



本页面中的内容受版权保护

移动开发系列丛书

eoeAndroid 51CTO

ZD.NET 机锋网 博客园

eoeAndroid开发社区创始人& CTO 姚尚朗

创新工场豌豆荚实验室联合创始人& CTO 冯锋

鼎力推荐



# Google Android 开发权威指南

李宁 编著



6大完整综合案例：新浪微博客户端、蓝牙聊天、全键盘输入法、月球登陆游戏、贪吃蛇游戏、笑脸连连看游戏。

- 本书基于最新的Android 2.3 SDK。
- 分析常用控件和API的源代码，帮助读者更深入地了解其内部实现原理。
- 超过200个例子、50000行代码。大多数实例稍加改动就可用于实际的项目中。
- 开发视频讲解光盘，帮助读者快速、无障碍地学通Android实战开发。
- 从事一线开发的作者提供微博、博客等在线答疑。

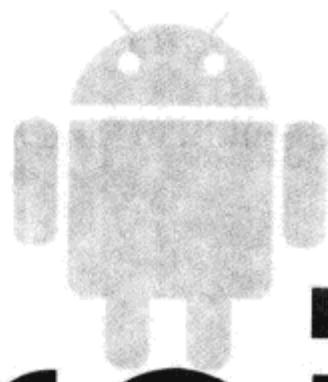


人民邮电出版社  
POSTS & TELECOM PRESS

本页面中的内容受版权保护

范例源程序  
视频文件

■■■ 移动开发系列丛书



# Android 开发权威指南

李宁 编著

人民邮电出版社  
北京 PDG

## 图书在版编目 (C I P) 数据

Android开发权威指南 / 李宁编著. — 北京 : 人民邮电出版社, 2011. 9  
ISBN 978-7-115-25714-7

I. ①A… II. ①李… III. ①移动终端—应用程序—程序设计—指南 IV. ①TN929. 53-62

中国版本图书馆CIP数据核字(2011)第120511号

## 内 容 提 要

本书内容上涵盖了用最新的 Android 版本开发的大部分场景。全书分 4 个部分，分别从 Android 基础介绍、环境搭建、SDK 介绍，到应用剖析、组件介绍、综合实例演示，以及符合潮流的、最新的移动开发技术，如 HTML5、OpenGL ES、NDK 编程、Android 测试驱动开发等几个方面讲述。从技术实现上，讲解了 6 大完整综合案例及源代码分析，分别是新浪微博客户端、蓝牙聊天、全键盘输入法、月球登陆（游戏）、贪吃蛇（游戏）、笑脸连连看（游戏）。

本书注重对实际动手能力的指导，在遵循技术研发知识体系严密性的同时，在容易产生错误、不易理解的环节上配备了翔实的开发情景截图；并将重要的知识点和开发技巧以“多学一招”、“扩展学习”、“技巧点拨”等的活泼形式呈现给读者。在程序实例的讲解方面，主要将实例安插在 Android 开发的精髓知识章节，这为读者学习与实践结合提供了很好的指导。

本书配套光盘包含开发视频及全部源程序，指导读者快速、无障碍地学通 Android 实战开发技术。

本书适合具备一定软件开发经验、想快速进入 Android 开发领域的程序员，具备一些手机开发经验的开发者和 Android 开发爱好者学习使用；也适合作为相关培训学校的 Android 培训教材。

## Android 开发权威指南

- 
- ◆ 编 著 李 宁
  - 责任编辑 张 涛
  - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号
  - 邮编 100061 电子邮件 315@ptpress.com.cn
  - 网址 <http://www.ptpress.com.cn>
  - 大厂聚鑫印刷有限责任公司印刷
  - ◆ 开本：800×1000 1/16
  - 印张：37.75
  - 字数：937 千字 2011 年 9 月第 1 版
  - 印数：1—4 000 册 2011 年 9 月河北第 1 次印刷

---

ISBN 978-7-115-25714-7

定价：79.00 元（附光盘）

读者服务热线：(010) 67132692 印装质量热线：(010) 67129223

反盗版热线：(010) 67171154

广告经营许可证：京崇工商广字第 0021 号

## 名家推荐

《Android 开发权威指南》是一本非常“有营养的书”。无论你是 Android 的初学者，还是专业的 Android 开发人员，你都能从本书中学习到 Android 开发中各种实用的编程知识和技巧。书中新颖和原创实例程序，也可以帮助读者学习和了解到 Android 商业程序开发的实战经验。同时，此书也是一本非常全面的书，如果您是一位专业的 Android 开发人员，它是一本不可多得的好工具书。

——创新工场豌豆荚实验室联合创始人、CTO 冯锋

本书作者李宁是 eoeAndroid 最元老的版主之一，一直活跃在 Android 开发第一线，在热心解答大家技术问题的同时积累了大量有价值的实战经验。很高兴看到作者能把自己第一手的开发经验和学习 Android 过程中的心得体会撰写成书贡献给 Android 的开发者们，帮助更多的 Android 开发者入门并系统性地提高自己。我个人非常乐意把这本书推荐给广大的 Android 开发者。

——eoeAndroid 开发者社区创始人，北京易联致远无线技术有限公司（eoe）  
联合创始人& CTO 姚尚朗（iceskysl）

随着移动互联网的发展，“安卓”已风靡全球，对于正在发展中的国内移动互联网来说，真的很稀缺安卓的人才及相关的开发书籍。这本书很好地满足了读者的需求，引领有志之士走入 Android 开发大门。书中内容设置合理，亮点频现，是不错的 Android 实用学习书籍。

——机锋网 (<http://www.gfan.com>)

在 IT 技术如火如荼的今天，移动开发已经成为了新的技术热点。市场对于移动开发的产品需求、人才需求日益强烈，不少的 IT 人才开始投入到移动开发的行列中。《Android 开发权威指南》正是一本帮助 IT 技术人群逐步了解、掌握并且精通移动开发技术的优秀教材。如果您希望成为一名 Android 开发高手却又不知道从何起步，那么，这本书绝对可以成为您的良师益友。

——51CTO (<http://www.51cto.com>)

本书是一本教你开发赚钱的 3G 互联网应用的书！帮你完全掌握 Android 并进入实战角色的权威指南！

国内基于 Android 的应用程序开发正处于一个蓬勃增长时期，这为软件开发者提供了一个淘金的好机会，也迫使大量此前不熟悉 Android 开发的程序员要迅速进入此领域，以便在经历了单机计算时代和传统互联网时代之后，能在移动互联网时代的元年开始赶上这趟车，成为了许多程序员的愿望。为了帮助读者更好地学习 Android 技术，并把知识应用到实战中，笔者特意创作了本书。

## 本书内容

本书内容全面，不仅详细讲解了 Android 框架、Android 组件、用户界面开发、游戏开发、数据存储、多媒体开发和网络开发等基础知识，而且还深入阐述了传感器、语音识别、桌面组件开发、多媒体开发、OpenGL ES、HTML5、Android NDK 编程、Android 平台测试等高级知识，最重要的是，用 6 大综合案例全面介绍了如何在 Android 平台上开发各种应用。

本书实战性强，书中的每个知识点都配有精心设计的示例，尤为值得一提的是，书中包含的实例和案例涵盖了各种常用的 Android 应用和经典 Android 游戏开发全过程，既可以作为范例进行实战演练，又可以将它们直接应用到实际开发中去，这将很好地帮助初学者尽快融入实战角色。

本书分为 4 部分，大致内容如下。

第一部分 准备篇，包括第 1 章和第 2 章，第 1 章 Android 开发简介、第 2 章 搭建和使用 Android 开发环境。

第二部分 基础篇，包括第 3 章~第 16 章，第 3 章 Android 程序设计基础、第 4 章 用户界面开发基础、第 5 章 控件（Widget）详解、第 6 章 菜单、第 7 章 信息提醒（对话框、Toast 与 Notification）、第 8 章 数据存储、第 9 章 Android 中的窗口——Activity、第 10 章 全局事件——广播（Broadcast）、第 11 章 跨应用数据源 Content Provider、第 12 章 服务（Service）、第 13 章 网络与通讯、第 14 章 多媒体开发、第 15 章 2D 游戏开发、第 16 章 有趣的 Android 应用。

第三部分 高级篇，包括第 17 章~第 22 章，第 17 章 HTML5 与移动 Web 开发、第 18 章 输入法开发、第 19 章 Android OpenGL ES 开发基础、第 20 章 OpenGL ES 的超酷效果、第 21 章 NDK 编程、第 22 章 测试驱动开发（TDD）。

第四部分 综合实例篇，包括第 23 章~第 28 章，第 23 章 蓝牙聊天、第 24 章 月球登陆（游戏）、第 25 章 全键盘输入法（应用）、第 26 章 贪吃蛇（游戏）、第 27 章 新浪微博客户端（应用）、第 28 章 笑脸连连看（游戏）。

## 本书适合我吗

当您走进书店，看到书的标题中熟悉的字眼“Android”，想了解这本书是否适合自己时，下面的提示对您的选购很有帮助：

- 您听说过 Android 吗？
- 您知道 Android Market 吗？
- 您听说许多开发者都在从事移动应用开发了吗？

如果上述问题中有一个以上是肯定的，可以很高兴地告诉您，拿在手中的这本书确实是这个方向上的，下面需要进一步确认一下：

- 您对软件开发有经验或者有兴趣吗？
- 您对 Java 语言有了解吗？
- 您做过手机应用开发吗？
- 您是 Android 爱好者吗？

如果上述问题，您的回答中有肯定的，那么您已经具备了阅读本书需要的基础，不用担心读不懂了，那么：

- 您想快速了解并进入 Android 应用开发吗？
- 您想找到一本系统介绍 Android 开发的参考资料吗？
- 您想选择一本有原理剖析又有真实例子演示的教材吗？
- 您想选一本通俗易懂，符合自己阅读习惯的图书吗？

如果如上问题中，您有大多数回答都是肯定的，那么非常恭喜您，现在拿着的这本书差不多正是您需要的，可以放心地带回去开始自己的 Android 之旅了。

如果还在犹豫，那么让下面几个提示告诉您，尽早开始学习的重要性：

- Android 是一个非常强大的手机平台，其让你可以快速切入无线互联网领域，赢取高薪职位；
- 在 Android Market 发布应用的数量在快速增长，早日发布可以体现自己的开发价值和乐趣；

- 掌握了 Android 开发就可以很快开发出供全球 Android 用户使用的应用，有人已经在 Android Market 赚到钱了！

## 本书特色

- 本书基于最新的 Android 2.3 SDK，所有的例子可以在 Android 2.3 SDK 的环境中运行。
  - 内容覆盖了 Android 开发的大部分场景，从 Android 开发环境搭建、SDK，到 Android 应用剖析、常见控件介绍、实例演示等方面。
  - 本书分析了部分常用控件和 API 的源代码，以使读者能更深刻地了解其内部实现原理。
  - 超过 200 个例子、50000 行代码。
  - 技术实现上，包括了新浪微博客户端、蓝牙聊天、全键盘输入法、月球登陆游戏、贪吃蛇游戏、笑脸连连看游戏 6 大综合实战案例。
  - 本书配有开发视频光盘，帮助读者快速、无障碍地学通 Android 实战开发。

## 本书约定

本书遵循如下约定。

符号和术语	含    义	示    例
技巧点拨	告诉读者开发的捷径	技巧点拨 单击 Eclipse 的“Window”>“Preferences”菜单项，打开“Preferences”对话框，在左侧找到 Android 节点，单击“Build”子节点，在右侧的“Build”设置页中的“Default debug keystore”文本框的值就是 debug.keystore 文件的路径
多学一招	对所讲知识的拓展	多学一招：通过手势打电话 每成功识别一个手势，就会返回与其对应的手势信息。其中包括了定义手势时输入的名称。我们也可以利用这个名称来调用其他的应用程序或做任何事件。例如，下面的代码通过手势名来调用拨打电话的程序

续表

符号和术语	含    义	示    例
扩展学习	表示对上面讲解知识的延展	扩展学习：与联系人关联的其他方式 除了使用 QuickContactBadge.assignContact Uri 方法将 QuickContactBadge 与联系人关联外，还可以使用如下两个方法与联系人关联
源程序解释	对一段代码进行功能和技术的总结性说明	编写上面代码时应注意如下几点： <ul style="list-style-type: none"><li>• 如果不知道接收到的广播包含哪些数据，可以从 Bundle.keySet()方法中获得这些数据的 key</li><li>• 由于接收到的短信内容是以字节数组形式保存</li><li>• 在最新的 Android SDK 版本中提供了新的 SmsMessage 类</li></ul>
工程目录	实例工程的演示内容	工程目录：src\ch19\ch19_rotate_triangle 本节将为读者展示第一个使用 OpenGL ES 制作的动画。这个动画是一个顺时针旋转的三角形，效果如图 19.7 所示
注意	对一个技术应用特意的提醒	注意： <i>在 Android 2.3 中不能像老版本的 Android 一样使用 IntBuffer.wrap(new int[]{...}) 初始化 IntBuffer 对象 (FloatBuffer 也是一样)，否则，系统会抛出异常。应直接使用 allocate Direct 方法为缓冲区分配空间，再使用 ByteBuffer.put 方法将原始数据放到缓冲区中</i>

## 本书适合

- 具备一定软件开发经验，想快速进入 Android 开发领域的程序员。
- 已经掌握 Android 的基础知识，想进一步提高 Android 开发技能的程序员。
- 为了个人兴趣和职业技能储备而想要学习 Android 的爱好者。
- 作为各大培训机构的培训教材。
- 想将本书作为枕边书，随时查阅 Android 的各种知识和技术的读者。

## 如何阅读本书

本书从基础入手，循序渐进地讲述了 Android 的主要功能和用法，使读者对其有完整的认识，掌握其结构框架。同时，从实战的角度出发，通过大量的示例程序和综合案例，让读者边学习边实践，更深刻地理解 Android 系统。

本书的大多数示例工程都是基于 Android 2.3 的（API Level = 9），因此，建议读者在导入本书实例之前将 Android SDK 和 ADT 升级到最新版本（安装和升级 Android SDK 和 ADT 的方法见 2.1.4 节和 2.1.5 节的内容）。当然，如果读者不想升级成最新的 Android SDK 和 ADT，本书的大多数实例仍可以运行在 Android 2.1 和 Android 2.2 两个版本上。而要做的只是修改 `AndroidManifest.xml` 文件中`<uses-sdk>`标签的 `android:minSdkVersion` 属性值，例如，该属性值为 7，表示可以运行在 Android 2.1 及以上版本的 Android SDK 上。如果不只想限定 Android SDK 版本，可以直接将`<uses-sdk>`标签删除。

本书的实例代码按每章进行组织。每一章的实例代码放在相应的目录中。例如，第 2 章的代码会放在 `src\ch02` 目录中。而且每个包含实例代码的章节开始部分都会给出示例代码的具体位置。图 1 和图 2 是源代码结构图。其中第 4 部分的综合案例源代码都放在 `demo` 目录中。



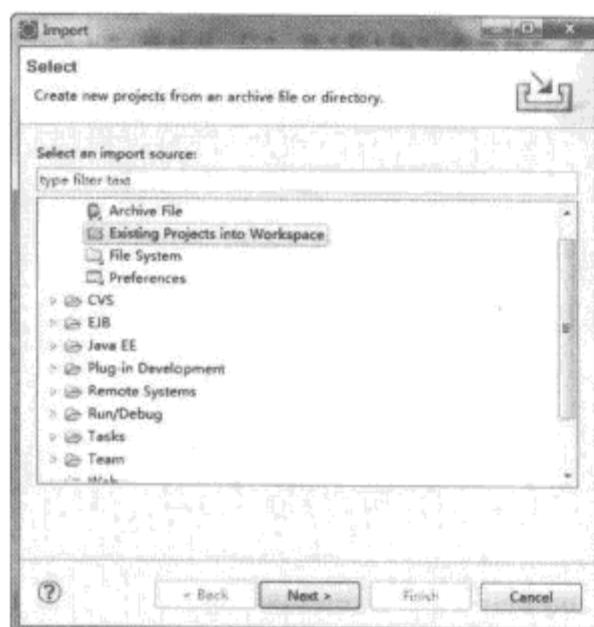
▲ 图 1 本书全部源代码的结构图



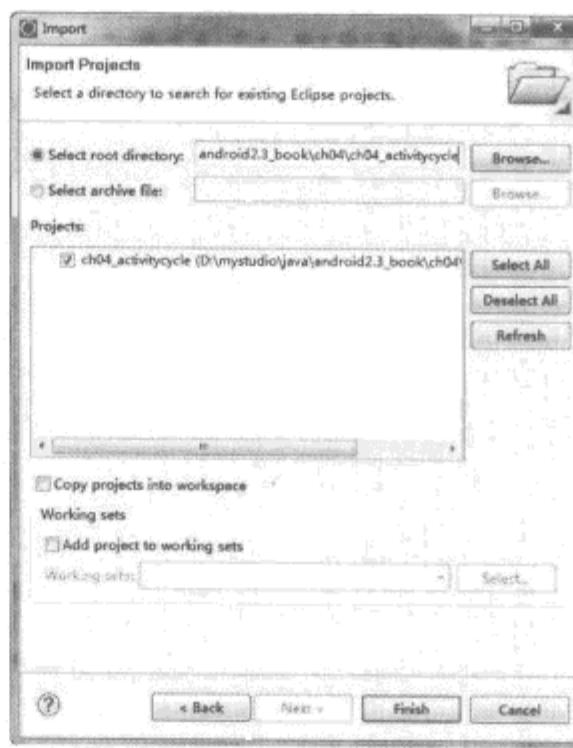
▲ 图 2 第 8 章的源代码结构图

下面是演示导入实例工程。首先启动 Eclipse，单击“File”>“Import”菜单项，打开“Import”对话框。选择“Existing Projects into Workspace”节点。如图 3 所示。

单击“Next”按钮进入下一步设置。单击“Browse”按钮选择要导入的实例工程，选择后的效果如图 4 所示。当选定实例工程后，单击“Finish”按钮开始导入实例工程。



▲ 图 3 打开“Import”对话框



▲ 图 4 导入实例工程

本书答疑网址为：<http://books.51happyblog.com>。作者 Blog 为：<http://blog.csdn.net/nokiaguy>，新浪微博账号：<http://weibo.com/androidguy>，电子邮箱：techcast@126.com。

## 本书的社区支持联盟

作为一本 Android 原创技术书籍，我们会充分吸取 Android 社区反馈的意见和建议，积极修订和完善，力求使得本书能更符合开发者的阅读习惯和使用要求，感谢这些为 Android 发展做出贡献的社区，感谢其对本书的支持，社区列举如下，排名不分先后。



eoeAndroid



ZD.NET



博客园  
cnblogs.com



51CTO



机锋网

## 第一部分 准备篇

<b>第1章 初识庐山真面目——Android 开发简介</b>	2
1.1 Android 的基本概念	2
1.1.1 Android 简介	2
1.1.2 Android 的版本	4
1.1.3 Android 的系统构架	5
1.1.4 Android 的应用程序框架	6
1.2 JIL Widget 介绍	7
1.3 小结	8
<b>第2章 工欲善其事，必先利其器——搭建和使用Android 开发环境</b>	9
2.1 开发包及工具的安装	9
2.1.1 开发 Android 程序都需要些什么	9
2.1.2 安装 JDK 和配置 Java 开发环境	10
2.1.3 Eclipse 的安装与汉化	11
2.1.4 安装 Android SDK	12
2.1.5 安装 Eclipse 插件 ADT	13

2.2 真实体验——编写第一个 Android 程序（随机绘制圆饼）	15
2.2.1 创建 Android 工程	15
2.2.2 在模拟器中运行 Android 程序	17
2.2.3 界面控件的布局	19
2.2.4 编写代码	20
2.2.5 调试程序	23
2.2.6 在手机上运行和调试程序	28
2.3 迁移 Android 工程可能发生的错误	29
2.4 不需要写一行代码的开发工具：AppInventor	31
2.4.1 AppInventor 简介	31
2.4.2 AppInventor 的下载和安装	31
2.4.3 用拖曳控件的方式设计界面	33
2.4.4 像拼图一样拼装代码	34
2.5 小结	37

## 第二部分 基础篇

<b>第3章 千里之行始于足下——Android 程序设计基础</b>	40
3.1 Android 应用程序框架	40

3.1.1 Android 项目的目录结构	40
3.1.2 AndroidManifest.xml 文件的结构	42

3.2	Android 应用程序中的 资源 .....	44	4.3.3	使用剪切板传递 数据 .....	63
3.3	Android 的应用程序 组件 .....	44	4.3.4	使用全局对象传递 数据 .....	65
3.3.1	Activity (Android 的窗体) .....	45	4.3.5	返回数据到前一个 Activity .....	67
3.3.2	Service (服务) .....	45	4.4	视图 (View) .....	68
3.3.3	Broadcast Receiver (广播接收器) .....	45	4.4.1	视图简介 .....	68
3.3.4	Content Provider (内容提供者) .....	46	4.4.2	使用 XML 布局文件 定义视图 .....	69
3.4	Android 程序的 UI 设计 .....	46	4.4.3	在代码中控制视图 .....	70
3.4.1	手工配置 XML 布局文件 .....	46	4.5	布局 (Layout) .....	72
3.4.2	ADT 自带的可视化 UI 设计器 .....	47	4.5.1	框架布局 (FrameLayout) .....	72
3.4.3	使用 DroidDraw 设计 UI 布局 .....	47	4.5.2	线性布局 (LinearLayout) .....	75
3.5	小结 .....	48	4.5.3	相对布局 (RelativeLayout) .....	79
<b>第 4 章</b>	<b>我的 UI 我做主—— 用户界面开发基础</b> .....	<b>49</b>	4.5.4	表格布局 (TableLayout) .....	81
4.1	Activity 的使用方法 .....	49	4.5.5	绝对布局 (AbsoluteLayout) .....	82
4.1.1	创建 Activity .....	49	4.5.6	重用 XML 布局 文件 .....	82
4.1.2	配置 Activity .....	50	4.5.7	优化 XML 布局 文件 .....	85
4.1.3	显示其他的 Activity (Intent 与 Activity) .....	52	4.5.8	查看 apk 文件中的 布局 .....	86
4.2	Activity 的生命周期 .....	55	4.6	小结 .....	87
4.3	在不同 Activity 之间 传递数据 .....	60	<b>第 5 章</b>	<b>良好的学习开端—— 控件 (Widget) 详解</b> .....	<b>88</b>
4.3.1	使用 Intent 传递 数据 .....	60	5.1	常用 XML 属性解析 .....	88
4.3.2	使用静态变量传 递数据 .....	62	5.1.1	android:id 属性 .....	88

5.1.3 android:layout_margin 属性	89
5.1.4 android:padding 属性	89
5.1.5 android:layout_weight 属性	90
5.1.6 android:layout_gravity 和 android:gravity 属性	90
5.1.7 android:visibility 属性	91
5.1.8 android:background 属性	91
5.1.9 指定单击事件方法 ( android:onClick 属性)	92
5.1.10 控件焦点属性 ( android:focusable 和 android:focusableInTouchMode )	92
5.2 TextView (显示文本的控件)	93
5.2.1 显示富文本 ( URL、不同大小、字体、颜色的文本 )	93
5.2.2 在 TextView 中显示表情图像和文字	97
5.2.3 单击链接弹出 Activity	100
5.2.4 为指定文本添加背景	103
5.2.5 带边框的 TextView	106
5.2.6 设置行间距	110
5.2.7 在未显示完的文本后面加省略号 ( ... )	111
5.2.8 用 TextView 实现走马灯效果	113
5.2.9 垂直滚动 TextView 中的文本	114
5.3 EditText (编辑文本的控件)	115
5.3.1 像 QQ 一样输入表情图像	115
5.3.2 在 EditText 中输入特定的字符	117
5.3.3 AutoCompleteTextView ( 自动完成输入内容的控件 )	118
5.4 按钮和复选框控件	120
5.4.1 Button ( 普通按钮控件 )	120
5.4.2 图文混排的按钮	122
5.4.3 ImageButton ( 图像按钮控件 )	124
5.4.4 RadioButton ( 选项按钮控件 )	124
5.4.5 ToggleButton ( 开关状态按钮控件 )	125
5.4.6 CheckBox ( 复选框控件 )	126
5.5 ImageView ( 显示图像的控件 )	128
5.5.1 ImageView 控件的基本用法	128
5.5.2 显示指定区域的图像	129
5.5.3 缩放和旋转图像	132
5.6 时间与日期控件	133
5.6.1 DatePicker ( 输入日期的控件 )	133
5.6.2 TimePicker ( 输入时间的控件 )	134

5.6.3	DatePicker、 TimePicker 与 TextView 同步 显示日期和时间	134
5.6.4	AnalogClock 和 DigitalClock (显示 时钟的控件)	136
5.7	进度条控件	137
5.7.1	ProgressBar (进度 条控件)	137
5.7.2	SeekBar (拖动 条控件)	139
5.7.3	设置 ProgressBar 和 SeekBar 的颜色 及背景图	140
5.7.4	RatingBar (评分 控件)	143
5.8	列表控件	145
5.8.1	ListView (普通 列表控件)	145
5.8.2	为 ListView 列表 项添加复选框 和选项按钮	147
5.8.3	对列表项进行增、 删、改操作	149
5.8.4	改变列表项的 背景色	153
5.8.5	ListActivity (封装 ListView 的 Activity)	154
5.8.6	ExpandableListView (可扩展的列表 控件)	155
5.8.7	Spinner (下拉 列表控件)	157
5.9	滚动控件	160
5.9.1	ScrollView (垂直 滚动控件)	160
5.9.2	HorizontalScrollView (水平滚动控件)	161
5.9.3	可垂直和水平 滚动的视图	162
5.9.4	Gallery (画廊 控件)	163
5.10	ImageSwitcher (图像 切换控件)	164
5.11	GridView (网格 控件)	166
5.12	TabHost (标签 控件)	168
5.13	ViewStub (惰性装载 控件)	169
5.14	小结	171
<b>第 6 章 友好的菜单——</b>		
	<b>Menu 介绍与实例</b>	172
6.1	菜单的基本用法	172
6.1.1	创建选项菜单 (Options Menu)	172
6.1.2	带图像的选项 菜单	173
6.1.3	关联 Activity	173
6.1.4	响应菜单的单击 动作	174
6.1.5	动态添加、修改和 删除选项菜单	175
6.1.6	带复选框和选项 按钮的子菜单	176
6.1.7	上下文菜单	178
6.1.8	菜单事件	179
6.1.9	从菜单资源中 装载菜单	180
6.2	菜单特效	181

6.2.1	自定义菜单 .....	181	7.3.3	用 PopupWindow 模拟 Toast 提示 信息框 .....	221
6.2.2	模拟 UCWeb 效果菜单 .....	184	7.4	通知 (Notification) .....	221
6.2.3	QuickContactBadge 与联系人菜单 .....	189	7.4.1	在状态栏上显示 通知信息 .....	222
6.3	小结 .....	192	7.4.2	Notification 的清除 动作 .....	224
<b>第 7 章</b>	<b>友好地互动交流—— 信息提醒 (对话框、 Toast 与 Notification) .....</b>	<b>193</b>	7.4.3	永久存在的 Notification .....	225
7.1	对话框的基本用法 .....	193	7.4.4	自定义 Notification .....	226
7.1.1	带 2 个按钮 (确认/ 取消) 的对话框 .....	193	7.5	小结 .....	227
7.1.2	带 3 个按钮 (覆盖/ 忽略/取消) 的对话框 .....	195	<b>第 8 章</b>	<b>移动的信息仓库—— 数据存储 .....</b>	<b>228</b>
7.1.3	简单列表对话框 .....	196	8.1	读写 key-value 对: SharedPreferences .....	228
7.1.4	单选列表对话框 .....	198	8.1.1	SharedPreferences 的基本用法 .....	228
7.1.5	多选列表对话框 .....	199	8.1.2	数据的存储位置 和格式 .....	229
7.1.6	进度对话框 .....	201	8.1.3	存取复杂类型 的数据 .....	230
7.1.7	登录对话框 .....	205	8.1.4	设置数据文件 的访问权限 .....	233
7.1.8	使用 Activity 托管 对话框 .....	207	8.1.5	可以保存设置的 Activity: PreferenceActivity .....	234
7.2	对话框的高级应用 .....	209	8.2	文件存储 .....	238
7.2.1	阻止单击按钮 关闭对话框 .....	209	8.2.1	openFileOutput 和 openFileInput 方法 .....	238
7.2.2	改变对话框的 显示位置 .....	213	8.2.2	读写 SD 卡中 的文件 .....	239
7.2.3	在对话框按钮和 内容文本中插入 图像 .....	215	8.2.3	SAX 引擎读取 XML 文件的原理 .....	241
7.2.4	改变对话框的 透明度 .....	216			
7.3	Toast .....	217			
7.3.1	Toast 的基本用法 .....	217			
7.3.2	永不关闭的 Toast .....	218			

8.2.4 将 XML 文件 转换成 Java 对象	242	9.2 自定义 Activity Action	264
8.2.5 文件压缩 ( Jar、Zip )	245	9.3 Activity 的高级应用	266
8.3 SQLite 数据库	249	9.3.1 ActivityGroup	266
8.3.1 SQLite 数据库 管理工具	249	9.3.2 自定义半透明 窗口	268
8.3.2 SQLiteOpenHelper 类与自动升级 数据库	251	9.3.3 Activity 之间切换 的动画效果	269
8.3.3 数据绑定与 SimpleCursor Adapter 类	252	9.4 小结	270
8.3.4 操作 SD 卡上的 数据库	255	<b>第 10 章 全局事件——广播 ( Broadcast )</b>	271
8.3.5 将数据库与应用 程序一起发布	256	10.1 什么是广播	271
8.3.6 内存数据库	257	10.2 接收系统广播	272
8.4 小结	258	10.2.1 短信拦截	272
<b>第 9 章 Android 中的窗口—— Activity</b>	259	10.2.2 用代码注册广播 接收器	274
9.1 调用其他程序中的 Activity	259	10.2.3 广播接收器的 优先级	275
9.1.1 直接拨号	259	10.2.4 来去电拦截	276
9.1.2 将电话号传入 拨号程序	259	10.2.5 截获屏幕休眠 与唤醒	280
9.1.3 调用拨号程序	260	10.2.6 开机自动运行	281
9.1.4 浏览网页	261	10.2.7 显示手机电池的 当前电量	282
9.1.5 向 E-mail 客户端 传递 E-mail 地址	261	10.3 发送广播	284
9.1.6 发送 E-mail	261	10.4 验证广播接收器是 否注册	285
9.1.7 查看联系人	262	10.5 小结	286
9.1.8 显示系统设置界面 ( 设置主界面、Wifi 设置界面 )	263	<b>第 11 章 跨应用数据源—— Content Provider</b>	287
9.1.9 启动处理音频的 程序	264	11.1 Content Provider 的 作用	287
		11.2 获得系统数据	288
		11.2.1 读取联系人信息	288
		11.2.2 查看收到的 短信	290

11.3 自定义 Content Provider	291	13.1.2 用 WebView 控件装载 HTML 代码	322
11.3.1 查询城市信息	291	13.2 访问 HTTP 资源	324
11.3.2 为 Content Provider 添加访问权限	297	13.2.1 提交 HTTP GET 和 HTTP POST 请求	324
11.4 小结	298	13.2.2 HttpURLConnection 类	326
<b>第 12 章 一切为用户服务——Service 基础与实例</b>	<b>299</b>	13.2.3 上传文件	327
12.1 Service 基础	299	13.3 客户端 Socket	330
12.1.1 Service 的生命周期	299	13.3.1 连接服务器	331
12.1.2 绑定 Activity 和 Service	302	13.3.2 扫描服务器打开的端口	331
12.1.3 开机启动 Service	305	13.3.3 发送和接收数据	333
12.1.4 判断 Service 是否已注册	306	13.3.4 获得无线路由分配给手机的 IP 地址	334
12.1.5 判断 Service 是否已开始	307	13.3.5 设置 Socket 选项	335
12.2 跨进程访问 (AIDL 服务)	308	13.4 服务端 Socket	339
12.2.1 什么是 AIDL 服务	308	13.4.1 手机服务器的实现	339
12.2.2 建立 AIDL 服务的步骤	308	13.4.2 利用 Socket 在应用程序之间通信	340
12.2.3 建立 AIDL 服务	308	13.5 蓝牙通信	342
12.2.4 传递复杂数据的 AIDL 服务	312	13.5.1 蓝牙简介	342
12.2.5 AIDL 与来电自动挂断	317	13.5.2 打开和关闭蓝牙设备	343
12.3 小结	319	13.5.3 搜索蓝牙设备	344
<b>第 13 章 做好应用桥梁——网络与通信</b>	<b>320</b>	13.5.4 蓝牙数据传输	346
13.1 WebView 控件	320	13.6 小结	351
13.1.1 用 WebView 控件浏览网页	320	<b>第 14 章 炫酷你的应用——多媒体开发</b>	<b>352</b>
		14.1 音乐	352
		14.1.1 播放音乐	352

14.1.2 录音	353	15.4.1 AnimationDrawable 与帧动画	392
14.2 视频	354	15.4.2 播放 Gif 动画	394
14.2.1 使用 VideoView 播放视频	354	15.5 补间 (Tween) 动画	397
14.2.2 使用 SurfaceView 播放视频	355	15.5.1 移动补间动画	397
14.2.3 录制视频	357	15.5.2 缩放补间动画	399
14.3 相机	357	15.5.3 旋转补间动画	402
14.3.1 调用系统的拍照 功能	358	15.5.4 透明度补间动画	403
14.3.2 自定义拍照功能	359	15.6 小结	404
14.4 铃声	364	第 16 章 有趣的 Android 应用	405
14.5 小结	366	16.1 传感器	405
<b>第 15 章 2D 游戏开发</b>	<b>367</b>	16.1.1 如何使用传感器	405
15.1 绘制游戏的画布	367	16.1.2 加速度传感器 (Accelerometer)	409
15.1.1 在 View 上实现 动画效果	367	16.1.3 重力传感器 (Gravity)	409
15.1.2 在 SurfaceView 上实现动画效果	371	16.1.4 光线传感器 (Light)	410
15.2 图形绘制基础	374	16.1.5 陀螺仪传感器 (Gyroscope)	411
15.2.1 绘制像素点	374	16.1.6 方向传感器 (Orientation)	411
15.2.2 绘制直线	374	16.1.7 其他传感器	412
15.2.3 绘制圆形	375	16.2 输入输出技术	413
15.2.4 绘制弧	375	16.2.1 语音识别	413
15.2.5 绘制文本	376	16.2.2 手势输入	415
15.2.6 综合绘制各种 图形	376	16.2.3 语音朗读 (TTS)	417
15.3 高级图像处理技术	380	16.3 Google 地图	419
15.3.1 绘制位图	380	16.4 GPS 定位	423
15.3.2 图像的透明度	382	16.5 桌面上的小东西	425
15.3.3 旋转图像	383	16.5.1 窗口小部件 (AppWidget)	425
15.3.4 路径	384	16.5.2 快捷方式	431
15.3.5 Shader 的渲染 效果	388	16.5.3 实时文件夹	433
15.4 帧 (Frame) 动画	392	16.6 应用更华丽—— 动态壁纸	435
		16.7 小结	441

## 第三部分 高 级 篇

<b>第 17 章 HTML5 与移动 Web 开发</b>	444	<b>18.3.2 编写输入法程序</b>	463
17.1 HTML5 简介	444	<b>18.3.3 输入法服务的生命周期</b>	466
17.2 HTML5 精彩效果演示	445	<b>18.3.4 预输入文本</b>	467
17.3 HTML5 在 Android 中的应用	447	<b>18.3.5 输入法设置</b>	467
17.4 HTML5 的画布 (Canvas)	448	<b>18.4 小结</b>	468
17.4.1 Canvas 概述	448	<b>第 19 章 Android OpenGL ES 开发基础</b>	469
17.4.2 检测浏览器是否支持 Canvas	449	19.1 OpenGL 简介	469
17.4.3 在 Web 页面中使用 Canvas	450	19.2 什么是 OpenGL ES	469
17.4.4 使用路径 (Path)	452	19.3 多边形	470
17.4.5 设置线条风格	453	19.4 颜色	474
17.4.6 设置填充类型	453	19.5 旋转三角形	475
17.4.7 填充矩形区域	454	19.6 旋转立方体	477
17.4.8 使用渐变色 (Gradient)	454	19.7 小结	478
17.4.9 拉伸画布对象	455	<b>第 20 章 OpenGL ES 的超酷效果</b>	479
17.4.10 在 Canvas 上绘制文本	456	20.1 保持平衡的旋转文本	479
17.4.11 使用阴影	457	20.2 左右摇摆的 Android 机器人	482
17.5 调试 JavaScript	458	20.3 纠缠在一起的旋转立方体	485
17.6 小结	459	20.4 透明背景的旋转立方体	486
<b>第 18 章 输入法开发</b>	460	20.5 触摸旋转的立方体	487
18.1 Android 输入法简介	460	20.6 2D 和 3D 的综合旋转效果	489
18.2 控制输入法	461	20.7 旋转立体天空	491
18.3 输入法实战	462	20.8 小结	493
18.3.1 实现输入法的步骤	463	<b>第 21 章 Android NDK 编程</b>	494
		21.1 Android NDK 简介	494

<b>21.2 安装、配置和测试</b>	<b>21.6 使用 NDK 调用</b>
NDK 开发环境 ..... 495	音频 API ..... 510
<b>21.2.1 系统和软件要求</b> ..... 495	<b>21.7 本地 Activity</b>
<b>21.2.2 下载和安装</b>	(Native Activity) ..... 511
Android NDK ..... 495	<b>21.8 Android NDK 配置</b>
<b>21.2.3 下载和安装</b>	文件详解 ..... 513
Cygwin ..... 495	<b>21.8.1 Android NDK</b>
<b>21.2.4 配置 Android NDK</b>	定义的变量 ..... 513
的开发环境 ..... 498	<b>21.8.2 Android NDK</b>
<b>21.3 第一个 NDK 程序：</b>	定义的函数 ..... 514
世界你好 ..... 499	<b>21.8.3 描述模块的变量</b> ..... 515
<b>21.3.1 编写和调用 NDK</b>	<b>21.8.4 配置 Application.mk</b>
程序 ..... 499	文件 ..... 516
<b>21.3.2 用命令行方式编译</b>	<b>21.9 小结</b> ..... 516
NDK 程序 ..... 501	<b>第 22 章 测试驱动开发</b>
<b>21.3.3 在 Eclipse 中集成</b>	(TDD) ..... 517
Android NDK ..... 502	<b>22.1 JUnit 测试框架</b> ..... 517
<b>21.4 背景不断变化的三角形</b>	<b>22.2 测试 Activity</b> ..... 517
(NDK 版	<b>22.3 测试 Content Provider</b> ..... 521
OpenGL ES) ..... 504	<b>22.4 测试 Service</b> ..... 523
<b>21.5 使用 NDK OpenGL ES</b>	<b>22.5 测试普通类</b> ..... 523
API 实现千变万化	<b>22.6 小结</b> ..... 525
的 3D 效果 ..... 507	

## 第四部分 综合实例篇

<b>第 23 章 Android 综合案例一——</b>	<b>23.6 小结</b> ..... 537
<b>蓝牙聊天</b> ..... 528	<b>第 24 章 Android 综合案例二——</b>
<b>23.1 蓝牙聊天主界面</b> ..... 528	<b>月球登陆（游戏）</b> ..... 538
<b>23.2 添加选项菜单</b> ..... 530	<b>24.1 游戏的玩法</b> ..... 538
<b>23.3 搜索和连接蓝牙</b>	<b>24.2 实现游戏界面</b> ..... 539
设备 ..... 530	<b>24.3 设置游戏难度</b> ..... 540
<b>23.4 使设备可被其他</b>	<b>24.4 开始游戏</b> ..... 541
蓝牙设备发现	<b>24.5 控制飞船喷火</b> ..... 544
<b>23.5 发送和接收聊天</b>	<b>24.6 控制飞船改变飞行</b>
信息 ..... 533	方向 ..... 544

24.7 判断飞船是否成功着陆	545	27.2 使用新浪微博开发 API	563																										
24.8 小结	546	27.3 创建和配置新浪微博客户端工程	564																										
<b>第 25 章 Android 综合案例三——全键盘输入法（应用）</b>	<b>547</b>	27.4 登录新浪微博	564																										
25.1 安装输入法	547	27.5 功能按钮	567																										
25.2 输入法的初始化工作	548	27.6 显示“我的首页”的微博	569																										
25.3 响应键盘操作	549	27.7 评论微博	572																										
25.4 根据 EditText 控件的属性显示不同的软键盘	551	27.8 转发微博	573																										
25.5 小结	553	27.9 写微博	574																										
<b>第 26 章 Android 综合案例四——贪吃蛇（游戏）</b>	<b>554</b>	27.10 小结	576																										
26.1 游戏玩法	554	<b>第 28 章 Android 综合案例六——笑脸连连看（游戏）</b>	<b>577</b>																										
26.2 游戏主界面设计	555	26.3 控制小蛇的移动	557	28.1 游戏玩法	577	26.4 小结	560	28.2 准备图像素材	578	<b>第 27 章 Android 综合案例五——新浪微博客户端（应用）</b>	<b>561</b>	28.3 实现主界面	578	27.1 新浪微博简介	561	28.4 随机生成连连看图像	580	27.1.1 新浪微博客户端	561	28.5 选中两个相同图像后消失	580	27.1.2 新浪微博开放 API	563	28.6 用定时器限制游戏时间	581			28.7 小结	582
26.3 控制小蛇的移动	557	28.1 游戏玩法	577																										
26.4 小结	560	28.2 准备图像素材	578																										
<b>第 27 章 Android 综合案例五——新浪微博客户端（应用）</b>	<b>561</b>	28.3 实现主界面	578																										
27.1 新浪微博简介	561	28.4 随机生成连连看图像	580																										
27.1.1 新浪微博客户端	561	28.5 选中两个相同图像后消失	580																										
27.1.2 新浪微博开放 API	563	28.6 用定时器限制游戏时间	581																										
		28.7 小结	582																										





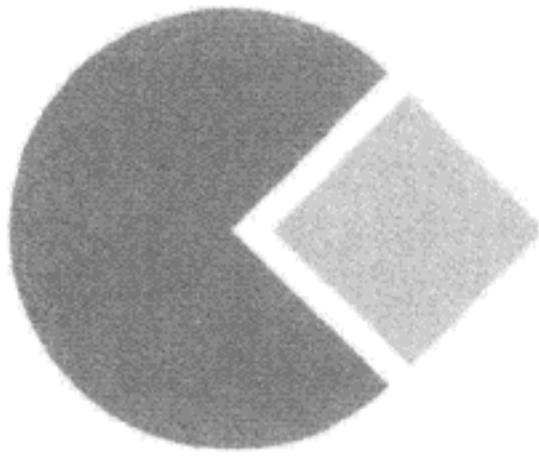
# 第一部分

## 准 备 篇

第 1 章 初识庐山真面目——  
Android 开发简介

第 2 章 工欲善其事，必先利其器——  
搭建和使用 Android 开发环境





# 第1章 初识庐山真面目—— Android 开发简介

自从 Google 公司在 2005 年收购了成立仅 22 个月的 Android 公司以来，在 Google 公司以及其他软硬件企业的不断推动下，Android 以其迅猛的发展速度成为了目前最流行的智能手机操作系统。

现在让我们先来回顾一下历史。世界上第一部装有 Android 系统的手机于 2008 年 9 月 23 日发布，这部手机被称为 T-Mobile G1。G1 的发布也标志着 Android 正式加入了智能手机操作系统的激烈竞争。Google 为了争取主动权，不惜重金举办 Android 开发者大赛，以便吸引更多开发者的目光，同时也可以使更多的优秀 Android 应用脱颖而出。Android 在最近几年的火热程度让很多国内外企业看到了 Android 的光明前途。不仅国外的众多企业加入了以 Google 为主导的开发手机联盟，而且国内的很多企业，包括中国移动、联想、魅族等，也加入了 Android 的大家庭。尤其值得一提的是中国移动推出了国内第一个基于 Android 的系统（OMS）。不久，联想也推出了基于 OMS 的第一款手机 O1。而这一切都发生在 G1 推出后的一年时间内。

2010 年是 Android 手机蓬勃发展的一年。在这一年里上市的 Android、OMS 等系统的手机种类多达数百款。然而，这也许只是个开始，在 2011 年，随着加入 Android 阵营的手机厂商不断增多，从事 Android 开发的程序员也在以几何级地增长着。Android 的前景将会变得更加光明。但不管是 Android 光明的前途，还是“钱”途，都需要千里之行，始于足下。对于那些还徘徊在 Android 大门之外的程序员，还在等什么呢？让 Android 成为你们要学习的下一种新技术的首选吧！当您读完这段时，说明您已经到达了 Android 大门之外，下面要做的一件事就是继续阅读本书，并让本书帮您打开这扇通往 Android 的圣殿之门，让我们一起开始 Android 之旅吧！

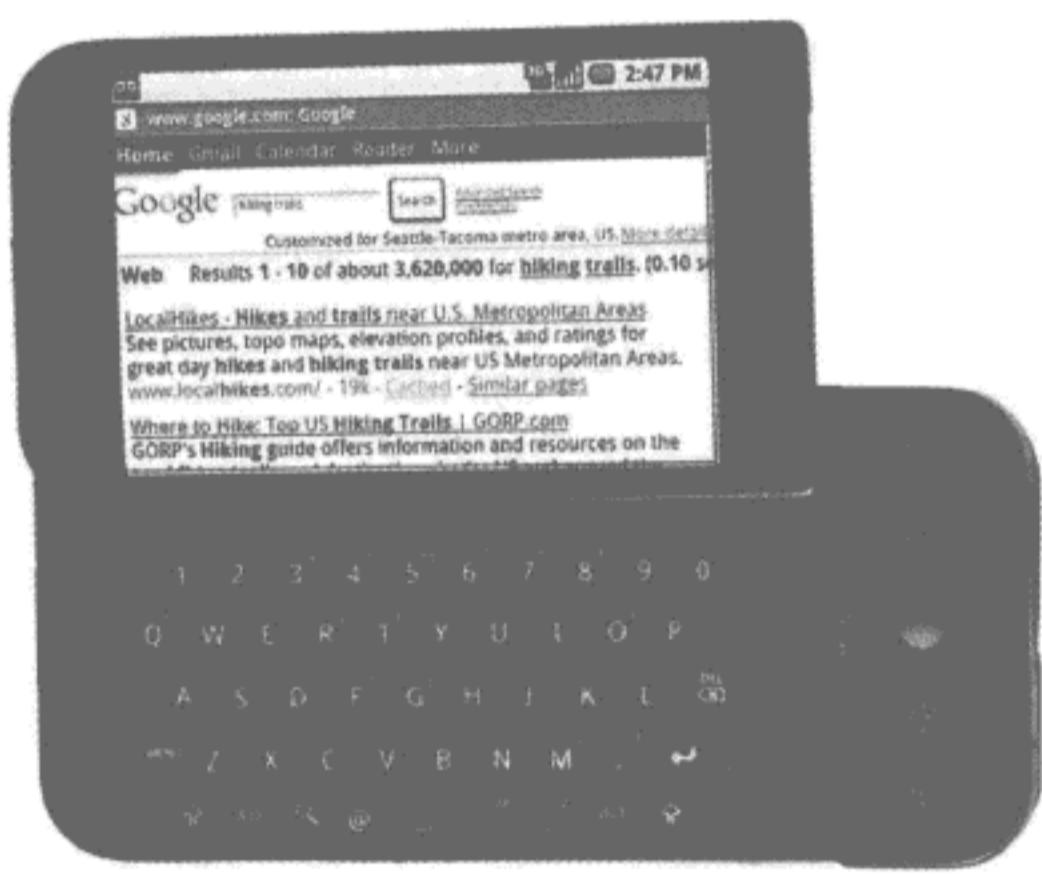
## 1.1 Android 的基本概念

Android 的中文意思是“机器人”。但在移动领域，大家一定会将 Android 与 Google 联系起来。Android 本身就是一个操作系统，只是这个操作系统是基于 Linux 内核的。Android 是一个由 30 多家科技公司和手机公司组成的“开放手机联盟”共同研发的，而且完全免费开源，这将大大降低手机设备的研发成本，甚至 Android 已成了“山寨”手机的首选操作系统。

### 1.1.1 Android 简介

Android 作为 Google 最具创新的产品之一，正受到越来越多的手机厂商、软件厂商、运营商及个人开发者的追捧。目前 Android 阵营主要包括 HTC（宏达电）、T-Mobile、高通、三星、LG、摩

托罗拉、ARM、软银移动、中国移动和华为等。虽然这些企业有着不同的性质，但它们都在 Android 平台的基础上不断创新，让用户体验到最优质的服务。尤其要提一下的是中国移动与播思公司联合研发的基于 Android 的 OMS 系统已取得了不俗的业绩。下面欣赏几款具有代表性的 Android 手机。第一款毫无疑问，就是世界上第一部 Android 手机 T-Mobile G1，如图 1.1 所示。这款手机带有一个物理键盘（硬键盘），可以通过侧划拉出。第二款则是继 G3 之后又一款创下不俗销售业绩的 HTC Desire，也称为 G7，如图 1.2 所示。G7 目前搭载了 Android 2.2 系统，仍然采用了 HTC Sense 界面，并支持移动热点。最后一款则是被称为 Android 机皇的三星 Galaxy S，也称为 I9000，如图 1.3 所示。这款手机配置了高达 1GHz 的 CPU 和 512MB 的内存。尽管 G7 也配置了 1GHz 的 CPU，但 I9000 配置的是较为先进的 Cortex A8 架构的 CPU，并且内置了 PowerVR 540 显示核心（硬件图形加速，玩 3D 游戏更流畅），在性能上要优于 G7 的 Snapdragon 处理器。而且更值得一提的是 I9000 内置了 8GB 的 ROM（相当于 PC 的硬盘），而 G7 内置的 ROM 只有 512MB。因此，I9000 可以比 G7 安装更多的应用程序，甚至可以不需要扩展 SD 卡。



▲ 图 1.1 T-Mobile G1



▲ 图 1.2 HTC G7

欣赏完这么多“超酷”的手机，现在来看一下 Android 到底有什么魔力，可以让众多的粉丝为之疯狂。据粗略统计，Android 至少有如下 8 项制胜“法宝”。

- **开放性。**Android 平台是免费、开源的。而且 Google 通过与运营商、设备制造商、开发商等机构形成了战略联盟，希望通过共同制定标准使 Android 成为一个开放式的生态系统。
- **应用程序的权限由开发人员决定。**编写过 Symbian、Java ME 程序的读者应该能体会到这些程序在发布时有多麻烦。如果访问到某些限制级的 API，不是出现各种各样的提示，就是根本无法运行。要想取消这些限制，就得向第三方的认证机构购买签名，而且价格不菲。而使用 Android 平台的应用程序就相对幸福得多。要使用限制级的 API，只需要在自己的应用程序中配置一下即可，完全是 DIY。这也在某种程度上降低了 Android 程序的开发成本。



▲ 图 1.3 三星 Galaxy S I9000

- 我的平台我作主。Android 上的所有应用程序都是可替换和扩展的，即使是拨号、Home 这样的核心组件也是一样。只要我们有足够的想象力，就可以缔造出一个独一无二、完全属于自己的 Android 世界。

- 应用程序之间的无障碍沟通。各个应用程序之间的通信一直令人头痛，而在 Android 平台上无疑是一种享受。在 Android 平台上的应用程序之间至少有 4 种通信方式。很难说哪一种方式更好，但它们的确托起了整个 Android 的应用程序框架。

- 拥抱 Web 时代。如果想在 Android 应用程序中嵌入 HTML、JavaScript，那真是再容易不过了。基于 Webkit 内核的 WebView 组件会完成一切。更值得一提的是，JavaScript 还可以和 Java 无缝地整合在一起。

- 物理键盘和虚拟键盘双管齐下。从 Android 1.5 开始，Android 同时支持物理键盘和虚拟键盘，从而可大大丰富用户的输入选择。尤其是虚拟键盘，已成为 Android 手机中主要的输入方式。

- 个性的充分体现。21 世纪是崇尚个性的时代。Android 也紧随时代潮流提供了众多体现个性的功能。例如，Widget、Shortcut、Live Wallpapers，无一不尽显手机的华丽与时尚。

- 舒适的开发环境。Android 的主流开发环境是 Eclipse + ADT + Android SDK。它们可以非常容易地集成到一起，而且在开发环境中运行程序要比 Symbian 这样的传统手机操作系统更快，调试更方便。

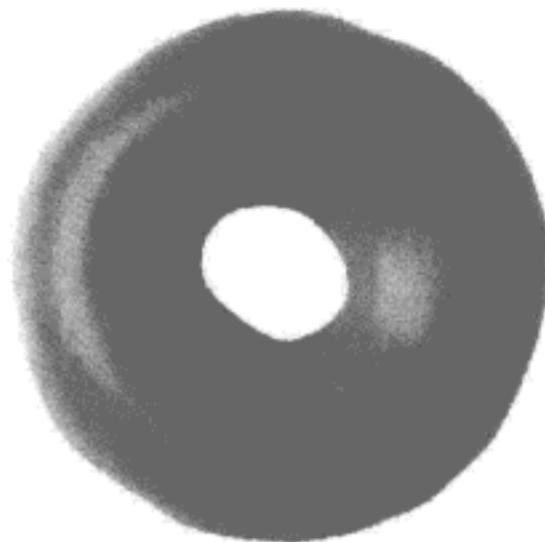
虽然 Android 的特点还有很多，但那已经不重要。重要的是现在 Android 已经成为万众瞩目的国际“巨星”，它的未来将令人充满期望。

### 1.1.2 Android 的版本

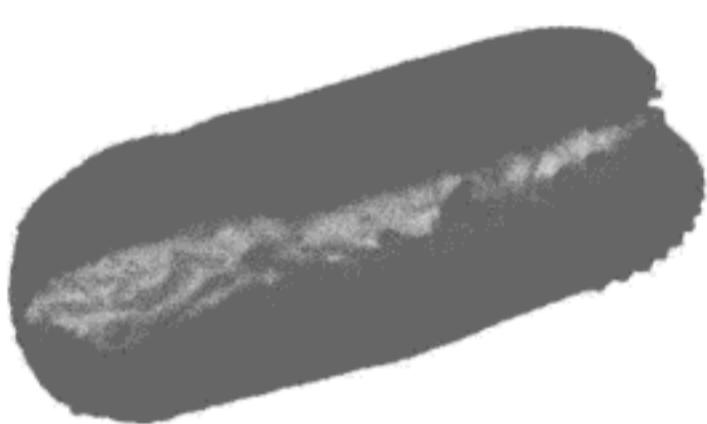
自从 Google 公司在 2007 年 11 月 5 日发布 Android 的第一个版本（Android 1.0）以来，Android 共经过了 9 次大的版本升级。除了最初发布的两个版本（Android 1.0 和 Android 1.1）由于不稳定而没有用甜点来命名外，其他的主要版本都使用了相应的甜点来命名，图 1.4 至图 1.8 所示是几款主要 Android 版本与其对应的甜点（建议在用餐之前阅读本小节的内容，可以大大增加食欲）。



▲ 图 1.4 Android 1.5 (纸杯蛋糕, Cupcake)



▲ 图 1.5 Android 1.6 (甜甜圈, Donut)



▲ 图 1.6 Android 2.0/2.1 (乳酪馅指形小饼, Eclair)



▲ 图 1.7 Android 2.2 (冻酸奶, Froyo)



▲ 图 1.8 Android 2.3 (姜饼, Gingerbread)

### 1.1.3 Android 的系统构架

通过前面的介绍，我们对 Android 的特点已经有了一个初步的了解。本节将介绍 Android 的系统构架。先来看看 Android 的体系结构，如图 1.9 所示。



▲ 图 1.9 Android 体系结构图

从图 1.9 可以看出 Android 分为 4 层，从高到低分别是应用层、应用框架层、系统运行库层和 Linux 内核层。下面对这 4 层进行简单的介绍。

- **应用层。**该层由运行在 Dalvik 虚拟机(Google 公司为 Android 专门设计的基于寄存器的 Java 虚拟机，运行 Java 程序的速度比 JVM 更快)上的应用程序（主要由 Java 语言编写）组成，例如，日历、地图、浏览器、联系人管理等。
- **应用框架层。**该层主要由 View、通知管理器(Notification Manager)、活动管理器(Activity Manager)等由开发人员直接调用的 API 组成(这些 API 主要也由 Java 语言编写)。
- **系统运行库层。**Java 本身是不能直接访问硬件的，要想让 Java 访问硬件，必须使用 NDK 才可以。NDK 是一些由 C/C++语言编写的库(主要是\*.so 文件)。这些程序也是该层的主要组成部分。该层主要包括 C 语言标准库、多媒体库、OpenGL ES、SQLite、Webkit、Dalvik 虚拟机等。也就是说，该层是对应用框架层提供支持的层。
- **Linux 内核层。**该层主要包括驱动、内存管理、进程管理、网络协议栈等组件。目前 Android 的版本基于 Linux 2.6 内核。

#### 1.1.4 Android 的应用程序框架

在开发大多数的 Android 应用程序时，通常直接与应用框架层进行交互，而应用框架层则负责与底层进行交互。因此，了解应用框架层的结构对开发 Android 应用程序至关重要。

应用程序框架实际上就是我们使用的 Android SDK 中的 Java 类、接口的集合。下面来看看 Android 的最新版本中包含了哪些主要的功能(具有相同功能的 Java 类、接口通常放到一个 package 中)。

- **android.app:** 提供高层的程序模型和基本的运行环境。
- **android.appwidget:** 包含了创建 Widget 的相关类，Widget 可以放在 Android 的桌面上。
- **android.bluetooth:** 提供了操作蓝牙设备的相关类。

- android.content: 提供了对各种设备上的数据进行访问和发布的相关类和接口。
- android.database: 提供了操作数据库的相关类和接口。
- android.gesture: 提供了手势操作的相关类和接口。
- android.graphics: 底层的图形库。主要包括画布、颜色过滤、点、矩形，可以将它们直接绘制到屏幕上。
- android.hardware: 操作硬件的库。由于不同的手机硬件不同，因此，这些库在不同的手机上不一定有效。
- android.inputmethodservice: 通过这个包中的接口和类，可以编写基于 Android 的输入法程序。
- android.location: 提供了与定位和相关服务的类和接口。
- android.media: 提供了与管理音频和视频相关的类和接口。
- android.net: 提供了与网络访问相关的类和接口。
- android.opengl: 提供 OpenGL 的工具。
- android.os: 提供了系统服务、消息传输和 IPC 机制。
- android.provider: 提供了与访问 Android 内容提供者相关的类和接口。
- android.sax: 用于访问 XML。
- android.speech: 用于文本转语音的库（Text To Speech）。
- android.telephony: 提供与拨打电话相关的交互。
- android.test: 用于测试 Android 应用程序的框架。
- android.util: 一些实用的工具，例如处理时间的类。
- android.view: 提供基础的用户界面接口框架。
- android.webkit: 默认的浏览器操作接口。
- android.widget: 包括了 Android SDK 提供的大部分 UI 控件。

## 1.2 JIL Widget 介绍

上一节已经知道了 OPhone 平台带了一个 BAE 引擎，在该引擎上运行的程序叫 JIL Widget。虽然在 OPhone 平台上的 BAE 主要运行的是 JIL Widget，但目前 BAE 也能兼容很多互联网上流行的 Widget，例如，App Dashboard Widget。

那么 JIL Widget 到底是一种什么样的程序呢？直接地说，就是 HTML+CSS+JavaScript+Webkit 的解决方案。可能很多读者会感到奇怪，怎么没找到 Java。这个问题算问到点上了。编写 JIL Widget 程序根本不需要 Java 语言。也就是说，JIL Widget 是一个完全用 Web 技术写的应用程序，所不同的是这个应用程序会自动调用基于 Webkit 内核的浏览器来运行。

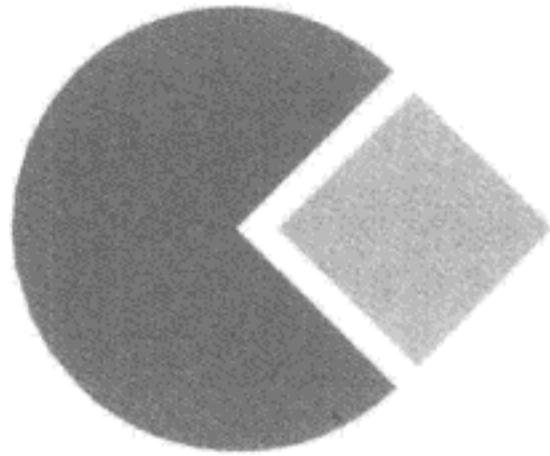
虽然 JIL Widget 应用程序由 HTML、CSS 和 JavaScript 写成，但可以通过 JavaScript 核心扩展模块中的 API 访问 OPhone 中的 API，例如，获得系统信息、播放视频、文件存储等。

## 1.3 小结

本章介绍了 Android 的基本概念。在写作本书时，Android 的最新版本是 Android 2.3 (Level = 9)，从 Android 1.0 算起，Android 已经过了 9 个 API 层次的变化。从 Android 2.0 (Level = 5) 开始，Android 无论在性能上，还是在稳定性上都已趋于成熟。到 Android 2.2 时，Android 开始支持很多非常酷的功能，例如，可以将 Android 手机变成一台无线路由器。

随着 Android 的不断升温，国内外也涌现出很多 Android 的衍生版本，例如，中国移动的 OPhone (OMS) 就是其中之一。OPhone 目前最新的版本是 OPhone 2.0，国内目前已经几十款手机安装了 OPhone 1.5 和 OPhone 2.0。

尽管目前无论是 Android 手机，还是 OPhone 手机，或是其他类似的手机，前景都显得很光明。但还是那句话：千里之行，始于足下。无论做 Android 程序多么有前途，都要先学会如何编写 Android 程序，而本书的内容几乎涵盖了 Android 的方方面面，希望广大的读者能通过对本书的学习使自己的 Android 技术达到更高的水平。



## 第2章 工欲善其事，必先利其器—— 搭建和使用 Android 开发环境

本章讲解了如何配置 Android 的开发环境，这也是学习 Android 开发的第一步，毕竟，除了那些顶级的技术大师，很少有人会尝试用记事本来编写较复杂的 Android 程序。在配置完 Android 的开发环境后，也许正在看本书的读者和作者一样，急于展示一下我们的成果。这也就进入了本章的第二个主题：在我们亲手配置的 Android 开发环境中开发第一个 Android 应用程序。虽然这个程序并不复杂，但仍然要编写少量的代码。可能有的读者第一次接触 Android，对 Android 的编程思想和架构还不太了解，而对于这个程序的理解可能也是云里雾里。不过别着急，还有更好、更妙的方法介绍给读者，而且妙就妙在不需要编写一行代码就可以完成一个十分有趣的 Android 应用程序，这就是 Google 推出的 App Inventor。虽然 App Inventor 与 Eclipse 开发环境比起来还只是个“玩具”，但足以使完全不懂 Android、甚至是 Java 的读者开发出非常有趣的 Android 应用程序。尽管本书仍然使用 Eclipse 开发环境进行讲解，但在没有完全接触 Android 编程之前，就可以像搭积木一样搭出一个有模有样的 Android 应用程序，实在是一件很有趣的事，这也会大大增强读者学习 Android 的信心。

### 2.1 开发包及工具的安装

虽然 Android 开发环境使用起来较为舒适，但要想使开发环境正常地工作，就要安装一大堆软件、SDK 和插件。不过读者不要被这些东西吓住了，虽然初次搭建 Android 开发环境比较费时费力，但凡是安装过这些东西的开发人员就会发现，安装过程并没有想象中复杂，而且本节对 Android 开发环境的搭建做了非常全面的阐述，读者只要按图索骥，便可水到渠成。

#### 2.1.1 开发 Android 程序都需要些什么

开发 Android 程序至少需要如下工具和开发包：

- JDK（1.6 及以上版本）；
- Eclipse（3.4 或以上版本）；
- Android SDK；
- ADT（Android Development Tools，开发 Android 程序的 Eclipse 插件）。

## 2.1.2 安装 JDK 和配置 Java 开发环境

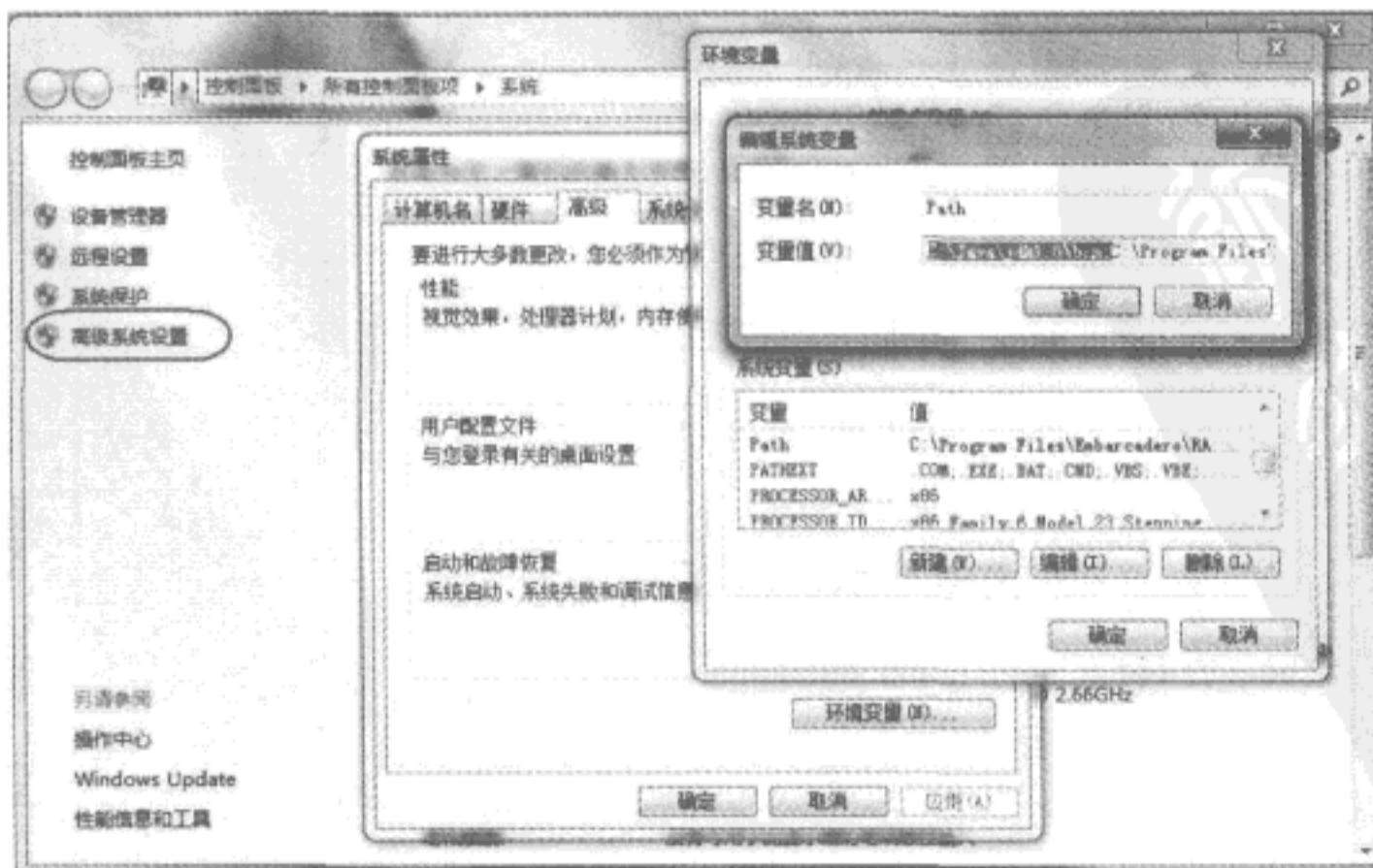
安装 JDK 是学习编写 Java 的各种程序（Java SE、Java EE、Java ME 以及本书所讲的 Android）的基础。很多 Java 的初学者总是不能很好地配置 Java 的开发环境，从而对进一步学习 Java 产生了畏惧心理。实际上，安装和配置 Java 的开发环境要比搭建 Android 开发环境简单得多。

安装和配置 Java 开发环境可按如下步骤进行。

(1) 从 <http://www.oracle.com/technetwork/java/javase/downloads/index.html> 网页下载最新版的 JDK。自从 Sun 被 Oracle 收购后，几乎所有曾经指向 sun.com 的 URL 都指向了 oracle.com。各位读者在访问以前的下载地址时要注意这一点。当然，如果读者不知道下载地址也不要紧，只要在 Google 的搜索框中输入“JDK download”，第一项就是 JDK 的下载地址。

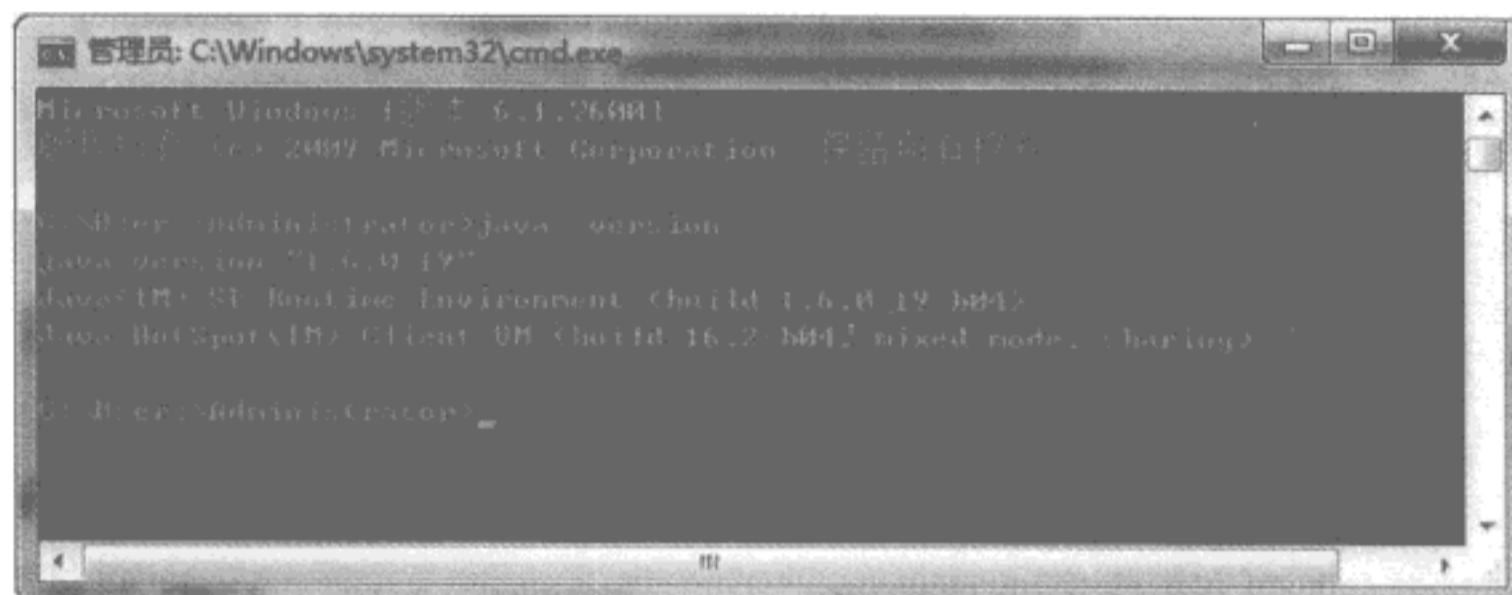
(2) 安装 JDK。如果读者下载的是 JDK，那么安装包中会包含 JDK 和 JRE 两部分。笔者建议将它们安装在同一个逻辑盘中。双击安装程序，选择安装的目录，单击“下一步”按钮，等待安装程序自动完成安装即可。

(3) 设置 Java 环境变量。右键单击“我的电脑”，选择“属性”菜单项，单击弹出窗口左上角的“高级系统设置”。在弹出的界面中选择“高级”选项卡，单击“环境变量”按钮。在弹出的“环境变量”对话框中的“系统变量”列表中找“Path”变量。如果找到，双击该变量显示设置对话框；如果没找到该变量，新建一个“Path”变量。并在变量设置对话框中设置 JDK 中 bin 目录的路径，如图 2.1 所示。按照同样的方法设置“JAVA\_HOME”变量，该变量的值指向 JDK 的根目录（如果未设置该变量，Eclipse 会由于找不到 JDK 而无法启动）。除了这两个变量外，还有一个“CLASSPATH”变量。这个变量指向多个全局搜索的目录或 jar 文件。在使用 java 和 javac 命令运行和编译程序时，如果未通过命令行指定所引用的类库，系统会在“CLASSPATH”变量指向的目录或 jar 文件中搜索相应的类库。当然，如果只使用 JDK 中标准的类库，并不需要设置该变量。注意：笔者写作本书时使用的是 Windows 7。如果是其他操作系统，如 Windows XP，设置过程略有差异，但基本的流程是一样的。



▲ 图 2.1 设置“Path”环境变量 (Windows 7)

在安装配置完成后，启动Windows控制台。输入`java -version`命令，如果输出类似图2.2所示的信息，说明我们已经成功安装和配置了Java的开发环境。



▲图2.2 输出JDK的版本信息

### 2.1.3 Eclipse的安装与汉化

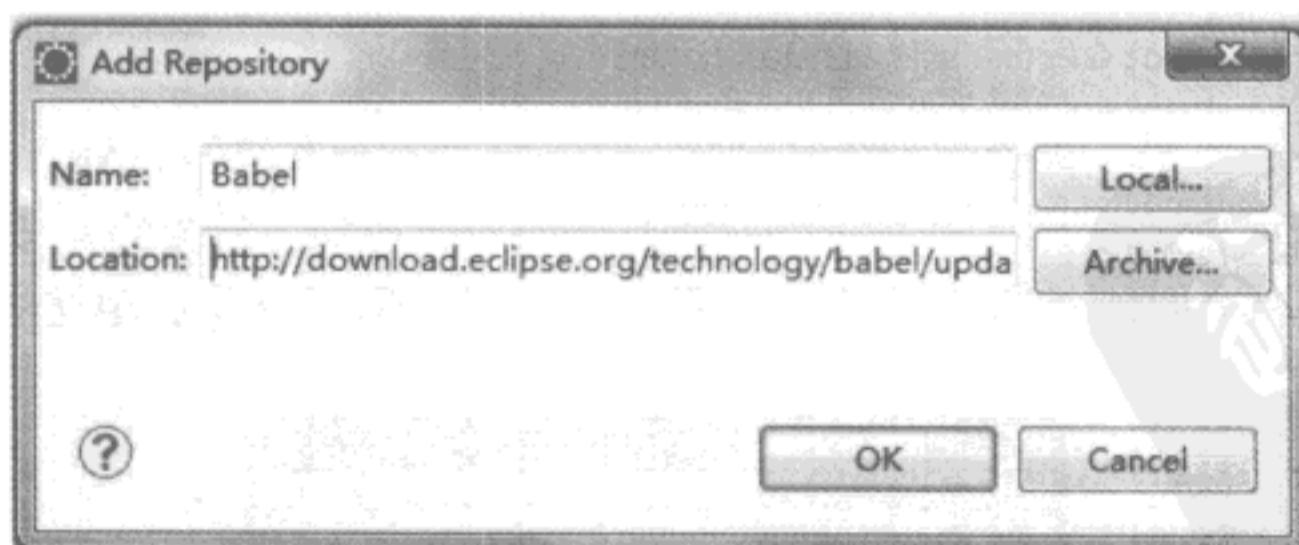
Eclipse的安装比较简单。只要安装完了JDK，从<http://www.eclipse.org/downloads>下载最新的Eclipse版本，解压后，运行根目录的`eclipse.exe`文件即可。

众所周知，Eclipse默认是英文界面。如果读者不喜欢或对英文界面不习惯，可以使用Eclipse发布的一个Babel项目。该项目包含了各个国家的语言包。使用该项目可以对Eclipse进行汉化。该项目的网址如下：

**■ [http://www.eclipse.org/projects/project\\_summary.php?projectid=technology.babel](http://www.eclipse.org/projects/project_summary.php?projectid=technology.babel)**

启动Eclipse，选中“Help”>“Install New Software”菜单项，单击右上角的“Add”按钮，在弹出的对话框中输入Babel和如下的网址，输入效果如图2.3所示。

**■ <http://download.eclipse.org/technology/babel/update-site/R0.8.1/helios1>**

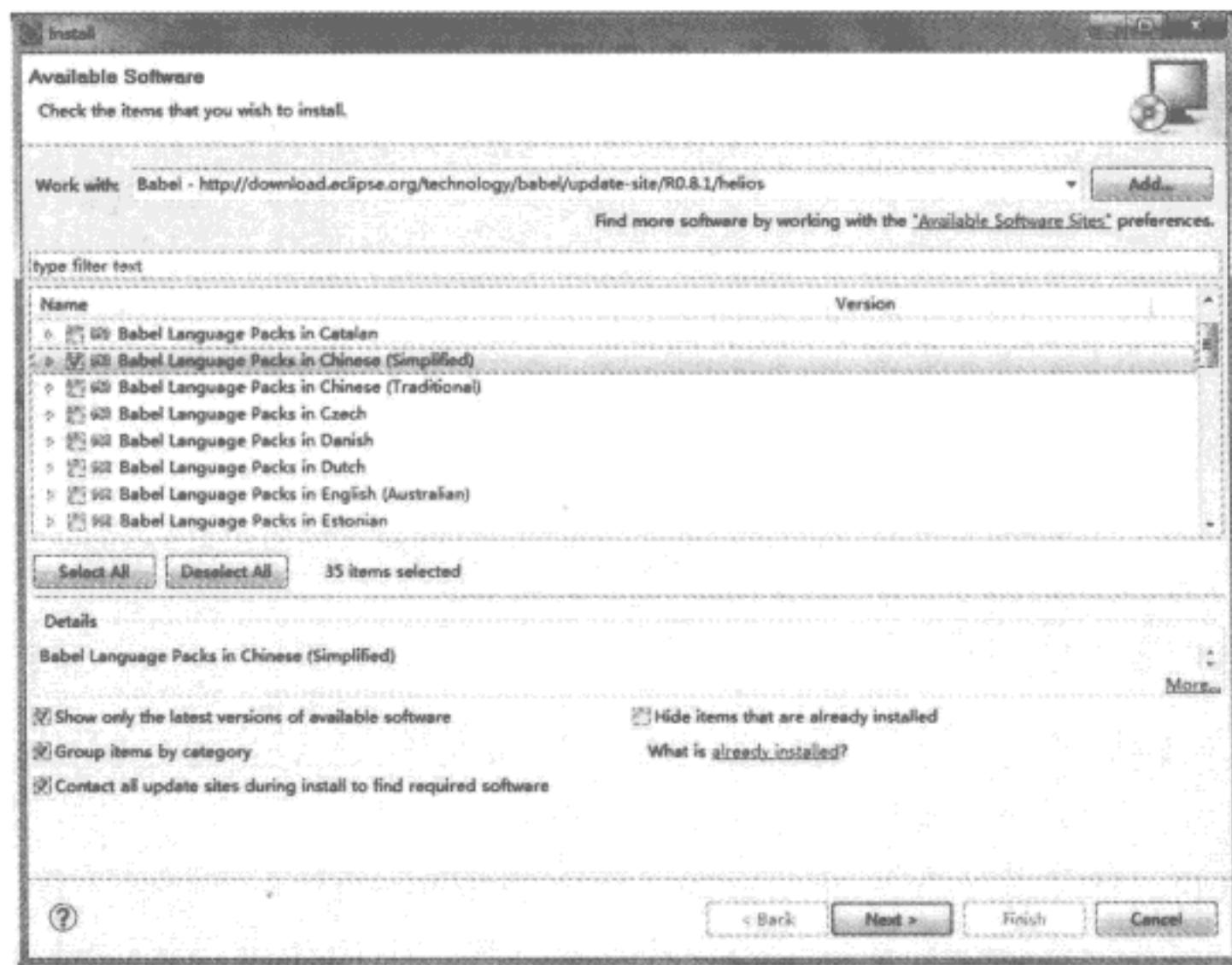


▲图2.3 输入Babel下载地址

在作者写作本书时，Eclipse的最新版本是Eclipse 3.6（太阳神，helios），如果读者使用的是其他Eclipse版本，需要从其他地址安装Babel，下面的网址是Babel项目的下载页面。

**■ <http://www.eclipse.org/babel/downloads.php>**

在输入完 Babel 下载地址后，在“Install”对话框的列表中会出现很多种语言包，选中简体中文语言包，如图 2.4 所示。



▲ 图 2.4 选择中文语言包

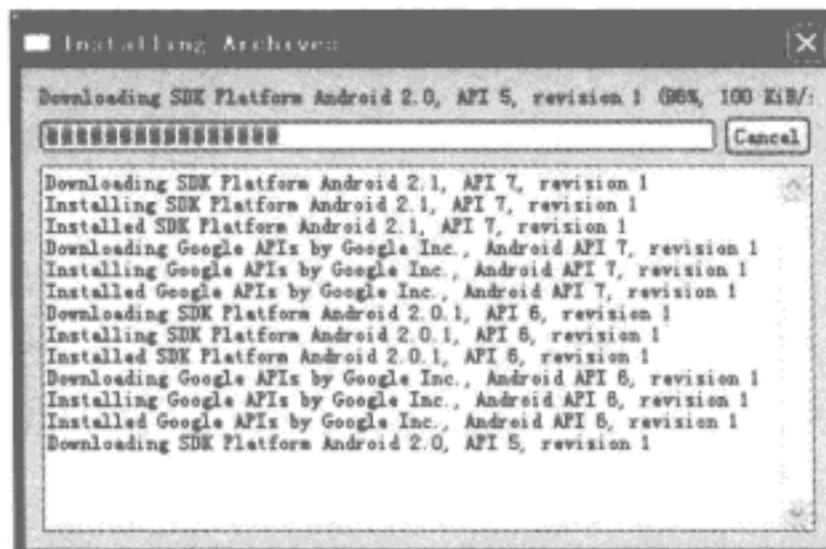
然后单击“Next”按钮，按照提示操作，最后单击“Finish”按钮即可完成 Babel 的安装。安装完 Babel 后，重启 Eclipse 即可完成汉化。

#### 2.1.4 安装 Android SDK

读者可以从下面的地址下载 Android SDK 的最新版本：

<http://developer.android.com/sdk/index.html>

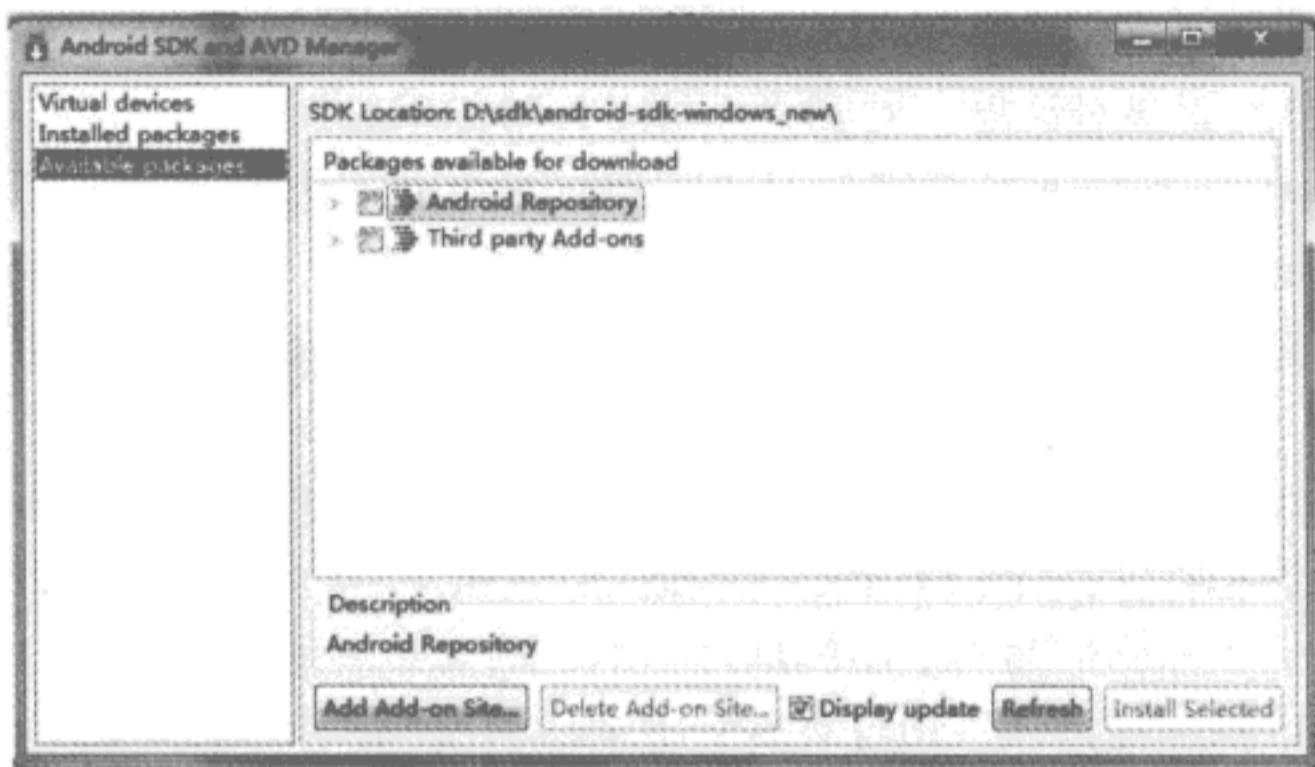
该版本可以同时安装 8 个 Android SDK 版本（1.1 至 2.3）。要注意，Android SDK 是在线安装。在安装 Android SDK 之前，要保证有稳定而快速的 Internet 连接。如果完全安装 Android SDK，安装时间会比较长，读者需要耐心等待。如果安装过程顺利，将会出现如图 2.5 所示的下载界面。



▲ 图 2.5 安装过程的下载界面

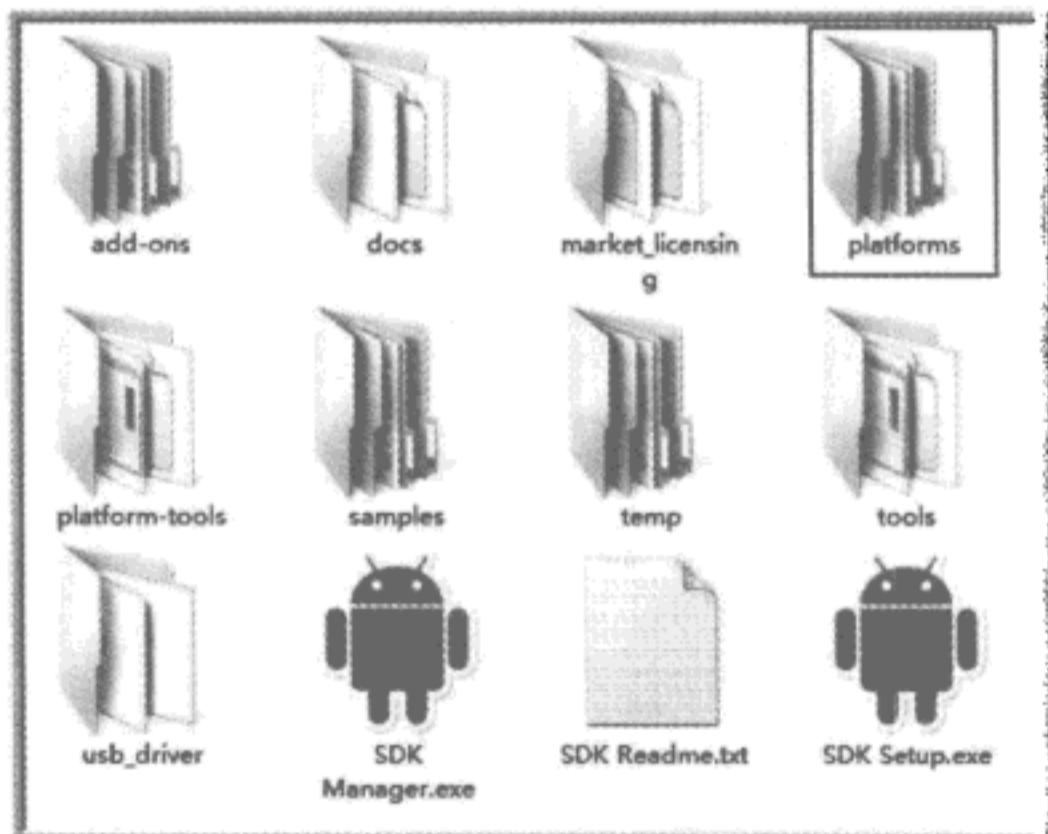
## 第2章 工欲善其事，必先利其器——搭建和使用Android开发环境

如果在读者的机器上已经安装了旧版本的Android SDK和ADT，可以在Eclipse中打开“Android SDK and AVD Manager”对话框，并选中左侧的“Available packages”列表项，然后单击右下角的“Refresh”按钮。如果当前已安装的Android版本不是最新的，会在如图2.6所示的列表中显示相应的安装项，选中相应的安装项，单击右下角的“Install Selected”按钮开始安装最新的Android版本。

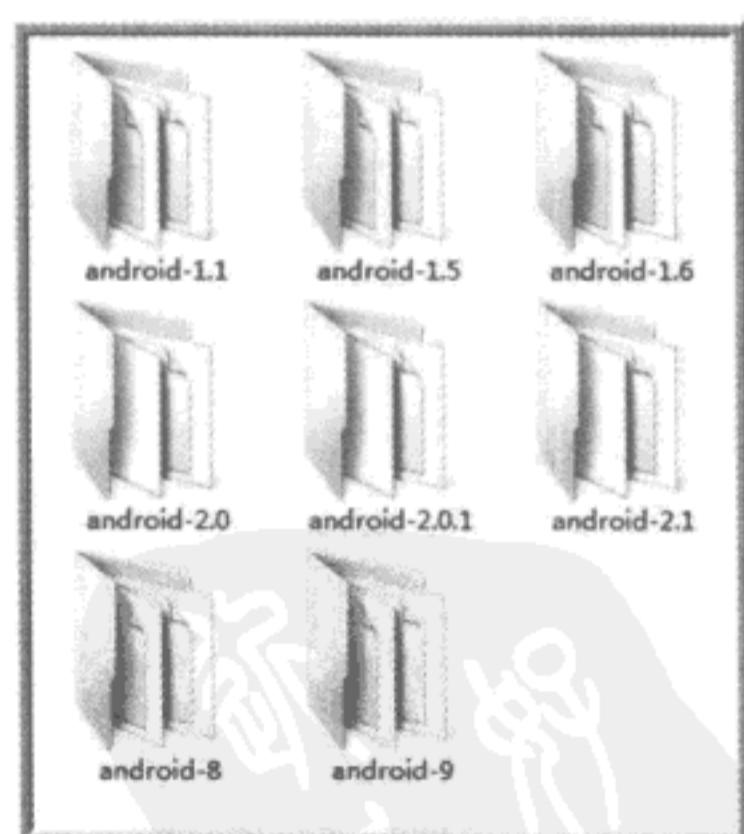


▲图2.6 安装最新的Android版本

Android SDK安装成功后，会看到如图2.7所示的Android SDK根目录结构。platforms目录包含当前Android SDK支持的所有版本，如图2.8所示。



▲图2.7 Android SDK根目录结构



▲图2.8 已经安装的所有Android SDK版本



在笔者写作本书时，Android的最新版本是2.3。在这个版本中将很多命令行程序（如adb.exe、aapt.exe）放在了platform-tools目录中，而不是原来的tools目录。

### 2.1.5 安装Eclipse插件ADT

在笔者写作本书时，ADT的最新版本是8.0.1。该版本必须下载最新的Android SDK才能使用。

读者可以在 Eclipse 中直接安装 ADT。

如果读者使用的是 Eclipse 3.4 (Ganymede), 选中“Help”>“Software Updates...”菜单项。在弹出的对话框中单击“Available Software”选项卡，然后单击“Add Site...”按钮，在弹出的对话框的文本框中输入如下地址：

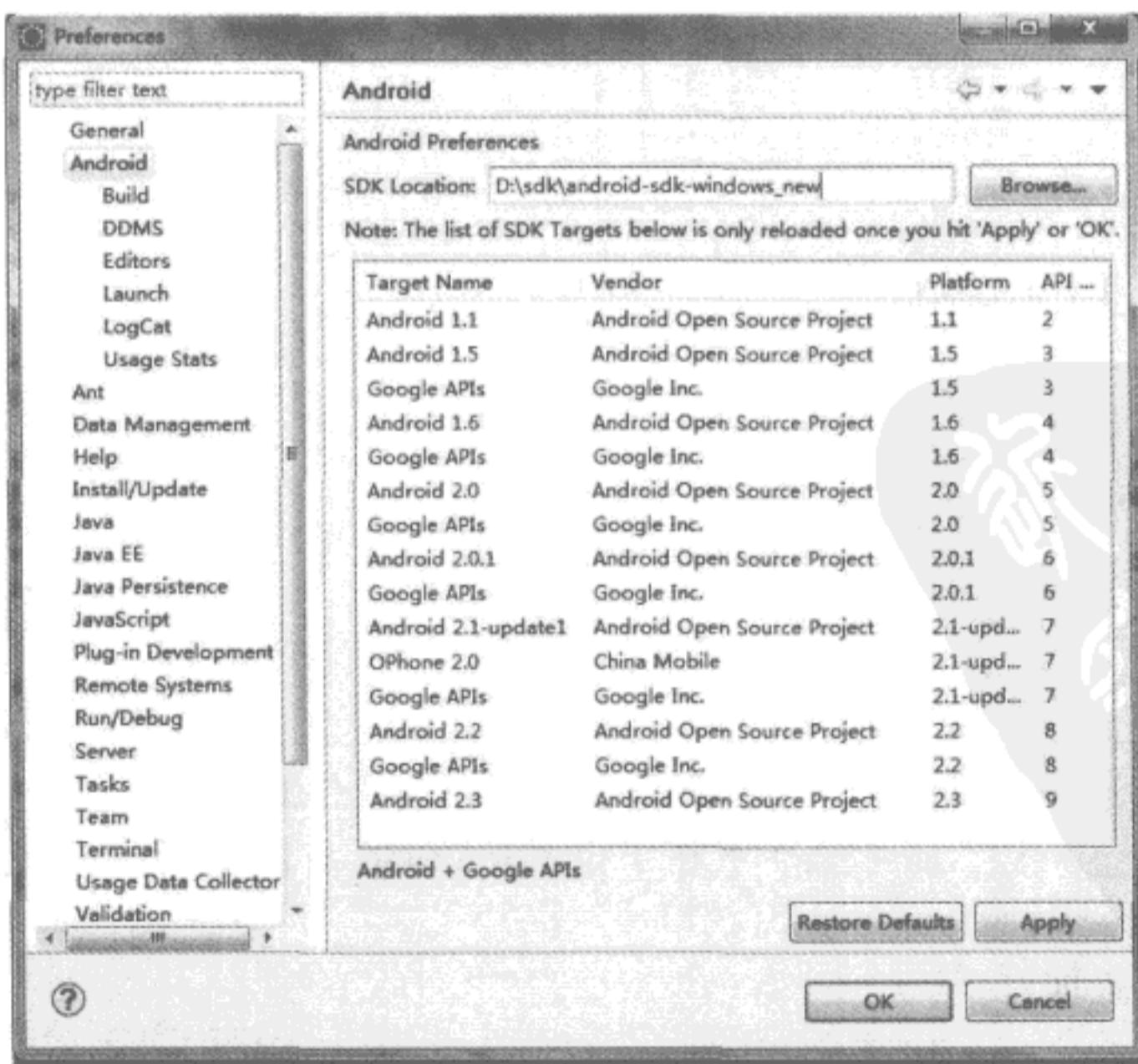
| <https://dl-ssl.google.com/android/eclipse>

单击“OK”按钮关闭该对话框。回到“Available Software”选项卡，选中刚才增加的地址，然后单击右侧的“Install”按钮开始安装 ADT 插件。在弹出的安装对话框中选择 Android DDMS 和 Android Development Tools 两项，单击“Next”按钮进入下一个安装界面，选择接受协议复选框，最后单击“Finish”按钮开始安装。当成功安装 ADT 后，重启 Eclipse 后就可以使用 ADT 来开发 Android 程序了。

如果使用的是 Eclipse 3.5(Galileo)或 Eclipse 3.6(Helios)，单击“Help”>“Install New Software...”菜单项，显示安装对话框，然后单击右侧的“Add...”按钮，在弹出的对话框的第 1 个文本框中输入一个名字，在第 2 个文本框中输入上面的地址。剩下的安装过程与 Eclipse 3.4 类似。读者可以参考 Eclipse 3.4 的安装过程或通过如下地址查看官方的安装文档。

| <http://androidappdocs.appspot.com/sdk/eclipse-adt.html>

安装完 ADT 后，还需要设置一下 Android SDK 的安装目录。单击“Window”>“Preferences”菜单项，在弹出的对话框中选中左侧的“Android”节点，在右侧的“SDK Location”文本框中输入 Android SDK 的安装目录，如图 2.9 所示。



▲ 图 2.9 设置 Android SDK 的安装目录

## 2.2

# 真实体验——编写第一个Android程序(随机绘制圆饼)

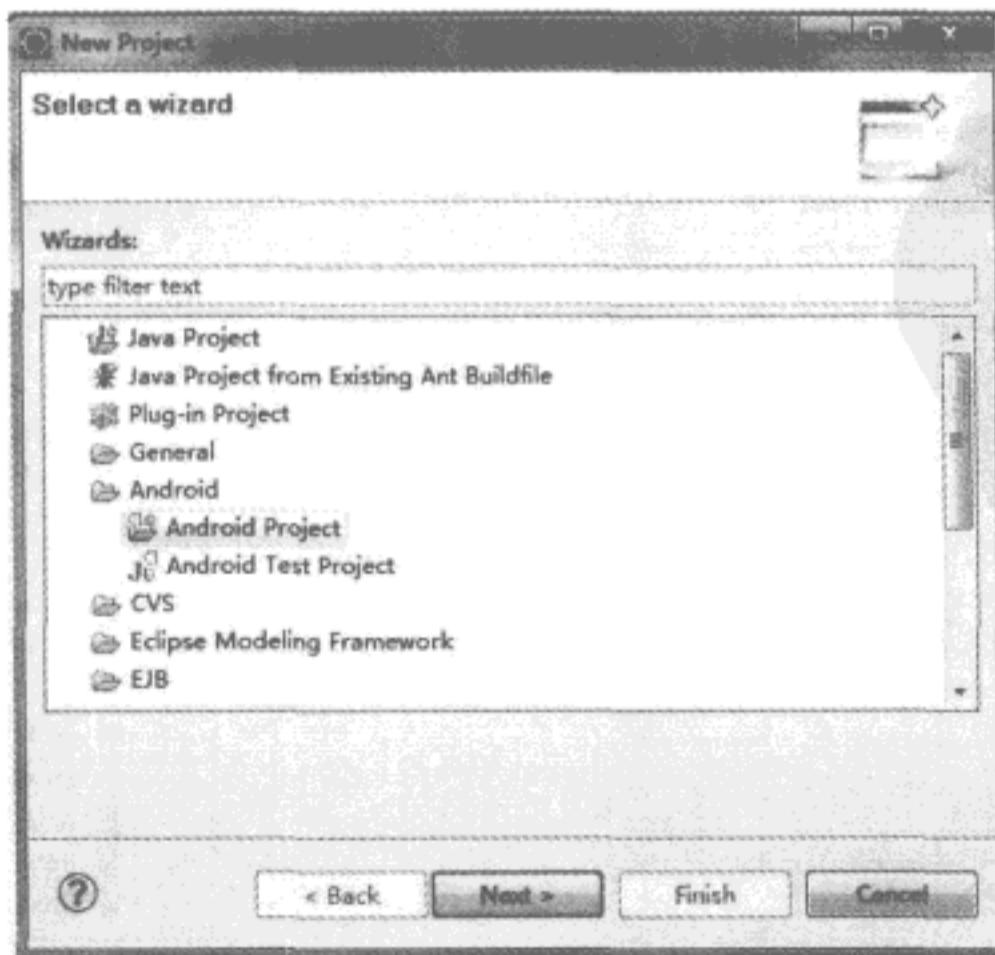
工程目录: src\ch02\ch02\_FirstAndroidApplication

在上一节中我们已经配置完了Android的开发环境,但并不太清楚所配置的开发环境是否真的可以开发Android应用程序。为了解除读者的这个疑虑,也为了使读者对开发Android应用程序的步骤有一个初步的了解,在本节将向读者展示一个完整的Android应用程序的开发过程。在很多书中常常会用Hello World或很简单的程序作为开篇的第一个例子。这样的例子虽然简单,但却不足以表现Android的魅力。因此,本节会使用一个稍微复杂一点的例子来演示开发Android程序的过程。这个例子的主要功能是在手机屏幕上随机绘制不同大小、不同位置、不同颜色的实心圆。可能很多初次接触Android的读者在看完本例后仍然有些迷惑,不过这不要紧,这个例子的目的只是让读者了解Android应用程序的基本开发步骤。随着对本书的深入学习,自然会对其中的细节有更深刻的认识。如果读者学习完本例后,仍然没有激发起对Android的兴趣,那么笔者推荐您继续阅读2.3节。在这一节将利用Google推出的App Inventor实现与本例完全一样的程序,可是不需要编写一行代码的哦(当然,也有最低的要求,就是会使用鼠标和键盘)!

### 2.2.1 创建Android工程

ADT提供了快速生成Android应用框架的功能。使用ADT通过Eclipse创建Android工程的步骤如下。

(1) 打开Eclipse开发工具,在Eclipse中选中“File”>“New”>“Android Project”菜单项,打开“New Android Project”对话框。如果在“New”菜单中没有“Android Project”子菜单项,可以选择“File”>“New”>“Project”菜单项。打开“New Project”对话框,并选择“Android”节点中的“Android Project”子节点,如图2.10所示。然后单击“Next”按钮来建立Android工程。



▲图2.10 “New Project”对话框

## Android 开发权威指南

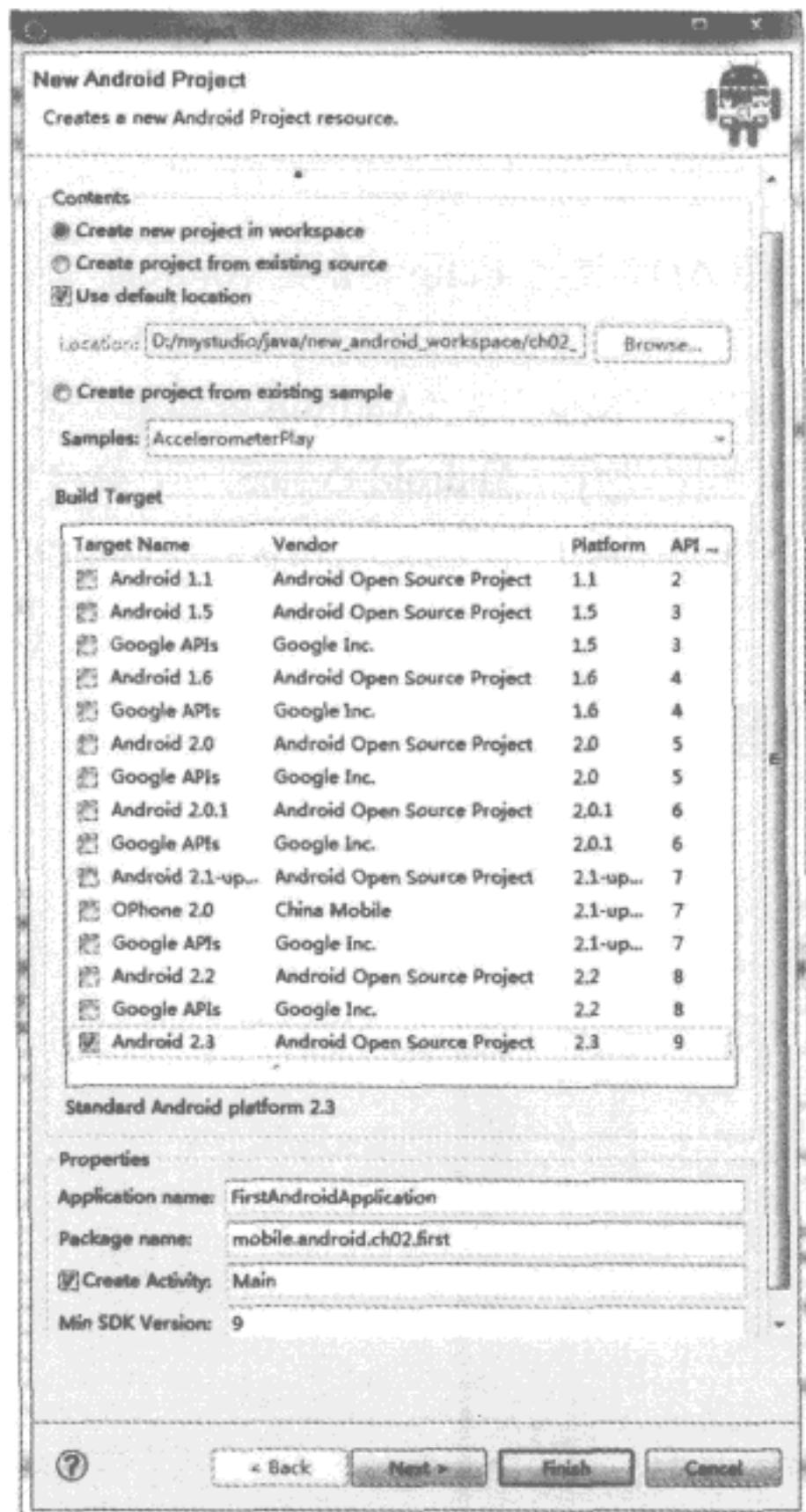
(2) 在对话框的文本框内中输入相应的内容。要输入的内容如表 2.1 所示。

**表 2.1 “New Android Project” 对话框的文本框中输入的内容**

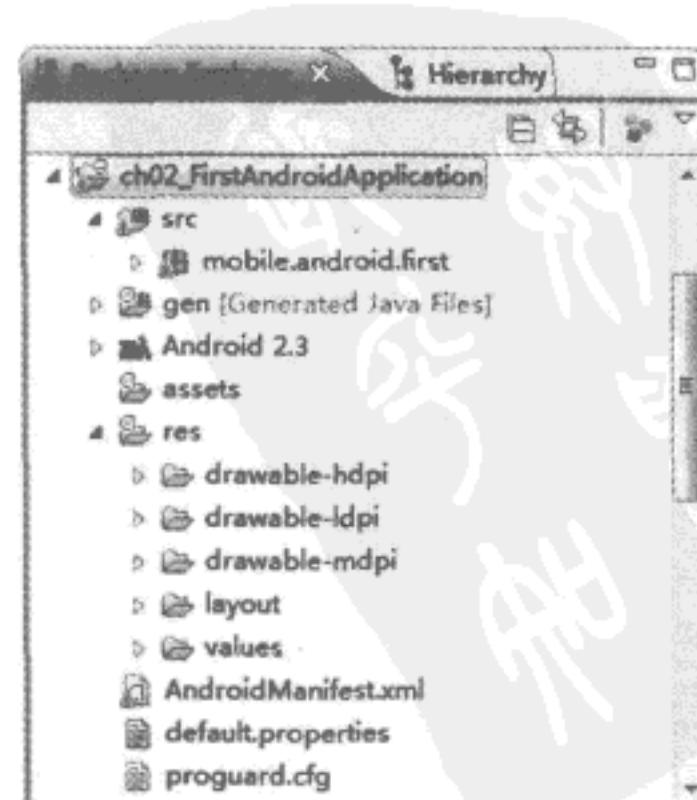
文 本 框	输入的内容
Project name	ch02_FirstAndroidApplication
Application name	FirstAndroidApplication
Package name	mobile.android.first
Create Activity	Main
Min SDK Version	9

输入完相应的内容后，在“Build Target”复选框中选择“Android2.3”。“New Android Project”对话框的最终效果如图 2.11 所示。

(3) 单击“Finish”按钮建立一个 Android 工程。在建立完 Android 工程后，会在 Eclipse 的“Package Explorer”视图中显示如图 2.12 所示的工程结构。



▲ 图 2.11 建立 Android 工程

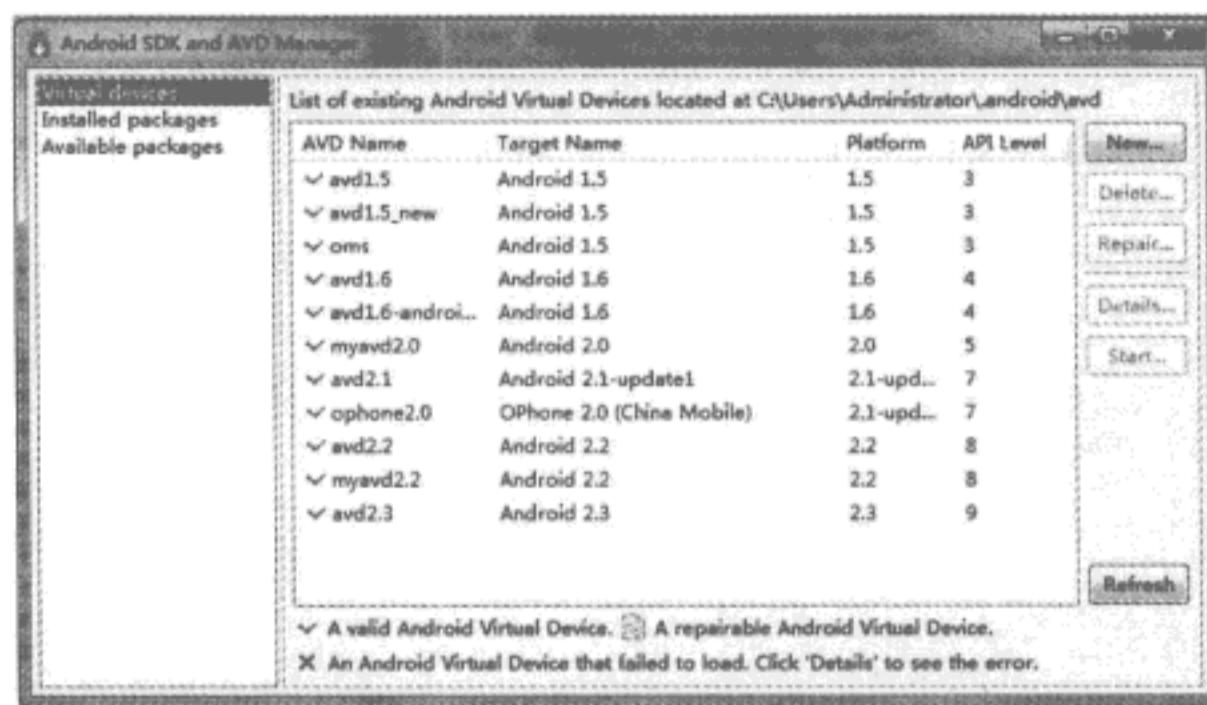


▲ 图 2.12 Android 工程结构

## 2.2.2 在模拟器中运行Android程序

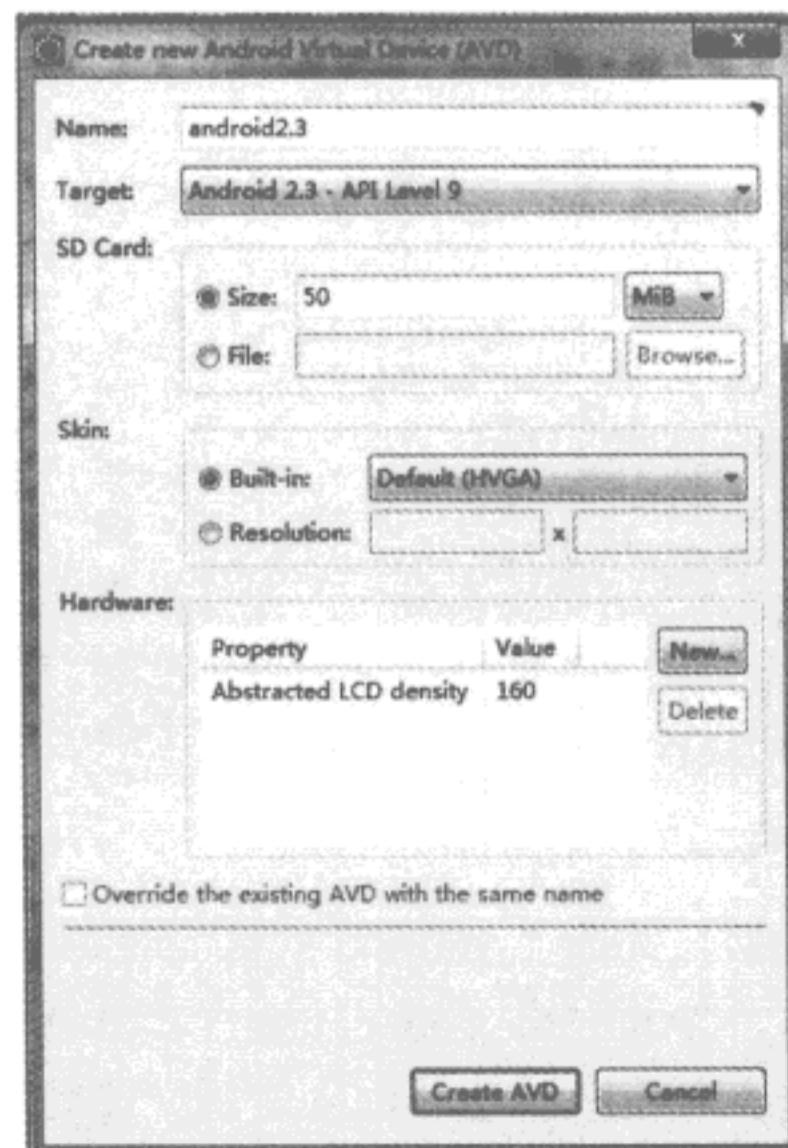
Android程序和普通的Java程序不同，不能直接运行，必须先启动一个Android模拟器，然后在Android模拟器中运行程序。运行Android程序的步骤如下。

(1) 如果模拟器还没有启动，读者可以在Eclipse中选中“Window”>“Android SDK and AVD Manager”菜单项(也可以单击Eclipse工具栏中的图标)，打开“Android SDK and AVD Manager”对话框，如图2.13所示。



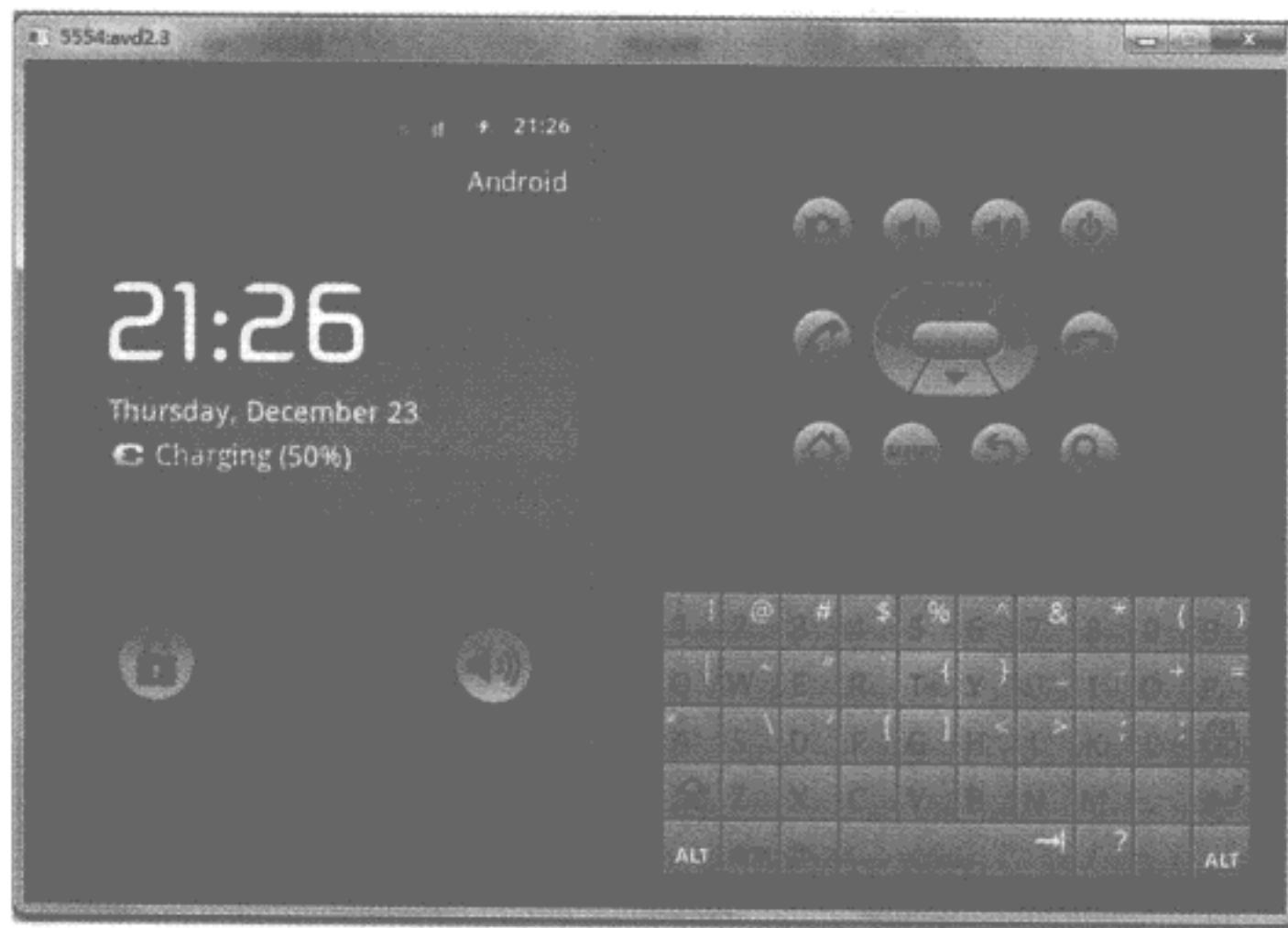
▲图2.13 “Android SDK and AVD Manager”对话框

(2) 要想运行模拟器，必须建立一个指定Android版本的AVD。单击“New”按钮，会弹出建立AVD的对话框，并输入如图2.14所示的内容。最后单击“Create AVD”按钮建立AVD。如果成功建立AVD，会在图2.13所示的界面的列表中显示刚建立的AVD。

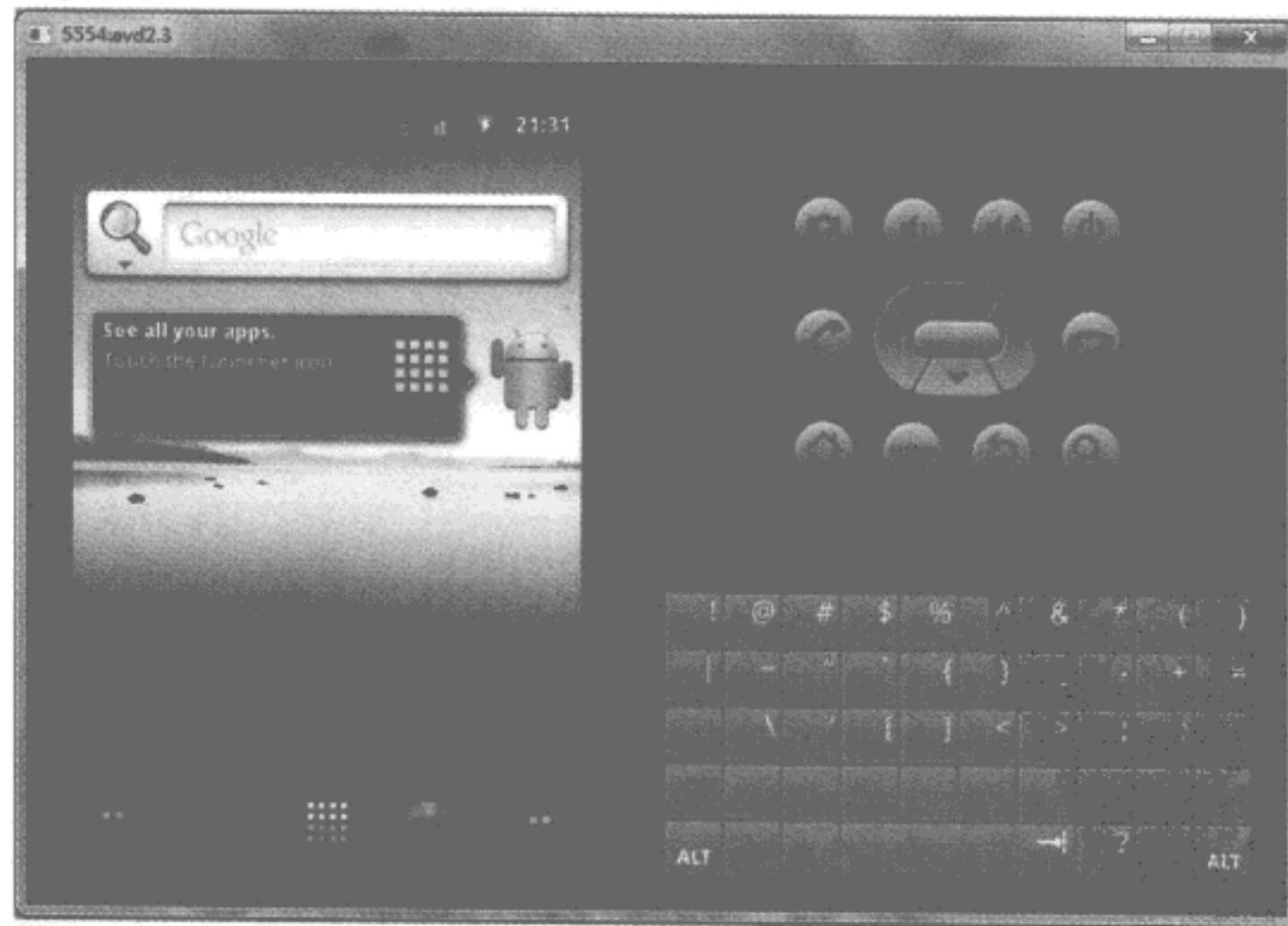


▲图2.14 建立AVD

(3) 选中图 2.13 列表中相应的 AVD，单击“Start”按钮，就会启动模拟器，如图 2.15 所示。用鼠标按住模拟器屏幕下方的“小锁”图标向右拖曳到“小喇叭”图标处，就会进入模拟器的 Home 屏，如图 2.16 所示。

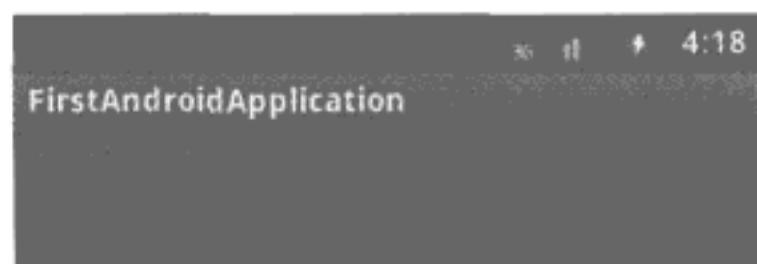


▲ 图 2.15 Android 2.3 模拟器



▲ 图 2.16 Android 2.3 模拟器的 Home 界面

(4) 成功启动 Android 模拟器后，关闭“Android SDK and AVD Manager”对话框。在 Android 工程右键菜单中选择“Run As”>“Android Application”菜单项来运行 Android 程序。读者可以在 Eclipse 中的“Console”视图中查看程序启动的进度信息。成功运行程序后，会在 Android 模拟器中显示如图 2.17 所示的信息。

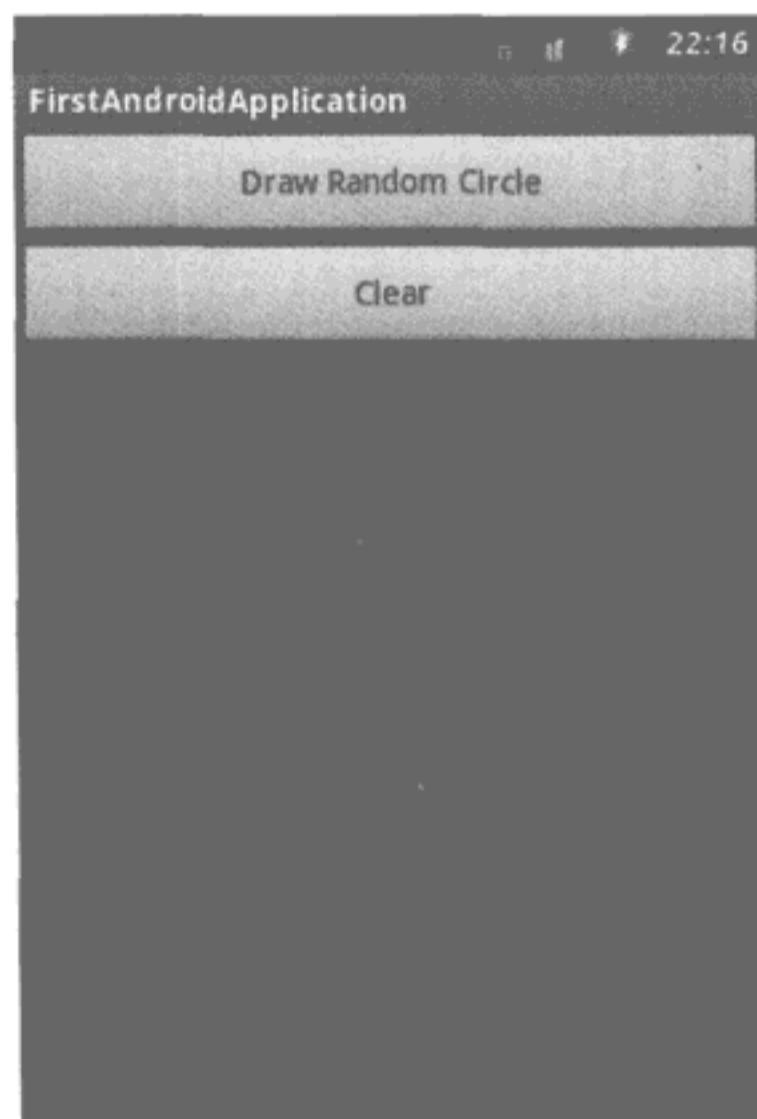


▲图 2.17 Android 程序在模拟器中的运行效果

### 2.2.3 界面控件的布局

Android 程序的界面控件可以直接通过 XML 布局文件来配置，当然，也可以直接通过程序来创建相应的控件。在本例中将结合这两种方式来完成界面的布局。

本例的界面由两个按钮和一个画布组成，如图 2.18 所示。两个按钮下面黑色的部分是画布，将要在这里随机绘制实心圆。两个按钮需要在 XML 布局文件中来配置，画布则通过代码来创建。



▲图 2.18 随机绘制实心圆程序的主界面

创建 Android 工程时，在 res/layout 目录中自动生成一个 main.xml 文件。打开这个文件，删除文件中的默认内容，并输入如下的代码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 下面的代码定义两个按钮 -->
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Draw Random Circle" android:onClick="onClick_DrawRandomCircle"/>
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Clear" android:onClick="onClick_Clear"/>
</LinearLayout>
```

## Android 开发权威指南

```
    android:text="Clear" android:onClick="onClick_Clear"/>
</LinearLayout>
```

在上面的代码中使用了<Button>标签来定义按钮，在本书后面的内容还会介绍更多的 Android 控件。在这里只要知道<Button>标签可以创建按钮即可。

### 2.2.4 编写代码

这个程序的核心是画布，也就是 Canvas 对象。要想在 Canvas 上绘制图形，需要将 Canvas 放在 View 上。因此，首先需要建立一个画布类，这个画布类是 View 的子类。当画布刷新时，会调用 onDraw 方法来重新绘制画布，我们可以从 onDraw 方法的参数来获得要绘制图形的 Canvas 对象。

由于画布重绘时会清空所有的内容，因此，要想绘制多个实心圆，需要使用一个 List 变量将曾经绘制的实心圆的相关信息（圆心坐标、半径、画笔颜色）保存起来，以便绘制下一个实心圆时重绘前面绘制的所有实心圆。如果不用 List 对象保留绘制历史，则只能绘制最后一个实心圆。下面来看看这个画布类的代码。

```
package mobile.android.ch02.first;

import java.util.ArrayList;
import java.util.List;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.view.View;

public class CircleCanvas extends View
{
    // 保存绘制历史
    public List<CircleInfo> mCircleInfos = new ArrayList<CircleCanvas.CircleInfo>();

    // 保存实心圆相关信息的类
    public static class CircleInfo
    {
        private float x;                      // 圆心横坐标
        private float y;                      // 圆心纵坐标
        private float radius;                 // 半径
        private int color;                    // 画笔的颜色

        public float getX()
        {
            return x;
        }
        public void setX(float x)
        {
            this.x = x;
        }
        public float getY()
        {
            return y;
        }
    }
}
```

## 第2章 工欲善其事，必先利其器——搭建和使用Android开发环境

```

public void setY(float y)
{
    this.y = y;
}
public float getRadius()
{
    return radius;
}
public void setRadius(float radius)
{
    this.radius = radius;
}
public int getColor()
{
    return color;
}
public void setColor(int color)
{
    this.color = color;
}
}
public CircleCanvas(Context context)
{
    super(context);
}
// 当画布重绘时调用该方法，Canvas 表示画布对象，可以在该对象上绘制基本的图形
@Override
protected void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    // 根据保存的绘制历史重绘所有的实心圆
    for (CircleInfo circleInfo : mCircleInfos)
    {
        Paint paint = new Paint();
        // 设置画笔颜色
        paint.setColor(circleInfo.getColor());
        // 绘制实心圆
        canvas.drawCircle(circleInfo.getX(), circleInfo.getY(), circleInfo.getRadius(),
            paint);
    }
}
}

```

下面我们来编写主程序。在创建工程时会要求输入一个“Create Activity”，由于我们输入的是“Main”，因此，生成的主类就是 Main.java。打开 Main.java 文件，输入如下的代码。

```

package mobile.android.ch02.first;

import java.util.Random;
import mobile.android.first.CircleCanvas.CircleInfo;
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;

```

## Android 开发权威指南

```

import android.view.ViewGroup;
import android.view.ViewGroup.LayoutParams;

public class Main extends Activity
{
    private CircleCanvas mCircleCanvas; // 定义一个画布类

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 装载布局文件(在2.2.3节配置的main.xml文件)
        ViewGroup viewGroup = (ViewGroup) getLayoutInflator().inflate(R.layout.main, null);
        mCircleCanvas = new CircleCanvas(this); // 创建CircleCanvas(画布类)对象
        // 将CircleCanvas对象添加到当前界面的视图中(两个按钮的下方)
        viewGroup.addView(mCircleCanvas, new LayoutParams(LayoutParams.FILL_PARENT, 350));
        setContentView(viewGroup);
    }
    // 开始随机绘制圆形(第一个按钮的单击事件)
    public void onClick_DrawRandomCircle(View view)
    {
        Random random = new Random();
        float randomX =(float)( 100 + random.nextInt(100)); // 随机生成圆心横坐标(100至200)
        float randomY =(float)( 100 + random.nextInt(100)); // 随机生成圆心纵坐标(100至200)
        float randomRadius =(float)( 20 + random.nextInt(40)); // 随机生成圆的半径(20至60)
        int randomColor = 0;
        // 产生0至100的随机数,若产生的随机数大于50,则画笔颜色为蓝色
        if(random.nextInt(100) > 50)
        {
            randomColor = Color.BLUE;
        }
        else
        {
            // 产生0至100的随机数,若产生的随机数大于50,则画笔颜色为红色
            if(random.nextInt(100) > 50)
                randomColor = Color.RED;
            // 否则,画笔颜色为绿色
            else
                randomColor = Color.GREEN;
        }
        CircleInfo circleInfo = new CircleInfo();
        circleInfo.setX(randomX);
        circleInfo.setY(randomY);
        circleInfo.setRadius(randomRadius);
        circleInfo.setColor(randomColor);
        mCircleCanvas.mCircleInfos.add(circleInfo); // 将当前绘制的实心圆信息加到List对象中
        mCircleCanvas.invalidate(); // 使画布重绘
    }
    // 清空画布(第二个按钮的单击事件)
    public void onClick_Clear(View view)
    {
        mCircleCanvas.mCircleInfos.clear(); // 清除绘制历史
    }
}

```

```

    mCircleCanvas.invalidate(); // 使画布重绘
}
}

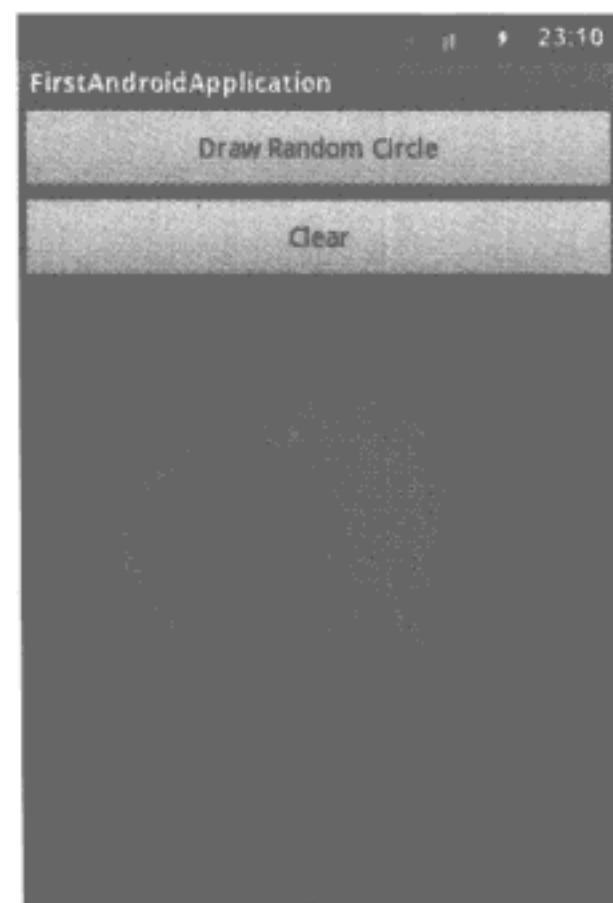
```

下面来运行程序，并不断用鼠标单击第一个按钮，会在画布中绘制多个不同颜色、不同半径、不同位置的实心圆，如图 2.19 所示。当单击第二个按钮后，会将当前绘制的所有实心圆清空。

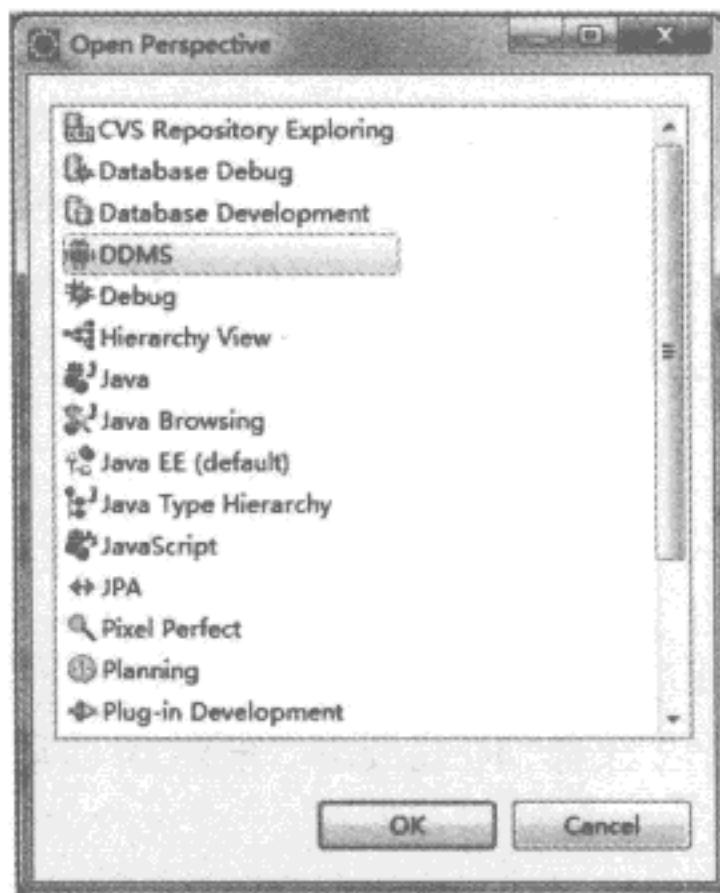
## 2.2.5 调试程序

在 Eclipse 开发工具中调试程序的方法很多，例如，可以在 ADT 插件提供的 DDMS（Dalvik Debug Monitor Service）透视图中查看进程状态；在 LogCat 视图中查看系统输出的调试信息以及自己使用 Log 类的相应方法输出的调试信息；在代码中设置断点进行跟踪。甚至还可以用 DDMS 透视图中带的工具进行更高级的调试，例如，可以对设备截屏，针对特定的进程查看正在运行的线程以及堆信息、广播状态信息、模拟电话呼叫、发送短信和地理坐标等。下面我们将详细介绍这些常用的调试程序的方法，并利用这些方法来调试前面编写的绘制随机实心圆的程序。

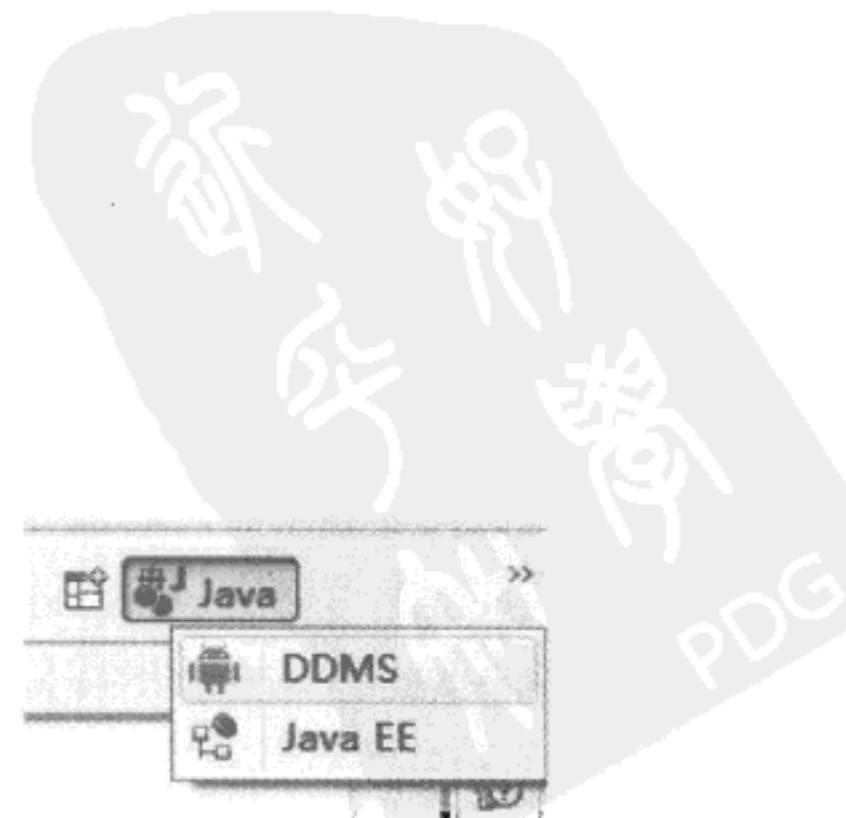
首先来看看如何使用 DDMS 透视图。在默认情况下，Eclipse 的主界面并没有显示 DDMS 透视图。读者需要选择“Window”>“Open Perspective”>“DDMS”菜单项来显示 DDMS 透视图。如果在“Open Perspective”菜单项中没有“DDMS”子菜单项，就选择“Window”>“Open Perspective”>“Other”菜单项，打开“Open Perspective”对话框，在列表中选中“DDMS”列表项，如图 2.20 所示。最后单击“OK”按钮关闭对话框。在完成前面的操作后，会在 Eclipse 主界面的右上角看到“DDMS”开关状态按钮。如果没有“DDMS”按钮，可以单击右侧的“>>”按钮显示其他未显示的按钮，找到“DDMS”按钮，选中即可，如图 2.21 所示。



▲ 图 2.19 随机绘制实心圆

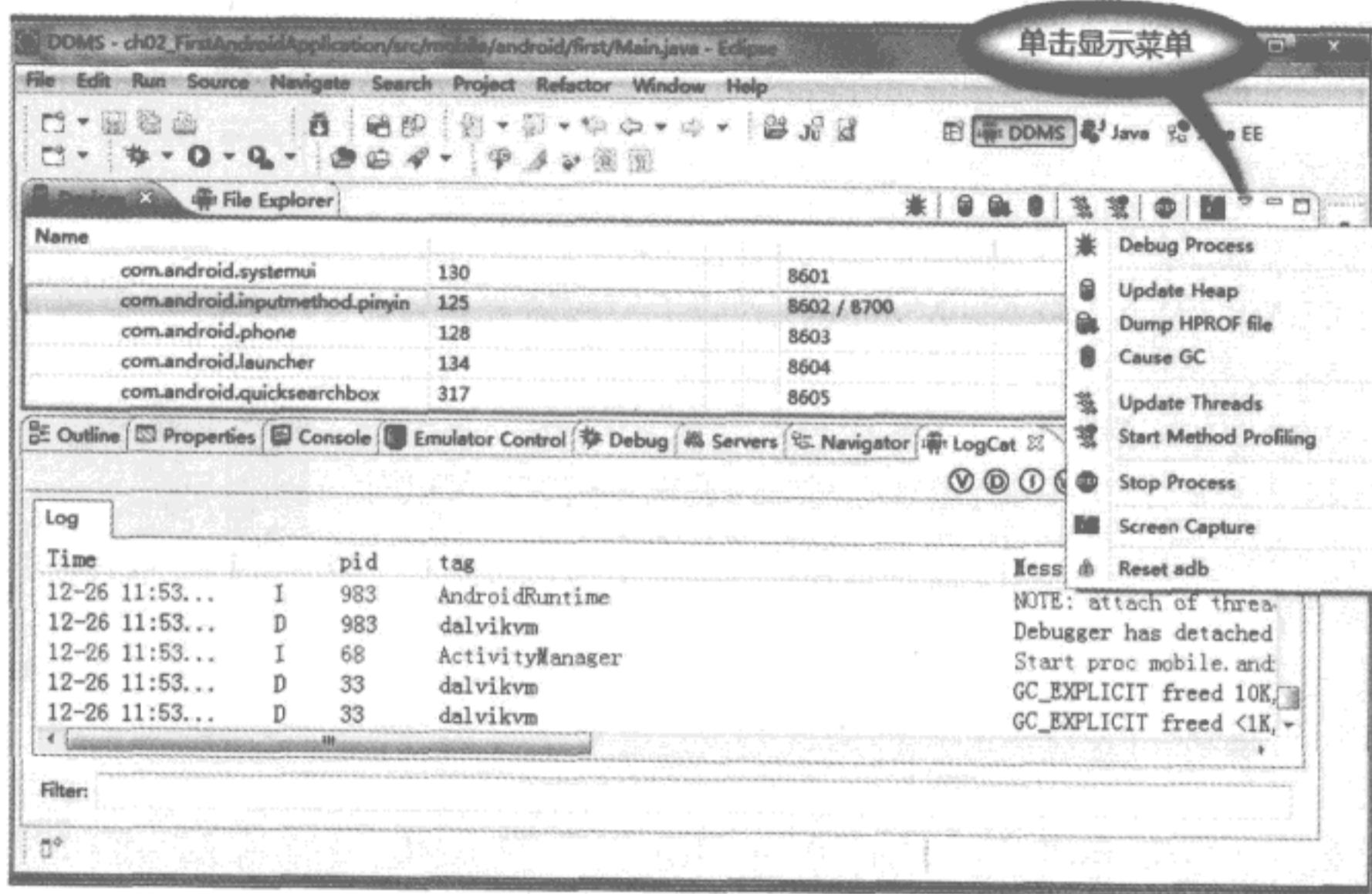


▲ 图 2.20 选择“DDMS”透视图



▲ 图 2.21 打开“DDMS”透视图

进入“DDMS”透视图后，可看到如图 2.22 所示的界面。在“DDMS”透视图中有很多视图，其中最常用的是“LogCat”视图。在该视图中可以显示系统或用户输出的调试信息。除此之外，在“Devices”视图右侧的菜单中提供了更多的调试功能。这些功能如下：



▲ 图 2.22 DDMS 透视图的主界面

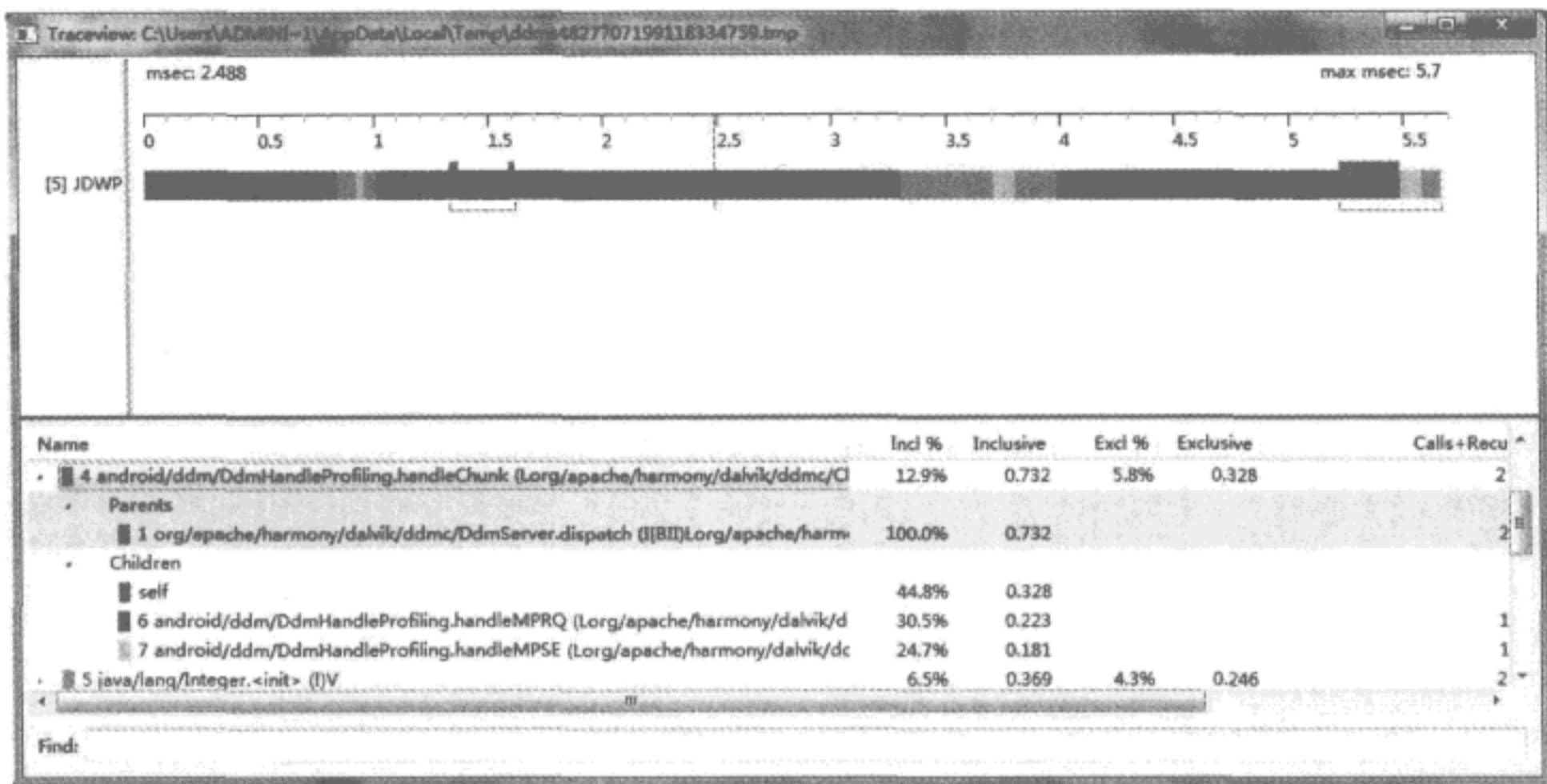
- Debug Process（调试进程）。
- Update Heap（更新堆）。
- Dump HPROF file（转存 HPROF 文件）。
- Cause GC（引起垃圾回收）。
- Update Threads（更新进程）。
- Start Method Profiling（开发方法执行分布图）。此功能可以分析程序中方法的执行情况。

首先应在如图 2.22 所示的“Devices”视图列表中选择要监视的进程，然后选择“Start Method Profiling”，该菜单项会变成“Stop Method Profiling”，过一段时间后，再选择“Stop Method Profiling”菜单项，就会弹出如图 2.23 所示的界面，在该界面中会显示所监视的进程中方法的执行时间分布情况。

- Stop Process（停止进程）。
- Screen Capture（屏幕截图）。
- Reset adb（重启 Android Debug Bridge）。如果 adb 调试桥运行不稳定（据笔者测试，adb 在运行一段时间后，有可能出现停止、无法上网等情况），可以选择“Reset adb”来重新启动“adb.exe”进程。

在熟悉完“DDMS”透视图后，我们就可以利用“DDMS”透视图进行调试了。在“DDMS”透视图中最常用的是“LogCat”视图。通过该视图可以查看系统及开发人员输出的调试信息。

## 第2章 工欲善其事，必先利其器——搭建和使用Android开发环境



▲图2.23 方法的执行分布图

如果想通过Java代码在“LogCat”视图中输出相应的调试信息，需要使用到`android.util.Log`类中的5个静态的方法，这5个静态方法如下：

- `Log.v(String tag, String msg);`
- `Log.d(String tag, String msg);`
- `Log.i(String tag, String msg);`
- `Log.w(String tag, String msg);`
- `Log.e(String tag, String msg);`

这5个方法的首字母(v、d、i、w、e)分别对应VERBOS、DEBUG、INFO、WARN、ERROR。这5个方法的区别并不大，只是输出信息的颜色不同。通过使不同的信息输出不同的颜色，可以在“LogCat”视图中更好地查看调试信息。除了这5个方法外，还有一个`Log.println`方法也可以在“LogCat”视图中输出调试信息，该方法的定义如下：

```
public static native int println(int priority, String tag, String msg);
```

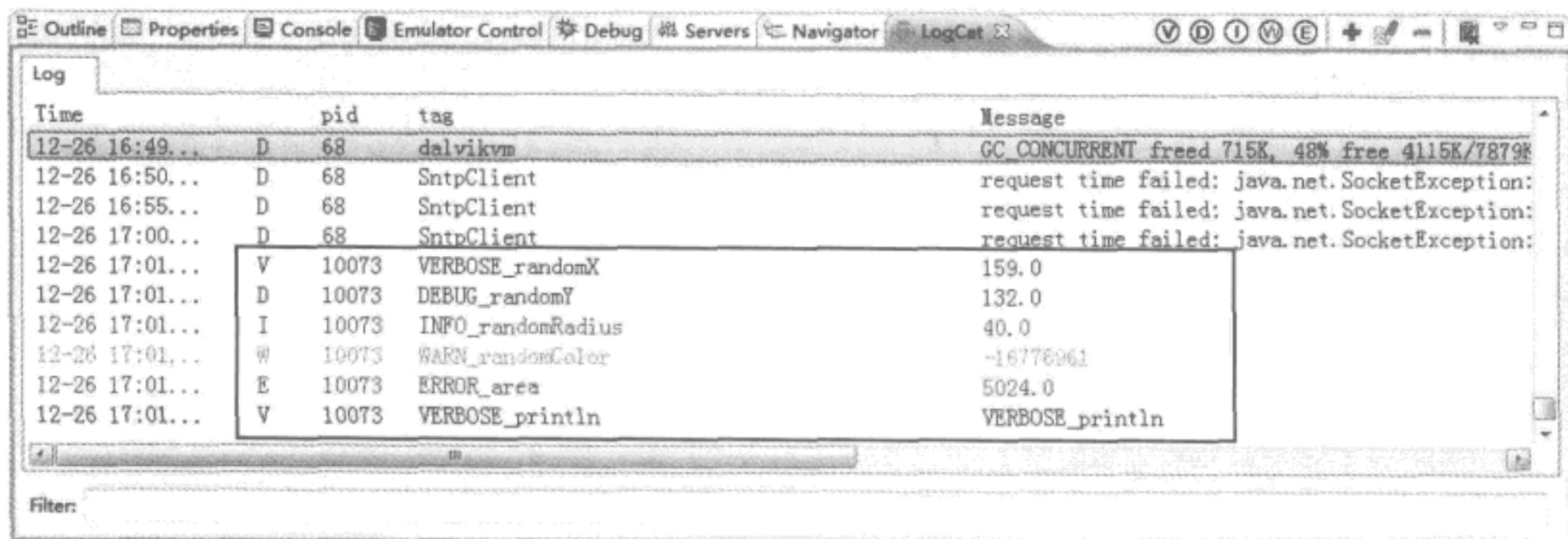
`println`是一个本地方法。上面介绍的5个输出日志的方法在内部都调用了`println`方法。`println`方法的后两个参数与这5个方法的相应参数的含义完全相同，而第一个参数表示要输出信息的优先级，实际上就是VERBOS、DEBUG、INFO、WARN和ERROR五个级别。分别用`Log.VERBOSE`、`Log.DEBUG`、`Log.INFO`、`Log.WARN`、`Log.ERROR`。下面的代码在随机绘制实心圆的例子的“DrawRandomCircle”按钮单击事件中输出这5个级别的调试信息，最后用`println`方法输出VERBOS级别的调试信息。

```
// 分别输出5个级别的调试信息
Log.v("VERBOSE_randomX", String.valueOf(randomX));
Log.d("DEBUG_randomY", String.valueOf(randomY));
Log.i("INFO_randomRadius", String.valueOf(randomRadius));
Log.w("WARN_randomColor", String.valueOf(randomColor));
```

## Android 开发权威指南

