该示例，系统默认显示OtherTest，当用户单击该Activity界面上的按钮时，系统将采用隐式Intent来启动SecondActivity。注意SecondActivity的加载模式是singleInstance，如果前一个示例还未退出，无论SecondActivity所在Task是否位于前台，系统都将再次把SecondActivity所在的Task转入前台，从而将SecondActivity显示出来。也就是说，系统将会再次显示如图4.23所示的界面。

4.4 Fragment详解

第2章介绍ActionBar时已经用到了Fragment，Fragment是Android 3.0引入的新API。Fragment代表了Activity的子模块，因此可以把Fragment理解成Activity片段（Fragment本身就是片段的意思）。Fragment拥有自己的生命周期，也可以接受它自己的输入事件。

4.4.1 Fragment概述及其设计初衷

Fragment必须被“嵌入”Activity中使用，因此，虽然Fragment也拥有自己的生命周期，但Fragment的生命周期会受它所在的Activity的生命周期控制。例如，当Activity暂停时，该Activity内的所有Fragment都会暂停；当Activity被销毁时，该Activity内的所有Fragment都会被销毁。只有当该Activity处于活动状态时，程序员才可通过方法独立地操作Fragment。

关于Fragment，可以归纳出如下几个特征。

➢ Fragment总是作为Activity界面的组成部分。Fragment可调用getActivity（）方法获取它所在的 Activity，Activity 可调用FragmentManager 的 findFragmentById（）或findFragmentByTag（）方法来获取Fragment。

➢ 在Activity运行过程中，可调用FragmentManager的add（）、remove（）、replace（）方法动态地添加、删除或替换Fragment。

➢ 一个Activity可以同时组合多个Fragment；反过来，一个Fragment也可被多个Activity复用。

➢ Fragment 可以响应自己的输入事件，并拥有自己的生命周期，但它们的生命周期直接被其所属的Activity的生命周期控制。

Android 3.0引入Fragment的初衷是为了适应大屏幕的平板电脑，由于平板电脑的屏幕比手机屏幕更大，因此可以容纳更多的UI组件，且这些UI组件之间存在交互关系。Fragment简化了大屏幕UI的设计，它不需要开发者管理组件包含关系的复杂变化，开发者使用Fragment对UI组件进行分组、模块化管理，就可以更方便地在运行过程中动态更新Activity的用户界面。

图4.24 F ragment的设计初衷

例如有如图4.24所示的新闻浏览界面，该界面需要在屏幕左边显示新闻列表，并在屏幕右边显示新闻内容，此时就可以在Activity中显示两个并排的Fragment：左边的Fragment显示新闻列表，右边的Fragment显示新闻内容。由于每个Fragment都拥有自己的生命周期，并可响应用户输入事件，因此可以非常方便地实现：当用户单击左边列表中的指定新闻时，右边的Fragment就会显示相应的新闻内容。图4.24所示左边的“平板电脑”部分显示了这种UI界面。

通过使用上面的Fragment设计机制，可以取代传统的让一个Activity显示新闻列表，另一个Activity显示新闻内容的设计。

由于Fragment是可复用的组件，因此如果需要在正常尺寸的手机屏幕上运行该应用，则可以改为使用两个Activity：ActivityA包含FragmentA、ActivityB包含FragmentB。其中ActivityA仅包含显示文章列表的FragmentA，而当用户选择一篇文章时，它会启动包含新闻内容的ActivityB，如图4.24所示右边的“手机”部分。由此可见，Fragment可以很好地支持图4.24所示的两种设计模式。

4.4.2 创建Fragment

与创建Activity类似，开发者实现的Fragment必须继承Fragment基类，Android提供了如图4.25所示的Fragment继承体系。

开发者实现的Fragment，可以根据需要继承图4.25所示的Fragment基类或它的任意子类。接下来，实现Fragment与实现Activity非常相似，它们都需要实现与 Activity 类似的回调方法，例如 onCreate（）、onCreateView（）、onStart（）、onResume（）、onPause（）、onStop（）等。

图4.25 F ragment继承体系

提示：

开发Fragment与开发Activity非常相似，区别只是开发Activity需要继承Activity或其子类；而开发Fragment需要继承Fragment及其子类。与此同时，只要将原来写在Activity回调方法中的代码“移到”Fragment的回调方法中即可。

通常来说，创建Fragment需要实现如下三个方法。

➢ **onCreate（）：**系统创建 Fragment 对象后回调该方法，在实现代码中只初始化想要在Fragment中保持的必要组件，当Fragment被暂停或者停止后可以恢复。

➢ **onCreateView（）：**当 Fragment 绘制界面组件时会回调该方法。该方法必须返回一个View，该View也就是该Fragment所显示的View。

➢ **onPause（）：**当用户离开该Fragment时将会回调该方法。

对于大部分Fragment而言，通常都会重写上面这三个方法。但是实际上开发者可以根据需要重写Fragment的任意回调方法，后面将会详细介绍Fragment的生命周期及其回调方法。

为了控制Fragment显示的组件，通常需要重写onCreateView（）方法，该方法返回的View将作为该Fragment显示的View组件。当Fragment绘制界面组件时将会回调该方法。

例如如下方法片段：

上面方法中的第一行粗体字代码使用LayoutInflater加载了/res/layout/目录下的fragment_book_detail.xml布局文件。最后一行粗体字代码返回该布局文件对应的View组件，这表明该Fragment将会显示该View组件。

实例：开发显示图书详情的Fragment

下面Fragment将会加载并显示一份简单的界面布局文件，并根据传入的参数来更新界面组件。该Fragment的代码如下。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\BookDetailFragment.java

上面的Fragment将会加载并显示res/layout/目录下的fragment_book_detail.xml界面布局文件。上面①号代码获取启动该Fragment时传入的ITEM_ID参数，并根据该ID获取BookContent的ITEM_MAP中的图书信息。

BookContent类用于模拟系统的数据模型，该模拟类的代码如下。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\model\BookContent.java

BookDetailFragment只是加载并显示一份简单的布局文件，这份布局文件中通过LinearLayout包含两个文本框。该布局文件的代码如下。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\res\layout\fragment_book_detail.xml

实例：创建ListFragment

如果开发ListFragment的子类，则无须重写onCreateView（）方法一与ListActivity类似的是，只要调用ListFragment的setAdapter（）方法为该Fragment设置Adapter即可。该ListFragment将会显示该Adapter提供的列表项。

本实例开发了一个ListFragment的子类。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\BookListFragment.java

为了控制ListFragment显示的列表项，只要调用ListFragment提供的setAdapter（）方法，即可让该ListFragment显示该Adapter所提供的多个列表项。

4.4.3 Fragment与Activity通信

为了在Activity中显示Fragment，还必须将Fragment添加到Activity中。将Fragment添加到Activity中有如下两种方式。

➢ 在布局文件中使用<fragment.../>元素添加Fragment，<fragment.../>元素的android：name属性指定Fragment的实现类。

➢ 在Java代码中通过FragmentTransaction对象的add（）方法来添加Fragment。

Activity的getFragmentManager（）方法可返回FragmentManager，FragmentManager对象的beginTransaction（）方法即可开启并返回FragmentTransaction对象。

下面Activity首先通过如下布局文件来使用前面定义的BookListFragment。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\res\layout\activity_book_twopane.xml

上面布局文件中的粗体字代码使用<fragment.../>元素添加了BookListFragment，该Activity的左边将会显示一个ListFragment，右边只是一个FrameLayout容器，该FrameLayout容器将会动态更新其中显示的Fragment。下面是该Activity的代码。

程序清单：
codes\04\4.4\FragmentTest\app\src\main\java\org\crazyit\app\MainActivity.java

上面程序中的①号粗体字代码就调用了FragmentTransaction的replace（）方法动态更新了ID为book_detail_container容器（也就是前面布局文件中的FrameLayout容器）中显示的Fragment。

将Fragment添加到Activity之后，Fragment必须与Activity交互信息，这就需要Fragment能获取它所在的Activity，Activity也能获取它所包含的任意的Fragment。可按如下方式进行。

➢ **Fragment获取它所在的Activity**：调用Fragment的getActivity（）方法即可返回它所在的Activity。

➢ **Activity 获取它包含的 Fragment**：调用 Activity 关联的FragmentManager 的findFragmentById（int i d）或findFragmentByTag（String ta g）方法即可获取指定的Fragment。

提示：

在界面布局文件中使用<fragment.../>元素添加Fragment时，可以为<fragment.../>元素指定android：id或android：tag属性，这两个属性都可用于标识该Fragment，接下来Activity将可通过findFragmentById（int id）或findFragmentByTag （String tag）来获取该Fragment。

除此之外，Fragment与Activity可能还需要相互传递数据，可按如下方式进行。

➢ **Activity向Fragment传递数据：**在Activity中创建Bundle数据包，并调用Fragment的setArguments（Bundle bundle）方法即可将Bundle数据包传给Fragment。

➢ **Fragment向Activity传递数据或Activity需要在Fragment运行中进行实时通信：**在Fragment中定义一个内部回调接口，再让包含该Fragment的Activity实现该回调接口，这样Fragment即可调用该回调方法将数据传给Activity。

上面示例定义了两个Fragment，并使用一个Activity来“组合”这两个Fragment，该示例的运行界面正是实现图4.24中示意界面的左边部分。使用大屏幕设备运行该程序，可以看到如图4.26所示的界面。

图4.26 组合Fragment实现UI界面

4.4.4 Fragment管理与Fragment事务

前面介绍了Activity与Fragment交互相关的内容，其实Activity管理Fragment主要依靠FragmentManager。

FragmentManager可以完成如下几方面的功能。

➢ 使用findFragmentById（）或findFragmentByTag（）方法来获取指定Fragment。

➢ 调用popBackStack（）方法将Fragment从后台栈中弹出（模拟用户按下BACK按键）。

➢ 调用addOnBackStackChangeListener（）注册一个监听器，用于监听后台栈的变化。

如果需要添加、删除、替换Fragment，则需要借助于FragmentTransaction对象，FragmentTransaction代表Activity对Fragment执行的多个改变。

提示：

FragmentTransaction也被翻译为Fragment事务。与数据库事务类似的是，数据库事务代表了对底层数组的多个更新操作；而Fragment事务则代表了Activity对Fragment执行的多个改变操作。

开发者可通过FragmentManager来获得FragmentTransaction，代码片段如下：

每个FragmentTransaction可以包含多个对Fragment的修改，比如包含调用了多个add（）、remove（）、和replace（）操作，最后调用commit（）方法提交事务即可。

在调用commit（）之前，开发者也可调用addToBackStack（）将事务添加到Back栈，该栈由Activity负责管理，这样允许用户按BACK按键返回到前一个Fragment状态。

在上面的示例代码中，newFragment 替换了当前界面布局中ID为fragment_container的容器内的Fragment，由于程序调用addToBackStack（）将该replace操作添加到了Back栈中，因此用户可以通过按BACK按键返回替换之前的状态。

前面介绍的示例在大屏幕的平板电脑上运行良好，但在小屏幕的手机上运行就会有问题：运行界面非常丑陋。图4.27显示了在小屏幕手机上的运行效果。

图4.27 小屏幕手机上运行组合Fragment实现的UI界面

如果希望开发的应用既可在大屏幕平板电脑上使用，也可在小屏幕手机上使用，可以考虑开发兼顾屏幕分辨率的应用。

实例：开发兼顾屏幕分辨率的应用

为了开发兼顾屏幕分辨率的应用，可以考虑在res/目录下为大屏幕、600dpi的屏幕建立相应的资源文件夹：values-large、values-sw600dp，在该文件夹下建立一个名为refs.xml的引用资源文件。该引用资源文件专门用于定义各种引用项。

本实例的引用资源文件中只有一项，下面是该引用资源文件的代码。

程序清单：
codes\04\4.4\SeniorFragmentTest\app\src\main\res\values-large\refs.xml

上面引用资源文件中指定activity_book_list引用/res/layout/目录下的activity_book_twopane.xml界面布局文件。

接下来在Activity加载R.layout.activity_book_list时将会根据运行平台的屏幕大小自动选择界面布局文件：在大屏幕的平板电脑上，R.layout.activity_book_list将会变成/res/layout/目录下的activity_book_twopane界面布局文件；在小屏幕的手机上，R.layout.activity_book_list依然引用/res/layout/目录下的activity_book_list.xml界面布局文件。

下面是activity_book_list.xml界面布局文件的代码。

程序清单：
codes\04\4.4\SeniorFragmentTest\app\src\main\res\layout\activity_book_list.xml

从上面的布局文件可以看出，该布局文件仅仅显示BookListFragment组件，这表明该界面布局文件中只是显示图书列表。

接下来加载该布局文件的Activity将会针对不同的屏幕分辨率分别进行处理。下面是该Activity的代码。

程序清单：
codes\04\4.4\SeniorFragmentTest\app\src\main\java\org\crazyit\app\BookListActivity.java

由于该实例为大屏幕设备定义了引用资源文件，因此该应用将会根据大屏幕加载对应的界面布局文件，因此上面程序中第一段粗体字代码会判断界面布局中是否包含ID为book_detail_container的组件，如果包含该组件，则表明是适应大屏幕的“双屏”界面；否则，该程序界面上只包含一个简单的BookListFragment列表组件。此时当用户单击列表项时，程序不再是简单地更换Fragment，而是启动BookDetailActivity来显示指定图书的详细信息。

BookDetailActivity只是一个简单的封装，它将直接复用前面已有的BookDetailFragment。BookDetailActivity的代码如下。

程序清单：
codes\04\4.4\SeniorFragmentTest\app\src\main\java\org\crazyit\app\BookDetailActivity.java

上面的粗体字代码创建了BookDetailFragment，并让该Activity显示该Fragment即可。

除此之外，该Activity还启用了ActionBar上的应用程序图标，允许用户点击该图标返回程序的主Acitivity。

在大屏幕设备上运行该实例，可以看到如图4.28所示的界面。

单击其中一个列表项，将可以看到4.29所示的界面。

图4.28 图书列表界面

图4.29 图书详情界面

4.5 Fragment的生命周期

与Activity类似的是，Fragment也存在如下状态。

➢ **运行状态：**当前Fragment位于前台，用户可见，可以获得焦点。

➢ **暂停状态：**其他Activity位于前台，该Fragment依然可见，只是不能获得焦点。

➢ **停止状态：**该Fragment不可见，失去焦点。

➢ **销毁状态：**该Fragment被完全删除，或该Fragment所在的Activity被结束。

提示：

Android文档只提到了Fragment的三个状态，官方文档没有提到Fragment的“销毁状态”。这也是合理的，因为处于“销毁状态”的Fragment基本不可用了，只能等着被回收。

图4.30（该图参照Android官方文档，有些地方与文档有差异，以实际运行结果为准）显示了Fragment的生命周期及相关回调方法。

从图4.30可以看出，在Fragment的生命周期中，如下方法会被系统回调。

➢ **onAttach（）：** 当该Fragment被添加到Activity时被回调。该方法只会被调用一次。

➢ **onCreate（Bundle savedStatus）：** 创建Fragment时被回调。该方法只会被调用一次。

➢ **onCreateView（）：** 每次创建、绘制该Fragment的View组件时回调该方法，Fragment 将会显示该方法返回的 View组件。

➢ **onActivityCreated（）：** 当Fragment所在的Activity被启动完成后回调该方法。

➢ **onStart（）：** 启动Fragment时被回调。

➢ **onResume（）：** 恢复Fragment时被回调，在 onStart（） 方 法 后 一 定 会 回 调onResume（）方法。

➢ **onPause（）：** 暂停Fragment时被回调。

➢ **onStop（）：** 停止Fragment时被回调。

➢ **onDestroyView（）：** 销毁该Fragment所包含的View组件时调用。

➢ **onDestroy（）：** 销毁Fragment时被回调。该方法只会被调用一次。

➢ **onDetach（）：** 将该Fragment从Activity中删除、替换完成时回调该方法，在onDestroy（）方法后一定会回调onDetach（）方法。该

方法只会被调用一次。

正如开发Activity时可以根据需要有选择性地覆盖指定方法一样，开发Fragment时也可根据需要有选择性地覆盖指定方法。其中最常见的就是覆盖onCreateView（）方法—该方法返回的View将由Fragment显示出来。

图4.30 F ragment的生命周期及相关回调方法

下面的Fragment覆盖了上面的11个生命周期方法，并在每个方法中增加了一行记录日志代码。该Fragment的代码如下。

程序清单：
codes\04\4.5\FragmentLifecycle\app\src\main\java\org\crazyit\app\LifecycleFragment.java

包含该Fragment的Activity的布局文件很简单：该布局文件中只包含一个容器来盛装Fragment，另外还包含几个按钮。此处不给出界面布局代码。

当使用Activity加载该Fragment时，可以在LogCat控制台看到如图4.31所示的输出。

图4.31 启动Fragment时回调的方法

如果单击程序中的“启动对话框风格的Activity”按钮，将启动一个对话框风格的Activity，当前Activity将会转入“暂停”状态，该Fragment已经

进入“暂停”状态，此时可以在LogCat控制台看到如图4.32所示的输出。

图4.32 暂停Fragment时回调的方法

关闭对话框风格的Activity，Fragment将会再次进入“运行”状态，将可以在LogCat控制台看到如图4.33所示的输出。

图4.33 恢复Fragment时回调的方法

单击程序界面上的“替换目标Fragment，并加入Back栈”，将可以在LogCat控制台看到如图4.34所示的输出。

图4.34 将Fragment加入Back栈时回调的方法

从图4.34可以看出，替换目标Fragment，并将它添加到Back栈中，此时该Fragment虽然不可见，但它并未被销毁，它只是被添加到后台的Back栈中。当用户按下手机的键时，该Fragment将会再次显示出来，此时可以在LogCat控制台看到如图4.35所示的输出。

图4.35 将Back栈中Fragment转入前台时回调的方法

单击程序界面上的“替换目标Fragment”或“退出”按钮，该Fragment将会被完全结束，Fragment将进入“销毁”状态，此时可以在LogCat控制台看到如图4.36所示的输出。

图4.36 结束Fragment时回调的方法

4.6 本章小结

本章详细介绍了Android四大组件之一：Activity。一个Android应用将会包含多个Activity，每个Activity通常对应于一个窗口。普通用户接触最多的就是Activity。学习本章的重点是掌握如何开发Activity，如何在AndroidManifest.xml文件中配置Activity。不仅如此，由于Android系统通常会由多个Activity组成，因此读者还需要掌握启动其他Activity的方法，包括如何利用Bundle在不同Activity之间通信，启动其他Activity并返回结果等。Activity在Android系统中运行，具有自身的生命周期，因此读者还需要清晰地掌握Activity的生命周期。

本章的另一个重点是掌握开发Fragment的方法，包括为Fragment开发界面、把Fragment添加到Activity、Fragment与Activity通信等内容。Fragment也具有自己的生命周期，因此读者也需要清晰地掌握Fragment的生命周期。

第5章 使用Intent和IntentFilter进行通信

本章要点

理解Intent对于Android应用的作用

使用Intent启动系统组件

Intent的Component属性的作用

Intent的Action属性的作用

Intent的Category属性的作用

为指定Action、Category的Intent配置对应的intent-filter

Intent的Data属性

Intent的Type属性

为指定Data、Type的Intent配置对应的intent-filter

Intent的Extra属性

Intent的Flag属性

使用Intent创建Tab页

前面介绍Activity时已经多次使用了Intent，当一个Activity需要启动另一个Activity时，程序并没有直接告诉系统要启动哪个Activity，而是通过Intent来表达自己的意图：需要启动哪个Activity。Intent的中文翻译就是“意图”的意思。

在这里有读者可能会产生一个疑问，假如甲Activity需要启动另一个Activity，为何不直接使用一个形如startActivity（Class activityClass）的方法呢？这样多么简单、明了啊。但实际上，这种方式虽然简洁，却明显背离了Android的理念，Android使用Intent来封装程序的“调用意图”，不管程序想启动一个Activity也好，想启动一个Service组件也好，想启动一个BroadcastReceiver也好，Android使用统一的Intent对象来封装这种“启动意图”，很明显使用Intent提供了一致的编程模型。

除此之外，使用Intent还有一个好处：在某些时候，应用程序只是想启动具有某种特征的组件，并不想和某个具体的组件耦合，如果使用形如startActivity（Class activityClass）的方法来启动特定组件，势必造成一种硬编码耦合，这样也不利于高层次的解耦。

提示：

如果读者有过Struts 2等MVC框架的编程经验，一定可以很好地理解此处Intent的设计，当Struts 2的Action处理完用户请求之后，它并不会直接返回特定的视图页面，而是返回一个逻辑视图名，开发者可以在配置文件中把该逻辑视图映射到任意的物理视图资源；Android系统的Intent设计有点类似于Struts 2框架中逻辑视图的设计。

总之，Intent封装Android应用程序需要启动某个组件的“意图”。不仅如此，Intent还是应用程序组件之间通信的重要媒介，正如从前面程序中所看到的，两个Activity可以把需要交换的数据封装成Bundle对象，然后使用Intent来携带Bundle对象，这样就实现了两个Activity之间的数据交换。

5.1 Intent对象简述

前面介绍Activity时已经多次使用了Intent，相信读者对Intent对象已经有了一个初步的认识，下面将对Intent对象进行更全面的介绍。

Android的应用程序包含三种重要组件：Activity、Service、BroadcastReceiver，应用程序采用了一致的方式来启动它们—都是依

靠Intent来启动的，Intent就封装了程序想要启动程序的意图。不仅如此，Intent还可用于与被启动组件交换信息。

表5.1显示了使用Intent启动不同组件的方法。

表5.1 使用Intent启动不同组件的方法

续表

上一章我们已经见到了如何使用Intent来启动Activity的示例，至于使用Intent来启动另外两种组件，本书后面章节中会有相关示例，此处暂不深入介绍。

Intent对象大致包含Component、Action、Category、Data、Type、Extra和Flag这7种属性，其中Component用于明确指定需要启动的目标组件，而Extra则用于“携带”需要交换的数据。

下面详细介绍Intent对象各属性的作用。

5.2 Intent的属性及intent-filter配置

Intent代表了Android应用的启动“意图”，Android应用将会根据Intent来启动指定组件，至于到底启动哪个组件，则取决于Intent的各属性。下面将详细介绍Intent的各属性值，以及Android如何根据不同属性值来启动相应的组件。

5.2.1 Component属性

Intent的Component属性需要接受一个ComponentName对象，ComponentName对象包含如下几个构造器。

➢ **ComponentName（String pkg，String cls）**：创建pkg所在包下的cls类所对应的组件。

➢ **ComponentName（Context pkg，String cls）**：创建pkg所对应的包下的cls类所对应的组件。

➢ **ComponentName（Context pkg，Class<? > cls）**：创建pkg所对应的包下的cls类所对应的组件。

上面几个构造器的本质是相同的，这说明创建一个ComponentName需要指定包名和类名一这就可以唯一地确定一个组件类，这样应用程序即可根据给定的组件类去启动特定的组件。

除此之外，Intent还包含了如下三个方法。

➢ **setClass（Context packageContext，Class<? >cls）**：设置该Intent将要启动的组件对应的类。

➢ **setClassName（Context packageContext，String className）**：设置该Intent将要启动的组件对应的类名。

➢ **setClassName（String packageName，String className）**：设置该Intent将要启动的组件对应的类名。

提示：

Android应用的Context代表了访问该应用环境信息的接口，而Android应用的包名则作为应用的唯一标识，因此Android应用的Context对象与该应用的包名有一一对应的关系。上面三个setClass（）方法正是指定组件的包名（分别通过Context指定或String指定）和实现类（分别通过Class指定或通过String指定）。

指定Component属性的Intent已经明确了它将要启动哪个组件，因此这种Intent也被称为显式Intent，没有指定Component属性的Intent被称为隐式Intent一隐式Intent没有明确指定要启动哪个组件，应用将会

根据Intent指定的规则去启动符合条件的组件，但具体是哪个组件则不确定。

提示：

比如一个女孩子有找男朋友的意图，此时有两种方式来表达她的意图：第一种，她明确指出，要找“梁山伯”做男朋友，这种明确指定要找某人的方式被称为显式Intent；第二种，她指出，要找“高”、“富”、“帅”做男朋友，至于到底是谁不重要，只要符合这三个条件即可，这种方式被称为隐式Intent。

下面的示例程序示范了如何通过显式Intent（指定了Component属性）来启动另一个Activity。该程序的界面布局很简单，界面中只有一个按钮，用户单击该按钮将会启动第二个Activity。此处不再给出该程序的界面布局文件。该程序的Java代码如下。

程序清单：
codes\05\5.2\ComponentAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

上面程序中的三行粗体字代码用于创建ComponentName对象，并将该对象设置成Intent对象的Component属性，这样应用程序即可根据该Intent的“意图”去启动指定组件。

实际上，上面三行粗体字代码完全可以简化为如下形式：

从上面的代码可以看出，当需要为Intent设置Component属性时，实际上Intent已经提供了一个简化的构造器，这样方便程序直接指定启动其他组件。

当程序通过Intent的Component属性（明确指定了启动哪个组件）启动特定组件时，被启动组件几乎不需要使用<intent-filter.../>元素进行配置。

程序的SecondActivity也很简单，它的界面布局中只包含一个简单的文本框，用于显示该Activity对应的Intent的Component属性的包名、类名。该Activity的Java代码如下。

程序清单：
codes\05\5.2\ComponentAttr\app\src\main\java\org\crazyit\intent\SecondActivity.java

运行上面的程序，通过第一个Activity中的按钮进入第二个Activity中，将可以看到如图5.1所示的界面。

图5.1 I ntent的Component属性

5.2.2 Action、Category属性与intent-filter配置

Intent的Action、Category属性的值都是一个普通的字符串，其中Action代表该Intent所要完成的一个抽象“动作”，而Category则用于为Action增加额外的附加类别信息。通常Action属性会与Category属性结合使用。

Action要完成的只是一个抽象动作，这个动作具体由哪个组件（或许是Activity，或许是BroadcastReceiver）来完成，Action这个字符串本身并不管。比如Android提供的标准Action：Intent.ACTION_VIEW，它只表示一个抽象的查看操作，但具体查看什么、启动哪个Activity来查看，Intent.ACTION_VIEW并不知道一这取决于Activity的<intent-filter.../>配置，只要某个Activity的<intent-filter.../>配置中包含了该ACTION_VIEW，该Actvitiy就有可能被启动。

提示：

有过Struts 2开发经验的读者都知道，当Struts 2的Action处理用户请求结束后，Action并不会直接指定“跳转”到哪个Servlet（通常是JSP，但JSP的本质就是Servlet），Action的处理方法只是返回一个普通字符串，然后在配置文件中配置该字符串对应到哪个Servlet。Struts 2之所以采用这种设计思路，就是为了把Action与呈现视图的Servlet分离开。类似的，Intent通过指定Action属性（属性值其实就是一个普通字符串），就可以把该Intent与具体的Activity分离，从而提供高层次的解耦。

下面通过一个简单的示例来示范Action属性（就是普通字符串）的作用。下面程序的第一个Activity非常简单，它只包括一个普通按钮，当用户单击该按钮时，程序会“跳转”到第二个Activity—但第一个Activity指定跳转的Intent时，并不以“硬编码”的方式指定要跳转的目标Activity，而是为Intent指定Action属性。此处不给出界面布局的代码，第一个Activity的Java代码如下。

程序清单：
codes\05\5.2\ActionAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

上面程序中的粗体字代码指定了根据Intent来启动Activity—但该Intent并未以“硬编码”的方式指定要启动哪个Activity，相信读者从上面程序中的粗体字代码无法看出该程序将要启动哪个Activity。那么到底程序会启动哪个Activity呢？这取决于Activity配置中<intent-filter.../>元素的配置。

<intent-filter.../>元素是AndroidManifest.xml文件中<activity.../>元素的子元素，前面已经介绍过，<activity.../>元素用于为应用程序配置

Activity，<activity.../>的<intent-filter.../>子元素则用于配置该Activity所能“响应”的Intent。

<intent-filter.../>元素里通常可包含如下子元素。

➢ 0～N个<action.../>子元素。

➢ 0～N个<category.../>子元素。

➢ 0～1个<data.../>子元素。

提示：

<intent-filter.../>元素也可以是<service.../>、<receiver.../>两个元素的子元素，用于表明它们可以响应的Intent。

<action.../>、<category.../>子元素的配置非常简单，它们都可指定android：name属性，该属性的值就是一个普通字符串。

当<activity.../>元素的<intent-filter.../>子元素里包含多个<action.../>子元素（相当于指定了多个字符串）时，就表明该Activity能响应Action属性值为其中任意一个字符串的Intent。

提示：

还是借用前面介绍的女孩子找男朋友的例子，基本上可以把Intent中的Action、Category属性，理解为她对男朋友的要求，每个Intent只能指定一个Action“要求”，但可以指定多个Category“要求”；IntentFilter（使用<intent-filter.../>元素配置）则用于声明该组件（比如Activity、Service、BroadcastReceiver）能满足的要求，每个组件可以声明自己满足多个Action要求、多个Category要求。只要某个组件能满足的要求大于、等于Intent所指定的要求，那么该Intent就能启动该组件。

由于上面的程序指定启动Action属性为ActionAttr.CRAZYIT_ACTION常量（常量值为org.crazyit.intent.action.CRAZYIT_ACTION）的Activity，也就要求被启动的Activity对应的配置元素的<intent-filter.../>元素里至少包括一个如下的<action.../>子元素：

需要指出的是，一个Intent对象最多只能包括一个Action属性，程序可调用Intent的setAction（String str）方法来设置Action属性值；但一个Intent对象可以包括多个Category属性，程序可调用Intent的addCategory （String str）方法来为Intent添加Category属性。当程序创建Intent时，该Intent默认启动Category属性值为Intent.CATEGORY_DEFAULT常量（常量值为android.intent.category.DEFAULT）的组件。

因此，虽然上面程序中的粗体字代码并未指定目标Intent的Category属性，但该Intent已有一个值为android.intent.category.DEFAULT的Category属性值，因此被启动Activity对应的配置元素的<intent-filter.../>元素里至少还包括一个如下的<category.../>子元素：

下面是被启动的Activity的完整配置。

程序清单：
codes\05\5.2\ActionAttr\app\src\main\AndroidManifest.xml

上面Activity配置的三行粗体字代码指定该Activity能响应具有指定Action属性值、指定Category属性值的Intent。其中第二行粗体字代码只是试验用的，对于本程序没有影响—它表明该Activity能响应Action属性值为helloWorld字符串、Category属性值为android.intent.category.DEFAULT的Intent，但我们的程序并未尝试启

动这样的Activity，读者可以自己尝试用这样的Intent来启动Activity，将会看到程序也是启动该Activity。

上面的配置代码中配置了一个实现类为SecondActivity的Activity，因此程序还应该提供这个Activity代码。代码如下。

图5.2 根据Intent的Action启动Activity

上面的程序代码很简单，它只是在启动时把启动该Activity的Intent的Action属性值显示在指定文本框内。运行上面的程序，并单击程序中的“启动指定Action、默认Category对应的Activity”按钮，将看到如图5.2所示的界面。

接下来的示例程序将会示范Category属性的用法。该程序的第一个Activity的代码如下。

程序清单：
codes\05\5.2\ActionCateAttr\app\src\main\java\org\crazyit\intent\MainActivity.java

上面程序中的两行粗体字代码指定了该Intent的Action属性值为org.crazyit.intent.action.CRAZYIT_ACTION字符串，并为该Intent添加了字符串为org.crazyit.intent.category.CRAZYIT_CATEGORY的Category属性。这意味着上面的程序所要启动的目标Activity里应该包含如下<action.../>子元素和<category.../>子元素：

下面是程序要启动的目标Action所对应的配置代码。

程序清单：
codes\05\5.2\ActionCateAttr\app\src\main\AndroidManifest.xml

上面配置Activity时也指定该Activity的实现类为SecondActivity，该实现类的代码如下。

程序清单：
codes\05\5.2\ActionCateAttr\app\src\main\java\org\crazyit\intent\SecondActivity.java

上面的程序也很简单，它只是在启动时把启动该Activity的Intent的Action、Catetory属性值分别显示在不同的文本框内。运行上面的程序，并单击程序中的“启动指定Action、指定Category对应的Activity”按钮，将看到如图5.3所示的界面。

图5.3 根据Intent的Action、Category启动Activity

5.2.3 指定Action、Category调用系统Activity

Intent代表了启动某个程序组件的“意图”，实际上Intent对象不仅可以启动本应用内程序组件，也可以启动Android系统的其他应用的程序组件，包括系统自带的程序组件—只要权限允许。

实际上Android内部提供了大量标准的Action、Catetory常量，其中用于启动Activity的标准的Action常量及对应的字符串如表5.2所示。

表5.2 启动Activity的标准的Action常量及对应的字符串

续表

标准的Category常量及对应的字符串如表5.3所示。

表5.3 标准的Category常量及对应的字符串

表5.2、表5.3所列出的都只是部分较为常用的Action常量、Category常量。关于Intent所提供的全部Action常量、Category常量，应参考Android API文档中关于Intent的说明。

下面将以两个实例来介绍Intent系统Action、系统Category的用法。

实例：查看并获取联系人电话

这个实例将会在程序中提供一个按钮，用户单击该按钮时会显示系统的联系人列表，当用户单击指定联系人之后，程序将会显示该联系人的名字、电话。

这个程序非常有用，比如我们要开发一个发送短信的程序，当用户编写短信完成之后，可能需要浏览联系人列表，并从联系人列表中选出短信接收人，那就可以用到该程序了。

该程序的界面布局代码如下。

程序清单：
codes\05\5.2\SysAction\app\src\main\res\layout\main.xml

上面的界面布局中包含了两个文本框、一个按钮，其中按钮用于浏览系统的联系人列表并选择其中的联系人。两个文本框分别用于显示联系人的名字和电话号码。

注意：

由于该程序会用到系统的联系人应用，因此读者在运行该程序之前应该先进入Android系统自带的Contacts程序，并通过该程序添加几个联系人。

该程序的Java代码如下。

程序清单：
codes\05\5.2\SysAction\app\src\main\java\org\crazyit\action\MainActivity.java

提示：

该实例使用了后面章节中关于ContentProvider的知识，如果读者一时看不懂这个程序，可以参考后面章节的讲解进行理解。

运行上面的程序，单击程序界面中的“查看联系人”按钮，程序将会显示如图5.4所示的界面。

在图5.4所示的联系人列表中单击某个联系人，系统将会自动返回上一个Activity，程序会在上一个Activity中显示所选联系人的名字和电话号码，如图5.5所示。

图5.4 查看联系人

图5.5 获取指定联系人的信息

上面的Intent对象除了设置Action属性之外，还设置了Type属性，关于Intent的Type属性的作用，等下将会进行更详细的介绍。

需要指出的是，由于上面的程序需要查看系统联系人信息，因此不要忘了向该应用的AndroidManifest.xml文件中增加相应的权限，也就是在AndroidManifest.xml文件中增加如下配置：

实例：返回系统Home桌面

本实例将会提供一个按钮，当用户单击该按钮时，系统将会返回Home桌面，就像单击模拟器右边的按钮一样。这也需要通过Intent来实现，程序为Intent设置合适的Action、Category属性，并根据该Intent来启动Activity即可返回Home桌面。该实例程序如下。

程序清单：
codes\05\5.2\ReturnHome\app\src\main\java\org\crazyit\intent\MainActivity.java

上面程序中的粗体字代码设置了Intent的Action属性值为android.intent.action.MAIN字符串、Category属性值为android.intent.category.HOME字符串，满足该Intent的Activity其实就是Android系统的Home桌面。因此，运行上面的程序时单击“返回桌面”按钮，即可返回Home桌面。

5.2.4 Data、Type属性与intent-filter配置

Data属性通常用于向Action属性提供操作的数据。Data属性接受一个Uri对象，一个Uri对象通常通过如下形式的字符串来表示：

Uri字符串总满足如下格式：

例如上面给出的content：//com.android.contacts/contacts/1，其中content是scheme部分，com.android.contacts是host部分，port部分被省略了，/contacts/1是path部分。

Type属性用于指定该Data属性所指定Uri对应的MIME类型，这种MIME类型可以是任何自定义的MIME类型，只要符合abc/xyz格式的字符串即可。

Data属性与Type属性的关系比较微妙，这两个属性会相互覆盖，例如：

➢ 如果为Intent先设置Data属性，后设置Type属性，那么Type属性将会覆盖Data属性。

➢ 如果为Intent先设置Type属性，后设置Data属性，那么Data属性将会覆盖Type属性。

➢ 如果希望Intent既有Data属性，也有Type属性，则应该调用Intent的setDataAndType（）方法。

下面的示例演示了Intent的Data属性与Type属性互相覆盖的情形，该示例的界面布局文件很简单，只是定义了三个按钮，并为三个按钮绑定了事件处理函数。

下面是该示例的Activity代码。

程序清单：
codes\05\5.2\DataTypeOverride\app\src\main\java\org\crazyit\intent\MainActivity.java

上面的三个事件监听方法分别为Intent设置了Data、Type属性，第一个事件监听方法先设置了Type属性，再设置了Data属性，这将导致Data属性覆盖Type属性，单击按钮激发该事件监听方法，将可以看到如图5.6所示的Toast输出。

从图5.6可以看出，此时的Intent只有Data属性，Type属性被覆盖了。

第二个事件监听方法先设置了Data属性，再设置了Type属性，这将导致Type属性覆盖Data属性，单击按钮激发该事件监听方法，将可以看到如图5.7所示的输出。

图5.6 Data属性覆盖Type属性

图5.7 Type属性覆盖Data属性

第三个事件监听方法同时设置了Data、Type属性，这样该Intent中才会同时具有Data、Type属性。

在AndroidManifest.xml文件中为组件声明Data、Type属性都通过<data.../>元素，<data.../>元素的格式如下：

上面的<data.../>元素支持如下属性。

➢ **mimeType**：用于声明该组件所能匹配的Intent的Type属性。

➢ **scheme**：用于声明该组件所能匹配的Intent的Data属性的scheme部分。

➢ **host**：用于声明该组件所能匹配的Intent的Data属性的host部分。

➢ **port**：用于声明该组件所能匹配的Intent的Data属性的port部分。

➢ **path**：用于声明该组件所能匹配的Intent的Data属性的path部分。

➢ **pathPrefix**：用于声明该组件所能匹配的Intent的Data属性的path前缀。

➢ **pathPattern**：用于声明该组件所能匹配的Intent的Data属性的path字符串模板。

Intent的Type属性也用于指定该Intent的要求，对应组件中<intent-filter.../>元素的<data.../>子元素的mimeType属性必须与此相同，才能启动该组件。

Intent的Data属性则略有差异，程序员为Intent指定Data属性时，Data属性的Uri对象实际上可分为scheme、host、port和path部分，此时并不要求被启动组件的<intent-filter.../>中<data.../>子元素的android：scheme、android：host、android：port、android：path完全满足。

Data属性的"匹配"过程则有些差别，它会先检查<intent-filter.../>里的<data.../>子元素，然后：

➢ 如果目标组件的<data.../>子元素只指定了android：scheme属性，那么只要Intent的Data属性的scheme部分与android：scheme属性值相同，即可启动该组件。

➢ 如果目标组件的<data.../>子元素只指定了 android：scheme、android：host 属性，那么只要Intent的Data属性的scheme、host部分与android：scheme、android：host属性值相同，即可启动该组件。

➢ 如果目标组件的<data.../>子元素指定了android：scheme、android：host、android：port属性，那么只要Intent的Data属性的scheme、host、port部分与android：scheme、android：host、android：host属性值相同，即可启动该组件。

提示：

如果<data.../>子元素只有android：port属性，没有指定android：host属性，那么android：port属性将不会起作用。

➢ 如果目标组件的<data.../>子元素只指定了 android：scheme、android：host、android：path属性，那么只要 Intent 的 Data 属性的scheme、host、path 部分与 android：scheme、android：host、android：path属性值相同，即可启动该组件。

提示：

如果<data.../>子元素只有android：path属性，没有指定android：host属性，那么android：path属性将不会起作用。

➢ 如果目标组件的<data.../>子元素指定了 android：scheme、android：host、android：port、android：path属性，那么就要求Intent的Data属性的scheme、host、port、path部分依次与 android：scheme、android：host、android：port、android：path 属性值相同，才可启动该组件。

下面的示例测试了Intent的Data属性与<data.../>元素配置的关系，该示例依次配置了如下5个Activity。

程序清单：
codes\05\5.2\DataTypeAttr\app\src\main\AndroidManifes

t.xml

上面的配置文件中配置了5个Activity，这5个Activity的实现类都非常简单，它们都仅在界面上显示一个TextView，并不显示其他内容。关于这5个Activity的<data.../>子元素配置说明如下。

➢ **第1个Activity**：只要Intent的Data属性的scheme是lee，即可启动该Activity。

➢ **第2个Activity**：只要Intent的Data属性的scheme是lee，且host是www.fkjava.org，port是8888，即可启动该Activity。

➢ **第3个Activity**：只要Intent的Data属性的scheme是lee，且host是www.fkjava.org，path是/mypath，即可启动该Activity。

➢ **第4个Activity**：需要Intent的Data属性的scheme是lee，且host是www.fkjava.org，port是8888，path是/mypath，才可启动该Activity。

➢ **第5个Activity**：需要Intent的Data属性的scheme是lee，且host是www.fkjava.org，port是8888，path是/mypath，type是abc/xyz，才可启动该Activity。

注意：

上面配置Activity的<intent-filter.../>元素时，<action.../>子元素的name属性是随意指定的，这是必需的。如果希望<data.../>子元素能正常起作用，至少要配置一个<action.../>子元素，但该子元素的android：name属性值可以是任意的字符串。

下面是第1个启动Activity的方法。

上面Intent的Data属性，只有scheme为lee，也就是只有第1个Activity符合条件，因此通过该方法启动Activity时，将可以看到启动如图5.8所示的Activity。

图5.8 只有scheme匹配的Activity

下面是第2个启动Activity的方法。

上面Intent的Data属性，scheme为lee，因此第1个Activity符合条件；且该Intent的Data属性的host为www.fkjava.org，port为8888，因此第2个Activity也符合条件。通过该方法启动Activity时，将可以看到启动如图5.9所示的选择Activity界面。

下面是第3个启动Activity的方法。

上面Intent的Data属性，scheme为lee，因此第1个Activity符合条件；且该Intent的Data属性的host为www.fkjava.org，path为/mypath，因此第3个Activity也符合条件。通过该方法启动Activity时，将可以看到启动如图5.10所示的选择Activity界面。

图5.9 scheme、host、port匹配的Activity

图5.10 scheme、host、path匹配的Activity

下面是第4个启动Activity的方法。

上面Intent的Data属性，scheme为lee，因此第1个Activity符合条件；且该Intent的Data属性的host为www.fkjava.org，port为8888，因此第2个Activity符合条件；该Intent的Data属性的host为www.fkjava.org，path为/mypath，因此第3个Activity符合条件；该Intent的Data属性的host为www.fkjava.org，port为8888，path为/mypath，因此第4个Activity也符合条件。通过该方法启动Activity时，将可以看到启动如图5.11所示的选择Activity界面。

图5.11 scheme、host、port、path匹配的Activity

下面是第5个启动Activity的方法。

上面的Intent 不仅指定了Data 属性，也指定了Type 属性，此时符合条件的只有第5 个Activity。通过该方法启动Activity时，将可以看到启动如图5.12所示的Activity。

图5.12 scheme、host、port、path、type匹配的Activity

实例：使用Action、Data属性启动系统Activity

一旦为Intent同时指定了Action、Data属性，Android就可根据指定的数据类型来启动特定的应用程序，并对指定数据执行相应的操作。

下面是几个Action属性、Data属性的组合。

➢ **ACTION_VIEW content：//com.android.contacts/contacts/1**：显示标识为1的联系人的信息。

➢ **ACTION_EDIT content：//com.android.contacts/contacts/1**：编辑标识为1的联系人的信息。

➢ **ACTION_DIAL content：//com.android.contacts/contacts/1**：显示向标识为1的联系人拨号的界面。

➢ **ACTION_VIEW tel：123**：显示向指定号码123拨号的界面。

➢ **ACTION_DIAL tel：123**：显示向指定号码123拨号的界面。

➢ **ACTION_VIEW content：//contacts/people/**：显示所有联系人列表的信息，通过这种组合可以非常方便地查看系统联系人。

本实例程序示范通过同时为Intent指定Action、Data属性来启动特定程序并操作相应的数据。下面程序的界面很简单，只包含3个按钮，其中一个按钮用于浏览指定网页；一个按钮用于编辑指定联系人信息；另一个按钮用于呼叫指定号码。

程序清单：
codes\05\5.2\ActionData\app\src\main\java\org\crazyit\intent\MainActivity.java

运行上面的程序，单击第一个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_VIEW，Data=http：//www.crazyit.org）对应的Activity，将看到打开www.crazyit.org的界面，如图5.13所示。

单击第二个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_EDIT，Data=content：//com.android.contacts/contacts/1）对应的Activity，将看到编辑标识为1的联系人界面，如图5.14所示。

单击第三个按钮，该按钮单击时启动Intent（Action=Intent.ACTION_DIAL，Data=tel：13800138000）对应的Activity，将看到程序向13800138000拨号的界面，如图5.15所示。

图5.13 使用Action、Data打开指定网页

图5.14 使用Action、Data属性编辑指定联系人

图5.15 使用Action、Data属性向指定号码拨号

5.2.5 Extra属性

Intent的Extra属性通常用于在多个Action之间进行数据交换，Intent的Extra属性值应该是一个Bundle对象，Bundle对象就像一个Map对象，它可以存入多个key-value对，这样就可以通过Intent在不同Activity之间进行数据交换了。关于Extra属性的用法前面已有示例，此处不再赘述。

5.2.6 Flag属性

Intent的Flag属性用于为该Intent添加一些额外的控制旗标，Intent可调用addFlags（）方法来添加控制旗标。

提示：

前面介绍启用ActionBar的程序图标返回主Activity时已经用到了Flag属性。比如前面介绍的Intent.FLAG_ACTIVITY_CLEAR_TOP旗标可用于清除当前Activity栈中的Activity。

除此之外，Intent还包含了如下常用的Flag旗标。

➢ **FLAG_ACTIVITY_BROUGHT_TO_FRONT**：如果通过该Flag启动的Activity已经存在，下次再次启动时，将只是把该Activity带到前台。例如，现在Activity栈中有Activity A，此时以该旗标启动Activity B（即Activity B 是以FLAG_ACTIVITY_BROUGHT_TO_FRONT旗标启动的），然后在Activity B中启动Activity C、D，如果此时在Activity D中再启动Activity B，将直接把Activity栈中的Activity B 带到前台。此时Activity栈中情形是Activity A、C、D、B。

➢ **FLAG_ACTIVITY_CLEAR_TOP**：该Flag相当于加载模式中的singleTask，通过这种Flag启动的Activity将会把要启动的Activity之上的Activity全部弹出Activity栈。例如，Activity栈中包含A、B、C、D四个Activity，如果采用该Flag从Activity D跳转到Activity B，那么此时Activity栈中只包含A、B两个Activity。

➢ **FLAG_ACTIVITY_NEW_TASK**：默认的启动旗标，该旗标控制重新创建一个新的Activity。

➢ **FLAG_ACTIVITY_NO_ANIMATION**：该旗标控制启动Activity时不使用过渡动画。

➢ **FLAG_ACTIVITY_NO_HISTORY**：该旗标控制被启动的Activity将不会保留在Activity栈中。例如，Activity栈中原来有A、B、C三个Activity，此时在Activity C中以该Flag启动Activity D，Activity D再启动Activity E，此时Activity栈中只有A、B、C、E四个Activity，Activity D不会保留在Actvity栈中。

➢ **FLAG_ACTIVITY_REORDER_TO_FRONT**：该Flag控制如果当前已有Activity，则直接将该Activity带到前台。例如，现在Activity栈中

有A、B、C、D四个Activity，如果使用FLAG_ACTIVITY_REORDER_TO_FRONT旗标来启动Activity B，那么启动后的Activity栈中情形为A、C、D、B。

➢ **FLAG_ACTIVITY_SINGLE_TOP**：该Flag相当于加载模式中的singleTop模式。例如，原来Activity栈中有A、B、C、D四个Activity，在Activity D中再次启动Activity D，Activity栈中依然还是A、B、C、D四个Activity。

Android为Intent提供了大量的Flag，每个Flag都有其特定的功能，具体请参考关于Intent的API文档。

5.3 使用Intent创建Tab页

前面已经介绍了如何使用TabActivity来创建Activity布局，前面添加Tab页时使用了TabHost.TabSpec提供的如下方法。

➢ **setContent（int viewId）**：直接将指定View组件设置成Tab页的Content。

实际上TabHost.TabSpec还提供了如下方法。

➢ **setContent（Intent intent）**：直接将指定Intent对应的Activity设置成Tab页的Content。

例如如下Activity代码。

程序清单：
codes\05\5.3\IntentTab\app\src\main\java\org\crazyit\intent\MainActivity.java

上面程序中的三行粗体字代码用于为TabHost.TabSpec设置Content，三行粗体字代码分别传入了三个Intent对象，这意味着该TabHost将直

接以三个Activity类作为Tab页。上面的程序还需要BeCalledActivity、CalledActivity、NoCallActivity 三个Activity类，不过这三个Activity都很简单，故此处不再给出。运行上面的程序，将看到如图5.16所示的界面。

图5.16 使用Intent创建Tab页

5.4 本章小结

本章主要介绍了Android系统中Intent的功能和用法，当Android应用需要启动某个组件时，总需要借助于Intent来实现。不管是启动Activity，还是启动Service、BroadcastReceiver组件，Android系统都是由Intent来实现的。简单地说，Android使用Intent封装了应用程序的“启动意图”，但这种“意图”并未直接与任何程序组件耦合，通过这种方式即可很好地提高系统的可扩展性和可维护性。学习本章需要重点掌握Intent的Component、Action、Category、Data、Type各属性的功能和用法，并掌握如何在AndroidManifest.xml文件中配置<intent-filter.../>元素。

第6章 Android应用的资源

本章要点

Android应用的资源和作用

Android应用的资源的存储方式

在XML布局文件中使用资源

在Java程序中使用资源

使用字符串资源

使用颜色资源

使用尺寸资源

使用数组资源

使用图片资源

使用各种Drawable资源

使用原始XML资源

使用布局资源

使用菜单资源

使用样式和主题资源

使用属性资源

使用原始资源

使用资源进行程序国际化

自适应不同屏幕的资源

经过前面的介绍，相信读者对Android应用已有了大致的了解。如果从物理存在形式来分，Android应用的源代码大致可分为如下三大类。

➢ **界面布局文件：**XML文件，文件中每个标签都对应于相应的View标签。

➢ **Java源文件：**应用中的Activity、Service、BroadcastReceiver、ContentProvider四大组件都是采用Java代码来实现的。

➢ **资源文件：**主要以各种XML文件为主，还可包括*.png、*.jpg、*.gif图片资源。

在传统Java应用中，初学者很容易犯一个错误：直接在Java源代码中使用如"crazyit.org"、"hello"这样的字符串，或直接使用123、0.9这样的数值，而且不添加任何注释。过了一段时间后，即使自己再去看原来写的程序代码，一时之间，也无法理解其中"crazyit.org"、"hello"字符串，123、0.9等数值的含义，这种方式就大大增加了程序的维护成本。这种直接在代码中定义的123、0.9等数值，也被称为“魔术数值”（就像表演魔术一样，其他人都搞不懂）。

为了改善这种情况，有经验的开发者会专门定义一个或多个接口或类，然后在其中以常量的形式来定义程序中用到的所有字符串、数值等，这些常量的名称十分明确，如ResultSet.TYPE_FORWARD_ONLY，相信有经验的读者一看到这个常量就大致明白了它的含义。这样的方式就可以很好地提高程序的可维护性。

使用接口或类的形式来定义程序中用到的字符串、数值，虽然已经部分提高了程序的解耦，但后期维护、进一步开发时，开发人员还得去

Java“代码海”中打捞那些定义字符串常量、数值常量的位置，因此还有可以提高的地方。

Android应用则对这种字符串常量、数值常量的定义做了进一步改进：Android允许把应用中用到的各种资源，如字符串资源、颜色资源、数组资源、菜单资源等都集中放到res目录中定义，应用程序则直接使用这些资源中定义的值。

Android应用中除了res目录用于存放资源之外，assets目录也用于存放资源。一般来说，assets目录下存放的资源代表应用无法直接访问的原生资源，应用程序需要通过AssetManager以二进制流的形式来读取资源。而res目录下的资源，Android SDK会在编译该应用时，自动在R.java文件中为这些资源创建索引，程序可直接通过R资源清单类进行访问。

我们前面介绍的很多示例都是直接将字符串值写在界面布局文件中或Java程序代码中，实际上那并不是一种好的方式。只是前面还未详细介绍Android应用的资源，为了避免读者产生畏难心理，并未使用资源文件而已。

6.1 应用资源概述

Android应用资源可分为两大类。

➢ 无法通过R资源清单类访问的原生资源，保存在assets目录下。

➢ 可通过R资源清单类访问的资源，保存在res目录下。

大部分时候提到Android应用资源时，往往都是指位于/res/目录下的应用资源，Android SDK会在编译该应用时在R类中为它们创建对应的索引项。

6.1.1 资源的类型以及存储方式

Android要求在res目录下用不同的子目录来保存不同的应用资源，表6.1大致显示了Android不同资源在/res/目录下的存储方式。

表6.1 Android应用资源的存储

提示：

对于Android 5.0版本而言，应用的/res/目录下不仅包含了drawable子目录，而且还提供了drawable-ldpi（低分辨率）、drawable-mdpi（中等分辨率）、drawable-hdpi（高分辨率）、drawable-xhdpi（超高分辨率）4个子目录，其实这4个子目录的作用就相当于drawable子目录。drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi四个子目录下存放的图片文件的文件名完全相同，只是分辨率不同一这种做法可以让系统根据屏幕分辨率来选择相应的图片。大部分程序、系统将会选择drawable-mdpi目录下的图片文件，如果系统使用高分辨率屏幕，那么系统将会选择drawable-hdpi目录下的图片文件。

6.1.2 使用资源

在Android应用中使用资源可分为在Java代码和XML文件中使用资源，其中Java代码用于为Android应用定义四大组件，而XML文件则用于为Android应用定义各种资源。

1.在Java代码中使用资源清单项

由于Android SDK会在编译应用时在R类中为/res/目录下所有资源创建索引项，因此在Java代码中访问资源主要通过R类来完成。其完整的语法格式为：

上面语法格式中各成分的说明如下。

➢ **<package_name>**：指定R类所在包，实际上就是使用全限定类名。当然，如果在Java程序中导入R类所在包，就可以省略包名。

➢ **<resource_type>**：R类中代表不同资源类型的子类，例如string代表字符串资源。

➢ **<resource_name>**：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是XML资源元素中由android:name属性所指定的名称。

例如如下代码片段：

2.在Java代码中访问实际资源

R资源清单类为所有的资源都定义了一个资源清单项，但这个清单项只是一个int类型的值，并不是实际的资源对象。在大部分情况下，Android应用的API允许直接使用int类型的资源清单项代替应用资源。

但有些时候，程序也需要使用实际的Android资源，为了通过资源清单项来获取实际资源，可以借助于Android提供的Resources类。

提示：

笔者把Resources类称为“Android资源访问总管家”，Resources类提供了大量的方法来根据资源清单ID获取实际资源。

Resources主要提供了如下两类方法。

➢ **getXxx（int id）**：根据资源清单ID来获取实际资源。

➢ **getAssets（）**：获取访问/assets/目录下资源的AssetManager对象。

Resources由Context调用getResources（）方法来获取。

下面的代码片段示范了如何通过Resources获取实际字符串资源。

3.在XML文件中使用资源

当定义XML资源文件时，其中的XML元素可能需要指定不同的值，这些值就可设置为已定义的资源项。在XML代码中使用资源的完整语法格式为：

上面语法格式中各成分的说明如下。

➢ **<package_name>**：指定资源类所在应用的包。如果所引用的资源和当前资源位于同一个包下，则<package_name>可以省略。

➢ **<resource_type>**：R类中代表不同资源类型的子类。

➢ **<resource_name>**：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是XML资源元素中由android:name属性所指定的名称。

例如如下代码片段在一份文件中定义了两种资源：

接下来与它位于同一包中的XML资源文件就可通过如下方式来使用资源：

下面将会对各种资源分别进行详细阐述。

6.2 字符串、颜色、尺寸资源

字符串资源、颜色资源、尺寸资源，它们对应的XML文件都将位于/res/values/目录下，它们默认的文件名以及在R类中对应的内部类如表6.2所示。

表6.2 字符串、颜色、尺寸资源表

下面会通过示例来介绍字符串资源、颜色资源和尺寸资源的用法。

6.2.1 颜色值的定义

Android中的颜色值是通过红（Red）、绿（Green）、蓝（Blue）三原色以及一个透明度（Alpha）值来表示的，颜色值总是以井号（#）开头，接下来就是Alpha-Red-Green-Blue的形式。其中Alpha值可以省略，如果省略了Alpha值，那么该颜色默认是完全不透明的。

Android颜色值支持常见的4种形式。

➢ **#RGB**：分别指定红、绿、蓝三原色的值（只支持0～f这16级颜色）来代表颜色。

➢ **#ARGB**：分别指定红、绿、蓝三原色的值（只支持0～f这16级颜色）及透明度（只支持0～f这16级透明度）来代表颜色。

➢ **#RRGGBB**：分别指定红、绿、蓝三原色的值（支持00～ff这256级颜色）来代表颜色。

➢ **#AARRGGBB**：分别指定红、绿、蓝三原色的值（支持00～ff这256级颜色）以及透明度（支持00～ff这256级透明度）来代表颜色。

在上面4种形式中，A、R、G、B都代表一个十六进制的数，其中A代表透明度，R代表红色数值，G代表绿色数值，B代表蓝色数值。

提示：

关于颜色与三原色的知识，请读者自行参考光学方面的知识。此处笔者粗略介绍一下三原色理论：白色光大致可“分解”为红、绿、蓝三种光，红、绿、蓝三种光可合并成白色光。当红、绿、蓝都是最大值时，三种光合并就会变成白色光；当三种光的值相等，但不是最大值时，三种光将会合并成灰色光；如果其中一种光或两种光的值更亮，那么三种光合并就会产生彩色的光。

6.2.2 定义字符串、颜色、尺寸资源文件

字符串资源文件位于/res/values/目录下，字符串资源文件的根元素是<resources...>，该元素里每个<string.../>子元素定义一个字符串常量，其中<string.../>元素的name属性指定该常量的名称，<string.../>元素开始标签和结束标签之间的内容代表字符串值，如以下代码所示：

如下文件是该示例的字符串资源文件。

程序清单：
codes\06\6.2\ValuesResTest\app\src\main\res\values\strings.xml

上面的程序代码中每个<string.../>元素定义一个字符串，其中<string.../>元素的name属性定义字符串的名称，<string>与</string>中间的内容就是该字符串的值。

颜色资源文件位于/res/values/目录下，颜色资源文件的根元素是<resources...>，该元素里每个<color.../>子元素定义一个字符串常量，其中<color.../>元素的name属性指定该颜色的名称，<color.../>元素开始标签和结束标签之间的内容代表颜色值，如以下代码所示：

如下文件是该示例的颜色资源文件。

程序清单：
codes\06\6.2\ValuesResTest\app\src\main\res\values\colors.xml

上面的程序代码中每个<color.../>元素定义一个字符串，其中<color.../>元素的name属性定义颜色的名称，<color>与</color>中间的内容就是该颜色的值。

尺寸资源文件位于/res/values/目录下，尺寸资源文件的根元素是<resources...>，该元素里每个<dimen.../>子元素定义一个尺寸常量，其中<dimen.../>元素的name属性指定该尺寸的名称，<dimen.../>元素开始标签和结束标签之间的内容代表尺寸值，如以下代码所示。

程序清单：
codes\06\6.2\ValuesResTest\app\src\main\res\values\dimens.xml

上面三份资源文件分别定义了字符串、颜色、尺寸资源，应用程序接下来既可在XML文件中使用这些资源，也可在Java代码中使用这些资源。

6.2.3 使用字符串、颜色、尺寸资源

正如前面介绍的，在XML文件中使用资源按如下语法格式：

下面程序的界面布局中大量使用了前面定义的资源。

程序清单：
codes\06\6.2\ValuesResTest\app\src\main\res\layout\main.xml

上面程序中的粗体字代码就是使用字符串资源、尺寸资源的代码。

在Java代码中使用资源按如下语法格式：

下面的Java程序同时使用了上面定义的三种资源。

程序清单：
codes\06\6.2\ValuesResTest\app\src\main\java\org\crazyit\res\MainActivity.java

上面程序中的粗体字代码分别使用了前面定义的字符串资源、颜色资源和尺寸资源。运行上面的程序，将可以看到如图6.1所示的界面。

与定义字符串资源类似的是，Android也允许使用资源文件来定义boolean常量。例如，在/res/values/目录下增加一个bools.xml文件，该文件的根元素也是<resources.../>，根元素内通过<bool.../>子元素来定义boolean常量，示例如下：

图6.1 使用字符串、颜色、尺寸资源

一旦在资源文件中定义了如上所示的资源文件之后，接下来在Java代码中按如下语法格式访问即可：

在XML文件中按如下格式即可访问资源：

例如，为了在Java代码中获取指定boolean变量的值，可通过如下代码来实现：

与定义字符串资源类似的是，Android也允许使用资源文件来定义整型常量。例如，在/res/values/目录下增加一个integers.xml文件（文件名可以自由选择），该文件的根元素也是<resources.../>，根元素内通过<integer.../>子元素来定义整型常量，示例如下：

一旦在资源文件中定义了如上所示的资源文件之后，接下来在Java代码中按如下语法格式访问即可：

在XML文件中按如下格式即可访问资源：

例如，为了在Java代码中获取指定整型变量的值，可通过如下代码来实现：

6.3 数组（Array）资源

上面的程序在Java代码中定义了两个数组，Android并不推荐在Java代码中定义数组，因为Android允许通过资源文件来定义数组资源。

Android采用位于/res/values目录下的arrays.xml文件来定义数组资源，定义数组时XML资源文件的根元素也是<resources.../>，该元素内可包含如下三种子元素。

➢ **<array..../>子元素**：定义普通类型的数组，例如Drawable数组。

➢ **<string-array..../>子元素**：定义字符串数组。

➢ **<integer-array..../>子元素**：定义整型数组。

一旦在资源文件中定义了数组资源之后，接下来就可以在Java程序中通过如下形式来访问资源了：

在XML文件中则可通过如下形式进行访问：

为了能在Java程序中访问到实际数组，Resources提供了如下方法。

➢ **String[] getStringArray（int id）**：根据资源文件中字符串数组资源的名称来获取实际的字符串数组。

➢ **int[] getIntArray（int id）**：根据资源文件中整型数组资源的名称来获取实际的整型数组。

➢ **TypedArray obtainTypedArray（int id）**：根据资源文件中普通数组资源的名称来获取实际的普通数组。

TypedArray代表一个通用类型的数组，该类提供了getXxx（int index）方法来获取指定索引处的数组元素。

下面为该应用程序增加如下数组资源文件。

程序清单：
codes\06\6.3\ArrayResTest\app\src\main\res\values\arrays.xml

定义了上面的数组资源之后，既可在XML文件中使用这些数组资源，也可在Java程序中使用这些数组资源。例如，如下界面布局文件中定义了一个ListView数组，并将android：entries属性值指定为一个数组。界面布局文件代码如下。

程序清单：
codes\06\6.3\ArrayResTest\app\src\main\res\layout\main.xml

接下来程序中无须定义数组，程序直接使用资源文件中定义的数组。程序代码如下。

程序清单：
codes\06\6.3\ArrayResTest\app\src\main\java\org\crazyit\res\MainActivity.java

上面程序中的粗体字代码就是使用数组资源的关键代码。运行上面的程序，将看到如图6.2所示的结果。

6.4 使用Drawable资源

Drawable资源是Android应用中使用最广泛的资源，也是Android应用中最灵活的资源，它不仅可以直接使用*.png、*.jpg、*.gif、*.9.png

等图片作为资源，也可以使用多种XML文件作为资源。只要一份XML文件可以被系统编译成Drawable子类的对象，那么这份XML文件即可作为Drawable资源。

图6.2 使用数组资源

Drawable资源通常保存在/res/drawable目录下，实际上可能保存在/res/drawable-ldpi、/res/drawable-mdpi、/res/drawable-hdpi、/res/drawable-xhdpi目录下。

下面详细介绍各种Drawable资源。

6.4.1 图片资源

图片资源是最简单的Drawable资源，只要把*.png、*.jpg、*.gif等格式的图片放入/res/drawable-xxx目录下，Android SDK就会在编译应用中自动加载该图片，并在R资源清单类中生成该资源的索引。

注意：

Android要求图片资源的文件名必须符合Java标识符的命名规则，否则Android SDK无法为该图片在R类中生成资源索引。

一旦系统在R资源清单类中生成了指定资源的索引，接下来就可以在Java类中使用如下语法格式来访问该资源：

在XML代码中则按如下语法格式来访问该资源：

为了在程序中获得实际的Drawable对象，Resources提供了Drawable getDrawable（int id）方法，该方法即可根据Drawable资源在R资源清

单类中的ID来获取实际的Drawable对象。

关于图片资源的示例，本书前面已有大量介绍，此处不再赘述。

6.4.2 StateListDrawable资源

StateListDrawable用于组织多个Drawable对象。当使用StateListDrawable作为目标组件的背景、前景图片时，StateListDrawable对象所显示的Drawable对象会随目标组件状态的改变而自动切换。

定义StateListDrawable对象的XML文件的根元素为<selector.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

➢ **android：color或android：drawable：**指定颜色或Drawable对象。

➢ **android：state_xxx：**指定一个特定状态。

例如如下语法格式：

StateListDrawable的<item.../>元素所支持的状态有如表6.3所示的几种。

表6.3 StateListDrawable的<item.../>元素所支持的状态

实例：高亮显示正在输入的文本框

通过前面介绍我们知道，使用EditText时可指定一个android：textColor属性，该属性用于指定文本框的文字颜色。前面介绍该属性时总是直接给它一个颜色值，因此该文本框的文字颜色总是固定的。

借助于StateListDrawable对象，可以让文本框中的文字颜色随文本框的状态动态改变。

为本应用提供如下Drawable资源文件。

程序清单：
codes\06\6.4\StateListDrawableTest\app\src\main\res\drawable\my_image.xml

上面的资源文件中指定了目标组件得到焦点、失去焦点时使用不同的颜色，接下来可以在定义EditText时使用该资源。

程序清单：
codes\06\6.4\StateListDrawableTest\app\src\main\res\layout\main.xml

图6.3 输入文本时高亮显示

该应用的Java程序代码不需要任何修改，只要显示该界面布局即可。运行该程序，将看到如图6.3所示的界面。

通过使用StateListDrawable不仅可以让文本框里文字的颜色随文本框状态的改变而切换，也可让按钮的背景图片随按钮状态的改变而切换，实际上StateListDrawable的功能非常灵活，它可以让各种组件的背景、前景随状态的改变而切换。

6.4.3 LayerDrawable资源

与StateListDrawable有点类似，LayerDrawable也可包含一个Drawable数组，因此系统将会按这些Drawable对象的数组顺序来绘制它们，索引最大的Drawable对象将会被绘制在最上面。

定义LayerDrawable对象的XML文件的根元素为<layer-list.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

➢ **android：drawable**：指定作为LayerDrawable元素之一的Drawable对象。

➢ **android：id**：为该Drawable对象指定一个标识。

➢ **android：buttom|top|left|button**：它们用于指定一个长度值，用于指定将该Drawable对象绘制到目标组件的指定位置。

例如如下语法格式：

实例：定制拖动条的外观

通过前面介绍我们知道，使用SeekBar时可指定一个android：progressDrawable属性，该属性可改变SeekBar的外观，借助于LayerDrawable即可改变SeekBar的规定、已完成部分的Drawable对象。

例如定义如下Drawable资源。

程序清单：
codes\06\6.4\LayerDrawableTest\app\src\main\res\drawable\my_bar.xml

除此之外，该示例还定义了如下LayerDrawable对象。

程序清单：
codes\06\6.4\LayerDrawableTest\app\src\main\res\drawable\layout_logo.xml

上面的程序中定义了三个“层叠”在一起的Drawable对象，接着在界面布局中使用上面的my_bar.xml定义的Drawable对象来改变SeekBar的外观，并通过ImageView来显示上面layout_logo的Drawable组件。界面布局的代码片段如下。

程序清单：
codes\06\6.4\LayerDrawableTest\app\src\main\res\layout\main.xml

该程序的代码无须任何改变，直接加载、显示上面的界面布局文件即可，运行该程序出现如图6.4所示的界面。

6.4.4 ShapeDrawable资源

图6.4 使用LayerDrawable资源

ShapeDrawable用于定义一个基本的几何图形（如矩形、圆形、线条等），定义ShapeDrawable的XML文件的根元素是<shape.../>元素，该元素可指定如下属性。

➢ **android：shape=["rectangle"|"oval"|"line"|"ring"]**：指定定义哪种类型的几何图形。

定义ShapeDrawable对象的完整语法格式如下：

下面通过实例来介绍ShapeDrawable资源的定义和使用。

实例：椭圆形、渐变背景的文本框

前面介绍TextView时知道该组件可指定一个android：background属性，该属性用于为该文本框指定背景。大部分时候，文本框的背景只是一个简单的图片，或者只是一个简单的颜色。

如果程序使用ShapeDrawable资源作为文本框的android：background属性，则可以在Android应用中做出各种外观的文本框。下面先定义如下的ShapeDrawable资源。