```
Log.e("ERROR_area", String.valueOf(3.14 * randomRadius * randomRadius));
// 使用 println 方法输出 VERBOSE 级别的调试信息
Log.println(Log.VERBOSE, "VERBOSE_println", "VERBOSE_println");
```

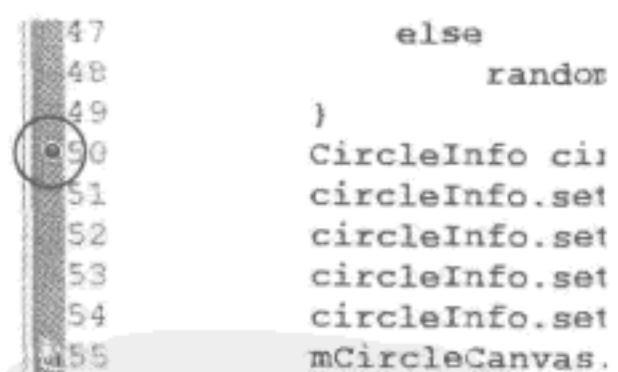
执行上面代码后的输出结果如图 2.24 所示。



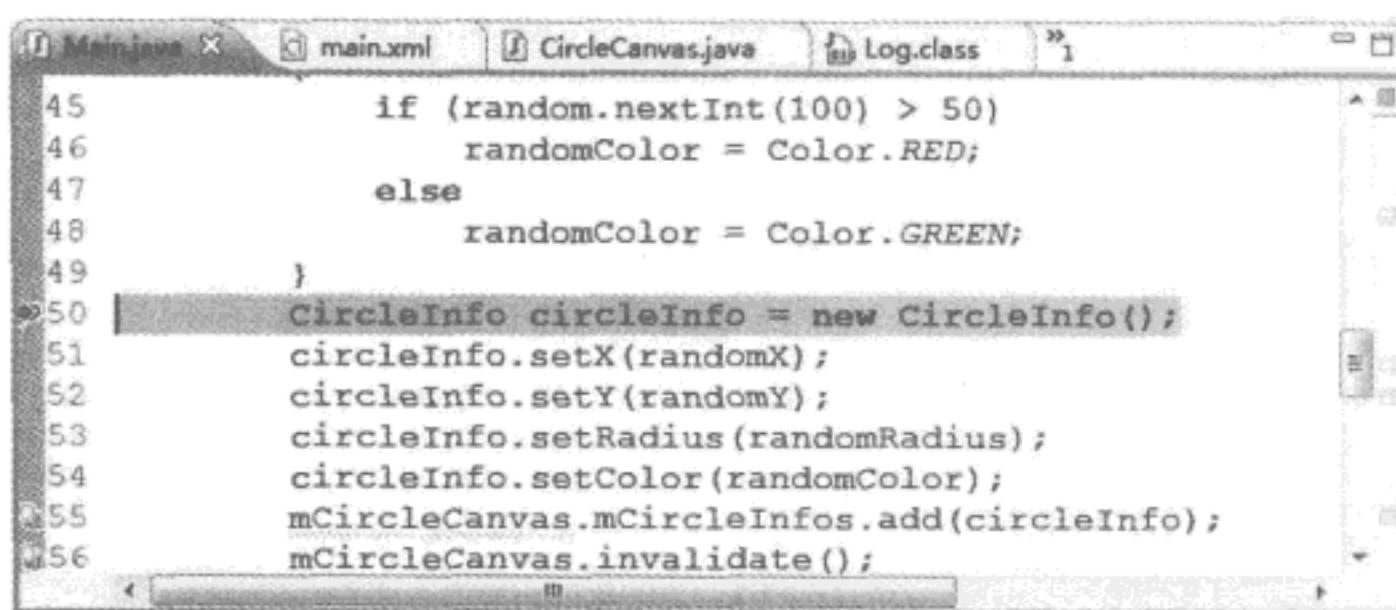
▲ 图 2.24 输出各个级别的调试信息

除了在“LogCat”视图中输出调试信息外，还可以通过在程序中设置断点的方式来调试程序。在 Eclipse 中打开要调试的 Java 源代码文件，并在要设置断点的位置前面的灰色竖条（如果光标所处的位置正好在当前方法内，则当前方法的垂直区间的灰色竖条会变成蓝色竖条）处双击，或在灰色竖条处右键菜单中选择“Toggle Breakpoint”菜单项来设置断点。在设置完断点后，会看到在灰色竖条处多了一个蓝色的实心圆点，如图 2.25 所示。如果想取消断点，可以再次双击已设置断点的位置，或在灰色竖条右键菜单中选择“Disable Breakpoint”菜单项。

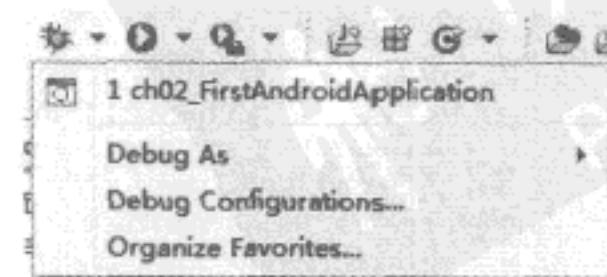
启动模拟器后，选择工程右键菜单中的“Run”>“Debug As”>“Android Application”菜单项，在模拟器中运行程序。然后单击第一个按钮，就会进入调试状态，程序会停止在断点处，如图 2.26 所示。如果已经运行过一遍程序，在 Eclipse 上方的工具栏的调试按钮中会出现以工程名命名的运行项，如图 2.27 所示。下次再运行程序时，直接单击该运行项即可。



▲ 图 2.25 设置断点



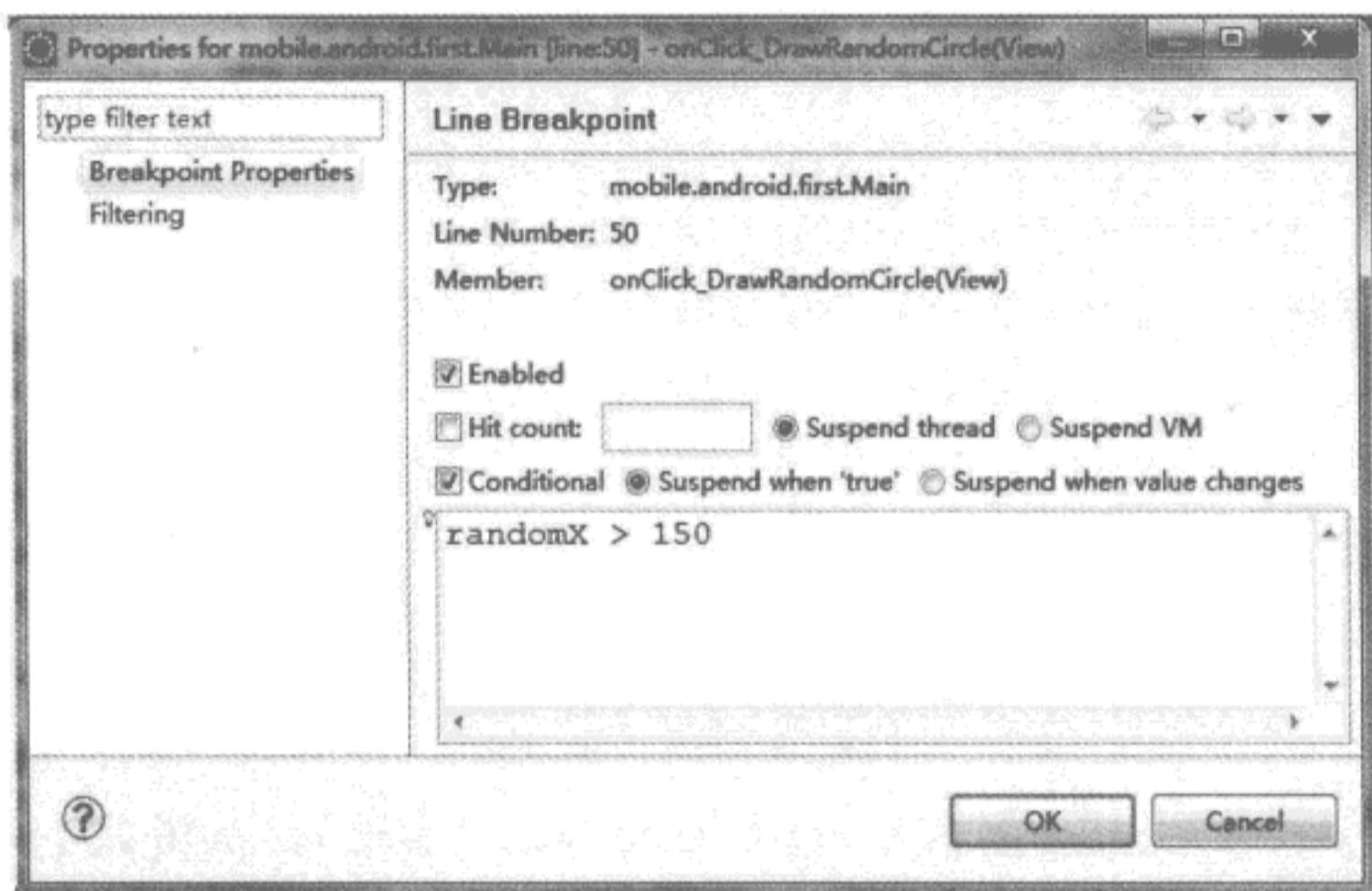
▲ 图 2.26 调试界面



▲ 图 2.27 调试菜单命令

## 第2章 工欲善其事，必先利其器——搭建和使用Android开发环境

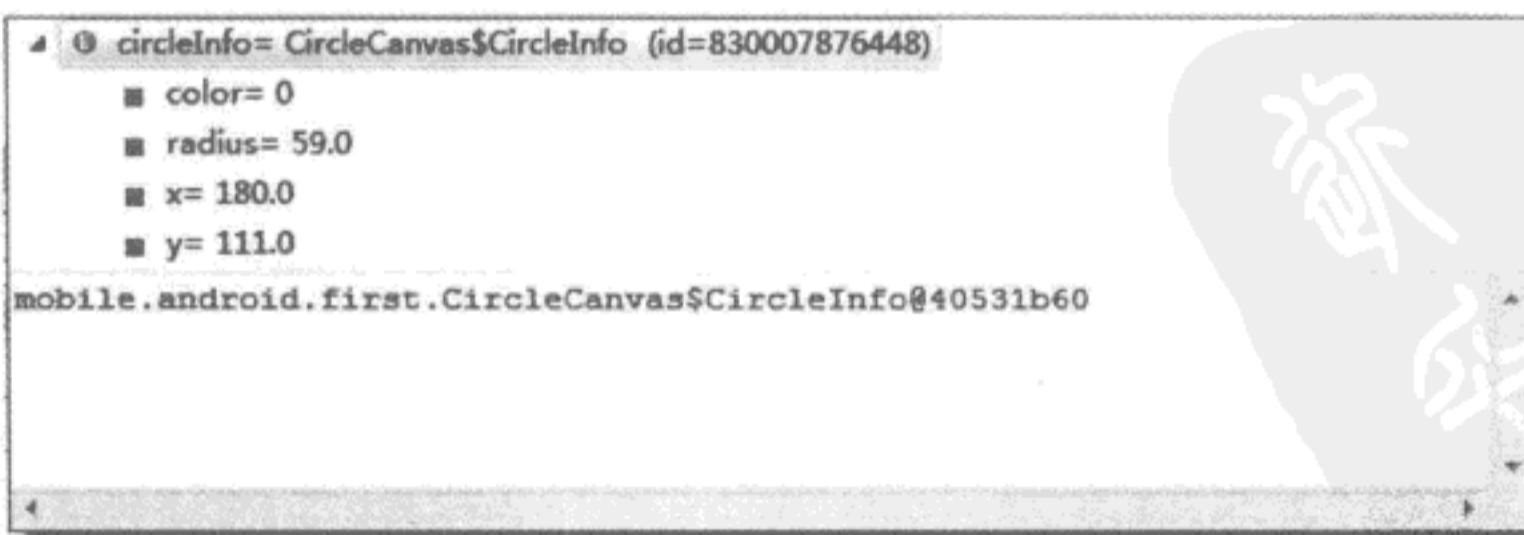
虽然通过断点来跟踪程序是非常直接的，但在默认情况下是无条件跟踪，也就是在任何条件下，程序只要执行到断点处就会中断程序的执行，并进入调试界面。但在很多时候，我们并不希望任何情况都进入中断状态。例如，我们只希望 randomX 的值在大于 150 时才进入中断状态。要达到这个目的，就要在断点处右键菜单中选择“Breakpoint Properties”菜单项打开调试属性对话框，在右侧的条件文本框中输入“randomX > 150”，如图 2.28 所示。然后单击“OK”按钮关闭对话框。



▲ 图 2.28 设置调试属性

这时再单击程序的第一个按钮，如果随机生成的 randomX 值不大于 150，程序仍然会正常运行，并不会进入中断状态。

如果想在调试状态获得某个变量的当前值，可以将鼠标指针放到变量所在的位置，这时会弹出包含当前变量值的信息框，如图 2.29 所示。



▲ 图 2.29 显示指定变量的值

本节介绍了调试 Android 程序的一些常用方法。读者在选择调试方式时可以根据自己的实际情况选择一种或几种调试方式。一般情况下，在 try{ ... }catch{ ... }语句的 catch 块中可以使用 Log.e 来输出程序中的错误信息。这样在程序抛出异常时可以很容易在“LogCat”视图中发现。

## 2.2.6 在手机上运行和调试程序

虽然 Android 模拟器可以满足大部分程序的需要,但仍然有一些程序无法在模拟器中正常运行。例如,在模拟器中无法模拟蓝牙(Bluetooth)进行通信。或者有一些使用 GPS 定位的程序通过模拟器来确定位置并不方便,有时甚至会使程序无法正常运行。在这些情况下,就要求将程序直接放到手机上运行。

如果只是在手机上运行 Android 程序并不困难。由于 Android 程序是以扩展名为.apk 的文件(相当于 Windows 中的 exe 文件)发布的,因此,只要将 apk 文件复制到手机的内存或 SD 卡上,然后安装即可。但这还不够,对于正在开发的程序,还需要直接在手机上调试。也就是说,需要把手机当成模拟器使用,并使用本书 2.2.5 节介绍的方法来调试程序。

在手机上调试程序,首先需要使用 USB 数据线将手机和电脑连接(目前手机还不能通过 Wifi 或蓝牙连接电脑进行调试),然后在手机上方的通知栏找到“更改 USB 连接类型”通知项(或叫类似的名字),选择 HTC Sync(注意,本例使用的是 HTC 的手机,对于其他厂商的手机,可能操作方法和选项有所不同,但基本上都类似),如图 2.30 所示。这时我们会在 DDMS 透视图中的 Devices 视图中看到多了一个设置,名字为 HT9BYL904399(可能根据电脑上连接的设备或手机不同,这个名字会有所不同,但都会出现一个区别于模拟器的名字),这时就可以像使用模拟器一样使用手机来调试和运行程序了。如果读者的手机仍然无法成功调试程序,需要选择手机中的“设备”>“应用程序”>“开发”项,在显示的界面中选中“USB 调试”复选框,如图 2.31 所示。这时程序就可以正常在手机上调试了。

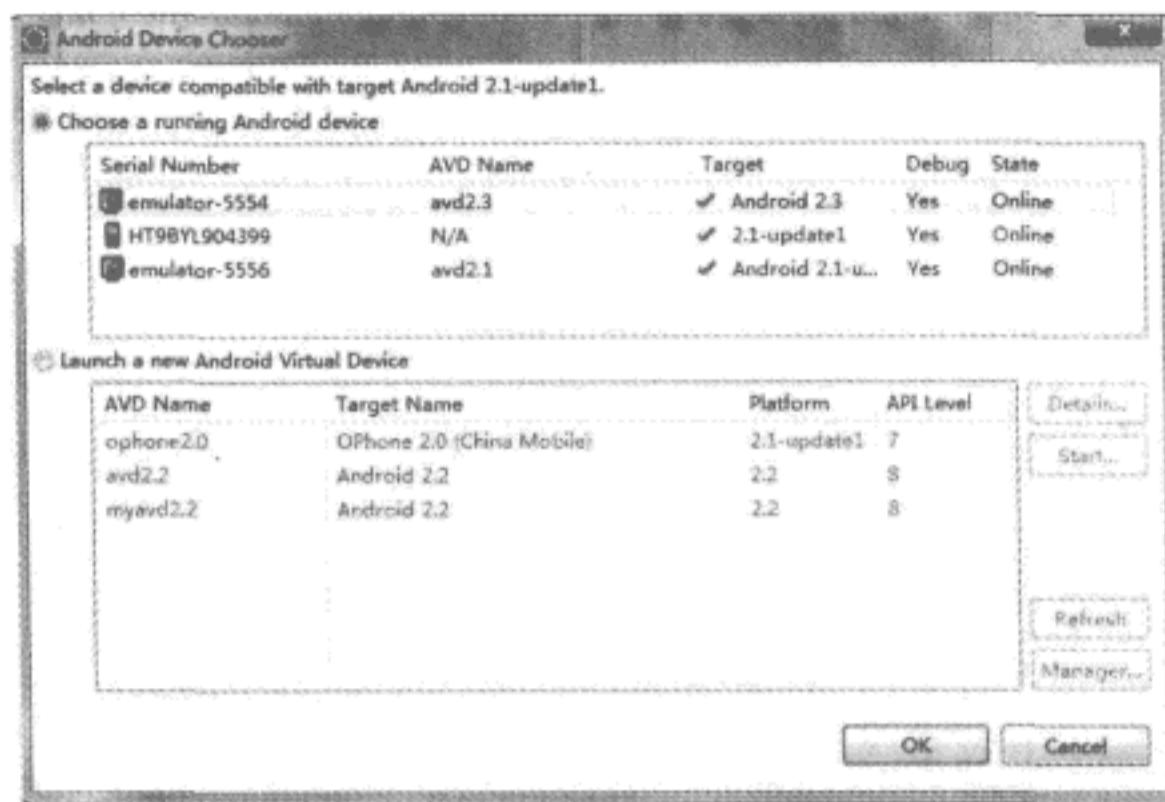


▲ 图 2.30 在手机上设置 HTC Sync



▲ 图 2.31 打开 USB 调试功能

如果读者既启动了模拟器,又通过 USB 数据线连接了手机,或启动了多个模拟器,连接了多个手机,在启动程序时,会弹出一个如图 2.32 所示的对话框,要求开发人员选择在哪个设备上运行。



▲图2.32 选择要运行程序的设备

## 2.3 迁移Android工程可能发生的错误

除了要自己编写程序外，还需要阅读大量别人写的程序，或是将其他机器上自己编写的源代码迁移到另一台机上。完成这些工作的第一步就是将工程导入到Eclipse中。如果包含源代码的Android工程在光盘中，建议先将其复制到硬盘上，然后在Eclipse中选择“File”>“Import”菜单项，打开“Import”对话框，选中“Existing Projects into Workspace”节点，然后单击“Next”按钮，并按提示导入已存在的工程。

虽然导入过程并不复杂，但导入后的程序有可能会出现错误，也就是在工程图标上出现一个红叉。经笔者长时间测试和总结，出现这种情况的原因可能是以下一种或几种。

- **JDK名字不同。**虽然在不同的电脑上都安装和配置了JDK，但在Eclipse中为JDK起的名字可能不同。在Eclipse中配置的JDK可以在“Preferences”对话框中的“Java”>“Installed JREs”节点对应的右侧列表中找到。如果发现和Android工程原来所有的Eclipse中的JDK名字不同，可以在这里修改一下。

- **引用的jar文件在目标机器上不存在。**在Android工程中可能引用了第三方的jar包，而且这些jar包保存在其他目录中。在迁移Android工程时未将这些包也迁移过来。解决的方法很简单，只需要将这些jar文件复制到目标机器上与原机器相同的目录中，或在Android工程中重新引用这些jar文件即可。如果为了以后迁移Android工程更方便，可以直接将这些jar包放到Android工程中。最好不要放在像src、res这些需要同步的目录中，建议在Android工程中建立一个lib目录（也可以建立若干子目录），将所有的jar包都放在lib或其子目录中，然后直接引用工程中的jar包。由于这种引用方式使用了相对目录，因此只要复制了完整的Android工程目录，就绝对不会出现由于jar文件没找到或路径错误的异常情况。

- **跨工程引用的工程可能包含错误或不存在。**在Eclipse中可以跨工程引用。例如，当很多Android工程都需要某个功能时，可以将这个功能单独封装在一个Eclipse工程中（Android和非Android工程都可以）。由于在开发过程中，Android工程和被引用的工程需要同时调试，因此，必须在Android工程中引用另外一个工程。如果我们在迁移Android工程的过程中未复制被引用的工程，

那么 Android 工程就会由于未找到被引用的工程而产生错误，这个错误非常类似于未找到引用的 jar 文件所产生的错误，只是未找到的是 Eclipse 工程，而不是 jar 文件。如果被引用工程也同样复制并导入到了新的 Eclipse 环境中，Android 工程仍然有错误，而 Android 工程中并未发现任何的出错迹象，可能是由于被引用工程的错误而导致 Android 工程产生错误。只要消除被引用工程中的错误，Android 工程中的错误自然消失。本节介绍的几种可能性仍然可以应用在被引用工程中的错误排除上。

- ADT 或 Android 的版本太低。如果目标机器上所安装的 ADT 或 Android 的版本太低，例如，要迁移的程序最低的要求是 Android SDK 2.2，而目标机器上最高的 Android SDK 版本只到 2.1，则 Android 工程会发生错误。排除错误的唯一方法就是升级到更高的 ADT 或 Android 版本。

- Android 工程中突然多出了同名的图像文件。这个问题如果遇到了会很郁闷。如果目标机器上没有旧版本的同名 Android 工程就不可能发生这种情况。但不幸的是，往往在目标机器上会存在同名的旧版本 Android 工程。例如，我们可能在单位时编写了一个新版本的 Android 程序，而晚上要将程序拿回家继续做，但家里的电脑中同一个 workspace（Eclipse 保存工程的目录）中有一个同名的旧版本的 Android 工程。而且更不幸的是，这个旧版本的 Android 工程中的图像资源（drawable 中的图像文件，将在后面详细介绍）在新版本的 Android 工程中换了扩展名。例如，旧版本中有一个 abc.jpg，在新版本中变成了 abc.png。而在粘贴新工程的过程中，并没有在删除旧工程后再粘贴，这时在新工程中就会出现两个同名（文件名相同，扩展名不同）的图像文件：abc.jpg 和 abc.png。这对于操作系统的文件系统是有效的，但对于 Android 的 ADT 却有大麻烦了。因为 ADT 会为每一个图像资源文件生成一个唯一的 ID（静态变量），这个 ID 就是图像资源文件的文件名（不包括扩展名）。这时就会产生冲突，因为在 Java 语言中不允许在同一个作用域中包含两个或多个同名的变量。因此，在 drawable 目录中不允许出现主文件名相同，而扩展名不同的两个或多个文件。解决的方法就是只保留一个这样的文件，其他的全部删除或移到其他的目录中。除了 drawable 目录中的图像文件可能引起这样的错误，源程序文件或其他资源文件同样可能引起这样的错误。例如，某个旧 Android 工程的某个 Java 源文件引用了第三方 jar 包中的某个类，而在新 Android 工程中使用了其他的类来代替 jar 包中的这个类，并且在新的 Android 工程中已经将这个 jar 包移除，这时如果在迁移 Android 工程时仍然保留了这个引用 jar 包中相应类的 Java 文件，那么就会出现引用错误。只要 Android 工程中有任何错误，Android 工程图标上就会显示一个红叉。因此，建议在复制新的 Android 工程之前先将旧 Android 工程完全删除，这样就不可能发生这样的情况了。

- 其他情况。如果读者已经将上述几种可能性一一排除后，Android 工程仍然有错误，那么情况可能会有些微妙（有的情况可能是由于 ADT 或是 Eclipse 的 Bug 引起的）。读者不妨先选择“Project”>“Clean”菜单项来清理一下 Android 工程，如果还没清除错误，可以将 Android 工程从“Package Explorer”视图中删除（不要彻底删除，只在工程列表中删除即可），然后再重新导入 Android 工程。当然，这些操作有可能仍然未解决读者的问题，这样情况就会显得更复杂了。当然，出现这种情况的可能性很小。但如果很不幸这种情况发生在你的身上，可以通过 Google 或其他搜索引擎来查找答案。只要这个问题你不是世界上第一个遇到的人，总会找到解决方法的。有时可能不是 Android 工程的问题，而是 Google 的事。例如，在 Android SDK 刚升级到 2.3 时，ADT 发布了最新的 8.0.0 版。这个版本也没有太大问题，但由于笔者的一个项目引用了另一个项目（跨工程引用），可能在这一 ADT 版本中对跨工程引用支持得不好，使 Android 工程出现异常。估计是这个 Bug 引

起了全球很多开发人员的关注，Google 的反应也比较快，在发布 ADT 8.0.0 之后不到 24 小时就发布了 ADT 8.0.1，经测试，已经修补了这个 Bug。所以读者在遇到这些棘手的问题时，并不需要慌乱，在全世界的开发人员中总会有人捷足先登，最先遇到这个问题。因此，解决方法就在互联网的某处，只需要耐心寻找，总会有所收获。

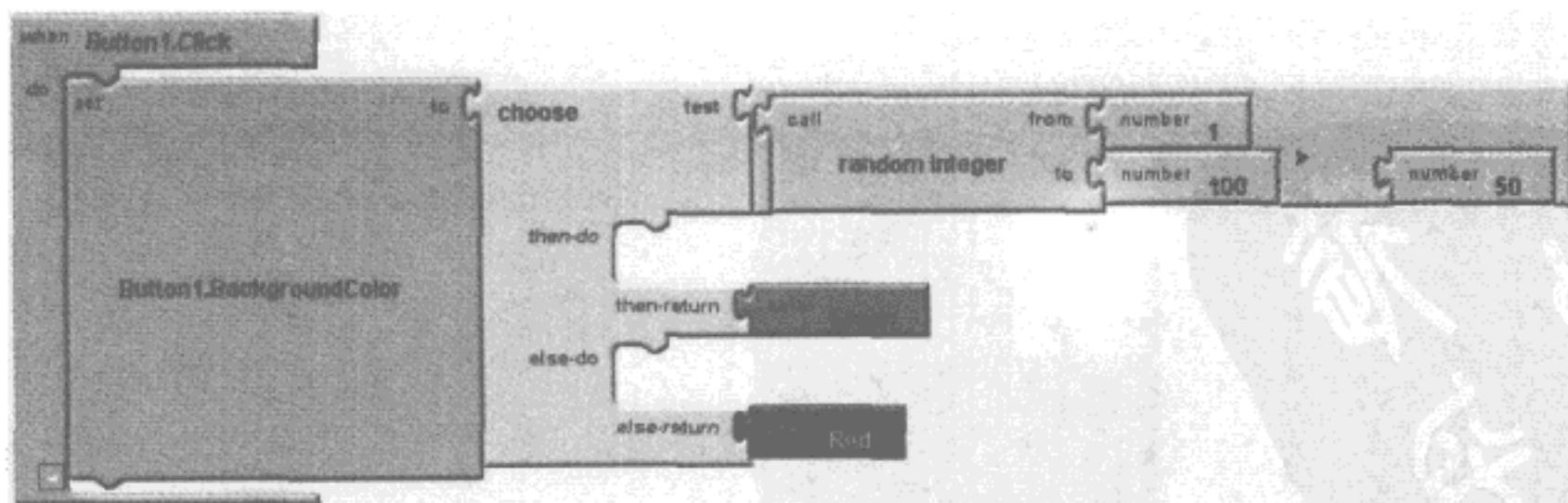
## 2.4 不需要写一行代码的开发工具：AppInventor

在 2.2 节中给出了本书第一个完整的 Android 应用程序。这个例子虽然并不复杂，但对于 Android 的初学者可能还有很多地方弄不清楚，如果要求独立来完成这个程序，很多初学者都做不到。不过这不要紧，本节将为这些读者提供另一个机会来重新完成这个例子。和 2.2 节不同的是，本节的编程方法不需要写一行代码，也不需要理解 Android 的框架结构，甚至不需要知道 Android 是何物。只要会使用鼠标和键盘，就可以在顷刻之间完成 2.2 节的例子。这就是 Google 的最新力作：AppInventor。下面就到了我们见证奇迹的时刻。

### 2.4.1 AppInventor 简介

从 AppInventor 本身的名字来看，可将其拆成两个单词：App 和 Inventor。App 表示应用程序，而 Inventor 是发明家。Inventor 这个单词充满了艺术气息。那么将这两个单词连接来，就是发明程序（要注意，不是编写程序，而是发明程序），而且要用艺术的方法。下面先来看一看用 AppInventor 发明出的程序是个什么样。图 2.33 是一个用 AppInventor 发明的程序。功能很简单，随机产生一个 1 至 100 的整数，如果随机数的值大于 50，则将按钮的背景色设为绿色，否则将按钮的背景色设为红色。

很多读者一开始看到这个程序，肯定以为是拼图，但这的的确确是一段程序，只是看上去有些像艺术品。看到这儿，肯定很多读者和我一样按耐不住了，想快点了解如何使用 AppInventor 来发明 Android 程序。OK，在接下来的部分将为读者逐一揭开心中的谜团。



▲ 图 2.33 用 AppInventor 发明的程序

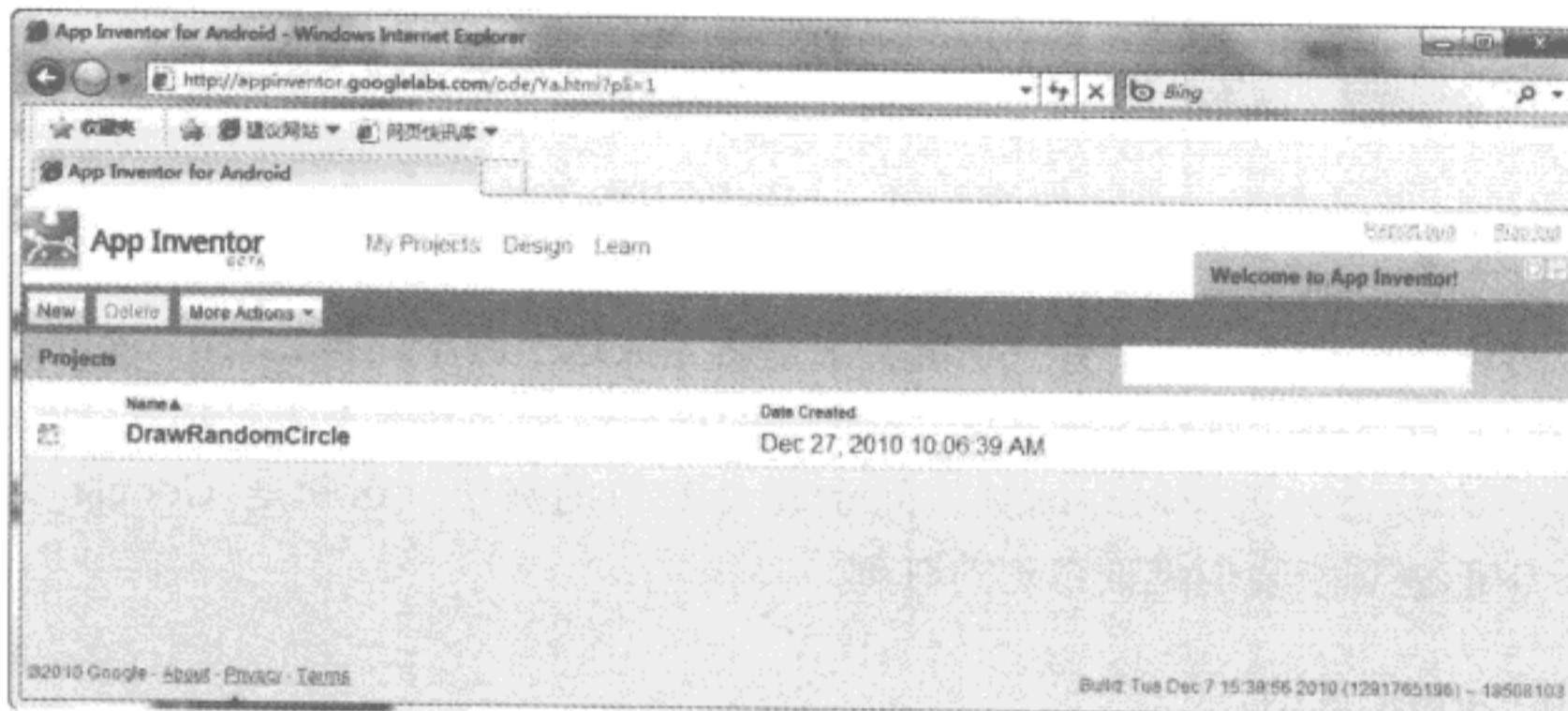
### 2.4.2 AppInventor 的下载和安装

AppInventor 分为两部分：界面设计和代码编写（也可称为代码拼图）。其中界面设计部分是基于 Web 的应用程序。如果只是设计界面，并不需要安装任何的软件或 SDK，只需要在浏览器中访问如下的 URL 即可。

## Android 开发权威指南

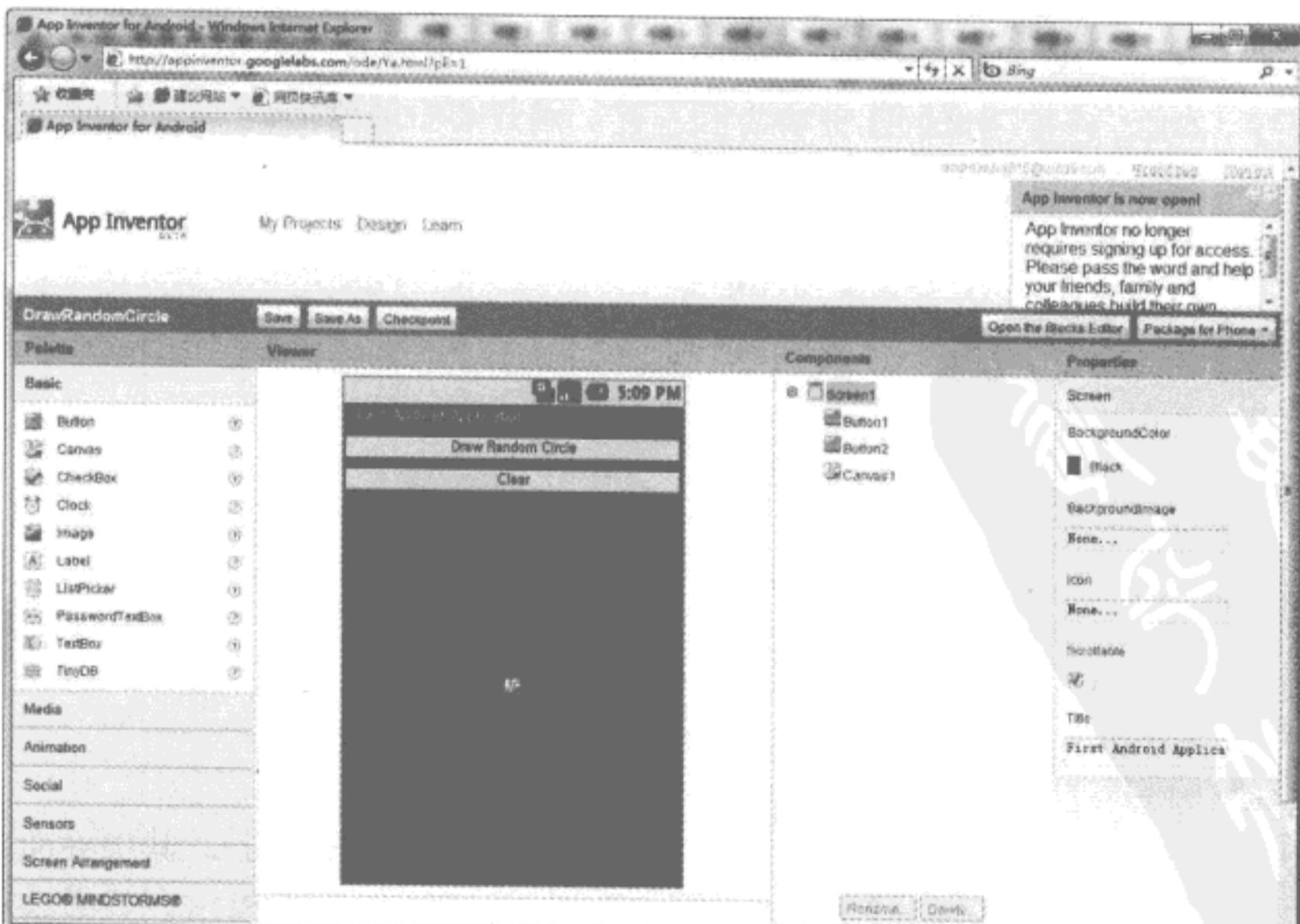
http://appinventor.googlelabs.com

使用 AppIntentor 之前需要有一个 Google Gmail 账号（没有 Gmail 账号的读者赶紧到 <http://mail.google.com> 去注册一个吧）。使用 GMail 登录后，直接访问上面的地址就可以进入如图 2.34 所示的 AppInventor 主界面了。



▲ 图 2.34 AppInventor 的主界面

单击“New”按钮可以新建一个 Android 工程。在本例中已经建立了一个 DrawRandomCircle 工程。单击“DrawRandomCircle”链接进入工程主界面，如图 2.35 所示。在这个界面中已经包含了设计好的 DrawRandomCircle 程序，在下一节将会详细介绍如何使用 AppInventor 设计 Android 应用程序的界面。

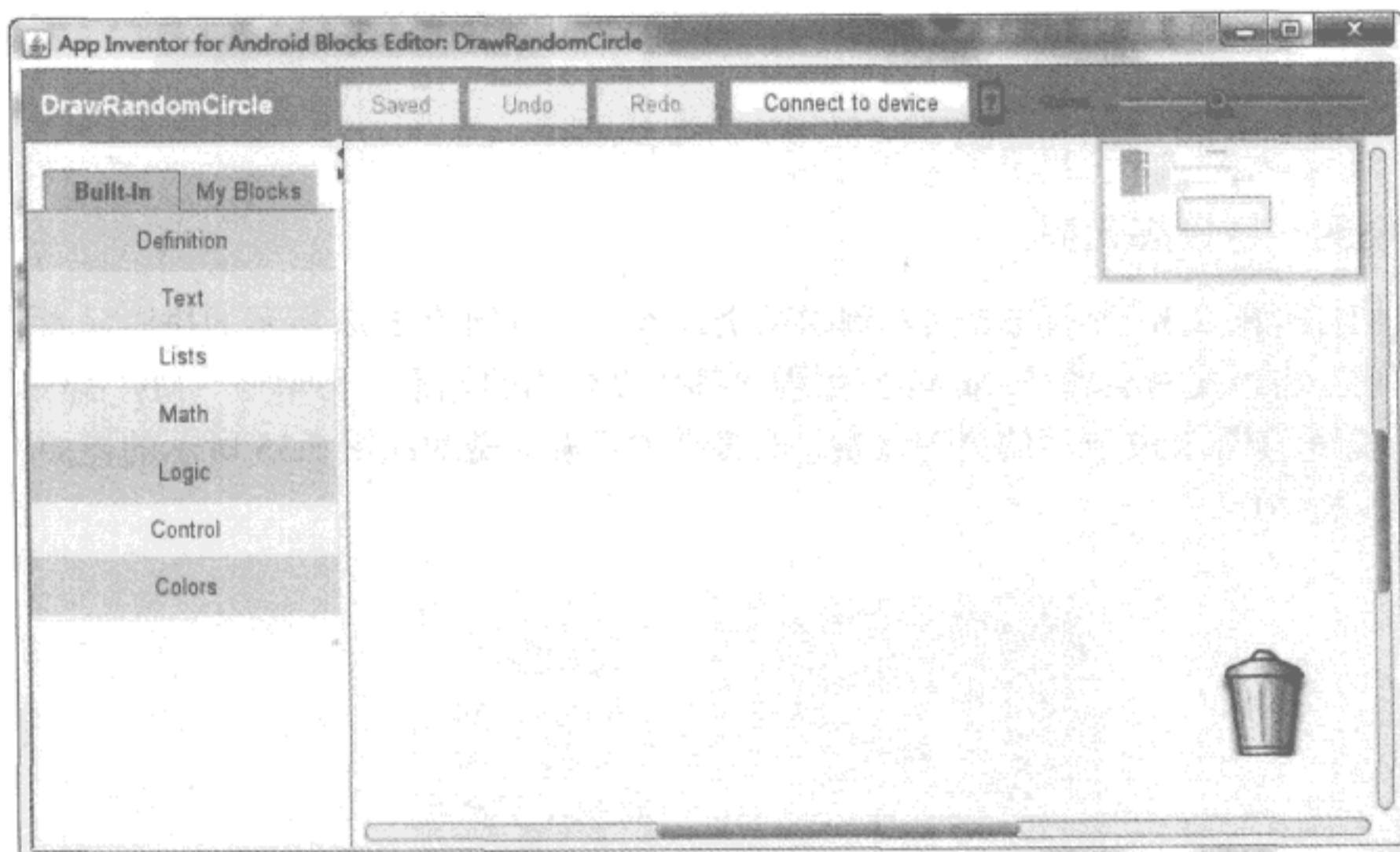


▲ 图 2.35 AppInventor 的设计界面

AppInventor 除了有一个设计界面的工具外，还有一个 C/S 结构的编码工具（更准确地说是拼装代码工具）。这个编码工具需要从如下 URL 下载一个安装程序。其中包含了 Android 模拟器以及相应的命令行工具、开发包等。

<http://appinventor.googlecode.com/learn/setup/index.html>

在安装完该程序后，单击图 2.35 所示的界面右侧的“Open the Blocks Editor”按钮，会首先要求下载或打开一个 JNLP 文件，这是 Java 的部署文件。直接选择打开该文件即可。打开该文件后需要等待一段时间，然后会出现一个如图 2.36 所示的界面。



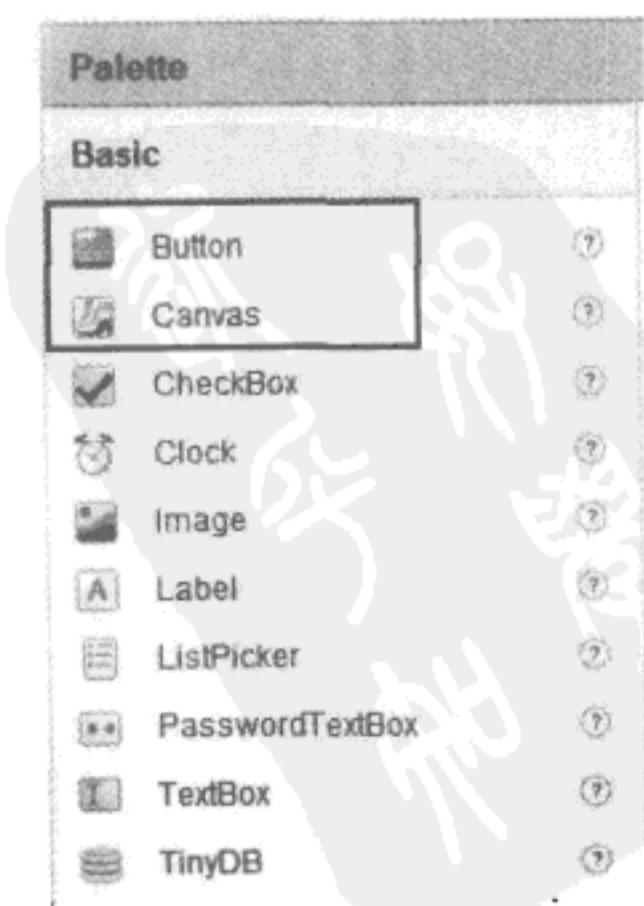
▲ 图 2.36 AppInventor 拼装代码界面

图 2.36 所示的界面左侧的列表是各种 Android 控件，中间空白的区域是拼装代码的区域。在 2.4.4 节我们将介绍如何拼装代码。

### 2.4.3 用拖曳控件的方式设计界面

现在重新回到图 2.35 所示的设计界面。首先需要在界面的垂直方向上放置两个按钮，并且在按钮下方放一个 Canvas。这两个控件在界面左侧的“Basic”页中的前两个位置，如图 2.37 所示。

实际上，在完成上述的设计后，就可以进入下一节进行拼装代码了。但为了和 2.2 节实现的例子的效果完全一样，还需要设置 Button 和 Canvas 的相应属性。我们可以在右侧



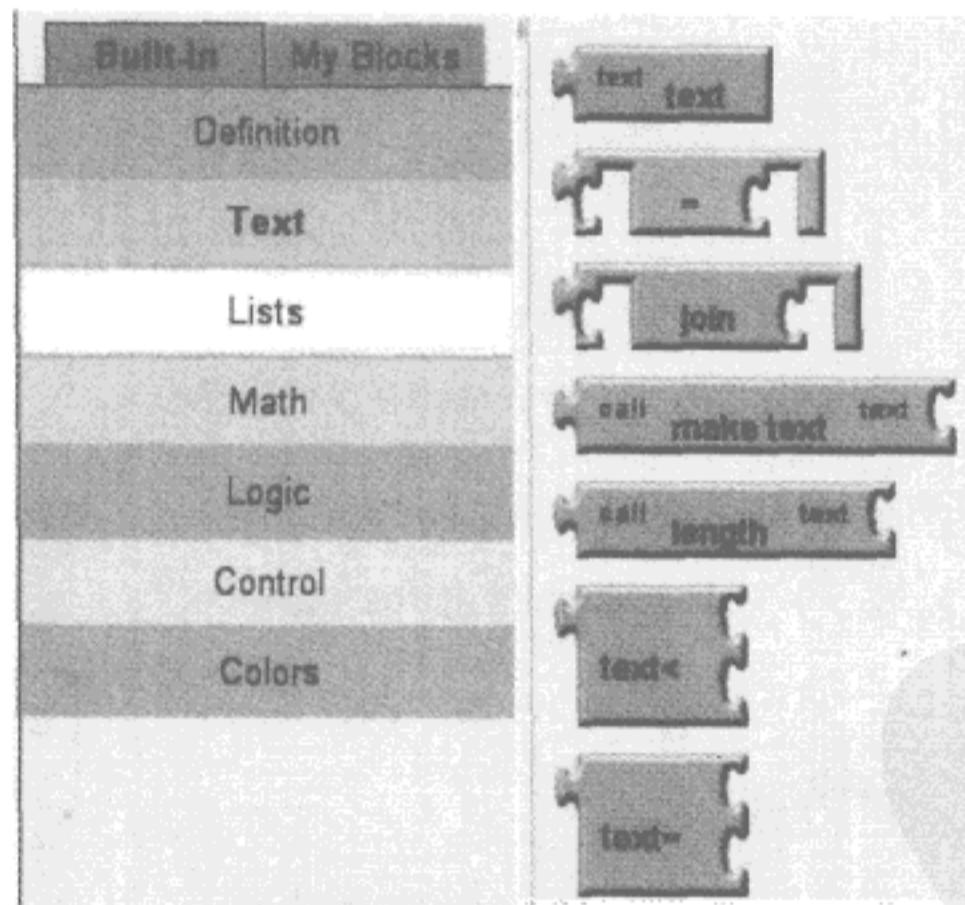
▲ 图 2.37 Button 和 Canvas 控件的位置

的“Properties”区域设置相应的属性。要设置属性的控件有 4 个，分别是 Screen1、Button1、Button2 和 Canvas1，属性值如下：

- Screen1.BackgroundColor: Black;
- Screen1.Title: First Android Application;
- Button1.Text: Draw Random Circle;
- Button1.Width: Fill Parent;
- Button2.Text: Clear;
- Button2.Width: Fill Parent;
- Canvas1.BackgroundColor: Black;
- Canvas1.Width: Fill Parent;
- Canvas1.Height: 350 pixels。

#### 2.4.4 像拼图一样拼装代码

首先我们打开图 2.36 所示的界面左侧的某个标签，会看到图 2.38 所示的拼图元素。这些元素都是公用元素，也就是所有的 Android 应用程序都可以使用的元素。再单击“My Blocks”标签以及 Button1，会看到图 2.39 所示的拼图元素。这些拼图元素就是和在图 2.35 所示的界面中绘制的控件相关的事件和动作。



▲ 图 2.38 公用的拼图元素

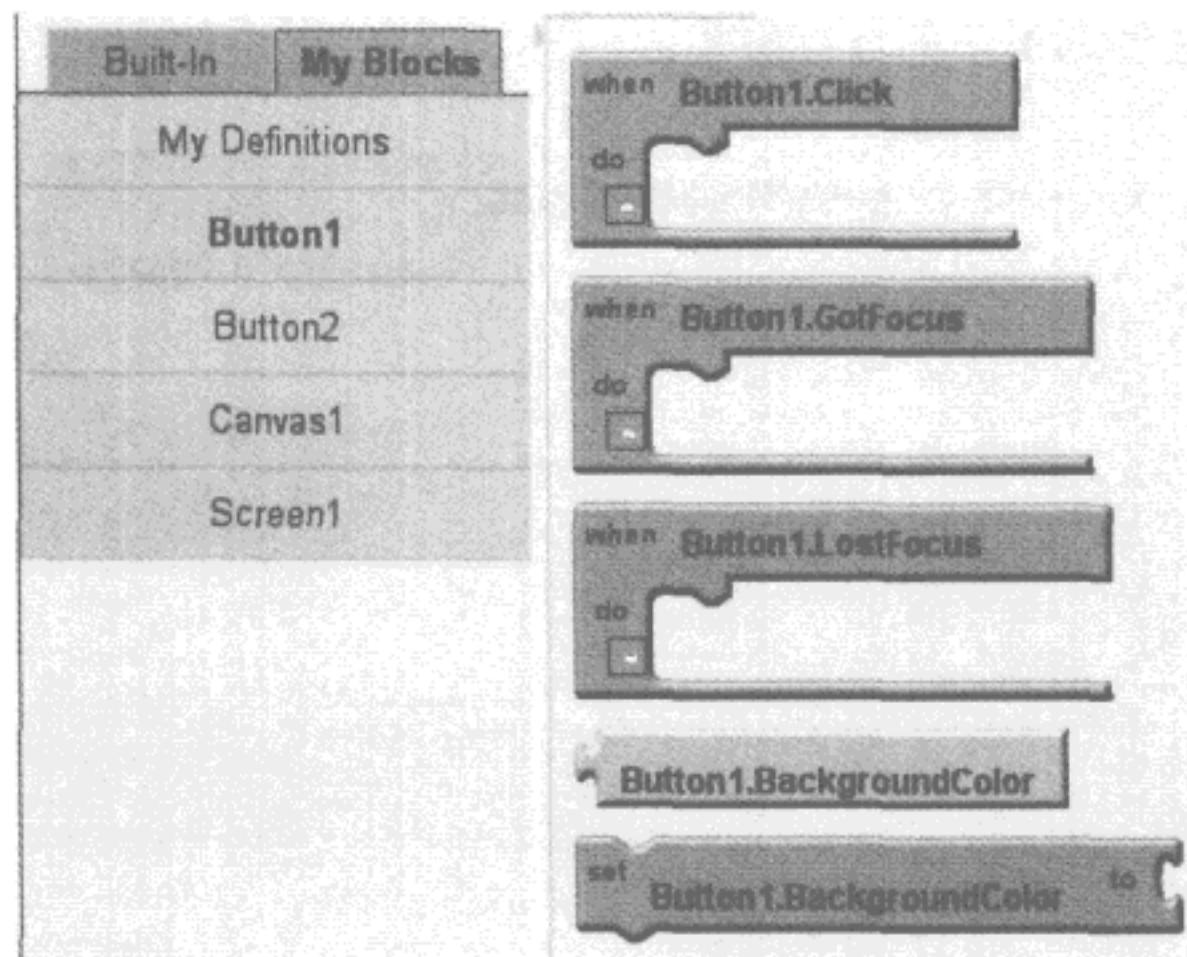
下面来看看如何使用这些拼图元素来拼装代码。首先我们来观察一下每一个拼图元素的形状，我们先来看看 Button1 的 Click 事件的拼图元素（图 2.39 示出拼图元素的第一个），如图 2.40 所示。这个拼图元素有一个半封闭的凹槽，内侧左上角有一个突起。再看一下 Canvas1 的 DrawCircle 拼图元素（在图 2.39 所示的界面 Canvas1 标签中），如图 2.41 所示。DrawCircle 拼图元素上方有一个凹陷的槽，下方有一个突起，在右侧有三个凹陷的槽。最后看看产生随机整数的拼图元素（在图 2.38 所示的界面 Math 标签中），如图 2.42 所示。这个拼图元素自动生成了 3 个更小的拼图元素，

## 第2章 工欲善其事，必先利其器——搭建和使用Android开发环境

其中有两个表示整数的拼图元素。从这3个拼图元素的位置来看，突起的部分正好嵌入到了凹槽中。从这一点可以推断突起是需要嵌入到凹槽中的，也就是说，代码片段需要通过突起和凹槽进行连接，从而形成一个完整的应用程序。以此类推，DrawCircle拼图元素上方的凹槽是可以和Click事件拼图元素内部的突起连在一起的，这样当单击Button1后，就会执行DrawCircle动作。

按照这种方式，我们就可以开始拼装绘制随机实心圆程序了。下面先看看这个程序需要哪些拼图元素，除了图2.40、图2.41和图2.42所示的3个拼图元素外，还需要如下几个拼图元素。

- Canvas1.PaintColor（在Canvas1标签中）；
- Choose（在Control标签中）；
- >（在Math标签中）；



▲图2.39 与拖曳控件相关的拼图元素

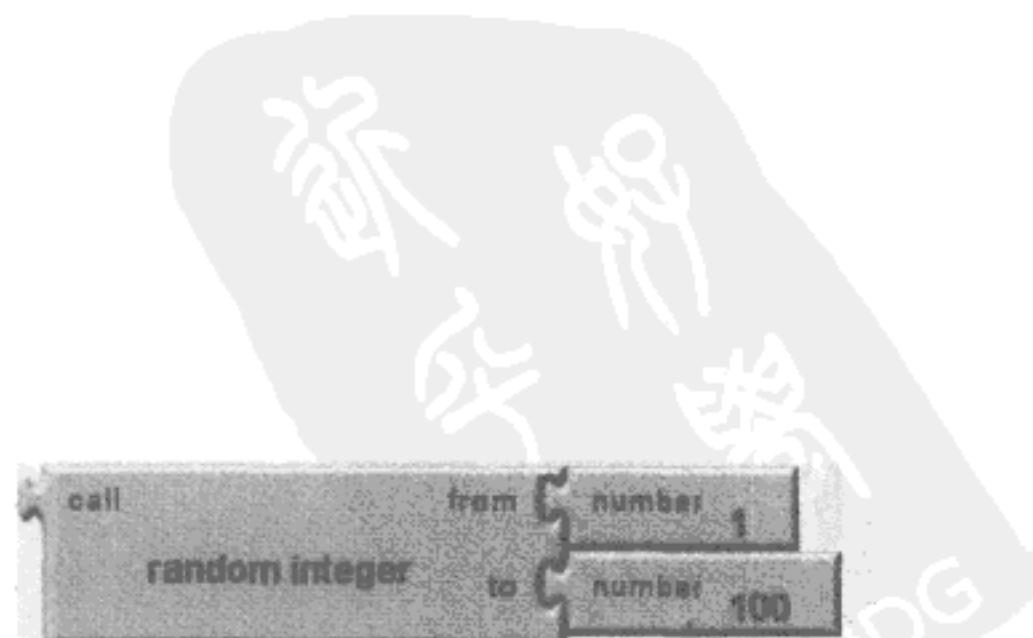
- Number（在Math标签中）；
- Blue（在Color标签中）；
- Red（在Color标签中）；
- Green（在Color标签中）；
- Button2.Click（在Button2标签中）；
- Canvas1.Clear（在Canvas1标签中）。



▲图2.40 Click事件拼图元素

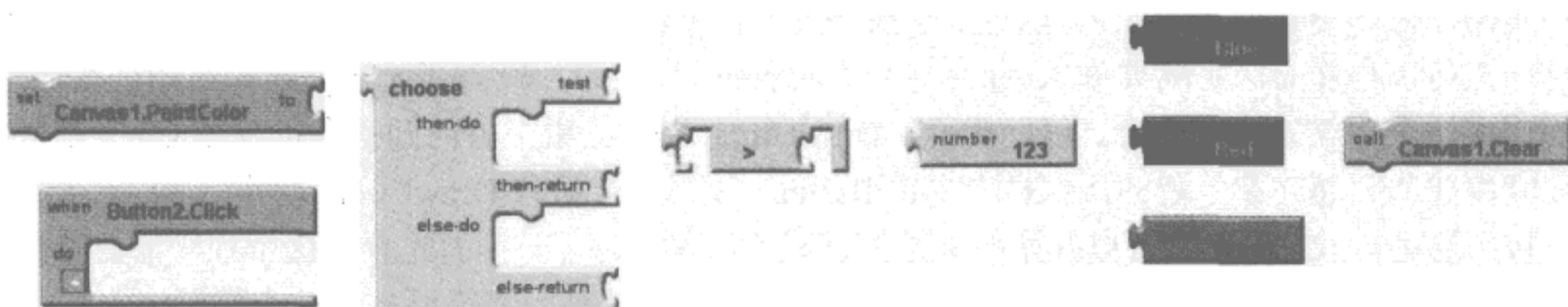


▲图2.41 DrawCircle拼图元素



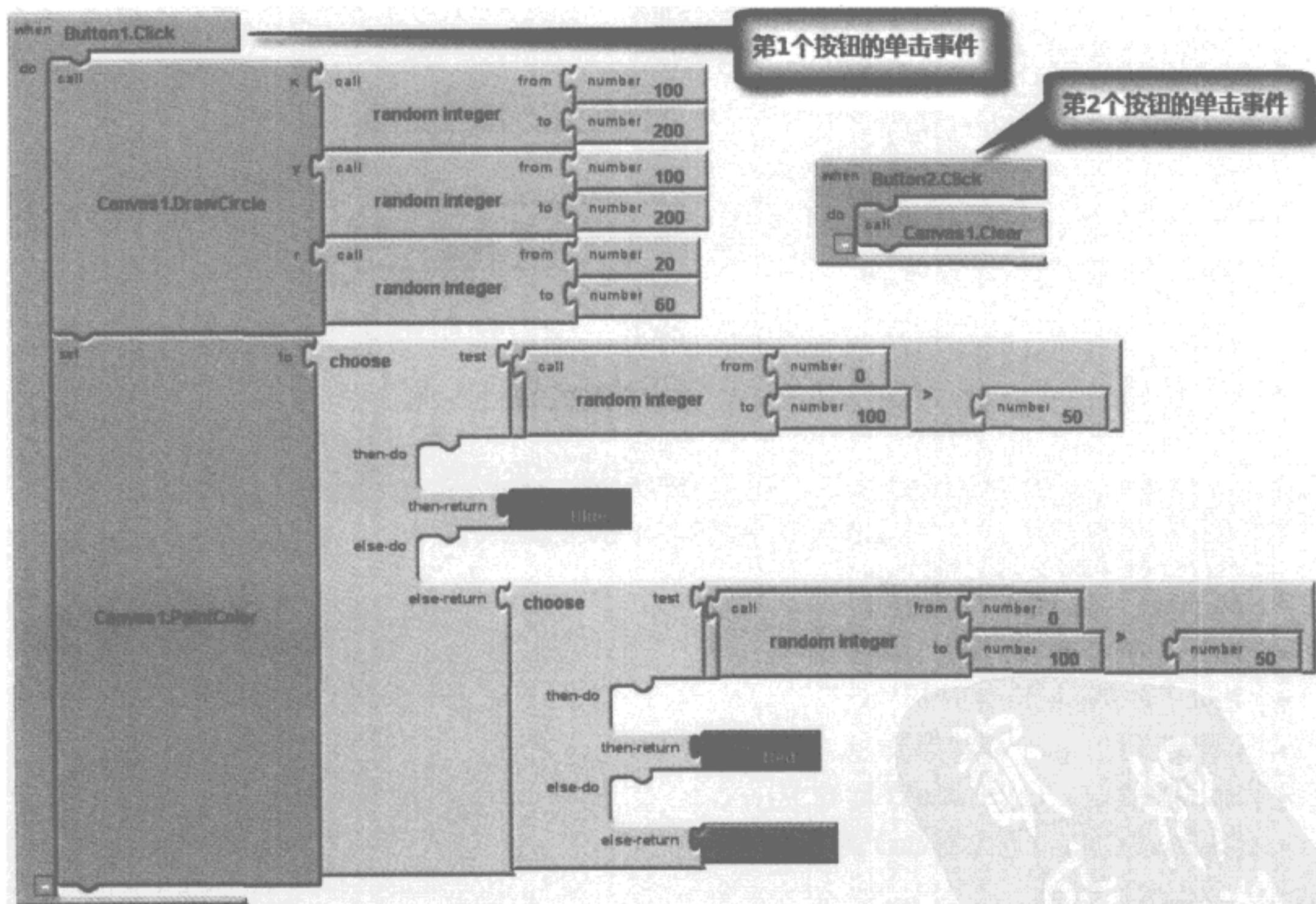
▲图2.42 产生随机数的拼图元素

这些拼图元素的形状如图2.43所示。



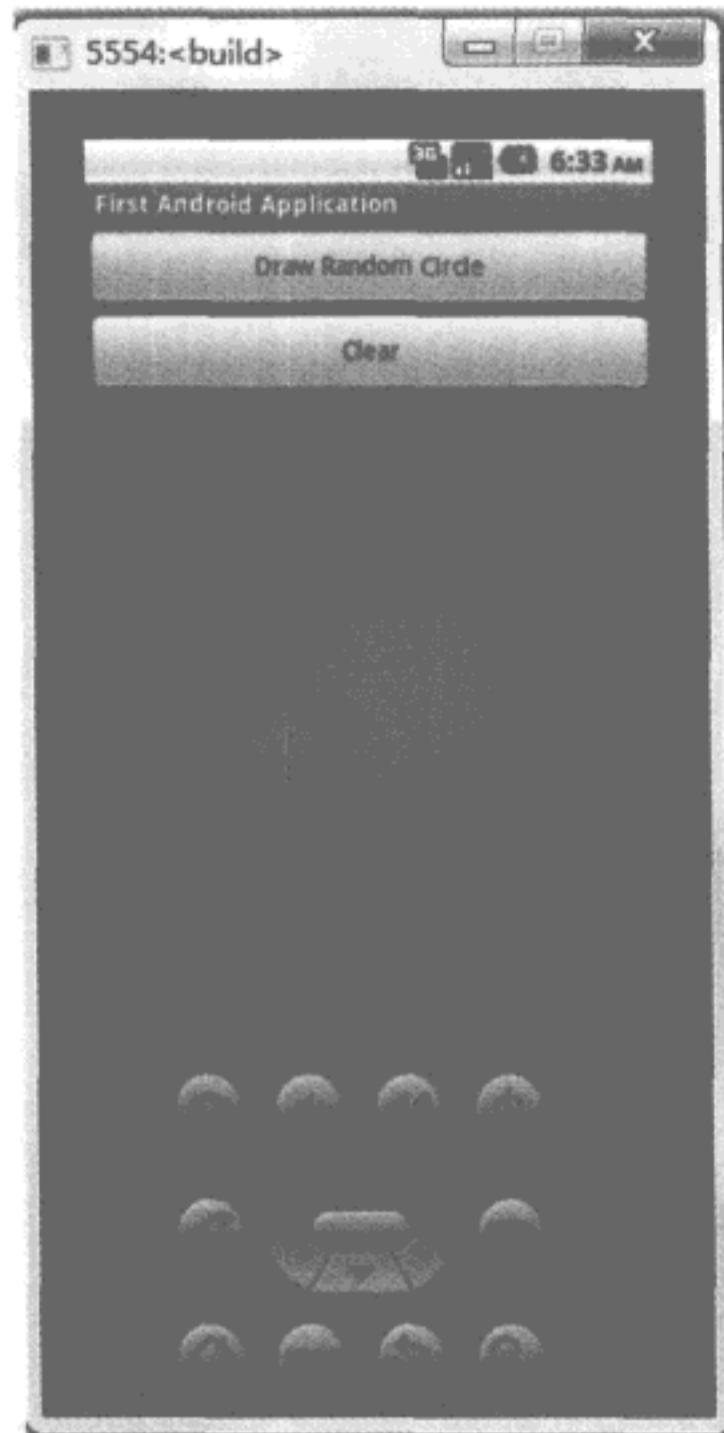
▲ 图 2.43 本例用到的拼图元素

现在我们就可以利用图 2.40 至图 2.43 所示的 12 个拼图元素来拼装本例的代码了，拼装的过程在本节就不详细介绍了，在这里只给出拼装后的结果，如图 2.44 所示。感兴趣的读者可以利用上述 12 个拼图元素按照图 2.44 所示的进行拼装。



▲ 图 2.44 拼装后的代码片段

在拼装完代码后，单击图 2.36 所示的界面右上角的“Connect to Device”按钮。如果这时已经启动了 Android 模拟器，那刚才拼装的程序会直接在模拟器上运行（如果通过 USB 连接了手机，并处于调试状态，也可以直接在手机上运行）。如果还没有启动模拟器，在打开图 2.36 所示的界面时会自动启动 Android 模拟器。在 AppInventor 自带的模拟器中运行的效果如图 2.45 所示。各位读者可以将图 2.45 和图 2.19 进行对比，看看效果是否一样。



▲图 2.45 在 AppInventor 自带的模拟器中运行程序的效果

## 2.5 小结

工欲善其事，必先利其器。搭建 Android 开发环境是学习 Android 的第一步。本章只介绍如何在 Eclipse 中配置 Android 环境。虽然也可以在 Netbeans 或其他 IDE 中开发 Android 程序。但到目前为止，Eclipse ADT + Android SDK 仍然是最成熟，并且使用最多的开发组合。如果读者在阅读本书之前就曾从事过 Java 的开发工作，很可能电脑中已经有 Java 的开发环境和 Eclipse 了，在这种情况下就好办多了，只需要安装 ADT 和 Android SDK 就可以进行 Android 开发了。当然，如果读者的电脑中什么都没有，那只能按照 2.1 节所介绍的步骤逐个安装软件和 SDK 了。

虽然有很多人喜欢编写代码，但天天泡在代码堆里总会或多或少郁闷。偶尔摆弄一会儿玩具也许会让人放松一下。Google 的 AppInventor 就是 Android 领域中的一个大玩具。虽然 AppInventor 目前还无法制作功能复杂的 Android 应用程序，但在玩拼图的同时还制作了一个有趣的 Android 应用，实在是一箭双雕。如果读者与笔者的想法一致，强烈建议看一下 2.4 节。因为在这一节就是我们见证奇迹的时刻。

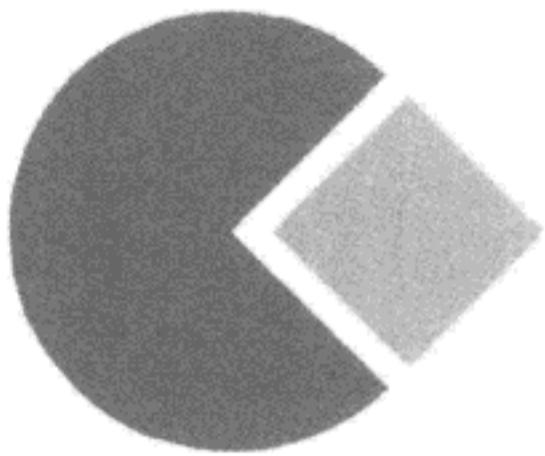




## 第二部分

### 基础篇

- 第3章 千里之行始于足下——  
Android 程序设计基础
- 第4章 我的 UI 我做主——用户界面开发基础
- 第5章 良好的学习开端——  
控件 (Widget) 详解
- 第6章 友好的菜单——Menu 介绍与实例
- 第7章 友好地互动交流——信息提醒  
(对话框、Toast 与 Notification)
- 第8章 移动的信息仓库——数据存储
- 第9章 Android 中的窗口——Activity
- 第10章 全局事件——广播 (Broadcast)
- 第11章 跨应用数据源——Content Provider
- 第12章 一切为用户服务——Service 基础与实例
- 第13章 做好应用桥梁——网络与通信
- 第14章 炫酷你的应用多媒体开发
- 第15章 2D 游戏开发
- 第16章 有趣的 Android 应用



## 第3章 千里之行始于足下—— Android 程序设计基础

通过上一章的学习，我们已经对搭建 Android 开发环境以及开发 Android 应用程序的基本步骤有了一个初步的了解。本章将使读者更近距离接触 Android，从而对 Android 应用程序的整体架构及其核心组件有更深入的了解。本章的内容是学习 Android 必须掌握的。通过对本章的学习，会为更深入地探索 Android 奠定基础。

### 3.1 Android 应用程序框架

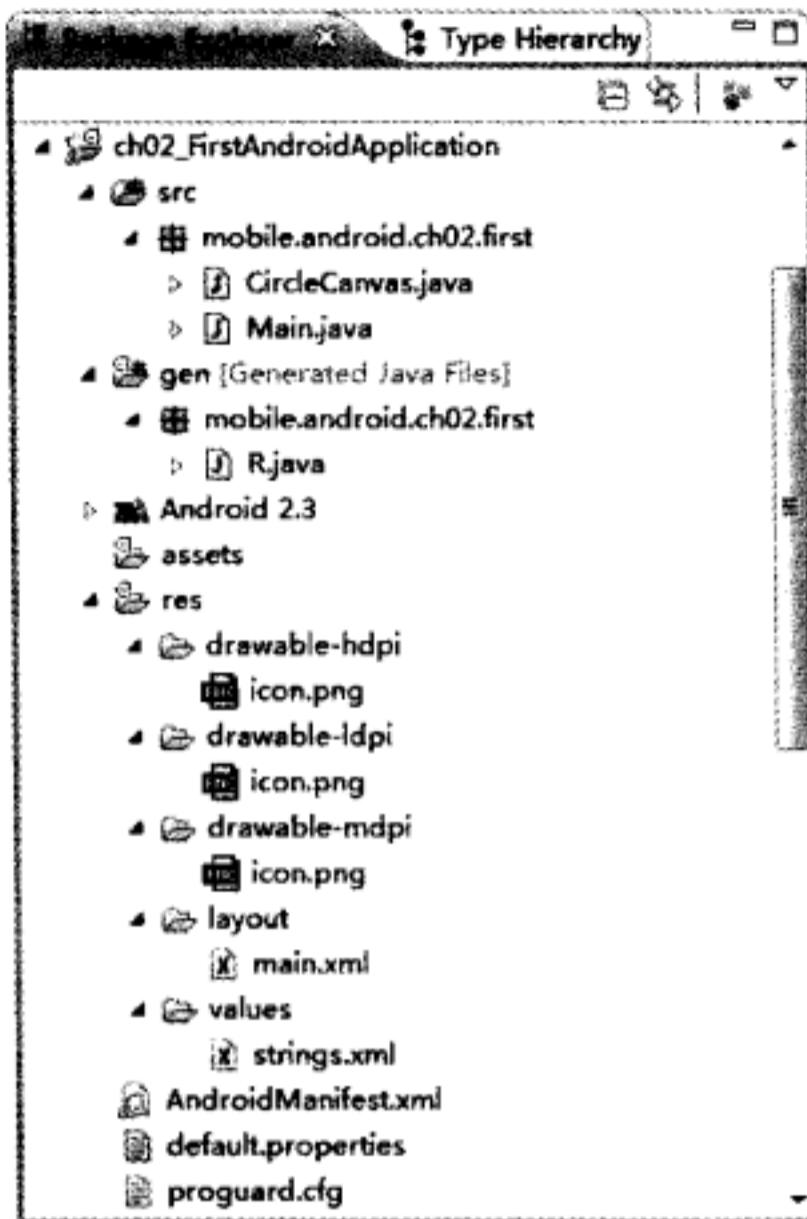
第 2 章中我们编写了一个随机绘制实心圆的 Android 程序。在 Android 工程的结构图（图 2.12）中可以看到，Android 工程有着复杂的目录结构，幸好这个目录结构不需要手工去建立，这一切将由 ADT 自动为我们完成。虽然如此，但了解一下 Android 工程目录中每一部分的作用会对编写 Android 程序有着非常大的帮助。如果读者曾经编写过 Java 控制台、GUI 程序，或是使用过其他语言，如 C/C++、Delphi（Pascal）、Visual Basic 等，会发现。用这些语言开发出来的程序都有一个共同点，就是都需要一个入口，Java、C/C++ 的入口就是众所周知的 main 函数（在面向对象语言中通常称为 main 方法），其他的语言也许没有 main 函数（方法），但仍然会有一个入口程序文件或其他的入口形式，总之，入口一定是单一的，也就是说，程序总是从唯一的一点启动。在 Android 程序中也会采用这样的原则。Android 程序的入口被称为 Main Activity，Activity 相当于 GUI 程序中的 Form（窗体），带有界面的 Android 应用程序一般都需要从 Main Activity 启动。

#### 3.1.1 Android 项目的目录结构

在这一节我们来仔细研究 Android 项目的目录结构，先看一看完整的目录结构，如图 3.1 所示。

从如图 3.1 所示的目录结构可以看出，Android 工程主要包含两个区域：源码区和资源区。与普通的 Java 工程一样，由开发人员自己编写的源代码被放在了 src 目录中（虽然这个目录也可以修改，但建议使用 Eclipse 默认的源码存放目录）。另外一个源码区是 gen 目录。这个目录里的源代码是自动生成的。最常见的是资源类 R（还可以自动生成其他的 Java 类，如根据 AIDL 服务定义文件生成的 Java 类，AIDL 服务定义文件将在后面详细介绍）。这个类在每一个 Android 工程中都会存在，因为在任何的 Android 工程中都会或多或少地有一些资源。每一个资源都会在 R 类中生成唯一的资源 ID。在 3.2 节将介绍 Android 中包含了哪些资源，在这里先看一下 R 类的代码，以便了解 Android 资源是如何与 Java 类中的 ID 对应的。下面的代码是上一章的随机绘制实心圆

工程中的 R 类。



▲ 图 3.1 Android 工程的目录结构

```
package mobile.android.first;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;           // 图像资源 ID
    }
    public static final class layout {
        public static final int main=0x7f030000;           // 布局资源 ID
    }
    public static final class string {
        public static final int app_name=0x7f040001;       // 字符串资源 ID
        public static final int hello=0x7f040000;          // 字符串资源 ID
    }
}
```

上面的代码中有 4 个资源 ID，图像资源 ID 和布局资源 ID 各 1 个以及 2 个字符串资源 ID。我们可以将这段代码与如图 3.1 所示的目录结构进行对照。在 res 目录中有 3 个图像资源目录 drawable-hdpi、drawable-mdpi 和 drawable-ldpi。实际上，需要将同一个图像文件按不同分辨率在这 3 个文件中各放一份。这么做的目的是为了满足不同分辨率的手机屏幕的需要。其中 drawable-hdpi 是高分辨率手机屏幕所使用的图像资源目录，高分辨率主要指 WVGA (480 \* 800) 和 FWVGA (480 \* 854) 或其他更高的分辨率。drawable-mdpi 是中分辨率手机屏幕所使用的图像资源目录，中分辨率是指 HVGA (320 \* 480) 或其他类似大小的分辨率。drawable-ldpi 是低分辨率手机屏幕所使用的

图像资源目录。低分辨率指 QVGA (240 \* 320) 以及其他类似的分辨率。程序运行时，Android 系统会检测当前手机的分辨率，并根据当前的分辨率到不同的图像资源目录中找相应的图像资源。

### **扩展学习：图像资源目录**

在 Android 1.5 或其他较低 Android 版本的工程目录中默认只有一个 drawable 目录，而不像 Android 2.3 工程目录有 3 个 drawable 目录。实际上，在 Android 1.5 的工程目录中的 drawable 目录也可以扩展成 drawable-hdpi、drawable-mdpi 和 drawable-ldpi。只是与 Android 1.5 相对应的 ADT 没有自动生成这 3 个目录，而只生成了一个 drawable 目录。在 Android 2.3 中为了更好地适应分辨率，所以默认就生成了适应不同屏幕分辨率的图像资源目录。这主要是因为在 Android 1.5 时代屏幕分辨率大多都是 320 \* 480，或更小的 240 \* 320。像 WVGA (480 \* 800) 这么大的分辨率的手机基本上是没有的，因此，只要在 drawable 中放置适应 320 \* 480 分辨率的图像资源就足够了。然而到了 Android 2.3 时代（实际上，从 Android 2.1 开始就自动生成这 3 个目录了），各种屏幕分辨率的手机（WVGA、HVGA 和 QVGA，除了这 3 种分辨率外，从 Android 2.2 开始，还支持更大的分辨率，适合于这种分辨率的图像资源放在 drawalbe-xhdpi 中，由于更大分辨率的屏幕主要用在平板电脑上，所以在手机中并不是很常用）充斥着 Android 世界，这也给很多没有经验的程序员造成了麻烦。因此，从 Android 2.1 开始的 ADT 就会自动生成分别适应 3 种不同分辨率大小的图像资源目录，以免只使用一个 drawable 目录而造成由于屏幕分辨率过大而使图像失真的后果。当然，如果手机的分辨率固定，也可以只使用 drawable 来保存图像资源文件。

除了 drawable 资源目录外，还有很多保存其他资源的目录，如 layout（保存布局文件）、values（保存字符串、数组、颜色等资源）等。这些资源将在后面详细介绍。

#### 3.1.2 AndroidManifest.xml 文件的结构

在图 3.1 所示的目录结构可以看出，除了众多的资源目录外，还有一个 AndroidManifest.xml 文件显得格外引人注目。实际上，这个文件是整个 Android 应用程序的核心。一个 Android 应用程序可以没有 Activity（也就是没有界面的程序），但必须要有一个 AndroidManifest.xml 文件。这个文件是 Android 应用程序中的核心配置文件，而且该文件必须在 Android 工程的根目录。AndroidManifest.xml 在 Android 应用程序中主要做了如下的工作。

- 定义应用程序的 Java 包。这个包名将作为应用程序的唯一标识。在 DDMS 透视图的“File Explorer”视图中可以看到 data\data 目录中的每一个目录名都代表着一个应用程序，而目录名本身就是在 AndroidManifest.xml 文件中定义的包名。
- 将在 3.3 节介绍的 4 个应用程序组件在使用之前，必须在 AndroidManifest.xml 文件中定义。定义的信息主要是与组件对应的类名以及这些组件所具有的能力。通过 AndroidManifest.xml 文件中的配置信息可以让 Android 系统知道如何处理这些应用程序组件。
  - 确定哪一个 Activity 将作为第一个运行的 Activity（也就是 Main Activity）。
  - 在默认情况下，Android 系统会限制使用某些 API，因此，需要在 AndroidManifest.xml 文件中为这些 API 授权后才可以使用它们。

- 可以授权与其他的应用程序组件进行交互。
- 可以在 `AndroidManifest.xml` 文件中配置一些特殊的类，这些类可以在应用程序运行时提供调试及其他的信息。但这些类只在开发和测试时使用，当应用程序发布时这些配置将被删除。
- 定义了 Android 应用程序所需要的最小 API 级别，Android 1.0 API Level = 1；Android 1.1 API Level = 2；Android 1.5 API Level = 3；Android 1.6 API Level = 4；Android 2.0 API Level = 5；Android 2.0.1 API Level = 6；Android 2.1 API Level = 7；Android 2.2 API Level = 8；Android 2.3 API Level = 9。
- 指定应用程序中引用的程序库。

下面来看一下 `AndroidManifest.xml` 文件的标准格式。其中所涉及到的标签会在后面的内容中逐渐讲到。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

## 3.2 Android 应用程序中的资源

在 3.1.1 节已经接触到了几种资源，这些资源所在的目录都是 res 的子目录。在 Android 应用程序生成 apk 时，这些资源将被封装在 apk 文件中。当然，除了 res 目录，在 Android 工程根目录中的 assets 目录也可以保存资源文件，关于如何使用 res 及 assets 目录中的资源将在后面的内容中详细介绍。Android 应用程序可以包含的常用资源如表 3.1 所示。

表 3.1 Android 应用程序中的常用资源

资源种类	所在目录	描述
动画（Animation）	帧（Frame）动画：res/drawable 补间（Tween）动画：res/anim	定义动画文件
颜色状态列表（Color State List）	res/color	定义根据视图（View）状态变化的颜色资源
可拉伸图像（Drawable）	res/drawable	使用 Android 所支持的图像格式或 XML 文件定义不同的图形
布局（Layout）	res/layout	定义描述应用程序 UI 的布局
菜单（Menu）	res/menu	定义应用程序菜单的内容
字符串（String）	res/values	定义字符串，可以通过 R.string 访问相应的资源
颜色（Color）	res/values	定义颜色值，可以通过 R.color 访问相应的资源
尺度（Dimen）	res/values	定义宽度、高度、位置等尺度信息，可以通过 R.dimention 访问相应的资源
风格（Style）	res/values	定义 UI 元素的格式和外观。可通过 R.style 类访问相应的资源
XML	res/xml	基于 XML 格式的资源。例如，PreferenceActivity 使用的描述配置界面的资源文件
RAW	res/raw	保存任意二进制文件。保存在该目录中的文件未被压缩，因此可通过 InputStream 从 apk 文件提取出来直接使用
ASSETS	assets	保存任意二进制文件。该目录与 res/raw 目录类似。但在 res/raw 目录中不能建立子目录，而在 assets 目录中可以建立任意层次的子目录（目录的层次只受操作系统的限制）

除了表 3.1 所示的资源外，还有像 Integer、Bool 这样的资源，这些资源都保存在 res/values 目录中，分别用 R.bool 和 R.integer 来引用。本节只是简单介绍一下 Android 所涉及的资源，在后面的章节将对 Android 资源进行详细介绍。

## 3.3 Android 的应用程序组件

在 Android 程序中没有入口点（Main 方法），取而代之的是一系列的应用程序组件，这些组件都可以单独实例化。本节将介绍 Android 支持的 4 种应用组件的基本概念。应用程序对外共享功能

一般也是通过这4种应用程序组件实现的。

### 3.3.1 Activity (Android 的窗体)

Activity 是 Android 的核心类，该类的全名是 android.app.Activity。Activity 相当于 C/S 程序中的窗体（Form）或 Web 程序的页面。每一个 Activity 提供了一个可视化的区域。在这个区域可以放置各种 Android 控件，例如，按钮、图像、文本框等。

在 Activity 类中有一个 onCreate 事件方法，一般在该方法中对 Activity 进行初始化。通过 setContentView 方法可以将 View 放到 Activity 上。绑定后，Activity 会显示 View 上的控件。

一个带界面的 Android 应用程序可以由一个或多个 Activity 组成。至于这些 Activity 如何工作，或者它们之间有什么依赖关系，则完全取决于应用程序的业务逻辑。例如，一种典型的设计方案是使用一个 Activity 作为主 Activity（相当于主窗体，程序启动时会首先显示这个 Activity），在这个 Activity 中通过菜单、按钮等方式显示其他的 Activity。在 Android 自带的程序中有很多都是这种类型的。

每一个 Activity 都会有一个窗口，在默认情况下，这个窗口是充满整个屏幕的，也可以将窗口变得比手机屏幕小，或者悬浮在其他的窗口上面。Activity 窗口中的可视化组件由 View 及其子类组成，这些组件按照 XML 布局文件中指定的位置在窗口上进行摆放。

### 3.3.2 Service (服务)

服务没有可视化接口，但可以在后台运行。例如，当用户进行其他操作时，可以利用服务在后台播放音乐，或者当来电时，可以利用服务同时进行其他操作。服务类必须从 android.app.Service 继承。

现在举一个非常简单的使用服务的例子。在手机中会经常使用播放音乐的软件，在这类软件中往往会有循环播放或随机播放的功能。虽然在软件中可能会有相应的功能（通过按钮或菜单进行控制），但用户可能会一边放音乐，一边在手机上做其他的事，例如，与朋友聊天、看小说等。在这种情况下，用户不可能当一首音乐放完后再回到软件界面去进行重放的操作。因此，可以在播放音乐的软件中启动一个服务，由这个服务来控制音乐的循环播放，而且服务对用户是完全透明的，这样用户完全感觉不到后台服务的运行，甚至可以在音乐播放软件关闭的情况下，仍然可以播放后台背景音乐。

除此之外，其他的程序还可以与服务进行通信。当与服务连接成功后，就可以利用服务中共享出来的接口与服务进行通信了。例如，控制音乐播放的服务允许用户暂停、重放、停止音乐的播放。

### 3.3.3 Broadcast Receiver (广播接收器)

广播接收器组件的唯一功能就是接收广播动作，以及对广播动作做出响应。有很多时候，广播动作是由系统发出的，例如，时区的变化、电池的电量不足、收到短信等。除此之外，应用程序还可以发送广播动作，例如，通知其他的程序数据已经下载完毕，并且这些数据已经可以使用了。

一个应用程序可以有多个广播接收器，所有的广播接收类都需要继承 android.content.BroadcastReceiver 类。

广播接收器与服务一样，都没有用户接口，但在广播接收器中可以启动一个 Activity 来响应广播动作，例如，通过显示一个 Activity 对用户进行提醒。当然，也可以采用其他的方法或几种方法的组合来提醒用户，例如，闪屏、震动、响铃、播放音乐等。

### 3.3.4 Content Provider ( 内容提供者 )

内容提供者可以为其他应用程序提供数据。这些数据可以保存在文件系统中，例如，SQLite 数据库或任何其他格式的文件。每一个内容提供者是一个类，这些类都需要从 android.content.ContentProvider 类继承。

在 ContentProvider 类中定义了一系列的方法，通过这些方法可以使其他的应用程序获得内容提供者所提供的数据。但在应用程序中不能直接调用这些方法，而需要通过 android.content.ContentResolver 类的方法来调用内容提供者类中提供的方法。

在 Android 系统中很多内嵌的应用程序，如联系人、短信等，都提供了 ContentProvider。其他的应用程序通过这些 Content Provider 可以对系统内部的数据实现增、删、改操作。例如，可以将指定电话号的短信内容从系统数据库中删除，并将该短信内容加密保存在自己的数据库中。这些删除系统短信的操作就需要通过 Content Provider 来完成。

## 3.4 Android 程序的 UI 设计

UI 布局设计是在 Android 程序的总体工作量中占了很大比重。大多数开发人员可能更习惯于直接编辑 XML 布局文件，这样做虽然更灵活，但效率较低。为了提高工作效率，可以考虑使用 ADT 自带的可视化 UI 设计器，或采用两种方式结合的方法来设计 UI 布局。除此之外，还有很多第三方的 UI 设计器可以使用，例如，DroidDraw 就是一款比较出色的 UI 设计器。

### 3.4.1 手工配置 XML 布局文件

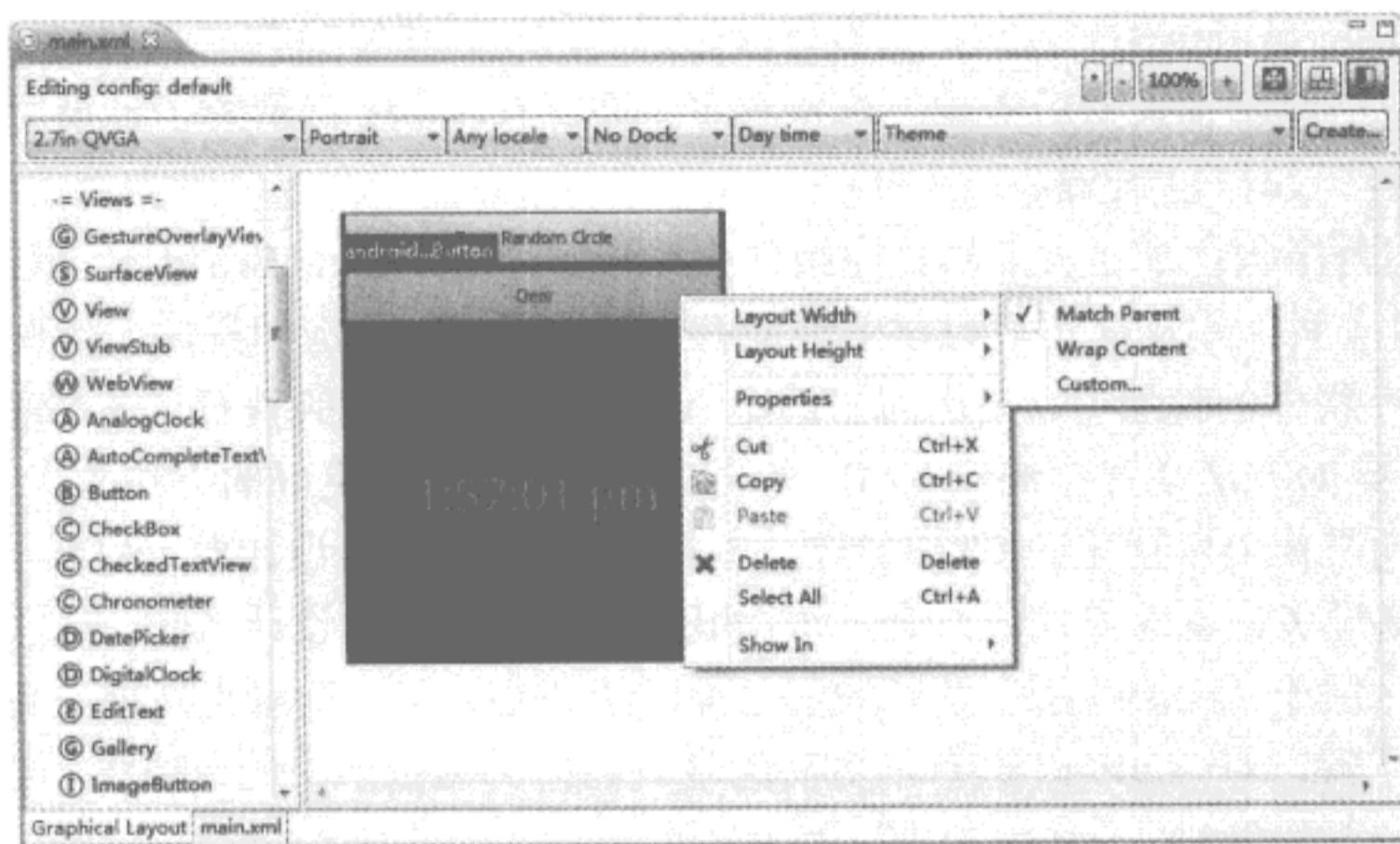
在 2.2.3 小节已经使用了手工的方式配置了一个 XML 布局文件 (main.xml)。手工配置布局文件的操作并不复杂，只要打开布局文件（双击布局文件可显示布局文件编辑器），按照一定的格式编辑布局文件即可。如果忘了某个控件的标签名或标签的属性，可以通过 Content Assist 快捷键显示所有的标签或属性。图 3.2 所示是编辑 XML 布局文件的界面。



▲ 图 3.2 XML 布局文件的编辑界面

### 3.4.2 ADT 自带的可视化 UI 设计器

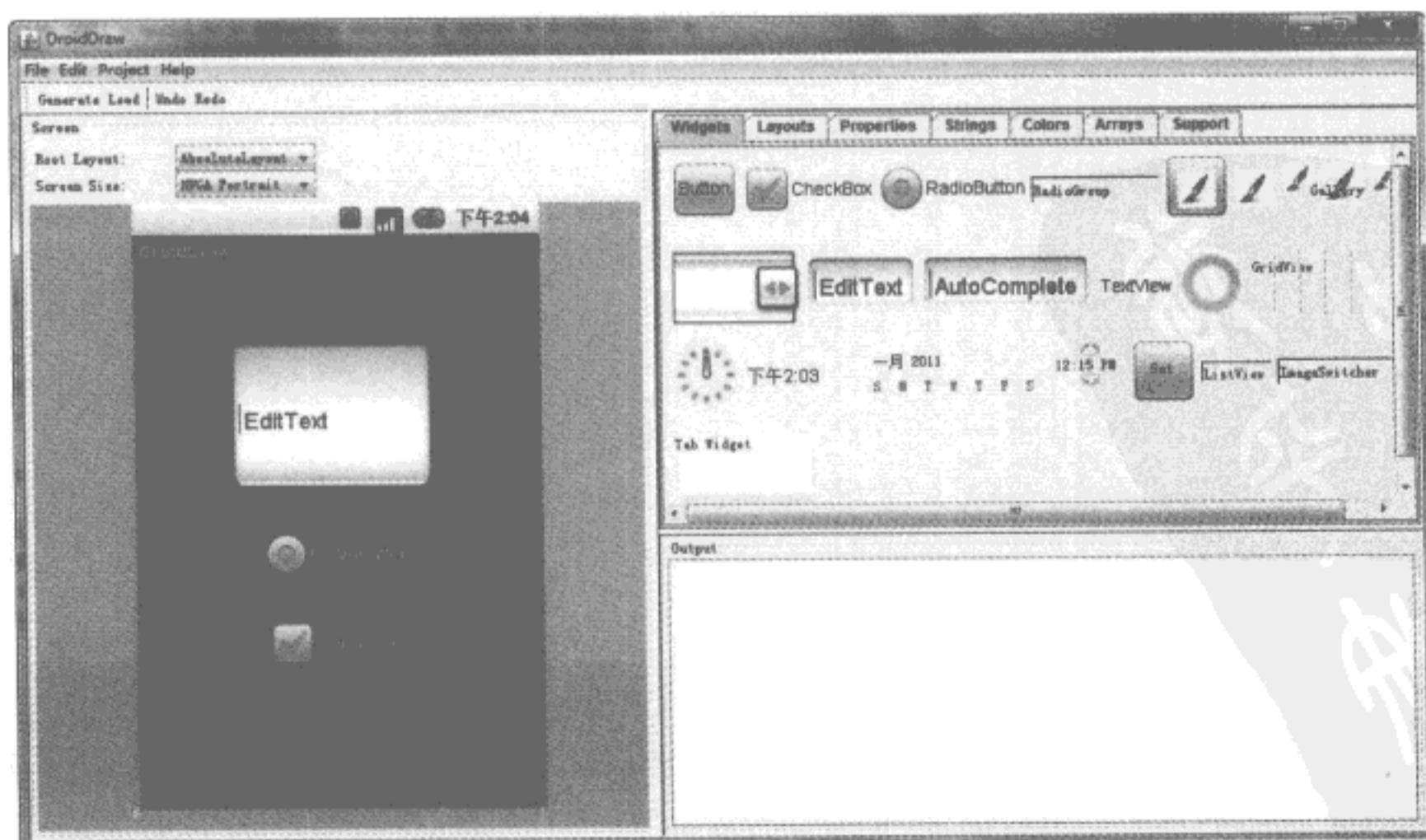
通过单击如图 3.2 所示界面下方的“Graphical Layout”标签，可以进入 ADT 自带的可视化 UI 设计器的界面。界面左侧是控件列表，右侧是可视化 UI 设计区域。通过控件的右键菜单的相应菜单项可以设置控件的属性，如图 3.3 所示。



▲ 图 3.3 可视化 UI 设计器

### 3.4.3 使用 DroidDraw 设计 UI 布局

DroidDraw 是一款第三方的 Android UI 设计工具。DroidDraw 可以通过拖曳的方式设计 UI，并可以将设计的结果保存成 XML 布局文件，生成 apk 文件以及直接在模拟器或手机上运行。读者可以从 <http://code.google.com/p/droiddraw> 下载 DroidDraw，其主界面如图 3.4 所示。



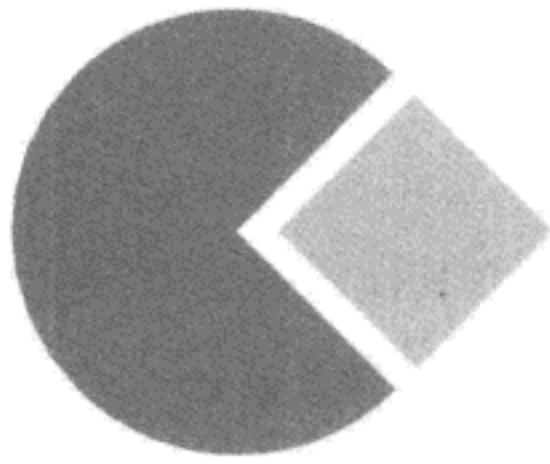
▲ 图 3.4 DroidDraw 的主界面

### 3.5 小结

本章主要介绍了 Android 应用程序的框架、资源、应用程序组件以及如何设计 Android UI。虽然 Android 工程的目录结构较复杂，但并不需要我们手工去建立。如果使用 ADT 来建立工程，会自动生成一个默认的目录结构。当然，我们可以根据需要对这个默认生成的目录结构进行修改。在 Android 应用程序中包含了大量的资源。这些资源主要保存在 res 目录的子目录中，在生成 apk 文件时会将这些资源一起打包到 apk 文件中。

Android 应用程序包含了 4 大应用程序组件：Activity、Service、Broadcast Receiver 和 Content Provider，通过这 4 种应用程序组件可以实现所有类型的 Android 应用程序。因此，一定要认真掌握这 4 种应用程序组件的使用方法，因为它们是 Android 应用程序的基石。在本章的最后介绍了如何设计 UI。最直接的方法是直接编辑 XML 布局文件。不过还有更好的方法来完成全部或部分 UI 的设计工作，这就是使用可视化的 UI 设计器。本章介绍了两个 UI 设计器：ADT 自带的 UI 设计器和 DroidDraw。通过多种设计方式的结合，可以既有效，又灵活地设计 Android 的 UI。

到本章为止，关于 Android 所有基础的部分都已经介绍完了。读者也从前三章中对 Android 的介绍有了大概的了解，对 Android 也有了更高的学习热情。在接下来的章节中，将陆续对 Android 中各种技术从理论到实践进行详细的讲解，再加上每一章以及后面两部分中大量的实例，会使读者以更轻松地方式学习 Android。让我们继续探索 Android 的精彩世界吧！



## 第4章 我的 UI 我做主—— 用户界面开发基础

在前几章已经多次接触到了 Activity，对 Activity 也有了一定的了解。Activity 是 Android 中唯一可视化的应用程序组件。然而，要编写更复杂的 Android 应用程序，还需要对 Activity 做进一步地研究。本章将带领读者深入了解隐藏在 Activity 中的奥秘。除了 Activity，视图（View）也是 Android 界面开发中必不可少的元素，而且 Activity 和 View 总是成对出现，那么这是为什么呢？本章将会揭晓答案。在本章的最后还详细介绍了 Android 支持的 5 种布局以及如何处理 UI 事件。

### 4.1 Activity 的使用方法

Activity 是 Android 中最核心的应用程序组件，也是大多数程序必须使用的用于显示界面的组件。因此，更好地掌握 Activity 的使用方法对更深入学习 Android 至关重要。本节将对 Activity 的各种使用方法做详细的阐述。

#### 4.1.1 创建 Activity

在 2.2 节的例子中由 ADT 自动生成了一个 Main 类，也就是程序的 Main Activity。我们从这个类的代码可以看出，Main 是 Activity 类的子类，而且覆盖了 Activity 类的 onCreate 方法。创建 Activity 的步骤如下。

(1) 建立一个普通的 Java 类，该类必须从 Activity 类或其子类中继承。除了 Activity 类外，还有像 ListActivity、TabActivity 一样的类，这些类也是 Activity 的子类。为了快速创建某些固定形式的 Activity，在 Android SDK 中提供了一些不需要动态装载 View 的 Activity 类，例如，如果要创建一个只包含列表的 Activity，就可以创建一个继承于 ListActivity 的类，这样就不需要再使用 XML 布局文件或 Java 代码向 Activity 中添加 ListView 控件了。关于 ListActivity 和 TabActivity 这两个类将在第 5 章详细介绍。在这里只要知道有这样一组 Activity 类即可。

(2) 覆盖 Activity 类的 onCreate 方法。实际上，一个类只要继承了 Activity 类就可以当成一个 Activity 来使用了。但这个类中并没有任何控件，因此，显示效果除了屏幕顶端有一个默认的标题栏外什么都没有。要想在 Activity 中添加控件，最直接的方法就是在 onCreate 方法中装载 XML 布局文件或使用 Java 代码来添加控件。如果要覆盖 onCreate 方法，必须调用 Activity 类的 onCreate 方法，也就是 super.onCreate(savedInstanceState)，否则显示 Activity 时会抛出异常。这是由于 Activity 类中没有不带参数的 onCreate 方法。如果不显示调用 super.onCreate(savedInstanceState)，系统会试

图调用 super.onCreate()方法（Activity 类中并没有这个方法）。

（3）在 onCreate 方法中使用 setContentView 装载 View。

在创建 Activity 时应注意如下几点。

- 在 Activity 中初始化控件一般在 onCreate 方法中完成。当然，也可以在其他的方法中完成。但不能在 Activity 的构造方法中初始化控件或装载 View，因为在构造方法中很多初始化的工作还没有完成，无法调用 setContentView 来装载 View，更无法初始化控件。

- 不要在 Activity 类中添加带参数的构造方法，如果非要添加带参数的构造方法，必须包含一个没有参数的构造方法。这是由于 Activity 创建对象并不是由开发人员完成的，而是通过 Intent 对象和 startActivity 方法在系统内部创建的。创建 Activity 对象时使用了没有参数的构造方法，因此，所有的 Activity 类及其子类必须包含一个没有参数的构造方法，如果在 Activity 类的子类中没有添加任何构造方法，则子类会自动创建一个构造方法，并调用 Activity 类的无参数构造方法。

#### 4.1.2 配置 Activity

创建完 Activity 并不能马上使用，还需要在 AndroidManifest.xml 文件中配置 Activity。下面是配置 Activity 的代码。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobile.android.first"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Main"
            android:label="@string/app_name"> <!-- app_name 在 res/values/strings.
            xml 文件中 --&gt;
            &lt;intent-filter&gt;
                &lt;action android:name="android.intent.action.MAIN" /&gt;
                &lt;category android:name="android.intent.category.LAUNCHER" /&gt;
            &lt;/intent-filter&gt;
        &lt;/activity&gt;
    &lt;/application&gt;
    &lt;uses-sdk android:minSdkVersion="7" /&gt;
&lt;/manifest&gt;</pre>

```

上面代码中的黑体字部分用于配置 Activity。每一个 Activity 都会对应 AndroidManifest.xml 文件中的一个<activity>标签。在<activity>标签中有一个必选的属性： android:name，该属性需要指定一个 Activity 的类名，例如，在第 2 章自动生成的 mobile.android.ch02.first.Main 类。指定 android:name 属性值有如下 3 种方式。

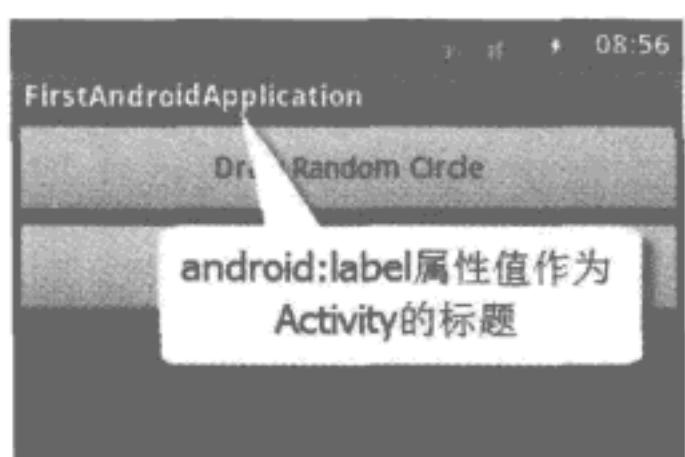
- 指定完整的类名（packagename+classname），例如，mobile.android.ch02.first.Main。
- 只指定类名，例如，.Main，其中 Main 前面的“.”是可选的。该类所在的包名需要在<manifest>标签的 package 属性中指定。本书的所有例子都使用这种方式来指定 Activity 的类名。
- 指定相对类名，这种方式类似于第 2 种方式，只是在<activity>标签的 android:name 属性中不仅指定类名，还有部分包名。例如，如果 Main 类在 mobile.android.ch02.first.abcd 包中，就可以

在<manifest>标签的 package 属性中指定 mobile.android.ch02.first，然后在<activity>标签的 android:name 属性中指定 .abcd.Main。

<activity>标签除了有 android:name 属性外，还有很多可选的属性，比较常用的有 android:label 和 android:icon。 android:label 属性可以指定一个字符串或资源 ID，应用程序中有很多地方都会使用 android:label 属性值，例如，在 Android 手机的应用程序列表中程序图标下方的文字；如果未使用 Activity.setTitle 方法设置 Activity 的标题，系统会将 android:label 属性值作为 Activity 的默认标题，如图 4.1 和图 4.2 所示。



▲ 图 4.1 图标下方的文字 ( android:label 属性值 )



▲ 图 4.2 Activity 的标题 ( android:label 属性值 )

如果<activity>标签未指定 android:label 属性，系统会使用<application>标签的 android:label 属性值，也就是说，<application>标签的 android:label 属性值是<activity>标签的 android:label 属性的默认值。

<activity>标签的 android:icon 属性必须指定一个图像资源 ID，这个资源 ID 所指定的图像将作为应用程序列表（如图 4.1 所示）中的程序图标。如果未指定<activity>标签的 android:icon 属性，系统会使用<application>标签的 android:icon 属性值来代替。

如果<application>和<activity>标签都未指定 android:label 属性，系统会使用 Main Activity 类的全名来代替 android:label 属性的值，例如，第 2 章例子中的 mobile.android.ch02.first.Main。 android:icon 属性与 android:label 属性类似，如果<activity>标签未指定 android:icon 属性，则使用<application>标签的属性值来代替，如果<application>标签也未指定 android:icon 属性，系统会使用默认的图像。

<activity>标签并不需要包含子标签就可以完成对 Activity 的定义，但在上面的代码中<activity>标签中包含了一个<intent-filter>标签。这个标签的作用是对 Activity 进行分类。在本例的<intent-filter>标签中使用了<action>和<category>标签。其中<action>标签表示 Activity 可接收的动作，<category>标签表示 Activity 所属的种类。实际上，<action>和<category>标签中的 android:name 属性值只是一个普通的字符串。在 Android 系统中预定义了一些标准的动作和种类，例如：

android.intent.action.MAIN 和 android.intent.category.LAUNCHER。其中 android.intent.action.MAIN 需要定义在 Main Activity 类的<activity>标签中。当 Android 系统运行程序时，会首先启动包含 android.intent.action.MAIN 动作的 Activity。作为 Main Activity，必须要使用 android.intent.category.LAUNCHER 作为其类别，表示该 Activity 可以显示在最顶层。

### 4.1.3 显示其他的 Activity ( Intent 与 Activity )

**工程目录: src\ch04\ch04\_show\_activity**

任何一个稍微复杂的 Android 应用程序都不太可能只有一个 Activity。如果在程序中包含了多个 Activity，就涉及在 Main Activity 中调用其他 Activity 的问题。然而，Android 的机制不允许像创建普通的 Java 类一样来创建 Activity 的对象实例，并显示 Activity。要想创建和显示 Activity，必须使用 android.content.Intent 作为中间的代理，并使用 startActivity 或 startActivityForResult 方法（将在后面详细介绍）创建并显示 Activity。例如，在 Android 工程中有一个 MyActivity，它的定义如下：

```
<activity android:name=".MyActivity">
</activity>
```

在 Main Activity 中显示 MyActivity 的代码如下：

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

在上面的代码中通过 Intent 类的构造方法指定了两个参数：this 和 MyActivity.class。让我们先看一下 Intent 构造方法的定义，再看一下为什么要传递这两个参数。

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

第一个参数需要指定一个 Context 类型的对象。由于本例是在 Main Activity 中调用 MyActivity，因此，指定 this 即可（Activity 是 Context 的子类）。第二个参数需要指定要显示的 Activity 类的 Class 对象，在本例中是 MyActivity.class。

如果不直接在 Intent 类的构造方法中指定这两个参数，也可以使用 Intent 类的 setClass 方法来指定，代码如下：

```
Intent intent = new Intent();
intent.setClass(this, MyActivity.class);
startActivity(intent);
```

上面调用 Activity 的方法可以称为显式调用，因为这种方法直接指定了要调用的 Activity。除了这种方法，还可以通过隐式调用来显示 Activity。

隐式调用仍然需要使用 Intent，但并不需要指定要调用的 Activity，而只要指定一个 Action 和相应的 Category 即可。在 4.1.2 节介绍了如何定义 Activity，其中涉及了<action>和<category>标签。实际上，这两个标签不光是供 Android 系统使用的。我们也可以将它们应用在自定义的 Activity 中。

<action>标签的 android:name 属性值虽然可以是任意的字符串，但笔者建议使用有意义的字符串，并要在程序中通过常量来引用。如果是自定义的种类，<category>标签的属性值至少要有一个 android.intent.category.DEFAULT。当然，一个<intent-filter>标签可以包含多个<action>和<category>标签，一个<activity>标签也可以包含多个<intent-filter>标签。当<intent-filter>标签中只有一个值为 android.intent.category.DEFAULT 的 category 时，并不需要在 Intent 对象中指定这个 category，如果包含了其他的 category，必须要使用 intent.addCategory 方法添加相应的 category。也许很多读者看到这对<action>和<category>标签的使用还是有些不理解，不过不要紧，下面让我们通过下面的例子来更好地理解<action>和<category>标签的使用方法。

下面使用一个例子来演示如何显式和隐式调用 Activity。通过隐式方式显示 Activity 可能会有两个或两个以上的 Activity 满足条件，这时会在屏幕上显示一个列表，允许让用户选择显示哪个 Activity。下面我们就来编写这个程序。

首先在工程中添加 3 个 Activity（实际上是 4 个 Activity，其中 Main 已经由 ADT 自动生成了）：MyActivity1、MyActivity2 和 MyActivity3。这 3 个 Activity 的配置代码如下：

```

<!-- Main Activity -->
<activity android:name=".Main" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<!-- 该 Activity 未设置任何的 intent-filter，用显式的方式来显示这个 Activity -->
<activity android:name=".MyActivity1" android:label="MyActivity1">
</activity>
<!-- 该 Activity 中定义了两个 intent-filter，其中在第二个 intent-filter 中定义了两个 category -->
<activity android:name=".MyActivity2" android:label="MyActivity2"
          android:icon="@drawable/icon1">
    <intent-filter>
        <action android:name="myaction1" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <!-- 这个 intent-filter 与 MyActivity3 中的 intent-filter 相同 -->
    <intent-filter>
        <action android:name="myaction2" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="mycategory" />
    </intent-filter>
</activity>
<activity android:name=".MyActivity3" android:label="MyActivity3"
          android:icon="@drawable/icon2">
    <!-- 这个 intent-filter 与 MyActivity2 的第二个 intent-filter 相同，使用这个 intent-filter
        隐式显示 Activity 时在屏幕上会弹出一个列表，允许用户选择显示哪个 Activity -->
    <intent-filter>
        <action android:name="myaction2" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="mycategory" />
    </intent-filter>
</activity>

```

## Android 开发权威指南

上面的配置代码中使用了两个图像文件：icon1.png 和 icon2.png。为了方便，将这两个图像文件都放在了 res/drawable 目录中。这样无论什么分辨率的屏幕都可以找到这两个图像文件。如果读者要将这两个图像放在默认生成的图像资源目录中，那么 res/drawable-hdpi、res/drawable-mdpi 和 res/drawable-ldpi 这 3 个目录都要放置同名，不同分辨率的图像文件。下面看看如何使用不同的方式来显示 Activity，代码如下：

```
package mobile.android.ch04.show.activity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Main extends Activity implements OnClickListener
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 从 XML 布局文件中创建 3 个按钮对象
        Button button1 = (Button) findViewById(R.id.button1);
        Button button2 = (Button) findViewById(R.id.button2);
        Button button3 = (Button) findViewById(R.id.button3);

        // 设置 3 个按钮的单击事件
        button1.setOnClickListener(this);
        button2.setOnClickListener(this);
        button3.setOnClickListener(this);
    }
    @Override
    public void onClick(View view)
    {
        Intent intent = null;
        switch (view.getId())
        {
            case R.id.button1:          // 显式调用 MyActivity1
                intent = new Intent(this, MyActivity1.class);
                startActivity(intent);
                break;
            case R.id.button2:          // 隐式调用 MyActivity2
                intent = new Intent("myaction1");
                startActivity(intent);
                break;
            case R.id.button3:          // 隐式调用 MyActivity2 和 MyActivity3
                intent = new Intent("myaction2");
                // 并不需要显式添加 android.intent.category.DEFAULT
                intent.addCategory("mycategory");
        }
    }
}
```

```

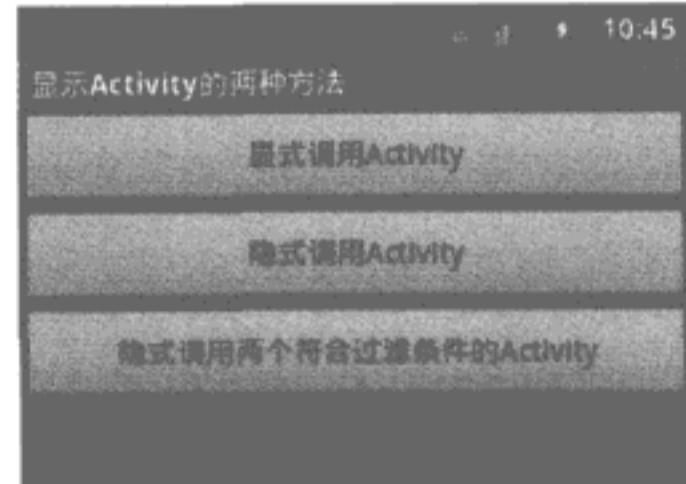
        startActivity(intent);
        break;
    }
}
)

```

如果<intent-filter>标签中使用了默认 category (android.intent.category.DEFAULT)，在隐式调用 Activity 时并不需要在 Intent 对象中使用 addCategory 方法指定。如果非要指定，在 Intent 类中定义了相应的常量：Intent.CATEGORY\_DEFAULT。使用下面的代码可添加 Intent.CATEGORY\_DEFAULT。

```
intent.addCategory(Intent.CATEGORY_DEFAULT);
```

运行程序，会显示 3 个按钮，如图 4.3 所示。当按第一个按钮时，会根据指定的 MyActivity1.class 来动态创建 MyActivity1 的对象实例，并显示 MyActivity1。按第二个按钮，系统就会查找包含 myaction1 的 Activity。如果找到，就会显示这个 Activity。当按第三个按钮时，由于 MyActivity2 和 MyActivity3 都包含了名为 myaction2 的动作，并都属于名为 mycategory 的种类，所以首先会弹出如图 4.4 所示的选择界面。用户可以选择其中一个 Activity 来运行。如果选择下面的“Use by default for this action”复选框，下次再按第三个按钮时，会直接运行上次选择的 Activity。



▲ 图 4.3 以不同方法显示 Activity 的按钮



▲ 图 4.4 选择显示哪个 Activity

## 4.2 Activity 的生命周期

工程目录：src\ch04\ch04\_activitycycle

在 Activity 从建立到销毁的过程中需要在不同的阶段调用 7 个生命周期方法。这 7 个生命周期方法的定义如下：

## Android 开发权威指南

```

protected void onCreate(Bundle savedInstanceState)
protected void onStart()
protected void onResume()
protected void onPause()
protected void onStop()
protected void onRestart()
protected void onDestroy()

```

上面 7 个生命周期方法分别在 4 个阶段按一定的顺序进行调用，这 4 个阶段如下。

- **开始 Activity：**在这个阶段依次执行 3 个生命周期方法，分别是 `onCreate`、`onStart` 和 `onResume`。

- **Activity 失去焦点：**如果在 Activity 获得焦点的情况下进入其他的 Activity 或应用程序，当前的 Activity 会失去焦点。在这一阶段会依次执行 `onPause` 和 `onStop` 方法。

- **Activity 重新获得焦点：**如果 Activity 重新获得焦点，会依次执行 3 个生命周期方法，分别是 `onRestart`、`onStart` 和 `onResume`。

- **关闭 Activity：**当 Activity 被关闭时系统会依次执行 3 个生命周期方法，分别是 `onPause`、`onStop` 和 `onDestroy`。

如果在这 4 个阶段执行生命周期方法的过程中不发生状态的改变，系统会按上面的描述依次执行这 4 个阶段中的生命周期方法，但如果在执行过程中改变了状态，系统会按更复杂的方式调用生命周期方法。

在执行的过程中可以改变系统的执行轨迹的生命周期方法是 `onPause` 和 `onStop`。如果在执行 `onPause` 方法的过程中 Activity 重新获得了焦点，然后又失去了焦点，系统将不会再执行 `onStop` 方法，而是按如下顺序执行相应的生命周期方法：

`onPause -> onResume-> onPause`

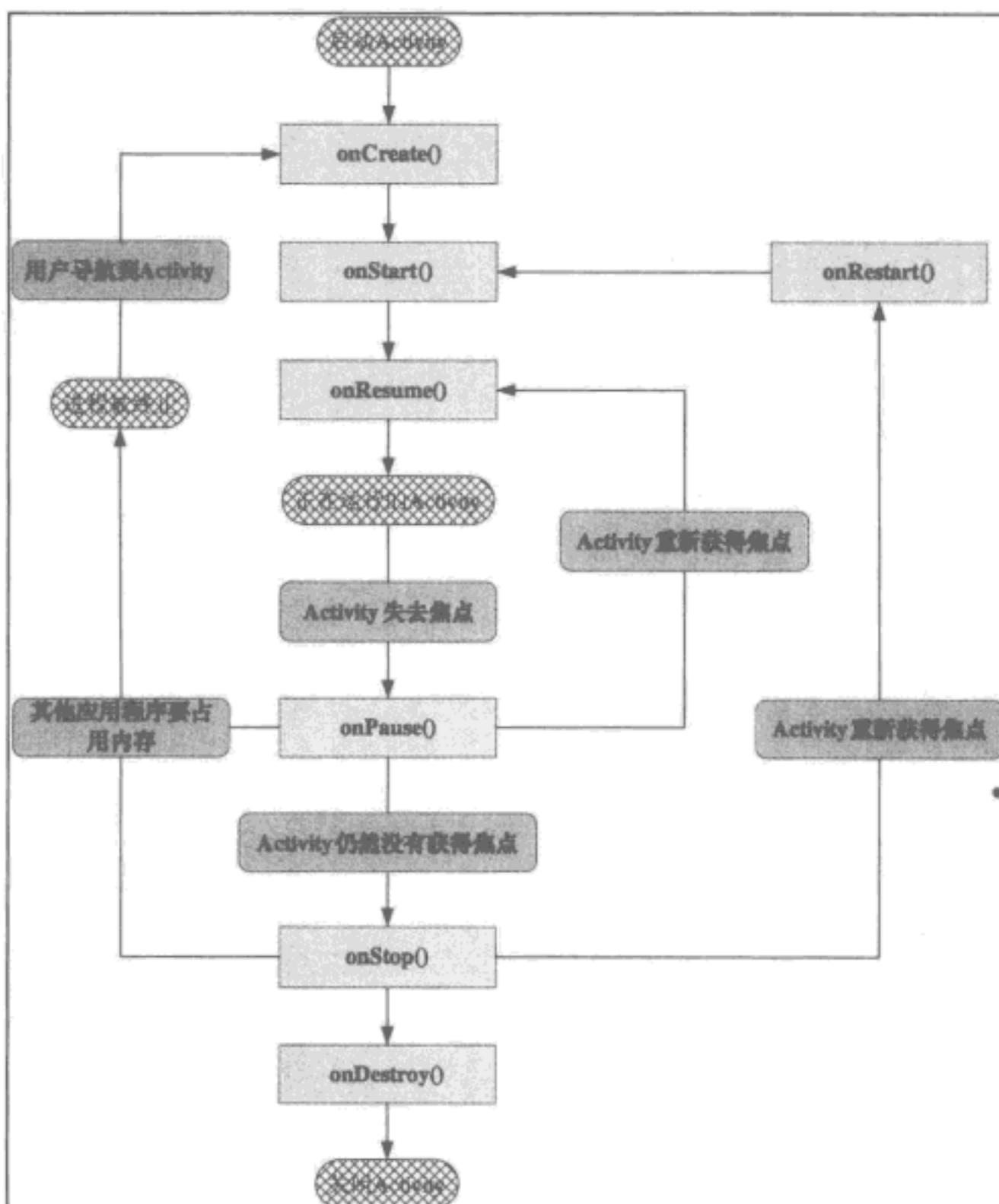
如果在执行 `onStop` 方法的过程中 Activity 重新获得了焦点，然后又失去了焦点，系统将不会执行 `onDestroy` 方法，而是按如下顺序执行相应的生命周期方法：

`onStop->onRestart->onStart->onResume->onPause->onStop`

图 4.5 详细描述了这一过程。

从如图 4.5 所示的 Activity 生命周期不难看出，在这个图中包含两层循环，第一层循环是 `onPause -> onResume-> onPause`，第二层循环是 `onStop->onRestart->onStart->onResume->onPause->onStop`。我们可以将这两层循环看成是整个 Activity 生命周期中的子生命周期。第一层循环称为焦点生命周期，第二层循环称为可视生命周期。也就是说，第一层循环在 Activity 焦点的获得与失去的过程中循环，在这一过程中，Activity 始终是可见的。第二层循环是在 Activity 可见与不可见的过程中循环，在这个过程中伴随着 Activity 焦点的获得与失去。也就是说，Activity 首先会被显示，然后会获得焦点，接着失去焦点，最后由于弹出其他的 Activity，使当前的 Activity 变成不可见。因此，Activity 有如下 3 种生命周期。

- 整体生命周期：`onCreate->... ... ->onDestroy`。
- 可视生命周期：`onStart->... ... ->onStop`。
- 焦点生命周期：`onResume->onPause`。



▲ 图 4.5 Activity 的生命周期

下面的代码覆盖了 Activity 类中的 7 个生命周期方法，并在每一个方法中向 LogCat 视图输出了相应的信息。

```

package mobile.android.ch04.activity.cycle;

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

public class Main extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        Log.d("onCreate", "onCreate Method is executed.");
    }
    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        Log.d("onDestroy", "onDestroy Method is executed.");
    }
}
  
```

```

}
@Override
protected void onPause()
{
    super.onPause();
    Log.d("onPause", "onPause Method is executed.");
}
@Override
protected void onRestart()
{
    super.onRestart();
    Log.d("onRestart", "onRestart Method is executed.");
}
@Override
protected void onResume()
{
    super.onResume();
    Log.d("onResume", "onResume Method is executed.");
}
@Override
protected void onStart()
{
    super.onStart();
    Log.d("onStart", "onStart Method is executed.");
}
@Override
protected void onStop()
{
    super.onStop();
    Log.d("onStop", "onStop Method is executed.");
}
}

```

在编写上面代码时应注意如下几点。

- 在 Android 应用程序中不能使用 System.out.println(...) 来输出信息，而要使用 Log 类中的静态方法输出调试信息。在本例中使用了 Log.d 方法输出调试信息，在 DDMS 透视图的 LogCat 视图中可以查看 Log.d 方法输出的信息。
- 在 Activity 的子类中覆盖这 7 个生命周期方法时应该在这些方法的一开始调用 Activity 类中的生命周期方法，否则系统会抛出异常。

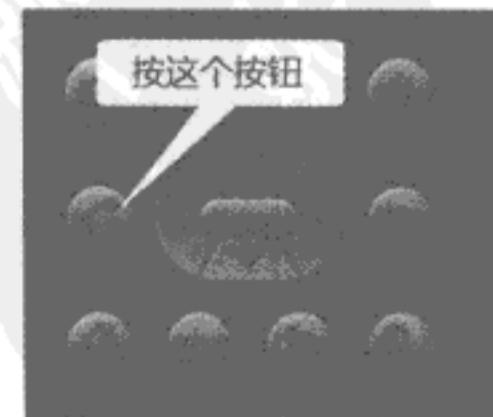
读者可按如下步骤来操作应用程序。

- (1) 运行应用程序。
- (2) 按模拟器上的接听按钮（如图 4.6 所示）进入“拨号”界面，然后退出这个界面。

- (3) 关闭应用程序。

完成上面 3 个步骤后，在 LogCat 视图中可以看到如图 4.7 所示的输出信息。

如图 4.7 所示的输出信息是在 Activity 生命周期的 4 个阶段输出的调试信息，除此之外，还有一些系统输出的信息。读者可以在【Filter】文本框中输入 executed 过滤掉其他信息，过滤效果如图 4.8 所示。



▲ 图 4.6 接听按钮位置

The screenshot shows the LogCat window with several log entries. Annotations with arrows point to specific lines:

- An arrow points to the first three lines (onCreate, onStart, onResume) with the text "启动应用程序".
- An arrow points to the lines (onPause, onStop) with the text "失去焦点".
- An arrow points to the lines (onRestart, onStart, onResume) with the text "重新获得焦点".
- An arrow points to the last three lines (onPause, onStop, onDestroy) with the text "退出应用程序".

```

Log
Time pid tag
09-15 21:06... D 626 dalvikvm
09-15 21:07... I 582 ActivityManager
09-15 21:07... D 839 onCreate
09-15 21:07... D 839 onStart
09-15 21:07... D 839 onResume
09-15 21:07... I 582 ActivityManager
09-15 21:07... D 626 InputConnectionWrapper
09-15 21:07... D 626 dalvikvm
09-15 21:07... V 839 KeyCharacterMap
09-15 21:07... V 839 KeyCharacterMap
09-15 21:07... I 582 ActivityManager
09-15 21:07... D 839 onPause
09-15 21:07... D 839 onStop
09-15 21:07... I 582 ActivityManager
09-15 21:07... D 839 dalvikvm
09-15 21:07... V 839 KeyCharacterMap
09-15 21:07... V 839 KeyCharacterMap
09-15 21:07... D 839 onRestart
09-15 21:07... D 839 onStart
09-15 21:07... D 839 onResume
09-15 21:07... D 626 dalvikvm
09-15 21:07... V 839 KeyCharacterMap
09-15 21:07... V 839 KeyCharacterMap
09-15 21:08... D 839 onPause
09-15 21:08... D 839 onStop
09-15 21:08... D 839 onDestroy
09-15 21:08... D 626 dalvikvm

Message
GC freed 973 objects / 52320 bytes ...
Starting activity: Intent { action=... }
onCreate Method is executed.
onStart Method is executed.
onResume Method is executed.
Displayed activity net.blogjava.mob...
showStatusIcon on inactive InputCon...
GC freed 274 objects / 13232 bytes ...
No keyboard for id 0
Using default keymap: /system/usr/k...
Starting activity: Intent { action=... }
onPause Method is executed.
onStop Method is executed.
Displayed activity com.android.cont...
GC freed 183 objects / 25832 bytes ...
No keyboard for id 0
Using default keymap: /system/usr/k...
onRestart Method is executed.
onStart Method is executed.
onResume Method is executed.
GC freed 1486 objects / 88664 bytes ...
No keyboard for id 0
Using default keymap: /system/usr/k...
onPause Method is executed.
onStop Method is executed.
onDestroy Method is executed.
GC freed 154 objects / 9152 bytes ...

```

▲图4.7 Activity生命周期的4个阶段输出的信息

The screenshot shows the LogCat window with a filter set to "executed". It displays the following log entries:

```

Log
Time pid tag
09-15 21:07... D 839 onCreate
09-15 21:07... D 839 onStart
09-15 21:07... D 839 onResume
09-15 21:07... D 839 onPause
09-15 21:07... D 839 onStop
09-15 21:07... D 839 onRestart
09-15 21:07... D 839 onStart
09-15 21:07... D 839 onResume
09-15 21:08... D 839 onPause
09-15 21:08... D 839 onStop
09-15 21:08... D 839 onDestroy

```

▲图4.8 只显示在生命周期方法中输出的调试信息

从如图4.7所示的4组输出信息中也可以看出Activity的3个生命周期，为了看起来更方便，使用黑框将这3个生命周期要调用的方法括起来，如图4.9所示。

The screenshot shows the LogCat window with a black rectangular box highlighting the lines for onStart, onRestart, and onResume. The log entries are as follows:

```

Log
Time pid tag
09-15 23:06... D 626 dalvikvm
09-15 23:06... I 582 ActivityManager
09-15 23:06... D 839 onCreate
09-15 23:06... D 839 onStart
09-15 23:06... D 839 onResume
09-15 23:06... I 582 ActivityManager
09-15 23:06... D 626 InputConnectionWrapper
09-15 23:06... V 839 KeyCharacterMap
09-15 23:06... V 839 KeyCharacterMap
09-15 23:06... I 582 ActivityManager
09-15 23:06... D 839 onPause
09-15 23:06... I 582 ActivityManager
09-15 23:06... D 839 onStop
09-15 23:06... I 582 ActivityManager
09-15 23:06... D 839 onRestart
09-15 23:06... D 839 onStart
09-15 23:06... D 839 onResume
09-15 23:06... D 839 onPause
09-15 23:06... D 839 onStop
09-15 23:06... D 839 onDestroy
09-15 23:06... D 623 dalvikvm

Message
GC freed 1059 objects / 47256 bytes ...
Starting activity: Intent { action=... }
onCreate Method is executed.
onStart Method is executed.
onResume Method is executed.
Displayed activity net.blogjava.mob...
showStatusIcon on inactive InputCon...
No keyboard for id 0
Using default keymap: /system/usr/k...
Starting activity: Intent { action=... }
onPause Method is executed.
onStop Method is executed.
Displayed activity com.android.cont...
No keyboard for id 0
Using default keymap: /system/usr/k...
onRestart Method is executed.
onStart Method is executed.
onResume Method is executed.
onPause Method is executed.
onStop Method is executed.
onDestroy Method is executed.
GC freed 102 objects / 5360 bytes i...
GC freed 9293 objects / 524240 byte...

```

▲图4.9 Activity的3个生命周期

## 4.3 在不同 Activity 之间传递数据

工程目录: src\ch04\ch04\_transmit\_data

如果读者以前使用其他语言或技术做过窗体切换的程序，会意识到在窗体切换的过程中可能伴随着数据的传递。那么在 Activity 之间切换时也不可避免要进行数据传递。例如，在单击列表中的某个列表项时，需要编辑与这个列表项相关的数据。这时就需要再显示一个 Activity，并将原始数据传递给这个 Activity，这就是一个典型的数据传递的过程。

在 Android 中传递数据的方法非常多。本节将介绍 4 种比较常用的数据传递方法。这 4 种数据传递方法如下：

- 通过 Intent 传递数据。
- 通过静态 (static) 变量传递数据。
- 通过剪切板 (Clipboard) 传递数据。
- 通过全局变量传递数据。

下面来分别看一下这 4 种数据传递的方法。

### 4.3.1 使用 Intent 传递数据

这是最常用的一种数据传递的方法。通过 Intent 类的.putExtra 方法可以将简单类型的数据或可序列化的对象保存在 Intent 对象中，然后在目标 Activity 中使用 getXxx (getInt、getString 等) 方法获得这些数据。将数据保存到 Intent 对象的代码如下：

```
Intent intent = new Intent(this, MyActivity1.class);
// 保存 String 类型的值
intent.putExtra("intent_string", "通过 Intent 传递的字符串");
// 保存 Integer 类型的值
intent.putExtra("intent_integer", 300);
Data data = new Data();
data.id = 1000;
data.name = "Android";
// 保存可序列化的对象
intent.putExtra("intent_object", data);
// 显示 MyActivity1
startActivity(intent);
```

在上面的代码中涉及到一个 Data 类，这个类是可序列化的，也就是实现了 java.io.Serializable 接口。Data 类的代码如下：

```
package mobile.android.ch04.transmit.data;
import java.io.Serializable;
public class Data implements Serializable
{
    public int id;
    public String name;
}
```

在 MyActivity1 类中获得上面保存的 3 个值 (String、Integer 和 Data 类型的值) 的代码如下：

```

package mobile.android.ch04.transmit.data;

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

public class MyActivity1 extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.myactivity);
        TextView textView = (TextView) findViewById(R.id.textview);
        // 获得 String 类型的值
        String intentString = getIntent().getStringExtra("intent_string");
        // 获得 Integer 类型的值
        int intentInteger = getIntent().getExtras().getInt("intent_integer");
        // 获得 Data 类型的值
        Data data = (Data) getIntent().getExtras().get("intent_object");

        StringBuffer sb = new StringBuffer();
        sb.append("intent_string: ");
        sb.append(intentString);
        sb.append("\n");
        sb.append("intent_integer: ");
        sb.append(intentInteger);
        sb.append("\n");
        sb.append("data.id: ");
        sb.append(data.id);
        sb.append("\n");
        sb.append("data.name: ");
        sb.append(data.name);
        sb.append("\n");
        // 在屏幕上输出传递过来的值
        textView.setText(sb.toString());
    }
}

```

运行程序后，单击如图 4.10 所示的界面的第一个按钮，会显示如图 4.11 所示的输出信息。



▲ 图 4.10 使用不同的方式在 Activity 之间传递数据



▲ 图 4.11 显示从 Intent 对象中获得的值

### 4.3.2 使用静态变量传递数据

虽然使用 Intent 对象可以很方便地在 Activity 之间传递数据，这也是官方推荐的数据传递方式。但 Intent 也有其局限性，就是 Intent 无法传递不能序列化的对象，也就是没有实现 java.io.Serializable 接口的类的对象。例如，在第 2 章接触到的 Canvas 对象就无法通过 Intent 对象进行传递。如果传递自定义类的对象，也必须实现 java.io.Serializable 接口才可以。如果这些类没有源代码，而且还没有实现 java.io.Serializable 接口，使用 Intent 对象就没有任何办法传递这些类的对象了。

不过天无绝人之路，Intent 对象不行，还是有其他方法的。静态变量就是一种非常方便、易用的传递数据的方法。例如，要向 MyActivity2 传递三个值：id、name 和 data，就需要在 MyActivity2 类中定义 3 个 public 的静态变量（在其他类中定义这些变量也可以），代码如下：

```
package mobile.android.ch04.transmit.data;

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

public class MyActivity2 extends Activity
{
    // 下面定义了 3 个静态变量
    public static String name;
    public static int id;
    public static Data data;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.myactivity);
        TextView textView = (TextView) findViewById(R.id.textview);

        StringBuffer sb = new StringBuffer();
        sb.append("name: ");
        sb.append(name);
        sb.append("\n");
        sb.append("id: ");
        sb.append(id);
        sb.append("\n");
        sb.append("data.id: ");
        sb.append(data.id);
        sb.append("\n");
        sb.append("data.name: ");
        sb.append(data.name);
        sb.append("\n");
        // 输出静态变量的值
        textView.setText(sb.toString());
    }
}
```

要想在 MyActivity2 中获得这些静态变量的值，就需要在显示 MyActivity2 之前（调用

startActivity 方法之前) 为这些静态变量赋值, 代码如下:

```
Intent intent = new Intent(this, MyActivity2.class);
// 下面的代码为 MyActivity2 中的 3 个静态变量赋值
MyActivity2.id = 3000;
MyActivity2.name = "保时捷";
MyActivity2.data = new Data();
MyActivity2.data.id = 1000;
MyActivity2.data.name = "Android";
startActivity(intent);
```

单击如图 4.10 所示界面中的第 2 个按钮后, 会显示如图 4.12 所示的输出信息。



▲ 图 4.12 显示通过静态变量传递过来的值

### 4.3.3 使用剪切板传递数据

在 Activity 之间传递还可以利用一些技巧。例如, 不管是 Windows, 还是 Linux (Android、Meego 或其他基于 Linux 内核的操作系统), 都会支持一种叫剪切板的技术。也就是某一个程序将一些数据复制到剪切板上, 然后其他的任何程序都可以从剪切板中获得这些数据。那么在 Activity 之间传递数据也可以利用这种技术。将字符串保存到剪切板的代码如下:

```
Intent intent = new Intent(this, MyActivity3.class);
// 获得管理剪切板的对象 (ClipboardManager)
ClipboardManager clipboard =
    (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
// 向剪切板保存字符串
clipboard.setText("通过 Clipboard 传递的数据");
startActivity(intent);
```

在上面代码使用了 getSystemService 方法获得了一个系统服务对象, 也就是 ClipboardManager 对象, 该对象用于管理系统剪切板, 并使用 ClipboardManager.setText 方法向剪切板保存了一个字符串。下面的代码从剪切板中获得这个字符串。

```
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
// 从剪切板获得字符串
String text = clipboard.getText().toString();
```

#### 扩展学习：通过剪切板保存复杂的数据

上面的读写剪切板的代码非常简单, 也容易使用。可惜遗憾的是, ClipboardManager 对象只支持向剪切板读写字符串, 并不支持其他类型的数据, 更别提复杂的对象了。当然, 如果是其他简单

类型的数据，如 int、bool 等，可以将其转换成字符串进行传递。那么如果是对象类型呢，比如前面涉及的 Data 对象能否通过剪切板进行传递呢？答案是肯定的。

由于 Data 是可序列化的对象，因此，完全可以将 Data 转换成 byte[] 类型的数据，然后将 byte[] 类型的数据再进行 Base64 编码（通过 E-mail 发送附件就是将附件转换成 Base64 格式的字符串发送的）转换成字符串，这样就可以使用剪切板进行数据传递了（通过 Intent 对象也可以利用这种方法来传递可序列化的对象）。下面的代码利用剪切板传递了一个 Data 对象。为了方便，也可以根据下面的代码编写一个通用的方法将可序列化的对象转换成字符串，这就留给感兴趣的读者来实现吧！

```

Intent intent = new Intent(this, MyActivity3.class);
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
Data clipboardData = new Data();
clipboardData.id = 6666;
clipboardData.name = "通过 Clipboard 传递的数据";
// 下面的代码开始将 clipboardData 对象转换成 Base64 格式的字符串
ByteArrayOutputStream baos = new ByteArrayOutputStream();
String base64Str = "";
try
{
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(clipboardData);
    // 使用 Base64.encodeToString 方法将 byte[] 数据转换成 Base64 字符串
    base64Str = Base64.encodeToString(baos.toByteArray(), Base64.DEFAULT);
    oos.close();
}
catch (Exception e)
{
}
// 向剪切板写入 Base64 格式的字符串
clipboard.setText(base64Str);
startActivity(intent);

```

下面的代码在 MyActivity3 中将 Base64 编码的字符串进行解码，并还原成 Data 对象。

```

ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);
// 从剪切板中获得 Base64 编码格式的字符串
String base64Str = clipboard.getText().toString();
// 将 Base64 格式的字符串还原成 byte[] 格式的数据
byte[] buffer = Base64.decode(base64Str, Base64.DEFAULT);
ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
try
{
    ObjectInputStream ois = new ObjectInputStream(bais);
    // 将 byte[] 数据还原成 Data 对象
    Data data = (Data) ois.readObject();
    // 输出原始的 Base64 格式的字符串和 Data 对象中的字段值
    textView.setText(base64Str + "\n\n" + data.id + "\n" + data.name);
}
catch (Exception e)
{
}

```

单击如图 4.10 所示界面的第 3 个按钮，会显示如图 4.13 所示的输出信息。



▲ 图 4.13 通过剪切板传递可序列化的对象

**注意** 由于 Base64 类是从 Android 2.2 开始支持的，因此，Android 2.1 及更低版本的 Android 无法通过 Android SDK API 进行 Base64 编码和解码。因此，需要使用第三方的类库（例如，common httpclient）才可以进行 Base64 编码。

#### 4.3.4 使用全局对象传递数据

虽然使用静态变量可以传递任何类型的数据，但官方并不建议这样做。如果在类中大量使用静态变量（尤其是使用很占资源的变量，例如，Bitmap 对象）可能会造成内存溢出异常，而且还可能因为静态变量在很多类中出现而造成代码难以维护和混乱，因此，在本节介绍一种更优雅的数据传递方式，这就是全局对象。这种方式可完全取代静态变量。

使用过 Java EE 的读者对 Java Web 的 4 个作用域一定非常清楚，这 4 个作用域从小到大分别是 Page、Request、Session 和 Application，其中 Application 域在应用程序的任何地方都可以访问，除非将 Web 服务器停止，否则这个 Application 域将一直存在。Android 中的全局对象非常类似于 Java Web 中的这个 Application 域，除非将 Android 应用程序彻底清除出内存，否则全局对象将一直可以访问。

接下来看一下如何使用全局对象。全局对象所对应的类必须是 android.app.Application 的子类。下面就是一个典型的全局类。

```
package mobile.android.ch04.transmit.data;
import android.app.Application;
public class MyApp extends Application
{
    public String country;
    public Data data = new Data();
}
```

**注意** 全局类中不需要定义静态变量，只需要定义成员变量即可，而且全局类中必须要有一个无参数的构造方法，或不编写任何代码的构造方法（系统会自动建立一个无参数的构造方法）。这个类和 Activity 类是一样的，由系统自动创建 MyApp 对象，因此，必须要有一个无参数的构造方法。

只编写一个全局类是不会自动创建全局类对象的，因为 Android 系统并不知道哪个类是全局类，因此，需要在 `AndroidManifest.xml` 文件中通过`<application>`标签的 `android:name` 属性指定这个类，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobile.android.ch04.transmit.data"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:name=".MyApp" android:icon="@drawable/icon" android:label="@string/app_name">
        ...
        <uses-sdk android:minSdkVersion="9" />
    </manifest>
```

指定全局类后，在程序运行后，`MyApp` 对象就会被自动创建，而且会一直在内存中驻留，直到应用程序彻底退出内存。

下面的代码首先获得了 `MyApp` 对象，然后为 `MyApp` 对象中的字段赋值。

```
// 获得 MyApp 对象
MyApp myApp = (MyApp) getApplicationContext();
myApp.country = "美国";
myApp.data.id = 1234;
myApp.data.name = "飞碟";
intent = new Intent(this, MyActivity4.class);
startActivity(intent);
```

在 `MyActivity4` 中获得了 `MyApp` 对象，并在屏幕上输出了 `MyApp` 对象中字段的值。

```
TextView textView = (TextView) findViewById(R.id.textview);
// 获得 MyApp 对象
MyApp myApp = (MyApp) getApplicationContext();
textView.setText("MyApp.country: " + myApp.country + "\nMyApp.data.id: "
    + myApp.data.id + "\nMyApp.data.name" + myApp.data.name);
```

单击如图 4.10 所示界面的第 4 个按钮，会显示如图 4.14 所示的输出信息。



▲ 图 4.14 使用全局对象传递数据

**归纳总结：**前面介绍了 4 种在不同 `Activity` 之间传递数据的方法。虽然这些方法在某种程度上可以互相取代，但根据具体情况使用不同的数据传递方法会使程序更利于维护，更加健壮。对于向其他 `Activity` 传递简单类型（`int`、`String`、`short`、`bool` 等）或可序列化的对象时，建议使用 `Intent` 对象进行数据传递。如果传递不可序列化的对象，可以采用静态变量或全局对象的方式，不过按照

官方的建议，最好采用全局对象的方式。另外，如果想使某些数据长时间驻留内存，以便程序随时取用，最好采用全局对象的形式，当然，如果数据不复杂，也可以采用静态变量。至于剪切板，如果不是非常特殊的情况，并不建议使用。因为这可能会影响到其他的程序（因为其他的程序可能会使用到剪切板）。

### 4.3.5 返回数据到前一个 Activity

在应用程序中，不仅要向 Activity 传递数据，而且要从 Activity 返回数据。虽然返回数据和传递数据类似，也可以采用前面介绍的 4 种方法，但一般建议采用 Intent 对象的方式来返回数据。使用这种方式返回数据，需要使用 startActivityForResult 方法来显示 Activity，代码如下：

```
Intent intent = new Intent(this, MyActivity5.class);
startActivityForResult(intent, 1);
```

其中 startActivityForResult 方法的第二个参数是一个 int 类型的请求码，可以是任意的整数，只是为了区分请求的来源，以便处理返回结果。现在建立一个 Activity（MyActivity5），在 XML 布局文件（myactivity5.xml）中添加一个文本框（EditText）和一个按钮，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:id="@+id/edittext" android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/button" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:onClick="onClick_Button" android:
        text="确定"/>
</LinearLayout>
```

**注意** 在<Button>标签中有一个 android:onClick 属性，通过该属性可以直接指定按钮单击事件的方法名。这样就可以在 MyActivity5 类中不创建按钮对象而处理按钮单击事件了。如果在程序中只处理单击事件，而不直接引用相应的对象，可以采用这种指定单击事件的方法。

MyActivity5 类中按钮单击事件方法的代码如下：

```
// 该方法就是在 android:onClick 属性中指定的方法名，方法定义与 Activity 中的单击事件方法一样
// 必须是 public 方法
public void onClick_Button(View view)
{
    // metEditText 是创建的 EditText 对象
    String value = metEditText.getText().toString();
    Intent intent = new Intent();
    intent.putExtra("value", value);
    // 通过 Intent 对象返回结果，setResult 方法的第一个参数是一个响应码，与请求码类似
    setResult(2, intent);
    // 关闭当前的 Activity
```

```

        finish();
    }
}

```

在显示 MyActivity5 的 Activity 中需要覆盖 onActivityResult 方法来处理返回的结果，代码如下：

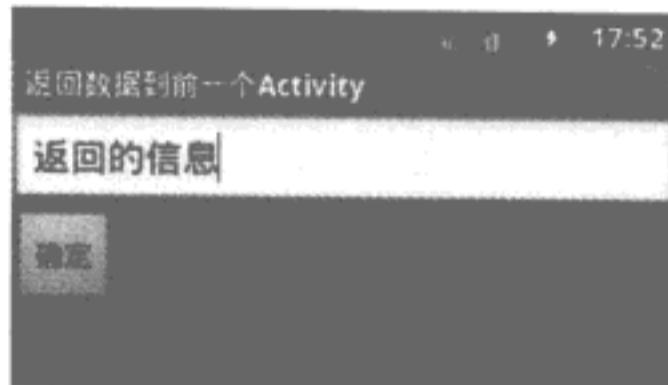
```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode)
    {
        case 1: // 请求码
            switch (resultCode)
            {
                case 2: // 响应码
                    setTitle(data.getStringExtra("value")); // 将返回值作为标题来设置
                    break;
                default:
                    break;
            }
            break;

        default:
            break;
    }
}

```

运行本程序，单击如图 4.10 所示界面的第 5 个按钮，在弹出的界面中输入如图 4.15 所示的内容，并单击“确定”按钮关闭当前界面，会看到主界面的标题已经变成了“返回的信息”。



▲ 图 4.15 录入要返回的信息

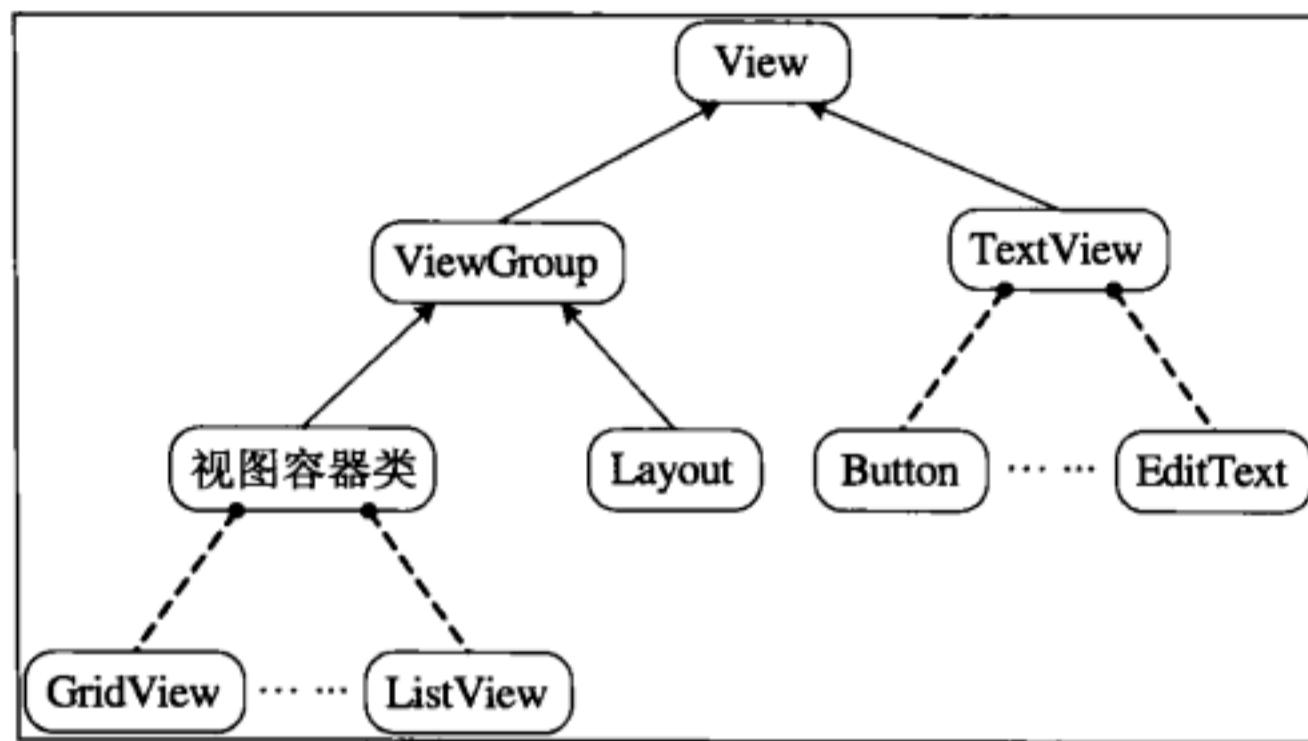
## 4.4 视图 (View)

在 Android 系统中，任何可视化控件都需要从 android.view.View 类继承。开发人员可以使用两种方式创建 View 对象，一种方式是使用 XML 来配置 View 的相关属性，然后再装载这些 View；另外一种方式是完全使用 Java 代码的方式来创建 View。本节将详细介绍如何使用这两种方式来创建 View 对象。

### 4.4.1 视图简介

Android 中的视图类可分为 3 种：布局（Layout）类、视图容器（View Container）类和视图类（例如，TextView 就是一个直接继承于 View 类的视图类）。这 3 种类都是 android.view.View 的子类。

`android.view.ViewGroup` 是一个容器类，该类也是 `View` 的子类，所有的布局类和视图容器类都是 `ViewGroup` 的子类，而视图类直接继承自 `View` 类。图 4.16 描述了 `View`、`ViewGroup` 及视图类的继承关系。



▲ 图 4.16 视图的继承关系

从如图 4.16 所示的继承关系可以看出，`Button`、`TextView`、`EditText` 都是视图类，`TextView` 是 `Button` 和 `EditText` 的父类。在 Android SDK 中还有很多这样的控件类。读者在学习后面的内容时会逐渐接触到这些控件。虽然 `GridView` 和 `ListView` 是 `ViewGroup` 的子类，但并不是直接子类，在 `GridView`、`ListView` 和 `ViewGroup` 之间还有几个视图容器类，从而形成了视图容器类的层次结构。虽然布局视图也属于容器视图，但由于布局视图具有排版功能，所以将这类视图单独作为一类。

#### 4.4.2 使用 XML 布局文件定义视图

XML 布局文件是 Android 系统中定义视图的常用方法。所有的 XML 布局文件必须保存在 `res/layout` 目录中。在第 2 章的例子中已经演示了在程序中装载 XML 布局文件的基本方法。

XML 布局文件的命名及定义需要注意如下几点。

- XML 布局文件的扩展名必须是 `xml`。
- 由于 ADT 会根据每一个 XML 布局文件名在 `R` 类中生成一个变量，这个变量名就是 XML 布局文件名，因此，XML 布局文件名（不包含扩展名）必须符合 Java 变量名的命名规则，例如，XML 布局文件名不能以数字开头。
- 每一个 XML 布局文件的根节点可以是任意的控件标签，如 `<LinearLayout>`、`<TextView>`。
- XML 布局文件的根节点必须包含 `android` 命名空间，而且命名空间的值必须是 `http://schemas.android.com/apk/res/android`。
- 为 XML 布局文件中的标签指定 ID 时需要使用这样的格式：`@+id/somestringvalue`，其中 `@+` 语法表示如果 ID 值在 `R.id` 类中不存在，则新产生一个与 ID 同名的变量，如果在 `R.id` 类中存在该变量，则直接使用这个变量。`somestringvalue` 表示 ID 值，例如，`@+id/textview1`。
- 由于每一个视图 ID 都会在 `R.id` 类中生成与之相对应的变量，因此，视图 ID 的值也要符合 Java 变量的命名规则，这一点与 XML 布局文件名的命名规则相同。

下面是一个标准的 XML 布局文件的内容：

```
<!-- main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView android:id="@+id/textview1" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="textview1" />
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="第一个按钮" />
</LinearLayout>
```

如果要使用上面的 XML 布局文件(main.xml)，通常需要在 onCreate 方法中使用 setContentView 方法指定 XML 布局文件的资源 ID，代码如下：

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

如果想获得在 main.xml 文件中定义的某个 View，可以使用如下代码：

```
TextView textView1 = (TextView) findViewById(R.id.textview1);
Button button1 = (Button) findViewById(R.id.button1);
```

在获得 XML 布局文件中的视图对象时需要注意如下几点。

- 在使用 findViewById 方法之前必须先使用 setContentView 方法装载 XML 布局文件，否则系统会抛出异常。也就是说，findViewById 方法要在 setContentView 方法后面使用。
- 虽然所有的 XML 布局文件中的视图 ID 都在 R.id 类中生成了相应的变量，但使用 findViewById 方法只能获得已经装载的 XML 布局文件中的视图对象。例如，有两个 XML 布局文件 test1.xml 和 test2.xml。在 test1.xml 文件中定义了一个<TextView>标签，android:id 属性值为 @+id/textview1，在 test2.xml 文件中也定义了一个<TextView>标签，android:id 属性值为 @+id/textview2。这时在 R.id 类中会生成两个变量：textview1 和 textview2。但通过 setContentView 方法装载 R.layout.test1 后，只能使用 findViewById 方法获得 test1.xml 中定义的视图对象。findViewById(R.id.textview2)则会返回 null。
- 在不同的 XML 布局文件中可以有相同 ID 值的视图，但在同一个 XML 布局文件中，虽然也可以有相同 ID 值的视图，但通过 ID 值获得视图对象时，只能获得按定义顺序的第一个视图对象，其他相同 ID 值的视图对象将无法获得。因此，在同一个 XML 布局文件中应尽量使视图的 ID 值唯一。

#### 4.4.3 在代码中控制视图

虽然使用 XML 布局文件可以非常方便地对控件进行布局，但若想控制这些控件的行为，仍然需要编写 Java 代码。

在4.4.2节介绍了使用`findViewById`方法获得指定的视图对象，当获得视图对象后，就可以使用Java代码来控制这些视图对象了。例如，下面的代码获得了一个`TextView`对象，并修改了`TextView`的文本。

```
TextView textView = (TextView) findViewById(R.id.textview1);
textView.setText("文本内容");
```

`setText`方法不仅可以直接使用字符串来修改`TextView`的文本，还可以使用字符串资源对`TextView`的文本进行修改，代码如下：

```
textView.setText(R.string.hello);
```

其中`R.string.hello`是字符串资源ID，系统会使用这个ID对应的字符串设置`TextView`的文本。

### ■ 注意

当`setText`方法的参数值是`int`类型时，会被认为这个参数值是一个字符串资源ID，因此，如果要将`TextView`的文本设为一个整数，需要将这个整数转换成`String`类型，例如，可以使用`textView.setText(String.valueOf(200))`将`TextView`的文本设为200。

任何应用程序都离不开事件。在Android应用程序中一般使用以`setOn`开头的方法来设置事件类的对象。例如，下面的代码为一个`Button`对象设置了单击事件。

```
Button button = (Button) findViewById(R.id.button1);
button.setOnClickListener(this);
```

在更高级的Android应用中，往往需要动态添加视图。要实现这个功能，最重要的是获得被添加的视图所在的容器对象，这个容器对象所对应的类需要继承`ViewGroup`类。

将其他的视图添加到当前的容器视图中需要如下几步：

- (1) 获得当前的容器视图对象。
- (2) 获得或创建待添加的视图对象。
- (3) 将相应的视图对象添加到容器视图中。

假设有两个XML布局文件：`test1.xml`和`test2.xml`。这两个XML布局文件的根节点都是`<LinearLayout>`，下面的代码获得了`test2.xml`文件中的`LinearLayout`对象，并将该对象作为`test1.xml`文件中的`<LinearLayout>`标签的子节点添加到`test1.xml`的`LinearLayout`对象中。

```
// 获得 test1.xml 中的 LinearLayout 对象
LinearLayout testLinearLayout1 = (LinearLayout) getLayoutInflater().inflate(R.layout.
test1, null);
// 将 test1.xml 中的 LinearLayout 对象设为当前容器视图
setContentView(testLinearLayout1);
// 获得 test2.xml 中的 LinearLayout 对象，并将该对象添加到 test1.xml 的 LinearLayout 对象中
LinearLayout testLinearLayout2 = (LinearLayout) getLayoutInflater().inflate(R.layout.
test2, testLinearLayout1);
```

其中`inflate`方法的第一个参数表示XML布局资源文件的ID，第二个参数表示获得容器视图对象后，要将该对象添加到哪个容器视图对象中。在这里是`testLinearLayout1`对象。如果不将获得的容器视图对象添加到任何其他的容器中，`inflate`方法的第二个参数需要设为`null`。

除了上面的添加方式外，也可以使用 `addView` 方法向容器视图中添加视图对象，但要将 `inflate` 方法的第 2 个参数值设为 `null`，代码如下：

```
// 获得 test1.xml 中的 LinearLayout 对象
LinearLayout testLinearLayout1 = (LinearLayout) getLayoutInflater().inflate(R.layout.test1, null);
// 将 test1.xml 中的 LinearLayout 对象设为当前容器视图
setContentView(testLinearLayout1);
// 获得 test2.xml 中的 LinearLayout 对象，并将该对象添加到 test1.xml 的 LinearLayout 对象中
LinearLayout testLinearLayout2 = (LinearLayout) getLayoutInflater().inflate(R.layout.test2, null);
testLinearLayout1.addView(testLinearLayout2);
```

除此之外，还可以完全使用 Java 代码创建一个视图对象，并将该对象添加到容器视图中，代码如下：

```
EditText editText = new EditText(this);
testLinearLayout1.addView(editText);
```

向容器视图添加视图对象时需要注意如下几点：

- 如果使用 `setContentView` 方法将容器视图设为当前视图后，还想向容器视图中添加新的视图或进行其他的操作，`setContentView` 方法的参数值应直接使用容器视图对象，因为这样可以向容器视图对象中添加新的视图。
- 一个视图只能有一个父视图，也就是说，一个视图只能被包含在一个容器视图中。因此，在向容器视图添加其他视图时，不能将 XML 布局文件中非根节点的视图对象添加到其他的容器视图中。例如，在前面的例子中不能将使用 `testLinearLayout2.findViewById(R.id.textView2)` 方法获得的 `TextView` 对象添加到 `testLinearLayout1` 对象中，这是因为这个 `TextView` 对象已经属于 `test2.xml` 中的 `<LinearLayout>` 标签了，不能再属于 `test1.xml` 中的 `<LinearLayout>` 标签了。

## 4.5 布局 (Layout)

为了适应各式各样的界面风格，Android 系统提供了 5 种布局。这 5 种布局是 `FrameLayout`（框架布局）、`LinearLayout`（线性布局）、`RelativeLayout`（相对布局）、`TableLayout`（表格布局）和 `AbsoluteLayout`（绝对布局）。利用这 5 种布局，可以将屏幕上的视图随心所欲地摆放，而且视图的大小和位置会随着手机屏幕大小的变化做出调整。

### 4.5.1 框架布局（FrameLayout）

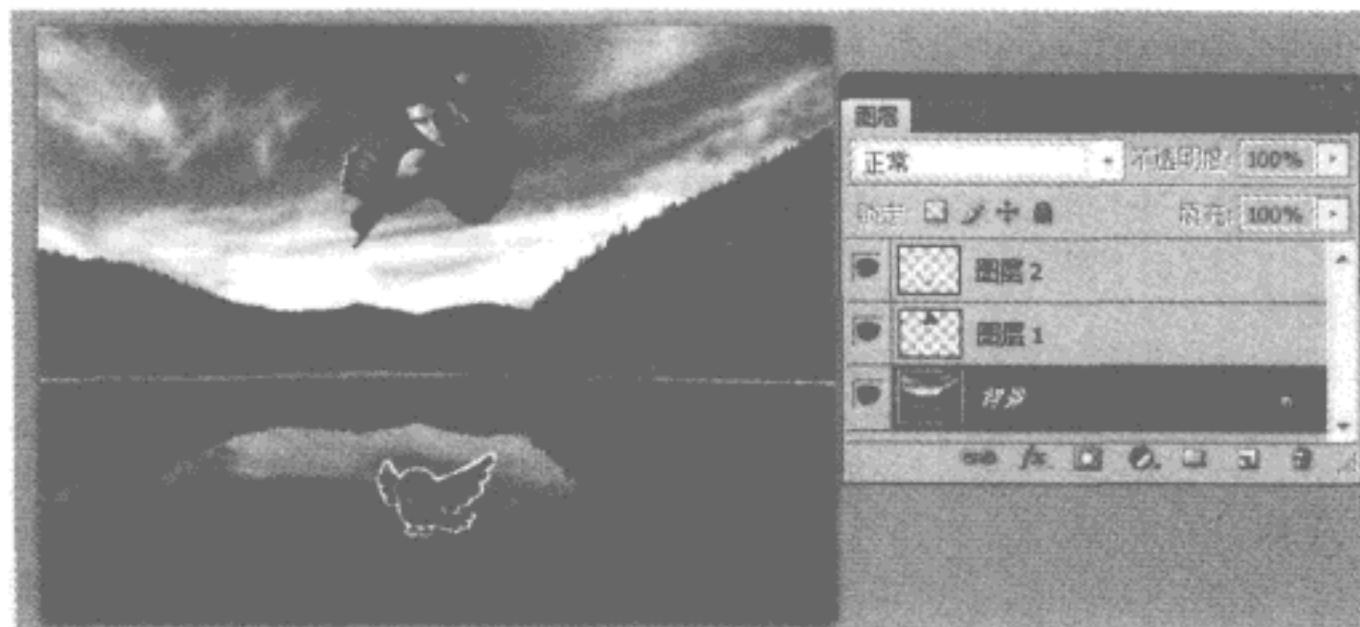
**工程目录：src\ch04\ch04\_framelayout**

框架布局是最简单的布局方式，所有添加到这个布局中的视图都以层叠的方式显示。第一个添加到框架布局中的视图显示在最底层，最后一个被放在最顶层，上一层的视图会覆盖下一层的视图。这种显示方式类似堆栈，栈顶的视图显示在最顶层，而栈底的视图显示在最底层。因此，也可以将 `FrameLayout` 称为堆栈布局。

框架布局在 XML 布局文件中应使用<FrameLayout>标签进行配置，如果使用 Java 代码，需要创建 android.widget.FrameLayout 对象。下面是一个典型的框架布局配置代码。

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
    <TextView android:id="@+id/textview" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button android:id="@+id/button" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</FrameLayout>
```

从框架布局的特性看，很像 Photoshop 中的图层。如果将框架布局中的视图放在不同的位置，大小也不同，就可以做出合成图的效果。图 4.17 是用 Photoshop 做的一个效果图。这个效果实际上是由 3 个图层（见右侧的图层列表）合成的。这 3 个图层分别是：背景、图层 1（超人）、图层 2（小鸟）。我们也可以利用框架布局和 ImageView 控件做出同样的效果。



▲ 图 4.17 Photoshop 中的图层

现在准备 3 张图（background.jpg、superman.png 和 bird.png），分别用 3 个 ImageView 显示。首先将 background.jpg 显示在第一个 ImageView 中（作为最底层的背景），superman.png 和 bird.png 的放置顺序可以随意（因为这两个图并不重合）。完整的 XML 布局文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <!-- 最底层的背景图 -->
    <ImageView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="@drawable/background"
        android:layout_gravity="center" />
    <!-- 显示小鸟图像 -->
    <ImageView android:layout_width="63dp"
        android:layout_height="46dp" android:background="@drawable/bird"
        android:layout_gravity="center" android:layout_marginTop="80dp" />
    <!-- 显示超人图像 -->
    <ImageView android:layout_width="85dp"
        android:layout_height="85dp" android:background="@drawable/superman"
        android:layout_gravity="center" android:layout_marginBottom="80dp" />
</FrameLayout>
```

在上面的代码中有几个属性需要说明一下。

- `android:layout_gravity`: 当前视图在父视图中的位置。在本例中该属性的值是 `center`, 表示在垂直和水平方向居中。
- `android:layout_marginTop`: 当前视图上边缘离某条基线的距离。例如, 当前的`<ImageView>`是居中的, 那么这条基线就是屏幕的中位线, 所以 `80dp` 就是`<ImageView>`顶端到这条基线的距离, 小鸟的图像会在基线下面的位置显示。
- `android:layout_marginBottom`: 当前视图下边缘离某条基线的距离。

本例还涉及一个计量单位 `dp`, 这个并不等同于像素点。在本节只要知道使用这个单位的长度、位置等属性值就会根据屏幕的分辨率自动调整。关于 Android 的各种计量单位和常用属性将在第 5 章详细介绍。

除了使用`<ImageView>`来显示背景图外, 还可以直接使用`<FrameLayout>`来显示背景图。因此, 上面的 XML 布局文件可以改成如下形式:

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:background="@drawable/background"
    android:layout_gravity="center">
    <!-- 显示小鸟图像 -->
    <ImageView android:layout_width="63dp" android:layout_height="46dp"
        android:background="@drawable/bird" android:layout_gravity="center"
        android:layout_marginTop="80dp" />
    <!-- 显示超人图像 -->
    <ImageView android:layout_width="85dp" android:layout_height="85dp"
        android:background="@drawable/superman" android:layout_gravity="center"
        android:layout_marginBottom="80dp" />
</FrameLayout>

```

运行程序, 会看到如图 4.18 所示的显示效果, 看看是否与图 4.17 所示的效果一样呢?



▲ 图 4.18 使用 FrameLayout 实现的图层效果

## 4.5.2 线性布局（LinearLayout）

**工程目录：**src\ch04\ch04\_linearlayout

线性布局是最常用的布局方式。线性布局在 XML 布局文件中使用<LinearLayout>标签进行配置，如果使用 Java 代码，需要创建 android.widget.LinearLayout 对象。

线性布局可分为水平线性布局和垂直线性布局。通过 android:orientation 属性可以设置线性布局的方向，该属性的可取值是 horizontal 和 vertical，默认值是 horizontal。当线性布局的方向是水平时，所有在<LinearLayout>标签中定义的视图都沿着水平方向线性排列。当线性布局的方向是垂直时，所有在<LinearLayout>标签中定义的视图都沿着垂直方向线性排列。

<LinearLayout>标签有一个非常重要的 gravity 属性，该属性用于控制布局中视图的位置。该属性可取的主要值如表 4.1 所示。如果设置多个属性值，需要使用“|”进行分隔。在属性值和“|”之间不能有其他符号（例如，空格、制表符等）。

**表 4.1** gravity 属性的取值

属性值	描述
top	将视图放到屏幕顶端
bottom	将视图放到屏幕底端
left	将视图放到屏幕左侧
right	将视图放到屏幕右侧
center_vertical	将视图按垂直方向居中显示
center_horizontal	将视图按水平方向居中显示
center	将视图按垂直和水平方向居中显示

在屏幕上添加 3 个按钮，并将它们右对齐，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="right">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="按钮 1" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="按钮 2" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="按钮 3" />
</LinearLayout>
```

使用上面的 XML 布局文件后，将得到如图 4.19 所示的效果。如果将 gravity 属性值改成 center，将得到如图 4.20 所示的效果。

<LinearLayout>标签中的子标签还可以使用 layout\_gravity 和 layout\_weight 属性来设置每一个视图的位置。



▲ 图 4.19 按钮右对齐



▲ 图 4.20 按钮中心对齐

`layout_gravity` 属性的可取值与 `gravity` 属性的可取值相同，表示当前视图在布局中的位置。`layout_weight` 属性是一个非负整数值，如果该属性值大于 0，线性布局会根据水平或垂直方向以及不同视图的 `layout_weight` 属性值占所有视图的 `layout_weight` 属性值之和的比例为这些视图分配自己所占用的区域，视图将按相应比例拉伸。例如，在`<LinearLayout>`标签中有两个`<Button>`标签，这两个标签的 `layout_weight` 属性值都是 1，并且`<LinearLayout>`标签的 `orientation` 属性值是 `horizontal`，这两个按钮都会被拉伸到屏幕宽度的一半，并显示在屏幕的正上方。如果 `layout_weight` 属性值为 0，视图会按原大小显示（不会被拉伸）。对于其余 `layout_weight` 属性值大于 0 的视图，系统将会减去 `layout_weight` 属性值为 0 的视图的宽度或高度，再用剩余的宽度和高度按相应的比例来分配每一个视图所占的宽度和高度。

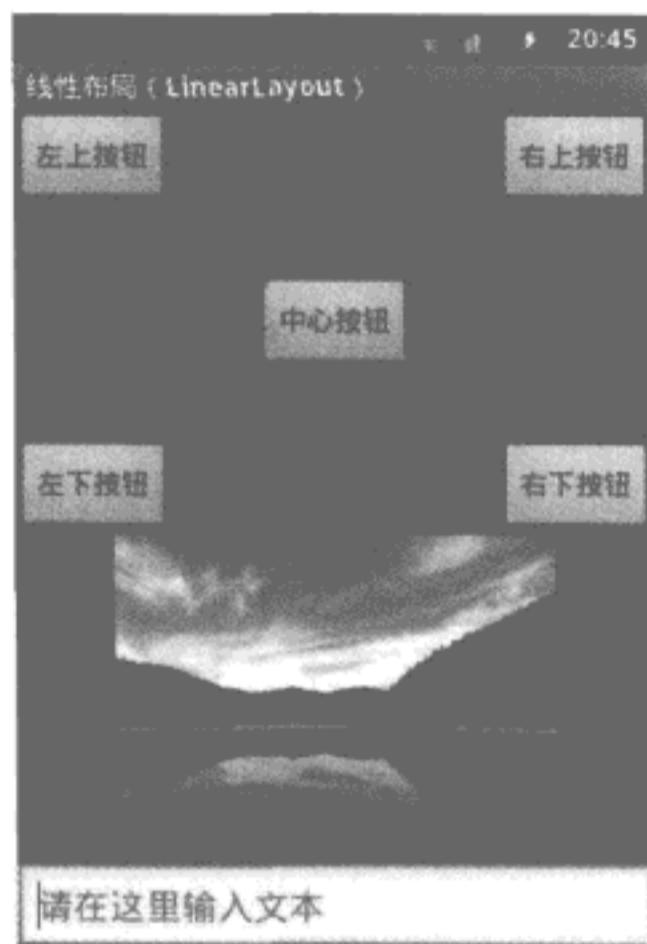
下面给出一个稍微复杂的线性布局的例子。在这个例子中将屏幕垂直分成相等的两部分，在第一部分的四角和中心分别放一个按钮，第二部分的最下方是一个文本输入框（`EditText`），放置一个 `ImageView`，用于显示图像。图 4.21 是最终的显示效果。

图 4.22 是布局中各个标签位置的示意图。我们可以看出，首先使用了两个`<LinearLayout>`标签将屏幕从垂直方向分成了两个相等部分。这两个`<LinearLayout>`标签应将 `android:height` 属性值设为 `fill_parent`，并且要将 `android:layout_weight` 属性值设为相等的值，如都设为 1。

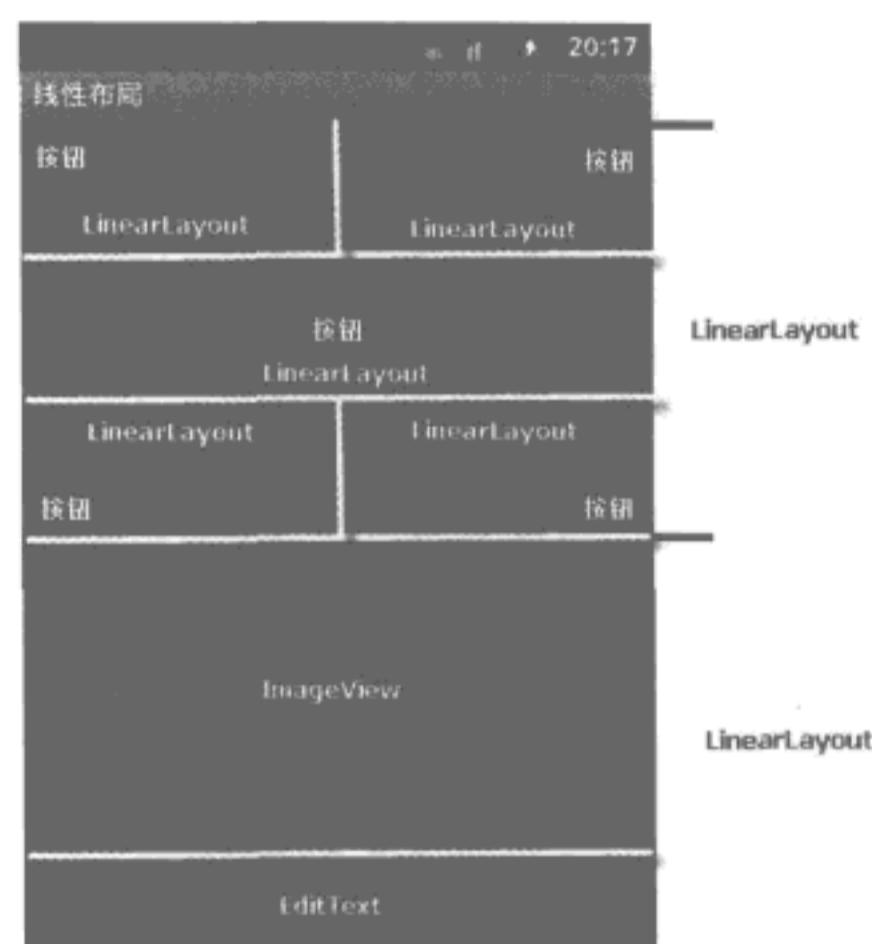
在第一部分又使用了 5 个`<LinearLayout>`标签将屏幕分成了 5 份，前两份和后两份分别要使用 `android:layout_weight` 属性进行等分，然后在每一个`<LinearLayout>`标签中使用 `android:gravity` 属性设置其中按钮的位置。

在第二部分的`<LinearLayout>`标签中包含了两个标签：`<ImageView>` 和 `<EditText>`。其中 `<EditText>` 放置在最下面，而`<ImageView>`则充满了`<EditText>`上方所有的空间。因此，`<ImageView>` 的 `android:height` 属性值要设为 `fill_parent`，不过这样一来`<ImageView>`就会在垂直方向充满第二部

分的整个空间，也就是说，把<EditText>挤没了。要想让<EditText>仍然可以显示在最下方，必须要设置<ImageView>标签的另外一个属性 android:layout\_weight。由于<EditText>上方只有一个<ImageView>标签，因此，android:layout\_weight 属性只要设为合法的值即可，在本例将该属性值设为 1。这样一来，<ImageView>就会先考虑其他未设置 android:layout\_weight 属性的标签（<EditText>就未设置该属性）的高度，然后再充满剩余的空间。这种方法经常被使用在屏幕最下方的按钮上。下面来看看完整的 XML 布局文件。



▲ 图 4.21 线性布局的例子



▲ 图 4.22 线性布局的排列位置

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1">
        <!-- 设置最上面两个按钮 -->
        <LinearLayout android:orientation="horizontal"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1">
            <!-- 包含左上角按钮的 LinearLayout 标签 -->
            <LinearLayout android:orientation="vertical"
                android:layout_width="fill_parent" android:layout_height="fill_parent"
                android:layout_weight="1">
                <Button android:layout_width="wrap_content"
                    android:layout_height="wrap_content" android:text="左上按钮"
                    android:layout_gravity="left" />
            </LinearLayout>
            <!-- 包含右上角按钮的 LinearLayout 标签 -->s
            <LinearLayout android:orientation="vertical"
                android:layout_width="fill_parent" android:layout_height="fill_parent"
                android:layout_weight="1">

```

```

        android:layout_weight="1">
    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="右上按钮"
            android:layout_gravity="right" />
    </LinearLayout>
</LinearLayout>
<!-- 包含中心按钮的 LinearLayout 标签 -->
<LinearLayout android:orientation="vertical"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1" android:gravity="center">
    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="中心按钮" />
</LinearLayout>
<!-- 设置最下面两个按钮 -->
<LinearLayout android:orientation="horizontal"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1">
    <!-- 包含左下角按钮的 LinearLayout 标签 -->
    <LinearLayout android:orientation="vertical"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1" android:gravity="left|bottom">
        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="左下按钮" />
    </LinearLayout>
    <!-- 包含右下角按钮的 LinearLayout 标签 -->
    <LinearLayout android:orientation="vertical"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1" android:gravity="right|bottom">
        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="右下按钮" />
    </LinearLayout>
</LinearLayout>
<LinearLayout android:orientation="vertical"
            android:layout_width="fill_parent" android:layout_height="fill_parent"
            android:layout_weight="1">
    <!-- 在第二部分上方显示的 ImageView 标签 -->
    <ImageView android:layout_width="fill_parent"
            android:layout_height="fill_parent" android:src="@drawable/background"
            android:layout_weight="1" />
    <!-- 在第二部分最下方显示的 EditText 标签 -->
    <EditText android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:hint="请在这里输入文本" />
</LinearLayout>
</LinearLayout>

```

### 扩展学习：<FrameLayout>标签的妙用

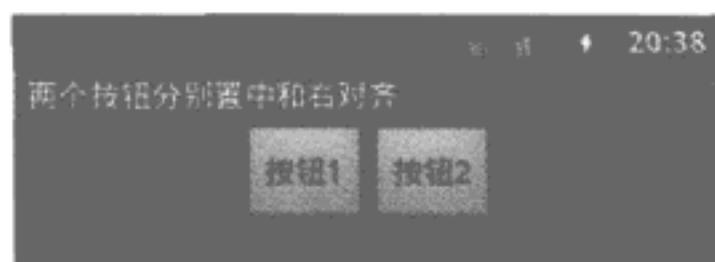
每一个都可以使用 `layout_gravity` 属性来安排自己的位置。那么现在我们要设计一个布局，有两个按钮，第一个按钮水平居中，第二个按钮右对齐，如图 4.23 所示。

读者可以先不看后面的答案，看看能否设计出如图 4.23 所示的布局。



▲图 4.23 居中和右对齐的两个按钮

由于本节讲的是线性布局，下面我们看看利用线性布局是否能很容易地做出如图 4.23 所示的效果。如果要两个按钮水平排列，需要将<LinearLayout>标签的 android:orientation 属性值设为 horizontal。如果我们将<LinearLayout>标签的 android:gravity 属性值设为 center\_horizontal，两个按钮就会连在一起，并且水平居中，如图 4.24 所示。如果将<LinearLayout>标签的 android:gravity 属性值设为 right，则两个按钮都会右对齐，如图 4.25 所示。总之，没有办法使用 android:gravity 属性使两个按钮分别居中和右对齐。而<Button>标签的 android:layout\_gravity 属性值水平居中和右对齐需要将<LinearLayout>标签的 android:orientation 属性值设为 vertical。因此，通过 android:layout\_gravity 属性也无法达到如图 4.23 所示的效果。



▲图 4.24 两个居中显示的按钮



▲图 4.25 右对齐的两个按钮

既然<LinearLayout>无法达到目的，那么我们可以考虑一下 4.5.1 节介绍的<FrameLayout>标签。利用这个标签的图层效果来达到目的。也就是说，如图 4.23 所示的两个按钮分别位于两个图层中，“按钮 1”在图层 1 中居中（将 android:layout\_gravity 属性值设为 center\_horizontal），“按钮 2”在图层 2 中右对齐（将 android:layout\_gravity 属性值设为 right）。这两个按钮在<FrameLayout>标签中先定义谁都没有关系。在 4.5.7 节我们还会看到使用<merge>来代替<FrameLayout>。实现如图 4.23 所示效果的布局代码如下：

**代码位置：src\ch04\ch04\_middle\_right\res\layout\main.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="按钮 1"
        android:layout_gravity="center_horizontal" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="按钮 2"
        android:layout_gravity="right" />
</FrameLayout>
```

### 4.5.3 相对布局 (RelativeLayout)

**工程目录：src\ch04\ch04\_relativelayout**

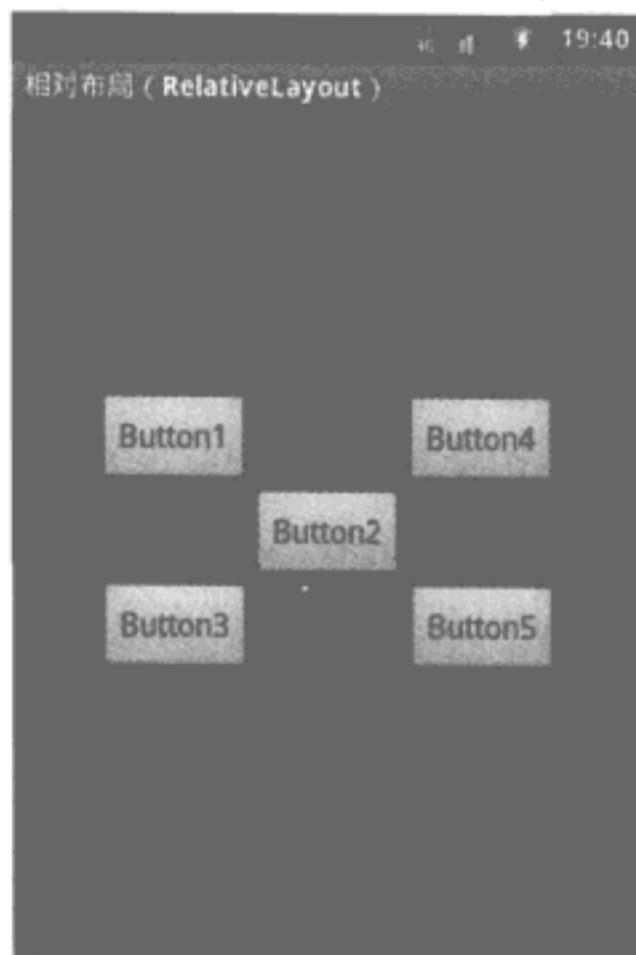
相对布局可以设置某一个视图相对于其他视图的位置，这些位置包括上、下、左、右。设置这

些位置的属性是 `android:layout_above`、`android:layout_below`、`android:layout_toLeftOf`、`android:layout_toRightOf`。除此之外，还可以通过 `android:layout_alignBaseline` 属性设置视图的底端对齐。

这 5 个属性的值必须是存在的资源 ID，也就是另一个视图的 `android:id` 属性值。下面的代码是一个典型的使用 `RelativeLayout` 的例子。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent" >
    <TextView android:id="@+id/textview1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="20dp"
        android:text="文本 1"/>
    <!-- 将这个 TextView 放在 textview1 的右侧 -->
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="20dp"
        android:text="文件 2" android:layout_toRightOf="@+id/textview1"/>
</RelativeLayout>
```

下面我们用相对布局实现如图 4.26 所示的梅花图案效果。



▲ 图 4.26 使用相对布局实现的梅花图案

如图 4.26 所示界面的基本思想是先将“Button1”放在左上角，然后将“Button2”放在“Button1”的右下侧，最后以“Button2”为轴心，放置“Button3”、“Button4”和“Button5”。在设置完 5 个按钮后，再将`<RelativeLayout>`标签的 `android:gravity` 属性值设为 `center`。布局的完整代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:gravity="center">
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="16dp"
        android:text="Button1" />
    <Button android:id="@+id/button2" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="16dp"
        android:text="Button2" />
    <Button android:id="@+id/button3" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="16dp"
        android:text="Button3" />
    <Button android:id="@+id/button4" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="16dp"
        android:text="Button4" />
    <Button android:id="@+id/button5" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="16dp"
        android:text="Button5" />
</RelativeLayout>
```

```

    android:layout_height="wrap_content" android:textSize="16dp"
    android:text="Button2" android:layout_toRightOf="@+id/button1"
    android:layout_below="@+id/button1" />
<Button android:id="@+id/button3" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:textSize="16dp"
    android:text="Button3" android:layout_toLeftOf="@+id/button2"
    android:layout_below="@+id/button2" />
<Button android:id="@+id/button4" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:textSize="16dp"
    android:text="Button4" android:layout_toRightOf="@+id/button2"
    android:layout_above="@+id/button2" />
<Button android:id="@+id/button5" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:textSize="16dp"
    android:text="Button5" android:layout_toRightOf="@+id/button2"
    android:layout_below="@+id/button2" />
</RelativeLayout>

```

#### 4.5.4 表格布局 (TableLayout)

工程目录: src\ch04\ch04\_tablelayout

表格布局可将视图按行、列进行排列。一个表格布局由一个`<TableLayout>`标签和若干`<TableRow>`标签组成。下面我们使用表格布局来实现如图 4.27 所示的效果。



▲ 图 4.27 表格布局的效果

完整的布局代码如下:

```

<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent" >
    <TableRow android:paddingTop="20dp" >
        <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:src="@drawable/christmas_background1_small" />
        <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:src="@drawable/christmas_background2_small" android:paddingLeft="20dp"/>
        <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:src="@drawable/christmas_background3_small" android:paddingLeft="20dp"/>
        <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:src="@drawable/christmas_background4_small" android:paddingLeft="20dp"/>
    </TableRow>
    <TableRow android:paddingTop="20dp">

```

```

<ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:src="@drawable/christmas_background5_small" />
<ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:src="@drawable/christmas_background6_small" android:paddingLeft="20dp"/>
<ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:src="@drawable/christmas_background7_small" android:paddingLeft="20dp"/>
<ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:src="@drawable/christmas_background8_small" android:paddingLeft="20dp"/>
</TableRow>
</TableLayout>

```

表格布局在实现行列效果中并不常用，一般会使用 GridView 控件来代替表格布局，该控件将在第 5 章详细介绍。

#### 4.5.5 绝对布局 ( AbsoluteLayout )

通过绝对布局，可以任意设置视图的位置。通过 android:layout\_x 和 android:layout\_y 属性可以设置视图的横坐标和纵坐标，如下面的代码所示：

```

<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_x="40dp" android:layout_y="80dp"
        android:text="按钮" />
</AbsoluteLayout>

```

#### 4.5.6 重用 XML 布局文件

**工程目录：src\ch04\ch04\_reusablelayout**

在一个复杂的应用程序中往往同样的布局要在多处使用。当然，最笨的方法是在所有需要的地方重复编写这些布局。这种方法固然可行，不过一旦需要修改布局，维护其一致性恐怕就要增加很大的工作量（有的布局可能会被重复使用上百次，甚至更多）。看到这，也许很多读者会和笔者有同样的忧虑，但当看完本节后，所有的忧虑都会统统消失。

在 XML 布局文件中提供了一个<include>标签（注意，include 首字母要小写），通过这个标签，可以在一个布局文件中引用另外一个布局文件。这样我们就可以将在多处使用的布局单独放在一个或多个布局文件中，然后在使用到这些布局文件时用<include>标签来引用。

例如，我们有一个叫 workspace\_screen.xml 的布局文件，在另一个布局文件中需要使用 3 次，那么可以使用如下的代码。

```

<com.android.launcher.Workspace
    android:id="@+id/workspace"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    launcher:defaultScreen="1">
    <!-- 引用三次 workspace_screen -->
    <include android:id="@+id/cell1" layout="@layout/workspace_screen" />
    <include android:id="@+id/cell2" layout="@layout/workspace_screen" />
    <include android:id="@+id/cell3" layout="@layout/workspace_screen" />

```

```
</com.android.launcher.Workspace>
```

`<include>`标签只有 `layout` 属性是必选的，该属性没有 `android` 命名空间前缀。该属性的值是布局文件的资源 ID，如 `@layout/workspace_screen`。在上面的代码中`<include>`标签还使用了一个 `android:id` 属性，实际上，该属性指定的是 `workspace_screen.xml` 布局文件中根节点的 `android:id` 属性值。如果根节点已经设置了 `android:id` 属性值，那么`<include>`标签的 `android:id` 属性值将覆盖 `workspace_screen.xml` 布局文件中根节点的 `android:id` 属性值。`<include>`标签还可以覆盖被引用的布局文件根节点的所有与布局有关的属性，也就是以“`android:layout_`”开头的属性，例如，`android:layout_width`、`android:layout_height` 等。通过覆盖属性值，可以使被引用的布局文件中的视图拥有不同的布局风格。例如，下面的布局代码引用了 `image_holder.xml` 文件两次，但只有第一个`<include>`标签覆盖了一些属性。

```
<!-- override the layout height and width -->
<include layout="@layout/image_holder"
    android:layout_height="fill_parent"
    android:layout_width="fill_parent" />
<!--下面的 include 标签没有覆盖任何属性 -->
<include layout="@layout/image_holder" />
```

### ■ 注意

如果想覆盖布局的尺寸，必须同时覆盖 `android:layout_weight` 和 `android:layout_height`。不能只覆盖其中一个属性，否则对这两个属性值的覆盖无效。

`<include>`标签在设计与设备相关的布局文件时非常有用，例如，当手机横屏（land）和竖屏（portrait）时可以使用不同的布局文件，但有可能很多视图的布局是相同的。这样就可以在 `layout` 目录中放置横屏和竖屏都需要用到的布局，而在 `layout-land` 和 `layout-port` 目录中的布局文件可以使用`<include>`标签引用这些布局。在本书的例子中将会看到多次使用`<include>`来设计布局。

`<include>`标签还可以简化布局文件的复杂度，例如，在 4.5.2 节使用了一个较复杂的布局文件。我们可以使用`<include>`标签将其简化成 5 个小的布局文件，从而使布局文件更容易阅读和使用。现在我们来看一下这 6 个布局文件。

### main.xml

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

我们从 `main.xml` 文件可以看出，已经将原来复杂的布局代码简化成了几行代码，其中使用`<include>`标签引用了 `first.xml` 和 `second.xml` 文件，它们分别表示布局的第一部分和第二部分。

### first.xml（第一部分的布局文件引用了 3 个更小的布局文件）

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```

    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_weight="1">
        <include layout="@layout/first_top" />
        <include layout="@layout/first_middle" />
        <include layout="@layout/first_bottom" />
    
```

在 first.xml 文件中引用了 first\_top.xml、first\_middle.xml 和 first\_bottom.xml 这 3 个布局文件，表示第一部分的上、中、下这 3 个小部分。这 3 个布局文件的内容如下：

### **first\_top.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_weight="1">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1">
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="左上按钮"
            android:layout_gravity="left" />
    </LinearLayout>
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1">
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="右上按钮"
            android:layout_gravity="right" />
    </LinearLayout>
</LinearLayout>

```

### **first\_middle.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_weight="1"
    android:gravity="center">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="中心按钮" />
</LinearLayout>

```

### **first\_bottom.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_weight="1">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1" android:gravity="left|bottom">
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="左下按钮" />
    </LinearLayout>
</LinearLayout>

```

```

</LinearLayout>
<LinearLayout android:orientation="vertical"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:layout_weight="1" android:gravity="right|bottom">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="右下按钮" />
</LinearLayout>
</LinearLayout>

```

下面来看看第二部分的布局文件。

### second.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_weight="1">
    <ImageView android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:src="@drawable/background"
        android:layout_weight="1" />
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:hint="请在这里输入文本" />
</LinearLayout>

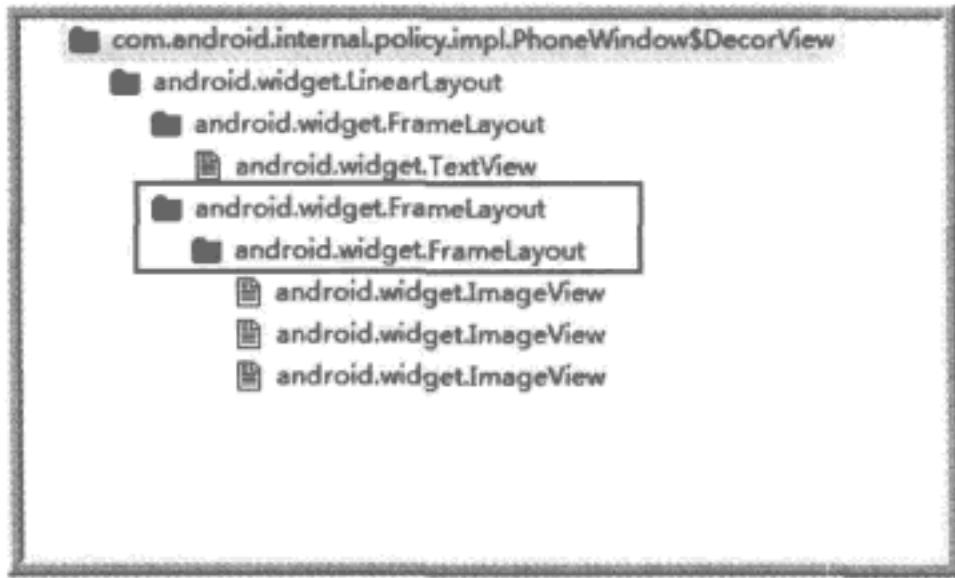
```

## 4.5.7 优化 XML 布局文件

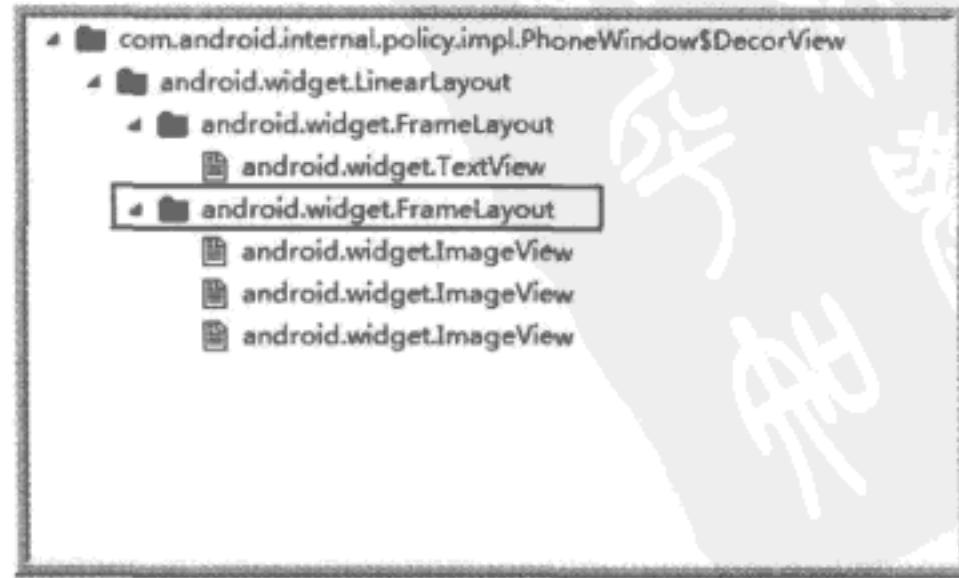
工程目录: src\ch04\ch04\_merge

让我们先使用 hierarchyviewer( Android SDK 自带的一个分析视图类层次的工具, 运行<Android SDK 安装目录>\tools\ hierarchyviewer.bat 可启动该工具) 分析 4.5.1 节的布局文件。启动 hierarchyviewer, 并运行 4.5.1 节中的例子, 然后单击 hierarchyviewer 界面上方的“Refresh Tree”按钮, 会在界面的左侧显示模拟器中运行的程序的当前界面的视图类层次结构, 如图 4.28 所示。

也许很多读者会感到奇怪, 为什么会出现两个<FrameLayout>标签呢? 明明在布局文件中只使用了一个<FrameLayout>标签。实际上, 任何一个布局文件, 无论根节点是<LinearLayout>、<RelativeLayout>还是<FrameLayout>, Android 系统都会在上一层添加一个<FrameLayout>。也就是说, 任何的布局文件都会被包含在<FrameLayout>标签中。了解了其中的原理, 也就不奇怪为什么会出现两个<FrameLayout>标签了。



▲ 图 4.28 未优化的视图类层次结构



▲ 图 4.29 优化后的视图类层次结构

为了解决这个问题，可以使用<merge>标签代替<FrameLayout>标签，系统在遇到<merge>标签后，会自动忽略这个标签，这样就只剩下一个<FrameLayout>标签了，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="@drawable/background"
        android:layout_gravity="center" />
    <ImageView android:layout_width="63dp"
        android:layout_height="46dp" android:background="@drawable/bird"
        android:layout_gravity="center" android:layout_marginTop="80dp"/>
    <ImageView android:layout_width="85dp"
        android:layout_height="85dp" android:background="@drawable/superman"
        android:layout_gravity="center" android:layout_marginBottom="80dp"/>
</merge>
```

再次运行这个程序，会在 hierarchyviewer 中看到如图 4.29 所示的视图类的层次结构，看看<FrameLayout>标签是否少了一个呢！

#### 注意

<merge>标签虽然能使视图类层次结构更简洁，但该标签只对<FrameLayout>标签有效。因为当<merge>标签被忽略时，上层还有一个<FrameLayout>标签。<merge>不能使用在其他的布局中，这是因为系统是不会将<merge>转换成<LinearLayout>、<RelativeLayout>标签的，再说系统也不知道如何转换。所以如果使用<merge>代替<LinearLayout>、<RelativeLayout>等布局标题，这些标签就消失了，这样就相当于仍然使用了<FrameLayout>标签。

#### 4.5.8 查看 apk 文件中的布局

可能很多初学者对 Android 中的布局还不太习惯，这就需要通过大量的练习来逐渐适应它。学习编写程序最好的方法无疑是看大量高质量的源代码，学习使用布局也是一样。现在网上有非常多的 Android 程序，它们的界面也非常漂亮。如果能得到这些软件的布局文件那真是再好不过了，可以通过模仿来更好地掌握 Android 的布局。但遗憾的是，这些软件大多都不开源，因此，无法直接获得这些软件的 XML 布局文件源代码。

虽然 apk 文件是 zip 格式的压缩文件，使用像 winzip 一类的解压软件很容易将 apk 文件解开，但里面的 XML 布局文件仍然是被编译的，全部都是二进制格式的内容。为了将其还原，我们可以借助一个叫 AXMLPrinter2 的工具。这个工具实际上就是一个 jar 包，因此，在使用它时需要 Java 的运行环境。假设该工具（AXMLPrinter2.jar）被放在 D:\lib 目录，在该目录下建立一个 axml.cmd 文件（只限于 Windows XP 及以上系统），打开该文件，并输入如下内容。

```
| java -jar D:\lib\AXMLPrinter2.jar %1 > %2
```

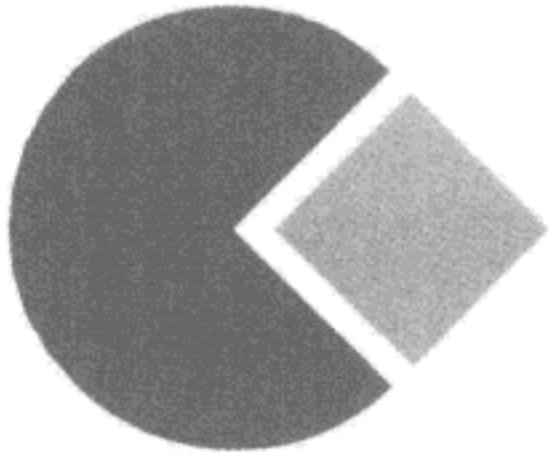
为了使用更方便，可以在 PATH 环境变量中加入 D:\lib。在控制台中进入任何一个包含 apk 文件的目录，将其解压，并找到一个 XML 布局文件，例如，main.xml，在控制台中输入如下的命令，

可将 main.xml 文件还原成 output.xml 文件，打开 output.xml 文件可直接查看其中的内容。

```
| axml.cmd main.xml output.xml
```

## 4.6 小结

本章介绍了设置用户界面必须掌握的知识，主要包括 Activity、View 及布局三部分。其中 Activity 之间的数据传递非常重要，因为几乎所有的程序都涉及 Activity 之间的数据传递（除非程序只包含一个 Activity），掌握并灵活应用这些不同的数据传递方法对开发 Android 应用程序会起到非常大的帮助。本章的最后介绍了 Android 支持的 5 种布局以及如何引用和优化 XML 布局文件，这对于编写大型、高效的应用程序至关重要。



## 第5章 良好的学习开端——控件（Widget）详解

所有带界面的 Android 程序都必须使用到 Android 控件。几乎所有的控件都在 android.widget 包中。要想实现绚丽的用户界面，必须认真学好本章的内容。本章将向读者展示 android.widget 包中所有控件的详细用法，并附有大量有价值的源代码供读者练习。

### 5.1 常用 XML 属性解析

由于 android.widget 包中的所有可视控件都是 android.view.View 的子类，因此，与这些控件相对应的 XML 属性（在布局文件中设置的属性）有一部分继承于 View 类的 XML 属性。也就是说，所有在 android.widget 包中的可视控件都拥有这些 XML 属性。虽然前面的内容已经多次使用过某些 XML 属性，但在本节还是需要重新讲解和总结一下，以便在学习控件的过程中对这些属性有更深入的理解，这也有利于我们继续学习 Android 的高级技术。

#### 5.1.1 android:id 属性

如果要在代码或 XML 布局文件中引用某个控件，该控件必须要设置 android:id 属性。这个属性需要设置一个引用（Reference）类型的值，格式如下：

| @+id/value 或 @+id/value

其中 value 表示任意符合 Java 语言标识符规范的字符串。因为系统要根据 value 在 R.id 类中生成一个 int 类型的变量，value 就是变量名。

第一种格式（@id/value）中的 value 必须在 R.id 类中存在，也就是说，要为当前控件指定一个在 R.id 类中已经存在的 int 类型变量作为其 ID 值，如果该变量不存在，XML 布局文件无法验证通过。第二种格式（@+id/value）在 @ 和 id 之间多了个加号（+），表示如果 value 指定的变量名在 R.id 中存在，则使用已经存在的变量值作为其 ID 值，否则，在 R.id 类中新建一个 int 类型的变量。笔者建议使用第二种形式，因为我们并不能保证 R.id 类中一定存在指定的变量。

#### 5.1.2 控件的宽度（android:layout\_width）和高度（android:layout\_height）

android:layout\_width 和 android:layout\_height 分别用来设置控件的宽度和高度，这两个属性是必选的。可设置的值除了 fill\_parent 和 wrap\_content 外，还可以设置准确的值。其中 fill\_parent 表

示控件的宽度或高度尽可能充满父控件的空间（如果当前控件是最上层，就是充满整个屏幕）。`wrap_content` 表示控件的宽度或高度根据控件中的内容来确定，也就是说，在满足完全显示控件内容的情况下，将宽度和高度设为最小。

如果设置准确的值需要带单位，一般可设为 `px` 或 `dp`（也可设为 `dip`）。`px` 表示屏幕的像素点，如 `android:layout_width="100px"`，`dp` 类似于像素点，但该单位与屏幕密度相关。如果屏幕的密度是 160（HVGA、分辨率不一般为 320\*480），`1dp` 相当于 `1px`。当屏幕的密度大于或小于 160 时，系统会将单位为 `dp` 的值换算成相应的像素点(`px`)。因此，`android:layout_width` 和 `android:layout_height` 的值使用 `dp` 作为单位会随着屏幕的密度变化而变化。当密度变化了，分辨率也会变化，如密度从 160 变成 240，分辨率会从 320\*480 变成了 480\*800（或其他相近的分辨率），如果这时仍然使用 `px` 作为单位，那么控件的宽度或高度仍然保持着原来的像素点数，这样看起来控件就在大密度的分辨率屏幕中看起来更小了。因此，笔者建议所有与宽度、高度、距离相关的属性都使用 `dp` 作为单位。

**扩展学习：**在 Android 系统中屏幕的密度表示每英寸（相当于 2.54 厘米）包含的像素点数。例如，密度为 160 的屏幕每英寸有 160 个像素点，密度为 240 的屏幕每英寸有 240 个像素点。因此，屏幕密度越大，显示效果越细腻。

### 5.1.3 android:layout\_margin 属性

设置控件到相邻的控件或边缘的距离可以使用 `android:layout_margin` 属性，该属性设置了 4 个方向的距离，也就是左 (Left)、右 (Right)、上 (Top)、下 (Bottom)。如果要想单独设置某一个方向的距离，可以单独使用 `android:layout_marginLeft`、`android:layout_marginRight`、`android:layout_marginTop` 和 `android:layout_marginBottom` 这 4 个属性。当同时设置 `android:layout_margin` 和这 4 个属性时，以 `android:layout_margin` 的属性值为准。例如，同时设置了 `android:layout_margin` 和 `android:layout_marginLeft` 的属性值，系统仍然会优先使用 `android:layout_margin` 属性值作为当前控件距离左侧相邻控件或边缘的距离。

上述 5 个属性在设置时仍然需要带单位的值，例如，`android:layout_margin="50dp"`。

### 5.1.4 android:padding 属性

`android:padding` 属性用于设置控件内容在 4 个方向距离控件边缘的距离。与 `android:layout_margin` 属性类似，也有 4 个属性用于分别设置控件内容在 4 个方向距离控件边缘的距离，这 4 个属性是 `android:paddingLeft`、`android:paddingTop`、`android:paddingRight`、`android:paddingBottom`。与 `android:layout_margin` 相同，`android:padding` 的优先级要比其余 4 个属性高。

初学者经常会混淆 `android:layout_margin` 和 `android:padding` 的概念。`android:layout_margin` 指的是控件与控件或控件与边缘之间的距离，而 `android:padding` 指的是控件内容距离控件边缘的距离，有点类似于 Word 的页边距。例如，`<TextView>` 标签的 `android:padding` 属性值为 `20dp`，表示 `TextView` 控件中的文本在 4 个方向（左、上、右、下）距离 `TextView` 的边缘都是 `20dp`。当然，系统会根据 `android:layout_width` 和 `android:layout_height` 属性值的不同只满足部分方向的边距。例如，如果当前控件的 `android:layout_height` 属性值为 `wrap_content`，而 `android:padding` 属性的值设得很大，这时控件就会只满足距离顶端的边距，而距离底边的边距会变得很小。

### 5.1.5 android:layout\_weight 属性

这个属性在进行均衡布局时非常有用。所谓均衡布局，是指两个或多个控件要占用等比例的区域，它们所占的比例不因屏幕方向变化、屏幕密度变化以及总宽度或高度变化而改变。例如，有3个按钮，要求在水平方向各占1/3的长度，就可以将这3个控件的 android:layout\_weight 属性设为相等的3个值（必须是正整数，不需要加任何单位，只设置一个正整数即可），例如， android:layout\_weight="1"。

android:layout\_weight 属性还有很多巧妙的使用方法，例如，在 4.5.2 节的例子中将<ImageView>标签的 android:layout\_weight 属性值设为 1， android:layout\_height 属性值为 fill\_parent，而<TextView>标签并未设置 android:layout\_weight 属性，而且 android:layout\_height 属性值为 wrap\_content。如果<TextView>在<ImageView>的下方，那么系统会首先考虑<TextView>的高度，在将<TextView>放在最底端时再使用剩余的高度分配给所有设置了 android:layout\_weight 属性的控件。如果在这个例子中不设置<ImageView>标签的 android:layout\_weight 属性，则 ImageView 控件会充满整个区域显示，也就是说无法再看到 TextView 控件了。

### 5.1.6 android:layout\_gravity 和 android:gravity 属性

这两个属性分别表示控件和控件中内容的位置。例如，在一个<LinearLayout>标签中包含了一个<Button>按钮，要想使 Button 控件在水平方向居中，即可以在<Button>标签中使用 android:layout\_gravity 属性，也可以在<LinearLayout>标签中使用 android:gravity 属性。这两个布局的代码如下：

在<Button>标签中使用 android:layout\_gravity 属性。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent" android:layout_
    height="fill_parent">
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="水平居中"
        android:layout_gravity="center_horizontal"/>
</LinearLayout>
```

在<LinearLayout>标签中使用 android:gravity 属性。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="center_horizontal">
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="水平居中" />
</LinearLayout>
```

android:layout\_gravity 和 android:gravity 可设置的属性值完全一样，常用的属性值如下。



- center\_horizontal: 水平居中。
- center\_vertical: 垂直居中。
- center: 在水平和垂直两个方向居中。
- left: 设置到左侧。
- right: 设置到右侧。
- top: 设置到顶端。
- bottom: 设置到底端。

如果使用 `android:layout_gravity` 属性，并不是上述的属性值都起作用。例如，`android:layout_gravity` 属性的值要根据`<LinearLayout>`标签的 `android:orientation` 属性值确定，如 `android:orientation` 属性值为 `vertical`，则只有 `center_vertical`、`left`、`right` 起作用，`center` 属性值只保留了水平居中的特性。如果将 `android:orientation` 属性值设为 `horizontal`，则只有 `center_vertical`、`top`、`bottom` 起作用，`center` 只保留了垂直居中的特性。如果要设置多个值，如让控件在屏幕的左下角，需要同时将 `android:layout_gravity` 属性值设为 `left` 和 `bottom`。在这种情况下，两个属性值中间要用竖线 (|) 连接，两个属性值和竖线之间不能有任何空格、制表符等字符。

### 5.1.7 android:visibility 属性

该属性决定了当前控件是否可见。可设置如下 3 个值。

- visible: 控件可见。
- invisible: 控件不可见，但保留控件的位置。
- gone: 控件不可见，也不保留控件的位置。

其中 `invisible` 和 `gone` 都可以使控件不可见。使用 `invisible` 时控件虽然不可见，但控件所占的位置还在，相当于完全透明的控件，而使用 `gone` 会使控件彻底消失。相邻的控件会来填充由于控件消失而留下的位置。

### 5.1.8 android:background 属性

该属性用于设置控件背景色或背景图。如果设置背景色，需要使用`#color` 形式的属性值，其中 `color` 表示一个十六进制的颜色值。颜色值可以有如下的几种形式：

- RGB；
- ARGB；
- RRGGBB；
- AARRGGBB。

A、R、G、B 表示一个 0 至 F 的十六进制的数。其中 R、G、B 表示三原色，也就是红、绿、蓝，A 表示透明度，也就是 Alpha 值。A、R、G、B 的取值范围都是 0~FF。R、G、B 的取值越大，颜色越深。如果 R、G、B 都等于 0，表示的颜色是黑色；都为 F，表示的颜色是白色。R、G、B 三个值相等时表示灰度值。R、G、B 总共可表示 16,777,216 (2 的 24 次方) 种颜色。A 取 0 时表示完全透明，取 F 时表示不透明。如果采用前两种颜色值表示法，颜色和透明度的取值范围是 0~F，这并不意味着是颜色范围的 256 个值的前 15 个，而是将每一个值扩展成两位。例如，#FO0 相

当于#FF0000; #A567 相当于#AA556677。从这一点可以看出，#RGB 和#ARGB 可设置的颜色值并不多，它们的限制条件是颜色值和透明度的 8 位字节的高 4 位和低 4 位相同。其他的颜色值必须使用后两种形式设置。

`android:background` 属性还可以设置背景图，格式为@drawable/resourceId，其中 resourceId 表示图像的资源 ID（位于 res/drawable 目录中的图像文件在 R.drawable 类中对应的 int 类型变量），例如，下面的代码设置了`<LinearLayout>`标签的背景图。

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

### 5.1.9 指定单击事件方法（`android:onClick` 属性）

如果在程序中不需要引用控件，而只想触发控件的单击事件，就不需要指定 `android:id` 属性。但如果指定 `android:id` 属性，就无法使用代码来装载控件，也就无法为控件指定事件处理对象（`listener` 对象）了这就使我们陷入了两难境地。

从 Android 1.6 开始，视图（View）多了一个 `android:onClick` 属性，通过该属性，可以直接指定当前装载布局文件的 Activity 类中单击事件的方法名。该方法的定义要与单击事件一致，方法名可以任意指定。如下面的代码所示。

```
// 该方法必须被声明为 public
public void onClick_Connect(View view)
{
    ...
}
```

在控件标签中可以使用 `android:onClick` 属性来指定该方法，代码如下：

```
<Button android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:text="连接" android:onClick="onClick_Connect"/>
```

除了 `android:onClick` 属性外，还有另外两个与单击事件相关的属性：`android:clickable` 和 `android:longClickable`。这两个属性都需要设置布尔类型的值，即 true 或 false。其中 `android:clickable` 属性的值为 false，表示当前控件不接受单击事件。如果 `android:longClickable` 属性的值为 false，表示当前控件不接受长按单击事件（按住控件不动，则触发长按单击事件）。

### 5.1.10 控件焦点属性（`android:focusable` 和 `android:focusableInTouchMode`）

大多数控件都可以获得焦点，也就是处在选中状态。如果 `android:focusable` 属性值为 true，表示可以通过键盘（虚拟键盘或物理键盘）或轨迹球将焦点移动到当前控件上。如果该属性值为 false，则无法将焦点移到当前控件。

在默认情况下，触摸一个按钮虽然可以触发控件的单击事件，但无法使控件处在焦点状态。而将 `android:focusableInTouchMode` 属性值设为 true，当触摸某个控件时，会先将焦点移动到被触摸

的控件上，然后需要再触摸该控件才会响应单击事件。但要注意，需要将 android:focusable 属性值设为 true，当前控件才可能获得焦点，否则当前控件无论使用任何方式都无法获得焦点。

## 5.2 TextView（显示文本的控件）

TextView 控件我想大家一定不陌生了，不仅在前面多次使用了这个控件，而且用 ADT 建立的 Eclipse Android 工程也会在默认生成的 XML 布局文件中添加一个 TextView 控件。所以几乎所有的 Android 初学者接触到的第一个控件都是 TextView，该控件在 XML 布局文件中使用<TextView>标签定义。

虽然前面的章节已经不止一次使用了 TextView 控件，读者对这个控件已经再熟悉不过了，但前面的部分涉及的只是 TextView 控件非常初级的用法，TextView 控件的功能还远不止显示文本这么简单。在本节将逐一介绍 TextView 控件最为诱人的功能。

### 5.2.1 显示富文本（URL、不同大小、字体、颜色的文本）

工程目录：src\ch05\ch05\_show\_richtext

在 TextView 类中预定义了一些类似 HTML 的标签，通过这些标签，可以使 TextView 控件显示不同颜色、大小、字体的文字。下面是几个比较常用的标签。

- <font>：设置颜色和字体。
- <big>：设置大号字。
- <small>：设置小号字。如果<big>和<small>都没有，表示正常字号。
- <i>：斜体。
- <b>：粗体。
- <tt>：等宽字体（Monospace）。
- <br>：换行（行与行之间没有空行，相当于“\n”）。
- <p>：换行（行与行之间有空行，相当于“\n\n”）。对于带标签的文本，直接使用“\n”无法换行，只能使用<br>或<p>。
- <a>：链接地址。
- <img>：插入图像。