程序清单：
codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_1.xml

再定义如下的ShapeDrawable资源。

程序清单：
codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_2.xml

接下来定义如下的ShapeDrawable资源。

程序清单：
codes\06\6.4\ShapeDrawableTest\app\src\main\res\drawable\my_shape_3.xml

定义了上面三个ShapeDrawable资源之后，接下来在界面布局文件中用这三个ShapeDrawable资源作为文本框的背景。界面布局文件代码如下。

程序清单：
codes\06\6.4\ShapeDrawableTest\app\src\main\res\layout\main.xml

使用Activity加载、显示上面的界面布局文件，将可以看到如图6.5所示的界面。

6.4.5 ClipDrawable资源

图6.5 使用ShapeDrawable资源

ClipDrawable代表从其他位图上截取的一个“图片片段”。在XML文件中定义ClipDrawable对象使用<clip.../>元素，该元素的语法为：

上面的语法格式中可指定如下三个属性。

➢ **android：drawable**：指定截取的源Drawable对象。

➢ **android：clipOrientation**：指定截取方向，可设置水平截取或垂直截取。

➢ **android：gravity**：指定截取时的对齐方式。

使用ClipDrawable对象时可调用setLevel（int level）方法来设置截取的区域大小，当level为0时，截取的图片片段为空；当level为10000时，截取整张图片。

下面以一个实例来说明ClipDrawable对象的用法。

实例：徐徐展开的风景

因为ClipDrawable对象可调用setLevel（int level）控制截取图片的部分，因此本实例只要设置一个定时器，让程序不断调用ClipDrawable的setLevel（int level）方法即可实现图片徐徐展开的效果。

程序先定义如下ClipDrawable对象。

程序清单：
codes\06\6.4\ClipDrawableTest\app\src\main\res\drawable\my_clip.xml

上面的程序控制从中间开始截取图片，截取方向为水平截取。接下来程序将通过一个定时器来定期修改ClipDrawable对象的level，即可实现图片徐徐张开的效果。

程序清单：
codes\06\6.4\ClipDrawableTest\app\src\main\java\org\crazyit\res\MainActivity.java

运行上面的程序，将看到如图6.6所示的结果。

图6.6 使用ClipDrawable对象

从图6.6所示的运行结果可以看出，通过使用这种徐徐展开的图片，用户会感觉就像进度条一样一实际上，在实际应用中完全可以使用这种ClipDrawable对象来实现图片进度条。

6.4.6 AnimationDrawable资源

AnimationDrawable代表一个动画，关于Android动画的知识本书后面还有更详细的介绍，本节只是先介绍一下如何定义AnimationDrawable

资源。Android既支持传统的逐帧动画（类似于电影方式，一张图片、一张图片地切换），也支持通过平移、变换计算出来的补间动画。

下面以补间动画为例来介绍如何定义AnimationDrawable资源。定义补间动画的XML资源文件以<set.../>元素作为根元素，该元素内可以指定如下4个元素。

➢ **alpha**：设置透明度的改变。

➢ **scale**：设置图片进行缩放变换。

➢ **translate**：设置图片进行位移变换。

➢ **rotate**：设置图片进行旋转。

定义动画的XML资源应该放在/res/anmi/路径下，当使用Android Studio创建一个Android应用时，默认不会包含该路径，开发者需要自行创建该路径。

定义补间动画的思路很简单：设置一张图片的开始状态（包括透明度、位置、缩放比、旋转度），并设置该图片的结束状态（包括透明度、位置、缩放比、旋转度），再设置动画的持续时间，Android系统会使用动画效果把这张图片从开始状态变换到结束状态。

设置补间动画的语法格式如下：

上面语法格式中包含了大量的fromXxx、toXxx属性，这些属性就用于定义图片的开始状态、结束状态。除此之外，当进行缩放变换（scale）、旋转（rotate）变换时，还需要指定pivotX、pivotY两个属性，这两个属性用于指定变换的“中心点”—比如进行旋转变换时，需要指定“旋轴点”；进行缩放变换时，需要指定“中心点”。

除此之外，上面<set.../>、<alpha.../>、<scale.../>、<translate.../>、<rotate.../>都可指定一个android：interpolator属性，该属性指定动画的变化速度，可以实现匀速、正加速、负加速、无规则变加速等，Android系统的R.anim类中包含了大量常量，它们定义了不同的动画速度，例如：

➢ **linear_interpolator**：匀速变换。

➢ **accelerate_interpolator**：加速变换。

➢ **decelerate_interpolator**：减速变换。

如果程序想让<set.../>元素下所有的变换效果使用相同的动画速度，则可指定android：shareInterpolator="true"。

例如，下面的资源文件定义了一个动画资源。

程序清单：
codes\06\6.4\AnimationDrawable\app\src\main\res\anim\my_anim.xml

上面的动画资源文件十分简单，它只指定了图片资源需要进行两种变换：缩放变换和位移变换。

一旦定义了上面的动画资源文件，接下来就可以在XML文件中按如下语法格式来访问它：

在Java代码中则按如下语法格式来访问：

为了在Java代码中获取实际的Animation对象，可调用AnimationUtils的如下方法：

➢ **loadAnimation(Context ctx,int resId)**

下面的程序示范了如何使用AnimationDrawable资源。

程序清单：
codes\06\6.4\AnimationDrawable\app\src\main\java\org\crazyit\res\MainActivity.java

图6.7 使用AnimationDrawable资源

上面的程序使用了界面布局中的两个组件：一个ImageView、一个Button，这是两个最普通的组件，故此处不再给出界面布局代码。运行上面的程序，将看到如图6.7所示的动画效果。

上面的程序中①号代码设置动画结束后保留图片的变换结果。本来Android的API文档中说明可以在<alpha.../>、<scale.../>、<translate.../>、<rotate.../>等元素中指定android：fillAfter为true来实现这个效果，但实际上要为<set.../>设置android：fillAfter为true才可以。

6.5 属性动画（Property Animation）资源

Animator代表一个属性动画，但它只是一个抽象类，通常会使用它的子类：AnimatorSet、ValueAnimator、ObjectAnimator、TimeAnimator。关于Android动画的知识，本书后面还有更详细的介绍，本节只是先介绍一下如何定义属性动画资源。

定义属性动画的XML资源文件能以如下三个元素中的任意一个作为根元素。

➢ **<set.../>**：它是一个父元素，用于包含<objectAnimator.../>、<animator.../>或<set.../>子元素，该元素定义的资源代表AnimatorSet对象。

➢ **<objectAnimator.../>**：用于定义ObjectAnimator动画。

➢ **<animator.../>**：用于定义ValueAnimator动画。

定义属性动画的语法格式如下：

实例：不断渐变的背景色

该实例将会使用属性动画来控制组件背景色不断渐变。该实例所使用的属性动画资源文件如下。

程序清单：
codes\06\6.5\AnimatorTest\app\src\main\res\animator\color_anim.xml

上面的代码定义了一个ObjectAnimator对象，接下来程序就可以通过属性动画来控制指定组件的背景色不断改变。下面是该实例的Activity代码。

程序清单：
codes\06\6.5\AnimatorTest\app\src\main\java\org\crazyit\res\MainActivity.java

上面程序中的粗体字代码使用 AnimatorInflater 工具类加载了指定动画资源文件，并将该动画资源文件转换为ObjectAnimator对象。接下来程序对MyAnimationView本身应用该动画，将可以看到该组件的背景色不断变化，如图6.8所示。

6.6 使用原始XML资源

图6.8 不断渐变的背景色

在某些时候，Android应用有一些初始化的配置信息、应用相关的数据资源需要保存，一般推荐使用XML文件来保存它们，这种资源就被称为原始XML资源。下面介绍如何定义、获取原始XML资源。

6.6.1 定义原始XML资源

原始XML资源一般保存在/res/xml/路径下一当使用Android Studio创建Android应用时，/res/目录下并没有包含xml子目录，开发者应该自行手动创建xml子目录。

接下来Android应用对原始XML资源没有任何特殊的要求，只要它是一份格式良好的XML文档即可。

一旦成功地定义了原始XML资源，接下来在XML文件中就可通过如下语法格式来访问它：

在Java代码中则按如下语法格式来访问：

为了在Java程序中获取实际的XML文档，可以通过Resources的如下两个方法来实现。

➢ **XmlResourceParser getXml（int id）**：获取XML文档，并使用一个XmlPullParser来解析该XML文档，该方法返回一个解析器对象（XmlResourceParser是XmlPullParser的子类）。

➢ **InputStream openRawResource（int id）**：获取XML文档对应的输入流。

大部分时候，我们可以直接调用getXml（int id）方法来获取XML文档，并对该文档进行解析。Android默认使用内置的Pull解析器来解析XML文件。

除了使用Pull解析之外，Java开发者还可使用DOM或SAX对XML文档进行解析。一般的Java应用会使用JAXP API来解析XML文档。对于实际的Java EE项目而言，使用JDOM或dom4j进行解析可能更加简便。

提示：

Pull解析器是一个开源项目，既可以用于Android应用，也可以用于Java EE应用。如果需要在Java EE应用中使用Pull解析器，则需要自行下载并添加Pull解析器的JAR包。不过Android平台已经内置了Pull解析器，而且Android系统本身也使用Pull解析器来解析各种XML文档，因此Android推荐开发者使用Pull解析器来解析XML文档。

Pull解析方式有点类似于SAX解析，它们都采用事件驱动的方式来进行解析。当Pull解析器开始解析之后，开发者可不断地调用Pull解析器的next（）方法获取下一个解析事件（开始文档、结束文档、开始标签、结束标签等），当处于某个元素处时，可调用XmlPullParser的getAttributeValue（）方法来获取该元素的属性值，也可调用XmlPullParser的nextText（）方法来获取文本节点的值。

如果开发者希望使用 DOM、SAX 或其他解析器来解析 XML 资源，那么可调用openRawResource （int id）方法来获取XML资源对应的输入流，这样即可自行选择解析器来解析该XML资源了。

提示：

使用其他XML解析器需要开发者自行下载并安装解析器的JAR包。关于DOM、SAX、JAXP、dom4j、JDOM的相关知识，或者读者需要获取更多关于XML的知识，请参考疯狂Java体系的《疯狂XML讲义》。

6.6.2 使用原始XML文件

下面为示例程序添加一个原始的XML文件，将该XML文件放到/res/xml目录下。该XML文件的内容很简单，如下所示。

程序清单：
codes\06\6.6\XmlResTest\app\src\main\res\xml\books.xml

接下来就可以在Java程序中获取该XML资源，并解析该XML资源中的信息了。Java程序如下。

程序清单：
codes\06\6.6\XmlResTest\app\src\main\java\org\crazyit\res\MainActivity.java

图6.9 使用原始XML资源

上面程序中的①号粗体字代码用于不断地获取Pull解析的解析事件，程序中第一行粗体字代码只要解析事件不等于XmlResourceParser.END_DOCUMENT（也就是还没有解析结束），程序就将一直解析下去，通过这种方式即可把整份XML文档的内容解析出来。

上面的程序中包含一个按钮和一个文本框，当用户单击该按钮时，程序将会解析指定XML文档，并把文档中的内容显示出来。运行该程序，然后单击“解析XML资源”按钮，程序显示如图6.9所示的界面。

6.7 使用布局（Layout）资源

实际上，从我们学习第一个Android应用开始，已经开始接触Android的Layout资源了，因此此处不会详细介绍Android的Layout资源的知识，只是对Layout资源进行简单的归纳。

Layout资源文件应放在/res/layout/目录下，Layout资源文件的根元素通常是各种布局管理器，比如LinearLayout、TableLayout、FrameLayout等，接着在该布局管理器中定义各种View组件即可。

一旦在Android项目中定义了Layout资源，接下来在XML文件中就可通过如下语法格式来访问它：

在Java代码中则按如下语法格式来访问：

6.8 使用菜单（Menu）资源

前面已经介绍过Android的菜单支持，前面分别介绍了如何使用Java代码来实现菜单和使用XML资源文件来定义菜单。

实际上，Android推荐使用XML资源文件来定义菜单，这样将会提供更好的解耦。由于前面介绍过如何使用XML资源文件定义菜单，因此此处不再详细介绍菜单资源文件的内容，只是对其进行简单的归纳。

Android菜单资源文件放在/res/menu目录下，菜单资源的根元素通常是<menu.../>元素，<menu.../>元素无须指定任何属性。

一旦在Android项目中定义了菜单资源，接下来在XML文件中就可通过如下语法格式来访问它：

在Java代码中则按如下语法格式来访问：

6.9 样式（Style）和主题（Theme）资源

样式和主题资源都用于对Android应用进行“美化”，只要充分利用Android应用的样式和主题资源，开发者就可以开发出各种风格的Android应用。

6.9.1 样式资源

如果我们经常需要对某个类型的组件指定大致相似的格式，比如字体、颜色、背景色等，如果每次都要为View组件重复指定这些属性，无疑会有大量的工作量，而且不利于项目后期的维护。

类似于Word，Word也提供了样式来管理格式：一个样式等于一组格式的集合，如果设置某段文本使用某个样式，那么该样式的所有格式将会整体应用于这段文本。Android的样式与此类似，Android样式也包含一组格式，为一个组件设置使用某个样式时，该样式所包含的全部格式将会应用于该组件。

提示：

一个样式相当于多个格式的集合，其他UI组件通过style属性来指定样式，这就相当于把该样式包含的所有格式同时应用于该UI组件。

Android的样式资源文件也放在/res/values/目录下，样式资源文件的根元素是<resources.../>元素，该元素内可包含多个<style.../>子元

素，每个<style.../>子元素定义一个样式。<style.../>元素指定如下两个属性。

➢ **name**：指定样式的名称。

➢ **parent**：指定该样式所继承的父样式。当继承某个父样式时，该样式将会获得父样式中定义的全部格式。当然，当前样式也可以覆盖父样式中指定的格式。

<style.../>元素内可包含多个<item.../>子元素，每个<item.../>子元素定义一个格式项。

例如，为应用定义如下样式资源文件。

程序清单：
codes\06\6.9\StyleResTest\app\src\main\res\values\my_style.xml

上面的样式资源中定义了两个样式，其中第二个样式继承了第一个样式，而且第二个样式中的textColor属性覆盖了父样式中的textColor属性。

一旦定义了上面的样式资源之后，接下来就可以在XML资源中按如下语法格式来使用样式了：

下面是该示例中的界面布局文件，该布局文件中包含两个文本框，这两个文本框分别使用两个样式。

程序清单：
codes\06\6.9\StyleResTest\app\src\main\res\layout\main.xml

上面的界面布局文件中并未为两个文本框指定任何格式，只是为它们分别指定了使用style1、style2的样式，这两个样式包含的格式就会应用到这两个文本框。运行上面的程序，将看到如图6.10所示的界面。

图6.10 使用样式资源

6.9.2 主题资源

与样式资源非常相似，主题资源的XML文件通常也放在/res/values/目录下，主题资源的XML文件同样以<resources.../>元素作为根元素，同样使用<style.../>元素来定义主题。

主题与样式的区别主要体现在：

➢ 主题不能作用于单个的View组件，主体应该对整个应用中的所有Activity起作用，或对指定的Activity起作用。

➢ 主题定义的格式应该是改变窗口外观的格式，例如窗口标题、窗口边框等。

实例：给所有窗口添加边框、背景

下面通过一个实例来介绍主题资源的用法。为了给所有窗口都添加边框、背景，先在/res/values/my_style.xml文件中增加一个主题，定义主体的<style.../>片段如下：

上面的主题定义中使用了两个Drawable资源，其中@drawable/star是一张图片；@drawable/window_border是一个ShapeDrawable资源，该资源对应的XML文件代码如下。

程序清单：
codes\06\6.9\StyleResTest\app\src\main\res\drawable\window_border.xml

定义了上面主题之后，接下来即可在Java代码中使用该主题了，例如如下代码：

大部分时候，在AndroidManifest.xml文件中对指定应用、指定Activity应用主题更加简单。如果我们想让应用中全部窗口使用该主题，那么只要为<application.../>元素添加android：theme属性即可。属性值是一个主题的名字，如以下代码所示：

如果你只是想让程序中的某个Activity拥有这个主题，那么可以修改<activity.../>元素，同样通过android：theme指定主题即可。

图6.11 使用主题资源

本应用在AndroidManifest.xml文件的<application.../>元素中添加了android：theme="@style/CrazyTheme"属性，运行程序可以看到如图6.11所示的界面。

从图6.11所示的效果可以看出，该窗口没有标题，窗口背景也被改变了，窗口全屏显示……这些都是自定义主题控制的。

提示：

可能会有读者觉得窗口边框弄得这么粗，显得很难看，其实读者可以自行控制。笔者之所以弄得这么粗，是为了让读者看到窗口边框的效

果。

Android中提供了几种内置的主题资源，这些主题通过查询Android.R.style类可以看到。例如前面介绍的对话框风格的窗口，我们只要采用如下代码来定义某个Activity即可。

与样式类似的是，Android主题同样支持继承。如果开发过程中还想利用某个主题，但需要对它进行局部修改，则可通过继承系统主题来实现自定义主题。例如如下代码片段：

上面定义的CrazyTheme主题继承了android.R.style.Theme.Dialog主题，那么接下来在该<style.../>元素中添加的<item.../>子元素就可覆盖系统主题的部分属性了。

6.9.3 Android 5.0新增的Material主题

Android 5.0 新增了Material设计的主题，开发者只要将App的主题设为android：Theme.Material.Xxx（后面Xxx可能要根据实际选择）即可启用Material主题。

通过使用Material主题可以让App具有更自然的动态效果和过渡风格，Material Design支持包括以下几个方面。

➢ 系统自带Material Design主题。

➢ 允许通过View的setElevation（）或setTranslationZ（）方法让组件在屏幕上浮起来。

➢ 当用户点击按钮、复选框和其他可触碰组件时，将会自动显示水波动画。

➢ 具有Material设计风格的动画和Activity过渡效果。

实际上，前面介绍的绝大部分示例都使用了Material主题，只有第2章极个别的示例为了示范某些特殊组件的用法才改变了应用的主题，读者可通过前面示例来感受Android 5.0的Material主题的效果。

6.10 属性（Attribute）资源

前面已经介绍过自定义View组件的开发，自定义View组件与Android系统提供的View组件一样，既可在Java代码中使用，也可在XML界面布局代码中使用。

当在XML布局文件中使用Android系统提供的View组件时，开发者可以指定多个属性，这些属性可以很好地控制View组件的外观行为。如果用户开发的自定义View组件也需要指定属性，就需要属性资源的帮助了。

属性资源文件也放在/res/values目录下，属性资源文件的根元素也是<resources.../>元素，该元素包含如下两个子元素。

➢ **attr子元素**：定义一个属性。

➢ **declare-styleable子元素**：定义一个styleable对象，每个styleable对象就是一组attr属性的集合。

当我们使用属性资源文件定义了属性之后，接下来就可以在自定义组件的构造器中通过AttributeSet对象来获取这些属性了。

例如，我们想开发一个默认带动画效果的图片，该图片显示时，自动从全透明变到完全不透明，为此需要开发一个自定义组件，但这个自定义组件需要指定一个额外的duration属性，该属性控制动画的持续时间。

为了在自定义组件中使用duration属性，需要先定义如下属性资源文件。

程序清单：
codes\06\6.10\AttrResTest\app\src\main\res\values\attrs.xml

上面的属性资源文件定义了该属性之后，至于到底在哪个View组件中使用该属性，该属性到底能发挥什么作用，就不归属性资源文件管了。属性资源文件所定义的属性到底可以发挥什么作用，取决于自定义组件的代码实现。

例如如下自定义的AlphaImageView，它获取了定义该组件所指定的duration属性之后，根据该属性来控制图片透明度的改变。程序代码如下。

提示：

在属性资源文件中定义<declare-styleable.../>元素时，也可在其内部直接使用<attr.../>定义属性，使用<attr.../>时指定一个format属性即可，例如上面我们可以指定<attr name="duration" format="integer"/>。

程序清单：
codes\06\6.10\AttrResTest\app\src\main\java\org\crazyit\res\AlphaImageView.java

上面程序中的粗体字代码用于获取定义AlphaImageView时指定的duration属性，并根据该属性计算图片透明度的变化幅度，接着程序

重写了ImageView的onDraw（Canvas canvas）方法，该方法启动了一个任务调度来控制图片透明度的改变。

上面的粗体字代码中的R.styleable.AlphaImageView、R.styleable.AlphaImageView_duration都是Android SDK根据属性资源文件自动生成的。

接下来在界面布局文件中使用AlphaImageView时可以为它指定一个duration属性，注意该属性位于“http：//schemas.android.com/apk/res/+项目子包”命名空间下，例如本应用的包名为org.crazyit.res，那么duration属性就位于“http：//schemas.android.com/apk/res/org.crazyit.res”命名空间下。

下面是该应用的界面布局文件的代码。

程序清单：
codes\06\6.10\AttrResTest\app\src\main\res\layout\main.xml

上面程序中的第一行粗体字代码用于导入http：//schemas.android.com/apk/res/org.crazyit.res命名空间，并指定该命名空间对应的短名前缀为crazyit；第二行粗体字代码用于为AlphaImageView组件指定自定义属性duration的属性值为60000。

主程序无须做任何特殊的控制，只要简单地加载并显示上面的界面布局文件，运行该程序时即可看到该图片从无到有、慢慢显示出来的效果。

6.11 使用原始资源

除了上面介绍的各种XML文件、图片文件之外，Android应用可能还需要用到大量其他类型的资源，比如声音资源等。实际上，声音对于Android应用非常重要，选择合适的音效可以让Android应用增色不少。

类似于声音文件及其他各种类型的文件，只要Android没有为之提供专门的支持，这种资源都被称为原始资源。Android的原始资源可以放在如下两个地方。

➢ 位于/res/raw/目录下，Android SDK会处理该目录下的原始资源，Android SDK会在R清单类中为该目录下的资源生成一个索引项。

➢ 位于/assets/目录下，该目录下的资源是更彻底的原始资源。Android 应用需要通过AssetManager来管理该目录下的原始资源。

Android SDK会为位于/res/raw/目录下的资源在R清单类中生成一个索引项，接下来在XML文件中可通过如下语法格式来访问它：

在Java代码中则按如下语法格式来访问：

通过上面的索引项，Android应用就可以非常方便地访问/res/raw/目录下的原始资源了，至于获取原始资源之后如何处理，则完全取决于实际项目的需要。

AssetManager是一个专门管理/assets/目录下原始资源的管理器类，AssetManager提供了如下两个常用方法来访问Assets资源。

➢ **InputStream open（String fileName）**：根据文件名来获取原始资源对应的输入流。

➢ **AssetFileDescriptor openFd（String fileName）**：根据文件名来获取原始资源对应的AssetFileDescriptor。AssetFileDescriptor代表了一项原始资源的描述，应用程序可通过AssetFileDescriptor来获取原始资源。

下面的程序示范了如何使用声音。先往应用的/res/raw/目录下放入一个bomb.mp3文件—Android SDK会自动处理该目录下的资源，会在R清单类中为它生成一个索引项：R.raw.bomb。

接下来再往/assets/目录下放入一个shot.mp3文件—需要通过AssetManager进行管理。

下面的程序中定义了两个按钮，其中一个按钮用于播放/res/raw/目录下的声音文件；另一个按钮用于播放/assets/目录下的声音文件。程序界面布局代码很简单，此处不再给出。

程序清单：
codes\06\6.11\RawResTest\app\src\main\java\org\crazyit\res\MainActivity.java

上面程序中的第一行粗体字代码用于获取/res/raw/目录下的原始资源文件；第二段粗体字代码则利用了AssetManager来获取/assets/目录下的原始资源文件。

上面的程序中利用了MediaPlayer来播放声音，MediaPlayer是Android提供的一个播放声音的类，本书后面还有关于该类更详细的介绍。

6.12 国际化和资源自适应

全球化的Internet需要全球化的软件。全球化的软件即意味着同一种版本的产品能够容易地适用于不同地区的市场。引入国际化的目的是为了提供自适应、更友好的用户界面，并不需要改变程序的逻辑功

能。国际化的英文单词是Internationalization，因为这个单词太长了，有时也简称I18N，其中I是这个单词的第一个字母，18表示中间省略的字母个数，而N代表这个单词的最后一个字母。

一个国际化支持很好的应用，会随着在不同区域使用而呈现出本地语言的提示。这个过程也被称为Localization，即本地化。类似于国际化可以称为I18N，本地化也可以称为L10N。

Android所采用的资源管理方式可以非常方便地实现程序国际化。

6.12.1 Java国际化的思路

Java程序的国际化的思路是将程序中的标签、提示等信息放在资源文件中，程序需要支持哪些国家、语言环境，就需要提供相应的资源文件。资源文件是key-value对，每个资源文件中的key是不变的，但value则随不同国家、语言改变。图6.12显示了Java程序国际化的思路。

图6.12 Java程序国际化的思路

Java程序的国际化主要通过如下三个类完成。

➢ **java.util.ResourceBundle**：用于加载一个国家、语言资源包。

➢ **java.util.Locale**：用于封装一个特定的国家/区域、语言环境。

➢ **java.text.MessageFormat**：用于格式化带占位符的字符串。

为了实现程序的国际化，必须先提供程序所需要的资源文件。资源文件的内容适合很多key-value对，其中key是程序使用的部分，而value则是程序界面的显示字符串。

资源文件的命名可以有如下三种形式。

➢ **baseName_language_country.properties**

➢ **baseName_language.properties**

➢ **baseName.properties**

其中baseName是资源文件的基本名，用户可以自由定义。而language和country都不可随意变化，必须是Java所支持的语言和国家。

6.12.2 Java支持的国家和语言

事实上，Java不可能支持所有的国家和语言，如需要获取Java所支持的国家和语言，可调用Locale类的getAvailableLocales方法获取，该方法返回一个Locale数组，该数组里包含了Java所支持的国家和语言。

下面的程序简单地示范了如何获取Java所支持的国家和语言。

程序清单：codes\06\6.12\LocaleList.java

程序的运行结果如图6.13所示。

图6.13 Java国际化所支持的国家和语言

通过该程序，我们就可以获得Java程序所支持的国家/语言环境。

提示：

虽然可以通过查阅相关资料来获取Java语言所支持的国家/语言环境，但如果这些资料不是随手可得的，则可以通过上面的程序来获得Java语言所支持的国家/语言环境。

6.12.3 完成程序国际化

对于如下最简单的程序：

这个程序的执行结果也很简单，肯定是打印出简单的“Hello World”字符串，不管在哪里执行都不会有任何改变！为了让该程序支持国际化，则肯定不能在程序中直接输出“Hello World”字符串，这种写法直接输出一个字符串常量，永远不会有任何改变。为了让程序可以输出不同的字符串，此处绝不可使用该字符串常量。

为了让上面输出的字符串常量可以改变，我们将需要输出的各种字符串（不同国家/语言环境对应不同的字符串）定义在资源包中。

我们为上面程序提供如下两个文件。

第一个文件mess_zh_CN.properties，该文件的内容为：

第二个文件mess_en_US.properties，该文件的内容为：

对于包含非西欧字符的资源文件，Java提供了一个工具来处理该文件：native2ascii，这个工具可以在%JAVA_HOME%/bin路径下找到。使用该工具的语法格式如下：

如果我们在命令行窗口输入如下命令：

上面的命令将生成一个aa.properties文件，该文件才是我们需要的资源文件，该文件看上去包含很多乱码，其实是非西欧字符的Unicode编码方式，这完全正常。将该文件重命名为mess_zh_CN.properties即可。

我们看到这两份文件文件名的baseName是相同的：mess。前面已经介绍了资源文件的三种命名方式，其中baseName后面的国家、语言必须是Java所支持的国家、语言组合。

将上面的Java程序修改成如下形式。

程序清单：codes\06\6.12\Hello.java

上面程序中的打印语句不再是直接打印“Hello World”字符串，而是打印从资源包中读取的信息。如果在中文环境下运行该程序，将打印出“您好！”；如果我们在“控制面板”中将机器的语言环境设置成美国，然后再次运行该程序，将打印出“Welcome You！”字符串。

通过上面的简单程序，我们可以体会到Java程序的国际化是多么简单！

从上面的程序可以看出，如果我们希望程序完成国际化，只需要将不同国家/语言（Locale）的提示信息分别以不同文件存放即可。例如，简体中文的语言资源文件就是Xxx_zh_CN.properties文件，而美式英语的语言资源文件就是Xxx_en_US.properties文件。

Java程序国际化的关键类是ResourceBundle，它有一个静态方法：getBundle（String baseName，Locale locale）。该方法将根据Locale加载资源文件，而Locale封装了一个国家、语言，例如简体中文的环境可以用简体中文的Locale代表，美式英语的环境可以用美式英语的Locale代表。

从上面的资源文件的命名中可以看出，不同国家、语言环境的资源文件的baseName是相同的，即baseName为mess的资源文件有很多个，不同国家、语言环境对应不同的资源文件。

例如，通过如下代码来加载资源文件：

上面的代码将会加载baseName为mess的系列资源文件的其中之一，到底加载其中的哪一个，则取决于myLocale，对于简体中文的Locale，则加载mess_zh_CN.properties文件。

一旦加载了该文件，该资源文件的内容就是多个key-value对，程序就根据key来获取指定信息，例如获取key为hello的消息，该消息是“您好！”—这就是Java程序国际化的过程。

对于美式英语的Locale，则加载mess_en_US.properties文件，该文件中的key为hello的消息是“Welcome You！”。

Java程序国际化的关键类是ResourceBundle，ResourceBundle根据不同的Loacle加载语言资源文件，再根据指定key取得已加载语言资源文件中的字符串即可。

6.12.4 为Android应用提供国际化资源

为Android程序提供国际化资源更加方便—因为Android本身就采用了XML资源文件来管理所有字符串消息，只要为各消息提供不同国家、语言对应的内容即可。

通过前面的介绍我们知道，Android应用使用res\values\目录下的资源文件来保存程序中用到的字符串消息，为了给这些消息提供不同国家、语言的版本，开发者需要为values目录添加几个不同的语言国家版本。不同values文件夹的命名方式为：

例如，如果希望下面的应用支持简体中文、美式英语两种环境，则需要在res目录下添加values-zh-rCN、values-en-rUS两个目录。

如果希望应用程序的图片也能随国家、语言环境改变，那么还需要为drawable目录添加几个不同的语言国家版本。不同drawable文件夹的命名方式为：

提示：

如果还需要为drawable目录按分辨率提供文件夹，则可以在后面追加分辨率后缀，例如drawable-zh-rCN-mdpi、drawable-zh-rCN-hdpi、drawable-zh-rCN-xhdpi、drawable-en-rUS-mdpi、drawable-en-rUS-hdpi、drawable-en-rUS-xhdpi等。

下面的程序分别为values目录、drawable目录创建了简体中文、美式英语两个版本，如图6.14所示。

在\res\drawable-zh-rCN\、\res\drawable-en-rUS\目录下分别添加logo.png图片，这两个图片并不相同，一个是简体中文环境的图片，一个是美式英语环境的图片。

在\res\values-en-rUS\、\res\values-zh-rCN\目录下分别创建strings.xml文件，很明显，该文件中存放的就是字符串资源。

其中\res\values-en-rUS\目录下的strings.xml是美式英语的字符串资源文件，而\res\values-zh-rCN\目录下的strings.xml是简体中文的字符串资源文件。\res\values-en-rUS\目录下的strings.xml文件内容如下：

图6.14 国际化资源文件

\res\values-zh-rCN\目录下的strings.xml文件内容如下：

正如从Java国际化中所看到的，不同语言的国际化资源文件中所有消息的key是相同的，在不同国家、语言环境下，消息资源key对应的value不同。

由于Android的消息资源本身就是采用XML文件来提供的，所以不需要额外的处理。接下来就可以在Android应用中使用这些国际化消息资源了。

6.12.5 国际化Android应用

Android的设计本身就是国际化的，当开发者在XML界面布局文件、Java代码中加载字符串资源时，Android的国际化机制就已经起作用了。

对于下面的界面布局文件。

程序清单：
codes\06\6.12\I18N\app\src\main\res\layout\main.xml

上面三行粗体字代码与前面的界面布局代码没有任何改变，它本身就没有把字符串内容“写死”在布局文件中，它本身就是去加载资源文件中的字符串值，此时Android的国际化机制就可发挥作用了—如果Android是简体中文环境，就加载\res\values-zh-rCN\strings.xml文件中的字符串资源、加载\res\drawable-zh-rCN目录下的Drawable资源；如果是美式英语环境，就加载\res\values-en-rUS\strings.xml文件中的字符串资源、加载\res\drawable-en-rUS目录下的Drawable资源。

与此类似的是，在Java代码中编程时，程序同样可以根据资源ID设置字符串内容，而不是以硬编码的方式设置为固定的字符串内容。例如如下程序代码。

程序清单：
codes\06\6.12\I18N\app\src\main\java\org\crazyit\res\MainActivity.java

上面的程序为tvShow设置文本内容时并未以硬编码的方式设置为固定的字符串内容，而是设置为消息资源的name—相当于国际化消息的key。类似的，Android系统的国际化机制同样可以发挥作用：如果Android是简体中文环境，就加载\res\values-zh-rCN\strings.xml文件中的字符串资源；如果是美式英语环境，就加载\res\values-en-rUS\strings.xml文件中的字符串资源。

图6.16 简体中文环境

将手机设为美式英语环境（通过Android系统的Settings→Language&keyboard settings→Select language→English（United States）进行设置），运行该程序即可看到如图6.15所示的界面。

将手机设为简体中文环境（通过Android系统的Settings→Language&input→language→中文（简体）进行设置），运行该程序即可看到如图6.16所示的界面。

从图6.15、图6.16可以看出，两个程序是同一个程序，只是由于它们的运行环境不同，Android系统会控制程序加载不同的资源文件，因此程序界面上的图片、字符串就完全不同了。

提示：

上面程序中的标题并未改变，这是因为这个标题对应的字符串资源只有一份，而且是保存在values目录下的，不管是简体中文环境，还是美式英语环境，系统总是加载这份资源文件，因此程序标题是固定的。如果需要对程序标题也进行国际化，不难，只要为程序标题对应的字符串消息名（app_name）分别提供美式英语、简体中文的消息资源即可。

6.13 自适应不同屏幕的资源

开发Android应用有一个比较烦人的地方是：Android设备的屏幕尺寸、分辨率差别非常大，而开发者开发的Android应用总希望能在所有Android设备上运行，因此开发Android应用就需要考虑不同屏幕的适应性问题。

提示：

相比之下，开发iOS应用要更简单，因为iOS只有几种特定的手机和平板电脑两种设备，它们的屏幕尺寸、分辨率都是固定的，因此需要考虑的设备更少。

前面已经提到，Android默认把drawable目录（存放图片等Drawable资源的目录）分为drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi这4个子目录，这4个子目录就是为不同分辨率准备的图片。

通常来说，屏幕资源需要考虑如下两个方面。

➢ **屏幕尺寸：**屏幕尺寸可分为small（小屏幕）、normal（中等屏幕）、large（大屏幕）、xlarge（超大屏幕）4种。

➢ **屏幕分辨率：**屏幕分辨率可分为ldpi（低分辨率）、mdpi（中等分辨率）、hdpi（高分辨率）、xhdpi（超高分辨率）、xxhdpi（超超高分辨率）5种。

➢ **屏幕方向：**屏幕方向可分为land（横屏）和port（竖屏）两种。

图6.17显示了不同屏幕尺寸、不同分辨率对应的通用说法。

图6.17 不同尺寸、不同分辨率的屏幕

当我们为不同尺寸的屏幕设置用户界面时，每种用户界面总有一个最低的屏幕尺寸要求（低于该屏幕尺寸就没法运行），上面这些通用说法中屏幕尺寸总有一个最低分辨率，该最低分辨率是以dp为单位 的，因此我们在定义界面布局时应该尽量考虑使用dp为单位。

下面是上面4种屏幕尺寸所需的最低尺寸。

➢ xlarge屏幕尺寸至少需要960dp×720dp。

➢ large屏幕尺寸至少需要640dp×480dp。

➢ normal屏幕尺寸至少需要470dp×320dp。

➢ small屏幕尺寸至少需要426dp×320dp。

通过上面的介绍不难发现，为了提供自适应不同屏幕的资源，最简单的做法就是为不同屏幕尺寸、不同屏幕分辨率提供相应的布局资源、Drawable资源。

➢ **屏幕分辨率：**可以为drawable目录增加后缀ldpi（低分辨率）、mdpi（中等分辨率）、hdpi（高分辨率）、xhdpi（超高分辨率），分别为不同分辨率的屏幕提供资源。

➢ **屏幕尺寸：**可以为layout、values等目录增加后缀small、normal、large和xlarge，分别为不同尺寸的屏幕提供相应资源。

注意：

从Android 3.2开始，Android建议直接使用真实的屏幕尺寸来定义屏幕尺寸。Android 3.2支持在layout、values目录后添加sw<N>dp（屏幕尺寸至少宽N个dp才能使用该资源）、w<N>dp（屏幕尺寸可用宽度为N个dp可使用该资源）、h<N>dp（屏幕尺寸可用高度为N个dp才能使用该资源）。例如可指定layout-sw600dp，表明该设备屏幕的宽度大于或等于600个dp时使用该目录下的布局资源。

➢ **屏幕方向：**可以为layout、values等目录增加后缀land和port，分别为横屏、竖屏提供相应的资源。

下面的示例在layout目录下定义了一个界面布局文件。

程序清单：
codes\06\6.13\DpiTest\app\src\main\res\layout\main.xml

上面的界面布局文件指定显示@drawable/a对应的图片资源，在hdpi设备上运行程序时，系统将会使用/res/drawable-hdpi目录下的图片。运行该程序，将看到如图6.18所示的界面。

在xhdpi设备上运行程序时（比如768×1280分辨率、320dpi），系统将会使用/res/drawable-xhdpi/目录下的图片。运行该程序，将看到如图6.19所示的界面。

图6.18 hdpi屏幕使用drawable-hdpi目录下的图片

图6.19 xhdpi屏幕使用drawable-xhdpi目录下的图片

下面的示例提供了两份布局文件，其中一份放在layout-normal目录下，一份放在layout-large目录下。layout-normal目录下的布局文件代

码如下。

程序清单：
codes\06\6.13\ScreenSizeTest\app\src\main\res\layout-normal\main.xml

layout-large目录下的布局文件代码如下。

程序清单：
codes\06\6.13\ScreenSizeTest\app\src\main\res\layout-large\main.xml

在正常尺寸的屏幕（分辨率为768×1280、320dpi）上运行该程序，系统将会加载layout-normal目录下的main.xml布局文件。运行该程序，将看到如图6.20所示的界面。

图6.20 正常尺寸的屏幕加载layout-normal目录下的界面布局文件

在大尺寸的屏幕（分辨率为720×1280、240dpi）上运行该程序，系统将会加载layout-large目录下的main.xml布局文件。运行该程序，将看到如图6.21所示的界面。

图6.21 大尺寸的屏幕加载layout-large目录下的界面布局文件

可能有读者感到奇怪：720×1280的分辨率不是比768×1280分辨率更低吗？怎么Android反而使用layout-large目录下的布局文件呢？这是因为手机屏幕大小不仅与分辨率有关，还与dpi有关。dpi的意思是Dots Per Inch，即每英寸包含的点数，这意味着在相同分辨率情况

下，dpi越高，屏幕反而越小；但dpi越高，图像显示效果就越细腻。因此720×1280、240dpi的屏幕其实比768×1280、320dpi的屏幕更大，故而Android在720×1280、240dpi的屏幕上会选择layout-large目录下的main.xml布局文件。

6.14 本章小结

本章主要介绍了Android应用资源的相关内容。Android应用资源是一种非常优秀的、高解耦设计，通过使用资源文件，Android应用可以把各种字符串、图片、颜色、界面布局等交给XML文件配置管理，这样就避免在Java代码中以硬编码的方式直接定义这些内容。学习本章需要掌握Android应用资源的存储方式、Android应用资源的使用方式。除此之外，Android应用的字符串资源、颜色资源、尺寸资源、数组资源、图片资源、各种Drawable资源、原始XML资源、布局资源、菜单资源、样式和主题资源、属性资源、原始资源等各种资源文件都需要重点掌握。

本章最后还介绍了Android应用的国际化。由于Android应用的国际化实际上是以Java国际化为基础的，因此本章还简单介绍了Java程序的国际化。这些内容都需要读者认真掌握。

第7章 图形与图像处理

本章要点

Android的图形处理基础

Bitmap与BitmapFactory

继承View在Android中绘图

掌握Canvas、Paint、Path等绘图API

双缓冲机制

使用Matrix对图像进行几何变换

通过drawBitmapMesh方法扭曲图像

使用不同的Shader类渲染图形

逐帧动画

补间动画

属性动画

开发自定义补间动画

SurfaceView的绘图机制

继承SurfaceView开发动画

正如前面所介绍的，决定Android应用是否被用户接受的重要方面就是用户界面，为了提供友好的用户界面，就需要在应用中使用图片了。

Android系统提供了丰富的图片功能支持，包括处理静态图片和动画等。

Android系统提供了ImageView显示普通的静态图片，也提供了AnimationDrawable来开发逐帧动画，还可通过Animation对普通图片使用补间动画。图形、图像处理不仅对Android系统的应用界面非常重要，而且Android系统上的益智类游戏、2D游戏都需要大量的图形、图像处理。所谓游戏，本质就是提供更逼真的、能模拟某种环境的用户界面，并根据某种规则来响应用户操作。为了提供更逼真的用户界面，需要借助于图形、图像处理。

学习本章内容之后，读者应该能熟练掌握Android系统的图形、图像处理，这样就可以在Android平台上开发出俄罗斯方块、五子棋等小游戏了。本章将会通过弹球游戏、飞行游戏雏型来帮助读者掌握Android 2D游戏开发的入门知识。

7.1 使用简单图片

前面的Android应用中已经大量使用了简单图片，图片不仅可以使用ImageView来显示，也可作为Button、Window的背景。从广义的角度来看，Android应用中的图片不仅包括*.png、*.jpg、*.gif等各种格式的位图，也包括使用XML资源文件定义的各种Drawable对象。

7.1.1 使用Drawable对象

为Android应用增加了Drawable资源之后，Android SDK会为这份资源在R清单文件中创建一个索引项：R.drawable.file_name。

接下来既可在XML资源文件中通过@drawable/file_name访问该Drawable对象，也可在Java代码中通过R.drawable.file_name访问该Drawable对象。

需要指出的是，R.drawable.file_name是一个int类型的常量，它只代表Drawable对象的ID，如果Java程序中需要获取实际的Drawable对

象，则可调用Resources的getDrawable（int id）方法来实现。

由于前面已经介绍了大量关于Drawable的示例，故此处不再给出示例。

7.1.2 Bitmap和BitmapFactory

Bitmap代表一个位图，BitmapDrawable里封装的图片就是一个Bitmap对象。开发者为了把一个Bitmap对象包装成BitmapDrawable对象，可以调用BitmapDrawable的构造器：

如果需要获取BitmapDrawable所包装的Bitmap对象，则可调用BitmapDrawable的getBitmap（）方法，如下面的代码所示：

除此之外，Bitmap还提供了一些静态方法来创建新的Bitmap对象，例如如下常用方法。

➢ **createBitmap（Bitmap source，int x，int y，int width，int height）**：从源位图source的指定坐标点（给定x、y）开始，从中"挖取"宽width、高height的一块出来，创建新的Bitmap对象。

➢ **createScaledBitmap（Bitmap src，int dstWidth，int dstHeight，boolean filter）**：对源位图src进行缩放，缩放成宽dstWidth、高dstHeight的新位图。

➢ **createBitmap（int width，int height，Bitmap.Config config）**：创建一个宽width、高height的新位图。

➢ **createBitmap（Bitmap source，int x，int y，int width，int height，Matrix m，boolean filter）**：从源位图 source 的指

定坐标点（给定 x、y）开始，从中“挖取”宽 width、高 height的一块出来，创建新的Bitmap对象，并按Matrix指定的规则进行变换。

BitmapFactory是一个工具类，它提供了大量的方法，这些方法可用于从不同的数据源来解析、创建Bitmap对象。BitmapFactory包含了如下方法。

➢ **decodeByteArray（byte[] data，int offset，int length）：** 从指定字节数组的offset位置开始，将长度为length的字节数据解析成Bitmap对象。

➢ **decodeFile（String pathName）：** 从pathName指定的文件中解析、创建Bitmap对象。

➢ **decodeFileDescriptor（FileDescriptor fd）：** 用于从FileDescriptor对应的文件中解析、创建Bitmap对象。

➢ **decodeResource（Resources res，int id）：** 用于根据给定的资源ID从指定资源中解析、创建Bitmap对象。

➢ **decodeStream（InputStream is）：** 用于从指定输入流中解析、创建Bitmap对象。

大部分时候，我们只要把图片放在/res/drawable/目录下，就可以在程序中通过该图片对应的资源ID来获取封装该图片的Drawable对象。但由于手机系统的内存比较小，如果系统不停地去解析、创建Bitmap对象，可能由于前面创建Bitmap所占用的内存还没有回收，而导致程序运行时引发OutOfMemory错误。

Android为Bitmap提供了两个方法来判断它是否已回收，以及强制Bitmap回收自己。

➢ **boolean isRecycled（）：** 返回该Bitmap对象是否已被回收。

➢ **void recycle（）：** 强制一个Bitmap对象立即回收自己。

除此之外，如果Android应用需要访问其他存储路径（比如SD卡）里的图片，那么都需要借助于BitmapFactory来解析、创建Bitmap对象。

下面开发一个查看/assets/目录下图片的图片查看器。该程序界面十分简单，只包含一个ImageView和一个按钮，当用户单击该按钮时程序会自动去搜寻/assets/目录下的下一张图片。此处不再给出界面布局代码，该程序的代码如下。

程序清单：
codes\07\7.1\BitmapTest\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的①号粗体字代码用于判断当前ImageView所显示的图片是否已被回收，如果该图片还未回收，则系统强制回收该图片；程序的②号粗体字代码调用了BitmapFactory从指定输入流解析并创建Bitmap对象。

7.2 绘图

除了使用已有的图片之外，Android应用还常常需要在运行时动态地生成图片，比如一个手机游戏，游戏界面看上去丰富多彩，而且可以随着用户动作而动态改变，这就需要借助于Android的绘图支持了。

7.2.1 Android绘图基础：Canvas、Paint等

有过Swing编程经验的读者都知道，在Swing中绘图的思路是需要开发一个自定义类，该自定义类继承JPanel，并重写JPanel的paint（Graphics g）方法即可。Android的绘图与此类似，Android的绘图应该继承View组件，并重写它的onDraw（Canvas canvas）方法即可。

重写onDraw（Canvas canvas）方法时涉及一个绘图API：Canvas，Canvas代表“依附”于指定View的画布，它提供了如表7.1所示的方法来绘制各种图形。

表7.1 Canvas的绘制方法

除了表7.1中所定义的各种方法之外，Canvas还提供了如下方法进行坐标变换。

➢ **rotate（float degrees，float px，float py）：** 对Canvas执行旋转变换。

➢ **scale（float sx，float sy，float px，float py）：** 对Canvas执行缩放变换。

➢ **skew（float sx，float sy）：** 对Canvas执行倾斜变换。

➢ **translate（float dx，float dy）：** 移动Canvas。向右移动dx距离（dx为负数即向左移动）；向下移动dy距离（dy为负数即向上移动）。

Canvas提供的方法还涉及一个API：Paint，Paint代表Canvas上的画笔，因此Paint类主要用于设置绘制风格，包括画笔颜色、画笔笔触粗细、填充风格等。Paint提供了如表7.2所示的方法。

表7.2 Paint的常用方法

续表

在Canvas提供的绘制方法中还用到了一个API：Path，Path代表任意多条直线连接而成的任意图形，当Canvas根据Path绘制时，它可以绘制出任意的形状。

下面的程序示范了如何在Android应用中绘制基本的几何图形。该程序的关键在于一个自定义View组件，程序重写该View组件的onDraw（Canvase）方法，接下来在该Canvas上绘制了大量几何图形。这个自定义View的代码如下。

程序清单：
codes\07\7.2\CanvasTest\app\src\main\java\org\crazyit\image\MyView.java

上面的程序中大量调用了Canvas的方法来绘制几何图形，而且程序的第二段粗体字代码还为Paint画笔设置了使用渐变、阴影，因此接下来绘制的几何图形将采用渐变填充，而且具有阴影。使用一个Activity来显示上面的MyView类，运行程序将看到如图7.1所示的效果。

Android的Canvas不仅可以绘制这种简单的几何图形，还可以直接将一个Bitmap绘制到画布上，这样就给了开发者巨大的灵活性，只要前期美工把应用程序所需的图片制作出来，后期开发时把这些图片绘制到Canvas上即可。

7.2.2 Path类

从上面的程序可以看出，Android提供的Path是一个非常有用的类，它可以预先在View上将N个点连成一条“路径”，然后调用Canvas的drawPath（path，paint）方法即可沿着路径绘制图形。实际上Android还为路径绘制提供了PathEffect来定义绘制效果，PathEffect包含了如下子类（每个子类代表一种绘制效果）。

图7.1 Android绘图入门

➢ ComposePathEffect

➢ CornerPathEffect

➢ DashPathEffect

➢ DiscretePathEffect

➢ PathDashPathEffect

➢ SumPathEffect

这些绘制效果使用语言来表述总显得有点空洞，下面通过一个程序来让读者理解这些绘制效果。该程序绘制7条路径，分别示范了不使用效果和使用上面6种效果的效果。

程序清单：
codes\07\7.2\PathTest\app\src\main\java\org\crazyit\image\MainActivity.java

图7.2 路径效果示例

正如上面的程序中所看到的，当定义DashPathEffect、PathDashPathEffect时可指定一个phase参数，该参数用于指定路径效果的相位，当该phase参数改变时，绘制效果也略有变化。上面的程序不停地改变phase参数，并不停地重绘该View组件，这将产生动画效果，如图7.2所示。

除此之外，Android的Canvas还提供了一个drawTextOnPath （String text，Path path，float hOffset，float vOffset，Paint paint）方法，该方法可以沿着Path绘制文本。其中hOffset参数指定水平偏移，vOffset参数指定垂直偏移。例如如下程序。

程序清单：
codes\07\7.2\PathText\app\src\main\java\org\crazyit\image\MainActivity.java

上面的程序三次调用了drawTextOnPath（）在View组件上绘制文本，此时绘制的文本并不是简单地水平排列，而是沿着指定路径绘制的。运行上面的程序，将看到如图7.3所示的效果。

7.2.3 绘制游戏动画

掌握了Canvas绘图知识之后，如果需要实现游戏动画也是非常简单的。动画其实就是不断地重复调用View组件的onDraw（Canvas canvas）方法，如果每次在View组件上绘制的图形并不相同，那么就成了习惯上所说的动画。

图7.3 沿着路径绘制文本

为了让View组件上绘制的图形发生改变（无非是位置、大小、角度等发生改变），这就需要程序采用变量来“记住”这些状态数据—如果需要游戏动画随用户操作而改变，就为用户动作编写事件监听器，在监听器中修改这些数据；如果需要游戏动画“自动”改变，也就是随时间的流逝而改变，那么就需要使用定时器（Timer），让Timer控制这些状态数据定期改变。

不管使用哪种方式，每次View组件上的图形状态数据发生改变时，都应该通知View组件重写调用onDraw（Canvas canvas）方法重绘该组件。通知View重绘可调用invalidate（在UI线程中）或postInvalidate（在非UI线程中）。

实例：采用双缓冲实现画图板

本实例要实现一个画图板，当用户在触摸屏上移动时，即可在屏幕上绘制任意的图形。实现手绘功能其实是一种假象：表面上看起来可以随用户在触摸屏上自由地画曲线，实际上依然利用的是Canvas的drawLine（）方法画直线，每条直线都是从上一次拖动事件的发生点画到本次拖动事件的发生点。当用户在触摸屏上移动时，两次拖动事件发生点的距离很小，多条极短的直线连接起来，肉眼看起来就是直线了。借助于Android提供的Path类，可以非常方便地实现这种效果。

需要指出的是，如果程序每次都只是从上次拖动事件的发生点绘一条直线到本次拖动事件的发生点，那么用户前面绘制的就会丢失。为了保留用户之前绘制的内容，程序要借助于“双缓冲”技术。

所谓双缓冲技术其实很简单：当程序需要在指定View上进行绘制时，程序并不直接绘制到该View组件上，而是先绘制到内存中的一个Bitmap图片（这就是缓冲区）上，等到内存中的Bitmap绘制好之后，再一次性地将Bitmap绘制到View组件上。

该程序需要一个自定义View，该View的代码如下。

程序清单：
codes\07\7.2\HandDraw\app\src\main\java\org\crazyit\image\DrawView.java

上面程序中的粗体字代码为触摸屏的拖动事件提供了响应一只是简单地修改了preX、preY两个属性，并通知该组件重绘。

在这个自定义View组件中，程序重写了该View的onDraw（Canvas canvas）方法，注意该方法中的②号代码，这行代码并不是调用该View的Canvas进行绘制，而是调用了缓存Bitmap的Canvas进行绘图，这表明是向缓冲区绘图。程序的②号粗体字代码将缓冲区中的BitMap对象绘制到View组件上一这就是所谓的"双缓冲"技术。

提供了上面的DrawView之后，接下来把该组件添加到主界面中，本程序无须使用界面布局文件，本程序直接在Activity中使用代码创建程序界面。

本程序还提供了菜单来设置画笔的颜色和笔触大小，该程序的菜单资源文件如下。

程序清单：
codes\07\7.2\HandDraw\app\src\main\res\menu\menu_main.xml

主程序负责加载、显示界面布局，加载、显示上面的菜单资源。除此之外，程序还要为各菜单项编写事件响应，程序代码如下。

程序清单：
codes\07\7.2\HandDraw\app\src\main\java\org\crazyit\image\MainActivity.java

上面的程序代码比较简单，当用户单击不同的菜单项之后，程序只要简单地修改DrawView组件内的Paint对象的颜色和笔触粗细即可。运

行上面的程序，将看到如图7.4所示的界面。

提示：

阅读过《疯狂Java讲义》一书的读者可能对这个程序感到很“眼熟”，实际上这个程序所用的“双缓冲”技术，编程思路都来自《疯狂Java讲义》一书中11.8节的示例。正如笔者前面提到的，Android开发并不难—如果读者有扎实的Java基础，再加上一定的界面编程经验，学习Android开发将非常轻松。

图7.4 双缓冲实现绘图板

实例：弹球游戏

下面的程序开发了一个简单的弹球游戏，其中小球和球拍分别以圆形区域和矩形区域代替，小球开始以随机速度向下运动，遇到边框或球拍时小球反弹；球拍则由用户控制，当用户按下向左、向右键时，球拍将会向左、向右移动。

程序清单：
codes\07\7.2\PinBall\app\src\main\java\org\crazyit\image\MainActivity.java

上面的程序提供了一个GameView，这个GameView很简单，它只是根据程序中小球的坐标、球拍的坐标来绘制小球和球拍。

除此之外，这个所谓的“游戏”本质上就是一个动画程序，该程序中可以“动”的内容很少，只有一个小球和一个球拍。其中小球的坐标由定时器定时修改（如程序中①号代码所示）；球拍的坐标则由键盘动作

来修改（如程序中②号代码所示）。运行上面的程序，将可以看到如图7.5所示的游戏界面。

虽然图7.5所示的弹球游戏还略嫌粗糙，但只要开发者找到一些“美丽”的图片，替换程序中的小球、球拍，再考虑在界面上方增加一些“障碍物”来增加游戏乐趣，以及为小球碰撞边界、碰撞球拍时增加音效，这个游戏将会变得“生动”起来。这个游戏的AWT版本同样来自于《疯狂Java讲义》，疯狂Java联盟（crazyit.org）站点上有大量已经完成的弹球游戏，读者也可利用那些游戏中的图片资源来完善这个游戏。

7.3 图形特效处理

图7.5 弹球游戏界面

Android除了前面介绍的这些图形支持之外，还提供了一些额外的更高级的图形特效支持，这些图形特效支持可以让开发者开发出更绚丽的UI界面。

7.3.1 使用Matrix控制变换

Matrix是Android提供的一个矩阵工具类，它本身并不能对图形或组件进行变换，但它可与其他API结合来控制图形、组件的变换。

使用Matrix控制图形或组件变换的步骤如下。

1 获取Matrix对象，该Matrix对象既可新创建，也可直接获取其他对象内封装的Matrix（例如Transformation对象内部就封装了Matrix）。

2 调用Matrix的方法进行平移、旋转、缩放、倾斜等。

3 将程序对Matrix所做的变换应用到指定图形或组件。

提示：

从这里的介绍可以看出，Matrix不仅可用于控制图形的平移、旋转、缩放、倾斜变换，也可控制View组件进行平移、旋转和缩放等。

Matrix提供了如下方法来控制平移、旋转和缩放。

➢ **setTranslate（float dx，float dy）**：控制Matrix进行平移。

➢ **setSkew（float kx，float ky，float px，float py）**：控制Matrix以px、py为轴心进行倾斜。kx、ky为X、Y方向上的倾斜距离。

➢ **setSkew（float kx，float ky）**：控制Matrix进行倾斜。kx、ky为X、Y方向上的倾斜距离。

➢ **setRotate（float degrees）**：控制Matrix进行旋转，degrees控制旋转的角度。

➢ **setRotate（float degrees，float px，float py）**：设置以px、py为轴心进行旋转，degrees控制旋转的角度。

➢ **setScale（float sx，float sy）**：设置Matrix进行缩放，sx、sy控制X、Y方向上的缩放比例。

➢ **setScale（float sx，float sy，float px，float py）**：设置Matrix以px、py为轴心进行缩放，sx、sy控制X、Y方向上的缩放比例。

一旦对Matrix进行了变换，接下来就可应用该Matrix对图形进行控制了。例如，Canvas就提供了一个drawBitmap（Bitmap bitmap，Matrix matrix，Paint paint）方法，调用该方法就可以在绘制bitmap时应用Matrix上的变换。

如下程序开发了一个自定义View，该自定义View可以检测到用户的键盘事件，当用户单击手机的方向键时，该自定义View会用Matrix对绘

制的图形进行旋转、倾斜变换。

程序清单：
codes\07\7.3\MatrixTest\app\src\main\java\org\crazyit\image\MyView.java

上面程序中的粗体字代码就是通过Matrix控制图片倾斜、缩放的关键代码，当用户单击手机的方向键时，事件处理器负责修改程序中sx（控制水平倾斜度）和scale（控制缩放比）两个参数。

把上面的自定义View放在Activity中显示出来，运行该程序将看到如图7.6所示的界面。

图7.6 使用Matrix控制倾斜

实例：移动游戏背景

借助于Bitmap的createBitmap方法可以“挖取”源位图的其中一块，这样可以在程序中通过定时器控制不断地“挖取”源位图不同位置的块，从而给用户看到背景移动“假象”。

假设要开发经典的“雷电”飞机游戏，为了给用户造成飞机不断飞行的错觉，可以通过这种方式来控制背景图片不断下移，用户就会感觉飞机在不断地向上飞行。

例如，如下程序实现了背景图片不断“下移”的效果，用户会感觉飞机在不断地向上飞行。

程序清单：
codes\07\7.3\MoveBack\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的①号粗体字代码根据startY控制“挖取”back位图中不同位置的块，并将它绘制在当前View上。程序的粗体字代码则通过定时器来控制startY不断地改变，这样即可看到back图片不断下移的效果。运行该程序，读者即可看到飞机向上飞行的“错觉”。

提示：

该程序并未为用户按键事件提供监听器，实际上读者完全可以为该View的按键事件绑定监听器，这样用户即可控制飞机飞行。这已经是雷电游戏的雏型了。如果读者想要自己更好地完善这个雷电游戏，可以从如下三个方面进行改进。

（1）为该View添加一个数组来控制随机出现的敌机的坐标，当然也要通过定时器来控制敌机坐标的改变。

（2）为该View添加一个数组来控制飞机所发出的子弹的坐标，当然也要通过定时器来控制子弹坐标的改变。

（3）定期检查子弹是否与敌机“碰撞”，检查敌机是否与自己的飞机“碰撞”如果发生碰撞，程序在“碰撞”点播放“爆炸”动画。本章后面会介绍在指定点播放“爆炸”动画的示例。

7.3.2 使用drawBitmapMesh扭曲图像

Canvas提供了一个drawBitmapMesh（Bitmap bitmap，int meshWidth，int meshHeight，float[] verts，int vertOffset，int[] colors，int colorOffset，Paint paint）方法，该方法可以对bitmap进行扭曲。这个方法非常灵活，如果用好这个方法，开发者可以在Android应用上开发出“水波荡漾”、“风吹旗帜”等各种扭曲效果。

drawBitmapMesh（）方法的关键参数说明如下。

➢ **bitmap**：指定需要扭曲的源位图。

➢ **meshWidth**：该参数控制在横向上把该源位图划分成多少格。

➢ **meshHeight**：该参数控制在纵向上把该源位图划分成多少格。

➢ **verts**：该参数是一个长度为（meshWidth+1）*（m eshHeight+1）*2的数组，它记录了扭曲后的位图各"顶点"（图 7.7 所示网格线的交点）位置。虽然它是一个一维数组，但实际上它记录的数据是形如（x0，y0）、（x1，y1）、（x2，y2）、…、（xN，yN）格式的数据，这些数组元素控制对bitmap位图的扭曲效果。

➢ **vertOffset**：控制 verts 数组中从第几个数组元素开始才对 bitmap 进行扭曲（忽略verOffset之前数据的扭曲效果）。

drawBitmapMesh（）方法对源位图扭曲时最关键的参数是meshWidth、meshHeight、verts，这三个参数对扭曲的控制如图7.7所示。

从图7.7可以看出，当程序希望调用drawBitmapMesh（）方法对位图进行扭曲时，关键是计算verts数组的值一该数组的值记录了扭曲后的位图上各"顶点"（图7.7所示网格线的交点）的坐标。

图7.7 drawBitmapMesh扭曲示意图

提示：

初学者往往容易对drawBitmapMesh（）方法感到不好理解，如果读者有Photoshop图形处理的经验，应该对图7.7所示的扭曲效果很熟悉一实际上该方法可以模拟Photoshop的扭曲"滤镜"。Android应用面向的终端用户都是普通人群，应用界面对这些用户的影响非常大，Android提供的drawBitmapMesh（）方法带给开发者一种非常灵活的

控制。在实际开发中，笔者甚至“贪心”地希望Android提供更多方法，这些方法可以更好地模拟Photoshop的各种“滤镜”。

实例：可揉动的图片

本实例程序将会通过drawBitmapMesh（）方法来控制图片的扭曲，当用户“触摸”图片的指定点时，该图片会在这个点被用户“按”下去—就像这张图片铺在“极软的床上”一样。

为了实现这个效果，程序要在用户触摸图片的指定点时，动态地改变verts数组里每个元素的位置（控制扭曲后每个顶点的坐标）—这种改变也简单：程序计算图片上每个顶点与触摸点的距离，顶点与触摸点的距离越小，该顶点向触摸点移动的距离越大。

下面是该程序的代码。

程序清单：
codes\07\7.3\MeshTest\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的粗体字代码是关键，该方法将会根据触摸点的位置（由cx、cy坐标控制）动态修改verts数组里所有数组元素的值，这样就控制了drawBitmapMesh（）方法的扭曲效果。

运行该程序并触碰该图片的任意位置，将可以看到如图7.8所示的效果。

7.3.3 使用Shader填充图形

前面介绍Paint时提到该Shader包含了一个setShader（Shader s）方法，该方法控制“画笔”的渲染效果—Android不仅可以使用颜色来填充

图形（包括前面介绍的矩形、椭圆、圆形等各种几何图形），也可以使用Shader对象指定的渲染效果来填充图形。

图7.8 扭曲图像

Shader本身是一个抽象类，它提供了如下实现类。

➢ **BitmapShader**：使用位图平铺的渲染效果。

➢ **LinearGradient**：使用线性渐变来填充图形。

➢ **RadialGradient**：使用圆形渐变来填充图形。

➢ **SweepGradient**：使用角度渐变来填充图形。

➢ **ComposeShader**：使用组合渲染效果来填充图形。

如果使用文字来描述这些渲染效果，不仅显得十分啰唆，而且极难讲清楚。但如果读者有“玩”Flash的经验，则应该对位图平铺、线性渐变、圆形渐变、角度渐变等名词十分熟悉，那么此处就无须笔者多费笔墨来描述这些渲染效果了，因为它们与Flash提供的位图平铺、线性渐变、圆形渐变、角度渐变完全一样。如果读者没有Flash经验，也无须担心，运行下面的程序即可明白各种Shader对象的渲染效果。

下面的程序中包含了5个按钮，当用户按下不同的按钮时系统将会设置Paint使用不同的Shader，这样读者即可看到不同Shader的效果。

程序清单：
codes\07\7.3\ShaderTest\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的第一段粗体字代码创建了多个Shader对象，第二段粗体字代码会根据用户按下不同的按钮来修改MyView对象Paint—MyView对象将会用该Paint来绘制一个矩形。随着MyView的Paint所使用的Shader不断改变，MyView对象所绘制的矩形也随之改变。

运行上面的程序，如果使用BitmapShader绘制矩形，将看到如图7.9所示的效果。

如果使用SweepGradient绘制矩形，将看到如图7.10所示的效果。

图7.9 位图平铺的渲染效果

图7.10 角度渐变的渲染效果

图7.9、图7.10显示了Android提供的BitmapShader、SweepGradient两种Shader的渲染效果，读者可自行运行该程序去查看其他Shader的渲染效果。

7.4 逐帧（Frame）动画

逐帧（Frame）动画是最容易理解的动画，它要求开发者把动画过程的每张静态图片都收集起来，然后由Android来控制依次显示这些静态图片，再利用人眼“视觉暂留”的原理，给用户造成“动画”的错觉。逐帧动画的动画原理与放电影的原理完全一样。

7.4.1 AnimationDrawable与逐帧动画

前面讲解定义Android资源时已经介绍了动画资源，事实上逐帧动画通常也是采用XML资源文件进行定义的。

定义逐帧动画非常简单，只要在<animation-list.../>元素中使用<item.../>子元素定义动画的全部帧，并指定各帧的持续时间即可。定义逐帧动画的语法格式如下：

在上面的语法格式中，android：onshot控制该动画是否循环播放，如果该参数指定为true，则动画将不会循环播放；否则该动画将会循环播放。每个<item.../>子元素添加一帧。

提示：

Android完全支持在Java代码中创建逐帧动画，如果开发者喜欢，则完全可以先创建AnimationDrawable对象，然后调用addFrame（Drawable frame，int duration）向该动画中添加帧，每调用一次addFrame（）方法，就向<animation-list.../>元素中添加一个<item.../>子元素。

一旦程序获取了AnimationDrawable对象之后，接下来就可用ImageView把AnimationDrawable显示出来一习惯上把AnimationDrawable设成ImageView的背景即可。

下面是逐帧动画的简单示例，该示例先使用如下代码定义了一个逐帧动画资源。

程序清单：
codes\07\7.4\FatPo\app\src\main\res\anim\fat_po.xml

上面的fat_po.xml文件定义了一个逐帧动画资源，接下来就可以在程序中使用ImageView来显示该动画了。

需要指出的是，AnimationDrawable代表的动画默认是不播放的，必须在程序中启动动画播放才可以。AnimationDrawable提供了如下两个方

法来开始、停止动画。

➢ **start（）：**开始播放动画。

➢ **stop（）：**停止播放动画。

下面的程序中包含两个按钮，其中一个按钮用于播放动画；另一个按钮可停止动画。

程序清单：
codes\07\7.4\FatPo\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的两行粗体字代码用于控制动画的播放和停止。运行该程序，单击程序界面中的“播放”按钮，将可以看到“功夫熊猫”开始表演，如图7.11所示。

图7.11 逐帧动画

实例：在指定点爆炸

前面介绍“雷电”飞机游戏时已经提到，当敌机与用户自己的飞机碰撞，或者自己的飞机发射的子弹与敌机碰撞时，都应该在碰撞点播放“飞机爆炸”的动画。

爆炸效果实际上是一个逐帧动画，开发者需要收集从开始爆炸到爆炸结束的所有静态图片，再将这些图片定义成一个逐帧动画，接着在碰撞点播放该逐帧动画即可。

本实例为了突出此处的“主题”—在指定点爆炸，并未增加飞机控制这些细节，只是简单地检测触摸屏事件，当用户触摸屏幕时，程序将会在触碰点“爆炸”。

该程序先使用如下代码来定义爆炸的逐帧动画资源。

程序清单：codes\07\7.4\Blast\res\anim\blast.xml

接下来就可以利用上面的blast.xml所定义的动画资源了—当程序检测到用户触摸屏幕时，程序就在该触摸点播放该动画资源。程序如下。

程序清单：
codes\07\7.4\Blast\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的第一段粗体字代码就是触摸屏事件的响应代码，该程序把MyView（ImageView的子类）实例移动到触摸事件的发生点，并播放动画，这就实现了在指定点爆炸的效果。

需要指出的是，程序中①号粗体字代码所定义的方法对于该程序不是必要的—即使删除该方法程序也完全正常，但这只是恰好因为爆炸效果的最后一帧是空白。换一种情况，如果爆炸动画的最后一帧不是空白，而程序又没有控制隐藏播放动画的ImageView，用户将会看到动画结束了，但动画效果依然残留在那里。为了解决这个问题，就可借助于①号粗体字代码所定义的方法，它会自动检测动画播放到最后一帧时隐藏该ImageView。

7.5 补间（Tween）动画

Android除了支持逐帧动画之外，也提供了对补间（Tween）动画的支持。补间动画就是指开发者只需指定动画开始、动画结束等“关键帧”，而动画变化的“中间帧”由系统计算并补齐。这也是笔者把Tween动画翻译为“补间动画”的原因。

7.5.1 Tween动画与Interpolator

有过Flash设计经验的人应该对补间动画有很好的概念。就笔者的经验来看，Flash的补间动画支持比Android的更简单、功能更强。对于没有Flash设计经验的读者，图7.12可作为补间动画的示意图。

图7.12 补间动画的示意图

从图7.12可以看出，对于补间动画而言，开发者无须“逐一”定义动画过程中的每一帧，他只要定义动画开始、结束的关键帧，并指定动画的持续时间即可。

从图7.12可以看出，补间动画所定义的开始帧、结束帧其实只是一些简单的变化，比如图形大小的缩放、旋转角度的改变等。Android使用Animation代表抽象的动画类，它包括如下几个子类。

➢ **AlphaAnimation**：透明度改变的动画。创建该动画时要指定动画开始时的透明度、结束时的透明度和动画持续时间。其中透明度可从0变化到1。

➢ **ScaleAnimation**：大小缩放的动画。创建该动画时要指定动画开始时的缩放比（以X、Y轴的缩放参数来表示）、结束时的缩放比（以X、Y轴的缩放参数来表示），并指定动画持续时间。由于缩放时以不同点为中心的缩放效果并不相同，因此指定缩放动画时还要通过pivotX、pivotY来指定“缩放中心”的坐标。