上面标签中的<img>将在下一节详细介绍，在本节只介绍前面几个标签的使用方法。这些标签虽然和 HTML 的标签类似，但并不具备 HTML 标签的全部功能。例如，<font>标签只支持 color 和 face 两个属性。

在使用这些标签时不能将带这些标签的字符串直接使用 TextView.setText 方法进行设置，而需要使用 Html.fromHtml 方法将带标签的字符串转换成 CharSequence 对象，然后再使用 TextView.setText 方法进行设置。

如果想在显示的文本中将 URL 地址、E-mail 地址、电话等特殊内容高亮显示，并在单击时触发相应的动作（URL 地址会调用浏览器显示该网址，电话会直接在拨号界面显示电话号），可以设置<TextView>标签的 android:autoLink 属性，该属性可设置的值如表 5.1 所示。



PDF

表 5.1

android:autoLink 属性可设置的值

autoLink 属性的值	功 能 描 述
none	不匹配任何链接（默认值）
web	匹配 Web 网址
email	匹配 E-mail 地址
phone	匹配电话号码
map	匹配映射地址
all	匹配所有的链接

我们可以根据需要设置 android:autoLink 属性的值，如果想匹配所有的特殊字符串，就将 android:autoLink 属性值设为 all。

下面我们来看一个例子，先在布局文件中放两个<TextView>标签，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 用于显示不同颜色、字体、大小的文字 -->
    <TextView android:id="@+id/textview1" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:padding="20dp" />
    <!-- 用于显示带 URL 地址、E-mail 地址、电话号的文本 -->
    <TextView android:id="@+id/textview2" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:textSize="20sp"
        android:autoLink="all" android:padding="2dp" />
</LinearLayout>
```

上面代码中的第二个<TextView>标签将 android:autoLink 属性值设为 all，表示匹配所有特殊的文本。下面我们用代码来为这两个 TextView 控件赋值。

```
package mobile.android.ch05.show.richtext;

import android.app.Activity;
import android.os.Bundle;
import android.text.Html;
import android.text.method.LinkMovementMethod;
import android.widget.TextView;

public class Main extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 从布局文件装载并创建两个 TextView 对象
        TextView textView1 = (TextView) findViewById(R.id.textview1);
        TextView textView2 = (TextView) findViewById(R.id.textview2);

        // 设置第一个 TextView 要显示的文本（带预定义标签的字符串）
    }
}
```

```

String html = "<font color='red'>I love Android.</font><br>";
html += "<font color='#0000FF' ><big><i>I love Android.</i></big></font><p>";
html += "<font color='@" + android.R.color.white+ "'><tt><b><big><u>I love
Android.</u></big><b></tt></font><p>";
html += "<big><a href='http://51happyblog.com'>我的网站: 51happyblog.com</a></big>";

// 将带预定义标签的字符串转换成 CharSequence 对象
CharSequence charSequence = Html.fromHtml(html);
// 为第一个 TextView 控件设置要显示的文本
textView1.setText(charSequence);
// 下面的语句非常重要，没有该语句，无法单击链接调用浏览器显示网页
textView1.setMovementMethod(LinkMovementMethod.getInstance());

// 设置第二个 TextView 控件要显示的文本
String text = "我的 URL: http://51happyblog.com\n";
text += "我的 Email: abcd@126.com\n";
text += "我的电话: +86 024-12345678";
textView2.setText(text);
textView2.setMovementMethod(LinkMovementMethod.getInstance());
}
}

```

运行程序，会显示如图 5.1 所示的效果。



▲ 图 5.1 显示不同颜色、字体、大小的文本

**注意** 在调用 `textView1.setText` 方法设置完文本后，还需要调用 `textView1.setMovementMethod` 方法设置一个 `MovementMethod` 对象。由于在本例中`<a>`标签是链接，因此，需要使用 `LinkMovementMethod.getInstance` 方法获得 `MovementMethod` 对象。该对象可以使单击链接时调用浏览器显示指定的网页。如果不设置 `MovementMethod` 对象，虽然可以正常显示`<a>`标签指定的链接，但单击链接后无任何反应。

**技巧点拨：**如何查看 Android SDK 的源代码。当我们要使用 Android SDK 中某个类时，由于官方文档和网上都没有详细的描述，或是由于其他原因，需要查看 Android SDK 中的相关源代码。

虽然可以直接下载 Android SDK 的源代码，并找到该文件进行查看，但这毕竟有些麻烦。在 Eclipse 中提供了一个功能，可以直接在 Eclipse 中查看源代码。首先按住 Ctrl 键，再用鼠标单击要查看源代码的类或接口，就可以直接跳到相应的源代码文件中（都在 Eclipse 中显示）。如果 Eclipse 未搜索到源代码，就需要指定源代码的位置。为了方便，我们可以直接将 Android SDK 的源代码放到 <Android SDK 安装目录>\platforms\android-9\sources 目录中。这样 Eclipse 就可以直接找到 Android SDK 的源代码中。

**源代码分析：**查看 TextView 中预定义的 Html 标签。Html 类及其 fromHtml 方法在官方文档中并没有详细地描述，我们可以按照前面的方法来查看 Html 的源代码。首先找到 Html.fromHtml 方法，代码如下：

```
public static Spanned fromHtml(String source, ImageGetter imageGetter,
                                TagHandler tagHandler) {
    Parser parser = new Parser();
    try {
        parser.setProperty(Parser.schemaProperty, HtmlParser.schema);
    } catch (org.xml.sax.SAXNotRecognizedException e) {
        // Should not happen.
        throw new RuntimeException(e);
    } catch (org.xml.sax.SAXNotSupportedException e) {
        // Should not happen.
        throw new RuntimeException(e);
    }
    HtmlToSpannedConverter converter =
        new HtmlToSpannedConverter(source, imageGetter, tagHandler,
                                   parser);
    return converter.convert();
}
```

上面的 fromHtml 方法是该方法的重载形式之一，也是参数最多的重载形式。现在只看第一个参数 source，其他参数将在后面详细介绍。Source 就是带标签的文本。要注意上面代码中的黑体字部分涉及一个 HtmlToSpannedConverter 类，从该类的名字看是用于转换 Html 文本的，也就是带标签的文本。我们再按住 Ctrl 键，并用鼠标单击该类，发现并不能跳转到该类。实际上，HtmlToSpannedConverter 类并没有单独定义在其他的源代码文件中，而是在 Html.java 中定义的一个非 public 的类（我们可以在 Html.java 文件中搜索得到）。找到 HtmlToSpannedConverter 类，会发现该类中有一个 handleStartTag 方法，该方法用于分析 Html 标签的开头部分（<tag>）。除了该方法外，还有另外一个 handleEndTag 方法，用于分析 Html 标签的结尾部分（</tag>）。这两个方法，只看一个即可。下面是 handleStartTag 方法的代码。

```
private void handleStartTag(String tag, Attributes attributes) {
    if (tag.equalsIgnoreCase("br")) {
        // We don't need to handle this. TagSoup will ensure that there's a </br> for each <br>
        // so we can safely emit the linebreaks when we handle the close tag.
    } else if (tag.equalsIgnoreCase("p")) {
        handleP(mSpannableStringBuilder);
    } else if (tag.equalsIgnoreCase("div")) {
        handleP(mSpannableStringBuilder);
    } else if (tag.equalsIgnoreCase("em")) {
```

```

        start(mSpannableStringBuilder, new Bold());
    } else if (tag.equalsIgnoreCase("b")) {
        start(mSpannableStringBuilder, new Bold());
    } else if (tag.equalsIgnoreCase("strong")) {
        start(mSpannableStringBuilder, new Italic());
    } else if (tag.equalsIgnoreCase("cite")) {
        start(mSpannableStringBuilder, new Italic());
    } else if (tag.equalsIgnoreCase("dfn")) {
        start(mSpannableStringBuilder, new Italic());
    } else if (tag.equalsIgnoreCase("i")) {
        start(mSpannableStringBuilder, new Italic());
    } else if (tag.equalsIgnoreCase("big")) {
        start(mSpannableStringBuilder, new Big());
    } else if (tag.equalsIgnoreCase("small")) {
        start(mSpannableStringBuilder, new Small());
    } else if (tag.equalsIgnoreCase("font")) {
        startFont(mSpannableStringBuilder, attributes);
    } else if (tag.equalsIgnoreCase("blockquote")) {
        handleP(mSpannableStringBuilder);
        start(mSpannableStringBuilder, new Blockquote());
    } else if (tag.equalsIgnoreCase("tt")) {
        start(mSpannableStringBuilder, new Monospace());
    } else if (tag.equalsIgnoreCase("a")) {
        startA(mSpannableStringBuilder, attributes);
    } else if (tag.equalsIgnoreCase("u")) {
        start(mSpannableStringBuilder, new Underline());
    } else if (tag.equalsIgnoreCase("sup")) {
        start(mSpannableStringBuilder, new Super());
    } else if (tag.equalsIgnoreCase("sub")) {
        start(mSpannableStringBuilder, new Sub());
    } else if (tag.length() == 2 &&
               Character.toLowerCase(tag.charAt(0)) == 'h' &&
               tag.charAt(1) >= '1' && tag.charAt(1) <= '6') {
        handleP(mSpannableStringBuilder);
        start(mSpannableStringBuilder, new Header(tag.charAt(1) - '1'));
    } else if (tag.equalsIgnoreCase("img")) {
        startImg(mSpannableStringBuilder, attributes, mImageGetter);
    } else if (mTagHandler != null) {
        mTagHandler.handleTag(true, tag, mSpannableStringBuilder, mReader);
    }
}
}

```

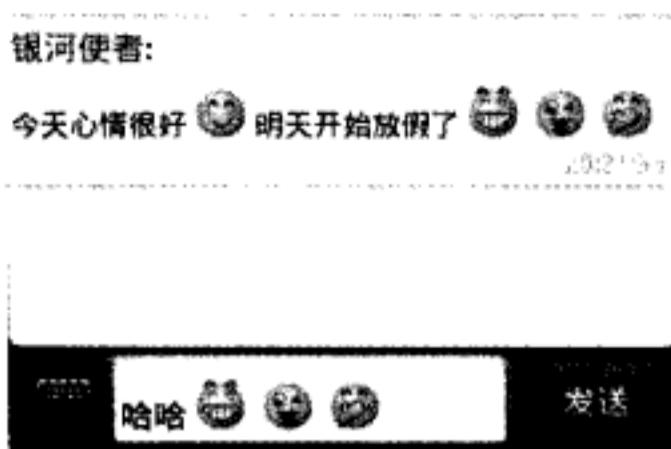
从上面的代码很容易看出，每一个 `equalsIgnoreCase` 方法的参数值就是一个 Html 标签，由于使用的是 `equalsIgnoreCase` 进行字符串比较，因此，Html 标签是不区分大小写的。每一个 if 子句中都是处理相应 Html 标签的方法。以后凡是遇到类似的问题，都可以利用查看源代码的方法来确定具体的细节。

## 5.2.2 在 TextView 中显示表情图像和文字

**工程目录:** src\ch05\ch05\_text\_image

有很多读者使用过腾讯 QQ for Android。在 QQ 的聊天界面，无论是在输入聊天内容的 EditText (将在 5.3 节详细讲解)，还是在显示聊天内容的列表 (主要使用 TextView 显示)，都可以使用图文

混排的方式显示内容，如图 5.2 所示。



▲ 图 5.2 QQ 显示聊天内容

很多初学者可能认为在 TextView 或 EditText 中同时显示图像很复杂，实现起来也很困难，其实恰好相反，向 TextView 或 EditText 中添加图像只比直接添加文本复杂一点点，而且就用 5.2.1 节介绍的<img>标签就可以很容易实现。

<img>标签只有一个 src 属性，该属性原则上应该指向一个图像地址或可以找到某个图像资源的唯一标识。但要注意的是，系统并不会直接根据 src 属性所指的值自动获得和显示图像，这一切都要我们去做。也就是说，src 属性值的含义只有设置它的开发人员才知道。解析 src 属性值的工作需要在 ImageGetter 对象的 getDrawable 方法中完成。ImageGetter 是一个接口，也许很多读者对这个接口比较陌生，但使用过 Html.fromHtml 方法的如下重载形式就会比较熟悉它。

```
| public static Spanned fromHtml(String source, ImageGetter imageGetter, TagHandler tagHandler);
```

fromHtml 方法有如下 3 个参数。

- source：包含 Html 标签（5.2.1 节介绍的标签）的字符串。
- imageGetter：ImageGetter 对象。当系统解析到<img>标签时就会调用 ImageGetter 对象的 getDrawable 方法，并将 src 属性值传入 getDrawable 方法。至于 src 属性值的具体含义，就要在 getDrawable 方法中确定了。getDrawable 方法返回一个 Drawable 对象。我们可以从 res/drawable 资源、SD 卡或网络获得图像资源，并封装成 Drawable 对象。
- tagHandler：TagHandler 对象。这个参数使用得并不多。当系统处理每一个标签时都会调用该对象的 handleTag 方法。如果不使用该参数，可将该参数值设为 null。

下面我们来看一个例子。先准备 5 个图像文件，分别是 image1.png、image2.png、image3.png、image4.png 和 image5.png，图像分辨率都是 48\*48（可以更大或更小），这 5 个图像文件都放在 res/drawable 目录中。

这个例子是在一个 TextView 控件中以不同的大小显示这 5 个图像，并在其中插入相应的文字。首先我们需要一个包含 Html 标签的文本，代码如下：

```
| String html = "图像 1<img src='image1' />图像 2<img src='image2' />图像 3<img src='image3' /><p>" +  
|     " 图像 4<a href='http://51happyblog.com'><img src='image4' /></a> 图像 5<img  
|     src='image5' />";
```

在上面代码中<img>标签的 src 属性值分别是这 5 个图像的文件名（不含扩展名），其中“图像 4”还使用了<a>标签将<img>标签括了起来，这样单击“图像 4”后面的图像，就可以直接打开浏

览器显示网页了。由于无法直接使用文件名来引用 res/drawable 中的图像资源，因此，在本例中使用了一个技巧，利用反射技术从 R.drawable 类中通过图像资源文件名获得了相应的图像资源 ID。实现的原理是 R.drawable 类中相应的资源 ID 变量名就是图像文件的文件名，所以可以利用反射技术获得 R.drawable 类中指定字段的值，实现代码如下：

```
// name 参数表示 res/drawable 中的图像文件名（不含扩展名）
public int getResourceId(String name)
{
    try
    {
        // 根据资源 ID 的变量名（也就是图像资源的文件名）获得 Field 对象
        Field field = R.drawable.class.getField(name);
        // 取得并返回资源 ID 字段（静态变量）的值
        return Integer.parseInt(field.get(null).toString());
    }
    catch (Exception e)
    {
    }
    return 0;
}
```

最后我们需要使用 Html.fromHtml 方法将含有 Html 标签的文本转换成 CharSequence 对象，代码如下：

```
TextView textView = (TextView) findViewById(R.id.textview);
// 设置 TextView 的文字颜色
textView.setTextColor(Color.BLACK);
// 设置 TextView 的背景颜色
textView.setBackgroundColor(Color.WHITE);
// 设置 TextView 的字体大小
textView.setTextSize(20);
// 使用 Html.fromHtml 方法转换包含 Html 标签的文本，需要指定 fromHtml 方法的第二个参数
CharSequence charSequence = Html.fromHtml(html, new ImageGetter()
{
    @Override
    public Drawable getDrawable(String source)
    {
        // 装载图像资源
        Drawable drawable = getResources().getDrawable(getResourceId(source));
        // 第三个图像文件按 50% 等比压缩显示 (24 * 24)
        if (source.equals("image3"))
            drawable.setBounds(0, 0, drawable.getIntrinsicWidth() / 2, drawable.getIntrinsicHeight() / 2);
        else // 其他的图像文件按原大小显示
            drawable.setBounds(0, 0, drawable.getIntrinsicWidth(), drawable.getIntrinsicHeight());
        return drawable;
    }
}, null);
textView.setText(charSequence);
```

```
// 只要使用了<a>标签，就需要设置MovementMethod 对象，否则<a>标签除了显示效果，并不起任何作用
textView.setMovementMethod(LinkMovementMethod.getInstance());
```

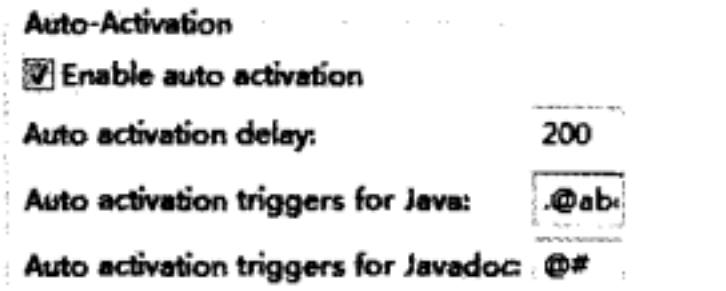
在 getDrawable 方法中使用获得图像资源的 Drawable 对象后，必须使用 Drawable.setBounds 方法设置图像的显示区域，否则显示区域的面积为 0，也就不会在 TextView 中显示图像了。通过 setBounds 方法还可以将原图像放大或缩小显示，在本例中将第 3 个图像文件等比缩小 50% 显示。本例的显示效果如图 5.3 所示。



▲ 图 5.3 图文混排的效果

### 技巧点拨：使 Eclipse 的代码编辑器变得更智能。

我们在前面已使用过很多 Android SDK 中的类或接口，但有时可能会忘记这些类或接口的全名是什么，只是记得头几个字母。虽然在 Eclipse 中输入变量、类或接口后再输入一个点（.），会自动列出所有的成员。但输入变量、类或接口时却不智能，当然，我们可以通过 Content Assist 快捷键来显示当前可能输入的内容列表，但这毕竟比较麻烦（总需要使用快捷键）。使用过 Visual Studio 高版本的读者会发现在 Visual Studio 中每输入一个字符都会自动显示提示输入的列表，幸运的是，这样的功能在 Eclipse 中也可以实现。选中“Window”>“Preferences”菜单项，打开“Preferences”对话框，在左侧的树中选中“Java”>“Editor”>“Content Assist”节点，在右侧下方会出现如图 5.4 所示的设置项。找到“Auto activation triggers for Java”输入框，在该输入框中默认只有一个点（.）（这下各位读者知道为什么输入“.”后会自动显示成员列表了吧），在“.”后面输入“@abcdefghijklmnopqrstuvwxyz”，保存设置后，在 Java 代码编辑器中输入 a 至 z 或 @（用于 Java 注释）后就会自动显示用于辅助输入的列表了。如果还想在输入其他字符时也显示辅助列表，可以在该文本框中输入这些字符。



▲ 图 5.4 设置显示辅助列表的字符

### 5.2.3 单击链接弹出 Activity

工程目录：src\ch05\ch05\_link\_activity

在5.2.1节和5.2.2节介绍了[<a>](#)标签以及TextView自动识别的特殊文本(网址、电话号码、E-mail等)，这些都可以通过单击来触发不同的动作。虽然这些单击动作已经可以满足大多数需要了，但如果读者想在单击链接时执行任意自定义的动作，那么本节的内容非看不可。

现在让我们使用5.2.1节介绍的方法重新查看Html.java文件的内容，随便找一个处理Html标签的方法，例如，endA方法，该方法用于处理[</a>](#)标签，我们会发现在该方法中如下的语句。

```
text.setSpan(new URLSpan(h.mHref), where, len, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

其中text是SpannableStringBuilder对象，该对象既可以修改文本内容，又可以将某段文本设置成一个Span，在Android中，Span表示一段文本的效果，例如，链接形式、图像、带背景色的文本等。

上面代码中使用setSpan方法将某个区间(由where和len指定的区间)的文本设置成URLSpan效果，也就是链接显示效果。其中URLSpan表示将文本设置成链接效果，该类是ClickableSpan的子类，在android.text.style包中可以找到URLSpan和ClickableSpan类。实际上，所有的Span类都在android.text.style包中。

我们再按照查看Html.java文件内容的方法查看URLSpan.java文件的内容，会看到一个onClick方法，代码如下：

```
// 覆盖ClickableSpan类中的onClick方法，onClick方法在ClickableSpan类中是抽象方法
@Override
public void onClick(View widget) {
    Uri uri = Uri.parse(getURL());
    Context context = widget.getContext();
    Intent intent = new Intent(Intent.ACTION_VIEW, uri);
    intent.putExtra(Browser.EXTRA_APPLICATION_ID, context.getPackageName());
    context.startActivity(intent);
}
```

在onClick方法中获得了[<a>](#)标签的href属性设置的URL，并调用相应的Activity来显示网页。

从onClick方法的源代码以及ClickableSpan类的名字可以得出一个结论。在5.2.1节和5.2.2节介绍的像电话、E-mail、网址、链接都是在ClickableSpan类的onClick方法中通过Action调用相应的Activity来显示不同的内容的。那么我们也可以采用类似的方法，也就是自己来实现onClick方法，这样就可以达到自定义单击动作的目的了。

说做就做，先准备两个TextView控件。在本例中我们使用SpannableString对象来设置Span，SpannableString和SpannableStringBuilder的区别是SpannableString不允许修改文本，只允许设置Span，而SpannableStringBuilder既允许修改文本，也允许设置Span。

下面的代码采用了隐式创建ClickableSpan对象实例的方法来设置Span，并在其中覆盖了onClick方法。

```
package mobile.android.ch05.link.activity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
```

## Android 开发权威指南

```

import android.text.SpannableString;
import android.text.Spanned;
import android.text.method.LinkMovementMethod;
import android.text.style.ClickableSpan;
import android.view.View;
import android.widget.TextView;

public class Main extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView textView1 = (TextView) findViewById(R.id.textview1);
        TextView textView2 = (TextView) findViewById(R.id.textview2);

        String text1 = "显示Activity1";
        String text2 = "显示Activity2";
        // 将文本转换成 SpannableString 对象
        SpannableString spannableString1 = new SpannableString(text1);
        SpannableString spannableString2 = new SpannableString(text2);
        // 将 text1 中的所有文本设置成 ClickableSpan 对象，并实现了 onClick 方法
        spannableString1.setSpan(new ClickableSpan()
        {
            // 在 onClick 方法中可以编写单击链接时要执行的动作
            @Override
            public void onClick(View widget)
            {
                Intent intent = new Intent(Main.this, Activity1.class);
                // 显示Activity1
                startActivity(intent);
            }
        }, 0, text1.length(), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
        // 将 text2 中的所有文本设置成 ClickableSpan 对象，并实现了 onClick 方法
        spannableString2.setSpan(new ClickableSpan()
        {
            // 在 onClick 方法中可以编写单击链接时要执行的动作
            @Override
            public void onClick(View widget)
            {
                Intent intent = new Intent(Main.this, Activity2.class);
                // 显示Activity2
                startActivity(intent);
            }
        }, 0, text1.length(), Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
        // 使用 SpannableString 对象设置两个 TextView 控件的内容
        textView1.setText(spannableString1);
        textView2.setText(spannableString2);
        // 在单击链接时凡是有要执行的动作，都必须设置 MovementMethod 对象
        textView1.setMovementMethod(LinkMovementMethod.getInstance());
        textView2.setMovementMethod(LinkMovementMethod.getInstance());
    }
}

```

现在我们来看一下 `setSpan` 方法，该方法有 4 个参数。第一个参数需要设置一个 `ClicableSpan` 对象，第二个和第三个参数分别表示文本中要设置成 Span 的某段文本的起始位置和终止位置的下一个字符的位置，也就是 `start` 和 `end`，最后一个参数是一个标志。在本例中设为 `Spanned.SPAN_EXCLUSIVE_EXCLUSIVE`，该标志在 `TextView` 控件中意义不大，但在 `EditText` 控件中表示在当前 Span 效果的前后输入字符时并不应用 Span 的效果。还可以设置如下几个类似的值。

- `Spanned.SPAN_EXCLUSIVE_INCLUSIVE`: 在 Span 前面输入的字符不应用 Span 的效果，在后面输入的字符应用 Span 效果。
- `Spanned.SPAN_INCLUSIVE_EXCLUSIVE`: 在 Span 前面输入的字符应用 Span 的效果，在后面输入的字符不应用 Span 效果。
- `Spanned.SPAN_INCLUSIVE_INCLUSIVE`: 在 Span 前后输入的字符都应用 Span 的效果。

本例的显示效果如图 5.5 所示。单击屏幕上的两个链接后，就会分别显示 Activity1 和 Activity2 的界面。



▲ 图 5.5 自定义单击链接的动作

#### 5.2.4 为指定文本添加背景

工程目录: `src\ch05\ch05_text_background`

从前面几节的内容可以得知设置字符串中的某个子字符串的样式（变成可单击的链接、设置字体等）需要如下几步。

- (1) 将字符串转换成 `SpannableString` 或 `SpannableStringBuilder` 对象。
- (2) 获得要设置样式的子字符串在原字符串中的开始位置和子字符串后面字符的位置，也就是 `start` 和 `end`。
- (3) 创建一个 Span 对象（所有 `android.text.style` 包中的 `XxxSpan` 类创建的对象的统称，`Xxx` 表示 `URL`、`BackgroundColor` 等类的前缀）。
- (4) 使用 `setSpan` 方法设置一个 Span 对象，也就是说，将要设置样式的子字符串转换成 Span 对象。
- (5) 用处理完的 `SpannableString` 或 `SpannableStringBuilder` 对象设置相应的控件（如 `TextView`、`EditText`、`Button` 等）。

在 Android SDK 的 `android.text.style` 包中提供了很多现成的 Span 对象，例如，`BackgroundColorSpan` 类就是一个很常用的 Span 类，该类的功能是设置指定字符串的背景色，使用方法如下：

```
TextView textview = (TextView) findViewById(R.id.textview);
String text = "<没有背景><黄色背景>";
```

```
// 第1步：将字符串转换成 SpannableString 对象
SpannableString spannableString = new SpannableString(text);
// 第2步：确定要设置的子字符串的 start 和 end
int start = 6;
int end = 12;
// 第3步：创建 BackgroundColorSpan 对象
BackgroundColorSpan backgroundColorSpan = new BackgroundColorSpan(Color.YELLOW);
// 第4步：使用 setSpan 方法将指定子字符串转换成 BackgroundColorSpan 对象
spannableString.setSpan(backgroundColorSpan, start, end, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
// 第5步：用 SpannableString 对象设置 TextView 控件
textview.setText(spannableString);
```

BackgroundColorSpan 只能设置文字的背景色，为了更加通用，我们来自己编写一个 ColorSpan 类，使其同时可以设置文字颜色和背景色（`android.text.style.ForegroundColorSpan` 类可以设置文字颜色，但并没有可同时设置背景和文字颜色的 Span 类）。在 5.2.3 节给出了一个通过继承 `ClickableSpan` 类来编写自定义 Span 类的例子，不过这个例子需要处理链接动作，所以必须要继承 `ClickableSpan` 类。而本例只要设置文字和背景颜色即可，并不需要处理任何动作，因此，只需要从 `CharacterStyle` 类继承即可。实际上，`ClickableSpan` 也是 `CharacterStyle` 的子类。可以设置文字和背景颜色的 `ColorSpan` 类的代码如下：

```
package mobile.android.ch05.text_background;

import android.text.TextPaint;
import android.text.style.CharacterStyle;

public class ColorSpan extends CharacterStyle
{
    private int mTextColor;
    private int mBackgroundColor;
    public ColorSpan(int textColor, int backgroundColor)
    {
        mTextColor = textColor;
        mBackgroundColor = backgroundColor;
    }
    // 覆盖了 CharacterStyle 类中的 updateDrawState 方法，并在该方法中设置了文字和背景颜色
    @Override
    public void updateDrawState(TextPaint tp)
    {
        tp.bgColor = mBackgroundColor;
        tp.setColor(mTextColor);
    }
}
```

在 `ColorSpan` 类中实现了 `CharacterStyle` 类的 `updateDrawState` 方法。该方法在系统开始绘制要设置样式的字符串之前调用，以便修改绘制文字的属性，例如，文字颜色、背景颜色等。其中 `TextPaint` 是 `Paint` 的子类。`Paint` 类用于描述绘制的属性，如画笔的颜色、画笔的粗细等。现在我们来同时使用 `BackgroundColorSpan` 和 `ColorSpan` 类设置文字和背景颜色，代码如下：

```
TextView textview = (TextView) findViewById(R.id.textview);
// 定义要显示的字符串
```

```

String text = "<没有背景><黄色背景>\n\n<蓝色背景, 红色文字>";
SpannableString spannableString = new SpannableString(text);
int start = 6;
int end = 12;
BackgroundColorSpan backgroundColorSpan = new BackgroundColorSpan(Color.YELLOW);
spannableString.setSpan(backgroundColorSpan, start, end, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
// <蓝色北京, 红色文字>子字符串的开始位置 (每一个 "\n" 算一个长度)
// 由于该子字符串在原字符串的最后, 因此, end 等于字符串的长度, 也就是 text.length()
start = 14;
// 创建ColorSpan 对象
ColorSpan colorSpan = new ColorSpan(Color.RED, Color.BLUE);
// 将指字文本转换成ColorSpan 对象
spannableString.setSpan(colorSpan, start, text.length(), Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
textview.setText(spannableString);

```

本例的显示效果如图 5.6 所示。



▲ 图 5.6 设置文字和背景颜色

**多学一招：**在 XML 布局文件中设置显示效果。在前面几节介绍的设置文字样式的方法基本都是通过代码实现的，只有 android:autoLink 属性值为 true 时可以识别 TextView 控件中文本的特殊内容（电话、Email、网址等）。那么如果不设置 android:autoLink 属性，而且不通过代码设置文本（但要使用 TextView.setMovementMethod 方法设置 MovementMethod 对象，否则单击链接时不会有任何动作），这就是在资源中设置标签。由于<TextView>的 android:text 属性不能设置特殊的字符，如“<”和“>”，因此，要想在 android:text 属性中设置这些特殊字符，需要使用字符串资源。

在 res/values/strings.xml 文件中加入如下的代码。

```

<string name="link_text">
    <a href="tel:12345678">打电话</a>
</string>

```

然后在<TextView>标签的 android:text 属性中引用这个字符串资源（不要设置 android:autoLink 属性），代码如下：

```

<TextView android:id="@+id/textview2" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:textSize="20sp"
    android:padding="20dp" android:text="@string/link_text" />

```

这时就会在屏幕上显示“打电话”的链接，单击该链接会弹出拨打电话的界面，并在电话输入框中显示“1-234-5678”。

**归纳总结：**前面几节介绍了使用如下 4 种方法设置文本样式。

1. 将 android:autoLink 属性值设为 true。系统会自动识别 E-mail、电话、网址等特殊文本。
2. 使用 Html 标签，例如，<font>、<img>等。不要设置 android:autoLink 属性。
3. 在 Java 代码中直接使用 Span 对象来设置文本样式。这种方法需要将文本转换成一个 SpannableString 或 SpannableStringBuilder 对象，然后在 SpannableString 或 SpannableStringBuilder 对象中使用 setSpan 方法将要设置样式的文本转换成相应的 Span 对象。
4. 在字符串资源中使用<a>标签（只支持<a>标签）设置可单击的链接。不要设置 android:audioLink 属性。

上面 4 种方法只要涉及单击动作，就必须使用 TextView.setMovementMethod 方法设置相应的 MovementMethod 对象。

### 5.2.5 带边框的 TextView

**工程目录：**src\ch05\ch05\_bordertextview

Android SDK 本身提供的 TextView 控件并不支持边框，但可以使用如下两种方法为 TextView 控件添加边框：

- 编写一个继承 TextView 类的自定义控件，并在 onDraw 事件方法中绘制边框。
- 使用 9-patch (\*.9.png) 格式的图像作为 TextView 的背景图来设置边框（这个背景图需要带一个边框）。

在 onDraw 方法中绘制边框非常容易，只需要在 TextView 控件中绘制上、下、左、右 4 个边即可。这个自定义控件类的代码如下：

```
package mobile.android.ch05.border.textview;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.widget.TextView;

public class BorderTextView extends TextView
{
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        Paint paint = new Paint();
        // 设置所绘制的边框颜色为黑色
        paint.setColor(android.graphics.Color.BLACK);
        // 绘制上边框
        canvas.drawLine(0, 0, this.getWidth() - 1, 0, paint);
        // 绘制左边框
        canvas.drawLine(0, 0, 0, this.getHeight() - 1, paint);
        // 绘制右边框
        canvas.drawLine(this.getWidth() - 1, 0, this.getWidth() - 1, this.getHeight() - 1, paint);
        // 绘制下边框
    }
}
```

```

    canvas.drawLine(0, this.getHeight() - 1, this.getWidth() - 1, this.getHeight() - 1, paint);
}
public BorderTextView(Context context, AttributeSet attrs)
{
    super(context, attrs);
}
}

```

每次装载 View 或 View 刷新时，系统就会重绘整个 View，并调用 View 的 onDraw 方法。因此，只要覆盖 onDraw 方法，就可以自己决定在 View 上绘制的内容。TextView 和上面实现的 BorderTextView 都是 View 的子类，因此，也具有这一特性。

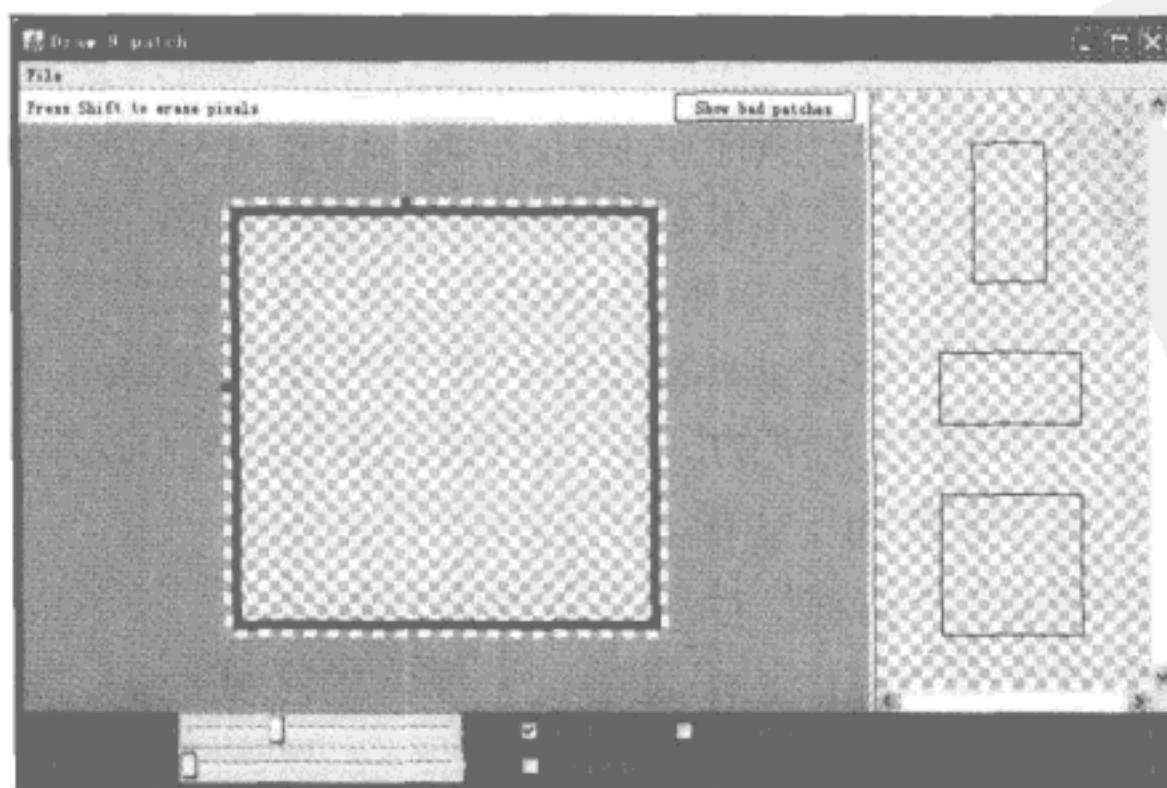
由于 BorderTextView 类在自定义的包中（mobile.android.ch05.border.textview），因此，必须在布局文件中指定 BorderTextView 的全名（packageName + className），代码如下：

```

<mobile.android.ch05.border.textview.BorderTextView
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:textColor="#000000" android:layout_marginTop="20dp"
    android:padding="10dp" android:layout_gravity="center" android:text="在画布上画边框"
/>

```

虽然可以直接使用带边框的图像来设置 TextView 控件的边框，但当 TextView 的大小变化时，背景图像上的边框也随之变粗或变细，这样看起来并不太舒服。为了解决这个问题，可以采用 9-patch 格式的图像来作为 TextView 控件的背景图。通过 Android SDK 提供的 Draw 9-patch（使用<Android SDK 安装目录>\tools\draw9patch.bat 命令来启动 Draw 9-patch）可以制作 9-patch 格式的图像。制作 9-patch 格式的图像也很简单，将事先做好的带边框的 png 图像（必须是 png 格式的图像）用这个工具打开，并在外边框的上方和左侧画一个像素点（效果如图 5.7 所示），然后保存即可。9-patch 格式的图像必须以 9.png 结尾，例如，abc.9.png。首先将制作的 9-patch 格式的图像放在 res\drawable 目录中，然后使用<TextView>标签的 android:background 属性指定相应的 9.png 格式的图像资源，9.png 图像资源和普通的图像资源的引用方法一样，只使用@drawable/border1 的格式，不带扩展名（9.png）。



▲ 图 5.7 使用 Draw 9-patch 工具制作 9-patch 格式的图像

本例的显示效果如图 5.8 所示。

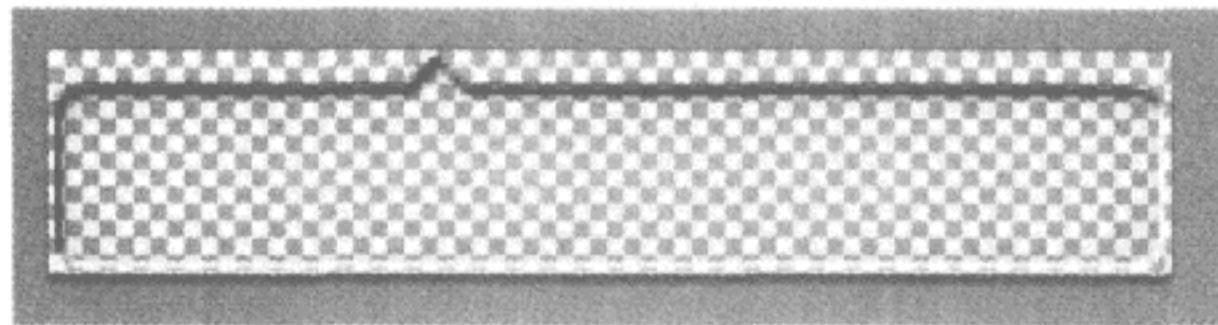


▲ 图 5.8 不同方式制作的 TextView 边框

### 扩展学习：可编程的 9-patch 格式图像。

Android 支持一种特殊的图像格式，这就是本节例子中用于为 TextView 加边框的 9-patch 格式的图像，扩展名是 9.png。实际上，这个 9-patch 图像很有意思，它是可编程的。也许有很多读者会奇怪，图像怎么还能编程？OK，现在就让我们来看个究竟。

在如图 5.8 所示的界面最后一个 TextView 带的边框上面有一个突起的小尖角。要实现这种效果，首先应使用 Photoshop 或其他图像处理工具做一个普通的透明 png 图，如图 5.9 所示。



▲ 图 5.9 透明格式的 png 图

假设我们只用如图 5.9 所示的 png 图像作为背景图，如果 TextView 控件拉伸或缩小，那么这个小尖角也同样会被拉伸或缩小。这种效果是我们所不希望看到的，最好的效果是无论 TextView 如何拉伸或缩小，小尖角永远保持如图 5.9 所示图像的大小，这就必须要使用 9-patch 格式的图像。

Android SDK 中提供了一个生成 9-patch 格式图像的工具 draw9patch。使用 draw9patch 编辑 png 图像时会在图像的四周各加一个像素宽度的区域。但要注意，这个区域并不是用于绘制边框的（边框已经绘制在 png 图像中了），而是用于“编程”的。

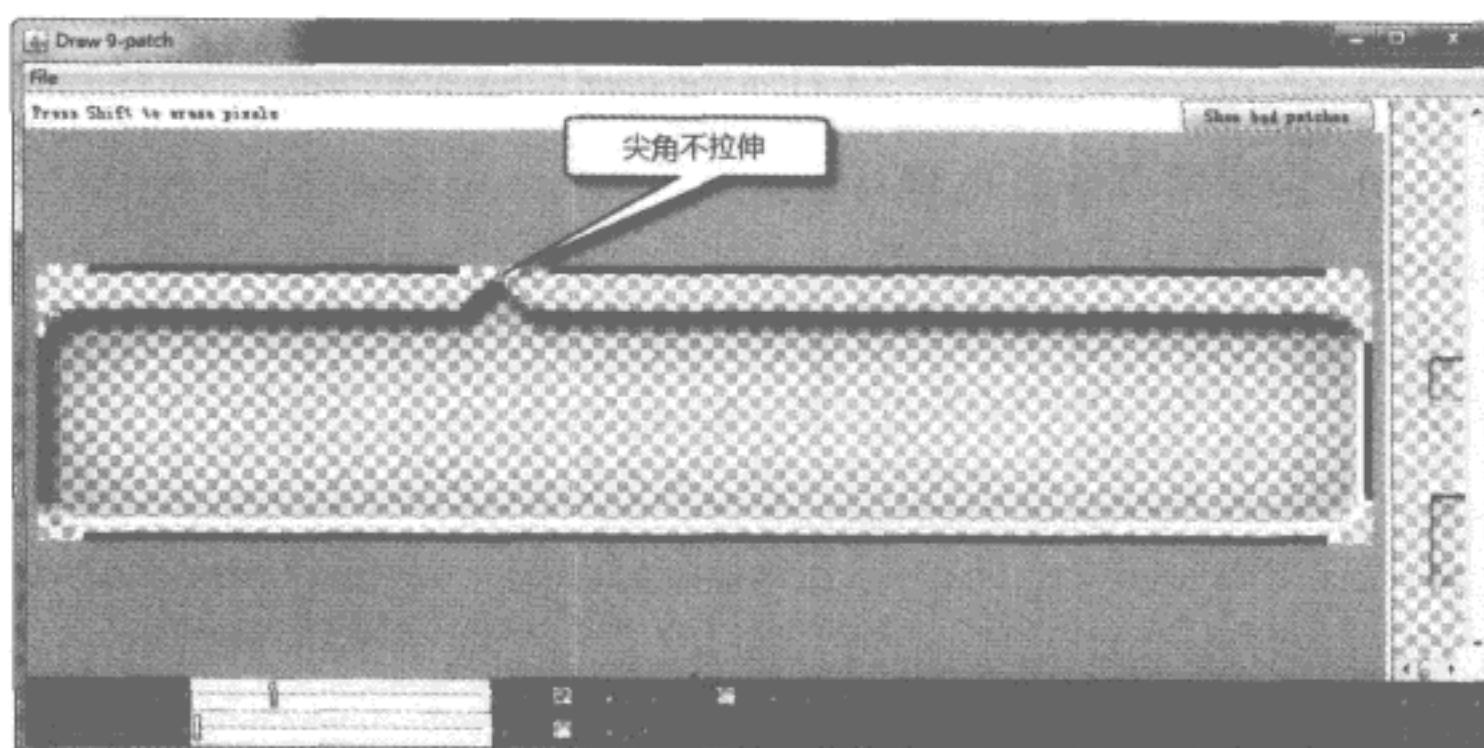
9-patch 格式的图像可以通过“编程”控制如下两个行为。

- 拉伸区域。水平拉伸区域通过上边多出的一个像素的区域控制，垂直拉伸区域通过左侧多出的一个像素的区域控制。
- 输出内容（包括文本、视图等）区域。水平输出内容的区域由下边多出的一个像素的区域



控制，垂直输出内容的区域由右侧多出的一个像素的区域控制。

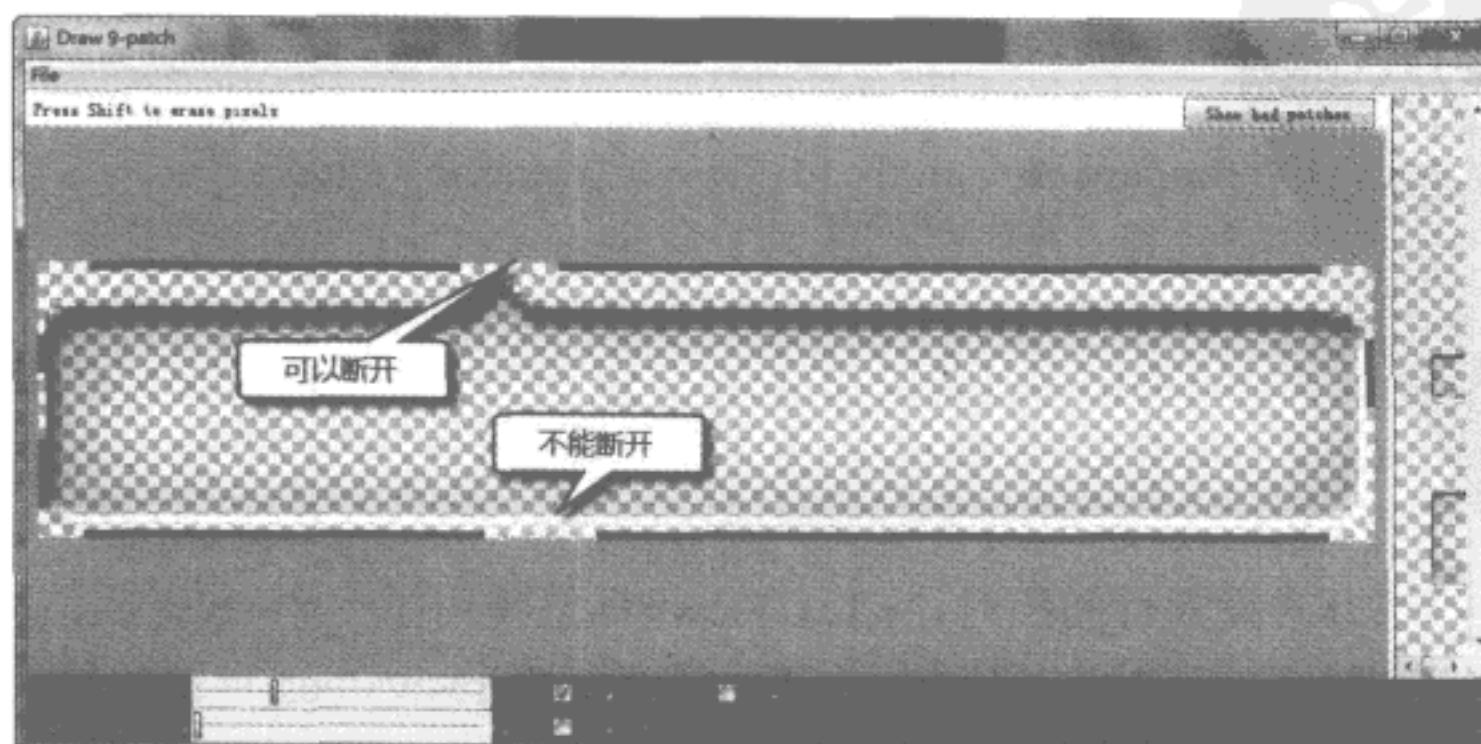
如果我们不想让如图 5.9 所示图像上方的小尖角拉伸，可以只设置小尖角两侧的区域水平拉伸。垂直方向的拉伸可以按照需要设置。例如，如果背景图有 4 个圆角，在水平和垂直方向的拉伸区域可以不包括这 4 个圆角。输出内容区域与拉伸区域类似，可根据需要设置文本输出的区域。设置方法也非常简单，只要用鼠标在图像上、下、左、右新增的一个像素宽的区域画线即可，所画的线就是相应的区域，如果要取消某处区域，按住 Shift 键，再用鼠标在相应区域再画一次即可。可按如图 5.10 所示的绘制区域将带尖角的 png 图像转换成 9-patch 格式的图像。



▲ 图 5.10 绘制 png 图像的拉伸区域和输出内容区域

如果使用如图 5.10 所示的 9-patch 图像作为视图控件的背景图，在拉伸或输出任何内容时都会在相应的区域中完成。如果原 png 图像过大或过小，还可以通过如图 5.10 所示界面下方的“Zoom”滑动杆调整图像可视区域的大小。

虽然拉伸区域和输出内容区域类似，但它们在绘制区域时有一个重要的区别，就是拉伸区域在绘制时中间可以有多处断开（如图 5.10 所示的效果中图像上方小尖角处断开了），而输出内容区域必须用连续的线段来描述区域，也就是说，中间不能有任何断裂处，如图 5.11 所示的绘制效果就会生成一个无效的 9-patch 图像。



▲ 图 5.11 生成了无效的 9-patch 图像

### 5.2.6 设置行间距

**工程目录:** src\ch05\ch05\_linespace

如果 TextView 控件中显示了多行文本，会有一个默认的行间距。但由于某些特殊的要求，需要改变默认的行间距，这就需要使用下面 3 种方法中的一种或几种来达到目的。

- 在布局文件中使用 android:lineSpacingExtra 或 android:lineSpacingMultiplier 属性设置行间距。其中 android:lineSpacingExtra 设置精确的行间距，例如，android:lineSpacingExtra="20dp"。 android:lineSpacingMultiplier 属性设置默认行间距的倍数。如果同时设置这两个属性，以较大行间距为准。
- 使用 Style 资源设置行间距，实际上这种方法与第一种方法类似，只是如果有多个控件需要设置行间距，使用 Style 会非常方便，也容易维护。
- 使用 setLineSpacing 方法设置行间距。

下面我们看一个完整的例子，这个例子演示了如何使用上面 3 种方法来分别设置 4 个 TextView 控件中文本的行间距。首先在布局文件中放 4 个<TextView>标签，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 使用 android:lineSpacingExtra 属性设置行间距 -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="#FFF"
        android:textColor="#000" android:text="第一行的文本\n第二行的文本（行间距为 20dp）"
        android:lineSpacingExtra="20dp" android:layout_margin="10dp" />
    <!-- 使用 android:lineSpacingMultiplier 属性设置行间距 -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="#FFF"
        android:textColor="#000" android:text="第一行的文本\n第二行的文本（行间距是默认行间距的
        1.8 倍）"
        android:lineSpacingMultiplier="1.8" android:layout_margin="10dp" />
    <!-- 使用 Style 设置行间距 -->
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="#FFF"
        android:textColor="#000" android:text="第一行的文本\n第二行的文本（行间距是标准行间距的
        1.5 倍）"
        style="@style/line_space" android:layout_margin="10dp" />
    <!-- 使用 setLineSpacing 方法设置行间距 -->
    <TextView android:id="@+id/textview" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="#FFF"
        android:textColor="#000" android:text="第一行的文本\n第二行的文本（用代码设置行间距）"
        android:layout_margin="10dp" />
</LinearLayout>
```

在使用 Style 资源方式设置行间距时，<TextView> 标签使用了一个 style 属性（没有 android 命名空间），该属性值指定了一个 Style 资源 ID。该资源必须在 res\values 目录中的文件中定义（可以是任何 XML 文件，在本例中直接在 strings.xml 文件定义了这个 Style），代码如下：

```
<style name="line_space">
    <item name="android:lineSpacingMultiplier">1.5</item>
</style>
```

从上面的代码可以看出，在 Style 中也是通过 android:lineSpacingMultiplier 属性设置的行间距，只是将设置的代码封装在了 Style 中。这样如果有多个控件需要设置行间距，可以直接使用 style 属性引用这个 Style，当要修改行间距时，直接修改这个 Style 即可。这相当于将某段常用的代码封装在一个方法中，在其他代码中多次调用该方法。因此笔者建议，如果需要设置行间距的控件很多时，建议使用 Style 进行设置。

下面来看看如何使用 Java 代码来设置行间距，代码如下：

```
textView.setLineSpacing(50, 1.2f);
```

setLineSpacing 方法有两个参数，都是 float 类型。第一个参数相当于 android:lineSpacingExtra 属性，第二个参数相当于 android:lineSpacingMultiplier 属性。至于系统会采用哪个参数值作为最终的行间距，要看哪个参数值所表示的行间距大了。

本例的显示效果如图 5.12 所示。



▲ 图 5.12 设置行间距

### 5.2.7 在未显示完的文本后面加省略号（...）

工程目录：src\ch05\ch05\_ignore\_text

当文本内容太多时，TextView 控件一屏无法显示完整，对于这种情况的一种处理方法就是在文本的最后显示一个省略号（...）。当然，这个省略号可以由我们来添加，但问题是在原字符串的哪里截断，并且最后添加省略号的过程是非常复杂的。不过不要担心，TextView 控件为我们提供了自动添加省略号的功能。

在默认的情况下，如果显示的文本太长，TextView 控件会在文本的适当位置截断，并在最后

加上省略号。更妙的是 TextView 控件不仅可以在最后加上省略号，还可以在最前面和中间加上省略号，甚至可以不显示省略号。下面的 XML 布局文件中有 3 个<TextView>标签，分别在 TextView 控件的前面、中间和后面加省略号。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 在最前面加省略号 -->
    <TextView android:id="@+id/textview1" android:layout_width="150dp"
        android:layout_height="wrap_content" android:text="维基百科的目标是建立拥有人类全部知识
        的百科全书。"
        android:singleLine="false" android:maxLines="2" android:ellipsize="start"
        android:background="#FFF" android:textColor="#000" android:textSize="20dp"
        android:layout_margin="10dp" />
    <!-- 在中间加省略号 -->
    <TextView android:id="@+id/textview2" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="维基百科的目标是建立拥有人类全部知识的
        百科全书。"
        android:singleLine="true" android:ellipsize="middle"
        android:background="#FFF" android:textColor="#000" android:textSize="20dp"
        android:layout_margin="10dp" android:padding="10dp" />
    <!-- 在最后面加省略号 -->
    <TextView android:id="@+id/textview3" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="维基百科的目标是建立拥有人类全部知
        识的百科全书。"
        android:singleLine="true" android:ellipsize="end" android:background="#FFF"
        android:textColor="#000" android:textSize="20dp"
        android:layout_margin="10dp" android:padding="10dp" />
</LinearLayout>
```

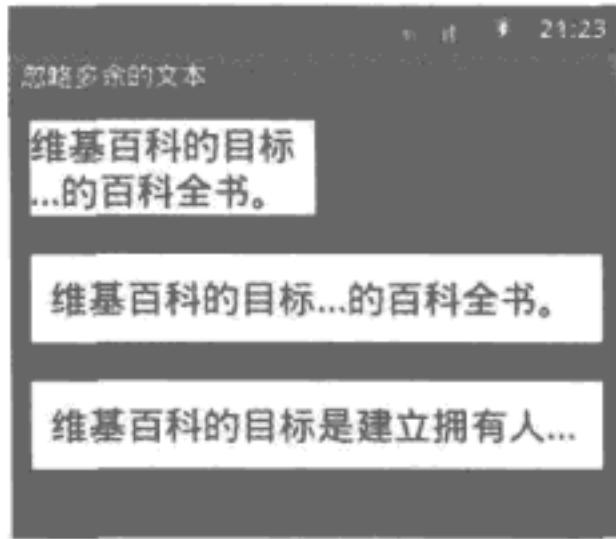
最后一个<TextView>标签的 android:ellipsize 属性可以省略（默认就在最后加省略号）。 android:ellipsize 属性值为 none 则不显示省略号。 android:ellipsize 属性也可以通过如下的 Java 代码设置。

```
TextView textView1 = (TextView) findViewById(R.id.textview1);
textView1.setEllipsize(TextUtils.TruncateAt.END);
```

其中 setEllipsize 方法的参数值是如下的枚举类型。

```
public enum TruncateAt
{
    START,
    MIDDLE,
    END,
    MARQUEE,
}
```

本例的显示效果如图 5.13 所示。



▲图 5.13 在 TextView 控件的开始、中间和结尾加省略号

### 5.2.8 用 TextView 实现走马灯效果

工程目录: src\ch05\ch05\_lamp

对于长文本的显示,除了上一节介绍的显示省略号的方法,还有一种显示方法就是水平滚动(可以滚动指定次数或无限次滚动),这种效果很像走马灯,因此也称为走马灯效果。

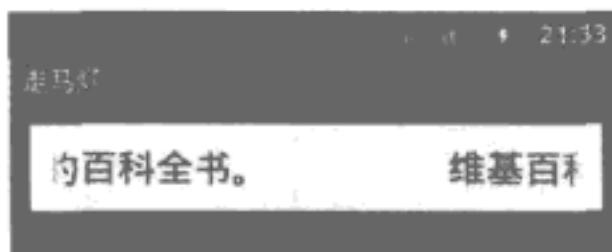
在 TextView 控件中以走马灯效果显示长文本除了要设置 android:ellipsize 属性外,还要设置 android:marqueeRepeatLimit(循环显示的次数)和 android:focusable 属性。设置 android:focusable 属性的目的是使 TextView 控件处于焦点状态,也就是说,只有处于焦点状态的 TextView 控件才会有走马灯效果。这 3 个属性需要设置如下的值。

- android:ellipsize="marquee"。
- android:marqueeRepeatLimit="marquee\_forever"或大于 0 的整数,例如, android:marqueeRepeatLimit="6", 其中 marquee\_forever 表示永远循环显示。
- android:focusable="true"。

下面的 XML 布局文件中放置了一个具有走马灯效果的 TextView 控件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/textview3" android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="维基百科的目标是建立拥有人类全部知识的百科全书。"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:marqueeRepeatLimit="marquee_forever"
        android:focusable="true"
        android:background="#FFF"
        android:textColor="#000" android:textSize="20dp"
        android:layout_margin="10dp" android:padding="10dp" />
</LinearLayout>
```

本例的显示效果如图 5.14 所示。



▲图 5.14 走马灯效果

### 5.2.9 垂直滚动 TextView 中的文本

工程目录: src\ch05\ch05\_vertical\_scroll\_text

这个世界上方法总是比问题多。前面介绍了显示大文本可以采用省略号或走马灯的方式，那么在这一节将向读者展示另外一种更完美的显示大文本的方法：在垂直方向滚动文本。

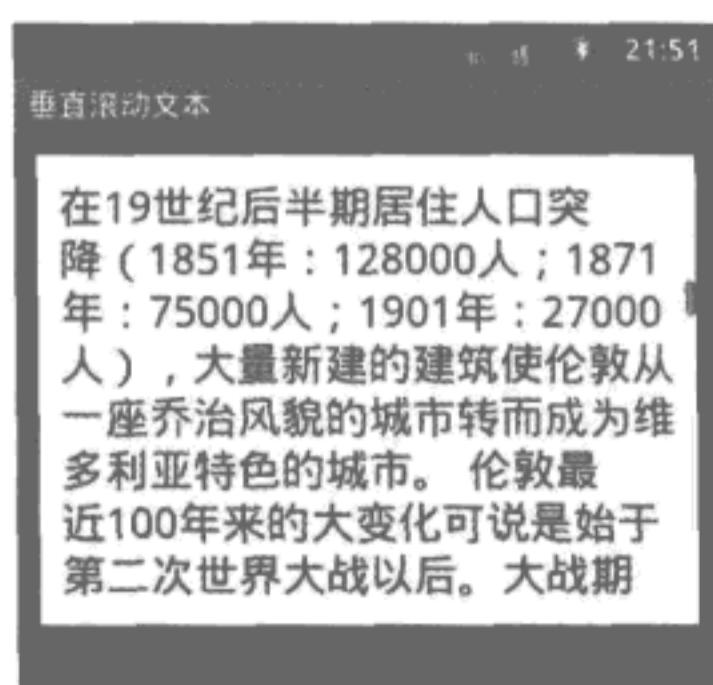
首先在 res\values\strings.xml 文件中使用<string>标签加入一个长一点的字符串资源，资源名是 big\_text，然后在布局文件中使用如下的代码来设置 TextView 控件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/textview" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:singleLine="false"
        android:maxLines="8" android:background="#FFF" android:textColor="#000"
        android:textSize="20dp" android:layout_margin="10dp"
        android:scrollbars="vertical"
        android:text="@string/big_text"
        android:scrollbarStyle="outsideOverlay"
        android:scrollbarFadeDuration="2000"
        android:padding="10dp"
    />
</LinearLayout>
```

上面的代码中与垂直滚动直接相关的有如下 4 个属性。

- android:scrollbars: 该属性的值必须设为 vertical。
- android:text: 只有比较长的文本（TextView 控件一屏无法显示完整的字符串）才可以滚动。
- android:scrollbarStyle: 该属性表示滚动条的位置。如果设为 outsideOverlay，滚动条会在文字的右侧显示；如果设为 insideOverlay，滚动条会在右侧文字上显示，也就是说，会覆盖文字的一部分（由于滚动条很窄，并没有覆盖多少）。
- android:scrollbarFadeDuration: 该属性表示滚动条从出现到消失（以渐变方式逐渐消失）的时间，单位是毫秒。在本例中设为出现滚动条 2 秒后消失。

出现滚动条的效果如图 5.15 所示。



▲ 图 5.15 垂直滚动文本

## 5.3 EditText（编辑文本的控件）

EditText 是除了 TextView 控件外的另一个非常重要的控件，EditText 类同时也是 TextView 的子类，因此，EditText 拥有 TextView 的一切 XML 属性及方法。EditText 与 TextView 的区别是 EditText 可以输入文本，而 TextView 一般只用于显示文本。TextView 虽然也可以输入文本，但并不具备输入文本所必须的功能，也就是说，用户体验并不好，因此并不建议将 TextView 直接作为文本编辑控件来使用。

在前面的章节中已经多次使用到了 EditText 控件，读者也已经了解了 EditText 控件的基本使用方法，现在再回顾一下 EditText 控件在布局文本中的使用方法，代码如下：

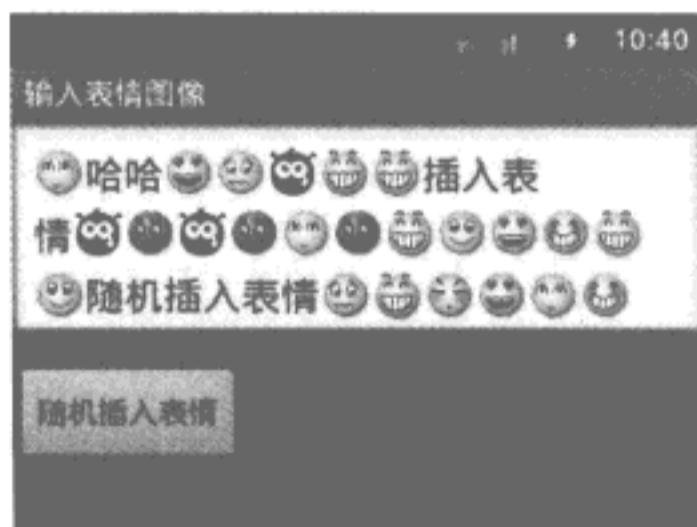
```
<EditText android:layout_width="wrap_content"
          android:layout_height="wrap_content" android:text="输入文本的组件"
          android:textColor="#000000" android:background="#FFFFFF"
          android:padding="20dp" android:layout_margin="10dp" />
```

从上面的代码可以看出，EditText 和 TextView 的使用方法完全一样，只需要将<TextView>换成<EditText>即可，几乎不需要做任何修改。当然，EditText 的功能还远不止输入文本这么简单，在本节将对 EditText 的一些高级功能进行详细讲解。

### 5.3.1 像 QQ 一样输入表情图像

工程目录：src\ch05\ch05\_edittext\_face

EditText 和 TextView 一样，也可以进行图文混排。所不同的是，TextView 只用于显示图文混排效果，而 EditText 不仅可显示，也可混合输入文字和图像，让我们先回顾一下如图 5.2 所示的 QQ 聊天输入框，在输入框中可以同时输入文字和表情图像。实际上，这种效果在 Android SDK 中只需要几行代码就可以实现。为了使读者更有学习的冲动，先来欣赏一下即将实现的效果，如图 5.16 所示。



▲ 图 5.16 在 EditText 控件中输入文字和图像

为了实现这个程序，首先来准备一些要用到的素材，也就是要在 EditText 控件中输入的图像文件。本例准备了 9 个 png 图像文件（face1.png 至 face9.png），都放在了 res\drawable 目录中。

接下来在屏幕上放一个只能显示 3 行（可输入多行）的 EditText 和一个 Button，布局文件的代码如下：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:id="@+id/edittext" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:lines="3" android:gravity="left|top"/>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="随机插入表情"
        android:onClick="onClick_RandomFace" android:layout_marginTop="10dp" />
</LinearLayout>

```

上面代码的<EditText>标签中将 android:gravity 属性值设为 left|top，以使输入的文本从左上角开始显示。如果不设置该属性，则输入的文本会从左侧中心位置开始显示（由于设置了 android:line="3"，因此，EditText 可同时显示三行的内容，所以会存在这个问题，如果只显示一行，则不存在这个问题）。

<Button>标签的 android:onClick 属性值指定了单击事件方法（onClick\_RandomFace），在该方法中随机获得了 face1.png 至 face9.png 中任意一个图像资源的 ID。最常用的方法是将这 9 个图像资源 ID 放到数组中，然后随机产生一个数组索引来获取相应的图像资源 ID。但本例未采用这种方法，而是采用了直接通过反射技术从 R.drawable 类中获得图像资源 ID 的方法。这种方法的好处是一旦图像资源非常多时，可以不需要在数组中逐个定义就可以获得任意的图像资源 ID。

在 5.2.2 节使用了<img>标签来插入图像，虽然在 EditText 控件中插入图像也可采用这种方法，但本例使用了另外一种更简单的方法，就是使用 android.text.style.ImageSpan 类来直接插入图像。下面来看看具体的实现代码。

```

public void onClick_RandomFace(View view)
{
    // 随机产生 1 至 9 的整数
    int randomId = 1 + new Random().nextInt(9);
    try
    {
        // 根据随机产生的 1 至 9 的整数从 R.drawable 类中获得相应资源 ID (静态变量) 的 Field 对象
        Field field = R.drawable.class.getDeclaredField("face" + randomId);
        // 获得资源 ID 的值，也就是静态变量的值
        int resourceId = Integer.parseInt(field.get(null).toString());
        // 根据资源 ID 获得资源图像的 Bitmap 对象
        Bitmap bitmap = BitmapFactory.decodeResource(getResources(), resourceId);
        // 根据 Bitmap 对象创建 ImageSpan 对象
        ImageSpan imageSpan = new ImageSpan(this, bitmap);
        // 创建一个 SpannableString 对象，以便插入用 ImageSpan 对象封装的图像
        SpannableString spannableString = new SpannableString("face");
        // 用 ImageSpan 对象替换 face
        spannableString.setSpan(imageSpan, 0, 4, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
        // 将随机获得的图像追加到 EditText 控件的最后
        edittext.append(spannableString);
    }
    catch (Exception e)
    {
    }
}

```

编写上面代码需要注意如下几点。

- 由于 R.drawable 中的资源 ID 都是 public 的静态变量，因此，可直接使用 Field.get 方法获得这些变量的值。如果是 private 或 protected 的变量，需要 field.setAccessible(true) 设置变量值的访问权限才可以读写这些变量。
- 使用 Field.get 方法获得变量值时，如果是静态变量，Field.get 方法的参数值设为 null 即可。如果不是静态变量，需要为 Field.get 方法指定一个变量所在类的对象作为参数值。
- 由于 EditText 类不能直接插入 Span 对象，因此，需要先使用 SpannableString 对象来封装 Span 对象（如本例中的 ImageSpan 对象），再将 SpannableString 对象插入到 EditText 控件中。

### 5.3.2 在 EditText 中输入特定的字符

**工程目录：src\ch05\ch05\_edittext\_special\_character**

EditText 控件可以通过多种方式指定允许输入的字符，例如，如果只想输入数字（0~9），可以使用如下 3 种方法：

- 将<EditText>标签的 android:digits 属性值设为 0123456789。
- 将<EditText>标签的 android:numeric 属性值设为 integer。
- 将<EditText>标签的 android:inputType 属性值设为 number。

本例将分别使用上面所述的 3 个属性来限制 EditText 控件的输入字符，XML 布局文件的代码如下：

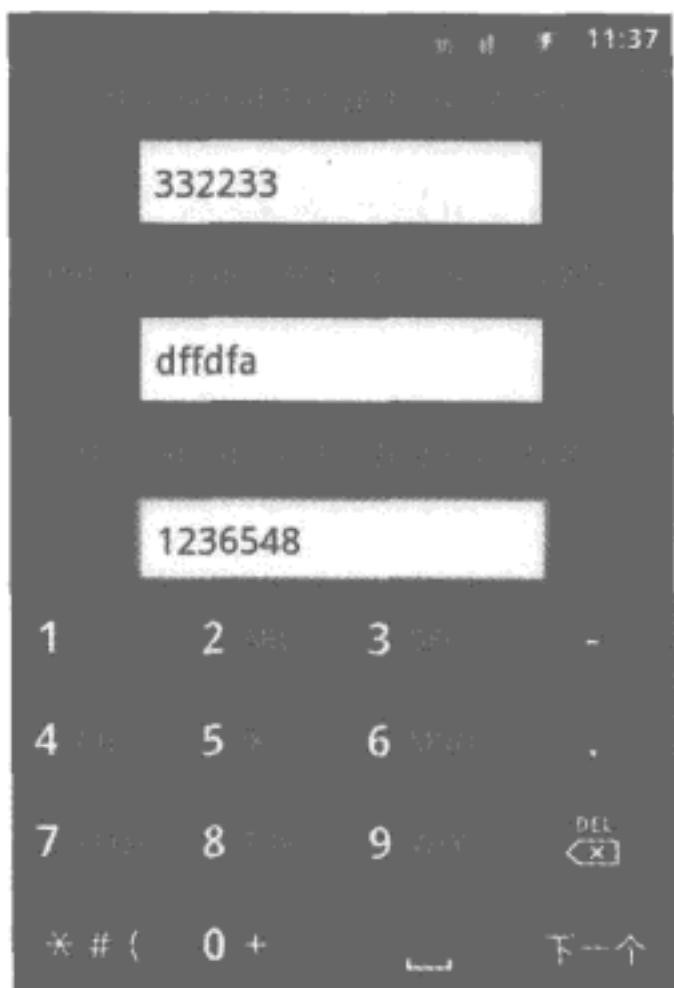
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="center_horizontal">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="使用 android:digits 属性 ( 输入数
        字 ) " />
    <EditText android:layout_width="200dp" android:layout_height="wrap_content"
        android:layout_margin="10dp" android:digits="0123456789" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="使用 android:digits 属性 ( 输入 26
        个小写字母 ) " />
    <EditText android:layout_width="200dp" android:layout_height="wrap_content"
        android:layout_margin="10dp" android:digits="abcdefghijklmnopqrstuvwxyz" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="使用 android:inputType 属性 ( 输
        入数字 ) " />
    <EditText android:layout_width="200dp" android:layout_height="wrap_content"
        android:layout_margin="10dp" android:inputType="number|textCapCharacters" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="使用 android:inputType 属性 ( 输
        入 E-mail ) " />
    <EditText android:layout_width="200dp" android:layout_height="wrap_content"
        android:layout_margin="10dp" android:inputType="textEmailAddress" />
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="使用 android:numeric 属性 ( 输入
```

```

        有符号的浮点数" />
<EditText android:layout_width="200dp" android:layout_height="wrap_content"
          android:layout_margin="10dp" android:numeric="decimal|signed"/>
</LinearLayout>

```

如果使用 `android:inputType` 属性设置允许输入的字符，当焦点落在该 `EditText` 控件上，显示的虚拟键盘会随着 `inputType` 属性值的不同而不同，例如，图 5.17 是用于输入数字的虚拟键盘，图 5.18 是用于输入 E-mail 的虚拟键盘。要注意的是，用于输入 E-mail 的 `EditText` 控件并不会限制输入非 E-mail 的字符，只是在虚拟键盘上多了一个“@”键而已。关于 `android:inputType` 和 `android:numeric` 属性的其他可选值，读者可以参阅官方的文档。



▲ 图 5.17 输入数字的虚拟键盘



▲ 图 5.18 输入 E-mail 的虚拟键盘

**多学一招：**细心的读者会发现，在如图 5.17 所示界面的后面几个 `EditText` 控件中输入文本时会弹出系统软键盘，并且整个界面会向上移，以便显示出当前正处于焦点的 `EditText` 控件。如果出于某种需要，不想让界面向上移，可以使用 `getWindow().setSoftInputMode` 方法将输入法模式设为 `WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE`，代码如下：

```
getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE);
```

### 5.3.3 AutoCompleteTextView（自动完成输入内容的控件）

工程目录：src\ch05\ch05\_AutoCompleteTextView

也许有的读者会想，有没有搞错，`AutoCompleteTextView` 看样子像是 `TextView` 的“近亲”，怎么放到 `EditText` 部分了。要是这么想，大家可被 `AutoCompleteTextView` 的名字给唬住了。实际上，`AutoCompleteTextView` 是对 `EditText` 的扩展（`EditText` 是 `AutoCompleteTextView` 的直接父类），也属于文本编辑控件，只是该控件可以像 Google 搜索框一样在编辑框下方列出可供选择的列表。由此看来，`AutoCompleteTextView` 改成 `AutoCompleteEditText` 更合适。

AutoCompleteTextView 控件在布局文件中使用<AutoCompleteTextView>标签来表示，该标签的使用方法与<EditText>标签相同。如果要让 AutoCompleteTextView 控件显示辅助输入列表，需要使用 AutoCompleteTextView 类的 setAdapter 方法指定一个 Adapter 对象，代码如下：

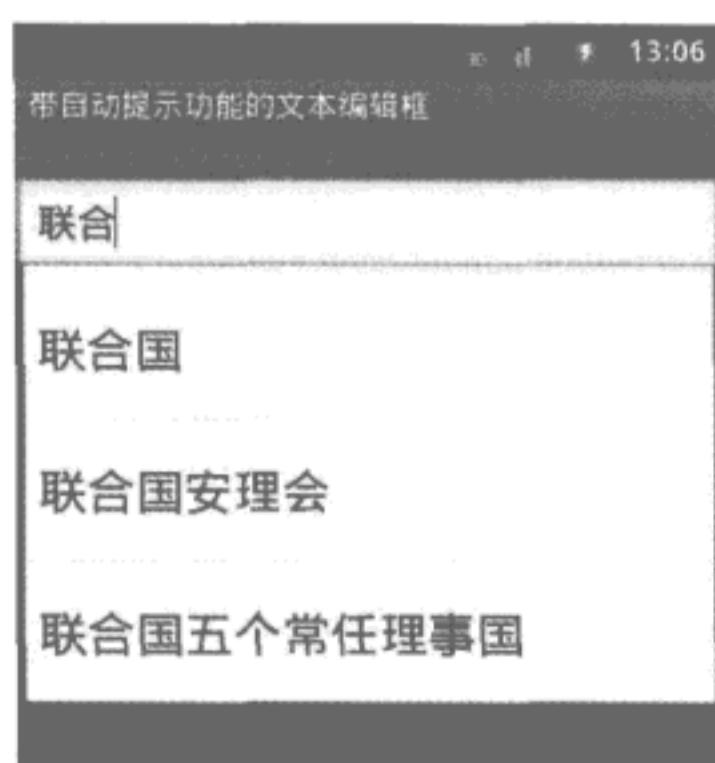
```
String[] autoString = new String[]
    { "联合国", "联合国安理会", "联合国五个常任理事国", "bb", "bcd", "bcdf", "Google", "Google
      Map",
      "Google Android" };
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_dropdown_item_1line, autoString);
AutoCompleteTextView autoCompleteTextView =
    (AutoCompleteTextView) findViewById(R.id.autoCompleteTextView);
autoCompleteTextView.setAdapter(adapter);
```

运行上面代码后，在文本框中输入“联合”，在编辑框下方会出现一个如图 5.19 所示的辅助输入列表，该列表会将 Adapter 指定的所有以“联合”开头的字符串列出来。

除了 AutoCompleteTextView 控件外，还可以使用 MultiAutoCompleteTextView 控件来完成连续输入的功能。也就是说，当输入完一个字符串后，在该字符串后面输入一个逗号（,），在逗号前后可以有任意多个空格，然后再输入一个字符串（例如“goo”），仍然会显示辅助输入的列表，但要使用 MultiAutoCompleteTextView 类的 setTokenizer 方法指定 MultiAutoCompleteTextView.CommaTokenizer 类的对象实例（该对象表示输入多个字符串时的分隔符为逗号），代码如下：

```
MultiAutoCompleteTextView multiAutoCompleteTextView =
    (MultiAutoCompleteTextView) findViewById(R.id.multiAutoCompleteTextView);
multiAutoCompleteTextView.setAdapter(adapter);
multiAutoCompleteTextView.setTokenizer(new
    MultiAutoCompleteTextView.CommaTokenizer());
```

运行上面的代码后，在屏幕的第 2 个编辑框中输入“联合国,” 后，再输入“goo”，会显示如图 5.20 所示的效果。



▲ 图 5.19 输入“联合”后显示的辅助输入列表



▲ 图 5.20 输入“联合国,goo”后显示的辅助输入列表

## 5.4 按钮和复选框控件

本节将介绍 Android SDK 中的按钮和复选框控件。按钮可分为多种，例如，普通按钮（Button）、图像按钮（ImageButton）、选项按钮（RadioButton）等。除此之外，复选框（CheckBox）也是 Android SDK 中非常重要的控件，通常用于多选的应用中。

### 5.4.1 Button（普通按钮控件）

工程目录：src\ch05\ch05\_button

Button 控件在前面的章节已经多次使用到了。Button 控件的基本使用方法与 TextView、EditText 并无太大的差异，例如，下面的代码在布局文件中配置了一个按钮。

```
<Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="我的按钮 1" />
```

最常用的按钮事件是单击事件，可以通过 Button 类的 setOnClickListener 方法设置处理单击事件的对象实例，也可以直接通过 android:onClick 属性指定单击事件方法。如果当前的类实现了 android.view.View.OnClickListener 接口，可以直接将 this 传入 setOnClickListener 方法，代码如下：

```
Button button1 = (Button) findViewById(R.id.button1);
button1.setOnClickListener(this);
```

在本节的例子中包含两个按钮，并在单击事件中通过 value 变量控制按钮放大或缩小（value=1 为放大，value=-1 为缩小），代码如下：

```
private int value = 1;
@Override
public void onClick(View view)
{
    Button button = (Button) view;
    // 如果按钮宽度等于屏幕宽度，按钮开始缩小
    if (value == 1 && button.getWidth() == getWindowManager().getDefaultDisplay().getWidth())
        value = -1;
    // 如果按钮宽度小于 100，按钮开始放大
    else if (value == -1 && button.getWidth() < 100)
        value = 1;
    // 以按钮宽度和高度的 10% 放大或缩小按钮
    button.setWidth(button.getWidth() + (int) (button.getWidth() * 0.1) * value);
    button.setHeight(button.getHeight() + (int) (button.getHeight() * 0.1) * value);
}
```

运行上面的代码后，将显示两个按钮，单击任何一个按钮后，该按钮都会放大，当按钮宽度等于屏幕宽度时，再次单击按钮时，按钮开始缩小。

为了使按钮更绚丽，还可以为按钮加上背景图。通过设置透明背景图，可以做出任何形状的按钮，这种按钮也可以称为“异形按钮”。异形按钮需要处理 3 个事件，这 3 个事件及各自的处理逻

辑如下。

- 触摸事件 (onTouch): 当触摸按钮时, 应该显示按钮被触摸后的状态。
- 焦点变化事件 (onFocusChange): 当焦点从一个按钮切换到另一个按钮时, 应该显示按钮被触摸时的状态 (image2.png), 并且将上一个焦点按钮设为未被触摸时的状态。
- 键盘事件 (onKey): 当某一个按钮获得焦点后, 按下手机或模拟器上的“确认”按钮后, 当前按钮应该被置成被按下的状态, 也就是这个按钮的第3张图, 因此, 需要为每一个按钮准备3张图, 分别是正常状态、触摸状态、按键被按下的状态。

现在我们来将本例的第二个按钮改成异形按钮, 所以需要准备按钮的3个状态的png图 (button1.png、button2.png、button3.png)。

先来编写触摸事件 (onTouch) 中的代码。

```
@Override
public boolean onTouch(View view, MotionEvent event)
{
    if (event.getAction() == MotionEvent.ACTION_UP)
    {
        // 当手指或鼠标抬起时恢复按钮的默认状态
        view.setBackgroundResource(R.drawable.button1);
    }
    else if (event.getAction() == MotionEvent.ACTION_DOWN)
    {
        // 当手指或鼠标按下时将按钮置为按下状态
        view.setBackgroundResource(R.drawable.button2);
    }
    return false;
}
```

在上面的代码中根据 `getAction` 方法获得了触摸的状态, 并使用 `setBackgroundColor` 方法为按钮设置了不同的背景图像。

当异形按钮处于焦点状态时, 需要将异形按钮的背景图设为被触摸状态, 代码如下:

```
// 当控件焦点变化时调用该事件方法, 当控件获得焦点时, hasFocus参数值为true, 否则为false
@Override
public void onFocusChange(View view, boolean hasFocus)
{
    if (hasFocus)
    {
        // 设置异形按钮获得焦点时的背景图
        imageButton.setBackgroundResource(R.drawable.button2);
    }
    else
    {
        // 设置异形按钮失去焦点时的背景图
        imageButton.setBackgroundResource(R.drawable.button1);
    }
}
```

当按手机或模拟器上的“确认”键时, 需要将当前获得焦点的异形按钮设为按键被按下的状态,

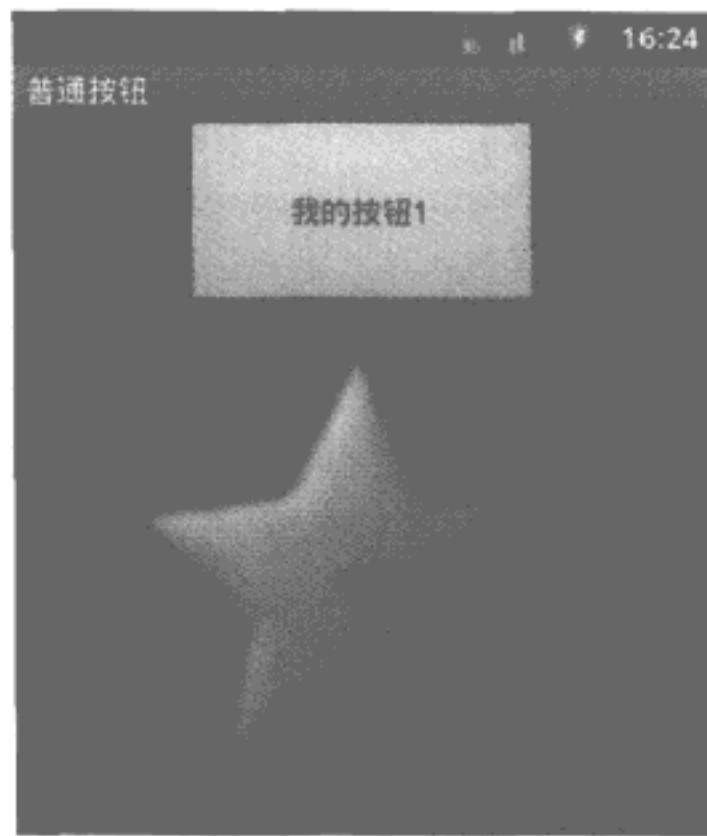
当松开“确认”键时，又恢复到获得焦点的状态，这个功能需要在处理键盘事件的 `onKey` 中完成，代码如下：

```
@Override
public boolean onKey(View view, int keyCode, KeyEvent event)
{
    if (KeyEvent.ACTION_DOWN == event.getAction())
    {
        // 设置异形按钮被按下时的背景图
        view.setBackgroundResource(R.drawable.button3);
    }
    else if (KeyEvent.ACTION_UP == event.getAction())
    {
        // 设置异形按钮被抬起时的背景图（焦点状态的背景图）
        view.setBackgroundResource(R.drawable.button2);
    }
    return false;
}
```

本例中两个按钮在默认情况下的显示效果如图 5.21 所示。当不断按屏幕上的两个按钮时，两个按钮会不断放大，当遇到屏幕边界时，会再缩小，如图 5.22 所示。



▲ 图 5.21 默认状态的两个按钮



▲ 图 5.22 单击放大的两个按钮

#### 5.4.2 图文混排的按钮

工程目录：src\ch05\ch05\_imagetextbutton

上一节介绍的只是在按钮上显示文字或图像，在本节将介绍如下两种同时在按钮上显示文字和图像的方法。

- 使用`<Button>`标签的 `android:drawableXxx` 属性，其中 `Xxx` 表示 `Top`、`Bottom`、`Left`、`Right`。这 4 个属性都是资源类型，需要指定图像资源的 ID，分别表示在按钮上、下、左、右插入一个图像。同时还可配合 `android:drawablePadding` 属性来设置图像到文字的距离。
- `Button` 和 `EditText` 一样，也是 `TextView` 的子类，因此，也可以采用与 `TextView`、`EditText`



同样的方法来实现图文混排。

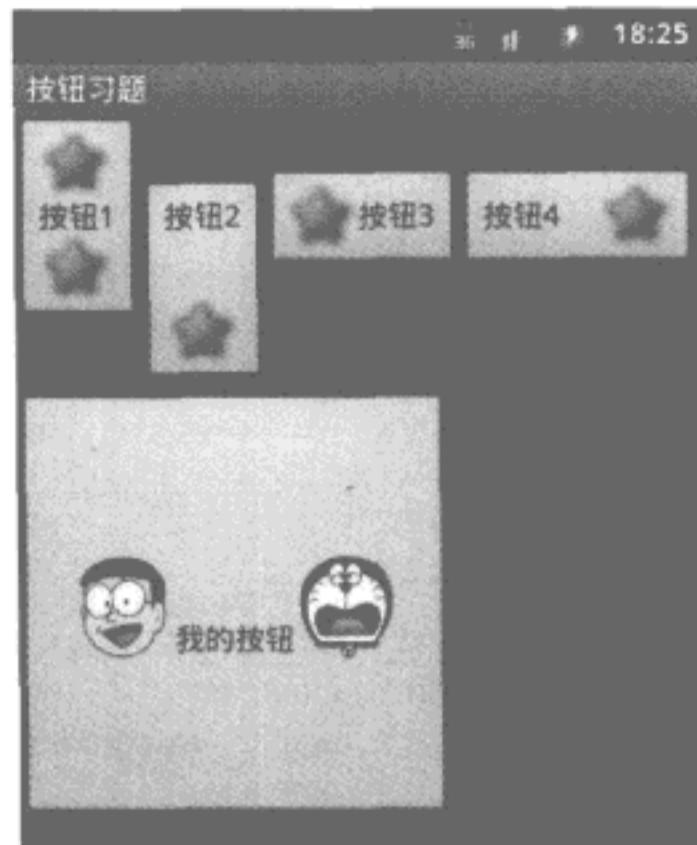
首先在布局文件中放 5 个按钮，其中前 4 个按钮分别使用 `android:drawableXxx` 属性在文字上、下、左、右插入一个图像，第 5 个按钮通过 `ImageSpan` 和 `SpannableString` 在文字两侧分别插入一个图像。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="120dp">
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:drawableTop="@drawable/star"
            android:text="按钮 1" />
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:drawableBottom="@drawable/star"
            android:text="按钮 2" android:drawablePadding="30dp" />
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:drawableLeft="@drawable/star"
            android:text="按钮 3" />
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:drawableRight="@drawable/star"
            android:text="按钮 4" android:drawablePadding="20dp" />
    </LinearLayout>
    <Button android:id="@+id/button" android:layout_width="200dp"
        android:layout_height="200dp" android:layout_marginTop="10dp" />
</LinearLayout>
```

下面的代码向第 5 个按钮上的文字两侧分别插入了两个图像。

```
// 处理文字左侧的图像
SpannableString spannableStringLeft = new SpannableString("left");
Bitmap bitmapLeft = BitmapFactory.decodeResource(getResources(), R.drawable.image_left);
ImageSpan imageSpanLeft = new ImageSpan(bitmapLeft, DynamicDrawableSpan.ALIGN_BOTTOM);
// 将 left 替换成 ImageSpan 对象
spannableStringLeft.setSpan(imageSpanLeft, 0, 4, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
// 处理文字右侧的图像
SpannableString spannableStringRight = new SpannableString("right");
Bitmap bitmapRight = BitmapFactory.decodeResource(getResources(), R.drawable.image_right);
ImageSpan imageSpanRight = new ImageSpan(bitmapRight);
// 将 right 替换成 ImageSpan 对象
spannableStringRight.setSpan(imageSpanRight, 0, 5, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
// 插入文字左侧的图像
button.append(spannableStringLeft);
// 插入文字
button.append("我的按钮");
// 插入文字右侧的图像
button.append(spannableStringRight);
```

本例的运行效果如图 5.23 所示。



▲ 图 5.23 图文混排的按钮

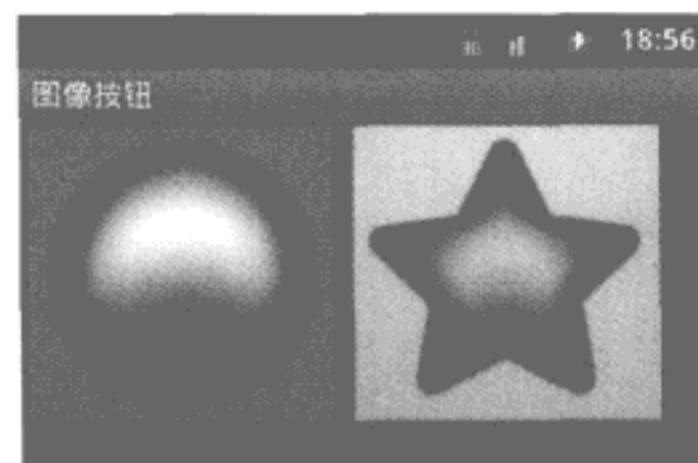
### 5.4.3 ImageButton (图像按钮控件)

工程目录: src\ch05\ch05\_imagebutton

ImageButton 可以作为图像按钮来使用。例如，下面的代码配置了两个带背景的图像按钮。

```
<ImageButton android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:src="@drawable/button1_1" />
<ImageButton android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:src="@drawable/button2_1" />
```

如果想在代码中修改 ImageButton 的图像，可以使用 ImageButton 类的 setImageResource 或其他类似的方法。本例的显示效果如图 5.24 所示。



▲ 图 5.24 图像按钮

**注意**

ImageButton 并不是 TextView 的子类，而是 ImageView 的子类，因此并没有 android:text 属性。如果要在 ImageButton 上输出文字，可以自定义一个控件，并在 onDraw 方法中将文字画在 ImageButton 上。

### 5.4.4 RadioButton (选项按钮控件)

工程目录: src\ch05\ch05\_radiobutton

选项按钮可用于多选一的应用中。如果想在选中某一个选项按钮后，其他的选项按钮都被设为

未选中状态，需要将<RadioButton>标签放在<RadioGroup>标签中。由于 RadioButton 是 Button 的子类（实际上，RadioButton 是 ComponentButton 的直接子类，而 ComponentButton 又是 Button 的直接子类），因此，在<RadioButton>标签中同样可以使用 android:drawableXxx 及 android:drawablePadding 属性。例如，下面的代码在屏幕上放置了 3 个选项按钮，其中第 3 个选项按钮周围显示了 4 个图像。

```
<RadioGroup android:layout_width="wrap_content" android:layout_height="wrap_content">
    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="选项 1" />
    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="选项 2" />
    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="选项 3"
        android:drawableLeft="@drawable/star" android:drawableTop="@drawable/circle"
        android:drawableRight="@drawable/star"
        android:drawableBottom="@drawable/circle" android:drawablePadding="20dp" />
</RadioGroup>
```

本例的显示效果如图 5.25 所示。



▲ 图 5.25 选项按钮

#### 5.4.5 ToggleButton (开关状态按钮控件)

工程目录: src\ch05\ch05\_togglebutton

**ToggleButton** 控件与 **Button** 控件的功能基本相同，但 **ToggleButton** 控件还提供了可以表示“开/关”状态的功能，这种功能非常类似于复选框（**CheckBox**）。**ToggleButton** 控件通过在按钮文字的下方显示一个绿色的指示条来表示“开/关”状态。至于绿色的指示条是表示“开”还是“关”，完全由开发人员自己决定。当指示条在绿色状态时，再次单击按钮，指示条就会变成白色。**ToggleButton** 控件的基本使用方法与 **Button** 控件相同，代码如下：

```
<ToggleButton android:layout_width="wrap_content" android:layout_height="wrap_content" />
```

虽然 **ToggleButton** 是 **Button** 的子类，但 **android:text** 属性并不起作用。在默认情况下，根据 **ToggleButton** 控件的不同状态，会在按钮上显示“关闭”或“开启”。如果要更改默认的按钮文本，可以使用 **android:textOff** 和 **android:textOn** 属性，代码如下：

```
<ToggleButton android:id="@+id/toggleButton" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:layout_marginLeft="30dp"
    android:textOff="打开电灯" android:textOn="关闭电灯" />
```

默认情况下，按钮上的指示条是白色的，如果要在 XML 布局文件中修改默认状态，可以使用 `android:checked` 属性，在代码中使用 `ToggleButton.setChecked` 方法。当 `checked` 属性值或 `setChecked` 方法的参数值为 `true` 时，指示条显示为绿色。

本例的显示效果如图 5.26 所示



▲ 图 5.26 ToggleButton 控件

#### 5.4.6 CheckBox (复选框控件)

工程目录: `src\ch05\ch05_dynamiccheckbox`

复选框通常用于多选的应用。基本的使用方法如下：

```
<CheckBox android:id="@+id/checkbox" android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

`CheckBox` 默认情况下是未选中状态。如果想修改这个默认值，可以将`<CheckBox>`标签的 `android:checked` 属性值设为 `true`，或使用 `CheckBox.setChecked` 方法设置 `CheckBox` 的状态。在代码中可以使用 `CheckBox.isChecked` 方法判断 `CheckBox` 是否被选中。如果 `isChecked` 方法返回 `true`，则表示 `CheckBox` 处于选中状态。

本节给出一个动态创建 `CheckBox` 控件的例子，也许很多读者会首先想到在代码中创建 `CheckBox` 对象。这样做虽然从技术上说没有任何问题，也属于比较常用的动态创建控件的方法，但在实际应用中，往往会为 `CheckBox` 设置很多属性，如果在代码中直接创建 `CheckBox` 对象，就需要手工设置 `CheckBox` 对象的属性，这样做是很麻烦的。

如果单纯考虑设置控件的属性，布局文件无疑是最简单的方法。我们自然会想到是否可以在布局文件中先配置一个或若干 `CheckBox`，然后以这些配置为模板来动态创建 `CheckBox` 对象呢？如果能想到这些，将会大大减少动态创建 `CheckBox` 对象的代码。

由于同一个控件不能拥有两个及以上的父控件，因此，不能直接在 `Activity` 中使用 `findViewById` 方法来获得 `CheckBox` 对象。如何解决这个问题呢？

实际上，`XML` 布局文件的根节点不仅可以是像`<LinearLayout>`、`<RelativeLayout>`一样的 `ViewGroup`，也可以直接使用 `View`，例如，`<CheckBox>`标签。因此，我们先来编写一个 `checkbox.xml` 文件来设置复选框。

#### checkbox.xml

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

```
    android:layout_height="wrap_content" />
```

为了单击“确认”按钮后显示被选中的复选框的文本，需要在复选框下方显示一个“确认”按钮。因此，在本例中还需要一个 main.xml 布局文件，用于定义“确认”按钮。

### **main.xml**

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

动态创建 CheckBox 对象的方法是定义一个 String 类型的数组，数组元素定义了 CheckBox 的文本，然后根据 String 数组的元素个数来动态创建 CheckBox 对象。动态创建 CheckBox 对象的步骤如下。

(1) 使用 getLayoutInflator().inflate(...)方法来装载 main.xml 布局文件，并返回一个 LinearLayout 对象 (linearLayout)。

(2) 使用 getLayoutInflater().inflate(...)方法来装载 checkbox.xml 布局文件，并返回一个 CheckBox 对象 (checkbox)。

(3) 根据 String 数组中的值设置 CheckBox 的文本。

(4) 调用 linearLayout.addView 方法将 checkbox 添加到 linearLayout 中。

(5) 根据 String 数组的元素重复执行第 (2) 步、第 (3) 步和第 (4) 步，直到处理完 String 数组中的最后一个元素为止。

动态创建 CheckBox 对象是在 onCreate 方法中完成的，代码如下：

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    String[] checkboxText = new String[]
    { "是学生吗？", "是否喜欢 Android？", "买车了吗？", "打算出国吗？" };
    super.onCreate(savedInstanceState);
    LinearLayout linearLayout = (LinearLayout) getLayoutInflator().inflate(
        R.layout.main, null);
    for (int i = 0; i < checkboxText.length; i++)
    {
        // 根据 checkbox.xml 文件创建 CheckBox 对象
        CheckBox checkbox = (CheckBox) getLayoutInflator().inflate(R.layout.checkbox, null);
        // 将 CheckBox 对象添加到 checkboxes 数组中，用于对 CheckBox 对象的检索
        checkboxes.add(checkbox);
        // 设置 CheckBox 对象的文本
        checkboxes.get(i).setText(checkboxText[i]);
        // 将 CheckBox 对象添加到界面的主布局中 (main.xml)，在按钮前面显示复选框
        linearLayout.addView(checkbox, i);
    }
    setContentView(linearLayout);
    Button button = (Button) findViewById(R.id.button);
```

```

        button.setOnClickListener(this);
    }
}

```

其中 `checkboxs` 是在 `Main` 类中定义的一个 `List<CheckBox>`类型的变量，用来保存动态创建的 `CheckBox` 对象。该变量需要在“确认”按钮的单击事件方法中使用，代码如下：

```

public void onClick(View view)
{
    String s = "";
    // 扫描所有的 CheckBox，以便获得被选中的复选框的文本
    for (CheckBox checkbox : checkboxes)
    {
        if (checkbox.isChecked())
            s += checkbox.getText() + "\n";
    }
    if ("".equals(s))
        s = "您还没选呢！";
    new AlertDialog.Builder(this).setMessage(s).setPositiveButton("关闭", null).show();
}

```

运行本例并选中相应的复选框，然后单击“确认”按钮，显示如图 5.27 所示的效果。



▲ 图 5.27 动态创建复选框

## 5.5 ImageView（显示图像的控件）

前面介绍过很多可以显示图像的控件，但显示图像对于这些控件来说都只是辅助功能，而本节要介绍的 `ImageView` 控件则是专门用来显示和控制图像的，例如，放大、缩小、旋转图像。

### 5.5.1 ImageView 控件的基本用法

**工程目录:** `src\ch05\ch05_imageview`

`ImageView` 控件可用于显示 Android 系统支持的图像，其支持的图像格式有 gif、jpg、png、bmp 等。在布局文件中使用`<ImageView>`标签来定义一个 `ImageView` 控件，代码如下：

```
<ImageView android:id="@+id/imageview" android:layout_width="wrap_content"
    android:background="#F00" android:layout_height="wrap_content"
    android:src="@drawable/icon" android:scaleType="center" />
```

在上面的代码中通过 `android:src` 属性指定了一个图像资源 ID，并使用 `android:scaleType` 属性指定 `ImageView` 控件显示图像的方式。例如，`center` 表示图像以不缩放的方式显示在 `ImageView` 控件的中心。如果将 `android:scaleType` 属性设为 `fitCenter`，表示将图像按比例缩放至合适的位置，并显示在 `ImageView` 控件的中心。通常在设计相框时将 `android:scaleType` 属性设为 `fitCenter`，这样可以使照片按比例显示在相框的中心。

```
<ImageView android:layout_width="300dp" android:layout_height="200dp"
    android:background="#FFF" android:src="@drawable/background"
    android:scaleType="fitCenter" android:padding="10dp" />
```

上面的代码直接设置了 `ImageView` 控件的宽度和高度，也可以在代码中设置和获得 `ImageView` 控件的宽度和高度。

```
ImageView imageView = (ImageView) findViewById(R.id.imageview);
// 设置 ImageView 控件的宽度和高度
imageView.setLayoutParams(new LinearLayout.LayoutParams(200, 100));
// 获得 ImageView 控件的宽度和高度，并将获得的值显示在 Activity 的标题栏上
setTitle("height:" + imageView.getLayoutParams().width + " height:" + imageView.getLayoutParams().height);
```

本例的显示如图 5.28 所示。



▲ 图 5.28 图像的显示方式

## 5.5.2 显示指定区域的图像

工程目录: src\ch05\ch05\_rectimageview

虽然 `ImageView` 控件可以用不同的缩放类型（通过 `scaleType` 属性设置）显示图像，但遗憾的

是 ImageView 控件只能显示整个图像。如果只想显示图像的某一部分，单纯使用 ImageView 就无能为力了。

尽管 ImageView 控件无法实现这个功能，但可以采用“曲线救国”的方法来达到只显示图像的某一部分的目的。该方法的基本原理是首先获得原图像的 Bitmap 对象，然后使用 Bitmap.createBitmap 方法将要显示的图像区域生成新的 Bitmap 对象，最后将这个新的 Bitmap 对象显示在 ImageView 控件上。

本例使用了一个分辨率为 500\*408 的图像文件 (dog.jpg)，当触摸该图像的某一点时，会将以该点为左上顶点的一个正方形区域复制到另一个 100\*100 的 ImageView 控件中。下面的代码定义了两个 ImageView 控件，分别用于显示原图像和原图某个区域的图像。

```
<!-- 显示原图像 -->
<ImageView android:id="@+id/imageview1" android:layout_width="fill_parent"
    android:background="#F00" android:layout_height="261dp" android:src="@drawable/
    background" />
<!-- 显示原图像的指定区域 -->
<ImageView android:id="@+id/imageview2" android:layout_width="100dp"
    android:background="#F00" android:layout_height="100dp"
    android:layout_marginTop="10dp" android:scaleType="fitCenter" />
```

由于本例所使用的图像的分辨率是 500\*408，而模拟器的分辨率是 320\*480，因此，需要将图像按等比例缩小。按比例计算，ImageView 控件的高度为 261。

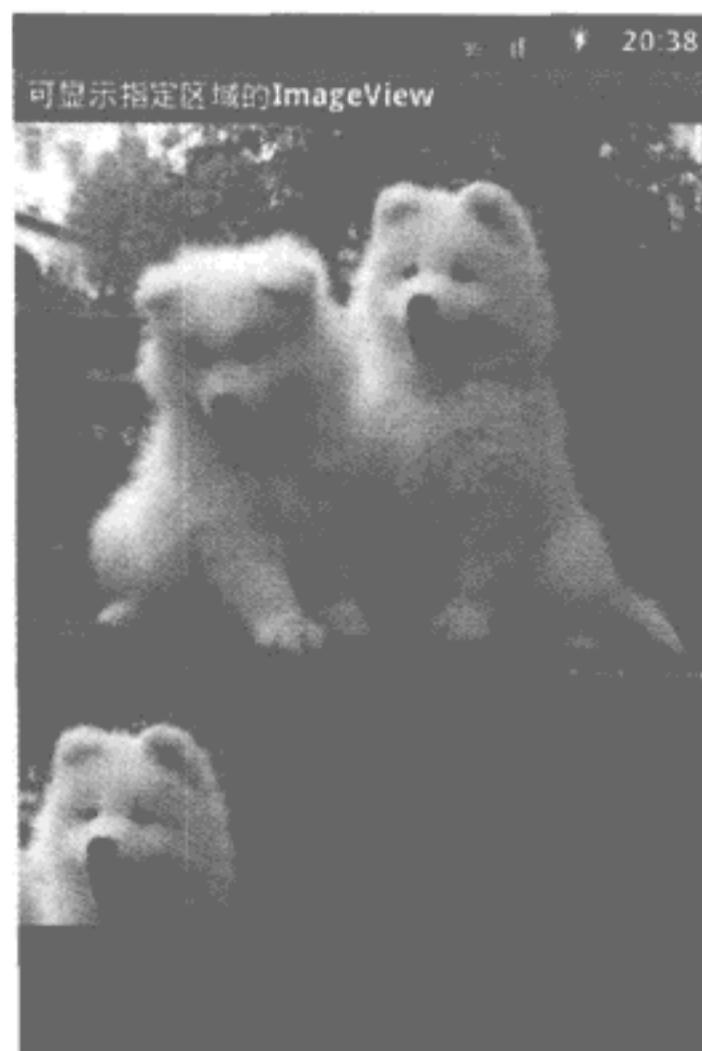
本例的核心代码在 ImageView 类的 onTouch 方法中，要捕捉 onTouch 事件，必须实现 OnTouchListener 接口。onTouch 方法的代码如下：

```
@Override
public boolean onTouch(View view, MotionEvent event)
{
    try
    {
        float scale = (float)(500.0 / 320);
        // 计算转换比例，320 为 ImageView 的宽度，也是屏幕的宽度
        int x = (int) (event.getX() * scale);
        int y = (int) (event.getY() * scale);
        int width = (int) (100 * scale);
        int height = (int) (100 * scale);
        BitmapDrawable bitmapDrawable = (BitmapDrawable) imageView1.getDrawable();
        // 从原图像上截取指定区域的图像，并将生成的 Bitmap 对象显示在第 2 个 ImageView 控件中
        imageView2.setImageBitmap(Bitmap.createBitmap(bitmapDrawable.getBitmap(), x, y,
            width, height));
    }
    catch (Exception e)
    {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
    }
    return false;
}
```

由于 dog.jpg 在 ImageView 控件中是按比例缩小显示的，因此，在复制图像区域时需要将触摸点及图像区域的 width 和 height 转换成实际图像的值。在上面的代码中首先计算了一个转换比例

(`scale` 变量), 其中 500 为原图像的宽度, 320 是 `ImageView` 控件中的宽度 (`ImageView` 的宽度和模拟器的宽度相同)。

运行本例后, 单击第 1 个 `ImageView` 控件中的某一点, 将显示如图 5.29 所示的效果。



▲ 图 5.29 显示图像的指定区域

**扩展学习：**如何装载大图像。我们可以通过 `android.graphics.BitmapFactory` 类的静态方法从不同来源获得 `Bitmap` 对象。例如, 下面的代码从 SD 卡获得了一个 `Bitmap` 对象。

```
Bitmap bitmap = BitmapFactory.decodeFile("/sdcard/abc.jpg");
```

虽然从 SD 卡、资源中装载图像, 并获得 `Bitmap` 对象很容易, 但这里有一个问题, 手机毕竟不是 PC, 内存和其他硬件资源有限 (至少在可预见的未来手机还无法和同一时代的 PC 相比), 无法一次性装载过大的图像 (例如, 装载文件大小在 1MB 以上的图像很可能会出现内存溢出错误)。那么在这种情况下, 就需要将图像按比例缩小来装载 (不是显示效果上的缩小, 而是图像本身的缩小, 这一点和 `ImageView` 控件按比例缩小图像不同), 这就需要使用 `decodeFile` 方法的第二个参数 (其他类似的方法, 例如, `decodeResource` 方法也有这个参数) 来设置缩小的比例。

`decodeFile` 的第二个参数的类型是 `android.graphics.BitmapFactory.Options`, `Options` 类有一个 `inSampleSize` 属性用于设置图像缩小的比例。该属性是 `int` 类型, 例如, 如果该属性的值是 5, 则以 1/5, 也就是 20% 的大小来缩小原图像。下面的代码将以 25% 的大小生成一个缩小版的 `Bitmap` 对象。

```
Options options = new Options();
options.inSampleSize = 4;
Bitmap bitmap = BitmapFactory.decodeFile("/sdcard/abc.jpg", options);
```

在获得 `Bitmap` 对象后, 就可以直接使用 `ImageView` 类的 `setImageBitmap` 方法将缩小版的图像在 `ImageView` 控件上显示了。

### 5.5.3 缩放和旋转图像

工程目录: `src\ch05\ch05_resize_round_image`

缩放图像的方法很多, 最简单的方法无疑是改变 `ImageView` 控件的大小, 但应将`<ImageView>`标签的 `android:scaleType` 属性值设为 `fitCenter`。旋转图像可以用 `android.graphics.Matrix` 类的 `setRotate` 方法来实现, 通过该方法可以指定旋转的度数。

本节的例子提供了两个滚动条, 第 1 个 `SeekBar`(将在后面详细介绍)用于缩放图像(`android:max` 属性值为 240), 第 2 个 `SeekBar` 用于旋转图像(`android:max` 属性值为 360, 也就是说, 通过该 `SeekBar` 可以使图像最多旋转 360°)。

为了自适应屏幕的宽度(图像放大到与屏幕宽度相等时为止), 在本例中没有直接将第 1 个 `SeekBar` 的 `android:max` 属性值设为 240, 而是使用如下代码来获得屏幕的宽度, 并将屏幕宽度与图像的最小宽度(在本例中是 `minWidth` 变量, 值为 80)的差作为 `android:max` 属性的值。

```
DisplayMetrics dm = new DisplayMetrics();
get WindowManager().getDefaultDisplay().getMetrics(dm);
seekBar1.setMax(dm.widthPixels - minWidth);
```

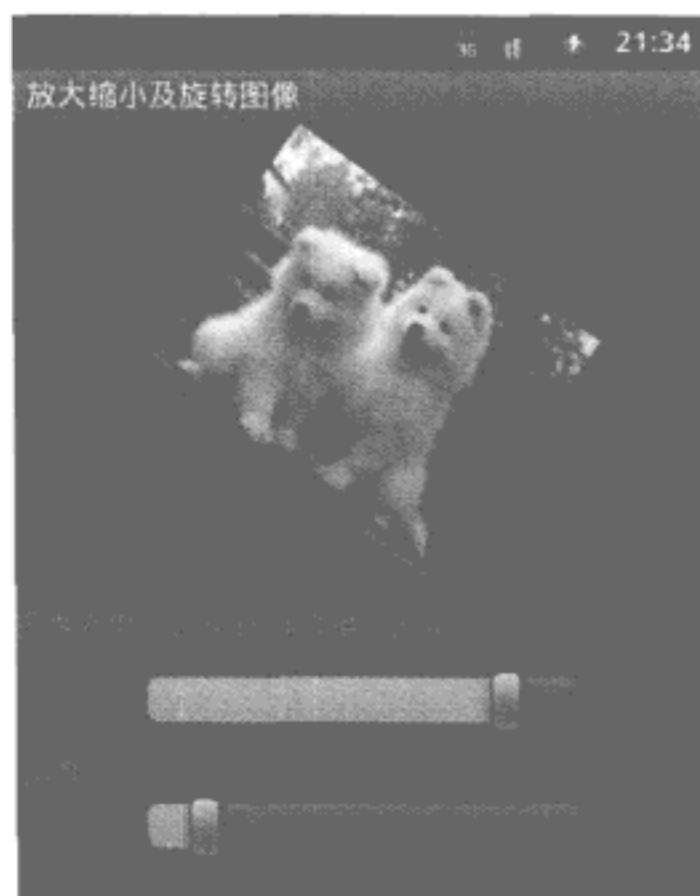
在 `SeekBar` 类的 `onProgressChanged` 事件方法中需要控制图像的缩放和旋转, 代码如下:

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
{
    // 处理图像缩放
    if (seekBar.getId() == R.id.seekBar1)
    {
        int newWidth = progress + minWidth;                                // 计算缩放后图像的新宽度
        int newHeight = (int) (newWidth * 3 / 4);                          // 计算缩放后图像的新高度
        // 设置 ImageView 的大小
        imageView.setLayoutParams(new LinearLayout.LayoutParams(newWidth, newHeight));
        textView1.setText("图像宽度: " + newWidth + " 图像高度: " + newHeight);
    }
    // 处理图像旋转
    else if (seekBar.getId() == R.id.seekBar2)
    {
        // 装载 dog.jpg 文件, 并返回该文件的 Bitmap 对象
        Bitmap bitmap = ((BitmapDrawable) getResources().getDrawable(R.drawable.dog)).getBitmap();
        // 设置图像的旋转角度
        matrix.setRotate(progress);
        // 旋转图像, 并生成新的 Bitmap 对象
        bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.getWidth(), bitmap.getHeight(),
            matrix, true);
        // 重新在 ImageView 组件中显示旋转后的图像
        imageView.setImageBitmap(bitmap);
        textView2.setText(progress + "度");
    }
}
```

在编写上面代码时需要注意以下几点。

- 由于在本例中允许图像的最小宽度是 80，因此，缩放后的新宽度应为 seekBar1 的当前进度与最小宽度（minWidth）之和。新高度可以根据新宽度计算出来。
- 由于图像的宽度和高度之比是 4:3，因此，显示图像的 ImageView 控件的 android:layout\_width 和 android:layout\_height 属性的值的比例也应该是 4:3，例如，在本例中这两个属性值分别是 200dp 和 150dp，并且为了保证图像和 ImageView 控件的大小相同，android:scaleType 属性值应设为 fitCenter。

运行本例后，拖动第 1 个和第 2 个SeekBar 对图像进行缩放和旋转，将显示如图 5.30 所示的效果。



▲ 图 5.30 缩放和旋转图像

## 5.6 时间与日期控件

在很多 Android 应用中都需要设置日期和时间。当然，最简单的设置日期和时间的方法是提供一个 EditText 控件，但这种方式显得不太友好。Android SDK 提供了两个控件：DatePicker 和 TimePicker，分别以可视化的方式输入日期和时间。除此之外，Android SDK 还提供了显示时间的两个控件：DigitalClock 和 AnalogClock，分别以数字方式和表盘方式显示时间。

### 5.6.1 DatePicker（输入日期的控件）

DatePicker 控件可用于输入日期。日期的输入范围是 1900-1-1~2100-12-31。DatePicker 控件的基本使用方法如下：

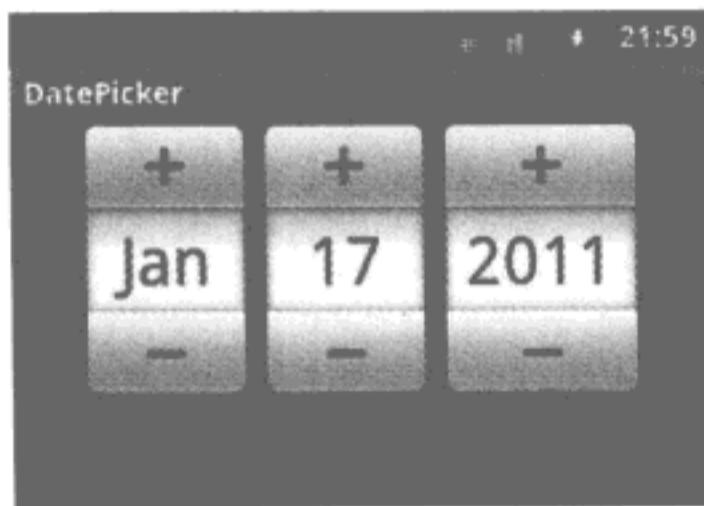
```
<DatePicker android:id="@+id/datepicker"
    android:layout_width="fill_parent" android:layout_height="wrap_content" />
```

通过 DatePicker 类的 getYear、getMonth 和 getDayOfMonth 方法可以分别获得 DatePicker 控件当前显示的年、月、日。通过 DatePicker 类的 init 方法对 DatePicker 控件进行初始化。init 方法的

定义如下：

```
public void init(int year, int monthOfYear, int dayOfMonth, OnDateChangedListener onDate
ChangedListener)
```

其中 `year`、`monthOfYear` 和 `dayOfMonth` 参数分别用来设置 `DatePicker` 控件的年、月、日。`onDateChangedListener` 参数用来设置 `DatePicker` 控件的日期变化事件对象。该对象对应的类必须实现 `android.widget.DatePicker.OnDateChangedListener` 接口。`DatePicker` 控件的显示效果如图 5.31 所示。



▲ 图 5.31 DatePicker 的显示效果

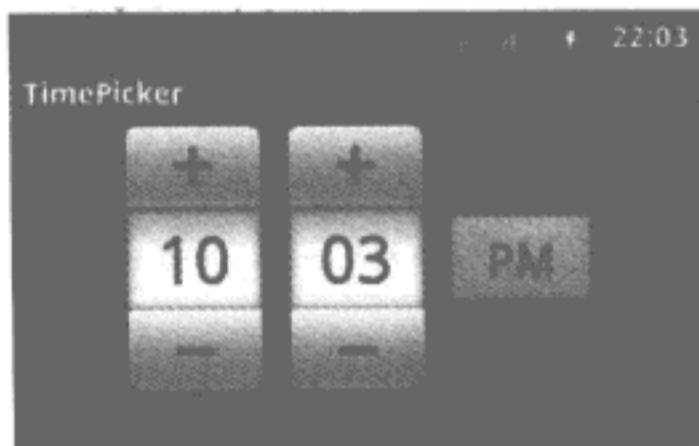
### 5.6.2 TimePicker (输入时间的控件)

`TimePicker` 控件用来输入时间（只能输入小时和分钟）。该控件的基本用法如下：

```
<TimePicker android:id="@+id/timepicker"
    android:layout_width="fill_parent" android:layout_height="wrap_content" />
```

`TimePicker` 在默认情况下是 12 小时进制，如图 5.32 所示。如果想以 24 小时进制显示时间，可以使用 `TimePicker` 类的 `setIs24HourView` 方法设置，以 24 小时进制显示时间的 `TimePicker` 控件如图 5.33 所示。

当 `TimePicker` 的时间变化时，会触发 `OnTimeChanged` 事件，但与 `DatePicker` 控件不同的是，`TimePicker` 通过 `setOnTimeChangedListener` 方法设置时间变化的事件对象，而 `DatePicker` 通过 `init` 方法设置日期变化的事件对象。



▲ 图 5.32 12 小时进制



▲ 图 5.33 24 小时进制

### 5.6.3 DatePicker、TimePicker 与 TextView 同步显示日期和时间

工程目录：src\ch05\ch05\_datetimepicker

在本例中使用了 DatePicker、TimePicker 和 TextView 控件。当 DatePicker 和 TimePicker 中的日期、时间变化时，TextView 会显示变化后的日期和时间。

```
package net.blogjava.mobile;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.DatePicker.OnDateChangedListener;
import android.widget.TimePicker.OnTimeChangedListener;

public class Main extends Activity implements OnDateChangedListener, OnTimeChangedListener
{
    private TextView textView;
    private DatePicker datePicker;
    private TimePicker timePicker;
    @Override
    public void onTimeChanged(TimePicker view, int hourOfDay, int minute)
    {
        // 调用 onDateChanged 事件方法在 TextView 中显示当前的日期和时间
        onDateChanged(null, 0, 0, 0);
    }
    @Override
    public void onDateChanged(DatePicker view, int year, int monthOfYear,
                             int dayOfMonth)
    {
        Calendar calendar = Calendar.getInstance();
        calendar.set(datePicker.getYear(), datePicker.getMonth(), datePicker
                     .getDayOfMonth(), timePicker.getCurrentHour(), timePicker.getCurrentMinute());
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy 年 MM 月 dd 日 HH:mm");
        // 在 TextView 中显示当前的日期和时间
        textView.setText(sdf.format(calendar.getTime()));
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        datePicker = (DatePicker) findViewById(R.id.datepicker);
        timePicker = (TimePicker) findViewById(R.id.timepicker);
        datePicker.init(2001, 1, 25, this);
        timePicker.setIs24HourView(true);
        timePicker.setOnTimeChangedListener(this);
    }
}
```

```

    textView = (TextView) findViewById(R.id.textview);
    // 在 TextView 上显示 DatePicker 及 TimePicker 上的日期和时间
    onDateChanged(null, 0, 0, 0);
}
}

```

运行本例后，将显示如图 5.34 所示的效果。



▲ 图 5.34 与 TextView 同步日期和时间

#### 5.6.4 AnalogClock 和 DigitalClock ( 显示时钟的控件 )

工程目录: `src\ch05\ch05_clock`

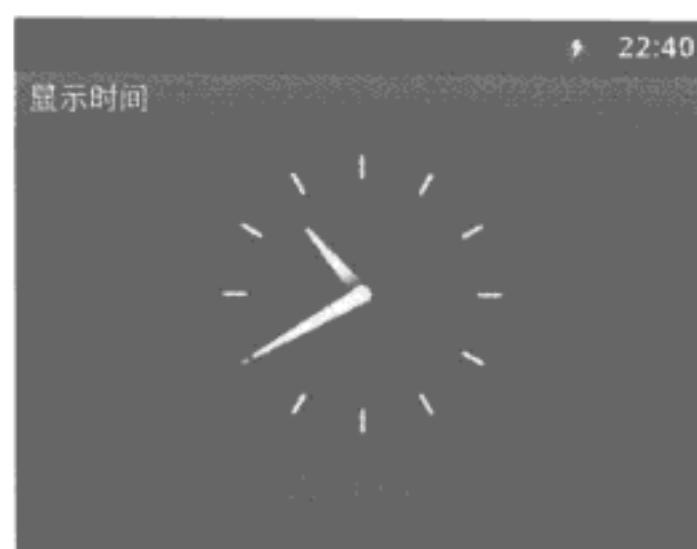
`AnalogClock` 控件用于以表盘方式显示当前时间，该控件只有两个指针（时针和分针）。使用方法如下：

```
<AnalogClock android:layout_width="fill_parent" android:layout_height="wrap_content" />
```

`DigitalClock` 控件用于以数字方式显示当前时间，该控件可以显示时、分、秒。使用方法如下：

```
<DigitalClock android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:textSize="18dp" />
```

本例的显示效果如图 5.35 所示。



▲ 图 5.35 显示时间的控件

## 5.7 进度条控件

任务或工作的进度是软件中经常要展现给用户的信息，这些信息的载体总是离不开进度条。在 Android SDK 中提供了一个 ProgressBar 控件，该控件可以向用户展示当前任务的进度。除此之外，SeekBar 和 RatingBar 控件从本质上讲也应属于进度条，只不过这两个控件对进度条的功能做了进一步改进，也可以将它们看作是进度条的变种。本节将详细介绍这 3 个控件的用法。

### 5.7.1 ProgressBar（进度条控件）

工程目录：src\ch05\ch05\_progressbar

ProgressBar 控件在默认情况下是圆形的进度条，可通过 style 属性将圆形进度条设为大、中、小 3 种形式，代码如下：

```
<!-- 圆形进度条（小） -->
<ProgressBar android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleSmallTitle" />
<!-- 圆形进度条（中） -->
<ProgressBar android:layout_width="wrap_content" android:layout_height="wrap_content" />
<!-- 圆形进度条（大） -->
<ProgressBar android:layout_width="wrap_content"
    android:layout_height="wrap_content" style="?android:attr/progressBarStyleLarge" />
```

ProgressBar 控件在默认情况下显示的是中型的圆形进度条，因此，要想显示中型的圆形进度条，并不需要设置 style 属性。

除了圆形进度条外，ProgressBar 控件还支持水平进度条，代码如下：

```
<ProgressBar android:id="@+id/progressBarHorizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleHorizontal" android:max="100"
    android:progress="30" android:secondaryProgress="60" android:layout_marginTop=
    "20dp" />
```

ProgressBar 控件的水平进度条支持两级进度，分别使用 android:progress 和 android:secondaryProgress 属性设置，进度条的总刻度使用 android:max 属性设置。在本例中 android:max 属性的值为 100，android:progress 和 android:secondaryProgress 属性的值分别是 30 和 60。也就是说，第一级进度和第二级进度分别显示在进度条总长度 30% 和 60% 的位置上。

android:max 属性的值不一定是 100，该值可以是任意一个合法的正整数，例如，360。一般来说，android:progress 和 android:secondaryProgress 属性的值要小于等于 android:max 属性的值。当然，如果这两个属性的值大于 android:max 属性的值，则会显示 100% 的状态。如果这两个属性的值小于 0，则会显示 0% 的状态。如果只想使用一级进度，可以只设置 android:progress 或 android:secondaryProgress 属性。

在代码中设置水平进度条的两级进度需要使用 ProgressBar 类的 setProgress 和 setSecondary

Progress 方法，代码如下：

```
ProgressBar progressBarHorizontal = (ProgressBar) findViewById(R.id.progressBarHorizontal);
progressBarHorizontal.setProgress((int) (progressBarHorizontal.getProgress() * 1.2));
progressBarHorizontal.setSecondaryProgress((int)
(progressBarHorizontal.getSecondaryProgress() * 1.2));
```

Android 系统还支持将水平和圆形进度条放在 Activity 的标题栏上。例如，将圆形进度条放在标题栏上可以在 onCreate 方法中使用如下代码：

```
// 必须在调用 setContentView 方法之前设置窗口的 Feature
requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
setContentView(R.layout.main);
setProgressBarIndeterminateVisibility(true); // 显示圆形进度条
```

如果要将水平进度条放在标题栏上，可以在 onCreate 方法中使用如下代码：

```
requestWindowFeature(Window.FEATURE_PROGRESS);
setContentView(R.layout.main);
setProgressBarVisibility(true); // 显示水平进度条
setProgress(3500); // 设置水平进度条的当前进度
```

将进度条放在标题栏上时应注意如下几点。

- requestWindowFeature 方法应在调用 setContentView 方法之前调用，否则系统会抛出异常。
- setProgressBarIndeterminateVisibility、setProgressBarVisibility 和 setProgress 方法要在调用 setContentView 方法之后调用，否则这些方法无效。
- 放在标题栏上的水平进度条不能设置进度条的最大刻度，这是因为系统已经将最大刻度值设为 10000，也就是说，用 setProgress 方法设置的进度应在 0~10000 之间。例如，本例中设为 3500，进度会显示在进度条总长的 35% 的位置上。

本例的显示效果如图 5.36 所示。单击“增加进度”和“减小进度”按钮后，最后一个进度条会以当前进度 20% 的速度递增和递减。



▲ 图 5.36 水平和圆形进度条

## 5.7.2 SeekBar(拖动条控件)

工程目录: src\ch05\ch05\_seekbar

SeekBar控件可以通过拖动滑杆改变当前的值。可以利用SeekBar来设置具有一定范围的变量值。例如，在5.5.3节的例子中使用了SeekBar控件来控制图像的旋转角度。那么在这个例子中SeekBar设置的变量就是图像旋转的角度(0至360)。下面我们再来看看SeekBar控件的基本用法，并介绍一下相关的事件。

SeekBar控件的使用方法与ProgressBar控件类似，代码如下：

```
<SeekBar android:id="@+id/seekbar" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:max="100" android:progress="30" />
<SeekBar android:id="@+id/seekbar2" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:max="100"
    android:progress="30" android:secondaryProgress="60"/>
```

虽然SeekBar是ProgressBar的子类，但一般SeekBar控件并不需要设置第二级进度(设置android:secondaryProgress属性)。如果设置了android:secondaryProgress属性，系统仍然会显示第二级的进度，不过并不会随着滑杆移动而递增或递减。

与SeekBar控件滑动相关的事件接口是OnSeekBarChangeListener，该接口定义了如下3个事件方法：

- public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser);
- public void onStartTrackingTouch(SeekBar seekBar);
- public void onStopTrackingTouch(SeekBar seekBar);

当按住滑杆后，系统会触发onStartTrackingTouch事件，在拉动滑杆进行滑动时，会触发onProgressChanged事件，松开滑杆后，会触发onStopTracking事件。

在本节的例子中有两个SeekBar控件，第1个SeekBar控件只设置了第一级进度，第2个SeekBar控件同时设置了第一级和第二级进度。这两个SeekBar控件共用一个实现OnSeekBarChangeListener接口的类(Main类)的对象。因此，需要在相应的事件方法中进行判断。例如，onProgressChanged事件方法中的代码如下：

```
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
{
    // 第一个SeekBar控件
    if (seekBar.getId() == R.id.seekbar1)
        textView2.setText("seekbar1的当前位置：" + progress);
    else // 第二个SeekBar控件
        textView2.setText("seekbar2的当前位置：" + progress);
}
```

运行本节的例子后，滑动两个SeekBar控件中的滑杆，显示效果如图5.37所示。



▲ 图 5.37 SeekBar 控件

### 5.7.3 设置 ProgressBar 和 SeekBar 的颜色及背景图

工程目录: src\ch05\ch05\_colorbar

5.7.1 节和 5.7.2 节介绍的 ProgressBar 和 SeekBar 控件的进度条都是黄色的,但为了设计出更酷的界面,有时需要改变进度条的颜色,或为进度条添加背景图。而 ProgressBar 和 SeekBar 类均未提供直接修改进度条颜色或背景图的方法或属性。不过这个问题可以通过<ProgressBar>、<SeekBar>标签的 android:progressDrawable 属性来解决。

一个完整的 ProgressBar 或 SeekBar 控件由如下 3 部分组成:

- 第一级进度条;
- 第二级进度条;
- 背景,也就是进度条未经过的地方。

因此,这两个控件的颜色也应该由 3 部分组成:第一级进度条颜色、第二级进度条颜色和背景颜色。在前面提到过,可以通过 android:progressDrawable 属性来设置 ProgressBar 和 SeekBar 的这 3 部分颜色。那么只使用一个属性如何设置 3 个颜色值呢?

我们在前面的章节中曾多次使用过 res\drawable 目录中的图像资源,知道这个目录保存的是图像文件(png、gif、jpg 等),但该资源目录中的图像资源并不一定是图像文件,还可以是 xml 文件。当然,在 res\drawable 目录中的 xml 文件与 res\values 或其他资源目录中的 xml 文件不一样, res\drawable 目录中的 xml 文件中实际上也定义了若干个图像资源,以及图像资源的各种状态。本节的例子通过 res\drawable 目录中的 xml 文件定义一组图像资源,并为每一个图像资源指定一个 ID,通过这些 ID 设置 ProgressBar 和 SeekBar 控件中的背景色(图像)、第一级进度条颜色(图像)、第二级进度条颜色(图像)。

设置上述 ProgressBar 和 SeekBar 控件的 3 种颜色的步骤如下。

(1) 确定这 3 种颜色后,使用绘图工具建立 3 个图像文件,并分别用 3 种不同的颜色填充这 3 个图像。图像的大小可任意选择。在本例中,progress.png 文件表示第一级进度条颜色;secondary.png 表示第二级进度条颜色;bg.png 表示背景颜色。

(2) 在 res\drawable 目录下建立一个 barcolor1.xml 文件,并输入如下代码:

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置背景色图像资源 -->
    <item android:id="@+id/background" android:drawable="@drawable/bg" />
    <!-- 设置第二级进度条颜色图像资源 -->
    <item android:id="@+id/secondaryProgress" android:drawable="@drawable/
secondary" />
```

```
<!-- 设置第一级进度条颜色图像资源 -->
<item android:id="@+id/progress" android:drawable="@drawable/progress" />
</layer-list>
```

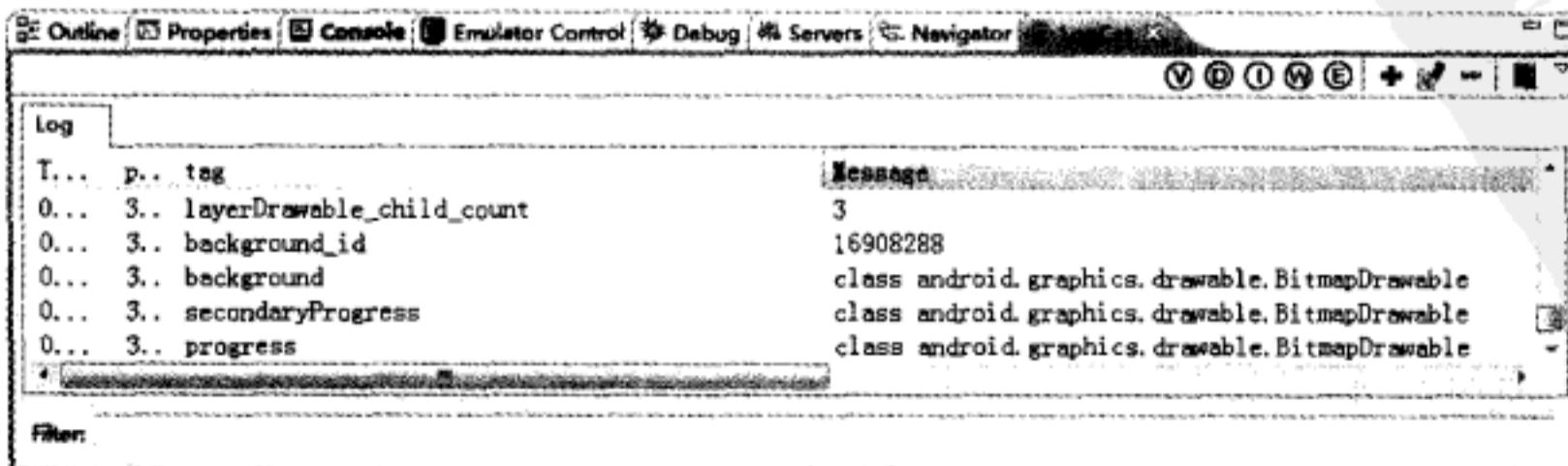
(3) 在<ProgressBar>和<SeekBar>标签中使用 android:progressDrawable 属性指定 barcolor1.xml 文件的资源 ID, 代码如下:

```
<ProgressBar android:id="@+id/progressBarHorizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    style="?android:attr/progressBarStyleHorizontal"
    android:max="100" android:progress="30" android:secondaryProgress="60"
    android:progressDrawable="@drawable/barcolor1" />
<SeekBar android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:max="100" android:progress="30"
    android:progressDrawable="@drawable/barcolor" />
```

在 barcolor1.xml 文件中设置了 3 个图像, 其中<item>标签中的 android:id 属性指的是 3 个索引, 系统会根据这 3 个索引来设置上述 3 种颜色。<item>的父标签是<layer-list>, 这个标签负责管理若干个 Drawable 对象, 其中<layer-list>必须是根节点。在程序中<layer-list>及<item>都会有对应的 Drawable 对象。例如, <layer-list>对应 LayerDrawable 对象, 在 barcolor1.xml 文件中的<item>标签对应于 BitmapDrawable 对象。因此, 我们可以使用下面的代码输出 barcolor1.xml 文件中的相应信息。

```
// 根据 barcolor1.xml 文件中的根节点(<layer-list>标签)来创建 LayerDrawable 对象
LayerDrawable layerDrawable = (LayerDrawable) getResources().getDrawable(R.drawable.barcolor1);
// 输出<layer-list>标签中子标签(<item>节点)数, 本例应为 3
Log.d("layerDrawable_child_count", String.valueOf(layerDrawable.getNumberOfLayers()));
// 输出第一个<item>标签中 android:id 属性值的十进制形式
Log.d("background_id", String.valueOf(layerDrawable.getId(0)));
// 输出第一个<item>标签对应的 Drawable 对象名(BitmapDrawable)
Log.d("background", String.valueOf(layerDrawable.getDrawable(0).getClass()));
// 输出第二个<item>标签对应的 Drawable 对象名(BitmapDrawable)
Log.d("secondaryProgress", String.valueOf(layerDrawable.getDrawable(1).getClass()));
// 输出第三个<item>标签对应的 Drawable 对象名(BitmapDrawable)
Log.d("progress", String.valueOf(layerDrawable.getDrawable(2).getClass()));
```

上面代码中的 layerDrawable.getDrawable 方法实际上获得的就是<item>标签的 android:drawable 属性值指向的图像资源的 BitmapDrawable 对象。运行上面的代码, 会在 LogCat 视图中输出如图 5.38 所示的信息。



▲ 图 5.38 输出 LayerDrawable 及其子对象的相关信息

我们也可以设置部分 ProgressBar 和 SeekBar 的颜色，例如，下面的代码只设置了背景色和第一级进度条的颜色，并且第一级进度条的颜色使用的并不是纯色的图像，而是一个 24\*24 的图像。如果图像不够宽，系统会自动以平铺方式显示该图像。

### barface.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/background" android:drawable="@drawable/bg" />
    <item android:id="@+id/progress" android:drawable="@drawable/face" />
</layer-list>
```

除了前面介绍的几种设置颜色的方法，还可以使用更高级的方式进行设置，甚至可以设置成渐变色。例如，barcolor2.xml 将背景色设为垂直方向的渐变色（由红到黑），而且增加了控件两头的圆角半径。

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/background">
        <shape>
            <corners android:radius="10dip" />
            <gradient android:startColor="#FFFF0000"
                      android:centerColor="#FF880000" android:centerY="0.75"
                      android:endColor="#FF110000" android:angle="270" />
        </shape>
    </item>
    <item android:id="@+id/secondaryProgress">
        <clip>
            <shape>
                <corners android:radius="10dp" />
                <gradient android:startColor="#FF00FF00"
                          android:centerColor="#FF00FF00" android:centerY="0.75"
                          android:endColor="#FF00FF00" android:angle="270" />
            </shape>
        </clip>
    </item>
    <item android:id="@+id/progress">
        <clip>
            <shape>
                <corners android:radius="10dp" />
                <gradient android:startColor="#fffffd300"
                          android:centerColor="#fffffb600" android:centerY="0.75"
                          android:endColor="#fffffc00" android:angle="270" />
            </shape>
        </clip>
    </item>
</layer-list>
```

在上面的代码中使用了很多新的标签，如<shape>、<clip>等，这些标签都对应新的 Drawable 对象，这些内容将在后面详细讲解，在这里只要知道 barcolor2.xml 直接通过颜色值设置了 ProgressBar 和 SeekBar 的（渐变）颜色、两侧的圆角半径等值。现在运行本节的例子，会看到如图 5.39 所示的效果。



▲ 图 5.39 设置 ProgressBar 和 SeekBar 的颜色

#### 5.7.4 RatingBar (评分控件)

工程目录: src\ch05\ch05\_ratingbar

在很多电子相册、网上书店、博客中都会有对照片、图书和文章进行评分的功能（很多评分系统都是满分为 5 分，分 10 个级，0~5，步长为 0.5）。在 Android SDK 中也提供了 RatingBar 控件用来完成类似的工作。

RatingBar 控件使用<RatingBar>标签进行配置。该标签有如下几个与评分相关的属性。

- android:numStars: 指定用于评分的五角星数，默认情况下是根据布局的设置尽量横向填充。
- android:rating: 指定当前的分数。
- android:stepSize: 指定分数的增量单位（步长），默认是 0.5。

下面的代码分别设置了不同的五角星数和步长。

```
<RatingBar android:id="@+id/ratingbar1" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:numStars="3" android:rating="2" />
<RatingBar android:id="@+id/ratingbar2" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:numStars="5" android:stepSize="0.1" />
```

除此之外，还可以为 RatingBar 控件设置不同的风格，代码如下：

```
<!-- 设置小五角星风格 -->
<RatingBar android:id="@+id/smallRatingbar" style="?android:attr/ratingBarStyleSmall"
    android:layout_marginLeft="5dip" android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
<!-- 设置指示五角星风格 -->
<RatingBar android:id="@+id/indicatorRatingbar" style="?android:attr/ratingBarStyleIndicator"
    android:layout_marginLeft="5dip" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:stepSize="0.1"/>
```

通过实现 android.widget.RatingBar.OnRatingBarChangeListener 接口可以监听 RatingBar 控件的动作。当 RatingBar 控件的分数变化后，系统会调用 OnRatingBarChangeListener 接口的 onRatingChanged 方法。可以在该方法中编写处理分数变化的代码，例如，更新小五角星风格和指示五角星风格的 RatingBar 控件的当前分数。

```

public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser)
{
    smallRatingBar.setRating(rating);           // 更新小五角星风格的 RatingBar 控件的当前分数
    indicatorRatingBar.setRating(rating);        // 更新指示五角星风格的 RatingBar 控件的当前分数
    if (ratingBar.getId() == R.id.ratingbar1)
        textView.setText("ratingbar1 的分数: " + rating);
    else
        textView.setText("ratingbar2 的分数: " + rating);
}

```

运行本节的例子后，为前两个 RatingBar 控件评分后，后两个 RatingBar 控件也会更新成相应的分数，显示效果如图 5.40 所示。



▲ 图 5.40 RatingBar 控件

### 扩展学习：?和@的区别。

在前面几节的控件标签的 style 属性中使用了?和@两个符号，也许很多读者并不是很清楚这两个符号有什么区别，那么在这里我们来讲一下它们各自所起的作用。

“@”在前面的章节多次使用过。功能就是引用系统或自定义的资源。例如，@drawable/icon 引用了 res\drawable 目录中的 icon.png 图像资源。在 style 属性中也可以使用@来引用事先定义的风格（style），相当于定义了一组控件属性的值，如果不同控件的多个属性需要设置相同的值，如多个<TextView>标签的 android:background 属性要设置同一个背景图，就可以定义如下的 style。

```

<style name="custom_style">
    <item name="android:background">#FFF</item>
</style>

```

在<TextView>标签中可以使用如下的代码来引用这个 style。

```
<TextView ... ... style = "@style/custom_style"/>
```

“?”则是用来引用系统主题属性的，“？”所指定的属性必须在系统主题中存在。系统主题是在 Android SDK 中预定义的。例如，设置 RatingBar 为小五角星风格的主题属性的代码如下：

```
<item name="ratingBarStyleSmall">@android:style/Widget.RatingBar.Small</item>
```

引用这个系统主题的代码如下：

```
<RatingBar ... style="?android:attr/ratingBarStyleSmall" />
```

使用“?”引用系统主题属性时还可以使用下面的简化形式。

```
<RatingBar ... style="?android:ratingBarStyleSmall" />
```

虽然<item>标签中定义了系统的一个 style (@android:style/Widget.RatingBar.Small)，但这个 style 对于非系统的代码是无法访问的（就像系统中的内部类一样），因此，不能使用下面的代码直接引用系统的 style，这一点要注意。

```
<RatingBar ... style="@android:style/ Widget.RatingBar.Small " />
```

## 5.8 列表控件

Android 中的列表控件非常灵活，可以自定义每一个列表项。实际上，每一个列表项就是一个 View。本节将介绍 Android SDK 中的 3 个列表控件：ListView、ExpandableListView 和 Spinner。其中 Spinner 就是在 Windows 中经常看到的下拉列表框。

### 5.8.1 ListView（普通列表控件）

**工程目录：**src\ch05\ch05\_listview

ListView 控件用于以列表的形式显示数据。ListView 控件采用 MVC 模式将前端显示与后端数据进行分离。也就是说，ListView 控件在装载数据时并不是直接使用 ListView.add 或类似的方法添加数据，而是需要指定一个 Adapter 对象，该对象相当于 MVC 模式中的 C（控制器，Controller）。ListView 相当于 MVC 模式中的 V（视图，View），用于显示数据。为 ListView 提供数据的 List 或数组相当于 MVC 模式中的 M（模型，Model）。

在 ListView 控件中通过 Adapter 对象获得需要显示的数据。在创建 Adapter 对象时需要指定要显示的数据（List 或数组对象），因此，要显示的数据与 ListView 之间通过 Adapter 对象进行连接，同时又互相独立。也就是说，ListView 只知道显示的数据来自 Adapter，并不知道这些数据是来自 List 还是数组。对于数据来说，只知道将这些数据添加到 Adapter 对象中，并不知道这些数据会被用于 ListView 控件或其他控件。

在操作 ListView 控件之前，先来定义一个 ListView 控件，代码如下：

```
<ListView android:id="@+id/lvCommonListView"
    android:layout_width="fill_parent" android:layout_height="wrap_content" />
```

向 ListView 控件装载数据之前需要创建一个 Adapter 对象，代码如下：

```
ArrayAdapter<String> aaData = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, data);
```

在上面的代码中创建了一个 android.widget.ArrayAdapter 对象。ArrayAdapter 类的构造方法需要一个 android.content.Context 对象，因此，在本例中使用当前 Activity 的对象实例（this）作为 ArrayAdapter 类的构造方法的第一个参数值。除此之外，ArrayAdapter 还需要完成如下两件事：

## Android 开发权威指南

- 指定列表项的布局文件的资源 ID;
- 指定在列表项中显示的数据。

其中布局文件的资源 ID 通过 `ArrayAdapter` 类的构造方法的第 2 个参数进行传递，列表项中显示的数据(`List`对象或数组)通过第 3 个参数进行传递。在本例中使用了 Android SDK 提供的 XML 布局文件 (`simple_list_item_1.xml`)，该布局文件对应的资源 ID 是 `android.R.layout.simple_list_item_1`。这个布局文件可以在<Android SDK 安装目录>\platforms\android-9\data\res\layout 目录中找到，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:paddingLeft="6dip"
    android:minHeight="?android:attr/listPreferredItemHeight"
/>
```

从上面的代码可以看出，在 `simple_list_item_1.xml` 文件中只定义了一个`<TextView>`标签，因此，使用这个布局文件相当于在 `ListView` 中只显示简单的文本列表项。

`ArrayAdapter` 类的构造方法的第 3 个参数值 (`data`) 是一个 `String[]` 对象，代码如下：

```
private static String[] data = new String[]
{
    "天地逃生",
    "保持通话",
    "乱世佳人(飘)",
    "怪侠一枝梅",
    "第五空间",
    "孔雀翎",
    "变形金刚 3 (真人版)",
    "星际传奇",
    "《大笑江湖》剧中，小鞋匠是小沈阳，他常强出头，由不懂武功的菜鸟变成武林第一高手；赵本山则是个武功高强的大盗，被不会武功的小沈阳打败；程野扮演赵本山的手下皮丘，经常拖累赵本山。其余角色都围绕小沈阳设置。"};

```

除了可以使用 `String[]` 对象作为 Adapter 的数据源外，还可以使用 `List` 对象作为 Adapter 的数据源。因此，可以使用 `List` 对象来代替上面代码中的 `data` 变量。

在创建完 `ArrayAdapter` 对象后，需要使用 `ListView` 类的 `setAdapter` 方法将 `ArrayAdapter` 对象与 `ListView` 控件绑定，代码如下：

```
ListView lvCommonListView = (ListView) findViewById(R.id.lvCommonListView);
lvCommonListView.setAdapter(aaData);
```

当调用 `setAdapter` 方法后，`ListView` 控件的每一个列表项都会使用 `simple_list_item_1.xml` 文件定义的模板来显示，并将 `data` 数组中的每一个元素赋值给每一个列表项（列表项就是在

simple\_list\_item\_1.xml 中定义的 TextView 控件)。

在默认情况下, ListView 控件选中的是第 1 项。如果想一开始就选中指定的列表项, 需要使用 ListView 类的 setSelection 方法进行设置, 代码如下:

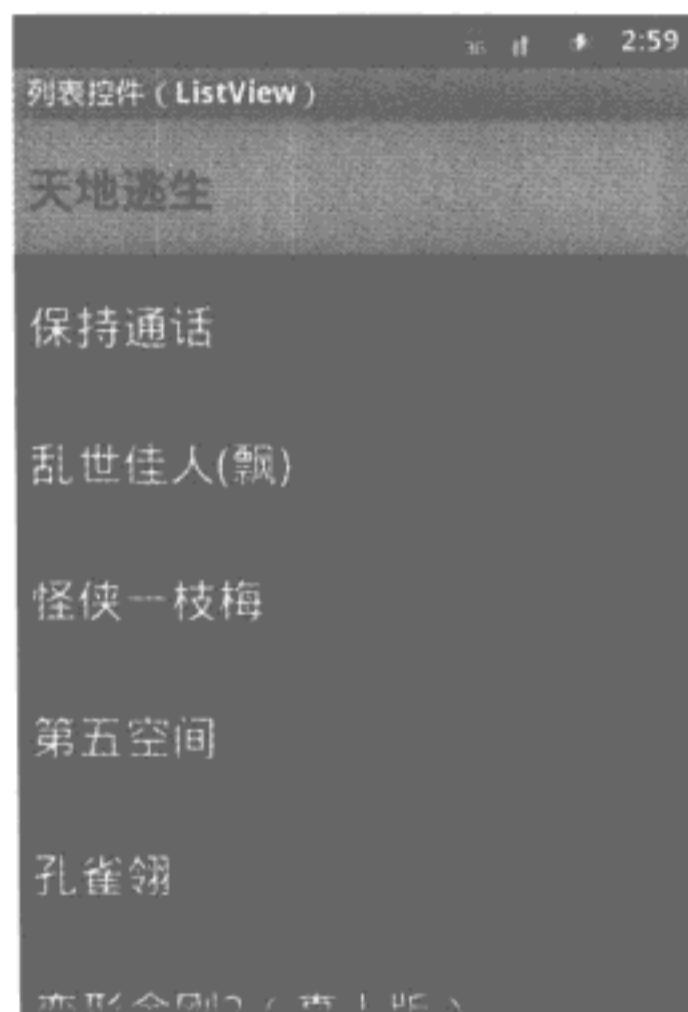
```
| lvCommonListView.setSelection(6); // 选中第 7 个列表项
```

与列表项相关的有如下两个事件:

- ItemSelected (列表项被选中时触发);
- ItemClick (单击列表项时触发)。

为了截获这两个事件, 需要分别实现 OnItemSelectedListener 和 OnItemClickListener 接口。在本例中分别在这两个接口的事件方法中输出了相应的日志信息, 读者可以在 LogCat 视图中查看这些事件的调用顺序。

运行本节的例子后, 将显示如图 5.41 所示的效果。



▲ 图 5.41 ListView 控件

## 5.8.2 为 ListView 列表项添加复选框和选项按钮

工程目录: src\ch05\ch05\_choicelistview

如果想选择多个列表项, 就需要在每个列表项上添加 RadioButton、CheckBox 等控件。当然, 向列表项添加控件的方法很多, 但 ListView 提供了一种非常简单的方式向列表项添加多选按钮 (RadioButton)。这种方法只需要使用 simple\_list\_item\_multiple\_choice.xml 布局文件即可, 该布局文件对应的资源 ID 如下:

```
| android.R.layout.simple_list_item_multiple_choice
```

除此之外, 可以向列表项添加 CheckBox 和 CheckedTextView (用对号作为被选择的标志) 控件。添加这两个控件分别需要使用 simple\_list\_item\_single\_choice.xml 和 simple\_list\_item\_checked.

xml 布局文件，这两个布局文件分别对应如下资源 ID：

```
|| android.R.layout.simple_list_item_single_choice
|| android.R.layout.simple_list_item_checked
```

虽然从表面上看，使用上述 3 个布局文件添加的是 RadioButton、CheckBox 和 CheckedTextView 控件，但实际上，在这 3 个布局文件中只使用了 CheckedTextView 控件。之所以会显示不同的风格，是因为设置了<CheckedTextView>标签的 android:checkMark 属性，例如，simple\_list\_item\_multiple\_choice.xml 文件的内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<CheckedTextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:checkMark="?android:attr/listChoiceIndicatorSingle"
    android:paddingLeft="6dip"
    android:paddingRight="6dip"
/>
```

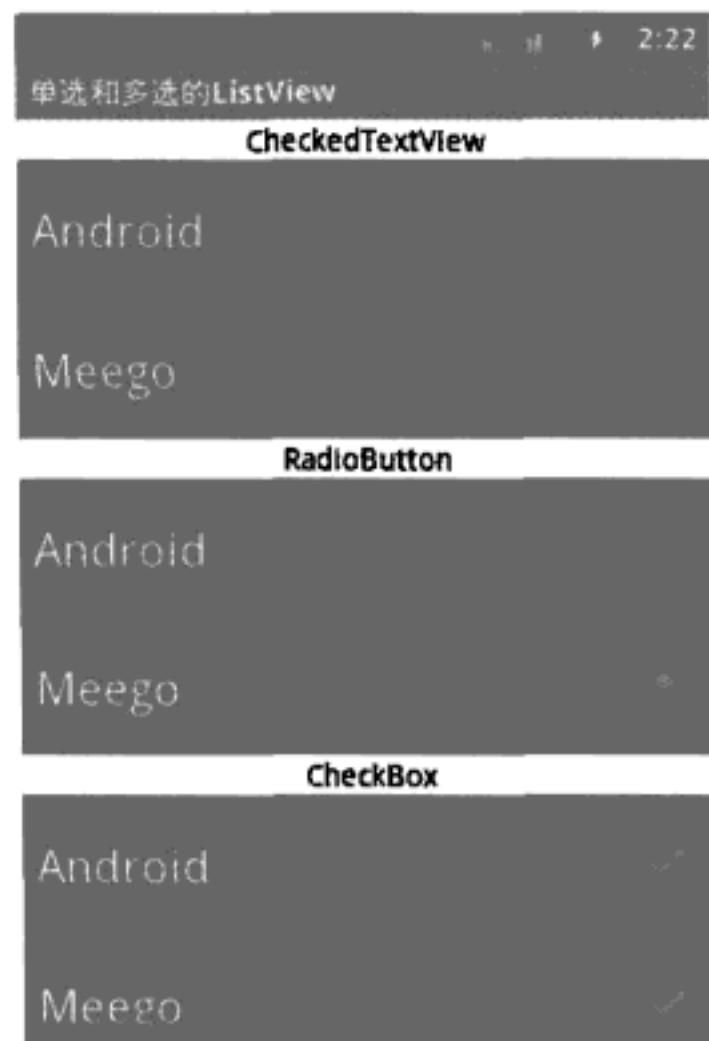
上面的代码通过 android:checkMark 属性设置了一个主题属性，从而可以使 CheckedTextView 变成拥有不同风格的选择控件。

本例在垂直方向显示了 3 个 ListView 控件，分别用来演示上述 3 个布局文件的效果。设置这 3 个 ListView 的代码如下：

```
String[] data = new String[] { "Android", "Meego" };
// CheckedTextView
ArrayAdapter<String> aaCheckedTextViewAdapter =
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_checked, data);
lvCheckedTextView.setAdapter(aaCheckedTextViewAdapter);
// 设置成单选模式
lvCheckedTextView.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
// RadioButton
ArrayAdapter<String> aaRadioButtonAdapter =
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_single_choice,
    data);
lvRadioButton.setAdapter(aaRadioButtonAdapter);
// 设置成单选模式
lvRadioButton.setChoiceMode(ListView.CHOICE_MODE_SINGLE);
// CheckBox
ArrayAdapter<String> aaCheckBoxAdapter =
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_multiple_choice,
    data);
lvCheckBox.setAdapter(aaCheckBoxAdapter);
// 设置成多选模式
lvCheckBox.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
```

如果只设置列表项的布局，在单击列表项时，相应的选项控件并不会被选中。因此，在设置列表项的布局后，还需要使用 ListView 类的 setChoiceMode 方法设置选择的模式（单选或多选）。

运行本例后，单击相应的列表项，将显示如图 5.42 所示的效果。



▲ 图 5.42 可单选和多选的 ListView 控件

### 5.8.3 对列表项进行增、删、改操作

工程目录: src\ch05\ch05\_dynamiclistview

对 ListView 控件的动态操作（添加、删除、修改列表项）往往是程序中必不可少的功能。本例中通过一个自定义的 Adapter 实现了动态向 ListView 中添加文本和图像列表项，并可以删除和修改某个被选中的列表项，以及清空所有的列表项。

编写一个自定义的 Adapter 类一般需要从 android.widget.BaseAdapter 类继承。在 BaseAdapter 类中有两个非常重要的方法：getView 和 getCount。其中 ListView 在显示某一个列表项时会调用 getView 方法来返回要显示的列表项的 View 对象。getCount 方法返回当前 ListView 控件中列表项的总数。在添加或删除列表项后，getCount 方法返回的值要进行调整，否则 ListView 可能会出现异常情况。

在本例中要向 ListView 添加两类列表项：文本列表项和图像列表项。因此，getView 方法要根据当前列表项返回 TextView 或 ImageView 对象。在添加文本列表项时直接使用 String 类型的值，添加图像列表项时使用图像资源 ID。因此，需要在自定义 Adapter 类（ViewAdapter）中编写两个方法用于添加文本和图像列表项，这两个方法的代码如下：

```
public void addText(String text)
{
    textIdList.add(text);
    notifyDataSetChanged();
}
public void addImage(int resId)
{
    textIdList.add(resId);
```

```

    notifyDataSetChanged();
}

```

在上面的代码中将文本列表项的字符串和图像列表项的资源 ID 都保存在 textIdList 变量中，该变量是一个 List 对象，定义代码如下：

```

private List textIdList = new ArrayList();

```

在添加完相应的数据后，需要使用 BaseAdapter 类的 notifyDataSetChanged 方法来通知 Adapter 对象数据已经变化，并由系统调用 getView 方法来返回相应的 View 对象。getView 方法的代码如下：

```

public View getView(int position, View convertView, ViewGroup parent)
{
    String inflater = Context.LAYOUT_INFLATER_SERVICE;
    LayoutInflator layoutInflater = (LayoutInflator) context.getSystemService(inflater);
    LinearLayout linearLayout = null;
    // 处理文本列表项
    if (textIdList.get(position) instanceof String)
    {
        // 装载 text.xml 布局文件
        linearLayout = (LinearLayout) layoutInflater.inflate(R.layout.text, null);
        TextView textView = ((TextView) linearLayout.findViewById(R.id.textview));
        textView.setText(String.valueOf(textIdList.get(position)));
    }
    // 处理图像列表项
    else if (textIdList.get(position) instanceof Integer)
    {
        // 状态 image.xml 布局文件
        linearLayout = (LinearLayout) layoutInflater.inflate(R.layout.image, null);
        ImageView imageView = (ImageView) linearLayout.findViewById(R.id.imageview);
        imageView.setImageResource(Integer.parseInt(String.valueOf(textIdList.get(position))));
    }
    return linearLayout;
}

```

在编写上面代码时应注意如下几点。

- 由于 BaseAdapter 类并不像 Activity 类有 getLayoutInflator()方法可以获得 LayoutInflator 对象，因此，需要使用 Context 类的 getSystemService 方法来获得 LayoutInflator 对象。
- 在本例中使用了两个 XML 布局文件（text.xml 和 image.xml）分别作为文本列表项和图像列表项的模板，这两个布局文件分别包含一个<TextView>和<ImageView>标签。
- 特别要注意的是 getView 方法的调用。ListView 会根据当前可视的列表项决定什么时候调用 getView 方法，调用几次 getView 方法。例如，ListView 中有 10000 个列表项，但 getView 方法并不会立刻调用 10000 次，而是根据当前屏幕上可见或即将显示的列表项调用 getView 方法，并通过 position 参数将当前列表项的位置（从 0 开始）传入 getView 方法。当然，开发人员一般不需要关心 ListView 是在什么时候调用 getView 方法的，而只需要关注于当前要返回的列表项（View 对象）即可。

- 由于文本列表项和图像列表项的数据是从 List 对象（textIdList 变量）中获得的，因此，要注意边界问题。也就是说，getCount 方法要返回正确的列表项个数，也就是 List 对象的元素个数，也可以认为 getView 方法的 position 参数值就是 List 对象中某个元素的索引。如果这时 getCount 方法返回了不正确的列表项个数（返回值比 List 对象中的元素个数还大），position 的值可能会超过 List 对象的边界，系统就会抛出异常。

在 ViewAdapter 类中除了添加列表项的方法外，还需要添加 3 个方法：remove（删除指定列表项）、removeAll（清空所有的列表项）和 modify（修改指定列表项），代码如下：

```
// 删除指定的列表项
public void remove(int index)
{
    if (index < 0) return;
    textIdList.remove(index);
    notifyDataSetChanged();
}

// 删除所有的列表项
public void removeAll()
{
    textIdList.clear();
    notifyDataSetChanged();
}

// 修改指定列表项
public void modify(int index, String text)
{
    if (index < 0)
        return;
    // 只修改文本列表项
    if (textIdList.get(index) instanceof String)
    {
        textIdList.set(index, text);
        notifyDataSetChanged();
    }
}
```

ViewAdapter 类还有一些其他的方法，这些方法的实现代码并不是很重要的，在这里并不详细解释这些代码，读者可以参阅本书提供的源代码。最后看一下 ViewAdapter 类的框架代码。

```
// ViewAdapter 为 Main 类的内嵌类
private class ViewAdapter extends BaseAdapter
{
    private Context context;
    private List textIdList = new ArrayList();
    @Override
    public View getView(int position, View convertView, ViewGroup parent){ ... ... }
    public ViewAdapter(Context context) { this.context = context; }
    @Override
    public long getItemId(int position){ return position; }
    @Override
    public int getCount(){ return textIdList.size(); }
    @Override
    public Object getItem(int position){ return textIdList.get(position); }
}
```

## Android 开发权威指南

```

public void addText(String text){ ... ... }
public void addImage(int resId) { ... ... }
public void remove(int index) { ... ... }
public void removeAll() { ... ... }
public void modify(int index, String text){... ...}
}

```

在创建完 ViewAdapter 类后，需要将该类的对象绑定到 ListView 上，代码如下：

```

lvDynamic = (ListView) findViewById(R.id.lvDynamic);
ViewAdapter viewAdapter = new ViewAdapter(this);
lvDynamic.setAdapter(viewAdapter);

```

本例在屏幕的正上方添加 5 个按钮，分别用来添加文本列表项、添加图像列表项、删除当前列表项、随机修改指定列表项和删除所有的列表项。这 5 个按钮共用同一个单击事件方法，代码如下：

```

public void onClick(View view)
{
    switch (view.getId())
    {
        // 添加文本列表项
        case R.id.btnAddText:
            int randomNum = new Random().nextInt(data.length);
            viewAdapter.addText(data[randomNum]);
            break;
        // 添加图像列表项
        case R.id.btnAddImage:
            viewAdapter.addImage(getImageResourceId());
            break;
        // 删除当前列表项
        case R.id.btnRemove:
            viewAdapter.remove(selectedIndex);
            selectedIndex = -1;
            break;
        // 修改当前列表项
        case R.id.btnModify:
            viewAdapter.modify(selectedIndex, data[new Random().nextInt(data.length)]);
            selectedIndex = -1;
        // 删除所有的列表项
        case R.id.btnRemoveAll:
            viewAdapter.removeAll();
            break;
    }
}

```

其中 data 变量为一个 String[] 对象，定义了在列表项中显示的文本集合。getImageResourceId 方法从 5 个图像资源中随机选择一个图像资源 ID 作为当前添加的列表项的图像资源 ID，代码如下：

```

private int getImageResourceId()
{
    int[] resourceIds = new int[]
    { R.drawable.item1, R.drawable.item2, R.drawable.item3, R.drawable.item4, R.drawable.
}

```

```

    item5 );
    return resourceIds[new Random().nextInt(resourceIds.length)];
}

```

运行本例后，添加一些文本和图像列表项，将显示如图 5.43 所示的效果。



▲ 图 5.43 动态添加、删除、修改列表项

#### 5.8.4 改变列表项的背景色

工程目录: src\ch05\ch05\_colorlistview

前面的章节中使用的 ListView 列表项在选中状态时背景色都是黄色的。实际上，可以将选中状态的背景色改成任意颜色，甚至是绚丽的图像。

改变列表项选中状态的背景色可以使用<ListView>标签的 android:listSelector 属性，也可以使用 ListView.setSelector 方法。例如，将背景色设为绿色的方法是先将一个绿色的 png 图 (green.png) 复制到 res\drawable 目录中，然后在<ListView>标签中设置 android:listSelector="@drawable/green"，或使用如下代码：

```

ListView listView = (ListView) findViewById(R.id.listview);
listView.setSelector(R.drawable.green);

```

在本例中有 3 个 RadioButton 控件，分别将列表项选中状态的背景颜色设置成默认颜色、绿色和光谱颜色。这 3 个 RadioButton 控件共享一个单击事件方法，代码如下：

```

public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.rbdefault:
            // 设置成默认背景颜色

```

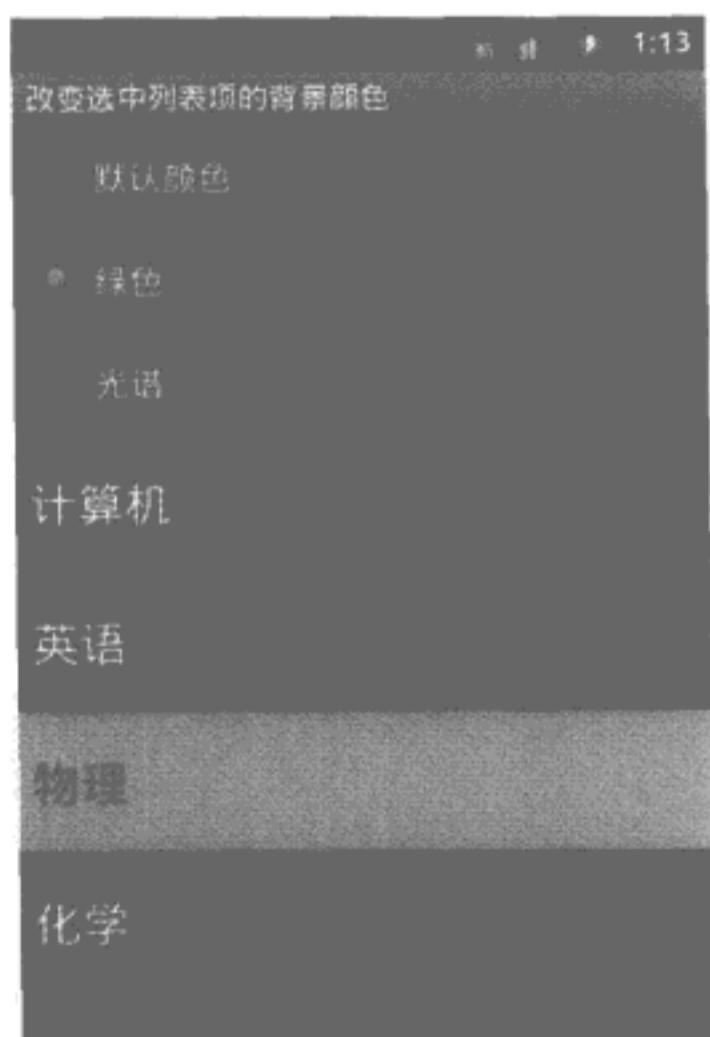
```

        listView.setSelector(defaultSelector);
        break;
    case R.id.rbGreen:
        // 设置绿色背景
        listView.setSelector(R.drawable.green);
        break;
    case R.id.rbSpectrum:
        // 设置光谱背景
        listView.setSelector(R.drawable.spectrum);
        break;
    }
}

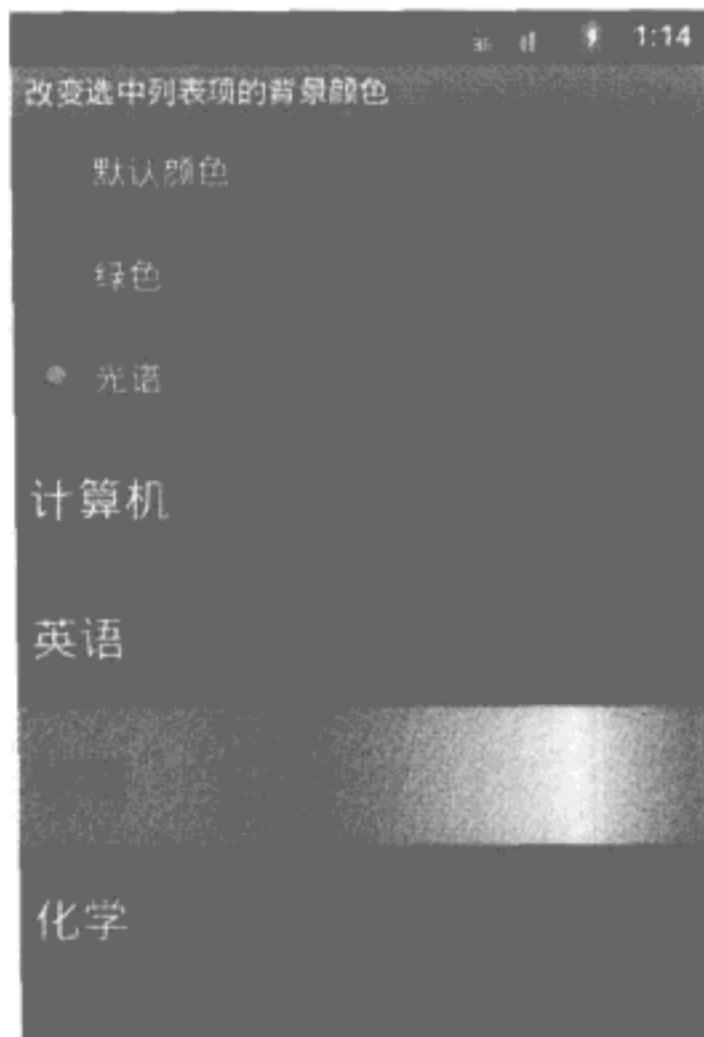
```

在上面代码中的 `defaultSelector` 是 `Drawable` 类型变量，该变量表示列表项被选中状态默认的背景颜色，通过 `ListView.getSelector` 方法可获得该值。

运行本例后，分别单击“绿色”和“光谱”选项按钮，将显示如图 5.44 和图 5.45 所示的效果。



▲ 图 5.44 设置绿色背景



▲ 图 5.45 设置光谱背景

### 5.8.5 ListActivity (封装 ListView 的 Activity)

`ListActivity` 实际上是 `ListView` 和 `Activity` 的结合体，也就是说，一个 `ListActivity` 就是内嵌一个 `ListView` 控件的 `Activity`。在 `ListActivity` 类的内部通过代码来创建 `ListView` 对象，因此，使用 `ListActivity` 并不需要使用布局文件来定义 `ListView` 控件。

如果在某些 `Activity` 中只包含一个 `ListView`，使用 `ListActivity` 是非常方便的。可以通过 `ListActivity` 类的 `setListAdapter` 方法来设置 `Adapter` 对象。该方法相当于调用了 `ListView` 类的 `setAdapter` 方法。

也可以通过 `ListActivity.getListView` 方法获得当前 `ListActivity` 的 `ListView` 对象，并像操作普通



的 ListView 对象一样操作 ListActivity 中的 ListView 对象。

### 5.8.6 ExpandableListView(可扩展的列表控件)

工程目录: src\ch05\ch05\_expandableListview

Android SDK 提供了一个可以展开的 ListView 控件 ExpandableListView。与菜单和子菜单类似, ExpandableListView 的列表项分为组列表项和子列表项, 单击组列表项(包含子列表项的列表项)后, 会显示当前列表项下的所有子列表项。

ExpandableListView 是 ListView 的直接子类, 因此, ExpandableListView 拥有 ListView 的一切特性。当然, 与 ListView 一样, ExpandableListView 类也有一个与之对应的 ExpandableListActivity 类, 该类包含一个 ExpandableListView 控件, 如果 Activity 上只有一个 ExpandableListView 控件, 建议直接使用 ExpandableListActivity 类来代替 Activity 类。

本节将使用 ExpandableListActivity 类来创建 ExpandableListView 对象, 并添加几个列表项和相应的子列表项。ExpandableListView 的用法与 ExpandableListActivity 非常相似, 读者可参考本例提供的代码来使用 ExpandableListView 控件。

与 ListActivity 一样, ExpandableListActivity 类也需要一个 Adapter 类。在本例中使用了一个定制的 Adapter 类 (MyExpandableListAdapter), 该类继承自 BaseExpandableListAdapter 类。在 MyExpandableListAdapter 类中有两个核心方法: getGroupView 和 getChildView。这两个方法分别用来返回列表项和子列表项的 View 对象, 代码如下:

```
public View getGroupView(int groupPosition, boolean isExpanded, View convertView,
ViewGroup parent)
{
    TextView textView = getGenericView();
    // 获得并设置列表项的文本, getGroup 方法从一个一维数组中获得相应的字符串
    textView.setText(getGroup(groupPosition).toString());
    return textView;
}
public View getChildView(int groupPosition, int childPosition,
    boolean isLastChild, View convertView, ViewGroup parent)
{
    TextView textView = getGenericView();
    // 获得并设置子列表项的文本, getChild 方法从一个二维数组中获得相应的字符串
    textView.setText(getChild(groupPosition, childPosition).toString());
    return textView;
}
```

在上面代码中使用了一个 getGenericView 方法, 在该方法中创建了一个 TextView 对象, 并设置了相应的属性, 代码如下:

```
public TextView getGenericView()
{
    AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
        ViewGroup.LayoutParams.FILL_PARENT, 64);
    TextView textView = new TextView(Main.this);
    textView.setLayoutParams(lp);
```

```

    textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
    textView.setPadding(36, 0, 0, 0);
    textView.setTextSize(20);
    return textView;
}

```

**ExpandableListActivity** 类也需要使用 `setListAdapter` 方法指定 Adapter 对象，代码如下：

```

ExpandableListAdapter adapter = new MyExpandableListAdapter();
setListAdapter(adapter);

```

当单击子列表项时会弹出一个菜单，因此，需要在 `onCreate` 方法中使用下面的代码将上下文菜单注册到 `ExpandableListView` 上。

```

registerForContextMenu(getExpandableListView());

```

在本例中，与上下文菜单相关的事件方法是 `onCreateContextMenu` 和 `onContextItemSelected`，当单击子列表项时系统会调用 `onCreateContextMenu` 方法创建弹出菜单，单击菜单项时系统会调用 `onContextItemSelected` 方法。这两个方法的实现代码如下：

```

// 创建上下文菜单
@Override
public void onCreateContextMenu(ContextMenu menu, View view,
        ContextMenuInfo menuInfo)
{
    ExpandableListContextMenuInfo info = (ExpandableListContextMenuInfo) menuInfo;
    // 获得当前列表项的类型
    int type = ExpandableListView.getPackedPositionType(info.packedPosition);
    // 获得当前列表项的文本
    String title = ((TextView) info.targetView).getText().toString();
    // 单击子菜单项时，弹出上下文菜单
    if (type == ExpandableListView.PACKED_POSITION_TYPE_CHILD)
    {
        menu.setHeaderTitle("弹出菜单");
        menu.add(0, 0, 0, title);
    }
}
// 响应菜单项单击事件
@Override
public boolean onContextItemSelected(MenuItem item)
{
    ExpandableListContextMenuInfo info = (ExpandableListContextMenuInfo) item
        .getMenuInfo();
    String title = ((TextView) info.targetView).getText().toString();
    Toast.makeText(this, title, Toast.LENGTH_SHORT).show();
    return true;
}

```

运行本节的例子后，长按第 1 个列表项“辽宁”的第 1 个子列表项“沈阳”，将显示如图 5.46 所示的效果。



▲ 图 5.46 可展开的 ListView

### 5.8.7 Spinner（下拉列表控件）

工程目录: src\ch05\ch05\_spinner

Spinner 控件用于显示一个下拉列表。该控件的用法与 ListView 控件类似，在装载数据时也需要创建一个 Adapter 对象，并在创建 Adapter 对象的过程中指定要装载的数据（数组或 List 对象）。例如，下面的代码分别使用 ArrayAdapter 和 SimpleAdapter 对象向两个 Spinner 控件添加数据。

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 开始创建第一个 Spinner 对象
    Spinner spinner1 = (Spinner) findViewById(R.id.spinner1);
    String[] mobileOS = new String[]
    { "Android", "IPhone", "Symbian", "Meego", "Window Phone7" };
    ArrayAdapter<String> aaAdapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, mobileOS);
    // 为第一个 Spinner 设置 Adapter 对象
    spinner1.setAdapter(aaAdapter);
    // 开始创建第二个 Spinner 对象
    Spinner spinner2 = (Spinner) findViewById(R.id.spinner2);
    // 第二个 Spinner 中的数据是一个 Map 对象的集合
    final List<Map<String, Object>> items = new ArrayList<Map<String, Object>>();
    Map<String, Object> item1 = new HashMap<String, Object>();
    item1.put("ivLogo", R.drawable.calendar);
    item1.put("tvApplicationName", "多功能日历");
    Map<String, Object> item2 = new HashMap<String, Object>();
    item2.put("ivLogo", R.drawable.eoemarket);
    item2.put("tvApplicationName", "eoMarket 客户端");
```

## Android 开发权威指南

```

        items.add(item1);
        items.add(item2);
SimpleAdapter simpleAdapter = new SimpleAdapter(this, items,
        R.layout.item, new String[]
        { "ivLogo", "tvApplicationName" }, new int[]
        { R.id.ivLogo, R.id.tvApplicationName });
// 为第二个 Spinner 设置 Adapter 对象
spinner2.setAdapter(simpleAdapter);
// 设置第二个 Spinner 的列表项选择事件，当选中某个列表项时，会在 Activity 的标题栏显示当前
// 列表项的文本内容
spinner2.setOnItemSelectedListener(new OnItemSelectedListener()
{
    @Override
    public void onItemSelected(AdapterView<?> parent, View view,
                               int position, long id)
    {
        setTitle(items.get(position).get("tvApplicationName").toString());
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent)
    {
    }
});
}
}

```

在设置第二个 Spinner 的 Adapter 对象时使用了一个 item.xml 布局文件（资源 ID 为 R.layout.item），该布局文件的内容如下：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <ImageView android:id="@+id/ivLogo" android:layout_width="60dp"
        android:layout_height="60dp" android:src="@drawable/icon"
        android:paddingLeft="10dp" />
    <TextView android:id="@+id/tvApplicationName" android:textColor="#000"
        android:layout_width="wrap_content" android:layout_height="fill_parent"
        android:textSize="16dp" android:gravity="center_vertical"
        android:paddingLeft="10dp" />
</LinearLayout>

```

在 item.xml 文件中定义了两个控件：<ImageView>和<TextView>。ID 分别为 ivLogo 和 tvApplicationName。由于 item.xml 是用于列表项的布局。也就是说，每一个列表项由一个图像和一个文本组成。那么在为 Spinner 设置数据时（Adapter 对象），就不能只设置一个文本了，需要同时指定每一个列表项要显示的图像资源 ID 和文本，所以在本例中将图像资源 ID 和文本放到 Map 对象中（每一个列表项的数据对应一个 Map 对象），而 Map 对象的 key 就是控件标签的 ID 值，这样系统就可以利用 Map 对象找到控件要显示的数据了。

运行本节的例子后，单击第 1 个和第 2 个 Spinner 控件右侧的下拉按钮，将显示如图 5.47 和图 5.48 所示的效果。



▲图 5.47 只显示文本的下拉列表框



▲图 5.48 带文本和图像的下拉列表框

### 扩展学习：SimpleAdapter 类。

在本例中使用了一个 SimpleAdapter 类来处理 Spinner 显示的数据，这个类可以将任何自定义的 XML 布局文件作为列表项来使用。我们先来看看 SimpleAdapter 类的构造方法的定义。

```
public SimpleAdapter(Context context, List<? extends Map<String, ?>> data, int resource,
String[] from, int[] to)
```

其中第 1 个参数 context 不必多说了，这个参数在前面已经多次提到过了，一般在 Activity 的子类中使用 this 作为该参数的值。现在需要着重说的是后 4 个参数。

data 是一个 List 类型的参数，而 List 对象的元素类型是一个 Map<String, ?>类型。我们可以回顾一下本例的列表项布局文件 item.xml 的内容。在该布局文件中定义了两个控件：ImageView 和 TextView，每一个列表项都要根据不同的情况设置 ImageView 的图像和 TextView 的文本。假设要添加两个列表项，就意味着需要设置 4 个值（每个列表项 2 个值）。每个列表项的值可以用一个 Map 对象来表示。key 表示相应控件的 id 变量名（在本例中是 ivLogo 和 tvApplicationName，注意，不是 ID 值，而是 R.id 类中的变量名），value 表示具体的值。在本例中，需要使用如下代码来设置这两个列表项的值：

```
Map<String, Object> item1 = new HashMap<String, Object>();
// 设置第 1 个列表项的数据
item1.put("ivLogo", R.drawable.calendar);
item1.put("ivApplicationName", "多功能日历");
Map<String, Object> item2 = new HashMap<String, Object>();
// 设置第 2 个列表项的数据
item2.put("ivLogo", R.drawable.eoemarket);
item2.put("ivApplicationName", "eoemarket 客户端");
List<Map<String, Object>> data = new ArrayList<Map<String, Object>>();
// 将两个 Map 对象添加到 List 对象中，该对象就是 SimpleAdapter 构造方法的第 2 个参数值
data.add(item1);
data.add(item2);
```

从上面的代码很容易知道 data 参数表示所有列表项的数据，List 对象的元素（Map 对象）

表示列表项的数据。

`SimpleAdapter` 类的构造方法的第 3 个参数 `resource` 表示列表项模板的资源 ID，在本例中是 `R.layout.item`。`from` 和 `to` 参数分别表示布局文件（`item.xml`）中控件标签在 `R.id` 类中的变量名（由于从 `android:id` 属性中获得的只是一个 `int` 类型的值，而不是变量名，所以要使用 `from` 指定相应的变量名）及 `android:id` 属性值。在本例中使用如下代码设置这两个参数的值：

```
String[] from = new String[] { "ivLogo", "tvApplicationName" };
int[] to = new int[] { R.id.ivLogo, R.id.tvApplicationName };
```

`from` 和 `to` 数组设置的控件顺序要一致，也就是说，`from` 的第 `n` 个元素要对应于 `to` 的第 `n` 个元素，但 `from` 和 `to` 数组的顺序可以和 `data` 参数中设置列表项的顺序不一致（只要在 `Map` 对象中能根据 `key` 查找到相应的 `value` 即可）。

在最后需要向 `List` 对象中添加 `SimpleAdapter` 所需的数据，并使用 `SimpleAdapter` 对象来为列表控件提供数据，代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    List<Map<String, Object>> appItems = new ArrayList<Map<String, Object>>();
    // 设置 data 参数的值，其中 resIds 和 applicationNames 保存列表项中相应组件的值
    for (int i = 0; i < applicationNames.length; i++)
    {
        Map<String, Object> appItem = new HashMap<String, Object>();
        appItem.put("ivLogo", resIds[i]);
        appItem.put("tvApplicationName", applicationNames[i]);
        appItems.add(appItem);
    }
    SimpleAdapter simpleAdapter = new SimpleAdapter(this, appItems,
        R.layout.main, new String[] { "tvApplicationName", "ivLogo" },
        new int[] { R.id.tvApplicationName, R.id.ivLogo });
    setListAdapter(simpleAdapter);
}
```

## 5.9 滚动控件

本节将介绍 Android SDK 中提供的几个可以使其中内容滚动的控件，这些控件包括 `ScrollView`（垂直滚动控件）、`HorizontalScrollView`（水平滚动控件）、`Gallery`（画廊控件）。

### 5.9.1 ScrollView（垂直滚动控件）

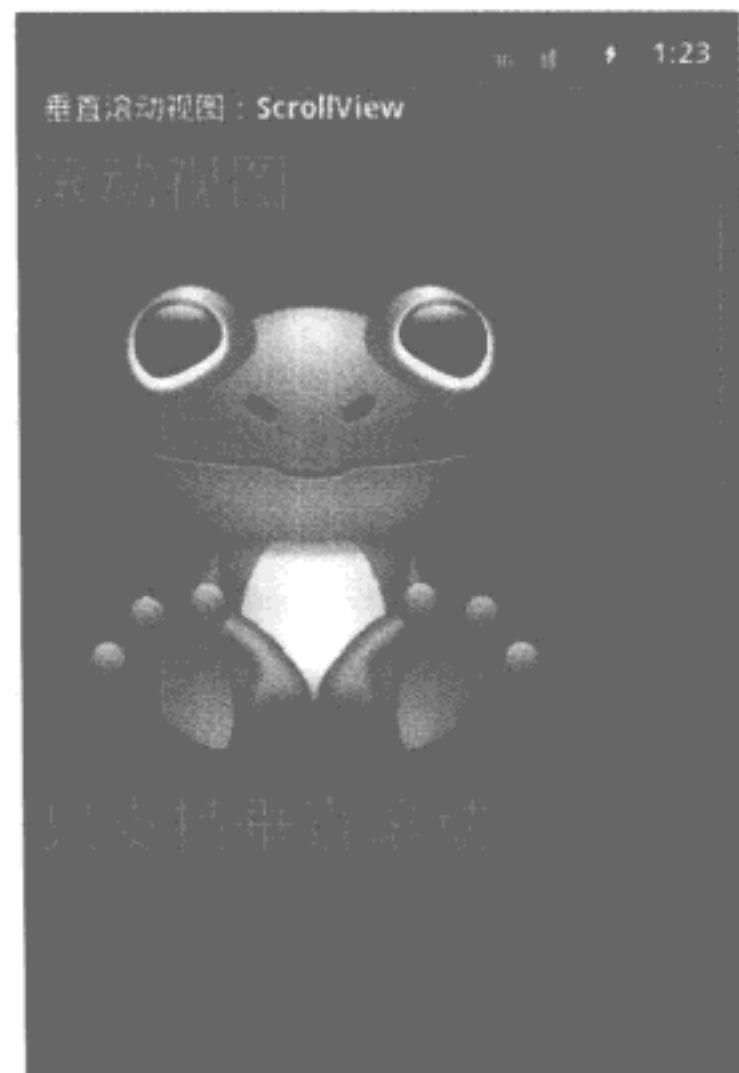
工程目录：src\ch05\ch05\_scrollview

`ScrollView` 控件只支持垂直滚动，而且在 `ScrollView` 中只能包含一个控件。通常在 `<ScrollView>` 标签中定义一个 `<LinearLayout>` 标签，并且将 `<LinearLayout>` 标签的 `android:orientation` 属性值设为 `vertical`，然后在 `<LinearLayout>` 标签中放置多个控件。如果 `<LinearLayout>` 标签中的控件所占用的总高度超过屏幕的高度，就会在屏幕右侧出现一个滚动条。通过单击手机的上、下按钮或上下拖动屏幕可以滚动视图来查看未显示的部分。

在本例的布局文件中配置了一些 TextView 和 ImageView 控件，由于控件的高度超过了屏幕的高度，因此，会在屏幕的右侧出现一个滚动条。布局文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="滚动视图"
            android:textSize="30dp" />
        <ImageView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/item1" />
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="只支持垂直滚动"
            android:textSize="30dp" />
        <ImageView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/item2" />
        <ImageView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/item3" />
    </LinearLayout>
</ScrollView>
```

运行本节的例子后，将显示如图 5.49 所示的效果。



▲ 图 5.49 ScrollView 控件

### 5.9.2 HorizontalScrollView (水平滚动控件)

工程目录：src\ch05\ch05\_horizontalscrollview

HorizontalScrollView 控件支持水平滚动，用法与 ScrollView 控件非常类似。在本例中仍然使用 5.9.1 节例子中使用的 5 个控件，只是将<ScrollView>改成<HorizontalScrollView>，将<LinearLayout>

的 android:orientation 属性值改成 horizontal，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<HorizontalScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="fill_parent">
        <!-- 省略了控件的定义 -->
        ...
    </LinearLayout>
</HorizontalScrollView>
```

运行本节的例子后，将显示如图 5.50 所示的效果。



▲ 图 5.50 HorizontalScrollView 控件

### 5.9.3 可垂直和水平滚动的视图

工程目录: src\ch05\ch05\_bothscrollview

如果将 ScrollView 和 HorizontalScrollView 控件结合使用，就可以实现垂直和水平滚动的效果。所谓结合，就是指在 <ScrollView> 标签中使用 <HorizontalScrollView> 标签，或在 <HorizontalScrollView> 标签中使用 <ScrollView> 标签，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <HorizontalScrollView android:layout_width="fill_parent" android:layout_height="wrap_content">
        <RelativeLayout android:orientation="horizontal"
            android:layout_width="fill_parent" android:layout_height="fill_parent">
            <!-- 此处省略组件的配置 -->
            ...
        </RelativeLayout>
    </HorizontalScrollView>
</ScrollView>
```

```
</RelativeLayout>
</HorizontalScrollView>
</ScrollView>
```

运行本例后，将显示如图 5.51 所示的效果。



▲ 图 5.51 可垂直和水平滚动的视图

#### 5.9.4 Gallery (画廊控件)

工程目录: src\ch05\ch05\_gallery

Gallery 控件一般用于显示图像列表，因此，也可称为画廊控件。Gallery 只能水平显示一行，而且支持水平滑动效果。也就是说，单击、选中或拖动 Gallery 中的图像，Gallery 中的图像列表会根据不同的情况向左或向右移动，直到显示到最后一个图像为止。因此，Gallery 和 ScrollView、HorizontalScrollView 类似，都属于可以使其内容滚动或滑动的控件。

一般情况下，Gallery 只能显示指定数目的图像（也可以是其他视图控件）。如果要想使 Gallery 循环显示图像，也就是说，当显示到最后一个图像时，会继续显示图像列表中的第 1 个图像，达到循环显示的效果。要想达到这个目的并不困难，只需要“欺骗”一下 Adapter 对象即可。

从前面章节的内容可以知道，BaseAdapter 类中的 getView 方法的调用与 getCount 方法的返回值有关。如果 getCount 方法返回 n，那么 getView 方法中的 position 参数值是绝不会大于 n - 1 的。因此，可以使 getCount 方法返回一个很大的数，例如，Integer.MAX\_VALUE。这样系统就会认为 ImageAdapter 对象中有非常多的 View 对象（Integer.MAX\_VALUE 的值超过 20 亿，可以认为是接近无穷大）。

这样做还会带来另外一个问题，如果 getCount 方法返回了一个很大的数，那么 position 参数的值也会很大，在这种情况下，如何根据这个 position 参数值获得相应的图像 ID 资源呢？不会有人去创建 Integer.MAX\_VALUE 大小的数组吧？当然，解决方法也很简单。假设有一个 resIds 数组（长

度为 15) 保存了 15 个图像资源 ID, 现在要使 Gallery 循环显示这 15 个图像, 如果 position 的值超过了 14, 可以使用取余的方法来循环取这个数组的值, 代码如下:

```
int imageResId = resIds[position % resIds.length];
```

下面还有一件重要事情要做, 就是设置 Gallery 中每个图像的显示风格, 首先需要获得图像背景的资源 ID。在 ImageAdapter 类的构造方法中编写如下代码:

```
TypedArray typedArray = obtainStyledAttributes(R.styleable.Gallery);
mGalleryItemBackground = typedArray.getResourceId(
    R.styleable.Gallery_android_galleryItemBackgrounds, 0);
```

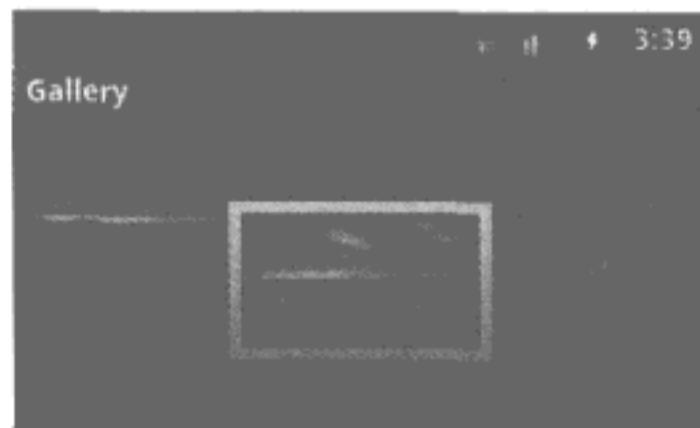
其中 R.styleable.Gallery 是 res/values/attrs.xml 文件中一个属性的资源 ID, 代码如下:

```
<declare-styleable name="Gallery">
    <attr name="android:galleryItemBackground" />
</declare-styleable>
```

使用<declare-styleable>标签声明的 style 在引用时会在 R.styleable 类中自动生成静态变量名是<declare-styleable>和<attr>标签的 name 属性值的组合, 中间用下划线连接。

在 getView 方法中需要设置 ImageView 控件的显示风格和图像资源, 代码如下:

```
public View getView(int position, View convertView, ViewGroup parent)
{
    ImageView imageView = new ImageView(mContext);
    // 通过取余的方式获得图像的资源 ID
    imageView.setImageResource(resIds[position % resIds.length]);
    imageView.setScaleType(ImageView.ScaleType.FIT_XY);
    imageView.setLayoutParams(new Gallery.LayoutParams(136, 88));
    imageView.setBackgroundResource(mGalleryItemBackground);
    return imageView;
}
```



▲ 图 5.52 循环显示图像的 Gallery 控件

## 5.10 ImageSwitcher ( 图像切换控件 )

**工程目录:** src\ch05\ch05\_imageswitcher

ImageSwitcher 控件可用于在不同图像之间切换, 其中切换的过程可以采用动画的方式 (如淡入淡出效果)。

ImageSwitcher 需要一个视图工厂 (ViewFactory) 来创建用于显示图像的 ImageView 对象。因



此，我们需要一个实现 `android.widget.ViewSwitcher.ViewFactory` 接口的类（在本例中 `Main` 类实现了 `ViewFactory` 接口）。`ViewFactory` 接口有一个 `makeView` 方法，该方法与 `BaseAdapter` 类中的 `getView` 方法类似，都是在控件显示数据时用于创建显示内容的视图对象，对于 `ImageSwitcher` 控件就是创建 `ImageView` 对象，对于 `BaseAdapter` 可以创建任何的视图对象。

下面先来看一下 `makeView` 方法的代码。

```
@Override
public View makeView()
{
    ImageView imageView = new ImageView(this);
    imageView.setBackgroundColor(0xFF000000);
    imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
    imageView.setLayoutParams(new ImageSwitcher.LayoutParams(
        LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
    return imageView;
}
```

在 `makeView` 方法中通过代码方式创建了一个 `ImageView` 对象，并设置了相应的属性值。接下来需要从布局文件中创建 `ImageSwitcher` 对象，并设置相应的属性。

```
imageSwitcher = (ImageSwitcher) findViewById(R.id.imageswitcher);
// 设置视图工厂，也就是实现 ViewFactory 接口的类的对象实例
imageSwitcher.setFactory(this);
// 设置切换图像的淡入效果
imageSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
    android.R.anim.fade_in));
// 设置切换图像的淡出效果
imageSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
    android.R.anim.fade_out));
```

上面代码中使用了一个 `imageswitcher.xml` 布局文件，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="wrap_content">
        <Button android:id="@+id/button1" android:orientation="horizontal"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="图像 1" />
        <Button android:id="@+id/button2" android:orientation="horizontal"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="图像 2" />
        <Button android:id="@+id/button3" android:orientation="horizontal"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:text="图像 3" />
    </LinearLayout>
    <ImageSwitcher android:id="@+id/imageswitcher"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:layout_marginTop="30dp" />
</LinearLayout>
```

在 imageswitcher.xml 文件中定义了 3 个按钮用于切换 3 个图像，按钮的单击事件处理代码如下：

```

@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.button1:
            // ImageSwitcher 切换到第一张图
            imageSwitcher.setImageResource(R.drawable.item1);
            break;
        case R.id.button2:
            // ImageSwitcher 切换到第二张图
            imageSwitcher.setImageResource(R.drawable.item2);
            break;
        case R.id.button3:
            // ImageSwitcher 切换到第三张图
            imageSwitcher.setImageResource(R.drawable.item3);
            break;
    }
}

```



本例的显示效果如图 5.53 所示。

图 5.53 ImageSwitcher 控件

## 5.11 GridView（网格控件）

工程目录：src\ch05\ch05\_gridview

从名字很容易看出，GridView 控件用于显示一个网格。实际上，GridView 与前面讲的 ListView、Spinner 等控件的使用方法类似，只是 GridView 在显示方式上有所不同。GridView 控件采用了二维表的方式来显示列表项（也可称为单元格），每一个单元格是一个 View 对象，在单元格上可以放置任何 Android SDK 支持的控件。

既然 GridView 采用了二维表的方式显示单元格，就需要设置二维表的行和列。设置 GridView 的列可以使用<GridView>标签的 columnWidth 属性，也可以使用 GridView 类的 setColumnWidth 方法设置列数。GridView 中的单元格会根据列数自动折行显示，因此，并不需要设置 GridView 的行数，但需要使用 android:numColumns 属性设置网格的列数，否则 GridView 只会显示一列。

在本例中使用了 SimpleAdapter 对象来指定 GridView 中每个单元格的数据（图像的资源 ID），在 GridView 的下方显示一个 ImageView 控件，当选中或单击某个单元格后，该单元格中的图像将被放大显示在这个 ImageView 控件中。下面是本例中的核心代码，在这些代码中创建了 SimpleAdapter 对象，并使用 GridView 类的 setAdapter 方法指定这个 SimpleAdapter 对象。

```

GridView gridView = (GridView) findViewById(R.id.gridview);
List<Map<String, Object>> cells = new ArrayList<Map<String, Object>>();
// resIds 是一个 int[] 类型变量，保存了显示在 GridView 中的图像资源 ID
// 每一个单元格都是一个 ImageView 组件，android:id 属性值是 imageview
for (int i = 0; i < resIds.length; i++)
{

```

```
Map<String, Object> cell = new HashMap<String, Object>();
cell.put("imageview", resIds[i]);
cells.add(cell);
}
SimpleAdapter simpleAdapter = new SimpleAdapter(this, cells,R.layout.cell, new String[]
{ "imageview" }, new int[]{ R.id.imageview });
gridView.setAdapter(simpleAdapter);
```

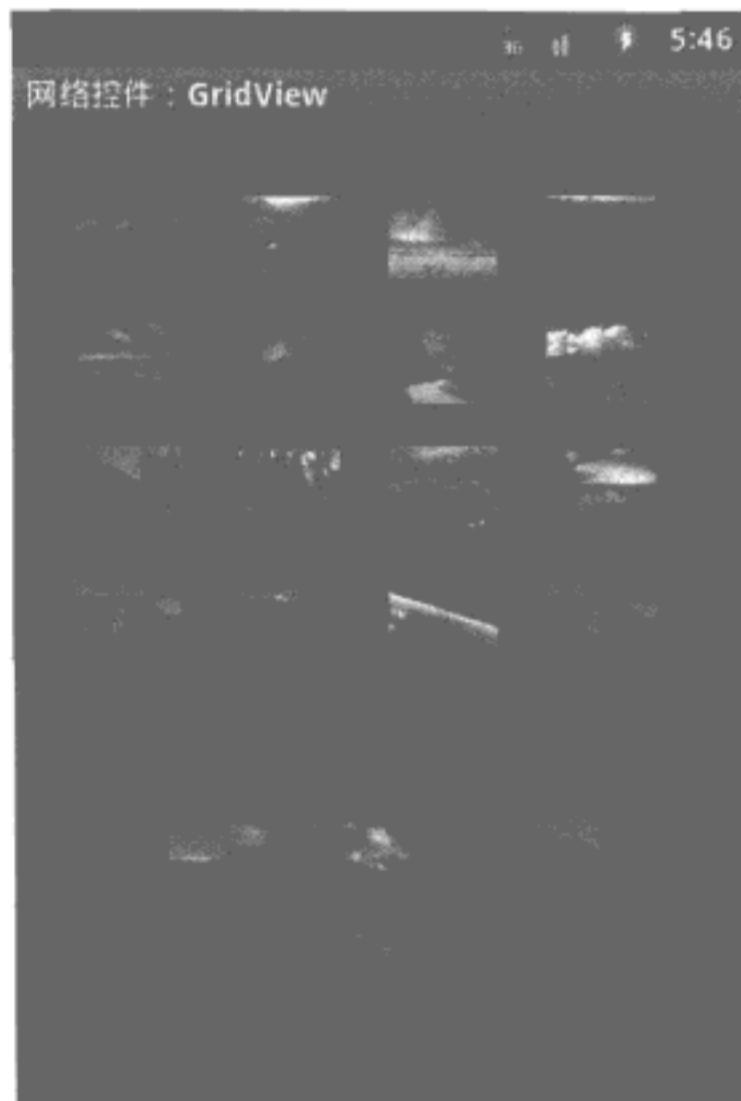
定义 GridView 控件的 gridview.xml 布局文件的代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="center_horizontal">
    <GridView android:id="@+id/gridview" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:numColumns="4"
        android:padding="20dp" android:horizontalSpacing="6dp"
        android:verticalSpacing="6dp" />
    <ImageView android:id="@+id/imageview" android:layout_width="fill_parent"
        android:layout_height="150dp" />
</LinearLayout>
```

当单击或选中 GridView 中的单元格后，将分别调用相应的事件方法，并在这些方法中执行如下代码来切换 ImageView 控件中的图像：

```
imageView.setImageResource(resIds[position]);
```

运行本节的例子后，选中或单击屏幕上方单元格中的图像，将在屏幕下方的 ImageView 控件中放大显示单元格中的图像，效果如图 5.54 所示。



▲ 图 5.54 GridView 控件

## 5.12 TabHost（标签控件）

工程目录: src\ch05\ch05\_tab

如果屏幕上需要放置很多控件，可能一屏放不下，除了使用滚动视图的方式外，还可以使用标签控件对屏幕进行分页显示。当单击标签控件的不同标签时，会显示当前标签的内容。在 Android 系统中每一个标签可以显示一个 View 或一个 Activity。

TabHost 是标签控件的核心类，也是标签的集合。每一个标签是 TabHost.TabSpec 对象。通过 TabHost 类的 addTab 方法可以添加多个 TabHost.TabSpec 对象。如果从布局文件中添加 View，首先需要建立一个布局文件，并且根节点要使用<FrameLayout>或<TabHost>标签。

在本例中建立了 3 个标签。在第 1 个标签中显示了一个 View，在 View 中有两个控件：Button 和 ImageView。另两个标签分别显示两个 Activity。布局文件的内容如下：

### main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent">
    <Button android:id="@+id/button" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="切换到第 3 个标签" />
</FrameLayout>
```

在创建 TabHost 对象时一般使用从 TabActivity 继承的类，在该类的 onCreate 方法中添加 3 个标签，代码如下：

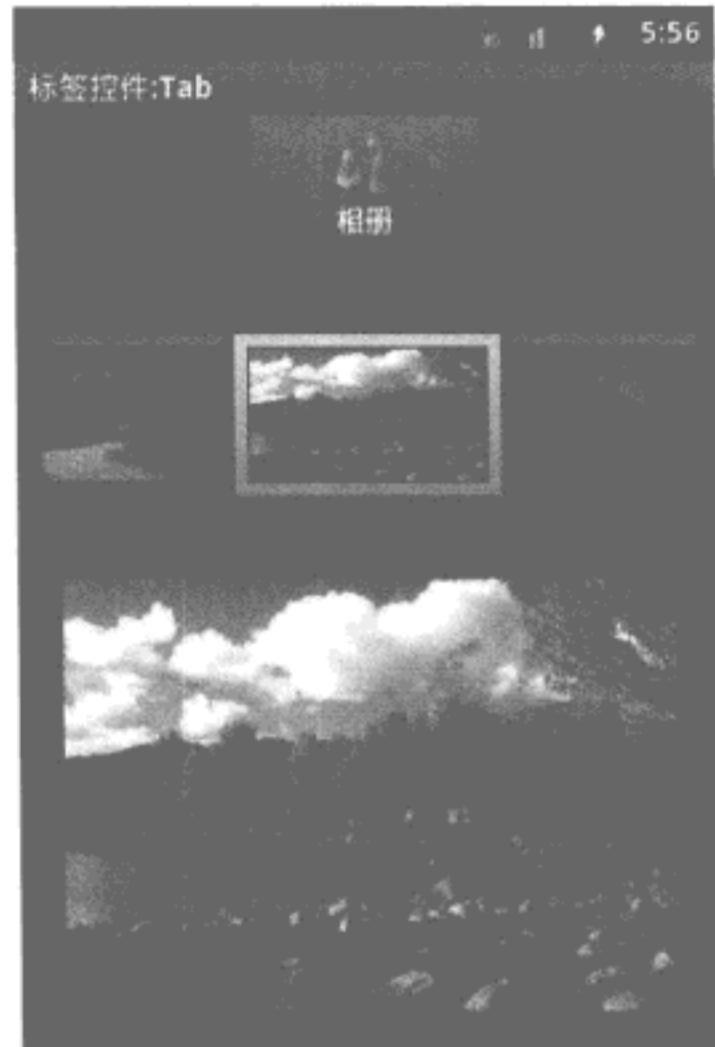
```
// 通过 TabActivity 类的 getTabHost 方法获得 TabHost 对象
TabHost tabHost = getTabHost();
// 装载 main.xml 布局文件，也就是上面给出的 XML 布局文件
LayoutInflater.from(this).inflate(R.layout.main, tabHost.getTabContentView(), true);
// 添加第 1 个标签，显示视图（按钮和 ImageView）
tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("切换标签").setContent(R.id.button));
// 添加第 2 个标签，在标签页上显示一个图像，并在该页中显示 GalleryActivity
tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator("相册")
    .getResources().getDrawable(R.drawable.icon1))
    .setContent(new Intent(this, GalleryActivity.class)));
// 添加第 3 个标签，在该标签中显示 RatingListView
tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("评分")
    .setContent(new Intent(this, RatingListView.class))));
```

在上面的代码中通过 TabHost.newTabSpec 方法创建了 TabSpecs 对象。newTabSpec 方法的参数表示标签的字符串标识。也就是说，通过该标识可以获得相应的标签。在单击第 1 个标签中的按钮后，可以切换到第 3 个标签。要完成这个功能可以使用标签的索引，也可以使用通过 newTabSpec 方法设置的标识。切换到第 3 个标签的代码如下：

```
// 标签索引从 0 开始
```

```
getTabHost().setCurrentTab(2);
// 或采用如下代码
// getTabHost().setCurrentTabByTag("tab3");
```

运行本节的例子后，切换到第2个标签的效果如图5.55所示。



▲ 图 5.55 切换到第 2 个标签页

## 5.13 ViewStub（惰性装载控件）

工程目录：src\ch05\ch05\_viewstub

4.5.6节介绍过一个<include>标签，该标签可以在布局文件中引用另外一个布局文件，并可以覆盖被引用布局文件根节点所有与布局相关的属性，也就是以 android:layout 开头的属性。通过<include>标签可以将一个非常庞大的布局文件分解成若干个较小的布局文件，而且这些小的布局文件也可以被多次引用，从而达到一个重用的目的。

<include>标签固然很好用，但有一个问题，就是布局文件中的控件并不一定在程序启动时全都用到，有一些控件只在特定的情况下才会被使用到。例如，一个阅读图书的软件只有在下载电子书时才需要显示进度条，在平时看书时都是装载的本地电子书，并不需要使用进度条。因此，在程序启动时完全可以先不加载这个进度条，但使用<include>标签引用这个包含进度条的布局文件时，不管三七二十一，将所有的控件全部装载到了内存中。也许有的读者会说，一个进度条占用不了多少系统资源，都装载也无所谓。这些读者也许是正确的，但如果装载的不是进度条，而是很多 ImageView 控件（显示了很大的图像），并且还不是在一个地方装载，那恐怕就会将可怜的手机资源消耗殆尽了。因此，我们急需一种机制来改变<include>标签的这种行为，只在需要时装载控件，这种机制就是本节要介绍的 ViewStub 控件。

ViewStub 是不可视的控件，它的作用与<include>标签基本相同，在布局文件中使用<ViewStub>

标签来引用其他的布局文件。但与<include>唯一的不同是 ViewStub 并不会马上装载引用的布局文件，只有在调用了 ViewStub.inflate 或 ViewStub.setVisibility(View.VISIBLE)方法后，ViewStub 才会装载引用的控件。下面先看两个布局文件。

### main.xml

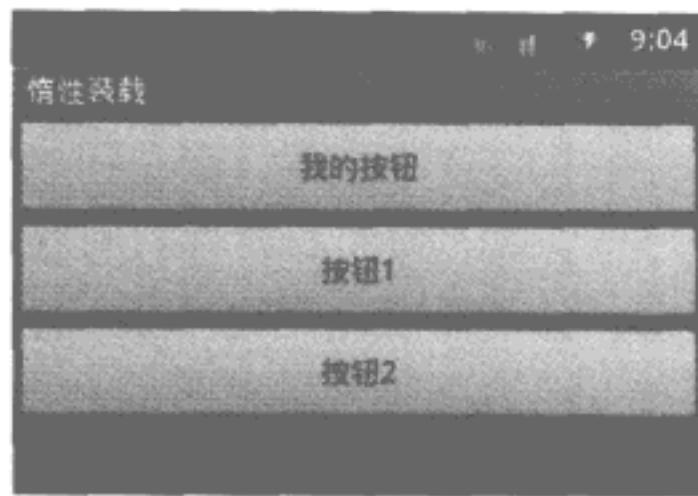
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="我的按钮"
        android:onClick="onClick_Button" />
    <include layout="@layout/custom" />
</LinearLayout>
```

### custom.xml

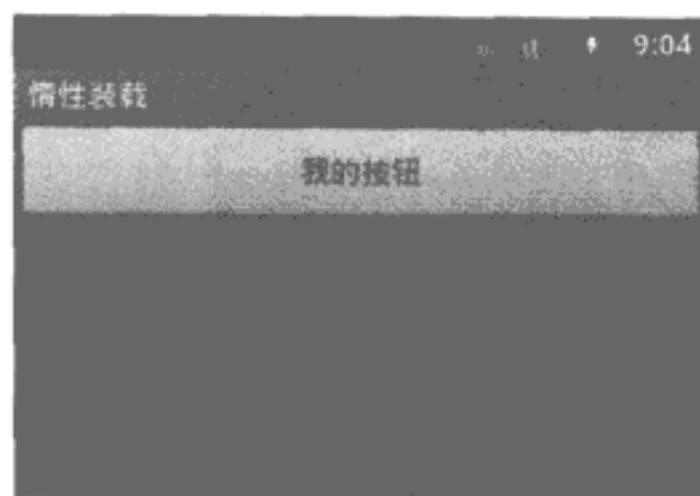
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="按钮 1" />
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="按钮 2" />
</LinearLayout>
```

在 main.xml 文件中使用了<include>标签来引用 custom.xml，在这种情况下，屏幕上会立即显示 3 个如图 5.56 所示的按钮。如果将<include>标签换成如下的代码，在程序启动时，只会显示在 main.xml 文件中的定义按钮，如图 5.57 所示。

```
<ViewStub android:id="@+id/viewstub" android:inflatedId="@+id/button_layout"
    android:layout="@layout/custom" android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```



▲ 图 5.56 使用<include>标签装载控件



▲ 图 5.57 使用<ViewStub>标签装载控件

在使用<ViewStub>标签引用布局文件后，还需要调用 ViewStub.inflate 或 ViewStub.setVisibility(View.VISIBLE)方法才能装载所引用的控件，代码如下：

```
public void onClick_Button(View v)
{
```

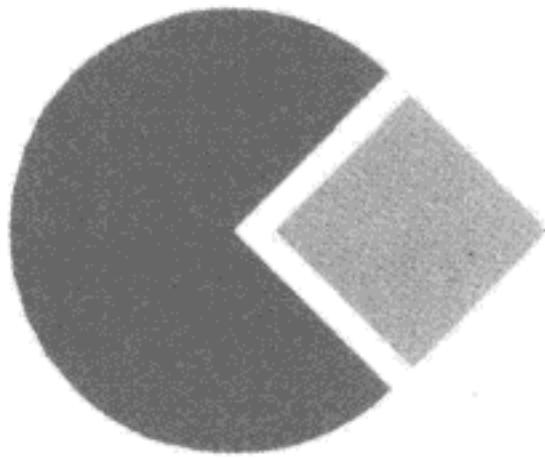
```
// ViewStub 控件只能获得一次，第二次再使用 findViewById 获得该 ViewStub 对象，则返回 null
View view = findViewById(R.id.viewstub);
if (view != null)
{
    // 或调用 ViewStub.inflate 方法
    // view = ((ViewStub) view).inflate();
    // 装载 ViewStub 引用的 custom.xml 文件中的控件
    ((ViewStub)view).setVisibility(View.VISIBLE);
}
else
{
    setTitle("view is null");
}
```

单击“我的按钮”后，会显示在 custom.xml 文件中定义的两个按钮，效果与图 5.56 完全一样。

**注意** <ViewStub>与<include>标签一样，也可以设置所引用布局文件中根节点所有与布局相关的属性。所不同的是<include>标签的 android:id 属性直接覆盖了所引用布局文件中根节点的 android:id 属性值，而<ViewStub>标签的 android:id 属性与普通控件标签的 android:id 属性一样，用于在代码中引用控件。在<ViewStub>标签中需要使用 android:inflatedId 属性覆盖所引用布局文件中根节点的 android:id 属性值。虽然<ViewStub>可完全取代<include>，但唯一的不足是<ViewStub>目前还无法取代<merge>。

## 5.14 小结

本章介绍的内容是 Android SDK 中的最核心的部分：控件。这些控件以及它们的变种会被应用在绝大多数的应用程序中。为了使读者更好地了解这些控件，在本章的开始部分首先介绍了几乎所有控件都会用到的属性，然后将 Android SDK 提供的标准控件分成了若干类别来介绍。这些类别包括“时间与日期控件”、“进度条控件”、“列表控件”、“滚动控件”。除此之外，对于比较重要或无法分类的控件用单独的部分进行了详细讲解。别看 Android SDK 提供了大量的控件，但这些控件很多都有着“近亲”关系，例如，TextView 就是很多控件类（EditText、Button）的父类。因此，本章用了很多篇幅详细讲解了 TextView 控件的使用方法以及各种技巧。值得一提的是最后介绍的 ViewStub 控件，可能很多初学者并不太了解或不经常使用这个控件。这里有一点需要说明，ViewStub 控件对美化界面起不到任何帮助，但如果想使程序运行得更有效率，笔者强烈建议在允许的情况下大量使用 ViewStub，这样会在某种程度上使程序变得更流畅，更节省系统资源。总之，本章所介绍的每一个控件都非常有实用价值，建议读者认真学习本章的内容。



# 第6章 友好的菜单—— Menu 介绍与实例

无论是 PC 上的程序，还是手机上的程序，都少不了本章的主人公：菜单。菜单在 Android 应用程序中占据了非常重要的位置。当屏幕上放不下按钮，也不想使用 Tab 时，使用菜单将是最好的选择。这样既可以节省屏幕空间，也可以为程序加入丰富的用户接口。在 Android SDK 中提供了一组默认的菜单（包括选项菜单、弹出菜单和上下文菜单）。除此之外，本章还介绍了实现类似 iPhone、UCWeb 的菜单效果。

## 6.1 菜单的基本用法

工程目录：src\ch06\ch06\_menu

菜单是 Android 系统中重要的用户接口之一。在 Android 系统中提供了丰富多彩的菜单，例如，系统的主菜单，也可称为选项菜单；带图像、复选框、选项按钮的菜单；上下文菜单。本节将对这些菜单的实现方法进行详细讲解。

### 6.1.1 创建选项菜单（Options Menu）

Activity 类的 `onCreateOptionsMenu` 事件方法用来创建选项菜单，该方法的定义如下：

```
public boolean onCreateOptionsMenu(Menu menu);
```

一般需要将创建选项菜单的代码放在 `onCreateOptionsMenu` 方法中。通过 `Menu.add` 方法可以添加一个选项菜单项。该方法有 4 种重载形式，它们的定义如下：

```
public MenuItem add(int titleRes);
public MenuItem add(CharSequence title);
public MenuItem add(int groupId, int itemId, int order, int titleRes);
public MenuItem add(int groupId, int itemId, int order, CharSequence title);
```

`add` 方法最多有 4 个参数，这些参数的含义如下。

- `groupId`: 菜单项的分组 ID，该参数一般用于带选项按钮的菜单（将在后面详细介绍）。参数值可以是负整数、0 和正整数。
- `itemId`: 当前添加的菜单项的 ID。该参数值可以是负整数、0 和正整数。
- `order`: 菜单显示顺序。Android 系统在显示菜单项时，根据 `order` 参数的值按升序从左到右、从上到下显示菜单项。参数值必须是 0 和正整数，不能为负整数。

- `titleRes` 或 `title`: 菜单项标题的字符串资源 ID 或字符串。

如果使用 `add` 方法的前两种重载形式, `groupId`、`itemId` 和 `order` 三个参数的值都为 0。这时菜单项的显示顺序就是菜单项的添加顺序。下面的代码添加了 3 个选项菜单项:

```
public boolean onCreateOptionsMenu(Menu menu)
{
    menu.add(1, 1, 1, "菜单项 1");
    menu.add(1, 2, 2, "菜单项 2");
    menu.add(1, 3, 3, "菜单项 3");
    return true;
}
```

Android 的选项菜单最多显示 6 个菜单项, 如果不足 6 个菜单项, 可根据实际情况来排列, 例如, 在有 5 个菜单项的情况下, 第 1 行会显示两个菜单项, 第 2 行会显示 3 个菜单项, 如图 6.1 所示。如果菜单项超过 6 个, 系统会显示前 5 个菜单项, 而最后一个菜单项的文本是“更多”或“More”, 如图 6.2 所示。单击该菜单项后, 会显示其余的菜单项。如果菜单项的文本过长, 系统会显示三行两列的选项菜单, 而不是如图 6.2 所示的两行三列的选项菜单, 而且过长的标题会从左到右移动显示。



▲ 图 6.1 有 5 个菜单项的选项菜单



▲ 图 6.2 超过 6 个菜单项的 Activity 菜单

### 6.1.2 带图像的选项菜单

从上一节中 `Add` 方法的定义可以看出, 该方法返回了一个 `MenuItem` 对象, 每一个 `MenuItem` 对象对应一个菜单项。可以通过 `MenuItem` 接口的相应方法来设置与菜单项相关的内容, 例如, 显示在菜单项上的图像。

在如图 6.1 所示的选项菜单中可以看到“删除”和“文件”菜单项都带有一个图像, 这个图像需要通过 `MenuItem.setIcon` 方法来添加, 该方法的定义如下:

```
// 通过图像资源 ID 装载图像
public MenuItem setIcon(int iconRes);
// 通过 Drawable 对象装载图像
public MenuItem setIcon(Drawable icon);
```

下面的代码设置了菜单项的图像:

```
MenuItem deleteMenuItem = menu.add(1, 1, "删除");
deleteMenuItem.setIcon(R.drawable.delete); // 设置“删除”菜单项的图像
```

### 6.1.3 关联 Activity

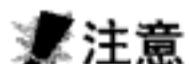
虽然可以通过代码显示一个 `Activity`, 但我们还有更简单的方法, 就是直接将 `Activity` 与菜单

项关联。做法非常简单，只需要使用 `MenuItem.setIntent` 方法指定一个 `Intent` 对象即可。`setIntent` 方法的定义如下：

```
| public MenuItem setIntent(Intent intent);
```

将一个 `Activity` 与菜单项关联后，单击该菜单项，系统会调用 `startActivity` 方法显示与菜单项关联的 `Activity`。下面的代码将 `AddActivity` 与“添加”菜单项关联，单击“添加”菜单项，系统就会显示 `AddActivity`。

```
| MenuItem addMenuItem = menu.add(1, 1, 1, "添加");
// 将 AddActivity 与“添加”菜单项进行关联
addMenuItem.setIntent(new Intent(this, AddActivity.class));
```



**注意** 如果设置了菜单项的单击事件，并且单击事件返回 `true`，则与菜单项关联的 `Activity` 将失效。也就是说，系统调用单击事件方法后，就不会再调用 `startActivity` 方法显示与菜单项关联的 `Activity` 了。

#### 6.1.4 响应菜单的单击动作

通过调用 `MenuItem.setOnMenuItemClickListener` 方法可以设置菜单项的单击事件，该方法有一个 `OnMenuItemClickListener` 类型的参数，处理菜单项的单击事件类必须实现 `OnMenuItemClickListener` 接口。下面的代码为“删除”菜单项设置了单击事件。

```
public class Main extends Activity implements OnMenuItemClickListener
{
    // 菜单项单击事件方法
    @Override
    public boolean onMenuItemClick(MenuItem item)
    {
        // 在这里编写菜单项单击事件的代码，可根据 item 参数的 getItemId 方法来确定单击的是哪个菜单项
        return true;
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuItem deleteMenuItem = menu.add(1, 2, 2, "删除");
        deleteMenuItem.setIcon(R.drawable.delete);
        deleteMenuItem.setOnMenuItemClickListener(this); // 设置“删除”菜单项的单击事件
    }
}
```

除了设置菜单项的单击事件外，还可以使用 `Activity` 类的 `onOptionsItemSelected` 和 `onMenuItemSelected` 方法来响应菜单项的单击事件。这两个方法的定义如下：

```
| public boolean onOptionsItemSelected(MenuItem item);
public boolean onMenuItemSelected(int featureId, MenuItem item);
```

这两个方法都有一个 item 参数，用于传递被单击的菜单项的 MenuItem 对象。可以根据 MenuItem 接口的相应方法（例如，getTitle 方法和 getItemId 方法）判断单击的是哪个菜单项。

既然有 3 种响应菜单项单击事件的方法，就会产生一个问题：如果同时使用这 3 种方法，它们都会起作用吗？如果都起作用，那么调用顺序如何呢？实际上，当 onMenuItemClick 方法返回 true 时，另两种单击事件的响应方式都会失效，也就是说，单击菜单项时，系统不会再调用 onOptionsItemSelected 和 onMenuItemSelected 方法了。如果未设置菜单项的单击事件（onMenuItemClick 方法），而同时使用了另外两种响应单击事件的方式，系统会根据在 onMenuItemSelected 方法中调用父类（Activity 类）的 onMenuItemSelected 方法的位置来决定先调用 onOptionsItemSelected 方法还是先调用 onMenuItemSelected 方法。

```
// 如果将 super.onOptionsItemSelected(...)放在 Log.d(...)后面调用,
// 系统会在执行完 onOptionsItemSelected 方法中的代码后再调用 onOptionsItemSelected 方法
@Override
public boolean onOptionsItemSelected(int featureId, MenuItem item)
{
    super.onOptionsItemSelected(featureId, item); // 这条语句调用了 onOptionsItemSelected 方法
    Log.d("onMenuItemSelected:itemId=", String.valueOf(item.getItemId()));
    return true;
}
```

### 6.1.5 动态添加、修改和删除选项菜单

在很多 Android 系统中，需要在程序的运行过程中根据具体情况动态地对选项菜单进行处理，例如，增加菜单项、修改菜单项的标题和图像。实现这个功能的关键是获得描述选项菜单的 Menu 对象。

Activity 类中的很多方法都可以获得 Menu 对象。例如，onCreateOptionsMenu 方法的 menu 参数就是 Menu 类型，我们要做的就是在 onCreateOptionsMenu 方法中将 Menu 对象保存在类变量中。下面的代码动态地向选项菜单中添加了 5 个菜单项：

```
public class Main extends Activity implements OnMenuItemClickListener,
    OnClickListener
{
    private Menu menu;
    private int menuItemId = Menu.FIRST;           // Menu.FIRST 的值是 1
    @Override
    public void onClick(View view)
    {
        // 只有单击手机上的“Menu”按钮，onCreateOptionsMenu 方法才会被调用，
        // 因此，如果不按“Menu”按钮，Main 类的 menu 变量的值是 null
        if (menu == null) return;
        // 向 Activity 菜单添加 5 个菜单项，菜单项的 id 从 10 开始
        for (int i = 10; i < 15; i++)
        {
            int id = menuItemId++;
            menu.add(1, id, id, "菜单" + i);
        }
    }

    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {
        switch (item.getItemId())
        {
            case R.id.item1:
                Toast.makeText(this, "选择了第一个菜单项", Toast.LENGTH_SHORT).show();
                break;
            case R.id.item2:
                Toast.makeText(this, "选择了第二个菜单项", Toast.LENGTH_SHORT).show();
                break;
            case R.id.item3:
                Toast.makeText(this, "选择了第三个菜单项", Toast.LENGTH_SHORT).show();
                break;
            case R.id.item4:
                Toast.makeText(this, "选择了第四个菜单项", Toast.LENGTH_SHORT).show();
                break;
            case R.id.item5:
                Toast.makeText(this, "选择了第五个菜单项", Toast.LENGTH_SHORT).show();
                break;
        }
        return true;
    }
}
```

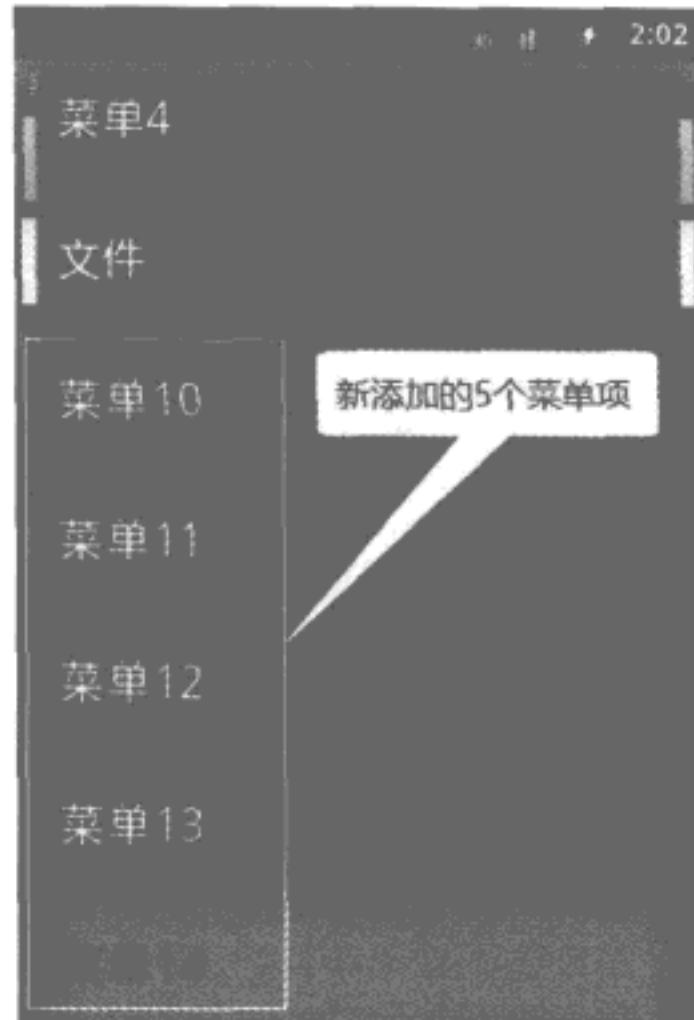
```

    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        this.menu = menu;           // 保存 Menu 变量
        return super.onCreateOptionsMenu(menu);
    }
    ...
}

```

运行程序后，单击模拟器上的“Menu”按钮（为了调用 `onCreateOptionsMenu` 方法以获得 `Menu` 对象），然后单击“动态添加 5 个菜单项”按钮，再次单击模拟器上的“Menu”按钮，会看到选项菜单中最后一个“更多”或“More”菜单项，单击“更多”或“More”菜单项，将显示如图 6.3 所示的效果。

既然有了 `Menu` 对象，修改和删除指定的菜单项就变得非常容易了，读者可以使用 `Menu` 接口的相应方法来完成这些工作。



▲ 图 6.3 动态添加的 5 个菜单项

### 6.1.6 带复选框和选项按钮的子菜单

传统的子菜单是以层次结构显示的，而 Android 中的子菜单采用了弹出式的显示方式。也就是当单击带有子菜单的菜单项后，父菜单会关闭，而在屏幕上会单独显示子菜单。

`Menu.addSubMenu` 方法用来添加子菜单。该方法有 4 种重载形式，它们的定义如下：

```

SubMenu addSubMenu(final CharSequence title);
SubMenu addSubMenu(final int titleRes);
SubMenu addSubMenu(final int groupId, final int itemId, int order, final CharSequence
title);
SubMenu addSubMenu(int groupId, int itemId, int order, int titleRes);

```

`addSubMenu` 方法和 `add` 方法的参数个数和类型完全相同，所不同的是它们的返回值类型。`addSubMenu` 方法返回了一个 `SubMenu` 对象（`SubMenu` 是 `Menu` 的子接口），可以通过 `SubMenu.add` 方法添加子菜单项。`SubMenu.add` 方法与 `Menu.add` 方法在功能和使用方法上完全相同，这两个 `add` 方法都会返回一个 `MenuItem` 对象。

在子菜单项上不能显示图像，但可以在子菜单的头部显示图像，不过子菜单项可以带复选框和选项按钮。例如，下面的代码向“文件”菜单项添加了 3 个子菜单项，并将第 1 个子菜单项设置成复选框类型，将后两个子菜单项设置成选项按钮类型，同时为子菜单头设置了图像。

```
public boolean onCreateOptionsMenu(Menu menu)
{
    // 添加子菜单
    SubMenu fileSubMenu = menu.addSubMenu(1, 1, 2, "文件");
    fileSubMenu.setIcon(R.drawable.file); // 设置在选项菜单中显示的图像
    fileSubMenu.setHeaderIcon(R.drawable.headerfile); // 设置子菜单头的图像
    MenuItem newItem = fileSubMenu.add(1, 2, 2, "新建");
    newItem.setCheckable(true); // 将第 1 个子菜单项设置成复选框类型
    newItem.setChecked(true); // 选中第 1 个子菜单项中的复选框
    MenuItem openMenuItem = fileSubMenu.add(2, 3, 3, "打开");
    MenuItem exitMenuItem = fileSubMenu.add(2, 4, 4, "退出");
    exitMenuItem.setChecked(true); // 将第 3 个子菜单项的选项按钮设为选中状态
    fileSubMenu.setGroupCheckable(2, true, true); // 将后两个子菜单项设置成选项按钮类型
}
```

在编写上面代码时应注意如下几点。

- 添加子菜单并不是直接在 `MenuItem` 下添加菜单项，而需要使用 `addSubMenu` 方法创建一个 `SubMenu` 对象，并在 `SubMenu` 下添加子菜单项。`SubMenu` 和 `MenuItem` 是平级的，这一点在添加子菜单时要注意。
- 将子菜单项设置成复选框类型，需要使用 `MenuItem.setCheckable` 方法。但设置成选项按钮类型，不需要使用 `setCheckable` 方法，而要将同一组的选项按钮（子菜单项）的 `groupId` 设置成相同的值，同时使用 `setGroupCheckable` 方法来设置这个 `groupId`。该方法的第一个参数指定子菜单项的 `groupId`，第 2 个参数必须为 `true`。如果第 3 个参数为 `true`，相同 `groupId` 的子菜单项会被设置成选项按钮效果；如果为 `false`，相同 `groupId` 的子菜单项会被设置成复选框效果。根据相同 `groupId` 设置的选项按钮和复选框除了显示效果，并没有什么其他的不同。在菜单项上加选项按钮或复选框主要是为了标志菜单项当前的状态，至于选中或未选中状态代表什么，完全由开发人员自己决定。

- 使用 `setChecked` 方法可以将复选框或选项按钮设置成选中状态。
- 选项菜单不支持嵌套子菜单，也就是说，不能在子菜单项下再建立子菜单，否则系统将抛出异常。

运行程序后，单击选项菜单中的“文件”菜单项，根据 `setGroupCheckable` 方法的第 3 个参数值是 `true` 或 `false`，显示的选项效果或复选框效果如图 6.4 和图 6.5 所示。



▲ 图 6.4 选项按钮效果



▲ 图 6.5 复选框按钮效果

### 6.1.7 上下文菜单

上下文菜单可以和任意 View 对象进行关联，例如，`TextView`、`EditText`、`Button` 等控件都可以关联上下文菜单。上下文菜单的显示效果和子菜单有些类似，也分为菜单头和菜单项。

要想创建上下文菜单，需要覆盖 `Activity.onCreateContextMenu` 方法，该方法的定义如下：

```
public void onCreateContextMenu(ContextMenu menu, View view, ContextMenuInfo menuInfo);
```

可以使用 `ContextMenu.setHeaderTitle` 和 `ContextMenu.setHeaderIcon` 方法设置上下文菜单头的标题和图像。上下文菜单项不能带图像，但可以带复选框或选项按钮（这一点和子菜单相同）。上下文菜单与选项菜单一样，也不支持嵌套子菜单。下面的代码创建一个包含 4 个菜单项的上下文菜单，其中最后一个菜单项包含两个子菜单项。

```
public void onCreateContextMenu(ContextMenu menu, View view, ContextMenuInfo menuInfo)
{
    super.onCreateContextMenu(menu, view, menuInfo);
    menu.setHeaderTitle("上下文菜单");
    menu.setHeaderIcon(R.drawable.face);
    // 添加 3 个上下文菜单项, Menu.NONE 的值是 0
    menu.add(0, menuItemId++, Menu.NONE, "菜单项 1").setCheckable(true).setChecked(true);
    menu.add(20, menuItemId++, Menu.NONE, "菜单项 2");
    // 选中第 2 个选项按钮
    menu.add(20, menuItemId++, Menu.NONE, "菜单项 3").setChecked(true);
    menu.setGroupCheckable(20, true, true);
    // 添加带子菜单的上下文菜单项
    SubMenu sub = menu.addSubMenu(0, menuItemId++, Menu.NONE, "子菜单");
    sub.add("子菜单项 1");
    sub.add("子菜单项 2");
}
```

上下文菜单与其他菜单不同的是必须注册到指定的 View 上才能显示。注册上下文菜单可以使用 Activity.registerForContextMenu 方法。下面的代码将当前 Activity 的上下文菜单注册到 Button、EditText 和 TextView 上。

```
Button button = (Button) findViewById(R.id.btnAddMenu);
EditText editText = (EditText) findViewById(R.id.edittext);
TextView textView = (TextView) findViewById(R.id.textview);
// 注册上下文菜单
registerForContextMenu(button);
registerForContextMenu(editText);
registerForContextMenu(textView);
```

当一个控件关联上下文菜单后，长按该控件，等一会就会显示上下文菜单，如运行上面的代码后，长按 TextView 控件，会显示如图 6.6 所示的上下文菜单。有一些控件已经有了自己的上下文菜单，例如，EditText，在这种情况下，系统会将我们自定义的上下文菜单项添加到视图自带的上下文菜单项的后面，如图 6.7 所示。



▲ 图 6.6 TextView 控件的上下文菜单



▲ 图 6.7 EditText 控件的上下文菜单

上下文菜单项的单击事件也可以使用单击事件类和 onMenuItemSelected 方法来响应，这和选项菜单、子菜单的响应方法相同。但对于上下文菜单来说，第 3 种响应单击事件的方式需要覆盖 Activity.onContextItemSelected 方法，该方法的定义如下：

```
public boolean onContextItemSelected(MenuItem item);
```

### 6.1.8 菜单事件

Activity 类还有一些与菜单相关的事件方法，这些方法的定义如下：

```
public boolean onPrepareOptionsMenu(Menu menu);
public void onOptionsMenuClosed(Menu menu);
public void onContextMenuClosed(Menu menu);
public boolean onMenuOpened(int featureId, Menu menu);
```

这些方法的含义如下。

- `onPrepareOptionsMenu`: 在显示选项菜单之前被调用。一般可用来修改即将显示的选项菜单。
- `onOptionsMenuClosed`: 在关闭选项菜单时被调用。
- `onContextMenuItemClosed`: 在关闭上下文菜单时被调用。
- `onMenuOpened`: 在显示选项菜单之前被调用。该方法在 `onPrepareOptionsMenu` 方法之后调用。

### 6.1.9 从菜单资源中装载菜单

前面介绍的各种类型的菜单都是通过代码添加的，Android SDK 还允许我们从菜单资源中装载菜单。Android 工程中的所有菜单资源都在 `res\menu` 目录中，例如，下面就是一个包含 3 个菜单项的菜单资源文件。

#### `file_menu.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="新建" />
    <item android:title="打开" />
    <item android:title="退出" />
</menu>
```

编写完了菜单资源，还必须在 `onCreateOptionsMenu` 或 `onCreateContextMenu` 方法中使用如下的代码来装载这个菜单资源。在本例中在 `onCreateContextMenu` 方法中将 `file_menu.xml` 文件中的内容作为上下文菜单添加。

```
getMenuInflater().inflate(R.menu.file_menu, menu);
```

由于在 6.1.7 节已经使用代码添加了一些上下文菜单，因此，`file_menu.xml` 文件中的 3 个菜单项将添加到原有菜单项的后面，如图 6.8 所示。



▲ 图 6.8 从菜单资源中装载菜单

## 6.2 菜单特效

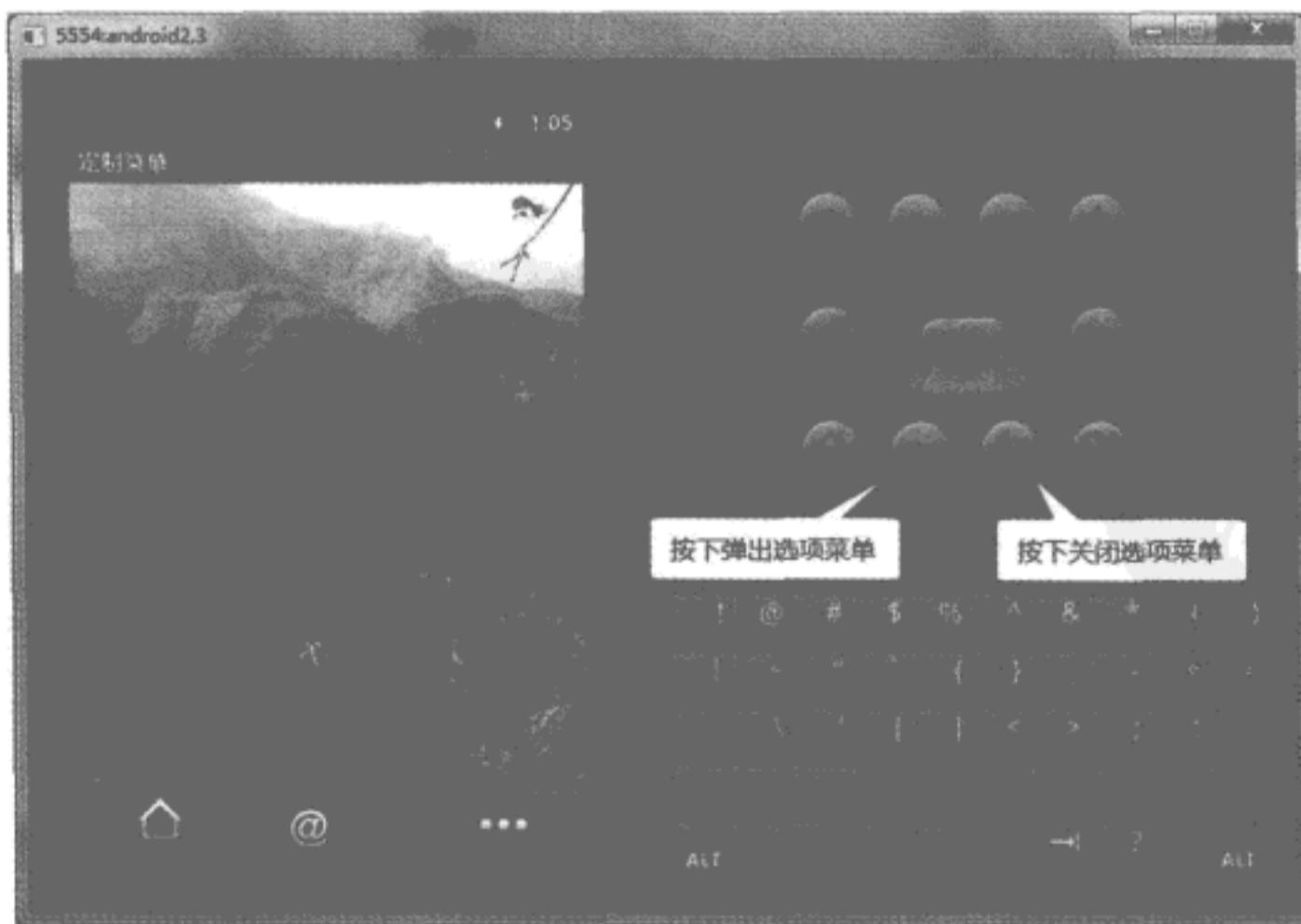
上一节介绍了如何实现 Android 的标准菜单。为了实现更绚丽的效果（例如，模拟 iPhone、UCWeb for Android 的菜单效果），需要使用更多的技巧。本节将为读者讲解这些自定义菜单的技巧，并引领读者逐步完成各种绚丽的菜单效果。

### 6.2.1 自定义菜单

工程目录：src\ch06\ch06\_custom\_menu

Android SDK 本身提供了一种默认创建菜单的机制，但通过这种机制创建的菜单虽然从功能上很完备，但界面效果实在是有点“土”。对于一个拥有绚丽界面的程序配上一个有点“土”的菜单，会使用户感觉很怪，甚至会使绚丽的界面大打折扣。实际上，对于如此灵活和强大的 Android 系统，修改菜单的样式只是小菜一碟，为程序加入漂亮菜单的方法很多。本小节先介绍一种比较常用的方法，就是通过 onKeyDown 事件方法和 PopupWindow 实现自定义的菜单。至于通过这种技术能否设计出绚丽的菜单效果，那就要看我们的设计、美学、心理学功底了。

通过 6.1.1 节介绍的选项菜单可以知道，通过按手机的“Menu”键（是手机上的硬按键，不同手机“Menu”键所在的位置会不同），可以弹出选项菜单，再按“Back”键，选项菜单会关闭。那么要想模拟选项菜单的弹出和关闭效果，只需要监听这两个键的按下事件即可，并且在“Menu”键按下时使用 PopupWindow 弹出一个窗口作为模拟的选项菜单。下面先来看看如图 6.9 所示的模拟选项菜单的效果。



▲ 图 6.9 定制选项菜单

从图 6.9 可以看出，在界面的下方显示了 3 个菜单项：“首页”、“我的”和“更多”。其中“我的”菜单项的文字和图像是左右水平排列，而另两个菜单项上的文字和图像是上下垂直排列的。实

际上，这种效果是由一个普通的布局文件（menu\_layout.xml）完成的，代码如下：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:gravity="bottom">
    <!-- 第一个菜单项：“首页” -->
    <LinearLayout android:id="@+id/home" android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:background="@drawable/button_normal_translucent"
        android:layout_weight="1">
        <ImageView android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:src="@drawable/home"
            android:paddingTop="5dp" />
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="首页"
            android:gravity="center" />
    </LinearLayout>
    <!-- 第二个菜单项：“我的” -->
    <LinearLayout android:orientation="horizontal"
        android:layout_width="fill_parent" android:layout_height="wrap_content"
        android:background="@drawable/button_normal" android:layout_weight="1"
        android:gravity="center">
        <ImageView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:src="@drawable/mine" />
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="我的" />
    </LinearLayout>
    <!-- 第三个菜单项
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:background="@drawable/button_normal"
        android:layout_weight="1">
        <ImageView android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:src="@drawable/more"
            android:paddingTop="18dp" />
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="更多"
            android:gravity="center" android:paddingTop="5dp"/>
    </LinearLayout>
</LinearLayout>

```

在编写上面代码之前，别忘了准备几个相关的图像，例如，本例使用了 5 个图像，分别是 image1.Png、image2.Png、image3.Png、image4.Png 和 image5.Png。其中 button\_normal\_translucent.png 用于“首页”菜单项的背景（半透明效果），button\_normal.png 用于“我的”和“更多”菜单项的背景。home.png、mine.png 和 more.png 分别用于这 3 个菜单项的图像。

下面来编写监听“menu”和“back”键按下动作的代码。按下“back”键要处理的任务有如下两个。

- 如果选项菜单已经弹出，关闭选项菜单。
- 如果选项菜单未弹出，或已经被关闭，直接关闭当前的 Activity，也就是调用 finish 方法。

为了区分上面两个任务，在程序中设置了一个 int 类型状态变量（state），当 state 为 1 时表示选项菜单已弹出，state 为 2 时表示选项菜单未弹出。下面我们看一下完整的实现代码。

```
package mobile.android.ch06.custom.menu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.PopupWindow;
import android.widget.Toast;

public class Main extends Activity
{
    private PopupWindow pop;
    private View layout;
    private int state = 2; // 状态变量，1：选项菜单已弹出，2：选项菜单未弹出

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

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        switch (keyCode)
        {
            case KeyEvent.KEYCODE_MENU: // 按下“menu”键的动作
                // 选项菜单已弹出，不再弹出新的窗口
                if (state == 1)
                    return false;
                // 装载选项菜单布局文件
                layout = getLayoutInflater().inflate(R.layout.menu_layout, null);
                // 创建 PopupWindow 对象，并在指定位置弹出用于显示菜单的窗口
                pop = new PopupWindow(layout, getWindowManager()
                        .getDefaultDisplay().getWidth(), getWindowManager()
                        .getDefaultDisplay().getHeight());
                // 设置弹出窗口的位置
                pop.showAtLocation(layout, Gravity.BOTTOM, 0, 0);
                View home = layout.findViewById(R.id.home);
                // 为“首页”菜单项添加单击事件
                home.setOnClickListener(new OnClickListener()
                {
                    @Override
                    public void onClick(View view)
                    {
                        Toast.makeText(Main.this, "单击定制菜单。", Toast.LENGTH_LONG).show();
                        // 单击“首页”菜单项后，关闭选项菜单
                        pop.dismiss();
                    }
                });
            }
        }
    }
```

```
// 重新设置状态变量
state = 2;
}
});
// 弹出选项菜单后，将状态变量设为 1，表示选项菜单已弹出
state = 1;
return false;
case KeyEvent.KEYCODE_BACK: // 按下“back”键的动作
    if (state == 1)
    {
        // 如果选项菜单已弹出，关闭它
        pop.dismiss();
        // 将状态变量设为选项菜单已关闭
        state = 2;
    }
    else if (state == 2)
    {
        // 如果选项菜单还没有显示，或已经关闭，则直接关闭当前的 Activity
        finish();
    }
    return false;
}
// 除“menu”和“back”按下事件外，仍需调用 Activity 类的 onKeyDown 方法来响应其他键的按下事件
return super.onKeyDown(keyCode, event);
}
```

在编写上面代码时应注意如下几点。

- 对于选项菜单来说，一般单击某个菜单项后，会执行一些动作，并且选项菜单会自动关闭。为了模拟这一过程，为“首页”菜单项添加了一个单击事件。当单击“首页”菜单项时，会弹出一个 Toast 提示信息，并且选项菜单会关闭。
  - 当执行完按下“menu”或“back”键的动作后，onKeyDown 方法应返回一个常量（false 或 true 都可以），不能再调用 super.onKeyDown 方法，否则在执行完定制的菜单项动作后，又会执行系统的默认动作。例如，当按下“back”键后，关闭弹出菜单后，连当前的 Activity 也一起关了。当然，如果是除了“menu”和“back”的其他键按下时还是需要调用 Activity 类的 onKeyDown 方法的（也就是 super.onKeyDown 方法），这样在程序中还可以响应其他的按键事件，否则程序除了“menu”和“back”键外，其他的键几乎都不好使了。
  - showAtLocation 方法用于控件弹出窗口的位置。该方法的第一个参数是一个 View 对象，实际上，showAtLocation 方法内部只是需要调用 View.getWindowToken 方法来获得一个 IBinder 对象。showAtLocation 方法的第二个参数表示弹出窗口的位置，本例中设置了弹出窗口在屏幕底部显示。最后两个参数分别表示水平和垂直偏移量。本例都设为 0，表示不发生偏移。因此，弹出窗口会在屏幕的最底部显示，也就是显示选项菜单的位置。

### 6.2.2 模拟 UCWeb 效果菜单

工程目录: src\ch06\ch06 ucweb menu

可能有很多读者使用过 Android 版的 UCWeb（也称为 UC 浏览器）。在 UCWeb 的主界面可以弹出包含很多图像的菜单，如图 6.10 所示。



▲ 图 6.10 UCWeb 主界面

实际上，这个弹出菜单也是一个弹出窗口，本节将介绍如何弹出类似的菜单。在上一节介绍了通过 onKeyDown 事件来捕捉“menu”键盘事件来弹出菜单，本节将介绍另外一种弹出自定义菜单的方法，即通过 onCreateOptionsMenu 和 onMenuOpened 两个方法相互配合来弹出自定义菜单。onCreateOptionsMenu 方法用于创建选项菜单，在显示选项菜单之前，系统会调用 onMenuOpened 方法，如果该方法返回 false，则在 onCreateOptionsMenu 方法中创建的选项菜单将不再显示。因此，可以在 onMenuOpened 方法中弹出用于显示自定义菜单的窗口。本例的完整实现代码如下：

```
package mobile.android.ch06_ucweb.menu;

import android.app.Activity;
import android.os.Bundle;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup.LayoutParams;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.GridView;
import android.widget.PopupWindow;
import android.widget.Toast;

public class Main extends Activity implements OnClickListener,OnItemClickListener {
    private PopupWindow popup;
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 创建在弹出窗口中显示的 GridView 对象
    GridView gvPopupWindow = (GridView) getLayoutInflater().inflate(R.layout.popup_
window, null);
    // 为 GridView 提供数据的 Adapter 对象
    GridAdapter gridAdapter = new GridAdapter(this);
    gvPopupWindow.setAdapter(gridAdapter);
    gvPopupWindow.setOnKeyListener(this);
    gvPopupWindow.setOnItemClickListener(this);
    // 创建用于显示菜单的 PopupWindow 对象。菜单是通过 GridView 对象显示网格状图像和文字
    popup = new PopupWindow(gvPopupWindow, LayoutParams.FILL_PARENT,
                           LayoutParams.WRAP_CONTENT);
    // 使 PopupWindow 可以获得焦点，以便可以通过轨迹球或上、下、左、右键来控制菜单项
    popup.setFocusable(true);
}
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
    // 当单击 GridView 中的每一项时，先关闭弹出窗口，然后显示一条提示信息
    popup.dismiss();
    Toast.makeText(this, Const.GRID_ITEM_TEXT_LIST[position], Toast.LENGTH_LONG).show();
}
@Override
public boolean onKey(View v, int keyCode, KeyEvent event)
{
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_BACK:
            // 捕捉 Back 键。如果窗口已经显示，关闭它
            if(popup.isShowing())
                popup.dismiss();
            break;
    }
    return false;
}
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    // 必须创建一项，否则系统不会调用 onMenuOpened 方法
    menu.add("menu");
    return super.onCreateOptionsMenu(menu);
}
@Override
public boolean onMenuOpened(int featureId, Menu menu)
{
    if (popup != null)
    {
        if (popup.isShowing())
        {
```

```
// 如果菜单已显示，关闭它
popup.dismiss();
}
else
{
    View layout = getLayoutInflater().inflate(R.layout.main, null);
    // 弹出菜单
    popup.showAtLocation(layout, Gravity.CENTER, 0, 0);
}
}
return false;
}
}
```

在上面代码中使用了一个 GridAdapter 类，该类用于向 GridView 对象提供数据，代码如下：

```
package mobile.android.ch06_ucweb.menu;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class GridAdapter extends BaseAdapter
{
    private Context mContext;
    private LayoutInflater mLayoutInflater;
    public GridAdapter(Context context)
    {
        mContext = context;
        mLayoutInflater = (LayoutInflater) context.getSystemService(Context.LAYOUT_
        INFLATER_SERVICE);
    }
    @Override
    public int getCount()
    {
        return Const.GRID_ITEM_ICON_ID_LIST.length;
    }
    @Override
    public Object getItem(int position)
    {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public long getItemId(int position)
    {
        return 0;
    }
    @Override
```

## Android 开发权威指南

```

public View getView(int position, View convertView, ViewGroup parent)
{
    // 此方法返回一个包含 ImageView 和 TextView 的 View 对象
    if (convertView == null)
        convertView = mLayoutInflater.inflate(R.layout.grid_item, null);
    ImageView ivGridItemIcon = (ImageView) convertView
        .findViewById(R.id.ivGridItemIcon);
    TextView tvGridItemText = (TextView) convertView
        .findViewById(R.id.tvGridItemText);
    // 设置在 ImageView 对象中显示的图像
    ivGridItemIcon.setImageResource(Const.GRID_ITEM_ICON_ID_LIST[position]);
    // 设置在 TextView 中显示的文本
    tvGridItemText.setText(Const.GRID_ITEM_TEXT_LIST[position]);
    return convertView;
}
}

```

在使用 `getView` 方法返回 `View` 对象时要注意，每一个 `View` 对象用于显示当前位置（`position` 指定的位置）的内容，对于 `GridView` 来说，就是一个单元格。对于 `ListView` 来说，就是一个列表项。不过如果每一个单元格或列表项中的控件都一样的话（如本例每一个单元格都只有一个 `ImageView` 和 `TextView`），并不需要总是创建新的 `View` 对象。`getView` 方法有一个 `convertView` 参数，如果该参数不为 `null`，则表示曾经创建的 `View` 对象。我们只需要获得这个 `View` 对象，并重新设置其中控件的值即可。从这一点可以看出，系统在创建每一个 `View` 对象后，并不会释放它们，而是将这些 `View` 对象保存了起来。当 `GridView` 或 `ListView` 进行滚动时，会将这些曾经创建的 `View` 对象再次传入 `getView` 方法，这样就可以重新利用这些 `View` 对象了。通过这种方式可以大大节省系统的资源（虽然 Java 有垃圾回收机制，但一般只有在系统资源快要耗尽时才会回收不需要的内存资源）。

本例还涉及一些图像文件，并且在 `Const` 类中定义了这些图像文件以及相对应的文本，以便在代码中使用它们。`Const` 类的代码如下：

```

package mobile.android.ch06_ucweb.menu;
public class Const
{
    public final static int[] GRID_ITEM_ICON_ID_LIST = new int[]
    { R.drawable.intercept_list, R.drawable.intercept_rule, R.drawable.intercepted_
record,
        R.drawable.location, R.drawable.incoming_and_outgoing_setting,
        R.drawable.privacy_manager, R.drawable.ip, R.drawable.dial,
        R.drawable.useful};
    public final static String[] GRID_ITEM_TEXT_LIST = new String[]
    { "拦截名单", "拦截规则", "拦截记录", "归属地查询", "来电设置", "隐私管理", "IP 电话设置",
        "通讯记录", "常用号码"};
}

```

除此之外，本例还涉及两个布局文件 `popup_window.xml` 和 `grid_item.xml`，其中 `popup_window.xml` 用于弹出窗口的布局，`grid_item.xml` 用于 `GridView` 对象中每一个单元格的布局。这两个布局文件的代码如下：

**popup\_window.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<GridView android:id="@+id/gvPopupWindow"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:gravity="center" android:padding="10dp" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:horizontalSpacing="10px"
    android:verticalSpacing="10px" android:stretchMode="columnWidth"
    android:columnWidth="80dp" android:numColumns="auto_fit"
    android:background="@drawable/popup_window_border"/>
```

**grid\_item.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:gravity="center">
    <ImageView android:id="@+id/ivGridItemIcon"
        android:layout_width="50dp" android:layout_height="50dp" />
    <TextView android:id="@+id/tvGridItemText"
        android:layout_width="wrap_content" android:layout_height="wrap_content" android:
        textColor="#FFF" />
</LinearLayout>
```

运行本程序，按下“menu”菜单，显示效果如图 6.11 所示。

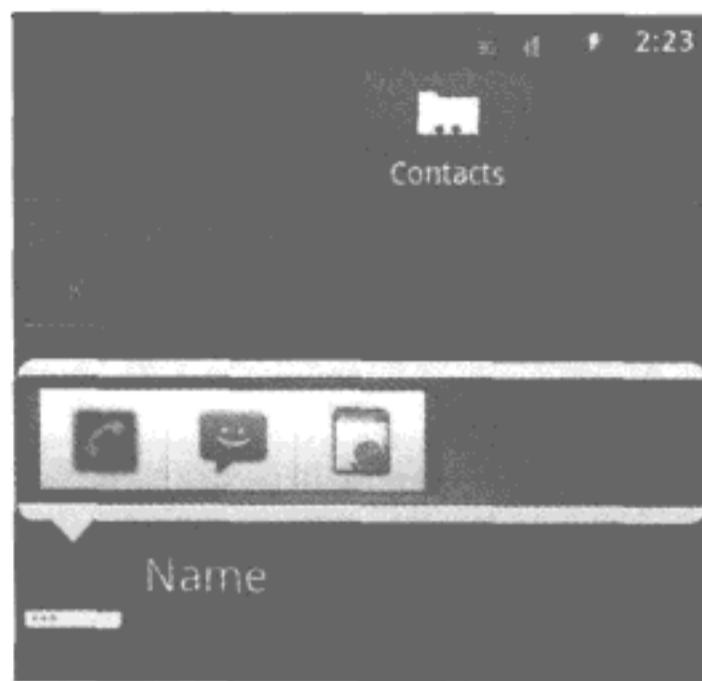


▲ 图 6.11 模拟 UCWeb 的弹出菜单

### 6.2.3 QuickContactBadge 与联系人菜单

工程目录: src\ch06\ch06\_QuickContactBadge

我们在使用 Android 内置的联系人功能时会发现，在单击某个联系人后，会弹出一个可以选择的图像菜单，其中包括拨打电话、发短信等功能，如图 6.12 所示。



▲ 图 6.12 联系人菜单

这个功能实际上是由 QuickContactBadge 控件完成的。QuickContactBadge 是 ImageView 的子类，因此，QuickContactBadge 完全可以作为一个 ImageView 来使用。但 QuickContactBadge 还可以查询系统的联系人列表，并根据联系人的相关信息显示如图 6.12 所示的快捷菜单。现在我们先在布局文件中放两个 QuickContactBadge 控件。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <QuickContactBadge android:id="@+id/badge1"
        android:layout_marginLeft="2dip" android:layout_marginRight="14dip"
        android:layout_marginTop="4dip" android:layout_marginBottom="3dip"
        android:layout_alignParentLeft="true" android:layout_alignParentTop="true"
        android:layout_height="wrap_content" android:layout_width="wrap_content"
        android:src="@drawable/ic_contact_picture" style="?android:attr/quickContact
        BadgeStyleWindowSmall" />
    <QuickContactBadge android:id="@+id/badge2"
        android:layout_marginLeft="2dip" android:layout_marginRight="14dip"
        android:layout_marginTop="4dip" android:layout_marginBottom="3dip"
        android:layout_alignParentLeft="true" android:layout_alignParentTop="true"
        android:layout_height="wrap_content" android:layout_width="wrap_content"
        android:src="@drawable/ic_contact_picture" style="?android:attr/quickContact
        BadgeStyleWindowLarge" />
</LinearLayout>
```

其中 style 属性指定了显示图像菜单的风格。如果 quickContactBadgeStyleWindowLarge 风格可以显示联系人的姓名，quickContactBadgeStyleWindowSmall 风格只显示图像菜单。在编写代码之前，先在系统联系人中添加两个联系人，并输入不同的姓名、E-mail 和电话，然后编写如下的代码：

```
package mobile.android.ch06.quickcontactbadge;

import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.ContactsContract.Contacts;
import android.widget.QuickContactBadge;
```

```

public class Main extends Activity
{
    static final String[] CONTACTS_SUMMARY_PROJECTION = new String[]
    { Contacts._ID, Contacts.DISPLAY_NAME, Contacts.STARRED,
        Contacts.TIMES_CONTACTED, Contacts.CONTACT_PRESENCE,
        Contacts.PHOTO_ID, Contacts.LOOKUP_KEY, Contacts.HAS_PHONE_NUMBER, };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        String select = "(((" + Contacts.DISPLAY_NAME + " NOTNULL) AND (
            + Contacts.HAS_PHONE_NUMBER + "=1) AND (
            + Contacts.DISPLAY_NAME + " != '' ))";
        // 查询所有的联系人
        Cursor cursor = getContentResolver().query(Contacts.CONTENT_URI,
            CONTACTS_SUMMARY_PROJECTION, select, null,
            Contacts.DISPLAY_NAME + " COLLATE LOCALIZED ASC");
        // 将记录指针移动到第一条记录
        cursor.moveToFirst();
        // 创建第一个QuickContactBadge 对象
        QuickContactBadge badge1 = (QuickContactBadge) findViewById(R.id.badge1);
        // 创建第二个QuickContactBadge 对象
        QuickContactBadge badge2 = (QuickContactBadge) findViewById(R.id.badge2);
        // 获得联系人的 ID
        long contactId = cursor.getLong(cursor.getColumnIndex(Contacts._ID));
        // 获得联系人的 lookup_key
        String lookupKey = cursor.getString(cursor.getColumnIndex(Contacts.LOOKUP_KEY));
        // 将当前联系人与 QuickContactBadge 控件关联
        badge1.assignContactUri(Contacts.getLookupUri(contactId, lookupKey));
        // 将记录指针移动到第二条记录
        cursor.moveToNext();
        contactId = cursor.getLong(cursor.getColumnIndex(Contacts._ID));
        lookupKey = cursor.getString(cursor.getColumnIndex(Contacts.LOOKUP_KEY));
        badge2.assignContactUri(Contacts.getLookupUri(contactId, lookupKey));
    }
}

```

在上面的代码中涉及了 Content Provider 和数据库技术，这两种技术将在后面的章节详细讨论。本例先从系统联系人数据库中查询到刚才加入的两个联系人，然后分别将两个联系人与两个 QuickContactBadge 控件相关联。由于本例需要获得联系人信息，因此，需要在 AndroidManifest.xml 文件中添加如下授权代码。

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

现在来运行程序，并分别单击两个 QuickContactBadge 控件，会显示不带联系人姓名和带联系人姓名的两种风格的图像菜单，如图 6.13 和图 6.14 所示。



QuickContactBadge 控件并不会自动显示联系人的头像，要显示联系人的头像，需要像 ImageView 控件一样使用 android:src 属性设置，或使用相应的方法来装载图像资源。



▲ 图 6.13 不显示联系人姓名



▲ 图 6.14 联系人姓名

### 扩展学习：与联系人关联的其他方式

除了使用 QuickContactBadge.assignContactUri 方法将 QuickContactBadge 与联系人关联外，还可以使用如下两个方法与联系人关联。

```
QuickContactBadge.assignContactFromEmail  
QuickContactBadge.assignContactFromPhone
```

这两个方法的定义如下：

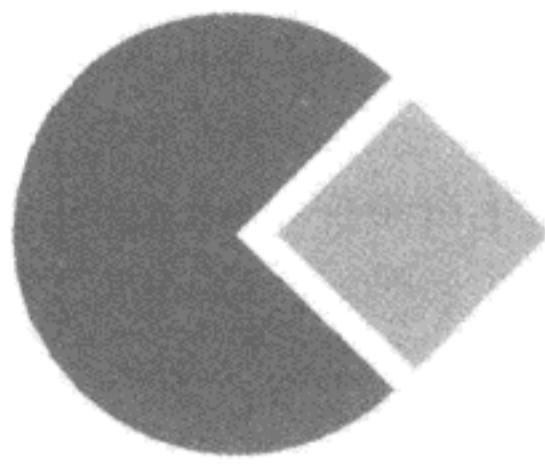
```
public void assignContactFromEmail(String emailAddress, boolean lazyLookup)  
public void assignContactFromPhone(String phoneNumber, boolean lazyLookup)
```

这两个方法分别可以通过 E-mail 和电话号码与联系人关联。如果 lazyLookup 参数值为 true，并不会立即通过 E-mail 或 Phone 查找联系人，直接 QuickContactBadge 控件被单击。如果使用这两个方法与联系人关联与 assignContactUri 方法有如下两点不同。

- 如果有多个联系人使用了同一个电话或 E-mail，则只显示第一个查到的联系人信息。
- 使用这两个方法与联系人关联，单击 QuickContactBadge 控件并不会显示图像菜单，而会直接跳到系统联系人界面。

## 6.3 小结

本章介绍了 Android SDK 支持的各种菜单（选项菜单、子菜单和上下文菜单）的创建和使用方法。除此之外，还介绍了两种自定义菜单的实现方法。这两种方法实际上都是通过弹出窗口实现了自定义菜单，只是处理弹出窗口的方式不同。通过弹出窗口和自定义布局文件，可以做出非常酷的菜单效果。在本章的最后还介绍了 QuickContactBadge 控件的使用方法，通过这个控件，可以将联系人头像和联系人快捷菜单结合在一起。



## 第7章 友好地互动交流——信息提醒 (对话框、Toast与Notification)

在 Android SDK 中提供了各种风格的信息提醒方式，最常用的毫无疑问应该是对话框。通过对对话框可以显示模式（也称为独占式，显示该窗口后，系统的其他窗口无法访问）窗口，我们可以在这个窗口上放置不同的按钮和各种控件，并向用户展示各种信息。除了对话框外，Android SDK 还允许使用另外两种方式（Toast 和 Notification）来显示提醒信息。其中 Toast 可以在任何状态下显示一个不可获得焦点的窗口。我们可以在窗口上放置任何控件，这一点和对话框很类似。Notification（通知）可以在 Android 状态栏（位于 Android 桌面的正上方，通过滑动可以拉下来）上显示图标和文字。我们会经常看到 Android 自带的软件以及第三方的应用在处于某种状态时显示的通知。例如，当我们收到短信后，系统会在状态栏显示短信的内容。

前面简单介绍了这 3 种信息提醒的方法，对于很多初学者来说，这些信息过于笼统。那么下面将逐渐为读者揭开对话框、Toast 以及 Notification 的面纱，并向读者展示关于这 3 种技术的很多高级内容。

### 7.1 对话框的基本用法

本节将向读者展示对话框的基本用法。Android 中的对话框需要使用 AlertDialog 类来显示，从该类的名字可以看出，主要用于显示提醒信息。不过这个对话框类可不仅仅能用来显示一些信息，我们可以在对话框中放置任何的控件，使其成为一个复杂且功能强大的用户接口。一个典型的例子是用 AlertDialog 做一个登录对话框。

#### 7.1.1 带 2 个按钮（确认/取消）的对话框

**工程目录:** src\ch07\ch07\_two\_button\_dialog

在介绍对话框之前，我们先来看一下 AlertDialog 类。该类并没有 public 的构造方法，因此，不能直接创建 AlertDialog 对象。为了创建 AlertDialog 对象，需要使用 Builder 类，该类是在 AlertDialog 类中定义的一个内嵌类。首先必须创建 AlertDialog.Builder 对象，然后通过 Builder.show 方法显示对话框，或通过 Builder.create 方法返回 AlertDialog 对象，再通过 AlertDialog.show 方法显示对话框。

很多程序总会问用户一些问题，例如，是否删除该文件？真的要退出程序吗？程序往往只允许用户做两种选择，也就是“Yes”或“No”，这就是一个典型的对话框。在对话框中一般只有两个

按钮（例如，“Yes”和“No”、“确定”和“取消”），以及一条简单的信息，用户单击不同按钮会根据具体的情况做出不同的处理。

显示这样的对话框的关键是如何显示两个按钮以及响应这两个按钮的单击事件，`AlertDialog`类提供了非常简单的方法来完成这个工作。通过 `AlertDialog` 类的 `setPositiveButton` 和 `setNegativeButton` 方法可以很容易地为对话框添加两个按钮。下面来看一下这两个方法的定义。

```
public Builder setPositiveButton(CharSequence text, final OnClickListener listener)
public Builder setPositiveButton(int textId, final OnClickListener listener)
public Builder setNegativeButton(int textId, final OnClickListener listener)
public Builder setNegativeButton(CharSequence text, final OnClickListener listener)
```

从上面的代码可以看出，`setPositiveButton` 和 `setNegativeButton` 方法各有两个重载形式。当然，这两个重载形式的功能完全相同，只是需要指定不同类型的按钮文本，其中 `text` 参数可直接指定文本或 `String` 变量，`textId` 参数需要指定一个字符串资源 ID（需要在 `res\values` 目录中的 `xml` 文件中定义），`listener` 参数要指定监听按钮单击事件的对象。一般来讲，使用 `setPositiveButton` 按钮来添加“确定”、“Yes”等按钮，用 `setNegativeButton` 方法来添加“取消”、“Cancel”等按钮。下面的代码用来显示一个询问用户是否下载文件的对话框。当单击“确定”或“取消”按钮后会各弹出一个没有按钮的对话框，以表明我们单击的是哪个按钮。在实际应用中需要换成相应的代码。

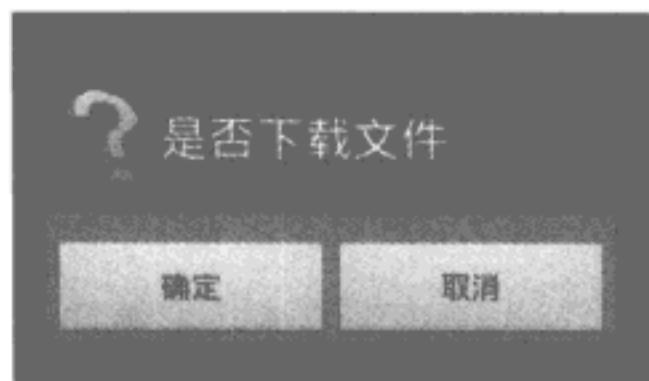
```
new AlertDialog.Builder(this).setIcon(R.drawable.question).setTitle(
    "是否下载文件").setPositiveButton("确定",
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            new AlertDialog.Builder(Main.this).setMessage("文件已经成功下载.").create().show();
        }
    }).setNegativeButton("取消",
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            new AlertDialog.Builder(Main.this).setMessage("您已经选择了取消按钮，该文件未被下载。")
                .create().show();
        }
    }).show();
```

在使用 `AlertDialog.Builder` 类来创建对话框时应注意如下几点。

- `setPositiveButton` 和 `setNegativeButton` 方法的第 2 个参数的数据类型是 `android.content.DialogInterface.OnClickListener`，而不是 `android.view.View.OnClickListener`。`View.OnClickListener` 接口是用在视图上的，这一点在使用时要注意。
- 使用 `show` 方法显示对话框是异步的。也就是说，当调用 `AlertDialog.Builder.show` 或 `AlertDialog.show` 方法显示对话框后，`show` 方法会立即返回，并且继续执行后面的代码。
- 单击使用 `setPositiveButton` 和 `setNegativeButton` 方法添加的按钮后，即使单击事件中不写任何代码，对话框也是会关闭的。

- 如果某个按钮单击后只需要关闭对话框，并不需要进行任何处理，`listener`参数值可以设为`null`。
- `AlertDialog`类还有很多`setXxx`方法用于指定对话框的其他资源。例如，本例中使用了一个`setIcon`方法来指定对话框的图标，这样会使对话框看上去更美观。

对话框的显示效果如图 7.1 所示。



▲ 图 7.1 “确定/取消”对话框

### 7.1.2 带 3 个按钮（覆盖/忽略/取消）的对话框

工程目录：src\ch07\ch07\_three\_button\_dialog

用 `AlertDialog` 类创建的对话框最多可以添加 3 个按钮。除了上一节介绍了两个添加按钮的方法外，还可以使用 `setNeutralButton` 方法向对话框添加第三个按钮，代码如下：

```
new AlertDialog.Builder(this).setIcon(R.drawable.question).setTitle(
    "是否覆盖文件? ").setPositiveButton("覆盖",
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            new AlertDialog.Builder(Main.this)
                .setMessage("文件已经覆盖.").create().show();
        }
    }).setNeutralButton("忽略", new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            new AlertDialog.Builder(Main.this).setMessage("忽略了覆盖文件的操作.")
                .create().show();
        }
    }).setNegativeButton("取消", new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            new AlertDialog.Builder(Main.this).setMessage("您已经取消了所有的操作.")
                .create().show();
        }
    }).show();
```

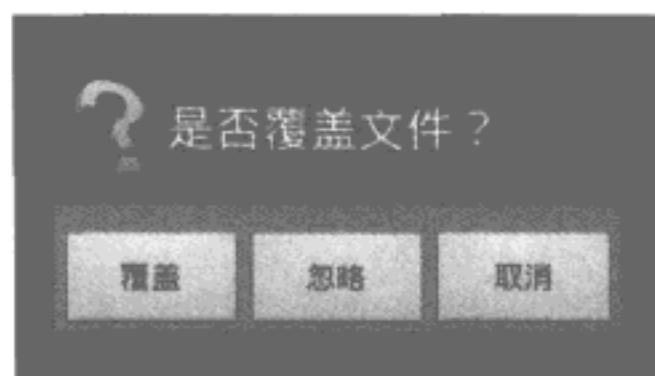
在编写上面代码时应注意如下几点。

- `setPositiveButton`、`setNeutralButton` 和 `setNegativeButton` 的调用顺序可以是任意的，但无论调用顺序是什么，使用 `setPositiveButton` 方法设置的按钮总会排在左起第 1 位，使用 `setNeutralButton`

方法设置的按钮总会排在左起第 2 位, 使用 `setNegativeButton` 方法设置的按钮总会排在左起第 3 位。

- 使用 `AlertDialog` 类创建的对话框最多只能有 3 个按钮, 因此, 就算多次调用这 3 个设置对话框按钮的方法, 最多也只能显示 3 个按钮。
- 这 3 个设置对话框按钮的方法虽然都可以调用多次, 但系统只以每一个方法最后一次调用为准。例如, `new AlertDialog.Builder(this).setPositiveButton("确定 1",...).setPositiveButton("确定 2",...)` 虽然调用了两次 `setPositiveButton` 方法, 但系统只以最后一次调用为准, 也就是说, 系统只会为对话框添加一个【确认 2】按钮, 而不会将【确认 1】和【确认 2】按钮都加到对话框上。

带 3 个按钮的对话框的显示效果如图 7.2 所示。



▲ 图 7.2 “覆盖/忽略/取消”对话框

### 7.1.3 简单列表对话框

工程目录: `src\ch07\ch07_simple_list_dialog`

通过 `AlertDialog.Builder` 类的 `setItems` 方法可以创建简单的列表对话框。实际上, 这种对话框相当于将 `ListView` 控件放在对话框上, 然后在 `ListView` 中添加若干简单的文本。`setItems` 方法的定义如下:

```
// itemsId 参数表示字符串数组的资源 ID, 该资源指定的数组会显示在列表中
public Builder setItems(int itemsId, final OnClickListener listener)
// items 参数表示用于显示在列表中的字符串数组
public Builder setItems(CharSequence[] items, final OnClickListener listener)
```

`setItems` 方法可以通过传递一个字符串数组资源 ID 或字符串数组变量或值的方式为对话框中的列表提供数据, 第 2 个参数用于监听列表项的单击事件。下面看一下 `OnClickListener.onClick` 方法的定义。

```
public void onClick(DialogInterface dialog, int which)
```

`onClick` 方法的第 1 个参数的数据类型是 `DialogInterface`, 由于 `AlertDialog` 类实现了 `DialogInterface` 接口, 因此, 该参数值可以是 `AlertDialog` 对象。在 `DialogInterface` 接口中又有两个用于关闭对话框的方法: `dismiss` 和 `cancel`。这两个方法的功能完全相同, 都是关闭对话框。所不同的是, `cancel` 方法除了关闭对话框外, 还会调用 `DialogInterface.OnCancelListener.onCancel` 方法。`DialogInterface.OnCancelListener` 对象需要使用 `AlertDialog.Builder.setOnCancelListener` 方法进行设置。`dismiss` 与 `cancel` 方法类似, 调用 `dismiss` 方法不仅会关闭对话框, 还会调用 `DialogInterface.OnDismissListener.onDismiss` 方法。除了 `dismiss` 和 `cancel` 方法外, 还有几个常量, 这些常量分别用于表示对话框的 3 个按钮的 ID。下面的代码弹出了一个显示省份的列表对话框。

## 第7章 友好地互动交流——信息提醒（对话框、Toast与Notification）

```

new AlertDialog.Builder(this).setTitle("选择省份")
.setItems(provinces, new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int which)
    {
        final AlertDialog ad = new AlertDialog.Builder(
            Main.this).setMessage(
            "您已经选择了：" + which + ":" + provinces[which])
        .show();
        android.os.Handler hander = new android.os.Handler();
        // 设置定时器，5秒后调用run方法
        hander.postDelayed(new Runnable()
        {
            @Override
            public void run()
            {
                // 调用AlertDialog.dismiss方法关闭对话框，也可以调用cancel方法
                ad.dismiss();
            }
        }, 5 * 1000); // postDelayed方法的第2个参数表示延迟调用run方法的时间，单位：毫秒
    }
}).show();

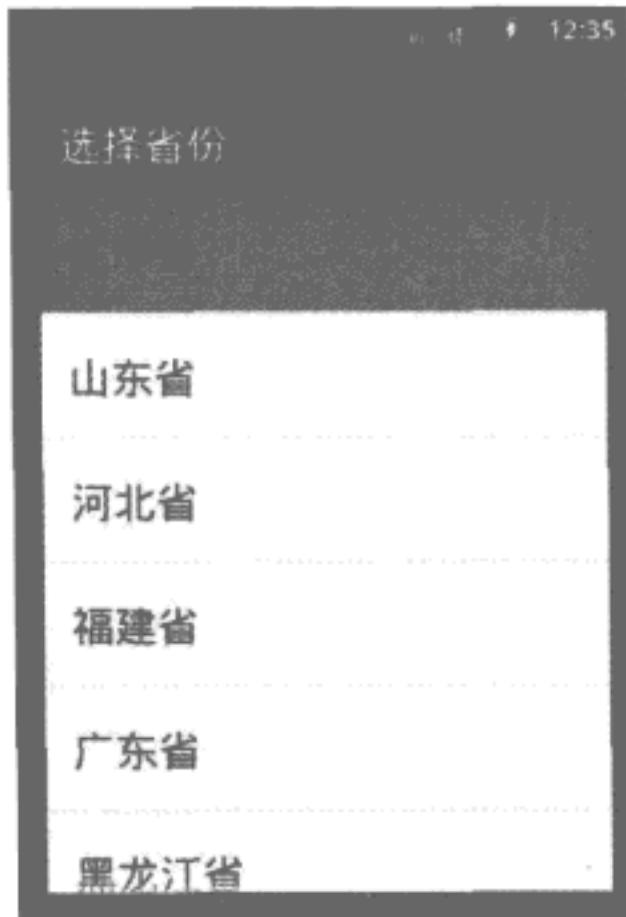
```

上面代码涉及一个 provinces 变量，该变量是 String 类型的数组，定义了列表对话框所需的数据，该变量的定义代码如下：

```
private String[] provinces = new String[]{"辽宁省", "山东省", "河北省", "福建省", "广东省", "黑龙江省"};
```

单击某个列表项后会弹出一个没有按钮的对话框，以便让我们知道单击的是哪个列表项。同时在代码中添加了一个定时器，5秒后这个没有按钮的对话框会自动关闭。

简单列表对话框的显示效果如图 7.3 所示。



▲ 图 7.3 简单列表对话框

### 7.1.4 单选列表对话框

**工程目录:** src\ch07\ch07\_single\_choice\_dialog

通过 AlertDialog.Builder 类的 setSingleChoiceItems 方法可以创建带单选按钮的列表对话框。setSingleChoiceItems 方法有如下 4 个重载形式：

```
// 从资源文件中装载数据
public Builder setSingleChoiceItems(int itemsId, int checkedItem, final OnClickListener
listener)
// 从数据集中装载数据
public Builder setSingleChoiceItems(Cursor cursor, int checkedItem, String labelColumn,
final OnClickListener listener)
// 从字符串数组中装载数据
public Builder setSingleChoiceItems(CharSequence[] items, int checkedItem, final
OnClickListener listener)
// 从ListAdapter 对象中装载数据
public Builder setSingleChoiceItems(ListAdapter adapter, int checkedItem, final
OnClickListener listener)
```

上面 4 个重载形式除了第 1 个参数外，其他的参数完全一样。这些参数的含义如下：

- 第 1 个参数：表示单选列表对话框的数据源。目前支持 4 种数据源，分别是数组资源 (itemsId)、数据集 (cursor)、字符串数组 (items) 和 ListAdapter 对象。
- checkedItem：表示默认选中的列表项。如果设置第 1 个列表项为选中状态，该参数值为 0。如果该参数值小于 0，表示所有的列表项都未被选中。
- listener：表示单击某个列表项时被触发的事件对象。
- labelColumn：如果数据源是数据集，数据集中的某一列会作为列表对话框的数据加载到列表控件中。该参数表示该列的名称（字段名）。

下面的代码显示一个带单选按钮的列表对话框。

```
new AlertDialog.Builder(this).setTitle("选择省份")
.setSingleChoiceItems(provinces, -1, new OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog, int which)
    {
        // index 是一个 int 类型的类变量，保存当前选中的列表项索引
        index = which;
    }
}).setPositiveButton("确定", new OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog, int which)
    {
        new AlertDialog.Builder(Main.this).setMessage(
            "您已经选择了：" + index + ":" + provinces[index])
        .show();
    }
}).setNegativeButton("取消", new OnClickListener()
```

```

{
    @Override
    public void onClick(DialogInterface dialog, int which)
    {
        new AlertDialog.Builder(Main.this).setMessage("您什么都未选择.").show();
    }
}).show();

```

单选列表对话框的效果如图 7.4 所示。当选中某个列表项后，再单击“确定”按钮，会弹出如图 7.5 所示的提示信息。



▲ 图 7.4 单选列表对话框



▲ 图 7.5 选中某个列表项后的提示信息

### 7.1.5 多选列表对话框

工程目录: src\ch07\ch07\_multi\_choice\_dialog

通过 `AlertDialog.Builder.setMultiChoiceItems` 方法可以创建带复选框的列表对话框。`setMultiChoiceItems` 方法有如下 3 个重载形式:

```

// 从资源文件中装载数据
public Builder setMultiChoiceItems(int itemId, boolean[] checkedItems, final
OnMultiChoiceClickListener listener)
// 从数据集中装载数据
public Builder setMultiChoiceItems(Cursor cursor, String isCheckedColumn, String
labelColumn, final OnMultiChoiceClickListener listener)
// 从字符串数组中装载数据
public Builder setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems,
final OnMultiChoiceClickListener listener)

```

上面 3 个重载形式除了第 1 个参数外，其他参数完全一样。这些参数的含义如下。

- 第 1 个参数：表示多选列表对话框的数据源。目前支持 3 种数据源，分别是数组资源、数

据集和字符串数组。

- **checkedItems:** 该参数的数据类型是 boolean[], 这个参数值的数组长度要和列表框中的列表项个数相同，该参数用于设置每一个列表项的默认值，如果为 true，表示当前的列表项是选中状态，否则表示未选中状态。
- **listener:** 表示选中某一个列表项时被触发的事件对象。
- **isCheckedColumn:** 该参数只用于数据集（Cursor）数据源，用于指定数据集的一列（字段名），该列必须是 Integer 类型。如果该列的值为 1，则同一记录 labelColumn 指定的列对应的列表项被选中；如果为 0，则未被选中。也就是说，对于数据集来说，某个列表项是否被选中，是由另外一列的字段值决定的。
- **labelColumn:** 只用于数据集。指定用于显示列表项的列的字段名。

下面的代码显示了一个多选列表对话框。

```
AlertDialog ad = new AlertDialog.Builder(this)
    .setIcon(R.drawable.image)
    .setTitle("选择省份")
    .setMultiChoiceItems(provinces, new boolean[]
    { false, true, false, true, false, false },
        new DialogInterface.OnMultiChoiceClickListener()
    {
        public void onClick(DialogInterface dialog,
            int whichButton, boolean isChecked)
        {
        }
    })
    .setPositiveButton("确定", new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int whichButton)
        {
            int count = lv.getCount();
            String s = "您选择了:";
            for (int i = 0; i < provinces.length; i++)
            {
                if (lv.getCheckedItemPositions().get(i))
                    s += i + ":" + lv.getAdapter().getItem(i) + " ";
            }
            if (lv.getCheckedItemPositions().size() > 0)
            {
                new AlertDialog.Builder(Main.this).setMessage(s).show();
            }
            else
            {
                new AlertDialog.Builder(Main.this).setMessage("您未选择任何省份").show();
            }
        }
    })
    .setNegativeButton("取消", null).create();
lv = ad.getListView();
ad.show();
```

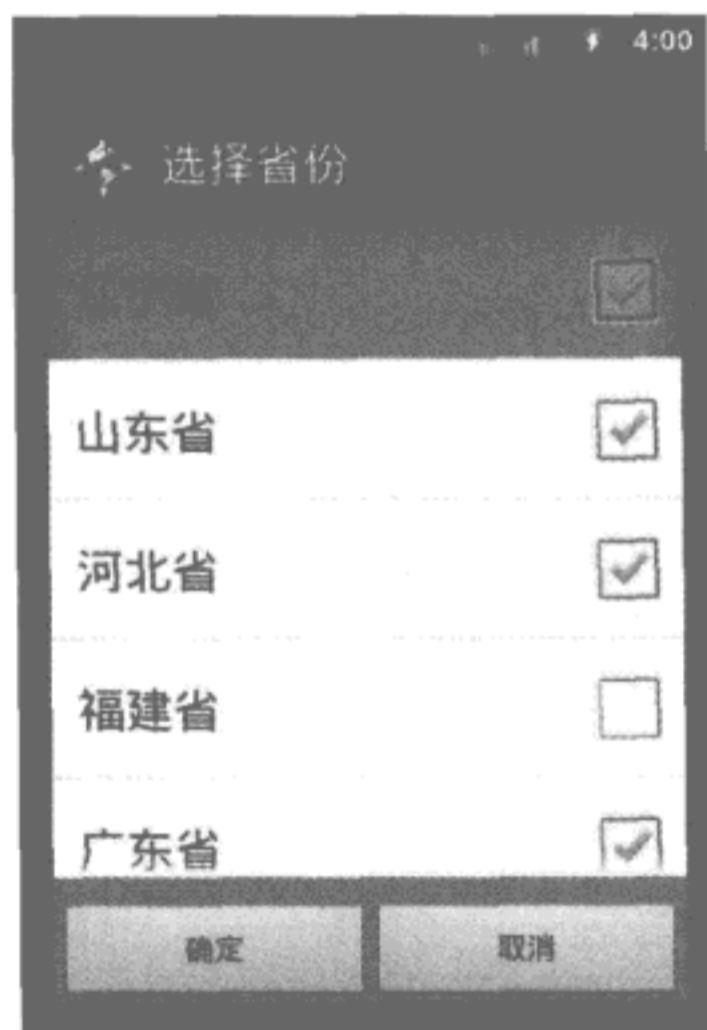
在编写上面代码时应注意如下几点。



- 必须指定 `setMultiChoiceItems` 方法的单击事件对象，也就是该方法的第 3 个参数，该参数值不能为 null，否则默认被选中的列表项无法置成未选中状态。对于默认未被选中的列表项没有任何影响。

- 由于在“确定”按钮的单击事件中需要引用 `AlertDialog` 变量，因此，先使用 `create` 方法返回 `AlertDialog` 对象，然后在单击事件中使用该变量。

多选列表对话框的显示效果如图 7.6 所示。选中一些列表项后，单击“确定”按钮，会弹出如图 7.7 所示的提示信息。



▲ 图 7.6 多选列表对话框



▲ 图 7.7 选中多个列表项后的提示信息

### 7.1.6 进度对话框

工程目录: `src\ch07\ch07_progressbar_dialog`

进度对话框通过 `android.app ProgressDialog` 类实现，该类是 `AlertDialog` 的子类，但与 `AlertDialog` 类不同，我们可以直接使用 `new` 关键字创建 `ProgressDialog` 对象。

进度对话框除了需要设置普通对话框必要的值外，还需要设置另外两个值：进度的最大值和当前的进度。这两个值分别由如下两个方法来设置：

```
// 设置进度的最大值
public void setMax(int max)
// 设置当前的进度
public void setProgress(int value)
```

初始进度必须使用 `setProgress` 方法设置，而递增进度除了可以使用 `setProgress` 方法设置外，还可以使用如下方法设置：

```
// 设置进度的递增量
public void incrementProgressBy(int diff)
```

`setProgress` 和 `incrementProgressBy` 方法的区别是 `setProgress` 方法设置的是进度的绝对值，而 `incrementProgressBy` 方法设置的是进度的增量。

与普通对话框一样，进度对话框最多也可以添加 3 个按钮，而且可以设置进度对话框的风格。进度对话框中进度条的默认风格是圆形的，可以使用如下代码将进度条风格设置成水平进度条风格：

```
// 创建 ProgressDialog 类的对象实例
ProgressDialog progressDialog = new ProgressDialog(this);
// 设置进度对话框为水平进度条风格
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```

下面我们看一个进度对话框的例子，在这个例子中演示了水平和圆形进度对话框的实现方法。其中进度对话框包含两个按钮：“暂停”和“取消”，单击“暂停”按钮后，进度对话框关闭，再次显示进度对话框时，进度条的起始位置从上一次关闭对话框的位置开始（仅限于水平进度条）。单击“取消”按钮后，进度对话框也会关闭，只是再次显示进度对话框时，进度的起始位置仍然从 0 开始。

要实现进度随着时间的变化而不断递增，需要使用多线程及定时器来完成这个工作。本例中使用 `Handler` 类来不断更新进度对话框的进度值。实现的方法是编写一个继承自 `Handler` 的类，并覆盖 `Handler.handleMessage` 方法。在该方法中设置新的进度和系统下一次调用 `handleMessage` 方法的时间间隔（单位是毫秒）。本例的实现代码如下：

```
package mobile.android.ch07.progressbar.dialog;

import java.util.Random;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class Main extends Activity implements OnClickListener
{
    // 最大进度
    private static final int MAX_PROGRESS = 100;
    private ProgressDialog progressDialog;
    private Handler progressHandler;
    private int progress;
    // 显示进度对话框，style 表示进度对话框的风格
    private void showProgressDialog(int style)
    {
        progressDialog = new ProgressDialog(this);
        progressDialog.setIcon(R.drawable.wait);
        progressDialog.setTitle("正在处理数据...");
        progressDialog.setMessage("请稍后...");
        // 设置进度对话框的风格
```

```
progressDialog.setProgressStyle(style);
// 设置进度对话框的进度最大值
progressDialog.setMax(MAX_PROGRESS);
// 设置进度对话框的“暂停”按钮
progressDialog.setButton("暂停", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int whichButton)
    {
        // 通过删除消息代码的方式停止定时器
        progressHandler.removeMessages(1);
    }
});
// 设置进度对话框的“取消”按钮
progressDialog.setButton2("取消", new DialogInterface.OnClickListener()
{
    public void onClick(DialogInterface dialog, int whichButton)
    {
        // 通过删除消息代码的方式停止定时器的执行
        progressHandler.removeMessages(1);
        // 恢复进度初始值
        progress = 0;
        progressDialog.setProgress(0);
    }
});
progressDialog.show();
progressHandler = new Handler()
{
    @Override
    public void handleMessage(Message msg)
    {
        super.handleMessage(msg);
        if (progress >= MAX_PROGRESS)
        {
            // 进度达到最大值，关闭对话框
            progress = 0;
            progressDialog.dismiss();
        }
        else
        {
            progress++;
            // 将进度递增 1
            progressDialog.incrementProgressBy(1);
            // 随机设置下一次递增进度（调用 handleMessage 方法）的时间（50+毫秒）
            progressHandler.sendEmptyMessageDelayed(1, 50 + new Random().nextInt(500));
        }
    }
};
// 设置进度初始值
progress = (progress > 0) ? progress : 0;
progressDialog.setProgress(progress);
progressHandler.sendEmptyMessage(1);
}
```

## Android 开发权威指南

```

@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.button1:
            // 显示进度条风格的进度对话框
            showProgressDialog(ProgressDialog.STYLE_HORIZONTAL);
            break;
        case R.id.button2:
            // 显示旋转指针风格的进度对话框
            showProgressDialog(ProgressDialog.STYLE_SPINNER);
            break;
    }
}
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Button button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(this);
    Button button2 = (Button) findViewById(R.id.button2);
    button2.setOnClickListener(this);
}
}

```

在编写上面代码时需要注意如下几点。

- 进度对话框在默认情况下是圆形进度条，如果要显示水平进度条，需要使用 `setProgressStyle` 方法进行设置。
- 调用 `sendEmptyMessage` 方法只能使 `handleMessage` 方法执行一次，要想以一定时间间隔循环执行 `handleMessage` 方法，需要在 `handleMessage` 方法中调用 `sendEmptyMessageDelayed` 方法来设置 `handleMessage` 方法下一次被调用之前的等待时间，这样就可以形成一个循环调用的效果。
- `sendEmptyMessage` 和 `sendEmptyMessageDelayed` 方法的第一个参数表示消息代码，这个消息代码用来标识消息队列中的消息。例如，使用 `sendEmptyMessageDelayed` 方法设置消息代码为 1 的消息在 (50+) 毫秒后调用 `handleMessage` 方法。可以利用这个消息代码删除该消息，这样系统就不会在 (50+) 毫秒之后调用 `handleMessage` 方法了。在本例的“暂停”和“取消”按钮单击事件中都使用 `removeMessages` 方法删除了消息代码为 1 的消息。
- 消息代码可以是任何 `int` 类型的值。
- 虽然 `ProgressDialog.getProgress` 方法可以获得当前进度，但只有在水平进度条风格的对话框中才有效。如果是圆形进度条，该方法永远返回 0。这就是在本例中为什么单独使用了一个 `progress` 变量来表示当前进度，而不使用 `getProgress` 方法来获得当前进度的原因。如果使用 `getProgress` 方法来代替 `progress` 变量，当进度条风格是圆形时，就意味着对话框将永远不会被关闭。
- 圆形进度对话框中的进度圆圈只是一个普通的动画，并没有任何表示进度的功能。这种对话框一般在很难估计准确时间和进度时使用。

## 第7章 友好地互动交流——信息提醒（对话框、Toast与Notification）

运行本例后，屏幕上将显示两个按钮，单击这两个按钮后，会分别显示水平和圆形进度对话框，效果如图 7.8 和图 7.9 所示。



▲ 图 7.8 水平风格进度对话框



▲ 图 7.9 圆形风格进度对话框

### 7.1.7 登录对话框

**工程目录:** src\ch07\ch07\_login\_dialog

虽然 AlertDialog 类提供了很多预定义的对话框，但这些对话框仍然不能完全满足系统的需求。为了创建更丰富的对话框，也可以采用与创建 Activity 同样的方法。也就是说，直接使用布局文件或代码创建视图对象，并将这些视图对象添加到对话框中。

AlertDialog.Builder.setView 方法可以将视图对象添加到当前的对话框中。可以使用下面的形式将一个视图对象添加到对话框中：

```
new AlertDialog.Builder(this)
    .setIcon(R.drawable.alert_dialog_icon).setTitle("自定义对话框")
    .setView(... ...)
    .show();
```

下面我们利用 setView 方法来设置一个登录对话框。本例使用布局文件对界面进行布局，代码如下：

#### login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 用户名文本输入框 -->
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

## Android 开发权威指南

```

    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp">
    <TextView android:layout_width="wrap_content" android:text="用户名: "
        android:textSize="20sp" android:layout_height="wrap_content" />
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
<!-- 密码文本输入框 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp">
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="密      码: "
        android:textSize="20sp" />
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:password="true" />
</LinearLayout>
</LinearLayout>

```

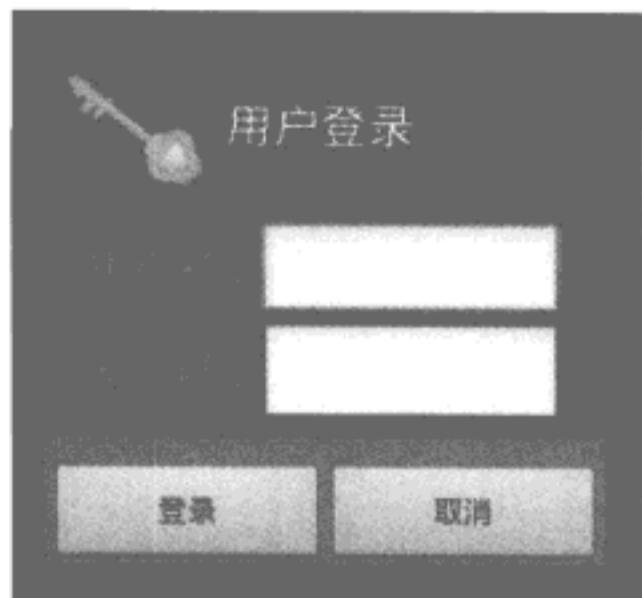
本例的实现代码如下：

```

// 从 login.xml 布局文件中装载 LinearLayout 对象
LinearLayout loginLayout = (LinearLayout) getLayoutInflater().inflate(R.layout.login,
null);
new AlertDialog.Builder(this).setIcon(R.drawable.login)
    .setTitle("用户登录").setView(loginLayout).setPositiveButton("登录",
        new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog,
            int whichButton)
        {
            // 编写处理用户登录的代码
        }
    }).setNegativeButton("取消",
        new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog,
            int whichButton)
        {
            // 取消用户登录，退出程序
        }
    }).show();

```

运行本例后，屏幕上会显示一个按钮，单击该按钮，将显示如图 7.10 所示的对话框。



▲ 图 7.10 登录对话框

### 7.1.8 使用 Activity 托管对话框

工程目录: src\ch07\ch07\_activity\_dialog

Activity 类提供了创建对话框的快捷方式。在 Activity 类中有一个 `onCreateDialog` 方法，该方法的定义如下：

```
protected Dialog onCreateDialog(int id)
```

当调用 `Activity.showDialog` 方法时，系统会调用 `onCreateDialog` 方法来返回一个 `Dialog` 对象（`AlertDialog` 是 `Dialog` 的子类）。`showDialog` 方法和 `onCreateDialog` 方法一样，也有一个 `int` 类型的 `id` 参数。该参数值将传入 `onCreateDialog` 方法。可以利用不同的 `id` 来建立多个对话框。

**注意** 对于同一个对话框的 ID，系统只在第 1 次调用 `showDialog` 方法时调用 `onCreateDialog` 方法。在第 1 次创建 `Dialog` 对象时系统会将该对象保存在 `Activity` 的缓存中，相当于一个 `Map` 对象，对话框的 ID 作为 `Map` 的 key，而 `Dialog` 对象作为 `Map` 的 value。当再次调用 `showDialog` 方法时，系统会根据 ID 从这个 `Map` 对象中获得第 1 次创建的 `Dialog` 对象，而不会再次调用 `onCreateDialog` 方法创建新的 `Dialog` 对象。除非调用 `Activity` 类的 `removeDialog(int id)` 方法删除了指定 ID 的 `Dialog` 对象。

下面我们看一个如何通过 `showDialog` 和 `onCreateDialog` 方法创建对话框的例子。

```
package mobile.android.ch07.activity.dialog;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ListView;

public class Main extends Activity implements OnClickListener {
```

```

private final int DIALOG_DELETE_FILE = 1;
private final int DIALOG_SIMPLE_LIST = 2;
private final int DIALOG_SINGLE_CHOICE_LIST = 3;
private final int DIALOG_MULTI_CHOICE_LIST = 4;
    ...
@Override
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.btnDeleteFile:
            showDialog(DIALOG_DELETE_FILE); // 显示删除文件确认对话框
            break;
        case R.id.btnSimpleList:
            showDialog(DIALOG_SIMPLE_LIST); // 显示简单列表对话框
            break;
        case R.id.btnSingleChoiceList:
            showDialog(DIALOG_SINGLE_CHOICE_LIST); // 显示单选列表对话框
            break;
        case R.id.btnMultiChoiceList:
            showDialog(DIALOG_MULTI_CHOICE_LIST); // 显示多选列表对话框
            break;
        case R.id.btnRemoveDialog:
            // 将所有的对话框从 Activity 的托管中删除
            removeDialog(DIALOG_DELETE_FILE);
            removeDialog(DIALOG_SIMPLE_LIST);
            removeDialog(DIALOG_SINGLE_CHOICE_LIST);
            removeDialog(DIALOG_MULTI_CHOICE_LIST);
            break;
    }
}
@Override
protected Dialog onCreateDialog(int id)
{
    // 根据不同的 id 创建相应的 Dialog 对象
    switch (id)
    {
        case DIALOG_DELETE_FILE:
            return new AlertDialog.Builder(this).... .create();
        case DIALOG_SIMPLE_LIST:
            return new AlertDialog.Builder(this).... .create();
        case DIALOG_SINGLE_CHOICE_LIST:
            return new AlertDialog.Builder(this).... .create();
        case DIALOG_MULTI_CHOICE_LIST:
            return new AlertDialog.Builder(this).... .create();
    }
    return null;
}
...
}

```

运行本例后，在屏幕上将显示 5 个按钮，前 4 个按钮分别显示 4 个对话框，最后一个按钮将所有的对话框从 Activity 的托管缓存中删除。



除了创建和显示对话框外，还可以使用 `Activity.dismissDialog` 方法关闭指定 ID 的对话框。如果想在对话框显示之前进行一些初始化，可以使用 `onPrepareDialog` 事件方法。该方法的定义如下：

```
protected void onPrepareDialog(int id, Dialog dialog)
```

## 7.2 对话框的高级应用

显示一个对话框并不复杂。但要想进行更多的控制，就需要使用一些技巧。例如，在单击对话框按钮后会自动关闭对话框，虽然这是默认动作，但我们可以利用反射技术来阻止对话框的关闭。除此之外，还可以修改对话框的外观（系统自带的对话框样式实在是有些简陋）、设置对话框的透明度等。

### 7.2.1 阻止单击按钮关闭对话框

**工程目录:** src\ch07\ch07\_prevent\_close\_dialog

在 7.1 节的很多例子中已经知道，单击对话框的按钮后会自动关闭对话框。那么在某些情况下，单击按钮后并不想关闭（至少是不想立刻关闭）对话框，这种需求使用 `AlertDialog` 是否能做到呢（虽然我们可以自己弄个 `Activity` 来完成这个工作，也可在 `View` 上自己放按钮，但这显示有些大炮打蚊子了，如果对话框上只有一行文本，费这么多劲太不值了）？遗憾的是通过 `AlertDialog` 暴露出来的 API 是无法做到的，但这个世界上的事只有想不到的，没有做不到的。首先我们来解剖一下 `AlertDialog`，看看为什么通过正常的途径无法做到。然后再按图索骥，来达到我们的目的。

既然要控制对放框的关闭行为，首先就得分析是哪些类、哪些代码使这个对话框关闭的。进入 `AlertDialog` 类的源代码。在 `AlertDialog` 类中只定义了一个变量：`mAlert`。这个变量的类型是 `AlertController`。`AlertController` 类是 `Android` 的内部类，在 `com.android.internal.app` 包中，无法通过正常的方式访问。也无法在 `Eclipse` 中通过按 `Ctrl` 键跟踪进源代码。但可以直接在 `Android` 源代码中找到 `AlertController.java` 文件。我们再回到 `AlertDialog` 类中。`AlertDialog` 类实际上只是一个架子，像设置按钮、设置标题等工作都是由 `AlertController` 类完成的，因此，`AlertController` 类才是关键。

找到 `AlertController.java` 文件（通过 `Windows` 或其他操作系统的文件搜索功能很容易在 `Android` 源代码中找到该文件）。打开后不要感到头晕哦，这个文件中的代码是很多的！不过这么多代码对本节的主题也没什么用处，并不需要过多地关注它们。下面就找一下控制按钮的代码。

在 `AlertController` 类的开头就会看到如下的代码：

```
View.OnClickListener mButtonHandler = new View.OnClickListener() {
    public void onClick(View v) {
        Message m = null;
        if (v == mButtonPositive && mButtonPositiveMessage != null) {
            m = Message.obtain(mButtonPositiveMessage);
        } else if (v == mButtonNegative && mButtonNegativeMessage != null) {
            m = Message.obtain(mButtonNegativeMessage);
        } else if (v == mButtonNeutral && mButtonNeutralMessage != null) {
            m = Message.obtain(mButtonNeutralMessage);
        }
        if (m != null) {
```

## Android 开发权威指南

```

        // 发送消息，以便调用按钮的单击事件方法
        m.sendToTarget();
    }
    // Post a message so we dismiss after the above handlers are executed
    mHandler.obtainMessage(ButtonHandler.MSG_DISMISS_DIALOG, mDialogInterface)
        .sendToTarget();
}
;

```

从这段代码中可以猜出来，前几行代码用来触发对话框中的 3 个按钮（Positive、Negative 和 Neutral）的单击事件，而最后的代码则用来关闭对话框（因为我们发现了 MSG\_DISMISS\_DIALOG、猜出来的）。

```

mHandler.obtainMessage(ButtonHandler.MSG_DISMISS_DIALOG,
mDialogInterface).sendToTarget();

```

上面的代码并不是直接来关闭对话框的，而是通过一个 Handler 来处理，代码如下：

```

private static final class ButtonHandler extends Handler {
    // Button clicks have Message.what as the BUTTON{1,2,3} constant
    private static final int MSG_DISMISS_DIALOG = 1;
    private WeakReference<DialogInterface> mDialog;
    public ButtonHandler(DialogInterface dialog) {
        mDialog = new WeakReference<DialogInterface>(dialog);
    }
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case DialogInterface.BUTTON_POSITIVE:
            case DialogInterface.BUTTON_NEGATIVE:
            case DialogInterface.BUTTON_NEUTRAL:
                ((DialogInterface.OnClickListener) msg.obj).onClick(mDialog.get(), msg.what);
                break;
            case MSG_DISMISS_DIALOG:
                ((DialogInterface) msg.obj).dismiss();
        }
    }
}

```

从上面代码的最后可以找到 `((DialogInterface) msg.obj).dismiss();`。现在看了这么多源代码，我们来总结一下对话框按钮单击事件的处理过程。在 AlertController 处理对话框按钮时会为每一个按钮添加一个 onClick 事件，而这个事件类的对象就是上面的 mButtonHandler。在这个单击事件中首先会通过发送消息的方式调用为按钮设置的单击事件（也就是通过 setPositiveButton 等方法的第二个参数设置的单击事件），在触发完按钮的单击事件后，会通过发送消息的方式调用 dismiss 方法来关闭对话框。而在 AlertController 类中定义了一个全局的 ButtonHandler 类型的 mHandler 变量。因此，只要使用我们自己 Handler 对象替换 ButtonHandler 就可以阻止调用 dismiss 方法来关闭对话框。下面先在自己的程序中建立一个新的 ButtonHandler 类（也可叫其他的类名）。

```

class ButtonHandler extends Handler {
    private WeakReference<DialogInterface> mDialog;
    public ButtonHandler(DialogInterface dialog)

```

```

    {
        mDialog = new WeakReference<DialogInterface>(dialog);
    }
    @Override
    public void handleMessage(Message msg)
    {
        switch (msg.what)
        {
            case DialogInterface.BUTTON_POSITIVE:
            case DialogInterface.BUTTON_NEGATIVE:
            case DialogInterface.BUTTON_NEUTRAL:
                ((DialogInterface.OnClickListener) msg.obj).onClick(mDialog.get(), msg.what);
                break;
        }
    }
}

```

我们可以看到，上面的类和 AlertController 中的 ButtonHandler 类很像，只是去掉了 switch 语句的最后一个 case 子句（用于调用 dismiss 方法）和相关的代码。

下面我们就要为 AlertController 中的 mHandler 变量重新赋值。由于 mHandler 是 private 变量，因此，在这里需要使用 Java 的反射技术来为 mHandler 赋值。由于在 AlertDialog 类中的 mAlert 变量同样也是 private，因此，也需要使用同样的反射技术来获得 mAlert 变量。代码如下：

先建立一个 AlertDialog 对象：

```

AlertDialog alertDialog = new AlertDialog.Builder( this )
    .setTitle( "我来了！" )
    .setMessage( "对话框内容" )
    .setIcon(R.drawable.icon)
    .setPositiveButton( "确定",
        new OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog,
                            int which)
        {
        }
    })
    .setNegativeButton( "取消" , new OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog, int which)
        {
            dialog.dismiss();
        }
    })
.create()

```

上面的对话框很普通，单击哪个按钮都会关闭对话框。接下来在调用 show 方法之前来修改一下 mHandler 变量的值，OK，下面让我们一起来见证奇迹。

```

try
{
    Field field = alertDialog.getClass().getDeclaredField("mAlert");

```

```

// 允许访问 private 变量
field.setAccessible(true);
// 获得 mAlert 变量的值
Object obj = field.get(alertDialog);
field = obj.getClass().getDeclaredField("mHandler");
field.setAccessible(true);
// 修改 mHandler 变量的值，使用新的 ButtonHandler 类
field.set(obj, new ButtonHandler(alertDialog));
}
catch (Exception e)
{
}
// 显示对话框
alertDialog.show();

```

我们发现，如果在创建对话框后运行上面的代码，单击对话框中的“确定”按钮不会关闭对话框（除非在代码中调用 `dismiss` 方法），单击取消按钮则会关闭对话框（因为调用了 `dismiss` 方法）。如果去了上面的代码，对话框又会恢复正常了。

虽然上面的代码已经解决了问题，但需要编写的代码仍然比较多，为此，我们也可采用另外一种方法来阻止关闭对话框，这种方法不需要定义任何的类。但这种方法需要用点技巧。由于系统通过调用 `dismiss` 来关闭对话框，那么我们可以在 `dismiss` 方法上做点文章。在系统调用 `dismiss` 方法时会首先判断对话框是否已经关闭，如果对话框已经关闭了，就会退出 `dismiss` 方法而不再继续关闭对话框了。因此，我们可以欺骗一下系统，当调用 `dismiss` 方法时我们可以让系统以为对话框已经关闭（虽然对话框还没有关闭），这样 `dismiss` 方法就失效了，因此也就无法调用 `dismiss` 方法来关闭对话框了。

下面让我们回到 `AlertDialog` 类的源代码中，再继续跟踪到 `Dialog` 类的源代码中。找到 `dismissDialog` 方法。查看 `dismiss` 方法的代码可知，`dismiss` 方法是通过 `dismissDialog` 方法来关闭对话框的，`dismissDialog` 方法的代码如下：

```

private void dismissDialog() {
    if (mDecor == null) {
        if (Config.LOGV) Log.v(LOG_TAG,
            " [Dialog] dismiss: already dismissed, ignore " );
        return ;
    }
    if ( ! mShowing) {
        if (Config.LOGV) Log.v(LOG_TAG,
            " [Dialog] dismiss: not showing, ignore " );
        return ;
    }
    mWindowManager.removeView(mDecor);
    mDecor = null ;
    mWindow.closeAllPanels();
    onStop();
    mShowing = false ;
    sendDismissMessage();
}

```

该方法后面的代码不用管它，先看 `if(!mShowing){ ... }` 这段代码。这个 `mShowing` 就是用于表示对话框是否已关闭的变量。因此，我们在代码中通过设置这个变量就可以使系统认为对话框已经关闭，就不再继续关闭对话框了。由于 `mShowing` 也是 `private` 变量，因此，也需要使用反射技术来设置这个变量。我们可以在对话框按钮的单击事件中设置 `mShowing`，代码如下：

```
try
{
    Field field = dialog.getClass().getSuperclass()
        .getDeclaredField("mShowing");
    field.setAccessible(true);
    // 将 mShowing 变量设为 false，好让系统认为这个对话框已经关闭
    field.set(dialog, false);
    // 这时再调用 dismiss 方法就失效了
    dialog.dismiss();
}
catch (Exception e)
{
}
```

从刚才分析过的 `AlertController` 类的源代码可知，在系统为对话框按钮添加的单击事件中主要完成了两部分工作。第一部分就是调用了由 `setPositiveButton`、`setNegativeButton` 和 `setNegativeButton` 指定的单击事件方法。第二部分则是关闭对话框。因此，上面的代码会在系统关闭对话框之前执行，所以在系统关闭对话框时，`mShowing` 变量的值已经被上面的代码修改了。

将上面的代码加到“确定”按钮的单击事件方法中，即使调用了 `dismiss` 方法也无法关闭对话框。如果要关闭对话框，只需再将 `mShowing` 设为 `true` 即可。要注意的是，在一个按钮单击事件方法里设置了 `mShowing` 变量，也会影响另一个按钮的关闭对话框功能，因此，需要在每一个按钮的单击事件里都设置 `mShowing` 变量的值。当然，为了方便，可以将这个功能封装在一个方法里，感兴趣的读者可以自己来完成这个工作。

## 7.2.2 改变对话框的显示位置

**工程目录：src\ch07\ch07\_position\_dialog**

在前面章节涉及的对话框都是在屏幕中心显示的，实际上，也可以根据需要在屏幕的上、下、左、右，甚至是任意位置显示对话框。

要想控制对话框的显示位置，需要获得对话框的 `Window` 对象，并通过 `Window` 对象的一些方法来控制对话框的显示位置。例如，下面的代码使对话框在屏幕的上方水平居中显示（不是正好顶屏幕上边缘，还有一段距离）。

```
AlertDialog alertDialog = new AlertDialog.Builder(this)
    .setMessage("在顶端显示对话框").setPositiveButton("确定", null).create();
Window window = alertDialog.getWindow();
// 调用 setGravity 方法使对话框在屏幕顶端显示
window.setGravity(Gravity.TOP);
alertDialog.show();
```

如果想在屏幕的其他位置显示，可以使用其他的常量，例如，左侧（`Gravity.LEFT`）、右侧

(Gravity.RIGHT) 和底端 (Gravity.BOTTOM)。如果想同时设置多个位置参数，例如，想让对话框同时在左上方显示，需要将多个与窗口位置相关的参数值进行位或操作，代码如下：

```
AlertDialog alertDialog = new AlertDialog.Builder(this)
    .setMessage("在顶端显示对话框").setPositiveButton("确定", null).create();
Window window = alertDialog.getWindow();
// 调用 setGravity 方法使对话框在屏幕左上方显示
window.setGravity(Gravity.TOP | Gravity.LEFT);
alertDialog.show();
```

通过设置 WindowManager.LayoutParams.x 和 WindowManager.LayoutParams.y 还可以使对话框在任意位置显示。其中 x 和 y 并不是相对于屏幕的绝对坐标，而是相对于对话框在中心位置显示的偏移量。x = 0、y = 0，对话框正好在屏幕中心显示。x < 0，对话框向左移动；x > 0，对话框向右移动。y < 0，对话框向上移动；y > 0，对话框向下移动。下面的代码将对话框向左偏移了 20 像素的位置，向上偏移了 120 个像素的位置。

```
AlertDialog alertDialog = new AlertDialog.Builder(this).setMessage("在任意位置显示对话框")
    .setPositiveButton("确定", null).create();
Window window = alertDialog.getWindow();
WindowManager.LayoutParams lp = window.getAttributes();
// 设置水平偏移量
lp.x = -20;
// 设置垂直偏移量
lp.y = -120;
window.setAttributes(lp);
alertDialog.show();
```

对话框在顶端、底端及任意位置显示的效果如图 7.11、图 7.12 和图 7.13 所示。



▲ 图 7.11 在屏幕顶端显示对话框



▲ 图 7.12 在屏幕底端显示对话框



▲ 图 7.13 在任意位置显示对话框

### 7.2.3 在对话框按钮和内容文本中插入图像

工程目录: src\ch07\ch07\_text\_image\_dialog

在 5.2.2 节曾介绍了如何在 TextView 控件中插入图像，这种技术在对话框中也同样适用。在对话框的内容和按钮文本中各插入了一个图像的代码如下所示。

```
AlertDialog alertDialog = new AlertDialog.Builder(this)
    .setMessage(
        Html.fromHtml("哈哈, <img src=''/>你好."), new ImageGetter()
    {
        @Override
        public Drawable getDrawable(String source)
        {
            Drawable drawable = getResources().getDrawable(R.drawable.face);
            drawable.setBounds(0, 0, 32, 32);
            return drawable;
        }
    }, null))
    .setPositiveButton(
        Html.fromHtml("<img src=''/>确定"), new ImageGetter()
    {
        @Override
        public Drawable getDrawable(String source)
        {
            Drawable drawable = getResources().getDrawable(R.drawable.ok);
            drawable.setBounds(0, 0, 20, 20);
            return drawable;
        }
    }, null), null)
.setNegativeButton(
```

```

        Html.fromHtml("<img src=' '/>取消", new ImageGetter()
    {
        @Override
        public Drawable getDrawable(String source)
        {
            Drawable drawable = getResources().getDrawable(R.drawable.cancel);
            drawable.setBounds(0, 0, 20, 20);
            return drawable;
        }
    }, null), null).create();
alertDialog.show();

```

显示的效果如图 7.14 所示。



▲ 图 7.14 在对话框文本中插入图像

### ■ 注意

5.2.1 节介绍的 HTML 标签在对话框中并不都有效，例如，通过`<font>`标签无法设置文字颜色，但经笔者测试，在 OPhone 中却可以通过`<font>`标签设置文字颜色。因此，读者在使用这项技术时应在不同的平台上进行测试才能保证兼容性。

## 7.2.4 改变对话框的透明度

工程目录: src\ch07\ch07\_transparency\_dialog

通过`WindowManager.LayoutParams.alpha`可以设置对话框的透明度。Alpha 的取值范围在 0.0f 至 1.0f 之间 (f 表示 float 类型的数字)。0.0f 表示完全透明 (这时对话框就看不见了)，1.0f 表示完全不透明。1.0f 也是 alpha 的默认值。下面的代码显示了两个对话框，上面的对话框的 alpha 值为 0.7f，下面的对话框的 alpha 值是 1.0f，读者可以将两个对话框做一个对比。

```

// 显示透明的对话框
AlertDialog alertDialog = new AlertDialog.Builder(this)
    .setMessage("透明对话框").setPositiveButton("确定", null).create();
Window window = alertDialog.getWindow();
WindowManager.LayoutParams lp = window.getAttributes();
// 设置透明度为 0.7f
lp.alpha = 0.7f;
window.setAttributes(lp);
alertDialog.show();
// 显示非透明的对话框
alertDialog = new AlertDialog.Builder(this).setMessage("在底端显示对话框")
    .setPositiveButton("确定", null).create();
window = alertDialog.getWindow();
window.setGravity(Gravity.BOTTOM);
alertDialog.show();

```

两个对话框的显示效果如图 7.15 所示。



▲ 图 7.15 半透明和不透明对话框

## 7.3 Toast

Toast 与对话框类似，也会在屏幕的某个位置弹出一个窗口，在窗口中可以显示文本、图像等信息。但与对话框不同的是，Toast 信息提示框不可获得焦点，而且在显示一定时间后会自动关闭。因此，在显示 Toast 信息提示框的同时，屏幕上的控件仍然可以继续操作。本节将详细介绍 Toast 的用法，并介绍一下 Toast 的高级用法。

### 7.3.1 Toast 的基本用法

工程目录: src\ch07\ch07\_toast

显示 Toast 提示信息框需要使用 android.widget.Toast 类。如果只想在 Toast 上显示文本信息，可以使用如下代码：

```
Toast textToast = Toast.makeText(this, "今天的天气真好! \n 哈, 哈, 哈!", Toast.LENGTH_LONG);
textToast.show();
```

在上面的代码中使用 Toast 类的静态方法创建了一个 Toast 对象。该方法的第 2 个参数表示要显示的文本信息，第 3 个参数表示 Toast 提示信息显示的时间。由于 Toast 信息提示框没有按钮，也无法通过手机按键关闭 Toast 信息提示框，因此，只能通过显示时间的长短控制 Toast 信息提示框的关闭。如果将第 3 个参数的值设为 Toast.LENGTH\_LONG，Toast 信息提示框会在显示较长的时间后再关闭。该参数值还可以设为 Toast.LENGTH\_SHORT，表示 Toast 信息提示框会在较短的时间内关闭。

创建 Toast 对象时要注意，在创建只显示文本的 Toast 对象时建议使用 `makeText` 方法，而不要直接使用 `new` 关键字创建 Toast 对象。虽然 Toast 类有 `setText` 方法，但不能在使用 `new` 关键字创建 Toast 对象后再使用 `setText` 方法设置 Toast 信息提示框的文本信息。也就是说，下面的代码会抛出异常。

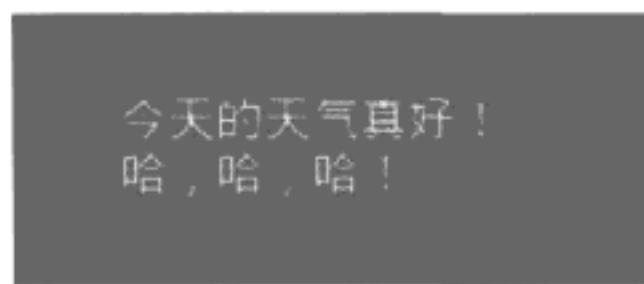
```
Toast toast = new Toast(this);
toast.setText("今天的天气真好! \n哈，哈，哈！");
toast.show(); // 执行这行代码会抛出异常
```

如果想在 Toast 信息提示框上显示其他内容，可以使用 Toast 类的 `setView` 方法设置一个 View 对象，代码如下：

```
View view = getLayoutInflater().inflate(R.layout.toast, null);
TextView textView = (TextView) view.findViewById(R.id.textview);
textView.setText("今天的天气真好! \n哈，哈，哈!");
Toast toast = new Toast(this);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(view);
toast.show();
```

也许看到这里读者会有这样的疑问：为什么使用 `new` 创建 Toast 对象后，能使用 `setView` 方法将一个 View 对象放在 Toast 信息提示框上，而不能使用 `setText` 方法来设置文本信息呢？其实原因很简单，大家看一下 `makeText` 方法的源代码就会猜得差不多。`makeText` 方法的代码也和上面的代码类似，同样是使用 `setView` 方法设置了一个 View 对象。因此，Toast 方法实际上就是通过一个 View 对象来显示信息的。如果在创建 Toast 对象时未使用 `makeText` 方法，而使用了 `new`，那么在调用 `setText` 方法时 View 对象还没有创建（Toast 类的 `setText` 方法并不会创建 View 对象），系统自然就会抛出异常了。

运行本节的例子后，单击界面的两个按钮，会分别显示纯文本的 Toast 信息提示框和带图像的 Toast 信息提示框，如图 7.16 和图 7.17 所示。



▲ 图 7.16 文本 Toast 信息提示框



▲ 图 7.17 图文混排的 Toast 信息提示框



**注意** 如果同时显示多个 Toast 信息提示框，系统会将这些 Toast 信息提示框放到队列中，等前一个 Toast 信息提示框关闭后才会显示下一个 Toast 信息提示框。也就是说，Toast 信息提示框是按顺序显示的。

### 7.3.2 永不关闭的 Toast

工程目录: src\ch07\ch07\_control\_toast

Toast 信息提示框之所以在显示一定时间后会自动关闭，是因为在系统中有一个 Toast 队列，系统会依次从队列中取出一个 Toast，并显示它。在显示一段时间后，再关闭，然后再显示下一个 Toast 信息提示框，直到 Toast 队列中所有 Toast 都显示完为止。那么有些时候需要这个 Toast 信息提示框长时间显示，直到需要关闭它时通过代码来控制，而不是让系统自动来关闭 Toast 信息提示框。不过这个要求对于 Toast 本身来说有些过分，因为 Toast 类并没有提供这个功能。虽然如此，但方法总比问题多。通过一些特殊的处理还是可以实现这个功能的，而且并不复杂。

从 7.3.1 节的内容可以知道，Toast 信息提示框需要调用 `Toast.show` 方法来显示。下面来看一下 `show` 方法的源代码。

```
public void show() {
    if (mNextView == null) {
        throw new RuntimeException("setView must have been called");
    }
    INotificationManager service = getService();
    String pkg = mContext.getPackageName();
    TN tn = mTN;
    try {
        // 将当前 Toast 加入到 Toast 队列
        service.enqueueToast(pkg, tn, mDuration);
    } catch (RemoteException e) {
        // Empty
    }
}
```

`show` 方法的代码并不复杂，可以很容易找到如下的代码。

```
service.enqueueToast(pkg, tn, mDuration);
```

从上面的代码可以很容易推断出它的功能是将当前的 Toast 加入到系统的 Toast 队列中。看到这里，各位读者应该想到，虽然 `show` 方法的表面功能是显示 Toast 信息提示框，但其实际的功能是将 Toast 加入到队列中，再由系统根据 Toast 队列来显示 Toast 信息提示框。那么我们经过更进一步地思考，可以大胆地做出一个初步的方案。既然系统的 Toast 队列可以显示 Toast 信息提示框，那么我们为什么不可以自己来显示它呢？这样不是可以自己来控制 Toast 的信息提示框的显示和关闭了吗！当然，这就不能再调用 `show` 方法来显示 Toast 信息提示框了（因为 `show` 方法会将 Toast 加入队列，这样我们就控制不了 Toast 了）。

既然初步方案已拟定，现在就来实施它。先在 `Toast` 类找一下还有没有其他的 `show` 方法。结果发现了一个 `TN` 类，该类是 `Toast` 的一个内嵌类。在 `TN` 类中有一个 `show` 方法，`TN` 是 `ITransientNotification.Stub` 的子类。从 `ITransientNotification` 和 `TN` 类中的 `show` 方法初步推断（因为 `Transient` 的中文意思是“短暂的”）系统是从 Toast 队列中获得了 Toast 对象后，利用 `TN` 对象的 `show` 方法显示 Toast，再利用 `TN.hide` 方法来关闭 Toast。首先声明，这只是假设，我们还不知道这么做是否可行！当然，这也是科学的研究的一般方法，先推断或假设，然后再证明推断或假设。

现在关键的一步是获得 `TN` 对象。遗憾的是 `TN` 被声明成 `private` 类型，外部无法访问。不过别着急，在 `Toast` 类中有一个 `mTN` 变量，虽然不是 `public` 变量，但仍然可以通过反射技术访问该变量。`mTN` 变量会在创建 `Toast` 对象时初始化。因此，只要获得 `mTN` 变量，就获得了 `TN` 对象。下

## Android 开发权威指南

面的代码显示了一个永远不会自动关闭的 Toast 信息提示框。

```
// 先创建一个 Toast 对象
Toast toast = Toast.makeText(this, "永不消失的 Toast", Toast.LENGTH_SHORT);
// 设置 Toast 信息提示框显示的位置（在屏幕顶部水平居中显示）
toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 0);
try
{
    // 从 Toast 对象中获得 mTN 变量
    Field field = toast.getClass().getDeclaredField("mTN");
    field.setAccessible(true);
    Object obj = field.get(toast);
    // TN 对象中获得了 show 方法
    Method method = obj.getClass().getDeclaredMethod("show", null);
    // 调用 show 方法来显示 Toast 信息提示框
    method.invoke(obj, null);
}
catch (Exception e)
{
}
```

上面的代码中 try{...}catch(...){...}语句中的代码是关键。先利用事先创建好的 Toast 对象获得了 mTN 变量，然后再利用反射技术获得 TN 对象的 show 方法。

关闭 Toast 和显示 Toast 的方法类似，只是需要获得 hide 方法，代码如下：

```
try
{
    // 需要将前面代码中的 obj 变量变成类变量。这样在多个地方就都可以访问了
    Method method = obj.getClass().getDeclaredMethod("hide", null);
    method.invoke(obj, null);
}
catch (Exception e)
{
}
```

上面的代码已经很完美地实现了通过代码控制 Toast 信息提示框显示和关闭的功能。但如果想实现得更完美，可以在 Android SDK 源代码中找一个叫 ITransientNotification.aidl 的文件（该文件是 AIDL 服务定义文件，将在后面详细介绍），并在 Android 工程的 src 目录中建一个 android.app 包，将这个文件放到这个包中，然后 ADT 会自动在 gen 目录中生成了一个 android.app 包，包中有一个 ITransientNotification.java 文件。由于 Android SDK 自带的 ITransientNotification 接口属于内部资源，外部程序无法访问，因此，只能将从 Toast 对象中获得的 mTN 变量转换成刚才生成的 ITransientNotification 对象了，这样就不需要使用反射技术获得 show 和 hide 方法了。经过改良的显示和关闭 Toast 信息提示框的代码如下：

```
ITransientNotification notification = (ITransientNotification) field.get(toast);
// 显示 Toast 信息提示框
notification.show();
// 关闭 Toast 信息提示框
notification.hide();
```



### 7.3.3 用 PopupWindow 模拟 Toast 提示信息框

工程目录: src\ch07\ch07\_popupwindow\_toast

使用 PopupWindow 也可以模拟 Toast 的效果。由于 Toast 信息提示框不可获得焦点，因此，也要使用 PopupWindow.setTouchable 方法将 PopupWindow 设为不可获得焦点状态。首先来编写一个用于显示窗口内容的布局文件。

#### toast.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:background="#00FFFFFF">
    <TextView android:id="@+id/tvMsg" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:background="@drawable/toast"
        android:textColor="#000" android:textSize="18sp" android:text="永不关闭的 Toast" />
</LinearLayout>
```

下面的代码在屏幕中心显示了一个窗口，在显示窗口的同时仍然可以使用窗口后面的控件。

```
View layout = getLayoutInflater().inflate(R.layout.toast, null);
PopupWindow popupWindow = new PopupWindow(layout, 200, 100);
popupWindow.setTouchable(false);
popupWindow.showAtLocation(layout, Gravity.CENTER_HORIZONTAL, 20, 0);
```

关闭窗口的代码如下：

```
popupWindow.dismiss();
```

窗口的显示效果如图 7.18 所示。



▲ 图 7.18 模拟 Toast 的窗口

## 7.4 通知 (Notification)

Notification 与对话框、Toast 无论从外观，还是从使用方法上都有着本质的区别。Notification 是 Android 中很理想的显示提示信息的方法。Notification 可以在 Android 桌面最上方的状态栏显示提示信息，采用这种方式并不会打扰用户正常操作手机。而且 Notification 风格的提示信息不仅可以显示文字，还可显示图像，甚至可以将控件加到上面，而且只要用户不清空，这些信息可永久保留在状态栏上。除此之外，Notification 还有更多吸引人的特性等待着我们挖掘，下面就让我们开始学习 Notification 吧！

PDG

### 7.4.1 在状态栏上显示通知信息

**工程目录: src\ch07\ch07\_notification**

Notification 需要用 NotificationManager 来管理。下面来看一下创建并显示一个 Notification 的步骤。创建和显示一个 Notification 需要如下 5 个步骤。

(1) 通过 getSystemService 方法获得一个 NotificationManager 对象。

(2) 创建一个 Notification 对象。每一个 Notification 对应一个 Notification 对象。在这一步需要设置显示在屏幕上方状态栏的通知消息、通知消息前方的图像资源 ID 和发出通知的时间，一般为当前时间。

(3) 由于 Notification 可以与应用程序脱离，也就是说，即使应用程序被关闭，Notification 仍然会显示在状态栏中。当应用程序再次启动后，又可以重新控制这些 Notification，如清除或替换它们。因此，需要创建一个 PendingIntent 对象。该对象由 Android 系统负责维护，因此，在应用程序关闭后，该对象仍然不会被释放。

(4) 使用 Notification 类的 setLatestEventInfo 方法设置 Notification 的详细信息。

(5) 使用 NotificationManager 类的 notify 方法显示 Notification 消息。在这一步需要指定标识 Notification 的唯一 ID，这个 ID 必须相对于同一个 NotificationManager 对象是唯一的，否则就会覆盖相同 ID 的 Notificaiton。

下面演练一下如何在状态栏中显示一个 Notification，代码如下：

```
// 第1步
NotificationManager notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
// 第2步
Notification notification = new Notification(R.drawable.icon, "您有新消息了", System.currentTimeMillis());
// 第3步
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, getIntent(), 0);
// 第4步
notification.setLatestEventInfo(this, "天气预报", "晴转多云", contentIntent);
// 第5步
notificationManager.notify(R.drawable.icon, notification);
```

上面的 5 行代码正好对应创建和显示 Notification 的 5 个步骤。在这里要解释一下的是 notify 方法的第一个参数。这个参数实际上表示 Notification 的 ID，是一个 int 类型的值。为了使这个值唯一，可以使用 res 目录中的某些资源 ID。例如，在上面的代码中使用了当前 Notification 显示的图像对应的资源 ID (R.drawable.icon) 作为 Notification 的 ID。当然，读者也可以使用其他的值作为 Notification 的 ID。

由于创建和显示多个 Notification 的代码类似，因此，本节的例子中编写了一个 showNotification 方法来显示 Notification，代码如下：

```
private void showNotification(String tickerText, String contentTitle, String contentText,
    int id, int resId)
```

```

{
    Notification notification = new Notification(resId, tickerText, System.currentTimeMillis());
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new Intent(this, Main.class), 0);
    notification.setLatestEventInfo(this, contentTitle, contentText, contentIntent);
    // notificationManager 是在类中定义的 NotificationManager 变量，在 onCreate 方法中已经创建
    notificationManager.notify(id, notification);
}

```

上面代码中的 `PendingIntent.getActivity` 方法用于返回一个 `PendingIntent` 对象，这个对象与一个 `Activity` 关联，在本例中与当前的 `Activity` 关联。将 Android 状态栏滑下来后，单击 `Notification`，就会显示这个关联的 `Activity`。如果 `Activity` 已经显示，仍然会再显示一个新的 `Activity`，并覆盖当前显示的 `Activity`。不过这些显示的 `Activity` 都是一样的，除了 `getActivity` 方法外，还有 `getBroadcast` 和 `getService` 方法。这两个方法用于在单击 `Notification` 时发送一条广播或启动一个服务。关于广播和服务的详细内容将在后面介绍。

下面的代码使用 `showNotification` 方法显示了 3 个 `Notification`。

```

showNotification("今天非常高兴", "今天考试得了全年级第一",
    "数学 100 分、语文 99 分、英语 100 分, yeah!", R.drawable.smile, R.drawable.smile);
showNotification("这是为什么呢?", "这道题为什么会出现错误呢?", "谁有正确答案啊.",
    R.drawable.why, R.drawable.why);
showNotification("今天心情不好", "也不知道为什么, 这几天一直很郁闷.", "也许应该去公园散心了",
    R.drawable.why, R.drawable.wrath);

```

其中第 2 个和第 3 个 `Notification` 使用的是同一个 ID(`R.drawable.why`)，因此，第 3 个 `Notification` 会覆盖第 2 个 `Notification`。

在显示 `Notification` 时还可以设置显示通知时的默认发声、震动和 Light 效果。要实现这个功能需要设置 `Notification` 类的 `defaults` 属性，代码如下：

```

notification.defaults = Notification.DEFAULT_SOUND;           // 使用默认的声音
notification.defaults = Notification.DEFAULT_VIBRATE;        // 使用默认的震动
notification.defaults = Notification.DEFAULT_LIGHTS;          // 使用默认的 Light
notification.defaults = Notification.DEFAULT_ALL;             // 所有的都使用默认值

```

### 注意

设置默认发声、震动和 Light 的方法是 `setDefaults`（具体实现详见光盘中的源代码）。该方法与 `showNotification` 方法的实现代码基本相同，只是在调用 `notify` 方法之前需要设置 `defaults` 属性（`defaults` 属性必须在调用 `notify` 方法之前调用，否则不起作用）。在设置默认震动效果时还需要在 `AndroidManifest.xml` 文件中通过 `<uses-permission>` 标签设置 `android.permission.VIBRATE` 权限。

如果要清除某个消息，可以使用 `NotificationManager.cancel` 方法，该方法只有一个参数，表示要清除的 `Notification` 的 ID。使用 `cancelAll` 可以清除当前 `NotificationManager` 对象中的所有 `Notification`。

运行本节的例子，单击屏幕上显示 `Notification` 的按钮，会显示如图 7.19 所示的消息。每一个

## Android 开发权威指南

消息会显示一会，然后就只显示整个 Android 系统（也包括其他应用程序）的 Notification（只显示图像部分），如图 7.20 所示。如果将状态栏拖下来，可以看到 Notification 的详细信息和发出通知的时间（也就是 Notification 类的构造方法的第 3 个参数值），如图 7.21 所示。单击“清除通知”按钮，会清除本应用程序显示的所有 Notification，清除后的效果如图 7.22 所示。



▲ 图 7.19 显示 Notification



▲ 图 7.20 只显示 Notification 的图像



▲ 图 7.21 显示 Notification 的详细信息



▲ 图 7.22 清除 Notification 后的效果

### 7.4.2 Notification 的清除动作

工程目录: src\ch07\ch07\_remove\_notification

在如图 7.21 所示的界面可以看到，屏幕的右上角有一个“清除”按钮，单击该按钮可以清除所有的 Notification。那么在清除后，往往需要做一些善后的工作，这就需要通过 Notification.deleteIntent 来完成。deleteIntent 也需要设置一个 PendingIntent 类型的变量，用于在清除所有的 Notification 时调用。可以将这个动作与 Activity、Broadcast 和 Service 关联。例如，下面的代码显示了一个 Notification，并将当前的 Activity 与清除动作相关联。

```
Notification notification = new Notification(R.drawable.smile,
    "收到短信了。", System.currentTimeMillis());
Intent intent = new Intent(this, Main.class);
// 传递数据
intent.putExtra("msg", "最近在忙什么？");
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
// 将当前 Activity 与清除动作关联
notification.deleteIntent = pendingIntent;
notification.setLatestEventInfo(this, "短信内容", "最近在忙什么？", pendingIntent);
notificationManager.notify(R.drawable.smile, notification);
```

如果想向响应删除动作的 Activity 传递数据，可以利用被 PendingIntent 封装的 Intent。例如，上面的代码将 Notification 消息的内容保存在了 Intent 对象中，这样在 Activity 中（一般在 onCreate 方法中）可以使用如下的代码获得传递过来的数据，并根据这些数据决定删除哪些数据。

```
String msg = getIntent().getStringExtra("msg");
if (msg != null)
    Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
```

运行本例后，单击“显示 Notification”按钮，会显示一个 Notification，然后无论是否关闭当前的 Activity，单击如图 7.21 所示的“清除”按钮后，都会显示关联的 Activity，并显示一个 Toast 信息框。

### 7.4.3 永久存在的 Notification

**工程目录：**src\ch07\ch07\_permanent\_notification

我们发现很多 Android 应用程序在显示 Notification 后，单击“清除”按钮并没有清除这些 Notification。这些无法被清除的 Notification 称为永久 Notification，这些 Notification 只能通过显示它们的程序清除。实现这种效果也很简单，只需要设置 Notification.flag 即可。例如，下面的代码显示了一个永久存在的 Notification。

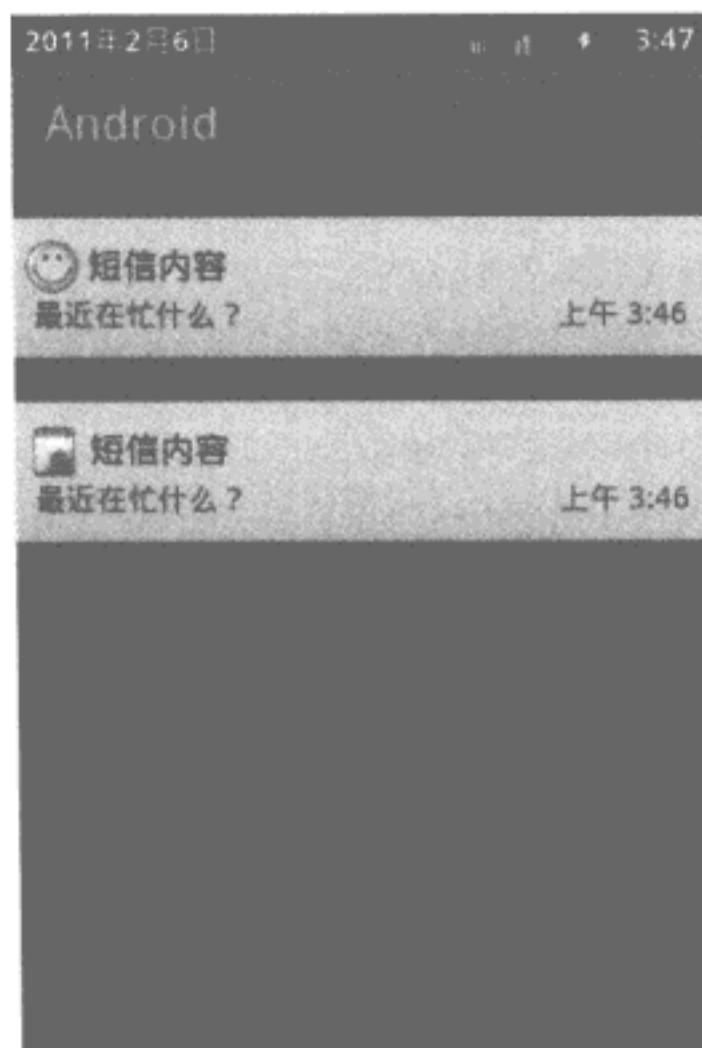
```
Notification notification = new Notification(R.drawable.smile, "收到短信了。", System.currentTimeMillis());
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, new Intent(this, Main.class), 0);
// 设置成永久存在的 Notification
notification.flags = Notification.FLAG_NO_CLEAR;
notification.setLatestEventInfo(this, "短信内容", "最近在忙什么？", pendingIntent);
notificationManager.notify(R.drawable.smile, notification);
```

使用上面的代码虽然可以使 Notification 无法通过点击“清除”按钮删除，但仍然显示在“通

知”一栏中。但有些应用程序，如“360 安全卫士”显示在了“正在运行的”一栏中。如果要将 Notification 加到这一栏，可以使用如下的代码。

```
Notification.flags = Notification.FLAG_ONGOING_EVENT;
```

在这两栏中显示的 Notification 效果如图 7.23 所示。



▲ 图 7.23 永久存在的 Notification

#### 7.4.4 自定义 Notification

工程目录: src\ch07\ch07\_custom\_notification

我们可以通过 Notification.contentView 来自定义 Notification。contentView 并不是一个 View，而是 RemoteViews 类型，RemoteView 在后面介绍 AppWidget 时还会遇到。RemoteView 只支持有限的几个控件和布局，这些控件和布局如下。

支持的布局:

- FrameLayout;
- LinearLayout;
- RelativeLayout.

支持的控件:

- AnalogClock;
- Button;
- Chronometer;
- ImageButton;
- ImageView;
- ProgressBar;

- **TextView。**

如果使用其他的布局和控件，系统会抛出异常。下面先来编写一个用于自定义 Notification 的布局文件。

### **notification.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/textview" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="自定义内容"
        android:textColor="#F00" android:textSize="20sp" android:gravity="center" />
    <ImageView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:src="@drawable/smile"
        android:layout_gravity="center" />
</LinearLayout>
```

下面的代码利用 notification.xml 自定义了一个 Notification。

```
Notification notification = new Notification(R.drawable.smile,
    "自定义 Notification.", System.currentTimeMillis());
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
    new Intent(this, Main.class), 0);
RemoteViews remoteViews = new RemoteViews(getPackageName(), R.layout.notification);
remoteViews.setTextViewText(R.id.textview, "更新自定义内容");
notification.contentView = remoteViews;
notification.contentIntent = pendingIntent;
notificationManager.notify(R.drawable.smile, notification);
```

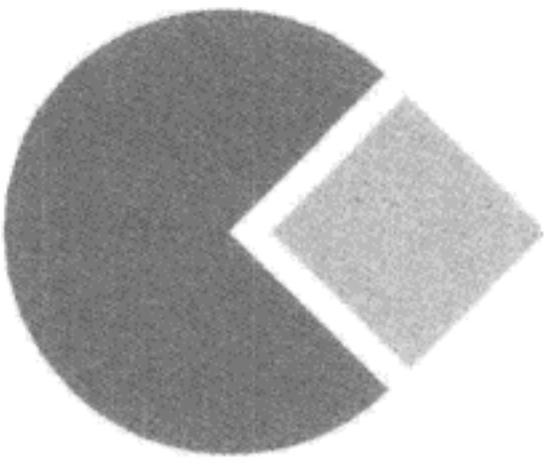
自定义 Notification 的效果如图 7.24 所示。



▲ 图 7.24 自定义 Notification

## 7.5 小结

本章介绍了 Android 中常用的显示提醒信息的方法，其中对话框是第一种显示提醒信息的方式。但对话框是独占显示的，也就是说，只有在关闭对话框后才能进行其他操作。而 Toast 和 Notification 的显示并不影响用户的当前操作，因此，除非必要，建议使用 Toast 或 Notification 来显示提醒信息。Notification 除了可以显示简单的信息外，还可以通过 RemoteViews 自定义 Notification，虽然只支持有限的几种布局和控件，但只要支持 ImageView 控件，我们就可以绘制出丰富多彩的内容。



## 第8章 移动的信息仓库——数据存储

在 Android 系统中提供了多种存储技术，这些存储技术可以将数据保存在各种存储介质上。例如，SharedPreferences 可以将数据保存在应用软件的私有存储区，这些存储区中的数据只能被写入这些数据的软件读取。除此之外，Android 系统还支持文件存储、SQLite 数据库、OBB 文件、云存储等。

### 8.1 读写 key-value 对：SharedPreferences

如果要问 Android SDK 中哪一种存储技术最容易理解和使用，答案毫无悬念，一定是 SharedPreferences。实际上，SharedPreferences 处理的就是一个 key-Value 对。例如，要保存产品的名称，可以将 key 设为 produceName，value 为实际的产品名。

#### 8.1.1 SharedPreferences 的基本用法

工程目录：src\ch08\ch08\_sharedpreferences

保存 key-value 对首先要指定一个文件名，然后使用类似 putString 的方法指定 key 和 value。SharedPreferences 也采用了同样的方法。使用 SharedPreferences 保存 key-value 对的步骤如下。

(1) 使用 Activity.getSharedPreferences 方法获得 SharedPreferences 对象。其中存储 key-value 的文件名称由 getSharedPreferences 方法的第一个参数指定。

(2) 使用 SharedPreferences.edit 方法获得 SharedPreferences.Editor 对象。

(3) 通过 SharedPreferences.Editor.putXxx 方法保存 key-value 对。其中 Xxx 表示 value 的不同数据类型。例如，Boolean 类型的 value 需要用 putBoolean 方法，字符串类型的 value 需要用 putString 方法。

(4) 通过 SharedPreferences.Editor.commit 方法保存 key-value 对。commit 方法相当于数据库事务中的提交（commit）操作，只有在事务结束后进行提交，才会将数据真正保存在数据库中。保存 key-value 也是一样，在使用 putXxx 方法指定了 key-value 对后，必须调用 commit 方法才能将 key-value 对真正保存在相应的文件中。

下面的代码使用 SharedPreferences 保存了两个 key-value 对。

```
// 获得 SharedPreferences 对象（第1步）
SharedPreferences mySharedPreferences = getSharedPreferences("test",Activity.MODE_PRIVATE);
```

```
// 获得 SharedPreferences.Editor 对象（第 2 步）
SharedPreferences.Editor editor = mySharedPreferences.edit();
// 使用 putXxx 方法保存数据（第 3 步）
editor.putString("name", "李宁");
editor.putString("habit", "Android、写作、旅游");
// 将数据保存在文件中（第 4 步）
editor.commit();
```

在执行上面的代码后，SharedPreferences 会将 key-value 对保存在 test.xml 文件（指定文件名时不需要加扩展名 xml）中。保存的具体位置和其他细节将在下一节详细介绍。

下面的代码使用 SharedPreferences 从 test.xml 文件中读取 name 和 habit 的值。

```
SharedPreferences sharedPreferences = getSharedPreferences("test", Activity.MODE_PRIVATE);
// 使用 getXxx 方法获得 value, getXxx 方法的第 2 个参数是 value 的默认值
String name = sharedPreferences.getString("name", "");
String habit = sharedPreferences.getString("habit", "");
Toast.makeText(this, "name: " + name + "\n" + "habit: " + habit, Toast.LENGTH_LONG).show();
```

运行上面的代码后，如果已经保存了 name 和 habit 的值，会显示如图 8.1 所示的 Toast 信息提示框。

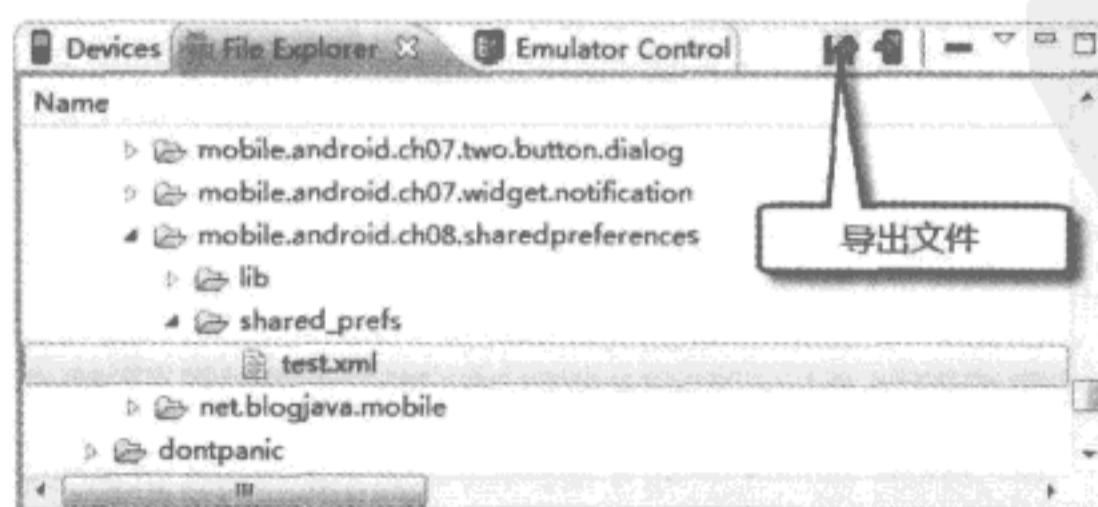
### 8.1.2 数据的存储位置和格式

在上一节介绍了用 SharedPreferences 存储和读取数据的方法，但这些数据被保存在哪里呢？仅从代码上是无法获得更多细节的。

实际上，SharedPreferences 将数据文件写在手机内存（ROM）私有的目录中。在模拟器中测试程序，可以通过 DDMS 透视图来查看数据文件的位置。打开 DDMS 透视图，进入“File Explorer”视图，找到 data/data 目录。在该目录下有若干个子目录，这些子目录名就是模拟器中安装的程序使用的包名（package name）。找到本例使用的包名（mobile.android.ch08.sharedpreferences），在该目录下有一个 shared\_prefs 子目录，在上一节建立的数据文件（test.xml）就保存在这个目录中，如图 8.2 所示。从这一点可以看出，用 SharedPreferences 生成的数据文件保存在 /data/data/<package name>/shared\_prefs 目录中。

name : 李宁  
habit : Android、写作、旅游

▲ 图 8.1 显示 name 和 habit 的值



▲ 图 8.2 SharedPreferences 生成的数据文件的存储位置

使用如图 8.2 所示的文件导出按钮将 test.xml 文件导出到本地后，查看该文件的内容可知 SharedPreferences 使用 XML 格式来保存数据。

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="name">李宁</string>
    <string name="habit">Android、写作、旅游</string>
</map>
```

### 8.1.3 存取复杂类型的数据

**工程目录: src\ch08\ch08\_sharedpreferences\_complex\_data**

前面介绍的 SharedPreferences 只能保存简单类型的数据，例如，String、int 等。如果想用 SharedPreferences 存取更复杂的数据类型（对象、图像等），就需要对这些数据进行编码。通常会将复杂类型的数据转换成 Base64 格式的编码，然后将转换后的数据以字符串的形式保存在 XML 文件中。

在本例中将一个 Product 对象和一个图像保存在 XML 文件中，并在程序重新运行后从 XML 文件装载 Product 对象和图像。Product 类的代码如下：

```
package mobile.android.ch08.sharedpreferences.complex.data;
import java.io.Serializable;
public class Product implements Serializable
{
    public String name;
    public int price;
}
```

现在找一个图像文件，并放在 res\drawable 目录中，本例的图像文件是 flower.png。下面的代码将该图像保存在 base64.xml 文件中。

```
try
{
    SharedPreferences sharedPreferences = getSharedPreferences("base64", Activity.MODE_PRIVATE);
    Editor editor = sharedPreferences.edit();
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    // 读取和压缩 flower.png，并将其压缩结果保存在 ByteArrayOutputStream 对象中
    BitmapFactory.decodeResource(getResources(), R.drawable.flower).compress(CompressFormat.JPEG,
        50, baos);
    // 对压缩后的字节进行 Base64 编码
    String imageBase64 = new String(Base64.encode(baos.toByteArray(), Base64.DEFAULT));
    // 保存转换后的 Base64 格式字符串
    editor.putString("image", imageBase64);
    editor.commit();
    baos.close();
}
catch (Exception e)
{
```

下面的代码从 base64.xml 文件中读取已保存的图像，并将其显示在 ImageView 控件中。

```
try
```

```

{
    SharedPreferences sharedPreferences = getSharedPreferences("base64", Activity.MODE_PRIVATE);
    // 读取 Base64 格式的图像数据
    String imageBase64 = sharedPreferences.getString("image", "");
    // 对 Base64 格式的字符串进行解码，还原成字节数组
    byte[] imageBytes = Base64.decode(imageBase64.getBytes(), Base64.DEFAULT);
    ByteArrayInputStream bais = new ByteArrayInputStream(imageBytes);
    ImageView imageView = (ImageView) findViewById(R.id.image);
    // 在 ImageView 控件上显示图像
    imageView.setImageDrawable(Drawable.createFromStream(bais, "image"));
    bais.close();
}
catch (Exception e)
{
}

```

存取对象的代码与存取图像的代码类似，首先这个对象必须可序列化（对应的类要实现 `java.io.Serializable` 接口），然后需要将这个可序列化的对象转换成字节数组，并进一步编码成 Base64 格式的字符串。下面的代码将 `Product` 对象保存在 `base64.xml` 文件中。

```

try
{
    Product product = new Product();
    product.name = "Android 手机";
    product.price = 2800;
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    // 将 Product 对象保存在 ObjectOutputStream 对象中
    oos.writeObject(product);
    SharedPreferences sharedPreferences = getSharedPreferences("base64", Activity.MODE_PRIVATE);
    Editor editor = sharedPreferences.edit();
    sharedPreferences = getSharedPreferences("base64", Activity.MODE_PRIVATE);
    // 将 Product 对象转换成 byte 数组，并将其进行 base64 编码
    String productBase64 = new String(Base64.encode(baos.toByteArray(), Base64.DEFAULT));
    // 将编码后的字符串保存到 base64.xml 文件中
    editor.putString("product", productBase64);
    editor.commit();
    oos.close();
}
catch (Exception e)
{
}

```

下面的代码从 `base64.xml` 文件中读取了 `Product` 对象，并显示对象中的属性值。

```

try
{
    SharedPreferences sharedPreferences = getSharedPreferences("base64", Activity.MODE_PRIVATE);
    // 从 base64.xml 文件中读取 Product 对象的 base64 格式字符串
    String base64Product = sharedPreferences.getString("product", "");

```

```

ByteArrayOutputStream baos = new ByteArrayOutputStream();
// 将 base64 格式字符串还原成 byte 数组
byte[] productBytes = Base64.decode(base64Product.getBytes(), Base64.DEFAULT);
ByteArrayInputStream bais = new ByteArrayInputStream(productBytes);
ObjectInputStream ois = new ObjectInputStream(bais);
// 将 byte 数组转换成 Product 对象
Product product = (Product) ois.readObject();
Toast.makeText(this,"name:" + product.name + "\nprice: " + product.price,
    Toast.LENGTH_LONG).show();
ois.close();
}
catch (Exception e)
{
}

```

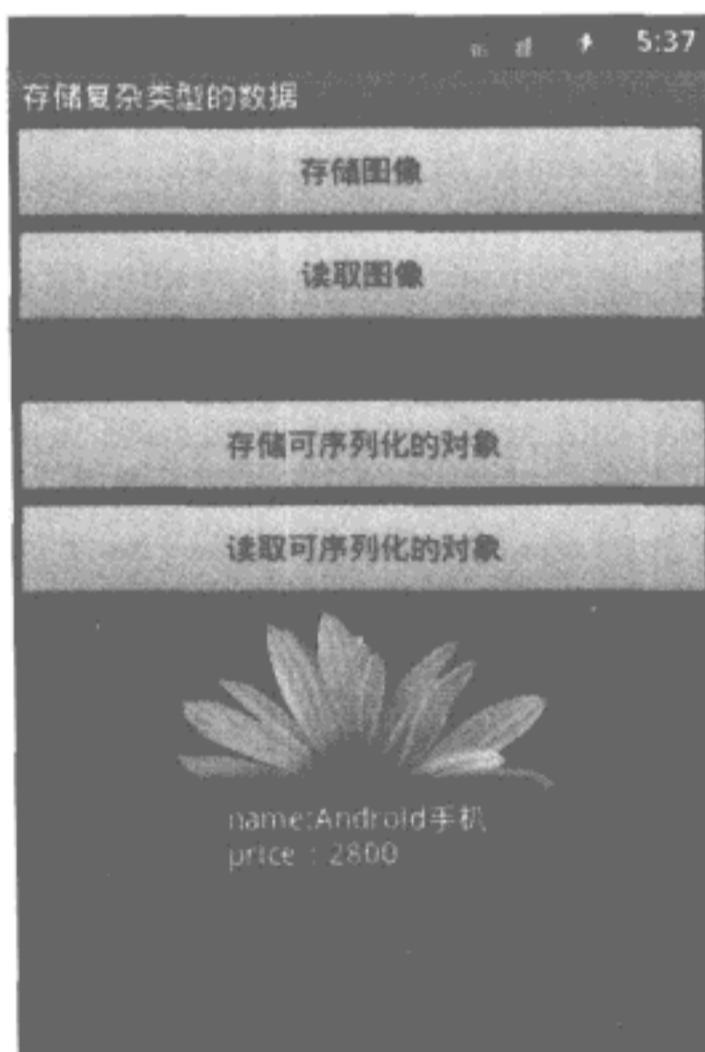
下面我们来看看保存在 base64.xml 文件中内容（由于 base64.xml 文件中的内容较多，在这里只给出其中的一部分）。

```

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="product">r00ABXNyADptb2JpbGUuYW5kcm9pZC5jaDA4... ...</string>
    <string name="image">/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDABALDA4M... ...</string>
</map>

```

运行本例后，读取并显示图像、Product 对象的效果如图 8.3 所示。



▲ 图 8.3 读写并显示图像、Product 对象

### ■ 注意

虽然可以采用编码的方式通过 SharedPreferences 保存任何类型的数据，但笔者并不建议使用 SharedPreferences 保存大量的数据。如果读者要存取更多的数据，可以使用后面要介绍的文件存储、SQLite 数据库等技术。

### 8.1.4 设置数据文件的访问权限

**工程目录:** src\ch08\ch08\_permission

众所周知, Android 系统并不是全新的操作系统, 而是在 Linux 内核基础上发展而来的一个移动操作系统(虽然 Android 也可运行在 PC 上, 但 Android 最初是为以手机为主的移动设备设计的, 因此, 我们习惯称它为移动操作系统)。既然本质上是 Linux, 那么自然就会拥有 Linux 的一些基本特征。

学习 Linux 必须要掌握的就是 Linux 的文件权限。Linux 与 Windows 不同, Windows 文件的很多特性是通过文件扩展名来识别的。例如, exe 是可执行文件, bat 是批处理文件。而在 Linux 中, 文件扩展名并不重要。一个文件是否可访问、可执行, 完全是由文件属性来决定的。

Linux 文件的属性可分为 4 段。第 1 段的取值如下:

[d]: 表示目录。

[ -]: 表示文件。

[l]: 表示链接文件。

[b]: 表示可供存储的接口设备文件。

[c]: 表示串口设备文件, 例如, 键盘、鼠标。

从第 2 段到第 4 段都由 3 个字母组成, 分别表示不同用户的读、写和执行权限, 含义如下:

[r]: 表示可读。

[w]: 表示可写。

[x]: 表示可执行。

如果不具备某个属性, 该项将以[-]代替, 例如, rw-、--x 等。

第 2 段表示文件所有者(创建文件的用户)拥有的权限, 第 3 段表示文件所有者所在的用户组中其他用户的权限, 第 4 段表示其他用户(非所有者所在的用户组中的用户)的权限。例如,-rw-rw---- 表示文件所有者及文件所有者所在的用户组中的用户可以对该文件进行读和写操作, 其他的用户无权访问该文件。

现在回到 Android 系统中。在前面曾多次使用 getSharedPreferences 方法获得 SharedPreferences 对象, getSharedPreferences 方法的第 2 个参数值使用了 Activity.MODE\_PRIVATE 常量。除了这个常量外, 还可以使用另外 3 个常量, 这 4 个常量用于指定文件的建立模式, 它们有一个重要的功能就是设置文件的属性。下面的代码分别使用这 4 个建立模式创建了 4 个文件, 读者可以观察文件的属性。

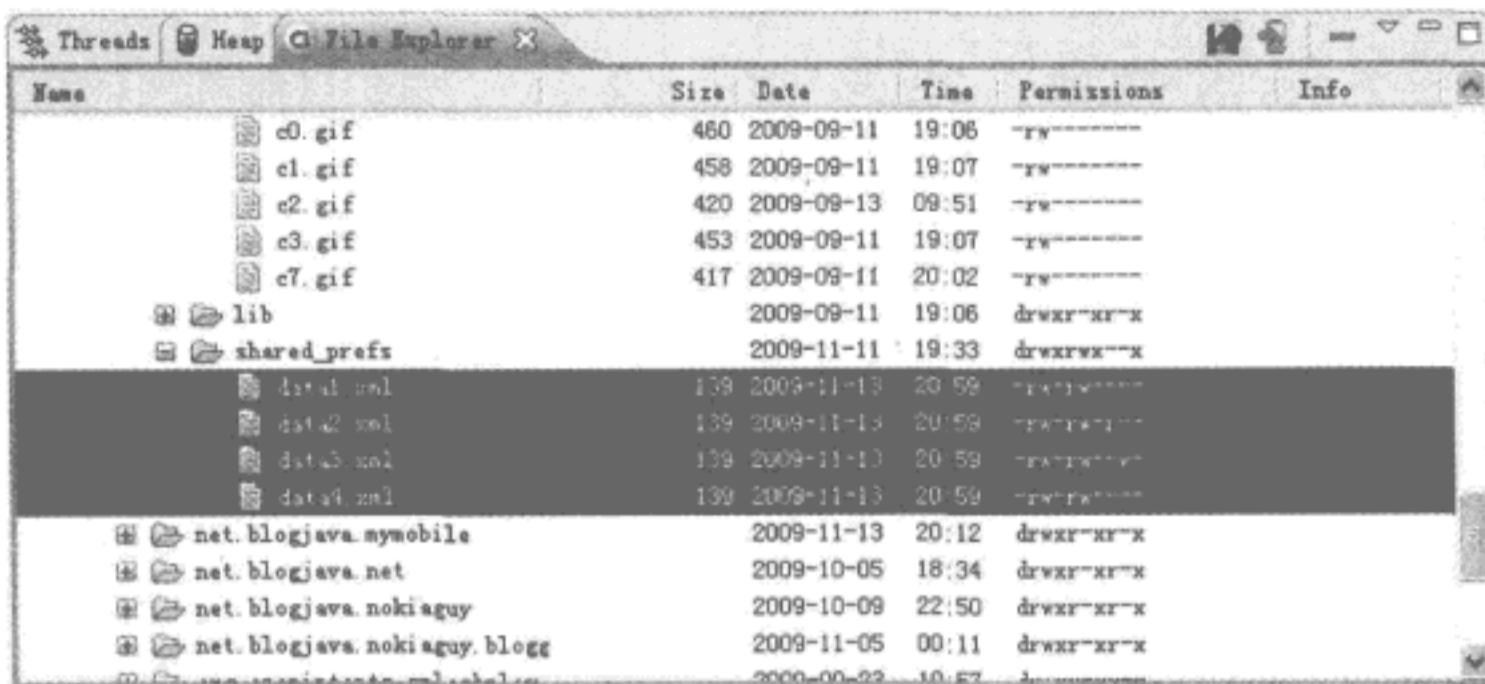
```
int[] modes = new int[]
{ Activity.MODE_PRIVATE, Activity.MODE_WORLD_READABLE,
  Activity.MODE_WORLD_WRITEABLE, Activity.MODE_APPEND };
for(int i = 0; i < modes.length; i++)
{
    SharedPreferences mySharedPreferences = getSharedPreferences(
        "data" + String.valueOf(i + 1), modes[i]);
    SharedPreferences.Editor editor = mySharedPreferences.edit();
    editor.putString("name", "bill");
```

```

    editor.commit();
}

```

运行上面的代码后，将在 shared\_prefs 目录中创建 4 个文件（data1.xml、data2.xml、data3.xml、data4.xml），这 4 个文件的属性如图 8.4 所示。



▲ 图 8.4 查看文件的属性

从图 8.4 可以看出，MODE\_WORLD\_READABLE 和 MODE\_WORLD\_WRITEABLE 分别设置其他用户的读和写权限，而使用 MODE\_PRIVATE 和 MODE\_APPEND 创建的文件对于其他用户都是不可访问的。

### 8.1.5 可以保存设置的 Activity：PreferenceActivity

工程目录：src\ch08\ch08\_preferences

由于 SharedPreferences 使用非常方便，所以经常用 SharedPreferences 保存配置信息。不过 Android SDK 提供了更容易的方法来实现配置界面，并且可以透明地保存配置信息。这就是 PreferenceActivity。

PreferenceActivity 是 Activity 的子类，该类封装了 SharedPreferences。因此，PreferenceActivity 的所有子类都会拥有保存 key-value 对的能力。

PreferenceActivity 提供了一些常用的控件，这些控件可以满足大多数配置界面的要求。与 Android SDK 的标准控件一样，这些 PreferenceActivity 控件既可以从 XML 文件创建，也可以从代码创建。比较常用的控件有如下 3 个。

- CheckBoxPreference：对应<CheckBoxPreference>标签。该控件相当于 CheckBox。
- EditTextPreference：对应<EditTextPreference>标签。单击该控件会弹出一个带 EditText 的对话框。
- ListPreference：对应<ListPreference>标签。单击该控件会弹出一个带 ListView 的对话框。

本节的例子将使用 XML 文件的方式创建设置界面。在 res 目录下建立一个 xml 子目录，并在该目录中建立一个 preference\_setting.xml 文件，并输入如下内容。

```

<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

```

```

<PreferenceCategory android:title="我的位置源">
    <CheckBoxPreference android:key="wireless_network"
        android:title="使用无线网络"
        android:summary="使用无线网络查看应用程序（例如 Google 地图）中的位置" />
    <CheckBoxPreference android:key="gps_satellite_setting"
        android:title="启用 GPS 卫星设置"
        android:summary="定位时，精确到街道级别（取消选择可节约电量）" />
</PreferenceCategory>
<PreferenceCategory android:title="个人信息设置">
    <CheckBoxPreference android:key="yesno_save_individual_info"
        android:title="是否保存个人信息" />
    <EditTextPreference android:key="individual_name"
        android:title="姓名" android:summary="请输入真实姓名" />
    <!-- 有一个子设置页 -->
    <PreferenceScreen android:key="other_individual_msg"
        android:title="其他个人信息" android:summary="是否工作、手机">
        <CheckBoxPreference android:key="is_an_employee"
            android:title="是否工作" />
        <EditTextPreference android:key="mobile"
            android:title="手机" android:summary="请输入真实的手机号" />
    </PreferenceScreen>
</PreferenceCategory>
</PreferenceScreen>

```

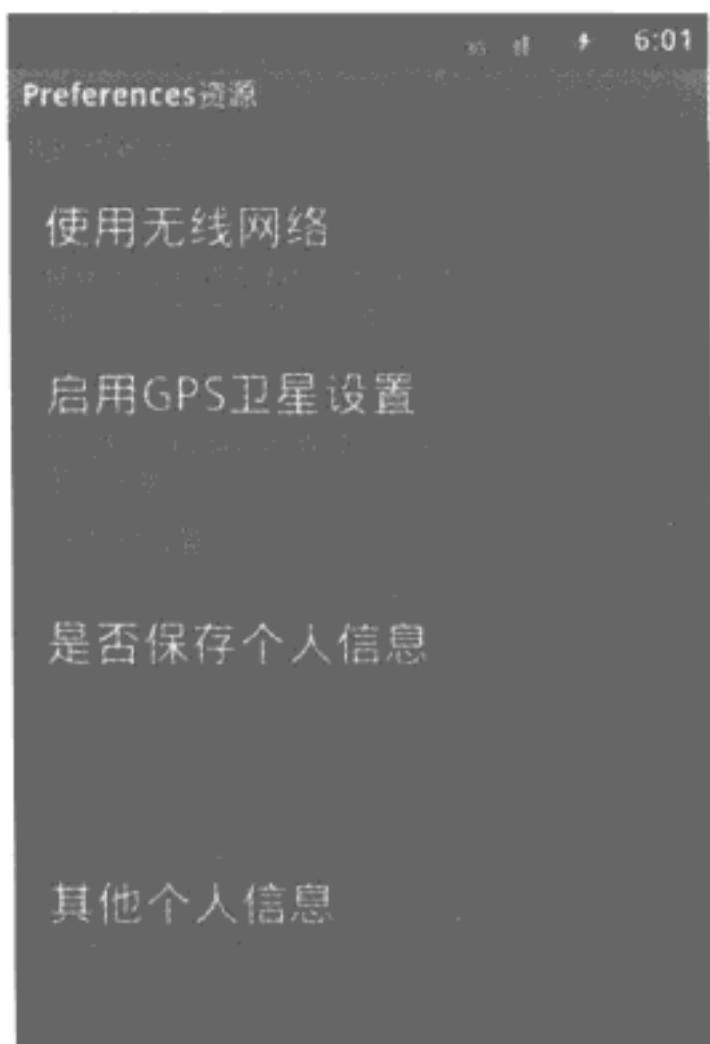
在编写上面代码时要注意如下几点。

- 一个设置界面对应一个<PreferenceScreen>标签。
- <PreferenceCategory>标签表示一个设置分类，title 属性表示分类名称，该名称会显示在设置界面上。
- 控件标签（<CheckBoxPreference>、<EditTextPreference>等）可以放在<PreferenceCategory>标签中，也可以不使用<PreferenceCategory>标签，而直接放在<PreferenceScreen>标签中，表示该控件不属于任何设置分类。
- 每一个控件标签（<CheckBoxPreference>、<EditTextPreference>等）都有一个 android:key 属性，该属性的值就是保存在 XML 文件中的 key-value 对中的 key。
- 如果使用嵌套<PreferenceScreen>标签，说明该设置页有一个子设置页，单击该设置页就会进入这个子设置页。
- android:title 和 android:summary 分别表示列表项的标题和摘要，标题用大字体显示在摘要上方，摘要用小字体显示。