➢ **TranslateAnimation**：位移变化的动画。创建该动画时只要指定动画开始时的位置（以X、Y坐标来表示）、结束时的位置（以X、Y坐标来表示），并指定动画持续时间即可。

➢ **RotateAnimation**：旋转动画。创建该动画时只要指定动画开始时的旋转角度、结束时的旋转角度，并指定动画持续时间即可。由于

旋转时以不同点为中心的旋转效果并不相同，因此指定旋转动画时还要通过pivotX、pivotY来指定“旋转轴心”的坐标。

一旦为补间动画指定了三个必要信息，Android就会根据动画的开始帧、结束帧、动画持续时间计算出需要在中间“补入”多少帧，并计算所有补入帧的图形。当用户浏览补间动画时，他眼中看到的依然是“逐帧动画”。

为了控制在动画期间需要动态“补入”多少帧，具体在动画运行的哪些时刻补入帧，需要借助于Interpolator。

提示：

Interpolator在笔者以前念大学时看到有些资料将它翻译为“插值”，这个翻译基本还可以：补间动画定义的是动画开始、结束的关键帧，Android需要在开始帧、结束帧之间动态地计算、插入大量的帧，而Interpolator用于控制“插入帧”的行为，因此翻译为插值也是合适的；现在也有些资料将Interpolator翻译得比较生僻，难以理解。为避免读者对各种翻译感到疑惑，本书后面笔者一律使用英文单词Interpolator，不会强行将它翻译成中文。

Interpolator根据特定算法计算出整个动画所需要动态插入帧的密度和位置。简单地说，Interpolator负责控制动画的变化速度，这就使得基本的动画效果（Alpha、Scale、Translate、Rotate）能以匀速变化、加速、减速、抛物线速度等各种速度变化。

Interpolator是一个接口，它定义了所有Interpolator都需要实现的方法：float getInterpolation（float input），开发者完全可以通过实现Interpolator来控制动画的变化速度。

Android为Interpolator提供了如下几个实现类，分别用于实现不同的动画变化速度。

➢ **LinearInterpolator**：动画以均匀的速度改变。

➢ **AccelerateInterpolator**：在动画开始的地方改变速度较慢，然后开始加速。

➢ **AccelerateDecelerateInterpolator**：在动画开始、结束的地方改变速度较慢，在中间的时候加速。

➢ **CycleInterpolator**：动画循环播放特定的次数，变化速度按正弦曲线改变。

➢ **DecelerateInterpolator**：在动画开始的地方改变速度较快，然后开始减速。

为了在动画资源文件中指定补间动画所使用的Interpolator，定义补间动画的<set.../>元素支持一个android：interpolator属性，该属性的属性值可指定为Android默认支持的Interpolator。例如：

➢**@android:anim/linear_interpolator**

➢**@android:anim/accelerate_interpolator**

➢**@android:anim/accelerate_decelerate_interpolator**

➢……

其实上面的写法很有规律，它们就是把系统提供的Interpolator实现类的类名的驼峰写法改为下画线写法即可。

一旦在程序中通过AnimationUtils得到了代表补间动画的Animation之后，接下来就可调用View的startAnimation（Animation anim）方法开始对该View执行动画了。

7.5.2 位置、大小、旋转度、透明度改变的补间动画

虽然Android允许在程序中创建Animation对象，但实际上一般都会采用动画资源文件来定义补间动画。前面已经介绍过定义补间动画资源

文件的格式，此处不再赘述。

下面以一个示例来介绍补间动画。该示例包括两个动画资源文件，其中第一个动画资源文件控制图片以旋转的方式缩小。该动画资源文件如下。

程序清单：
codes\07\7.5\TweenAnim\app\src\main\res\anim\anim.xml

上面的动画资源指定动画匀速变化，同时进行缩放、透明度、旋转三种改变，动画持续时间为3秒。

第二个动画资源文件则控制图片以动画的方式恢复回来。该动画资源文件如下。

程序清单：
codes\07\7.5\TweenAnim\app\src\main\res\anim\reverse.xml

定义动画资源之后，接下来就可以利用AnimationUtils工具类来加载指定的动画资源了，加载成功后会返回一个Animation，该对象即可控制图片或视图播放动画。

下面的程序将会负责加载动画资源，并使用Animation来控制图片播放动画。

程序清单：
codes\07\7.5\TweenAnim\app\src\main\java\org\crazyit\image\MainActivity.java

正如上面两行粗体字代码所看到的，当用户单击程序中的指定按钮时，程序对flower图片播放第一个动画；程序使用定时器设置3.5秒后对flower图片播放第二个动画。运行该程序，单击按钮将看到程序中间的图片先旋转着缩小、变淡，然后旋转着放大，透明度也逐渐恢复正常，如图7.13所示。

实例：蝴蝶飞舞

图7.13 补间动画

很多实际的动画往往同时运行两个动画，比如我们要做一个小游戏，需要让用户控制游戏中的主角移动—当主角移动时，不仅要控制它的位置改变，还应该在它移动时播放逐帧动画来让用户感觉更“逼真”。

本实例将会结合逐帧动画和补间动画来开发一个“蝴蝶飞舞”的效果。在这个实例中，蝴蝶飞行时的振翅效果是逐帧动画；蝴蝶飞行时的位置改变是补间动画。

先为该实例定义如下动画资源。

程序清单：
codes\07\7.5\Butterfly\app\src\main\res\anim\butterfly.xml

定义了上面逐帧动画的动画资源后，接下来在程序中使用一个ImageView显示该动画资源，即可看到蝴蝶“振翅”的效果了。由于蝴蝶飞舞主要是位移改变，接下来可以在程序中通过TranslateAnimation

以动画的方式改变ImageView的位置，这样就可实现“蝴蝶飞舞”的效果了。程序如下。

程序清单：
codes\07\7.5\butterfly\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的①号粗体字代码位于Handler的消息处理方法内，这样程序每隔0.2秒即对该ImageView执行一次位移动画；程序的②号粗体字代码则用于播放butterfly动画（蝴蝶振翅效果）。运行上面的程序，单击蝴蝶，即可看到屏幕上有只蝴蝶从左向右飞舞，如图7.14所示。

图7.14 蝴蝶飞舞

7.5.3 自定义补间动画

Android 提供了 Animation 作为补间动画抽象基类，并且为该抽象基类提供了AlphaAnimation、RotateAnimation、ScaleAnimation、TranslateAnimation 四个实现类，这四个实现类只是补间动画的四种基本形式：透明度改变、旋转、缩放、位移。在实际项目中可能还需要一些更复杂的动画，比如让图片在“三维”空间内进行旋转动画等，这就需要开发者自己开发补间动画了。

自定义补间动画并不难，需要继承Animation，继承Animation时关键是要重写该抽象基类的applyTransformation（float interpolatedTime，Transformation t）方法，该方法中两个参数说明如下。

➢ **interpolatedTime**：代表了动画的时间进行比。不管动画实际的持续时间如何，当动画播放时，该参数总是自动从0变化到1。

➢ **Transformation**：代表了补间动画在不同时刻对图形或组件的变形程度。

从上面的介绍可以看出，实现自定义动画的关键就在于重写applyTransformation（）方法时，根据interpolatedTime时间来动态地计算动画对图片或视图的变形程度。

Transformation代表了对图片或视图的变形程度，该对象里封装了一个Matrix对象，对它所包装的Matrix进行位移、倾斜、旋转等变换时，Transformation将会控制对应的图片或视图进行相应的变换。

为了控制图片或视图进行三维空间的变换，还需要借助于Android提供的一个Camera，这个Camera并非代表手机的摄像头，而是一个空间变换工具，作用有点类似于Matrix，但功能更强大。

Camera提供了如下常用的方法。

➢ **getMatrix（Matrix matrix）**：将Camera所做的变换应用到指定matrix上。

➢ **rotateX（float deg）**：使目标组件沿X轴旋转。

➢ **rotateY（float deg）**：使目标组件沿Y轴旋转。

➢ **rotateZ（float deg）**：使目标组件沿Z轴旋转。

➢ **translate（float x，float y，float z）**：使目标组件在三维空间里进行位移变换。

➢ **applyToCanvas（Canvas canvas）**：把Camera所做的变换应用到Canvas上。

从上面的方法可以看出，Camera主要用于支持三维空间的变换，那么手机中三维空间的坐标系统是怎样的呢？图7.15显示了手机屏幕上的三维坐标系统。

图7.15 手机屏幕上的三维坐标系统

当Camera控制图片或View沿X、Y或Z轴旋转时，被旋转的图片或View将会呈现出三维透视的效果。图7.16显示了一张图片沿着Y轴旋转的效果。

图7.16 在三维空间内沿Y轴旋转的示意图

提示：

关于三维透视理论的知识，此处限于篇幅不便深入解释，如果读者有3ds Max或Maya之类的设计经验，应该很容易理解三维透视的基本理论；否则建议读者自行阅读相关内容。

下面程序将会利用Camera来自定义在三维空间的动画，该程序的自定义动画类的代码如下。

程序清单：
\codes\07\7.5\ListViewTween\app\src\main\java\org\crazyit\image\MyAnimation.java

上面的程序中自定义动画的关键就是两行粗体字代码，这两行代码将会根据动画的进行时间来控制View在X、Y轴上的旋转—这就会具有在三维空间内旋转的效果。

提供了上面的自定义动画类之后，接下来既可用该自定义动画类来控制图片，也可控制View组件，因为动画就是通过调用View的startAnimation（Animation anim）来启动的。下面是使用MyAnimation执行动画的程序代码。

程序清单：
codes\07\7.5\ListViewTween\app\src\main\java\org\crazyit\image\MainActivity.java

上面程序中的粗体字代码对ListView使用MyAnimation播放动画，这意味着程序中ListView将会以三维变换的动画方式出现，如图7.17所示。

图7.17 三维空间变换的动画

7.6 属性动画

前面介绍Android资源时已经提到了属性动画，从某种角度来看，属性动画是增强版的补间动画，属性动画的强大可以体现在如下两方面。

➢ 补间动画只能定义两个关键帧在“透明度”、“旋转”、“缩放”、“位移”4 个方面的变化，但属性动画可以定义任何属性的变化。

➢ 补间动画只能对UI组件执行动画，但属性动画几乎可以对任何对象执行动画（不管它是否显示在屏幕上）。

与补间动画类似的是，属性动画也需要定义如下几个属性。

➢ 动画持续时间。该属性的默认值是300 ms。在属性动画资源文件中通过android：duration属性指定。

➢ 动画插值方式。该属性的作用与补间动画中插值属性的作用基本类似。在属性动画资源文件中通过android：interpolator属性指定。

➢ 动画重复次数。指定动画重复播放的次数。在属性动画资源文件中通过android：repeatCount属性指定。

➢ 重复行为。指定动画播放结束后、重复下次动画时，是从开始帧再次播放到结束帧，还是从结束帧反向播放到开始帧。在属性动画资源文件中通过android：repeatMode属性指定。

➢ 动画集。开发者可以将多个属性动画合并成一组，既可让这组属性动画按次序播放，也可让这组属性动画同时播放。在属性动画资源文件中通过<set..../>元素来组合，该元素的android：ordering属性指定该组动画是按次序播放，还是同时播放。

➢ 帧刷新频率。指定每隔多长时间播放一帧。该属性的默认值为10 ms。

7.6.1 属性动画的API

属性动画涉及的API如下。

➢ **Animator**：它提供了创建属性动画的基类，基本上不会直接使用该类。通常该类只用于被继承并重写它的相关方法。

➢ **ValueAnimator**：属性动画主要的时间引擎，它负责计算各个帧的属性值。它定义了属性动画的绝大部分核心功能，包括计算各帧的相关属性值，负责处理更新事件，按属性值的类型控制计算规则。属性动画主要由两方面组成：①计算各帧的相关属性值；②为指定对象设置这些计算后的值。ValueAnimator只负责第一方面内容，因此程序员必须根据ValueAnimator计算并监听值更新来更新对象的相关属性值。

➢ **ObjectAnimator**：它是ValueAnimator的子类，允许程序员对指定对象的属性执行动画。在实际应用中，ObjectAnimator 使用起来更加简单，因此更加常用。在少数场景下，由于ObjectAnimator存在一些限制，可能需要考虑使用ValueAnimator。

➢ **AnimatorSet**：它是Animator的子类，用于组合多个Animator，并指定多个Animator是按次序播放，还是同时播放。

除此之外，属性动画还需要利用一个Evaluator（计算器），该工具类控制属性动画如何计算属性值。Android提供了如下Evaluator。

➢ **IntEvaluator**：用于计算int类型属性值的计算器。

➢ **FloatEvaluator**：用于计算float类型属性值的计算器。

➢ **ArgbEvaluator**：用于计算以十六进制形式表示的颜色值的计算器。

➢ **TypeEvaluator**：它是计算器接口，开发者可以通过实现该接口来实现自定义计算器。如果需要对 int、float 或者颜色值以外类型的属性执行属性动画，可能需要实现TypeEvaluator接口来实现自定义计算器。

1.使用ValueAnimator创建动画

使用ValueAnimator创建动画可按如下4个步骤进行。

1 调用ValueAnimator的ofInt（）、ofFloat（）或ofObject（）静态方法创建ValueAnimator实例。

2 调用ValueAnimator的setXxx（）方法设置动画持续时间、插值方式、重复次数等。

3 调用ValueAnimator的start（）方法启动动画。

4 为ValueAnimator注册AnimatorUpdateListener监听器，在该监听器中可以监听ValueAnimator计算出来的值的改变，并将这些值应用到指定对象上。

例如如下代码片段：

上面的例子实现了在1000ms内，值从0到1的变化。

除此之外，开发者也可以提供一个自定义的Evaluator计算器，例如如下代码：

在上面的代码片段中，ValueAnimator仅仅是计算动画过程中变化的值，并没有把这些计算出来的值应用到任何对象上，因此也不会显示任何动画。

如果希望使用ValueAnimator创建动画，还需要注册一个监听器：AnimatorUpdateListener，该监听器负责更新对象的属性值。在实现这个监听器时，可以通过getAnimatedValue（）方法来获取当前帧的值，并将该计算出来的值应用到指定对象上。当该对象的属性持续改变时，该对象也就呈现出动画效果了。

2.使用ObjectAnimator创建动画

ObjectAnimator继承了ValueAnimator，因此它可以直接将ValueAnimator在动画过程中计算出来的值应用到指定对象的指定属性上（ValueAnimator则需要注册一个监听器来完成这个工作）。因此使用ObjectAnimator就不需要注册AnimatorUpdateListener监听器了。

使用ObjectAnimator的ofInt（）、ofFloat（）或ofObject（）静态方法创建ObjectAnimator时，需要指定具体的对象，以及对象的属性名。

例如如下代码片段：

与ValueAnimator不同的是，使用ObjectAnimator有如下几个注意点。

➢ 要为该对象对应的属性提供 setter 方法，如上例中需要为 foo 对象提供 setAlpha（float value）方法。

➢ 调用ObjectAnimator的ofInt（）、ofFloat（）或ofObject（）工厂方法时，如果values...参数只提供了一个值（本来需要提供开始值和结束值），那么该值会被认为是结束值。该对象应该为该属性提供一个getter方法，该getter方法的返回值将被作为开始值。

➢ 如果动画的对象是 View，为了能显示动画效果，可能还需要在 onAnimationUpdate（）事件监听方法中调用 View.invalidate（）方法来刷新屏幕的显示，比如对 Drawable 对象的color属性执行动画。但View定义的setter方法，如 setAlpha（）和setTranslationX（）等方法，都会自动地调用invalidate（）方法，因此不需要额外地调用invalidate（）方法。

7.6.2 使用属性动画

属性动画既可作用于UI组件，也可作用于普通的对象（即使它没有在UI界面上绘制出来）。

定义属性动画有如下两种方式。

➢ 使用ValueAnimator或ObjectAnimator的静态工厂方法来创建动画。

➢ 使用资源文件来定义动画。

使用属性动画的步骤如下。

1 创建ValueAnimator或ObjectAnimator对象—既可从XML资源文件加载该动画资源，也可直接调用ValueAnimator或ObjectAnimator的静态工厂方法来创建动画。

2 根据需要为Animator对象设置属性。

3 如果需要监听Animator的动画开始事件、动画结束事件、动画重复事件、动画值改变事件，并根据事件提供相应的处理代码，则应该为Animator对象设置事件监听器。

4 如果有多个动画需要按次序或同时播放，则应使用AnimatorSet组合这些动画。

5 调用Animator对象的start（）方法启动动画。

下面的示例示范了如何利用属性动画来控制“小球”掉落动画。该示例会监听用户在屏幕上的“触屏”时间，程序会在屏幕的触摸点绘制一个小球，并用动画控制该小球向下掉落。

该示例的界面布局文件非常简单，界面布局文件中只有一个LinearLayout，因此此处不再给出界面布局文件。下面是该示例的Activity代码。

程序清单：
codes\07\7.6\AnimatorTest\app\src\main\java\org\crazyit\image\MainActivity.java

上面的程序中第一段粗体字代码创建了两个ObjectAnimator对象，其中第一个ObjectAnimator控制小球的“下落”；第二个ObjectAnimator则用于控制该小球以“渐隐”的方式隐藏—程序为fadeAnim绑定了事件监听器，当fadeAnim动画播放完成时，程序将小球删除。

上面程序的属性动画并没有控制 UI组件，而是对一个自定义ShapeHolder对象起作用。为了能让UI界面不断刷新，看到UI界面上小球的动画效果，程序为两个动画都绑定了AnimatorUpdateListener监听器，该监听器用于实现当目标对象（小球）的属性发生改变时，该组件会重绘界面—如以上程序中①号代码所示。

上面的程序中第三段粗体字代码用于创建一个圆形渐变，创建圆形渐变的起始颜色的red、green、blue值都是随机的，结束颜色值是起始颜色值的四分之一。读者可能对计算颜色值的代码有点疑惑：

上面的代码其实很简单，解释如下。

➢ 0xff000000代表透明度为ff，也就是完全不透明。

➢ red代表一个0～255（0～ff）的随机整数，但这个整数要添加到0xff**00**0000中加粗的两个“位”上，也就是要将red的值左移（16位，对应为十六进制数的4位），这就是red<<16的原因。

➢ green代表一个0～255（0～ff）的随机整数，但这个整数要添加到0xff00**00**00中加粗的两个“位”上，也就是要将green的值左移（8位，对应为十六进制数的2位），这就是red<<8的原因。

➢ blue代表一个0～255（0～ff）的随机整数，但这个整数要添加到0xff0000**00**中加粗的两个“位”上，因此blue值就不需要位移了。

将red、green、blue值位移后的结果加起来就得到了实际的颜色值，但为了有更好的计算性能，本代码直接使用按位或（|）来累加这些值。

上面的示例还用到了一个ShapeHolder类，该类只是负责包装ShapeDrawable对象，并为x、y、width、height、alpha等属性提供setter、getter方法，以方便ObjectAnimator动画控制它。下面是ShapeHolder类的代码。

运行该程序，用户在屏幕上拖动手指时，将可以看到如图7.18所示的小球下落的动画。

如果为小球增加更多的动画，让小球落到底端时产生弹跳动画，并且再次弹起来，则可以使该示例更加完善。

图7.18 小球下落

实例：大珠小珠落玉盘

该实例是对上一个示例的改进，主要是为小球增加了几个动画，控制小球落到底端时小球被压扁，小球会再次弹起，这样就可以开发出“大珠小珠落玉盘”的弹跳动画。

该实例的Activity代码如下。

程序清单：
codes\07\7.6\BouncingBalls\app\src\main\java\org\crazyit\image\MainActivity.java

上面的实例对小球使用了更多的动画，因此小球下落后，会再次弹起。

该实例的ShapeHolder需要对width、height增加动画，因此该ShapeHolder需要增加对width、height的setter和getter方法。下面是该实例中ShapeHolder的代码。

程序清单：
codes\07\7.6\BouncingBalls\app\src\main\java\org\crazyit\image\ShapeHolder.java

运行该实例，可以看到小球在底端压扁并弹起的动画，如图7.19所示。

7.7 使用SurfaceView实现动画

图7.19 大珠小珠落玉盘

虽然前面大量介绍了使用自定义View来进行绘图，但View的绘图机制存在如下缺陷。

➢ View缺乏双缓冲机制。

➢ 当程序需要更新View上的图片时，程序必须重绘View上显示的整张图片。

➢ 新线程无法直接更新View组件。

由于View存在上述缺陷，所以通过自定义View来实现绘图，尤其是游戏中的绘图时性能并不好。Android提供了一个SurfaceView来代替View，在实现游戏绘图方面，SurfaceView比View更加出色，因此一般推荐使用SurfaceView。

7.7.1 SurfaceView的绘图机制

SurfaceView一般会与SurfaceHolder结合使用，SurfaceHolder用于向与之关联的SurfaceView上绘图，调用SurfaceView的getHolder（）方法即可获取SurfaceView关联的SurfaceHolder。

SurfaceHolder提供了如下方法来获取Canvas对象。

➢ **Canvas lockCanvas（）：** 锁定整个SurfaceView对象，获取该SurfaceView上的Canvas。

➢ **Canvas lockCanvas（Rect dirty）：** 锁定 SurfaceView 上 Rect 划分的区域，获取该SurfaceView上的Canvas。

当对同一个SurfaceView调用上面两个方法时，两个方法所返回的是同一个Canvas对象。但当程序调用第二个方法获取指定区域的Canvas时，SurfaceView将只对Rect所“圈”出来的区域进行更新，通过这种方式可以提高画面的更新速度。

当通过lockCanvas（）获取了指定SurfaceView上的Canvas之后，接下来程序就可以调用Canvas进行绘图了，Canvas绘图完成后通过如下方法来释放绘图、提交所绘制的图形。

➢ **unlockCanvasAndPost(canvas)**

需要指出的是，当调用SurfaceHolder的unlockCanvasAndPost（）方法之后，该方法之前所绘制的图形还处于缓冲区中，下一次lockCanvas（）方法锁定的区域可能会“遮挡”它。

下面的程序示范了SurfaceView的绘图机制。该程序将会采用SurfaceView绘制一个游戏动画：程序通过继承SurfaceView来实现一个自定义View。下面是该自定义View的代码。

程序清单：
codes\07\7.7\SurfaceViewTest\app\src\main\java\org\crazyit\image\MainActivity.java

上面的程序使用FishView自身作为SurfaceHolder的Callback实例，并实现了Callback中定义的如下三个方法。

➢ **void surfaceChanged（SurfaceHolder holder，int format，int width，int height）**：当一个SurfaceView的格式或大小发生改变时回调该方法。

➢ **void surfaceCreated（SurfaceHolder holder）**：当SurfaceView被创建时回调该方法。

➢ **void surfaceDestroyed（SurfaceHolder holder）**：当SurfaceView将要被销毁时回调该方法。

上面FishView所实现的surfaceCreated（）方法中调用了该类的resume（）方法，该方法将会启动一个负责更新游戏图像的线程，这样即可保证FishView上的动画不停地改变。当SurfaceView被销毁时，程序会回调它的surfaceDestroyed（）方法，FishView所实现的该方法则负责停止更新游戏图像的线程，这样即可释放线程资源。

图7.20 SurfaceView的绘图机制

FishView中定义了一个UpdateViewThread线程，这个线程可通过SurfaceHolder来绘制SurfaceView上绘制的图像（这也是SurfaceView与普通View的显著区别之一）。程序中①号粗体字代码用于锁定SurfaceView，准备开始绘制，接下来程序即可在SurfaceView上执行任意绘制。程序对SurfaceView绘制完成之后，程序中②号粗体字代码解锁画布，并将绘制内容显示出来。

该程序的界面布局文件只是简单地显示该FishView，主程序也很简单，此处不再给出。运行该程序，即可看到如图7.20所示的"小鱼游动"的界面。

实例：基于SurfaceView开发示波器

SurfaceView与普通View还有一个重要的区别：View的绘图必须在当前UI线程中进行—这也是前面程序需要更新View组件时总要采用Handler处理的原因；但SurfaceView就不会存在这个问题，因此SurfaceView的绘图是由SurfaceHolder来完成的。

对于View组件，如果程序需要花较长的时间来更新绘图，那么主UI线程将会被阻塞，无法响应用户的任何动作；而SurfaceHolder则会启用新的线程去更新SurfaceView的绘制，因此不会阻塞主UI线程。

一般来说，如果程序或游戏界面的动画元素较多，而且很多动画元素的移动都需要通过定时器来控制，就可以考虑使用SurfaceView，而不是View。例如，下面我们使用SurfaceView开发一个示波器程序，该程序将会根据用户单击的按钮在屏幕上自动绘制正弦波或余弦波。

该程序的界面布局很简单，包含两个按钮和一个SurfaceView，程序每次绘制时只需要绘制（更新）当前点的波形，前面已经绘制的波形无须更新，这就利用了SurfaceHolder的lockCanvas（Rect r）方法。

该程序的代码如下。

程序清单：
codes\07\7.7\ShowWave\app\src\main\java\org\crazyit\image\MainActivity.java

从上面程序中的三行粗体字代码可以看出，当程序每次绘制正弦波、余弦波上的当前点时，程序都无须重绘整个画面，SurfaceHolder只要锁定当前绘制点的小范围即可，系统更新画面时也只要更新这个范围即可，因此具有较好的画面性能。运行该程序，将看到如图7.21所示的结果。

图7.21 示波器

7.8 本章小结

本章主要介绍了Android的图形、图像处理，这种图形、图像处理不仅对Android界面开发十分重要，而且也是开发Android 2D游戏的基础。学习本章需要重点掌握Android丰富的绘图API，包括Canvas、Paint、Path等类。除此之外，读者还需要掌握Android绘图的双缓冲机制，以及利用Matrix对图形进行几何变换等内容。除此之外，Android提供了逐帧动画、补间动画、属性动画支持，尤其需要重点掌握属性动画。为了更好地开发游戏动画界面，Android专门提供了SurfaceView，本章详细介绍了SurfaceView的绘图机制，并讲解了如何继承SurfaceView来开发动画。

第8章 Android数据存储与IO

本章要点

SharedPreferences的概念和作用

使用SharedPreferences保存程序的参数、选项

读写其他应用的SharedPreferences

Android的文件IO

读写SD卡上的文件

了解SQLite数据库

使用Android的API操作SQLite数据库

使用sqlite3工具管理SQLite数据库

SQLiteOpenHelper类的功能和用法

Android的手势支持

手势检测

向手势库中添加手势

识别用户手势

所有应用程序都必然涉及数据的输入、输出，Android应用也不例外，应用程序的参数设置、程序运行状态数据这些都需要保存到外部存储器上，这样系统关机之后数据才不会丢失。Android应用是使用Java语言来开发的，因此开发者在Java IO中的编程经验大部分都可“移植”到

Android应用开发上。Android系统还提供了一些专门的IO API，通过这些API可以更有效地进行输入、输出。

如果应用程序只有少量数据需要保存，那么使用普通文件就可以了；但如果应用程序有大量数据需要存储、访问，就需要借助于数据库了。Android系统内置了SQLite数据库，SQLite数据库是一个真正轻量级的数据库，它没有后台进程，整个数据库就对应于一个文件，这样可以非常方便地在不同设备之间移植。Android不仅内置了SQLite数据库，而且为访问SQLite数据库提供了大量便捷的API。本章将会详细介绍如何在Android应用中使用SQLite数据库。

8.1 使用SharedPreferences

有些时候，应用程序有少量的数据需要保存，而且这些数据的格式很简单，都是普通的字符串、标量类型的值等，比如应用程序的各种配置信息（如是否打开音效、是否使用振动效果等）、小游戏的玩家积分（如扫雷英雄榜之类的）等，对于这种数据，Android提供了SharedPreferences进行保存。

8.1.1 SharedPreferences与Editor简介

SharedPreferences保存的数据主要是类似于配置信息格式的数据，因此它保存的数据主要是简单类型的key-value对。

SharedPreferences接口主要负责读取应用程序的Preferences数据，它提供了如下常用方法来访问SharedPreferences中的key-value对。

➢ **boolean contains（String key）**：判断SharedPreferences是否包含特定key的数据。

➢ **abstract Map<String，？>getAll（）**：获取SharedPreferences数据里全部的key-value对。

➢ **boolean getXxx（String key，xxx defValue）：**获取SharedPreferences数据里指定key对应的value。如果该key不存在，则返回默认值defValue。其中xxx可以是boolean、float、int、long、String等各种基本类型的值。

SharedPreferences接口本身并没有提供写入数据的能力，而是通过SharedPreferences的内部接口，SharedPreferences调用edit（）方法即可获取它所对应的Editor对象。Editor提供了如下方法来向SharedPreferences写入数据。

➢ **SharedPreferences.Editor clear（）：**清空SharedPreferences里所有数据。

➢ **SharedPreferences.Editor putXxx（String key，xxx value）：**向SharedPreferences存入指定key对应的数据。其中xxx可以是boolean、float、int、long、String等各种基本类型的值。

➢ **SharedPreferences.Editor remove（String key）：**删除SharedPreferences里指定key对应的数据项。

➢ **boolean commit（）：**当Editor编辑完成后，调用该方法提交修改。

提示：

从用法角度来看，SharedPreferences和SharedPreferences.Editor组合起来非常像Map，其中SharedPreferences负责根据key读取数据，而SharedPreferences.Editor则用于写入数据。

SharedPreferences本身是一个接口，程序无法直接创建SharedPreferences实例，只能通过Context提供的getSharedPreferences（String name，int mode）方法来获取SharedPreferences实例，该方法的第二个参数支持如下几个值。

➢ **Context.MODE_PRIVATE**：指定该SharedPreferences数据只能被本应用程序读写。

➢ **Context.MODE_WORLD_READABLE**：指定该SharedPreferences 数据能被其他应用程序读，但不能写。

➢ **Context.MODE_WORLD_WRITEABLE**：指定该SharedPreferences数据能被其他应用程序读写。

提示：

从Android 4.2开始，Android不再推荐使用MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE这两种模式，因为这两种模式允许其他应用程序来读或写本应用创建的数据，因此容易导致安全漏洞。如果应用程序确实需要把内部数据暴露出来供其他应用访问，则应该使用本书后面介绍的ContentProvider。

下面介绍对SharedPreferences的简单读写。

8.1.2 SharedPreferences的存储位置和格式

下面的程序示范了如何向SharedPreferences中写入、读取数据。该程序的界面很简单，它只是提供了两个按钮，其中一个用于写入数据；另一个用于读取数据，故此处不再给出界面布局文件。程序代码如下。

程序清单：
codes\08\8.1\SharedPreferencesTest\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一段粗体字代码用于读取SharedPreferences数据，当程序所读取的SharedPreferences数据文件根本不存在时，程序也返回

默认值，并不会抛出异常；第二段粗体字代码用于写入SharedPreferences数据，由于SharedPreferences并不支持写入Date类型的值，故程序使用了SimpleDateFormat将Date格式化成字符串后写入。

运行上面的程序，单击程序中“写入数据”按钮，程序将完成SharedPreferences写入，写入完成后打开DDMS的File Explorer面板（参考第1章介绍的方法来打开DDMS的File Explorer面板），然后展开文件浏览树，将看到如图8.1所示的窗口。

图8.1 SharedPreferences的存储路径

从图8.1可以看出，SharedPreferences数据总是保存在/data/data/<package name>/shared_prefs目录下，SharedPreferences数据总是以XML格式保存。通过File Explorer面板的导出文件按钮导出该XML文档，打开该XML文档可以看到如下文件内容：

从上面的文件不难看出，SharedPreferences数据文件的根元素是<map.../>元素，该元素里每个子元素代表一个key-value对，当value是整数类型时，使用<int.../>子元素；当value是字符串类型时，使用<string.../>子元素……依此类推。

图8.2 读取SharedPreferences数据

单击程序界面中的“读取数据”按钮，程序弹出一个Toast对话框显示上次写入的数据，如图8.2所示。

实例：记录应用程序的使用次数

这个简单的实例可以记录应用程序的使用次数：当用户第一次启动该应用程序时，系统创建SharedPreferences来记录使用次数。用户以后启动应用程序时，系统先读取SharedPreferences中记录的使用次数，然后将使用次数加1。

本实例程序的代码如下。

程序清单：
codes\08\8.1\UseCount\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一行粗体字代码用于读取SharedPreferences中记录的使用次数；第二行粗体字代码将使用次数增加1，并再次将使用次数写入SharedPreferences中。

8.2 File存储

读者学习Java SE的时候都知道Java提供了一套完整的IO流体系，包括FileInputStream、FileOutputStream等，通过这些IO流可以非常方便地访问磁盘上的文件内容。Android同样支持以这种方式来访问手机存储器上的文件。

8.2.1 openFileOutput和openFileInput

Context提供了如下两个方法来打开应用程序的数据文件夹里的文件IO流。

➢ **FileInputStream openFileInput（String name）**：打开应用程序的数据文件夹下的name文件对应的输入流。

➢ **FileOutputStream openFileOutput（String name，int mode）**：打开应用程序的数据文件夹下的name文件对应的输出流。

上面两个方法分别用于打开文件输入流、输出流，其中第二个方法的第二个参数指定打开文件的模式，该模式支持如下值。

➢ **MODE_PRIVATE**：该文件只能被当前程序读写。

➢ **MODE_APPEND**：以追加方式打开该文件，应用程序可以向该文件中追加内容。

➢ **MODE_WORLD_READABLE**：该文件的内容可以被其他程序读取。

➢ **MODE_WORLD_WRITEABLE**：该文件的内容可由其他程序读写。

提示：

与前面介绍的类似，从Android 4.2开始，Android不推荐使用MODE_WORLD_READABLE、MODE_WORLD_WRITEABLE两种模式。

除此之外，Context还提供了如下几个方法来访问应用程序的数据文件夹。

➢ **getDir（String name，int mode）**：在应用程序的数据文件夹下获取或创建name对应的子目录。

➢ **File getFilesDir（）**：获取应用程序的数据文件夹的绝对路径。

➢ **String[] fileList（）**：返回应用程序的数据文件夹下的全部文件。

➢ **deleteFile（String）**：删除应用程序的数据文件夹下的指定文件。

下面的程序简单示范了如何读写应用程序数据文件夹内的文件。该程序的界面布局同样很简单，只包含两个文本框和两个按钮，其中第一

组文本框和按钮用于处理写入，文本框用于接受用户输入，当用户单击“写入”按钮时，程序将会把数据写入文件；第二组文本框和按钮用于处理读取，当用户单击“读取”按钮时，该文本框显示文件中的数据。程序代码如下。

注意：

由于本书主要介绍Android应用开发的相关知识，因此Java IO支持的相关内容则不在本书介绍的范围之内，如果读者对Java IO还不熟悉，建议先阅读疯狂Java体系的《疯狂Java讲义》。

程序清单：
codes\08\8.2\FileTest\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一段粗体字代码用于读取应用程序的数据文件，第二段粗体字代码用于向应用程序的数据文件中追加内容。从上面的粗体字代码可以看出，当Android系统调用Context的openFileInput（）、openFileOutput（）打开文件输入流或输出流之后，接下来IO流的用法与Java SE中IO流的用法完全一样：直接用节点流读写也行，用包装流包装之后再处理也没问题。

当单击程序界面中的“写入”按钮时，用户在第一个文本框中输入的内容将会被保存到应用程序的数据文件中，打开File Explorer面板，可以看到如图8.3所示的界面。

从图8.3可以看出，应用程序的数据文件默认保存在/data/data/<package name>/files目录下。

图8.3 应用程序的数据文件

8.2.2 读写SD卡上的文件

当程序通过Context的openFileInput（）或openFileOutput（）来打开文件输入流、输出流时，程序所打开的都是应用程序的数据文件夹里的文件，这样所存储的文件大小可能比较有限一毕竟手机内置的存储空间是有限的。

为了更好地存取应用程序的大文件数据，应用程序需要读写SD卡上的文件。SD卡大大扩充了手机的存储能力。

读写SD上的文件请按如下步骤进行。

1 调用Environment的getExternalStorageState（）方法判断手机上是否插入了SD卡，并且应用程序具有读写SD卡的权限。例如使用如下代码：

2 调用Environment的getExternalStorageDirectory（）方法来获取外部存储器，也就是SD卡的目录。

3 使用FileInputStream、FileOutputStream、FileReader或FileWriter读写SD卡里的文件。

应用程序读写SD卡上的文件有如下两个注意点。

➢ 手机上应该已插入SD卡。对于模拟器来说，可通过mksdcard命令来创建虚拟存储卡。关于虚拟存储卡的管理请参考第1章。

➢ 为了读写SD卡上的数据，必须在应用程序的清单文件（AndroidManifest.xml）中添加读写SD卡的权限。例如如下配置：

下面的程序示范了如何读写SD卡上的文件。该程序的主界面与上一个程序的界面完全相同，只是该程序数据读写是基于SD卡的。该程序代码如下。

程序清单：
codes\08\8.2\SDCardTest\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一段粗体字代码用于读取SD卡中指定文件的内容；第二段粗体字代码则使用RandomAccessFile向SD卡中的指定文件追加内容—如果使用FileOutputStream向指定文件写入数据，FileOutputStream会把原有的文件内容清空，那就不是追加文件内容了。由此可见，当程序直接使用FileOutputStream进行输出时，比使用Context的openFileOutput（）方便。

图8.4 存入SD卡的数据

运行上面的程序，在第一个文本框内输入一些字符串，然后单击“写入”按钮，即可将数据写入底层SD卡中。打开File Explorer，即可看到如图8.4所示的界面。

提示：

对于Genymotion模拟器而言，它模拟的SD卡对应的路径为/mnt/shell/emulated/0。

需要指出的是，当开发者直接在Eclipse中运行Android应用程序时，Eclipse默认启动的模拟器是不带SD卡的。为了让Eclipse启动的模拟器带上SD卡，可以通过“Run As”→“Run Configurations”菜单项打开如图

8.5所示的对话框，通过该对话框下面的附加选项来使用SD卡（实际上就是在启动模拟器时增加-sdcard选项）。

提示：

如果开发者不想使用Environment.getExternalStorageDirectory（）这么复杂的语句来获取SD卡的路径，则完全可以使用/mnt/sdcard/路径代表SD卡的路径，然后程序通过判断/mnt/sdcard/路径是否存在就可知道手机是否已插入了SD卡。

图8.5 指定启动模拟器时使用SD卡

实例：SD卡文件浏览器

下面我们将会利用Java的File类开发一个SD卡文件浏览器，该程序直接使用/mnt/sdcard来访问系统的SD卡目录，然后通过File的listFiles（）方法来获取指定目录下的全部文件和文件夹。

当程序启动时，系统启动获取/mnt/sdcard/目录下的全部文件、文件夹，并使用ListView将它们显示出来；当用户单击ListView的指定列表项时，系统将会显示该列表项下的全部文件夹和文件。

该程序的界面布局文件如下。

程序清单：
codes\08\8.2\SDFileExplorer\app\src\main\res\layout\main.xml

该程序主要利用了File的listFiles（）来列出指定目录下的全部文件，程序代码如下。

程序清单：
codes\08\8.2\SDFileExplorer\app\src\main\java\org\crazyit\io\MainActivity.java

上面的程序中①号粗体字方法使用File[]数组来填充ListView，填充时程序会根据File[]数组里的数据元素（File对象）代表的是文件还是文件夹来选择使用文件图标或文件夹图标，

图8.6 SD卡文件浏览器

如该方法中粗体字代码所示。

当用户单击ListView中某个列表项时，程序会将用户单击的列表项所对应的文件夹当成currentParent处理，并再次调用inflateListView（File[] files）方法来列出当前文件夹下的所有文件。

运行上面的程序，将看到如图8.6所示的界面。

正如图8.6所示，该程序就像SD卡资源管理器一样可以非常方便地浏览SD卡里包含的全部文件。

提示：

从图8.6所示界面中可以看到/mnt/sdcard/abc目录下包含了haha、hehe两个目录和color.jpg文件。这就要求开发者的模拟器的SD卡里带有这些目录和文件，为了在SD卡里创建目录，可先运行adb shell命令来启动Android的Shell窗口—由于Android的内核就是Linux，因此可在该Shell窗口里执行ls、mkdir等常见的Linux命令。

8.3 SQLite数据库

Android系统集成了一个轻量级的数据库：SQLite，SQLite并不想成为像Oracle、MySQL那样的数据库。SQLite只是一个嵌入式的数据库引擎，专门适用于资源有限的设备（如手机、PDA等）上适量数据存取。

虽然SQLite支持绝大部分SQL 92语法，也允许开发者使用SQL语句操作数据库中的数据，但SQLite并不像Oracle、MySQL数据库那样需要安装、启动服务器进程，SQLite数据库只是一个文件。

提示：

从本质上来看，SQLite的操作方式只是一种更为便捷的文件操作。后面我们会看到，当应用程序创建或打开一个SQLite数据库时，其实只是打开一个文件准备读写，因此有人说SQLite有点像Microsoft的Access（实际上SQLite功能要强大得多）。可能有读者会问，如果实际项目中有大量数据需要读写，而且需要面临大量用户的并发存储怎么办呢？对于这种情况，本身就不应该把数据存放在手机的SQLite数据库里—毕竟手机还是手机，它的存储能力、计算能力都不足以让它充当服务器的角色。

8.3.1 SQLiteDatabase简介

Android 提供了 SQLiteDatabase 代表一个数据库（底层就是一个数据库文件），一旦应用程序获得了代表指定数据库的SQLiteDatabase 对象，接下来就可通过SQLiteDatabase对象来管理、操作数据库了。

SQLiteDatabase提供了如下静态方法来打开一个文件对应的数据库。

➢ **static SQLiteDatabase openDatabase（String path，SQLiteDatabase.CursorFactoryfactory，int flags）**：打开path文件所代表的SQLite数据库。

➢ **static SQLiteDatabase openOrCreateDatabase（File file，SQLiteDatabase.CursorFactory factory）**：打开或创建（如果不存在）file文件所代表的SQLite数据库。

➢ **static SQLiteDatabase openOrCreateDatabase（String path，SQLiteDatabase.CursorFactory factory）**：打开或创建（如果不存在）path文件所代表的SQLite数据库。

在程序中获取SQLiteDatabase对象之后，接下来就可调用SQLiteDatabase的如下方法来操作数据库了。

➢ **execSQL（String sql，Object[] bindArgs）**：执行带占位符的SQL语句。

➢ **execSQL（String sql）**：执行SQL语句。

➢ **insert（String table，String nullColumnHack，ContentValues values）**：向指定表中插入数据。

➢ **update（String table，ContentValues values，String whereClause，String[] whereArgs）**：更新指定表中的特定数据。

➢ **delete（String table，String whereClause，String[] whereArgs）**：删除指定表中的特定数据。

➢ **Cursor query（String table，String[] columns，String whereClause，String[] whereArgs，String groupBy，String having，String orderBy）**：对指定数据表执行查询。

➢ **Cursor query（String table，String[] columns，String whereClause，String[] whereArgs，String groupBy，String having，String orderBy，String limit）**：对指定数据表执行查询。limit参数控制最多查询几条记录（用于控制分页的参数）。

➢ **Cursor query（boolean distinct，String table，String[] columns，String whereClause，String[]whereArgs，String groupBy，String having，String orderBy，String limit）**：对指定数据表执行查询。其中第一个参数控制是否去除重复值。

➢ **rawQuery（String sql，String[] selectionArgs）**：执行带占位符的SQL查询。

➢ **beginTransaction（）**：开始事务。

➢ **endTransaction（）**：结束事务。

从上面的方法不难看出，其实SQLiteDatabase的作用有点类似于JDBC的Connection接口，但SQLiteDatabase提供的方法更多，比如insert、update、delete、query等方法，其实这些方法完全可通过执行SQL语句来完成，但Android考虑到部分开发者对SQL语法不熟悉，所以提供这些方法帮助开发者以更简单的方式来操作数据表中的数据。

上面的查询方法都是返回一个Cursor对象，Android中的Cursor类似于JDBC的ResultSet，Cursor同样提供了如下方法来移动查询结果的记录指针。

➢ **move（int offset）**：将记录指针向上或向下移动指定的行数。offset为正数就是向下移动；为负数就是向上移动。

➢ **boolean moveToFirst（）**：将记录指针移动到第一行，如果移动成功则返回true。

➢ **boolean moveToLast（）**：将记录指针移动到最后一行，如果移动成功则返回true。

➢ **boolean moveToNext（）**：将记录指针移动到下一行，如果移动成功则返回true。

➢ **boolean moveToPosition（int position）**：将记录指针移动到指定行，如果移动成功则返回true。

➢ **boolean moveToPrevious（）**：将记录指针移动到上一行，如果移动成功则返回true。一旦将记录指针移动到指定行之后，接下来就可以调用Cursor的getXxx（）方法获取该行的指定列的数据了。

提示：

其实如果读者具有 JDBC 编程的经验，则完全可以把 SQLiteDatabase当成JDBC中Connection和Statement的混合体—因为SQLiteDatabase既代表了与数据库的连接，也可直接用于执行SQL操作；而Android中Cursor则可当成ResultSet而且Cursor提供了更多便捷的方法来操作结果集。实际上对于一个Java程序员来说，JDBC几乎是必备的编程基础，如果读者需要详细学习JDBC的相关知识，可以参考疯狂Java体系的《疯狂Java讲义》一书。

8.3.2 创建数据库和表

前面已经讲到，使用SQLiteDatabase的静态方法即可打开或创建数据库，例如如下代码：

上面的代码就用于打开或创建一个SQLite数据库，如果/mnt/db/目录下的temp.db3文件（该文件就是一个数据库）存在，那么程序就是打开该数据库；如果该文件不存在，则上面的代码将会在该目录下创建temp.db3文件（即对应于数据库）。

上面的代码中没有指定SQLiteDatabase.CursorFactory参数，该参数是一个用于返回Cursor的工厂，如果指定该参数为null，则意味着使用默认的工厂。

上面的代码即可返回一个SQLiteDatabase对象，该对象的execSQL（）方法可执行任意的SQL语句，因此可通过如下代码在程序中创建

数据表：

在程序中执行上面的代码即可在数据库中创建一个数据表。

8.3.3 使用SQL语句操作SQLite数据库

正如前面提到的，SQLiteDatabase的execSQL（）方法可执行任意的SQL语句，包括带占位符的SQL语句。但由于该方法没有返回值，因此一般用于执行DDL语句或DML语句；如果需要执行查询语句，则可调用SQLiteDatabase的rawQuery（String sql，String[] selectionArgs）方法。

例如如下代码可用于执行DML语句：

下面的程序示范了如何在Android应用中操作SQLite数据库。该程序提供了两个文本框，用户可以在这两个文本框中输入内容，当用户单击“插入”按钮时这两个文本框的内容将会被插入数据库。

程序清单：
codes\08\8.3\DBTest\app\src\main\java\org\crazyit\db\MainActivity.java

上面的程序中①号粗体字代码用于创建或打开SQLite数据库。当用户单击程序中的“插入”按钮时，程序会调用②号粗体字代码向底层数据表中插入一行记录，并执行查询语句，把底层数据表中的记录查询出来，而且使用ListView将查询结果（Cursor）显示出来。

程序中③号粗体字代码用于将Cursor封装成SimpleCursorAdapter，这个SimpleCursorAdapter实现了Adapter接口，因此可以作为ListView或GridView的内容适配器。

SimpleCursorAdapter的构造器参数与SimpleAdapter的构造器参数大致相同，区别是SimpleAdapter负责封装集合元素为Map的List，而SimpleCursorAdapter负责封装Cursor—如果我们把Cursor里的结果集当成List集合，Cursor里的每一行当成Map处理（以数据列的列名为key，数据列的值为value），那么SimpleCursorAdapter与SimpleAdapter就统一起来了。

图8.7 数据库访问示例

运行上面的程序，将看到如图8.7所示的界面。

正如上面的程序中所看到的，当程序不断地向底层数据表中插入数据时，程序中ListView将可以把底层数据表中数据显示出来。

需要指出的是，使用SimpleCursorAdapter封装Cursor时要求底层数据表的主键列的列名为_id，因为SimpleCursorAdapter只能识别列名为_id的主键。因此上面的程序创建数据表时指定了主键列的列名为_id；否则就会出现java.lang.IllegalArgumentException：column'_id'does not exist错误。

上面的程序中我们重写了Activity的onDestroy（）方法，当应用程序退出该Activity时将会回调该方法，程序在该方法中关闭了SQLiteDatabase—就像JDBC编程中需要关闭Statement和Connection一样，这里也需要关闭SQLiteDatabase；否则可能引发资源泄漏。总结起来，使用SQLiteDatabase进行数据库操作的步骤如下。

1 获取SQLiteDatabase对象，它代表了与数据库的连接。

2 调用SQLiteDatabase的方法来执行SQL语句。

3 操作SQL语句的执行结果，比如用SimpleCursorAdapter封装Cursor。

4 关闭SQLiteDatabase，回收资源。

8.3.4 使用sqlite3工具

在 Android S DK 的 platform-tools 目录下提供了一个 sqlite3.exe 文件，它是一个简单的SQLite数据库管理工具，类似于MySQL提供的命令行窗口。有些时候，开发者利用该工具来查询、管理数据库。

例如，我们把上面的应用程序所生成的my.db3导出到本地计算机上的F：\盘根目录下，接下来可运行如下命令来启动SQLite数据库：

运行上面的命令，可以看到如图8.8所示的窗口。

图8.8 使用sqlite3工具

从图8.8可以看出，sqlite3中常用的命令如下。

➢**.databases**：查看当前数据库。

➢**.tables**：查看当前数据库里的数据表。

➢**.help**：查看sqlite3支持的命令。

当然，sqlite3还支持一些常用的命令，当开发者在图8.8所示窗口中输入.help之后，该工具将会列出这些命令。除此之外，SQLite数据库还支持绝大部分常用的SQL语句，开发者可在图8.8所示窗口中运行各种DDL、DML、查询语句来测试它们。

提示：

SQLite数据库所支持的SQL语句与MySQL大致相同，开发者完全可以把已有的MySQL经验“移植”到SQLite数据库上。当然，当Android应用提示某条SQL语句有语法错误时，最好先利用sqlite3这个工具类测试这条语句，以保证这条SQL语句的语法正确。

需要指出的是，SQLite内部只支持 NULL、INTEGER、REAL（浮点数）、TEXT（文本）和BLOB（大二进制对象）这5种数据类型，但实际上SQLite完全可以接受varchar（n）、char（n）、decimal（p，s）等数据类型，只不过SQLite会在运算或保存时将它们转换为上面5种数据类型中相应的类型。

除此之外，SQLite还有一个特点：它允许把各种类型的数据保存到任何类型字段中，开发者可以不用关心声明该字段所使用的数据类型。例如，程序可以把字符串类型的值存入INTEGER类型的字段中，也可以把数值类型的值存入布尔类型的字段中……但有一种情况例外：定义为INTEGER PRIMARY KEY的字段只能存储64位整数，当向这种字段中保存除整数以外的其他类型的数据时，SQLite会产生错误。

由于SQLite允许存入数据时忽略底层数据列实际的数据类型，因此在编写建表语句时可以省略数据列后面的类型声明。例如，如下SQL语句对于SQLite也是正确的：

开发者可以把MySQL开发经验直接移植到SQLite数据库上，如果开发者需要了解更多关于SQL语法、MySQL数据库的知识，请参考疯狂Java体系的《疯狂Java讲义》一书。

8.3.5 使用特定方法操作SQLite数据库

如果开发者对SQL语法不熟悉，甚至以前从未使用过任何数据库，Android的SQLiteDatabase提供了insert、upate、delete或query语句来操作数据库。

提示：

虽然Android提供了这些所谓的“便捷”方法来操作SQLite数据库，但在笔者看来这些方法纯属“鸡肋”，对于一个程序员而言，SQL语法可以说是基本功中的基本功—你见过不会1+1=2的数学工作者吗？

不过，既然Android提供了这些方法，这里也简单介绍一下。

1.使用insert方法插入记录

SQLiteDatabase的insert方法的签名为long insert（String table，String nullColumnHack，ContentValues values），这个插入方法的参数说明如下。

➢ **table**：代表想插入数据的表名。

➢ **nullColumnHack**：代表强行插入null值的数据列的列名。当values参数为null或不包含任何key-value对时该参数有效。

➢ **values**：代表一行记录的数据。

insert方法插入的一行记录使用ContentValues存放，ContentValues类似于Map，它提供了put（String key，Xxx value）（其中key为数据列的列名）方法用于存入数据，getAsXxx（String key）方法用于取出数据。

例如如下语句：

不管第三个参数是否包含数据，执行insert（）方法总会添加一条记录，如果第三个参数为空，则会添加一条除主键之外其他字段值都为null的记录。

insert（）方法的底层实际上依然是通过构造insert SQL语句来进行插入的，因此它生成的SQL语句总是形如下面的语句：

此时如果第三个参数为null或其中key-value对的数量为0，由于insert（）方法还会按此方式生成一条insert语句，此时的insert语句为：

上面的SQL语句显然有问题。为了满足SQL语法的需要，insert语句必须给定一个列名，如：insert into person（name） values（null），这个name列名就由第二个参数来指定。由此可见，当ContentValues为null或它包含的key-value对的数量为0时，第二个参数就会起作用了。

一般来说，第二个参数指定的列名不应该是主键列的列名，也不应该是非空列的列名；否则强行往这些数据列插入null会引发异常。

2.使用update方法更新记录

SQLiteDatabase的update方法的签名为update（String table，ContentValues values，String whereClause，String[] whereArgs），这个更新方法的参数说明如下。

➢ **table**：代表想更新数据的表名。

➢ **values**：代表想更新的数据。

➢ **whereClause**：满足该whereClause子句的记录将会被更新。

➢ **whereArgs**：用于为whereClause子句传入参数。

该方法返回受此update语句影响的记录的条数。

例如，我们想更新person_inf表中所有主键大于20的人的人名，可调用如下方法：

实际上update方法底层对应的SQL语句如下：

其中whereArgs参数用于向whereClause中传入参数。

3.使用delete方法删除记录

SQLiteDatabase的delete方法的签名为delete（String table，String whereClause，String[] whereArgs），这个删除方法的参数说明如下。

➢ **table**：代表想删除数据的表名。

➢ **whereClause**：满足该whereClause子句的记录将会被删除。

➢ **whereArgs**：用于为whereClause子句传入参数。

该方法返回受此delete语句影响的记录的条数。

例如，我们想删除person_inf表中所有人名以“孙”开头的记录，可调用如下方法：

实际上delete方法底层对应的SQL语句如下：

4.使用query方法查询记录

SQLiteDatabase的query方法的签名为Cursor query（boolean distinct，String table，String[] columns，String whereClause，String[] selectionArgs，String groupBy，String having，String orderBy，String limit），这个query方法的参数说明如下。

➢ **distinct**：指定是否去除重复记录。

➢ **table**：执行查询数据的表名。

➢ **columns**：要查询出来的列名。相当于select语句select关键字后面的部分。

➢ **whereClause**：查询条件子句，相当于select语句where关键字后面的部分，在条件子句中允许使用占位符“？”。

➢ **selectionArgs**：用于为whereClause子句中的占位符传入参数值，值在数组中的位置与占位符在语句中的位置必须一致；否则就会有异常。

➢ **groupBy**：用于控制分组。相当于select语句group by关键字后面的部分

➢ **having**：用于对分组进行过滤。相当于select语句having关键字后面的部分

➢ **orderBy**：用于对记录进行排序。相当于 select 语句 order b y 关键字后面的部分，如personid desc，age asc。

➢ **limit**：用于进行分页。相当于select语句limit关键字后面的部分，如5，10。

看到这个方法的设计，笔者忍不住想再次表达对这个方法的不满：如果读者完全不懂SQL语句，那需要花多少时间才能理解这个方法中这么多参数的设置。如果读者愿意花时间去理解这个方法中各参数的设置，那么所花的时间已经足够去掌握这条select语句的语法格式了。

当然，这个query（）方法也并非完全一无是处：当应用程序需要进行“条件不确定”的查询（即查询条件需要动态改变的查询）时，使用这个query（）方法可以避免手动拼接SQL语句。

例如，想查询出person_inf表中人名以“孙”开头的记录，可使用如下语句：

8.3.6 事务

SQLiteDatabase中包含如下两个方法来控制事务。

➢ **beginTransaction（）：** 开始事务。

➢ **endTransaction（）：** 结束事务。

除此之外，SQLiteDatabase还提供了如下方法来判断当前上下文是否处于事务环境中。

➢ **inTransaction（）：** 如果当前上下文处于事务中，则返回true；否则返回false。

当程序执行endTransaction（）方法时将会结束事务—那到底是提交事务，还是回滚事务呢？这取决于SQLiteDatabase是否调用了setTransactionSuccessful（）方法来设置事务标志，如果程序在事务执行中调用该方法设置了事务成功则提交事务；否则程序将会回滚事务。

示例代码如下：

8.3.7 SQLiteOpenHelper类

在上一个示例程序中，我们为了判断底层数据库是否包含news_inf数据表，采用的处理方法十分烦琐：程序先尝试向news_inf数据表中插入记录，如果程序抛出异常，在异常捕获的catch块中创建news_inf数据表，然后再插入记录。那么是否有一种更优雅的方式来处理这种问题呢？有，Android提供了SQLiteOpenHelper类来处理这种问题。

在实际项目中很少使用SQLiteDatabase的方法来打开数据库，通常都会继承SQLiteOpenHelper开发子类，并通过该子类的getReadableDatabase（）、getWritableDatabase（）方法打开数据库。

SQLiteOpenHelper是Android提供的一个管理数据库的工具类，可用于管理数据库的创建和版本更新。一般的用法是创建SQLiteOpenHelper的子类，并扩展它的onCreate （SQLiteDatabase db）和onUpgrade （SQLiteDatabase db，int oldVersion，int newVersion）方法。

SQLiteOpenHelper包含如下常用的方法。

➢ **synchronized SQLiteDatabase getReadableDatabase（）：**以读写的方式打开数据库对应的SQLiteDatabase对象。

➢ **synchronized SQLiteDatabase getWritableDatabase（）：**以写的方式打开数据库对应的SQLiteDatabase对象。

➢ **abstract void onCreate（SQLiteDatabase db）：**当第一次创建数据库时回调该方法。

➢ **abstract void onUpgrade（SQLiteDatabase db，int oldVersion，int newVersion）：**当数据库版本更新时回调该方法。

➢ **synchronized void close（）：**关闭所有打开的SQLiteDatabase对象。

从上面的方法介绍中不难看出，SQLiteOpenHelper提供了getReadableDatabase（）、getWritableDatabase（）两个方法用于打开数据库连接，并提供了close（）方法来关闭数据库连接，而开发者需要做的就是重写它的两个抽象方法。

➢ **onCreate（SQLiteDatabase db）：**用于初次使用软件时生成数据库表。当调用SQLiteOpenHelper的getWritableDatabase（）或者getReadableDatabase（）方法获取用于操作数据库的SQLiteDatabase实例时，如果数据库不存在，Android系统会自动生成一个数据库，接着调用 onCreate（）方法，onCreate（）方法在初次生成数据库时才会被调用。重写onCreate（）方法时，可以生成数据库表结构，以添加应用使用到的一些初始化数据。

➢ **onUpgrade（SQLiteDatabase db，int oldVersion，int newVersion）：**用于升级软件时更新数据库表结构，此方法在数据库的版本发生变化时会被调用，该方法调用时oldVersion 代表数据库之前的版本号，newVersion 代表数据库当前的版本号。那么在哪里指定数据库的版本号呢？当程序创建 SQLiteOpenHelper 对象时，必须指定一个version参数，该参数就决定了所使用的数据库的版本—也就是说，数据库的版本是由程序员控制的。只要某次创建SQLiteOpenHelper时指定的数据库版本号高于之前指定的版本号，系统就会自动触发 onUpgrade（SQLiteDatabase db，int o ldVersion，int newVersion）方法，程序就可以在onUpgrade（）方法里面根据原版本号和目标版本号进行判断，即可根据版本号进行必需的表结构更新。

提示：

实际上，当应用程序升级表结构时，完全可能因为已有的数据导致升级失败。在这种时候程序可能需要先对数据进行转储，清空数据表中的记录，接着对数据表进行更新，当数据表更新完成后再将数据保存回来。

一旦得到了SQLiteOpenHelper对象之后，程序无须使用SQLiteDatabase的静态方法创建SQLiteDatabase实例，而且可以使用getWritableDatabase（）或getReadableDatabase（）方法来获取一个用于操作数据库的SQLiteDatabase实例。

其中getWritableDatabase（） 方法以写的方式打开数据库，一旦数据库的磁盘空间满了，数据库就只能读而不能写，倘若使用getWritableDatabase（）打开数据库就会出错。getReadableDatabase（）方法先以读写方式打开数据库，如果数据库的磁盘空间满了，就会打开失败，当打开失败后会继续尝试以只读方式打开数据库。

下面以一个实例程序来说明SQLiteOpenHelper的功能与用法。

实例：英文生词本

该实例允许用户将自己不熟悉的单词添加到系统数据库中，当用户需要查询某个单词或解释时，只要在程序中输入相应的关键词，程序中相应的条目就会显示出来。

该程序使用了SQLiteOpenHelper的子类来管理数据库连接，当程序通过SQLiteOpenHelper的子类来获取数据库连接时，如果数据库还不存在，系统会自动调用onCreate（SQLiteDatabase db）方法来创建数据库；如果程序创建SQLiteOpenHelper的子类的实例时传入的数据库版本号高于之前的版本号，系统会自动调用onUpgrade（SQLite Database db，int oldVersion，int newVersion）方法来更新数据库。

下面是SQLiteOpenHelper的子类代码。

程序清单：
codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\MyDatabaseHelper.java

上面的MyDatabaseHelper继承了SQLiteOpenHelper，并重写了其基类的onCreate（SQLiteDatabase db）方法，该方法中执行的建表语句用于初始化系统数据表。如果用户第一次使用该程序，系统将会自动调用onCreate（SQLiteDatabase db）方法来初始化底层数据库。

MyDatabaseHelper工具类的作用主要是管理数据库的初始化，并允许应用程序通过该工具类来获取SQLiteDatabase对象。接下来的程序就可通过该工具类获取SQLiteDatabase对象，并利用该对象操作数据库了。

本实例的程序界面比较简单，它提供了输入框让用户添加生词，并提供输入框让用户输入要查询的关键词，用户在查询文本框中输入关键词后单击“查找”按钮即可显示相应的生词条目。该程序的界面如图8.9所示。

图8.9所示的界面中前两个文本框用于添加生词；第三个文本框用于输入想查询的关键词，当用户输入相应的关键词，并单击“查找”按钮后，系统将会查询相应的条目。下面是该生词本的主程序。

图8.9 生词本界面

程序清单：
codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\MainActivity.java

上面的程序中第一行粗体字代码用于根据SQLiteOpenHelper来获取SQLiteDatabase，用于执行插入数据、查询数据。程序重写了Activity的onDestroy（）方法，并在该方法中调用SQLiteOpenHelper的close（）方法来关闭数据库连接。

当程序第一次调用getWritableDatabase（）或getReadableDatabase（）方法后，SQLiteOpenHelper会缓存已获得的SQLiteDatabase实例，在正常情况下SQLiteDatabase实例会维持数据库的打开状态，因此程序退出时应该关闭不再使用的SQLiteDatabase。一旦SQLiteOpenHelper缓存了SQLiteDatabase实例之后，多次调用getWritableDatabase（）或getReadableDatabase（）方法得到的都是同一个SQLiteDatabase实例。

上面的程序执行查询后并未在该Activity中显示查询得到的条目，而是使用另一个Activity：ResultActivity来显示查询结果，ResultActivity只是提供了一个ListView来显示查询得到的条目。为了把dict查询得到的条目传给ResultActivity，程序将查询得到的条目封装成List<Map<String，String>>后传给ResultActivity，ResultActivity就可利用ListView来显示查询结果了。

下面是ResultActivity的代码。

程序清单：
codes\08\8.3\Dict\app\src\main\java\org\crazyit\db\ResultActivity.java

上面的ResultActivity只是一个普通的Activity，但我们在AndroidManifest.xml文件中将该Activity设为对话框风格的Activity，这样就可让应用程序以对话框来显示查询结果了。查询结果如图8.10所示。

8.4 手势（Gesture）

所谓手势，其实是指用户手指或触摸笔在触摸屏上的连续触碰行为，比如在屏幕上从左至右划出的一个动作，就是手势；再比如在屏幕上画出一个圆圈也是手势。手势这种连续的触碰会形成某个方向上的移

动趋势，也会形成一个不规则的几何图形。Android对两种手势行为都提供了支持。

图8.10 查询生词

➢ 对于第一种手势行为，Android提供了手势检测，并为手势检测提供了相应的监听器。

➢ 对于第二种手势行为，Android 允许开发者添加手势，并提供了相应的 API 识别用户手势。

8.4.1 手势检测

Android为手势检测提供了一个GestureDetector类，GestureDetector实例代表了一个手势检测器，创建GestureDetector时需要传入一个GestureDetector.OnGestureListener实例，GestureDetector.OnGestureListener就是一个监听器，负责对用户的手势行为提供响应。

GestureDetector.OnGestureListener里包含的事件处理方法如下。

➢ **boolean onDown（MotionEvent e）**：当触碰事件按下时触发该方法。

➢ **boolean onFling（MotionEvent e1，MotionEvent e2，float velocityX，float velocityY）**：当用户手指在触摸屏上"拖过"时触发该方法。其中velocityX、velocityY代表"拖过"动作在横向、纵向上的速度。

➢ **abstract void onLongPress（MotionEvent e）**：当用户手指在屏幕上长按时触发该方法。

➢ **boolean onScroll（MotionEvent e1，MotionEvent e2，float distanceX，float distanceY）**：当用户手指在屏幕上"滚动"时触发该方法。

➢ **void onShowPress（MotionEvent e）**：当用户手指在触摸屏上按下，而且还未移动和松开时触发该方法。

➢ **boolean onSingleTapUp（MotionEvent e）**：用户手指在触摸屏上的轻击事件将会触发该方法。

关于GestureDetector.OnGestureListener监听器里各方法的触发时机，仅从文字上表述总显得比较抽象而且难于理解，下面将以一个最简单的例子来让读者理解各方法的触发时机。

使用Android的手势检测只需两个步骤。

1 创建一个GestureDetector对象。创建该对象时必须实现一个GestureDetector.On GestureListener监听器实例。

2 为应用程序的Activity（偶尔也可为特定组件）的TouchEvent事件绑定监听器，在事件处理中指定把Activity（或特定组件）上的TouchEvent事件交给GestureDetector处理。

经过上面两个步骤之后，Activity（或特定组件）上的TouchEvent事件就会交给GestureDetector处理，而GestureDetector就会检测是否触发了特定的手势动作。

下面的程序测试了用户的不同动作到底触发哪种手势动作。

程序清单：
codes\08\8.4\GestureTest\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一行粗体字代码创建了一个GestureDetector对象，创建该对象时传入了this作为参数，这表明该Activity本身将会作为GestureDetector.OnGestureListener监听器，所以该Activity实现了该接口，并实现了该接口里的全部方法。程序中第二行粗体字代码指定把Activity上的TouchEvent交给GestureDetector处理。

运行上面的程序，当用户随意地在屏幕上触碰时，程序将会检测到用户到底执行了哪些手势，如图8.11所示。

图8.11 手势检测

掌握了用户在屏幕上的哪些动作对应于哪些手势之后，接下来再以两个实例来介绍Android手势检测在实际项目中的应用。

实例：通过手势缩放图片

前面介绍过图片缩放的技术实现，通过Matrix即可实现图片缩放，但之前的图片缩放要么通过按钮控制，要么通过SeekBar来控制，这些缩放方式太“传统”了。而本实例所介绍的图片缩放则“炫”得多：用户只要在图片上随意地“挥动”手指，图片就可被缩放—从左向右挥动时图片被放大，从右向左挥动时图片被缩小；挥动速度越快，缩放比越大。

该实例的程序界面布局很简单，只在界面中间定义一个ImageView来显示图片即可。该程序的思路是使用一个GestureDetector来检测用户手势，并根据用户手势在横向的速度缩放图片。该程序代码如下。

程序清单：
codes\08\8.4\GestureZoom\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序的处理方式与前一个程序的处理方式如出一辙：第一行粗体字代码用于创建GestureDetector对象，第二行粗体字代码指定把Activity的OnTouch事件交给GestureDetector处理。

该程序与前一个程序的不同之处在于：程序实现GestureDetector.OnGestureListener监听器时，只实现了②号代码所标出的onFling（MotionEvent event1，MotionEvent event2，float velocityX，float velocityY）方法，并在该方法内根据velocityX参数（横向上的拖动速度）来计算图片缩放比，这样该程序即可根据用户的"手势"来缩放图片了。

实例：通过手势实现翻页效果

本实例的手势检测思路还是一样：把Activity的TouchEvent交给GestureDetector处理。这个程序的特殊之处在于：该程序使用了一个ViewFlipper组件，ViewFlipper可使用动画控制多个组件之间的切换效果。

本程序通过GestureDetector来检测用户的手势动作，并根据手势动作来控制ViewFlipper包含的View组件的切换，从而实现翻页效果。

该程序的界面布局代码如下。

程序清单：
codes\08\8.4\GestureFlip\app\src\main\res\layout\main.xml

上面的界面布局文件中定义了一个ViewFlipper组件，该组件可控制多个View的动画切换。该实例的程序代码如下。

程序清单：
codes\08\8.4\GestureFlip\app\src\main\java\org\crazyit

io\MainActivity.java

该程序同样只是实现了GestureDetector.OnGestureListener的onFling（）方法。上面的程序中粗体字代码负责实现：当event1.getX（）-event2.getX（）的距离大于特定距离时，即可判断用户手势为从右向左滑动，此时设置ViewFlipper采用动画方式切换为上一个View；当event2.getX（）-event1.getX（）的距离大于特定距离时，即可判断用户手势为从左向右滑动，此时设置ViewFlipper采用动画方式切换为下一个View—这样就实现了所谓的“翻页”效果。

运行上面的程序，并在屏幕上执行“拖动”手势，即可看到ViewFlipper内组件切换的效果，如图8.12所示。

图8.12 采用手势检测实现翻页效果

8.4.2 增加手势

Android除了提供手势检测之外，还允许应用程序把用户手势（多个持续的触摸事件在屏幕上形成特定的形状）添加到指定文件中，以备以后使用—如果程序需要，当用户下次再次画出该手势时，系统将可识别该手势。

Android使用GestureLibrary来代表手势库，并提供了GestureLibraries工具类来创建手势库。GestureLibraries提供了如下4个静态方法从不同位置加载手势库。

➢ **static GestureLibrary fromFile（String path）**：从path代表的文件中加载手势库。

➢ **static GestureLibrary fromFile（File path）**：从path代表的文件中加载手势库。

➢ **static GestureLibrary fromPrivateFile（Context context，String name）**：从指定应用程序的数据文件夹的name文件中加载手势库。

➢ **static GestureLibrary fromRawResource（Context context，int resourceId）**：从resourceId所代表的资源中加载手势库。

一旦在程序中获得了GestureLibrary对象之后，该对象提供了如下方法来添加手势、识别手势。

➢ **void addGesture（String entryName，Gesture gesture）**：添加一个名为entryName的手势。

➢ **Set<String>getGestureEntries（）**：获取该手势库中的所有手势的名称。

➢ **ArrayList<Gesture> getGestures（String entryName）**：获取entryName名称对应的全部手势。

➢ **ArrayList<Prediction>recognize（Gesture gesture）**：从当前手势库中识别与gesture匹配的全部手势。

➢ **void removeEntry（String entryName）**：删除手势库中entryName对应的手势。

➢ **void removeGesture（String entryName，Gesture gesture）**：删除手势库中entryName、gesture对应的手势。

➢ **boolean save（）**：当向手势库中添加手势或从中删除手势后调用该方法保存手势库。

Android除了提供了GestureLibraries、GestureLibrary来管理手势之外，还提供了一个专门的手势编辑组件：GestureOverlayView，该组件就像一个“绘图组件”，只是用户在组件上绘制的不是图形，而是手势。

为了监听GestureOverlayView组件上的手势事件，Android为GestureOverlayView提供了OnGestureListener、OnGesturePerformedListener、OnGesturingListener三个监听器接口，这些监听器所包含的方法分别用于响应手势开始、结束、完成、取消等事件，开发者可根据实际需要来选择不同的监听器——一般来说，OnGesturePerformedListener是最常用的监听器，它可用于在手势事件完成时提供响应。

下面的应用程序在界面布局中使用了GestureOverlayView。

程序清单：
codes\08\8.4\AddGesture\app\src\main\res\layout\main.xml

由于GestureOverlayView并不是标准的视图组件，因此在界面布局中使用该组件时需要使用全限定类名。

上面的程序中使用GestureOverlayView组件时指定了一个android:gestureStrokeType参数，该参数控制手势是否需要多一笔完成。大部分时候，一个手势只要一笔就可以完成，此时可将该参数设为single。如果该手势需要多笔来完成，则将该参数设为multiple。

接下来程序将会为GestureOverlayView添加一个OnGesturePerformedListener监听器，当手势事件完成时，该监听器会打开一个对话框，让用户选择保存该手势。程序代码如下。

程序清单：
codes\08\8.4\AddGesture\app\src\main\java\org\crazyit

io\MainActivity.java

上面程序中的第一行粗体字代码用于为GestureOverlayView绑定OnGesturePerformedListener监听器，该监听器用于在手势完成时提供响应一它的响应就是打开一个对话框。该对话框的界面布局代码如下。

程序清单：
codes\08\8.4\AddGesture\app\src\main\res\layout\save.xml

AddGesture程序中的第二段粗体字代码是在对话框中完成的，这段粗体字代码用于从SD卡的指定文件中加载手势库，并添加用户刚刚输入的手势。

运行该程序，将看到如图8.13所示的界面。

用户可在图8.13所示的界面中随意地“绘制”手势，绘制完成后OnGesturePerformedListener监听器将会打开如图8.14所示的对话框。

图8.13 添加手势

图8.14 保存手势

当用户单击“保存”按钮后，程序将会调用GestureLibrary的addGesture（）方法来添加手势，并调用save（）方法来保存手势。

一旦用户通过该程序建立了自己的手势库，接下来就可在其他程序中使用该手势库了。

注意：

上面的程序需要将手势库保存在SD卡上，因此还需要授予该应用程序读写SD卡的权限。

8.4.3 识别用户手势

前面已经提到，GestureLibrary提供了recognize（Gesture ges）方法来识别手势，该方法将会返回该手势库中所有与ges匹配的手势—两个手势的图形越相似，相似度越高。

recognize（Gesture ges）方法的返回值为ArrayList<Prediction>，其中Prediction封装了手势的匹配信息，Prediction对象的name属性代表了匹配的手势名，score属性代表了手势的相似度。

下面的程序将会利用前一个程序所创建的手势库来识别手势。该程序的界面很简单，只是在界面中定义了一个GestureOverlayView组件，允许用户在该组件上输入手势。程序为该组件绑定了OnGesturePerformedListener监听器，该监听器检测到用户手势完成时，就会调用手势库来识别用户输入的手势。

该程序的代码如下。

程序清单：
codes\08\8.4\RecogniseGesture\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的粗体字代码就负责调用前一个程序的手势库来识别用户刚输入的手势，用户只要在屏幕上绘制一个大致与之前相似的手势，即可看到如图8.15所示的结果。

注意：

该程序需要读取SD卡上的手势文件，因此该程序还需要授予读取SD卡的权限。

8.5 自动朗读（TTS）

图8.15 识别手势

Android提供了自动朗读支持。自动朗读支持可以对指定文本内容进行朗读，从而发出声音；不仅如此，Android的自动朗读支持还允许把文本对应的音频录制成音频文件，方便以后播放。这种自动朗读支持的英文名称为TextToSpeech，简称TTS。借助于TTS的支持，可以在应用程序中动态地增加音频输出，从而改善用户体验。

Android的自动朗读支持主要通过TextToSpeech来完成，该类提供了如下一个构造器。

➢ **TextToSpeech(Context context,TextToSpeech.OnInitListener listener)**

从上面的构造器不难看出，当创建TextToSpeech对象时，必须先提供一个OnInitListener监听器—该监听器负责监听TextToSpeech的初始化结果。

一旦在程序中获得了TextToSpeech对象之后，接下来即可调用TextToSpeech的setLanguage（Locale loc）方法来设置该TTS发声引擎应使用的语言、国家选项。

提示：

由于不同的文字，在不同的语言、国家中的发音是不同的，尤其是欧美，他们所使用的都是字母文字，因此一段文本内容，使用不同的语言、国家选项来朗读，发音效果是截然不同的。目前Android的TTS暂时不支持中文。

如果调用setLanguage（Locale loc）的返回值是“TextToSpeech.LANG_COUNTRY_AVAILABLE”，则说明当前TTS系统可以支持所设置的语言、国家选项。

对TextToSpeech设置完成后，就可调用它的方法来朗读文本了，具体方法可参考TextToSpeech的API文档。TextToSpeech类中最常用的方法是如下两个。

➢ **speak(CharSequence text,int queueMode,Bundle params,String utteranceId)**

➢ **synthesizeToFile (CharSequence text,Bundle params,File file,String utteranceId)**

上面两个方法都用于把text文字内容转换为音频，区别只是speak（）方法是播放转换的音频，而synthesizeToFile（）方法是把转换得到的音频保存成声音文件。

上面两个方法中的params都用于指定声音转换时的参数。speak（）方法中的queueMode参数指定TTS的发音队列模式，该参数支持如下两个常量。

➢ **TextToSpeech.QUEUE_FLUSH**：如果指定该模式，当TTS调用speak（）方法时，它会中断当前实例正在运行的任务（也可以理解为清除当前语音任务，转而执行新的语音任务）。

➢ **TextToSpeech.QUEUE_ADD**：如果指定为该模式，当TTS调用speak（）方法时，会把新的发音任务添加到当前发音任务列队之后—

也就是等任务队列中的发音任务执行完成后再来执行speak（）方法指定的发音任务。

当程序用完了TextToSpeech对象之后，可以在Activity的OnDestroy（）方法中调用它的shutdown（）来关闭TextToSpeech，释放它所占用的资源。

归纳起来，使用TextToSpeech的步骤如下。

1 创建TextToSpeech对象，创建时传入OnInitListener监听器监听创建是否成功。

2 设置TextToSpeech所使用的语言、国家选项，通过返回值判断TTS是否支持该语言、国家选项。

3 调用speak（）或synthesizeToFile（）方法。

4 关闭TTS，回收资源。

下面的程序示范了如何利用TTS来朗读用户所输入的文本内容。

程序清单：
codes\08\8.5\Speech\app\src\main\java\org\crazyit\io\MainActivity.java

上面程序中的第一行粗体字代码创建了一个TextToSpeech对象，第二行粗体字代码设置使用美式英语进行朗读。接下来程序分别提供了两个按钮，一个按钮用于执行朗读，一个按钮用于将文本内容的朗读音频保存成声音文件，分别通过调用TextToSpeech对象的speek（）和synthesizeToFile（）方法来完成—如上面程序中的后两行粗体字代码所示。

图8.16 TTS

运行上面的程序，将可以看到如图8.16所示的界面。

在图8.16所示的界面中，当用户单击“朗读”按钮后，系统将会调用TTS的speek（）方法来朗读文本框中的内容；当用户单击“记录声音”按钮后，系统将会调用synthesizeToFile（）方法把文本框中的文本对应的朗读音频记录到SD卡的声音文件中—单击该按钮后将可以在SD卡的根目录下生成一个sound.wav文件，该文件可以被导出，在其他音频播放软件中播放。

程序重写Activity的onDestroy（）方法，并在该方法中关闭了TextToSpeech对象，回收了它的资源。

8.6 本章小结

本章主要介绍了Android的输入、输出支持，如果读者之前已有Java IO的编程经验，那么可以直接把 Java I O 的编程经验移植到 Android 上。当然 Android 为文件 IO 提供了openFileOutput和openFileInput两个便捷的方法，用起来十分方便；为记录、访问应用程序的参数、选项提供了SharedPreferences工具类，这样可以非常方便地读写应用程序的参数、选项。除此之外，学习本章需要重点掌握的内容就是SQLite数据库，Android系统内置了SQLite数据库，而且为访问SQLite数据库提供了大量方便的工具类，这需要读者掌握并能熟练地使用它们。

本章后面还介绍了Android提供的“另类”IO：手势支持和自动朗读。Android的手势支持体现在两方面：手势检测与手势识别，前者属于事件处理方面，后者属于系统IO方面。熟练运用Android系统的手势支持可以开发出一些更新奇的应用。自动朗读则用于把文本转换成声音。

第9章 使用 ContentProvider实现数据共享

本章要点

理解ContentProvider的功能与意义

ContentProvider类的作用和常用方法

Uri对ContentProvider的作用

理解ContentProvider与ContentResolver的关系

实现自己的ContentProvider

配置ContentProvider

使用ContentResolver操作数据

操作系统ContentProvider提供的数据

监听ContentProvider的数据改变

ContentObserver类的作用和常用方法

监听系统ContentProvider的数据改变

当在系统中部署一个又一个Android应用之后，系统里将会包含多个Android应用，有时候就需要在不同的应用之间共享数据。比如现在有一个短信接收应用，用户想把接收到的陌生短信的发信人添加到联系人管理应用中，就需要在不同应用之间共享数据。对于这种需要在不同应用之间共享数据的需求，当然可以让一个应用程序直接去操作另一个应用程序所记录的数据，比如操作它所记录的

SharedPreferences、文件或数据库等。这种方式不仅比较麻烦，而且存在严重的安全漏洞，因此Android 4.2不再推荐使用这种方式，而是推荐使用本章介绍的ContentProvider。

为了在应用程序之间交换数据，Android提供了ContentProvider，它是不同应用程序之间进行数据交换的标准API，当一个应用程序需要把自己的数据暴露给其他程序使用时，该应用程序就可通过提供ContentProvider来实现；其他应用程序就可通过ContentResolver来操作ContentProvider暴露的数据。

ContentProvider也是Android应用的四大组件之一，与Activity、Service、BroadcastReceiver相似，它们都需要在AndroidManifest.xml文件中进行配置。

一旦某个应用程序通过ContentProvider暴露了自己的数据操作接口，那么不管该应用程序是否启动，其他应用程序都可通过该接口来操作该应用程序的内部数据，包括增加数据、删除数据、修改数据、查询数据等。

9.1 数据共享标准：ContentProvider

ContentProvider是不同应用程序之间进行数据交换的标准API，ContentProvider以某种Uri的形式对外提供数据，允许其他应用访问或修改数据；其他应用程序使用ContentResolver根据Uri去访问操作指定数据。

提示：

对于初学者而言，对于ContentProvider、ContentResolver两个核心API的作用可能需要花很长时间去理解。但这里有一个简单的类比，可以把ContentProvider当成Android系统内部的“网站”，这个网站以固定的Uri对外提供服务；而ContentResolver则可当成Android系统内部的HttpClient，它可以向指定Uri发送“请求”（实际上是调用

ContentResolver的方法），这种请求最后委托给ContentProvider处理，从而实现对“网站”（即ContentProvider）内部数据进行操作。

9.1.1 ContentProvider简介

如果把ContentProvider当成一个“网站”来看，那么如何对外提供数据呢？是否需要像Java Web开发一样编写JSP、Servlet之类呢？不需要。如果那样就太复杂了，毕竟ContentProvider只是提供数据的访问接口，并不是像一个网站一样对外提供完整的页面。

如果把ContentProvider当成一个“网站”来看，那么如何完整地开发一个ContentProvider呢？步骤其实很简单，如下所示。

1 定义自己的ContentProvider类，该类需要继承Android提供的ContentProvider基类。

2 向Android系统注册这个“网站”，也就是在AndroidManifest.xml文件中注册这个ContentProvider，就像注册Activity一样。注册ContentProvider时需要为它绑定一个Uri。

向Android系统中注册ContentProvider只要在<application.../>元素下添加如下子元素即可：

提示：

虽然我们可以把ContentProvider当成一个“网站”来看，但Android官方文档并没有这种提法。Android要求注册ContentProvider时指定authorities属性，该属性的值就相当于该网站的域名。但需要提醒读者：面试时千万不要这么说，因为你的面试官可能由于自身技术层次看不到这个高度，而并不认同这个说法，从而导致面试失败。

当我们通过上面配置文件注册了DictProvider之后，其他应用程序就可通过该Uri来访问DictProvider所暴露的数据了。

那么DictProvider到底如何暴露它所提供的数据呢？其实很简单，应用程序对数据的操作无非就是CRUD操作，因此DictProvider除了需要继承ContentProvider之外，还需要提供如下几个方法。

➢ **public boolean onCreate（）：**该方法在ContentProvider创建后会被调用，当其他应用程序第一次访问 ContentProvider 时，该 ContentProvider 会被创建出来，并立即回调该onCreate（）方法。

➢ **public Uri insert（Uri uri，ContentValues values）**：根据该Uri插入values对应的数据。

➢ **public int delete（Uri uri，String selection，String[] selectionArgs）**：根据Uri删除selection条件所匹配的全部记录。

➢ **public int update（Uri uri，ContentValues values，String selection，String[]selectionArgs）**：根据Uri修改selection条件所匹配的全部记录。

➢ **public Cursor query（Uri uri，String[] projection，String selection，String[] selectionArgs，String sortOrder）**：根据Uri查询出selection条件所匹配的全部记录，其中projection就是一个列名列表，表明只选择出指定的数据列。

➢ **public String getType（Uri uri）**：该方法用于返回当前Uri所代表的数据的MIME类型。如果该 Uri 对应的数据可能包括多条记录，那么 MIME 类型字符串应该以vnd.android.cursor.dir/开头；如果该Uri对应的数据只包含一条记录，那么MIME类型字符串应该以vnd.android.cursor.item/开头。

通过介绍不难发现，对于ContentProvider而言，Uri是一个非常重要的概念，下面详细介绍Uri的相关知识。

9.1.2 Uri简介

在介绍Android系统的Uri之前，先来看一个最常用的互联网URL。例如，想访问疯狂Java联盟的某个页面，应该在浏览器中输入如下Uri：

对于上面这个URL，可分为如下三个部分。

➢ **http：//**：URL的协议部分，只要通过HTTP协议来访问网站，这个部分就是固定的。

➢ **www.crazyit.org**：域名部分。只要访问指定的网站，这个部分就是固定的。

➢ **ethos.php**：网站资源部分。当访问者需要访问不同资源时，这个部分是动态改变的。ContentProvider要求的Uri与此类似，例如如下Uri：

它也可分为如下三个部分。

➢ **content：//**：这个部分是Android的ContentProvider规定的，就像上网的协议默认是http：//一样。暴露ContentProvider、访问ContentProvider的协议默认是content：//。

➢ **org.crazyit.providers.dictprovider**：这个部分就是ContentProvider的authorities。系统就是由这个部分来找到操作哪个ContentProvider的。只要访问指定的ContentProvider，这个部分就是固定的。

➢ **words**：资源部分（或者说数据部分）。当访问者需要访问不同资源时，这个部分是动态改变的。

需要指出的是，Android的Uri所能表达的功能更丰富，它还可以支持如下Uri：

此时它要访问的资源为word/2，这意味着访问word数据中ID为2的记录。

还有如下形式：

此时它要访问的资源为word/2/word，这意味着访问word数据中ID为2的记录的word字段。

如果想访问全部数据，即可使用下面所示的形式：

虽然大部分使用ContentProvider所操作的数据都来自于数据库，但有时候这些数据也可来自于文件、XML或网络等其他存储方式，此时支持的Uri也可以改为如下形式：

上面的Uri表示操作word节点下的detail节点。

为了将一个字符串转换成Uri，Uri工具类提供了parse（）静态方法。例如，如下代码即可将字符串转换为Uri：

9.1.3 使用ContentResolver操作数据

前面已经提到，ContentProvider相当于一个“网站”，它的作用是暴露可供操作的数据；其他应用程序则通过ContentResolver来操作ContentProvider所暴露的数据，ContentResolver相当于HttpClient。

Context提供了如下方法来获取ContentResolver对象。

➢ **getContentResolver（）**：获取该应用默认的ContentResolver。

一旦在程序中获得了ContentResolver对象之后，接下来就可调用ContentResolver的如下方法来操作数据了。

➢ **insert（Uri url，ContentValues values）**：向Uri对应的ContentProvider中插入values对应的数据。

➢ **delete（Uri url，String where，String[] selectionArgs）**：删除Uri对应的ContentProvider中where提交匹配的数据。

➢ **update（Uri uri，ContentValues values，String where，String[] selectionArgs）**：更新Uri对应的ContentProvider中where提交匹配的数据。

➢ **query（Uri uri，String[] projection，String selection，String[] selectionArgs，StringsortOrder）**：查询Uri对应的ContentProvider中where提交匹配的数据。

一般来说，ContentProvider是单实例模式的，当多个应用程序通过ContentResolver来操作ContentProvider提供的数据时，ContentResolver调用的数据操作将会委托给同一个ContentProvider处理。

9.2 开发ContentProvider

对于许多初学者而言，理解ContentProvider暴露数据的方式是一个难点。ContentProvider总是需要与ContentResolver结合使用，ContentProvider负责暴露数据，因此ContentProvider需要与ContentResolver结合学习。

9.2.1 ContentProvider与ContentResolver的关系

从ContentResolver、ContentProvider和Uri的关系来看，无论是ContentResolver，还是ContentProvider，它们所提供的CRUD方法的第一个参数都是Uri。也就是说，Uri是ContentResolver和ContentProvider进行数据交换的标识。ContentResolver对指定Uri执行CRUD等数据操作，但Uri并不是真正的数据中心，因此这些CRUD操作会委托给该Uri对应的ContentProvider来实现。通常来说，假如A应用通过ContentResolver执行CRUD操作，这些CRUD操作都需要指定Uri参数，Android系统就根据该Uri找到对应的ContentProvider（该ContentProvider通常属于B应用），ContentProvider则负责实现CRUD方法，完成对底层数据的增、删、改、查等操作，这样就可以让A应用访问、修改B应用的数据了。

ContentResolver、Uri、ContentProvider三者之间的关系如图9.1所示。

图9.1 ContentRosolver.Uri与ContentProvider的关系

从图9.1可以看出，以指定Uri为标识，ContentResolver可以实现"间接调用"ContentProvider的CRUD方法。

➢ 当 A 应用调用 ContentResolver 的 insert（）方法时，实际上相当于调用了该 Uri 对应的ContentProvider（该ContentProvider属于B应用）的insert（）方法。

➢ 当A应用调用ContentResolver的update（）方法时，实际上相当于调用了该Uri对应的ContentProvider（该ContentProvider属于B应用）的update（）方法。

➢ 当A应用调用ContentResolver的delete （）方法时，实际上相当于调用了该Uri对应的ContentProvider（该ContentProvider属于B应用）的delete（）方法。

➢ 当A应用调用ContentResolver的query （）方法时，实际上相当于调用了该Uri对应的ContentProvider（该ContentProvider属于B应用）的query（）方法。

通过上面这种关系，即可实现让A应用访问、使用B应用底层的数据。

9.2.2 开发ContentProvider子类

开发ContentProvider只要如下两步。

1 开发一个ContentProvider子类，该子类需要实现query（）、insert（）、update（）和delete（）等方法。

2 在AndroidManifest.xml文件中注册该ContentProvider，指定android：authorities属性。

在上面两步中，ContentProvider子类实现的query（）、insert（）、update（）和delete（）方法，并不是给该应用本身调用的，而是供其他应用来调用的。正如前面提到的，当其他应用通过ContentResolver调用query（）、insert（）、update（）和delete（）方法执行数据访问时，实际上就是调用指定Uri对应的ContentProvider的query（）、insert（）、update（）和delete（）方法。

如何实现ContentProvider的query（）、insert（）、update（）和delete（）方法，完全由程序员决定一程序员想怎么暴露该应用数据，就怎么实现这4个方法。在极端情况下，开发者只对这些方法提供空实现也是可以的。

例如下面的示例ContentProvider，该ContentProvider虽然实现了query（）、insert（）、update（）和delete（）方法，但并未真正对底层数据进行访问，只是输出了一行字符串。下面是该ContentProvider的代码。

程序清单：
codes\09\9.2\FirstProvider\app\src\main\java\org\crazyit\content\FirstProvider.java

上面4个粗体字方法实现了query（）、insert（）、update（）和delete（）方法，这4个方法用于供其他应用通过ContentRosovler调用。

在该ContentProvider供其他应用调用之前，还需要先配置该ContentProvider。

9.2.3 配置ContentProvider

Android应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加了<provider.../>子元素即可配置ContentProvider。例如如下的配置片段：

从上面的配置片段可以看出，配置ContentProvider时通常指定如下属性。

➢ **name**：指定该ContentProvider的实现类的类名。

➢ **authorities**：指定该ContentProvider对应的Uri（相当于为该ContentProvider分配一个域名）。

➢ **android**：**exported**：指定该ContentProvider是否允许其他应用调用。如果将该属性设为false，那么该ContentProvider将不允许其他应用调用。

为了配置上面的ContentProvider，只要在<application.../>元素中添加如下子元素即可。

程序清单：
codes\09\9.2\FirstProvider\app\src\main\AndroidManifest.xml

上面的配置指定了该ContentProvider被绑定到“content：//org.crazyit.providers.firstprovider”—这意味着当其他应用的ContentResovler向该Uri执行query（）、insert（）、update（）和delete（）方法时，实际上是调用该ContentProvider的query（）、insert（）、update（）和delete（）方法。

➢ ContentResovler调用方法时参数将会传给该ContentProvider的query（）、insert（）、update（）和delete（）方法。

➢ ContentResovler 调用方法的返回值，也就是 ContentProvider 执行query（）、insert（）、update（）和delete（）方法的返回值。

提示：

由于该应用并未提供Activity，因此该应用没有任何界面。在Android Studio中运行该应用时，Android Studio可能显示如图9.2所示的提示窗口，只要按图所示方式选择不加载任何Activity即可正常部署该应用。

图9.2 选择不加载任何Activity

9.2.4 使用ContentResolver调用方法

Context提供了getContentResovler（）方法，这表明Activity、Service等组件都可通过getContentResovler（）方法获取ContentResolver对象。

获取了ContentResovler对象之后，接下来就可调用ContentResolver的query（）、insert（）、update（）和delete（）方法了—实际上是指定Uri对应的ContentProvider的query（）、insert（）、update（）和delete（）。

下面示例的界面布局文件中只包含了4个按钮，分别用于激发调用ContentProvider的query（）、insert（）、update（）和delete（）方法。由于该示例的界面布局文件很简单，故此处不再给出代码。

下面是该示例的Activity代码。

程序清单：
codes\09\9.2\FirstResolver\app\src\main\java\org\crazyit\resolver\MainActivity.java

上面的4行粗体字代码通过ContentResolver调用query（）、insert（）、update（）和delete（）方法，实际上就是调用uri参数对应的ContentProvider的query（）、insert（）、update（）和delete（）方法，也就是前面的FirstProvider的query（）、insert（）、update（）和delete（）方法。

运行上面的程序，并依次单击该应用的4个按钮，将可以看到LogCat生成如图9.3所示的输出。

从图9.3所示的输出可以看出，当Android应用通过ContentResolver调用query（）、insert（）、update（）和delete（）方法时，实际上就是调用FirstProvider（该Uri对应的ContentProvider）的query（）、insert（）、update（）和delete（）方法。

当用户单击程序界面上的“插入”按钮时，将可以在程序界面中看到如图9.4所示的输出。

图9.3 C ontentProvider的方法被调用

图9.4 通过ContentResolver调用远程ContentProvider的方法

从图9.4可以看出，调用ContentResovler的insert（）方法的返回值为null，这是因为FirstProvider实现insert（）方法时返回了null。

9.2.5 创建ContentProvider的说明

通过上面的介绍可以看出，ContentProvider不像Activity存在复杂的生命周期，ContentProvider只有一个onCreate（）生命周期方法—当其他应用通过ContentResolver第一次访问该ContentProvider时，onCreate（）方法将会被回调，onCreate（）方法只会被调用一次；ContentProvider提供的query（）、insert（）、update（）和delete（）方法则由其他应用通过ContentResolver调用。

开发ContentProvider时所实现的query（）、insert（）、update（）和delete（）方法的第一个参数为Uri，该参数由ContentResolver调用这些方法时传入。

前面介绍的示例由于并未真正对数据进行操作，因此ContenProvider并未对Uri参数进行任何判断。实际上，为了确定该ContentProvider实际能处理的Uri，以及确定每个方法中Uri参数所操作的数据，Android系统提供了UriMatcher工具类。

UriMatcher工具类主要提供了如下两个方法。

➢ **void addURI（String authority，String path，int code）：**该方法用于向UriMatcher对象注册Uri。其中authority和path组合成一个Uri，而code则代表该Uri对应的标识码。

➢ **int match（Uri uri）：**根据前面注册的Uri来判断指定Uri对应的标识码。如果找不到匹配的标识码，该方法将会返回-1。

例如，我们先通过如下代码来创建UriMatcher对象：

上面创建的UriMatcher对象注册了两个Uri，其中 org.crazyit.providers.dictprovider/words对应的标识码为1； org.crazyit.providers.dictprovider/word/＃对应的标识码为2，＃为通配符。

这就意味着如下匹配结果：

至于到底需要为UriMatcher对象注册多少个Uri，则取决于系统的业务需求。

对于content：//org.crazyit.providers.dictprovider/words这个Uri，它的资源部分为words，这种资源通常代表了访问所有数据项；对于content：//org.crazyit.providers.dictprovider/word/2这个Uri，它的资源部分通常代表访问指定数据项，其中最后一个数值往往代表了该数据的ID。

除此之外，Android还提供了一个ContentUris工具类，它是一个操作Uri字符串的工具类，提供了如下两个工具方法。

➢ **withAppendedId（uri，id）：**用于为路径加上ID部分。例如：

➢ **parseId（uri）**：用于从指定Uri中解析出所包含的ID值。例如：

掌握上面的知识之后，接下来我们将对前面介绍的生词本应用进行改进，增加ContentProvider，这样即可允许其他应用程序通过ContentResolver来操作生词本应用的数据。

实例：使用ContentProvider共享生词本数据

正如前面访问系统ContentProvider所见到的，系统一般都会把ContentProvider的Uri、数据列等信息以常量的形式公开出来，以方便访问。为此，我们也为该ContentProvider定义一个工具类，该类中只是包含一些public static的常量。该工具类的代码如下。

程序清单：
codes\09\9.2\DictProvider\app\src\main\java\org\crazyit\content\Words.java

上面的工具类只是定义了一些简单的常量，这个工具类的作用就是告诉其他应用程序，访问该ContentProvider的一些常用入口。

提示：

实际上我们也可以不提供Words工具类，而是在ContentProvider中直接使用字符串来定义提供服务的Uri；接下来再采用文档去告诉其他应用程序访问该ContentProvider的入口。但这种方式的可维护性并不好，因此在实际项目中（包括Android系统的ContentProvider）都会通过采用工具类来定义各种常量的方式进行处理。

本实例只是在前面介绍的生词本应用的基础上进行修改，因此本应用可以直接使用前面所开发的SQLiteOpenHelper类。

接下来我们开发一个ContentProvider子类，并重写其中的增、删、改、查等方法。类代码如下。

程序清单：
codes\09\9.2\DictProvider\app\src\main\java\org\crazyit\content\DictProvider.java

上面的DictProvider类很简单，它除了继承系统的ContentProvider之外，还实现了操作数据的增、删、改、查等方法，这样该ContentProvider就开发完成了。DictProvider真正调用了SQLiteDatabase对底层数据执行增、删、改、查，当其他应用通过ContentResolver来调用该DictProvider的query（）、insert（）、update（）和delete（）方法时，就可以真正访问、操作该ContentProvider所在应用的底层数据了。

接下来需要在AndroidManifest.xml文件中注册该ContentProvider，这就需要在AndroidManifest.xml文件中增加如下配置片段：

至此，暴露生词本数据的ContentProvider开发完成。为了测试该ContentProvider的开发是否成功，接下来再开发一个应用程序，该应用程序将会通过ContentResolver来操作生词本中的数据。

该程序同样提供了添加生词、查询生词的功能，只是该程序自己并不保存数据，而是访问前面DictProvider所共享的数据。下面是使用ContentResolver的类的代码。

程序清单：
codes\09\9.2\DictResolver\app\src\main\java\org\crazyit\resolver\MainActivity.java

上面程序中的第一行粗体字代码用于向DictProvider所共享的数据中添加记录；第二行粗体字代码则用于查询DictProvider所共享的数据。

运行该程序需要先部署前面介绍的DictProvider，因为该程序所操作的数据实际上来自于DictProvider。当用户通过该程序来添加生词时，实际上是添加到了前面所介绍的DictProvider应用中；当用户通过该程序来查询生词时，实际上是查询DictProvider应用中的生词。

9.3 操作系统的ContentProvider

实际上，Android系统本身提供了大量的ContentProvider，例如联系人信息、系统的多媒体信息等，开发者自己开发的Android应用也可通过ContentResolver来调用系统ContentProvider提供的query（）、insert（）、update（）和delete（）方法，这样开发者即可通过这些ContentProvider来获取Android内部的数据了。

使用ContentResolver操作系统ContentProvider数据的步骤依然是两步。

1 调用Context的getContentResolver（）获取ContentResolver对象。

2 根据需要调用ContentResolver的insert（）、delete（）、update（）和query方法操作数据。

为了操作系统提供的ContentResolver，需要了解该ContentProvider的Uri，以及该ContentProvider所操作的数据列的列名，可以通过查阅Android官方文档来获取这些信息。

9.3.1 使用ContentProvider管理联系人

Android系统提供了Contacts应用程序来管理联系人，而且Android系统还为联系人管理提供了ContentProvider，这就允许其他应用程序以ContentResolver来管理联系人数据。

Android系统用于管理联系人的ContentProvider的几个Uri如下。

➢ **ContactsContract.Contacts.CONTENT_URI**：管理联系人的Uri。

➢ **ContactsContract.CommonDataKinds.Phone.CONTENT_URI**：管理联系人的电话的Uri。

➢ **ContactsContract.CommonDataKinds.Email.CONTENT_URI**：管理联系人的 E-mail的Uri。

了解了联系人管理ContentProvider的Uri之后，接下来就可在应用程序中通过ContentResolver去操作系统的联系人数据了。

下面示例程序中包含两个按钮，其中一个按钮用于查询系统的联系人数据，该按钮所绑定的事件监听器代码如下。

程序清单：
codes\09\9.3\ContactProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

上面程序中的第一行粗体字代码使用ContentResolver向ContactsContract.Contacts.CONTENT_URI查询数据，将可以把系统中

所有联系人信息查询出来；第二行粗体字代码使用ContentResolver向ContactsContract.CommonDataKinds.Phone.CONTENT_URI查询数据，用于查询指定联系人的电话信息；第三行粗体字代码使用ContentResolver向ContactsContract.CommonDataKinds.Email.CONTENT_URI查询数据，用于查询指定联系人的E-mail信息。

上面的程序通过ContentResolver将联系人信息、电话信息、E-mail信息查询出来之后，使用了一个ExpandableListView来显示所有联系人信息。

运行该程序，并单击“查询”按钮，程序将会使用ExpandableListView显示所有联系人信息，如图9.5所示。

图9.5 查询联系人信息

需要指出的是，上面的应用程序需要读取、添加联系人信息，因此要记得在AndroidManifest.xml文件中为该应用程序授权。也就是在该文件的根元素中添加如下元素：

该程序界面上提供了三个文本框，用户可以在这三个文本框中输入联系人名字、电话号码、E-mail地址，然后单击“添加”按钮。该按钮绑定事件监听器的代码如下。

程序清单：
codes\09\9.3\ContactProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

上面程序中的第一行粗体字代码通过ContentResolver添加一个联系人记录；第二行粗体字代码通过ContentResolver为指定联系人添加一个电话号码；第三行粗体字代码通过ContentResolver为指定联系人添加一个E-mail地址。

如果用户在程序的三个文本框中分别输入联系人姓名、电话号码、E-mail地址，并单击“添加”按钮，将可以看到该联系人信息被添加到系统中。启动Android系统的Contacts程序，可以看到如图9.6所示的界面。

单击图9.6所示界面中的“沙和尚”列表项，系统将打开该联系人的联系方式详情，如图9.7所示。

图9.6 通过ContentResolver添加联系人

图9.7 通过ContentResolver添加联系方式详情

从图9.7中可以看出，前面在程序中添加的mobile类型的电话号码、work类型的E-mail地址都显示出来了，这就表明通过ContentResolver添加联系人成功。

提示：

可能有读者对mobile类型的电话号码、work类型的E-mail地址感到迷惑，这是因为一个联系人可能有多个电话号码，比如移动电话（mobile类型）、工作电话（work类型）、家庭电话（home类型）等，E-mail地址也与此类似。实际上，好像只有早期手机上每个联系人只能添加一个电话，现在的手机应该都支持为一个联系人关联多个电话号码。

9.3.2 使用ContentProvider管理多媒体内容

Android提供了Camera程序来支持拍照、拍摄视频，用户拍摄的照片、视频都将存放在固定的位置。有些时候，其他应用程序可能需要直接访问Camera所拍摄的照片、视频等，为了处理这种需求，Android同样为这些多媒体内容提供了ContentProvider。

Android为多媒体提供的ContentProvider的Uri如下。

➢ **MediaStore.Audio.Media.EXTERNAL_CONTENT_URI**：存储在外部存储器（SD卡）上的音频文件内容的ContentProvider的Uri。

➢ **MediaStore.Audio.Media.INTERNAL_CONTENT_URI**：存储在手机内部存储器上的音频文件内容的ContentProvider的Uri。

➢ **MediaStore.Images.Media.EXTERNAL_CONTENT_URI**：存储在外部存储器（SD卡）上的图片文件内容的ContentProvider的Uri。

➢ **MediaStore.Images.Media.INTERNAL_CONTENT_URI**：存储在手机内部存储器上的图片文件内容的ContentProvider的Uri。

➢ **MediaStore.Video.Media.EXTERNAL_CONTENT_URI**：存储在外部存储器（SD卡）上的视频文件内容的ContentProvider的Uri。

➢ **MediaStore.Video.Media.INTERNAL_CONTENT_URI**：存储在手机内部存储器上的视频文件内容的ContentProvider的Uri。

下面示例程序界面中包含两个按钮，一个“查看”按钮，用于查看多媒体数据中的所有图片；一个“添加”按钮，用于向多媒体数据中添加图片。

该程序中“查看”按钮所绑定的事件监听器的代码如下。

程序清单：
codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

该程序中粗体字代码使用ContentResolver向MediaStore.Audio.Images.EXTERNAL_CONTENT_URI查询数据，这将查询出所有位于外部存储器上的图片信息。查询出图片信息之后，该程序使用了一个ListView来显示这些图片信息。

本应用需要读取外部存储设备中的多媒体信息，因此必须为该应用增加读取外部存储设备的权限。而且该应用还需要向外部存储设备写入图片，因此也需要增加写入外部存储设备的权限。也就是在AndroidManifest.xml文件中增加如下配置：

当用户单击“查看”按钮之后，系统将会显示如图9.8所示的列表。

注意图9.8所示列表中的列表项，该程序为该列表的列表项单击事件绑定了事件监听器，代码如下。

图9.8 查看系统图片列表

程序清单：
codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

正如上面的程序中粗体字代码所示，当用户单击指定列表项之后，程序会使用一个包含ImageView的对话框来显示该列表项所对应的图片。当用户单击指定列表项之后，系统将显示如图9.9所示的对话框。

图9.9 查看图片

提示：

通过该示例程序，可以非常方便地实现Android图片浏览器：Android图片浏览器并不需要程序员去遍历SD卡上的图片文件，直接查询系统多媒体ContentProvider即可获取系统中所有的图片信息。

该程序中的"添加"按钮用于将本程序中的指定图片添加到多媒体数据中—实际上完全可以把任意图片添加到多媒体数据中。程序为"添加"按钮绑定事件监听器的代码如下。

程序清单：
codes\09\9.3\MediaProviderTest\app\src\main\java\org\crazyit\content\MainActivity.java

上面程序中的粗体字代码使用了ContentResolver向MediaStore.Audio.Images.EXTERNAL_CONTENT_URI插入一条记录，但此时还未把真正的图片数据插入进去。程序中①号粗体字代码打开了刚插入的图片的Uri对应的OutputStream，接下来程序调用Bitmap的compress方法把实际的图片内容保存到OutputStream中，这样就会把实际图片内容存入系统。

9.4 监听ContentProvider的数据改变

前面介绍的是当ContentProvider将数据共享出来之后，ContentResolver会根据业务需要去主动查询ContentProvider所共享的数据。有些时候，应用程序需要实时监听ContentProvider所共享数据的改变，并随着ContentProvider的数据的改变而提供响应，这就需要利用ContentObserver了。

9.4.1 ContentObserver简介

前面介绍开发ContentProvider时，不管实现insert、delete、update方法中的哪一个，只要该方法导致ContentProvider数据的改变，程序就调用如下代码：

这行代码可用于通知所有注册在该Uri上的监听者：该ContentProvider所共享的数据发生了改变。

为了在应用程序中监听ContentProvider数据的改变，需要利用Android提供的ContentObserver基类。监听ContentProvider数据改变的监听器需要继承ContentObserver类，并重写该基类所定义的onChange（boolean selfChange）方法—当它所监听的ContentProvider数据发生改变时，该onChange（）方法将会被触发。

为了监听指定ContentProvider的数据变化，需要通过ContentResolver向指定Uri注册ContentObserver监听器。ContentResolver提供了如下方法来注册监听器。

➢ **registerContentObserver (Uri uri,boolean notifyForDescendents,ContentObserverobserver)**

该方法中的三个参数说明如下。

➢ **uri**：该监听器所监听的ContentProvider的Uri。

➢ **notifyForDescendents**：如果该参数设为true，假如注册监听的Uri 为content：//abc，那么 Uri 为 content：//abc/xyz、content：//abc/xyz/foo 的数据改变时也会触发该监听器；如果该参数设为false，假如注册监听的Uri为content：//abc，那么只有content：//abc的数据发生改变时才会触发该监听器。

➢ **observer**：监听器实例。

例如，如下代码片段可用于为指定Uri注册监听器：

上面的代码中SmsObserver就是ContentObserver的子类。

实例：监听用户发出的短信

本实例通过监听Uri为content：//sms的数据改变即可监听到用户短信的数据改变，并在监听器的onChange（boolean selfChange）方法里查询Uri为content：//sms/outbox的数据，这样即可获取用户正在发送的短信（用户正在发送的短信保存在发件箱内）。

该程序代码如下。

程序清单：
codes\09\9.4\MonitorSms\app\src\main\java\org\crazyit\content\MainActivity.java

上面程序中的第一行粗体字代码用于监听Uri为content：//sms的数据改变，就可以监听到用户短信数据的改变；第二行粗体字代码用于查询content：//sms/outbox的全部数据，也就是查询发件箱（正在发送的短信保存在发件箱内）内的全部短信，这样就可获取用户正在发送的短信详情。

运行该程序，在不关闭该程序的情况下打开Android系统内置的“Messaging”程序发送短信一直接向本机号码发送即可。当用户发送短信时，可以在LogCat面板看到如图9.10所示的输出。

图9.10 监听用户发送的短信

本程序需要读取系统短信的内容，因此还需要为该应用增加读取短信的权限，也就是需要在AndroidManifest.xml文件中增加如下配置：

提示：

由于Genymotion模拟器并未提供模拟发送短信的功能，因此本实例需要使用Android自带的AVD模拟器进行测试。

这个监听用户发送短信的程序采用Activity来实现并不合适—因为用户必须先主动打开该Activity，并在保持该Activity不关闭的情况下，用户所发送的短信才会被监听到。这显然不太符合实际需求场景，在实际情况下，可能更希望该程序以后台进程的方式"不知不觉"地运行，这就需要利用Android的Service组件，接下来的章节将介绍Android提供的Service组件。

9.5 本章小结

本章主要介绍了Android系统中ContentProvider组件的功能和用法，ContentProvider的本质就像一个"小网站"，它可以把应用程序的数据按照"固定规范"暴露出来，其他应用程序就可通过ContentProvider暴露的接口来操作内部的数据了。可以这样说，ContentProvider是Android系统内不同程序之间进行数据交换的标准接口。学习本章需要重点掌握三个API的用法：ContentResolver、ContentProvider和ContentObserver，其中Content Resolver用于操作ContentProvider提供的数据；ContentObserver用于监听ContentProvider的数据改变；而ContentProvider则是所有ContentProvider组件的基类。

第10章 Service与BroadcastReceiver

本章要点

Service组件的作用和意义

创建、配置Service

启动、停止Service

绑定本地Service并与之通信

Service的生命周期

IntentService的功能和用法

TelephonyManager的功能和用法

监听手机电话

SmsManager的功能和用法

监听手机短信

AudioManager的功能和用法

Vibrator的功能和用法

AlarmManager的功能和用法

BroadcastReceiver组件的作用和意义

开发、配置BroadcastReceiver组件

发送广播、发送有序广播

使用BroadcastReceiver接收系统广播

Service是Android四大组件中与Activity最相似的组件，它们都代表可执行的程序，Service与Activity的区别在于：Service一直在后台运行，它没有用户界面，所以绝不会到前台来。一旦Service被启动起来之后，它就与Activity一样，它完全具有自己的生命周期。关于程序中Activity与Service的选择标准是：如果某个程序组件需要在运行时向用户呈现某种界面，或者该程序需要与用户交互，就需要使用Activity；否则就应该考虑使用Service了。

开发者开发Service的步骤与开发Activity的步骤很像，开发Service组件需要先开发一个Service子类，然后在AndroidManifest.xml文件中配置该Service，配置时可通过<intent-filter.../>元素指定它可被哪些Intent启动。

Android系统本身提供了大量的Service组件，开发者可通过这些系统Service来操作Android系统本身。

本章还将向读者介绍BroadcastReceiver组件。BroadcastReceiver组件就像一个全局的事件监听器，只不过它用于监听系统发出的Broadcast。通过使用BroadcastReceiver，即可在不同应用程序之间通信。

10.1 Service简介

Service组件也是可执行的程序，它也有自己的生命周期。创建、配置Service与创建、配置Activity的过程基本相似，下面详细介绍Android Service的开发。

10.1.1 创建、配置Service

就像开发Activity需要两个步骤：① 开发Activity子类；② 在AndroidManifest.xml文件中配置Activity。开发Service也需要两个步骤。

1 定义一个继承Service的子类。

2 在AndroidManifest.xml文件中配置该Service。

Service与Activity还有一点相似之处，它们都是从Context派生出来的，因此它们都可调用Context里定义的如getResources（）、getContentResolver（）等方法。

与Activity相似的是，Service中也定义了一系列生命周期方法，如下所示。

➢ **IBinder onBind（Intent intent）**：该方法是Service子类必须实现的方法。该方法返回一个IBinder对象，应用程序可通过该对象与Service组件通信。

➢ **void onCreate（）**：在该Service第一次被创建后将立即回调该方法。

➢ **void onDestroy（）**：在该Service被关闭之前将会回调该方法。

➢ **void onStartCommand（Intent intent，int flags，int startId）**：该方法的早期版本是void onStart （Intent intent，int startId），每次客户端调用startService（Intent）方法启动该Service时都会回调该方法。

➢ **boolean onUnbind（Intent intent）**：当该Service上绑定的所有客户端都断开连接时将会回调该方法。

下面的类定义了一个Service组件。

程序清单：
codes\10\10.1\FirstService\app\src\main\java\org\crazyit\service\FirstService.java

上面这个Service什么也没干—它只是重写了Service组件的onCreate（）、onStartCommand（）、onDestroy（）、onBind（）等方法，重写这些方法时只是简单地输出了一条字符串。

提示：

虽然这个Service什么都没干，但实际上它是Service组件的框架，如果希望Service组件做某些事情，那么只要在onCreate（）或onStartCommand（）方法中定义相关业务代码即可。

定义了上面的Service之后，接下来需要在AndroidManifest.xml文件中配置该Service，配置Service使用<service.../>元素。与配置Activity相似的是，配置Service时也可为<service.../>元素配置<intent-filter.../>子元素，用于说明该Service可被哪些Intent启动。

在AndroidManifest.xml文件中增加如下配置片段来配置该Service：

从上面的配置片段不难看出，配置Service与配置Activity的差别并不大，只是配置Service使用<service.../>元素，而且无须指定android:label属性—因为Service没有界面，总是位于后台运行，为该Service指定标签没有太大的意义。

当该Service开发完成之后，接下来就可在程序中运行该Service了。在Android系统中运行Service有如下两种方式。

➢ **通过Context的startService（）方法：**通过该方法启动Service，访问者与Service之间没有关联，即使访问者退出了，Service也仍然运行。

➢ **通过Context的bindService（）方法：**使用该方法启动Service，访问者与Service绑定在一起，访问者一旦退出，Service也就终止了。

下面先示范第一种方式运行Service。

10.1.2 启动和停止Service

下面的程序使用Activity作为Service的访问者，该Activity的界面中包含两个按钮，一个按钮用于启动Service，一个按钮用于关闭Service。

该Activity的代码如下。

程序清单：
codes\10\10.1\FirstService\app\src\main\java\org\crazyit\service\MainActivity.java

从上面程序中的粗体字代码不难看出，启动、关闭Service十分简单，调用Context里定义的startService（）、stopService（）方法即可启动、关闭Service。

注意：

可能有读者会注意到本程序使用了显式Intent（直接指定要启动的目标组件的实现类）来启动Service，但本书第2版使用的却是隐式Intent启动Service。这是因为从Android 5.0开始，Google要求必须使用显式Intent启动Service组件。

运行该程序，通过程序界面先启动Service，再关闭Service，将可以在Android Studio的logcat面板看到如图10.1所示的输出。

图10.1 启动、关闭Service

如果在不关闭Service的情况下，连续三次单击“启动Service”按钮，程序将会连续三次启动Service，此时在Android S tudio的logcat面板可以看到如图10.2所示的输出。

图10.2 连续启动Service

从图10.2可以看出，每当Service被创建时会回调onCreate（）方法，每次Service被启动时都会回调onStartCommand（）方法—多次启动一个已有的Service组件将不会再回调onCreate（）方法，但每次启动时都会回调onStartCommand（）方法。

10.1.3 绑定本地Service并与之通信

当程序通过startService（）和stopService（）启动、关闭Service时，Service与访问者之间基本上不存在太多的关联，因此Service和访问者之间也无法进行通信、交换数据。

如果Service和访问者之间需要进行方法调用或交换数据，则应该使用bindService（）和unbindService（）方法启动、关闭Service。

Context的bindService（）方法的完整方法签名为：bindService（Intent service，ServiceConnection conn，int flags），该方法的三个参数解释如下。

➢ **service**：该参数通过Intent指定要启动的Service。

➢ **conn**：该参数是一个ServiceConnection对象，该对象用于监听访问者与Service之间的连接情况。当访问者与 Service 之间连接成功时将回调该 ServiceConnection 对象的onServiceConnected

（ComponentName name，IBinder service）方法；当Service所在的宿主进程由于异常中止或其他原因终止，导致该 Service 与访问者之间断开连接时回调该ServiceConnection对象的onServiceDisconnected（ComponentName name）方法。

注意：

当调用者主动通过unBindService（）方法断开与Service的连接时，ServiceConnection对象的onServiceDisconnected （ComponentName name）方法并不会被调用。

➢ **flags**：指定绑定时是否自动创建Service（如果Service还未创建）。该参数可指定为0 （不自动创建）或BIND_AUTO_CREATE（自动创建）。

注意到ServiceConnection对象的onServiceConnected（）方法中有一个IBinder对象，该对象即可实现与被绑定Service之间的通信。

当开发Service类时，该Service类必须提供一个IBinder onBind（Intent intent）方法，在绑定本地Service的情况下，onBind（Intent intent）方法所返回的IBinder对象将会传给ServiceConnection对象里onServiceConnected（ComponentName name，IBinder service）方法的service参数，这样访问者就可通过该IBinder对象与Service进行通信了。

提示：

IBinder对象相当于Service组件的内部钩子，该钩子关联到绑定的Service组件，当其他程序组件绑定该Service时，Service将会把IBinder对象返回给其他程序组件，其他程序组件通过该IBinder对象即可与Service组件进行实时通信。

实际上开发时通常会采用继承Binder（IBinder的实现类）的方式实现自己的IBinder对象。

下面的程序示范了如何在Acticity中绑定本地Service，并获取Service的运行状态。该程序的Service类需要“真正”实现onBind（）方法，并让该方法返回一个有效的IBinder对象。该Service类的代码如下。

程序清单：
codes\10\10.1\BindService\app\src\main\java\org\crazyit\service\BindService.java

上面Service类的粗体字代码实现了onBind（）方法，该方法返回了一个可访问该Service状态数据（count值）的IBinder对象，该对象将被传给该Service的访问者。

上面程序中的①号代码通过继承Binder类实现了一个IBinder对象，这个MyBinder类是Service的内部类，这对于绑定本地Service并与之通信的场景是一种常见的情形。

接下来定义一个Activity来绑定该Service，并在该Activity中通过MyBinder对象访问Service的内部状态。该Activity的界面上包含三个按钮，第一个按钮用于绑定Service；第二个按钮用于解除绑定；第三个按钮则用于获取Service的运行状态。该Activity的代码如下。

程序清单：
codes\10\10.1\BindService\app\src\main\java\org\crazyit\service\MainActivity.java

上面程序中的①号粗体字代码用于在该Activity与Service连接成功时获取Service的onBind（）方法所返回的MyBinder对象；②号粗体字代码即可通过MyBinder对象来访问Service的运行状态了。

运行该程序，单击程序界面中的“绑定Service”按钮，即可看到Android Studio的logcat有如图10.3所示的输出。

在该Activity中绑定Service之后，该Activity还可通过MyBinder对象来获取Service的运行状态，如果用户单击程序界面上的“获取Service状态”按钮，即可看到如图10.4所示的输出。

图10.3 绑定Service

图10.4 访问Service的运行状态

从图10.4所示的输出可以看到，该Activity可以非常方便地访问到Service的运行状态。虽然本程序只是一个简单的示例，该Activity只是访问了Service的一个简单count值，但实际上完全可以让MyBinder去操作Service中更多的数据—到底需要访问Service的多少数据，完全取决于实际业务的需要。

提示：

对于Service的onBind（）方法所返回的IBinder对象来说，它可被当成该Service组件所返回的代理对象，Service允许客户端通过该IBinder对象来访问Service内部的数据，这样即可实现客户端与Service之间的通信。

如果我们单击程序界面上的“解除绑定”按钮，即可在Android Studio的logcat中看到如图10.5所示的输出。

图10.5 解除绑定

正如图10.5中所示，当程序调用unbindService（）方法解除对某个Service的绑定时，系统会先回调该Service的onUnbind（）方法，然后再回调onDestroy（）方法。

与多次调用startService（）方法启动Service不同的是，多次调用bindService（）方法并不会执行重复绑定。对于前一个示例程序，用户每单击“启动Service”按钮一次，系统就会回调Service的onStartCommand（）方法一次；对于这个示例程序，不管用户单击“绑定Service”按钮多少次，系统只会回调Service的onBind（）方法一次。

10.1.4 Service的生命周期

通过前面两个示例，读者应该已经大致明白Service的生命周期了。随着应用程序启动Service方式的不同，Service的生命周期也略有差异。

如果应用程序通过startService（）方法来启动Service，Service的生命周期如图10.6左边所示。

图10.6 Service的生命周期

如果应用程序通过bindService（）方法来启动Service，Service的生命周期如图10.6右边所示。

Service生命周期还有一种特殊的情形—如果Service已由某个客户端通过startService（）方法启动了，接下来其他客户端调用bindService（）方法绑定该Service后，再调用unbindService（）方法解除绑定，最后又调用bindService（）方法再次绑定到Service，这个过程所触发的生命周期方法如下。

onCreate（）→onStartCommand（）→onBind（）→onUnbind（）（重写该方法时返回了true）→onRebind（）

在上面这个触发过程中，onCreate（）是创建该Service后立即调用的，只有当该Service被创建时才会被调用；onStartCommand （）方法则是由客户端调用startService（）方法时触发的。图10.7所示的logcat显示了上面生命周期的输出。

图10.7 先启动、再绑定的Service生命周期

在图10.7所示的输出中，可以看到Service的onRebind（）方法被回调了。如果希望该方法被回调，除了需要该Service是由Activity的startService（）方法启动之外，还需要Service子类重写onUnbind（）方法时返回true。

注意：

在图10.7所示的输出中，并没有发现Service回调onDestroy（）方法，这是因为该Service并不是由Activity通过bindService（）方法启动的（该Service事先已由Activity通过startService（）方法启动了），因此当Activity调用unBindService（）方法取消与该Service的绑定时，该Service也不会终止。

由此可见，当Activity调用bindService（）绑定一个已启动的Service时，系统只是把Service内部IBinder对象传给Activity，并不会把该Service生命周期完全“绑定”到该Activity，因而当Activity调用unBindService（）方法取消与该Service的绑定时，也只是切断该Activity与Service之间的关联，并不能停止该Service组件。

10.1.5 使用IntentService

IntentService是Service的子类，因此它不是普通的Service，它比普通的Service增加了额外的功能。

先看Service本身存在的两个问题。

➢ Service不会专门启动一个单独的进程，Service与它所在应用位于同一个进程中。

➢ Service不是一条新的线程，因此不应该在Service中直接处理耗时的任务。

提示：

如果开发者需要在Service中处理耗时任务，建议在Service中另外启动一条新线程来处理该耗时任务。就像在前面BindService中所看到的，程序在BindService的onCreate（）方法中启动了一条新线程来处理耗时任务。可能有读者感到疑惑：直接在其他程序组件中启动子线程来处理耗时任务不行吗？这种方式也不可靠，由于Activity可能会被用户退出，而BroadcastReceiver的生命周期本身就很短。可能出现的情况是：在子线程还没有结束的情况下，Activity已经被用户退出了，或者BroadcastReceiver已经结束了。在Activity已经退出、BroadcastReceiver已经结束的情况下，此时它们所在的进程就变成了空进程（没有任何活动组件的进程），系统需要内存时可能会优先终止该进程。如果宿主进程被终止，那么该进程内的所有子线程也会被中止，这样就可能导致子线程无法执行完成。

而IntentService正好可以弥补Service的上述两个不足：IntentService将会使用队列来管理请求Intent，每当客户端代码通过Intent请求启动IntentService时，IntentService会将该Intent加入队列中，然后开启一条新的worker线程来处理该Intent。对于异步的startService（）请求，IntentService会按次序依次处理队列中的Intent，该线程保证同一时刻只处理一个Intent。由于IntentService使用新的worker线程处理Intent请求，因此IntentService不会阻塞主线程，所以IntentService自己就可以处理耗时任务。

归纳起来，IntentService具有如下特征。

➢ IntentService会创建单独的worker线程来处理所有的Intent请求。

➢ IntentService会创建单独的worker线程来处理onHandleIntent（）方法实现的代码，因此开发者无须处理多线程问题。

➢ 当所有请求处理完成后，IntentService 会自动停止，因此开发者无须调用 stopSelf（）方法来停止该Service。

➢ 为Service的onBind（）方法提供了默认实现，默认实现的onBind（）方法返回null。

➢ 为Service的onStartCommand（）方法提供了默认实现，该实现会将请求Intent添加到队列中。

从上面的介绍可以看出，扩展IntentService实现Service无须重写onBind（）、onStartCommand（）方法，只要重写onHandleIntent（）方法即可。

下面的示例程序界面中包含了两个按钮，分别用于启动普通Service和IntentService，两个Service都需要处理耗时任务。该程序的界面布局代码很简单，这里不再给出。主程序Activity代码如下。

程序清单：
codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MainActivity.java

上面Activity的两个事件处理方法中分别启动了MyService和MyIntentService，其中MyService是继承Service的子类，而MyIntentService则是继承IntentService的子类。

下面是MyService类的代码。

程序清单：
codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MyService.java

上面MyService在onStartCommand（）方法中使用线程暂停的方式模拟了耗时任务，该线程暂停了20秒，相当于该耗时任务需要执行20秒。由于普通Service的执行会阻塞主线程，因此启动该线程将会导致程序出现ANR（Application Not Responding）异常。

下面是MyIntentService类的代码。

程序清单：
codes\10\10.1\IntentServiceTest\app\src\main\java\org\crazyit\service\MyIntentService.java

从上面的代码可以看出，MyIntentService继承了IntentService，并不需要实现onBind（）、onStartCommand（）方法，只要实现onHandleIntent（）方法即可，在该方法中定义该Service需要完成的任务。本示例的onHandleIntent（）方法也用线程暂停的方式模拟了耗时任务，线程同样暂停了20秒。但由于IntentService会使用单独的线程来完成该耗时任务，因此启动MyIntentService不会阻塞前台线程。

运行该示例，如果单击界面上的“启动普通Service”按钮，将会激发startService（）方法，该方法将会启动MyService去执行耗时任务，此时将会导致程序UI线程被阻塞（程序界面失去响应），而且由于阻塞时间太长，因此将会看到如图10.8所示的ANR异常。

图10.8 使用普通Service执行耗时任务导致ANR异常

相反，如果调用“启动IntentService”来启动MyIntentService，虽然MyIntentService也需要执行耗时任务，但由于MyIntentService会使用

单独的worker线程，因此MyIntentService不会阻塞前台的UI线程，所以程序界面不会失去响应。

10.2 电话管理器（TelephonyManager）

TelephonyManager是一个管理手机通话状态、电话网络信息的服务类，该类提供了大量的getXxx（）方法来获取电话网络的相关信息。

在程序中获取TelephonyManager十分简单，只要调用如下代码即可：

接下来就可以通过TelephonyManager获取相关信息或者进行相关操作了。

实例：获取网络和SIM卡信息

通过TelephonyManager提供的一系列方法即可获取手机网络、SIM卡的相关信息，该程序使用了一个ListView来显示网路和SIM卡的相关信息。

该程序代码如下。

程序清单：
codes\10\10.2\TelephonyStatus\app\src\main\java\org\crazyit\manager\MainActivity.java

由于该应用需要获取手机位置和手机状态，因此程序还需要在AndroidManifest.xml文件中增加如下配置片段：

运行该程序，将可以看到如图10.9所示的输出。

图10.9 获取SIM卡信息和网络状态

TelephonyManager除了提供一系列的getXxx（）方法来获取网络状态和SIM卡信息之外，还提供了一个listen（PhoneStateListener listener，int events）方法来监听通话状态。下面通过该方法来监听手机来电信息。

实例：监听手机来电

本实例程序通过监听TelephonyManager的通话状态来监听手机的所有来电。该程序代码如下。

程序清单：
codes\10\10.2\MonitorPhone\app\src\main\java\org\crazyit\manager\MainActivity.java

上面的程序中首先创建了一个PhoneStateListener，它是一个通话状态监听器，该监听器可用于对TelephonyManager进行监听。当手机来电铃响时，程序将会把来电号码记录到文件中，如上面的粗体字代码所示。

运行上面的程序，在保证该程序运行的状态下，启动另一个模拟器呼叫该电话。接下来就可以在Monitor的File Explorer面板的data/data/org.crazyit.manager/files目录下看到一个phoneList文件，将该文件导出到计算机上，查看该文件内容即可看到如下信息：

从上面的文件内容可以看出，该程序记录了来自另一个模拟器的电话呼入信息。

如果我们把这段代码放在后台执行的Service中运行，并且设置Service组件随着系统开机自动运行，那么这种监听就可以“神不知鬼不觉”了。本章后面会介绍如何让Service随系统开机自动运行。

需要指出的是，由于该程序需要获取手机的通话状态，因此必须在AndroidManifest.xml文件中增加如下权限配置代码：

10.3 短信管理器（SmsManager）

SmsManager是Android提供的另一个非常常见的服务，SmsManager提供了一系列sendXxxMessage（）方法用于发送短信，不过就现在实际应用来看，短信通常都是普通的文本内容，也就是调用sendTextMessage（）方法进行发送即可。

实例：发送短信

本实例程序十分简单，程序提供一个文本框让用户输入收件人号码，一个文本框让用户输入短信内容，接下来单击“发送”按钮即可将短信发送出去。

程序代码如下。

程序清单：
codes\10\10.3\SendSms\app\src\main\java\org\crazyit\manager\MainActivity.java

从上面程序的粗体字代码可以看出，使用SmsManager发送短信十分简单，简单地调用sendTextMessage（）方法即可发送。

上面的程序中用到了一个PendingIntent对象，PendingIntent是对Intent的包装，一般通过调用PendingIntent的getActivity（）、getService（）、getBroadcastReceiver（）静态方法来获取PendingIntent对象。与Intent对象不同的是，PendingIntent通常会传给其他应用组件，从而由其他应用程序来执行PendingIntent所包装的“Intent”。

该程序需要调用SmsManager来发送短信，因此还需要授予该程序发送短信的权限，也就是在AndroidManifest.xml文件中增加如下代码：

实例：短信群发

短信群发也是一个十分实用的功能，逢年过节，很多人都喜欢通过短信群发向自己的朋友表示祝福。短信群发可以将一条短信同时向多个人发送。短信群发的实现十分简单，只要让程序遍历每个收件人号码并依次向每个收件人发送短信即可。

该程序提供了一个带列表框的对话框供用户选择收件人号码，代码如下。

程序清单：
codes\10\10.3\GroupSend\app\src\main\java\org\crazyit\manager\MainActivity.java

该程序的实现也很简单，程序提供了一个带列表框的对话框供用户选择群发短信的收件人号码，并使用了一个ArrayLsit<String>集合来保

存所有的收件人号码。为了实现群发功能，程序使用循环遍历ArrayList<String>中的每个号码，并使用SmsManager依次向每个号码发送短信即可，如上面的粗体字代码所示。

注意：

上面的程序不仅需要调用SmsManager发送短信，也需要访问系统的联系人信息，因此不要忘记了在AndroidManifest.xml文件中给该程序授予相应的权限。

还有一点需要指出，该程序有一个潜在的风险：该程序直接在主线程中采用循环向多个人发送短信，如果需要发送短信的人太多且网络延迟严重，群发短信就会变成一个耗时任务，此时可以考虑使用IntentService来群发短信，群发完成后通过广播通知前台Activity。

10.4 音频管理器（AudioManager）

在某些时候，程序需要管理系统音量，或者直接让系统静音，这就可借助于Android提供的AudioManager来实现。程序一样是调用getSystemService（）方法来获取系统的音频管理器，接下来就可调用AudioManager的方法来控制手机音频了。

10.4.1 AudioManager简介

在程序获取了AudioManager对象之后，接下来就可调用AudioManager的如下常用方法来控制手机音频了。

➢ **adjustStreamVolume（int streamType，int direction，int flags）**：调整手机指定类型的声音。其中第一个参数streamType指定声音类型，该参数可接受如下几个值。

➢ **STREAM_ALARM**：手机闹铃的声音。

➢ **STREAM_DTMF**：DTMF音调的声音。

➢ **STREAM_MUSIC**：手机音乐的声音。

➢ **STREAM_NOTIFICATION**：系统提示的声音。

➢ **STREAM_RING**：电话铃声的声音。

➢ **STREAM_SYSTEM**：手机系统的声音。

➢ **STREAM_VOICE_CALL**：语音电话的声音。

第二个参数指定对声音进行增大还是减小；第三个参数是调整声音时的标志，例如指定FLAG_SHOW_UI，则调整声音时显示音量进度条。

➢ **setMicrophoneMute（boolean on）**：设置是否让麦克风静音。

➢ **setMode（int mode）**：设置声音模式，可设置的值有NORMAL、RINGTONE和IN_CALL。

➢ **setRingerMode（int ringerMode）**：设置手机的电话铃声模式。可支持如下几个属性值。

➢ **RINGER_MODE_NORMAL**：正常的手机铃声。

➢ **RINGER_MODE_SILENT**：手机铃声静音。

➢ **RINGER_MODE_VIBRATE**：手机振动。

➢ **setSpeakerphoneOn（boolean on）**：设置是否打开扩音器。

➢ **setStreamMute（int streamType，boolean state）**：将手机的指定类型的声音调整为静音。其中streamType参数与adjustStreamVolume（）方法中第一个参数的意义相同。

➢ **setStreamVolume（int streamType，int index，int flags）**：直接设置手机的指定类型的音量值。其中streamType参数与adjustStreamVolume（）方法中第一个参数的意义相同。

下面将通过一个简单的实例来示范AudioManager控制手机音频。

实例：使用AudioManager控制手机音频

本程序提供了一个按钮用于播放音乐，系统使用MediaPlayer播放音乐。程序还提供了两个按钮用于控制音乐声音量的提高、降低，并使用一个ToggleButton来控制是否静音。

该程序的界面比较简单，只是包含几个简单的按钮。该程序代码如下。

程序清单：
codes\10\10.4\AudioTest\app\src\main\java\org\crazyit\manager\MainActivity.java

上面程序中的第一段粗体字代码使用AudioManager的adjustStreamVolume（）方法提高播放音乐的音量；第二段粗体字代码使用AudioManager的adjustStreamVolume（）方法降低播放音乐的音量；第三段粗体字代码则调用AudioManager的setStreamMute（）方法将音乐设为静音。

运行该程序，单击程序界面上“增大音量”或“降低音量”按钮，系统播放音乐的音量将会随之改变，如图10.10所示。

图10.10 使用AudioManager管理音频

10.5 振动器（Vibrator）

在某些时候，程序需要启动系统振动器，比如手机静音时使用振动提示用户；再比如玩游戏时，当系统碰撞、爆炸时使用振动带给用户更逼真的体验等。总之，振动是除视频、声音之外的另一种“多媒体”，充分利用系统的振动器会带给用户更好的体验。

系统获取Vibrator也是调用Context的getSystemService（）方法即可，接下来就可调用Vibrator的方法来控制手机振动了。

10.5.1 Vibrator简介

Vibrator的使用比较简单，它只有三个简单的方法来控制手机振动。

➢ **vibrate（long milliseconds）**：控制手机振动milliseconds毫秒。

➢ **vibrate（long[] pattern，int repeat）**：指定手机以pattern指定的模式振动。例如指定pattern为new int[400，800，1200，1600]，就是指定在400ms、800ms、1200ms、1600ms这些时间点交替启动、关闭手机振动器；其中repeat指定pattern数组的索引，指定对pattern数组中从repeat索引开始的振动进行循环。

➢ **cancel（）**：关闭手机振动。

掌握这些方法之后，接下来就可在程序中通过Vibrator来控制手机振动了。

10.5.2 使用Vibrator控制手机振动

下面这个程序十分简单，程序几乎不需要界面，重写了Activity的onTouchEvent（MotionEvent event）方法，这样使得用户触碰手机触摸屏时将会启动手机振动。

程序清单：
codes\10\10.5\VibratorTest\app\src\main\java\org\crazyit\manager\MainActivity.java

上面程序中的第一行粗体字代码用于获取系统的Vibrator对象，第二行粗体字代码用于控制手机振动2秒。

需要指出的是，程序控制手机振动需要得到相应的权限，因此不要忘了在AndroidManifest.xml文件中增加如下授权配置代码：

在模拟器中运行该程序看不出振动效果，建议读者将该程序部署到真机上运行。

10.6 手机闹钟服务（AlarmManager）

AlarmManager通常的用途就是用来开发手机闹钟，但实际上它的作用不止于此。它的本质是一个全局定时器，AlarmManager可在指定时间或指定周期启动其他组件（包括Activity、Service、BroadcastReceiver）。

10.6.1 AlarmManager简介

AlarmManager不仅可用于开发闹钟应用，还可作为一个全局定时器使用，在Android应用程序中也是通过Context的getSystemService（）方法来获取AlarmManager对象的。一旦程序获取了AlarmManager对象之后，就可调用它的如下方法来设置定时启动指定组件了。

➢ **set（int type，long triggerAtTime，PendingIntent operation）：**设置在triggerAtTime时间启动由operation参数指定的组件。其中第一个参数指定定时服务的类型，该参数可接受如下值。

• **ELAPSED_REALTIME**：指定从现在开始时间过了一定时间后启动operation 所对应的组件。

• **ELAPSED_REALTIME_WAKEUP**：指定从现在开始时间过了一定时间后启动operation所对应的组件。即使系统处于休眠状态也会执行operation所对应的组件。

• **RTC**：指定当系统调用System.currentTimeMillis（）方法的返回值与triggerAtTime相等时启动operation所对应的组件。