在 PreferenceActivity.onCreate 方法中并不需要设置布局文件，只需要使用如下代码装载 preference\_setting.xml 文件即可：

```
| addPreferencesFromResource(R.xml.preference_setting);
```

现在运行程序，会显示如图 8.5 所示的设置页面。当单击“姓名”列表项时，会弹出一个带 EditText 控件的对话框，如图 8.6 所示。

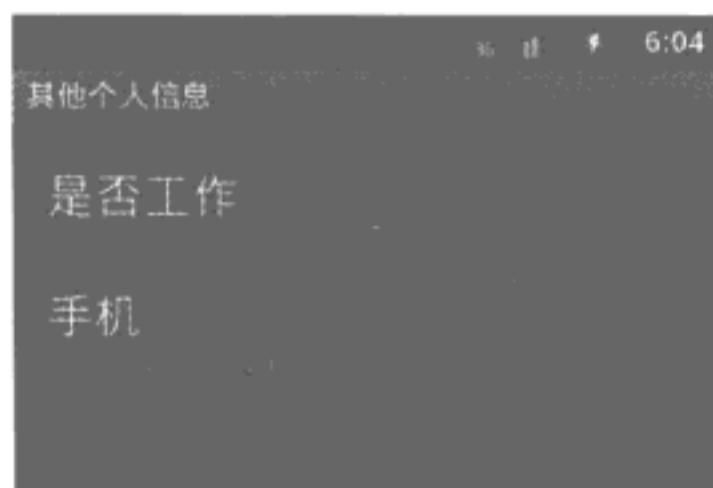


▲ 图 8.5 设置界面



▲ 图 8.6 弹出带 EditText 组件的对话框

单击最后一个列表项，会进入如图 8.7 所示的子设置界面。



▲ 图 8.7 子设置界面

单击某一个列表项时会调用 `onPreferenceTreeClick` 事件方法。在本例中，如果取消“是否保存个人信息”列表项的复选框的选中状态，“姓名”列表项会变为不可选状态。实现这个功能正好用到 `onPreferenceTreeClick` 事件方法，代码如下：

```
public boolean onPreferenceTreeClick(PreferenceScreen preferenceScreen,
    Preference preference)
{
    // 判断选中的是否为“是否保存个人信息”列表项的复选框
    if ("yesno_save_individual_info".equals(preference.getKey()))
    {
        // 设置“姓名”列表项为可选或不可选
        findPreference("individual_name").setEnabled(!findPreference("individual_name").isEnabled());
    }
    return super.onPreferenceTreeClick(preferenceScreen, preference);
}
```

在单击“姓名”列表项弹出的对话框的 EditText 控件中输入姓名后，单击“确定”按钮，会用输入的值作为“姓名”列表项的 Summary。为了捕获列表项的值改变的事件，需要使用 onPreferenceChange 事件方法，代码如下：

```
public boolean onPreferenceChange(Preference preference, Object newValue)
{
    // 设置“姓名”列表项中 Summary 的值
    preference.setSummary(String.valueOf(newValue));
    // 该方法必须返回 true，否则无法保存设置的值
    return true;
}
```

最后在 onCreate 方法中还需要做如下几项工作。

- 我们可以修改 PreferenceActivity 保存数据使用的 XML 文件的名称。在默认情况下，保存 key-value 对的 XML 文件是<package name>\_preferences.xml。在本例中就是 mobile.android.ch08.preferences\_preferences.xml。
- 设置“姓名”列表项的 Summary。该值需要从保存 key-value 对的 XML 文件中读取。
- 设置“姓名”列表项是否可用。该值根据“是否保存个人信息”列表项的复选框是否被选中来设置。
- 每一个列表项是一个 Preference 对象。由于“姓名”列表项使用了 onPreferenceChange 方法，因此，需要使用 PreferenceActivity.setOnPreferenceChangeListener 方法设置包含该事件方法的对象。在本例中是 this，因此，Main 类需要实现 OnPreferenceChangeListener 接口。

onCreate 方法的完整代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // 改变 PreferenceActivity 保存数据使用的 XML 文件的名称
    getPreferenceManager().setSharedPreferencesName("setting");
    addPreferencesFromResource(R.xml.preference_setting);
    // 获得“姓名”列表项对应的 Preference 对象
    Preference individualNamePreference = findPreference("individual_name");
    // 获得指向 setting.xml 文件的 SharedPreferences 对象
    SharedPreferences sharedpreferences= individualNamePreference.getSharedPreferences();
    // 设置“姓名”列表项的 Summary
    individualNamePreference.setSummary(sharedpreferences.getString("individual_name",
    ""));
    // 设置“姓名”列表项是否可用
    if (sharedpreferences.getBoolean("yesno_save_individual_info", false))
        individualNamePreference.setEnabled(true);
    else
        individualNamePreference.setEnabled(false);
    // 设置包含 onPreferenceChange 事件方法的对象实例
    individualNamePreference.setOnPreferenceChangeListener(this);
}
```

## 8.2 文件存储

从上一节可以知道，`SharedPreferences` 只能保存 key-value 对，虽然可以采用 Base64 编码的方式保存更复杂的数据，但仍然会受到很多限制。然而，文件存取的核心就是输入流和输出流，`SharedPreferences` 在底层同样也采用了这些流技术。如果想对文件随心所欲地控制，直接使用流是最好的选择。本节将详细介绍如何使用流、File 等底层的文件存取技术来操作文件，同时还介绍了如何操作压缩文件（jar、zip）。

### 8.2.1 openFileOutput 和 openFileInput 方法

工程目录：src\ch08\ch08\_fileoutputinput

谁是 `SharedPreferences` 的“近亲”，那么本节将告诉你答案。`openFileOutput` 和 `openFileInput` 方法与 `SharedPreferences` 在某些方面非常相似。让我们先回忆一下 `SharedPreferences` 对象是如何创建的。

```
SharedPreferences mySharedPreferences = getSharedPreferences("file", Activity.MODE_PRIVATE);
```

看看上面的代码，可能很多读者已经回忆起来了，这是使用 `SharedPreferences` 的第 1 步：创建 `SharedPreferences` 对象。`getSharedPreferences` 方法的第一个参数指定要保存在手机内存中的文件名（不包括扩展名，扩展名为 xml）。第 2 个参数表示 `SharedPreferences` 对象创建 XML 文件时设置的文件属性。

下面来看看 `openFileOutput` 方法如何返回一个 `OutputStream` 对象。

```
OutputStream os = openFileOutput("file.txt", Activity.MODE_PRIVATE);
```

从上面的代码可以看出，`openFileOutput` 方法的两个参数与 `getSharedPreferences` 方法类似，只是第一个参数指定的文件名多了一个扩展名。从这两个方法可以看出，第 1 个参数只指定了文件名，并未包含文件路径，因此，这两个方法只能将文件保存在手机内存中固定的路径。在前面已经知道，`SharedPreferences` 将 XML 文件保存在 /data/data/<package name>/shared\_prefs 目录中，而 `openFileOutput` 方法将文件保存在 /data/data/<package name>/files 目录中。

在使用 `openFileInput` 方法获得 `InputStream` 对象来读取文件中的数据时，只需要指定文件名即可。

```
InputStream is = openFileInput("file.txt");
```

使用 `openFileOutput` 和 `openFileInput` 方法获得 `OutputStream` 及 `InputStream` 对象来读写文件的完整代码如下：

```
// 向文件写入内容
OutputStream os = openFileOutput("file.txt", Activity.MODE_PRIVATE);
String str1 = "《Android/OPhone 开发完全讲义》";
os.write(str1.getBytes("utf-8"));
os.close();
```

```
// 读取文件的内容
InputStream is = openFileInput("file.txt");
byte[] buffer = new byte[100];
int byteCount = is.read(buffer);
String str2 = new String(buffer, 0, byteCount, "utf-8");
TextView textView = (TextView) findViewById(R.id.textview);
textView.setText(str2);
is.close();
```

运行本节的例子后，将看到如图 8.8 所示的输出信息。



▲ 图 8.8 使用 openFileOutput 和 openFileInput 方法存取数据

虽然 openFileOutput 和 openFileInput 方法可以获得操作文件的 OutputStream 及 InputStream 对象，而且通过流对象可以任意处理文件中的数据，但这两个方法与 SharedPreferences 一样，只能在手机内存卡的指定目录建立文件，因此，它们在使用上仍然有一定的局限性。在下一小节读者可以看到如何使用更高级的方法读写 SD 卡中的文件内容。

### 8.2.2 读写 SD 卡中的文件

工程目录: src\ch08\ch08\_sdcard\_fileoutputinput

在 Android 中也可以使用传统的方法来操作文件。例如，FileInputStream 和 FileOutputStream 用来读写指定路径的文件。在本小节将学习一下如何利用这两个类将 apk 文件中的图像资源保存到 SD 卡上，并从 SD 卡读取图像文件显示在 ImageView 控件中。在运行本例前，请确定 Android 模拟器或手机中已安装了 SD 卡。

首先准备一个图像文件 (image.jpg)，将该文件放到 assets 目录中，并使用如下的代码将 image.jpg 保存到 SD 卡的根目录。

```
try
{
    // 创建用于将图像保存到 SD 卡上的 FileOutputStream 对象
    FileOutputStream fos = new FileOutputStream(
        android.os.Environment.getExternalStorageDirectory() + "/image.jpg");
    // 打开 assets 目录中的 image.jpg 文件，并返回 InputStream 对象
    InputStream is = getResources().getAssets().open("image.jpg");
    // 定义一个 byte 数组，用于保存每次向 SD 卡中文件写入的数据，最多 8K
    byte[] buffer = new byte[8192];
    int count = 0;
    // 循环写入数据
    while ((count = is.read(buffer)) >= 0)
    {
```

```

        fos.write(buffer, 0, count);
    }
    fos.close();
    is.close();
    Toast.makeText(this, "已成功将图像文件写到 SD 卡上。", Toast.LENGTH_LONG).show();
}
catch (Exception e)
{
}

```

**注意**

虽然确定 SD 卡的路径是可以直接使用 “/sdcard” 的，但在实际应用中建议使用 android.os.Environment.getExternalStorageDirectory 方法获得 SD 卡的路径。这样一旦系统改变了路径，应用程序会立刻获得最新的 SD 卡路径，这样做会使程序更健壮。

下面的代码从 SD 卡中读取 image.jpg 文件，并将图像显示在 ImageView 控件中。

```

// 指定 SD 卡中的图像文件名
String filename = android.os.Environment.getExternalStorageDirectory() + "/image.jpg";
// 判断该文件是否存在
if (!new File(filename).exists())
{
    Toast.makeText(this, "还没有往 SD 卡写入图像文件", Toast.LENGTH_LONG).show();
    return;
}
ImageView imageView = (ImageView) findViewById(R.id.imageview);
try
{
    FileInputStream fis = new FileInputStream(filename);
    // 从文件的输入流装载 Bitmap 对象
    Bitmap bitmap = BitmapFactory.decodeStream(fis);
    // 将图像显示在 ImageView 控件中
    imageView.setImageBitmap(bitmap);
    fis.close();
}
catch (Exception e)
{
}

```

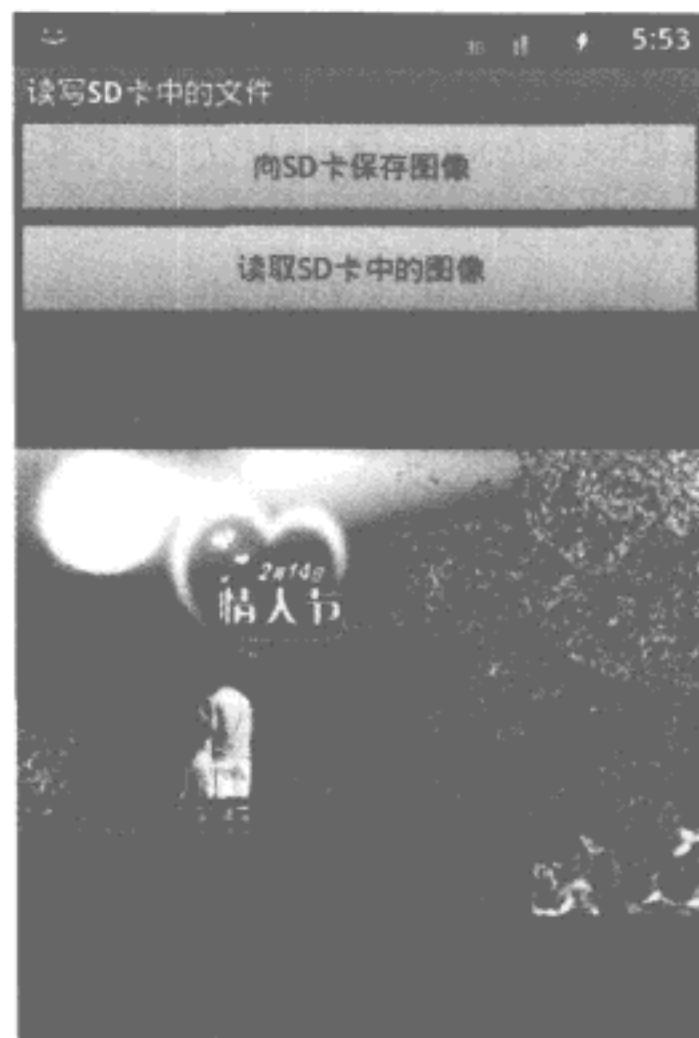
虽然上面两段代码已经完美地读写了 SD 卡中的图像文件，但从 Android 2.x 开始，默认不允许向 SD 卡中写文件，因此，需要在 `AndroidManifest.xml` 文件加入如下代码打开写文件的权限。

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

运行本例，先向 SD 卡中写入图像文件，再从 SD 卡装载图像，显示效果如图 8.9 所示。

**多学一招：**Android 2.3 提供了对不带 SD 卡的设备的离线支持，当然，对于没有 SD 卡的情况，也可以采用虚拟 SD 卡分区，因为 Android 是基于 Linux 内核的，在 Linux 中，任何设备和存储介质都会被映射成文件或目录。对于 SD 卡来说，一般会被映射成 “/sdcard”。即使没有物理的 SD 卡，在程序中也同样可以使用 “/sdcard” 来访问虚拟的 SD 卡。例如，Google Nexus S 手机虽不允许插入扩展的 SD 卡，但手机可用于存储的内存高达 16GB，在默认情况下会将一部分存储空间分给虚

拟的 SD 卡使用（一部分用于安装 Android 操作系统）。



▲ 图 8.9 读写 SD 卡中的图像文件

既然有虚拟和物理两种 SD 卡，我们就需要知道哪些 SD 卡是虚拟的，哪些 SD 卡是物理的。通过下面的代码可以知道当前设备中是哪种 SD 卡。

```
| android.os.Environment.getExternalStorageRemovable()
```

`isExternalStorageRemovable` 方法返回 `true`，表示 SD 卡是物理的，如果该方法返回 `false`，表示 SD 卡是虚拟的。

### 8.2.3 SAX 引擎读取 XML 文件的原理

从前面的内容知道，`SharedPreferences` 操作的文件是 XML 格式的，只是 `SharedPreferences` 将操作 XML 文件的具体细节隐藏了。本节将向读者展示如何通过 SAX 技术操作 XML 文件。

虽然可以使用很多第三方的 jar 包来操作 XML，但 Android SDK 本身已经提供了操作 XML 文件的类库，这就是 SAX。使用 SAX 处理 XML 需要一个 `Handler` 对象，一般会使用一个 `org.xml.sax.helpers.DefaultHandler` 的子类来创建 `Handler` 对象。

SAX 技术在处理 XML 文件时并不一次性把 XML 文件装入内存，而是一边读一边解析。因此，这就需要处理如下 5 个分析点，也可称为分析事件。

- 开始分析 XML 文件。该分析点表示 SAX 引擎刚开始处理 XML 文件，还没有读取 XML 文件中的内容。该分析点对应于 `DefaultHandler.startDocument` 方法，可以在该方法中做一些初始化的工作。
- 开始处理每一个 XML 标签，也就是遇到`<product>`、`<item>`这样的起始标记。SAX 引擎每次扫描到新的 XML 标签的起始标记时会触发这个分析事件，对应的事件方法是 `startElement`。在该方法中可以获得当前标签的名称、属性的相关信息。

- 处理完一个 XML 标签，也就是遇到</product>、</item>这样的结束标记。该分析点对应的事件方法是 `endElement`，在该事件中可以获得当前处理完的标签的全部信息。
- 处理完 XML 文件。如果 SAX 引擎将整个 XML 文件的内容都扫描完了，就到了这个分析点，该分析点对应的事件方法是 `endDocument`。该事件方法可能不是必需的，如果最后有一些收尾工作，如释放一些资源，可以在该方法中完成。
- 读取字符分析点。这是最重要的分析点，如果没有这个分析点，前 4 步的处理相当于白跑一遍，虽然读取了 XML 文件中的所有内容，但并未处理这些内容，而这个分析点所对应的 `characters` 事件方法的主要作用就是处理 SAX 引擎读取的 XML 文件中的内容，更准确地说是保存 XML 标签的内容，也就是<product>abc</product>中的 abc。

了解了 SAX 引擎读取 XML 文件的原理，使用起来就容易多了，读者在下一小节将会看到如何将 XML 文件转换成一个 Java 对象。

#### 8.2.4 将 XML 文件转换成 Java 对象

**工程目录：src\ch08\ch08\_xml\_object**

本小节给出了一个例子来演示如何利用 SAX 技术操作 XML 文件，并将有一定格式的 XML 文件转换成 Java 对象。先按照下面的代码准备一个 XML 文件(products.xml)，并将该文件放到 res\raw 目录中。

##### products.xml

```
<?xml version="1.0" encoding="utf-8"?>
<products>
    <product>
        <id>10</id>
        <name>电脑</name>
        <price>2067.25</price>
    </product>
    <product>
        <id>20</id>
        <name>微波炉</name>
        <price>520</price>
    </product>
    <product>
        <id>30</id>
        <name>洗衣机</name>
        <price>2400</price>
    </product>
</products>
```

在上面的 XML 文件中定义了 3 个<product>标签，在本例中可以将这 3 个<product>标签分别映射成 3 个 Product 对象。因此，需要定义一个 Product 类，代码如下：

```
package mobile.android.ch08.xml.object;

public class Product
{
    private int id;
```

```

private String name;
private float price;
...
// 此处省略了属性的 getter 和 setter 方法
}

```

上面 XML 文件<product>标签中的 3 个子标签的值与 Product 类的 3 个属性对应。

XML2Product 类是本例的核心，该类是 DefaultHandler 的子类，负责处理在 8.2.3 小节介绍的 5 个分析点事件。该类的代码如下：

```

package mobile.android.ch08.xml.object;

import java.util.ArrayList;
import java.util.List;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class XML2Product extends DefaultHandler
{
    private List<Product> products;           // 该变量用于保存转换后的结果
    private Product product;
    private StringBuffer buffer = new StringBuffer();
    public List<Product> getProducts()
    {
        return products;
    }
    @Override
    public void characters(char[] ch, int start, int length) throws SAXException
    {
        buffer.append(ch, start, length);
        super.characters(ch, start, length);
    }
    @Override
    public void endElement(String uri, String localName, String qName) throws SAXException
    {
        if (localName.equals("product"))
        {
            // 处理完<product>标签后，将 Product 对象添加到 products 中
            products.add(product);
        }
        else if (localName.equals("id"))
        {
            // 设置 id 属性的值
            product.setId(Integer.parseInt(buffer.toString().trim()));
            // 将保存标签内容的缓存区清空
            buffer.setLength(0);
        }
        else if (localName.equals("name"))
        {
            // 设置 name 属性的值
            product.setName(buffer.toString().trim());
            // 将保存标签内容的缓存区清空
        }
    }
}

```

```

        buffer.setLength(0);
    }
    else if (localName.equals("price"))
    {
        // 设置 price 属性的值
        product.setPrice(Float.parseFloat(buffer.toString().trim()));
        // 将保存标签内容的缓存区清空
        buffer.setLength(0);
    }
    super.endElement(uri, localName, qName);
}
@Override
public void startDocument() throws SAXException
{
    // 开始分析 XML 文档，创建 List 对象用于保存分析完的 Product 对象
    products = new ArrayList<Product>();
}
@Override
public void startElement(String uri, String localName, String qName,
    Attributes attributes) throws SAXException
{
    if (localName.equals("product"))
    {
        // 如果开始分析的是<product>标签，创建一个 Product 对象
        product = new Product();
    }
    super.startElement(uri, localName, qName, attributes);
}
}

```

让我们看看 XML2Product 类在这 5 个分析点事件方法中做了哪些事情。

- **startDocument:** 第 1 个分析点事件方法。在该方法中创建了用于保存转换结果的 List<Product>对象。
- **startElement:** 第 2 个分析点事件方法。SAX 引擎分析到每一个<product>标签时，在该方法中都会创建一个 Product 对象。
- **endElement:** 第 3 个分析点事件方法。该方法中的代码最复杂，但如果仔细看一下，其实很简单。当 SAX 引擎每分析完一个 XML 标签后，会将该标签中的内容保存在 Product 对象的相应属性中。
- **endDocument:** 第 4 个分析点事件方法。在该方法中什么都没做，也没覆盖这个方法。
- **characters:** 第 5 个分析点事件方法。虽然该方法中的代码由开发人员编写的只有一行，但十分关键。在该方法中将 SAX 引擎扫描到的内容保存在 buffer 变量中，而在 endElement 方法中要使用该变量中的值来为 Product 对象中的属性赋值。

下面来看看如何使用 XML2Product 类将一个 XML 文件转换成 Java 对象。

```

try
{
    // 获得 res/raw/products.xml 文件中 InputStream 对象
    InputStream is = getResources().openRawResource(R.raw.products);
}

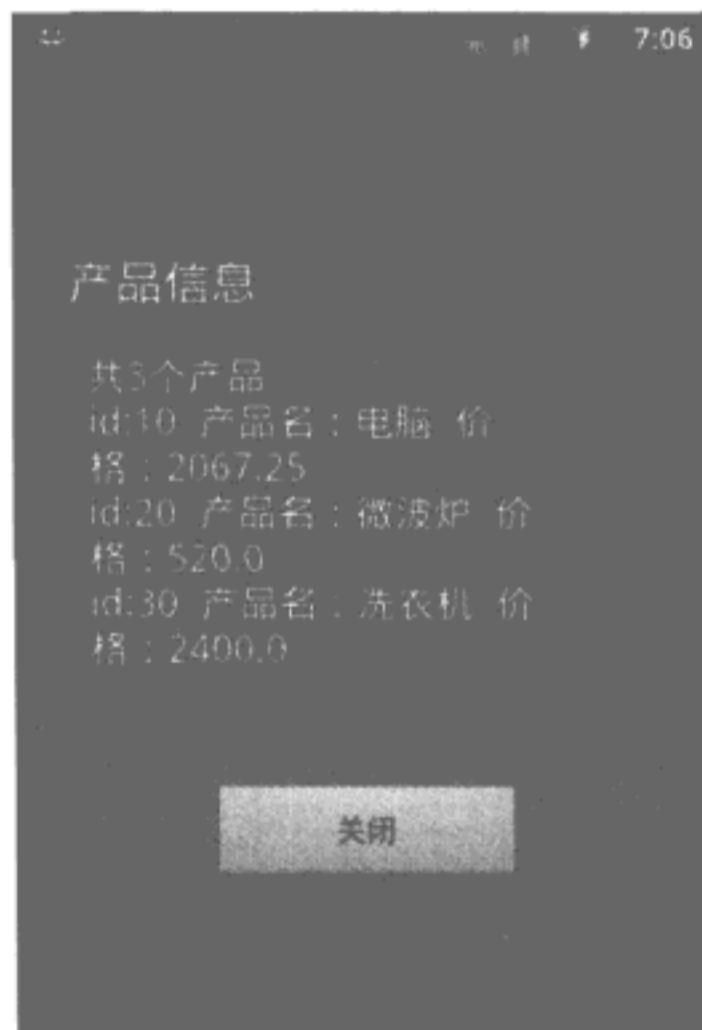
```

```

XML2Product xml2Product = new XML2Product();
// 开始分析products.xml 文件
android.util.Xml.parse(is, Xml.Encoding.UTF_8, xml2Product);
// 下面的代码用于输出转换后的 Java 对象的内容
List<Product> products = xml2Product.getProducts();
String msg = "共" + products.size() + "个产品\n";
for (Product product : products)
{
    msg += "id:" + product.getId() + " 产品名：" + product.getName()
        + " 价格：" + product.getPrice() + "\n";
}
new AlertDialog.Builder(this).setTitle("产品信息").setMessage(msg)
    .setPositiveButton("关闭", null).show();
}
catch (Exception e)
{
}

```

将 XML 文件的内容转换成 Java 对象，并输出对象属性信息的效果如图 8.10 所示。



▲ 图 8.10 显示 XML 文件中的内容

### 8.2.5 文件压缩 ( Jar、Zip )

工程目录: src\ch08\ch08\_jar\_zip

Android SDK 中提供可以将文件压缩成 jar 和 zip 包的功能，这两个功能分别由 java.util.jar 和 java.util.zip 包中的若干类和接口来完成。压缩成 jar 和 zip 包的方法类似，压缩文件的基本步骤如下。

- (1) 创建一个 JarOutputStream 或 ZipOutputStream 对象，分别用来生成 jar 或 zip 文件。
- (2) 创建了 (Jar | Zip) OutputStream 对象，还不能直接向输出流中写入数据。由于压缩文件中也需要使用文件名和路径来标识被压缩的文件，因此，需要为每一个要压缩的文件创建一个

JarEntry 或 ZipEntry 对象。每一个 (Jar | Zip) Entry 对象代表一个被压缩的文件，并通过 (Jar | Zip) Entry 对象指定被压缩文件在压缩包中的文件名和路径。

- (3) 调用 (Jar | Zip) OutputStream.putNextEntry 方法设置当前打开的 (Jar | Zip) Entry 对象。
- (4) 向 (Jar | Zip) OutputStream 对象中写入数据。
- (5) 调用 (Jar | Zip) OutputStream.closeEntry 方法关闭当前打开的 (Jar | Zip) Entry 对象。如果还有待压缩的文件，回到第 2 步继续执行。

解压缩的步骤如下。

- (1) 创建一个 JarInputStream 或 ZipInputStream 对象。该对象一般通过 FileInputStream 对象指定要解压的文件。
- (2) 使用 (Jar | Zip) InputStream.getNextEntry 方法枚举压缩包中所有的文件。如果 getNextEntry 方法返回 null，表示所有的压缩文件已经被处理完。
- (3) 通过 (Jar | Zip) Entry.getName 方法获得文件压缩后的路径和文件名（为了保证解压后的文件名与被压缩前是一致的，当然，也可以使用其他文件名），并使用 FileOutputStream 对象指定已解压的文件。
- (4) 向 FileOutputStream 对象输出已解压的数据流。
- (5) 调用 (Jar | Zip) Entry.closeEntry 方法关闭当前打开的 (Jar | Zip) Entry 对象。如果压缩包中还有要解压的文件，回到第 2 步继续执行。

下面我们来具体看一下压缩和解压的代码。

将一个文件压缩成 file.jar，代码如下：

```

try
{
    // 使用 FileOutputStream 对象指定一个要输出的压缩文件 (file.jar)
    FileOutputStream fos = new FileOutputStream(
        android.os.Environment.getExternalStorageDirectory() + "/file.jar");
    // 第1步：创建 JarOutputStream 对象
    JarOutputStream jos = new JarOutputStream(fos);
    // 第2步：创建一个 JarEntry 对象，并指定待压缩文件在压缩包中的文件名 (strings.xml)
    JarEntry jarEntry = new JarEntry("strings.xml");
    // 第3步：使用 putNextEntry 方法打开当前的 JarEntry 对象
    jos.putNextEntry(jarEntry);
    InputStream is = getResources().getAssets().open("strings.xml");
    byte[] buffer = new byte[8192];
    int count = 0;
    // 第4步：写入数据
    while ((count = is.read(buffer)) >= 0)
    {
        jos.write(buffer, 0, count);
    }
    is.close();
    // 第5步：关闭当前的 JarEntry 对象
    jos.closeEntry();
    jos.close();
    Toast.makeText(this, "成功将 strings.xml 文件以 jar 格式压缩。", Toast.LENGTH_LONG).show();
}

```

```

}
catch (Exception e)
{
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}

```

解压 file.jar 文件的代码如下：

```

try
{
    // 定义要解压的文件
    String filename = android.os.Environment.getExternalStorageDirectory() + "/file.jar";
    if (!new File(filename).exists())
    {
        Toast.makeText(this, "压缩文件不存在。", Toast.LENGTH_LONG).show();
        return;
    }
    // 使用 FileInputStream 对象指定要解压的文件
    FileInputStream fis = new FileInputStream(filename);
    // 第1步：创建 JarInputStream 对象来读取压缩文件（file.jar）
    JarInputStream jis = new JarInputStream(fis);
    // 第2步：调用 getNextJarEntry 方法打开压缩包中第一个文件（如果有多个压缩文件，可多次调用该方法）
    JarEntry jarEntry = jis.getNextJarEntry();
    // 第3步：输出已解压的文件
    FileOutputStream fos = new FileOutputStream(
        android.os.Environment.getExternalStorageDirectory() + "/" + jarEntry.getName());
    byte[] buffer = new byte[8192];
    int count = 0;
    // 第4步：输出已解压文件的字节流
    while ((count = jis.read(buffer)) >= 0)
    {
        fos.write(buffer, 0, count);
    }
    // 第5步：调用 closeEntry 方法关闭当前的 JarEntry 对象
    jis.closeEntry();
    jis.close();
    fos.close();
    Toast.makeText(this, "成功解压 jar 格式的文件。", Toast.LENGTH_LONG).show();
}
catch (Exception e)
{
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}

```

下面来看一下如何将多个文件压缩成 zip 包。

```

try
{
    // 指定了两个待压缩的文件，都在 assets 目录中
    String[] filenames = new String[]{"main.xml", "strings.xml"};
    FileOutputStream fos = new FileOutputStream(
        android.os.Environment.getExternalStorageDirectory() + "/file.zip");
    ZipOutputStream zos = new ZipOutputStream(fos);
    int i = 1;
    // 枚举 filenames 中的所有待压缩文件

```

```

while (i <= filenames.length)
{
    // 从 filenames 数组中取出当前待压缩的文件名，作为压缩后的名称，以保证压缩前后文件名一致
    ZipEntry zipEntry = new ZipEntry(filenames[i - 1]);
    // 打开当前的 ZipEntry 对象
    zos.putNextEntry(zipEntry);
    InputStream is = getResources().getAssets().open(filenames[i - 1]);
    byte[] buffer = new byte[8192];
    int count = 0;
    // 写入数据
    while ((count = is.read(buffer)) >= 0)
    {
        zos.write(buffer, 0, count);
    }
    zos.flush();
    // 关闭当前的 ZipEntry 对象
    zos.closeEntry();
    is.close();
    i++;
}
zos.finish();
zos.close();
Toast.makeText(this, "成功将 main.xml、strings.xml 文件以 zip 格式压缩。",
    Toast.LENGTH_LONG).show();

}
catch (Exception e)
{
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}

```

解压 file.zip 文件的代码如下：

```

try
{
    // 指定待解压的文件
    String filename = android.os.Environment.getExternalStorageDirectory() + "/file.zip";
    if (!new File(filename).exists())
    {
        Toast.makeText(this, "压缩文件不存在。", Toast.LENGTH_LONG).show();
        return;
    }
    FileInputStream fis = new FileInputStream(filename);
    ZipInputStream zis = new ZipInputStream(fis);
    ZipEntry zipEntry = null;
    // 通过不断调用 getNextEntry 方法来解压 file.zip 中的所有文件
    while ((zipEntry = zis.getNextEntry()) != null)
    {
        FileOutputStream fos = new FileOutputStream(
            android.os.Environment.getExternalStorageDirectory() + "/" + zipEntry.
            getName());
        byte[] buffer = new byte[8192];
        int count = 0;
        while ((count = zis.read(buffer)) >= 0)
        {
            fos.write(buffer, 0, count);
        }
    }
}

```

```

    }
    zis.closeEntry();
    fos.close();
}
zis.close();
Toast.makeText(this, "成功解压 jar 格式的文件。", Toast.LENGTH_LONG).show();
}
catch (Exception e)
{
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}

```

## 8.3 SQLite 数据库

现在终于到讲解数据库的时间了，数据库也是 Android 存储方案的核心，在 Android 系统中使用了 SQLite 数据库。SQLite 是非常轻量的数据库，从 SQLite 的标志是一根羽毛可以看出 SQLite 的目标就是无论是过去、现在，还是将来，SQLite 都将以轻量级数据库的姿态出现。SQLite 虽然轻量，但在执行某些简单的 SQL 语句时甚至比 MySQL 和 Postgresql 还快。很多读者是第一次接触 SQLite 数据库，因此，本节会首先介绍一下如何在 PC 上创建和管理 SQLite 数据，然后介绍一下如何将控件（例如 ListView、Gallery）与 SQLite 数据库进行绑定，最后会介绍一些技巧，例如，如何将 SQLite 数据库封装在 apk 中与应用程序一起发布。

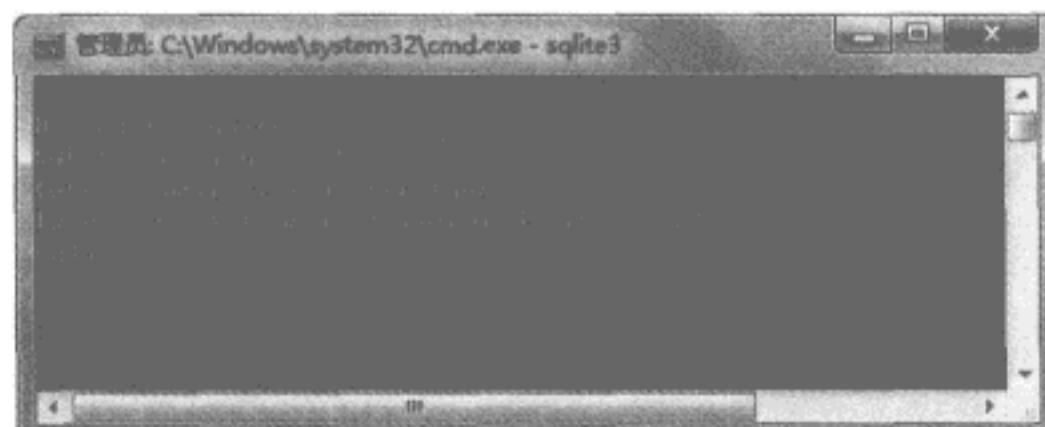
### 8.3.1 SQLite 数据库管理工具

在学习一种新技术之前，首先要做的是在自己的计算机上安装可以使用这种技术的工具。当然，这也非常符合一句成语：工欲善其事，必先利其器。虽然使用好的工具并不能使自己更好地掌握这种技术，但却能使我们的工作效率大大提升。

言归正传，现在先看看官方为我们提供了什么工具来操作 SQLite 数据库。进入官方的下载页面，网址如下：

<http://www.sqlite.org/download.html>

在下载页面中找到 Windows 版的二进制下载包。在笔者写作本书时，SQLite 的最新版本是 SQLite 3.7.5，因此，要下载的文件是 sqlite-shell-win32-x86-3070500.zip。将这个 zip 文件解压，发现在解压目录中只有一个文件：sqlite3.exe，这个文件就是操作 SQLite 数据库的工具（是不是很轻量！连工具都只有一个）。它是一个命令行程序，运行这个程序，进入操作界面，如图 8.11 所示。



▲ 图 8.11 SQLite 的命令行控制台

在控制台中可以输入 SQL 语句或控制台命令，所有的 SQL 语句后面必须以分号（;）结尾，控制台命令必须以实心点（.）开头，例如，.help（显示帮助信息）；.quit（退出控制台）；.tables（显示当前数据库中的所有表名）。

虽然可以在 SQLite 的控制台中输入 SQL 语句来操作数据库，但输入大量的命令会使工作量大大增加，因此，必须要使用所谓的“利器”来取代这个控制台程序。

SQLite 提供了各种类型的程序接口，因此，可以管理 SQLite 数据库的工具非常多，下面是几个比较常用的 SQLite 管理工具。

### SQLite Database Browser

<http://sourceforge.net/projects/sqlitebrowser>

### SQLite Expert Professional

<http://www.sqliteexpert.com>

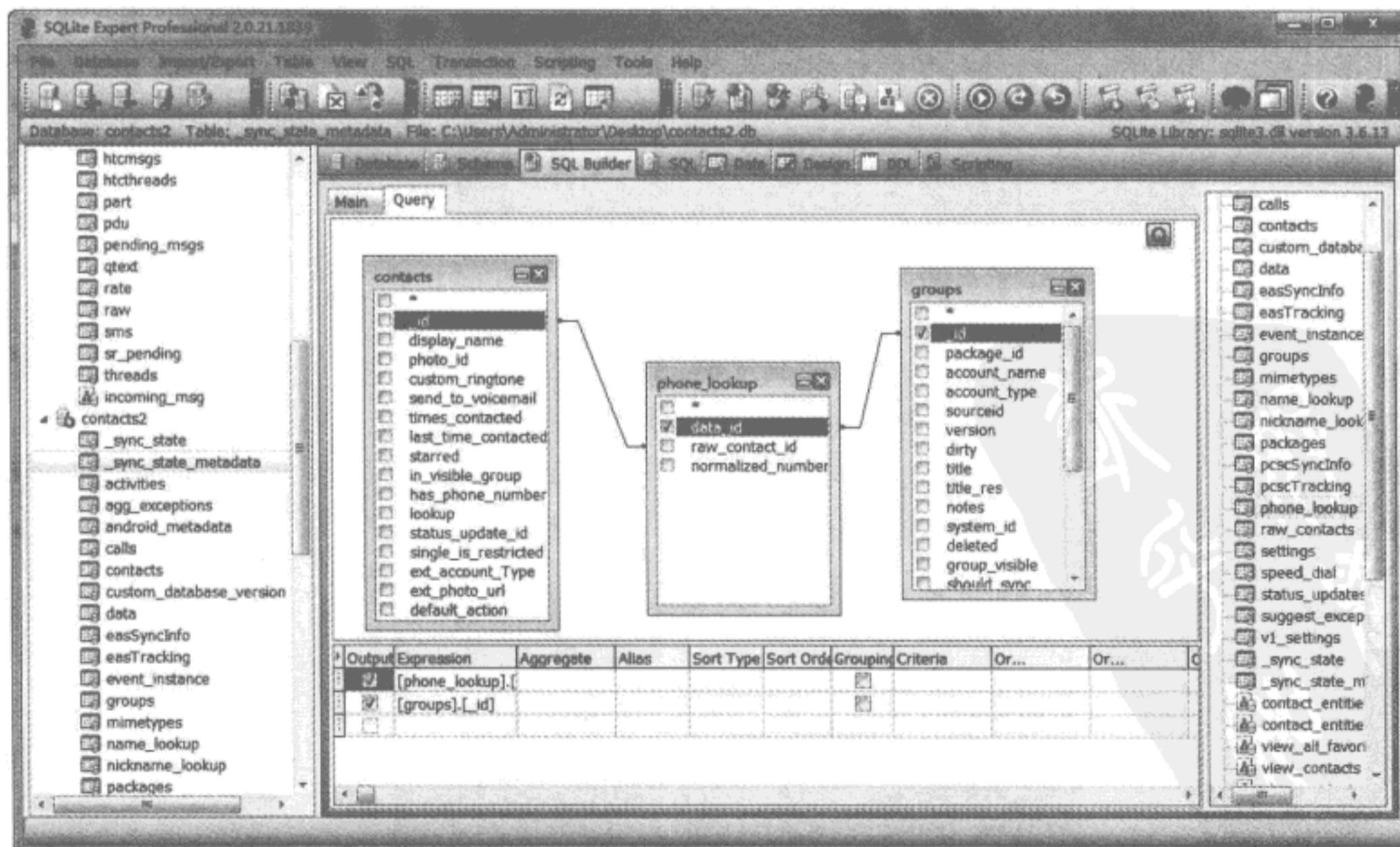
### SQLite Developer

<http://www.sqlitedeveloper.com>

### sqliteSpy

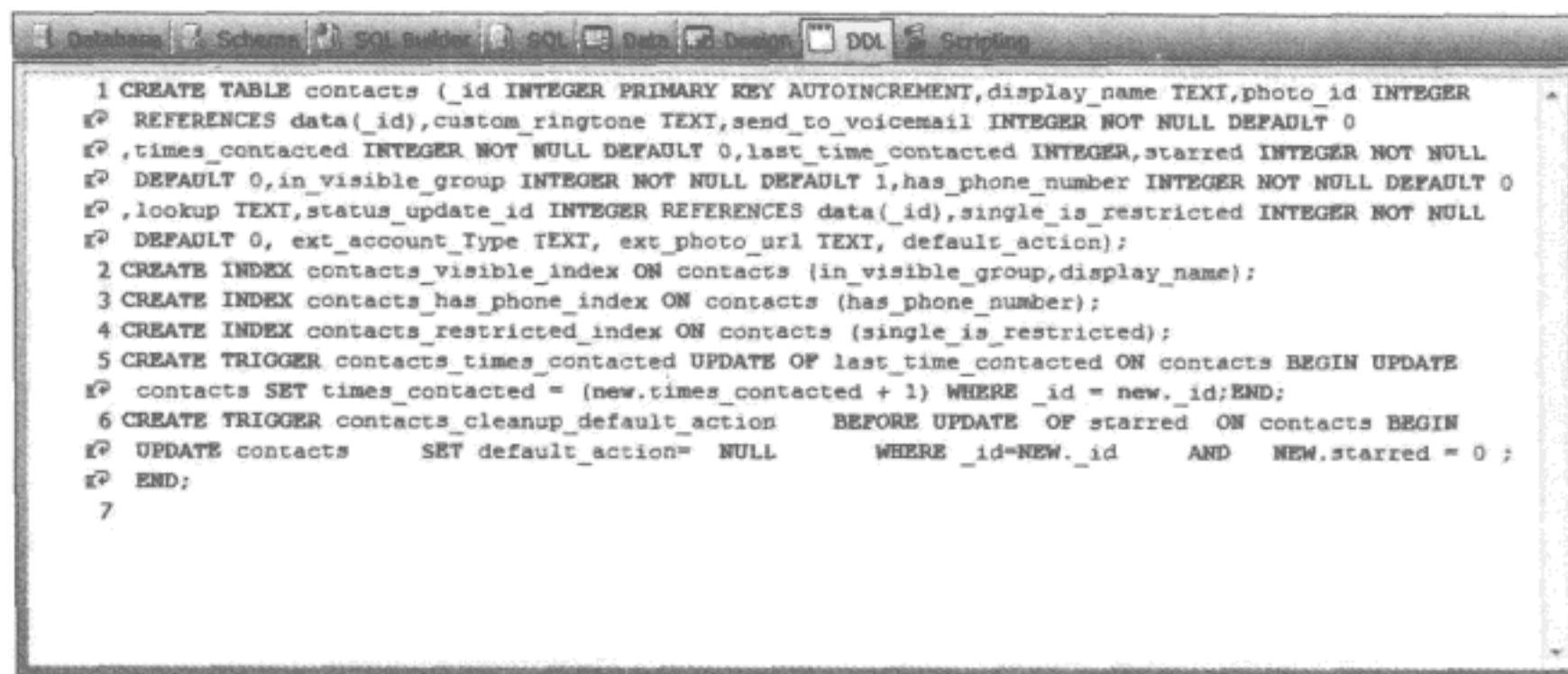
<http://www.softpedia.com/progDownload/SQLiteSpy-Download-107386.html>

笔者在写作本书时使用了 SQLite Expert Professional，这也是笔者推荐使用的 SQLite 管理工具。该工具拥有大量的可视化功能，例如，建立数据库、建立表、SQL Builder 等工具。图 8.12 是 SQLite Expert Professional 的主界面。



▲ 图 8.12 SQLite Expert Professional 的主界面

SQLite数据库支持的功能比较多，除了支持表、视图、索引、触发器等数据库必备的功能外，还支持事务等高级特性。如果读者想了解具体的SQL语句的写法，可以利用SQLite Expert Professional工具或查看官方文档(<http://www.sqlite.org/lang.html>)。例如，想了解建表的SQL语句，可以使用SQLite Expert Professional建立一个表，并选中该表或选中已经存在的表(在如图8.12所示的界面左侧是表和视图的列表)，然后打开“DDL”页面，就会看到建立该表所需要的Create Table语句的具体用法，如图8.13所示。



```

1 CREATE TABLE contacts (_id INTEGER PRIMARY KEY AUTOINCREMENT,display_name TEXT,photo_id INTEGER
2 REFERENCES data(_id),custom_ringtone TEXT,send_to_voicemail INTEGER NOT NULL DEFAULT 0
3 ,times_contacted INTEGER NOT NULL DEFAULT 0,last_time_contacted INTEGER,starred INTEGER NOT NULL
4 DEFAULT 0,in_visible_group INTEGER NOT NULL DEFAULT 1,has_phone_number INTEGER NOT NULL DEFAULT 0
5 ,lookup TEXT,status_update_id INTEGER REFERENCES data(_id),single_is_restricted INTEGER NOT NULL
6 DEFAULT 0, ext_account_Type TEXT, ext_photo_url TEXT, default_action);
7 CREATE INDEX contacts_visible_index ON contacts (in_visible_group,display_name);
8 CREATE INDEX contacts_has_phone_index ON contacts (has_phone_number);
9 CREATE INDEX contacts_restricted_index ON contacts (single_is_restricted);
10 CREATE TRIGGER contacts_times_contacted UPDATE OF last_time_contacted ON contacts BEGIN UPDATE
11 contacts SET times_contacted = (new.times_contacted + 1) WHERE _id = new._id;END;
12 CREATE TRIGGER contacts_cleanup_default_action BEFORE UPDATE OF starred ON contacts BEGIN
13 UPDATE contacts SET default_action= NULL WHERE _id=NEW._id AND NEW.starred = 0 ;
14 END;
15

```

▲图8.13 查看建表的SQL语句

### 8.3.2 SQLiteOpenHelper类与自动升级数据库

`android.database.sqlite.SQLiteDatabase`是Android SDK中操作数据库的核心类之一，使用`SQLiteDatabase`可以打开数据库，也可以对数据库进行操作。然而，为了数据库升级的需要以及使用更方便，往往使用`SQLiteOpenHelper`的子类来完成创建、打开数据库及各种数据库的操作。

`SQLiteOpenHelper`是一个抽象类，在该类中有如下两个抽象方法，`SQLiteOpenHelper`的子类必须实现这两个方法。

```

public abstract void onCreate(SQLiteDatabase db);
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion);

```

`SQLiteOpenHelper`会自动检测数据库文件是否存在，如果数据库文件存在，会打开这个数据库，在这种情况下，并不会调用`onCreate`方法。如果数据库文件不存在，`SQLiteOpenHelper`首先会创建一个数据库文件，然后打开这个数据库，最后会调用`onCreate`方法。因此，`onCreate`方法一般用来在新创建的数据库中建立表、视图等数据库组件。也就是说，`onCreate`方法在数据库文件第一次被创建时调用。

先看看`SQLiteOpenHelper`类的构造方法再解释`onUpgrade`方法何时会被调用。

```

public SQLiteOpenHelper(Context context, String name, CursorFactory factory, int version);

```

其中`name`参数表示数据库文件名(不包含文件路径)，`SQLiteOpenHelper`会根据这个文件名创建数据库文件。`version`表示数据库的版本号，如果当前传递的数据库版本号比上次创建或升级的数据库版本号高，`SQLiteOpenHelper`就会调用`onUpgrade`方法。也就是说，当数据库第1次创建

时会有一个初始的版本号。当需要对数据库中表、视图等组件升级时可以增大版本号，这时 SQLiteOpenHelper 会调用 `onUpgrade` 方法，当调用完 `onUpgrade` 方法后，系统会更新数据库的版本号。这个当前的版本号就是通过 `SQLiteOpenHelper` 类的最后一个参数 `version` 传入 `SQLiteOpenHelper` 对象的，因此，在 `onUpgrade` 方法中一般会首先删除要升级的表、视图等组件，再重新创建它们。也许很多读者看到这里还是比较模糊，不知如何应用 `SQLiteOpenHelper` 来操作数据库，不过这不要紧，在下一小节将详细演示 `SQLiteOpenHelper` 类的使用方法。下面来总结一下 `onCreate` 和 `onUpgrade` 方法的调用过程。

- 如果数据库文件不存在，`SQLiteOpenHelper` 在自动创建数据库后只会调用 `onCreate` 方法，在该方法中一般需要创建数据库中的表、视图等组件。在创建之前，数据库是空的，因此，不需要先删除数据库中相关的组件。
- 如果数据库文件存在，并且当前的版本号高于上次创建或升级时的版本号，`SQLiteOpenHelper` 会调用 `onUpgrade` 方法，调用该方法后，会更新数据库版本号。在 `onUpgrade` 方法中除了创建表、视图等组件外，还需要首先删除这些组件，因此，在调用 `onUpgrade` 方法之前，数据库是存在的，里面还有很多数据库组件。

综合上述两点，可以得出一个结论，如果数据库文件不存在，只有 `onCreate` 方法被调用（该方法只会在创建数据库时被调用 1 次）；如果数据库文件存在，并且当前版本较高，会调用 `onUpgrade` 方法来升级数据库，并更新版本号。

### 8.3.3 数据绑定与 SimpleCursorAdapter 类

**工程目录:** src\ch08\ch08\_simplecursoradapter

很多时候需要将表中的数据显示在 `ListView`、`Gallery` 等控件中，虽然可以直接使用 `BaseAdapter` 进行处理，但工作量比较大。为此，Android SDK 提供了一个专用于数据绑定的 Adapter 类：`SimpleCursorAdapter`。

`SimpleCursorAdapter` 与 `SimpleAdapter` 的使用方法非常接近，只是将数据源从 `List` 对象换成了 `Cursor` 对象，而且 `SimpleCursorAdapter` 类构造方法的第 4 个参数 `from` 表示 `Cursor` 对象中的字段，而 `SimpleAdapter` 类构造方法的第 4 个参数 `from` 表示 `Map` 对象中的 `key`。除此之外，这两个 Adapter 类的使用方法完全相同。

下面是 `SimpleCursorAdapter` 类构造方法的定义。

```
| public SimpleCursorAdapter(Context context, int layout, Cursor c, String[] from, int[] to)
```

本小节的例子中会通过 `SimpleCursorAdapter` 类将表中的数据显示在 `ListView` 上，也就是说，`ListView` 控件会显示表中的全部记录。在显示数据之前，需要先编写一个 `DBService` 类，该类是 `SQLiteOpenHelper` 的子类，用于操作数据库，代码如下：

```
package mobile.android.ch08.simple.cursor.adapter;

import java.util.Random;
import android.content.Context;
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DBService extends SQLiteOpenHelper {
    private final static int DATABASE_VERSION = 1;
    private final static String DATABASE_NAME = "test.db";
    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE [t_test] (" + "[_id] AUTOINC,"
            + "[name] VARCHAR(20) NOT NULL ON CONFLICT FAIL,"
            + "CONSTRAINT [sqlite_autoindex_t_test_1] PRIMARY KEY ([_id]));"
        db.execSQL(sql);
        // 向t_test表中插入20条记录
        Random random = new Random();
        for (int i = 0; i < 20; i++) {
            String s = "";
            // 随机生成长度为10的字符串
            for (int j = 0; j < 10; j++) {
                char c = (char) (97 + random.nextInt(26));
                s += c;
            }
            // 执行insert语句
            db.execSQL("insert into t_test(name) values(?)", new Object[]{s});
        }
    }
    public DBService(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    // 由于不打算对test.db进行升级，因此，在该方法中没有任何代码
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
    // 执行select语句
    public Cursor query(String sql, String[] args) {
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(sql, args);
        return cursor;
    }
}
```

本例不需要对 test.db 进行升级，因此，只在 DBService.onCreate 方法中有创建数据表的代码。DBService 类创建了一个 test.db 数据库文件，并在该文件中创建了 t\_test 表。该表包含两个字段：\_id 和 name，其中\_id 是自增字段，并且是主索引。

下面来编写 Main 类，Main 是 ListActivity 的子类。在该类的 onCreate 方法中创建了 DBService 对象，然后通过 DBService.query 方法查询出 t\_test 表中的所有记录，并返回 Cursor 对象。Main 类的代码如下：

```
package mobile.android.ch08.simple.cursor.adapter;

import net.blogjava.mobile.db.DBService;
import android.app.ListActivity;
import android.database.Cursor;
import android.os.Bundle;
import android.widget.SimpleCursorAdapter;

public class Main extends ListActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        DBService dbService = new DBService(this);
        Cursor cursor = dbService.query("select * from t_test", null);
        SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_expandable_list_item_1, cursor,
                new String[]{"name"}, new int[]{ android.R.id.text1});
        setListAdapter(simpleCursorAdapter);
    }
}
```

SimpleCursorAdapter 类构造方法的第 4 个参数表示返回的 Cursor 对象中的字段名，第 5 个参数表示要显示该字段值的控件 ID，该控件在第 2 个参数指定的布局文件中定义。

运行本例后，将显示如图 8.14 所示的效果。



▲ 图 8.14 使用 SimpleCursorAdapter 操作数据

**注意**

在绑定数据时，Cursor 对象返回的记录集中必须包含一个名为“\_id”的字段，否则将无法完成数据绑定，也就是说 SQL 语句不能是 select name from t\_contacts。如果在数据表中没有“\_id”字段，可以将某个唯一索引字段或主键的别名（Alias）设为“\_id”。

**多学一招：数据库文件存在哪了？**

仅看到本小节的例子建立了 SQLite 数据库文件，那么数据库文件被放到哪个目录了呢？如果使用 SQLiteOpenHelper 类的 getReadableDatabase 或 getWritableDatabase 方法获得 SQLiteDatabase 对象，系统会在手机内存的 /data/data/<package name>/databases 目录中创建或寻找数据库文件。当然，使用这两个方法也只能打开这个目录中的数据库文件。读者将在后面的部分看到如何操作 SD 卡上的数据库文件。

### 8.3.4 操作 SD 卡上的数据库

**工程目录：**src\ch08\ch08\_sdcard\_database

使用 android.database.sqlite.SQLiteDatabase 类可以操作任何路径下有权访问的数据库文件，通过 SQLiteDatabase.openOrCreateDatabase 方法可以打开已存在的数据库，或创建新的数据库。openOrCreateDatabase 方法的定义如下：

```
public static SQLiteDatabase openOrCreateDatabase(String path, CursorFactory factory)
```

其中 path 表示数据库文件的路径，factory 参数值一般设为 null 即可。下面的代码创建了一个数据库文件（sdcard\_test.db）和一个表（t\_test），并向 t\_test 表中添加了两条记录，最后查询并显示其中一条记录的信息。

```
// 定义数据库文件的路径
String filename = android.os.Environment.getExternalStorageDirectory() + "/sdcard_
test.db";
// 定义创建表的 SQL 语句
String createTableSQL = "CREATE TABLE [t_test] (" + "[id] INTEGER,"
    + "[name] VARCHAR(20),[memo] TEXT,"
    + "CONSTRAINT [sqlite_autoindex_t_test_1] PRIMARY KEY ([id]));";
File file = new File(filename);
if (file.exists())
{
    file.delete();
}
// 创建并打开数据库文件（因为前面已经删除了已存在的同名文件）
SQLiteDatabase database = SQLiteDatabase.openOrCreateDatabase(filename, null);
// 创建 t_test 表
database.execSQL(createTableSQL);
// 创建一个 ContentValues 对象，表示要插入的记录行
ContentValues contentValues = new ContentValues();
// 开始设置三个字段的值
contentValues.put("id", 1);
contentValues.put("name", "Mike");
```

```

contentValues.put("memo", "Student");
// 向 t_test 表插入一行记录, database_insert 方法的第 2 个参数一般设为 null 即可
database.insert("t_test", null, contentValues);
// 定义插入记录的 SQL 语句
String insertSQL = "insert into t_test(id, name, memo) values(?, ?, ?)";
// 插入一行记录
database.execSQL(insertSQL, new Object[]{2, "John", "老师"});
// 定义查询记录的 SQL 语句
String selectSQL = "select name, memo from t_test where name=?";
// 查询记录
Cursor cursor = database.rawQuery(selectSQL, new String[]{"John"});
// 将记录指针指向第一条记录
cursor.moveToFirst();
// 显示当前记录中字段的值
Toast.makeText(this, cursor.getString(0) + " " + cursor.getString(1),
    Toast.LENGTH_LONG).show();
database.close();

```

操作 SQLite 数据库应注意如下几点。

- 对数据库的增、删、改、查都有两种方法。一种是使用 rawQuery 方法直接执行 SQL 语句，另一种是使用 SQLiteDatabase 类的相应方法来操作，例如，插入记录可以使用 SQLiteDatabase.insert 方法。
- 查询记录后获得的 Cursor 对象需要使用 moveToFirst、moveToNext、moveToPosition(position) 等方法将记录指针移动相应的位置（因为一开始记录指针在第一条记录的前面），否则操作数据库会抛出异常。

### 8.3.5 将数据库与应用程序一起发布

**工程目录:** src\ch08\ch08\_apk\_database

在前面的例子中都是在程序第一次启动时创建了数据库，也就是说，数据库文件是由应用程序负责创建的。一般初始状态的数据表中没有记录，就算有记录，也是由应用程序在创建数据库时添加的。在应用程序发布时并不包含带数据的数据库，但在很多情况下，应用程序需要连同带数据的数据库一起发布，这样就需要通过某种机制打开 apk 文件中带的数据库。

要满足上述需求，一般要解决如下两个技术问题：

- 如何将数据库文件连同应用程序一起发布；
- 如何打开与应用程序一起发布的数据库。

第 1 个问题比较好解决，可以事先利用前面介绍的数据库管理工具在 PC 上建立一个数据库文件，并手工或通过程序向数据表中添加相应的记录，然后将该数据库文件放到<Eclipse Android 工程目录>\res\raw 目录中。

现在来解决第 2 个问题。在发布 apk 文件时，数据库文件被打包在 apk 文件中，那么如何打开这个 apk 文件呢？事实上，并不能直接打开 apk 包中的数据库，因为如果数据库文件的尺寸有变化，就意味着 apk 文件的尺寸会变化。apk 相当于 Windows 中的 exe 文件。大家试想，exe 文件在启动时，文件大小怎么可能会发生变化呢？因此，在第 1 次运行程序时，需要将数据库文件复制到内存或 SD 卡的相应目录。复制的方法也很简单，使用 openRawResource 方法可以获得 res\raw 目录中

关联资源文件的 `InputStream` 对象。有了 `InputStream` 对象，复制文件就简单了。

当然，也可以将数据库文件与 apk 作为两个单独的文件发布，或数据库从网上下载，这样就省略了复制文件这一步，但这样会造成发布不便以及耗费过多网络流量的问题。至于使用哪种方式来发布应用程序，读者可根据实际情况来决定。

下面来做个实验，首先将一个 `apk_test.db` 数据库（至少含有两个字段，一条记录）复制到 `res\raw` 目录中，并编写下面的代码，将 `apk_test.db` 复制到 SD 卡的根目录，然后查询该数据库中的 `t_test` 表，并显示相应的字段值。

```

try
{
    // 获得 res\raw\apk_test.db 文件的 InputStream 对象
    InputStream is = getResources().openRawResource(R.raw.apk_test);
    // 在 SD 卡根目录创建一个数据库文件
    FileOutputStream fos = new FileOutputStream("/sdcard/apk_test.db");
    byte[] buffer = new byte[8192];
    int count = 0;
    // 开始复制数据库
    while ((count = is.read(buffer)) >= 0)
    {
        fos.write(buffer, 0, count);
    }
    fos.close();
    is.close();
    // 打开 SD 卡上的 apk_test.db 数据库
    SQLiteDatabase sqLiteDatabase = SQLiteDatabase.openOrCreateDatabase("/sdcard/apk_
test.db", null);
    // 查询 t_test 表中的所有记录
    Cursor cursor = sqLiteDatabase.rawQuery("select * from t_test", null);
    // 将记录指针移到第 1 条记录
    if (cursor.moveToFirst())
    {
        // 显示当前记录的第 2 个字段的值
        Toast.makeText(this, cursor.getString(1), Toast.LENGTH_LONG).show();
    }
    cursor.close();
    sqLiteDatabase.close();
}
catch (Exception e)
{
}

```

由于复制文件也需要向 SD 卡中写数据，因此，本例仍然需要在 `AndroidManifest.xml` 文件中加入如下的代码来打开向 SD 卡写数据的权限。

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

### 8.3.6 内存数据库

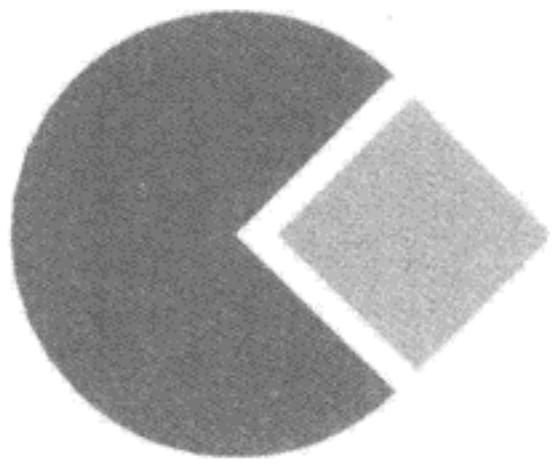
工程目录：src\ch08\ch08\_memory\_database

使用 `SQLiteDatabase` 操作 SQLite 数据库并不一定将数据库放到 SD 卡或手机内置存储器上，可以通过 `SQLiteDatabase.create` 方法创建一个内存数据库。该数据库只存在于手机的 RAM 中，当程序退出后，数据库中的数据会自动释放。下面来看一个建立内存数据库的例子。

```
String createTableSQL = "CREATE TABLE [t_test] (" + "[id] INTEGER,"  
    + "[name] VARCHAR(20), [memo] TEXT,"  
    + "CONSTRAINT [sqlite_autoindex_t_test_1] PRIMARY KEY ([id]));"  
// 创建内存数据库，该数据库只存在于手机的 RAM 中，并不保存在 SD 卡或手机内置存储器上  
SQLiteDatabase sqLiteDatabase = SQLiteDatabase.create(null);  
// 在内存数据库中创建表  
sqLiteDatabase.execSQL(createTableSQL);  
String insertSQL = "insert into t_test(id, name) values(?,?)";  
sqLiteDatabase.execSQL(insertSQL, new Object[] { 1, "老毕" });  
String selectSQL = "select name from t_test";  
// 查询内存数据库中的记录  
Cursor cursor = sqLiteDatabase.rawQuery(selectSQL, null);  
cursor.moveToFirst();  
// 显示内存数据库中的数据  
Toast.makeText(this, cursor.getString(0) ,Toast.LENGTH_LONG).show();  
sqLiteDatabase.close();
```

## 8.4 小结

本章主要介绍了 Android SDK 提供的各种数据存储的方法，主要包括 `SharedPreferences`、文件存储(文件流、jar 和 zip 压缩文件)、XML 和 SQLite 数据库 4 种数据存储方式。其中 `SharedPreferences` 在使用上较简单，主要保存文本形式的 key-value 对。`XML` 则可以保存更复杂的，带有一定结构的文本内容。如果要保存大量的二进制数据，可以直接使用文件流。如果文件过大，可以考虑使用 jar 或 zip 对其进行压缩。对于需要随机查询的数据，可以考虑将其保存在 SQLite 数据库中。



## 第9章 Android 中的窗口——Activity

Activity 是 Android 中 4 大应用程序组件之一，只要是程序中涉及界面的显示，就必须要用到 Activity。4.3 节已经介绍了如何调用 Activity 以及在不同 Activity 之间传递数据，然而这些都是在同一个应用程序内部完成的。在实际应用中，往往需要调用其他应用程序中的 Activity，例如，应用程序中如果用到 Wifi 功能，需要进入 Wifi 设置界面来进行相应的设置，除了可以直接通过 API 操作 Wifi 外，最简单的方法就是直接进入系统的 Wifi 设置界面。本章将介绍如何调用一些常用的系统 Activity。除此之外，还介绍了一些 Activity 的高级应用，例如，Activity 之间切换的动画、自定义半透明 Activity 等。

### 9.1 调用其他程序中的 Activity

工程目录：src\ch09\ch09\_invokeotherapp

通过对 4.3 节的学习可以知道，Activity 不能像调用普通 Java 类一样直接调用，而要通过 Intent 对象和 startActivity 或 startActivityForResult 方法调用 Activity，其中 Intent 对象用于指定要调用哪个 Activity。实际上，Intent 对象不仅可以指定应用程序内容的 Activity，也可以指定其他应用程序中的 Activity。本节将介绍如何利用 Intent 对象调用其他应用程序中的 Activity。

#### 9.1.1 直接拨号

直接拨号可以调用系统的拨号功能拨打电话。拨号功能对应的 Action 是 Intent.ACTION\_CALL，使用这个 Action 必须要指定一个 Uri，代码如下：

```
Intent callIntent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:12345678"));
startActivity(callIntent);
```

在执行上面的代码后，系统会拨打指定的电话号，拨号界面效果如图 9.1 所示。

#### 9.1.2 将电话号传入拨号程序

如果不直接拨打指定的电话号，而只想将电话号自动传入 Android 内置的拨号程序，然后再做进一步的处理，需要使用 Intent.ACTION\_DIAL，该 Action 也需要一个“tel:电话号”格式的 Uri，代码如下：

## Android 开发权威指南

```
Intent dialIntent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:87654321"));
startActivity(dialIntent);
```



▲ 图 9.1 直接拨号

运行上面的代码，将显示如图 9.2 所示的“拨号程序”界面。

### 9.1.3 调用拨号程序

如果不将电话号传入拨号程序，而只想启动拨号程序，可以使用如下的代码：

```
Intent touchDialerIntent = new Intent("com.android.phone.action.TOUCH_DIALER");
startActivity(touchDialerIntent);
```

执行上面的代码后，会调用系统的拨号程序，效果如图 9.3 所示。



▲ 图 9.2 将电话号传入拨号程序



▲ 图 9.3 系统的拨号程序

#### 9.1.4 浏览网页

Android SDK 内置的 Web 浏览器也对外提供了 Action，可以通过这个 Action 来传递一个 Web 网址，并通过 Web 浏览器来打开这个 Web 网址，代码如下：

```
Intent webIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://nokiaguy.blogjava.net"));
startActivity(webIntent);
```

#### 9.1.5 向 E-mail 客户端传递 E-mail 地址

E-mail 客户端提供了一个 Action，可以通过这个 Action 将一个 E-mail 地址发送到 E-mail 客户端用于输入 E-mail 的文本框，代码如下：

```
Uri uri = Uri.parse("mailto:xxx@abc.com"); // 指定一个 E-mail 地址，前面必须加mailto
Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
startActivity(intent);
```

运行上面的代码后，会显示如图 9.4 所示的效果。



▲ 图 9.4 E-mail 客户端

##### ■ 注意

在向 E-mail 客户端发送 E-mail 地址时应在手机或模拟器中至少配置一个 E-mail，否则系统无法显示如图 9.4 所示的 E-mail 客户端界面。

#### 9.1.6 发送 E-mail

在很多情况下需要传递的不仅是 E-mail 地址，还包括 E-mail 标题、E-mail 内容等实质性的信息，传递这些信息的 Action 是 Intent.ACTION\_SEND，代码如下：

```
Intent sendEmailIntent = new Intent(Intent.ACTION_SEND);
```

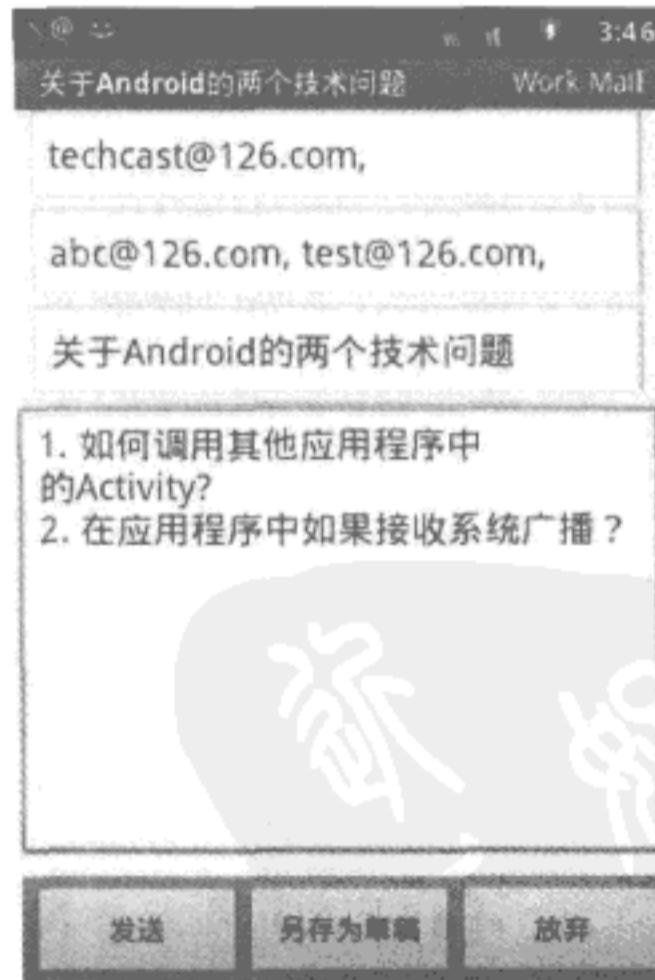
## Android 开发权威指南

```
// 要发送的信息需要通过 putExtra 方法指定
// 指定要发送的目标 E-mail
sendEmailIntent.putExtra(Intent.EXTRA_EMAIL, new String[]{"techcast@126.com"});
// 指定两个抄送的 E-mail 地址
sendEmailIntent.putExtra(Intent.EXTRA_CC, new String[]{"abc@126.com",
"test@126.com"});
// 指定 E-mail 标题
sendEmailIntent.putExtra(Intent.EXTRA_SUBJECT, "关于 Android 的两个技术问题");
// 指定 E-mail 内容
sendEmailIntent.putExtra(Intent.EXTRA_TEXT,
    "1. 如何调用其他应用程序中的 Activity?\n2. 在应用程序中如果接收系统广播? ");
// 指定 E-mail 的内容是纯文本
sendEmailIntent.setType("text/plain");
// 建立一个自定义选择器，并由用户选择使用哪一个客户端发送消息
startActivity(Intent.createChooser(sendEmailIntent, "选择发送消息的客户端"));
```

特别要提一下的是 Intent.createChooser 方法，该方法可以创建一个自定义的选择器。在 Android 系统中支持 Intent.ACTION\_SEND 动作的可能不只有 E-mail 客户端，因此，执行上面代码后一般不会直接进入发送 E-mail 的界面，而是会弹出一个类似如图 9.5 所示选择发送消息客户端的菜单(会根据模拟器或手机中安装软件的不同有所区别，其他的 Action 也可能会发生这种情况)。单击“发送电子邮件”菜单项时，就会进入发送 E-mail 的客户端，如图 9.6 所示，直接单击“发送”按钮即可发送 E-mail。



▲ 图 9.5 选择发送消息的客户端



▲ 图 9.6 传入完整的 E-mail 信息

### 9.1.7 查看联系人

查看联系人的 Action 是“com.android.contacts.action.LIST\_CONTACTS”，下面的代码可以直接显示联系人列表。

```
Intent contactListIntent = new Intent("com.android.contacts.action.LIST_CONTACTS");
```

```
| startActivity(contactListIntent);
```

执行上面的代码，显示效果如图 9.7 所示。



▲图 9.7 联系人列表

### 9.1.8 显示系统设置界面（设置主界面、Wifi 设置界面）

显示系统设置主界面的 Action 是“`android.settings.SETTINGS`”，使用下面的代码来显示系统设置的主界面。

```
| Intent settingsIntent = new Intent("android.settings.SETTINGS");
| startActivity(settingsIntent);
```

显示 Wifi 设置界面的 Action 是“`android.settings.WIFI_SETTINGS`”，使用下面的代码来显示 Wifi 设置界面。

```
| Intent wifiSettingsIntent = new Intent("android.settings.WIFI_SETTINGS");
| startActivity(wifiSettingsIntent);
```

执行上面两段代码，显示的系统设置主界面和 Wifi 设置界面的效果如图 9.8 和图 9.9 所示。



▲图 9.8 系统设置主界面



▲图 9.9 Wifi 设置界面

### 9.1.9 启动处理音频的程序

可以通过 Intent.ACTION\_GET\_CONTENT 动作来选择拥有相同类型的应用，如下面的代码会显示一个包含所有处理音频的程序列表。

```
Intent audioIntent = new Intent(Intent.ACTION_GET_CONTENT);
audioIntent.setType("audio/*");
startActivity(Intent.createChooser(audioIntent, "选择音频程序"));
```

在上面的代码中通过 setType 方法设置了应用程序的类型：“audio/\*”，该类型表示选择系统中所有支持音频功能的应用。在默认的 Android 系统中，如果执行上面代码，将会显示如图 9.10 所示的菜单，单击相应的菜单项会进入指定的应用程序。



▲ 图 9.10 音频程序选择菜单

## 9.2 自定义 Activity Action

工程目录：src\ch09\ch09\_custom\_action、src\ch09\ch09\_client\_action

在 4.1.3 小节曾介绍了在 AndroidManifest.xml 文件中为 Activity 定义 Action，但都是在应用程序内部调用，实际上，这些自定义的 Action 同样可以在其他应用程序中调用。现在我们来看一个实际的例子，其中 ch09\_custom\_action 工程中有一个 MyActivity，该类的定义代码如下：

```
<activity android:name=".MyActivity" android:label="MyActivity">
    <intent-filter>
        <!-- 自定义的 Action -->
        <action android:name="android.intent.action.MyActivity" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

在 ch09\_custom\_action 和 ch09\_client\_action 中都可以使用下面的代码来调用 MyActivity。

```
Intent intent = new Intent("android.intent.action.MyActivity");
startActivity(intent);
```

如果我们定义的 Action 在系统中还有其他的程序也定义了，在使用该 Action 时就会弹出类似如图 9.10 所示的选择菜单。例如，我们按照如下的代码来修改 MyActivity 的定义。

```
<activity android:name=".MyActivity" android:label="MyActivity">
<intent-filter>
    <action android:name="android.intent.action.MyActivity" />
    <action android:name="android.intent.action.GET_CONTENT" />
    <data android:mimeType="audio/*" />
    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>
```

在上面的代码中为 MyActivity 添加了一个 Action 和一个 mimeType，首先运行 ch09\_custom\_action，然后在 ch09\_client\_action 中使用如下的代码来调用 MyActivity。

```
Intent intent = new Intent("android.intent.action.GET_CONTENT");
intent.putExtra("data", "参数值");
intent.setType("audio/*");
startActivity(intent);
```

现在来运行 ch09\_client\_action，并通过上面的代码来访问 MyActivity，会弹出如图 9.11 所示的 3 个菜单项。如果在 MyActivity 类中的 onCreate 方法中使用下面的代码可以获得由 ch09\_client\_action 传递的 data 参数值。

```
String data = getIntent().getStringExtra("data");
if (data != null)
    setTitle(data);
```



▲ 图 9.11 拥有音频处理动作的 MyActivity

要注意的是，如果在定义 MyActivity 时加上了 mimeType，那么在使用 android.intent.action.MyActivity 动作时也要加上 mimeType，否则会抛出无法找到相应 Action 的异常，代码如下：

```
Intent intent = new Intent("android.intent.action.MyActivity");
intent.setType("audio/*");
startActivity(intent);
```

## 9.3 Activity 的高级应用

Android SDK 还支持一些比较高级的应用，例如，可以通过 ActivityGroup 将多个 Activity 放在一个 Activity 中显示、自定义半透明窗口以及 Activity 之间的切换动画等。

### 9.3.1 ActivityGroup

工程目录：src\ch09\ch09\_activitygroup

ActivityGroup 是 Activity 的子类，通过 ActivityGroup 可以将多个 Activity 放到一个 Activity 上显示，这样可以单独实现屏幕的某些部分，然后将它们组合在一起。ActivityGroup 的关键是通过 LocalActivityManager.startActivity 方法获得 Activity 的最顶层窗口（Window 对象），再通过 Window.getDecorView 方法获得窗口的最顶层视图。

现在来准备 3 个 Activity（Activity1、Activity2 和 Activity3），这 3 个 Activity 使用的布局文件分别为 activity1.xml、activity2.xml 和 activity3.xml，代码如下：

#### activity1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Activity1" android:onClick=
        "onClick_Activity1" />
</LinearLayout>
```

#### activity2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Activity2" android:onClick=
        "onClick_Activity2" />
</LinearLayout>
```

#### activity3.xml

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

```

    android:layout_height="fill_parent">
    <Button android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Activity3" android:onClick=
        "onClick_Activity3" />
</LinearLayout>

```

从上面的代码可以看出，这 3 个布局文件中都包含了一个按钮，如果要将这 3 个 Activity 都放在一个 Activity 上显示，就会出现 3 个按钮。下面来看一下主界面的布局文件。

### main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:id="@+id/header" android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="120dp">
    </LinearLayout>
    <LinearLayout android:id="@+id/body" android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:layout_weight="1">
    </LinearLayout>
    <LinearLayout android:id="@+id/footer" android:orientation="vertical"
        android:layout_width="fill_parent" android:layout_height="100dp"
        android:layout_gravity="bottom">
        <Button android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:onClick="onClick_Footer_Button"
            android:text="Footer_Button" />
    </LinearLayout>
</LinearLayout>

```

从 main.xml 文件中可以看出，在最顶层的<LinearLayout>标签中包含了 3 个<LinearLayout>标签，在后面的程序中会将 Activity1、Activity2 和 Activity3 分别添加到这 3 个<LinearLayout>标签中，在最后一个<LinearLayout>标签中有一个<Button>标签，因此，屏幕最后应为 4 个按钮。添加 Activity 的主程序代码如下：

```

package mobile.android.ch09.activitygroup;

import android.app.ActivityGroup;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.Toast;

public class Main extends ActivityGroup
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout header = (LinearLayout) findViewById(R.id.header);
        LinearLayout body = (LinearLayout) findViewById(R.id.body);

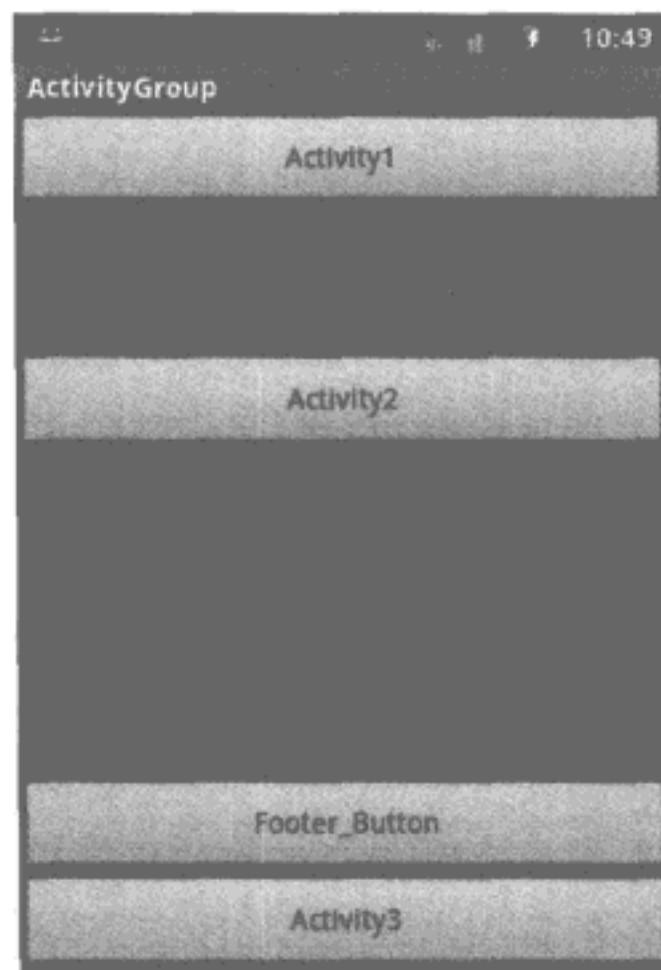
```

```

LinearLayout footer = (LinearLayout) findViewById(R.id.footer);
// 获得 Activity1 中的顶层视图
View activity1 = getLocalActivityManager().startActivity("activity1",
    new Intent(Main.this, Activity1.class)).getDecorView();
// 获得 Activity2 中的顶层视图
View activity2 = getLocalActivityManager().startActivity("activity2",
    new Intent(Main.this, Activity2.class)).getDecorView();
// 获得 Activity3 中的顶层视图
View activity3 = getLocalActivityManager().startActivity("activity3",
    new Intent(Main.this, Activity3.class)).getDecorView();
// 将 Activity1 添加到第一个<LinearLayout>标签中
header.addView(activity1);
// 将 Activity2 添加到第二个<LinearLayout>标签中
body.addView(activity2);
// 将 Activity3 添加到第三个<LinearLayout>标签中
footer.addView(activity3);
}
public void onClick_Footer_Button(View view)
{
    Toast.makeText(this, "Footer_Button", Toast.LENGTH_LONG).show();
}
}

```

运行上面的代码，显示效果如图 9.12 所示。



▲ 图 9.12 同时显示多个 Activity

### 9.3.2 自定义半透明窗口

工程目录: src\ch09\ch09\_translucence\_window

Android 应用程序默认的窗口颜色是黑色的，而且不透明。为了使程序的界面更绚丽，可以更换 Activity 的背景。如果背景图是半透明的，那么就可以很容易实现半透明的 Activity。

在实现半透明 Activity 之前，需要用 Photoshop 或其他图像处理软件做一个半透明的 png 图(透

明度可根据需要设置)。由于作为背景时需要对 png 图像进行拉伸,因此,需要将 png 图转换成 9.png 格式的图像(关于 9.png 格式图像的细节详见 5.2.5 小节的内容)。通过 9.png 图像可以确定图像的哪些部分可以拉伸,哪些部分不可以拉伸。然后在 res\values 目录中创建一个 xml 主题文件(为了方便,本例就直接将主题放在 strings.xml 文件中了),并编写如下的主题。

```
<style name="MyTheme" parent="@android:style/Theme.Dialog">
    <item name="android:windowBackground">@drawable/msg_background</item>
</style>
```

上面的代码覆盖了系统的 Theme.Dialog 主题(该主题可以将 Activity 变成一个类似对话框的窗口),并覆盖了 android:windowBackground 属性,将该属性值设为刚才建立的半透明图像的 ID。现在修改 AndroidManifest.xml 文件中的定义,代码如下:

```
<activity android:name=".Main" android:label="@string/app_name" android:theme="@style/MyTheme">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

在<activity>标签中通过 android:theme 属性指定了刚才建立的主题,最后可根据需要设置布局文件中顶层标签(本例中是<LinearLayout>)的宽度和高度。现在来运行程序,会看到如图 9.13 所示的效果。



▲ 图 9.13 半透明窗口

### 9.3.3 Activity 之间切换的动画效果

工程目录: src\ch09\ch09\_activity\_animation

默认情况下两个 Activity 之间的切换是直接显示另一个 Activity,为了使程序看起来更加有趣,

可以在 Activity 之间切换时加上动画效果。

如果在 `startActivity` 或 `finish` 后调用 `Activity.overridePendingTransition` 方法，并指定显示和关闭 Activity 的动画效果，就会以动画方式显示和关闭 Activity。`overridePendingTransition` 方法的定义如下：

```
public void overridePendingTransition(int enterAnim, int exitAnim);
```

该方法有两个参数，都是动画资源 ID，其中 `enterAnim` 表示显示 Activity 时的动画，`exitAnim` 表示关闭 Activity 时的动画。这里的动画资源是保存在 `res\anim` 目录中的 xml 动画文件。关于动画文件的内容将在后面详细讲解，在这里只要知道这些动画的原理和 Flash 类似，都称为补间动画。也就是说，只要定义起始和结束的状态以及动画轨迹，系统就会自动生成动画的中间状态。在本例中我们使用了 2 个动画文件：`fade.xml` 和 `hyperspace.xml`，分别表示淡入淡出和立体飞出的效果。

下面的代码分别以淡入淡出和立体飞出的效果显示一个 Activity。

### 淡入淡出效果

```
Intent intent = new Intent(this, AnimationActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.fade, R.anim.fade);
```

### 立体飞出效果

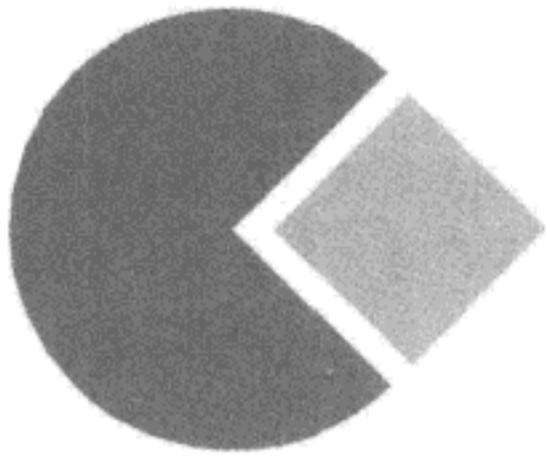
```
Intent intent = new Intent(this, AnimationActivity.class);
startActivity(intent);
overridePendingTransition(R.anim.hyperspace, R.anim.hyperspace);
```

如果 `overridePendingTransition` 在 `finish` 之后调用，系统会以动画方式关闭 Activity。下面的代码以淡入淡出效果来关闭 Activity。

```
Intent intent = new Intent(this, AnimationActivity.class);
finish();
overridePendingTransition(R.anim.fade, R.anim.fade);
```

## 9.4 小结

本章主要介绍了调用系统的 Activity 以及自定义 Activity Action。通过 Activity Action 技术可以将自己的应用程序中某些 UI 功能对外开放，从而实现应用程序之间在 UI 上的共享。除此之外，还介绍了 Activity 的一些高级技术，包括在同一个 Activity 中显示多个 Activity 的 ActivityGroup、半透明窗口以及 Activity 之间的切换动画。利用这些技术可以使 Android 应用程序得到不少加分。



## 第 10 章 全局事件——广播（Broadcast）

广播是 Android SDK 的四大组件中唯一需要被动接收数据的组件。也就是说，对于 Activity、ContentProvider 和 Service，都可以主动去调用，并获得返回数据，而负责接收 Broadcast 数据的接收器却永远不会知道什么时候可以接收到广播。从这种表现形式上看，很像面向对象中的事件（Event）。对于事件（例如，onClick、onKeydown）来说，从来不会预知用户什么时候触发它们，只能默默地、静静地等待不可预知的事件发生。因此，广播也可以被称为全局事件，或系统事件。

### 10.1 什么是广播

事件（Event）是面向对象中的核心概念。当用户操作窗口中的控件时，就会触发相应的事件。例如，用户按下按钮后，会触发单击（onClick）事件。当然，这种事件都是在应用程序内部发生的。对于应用程序内部的事件很容易实现，也很容易理解，但对于操作系统的事件就需要做更复杂的处理。

不管读者是否开发过 Windows 桌面程序，都会接触到一些可以检测键盘事件的程序。例如，有些程序无论在任何状态，都可以通过按下键盘的某个键或某些键的组合来运行程序的某些功能。毫无疑问，这肯定是用户触发了键盘事件，但问题是，这些程序是如何来检测并不属于自己程序的键盘事件的呢？

在 Windows 中有很多机制可以与应用程序进行交互。然而，最重要的，也是开发 Windows 系统程序必须掌握的技术就是钩子（Hook）。将这种技术比喻成钩子很形象，不管我们是在晾很多衣服时，还是在挂其他什么东西，总会栓一条绳子，然后将很多带钩子的挂件（可能是衣服架或其他类似的东西）按顺序挂在绳子上。当然，说了这么多，可能很多读者认为广播和晾衣服有什么关系，在这里说的意思是连动原则，以晾衣服为例只是让大家有一个感性的认识。

我们可以想象，一条水平栓着的绳子上面挂着很多带钩的衣服架。如果绳子的一端抖动，就会产生波浪效应（这里指的是理论上的），离抖动源最近的衣服架会最先感到震动，然后是离抖动源更远的衣服架，直到最后一个衣服架感知到了震动。

言归正传。我们可以将 Windows 的某个系统事件比喻成一条晾衣服的绳子，而每一个要接收这个系统事件的程序比喻成衣服，那么这些衣服要想能感知到绳子的震动（事件被触发），就需要使用衣服架（钩子）将其挂到绳子上。而且根据所挂的位置不同，感知震动的顺序也会不同（从技术角度被称为优先级）。当然，如果不想要感知震动（捕获系统事件），把晾衣架摘下来就行了，很简单。

虽然 Android 和 Windows 有着本质的不同，但天才的 idea 总是有着惊人的相似点。Android 的广播技术也采用了类似 Windows Hook 的技术（为了轻松下，也可以叫晾衣服技术）。当 Android 系统发生某些事件时（例如，收到短信、来电、电量报警）就会向整个 Android 系统发出消息（消息本身相当于晾衣服绳，而发送消息相当于抖动晾衣服绳），这种消息被称为广播。如果只有消息是毫无意义的，就像说出的话没有人听，自己也就觉得无趣了。与消息（广播）对应的叫接收器，这个可以看作是衣服架。现在要做的就是将衣服架挂到绳子上，也就是注册接收器。因此，一个广播的完整处理过程和晾衣服非常像，首先要注册接收器（将衣服架挂到绳子上），然后发送广播（抖动晾衣服绳）。如果衣服不从绳子上拿下来，只要绳子一端抖动，所有的衣服就会按离抖动源的远近先后感觉到震动。这也说明只要不注销（将衣服从绳子上拿下来），就永远可以接收到系统发送的消息（绳子的抖动）。

本节给出了一个生活中的例子来解释什么是广播技术，从这个例子可以看出，广播主要涉及接收和优先级两个方面。只要将广播接收器注册到某个消息上，不管有多少接收器，这些接收器都会按不同的优先级接收到这个广播，然后再做进一步处理。

## 10.2 接收系统广播

广播的最大用途是接收系统发出的消息。例如，当收到短信时，可以截获发送短信的电话号码和短信内容，当来电时可以截获来电的各种状态（响铃、接听、挂断）。

### 10.2.1 短信拦截

工程目录：src\ch10\ch10\_sms\_receiver

首先来编写一个拦截短信广播的接收器类，该类必须从 android.content.BroadcastReceiver 类继承，代码如下：

```
package mobile.android.ch10.sms.receiver

import java.util.Set;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.SmsMessage;
import android.util.Log;
import android.widget.Toast;

public class ShortMessageReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        if (bundle != null)
        {
            Set<String> keys = bundle.keySet();
```

```
// 查看收到的广播包含哪些数据
for (String key : keys)
{
    Log.d("key", key);
}
// 获得收到的短信数据
Object[] objArray = (Object[]) bundle.get("pdus");
// 定义封装短信内容的 SmsMessage 对象数组
SmsMessage[] messages = new SmsMessage[objArray.length];
// 循环处理收到的所有短信
for (int i = 0; i < objArray.length; i++)
{
    // 将每条短信数据转换成 SmsMessage 对象
    messages[i] = SmsMessage.createFromPdu((byte[]) objArray[i]);
    // 获得发送短信的电话号码和短信内容
    String s = "手机号: " + messages[i].getOriginatingAddress() + "\n";
    s += "短信内容: " + messages[i].getDisplayMessageBody();
    // 显示发送短信的电话号码和短信内容
    Toast.makeText(context, s, Toast.LENGTH_LONG).show();
}
}
```

编写上面代码时应注意如下几点。

- 如果不知道接收到的广播包含哪些数据，可以从 `Bundle.keySet()`方法中获得这些数据的 key，将其输出到 LogCat 视图中以便查看详细情况。
    - 由于接收到的短信内容是以字节数组形式保存的，为了便于使用这些数据，需要使用 `SmsMessage.createFromPdu` 方法将这些字节数组形式的数据转换成 `SmsMessage` 对象。
    - 在最新的 Android SDK 版本中提供了新的 `SmsMessage` 类，该类在 `android.telephony` 包中。老版本的 Android SDK 使用 `android.telephony.gsm.SmsMessage` 类，虽然在新版的 Android SDK 中是不同的 package，但同名的类都存在。笔者并不建议使用 `android.telephony.gsm` 包中的 `SmsMessage` 类，因为在未来的 Android SDK 版本中该类可能会被去除。
    - 由于接收器可能接收到多条短信，因此，通过“pdus”返回了一个短信数组（`byte[]`）。广播接收器需要在 `AndroidManifest.xml` 文件中注册，代码如下：

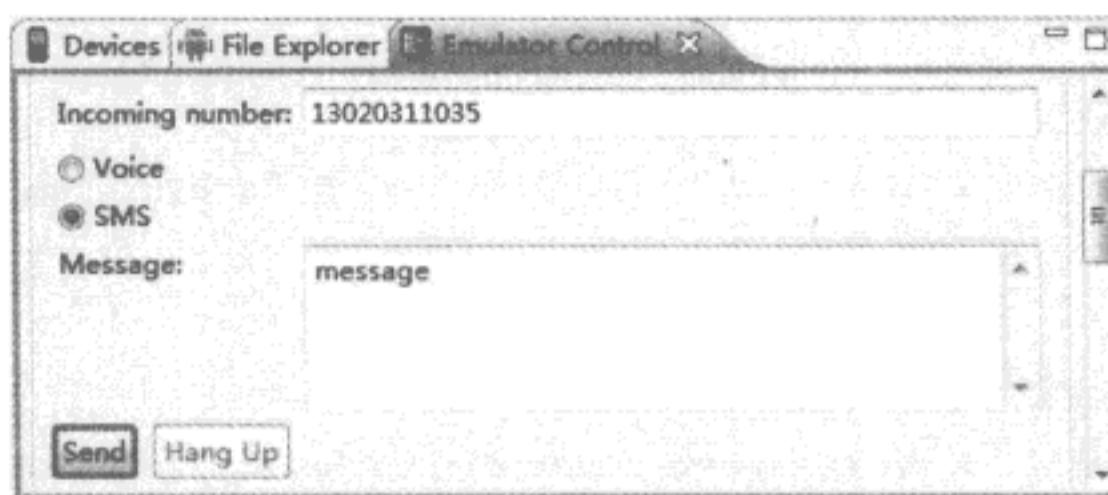
```
<receiver android:name=". ShortMessageReceiver"
    android:enabled="true">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

其中 android.provider.Telephony.SMS\_RECEIVED 是接收到短信的广播动作，必须指定该动作，**ShortMessageReceiver** 才能接收到系统的短信广播。

由于 Android 的安全机制, 必须在 `AndroidManifest.xml` 文件中使用如下的代码打开接收短信的权限, 程序才能够正常接收到短信广播。

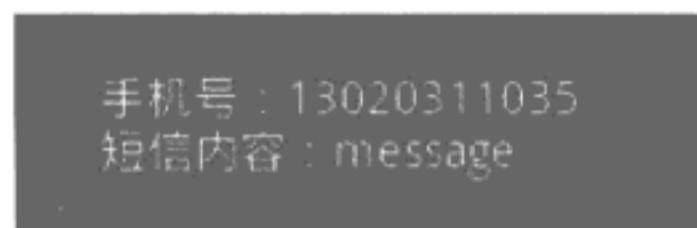
```
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

运行本例，并在“Emulator Control”视图中模拟发送短信，发送短信的电话号和短信内容如图 10.1 所示，然后单击“Send”按钮发送短信。



▲ 图 10.1 模拟发送短信

在发送短信后，程序就会接收到广播，并显示如图 10.2 所示的信息。



▲ 图 10.2 显示发送短信的电话号和短信内容

### ■ 注意

即使注册广播接收器的程序关闭，接收器仍然会接收到广播，除非从模拟器或手机中卸载程序或注销接收器，否则无法阻止接收器接收广播。

## 10.2.2 用代码注册广播接收器

工程目录: src\ch10\ch10\_register\_receiver

如果在 `AndroidManifest.xml` 文件中配置广播接收器，程序在安装后就会自动注册广播接收器。如果想在适当的时候注册广播接收器，在使用完将其注销，就需要直接使用 Java 代码来完成这个功能。

注册广播接收器的方法是 `registerReceiver`，注销广播接收器的方法是 `unregisterReceiver`，这两个方法的定义如下：

```
public Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
public void unregisterReceiver(BroadcastReceiver receiver)
```

其中 `receiver` 参数表示广播接收器对象，`filter` 参数相当于设置`<intent-filter>`标签中的内容。下面的代码重新注册了上一小节编写的短信广播接收器。

```
package mobile.android.ch10.register.receiver;

import android.app.Activity;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
```

```

import android.widget.Toast;

public class Main extends Activity
{
    private ShortMessageReceiver shortMessageReceiver;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建广播接收器的对象
        shortMessageReceiver = new ShortMessageReceiver();
    }
    public void onClick_Register_Broadcast(View view)
    {
        // 注册接收短信广播的接收器
        registerReceiver(shortMessageReceiver,
                         new IntentFilter("android.provider.Telephony.SMS_RECEIVED"));
        Toast.makeText(this, "注册成功", Toast.LENGTH_LONG).show();
    }
    public void onClick_Unregister_Broadcast(View view)
    {
        // 注销接收短信广播的接收器
        unregisterReceiver(shortMessageReceiver);
        Toast.makeText(this, "注销成功", Toast.LENGTH_LONG).show();
    }
}

```

### 10.2.3 广播接收器的优先级

**工程目录: src\ch10\ch10\_receiver\_priority**

在 Android 系统中接收某种广播的接收器可能不只注册了一个，那么如果注册了多个接收同一种广播的接收器，就存在一个调用顺序的问题，也称为调用优先级。

通过<intent-filter>标签的 android:priority 属性可以设置接收器的调用优先级。该属性值是一个整数，数值越大，优先级越高。

现在我们来做个实验，将 10.2.1 节的 ShortMessageReceiver 类再复制一份，分别取名为 ShortMessageReceiver1 和 ShortMessageReceiver2，这两个类都会接收短信广播，功能是完全一样的，只是需要在显示短信内容时加个标记以区分是由哪个接收器显示的信息。下面是 AndroidManifest.xml 文件的完整代码，其中配置了这两个接收器。

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobile.android.ch10.receiver.priority"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Main"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

```

```

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<receiver android:name=".ShortMessageReceiver1" android:enabled="true" >
    <intent-filter android:priority="500">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
<receiver android:name=".ShortMessageReceiver2" android:enabled="true" >
    <intent-filter android:priority="1000">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
</application>
<uses-sdk android:minSdkVersion="9" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
</manifest>

```

在上面的代码中将 ShortMessageReceiver1 的优先级设为 500，而 ShortMessageReceiver2 的优先级设为 1000，这时在收到短信时，ShortMessageReceiver2 会先被调用，然后会调用 ShortMessageReceiver1。当然，如果不设置优先级，对于同一个应用程序中的广播接收器会按照在 AndroidManifest.xml 文件中定义的顺序调用。

#### ■ 注意

广播接收器的优先级只对同步处理方式起作用，如果在接收器中使用了异步处理方式，则调用的顺序除了与优先级有关，还与 Android 系统的线程调用有关。

### 多学一招：在手机上如何测试短信

在模拟器上测试短信很容易，也没什么成本，而手机就无法像模拟器一样模拟发送短信了。当然，可以用其他的手机发短信，但这需要一定的成本，对于个人开发者可能并不值得这么做。为了尽可能节省成本，可以利用一些免费发送短信的软件（如飞信），或有一些网站通过手机注册可以用某个号码给手机发送注册码或其他信息（如中国移动网上营业厅可以用 10086 发送随机登录密码），也可以利用这些网站来测试程序的短信功能。

#### 10.2.4 来去电拦截

**工程目录：**src\ch10\ch10\_call\_in\_out\_receiver

监听电话状态以用于来电拦截，来电（监听电话状态）和去电的广播动作如下：

来电： android.intent.action.PHONE\_STATE  
去电： android.intent.action.NEW\_OUTGOING\_CALL

来电可分解为 3 个状态：未接电话时的响铃、接听电话和挂断电话（可能是对方挂断的，也可能是自己挂断的）。监听这 3 个状态的接收器的代码如下：

```

package mobile.android.ch10.call.in.out.receiver;
import java.lang.reflect.Field;

```

```
import java.lang.reflect.Method;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.TelephonyManager;
import android.util.Log;
import android.view.Gravity;
import android.widget.Toast;

public class InCallReceiver extends BroadcastReceiver
{
    private static Object obj;
    // 显示永不关闭的Toast信息框
    public static void showToast(Context context, String msg)
    {
        Toast toast = Toast.makeText(context, msg, Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0, 0);
        try
        {
            Field field = toast.getClass().getDeclaredField("mTN");
            field.setAccessible(true);
            obj = field.get(toast);
            Method method = obj.getClass().getDeclaredMethod("show", null);
            method.invoke(obj, null);
        }
        catch (Exception e)
        {
        }
    }
    // 关闭Toast信息框
    public static void closeToast()
    {
        if (obj != null)
        {
            try
            {
                Method method = obj.getClass().getDeclaredMethod("hide", null);
                method.invoke(obj, null);
            }
            catch (Exception e)
            {
            }
        }
    }
    @Override
    public void onReceive(final Context context, final Intent intent)
    {
        // 获得电话管理服务，以便获得电话的状态
        TelephonyManager tm = (TelephonyManager) context
            .getSystemService(Service.TELEPHONY_SERVICE);
        switch (tm.getCallState())
        {
            case TelephonyManager.CALL_STATE_RINGING:           // 响铃

```

```
// 获得来电的电话号
String incomingNumber = intent.getStringExtra("incoming_number");
showToast(context, incomingNumber);
break;
case TelephonyManager.CALL_STATE_OFFHOOK: // 接听电话
    Log.d("call_state", "offhook");
    break;
case TelephonyManager.CALL_STATE_IDLE: // 挂断电话
    closeToast();
}
}
```

上面代码中的 `showToast` 方法显示了一个永不关闭的 `Toast` 信息框，`closeToast` 方法关闭这个 `Toast` 信息框。关于永不关闭的 `Toast` 信息框请参阅 7.3.2 小节的内容。

在接收广播时，系统会为每一次接收广播单独创建一个广播接收器对象，即使是同一个广播的多次接收。因此，当电话处于不同状态时，实际上是在不同的 `InCallReceiver` 对象中发生的，所以要使用静态变量来保存 `Toast` 对象，否则 `closeToast` 将无法获得在上一个状态创建的 `Toast` 对象，也就无法关闭 `Toast` 信息框了。

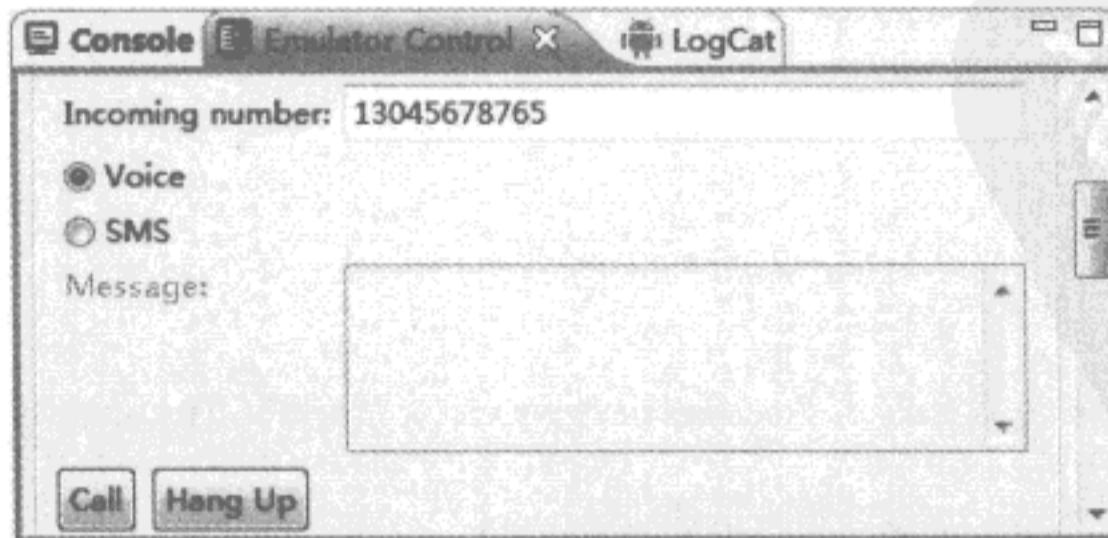
配置 InCallReceiver 的代码如下：

```
<receiver android:name=".InCallReceiver" android:enabled="true">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
    </intent-filter>
</receiver>
```

监听来电状态需要使用下面的代码设置权限。

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

运行程序，在“Emulator Control”视图中模拟来电，如图 10.3 所示。



▲ 图 10.3 模拟来电

在模拟器接到来电后，除了会显示正常的来电界面外，在屏幕上方还会通过 Toast 信息框显示来电的电话号，如图 10.4 所示。除非挂断电话，这个 Toast 信息框是不会自己消失的。



▲ 图 10.4 来电显示电话号

接收去电广播的接收器代码如下：

```
package mobile.android.ch10.call.in.out.receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class OutCallReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 获得去电的电话号
        String outcomingNumber = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        InCallReceiver.showToast(context, outcomingNumber);
    }
}
```

OutCallReceiver 类的配置代码如下：

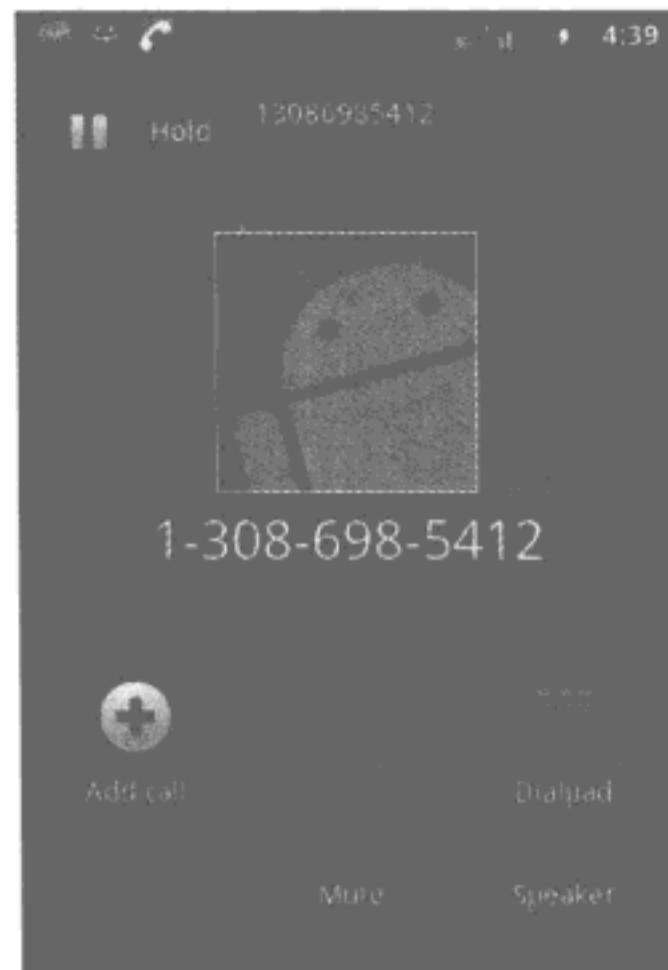
```
<receiver android:name=".OutCallReceiver" android:enabled="true">
    <intent-filter>
        <action android:name="android.intent.action.NEW_OUTGOING_CALL" />
    </intent-filter>
</receiver>
```

监听去电除了要授予 android.permission.READ\_PHONE\_STATE 权限外，还要使用下面的代码授予另一个权限。

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

启动程序，然后使用模拟器右侧的接听按钮启动电话拨号界面，输入一个电话号，然后拨打电

话，除了会显示拨打电话的界面外，在屏幕上方还会通过 Toast 信息框显示拨打的电话号，如图 10.5 所示。



▲ 图 10.5 去电显示电话号

### 10.2.5 截获屏幕休眠与唤醒

工程目录: src\ch10\ch10\_screen\_on\_off\_receiver

按手机上的挂断按钮后，手机会进入休眠状态（屏幕变黑）。当再次按下手机上的任意按钮后，屏幕会唤醒（屏幕变亮）。这两个状态可以通过如下两个动作拦截。

休眠状态: Intent.ACTION\_SCREEN\_ON

唤醒状态: Intent.ACTION\_SCREEN\_OFF

下面来编写一个用于接收修改和唤醒广播的接收器。

```
package mobile.android.ch10.screen.on.off.receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.util.Log;

public class ScreenOnOffReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 接收屏幕唤醒状态的广播
        if (Intent.ACTION_SCREEN_ON.equals(intent.getAction()))
        {
            Log.d("screen", "ok");
        }
    }
}
```

```

    // 接收屏幕休眠状态的广播
    else if (Intent.ACTION_SCREEN_OFF.equals(intent.getAction()))
    {
        Log.d("screen", "off");
    }
}
}

```

ScreenOnOffReceiver 类同时接收了两个广播，并使用 Intent.getAction 方法来判断当前接收的是哪个广播。在 Android SDK 中并不是每一个广播都需要设置权限，例如，本例接收的这两个广播就不需要设置任何的访问权限。下面使用代码来注册广播接收器。

```

package mobile.android.ch10.screen.on.off.receiver;

import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

public class Main extends Activity
{
    private ScreenOnOffReceiver screenOnOffReceiver;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        screenOnOffReceiver = new ScreenOnOffReceiver();
        IntentFilter intentFilter = new IntentFilter();
        // 设置屏幕唤醒广播的动作
        intentFilter.addAction(Intent.ACTION_SCREEN_ON);
        // 设置屏幕休眠广播的动作
        intentFilter.addAction(Intent.ACTION_SCREEN_OFF);
        registerReceiver(screenOnOffReceiver, intentFilter);
    }
}

```

运行程序后，使模拟器或手机屏幕唤醒和休眠，会在“LogCat”视图中输出不同的信息。

#### 注意

屏幕唤醒和休眠广播只能在代码中注册，如果在 AndroidManifest.xml 文件中注册将不起作用。

### 10.2.6 开机自动运行

工程目录: src\ch10\ch10\_boot\_complete\_receiver

开机自启动程序实际上也是接收了一个手机启动完成的广播，然后在广播接收器中做进一步地处理，例如，启动一个 Activity。

启动完成的广播动作如下：

android.intent.action.BOOT\_COMPLETED

本节的例子要实现一个开机自动运行 Activity 的程序，实现步骤如下。

(1) 编写一个 StartupReceiver 类，该类是 BroadcastReceiver 的子类，用于接收系统广播，代码如下：

```
package mobile.android.ch10.boot.complete.receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class StartupReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Intent mainIntent = new Intent(context, Main.class);
        // 在广播接收器中显示 Activity，必须要设置 FLAG_ACTIVITY_NEW_TASK 标志
        mainIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(mainIntent);
    }
}
```

(2) 在 AndroidManifest.xml 文件中配置 StartupReceiver 类，代码如下：

```
<receiver android:name="StartupReceiver">
    <intent-filter>
        <!-- 指定要接收的 Broadcast Action -->
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

现在运行这个应用程序，运行完毕后，即可关闭程序，然后重启模拟器，会发现模拟器在启动后总是会先运行本例的程序，运行效果如图 10.6 所示。



▲ 图 10.6 开机自启动

### 10.2.7 显示手机电池的当前电量

**工程目录：src\ch10\ch10\_battery\_changed\_receiver**

如果在手机上进行某些耗电的工作，提前查一下手机电池当前的电量是一个好主意，如发现手机电池的电量不足，可以提前充电，这样可以避免手机在使用过程中没电的尴尬。

实际上，查看电池的电量也需要接收一个系统广播，只是本例中实现的接收器不是在 AndroidManifest.xml 文件中使用<receiver>标签定义的，而是在程序中通过 registerReceiver 方法进行注册的。本例中创建了一个 BroadcastReceiver 类型的 batteryChangedReceiver 变量，用于接收手



机电量变化的 Broadcast Action。

```
package mobile.android.ch10.batter.changed.receiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.widget.TextView;

public class Main extends Activity
{
    private TextView tvBatteryChanged;
    private BroadcastReceiver batteryChangedReceiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
            // 判断接收到的是否为电量变化的 Broadcast Action
            if (Intent.ACTION_BATTERY_CHANGED.equals(intent.getAction()))
            {
                // level 表示当前电量的值
                int level = intent.getIntExtra("level", 0);
                // scale 表示电量的总刻度
                int scale = intent.getIntExtra("scale", 100);
                // 将当前电量换算成百分比的形式
                tvBatteryChanged.setText("电池用量: " + (level * 100 / scale) + "%");
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvBatteryChanged = (TextView) findViewById(R.id.tvBatteryChanged);
        // 注册 Receiver
        registerReceiver(batteryChangedReceiver, new IntentFilter(
            Intent.ACTION_BATTERY_CHANGED));
    }
}
```

运行本例后，显示效果如图 10.7 所示。



▲ 图 10.7 显示电池的剩余电量

## 10.3 发送广播

工程目录: src\ch10\ch10\_send\_broadcast, src\ch10\ch10\_custom\_receiver

我们可以通过 sendBroadcast 方法发送广播。sendBroadcast 方法的定义如下:

```
// 通过 intent 参数可指定一个广播动作
public void sendBroadcast(Intent intent)
```

下面的代码发送了一个广播，并添加了广播数据和 category。

```
// 指定广播动作
Intent broadcastIntent = new Intent("mobile.android.ch10.MYBROADCAST");
// 添加 category
broadcastIntent.addCategory("mobile.android.ch10.mycategory");
// 设置广播数据
broadcastIntent.putExtra("name", "broadcast_data");
// 发送广播
sendBroadcast(broadcastIntent);
Toast.makeText(this, "广播发送成功.", Toast.LENGTH_LONG).show();
```

在 ch10\_custom\_receiver 工程中需要编写一个用于接收自定义广播的接收器，代码如下:

```
package mobile.android.ch10.custom_receiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class CustomReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        if ("mobile.android.ch10.MYBROADCAST".equals(intent.getAction()))
        {
            // 获得广播数据
            String name = intent.getStringExtra("name");
            Toast.makeText(context, name, Toast.LENGTH_LONG).show();
        }
    }
}
```

CustomReceiver 类的配置代码如下:

```
<receiver android:name=".CustomReceiver">
    <intent-filter>
        <action android:name="mobile.android.ch10.MYBROADCAST" />
        <category android:name="mobile.android.ch10.mycategory" />
    </intent-filter>
</receiver>
```

先运行 ch10\_custom\_receiver 程序，然后可以关闭该程序，最后运行 ch10\_send\_broadcast 程序，并单击“发送广播”按钮，首先会显示“广播发送成功”信息，当 ch10\_custom\_receiver 接收到广播后，会显示“broadcast\_data”信息。

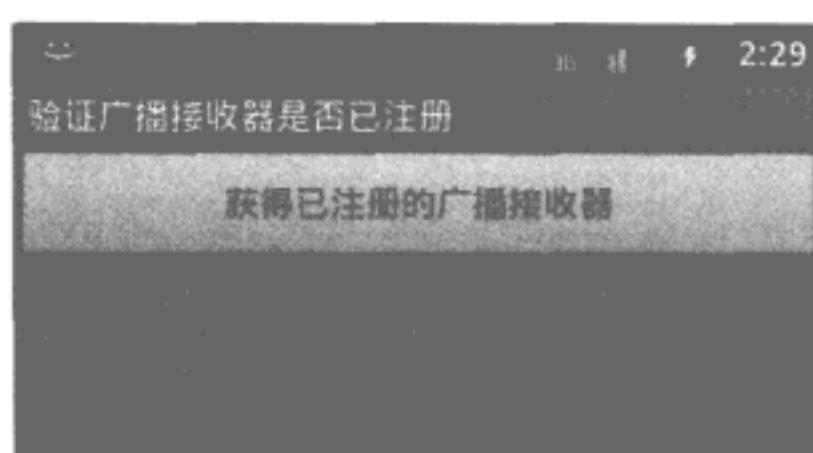
## 10.4 验证广播接收器是否注册

工程目录：src\ch10\ch10\_validate\_receiver

不仅可以通过 Android SDK 的 API 发送、接收广播，还可以查询系统中已经注册了哪些广播。可以通过这种方法判断某个广播是已注册了至少一个接收器。例如，可以通过下面的代码判断 10.3 节编写的广播接收器（CustomReceiver）是否已注册。

```
// 获得 PackageManager 对象
PackageManager packageManager = getPackageManager();
// 指定要查询广播的动作
Intent intent = new Intent("mobile.android.ch10.MYBROADCAST");
// 返回已查询到的广播接收器集合，如果没有符合条件的广播，List 对象的长度为 0
List<ResolveInfo> resolveInfos = packageManager.queryBroadcastReceivers(intent,
    PackageManager.GET_INTENT_FILTERS);
// 显示查询到的广播接收器的数量
Toast.makeText(this, "已发现" + resolveInfos.size() + "个接收短信广播的接收器。",
    Toast.LENGTH_LONG).show();
TextView textView = (TextView) findViewById(R.id.textview);
String s = "";
// 显示查询到的广播接收器的信息
for (ResolveInfo resolveInfo : resolveInfos)
{
    s += String.valueOf(resolveInfo.toString()) + "\n\n";
}
textView.setText(s);
```

运行上面的代码，如果 ch10\_custom\_receiver 已经运行过，会显示如图 10.8 所示的信息。如果查询结果为 0，则说明要查询的广播并没有注册任何接收器。



▲ 图 10.8 查询已注册的广播接收器

### 扩展学习：查询 Activity Action 是否存在

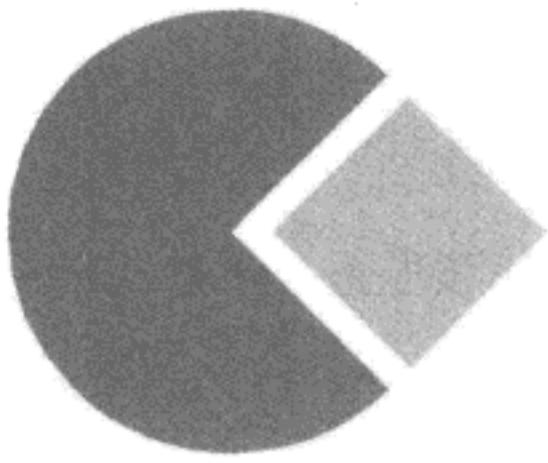
PackageManager 对象不仅能查询广播接收器，也可以查询 Activity 以及下一章要讨论的 Content Provider。例如，通过下面的代码可以查询 Activity 动作 com.android.phone.action.TOUCH\_DIALER

是否存在，这在设计插件系统时非常重要。如果程序的某个菜单要调用其他程序提供的 Activity，可以先查询一下这个 Activity 对应的 Action 是否存在，如果不存在，可以直接将菜单隐藏或设为不可选状态。

```
PackageManager packageManager = getPackageManager();
// 指定要查询的Activity Action
Intent intent = new Intent("com.android.phone.action.TOUCH_DIALER");
List<ResolveInfo> resolveInfos = packageManager
    .queryIntentActivities(intent,
        PackageManager.GET_INTENT_FILTERS);
Toast.makeText(this, "已发现" + resolveInfos.size() + "个Activity.",
    Toast.LENGTH_LONG).show();
TextView textView = (TextView) findViewById(R.id.textview);
String s = "";
for (ResolveInfo resolveInfo : resolveInfos)
{
    s += String.valueOf(resolveInfo.toString()) + "\n\n";
}
textView.setText(s);
```

## 10.5 小结

本章主要介绍了 Android 中的四大组件之一的广播。广播相当于系统事件或全局事件，当 Android 系统中任何程序有动作时，如果想通知其他的程序，采用广播的方式进行传播是非常有效的，因为广播从理论上可以将一个动作传播给任意多个程序（当然，广播接收器的数量会受到系统的限制）。除了接收广播，还可以利用 Android SDK 的 API 发送广播以及查询系统中已注册的广播接收器。



# 第 11 章 跨应用数据源—— Content Provider

内容提供者（Content Provider）提供了对内容和外部数据进行增、删、改、查的功能。实际上，Content Provider 就相当于跨应用的数据操作，也可以看作是操作数据库的代理。那么 Android SDK 为什么要提供一个中间层来操作数据库呢？其中的原因有很多，本章将逐一揭示其中的奥秘。

## 11.1 Content Provider 的作用

Content Provider 相当于数据的接口，通过 Content Provider 可以将程序内部使用的数据向其他程序公开，这样其他的程序就可以通过 Context.getContentResolver 方法获得 ContentResolver 对象，并使用 ContentResolver.insert、ContentResolver.delete、ContentResolver.update、ContentResolver.query 四个方法对程序内部的数据进行增、删、改、查操作。

那么为什么 Android SDK 要提供一个数据接口来访问数据呢？原因可能有很多种，但提供 Content Provider 至少起到如下的作用。

- 由于很多程序内部的数据库文件都保存在私有的目录中，其他的程序使用常规的方法无法访问这些数据库，因此，通过 Content Provider 可以使外部程序访问私有数据库。这有些类似 Java 类中的 private 变量和 setter、getter 方法。由于 Java 中并没有与属性相关的语法，因此，通过 setter 和 getter 方法可以对 Java 类的 private 变量进行访问，private 变量就相当于私有数据，而 setter 和 getter 方法相当于 Content Provider。

- 由于程序内部使用的数据库结构可能很复杂，例如，数据通过连接若干个表和视图连接而获得。对于不了解数据结构的开发人员，即使可以访问私有数据库，也无从下手，因此，通过 Content Provider 可以向开发人员提供一个更人性化的查询结果集。至于连接表和视图以及为字段重命名、组合字段等复杂操作都隐藏在 query 方法中。

- 我们知道，由于安全原因，Android SDK 提供了一些安全机制，也就是通过<uses-permission> 标签设置的权限。虽然通过 Content Provider 可以获得任何的私有数据（只要程序为私有数据提供了 Content Provider），但出于安全考虑，有一些数据需要设置权限才能使用 Content Provider 正常操作数据，这种访问控制就可以通过 Content Provider 机制完成。这种方式有些类似于 JVM，作为应用程序和操作系统之间的一层，可以有效地控制应用程序的访问权限。

## 11.2 获得系统数据

Android SDK 提供了很多 Content Provider 可以读写系统的信息，例如，读取系统联系人的信息、读取收信箱的短信内容，这些都需要通过 query 方法来查询，当然，通过与其对应的 insert、delete 和 update 方法也可以修改这些数据。

### 11.2.1 读取联系人信息

工程目录：src\ch11\ch11\_contact\_content\_provider

调用其他程序的 Content Provider 与调用 Activity 类似，也需要一个字符串来描述。调用 Activity 叫 Activity Action，调用 Content Provider 叫 URI，而且这个 URI 必须以“content://”开头。

本节将介绍我们调用的第一个系统 Content Provider，这个 Content Provider 可以获得系统的联系人信息。在编写程序之前，请务必保证自己的模拟器或手机中有一些联系人信息，没有可以添加几个。

通过 Content Provider 查询系统数据需要调用 ContentResolver.query 方法，该返回的定义如下：

```
public final Cursor query(Uri uri, String[] projection,
                           String selection, String[] selectionArgs, String sortOrder)
```

query 方法返回了一个 Cursor 对象，这个 Cursor 对象和查询 SQLite 数据库返回的 Cursor 对象完全一样，可以直接访问 Cursor 对象中的数据，也可以将其和 CursorAdapter 一起使用。query 方法有 5 个参数，这 5 个参数的含义如下。

- uri：表示 Content Provider 的 Uri。例如，Uri.parse("content://sms/inbox")。
- projection：相当于 SQL 语言中 select 和 from 之间的部分，也就是查询结果返回的字段，例如，“name,salary”。
- selection：相当于 SQL 语言中 where 子句后面的部分，也就是查询条件。例如，“name = ? and salary > ?”。
- selectionArgs：如果 selection 参数的值包含了参数，也就是问号（?），selectionArgs 则表示这些参数值。如果 selection 参数的值不包含任何参数，selectionArgs 参数值为 null，例如，new String[]{ “bill”, “1200” }
- sortOrder：相当于 SQL 语句中 order by 子句后面的部分，也就是要排序的字段，例如，“name, salary desc”。

#### 注意

在设置 projection、selection、sortOrder 这几个参数时，不能加与其对应的关键字，例如，projection 参数值不能设成“select name, salary”，而应该设成“name, salary”。如果不设置这些参数，可以将它们的值设为 null。

下面来看看如何通过 Content Provider 获得系统的联系人信息。

```
ListView listView = (ListView) findViewById(R.id.listview);
```

```
// 查询系统中所有的联系人
Cursor cursor = getContentResolver().query(
    ContactsContract.Contacts.CONTENT_URI, null, null, null, null);
// 根据 cursor 创建 SimpleCursorAdapter 对象
SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1, cursor, new String[]
    { ContactsContract.Contacts.DISPLAY_NAME }, new int[]
    { android.R.id.text1 });
// 在 ListView 控件中显示联系人列表
listView.setAdapter(simpleCursorAdapter);
```

在上面的代码中只使用了 Uri，其他的参数值都为 null。uri 参数的值是一个常量，该常量的定义如下：

```
public static final String AUTHORITY = "com.android.contacts";
public static final Uri AUTHORITY_URI = Uri.parse("content://" + AUTHORITY);
public static final Uri CONTENT_URI = Uri.withAppendedPath(AUTHORITY_URI, "contacts");
```

从上面的代码可以看出，CONTENT\_URI 实际上是一些常量的组合，如果不使用这些常量，可以使用如下的 Uri 来查询联系人信息。

```
Uri uri = Uri.parse("content://com.android.contacts/contacts");
```

返回查询结果集后，就可以像查询 SQLite 数据库获得的 Cursor 对象一样来操作了。在本例中只使用了返回结果集中的联系人姓名，也就是 display\_name，用 DISPLAY\_NAME 常量表示。

查询系统联系人信息需要在 AndroidManifest.xml 文件中使用下面的代码打开权限。

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

运行本例，会看到如图 11.1 所示的显示效果。



▲ 图 11.1 获得系统联系人信息

### 扩展学习：使用 Content Provider 对数据进行增、删、改操作

除了 query 方法外，还有 3 个方法（insert、delete 和 update）可以对数据进行增、删、改操作。这 3 个方法的定义如下：

```
public final Uri insert(Uri url, ContentValues values)
public final int delete(Uri url, String where, String[] selectionArgs)
public final int update(Uri uri, ContentValues values, String where, String[]
selectionArgs)
```

上面 3 个方法的很多参数与 query 方法相同，但 values 参数是我们以前未接触到的。该参数的类型是 ContentValues。实际上，ContentValues 相当于 Map 的扩展（在 ContentValues 内部也封装了一个 HashMap 对象来保存数据）。用于设置插入、更新的字段名和字段值。key 表示字段名，value 表示字段值。例如，下面的代码插入了一条记录。

```
ContentValues contentValues = new ContentValues();
contentValues.put("name", "bill");
contentValues.put("salary", 1200);
getContentResolver().insert(Uri.parse("content://mycontentprovider"), contentValues);
```

### 11.2.2 查看收到的短信

**工程目录:** src\ch11\ch11\_sms\_content\_provider

我们可以使用“content://sms/inbox”来读取系统的短信（收到的短信），代码如下：

```
ListView lvShortMessages = (ListView) findViewById(R.id.lvShortMessages);
// 按照指定条件查询系统中收到的短信短信
Cursor cursor = getContentResolver().query(
    Uri.parse("content://sms/inbox"), null, "address like ?",
    new String[] { "1%" }, null);
SMSAdapter smsAdapter = new SMSAdapter(this, cursor);
lvShortMessages.setAdapter(smsAdapter);
```

上面的代码只能读取系统中收到的短信，如果想查询所有的短信，可以使用“content://sms”。本例不仅使用了一个 Uri，还使用了 where 条件进行查询。这个 where 条件相当于“where address like ‘1%’”，也就是所有以“1”开头的电话号，其中 address 是查询结果集中的字段，表示电话号。在测试本程序之前，可以向模拟器发几条电话号以“1”开头的短信。

获得的短信内容通过 SMSAdapter 类显示在列表中，代码如下：

```
package mobile.android.ch11.sms.content.provider;

import android.content.Context;
import android.database.Cursor;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.CursorAdapter;
import android.widget.TextView;

public class SMSAdapter extends CursorAdapter
{
    private LayoutInflater layoutInflater;
    public SMSAdapter(Context context, Cursor cursor)
    {
        super(context, cursor);
        layoutInflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }
    @Override
    public View newView(Context context, Cursor cursor, ViewGroup parent)
    {
```

```

// 装载列表项的布局
View view = layoutInflater.inflate(R.layout.item, null);
return view;
}
@Override
public void bindView(View view, Context context, Cursor cursor)
{
    TextView tvPhoneNumber = (TextView) view
        .findViewById(R.id.tvPhoneNumber);
    TextView tvContent = (TextView) view.findViewById(R.id.tvContent);
    // 显示电话号
    tvPhoneNumber.setText(cursor.getString(cursor.getColumnIndex("address")));
    // 显示短信内容
    tvContent.setText(cursor.getString(cursor.getColumnIndex("body")));
}
}

```

运行本例后，会看到如图 11.2 所示的显示效果。



▲ 图 11.2 显示短信内容

## 11.3 自定义 Content Provider

我们自己开发的程序也可通过 Content Provider 向外共享数据，所有的 Content Provider 都必须从 ContentProvider 类继承。由于 ContentProvider 是一个抽象类，因此，所有的抽象方法（例如，query、insert 等）都必须实现。在编写完 Content Provider 类后，必须在 AndroidManifest.xml 文件中注册才可以使用。

### 11.3.1 查询城市信息

**工程目录：**src\ch11\ch11\_region\_content\_provider, src\ch11\ch11\_invoke\_content\_provider

本小节将实现一个可以查询全国省份和城市的 Content Provider。下面先看看实现 Content Provider 的步骤。

- (1) 编写一个类，该类必须继承自 ContentProvider。
- (2) 实现 ContentProvider 类中所有抽象方法（可以用 Eclipse 自动生成这些方法的框架）。
- (3) 定义 Content Provider 的 URI。URI 分为 authority 和 path 两部分，其中 authority 就是“content://authority”中的 authority，相当于网址中的域名，而 path 就是“content://authority/”后面

的部分，与网址中的路径类似。

(4) 使用 UriMatcher 对象映射 Uri 和返回代码。

(5) 根据实际的需要实现相应的方法。例如，我们只想对数据进行只读操作，可以只实现 query 方法， insert、delete 和 update 方法可以直接返回 null。

(6) 在 AndroidManifest.xml 文件中使用<provider>标签注册 Content Provider。

当按照上面的步骤编写和注册完 Content Provider 类后，运行程序。Content Provider 就会和 Broadcast Receiver 一样，随着手机或模拟器的启动而自动常驻内存。除非卸载包含 Content Provider 的程序，否则这个 Content Provider 可以一直使用。

在编写程序之前，先准备一个包含全国所有省市数据的 SQLite 数据库，该数据库已经包含在 ch11\_region\_content\_provider 工程的 assets 目录中。本例要采用 8.3.5 小节的方法将数据库与程序一起发布，然后当程序第一次启动时会将数据库文件复制到 SD 卡的根目录。数据库中有两个表 (t\_cities 和 t\_provinces) 和一个视图 (v\_cities\_province)，数据和结构如图 11.3、图 11.4 和图 11.5 所示。

RecNo	city_code	city_name	province_code
Click here to define a filter			
1	0991	乌鲁木齐	991
2	0906	阿勒泰	991
3	0909	博州	991

▲ 图 11.3 t\_cities 表

RecNo	province_code	province_name
Click here to define a filter		
1	991	新疆
2	24	辽宁
3	27	湖北

▲ 图 11.4 t\_provinces 表

RecNo	city_code	city_name	province_code	province_name
Click here to define a filter				
1	0991	乌鲁木齐	991	新疆
2	0906	阿勒泰	991	新疆
3	0909	博州	991	新疆
4	0901	塔城	991	新疆

▲ 图 11.5 v\_cities\_province

RegionContentProvider 类是 ContentProvider 的子类，由于本例只用于查询省、市的信息，所以只实现了 query 方法，其他的方法都返回 null。RegionContentProvider 类的代码如下：

```
package mobile.android.ch11.region.content.provider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import android.content.ContentProvider;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
import android.util.Log;

public class RegionContentProvider extends ContentProvider
```

```
{  
    private static UriMatcher uriMatcher;  
    // 定义 authority  
    private static final String AUTHORITY = "mobile.android.ch11.regioncontentprovider";  
    // 下面 4 个常量是返回码  
    private static final int CITIES = 1;  
    private static final int CITY_CODE = 2;  
    private static final int CITY_NAME = 3;  
    private static final int CITIES_IN_PROVINCE = 4;  
    private SQLiteDatabase database;  
  
    static  
    {  
        // 开始映射 Uri 和返回码  
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);  
        // 用于查询所有城市的 Uri  
        uriMatcher.addURI(AUTHORITY, "cities", CITIES);  
        // 用于根据城市代码查询城市信息的 Uri  
        uriMatcher.addURI(AUTHORITY, "code/#", CITY_CODE);  
        // 用于根据城市名称查询城市信息的 Uri  
        uriMatcher.addURI(AUTHORITY, "name/*", CITY_NAME);  
        // 用于根据省名称查询省内所有城市的信息  
        uriMatcher.addURI(AUTHORITY, "cities_in_province/*", CITIES_IN_PROVINCE);  
    }  
    // 复制和打开数据库  
    private SQLiteDatabase openDatabase()  
    {  
        try  
        {  
            String databaseFilename = "/sdcard/region.db";  
            if (!(new File(databaseFilename)).exists())  
            {  
                InputStream is = getContext().getResources().getAssets()  
                    .open("region.db");  
                FileOutputStream fos = new FileOutputStream(databaseFilename);  
                byte[] buffer = new byte[8192];  
                int count = 0;  
                while ((count = is.read(buffer)) > 0)  
                {  
                    fos.write(buffer, 0, count);  
                }  
  
                fos.close();  
                is.close();  
            }  
            SQLiteDatabase database = SQLiteDatabase.openOrCreateDatabase(  
                databaseFilename, null);  
            return database;  
        }  
        catch (Exception e)  
        {  
            Log.d("error", e.getMessage());  
        }  
        return null;  
    }  
}
```

## Android 开发权威指南

```

@Override
public boolean onCreate()
{
    // 在创建 Content Provider 时返回数据库
    database = openDatabase();
    return true;
}
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs, String sortOrder)
{
    Cursor cursor = null;
    // 根据 Uri 获得返回码
    switch (uriMatcher.match(uri))
    {
        case CITIES:                      // 查询所有的城市信息
            cursor = database.query("v_cities_province", projection,
                selection, selectionArgs, null, null, sortOrder);
            break;
        case CITY_CODE:                   // 根据城市代码查询城市信息
            String cityCode = uri.getPathSegments().get(1);
            if (selection == null)
                selection = "city_code='"
                    + cityCode + "'";
            else
                selection += " and (city_code='"
                    + cityCode + "'')";
            cursor = database.query("t_cities", projection, selection,
                selectionArgs, null, null, sortOrder);
            break;
        case CITY_NAME:                  // 根据城市名称查询城市信息
            String cityName = uri.getPathSegments().get(1);
            if (selection == null)
                selection = "city_name='"
                    + cityName + "'";
            else
                selection += " and (city_name='"
                    + cityName + "')";    // 组合查询条件
            cursor = database.query("t_cities", projection, selection,
                selectionArgs, null, null, sortOrder);
            break;
        case CITIES_IN_PROVINCE:         // 根据省名称查询省内的所有城市信息
            String provinceName = uri.getPathSegments().get(1);
            if (selection == null)
                selection = "province_name='"
                    + provinceName + "'";
            else
                selection += " and (province_name='"
                    + provinceName + "')";
            cursor = database.query("v_cities_province", projection, selection,
                selectionArgs, null, null, sortOrder);
            break;
        default:
            throw new IllegalArgumentException("<" + uri + ">格式不正确.");
    }
    return cursor;
}
@Override
public String getType(Uri uri)
{
    return null;
}

```

```
}

@Override
public Uri insert(Uri uri, ContentValues values)
{
    return null;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs)
{
    return 0;
}

@Override
public int update(Uri uri, ContentValues values, String selection, String[]
selectionArgs)
{
    return 0;
}

}
```

编写上面代码时应了解以下几点。

- 所有的 Uri 一般需要通过 UriMatcher 对象映射成 int 类型的返回码，因为通过 int 类型的返回码更容易判断是哪个 Uri。通过 UriMatcher.addURI 方法可以为每一个 Uri 设置一个返回码。addURI 有 3 个参数，第一个参数表示 authority，第二个参数表示 path，第三个参数表示返回码。
- 如果 path 只包含一个普通的字符串，例如，abc，那么 Uri 为“content://authority/abc”。如果 path 后面跟着星号（\*），表示任意的字符串，例如，如果 path 的值是“abc/\*”，那么“content://authority/abc/x”和“content://authority/abc/xyz”都会与这个 Uri 匹配。当 path 后面是井号（#）时，表示任意的数字，例如，path 的值是“abc/#”，那么“content://authority/abc/345”、“content://authority/abc/555”都会与这个 Uri 匹配。
- 当应用程序启动后，如果在 AndroidManifest.xml 文件中注册了 RegionContentProvider，会自动创建 RegionContentProvider 对象，并调用 onCreate 方法。因此，程序第一次启动后会自动将 assets 目录中的数据库文件复制到 SD 的根目录，并打开该数据库。
- 在 query 方法中根据 UriMatcher.match 方法将 Uri 转换成返回码，并在 switch 语句中的不同分支处理不同的 Uri。
- 尽管可以使用 SQLiteDatabase.rawQuery 方法返回 Cursor 对象，但由于 query 方法是将 SQL 语句分成了若干个部分进行传递的，因此，如果不是必须，建议使用 SQLiteDatabase.query 方法返回 Cursor 对象，因为这两个 query 方法的参数相同，可以将参数直接传给 SQLiteDatabase.query 方法。
- 通过 Uri.getPath 方法可以返回完整的路径，例如，“content://authority/code/024”，使用 getPath 方法会返回“/code/024”。通过 Uri.getPathSegments()方法可以将 path 进行分解，例如，对前面的 Uri 使用 getPathSegments()方法可以返回一个长度为 2 的 List<String> 对象。第一个元素的值是“code”，第二个元素的值是“024”，因此，我们可以使用 getPathSegments 直接获得城市代码、城市名称和省名称。
- 如果通过 Uri 来传递查询条件，会涉及条件组合的问题。如果 ContentProvider.query 方法的 selection 参数值为 null，直接使用 Uri 指定的条件即可，例如，“city\_code=‘024’”。如果 selection 不为 null，则需要使用 and 将两个条件进行组合。

下面来完成实现 Content Provider 的最后一步，注册 Content Provider，打开 AndroidManifest.xml 文件，在<application>标签中添加如下的代码。

```
<provider android:name="RegionContentProvider"
          android:authorities="mobile.android.ch11.regioncontentprovider" />
```

其中 android:name 属性表示 Content Provider 类名，也就是 RegionContentProvider。 android:authorities 属性表示 authority，必须和 UriMatcher.addURI 方法的第一个参数值一致。

由于本例需要向 SD 卡写数据，因此，需要使用下面的代码打开写权限。

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

ch11\_invoke\_content\_provider 程序使用下面的代码获得了所有城市的信息，以及通过城市代码和城市名称查询了相应的信息。

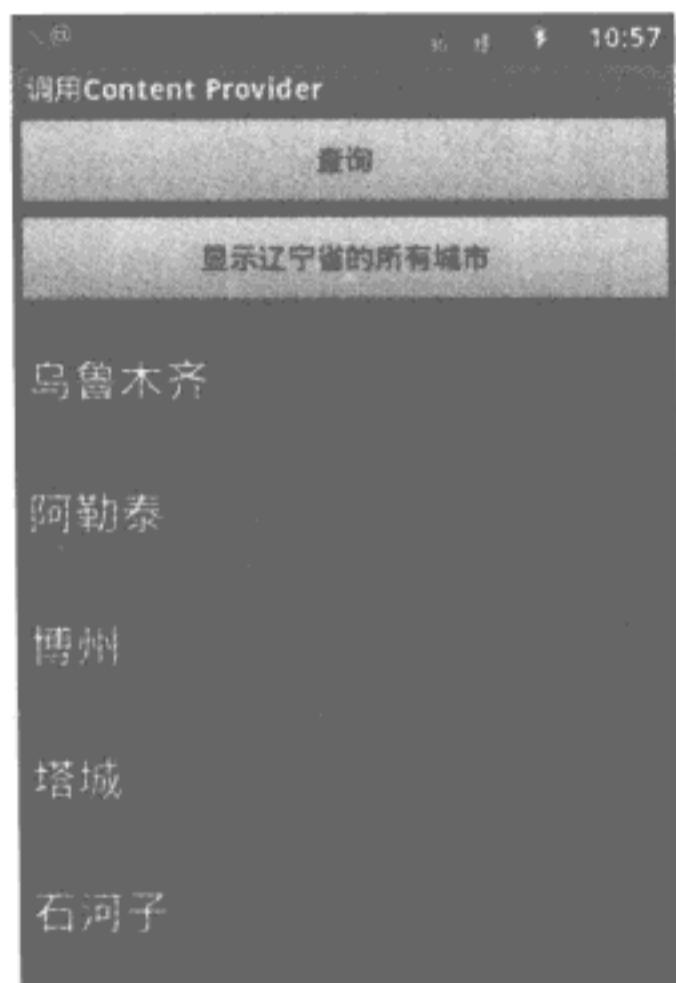
```
ContentResolver contentResolver = getContentResolver();
// 指定用于查询所有城市信息的 Uri
Uri uri = Uri.parse("content://mobile.android.ch11.regioncontentprovider/cities");
// 查询所有的城市信息，由于 CursorAdapter 对象需要 "_id" 字段，因此，将 city_code 的别名设为 "_id"
Cursor cursor = contentResolver.query(uri, new String[]
{ "city_code as _id", "city_name", "province_code" }, null, null, null);
SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1, cursor, new String[]{ "city_name" }, new
int[]{ android.R.id.text1 });
ListView lvCities = (ListView) findViewById(R.id.lvCities);
lvCities.setAdapter(simpleCursorAdapter);
// 指定根据城市代码查询城市信息的 Uri
uri = Uri.parse("content://mobile.android.ch11.regioncontentprovider/code/024");
// 根据城市代码查询城市信息
cursor = contentResolver.query(uri, null, null, null, null);
if (cursor.moveToFirst())
{
    Toast.makeText(this, "024: " + cursor.getString(cursor.getColumnIndex("city_name")),
        Toast.LENGTH_LONG).show();
}
// 指定根据城市名称查询城市信息的 Uri
uri = Uri.parse("content://mobile.android.ch11.regioncontentprovider/name/沈阳");
// 根据城市名称查询城市信息
cursor = contentResolver.query(uri, null, null, null, null);
if (cursor.moveToFirst())
{
    Toast.makeText(this, "沈阳: " + cursor.getString(cursor.getColumnIndex("city_code")),
        Toast.LENGTH_LONG).show();
}
```

下面的代码查询了辽宁省的所有城市信息。

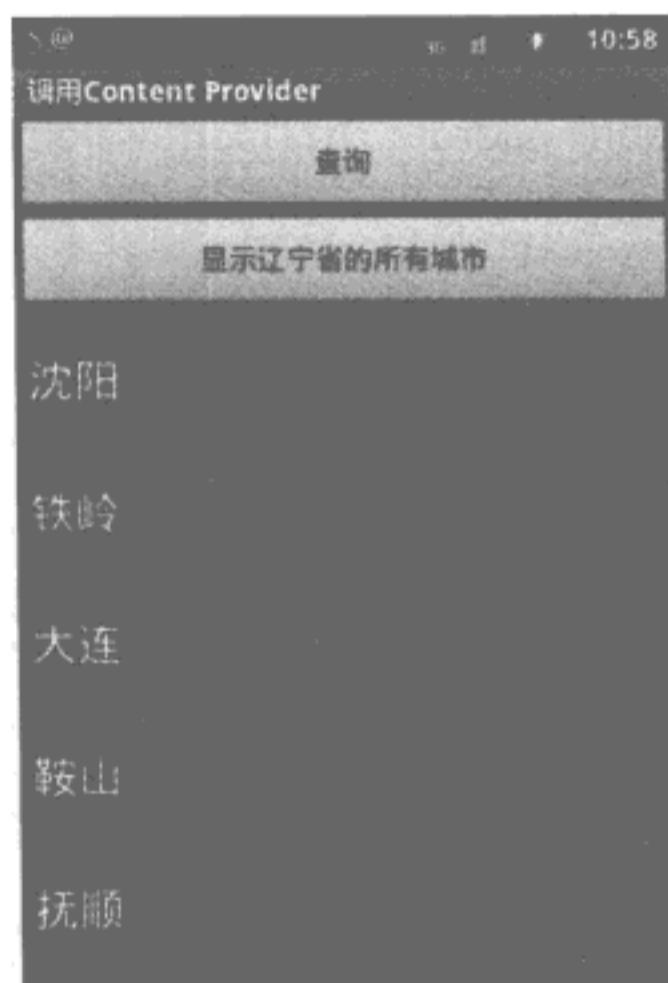
```
ContentResolver contentResolver = getContentResolver();
// 指定根据省名称查询省内城市信息的 Uri
Uri uri = Uri.parse("content://mobile.android.ch11.regioncontentprovider/cities_in_
province/辽宁");
// 根据省名称查询省内城市的信息
Cursor cursor = contentResolver.query(uri, new String[]
{ "city_code as _id", "city_name", "province_code" }, null, null, "city_code");
```

```
SimpleCursorAdapter simpleCursorAdapter = new SimpleCursorAdapter(this,
    android.R.layout.simple_list_item_1, cursor, new String[]{ "city_name" }, new
    int[]{ android.R.id.text1 });
ListView lvCities = (ListView) findViewById(R.id.lvCities);
lvCities.setAdapter(simpleCursorAdapter);
```

在执行上面代码时，应先运行 ch11\_region\_content\_provider 程序。显示所有城市和辽宁省的城市名称的效果如图 11.6 和图 11.7 所示。



▲ 图 11.6 显示所有城市的名称



▲ 图 11.7 显示辽宁省的城市名称

### 11.3.2 为 Content Provider 添加访问权限

工程目录: src\ch11\ch11\_permission\_region\_content\_provider  
src\ch11\ch11\_permission\_invoke\_content\_provider

在访问很多系统提供的 Content Provider 时需要在调用时打开访问权限。实际上，我们也可以为自己编写的 Content Provider 添加权限。在调用时只有使用<uses-permission>标签设置了访问权限才能访问 Content Provider。

现在将上一小节的两个工程各复制一份（ch11\_permission\_region\_content\_provider 和 ch11\_permission\_invoke\_content\_provider）。首先为 ch11\_permission\_region\_content\_provider 来添加权限，打开 AndroidManifest.xml 文件。按如下代码修改<provider>标签。在本例中将 authority 设了一个新值，读者也可以将其设成任何的值。

```
<provider android:name="RegionContentProvider"
    android:authorities="mobile.android.ch11.permission.regioncontentprovider"
    android:readPermission="mobile.android.ch11.permission.regioncontent
    provider.READ_REGION"
/>
```

上面的代码设置了<provider>标签的 android:readPermission 属性，用该属性设置的权限只控制 query 方法，也就是读权限。如果想设置写权限，可以设置 android:writePermission 属性。当然，也

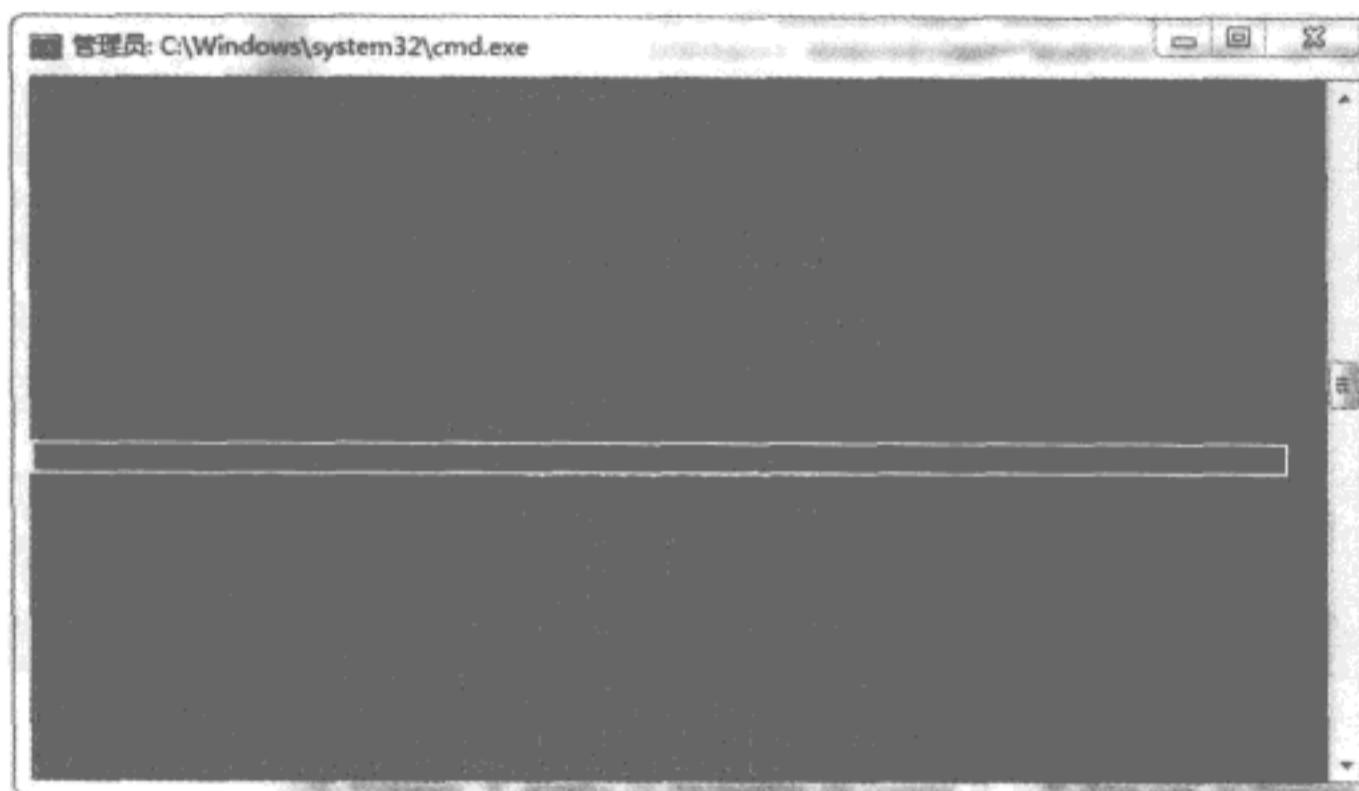
可以通过 `android:permission` 属性同时设置读写权限，这时就不需要设置 `android:readPermission` 和 `android:writePermission` 属性了。

光指定访问权限还不行，还需要使用`<permission>`标签定义这个权限，代码如下：

```
<permission
    android:name="mobile.android.ch11.permission.regioncontentprovider.READ_REGION"
    android:protectionLevel="normal" android:label="@string/label"
    android:description="@string/description" />
```

如果权限注册成功，在 Windows 控制台输入如下的命令，会看到如图 11.8 所示的权限列表中包含有 `mobile.android.ch11.permission.regioncontentprovider.READ_REGION` 权限。

```
| adb shell pm list permissions
```



▲ 图 11.8 系统中注册的权限列表

现在来修改 `ch11_permission_invoke_content_provider` 工程的代码。打开 `AndroidManifest.xml` 文件，加上如下的代码打开读权限。

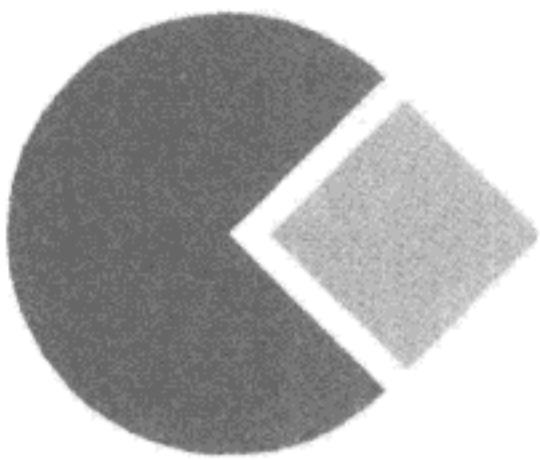
```
<uses-permission
    android:name="mobile.android.ch11.permission.regioncontentprovider.READ_REGION" />
```

由于 Content Provider 的 Uri 已经修改，因此，调用者也需要修改 Uri，例如，查询辽宁省所有城市的 Uri 如下：

```
Uri uri = Uri.parse("content://mobile.android.ch11.permission.regioncontentprovider/
cities_in_province/辽宁");
```

## 11.4 小结

本章主要介绍了 Content Provider 的使用方法，Content Provider 主要用于与其他程序共享数据。Android 系统本身也提供了很多 Content Provider（例如，联系人、短信等），第三方程序可以通过这些 Content Provider 获得系统数据。如果我们自己的程序有数据需要共享，也可以定制 Content Provider。



## 第 12 章 一切为用户服务—— Service 基础与实例

服务（Service）是 Android 系统中 4 个应用程序组件之一。服务主要用于两个目的：后台运行和跨进程访问。通过启动一个服务，可以在不显示界面的前提下在后台运行指定的任务，这样可以不影响用户做其他事情。通过 AIDL 服务可以实现不同进程之间的通信，这也是服务的重要用途之一。

### 12.1 Service 基础

Service 并没有实际界面，而是一直在 Android 系统的后台运行。一般使用 Service 为应用程序提供一些服务，或不需要界面的功能，例如，从 Internet 下载文件、控制 Video 播放器等。本节主要介绍 Service 的启动和结束过程（Service 的生命周期）以及启动 Service 的各种方法。

#### 12.1.1 Service 的生命周期

**工程目录：**src\ch12\ch12\_servicelifecycle

Service 与 Activity 一样，也有一个从启动到销毁的过程，但 Service 的这个过程比 Activity 简单得多。Service 从启动到销毁的过程只会经历如下 3 个阶段：

- 创建服务；
- 开始服务；
- 销毁服务。

一个服务实际上是一个继承自 android.app.Service 的类，当服务经历上面 3 个阶段后，会分别调用 Service 类中的 3 个事件方法进行交互，这 3 个事件方法如下：

```
public void onCreate();                                // 创建服务
public void onStart(Intent intent, int startId);    // 开始服务
public void onDestroy();                             // 销毁服务
```

一个服务只会创建一次，销毁一次，但可以开始多次，因此，onCreate 和 onDestroy 方法只会被调用一次，而 onStart 方法会被调用多次。

下面编写一个服务类，具体看一下服务的生命周期由开始到销毁的过程。

```
package mobile.android.ch12.service.lifecycle;
import android.app.Service;
```

## Android 开发权威指南

```

import android.content.Intent;
import android.os.IBinder;
import android.util.Log;

// MyService 是一个服务类，该类必须从 android.app.Service 类继承
public class MyService extends Service
{
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
    // 当服务第 1 次创建时调用该方法
    @Override
    public void onCreate()
    {
        Log.d("MyService", "onCreate");
        super.onCreate();
    }
    // 当服务销毁时调用该方法
    @Override
    public void onDestroy()
    {
        Log.d("MyService", "onDestroy");
        super.onDestroy();
    }
    // 当开始服务时调用该方法
    @Override
    public void onStart(Intent intent, int startId)
    {
        Log.d("MyService", "onStart");
        super.onStart(intent, startId);
    }
}

```

在 MyService 类中覆盖了 Service 类的 3 个生命周期方法，并在这些方法中输出了相应的日志信息，以便更容易地观察事件方法的调用情况。

读者在编写 Android 的应用组件时要注意，不管是编写什么组件（例如，Activity、Service 等），都需要在 AndroidManifest.xml 文件中进行配置，MyService 类也不例外。配置这个服务类很简单，只需要在 AndroidManifest.xml 文件的<application>标签中添加如下代码即可：

```
<service android:enabled="true" android:name=".MyService" />
```

其中 android:enabled 属性的值为 true，表示 MyService 服务处于激活状态。虽然目前 MyService 是激活的，但系统仍然不会启动 MyService，要想启动这个服务。必须显式地调用 startService 方法。如果想停止服务，需要显式地调用 stopService 方法，代码如下：

```

public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.btnStartService:

```

```

        startService(serviceIntent);           // 单击 "Start Service" 按钮启动服务
        break;
    case R.id.btnStopService:
        stopService(serviceIntent);          // 单击 "Stop Service" 按钮停止服务
        break;
    }
}

```

其中 `serviceIntent` 是一个 Intent 对象，用于指定 MyService 服务，创建该对象的代码如下：

```
Intent serviceIntent = new Intent(this, MyService.class);
```

运行该例子后，会显示如图 12.1 所示的界面。



▲ 图 12.1 开始和停止服务

第 1 次单击 “Start Service” 按钮后，在 LogCat 视图的 Message 列会输出如下两行信息：

```
onCreate
onStart
```

然后单击 “Stop Service” 按钮，会在 Message 列中输出如下信息：

```
onDestroy
```

下面按如下的单击按钮顺序重新测试一下本例。

```
"Start Service" → "Stop Service" → "Start Service" → "Start Service" → "Start Service"
→ "Stop Service"
```

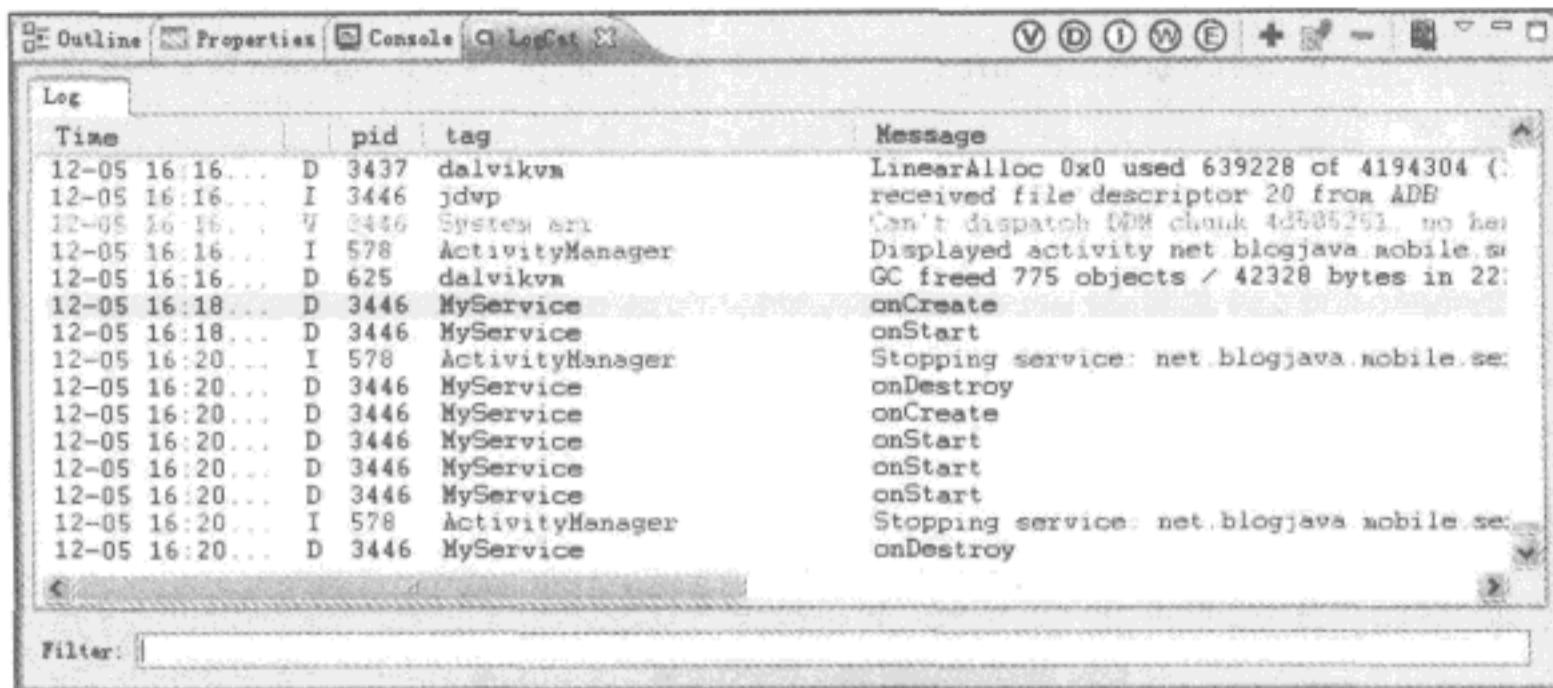
测试完程序，就会看到如图 12.2 所示的输出信息。可以看出，只在第 1 次单击 “Start Service” 按钮后会调用 `onCreate` 方法，如果在未单击 “Stop Service” 按钮时多次单击 “Start Service” 按钮，系统只在第 1 次单击 “Start Service” 按钮时调用 `onCreate` 和 `onStart` 方法，再单击该按钮时，系统只会调用 `onStart` 方法，而不会再次调用 `onCreate` 方法。

在讨论完服务的生命周期后，再来总结一下创建和开始服务的步骤。创建和开始一个服务需要如下 3 步。

(1) 编写一个服务类，该类必须继承自 `android.app.Service`。`Service` 类涉及 3 个生命周期方法，但这 3 个方法并不一定在子类中覆盖，读者可根据不同需求来决定使用哪些生命周期方法。在 `Service` 类中有一个 `onBind` 方法，该方法是一个抽象方法，在 `Service` 的子类中必须覆盖，这个方法当 `Activity` 与 `Service` 绑定时被调用（将在 12.1.3 小节详细介绍）。

(2) 在 `AndroidManifest.xml` 文件中使用`<service>`标签来配置服务，一般需要将`<service>`标签的 `android:enabled` 属性值设为 `true`，并使用 `android:name` 属性指定在第 1 步建立的服务类名。

(3) 如果要开始一个服务，使用 `startService` 方法，停止一个服务要使用 `stopService` 方法。



▲ 图 12.2 服务的生命周期方法的调用情况

### 12.1.2 绑定 Activity 和 Service

工程目录: `src\ch12\ch12_serviceactivity`

如果使用 12.1.1 小节介绍的方法启动服务，并且未调用 `stopService` 来停止服务，这个服务就会随着 Android 系统的启动而启动，随着 Android 系统的关闭而关闭。也就是服务会在 Android 系统启动后一直在后台运行，直到 Android 系统关闭或调用 `stopService` 方法后服务才停止。但有时我们希望在启动服务的 Activity 关闭后服务自动关闭，这就需要将 Activity 和 Service 绑定。

通过 `bindService` 方法可以将 Activity 和 Service 绑定。`bindService` 方法的定义如下：

```
public boolean bindService(Intent service, ServiceConnection conn, int flags)
```

该方法的第一个参数表示与服务类相关联的 Intent 对象，第 2 个参数的类型是 `ServiceConnection`，负责连接 Intent 对象指定的服务。通过 `ServiceConnection` 对象可以获得连接成功或失败的状态，并可以获得连接后的服务对象。第 3 个参数是一个标志位，一般设为 `Context.BIND_AUTO_CREATE`。

下面重新编写 12.1.1 小节的 `MyService` 类，在该类中增加了几个与绑定相关的事件方法。

```
package mobile.android.ch12.service.activity;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class MyService extends Service
{
    private MyBinder myBinder = new MyBinder();
    // 成功绑定后调用该方法
    @Override
    public IBinder onBind(Intent intent)
    {
```

```

    Log.d("MyService", "onBind");
    return myBinder;
}
// 重新绑定时调用该方法
@Override
public void onRebind(Intent intent)
{
    Log.d("MyService", "onRebind");
    super.onRebind(intent);
}
// 解除绑定时调用该方法
@Override
public boolean onUnbind(Intent intent)
{
    Log.d("MyService", "onUnbind");
    return super.onUnbind(intent);
}
@Override
public void onCreate()
{
    Log.d("MyService", "onCreate");
    super.onCreate();
}
@Override
public void onDestroy()
{
    Log.d("MyService", "onDestroy");
    super.onDestroy();
}
@Override
public void onStart(Intent intent, int startId)
{
    Log.d("MyService", "onStart");
    super.onStart(intent, startId);
}
public class MyBinder extends Binder
{
    MyService getService()
    {
        return MyService.this;
    }
}
}

```

现在定义一个 MyService 变量和一个 ServiceConnection 变量，代码如下：

```

private MyService myService;
private ServiceConnection serviceConnection = new ServiceConnection()
{
    // 连接服务失败后，该方法被调用
    @Override
    public void onServiceDisconnected(ComponentName name)
    {
        myService = null;
        Toast.makeText(Main.this, "Service Failed.", Toast.LENGTH_LONG).show();
    }
}

```

```

    }
    // 成功连接服务后，该方法被调用。在该方法中可以获得 MyService 对象
    @Override
    public void onServiceConnected(ComponentName name, IBinder service)
    {
        // 获得 MyService 对象
        myService = ((MyService.MyBinder) service).getService();
        Toast.makeText(Main.this, "Service Connected.", Toast.LENGTH_LONG).show();
    }
}

```

最后使用 bindService 方法来绑定 Activity 和 Service，代码如下：

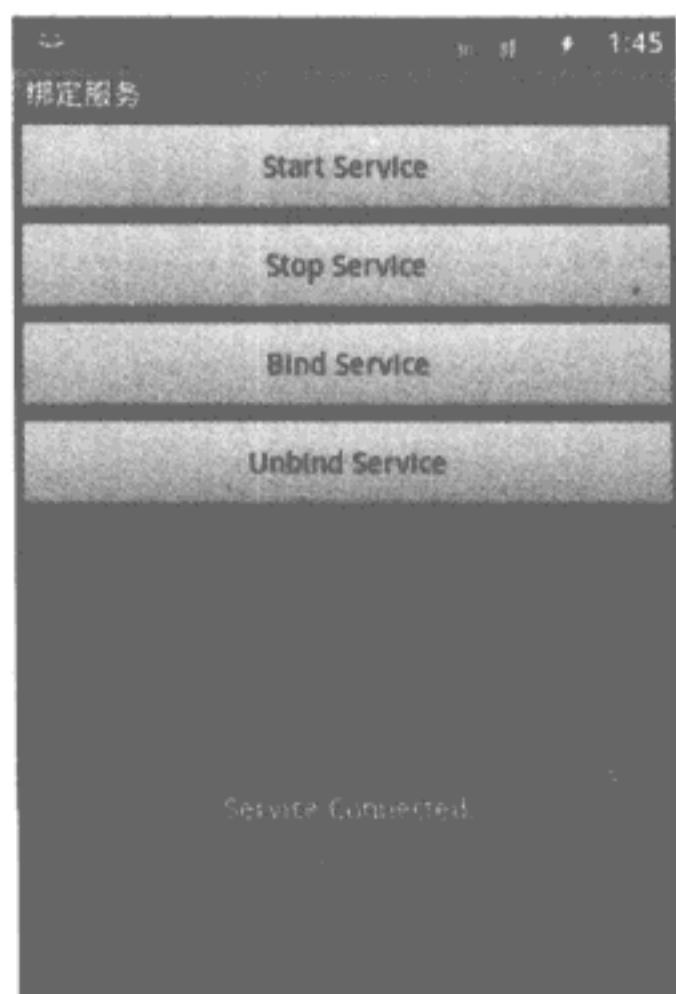
```
bindService(serviceIntent, serviceConnection, Context.BIND_AUTO_CREATE);
```

如果想解除绑定，可以使用下面的代码：

```
unbindService(serviceConnection);
```

在 MyService 类中定义了一个 MyBinder 类，该类用于获得 MyService 对象。ServiceConnection.onServiceConnected 方法的第 2 个参数是一个 IBinder 类型的变量，将该参数转换成 MyService.MyBinder 对象，并使用 MyBinder.getService 方法获得 MyService 对象。在获得 MyService 对象后，就可以在 Activity 中随意操作 MyService 了。

运行本节的例子，单击“Bind Service”按钮，如果绑定成功，会显示如图 12.3 所示的信息提示框。关闭应用程序后，会看到在 LogCat 视图中输出了 onUnbind 和 onDestroy 信息，表明在关闭 Activity 后，服务先被解除绑定，最后被销毁。如果先启动（调用 startService 方法）一个服务，然后再绑定（调用 bindService 方法）服务，会怎么样呢？在这种情况下，虽然服务仍然会成功绑定到 Activity 上，但在 Activity 关闭后，服务虽然会被解除绑定，但并不会被销毁，也就是说，MyService 类的 onDestroy 方法不会被调用。



▲ 图 12.3 绑定服务

### 12.1.3 开机启动 Service

工程目录: src\ch12\ch12\_startupservice

在 12.1.1 小节和 12.1.2 小节都是先启动了一个 Activity, 然后在 Activity 中启动服务, 如果是这样, 在启动服务时必须要先启动一个 Activity。很多时候这样做有些多余, 阅读完 10.2.6 小节的内容, 会发现我们可以利用 Broadcast Receiver 在 Android 系统启动时运行一个 Activity。也许我们会从中得到一些启发, 既然可以在 Broadcast Receiver 中启动 Activity, 为什么不能启动 Service 呢? 说做就做, 现在让我们来验证一下这个想法。

先编写一个服务类, 这个服务类没什么特别的, 仍然使用前面两小节编写的 MyService 类即可。在 AndroidManifest.xml 文件中配置 MyService 类的代码也相同。

下面来完成最关键的一步, 就是建立一个 BroadcastReceiver, 代码如下:

```
package mobile.android.ch12.startup.service;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class StartupReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Intent serviceIntent = new Intent(context, MyService.class);
        // 启动 Service
        context.startService(serviceIntent);
        Intent activityIntent = new Intent(context, MessageActivity.class);
        // 要想在 Service 中启动 Activity, 必须设置如下标志
        activityIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
        context.startActivity(activityIntent);
    }
}
```

在 StartupReceiver.onReceive 方法中完成了两项工作: 启动服务和显示一个 Activity 来提示服务启动成功。其中 MessageActivity 是一个普通的 Activity 类, 只是该类在配置时使用了“@android:style/Theme.Dialog”主题, 因此, 如果服务启动成功, 会显示如图 12.4 所示的信息。

如果安装本例后, 在重新启动模拟器后并未出现如图 12.4 所示的信息提示框, 最大的可能是没有在 AndroidManifest.xml 文件中配置 BroadcastReceiver 和 Service, 下面来看一下 AndroidManifest.xml 文件的完整代码。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.blogjava.mobile.startupservice" android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".MessageActivity" android:theme="@android:style/Theme.
        Dialog">
            <intent-filter>
                <category android:name="android.intent.category.LAUNCHER" />
            
```

```

        </intent-filter>
    </activity>
    <receiver android:name="StartupReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
    <service android:enabled="true" android:name=".MyService" />
</application>
<uses-sdk android:minSdkVersion="3" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
</manifest>

```

现在运行本例，然后重启一下模拟器，看看 LogCat 视图中是否输出了相应的日志信息。



▲ 图 12.4 在 BroadcastReceiver 中启动服务

#### 12.1.4 判断 Service 是否已注册

工程目录: src\ch12\ch12\_query\_service

Android SDK 并未直接提供 API 来判断某个 Service 是否已注册，但可以通过 PackageManager.queryIntentServices 方法根据 IntentFilter 来查询系统中某个或某组服务。

首先修改 12.1.2 小节的例子中服务的注册代码，添加一个<intent-filter>标签以及一个<action>标签。

```

<service android:enabled="true" android:name=".MyService">
    <intent-filter>
        <action android:name="mobile.android.ch12.service.activity.MyService" />
    </intent-filter>
</service>

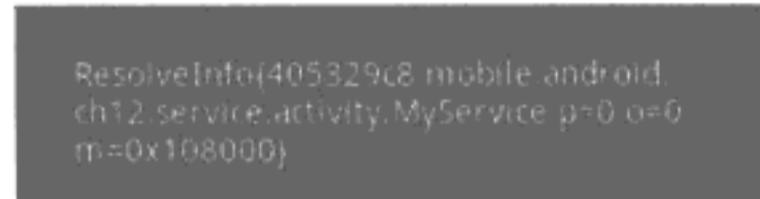
```

其中<action>标签的 android:name 属性值就相当于要查找的 ID。查找这个服务的代码如下：

```
PackageManager packageManager = getPackageManager();
```

```
// 指定要查询的 Service Action
Intent intent = new Intent("mobile.android.ch12.service.activity.MyService");
// 查询服务
List<ResolveInfo> resolveInfos = packageManager.queryIntentServices(
    intent, PackageManager.GET_INTENT_FILTERS);
if (resolveInfos.size() > 0)
{
    ResolveInfo resolveInfo = resolveInfos.get(0);
    // 显示查询到的服务信息
    Toast.makeText(this, resolveInfo.toString(), Toast.LENGTH_LONG).show();
}
else
{
    Toast.makeText(this, "服务还没有注册。", Toast.LENGTH_LONG).show();
}
```

首先运行 12.1.2 小节中的例子，然后单击“Start Service”按钮，最后执行上面的代码，会显示如图 12.5 所示的弹出信息。



▲ 图 12.5 显示已注册的服务信息

### 12.1.5 判断 Service 是否已开始

工程目录: src\ch12\ch12\_service\_state

`PackageManager.queryIntentServices` 方法只能查询已注册的服务，不管这个服务是否正在运行，都可以被查到。如果只想确定某个服务是否正在运行，就需要使用 `ActivityManager.getRunningServices` 方法获得系统中所有正在运行的服务，然后在其中查询指定的服务。`getRunningServices` 方法的定义如下：

```
public List<RunningServiceInfo> getRunningServices(int maxNum)
```

`maxNum` 参数表示返回正在运行的服务的最大数，如果正在运行的服务小于 `maxNum`，则按实际数返回。一般可以将该参数值设为较大的数，如 100。

本例仍然判断 12.1.2 小节中的例子的服务是否正在运行。首先运行 12.1.2 小节中的例子，然后单击“Start Service”按钮开始服务。

下面的代码可以查询这个服务是否正处于运行状态。

```
ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
// 返回所有正在运行的服务信息
List<ActivityManager.RunningServiceInfo> runningServiceInfos = activityManager.getRunningServices(100);
for (int i = 0; i < runningServiceInfos.size(); i++)
{
    ActivityManager.RunningServiceInfo runningServiceInfo = runningServiceInfos.get(i);
    // 通过服务的类名判断指定服务是否正在运行
```

```

if("mobile.android.ch12.service.activity.MyService".equals(runningServiceInfo.service
.getClassName()))
{
    Toast.makeText(this, "服务正在运行...", Toast.LENGTH_LONG).show();
    return;
}
Toast.makeText(this, "服务没有开始", Toast.LENGTH_LONG).show();

```

上面的代码通过服务的类名判断指定的服务是否正在运行。在 12.1.2 小节例子中的服务正在运行时执行上面的代码，会显示“服务正在运行...”信息框。单击 12.1.2 小节例子中的“Stop Service”按钮停止服务，会显示“服务没有开始”信息框。

## 12.2 跨进程访问（AIDL 服务）

Android 系统中的进程之间不能共享内存，因此，需要提供一些机制在不同进程之间进行数据通信，前面介绍的 Activity、Broadcast 和 Content Provider 都可以跨进程通信。现在我们已经了解了 4 个 Android 应用程序组件中的 3 个（Activity、Broadcast 和 Content Provider）都可以进行跨进程访问，另外一个 Android 应用程序组件 Service 同样可以。这就是本节要介绍的 AIDL 服务。

### 12.2.1 什么是 AIDL 服务

本章前面的部分介绍了开发人员如何定制自己的服务，但这些服务并不能被其他的应用程序访问，为了使其他的应用程序也可以访问本应用程序提供的服务，Android 系统采用了远程过程调用（Remote Procedure Call，RPC）方式来实现。与很多其他的基于 RPC 的解决方案一样，Android 使用一种接口定义语言（Interface Definition Language，IDL）来公开服务的接口，因此，可以将这种跨进程访问的服务称为 AIDL（Android Interface Definition Language）服务。

### 12.2.2 建立 AIDL 服务的步骤

建立 AIDL 服务要比建立普通的服务复杂一些，具体步骤如下。

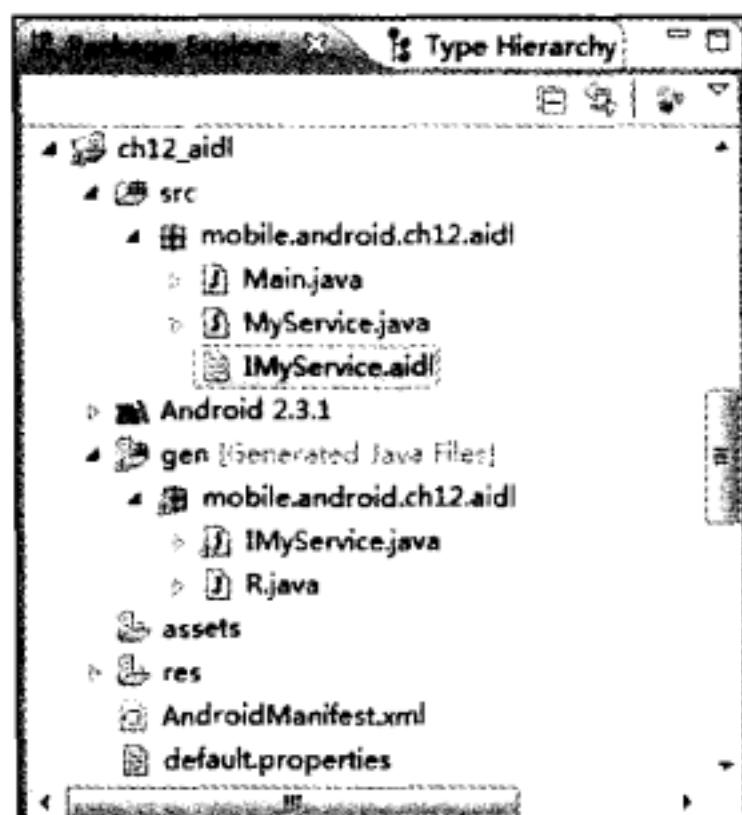
- (1) 在 Eclipse Android 工程的 Java 源文件目录中建立一个扩展名为 aidl 的文件，该文件的语法类似于 Java 代码，但会稍有不同。详细介绍见 12.2.3 小节的内容。
- (2) 如果 aidl 文件的内容是正确的，ADT 会自动生成一个 Java 接口文件 (\*.java)。
- (3) 建立一个服务类（Service 的子类）。
- (4) 实现由 aidl 文件生成的 Java 接口。
- (5) 在 AndroidManifest.xml 文件中配置 AIDL 服务，尤其要注意的是，<action> 标签中 android:name 的属性值就是客户端要引用该服务的 ID，也就是 Intent 类构造方法的参数值。

### 12.2.3 建立 AIDL 服务

工程目录：src\ch12\ch12\_aidl、src\ch12\ch12\_aidlclient

本小节将建立一个简单的 AIDL 服务，这个 AIDL 服务只有一个 `getValue` 方法，该方法返回一个 `String` 类型的值。在安装完服务后，会在客户端调用这个 `getValue` 方法，并将返回值在 `TextView` 控件中显示。建立这个 AIDL 服务的步骤如下。

(1) 建立一个 aidl 文件。在 Java 源文件目录中建立一个 `IMyService.aidl` 文件。`IMyService.aidl` 文件的位置如图 12.6 所示。



▲ 图 12.6 `IMyService.aidl` 文件的位置

`IMyService.aidl` 文件的内容如下：

```
package mobile.android.ch12.aidl;
interface IMyService
{
    String getValue();
}
```

`IMyService.aidl` 文件的内容与 Java 代码非常相似，但要注意，不能加修饰符（例如，`public`、`private`）、AIDL 服务不支持的数据类型（例如，`InputStream`、`OutputStream`）等内容。

(2) 如果 `IMyService.aidl` 文件中的内容输入正确，ADT 会自动生成一个 `IMyService.java` 文件，读者一般并不需要关心这个文件的具体内容，也不需要维护这个文件。关于该文件的具体内容，读者可以查看随书源代码。

(3) 编写一个 `MyService` 类。`MyService` 类继承自 `Service`，在 `MyService` 类中定义了一个内嵌类 (`MyServiceImpl`)，该类继承自 `IMyService.Stub`。`MyService` 类的代码如下：

```
package mobile.android.ch12.aidl;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;

public class MyService extends Service
{
    public class MyServiceImpl extends IMyService.Stub
```

```

    {
        @Override
        public String getValue() throws RemoteException
        {
            return "《Android/OPhone 开发完全讲义》";
        }
    }
    @Override
    public IBinder onBind(Intent intent)
    {
        return new MyServiceImpl();
    }
}

```

在编写上面代码时要注意如下几点。

- IMyService.Stub 是根据 IMyService.aidl 文件自动生成的，一般并不需要了解这个类的内容，只需要编写一个继承自 IMyService.Stub 的类（MyServiceImpl 类）即可。
- onBind 方法必须返回 MyServiceImpl 对象，否则客户端无法获得服务对象。

(4) 在 AndroidManifest.xml 文件中配置 MyService 类，代码如下：

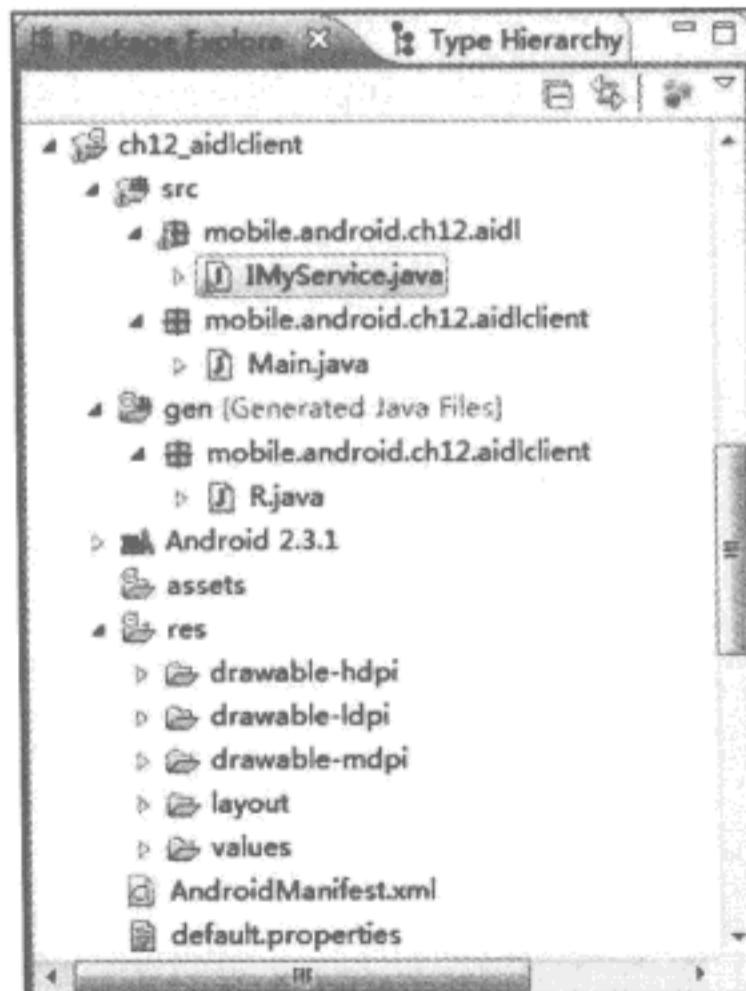
```

<service android:name=".MyService" >
    <intent-filter>
        <action android:name="mobile.android.ch12.aidl. IMyservice" />
    </intent-filter>
</service>

```

其中“mobile.android.ch12. IMyservice”是客户端用于访问 AIDL 服务的 ID。

下面来编写客户端的调用代码。首先新建一个 Eclipse Android 工程（ch12\_aidlclient），并将自动生成的 IMyService.java 文件连同包目录一起复制到 ch12\_aidlclient 工程的 src 目录中，如图 12.7 所示。



▲ 图 12.7 IMyService.java 文件在 ch12\_aidlclient 工程中的位置

调用 AIDL 服务首先要绑定服务，然后才能获得服务对象，代码如下：