• **RTC_WAKEUP**：指定当系统调用 System.currentTimeMillis（）方法的返回值与triggerAtTime 相等时启动 operation 所对应的组件。即使系统处于休眠状态也会执行operation所对应的组件。

➢ **setInexactRepeating（int type，long triggerAtTime，long interval，PendingIntentoperation）**：设置一个非精确的周期性任务。例如，我们设置Alarm每个小时启动一次，但系统并不一定总在每个小时的开始启动Alarm服务。

➢ **setRepeating（int type，long triggerAtTime，long interval，PendingIntent operation）**：设置一个周期性执行的定时服务。

➢ **cancel（PendingIntent operation）**：取消AlarmManager的定时服务。

掌握了AlarmManager的如上功能之后，接下来我们通过两个示例来示范AlarmManager在实际开发中的用途。

10.6.2 设置闹钟

这个程序比较简单，程序提供一个按钮让用户来设置闹铃时间（单击该按钮将会打开一个时间设置对话框），当用户设置好闹铃时间之

后，即使退出该程序，到了预设时间AlarmManager也一样会启动指定组件—这是因为AlarmManager是一个全局定时器的缘故。

该程序的界面布局很简单，程序界面上只有一个简单的按钮。程序代码如下。

程序清单：
codes\10\10.6\AlarmTest\app\src\main\java\org\crazyit\manager\MainActivity.java

上面程序中的粗体字代码控制AlarmManager将会在Calendar对应的时间启动pi对应的Activity组件，而且程序设置了AlarmManager.RTC_WAKEUP选项，这意味着即使系统处于关机状态，到了系统预设时间，AlarmManager也会控制系统去执行pi对应的Activity组件。

上面的程序中AlarmManager需要启动的Activity为AlarmActivity，它是一个非常简单的Activity，甚至不需要程序界面，当该Activity加载时打开一个对话框提示闹钟时间到，并播放一段“激昂”的音乐提醒用户。AlarmActivity代码如下。

程序清单：
codes\10\10.6\AlarmTest\app\src\main\java\org\crazyit\manager\AlarmActivity.java

上面Activity的作用只是通过一个对话框、一段音乐来提醒用户：闹钟时间到了。

不要忘记在AndroidManifest.xml文件中配置AlarmActivity，如果用户忘记配置该Activity，系统甚至不会提示用户：该Activity不存在！只是

到了系统预设时间之后，程序将什么也不干。

运行该程序，简单地把闹钟时间设为下一分钟，即使我们退出该程序，过一分钟之后也将看到如图10.11所示的闹钟提示。

图10.11 闹钟应用

需要说明的是，从Android 4.4（API 19）开始，AlarmManager的机制是非准确激发的，操作系统会偏移（shift）闹钟来最小化唤醒和电池消耗。不过AlarmManager新增了如下两个方法来支持精确激发。

➢ **setExact（int type，long triggerAtMillis，PendingIntent operation）**：设置闹钟将在精确的时间被激发。

➢ **setWindow（int type，long windowStartMillis，long windowLengthMillis，PendingIntentoperation）**：设置闹钟将在精确的时间段内被激发。

对于targetSdkVersion在API 19之前应用仍将继续使用以前的行为，所有闹钟都会使用精确激发。

实例：定时更换壁纸

本实例程序将会通过AlarmManager来周期性地调用某个Service，从而让系统实现定时更换壁纸的功能。

更换壁纸的API为WallpaperManager，它提供了clear（）方法来清除壁纸，还提供了如下方法来设置系统的壁纸。

➢ **setBitmap（Bitmap bitmap）**：将壁纸设置为bitmap所代表的位图。

➢ **setResource（int resid）**：将壁纸设置为resid资源所代表的图片。

➢ **setStream（InputStream data）**：将壁纸设置为data数据所代表的图片。

该程序的界面中只有两个按钮，其中一个按钮用于启动定时更换壁纸，另一个按钮用于关闭定时更换壁纸。该程序的代码如下。

程序清单：
codes\10\10.6\AlarmChangeWallpaper\app\src\main\java\org\crazyit\manager\MainActivity.java

上面程序中的粗体字代码指定程序每隔5秒执行一次pi所代表的组件。程序中pi代表了ChangeService组件，因此还需要为该应用程序提供一个ChangeService组件，该组件的代码如下。

程序清单：
codes\10\10.6\AlarmChangeWallpaper\app\src\main\java\org\crazyit\manager\ChangeService.java

上面的程序重写了Service的onStartCommand（）方法，这意味着程序每次启动该Service时都会执行onStartCommand（）方法中的代码，而onStartCommand（）方法中的粗体字代码负责更换系统的壁纸。

为了允许该程序改变壁纸，还需要在AndroidManifest.xml文件中为该程序授予相应的权限，也就是在AndroidManifest.xml文件中增加如下代码：

运行该程序，并单击程序界面上的“启动定时更换”按钮，接着退出该程序，返回到系统桌面，将可以看到系统桌面每5秒更换一次，如图10.12所示。

当然，这个程序还有值得改进的地方，这个程序所更换的壁纸只是程序预设的几张图片，如果把这个程序与前面介绍的SD卡文件浏览器结合起来，让用户可以添加SD卡中的图片作为可供更换的壁纸图片，那么这个程序就算比较完善了。

10.7 接收广播消息

Android系统的四大组件还有一种BroadcastReceiver，这种组件本质上就是一个全局监听器，用于监听系统全局的广播消息。由于BroadcastReceiver是一个全局监听器，因此它可以非常方便地实现系统中不同组件之间的通信。例如，我们希望客户端程序与startService（）方法启动的Service之间通信，就可以借助于BroadcastReceiver来实现。

图10.12 定时更换壁纸

10.7.1 BroadcastReceiver简介

BroadcastReceiver用于接收程序（包括用户开发的程序和系统内建的程序）所发出的Broadcast Intent，与应用程序启动Activity、Service相同的是，程序启动BroadcastReceiver也只需要两步。

1 创建需要启动的BroadcastReceiver的Intent。

2 调用Context的sendBroadcast（）或sendOrderedBroadcast（）方法来启动指定的BroadcastReceiver。

当应用程序发出一个Broadcast Intent之后，所有匹配该Intent的BroadcastReceiver都有可能被启动。

与Activity、Service具有完整的生命周期不同，BroadcastReceiver本质上只是一个系统级的监听器一它专门负责监听各程序所发出的Broadcast。

提示：

前面介绍的各种OnXxxListener只是程序级别的监听器，这些监听器运行在指定程序所在进程中，当程序退出时，OnXxxListener监听器也就随之关闭了。但BroadcastReceiver属于系统级的监听器，它拥有自己的进程，只要存在与之匹配的 Intent被广播出来，BroadcastReceiver就会被激发。

由于BroadcastReceiver本质上属于一个监听器，因此实现BroadcastReceiver的方法也十分简单，只要重写BroadcastReceiver的onReceive（Context context，Intent intent）方法即可。

一旦实现了BroadcastReceiver，接下来就应该指定该BroadcastReceiver能匹配的Intent，此时有两种方式。

➢ 使用代码进行指定，调用 BroadcastReceiver 的 Context 的registerReceiver

（BroadcastReceiver receiver，IntentFilter filter）方法指定。例如如下代码：

➢ 在AndroidManifest.xml文件中配置。例如如下代码：

每次系统Broadcast事件发生后，系统就会创建对应的BroadcastReceiver实例，并自动触发它的onReceive（） 方法，onReceive（）方法执行完后，BroadcastReceiver实例就会被销毁。

提示：

与Activity组件不同的是，当系统通过Intent启动指定了Activity组件时，如果系统没有找到合适的Activity组件，则会导致程序异常中止；但系统通过Intent激发BroadcastReceiver时，如果找不到合适的BroadcastReceiver组件，应用不会有任何问题。

如果BroadcastReceiver的onReceive（）方法不能在10秒内执行完成，Android会认为该程序无响应。所以不要在BroadcastReceiver的onReceive（）方法里执行一些耗时的操作；否则会弹出ANR（Application No Response）对话框。

如果确实需要根据Broadcast来完成一项比较耗时的操作，则可以考虑通过Intent启动一个Service来完成该操作。不应考虑使用新线程去完成耗时的操作，因为BroadcastReceiver本身的生命周期很短，可能出现的情况是子线程可能还没有结束，BroadcastReceiver就已经退出了。

如果BroadcastReceiver所在的进程结束了，虽然该进程内还有用户启动的新线程，但由于该进程内不包含任何活动组件，因此系统可能在内存紧张时优先结束该进程。这样就可能导致BroadcastReceiver启动的子线程不能执行完成。

10.7.2 发送广播

在程序中发送广播十分简单，只要调用Context的sendBroadcast（Intent intent）方法即可，这条广播将会启动intent参数所对应的BroadcastReceiver。

下面简单的程序示范了如何发送Broadcast、使用BroadcastReceiver接收广播。该程序的Activity界面中包含一个按钮，当用户单击该按钮时程序会向外发送一条广播。该程序的代码如下。

程序清单：
codes\10\10.7\Broadcast\app\src\main\java\org\crazyit\broadcast\MainActivity.java

上面程序中的粗体字代码用于创建一个Intent对象，并使用该Intent对象对外发送一条广播。该程序所使用的BroadcastReceiver代码如下。

程序清单：
codes\10\10.7\Broadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver.java

正如上面的程序中看到的，当符合该MyReceiver的广播出现时，该MyReceiver的onReceive（）方法将会被触发，从而在该方法中显示广播所携带的消息。

上面发送广播的程序中指定发送广播时所用的Intent的Action为org.crazyit.action.CRAZY_BROADCAST，这就需要配置上面的BroadcastReceiver应监听Action为该字符串的Intent，在AndroidManifest.xml文件中增加如下配置即可：

运行该程序，并单击程序界面中的“发送广播”按钮，将看到程序有如图10.13所示的提示。

图10.13 响应广播

10.7.3 有序广播

Broadcast被分为如下两种。

➢ **Normal Broadcast（普通广播）：** Normal Broadcast是完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，消息传递的效率比较高。但缺点是接收者不能将处理结果传递给下一个接收者，并且无法终止Broadcast Intent的传播。

➢ **Ordered Broadcast（有序广播）：** Ordered Broadcast的接收者将按预先声明的优先级依次接收Broadcast。比如A的级别高于B、B的级别高于C，那么Broadcast先传给A，再传给B，最后传给C。优先级别声明在<intent-filter.../>元素的android：priority属性中，数越大优先级别越高，取值范围为-1000～1000，也可以调用 IntentFilter 对象的setPriority（）设置优先级别。Ordered Broadcast接收者可以终止Broadcast Intent的传播，Broadcast Intent的传播一旦终止，后面的接收者就无法接收到Broadcast。另外，Ordered Broadcast的接收者可以将数据传递给下一个接收者，比如A得到Broadcast后，可以往它的结果对象中存入数据，当Broadcast传给B时，B可以从A的结果对象中得到A存入的数据。

Context提供的如下两个方法用于发送广播。

➢ **sendBroadcast（）：** 发送Normal Broadcast。

➢ **sendOrderedBroadcast（）：** 发送Ordered Broadcast。

对于Ordered Broadcast而言，系统会根据接收者声明的优先级别按顺序逐个执行接收者，优先接收到Broadcast的接收者可以终止Broadcast，调用BroadcastReceiver的abortBroadcast（）方法即可终止Broadcast。如果Broadcast被前面的接收者终止，后面的广播接收者就再也无法获取到Broadcast了。

不仅如此，对于Ordered Broadcast而言，优先接收到Broadcast的接收者可以通过setResultExtras（Bundle）方法将处理结果存入Broadcast中，然后传给下一个接收者，下一个接收者通过代码Bundle bundle=getResultExtras（true）可以获取上一个接收者存入的数据。

提示：

系统收到短信，发出的Broadcast属于Ordered Broadcast。如果想阻止用户收到短信，可以通过设置优先级，让自定义的BroadcastReceiver先获取到Broadcast，然后终止Broadcast。

接下来介绍一个发送有序广播的示例，该程序的Activity界面上只有一个普通按钮，用于发送一条有序广播。该程序代码如下。

程序清单：
codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MainActivity.java

上面程序中的粗体字代码指定了Intent的Action属性，再调用sendOrderedBroadcast（）方法来发送有序广播。对于有序广播而言，它会按优先级依次触发每个BroadcastReceiver的onReceive（）方法。

下面的程序先定义了第一个BroadcastReceiver。

程序清单：
codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver.java

上面的BroadcastReceiver不仅处理了它所接收到的消息，而且向处理结果中存入了key为first的消息，这个消息将可以被第二个

BroadcastReceiver解析出来。

上面程序中的①号粗体字代码用于取消广播，如果保持这条代码生效，那么优先级比MyReceiver低的BroadcastReceiver都将不会被触发。

在AndroidManifest.xml文件中部署该BroadcastReceiver，并指定其优先级为20，配置片段如下：

接下来为程序提供第二个BroadcastReceiver，这个BroadcastReceiver将会解析前一个BroadcastReceiver所存入的key为first的消息。该BroadcastReceiver的代码如下。

程序清单：
codes\10\10.7\SortedBroadcast\app\src\main\java\org\crazyit\broadcast\MyReceiver2.java

上面程序中的粗体字代码用于解析前一个BroadcastReceiver存入结果中的key为first的消息，在AndroidManifest.xml文件中配置该BroadcastReceiver，并指定其优先级为0。配置片段如下：

图10.14 获取前一个BroadcastReceiver存入结果中的消息

根据上面的配置可以看出，该程序中包含两个BroadcastReceiver，其中MyReceiver的优先级更高，MyReceiver2的优先级略低。如果将程序中①号粗体字代码注释掉，那么程序中MyReceiver2将可以被触发，并解析得到MyReceiver存入结果中的key为first的消息，因此看到如图10.18所示的输出。

如果不注释掉程序中①号粗体字代码，这行代码将会阻止消息广播，这样消息将传不到MyReceiver2了。

实例：基于Service的音乐播放器

前面已经提到BroadcastReceiver是一个全局监听器，因此BroadcastReceiver也就提供了让不同组件之间进行通信的新思路。比如程序有一个Activity、一个Service，而且该Service是通过startService（）方法启动起来的，在正常情况下，这个Activity与通过startService（）方法启动的Service之间无法通信，但借助于BroadcastReceiver的帮助，程序就可以实现两者之间的通信了。

本实例程序开发了一个基于Service组件的音乐盒，程序的音乐将会由后台运行的Service组件负责播放，当后台的播放状态发生改变时，程序将会通过发送广播通知前台Activity更新界面；当用户单击前台Activity的界面按钮时，系统将通过发送广播通知后台Service来改变播放状态。

前台Activity的界面很简单，它只有两个按钮，分别用于控制播放/暂停、停止；另外还有两个文本框，用于显示正在播放的歌曲名、歌手名。前台Activity的代码如下。

程序清单：
codes\10\10.7\MusicBox\app\src\main\java\org\crazyit\broadcast\MainActivity.java

上面程序的关键代码就是两段粗体字代码：

➢ 第一段粗体字代码用于响应后台Service所发出的广播，该程序将会根据广播Intent里的消息来改变播放状态，并更新程序界面中按钮的图标：当正在播放时，显示暂停图标；当正在暂停时，显示播放图

标。并根据传回来的 current 数据来更新 title、author两个文本框所显示的文本—显示当前正在播放的歌曲的歌名和歌手。

➢ 第二段粗体字代码则根据用户单击的按钮发送广播，发送广播时会把所按下的按钮的标识发送出来。发送的广播将激发后台 Service 的BroadcastReceiver，该BroadcastReceiver将会根据广播消息来改变播放状态。

与之对应的是，该程序的后台Service也一样，它会在播放状态发生改变时对外发送广播（广播将会激发前台Activity的BroadcastReceiver）；它也会采用BroadcastReceiver监听来自前台Activity所发出的广播。后台Service的代码如下。

程序清单：
codes\10\10.7\MusicBox\app\src\main\java\org\crazyit\broadcast\MusicService.java

上面程序中的粗体字代码用于接收来自前台Activity所发出的广播，并根据广播的消息内容改变Service的播放状态，当播放状态改变时，该Service对外发送一条广播，广播消息将会被前台Activity接收，前台Activity将会根据广播消息去更新程序界面。

除此之外，为了让该音乐播放器能按顺序依次播放每首歌曲，程序为MediaPlayer增加了OnCompletionListener监听器，当MediaPlayer播放完成后将让它自动播放下一首歌曲，如程序中①号代码所示。

图10.15 音乐播放器

运行该程序，将看到如图10.15示的界面。

由于该程序采用了后台Service来播放音乐，因此即使用户退出该程序，后台也依然会播放音乐，这就是该程序的独特之处。如果用户希望停止播放，只要单击图10.15所示界面中的“停止”按钮，前台Activity就会发送广播通知后台Service停止播放。

该程序用到了两个BroadcastReceiver，但已经在程序中注册了两个BroadcastReceiver所监听的IntentFilter，因此无须在AndroidManifest.xml文件中注册这两个BroadcastReceiver，只要注册该程序所用的前台Activity、后台Service即可。

10.8 接收系统广播消息

除了接收用户发送的广播之外，BroadcastReceiver还有一个重要的用途：接收系统广播。如果应用需要在系统特定时刻执行某些操作，就可以通过监听系统广播来实现。Android的大量系统事件都会对外发送标准广播。下面是Android常见的广播Action常量（具体请参考Android API文档中关于Intent的说明）。

- **ACTION_TIME_CHANGED**：系统时间被改变。
- **ACTION_DATE_CHANGED**：系统日期被改变。
- **ACTION_TIMEZONE_CHANGED**：系统时区被改变。
- **ACTION_BOOT_COMPLETED**：系统启动完成。
- **ACTION_PACKAGE_ADDED**：系统添加包。
- **ACTION_PACKAGE_CHANGED**：系统的包改变。
- **ACTION_PACKAGE_REMOVED**：系统的包被删除。
- **ACTION_PACKAGE_RESTARTED**：系统的包被重启。
- **ACTION_PACKAGE_DATA_CLEARED**：系统的包数据被清空。

➢ **ACTION_BATTERY_CHANGED**：电池电量改变。

➢ **ACTION_BATTERY_LOW**：电池电量低。

➢ **ACTION_POWER_CONNECTED**：系统连接电源。

➢ **ACTION_POWER_DISCONNECTED**：系统与电源断开。

➢ **ACTION_SHUTDOWN**：系统被关闭。

通过使用BroadcastReceiver来监听特殊的广播，即可让应用随系统执行特定的操作。

实例：开机自动运行的Service

前面已经介绍了多个程序，需要使用开机自动运行的Service，例如监听用户来电、监听用户短信、拦截黑名单电话……为了让Service随系统启动自动运行，可以让BroadcastReceiver监听Action为ACTION_BOOT_COMPLETED常量的Intent，然后在BroadcastReceiver中启动特定Service即可。

该程序所用的BroadcastReceiver代码如下。

程序清单：
codes\10\10.8\LaunchService\app\src\main\java\org\crazyit\broadcast\LaunchReceiver.java

该LaunchReceiver的代码十分简单，只要在onReceive（）方法中启动指定Service即可，关键是要让LaunchReceiver监听系统开机发出的广播，因此需要在AndroidManifest.xml文件中采用如下代码配置该BroadcastReceiver：

除此之外，为了让程序能访问系统开机事件，还需要为应用程序增加如下权限：

至于程序中用到的LaunchService，则可以是用户开发的任意Service，既可以是监听用户来电的Service，也可以是监听用户短信、拦截黑名单电话等的Service，此处不再给出该Service的代码。

实例：短信提醒

前面已经提到，当系统收到短信时，系统会对外发送一个有序广播，该广播的Intent的Action为android.provider.Telephony.SMS_RECEIVED。因此，只要在程序中开发一个对应的BroadcastReceiver即可监听到系统收到的短信。

该程序对应的BroadcastReceiver代码如下。

程序清单：
codes\10\10.8\MonitorSms\app\src\main\java\org\crazyit\broadcast\SmsReceiver.java

上面的程序重写了BroadcastReceiver的onReceive（）方法，并在该方法中获取所收到的短信内容，然后使用Toast显示短信内容。

为了让该程序在系统的短信接收程序之前被启动，我们将该BroadcastReceiver的优先级设得高一些，在AndroidManifest.xml文件中通过如下代码来配置该BroadcastReceiver：

上面的配置片段中粗体字代码指定了该BroadcastReceiver的优先级为1000，这样它就可以在系统短信接收程序之前被触发了。为了让该程

序拥有读取短信的权限，还需要在AndroidManifest.xml文件中为该程序进行授权，授权的配置代码如下：

由于该BroadcastReceiver将会位于系统短信接收程序之前被启动，如果保持该程序中①号粗体字代码生效，那么该程序接收到短信广播之后，广播将不会被传播到系统的短信接收程序—也就是系统本身将不会收到短信。

运行该程序，然后再启动另一个模拟器向该程序所在模拟器发送短信，将可以看到如图10.16所示的界面。

图10.16 短信提醒

当然，该程序也可改变成所谓的"短信窃听"程序，只要让该程序接收到短信之后并不将该短信显示出来，而是将短信存入系统中，或者再以短信的方式发送到指定手机，这样就实现了"短信窃听"功能，不过这样做是"不道德"的。

实例：手机电量提示

当手机电量发生改变时，系统会对外发送Intent的Action为ACTION_BATTERY_CHANGED常量的广播；当手机电量过低时，系统会对外发送Intent的Action为ACTION_BATTERY_LOW常量的广播。通过开发监听对应Intent的BroadcastReceiver，即可让系统对手机电量进行提示。

下面的程序即可对手机电量过低进行提示。

程序清单：
codes\10\10.8\MonitorBattery\app\src\main\java\org\crazyit\broadcast\BatteryReceiver.java

上面程序中的粗体字代码即可通过接收到的广播消息获取系统总电量、当前电量，如果当前电量小于总电量的15%，则程序会显示电量过低的提示信息。

10.9 本章小结

前面已经介绍了Activity和ContentProvider两个组件，再加上本章所介绍的Service、BroadcastReceiver两个组件，这就是Android系统的四大组件了。学习Service需要重点掌握创建、配置Service组件，以及如何启动、停止Service；不仅如此，如何开发IntentService也是需要重点掌握的内容。学习BroadcastReceiver需要掌握创建、配置BroadcastReceiver组件，还需要掌握在程序中发送Broadcast的方法。除此之外，本章还介绍了大量系统Service的功能和用法，包括TelephonyManager、SmsManager、AudioManager、Vibrator、AlarmManager等，需要读者熟练掌握并能熟练使用。

第11章 多媒体应用开发

本章要点

音频和视频

使用MediaPlayer播放音频

使用SoundPool播放音频

使用VideoView播放视频

使用MediaPlayer与SurfaceView播放视频

使用MediaRecorder录制音频

控制摄像头拍照

控制摄像头录制视频短片

Android应用面向的是普通个人用户，这些用户往往会更加关注用户体验，因此为Android应用增加动画、视频、音乐等多媒体功能十分必要。就目前的手机发展趋势来看，手机已经不再是单一的通信工具，已经发展成集照相机、音乐播放器、视频播放器、个人小型终端于一体的智能设备，因此为手机提供音频录制、播放，视频录制、播放的功能十分重要。

Android提供了常见音频、视频的编码、解码机制，就像之前所用过的MediaPlayer类，Android支持的音频格式有MP3、WAV和3GP等，支持的视频格式有MP4和3GP等。

借助于这些多媒体支持类，我们可以非常方便地在手机应用中播放音频、视频等，这些多媒体数据既可是来自于Android应用的资源文件，

也可是来自于外部存储器上的文件，甚至可以是来自于网络的文件流。不仅如此，Android也提供了对摄像头、麦克风的支持，因此也可以十分方便地从外部采集照片、视频、音频等多媒体信息。

11.1 音频和视频的播放

Android提供了简单的API来播放音频、视频，下面将会详细介绍如何使用它们。

11.1.1 使用MediaPlayer播放音频

前面已经提及如何使用MediaPlayer播放音频的例子，使用MediaPlayer播放音频十分简单，当程序控制MediaPlayer对象装载音频完成之后，程序可以调用MediaPlayer的如下三个方法进行播放控制。

➢ **start（）**：开始或恢复播放。

➢ **stop（）**：停止播放。

➢ **pause（）**：暂停播放。

为了让MediaPlayer来装载指定音频文件，MediaPlayer提供了如下简单的静态方法。

➢ **static MediaPlayer create（Context context，Uri uri）**：从指定Uri来装载音频文件，并返回新创建的MediaPlayer对象。

➢ **static MediaPlayer create（Context context，int resid）**：从resid资源ID对应的资源文件中装载音频文件，并返回新创建的MediaPlayer对象。

上面两个方法用起来非常方便，但这两个方法每次都会返回新创建的MediaPlayer对象，如果程序需要使用MediaPlayer循环播放多个音频

文件，使用 MediaPlayer的静态create（）方法就不太合适了，此时可通过MediaPlayer的setDataSource（）方法来装载指定的音频文件。MediaPlayer提供了如下方法来指定装载相应的音频文件。

➢ **setDataSource（String path）**：指定装载path路径所代表的文件。

➢ **setDataSource（FileDescriptor fd，long offset，long length）**：指定装载fd所代表的文件中从offset开始、长度为length的文件内容。

➢ **setDataSource（FileDescriptor fd）**：指定装载fd所代表的文件。

➢ **setDataSource（Context context，Uri uri）**：指定装载uri所代表的文件。

执行上面所示的setDataSource（）方法之后，MediaPlayer并未真正去装载那些音频文件，还需要调用MediaPlayer的prepare（）方法去准备音频，所谓“准备”，就是让MediaPlayer真正去装载音频文件。

因此使用已有的MediaPlayer对象装载“下一首”歌曲的代码模板为：

除此之外，MediaPlayer还提供了一些绑定事件监听器的方法，用于监听MediaPlayer播放过程中所发生的特定事件。绑定事件监听器的方法如下。

➢ **setOnCompletionListener（MediaPlayer.OnCompletionListener listener）**：为MediaPlayer的播放完成事件绑定事件监听器。

➢ **setOnErrorListener（MediaPlayer.OnErrorListener listener）**：为MediaPlayer的播放错误事件绑定事件监听器。

➢ **setOnPreparedListener（MediaPlayer.OnPreparedListener listener）**：当 MediaPlayer调用prepare（）方法时触发该监听器。

➢ **setOnSeekCompleteListener（MediaPlayer.OnSeekCompleteListener listener）**：当MediaPlayer调用seek（）方法时触发该监听器。

因此可以在创建一个MediaPlayer对象之后，通过为该MediaPlayer绑定监听器来监听相应的事件。例如如下代码：

下面简单归纳一下使用MediaPlayer播放不同来源的音频文件。

1.播放应用的资源文件

播放应用的资源文件需要两步即可。

1 调用MediaPlayer的create（Context context，int resid）方法装载指定的资源文件。

2 调用MediaPlayer的start（）、pause（）、stop（）等方法控制播放即可。例如如下代码：

提示：

音频资源文件一般放在Android应用的/res/raw目录下。

2.播放应用的原始资源文件

播放应用的资源文件按如下步骤执行。

1 调用Context的getAssets（）方法获取应用的AssetManager。

2 调用AssetManager对象的openFd（String name）方法打开指定的原始资源，该方法返回一个AssetFileDescriptor对象。

3 调用AssetFileDescriptor的getFileDescriptor（）、getStartOffset（）和getLength（）方法来获取音频文件的FileDescriptor、开始位置、长度等。

4 创建MediaPlayer对象（或利用已有的MediaPlayer对象），并调用MediaPlayer对象的setDataSource（FileDescriptor fd，long offset，long length）方法来装载音频资源。

5 调用MediaPlayer对象的prepare（）方法准备音频。

6 调用MediaPlayer的start（）、pause（）、stop（）等方法控制播放即可。

注意：

虽然MediaPlayer提供了setDataSource（FileDescriptor fd）方法来装载指定的音频资源，但实际使用时这个方法似乎有问题：不管程序调用openFd（String name）方法时指定打开哪个原始资源，MediaPlayer将总是播放第一个原始音频资源。

例如如下代码片段：

3.播放外部存储器上的音频文件

播放外部存储器上的音频文件按如下步骤执行。

1 创建MediaPlayer对象（或利用已有的MediaPlayer对象），并调用MediaPlayer对象的setDateSource（String path）方法装载指定的音频文件。

2 调用MediaPlayer对象的prepare（）方法准备音频。

3 调用MediaPlayer的start（）、pause（）、stop（）等方法控制播放即可。例如如下代码：

4.播放来自网络的音频文件

播放来自网络的音频文件有两种方式：①直接使用MediaPlayer的静态create（Context context，Uri uri）方法；②调用MediaPlayer的setDataSource（Context context，Uri uri）方法装载指定Uri对应的音频文件。

以第二种方式播放来自网络的音频文件的步骤如下。

1 根据网络上的音频文件所在的位置创建Uri对象。

2 创建MediaPlayer对象（或利用已有的MediaPlayer对象），并调用MediaPlayer对象的setDateSource（Context context，Uri uri）方法装载Uri对应的音频文件。

3 调用MediaPlayer对象的prepare（）方法准备音频。

4 调用MediaPlayer的start（）、pause（）、stop（）等方法控制播放即可。例如如下代码片段：

MediaPlayer除了调用prepare（）方法来准备声音之外，还可以调用prepareAsync（）来准备声音。prepareAsync（）与普通prepare（）方法的区别在于，prepareAsync（）是异步的，它不会阻塞当前的UI线程。

归纳起来，MediaPlayer的状态图如图11.1所示。

由于前面已经提供了大量使用MediaPlayer播放声音的例子，故此处不再介绍使用MediaPlayer的简单方法。

11.1.2 音乐特效控制

Android可以控制播放音乐时的均衡器、重低音、音场及显示音乐波形等，这些都是靠AudioEffect及其子类来完成的，它包含如下常用子类。

图11.1 M ediaPlayer的状态图

- **AcousticEchoCanceler**：取消回声控制器。
- **AutomaticGainControl**：自动增益控制器。
- **NoiseSuppressor**：噪音压制控制器。
- **BassBoost**：重低音控制器。
- **Equalizer**：均衡控制器。
- **PresetReverb**：预设音场控制器。
- **Visualizer**：示波器。

上面的子类中前三个子类的用法很简单，只要调用它们的静态create（）方法创建相应的实例，然后调用它们的isAvailable（）方法判断是否可用，再调用setEnabled（boolean enabled）方法启用相应效果即可。

下面是启用取消回声功能的示意代码：

下面是启用自动增益功能的示意代码：

下面是启用噪音压制功能的示意代码：

BassBoost、Equalizer、PresetReverb、Visualizer这4个类，都需要调用构造器来创建实例。创建实例时，同样需要传入一个audioSession参数，为了启用它们，同样需要调用AudioEffect基类的setEnabled（true）方法。

获取BassBoost对象之后，可调用它的setStrength（short strength）方法来设置重低音的强度。

获取PresetReverb对象之后，可调用它的setPreset（short preset）方法设置使用预设置的音场。Equalizer提供了getNumberOfPresets（）方法获取系统所有预设的音场，并提供了getPresetName（）方法获取预设音场名称。

获取Equalizer对象之后，可调用它的getNumberOfBands（）方法获取该均衡器支持的总频率数，再调用getCenterFreq（short band）方法根据索引来获取频率。当用户想为某个频率的均衡器设置参数值时，可调用setBandLevel（short band，short level）方法进行设置。

Visualizer对象并不用于控制音乐播放效果，它只是显示音乐的播放波形。为了实时显示该示波器的数据，需要为该组件设置一个OnDataCaptureListener监听器，该监听器将负责更新波形显示组件的界面。

下面用一个实例来示范上面4个特效类的用法。

实例：音乐的示波器、均衡、重低音和音场

本实例无须界面布局文件，使用一个LinearLayout容器来盛装一个示波器View组件，该示波器View组件将负责绘制Visualizer传过来的数据；LinearLayout添加多个SeekBar来控制Equalizer支持的所有频率的均衡值；LinearLayout还添加一个SeekBar来控制重低音的强度；LinearLayout还添加一个Spinner让用户选择预设音场。

下面是该实例的Activity代码。

程序清单：
codes\11\11.1\MediaPlayerTest\app\src\main\java\org\crazyit\media\MainActivity.java

上面的程序中定义了4个方法：setupVisualizer（）、setupEqualizer（）、setupBassBoost（）和setupPresetReverb（），这4个方法分别创建Visualizer、Equalizer、BassBoost和PresetReverb对象，并调用它们的方法控制音乐特效。

上面的实例中包含了一个MyVisualizerView内部类，该内部类会根据Visualizer传过来的数据动态绘制波形效果。本例提供了三种波形：块状波形、柱状波形和曲线波形—当用户单击MyVisualizerView组件时将会切换波形。

运行该实例，可以看到如图11.2所示的效果。

如果用户单击该程序界面最上方的示波器组件，该组件将会切换一种波形显示。单击下方的音场，将可以看到系统支持的所有预设音场，如图11.3所示。

图11.2 示波器、均衡、重低音和音场

图11.3 曲线示波器和全部预设音场

11.1.3 使用SoundPool播放音效

如果应用程序经常需要播放密集、短促的音效，这时还用MediaPlayer就显得有些不合适了。MediaPlayer存在如下缺点：

➢ 资源占用量较高，延迟时间较长。

➢ 不支持多个音频同时播放。

除了前面介绍的使用MediaPlayer播放音频之外，Android还提供了SoundPool来播放音效，SoundPool使用音效池的概念来管理多个短促的音效，例如它可以开始就加载20个音效，以后在程序中按音效的ID进行播放。

SoundPool主要用于播放一些较短的声音片段，与MediaPlayer相比，SoundPool的优势在于CPU资源占用量低和反应延迟小。另外，SoundPool还支持自行设置声音的品质、音量、播放比率等参数。

Android系统SoundPool提供了一个Builder内部类，该内部类专门用于创建SoundPool。

提示：

从Android 5.0开始，SoundPool的构造器被设为过时了，因此推荐使用SoundPool.Builder来创建SoundPool对象。

一旦得到了SoundPool对象之后，接下来就可调用SoundPool的多个重载的load（）方法来加载声音了。SoundPool提供了如下4个load（）

方法。

➢ **int load（Context context，int resId，int priority）：**从resId所对应的资源加载声音。

➢ **int load（FileDescriptor fd，long offset，long length，int priority）：**加载fd所对应的文件中从offset开始、长度为length的声音。

➢ **int load（AssetFileDescriptor afd，int priority）：**从afd所对应的文件中加载声音。

➢ **int load（String path，int priority）：**从path对应的文件去加载声音。

上面4个方法中都有一个priority参数，该参数目前还没有任何作用，Android建议将该参数设为1，保持和未来的兼容性。

上面4个方法加载声音之后，都会返回该声音的ID，以后程序就可以通过该声音的ID来播放指定声音了。SoundPool提供的播放指定声音的方法如下。

➢ **int play（int soundID，float leftVolume，float rightVolume，int priority，int loop，floatrate）：**该方法的第一个参数指定播放哪个声音；leftVolume、rightVolume 指定左、右的音量；priority指定播放声音的优先级，数值越大，优先级越高；loop指定是否循环，0为不循环，-1为循环；rate指定播放的比率，数值可从0.5到2，1为正常比率。

提示：

为了更好地管理SoundPool所加载的每个声音的ID，程序一般会使用一个HashMap <Integer，Integer>对象来管理声音。

归纳起来，使用SoundPool播放声音的步骤如下。

1 调用SoundPool.Builder的构造器创建SoundPool.Builder对象，并可通过该Builder对象为SoundPool设置属性。

2 调用SoundPool的构造器创建SoundPool对象。

3 调用SoundPool对象的load（）方法从指定资源、文件中加载声音。最好使用HashMap< Integer，Integer>来管理所加载的声音。

4 调用SoundPool的play（）方法播放声音。

下面的程序示范了如何使用SoundPool来播放音效。该程序提供了三个按钮，分别用于播放不同的声音。该程序的界面十分简单，故此处不再给出界面布局代码。程序代码如下。

程序清单：
codes\11\11.1\SoundPoolTest\app\src\main\java\org\crazyit\media\MainActivity.java

上面程序中的①号粗体字代码用于创建SoundPool对象；②号粗体字代码用于使用SoundPool加载多个不同的声音；③号粗体字代码则用于根据声音ID来播放指定声音。这就是使用SoundPool播放声音的标准过程。

实际使用SoundPool播放声音时有如下几点需要注意：SoundPool虽然可以一次性加载多个声音，但由于内存限制，因此应该避免使用SoundPool来播放歌曲或者做游戏背景音乐，只有那些短促、密集的声音才考虑使用SoundPool进行播放。

虽然SoundPool比MediaPlayer的效率好，但也不是绝对不存在延迟问题，尤其在那些性能不太好的手机中，SoundPool的延迟问题会更严重。

11.1.4 使用VideoView播放视频

为了在Android应用中播放视频，Android提供了VideoView组件，它就是一个位于android.widget包下的组件，它的作用与ImageView类似，只是ImageView用于显示图片，而VideoView用于播放视频。

使用VideoView播放视频的步骤如下。

1 在界面布局文件中定义VideoView组件，或在程序中创建VideoView组件。

2 调用VideoView的如下两个方法来加载指定视频。

➢ setVideoPath（String path）：加载path文件所代表的视频。

➢ setVideoURI（Uri uri）：加载uri所对应的视频。

3 调用VideoView的start（）、stop（）、pause（）方法来控制视频播放。

实际上与VideoView一起结合使用的还有一个MediaController类，它的作用是提供一个友好的图形控制界面，通过该控制界面来控制视频的播放。

下面的程序示范了如何使用VideoView来播放视频。该程序提供了一个简单的界面，界面布局代码如下。

程序清单：
codes\11\11.1\VideoViewTest\app\src\main\res\layout\main.xml

上面的界面布局中定义了一个VideoView组件，接下来就可以在程序中使用该组件来播放视频了。播放视频时还结合了MediaController来控制视频的播放，该程序代码如下。

程序清单：
codes\11\11.1\VideoViewTest\app\src\main\java\org\crazyit\media\MainActivity.java

上面程序中的①号粗体字代码用于让VideoView加载指定的视频文件，接下来该VideoView就可以用于播放该视频了；接下来②、③号粗体字代码用于建立VideoView与MediaController之间的关联，这样就不需要开发者自己去控制视频的播放、暂停等了，让MediaController进行控制即可。

由于该程序需要读取SD卡上的视频文件，因此还需要在AndroidManifest.xml文件中增加如下代码片段：

运行该程序，在保证/mnt/sdcard/movie.mp4视频文件存在的前提下，将可以看到如图11.4所示的界面。

图11.4 使用VedioPlayer播放视频

图11.4所示界面中的快进键、暂停键、后退键以及播放进度条就是由MediaController所提供的。

提示：

运行该程序可能会遇到一些问题，如果读者使用了一些非标准的MP4、3GP文件，那么该应用程序将无法播放，因此建议读者自行使用手机录制一段兼容各种手机的标准的MP4、3GP视频文件。

11.1.5 使用MediaPlayer和SurfaceView播放视频

使用VideoView播放视频简单、方便，但有些早期的开发者还是更喜欢使用MediaPlayer来播放视频，但由于MediaPlayer主要用于播放音频，因此它没有提供图像输出界面，此时就需要借助于SurfaceView来显示MediaPlayer播放的图像输出。

使用MediaPlayer播放视频的步骤如下。

1 创建MediaPlayer对象，并让它加载指定的视频文件。

2 在界面布局文件中定义SurfaceView组件，或在程序中创建SurfaceView组件，并为SurfaceView的SurfaceHolder添加Callback监听器。

3 调用MediaPlayer对象的setDisplay（SurfaceHolder sh）方法将所播放的视频图像输出到指定的SurfaceView组件。

4 调用MediaPlayer对象的start（）、stop（）和pause（）方法控制视频的播放。

下面的程序使用了MediaPlayer和SurfaceView来播放视频，并为该程序提供了三个按钮来控制视频的播放、暂停和停止。该程序的界面布局比较简单，故此处不再给出界面布局文件。该程序的代码如下。

程序清单：
codes\11\11.1\SurfaceViewPlayVideo\app\src\main\java\org\crazyit\media\MainActivity.java

从上面的代码不难看出，使用MediaPlayer播放视频与播放音频的步骤大同小异，关键的区别在于①号粗体字代码，设置使用SurfaceView来显示MediaPlayer播放时的图像输出。当然，由于程序需要使用

SurfaceView来显示MediaPlayer的图像输出，因此程序需要一些代码来维护SurfaceView、SurfaceHolder对象。

该程序同样需要访问SD卡上的视频文件，因此也需要在AndroidManifest.xml文件中增加如下配置：

运行上面的程序，将可以看到如图11.5所示的播放界面。

从上面的开发过程不难看出，使用MediaPlayer播放视频要复杂一些，而且需要自己开发控制按钮来控制视频播放，因此一般推荐使用VideoView来播放视频。

图11.5 使用MediaPlayer和SurfaceView播放视频

11.2 使用MediaRecorder录制音频

手机一般都提供了麦克风硬件，而Android系统就可以利用该硬件来录制音频了。

为了在Android应用中录制音频，Android提供了MediaRecorder类。使用MediaRecorder录制音频的过程很简单，按如下步骤进行即可。

1 创建MediaRecorder对象。

2 调用MediaRecorder对象的setAudioSource（）方法设置声音来源，一般传入MediaRecorder.AudioSource.MIC参数指定录制来自麦克风的声音。

3 调用MediaRecorder对象的setOutputFormat（）方法设置所录制的音频文件格式。

4 调用MediaRecorder对象的setAudioEncoder（）、setAudioEncodingBitRate（int bitRate）、setAudioSamplingRate（int samplingRate）方法设置所录制的声音编码格式、编码位率、采样率等，这些参数将可以控制所录制的声音品质、文件大小。一般来说，声音品质越好，声音文件越大。

5 调用MediaRecorder的setOutputFile（String path）方法设置所录制的音频文件的保存位置。

6 调用MediaRecorder的prepare（）方法准备录制。

7 调用MediaRecorder对象的start（）方法开始录制。

8 录制完成，调用MediaRecorder对象的stop（）方法停止录制，并调用release（）方法释放资源。

注意：

上面的步骤中第3步和第4步两个步骤千万不能搞反；否则程序将会抛出IllegalStateException异常。

图11.6显示了MediaRecorder的状态图。

图11.6 MediaRecorder的状态图

实例：录制音乐

本实例程序示范了如何使用MediaRecorder来录制声音。该程序的界面布局很简单，只提供了两个简单的按钮来控制录音开始、停止，故此处不再给出界面布局文件。程序代码如下。

程序清单：
codes\11\11.2\RecordSound\app\src\main\java\org\crazyit\media\MainActivity.java

上面的程序中大段的粗体字代码用于设置录音的相关参数，比如输出文件的格式、声音来源等。程序中①号粗体字代码控制MediaRecorder开始录音；当用户单击“停止录制”按钮时，程序中②号粗体字代码控制MediaRecorder停止录音，③号粗体字代码用于释放资源。

图11.7 录音界面

运行该程序，将看到如图11.7所示的界面。

单击图11.7所示界面中第一个按钮开始录音，单击第二个按钮则可结束录音。录音完成后可以看到/mnt/sdcard/目录下生成一个sound.amr文件，这就是刚刚录制的音频文件—Android模拟器将会直接使用宿主电脑上的麦克风，因此如果读者的电脑上有麦克风，那么该程序即可正常录制声音。

上面的程序需要使用系统的麦克风进行录音，因此需要向该程序授予录音的权限。除此之外，还需要授予该程序向外部存储设备写入数据的权限，也就是在AndroidManifest.xml文件中增加如下配置：

注意：

Android官方API已经指出，最新版的MediaRecorder目前不能在模拟器上运行，必须在真机上运行，因此读者需要在真机上测试该应用。

11.3 控制摄像头拍照

现在的手机一般都会提供相机功能，有些相机的镜头甚至支持1000万以上像素，有些甚至支持光学变焦，这些手机已经变成了专业数码相

机。为了充分利用手机上的相机功能，Android应用可以控制拍照和录制视频。

11.3.1 使用Android 5.0的Camera v2拍照

Android 5.0对拍照API进行了全新的设计，新增了全新设计的Camera v2 API，这些API不仅大幅提高了Android系统拍照的功能，还能支持RAW照片输出，甚至允许程序调整相机的对焦模式、曝光模式、快门等。

Android 5.0的Camera v2主要涉及如下API。

➢ **CameraManager**：摄像头管理器。这是一个全新的系统管理器，专门用于检测系统摄像头、打开系统摄像头。除此之外，调用CameraManager的getCameraCharacteristics（String）方法即可获取指定摄像头的相关特性。

➢ **CameraCharacteristics**：摄像头特性。该对象通过CameraManager来获取，用于描述特定摄像头所支持的各种特性。

➢ **CameraDevice**：代表系统摄像头。该类的功能类似于早期的Camera类。

➢ **CameraCaptureSession**：这是一个非常重要的API，当程序需要预览、拍照时，都需要先通过该类的实例创建Session。而且不管预览还是拍照，也都是由该对象的方法进行控制的，其中控制预览的方法为setRepeatingRequest（）；控制拍照的方法为capture（）。为了监听CameraCaptureSession的创建过程，以及监听CameraCaptureSession的拍照过程，Camera v2 API为CameraCaptureSession提供了StateCallback、CaptureCallback等内部类。

➢ **CameraRequest 和 CameraRequest.Builder**：当程序调用setRepeatingRequest（）方法进行预览时，或调用 capture（）方法

进行拍照时，都需要传入 CameraRequest 参数。CameraRequest代表了一次捕获请求，用于描述捕获图片的各种参数设置，比如对焦模式、曝光模式……总之，程序需要对照片所做的各种控制，都通过 CameraRequest 参数进行设置。CameraRequest.Builder则负责生成CameraRequest对象。

理解了上面API的功能和作用之后，接下来即可使用Camera v2 API来控制摄像头拍照了。控制拍照的步骤大致如下。

1 调用CameraManager的openCamera（String cameraId，CameraDevice.StateCallback callback，Handler handler）方法打开指定摄像头。该方法的第一个参数代表要打开的摄像头ID；第二个参数用于监听摄像头的状态；第三个参数代表执行callback的Handler，如果程序希望直接在当前线程中执行callback，则可将handler参数设为null。

2 当摄像头被打开之后，程序即可获取CameraDevice—即根据摄像头ID获取了指定摄像头设备，然后调用CameraDevice的createCaptureSession（List<Surface> outputs，CameraCaptureSession.StateCallback callback，Handler handler）方法来创建CameraCaptureSession。该方法的第一个参数是一个List集合，封装了所有需要从该摄像头获取图片的Surface，第二个参数用于监听CameraCaptureSession的创建过程；第三个参数代表执行callback的Handler，如果程序希望直接在当前线程中执行callback，则可将handler参数设为null。

3 不管预览还是拍照，程序都调用CameraDevice的createCaptureRequest（int templateType）方法创建CaptureRequest.Builder，该方法支持TEMPLATE_PREVIEW（预览）、TEMPLATE_RECORD（拍摄视频）、TEMPLATE_STILL_CAPTURE（拍照）等参数。

4 通过第3步所调用方法返回的CaptureRequest.Builder设置拍照的各种参数，比如对焦模式、曝光模式等。

5 调用CaptureRequest.Builder的build（）方法即可得到CaptureRequest对象，接下来程序可通过CameraCaptureSession的setRepeatingRequest（）方法开始预览，或调用capture（）方法拍照。

实例：拍照时自动对焦

本实例示范了使用Camera v2来进行拍照。当用户按下拍照键时，该应用会自动对焦，当对焦成功时拍下照片。该程序的界面中提供了一个自定义TextureView来显示预览取景，十分简单。该自定义TextureView类的代码如下。

程序清单：
codes\11\11.3\CameraV2Test\app\src\main\java\org\crazyit\media\MainActivity.java

接下来的MainActivity将会使用CameraManager来打开CameraDevice，并通过CameraDevice创建CameraCaptureSession，然后即可通过CameraCaptureSession进行预览或拍照了。该Activity的代码如下。

程序清单：
codes\11\11.3\CameraV2Test\app\src\main\java\org\crazyit\media\MainActivity.java

上面程序中的①号粗体字代码用于打开系统摄像头，openCamera（）方法的第一个参数代表请求打开的摄像头ID，此处传入的摄像头ID为"0"，这代表打开设备后置摄像头；如果需要打开设备指定摄像头（比如前置摄像头），可以在调用openCamera（）方法时传入相应的摄像头ID。

提示：

CameraManager提供了getCameraIdList（）方法来获取设备的摄像头列表，还提供了getCameraCharacteristics（String cameraId）方法来获取指定摄像头的特性。例如如下代码：

上面程序中的①号粗体字代码打开后置摄像头时传入了一个stateCallback参数，该参数代表的对象可检测摄像头的状态改变，当摄像头的状态发生改变时，程序将会自动回调该对象的相应方法。该程序的关键是重写了stateCallback的onOpened（CameraDevice cameraDevice）方法—当摄像头被打开时将会自动激发该方法，通过该方法的参数即可让程序获取被打开的摄像头设备。除此之外，程序在onOpened（）方法的②号粗体字代码处调用createCameraPreviewSession（）方法创建了CameraCaptureSession，并开始预览取景。

createCameraPreviewSession（）方法中的③号粗体字代码调用了CameraDevice的createCaptureSession（）方法来创建CameraCaptureSession，调用该方法时也传入了一个CameraCaptureSession.StateCallback参数，这样即可保证当CameraCaptureSession被创建成功之后立即开始预览。

createCameraPreviewSession（）方法的第一行粗体字代码将texture组件添加为previewRequestBuilder的target，这意味着程序通过previewRequestBuilder获取的图像数据将会被显示在texture组件上。

程序重写了CameraCaptureSession.StateCallback的onConfigured（）方法—当CameraCaptureSession创建成功时将会自动回调该方法，该方法先通过previewRequestBuilder设置了预览参数，然后调用CameraCaptureSession对象的setRepeatingRequest（）方法开始预览。

当单击程序界面上的拍照按钮时，程序将会激发该Activity的captureStillPicture（）方法。该方法的实现逻辑同样很简单：程序先创建一个CaptureRequest.Builder对象，该方法中第一行粗体字代码将ImageReader添加成CaptureRequest.Builder的target—这意味着当程序拍照时，图像数据将会被传给此ImageReader。接下来程序通过CaptureRequest.Builder设置了拍照参数，然后通过⑤号粗体字代码调用CameraCaptureSession的capture（）方法拍照即可，调用该方法时也传入了CaptureCallback参数，这样可以保证拍照完成之后会重新开始预览。

注意：

该应用打开摄像头、创建CameraCaptureSession、预览、拍照时都没有传入Handler参数，这意味着程序直接在主线程中完成相应的Callback任务，这样可能导致程序响应变慢。对于实际的应用，我们建议传入Handler参数，这样即可让Handler使用新线程来执行Callback任务，这样才可提高应用的响应速度。

由于该程序需要使用手机的摄像头，因此还需要在AndroidManifest.xml文件中增加如下配置：

在Genymotion模拟器上运行该程序可能看到如图11.8所示的预览界面—这是因为Genymotion模拟器可以使用宿主电脑上的摄像头作为相机摄像头。

为了让模拟器能显示图11.8所示的预览界面，建议读者启用Genymotion模拟器的摄像头支持：单击Genymotion模拟器右边的摄像头图标，即可看到如图11.9所示的对话框。按该图上标出的提示即可打开Genymotion模拟器的摄像头支持。

运行该程序，按下右下角的"拍照"键，程序将会把拍得的照片保存下来，界面上也会显示该照片的存储目录。

图11.8 预览界面

图11.9 打开Genymotion模拟器的摄像头

11.3.2 录制视频短片

MediaRecorder除了可用于录制音频之外，还可用于录制视频。使用MediaRecorder录制视频与录制音频的步骤基本相同。只是录制视频时不仅需要采集声音，还需要采集图像。为了让MediaRecorder录制时采集图像，应该在调用setAudioSource（int audio_source）方法时再调用setVideoSource（int video_source）方法来设置图像来源。

除此之外，还需要在调用setOutputFormat（）方法设置输出文件格式之后执行如下步骤。

1 调用MediaRecorder对象的setVideoEncoder（）、setVideoEncodingBitRate（int bitRate）、setVideoFrameRate（）方法设置所录制的视频编码格式、编码位率、每秒多少帧等，这些参数可以控制所录制的视频品质、文件大小。一般来说，视频品质越好，视频文件越大。

2 调用MediaRecorder的setPreviewDisplay（Surface sv）方法设置使用哪个SurfaceView来显示视频预览。

剩下的代码则与录制音频的代码基本相同。

实例：录制生活短片

本实例示范了如何录制视频。该程序的界面中提供了两个按钮用于控制开始、结束录制；除此之外，程序界面中还提供了一个SurfaceView来显示视频预览。该程序的界面布局文件如下。

程序清单：
codes\11\11.3\RecordVideo\app\src\main\res\layout\main.xml

提供了上面所示的界面布局文件之后，接下来就可以在程序中使用MediaRecorder来录制视频了。录制视频与录制音频的步骤基本相似，只是需要额外设置视频的图像来源、视频格式等。除此之外，还需要设置使用SurfaceView显示视频预览。录制视频的程序代码如下。

程序清单：
codes\11\11.3\RecordVideo\app\src\main\java\org\crazyit\media\MainActivity.java

上面程序中的粗体字代码设置了视频所采集的图像来源，以及视频的压缩格式、视频分辨率等属性，程序的①号粗体字代码则用于设置使用指定SurfaceView显示指定视频预览。

运行该程序需要使用麦克风录制声音，需要使用摄像头采集图像，这些都需要授予相应的权限；不仅如此，由于录制视频时视频文件增大

得较快，可能需要使用外部存储器，因此需要对应用程序授予相应的权限，也就是需要在AndroidManifest.xml文件中增加如下授权配置：

当在Genymotion模拟器上运行该程序时，由于Genymotion模拟器可以直接使用宿主电脑上的摄像头作为相机摄像头，因此在该模拟器上运行该程序可以看到如图11.10所示的界面。

11.4 Android 5.0新增的屏幕捕捉

在Android 5.0以前，如果开发者要对Android屏幕进行实时截图，往往被折腾得焦头烂额。Android 5.0的出现改变了这种现状，Android 5.0新增了MediaProjectionManager管理器，该管理器可以非常方便地实现屏幕捕捉功能。

使用MediaProjectionManager实现屏幕捕捉的步骤如下。

1 以MEDIA_PROJECTION_SERVICE为参数，调用Context.getSystemService（）方法即可获取MediaProjectionManager实例。

图11.10 录制视频短片

2 调用MediaProjectionManager对象的createScreenCaptureIntent（）方法创建一个屏幕捕捉的Intent。

3 调用startActivityForResult（）方法启动第2步得到的Intent，这样即可启动屏幕捕捉的Intent。

4 重写onActivityResult（）方法，在该方法中通过MediaProjectionManager对象来获取MediaProjection对象，在该对象中即可获取被捕获的屏幕。

下面示例示范了Android 5.0的屏幕捕捉功能。该应用界面上只包含一个ToggleButton和一个SurfaceView，其中ToggleButton用于打开屏幕捕捉功能，而SurfaceView则用于显示屏幕捕捉的图像。由于程序界面非常简单，故此处不再给出界面设计文件。

下面是该应用的Activity代码：

程序清单：
codes\11\11.4\CaptureScreen\app\src\main\java\org\crazyit\media\MainActivity.java

上面程序中的①号粗体字代码获取了屏幕捕捉的核心API：MediaProjectionManager，接下来程序即可通过该API来执行屏幕捕捉了。程序中②号粗体字代码调用MediaProjectionManager的createScreenCaptureIntent（）方法创建了一个屏幕捕捉的Intent，接下来③号粗体字代码启动该Intent即可开始屏幕捕捉。

当程序通过启动Activity开始屏幕捕捉之后，接下来程序重写了Activity的onActivityResult（）方法，并在该方法中通过④号粗体字代码获取了MediaProjection对象，程序调用MediaProjection的方法就可以将捕捉的图像显示到指定Surface中—如上面程序中最后一行代码所示。

图11.11 屏幕捕捉

编译、运行该程序，并打开程序界面上的ToggleButton，即可看到如图11.11所示的效果。

11.5 本章小结

音频和视频都是非常重要的多媒体形式，Android系统为音频、视频等多媒体的播放、录制提供了强大的支持。学习本章需要重点掌握如何使用MediaPlayer、SoundPool播放音频，如何使用VideoView、MediaPlayer播放视频。除此之外，还需要掌握通过MediaRecorder录制音频的方法，以及通过Android 5.0新增的Camera v2控制摄像头拍照、录制视频的方法。

第12章 OpenGL与3D开发

本章要点

3D编程的概念和基本理论

OpenGL与OpenGL ES

Android的OpenGL ES支持

在Android环境中使用OpenGL ES

利用OpenGL ES绘制2D图形

旋转图形

从2D到3D的转换

利用OpenGL ES绘制3D图形

通过OpenGL ES对3D图形应用贴图

前面介绍过Android系统的图形、图像处理。通过Android提供的图形、图形处理API，开发者可以非常方便地处理二维图形的处理、开发2D游戏等。但现在这个时代的用户显然并不满足于2D操作界面和2D游戏，3D技术已经被广泛应用于PC游戏上，3D技术下一步将要占领的肯定是手机平台。而Android系统已经为3D技术准备好了，Android系统完全内置了OpenGL ES（OpenGL for Embedded System）支持，也就是说，开发者可以在Android平台上使用OpenGL ES API来开发3D应用。

OpenGL本身是高效、简洁的开放图形库接口，它定义了一个跨编程语言、跨平台的编程接口规范，主要用于三维图形编程。但在手机等手

持终端上运行OpenGL有些不太合适，所以Android系统内置的是OpenGL ES支持。本章将会向读者介绍3D开发的基本知识，并介绍OpenGL ES编程的入门知识。但由于OpenGL ES的内容本身很多，而本书并不是一本专门介绍OpenGL ES编程的专著，所以并未过多地深入OpenGL ES编程，即使读者没有任何3D编程的知识，阅读本章也不会有任何障碍。

12.1 3D图形与3D开发的基本知识

现在的互联网上，虽然还有一些2D游戏存在，但3D游戏已经逐渐成为主流。毕竟3D游戏具有更逼真的界面，能带给玩家更好的用户体验。就目前的技术来看，开发3D界面的各种技术已经非常成熟，为开发3D界面、3D游戏提供了基础。

初学者可能会把3D开发想象得十分复杂。当然，3D游戏开发肯定要比2D游戏开发更加复杂，毕竟开发3D界面需要的数据更多。

在介绍3D图形开发之前，先从2D图形开发入手。

先来看如何定义一个2D的三角形。对于2D的三角形来说，它只需要三个点，而且这三个点位于同一个平面上，因此程序只要为每个点指定X、Y两个坐标值即可。

接下来看如何定义一个3D的三棱锥。一个三棱锥需要4个点，而且这4个点都不是位于同一个平面上，因此程序需要为每个点指定X、Y、Z三个坐标值。

图12.1 2D到3D的转换

从图12.1可以看出，3D图形需要处理的数据比2D图形要多得多。

从图12.1看到的只是最简单的2D图形—三角形和最简单的3D图形—三棱锥，但在实际应用中，应用程序需要呈现出来的2D图形可能由很多

曲线组成；类似地，应用程序需要呈现的3D图形也可能是由很多曲面组成的。

前面介绍2D绘图时，介绍了开发随手画图的程序，当程序希望在2D界面上绘制一条任意的曲线时，如果把这条曲线放大了来看（放得足够大），用户将会发现这条曲线其实是由许多足够短的直线连接起来的。

对于一个3D图形来说，即使用户眼中看到的是一个“圆滑曲面”的3D图形，实际上它也依然是由多个足够小的平面组成的，图12.2所示的就是从3d max里截取出来的一个3D图形。

正如图12.2所示，左边是用户希望看到的3D图形，但这个3D图形并不是程序员希望控制的。实际上程序员要控制的是右边的“网格图”，为了实现这个3D图形，程序员需要定义两方面的数据。

➢ 3D图形的每个顶点（Vertex）的位置，每个顶点的位置都需要X、Y、Z三个坐标值。

➢ 3D图形每个面由哪些顶点组成。

当程序给出了上面两方面的数据之后，接下来就可通知3D绘制接口来绘制这个3D图形了—就像绘制2D图形一样，程序员需要做的就是给出2D图形中各顶点的坐标。

为了定义3D图形每个顶点的位置，程序员需要给出X、Y、Z三个坐标值。为此，我们先要简单介绍一下Android的3D支持的三维坐标系统。

Android的3D坐标系统与2D坐标系统完全不同，图12.3显示了Android的三维坐标系统。

图12.2 3D图形的组成

图12.3 三维坐标系统

从图12.3可以看出，在Android的三维坐标系统中，坐标原点位于中央，X轴从左向右延伸，原点左边的值为负数，右边为正数；Y轴从下向上延伸，原点下边的值为负数，上边为正数；Z轴从屏幕里面向外面延伸，屏幕里面为负数，外面为正数。

了解了这个三维坐标系统，我们就可以根据该三维坐标系统来定义3D图形每个顶点的位置了。

12.2 OpenGL和OpenGL ES简介

OpenGL的全称是Open Graphics Library，即开放的图形库接口，它定义了一个跨编程语言、跨平台的编程接口规范，它主要用于三维图形（实际上二维图形也可以）编程。OpenGL的前身是SGI公司为其图形工作站开发的IRIS GL。IRIS GL是一个工业标准的3D图形软件接口，虽然功能强大，但是移植性不好，于是SGI公司便在IRIS GL的基础上开发了OpenGL。

OpenGL体系简单，而且具有跨平台的特性，它不像Direct3D（Microsoft开发的3D图形库接口，OpenGL的最有力的竞争对手）只能在Windows系统上运行，因此OpenGL具有很广泛的适应性一它不仅适用于大型图形工作站，也适用于PC。

在图形工作站、PC上，OpenGL都可以工作良好，但三维图形计算必须需要处理大量数据，因此在一些如手机之类的小型设备上，如果希望使用OpenGL就比较困难了。为此，Khronos集团为OpenGL提供了一个子集：OpenGL ES（OpenGL for Embedded System）。

Khronos是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准，Khronos协会针对手机、PDA和游戏主机等嵌入式设备设置了OpenGL ES。

OpenGL ES是免费的、跨平台的、功能完善的2D/3D图形库接口API，它针对多种嵌入式系统（包括控制台、移动电话、手持设备、家电设备和汽车）专门设计，它是一个精心提取出来的OpenGL的子集。

OpenGL ES剔除了OpenGl中glBegin/glEnd、四边形（GL_QUADS）、多边形（GL_POLYGONS）等许多非绝对必要的特性。经过多年发展，目前的OpenGL ES主要有两个版本：OpenGL ES 1.x 针对固定管线硬件；OpenGL ES 2.x 针对可编程管线硬件。

OpenGL ES 1.0 是以 OpenGL 1.3 规范为基础的，OpenGL ES 1.1 是以 OpenGL 1.5 规范为基础的，它们分别支持common 和 common lite两种profile。common lite profile只支持定点实数，而common profile既支持定点数又支持浮点数，common profile发布于2005年8月，引入了对可编程管线的支持。

目前Android 5.0已经支持OpenGL ES 3.1，而且Android扩展包为开发者提供了高性能的2D和3D图形API，包括计算着色器、模板纹理、加速视觉效果、高级纹理渲染、Tessellation着色器、几何着色器、ASTC纹理压缩、样本缺失值插补和着色等强大功能，能够适用于不同品牌的图形处理器。

Android专门为OpenGL支持提供了android.opengl包，在该包下提供了GLSurfaceView、GLU、GLUtils等工具类，通过这些工具类在Android应用中使用OpenGL ES更加方便。

12.3 绘制2D图形

掌握了上面关于3D图形开发的基本知识之后，下面可以利用Android的OpenGL ES支持来进行3D开发了。在真正3D图形之前，还需要先学习2D开发。

12.3.1 在Android应用中使用OpenGL ES

Android为OpenGL ES支持提供了GLSurfaceView组件，这个组件用于显示3D图形。GLSurfaceView本身并不提供绘制3D图形的功能，而是由GLSurfaceView.Renderer来完成SurfaceView中3D图形的绘制。

归纳起来，在Android中使用OpenGL ES需要三个步骤。

1 创建GLSurfaceView组件，使用Activity来显示GLSurfaceView组件。

2 为GLSurfaceView组件创建GLSurfaceView.Renderer实例，实现GLSurfaceView.Renderer类时需要实现该接口里的三个方法。

➢ **abstract void onDrawFrame（GL10 gl）：** Renderer对象调用该方法绘制GLSurfaceView的当前帧。

➢ **abstract void onSurfaceChanged（GL10 gl，int width，int height）：** 当GLSurfaceView的大小改变时回调该方法。

➢ **abstract void onSurfaceCreated（GL10 gl，EGLConfig config）：** 当GLSurfaceView被创建时回调该方法。

3 调用GLSurfaceView组件的setRenderer（）方法指定Renderer对象，该Renderer对象将会完成GLSurfaceView里3D图形的绘制。

从上面的介绍不难看出，实际上绘制3D图形的难点不是如何使用GLSurface组件，而是如何实现Renderer类。实现Renderer类时需要实现三个方法，这三个方法都有一个GL10形参，它就代表了OpenGL ES的“绘制画笔”，读者可以把它想象成Swing 2D绘图中的Graphics，也可以想象成Android 2D绘图中的Canvas组件一当我们希望Renderer绘制3D图形时，实际上是调用GL10的方法来进行绘制的。

当SurfaceView被创建时，系统会回调Renderer对象的onSurfaceCreated（）方法，该方法可以对OpenGL ES执行一些无须任何改变的初始化。例如如下初始化代码：

GL10就是OpenGL ES的绘图接口，虽然这里看到的是一个GL10，但实际上它也是GLES31的实例，读者可通过gl instanceof GL11判断它是否为GL11接口的实例。

上面的方法中用到了GL10的一些初始化方法，关于这些方法说明如下。

➢ **glDisable（int cap）**：用于禁用OpenGL ES某个方面的特性。该方法中第一行代码用于关闭抗抖动，这样可以提高性能。

➢ **glHint （int target，int mode）**：用于对OpenGL ES某方面进行修正。

➢ **glclearColor（float red，float green，float blue，float alpha）**：用于设置OpenGL E S"清屏"所用的颜色，4个参数分别设置红、绿、蓝、透明度值—0为最小值，1为最大值。例如设置gl.glClearColor（0，0，0，0），就是用黑色"清屏"。

➢ **glShadeModel（int mode）**：用于设置OpenGL ES的阴影模式。此处设为阴影平滑模式。

➢ **glEnable （int cap）**：该方法与glDisable（int cap）方法相对，用于启用OpenGL ES某方面的特性，此处用于启动OpenGL E S的深度测试。所谓深度测试，就是让OpenGL E S负责跟踪每个物体在Z轴上的深度，这样就可避免后面的物体遮挡前面的物体。

当SurfaceView组件的大小发生改变时，系统会回调Renderer对象的onSurfaceChanged（）方法，因此该方法通常用于初始化3D场景。例如如下初始化代码：

上面的方法中用到了GL10的一些初始化方法，关于这些方法说明如下。

➢ **glViewport（int x，int y，int width，int height）：**用于设置3D视窗的位置与大小。其中前两个参数指定该视窗的位置；后两个参数指定该视窗的宽、高。

➢ **glMatrixMode（int mode）：**用于设置视图的矩阵模型。通常可接受 GL10.GL_PROJECTION、GL10.GL_MODELVIEW两个常量值。

➢ 当调用gl.glMatrixMode（GL10.GL_PROJECTION）；代码后，指定将屏幕设为透视图（要想看到逼真的三维物体，这是必要的），这意味着越远的东西看起来越小；当调用gl.glMatrixMode（GL10.GL_MODELVIEW）；代码后，即将当前矩阵模式设为模型视图矩阵，这意味着任何新的变换都会影响该矩阵中的所有物体。

➢ **glLoadIdentity（）：**相当于reset（）方法，用于初始化单位矩阵。

➢ **glFrustumf （float left，float right，float bottom，float top，float zNear，float zFar）：**用于设置透视投影的空间大小。前两个参数用于设置X轴上的最小坐标值、最大坐标值；中间两个参数用于设置Y轴上的最小坐标值、最大坐标值；后两个参数用于设置Z轴上所能绘制的场景深度的最小值、最大值。

例如我们调用如下代码：

这意味着如果有一个二维矩形，它的4个顶点的坐标分别为（-0.8，1）、（0.8，1）、（0.8，-1）（-0.8，-1），这个矩形将会占满整个视窗。

提示：

前面已经指出，三维坐标系统与二维坐标系统并不相同，二维坐标系统上的坐标值通常就直接使用系统的像素数量；但三维坐标系统的坐标值则取决于glFrustumf（）方法的设置，当我们调用gl.glFrustumf

（-0.8，0.8，-1，1，1，10）；方法时，意味着该三维坐标系统的X轴最左边的坐标值为-0.8，最右边的坐标值为0.8；Y轴最上面的坐标值为1.0，最下面的坐标值为-1.0。

GLSurfaceView上的所有3D图形都是由Renderer的onDrawFrame（GL10 gl）方法绘制出来的，重写该方法时就要把所有3D图形都绘制出来，该方法通常以如下形式开始：

接下来在onDrawFrame（）方法中就可以调用GL10的方法开始绘制了。下面会介绍GL10所提供的常见的绘制方法。

12.3.2 绘制平面上的多边形

前面已经说过，计算机里的3D图形其实是由许多个平面组合而成的。所谓"绘制3D图形"，其实就是通过多个平面图形形成的。下面先从绘制平面图形开始。

调用GL10图形绘制2D图形的步骤如下。

1 调用GL10的glEnableClientState（GL10.GL_VERTEX_ARRAY）方法启用顶点坐标数组。

2 调用GL10的glEnableClientState（GL10.GL_COLOR_ARRAY）方法启用顶点颜色数组。

3 调用GL10的glVertexPointer（int size，int type，int stride，Buffer pointer）方法设置顶点的位置数据。这个方法中pointer参数用于指定顶点坐标值，但这里并未使用三维数组来指定每个顶点X、Y、Z坐标的值，pointer依然是一个一维数组，其格式为（x1，y1，z1，x2，y2，z2，x3，y3，z3，…，xN，yN，zN）；也就是该数组里将会包含3N个数值，每3个值指定一个顶点的X、Y、Z坐标值。第一个参数size指定多少个元素指定一个顶点位置，该size参数通常总是3；type参数指定顶点坐标值的类型，如果顶点坐标值为float类型，则指定为

GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为GL10.GL_FIXED。

4 调用GL10的glColorPointer （int size，int type，int stride，Buffer pointer）方法设置顶点的颜色数据。这个方法中pointer参数用于指定顶点的颜色值，pointer依然是一个一维数组，其格式为（r1，g1，b1，a1，r2，g2，b2，a2，r3，g3，b3，a3，…，rN，gN，bN，aN）；也就是该数组里将会包含4N个数值，每4个值指定一个顶点颜色的红、绿、蓝、透明度的值。第一个参数size指定多少个元素指定一个顶点位置，该size参数通常总是4；type参数指定顶点坐标值的类型，如果顶点坐标值为float类型，则指定为GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为GL10.GL_FIXED。

5 调用GL10的glDrawArrays （int mode，int first，int count）方法绘制平面。该方法的第一个参数指定绘制图形类型，第二个参数指定从哪个顶点开始绘制，第三个参数指定总共绘制的顶点数量。

6 绘制完成后，调用GL10的glFinish（）方法结束绘制，并调用glDisableClientState（int）方法来停用顶点坐标数据、顶点颜色数据。

掌握上面的步骤之后，接下来通过示例程序来绘制几个简单的图形。

先为该程序提供一个Renderer实现类，该实现类的代码如下。

程序清单：
codes\12\12.3\Polygon\app\src\main\java\org\crazyit\opengl\MyRenderer.java

在本书的第1版中，为了将int[]数组转换为Open GL ES所需的IntBuffer，只要调用IntBuffer的wrap（）方法进行包装即可，但现在Android平台的要求更加严格，要求该Buffer必须是native Buffer（因此使用ByteBuffer的allocateDirect（）方法进行创建），并且该Buffer必须是排序的（因此调用了ByteBuffer的order（）方法进行排序）。

上面程序中的粗体字代码就是使用GL10绘制图形的关键代码：加载顶点位置数据；加载顶点颜色数据；调用GL10的glDrawArrays（）方法绘制即可。由于加载顶点位置数据、顶点颜色数据时都需要Buffer对象，因此程序在MyRenderer类的构造器中把这些顶点位置数据、顶点颜色数据都包装成了相应的FloatBuffer、IntBuffer。

上面的程序中①号代码调用了GL10的glTranslatef（-0.32f，0.35f，-1f）方法，这个glTranslatef（）方法的作用就类似于Android 2D绘图中Matrix的setTranslate（float dx，float dy）方法，它们都用于移动绘图中心，区别只是2D绘图中Matrix的setTranslate（）方法只要指定在X、Y轴上的移动距离，而GL10的glTranslatef（）方法需要指定在X、Y、Z轴上的移动距离。在绘制图形之前，先调用GL10的glTranslatef（float，float，float）方法即可保证把图形绘制在指定的中心点。

上面的程序中②号代码还调用了glColor4f（1.0f，0.2f，0.2f，0.0f）方法设置使用纯色填充。设置使用纯色填充时需要禁用顶点颜色数组，如②号代码后的一行代码。

在Activity中定义一个GLSurfaceView，并使用上面的Renderer进行绘制，程序如下。

程序清单：
codes\12\12.3\polygon\app\src\main\java\org\crazyit\opengl\MainActivity.java

运行上面的程序，可以看到如图12.4所示的输出。

很多读者会对图12.4所示绘制的图形感到奇怪：第二个图形（右上角的正方形）和第三个图形（左下角的图形）都有完全相同的4个坐标点，只是定义4个坐标点的顺序略有不同，为何绘制的图形存在这么大区别呢?

再来看GL10提供的glDrawArrays（int mode，int first，int count）方法，该方法的第一个参数指定绘制模式，可指定为如下两个值。

➢ **GL10.GL_TRIANGLES**：绘制三角形。

➢ **GL10.GL_TRIANGLE_STRIP**：用多个三角形来绘制多边形。

图12.4 使用OpenGL ES绘制2D图形

提示：

前面介绍OpenGL与OpenGL ES的区别时已经指出，OpenGL ES剔除了OpenGl中的四边形（GL_QUADS）、多边形（GL_POLYGONS）支持，也就是OpenGL ES只能绘制三角形组成3D图形。

当调用glDrawArrays（）方法时，如果将mode参数指定为GL10.GL_TRIANGLES，则绘制简单的三角形；如果将mode参数指定为GL10.GL_TRIANGLE_STRIP，那么系统将会沿着给出的顶点数据来绘制三角形。

对于上面程序中的第二个图形，程序给出4个顶点的顺序，如图12.5所示。

当指定了glDrawArrays（int mode，int first，int count）方法的第一个参数为GL10.GL_TRIANGLE_STRIP时，系统总会从first顶点开始，每3个顶点绘制一个三角形。例如调用代码：gl.glDrawArrays（GL10.GL_TRIANGLE_STRIP，0，4）；，这意味着将会绘制两个三角形，分别由0、1、2三个顶点组成的三角形和由1、2、3三个顶点组

成的三角形。因此，对于第二个图形而言，gl.glDrawArrays（GL10.GL_TRIANGLE_STRIP，0，4）；所绘制的图形如图12.6所示。

图12.5 第二个图形的顶点数据

图12.6 OpenGL ES使用三角形绘制正方形

对于第三个图形而言，虽然它的4个顶点的位置与第二个图形顶点的位置完全相同，但由于4个顶点的顺序有所不同，所以它也是绘制两个三角形（由0、1、2顶点组成第一个三角形和由1、2、3顶点组成第二个三角形），如图12.7所示（浅色边界为第二个三角形的边界）。如图12.7所示的图形就与上面程序所绘制的第三个图形相同了。

图12.7 O penGL ES使用三角形绘制正方形

讲到这里，可能有读者对3D开发感到害怕了一现在还只是绘制了几个简单的图形，程序员不仅要给出图形每个顶点的位置信息，还要按指定顺序来排列这些顶点，这也太复杂了吧！不要担心，现在所使用的glDrawArrays（）方法是有点复杂，实际上3D图形中每个顶点的坐标值不需要由程序员计算、给出；顶点的排列顺序也无须由程序员排列。

提示：

可以想象，如果3D场景中每个物体的所有顶点都由程序员来计算、定义，那几乎是不可想象的一如果让我们定义一个怪兽的头部，需要多少个顶点？每个顶点的位置到底应该在哪里？把一个程序员算到死都有可能！这时候通常会借助于3d max、Maya等三维建模工具，当我

们把一个怪兽头部的模型建立出来后，这个物体的所有顶点坐标值及顶点的排列顺序都可以导出来。OpenGL可以直接导入这些三维建模工具所建立的模型。

12.3.3 旋转

GL10提供了一个glRotatef（float angle，float x，float y，float z）方法，用于控制旋转。该方法中angle控制旋转角度；而x、y、z参数则共同决定了旋转轴的方向。

本质上，glRotatef（float angle，float x，float y，float z）方法的作用与glTranslatef（float x，float y，float z）方法相似，只是glTranslatef（float x，float y，float z）方法控制图形中心移动；而glRotatef（float angle，float x，float y，float z）方法控制图形沿着指定旋转轴转动指定角度。

因此，只要在调用glTranslatef（）方法控制图形移动之后，再调用glRotatef（）控制图形旋转即可，如果希望看到指定图形不断旋转，则只要在onDrawFrame（GL10 gl）方法中不断增加旋转角度即可。

下面是该程序所用的Renderer实现类。

程序清单：
codes\12\12.3\RotatePolygon\app\src\main\java\org\crazyit\opengl\MyRenderer.java

上面程序中的粗体字代码定义了一个rotate变量，该变量用于控制程序中两个图形的旋转角度，其中第二个图形沿着Z轴旋转，第三个图形则沿着Y轴旋转。运行该程序，将可以看到第二个图形、第三个图形不断旋转的效果，如图12.8所示。

掌握了使用OpenGL ES通过三角形绘制平面图形之后，接下来就可以调用OpenGL ES来绘制3D图形了—绘制3D图形需要定义更多的顶点数据，而且需要更好地控制哪些顶点需要组成三角形。

12.4 绘制3D图形

图12.8 控制图形变换

如果定义的顶点不在同一个平面上，并且使用三角形把合适的顶点连接起来，就可以绘制出3D图形了。

12.4.1 构建3D图形

正如前面所介绍的，使用OpenGL ES绘制3D图形的步骤与绘制2D图形的步骤大致相同，只是绘制3D图形需要定义更多的顶点数据，而且3D图形需要绘制更多的三角形。

如果还使用前面介绍的glDrawArrays（int mode，int first，int count）进行绘制，我们将会面临巨大的考验：到底要按怎样的顺序来组织三维空间上的多个顶点，才能绘制出我们需要的三角形？为了解决这个问题，GL10提供了如下方法。

➢ **glDrawElements（int mode，int count，int type，Buffer indices）**：根据indices指定的索引点来绘制三角形。该方法的第一个参数指定绘制的图形类型，可设为GL10.GL_TRIANGLES或GL10.GL_TRIANGLE_STRIP；第二个参数指定一共包含多少个顶点。indices 参数最重要，它包装了一个长度为 3N 的数组，比如让该参数包装{0，2，3，1，4，5}数组，这意味着告诉OpenGL ES要绘制两个三角形，第一个三角形的三个顶点为0、2、3顶点，第二个三角形的三个顶点为1、4、5顶点。

由此可见，如果希望在程序中使用glDrawElements（int mode，int count，int type，Buffer indices）方法来绘制3D图形，不仅需要指定每个3D图形的顶点位置信息，也需要指定3D图形的每个面由哪三个顶点组成。

下面的程序使用glDrawElements（int mode，int count，int type，Buffer indices）方法绘制了两个简单的3D图形：三棱锥和立方体。该程序的Activity依然没有变化，故下面只给出Renderer实现类的代码。

程序清单：
codes\12\12.4\Simple3D\app\src\main\java\org\crazyit\opengl\MyRenderer.java

从上面的程序不难看出，绘制3D图形的步骤与绘制2D图形的步骤基本相似，区别只是绘制3D图形不仅需要定义各顶点位置的坐标，还需要定义3D图形的各个三角面由哪些顶点组成，例如上面的程序中粗体字代码所示—为了定义一个立方体，除了给出这个立方体的8个顶点位置坐标之外，还需要定义组成该立方体的12个三角形。图12.9显示了该立方体上8个顶点的位置。

接下来程序需要把图12.9所示正方形的6个面（由12个三角形组成）定义出来，依次指定每个三角形包括哪三个顶点，也就是程序中粗体字代码所定义的数组。

运行上面的程序，将可以看到如图12.10所示的3D图形。

图12.9 立方体的8个顶点的位置示意图

图12.10 使用OpenGL绘制3D图形

从上面的程序可以看出，为了在程序中绘制一个简单的3D立方体，程序需要定义12个三角面，这种规则的立方体还可以由程序员计算并指定，但如果遇到更复杂的3D物体，就必须借助于三维建模软件了。

12.4.2 应用纹理贴图

为了让3D图形更加逼真，我们需要为这些3D图形应用纹理贴图。比如开发一个怪兽头部，如果只是为这个头部绘制五颜六色的"皮肤"，那也太假了吧。如果考虑为这个怪兽应用"鳄鱼皮肤"的纹理贴图，或者应用"蟒蛇皮肤"的纹理贴图，这个怪兽头部就逼真多了，甚至会带给人一种恐怖的感觉。

为了在OpenGL ES中启用纹理贴图功能，可以在Renderer实现类的onSurfaceCreated（GL10 gl，EGLConfig config）方法中启用纹理贴图。例如如下代码：

然后需要准备一张图片来作为纹理贴图了，建议该图片的长、宽是2的N次方，比如长、宽为256、512等。把这张准备贴图的位图放在Android项目的res/drawable/目录下，以方便应用程序加载该图片资源。

接下来程序开始加载该图片并生成对应的纹理贴图。例如如下方法：

上面的程序中用到了GL10的如下方法。

➢ **glGenTextures（int n，int[] textures，int offset）**：该方法指定一次性生成n个纹理。该方法所生成的纹理代号放入其中的textures数组中，offset指定从第几个数组元素开始存放纹理代号。

➢ **glBindTexture（int target，int texture）**：该方法用于将texture纹理绑定到target目标上。

➢ **glTexParameterf（int target，int pname，float param）**：该方法用于为target纹理目标设置属性，其中第二个参数是属性名，第三个参数是属性值。

上面的程序中有4行代码调用了glTexParameterf（int target，int pname，float param）方法，程序设置了当纹理被放大时使用GL10.GL_LINEAR滤波方式；当纹理被缩小时使用GL10.GL_NEAREST滤波方式。一般来说，使用GL10.GL_LINEAR滤波方式有较好的效果，但系统开销略微大了一些，具体采用哪种滤波方式则取决于目标机器本身的性能。

上面程序的最后一行调用了GLUtils工具类的方法来加载指定位图，并根据位图来生成纹理，通过上面的代码即可得到一个用于贴图的纹理了。

在3D绘制中进行纹理贴图也很简单，与设置顶点颜色的步骤相似，只要三步。

1 设置启用贴图坐标数组。

2 设置贴图坐标的数组信息。

3 调用GL10的glBindTexture（int target，int texture）方法执行贴图。

下面的程序示范了如何为一个立方体进行贴图，而且这个程序还提供了手势检测器，允许用户通过手势来改变该立方体的角度。

程序清单：
codes\12\12.4\Texture3D\app\src\main\java\org\crazyit\opengl\MainActivity.java

上面的程序中①、②、③号代码就完成了纹理贴图的三个步骤。

程序中粗体字代码用于计算用户手势在X方向、Y方向上的速度，并根据X方向、Y方向上的速度来改变立方体的旋转角度，这样就可以让该立方体随用户的手势而转动了。

可能有读者会发现上面这个立方体的顶点坐标看上去很“可怕”，怎么一个立方体需要这么多顶点呢？实际上是因为笔者实在不想在纸上先画出这个立方体，然后数立方体的每个三角面由哪些顶点组成，再计算每个三角面的贴图坐标—这太耗时间了。笔者直接使用3d max建了一个立方体模型，并从中导出了模型的每个顶点的位置信息、每个三角面由哪些顶点组成，以及贴图的坐标信息。

3d max就不会像我们之前那样“复用”立方体的顶点—它直接计算该立方体需要12个三角面，每个三角面需要3个顶点，这样一共是36个顶点—其实有大量顶点的位置是相同，但3d max不管这些。它认为这个立方体需要36个顶点，每个顶点的位置需要X、Y、Z三个坐标值，因此导出这个立方体的顶点坐标信息的数组就是一个长度为108的数组。

提示：

由于本书只是介绍简单的OpenGL ES编程，因此涉及3d max的内容就此打住，如果读者还有更多兴趣，则可以参考3d max的相关资料。

运行上面的程序，用户将可以看到一个具有纹理贴图的三维立方体，用户可以通过拖动手势来旋转该立方体，程序效果如图12.11所示。

需要指出的是，OpenGL ES虽然只是OpenGL的一个子集，但其实它所包含的功能也很多，而本书的重点是介绍Android所支持的3D开发，限于篇幅，不可能深入介绍OpenGL ES编程的全部内容，如果读者希望深入了解OpenGL ES编程的内容，可以自行参考相关书籍和资料。

图12.11 具有纹理贴图的立方体

12.5 本章小结

本章主要介绍了Android 3D编程的入门知识，简要介绍了2D绘图和3D绘图的联系与区别，也简要介绍了OpenGL与OpenGL ES。Android系统内置了对OpenGL ES的支持，开发者可以在Android应用中通过OpenGL ES来绘制3D图形。学习本章需要掌握3D绘图的基本理论、三维坐标系统等。除此之外，还需要重点掌握如何在Android环境中使用OpenGL ES，以及通过OpenGL ES绘制2D图形、3D图形的方法。

第13章 Android网络应用

本章要点

TCP协议的基础

使用ServerSocket建立服务器

使用Socket进行网络通信

在网络通信中加入多线程支持

使用URL读取网络资源

使用URLConnection提交请求

使用HttpURLConnection

Apache HttpClient的基础知识

使用HttpClient维护用户状态、发送请求、获取响应

使用WebView浏览网页

使用WebView加载、显示HTML代码

使用WebView中JavaScript脚本调用Android方法

Web Service的基本知识

Web Service平台简介

使用ksoap2-android项目来调用远程Web Service

手机本身是作为手持终端来使用的，因此它的计算能力、存储能力都是有限的。它的主要优势是携带方便，可以随时打开，而且手机通常总是处于联网状态，因此网络支持对于手机应用的重要性不言而喻。

Android完全支持JDK本身的TCP、UDP网络通信API，也可以使用ServerSocket、Socket来建立基于TCP/IP协议的网络通信还可以使用DatagramSocket、Datagrampacket、MulticastSocket来建立基于UDP协议的网络通信。如果读者有Java网络编程的经验，这些经验完全适用于Android应用的网络编程。Android也支持JDK提供的URL、URLConnection等网络通信API。

提示：

限于篇幅，本章并未涉及UDP协议编程的相关内容，如果读者需要掌握基于UDP协议，使用DatagramSocket、Datagrampacket、MulticastSocket进行网络编程的内容，可以参考疯狂Java体系的《疯狂Java讲义》。

不仅如此，Android还内置了HttpClient，这样可以非常方便地发送HTTP请求，并获取HTTP响应，通过内置HttpClient，Android简化了与网站之间的交互。令人遗憾的是，Android并未内置对Web Service的支持，为了弥补这种不足，本章将会介绍如何利用ksoap2-android项目在Android应用中调用远程Web Service。

13.1 基于TCP协议的网络通信

TCP/IP通信协议是一种可靠的网络协议，它在通信的两端各建立一个Socket，从而在通信的两端之间形成网络虚拟链路。一旦建立了虚拟的网络链路，两端的程序就可以通过虚拟链路进行通信了。Java对基于TCP协议的网络通信提供了良好的封装，Java使用Socket对象来代表两端的通信接口，并通过Socket产生IO流来进行网络通信。

13.1.1 TCP协议基础

IP协议是Internet上使用的一个关键协议，它的全称是Internet Protocol，即Internet协议，通常简称IP协议。通过使用IP协议，使Internet成为一个允许连接不同类型的计算机和不同操作系统的网络。

要使两台计算机彼此之间进行通信，两台计算机必须使用同一种"语言"，IP协议只保证计算机能发送和接收分组数据。IP协议负责将消息从一个主机传送到另一个主机，消息在传送的过程中被分割成一个个小包。

尽管计算机通过安装IP软件，保证了计算机之间可以发送和接收数据，但IP协议还不能解决数据分组在传输过程中可能出现的问题。因此，若要解决可能出现的问题，连接上Internet的计算机还需要安装TCP协议来提供可靠并且无差错的通信服务。

TCP协议被称为一种端对端协议。这是因为它为两台计算机之间的连接起了重要作用：当一台计算机需要与另一台远程计算机连接时，TCP协议会让它们建立一个连接一用于发送和接收数据的虚拟链路。

TCP协议负责收集这些信息包，并将其按适当的次序放好传送，在接收端收到后再将其正确地还原。TCP协议保证了数据包在传送中准确无误。TCP协议使用重发机制：当一个通信实体发送一个消息给另一个通信实体后，需要收到另一个通信实体的确认信息，如果没有收到另一个通信实体的确认信息，则会再次重发刚才发送的消息。

通过这种重发机制，TCP协议向应用程序提供可靠的通信连接，使它能够自动适应网上的各种变化。即使在Internet暂时出现堵塞的情况下，TCP也能够保证通信的可靠。

图13.1显示了TCP协议控制两个通信实体互相通信的示意图。

图13.1 TCP协议的通信示意图

综上所述，虽然IP和TCP这两个协议的功能不尽相同，也可以分开单独使用，但它们是在同一时期作为一个协议来设计的，并且在功能上也是互补的。只有两者结合，才能保证 Internet 在复杂的环境下正常运行。凡是要连接到Internet的计算机，都必须同时安装和使用这两个协议，因此在实际中常把这两个协议统称为TCP/IP协议。

13.1.2 使用ServerSocket创建TCP服务器端

从图13.1中看上去TCP通信的两个通信实体之间并没有服务器端、客户端之分，但那是两个通信实体已经建立虚拟链路之后的示意图。在两个通信实体没有建立虚拟链路之前，必须有一个通信实体先做出“主动姿态”，主动接收来自其他通信实体的连接请求。

Java中能接收其他通信实体连接请求的类是ServerSocket，ServerSocket对象用于监听来自客户端的Socket连接，如果没有连接，它将一直处于等待状态。ServerSocket包含一个监听来自客户端连接请求的方法。

➢ **Socket accept（）：** 如果接收到一个客户端Socket的连接请求，该方法将返回一个与连接客户端Socket对应的Socket（如图13.1所示，每个TCP连接有两个Socket）；否则该方法将一直处于等待状态，线程也被阻塞。

为了创建ServerSocket对象，ServerSocket类提供了如下几个构造器。

➢ **ServerSocket（int port）：** 用指定的端口port来创建一个ServerSocket。该端口应该有一个有效的端口整数值0～65535。

➢ **ServerSocket（int port，int backlog）：** 增加一个用来改变连接队列长度的参数backlog。

➢ **ServerSocket（int port，int backlog，InetAddress localAddr）：** 在机器存在多个 IP地址的

情况下，允许通过localAddr这个参数来指定将ServerSocket绑定到指定的IP地址。

当ServerSocket使用完毕后，应使用ServerSocket的close（）方法来关闭该ServerSocket。在通常情况下，服务器不应该只接收一个客户端请求，而应该不断地接收来自客户端的所有请求，所以Java程序通常会通过循环不断地调用ServerSocket的accept（）方法，如以下代码片段所示。

提示：

上面的程序中创建ServerSocket没有指定IP地址，则该ServerSocket将会绑定到本机默认的IP地址。程序中使用30000作为该ServerSocket的端口号，通常推荐使用1024以上的端口，主要是为了避免与其他应用程序的通用端口冲突。

由于手机无线上网的IP地址通常都是由移动运营公司动态分配的，一般不会有自己固定的IP地址，因此很少在手机上运行服务器端，服务器端通常运行在有固定IP地址的服务器上。本节所介绍的应用程序的服务器端也是运行在PC上的。

13.1.3 使用Socket进行通信

客户端通常可以使用Socket的构造器来连接到指定服务器，Socket通常可提供如下两个构造器。

➢ **Socket（InetAddress/String remoteAddress，int port）：**创建连接到指定远程主机、远程端口的Socket，该构造器没有指定本地地址、本地端口，默认使用本地主机的默认IP地址，默认使用系统动态分配的端口。

➢ **Socket（InetAddress/String remoteAddress，int port，InetAddress localAddr，intlocalPort）：**创建连接到指定远程主机、远程端口的Socket，并指定本地IP地址和本地端口，适用于本地主机有多个IP地址的情形。

上面两个构造器中指定远程主机时既可使用InetAddress来指定，也可直接使用String对象来指定，但程序通常使用String对象（如192.168.2.23）来指定远程IP地址。当本地主机只有一个IP地址时，使用第一个方法更为简单，如以下代码所示。

当程序执行上面的粗体字代码时，该代码将会连接到指定服务器，让服务器端的ServerSocket的accept（）方法向下执行，于是服务器端和客户端就产生一对互相连接的Socket。

提示：

上面的程序连接到"远程主机"的IP地址使用的是192.168.1.88，这个IP地址就是笔者运行服务器端程序的IP地址。

当客户端、服务器端产生了对应的Socket之后，此时就到了如图13.1所示的通信示意图，程序无须再区分服务器、客户端，而是通过各自的Socket进行通信。Socket提供了如下两个方法来获取输入流和输出流。

➢ **InputStream getInputStream（）：**返回该Socket对象对应的输入流，让程序通过该输入流从Socket中取出数据。

➢ **OutputStream getOutputStream（）：**返回该Socket对象对应的输出流，让程序通过该输出流向Socket中输出数据。

看到这两个方法返回的InputStream和OutputStream，读者应该可以明白Java在设计IO体系上的苦心了：不管底层的IO流是怎样的节点流，文件流也好，网络Socket产生的流也好，程序都可以将其包装成

处理流，从而提供更多方便的处理。下面以一个最简单的网络通信程序为例来介绍基于TCP协议的网络通信。

提示：

如果读者朋友阅读过疯狂Java体系的《疯狂Java讲义》，可能会发现本章的示例程序、文字内容与《疯狂Java讲义》介绍网络编程的一章有较多相同的内容。实际上也是这样的，因为Android网络通信还是依赖于JDK在java.net、java.io两个包下的API。

下面的服务器端程序需要在PC上运行，该程序非常简单，因此不需要建立Android项目，直接定义一个Java类，并运行该Java类即可。它仅仅建立ServerSocket监听，并使用Socket获取输出流输出。

程序清单：codes\13\13.1\SimpleServer\SimpleServer.java

上面程序中的①号粗体字代码建立了一个ServerSocket，该ServerSocket在30000端口监听，该ServerSocket将会一直监听，等待客户端程序的连接。程序接下来的4行粗体字代码用于打开Socket对应的输出流，并向输出流中写入一段字符串数据。

注意：

上面的程序并未把OutputStream流包装成PrintStream，然后使用PrintStream直接输出整个字符串，这是因为该服务器端程序运行于Windows主机上，当直接使用PrintStream输出字符串时默认使用系统平台的字符串（即GBK）进行编码；但该程序的客户端是Android应用，运行于Linux平台（Android是Linux内核的）上，因此当客户端读取网络数据时默认使用UTF-8字符集进行解码，这样势必引起乱码。为了保证客户端能正常解析到数据，此处手动控制字符串的编码，强行指定使用UTF-8字符集进行编码，这样就可以避免乱码问题了。

接下来的客户端程序也非常简单，它仅仅使用Socket建立与指定IP地址、指定端口的连接，并使用Socket获取输入流读取数据。该客户端程序是一个Android应用，因此还是需要先建立Android项目，该程序的界面中包含一个文本框，用于显示从服务器端读取的字符串数据。

程序清单：
codes\13\13.1\SimpleClient\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的①号粗体字代码是使用ServerSocket和Socket建立网络连接的代码，接下来的几行粗体字代码是通过Socket获取输入流、输出流进行通信的代码。通过程序不难看出，一旦使用ServerSocket、Socket建立网络连接之后，程序通过网络通信与普通IO就没有太大的区别了。

该程序与本书第1版中的程序有所区别：本书第1版为了简化编程、突出网络编程的主题，因此直接在UI线程中建立网络连接，通过网络读取服务器响应数据—但在实际项目中并不推荐这么做，因为建立网络连接、网络通信是不稳定的，它所需要的时间也不确定，因此直接在UI线程中建立网络连接、通过网络读取数据可能阻塞UI线程，导致Android应用失去响应。在最新的Android平台上，Android已经不允许在UI线程中建立网络连接，因此上面的示例启动了一条新线程来建立网络连接，并读取网络数据。

注意：

新版Android平台与Android 2.3.x在网络编程这一章最大的区别是，新版Android平台不允许直接在UI线程中建立网络连接、访问网络资源，因此本章后面的所有示例程序都会将建立网络连接、访问网络资源的操作放在新线程中完成。

该Android应用需要访问互联网，因此还需要为该应用赋予访问互联网的权限，也就是在AndroidManifest.xml文件中增加如下配置片段：

图13.2 与远程服务器交互

先运行上面程序中的SimpleServer类，将看到服务器一直处于等待状态，因为服务器使用了死循环来接收来自客户端的请求；再运行客户端AndroidClient类，将可看到程序输出："来自服务器的数据：您好，您收到了服务器的新年祝福！"，如图13.2所示，这表明客户端和服务器端通信成功。

上面的程序为了突出通过ServerSocket和Socket建立连接并通过底层IO流进行通信的主题，程序没有进行异常处理，也没有使用finally块来关闭资源。

在实际应用中，程序可能不想让执行网络连接、读取服务器数据的进程一直阻塞，而是希望当网络连接、读取操作超过合理时间之后，系统自动认为该操作失败，这个合理时间就是超时时长。Socket对象提供了一个setSoTimeout（int timeout）来设置超时时长，如下面的代码片段所示。

为Socket对象指定了超时时长之后，如果使用Socket进行读写操作完成之前已经超出了该时间限制，那么这个方法就会抛出SocketTimeoutException异常，程序可以对该异常进行捕获，并进行适当处理，如以下代码所示。

假设程序需要为Socket连接服务器时指定超时时长，即经过指定时间后，如果该Socket还未连接到远程服务器，则系统认为该Socket连接超时。但Socket的所有构造器里都没有提供指定超时时长的参数，所以程序应该先创建一个无连接的Socket，再调用Socket的connect（）

方法来连接远程服务器，而connect（）方法就可以接受一个超时时长参数，如以下代码所示。

13.1.4 加入多线程

前面服务器端和客户端只是进行了简单的通信操作：服务器接收到客户端连接之后，服务器向客户端输出一个字符串，而客户端也只是读取服务器的字符串后就退出了。在实际应用中的客户端则可能需要和服务器端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据；客户端也需要不断地读取服务器数据，并向服务器写入数据。

当使用传统BufferedReader的readLine（）方法读取数据时，在该方法成功返回之前，线程被阻塞，程序无法继续执行。考虑到这个原因，服务器应该为每个Socket单独启动一条线程，每条线程负责与一个客户端进行通信。

客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

下面考虑实现一个简单的C/S聊天室应用。服务器端应该包含多条线程，每个Socket对应一条线程，该线程负责读取Socket对应输入流的数据（从客户端发送过来的数据），并将读到的数据向每个Socket输出流发送一遍（将一个客户端发送的数据"广播"给其他客户端），因此需要在服务器端使用List来保存所有的Socket。

下面是服务器端的实现代码。程序为服务器提供了两个类：一个是创建ServerSocket监听的主类；另一个是负责处理每个Socket通信的线程类。

程序清单：codes\13\13.1\MultiThreadServer\MyServer.java

上面的程序是服务器端只负责接收客户端Socket的连接请求，每当客户端Socket连接到该ServerSocket之后，程序将对应Socket加入socketList集合中保存，并为该Socket启动一条线程，该线程负责处理该Socket所有的通信任务，如程序中4行粗体字代码所示。服务器端线程类的代码如下。

程序清单：
codes\13\13.1\MultiThreadServer\ServerThread.java

上面的服务器端线程类不断读取客户端数据，程序使用readFromClient（）方法来读取客户端数据，如果在读取数据过程中捕获到IOException异常，则表明该Socket对应的客户端Socket出现了问题（到底什么问题我们不管，反正不正常），程序就将该Socket从socketList中删除，如readFromClient（）方法中①号粗体字代码所示。

当服务器线程读到客户端数据之后，程序遍历socketList集合，并将该数据向socketList集合中的每个Socket发送一次—该服务器线程将把从Socket中读到的数据向socketList中的每个Socket转发一次，如run（）线程执行体中的粗体字代码所示。

提示：

上面程序中的②号粗体字代码将网络的字节输入流转换为字符输入流时，指定了转换所用的字符串：UTF-8，这也是由于客户端写过来的数据是采用UTF-8字符集进行编码的，所以此处的服务器端也要使用UTF-8字符集进行解码。当需要编写跨平台的网络通信程序时，使用UTF-8字符集进行编码、解码是一种较好的解决方案。

每个客户端应该包含两条线程：一条负责生成主界面，响应用户动作，并将用户输入的数据写入Socket对应的输出流中；另一条负责读

取Socket对应输入流中的数据（从服务器发送过来的数据），并负责将这些数据在程序界面上显示出来。

客户端程序同样是一个Android应用，因此需要创建一个Android项目，这个Android应用的界面中包含两个文本框：一个用于接收用户的输入；另一个用于显示聊天信息。界面中还有一个按钮，当用户单击该按钮时，程序向服务器发送聊天信息。该程序的界面布局代码如下。

程序清单：
codes\13\13.1\MultiThreadClient\app\src\main\res\layout\main.xml

客户端的Activity负责生成程序界面，并为程序的按钮单击事件绑定事件监听器，当用户单击按钮时向服务器发送信息。客户端的Activity代码如下。

程序清单：
codes\13\13.1\MultiThreadClient\app\src\main\java\org\crazyit\net\MainActivity.java

当用户单击该程序界面中的“发送”按钮后，程序将会把input输入框中的内容发送给clientThread的revHandler对象，clientThread负责将用户输入的内容发送给服务器。

为了避免UI线程被阻塞，该程序将建立网络连接、与网络服务器通信等工作都交给ClientThread线程完成，因此该程序在①号粗体字代码处启动ClientThread线程。

由于Android不允许子线程访问界面组件，因此上面的程序定义了一个Handler来处理来自子线程的消息，如程序中②号粗体字代码所示。

ClientThread子线程负责建立与远程服务器的连接，并负责与远程服务器通信，读到数据之后便通过Handler对象发送一条消息；当ClientThread子线程收到UI线程发送过来的消息（消息携带了用户输入的内容）后，还负责将用户输入的内容发送给远程服务器。该子线程的代码如下。

程序清单：
codes\13\13.1\MultiThreadClient\app\src\main\java\org\crazyit\net\ClientThread.java

上面线程的功能也非常简单，它只是不断地获取Socket输入流中的内容，当读到Socket输入流中的内容后，便通过Handler对象发送一条消息，消息负责携带读到的数据，如上面程序中第一段粗体字代码所示。除此之外，该子线程还负责读取UI线程发送的消息，接收到消息之后，该子线程负责将消息中携带的数据发送给远程服务器，如上面程序中第二段粗体字代码所示。

图13.3 支持多线程的TCP客户端

先运行上面程序中的MyServer类，该类运行后只是作为服务器，看不到任何输出。接着可以运行Android客户端—相当于启动聊天室客户端登录该服务器，接下来在任何一个Android客户端输入一些内容后单击“发送”按钮，将可以看到所有客户端（包括自己）都会收到刚刚输入的内容，如图13.3所示，这就粗略实现了一个C/S结构聊天室的功能。

借助于此处介绍的网络通信机制，我们可以在Android平台上开发大量功能强大的网络通信程序，比如《疯狂 Java 讲义》中所介绍的网络五子棋、网络斗地主等，这些程序的网络通信部分都可按上面介绍的方式来实现，只是不同游戏可能需要在网络上交换不同类型的数据，

这可能需要封装自己的网络协议。关于基于 Socket 通信的更详细介绍，读者可以对照《疯狂Java讲义》中网络编程的相关知识，这些知识是完全相通的。

13.2 使用URL访问网络资源

URL（Uniform Resource Locator）对象代表统一资源定位器，它是指向互联网"资源"的指针。资源可以是简单的文件或目录，也可以是对更复杂的对象的引用，例如对数据库或搜索引擎的查询。就通常情况而言，URL可以由协议名、主机、端口和资源组成，即满足如下格式：

例如如下的URL地址：

提示：

JDK中还提供了一个URI（Uniform Resource Identifiers）类，其实例代表一个统一资源标识符，Java的URI不能用于定位任何资源，它的唯一作用就是解析。与此对应的是，URL则包含一个可打开到达该资源的输入流，因此我们可以将URL理解成URI的特例。

URL类提供了多个构造器用于创建URL对象，一旦获得了URL对象之后，就可以调用如下常用方法来访问该URL对应的资源了。

➢ **String getFile（）**：获取此URL的资源名。

➢ **String getHost（）**：获取此URL的主机名。

➢ **String getPath（）**：获取此URL的路径部分。

➢ **int getPort（）**：获取此 URL 的端口号。

➢ **String getProtocol（）：** 获取此 URL 的协议名称。

➢ **String getQuery（）：** 获取此 URL 的查询字符串部分。

➢ **URLConnection openConnection（）：** 返回一个 URLConnection 对象，它表示到 URL所引用的远程对象的连接。

➢ **InputStream openStream（）：** 打开与此URL的连接，并返回一个用于读取该URL资源的InputStream。

13.2.1 使用URL读取网络资源

URL对象中前面几个方法都非常容易理解，而该对象提供的openStream（）可以读取该URL资源的InputStream，通过该方法可以非常方便地读取远程资源。

下面的程序示范了如何通过URL类读取远程资源。

程序清单：
codes\13\13.2\URLTest\app\src\main\java\org\crazyit\net\MainActivity.java

上面的程序两次调用了URL对象的openStream（）方法打开URL对应的资源的输入流，程序第一次使用BitmapFactory的decodeStream（InputStream）方法来解析该输入流中的图片；第二次则使用IO将输入流中的图片下载到本地。

该程序同样需要访问互联网，因此需要授予该程序访问网络的权限，也就是需要在AndroidManifest.xml文件中增加如下授权代码：

运行该程序，将可以看到如图13.4所示的输出。

如图13.4所显示的图片就是程序中URL对象所对应的图片，运行该程序不仅可以显示该URL对象所对应的图片，而且还会在手机文件系统的/data/data/org.crazyit.net/files/目录下生成crazyit.png图片，该图片就是通过URL从网络上下载的图片。

13.2.2 使用URLConnection提交请求

图13.4 使用URL读取网络图片

URL的openConnection（）方法将返回一个URLConnection对象，该对象表示应用程序和URL之间的通信连接。程序可以通过URLConnection实例向该URL发送请求，读取URL引用的资源。

通常创建一个和URL的连接，并发送请求、读取此URL引用的资源需要如下几个步骤。

1 通过调用URL对象的openConnection（）方法来创建URLConnection对象。

2 设置URLConnection的参数和普通请求属性。

3 如果只是发送GET方式的请求，那么使用connect方法建立和远程资源之间的实际连接即可；如果需要发送POST方式的请求，则需要获取URLConnection实例对应的输出流来发送请求参数。

4 远程资源变为可用，程序可以访问远程资源的头字段，或通过输入流读取远程资源的数据。

在建立和远程资源的实际连接之前，程序可以通过如下方法来设置请求头字段。

➢ **setAllowUserInteraction**：设置该URLConnection的allowUserInteraction请求头字段的值。

➢ **setDoInput**：设置该URLConnection的doInput请求头字段的值。

➢ **setDoOutput**：设置该URLConnection的doOutput请求头字段的值。

➢ **setIfModifiedSince**：设置该URLConnection的ifModifiedSince请求头字段的值。

➢ **setUseCaches**：设置该URLConnection的useCaches请求头字段的值。

除此之外，还可以使用如下方法来设置或增加通用的头字段。

➢ **setRequestProperty（String key，String value）**：设置该URLConnection的key请求头字段的值为value，如以下代码所示。

➢ **addRequestProperty（String key，String value）**：为该URLConnection的key请求头字段增加 value 值，该方法并不会覆盖原请求头字段的值，而是将新值追加到原请求头字段中。

当远程资源可用之后，程序可以使用以下方法来访问头字段和内容。

➢ **Object getContent（）**：获取该URLConnection的内容。

➢ **String getHeaderField（String name）**：获取指定响应头字段的值。

➢ **getInputStream（）**：返回该 URLConnection 对应的输入流，用于获取 URLConnection响应的内容。

➢ **getOutputStream（）**：返回该URLConnection对应的输出流，用于向URLConnection发送请求参数。

注意：

如果既要使用输入流读取URLConnection响应的内容，也要使用输出流发送请求参数，一定要先使用输出流，再使用输入流。

getHeaderField（）方法用于根据响应头字段来返回对应的值。而某些头字段由于经常需要访问，所以Java提供了以下方法来访问特定响应头字段的值。

➢ **getContentEncoding（）：** 获取content-encoding响应头字段的值。

➢ **getContentLength（）：** 获取content-length响应头字段的值。

➢ **getContentType（）：** 获取content-type响应头字段的值。

➢ **getDate（）：** 获取date响应头字段的值。

➢ **getExpiration（）：** 获取expires响应头字段的值。

➢ **getLastModified（）：** 获取last-modified响应头字段的值。

下面的程序示范了如何向Web站点发送GET请求、POST请求，并从Web站点取得响应。该程序中用到一个发送GET、POST请求的工具类，该工具类的代码如下。

程序清单：
codes\13\13.2\GetPostTest\app\src\main\java\org\crazyit\net\GetPostUtil.java

从上面的程序可以看出，如果需要发送GET请求，只要调用URLConnection的connect（）方法去建立实际的连接即可，如以上程

序中①号粗体字代码所示。如果需要发送POST请求，则需要获取URLConnection的OutputStream，然后再向网络中输出请求参数，如以上程序中②号粗体字代码所示。

提供了上面发送GET请求、POST请求的工具类之后，接下来就可以在Activity类中通过该工具类来发送请求了。该程序的界面中包含两个按钮，一个按钮用于发送GET请求，一个按钮用于发送POST请求。程序还提供了一个EditText来显示远程服务器的响应。该程序的界面布局很简单，故此处不再给出界面布局文件。该程序的Activity代码如下。

程序清单：
codes\13\13.2\GetPostTest\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的两行粗体字代码分别用于发送GET请求、POST请求，该程序所发送的GET请求、POST请求都是向本地局域网内http：//192.168.1.88：8888/abc应用下的两个页面发送的，这个应用实际上是部署在笔者本机上的Web应用。

提示：

abc这个Web应用的代码在光盘的codes\13\13.2路径下，这个Web应用需要部署在Web服务器（比如Tomcat 8.0）中才可使用。关于如何开发Web应用，如何安装、部署Web应用，可参考疯狂Java体系的《轻量级Java EE企业应用实战》一书。

在Web服务器中成功部署abc应用之后，运行上面的Android应用，单击“发送GET请求”按钮，将可以看到如图13.5所示的输出。

如果单击“发送POST请求”按钮，程序将会向abc应用下的login.jsp页面发送请求，并提交name=crazyit.org&pass=leegang请求参数，此时将可以看到如图13.6所示的输出。

图13.5 使用URLConnection对外发送GET请求

图13.6 使用URLConnection对外发送POST请求

从上面的介绍可以发现，借助于URLConnection类的帮助，应用程序可以非常方便地与指定站点交换信息，包括发送GET请求、POST请求，并获取网站的响应等。

13.3 使用HTTP访问网络

前面介绍了URLConnection可以非常方便地与指定站点交换信息，URLConnection还有一个子类：HttpURLConnection，HttpURLConnection在URLConnection的基础上做了进一步改进，增加了一些用于操作HTTP资源的便捷方法。

13.3.1 使用HttpURLConnection

HttpURLConnection继承了URLConnection，因此也可用于向指定网站发送GET请求、POST请求。它在URLConnection的基础上提供了如下便捷的方法。

➢ **int getResponseCode（）**：获取服务器的响应代码。

➢ **String getResponseMessage（）**：获取服务器的响应消息。

➢ **String getRequestMethod（）**：获取发送请求的方法。

➢ **void setRequestMethod（String method）**：设置发送请求的方法。

下面通过一个实用的示例来示范使用HttpURLConnection实现多线程下载。

实例：多线程下载

使用多线程下载文件可以更快地完成文件的下载，因为客户端启动多条线程进行下载就意味着服务器也需要为该客户端提供相应的服务。假设服务器同时最多服务100个用户，在服务器中一条线程对应一个用户，100条线程在计算机内并发执行，也就是由CPU划分时间片轮流执行，如果A应用使用了99条线程下载文件，那么相当于占用了99个用户的资源，自然就拥有了较快的下载速度。

注意：

实际上并不是客户端并发的下载线程越多，程序的下载速度就越快，因为当客户端开启太多的并发线程之后，应用程序需要维护每条线程的开销、线程同步的开销，这些开销反而会导致下载速度降低。

为了实现多线程下载，程序可按如下步骤进行。

1 创建URL对象。

2 获取指定URL对象所指向资源的大小（由getContentLength（）方法实现），此处用到了HttpURLConnection类。

3 在本地磁盘上创建一个与网络资源相同大小的空文件。

4 计算每条线程应该下载网络资源的哪个部分（从哪个字节开始，到哪个字节结束）。

5 依次创建、启动多条线程来下载网络资源的指定部分。

该程序提供的下载工具类的代码如下。

程序清单：
codes\13\13.3\MultiThreadDown\app\src\main\java\org\crazyit\net\DownUtil.java

上面的DownUtil工具类中包括一个DownloadThread内部类，该内部类的run（）方法负责打开远程资源的输入流，并调用InputStream的skip（int）方法跳过指定数量的字节，这样就让该线程读取由它自己负责下载的部分了。

提示：

可能有读者会发现，上面这个DownUtil类与《疯狂Java讲义》中介绍的多线程下载工具类基本相似。实际上也是这样，因此这个类并未用到任何与Android相关的知识，其本质就是使用JDK提供的HttpURLConnection类。

备注：

由于最新的Android平台上调用InputStream的skip（）方法时，并不能总是准确地跳过对应的字节数，因此程序专门实现了一个skipFully（）方法来跳过InputStream的指定字节数。上面程序中第二行粗体字代码调用了skipFully（）方法来跳过指定字节数，而被注释的第三行粗体字代码则直接调用了InputStream的skip（）方法—由于这个方法不太可靠，因此本程序没有直接使用该方法。

提供了上面的DownUtil工具类之后，接下来就可以在Activity中调用该DownUtil类来执行下载任务了。该程序界面中包含两个文本框：一个用于输入网络文件的源路径；另一个用于指定下载到本地的文件的文件名。该程序的界面比较简单，故此处不再给出界面布局代码。该程序的Activity代码如下。

程序清单：
codes\13\13.3\MultiThreadDown\app\src\main\java\org\crazyit\net\MainActivity.java

上面的Activity不仅使用了DownUtil来控制程序下载，而且程序还启动了一个定时器，该定时器控制每隔0.1秒查询一次下载进度，并通过程序中的进度条来显示任务的下载进度。

该程序不仅需要访问网络，还需要访问系统SD卡，在SD卡中创建文件，因此必须授予该程序访问网络、访问SD卡文件的权限，也就是在AndroidManifest.xml文件中增加如下配置：

运行该程序，将看到如图13.7所示的下载界面。

图13.7 多线程下载

提示：

上面的程序已经实现了多线程下载的核心代码，如果要实现断点下载，则还需要额外增加一个配置文件（读者可以发现所有的断点下载工具都会在下载开始时生成两个文件：一个是与网络资源具有相同大小的空文件；一个是配置文件），该配置文件分别记录每条线程已经下载到了哪个字节，当网络断开后再次开始下载时，每条线程根据配置文件里记录的位置向后下载即可。

13.3.2 使用Apache HttpClient

在一般情况下，如果只是需要向Web 站点的某个简单页面提交请求并获取服务器响应，则完全可以使用前面所介绍的HttpURLConnection

来完成。但在绝大部分情况下，Web站点的网页可能没这么简单，这些页面并不是通过一个简单的URL就可访问的，可能需要用户登录而且具有相应的权限才可访问该页面。在这种情况下，就需要涉及Session、Cookie的处理了，如果打算使用HttpURLConnection来处理这些细节，当然也是可能实现的，只是处理起来难度就大了。

为了更好地处理向Web站点请求，包括处理Session、Cookie等细节问题，Apache开源组织提供了一个HttpClient项目，看它的名称就知道，它是一个简单的HTTP客户端（并不是浏览器），可以用于发送HTTP请求，接收 HTTP响应。但不会缓存服务器的响应，不能执行HTML页面中嵌入的JavaScript代码；也不会对页面内容进行任何解析、处理。

提示：

简单来说，HttpClient就是一个增强版的HttpURLConnection，HttpURLConnection可以做的事情HttpClient全部可以做；HttpURLConnection没有提供的有些功能，HttpClient也提供了，但它只是关注于如何发送请求、接收响应，以及管理HTTP连接。

Android已经成功地集成了HttpClient，这意味着开发人员可以直接在Android应用中使用HttpClient来提交请求、接收响应。

使用HttpClient发送请求、接收响应很简单，只要如下几步即可。

1 创建HttpClient对象。

2 如果需要发送GET请求，则创建HttpGet对象；如果需要发送POST请求，则创建HttpPost对象。

3 如果需要发送请求参数，则可调用HttpGet、HttpPost共同的setParams（HttpParams params）方法来添加请求参数；对于HttpPost对象而言，也可调用setEntity（HttpEntity entity）方法来设置请求参数。

4 调用HttpClient对象的execute（HttpUriRequest request）方法发送请求，执行该方法返回一个HttpResponse。

5 调用HttpResponse的getAllHeaders（）、getHeaders（String name）等方法可获取服务器的响应头；调用HttpResponse的getEntity（）方法可获取HttpEntity对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

实例：访问被保护资源

下面的Android应用需要向指定页面发送请求，但该页面并不是一个简单的页面，只有当用户已经登录，而且登录用户的用户名是crazyit.org时才可访问该页面。如果使用HttpUrlConnection来访问该页面，那么需要处理的细节就太复杂了。下面将会借助于HttpClient来访问被保护页面。

访问Web应用中被保护页面，如果使用浏览器则十分简单，用户通过系统提供的登录页面登录系统，浏览器会负责维护与服务器之间的Session，如果用户登录的用户名、密码符合要求，就可以访问被保护资源了。

为了通过HttpClient来访问被保护页面，程序同样需要使用HttpClient来登录系统，只要应用程序使用同一个HttpClient发送请求，HttpClient就会自动维护与服务器之间的Session状态。也就是说，程序第一次使用 HttpClient登录系统后，接下来使用HttpClient即可访问被保护页面了。

提示：

虽然此处给出的实例只是访问被保护页面，但访问其他被保护资源也与此类似。程序只要第一次通过HttpClient登录系统，接下来即可通过该HttpClient访问被保护资源了。

下面是该程序的代码。

程序清单：
codes\13\13.3\HttpClientTest\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的①、②号代码先创建了一个HttpGet对象，接下来程序调用HttpClient的execute（）方法发送GET请求；程序中③、④号代码先创建了一个HttpPost对象，接下来程序调用了HttpClient的execute（）方法发送POST请求。上面的GET请求用于获取服务器上的被保护页面，POST请求用于登录系统。

运行该程序，单击“访问页面”按钮，将可以看到如图13.8所示的页面。

提示：

运行该程序需要Web应用的支持，读者应该先将codes/13/13.3目录下的foo应用部署到Web服务器（如Tomcat 7.0）中，然后再运行该应用。

从图13.8可以看出，程序直接向指定Web应用的被保护页面secret.jsp发送请求，程序将无法访问被保护页面，于是看到图13.8所示的页面。单击图13.8所示页面中的“登录系统”按钮，系统将会显示如图13.9所示的登录对话框。

在图13.9所示对话框的两个输入框中分别输入“crazyit.org”、“leegang”，然后单击“登录”按钮，系统将会向Web站点的login.jsp页面发送POST请求，并将用户输入的用户名、密码作为请求参数。如果用户名、密码正确，即可看到登录成功的提示。

登录成功后，HttpClient将会自动维护与服务器之间的连接，并维护与服务器之间的Session状态，再次单击程序中的“访问页面”按钮，即可

看到如图13.10所示的输出。

图13.8 访问被保护页面

图13.9 登录对话框

图13.10 访问被保护资源

从图13.10可以看出，此时使用HttpClient发送GET请求即可正常访问被保护资源，这就是因为前面使用了 HttpClient 登录了系统，而且 HttpClient 可以维护与服务器之间的 Session连接。

从上面的编程过程不难看出，使用Apache的HttpClient更加简单，而且它比HttpURLConnection提供了更多的功能。

13.4 Android 5.0增强的WebView

Android提供了WebView组件，从表面上看，这个组件与普通ImageView差不多，但实际上这个组件的功能要强大得多，WebView组件本身就是一个浏览器实现，Android 5.0增强的WebView基于Chromium M37，直接支持WebRTC、WebAudio和WebGL。

Chromium M37也包括对Web组件规范的原生支持，如自定义元素、阴影DOM、HTML导入和模板等，这意味着开发者可以直接在WebView中使用聚合（Polymer）和Material设计。

13.4.1 使用WebView浏览网页

WebView的用法与普通ImageView组件的用法基本相似，它提供了大量方法来执行浏览器操作，例如如下常用方法。

➢ **void goBack（）**：后退。

➢ **void goForward（）**：前进。

➢ **void loadUrl（String url）**：加载指定URL对应的网页。

➢ **boolean zoomIn（）**：放大网页。

➢ **boolean zoomOut（）**：缩小网页。

当然，WebView组件还包含了大量方法，具体以Android API文档为准。

下面的程序将基于WebView来开发一个简单的浏览器。

实例：迷你浏览器

该程序的界面中包含两个组件：一个文本框用于接收用户输入的URL；一个WebView用于加载并显示该URL对应的页面。该程序的界面布局代码如下。

程序清单：
codes\13\13.4\MiniBrowser\app\src\main\res\layout\main.xml

下面程序则主要通过WebView的loadUrl（String url）来加载、显示指定URL对应的页面。**程序清单：**
codes\13\13.4\MiniBrowser\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的粗体字代码是该程序的关键，程序调用WebView的loadUrl（String url）方法加载、显示该URL对应的网页—至于该WebView如何发送请求，如何解析服务器响应，这些细节对用户来说是透明的。

该应用需要访问互联网，同样需要在AndroidManifest.xml文件中增加如下配置：

运行该程序，在文本框中输入想访问的站点，并单击手机的“搜索”按钮，将可以看到如图13.11所示的输出。

正如从图13.11所看到的，使用WebView开发浏览器十分简单，如果读者愿意多花时间对该程序界面进行美化，并为程序提供前进、后退、刷新等按钮，即可开发出一个实用的浏览器来代替Android系统自带的浏览器。

提示：

Android系统自带的浏览器其实也是基于开源的WebKit引擎实现的。

图13.11 使用WebView浏览指定网页

13.4.2 使用WebView加载HTML代码

前面看到使用EditText显示HTML字符串时十分别扭，EditText不会对HTML标签进行任何解析，而是直接把所有HTML标签都显示出来—就像用普通记事本显示一样；如果应用程序想重新对HTML字符串进行解析，当成HTML页面来显示，也是可以的。

WebView提供了一个loadData（String data，String mimeType，String encoding）方法，该方法可用于加载并显示HTML代码。但在实

际使用过程中，笔者发现这个方法有一个小问题：当它加载包含中文的HTML内容时，WebView将会显示乱码。

好在WebView还提供了一个loadDataWithBaseURL（String baseUrl，String data，String mimeType，String encoding，String historyUrl）方法，该方法是loadData（String data，String mimeType，String encoding）方法的增强版，它不会产生乱码。关于该方法的几个参数简单说明一下。

➢ **data**：指定需要加载的HTML代码。

➢ **mimeType**：指定HTML代码的MIME类型，对于HTML代码可指定为text/html。

➢ **encoding**：指定HTML代码编码所用的字符集。比如指定为GBK。

下面的程序简单示范了如何使用WebView来加载HTML代码。

程序清单：
codes\13\13.4\ViewHtml\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的粗体字代码就是该程序的关键，这行代码负责加载指定的HTML页面，并将它显示出来。运行该程序，将可以看到如图13.12所示的输出。

图13.12 使用WebView加载、显示HTML代码

13.4.3 使用WebView中的JavaScript调用Android方法

很多时候，WebView加载的页面上是带JavaScript脚本的，比如页面上有一个按钮，用户单击按钮时将会弹出一个提示框，或打开一个列表框等。由于该按钮是HTML页面上的按钮，它只能激发一段JavaScript脚本，这就需要让JavaScript脚本来调用Android方法了。

为了让WebView中的JavaScript脚本调用Android方法，WebView提供了一个配套的WebSettings工具类，该工具类提供了大量方法来管理WebView的选项设置，其中它的setJavaScriptEnabled（true）即可让WebView中的JavaScript脚本来调用Android方法。除此之外，为了把Android对象暴露给WebView中的JavaScript代码，WebView提供了addJavascriptInterface（Object object，String name）方法，该方法负责把object对象暴露成JavaScript中的name对象。

从上面的介绍可以看出，在WebView的JavaScript中调用Android方法只要如下三个步骤。

1 调用WebView关联的WebSettings的setJavaScriptEnabled（true）启用JavaScript调用功能。

2 调用WebView的addJavascriptInterface（Object object，String name）方法将object对象暴露给JavaScript脚本。

3 在JavaScript脚本中通过刚才暴露的name对象调用Android方法。

提示：

如果读者有DWR开发经验，应该很好理解Android此处的设计，DWR通过使用配置，可以让服务端的Java对象暴露给JavaScript脚本；Android则通过WebView的addJavascriptInterface（）方法把Android应用中的对象暴露给JavaScript脚本一最后实现的效果是相同的：JavaScript脚本可以直接调用Java对象的方法。

下面的示例示范了如何在JavaScript中调用Android方法。该示例的界面布局很简单，它包含了一个普通的WebView组件，用于显示HTML页面。该示例的Activity代码如下。

程序清单：
codes\13\13.4\JsCallAndroid\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的第一行粗体字代码开启了JavaScript调用Android方法的功能，第二行粗体字代码则负责将Androd应用中的MyObject对象暴露给JavaScript脚本，暴露成JavaScript脚本中名为myObj的对象。

MyObject是一个自定义的Java类，开发者可以根据业务需要提供任意多的方法，本示例只为MyObject定义了两个方法。下面是MyObject类的代码。

程序清单：
codes\13\13.4\JsCallAndroid\app\src\main\java\org\crazyit\net\MyObject.java

正如上面的代码所示，MyObject中包含了两个方法—showToast（）和showList（）方法，且这两个方法使用了@JavascriptInterface修饰，因此这两个方法将会暴露给JavaScript脚本，从而允许JavaScript脚本通过myObj来调用这两个方法。下面是HTML页面的代码。

程序清单：
codes\13\13.4\JsCallAndroid\app\src\main\assets\test.html

正如上面的两行粗体字代码所示，当用户单击该页面上的两个按钮时，该页面的JavaScrip脚本会通过myObj调用Android方法。运行该示例，单击第一个按钮，可以看到如图13.13所示的界面。

如果用户单击第二个按钮，该页面的JavaScript脚本将会通过myObj调用Android的showList（）方法，此时将看到如图13.14所示的对话框。

图13.13 JS调用Android方法

图13.14 JS调用Android方法

13.5 使用Web Service进行网络编程

Android应用通常都运行在手机平台上，手机系统的硬件资源是有限的，不管是存储能力还是计算能力都有限，在Android系统上开发、运行一些单用户、小型应用是可能的，但对于需要进行大量的数据处理、复杂计算的应用，还是只能部署在远程服务器上，Android应用将只是充当这些应用的客户端。

为了让Android应用与远程服务器进行交互，可以借助于Java的RMI技术，但这要求远程服务器程序必须采用Java实现；也可以借助于CORBA技术，但这种技术显得过于复杂。除此之外，Web Service是一种不错的选择。

13.5.1 Web Service平台概述

Web Service平台主要涉及的技术有SOAP（Simple Object Access Protocol，简单对象访问协议）、WSDL（Web Service Description Language，Web Service描述语言）、UDDI（Universal Description，Description and Integration，统一描述、发现和整合协议）。

1.SOAP（简单对象访问协议）

SOAP是一种具有扩展性的XML消息协议。SOAP允许一个应用程序向另一个应用程序发送XML消息，SOAP消息是从SOAP发送者传至SOAP接收者的单路消息，任何应用程序均可作为发送者或接收者。SOAP仅定义消息结构和消息处理的协议，与底层的传输协议独立。因此，SOAP协议能通过HTTP、JMS或SMTP协议传输。

SOAP依赖于XML文档来构建，一条SOAP消息就是一份特定的XML文档，SOAP消息包含如下三个主要元素。

➢ <Envelope.../>根元素，SOAP消息对应的XML文档以该元素作为根元素。

➢ 可选的<Header../>元素，包含SOAP消息的头信息。

➢ 必需的<Body../>元素，包含所有的调用和响应信息。

就目前的SOAP消息的结构来看，<Envelope.../>根元素通常只能包含两个子元素，其中第一个子元素是可选的<Header../>元素，第二个子元素是必需的<Body../>元素。

图13.15显示了SOAP消息的基本结构。

图13.15 SOAP消息的基本结构

2.WSDL（Web Service描述语言）

WSDL使用XML描述Web Service，包括访问和使用Web Service所必需的信息，定义该Web Service的位置、功能及如何通信等描述信息。

一般来说，只要调用者能够获取Web Service对应的WSDL，就可以从中了解它所提供的服务及如何调用Web Service了。因为一份WSDL文件清晰地定义了三个方面的内容。

➢ **WHAT部分**：用于定义Web Service所提供的操作（或方法），也就是Web Service能做些什么。由WSDL中的<types.../>、<message.../>和<portType.../>元素定义。

➢ **HOW部分**：用于定义如何访问Web S ervice，包括数据格式详情和访问Web S ervice操作的必要协议。也就是定义了如何访问Web Service。

➢ **WHERE部分**：用于定义Web Service位于何处。该部分使用<service.../>元素定义，可在WSDL文件的最后部分看到<service.../>元素。

一份WSDL文档通常可分为两个部分。

➢ 第一个部分定义了服务接口，它在 WSDL 中由<message.../>和<portType.../>两个元素组成，其中<message.../>元素定义了操作的交互方式；而<portType.../>元素里则可包含任意数量的<operation.../>元素，每个<operation.../>元素代表一个允许远程调用的操作（即方法）。

➢ 第二个部分定义了服务实现，它在 WSDL 中由<binding.../>和<service.../>两个元素组成，其中<binding.../>元素定义使用特定的通信协议、数据编码模型和底层通信协议，将 Web S ervice 服务接口定义映射到具体实现；而<service.../>元素则包含一系列的<port.../>子元素，<port.../>子元素将会把绑定机制、服务访问协议和端点地址结合在一起。

图13.16显示了WSDL文档的结构模型。

3.UDDI（统一描述、发现和整合协议）

UDDI是一套信息注册规范，它具有如下特点。

➢ 基于Web。

➢ 分布式。

UDDI包括一组允许企业向外注册Web Service，以使其他企业发现访问的实现标准。UDDI的核心组件是UDDI注册中心，它使用XML文件来描述企业及其提供的Web Service。

通过使用UDDI，Web Service提供者可以对外注册Web Service，从而允许其他企业来调用该企业注册的Web Service。Web Service提供者通过UDDI注册中心的Web界面，将它所提供的Web Service的信息加入UDDI注册中心，该Web Service就可以被发现和调用了。

Web Service使用者也通过UDDI注册中心查找、发现自己所需的服务。当Web Service使用者找到自己所需的服务之后，可以将自己绑定到指定的Web Service提供者，再根据该Web Service对应的WSDL文档来调用对方的服务。

Web Service大致的运行模式如图13.17所示。

图13.16 WSDL文档的结构模型

图13.17 Web Service运行模式示意图

13.5.2 使用Android应用调用Web Service

Java本身提供了丰富的Web Service支持，比如Sun公司制定的JAX-WS 2规范，还有Apache开源组织所提供的Axis1、Axis2、CXF等，这些技术不仅可以用于非常方便地对外提供Web Service，也可以用于简化Web Service的客户端编程。

对于手机等小型设备而言，它们的计算资源、存储资源都十分有限，因此Android应用不大可能需要对外提供Web Service，Android应用通

常只是充当Web Service的客户端，调用远程Web Service。

Google为Android平台开发Web Service客户端提供了ksoap2-android项目，但这个项目并未直接集成在Android平台中，还需要开发人员自行下载。

为Android应用增加ksoap2-android支持请按如下步骤进行。

1 登录http：//code.google.com/p/ksoap2-android/站点，该站点有介绍下载ksoap2-android项目的方法。

2 下载ksoap2-android项目的ksoap2-android-assembly-3.4.0-jar-with-dependencies.jar包。如果读者下载有困难，也可直接使用光盘中codes\13\13.5目录下的该文件。

3 将下载得到的JAR包添加到Android项目的libs目录下，并通过Project面板选中该JAR包后，通过右键菜单的“Add as library”菜单项添加该JAR包，即可在Android Studio左边看到如图13.18所示的项目管理树。

图13.18 为Android项目添加ksoap2-android包

为Android项目添加了ksoap2-android包之后，接下来借助于ksoap2-android项目来调用Web Service所暴露的操作。

使用ksoap2-android调用Web Service操作的步骤如下。

1 创建HttpTransportSE对象，该对象用于调用Web Service操作。

2 创建SoapSerializationEnvelope对象。

提示：

从名称来看，SoapSerializationEnvelope代表一个SOAP消息封包；但ksoap2-android项目对SoapSerializationEnvelope的处理比较特殊，它是HttpTransportSE调用Web Service时信息的载体—客户端需要传入的参数，需要通过SoapSerializationEnvelope对象的bodyOut属性传给服务器；服务器响应生成的SOAP消息也通过该对象的bodyIn属性来获取。

3 创建SoapObject对象，创建该对象时需要传入所要调用Web Service的命名空间、Web Service方法名。

4 如果有参数需要传给Web Service服务器端，则调用SoapObject对象的addProperty（String name，Object value）方法来设置参数，该方法的name参数指定参数名；value参数指定参数值。

5 调用SoapSerializationEnvelope的setOutputSoapObject（）方法，或者直接对bodyOut属性赋值，将前两步创建的SoapObject对象设为SoapSerializationEnvelope的传出SOAP消息体。

6 调用对象的call（）方法，并以SoapSerializationEnvelope作为参数调用远程Web Service。

7 调用完成后，访问SoapSerializationEnvelope对象的bodyIn属性，该属性返回一个SoapObject对象，该对象就代表了Web Service的返回消息。解析该SoapObject对象，即可获取调用Web Service的返回值。

实例：调用基于CXF的Web Service

下面的程序使用CXF开发了一个Web Service，该Web Service对应的WSDL文档如图13.19所示。

提示：

使用Ant通过光盘中codes\13\13.5\WsServer目录下的build.xml文件运行Web Service服务器端—当服务器端运行起来之后才可通过浏览器看到如图13.19所示的WSDL文档。

正如图13.19所看到的，Web Service客户端只要访问到Web Service的WSDL文档，即可根据该文档来获取调用Web Service操作的必要信息。

图13.19 Web Service提供的WSDL文档

提示：

由于本书的重点是介绍Android应用调用Web Service，因此不会涉及如何使用CXF开发Web Service，以及Web Service的WSDL文档中各元素的作用、意义等知识；如果读者对Web Service服务器端的开发有兴趣，可以参考疯狂Java体系的《疯狂XML讲义》一书。

通过图13.19所示的WSDL文档了解到调用Web Service的方法名、所需的参数名之后，接下来就可以在Android程序中通过ksoap2-android调用Web Service操作了。

下面的程序示范了如何通过ksoap2-android来调用Web Service操作。该程序的界面很简单，界面中只定义了两个文本框来装载服务器响应，因此此处不再给出界面布局代码。该程序的Activity代码如下。

程序清单：
codes\13\13.5\CallWs\app\src\main\java\org\crazyit\net\MainActivity.java

上面程序中的粗体字代码就代表了使用ksoap2-android来调用Web Service操作的关键的7个步骤。

Web Service服务器端运行起来之后，运行上面的Android应用，即可看到如图13.20所示的输出。

从图13.20所示的两个文本框中看到的内容就是调用Web Service所返回的数据。由此可见，不管远程Web Service提供的服务功能多么强大、业务实现多么复杂，对于Android客户端是完全透明的，Android只要送出相应的请求参数，服务器就会返回包含结果的SOAP消息。借助于Web Service这个桥梁，在Android应用中就可以实现功能非常强大的应用—反正具体的业务逻辑由Web Service服务器端实现，Android客户端只要调用Web Service服务即可。

图13.20 调用Web Service

不管远程Web Service的功能多么复杂，对于Android客户端而言，它只要通过Web Service获取对方提供的数据，并将这些数据“整合”到自己的应用中即可，因此Web Service为Android应用提供了强大的后台支持。

13.6 本章小结

本章主要介绍了Android网络编程的相关知识。由于Android完全支持JDK网络编程中的ServerSocket、Socket、DatagramSocket、Datagrampacket、MulticastSocket等API，也支持JDK内置的URL、URLConnection、HttpURLConnection等工具类，因此如果读者已经具有网络编程的经验，这些经验完全适用于Android网络编程。除此之外，Android还内置了Apache HttpClient支持，通过HttpClient可以非常方便地维持与服务器的会话状态、发送请求、获取响应等。虽然Android本身没有内置Web Service支持，但本章介绍了通过ksoap2-android项目调用远程Web Service的相关内容，这也是需要读者掌握的内容。

第14章 管理Android手机桌面

本章要点

Android手机桌面的概念

改变手机壁纸的API

动态壁纸的API

开发动态壁纸

向Android桌面添加快捷方式

桌面控件的概念

添加桌面控件

带数据集的桌面控件

RemoteViewsService和RemoteViewsFactory

Android系统提供了一个桌面—也就是用户启动后第一次看到的界面，如图14.1所示。从图14.1可以看出，手机桌面的作用类似于PC的桌面，桌面上通常用于放置一些常用的程序和功能。

在Android桌面上首先看到的壁纸，也就是手机桌面上的那张图片，接着可以看到手机桌面上规则排列的多个图标，这些图标就是Android桌面组件，分别代表快捷方式与桌面控件；每个快捷方式只占用桌面的一个摆放位置；桌面控件则可以很大，一个桌面控件就可以占据多个摆放位置。

Android系统提供了很好的可扩展性，开发者完全可以在程序中管理Android桌面，包括改变系统壁纸、管理快捷方式与创建桌面控件等。

图14.1 Android桌面

14.1 管理手机桌面

在默认情况下，Android 系统的桌面上除了一张壁纸之外，并不会显示如图14.1所示的组件。Android系统允许普通用户动态地添加、删除桌面组件。

14.1.1 删除桌面组件

在默认情况下，Android 系统的桌面上会显示两个桌面组件：相机和模拟时钟，模拟时钟就是搜索条下面的组件。一旦熟悉了Android系统之后，用户可能并不希望总是看到这两个组件，此时可能会考虑删除该组件。