```
package mobile.android.ch12.aidlclient;

import net.blogjava.mobile.aidl.IMyService;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Main extends Activity implements OnClickListener
{
    private IMyService myService = null;
    private Button btnInvokeAIDLService;
    private Button btnBindAIDLService;
    private TextView textView;
    private ServiceConnection serviceConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            // 获得服务对象
            myService = IMyService.Stub.asInterface(service);
            btnInvokeAIDLService.setEnabled(true);
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
        }
    };
    @Override
    public void onClick(View view)
    {
        switch (view.getId())
        {
            case R.id.btnBindAIDLService:
                // 绑定 AIDL 服务
                bindService(new Intent("mobile.android.ch12.aidl.IMyService"),
                           serviceConnection, Context.BIND_AUTO_CREATE);
                break;
            case R.id.btnInvokeAIDLService:
                try
                {
                    textView.setText(myService.getValue()); // 调用服务端的 getValue 方法
                }
                catch (Exception e)
                {
                }
        }
    }
}
```

```

        break;
    }
}

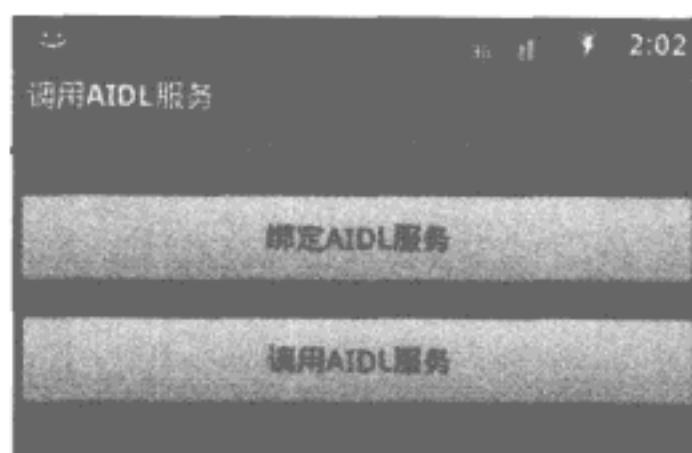
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    btnInvokeAIDLService = (Button) findViewById(R.id.btnInvokeAIDLService);
    btnBindAIDLService = (Button) findViewById(R.id.btnBindAIDLService);
    btnInvokeAIDLService.setEnabled(false);
    textView = (TextView) findViewById(R.id.textview);
    btnInvokeAIDLService.setOnClickListener(this);
    btnBindAIDLService.setOnClickListener(this);
}
}

```

在编写上面代码时应注意如下几点。

- 使用 `bindService` 方法来绑定 AIDL 服务。其中需要使用 `Intent` 对象指定 AIDL 服务的 ID，也就是`<action>`标签中 `android:name` 属性的值。
- 在绑定时需要一个 `ServiceConnection` 对象。创建 `ServiceConnection` 对象的过程中如果绑定成功，系统会调用 `ServiceConnection.onServiceConnected` 方法，通过该方法的 `service` 参数值可获得 AIDL 服务对象。

首先运行 AIDL 服务程序，然后运行客户端程序，单击“绑定 AIDL 服务”按钮，如果绑定成功，“调用 AIDL 服务”按钮会变为可选状态；单击这个按钮，会输出 `getValue` 方法的返回值，如图 12.8 所示。



▲ 图 12.8 调用 AIDL 服务的客户端程序

#### 12.2.4 传递复杂数据的 AIDL 服务

工程目录: `src\ch12\ch12_complextpeaidl`、`src\ch12\ch12_complextpeaidlclient`

AIDL 服务只支持有限的数据类型，因此，如果用 AIDL 服务传递一些复杂的数据就需要做更一步处理。AIDL 服务支持的数据类型如下。

- Java 的简单类型 (`int`、`char`、`boolean` 等)。不需要导入 (`import`)。
- `String` 和 `CharSequence`。不需要导入 (`import`)。
- `List` 和 `Map`。但要注意，`List` 和 `Map` 对象的元素类型必须是 AIDL 服务支持的数据类型。不需要导入 (`import`)。

- AIDL 自动生成的接口。需要导入（import）。
- 实现 android.os.Parcelable 接口的类。需要导入（import）。

传递不需要 import 的数据类型值的方式相同，传递一个需要 import 的数据类型值（例如，实现 android.os.Parcelable 接口的类）的步骤略显复杂。除了要建立一个实现 android.os.Parcelable 接口的类外，还需要为这个类单独建立一个 aidl 文件，并使用 `Parcelable` 关键字进行定义。具体的实现步骤如下：

- (1) 建立一个 IMyService.aidl 文件，并输入如下代码：

```
package mobile.android.ch12.complex.type.aidl;
import mobile.android.ch12.complex.type.aidl.Product;
interface IMyService
{
    Map getMap(in String country, in Product product);
    Product getProduct();
}
```

在编写上面代码时要注意如下几点。

- `Product` 是一个实现 `android.os.Parcelable` 接口的类，需要使用 `import` 导入这个类。
- 如果方法的类型是非简单类型，例如，`String`、`List` 或自定义的类，需要使用 `in`、`out` 或 `inout` 进行修饰，其中 `in` 表示这个值被客户端设置；`out` 表示这个值被服务端设置；`inout` 表示这个值既被客户端设置，又被服务端设置。

- (2) 编写 `Product` 类。该类是用于传递的数据类型，代码如下：

```
package .mobile.android.ch12.complex.type.aidl;

import android.os.Parcel;
import android.os.Parcelable;

public class Product implements Parcelable
{
    private int id;
    private String name;
    private float price;
    public static final Parcelable.Creator<Product> CREATOR = new Parcelable.Creator<Product>()
    {
        public Product createFromParcel(Parcel in)
        {
            return new Product(in);
        }

        public Product[] newArray(int size)
        {
            return new Product[size];
        }
    };
    public Product()
    {
    }
    private Product(Parcel in)
```

```

    {
        readFromParcel(in);
    }
    @Override
    public int describeContents()
    {
        return 0;
    }
    public void readFromParcel(Parcel in)
    {
        id = in.readInt();
        name = in.readString();
        price = in.readFloat();
    }
    @Override
    public void writeToParcel(Parcel dest, int flags)
    {
        dest.writeInt(id);
        dest.writeString(name);
        dest.writeFloat(price);
    }
    // 此处省略了属性的 getter 和 setter 方法
    ...
}

```

在编写 Product 类时应注意如下几点。

- Product 类必须实现 android.os.Parcelable 接口。该接口用于序列化对象。在 Android 中之所以使用 Parcelable 接口序列化，而不是 java.io.Serializable 接口，是因为 Google 在开发 Android 时发现 Serializable 序列化的效率并不高，因此，特意提供了一个 Parcelable 接口来序列化对象。
- 在 Product 类中必须有一个静态常量，常量名必须是 CREATOR，而且 CREATOR 常量的数据类型必须是 Parcelable.Creator。
- 在 writeToParcel 方法中需要将要序列化的值写入 Parcel 对象。

(3) 建立一个 Product.aidl 文件，并输入如下内容：

```
parcelable Product;
```

(4) 编写一个 MyService 类，代码如下：

```

package mobile.android.ch12.complex.type.aidl;

import java.util.HashMap;
import java.util.Map;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;
// AIDL 服务类
public class MyService extends Service
{
    public class MyServiceImpl extends IMyService.Stub
    {

```



```

@Override
public Product getProduct() throws RemoteException
{
    Product product = new Product();
    product.setId(1234);
    product.setName("汽车");
    product.setPrice(31000);
    return product;
}
@Override
public Map getMap(String country, Product product) throws RemoteException
{
    Map map = new HashMap<String, String>();
    map.put("country", country);
    map.put("id", product.getId());
    map.put("name", product.getName());
    map.put("price", product.getPrice());
    map.put("product", product);
    return map;
}
@Override
public IBinder onBind(Intent intent)
{
    return new MyServiceImpl();
}
}

```

(5) 在 AndroidManifest.xml 文件中配置 MyService 类，代码如下：

```

<service android:name=".MyService" >
    <intent-filter>
        <action android:name="mobile.android.ch12.complex.type.aidl.IMyService" />
    </intent-filter>
</service>

```

在客户端调用 AIDL 服务的方法与 12.2.3 小节介绍的方法相同，首先将 IMyService.java 和 Product.java 文件复制到客户端工程（ch12\_complextypeaidlclient），然后绑定 AIDL 服务，并获得 AIDL 服务对象，最后调用 AIDL 服务中的方法。完整的客户端代码如下：

```

package mobile.android.ch12.complex.type.aidlclient;

import net.blogjava.mobile.complex.type.aidl.IMyService;
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

```

## Android 开发权威指南

```
import android.widget.TextView;

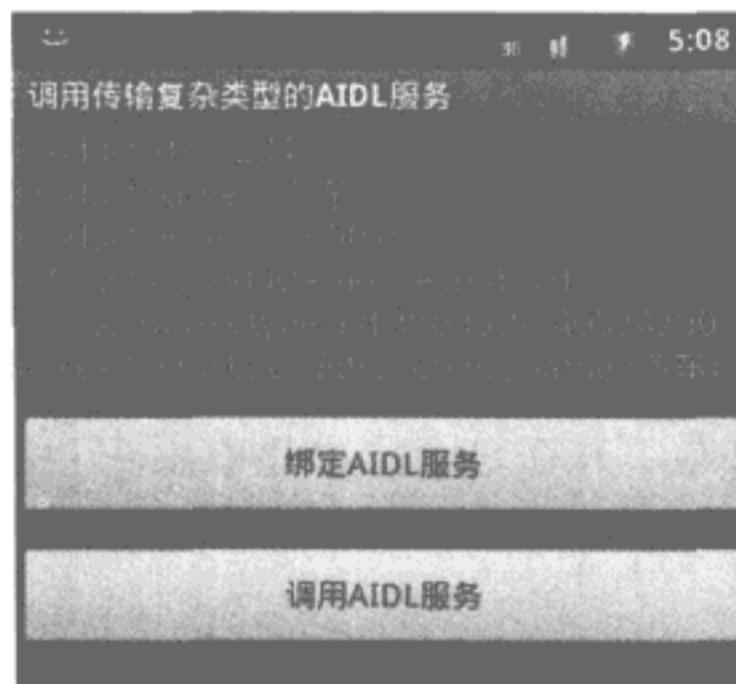
public class Main extends Activity implements OnClickListener
{
    private IMyService myService = null;
    private Button btnInvokeAIDLService;
    private Button btnBindAIDLService;
    private TextView textView;
    private ServiceConnection serviceConnection = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            // 获得 AIDL 服务对象
            myService = IMyService.Stub.asInterface(service);
            btnInvokeAIDLService.setEnabled(true);
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
        }
    };
    @Override
    public void onClick(View view)
    {
        switch (view.getId())
        {
            case R.id.btnBindAIDLService:
                // 绑定 AIDL 服务
                bindService(new Intent("net.blogjava.mobile.complex.type.aidl.IMyService"),
                           serviceConnection, Context.BIND_AUTO_CREATE);
                break;
            case R.id.btnInvokeAIDLService:
                try
                {
                    String s = "";
                    // 调用 AIDL 服务中的方法
                    s = "Product.id = " + myService.getProduct().getId() + "\n";
                    s += "Product.name = " + myService.getProduct().getName() + "\n";
                    s += "Product.price = " + myService.getProduct().getPrice() + "\n";
                    s += myService.getMap("China", myService.getProduct()).toString();
                    textView.setText(s);
                }
                catch (Exception e)
                {
                }
                break;
        }
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```

super.onCreate(savedInstanceState);
setContentView(R.layout.main);
btnInvokeAIDLService = (Button) findViewById(R.id.btnInvokeAIDLService);
btnBindAIDLService = (Button) findViewById(R.id.btnBindAIDLService);
btnInvokeAIDLService.setEnabled(false);
textView = (TextView) findViewById(R.id.textview);
btnInvokeAIDLService.setOnClickListener(this);
btnBindAIDLService.setOnClickListener(this);
}
}

```

首先运行服务端程序，然后运行客户端程序，单击“绑定 AIDL 服务”按钮，待成功绑定后，单击“调用 AIDL 服务”按钮，会输出如图 12.9 所示的内容。



▲ 图 12.9 调用传递复杂数据的 AIDL 服务

## 12.2.5 AIDL 与来电自动挂断

**工程目录:** src\ch12\ch12\_call\_aidl

虽然可以通过 Activity Action 来拨打电话，但使用常规的方法无法挂断电话。不过根据 7.3.2 小节介绍的技术为我们提供了一种通过程序挂断电话的方法。虽然这种方法并未直接调用服务，但却使用了 AIDL 文件自动生成接口。

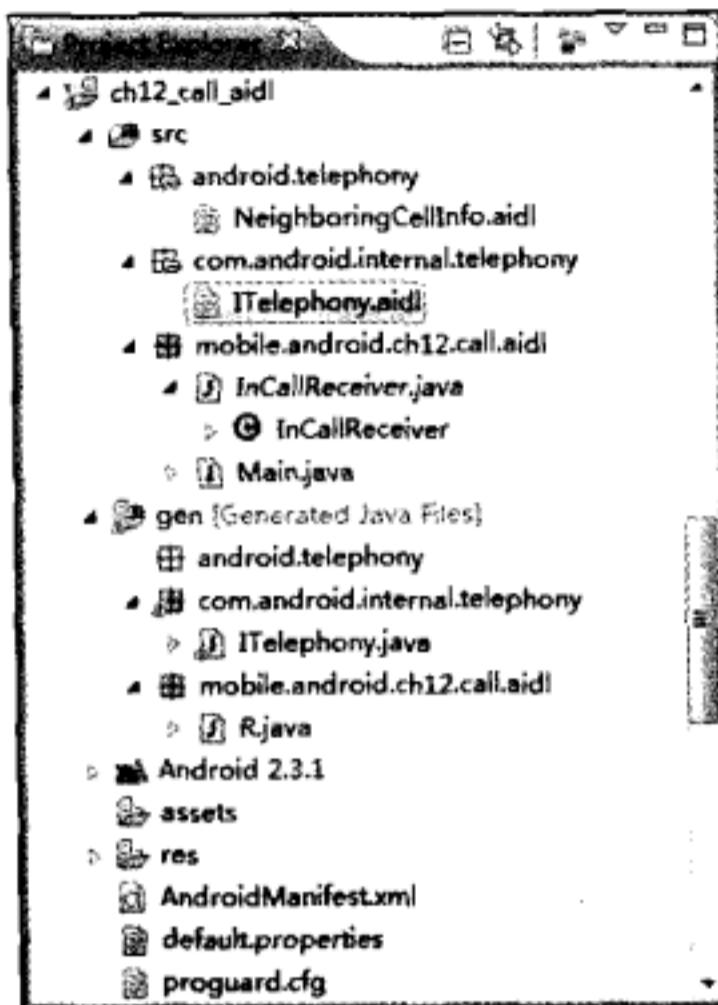
在 Android SDK 源代码中可以找到如下的接口。

com.android.internal.telephony.ITelephony

这个接口在外部是无法访问的，只有将程序嵌入到 Android SDK 内部才能访问到，这个接口提供了一个 endCall 方法可以挂断电话。现在就想办法来调用 Itelephony.endCall 方法。

尽管不能直接访问 ITelephony 接口，但我们发现在 TelephonyManager 类中有一个 getITelephony 方法可以返回一个 ITelephony 对象。不过 getITelephony 是 private 方法，但这不要紧，我们可以通过 Java 的反射技术来调用该方法。

在调用 getITelphony 方法获得 ITelephone 对象之前，先在 Android SDK 的源代码中找到 NeighboringCellInfo.aidl 和 ITelephony.aidl 文件，并将这两个文件连同其所在的包复制到 ch12\_call\_aidl 工程的源代码目录中，如图 12.10 所示。



▲ 图 12.10 aidl 文件的位置

ADT 会根据 Itelephone.aidl 文件自动生成 ITelephony.java 文件，如图 12.10 中 gen 目录所示。下面编写一个接收来电的广播接收器，并在这个广播接收器中自动挂断指定电话号的来电。

```
package mobile.android.ch12.call.aidl;

import java.lang.reflect.Method;
import com.android.internal.telephony.ITelephony;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.TelephonyManager;
import android.widget.Toast;

public class InCallReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        TelephonyManager tm = (TelephonyManager) context
            .getSystemService(Service.TELEPHONY_SERVICE);
        switch (tm.getCallState())
        {
            case TelephonyManager.CALL_STATE_RINGING: // 响铃
                // 获得来电的电话号
                String incomingNumber = intent.getStringExtra("incoming_number");
                // 当来电是 12345678 时挂断电话
                if ("12345678".equals(incomingNumber))
                {
                    try
                    {
                        TelephonyManager telephonyManager = (TelephonyManager) context
                            .getSystemService(Service.TELEPHONY_SERVICE);
                        telephonyManager.endCall();
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
        }
    }
}
```

```
        .getSystemService(Service.TELEPHONY_SERVICE);
        // 获得 TelephonyManager 的 Class 对象
        Class<TelephonyManager> telephonyManagerClass = TelephonyManager.
        class;
        // 获得 TelephonyManager.getITelephony 方法的 Method 对象
        Method telephonyMethod = telephonyManagerClass
            .getDeclaredMethod("getITelephony", (Class[]) null);
        // 允许访问 private 方法
        telephonyMethod.setAccessible(true);
        // 调用 getITelephony 方法返回 ITelephony 对象
        ITelephony telephony = (com.android.internal.telephony.ITelephony)
        telephonyMethod
            .invoke(telephonyManager, (Object[]) null);
        // 挂断电话
        telephony.endCall();
    }
    catch (Exception e)
    {
        Toast.makeText(context, e.getMessage(), Toast.LENGTH_LONG).show();
    }
}
break;
}
}
```

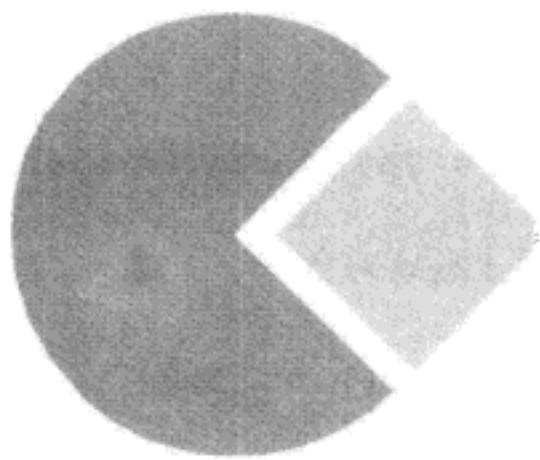
InCallReceiver 类需要在 AndroidManifest.xml 文件中设置如下两个权限。

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

读者可以模拟拨打电话“12345678”，模拟器刚一接到电话，就会自动挂断。

## 12.3 | 小结

本章主要介绍了 Android 系统中的服务（Service）技术，Service 是 Android 中 4 个应用程序组件之一。服务除了可以在程序内部调用外，还可以使用 AIDL 服务实现跨应用的调用，这种方式有些类似 Windows 中的 COM 组件，其中的 AIDL 文件应用很广泛，可以利用 AIDL 文件自动生成接口文件，并可以将相应的对象转换成指定接口，这大大方便了服务的调用。



## 第 13 章 做好应用桥梁——网络与通信

网络实际上并不属于 Android 的范畴，但由于 Android 使用 Java 作为主要的开发语言，而且 Android 中的 Java 语言与 Java SE、Java EE 中使用的 Java 语言在网络方面非常相似，因此，学习如何使用 Java 语言访问网络资源对开发 Android 网络程序有非常大的帮助。Java 语言可以很好地支持使用 TCP 读取网络数据，网络通信分为客户端 Socket 和服务端 Socket。本章将逐一讨论这些常用的 Java 网络技术。除此之外，本章还介绍了如何通过蓝牙技术在手机之间进行通信。由于蓝牙通信使用了与 Socket 类似的方式，因此，熟练掌握 Java 网络技术会使学习蓝牙通信变得更容易。

### 13.1 WebView 控件

本节将介绍一个可以浏览 Web 页的控件：WebView。WebView 控件可以在自己的应用程序中显示本地或 Internet 上的网页。WebView 是一个使用 WebKit 引擎的浏览器控件，因此，可以将 WebView 当成一个完整的浏览器使用。WebView 不仅支持 HTML、CSS 等静态元素，还支持 JavaScript，而且在 JavaScript 中还可以调用 Java 的方法。

#### 13.1.1 用 WebView 控件浏览网页

**工程目录：**src\ch13\ch13\_browser

浏览网页是 WebView 控件最基本的功能，通过 WebView.loadUrl 方法可直接装载任何有效的网址，例如下面的代码将显示 <http://nokiaguy.blogjava.net> 页面的内容。

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.loadUrl("http://nokiaguy.blogjava.net");
```

WebView 控件不仅可以浏览 Internet 上的网页，也可以浏览本地的网页文件或任何 WebView 支持的文件，代码如下：

```
webView.loadUrl("file:///sdcard/images.jpg");
webView.loadUrl("file:///sdcard/test.html");
```

除了可以浏览网页外，WebView 控件也和大多数浏览器一样，可以缓存浏览历史页面，并使用如下代码向后和向前浏览历史页面：

```
webView.goBack(); // 向后浏览历史页面
```

```
webView.goBack(); // 向前浏览历史页面
```

如果想清除缓存内容，可以使用 clearCache 方法，代码如下：

```
webView.clearCache();
```

下面给出一个使用 WebView 控件浏览网页的例子。本例的功能是在 EditText 控件中输入要浏览的网址，然后单击网址输入框右侧的按钮，就会在 WebView 控件中显示相应的页面，效果如图 13.1 所示。当多次浏览网页后，可以通过如图 13.2 所示的选项菜单向后和向前浏览历史网页。



▲ 图 13.1 手机浏览器



▲ 图 13.2 手机浏览器的选项菜单

下面来看一下查询按钮的 onClick 方法的代码。

```
public void onClick(View view)
{
    String url = etAddress.getText().toString();
    if (URLUtil.isNetworkUrl(url))
        webView.loadUrl(url);
    else
        Toast.makeText(this, "输入的网址不正确.", Toast.LENGTH_LONG).show();
}
```

在上面的代码中首先使用 URLUtil.isNetworkUrl 方法来判断用户输入的 URL 是否有效，如果用户输入了无效的 URL，系统会显示一个 Toast 信息框来提醒用户输入正确的 URL。

通过选项菜单的两个菜单项可以向后和向前浏览历史页面，菜单项的 onMenuItemClick 事件方法的代码如下：

```
public boolean onMenuItemClick(MenuItem item)
{
    switch (item.getItemId())
    {
        case 0:
```

```

        // 向后 (back) 浏览页面
        webView.goBack();
        break;
    case 1:
        // 向前 (Forward) 浏览页面
        webView.goForward();
        break;
    }
    return false;
}

```

**注意**

由于 WebView 需要访问 Internet 资源，所以需要在 AndroidManifest.xml 文件中设置“`android.permission.INTERNET`”权限。

### 13.1.2 用 WebView 控件装载 HTML 代码

**工程目录:** `src\ch13\ch13_loadhtml`

WebView 控件不仅可以通过 URL 装载网页，也可以直接装载 HTML 代码。WebView 类有两个方法可以装载 HTML 代码，这两个方法的定义如下：

```

public void loadData(String data, String mimeType, String encoding);
public void loadDataWithBaseUrl(String baseUrl, String data,
    String mimeType, String encoding, String failUrl)

```

其中 `loadData` 方法的参数含义如下。

- `data`: HTML 代码。
- `mimeType`: Mime 类型，一般为 `text/html`。
- `encoding`: HTML 代码的编码，例如 `GBK`、`utf-8`。

`loadDataWithBaseUrl` 方法的参数含义如下。

- `baseUrl`: 获得相对路径的根 URL，如果设为 `null`，默认值是 `about:blank`。
- `failUrl`: 如果 HTML 代码装载失败或为 `null` 时，WebView 控件会装载这个参数指定的 URL。
- 其他的参数与 `loadData` 方法的参数含义相同。

虽然 `loadData` 和 `loadDataWithBaseUrl` 方法都可以装载 HTML 代码，但经笔者测试，`loadData` 方法在装载包含中文的 HTML 代码时会出现乱码，而 `loadDataWithBaseUrl` 方法没有任何问题。因此笔者建议使用 `loadDataWithBaseUrl` 方法来装载 HTML 代码。

WebView 控件在默认情况下不支持 JavaScript，为了使 WebView 控件支持 JavaScript，需要使用 `setJavaScriptEnabled` 和 `setWebChromeClient` 方法进行设置，其中 `setWebChromeClient` 方法用来设置 JavaScript 处理器。本例中使用 `loadDataWithBaseUrl` 方法装载显示一个表格的 HTML 代码，在这些代码中使用了 JavaScript，因此，需要将 WebView 控件的 JavaScript 功能打开，代码如下：

```

WebView webView = (WebView) findViewById(R.id.webview);
String html = "<html>" +
    "<body>" +
    "图书封面<br>" +
    "<table width='200' border='1' >"
```

```

+ "<tr>
+ <td><a onclick='alert(\"《Android/OPhone 开发完全讲义》\")'><img style='margin:
10px' src='file:///android_asset/android_ophone_shupi.jpg' width='100' /></a></td>"
+ <td><a onclick='alert(\"《人人都玩开心网》\")'><img style='margin:10px' src=
'http://www.blogjava.net/images/blogjava_net/nokiaguy/10113411c.jpg' width=
'100' /></td>
+ "</tr>
+ <tr>
+ <td><img style='margin:10px' src='http://images.china-pub.com/ebook25001-
30000/27518/zcover.jpg' width='100' /></td>
+ <td><img style='margin:10px' src='http://images.china-pub.com/ebook30001-
35000/34838/zcover.jpg' width='100' /></td>
+ "</tr>" + "</table>" + "</body>" + "</html>";
// 开始装载 HTML 代码
webView.loadDataWithBaseUrl("图书名", html, "text/html", "utf-8", null);
// 打开 JavaScript 功能
webView.getSettings().setJavaScriptEnabled(true);
// 设置处理 JavaScript 的引擎
webView.setWebChromeClient(new WebChromeClient());

```

在上面的 HTML 代码中除了显示几个 Internet 图像资源外，还显示了一个本地的图像，该图像文件位于工程中的 assets 目录中。如果要在 HTML 代码中显示 apk 文件中的 assets 目录中的图像，必须以“file://android\_asset”开头，后面跟路径和文件名。要注意，“android\_asset”中的 asset 后面没有“s”。

运行本例后，显示的效果如图 13.3 所示。单击页面上的图像，会执行 JavaScript 代码（显示一个对话框），效果如图 13.4 所示。



▲ 图 13.3 装载 HTML 代码



▲ 图 13.4 单击图像显示对话框

## 13.2 访问 HTTP 资源

HTTP 是 Internet 中广泛使用的协议，几乎所有的计算机语言和 SDK 都会不同程度地支持 HTTP，而以网络著称的 Google 公司自然也会使 Android SDK 拥有强大的 HTTP 访问能力。在 Android SDK 中可以采用多种方式使用 HTTP，例如 HttpURLConnection、HttpGet、HttpPost 等。本节将介绍如何利用这些技术来访问 HTTP 资源。

### 13.2.1 提交 HTTP GET 和 HTTP POST 请求

工程目录：src\ch13\ch13\_httpgetpost、src\ch13\querybooks

本节将介绍 Android SDK 集成的 Apache HttpClient 模块。要注意的是，这里的 Apache HttpClient 模块是 HttpClient 4.0(`org.apache.http.*`)，而不是 Jakarta Commons HttpClient 3.x(`org.apache.commons.httpclient.*`)。

在 HttpClient 模块中涉及两个重要的类：`HttpGet` 和 `HttpPost`。这两个类分别用来提交 HTTP GET 和 HTTP POST 请求。为了测试本小节的例子，需要先编写一个 Servlet 程序，用来接收 HTTP GET 和 HTTP POST 请求。关于 Servlet 程序的源代码，读者可以查看 querybooks 工程中的源文件。在运行本例之前，需要先在计算机上安装 Tomcat，并将 querybooks 工程直接复制到<Tomcat 安装目录>\webapps 目录即可，然后启动 Tomcat。在浏览器地址栏中输入如下 URL：

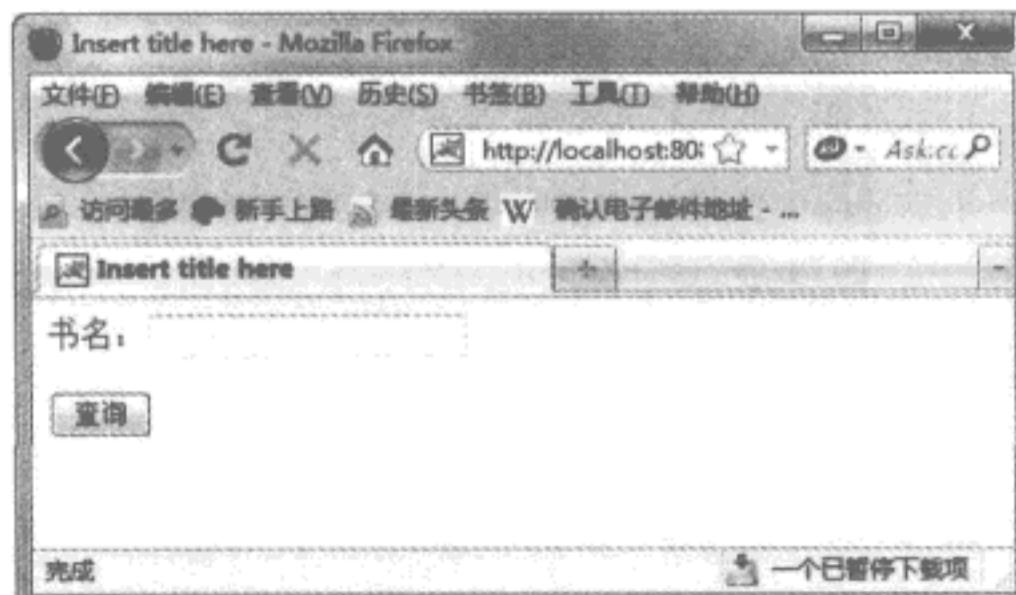
http://localhost:8080/querybooks/query.jsp

如果出现如图 13.5 所示的页面，说明 querybooks 已经安装成功。

在 querybooks 工程中有一个 `QueryServlet` 类，访问这个类的 URL 如下：

http://192.168.17.81:8080/querybooks/QueryServlet?bookname=development

其中“192.168.17.81”是 PC 的 IP 地址，`bookname` 是 `QueryServlet` 的请求参数，表示图书名，通过该参数来查询图书信息。在如图 13.5 所示的页面中的文本框内输入“development”，然后单击“查询”按钮，页面会以 HTTP POST 方式向 `QueryServlet` 提交请求信息，如果成功提交，将显示如图 13.6 所示的内容。



▲ 图 13.5 querybooks 的测试页面



▲ 图 13.6 返回的响应信息

现在我们要通过 `HttpGet` 和 `HttpPost` 向 `QueryServlet` 提交请求信息，并将返回的结果显示在 `TextView` 控件中。无论是使用 `HttpGet`，还是使用 `HttpPost`，都必须通过如下 3 步来访问 HTTP 资源。

(1) 创建 `HttpGet` 或 `HttpPost` 对象，将要请求的 URL 通过构造方法传入 `HttpGet` 或 `HttpPost` 对象。

(2) 使用 `DefaultHttpClient.execute` 方法发送 HTTP GET 或 HTTP POST 请求，并返回 `HttpResponse` 对象。

(3) 通过 `HttpResponse.getEntity` 方法返回响应信息，并进行相应的处理。

如果使用 `HttpPost` 方法提交 HTTP POST 请求，还需要使用 `HttpPost.setEntity` 方法设置请求参数。

本例使用了两个按钮分别提交 HTTP GET 和 HTTP POST 请求，并从 `EditText` 控件中获得请求参数（`bookname`）值，最后将返回的结果显示在 `TextView` 控件中。两个按钮共用一个 `onClick` 事件方法，代码如下：

```
public void onClick(View view)
{
    // 读者需要将本例中的 IP 地址换成自己机器的 IP 地址
    String url = "http://192.168.17.81:8080/querybooks/QueryServlet";
    TextView tvQueryResult = (TextView) findViewById(R.id.tvQueryResult);
    EditText etBookName = (EditText) findViewById(R.id.etBookName);
    HttpResponse httpResponse = null;
    try
    {
        switch (view.getId())
        {
            // 提交 HTTP GET 请求
            case R.id.btnGetQuery:
                // 向 url 添加请求参数
                url += "?bookname=" + etBookName.getText().toString();
                // 第 1 步：创建 HttpGet 对象
                HttpGet httpGet = new HttpGet(url);
                // 第 2 步：使用 execute 方法发送 HTTP GET 请求，并返回 HttpResponse 对象
                httpResponse = new DefaultHttpClient().execute(httpGet);
                // 判断请求响应状态码，状态码为 200 表示服务端成功响应了客户端的请求
                if (httpResponse.getStatusLine().getStatusCode() == 200)
                {
                    // 第 3 步：使用 getEntity 方法获得返回结果
                    String result = EntityUtils.toString(httpResponse.getEntity());
                    // 去掉返回结果中的 "\r" 字符，否则会在结果字符串后面显示一个小方格
                    tvQueryResult.setText(result.replaceAll("\r", ""));
                }
                break;
            // 提交 HTTP POST 请求
            case R.id.btnPostQuery:
                // 第 1 步：创建 HttpPost 对象
                HttpPost httpPost = new HttpPost(url);
                // 设置 HTTP POST 请求参数必须用 NameValuePair 对象
```

```

List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new BasicNameValuePair("bookname", etBookName
    .getText().toString()));
// 设置 HTTP POST 请求参数
httpPost.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));
// 第 2 步：使用 execute 方法发送 HTTP POST 请求，并返回 HttpResponse 对象
httpResponse = new DefaultHttpClient().execute(httpPost);
if (httpResponse.getStatusLine().getStatusCode() == 200)
{
    // 第 3 步：使用 getEntity 方法获得返回结果
    String result = EntityUtils.toString(httpResponse.getEntity());
    // 去掉返回结果中的 "\r" 字符，否则会在结果字符串后面显示一个小方格
    tvQueryResult.setText(result.replaceAll("\r", ""));
}
break;
}
catch (Exception e)
{
    tvQueryResult.setText(e.getMessage());
}
}
}

```

运行本例，在文本编辑框中输入“development”，并单击“GET 查询”和“POST 查询”按钮，会在屏幕下方显示如图 13.7 和图 13.8 所示的信息。



▲ 图 13.7 Get 请求查询结果



▲ 图 13.8 Post 请求查询结果

### 13.2.2 HttpURLConnection 类

`java.net.HttpURLConnection` 类是另外一种访问 HTTP 资源的方式，`HttpURLConnection` 类可以完全取代 `HttpGet` 和 `HttpPost` 类。使用 `HttpURLConnection` 访问 HTTP 资源的步骤如下：

(1) 使用 `java.net.URL` 封装 HTTP 资源的 url，并使用 `openConnection` 方法获得 `HttpURLConnection` 对象，代码如下：

```

URL url = new URL("http://www.blogjava.net/nokiaguy/archive/2009/12/14/305890.html");
HttpURLConnection httpURLConnection = (HttpURLConnection) url.openConnection();

```

(2) 设置请求方法，例如 GET、POST 等，代码如下：

```
| httpURLConnection.setRequestMethod("POST");
```

要注意的是, `setRequestMethod` 方法的参数值必须大写, 例如 GET、POST 等。

(3) 设置输入输出及其他开关。如果要读取 HTTP 资源或向服务端上传数据, 需要使用如下代码进行设置:

```
// 读取 HTTP 资源, 需要将 setDoInput 方法的参数值设为 true  
httpURLConnection.setDoInput(true);  
// 上传数据, 需要将 setDoOutput 方法的参数值设为 true  
httpURLConnection.setDoOutput(true);
```

`HttpURLConnection` 类还包含更多的选项, 例如, 使用下面的代码可以禁止 `HttpURLConnection` 使用缓存。

```
| httpURLConnection.setUseCaches(false);
```

(4) 设置 HTTP 请求头。在很多情况下, 要根据实际情况设置一些 HTTP 请求头, 例如下面的代码设置了 `Charset` 请求头的值为 UTF-8。

```
| httpURLConnection.setRequestProperty("Charset", "UTF-8");
```

(5) 输入和输出数据。这一步是对 HTTP 资源的读写操作, 也就是通过 `InputStream` 和 `OutputStream` 读取和写入数据。下面的代码获得了 `InputStream` 对象和 `OutputStream` 对象。

```
| InputStream is = httpURLConnection.getInputStream();  
OutputStream os = httpURLConnection.getOutputStream();
```

至于是先读取还是先写入数据, 需要根据具体情况而定。

(6) 关闭输入输出流。虽然关闭输入输出流并不是必需的, 在应用程序结束后, 输入输出流会自动关闭, 但显式关闭输入输出流是一个好习惯。关闭输入输出流的代码如下:

```
| is.close();  
os.close();
```

### 13.2.3 上传文件

工程目录: `src\ch13\ch13_uploadfile`、`src\ch09\upload`

本例使用 `HttpURLConnection` 实现了一个上传文件的应用程序, 该程序可以将手机上的文件上传到服务端。本例所使用的服务端程序在 `src\ch13\upload` 目录中, 读者可以将 `upload` 目录直接复制到<Tomcat 安装目录>\webapps 目录中, 然后启动 Tomcat, 在浏览器地址栏中输入如下 URL:

```
| http://localhost:8080/upload/upload.jsp
```

如果在浏览器中显示如图 13.9 所示的页面, 说明服务端程序已经安装成功。这个服务端程序负责接收客户端上传的文件, 并将成功上传的文件保存在 D:\upload 目录中, 如果该目录不存在, 系统会自动创建该目录。读者可以使用如图 13.9 所示的页面上传一个文件, 观察一下效果。



▲ 图 13.9 上传文件的页面

下面来实现 Android 版的文件上传客户端。在这个例子中使用了一个自定义的文件浏览器控件，该控件的详细实现读者可以查看本书提供的源代码。该控件用来浏览 SD 卡中的文件和目录，浏览文件的效果如图 13.10 所示，当单击一个文件时，系统会上传该文件，上传成功后的效果如图 13.11 所示。读者可以在 D:\upload 目录看到上传的文件。



▲ 图 13.10 浏览 SD 卡中的文件



▲ 图 13.11 成功上传文件

实现本例的关键是了解文件上传的原理。为了分析文件上传的原理，笔者使用了 HttpAnalyzer 来截获如图 13.9 所示的页面上传文件的 HTTP 请求信息。从“stream”标签页可以看到原始的 HTTP 请求信息，如图 13.12 所示。

从图 13.12 可以看出，上传文件的 HTTP 请求信息分为如下几部分。

- 分界符。由两部分组成，分别是两个连字符（--）和一个任意字符串。使用浏览器上传文件一般为“-----数字”，前两个连字符是分界符，分界符为单独一行。
- 上传文件的相关信息。这些信息包括请求参数名、上传文件名、文件类型，但并不仅限于此，例如 Content-Disposition: form-data; name="file"; filename="abc.jpg"。
- 上传文件的内容。字节流形式。
- 文件全部上传后的结束符。这个符号在图 13.12 中并没有显示出来，当上传的文件是最后

一个时，在 HTTP 请求信息的结尾就会出现这个符号字符串。结束符和分界符类似，只是在分界符后面再加两个连字符，例如，“-----218813199810322--”就是一个结束符。

▲ 图 13.12 上传文件的 HTTP 请求信息

当单击如图 13.10 所示列表中的某个文件时，会调用 SD 卡浏览器控件的 `onFileItemClick` 事件方法，在该方法中负责上传当前单击的文件，代码如下：

```

dos.writeBytes(twoHyphens + boundary + end);
// 设置与上传文件相关的信息
dos.writeBytes("Content-Disposition: form-data; name=\"file\"; filename=\""
+ filename.substring(filename.lastIndexOf("/") + 1) + "\"" + end);
// 在上传文件信息与文件内容之间必须有一个空行
dos.writeBytes(end);
// 开始上传文件
FileInputStream fis = new FileInputStream(filename);
byte[] buffer = new byte[8192];           // 8k
int count = 0;
// 读取文件内容，并写入 OutputStream 对象
while ((count = fis.read(buffer)) != -1)
{
    dos.write(buffer, 0, count);
}
fis.close();
// 新起一行
dos.writeBytes(end);
// 设置结束符号（在分界符后面加两个连字符）
dos.writeBytes(twoHyphens + boundary + twoHyphens + end);
dos.flush();
// 开始读取从服务端传过来的信息
InputStream is = httpURLConnection.getInputStream();
InputStreamReader isr = new InputStreamReader(is, "utf-8");
BufferedReader br = new BufferedReader(isr);
String result = br.readLine();
Toast.makeText(this, result, Toast.LENGTH_LONG).show();
dos.close();
is.close();
}
catch (Exception e)
{
}
}
}

```

在编写上面代码时应注意如下几点。

- 在本例中，分界符中的任意字符串使用了“\*\*\*\*\*”，而不是浏览器使用的“-----数字”。
- 分界符中的任意字符串必须在 Content-Type 请求头中指定，好让服务端可以识别多个上传文件的数据。
- 在上传文件信息与上传文件内容之间必须有一个空行。

### 13.3 客户端 Socket

网络编程分为客户端和服务端两部分，而 Socket 类是负责处理客户端通信的 Java 类。通过这个类可以连接到指定 IP 或域名的服务器上，并且可以和服务器交互数据。本节将详细讨论 Socket 类的使用，内容包括 Socket 类各式各样的连接方式、get 和 set 方法、连接过程中的超时以及关闭网络连接等。

### 13.3.1 连接服务器

在客户端可以通过两种方式来连接服务器，一种是通过 IP 的方式来连接服务器，而另外一种是通过域名方式来连接服务器。实际上这两种方式从本质上来看是一种方式，在底层客户端都是通过 IP 来连接服务器的。但这两种方式有一定的差异，如果通过 IP 方式来连接服务器，客户端只简单地根据 IP 进行连接，如果通过域名来连接服务器，客户端必须通过 DNS 将域名解析成 IP，然后再根据这个 IP 来进行连接。

通过 `Socket` 类连接服务器最常用的方法就是通过 `Socket` 类的构造函数将 IP 或域名以及端口号作为参数传入 `Socket` 对象中。`Socket` 类的构造方法有很多重载形式，在这一小节只讨论一种最简单的重载形式，定义如下：

```
| public Socket(String host, int port)
```

这个构造方法有两个参数：`host` 和 `port`，分别表示服务器名（IP 或域名）和端口号。下面的代码用于连接指定的服务器。

```
| Socket socket = new Socket("www.csdn.net", 80);
```

### 13.3.2 扫描服务器打开的端口

工程目录：src\ch13\ch13\_scan\_server\_ports

`Socket` 的一个非常有趣的应用就是通过不断连接服务器的各个端口来测试哪个端口已打开。本小节给出一个用于扫描服务器已打开端口（端口号扫描范围：1 至 1000）的例子。

`Socket` 类除了通过构造方法指定服务器地址和端口号外，还可以通过 `Socket.connect` 方法指定。`connect` 方法的定义如下：

```
| public void connect(SocketAddress remoteAddr, int timeout) throws IOException
```

`connect` 方法第一个参数的类型是 `SocketAddress`，用来设置服务器地址和端口号。第二个参数 `timeout` 表示连接超时，单位是毫秒。我们可以通过如下的代码来连接服务器。

```
| SocketAddress socketAddress = new InetSocketAddress("192.168.17.81", 80);  
| Socket socket = new Socket();  
| socket.connect(socketAddress, 50); // 如果超过 50 毫秒仍然没有连接上服务器，则会抛出异常
```

下面看看扫描服务器端口的完整代码。

```
package mobile.android.ch13.scan.server.ports;  
  
import java.net.InetSocketAddress;  
import java.net.Socket;  
import java.net.SocketAddress;  
import android.app.Activity;  
import android.os.Bundle;  
import android.os.Handler;  
import android.os.Message;  
import android.view.View;
```

## Android 开发权威指南

```

import android.widget.TextView;
import android.widget.Toast;

public class Main extends Activity
{
    private TextView tvPorts;
    private Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 将已打开的端口号显示在 TextView 控件上
            tvPorts.append(String.valueOf(msg.what) + ":ok\n");
            super.handleMessage(msg);
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvPorts = (TextView) findViewById(R.id.tvPorts);
    }
    // 用于扫描服务器端口的类
    class ScanPorts extends Thread
    {
        private int minPort, maxPort;
        // 指定要扫描的最小和最大端口
        public ScanPorts(int minPort, int maxPort)
        {
            this.minPort = minPort;
            this.maxPort = maxPort;
        }
        public void run()
        {
            // 开始扫描服务器端口
            for (int i = minPort; i <= maxPort; i++)
            {
                try
                {
                    Socket socket = new Socket();
                    // 192.168.17.81 是笔者机器上的 IP 地址，读者需要将其改成自己机器上的 IP 地址
                    SocketAddress socketAddress = new InetSocketAddress("192.168.17.81", i);
                    // 连接服务器
                    socket.connect(socketAddress, 50);
                    // 如果端口已打开，通过 Handler 将该端口显示在 TextView 控件上
                    handler.sendMessage(i);
                    socket.close();
                }
                catch (Exception e)
                {
                }
            }
            handler.post(new Runnable()

```

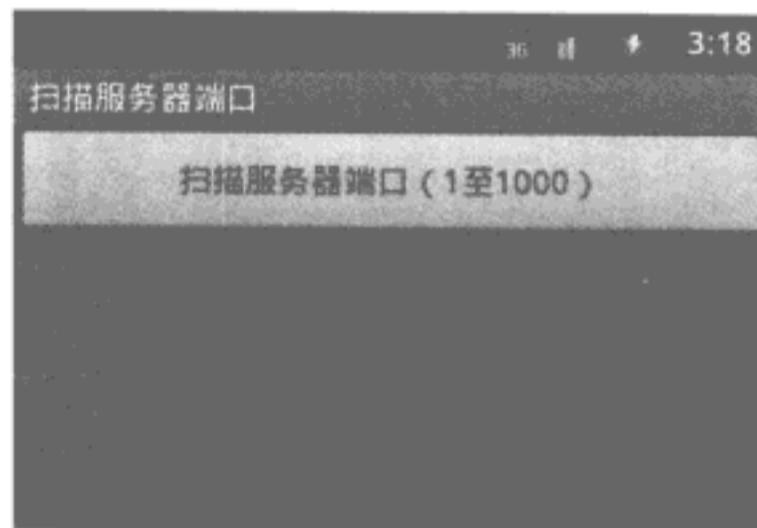
```

    {
        @Override
        public void run()
        {
            // 扫描完成
            Toast.makeText(Main.this, "扫描完成。", Toast.LENGTH_LONG).show();
        }
    });
}

public void onClick_Scan_Server_Ports(View view)
{
    Thread thread = new Thread(new ScanPorts(1, 1000));
    thread.start();
    Toast.makeText(this, "开始扫描", Toast.LENGTH_LONG).show();
}
}

```

上面的代码通过线程异步扫描服务器的端口号。如果扫描到某个端口号，会利用 Handler 将端口号显示在 TextView 控件上。运行本例，单击界面最上方的按钮，过一段时间，会不断有扫描到的端口显示在按钮下方，如图 13.13 所示。



▲ 图 13.13 扫描服务器已打开的端口

### 13.3.3 发送和接收数据

工程目录: src\ch13\ch13\_socket\_data\_transmit

Socket.getInputStream 和 Socket.getOutputStream 方法分别用来得到用于读取和写入数据的 InputStream 和 OutputStream 对象。在这里，InputStream 读取的是服务端向客户端发送过来的数据，而 OutputStream 允许客户端向服务端发送数据。

在编写实际的网络客户端程序时，是使用 getInputStream，还是使用 getOutputStream，以及先使用谁应根据具体的情况决定。下面的代码连接 51happyblog.com 的 80 端口（一般为 HTTP 协议使用的默认端口），并读取服务端的返回信息。

```

try
{
    Socket socket = new Socket("51happyblog.com", 80);
    // 向服务端程序发送数据
    OutputStream os = socket.getOutputStream();
    OutputStreamWriter osw = new OutputStreamWriter(os);

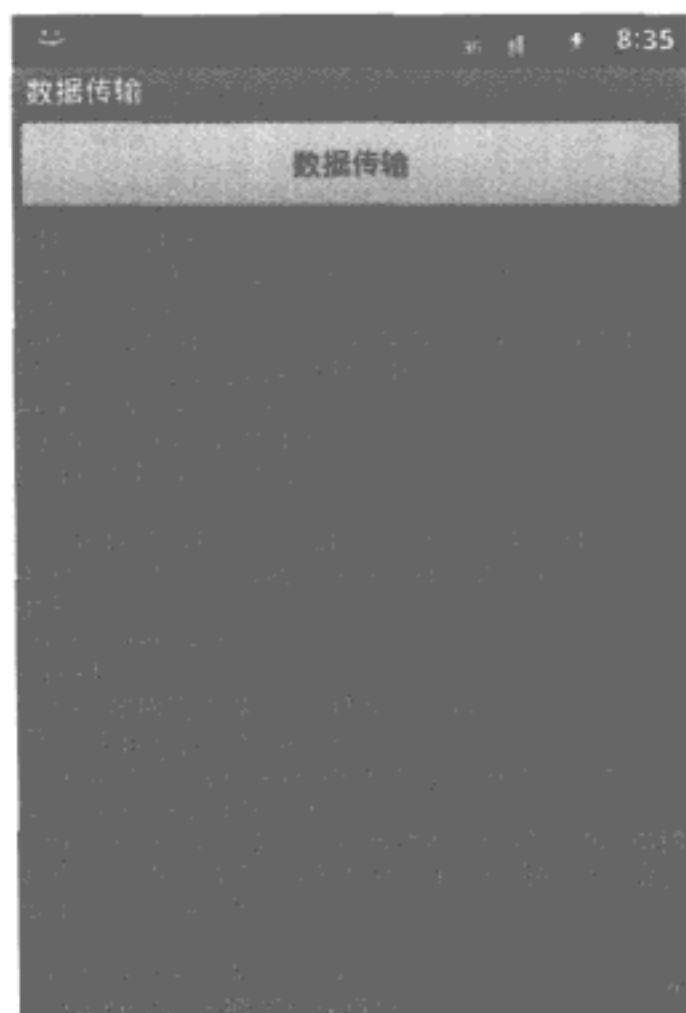
```

```

BufferedWriter bw = new BufferedWriter(osw);
bw.write("GET / HTTP/1.1\r\nHost:51happyblog.com\r\n\r\n");
bw.flush();
// 从服务端程序接收数据
InputStream is = socket.getInputStream();
InputStreamReader isr = new InputStreamReader(is);
BufferedReader br = new BufferedReader(isr);
String s = "";
TextView textView = (TextView) findViewById(R.id.textview);
while ((s = br.readLine()) != null)
    textView.append(s + "\n");
socket.close();
}
catch (Exception e)
{
}

```

由于 51happyblog.com 只接收 HTTP 请求，因此，需要向服务端发送基本的 HTTP 头信息，必须的头信息包括命令类型、HTTP 版本（GET/HTTP/1.1）和域名（Host: 51happyblog.com）。运行本例，单击“数据传输”按钮，会在按钮下方显示从服务端读取的信息，如图 13.14 所示。



▲ 图 13.14 访问 HTTP 资源

### 13.3.4 获得无线路由分配给手机的 IP 地址

**工程目录:** src\ch13\ch13\_ip\_address

手机除了可以通过 GPRS（2G、3G）上网外，最常用的上网方式就是 Wifi。如果通过 GPRS 上网，本地的 IP 地址除了 127.0.0.1 外，还会有一个虚拟的 IP 地址。这个 IP 地址实际上作用并不大，并不像在 PC 上使用 ADSL 拨号那样获得的 IP 地址可在全球范围内访问。

然而，通过 Wifi 联网时，如果手机设置为自动获取 IP 地址，就无法确切知道无线路由为手机分配了哪个 IP 地址。这时，就需要通过 java.net.NetworkInterface 来获得这个 IP 地址，以便可以使

这个 IP 地址和其他连接在无线路由器上的设备通信。

通过 `NetWorkInterface.getNetworkInterfaces` 方法可以获得本机上所有的网络设备(包括硬件网卡和虚拟网络设备),如果在手机上调用这个方法,可以获得手机中的 Wifi 模块(相当于无线网卡),然后可以进一步获得绑定在这个 Wifi 模块上的 IP 地址(至少会有一个表示本地的 127.0.0.1)。获得手机本地 IP 地址的代码如下:

```

try
{
    TextView tvIPs = (TextView) findViewById(R.id.tvIPs);
    // 获得手机上的所有网络设备(Wifi 模块)
    Enumeration<NetworkInterface> netInterfaces = netInterfaces = NetworkInterface.
    getNetworkInterfaces();
    // 枚举所有的网络设备
    while (netInterfaces.hasMoreElements())
    {
        NetworkInterface ni = netInterfaces.nextElement();
        // 获得与当前网络设备绑定的 IP 地址
        Enumeration<InetAddress> ips = ni.getInetAddresses();
        while (ips.hasMoreElements())
        {
            // 将绑定的 IP 地址显示在 TextView 控件中
            tvIPs.append(ips.nextElement().getHostAddress() + "\n\n");
        }
    }
}
catch (Exception e)
{
}

```

运行程序,单击“获得客户端 IP 地址”按钮,在按钮下方会输出如图 13.15 所示的 IP 地址。其中 192.168.1.101 就是无线路由器分配给手机的 IP 地址。



▲ 图 13.15 显示客户端 IP 地址

### 13.3.5 设置 Socket 选项

Socket 选项可以指定 Socket 发送和接收数据的方式,比较常用的有如下 8 个选项,这 8 个选项都定义在 `java.net.SocketOptions` 接口中,代码如下:

```

public final static int TCP_NODELAY = 0x0001;
public final static int SO_REUSEADDR = 0x04;
public final static int SO_LINGER = 0x0080;
public final static int SO_TIMEOUT = 0x1006;

```

```
public final static int SO_SNDBUF = 0x1001;
public final static int SO_RCVBUF = 0x1002;
public final static int SO_KEEPALIVE = 0x0008;
public final static int SO_OOBINLINE = 0x1003;
```

这 8 个选项除了第一个没有 SO 前缀外，其他 7 个选项都以 SO 作为前缀，其实这个 SO 就是 Socket Option 的缩写，因此，在 Java 中约定所有以 SO 为前缀的常量都表示 Socket 选项。当然，也有例外，如 TCP\_NODELAY。在 Socket 类中为每一个选项提供了一对 get 和 set 方法，分别用来获得和设置这些选项。

### 1. TCP\_NODELAY

```
public boolean getTcpNoDelay() throws SocketException
public void setTcpNoDelay(boolean on) throws SocketException
```

在默认情况下，客户端向服务器发送数据时，会根据数据包的大小决定是否立即发送。当数据包中的数据很少时，如只有 1 个字节，而数据包的头却有几十个字节（IP 头+TCP 头）时，系统会在发送之前先将较小的包合并到较大的包后，一起将数据发送出去。在发送下一个数据包时，系统会等待服务器对前一个数据包的响应，当收到服务器的响应后，再发送下一个数据包，这就是所谓的 Nagle 算法。在默认情况下，Nagle 算法是开启的。

这种算法虽然可以有效地改善网络传输的效率，但对于网络速度比较慢，而且对实时性的要求比较高的情况下（如游戏、Telnet 等），使用这种方式传输数据会使客户端有明显的停顿现象。因此，最好的解决方案就是需要 Nagle 算法时就使用它，不需要时就关闭它，而使用 setTcpToDelay 方法正好可以满足这个需求。当使用 setTcpNoDelay(true) 将 Nagle 算法关闭后，客户端每发送一次数据，无论数据包的大小都会将这些数据发送出去。

### 2. SO\_REUSEADDR

```
public boolean getReuseAddress() throws SocketException
public void setReuseAddress(boolean on) throws SocketException
```

通过这个选项，可以使多个 Socket 对象绑定在同一个端口上，这种处理机制对于随机绑定端口的 Socket 对象没有什么影响，但对于绑定在固定端口的 Socket 对象就可能会抛出“Address already in use: JVM\_Bind” 异常。因此，使用这个选项可以避免抛出这个异常。

使用 SO\_REUSEADDR 选项时应注意如下几点。

- 必须在调用 bind 方法之前使用 setReuseAddress 方法来打开 SO\_REUSEADDR 选项，因此，要想使用 SO\_REUSEADDR 选项，就不能通过 Socket 类的构造方法来绑定端口。
- 必须将绑定同一个端口的所有的 Socket 对象的 SO\_REUSEADDR 选项都打开才能起作用。

### 3. SO\_LINGER

```
public int getSoLinger() throws SocketException
public void setSoLinger(boolean on, int linger) throws SocketException
```

这个 Socket 选项可以影响 close 方法的行为。在默认情况下，当调用 close 方法后，将立即返

回；如果这时仍然有未被送出的数据包，那么这些数据包将被丢弃。如果将 linger 参数设为一个正整数  $n$  时（ $n$  的值最大是 65535），在调用 close 方法后，将最多被阻塞  $n$  秒。在这  $n$  秒内，系统会尽量将未送出的数据包发送出去；如果超过了  $n$  秒，还有未发送的数据包，这些数据包将全部被丢弃；而 close 方法会立即返回。如果将 linger 设为 0，和关闭 SO\_LINGER 选项的作用是一样的。

如果底层的 Socket 实现不支持 SO\_LINGER 就会抛出 SocketException 异常，当给 linger 参数传递负数时，setSoLinger 还会抛出一个 IllegalArgumentException 异常。可以通过 getSoLinger 方法得到延迟关闭的时间，如果返回 -1，则表明 SO\_LINGER 是关闭的。例如，下面的代码将延迟关闭的时间设为 1 分钟：

```
| if(socket.getSoLinger() == -1) socket.setSoLinger(true, 60);
```

#### 4. SO\_TIMEOUT

```
| public int getSoTimeout() throws SocketException
| public void setSoTimeout(int timeout) throws SocketException
```

可以通过这个选项来设置读取数据超时。当输入流的 read 方法被阻塞时，如果设置 timeout（timeout 的单位是毫秒），那么系统在等待了 timeout 毫秒后会抛出一个 InterruptedIOException 异常。

如果将 timeout 设为 0，就意味着 read 将会无限等待下去，直到服务端程序关闭这个 Socket，这也是 timeout 的默认值，如下面的语句将读取数据超时设为 30 秒：

```
| socket1.setSoTimeout(30 * 1000);
```

当底层的 Socket 实现不支持 SO\_TIMEOUT 选项时，这两个方法将抛出 SocketException 异常。不能将 timeout 设为负数，否则 setSoTimeout 方法将抛出 IllegalArgumentException 异常。

#### 5. SO\_SNDBUF

```
| public int getSendBufferSize() throws SocketException
| public void setSendBufferSize(int size) throws SocketException
```

在默认情况下，输出流的发送缓冲区是 8096 个字节（8KB），这个值是 Java 建议的输出缓冲区的大小。如果这个默认值不能满足要求，可以用 setSendBufferSize 方法来重新设置缓冲区的大小。但最好不要将输出缓冲区设得太小，否则会导致传输数据过于频繁，从而降低网络传输的效率。

如果底层的 Socket 实现不支持 SO\_SNDBUF 选项，这两个方法将会抛出 SocketException 异常。必须将 size 设为正整数，否则 setSendBufferSize 方法将抛出 IllegalArgumentException 异常。

#### 6. SO\_RCVBUF

```
| public int getReceiveBufferSize() throws SocketException
| public void setReceiveBufferSize(int size) throws SocketException
```

这个选项与 SO\_SNDBUF 类似，它是用来设置接收缓冲区的大小，默认也是 8KB。

#### 7. SO\_KEEPALIVE

```
| public boolean getKeepAlive() throws SocketException
| public void setKeepAlive(boolean on) throws SocketException
```

如果将这个 Socket 选项打开，客户端 Socket 每隔一段的时间（大约两个小时）就会利用空闲的连接向服务器发送一个数据包。这个数据包并没有其他的作用，只是为了检测一下服务器是否仍处于活动状态。如果服务器未响应这个数据包，在一段时间后，客户端 Socket 再发送一个数据包。如果在一段时间内，服务器还没响应，那么客户端 Socket 将关闭。如果将这个 Socket 选项关闭，客户端 Socket 在服务器无效的情况下可能会长时间不会关闭。SO\_KEEPALIVE 选项在默认情况下是关闭的。

## 8. SO\_OOBINLINE

```
public boolean getOOBInline() throws SocketException
public void setOOBInline(boolean on) throws SocketException
```

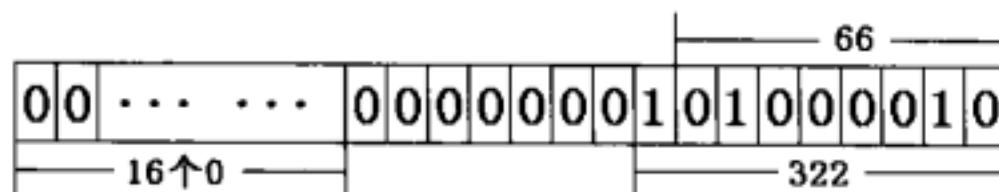
如果这个 Socket 选项打开，可以通过 `Socket.sendUrgentData` 方法向服务器发送一个单字节的数据，这个单字节数据并不经过输出缓冲区，而是立即发出。虽然在客户端并没有使用 `OutputStream` 向服务器发送数据，但在服务端程序中这个单字节的数据是和其他的普通数据混在一起的。因此，在服务端程序中并不知道由客户端发过来的数据是由 `OutputStream` 还是由 `sendUrgentData` 发过来的。下面是 `sendUrgentData` 方法的定义：

```
public void sendUrgentData(int data) throws IOException
```

虽然 `sendUrgentData` 的 `data` 参数是 `int` 类型，但只有这个 `int` 类型的低字节被发送，其他的三个字节被忽略。下面的代码演示了如何使用 `SO_OOBINLINE` 选项来发送英文字母和汉字。

```
Socket socket = new Socket("127.0.0.1", 1234);
socket.setOOBInline(true);
OutputStream out = socket.getOutputStream();
OutputStreamWriter outWriter = new OutputStreamWriter(out);
outWriter.write(67); // 向服务器发送字符 "C"
outWriter.write("hello world\r\n");
socket.sendUrgentData(65); // 向服务器发送字符 "A"
socket.sendUrgentData(322); // 向服务器发送字符 "B"
outWriter.flush();
socket.sendUrgentData(214); // 向服务器发送汉字 "中"
socket.sendUrgentData(208); // 向服务器发送汉字 "国"
socket.sendUrgentData(185); // 向服务器发送汉字 "国"
socket.sendUrgentData(250);
socket.close();
```

上面的代码使用了 `sendUrgentData` 方法向服务器发送了字符 “A” (65) 和 “B” (66)，但发送 “B” 时实际发送的是 322（直接发送 66 也可以，本例只是做一个实验），由于 `sendUrgentData` 只发送整型数的低字节，因此，实际发送的是 66。十进制整数 322 的二进制形式如图 13.16 所示。



▲ 图 13.16 十进制整数 322 的二进制形式

在上面代码中使用 flush 方法将缓冲区中的数据发送到服务器，我们可以从输出结果发现一个问题，在 Client 类中先后向服务器发送了“C”、“hello world\r\n”、“A”、“B”，而服务端程序获得的数据却是 ABCHello world，这说明 sendUrgentData 方法会立刻将数据发送出去，而使用 write 发送数据，除非数据超过了发送缓冲区的大小或关闭了 Socket，否则必须要使用 flush 方法才会真正将数据发送出去（flush 方法用于发送未满的缓冲区中的数据，并清空缓冲区）。



**注意** 在使用 setOOBInline 方法打开 SO\_OOBINLINE 选项时必须在客户端和服务端程序同时使用 setOOBInline 方法打开这个选项，否则无法正常用 sendUrgentData 来发送数据。

## 13.4 服务端 Socket

ServerSocket 对象用于将服务端的 IP 地址和端口绑定，从而客户端可以通过同一网段的 IP 地址和端口访问服务端程序。因此，ServerSocket 也叫服务端 Socket，主要用于实现服务器程序。

### 13.4.1 手机服务器的实现

工程目录：src\ch13\ch13\_serversocket

服务器程序一般都是部署在高端服务器或 PC 上的，随着智能手机性能的不断提高，也可以将计算量不大的服务程序部署在手机上，成为移动服务器。部署服务的手机一般会通过 Wifi 连到无线路上，这样手机就可以通过无线路由分配的 IP 地址与其他的计算机或智能设备进行通信了。

本小节利用 ServerSocket 实现一个简单的手机服务程序。读者在测试随书光盘中的代码之前，请确保手机已通过 Wifi 连接到无线路上。笔者的无线路由分配给手机的 IP 地址是 192.168.1.101，如果读者的手机分配的是其他 IP 地址，请修改源程序，并重新编译再进行测试。

本例通过在一个线程中创建一个 ServerSocket 对象，并将其绑定在无线路由分配给手机的 IP 地址上，端口是 1234。当在 PC 的浏览器（IE、Firefox 等）中访问 “<http://192.168.1.101:1234>” 时，会在浏览器中输出手机设备唯一编号的 hashcode。读者可以利用这个例子来实现更复杂的移动服务。

负责创建 ServerSocket 对象、绑定 IP 地址和端口以及接收客户端数据的线程类的代码如下：

```
class ServerSocketThread extends Thread
{
    @Override
    public void run()
    {
        try
        {
            ServerSocket serverSocket = new ServerSocket();
            // 绑定 IP 地址和端口，读者要将 192.168.1.101 换成无线路由分配给自己手机的 IP 地址
            serverSocket.bind(new InetSocketAddress("192.168.1.101", 1234));
            while (true)
            {

```

```

    // accept 方法用于接收客户端的请求，如果没有请求，会处于阻塞状态
    Socket socket = serverSocket.accept();
    final TelephonyManager telephonyManager = (TelephonyManager) getBaseContext()
        .getSystemService(Context.TELEPHONY_SERVICE);
    // 向客户端输出手机设备号的 hashcode
    socket.getOutputStream().write(("DeviceId Hashcode: " + String.valueOf(
        telephonyManager.getDeviceId()).hashCode()).getBytes());
    socket.close();
}
}
catch (Exception e)
{
}
}

```

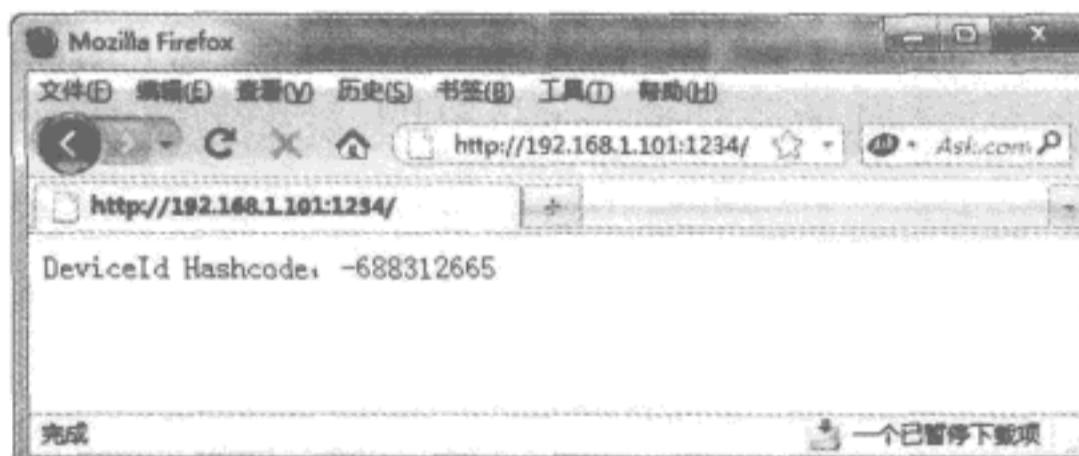
获得手机的设备号需要在 `AndroidManifest.xml` 文件中使用如下的代码设置权限。

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

运行程序，单击“开始服务”按钮，然后在 IE、Firefox 等浏览器中输入如下的 URL。

```
http://192.168.1.101:1234
```

在浏览器中会输出如图 13.17 所示的信息。



▲ 图 13.17 访问手机服务器

### 13.4.2 利用 Socket 在应用程序之间通信

工程目录：`src\ch13\ch13_data_server`、`src\ch13\ch13_data_client`

4.3 节介绍了如何在不同的 Activity 之间传输数据，在第 9、第 10、第 11、第 12 章讨论了如何通过 Android 的四大应用程序组件在不同应用程序之间传输数据。虽然前面的章节介绍了很多数据传输方法，但那并不是全部，本小节将介绍一种通过 Socket 在应用程序之间传输数据的方法。这种方法也很简单，就是把两个要相互传输数据的程序的其中一个作为服务器，另一个作为客户端，或互为服务器和客户端，并且通过 `OutputStream` 和 `InputStream` 在服务端和客户端之间传输数据。

本例由服务端（`ch13_data_server`）和客户端（`ch13_data_client`）两个程序组成。服务端程序接收客户端发来的一个随机整数，并将其显示在 `TextView` 控件上。客户端会通过 127.0.0.1 和端口 1234 连接服务端来发送随机整数。服务端接收数据的代码如下：

```
package mobile.android.ch13.data.server;
```

```
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import java.net.ServerSocket;
import java.net.Socket;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.TextView;

public class Main extends Activity
{
    private TextView textView;
    private Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            super.handleMessage(msg);
            // 向TextView控件添加随机整数
            textView.append(String.valueOf(msg.obj) + "\n\n");
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView = (TextView) findViewById(R.id.textview);
        new ServerSocketThread().start();
    }
    class ServerSocketThread extends Thread
    {
        @Override
        public void run()
        {
            try
            {
                ServerSocket serverSocket = new ServerSocket();
                serverSocket.bind(new InetSocketAddress("127.0.0.1", 1234));
                while (true)
                {
                    Socket socket = serverSocket.accept();
                    InputStream is = socket.getInputStream();
                    InputStreamReader isr = new InputStreamReader(is);
                    BufferedReader br = new BufferedReader(isr);
                    Message message = new Message();
                    // 读取客户端发送过来的一行文件信息(随机整数)
                    message.obj = br.readLine();
                    // 使用Handler发送消息来向TextView控件添加随机整数
                    handler.sendMessage(message);
                    socket.close();
                }
            }
        }
    }
}
```



```
        }
    }
    catch (Exception e)
    {
        }
    }
}
```

客户端传输数据的代码如下：

```
try
{
    Socket socket = new Socket("127.0.0.1", 1234);
    Random random = new Random();
    String data = String.valueOf(random.nextInt());
    // 发送数据
    socket.getOutputStream().write(data.getBytes());
    socket.getOutputStream().flush();
    socket.close();
    Toast.makeText(this, "数据已发送", Toast.LENGTH_LONG).show();
}
catch (Exception e)
{
}
```

首先运行服务端程序，再运行客户端程序，然后单击几次客户端程序的“发送数据”按钮，关闭客户端程序后，会看到如图 13.18 所示的显示信息。



▲ 图 13.18 通过 Socket 发送数据

13.5 蓝牙通信

本小节介绍的蓝牙(Bluetooth)要求的最低版本是Android 2.0。由于Android模拟器不支持蓝牙，因此需要在Android 2.0及以上版本的真机上测试本节的例子。笔者使用了Google Nexus S(Android 2.3)和HTC Hero(G3, Android 2.1)两部手机测试了本节的例子，读者也可以使用其他型号的手机进行测试。

蓝牙是一种重要的短距离无线通信协议，广泛应用于各种设备（计算机、手机、汽车等）中。为了使读者更好地使用蓝牙技术，本节从实用的角度介绍了蓝牙的基本原理和使用方法。

### 13.5.1 蓝牙简介

蓝牙这个名字来源于 10 世纪丹麦国王 Harald Blatand，英文名字是 Harold Bluetooth。在无线行

业协会组织人员讨论后，有人认为用 Blatand 国王的名字命名这种无线技术是再好不过了，这是因为 Blatand 国王将挪威、瑞典和丹麦统一起来，这就如同这项技术将统一无线通信领域一样。至此，蓝牙的名字也就这样定了下来。

蓝牙采用了分散式网络结构以及快跳频和短包技术，支持点对点及点对多点的通信，工作在全球通用的 2.4GHz ISM（即工业、科学、医学）频度。根据不同的蓝牙版本，传输速度会差很多，例如，最新的蓝牙 3.0 传输速度为 3Mbit/s，而未来的蓝牙 4.0 技术从理论上可达到 60Mbit/s。

蓝牙协议分为 4 层，即核心协议层、电缆替代协议层、电话控制协议层和采纳的其他协议层，这 4 种协议中最重要的是核心协议层。蓝牙的核心协议层包括基带、链路管理、逻辑链路控制和适应协议四部分。其中链路管理（LMP）负责蓝牙组件间连接的建立。逻辑链路控制与适应协议（L2CAP）位于基带协议层上，属于数据链路层，是一个为高层传输和应用层协议屏蔽基带协议的适配协议。

蓝牙技术作为目前比较常用的无线通信技术，早已成为手机的标配之一，基于 Android 的手机也不例外。但遗憾的是，Android 1.x 对蓝牙的支持非常不完善，只支持像蓝牙耳机一样的设备，并不支持蓝牙数据传输等高级特性。不过，Android 2.0 终于加入了完善的蓝牙支持。

### 13.5.2 打开和关闭蓝牙设备

工程目录：src\ch13\ch13\_control\_bluetooth\_device

与蓝牙相关的类和接口位于 android.bluetooth 包中。在使用蓝牙之前，需要在 AndroidManifest.xml 文件中打开相应的权限，代码如下：

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

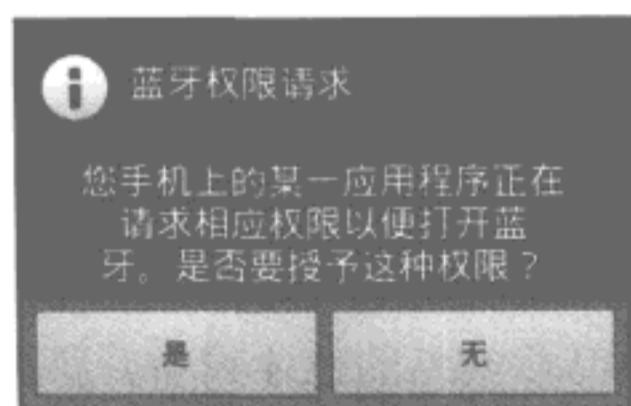
BluetoothAdapter 是蓝牙中的核心类，下面的代码创建了 BluetoothAdapter 对象。

```
private BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

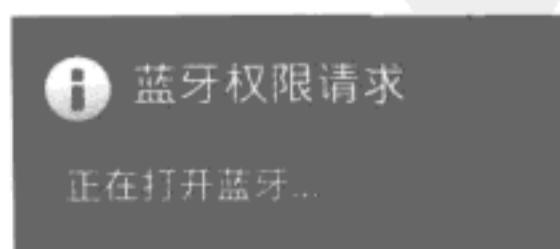
使用下面的代码可以打开蓝牙。

```
Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableIntent, 1);
```

在执行上面代码后，如果这时蓝牙未打开，会弹出如图 13.19 所示的对话框，询问是否打开蓝牙。如果单击“是”按钮，会显示 13.20 所示的“正在打开蓝牙”状态信息框。大概 5 秒左右，在状态栏中会显示蓝牙标记，如图 13.21 白框中所示。



▲ 图 13.19 询问是否打开蓝牙



▲ 图 13.20 “正在打开蓝牙”状态信息框



▲ 图 13.21 蓝牙已打开

直接调用 `BluetoothAdapter.enable` 方法也可以打开蓝牙，代码如下：

```
bluetoothAdapter.enable();
```

要关闭蓝牙，可以使用下面的代码：

```
bluetoothAdapter.disable();
```



**注意** 使用 `enable` 方法和 `BluetoothAdapter.ACTION_REQUEST_ENABLE` 虽然都可以打开蓝牙，但 `enable` 方法并不会出现如图 13.19 和图 13.20 所示的提示框和信息框，只会无声息地开启蓝牙设备。

### 13.5.3 搜索蓝牙设备

工程目录：src\ch13\ch13\_search\_bluetooth\_device

与其他蓝牙设备通信之前需要搜索周围的蓝牙设备。要想自己的手机被其他的蓝牙设备搜索到，需要进入蓝牙设置界面（如图 13.22 所示），选中“可检测性”复选框，这样自己的手机就可以被其他蓝牙设备搜索到了。单击“扫描查找设备”列表项，系统就会搜索周围的蓝牙设备。



▲ 图 13.22 蓝牙设置界面

如果手机中已经和某些蓝牙设备绑定，可以使用 `BluetoothAdapter.getBondedDevices` 方法获得已绑定的蓝牙设备列表。搜索周围的蓝牙设备使用 `BluetoothAdapter.startDiscovery` 方法，搜索到的蓝牙设备通过广播返回，因此，需要注册广播接收器来获得已搜索到的蓝牙设备。获得已绑定的蓝牙设备信息以及搜索蓝牙设备的完整代码如下：

```
package mobile.android.ch13.search.bluetooth.device;

import java.util.Set;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
import android.widget.TextView;

public class Main extends Activity
{
    private BluetoothAdapter bluetoothAdapter;
    private TextView tvDevices;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
        setContentView(R.layout.main);
        tvDevices = (TextView) findViewById(R.id.tvDevices);
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        // 获得所有已绑定的蓝牙设备
        Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();

        if (pairedDevices.size() > 0)
        {
            for (BluetoothDevice device : pairedDevices)
            {
                // 将已绑定的蓝牙设备的名称和地址显示在 TextView 控件中
                tvDevices.append(device.getName() + ":" + device.getAddress() + "\n");
            }
        }
        // 注册用于接收已搜索到的蓝牙设备的 Receiver
        IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        this.registerReceiver(receiver, filter);
        // 注册搜索完成时的 Receiver
        filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        this.registerReceiver(receiver, filter);
    }
    public void onClick_Search(View view)
    {
        setProgressBarIndeterminateVisibility(true);
        setTitle("正在扫描...");
        // 如果这时正好在搜索，先取消搜索
        if (bluetoothAdapter.isDiscovering())
        {
            bluetoothAdapter.cancelDiscovery();
        }
    }
}
```

```

    // 开始搜索蓝牙设备
    bluetoothAdapter.startDiscovery();
}
private final BroadcastReceiver receiver = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();
        // 获得已搜索到的蓝牙设备
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {

            BluetoothDevice device = intent
                .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // 搜索到的设备不是已绑定的蓝牙设备
            if (device.getBondState() != BluetoothDevice.BOND_BONDED)
            {
                // 将搜索到的新蓝牙设备显示在 TextView 控件中
                tvDevices.append(device.getName() + ":" + device.getAddress() + "\n");
            }
        }
        // 搜索完成
        else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
        {
            setProgressBarIndeterminateVisibility(false);
            setTitle("搜索蓝牙设备");
        }
    }
};
}

```

运行本例，单击“搜索蓝牙设备”按钮，系统开始搜索，如果搜索到蓝牙设备，会显示在按钮下方的 TextView 控件中，如图 13.23 所示。



▲ 图 13.23 搜索蓝牙设备

#### 13.5.4 蓝牙数据传输

工程目录: src\ch13\ch13\_bluetooth\_socket

蓝牙传输数据的方法与 Socket 类似。在网络中使用 Socket 和 ServerSocket 控制客户端和服务端的数据读写，而蓝牙通信也由客户端和服务端 Socket 来完成。蓝牙客户端 Socket 是 BluetoothSocket，蓝牙服务端 Socket 是 BluetoothServerSocket，这两个类都在 android.bluetooth 包中。



无论是 BluetoothSocket，还是 BluetoothServerSocket，都需要一个 UUID（全局唯一标识符，Universally Unique Identifier），格式如下：

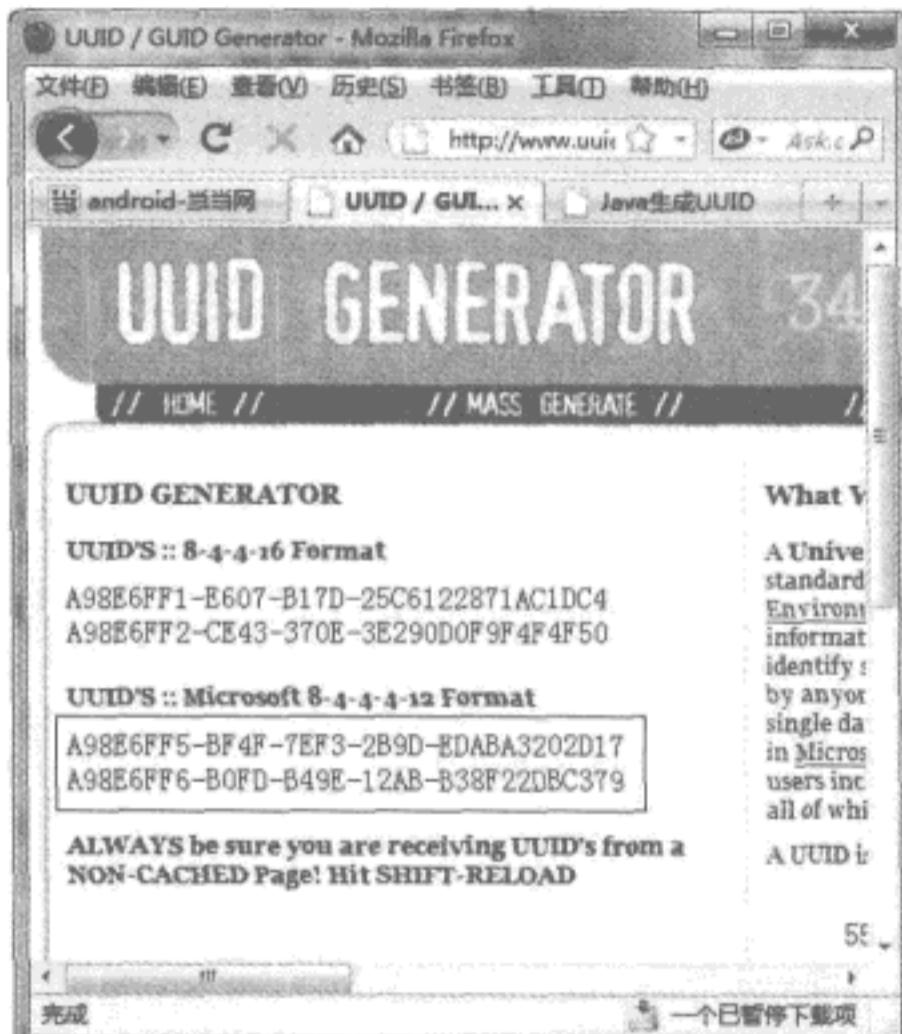
| xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

UUID 的格式被分成 5 段，其中中间 3 段的字符数相同，都是 4，第 1 段是 8 个字符，最后一段是 12 个字符，所以 UUID 实际上是一个 8-4-4-4-12 的字符串，只是这个字符串要求永不重复。

获得 UUID 的方法非常多。例如，可以从下面的页面直接获得 UUID 字符串，每刷新一次页面，页面的左上角就会生成两个新的 UUID。

| <http://www.uuidgenerator.com>

生成 UUID 的页面如图 13.24 所示。



▲ 图 13.24 生成 UUID 的页面

### ■ 注意

如图 13.24 所示的页面产生了两组 UUID，其中第 1 组 UUID 的格式为 8-4-4-16，也就是 4 段的 UUID，第 2 组就是前面介绍的 8-4-4-4-12 格式的 UUID，我们应选择第 2 组 UUID，也就是黑框中的 UUID。如果选择第 1 组中的 UUID，蓝牙 Socket 会抛出异常。

JDK 本身也提供了生成 UUID 的 API，代码如下：

| String uuid = java.util.UUID.randomUUID().toString();

### 答疑解惑：

也许有的读者会问，使用蓝牙时为什么需要这个 UUID 呢？实际上，这个 UUID 与网络 Socket 的端口类似。网络客户端在访问服务端时，要指定 IP 地址和端口号。其中 IP 地址相当于蓝牙模块

的地址（相当于网卡的 Mac 地址）。同一台服务器可能运行多个网络服务，这些服务必须拥有不同的端口号。客户端为了连接指定的网络服务，也必须指定这个服务的端口号。

如果手机中安装了多个蓝牙应用程序，而且都处于运行状态，那么与另一部手机通过蓝牙连接时到底使用哪一个程序呢？这就需要使用 UUID 来区分。也就是说，两部手机之间通过蓝牙通信的软件必须使用同一个 UUID（相当于网络客户端访问服务端要指定服务的端口号一样），否则无法成功进行连接，这就是为什么很多时候不同的文件管理器无法通过蓝牙传输文件的原因。

本节的例子可以通过蓝牙传输字符串，首先需要编写一个接收蓝牙客户端请求的类，代码如下：

```

private class AcceptThread extends Thread
{
    private BluetoothServerSocket serverSocket;
    private BluetoothSocket socket;
    private InputStream is;
    private OutputStream os;
    public AcceptThread()
    {
        try
        {
            // 创建 BluetoothServerSocket 对象
            serverSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME,
                MY_UUID);
        }
        catch (IOException e)
        {
        }
    }
    public void run()
    {
        try
        {
            // 等待接收蓝牙客户端的请求
            socket = serverSocket.accept();
            is = socket.getInputStream();
            os = socket.getOutputStream();
            // 通过循环不断接收客户端发过来的数据，如果客户端暂时没发数据，则 read 方法处于阻塞状态
            while (true)
            {
                byte[] buffer = new byte[128];
                int count = is.read(buffer);
                Message msg = new Message();
                msg.obj = new String(buffer, 0, count, "utf-8");
                // 通过 Toast 信息框显示客户端发过来的信息
                handler.sendMessage(msg);
            }
        }
        catch (Exception e)
        {
        }
    }
}

```

在编写 AcceptThread 类时应了解如下几点。

- BluetoothAdapter.listenUsingRfcommWithServiceRecord 方法用于创建 BluetoothServerSocket 对象。listenUsingRfcommWithServiceRecord 方法的第一个参数表示蓝牙服务的名称，可以是任意字符串，第 2 个参数就是 UUID。本例生成了一个固定的 UUID，读者也可以使用其他的 UUID。
- 通过 BluetoothServerSocket.accept 方法收到客户端的请求后，accept 方法会返回一个 BluetoothSocket 对象，可以通过该对象获得读写数据的 InputStream 和 OutputStream 对象。如果想编写类似聊天的程序，可以在循环中不断读取客户端的数据。
- InputStream.read 方法在服务端未发送数据时处于阻塞状态，直到服务端发过来数据，才会执行后面的语句。

编写完 AcceptThread 类后，在 onCreate 方法中需要创建 AcceptThread 对象来监听客户端的请求，代码如下：

```
AcceptThread acceptThread = new AcceptThread();
acceptThread.start();
```

本例与 13.4.4 节的例子一样，也可以搜索蓝牙设备（请参考本例的源代码）。当搜索到蓝牙设备后，会显示在搜索按钮下方的 ListView 控件中，然后单击列表项，就会向已连接的蓝牙设备发送信息。服务端如果接收到客户端发送的信息后，会显示该信息。下面看一下单击列表项发送信息的代码。

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
    String s = arrayAdapter.getItem(position);
    // 获得要连接的蓝牙设备的地址
    String address = s.substring(s.indexOf(":") + 1).trim();
    try
    {
        if (bluetoothAdapter.isDiscovering())
        {
            // 如果这时正在搜索蓝牙设备，取消搜索
            this.bluetoothAdapter.cancelDiscovery();
        }
        try
        {
            if (device == null)
            {
                // 获得蓝牙设备，相当于网络客户端 Socket 指定 IP 地址
                device = bluetoothAdapter.getRemoteDevice(address);
            }
            if (clientSocket == null)
            {
                // 通过 UUID 连接蓝牙设备，相当于网络客户端 Socket 指定端口号
                clientSocket = device.createRfcommSocketToServiceRecord(MY_UUID);
                // 开始连接蓝牙设备
                clientSocket.connect();
                // 获得向服务端发送数据的 OutputStream 对象
            }
        }
    }
}
```

```

        os = clientSocket.getOutputStream();
    }
}
catch (IOException e)
{
}
if (os != null)
{
    // 向服务端发送一个字符串
    os.write("发送信息到其他蓝牙设备.".getBytes("utf-8"));
    Toast.makeText(this, "信息发送成功.", Toast.LENGTH_LONG).show();
}
else
{
    Toast.makeText(this, "信息发送失败.", Toast.LENGTH_LONG).show();
}
}
catch (Exception e)
{
    Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();
}
}
}

```

本例同时拥有蓝牙服务端和客户端的功能。准备两部 Android 手机，并开启蓝牙功能，需要在蓝牙设置中进行配对，这样在程序启动后会直接将其他的蓝牙设备显示在 ListView 控件中，如图 13.25 所示。



▲ 图 13.25 蓝牙通信

单击列表项后，会看到客户端和服务端弹出相应的 Toast 信息框。

### 扩展学习：

如果在测试本例之前未通过系统的搜索来配对蓝牙程序，可以单击如图 13.25 所示界面中的“搜索”按钮来搜索蓝牙设备。但要注意，Android 手机默认情况下，即使蓝牙已开启，也不能搜索到当前的设备。需要在蓝牙设备界面中选择“可检测性”复选框（如图 13.26 所示），才可以在指定时间（一般为 2 分钟）内搜索到当前的 Android 手机。

当然，我们也可以通过程序来设置蓝牙手机的可检测性，代码如下：

```

Intent enabler = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivityForResult(enabler, REQUEST_DISCOVERABLE);

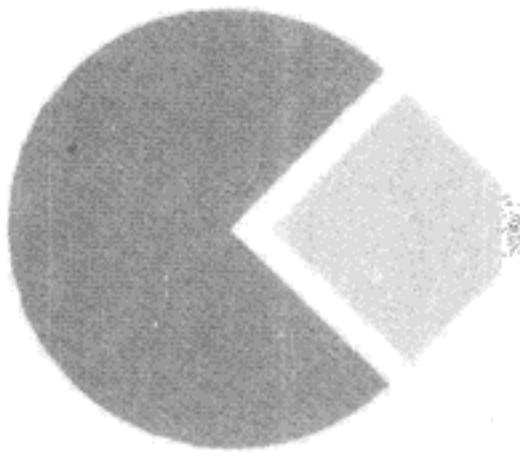
```



▲ 图 13.26 设置蓝牙手机的可检测性

## 13.6 小结

本章主要介绍了 Android SDK 提供的各种通信 API。从技术层面来看，分为网络通信和蓝牙通信。网络通信的核心是 Socket 和 ServerSocket，而 HTTP 通信是建立在网络 Socket 基础上的一种通信方式。蓝牙通信与网络通信在使用方法上类似，都通过 InputStream 和 OutputStream 来传递数据，但它们不同的是网络通信需要指定服务器的 IP（或域名）地址和端口号，而蓝牙通信需要指定要连接的蓝牙设备的蓝牙模块地址和 UUID。



## 第 14 章 炫酷你的应用——多媒体开发

现在的手机拥有多媒体功能已经不是什么新鲜事了，最基本的多媒体功能是播放音乐、录音。自从将摄像头作为手机的标准配件（除了特别便宜的手机，几乎所有的手机都会配摄像头）起，手机又多了一项任务：拍照，而摄像头也成为在手机中增加的最成功的模块。除了这些，像闹钟、铃声、播放视频、录像等，也成为大多数手机的标配。Android 手机作为智能手机的新贵，这些多媒体功能自然会非常完美地支持，本章将详细介绍如何使用 Android 手机中的这些多媒体功能。

### 14.1 音乐

Android 可以播放多种音频格式，最常使用的当属 mp3。Android SDK 中提供了一个 MediaPlayer 控件来播放包括 mp3 在内的音频文件，而 MediaRecorder 与 MediaPlayer 正好相反，MediaRecorder 控件可以利用手机的麦克风录音，并将录制的结果保存成音频文件。

#### 14.1.1 播放音乐

工程目录: src\ch14\ch14\_mediaplayer

使用 android.media.MediaPlayer 类可以播放音频资源，这些音频资源可以包含在 apk 文件中，也可以保存在 SD 卡或手机内存中。

播放包含在 apk 中的音频资源的代码如下：

```
// 通过MediaPlayer类的create方法指定res\raw目录中的音频资源(mp3文件)，并创建MediaPlayer对象
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.music);
if (mediaPlayer != null)
    mediaPlayer.stop();
// 在播放音频资源之前，必须调用prepare方法完成一些准备工作
mediaPlayer.prepare();
// 开始播放音频
mediaPlayer.start();
```

如果要播放保存在 SD 卡或手机内存中的音频文件，需要使用下面的代码。

```
MediaPlayer mediaPlayer = new MediaPlayer();
// 指定音频文件的路径
mediaPlayer.setDataSource("/sdcard/music.mp3");
mediaPlayer.prepare();
mediaPlayer.start();
```

暂停和停止播放可以使用 MediaPlayer 类的 pause 和 stop 方法，代码如下：

```
MediaPlayer.pause(); // 暂停播放
MediaPlayer.stop(); // 停止播放
```

MediaPlayer 类还支持播放过程中的事件，例如当播放完音频资源时，会触发 onCompletion 事件。可以在该事件方法中释放音频资源，以便其他应用程序可以使用该资源。代码如下：

```
public void onCompletion(MediaPlayer mp)
{
    // 释放音频资源
    mp.release();
    setTitle("资源已经释放");
}
```

使用下面的代码指定 onCompletion 事件监听对象。

```
 mediaPlayer.setOnCompletionListener(this); // 当前类实现 OnCompletionListener 接口
```

运行本例，会显示如图 14.1 所示的界面，单击相应的按钮，可以播放音频、暂停和停止播放音频。但要注意，在单击“播放 SD 卡中的 MP3 文件”按钮之前，在 SD 卡的根目录中要有一个 music.mp3 文件，否则系统不会正常播放 MP3 文件。



▲ 图 14.1 播放音频文件

### 14.1.2 录音

工程目录：src\ch14\ch14\_recorder

使用 android.media.MediaRecorder 类可以通过手机上的内置麦克风录音，代码如下：

```
File recordAudioFile = File.createTempFile("record", ".amr");
MediaRecorder mediaRecorder = new MediaRecorder();
// 指定音频来源（麦克风）
mediaRecorder.set AudioSource(MediaRecorder.AudioSource.MIC);
// 指定音频输出格式（MPGE4）
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
// 指定音频编码方式
mediaRecorder.set AudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);
// 指定录制文件的路径
```

```

mediaRecorder.setOutputFile(recordAudioFile.getAbsolutePath());
mediaRecorder.prepare();
// 开始录音
mediaRecorder.start();

```

在编写上面代码时应注意如下几点。

- 录音前要使用 setXxx (set AudioSource、setOutputFormat 等) 方法设置录制的音频属性和输出文件路径。
- 音频文件使用了临时文件。这是由于临时文件每次生成的文件名不同，可以保证在不删除文件的情况下不会覆盖以前录制的音频文件。
- MediaRecorder 和 MediaPlayer 一样，在调用 start 方法录音之前，也需要使用 prepare 方法完成一些准备工作。

停止录音可以使用下面的代码。在停止录音后，最后需要释放录制的音频文件，以便其他应用程序可以继续使用这个音频文件。

```

// 停止录音
mediaRecorder.stop();
// 释放录制的音频文件
mediaRecorder.release();

```

如果想删除录制的音频文件，需要在停止录制后，执行如下代码：

```
recordAudioFile.delete();
```

运行本例，会看到如图 14.2 所示的界面，单击相应的按钮后，可以录制、停止、播放和删除录制的音频文件。播放音频文件可以使用上一小节介绍的 MediaPlayer 类。



▲ 图 14.2 录音

## 14.2 视频

Android SDK 提供了多种方式播放视频资源。例如，最简单的可以使用 VideoView 来播放视频资源。除此之外，还可以使用更灵活的 SurfaceView 来播放视频资源。大多数手机自带的摄像头不仅有拍照功能，也可以录制视频。当然，利用摄像头录制的视频同样可以使用 VideoView 和 SurfaceView 来播放。

### 14.2.1 使用 VideoView 播放视频

工程目录：src\ch14\ch14\_videoview

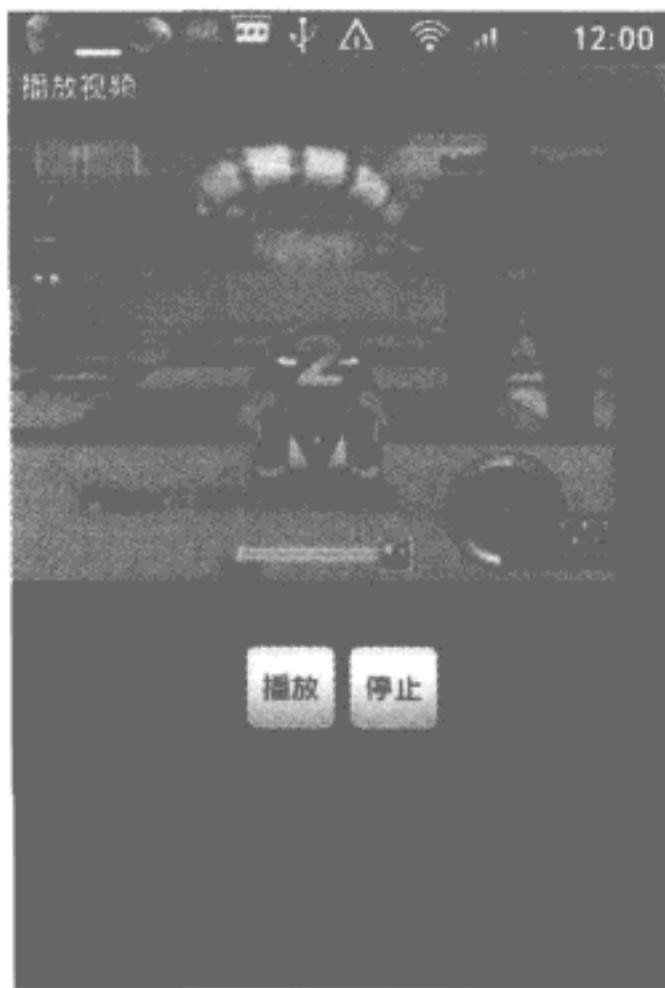
使用 `android.widget.VideoView` 类可以播放 mp4 的 H.264、3gp 和 wmv 格式的视频文件（播放其他格式的视频文件需要移植解码器）。本节的例子将播放一个 3gp 格式的视频文件。在播放视频之前，需要在 XML 布局文件中放置一个`<VideoView>`标签，代码如下：

```
<VideoView android:id="@+id/videoView" android:layout_width="320px"
    android:layout_height="240px" />
```

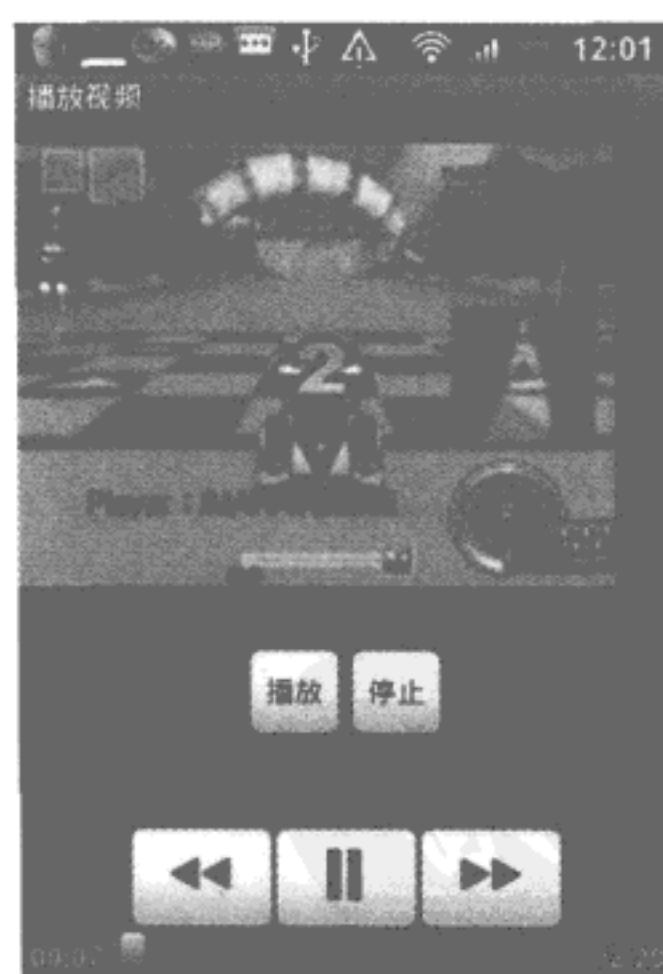
播放视频的代码如下：

```
// 指定要播放的视频文件
videoView.setVideoURI(Uri.parse("file:///sdcard/video.3gp"));
// 设置视频控制器
videoView.setMediaController(new MediaController(this));
// 开始播放视频
videoView.start();
```

运行程序，并单击“播放”按钮，会播放 SD 卡中的 video.3gp 文件（读者在运行程序前，请将 `src\ch14\ch14_videoview\video.3gp` 文件复制到 SD 卡的根目录），播放的效果如图 14.3 所示。在上面代码中使用 `setMediaController` 方法设置了一个媒体控制器。当触摸播放界面时，会在屏幕下方显示一个媒体控制器，如图 14.4 所示。通过这个媒体控制器，可以快进、快退和暂停视频，也可以调整当前播放的视频位置，并查看视频总时间和当前已播放的时间。



▲ 图 14.3 播放视频



▲ 图 14.4 媒体控制器

如果想控制视频的暂停和停止，可以使用下面的代码。

```
videoView.pause();           // 暂停视频的播放
videoView.stopPlayback();    // 停止视频的播放
```

## 14.2.2 使用 SurfaceView 播放视频

工程目录：src\ch14\ch14\_surfaceview

虽然 VideoView 控件可以播放视频，但播放的位置和大小并不受我们控制。为了对视频有更多的控制权，可以使用 MediaPlayer 配合 SurfaceView 来播放视频。

在使用 SurfaceView 控件之前需要创建 SurfaceHolder 对象，并进行相应的设置，代码如下：

```
SurfaceView surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
SurfaceHolder surfaceHolder = surfaceView.getHolder();
surfaceHolder.setFixedSize(100, 100);
surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

其中 setFixedSize 方法用来设置播放视频界面的固定大小。但经笔者测试，setFixedSize 方法的两个参数设置成多少，视频界面也会尽量充满整个 SurfaceView 控件，如图 14.5 所示。但如果不对该方法，视频会以实际大小播放，其他的区域会显示成黑色，如图 14.6 所示。



▲ 图 14.5 充满整个 SurfaceView 控件播放视频



▲ 图 14.6 以实际大小播放视频

播放视频的代码如下：

```
mediaPlayer = new MediaPlayer();
// 设置音频流类型
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
// 设置用于播放视频的 SurfaceView 控件
mediaPlayer.setDisplay(surfaceHolder);
try
{
    // 指定视频文件
    mediaPlayer.setDataSource("/sdcard/video.3gp");
    mediaPlayer.prepare();
    mediaPlayer.start();
}
catch (Exception e)
{}
```

使用 MediaPlayer 播放视频的关键是指定用于显示视频的 SurfaceView 对象（通过 setDisplay 方法设置）。至于暂停和停止视频的播放，可以直接使用 MediaPlayer 类的 pause 和 stop 方法。

### 14.2.3 录制视频

工程目录: src\ch14\ch14\_record\_video

手机自带的摄像头不仅可以拍照，也可以录像。更有趣的是，我们可以通过程序来调用系统的摄像功能。录像界面需要使用一个 Activity Action 来调用，代码如下：

```
Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE); // 指定录像界面的Activity Action
startActivityForResult(intent, 1);
```

其中 startActivityForResult 方法的第二个参数值 1 表示请求代码。

当录像完成后，会通过 onActivityResult 方法传回新录制的视频文件路径，得到这个路径后，可以进一步处理视频文件，例如，下面的代码使用 VideoView 控件立即播放刚录制的视频。

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1)
    {
        if (resultCode == Activity.RESULT_OK)
        {
            Uri uri = data.getData(); // 获得视频文件的Content Provider Uri
            // 通过Content Provider 查询刚录制的视频
            Cursor cursor = this.getContentResolver().query(uri, null, null, null, null);
            if (cursor.moveToFirst())
            {
                // 获得视频文件的绝对路径
                String videoPath = cursor.getString(cursor.getColumnIndex("_data"));
                // 指定要播放的视频文件
                videoView.setVideoURI(Uri.parse(videoPath));
                // 设置视频控制器
                videoView.setMediaController(new MediaController(this));
                // 开始播放视频
                videoView.start();
            }
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

onActivityResult 方法并不会直接返回视频文件的路径，而是返回一个 Content Provider Uri，这个 Uri 类似下面的形式。

```
content://media/external/video/media/23
```

通过这个 Uri 可以查询到视频文件的详细信息，当然，也包括视频文件的绝对路径。最后在 VideoView 控件中立即播放这个刚录制的视频文件。

## 14.3 相机

14.2.3 小节演示了如何在自己的程序中调用系统的录像功能，实际上，摄像头最常用的功能

仍然是拍照。除了可以调用系统的拍照功能外，还可以自定义拍照界面，从而设计出更复杂的拍照程序。

### 14.3.1 调用系统的拍照功能

工程目录：src\ch14\ch14\_systemcamera

读者可以先试试自己手机上的拍照功能，可能由于手机型号不同，拍照的方式和过程也可能不一样。在 HTC 的大多数 Android 手机上进行拍照会由系统自动对焦，在对焦的过程中，屏幕上会出现一个白色的对焦符号（类似于中括号），如果对焦成功，这个对焦符号就会变成绿色，当对焦成功后，按手机下方的“呼吸灯”按钮进行拍照（不同手机的拍照方式可能不同，例如，可以直接触摸屏幕拍照）。在拍照后手机屏幕下方会出现两个按钮：“完成”和“拍照”。如果对照片满意，单击“完成”按钮结束拍照。如果对照片不满意，单击“拍照”按钮继续拍照，上一次拍的照片将丢失。

调用系统拍照功能与调用录像功能类似，也需要使用通过 Activity Action 完成。与拍照功能对应的 Activity Action 是 android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE。拍完照后，onActivityResult 方法会返回图像文件的 Bitmap 对象，可以直接使用这个 Bitmap 对象将图像显示在 ImageView 控件中。

调用系统拍照功能的代码如下：

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(intent, 1);
```

处理拍照返回的图像数据的代码如下：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1)
    {
        if (resultCode == Activity.RESULT_OK)
        {
            // 拍照 Activity 保存图像数据的 key 是 data，返回的数据类型是 Bitmap 对象
            Bitmap cameraBitmap = (Bitmap) data.getExtras().get("data");
            // 在 ImageView 控件中显示拍摄的照片
            imageView.setImageBitmap(cameraBitmap);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

在默认情况下，系统将照片保存在 SD 卡的 DCIM\100MEDIA 目录中，不同型号的手机可能保存的目录不同。

运行程序，单击“拍照”按钮进行拍照，拍完照后，会将所拍的照片显示在 ImageView 控件中，如图 14.7 所示。



▲ 图 14.7 调用系统的拍照功能

### 14.3.2 自定义拍照功能

工程目录: `src\ch14\ch14_customcamera`

拍照的核心类是 `android.hardware.Camera`, 通过 `Camera.open` 方法可以获得 `Camera` 对象, 并通过 `Camera.startPreview` 方法开始拍照, 最后通过 `Camera` 类的 `takePicture` 方法结束拍照, 并在相应的事件中处理照片数据。

在拍照之前, 还需要做如下的准备工作。

- 指定用于显示拍照过程影像的容器, 一般为 `SurfaceView` 对象。
- 在拍照过程中涉及一些状态的变化, 这些状态包括开始拍照 (对应 `surfaceCreated` 方法); 拍照状态变化 (例如图像格式或方向, 对应 `surfaceChanged` 方法); 结束拍照 (对应 `surfaceDestroyed` 方法)。这 3 个方法都是在 `SurfaceHolder.Callback` 接口中定义的, 因此, 需要使用 `SurfaceHolder.addCallback` 方法指定 `SurfaceHolder.Callback` 对象, 以便捕捉这 3 个事件。
- 拍完照后需要处理照片数据。处理这些数据的工作需要在 `PictureCallback.onPictureTaken` 方法中完成, 当调用 `Camera` 类的 `takePicture` 方法后, `onPictureTaken` 事件方法被调用。
- 如果需要自动对焦, 需要调用 `Camera.autoFocus` 方法。该方法需要一个 `AutoFocusCallback` 类型的参数值。`AutoFocusCallback` 是一个接口, 在该接口中定义了一个 `onAutoFocus` 方法, 摄像头正在对焦或对焦成功都会调用该方法。

为了使拍照功能更容易使用, 本节的例子将拍照功能封装在了 `Preview` 类中, 代码如下:

```
// CameraPreview 类中的静态方法, 用于根据手机方向获得相机预览画面旋转的角度
public static int getPreviewDegree(Activity activity)
{
    // 获得手机的方向
    int rotation = activity.getWindowManager().getDefaultDisplay().getRotation();
    int degree = 0;
    // 根据手机的方向计算相机预览画面应该旋转的角度
    switch (rotation)
    {
        case Surface.ROTATION_0:
            degree = 90;
        case Surface.ROTATION_90:
            degree = 0;
        case Surface.ROTATION_180:
            degree = 270;
        case Surface.ROTATION_270:
            degree = 180;
    }
    return degree;
}
```

```

        break;
    case Surface.ROTATION_90:
        degree = 0;
        break;
    case Surface.ROTATION_180:
        degree = 270;
        break;
    case Surface.ROTATION_270:
        degree = 180;
        break;
    }
    return degree;
}
class Preview extends SurfaceView implements SurfaceHolder.Callback
{
    private SurfaceHolder holder;
    private Camera camera;
    // 创建一个 PictureCallback 对象，并实现其中的 onPictureTaken 方法
    private PictureCallback pictureCallback = new PictureCallback()
    {
        // 该方法用于处理拍摄后的照片数据
        @Override
        public void onPictureTaken(byte[] data, Camera camera)
        {
            // data 参数值就是照片数据，将这些数据以 key-value 形式保存，以便其他调用该 Activity 的程序可
            // 以获得照片数据
            getIntent().putExtra("bytes", data);
            setResult(20, getIntent());
            // 停止照片拍摄
            camera.stopPreview();
            camera = null;
            // 关闭当前的 Activity
            finish();
        }
    };
    // Preview 类的构造方法
    public Preview(Context context)
    {
        super(context);
        // 获得 SurfaceHolder 对象
        holder = getHolder();
        // 指定用于捕捉拍照事件的 SurfaceHolder.Callback 对象
        holder.addCallback(this);
        // 设置 SurfaceHolder 对象的类型
        holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    // 开始拍照时调用该方法
    public void surfaceCreated(SurfaceHolder holder)
    {
        // 获得 Camera 对象
        camera = Camera.open();
        try
        {

```

```
// 设置用于显示拍照影像的 SurfaceHolder 对象
camera.setPreviewDisplay(holder);
}
catch (IOException exception)
{
    // 释放手机摄像头
    camera.release();
    camera = null;
}
}

// 停止拍照时调用该方法
public void surfaceDestroyed(SurfaceHolder holder)
{
    // 释放手机摄像头
    camera.release();
}

// 拍照状态变化时调用该方法
public void surfaceChanged(final SurfaceHolder holder, int format, int w, int h)
{
    try
    {
        Camera.Parameters parameters = camera.getParameters();
        // 设置照片格式
        parameters.setPictureFormat(PixelFormat.JPEG);

        // 设置相机预览的尺寸
        parameters.setPreviewSize(w, h);
        // 设置保存的图像尺寸
        parameters.setPictureSize(w, h);
        camera.setParameters(parameters);
        // 设置相机预览时的方向（当手机 360° 旋转时，始终保持预览画面的正常）
        camera.setDisplayOrientation(getPreviewDegree(CameraPreview.this));
        // 开始拍照
        camera.startPreview();
        // 准备用于表示对焦状态的图像（类似如图 14.8 所示的对焦符号）
        ivFocus.setImageResource(R.drawable.focus1);
        LayoutParams layoutParams = new LayoutParams(
            LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT);
        ivFocus.setScaleType(ScaleType.CENTER);
        addContentView(ivFocus, layoutParams);
        ivFocus.setVisibility(VISIBLE);
        // 自动对焦
        camera.autoFocus(new AutoFocusCallback()
        {
            @Override
            public void onAutoFocus(boolean success, Camera camera)
            {
                if (success)
                {
                    // success 为 true 表示对焦成功，改变对焦状态图像（一个绿色的 png 图像）
                    ivFocus.setImageResource(R.drawable.focus2);
                }
            }
        });
    }
}
```

```

        });
    }
    catch (Exception e)
    {
    }
}
// 停止拍照，并将拍摄的照片传入 PictureCallback 接口的 onPictureTaken 方法
public void takePicture()
{
    if (camera != null)
    {
        camera.takePicture(null, null, pictureCallback);
    }
}
}
}

```

在编写 Preview 类时应注意如下几点。

- Camera.takePicture 方法有 3 个参数，都是回调对象，但比较常用的是最后一个参数。当拍照后会调用该参数指定对象中的 onPictureTaken 方法，一般可以在该方法中对照片数据做进一步处理。例如，在本例中使用 putExtra 方法以 key-value 对保存了照片数据。
- 当手机摄像头的状态变化时，例如手机由纵向变成横向，或分辨率发生变化后，很多参数需要重新设置，这时系统就会调用 SurfaceHolder.Callback 接口的 surfaceChanged 方法。因此，可以在该方法中对摄像头的参数进行设置，包括调用 startPreview 方法进行拍照。
- 根据手机的方向（0°、90°、180°、270°），需要设置相机预览画面的角度。否则，当手机方向改变时，相机预览画面可能倒过来，或旋转 90°。
- 如果想设置照片的实际分辨率，需要使用 Camera.Parameters.setPictureSize 方法进行设置。
- 本例中通过在 CameraActivity 中添加 ImageView 的方式在预览界面显示了一个表示对焦状态的图像。这个图像文件有两个：focus1.png 和 focus2.png。其中 focus1.png 是白色的透明图像，表示正在对焦。focus2.png 是绿色的透明图像，表示对焦成功。在开始拍照后，先显示 focus1.png，当对焦成功后，系统会调用 AutoFocusCallback.onAutoFocus 方法。在该方法中将 ImageView 中显示的图像变成 focus2.png，表示对焦成功，这时就可以进行拍照了。
- 在拍完照后需要调用 Camera.release 方法释放手机摄像头，否则如果不重启手机，其他的应用程序无法再使用摄像头进行拍照。

本例通过触摸拍照预览界面进行拍照。因此，需要使用 Activity.onTouchEvent 方法来处理屏幕触摸事件，代码如下：

```

public boolean onTouchEvent(MotionEvent event)
{
    if (event.getAction() == MotionEvent.ACTION_DOWN)
        // 拍照
        preview.takePicture();
    return super.onTouchEvent(event);
}

```

在编写完 CameraPreview 类后，可以在其他的类中使用如下代码显示 CameraPreview，显示 CameraPreview 后会自动进行拍照：

```
Intent intent = new Intent(this, CameraPreview.class);
startActivityForResult(intent, 1);
```

在关闭 CameraPreview 后（可能是拍照成功，也可能是取消拍照），可以通过 onActivityResult 方法来获得成功拍照后的照片数据，代码如下：

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == 1)
    {
        // 拍照成功后，响应码是 20
        if (resultCode == 20)
        {
            Bitmap cameraBitmap;
            // 获得照片数据（byte 数组形式）
            byte[] bytes = data.getExtras().getByteArray("bytes");
            // 将 byte 数组转换成 Bitmap 对象
            cameraBitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
            // 根据拍摄的方向旋转图像（纵向拍摄时需要将图像旋转 90 度）
            Matrix matrix = new Matrix();
            matrix.setRotate(CameraPreview.getPreviewDegree(this));
            cameraBitmap = Bitmap.createBitmap(cameraBitmap, 0, 0,
                cameraBitmap.getWidth(), cameraBitmap.getHeight(), matrix, true);
            // 将照片保存在 SD 卡的根目录（文件名是 camera.jpg）
            File myCaptureFile = new File("/sdcard/camera.jpg");
            try
            {
                BufferedOutputStream bos = new BufferedOutputStream(
                    new FileOutputStream(myCaptureFile));
                cameraBitmap.compress(Bitmap.CompressFormat.JPEG, 100, bos);
                bos.flush();
                bos.close();
                imageView.setImageBitmap(cameraBitmap);
            }
            catch (Exception e)
            {
            }
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```

在编写上面代码时应注意如下几点。

- 由于手机方向不同，拍出的照片方向也不同。例如，如果手机纵向拍摄，拍出的照片是横向的，因此，需要后期对照片进行旋转处理，以恢复照片正常的效果。
- 由于直接使用 Camera 类进行拍照时，系统不会自动保存照片，因此，就需要在处理照片时自行确定照片的存储位置，并保存照片。这种方法的优点是灵活，缺点是需要写更多的代码。至于是选择系统提供的拍照功能，还是选择自己实现拍照功能，可根据具体的情况而定。如果对照片保存的位置没什么要求，而且对照片的分辨率要求不高。可以使用系统提供的拍照功能，否则，就要自己来实现拍照功能了。

最后需要在 `AndroidManifest.xml` 文件中设置拍照的权限许可（在调用系统提供的拍照功能时并不需要设置拍照权限许可），代码如下：

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

## 14.4 铃声

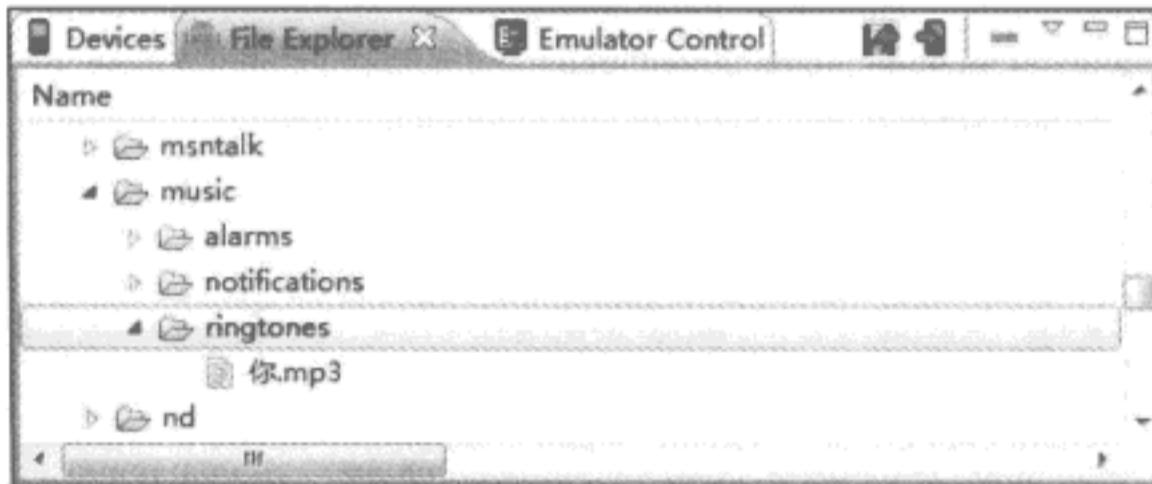
工程目录：src\ch14\ch14\_ringtone

经常有很多朋友会下载一些 mp3 音乐作为来电、闹铃、通知的声音。Android 提供了 `RingtoneManager` 类来专门操作各种铃声，因此，通过程序也可以对系统的这些铃声进行设置。

Android 本身提供了一些默认的铃声文件，这些文件都放在 “/system/media/audio” 目录中。如果是我们自己下载的铃声文件（一般为 mp3 音频文件），可以放在 SD 卡的 music 目录。铃声主要有 3 种：一般的铃声（如来电铃声）、闹钟铃声和通知铃声。这 3 种铃声要分别放在如下的目录。

- `/sdcard/music/ringtones`: 一般的铃声，如来电铃声。
- `/sdcard/music/alarms`: 闹钟铃声。
- `/sdcard/music/notifications`: 警告或通知铃声。

下面我们先下载一个 mp3 文件，并将其放到“`/sdcard/music/ringtones`”目录中，如图 14.8 所示。



▲ 图 14.8 复制 mp3 文件到 ringtones 目录

设置系统的铃声首先要通过 `Activity Action` 调用铃声设置界面，与铃声设置界面对应的 `Activity Action` 是 `RingtoneManager.ACTION_RINGTONE_PICKER`。可通过如下 3 个常量来设置不同类型的铃声。

- `RingtoneManager.TYPE_RINGTONE`: 来电铃声。
- `RingtoneManager.TYPE_ALARM`: 闹钟铃声。
- `RingtoneManager.TYPE_NOTIFICATION`: 通知铃声。

在选择完要设置的铃声后，需要调用 `RingtoneManager.setActualDefaultRingtoneUri` 方法设置新的铃声。

下面的代码分别显示了上述 3 种铃声的设置界面，这 3 个界面基本相同，只是会从不同的目录装载音频文件。

显示设置来电铃声的界面，如图 14.9 所示。

```
Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
```

```
// 设置铃声类型
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
    RingtoneManager.TYPE_RINGTONE);
// 设置界面标题
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置来电铃声");
startActivityForResult(intent, 1);
```

显示设置闹钟铃声的界面，如图 14.10 所示。

```
Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
// 设置铃声类型
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
    RingtoneManager.TYPE_ALARM);
// 设置界面标题
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置闹铃声音");
startActivityForResult(intent, 2);
```

显示设置通知铃声的界面，如图 14.11 所示。

```
Intent intent = new Intent(RingtoneManager.ACTION_RINGTONE_PICKER);
// 设置铃声类型
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE,
    RingtoneManager.TYPE_NOTIFICATION);
// 设置界面标题
intent.putExtra(RingtoneManager.EXTRA_RINGTONE_TITLE, "设置通知铃声");
startActivityForResult(intent, 3);
```



▲ 图 14.9 设置来电铃声



▲ 图 14.10 设置闹铃铃声



▲ 图 14.11 设置通知铃声

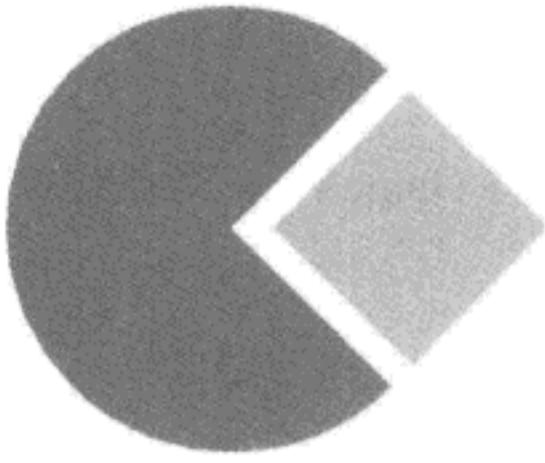
选中要设置的铃声后，单击“确定”按钮，会调用 `onActivityResult` 方法设置相应的铃声，代码如下：

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

```
super.onActivityResult(requestCode, resultCode, data);
if (resultCode != RESULT_OK)
{
    return;
}
// 获得选中铃声的Uri
Uri uri = data.getParcelableExtra(RingtoneManager.EXTRA_RINGTONE_PICKED_URI);
if (uri != null)
{
    switch (requestCode)
    {
        case 1: // 设置来电铃声
            RingtoneManager.setActualDefaultRingtoneUri(this,
                RingtoneManager.TYPE_RINGTONE, uri);
            break;
        case 2: // 设置闹铃铃声
            RingtoneManager.setActualDefaultRingtoneUri(this,
                RingtoneManager.TYPE_ALARM, uri);
            break;
        case 3: // 设置通知铃声
            RingtoneManager.setActualDefaultRingtoneUri(this,
                RingtoneManager.TYPE_NOTIFICATION, uri);
    }
}
```

## 14.5 小结

本章主要介绍了 Android SDK 提供的多媒体 API，主要四大块：音频、视频、拍照、铃声，这几部分是最基础的多媒体 API，在很多应用中都会用得到，希望读者根据本书的内容仔细地学习和实践。



## 第 15 章 2D 游戏开发

游戏是 Android 中最大众化的应用，大多数的 Android 手机用户都或多或少地接触过一款或几款不同类型的游戏。从游戏的宏观表现形式看，分为 2D 和 3D 游戏，其中 2D 游戏主要通过画布（Canvas）来绘制各种游戏元素，而 3D 游戏通常会借助 OpenGL ES 及更高级的游戏引擎进行渲染，效果更逼真，可玩性更强。关于 3D 游戏的部分将后面详细讨论，而本章主要介绍编写 2D 游戏需要掌握的基础知识及一些技巧。

### 15.1 绘制游戏的画布

编写过游戏或图形处理程序的读者都会知道，无论使用哪种开发语言，都会有画布（Canvas）或类似的概念。通过 Canvas 可以绘制很多基本的图形元素，例如，像素点、直线、圆、矩形等。Android SDK 为我们提供了两个可操作 Canvas 的类：View 和 SurfaceView，这两个类都可以获得并操作 Canvas。

#### 15.1.1 在 View 上实现动画效果

工程目录：src\ch15\ch15\_game\_view

前面曾经多次提到 View 类，该类是很多控件的父类。通过覆盖 View.onDraw 方法，可以利用 onDraw 方法的 canvas 参数获得 Canvas 对象，然后可以通过 Canvas 对象中的方法绘制各种基本的图形。通过 View.invalidate 方法刷新 View，并调用 onDraw 方法重绘画布，这时画布中所有绘制的图形都会被清空。因此，需要根据新的数据重绘所有的图形。

为了演示如何通过 View 实现动画，本节给出一个有趣的例子。本例启动时会在屏幕正中心绘制一个白色实心的圆，当鼠标或手指触摸到屏幕的其他位置时，实心圆会以一定的速度向触摸的地方移动，直到圆心和触摸点重合。

实现这个动画效果的步骤如下。

- (1) 编写一个类（GameView），该类继承自 View。
- (2) 覆盖 View.onDraw 方法，并在该方法中根据当前的圆心绘制白色实心圆。
- (3) 为了截获触摸事件，GameView 类需要实现 OnTouchListener 接口。
- (4) 通常动画效果要用多线程来处理，因此，需要编写一个处理动画效果的线程类（AnimThread）。
- (5) 在 AnimThread.run 方法中实现动画效果。最常用的方法是在循环中不断改变圆心坐标，然后调用 View.invalidate 方法重绘画布，并且在每次改变圆心坐标后进行一定时间的延迟。

(6) 在 OnTouchListener.onTouch 方法中处理触摸事件。当触摸屏幕的某处时，首先要获得当前触摸点的坐标，然后通过 Thread.start 方法启动线程以便开始动画。

(7) 在 Activity 类中创建 GameView 对象，并使用 setContentView 方法将 GameView 对象放在 Activity 上。

现在我们先来看一下 GameView 类的代码。

```
package mobile.android.ch15.game.view;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Handler;
import android.os.Message;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;

public class GameView extends View implements OnTouchListener
{
    public float x;          // 圆心当前横坐标
    public float y;          // 圆心当前纵坐标
    public GameView(Context context)
    {
        super(context);
        // 设置 GameView 对象的触摸事件监听器
        setOnTouchListener(this);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        Paint paint = new Paint();
        paint.setColor(Color.WHITE);
        // 绘制白色实心圆
        canvas.drawCircle(x, y, 20, paint);
    }
    @Override
    public boolean onTouch(View view, MotionEvent event)
    {
        // 当触摸屏幕的某处时，开始移动白色实心圆的动画
        AnimThread animThread = new AnimThread(this, event.getX(), event.getY());
        Thread thread = new Thread(animThread);
        thread.start();
        return false;
    }
    private Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 刷新画布，重绘实心圆
            ((View) msg.obj).invalidate();
            super.handleMessage(msg);
        }
    }
}
```

```
}

};

// 处理动画的线程类
class AnimThread implements Runnable
{
    private float newX, newY;      // 保存触摸点的横纵坐标
    private View view;
    public AnimThread(View view, float newX, float newY)
    {
        this.newX = newX;
        this.newY = newY;
        this.view = view;
    }
    @Override
    public void run()
    {
        // 由于触摸点到上一次实心圆的横纵坐标距离并不相等，因此，需要计算出横纵坐标移动的
        // 比例。纵坐标(Y)移动的偏移量等于横坐标(X)移动的偏移量*scale
        float scale = Math.abs(newY - y) / Math.abs(newX - x);
        while (newX != x && newY != y)
        {
            // 向右移动
            if (newX > x)
            {
                x += 1;
                if (newX < x)
                    x = newX;
            }
            // 向左移动
            else if (newX < x)
            {
                x -= 1;
                if (newX > x)
                    x = newX;
            }
            // 向下移动
            if (newY > y)
            {
                y += scale;
                if (newY < y)
                    y = newY;
            }
            // 向上移动
            else if (newY < y)
            {
                y -= scale;
                if (newY > y)
                    y = newY;
            }
        }
        try
        {
            Message msg = new Message();
            msg.obj = view;
            // 向Handler发送消息刷新画布，以便重绘实心圆
            handler.sendMessage(msg);
            // 延迟50毫秒
        }
    }
}
```

```
        Thread.sleep(50);
    }
    catch (Exception e)
    {
        }
    }
}
```

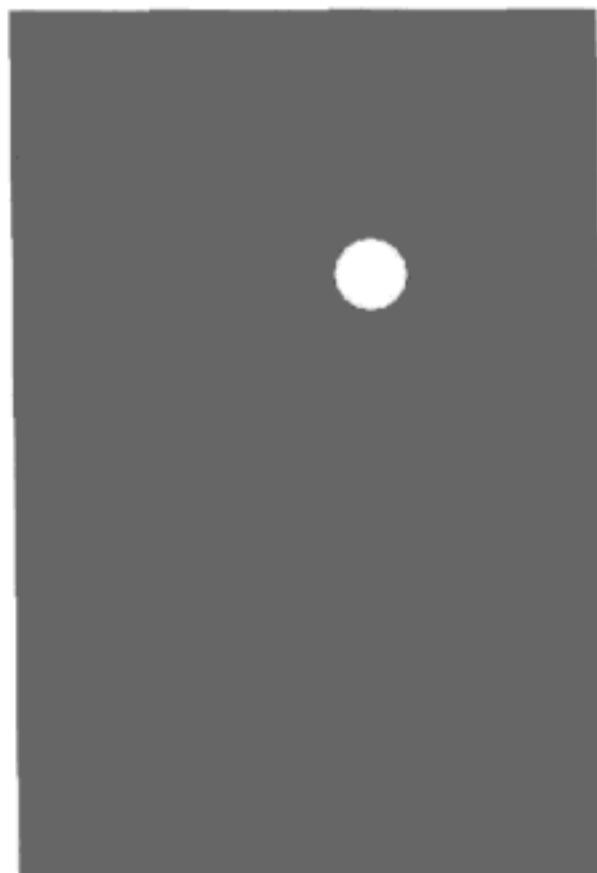
注意

在线程中不能直接访问 UI 控件，也不能调用任何影响 UI 控件变化的方法，例如，本例中调用 `View.invalidate` 方法必须在 `Handler.handleMessage` 方法中完成。

最后一步就是在 Activity 上显示 GameView 对象，代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
    // 全屏显示
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
    super.onCreate(savedInstanceState);
    GameView gameView = new GameView(this);
    // 设置圆心横坐标的初始值（水平居中）
    gameView.x = getWindowManager().getDefaultDisplay().getWidth() / 2;
    // 设置圆心纵坐标的初始值（垂直居中）
    gameView.y = getWindowManager().getDefaultDisplay().getHeight() / 2;
    // 使 GameView 充满整个屏幕
    setContentView(gameView, new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT));
}
```

运行本例后，用鼠标指针单击屏幕的某处，实心圆就会移动到鼠标指针单击的位置，如图 15.1 所示。



▲ 图 15.1 View 上的动画

### 15.1.2 在 SurfaceView 上实现动画效果

工程目录: src\ch15\ch15\_game\_surfaceview

上一小节介绍了 View，并使用 View 实现了一个移动实心圆的动画。虽然 View 在功能上可以实现任何复杂的动画效果，但开发复杂游戏时，View 往往不能满足我们的要求，这时就必须使用 SurfaceView 来代替 View。SurfaceView 与 View 相比有如下特点。

- 使用 SurfaceView 开发时可以直接获得 Canvas 对象，而不像 View，必须要在 onDraw 方法中获得 Canvas 对象。
- SurfaceView 支持双缓冲区技术，绘制图形的效率更高。
- SurfaceView 可以在非 UI 线程中直接绘制图形，而 View 必须要使用 Handler 对象发送消息，通知 UI 线程绘制图形。

如果绘制很复杂的图形时，SurfaceView 的第 2 个和第 3 个特点将大大提高系统的性能。

使用 SurfaceView 和使用 View 类似，必须编写一个继承自 SurfaceView 的类，并且需要一个实现 SurfaceHolder.Callback 接口的类，一般继承自 SurfaceView 的类实现该接口即可。本节将利用 SurfaceView 重新实现上一节移动实心圆的例子。首先建立一个继承自 SurfaceView 的类（GameSurfaceView），该类同时实现了 SurfaceHolderCallback 接口。SurfaceHolder.Callback 接口包含了如下 3 个事件方法。

- surfaceCreated: SurfaceView 对象被创建时触发。
- surfaceChanged: SurfaceView 的大小发生变化时触发。
- surfaceDestroyed: SurfaceView 对象被销毁时触发。

其中后两个事件方法在本例中并未使用，在 surfaceCreated 方法中绘制了初始的实心圆。GameSurfaceView 的完整代码如下：

```
package mobile.android.ch15.game.surfaceview;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.View.OnTouchListener;

public class GameSurfaceView extends SurfaceView implements
    SurfaceHolder.Callback, OnTouchListener
{
    public float x;
    public float y;
    private SurfaceHolder surfaceHolder;

    public GameSurfaceView(Context context)
    {
        super(context);
        surfaceHolder = getHolder();
        surfaceHolder.setFormat(PixelFormat.TRANSLUCENT);
        surfaceHolder.addCallback(this);
        surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    @Override
    protected void onDraw(Canvas canvas)
    {
        Paint paint = new Paint();
        paint.setColor(Color.BLUE);
        paint.setStyle(Paint.Style.FILL);
        paint.setStrokeWidth(2);
        canvas.drawCircle(x, y, 50, paint);
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder)
    {
        Thread thread = new Thread()
        {
            public void run()
            {
                while(true)
                {
                    try
                    {
                        sleep(100);
                    }
                    catch(InterruptedException e)
                    {
                        e.printStackTrace();
                    }
                    if(x < 300)
                    {
                        x += 1;
                    }
                    else
                    {
                        y += 1;
                    }
                    if(y > 300)
                    {
                        break;
                    }
                    draw();
                }
            }
        };
        thread.start();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height)
    {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder)
    {
    }

    private void draw()
    {
        Canvas canvas = surfaceHolder.lockCanvas();
        onDraw(canvas);
        surfaceHolder.unlockCanvasAndPost(canvas);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event)
    {
        return false;
    }
}
```

## Android 开发权威指南

```
setOnTouchListener(this);
// 实例化 SurfaceHolder
surfaceHolder = this.getHolder();
// 添加回调
surfaceHolder.addCallback(this);
this.setFocusable(true);
}
@Override
public void surfaceCreated(SurfaceHolder holder)
{
    // 绘制初始的实心圆
    drawCircle();
}
@Override
public void surfaceChanged(SurfaceHolder holder, int format, int width,
                           int height)
{
}
@Override
public void surfaceDestroyed(SurfaceHolder holder)
{
}
@Override
public boolean onTouch(View view, MotionEvent event)
{
    // 创建处理动画效果的线程
    AnimThread animThread = new AnimThread( event.getX(), event.getY());
    Thread thread = new Thread(animThread);
    thread.start();
    return false;
}
class AnimThread implements Runnable
{
    private float newX, newY;

    public AnimThread(float newX, float newY)
    {
        this.newX = newX;
        this.newY = newY;
    }
    @Override
    public void run()
    {
        float scale = Math.abs(newY - y) / Math.abs(newX - x);
        while (newX != x && newY != y)
        {
            if (newX > x)
            {
                x += 1;
                if (newX < x)
                    x = newX;
            }
            else if (newX < x)
            {
```

```
x -= 1;
if (newX > x)
    x = newX;
}
if (newY > y)
{
    y += scale;
    if (newY < y)
        y = newY;
}
else if (newY < y)
{
    y -= scale;
    if (newY > y)
        y = newY;
}
try
{
    // 在线程中直接绘制实心圆
    drawCircle();
    Thread.sleep(50);
}
catch (Exception e)
{
}
}
}

public void drawCircle()
{
    if (surfaceHolder == null)
        return;
    // 获得 Canvas 对象
    Canvas canvas = surfaceHolder.lockCanvas();
    // 清空屏幕
    canvas.drawColor(Color.BLACK);
    if (canvas == null)
        return;
    Paint paint = new Paint();
    paint.setColor(Color.WHITE);
    canvas.drawCircle(x, y, 20, paint);
    // 释放 Canvas 对象
    surfaceHolder.unlockCanvasAndPost(canvas);
}
```

从上面代码中的 `drawCircle` 方法中可以看出，在 `SurfaceView` 上绘制图形的步骤如下。

- (1) 使用 `SurfaceHolder.lockCanvas` 方法获得 `Canvas` 对象。
- (2) 如果需要清空屏幕，调用 `Canvas.drawColor` 方法。
- (3) 调用 `Canvas` 对象的方法绘制相应的图形。
- (4) 使用 `SurfaceHolder.unlockCanvasAndPost` 方法释放 `Canvas` 对象，并将缓冲区绘制的图形一次性绘制到画布上。

## 15.2 图形绘制基础

通过 Canvas 对象可以绘制各种基本的图形、文本、位图等。无论使用 View，还是使用 SurfaceView，Canvas 的使用方法都是一样的，本节将详细讨论如何使用 Canvas 绘制基本的图形元素，并在最后给出一个综合绘制图形的例子。

### 15.2.1 绘制像素点

像素点是一切图形元素的基础，使用 Canvas.drawPoint 方法可以在指定坐标绘制一个像素点，或指定一组坐标绘制多个像素点。drawPoint 方法有 3 个重载形式，定义如下：

```
public native void drawPoint(float x, float y, Paint paint);           // 绘制一个像素点
public native void drawPoints(float[] pts, int offset, int count, Paint paint);
public void drawPoints(float[] pts, Paint paint);                         // 绘制多个像素点
```

参数的含义如下。

- x：像素点的横坐标。
- y：像素点的纵坐标。

• paint：描述像素点属性的 Paint 对象。可设置像素点的大小、颜色等属性。绘制其他图形元素的 Paint 对象与绘制像素点的 Paint 对象的含义相同。在绘制具体的图形元素时可根据实际情况设置 Paint 对象。

• pts：drawPoints 方法可一次性绘制多个像素点，pts 参数表示多个像素点的坐标，该数组元素个数必须是偶数，两个一组为一个像素点的坐标。

• offset：drawPoints 方法可以取 pts 数组中的一部分连续元素作为像素点的坐标，因此，需要通过 offset 参数来指定取得数组中连续元素的第 1 个元素的位置，也就是元素偏移量从 0 开始。例如，要从第 3 个元素开始取数组元素，那么 offset 参数的值就是 2。

• count：要获得的数组元素个数。count 必须为偶数（两个数组元素为一个像素点的坐标）。

### 15.2.2 绘制直线

使用 Canvas.drawLine 方法可以绘制一条或多条直线，drawLine 方法有 3 个重载形式，定义如下：

```
public void drawLine(float startX, float startY, float stopX, float stopY, Paint paint);          // 绘制一条直线
public native void drawLines(float[] pts, int offset, int count, Paint paint);                   // 绘制多条直线
public void drawLines(float[] pts, Paint paint);                                                 // 绘制多条直线
```

参数的含义如下。

- startX：直线开始端点的横坐标。

- startY: 直线开始端点的纵坐标。
- stopX: 直线结束端点的横坐标。
- stopY: 直线结束端点的纵坐标。
- pts: 绘制多条直线时的端点坐标集合。4 个数组元素（两个为开始端点的坐标，两个为结束端点的坐标）为 1 组，表示一条直线。例如画两条直线，pts 数组就应该有 8 个元素，前 4 个数组元素为第 1 条直线的两个端点的坐标，后 4 个数组元素为第 2 条直线的两个端点的坐标。
- offset: pts 数组中元素的偏移量。
- count: 取得 pts 数组中元素的个数。该参数值需为 4 的整数倍。

### 15.2.3 绘制圆形

使用 `Canvas.drawCircle` 方法可以绘制圆形，`drawCircle` 方法的定义如下：

```
public void drawCircle(float cx, float cy, float radius, Paint paint);
```

参数的含义如下。

- cx: 圆心的横坐标。
- cy: 圆心的纵坐标。
- radius: 圆的半径。

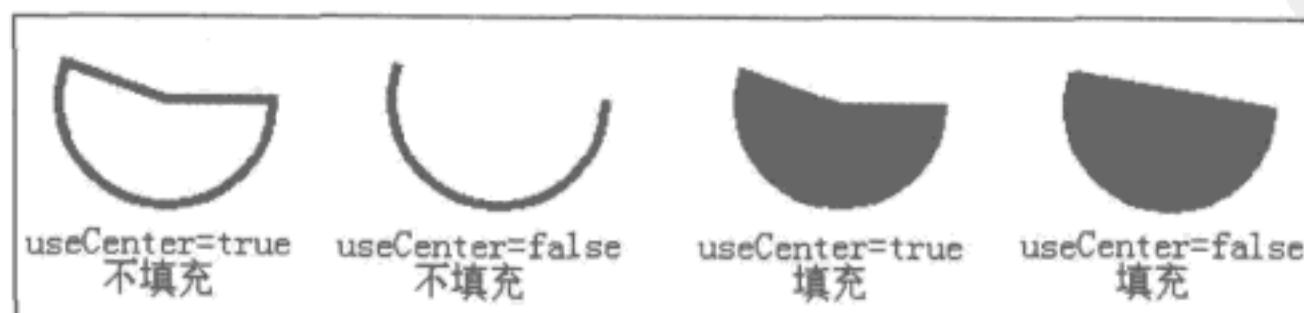
### 15.2.4 绘制弧

使用 `Canvas.drawArc` 方法可以绘制弧，`drawArc` 方法的定义如下：

```
public void drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter,
Paint paint);
```

参数的含义如下。

- oval: 弧的外切矩形的坐标。需要设置该矩形的左上角和右下角的坐标，也就是 `oval.left`、`oval.top`、`oval.right` 和 `oval.bottom`。
- startAngle: 弧的起始角度。
- sweepAngle: 弧的结束角度。如果 `sweepAngle-startAngle` 的绝对值大于等于 360，`drawArc` 画的就是一个圆或椭圆（如果是 `oval` 指定的坐标，画出来的是长方形）。
- useCenter: 如果该参数值为 `true`，在画弧时弧的两个端点会连接圆心。如果该参数值为 `false`，则只会画弧，效果如图 15.2 所示。前两个弧未设置填充状态，后两个弧设置了填充状态。关于填充状态的设置方法将在 15.2.6 小节详细介绍。



▲ 图 15.2 填充和设置 `useCenter` 参数的效果

### 15.2.5 绘制文本

我们可以使用 `Canvas.drawText` 或 `Canvas.drawPosText` 方法绘制文本，这两个方法的定义如下：

```
// 绘制 text 指定的文本
public native void drawText(String text, float x, float y, Paint paint);
// 绘制 text 指定的文本。文本中的每一个字符的起始坐标由 pos 数组中的值决定
public void drawPosText(String text, float[] pos, Paint paint);
// 绘制 text 指定的文本。text 中的每一个字符的起始坐标由 pos 数组中的值决定，并且可以选择 text 中的某一段
// 连续的字符绘制
public void drawPosText(char[] text, int index, int count, float[] pos, Paint paint);
```

参数的含义如下。

- `text`: `drawText` 方法中的 `text` 参数表示要绘制的文本。`drawPostText` 方法中的 `text` 虽然也表示要绘制的文本，但每一个字符的坐标需要单独指定。如果未指定某个字符的坐标，系统会抛出异常。
- `x`: 绘制文本的起始点的横坐标。
- `y`: 绘制文本的起始点的纵坐标。
- `index`: 选定的字符集合在 `text` 数组中的索引。
- `count`: 选定的字符集中的字符个数。

### 15.2.6 综合绘制各种图形

工程目录: `src\ch15\ch15_draw`

本例在 `View.onDraw` 方法中绘制图形，与使用 `SurfaceView` 绘制图形的方法类似。首先需要一个继承自 `View` 的 `MyView` 类，并在 `MyView` 类中覆盖 `onDraw` 方法。在本例中绘制了像素点、直线、正方形、三角形、圆形、弧、椭圆和文本。

当 `View` 重绘时会调用 `View.onDraw` 方法，通过调用 `View.invalidate` 方法可以重绘 `View`。在 15.2.2 小节曾介绍过，绘制多条直线的 `drawLines` 方法需要为每一条直线指定两个点的坐标，共 4 个值。如果要绘制 10 条直线，就需要指定 40 个值。如果要绘制三角形、梯形、五角星这样的直线端点重合的图形，就需要很多点的坐标值。这些工作很多都是可以避免的，例如，三角形的每一条边的终点就是另一条边的起点。如果用 `drawLines` 方法，就需要为这个三角形设置 6 个坐标（12 个值），而其中有 3 个坐标是多余的。使用 `drawLines` 方法绘制三角形的代码如下：

```
Paint paint = new Paint();
canvas.drawLines(new float[]{ 160, 70, 230, 150, 230, 150, 170, 155, 170, 155, 160, 70 },
paint);
```

从上面的代码可以看出，`drawLines` 方法的第一个参数值包含 12 个值（6 个坐标）。黑色字体部分并不是必需的。这部分值与前面的某组坐标是相同的，如果绘制的直线是首尾相接，完全可以将这些坐标省略。为此，我们编写了一个 `drawLinesExt` 方法来省略这些不必要的坐标值，代码如下：

```
// pts 只要设置图形的顶点坐标即可，例如，三角形只需要设置 3 个顶点坐标
private void drawLinesExt(Canvas canvas, float[] pts, Paint paint)
{
    for (int i = 0; i < pts.length; i += 2)
```

```

    {
        int stopXIndex = i + 2;
        int stopYIndex = i + 3;
        // 设置最后一个点的横坐标索引为起始点的横坐标索引
        if (stopXIndex > pts.length - 1)
            stopXIndex = 0;
        // 设置最后一个点的纵坐标索引为起始点的纵坐标索引
        if (stopYIndex > pts.length - 1)
            stopYIndex = 1;
        canvas.drawLine(pts[i], pts[i + 1], pts[stopXIndex], pts[stopYIndex], paint);
    }
}

```

`drawLinesExt`方法的基本原理很简单，通过设置多边形顶点坐标，然后使用`drawLine`方法在相邻顶点之间绘制直线即可。使用`drawLinesExt`方法画三角形的代码如下：

```
// 少了3个坐标(6个值)
drawLinesExt(canvas, new float[]{ 160, 70, 230, 150, 170, 155}, paint);
```

本例使用触摸事件(`onTouchEvent`)来控制View的刷新。当触摸屏幕时，程序会改变`Paint`、`useCenter`等参数的值，并用`invalidate`方法来刷新View。`MyView`类继承自View，用于绘制各种图形，代码如下：

```

class MyView extends View
{
    private Paint paint1 = new Paint();
    private Paint paint2 = new Paint();
    private Paint paint3 = new Paint();
    private boolean useCenter = true;
    // 用于设置绘制文本的字体大小(5个文本)
    private float[] textSizeArray = new float[]{ 15, 18, 21, 24, 27 };
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 根据useCenter来判断当前的状态
        if (useCenter)
        {
            useCenter = false;
            // 设置画笔的颜色
            paint1.setColor(Color.RED);
            paint2.setColor(Color.BLACK);
            paint3.setColor(Color.GREEN);
            // 设置画笔的宽度
            paint1.setStrokeWidth(6);
            paint2.setStrokeWidth(4);
            paint3.setStrokeWidth(2);
        }
        else
        {
            useCenter = true;
            // 设置画笔的颜色
            paint1.setColor(Color.BLACK);
        }
    }
}

```

```

        paint2.setColor(Color.RED);
        paint3.setColor(Color.BLUE);
        // 设置画笔的宽度
        paint1.setStrokeWidth(2);
        paint2.setStrokeWidth(4);
        paint3.setStrokeWidth(6);
    }
    // 每次触摸屏幕时将字体大小倒置，也就是将第 1 个和第 n 个元素交换，第 2 个和第 n-1 个元素交换，依此类推
    for (int i = 0; i < textSizeArray.length / 2; i++)
    {
        float textSize = textSizeArray[i];
        textSizeArray[i] = textSizeArray[textSizeArray.length - i - 1];
        textSizeArray[textSizeArray.length - i - 1] = textSize;
    }
    // 刷新 View
    invalidate();
    return super.onTouchEvent(event);
}
public MyView(Context context)
{
    super(context);
    setBackgroundColor(Color.WHITE);
    paint1.setColor(Color.BLACK);
    paint1.setStrokeWidth(2);
    paint2.setColor(Color.RED);
    paint2.setStrokeWidth(4);
    paint3.setColor(Color.BLUE);
    paint3.setStrokeWidth(6);
}
// 扩展画多条直线的方法
private void drawLinesExt(Canvas canvas, float[] pts, Paint paint)
{
    for (int i = 0; i < pts.length; i += 2)
    {
        int stopXIndex = i + 2;
        int stopYIndex = i + 3;
        // 设置最后一个点的横坐标索引为起始点的横坐标索引
        if (stopXIndex > pts.length - 1)
            stopXIndex = 0;
        // 设置最后一个点的纵坐标索引为起始点的纵坐标索引
        if (stopYIndex > pts.length - 1)
            stopYIndex = 1;
        canvas.drawLine(pts[i], pts[i + 1], pts[stopXIndex], pts[stopYIndex], paint);
    }
}
@Override
protected void onDraw(Canvas canvas)
{
    // 绘制像素点
    canvas.drawPoint(60, 120, paint3);
    canvas.drawPoint(70, 130, paint3);
    canvas.drawPoints(new float[]{ 70, 140, 75, 145, 75, 160 }, paint2);
    // 绘制直线
}

```

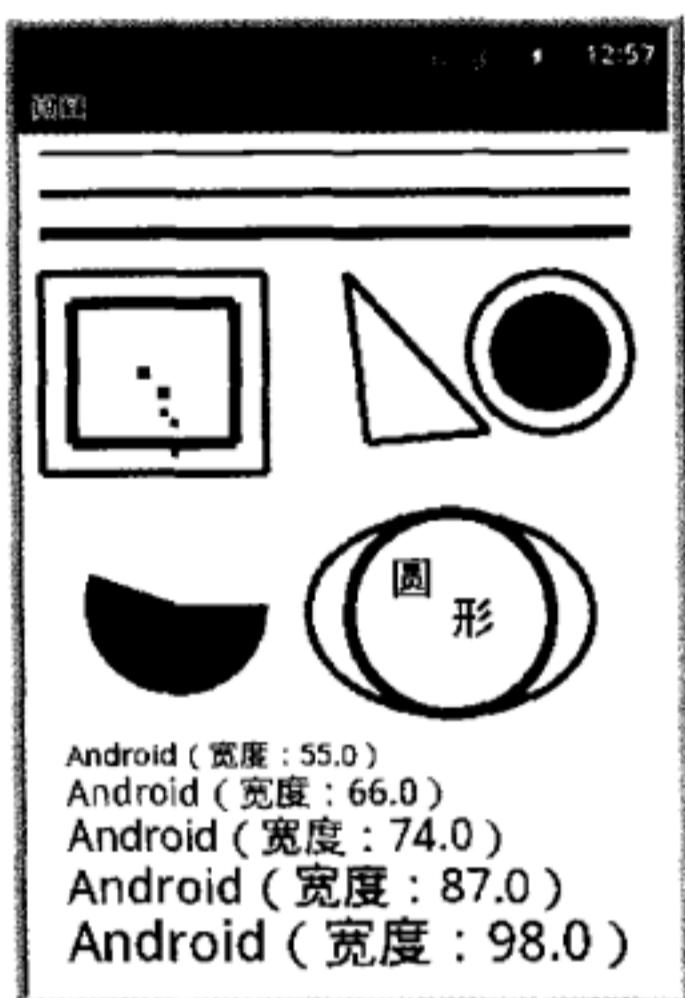
```
canvas.drawLine(10, 10, 300, 10, paint1);
canvas.drawLine(10, 30, 300, 30, paint2);
canvas.drawLine(10, 50, 300, 50, paint3);
// 绘制正方形
drawLinesExt(canvas, new float[]{ 10, 70, 120, 70, 120, 170, 10, 170}, paint2);
drawLinesExt(canvas, new float[]{ 25, 85, 105, 85, 105, 155, 25, 155}, paint3);
// 绘制三角形
drawLinesExt(canvas, new float[]{ 160, 70, 230, 150, 170, 155}, paint2);
// 设置非填充状态
paint2.setStyle(Style.STROKE);
// 画空心圆
canvas.drawCircle(260, 110, 40, paint2);
// 设置填充状态
paint2.setStyle(Style.FILL);
// 画实心圆
canvas.drawCircle(260, 110, 30, paint2);
RectF rectF = new RectF();
rectF.left = 30;
rectF.top = 190;
rectF.right = 120;
rectF.bottom = 280;
// 画弧
canvas.drawArc(rectF, 0, 200, useCenter, paint2);
rectF.left = 140;
rectF.top = 190;
rectF.right = 280;
rectF.bottom = 290;
paint2.setStyle(Style.STROKE);
// 画空心椭圆
canvas.drawArc(rectF, 0, 360, useCenter, paint2);
rectF.left = 160;
rectF.top = 190;
rectF.right = 260;
rectF.bottom = 290;
paint3.setStyle(Style.STROKE);
// 画空心圆
canvas.drawArc(rectF, 0, 360, useCenter, paint3);
float y = 0;
// 绘制文本
for (int i = 0; i < textSizeArray.length; i++)
{
    paint1.setTextSize(textSizeArray[i]);
    paint1.setColor(Color.BLUE);
    // 获得文本的宽度可以用measureText 方法
    canvas.drawText("Android ( 宽度: " + paint1.measureText("Android")
        + " ) ", 20, 315 + y, paint1);
    y += paint1.getTextSize() + 5;
}
paint1.setTextSize(22);
// 绘制文本, 单独设置每一个字符的坐标。第1个坐标(180,230)是“圆”的坐标,
// 第2个坐标(210,250)是“形”的坐标
canvas.drawPosText("圆形", new float[]{180,230, 210,250}, paint1);
```

```

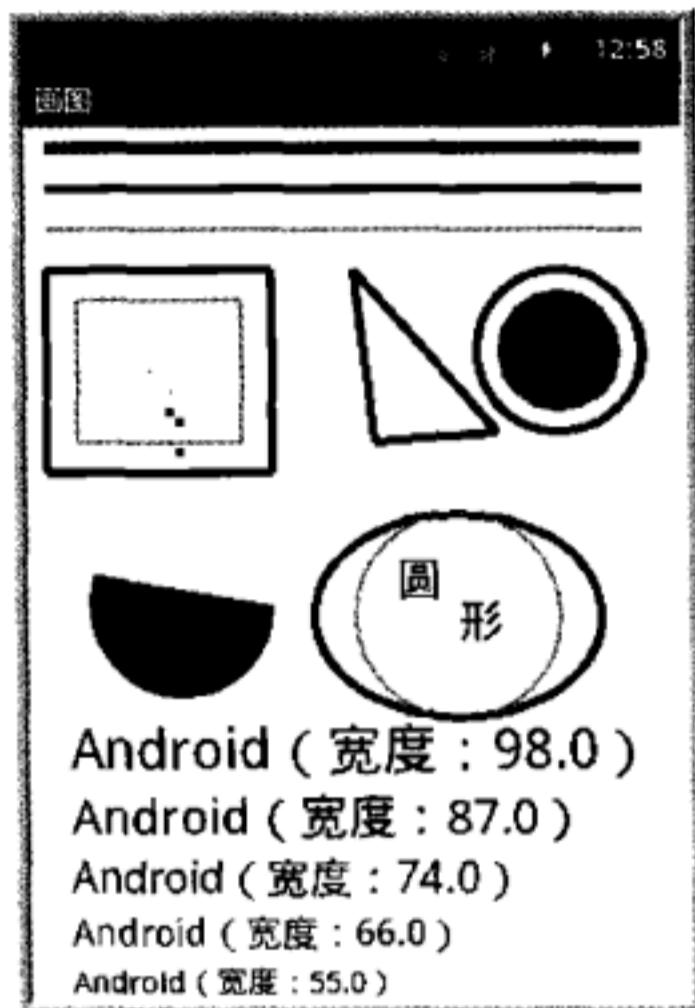
    }
}

```

运行本例，会看到如图 15.3 所示的显示效果，触摸屏幕后，会看到如图 15.4 所示的显示效果。



▲ 图 15.3 绘制图形的效果一



▲ 图 15.4 绘制图形的效果二

## 15.3 高级图像处理技术

本节将介绍一些比较高级的图像技术，这些技术主要包括绘制位图、旋转、路径、图像渲染等。

### 15.3.1 绘制位图

工程目录: `src\ch15\ch15_drawbitmap`

在 Canvas 上绘制位图有如下两种方法。

- 直接绘制 Bitmap 对象；
- 使用 Drawable.draw 方法绘制位图。

直接绘制 Bitmap 对象首先需要创建 Bitmap 对象。通过 BitmapFactory 类的各种方法可以从不同资源创建 Bitmap 对象，例如，可以使用 BitmapFactory.decodeStream 方法从一个 InputStream 创建 Bitmap 对象。

使用 Drawable.draw 方法绘制位图，首先要获得图像资源的 Drawable 对象，然后使用 Drawable.draw 方法绘制位图。

下面给出一个例子来演示这两种绘制位图的方法，代码如下：

```

private static class MyView extends View
{
    private Bitmap bitmap1;
    private Bitmap bitmap2;

```

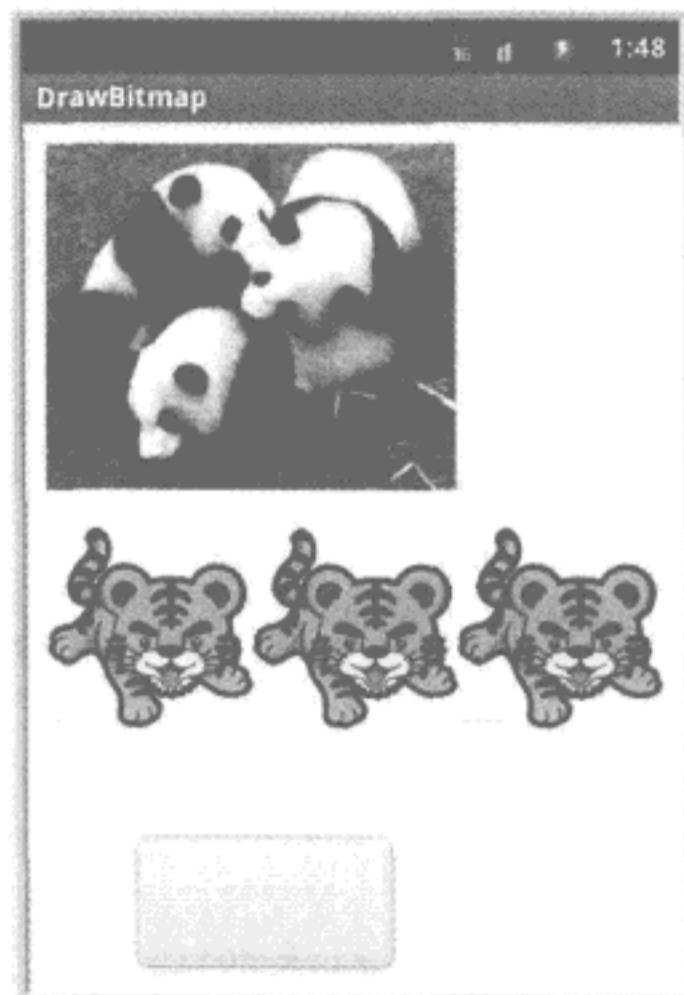
```
private Bitmap bitmap3;
private Bitmap bitmap4;
private Drawable drawable;
public MyView(Context context)
{
    super(context);
    setBackgroundColor(Color.WHITE);
    java.io.InputStream is = context.getResources().openRawResource(R.drawable.panda);
    BitmapFactory.Options opts = new BitmapFactory.Options();
    // 设置图像放缩比例
    opts.inSampleSize = 2;
    bitmap1 = BitmapFactory.decodeStream(is, null, opts);
    is = context.getResources().openRawResource(R.drawable.tiger);
    bitmap2 = BitmapFactory.decodeStream(is);
    int w = bitmap2.getWidth();
    int h = bitmap2.getHeight();
    int[] pixels = new int[w * h];
    // 复制 bitmap2 的所有像素颜色值 (pixels 数组)
    bitmap2.getPixels(pixels, 0, w, 0, 0, w, h);
    // 将 bitmap2 复制两份 (bitmap3 和 bitmap3)
    bitmap3 = Bitmap.createBitmap(pixels, 0, w, h, Bitmap.Config.ARGB_8888);
    bitmap4 = Bitmap.createBitmap(pixels, 0, w, h, Bitmap.Config.ARGB_4444);
    // 获得图像资源的 Drawable 对象
    drawable = context.getResources().getDrawable(R.drawable.button);
    // 设置绘制位图的左上角坐标、宽度和高度
    drawable.setBounds(50, 350, 180, 420);
}
@Override
protected void onDraw(Canvas canvas)
{
    // 用第 1 种方法绘制位图
    canvas.drawBitmap(bitmap1, 10, 10, null);
    canvas.drawBitmap(bitmap2, 10, 200, null);
    canvas.drawBitmap(bitmap3, 110, 200, null);
    canvas.drawBitmap(bitmap4, 210, 200, null);

    // 用第 2 种方法绘制位图
    drawable.draw(canvas);
}
}
```

在编写上面代码时应注意如下几点。

- 本例中的 drawBitmap 方法并不需要设置 Paint 对象，因此将 drawBitmap 方法的最后一个参数值设为 null。
- BitmapFactory.Options.inSampleSize 属性表示原位图与绘制的位图的比例。如果该属性值为 1，表示原位图和绘制的位图的大小比例是 1:1；如果该属性值为 2，表示按原位图 50% (2:1) 的大小绘制位图。

运行本例，显示效果如图 15.5 所示。



▲ 图 15.5 使用两种方法绘制位图

### 15.3.2 图像的透明度

工程目录: src\ch15\ch15\_alpha

Android 系统支持的颜色由 4 个值组成, 前 3 个值为 RGB, 也就是我们常说的三原色(红、绿、蓝), 最后一个值是 A, 也就是 Alpha, 这 4 个值都在 0~255。颜色值越小, 表示该颜色越淡; 颜色值越大, 表示该颜色越深。如果 RGB 都为 0, 就是黑色; 如果 RGB 都为 255, 就是白色。Alpha 也需要在 0~255 变化。Alpha 的值越小, 颜色就越透明; Alpha 的值越大, 颜色就越不透明。当 Alpha 的值为 0 时, 颜色完全透明, 完全透明的位图或图形将从 View 上消失。当 Alpha 的值为 255 时, 颜色不透明。从 Alpha 的特性可知, 设置颜色的透明度实际上就是设置 Alpha 值。

设置颜色的透明度可以通过 Paint.setAlpha 方法来完成。下面来看一个可以调节图像透明度的例子, 在这个例子中通过一个SeekBar 控件改变图像的 Alpha 值(透明度)。显示图像的 MyView 类的代码如下:

```
private class MyView extends View
{
    private Bitmap bitmap;
    public MyView(Context context)
    {
        super(context);
        InputSt ream is = getResources().openRawResource(R.drawable.image);
        bitmap = BitmapFactory.decodeStream(is);
        setBackgroudColor(Color.WHITE);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        Paint paint = new Paint();
        // 设置透明度 (Alpha 值), alpha 是在 Main 类中定义的一个 int 类型的变量
        paint.setAlpha(alpha);
    }
}
```

```
// 绘制位图
canvas.drawBitmap(bitmap, new Rect(0, 0, bitmap.getWidth(), bitmap
    .getHeight()), new Rect(10, 10, 310, 235), paint);
}
}
```

上面代码中的 `drawBitmap` 方法的第 2 个参数表示原位图的复制区域，在本例中需要复制整个原位图。第 3 个参数表示绘制的目标区域。

```
// SeekBar 控件的 onProgressChanged 事件方法的代码如下：
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser)
{
    alpha = progress;
    setTitle("alpha:" + progress);
    // 重绘 View
    myView.invalidate();
}
```

运行本例，将滑杆移动到靠左和靠右的位置，将会看到如图 15.6 和图 15.7 所示的效果。



▲ 图 15.6 透明度为 81 的效果图



▲ 图 15.7 透明度为 212 的效果

### 15.3.3 旋转图像

工程目录: `src\ch15\ch15_rotate_image`

使用 `Matrix.setRotate` 方法可以对图像进行任意角度的旋转。`setRotate` 方法的定义如下：

```
public void setRotate(float degrees)
public void setRotate(float degrees, float px, float py)
```

`setRotate` 方法有两个重载形式。第一个重载形式以图像的中心作为轴心旋转 `degrees` 度。`degrees` 参数值大于 0，图像顺时针旋转；小于 0，则逆时针旋转。`setRotate` 方法的第二个重载形式的 `px` 和 `py` 参数表示图像旋转的轴心坐标。如果使用这个重载形式，图像将以这个坐标为轴心旋转。下面的代码将一个图像顺时针旋转了 45 度，并将旋转后的结果显示在 `ImageView` 控件中。

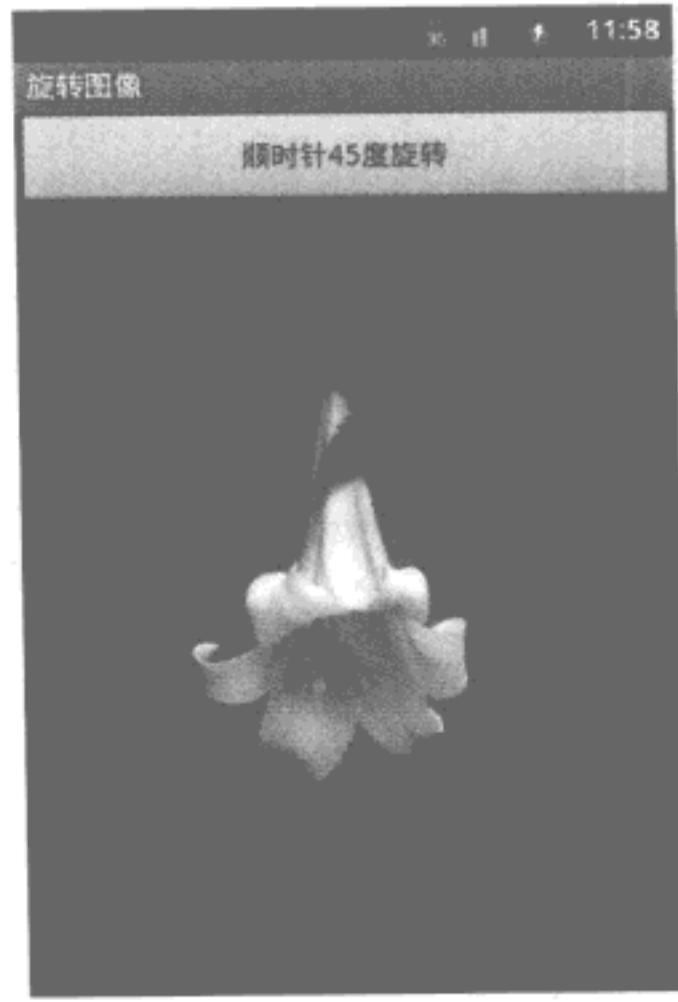
```
ImageView imageView = (ImageView) findViewById(R.id.imageview);
Matrix matrix = new Matrix();
```

```
// 顺时针旋转 45 度
matrix.setRotate(45);
Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);
// 旋转图像，并生成旋转后的 Bitmap 对象
bitmap = Bitmap.createBitmap(bitmap, 0, 0, bitmap.getWidth(), bitmap.getHeight(), matrix,
true);
// 将旋转后的结果显示在 ImageView 控件中
imageView.setImageBitmap(bitmap);
```

旋转前和旋转后的效果如图 15.8 和图 15.9 所示。



▲ 图 15.8 旋转前的效果图



▲ 图 15.9 顺时针旋转后的效果

除此之外，还可以使用 `Canvas.setMatrix` 方法设置 `Matrix` 对象，并直接使用 `drawBitmap` 来绘制旋转后的图像，代码如下：

```
protected void onDraw(Canvas canvas)
{
    Matrix matrix = new Matrix();
    // 设置要旋转的角度（120 度），160 和 240 是图像旋转的轴心坐标
    matrix.setRotate(120, 160, 240);
    canvas.setMatrix(matrix);
    // 在指定位置绘制图像
    canvas.drawBitmap(bitmap, 88, 169, null);
}
```

### 15.3.4 路径

工程目录：src\ch15\ch15\_image\_path

如果读者用过 Photoshop，会对路径的概念很熟悉。在 Photoshop 中通过路径可以画出一个区域，并可以剪切、复制这个区域的图像。路径可以是封闭的，也可以是开放的（由多条线段组成），称为封闭路径或开放路径。

Canvas 类也提供了绘制路径的功能，通过 `Canvas.drawPath` 方法，可以画出封闭路径和开放路径，并可以在路径上实现一些特殊的效果。下面先看看 `drawPath` 方法的定义。

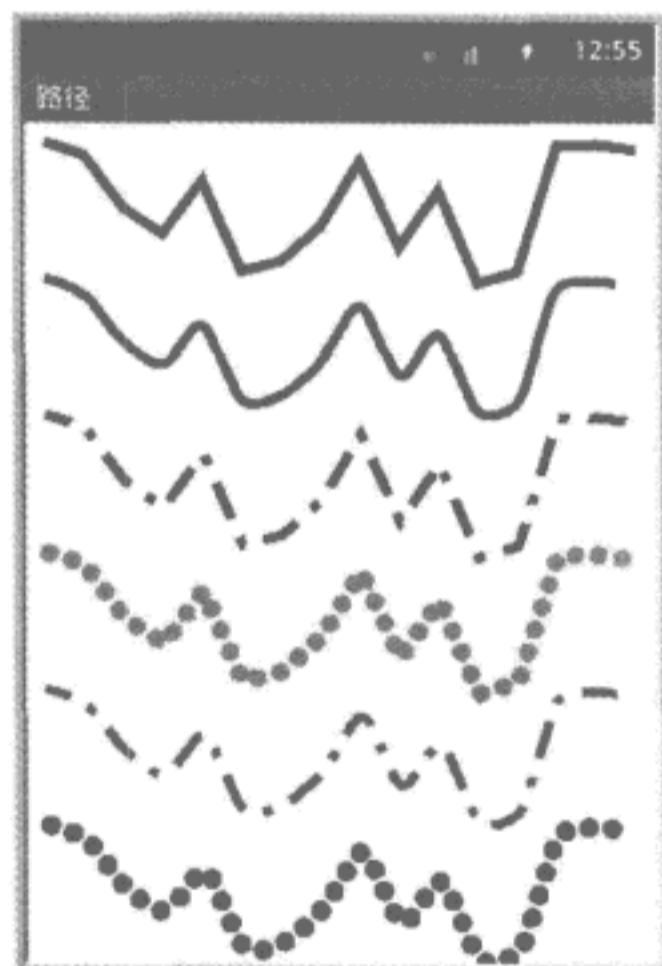
```
| public void drawPath(Path path, Paint paint);
```

`drawPath` 方法需要两个参数：path 和 paint，其中 path 参数非常重要，用于绘制路径的轨迹，例如对于开放路径，需要绘制组成路径的多条线段，如果是封闭路径，需要绘制封闭路径的形状（如圆形、椭圆等）。paint 参数用于指定特效、颜色等路径属性。

本小节将绘制一系列的开放路径。首先创建一个 Path 对象，并绘制组成路径的多条线段，代码如下：

```
private Path makeFollowPath()
{
    // 创建 Path 对象
    Path p = new Path();
    p.moveTo(0, 0);
    for (int i = 1; i <= 15; i++)
    {
        // 随机生成线段的纵坐标，并绘制当前的路径线段
        p.lineTo(i * 20, (float) Math.random() * 70);
    }
    return p;
}
```

本小节绘制的一系列开放路径由不同的特效组成，这些特效如图 15.10 所示。



▲ 图 15.10 开放路径的特效

实现如图 15.10 所示的特效需要创建 `PathEffect` 对象。`PathEffect` 类有很多子类，分别表示不同的特效，下面来创建这些特效对象。

```
| private void makeEffects(PathEffect[] e, float phase)
```

```

    {
        e[0] = null;      // 没有效果
        e[1] = new CornerPathEffect(10);
        e[2] = new DashPathEffect(new float[]{ 20, 10, 5, 10 }, phase);
        e[3] = new PathDashPathEffect(makeCirclePath(), 12, phase, PathDashPathEffect.
            Style.ROTATE);
        e[4] = new ComposePathEffect(e[2], e[1]);
        e[5] = new ComposePathEffect(e[3], e[1]);
    }
}

```

`makeEffects` 方法的 `e` 参数是一个 `PathEffect` 类型的数组，该数组有 6 个元素，其中第 1 个元素没有任何特效（元素值为 `null`），其余的数组元素分别对应 5 个特效对象，这 5 个特效也对应如图 15.10 所示的后 5 个路径。下面来分别解释一下这些特效类的用法。

### 1. CornerPathEffect 类

该类将线段与线段之间的夹角转换成圆角，构造方法的参数表示圆角的半径。

### 2. DashPathEffect 类

该类用于绘制虚线路径。该类的构造方法有两个参数，第 1 个参数表示虚线线段的长度和虚线之间的间隔。该参数值是一个 `float` 数组，数组长度必须是偶数，而且必须大于等于 2，也就是说，指定一条虚线线段的长度后，必须指定该虚线线段与后面的虚线线段的距离。通过第 1 个参数可以指定长度和距离不等的虚线路径，例如绘制如图 15.11 所示的虚线，需要指定 4 个值，`new float[]{10, 4, 6, 4}`。这 4 个值分别是长线段的长度（10）、长线段与短线段的距离（4）、短线段的长度（6）和短线段与长线段的距离（4）。



▲ 图 15.11 不等宽的虚线路径

第 2 个参数表示绘制路径的偏移量。如果该参数值不断增大或减小，会呈现路径向前或向后移动的效果。

### 3. PathDashPathEffect 类

该类与 `DashPathEffect` 的功能类似，但该类的功能更强大，可以单独组成虚线路径（实际上，已经不只是虚线线段了，可以是任何图形）的图形。从 `PathDashPathEffect` 类的名称可以看出，该类名将 `Path` 和 `DashPathEffect` 组合，意思是组成虚线路径的每一条虚线可以是一个 `Path` 对象。下面看一下 `PathDashPathEffect` 类构造方法的定义就会完全清楚这一点。

```
| public PathDashPathEffect(Path shape, float advance, float phase, Style style);
```

构造方法的第 1 个参数的类型是 `Path` 对象，说明绘制路径时需要指定一个 `Path` 对象，而这个 `Path` 对象相当于 `DashPathEffect` 对象绘制路径时的虚线线段。

这些参数的含义如下。

- **shape:** 用于绘制虚线图形的 Path 对象。
- **advance:** 两个虚线图形之间的距离。
- **phase:** 绘制路径的偏移量。如果该参数值不断地增大或减小，会呈现路径向前或向后移动的效果。
- **style:** 表示如何在路径的不同位置放置 shape 所绘制的图形。

在 PathDashPathEffect 类中使用了一个 makeCirclePath 方法返回 Path 对象，makeCirclePath 方法通过 Path.addCircle 方法绘制了一个圆形路径。该方法的代码如下：

```
private Path makeCirclePath()
{
    Path p = new Path();
    // 组成路径的图形元素是一个实心圆，如图 15.10 所示的第 4 个路径的效果
    p.addCircle(0, 0, 5, Direction.CCW);
    return p;
}
```

#### 4. ComposePathEffect 类

该类可以将两种特效组合在一起，例如，如图 15.10 所示的第 5 个路径将路径 2 和路径 3 的特效组合在一起。路径 3 是虚线路径，但这个路径的拐角并不圆滑，因此，使用路径 2 的特效（CornerPathEffect 对象）将路径 3 的拐角变得圆滑。路径 6 也是一样，将路径 2 和路径 4 的特效组合，以使特效 6 的拐角变得圆滑。ComposePathEffect 类的构造方法的第一个参数必须是形状特效（DashPathEffect 对象、PathDashPathEffect 对象），第二个参数必须是外观特效（CornerPathEffect 对象）。也就是说，这两个参数不能颠倒。例如，不能把路径 5 中 ComposePathEffect(e[2], e[1]) 的 e[2] 和 e[1] 颠倒过来，否则无法生成新的特效。

下面来看一下负责显示路径特效的 MyView 类的代码。MyView 类中部分代码在前面已经给出，关于这一部分代码读者可参阅前面的内容。

```
private class MyView extends View
{
    private Paint paint;
    private Path path;
    private PathEffect[] effects;
    private int[] colors;
    private float phase;
    // 创建特效对象
    private void makeEffects(PathEffect[] e, float phase){ ... ... }
    public MyView(Context context)
    {
        super(context);
        paint = new Paint();
        paint.setStyle(Paint.Style.STROKE);
        // 设置路径线段的宽度
        paint.setStrokeWidth(5);
        // 创建路径对象（Path 对象）
        path = makeFollowPath();
        effects = new PathEffect[6];
    }
}
```

```

    // 设置 6 条路径的颜色
    colors = new int[]
    { Color.BLACK, Color.RED, Color.BLUE, Color.GREEN, Color.MAGENTA, Color.BLACK };
}
@Override
protected void onDraw(Canvas canvas)
{
    canvas.drawColor(Color.WHITE);
    RectF bounds = new RectF();
    canvas.translate(10 - bounds.left, 10 - bounds.top);
    // 生成路径特效
    makeEffects(effects, phase);
    // 偏移量不断增大，以产生路径不断向前移动的效果
    phase += 1;
    invalidate();
    // 开始绘制 6 条路径
    for (int i = 0; i < effects.length; i++)
    {
        // 设置当前路径的特效
        paint.setPathEffect(effects[i]);
        paint.setColor(colors[i]);
        // 绘制路径
        canvas.drawPath(path, paint);
        canvas.translate(0, 70);
    }
}
// 创建 Path 对象
private Path makeFollowPath() { ... ... }
// 创建绘制路径的 Path 对象（一个实心圆）
private Path makeCirclePath() { ... ... }

```

### 15.3.5 Shader 的渲染效果

工程目录: src\ch15\ch15\_shader

Android SDK 提供了一些渲染图像的类, 例如, BitmapShader、LinearGradient、ComposeShader、RadialGradient 和 SweepGradient, 这些类都继承自 Shader, 用于对图像、颜色进行渲染。这 5 个渲染类的功能如下。

- **BitmapShader:** 将图像按椭圆或弧的形状绘制。
- **LinearGradient:** 线性渐变。
- **RadialGradient:** 径向渐变。
- **SweepGradient:** 角度渐变 (光照效果)。
- **ComposeShader:** 允许将两种渲染效果组合到一起, 因此, 这种渲染可以称为组合渲染或混合渲染。

下面来看一下这 5 个渲染类的构造方法及其参数的含义。

#### BitmapShader 类的构造方法

```
| public BitmapShader(Bitmap bitmap, TileMode tileX, TileMode tileY)
```

参数的含义。

- bitmap: 原图像的 Bitmap 对象。BitmapShader 对象将从 Bitmap 中图像的左上角开始截取椭圆或弧形的图像。

• tileX: 水平方法绘制图像的模式。如图像的宽度是 100，我们要在 200 的宽度绘制，那么可以将该参数值设为 TileMode.REPEAT，表示重复绘制图像，效果类似 Windows 桌面设置背景图时的平铺效果。

• tileY: 垂直方向绘制图像的模式。如果绘制的高度大于图像本身的高度，该参数值就会起作用。例如，该参数值为 TileMode.MIRROR，表示图像在垂直方向以镜像形式绘制。这种效果有些类似岸边的风景在水中的投影。

### LinearGradient 类的构造方法

```
public LinearGradient(float x0, float y0, float x1, float y1, int colors[], float positions[], TileMode tile)
```

参数的含义。

- x0、y0、x1、y1: 这 4 个参数表示一条线段的两个端点的坐标。渐变颜色会延着这条线段绘制。
- colors: 渐变颜色值，可设置多个颜色值。LinearGradient 对象会依次取 colors 数组中的颜色值。
- positions: 表示每一种颜色处于渐变的相对位置。如果该参数值为 null，表示均匀分布每一种颜色。
- tile: 平铺的方式。

### RadialGradient 类的构造方法

```
public RadialGradient(float x, float y, float radius, int colors[], float positions[], TileMode tile)
```

参数的含义。

- x 和 y: 径向渐变的中心点坐标。
- radius: 径向渐变的半径。
- colors: 径向渐变的颜色。
- positions: 每一种颜色在径向渐变中的相对位置。如果该参数值为 null，则平均分配每一种颜色。
- tile: 平铺的方式。

### SweepGradient 类的构造方法

```
public SweepGradient(float cx, float cy, int colors[], float positions[])
```

参数的含义。

- cx 和 cy: 光照点的坐标。
- colors: 光照渐变的颜色。
- positions: 每一种颜色在光照渐变中的相对位置。如果该参数值为 null，则平均分配每一种颜色。

对于一些 3D 立体效果的渐变可以尝试用角度渐变来渲染一个圆锥形，类似光照的效果。

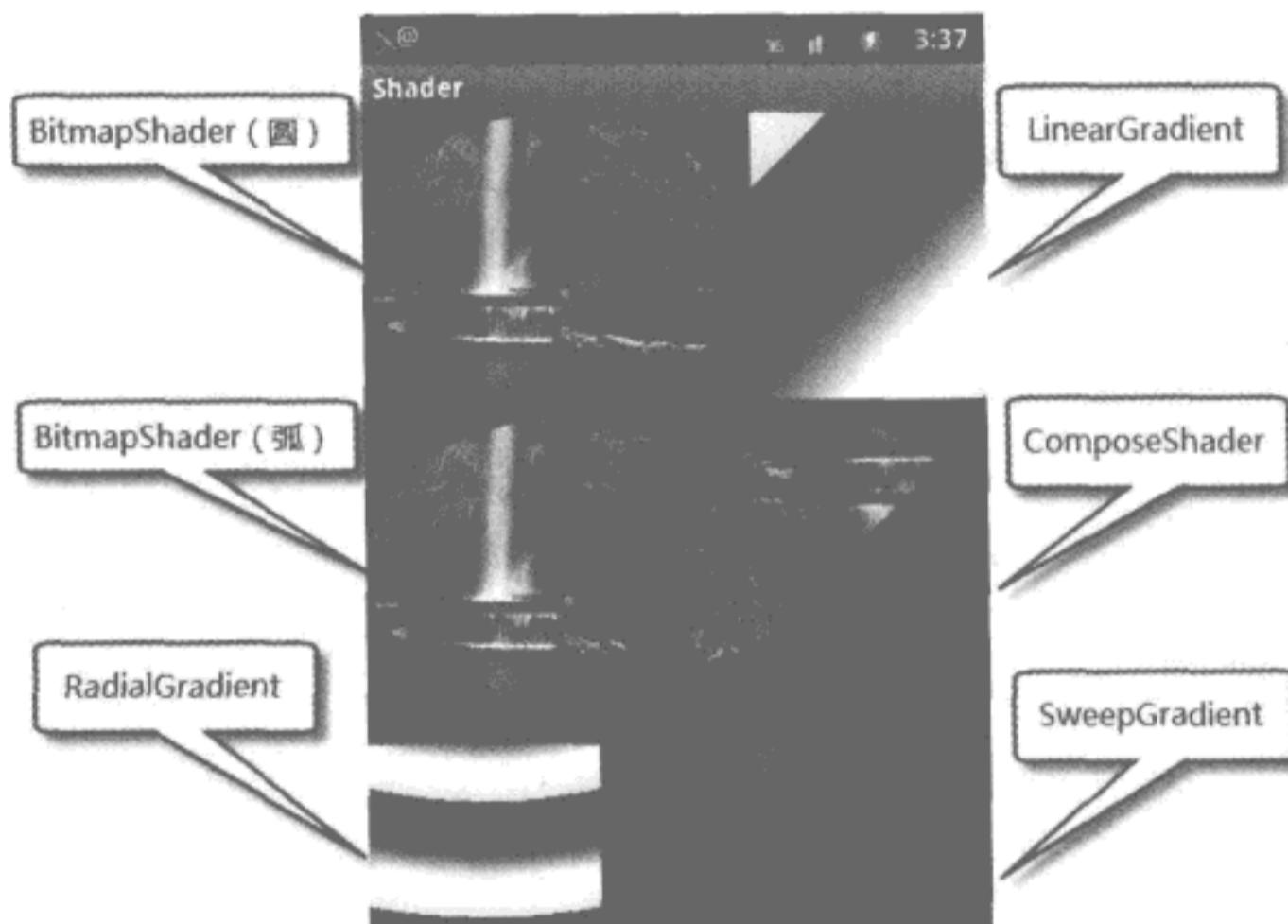
## ComposeShader 类的构造方法