删除Android桌面组件的步骤如下。

1 在屏幕上长按指定组件，直到桌面上方出现“删除”图标。

2 将指定组件拖到桌面上方的“删除”图标上，如图14.2所示。

14.1.2 添加桌面组件

Android桌面组件可分为快捷方式与桌面控件两种，两种组件的添加方式略有不同。

为了在Android桌面上添加快捷方式，可以按如下步骤进行。

1 进入手机的程序列表界面，长按需要添加快捷方式的程序，此时可以看到程序图标会自动切换到桌面，如图14.3所示。

图14.2 删除桌面组件

图14.3 添加快捷方式

2 拖动该程序图标，将它放到指定位置即可。

为了在Android桌面上添加桌面控件，可以按如下步骤进行。

1 长按手机桌面，然后单击屏幕右下角的“WIDGETS”按钮，系统进入如图14.4所示的Widget列表界面。

2 长按需要添加的桌面控件，此时可以看到桌面控件会自动切换到桌面，如图14.5所示。

图14.4 桌面控件列表

图14.5 添加桌面控件

3 拖动该桌面控件，将它放到指定位置即可。

Android系统提供的上面的操作能让用户动态地管理手机桌面，但这些能在桌面显示的组件都是Android系统本身提供的，接下来将会介绍如何通过应用程序来管理桌面组件。

14.2 改变手机壁纸

前面讲解系统服务时已经介绍了一个“定时更换壁纸”的应用，在那个应用中，当用户启动相应的Service之后，该Service将会根据AlarmManager来定时改变手机壁纸。

Android允许使用WallpaperManager来改变手机壁纸，该对象中改变手机壁纸的方法如下。

➢ **setBitmap（Bitmap bitmap）**：将壁纸设置为bitmap所代表的位图。

➢ **setResource（int resid）**：将壁纸设置为resid资源所代表的图片。

➢ **setStream（InputStream data）**：将壁纸设置为data数据所代表的图片。

前面介绍的这种改变手机壁纸的方式只是动态地切换不同图片作为手机壁纸，此处不再介绍调用WallpaperManager来改变手机壁纸的示例。

除此之外，Android系统还提供了一种动态壁纸的功能，例如用户在手机桌面上长按，系统将会显示如图14.6所示的更改壁纸的方式。

在 Android 手机屏幕下方会看到大量系统内置的壁纸，其中前面几项是普通的静态图片壁纸，后面则包含了图14.6所示的"动态壁纸"列表项，这是Android系统默认提供的几个动态壁纸。除此之外，开发者还可以开发任意的动态壁纸。

图14.6 动态壁纸

14.2.1 开发动态壁纸（Live Wallpapers)

所谓动态壁纸，就是指手机桌面不再是简单的图片，而是运行中的动画，这个动画是由程序实时绘制的，因此被称为动态壁纸。

为了开发动态壁纸，Android提供了WallpaperService基类，动态壁纸的实现类需要继承该基类。在Android应用中开发动态壁纸的步骤如

下。

1 开发一个子类继承WallpaperService基类。

2 继承WallpaperService基类时必须重写onCreateEngine（）方法，该方法返回WallpaperService.Engine子类对象。

3 开发者需要实现WallpaperService.Engine子类，并重写其中的public void onVisibilityChanged （boolean visible）、public void onOffsetsChanged（）方法。不仅如此，由于WallpaperService.Engine子类采用了与SurfaceView相同的绘图机制，因此还可有选择性地重写SurfaceHolder.Callback中的三个方法。重写这些方法时可通过SurfaceHolder动态地绘制图形。

实例：蜿蜒壁纸

本实例将通过一个不断变换的矩形在桌面上绘制动态壁纸。下面的LiveWallPaper类就代表了动态壁纸服务，程序代码如下。

程序清单：
codes\14\14.2\LiveWallPaper\app\src\main\java\org\crazyit\desktop\LiveWallpaper.java

上面程序中的粗体字代码就是实现动态壁纸Service的关键代码。这两段粗体字代码重写了WallpaperService.Engine的onVisibilityChanged（）、onOffsetsChanged（）方法，并指定当桌面显示时调用drawFrame（）方法进行绘制，drawFrame（）方法绘制完成后通过Handler对象指定0.1秒后重绘，如②号粗体字代码所示。

上面的程序为了实现“蜿蜒前行”的效果，使用循环控制绘制了count个矩形，每绘制一次，count的值将会加1；而且程序控制对Canvas进行位移、旋转两种坐标变换，通过这种方式即可实现“蜿蜒前行”的动画效果。

定义了该Service类之后，接下来还需要在AndroidManifest.xml文件中配置该Service。配置动态壁纸Service与配置普通Service存在小小的区别，它需要指定如下两项。

➢ 指定运行动态壁纸，需要android.permission.BIND_WALLPAPER权限。

➢ 为动态壁纸指定meta-data配置。

在AndroidManifest.xml文件中配置动态壁纸，也就是需要增加如下配置片段。

程序清单：
codes\14\14.2\LiveWallPaper\app\src\main\AndroidManifest.xml

上面的配置文件中粗体字代码就是配置动态壁纸的关键代码。上面的配置文件中指定了动态壁纸的meta-data放在@xml/livewallpaper中定义，因此程序还需要在res\xml\目录下增加一个livewallpaper.xml文件，该文件内容如下。

程序清单：
codes\14\14.2\LiveWallPaper\app\src\main\res\xml\livewallpaper.xml

把上面的程序部署到模拟器上，该程序不能直接运行，需要按前面介绍的步骤来设置动态壁纸。

用户也可通过Android手机的Settings应用来设置动态壁纸，用户通过单击“Settings”→“Display”→“Wallpaper”→“Live Wallpapers”即可看到如图14.7所示的列表项。

单击“蜿蜒壁纸”项，系统进入预览该动态壁纸的界面，如图14.8所示。

在图14.8所示的界面中可以看到“蜿蜒前行”的动画不停执行，单击“Set wallpaper”按钮，即可应用我们刚刚所开发的动态壁纸程序。再次切换到Android系统桌面，将可看到桌面上显示如图14.9所示的效果。

图14.7 选择动态壁纸

图14.8 预览动态壁纸

图14.9 动态壁纸效果

虽然上面的程序介绍的动态壁纸只是在桌面上绘制蜿蜒前行的矩形，但是这种动态壁纸可以提供开发者在桌面上自由绘图的能力，因此到底要在系统桌面上绘制什么，完全取决于用户自己的选择。

14.3 通过程序添加快捷方式

对于一个希望拥有更多用户的应用来说，用户桌面可以说是所有软件的必争之地，如果用户在手机桌面上建立了该软件的快捷方式，用户

将会更频繁地使用该软件。因此，所有的Android程序都应该允许用户把软件的快捷方式添加到桌面上。

在程序中把一个软件的快捷方式添加到桌面上，只需要如下三步即可。

1 创建一个添加快捷方式的Intent，该Intent的Action属性值应该为com.android.launcher.action.INSTALL_SHORTCUT。

2 通过为该Intent添加Extra属性来设置快捷方式的标题、图标及快捷方式对应启动的程序。

3 调用sendBroadcast（）方法发送广播即可添加快捷方式。

提示：

就像Windows时代，用户桌面是“兵家”必争之地，所有应用总是试图在用户桌面上创建自己的快捷方式，这样就可以让用户时时刻刻看到自己的程序，让用户经常使用自己的程序，从而依赖自己的程序。但需要注意的是，也有些用户对这种行为比较反感，他们喜欢自己的桌面简简单单，因此他们会把这些程序添加的图标删除，甚至会直接把程序卸载。所以希望读者注意，不要太激进，引起用户反感会导致程序被删除。

实例：让程序占领桌面

下面的程序是对第7章中一个动画程序的修改。该程序提供了一个按钮，用户单击该按钮即可在桌面上建立该程序的快捷方式。程序代码如下。

程序清单：
codes\14\14.3\AddShortcut\app\src\main\java\org\crazyit\desktop\MainActivity.java

上面程序中的粗体字代码就是为程序添加快捷方式的关键代码，其中①、②、③号粗体字代码就是前面所介绍的第1～3步。

在程序中添加快捷方式需要相应的权限，因此别忘了在AndroidManifest.xml文件中添加如下配置片段：

运行该程序，并单击该程序界面中的“添加快捷键”按钮，即可在桌面上添加一个快捷方式。返回桌面，即可在桌面上看到如图14.10所示的快捷方式。

上面的实例把添加快捷方式的代码放在事件处理方法中完成，这表明只有当用户单击该按钮时才会添加快捷方式—这种方式比较温和；如果开发者希望强行在用户桌面上添加快捷方式，则可以将上面添加快捷方式的代码放在onCreate（）方法中—这种方式容易引起用户反感。

图14.10 快捷方式

14.4 管理桌面控件

所谓桌面控件，就是指能直接显示在Android系统桌面上的小程序，比如在图14.1中所看到的直接显示在桌面上的模拟时钟。一般来说，开发者可以把一些用户使用十分频繁的程序，比如时钟、指南针、日历等程序做成桌面控件，这样用户就可以直接在桌面上看到程序的运行界面了。

14.4.1 开发桌面控件

桌面控件是通过BroadcastReceiver的形式来进行控制的，因此每个桌面控件都对应于一个BroadcastReceiver。为了简化桌面控件的开发，Android系统提供了一个AppWidgetProvider类，它就是BroadcastReceiver的子类。也就是说，开发者开发桌面控件只要继承AppWidgetProvider类即可。

为了开发桌面控件，开发者只要开发一个继承 AppWidgetProvider 的子类，并重写AppWidgetProvider不同状态的生命周期方法即可。AppWidgetProvider里提供了如下4个不同的生命周期方法。

➢ **onUpdate（）：** 负责更新桌面控件的方法；实现桌面控件通常会考虑重写该方法。

➢ **onDeleted（）：** 当一个或多个桌面控件被删除时回调该方法。

➢ **onEnabled（）：** 当接收到ACTION_APPWIDGET_ENABLED Broadcast时回调该方法。

➢ **onDisabled（）：** 当接收到ACTION_APPWIDGET_DISABLED Broadcast时回调该方法。

一般来说，开发桌面控件只需要定义一个AppWidgetProvider的子类，并重写它的onUpdate（）方法即可。重写该方法按如下步骤进行。

1 创建一个RemoteViews对象，创建该对象时可以指定加载指定的界面布局文件。

2 如果需要改变上一步所加载的界面布局文件的内容，则可通过RemoteViews对象进行修改。

提示：

一般来说，RemoteViews所加载的界面中主要包含ImageView和TextView两种组件，RemoteViews提供了修改这两种组件的内容的方法。

3 创建一个ComponentName对象。

4 调用AppWidgetManager更新桌面控件。

提示：

将上面4个步骤归纳起来，其核心代码就是使用AppWidgetManager通过RemteViews来更新AppWidgetProvider的子类实例（需将它包装成ComponentName对象）。

下面的示例程序中的桌面控件十分简单，该控件中只是包含了一张简单的图片，此处不再给出界面布局文件。添加桌面控件的代码如下。

程序清单：
codes\14\14.4\DesktopApp\app\src\main\java\org\crazyit\desktop\DesktopApp.java

上面的程序中重写了AppWidgetProvider的onUpdate（）方法，程序中①、②、③、④号粗体字代码正好对应于上面所介绍的4个步骤。

由于AppWidgetProvider继承了BroadcastReceiver，因此AppWidgetProvider的本质还是一个BroadcastReceiver，为此需要在AndroidManifest.xml文件中使用<receiver.../>元素来配置它，配置该元素时需要为它指定相应的<intent-filter.../>和<meta-data.../>。

为了在系统中添加该桌面控件，还需要在AndroidManifest.xml文件中添加如下配置片段：

上面配置文件中的粗体字代码指定该桌面控件使用@xml/appwidget_provider作为meta-data，因此还需要在应用的res\xml目录下添加appwidget_provider.xml文件。该文件的内容如下。

程序清单：
codes\14\14.4\DesktopApp\app\src\main\res\xml\appwidget_provider.xml

上面的XML配置文件中使用了<appwidget-provider.../>元素来描述桌面控件的基本信息，其中的注释已经详细说明了各属性的作用。

把该应用安装到Android系统上，再次进入图14.4所示的Widget列表界面，将会看到如图14.11所示的界面。

正如图14.11所示，刚才开发的桌面控件已经显示出来了，长按该列表项就会把该小控件添加到桌面上。再次返回桌面，将会看到如图14.12所示的桌面控件。

图14.11 添加桌面控件

图14.12 桌面控件

从图14.12可以看出，通过使用桌面控件，开发者可以把应用程序的运行界面直接放到桌面上，至于如何控制程序界面上显示的内容，完全取决于项目的业务需求。下面以开发一个液晶时钟为例，来介绍桌面控件的用法。

实例：液晶时钟

为了实现一个液晶时钟的桌面组件，开发者需要在程序界面上定义8个ImageView，其中6个ImageView用于显示小时、分钟、秒钟的数字，另外两个ImageView用于显示小时、分钟、秒钟之间的冒号。

为了让桌面组件实时地显示当前时间，程序需要每隔1秒更新一次程序界面上的6个ImageView，让它们显示当前小时、分钟、秒钟的数字即可。

液晶时钟的代码如下。

程序清单：
codes\14\14.4\LedClock\app\src\main\java\org\crazyit\desktop\LedClock.java

上面程序中的粗体字代码将会根据程序的时间字符串动态更新6个ImageView所显示的液晶数字图片，这样即可通过液晶数字来显示当前时间了。

在AndroidManifest.xml文件中添加如下代码片段来定义该桌面控件：

上面配置文件中的粗体字代码指定了该液晶时钟的meta-data为@xml/my_clock，还需要在res/xml路径下增加一个my_clock.xml文件，该文件的内容与前一个桌面控件的meta-data文件大致相同，此处不再给出。

通过Android系统把该液晶时钟小控件添加到桌面上，将可以看到Android桌面显示如图14.13所示的液晶时钟。

图14.13 液晶时钟控件

正如图14.13中的液晶时钟所示，这种桌面控件显示了程序的运行状态，它的运行界面可以是动态改变的。因此还可以通过桌面控件把股

票走势图之类添加到桌面上，让用户可以一眼看到。总之，这种桌面控件给了开发者更多的想象空间。

14.4.2 显示带数据集的桌面控件

Android为RemoteViews提供了如下方法。

➢ **setRemoteAdapter（int viewId，Intent intent）**：该方法可以使用Intent更新RemoteViews中viewId对应的组件。

上面方法的Intent参数应该封装一个RemoteViewsService参数，该参数虽然继承了Service组件，但它的主要作用是为RemoteViews中viewId对应的组件提供列表项。

由于Intent参数负责提供列表项，因此viewId参数对应的组件可以是ListView、GridView、StackView和AdapterViewFlipper等，这些组件都是AdapterView的子类，由此可见，RemoteViewsService负责提供的对象，应该是一个类似于Adapter的对象。

RemoteViewsService通常用于被继承，继承该基类时需要重写它的onGetViewFactory（）方法，该方法需要返回一个类似于Adapter的对象一但不是Adapter，而是RemoteViewsFactory对象，RemoteViewsFactory的功能完全类似于Adapter。

下面是本示例提供的RemoteViewsService类。

程序清单：
codes\14\14.4\AppWidgetCollection\app\src\main\java\org\crazyit\desktop\StackWidgetService.java

上面程序中的①号粗体字代码返回了一个StackRemoteViewsFactory对象，该对象的作用类似于Adapter对象，就是负责返回多个列表项。与

普通Adapter不同的是，Adapter返回的多个列表项只要是View组件即可；但StackRemoteViewsFactory对象返回的列表项必须是RemoteViews组件，因此上面程序中的粗体字代码重写getViewAt（int position）方法时需要创建并返回一个RemoteViews对象。

上面程序加载的布局文件是位于/res/layout目录下的widget_item.xml，该文件只包含一个简单的ImageView组件。该布局文件代码如下。

程序清单：
codes\14\14.4\AppWidgetCollection\app\src\main\res\layout\widget_item.xml

提供了StackWidgetService之后，接下来开发AppWidgetProvider子类与开发普通AppWidgetProvider子类的步骤基本相同，只是此时不再调用RemoteViews的setImageViewResource（）或setTextViewText（），而是调用setRemoteAdapter（int viewId，Intent intent）方法。

下面是本示例中AppWidgetProvider子类的代码。

程序清单：
codes\14\14.4\AppWidgetCollection\app\src\main\java\org\crazyit\desktop\StackWidgetProvider.java

上面的①号粗体字代码调用RemoteViews的setRemoteAdapter（）方法进行设置，该方法负责为RemoteViews中的StackView设置RemoteViewsFactory对象—该RemoteViewsFactory对象负责提供多个列表项。接下来②号粗体字代码调用AppWidgetManager的

updateAppWidget（）进行更新，这行代码与开发普通的AppWidgetProvider基本相同。

由于这种带数据集的桌面控件同时需要AppWidgetProvider和RemoteViewsService，因此在AndroidManifest.xml文件中需要同时配置<receiver.../>元素和<service.../>元素。本示例的配置片段如下。

程序清单：
codes\14\14.4\AppWidgetCollection\app\src\main\AndroidManifest.xml

该示例同样需要在/res/xml目录下添加一个stackwidgetinfo.xml元数据配置文件，该文件与前面开发普通AppWidgetProvider基本相似，此处不再给出。

将该应用部署到模拟器上，并将该桌面控件添加到桌面上，将可以看到如图14.14所示的桌面控件。

图14.14 带数据集的桌面控件

14.5 本章小结

Android系统的桌面是用户每天接触最多的界面，如果用户把我们开发的应用添加到系统桌面上，则可以大大增加用户对该软件的依赖。本章主要介绍了如何管理Android手机桌面，包括开发动态壁纸、管理手机桌面上的快捷方式、管理桌面小控件、显示带数据集的桌面小控件等，这些内容是读者需要重点掌握的。

第15章 传感器应用开发

本章要点

手机传感器的概念和作用

定义监听器监听传感器的数据

加速度传感器

方向传感器

陀螺仪传感器

磁场传感器

重力传感器

线性加速度传感器

温度传感器

光传感器

压力传感器

心率传感器

使用方向传感器开发指南针

使用方向传感器开发水平仪

Android系统提供了对传感器的支持，如果手机设备的硬件提供了这些传感器，Android应用可以通过传感器来获取设备的外界条件，包括手

机设备的运行状态、当前摆放方向、外界的磁场、温度和压力等。Android系统提供了驱动程序去管理这些传感器硬件，当传感器硬件感知到外部环境发生改变时，Android系统负责管理这些传感器数据。

对于Android应用开发者来说，开发传感器应用十分简单，开发者只要为指定传感器注册一个监听器即可，当外部环境发生改变时，Android系统会通过传感器获取外部环境的数据，并将数据传给监听器的监听方法。

通过在Android应用中添加传感器，可以充分激发开发者、用户的想象力，可以开发出各种新奇的程序，比如电子罗盘、水平仪等。除此之外，还可以利用传感器开发各种游戏，必须通过传感器来感知用户动作，从而在游戏中提供对应的响应。

15.1 利用Android的传感器

在Android系统中开发传感器应用十分简单，因为Android系统为传感器支持提供了强大的管理服务。

开发传感器应用的步骤如下。

1 调用Context的getSystemService（Context.SENSOR_SERVICE）方法获取Sensor Manager对象，SensorManager对象代表系统的传感器管理服务。

2 调用SensorManager的getDefaultSensor（int type）方法来获取指定类型的传感器。

3 通常选择在Activity的onResume（）方法中调用SensorManager的registerListener（）方法为指定传感器注册监听器，程序通过实现监听器即可获取传感器传回来的数据。

SensorManager提供的注册传感器的方法为registerListener（SensorEventListener listener，Sensor sensor，int rate），该方法

中三个参数说明如下。

➢ **listener**：监听传感器事件的监听器。该监听器需要实现SensorEventListener接口。

➢ **sensor**：传感器对象。

➢ **rate**：指定获取传感器数据的频率。

该方法中rate可以获取传感器数据的频率，它支持如下几个频率值。

➢ **SensorManager.SENSOR_DELAY_FASTEST**：最快。延迟最小，只有特别依赖于传感器数据的应用推荐采用这种频率，该种模式可能造成手机电量大量消耗。由于传递的为原始数据，算法处理不好将会影响应用的性能。

➢ **SensorManager.SENSOR_DELAY_GAME**：适合游戏的频率。一般有实时性要求的应用适合使用这种频率。

➢ **SensorManager.SENSOR_DELAY_NORMAL**：正常频率。一般对实时性要求不是特别高的应用适合使用这种频率。

➢ **SensorManager.SENSOR_DELAY_UI**：适合普通用户界面的频率。这种模式比较省电，而且系统开销也很小，但延迟较大，因此只适合在普通小程序中使用。

下面将会按照上面的步骤来开发一个加速度传感器应用。该程序的界面很简单，提供一个文本框来显示加速度值即可，此处不再给出界面布局代码。该应用的Activity代码如下。

程序清单：
codes\15\15.1\AccelerometerTest\app\src\main\java\org\crazyit\sensor\MainActivity.java

上面程序中的①号粗体字代码用于获取系统的传感器管理服务，②号粗体字代码则用于获取加速度传感器，并为该传感器注册监听器。本程序直接使用了Activity充当传感器监听器，因此该Activity实现了SensorEventListener接口，并实现了该接口中的两个方法。

➢ **onSensorChanged（）：** 当传感器的值发生改变时触发该方法。

➢ **onAccuracyChanged（）：** 当传感器的精度发生改变时触发该方法。

上面的程序实现onSensorChanged（）方法时通过SensorEvent对象的values（）方法来获取传感器的值，不同传感器所返回的值的个数是不等的。对于加速度传感器来说，它将返回三个值，分别代表手机设备在X、Y、Z三个方向上的加速度。

需要指出的是，传感器的坐标系统与屏幕坐标系统不同，传感器坐标系统的X轴沿屏幕向右；Y轴则沿屏幕向上，Z轴则垂直于屏幕向外。图15.1显示了传感器的坐标系统。

从图15.1所示的坐标系统大致可以看出，当拿着手机横向左右移动时，可能产生X轴上的加速度；拿着手机前后移动时，可能产生Z轴上的加速度；当拿着手机竖向上下移动时，可能产生Y轴上的加速度。

运行上面的程序，将会看到如图15.2所示的输出。

图15.1 传感器的坐标系统

图15.2 加速度传感器

从图15.2可以看出，当用户拿着手机“晃动”时，即可看到程序能检测到传感器返回的加速度值。

Android模拟器默认并没有提供传感器支持，因此本章的所有示例程序都应使用真机进行测试，才可看到正确的运行效果。

15.2 Android的常用传感器

上一节介绍了如何监听Android设备的加速度，只要通过SensorManager为加速度传感器注册监听器即可。实际上Android系统对所有类型的传感器的处理都完全一样，只是传感器的类型有所区别而已。

15.2.1 方向传感器

方向传感器用于感应手机设备的摆放状态。方向传感器可以返回三个角度，这三个角度即可确定手机的摆放状态。

关于方向传感器返回的三个角度说明如下。

➢ **第一个角度**：表示手机顶部朝向与正北方的夹角。当手机绕着Z轴旋转时，该角度值发生改变。例如当该角度为0度时，表明手机顶部朝向正北；该角度为90度时，代表手机顶部朝向正东；该角度为180度时，代表手机顶部朝向正南；该角度为270度时，代表手机顶部朝向正西。

➢ **第二个角度**：表示手机顶部或尾部翘起的角度。当手机绕着X轴倾斜时，该角度值发生变化，该角度的取值范围是-180～180度。假设将手机屏幕朝上水平放在桌子上，如果桌子是完全水平的，该角度值应该是0度。假如从手机顶部开始抬起，直到将手机沿X轴旋转180度（屏幕向下水平放在桌面上），在这个旋转过程中，该角度值会从0度变化到-180 度。也就是说，从手机顶部抬起时，该角度的值会逐渐减小，直到等于-180度；如果从手机底部开始抬起，直到将手机沿X轴旋转180度（屏幕向下水平放在桌面上），该角度的值会从0度变化

到180度。也就是说，从手机底部抬起时，该角度的值会逐渐增大，直到等于180度。

➢ **第三个角度**：表示手机左侧或右侧翘起的角度。当手机绕着Y轴倾斜时，该角度值发生变化。该角度的取值范围是-90～90度。假设将手机屏幕朝上水平放在桌面上，如果桌面是完全水平的，该角度值应为 0 度。假如将手机左侧逐渐抬起，直到将手机沿 Y轴旋转90度（手机与桌面垂直），在这个旋转过程中，该角度值会从0度变化到-90度。也就是说，从手机左侧抬起时，该角度的值会逐渐减小，直到等于-90度；如果从手机右侧开始抬起，直到将手机沿Y轴旋转90度（手机与桌面垂直），该角度的值会从0度变化到90度。也就是说，从手机右侧抬起时，该角度的值会逐渐增大，直到等于90度。

通过在应用程序中使用方向传感器，应用程序就可检测到手机设备的摆放状态，比如手机顶部的朝向，手机目前的倾斜角度等。借助于方向传感器，可以开发出指南针、水平仪等有趣的应用。

此处不单独介绍方向传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.2 陀螺仪传感器

陀螺仪传感器用于感应手机设备的旋转速度。陀螺仪传感器可以返回设备绕X、Y、Z这三个坐标轴（陀螺仪传感器的坐标系统与加速度传感器的坐标系统完全相同）的旋转速度。旋转速度的单位是弧度/秒，旋转速度为正值代表逆时针旋转，负值代表顺时针旋转。

关于陀螺仪传感器返回的三个角速度说明如下。

➢ **第1个值**：代表该设备绕X轴旋转的角速度。

➢ **第2个值**：代表该设备绕Y轴旋转的角速度。

➢ **第3个值**：代表该设备绕Z轴旋转的角速度。

此处不单独介绍陀螺仪传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.3 磁场传感器

磁场传感器主要用于读取手机设备外部的磁场强度。即使周围没有任何直接的磁场，手机设备也始终会处于地球磁场中。随着手机设备摆放状态的改变，周围磁场在手机的X、Y、Z方向上的影响会发生改变。

磁场传感器会返回三个数据，分别代表周围磁场分解到X、Y、Z三个方向上的磁场分量，磁场数据的单位是微特斯拉（μT）。

此处不单独介绍磁场传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.4 重力传感器

重力传感器会返回一个三维向量，这个三维向量可显示重力的方向和强度。重力传感器的坐标系统与加速度传感器的坐标系统是相同的。

注意：

当设备处于静止状态时，重力传感器的输出与加速度传感器的输出应该是相同的。

15.2.5 线性加速度传感器

线性加速度传感器返回一个三维向量显示设备在各方向上的加速度（不包含重力加速度）。线性加速度传感器的坐标系统与加速度传感器的坐标系统相同。

线性加速度传感器、重力传感器、加速度传感器这三者输出值的关系如下：

15.2.6 温度传感器

温度传感器用于获取手机设备所处环境的温度。温度传感器会返回一个数据，代表手机设备周围的温度，单位是摄氏度。

此处不单独介绍温度传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.7 光传感器

光传感器用于获取手机设备所处环境的光的强度。光传感器会返回一个数据，代表手机设备周围的光的强度，该数据的单位是勒克斯（lux）。

此处不单独介绍光传感器的应用，后面会通过一个示例来集中介绍所有传感器的用法。

15.2.8 压力传感器

压力传感器用于获取手机设备所处环境的压力的大小。压力传感器会返回一个数据，代表手机设备周围的压力的大小。

此处不单独介绍压力传感器的应用，后面将通过一个示例来集中介绍所有传感器的用法。

正如前面所介绍的，Android系统对所有传感器的处理方式完全相同，接下来通过一个示例程序来介绍上面这些传感器的用法。该程序界面只是提供了几个文本框，分别用于显示不同的传感器数据。该程序代码如下。

程序清单：
codes\15\15.2\SensorTest\app\src\main\java\org\crazyit\sensor\MainActivity.java

上面的程序一样遵守前面介绍的传感器编程步骤：①号粗体字代码在Activity的onCreate（）方法中获取SensorManager对象。程序中大段的粗体字代码在Activity的onResume（）方法中为指定类型的传感器注册监听器，本程序为8种类型的传感器注册了监听器；实现onSensorChanged（SensorEvent event）方法就实现了传感器监听器，实现监听器方法时即可获取传感器所传回来的数据。

在真机中调试该程序，运行该程序，即可看到如图15.3所示的结果。

从图15.3可以看出，该程序并不能获取温度传感器和压力传感器的值，这是因为笔者的手机不支持温度传感器与压力传感器的缘故。

图15.3 传感器应用

15.2.9 Android 5.0新增的心率传感器

心率传感器是Android 5.0新增的一种传感器，这种传感器可以返回佩戴该设备的人每分钟的心跳次数。该传感器返回的数据的准确性可通过SensorEvent的accuracy进行判断，如果该属性值为SENSOR_STATUS_UNRELIABLE或SENSOR_STATUS_NO_CONTACT，则表明传感器返回的心率值是不太可靠的，应该被丢弃。

注意：

这种传感器需要android.permission.BODY_SENSORS权限，如果该程序没有获取这种权限，SensorManager.getSensorsList（）、SensorManager.getDefaultSensor（）都不能返回这种传感器。

15.3 传感器应用案例

对传感器的支持是Android系统的特性之一，通过使用传感器可以轻易开发出各种有趣的应用。下面将会通过使用方向传感器来开发指南针和水平仪两个有趣的应用。

实例：指南针

前面介绍方向传感器时已经指出，方向传感器传回来的第一个参数值就是代表手机绕Z轴转过的角度，也就是手机顶部与正北的夹角，通过在程序中检测该夹角就可以开发出指南针。

开发指南针的思路很简单：先准备一张指南针图片，该图片上方向指针指向北方。接下来开发一个检测方向的传感器，程序检测到手机顶部绕Z轴转过多少度，让指南针图片反向转过多少度即可。由此可见，指南针应用只要在界面中添加一张图片，并让图片总是反向转过方向传感器返回的第一个角度值即可。下面是该程序的代码。

程序清单：
codes\15\15.3\Compass\app\src\main\java\org\crazyit\compass\MainActivity.java

指南针程序的关键代码就是程序中的粗体字代码，该程序检测到手机绕Z轴转过的角度，也就是手机Y轴与正北方向的夹角，然后让指南针图片反向转过相应的角度即可。

在真机中调试该程序，运行该程序，即可看到如图15.4所示的结果。

实例：水平仪

这里介绍的水平仪就是那种比较传统的水平仪，在一个透明的圆盘中充满某种液体，液体中留有一个气泡，当一端翘起时，该气泡将会浮向翘起的一端。

图15.4 指南针

前面介绍方向传感器时已经指出，方向传感器会返回三个角度值，其中第二个角度值代表底部翘起的角度（当顶部翘起时为负值）；第三个角度值代表右侧翘起的角度（当左侧翘起时为负值）；根据这两个角度值就可开发出水平仪了。

假设我们以大透明圆盘的中心为原点，当手机顶部翘起时，气泡应该向顶部移动，也就是气泡位置的Y坐标（2D绘图坐标系，屏幕左上角为原点）应减小；当手机底部翘起时，气泡应该向底部移动，也就是气泡位置的Y坐标应增大—假设气泡开始位于大透明圆盘的中心，气泡的 Y 坐标的改变正好与方向传感器返回的第二个参数代表的角度的正负相符，因此根据方向传感器返回的第二个参数来计算气泡的 Y 坐标即可；与此类似，当手机左侧翘起时，气泡应该向左侧移动，也就是气泡位置的X坐标（2D绘图坐标系，屏幕左上角为原点）应减小；当手机右侧翘起时，气泡应该向右侧移动，也就是气泡位置的 X 坐标应增大—假设气泡开始位于大透明圆盘的中心，气泡的 X 坐标的改变正好与方向传感器返回的第三个参数代表的角度的正负相符，因此根据方向传感器返回的第三个参数来计算气泡的X坐标即可。

通过上面介绍的方式来动态改变程序界面中气泡的位置—手机哪端翘起，水平仪中的气泡就浮向哪端，这就是水平仪的实现思想。

该程序用了一个自定义View，该自定义View很简单，就是绘制透明圆盘和气泡—其中气泡的位置会动态改变。该自定义View的代码如下。

程序清单：
codes\15\15.3\Gradienter\app\src\main\java\org\crazyit\sensor\MyView.java

正如上面程序中的粗体字代码所示，该自定义View会根据bubbleX、bubbleY动态地绘制气泡的位置，而这个bubbleX、bubbleY就需要根据方向传感器返回的第三个角度、第二个角度来动态计算。

下面是该程序中Activity的代码。

程序清单：
codes\15\15.3\Gradienter\app\src\main\java\org\crazyit\sensor\MainActivity.java

该程序的关键代码就是程序中粗体字代码部分，这些代码实现了前面介绍的设计：程序检测方向传感器返回的第二个、第三个角度值，并根据第二个角度值来计算气泡的Y坐标，根据第三个角度值来计算X坐标；计算完成后通知系统重绘MyView组件即可。

如果需要在模拟器中调试该程序，请先启动SensorSimulator的PC端，并让SensorSimulator的手机端程序与PC端程序建立连接。然后运行该程序，即可看到如图15.5所示的结果。

图15.5 水平仪

从图15.5可以看到水平仪中间的气泡，通过该气泡所在的位置即可大致确定手机底下的支撑是否水平。当气泡位于仪表盘中的红色“╋”标识处时，即可认为手机底下的支撑完全水平。

15.4 本章小结

Android系统的特色之一就是支持传感器，通过传感器可以获取手机设备运行的外界信息，包括手机运动的加速度、手机摆放方向等。学习本章需要重点掌握Android传感器支持的API，包括如何通过SensorManager注册传感器监听器，如何使用SensorEventListener监听传感器数据等。除此之外，读者还需要掌握Android的加速度传感器、方向传感器、陀螺仪传感器、磁场传感器、重力传感器、线性加速度传感器、温度传感器、光传感器、压力传感器和心率传感器等常见传感器的功能与用法。

第16章 GPS应用开发

本章要点

GPS的概念和用途

Android提供的GPS支持

LocationManager和LocationProvider

获取系统的LocationProvider列表

根据名字获取指定的LocationProvider

根据Criteria获取满足条件的LocationProvider

通过模拟器发送GPS定位信息

通过LocationListener监听GPS定位信息

临近警告支持

GPS 是英文Global Positioning System（全球定位系统）的简称，GPS是20世纪70年代由美国陆海空三军联合研制的新一代空间卫星导航定位系统。从这个介绍不难发现，GPS的作用就是为全球的物体提供定位功能。

GPS定位系统由三部分组成，即GPS卫星组成的空间部分、若干地球站组成的控制部分和普通用户手中的接收机这三个部分。对于手机用户来说，手机就是GPS定位系统的接收机，也就是说，GPS定位需要手机的硬件支持GPS功能。

GPS的空间部分由GPS卫星组成，覆盖于全球上空的GPS卫星星座，必须保证在各处能时时观测到高度角为15°以上的4颗卫星，这样才能保

证GPS系统的准确定位。目前，GPS卫星星座共有24颗GPS卫星，均匀分布在倾角为55°的6个轨道上，保证在地面的任意一点都可同时观测到24颗卫星。

GPS定位系统听上去专业、高深，实际上也是一门高新技术，但对于Android应用开发的程序员来说，开发提供GPS功能的应用程序十分简单。就像Android为电话管理支持提供了TelephonyManager类、为音频管理支持提供了AudioManager类，Android为支持GPS则提供了LocationManager类，通过LocationManager类及其他几个辅助类，开发人员可以非常方便地开发出GPS应用。

16.1 支持GPS的核心API

Android为GPS功能支持专门提供了一个LocationManager类，它的作用与TelephonyManager、AudioManager等服务类的作用相似，所有GPS定位相关的服务、对象都将由该对象来产生。

与程序中获取TelephonyManager、AudioManager的方法相似，程序并不能直接创建LocationManager的实例，而是通过调用Context的getSystemService（）方法来获取，例如如下代码：

一旦在程序中获得了LocationManager对象之后，接下来即可调用LocationManager的方法来获取GPS定位的相关服务和对象了。LocationManager提供了如下常用的方法。

➢ **boolean addGpsStatusListener（GpsStatus.Listener listener）**：添加一个监听GPS状态的监听器。

➢ **void addProximityAlert（double latitude，double longitude，float radius，long expiration，PendingIntent intent）**：添加一个临近警告。

➢ **List<String>getAllProviders（）**：获取所有的LocationProvider列表。

➢ **String getBestProvider（Criteria criteria，boolean enabledOnly）**：根据指定条件返回最优的LocationProvider对象。

➢ **GpsStatus getGpsStatus（GpsStatus status）**：获取GPS状态。

➢ **Location getLastKnownLocation（String provider）**：根据LocationProvider获取最近一次已知的Location。

➢ **LocationProvider getProvider（String name）**：根据名称来获取LocationProvider。

➢ **List<String>getProviders（Criteria criteria，boolean enabledOnly）**：根据指定条件获取满足该条件的全部LocationProvider的名称。

➢ **List<String>getProviders（boolean enabledOnly）**：获取所有可用的LocationProvider。➢ **boolean isProviderEnabled（String provider）**：判断指定名称的LocationProvider是否可用。

➢ **void removeGpsStatusListener（GpsStatus.Listener listener）**：删除GPS状态监听器。

➢ **void removeProximityAlert（PendingIntent intent）**：删除一个临近警告。

➢ **void requestLocationUpdates（String provider，long minTime，float minDistance，PendingIntent intent）**：通过指定的 LocationProvider 周期性地获取定位信息，并通过intent启动相应的组件。

➢ **void requestLocationUpdates（String provider，long minTime，float minDistance，LocationListener listener）：** 通过指定的LocationProvider周期性地获取定位信息，并触发listener所对应的触发器。

在上面的方法列表中涉及GPS定位支持的另一个重要的API：LocationProvider（定位提供者），LocationProvider对象就是定位组件的抽象表示，通过LocationProvider可以获取该定位组件的相关信息。LocationProvider提供了如下常用方法。

➢ **int getAccuracy（）：** 返回该LocationProvider的精度。

➢ **String getName（）：** 返回该LocationProvider的名称。

➢ **int getPowerRequirement（）：** 获取该LocationProvider的电源需求。

➢ **boolean hasMonetaryCost（）：** 返回该LocationProvider是收费的还是免费的。

➢ **boolean meetsCriteria（Criteria criteria）：** 判断该LocationProvider是否满足Criteria条件。

➢ **boolean requiresCell（）：** 判断该LocationProvider是否需要访问网络基站。

➢ **boolean requiresNetwork（）：** 判断该LocationProvider是否需要网络数据。

➢ **boolean requiresSatellite（）：** 判断该LocationProvider是否需要访问基于卫星的定位系统。

➢ **boolean supportsAltitude（）：** 判断该LocationProvider是否支持高度信息。

➢ **boolean supportsBearing（）：** 判断该LocationProvider是否支持方向信息。

➢ **boolean supportsSpeed（）：** 判断该LocationProvider是否支持速度信息。

除此之外，GPS定位支持还有一个API：Location，它就是一个代表位置信息的抽象类，提供了如下方法来获取定位信息。

➢ **float getAccuracy（）：** 获取定位信息的精度。

➢ **double getAltitude（）：** 获取定位信息的高度。

➢ **float getBearing（）：** 获取定位信息的方向。

➢ **double getLatitude（）：** 获取定位信息的纬度。

➢ **double getLongitude（）：** 获取定位信息的经度。

➢ **String getProvider（）：** 获取提供该定位信息的LocationProvider。

➢ **float getSpeed（）：** 获取定位信息的速度。

➢ **boolean hasAccuracy（）：** 判断该定位信息是否有精度信息。

➢ **boolean hasAltitude（）：** 判断该定位信息是否有高度信息。

➢ **boolean hasBearing（）：** 判断该定位信息是否有方向信息。

➢ **boolean hasSpeed（）：** 判断该定位信息是否有速度信息。

上面三个API就是Android GPS定位支持的三个核心API，使用它们来获取GPS定位信息的通用步骤如下。

1 获取系统的LocationManager对象。

2 使用LocationManager，通过指定LocationProvider来获取定位信息，定位信息由Location对象来表示。

3 从Location对象中获取定位信息。

下面将会使用这三个核心API进行系统定位。

16.2 获取LocationProvider

通过前面的介绍可以看出，Android的定位信息由LocationProvider对象来提供，该对象代表一个抽象的定位组件。在开始编程之前，需要先获得LocationProvider对象。

16.2.1 获取所有可用的LocationProvider

LocationManager提供了一个getAllProviders（）方法来获取系统所有可用的LocationProvider，下面的示例程序将可以列出系统所有的LocationProvider。该程序界面很简单，界面中只提供一个ListView来显示所有的LocationProvider即可，故不再给出界面布局代码。该程序的代码如下。

程序清单：
codes\16\16.2\AllProvidersTest\src\org\crazyit\gps\MainActivity.java

图16.1 获取系统所有的LocationProvider

上面程序中的粗体字代码就可获取系统中所有LocationProvider的名称。运行上面的程序，可以看到如图16.1所示的输出。

从图16.1所示的运行结果可以看出，当前模拟器所有可用的LocationProvider有如下两个。

➢ **passive**：由LocationManager.PASSIVE_PROVIDER常量表示。

➢ **gps**：由LocationManager.GPS_PROVIDER常量表示，代表通过GPS获取定位信息的LocationProvider对象。

上面列出的LocationProvider中最常用的是LocationProviderGPS_PROVIDER。

提示：

除了上面列出的passive和gps两个LocationProvider之外，还有一个名为network的LocationProvider，由LocationManager.NETWORK_PROVIDER常量表示，代表通过移动通信网络获取定位信息的LocationProvider对象。

16.2.2 通过名称获得指定LocationProvider

程序调用LocationManager的getAllProviders（）方法获取所有LocationProvider时返回的是List<String>集合，集合元素为LocationProvider的名称，为了获取实际的LocationProvider对象，可借助于LocationManager的LocationProvider getProvider（String name）方法。

例如以下代码：

16.2.3 根据Criteria获得LocationProvider

前面的程序调用LocationManager的getAllProviders（）方法返回了系统所有可用的LocationProvider，但大部分时候，应用程序可能希望得到符合指定条件的LocationProvider，这就需要借助于LocationManager的getBestProvider（Criteria criteria，boolean enabledOnly）方法来获取了。

上面方法中的Criteria就代表了一个"过滤"条件，该方法将只返回符合该Criteria的LocationProvider。Criteria提供了如下常用的方法来设置条件。

➢ **setAccuracy（int accuracy）**：设置对LocationProvider的精度要求。

➢ **setAltitudeRequired（boolean altitudeRequired）**：设置要求LocationProvider能提供高度信息。

➢ **setBearingRequired（boolean bearingRequired）**：设置要求LocationProvider能提供方向信息。

➢ **setCostAllowed（boolean costAllowed）**：设置要求LocationProvider是否免费。

➢ **setPowerRequirement（int level）**：设置对LocationProvider耗电量的要求。

➢ **setSpeedRequired（boolean speedRequired）**：设置要求LocationProvider能提供速度信息。

下面的程序示范了如何获取系统中免费的LocationProvider，并且该LocationProvider必须能提供高度、速度信息等。

程序清单：
codes\16\16.2\FreeProvidersTest\src\org\crazyit\gps\MainActivity.java

上面程序中的粗体字代码创建了一个Criteria对象，并通过Criteria设置了LocationProvider必须满足的条件。运行该程序，即可列出所有符合Criteria条件的LocationProvider。

16.3 获取定位信息

获取了LocationManager对象之后，接下来就可通过指定LocationProvider来获取定位信息了。

16.3.1 通过模拟器发送GPS信息

Android模拟器本身并不能作为GPS接收机，因此无法得到GPS定位信息，但为了方便程序员测试GPS应用，Genymotion模拟器可以发送模拟的GPS定位信息。

启动Genymotion模拟器之后，打开Genymotion模拟器右边栏的GPS图标即可向模拟器发送GPS定位信息，如图16.2所示。

在图16.2所示的“GPS”面板中输入经度值、纬度值，即可向模拟器发送GPS定位信息。

16.3.2 获取定位数据

图16.2 向模拟器发送GPS定位信息

下面程序示范了如何通过手机实时地获取定位信息，包括用户所在的经度、纬度、高度、方向、移动速度等。该程序的界面很简单，只提供一个文本框来显示用户的定位信息即可，故此处不再给出程序界面代码。该程序代码如下。

程序清单：
codes\16\16.3\LocationTest\src\org\crazyit\gps\MainActivity.java

上面程序中的粗体字代码用于从Location中获取定位信息，包括用户的经度、纬度、高度、方向和移动速度等信息。程序中①号粗体字代码通过LocationManager设置了一个监听器，该监听器负责每隔3秒向LocationProvider请求一次定位信息，当LocationProvider可用时、不可用时或提供的定位信息发生改变时，系统会回调updateView（Location newLocation）来更新EditText中显示的定位信息。

该程序需要有访问GPS信号的权限，因此需要在AndroidManifest.xml文件中增加如下授权代码片段：

图16.3 实时获取定位信息

运行该程序，然后通过DDMS的Emulator Control面板来发送GPS定位信息，即可看到该程序显示如图16.3所示的输出。

由于该程序每隔3秒就会向GPS LocationProvider获取一次定位信息，这样上面的程序界面上总可以实时显示该用户的定位信息。

如果把该程序与Google Map结合，让该程序根据GPS提供的信息实时地显示用户在地图上的位置，即可开发出GPS导航系统。下一章会介绍相关内容。

16.4 临近警告

前面介绍LocationManager时已经提到，该API提供了一个addProximityAlert（double latitude，double longitude，float radius，long expiration，PendingIntent intent）方法，该方法用于添加一个临近警告。

所谓临近警告的示意图如图16.4所示。

也就是当用户手机不断地临近指定固定点，与该固定点的距离小于指定范围时，系统可以触发相应的处理。

图16.4 临近警告的示意图

添加临近警告的方法的参数说明如下。

➢ **latitude**：指定固定点的经度。

➢ **longitude**：指定固定点的纬度。

➢ **radius**：指定一个半径长度。

➢ **expiration**：指定经过多少毫秒后该临近警告就会过期失效。-1指定永不过期。

➢ **intent**：指定临近该固定点时触发该intent对应的组件。

下面的程序示范了如何检测手机是否进入“疯狂软件教育中心”附近，该程序几乎没有界面。当程序启动后，程序就会添加一个临近警告，当用户临近“疯狂软件教育中心”所在经度、纬度时，系统会显示提示。该程序代码如下。

程序清单：
codes\16\16.4\ProximityTest\src\org\crazyit\gps\MainActivity.java

上面程序中的粗体字代码用于添加临近警告，当用户手机临近指定经度、纬度确定的点时，系统会启动pi所对应的组件。pi对应的组件是一个BroadcastReceiver，它的代码如下。

程序清单：
codes\16\16.4\ProximityTest\src\org\crazyit\gps\ProximityAlertReciever.java

该BroadcastReceiver被激发后的处理非常简单，程序通过Intent传递过来的消息判断设备是进入指定区域还是离开指定区域，并根据不同的状态提示不同的信息。

运行该程序，并通过GPS模拟信息面板输入疯狂软件教育中心的经度、纬度（113.39、23.13），即可看到如图16.5所示的提示。

如果接下来在GPS模拟信息面板中输入其他的经度、纬度，就意味着该设备离开了该区域，于是可以看到如图16.6所示的提示。

图16.5 进入区域的提示

图16.6 离开区域的提示

16.5 本章小结

本章主要介绍了Android提供的GPS定位支持，目前的绝大部分Android手机都提供了GPS硬件支持，都可以作为GPS定位系统的接收机，而开发者要做的就是从Android系统中获取GPS定位信息。学习本章的重点是掌握LocationManager、LocationProvider与LocationListener等API的功能和用法，并可以通过它们来监听、获取GPS定位信息。

一旦在应用程序中获取了GPS定位信息之后，接下来就可以通过这种定位信息在Google Map上进行定位、跟踪等，这就需要结合下一章所介绍的Map服务了。

第17章 整合高德Map服务

本章要点

了解Map服务

掌握调用第三方Map服务的方法

获取第三方Map服务的API Key

添加第三方Map服务的SDK

根据GPS信息在地图上定位

根据GPS信息在地图上跟踪用户轨迹

调用地址解析服务

根据地址在地图上定位

调用第三方的导航服务

上一章介绍了如何使用Android的GPS来获取设备的定位信息，但这种方式得到的定位信息只不过是一些数字的经度、纬度值，如果这些经度、纬度值不能以更形象、直观的方式显示出来，对于大部分普通用户而言，这些经度、纬度数据几乎没有任何价值。

为了让上一章介绍的GPS信息“派上”用场，本章将会详细介绍Android调用第三方的 Map服务，本书将以高德Map服务为例来介绍如何在Android应用中嵌入高德地图。由于众所周知的某些原因，现在Google的各种服务基本上都已经被封锁了，因此本书删除了第2版中调用Google地图服务的相关知识，改为讲授在Android应用中调用第三方地

图服务。如果把上一章获得的GPS信息与本章的Map应用结合起来，就可以非常方便地开发出定位、导航等应用程序。

提示：

如果读者依然需要学习关于调用Google地图服务的相关知识，请参考本书第2版。

17.1 调用高德Map服务

国内比较常用的地图服务有高德地图服务（相对比较专业，iOS设备也采用这种地图）、百度地图服务等，不管使用哪一家的地图服务，大致的调用步骤都基本相似，读者只要熟练掌握其中一种即可。

17.1.1 获取Map API Key

为了在应用程序中调用第三方Map服务，必须先获取第三方Map服务的API Key。此处以获取高德地图服务的API Key为例进行介绍，获取步骤如下。

1 首先找到该App的数字证书的keystore的存储路径，此时可分为两种情况。

➢ 如果是作为实际产品发布的签名App，该App的数字证书的keystore将保存在用户自定义的路径中，该自定义路径也就是第1章中图1.42创建数字证书时所指定的存储路径。

➢ 如果是作为调试阶段的调试 App，该 App 的数字证书的 keystore 通常保存在ANDROID_SDK_HOME环境变量对应的路径的.android/目录下，在该目录下可以找到一个debug.keystore文件，该文件就是调试App的数字证书的keystore的存储路径。比如笔者的ANDROID_SDK_HOME环境变量值为D：\AVD，那么对应的keystore的存储路径为D：\AVD\.android\debug.keystore。

2 为了获取第三方Map服务的API Key，需要先用JDK提供的keytool工具查看keystore的认证指纹。启动命令行窗口，输入如下命令：

将上面命令中的<Android keystore的存储位置>替换成App的数字证书的keystore的存储位置（如果发布Android应用，则应该使用本公司的keystore的存储路径）。

运行上面的命令，系统将会提示“输入keystore密码”，输入Android模拟器的keystore的默认密码：android，系统将会显示该keystore对应的认证指纹。图17.1显示了查看keystore的认证指纹的详细过程。

图17.1 查看调试App的数字证书的keystore的认证指纹

提示：

如果运行keytool工具时系统提示“找不到该命令”，则说明还未在PATH环境变量中增加%JAVA_HOME%/bin路径，其中%JAVA_HOME%代表JDK的安装路径—JDK的安装路径的bin子目录下应该包含java.exe、javac.exe和keytool.exe工具。

3 记住图17.1所显示的SHA1对应的认证指纹，登录http：//id.amap.com/站点，系统显示如图17.2所示的登录高德地图的页面。

图17.2 登录高德地图

4 如果读者已有高德地图的用户名、密码，那么输入用户名、密码登录即可。如果读者还没有高德地图的用户名、密码，则需要通过该页面右边的“立即免费注册”链接注册一个新的账户。

5 注册完成后系统会提示用户还没有“成为开发者”，读者可通过页面中间的链接申请“成为开发者”，在申请成为开发者时，页面会提示用户输入邮箱和手机号码（请输入真实的邮箱和手机号码，因为高德会采用短信+邮箱的方式进行验证），提交请求，并通过邮件激活之后，此时就拥有了一个高德的开发者账号。

6 登录http：//lbs.amap.com/console/页面，即可看到如图17.3所示的页面。

图17.3 申请Key

提示：

如果读者第一次注册、登录高德地图的网站，可能看不到上面列出的maptest那条记录，这条记录是本书为测试Map应用所获取的Key。

7 单击“获取KEY”按钮，将显示如图17.4所示的输入页面。

图17.4 填写申请Key的信息

8 在文本框中输入keytool工具查看到的SHA1指纹，单击“获取KEY”按钮，系统返回如图17.3所示的页面，即可在该页面中看到该应用（包名+SHA1签名即可唯一地确定一个应用）对应的API Key。

17.1.2 高德地图入门

一旦为指定应用申请了API Key之后，接下来即可非常简单地在应用中使用高德地图了。高德地图提供了一个MapView组件，这个MapView继承了FrameLayout，因此它的本质就是一个普通的容器控件，所以开发者可以直接将该MapView添加到应用界面上。

MapView只是一个容器，真正为MapView提供地图支持的是AMap类，MapView可通过getMap（）方法来获取它所封装的AMap对象，AMap对象则提供了大量的方法来控制地图。

注意：

前面在申请API Key时填写的应用包名是什么，那么此处新建项目的包名也必须与之相同。

为了给Android应用添加高德地图支持，必须为该应用添加高德地图的SDK。添加高德地图的SDK请按如下步骤进行。

1 登录http：//lbs.amap.com/api/android-sdk/down/站点，即可看到如图17.5所示的页面。

图17.5 下载高德地图的SDK

2 由于本书将以高德3D地图为例，因此读者应该下载图17.5所示列表中的第1项（3D地图）和第 3 项（支持地址解析和反向地址解析等功能）。下载完成后得到Android_3DMap_V2.4.0.jar_.zip和AMap_Services_V2.3.1.jar_.zip两个压缩包。

3 将上面两个压缩包解压，解压之后得到两个JAR包，以及armeabi、armeabi-v7a、x86这三个文件夹。

4 将第3步解压得到的两个JAR包复制到Android应用的app/libs/目录下，再将Android Studio左边面板切换成Project面板，然后选中这两个JAR包并单击右键，通过右键菜单中的“Add As Library...”菜单项将这两个JAR包添加到该应用中。

5 在Android应用的app/src/main/目录下新建一个jniLibs子目录，并将第3步解压得到的armeabi、armeabi-v7a、x86这三个文件夹复制到该目录下。

提示：

实际上第4步是通过Android Studio为App添加第三方JAR包的方法；第5步则是通过Android Studio为App添加第三方*.so包（*.so文件似乎是Linux平台的动态链接库文件，类似于Windows平台的*.dll文件）的方法。如果读者使用的是Eclipse+ADT开发平台，那就更简单了，直接将两个JAR包和armeabi、armeabi-v7a、x86这三个文件夹复制到Android应用的libs目录下即可，Eclipse会自动加载它们。

6 接下来打开Android应用的AndroidManifest.xml文件，在该文件的<application.../>元素内添加如下<meta-data.../>子元素：

千万不要直接照着书上输入，因为上面<meta-data.../>元素用于启用高德地图支持，而其中的android：value属性值应该填写前面申请得到的API Key。

7 在Android应用的AndroidManifest.xml文件中添加如下权限：

经过上面步骤，高德地图SDK添加完成，剩下的事情就是使用MapView组件了，而MapView组件与普通Android组件的区别并不大。本应用的界面设计文件如下。

程序清单：
codes\17\17.1\AMapTest\app\src\main\res\main.xml

正如从上面粗体字代码所看到的，程序使用MapView的方式与使用普通Android View并没有太大的区别。需要指出的是，MapView要求在其所在的Activity的生命周期方法中回调该MapView的生命周期方法。下面是该应用的Activity代码。

程序清单：
codes\17\17.1\AMapTest\app\src\main\java\org\crazyit\map\MainActivity.java

正如从上面粗体字代码所看到的，程序只是简单地获取了界面上的MapView组件，并在该Activity的生命周期方法内回调了MapView的生命周期方法，整个应用就完成了。本应用为了简单示范AMap的功能，程序为界面上的ToggleButton添加了事件监听器，用于切换地图的显示方式。

运行该程序，切换到卫星地图，即可看到如图17.6所示的地图界面。

图17.6 地图

17.2 根据GPS信息在地图上定位

通过MapView获取AMap对象之后，接下来即可通过AMap来控制地图的显示形式和显示外观了。AMap提供了如下常用方法来控制地图。

➢ **setLocationSource（LocationSource locationSource）**：设置定位数据源。

➢ **setMapType（int type）**：设置地图显示的类型。

➢ **setMyLocationEnabled（boolean paramBoolean）**：设置定位层是否显示。

➢ **setMyLocationRotateAngle（float rotate）**：设置定位图片旋转的角度，从正北向开始，逆时针计算。

➢ **setMyLocationStyle（MyLocationStyle style）**：设置定位（当前位置）的绘制样式。

➢ **setMyLocationType（int type）**：设置定位的类型。该方法支持三个参数值，分别为LOCATION_TYPE_LOCATE，表示只在第一次定位时移动到地图中心点；LOCATION_TYPE_MAP_FOLLOW，表示总是定位、移动到地图中心点；LOCATION_TYPE_MAP_ROTATE，表示总是定位、移动到地图中心点，跟踪并根据方向旋转地图。

➢ **setPointToCenter（int x，int y）**：设置屏幕上的某个点为地图中心点。

➢ **setTrafficEnabled（boolean enabled）**：设置是否显示交通状况。

除此之外，AMap还提供了大量的setOnXxxListener（）方法，这些方法都需要传入相应的监听器实现类，通过这些监听器可以让地图响应用户操作。

如果用户希望向地图上添加自定义的图片或形状，AMap提供了如下常用方法。

➢ **addArc（ArcOptions options）**：在地图上添加一段扇形覆盖物（Arc）。

➢ **addCircle（CircleOptions options）**：在地图上添加圆形（Circle）覆盖物。

➢ **addGroundOverlay（GroundOverlayOptions options）**：在地图上添加图片。

➢ **addMarker（MarkerOptions options）**：在地图上添加标记（Marker）。

➢ **addMarkers（java.util.ArrayList<MarkerOptions>list，boolean moveToCenter）**：在地图上添加多个标记，并设置是否移动到屏幕中间。

➢ **Polygon addPolygon（PolygonOptions options）**：在地图上添加一个多边形（Polygon）。

➢ **Polyline addPolyline（PolylineOptions options）**：在地图上添加多条线段（Polyline）。

➢ addTileOverlay（TileOverlayOptions options）：在地图上添加tile overlay图层。

不管程序需要向地图上添加哪种东西，程序的操作步骤都大致相同，都可按如下步骤进行。

1 创建一个XxxOptions，比如要添加Marker对象，程序就需要先创建一个MarkerOptions；比如添加Polyline，就需要创建PolylineOptions。

2 调用XxxOptions的各种setter方法来设置属性。

3 调用AMap对象的addXxx（）方法添加即可。

提示：

高德地图的API文档本来就是用中文写成的，因此读者可以直接通过高德官网来了解AMap的各方法的功能和用法。

为了表示Map上的指定点，高德地图提供了LatLng类。LatLng十分简单，它就是对纬度、经度的封装。除此之外，高德地图还提供了一个CameraUpdate类（该类并未提供构造器，程序应该通过CameraUpdateFactory来创建该类的实例），CameraUpdate可控制地图的缩放级别、定位、倾斜角度等信息，程序调用 AMap 的moveCamera（CameraUpdate u pdate）方法根据CameraUpdate对地图进行缩放、定位和倾斜。

掌握了高德地图的上面常用的API之后，接下来就可以开发Android的Map应用了。

下面的程序示范了如何根据经度、纬度在地图上定位。该程序的界面提供了文本框让用户输入经度、纬度，也允许用户设置通过GPS信号在地图上定位。该程序的界面布局代码如下。

程序清单：
codes\17\17.2\LocationMap\app\src\main\res\layout\main.xml

提供了上面介绍的布局之后，接下来就可以在程序中根据用户输入的经度、纬度来进行定位，也可以根据GPS传入的信号进行定位。根据经度、纬度在AMap上定位的步骤如下。

1 根据程序获取的经度、纬度值创建LatLng对象。

2 调用CameraUpdateFactory的changeLatLng（）方法创建改变地图中心的Camera对象。

3 调用AMap的moveCamera（CameraUpdate update）即可控制地图定位到指定位置。该示例程序的代码如下。

程序清单：
codes\17\17.2\LocationMap\app\src\main\java\org\crazyit\map\MainActivity.java

上面程序中的①、②、③号粗体字代码分别用于创建按LatLng对象，并根据LatLng对象创建改变地图中心点的CameraUpdate对象，将地图定位到指定位置点。

程序中第二段粗体字代码示范了如何向地图上添加MarkerOptions—实际上是向地图上添加一个Marker。

运行上面的程序，将看到如图17.7所示的结果。

如果读者开发该程序时一切正常，将可以看到如图17.7所示的地图，在应用界面上方的文本框中输入合适的经度、纬度，地图将会定位到指定的位置。

图17.7所示的地图界面上有3个Marker，其中最下面一个Marker会不停地闪烁，这是因为程序给最下面一个Marker设置了3个不同颜色图标。

如果用户选中界面上的“GPS定位”单选钮，程序将会使用LocationManager不断地获取GPS信号，并根据GPS信号来调整地图中心，而且程序还在地图定位点添加了一个汽车图标。运行程序可看到如图17.8所示的界面。

图17.7 根据经纬度定位

图17.8 根据GPS定位

17.3 执行定位

前面介绍的在地图上定位的例子需要用户输入定位点的经度、纬度才能进行定位，这种方式显然不太现实：普通用户不太可能记住某个位

置的经度、纬度。对于普通用户来说，根据地址进行定位才是更有实用价值的地图。本节将会介绍如何根据地址在地图上定位。

17.3.1 地址解析与反向地址解析

通常来说，地图定位必须根据经度、纬度来完成，因此，如果需要让程序根据地址进行定位，则需要先把地址解析成经度、纬度。这里涉及如下两个基本概念。

➢ **地址解析**：把普通用户能看到的字符串地址转换为经度、纬度。

➢ **反向地址解析**：把经度、纬度值转换成普通的字符串地址。

高德地图搜索服务为地址解析提供了GeocodeSearch工具类，该工具类提供了如下方法来进行地址解析和反向地址解析。

➢ **RegeocodeAddress getFromLocation（RegeocodeQuery query）**：根据给定的经纬度和最大结果数返回反向地址解析的结果。

➢ **getFromLocationAsyn（RegeocodeQuery query）**：该方法与前一个方法类似，只是该方法以异步方式进行。

➢ **java.util.List<GeocodeAddress> getFromLocationName（GeocodeQuery query）**：根据给定的地理名称、城市来返回地址解析的结果列表。

➢ **getFromLocationNameAsyn（GeocodeQuery query）**：该方法与前一个方法类似，只是该方法以异步方式进行。

虽然GeocodeSearch工具类提供了上面方法来进行地址解析和反向地址解析，但实际上这个类还是需要调用网络上高德的查询服务。很明显，Android平台不大可能把地球上的所有地名与经度、纬度之间的映射关系都存储在手机系统中。

由于Android平台要求所有的网络访问都不能直接放在UI线程中进行，因此Android应用进行地址解析或反向地址解析时都应该采用异步方式进行。当程序采用异步方式进行地址解析或反向地址解析时，程序还必须为GeocodeSearch设置一个监听器，当解析完成时，该监听器所包含的方法将会被触发。

借助于GeocodeSearch提供的地址解析、反向地址解析的功能，接下来只要按如下步骤操作即可完成地址解析和反向地址解析。

1 创建GeocodeSearch对象，并为该对象设置解析监听器。

2 如果要进行地址解析，程序先创建GeocodeQuery对象，该对象封装了要解析的地理名称、城市等信息；如果要进行反向地址解析，程序先创建LatLonPoint对象，该对象封装了经度、纬度信息。

3 调用GeocodeSearch对象的方法执行地址解析或反向地址解析。

下面的示例示范了如何使用GeocodeSearch来进行地址解析、反向地址解析。该应用程序的Activity代码如下。

程序清单：
codes\17\17.3\GeocoderTest\app\src\main\java\org\crazyit\map\MainActivity.java

上面程序中的①号粗体字代码为GeocodeSearch对象设置了一个解析监听器，当程序调用该对象的方法执行异步解析完成后，程序将会自动触发该监听器包含的方法。

程序中②号粗体字代码以异步方式执行地址解析，在②号代码之前先创建了一个GeocodeQuery对象，该对象包含要解析的地理名称、城市等信息。

程序中③号粗体字代码以异步方式执行反向地址解析，执行③号代码时传入了一个LatLonPoint对象，该对象就包装了要进行反向地址解析的经度、纬度。

运行上面的程序，在文本框中输入某个地名，单击“解析”按钮，将可看到如图17.9所示的结果。

在图17.9所示的输入经度、纬度文本框中输入经度、纬度值，然后单击“反向解析”按钮，即可看到程序有如图17.10所示的输出。

图17.9 地址解析

图17.10 反向地址解析

17.3.2 执行定位

下面的应用程序是对17.2节示例程序的改进，该应用程序不需要用户输入经度、纬度值，只要用户输入目标地址，程序就会调用GeocodeSearch对目标地址执行地址解析，将其转换为经度、纬度值，再控制地图定位到指定地址点即可。

该程序代码如下。

程序清单：
codes\17\17.3\AddrLocMap\app\src\main\java\org\crazyit\map\MainActivity.java

上面程序中的①号粗体字代码调用GeocodeSearch对象执行异步的地址解析，getFromLocationNameAsy（）方法解析完成后将会触发解析监听器的onGeocodeSearched（）方法。上面程序的重点就是重写监听器的这个方法，程序在该方法中完成了如下三件事情。

➢ 根据地址解析得到的经度、纬度创建一个CameraUpdate，并将地图中心改变到此处。

➢ 根据地址解析得到的经度、纬度添加一个GroundOverlay（自定义图片）。

➢ 根据地址解析得到的经度、纬度添加一个Circle （圆形区域）。

上面三件事情的后两件事情的本质是一样的，因此程序的添加步骤也大同小异。编译、运行该程序，并在程序界面上的文本框中输入某个地址，然后单击"定位"按钮，即可看到如图17.11所示的结果。

经过前面的介绍不难发现，在Android应用中整合第三方Map服务其实比较简单，通过为Android应用整合第三方Map服务，可以开发出功能更强大的Android应用，比如为 SNS 系统增加地图支持，即可让用户实时查询好友的位置等。总之，通过在Android应用中整合第三方Map服务功能，就给开发Android应用提供了更多的可能性。

图17.11 根据地址定位

17.4 GPS导航

高德查询服务还可让开发者查询、规划路线。高德查询服务提供了RouteSearch查询类，专门用于查询各种路线。该查询类支持的查询功能由如下方法实现：

➢ **BusRouteResult calculateBusRoute（BusRouteQuery query）**：根据指定参数来查询公交路线。

➢ **void calculateBusRouteAsyn（BusRouteQuery query）**：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。

➢ **DriveRouteResult calculateDriveRoute（DriveRouteQuery query）**：根据指定参数来查询驾车路线。

➢ **void calculateDriveRouteAsyn（DriveRouteQuery query）**：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。

➢ **WalkRouteResult calculateWalkRoute（WalkRouteQuery query）**：根据指定参数来查询步行路线。

➢ **void calculateWalkRouteAsyn（WalkRouteQuery query）**：该方法与前一个方法功能类似，只是该方法以异步方式执行查询。

与前面各种XxxSearch类相似的是，RouteSearch查询的数据同样来自网络，因此在Android应用中调用这些查询方法，需要使用异步方式进行查询。为了使用RouteSearch执行异步查询，程序还需要为RouteSearch指定一个查询监听器（实现了RouteSearch.OnRouteSearchListener接口的对象）。

在最简单的情况下，程序只要通过RouteSearch查询、规划路线，然后实时地显示用户的当前位置，并提示用户是否正在规划路线上行驶，这样就可以实现一个最简单的导航软件了。

1 创建RouteSearch对象，并为该对象设置对应的监听器。

2 根据要查询路线类型的不同，分别创建对应的XxxQuery对象（比如查询公交路线，需要创建BusRouteQuery；查询步行路线，创建WalkRouteQuery；查询驾车路线，创建DriveRouteQuery），这些

XxxQuery中封装了相应的路线信息，比如起点、终点、途经点、绕行区域等。

3 调用RouteSearch对象的方法执行查询。

当程序通过重写RouteSearch监听器的方法获取查询返回的路线之后，该规划路线的本质其实就是地图上的多条线段，因此程序只要通过AMap的addPolyline（）把这些规划路线绘制上去即可。

下面的示例示范了如何开发一个简单的GPS导航软件，该导航软件主要实现了两个功能：①通过RouteSearch查询、规划行车路线；②根据GPS信号显示用户的当前行驶位置。

该程序的界面设计文件比较简单，只是包含一个用于输入目的地的文本框和高德的MapView，此处不再给出界面设计文件。下面是该程序的Activity类代码。

程序清单：
codes\17\17.4\Navigation\app\src\main\java\org\crazyit\map\MainActivity.java

使用RouteSearch获取导航路线只要如下3步。

上面程序中的第一段粗体字代码分别创建了GeocodeSearch和RouteSearch对象，并为这两个对象设置了查询监听器。其中GeocodeSearch用于对用户输入的目的地执行地址解析；而RouteSearch则用于查询、规划行车路线。

当该Activity加载完成后，程序将会通过LocationManager的方法不间断地获取GPS信息，获取GPS信息之后，程序调用①号方法将用户的

实时位置绘制在地图上。

当用户按下应用界面上的“规划路线”按钮时，程序将会通过GeocodeSearch的getFromLocationNameAsyn（）方法执行异步地址解析，因此当解析完成后程序将会自动激发②号方法，该方法中先获取了地址解析所获得的经度、纬度，然后创建了一个DriveRouteQuery对象（代表驾车路线查询），最后调用RouteSearch的方法执行异步查询即可，如该方法中的粗体字代码所示。

但RouteSearch查询、规划路线完成时，程序将会自动激发③号方法，通过该方法的参数即可获取RouteSearch查询返回的多条规划路线。但此处为了简化程序，直接使用了第一条规划路线，然后程序通过AMap的addPolyline（）方法将规划路线底层封装的多条线段绘制在地图上，这样就可向用户显示规划路线了。

运行该程序，在“目标地”文本框中输入广州的某个地名，按下“规划路线”按钮即可看到如图17.12所示效果。

图17.12 GPS导航

注意：

运行该程序时一定要打开模拟器的GPS支持；否则程序将会因为无法获取GPS信号而出错。

17.5 本章小结

本章主要介绍了在Android系统中调用第三方Map的方法。学习本章需要掌握在Android应用中整合第三方各种服务的方法，包括如何获取第三方服务的API Key、添加第三方服务的SDK等。除此之外，读者需要重点掌握根据GPS信息在地图上定位、跟踪的方法。不仅如此，如果结合了Android的地址解析服务，Android应用还可以根据地址信息在

地图上定位，这也是读者需要掌握的内容。本章也介绍了高德查询服务提供的导航支持。

第18章 合金弹头

本章要点

开发射击类游戏的基本方法

游戏的界面分解和分析

游戏界面组件的分析和实现

怪物的移动和发射子弹

实现角色移动、跳跃、发射子弹等行为

检测子弹是否命中目标

实现游戏的绘图工具类

管理游戏资源

继承SurfaceView实现游戏主组件

掌握SurfaceView的绘图机制

使用多线程实现游戏动画

实现游戏的Activity

本章将会介绍一款经典的射击类游戏：合金弹头，合金弹头游戏要求玩家控制自己的角色不断前行，并发射子弹去射击沿途遇到的各种怪物，同时还要躲避怪物发射的子弹。当然，由于完整的合金弹头游戏涉及的地图场景很多，而且怪物种类也很多，因此本章对该游戏进行了适当的简化。本游戏只实现了一个地图场景，并将之设置为无限地图，并且只实现了3种类型的怪物，但只要读者真正掌握了本章的内

容，当然也就能从一个地图扩展为多个地图；也可以从3种类型的怪物扩展出多种类型的怪物了。

对于Android学习者来说，学习开发这个小程序难度适中，而且能很好地培养学习者的学习乐趣。开发者需要从程序员的角度来看待玩家面对的游戏界面，游戏界面上的每个怪物、每个角色、每颗子弹、每个能与玩家交互的东西，都应该在程序中通过类的形式来定义它们，这样才能更好地用面向对象的方式来解决问题。

18.1 合金弹头游戏简介

合金弹头是一款早期风靡一时的射击类游戏，这款游戏的节奏感非常强，让大部分男同胞充满童年回忆。实际上，现在也非常流行将一些早期游戏移植到手机平台上，以充分满足广大玩家的怀旧情怀。

图18.1显示了合金弹头的游戏界面。

这款游戏的玩法很简单，玩家控制角色不断地向右前进，角色可通过跳跃来躲避敌人（也可统称为怪物）发射的子弹和地上的炸弹，玩家也可控制角色发射子弹来打死右边的各种敌人。对于完整的合金弹头游戏，它会包含很多“关卡”，每个关卡都是一种地图，每个关卡都包含了大量不同的怪物。但本章由于篇幅关系，只做了一种地图，而且这种地图是“无限循环”的—也就是说，玩家只能一直向前去消灭不同的怪物，无法实现“通关”。

图18.1 合金弹头

提示：

如果读者有兴趣把这款游戏改成包含很多关卡的游戏，那就需要准备大量的背景图片。然后为不同的地图加载不同的背景图片，让地图不要无限循环即可，并为不同地图使用不同的怪物。

18.2 开发游戏界面组件

在开发游戏之前，首先需要从程序员的角度来分析游戏界面，并逐步实现游戏界面上的各种组件。

18.2.1 游戏界面分析

对于图18.1所示的游戏界面，从普通玩家的角度来看，他会看到游戏界面上有受玩家控制移动、跳跃、发射子弹的角色，还有不断发射子弹的敌人、地上有炸弹、天空中有正在爆炸的飞机……乍看上去会给人眼花缭乱的感觉。

如果从程序员的角度来看，游戏界面大致可分为如下组件。

➢ **游戏背景**：只是一张静止的图片。

➢ **角色**：该角色可以站立、走动、跳跃、射击。

➢ **怪物**：怪物类代表了游戏界面上所有的敌人，包括拿枪的敌人、地上的炸弹、天空中的飞机……虽然这些怪物的图片不同、发射的子弹不同，攻击力也可能不同，但这些只是实例与实例之间的差异，因此程序只要为怪物定义一个类即可。

➢ **子弹**：不管是角色发射的子弹还是怪物发射的子弹，都可归纳为子弹类。虽然不同子弹的图片不同，攻击力不同，但这些只是实例与实例之间的差异，因此程序只要为子弹定义一个类即可。

从上面介绍不难看出，开发这款游戏，主要就是实现上面的角色、怪物和子弹3个类。

18.2.2 实现“怪物”类

由于不同怪物之间会存在各种差异，那么此处就需要为怪物类定义相应的实例变量来记录这些差异。不同怪物之间可能存在如下差异。

➢ 怪物的类型。

➢ 代表怪物位置的X、Y坐标。

➢ 标识怪物是否已经死亡的旗标。

➢ 绘制怪物图片左上角的X、Y坐标。

➢ 绘制怪物图片右下角的X、Y坐标。

➢ 怪物发射的所有子弹。（有的怪物不会发射子弹）

➢ 怪物未死亡时所有的动画帧图片和怪物死亡时所有的动画帧图片。

提示：

本程序并未把怪物的所有动画帧图片直接保存在怪物实例中，本程序将会专门使用一个工具类来保存所有角色、怪物的所有动画帧图片。

为了让游戏界面的角色、怪物都能"动起来"，程序的实现思路是这样的：程序会专门启动一条独立的线程，这条线程负责控制角色、怪物不断地更换新的动画帧图片—因此程序需要为怪物增加一个成员变量来记录当前游戏界面正在绘制怪物动画的第几帧，而负责动画的独立线程只要不断地调用怪物的绘制方法即可—实际上该绘制方法每次只是绘制一张静态图片（这张静态图片是怪物动画的其中一帧）。

下面是怪物类的成员变量部分。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

上面的成员变量即可记录该怪物实例的各种状态。实际上以后程序要升级，比如为怪物增加更多的特征，如怪物可以拿不同的武器，怪物可以穿不同的衣服，怪物可以具有不同的攻击力……这些都可考虑定义成怪物的成员变量。

下面是怪物类的构造器，该构造器只要传入一个type参数即可，该type参数告诉系统，该怪物是哪种类型的怪物。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

从上面的粗体字代码可以看出，程序在创建怪物实例时，不仅负责初始化怪物的type成员变量，还会根据怪物类型来设置怪物的X、Y坐标。

➢ 如果怪物是炸弹和拿枪的人（都在地面上），那么它们的Y坐标与角色默认的Y坐标（在地面上）相同。如果怪物是飞机，那么怪物的Y坐标是随机计算的。

➢ 不管什么怪物，它的X坐标都是随机计算的。

上面程序中还用到一个Util工具类，该工具类仅仅包含一个计算随机数的方法。下面是该工具类的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Util.java

前面已经介绍了绘制怪物动画的思路：程序将会采用后台线程来控制不断地绘制怪物动画的下一帧，但实际上每次绘制的只是怪物动画的

某一帧。下面是绘制怪物的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

上面代码包含两个方法，其中draw（Canvas canvas）方法只是简单地对怪物类型进行了判断，并针对不同怪物类型使用不同的怪物动画。

draw（Canvas canvas）方法总是调用drawAni（Canvas canvas，Bitmap[] bitmapArr）方法来绘制怪物，调用后者时根据怪物类型的不同、怪物是否死亡将会传入不同的位图数组—每个位图数组就代表一组动画帧的所有位图。

drawAni（）方法中的①号粗体字代码就是根据drawIndex来获取当前帧对应的位图，而程序执行drawAni（）方法时，②号粗体字代码可以控制drawIndex自加一次，这样即可保证下次调用drawAni（）方法时就会绘制动画的下一帧。

drawAni（）方法还涉及一个drawCount变量，这个变量是控制动画刷新速度的计数器—程序在③号粗体字代码处进行了控制：只有当drawCount计数器的值大于6（对于其他类型的怪物，该值为4）时才会调用drawIndex++，这意味着当怪物类型是TYPE_MAN时，drawAni（）方法至少调用6次之后才会将drawIndex加1（即绘制下一帧位图）；当怪物是其他类型时，drawAni（）方法至少调用4次之后才会将drawIndex加1（即绘制下一帧位图）—这是因为程序中控制动画刷新的线程的刷新频率是固定的，即如后台线程控制每隔40ms调用一次怪物的drawAni（）方法，但如果每隔40ms就更新一次动画帧的话，那么游戏界面上的所有怪物“动”的速度都是一样的（每隔40ms刷新一次），而且它们都动得非常快。为了解决这个问题，程序就需要使用

drawCount来控制不同怪物实际每隔多少毫秒更新一次动画帧。对于上面代码来说，如果怪物类型是TYPE_MAN，则只有当drawCount计数器大于6时，才会更新一次动画帧，这意味着实际上每隔240ms才会更新一次动画帧；如果是其他类型的怪物，那么只有当drawCount计数器大于4时，才会更新一次动画帧，这意味着实际上每隔160ms才会更新一次动画帧。

提示：

如果游戏中还有更多类型的怪物，且这些怪物的动画帧具有不同的更新速度，那么程序还需要进行更细致的判断。

drawAni（）方法还涉及一个dieMaxDrawCount变量，这个变量用于控制怪物的死亡动画只会被绘制一次—在怪物临死之前，程序都必须播放怪物的死亡动画，该动画播放完成，该怪物就应该从地图上删除。当怪物已经死亡（isDie为真）且还未绘制死亡动画的任何帧时（dieMaxDrawCount等于初始值），程序在⑤号粗体字代码处将dieMaxDrawCount设置为与死亡动画的总帧数相等，程序每次调用drawAni（）方法时，⑥号粗体字代码都会把dieMaxDrawCount减1，当dieMaxDrawCount变为0时，表明该怪物的死亡动画的所有帧都绘制完成，接下来程序即可将该怪物从地图上删除了—在后面的MonsterManager类中将会看到程序根据怪物的dieMaxDrawCount为0来从地图上删除怪物的代码。

Monster还包含了startX、startY、endX、endY四个变量，这四个变量就代表了怪物当前帧所覆盖的矩形区域，因此，如果程序需要判断该怪物是否被子弹打中，只要子弹出现在该矩形区域内，即可判断怪物被子弹打中了。下面是判断怪物是否被子弹打中的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

接下来为怪物实现发射子弹的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

怪物发射子弹的方法是addBullet（），该方法需要调用getBulletType（）方法来判断该怪物所发射的子弹类型（不同怪物可能需要发射不同的子弹），如果getBulletType（）方法返回0，即代表这种怪物不发射子弹。

一旦确定了这种怪物发射子弹的类型，程序就可根据不同怪物计算子弹的初始X、Y坐标—基本上，子弹的X、Y坐标保持与怪物当前的X、Y坐标相同，再进行适当微调即可。程序最后两行粗体字代码创建了一个Bullet对象（子弹实例），并将新的Bullet对象添加到bulletList集合中。

当怪物发射了子弹之后，程序还需要绘制该怪物的所有子弹。下面是绘制怪物发射的所有子弹的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

上面程序中的updateShift（int shift）方法负责将怪物所有的子弹全部左移shift距离，这是因为界面上角色会不断地向右移动，角色会产生一个shift偏移，所以程序就需要将怪物（包括它的所有子弹）全部左移shift距离，这样才会产生逼真的效果。

上面程序中的粗体字代码使用deleteList集合收集所有越过屏幕的子弹，然后⑦号粗体字代码负责删除deleteList集合包含的所有子弹—这样即可把所有越过屏幕的子弹删除掉。

接下来程序采用循环遍历了该怪物发射的所有子弹，先获取子弹对应的位图，然后调用子弹的move（）方法控制子弹移动。上面方法中的最后一行粗体字代码负责绘制子弹位图。

Monster类还需要定义一个方法，用于判断怪物的子弹是否打中角色，如果打中角色，则删除该子弹。下面是该方法的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Monster.java

18.2.3 实现怪物管理类

由于游戏界面上会出现很多个怪物，因此程序需要额外定义一个怪物管理类来专门负责管理怪物的随机产生、死亡等行为。

为了有效地管理游戏界面上所有活着的怪物和已死的怪物（保存已死的怪物是为了绘制死亡动画），为怪物管理类定义如下成员变量。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

接下来在怪物管理类中定义一个随机生成怪物的工具方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

前面已经指出，当玩家控制游戏界面的角色不断地向右移动时，程序界面上的所有怪物、怪物的子弹都必须不断地左移，因此程序需要在MonsterManager类中定义一个控制所有怪物及其子弹不断左移的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

上面程序中的①号粗体字代码处于循环体之内，该循环将会控制把所有活着的怪物及其子弹全部都左移shift距离，如果移动之后的怪物的X坐标超出了屏幕范围，程序就会清除该怪物；②号粗体字代码同样处于循环体之内，②号代码的处理方式与①号代码的处理方式几乎是一样的，只是②号代码负责处理的是界面上已死的怪物。

上面程序中的最后一行粗体字代码则负责将玩家发射的所有子弹都左移shift距离—这也是必要的，原因与怪物发射的子弹都需要左移shift距离一样。

接下来要为MonsterManager实现一个新的方法，该方法可用于检查界面上的怪物是否将要死亡，将要死亡的怪物将会从monsterList集合中删除，并添加到dieMonsterList集合中，然后程序将会负责绘制它们的死亡动画。

下面为MonsterManager类增加一个checkMonster（）方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

上面这个方法的判断逻辑非常简单，程序把怪物分为两类进行处理。

➢ 如果怪物是地上的炸弹，只要炸弹炸到角色，炸弹也就即将死亡。上面程序中第一行粗体字代码处理了怪物是炸弹的情形。

➢ 对于其他类型的怪物，程序则需要遍历角色发射的子弹，只要任意一颗子弹打中了怪物，即可判断怪物即将死亡。上面程序中第二行粗体字代码正是遍历玩家所发射的子弹的代码。

最后MonsterManager还需要定义一个绘制所有怪物的方法。该方法的实现逻辑也非常简单，程序只要分别遍历该类的dieMonsterList和monsterList集合，并将集合中所有怪物绘制出来即可。对于dieMonsterList集合中的怪物，它们都是将要死亡的怪物，因此只要将它们所有的死亡动画帧都绘制一次，接下来就应该清除这些怪物了—Monster实例的dieMaxDrawCount成员变量为0时就代表所有死亡动画帧都绘制了一次。

下面是该drawMonster（）方法的代码，该方法就负责绘制所有怪物。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\MonsterManager.java

上面程序中的第一行粗体字代码负责遍历所有活着的怪物，并将它们绘制出来；第二行粗体字代码则负责遍历所有将要死亡的怪物，并将它们绘制出来。程序中③号粗体字代码检查该怪物的dieMaxDrawCount是否为0，如果为0，则表明该怪物已死亡、且该怪物的死亡动画所有帧都播放完成，应该将它们彻底删除。

18.2.4 实现“子弹”类

本游戏的子弹类比较简单，本游戏中的子弹不会产生爆炸效果。对子弹的处理思路是：只要子弹打中目标，子弹就会自动消失。正因为本游戏的子弹类比较简单，因此子弹类只需要定义如下属性即可。

➢ 子弹的类型。

➢ 子类的X、Y坐标。

➢ 子弹的射击方向（向左或向右）。

➢ 子弹在垂直方向（Y方向）上的加速度。

基于上面分析，程序为Bullet类定义了如下成员变量。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

接下来可以为Bullet类定义一个有参数的构造器，该构造器用于对子弹的类型，X、Y坐标，方向执行初始化。该构造器的代码比较简单，此处不再给出。

本游戏中不同怪物、角色发射的子弹都各不相同，因此不同类型的子弹将会采用不同的位图，这样玩家就会觉得不同怪物、角色发射的子弹都各不相同了。下面是Bullet类根据子弹类型来获取对应位图的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

从上面程序可以看出，根据子弹类型来获取对应位图的处理方式非常简单，程序只是用一个switch分支语句对子弹类型进行判断，然后根据不同的子弹类型取得相应的位图即可一实际上这里只是开发者的一种约定，并没有规定哪种怪物必须发射怎样的子弹，因此读者可以修改此处的代码，从而改变角色、怪物发射子弹的外观。

接下来程序还可以计算子弹在水平方向、垂直方向上的移动速度。下面是这两个方法的代码实现。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

从上面代码可以看出，当程序要计算子弹在X方向上的速度时，程序首先判断该子弹的射击方向是否向右，如果子弹的射击方向为向右，那么子弹在X方向上的速度为正值（保证子弹不断地向右移动）；如果子弹的射击方向为向左，那么子弹在X方向上的速度为负值（保证子弹不断地向左移动）。

接下来程序计算子弹在X方向上的速度时就非常简单了，除了第1种子弹以12为基数来计算X方向上的速度之外，其他子弹都是以8为基数来计算X方向上的速度的，这意味着只有第1种子弹的速度是最快的。

提示：

计算子弹速度时还乘以了ViewManager.scale缩放因子，其中ViewManager.scale缩放因子代表了整个界面与屏幕之间的缩放比。

计算Y方向上的速度时，程序的计算逻辑也非常简单。如果该子弹的yAccelate不为0（Y方向上的加速度不为0），则直接以yAccelate作为子弹在Y方向上的速度，这是因为程序设定：当玩家在跳起过程中，玩家射出的子弹应该斜向上发射；当玩家在降落的过程中，玩家射出

的子弹应该斜向下发射。除此之外，程序还使用switch语句对子弹的类型进行了判断：当子弹为第3种子弹时，子弹将会具有Y方向上的速度（这意味着子弹会不断地向下移动）—这是因为本程序设定飞机发射的炮弹是第3种子弹，这种子弹会模拟飞机投弹斜向下移动。

程序计算出了子弹在X方向、Y方向上的移动速度之后，接下来计算该子弹的移动就非常简单了，用X坐标加上X方向上的速度，Y坐标加上Y方向上的速度即可。下面是控制子弹移动的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Bullet.java

18.2.5 实现“角色”类

游戏的角色类（也就是受玩家控制的那个人）和怪物类其实差不多，它们具有很多相似的地方，因此它们在类实现上有很多相似之处。不过由于角色需要受玩家控制，它的动作比较多，因此程序需要额外为角色定义一个成员变量，用于记录该角色正在执行的动作，并且需要将角色的头部、腿部分开进行处理。

下面是Player类的成员变量。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

上面程序中的粗体字代码成员变量正是角色类与怪物类的差别所在，由于角色有名字、生命值（hp）、动作、移动方式这些特殊的状态，因此程序为角色定义了name、hp、action、move这些成员变量。

上面程序还为Player类定义了一个leftShootTime变量，该变量的作用有两个。

➢ 当角色的leftShootTime不为0时，表明角色当前正处于射击状态，因此此时角色的头部动画必须使用射击的动画帧。

➢ 当角色的leftShootTime不为0时，表明角色当前正处于射击状态，因此角色不能立即发射下一枪—必须等到leftShootTime为0时，角色才能发射下一枪。这意味着：即使用户按下界面上的“射击”按钮，也必须等到角色上一枪发射完成才会发射下一枪。

上面程序中的最后6行粗体字代码是绘制角色位图相关的成员变量，从这些成员变量可以看出，程序把角色按头部、腿部分开处理，因此程序需要为头部、腿部分开定义相应的成员变量。

Player类的构造器、初始化方法与Monster类的大致相似，此处不再介绍。

为了计算角色的方向（程序需要根据角色的方向来绘制角色），程序为Player提供了如下方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

从上面代码可以看出，程序可根据角色的action来计算角色的方向，只要action变量的值为奇数，即可判断该角色的方向为向右。

在介绍Monster类时已经提出，为了更好地在屏幕上绘制Monster对象以及所有子弹，程序需要根据角色在游戏界面上的位移来进行偏移，因此程序需要为Player方法来计算角色在游戏界面上的位移。下面是Player类中计算位移的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

从上面粗体字代码可以看出，程序计算角色位移的方法非常简单，只要用角色的初始X坐标减去角色当前X坐标即可。

游戏绘制角色、绘制角色动画的方法，与绘制怪物、绘制怪物动画的方法基本相似，只是程序需要分开绘制角色头部、腿部，读者可参考光盘代码来理解绘制角色、绘制角色动画的方法。

为了在游戏界面左上角绘制角色的名字、头像、生命值，Player类提供了如下方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

上面方法的实现非常简单，第一段粗体字代码调用Graphics绘制头像位图，第二段、第三段粗体字代码分别调用drawBorderString（）方法来绘制包边字，包边字的内容显示了角色的名字和生命值。

角色是否被子弹打中的方法与怪物是否被子弹打中的方法基本相似：只要判断子弹出现在角色图片覆盖的区域中，即可判断子弹打中了角色。

与怪物类相似的是，Player类同样也需要提供绘制子弹的方法，该方法负责绘制该角色发射的所有子弹，而且在绘制子弹之前，应该先判断子弹是否已越过屏幕边界，如果子弹越过屏幕边界，就应该将其清

除。由于绘制子弹的方法与Monster类中绘制子弹的方法大致相似，此处不再赘述。

由于角色发射子弹是受玩家单击按钮控制的，但本游戏的设定是角色发射子弹之后，必须等待一定时间才能发射下一发子弹，因此程序为Player定义了一个leftShootTime计数器，只要该计数器不等于0，角色就处于发射子弹的状态，角色不能发射下一发子弹。

下面是程序leftShootTime的getter方法与发射子弹的方法代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

正如从上面粗体字代码所看到的，程序每次发射子弹时都会将leftShootTime设为最大值，而leftShootTime会随着动画帧的绘制不断地自减，只有当leftShootTime为0时才可判断角色已结束射击状态。这样后面程序控制角色发射子弹时，也需要先判断leftShootTime的值：只有当leftShootTime的值小于、等于0时（角色不处于发射状态），角色才可以发射子弹。

由于玩家还可以控制界面上的角色移动、跳动，因此程序还需要实现角色移动、角色移动与跳跃之间的关系。程序为Player提供了如下两个方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\comp\Player.java

18.3 实现绘图工具类

上面程序中大量使用了Graphics绘图工具类，这是本游戏对Android绘图的Canvas提供的一个封装，Graphics类包含了各种常见的绘制几何图形、绘制位图的方法。该工具类中用于绘制几何图形的方法如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

需要说明的是，由于本游戏暂时并未需要绘制直线、矩形、弧等功能，因此上面3个方法其实在本游戏中并未用到，这3个方法只是为后续开发打下基础。

Graphics类中用于绘制字符串和包边字符串的方法如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

关于绘制包边字符串的逻辑其实也很简单，从程序中两行粗体字代码可以看出，其实就是先采用描边方式绘制字符串，再采用填充方式绘制字符串，这样即可形成包边字符串。

除此之外，Graphics还涉及如下绘制位图、处理位图的方法。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\game\Graphics.java

上面程序一共定义了4个绘制位图和处理位图的方法。

➢ **drawMatrixImage**：该方法用于从源位图中的 srcX、srcY 点开始，挖取宽 width、高height的区域，并对该图片进行trans变换、缩放scale（当scale为20时表示不缩放）、旋转degree角度后绘制到Canvas的drawX、drawY处。

➢ **drawImage**：该方法用于从源位图中的 srcX、srcY 点开始，挖取宽 width、高 height的区域，并将这块区域的图片绘制到Canvas的destX、destY处。

➢ **scale**：该方法用于对图片进行缩放。

➢ **mirror**：该方法用于对图片进行镜像变换。

上面4个方法的代码实现，读者可参考本书前面介绍Canvas、Matrix的部分进行理解。需要说明的是，本游戏并未用到上面的mirror方法，这个方法只是为游戏的后续开发做准备。

18.4 加载、管理游戏图片

为了统一管理游戏中所有的图片、声音资源，本游戏开发了一个ViewManager工具类，该工具类除了加载、管理游戏的图片、声音资源之外，还负责绘制游戏主界面。

ViewManager定义了如下成员变量来管理游戏涉及的图片、声音资源。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

正如上面程序中前两行粗体字代码所看到的，程序使用了SoundPool来管理游戏中的各种音效，比如角色发射子弹的声音、怪物被打中、爆炸的声音等。

除此之外，该类大量使用了位图数组来存储游戏需要使用的动画帧。

该类定义了一个loadResource（）方法来加载所有的图片、声音资源。该方法的代码如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

上面程序中第一段粗体字代码创建了SoundPool对象，这样即可使用SoundPool来管理短促的游戏音效。接下来第二段粗体字代码大量调用createBitmapByID（）工具方法来加载位图，这些代码就是根据需要为游戏中角色、怪物的动画帧加载位图。实际上，上面程序中还包含大量类似的代码，这些代码都用于加载游戏中其他动画帧的位图。

提示：

随着游戏规模的增大，游戏可能需要添加更多的怪物、更多的角色，那么此处加载动画帧的代码将会更多。

上面程序代码中加载动画帧的位图时用到一个createBitmapByID（）工具方法，该方法的代码如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

除此之外，ViewManager还定义了一个drawGame（）方法，该方法负责整个游戏场景。该方法的实现思路就是先绘制游戏地图，然后绘制游戏角色，最后绘制所有的怪物即可。下面是drawGame（）方法的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\ViewManager.java

上面方法中第一行粗体字代码使用drawImage（）方法来绘制背景位图，第二行粗体字代码依然使用了 drawImage（）方法来绘制背景位图—这是因为当角色在地图上不断地向右移动时，随着地图不断地向左拖动，地图就会不能完全覆盖屏幕右边，此时需要再绘制一张背景位图，这样才可以拼成完成的地图—这样就形成了无限循环的游戏地图。

18.5 实现游戏界面

至此，合金弹头游戏所需要的各种游戏元素准备完成，游戏所需的各种资源也准备完成了，接下来只要在程序界面上将这些游戏元素显示出来即可。

18.5.1 实现游戏Activity

先看本游戏的Activity。本游戏的Activity一样需要继承Android的Activity基类，并重写其生命周期方法。该Activity的代码如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\MainActivity.java

从上面Activity的3行粗体字代码可以看出，该游戏的Activity非常简单，程序只是加载main.xml界面设计文件作为游戏界面，然后程序创建了一个自定义组件：GameView实例，并将该实例添加到FrameLayout容器中，其中FrameLayout是main.xml界面设计文件包含的根UI组件。

上面的MainActivity重写了onResume（）、onPause（）两个生命周期方法，这样可以实现当游戏暂停时暂停播放背景音乐，当游戏恢复时继续播放背景音乐。

除此之外，上面Activity还设置了窗口的软键盘输入方式，而且本游戏只支持横屏方式，因此应该在AndroidManifest.xml文件中对该Activity进行如下配置。

程序清单：
codes\18\MetalSlug\app\src\main\AndroidManifest.xml

从上面代码可以看出，该游戏的主界面主要由GameView这个自定义组件来负责提供。下面详细介绍GameView组件的实现。

18.5.2 实现主视图

本GameView为了获取更好的游戏性能，程序让GameView继承SurfaceView，而不是继承普通的View，这是因为SurfaceView在游戏绘图方面比普通的View更出色。

GameView主要负责管理界面上的角色、当前游戏场景，以及绘图相关的API。GameView所需的成员变量如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

上面程序中第一行粗体字代码定义了该GameView要维护的角色，接下来三行粗体字代码定义的Canvas、Paint、SurfaceHolder都是绘图相关的API，最后一行粗体字代码定义的gStage则用于记录当前游戏处于何种场景：这是因为本GameView将会负责绘制所有的游戏场景，包括游戏开始时的登录场景、游戏失败的场景、正在游戏的场景。

除此之外，GameView还定义了一个stageList来保存游戏已经加载的所有场景。

GameView的构造器主要用于对上面程序中的各种成员变量执行初始化，该构造器此处不再赘述。

由于GameView会负责绘制游戏的所有场景，因此程序为GameView提供了如下方法来处理各种场景。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

从上面粗体字代码可以看出，doStage（）方法的处理逻辑非常简单，该程序根据GameView所处的场景来执行相应的方法，而这些方法则分别负责绘制游戏的不同场景。

➢ **doInit**：该方法负责执行初始化。

➢ **doLogin**：该方法负责绘制游戏登录界面。

➢ **doGame**：该方法负责绘制游戏界面。

➢ **doLose**：该方法负责绘制游戏失败界面。

GameView的doStage（）方法则由游戏线程负责调度，该线程会不断地执行doStage（）方法，这样程序只要改变GameView的stage变量，即可让GameView绘制不同的场景。

上面方法还涉及一个step参数，该参数用于控制当前场景处于哪个步骤。为了更好地管理这些步骤，GameView还提供了如下常量：

下面是doInit（）方法的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

从上面最后一行代码可以看出，doInit（）方法执行完成后，将会进入登录场景。下面是GameView的doLogin（）方法的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

doLogin（）方法的处理逻辑比较清晰，如果登录场景当前处于INIT步骤，程序将会创建一个loginView，并向loginView中添加一个按钮。接下来程序在①号粗体字代码处通过Handler将loginView添加到主界面中；如果游戏场景当前处于CLEAN步骤，程序将会在②号粗体字代码处通过Handler将loginView从界面上删除。

下面是setViewHandler的创建代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

从上面粗体字代码即可看出，setViewHandler的功能就是将接收到的消息携带的组件添加到主界面上。delViewHandler的实现代码与此相似，只是该Handler负责从界面上删除组件而已。

当游戏开始之后，游戏登录界面如图18.2所示。

图18.2 游戏登录界面

从登录场景上按钮的事件监听器代码可以看出，当玩家按下图18.2所示界面上的"start"按钮时，程序将会进入游戏场景，游戏场景由doGame（）负责提供。下面是doGame（）方法的代码。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

doGame（）方法稍微复杂一些，doGame（）方法可分为如下几种情况。

➢ 当游戏处于初始化步骤时，doGame（）方法会创建一个RelativeLayout 容器，并向该容器中添加4个按钮（这4个按钮将会放

在界面最下方）。最后由程序中③号粗体字代码将该RelativeLayout容器添加到主界面上。

➢ 当游戏处于逻辑步骤（LOGIC）时，doGame（）方法会调用MonsterManager、Player的方法来管理怪物、处理角色跳跃和移动、角色死亡等事件。

➢ 当游戏处于清除步骤时，doGame（）方法将会使用④号粗体字代码将该 RelativeLayout容器从主界面上清除。

➢ 当游戏处于绘图步骤（PAINT）时，doGame（）方法将会调用ViewManager的drawGame（）方法来绘制整个游戏界面。

从上面代码可以看出，虽然doGame（）方法的代码比较多，但其实这些代码主要就是创建了RelativeLayout容器，并为之添加了4个按钮。当游戏处于初始化步骤时，doGame（）方法添加的4个按钮按如图18.3所示的方式分布。

图18.3 游戏界面上的4个按钮

当游戏失败时，GameView将会使用doLose（）方法来显示失败界面。doLose（）方法的代码如下。

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

从上面代码可以看出，doLose（）与doLogin（）的实现逻辑基本相似，同样只是在界面上显示一个按钮，用于"原地复活"游戏。如图18.4所示是游戏失败时显示的界面。

程序最后还需要定义一条线程，该线程负责让游戏界面上的所有角色和怪物“动”起来。下面是游戏线程的代码。

图18.4 游戏失败界面

程序清单：
codes\18\MetalSlug\app\src\main\java\org\crazyit\metalslug\GameView.java

从上面线程的run（）方法中的粗体字代码可以看出，该线程所做的事情非常简单：只要游戏还未退出，该线程就会每隔40ms执行一次stageLogic（）方法（处理场景逻辑的方法）和doStage（）方法（根据当前场景执行绘制的方法），这意味着该游戏将会每隔40ms就会重绘一次游戏界面，这样即可让整个游戏界面上的角色、怪物都“动”起来。

最后程序还会在SurfaceHolder.Callback的surfaceCreated（）方法中启动该线程。下面是程序重写的surfaceCreated（）方法的代码。

上面粗体字代码就是创建并启动游戏线程的代码。当SurfaceView被加载时，surfaceCreated（）方法将会被自动激发，这样整个游戏线程就被启动起来了，从而控制整个游戏界面上的角色、怪物都“动”起来。

18.6 本章小结

本章开发了一个非常经典的射击类游戏：合金弹头。开发这款流行的小游戏难度适中，而且能充分激发学习热情，对于Android学习者来说

是一个不错的选择。学习本章可以帮助读者掌握开发手机游戏的基本功：开发者应该用面向对象的方式来定义界面上的所有角色、怪物、子弹……一些能与玩家交互的元素。除此之外，游戏的动画管理，以及动画后面的线程支持，还有游戏的发射子弹，子弹是否打中目标，玩家控制角色移动、跳跃、射击等行为的处理，都值得读者好好学习，这些都是开发手机游戏需要重点掌握的能力。

第19章 电子拍卖系统

本章要点

Android应用与传统应用整合的意义

Android应用整合传统应用的方式

基于HttpClient、JSON数据交换的整合方式

JSON基本知识

JSON语法

开发服务器端生成JSON响应的Servlet

开发Android客户端界面

使用HttpClient发送请求

使用HttpClient获取服务器响应

将服务器响应转换为JSON对象或数组

通过Android客户端加载服务器响应

本章介绍了一个实用的Android应用：电子拍卖系统。本章所介绍的应用不再是普通的Android项目，而是一个Android+Struts 2+Spring 4+Hibernate 4整合的应用。Struts 2+Spring 4+Hibernate 4提供了一个B/S结构的电子拍卖系统，对于使用PC的用户而言，他们可以使用浏览器来访问该系统；对于使用Android手机的用户而言，可以选择安装Android客户端程序，这样即可通过手机来使用该拍卖系统了。

与一般图书中所介绍的toy项目不同，本应用的服务器端采用了完整的Java EE应用架构：在技术实现上依赖于最流行的Struts 2+Spring+Hibernate组合；应用架构采用了最流行的、具有高度可扩展性的控制器层+业务逻辑层+DAO层的分层架构。Android客户端通过网络与服务器端的控制器组件交互，整个应用具有极好的示范性。

本应用的用户界面兼顾了手机和平板电脑两种设备，因此必须为手机屏幕和平板电脑屏幕设计用户界面。但由于该Android应用界面大量采用了Fragment，而Fragment代表了可复用的“Activity片段”，因此兼顾两种屏幕的界面复用了共同的Fragment。

提示：

Android系统发展的大势是与传统服务器应用系统整合，因为手机的硬件资源毕竟是有限的—计算能力有限，存储能力也有限，所以Android应用更适合作为应用的客户端—手机携带方便，可以随时随地开机运行应用，而且可以随时访问网络，并通过网络与服务器端应用交互，获取服务器端的数据。在未来的几年内，运行在手机上的电子商务客户端（各种电商系统都已有了手机客户端）、证券系统客户端、金融系统客户端将会大量出现。

由于本章介绍的重点是Android开发，因此将会详细介绍Android应用的开发，以及如何让Android应用与远程服务器交互。本章不会全面、详细地介绍服务器端的Spring、Hibernate开发，这些内容不是短短的一章可以讲完的。如果读者需要详细学习Spring、Hibernate相关知识，请参考疯狂Java体系的《轻量级Java EE企业应用实战》一书。

19.1 系统功能简介和架构设计

本章介绍的应用不再是简单的单机小应用，而是一个Android与传统服务器应用整合的应用，在这个应用中，服务器端程序依然保持了良好的应用架构，而客户端则使用Android程序充当。

19.1.1 系统功能简介

本章所介绍的系统是一个功能不太复杂的电子拍卖系统，本系统从实际电子商务平台上抽取，只取出其中部分核心功能实现，以作为示范应用，向读者展示一种良好的程序架构。

本章的电子拍卖系统其实就是一个电子商务平台，只要将该系统部署在互联网上，全球的客户都可以在该系统上发布想售出的商品，也可以对拍卖中的商品参与竞价。整个过程无须任何人工干预，由系统自动完成。

如果系统中提供与电子银行的接口，将可以通过电子银行的操作，实现买家对卖家的自动付款。一旦付款成功，就可以利用全球物流供应系统将拍卖物品发送到买家手中。可见，这种电子拍卖系统是一种开放式的、成本极其低廉的系统，大部分工作无须人工干预，系统自动完成管理。当然，由于本系统是一个示范系统，因此不提供与电子银行的接口，只是模拟了用户添加拍卖物品，用户参与拍卖的基本行为。拍卖结束后，系统会判断物品是否被最高竞价者获得。

该电子拍卖系统模拟了淘宝系统部分功能，抽取了实际电子拍卖系统部分功能，但没有提供如个人身份认证、信用管理等细节问题，本系统主要实现了电子拍卖系统中的核心功能。

本项目的服务器端是一个完整的Java EE项目，服务器端采用了控制器层、业务逻辑层、DAO层的分层架构。Android客户端应用只负责与服务器的控制器组件交互，Android应用采用Apache HttpClient向服务器端的控制器发送请求并获取服务器响应，这样即可实现Android系统与电子拍卖系统之间的通信。

本系统要求用户参与拍卖之前，必须登录系统。本系统提供了系统登录验证，登录验证在服务器端通过Filter实现，Filter拦截用户请求，并判断Session中是否保存了当前用户ID，如果保存了用户ID，即该用户已经登录；否则没有登录。

对于物品的管理，本系统可以查询拍卖物品，添加拍卖物品，增加物品种类，竞价处理，以及发送邮件通知用户所参与的竞价。

➢ 注册用户可以添加拍卖物品，添加物品种类。在添加之前必须登录系统。

➢ 注册用户可以浏览当前拍卖中的物品，以及流拍的物品。

➢ 注册用户可以参与竞价，参与的竞价系统将通过邮件通知用户。

19.1.2 系统架构设计

本系统的服务器采用Java EE的分层结构，分为视图层、控制器层、业务逻辑层和DAO层。分层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

中间层采用Struts 2+Spring 4+Hibernate 4，为了分离控制层与业务逻辑层，又可细分为：

➢ 控制器层，就是MVC模式里面的“C”（Controller），负责表现层与业务逻辑层的交互，调用业务逻辑层，并将业务数据返回给表现层来显示。MVC框架采用流行的Struts 2。

➢ Service层（业务逻辑层），负责实现业务逻辑，对DAO对象进行正面模式的封装。

➢ DAO层（数据访问对象层），负责与持久化对象交互，封装了数据的增、删、查、改原子操作。

➢ Domain O bject 层（持久化对象层），通过实体/关系映射工具将关系型数据库的数据映射成对象，实现以面向对象方式操作数据库。这个系统采用 Hibernate 作为 O/R Mapping框架。

本系统使用MySQL数据库存放数据。

服务器端应用的总体架构图如图19.1所示。

图19.1 服务器端应用的总体架构图

当采用Android应用作为客户端时，Android应用可以通过网络与服务器端交互，Android应用将会通过Apache HttpClient向服务器端的控制器发送请求，并获取服务器响应，服务器响应将会采用JSON数据格式——可以更有效地进行数据交互。

Android应用向服务器端的控制器发送请求，此处的控制器并不是Struts 2的Action，而是直接采用Servlet充当。在Servlet 3.1规范中，开发Servlet已经不再需要在web.xml文件中进行配置了，因此开发起来十分方便。

图19.2显示了Android客户端与服务器整合的架构图。

图19.2 Android客户端与服务器整合的架构图

细心的读者可能已经发现，图19.1与图19.2所示的结构十分相似，服务器端应用的结构基本不需要改变，只要在传统Java EE应用的基础上增加系列Servlet，这些Servlet负责向Android客户端提供响应即可。

图19.1与图19.2十分相似正好说明了Java EE应用架构的优势：当整个应用的某一层需要改变或重构时，应用系统能最大限度地“复用”以前的应用组件，而不需要重新开发，这样才能保证以前编写的项目代码具有高度的保值性。也正是由于上面两个结构图的相似性，才可让我们非常快速地为该应用增加各种平台的客户端系统。

19.2 JSON简介

Android客户端与服务器端通信时需要一种合适的数据交换格式，本系统采用了JSON作为Android客户端与服务器端的数据交换格式。

提示：

实际上还可以考虑使用Web Service来作为Android应用与服务器端的通信技术，对于Web Service而言，底层采用XML作为数据交换格式。

JSON的全称是JavaScript Object Notation，即JavaScript对象符号，它是一种轻量级的数据交换格式。JSON的数据交换格式既适合人来读/写，也适合计算机本身解析和生成。最早的时候，JSON是JavaScript语言的数据交换格式，后来慢慢发展成一种语言无关的数据交换格式，这一点非常类似于XML。

JSON主要在类似于C的编程语言中广泛使用，这些语言包括C、C++、C＃、Java、JavaScript、Perl、Python等。JSON提供了多种语言之间完成数据交换的能力，因此，JSON也是一种非常理想的数据交换格式。JSON主要有如下两种数据结构。

➢ 由key-value对组成的数据结构。这种数据结构在不同的语言中有不同的实现。例如，在JavaScript中是一个对象，在Java中是一种Map结构，在C语言中则是一个struct，在其他语言中可能有record、dictionary、hash table等。

➢ 有序集合。这种数据结构在不同语言中可能有list、vector、数组和序列等实现。

上面两种数据结构在不同的语言中都有对应的实现。因此，这种简便的数据表示方式完全可以实现跨语言，所以可以作为程序设计语言中通用的数据交换格式。在JavaScript中主要有两种JSON语法，其中一种用于创建对象；另一种用于创建数组。

19.2.1 使用JSON语法创建对象

使用JSON语法创建对象是一种更简单的方式，既可避免书写函数，也可避免使用new关键字，而是直接获取一个JavaScript对象。对于早期

的JavaScript版本，如果要使用JavaScript创建一个对象，在通常情况下可能会这样写：

从JavaScript 1.2开始，创建对象有了一种更快捷的语法，语法如下：

这种语法就是一种JSON语法。显然，使用JSON语法创建对象更加简捷、方便。图19.3显示了这种语法示意图。

图19.3 JSON创建对象的语法示意图

从图19.3可以看出，创建对象object时，总以{开始，以}结束，对象的每个属性名和属性值之间以英文冒号（：）隔开，多个属性定义之间以英文逗号（，）隔开。语法格式如下：

必须注意的是，并不是每个属性定义后面都有英文逗号（，），必须在后面还有属性定义时才需要逗号（，）。因此，下面的对象定义是错误的。

因为sex属性定义后多出一个英文逗号，最后一个属性定义的后面应直接以} 结束，不再有英文逗号（，）。

当然，使用JSON语法创建JavaScript对象时，属性值不仅可以是普通字符串，也可以是任何基本数据类型，还可以是函数、数组，甚至是另外一个JSON语法创建的对象。例如：

19.2.2 使用JSON语法创建数组

使用JSON语法创建数组也是非常常见的情形，在早期的JavaScript语法里，我们通过如下方式来创建数组。

或者，通过如下方式创建数组：

但如果我们使用JSON语法，则可以通过如下方式创建数组：

如图19.4所示是JSON创建数组的语法示意图。

图19.4 JSON创建数组的语法示意图

正如从图19.4中所见到的，JSON创建数组总是以英文方括号（[）开始，然后依次放入数组元素，元素与元素之间以英文逗号（，）隔开，最后一个数组元素后面不需要英文逗号，但以英文反方括号（]）结束。使用JSON创建数组的语法格式如下：

与使用JSON语法创建对象相似的是，数组的最后一个元素后面不能有英文逗号（，）。

鉴于JSON语法的简单易用，而且作为数据传输载体时，数据传输量更小，假设需要交换一个对象person，其name属性为yeeku，gender属性为male，age属性为29，使用JSON语法可以简化为如下形式：

但如果使用XML数据交换格式，则需要采用如下格式：

对比这两种表示方式，第一种方式明显比第二种方式更加简洁，数据传输量也更小。

19.2.3 Java的JSON支持

当服务器返回一个满足JSON格式的字符串后，我们可以利用JSON项目提供的工具类将该字符串转换为JSON对象或JSON数组。

幸运的是，Android系统内置了对JSON的支持，在Android SDK的org.json包下提供了JSONArray、JSONObject、JSONStringer和JSONExeception等类，通过这些类即可非常方便地完成JSON字符串与JSONArray、JSONObject之间的相互转换。

JDK默认并未提供对JSON的支持，不过我们可以登录http：//www.json.org/java/index.html，在该页面可以看到JSONArray、JSONObject、JSONStringer和JSONExeception等类的源代码，读者可以自行下载这些源代码，把它们加入项目中即可使用。

提示：

笔者已经下载了这些源代码，并把它们编译成了*.class文件，将它们打包成了json.jar包，该JAR包位于光盘的codes\19\auction\WEB-INF\lib目录下。读者直接把该JAR包添加到项目的类加载路径下即可使用。

Java的JSON支持主要依赖于JSONArray和JSONObject两个类，其中：

➢ JSONArray代表一个JSON数组，它可完成Java集合（集合元素可以是对象）与JSON字符串之间的相互转换。

➢ JSONObject代表一个JSON对象，它可完成Java对象与JSON字符串之间的相互转换。

19.3 发送请求的工具类

本系统采用Apache HttpClient与远程服务器通信。为了简化HttpClient的用法，本系统定义了一个工具类对HttpClient进行封装。该工具类定义了如下两个方法来发送请求。

➢ **getRequest（）**：发送GET请求。

➢ **postRequest（）**：发送POST请求。

该工具类的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\util\HttpUtil.java

上面的HttpUtil中两行粗体字代码定义了getRuquest（）和postRequest（）两个方法，这两个方法用于向服务器发送请求，返回服务器的响应。在HttpUtil中提供这两个方法之后，接下来在Android应用中只要调用这两个方法即可实现与服务器的通信。

19.4 用户登录

在使用该电子拍卖系统之前，用户必须先登录系统。用户在Android应用中输入用户名、密码，单击“登录”按钮，程序即可通过HttpUtil向服务器发送请求，通过服务器来验证用户所输入的用户名、密码是否正确。

19.4.1 处理登录的Servlet

处理用户登录的Servlet只是前端控制器，它的作用只有三个。

➢ 获取请求参数。

➢ 调用业务逻辑组件的方法来处理用户请求。

➢ 根据处理结果来生成输出。

由于本项目的服务器组件都是部署在Spring容器中的，因此程序需要获取Spring容器中的业务逻辑组件，而不是自行创建业务逻辑组件。

由于所有Servlet都需要获取Spring容器中的业务逻辑组件，因此本程序提供了一个Servlet基类，该基类中定义了一个方法来获取Web应用中的Spring容器。该Servlet基类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\base\BaseServlet.java

正如从上面的粗体字代码所看到的，该Servlet基类中定义了一个getCtx（）方法，通过该方法即可获取Spring容器，这意味着应用中的所有Servlet都可通过该方法来获取Spring容器。接下来服务器端的所有应用程序只要继承该Servlet，并通过该方法来获取Spring容器即可。

下面是处理用户登录的Servlet类代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\LoginServlet.java

该Servlet中第一行粗体字代码使用@WebServlet指定了该Servlet的URL为/android/login.jsp。第二行粗体字代码调用了业务逻辑组件的validateLogin（）方法来处理用户登录。如果用户登录成功，程序将用户的ID放入HTTP session中，方便程序跟踪用户的登录状态。

如果读者对Spring、Hibernate等框架不熟悉也不要紧，可以直接使用LoginServlet访问系统数据库，并根据用户输入的用户名、密码返回该用户的ID，该Android客户端一样可以正常工作。只是采用这种方式开发的应用不太正规，后期的扩展性很差而已。

19.4.2 用户登录客户端

Android客户端中用户登录界面有两个文本框，用于接收用户输入的用户名、密码信息。下面是用户登录的界面布局XML文档。

程序清单：
codes\19\AuctionClient\app\src\main\res\layout\login.xml

该界面布局的效果如图19.5所示。

图19.5 用户登录界面

在图19.5所示的界面中输入用户名、密码之后，如果用户单击"登录"按钮将会激发登录处理，也就是通过HttpUtil向服务器发送请求。用户登录的Activity代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\Login.java

上面Activity的query（）方法中两行粗体字代码用于向指定URL发送请求，并将服务器响应封装成JSONObject。

上面的程序为登录按钮的单击事件绑定了事件监听器，当用户单击“登录”按钮时，程序先执行输入校验，如①号粗体字代码所示；接下来执行登录处理，如②号粗体字代码所示—程序调用了loginPro来处理用户的登录请求。

如果用户输入的用户名、密码不正确，系统将会调用DialogUtil来显示对话框，对话框提示登录失败。由于本系统经常需要显示各种对话框，因此程序专门把它定义成一个独立的类。DialogUtil类的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\util\DialogUtil.java

如果登录成功，系统启动AuctionClientActivity，这个Activity相当于系统主界面，用户可通过该界面提供的ListView进入各功能。

AuctionClientActivity的界面布局文件可分为两个：为普通手机屏幕提供的应用界面和为平板电脑屏幕提供的界面。下面先看为普通手机屏幕提供的界面布局文件。

程序清单：
codes\19\AuctionClient\app\src\main\res\layout\activity_main.xml

上面程序界面的主要组件就是AuctionListFragment，该Fragment显示一个ListView，该ListView中每个列表项代表一个系统功能。

下面再看为平板电脑屏幕提供的界面布局文件。

程序清单：
codes\19\AuctionClient\app\src\main\res\layout-sw480dp\activity_main.xml

上面的程序界面中同样包含了一个AuctionListFragment。除此之外，该程序界面中还包含了一个FrameLayout容器，该容器负责装载对应功能的Fragment组件。

由于layout和layout-sw480dp目录下分别包含了activity_main.xml界面布局文件，因此AuctionClientActivity会根据屏幕尺寸自动加载不同目录下的界面布局文件。除此之外，该Activity必须根据界面布局来进行处理：如果程序加载layout目录下的activity_main.xml布局文件，用户点击功能项时，系统将会启动相应的Activity来显示功能；如果程序加载layout-sw480dp目录下的activity_main.xml布局文件，用户点击功能项时，系统将会使用FrameLayout加载相应功能的Fragment。

AuctionClientActivity的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AuctionClientActivity.java

从上面的代码可以看出，如果在手机上运行，AuctionClientActivity主要提供一个ListView，当用户单击不同的列表项时，程序将会启动不

同的Activity。AuctionClientActivity的运行效果如图19.6所示。

用户单击图19.6所示的“主菜单”（其实是ListView）的任一个列表项，系统将会启动对应的Activity，从而允许用户执行相应的操作。

但如果在平板电脑上运行，AuctionClientActivity则提供一个ListView和一个FrameLayout容器，当用户单击不同的列表项时，程序将使用FrameLayout装载相应功能的Fragment。在平板电脑上运行AuctionClientActivity的效果如图19.7所示。

图19.6 系统主菜单（手机）

图19.7 系统主界面（平板电脑）

用户单击图19.7所示的“主菜单”（其实是ListView）的任一个列表项，系统将会使用右边的FrameLayout装载相应功能的Fragment。

19.5 查看流拍物品

当用户单击“浏览流拍物品”时，程序将使用FrameLayout装载相应的Fragment，启动相应的Activity装载相应的Fragment，并显示系统中的流拍物品。

19.5.1 查看流拍物品的Servlet

查看流拍物品的Servlet也只是前端控制器，它会调用Spring容器中业务逻辑组件的方法，通过该方法获取系统中的流拍物品。

该Servlet类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewFailServlet.java

上面程序中的粗体字代码调用了Spring容器中业务逻辑组件的方法来获取系统中的流拍物品。程序的第二行粗体字代码把Java集合包装成JSONArray对象，程序中①号粗体字代码用于把JSONArray对象转换为JSON字符串，并作为响应输出到客户端。

如果直接向服务器的/android/viewFail.jsp发送请求，将可以看到如图19.8所示的输出。

图19.8 浏览流拍物品的JSON响应

图19.8显示的字符串就是典型的JSON格式字符串，Android客户端程序只要调用JSONArray类的构造器即可把它转换为JSONArray—就是一个JSON数组。

提示：

与前面登录的Servlet类似，如果读者对Spring、Hibernate等技术不熟，也可以直接在Servlet中访问数据库，并获取系统中所有的流拍物品。只要该Servlet能获取到所有的流拍物品，并把它们转换为图19.8所示的JSON字符串输出，该Android客户端就可正常工作。

19.5.2 查看流拍物品客户端

下面是查看流拍物品的程序界面布局代码。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\view_item.xml

上面界面布局中的主要组件就是一个ListView，用于显示多个物品。查看流拍物品的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ViewItemFragment.java

上面程序中的①号粗体字代码用于向指定URL发送请求，并把服务器响应包装成JSONArray对象，这就完成了Android客户端与服务器的交互。JSONArray对象的本质就是一个数组，它提供了如下常用方法。

➢ **length（）：**返回该JSON数组的长度。

➢ **optJSONObject（int index）：**获取指定索引处的JSONObject对象。

➢ **optJSONArray（int index）：**获取指定索引处的JSONArray对象。

➢ **optXxx（int index）：**获取指定索引处的数组元素。

对于开发者而言，当程序中获取到JSONArray对象之后，完全可以把它当成数组处理。

本程序提供了一个JSONArrayAdapter类，它的本质是一个Adapter，该Adapter用于对JSON数组进行包装，并作为ListView的内容Adapter。JSONArrayAdapter类的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\au

ction\client\JSONArrayAdapter.java

上面的Adapter负责为ListView提供列表项，ListView的列表项可以是如下两种情况。

➢ 当 ListView的列表项有图标时，该 Adapter 所提供的每个列表项是一个 LinearLayout （里面包括一个图标和一个TextView）。

➢ 当 ListView 的列表项没有图标时，该 Adapter 提供的每个列表项就是一个普通的TextView。

上面的JSONArrayAdapter可以多次复用，因此程序把它单独定义成一个Adapter类，这样可以多次使用该Adapter来封装JSONArray对象。

在平板电脑上查看流拍物品，将看到如图19.9所示的界面。

从图19.9所示的列表中只能看到流拍物品的物品名，如果用户希望看到该物品的详情，则可以单击该物品对应的列表项，此时程序将会触发ViewItemFragment中的viewItemDetail（position）方法，如ViewItemFragment类中的③号粗体字代码所示。

viewItemDetail（position）方法会使用对话框加载系统中的detail.xml界面布局文件，并显示该流拍物品的详情。当用户单击图19.9所示的列表项时，系统将会显示如图19.10所示的对话框。

图19.9 在平板电脑上查看流拍物品

上面的ViewItemFragment除了用于显示流拍物品之外，也可用于查看竞得物品，因为两个功能的前端实现基本是一致的，只是Android需要调用的服务器端方法不同而已。

如果在手机上使用，系统还需要使用一个 Activity 来“盛装”该Fragment。由于该系统中有大量Fragment需要使用Activity来显示，因此本系统专门开发了一个FragmentActivity，它只是用于显示一个Fragment。该Activity的代码如下。

图19.10 查看物品详情

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\app\base\FragmentActivity.java

上面的粗体字代码定义了一个抽象方法，该抽象方法将会返回一个Fragment，而该Activity的作用就是加载、显示该Fragment。

FragmentActivity用于被其他Activity继承，继承它的Activity只要重写getFragment（）方法即可。下面是ViewItem Activity的代码。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ViewItem.java

从上面的代码不难看出，ViewItem Activity只是显示ViewItemFragment而已—在显示该Fragment之前，向该Fragment传入参数。

在普通手机上查看流拍物品，将可以看到如图19.11所示的界面。

图19.11 在手机上查看流拍物品

19.6 管理物品种类

管理物品种类包括浏览系统中的物品种类、添加物品种类两大主要功能。Android客户端主要充当用户交互的客户端：显示系统中物品种类；添加种类时提供输入框供用户输入种类名称、种类描述等必要信息。

19.6.1 浏览物品种类的Servlet

浏览物品种类的Servlet同样只是充当控制器，调用业务逻辑组件的业务方法来获取系统中的物品种类。该Servlet的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewKindServlet.java

与前一个Servlet的实现基本相似，该Servlet也是调用Spring容器中业务逻辑组件相应的业务方法，并把方法返回的字符串包装成JSONArray，再把JSONArray转换为字符串作为服务器响应。

直接使用浏览器浏览上面的Servlet，将可以看到如图19.12所示的JSON字符串输出。

图19.12 JSON字符串响应

19.6.2 查看物品种类

在Android客户端查看物品种类，只要先通过HttpUtil向服务器发送请求，并把服务器响应字符串转换成JSONArray对象，再使用Adapter包装JSONArray对象，并使用ListView进行显示即可。

查看物品种类的界面布局代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\res\layout\manage_kind.xml

上面的界面布局中定义了一个ListView来显示系统中的物品种类，程序只要把服务器返回的物品种类包装成Adapter，并使用该ListView来显示所有的物品种类即可。下面是显示物品种类的Fragment代码。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageKindFragment.java

上面程序的onCreateView（）方法中的粗体字代码是实现向服务器发送请求，把服务器响应转换成JSONArray对象，并使用ListView显示物品种类的核心代码。由于物品种类包含的信息量并不大，只有种类名称和种类描述两种，因此程序使用了KindArrayAdapter来包装JSONArray对象，KindArrayAdapter提供的列表项既包括种类名称，也包括种类描述。下面是该Adapter类的代码。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\KindArrayAdapter.java

从上面程序中的粗体字代码可以看出，该Adapter提供的每个列表项不仅包括种类名称，而且包括种类描述，这样用户查看种类时一目了然。

用户查看物品种类时将可看到如图19.13所示的界面。

图19.13 在平板电脑上查看物品种类

为了在手机上运行该应用，程序提供了一个ManageKind Activity来包装、显示该Fragment。ManageKind的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageKind.java

图19.14 在手机上查看物品种类

正如上面的粗体字代码所示，ManageKind仅仅是包装、显示ManageKindFragment，因此在手机上运行该程序时，将可以看到如图19.14所示的界面。

在图19.13、图19.14所示界面中有一个"添加种类"按钮，当用户单击该按钮时，Fragment将会回调它所在Activity的onItemSelected（）方法。对于在平板电脑上运行的Activity，onItemSelected（）方法将会让FrameLayout装载AddKindFragment；而在手机上运行的Activity，onItemSelected（）方法将会启动AddKindActivity（该Activity仅仅是包装、显示AddKindFragment）。

19.6.3 添加种类的Servlet

添加物品种类的Servlet也是调用业务逻辑组件的方法来完成添加的。该Servlet的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddKindServlet.java

上面Servlet调用业务逻辑组件的方法添加物品后，直接返回了添加结果的字符串，因此Android客户端只要直接显示该添加结果即可。

19.6.4 添加物品种类

添加物品种类的界面上包括两个输入框，用于接受用户输入种类名称和种类描述，并提供添加、取消两个按钮。该界面布局比较简单，此处不再给出界面布局代码。

添加物品种类的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddKindFragment.java

上面的Fragment先对用户输入的种类名称、种类描述进行输入校验，然后调用addKind（）方法来添加物品种类，该方法利用HttpUtil发送POST请求完成添加。添加成功后，可以看到系统显示如图19.15所示的对话框。

图19.15 在平板电脑上添加物品种类

为了兼顾手机屏幕，同样需要定义Activity来加载、显示该Fragment。AddKind Activity的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddKind.java

正如上面的粗体字代码所示，AddKind什么都没干，仅仅是加载、显示AddKindFragment。在手机上添加物品种类，如果添加成功，将看到如图19.16所示的对话框。

19.7 管理拍卖物品

管理拍卖物品包括浏览自己的拍卖物品、添加拍卖物品两大主要功能。Android客户端主要充当用户交互的客户端：显示当前用户的拍卖物品；添加拍卖物品时提供输入框供用户输入物品名称、物品描述等必要信息。

图19.16 在手机上添加物品种类

19.7.1 查看自己的拍卖物品的Servlet

查看自己的拍卖物品的Servlet调用业务逻辑组件的业务逻辑方法来获取自己的拍卖物品，接下来该Servlet将会把该物品列表包装成JSONArray对象，再转换成JSON字符串后输出。

下面是该Servlet的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewOwnerItemServlet.java

该Servlet中的粗体字代码调用了业务逻辑方法来获取当前用户所有处于拍卖中的物品。直接使用浏览器向该Servlet发送请求，将可以看到该Servlet输出如图19.17所示的JSON字符串。

图19.17 JSON字符串响应

当服务器返回如图19.17所示的JSON字符串响应之后，Android客户端就可以把它转换成JSONArray对象了，并从中获取详细的物品信息。

19.7.2 查看自己的拍卖物品

查看拍卖物品的界面使用ListView来显示物品列表。该界面布局的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\res\manage_item.xml

上面的界面布局中定义了一个ListView来显示当前用户的拍卖物品。除此之外，该界面还包含了几个按钮，其中一个按钮用于启动添加物品的用户界面。

管理物品的Fragment只要通过HttpUtil向服务器发送请求，并把服务器响应转换成JSONArray对象，然后把JSONArray对象包装成Adapter，再使用ListView来显示这些物品即可。管理物品的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ManageItemFragment.java

上面程序中的粗体字代码使用了HttpUtil向服务器发送请求，并使用ListView来显示服务器返回的物品列表。当用户在平板电脑上进入管理物品的界面时，将会看到如图19.18所示的界面。

图19.18 管理自己的拍卖物品

图19.18只是列出拍卖物品的物品名，如果用户需要查看该物品的详情，则可以选择单击代表该物品的列表项，程序将会触发viewItemInBid（position）方法（如上面程序中①号粗体字代码所示），该方法将会启动一个对话框来显示该物品的详情。该对话框中定义了多个文本框，多个文本框显示了该物品的详情。

用户单击物品列表中的指定物品时，将可以看到如图19.19所示的界面。

图19.19 查看物品详情

为了兼顾手机屏幕，同样需要定义Activity来加载、显示该Fragment。ManageItem Activity的代码如下。

在手机上查看物品详情，将可以看到如图19.20所示的界面。

在图19.18、图19.20所示界面上都包含一个“添加物品”按钮，用户单击该按钮即可添加拍卖物品。

19.7.3 添加拍卖物品的Servlet

添加拍卖物品的Servlet将会调用业务逻辑组件的方法来添加物品，添加物品时Servlet先解析请求参数，这些请求参数代表了新增物品的属性，再调用业务逻辑方法即可。该Servlet的代码如下。

图19.20 在手机上查看物品详情

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddItemServlet.java

上面程序中的粗体字代码用于调用业务逻辑方法来添加物品，如果添加成功，该Servlet将会输出添加成功的响应；否则将会输出添加失败的响应。

提供该Servlet之后，接下来Android客户端就可向该Servlet发送请求来添加物品了。

19.7.4 添加拍卖物品

添加拍卖物品的程序界面中包含多个文本框，这些文本框用于接受用户输入的物品属性。添加拍卖物品的界面布局文件如下。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\add_item.xml

上面的界面布局中包含了两个Spinner组件，它们定义了两个列表框供用户选择有效时间和物品种类。对于选择有效时间的Spinner来说，程序可以直接指定一个数组作为它的列表项；但对于选择物品种类的Spinner来说，程序必须加载系统中的所有物品种类来作为列表项。因此，为了在Spinner中加载物品种类，程序必须

向/android/viewKind.jsp发送请求，然后把响应包装成Adapter，接下来使用Spinner来显示种类列表。

当用户在平板电脑上进入添加拍卖物品的界面之后，将会看到如图19.21所示的输入界面。

图19.21 添加拍卖物品

从图19.21所示的界面可以看出，程序最下面的物品种类就是从系统中加载的。当用户填写了拍卖物品的详情之后，程序将会向/android/addItem.jsp发送请求来添加拍卖物品。添加拍卖物品的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddItemFragment.java

上面程序中的①号粗体字代码向/android/viewKind.jsp发送请求，并把服务器响应包装成JSONArray对象，然后使用Spinner把这些物品种类显示出来即可。

当填写了拍卖物品的详情之后，用户单击“添加”按钮将先执行输入校验，然后调用addItem（）方法来添加物品，如上面程序中两行粗体字代码所示。

如果用户添加拍卖物品成功，将会看到如图19.22所示的对话框。

图19.22 在平板电脑上添加拍卖物品成功

为了兼顾手机屏幕，同样需要定义Activity来加载、显示该Fragment。AddItem Activity的代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddItem.java

从上面的粗体字代码不难看出，AddItem Activity仅仅是加载并显示AddItemFragment。在手机上添加物品成功后，可以看到如图19.23所示的界面。

19.8 参与竞拍

用户可以通过物品种类来浏览系统中的拍卖物品，找到合适的拍卖物品之后，可以对该物品进行竞价。

图19.23 在手机上添加拍卖物品成功

19.8.1 选择物品种类

前面介绍的/android/viewKind.jsp可以生成所有物品种类的响应，Android客户端只要向该Servlet发送请求即可显示物品种类供用户选择。

选择物品种类的界面上主要包含一个ListView，该ListView列出系统中全部物品种类，当用户单击某个物品种类时，程序将会显示该种类下的全部物品。

选择物品种类的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ChooseKindFragment.java

程序中①号粗体字代码用于向指定URL发送请求，并将服务器响应转换成JSONArray对象，接下来程序只要把JSONArray对象包装成Adapter，并使用ListView显示种类列表即可。

当用户单击指定物品种类时，程序将会回调该Fragment所在Activity的onItemSelecte（）方法，并把当前物品种类作为参数传过去，而该Fragment所在Activity将会负责加载Fragment，或启动对应的Activity来显示该种类下的全部拍卖物品。

19.8.2 根据种类浏览物品的Servlet

根据种类浏览拍卖物品的Servlet调用业务逻辑方法来获取所有物品。该Servlet所做的就是调用业务逻辑方法来查询指定种类下的全部物品，并把这些物品包装成JSONArray，再转换成JSON字符串输出。

下面是该Servlet的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ItemListServlet.java

上面程序中的粗体字代码调用了业务逻辑方法来获取指定种类的全部物品，接下来程序将这些物品包装成JSONArray，再转换成JSON字符串输出。

直接使用浏览器访问该Servlet，将看到如图19.24所示的JSON字符串。

图19.24 服务器响应的JSON字符串

19.8.3 根据种类浏览物品

服务器端Servlet生成图19.24所示的JSON字符串响应后，Android客户端只要向该Servlet发送请求并把该响应转换为JSONArray对象，再将该JSONArray对象包装成Adapter，并使用ListView来显示这些物品即可。

下面是ChooseItemFragment的代码。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\ChooseItemFragment.java

上面程序中的粗体字代码用于向指定Servlet发送请求，并把服务器响应转换成JSONArray对象，再将该对象包装成Adapter，并使用ListView来显示这些物品。

当用户在平板电脑上选择指定物品种类时，将会看到如图19.25所示的物品列表。

图19.25 在平板电脑上查看指定种类的拍卖物品

图19.25中列出了指定种类下所有的拍卖物品，用户可以单击指定物品进入拍卖界面。

提示：

系统也为兼容手机屏幕提供了ChooseItem Activity，该Activity的作用也只是包装并显示ChooseItemFragment，此处不再给出ChooseItem的代码。

19.8.4 参与竞价的Servlet

参与竞价的Servlet调用业务逻辑组件的业务方法来添加竞价记录，如果竞价成功，程序直接生成竞价成功的响应提示；如果竞价失败，程序生成竞价失败的响应提示。参与竞价的Servlet代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddBidServlet.java

上面的Servlet中粗体字代码调用了业务逻辑方法来添加竞价记录，每次用户参与竞拍就是添加一条竞价记录，也就是向该Servlet发送一个请求。

19.8.5 参与竞价

当用户选择指定物品参与竞价时，系统将会显示该物品的当前详情，例如起拍价格、当前最高竞价等，界面下方提供一个输入框供用户输入竞拍价。

用于竞价的界面布局文件如下。

程序清单：codes\19\AuctionClient\app\src\main\res\layout\add_bid.xml

从上面的界面布局可以看出，当用户选择竞拍物品之后，程序会显示该物品的详情，界面下方包含一个文本框供用户填写竞拍价格。该界面如图19.26所示。

图19.26 参与竞价

当用户在图19.26所示界面中输入竞拍价格，单击“竞价”按钮后，程序将会向/android/addBid.jsp发送请求。添加竞价记录。

参与竞价的Fragment代码如下。

程序清单：
codes\19\AuctionClient\app\src\main\java\org\crazyit\auction\client\AddBidFragment.java

从上面的程序可以看出，当用户单击“竞价”按钮后，程序将先执行输入校验，再调用addBid（）方法来添加竞价记录，如上面程序中的①、②两行粗体字代码所示。

如果用户竞价成功，系统将显示如图19.27所示的对话框。

图19.27 竞价成功

提示：

系统也为兼容手机屏幕提供了AddBid Activity，该Activity的作用也只是包装并显示AddBidFragment，此处不再给出该类的代码。

19.9 权限控制

前面介绍时已经看到，服务器端程序在处理用户登录时，如果用户登录成功，系统会把用户ID放入HTTP session中，方便系统跟踪用户的登录状态。

对于Android客户端程序来说，由于Android客户端采用了Apache HttpClient来发送请求、获取响应，因此HttpClient会自动维护与服务器之间的登录状态。在有效时间之内，服务器端程序可以跟踪到Android客户端的登录状态。

本系统要求只有登录用户才能使用系统功能，因此程序考虑在服务器端使用Filter进行控制，Filter只要拦截匹配的/android/*（匹配该URL的Servlet向Android客户端提供响应）的URL即可。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\Authority.java

正如从上面的粗体字代码所看到的，Filter要求HTTP session中的userId属性不为null，且userId属性大于0，这样就可以判定该用户已经登录系统；否则该Filter不会“放行”请求，而是直接向客户端生成提示信息。

注意：

笔者见过一些与该应用类似的Android应用的权限控制，它们并没有在服务器端进行权限控制，而是只要求用户在第一次使用时登录系统，但实际上这是不够的。如果应用程序不在服务器端进行权限控制，而是只在客户端要求用户登录，那么这种系统的安全控制十分脆弱。

对于绝大部分普通用户来说，这样的安全控制可能没有太大的问题；但对于恶意用户来说，他可以采用多种方法来绕过客户端的登录要求—最简单的方法比如反编译Android应用，让用户在启动程序时立即进

入Main Activity，如果服务器不再进行权限控制，那么整个应用就“赤裸裸”地暴露出来了，这将是一件可怕的事情。所以这里系统不仅要求用户第一次使用时进行登录，而且程序还在服务器端进行了控制，这样才可保证系统的安全性。

19.10 本章小结

本章介绍了一个非常实用的Android应用，Android应用充当电子拍卖系统的客户端，服务器端则采用Struts 2+Spring+Hibernate的技术组合，架构上采用了控制器层、业务逻辑层、DAO层的分层架构，保证整个项目具有极好的可扩展性和可维护性。由于本书是一本介绍Android开发的图书，因此本章并未介绍服务器端的业务逻辑组件、DAO组件的实现，而是重点介绍了Android客户端的实现，包括为Android客户端提供响应的Servlet实现，Android客户端的界面布局，Activity实现等。本章的Android客户端通过Apache HttpClient与服务器交互，服务器与客户端之间采用JSON作为数据交换格式。读者学习本章需要重点掌握Android应用与传统企业应用整合的方式，二者之间通过网络进行数据交换的方式。