```
public ComposeShader(Shader shaderA, Shader shaderB, PorterDuff.Mode mode)
```

参数的含义。

- **shaderA** 和 **shaderB**: 用于混合渲染的两个 **Shader** 对象。
- **mode**: 混合模式。

上面 5 种渲染效果如图 15.12 所示。其中 **BitmapShader** 分别绘制了椭圆和弧形的图像。



▲ 图 15.12 渲染效果

使用这 5 种渲染类的完整代码如下：

```
package mobile.android.ch15.shader;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.BitmapShader;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.ComposeShader;
import android.graphics.LinearGradient;
import android.graphics.Paint;
import android.graphics.PorterDuff;
import android.graphics.RadialGradient;
import android.graphics.Shader;
import android.graphics.Shader.TileMode;
import android.graphics.SweepGradient;
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.ArcShape;
import android.graphics.drawable.shapes.OvalShape;
import android.view.View;
```

```
public class ShaderView extends View
{
    private Bitmap bitmap;
    private Shader bitmapShader;
    private Shader linearGradient;
    private Shader composeShader;
    private Shader radialGradient;
    private Shader sweepGradient;

    public ShaderView(Context context)
    {
        super(context);
        // 装载 res\drawable 目录中的图像资源
        bitmap = BitmapFactory.decodeResource(context.getResources(), .drawable.image1);
        // 截取图像中的一部分
        bitmap = Bitmap.createBitmap(bitmap, 200, 260, 200, 150);
        // 创建 BitmapShader 对象
        bitmapShader = new BitmapShader(bitmap, TileMode.REPEAT, TileMode.MIRROR);

        // 创建 LinearGradient 对象
        linearGradient = new LinearGradient(0, 0, 120, 120, new int[]
        { Color.RED, Color.BLUE, Color.WHITE, Color.YELLOW }, null, TileMode.REPEAT);

        // 创建 ComposeShader 对象。该对象将 BitmapShader 和 LinearGradient 两种效果组合在了一起
        composeShader = new ComposeShader(bitmapShader, linearGradient, PorterDuff.Mode.
        DARKEN);

        // 创建 RadialGradient 对象
        radialGradient = new RadialGradient(60, 120, 60, new int[]
        { Color.RED, Color.BLUE, Color.WHITE, Color.YELLOW }, null, TileMode.REPEAT);

        // 创建 SweepGradient 对象
        sweepGradient = new SweepGradient(300, 300, new int[]
        { Color.BLUE, Color.RED, Color.WHITE, Color.YELLOW }, null);
    }

    @Override
    protected void onDraw(Canvas canvas)
    {
        ShapeDrawable shapeDrawable = new ShapeDrawable(new OvalShape());
        Paint paint = shapeDrawable.getPaint();
        paint.setShader(bitmapShader);
        shapeDrawable.setBounds(0, 0, bitmap.getWidth(), bitmap.getHeight());
        // 绘制椭圆图像
        shapeDrawable.draw(canvas);

        shapeDrawable.setShape(new ArcShape(60, 240));
        shapeDrawable.setBounds(0, 160, 200, 310);
        // 绘制弧形图像
        shapeDrawable.draw(canvas);

        paint.setShader(linearGradient);
        // 绘制线性渐变颜色
        canvas.drawRect(200, 0, 320, 150, paint);
    }
}
```

```

    paint.setShader(composeShader);
    // 将混合渲染效果绘制在圆中
    canvas.drawCircle(240, 240, 80, paint);

    paint.setShader(radialGradient);
    // 绘制径向渐变颜色
    canvas.drawRect(0, 330, 120, 460, paint);

    paint.setShader(sweepGradient);
    // 绘制角度渐变颜色
    canvas.drawRect(200, 330, 320, 460, paint);
    super.onDraw(canvas);
}
}

```

## 15.4 帧 (Frame) 动画

如果读者使用过 Flash，一定对帧动画非常熟悉。帧动画实际上就是由若干图像组成的动画，这些图像会以一定的时间间隔进行切换。电影就是非常典型的帧动画。一般电影的播放频率是每秒 25 帧。也就是说，电影在 1 秒钟之内会以相同的时间间隔连续播放 25 幅电影静态画面。由于人的视觉暂留，在这样的播放频率下，电影看起来才是连续的。本节将介绍如何使用 AnimationDrawable 和静态图像来制作帧动画。

### 15.4.1 AnimationDrawable 与帧动画

工程目录: src\ch15\ch15\_frame

Android 中的帧动画需要在一个动画文件中指定动画的静态图像和每一个静态图像的停留时间（单位：毫秒）。一般可以将所有图像的停留时间设为同一个值。动画文件采用了 XML 格式，该文件需要放在 res\anim 目录中。先来建立一个简单的动画文件，首先在 res\anim 目录中建立一个 test.xml 文件，然后输入如下内容：

```

<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:
    oneshot="false">
    <item android:drawable="@drawable/image1" android:duration="200" />
    <item android:drawable="@drawable/image2" android:duration="100" />
    <item android:drawable="@drawable/image3" android:duration="200" />
    <item android:drawable="@drawable/image4" android:duration="150" />
    <item android:drawable="@drawable/image5" android:duration="100" />
    <item android:drawable="@drawable/image6" android:duration="200" />
</animation-list>

```

从 anim.xml 文件的内容可以看出，一个标准的动画文件由一个`<animation-list>`标签和若干`<item>`标签组成。其中`<animation-list>`标签的一个关键属性是`android:oneshot`，如果该属性值为 true，表示帧动画只播放一遍，也就是在显示完最后一帧图像后，动画就会停止。如果该属性值为 false，表示帧动画会循环播放。`android:oneshot` 是可选属性，默认值是 false。

<item>标签的 android:drawable 属性指定了动画中的静态图像资源 ID，帧动画的播放顺序就是<item>标签的定义顺序。 android:duration 属性指定了每个图像的停留时间（也可以理解为图像之间切换的间隔时间），在 test.xml 文件中为每个图像设置了不同的停留时间。 android:drawable 和 android:duration 都是必选属性，不能省略。

编写完动画文件后，就需要装载动画文件，并创建 AnimationDrawable 对象。AnimationDrawable 是 Drawable 的子类，并在 Drawable 的基础上提供了控制动画的功能。读者可以使用如下代码根据 test.xml 文件创建 AnimationDrawable 对象：

```
AnimationDrawable animationDrawable =  
    (AnimationDrawable) getResources().getDrawable(R.anim.test);
```

在创建完 AnimationDrawable 对象后，可以使用下面的代码将 AnimationDrawable 对象作为 ImageView 控件的背景。

```
ImageView imageview = (ImageView) findViewById(R.id.imageview);  
imageview.setBackgroundDrawable(animationDrawable);
```

除了可以使用 getDrawable 方法装载 test.xml 文件外，还可以使用 setBackgroundResource 方法装载 test.xml 文件，并通过 getBackground 方法获得 AnimationDrawable 对象，代码如下：

```
ImageView imageview = (ImageView) findViewById(R.id.imageview);  
imageview.setBackgroundResource(R.anim.test);  
Object backgroundObject = ivAnimView.getBackground();  
animationDrawable = (AnimationDrawable) backgroundObject;
```

有了 AnimationDrawable 对象，就可以通过 AnimationDrawable 类的方法来控制帧动画。 Animation- Drawable 类中与帧动画相关的方法如下。

- start: 开始播放帧动画。
- stop: 停止播放帧动画。

• setOneShot: 设置是否只播放一遍帧动画。该方法的参数值与动画文件中的<animation-list>标签的 android:oneshot 属性值的含义相同。参数值为 true 表示只播放一遍帧动画，参数值为 false 表示循环播放帧动画。默认值为 false。

• addFrame: 向 AnimationDrawable 对象中添加新的帧。该方法有两个参数，第 1 个参数是一个 Drawable 对象，表示添加的帧，该参数值可以是静态图像，也可以是另一个动画。第 2 个参数表示新添加帧的停留时间。如果新添加的帧是动画，那么这个停留时间就是新添加的动画可以播放的时间。如果在播放时间内，新添加的动画仍然未播放完，仍然会切换到下一帧静态图像或动画。

• isOneShot: 判断帧动画是否只播放一遍。该方法返回通过 setOneShot 方法或 android:oneshot 属性设置的值。

• isRunning: 判断帧动画是否正在播放。如果返回 true，表示帧动画正在播放。返回 false 表示帧动画已停止播放。

- getNumberOfFrames: 返回动画的帧数，也就是<animation-list>标签中的<item>标签数。
- getFrame: 根据帧索引获得指定帧的 Drawable 对象。帧从 0 开始。
- getDuration: 获得指定帧的停留时间。

运行本例，单击“播放动画”按钮和“停止动画”按钮来控制动画的播放和停止，效果如图 15.13 所示。



▲ 图 15.13 播放动画的效果

### 扩展学习：动态添加动画或图像以及动画的透明度

在播放事先定义的动画的同时，还可以使用 `AnimationDrawable.addFrame` 方法为动画添加动画或图像。`addFrame` 方法的定义如下：

```
| public void addFrame(Drawable frame, int duration)
```

其中 `frame` 表示要添加的 `Drawable` 对象。该对象可以是静态图像，也可以是 `res\anim` 目录中定义的动画。`Duration` 表示新添加的动画或静态图的停留时间。下面的代码向正在播放的动画结尾添加了一个新的动画。

```
animationDrawable1 = (AnimationDrawable) getResources().getDrawable(R.anim.frame_animation1);
// 添加动画，动画停留（播放）时间是 2 秒
animationDrawable.addFrame(animationDrawable1, 2000);
```

在添加新的动画后，`AnimationDrawable` 对象会接着新添加的动画继续播放。再次循环播放时，仍然会把原来的动画和新添加的动画在一起播放。

如果想实现动画的半透明效果，可以使用 `AnimationDrawable.setAlpha` 方法。该方法只有一个 `int` 类型的值，该值的范围是 0~255。如果参数值是 0，表示图像完全透明；如果参数值是 255，表示图像完全不透明。

#### 15.4.2 播放 Gif 动画

工程目录：src\ch15\ch15\_play\_gif

Android SDK 并没有提供播放 Gif 动画的 API（只能显示静态的 Gif 图像），而 Gif 动画是最常用，也最容易制作的动画格式。如果要是能直接将 Gif 动画应用到游戏中，那真是再好不过了。

虽然无法使用 Android SDK API 播放 Gif 动画，但可以利用其他的方法弥补这一不足，最直接的方法是将 Gif 动画打散（变成若干个静态的 Gif 图像文件），然后用 15.4.1 小节介绍的帧动画来播放这些单独的静态图像。这种方法虽然可行，但比较麻烦。在本小节将介绍另外一种直接播放 Gif 动画的方法。

要想播放 Gif 动画，首先要分别获得 Gif 动画中的单个图像（只在内存中获得，并不需要保存成文件的形式）。为此，本例提供了一个 jar 包（gif\_lib.jar）用于对 Gif 动画进行解码，并逐一获得 Gif 动画中的静态图像。在 gif\_lib.jar 包中有一个 GifFrames 类，通过 GifFrames.getImage 方法可以获得 Gif 动画当前的静态图像。通过 GifFrames.nextFrame 方法可以切换到下一个静态图像，并通过 getImage 方法获得这个静态图像。

在编写本例的代码之前，先准备一个 Gif 动画文件（bird.gif），将该文件放在 assets 目录中（要播放的 Gif 动画文件必须放在 assets 或 res\raw 目录中，不能放在 res\drawable 资源目录中），然后使用 GifFrames.createGifFrames 方法根据 bird.gif 文件创建 GifFrames 对象，并在线程中逐一获得 Gif 动画中的图像文件，并将其显示在 ImageView 控件中。完整的代码如下：

```
package mobile.android.ch15.play.gif;

import java.io.InputStream;
import android.app.Activity;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.ImageView;

public class Main extends Activity
{
    private ImageView imageView;
    private GifFrames gifFrames;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        imageView = (ImageView) findViewById(R.id.imageview);
        try
        {
            // 获得 Gif 动画的 InputStream 对象
            InputStream is = getResources().getAssets().open("bird.gif");
            // 创建 GifFrames 对象
            gifFrames = GifFrames.createGifFrames(is);
        }
        catch (Exception e)
        {
```

```

        }
    }
    public void onClick_PlayGif(View view)
    {
        // 开始播放动画的线程
        new GifAnimThread().start();
    }
    private Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            super.handleMessage(msg);
            Bitmap bitmap = (Bitmap) msg.obj;
            // 将当前帧的图像显示在 ImageView 控件中
            imageView.setImageBitmap(bitmap);
        }
    };
    // 播放 Gif 动画的线程
    class GifAnimThread extends Thread
    {
        @Override
        public void run()
        {
            super.run();
            try
            {
                while (true)
                {
                    // 获得当前帧的图像
                    Bitmap bitmap = gifFrames.getImage();
                    // 切换到下一帧
                    gifFrames.nextFrame();
                    if (bitmap != null)
                    {
                        Message msg = new Message();
                        msg.obj = bitmap;
                        // 在线程中必须使用 Handler 来更新控件中的内容
                        handler.sendMessage(msg);
                    }
                    // 每 150 毫秒播放一帧动画
                    sleep(150);
                }
            }
            catch (Exception e)
            {
            }
        }
    }
}

```

运行程序，单击“播放”按钮，会播放 Gif 动画，效果如图 15.14 所示。



▲ 图 15.14 播放 Gif 动画

## 15.5 补间 (Tween) 动画

如果动画中的图像变化比较有规律，可以采用自动生成中间图像的方式来生成动画，例如图像的移动、旋转、缩放等。当然，还有更复杂的情况，例如由正方形变成圆形、圆形变成椭圆形，这些变化过程中的图像都可以根据一定的数学算法自动生成。我们只需要指定动画的第一帧和最后一帧的图像即可，这种自动生成中间图像的动画被称为补间 (Tween) 动画。

补间动画的优点是节省硬盘空间，因为这种动画只需要提供两帧图像（第一帧和最后一帧），其他的图像都由系统自动生成。当然，这种动画也有一定的缺点，就是在动画很复杂时无法自动生成中间图像，例如由电影画面组成的动画，由于每幅画面过于复杂，系统无法预料下一幅画面是什么样子。因此，这种复杂的动画只能使用帧动画来完成。本节将介绍 Android SDK 提供的 4 种补间动画效果：移动、缩放、旋转和透明度。Android SDK 并未提供更复杂的补间动画，如果要实现更复杂的补间动画，需要开发人员自己编码来完成。

### 15.5.1 移动补间动画

工程目录：src\ch15\ch15\_translate\_tween

移动是最常见的动画效果，可以通过配置动画文件或编写 Java 代码来实现补间动画的移动效果。补间动画文件需要放在 res\anim 目录中，在动画文件中通过<translate>标签设置移动效果。首先在 res\anim 目录下建一个动画文件 translate\_tween.xml，该文件的内容如下：

```
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:fromXDelta="0" android:toXDelta="320" android:fromYDelta="0"
    android:toYDelta="0" android:duration="2000" />
```

从上面的配置代码可以看出，`<translate>`标签中设置了 6 个属性，这 6 个属性的含义如下。

- `android:interpolator`: 表示动画渲染器。通过 `android:interpolator` 属性可以设置 3 个动画渲染器，`accelerate_interpolator`(动画加速器)、`decelerate_interpolator`(动画减速器)和 `accelerate_decelerate_interpolator` (动画加速减速器)。动画加速器使动画在开始时速度最慢，然后逐渐加速。动画减速器使动画在开始时速度最快，然后逐渐减速。动画加速减速器使动画在开始和结束时速度最慢，但在前半部分时开始加速，在后半部分时开始减速。

- `android:fromXDelta`: 动画起始位置的横坐标。
- `android:toXDelta`: 动画结束位置的横坐标。
- `android:fromYDelta`: 动画起始位置的纵坐标。
- `android:toYDelta`: 动画结束位置的纵坐标。
- `android:duration`: 动画的持续时间，单位是毫秒。也就是说，动画要在 `android:duration` 属性指定的时间内从起始点移动到结束点。

装载补间动画文件需要使用 `android.view.animation.AnimationUtils.loadAnimation` 方法，该方法的定义如下：

```
| public static Animation loadAnimation(Context context, int id);
```

其中 `id` 表示动画文件的资源 ID。装载 `translate_tween.xml` 文件的代码如下：

```
| Animation animation = AnimationUtils.loadAnimation(this, R.anim.translate_tween);
```

在布局文件中放一个 `EditText` 控件，将补间动画应用到 `EditText` 控件上的方式有如下两种。

(1) 使用 `EditText.startAnimation` 方法，代码如下：

```
| editText.startAnimation(animation);
```

(2) 使用 `Animation.start` 方法，代码如下：

```
| // 绑定补间动画
| editText.setAnimation(animation);
| // 开始动画
| animation.start();
```

使用上面两种方式开始补间动画都只执行一次。如果想循环显示动画，需要使用如下代码将动画设置成循环状态：

```
| animation.setRepeatCount(Animation.INFINITE);
```

上面的代码在开始动画之前和之后执行都没有问题。

如果想通过 Java 代码实现移动补间动画，可以创建 `android.view.animation.TranslateAnimation` 对象。`TranslateAnimation` 类构造方法的定义如下：

```
| public TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float
| toYDelta);
```

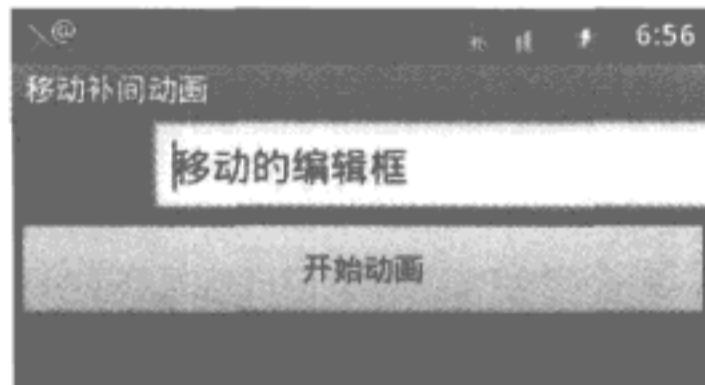
通过 `TranslateAnimation` 类的构造方法可以设置动画起始位置和结束位置的坐标。在创建

TranslateAnimation 对象后，可以通过 TranslateAnimation 类的如下方法设置移动补间动画的其他属性。

- **setInterpolator:** 设置动画渲染器。该方法的参数类型是 Interpolator，在 Android SDK 中提供了一些动画渲染器，例如 LinearInterpolator、AccelerateInterpolator 等，其中部分动画渲染器可以在动画文件的<translate>标签的 android:interpolator 属性中设置，而有的动画渲染器需要使用 Java 代码设置。
- **setDuration:** 设置动画的持续时间。该方法相当于设置了<translate>标签的 android:duration 属性。

补间动画有 3 个状态，分别是动画开始、动画结束、动画循环。要想监听这 3 个状态，需要实现 android.view.animation.Animation.AnimationListener 接口。该接口定义了 3 个方法，分别是 onAnimationStart、onAnimationEnd 和 onAnimationRepeat，这 3 个方法分别在动画开始、动画结束和动画循环时调用。

运行本例，单击“开始动画”按钮，按钮上方的编辑框就会从左向右水平加速移动，并不断循环这个过程。效果如图 15.15 所示。



▲ 图 15.15 移动补间动画（水平移动的 EditText 控件）

### 15.5.2 缩放补间动画

工程目录: src\ch15\ch15\_scale\_tween

通过<scale>标签可以定义缩放补间动画。下面的代码定义了一个标准的缩放补间动画。

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXScale="0.2" android:toXScale="1.0" android:fromYScale="0.2"
    android:toYScale="1.0" android:pivotX="50%" android:pivotY="50%"
    android:duration="2000" />
```

<scale>标签和<translate>标签中有些属性是相同的，而有些属性是<scale>标签特有的，这些属性的含义如下。

- **android:fromXScale:** 表示沿 x 轴缩放的起始比例。
- **android:toXScale:** 表示沿 x 轴缩放的结束比例。
- **android:fromYScale:** 表示沿 y 轴缩放的起始比例。
- **android:toYScale:** 表示沿 y 轴缩放的结束比例。
- **android:pivotX:** 表示沿 x 轴方向缩放的支点位置。如果该属性值为 50%，则支点在沿 x 轴的中心位置。
- **android:pivotY:** 表示沿 y 轴方向缩放的支点位置。如果该属性值为 50%，则支点在沿 y 轴

的中心位置。

其中前 4 个属性的取值规则如下。

- 0.0：表示收缩到没有。
- 1.0：表示正常不收缩。
- 大于 1.0：表示将控件放大到相应的比例，例如值为 1.5，表示放大到原控件的 1.5 倍。
- 小于 1.0：表示将控件缩小到相应的比例，例如值为 0.5，表示缩小到原控件的 50%。

如果想通过 Java 代码实现缩放补间动画，可以创建 `android.view.animation.ScaleAnimation` 对象。`ScaleAnimation` 类构造方法的定义如下：

```
public ScaleAnimation(float fromX, float toX, float fromY, float toY, float pivotX, float pivotY)
```

通过 `ScaleAnimation` 类的构造方法可以设置上述 6 个属性值。设置其他属性的方法与移动补间动画相同。

下面编写一个实际的例子，在这个例子中有两个动画文件（`to_small.xml` 和 `to_large.xml`）。其中 `to_small.xml` 文件使控件从原始大小加速变小，而 `to_large.xml` 文件使控件从缩小状态减速变大（恢复原始大小）。这两个动画文件的代码如下：

#### **to\_small.xml**

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:fromXScale="1.0" android:toXScale="0.2" android:fromYScale="1.0"
    android:toYScale="0.2" android:pivotX="50%" android:pivotY="50%"
    android:duration="500" />
```

#### **to\_large.xml**

```
<scale xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXScale="0.2" android:toXScale="1.0" android:fromYScale="0.2"
    android:toYScale="1.0" android:pivotX="50%" android:pivotY="50%"
    android:duration="500" />
```

现在准备一个红心的图像。本例的实现效果是当红心变小后，再开始变大，并不断重复。完整的实现代码如下：

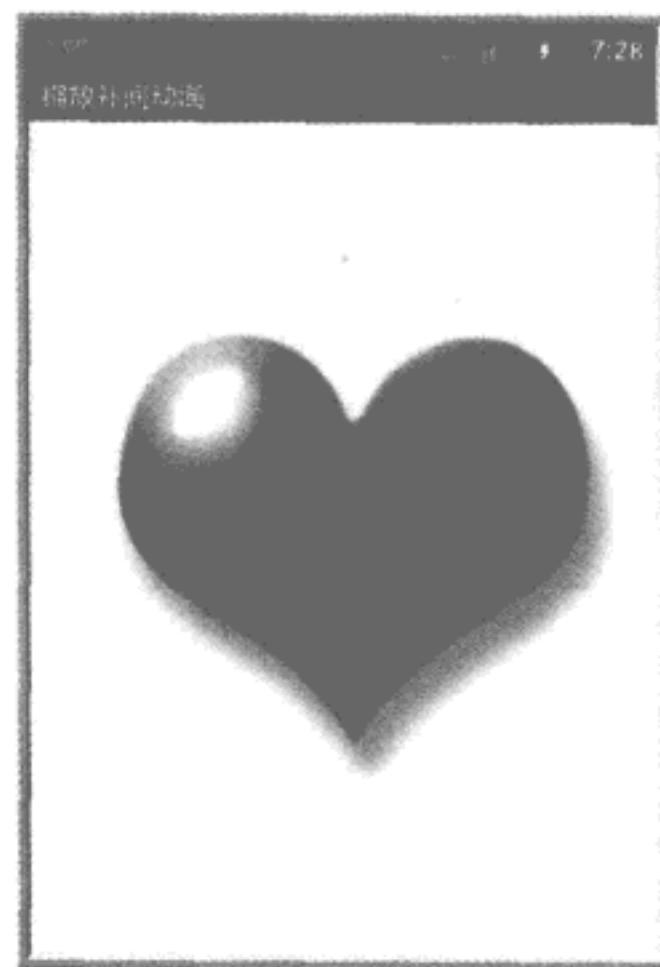
```
package mobile.android.ch15.scale;

import android.app.Activity;
import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.Animation.AnimationListener;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

public class Main extends Activity implements AnimationListener
{
    private Animation toLargeAnimation;
    private Animation toSmallAnimation;
```

```
private ImageView imageView;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    imageView = (ImageView) findViewById(R.id.imageview);
    toLargeAnimation = AnimationUtils.loadAnimation(this, R.anim.to_large);
    toSmallAnimation = AnimationUtils.loadAnimation(this, R.anim.to_small);
    toLargeAnimation.setAnimationListener(this);
    toSmallAnimation.setAnimationListener(this);
    imageView.startAnimation(toSmallAnimation);
}
@Override
public void onAnimationStart(Animation animation)
{
}
@Override
public void onAnimationEnd(Animation animation)
{
    // 不断切换播放的动画
    if (animation.hashCode() == toLargeAnimation.hashCode())
        imageView.startAnimation(toSmallAnimation);
    else
        imageView.startAnimation(toLargeAnimation);
}
@Override
public void onAnimationRepeat(Animation animation)
{
}
}
```

运行本例，会看到屏幕上有一颗跳动的红心，效果如图 15.16 所示。



▲ 图 15.16 缩放补间动画（跳动的心）

### 15.5.3 旋转补间动画

工程目录: src\ch15\ch15\_rotate\_tween

通过<rotate>标签可以定义旋转补间动画。下面的代码定义了一个标准的旋转补间动画。

```
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator" android:from
    Degrees="0"
    android:toDegrees="360" android:pivotX="50%" android:pivotY="50%"
    android:duration="10000" android:repeatMode="restart" android:repeatCount
    ="infinite"/>
```

其中<rotate>标签有两个特殊的属性。它们的含义如下。

- android:fromDegrees: 表示旋转的起始角度。
- android:toDegrees: 表示旋转的结束角度。

在<rotate>标签中还使用如下两个属性设置旋转的次数和模式。

- android:repeatCount: 设置旋转的次数。该属性需要设置一个整数值，如果该值为 0，表示不重复播放动画。也就是说，对于上面的旋转补间动画，只从 0° 旋转到 360°，动画就会停止。如果属性值大于 0，动画会再播放该属性指定的次数。例如，如果 android:repeatCount 属性值为 1，动画除了正常播放一次外，还会再播放一次。也就是说，前面的旋转补间动画会顺时针旋转两周。如果想让补间动画永不停止，可以将 android:repeatCount 属性值设为 infinite 或 -1。该属性的默认值是 0。

- android:repeatMode: 设置重复的模式，默认值是 restart。该属性只有当 android:repeatCount 属性值大于 0 或是 infinite 时才起作用。android:repeatMode 属性值除了可以是 restart 外，还可以设为 reverse，表示偶数次显示动画时会做与动画文件定义的方向相反的动作。例如，上面定义的旋转补间动画会在第 1、3、5…… $2n-1$  圈顺时针旋转，而在 2、4、6…… $2n$  圈逆时针旋转。如果想使用 Java 代码来设置该属性，可以使用 Animation.setRepeatMode 方法，该方法只接收一个 int 类型的参数。可取的值是 Animation.RESTART 和 Animation.REVERSE。

如果想通过 Java 代码实现旋转补间动画，可以创建 android.view.animation.RotateAnimation 对象。RotateAnimation 类构造方法的定义如下：

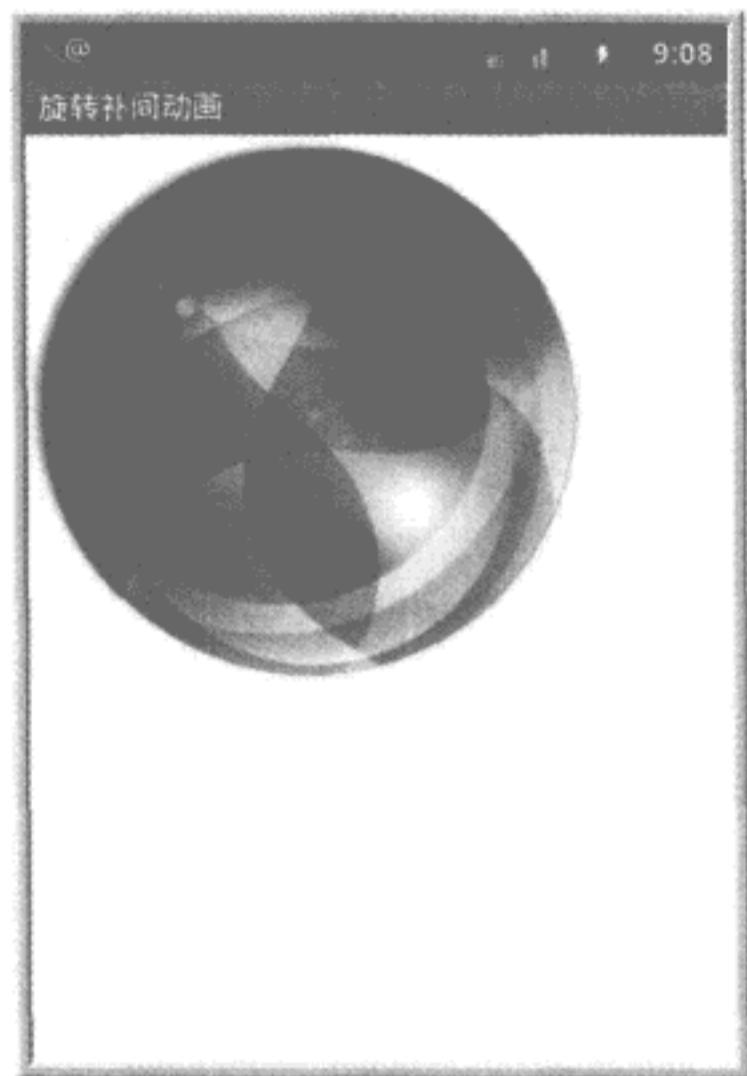
```
public RotateAnimation(float fromDegrees, float toDegrees, float pivotX, float pivotY);
```

通过 RotateAnimation 类的构造方法可以设置旋转开始角度 (fromDegrees)、旋转结束角度 (toDegrees)、旋转支点横坐标 (pivotX) 和旋转支点纵坐标 (pivotY)。

对一个 ImageView 控件应用旋转补间动画的代码如下：

```
ImageView imageView = (ImageView) findViewById(R.id.imageview);
Animation animation = AnimationUtils.loadAnimation(this, R.anim.rotate_tween);
imageView.startAnimation(animation);
```

运行本例，会看到 ImageView 控件中的图像顺时针旋转了起来，如图 15.17 所示。



▲ 图 15.17 旋转补间动画（自转的球）

#### 15.5.4 透明度补间动画

工程目录: src\ch15\ch15\_alpha\_tween

通过<alpha>标签可以定义透明度补间动画。下面的代码定义了一个标准的透明度补间动画。

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"  
    android:interpolator="@android:anim/accelerate_interpolator"  
    android:fromAlpha="1.0" android:toAlpha="0.1" android:duration="5000" />
```

其中 android:fromAlpha 和 android:toAlpha 属性分别表示起始透明度和结束透明度，这两个属性的值都在 0.0~1.0。属性值为 0.0 表示完全透明，属性值为 1.0 表示完全不透明。

如果想通过 Java 代码实现透明度补间动画，可以创建 android.view.animation.AlphaAnimation 对象。AlphaAnimation 类构造方法的定义如下：

```
public AlphaAnimation(float fromAlpha, float toAlpha);
```

通过 AlphaAnimation 类的构造方法可以设置起始透明度(fromAlpha)和结束透明度(toAlpha)。将透明度补间动画应用到 ImageView 控件的代码如下：

```
ImageView imageView = (ImageView) findViewById(R.id.imageview);  
Animation animation = AnimationUtils.loadAnimation(this,R.anim.alpha_tween);  
imageView.startAnimation(animation);
```

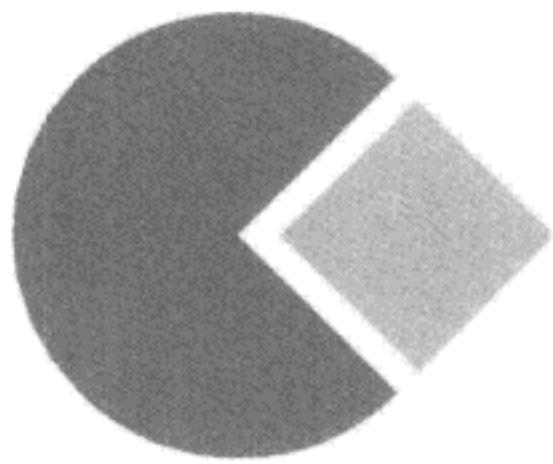
运行程序，单击“开始”按钮，按钮下方的球会逐渐变暗，效果如图 15.18 所示。



▲ 图 15.18 透明度补间动画（逐渐变暗的球）

## 15.6 小结

本章主要介绍了 Android SDK 支持的 2D 绘图和动画技术，这些技术都可以应用到游戏中。如果只是绘制图形，可以使用 View 或 SurfaceView，虽然它们都可以使用 Canvas 绘制图形，功能也完全一样，但 SurfaceView 支持双缓冲技术，如果绘制复杂的图形，使用 SurfaceView 会更快速，效果会更好。Android SDK 还支持帧动画和 4 种补间动画，它们都需要使用 res\anim 目录中的动画文件来播放动画。



## 第 16 章 有趣的 Android 应用

Android SDK 提供了很多有趣的 API，例如，可以通过变换手机的位置和方向来控制游戏中的赛车；当外部光线很强或很弱时适当调整手机的屏幕亮度；通过 GPS 在地图上定位；像动画一样的桌面壁纸；通过语音快速录入文字。这些功能都可以通过传感器、GPS 等 API 实现。本章将详细介绍这些 API 的使用方法。

### 16.1 传感器

传感器目前已经成为大多数智能手机的标配，例如，Android、iPhone、Window Phone 7 都加入了各式各样的传感器。比较常用的传感器包括方向传感器、光线传感器、压力传感器、磁场传感器等。本节将详细介绍这些传感器的使用方法。

#### 16.1.1 如何使用传感器

工程目录：src\ch16\ch16\_sensor

我们可以通过如下三步使用传感器。

- (1) 编写一个截获传感器事件的类。该类必须实现 android.hardware.SensorEventListener 接口。
- (2) 获得传感器管理对象（SensorManager 对象）。
- (3) 使用 SensorManager.registerListener 方法注册指定的传感器。

通过上面三步已经搭建了传感器应用程序的框架，而具体的工作需要在 SensorEventListener 接口的 onSensorChanged 和 onAccuracyChanged 方法中完成。SensorEventListener 接口的定义如下：

```
package android.hardware;
public interface SensorEventListener
{
    // 传感器数据变化时调用
    public void onSensorChanged(SensorEvent event);
    // 传感器精确度变化时调用
    public void onAccuracyChanged(Sensor sensor, int accuracy);
}
```

SensorManager 对象通过 getSystemService 方法获得，代码如下：

```
SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

通常手机中包含了若干个传感器模块（如方向传感器、光线传感器等），因此，注册传感器需要指定传感器的类型，如下面的代码注册了光线传感器。

```
sensorManager.registerListener(this,
    sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT),
    SensorManager.SENSOR_DELAY_FASTEST);
```

`registerListener` 方法有 3 个参数，第 1 个参数是实现 `SensorEventListener` 接口的对象，第 2 个参数用于指定传感器的类型，Android SDK 预先定义了表示各种传感器的常量，这些常量都被放在 `Sensor` 类中，例如，上面代码中的 `Sensor.TYPE_LIGHT`。第 3 个参数表示传感器获得数据的速度。该参数可设置的常量如下。

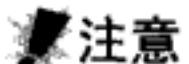
- `SENSOR_DELAY_FASTEST`: 以最快的速度获得传感器数据。
- `SENSOR_DELAY_GAME`: 适合于在游戏中获得传感器数据。
- `SENSOR_DELAY_UI`: 适合于在 UI 控件中获得传感器数据。
- `SENSOR_DELAY_NORMAL`: 以一般的速度获得传感器的数据。

上面 4 种类型获得传感器数据的速度依次递减，从理论上说，获得传感器数据的速度越快，消耗的系统资源越大，因此建议读者根据实际情况选择适当的速度获得传感器的数据。

如果想停止获得传感器数据，可以使用 `unregisterSensor` 方法注销传感器事件对象。`unregisterSensor` 方法的定义如下：

```
public void unregisterListener(SensorEventListener listener)
public void unregisterListener(SensorEventListener listener, Sensor sensor)
```

`unregisterSensor` 方法有两个重载形式，第一个重载形式用于注销所有的传感器对象，第二个重载形式用于注销指定传感器的事件对象，其中 `Sensor` 对象通过 `SensorManager.getDefaultSensor` 方法获得。`getDefaultSensor` 方法只有一个 `int` 类型的参数，表示传感器的类型，如 `Sensor.TYPE_LIGHT` 表示光线传感器。



一个传感器对象可以处理多个传感器，也就是说，一个实现 `SensorEventListener` 接口的类可以接收多个传感器传回的数据。为了区分不同的传感器，需要使用 `Sensor.getType` 方法来获得传感器的类型。`getType` 方法将在本节的例子中详细介绍。

通过 `SensorManager.getSensorList` 方法可以获得指定传感器的信息，也可以获得手机支持的所有传感器的信息，代码如下：

```
// 获得光线传感器
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_LIGHT);
// 获得手机支持的所有传感器
List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
```

下面给出一个完整的例子来演示如何获得传感器传回的数据。本例从如下 4 个传感器获得数据，同时输出了测试手机中支持的所有传感器名称。

- 加速度传感器 (`Sensor.TYPE_ACCELEROMETER`)。



- 磁场传感器 (Sensor.TYPE\_MAGNETIC\_FIELD)。
  - 光线传感器 (Sensor.TYPE\_LIGHT)。
  - 方向传感器 (TYPE\_ORIENTATION)。

本例需要在真机上运行。由于不同的手机可能支持的传感器不同（有的手机并不支持 Android SDK 中定义的所有传感器），因此，如果运行程序后，无法显示某个传感器的数据，说明当前的手机并不支持这个传感器。笔者已使用 Google Nexus S 测试了本例。如果读者使用的也是 Google Nexus S，则会输出与如图 16.1 所示类似的信息。



▲ 图 16.1 获得传感器传回的数据

本例的完整代码如下：

```
package mobile.android.ch16.sensor;

import java.util.List;
import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.widget.TextView;

public class Main extends Activity implements SensorEventListener
{
    private TextView tvAccelerometer;
    private TextView tvMagnetic;
    private TextView tvLight;
```

## Android 开发权威指南

```
private TextView tvOrientation;
private TextView tvSensors;

@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 获得 SensorManager 对象
    SensorManager sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

    // 注册加速度传感器
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_FASTEST);

    // 注册磁场传感器
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
        SensorManager.SENSOR_DELAY_FASTEST);

    // 注册光线传感器
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT),
        SensorManager.SENSOR_DELAY_FASTEST);

    // 注册方向传感器
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
        SensorManager.SENSOR_DELAY_FASTEST);

    tvAccelerometer = (TextView) findViewById(R.id.tvAccelerometer);
    tvMagnetic = (TextView) findViewById(R.id.tvMagnetic);
    tvLight = (TextView) findViewById(R.id.tvLight);
    tvOrientation = (TextView) findViewById(R.id.tvOrientation);
    tvSensors = (TextView) findViewById(R.id.tvSensors);

    // 获得当前手机支持的所有传感器
    List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
    for(Sensor sensor:sensors)
    {
        // 输出当前传感器的名称
        tvSensors.append(sensor.getName() + "\n");
    }
}
@Override
public void onSensorChanged(SensorEvent event)
{
    // 通过 getType 方法获得当前传回数据的传感器类型
    switch (event.sensor.getType())
    {
        case Sensor.TYPE_ACCELEROMETER:           // 处理加速度传感器传回的数据
            String accelerometer = "加速度\n" + "X: " + event.values[0] + "\n"
                + "Y: " + event.values[1] + "\n" + "Z: " + event.values[2] + "\n";
            tvAccelerometer.setText(accelerometer);
    }
}
```

```

        break;
    case Sensor.TYPE_LIGHT:           // 处理光线传感器传回的数据
        tvLight.setText("亮度: " + event.values[0]);
        break;
    case Sensor.TYPE_MAGNETIC_FIELD:   // 处理磁场传感器传回的数据
        String magentic = "磁场\n" + "X: " + event.values[0] + "\n" + "Y: "
            + event.values[1] + "\n" + "Z: " + event.values[2] + "\n";
        tvMagnetic.setText(magnetic);
        break;
    case Sensor.TYPE_ORIENTATION:     // 处理方向传感器传回的数据
        String orientation = "方向\n" + "X: " + event.values[0] + "\n"
            + "Y: " + event.values[1] + "\n" + "Z: " + event.values[2] + "\n";
        tvOrientation.setText(orientation);
        break;
    }
}
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy)
{
}
}

```

上面的代码中使用了 `event.values` 数组中的数据来获得传感器传回的数据，这个 `values` 数组非常重要，它的长度为 3，但不一定每一个数组元素都有意义。对于不同的传感器，每个数组元素的含义不同，在下面的部分将详细介绍不同传感器中 `values` 数组各个元素的含义。

**注意** 虽然使用 `Sensor.TYPE_ALL` 可以获得手机支持的所有传感器信息，但不能使用 `Sensor.TYPE_ALL` 注册所有的传感器，也就是 `getDefaultSensor` 方法的参数值必须是某个传感器的类型常量，而不能是 `Sensor.TYPE_ALL`。

### 16.1.2 加速度传感器（Accelerometer）

加速度传感器的类型常量是 `Sensor.TYPE_ACCELEROMETER`。`values` 数组中 3 个元素的含义如下。

- `values[0]`: 沿 x 轴方向加速度。
- `values[1]`: 沿 y 轴方向加速度。
- `values[2]`: 沿 z 轴方向加速度，也就是重力加速度。

根据手机移动的速度不同，`values` 数组中的 3 个值会不断变化。例如，将手机平放在桌子上（桌子尽量平整），并将手机从左侧向右侧加速推动，`values[0]` 的值会大于 0。如果只将手机平放在桌子上，`values[2]` 的值会在 9.81 左右，这是地球的重力加速度（每秒 9.8 米）。如果要获得手机的实际加速度，需要用 `values[2]` 减去 9.81。如果将手机面朝上水平放在桌子上，并垂直向天空方向加速移动，手机的实际加速度是  $A \text{ m/s}^2$ ，那么 `values[2]` 的值就是  $(A + 9.81) \text{ m/s}^2$ 。

### 16.1.3 重力传感器（Gravity）

加速度传感器的类型常量是 `Sensor.TYPE_GRAVITY`，重力传感器与加速度传感器使用同一套坐标系。`values` 数组中 3 个元素分别表示了 x 轴、y 轴、z 轴的重力大小。Android SDK 定义了一些

常量，用于表示星系中行星、卫星和太阳表面的重力。下面就来温习一下天文知识。

```
// 太阳 (Sun) 表面的重力
public static final float GRAVITY_SUN = 275.0f;
// 水星 (Mercury) 表面的重力
public static final float GRAVITY_MERCURY = 3.70f;
// 金星 (Venus) 表面的重力
public static final float GRAVITY_VENUS = 8.87f;
// 地球 (Earth) 表面的重力，也就是我们所熟悉的 9.8m/s^2
public static final float GRAVITY_EARTH = 9.80665f;
// 月球 (Moon) 表面的重力
public static final float GRAVITY_MOON = 1.6f;
// 火星 (Mars) 表面的重力
public static final float GRAVITY_MARS = 3.71f;
// 木星 (Jupiter) 表面的重力
public static final float GRAVITY_JUPITER = 23.12f;
// 土星 (Saturn) 表面的重力
public static final float GRAVITY_SATURN = 8.96f;
// 天王星 (Uranus) 表面的重力
public static final float GRAVITY_URANUS = 8.69f;
// 海王星 (Neptune) 表面的重力
public static final float GRAVITY_NEPTUNE = 11.0f;
// 冥王星 (Pluto) 表面的重力，虽然冥王星已不属于太阳系的行星了，但还是算它一个吧！
public static final float GRAVITY_PLUTO = 0.6f;
/* 死亡星球 (白矮星) 表面的重力。不过这个常量定义得好像有些问题。白矮星体积是很小，但质量很大，最终有可能形成黑洞。可能是因为白矮星的重力很大，无法用 float 类型的值表示，所有设了个很小的值来反向表示。不过这个常量基本上很少用，我们权且将它当成一个普通的小数吧！ */
public static final float GRAVITY_DEATH_STAR_I = 0.000000353036145f;
// 岛屿表面的重力
public static final float GRAVITY_THE_ISLAND = 4.815162342f;
```

#### 16.1.4 光线传感器 (Light)

光线传感器的类型常量是 Sensor.TYPE\_LIGHT，values 数组只有第一个元素 (values[0]) 有意义，表示光线的强度，最大的值是 120000.0f。Android SDK 将光线强度分为不同的等级，每一个等级的最大值由一个常量表示，这些常量都定义在 SensorManager 类中，代码如下：

```
// 最强的光线强度（估计只有沙漠地带才能达到这个值）
public static final float LIGHT_SUNLIGHT_MAX = 120000.0f;
// 万里无云时阳光直射的强度
public static final float LIGHT_SUNLIGHT = 110000.0f;
// 有阳光，但被云彩抵消了部分光线时的强度
public static final float LIGHT_SHADE = 20000.0f;
// 多云时的光线强度
public static final float LIGHT_OVERCAST = 10000.0f;
// 太阳刚刚升起时（日出）的光线强度
public static final float LIGHT_SUNRISE = 400.0f;
// 在阴雨天，没有太阳时的光线强度
public static final float LIGHT_CLOUDY = 100.0f;
// 夜晚有月亮时的光线强度
public static final float LIGHT_FULLMOON = 0.25f;
```

```
// 夜晚没有月亮时的光线强度（当然，也不能有路灯，就是漆黑一片）
public static final float LIGHT_NO_MOON      = 0.001f;
```

上面的 8 个常量只是临界值，读者在实际使用光线传感器时要根据实际情况确定一个范围。例如，当太阳逐渐升起时，values[0]的值很可能会超过 LIGHT\_SUNRISE。当 values[0]的值逐渐增大时，就会逐渐越过 LIGHT\_OVERCAST，而达到 LIGHT\_SHADE，当然，如果天特别好的话，也可能会达到 LIGHT\_SUNLIGHT，甚至更高。

### 16.1.5 陀螺仪传感器 ( Gyroscope )

陀螺仪传感器的类型常量是 Sensor.TYPE\_GYROSCOPE，values 数组的 3 个元素表示的含义如下。

- values[0]: 沿 x 轴旋转的角速度。
- values[1]: 沿 y 轴旋转的角速度。
- values[2]: 沿 z 轴旋转的角速度。

当手机逆时针旋转时，角速度为正值；顺时针旋转时，角速度为负值。陀螺仪传感器经常被用来计算手机已转动的角度，代码如下：

```
private static final float NS2S = 1.0f / 1000000000.0f;
private float timestamp;
public void onSensorChanged(SensorEvent event)
{
    if (timestamp != 0)
    {
        // event.timestamp 表示当前的时间，单位是纳秒（一百万分之一毫秒）
        final float dT = (event.timestamp - timestamp) * NS2S;
        angle[0] += event.values[0] * dT;
        angle[1] += event.values[1] * dT;
        angle[2] += event.values[2] * dT;
    }
    timestamp = event.timestamp;
}
```

上面代码中通过陀螺仪传感器相邻两次获得数据的时间差 (dT) 来分别计算在这段时间内手机沿 x 轴、y 轴、z 轴旋转的角度，并将值分别累加到 angle 数组的不同元素上。

#### 注意

由于一般的手机带的只是初级的陀螺仪传感器，很容易受到外部因素的影响，因此，通过 values 数组获得的角速度并不一定很准确，读者在使用时要注意这一点。

### 16.1.6 方向传感器 ( Orientation )

方向传感器的类型常量是 Sensor.TYPE\_ORIENTATION，values 数组的 3 个元素表示的含义如下。

- values[0]: 该值表示方位，也就是手机绕着 z 轴旋转的角度。0 表示北 (North)；90 表示东 (East)；180 表示南 (South)；270 表示西 (West)。如果 values[0]的值正好是这 4 个值，并且手机是水平放置，表示手机的正前方就是这 4 个方向，可以利用这个特性来实现电子罗盘。

- values[1]: 该值表示倾斜度，或手机翘起的程度，当手机绕着 x 轴倾斜时该值发生变化。values[1]的取值范围是  $-180 \leqslant \text{values}[1] \leqslant 180$ ，假设将手机屏幕朝上水平放在桌子上，这时如果桌

子是完全水平的，`values[1]`的值应该是 0（由于很少有桌子是绝对水平的，因此该值很可能不为 0，但一般都是−5 和 5 之间的某个值）。这时从手机顶部开始抬起，直到将手机沿 x 轴旋转 180°（屏幕向下水平放在桌面上），在这个旋转过程中，`values[1]`会在 0 到−180 之间变化，也就是说，从手机顶部抬起时，`values[1]`的值会逐渐变小，直到等于−180。如果从手机底部开始抬起，直到将手机沿 x 轴旋转 180 度，这时 `values[1]`会在 0 到 180 之间变化，也就是说 `values[1]`的值会逐渐增大，直到等于 180。可以利用 `values[1]`和下面要介绍的 `values[2]`来测量桌子等物体的倾斜度。

- `values[2]`：表示手机沿着 y 轴的滚动角度，取值范围是−90≤`values[2]`≤90。假设将手机屏幕朝上水平放在桌面上，这时如果桌面是平的，`values[2]`的值应为 0。将手机左侧逐渐抬起时，`values[2]`的值逐渐变小，直到手机垂直于桌面放置，这时 `values[2]`的值是−90。将手机右侧逐渐抬起时，`values[2]`的值逐渐增大，直到手机垂直于桌面放置，这时 `values[2]`的值是 90。在垂直位置时继续向右或向左滚动，`values[2]`的值会继续在−90 至 90 之间变化。

### 扩展学习

在 Android 2.3 的 SDK 中并不推荐使用 `Sensor.TYPE_ORIENTATION`，但经笔者测试，Google Nexus S 手机上 `values[1]`和 `values[2]`的值都没有问题，而 `values[0]`的值有异常。因此，很多电子罗盘程序在 Android 2.3 上会出现错误的指向。官方文档推荐使用 `SensorManager.getOrientation` 方法来获得正确的方向。

```
float[] R = new float[16];
float[] I = new float[16];
float[] outR = new float[16];
sensorManager.getRotationMatrix(R, I, accelerometerValues, geomagneticValues);
sensorManager.remapCoordinateSystem(R,
    SensorManager.AXIS_Z, SensorManager.AXIS_MINUS_X, outR);
float[] actual_orientation = new float[3];
// 获得新的方向数据
actual_orientation = sensorManager.getOrientation(outR, actual_orientation);
String orientation = "方向\n" + "X: " + actual_orientation [0] + "\n"
    + "Y: " + actual_orientation [1] + "\n" + "Z: "
    + actual_orientation [2] + "\n";
tvOrientation.setText(orientation);
```

其中 `accelerometerValues` 和 `geomagneticValues` 都是 `float[]`类型变量，分别是加速度传感器和磁场传感器的 `values` 数组值。

#### 16.1.7 其他传感器

前面介绍了加速度传感器、重力传感器、光线传感器、陀螺仪传感器以及方向传感器，除了这些传感器外，Android SDK 还支持如下的几种传感器。关于这些传感器的使用方法以及与这些传感器相关的常量、方法，读者可以参阅官方文档。

- 近程传感器 (`Sensor.TYPE_PROXIMITY`)。
- 线性加速度传感器 (`Sensor.TYPE_LINEAR_ACCELERATION`)。
- 旋转向量传感器 (`Sensor.TYPE_ROTATION_VECTOR`)。
- 磁场传感器 (`Sensor.TYPE_MAGNETIC_FIELD`)。



- 压力传感器 (Sensor.TYPE\_PRESSURE)。
- 温度传感器 (Sensor.TYPE\_TEMPERATURE)。

### ■ 注意

虽然 Android SDK 定义了十多种传感器，但并不是每一部手机都完全支持这些传感器，例如，Google Nexus S 支持其中的 9 种传感器（不支持压力和温度传感器），而 HTC G7 只支持其中的 5 种传感器。如果使用了手机不支持的传感器，一般不会抛出异常，但也无法获得传感器传回的数据。读者在使用传感器时最好先判断当前的手机是否支持所使用的传感器。

## 16.2 输入输出技术

Android SDK 不仅提供了通过普通输入法（英文、拼音、笔划等）输入字符的功能，也提供了很多有趣的输入输出方式。例如，可以使用内置的语音识别技术来录入英文或中文字符，也可通过手势来完成一些特定的动作。当然，还可以利用 TTS 技术朗读文本。

### 16.2.1 语音识别

工程目录: src\ch16\ch16\_voice\_recognizer

语音识别实际上也是通过 Activity Action 调用系统提供的一个 Activity 来完成的，这个 Action 是 RecognizerIntent.ACTION\_RECOGNIZE\_SPEECH。在调用这个 Activity 之前，需要设置如下的识别模型。

- RecognizerIntent.LANGUAGE\_MODEL\_FREE\_FORM：从本机搜索识别数据。
- RecognizerIntent.LANGUAGE\_MODEL\_WEB\_SEARCH：从 Web 上搜索识别数据。

一般使用上面哪个识别模型都可以，第 1 个模型并不需要手机连入互联网，但识别的准确率略差一些，而第 2 个模型需要手机连入互联网，但识别率较高。显示语音识别窗口后，就可以说话了，等待系统成功识别后，会自动关闭识别窗口，这时会通过 onActivityResult 方法返回识别出的字符串。本例的完整代码如下：

```
package mobile.android.ch16.voice.recognizer;

import java.util.ArrayList;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.speech.RecognizerIntent;
import android.view.View;
import android.widget.EditText;

public class Main extends Activity
{
    private EditText editText;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
    }
}
```

```

        setContentView(R.layout.main);
        editText = (EditText) findViewById(R.id.edittext);
    }
    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
    {
        super.onActivityResult(requestCode, resultCode, data);
        if (requestCode == 1)
        {
            if (resultCode == Activity.RESULT_OK)
            {
                // 获得所有可能识别出的字符串
                ArrayList<String> matches = data
                    .getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
                if (matches.size() > 0)
                {
                    // 取第一个识别出的字符串，显示在 EditText 控件中
                    editText.setText(matches.get(0));
                }
            }
        }
    }
    public void onClick_Voice_Recognizer(View view)
    {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                       RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "语音录入");
        // 显示语音识别窗口
        startActivityForResult(intent, 1);
    }
}

```

运行本例，单击“语音识别”按钮，并说一句话，会看到语音识别窗口正在处理，如图 16.2 所示。当成功识别后，会调用 `onActivityResult` 方法返回识别结果，并将结果显示在按钮下方的 `EditText` 控件中。



▲ 图 16.2 语音识别

## 16.2.2 手势输入

工程目录: src\ch16\ch16\_gesture

看到“手势”这个词，千万不要以为是像哑语一样的动作手势，实际上，这里的手势就是指手写输入，只是叫“手势”更形象些。在手机中经常会使用手写输入，这就是所谓的手势。本小节要介绍的手势与手写输入法类似，但不同的是手写输入法一次只能输入一个汉字或字母，这里介绍的每个手势可以对应一个字符串或其他更复杂的内容，也就是说，通过在手机屏幕上画一个手势，可以直接输入一个字符串、图像等内容。除此之外，还可以将某个手势与指定的应用程序相关联，例如，通过手势可以拨打电话。

在使用手势之前，需要建立一个手势文件。在识别手势时，需要装载这个手势文件，并通过手势文件中的描述来识别手势。

从 Android 1.6 开始，发行包中都带了一个 GestureBuilder 工程，该工程可用来建立手势文件，读者可以在<Android SDK 安装目录>\samples\android-9 目录中找到该工程。如果读者使用的是其他 Android 版本，需要将 android-9 改成其他的名字，例如 android-8。

在模拟器或手机上安装并运行 GestureBuilder，会显示如图 16.3 所示的界面，单击“Add gesture”按钮增加一个手势。在增加手势界面上方的文本框输入一个手势名（在识别手势后，系统会返回该名称），并在下方的空白处随意画一些手势轨迹，如图 16.4 所示。要注意的是，系统允许多个手势对应于同一个手势名。读者可以采用同样的方法多增加几个手势。在创建完手势后，读者会看到 SD 卡的根目录多了一个 gestures 文件，该文件是二进制格式。



▲ 图 16.3 手势创建器的主界面



▲ 图 16.4 增加一个手势

要想使用预先创建的手势，首先要使用下面的代码装载手势文件（gesture），代码如下：

```
gestureLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
if (gestureLibrary.load())
```

```

{
    setTitle("手势文件装载成功(输出文本).");
    GestureOverlayView gestureOverlayView = (GestureOverlayView) findViewById(R.id.gestures);
    // 添加接收手势识别的事件
    gestureOverlayView.addOnGesturePerformedListener(this);
}
else
{
    setTitle("手势文件装载失败.");
}

```

本例将前面创建的 `gesture` 文件放到 `res\raw` 目录中，上面的代码涉及了一个绘制手势的 `GestureOverlayView` 控件。由于 `GestureOverlayView` 并不属于标准的 Android SDK 控件，因此，在引用时要使用 `package +classname` 的形式，代码如下：

```

<android.gesture.GestureOverlayView
    android:id="@+id/gestures" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gestureStrokeType="multiple" />

```

其中 `android:gestureStrokeType` 属性值为 `multiple`，表示可以绘制多个手势（一个手势需要一笔画出）。

最后一步就是实现 `OnGesturePerformedListener` 接口，并在 `onGesturePerformed` 方法中处理手势识别结果，代码如下：

```

public void onGesturePerformed(GestureOverlayView overlay, Gesture gesture)
{
    // 获得可能与手势对应的结果
    ArrayList<Prediction> predictions = gestureLibrary.recognize(gesture);
    if (predictions.size() > 0)
    {
        StringBuilder sb = new StringBuilder();
        int n = 0;
        if (predictions.size() > 0)
        {
            // 获得识别率
            Prediction prediction = predictions.get(0);
            // 如果 prediction.score 大于 1，说明当前绘制的手势已很接近预先定义的手势
            if (prediction.score > 1.0)
            {
                sb.append("匹配结果，name:" + prediction.name);
            }
            // 通过 Toast 信息框显示识别结果
            Toast.makeText(this, sb.toString(), Toast.LENGTH_SHORT).show();
        }
    }
}

```

我们在前面已预定义了如图 16.5 所示的手势，现在运行本例，在屏幕上绘制如图 16.6 所示的手势，会显示如图 16.7 所示的识别结果。



▲图 16.5 预定义的手势



▲图 16.6 绘制的手势

匹配结果，name:今天很热

▲图 16.7 识别结果

### 多学一招：通过手势打电话

每成功识别一个手势，就会返回与其对应的手势信息，其中包括了定义手势时输入的名称。我们也可以利用这个名称来调用其他的应用程序或做任何事件。例如，下面的代码通过手势名来调用拨打电话的程序。

```
if ("action_call".equals(prediction.name))
{
    // 拨打电话
    Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:12345678"));
}
```

### 16.2.3 语音朗读（TTS）

工程目录: src\ch16\ch16\_tts

方便输入信息还不够，如果让手机根据文本读出输入的内容那岂不是更人性化了。Android SDK 从 1.6 开始提供的 TTS（Text To Speech）技术可以完成这个工作。

TTS 技术的核心是 android.speech.tts.TextToSpeech 类，要想使用 TTS 技术朗读文本，需要做如下两项工作。

- 初始化 TTS，如指定朗读的语言。
- 指定要朗读的文本。

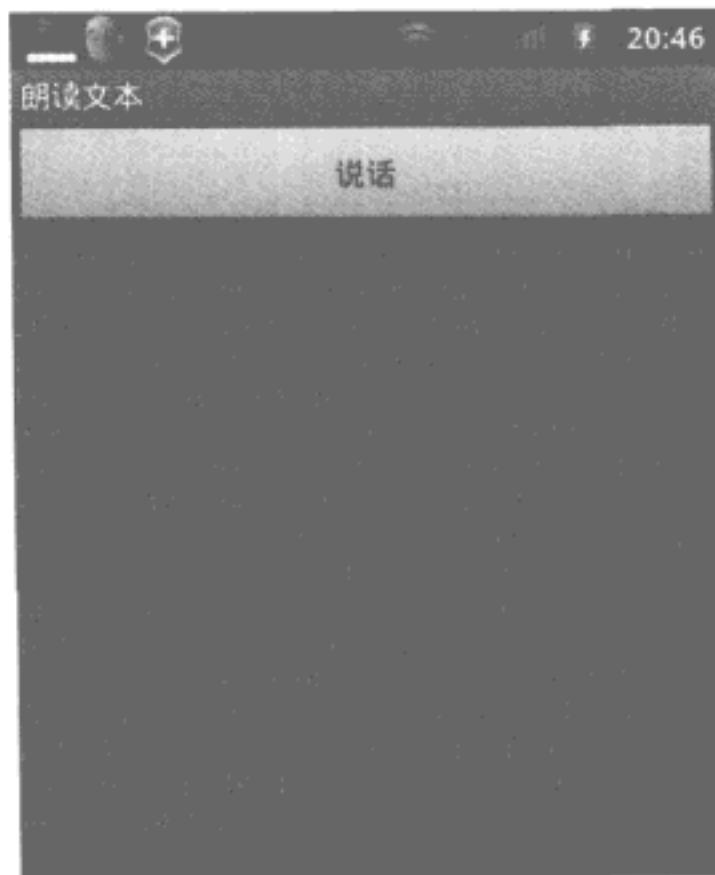
初始化 TTS 需要在 `onInit` 事件方法中完成，要使用该事件方法需要实现 `TextToSpeech.OnInitListener` 接口，在本例中的 `Main` 类实现了这个接口。创建 `TextToSpeech` 对象的代码如下：

```
// tts 是 TextToSpeech 类型的对象，构造方法的第一个参数是 Context 类型的值，第二个参数需要
// 指定 TextToSpeech.OnInitListener 对象实例
tts = new TextToSpeech(this, this);
初始化 TTS 的代码如下：
public void onInit(int status)
{
    if (status == TextToSpeech.SUCCESS)
    {
        // 指定当前朗读的语言是英文
        int result = tts.setLanguage(Locale.US);
        if (result == TextToSpeech.LANG_MISSING_DATA
            || result == TextToSpeech.LANG_NOT_SUPPORTED)
        {
            Toast.makeText(this, "Language is not available.", Toast.LENGTH_SHORT).show();
        }
    }
}
```

下面的代码为使用 `speak` 方法朗读文本。

```
public void onClick(View view)
{
    tts.speak(textView.getText().toString(), TextToSpeech.QUEUE_FLUSH, null);
}
```

其中 `speak` 方法的第一个参数表示要朗读的文本。运行本例，单击“说话”按钮，会朗读按钮下方的文字，如图 16.8 所示。



▲ 图 16.8 朗读文本

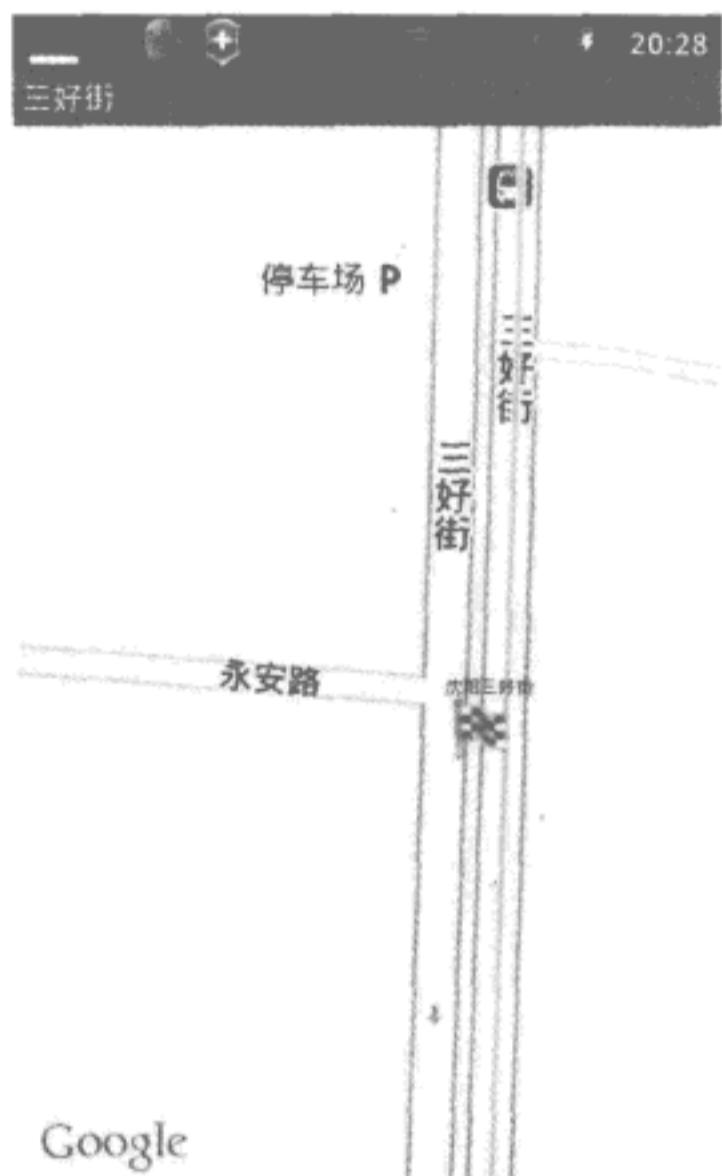
**■ 注意**

目前 TTS 只支持以英语为首的几种欧美语言，中文、日文等亚洲语言暂不支持。

## 16.3 Google 地图

工程目录: src\ch16\ch16\_google\_map

Google 是以搜索引擎闻名,但 Google 不仅仅有搜索引擎,Google 著名的代码托管服务(<http://code.google.com>)包含了大量官方及第三方的优秀应用,其中 Android 就是最受关注的应用之一,除此之外,Google Maps API 也被大量应用在各种场合。通过 Google Maps API 可以从 Google 下载地图,并通过经纬度或地点描述来确定具体的位置。本节的例子就是通过 Google 地图和 Google Maps API 来显示“沈阳三好街”的具体位置,并在该位置显示图像和文字作为标记,如图 16.9 所示。



▲ 图 16.9 Google 地图定位

下载和访问 Google 地图需要用到 com.google.android.maps.MapView 类,在开发基于 Google 地图的程序时需要使用支持 Google API 的 AVD 设备,在运行模拟器时也应启动支持 Google API 的模拟器。MapView 控件和其他控件的使用方法类似,只需要在布局文件中定义即可,但要想成功下载 Google 地图,还需要申请一个密钥。此密钥可以通过如下地址免费申请:

| <http://code.google.com/intl/zh-CN/android/maps-api-signup.html>

在申请密钥之前,需要一个用于签名的密钥文件,一般以.keystore 结尾。如果用于开发程序,可以使用 debug.keystore 文件。该文件的路径如下:

| C:\Documents and Settings\Administrator\.android\debug.keystore

### 技巧点拨：

单击 Eclipse 的“Window”>“Preferences”菜单项，打开“Preferences”对话框，在左侧找到 Android 节点，单击“Build”子节点，在右侧的“Build”设置页中的“Default debug keystore”文本框的值就是 debug.keystore 文件的路径。

在 Windows 控制台中进入 C:\Documents and Settings\Administrator.android 目录，并输入如下命令：

```
| keytool -list -keystore debug.keystore
```

当要求输入密码时，输入 android，按回车键后会在控制台中显示 debug.keystore 文件的认证指纹，如图 16.10 所示白色框中的内容。

获得认证指纹后，在申请密钥的页面下方选中同意协议复选框，并在“My certificate's MD5 fingerprint”文本框中输入认证指纹，单击“Generate API Key”按钮。如果输入的认证指纹是有效的，会产生一个新的页面，该页面中包含了申请的密钥，如图 16.11 所示黑色框中的内容。



▲ 图 16.10 获得 debug.keystore 的认证指纹



▲ 图 16.11 成功获得密钥



在获得密钥后，需要在 XML 布局文件中定义 MapView 控件，并使用 android:apiKey 属性指定这个密钥，代码如下：

```
<com.google.android.maps.MapView  
    android:id="@+id/mapview" android:layout_width="fill_parent"  
    android:layout_height="fill_parent" android:apiKey=" 0OGUYtKYCpMsLowG1Z6Ck142ja  
    Ji3fz1LAowAdg" />
```

在使用MapView 控件时应注意如下几点。

- 由于MapView 控件在 com.google.android.maps 包中，该包属于 Android SDK 附带的 jar 包（在<Android SDK 安装目录>\add-ons 目录中可以找到相应 Android SDK 版本的 jar 文件），并不属于 Android SDK 的标准控件，因此，在定义MapView 控件时必须带上包名。
- 虽然安装 ADT 会产生一个 debug.keystore 文件，但在笔者机器上的 debug.keystore 和读者机器上的 debug.keystore 是不一样的。读者在运行本例之前，应先使用自己机器上的 debug.keystore 文件获得认证指纹，并申请密钥，再用所获得的密钥替换 android:apiKey 属性值。
- 如果读者要发布基于 Google Map 的应用程序，需要使用自己生成的 keystore 文件重新获得密钥，android:apiKey 的值应为新获得的密钥。
- 由于MapView 需要访问互联网，因此，在 AndroidManifest.xml 文件中需要使用 android.permission. INTERNET 打开互联网访问权限。

在地图中定位需要使用经度和纬度，本例通过 Geocoder 类的 getFromLocationName 方法获得了“沈阳三好街”的准确位置（经纬度），并根据获得的经纬度创建了 GeoPoint 对象，以便在地图上定位。如果想在地图上添加其他元素，例如添加一个图像，可以使用 Overlay 对象。通过 Overlay.draw 方法可以在地图上绘制任意的图形。本例的完整代码如下：

```
package mobile.android.ch16.google.map;  
  
import java.util.List;  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.graphics.Point;  
import android.location.Address;  
import android.location.Geocoder;  
import android.os.Bundle;  
import android.view.Menu;  
import com.google.android.maps.GeoPoint;  
import com.google.android.maps.MapActivity;  
import com.google.android.maps.MapController;  
import com.google.android.maps.MapView;  
import com.google.android.maps.Overlay;  
  
public class Main extends MapActivity  
{  
    private MapController mapController;  
    private GeoPoint geoPoint;  
    @Override
```

```

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 从 XML 布局文件获得MapView 对象
    MapView mapView = (MapView) findViewById(R.id.mapview);
    // 允许通过触摸拖动地图
    mapView.setClickable(true);
    // 当触摸地图时在地图下方会出现缩放按钮，几秒后就会消失
    mapView.setBuiltInZoomControls(true);
    // 获得 MapController 对象，mapController 是一个在类中定义的 MapController 类型变量
    mapController = mapView.getController();
    // 创建 Geocoder 对象，用于获得指定地点的地址
    Geocoder gc = new Geocoder(this);
    // 将地图设为 Traffic 模式
    mapView.setTraffic(true);
    try
    {
        // 查询指定地点的地址
        List<Address> addresses = gc.getFromLocationName("沈阳三好街", 5);
        // 根据经纬度创建 GeoPoint 对象
        geoPoint = new GeoPoint(
            (int) (addresses.get(0).getLatitude() * 1E6),
            (int) (addresses.get(0).getLongitude() * 1E6));
        setTitle(addresses.get(0).getFeatureName());
    }
    catch (Exception e)
    {
    }
    // 创建 MyOverlay 对象，用于在地图上绘制图形
    MyOverlay myOverlay = new MyOverlay();
    // 将 MyOverlay 对象添加到 MapView 控件中
    mapView.getOverlays().add(myOverlay);
    // 设置地图的初始大小，范围在 1 和 21 之间。1：最小尺寸，21：最大尺寸
    mapController.setZoom(20);
    // 以动画方式进行定位
    mapController.animateTo(geoPoint);
}
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    return super.onCreateOptionsMenu(menu);
}
@Override
protected boolean isRouteDisplayed()
{
    return false;
}
class MyOverlay extends Overlay
{
    @Override
    public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
    {
}

```

```
Paint paint = new Paint();
paint.setColor(Color.RED);
Point screenPoint = new Point();
// 将“沈阳三好街”在地图上的位置转换成屏幕的实际坐标
mapView.getProjection().toPixels(geoPoint, screenPoint);
Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.target);
// 在地图上绘制图像
canvas.drawBitmap(bmp, screenPoint.x, screenPoint.y, paint);
// 在地图上绘制文字
canvas.drawText("沈阳三好街", screenPoint.x, screenPoint.y, paint);
return super.draw(canvas, mapView, shadow, when);
}
```

在编写上面代码时应注意如下几点。

- MapView 有 3 种地图模式，分别是交通（Traffic）、卫星（Satellite）和街景（StreetView）。在上面的代码中使用 `setTraffic` 方法将地图设成交通地图模式，还可以通过 `setSatellite` 和 `setStreetView` 方法将地图设成卫星和街景视图。

## 16.4 GPS 定位

工程目录: src\ch16\ch16\_gps

要想定位到当前的位置,需要利用手机中的 GPS 模块,使用 GPS 首先需要获得 LocationManager 服务,代码如下:

```
LocationManager locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

通过 LocationManager 类的 getBestProvider 方法可以获得当前的位置，但需要通过 Criteria 对象指定一些参数，代码如下：

```
Criteria criteria = new Criteria();
// 获得最好的定位效果
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setCostAllowed(false);
// 使用省电模式
criteria.setPowerRequirement(Criteria.POWER_LOW);
```

## Android 开发权威指南

```
// 获得当前的位置提供者
String provider = locationManager.getBestProvider(criteria, true);
// 获得当前的位置
Location location = locationManager.getLastKnownLocation(provider);
// 获得当前位置的纬度
Double latitude = location.getLatitude() * 1E6;
// 获得当前位置的经度
Double longitude = location.getLongitude() * 1E6;
```

在获得当前位置的经纬度后，剩下的工作就和 16.3 节的例子一样了，只是在本例中还输出了与当前位置相关的信息，代码如下：

```
Geocoder gc = new Geocoder(this);
List<Address> addresses = gc.getFromLocation(location.getLatitude(), location.getLongitude(), 1);
if (addresses.size() > 0)
{
    msg += "AddressLine: " + addresses.get(0).getAddressLine(0) + "\n";
    msg += "CountryName: " + addresses.get(0).getCountryName() + "\n";
    msg += "Locality: " + addresses.get(0).getLocality() + "\n";
    msg += "FeatureName: " + addresses.get(0).getFeatureName();
}
textView.setText(msg);
```

本例需要使用如下代码在 AndroidManifest.xml 文件中打开相应的权限：

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

在手机上运行本节的例子，会显示如图 16.12 所示的效果。



▲ 图 16.12 定位到当前位置

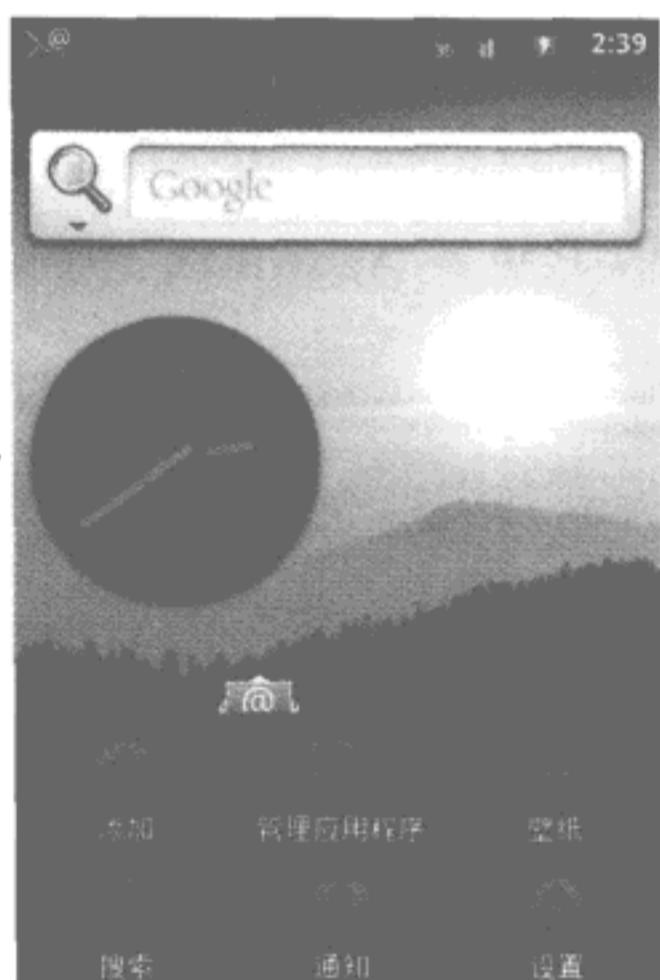
## 16.5 桌面上的小东西

对于 Android 手机玩家，Android 的桌面总是充斥着各种各样的小玩意，真叫人眼花缭乱。除了 Android 自带的天气预报、各种图标等，还有第三方应用带的各式窗口小部件（AppWidget），快速查看数据的文件夹。当然，从 Android 2.1 开始，还支持动态壁纸（将在 16.6 节详细介绍），从而使 Android 桌面更生动活泼。

### 16.5.1 窗口小部件（AppWidget）

工程目录：src\ch16\ch16\_appwidget

从 Android SDK 1.5 开始，AppWidget 被引入到 Android SDK 中。开发人员可以利用 Android SDK 提供的框架开发 Widget，这些 Widget 可以被拖到桌面以便和用户交互。在 Android 模拟器或 Android 手机中都包含大量的 Widget。将 Widget 放到桌面上也非常简单，在 Android 模拟器中可以单击“Menu”按钮，在弹出的选项菜单中单击“添加”菜单项（如图 16.13 所示），并在弹出的子菜单中单击“窗口小部件”菜单项来添加相应的 Widget。添加 Widget 后的桌面效果如图 16.14 所示。在手机上添加 Widget，会根据不同型号的手机有所不同，例如，HTC 的 Android 手机采用了 HTC Sense 界面，可以通过单击屏幕右下方的加号（+）按钮，在弹出的列表中单击“窗口小部分”来选择要添加的 Widget。当然，也可以通过单击手机下方的“Menu”按钮，并在弹出的选项菜单中单击“添加到主页”菜单项，会弹出与按加号（+）按钮同样的选项列表。而 Google Nexus S 手机由于采用了标准的 Android 界面，因此，需要按手机下方的“Menu”按钮（触摸式按钮）弹出选项列表。大多数 Android 手机通过长按屏幕的空白处也可以弹出选项列表。



▲ 图 16.13 在模拟器桌面上添加 Widget



▲ 图 16.14 在真机桌面上添加 Widget ( HTC Hero )

下面给出一个例子来演示如何实现一个 AppWidget。这个例子可以将 EditText 控件中的字符串显示在 AppWidget 中，并显示当天的日期，而且每 24 小时日期会自动更新。实现 AppWidget 的步骤如下。

### (1) 编写 AppWidget 的布局文件 (appwidget\_provider.xml)

```
<?xml version="1.0" encoding="utf-8"?>
// 用于显示 EditText 控件输入的内容和当前的日期
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/appwidget_text" android:layout_width="wrap_content"
    android:layout_height="wrap_content" android:textColor="#FF000000"/>
```

AppWidget 并不支持所有的 Android SDK 标准控件，当然，也不支持第三方的控件。AppWidget 支持的控件如下：

1) 用于布局的控件。

- FrameLayout;
- LinearLayout;
- RelativeLayout。

2) 可视控件类。

- AnalogClock;
- Button;
- Chronometer;
- ImageButton;
- ImageView;
- ProgressBar;
- TextView。

除了上面 10 个控件外，不能在 AppWidget 的布局文件中使用其他任何控件，否则系统会抛出异常。

### (2) 编写 MyAppWidgetProvider 类。

MyAppWidgetProvider 是 AppWidget 的核心类，主要完成更新 AppWidget 中的内容、删除 AppWidget 等工作。

```
package mobile.android.ch16.appwidget;

import java.text.SimpleDateFormat;
import java.util.Date;
import android.appwidget.AppWidgetManager;
import android.appwidget.AppWidgetProvider;
import android.content.Context;
import android.widget.RemoteViews;

public class MyAppWidgetProvider extends AppWidgetProvider
{
    // 更新 AppWidget 时调用
    @Override
```

```

public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds)
{
    final int N = appWidgetIds.length;
    for (int i = 0; i < N; i++)
    {
        // 获得桌面上所有的 AppWidget 的 ID
        int appWidgetId = appWidgetIds[i];
        // 获得配置界面设置的字符串
        String titlePrefix = MyAppWidgetConfigure.loadTitlePref(context, appWidgetId);
        // 更新 AppWidget 中的内容
        updateAppWidget(context, appWidgetManager, appWidgetId, titlePrefix);
    }
}
@Override
public void onDeleted(Context context, int[] appWidgetIds)
{
    // 获得本 AppWidget 所有被删除的实例数
    final int N = appWidgetIds.length;
    for (int i = 0; i < N; i++)
    {
        // 删除所有被删除 AppWidget 的配置信息
        MyAppWidgetConfigure.deleteTitlePref(context, appWidgetIds[i]);
    }
}
static void updateAppWidget(Context context,
                            AppWidgetManager appWidgetManager, int appWidgetId,
                            String titlePrefix)
{
    SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");
    // 设置要显示在 AppWidget 中的文本
    CharSequence text = MyAppWidgetConfigure.loadTitlePref(context,
        appWidgetId) + "\n" + simpleDateFormat.format(new Date());
    // 装载在第 1 步编写的布局文件
    RemoteViews views = new RemoteViews(context.getPackageName(),
        R.layout.appwidget_provider);
    // 设置 AppWidget 显示的内容
    views.setTextViewText(R.id.appwidget_text, text);
    appWidgetManager.updateAppWidget(appWidgetId, views);
}
}

```

### (3) 配置 MyAppWidgetProvider

在 res 目录建立一个 xml 子目录, 然后在 xml 目录中建立一个 appwidget\_provider.xml 文件(与第 1 步建立的 appwidget\_provider.xml 文件虽然同名, 但含义不同, 读者要注意这一点), 并输入如下内容。

```

<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="60dp"

```

```

    android:minHeight="30dp"
    android:updatePeriodMillis="86400000"
    android:initialLayout="@layout/appwidget_provider"
    android:configure="mobile.android.ch16.appwidget.MyAppWidgetConfigure">
</appwidget-provider>

```

在编写和配置 `MyAppWidgetProvider` 类时应了解如下几点。

- `android:minWidget` 和 `android:minHeight` 属性用于设置 AppWidget 所占用的空间大小。
- `android:initialLayout` 属性用于指定 AppWidget 布局文件的资源 ID。
- `android:configure` 属性用于指定一个配置界面的资源 ID。当添加 AppWidget 时会显示这个界面，以便对 AppWidget 做更进一步地配置。
- 配置文件中的 `android:updatePeriodMillis` 属性值设为 0，表示不使用更新 Widget 的方式来更新当前时间。如果设为正整数，表示一定时间（单位：秒）内更新 AppWidget 中的内容。在本例中该属性值为 86400000，表示 1 天（24 小时）。
- 如果 `android:updatePeriodMillis` 属性值为 0，系统在装载 Widget 时仍然会调用 `onUpdate` 方法。因此，可以在该方法中做一些初始化的工作。
- 每一个 Widget 都会有一个唯一的 ID，在 `onUpdate` 方法中可能需要更新多个 Widget，因此，这些 Widget 的 ID 通过 `appWidgetIds` 参数传入 `onUpdate` 方法中。开发人员可以选择在 `onUpdate` 中更新 Widget，或在 `onUpdate` 方法中保存 Widget 的 ID，以便在其他的方法中更新 Widget。
- 当删除一个或多个 AppWidget 时，会调用 `onDeleted` 方法，并通过 `appWidgetIds` 参数传入被删除的 AppWidget，以便做更进一步地处理。例如，释放一些由 AppWidget 占用的资源。

#### (4) 编写配置界面的布局文件

在 `res\layout` 目录中建立一个 `appwidget_configure.xml` 文件，并输入如下内容。

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView android:layout_width="match_parent"
        android:layout_height="wrap_content" android:text="输入的文本会被显示在AppWidget上" />
    <EditText android:id="@+id/appwidget_prefix"
        android:layout_width="match_parent" android:layout_height="wrap_content" />
    <Button android:id="@+id/save_button" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="确定" />
</LinearLayout>

```

#### (5) 编写 `MyAppWidgetConfiguration` 类

```

package mobile.android.ch16.appwidget;
import android.app.Activity;
import android.appwidget.AppWidgetManager;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;

```

```
import android.view.View;
import android.widget.EditText;
public class MyAppWidgetConfigure extends Activity
{
    private static final String PREFS_NAME = "mobile.android.ch16.appwidget.MyAppWidget-
Provider";
    private static final String PREF_PREFIX_KEY = "prefix_";
    int mAppWidgetId = AppWidgetManager.INVALID_APPWIDGET_ID;
    EditText mAppWidgetPrefix;
    public MyAppWidgetConfigure()
    {
        super();
    }
    @Override
    public void onCreate(Bundle icicle)
    {
        super.onCreate(icicle);
        setResult(RESULT_CANCELED);
        setContentView(R.layout.appwidget_configure);
        mAppWidgetPrefix = (EditText) findViewById(R.id.appwidget_prefix);
        findViewById(R.id.save_button).setOnClickListener(mOnClickListener);
        Intent intent = getIntent();
        Bundle extras = intent.getExtras();
        if (extras != null)
        {
            // 获得当前添加的 AppWidget 的 ID
            mAppWidgetId = extras.getInt(AppWidgetManager.EXTRA_APPWIDGET_ID,
                AppWidgetManager.INVALID_APPWIDGET_ID);
        }
        if (mAppWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID)
        {
            finish();
        }
        // 设置 EditText 控件的初始值
        mAppWidgetPrefix.setText(loadTitlePref(MyAppWidgetConfigure.this, mAppWidgetId));
    }
    View.OnClickListener mOnClickListener = new View.OnClickListener()
    {
        // 确定按钮的单击事件
        public void onClick(View v)
        {
            final Context context = MyAppWidgetConfigure.this;
            String titlePrefix = mAppWidgetPrefix.getText().toString();
            // 保存设置
            saveTitlePref(context, mAppWidgetId, titlePrefix);
            AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
            MyAppWidgetProvider.updateAppWidget(context, appWidgetManager,
                mAppWidgetId, titlePrefix);
            Intent resultValue = new Intent();
            // 传回当前设置的 AppWidget 的 ID
            resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
            setResult(RESULT_OK, resultValue);
        }
    };
}
```

```

        finish();
    }
}

// 保存 EditText 控件中输入的内容
static void saveTitlePref(Context context, int appWidgetId, String text)
{
    SharedPreferences.Editor prefs = context.getSharedPreferences(
        PREFS_NAME, 0).edit();
    prefs.putString(PREF_PREFIX_KEY + appWidgetId, text);
    prefs.commit();
}

// 装载配置信息 (EditText 控件中输入的内容)
static String loadTitlePref(Context context, int appWidgetId)
{
    SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, 0);
    String prefix = prefs.getString(PREF_PREFIX_KEY + appWidgetId, null);
    if (prefix != null)
    {
        return prefix;
    }
    else
    {
        return context.getString(R.string.appwidget_prefix_default);
    }
}

// 删除配置信息
static void deleteTitlePref(Context context, int appWidgetId)
{
    SharedPreferences prefs = context.getSharedPreferences(PREFS_NAME, 0);
    String prefix = prefs.getString(PREF_PREFIX_KEY + appWidgetId, null);
    prefs.edit().remove(prefix).commit();
}
}

```

#### (6) 在 AndroidManifest.xml 文件中配置 MyAppWidgetProvider 和 MyAppWidgetConfiguration。

```

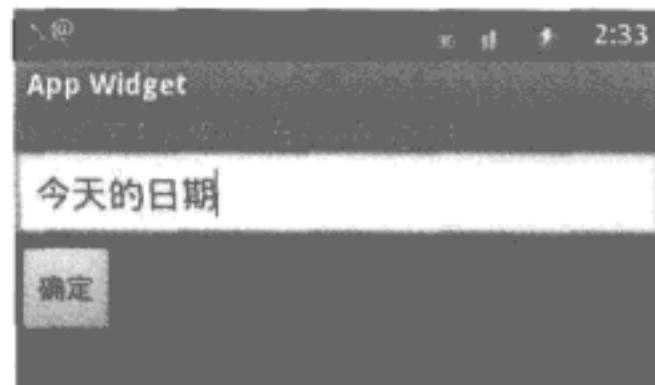
<receiver android:name=".MyAppWidgetProvider">
    <meta-data android:name="android.appwidget.provider"
        android:resource="@xml/appwidget_provider" />
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
</receiver>
<activity android:name=".MyAppWidgetConfigure">
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
    </intent-filter>
</activity>

```

运行本例，单击“Menu”按钮，在弹出的列表中单击“窗口小部件”，会显示如图 16.15 所示的列表，选择“App Widget”，会弹出如图 16.16 所示的界面，在 EditText 控件中输入一个字符串，单击“确定”按钮，就会在桌面上显示所输入的字符串以及当前的日期，如图 16.17 所示。



▲ 图 16.15 添加 AppWidget



▲ 图 16.16 配置 AppWidget



▲ 图 16.17 AppWidget 的效果

## 16.5.2 快捷方式

工程目录: src\ch16\ch16\_shortcut

像 Windows、Linux 等图形操作系统一样, Android 也可以将常用的应用程序作为快捷方式放到桌面上。我们在添加 AppWidget 时发现在列表菜单中有一个“快捷方式”菜单项, 单击这个菜单项, 会列出当前手机中所有可放到桌面上的快捷方式, 我们可以直接将快捷方式添加到这个菜单中, 也可以将快捷方式添加到桌面上(并不在这个菜单中显示)。图 16.18 是模拟器上的快捷方式列表, 其中黑框中的“打电话”菜单项就是本节要添加的快捷方式。这个快捷方式可以显示拨号界面, 并自动输入一个固定的电话号。在添加这个快捷方式的同时, 会在桌面上创建另外一个快捷方式“Wifi 设置”, 可以通过它直接显示 Wifi 设置界面。



▲ 图 16.18 快捷方式列表

所有可以接收 android.intent.action.CREATE\_SHORTCUT 动作的 Activity 都会显示在如图 16.18 所示的菜单项中，因此，我们可以使用下面的代码配置一个 Activity。

```
<activity android:name=".CallShortcut" android:label="打电话"
    android:icon="@drawable/call">
    <intent-filter>
        <action android:name="android.intent.action.CREATE_SHORTCUT" />
    </intent-filter>
</activity>
```

下面来编写 CallShortcut 类，代码如下：

```
package mobile.android.ch16.shortcut;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.os.Parcelable;

public class CallShortcut extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        if (Intent.ACTION_CREATE_SHORTCUT.equals(getIntent().getAction()))
        {
            Intent addShortcutIntent = new Intent();
            addShortcutIntent.putExtra(Intent.EXTRA_SHORTCUT_NAME, "打电话");
            // 指定桌面快捷方式的图像
            Parcelable icon = Intent.ShortcutIconResource.fromContext(this, R.drawable.call);
            // 设置桌面快捷方式的图像
            addShortcutIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, icon);
            // 指定快捷方式要调用的程序
            Intent callIntent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:12345678"));
            addShortcutIntent.putExtra(Intent.EXTRA_SHORTCUT_INTENT, callIntent);
            // 通过服务添加另一个快捷方式 (Wifi 设置)
            startService(new Intent(this, WifiShortcut.class));
            setResult(RESULT_OK, addShortcutIntent);
        }
        else
        {
            setResult(RESULT_CANCELED);
        }
        finish();
    }
}
```

当单击如图 16.18 所示的“打电话”菜单项时，系统会调用 CallShortcut，这时也就将“打电话”快捷方式添加到桌面上了。当然，也可以通过这种方式正常调用程序。在上面的代码中调用了一个服务（WifiShortcut）添加了“Wifi 设置”快捷方式。在 WifiShortcut.onStart 方法中添加了这个快捷方式，代码如下：

```
public void onStart(Intent intent, int startId)
{
    // 指定安装快捷方式的 Action
    Intent installShortCut = new Intent("com.android.launcher.action.INSTALL_SHORTCUT");
    installShortCut.putExtra(Intent.EXTRA_SHORTCUT_NAME, "Wifi 设置");
    Parcelable icon = Intent.ShortcutIconResource.fromContext(this,R.drawable.wifi);
    installShortCut.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE, icon);
    // 指定快捷方式要执行的 Action
    Intent wifiSettingsIntent = new Intent("android.settings.WIFI_SETTINGS");
    installShortCut.putExtra(Intent.EXTRA_SHORTCUT_INTENT,wifiSettingsIntent);
    // 发送安装快捷方式的广播
    sendBroadcast(installShortCut);
    super.onStart(intent, startId);
}
```

运行本例，并不会显示任何界面。长按桌面空白处，在弹出菜单中选择“快捷方式”菜单项，在弹出的快捷方式列表中选择“打电话”，系统会同时在桌面上建两个快捷方式：“打电话”和“Wifi 设置”，如图 16.19 所示。



▲ 图 16.19 快捷方式

### 16.5.3 实时文件夹

工程目录：src\ch16\ch16\_livefolder

前面介绍的快捷方式可以调用应用程序中的 Activity，而实时文件夹可以访问其他应用程序中

的数据，例如，联系人、电子邮件、短信等。

实时文件夹通过 ContentProvider 来获得其他应用程序中的数据。我们可以在 Activity.onCreate 方法中安装一个实时文件夹，代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // 判断是否为创建快捷方式的 Action
    if (LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(getIntent().getAction()))
    {
        Intent intent = new Intent();
        // 设置获得其他应用程序数据的 Uri。这个 Uri 获得了联系人信息
        intent.setData(Uri.parse("content://contacts/people"));
        // 设置单击实时文件夹中某项时（在这里是某个联系人）要进行的动作（本例为直接拨打电话）
        intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_BASE_INTENT,
                       new Intent(Intent.ACTION_CALL, Uri.parse("content://contacts/people")));
        // 设置实时文件夹在桌面上显示的标题
        intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, "电话本");
        intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON,
                       Intent.ShortcutIconResource.fromContext(this, R.drawable.phone));
        // 设置实时文件夹的显示模式，本例为列表模式，还可以设为 LiveFolders.DISPLAY_MODE_GRID 模式
        // （网格模式）
        intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
                       LiveFolders.DISPLAY_MODE_LIST);
        setResult(RESULT_OK, intent);
    }
    else
    {
        setResult(RESULT_CANCELED);
    }
    finish();
}
```

配置建立实时文件夹的 Activity 的代码如下：

```
<activity android:name=".AddLiveFolder" android:label="电话本" android:icon="@drawable/
phone">
    <intent-filter>
        <action android:name="android.intent.action.CREATE_LIVE_FOLDER" />
    </intent-filter>
</activity>
```

其中 android:label 属性表示在实时文件夹列表中显示的名称，android:icon 属性表示在实时文件夹列表中显示的图标。建立实时文件夹的 Activity 必须指定 CREATE\_LIVE\_FOLDER 动作，只要将 AddLiveFolder 添加到任何的 Android 应用程序中，并启动应用程序，实时文件夹会自动添加到如图 16.20 所示的列表中，其中黑色框中的实时文件夹是本例建立的。选中这个实时文件夹，会像快捷方式一样将该实时文件夹添加到桌面上，然后单击实时文件夹，会显示如图 16.21 所示的联系人列表。单击某个联系人，会直接拨打该联系人的电话。



▲ 图 16.20 实时文件夹列表



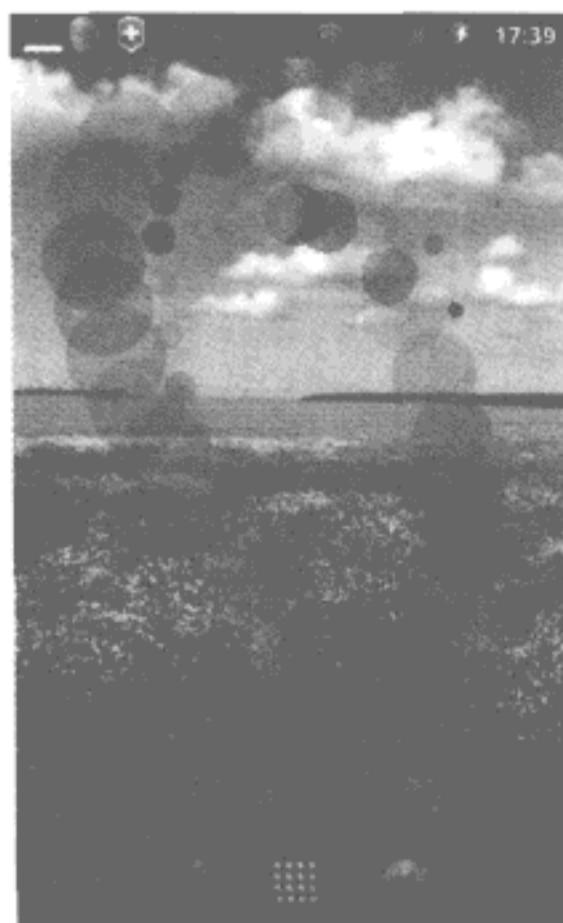
▲ 图 16.21 显示联系人列表

## 16.6 应用更华丽——动态壁纸

工程目录: `src\ch16\ch16_live_wallpaper`

在手机桌面放一张漂亮的图像是一件非常酷的事情。不过，这还不够酷。如果可以触摸桌面的空白处，会随着触摸的位置不同而发生各种变化，那岂不是更棒了。如果大家都是这么认为的，那么 Android 2.1 及以后的版本将会满足我们的要求，因为 Android 2.1 及以后的版本提供了动态壁纸，可以在上面绘制任何图像或图形，甚至是基于 3D 的渲染。

也许很多读者还不清楚什么是动态壁纸，那么现在先看一下本节实现的例子。当触摸屏幕的任何空白位置，会显示一个彩色的实心圆（颜色是随机变化的），如图 16.22 所示。要使用动态壁纸，需要在壁纸列表中选择“动态壁纸”菜单项，然后在弹出的动态壁纸列表中选择要设置的壁纸，如图 16.23 所示。



▲ 图 16.22 动态壁纸的效果



▲ 图 16.23 选择动态壁纸

动态壁纸的核心是一个服务类，该类必须继承自 android.service.wallpaper.WallpaperService 类。本例的服务类是 LiveWallpaperService，在该类中定义了一个继承自 WallpaperService.Engine 的 WallPaperEngine 类，用于处理动态壁纸的核心业务。LiveWallpaperService 类的代码如下：

```
package mobile.android.ch16.live.wallpaper;

import android.content.SharedPreferences;
import android.service.wallpaper.WallpaperService;
import android.view.MotionEvent;
import android.view.SurfaceHolder;

public class LiveWallpaperService extends WallpaperService
{
    @Override
    public Engine onCreateEngine()
    {
        return new WallPaperEngine();           // 创建动态壁纸引擎
    }
    // 定义实时壁纸引擎类
    public class WallPaperEngine extends Engine
    {
        private LiveWallpaperPainting painting;
        // 在构造方法中需要读取配置文件中的信息，以确定绘制的彩色实心圆的半径
        public WallPaperEngine()
        {
            SurfaceHolder holder = getSurfaceHolder();
            painting = new LiveWallpaperPainting(holder, getApplicationContext(), 0);
        }
        @Override
        public void onCreate(SurfaceHolder surfaceHolder)
        {
            super.onCreate(surfaceHolder);
            setTouchEventsEnabled(true);
        }
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            painting.stopPainting();
        }
        @Override
        public void onVisibilityChanged(boolean visible)
        {
            if (visible)
            {
                painting.resumePainting();
            }
            else
            {
                painting.pausePainting();
            }
        }
        @Override
        public void onSurfaceChanged(SurfaceHolder holder, int format, int width, int height)
```

```
{  
    super.onSurfaceChanged(holder, format, width, height);  
    painting.setSurfaceSize(width, height);  
}  
@Override  
public void onSurfaceCreated(SurfaceHolder holder)  
{  
    super.onSurfaceCreated(holder);  
    // 当 surface (绘制实时壁纸的界面) 创建后, 开始绘制彩色实心圆  
    painting.start();  
}  
// 当 Surface 销毁时需要停止绘制壁纸  
@Override  
public void onSurfaceDestroyed(SurfaceHolder holder)  
{  
    super.onSurfaceDestroyed(holder);  
    boolean retry = true;  
    painting.stopPainting();  
    while (retry)  
    {  
        try  
        {  
            painting.join();  
            retry = false;  
        }  
        catch (InterruptedException e)  
        {  
        }  
    }  
}  
@Override  
public void onTouchEvent(MotionEvent event)  
{  
    super.onTouchEvent(event);  
    painting.doTouchEvent(event);  
}  
}  
}
```

在上面代码中涉及到一个 LiveWallpaperPainting 类，该类通过线程不断扫描用户在屏幕上触摸的点，然后根据触摸点绘制彩色实心圆，该类的代码如下：

```
package mobile.android.ch16.live.wallpaper;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Paint;  
import android.graphics.drawable.BitmapDrawable;  
import android.view.MotionEvent;  
import android.view.SurfaceHolder;  
public class LiveWallpaperPainting extends Thread
```

```
(

    private SurfaceHolder surfaceHolder;
    private Context context;
    private boolean wait;
    private boolean run;
    /* 尺寸和半径 */
    private int width;
    private int height;
    private int radius;
    /** 触摸点 */
    private List<TouchPoint> points;
    /* 时间轨迹 */
    private long previousTime;
    public LiveWallpaperPainting(SurfaceHolder surfaceHolder, Context context, int radius)
    {
        this.surfaceHolder = surfaceHolder;
        this.context = context;
        // 直到 surface 被创建和显示时才开始动画
        this.wait = true;
        // 初始化触摸点
        this.points = new ArrayList<TouchPoint>();
        // 初始化半径
        this.radius = radius;
    }
    // 通过设置页面可以改变圆的半径
    public void setRadius(int radius)
    {
        this.radius = radius;
    }
    // 暂停实时壁纸的动画
    public void pausePainting()
    {
        this.wait = true;
        synchronized (this)
        {
            this.notify();
        }
    }
    // 恢复在实时壁纸上绘制彩色实心圆
    public void resumePainting()
    {
        this.wait = false;
        synchronized (this)
        {
            this.notify();
        }
    }
    // 停止在实时壁纸上绘制彩色实心圆
    public void stopPainting()
    {
        this.run = false;
        synchronized (this)
    }
}
```



```
{  
    this.notify();  
}  
}  
@Override  
public void run()  
{  
    this.run = true;  
    Canvas canvas = null;  
    while (run)  
    {  
        try  
        {  
            canvas = this.surfaceHolder.lockCanvas(null);  
            synchronized (this.surfaceHolder)  
            {  
                // 绘制彩色实心圆和背景图  
                doDraw(canvas);  
            }  
        } finally  
        {  
            if (canvas != null)  
            {  
                this.surfaceHolder.unlockCanvasAndPost(canvas);  
            }  
        }  
        // 如果不需要动画则暂停动画  
        synchronized (this)  
        {  
            if (wait)  
            {  
                try  
                {  
                    wait();  
                }  
                catch (Exception e)  
                {  
                }  
            }  
        }  
    }  
}  
public void setSurfaceSize(int width, int height)  
{  
    this.width = width;  
    this.height = height;  
    synchronized (this)  
    {  
        this.notify();  
    }  
}  
public void doTouchEvent(MotionEvent event)  
{  
    synchronized (this.points)  
    {  
}
```

## Android 开发权威指南

```
int color = new Random().nextInt(Integer.MAX_VALUE);
// 将用户触摸屏幕的点信息保存在 points 中，以便在 run 方法中扫描这些点，并绘制彩色实心圆
points.add(new TouchPoint((int) event.getX(), (int) event.getY(),
    color, Math.min(width, height) / this.radius));
}
this.wait = false;
synchronized (this)
{
    notify();
}
}
private void doDraw(Canvas canvas)
{
    long currentTime = System.currentTimeMillis();
    long elapsed = currentTime - previousTime;
    if (elapsed > 20)
    {
        BitmapDrawable bitmapDrawable =
            (BitmapDrawable) context.getResources().getDrawable(R.drawable.background);
        // 绘制实时壁纸的背景图
        canvas.drawBitmap(bitmapDrawable.getBitmap(), 0, 0, new Paint());
        // 绘制触摸点
        Paint paint = new Paint();
        List<TouchPoint> pointsToRemove = new ArrayList<TouchPoint>();
        synchronized (this.points)
        {
            for (TouchPoint point : points)
            {
                paint.setColor(point.color);
                point.radius -= elapsed / 20;
                if (point.radius <= 0)
                {
                    pointsToRemove.add(point);
                }
                else
                {
                    canvas.drawCircle(point.x, point.y, point.radius, paint);
                }
            }
            points.removeAll(pointsToRemove);
        }
        previousTime = currentTime;
        if (points.size() == 0)
        {
            wait = true;
        }
    }
}
// 保存绘制的彩色实心圆的信息
class TouchPoint
{
    int x;
    int y;
    int color;
    int radius;
```

```
public TouchPoint(int x, int y, int color, int radius)
{
    this.x = x;
    this.y = y;
    this.radius = radius;
    this.color = color;
}
}
```

本例还涉及几个配置文件。首先应在 `AndroidManifest.xml` 文件中配置 `LiveWallpaperService`, 代码如下:

```
<service android:name="LiveWallpaperService" android:enabled="true"
    android:icon="@drawable/icon" android:label="@string/app_name"
    android:permission="android.permission.BIND_WALLPAPER">
    <intent-filter android:priority="1">
        <action android:name="android.service.wallpaper.WallpaperService" />
    </intent-filter>
    <meta-data android:name="android.service.wallpaper"
        android:resource="@xml/wallpaper" />
</service>
```

在 `res\Xml` 目录中建立一个 `wallpaper.xml` 文件, 该文件需要在 `AndroidManifest.xml` 文件中的 `<meta-data>` 标签进行设置 (就是 `android:resource` 属性的值)。`wallpaper.xml` 文件的内容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<wallpaper xmlns:android="http://schemas.android.com/apk/res/android"
    android:thumbnail="@drawable/icon" android:description="@string/description"/>
```

## 16.7 小结

本章介绍了 Android SDK 提供的一些有趣的 API, 其中传感器 API 在很多游戏中经常被使用。GPS API 是 LBS (Location Based Service, 基于位置的服务) 的核心。除此之外, Google 地图、AppWidget、动态壁纸等应用也为 Android 手机增色不少。







## 第三部分

# 高级篇

第 17 章 HTML5 与移动 Web 开发

第 18 章 输入法开发

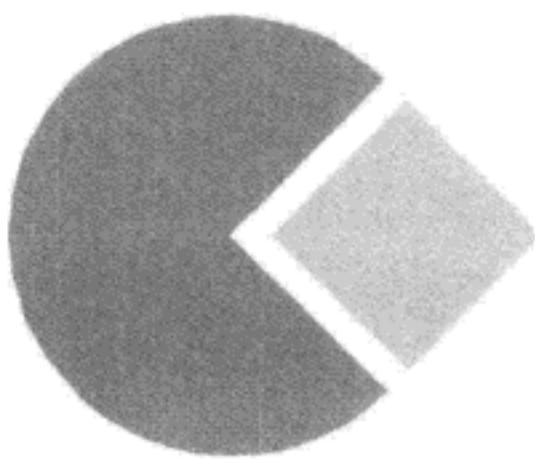
第 19 章 Android OpenGL ES 开发基础

第 20 章 OpenGL ES 的超酷效果

第 21 章 Android NDK 编程

第 22 章 测试驱动开发 (TDD)





## 第 17 章 HTML5 与移动 Web 开发

HTML 在 Web 领域已经叱咤风云多年，也是 Web 应用的三架马车（另两个是 CSS 和 JavaScript）之一。然而，由于 HTML 本身的限制，使其无法开发更绚、更酷的 Web 应用。通常这些应用都是使用 Flash、Silverlight 等富客户端工具开发的。在 PC 上使用 Flash 和 Silverlight 并没有什么不妥，但在移动领域，Flash 和 Silverlight 并没有成为所有移动操作系统的标配，iPhone 甚至连 Flash 都不支持。Silverlight 目前只能在微软的 Windows Phone 7 中运行，所以使用这两种富客户端技术，在移动领域恐怕举步维艰。但不管是哪一种移动操作系统，都支持 HTML 和 JavaScript，因此，我们可以通过 HTML 配合 JavaScript、CSS 来编写跨移动平台的应用程序。

然而，老版本的 HTML（4.x 及以前的版本）并不能实现像 Flash、Silverlight 那样丰富的渲染效果。不过这一切目前已经开始改变了，这就是本章要隆重推出的 HTML5。HTML5 在 HTML4 的基础上增加了很多超酷的特性，例如，提供了 Canvas 功能，只要有足够的想象力，就可以做出完全与 Flash 媲美的效果。Web 应用给我们的印象通常是客户端（HTML、CSS、JavaScript）只作为 UI 来展示数据，而这些数据都是保存在服务端（客户端最多通过 Cookie 存一些很简单的状态数据），不过这些印象恐怕要被 HTML5 颠覆了。HTML5 为我们增加了在 Web 客户端存储数据的能力，通过这些技术，可以很容易地实现只由静态页面组成的交互式 Web 应用。总之，HTML5 为我们带来的惊喜还很多。那么本章就将这些惊喜带给读者。

### 17.1 HTML5 简介

HTML5 是 HTML（Hyper Text Markup Language）的新标准。可以在下面的页面中查看 HTML5 规范。

<http://dev.w3.org/html5/spec/Overview.html>

HTML5 为我们提供了丰富的功能，例如，经常被使用到 Web 游戏中的 Canvas API 就是 HTML5 提供的最酷的功能之一。

Canvas（画布）的概念最初源于苹果公司的 Mac OS X 系统上使用的 WebKit 引擎（浏览器引擎，目前 Android 等手机操作系统也都使用该引擎），一开始，通过 Canvas 来实现 Dashboard Widgets（Web 版的仪表盘部件）。在引入 Canvas 之前，实现这些功能只能在网页中嵌入 Flash，或使用 SVG（Scalable Vector Graphics）、Vector Markup Language 等技术。不过在引入 Canvas 后，完全可以使用 HTML 和 JavaScript 来完成这个工作。

Web Storage（Web 存储）API 是 HTML5 的另一个重要特性。传统的 Web 程序只能通过客户

端浏览器将少得可怜的数据保存在 Cookie 中，Cookie 通过 key-value 保存数据，而且只能保存文本数据。而在 HTML5 中支持两种本地存储技术：DOM Storage 和 Web SQL Database，其中 DOM Storage 与 Cookie 类似，也用于存储 key-value 对象，但 DOM Storage 更易用。而 Web SQL Database 的功能更强大，提供了基本的关系数据库功能。

除此之外，HTML5 还提供了很多其他的超值功能，例如，音频/视频 API、地理位置 API、WebSocket API 和 Form API 等。通过使用这些技术，将大大丰富我们的 Web 应用程序。

## 17.2 HTML5 精彩效果演示

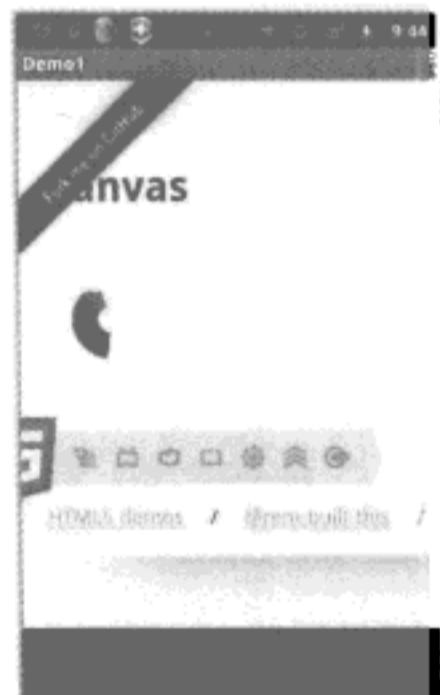
本节演示一些用 HTML5 实现的效果，看一看是不是非常酷！由于 HTML5 是新的 HTML 标准，浏览本节给出的页面时要尽量用最新的浏览器，如 FireFox 3.5、IE 9、最新基于 WebKit 引擎的浏览器。

### 效果 1：旋转的圆圈（图 17.1 和图 17.2）。

<http://HTML5demos.com/canvas>



▲ 图 17.1 在 FireFox 中浏览的效果



▲ 图 17.2 在 Nexus S 中浏览的效果

### 效果 2：颜色随着鼠标变化（图 17.3 和图 17.4）。

<http://HTML5demos.com/canvas-grad>



▲ 图 17.3 在 FireFox 中浏览的效果



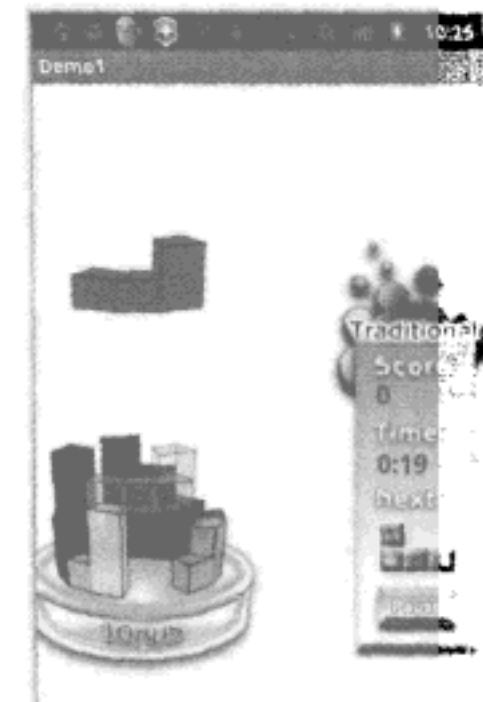
▲ 图 17.4 在 Nexus S 中浏览的效果

### 效果 3：3D 俄罗斯方块（图 17.5 和图 17.6）。

<http://www.benjoffe.com/code/games/torus/>



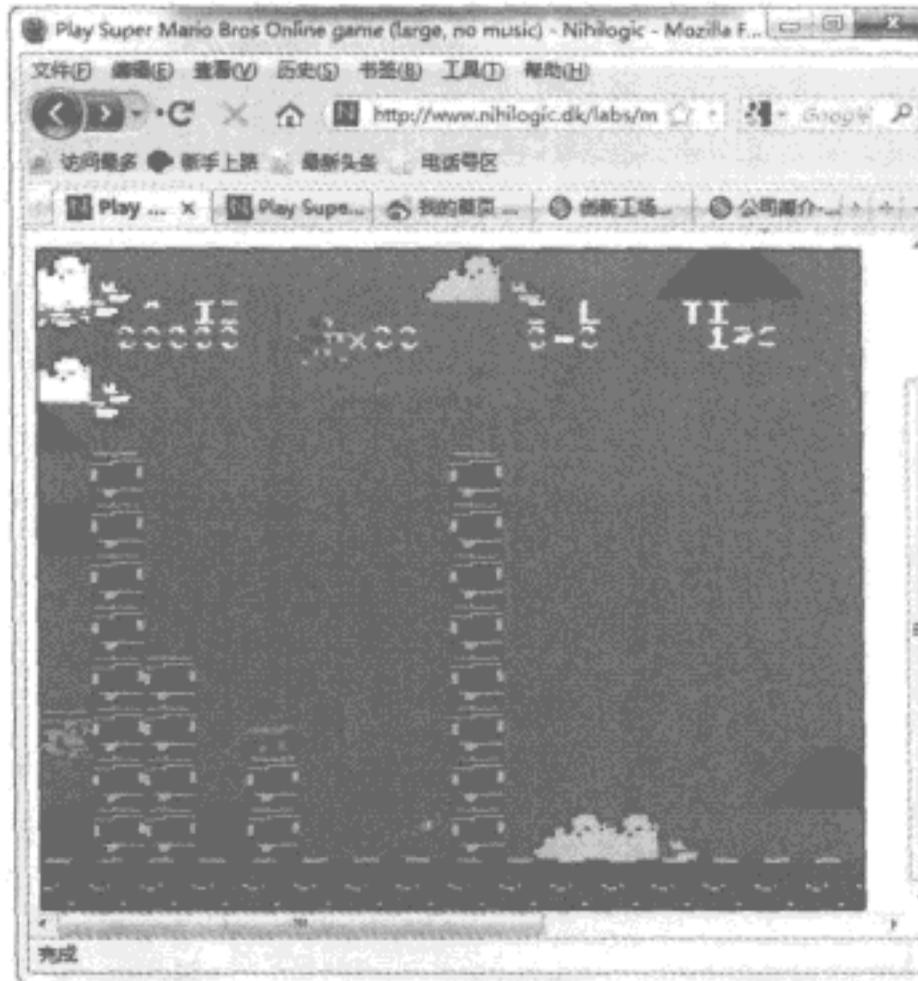
▲ 图 17.5 在 FireFox 中浏览的效果



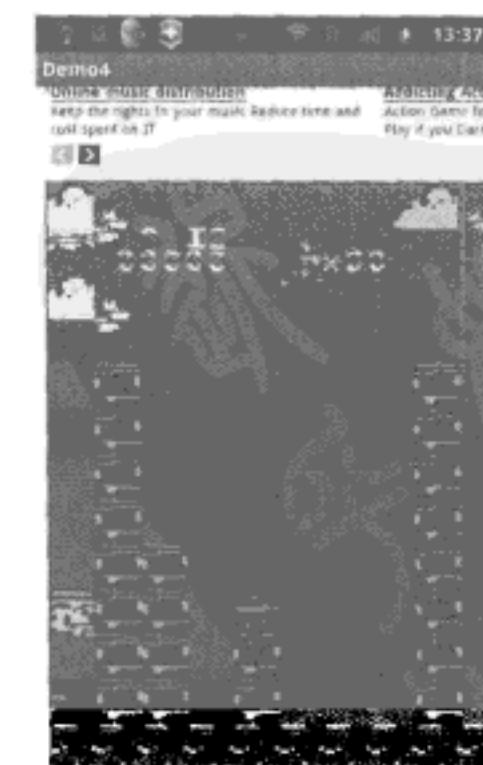
▲ 图 17.6 在 Nexus S 中浏览的效果

### 效果 4：超级玛丽（图 17.7 和图 17.8）。

[http://www.nihilogic.dk/labs/mario/mario\\_large\\_nomusic.htm](http://www.nihilogic.dk/labs/mario/mario_large_nomusic.htm)



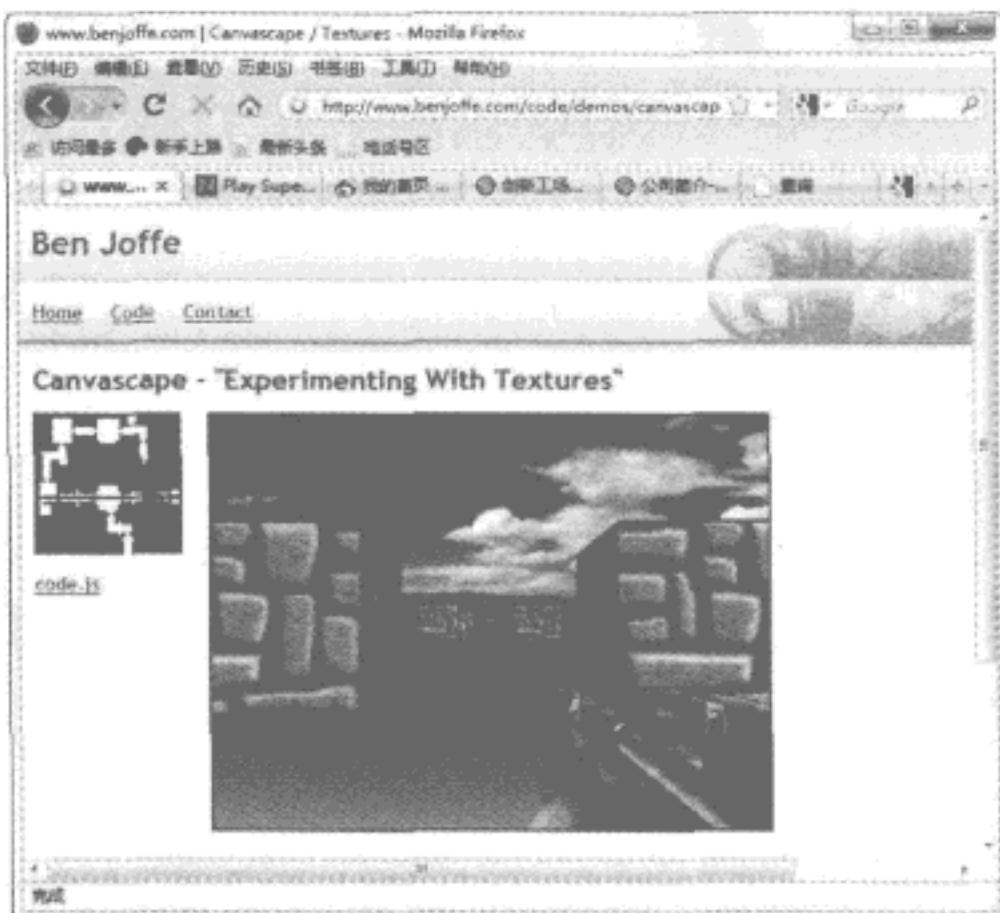
▲ 图 17.7 在 FireFox 中浏览的效果



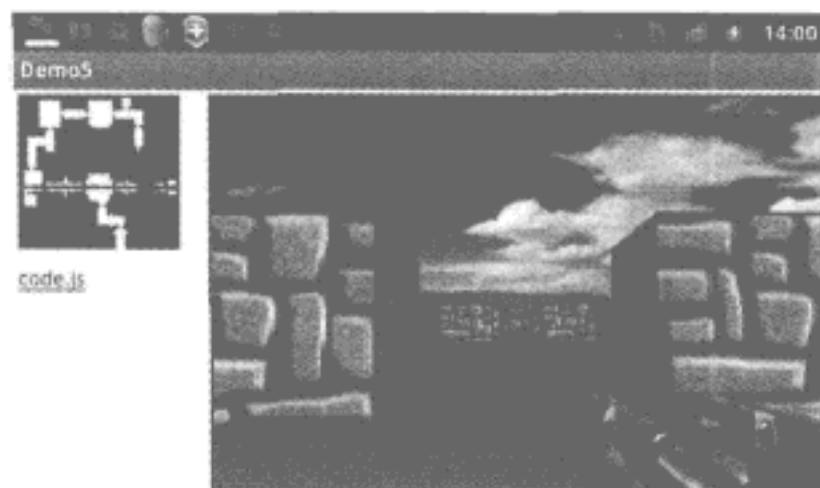
▲ 图 17.8 在 Nexus S 中浏览的效果

### 效果 5：射击游戏（图 17.9 和图 17.10）。

[http://www.benjoffe.com/code/demos/canvasape/textures](http://www.benjoffe.com/code/demos/canvascape/textures)



▲ 图 17.9 在 FireFox 中浏览的效果



▲ 图 17.10 在 Nexus S 中浏览的效果

## 17.3 HTML5 在 Android 中的应用

工程目录: src\ch17\ch17\_html5\_demo

虽然 HTML5 最初被使用在电脑的浏览器中,但最近也逐渐渗透到了移动领域。由于目前大多数智能手机的浏览器都支持 HTML5,甚至比 PC 浏览器的支持率更高。而且 HTML5 的功能强大、灵活、轻便、跨平台及更容易维护等优点,使 HTML5 在某种程度上完全可以取代智能设备本地化的应用程序。

在 Android 中使用 HTML5 可以有多种方法,最直接的方法就是在浏览器中输入 URL 来访问基于 HTML5 的页面,但这种方式无法在客户端进行更多地控制。例如,由于手机的分辨率不同,所以页面的大小也要随着手机分辨率的不同而变化。最简单的方法就是为每一个分辨率单独做一个页面,客户端根据当前手机分辨率的不同访问不同的页面。要实现这样的功能就需要使用本地化的程序,最简单的方法就是使用 WebView 控件来访问页面。首先获得当前手机的分辨率(宽度和高度),然后根据具体的分辨率访问合适的页面。下面的代码考虑了两个分辨率:320×480 和 480×800。

```
WebView webView = (WebView) findViewById(R.id.webview);
// 获得屏幕的宽度
int width = getWindowManager().getDefaultDisplay().getWidth();
// 获得屏幕的高度
int height = getWindowManager().getDefaultDisplay().getHeight();
webView.getSettings().setJavaScriptEnabled(true);
webView.setWebChromeClient(new WebChromeClient());
// 分辨率: 320*480
if (width == 320 && height == 480)
    webView.loadUrl("http://HTML5demos.com/canvas");
```

```
// 分辨率: 480*800
else if (width == 480 && height == 800)
    webView.loadUrl("http://www.benjoffe.com/code/demos/canvascape/textures");
```

当然，也可以利用 Android 资源来判断分辨率。例如，可以在 res 目录中建立两个子目录：values-hdpi 和 values-mdpi，分别保存高分辨率和中分辨率的字符串资源，并将适应不同分辨率的页面 URL 作为资源放到这些资源文件中，这样系统就会根据屏幕分辨率获得不同页面的 URL。

除此之外，还可以将 HTML5 页面保存在本地，如 SD 卡、assets 或 raw 目录中。程序中的帮助或不需要实时更新的 Web 应用可以采用这种方式。

### **多学一招：使用 JavaScript 判断手机屏幕的密度类型**

手机屏幕的密度分为低密度（240×320）、中密度（320×480）和高密度（480×800）。通过 window.devicePixelRatio 属性可以获得当前手机屏幕的密度类型。如果该属性值为 1.5，表示高密度；属性值为 1，表示中密度；属性值为 0.75，表示低密度。例如，下面的 JavaScript 代码运行在不同屏幕密度的手机中会显示不同的提示信息。

```
if (window.devicePixelRatio == 1.5)
{
    alert("高密度屏幕");
}
else if (window.devicePixelRatio == 1)
{
    alert("中密度屏幕");
}
else if (window.devicePixelRatio == 0.75)
{
    alert("低密度屏幕");
}
```

## **17.4 HTML5 的画布（Canvas）**

虽然 HTML5 支持多种新的技术，例如，Canvas、本地存储、WebSocket 等，但由于 HTML5 目前并没有大范围地普及，几乎没有完全支持 HTML5 的浏览器，大多数浏览器只支持 HTML5 的部分功能。不过经测试，在 HTML5 的众多新功能中，对 Canvas 的支持最广泛，几乎所有支持 HTML5 的浏览器都支持 Canvas。因此，本节着重介绍 Canvas。读者可在手机浏览器或 FireFox 3.x、IE 9 等浏览器中测试本节的例子。

### 17.4.1 Canvas 概述

HTML5 中的 Canvas 与 Android SDK 中的 Canvas 类似，也是用于绘图的区域。当我们在页面中放一个 Canvas 时，浏览器就会在页面上创建一个矩形的区域。如果不指定 Canvas 的尺寸，默认会创建宽度为 300 像素，高度为 150 像素的矩形绘图区域。

在 Web 页面中使用<canvas>标签来定义 Canvas，代码如下：

```
<canvas></canvas>
```

一旦在页面上添加一个 Canvas，就可以使用 JavaScript 按照我们的想法来控制 Canvas，例如，在 Canvas 上绘制图像、直线、曲线、文本等，甚至可以在 Canvas 上添加动画效果。

HTML5 Canvas API 与 Android SDK 提供的 Canvas API 非常类似，如果读者看过第 15 章的内容，应该会对这些 API 非常熟悉，上手 HTML5 Canvas API 也会很快。

使用 JavaScript 控制 Canvas 的第 1 步就是获得 Canvas 的 Context 对象，然后我们可以调用 Context 对象的相应方法来操作 Canvas，最后应用这些动作，使其真正生效，这个过程与数据库的事务类似。开始事务后，执行的任何操作都是临时的，只有在最后执行 commit 操作来提交事务，对数据库的修改才真正生效。

虽然 Canvas 功能强大，但并不是所有的时候都必须使用 Canvas。如果 HTML5 的其他组件能很好地完成任务，就没有必要使用 Canvas 了。例如，可以简单地使用<H1>、<H2>来设置不同的标题字体，就没有必要使用 Canvas 来绘制它们。

真正健壮的 Web 应用可以在浏览器不支持 HTML5 的某些特性时提醒用户，例如，下面的代码在浏览器不支持 Canvas 时会显示一行信息“浏览器不支持 HTML5 Canvas，您该换浏览器了”。

```
<canvas>
    浏览器不支持 HTML 5 Canvas，您该换浏览器了
</canvas>
```

#### 17.4.2 检测浏览器是否支持 Canvas

工程目录：src\ch17\ch17\_detect\_html5

在使用 Canvas 之前，最好先确认一下当前浏览器是否支持 Canvas。虽然可以像上一节那样直接在<canvas></canvas>标签中放置文本来提示浏览器是否支持 Canvas，但这样做并不灵活。我们可以使用 JavaScript 动态创建一个 Canvas 对象，并获得 Canvas 的 Context。如果成功，则浏览器支持 Canvas；否则，浏览器不支持 Canvas。下面的代码是使用 JavaScript 来测试当前浏览器是否支持 Canvas，不管是否支持，都会在浏览器中输出一行文本。

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html" ; charset="utf-8">
</head>
<body>
    <label id="text"></label>
    <script language="javascript">
        try
        {
            // 创建 Canvas 对象，并获得 Canvas 的 Context
            document.createElement("canvas").getContext("2d");
            document.getElementById("text").innerHTML = "当前浏览器支持 HTML5 Canvas.";
        }
        catch (e)
        {
            document.getElementById("text").innerHTML = "当前浏览器不支持 HTML5 Canvas.";
        }
    </script>
</body>
</html>
```

本小节的例子以及 17.4 节的其他例子都在 Nexus S 手机中进行了测试，读者可以将相应的 Web 页放在 PC 或其他可访问的空间上，并在手机浏览器中进行测试，以便了解自己手机的浏览器是否支持 Canvas。为了方便，这些例子都将 Web 页面（test.html）放在了 Android 工程的 assets 目录中，并通过下面的代码装载 Web 页面。

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.getSettings().setJavaScriptEnabled(true);
webView.setWebChromeClient(new WebChromeClient());
StringBuilder html = new StringBuilder();
try
{
    InputStream is = getResources().getAssets().open("test.html");
    InputStreamReader isr = new InputStreamReader(is, "utf-8");
    BufferedReader br = new BufferedReader(isr);
    String s = "";
    int count = 0;
    while ((s = br.readLine()) != null)
    {
        html.append(s + "\r\n");
    }
    br.close();
}
catch (Exception e)
{
}
// 装载 Web 页面
webView.loadDataWithBaseUrl(null, html.toString(), "text/html", "utf-8", null);
```

### 17.4.3 在 Web 页面中使用 Canvas

**工程目录:** src\ch17\ch17\_canvas

在 Web 页面中使用 Canvas 的第 1 步就是使用<canvas>标签定义一个 Canvas，一般需要使用 width 和 height 属性设置 Canvas 的宽度和高度，代码如下：

```
<canvas height="200" width="200"></canvas>
```

Canvas 默认是不带边框的，我们也可以使用 CSS 为 Canvas 加一个边框，代码如下：

```
<canvas id="test_canvas" style="border: 1px solid;" width="200" height="200">
</canvas>
```

在<canvas>标签中设置 id 属性是必要的，因为可以很容易使用 JavaScript 来获得并访问 Canvas 对象。下面的页面有两个 Canvas，第 1 个 Canvas 没有边框，在 Canvas 上绘制了一条直线；第 2 个 Canvas 有边框，在 Canvas 上绘制了两条直线。

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<label>没有边框的 Canvas</label>
<p />
<canvas id="one_line" height="50" width="50"></canvas>
<p />
<label>带边框的 Canvas</label>
<p />
```

```

<canvas id="two_line" style="border: 1px solid;" width="200" height="200"> </canvas>
<script>
    function draw()
    {
        // 开始在第 1 个 Canvas 中绘制一条直线
        var canvas = document.getElementById('one_line');
        var context = canvas.getContext('2d');
        context.beginPath();
        context.moveTo(1, 1);
        context.lineTo(40, 40);
        // 只有调用 stroke 方法，才会真正在 Canvas 上绘制图形
        context.stroke();
        // 开始在第 2 个 Canvas 中绘制两条直线
        canvas = document.getElementById('two_line');
        context = canvas.getContext('2d');
        context.beginPath();
        context.moveTo(70, 140);
        context.lineTo(140, 70);
        context.moveTo(70, 70);
        context.lineTo(140, 140);
        // 只有调用 stroke 方法，才会真正在 Canvas 上绘制图形
        context.stroke();
    }
    window.addEventListener("load", draw, true);
</script>

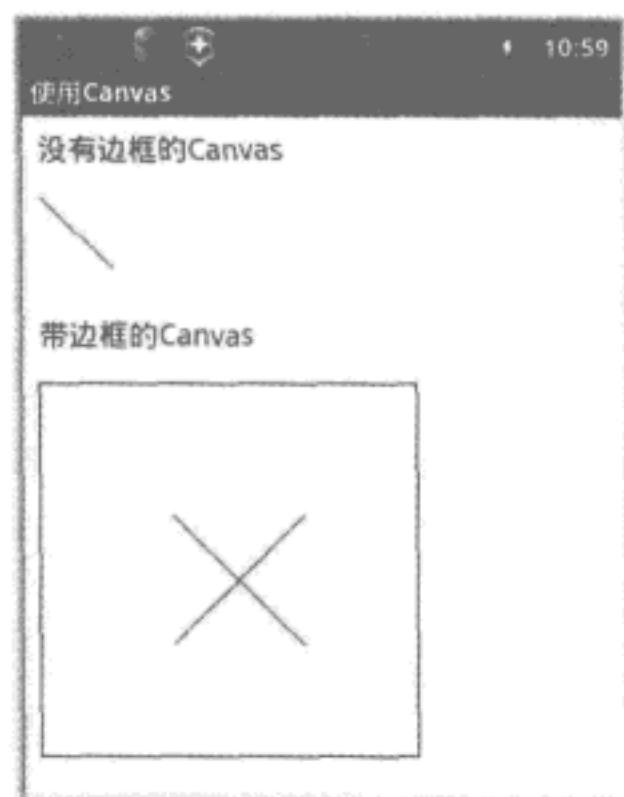
```

上面的代码首先获得了 Canvas 对象和绘制二维（2D）图像的 Context 对象；然后使用 `moveTo` 方法移动当前坐标，并使用 `lineTo` 方法从当前坐标开始绘制一条直线；最后调用 `stroke` 方法在 Canvas 上绘制图像（相当于提交事务）。

### ■ 注意

前面的代码多次调用了 `getContext` 方法，该方法的参数值都是“2d”，这表示获得了只用于绘制二维图形的 Context 对象。目前还不支持 3D，可能在后续的版本中支持绘制 3D 图形的 Context。

运行本例，会看到如图 17.11 所示的效果。



▲ 图 17.11 在 Web 页面中使用 Canvas

#### 17.4.4 使用路径 ( Path )

**工程目录:** src\ch17\ch17\_canvas\_path

虽然我们已经接触到很多在 Canvas 上画线的程序，但这些都较简单。本节将接触到更复杂的绘图技术：Path。在 HTML5 Canvas API 中 Path 可以是任何形状的图形，简单的直线就是一个路径。

使用 Path，首先应调用 beginPath 方法开始路径，然后可以绘制各种图形，甚至进行更复杂的操作，如绘制曲线，甚至是子路径（subpath）。

一旦开始了路径，我们就可以使用 Context 中的各种路径方法绘制图形了，现在已经接触到了如下两个简单的路径方法。

- `moveTo(x, y)`: 移动当前的位置到新的位置，并不绘制任何图形。
- `lineTo(x, y)`: 移动当前的位置到新的位置，并在移动的两点之间绘制一条直线。

实际上，调用上面两个方法只是模拟绘制，并不会真正地将直线绘制在 Canvas 上。只有在最后调用 `stroke` 方法，才会根据前面的模拟绘制记录在 Canvas 中绘制图形。

除了 `stroke` 方法外，还可以调用 `closePath` 方法来关闭路径。在关闭路径之前，会将所有的路径绘制在 Canvas 上，这相当于 Web 应用中的服务端缓冲区。当缓冲区未写满时，除了调用 `flush` 方法将缓冲区的内容送到客户端外，在 `OutputStream` 关闭之前，系统会将缓冲区未送出的数据送到客户端。下面的代码绘制了更复杂的图形，最后使用了 `closePath` 方法关闭路径。

```
<canvas id="tree" width="200" height="200"> </canvas>
<script>
    function createPath(context)
    {
        context.beginPath();
        context.moveTo(-25, -50);
        context.lineTo(-10, -80);
        context.lineTo(-20, -80);
        context.lineTo(-5, -110);
        context.lineTo(-15, -110);

        context.lineTo(0, -140);
        context.lineTo(15, -110);
        context.lineTo(5, -110);
        context.lineTo(20, -80);
        context.lineTo(10, -80);
        context.lineTo(25, -50);
        // 关闭路径
        context.closePath();
    }
    function drawTree()
    {
        var canvas = document.getElementById('tree');
        var context = canvas.getContext('2d');
        // 保存当前绘制状态
        context.save();
        // 将坐标系向右下方移动
        context.translate(60, 140);
        createPath(context);
    }
</script>
```

```
context.stroke();
// 恢复最初的坐标系
context.restore();
}
window.addEventListener("load", drawTree, true);
</script>
```

运行本例，显示效果如图 17.12 所示。



▲ 图 17.12 路径

#### 17.4.5 设置线条风格

工程目录: src\ch17\ch17\_stroke

HTML5 Canvas API 提供了多个属性用于设置线条的风格。例如，`lineWidth` 用于设置线条的粗细，`lineJoin` 用于设置线条连接处的类型。如果该属性值为“round”，线条连接处为圆角。`strokeStyle` 用于设置线条的颜色。下面的代码使用这 3 个属性改变了上一小节绘制的“小树”的风格。

```
// 设置绘制线条为 6 个像素粗
context.lineWidth = 6;
// 设置线条连接处为圆角
context.lineJoin = 'round';
// 设置线条的颜色
context.strokeStyle = '#669900';
```

运行本例，会看到如图 17.13 所示的效果。



▲ 图 17.13 改变绘制图形的风格

#### 17.4.6 设置填充类型

工程目录: src\ch17\ch17\_fill

除了设置绘制线条的风格，还可以使用 `fillStyle` 设置填充颜色，代码如下：

```
context.fillStyle = '#996600';
context.fill();
```

上面的代码必须在绘制完所有线条后执行（调用 `stroke` 方法之前）。运行本例，显示效果如图 17.14 所示。



▲ 图 17.14 填充颜色

#### 17.4.7 填充矩形区域

工程目录: `src\ch17\ch17_fill_rect`

使用 `fillRect` 可以绘制一个实心的矩形，通过 `fillStyle` 可以设置矩形的填充颜色，代码如下：

```
context.fillStyle = '#996600';
context.fillRect(-5, -50, 10, 50);
```

运行本例，会看到如图 17.15 所示的效果。



▲ 图 17.15 填充矩形区域

#### 17.4.8 使用渐变色 (Gradient)

工程目录: `src\ch17\ch17_gradient`

是不是看着我们以前画的那棵小树很单调，本节就用更高级的方法来绘制它。绘制这棵树需要利用本节要介绍的渐变色。渐变色允许我们使用渐变算法来绘制图形，这些算法可以利用不同的颜色、线条类型、填充类型等技术来渲染图形。

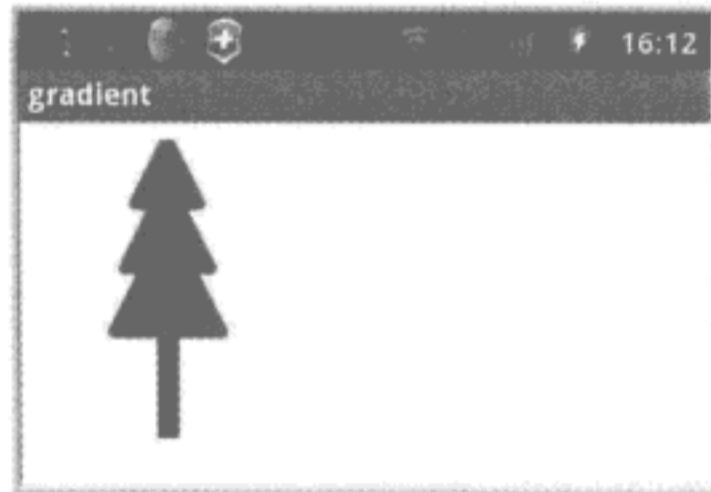
我们可按如下 3 步来创建一个 `Gradient`。

- (1) 创建 `Gradient` 对象。
- (2) 设置 `Gradient` 对象的颜色变化的颜色带。
- (3) 设置 `Gradient` 对象的 `fillStyle` 或 `strokeStyle`。

下面的代码设置了 3 个颜色带，使小树的树干颜色水平方向渐变。

```
// 创建一个 Gradient 对象
var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);
// 设置树干开始的颜色
trunkGradient.addColorStop(0, '#663300');
// 设置树干中间的颜色
trunkGradient.addColorStop(0.4, '#996600');
// 设置树干右边缘的颜色
trunkGradient.addColorStop(1, '#552200');
// 使用 Gradient 重新绘制树干
context.fillStyle = trunkGradient;
context.fillRect(-5, -50, 10, 50);
```

运行本例，显示效果如图 17.16 所示。



▲ 图 17.16 渐变色

#### 17.4.9 拉伸画布对象

工程目录: src\ch17\ch17\_scale

在绘制图形之前调用 `scale` 方法可以按比例放大或缩小绘制的图形。例如，下面的代码将前面绘制的小树放大一倍。

```
<canvas id="tree" width="200" height="300"> </canvas>
<script>
    function createPath(context)
    {
        context.beginPath();
        context.lineWidth = 6;
        context.lineJoin = 'round';
        context.strokeStyle = '#669900';

        context.moveTo(-25, -50);
        context.lineTo(-10, -80);
        context.lineTo(-20, -80);
        context.lineTo(-5, -110);
        context.lineTo(-15, -110);

        context.lineTo(0, -140);
        context.lineTo(15, -110);
        context.lineTo(5, -110);
        context.lineTo(20, -80);
```

```

        context.lineTo(10, -80);
        context.lineTo(25, -50);
        context.fillStyle = '#996600';
        context.fill();

        var trunkGradient = context.createLinearGradient(-5, -50, 5, -50);
        trunkGradient.addColorStop(0, '#663300');
        trunkGradient.addColorStop(0.4, '#996600');
        trunkGradient.addColorStop(1, '#552200');
        context.fillStyle = trunkGradient;
        context.fillRect(-5, -50, 10, 50);

        context.closePath();
    }
    function drawTree()
    {
        var canvas = document.getElementById('tree');
        var context = canvas.getContext('2d');
        context.save();
        context.translate(120, 300);
        // 将绘制的图形放大一倍
        context.scale(2, 2);
        createPath(context);
        context.stroke();
        context.restore();
    }
    window.addEventListener("load", drawTree, true);
</script>

```

`scale` 方法有两个参数，分别表示图形在 `x` 轴（水平）和 `y` 轴（垂直）方向放大或缩小的比例。运行本例，显示效果如图 17.17 所示。



▲ 图 17.17 将图形放大一倍

#### 17.4.10 在 Canvas 上绘制文本

工程目录: src\ch17\ch17\_canvas\_text

使用 `fillText` 方法可以在 Canvas 上绘制文本。`fillText` 方法的定义如下：

```
| fillText (text, x, y, maxwidth)
```

其中 `text` 表示要绘制的文本，`x`、`y` 表示文本左上角的坐标，`Maxwidth` 表示绘制文本所占的宽度。下面的代码在小树下面绘制了“圣诞树”3个字。

```
// 设置字号、字体  
context.font = "30px impact";  
// 设置文字颜色  
context.fillStyle = '#FF0000';  
// 设置文字在 maxwidth 参数指定区域的对齐方式  
context.textAlign = 'center';  
// 绘制文本  
context.fillText('圣诞树', 10, 30, 400);
```

运行本例，显示的效果如图 17.18 所示。



▲ 图 17.18 绘制文字

#### 17.4.11 使用阴影

工程目录: `src\ch17\ch17_shadow`

使用 `shadowOffsetX`、`shadowOffsetY`、`shadowColor`、`shadowBlur` 属性可以绘制图形的阴影。例如，下面的代码为小树绘制了一个阴影。

```
// 设置阴影的颜色为黑色以及 20% 的透明  
context.shadowColor = 'rgba(0, 0, 0, 0.2)';  
// 设置阴影在水平方向向右偏移 15 个像素  
context.shadowOffsetX = 15;  
// 设置阴影在垂直方向向下偏移 10 个像素  
context.shadowOffsetY = -10;  
// 设置阴影模糊度  
context.shadowBlur = 2;
```

运行本例，显示的效果如图 17.19 所示。



▲ 图 17.19 绘制阴影

## 17.5 调试 JavaScript

工程目录: src\ch17\ch17\_debug

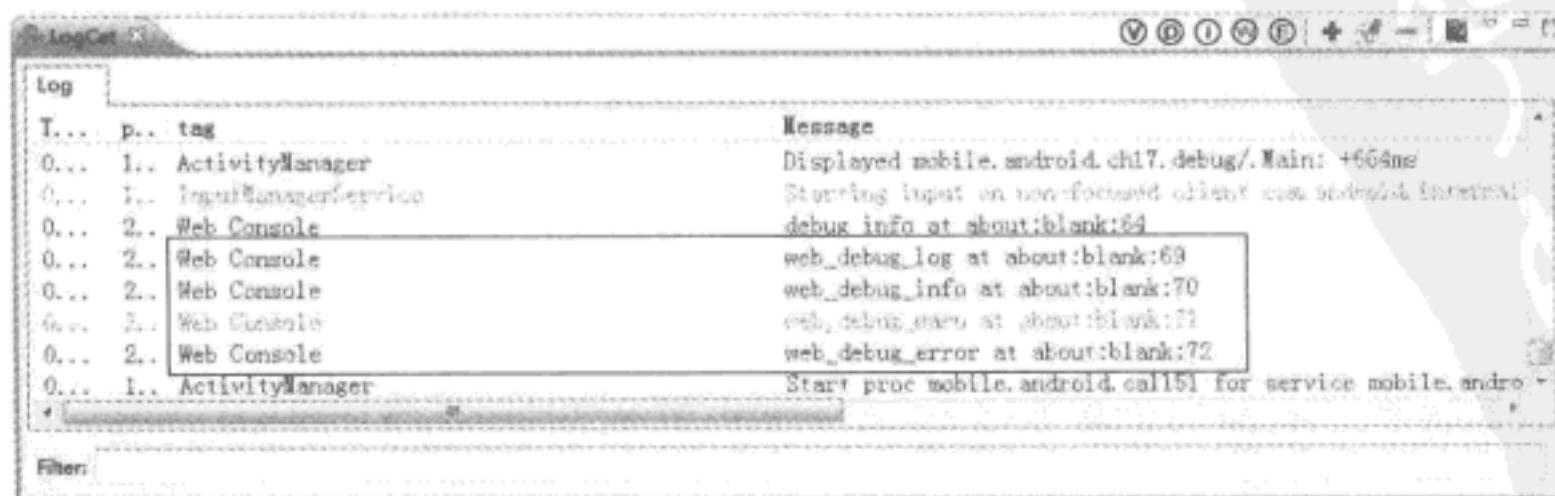
通常调试 JavaScript 都是使用 alert 方法弹出一个对话框来显示调试信息,但在 Android 中可以使用如下 4 个方法在 LogCat 视图中输出调试信息。

- console.log(String)。
- console.info(String)。
- console.warn(String)。
- console.error(String)。

下面的代码分别使用这 4 个方法在 LogCat 视图中输出了 4 行调试信息。

```
console.log("web_debug_log");
console.info("web_debug_info");
console.warn("web_debug_warn");
console.error("web_debug_error");
```

输出效果如图 17.20 所示。



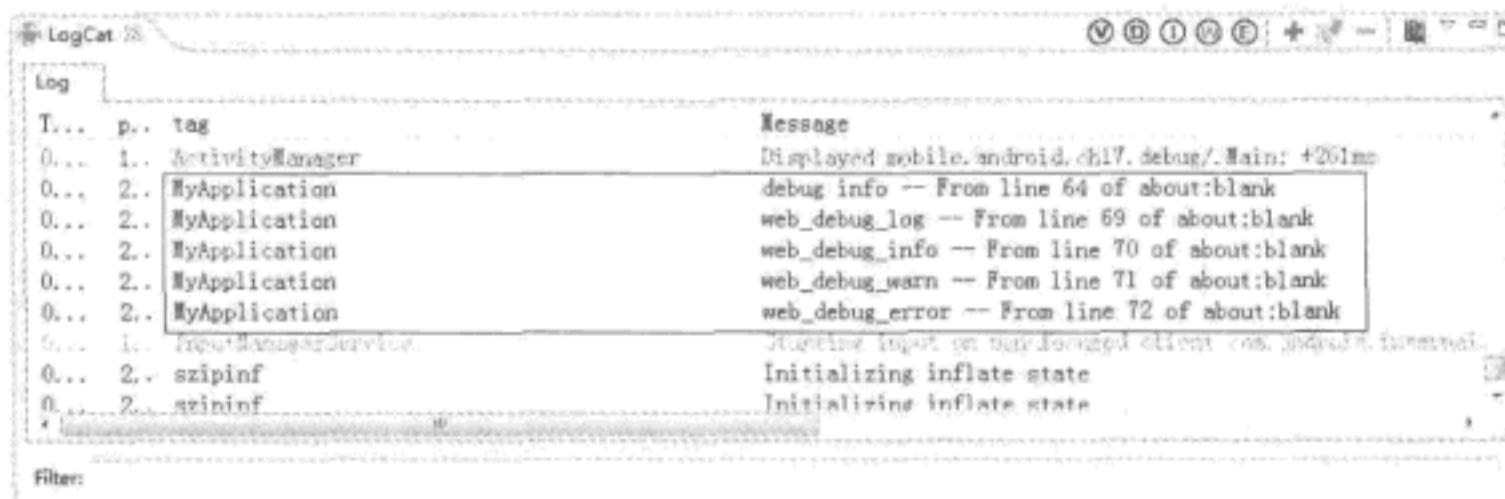
▲ 图 17.20 使用 JavaScript 输出调试信息

如果使用 WebView 控件执行 JavaScript 代码，可以使用下面的代码改变调试信息的输出格式。

```
webView.setWebChromeClient(new WebChromeClient()
{
    @Override
    public boolean onConsoleMessage(ConsoleMessage consoleMessage)
    {
        // 新的调试信息输出格式
        Log.d("MyApplication", consoleMessage.message()
            + " -- From line " + consoleMessage.lineNumber()
            + " of " + consoleMessage.sourceId());

        return true;
    }
});
```

再次运行本例，会看到如图 17.21 所示的输出信息。

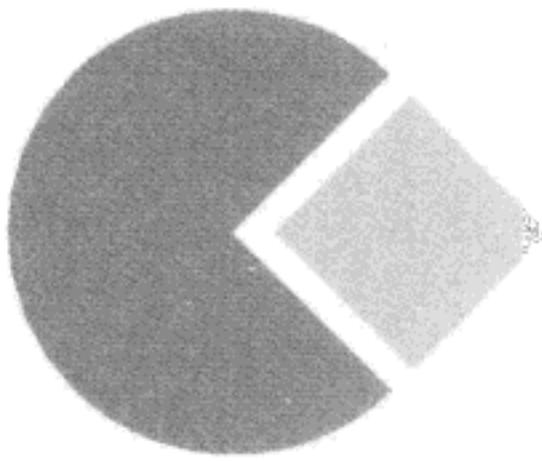


▲ 图 17.21 改变格式后的输出的调试信息

**注意** onConsoleMessage 方法一般返回 true，如果该方法返回 false，新格式的调试信息和默认格式的调试信息会同时输出。也就是说，每一行调试信息会变成两条，共 8 行调试信息。

## 17.6 小结

本章介绍了 HTML5 以及在 Android 中的应用。HTML5 是最近几年才刚刚兴起的 Web 技术。利用 HTML5 可以实现与 Flash、SilverLight 类似的效果。由于目前几乎没有浏览器支持 HTML5 的全部特性，因此，本章只介绍了支持比较广泛的 HTML5 Canvas API，其他的 HTML5 API 读者可以查看 HTML5 规范。由于目前手机平台众多，而各个平台的应用程序又不兼容，这就造成了需要同时开发多个平台版本才能满足众多手机用户的需求。而 HTML5 的跨平台特性正好满足了可以同时运行于多个手机平台的需求，因此，HTML5 在未来的移动领域有着非常广阔的应用前景。



## 第 18 章 输入法开发

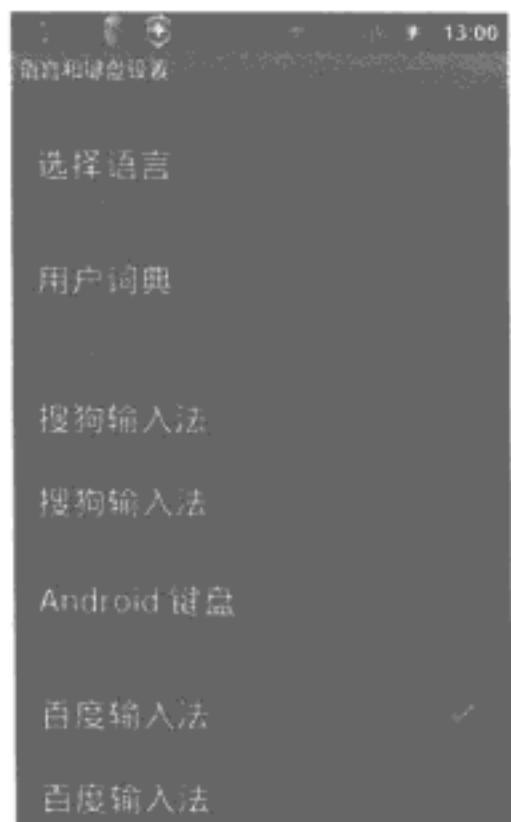
输入法是各类操作系统必备的程序。虽然输入英文可以只使用键盘，但各个国家的文字以及很多专有的符号不同，因此就需要专门的程序来输入这些文字和符号，这就是输入法。

Android 系统中内置了标准的输入法，但这远不能满足我们的需要。目前在 Android Market 上已经有很多第三方的输入法程序可供我们选择，不过 Android 拥有的强大扩展性允许任何人定制更专业的输入法，以满足特殊的需求。为了使对输入法感兴趣的读者可以实现自己的输入法，本章详细介绍了 Android 输入法的实现过程。

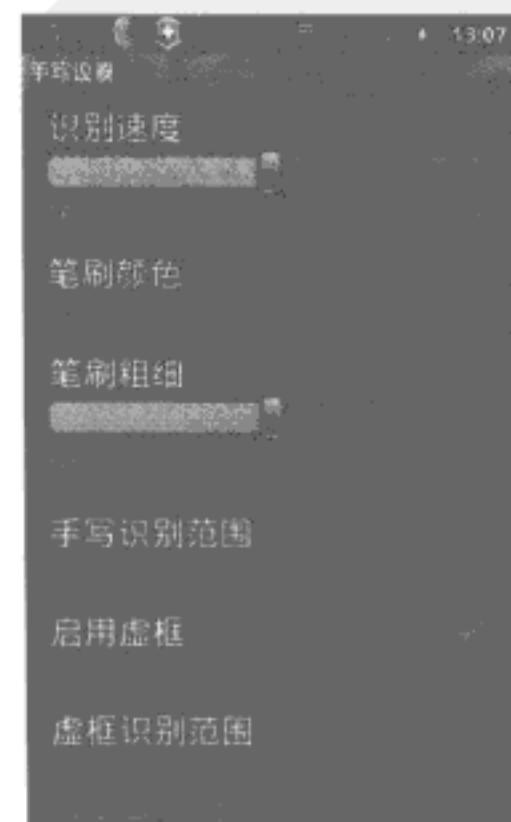
### 18.1 Android 输入法简介

从 Android 1.5 开始，Android SDK 提供了一个 IMF（Input Method Framework，输入法框架），用于开发类似软键盘的输入法程序。IMF 被设计成支持不同的输入法形式，包括软键盘、手写识别、硬键盘映射等。

除了系统自带的标准输入法，后来安装的输入法都需要在 Android 设置中的“语言和键盘”设置界面开启输入法，如果该输入法带有设置页面，可以对输入法进行设置，以符合我们的使用要求。图 18.1 是 Nexus S 手机上的“语言和键盘”设置界面。在这部手机上安装了“搜狗输入法”和“百度输入法”，这两个输入法各自带有一个设置项，图 18.2 是“百度输入法”中用于设置手写识别属性的界面。

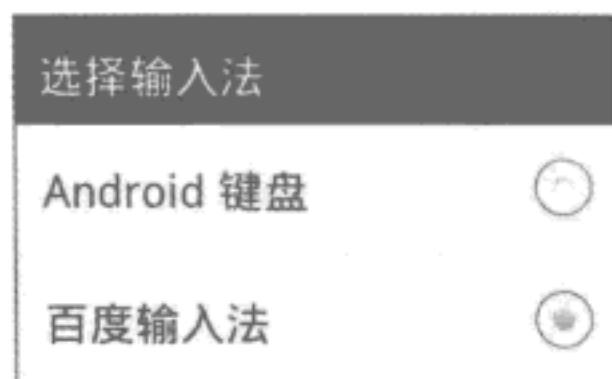


▲ 图 18.1 “语言和键盘”设置界面



▲ 图 18.2 “百度输入法”手写识别设置界面

开启输入法后，随便找一个带 EditText 控件的程序，然后长按 EditText 控件，会弹出一个菜单，其中有一个“输入法”菜单项。单击这个菜单项，会弹出如图 18.3 所示的“选择输入法”菜单。除了 Android 自带的“Android 键盘”外，其他菜单项都是在如图 18.1 所示的设置界面开启的输入法。由于“搜狗输入法”尚未开启，所以在如图 18.3 所示的菜单中只有“百度输入法”。

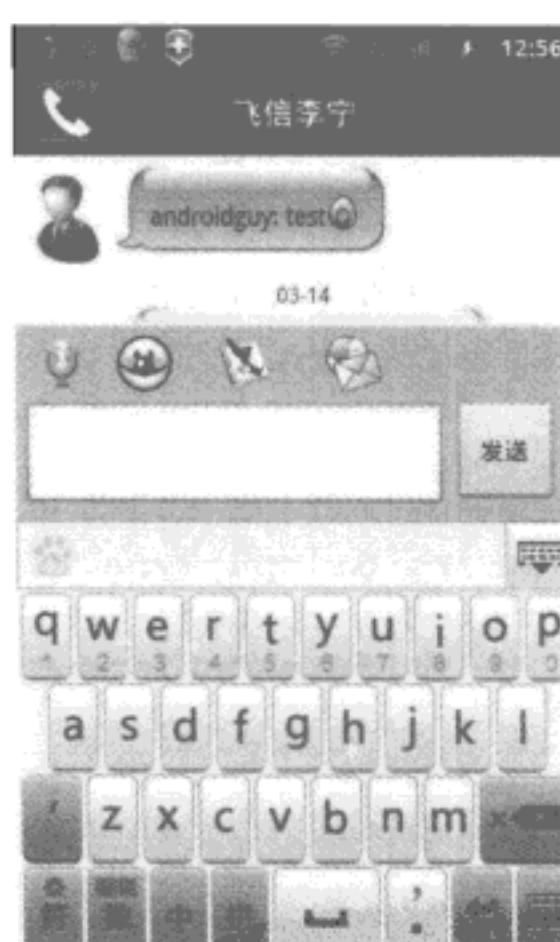


▲ 图 18.3 选择输入法

选择输入后，触摸 EditText 控件，就会弹出相应的输入法界面。图 18.4 是 Android 自带的标准输入法，图 18.5 是“百度输入法”。



▲ 图 18.4 Android 自带的输入法



▲ 图 18.5 百度输入法

## 18.2 控制输入法

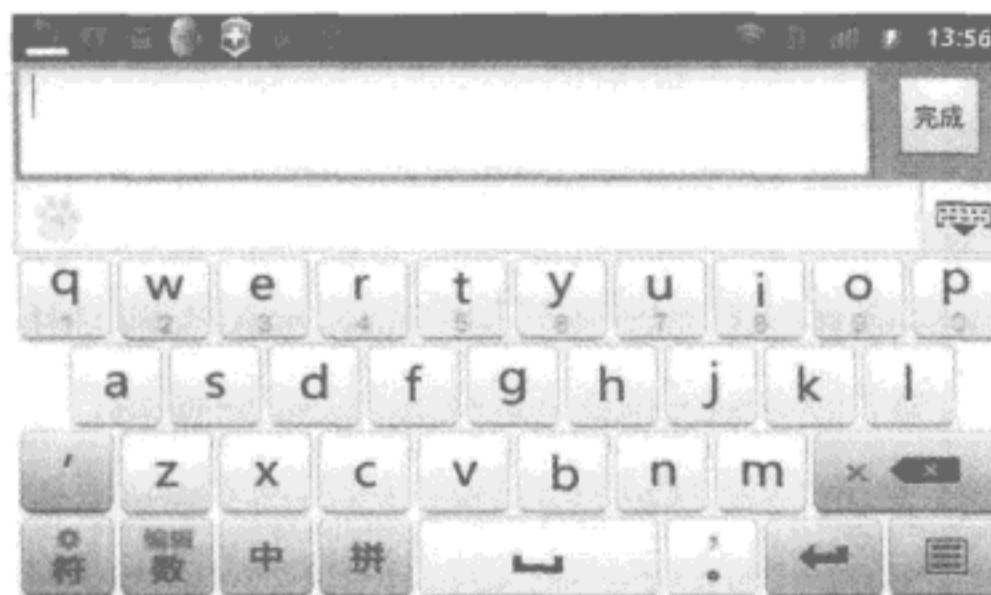
当 EditText 处于焦点状态，并显示输入法界面时，系统会自动调整界面布局，以便尽可能让正在处于编辑状态的 EditText 控件不被输入法界面覆盖。如果手机横屏时，系统会将输入法调整成全屏。也就是说，只保留正在输入的 EditText 控件，其他的所有控件都被隐藏了。图 18.6 是“百度输入法”横屏的效果。

通过设置<EditText>标签的 android:inputType 属性可以按照要求显示不同的输入法界面。 android:inputType 可以设置如下 3 种类型的值。

- 按类划分的字符。例如，普通文本 (plain text)、数字 (number)、电话 (phone)、日期时

间 (datetime)。对于每一个类别，输入法界面都只显示与该类别相关的字符按键，例如，数字键盘只显示 0 至 9，共 10 个数字按键以及一些常用的标点符号。

- 额外增加的字符。在标准的输入法界面增加一些特殊的字符按键。例如，`android:inputType` 属性值为 `textEmailAddress`，在弹出的虚拟键盘中就会多出一个“@”字符键，用于输入 E-mail 中间的“@”字符。
- 附加的输入规则。例如，大写 (CapSentences)、文本自动校正 (textAutoCorrect)、文本多行输入 (textMultiLine) 等。



▲ 图 18.6 “百度输入法”横屏的效果

下面是一个设置了 `android:inputType` 和 `android:imeOptions` 属性的 `EditText` 控件。

```
<EditText android:id="@+id/editInput"
    android:layout_width="0dip"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:inputType="textShortMessage|textAutoCorrect|textCapSentences|
    textMultiLine"
    android:imeOptions="actionSend|flagNoEnterAction"
    android:maxLines="4"
    android:maxLength="2000"
    android:hint="@string/compose_hint"/>
```

除了可以控制软键盘显示的字符按钮外，还可以控制当前窗口相对软键盘的行为。我们通常可以在 `AndroidManifest.xml` 文件中设置 `<activity>` 标签的 `android:windowSoftInputMode` 属性来控制窗口的这些行为。例如，下面的代码设置了在切换到 `EditContactActivity` 时立刻显示当前输入法的软键盘（不管是否有获得焦点的 `EditText` 控件），而且并不调整界面布局。

```
<activity name="EditContactActivity"
    android:windowSoftInputMode="stateVisible| adjustPan ">
    ...
</activity>
```

## 18.3 输入法实战

工程目录：src\ch18\ch18\_simple\_inputmethod

本节将带领读者实现一个简单的输入法程序。这个输入法很简单，只允许输入 4 个字符串（通

过 4 个按钮输入)。单击“Hide”按钮隐藏输入法。本节会逐步完成这个输入法程序，例如，会加入组合文字、设置等功能。

### 18.3.1 实现输入法的步骤

输入法程序的核心是一个服务类，这个类必须继承自 `InputMethodService`。下面先来看看实现一个基本的输入法程序的步骤。

- (1) 建立一个继承自 `android.inputmethodservice.InputMethodService` 的类，称为输入法的服务类。
- (2) 在 `AndroidManifest.xml` 文件中配置这个服务类。
- (3) 编写一个用于显示软键盘的布局文件。
- (4) 覆盖 `InputMethodService` 类的 `onCreateInputView` 方法。
- (5) `onCreateInputView` 方法需要返回与第 3 步建立的布局文件对应的 `View` 对象。在返回之前，一般需要设置相应控件的事件，如软键盘按钮单击事件。
- (6) 在输入法服务类或其他类中编写响应软键盘中按键事件的代码，如按钮单击事件、物理键盘事件等。

### 18.3.2 编写输入法程序

这一小节我们来实现一个简单的输入法程序。接着上一节介绍的步骤，首先来建立一个 `AndroidInputMethodService` 类，该类继承自 `InputMethodService`，然后在 `AndroidManifest.xml` 文件中配置 `AndroidInputMethodService`，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobile.android.ch18.simple.inputmethod" android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="9" />
    <application android:icon="@drawable/icon" android:label="简单输入法">
        <service android:name="mobile.android.ch18.simple.inputmethod.AndroidInput-
            MethodService"
            android:permission="android.permission.BIND_INPUT_METHOD">
            <intent-filter>
                <action android:name="android.view.InputMethod" />
            </intent-filter>
            <meta-data android:name="android.view.im" android:resource="@xml/method" />
        </service>
    </application>
</manifest>
```

配置输入法服务时必须设置“`android.permission.BIND_INPUT_METHOD`”权限，并且在 `<intent-filter>` 标签中添加一个“`android.view.InputMethod`”动作。

在 `<service>` 标签中还加了一个 `<meta-data>` 标签，用于配置输入法，也就是在“语言与键盘”设置界面可以看到我们编写的输入法，其中 `android:resource` 属性指定了一个输入法资源 ID。这个资源文件 (`method.xml`) 在 `res\Xml` 目录中，代码如下：

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

```
<input-method xmlns:android="http://schemas.android.com/apk/res/android" />
```

<input-method>标签还有几个属性可以定制输入法，这些内容将在后面详细介绍。

下面进行第 3 步，编写一个布局文件。这个布局文件实际上就是软键盘的布局。在这个布局中有 5 个水平排列的按钮，其中前 4 个用于输入 4 个字符串（就是<Button>标签的 android:text 属性值），最后一个按钮用于隐藏软键盘。

### keyboard.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Android" />
    <Button android:id="@+id/button2" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="安卓"
        android:layout_marginLeft="6dp" />
    <Button android:id="@+id/button3" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Google"
        android:layout_marginLeft="6dp" />
    <Button android:id="@+id/button4" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="谷歌"
        android:layout_marginLeft="6dp" />
    <Button android:id="@+id/btnHide" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Hide"
        android:layout_marginLeft="6dp" />
</LinearLayout>
```

接下来的步骤都是修改 AndroidInputMethodService 类的代码，首先覆盖 onCreateInputView 方法，然后在 onCreateInputView 方法中装载 keyboard.xml 布局文件，并设置按钮的单击事件，最后返回软键盘的 View 对象。

```
public View onCreateInputView()
{
    // 装载 keyboard.xml 文件
    View view = getLayoutInflater().inflate(R.layout.keyboard, null);
    // 设置布局中 5 个按钮的单击事件
    view.findViewById(R.id.button1).setOnClickListener(this);
    view.findViewById(R.id.button2).setOnClickListener(this);
    view.findViewById(R.id.button3).setOnClickListener(this);
    view.findViewById(R.id.button4).setOnClickListener(this);
    view.findViewById(R.id.btnHide).setOnClickListener(this);
    // 返回 View 对象
    return view;
}
```



输入法界面（软键盘）并不需要我们自己建立 Activity，这个 Activity 是由系统提供的，而我们只需要提供在 Activity 上显示的 View 对象即可，也就是 onCreateInputView 方法的返回值。

最后一步需要处理按钮的单击动作，AndroidInputMethodService 类需要实现 OnClickListener 接口。OnClickListener.onClick 方法的代码如下：

```
public void onClick(View view)
{
    if (view.getId() == R.id.btnExit)
    {
        // 隐藏软键盘
        hideWindow();
    }
    else
    {
        Button button = (Button) view;
        // 获得 InputConnection 对象
        InputConnection inputConnection = getCurrentInputConnection();
        // 向当前获得焦点的 EditText 控件输出文本
        // commitText 方法第 2 个参数值为 1，表示在当前位置插入文本
        inputConnection.commitText(button.getText(), 1);
    }
}
```

运行本例后，并不会显示任何界面。现在进入手机设置的“语言和键盘”设置界面，会在最下面看到“简单输入法”的列表项，选中这个列表项，会弹出如图 18.7 所示的对话框，单击“确定”按钮，会将“简单输入法”列表项选中，然后关闭当前窗口即可。



▲ 图 18.7 开启“简单输入法”

现在随便找个带 EditText 控件的程序，长按 EditText 控件，在弹出的菜单中选择“输入法”菜单项，会弹出一个“选择输入法”菜单，如图 18.8 所示。单击“简单输入法”菜单项切换到我们刚编写的输入法，这时当 EditText 控件处于焦点状态时，就会在屏幕的下方显示 5 个按钮，如图 18.9 所示。

单击如图 18.9 所示界面下方左侧的 4 个按钮，会将按钮上的文字输入到 EditText 控件中。



▲ 图 18.8 选择“简单输入法”



▲ 图 18.9 “简单输入法”的软键盘

### 18.3.3 输入法服务的生命周期

输入法服务与普通的服务一样，也有一个生命周期。上一小节使用的 `onCreateInputView` 方法就是生命周期中的一个方法。除此之外，还有一些其他的生命周期方法。图 18.10 是输入法服务生命周期方法的调用次序。



▲ 图 18.10 输入法服务的生命周期

在选择输入法时，`onCreate`方法只调用一次，然后在输入控件处于焦点状态时，`onCreateInputView`和`onCreateCandidatesView`会各调用一次，然后会调用`onStartInputView`方法，这时会进入生命周期的一个循环。如果当前窗口有多个输入控件，焦点从弹出软键盘的输入控件切换到其他的输入控件时，系统会首先调用`onFinishInput`方法，然后会再次调用`onStartInputView`方法在新的输入控件获得焦点后显示软键盘。如果这时选择其他的输入法，系统就会调用`onDestroy`方法结束当前的输入法。因此，可以在`onDestroy`方法中加入释放输入法所使用的资源的代码。

### 18.3.4 预输入文本

通过`InputConnection.setComposingText`方法可以预输入文本。当输入一个拼音时，在输入控件中显示正在输入的拼音。如果在输入的过程中想输入英文，而不是拼音，就可以使用这种预输入的方法。例如，我们输入一个“Shang”，可以在弹出的汉字列表中选择“上”，当然，也可以直接将“Shang”输入到控件中。

预输入文本一般会在输入的文本下方显示下划线，直到将其替换成正式输入的文本，否则下划线一直存在。现在来修改18.3.2节例子中的`onClick`方法，代码如下：

```
public void onClick(View view)
{
    if (view.getId() == R.id.btnClose)
    {
        hideWindow();
    }
    else
    {
        Button button = (Button) view;
        InputConnection inputConnection = getCurrentInputConnection();

        if (button.getId() == R.id.button1)
        {
            // 设置预输入文本，setComposingText方法的第2个参数值为1，表示在当前位置插入预输入文本
            inputConnection.setComposingText(button.getText(), 1);
        }
        else
        {
            inputConnection.commitText(button.getText(), 1);
        }
    }
}
```

再次运行本例，当单击“Android”按钮时，会在`EditText`控件输入一个预输入文本（文字下方有下划线），如果这时再使用`commitText`方法输入文本，会将预输入文本（也就是“Android”）替换成用`commitText`方法输入的文本。因此，当单击其他三个按钮时，并不会在当前位置插入文本，而是将“Android”替换了相应文本。如果单击“Hide”按钮隐藏软键盘，就会将“Android”真正输入到`EditText`控件中。

### 18.3.5 输入法设置

在定义输入法服务时指定了一个`method.xml`文件，通过该文件中的`<input-method>`标签的

android:settingsActivity 属性可以指定输入法设置窗口。

```
<?xml version="1.0" encoding="utf-8"?>
<input-method xmlns:android="http://schemas.android.com/apk/res/android"
    android:settingsActivity="mobile.android.ch18.simple.inputmethod.InputMethodSetting" />
```

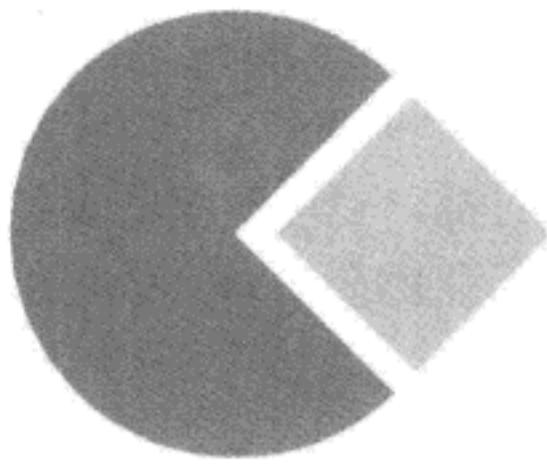
使用上面的代码替换 18.3.2 小节例子中的 method.xml 文件的内容，并运行程序。在“语言和键盘”设置窗口中除了带复选框的“简单输入法”列表项，还多了一个“简单输入法”列表项。如图 18.11 所示。单击该列表项，会显示 android:settingsActivity 属性指定的 Activity。



▲ 图 18.11 “简单输入法”设置

## 18.4 小结

本章介绍了开发 Android 输入法的基本方法。如果想开发一个完整的输入法还有很多工作要做，InputMethodService 类中的很多生命周期方法可能都用得上。在本书的第四部分还实现了一个较完整的输入法，读者可以参考这个程序来实现更复杂的输入法。



## 第 19 章 Android OpenGL ES 开发基础

说起游戏，几乎所有的人都不会陌生。从最初的单机 2D 游戏，到现在的网络 3D 游戏，无论从显示效果，还是从娱乐性上都有了显著的提高，这其中最重要的功臣就是扮演着重要角色的 3D 图形库，目前比较常用的有 Windows 中的 DirectX 和跨平台的 OpenGL。而手机 3D 游戏目前才刚刚兴起，除了微软的 Windows Mobile（现在改名为 Windows Phone）使用 DirectX 外，其他的手机操作系统基本都使用 OpenGL ES 或类似的技术作为 3D 图形库。了解并掌握 OpenGL ES 对从事手机游戏编程的开发人员尤其重要，而且现在正是 OpenGL ES 蓬勃发展的时期。

### 19.1 OpenGL 简介

严格地讲，OpenGL 被定义为“图形硬件的开发接口”。从本质上说，它是一个 3D 图形和模型库，具有高度的可移植性，并且速度非常快。使用 OpenGL，您可以创建绚丽、优雅的 3D 图像，其视觉效果可以接近光线追踪仪。

OpenGL 并不像 Java、C 或 C++一样是门编程语言，它更像一个 C 运行时的函数库（也有 Java、C++、Python 等语言的封装），提供了一些预包装的功能。事实上，并不存在所谓的“OpenGL 程序”，而是开发人员使用的语言恰好使用了当前语言封装的 OpenGL 函数库。就像使用 Windows API 访问文件或 Internet 一样，我们可以使用 OpenGL 来创建实时的 3D 图形。

一般而言，OpenGL 是供那些专门设计用来显示和操纵 3D 图形的计算机硬件所使用的。纯软件的 OpenGL 实现也是可能的，Microsoft 所采用的 OpenGL 实现方案就属于此类范畴。使用纯软件的 OpenGL 实现，渲染的速度可能会受到影响（也许这就是 OpenGL 在 Windows 中给人感觉速度不如 DirectX 快的原因），并且一些特殊的效果可能无法实现。但是，使用软件实现意味着我们的程序的移植性大大地增强，可以运行在未安装 3D 图形加速卡的系统。

OpenGL 具有多种用途，从 CAD 工程和建筑应用程序到实现那些令人恐怖的电影特效的建模程序都有 OpenGL 的身影。随着硬件加速和高速的 CPU 越来越普及，3D 图形已经渗透到各类应用程序中，而不仅仅局限于游戏和设计领域。

### 19.2 什么是 OpenGL ES

OpenGL ES（OpenGL for Embedded Systems）与 OpenGL 类似，也是用于编写高级 3D 图形程

序的 API，但所不同的是，OpenGL ES 被广泛用于移动设备，例如，Android、iPhone 的 SDK 中都集成了 OpenGL ES API，而 OpenGL 一般被用在 PC 或服务器上。之所以不直接将 OpenGL 用在移动设备上，主要是因为 OpenGL API 的某些操作对硬件（CPU、GPU）要求过高，目前的移动设备的硬件还远不能与同时代的 PC 相比。因此，一个叫 Khronos 的图形软硬件行业协会对 OpenGL 进行了裁减，最终形成了 OpenGL ES。

### 19.3 多边形

**工程目录：**src\ch19\ch19\_polygon

在介绍如何绘制多边形之前，先了解一下 OpenGL ES 的坐标系。当调用 GL10. glLoadIdentity 方法后，实际上是将当前点移动到了屏幕中心，而屏幕中心点正是 OpenGL 坐标系的原点。坐标系是三维的，也就是沿  $x$ 、 $y$ 、 $z$  轴 3 个方向。读者可以将自己的手机屏幕朝上平放在桌面上， $x$  轴就是手机屏幕从左到右的方向， $y$  轴就是手机屏幕从下到上的方向， $z$  轴就是从桌面到天空的方向，这 3 个坐标轴都以手机屏幕中心为原点。 $x$  轴在屏幕中心左侧的点为负值，右侧的点为正值； $y$  轴在屏幕中心下方的点为负值，上方的点为正值； $z$  轴在屏幕下方的点为负值，在屏幕上方的点为正值。

了解 OpenGL ES 的坐标系之后，就可以使用 OpenGL ES 的框架来绘制多边形了。首先建立一个 MyRender 类，该类是 Renderer 的子类，然后在 onSurfaceChanged 方法中做一些初始化的工作，代码如下：

```
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    float ratio = (float) width / height;
    // 设置 OpenGL 场景的大小
    gl.glViewport(0, 0, width, height);
    // 设置投影矩阵
    gl.glMatrixMode(GL10.GL_PROJECTION);
    // 重置投影矩阵
    gl.glLoadIdentity();
    // 设置 x、y、z 轴方向可设置的最远距离
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    // 选择模型观察矩阵
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    // 重置模型观察矩阵
    gl.glLoadIdentity();
}
```

众所周知，三角形是由 3 个顶点组成的。OpenGL ES 的坐标系是三维的，因此每一个顶点坐标都由 3 个值组成。本例将图形绘制在  $z$  轴的原点处，因此所有坐标的第 3 个值都为 0。下面先定义三角形的 3 个顶点的坐标。

```
private IntBuffer triangleBuffer;
private int[] triangleVertices = new int[]
```

```

{ 0, one, 0,           // 上顶点
-one, -one, 0,         // 左下顶点
one, -one, 0 };        // 右下顶点

```

四边形由 4 个顶点组成，下面定义了这 4 个顶点的坐标。

```

private IntBuffer quaterBuffer;
private int[] quaterVertices = new int[]
{ one, one, 0,           // 右上角顶点
-one, one, 0,           // 左上角顶点
one, -one, 0,           // 右下角顶点
-one, -one, 0 };        // 左下角顶点

```

由于绘制多边形的方法需要 IntBuffer 对象，因此，需要为三角形和四边形各定义一个 IntBuffer 类型的变量，这两个变量需要在 onSurfaceCreated 方法（创建绘制 3D 图形的窗口时调用该方法）中初始化，代码如下：

```

public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 根据三角形顶点数创建一个 ByteBuffer 对象。
    // 由于一个 int 类型的值占 4 个字节，
    // 因此，应分配的空间大小应为顶点坐标数的 4 倍
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect(triangleVertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    triangleBuffer = byteBuffer.asIntBuffer();
    // 将三角形顶点坐标放到 IntBuffer 类型变量中
    triangleBuffer.put(triangleVertices);
    // 将缓冲区指针指向第 1 个字节的位置
    triangleBuffer.position(0);

    byteBuffer = ByteBuffer.allocateDirect(quaterVertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    quaterBuffer = byteBuffer.asIntBuffer();
    quaterBuffer.put(quaterVertices);
    quaterBuffer.position(0);
}

```

**注意** 在 Android 2.3 中不能像老版本的 Android 一样使用 IntBuffer.wrap(new int[]{...}) 初始化 IntBuffer 对象（FloatBuffer 也是一样），否则系统会抛出异常。应直接使用 allocateDirect 方法为缓冲区分配空间，再使用 ByteBuffer.put 方法将原始数据放到缓冲区中。

本例在左侧绘制一个四边形，在右侧绘制一个三角形，所以需要使用 glTranslatef 方法将坐标原点移至三角形的位置，如下面的代码将坐标原点延 x 轴向右移动 1.5 个单位，y 轴不动，z 轴移入屏幕 6 个单位：

```
| gl.glTranslatef(1.5f, 0.0f, -6.0f);
```

当前的中点已经移到了屏幕的右侧，并且将视图推入屏幕背后足够的距离以便可以看见全部的场景。要注意的是，这里移动的单位必须小于使用 `glFrustumf` 方法设置的最远距离，否则显示不出来。例如， $z$  轴的最远距离是 10， $z$  轴中点向屏幕背后移动不能超过 10。不管是绘制三角形，还是绘制四边形，都需要设置顶点，因此要使用如下代码告诉 OpenGL 要设置顶点这个功能：

```
| gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
```

在绘制多边形之前，需要指定顶点以及与顶点相关的信息，下面的代码为三角形和四边形指定了顶点信息。

```
| // 设置三角形的顶点坐标  
| gl.glVertexPointer(3, GL10.GL_FIXED, 0, triggerBuffer);  
| // 设置正方形的顶点坐标  
| gl.glVertexPointer(3, GL10.GL_FIXED, 0, quaterBuffer);
```

其中 `glVertexPointer` 方法的第一个参数表示坐标系的维度（要注意，该参数不是坐标数组的尺寸）。由于 OpenGL 是三维坐标系，因此该参数值是 3。第 2 个参数表示顶点的类型，本例中的数据是固定的，所以使用 `GL_FIXED` 表示固定的顶点。第 3 个参数表示步长，第 4 个参数表示顶点缓存（也就是前面定义的两个坐标数组）。

最后一步是使用 `glDrawArrays` 方法绘制三角形和四边形，代码如下：

```
| gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);  
| gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
```

现在绘制三角形和四边形的代码已经完成了，下面是完整的绘制代码。

```
public void onDrawFrame(GL10 gl)  
{  
    // 清除屏幕和深度缓存  
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);  
    // 允许设置顶点  
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);  
    // 重置当前的模型观察矩阵  
    gl.glLoadIdentity();  
    // 左移 1.5 单位，并移入屏幕 6.0 个单位  
    gl.glTranslatef(1.5f, 0.0f, -6.0f);  
    // 设置三角形的顶点坐标  
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, triggerBuffer);  
    // 绘制三角形  
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);  
    // 重置当前的模型观察矩阵  
    gl.glLoadIdentity();  
    // 左移 2.0 个单位，并移入屏幕 6.0 个单位  
    gl.glTranslatef(-2.0f, 0.0f, -6.0f);  
    // 设置正方形的顶点坐标  
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, quaterBuffer);  
    // 绘制正方形
```

```

gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
// 在开启顶点设置功能后，必须使用下面的代码关闭（取消）顶点设置功能
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
}

```

在编写完 MyRender 类后，需要在主类（Main）的 onCreate 方法中创建 MyRender 对象，代码如下：

```

public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    GLSurfaceView glView = new GLSurfaceView(this);
    MyRender myRender = new MyRender();
    glView.setRenderer(myRender);
    setContentView(glView);
}

```

运行本例，会显示如图 19.1 所示的图形。

在绘制四边形时使用了 GL10.GL\_TRIANGLE\_STRIP，代码如下：

```
gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
```

实际上，还有另外一种绘制多边形的方式，就是使用 GL10.GL\_TRIANGLE\_FAN。这两种绘制多边形的方式有什么区别呢？现在先将 GL10.GL\_TRIANGLE\_FAN 换成 GL10.GL\_TRIANGLE\_STRIP，然后运行本例，效果如图 19.2 所示。

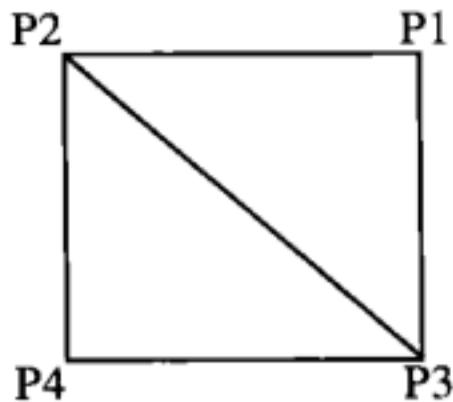


▲ 图 19.1 绘制三角形和四边形

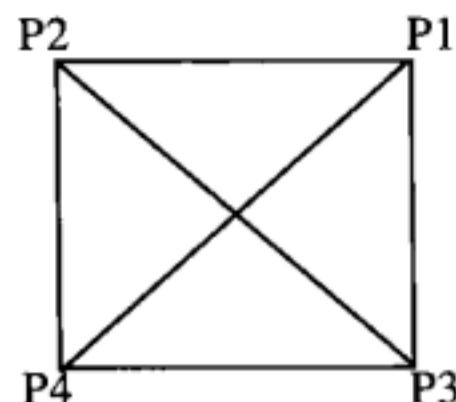


▲ 图 19.2 使用 GL10.GL\_TRIANGLE\_STRIP 绘制多边形

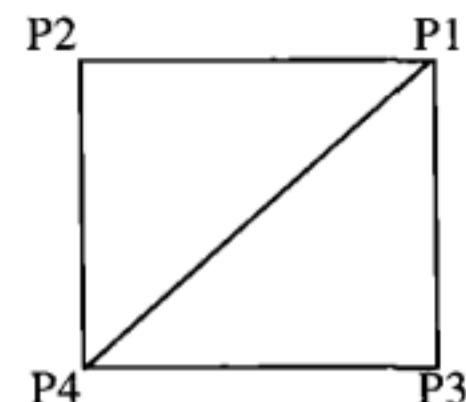
也许很多读者看到如图 19.1 所示的效果会感到奇怪，明明画的是四边形，怎么画出五边形。实际上，GL10.GL\_TRIANGLE\_STRIP 和 GL10.GL\_TRIANGLE\_FAN 都需要通过三角形绘制多边形，只是它们绘制三角形时取顶点的规则不同。假设四边形的 4 个顶点的定义顺序是 P1、P2、P3、P4。通过 GL10.GL\_TRIANGLE\_STRIP 绘制多边形时是按 (P1、P2、P3)、(P2、P3、P4) 取的顶点，每一组顶点就是一个三角形的顶点。如果这 4 个顶点按如图 19.3 所示的顶点取值，那么按 GL10.GL\_TRIANGLE\_STRIP 的规则正好沿反斜杠“\”方向画两个三角形。而 GL10.GL\_TRIANGLE\_FAN 是按 (P1、P2、P3)、(P1、P3、P4) 取的三角形顶点。如果在这种情况下仍然按如图 19.3 所示的取点规则，就会画出如图 19.4 所示的两个三角形，会造成左边缺一个三角形的效果。要想使用 GL10.GL\_TRIANGLE\_FAN 绘制多边形，必须采用如图 19.5 所示的取点规则。由此可见，多边形的顶点不能按任意顺序定义，只能根据使用取三角形顶点的模式来定义多边形的顶点。



▲ 图 19.3 成功绘制四边形



▲ 图 19.4 绘制四边形失败



▲ 图 19.5 成功绘制四边形

## 19.4 颜色

**工程目录:** `src\ch19\ch19_color`

通过上一节的学习我们已经可以使用 OpenGL ES 绘制三角形和四边形了，本节将介绍为三角形和四边形填充颜色。下面先看看如图 19.6 所示的填充颜色的效果。

如图 19.6 所示，3 个图形都使用了渐变颜色进行填充。对图形着色需要从三角形和四边形的顶点定义起始颜色。每一种颜色由 4 个值组成，这 4 个值是 R、G、B、A，其中 A 是透明度。下面的代码是为三角形定义的一个颜色数组。

```
int one = 0x10000;
private IntBuffer colorBuffer;
private int[] colorVertices = new int[]
{ one, 0, 0, one, 0, one, 0, 0, one, one, one };
```

为多边形着色也需要先开启颜色渲染功能，代码如下：

```
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
```

然后通过 `glColorPointer` 方法可以进行着色，代码如下：

```
gl.glColorPointer(4, GL10.GL_FIXED, 0, colorBuffer);
```

`glColorPointer` 方法的 4 个参数的含义与 `glVertexPointer` 方法的相应参数类似，但要注意的是第一个参数表示每一个颜色的值的数目（R、G、B、A）。

对多边形着色后，需要使用 `glDisableClientState` 方法关闭颜色渲染功能，代码如下：

```
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
```

如想使用单调着色，可以直接调用 `glColor4f` 方法设置颜色值，代码如下：

```
// 设置颜色 (R、G、B、A)
gl glColor4f(1.0f, 0.0f, 0.0f, 0.0f);
```

`glColor4f` 不需要开启颜色渲染功能，因此，需要在调用 `glColor4f` 方法之前使用 `glDisableClientState` 方法关闭颜色渲染功能，否则 `glColor4f` 方法不起作用。

与上一节的例子相同，也需要在 `onSurfaceCreated` 方法中对 `IntBuffer` 及 `FloatBuffer` 类型的变量



▲ 图 19.6 填充颜色后的图形

初始化。

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect(triangleVertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    triangleBuffer = byteBuffer.asFloatBuffer();
    triangleBuffer.put(triangleVertices);
    triangleBuffer.position(0);

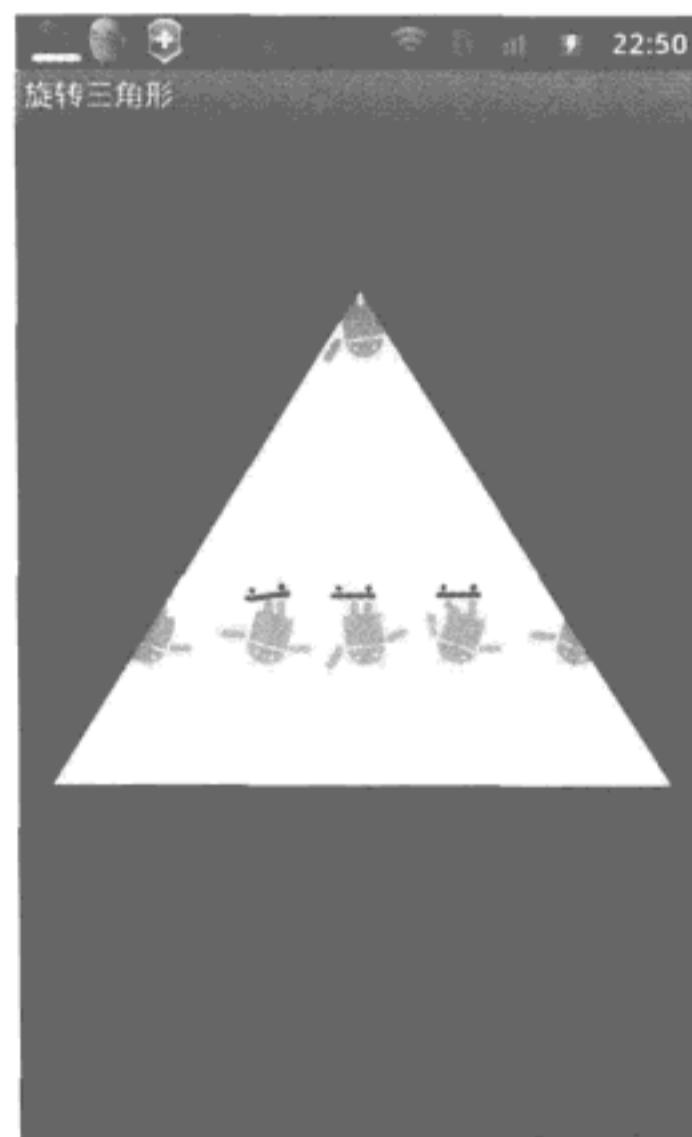
    byteBuffer = ByteBuffer.allocateDirect(quaterVertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    quaterBuffer = byteBuffer.asFloatBuffer();
    quaterBuffer.put(quaterVertices);
    quaterBuffer.position(0);

    byteBuffer = ByteBuffer.allocateDirect(colorVertices.length * 4);
    byteBuffer.order(ByteOrder.nativeOrder());
    colorBuffer = byteBuffer.asIntBuffer();
    colorBuffer.put(colorVertices);
    colorBuffer.position(0);
}
```

## 19.5 旋转三角形

工程目录: `src\ch19\ch19_rotate_triangle`

本节将为读者展示第一个使用 OpenGL ES 制作的动画, 这个动画是一个顺时针旋转的三角形, 效果如图 19.7 所示。



▲ 图 19.7 旋转三角形

本例使用了 ETC1 格式的文件（res\raw\androids.pkm）来存储旋转三角形中的数据，并使用 ETC1Util.loadTexture 方法来装载这些数据，代码如下：

```
private class CompressedTextureLoader implements MyRenderer.TextureLoader
{
    public void load(GL10 gl)
    {
        InputStream input = getResources().openRawResource(R.raw.androids);
        try
        {
            ETC1Util.loadTexture(GLES10.GL_TEXTURE_2D, 0, 0, GLES10.GL_RGB,
                GLES10.GL_UNSIGNED_SHORT_5_6_5, input);
        }
        catch (IOException e)
        {
        }
        finally
        {
            try
            {
                input.close();
            }
            catch (IOException e)
            {

            }
        }
    }
}
```

旋转三角形非常简单，只需要在 onDrawFrame 方法中使用 glRotatef 不断变化图形当前的角度即可，代码如下：

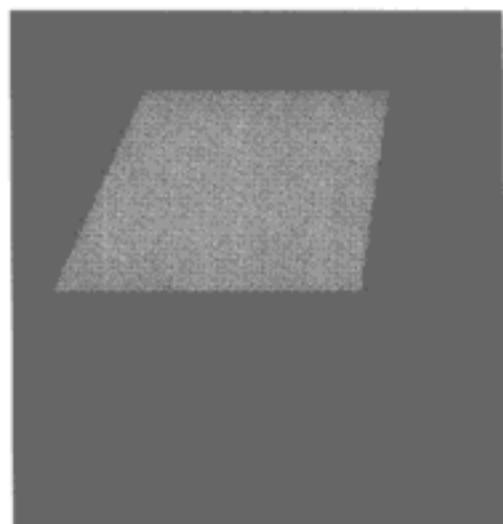
```
public void onDrawFrame(GL10 gl)
{
    glDisable(GL_DITHER);
    glTexEnvx(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 1.0f, 0.0f);
    glEnableClientState(GL_VERTEX_ARRAY);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);
    setActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, mTextureID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    long time = SystemClock.uptimeMillis() % 4000L;
    float angle = 0.090f * ((int) time);
    // 旋转三角形
    glRotatef(angle, 0, 0, 1.0f);
    // 重新绘制图形，这时 onDrawFrame 方法会再次被调用
    mTriangle.draw(gl);
}
```

## 19.6 旋转立方体

工程目录: src\ch19\ch19\_rotate\_cube

本节的例子将在屏幕中心绘制一个立方体，并且使立方体旋转，效果如图 19.8 所示。



▲ 图 19.8 旋转的立方体

立方体由 6 个正方形组成，而且这 6 个正方形坐标要按顺时针方向定义，千万不能既顺时针又逆时针定义。下面是定义立方体各顶点坐标的代码。

```
private IntBuffer quaterBuffer;
private int[] quaterVertices = new int[]
{ one, one, -one, -one, one, -one, one, one, one, -one, one, one,
one, -one, one, -one, -one, one, one, -one, -one, -one, -one, -one,
one, one, one, -one, -one, one, one, -one, -one, -one, -one, one,
one, -one, -one, -one, -one, one, one, -one, -one, -one, one, -one,
-one, one, one, -one, -one, -one, one, -one, -one, -one, -one, one,
one, one, -one, one, one, -one, -one, one, -one, -one, -one, one, one, };
```

下面的代码定义了立方体各个面的颜色。

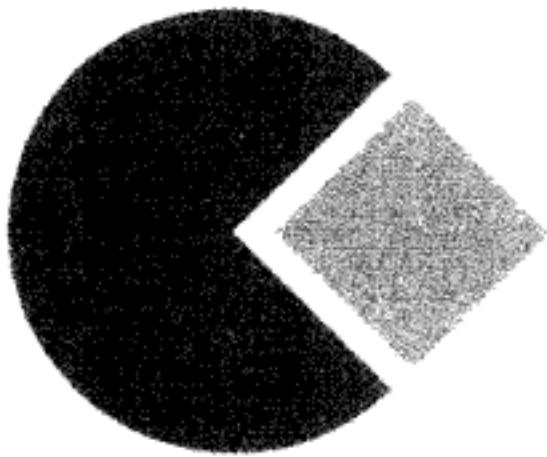
```
private IntBuffer colorBuffer;
private int[] colors = new int[]
{
    one / 2, one, 0, one, one / 2, one, 0, one, one / 2, one, 0, one,
    one, one / 2, 0, one, one, one / 2, 0, one, one, one / 2, 0, one,
    one, one / 2, 0, one, one, 0, one, one, one, 0, one, one, one,
    0, one, one, 0, one, one, 0, 0, one, one, 0, 0, one, one, 0,
    0, one, one, 0, 0, one, 0, one, one, 0, 0, one, one, 0, 0, one, one,
    one, 0, one, one, 0, one, one, 0, one, one, 0, one, one, 0, one,
    one, };
```

下面的代码用于绘制立方体。

```
//绘制立方体
for (int i = 0; i < 6; i++)
{
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, i * 4, 4);
```

## 19.7 小结

本章讲解了如何使用 OpenGL ES API 绘制 2D、3D 图形。虽然 OpenGL ES 是 OpenGL 的子集，但 OpenGL ES 的功能已经足够让我们实现超级绚丽的效果了，而且也可以通过某些技巧弥补从 OpenGL 去掉的 API。如果手机中有硬件图形卡（如 Nexus S），运行复杂的 3D 效果会更加流畅。



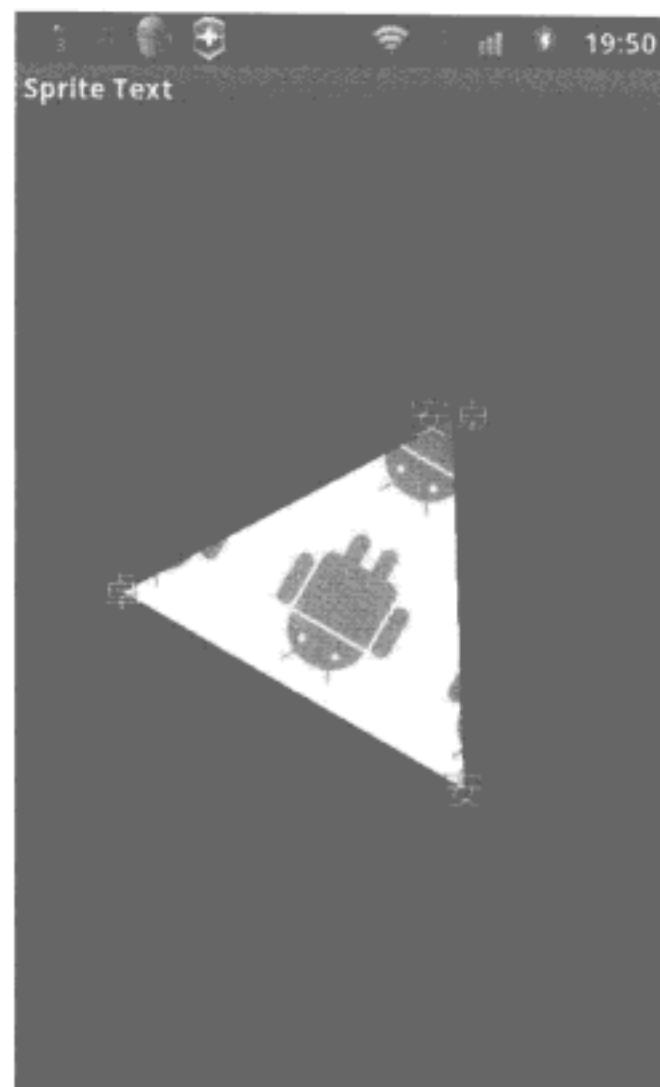
## 第 20 章 OpenGL ES 的超酷效果

本章使用 OpenGL ES API 实现了很多非常酷的效果。例如，可保持平衡的文本和旋转的三角形、左右摇摆的机器人、变换万千的彩色立方体、旋转立体天空。这些效果充分说明了 OpenGL ES 在 Android 上也能完成令人叹为观止的效果，尤其是在有硬件图形卡和高分辨率屏幕的手机上效果会更出色。

### 20.1 保持平衡的旋转文本

工程目录：src\ch20\ch20\_sprite\_text

19.5 节实现了一个旋转的三角形，本节将实现一个更有趣的旋转动画。在屏幕中心仍然有一个旋转的三角形，而三角形的 3 个顶点分别显示 3 个字符串（“安卓”、“安”、“卓”）。当三角形旋转时，这 3 个字符串也跟着三角形旋转，但字符串始终保持着水平状态，如图 20.1 所示。



▲ 图 20.1 保持平衡的旋转文本

本例的核心类是 MyRenderer。首先需要在 MyRenderer 类的构造方法中进行一些初始化的工作，

代码如下：

```
public MyRenderer(Context context)
{
    mContext = context;
    // 创建用于描述旋转三角形的对象
    mTriangle = new Triangle();
    // Projector 对象用于确定三角形顶点文字的当前位置。
    // 如果不使用该对象，文字将不会随着三角形旋转
    mProjector = new Projector();
    // Paint 对象用于设置三角形顶点文字的颜色、大小等属性
    mLabelPaint = new Paint();
    mLabelPaint.setTextSize(32);
    mLabelPaint.setAntiAlias(true);
    // 顶点文字的颜色为黄色
    mLabelPaint.setARGB(0xff, 0xff, 0xff, 0x00);
}
```

初始化 OpenGL ES 的工作需要在 `onSurfaceCreated` 方法中完成，代码如下：

```
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 如果打开这个开关，在颜色组件或者索引被写进颜色缓存之前抖动。
    // 虽然提升了图像质量，但会影响性能。因此，关闭了抖动开关
    gl.glDisable(GL10.GL_DITHER);

    // 对 OpenGL ES 进行一些初始化工作
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);

    gl.glClearColor(0f, 0f, 0f, 1);
    gl.glShadeModel(GL10.GL_SMOOTH);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    gl.glEnable(GL10.GL_TEXTURE_2D);

    // 创建纹理效果。每次创建 OpenGL ES 画布时必须做这项工作
    int[] textures = new int[1];
    gl glGenTextures(1, textures, 0);

    mTextureID = textures[0];
    gl glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);

    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
                       GL10.GL_NEAREST);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MAG_FILTER,
                       GL10.GL_LINEAR);

    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
                       GL10.GL_CLAMP_TO_EDGE);
    gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
                       GL10.GL_CLAMP_TO_EDGE);

    gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
                 GL10.GL_REPLACE);
```

```
// 开始读取在三角形上显示的图像(一个Android小机器人)
InputStream is = mContext.getResources().openRawResource(R.raw.robot);
Bitmap bitmap;
try
{
    bitmap = BitmapFactory.decodeStream(is);
}
finally
{
    try
    {
        is.close();
    }
    catch (IOException e)
    {
    }
}
// 确定绘制图像的格式
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
bitmap.recycle();
// LabelMaker 对象封装了所有显示在顶点的文本
mLabels = new LabelMaker(true, 256, 64);
mLabels.initialize(gl);
// 开始设置在三角形顶点显示的文字
mLabels.beginAdding(gl);
mLabelA = mLabels.add(gl, "安", mLabelPaint);
mLabelB = mLabels.add(gl, "卓", mLabelPaint);
mLabelC = mLabels.add(gl, "安卓", mLabelPaint);
mLabels.endAdding(gl);
}
```

最关键的一步就是在 `onDrawFrame` 方法中绘制图形和文字，并不断用 `GL10.glRotatef` 方法旋转三角形和文字。

```
public void onDrawFrame(GL10 gl)
{
    gl.glDisable(GL10.GL_DITHER);
    gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
                 GL10.GL_MODULATE);

    // 在绘图之前，首先应该做的就是清除屏幕。最有效的清除屏幕的方式就是调用 glClear 方法
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

    // 下面的代码开始绘制图形和文字

    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();

    GLU.gluLookAt(gl, 0.0f, 0.0f, -2.5f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

    gl.glActiveTexture(GL10.GL_TEXTURE0);
```

```

gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
    GL10.GL_REPEAT);
gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
    GL10.GL_REPEAT);

long time = SystemClock.uptimeMillis() % 4000L;
// 利用当前时间计算当前的图形角度
float angle = 0.090f * ((int) time);
gl.glRotatef(angle, 0, 0, 1.0f);
// 将图形放大一倍
gl.glScalef(2.0f, 2.0f, 2.0f);
// 绘制三角形
mTriangle.draw(gl);
// 确定文字的当前位置
mProjector.getCurrentModelView(gl);
// 开始绘制文字
mLabels.beginDrawing(gl, mWidth, mHeight);
drawLabel(gl, 0, mLabelA);
drawLabel(gl, 1, mLabelB);
drawLabel(gl, 2, mLabelC);
mLabels.endDrawing(gl);
}

```

## 20.2 左右摇摆的 Android 机器人

工程目录: src\ch20\ch20\_sway\_robot

想不想让很多小机器人摆来摆去? 如果想的话, 就看看本节的例子, 也许读者会感到很有趣。这个例子会在屏幕中间显示一个白色背景的矩形, 在其中会显示 4 行 2 列, 共 8 个小机器人。这个矩形连同 8 个小机器人会左右摇摆, 在摇摆的同时, 矩形和机器人会变形扭曲, 如图 20.2 所示。



▲ 图 20.2 左右摇摆的机器人

首先来绘制白色背景矩形和 8 个小机器人。绘制工作是由 Grid 类完成的，Grid 类的构造方法会对网络进行一些初始化，代码如下：

```
public Grid(int w, int h)
{
    ...
    // 初始化网格数据
    int i = 0;
    for (int y = 0; y < quadH; y++)
    {
        for (int x = 0; x < quadW; x++)
        {
            char a = (char) (y * mW + x);
            char b = (char) (y * mW + x + 1);
            char c = (char) ((y + 1) * mW + x);
            char d = (char) ((y + 1) * mW + x + 1);

            mIndexBuffer.put(i++, a);
            mIndexBuffer.put(i++, c);
            mIndexBuffer.put(i++, b);

            mIndexBuffer.put(i++, b);
            mIndexBuffer.put(i++, c);
            mIndexBuffer.put(i++, d);
        }
    }
}
```

Grid 类的核心方法是 draw。在 onDrawFrame 方法中通过调用 Grid.draw 方法来绘制矩形的 8 个小机器人。

```
public void draw(GL10 gl)
{
    GL11 gl11 = (GL11) gl;
    GL11Ext gl11Ext = (GL11Ext) gl;

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl11.glBindBuffer(GL11.GL_ARRAY_BUFFER, mVertexBufferObjectId);
    gl11.glVertexPointer(3, GL10.GL_FLOAT, VERTEX_SIZE, 0);
    gl11.glTexCoordPointer(2, GL10.GL_FLOAT, VERTEX_SIZE,
                           VERTEX_TEXTURE_BUFFER_INDEX_OFFSET * FLOAT_SIZE);
    gl.glEnableClientState(GL11Ext.GL_MATRIX_INDEX_ARRAY_OES);
    gl.glEnableClientState(GL11Ext.GL_WEIGHT_ARRAY_OES);

    gl11Ext.glWeightPointerOES(2, GL10.GL_FLOAT, VERTEX_SIZE,
                               VERTEX_WEIGHT_BUFFER_INDEX_OFFSET * FLOAT_SIZE);
    gl11Ext.glMatrixIndexPointerOES(2, GL10.GL_UNSIGNED_BYTE, VERTEX_SIZE,
                                    VERTEX_PALETTE_INDEX_OFFSET);

    gl11.glBindBuffer(GL11.GL_ELEMENT_ARRAY_BUFFER, mElementBufferObjectId);
    // 开始绘制矩形和 8 个小机器人
    gl11.glDrawElements(GL10.GL_TRIANGLES, mIndexCount,
                        GL10.GL_UNSIGNED_SHORT, 0);
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
```

```

    gl.glDisableClientState(GL11Ext.GL_MATRIX_INDEX_ARRAY_OES);
    gl.glDisableClientState(GL11Ext.GL_WEIGHT_ARRAY_OES);
    gl11.glBindBuffer(GL11.GL_ARRAY_BUFFER, 0);
    gl11.glBindBuffer(GL11.GL_ELEMENT_ARRAY_BUFFER, 0);
}

```

下面看一下本例中最核心的代码 `MyRenderer.onDrawFrame`，在该类中绘制了所有的图形，并产生了动画效果。

```

public void onDrawFrame(GL10 gl)
{
    // 关闭抖动
    gl.glDisable(GL10.GL_DITHER);
    gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
        GL10.GL_MODULATE);
    // 清屏
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    gl.glEnable(GL10.GL_CULL_FACE);

    // 开始绘制图形
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

    gl.glActiveTexture(GL10.GL_TEXTURE0);
    gl glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
        GL10.GL_REPEAT);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
        GL10.GL_REPEAT);

    long time = SystemClock.uptimeMillis() % 4000L;

    // 产生摆动效果（也就是利用三角函数使角度波动）
    double animationUnit = ((double) time) / 4000;
    float unitAngle = (float) Math.cos(animationUnit * 2 * Math.PI);
    float angle = unitAngle * 135f;

    gl.glEnable(GL11Ext.GL_MATRIX_PALETTE_OES);
    gl.glMatrixMode(GL11Ext.GL_MATRIX_PALETTE_OES);
    GL11Ext gl11Ext = (GL11Ext) gl;

    gl11Ext.glCurrentPaletteMatrixOES(0);
    gl11Ext.glLoadPaletteFromModelViewMatrixOES();
    // 摆动图形
    gl.glRotatef(angle, 0, 0, 1.0f);
    gl11Ext.glCurrentPaletteMatrixOES(1);
    gl11Ext.glLoadPaletteFromModelViewMatrixOES();
    // 绘制矩形和 8 个小机器人
    mGrid.draw(gl);
}

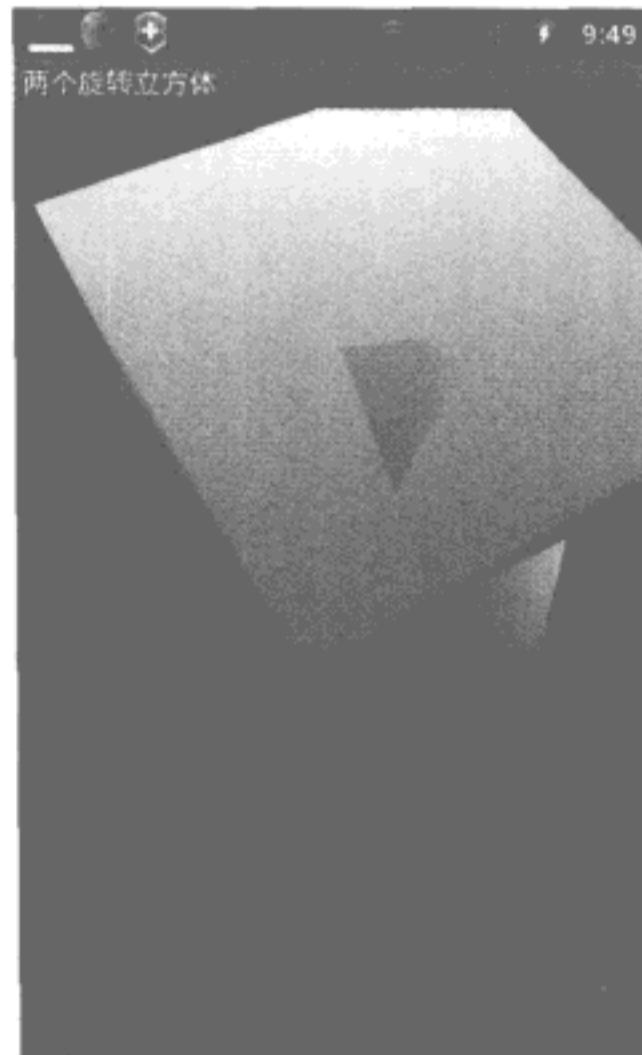
```

```
    gl.glDisable(GL11Ext.GL_MATRIX_PALETTE_OES);  
}
```

## 20.3 纠缠在一起的旋转立方体

工程目录: src\ch20\ch20\_two\_rotate\_cube

19.6 小节实现了一个旋转的立方体, 然而, 这个立方体只是按固定方向旋转, 而且只有一个。本小节将实现两个纠缠在一起的旋转立方体, 效果如图 20.3 所示。



▲ 图 20.3 纠缠在一起的旋转立方体

本例中的 `onDrawFrame` 方法需要处理两个立方体不同的旋转角度, 并调用 `Cube.draw` 方法绘制两个立方体。其中 `Cube` 类用于描述立方体的数据和绘制立方体。

```
public void onDrawFrame(GL10 gl)  
{  
    gl.glClearColor(0f, 0f, 0f, 1f);  
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);  
  
    // 开始绘制旋转立方体  
  
    gl.glMatrixMode(GL10.GL_MODELVIEW);  
    gl.glLoadIdentity();  
    gl.glTranslatef(0, 0, -3.0f);  
    // 设置第 1 个旋转立方体的角度  
    gl.glRotatef(mAngle, 0, 1, 0);  
    gl.glRotatef(mAngle * 0.25f, 1, 0, 0);  
  
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);  
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
```

```
// 绘制第 1 个旋转立方体
mCube.draw(gl);
// 设置第 2 个旋转立方体的角度
gl.glRotatef(mAngle * 2.0f, 0, 1, 1);
gl.glTranslatef(0.5f, 0.5f, 0.5f);
// 绘制第 2 个旋转立方体
mCube.draw(gl);
// 确定下一个要旋转的角度
mAngle += 1.2f;
}
```

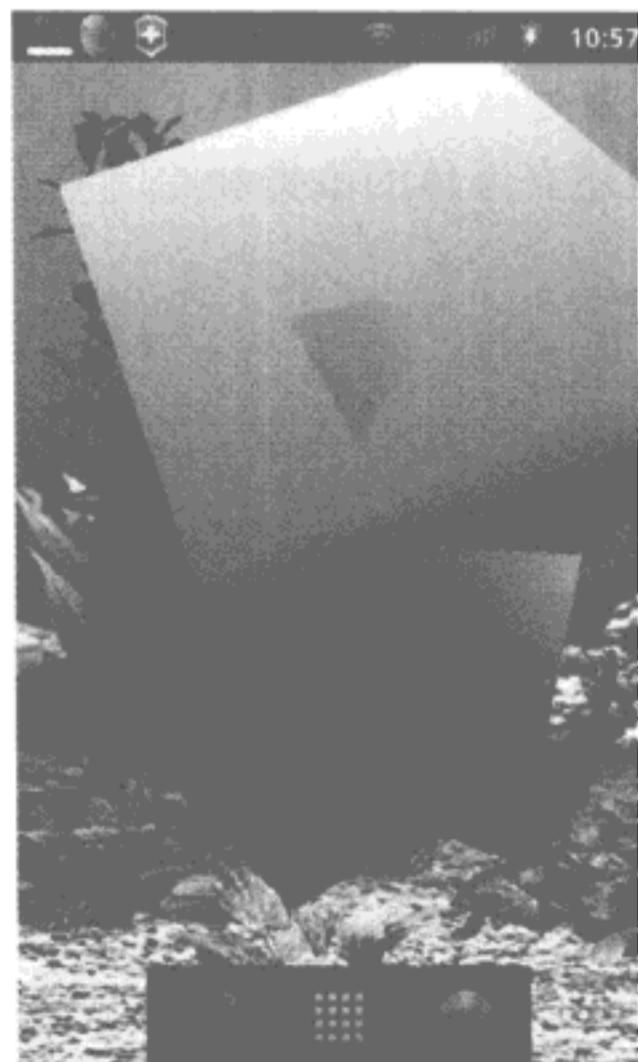
Cube 类除了定义和初始化立方体顶点数据外，还有一个重要的 draw 方法用于绘制立方体，代码如下：

```
public void draw(GL10 gl)
{
    gl.glFrontFace(gl.GL_CW);
    gl.glVertexPointer(3, gl.GL_FIXED, 0, mVertexBuffer);
    gl glColorPointer(4, gl.GL_FIXED, 0, mColorBuffer);
    // 绘制立方体
    gl.glDrawElements(gl.GL_TRIANGLES, 36, gl.GL_UNSIGNED_BYTE, mIndexBuffer);
}
```

## 20.4 透明背景的旋转立方体

工程目录：src\ch20\ch20\_translucent\_rotate\_cube

本小节的例子与上一节的例子基本实现方法相同，只是将背景设为透明。效果如图 20.4 所示。背景是手机的动态桌面。



▲ 图 20.4 透明背景的旋转立方体

将背景设为透明首先要将 Activity 设为透明，在 AndroidManifest.xml 文件中按如下代码修改 Activity 的配置。

```
<activity android:name=".Main"
    android:label="@string/app_name" android:theme="@style/Theme.Translucent">
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

其中 Theme.Translucent 主题的代码（在 res\values\strings.xml 文件中定义）如下：

```
<style name="Theme.Translucent" parent="android:style/Theme.Translucent">
<item name="android:windowBackground">@drawable/translucent_background</item>
<item name="android:windowNoTitle">true</item>
<item name="android:colorForeground">#fff</item>
</style>
```

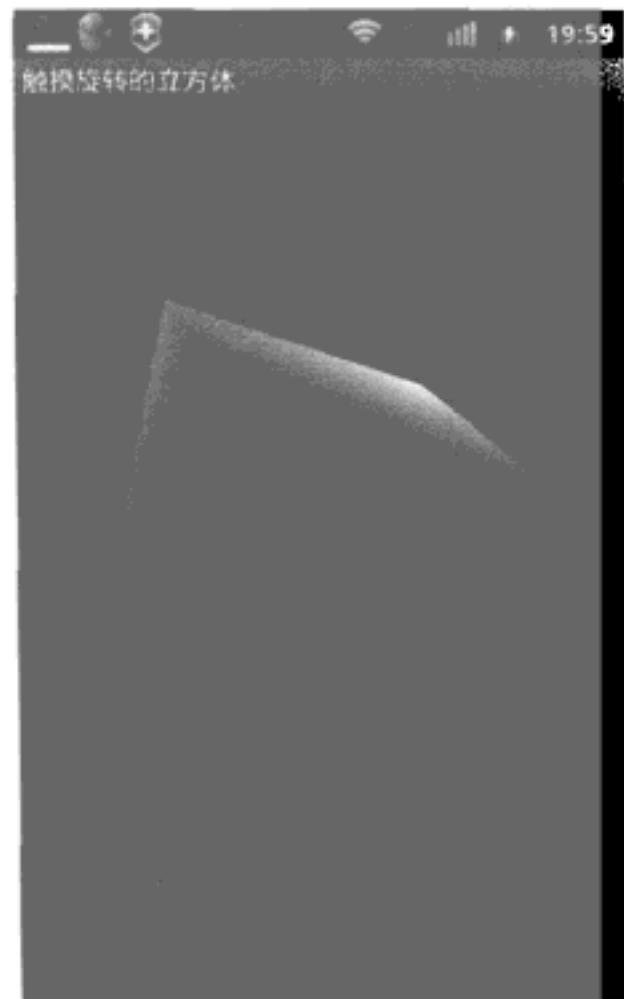
最后需要使用如下的代码将 OpenGL ES 的画布设为透明。

```
gl.glClearColor(0, 0, 0, 0);
```

## 20.5 触摸旋转的立方体

工程目录：src\ch20\ch20\_touch\_rotate\_cube

前面实现的旋转立方体都是在 onDrawFrame 方法中通过 glRotatef 方法不断变化角度而自动旋转的，而本小节会利用手指在屏幕上移动来控制立方体的滚动。如果手机上有轨迹球，也可使用轨迹球来控制立方体的滚动，效果如图 20.5 所示。



▲ 图 20.5 通过手指移动或轨迹球控制的立方体

**TouchSurfaceView** 类负责处理手指触摸和轨迹球动作。该类继承自 **GLSurfaceView**。**TouchSurfaceView** 类中的 **onTouchEvent** 和 **onTrackballEvent** 方法分别用于处理手指触摸和轨迹球动作，代码如下：

```
// 处理轨迹球动作
public boolean onTrackballEvent(MotionEvent e)
{
    // 根据轨迹球的滚动变化 x 轴方向的角度
    mRenderer.mAngleX += e.getX() * TRACKBALL_SCALE_FACTOR;
    // 根据轨迹球的滚动变化 y 轴方向的角度
    mRenderer.mAngleY += e.getY() * TRACKBALL_SCALE_FACTOR;
    // 滚动立方体
    requestRender();
    return true;
}
// 处理手指触摸动作
public boolean onTouchEvent(MotionEvent e)
{
    float x = e.getX();
    float y = e.getY();
    switch (e.getAction())
    {
        case MotionEvent.ACTION_MOVE:
            float dx = x - mPreviousX;
            float dy = y - mPreviousY;
            // 根据手指移动计算 x 轴方向的角度
            mRenderer.mAngleX += dx * TOUCH_SCALE_FACTOR;
            // 根据手指移动计算 y 轴方向的角度
            mRenderer.mAngleY += dy * TOUCH_SCALE_FACTOR;
            // 滚动立方体
            requestRender();
    }
    mPreviousX = x;
    mPreviousY = y;
    return true;
}
```

**TouchSurfaceView** 类的内部有一个内嵌类：**CubeRenderer**，用于绘制立方体，代码如下：

```
private class CubeRenderer implements GLSurfaceView.Renderer
{
    private Cube mCube;
    public float mAngleX;
    public float mAngleY;

    public CubeRenderer()
    {
        mCube = new Cube();
    }
    public void onDrawFrame(GL10 gl)
    {
        // 将屏幕背景设为黑色
    }
}
```

```

gl.glClearColor(0f, 0f, 0f, 1f);
gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

// 开始绘制图形

gl.glMatrixMode(GL10.GL_MODELVIEW);
gl.glLoadIdentity();
gl.glTranslatef(0, 0, -3.0f);
// 设置立方体滚动角度
gl.glRotatef(mAngleX, 0, 1, 0);
gl.glRotatef(mAngleY, 1, 0, 0);

gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
// 开始绘制立方体，其中 mCube 是 Cube 类型变量。Cube 类与前面例子中的 Cube 类完全相同。
mCube.draw(gl);
}

public void onSurfaceChanged(GL10 gl, int width, int height)
{
    gl.glViewport(0, 0, width, height);
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
}

public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    gl.glDisable(GL10.GL_DITHER);
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
    gl.glClearColor(1, 1, 1, 1);
    gl.glEnable(GL10.GL_CULL_FACE);
    gl.glShadeModel(GL10.GL_SMOOTH);
    gl.glEnable(GL10.GL_DEPTH_TEST);
}
}

```

## 20.6 2D 和 3D 的综合旋转效果

工程目录: src\ch20\ch20\_combination\_rotate

本小节的例子将前面介绍的几种旋转效果综合在了一起，最外面有一个顺时针旋转的三角形，里面有不断旋转的立方体，效果如图 20.6 所示。

本例的核心类是 MyRenderer，该类继承自 GLSurfaceView.Renderer。MyRenderer.onDrawFrame 方法的代码如下：

```

public void onDrawFrame(GL10 gl)
{
    GL11ExtensionPack gl11ep = (GL11ExtensionPack) gl;
    gl11ep.glBindFramebufferOES(GL11ExtensionPack.GL_FRAMEBUFFER_OES, mFramebuffer);
    drawOffscreenImage(gl, mFramebufferWidth, mFramebufferHeight);
}

```

## Android 开发权威指南

```

gl11ep.glBindFramebufferOES(GL11ExtensionPack.GL_FRAMEBUFFER_OES, 0);
// 绘制屏幕上的图形
drawOnscreen(gl, mSurfaceWidth, mSurfaceHeight);
}

```



▲ 图 20.6 综合旋转效果

上面代码中涉及了两个重要的方法：`drawOffscreenImage` 和 `drawOnscreen`，其中 `drawOffscreenImage` 绘制了三角形内部的图形，`drawOnscreen` 方法绘制了三角形。

```

private void drawOffscreenImage(GL10 gl, int width, int height)
{
    gl.glViewport(0, 0, width, height);
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);

    gl.glEnable(GL10.GL_CULL_FACE);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    // 将背景，也就是三角形颜色设为蓝色
    gl.glClearColor(0, 0, 1, 0);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0, 0, -3.0f);
    // 旋转三角形中心的立方体
    gl.glRotatef(mAngle, 0, 1, 0);
    gl.glRotatef(mAngle * 0.25f, 1, 0, 0);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    // 绘制三角形中心的立方体
    mCube.draw(gl);
}

```

```
// 旋转三角形角落处的立方体
gl.glRotatef(mAngle * 2.0f, 0, 1, 1);
gl.glTranslatef(0.5f, 0.5f, 0.5f);

// 绘制三角形角落的立方体
mCube.draw(gl);

mAngle += 1.2f;

gl.glDisable(GL10.GL_CULL_FACE);
gl.glDisable(GL10.GL_DEPTH_TEST);
gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
}

private void drawOnscreen(GL10 gl, int width, int height)
{
    gl.glViewport(0, 0, width, height);
    float ratio = (float) width / height;
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();
    gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);
    // 将背景设为黑色
    gl.glClearColor(0, 0, 0, 1);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl glBindTexture(GL10.GL_TEXTURE_2D, mTargetTexture);
    gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
        GL10.GL_REPLACE);

    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 1.0f, 0.0f);

    gl glEnableClientState(GL10.GL_VERTEX_ARRAY);
    gl glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

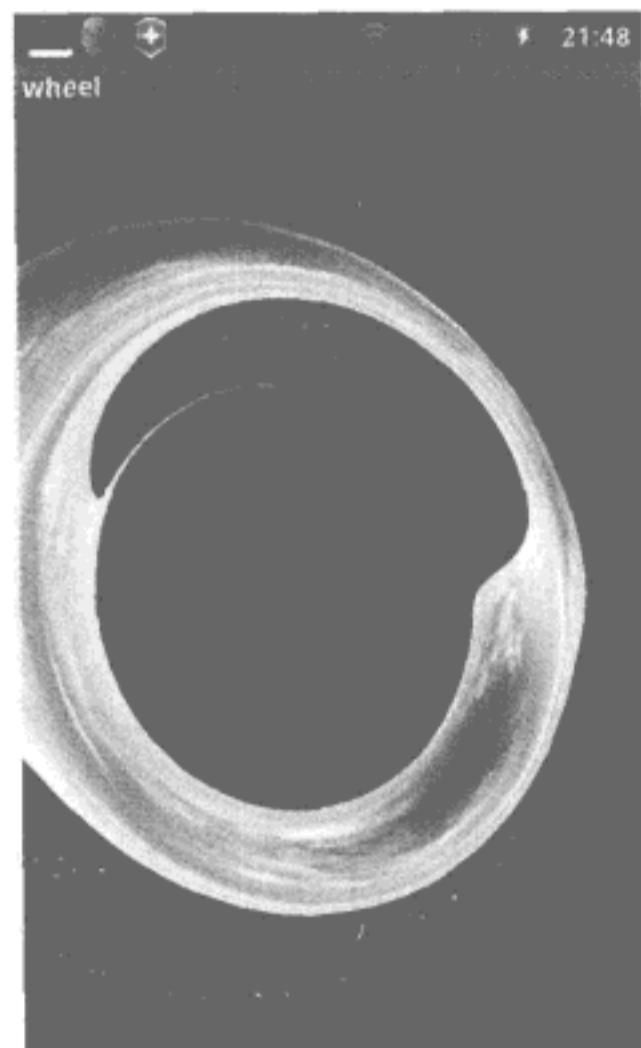
    gl.glActiveTexture(GL10.GL_TEXTURE0);
    long time = SystemClock.uptimeMillis() % 4000L;
    float angle = 0.090f * ((int) time);
    // 旋转三角形
    gl.glRotatef(angle, 0, 0, 1.0f);
    // 绘制三角形
    mTriangle.draw(gl);

    // 为了不影响其他的 Renderer, 恢复到默认的状态
    gl glBindTexture(GL10.GL_TEXTURE_2D, 0);
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
}
```

## 20.7 旋转立体天空

工程目录: src\ch20\ch20\_rotate\_sky

本小节的例子的效果看上去会很炫，一个平面的天空背景被处理成立体圆形，像个游泳圈，并且不断在屏幕上旋转，效果如图 20.7 所示。



▲ 图 20.7 旋转立体天空

与前面的例子一样，处理旋转效果的方法仍然是 `onDrawFrame`，代码如下：

```
public void onDrawFrame(GL10 gl)
{
    gl.glClearColor(0, 0, 0, 0);
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    gl.glEnable(GL10.GL_DEPTH_TEST);
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();

    GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
    // 旋转图形
    gl.glRotatef(mAngle, 0, 1, 0);
    gl.glRotatef(mAngle * 0.25f, 1, 0, 0);

    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

    if (mContextSupportsCubeMap)
    {
        gl.glActiveTexture(GL10.GL_TEXTURE0);

        gl.glEnable(GL11ExtensionPack.GL_TEXTURE_CUBE_MAP);
        // 绑定立体天空效果的纹理
        gl glBindTexture(GL11ExtensionPack.GL_TEXTURE_CUBE_MAP,
                        mCubeMapTextureID);
        GL11ExtensionPack gl11ep = (GL11ExtensionPack) gl;
        gl11ep.glTexGeni(GL11ExtensionPack.GL_TEXTURE_GEN_STR,
                        GL11ExtensionPack.GL_TEXTURE_GEN_MODE,
```

```
GL11ExtensionPack.GL_REFLECTION_MAP);

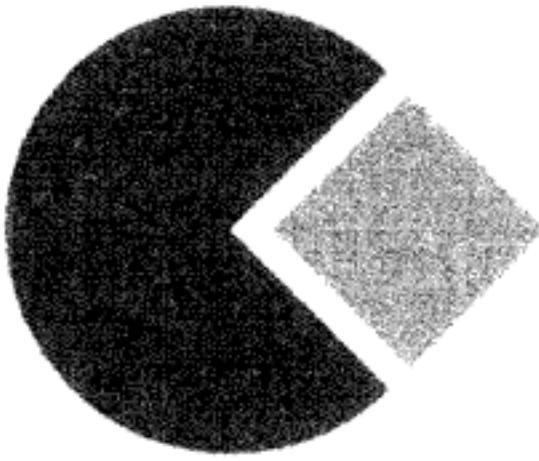
gl.glEnable(GL11ExtensionPack.GL_TEXTURE_GEN_STR);

gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
              GL10.GL_DECAL);
}

// 绘制立体天空
mGrid.draw(gl);
if (mContextSupportsCubeMap) {
    gl.glDisable(GL11ExtensionPack.GL_TEXTURE_GEN_STR);
}
mAngle += 1.2f;
}
```

## 20.8 小结

本章介绍了几个用 OpenGL ES 实现的特效，这些特效对于 OpenGL ES 来说只是小菜一碟，读者可以从 Android Market 下载一些 3D 游戏来体会一下 OpenGL ES 的强大。本章的例子只给出了核心代码，完整的实现请读者参阅随书光盘中的源代码。



## 第 21 章 Android NDK 编程

前面的章节一直在讲如何用 Java 来编写 Android 应用程序，自从 Android SDK 1.5 开始，Google 公司就发布了 Android NDK。通过 NDK，开发人员可以使用 C/C++ 开发 Android 应用程序的部分功能。本章将详细介绍 Android NDK 的下载、安装和配置，以及如何将 NDK 和 SDK 结合起来开发 Android 应用程序。

### 21.1 Android NDK 简介

Android NDK (Native Development Kit) 是一套允许开发人员将本地代码嵌入 Android 应用程序的开发包。众所周知，Android 应用程序运行在 Dalvik 虚拟机上，而 NDK 允许开发人员将 Android 应用程序中的部分功能（由于 NDK 只开放了部分接口，因此，无法使用 NDK 编写完整的 Android 应用程序）用 C/C++ 语言来实现，并将这部分 C/C++ 代码编译成可直接运行在 Android 平台上的本地代码（也就是绕过 Dalvik 虚拟机，直接在 Android 平台上运行）。这些本地代码以动态链接库 (.so 文件) 的形式存在。NDK 的这个特性既有利于代码的重用（动态链接库可以被多个 Android 应用程序使用），也可以在某种程度上提高程序的运行效率。

NDK 可以提供以下内容：

- 提供了一套工具集，这套工具集可以将 C/C++ 源代码生成本地代码。
- 用于定义 NDK 接口的 C 头文件 (\*.h) 和实现这些接口的库文件。
- 一套编译系统。可以通过非常少的配置生成目标文件。
- 自从 Android 2.3 开始，支持本地 Activity (Native Activity)。

最新版的 Android NDK 支持 ARMv5TE 机器指令，并且提供大量的 C 语言库，包括 libm (Math 库)、OpenGL ES、JNI 接口以及其他库。

虽然在程序中使用 NDK 可以大大提高运行速度，但使用 NDK 也会带来很多副作用。例如，使用 NDK 并不是总会提高应用程序的性能，但却会 100% 增加程序的复杂度。而且使用 NDK 必须要自己控制内存的分配和释放，这样将无法利用 Dalvik 虚拟机来管理内存，也会给应用程序带来很大的风险。因此，笔者建议应根据具体的情况适度使用 NDK。例如，需要大幅度提高程序性能或需要保密（因为 Java 生成的目标文件很容易被反编译）的情况下就可以使用 NDK 来生成相应的本地代码。

## 21.2 安装、配置和测试 NDK 开发环境

Android NDK 的安装相比 Android SDK 要稍微复杂一些，除了要安装 NDK 外，还需要安装 C/C++运行环境，并进行相应的配置。本节将详细介绍配置 NDK 和 C/C++运行环境的过程。

### 21.2.1 系统和软件要求

本小节将介绍一下 Android NDK 支持的编译器和 Android SDK 的版本，以及兼容的操作系统平台。

#### Android SDK

- 在使用 Android NDK 之前必须安装 Android SDK。
- Android SDK 必须是 1.5 及以上版本。Android SDK1.0 和 1.1 不支持 Android NDK。
- 要在 Android NDK 中使用 OpenGL ES 1.1，Android SDK 要求 1.6 及以上版本。如果使用 OpenGL ES 2.0，Android SDK 要求 2.0 及以上版本。

#### 支持的操作系统

- Windows XP (32 位)。
- Windows Vista (32 或 64 位)。
- Mac OS X 10.4.8 及以上版本 (仅支持 x86 Mac OS)。
- Linux (32 或 64 位)。

#### 开发工具

- Make 工具要求 GNU Make 3.81 及以上版本。低版本的 Make 有可能也支持，但 Google 公司的 Android NDK 开发团队并未对低版本的 Make 进行测试。
- 对于 Windows 平台，要求 Cygwin 1.7 或更高版本。

### 21.2.2 下载和安装 Android NDK

可以从如下地址下载 Android NDK 的最新版本：

| <http://developer.android.com/intl/zh-CN/sdk/index.html>

在写作本书时，Android NDK 的最高版本是 Android NDK r5b，读者可以根据自己使用的操作系统下载相应的 Android NDK。下载后，将 Android NDK 的压缩包解压缩，目录结构如图 21.1 所示。

### 21.2.3 下载和安装 Cygwin

如果读者在 Windows 下使用 Android NDK，则需要下载 Cygwin。当然，如果读者在其他操作系统下使用 Android NDK 则不需要进行这一步。

## Android 开发权威指南



▲ 图 21.1 Android NDK 的目录结构

Cygwin 是一套在 Windows 下模拟 Linux 编译环境的工具集，包括如下两部分。

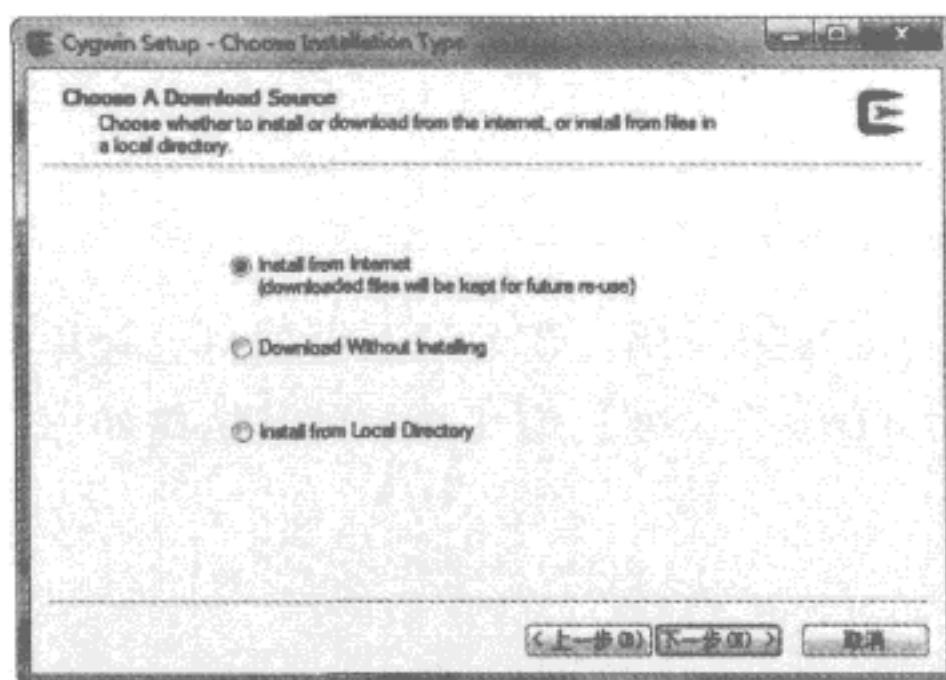
- 一个 cygwin1.dll 文件。该文件模拟了真实的 Linux API，是一个 API 模拟层。开发人员可以将基于 Linux 的 C/C++源代码在 Cygwin 中进行编译，在编译的过程中，如果 C/C++源代码中调用了 Linux 中的 API，Cygwin 就会利用 cygwin1.dll 来编译 C/C++源代码，从而可以在 Windows 中生成 Linux 下的.so 文件。
- 模拟 Linux 编译环境的工具集。

读者可以从如下地址下载 Cygwin 的最新版本：

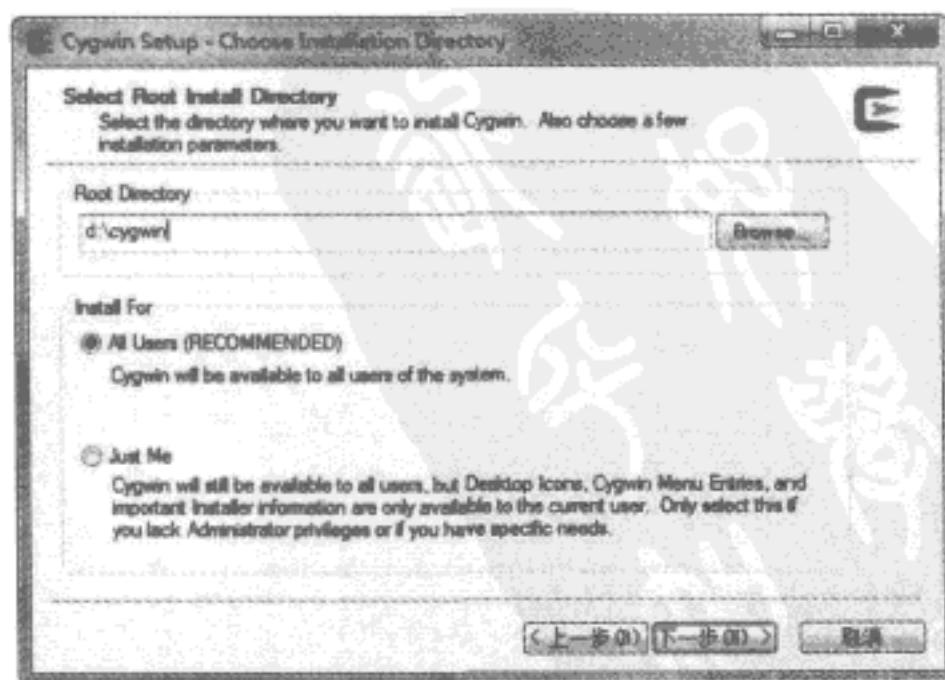
| <http://www.cygwin.com>

在写作本书时，Cygwin 的最新版本是 1.7.5。

由于完整的 Cygwin 安装包很大，因此，Cygwin 只提供了一个在线安装程序进行下载，在安装过程中需要稳定快速的互联网环境。安装文件只有一个 setup.exe，运行该程序，选中如图 21.2 所示的第一个选项，然后进入下一个设置界面，默认情况下 Cygwin 的安装目录是 C:\cygwin，也可以选择其他安装目录，如图 21.3 所示。

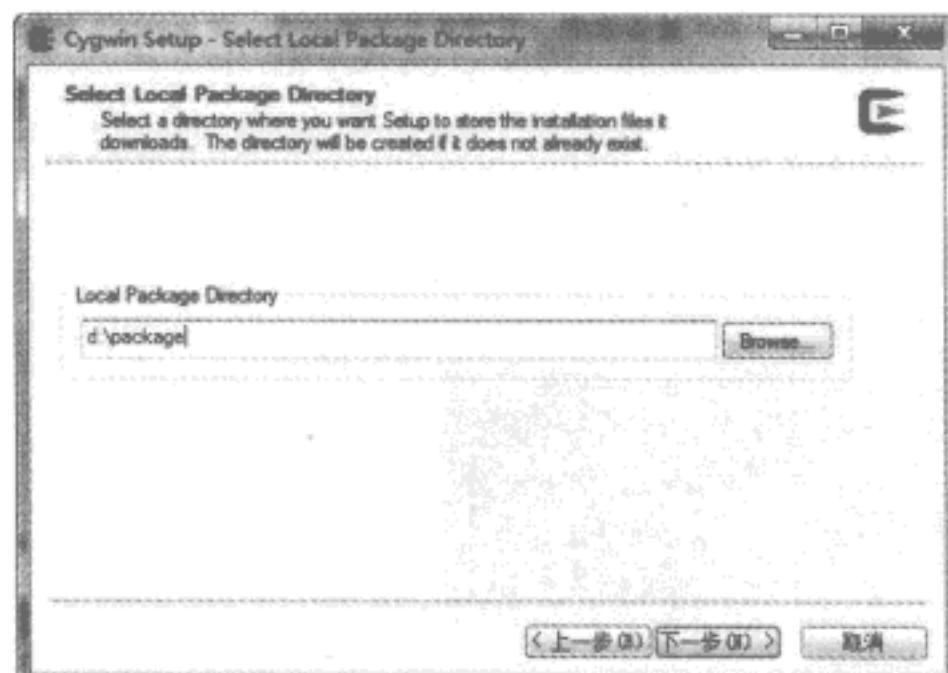


▲ 图 21.2 选择 Cygwin 的安装方式

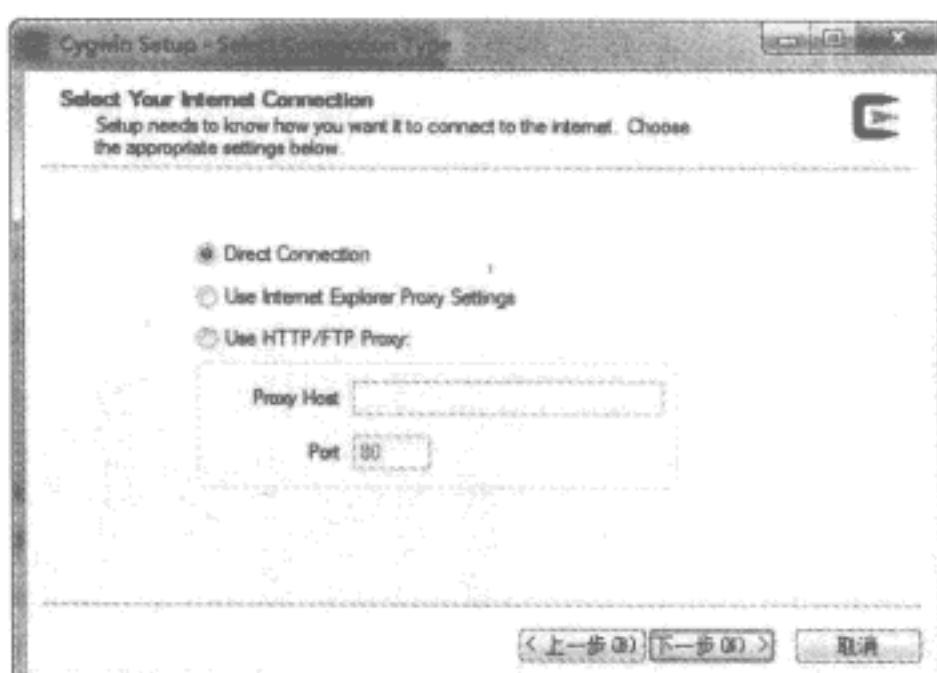


▲ 图 21.3 设置 Cygwin 的安装目录

进入下一个设置界面后，需要指定一个 Cygwin 下载临时目录，如图 21.4 所示。该目录可以任意设置，在安装完 Cygwin 后，可以将该目录删除。然后设置网络连接方式，如图 21.5 所示。



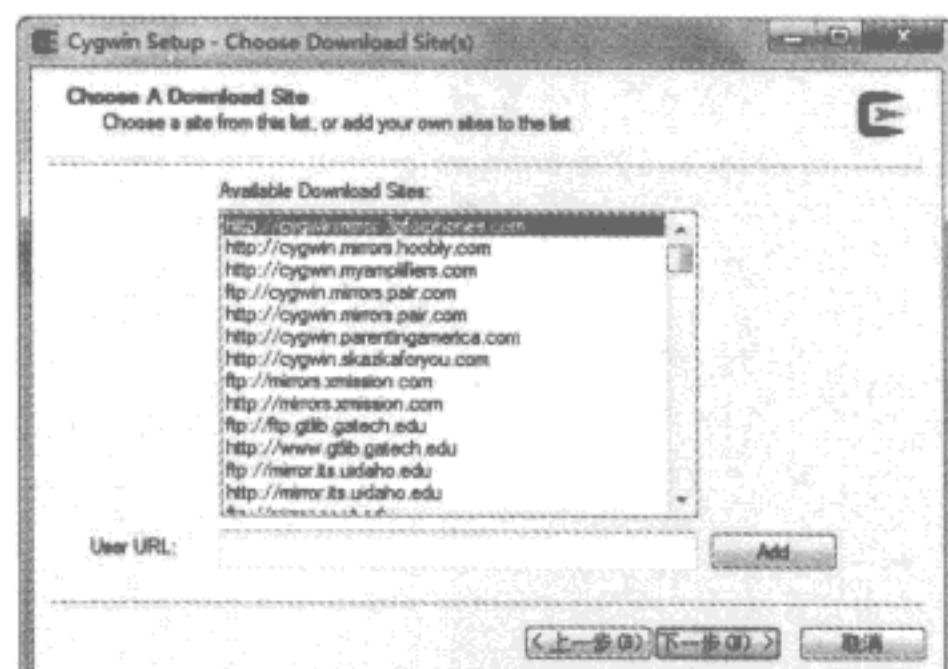
▲ 图 21.4 设置 Cygwin 的下载临时目录



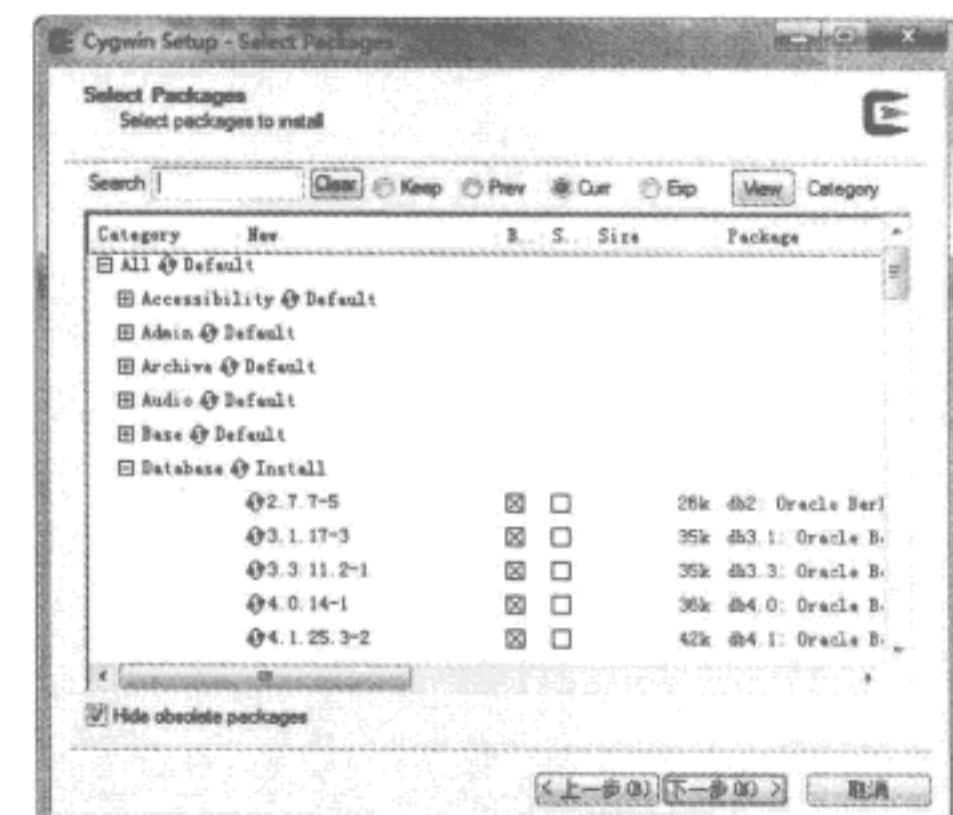
▲ 图 21.5 选择网络连接方式

进入下一个设置界面后，读者可以根据自己所在的地区选择一个速度较快的下载地址，如图 21.6 所示。

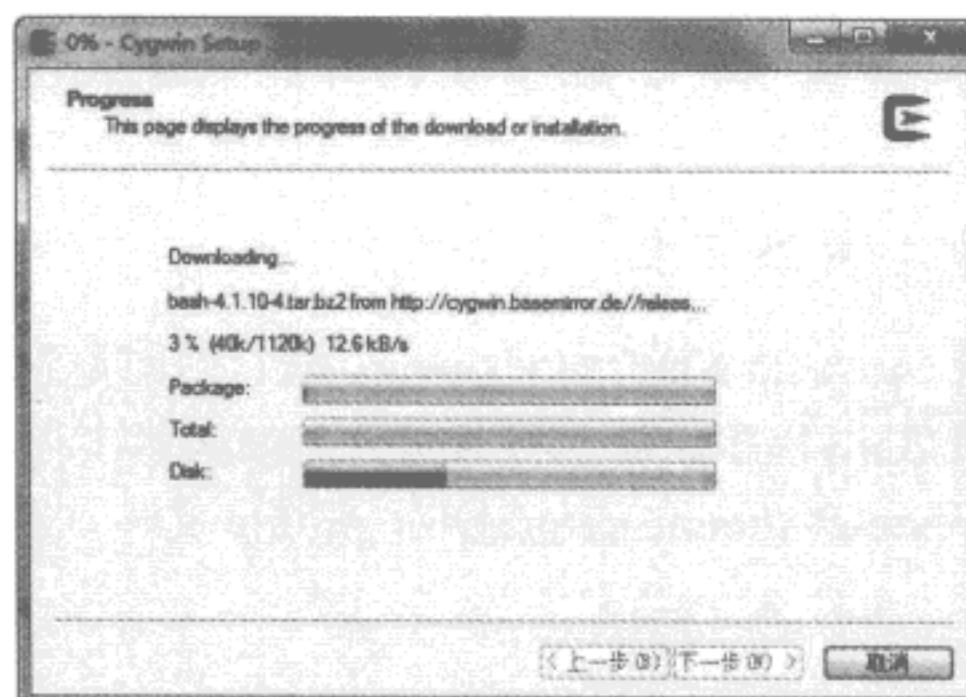
选择下载地址后，单击“下一步”按钮进入选择安装包界面，如图 21.7 所示。选择相应的安装包后，单击“下一步”按钮开始正式下载和安装。安装进度界面如图 21.8 所示。



▲ 图 21.6 选择下载网址



▲ 图 21.7 选择安装包



▲ 图 21.8 安装界面

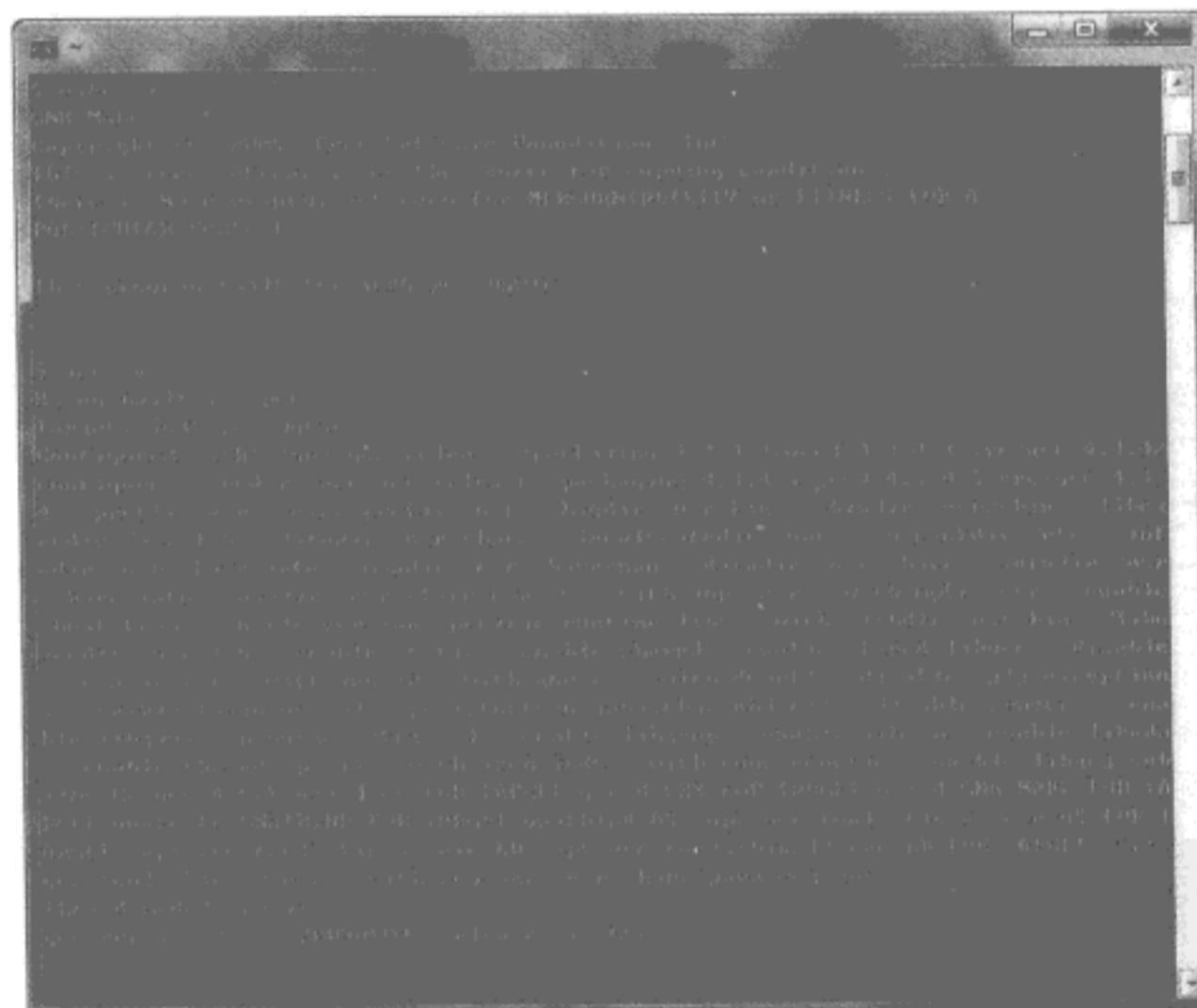
## Android 开发权威指南

安装 Cygwin 后，通过桌面上的 Cygwin 图标或 Cygwin 安装根目录中的 Cygwin.bat 文件可启动 Cygwin。Cygwin 是一个类似 Linux 控制台的程序，可以在 Cygwin 控制台中输入 Linux 命令，界面如图 21.9 所示。



▲ 图 21.9 Cygwin 控制台

下面来验证一下 make 和 gcc(C 语言编译器)的版本。在 Cygwin 控制台中输入 make -v 和 gcc-v 命令，会输出如图 21.10 所示的信息，其中 GNU Make 的版本是 3.81，gcc 的版本是 4.3.4，完全满足开发环境的最低要求。下一小节将介绍如何将 Cygwin 和 Android NDK 连接起来组成一个完整的开发环境。



▲ 图 21.10 验证 make 和 gcc 的版本

### 21.2.4 配置 Android NDK 的开发环境

虽然在前两个小节安装了 Android NDK 和 Cygwin，但它们都是独立的环境，要想使用 Cygwin 来编译基于 Android NDK 的 C/C++ 程序还需要将 Android NDK 和 Cygwin 进行整合。

打开<Cygwin 安装目录>\home\Administrator\ bash\_profile 文件，并在该文件中添加如下内容：

```
ANDROID_NDK_ROOT=/cygdrive/d/sdk/android-ndk-r5b
export ANDROID_NDK_ROOT
```

其中第 1 行设置了 Android NDK 的本地路径。要注意的是，路径前面必须以“/cygdrive/”开

头。Android NDK 的安装目录是 d:\sdk\android-ndk-r5b，在设置 Android NDK 的本地路径时应将路径改成“d:/sdk/android-ndk-r5b”。

如果读者使用过 Linux 或 UNIX，应该对 export 很了解。这个 Shell 命令用于导出环境变量，也就是 ANDROID\_NDK\_ROOT，这一点也与 Windows 不同。在 Windows 中，环境变量只要设置了就可以直接使用。而在 Linux/UNIX 下，必须使用 export 命令导出环境变量才可以使用。

.bash\_profile 文件所在的目录（<Cygwin 安装目录>\home\Administrator）也是 Cygwin 的根目录，也就是 Cygwin 控制台一开始进入的目录。

## 21.3 第一个 NDK 程序：世界你好

工程目录：src\ch21\ch21\_jni\_helloworld

本例实现了一个简单的 NDK 程序。NDK 程序 (.so 库文件) 通过 stringFromJNI 方法返回一个“世界你好”的字符串，并将该字符串显示在 TextView 控件上。本节主要讨论了如下内容。

- NDK 程序（用 C 语言编写）；
- 编译 NDK 程序；
- 调用 NDK 程序；
- 集成 Eclipse 和 NDK 开发环境。

### 21.3.1 编写和调用 NDK 程序

编写 NDK 程序的步骤如下。

- (1) 创建一个 Eclipse Android 工程，工程名为 ch21\_jni\_helloworld。
- (2) 建立一个调用 NDK 程序的 Java 类，代码如下：

```
package ch21.jni.helloworld;
import android.app.Activity;
import android.widget.TextView;
import android.os.Bundle;

public class HelloWorldJni extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        // 调用 stringFromJNI 方法，并将返回值显示在 TextView 控件中
        tv.setText(stringFromJNI());
        setContentView(tv);
    }
    // 定义 JNI 方法，也可称为本地方法
    public native String stringFromJNI();

    static
    {
```

```

    // 装载.so 库文件
    System.loadLibrary("hello-jni");
}
}

```

调用 JNI 方法与调用 Java 方法类似，但要使用 `native` 关键字来定义本地方法，并在 `static` 块中使用 `System.loadLibrary` 方法装载 NDK 库文件（.so 文件）。

（3）在 ch21\_jni\_helloworld 工程中建立一个 `jni` 目录。

（4）在 `jni` 目录中建立一个 `hello-jni.c` 文件，并输入如下的代码。

```

#include <string.h>
#include <jni.h>

jstring Java_ch21_jni_helloworld_HelloWorldJni_stringFromJNI( JNIEnv* env,
                                                               jobject thiz )
{
    /* 返回一个字符串，该字符串会显示在 TextView 控件上 */
    return (*env)->NewStringUTF(env, "世界你好");
}

```

上面代码中的函数从表面看只是一个普通的 C 语言函数，但这个函数和普通的 C 语言函数有如下几点不同。

- 该函数的返回值类型和参数类型都是在 JNI 的头文件中定义的类型（如 `jstring`、`jobject` 等），这些类型与 Java 中的数据类型对应，例如，`jstring` 对应 Java 中的 `String`，`jobject` 对应 Java 中的 `Object`。在定义被 Java 调用的 JNI 函数时必须使用这些数据类型，否则 Java 无法成功调用这些函数。

- 从 `HelloWorldJni` 类的代码可以看出，调用的方法是 `stringFromJNI`，而 `hello-jni.c` 中的方法却是 `Java_ch21_jni_helloworld_HelloWorldJni_stringFromJNI`。实际上，这里涉及一个命名规则，JNI 方法名的命名规则是 `Java_xxx_MethodName`，从这个命名规则可以看出，JNI 方法名分为三段，中间用下划线“\_”分割。第一段是“Java”，这是固定的，最后一段就是在 Java 中调用的实际方法名，而中间一段“xxx”实际上就是调用 JNI 方法的类的全名（package + classname），只是将中间的点“.”换成了下划线“\_”，所以最终的 JNI 方法名是 `Java_ch21_jni_helloworld_HelloWorldJni_stringFromJNI`。

- 上面的函数有两个参数：`env` 和 `thiz`。这两个参数必须包含在 JNI 函数中，而且必须是头两个参数，其中 `env` 表示 JNI 的调用环境，`thiz` 表示定义 `native` 方法的 Java 类的对象本身。

（5）在 `jni` 目录中建立一个 `Android.mk` 文件，并输入如下内容。

```

LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE      := hello-jni
LOCAL_SRC_FILES  := hello-jni.c

include $(BUILD_SHARED_LIBRARY)

```

其中 `my-dir` 是系统变量，表示当前目录。`LOCAL_PATH` 变量必须出现在 `Android.mk` 文件的最开始位置，用来指定 C/C++ 源文件（在本例中是 `hello-jni.c` 文件）的位置。`LOCAL_MODULE` 变量

用于指定生成的.so 文件名，例如，本例中 LOCAL\_MODULE 变量的值是 hello-jni，生成的.so 文件就是 libhello-jni.so。LOCAL\_SRC\_FILES 变量用于指定待编译的 C/C++ 源文件。

### 21.3.2 用命令行方式编译 NDK 程序

上一小节编写的 NDK 程序还不能在手机上运行，原因是还没有生成.so 文件。本节将介绍如何通过命令行方式将 NDK 程序编译生成.so 文件。编译 NDK 程序的步骤如下。

- (1) 在<Android NDK 安装目录>建立一个 apps 目录，并在该目录下建立一个 hello-jni 目录。
- (2) 将 ch21\_jni\_helloworld 工程中的 jni 目录复制到<Android NDK 安装目录>\apps\hello-jni 目录中。
- (3) 在<Android NDK 安装目录>\apps\hello-jni 目录中建立一个 Application.mk 文件，并输入如下内容。

```
APP_PROJECT_PATH := $(call my-dir)
APP_MODULES      := hello-jni
```

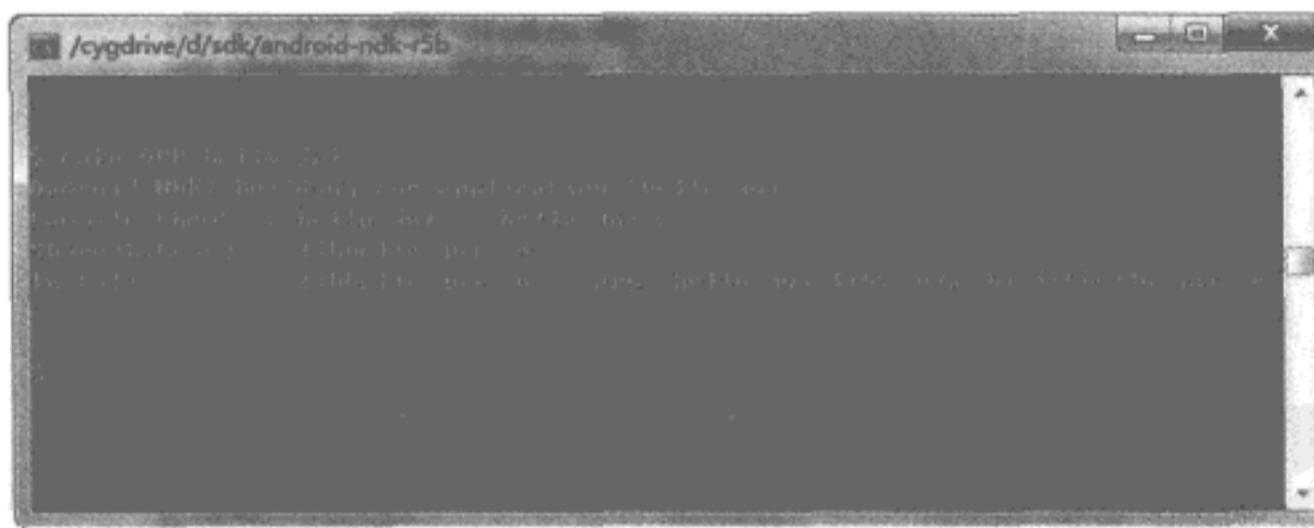
- (4) 进入 Cygwin 控制台，输入如下的命令。

```
cd $ANDROID_NDK_ROOT
```

- (5) 在 Cygwin 控制台中输入如下的命令编译 NDK 程序。

```
make APP=hello-jni
```

如果在 Cygwin 控制台输出如图 21.11 所示的信息，说明 NDK 程序已成功编译。



▲ 图 21.11 成功编译 NDK 程序

- (6) 进入<Android NDK 安装目录>\apps\hello-jni 目录，我们会看到两个目录：jni 和 libs。其中 jni 是源代码所有的目录，libs 就是编译 NDK 程序生成的目录。在<Android NDK 安装目录>\apps\hello-jni\libs\armeabi 目录中会看到一个 libhello-jni.so 文件，将 libs 目录直接复制到 ch21\_jni\_helloworld 目录中即可。现在运行程序，会输出如图 21.12 所示的信息。



▲ 图 21.12 成功调用 JNI 方法

### 21.3.3 在 Eclipse 中集成 Android NDK

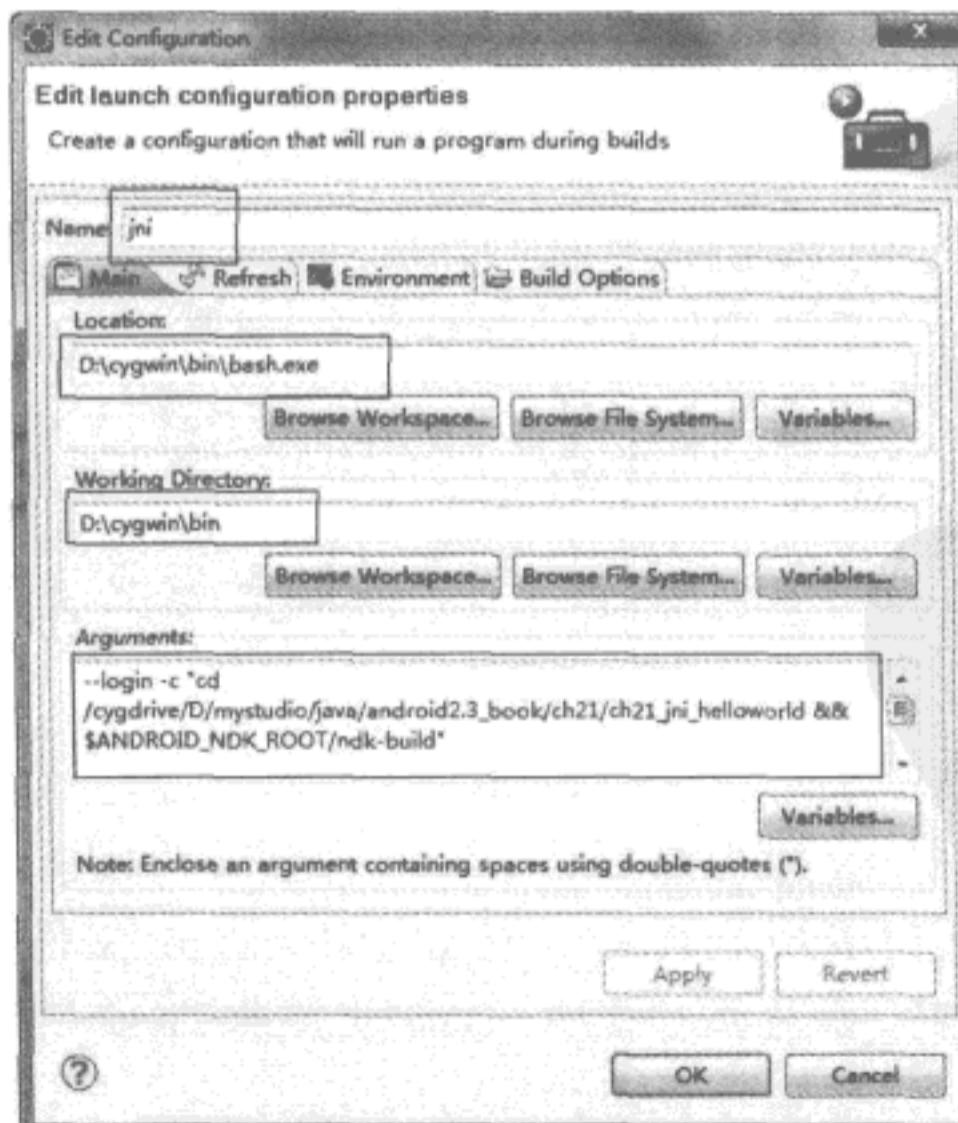
虽然使用 21.3.2 节的方法编译 NDK 程序完全可以满足开发的要求，但可能大多数读者都会很惊讶地说声“太麻烦了”。为了生成一个.so 文件，需要将目录、文件复制来复制去，太影响开发效率了。不过这一点读者大可放心，Google 公司既然提供了 Android NDK，就不会让开发人员使用这么“愚蠢”的方法来开发 NDK 程序。在 Eclipse 中只需要简单的配置，就可以将 NDK 集成到 Eclipse 中。当修改 C/C++ 源代码时，运行 Android 程序，NDK 会首先编译 C/C++ 源代码，然后再生成 apk 文件，并将 apk 文件安装在手机或模拟器上，这样 C/C++ 程序就可以和 Java 程序一样在 Eclipse 中修改、编译和运行了。在 Eclipse 中集成 NDK 的步骤如下。

(1) 在 Android 工程右键菜单中单击“Properties”菜单项，在打开的“Properties”对话框中选择左侧列表中的“Builders”列表项。单击右侧的“New”按钮，在打开的“Choose configuration type”对话框中选择“Program”列表项，单击“OK”按钮，会打开“Edit Configuration”对话框。

(2) 在“Name”文本框中输入“jni”(也可以是其他名称)。

(3) 进入“Main”选项卡，需要输入的文本框是“Location”、“Working Directory”和“Arguments”，内容分别为如下 3 个字符串。设置效果如图 21.13 所示。

- D:\cygwin\bin\bash.exe。
- D:\cygwin\bin。
- --login -c "cd /cygdrive/D/mystudio/java/android2.3\_book/ch21/ch21\_jni\_helloworld && \$ANDROID\_NDK\_ROOT/ndk-build"。

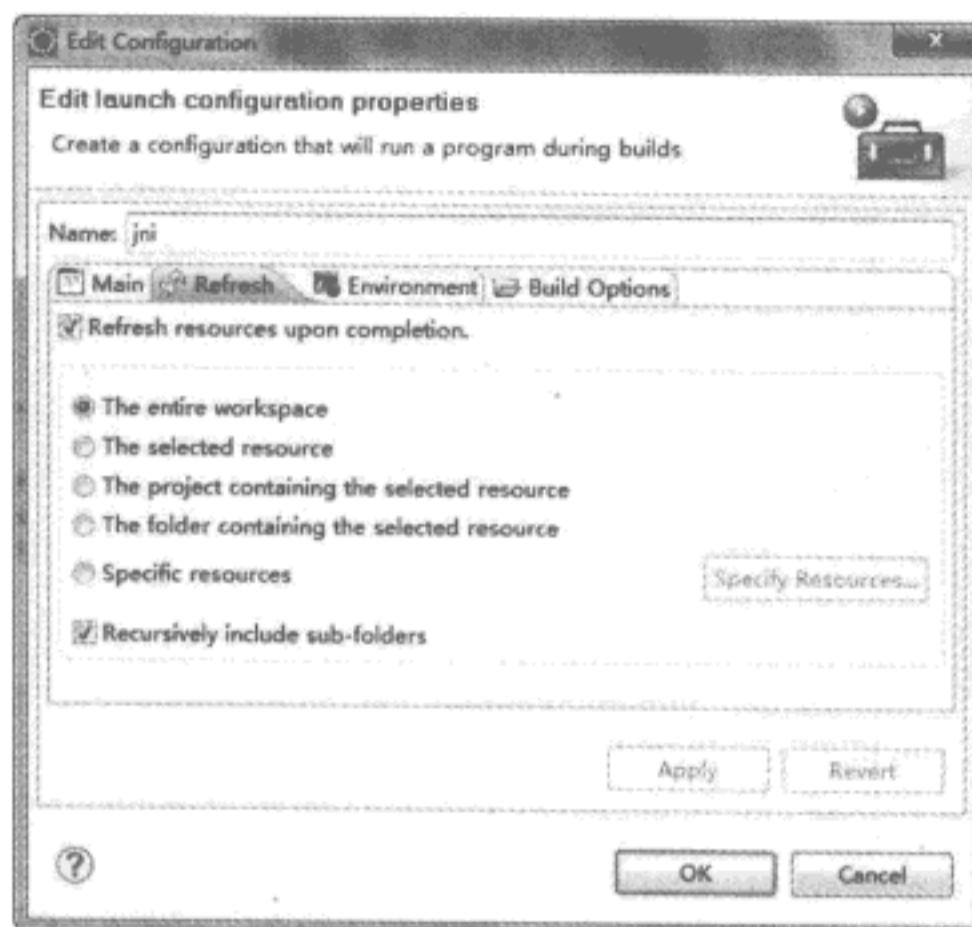


▲ 图 21.13 设置“Main”选项卡

设置“Main”选项卡时需要指定 Cygwin 的 bin 路径以及 bash.exe 文件的路径。如果读者将 cygwin 安装在了其他的目录，需要将“D:\cygwin”换成读者机器上 cygwin 的安装目录。Arguments 文本框中

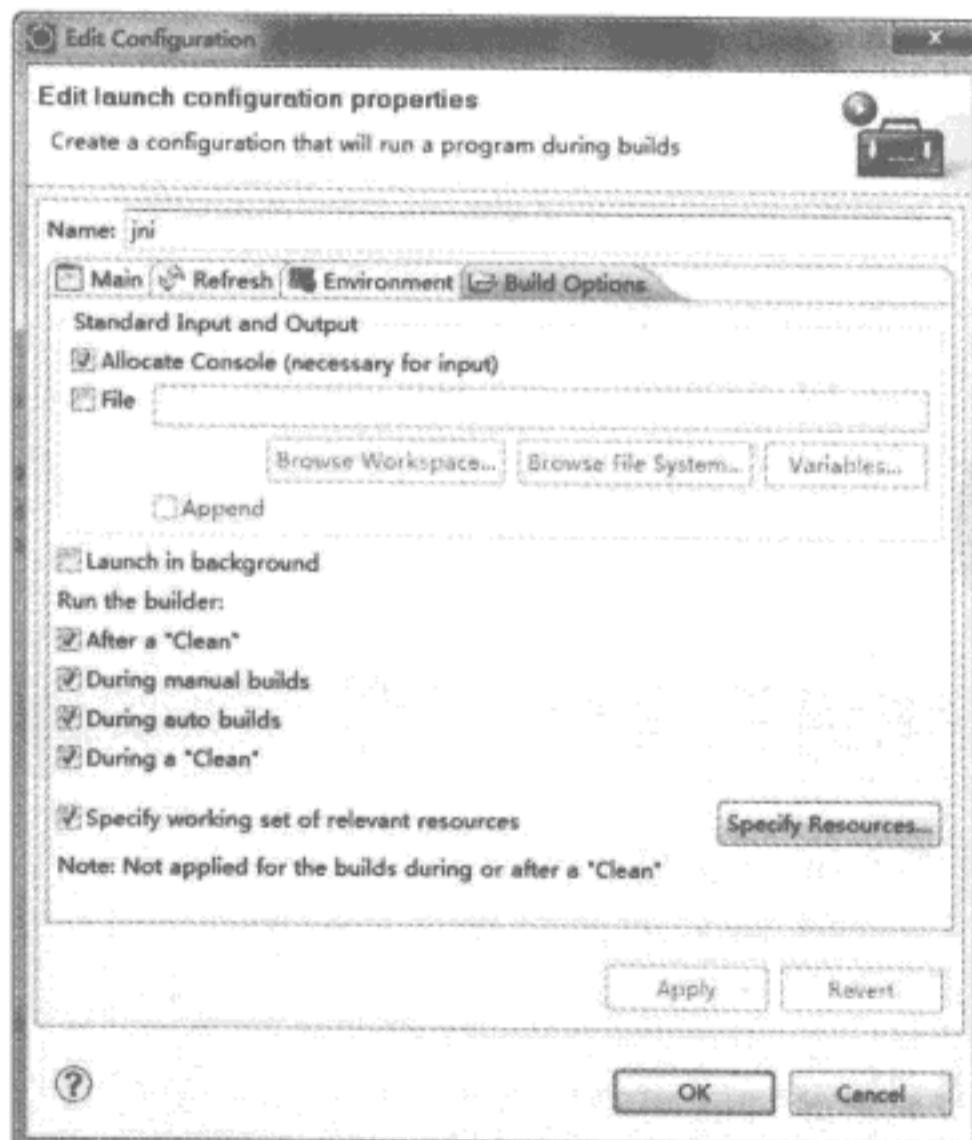
需要输入要执行的命令，其中“D/mystudio/java/android2.3\_book/ch21/ch21\_jni\_helloworld”是 ch21\_jni\_helloworld 工程的路径，读者需要将该路径换成自己机器上 ch21\_jni\_helloworld 工程的路径。

(4) 进入“Refresh”选项卡，选中“Refresh resources upon completion”复选框，如图 21.14 所示。



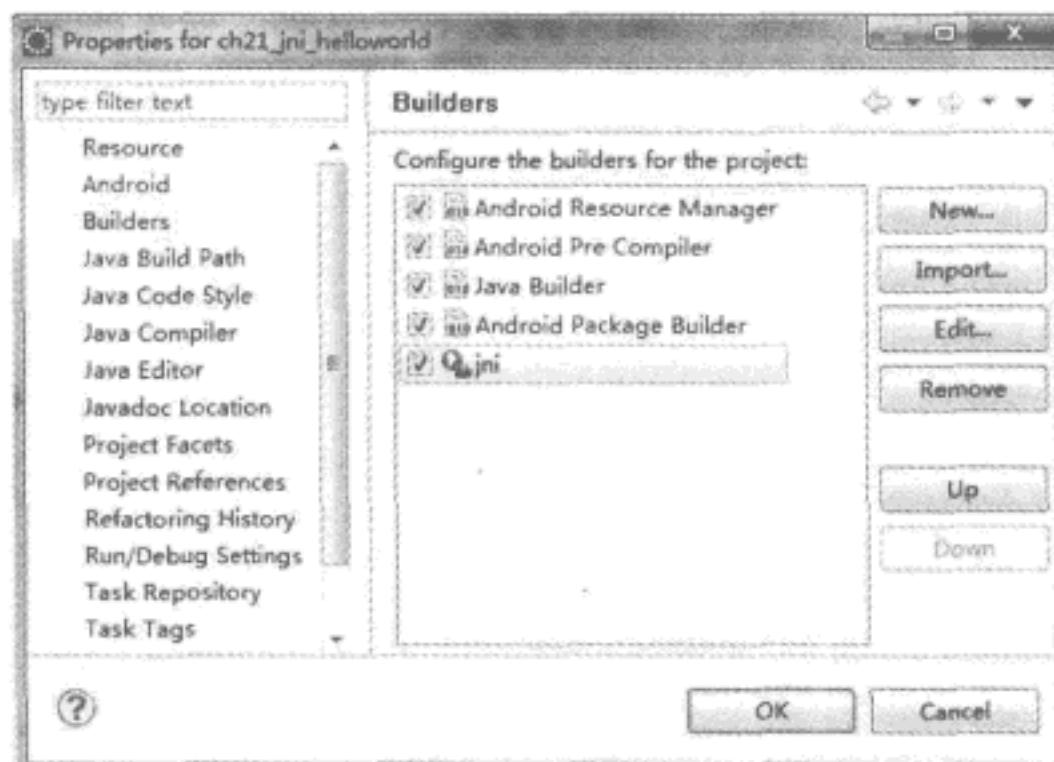
▲ 图 21.14 设置“Refresh”选项卡

(5) 进入“Build Options”选项卡，按如图 21.15 所示进行设置。在选择“Specify working set of relevant resources”复选框时，需要单击右下角的“Specify Resources”按钮指定 Android 工程的 jni 目录（该目录包含了 C/C++源代码和 Android.mk 文件）。



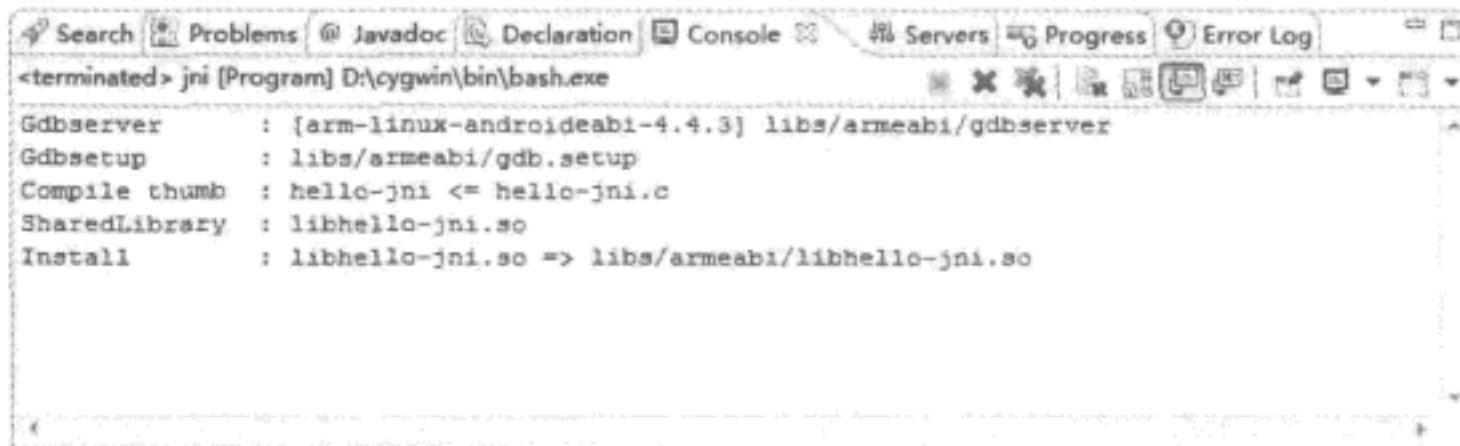
▲ 图 21.15 设置“Build Options”选项卡

(6) 设置完“Edit Configuration”对话框后，单击“OK”按钮关闭该对话框，我们会在右侧的“Builders”列表中看到一个“jni”列表项，如图 21.16 所示。



▲ 图 21.16 新建了一个 Builder 项

(7) 单击“OK”按钮关闭“Properties”对话框，然后运行 ch21\_jni\_helloworld 工程，ADT 首先会编译 NDK 程序，并在“Console”视图中输出如图 21.17 所示的编译信息，然后会编译 Android 程序，并在手机或模拟器上安装并运行程序。



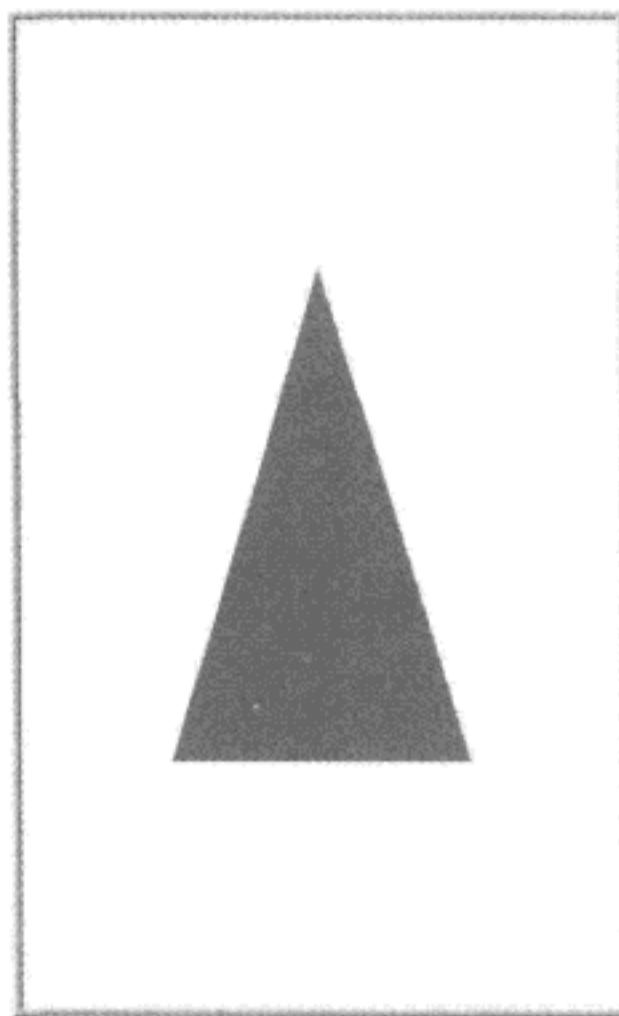
▲ 图 21.17 编译 NDK 程序的日志

(8) 现在来修改 jni 目录中的 hello-jni.c 文件，例如，改变返回的字符串，然后再次运行程序，会看到 ADT 会先编译 NDK 程序（仍会输出如图 21.17 所示的日志信息），在手机或模拟器上运行程序后，会看到输出字符串已经改变了。

## 21.4 背景不断变化的三角形（NDK 版 OpenGL ES）

工程目录：src\ch21\ch21\_jni\_opengl\_es

Android NDK 不仅可以使用基本的 C/C++ 语法来编写程序，也可以调用很多高级功能，例如，OpenGL ES 就是 NDK 的功能之一。虽然可以使用 Java 代码通过 Android SDK 提供的 OpenGL ES API 调用 OpenGL ES，但对于复杂的 3D 游戏，使用 C/C++ 来编写其核心代码几乎成为游戏开发者的共识。本例将利用 NDK 技术编写一个简单的 OpenGL ES 程序。该程序在屏幕中间绘制一个绿色三角形，并且背景不断变化，如图 21.18 所示。



▲ 图 21.18 背景不断变化的三角形

本例基本框架的搭建与 21.3 节的例子类似，配置完 ch21\_jni\_opengl\_es 工程后，编写一个 MyRenderer 类，代码如下：

```
package mobile.android.ch21.jni.opengl.es;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView;

public class MyRenderer implements GLSurfaceView.Renderer
{
    public void onDrawFrame(GL10 gl)
    {
        // 调用 JNI 方法
        GL2JNILib.step();
    }
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        // 调用 JNI 方法
        GL2JNILib.init(width, height);
    }
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
    }
}
```

从第 19 章和第 20 章的例子可知，继承自 GLSurfaceView.Renderer 的类是 OpenGL ES 的核心类，主要负责初始化 OpenGL ES，以及绘制 OpenGL ES 图形。但从上面代码看出，onDrawFrame 和 onSurfaceChanged 方法中只有一行代码，分别调用了 step 和 init 方法。这两个方法实际上是在 GL2JNILib 类中定义的，代码如下：

```

package mobile.android.ch21.jni.opengl.es;

public class GL2JNILib
{
    static
    {
        System.loadLibrary("gl2jni");
    }
    public static native void init(int width, int height);
    public static native void step();
}

```

在 jni 目录中建立一个 gl\_code.cpp 文件。init 和 step 方法需要在 gl\_code.cpp 中实现，代码如下：

```

/* step 方法的原型 */
JNIEXPORT void JNICALL Java_mobile_android_ch21_jni_opengl_es_GL2JNILib_init(JNIEnv *
env, jobject obj, jint width, jint height)
{
    setupGraphics(width, height);
}

/* step 方法的原型 */
JNIEXPORT void JNICALL Java_mobile_android_ch21_jni_opengl_es_GL2JNILib_step(JNIEnv *
env, jobject obj)
{
    renderFrame();
}

```

上面两个方法分别调用了一个用 C++ 语言实现的方法，其中 setupGraphic 方法用于初始化 OpenGL ES，代码如下：

```

bool setupGraphics(int w, int h) {
    printGLString("Version", GL_VERSION);
    printGLString("Vendor", GL_VENDOR);
    printGLString("Renderer", GL_RENDERER);
    printGLString("Extensions", GL_EXTENSIONS);
    LOGI("setupGraphics(%d, %d)", w, h);
    gProgram = createProgram(gVertexShader, gFragmentShader);
    if (!gProgram) {
        LOGE("Could not create program.");
        return false;
    }
    gvPositionHandle = glGetUniformLocation(gProgram, "vPosition");
    checkGlError("glGetAttribLocation");
    LOGI("glGetAttribLocation(\"vPosition\") = %d\n",
         gvPositionHandle);

    glViewport(0, 0, w, h);
    checkGlError("glViewport");
    return true;
}

```

renderFrame 方法用于绘制 OpenGL ES 图形，代码如下：

```

void renderFrame() {
    static float grey;
    /* 背景颜色值逐渐递增 */
    grey += 0.01f;
    /* 如果背景值颜色超过 1.0f, 恢复到 0.0f */
    if (grey > 1.0f) {
        grey = 0.0f;
    }
    /* 设置背景色 */
    glClearColor(grey, grey, grey, 1.0f);
    checkGlError("glClearColor");
    glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
    checkGlError("glClear");

    glUseProgram(gProgram);
    checkGlError("glUseProgram");

    glVertexAttribPointer(gvPositionHandle, 2, GL_FLOAT, GL_FALSE, 0, gTriangleVertices);
    checkGlError("glVertexAttribPointer");
    glEnableVertexAttribArray(gvPositionHandle);
    checkGlError("	glEnableVertexAttribArray");
    /* 开始绘制三角形 */
    glDrawArrays(GL_TRIANGLES, 0, 3);
    checkGlError("glDrawArrays");
}

```

从上面的代码可以看出, C/C++版的 OpenGL ES API 与 Java 版的 OpenGL ES API 在方法(函数)名上基本是一样的, 例如, Java 中的 `GL10.glClearColor` 方法对应于 C/C++中的 `glClearColor` 函数, Java 中的 `GL10.glDrawArrays` 方法对应于 C/C++中的 `glDrawArrays` 函数。因此, Java 中的 OpenGL ES 经验也能应用在 NDK 版的 OpenGL ES 上。

最后需要在 jni 目录中建立一个 `Android.mk` 文件, 并输入如下的内容。

```

LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE      := libgl2jni
LOCAL_CFLAGS      := -Werror
LOCAL_SRC_FILES   := gl_code.cpp
LOCAL_LDLIBS       := -llog -lGLESv2

include $(BUILD_SHARED_LIBRARY)

```

## 21.5 使用 NDK OpenGL ES API 实现千变万化的 3D 效果

工程目录: `src\ch21\ch21_jni_san_angeles`

本节将实现一个更复杂的 3D 效果, 屏幕上会显示很多 3D 图形, 并千变万化(例如, 高塔、城堡等), 如 21.19 所示。



▲ 图 21.19 千变万化的 3D 图形

MyRenderer 类负责绘制 3D 图形，代码如下：

```
package mobile.android.ch21.jni.san.angeles;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;
import android.opengl.GLSurfaceView;

public class MyRenderer implements GLSurfaceView.Renderer
{
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        nativeInit();
    }

    public void onSurfaceChanged(GL10 gl, int w, int h)
    {
        nativeResize(w, h);
    }

    public void onDrawFrame(GL10 gl)
    {
        nativeRender();
    }

    // 声明 JNI 方法
    private static native void nativeInit();
    private static native void nativeResize(int w, int h);
    private static native void nativeRender();
}
```

MyRenderer 类中定义了 3 个 JNI 方法：nativeInit、nativeResize 和 nativeRender，分别用来初始化 OpenGL ES，处理画布改变尺寸时的动作以及绘制 3D 图形，这 3 个方法的实现如下。



## nativeInit 方法

```
void Java_mobile_android_ch21_jni_san_angeles_MyRenderer_nativeInit( JNIEnv* env )
{
    // 初始化 OpenGL ES, 源代码在 jni 目录的 importgl.c 文件中
    importGLInit();
    // 设置 OpenGL ES 的一些开关, 源代码在 jni 目录的 demo.c 文件中
    appInit();
    gAppAlive = 1;
    sDemoStopped = 0;
    sTimeOffsetInit = 0;
}
```

## nativeResize 方法

```
void Java_mobile_android_ch21_jni_san_angeles_MyRenderer_nativeResize( JNIEnv* env,
 jobject thiz, jint w, jint h )
{
    sWindowWidth = w;
    sWindowHeight = h;
    __android_log_print(ANDROID_LOG_INFO, "SanAngeles", "resize w=%d h=%d", w, h);
}
```

## nativeRender 方法

```
void Java_mobile_android_ch21_jni_san_angeles_MyRenderer_nativeRender( JNIEnv* env )
{
    long curTime;
    if (sDemoStopped) {
        curTime = sTimeStopped + sTimeOffset;
    } else {
        curTime = _getTime() + sTimeOffset;
        if (sTimeOffsetInit == 0) {
            sTimeOffsetInit = 1;
            sTimeOffset = -curTime;
            curTime = 0;
        }
    }
    // 绘制图形, 源代码在 jni 目录的 demo.c 文件中
    appRender(curTime, sWindowWidth, sWindowHeight);
}
```

最后在 jni 目录建立一个 Android.mk 文件，并输入如下内容。

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := sanangeles
LOCAL_CFLAGS := -DANDROID_NDK \
    -DDISABLE_IMPORTGL
LOCAL_SRC_FILES := \
    importgl.c \
    demo.c \
    app-android.c \
LOCAL_LDLIBS := -lGLESv1_CM -ldl -llog
```

```
| include $(BUILD_SHARED_LIBRARY)
```

## 21.6 使用 NDK 调用音频 API

工程目录: src\ch21\ch21\_jni\_native\_audio

本例通过 NDK 调用音频 API 播放内嵌在 apk 中的 MP3 音乐文件，以及录音与回放。启动程序，会显示如图 21.20 所示的界面，单击“播放内嵌 MP3”按钮会播放一段音乐，单击“录音”按钮开始录音，单击“回放”按钮播放录制的声音。



▲ 图 21.20 调用 NDK 音频 API

本例需要声明如下几个 JNI 方法。

```
// 下面两个方法完成一些初始化工作
public static native void createEngine();
public static native void createBufferQueueAudioPlayer();
// 打开MP3音频文件
public static native boolean createAssetAudioPlayer(
    AssetManager assetManager, String filename);
// 播放或停止MP3音频
public static native void setPlayingAssetAudioPlayer(boolean.isPlaying);
// 回放录音
public static native boolean playback();
// 初始化录音设备
public static native boolean createAudioRecorder();
// 开始录音
public static native void startRecording();
```

播放 MP3 音乐文件的代码如下：

```
// created 为 boolean 类型
if (!created)
{
    created = createAssetAudioPlayer(assetManager, "background.mp3");
}
if (created)
{
    isPlayingAsset = !isPlayingAsset;
    setPlayingAssetAudioPlayer(isPlayingAsset);
}
```

录音的代码如下：

```
// created 为 boolean 类型
if (!created)
{
    created = createAudioRecorder();
}
if (created)
{
    startRecording();
}
```

回放的代码如下：

```
playback();
```

由于 NDK 程序的代码较多，本节只给出播放 MP3 音频文件的代码。本例的详细实现请参阅随书光盘源代码。

```
void Java_mobile_android_ch21_jni_nativeaudio_NativeAudio_setPlayingAssetAudioPlayer
(JNIEnv* env,
     jclass clazz, jboolean.isPlaying)
{
    SLresult result;

    // 确认音频播放器是否被成功创建
    if (NULL != fdPlayerPlay) {

        // 设置播放器的状态（如果当前正在播放，则暂停播放，否则播放音乐
        result = (*fdPlayerPlay)->SetPlayState(fdPlayerPlay, isPlaying ?
            SL_PLAYSTATE_PLAYING : SL_PLAYSTATE_PAUSED);
        assert(SL_RESULT_SUCCESS == result);
    }
}
```

## 21.7 本地 Activity ( Native Activity )

工程目录：src\ch21\ch21\_jni\_native\_activity

从 Android 2.3 开始，允许直接使用 C/C++ 来编写 Activity，被称为 NativeActivity。如果使用 NativeActivity，这个 Activity 就会脱离 Dalvik 虚拟机的管理。由于本例只包含一个 Native Activity，因此，并不需要编写任何 Java 代码，只要在 jni\main.c 文件中编写 C/C++ 代码即可，例如，下面的代码是应用程序的入口，相当于 main 函数。

```
void android_main(struct android_app* state) {
    struct engine engine;

    app_dummy();

    memset(&engine, 0, sizeof(engine));
    state->userData = &engine;
    state->onAppCmd = engine_handle_cmd;
```

## Android 开发权威指南

```

state->onInputEvent = engine_handle_input;
engine.app = state;

// 准备加速传感器
engine.sensorManager = ASensorManager_getInstance();
engine.accelerometerSensor = ASensorManager_getDefaultSensor(engine.sensorManager,
    ASENSOR_TYPE_ACCELEROMETER);
engine.sensorEventQueue = ASensorManager_createEventQueue(engine.sensorManager,
    state->looper, LOOPER_ID_USER, NULL, NULL);
if (state->savedState != NULL) {
    engine.state = *(struct saved_state*)state->savedState;
}

// loop waiting for stuff to do.

while (1) {
    // 开始读队列中的事件
    int ident;
    int events;
    struct android_poll_source* source;

    while ((ident=ALooper_pollAll(engine.animating ? 0 : -1, NULL, &events,
        (void**)&source)) >= 0) {
        if (source != NULL) {
            source->process(state, source);
        }
        if (ident == LOOPER_ID_USER) {
            if (engine.accelerometerSensor != NULL) {
                ASensorEvent event;
                while (ASensorEventQueue_getEvents(engine.sensorEventQueue,
                    &event, 1) > 0) {
                    LOGI("accelerometer: x=%f y=%f z=%f",
                        event.acceleration.x, event.acceleration.y,
                        event.acceleration.z);

                    if (state->destroyRequested != 0) {
                        engine_term_display(&engine);
                        return;
                    }
                }
            }
            if (engine.animating) {
                engine.state.angle += .01f;
                if (engine.state.angle > 1) {
                    engine.state.angle = 0;
                }
                engine_draw_frame(&engine);
            }
        }
    }
}

```

Android.mk 文件的内容如下：

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)
LOCAL_MODULE    := native-activity
LOCAL_SRC_FILES := main.c
LOCAL_LDLIBS    := -llog -landroid -lEGL -lGLESv1_CM
LOCAL_STATIC_LIBRARIES := android_native_app_glue

include $(BUILD_SHARED_LIBRARY)

$(call import-module,android/native_app_glue)
```

Native Activity 也同样需要在 AndroidManifest.xml 文件中进行配置，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobile.android.ch21.jni.nativeactivity"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <application android:label="@string/app_name" android:hasCode="false">
        <!-- 配置 Native Activity -->
        <activity android:name="android.app.NativeActivity"
            android:label="@string/app_name"
            android:configChanges="orientation|keyboardHidden">
            <meta-data android:name="android.app.lib_name"
                android:value="native-activity" />
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

运行程序，会显示一个没有标题栏的 Activity，背景颜色会不断变化。

## 21.8 Android NDK 配置文件详解

在前面的部分了解到 NDK 程序有两个配置文件：Android.mk 和 Application.mk，其中 Android.mk 文件主要描述了如何编译 C/C++ 源代码，Application.mk 文件用于描述当前应用程序需要哪些模块。本节将详细介绍这两个文件中常用的变量和函数。

### 21.8.1 Android NDK 定义的变量

在系统分析 Android.mk 文件之前，会定义一些全局变量。在某些情况下，系统可以对 Android.mk 文件分析多次，而每次分析时，这些变量的值可能会不一样。下面介绍一下这些变量。

- **CLEAR\_VARS:** 指定一个用于清空几乎所有以“LOCAL\_”开头的变量(除了 LOCAL\_PATH

变量) 的 GNU Make 脚本文件。在 Android.mk 文件的第 2 行(第 1 行设置 LOCAL\_PATH 变量)必须执行这个脚本, 例如, include \$(CLEAR\_VARS)。

- **BUILD\_SHARED\_LIBRARY:** 指定一个建立共享库的 GNU Make 脚本文件。该脚本文件会根据以“LOCAL\_”开头的变量决定如何生成共享库, 其中 LOCAL\_MODULE 和 LOCAL\_SRC\_FILES 是必须设置的两个变量。该变量的用法: include \$(BUILD\_SHARED\_LIBRARY), 生成的共享库文件名是 lib\$(LOCAL\_MODULE).so。

- **BUILD\_STATIC\_LIBRARY:** 指定一个建立静态库的 GNU Make 脚本文件。静态库不能被复制到 Android 应用程序包 (apk 文件) 中, 但可以用于建立共享库。使用该变量的用法: include \$(BUILD\_STATIC\_LIBRARY), 生成的静态库文件名是 \$(LOCAL\_MODULE).a。

- **TARGET\_ARCH:** 编译 Android 的目标 CPU 架构的名称。例如, 与 ARM 兼容的 CPU 架构名称为 arm。

- **TARGET\_PLATFORM:** 指定分析 Android.mk 文件的 Android 平台名称。

- **TARGET\_ARCH\_ABI:** 用于分析 Android.mk 的目标 CPU+ABI 的名称, 在这里 ABI 是指应用程序二进制接口 (Application Binary Interface)。所有基于 ARM 的 ABI 都必须将 TARGET\_ARCH 变量的值设为 arm, 但可以设置不同的 TARGET\_ARCH\_ABI 变量值。

- **TARGET\_ABI:** 该变量用于连接目标平台和 ABI, 也就是 \$(TARGET\_PLATFORM)-\$(TARGET\_ARCH\_ABI), 主要用来测试真实设备中特定的目标系统映像 (Target System Image)。

## 21.8.2 Android NDK 定义的函数

在 Android NDK 中还定义了很多 GNU Make 函数, 这些函数需要使用如下语法格式来调用, 并返回文本信息:

```
| $(call <function>)
```

下面详细介绍一下这些函数的功能及用法。

- **my-dir:** 返回 Android.mk 文件所在的目录, 该函数一般用于设置 LOCAL\_PATH 变量。用法: LOCAL\_PATH := \$(call my-dir)。

- **all-subdir-makefiles:** 返回 Android.mk 文件所在目录中所有包含 Android.mk 文件的子目录列表。例如, 有如下的目录结构:

```
| sources/foo/Android.mk
| sources/foo/lib1/Android.mk
| sources/foo/lib2/Android.mk
```

在 sources/foo 目录的 Android.mk 文件中使用了 include \$(call all-subdir-makefiles), 这个 Android.mk 文件会自动包含 lib1 和 lib2 目录中的 Android.mk 文件。

要注意的是, 这个函数可以进行深度嵌套搜索, 但在默认情况下, NDK 只寻找/\*/Android.mk 一级的文件, 也就是只在当前 Android.mk 文件所在目录的直接子目录中寻找 Android.mk 文件。

- **this-makefile:** 返回当前 GNU Makefile 的路径。
- **parent-makefile:** 返回当前调用树中父一级的 Makefile 的路径。

- **grand-parent-makefile:** 从这个函数的名字不难看出它的功能，返回 parent 的 parent makefile 的路径。

### 21.8.3 描述模块的变量

本节将详细介绍用于描述模块的变量，这些变量可以定义在 `include $(CLEAR_VARS)` 和 `$(BUILD_XXXXX)` 之间。

- **LOCAL\_PATH:** 该变量用于指定当前 `Android.mk` 文件所在的路径，这个变量必须在 `Android.mk` 文件的第 1 行定义。用法：`LOCAL_PATH := $(call my-dir)`。

• **LOCAL\_MODULE:** 该变量指定了模块的名字。模块名必须在所有模块名中是唯一的，而且不能包含空白分隔符（例如，空格、Tab 等）。该变量必须在执行 `$(BUILD_XXXXX)` 脚本之前定义。模块名决定了生成的库文件名，例如，模块名为 `search`，生成的动态库文件名为 `libsearch.so`。而在引用模块时（在 `Android.mk` 或 `Application.mk` 文件中引用）只能使用定义的模块名（如 `search`），而不能使用库文件名（如 `libsearch.so`）。

• **LOCAL\_SRC\_FILES:** 该变量指定了参与模块编译的 C/C++ 源文件名。这些源文件会被传递给编译器，然后编译器会自动计算这些源文件之间的依赖关系。这些源文件的名称或路径都相对于 `LOCAL_PATH`，如果指定多个源文件，中间可以用空格分隔。路径需要使用 UNIX 风格的斜杠（/）。在 Windows 环境下也要使用斜杠表示路径，例如，`LOCAL_SRC_FILES = fun.c product/fun1.c product/fun2.c`。

• **LOCAL\_CPP\_EXTENSION:** 该变量是可选的，用于设置 C++ 源文件的扩展名，默认值是 “.cpp”，但可以通过该变量改变默认的扩展名，例如，`LOCAL_CPP_EXTENSION := .cxx`。

• **LOCAL\_C\_INCLUDES:** 该变量是可选的，用于设置 C/C++ 源文件的搜索路径列表。这些路径相对于 NDK 的根目录，例如，`LOCAL_C_INCLUDES := sources/foo`。也可以利用其他的变量设置该变量，例如，`LOCAL_C_INCLUDES := $(LOCAL_PATH)/..foo`。该变量需要在任何标志变量（如 `LOCAL_CFLAGS`、`LOCAL_CPPFLAGS`）前设置。

• **LOCAL\_CFLAGS:** 该变量是可选的，用于设置编译 C/C++ 源文件所需要的编译器标志。在 Android NDK Revision 1 中该变量仅仅被应用于 C 语言，要设置 C++ 编译器的标志可以使用 `LOCAL_CPPFLAGS` 变量。

• **LOCAL\_CPPFLAGS:** 该变量是可选的，用于设置 C++ 源文件的编译器标志。该变量设置的编译器标志将加在 `LOCAL_CFLAGS` 变量设置的编译器标志后面。在 Android NDK Revision 1 中，该变量设置的编译器标志可应用于 C 和 C++ 编译器中。

• **LOCAL\_CXXFLAGS:** `LOCAL_CPPFLAGS` 变量的别名，官方并不建议使用该变量，因为在以后的 NDK 版本中该变量可能会被删除。

• **LOCAL\_STATIC\_LIBRARIES:** 静态库模块列表。该变量用于在生成共享库时将静态库链接到共享库中，注意该变量只在共享库模块中起作用。

• **LOCAL\_SHARED\_LIBRARIES:** 指定生成的静态库或共享库在运行时依赖的共享库模块列表。这些依赖信息被写入生成的静态库或共享库中。

• **LOCAL\_LDLIBS:** 指定附加的链接标志，这些标志被用来建立模块。这些标志需要使用前缀 -l，例如，“`LOCAL_LDLIBS := -lsearch`” 表示当前的模块在运行时需要依赖于 `libsearch.so`。

- **LOCAL\_ALLOW\_UNDEFINED\_SYMBOLS**: 默认情况下，在建立共享库时如果遇到未定义的引用，系统会抛出 undefined symbol 错误。但如果出于某些原因需要关闭未定义检查，就需要将该变量的值设为 true。但要注意，如果将该变量设为 true，生成的动态库在运行时可能出错。
- **LOCAL\_ARM\_MODE**: 在默认情况下，ARM 架构下的二进制文件都在 thumb 模式下产生，在这种模式下，每一个指令都是 16 位的。如果要强迫生成 32 位的指令，可以将该变量的值设成 arm。要注意的是，还可以通过在 LOCAL\_SRC\_FILES 变量加 arm 后缀的方式指定的 C/C++源文件生成 32 位的二进制文件，例如 LOCAL\_SRC\_FILES := foo.c bar.c.arm。

#### 21.8.4 配置 Application.mk 文件

Application.mk 文件用于描述当前应用程序需要哪些模块，该文件必须放在<Android NDK 安装目录>\apps\<myapp>目录中，其中<myapp>是当前应用程序的目录。Application.mk 与 Android.mk 一样，也使用了 GNU Makefile 语法。系统也为该文件定义了一些变量，这些变量的含义如下。

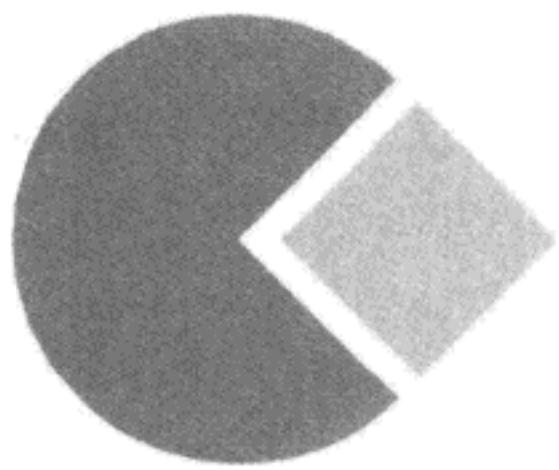
- **APP\_MODULES**: 该变量指定了当前应用程序中需要的模块列表（这些模块在 Android.mk 文件中定义）。如果指定多个模块，中间需要使用空格分隔。
- **APP\_PROJECT\_PATH**: 该变量指定了应用程序工程根目录的绝对路径。该路径也被用来复制/安装生成的共享库（lib\*.so 文件）。
- **APP\_OPTIM**: 该变量是可选的。可设置的值为 release 和 debug，分别用于表示发行模式和调试模式。通过设置这个变量可以改变编译器生成目标文件的优化层次。该变量的默认值是 release，在发行模式下生成的二进制文件会高度优化，而调试（debug）模式产生的优化二进制文件更适合于调试程序。要注意的是，在 release 和 debug 模式下都可以进行调试，只是在 release 模式下提供了较少的调试信息。例如，在调试会话中，某些变量由于被优化而不能被监视；经过重构的代码使按步（stepping）跟踪变得非常困难。
- **APP\_CFLAGS**: C 编译器的标志。当编译任何模块的 C 语言源代码时可以使用该变量。通过该变量可以改变在 Android.mk 文件中设置的相应的 C 编译器标志，以便满足当前应用程序的需要。
- **APP\_CXXFLAGS**: 与 APP\_CFLAGS 变量类似，只是用于设置 C++ 编译器的标志。
- **APP\_CPPFLAGS**: 与 APP\_CFLAGS 类似，只是用于设置 C/C++ 编译器的标志。

配置 Application.mk 文件的例子如下：

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES      := ch21_jni_helloworld
```

## 21.9 小结

本章主要介绍了 NDK 编程的基本方法。如果在 Windows 中使用 NDK，除了要安装 Android SDK 和 Eclipse 外，还要安装 Cygwin 和 Android NDK。在 Java 中调用 NDK 程序 (.so 文件) 需要先将 C/C++ 代码编译成 .so 文件，最直接的方法是通过命令进行编译，但这种方式并不适合开发。因此，本章介绍了另外一种将 NDK 和 Eclipse 集成的方法。除此之外，本章还给出了一些精彩的例子（如 NDK OpenGL ES API、NDK Audio API、Native Activity 等）来演示如何使用 NDK 开发各种类型的应用程序。本章的最后讲解了 Android.mk 和 Application.mk 文件中常用的变量和函数。



## 第 22 章 测试驱动开发（TDD）

测试应用程序有很多方法，例如，黑盒测试、白盒测试、迭代测试等，然而，这些方法都是从宏观上描述测试的。为了在技术上保障测试的效果，Kent Beck（也是极限编程的创始人）提出了在结果上进行限制的测试方法，也就是在编写程序之前，先确定程序中的变量、控件等元素允许的值。如果在编写程序时，变量、控件中的值与事先确定的值不相符，就说明程序的某处有 bug，这种测试方法就是 TDD（Test Driven Development，测试驱动开发）。TDD 与 OpenGL ES 一样，并不是具体的软件或程序库，只是一套标准或一套 API。目前基于各种语言的 TDD 框架很多，在 Android SDK 中也提供了一套测试框架（JUnit），可用于对 Android 应用程序进行 TDD 测试。本章将详细讨论如何使用 JUnit 测试 Android 应用程序。

### 22.1 JUnit 测试框架

Android 测试框架已被集成到 ADT 中，提供了强大的工具可以帮助我们测试应用程序的方方面面。下面看一下测试框架的主要特性。

- Android 的测试框架基于 JUnit。我们可以在不需要调用 Android SDK API 的情况下测试 Java 类，或通过 JUnit 扩展测试 Android 应用程序组件。我们可以直接使用 `AndroidTestCase` 进行一般性的测试（测试普通的 Java 类），然后再逐渐深入更复杂的测试。
- Android JUnit 扩展提供了特定组件的测试案例类，这些类可以很容易地创建 mock 对象，并可以控制组件的生命周期。
- 测试框架可以像其他 Android SDK API 一样使用，因此，测试和建立测试并不需要掌握新的工具或技术。
- JUnit 和 ADT 集成在了一起，我们可以用可视化的方法观察测试结果。
- Android SDK 还提供了 `monkeyrunner`，一套使用 Python 编写的测试 API，以及相应的命令行工具。`Monkey` 可用于对应用程序进行压力测试。

下面看一下 Android 测试框架的结构图，如图 22.1 所示。

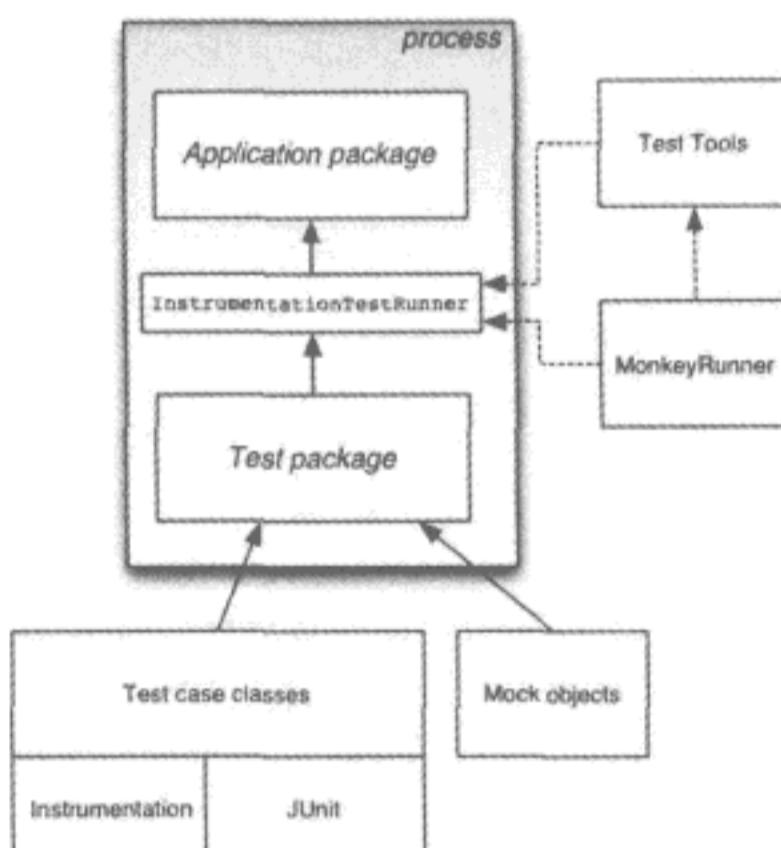
### 22.2 测试 Activity

工程目录：`src\ch22\ch22_helloworld`    `src\ch22\ch22_helloworld_test`

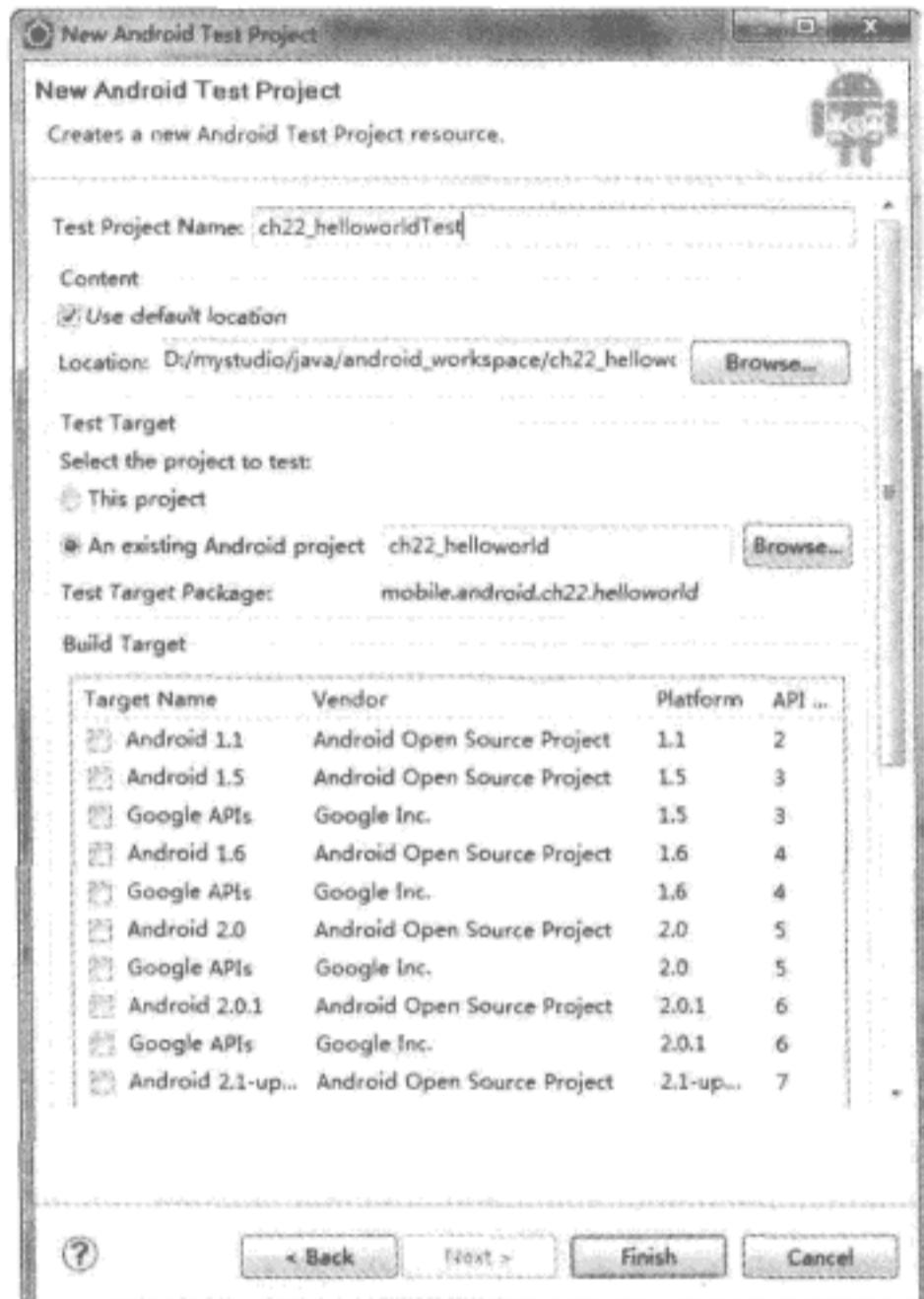
JUnit 测试 Activity 的基本原理就是在测试工程中引用被测试的工程（跨工程引用），然后在测试工程中访问被测试工程的类，并利用测试框架的 API 进行测试，最后运行测试工程，会在 JUnit 视图中显示测试结果。下面看一下测试 Activity 的具体步骤。

### (1) 建立测试工程

在 Eclipse 中单击“New”>“Other”>“Android”>“Android Test Project”，新建一个测试工程。建立测试工程时在“Test Target”处选择“An existing Android Project”，并单击“Browse”按钮选择被测试的工程，如图 22.2 所示。其中 ch22\_helloworld 是一个简单的 Android 工程，界面只有一个 TextView 控件。



▲ 图 22.1 Android 测试框架的结构图



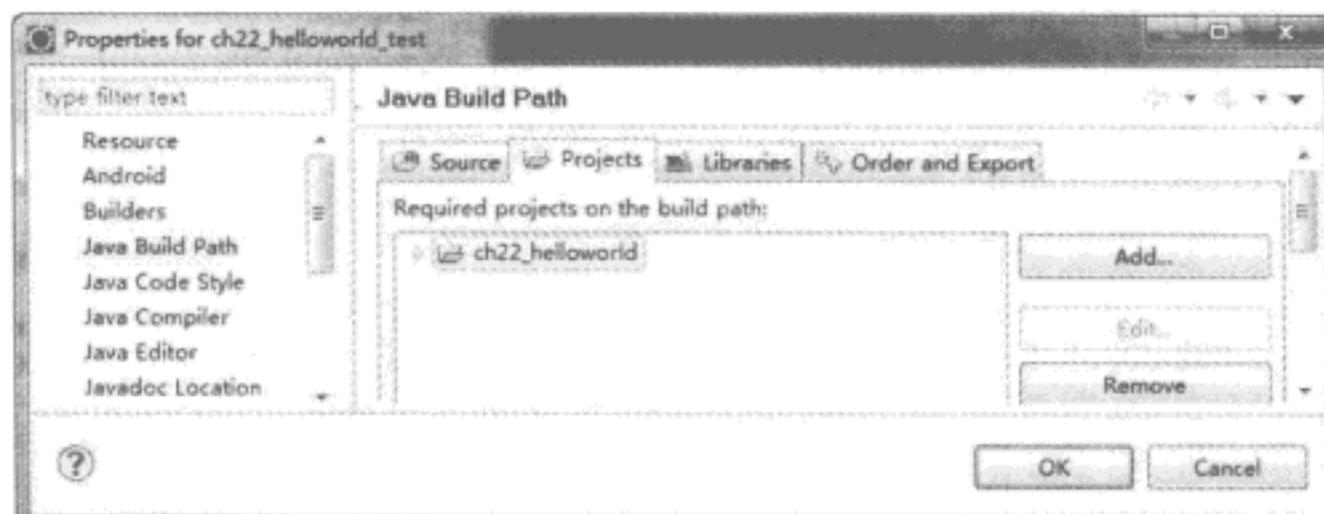
▲ 图 22.2 建立测试工程

查看测试工程的 `AndroidManifest.xml` 文件，会发现多了很多以前从未见过的标签。

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="mobile.android.ch22.helloworld.test"
  android:versionCode="1"
  android:versionName="1.0">
  <uses-sdk android:minSdkVersion="9" />
  <!-- 指定被测试的工程 -->
  <instrumentation android:targetPackage="mobile.android.ch22.helloworld"
    android:name="android.test.InstrumentationTestRunner" />
  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <uses-library android:name="android.test.runner" />
  </application>
</manifest>
  
```

打开测试工程的“Properties”对话框，会发现“Java Build Path”页的“Projects”标签已经引用了被测试工程(ch22\_helloworld)，如图 22.3 所示。因此，在 ch22\_helloworld\_test 工程中是可以访问 ch22\_helloworld 工程中的类的（该工程中只有一个 Main 类）。



▲ 图 22.3 测试工程引用了被测试工程

## (2) 编写测试代码

在 ch22\_helloworld\_test 工程中建立一个 HelloWorldTest.java 文件，该文件用于测试 ch22\_helloworld 工程。首先 HelloWorldTest 类应继承自 ActivityInstrumentationTestCase2，接下来需要覆盖 ActivityInstrumentationTestCase2.setUP 方法来初始化测试案例，最后要编写一些测试方法对 ch22\_helloworld 工程中的 TextView 控件及其值进行限制。现在先看一下 HelloWorldTest 类的代码。

```
package mobile.android.ch22.helloworld.test;

import mobile.android.ch22.helloworld.Main;
import android.app.Activity;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class HelloWorldTest extends ActivityInstrumentationTestCase2<Main>
{
    private Activity activity;
    private TextView textView;

    public HelloWorldTest()
    {
        super("mobile.android.ch22.helloworld", Main.class);
    }
    // 初始化测试案例
    @Override
    protected void setUp() throws Exception
    {
        super.setUp();
        // 获得要测试的 Activity 对象
        activity = this.getActivity();
        textView = (TextView) activity.findViewById(mobile.android.ch22.helloworld.R.id.textview);
    }
    // 测试 TextView 控件是否存在
    public void testPreconditions()
    {
```

```

        assertNotNull(textView);
    }
    // 测试 TextView 控件的文本是否为“世界你好”
    public void testText()
    {
        assertEquals("世界你好", (String) textView.getText());
    }
}

```

测试类中的 `setUp` 方法相当于 `Activity.onCreate` 方法，用于初始化测试案例，如获得 `TextView` 对象。测试类中的测试方法（本例中是 `testPreconditions` 和 `testText`）可以任意取名。测试框架会利用反射技术自动调用这些方法。

`assertNotNull` 方法用于检测对象是否为 `null`，如果从布局文件中无法获得 `TextView` 对象，`assertNotNull` 方法就会将错误信息输出到 JUnit 视图上。例如，虽然资源 ID 存在，但并不是在当前的布局文件中定义的，这是一个经常遇到的 bug。在这种情况下，获得的 `TextView` 对象就为 `null`。如果使用 `assertNotNull`，就会立刻发现这个 bug。

`assertEquals` 方法用于检测两个字符串是否相同。本例用于限定 `TextView` 控件的文本只能是“世界你好”。如果 `ch22_helloworld` 是一个默认的 Android 工程，`TextView` 控件的默认文本是“Hello World, Main!”，这时 `assertEquals` 就会将错误信息输出到 JUnit 视图。

### (3) 进行测试

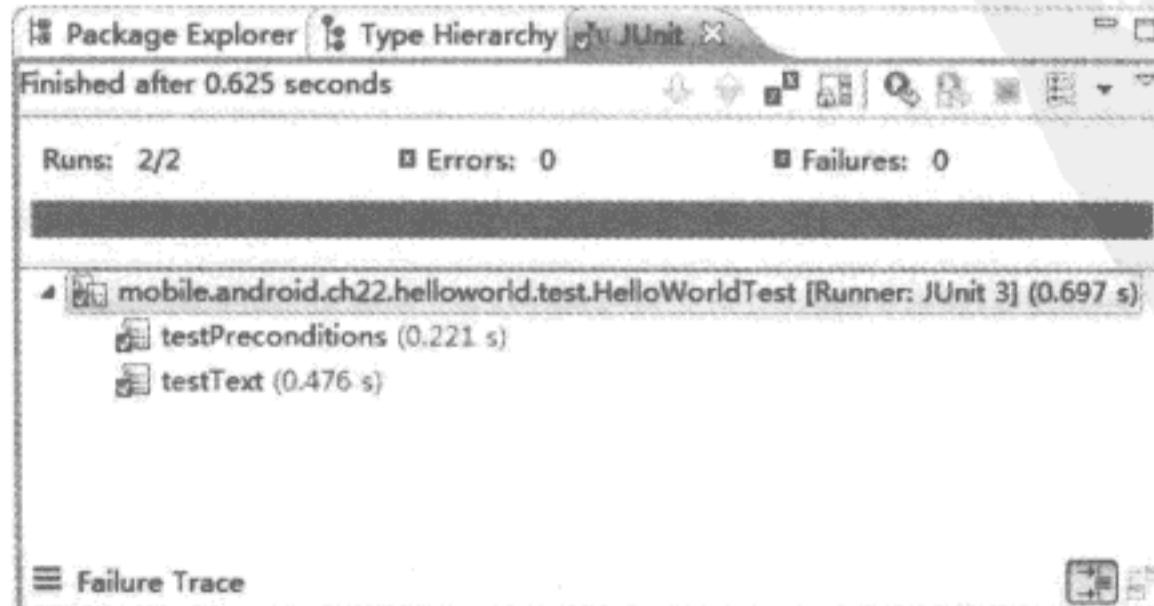
如果 JUnit 视图未显示，单击“Window”>“Show View”>“Other”打开“Show View”对话框，选中“Java”>“JUnit”节点，单击“OK”按钮显示 JUnit 视图。

在测试工程右键菜单中单击“Run As”>“Android JUnit Test”菜单项运行测试案例。如果 `TextView` 控件中的文本不正确，会显示如图 22.4 所示的错误信息。

如果被测试工程没有任何错误，在 JUnit 视图中会显示如图 22.5 所示的信息。



▲ 图 22.4 发现被测试工程有 bug



▲ 图 22.5 测试成功

## 22.3 测试 Content Provider

工程目录: src\ch22\ch22\_contentprovider\_test

测试 Content Provider 需要编写一个继承自 ProviderTestCase2 的测试类。本节将测试 11.3.1 节实现的获得城市信息的 Content Provider。首先在 Eclipse 中导入 ch11\_region\_content\_provider 工程, 然后建立一个测试工程 (别忘了在建立测试工程时选择 ch11\_region\_content\_provider)。

在测试工程中建立一个 ContentProviderTest 类, 并编写如下代码。

```
package mobile.android.ch11.region.content.provider.test;

import mobile.android.ch11.region.content.provider.RegionContentProvider;
import android.content.ContentProvider;
import android.database.Cursor;
import android.net.Uri;
import android.test.ProviderTestCase2;

public class ContentProviderTest extends ProviderTestCase2<RegionContentProvider>
{
    private ContentProvider contentProvider;
    private Cursor cursor;
    private String city;

    public ContentProviderTest()
    {
        super(RegionContentProvider.class,
              "mobile.android.ch11.regioncontentprovider");
    }
    // 初始化测试案例
    @Override
    protected void setUp() throws Exception
    {
        super.setUp();
        try
        {
            // 获得要测试的 ContentProvider 对象
            contentProvider = getProvider();
            Uri uri = Uri
                .parse("content://mobile.android.ch11.regioncontentprovider/code/024");
            cursor = contentProvider.query(uri, null, null, null, null);
            if (cursor.moveToFirst())
            {
                // 获得城市名称
                city = cursor.getString(cursor.getColumnIndex("city_name"));
            }
        }
        catch (Exception e)
        {
        }
    }
    // 测试 Uri 是否正确
```

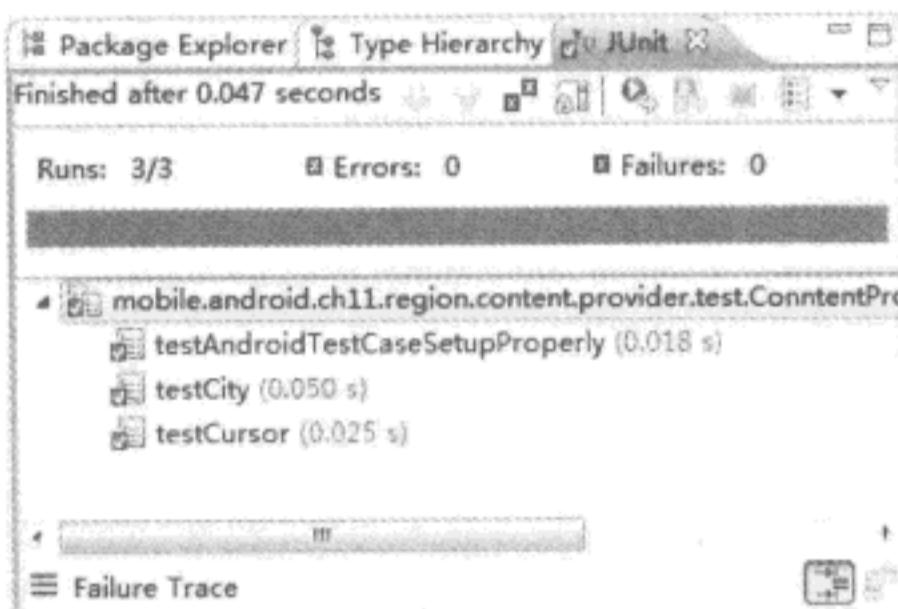
## Android 开发权威指南

```

public void testCursor() throws Exception
{
    assertNotNull(cursor);
}
// 测试返回的城市名称是否正确
public void testCity()
{
    assertEquals("沈阳", city);
}
}

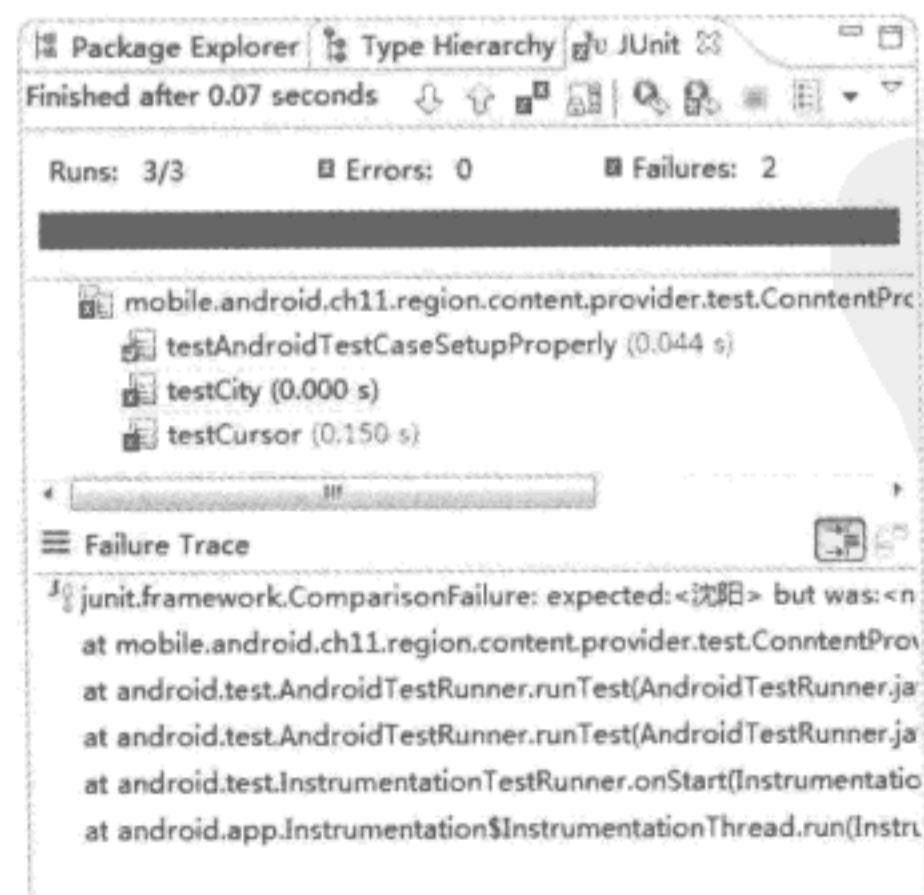
```

ContentProviderTest 类在 setUp 方法中通过 getProvider 方法获得了要测试的 ContentProvider 对象，然后调用 ContentProvider.query 方法查询城市信息，并通过 testCursor 和 testCity 方法测试 Uri 和返回的城市名称是否正确。如果 Uri 错误，cursor 变量的值为 null。当 Uri 和返回的城市名称正确时，运行测试案例，会显示如图 22.6 所示的反馈信息。



▲ 图 22.6 ContentProvider 测试成功

读者可以试着修改 ContentProviderTest 类中的 Uri，再运行测试程序，会看到如图 22.7 所示的反馈信息。



▲ 图 22.7 ContentProvider 测试失败

## 22.4 测试 Service

工程目录: src\ch22\ch22\_service\_test

测试 Service 需要编写一个继承自 ServiceTestCase 的测试类。本节将测试 12.1.1 节实现的演示服务生命周期的 Service。首先在 Eclipse 中导入 ch12\_servicelifecycle 工程，然后建立一个测试工程（别忘了在建立测试工程时选择 ch12\_servicelifecycle）。

在测试工程中建立一个 ServiceTest 类，并编写如下代码。

```
package mobile.android.ch12.service.lifecycle.test;

import mobile.android.ch12.service.lifecycle.MyService;
import mobile.android.ch12.service.lifecycle.R;
import android.content.Intent;
import android.test.ServiceTestCase;

public class ServiceTest extends ServiceTestCase<MyService>
{
    private MyService service;
    public ServiceTest()
    {
        super(MyService.class);
    }
    @Override
    protected void setUp() throws Exception
    {
        super.setUp();
        Intent intent = new Intent(mContext, MyService.class);
        startService(intent); // 启动服务
        service = getService(); // Service 对象
    }
    // 测试是否成功获得 Service 对象
    public void testService()
    {
        assertNotNull(service);
    }
    // 测试被测试工程中某个字符串资源的值是否符合要求
    public void testResource()
    {
        assertEquals("服务的生命周期", service.getString(R.string.app_name));
    }
}
```

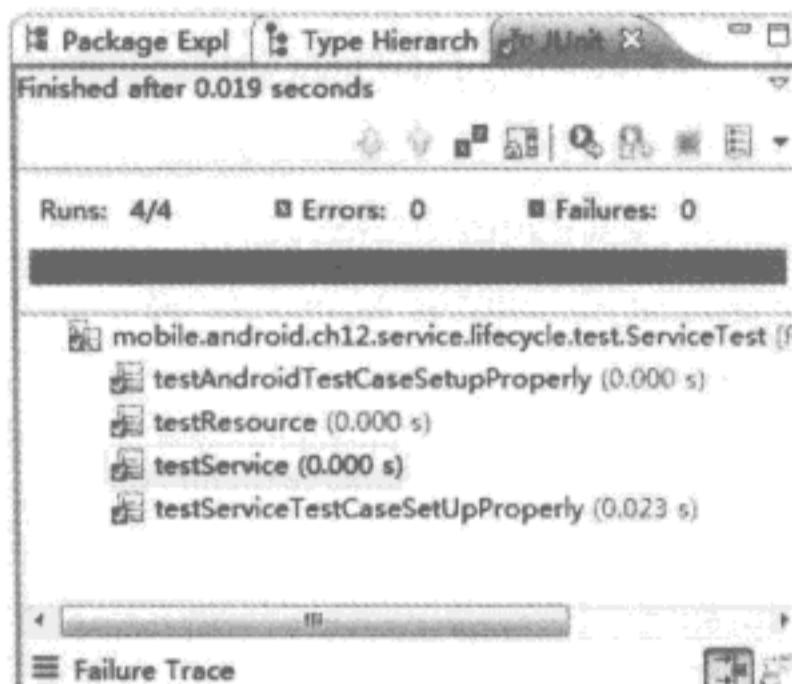
运行测试程序，如果满足所有测试条件，会显示如图 22.8 所示的反馈信息。

## 22.5 测试普通类

工程目录: src\ch22\ch22\_testclass

PDG

JUnit 不仅能测试 Android 的应用程序组件（Activity、ContentProvider 和 Service），还可以利用 `AndroidTestCase` 类测试普通的 Java 类。首先建立一个测试工程，这个工程并不需要测试任何其他的工程，因此，在建立测试工程时要选择“`This Project`”。



▲ 图 22.8 Service 测试成功

在 `ch22_testclass` 工程中建立一个 `MyClass` 类，并编写如下代码。

```
package mobile.android.ch22.testclass;

public class MyClass
{
    private String name = "Android";
    public String getName()
    {
        return name;
    }
}
```

建立一个 `Test` 类，该类继承自 `AndroidTestCase`，并编写如下代码。

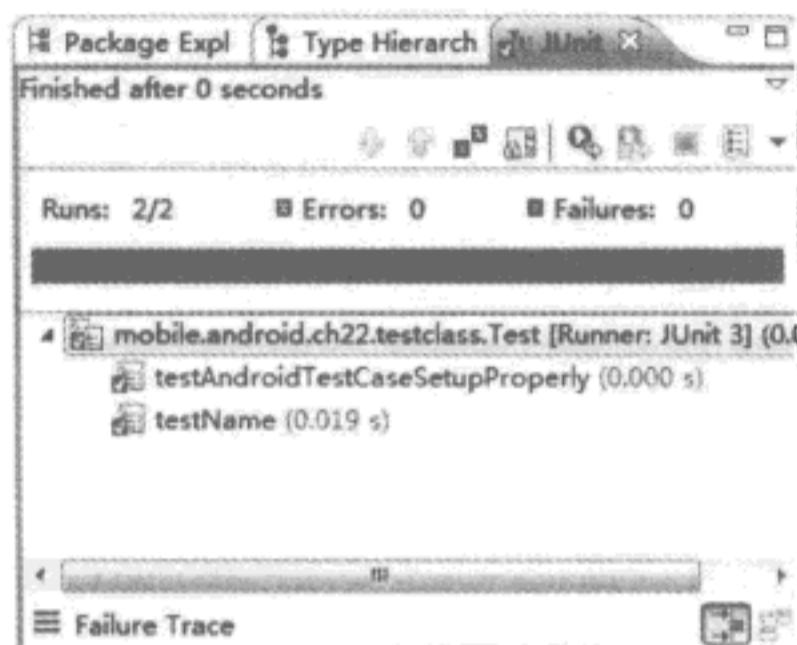
```
package mobile.android.ch22.testclass;

import android.test.AndroidTestCase;

public class Test extends AndroidTestCase
{
    private MyClass myClass;
    @Override
    protected void setUp() throws Exception
    {
        super.setUp();
        myClass = new MyClass();
    }
    // 测试 MyClass.getName 方法的值
    public void testName()
    {
        assertEquals("Android", myClass.getName());
    }
}
```



运行测试程序，如果 getName 方法返回值是“Android”，会显示如图 22.9 所示的反馈信息。



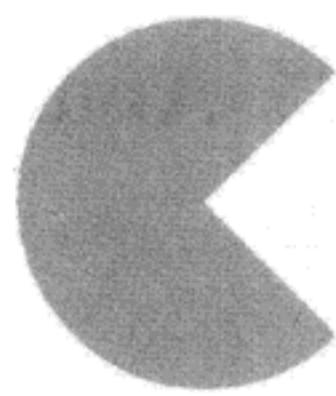
▲ 图 22.9 Java 类测试成功

## 22.6 小结

本章介绍了 Android SDK 提供的测试框架，该测试框架基于 JUnit。测试框架不仅可以测试普通的 Java 类，也可以测试 Android 的应用程序组件，如 Activity、ContentProvider 和 Service。







## 第四部分

### 综合实例篇

第 23 章 Android 综合案例一——  
蓝牙聊天

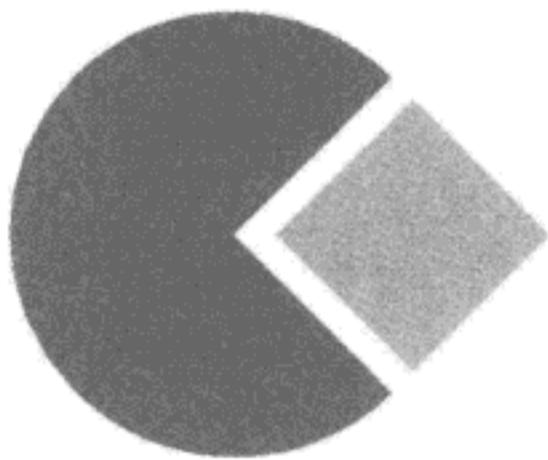
第 24 章 Android 综合案例二——  
月球登录（游戏）

第 25 章 Android 综合案例三——  
全键盘输入法（应用）

第 26 章 Android 综合案例四——  
贪吃蛇（游戏）

第 27 章 Android 综合案例五——  
新浪微博客户端（应用）

第 28 章 Android 综合案例六——  
笑脸连连看（游戏）



## 第 23 章 Android 综合案例—— 蓝牙聊天

本章将实现一个完整的点对点蓝牙聊天程序，要测试本章的例子需要两部装有 Android 2.1 及以上版本的手机。如果两部手机通过本例成功连接，可以互相发送文本信息进行聊天。本例的工程目录是 src\demo\bluetooth\_chat。

### 23.1 蓝牙聊天主界面

蓝牙聊天的主界面(BluetoothChat 类)比较简单，只有一个 EditText 和一个 Button 控件。布局文件为 main.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ListView android:id="@+id/in" android:layout_width="match_parent"
        android:layout_height="match_parent" android:stackFromBottom="true"
        android:transcriptMode="alwaysScroll" android:layout_weight="1" />
    <LinearLayout android:orientation="horizontal"
        android:layout_width="match_parent" android:layout_height="wrap_content">
        <!-- 输入聊天信息的 EditText 控件 -->
        <EditText android:id="@+id/edit_text_out"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:layout_weight="1" android:layout_gravity="bottom" />
        <!-- 发送聊天信息的按钮 -->
        <Button android:id="@+id/button_send" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="@string/send" />
    </LinearLayout>
</LinearLayout>
```

主界面使用了定制的标题栏，在标题栏的右侧显示蓝牙设备的状态。在 onCreate 方法中还检测了蓝牙设备是否开启，代码如下：

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

    // 设置定制标题栏的布局
    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
    setContentView(R.layout.main);
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
```

```

        R.layout.custom_title);

// 设置标题栏上显示的内容
mTitle = (TextView) findViewById(R.id.title_left_text);
mTitle.setText(R.string.app_name);
mTitle = (TextView) findViewById(R.id.title_right_text);

// 获得蓝牙适配器
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

// 如果 mBluetoothAdapter 为 null，说明当前手机没有蓝牙模块
if (mBluetoothAdapter == null)
{
    Toast.makeText(this, "当前手机不支持蓝牙。",
        Toast.LENGTH_LONG).show();
    finish();
    return;
}
}

```

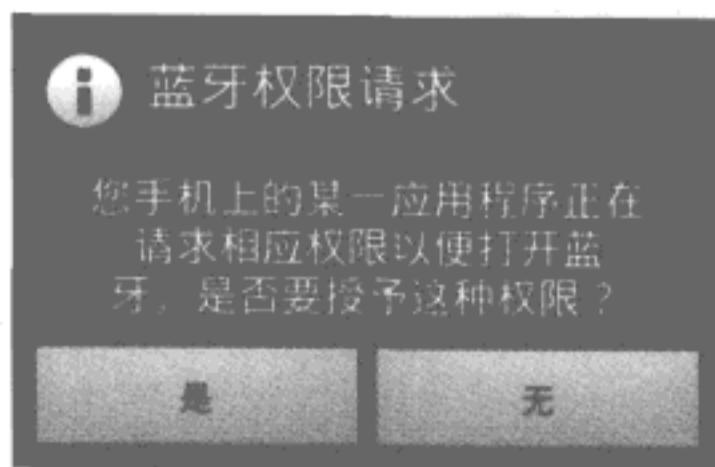
在上面代码中涉及一个 `custom_title` 布局，这个布局可以任意设置标题栏的内容。`custom_title.xml` 布局文件的内容如下：

```

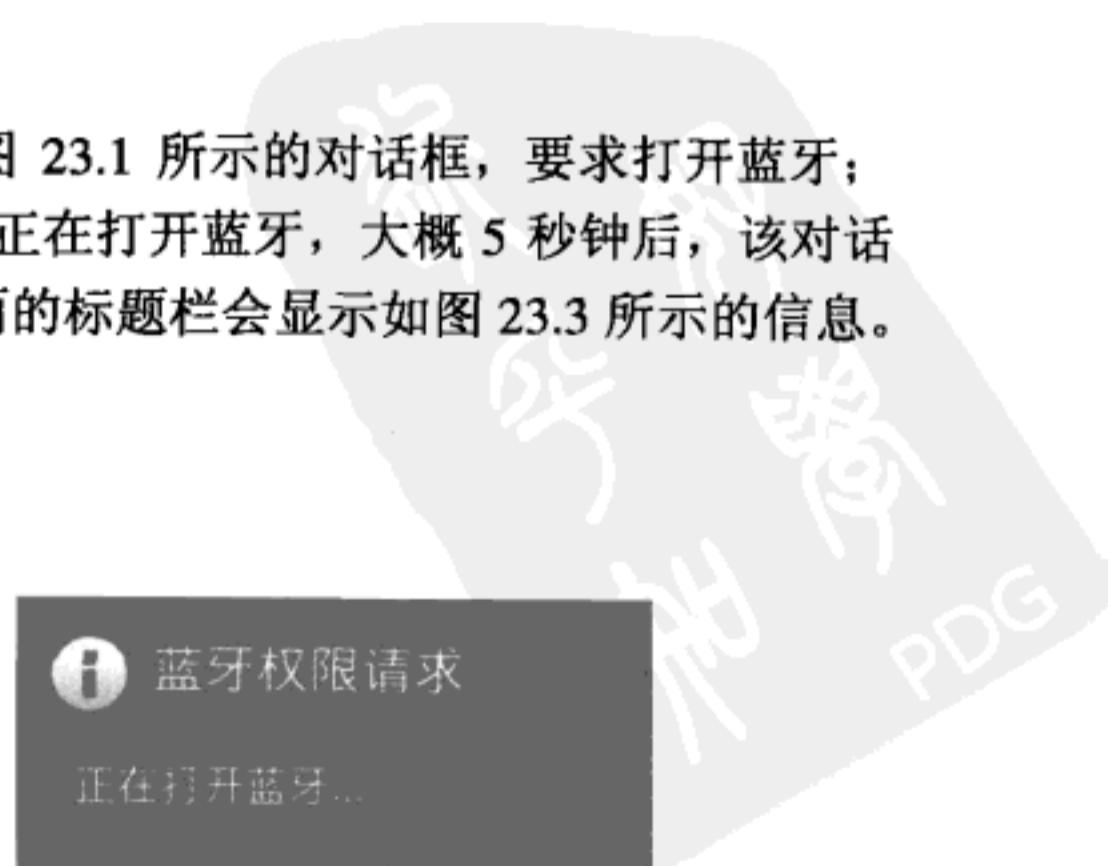
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:gravity="center_vertical">
    <!-- 标题栏左侧的 TextView 控件，用于显示程序名称 -->
    <TextView android:id="@+id/title_left_text"
        android:layout_alignParentLeft="true" android:ellipsize="end"
        android:singleLine="true" style="?android:attr/windowTitleStyle"
        android:layout_width="wrap_content" android:layout_height="match_parent"
        android:layout_weight="1" />
    <!-- 标题栏右侧的 TextView 控件，用于显示蓝牙设备当前的状态 -->
    <TextView android:id="@+id/title_right_text"
        android:layout_alignParentRight="true" android:ellipsize="end"
        android:singleLine="true" android:layout_width="wrap_content"
        android:layout_height="match_parent" android:textColor="#fff"
        android:layout_weight="1" />
</RelativeLayout>

```

运行蓝牙聊天程序，如果蓝牙还没有打开，会弹出如图 23.1 所示的对话框，要求打开蓝牙；单击“是”按钮后，会显示如图 23.2 所示的对话框，提示正在打开蓝牙，大概 5 秒钟后，该对话框自动消失，如果手机支持蓝牙，则蓝牙会被打开。主界面的标题栏会显示如图 23.3 所示的信息。



▲ 图 23.1 要求打开蓝牙



▲ 图 23.2 正在打开蓝牙



▲ 图 23.3 蓝牙聊天主界面（未连接）

## 23.2 添加选项菜单

蓝牙聊天的主界面有一个选项菜单，菜单资源文件是 option\_menu.xml，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/scan"
        android:icon="@android:drawable/ic_menu_search"
        android:title="@string/connect" />
    <item android:id="@+id/discoverable"
        android:icon="@android:drawable/ic_menu_mylocation"
        android:title="@string/discoverable" />
</menu>
```

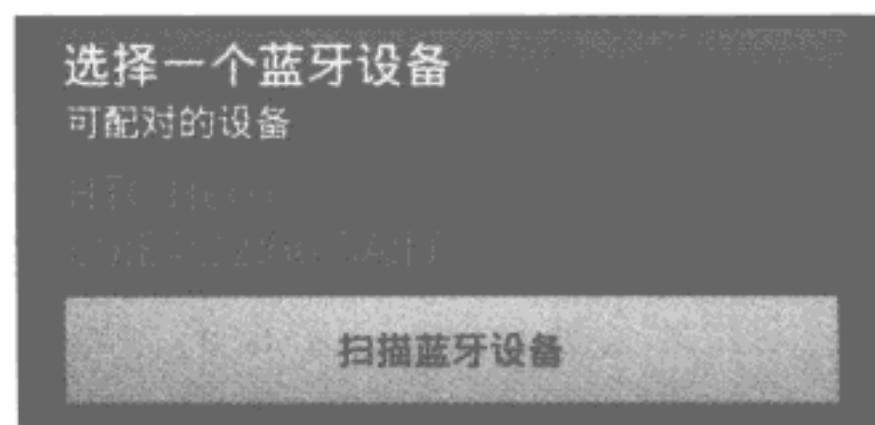
这两个菜单项，一个用于连接蓝牙设备，一个用于使当前手机可被其他蓝牙设备发现。选项菜单的效果如图 23.4 所示。



▲ 图 23.4 主界面的选项菜单

## 23.3 搜索和连接蓝牙设备

单击“连接蓝牙设备”菜单项，会显示如图 23.5 所示的搜索和选项蓝牙设备界面。



▲ 图 23.5 搜索蓝牙设备

打开 DeviceListActivity.java 文件，会看到 onCreate 方法的主要功能是初始化设备列表，将已经配对的设备显示在如图 23.5 所示的列表中，代码如下；

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

```
// 为标题栏加上圆形进度条
requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
setContentView(R.layout.device_list);

// 设置关闭当前 Activity 的返回值
setResult(Activity.RESULT_CANCELED);

// 初始化搜索按钮
Button scanButton = (Button) findViewById(R.id.button_scan);
scanButton.setOnClickListener(new OnClickListener()
{
    public void onClick(View v)
    {
        doDiscovery();
        v.setVisibility(View.GONE);
    }
});

// 初始化用于保存已配对的蓝牙设备的 ArrayAdapter 对象
mPairedDevicesArrayAdapter = new ArrayAdapter<String>(this,
    R.layout.device_name);
// 初始化用于保存新搜索到的蓝牙设备的 ArrayAdapter 对象
mNewDevicesArrayAdapter = new ArrayAdapter<String>(this,
    R.layout.device_name);
ListView pairedListView = (ListView) findViewById(R.id.paired_devices);
pairedListView.setAdapter(mPairedDevicesArrayAdapter);
pairedListView.setOnItemClickListener(mDeviceClickListener);

ListView newDevicesListView = (ListView) findViewById(R.id.new_devices);
newDevicesListView.setAdapter(mNewDevicesArrayAdapter);
newDevicesListView.setOnItemClickListener(mDeviceClickListener);

// 注册用一个蓝牙设备被搜索到广播的接收器
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);

// 注册搜索完成的接收器
filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);

mBtAdapter = BluetoothAdapter.getDefaultAdapter();

// 获得当前已经配对的设备
Set<BluetoothDevice> pairedDevices = mBtAdapter.getBondedDevices();

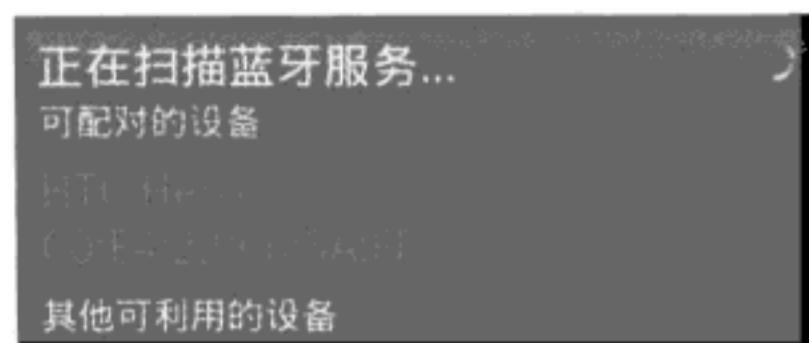
// 如果当前有已经配对的设备，将其显示在列表中
if (pairedDevices.size() > 0)
{
    findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
    for (BluetoothDevice device : pairedDevices)
    {
        mPairedDevicesArrayAdapter.add(device.getName() + "\n"
            + device.getAddress());
    }
}
```

```

    }
    else
    {
        String noDevices = getResources().getText(R.string.none_paired)
            .toString();
        mPairedDevicesArrayAdapter.add(noDevices);
    }
}

```

单击“扫描蓝牙设备”按钮，会搜索周围的蓝牙设备，搜索界面如图 23.6 所示。



▲ 图 23.6 搜索蓝牙设备

下面的代码用于处理搜索到的蓝牙设备以及搜索完成的动作。

```

private final BroadcastReceiver mReceiver = new BroadcastReceiver()
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        String action = intent.getAction();

        // 每发现一个设备，会执行下面的代码
        if (BluetoothDevice.ACTION_FOUND.equals(action))
        {
            // 获得 BluetoothDevice 对象
            BluetoothDevice device = intent
                .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // 如果该设备已经配对，则忽略该设备
            if (device.getBondState() != BluetoothDevice.BOND_BONDED)
            {
                // 设备未配对，将其加到 ArrayAdapter 对象中，以便可显示在列表中
                mNewDevicesArrayAdapter.add(device.getName() + "\n"
                    + device.getAddress());
            }
        }
        // 当搜索完成时执行下面的代码
        else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
        {
            setProgressBarIndeterminateVisibility(false);
            setTitle(R.string.select_device);
            // 如果未搜索到任何设备，在当前界面显示提示信息
            if (mNewDevicesArrayAdapter.getCount() == 0)
            {
                String noDevices = getResources().getText(
                    R.string.none_found).toString();
                mNewDevicesArrayAdapter.add(noDevices);
            }
        }
    }
}

```



```
}; } }
```

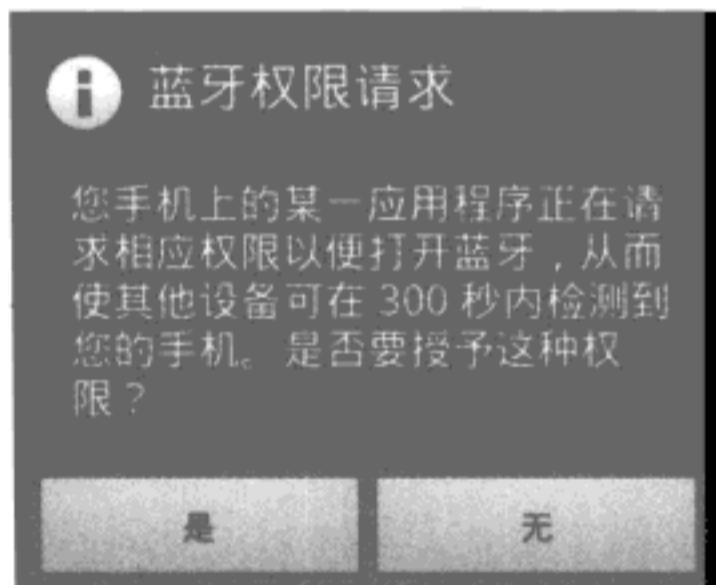
如果搜索到新的蓝牙设备，或存在已配对的蓝牙设备。会在当前界面显示这些蓝牙设备，单击某一个蓝牙设备，如果连接成功，会在主界面标题栏显示如图 23.7 所示的信息。



▲ 图 23.7 蓝牙设备连接成功

## 23.4 | 使设备可被其他蓝牙设备发现

单击主界面的“可被发现”菜单项，如果当前手机不可被发现，会弹出如图 23.8 所示的对话框，单击“是”按钮，会在 300 秒之内使其他蓝牙设备可发现当前手机。



▲ 图 23.8 使当前手机可被发现

下面的代码用于设置可被发现状态。

```
private void ensureDiscoverable()
{
    if (mBluetoothAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
    {
        Intent discoverableIntent = new Intent(
                BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverableIntent.putExtra(
                BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
        startActivityForResult(discoverableIntent);
    }
}
```

## 23.5 发送和接收聊天信息

当单击主界面的“发送”按钮，会将文本框中输入的信息发送给另一端的蓝牙设备，代码如下：

```

private void sendMessage(String message)
{
    // 在发送消息之前核对蓝牙设备是否成功连接
    if (mChatService.getState() != BluetoothChatService.STATE_CONNECTED)
    {
        Toast.makeText(this, R.string.not_connected, Toast.LENGTH_SHORT)
            .show();
        return;
    }

    // 检查文本输入框中是否有文本
    if (message.length() > 0)
    {
        // 获得发送消息的字节形式的数据
        byte[] send = message.getBytes();
        mChatService.write(send);
        mOutStringBuffer.setLength(0);
        mOutEditText.setText(mOutStringBuffer);
    }
}

```

接收聊天信息由 `BluetoothChatService` 类完成，其中 `AcceptThread` 是一个线程类，用于检测其他蓝牙设备的连接，代码如下；

```

private class AcceptThread extends Thread
{
    // 蓝牙服务端 Socket
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread()
    {
        BluetoothServerSocket tmp = null;

        // 创建一个临时的蓝牙服务端 Socket
        try
        {
            tmp = mAdapter
                .listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        }
        catch (IOException e)
        {

        }
        mmServerSocket = tmp;
    }

    public void run()
    {
        setName("AcceptThread");
        BluetoothSocket socket = null;
        // 监听蓝牙设备是否被连接
        while (mState != STATE_CONNECTED)
        {
            try
            {

```

```
// 如果连接成功，返回蓝牙客户端 Socket 对象
socket = mmServerSocket.accept();
}
catch (IOException e)
{
    Log.e(TAG, "accept() failed", e);
    break;
}
if (socket != null)
{
    synchronized (BluetoothChatService.this)
    {
        // 处理各种状态
        switch (mState)
        {
            case STATE_LISTEN:
            case STATE_CONNECTING:
                // 开始处理输入输出数据的线程
                connected(socket, socket.getRemoteDevice());
                break;
            case STATE_NONE:
            case STATE_CONNECTED:
                // 关闭蓝牙 Socket
                try
                {
                    socket.close();
                }
                catch (IOException e)
                {
                }
                break;
        }
    }
}
public void cancel()
{
    try
    {
        mmServerSocket.close();
    }
    catch (IOException e)
    {
    }
}
}
```

当接收到蓝牙客户端 Socket 后，就需要用 InputStream 和 OutputStream 来接收和发送数据，这就需要一个处理数据传输的线程类 ConnectedThread，代码如下：

```
private class ConnectedThread extends Thread
{
    private final BluetoothSocket mmSocket;
```

```
private final InputStream mmInStream;
private final OutputStream mmOutStream;

public ConnectedThread(BluetoothSocket socket)
{
    mmSocket = socket;
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    // 获得蓝牙 Socket 的 InputStream 和 OutputStream
    try
    {
        tmpIn = socket.getInputStream();
        tmpOut = socket.getOutputStream();
    }
    catch (IOException e)
    {
    }
    mmInStream = tmpIn;
    mmOutStream = tmpOut;
}
public void run()
{
    byte[] buffer = new byte[1024];
    int bytes;

    // 不断监视 InputStream，一旦蓝牙客户端 Socket 发过来信息，就会立即接收到
    while (true)
    {
        try
        {
            // 读蓝牙客户端发送过来的信息，如果未发送任何信息，该条语句被阻塞
            bytes = mmInStream.read(buffer);

            // 更新聊天记录列表中的内容
            mHandler.obtainMessage(BluetoothChat.MESSAGE_READ, bytes,
                    -1, buffer).sendToTarget();
        }
        catch (IOException e)
        {
            connectionLost();
            break;
        }
    }
}

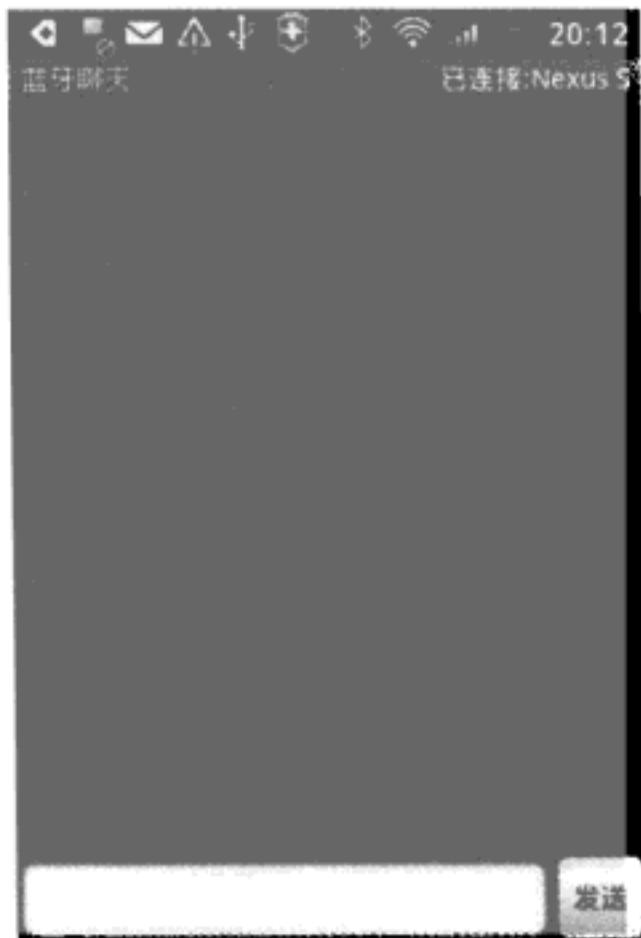
// 向另一个蓝牙设备发送字节数据
public void write(byte[] buffer)
{
    try
    {
        mmOutStream.write(buffer);

        // 更新聊天记录列表中的内容
        mHandler.obtainMessage(BluetoothChat.MESSAGE_WRITE, -1, -1,

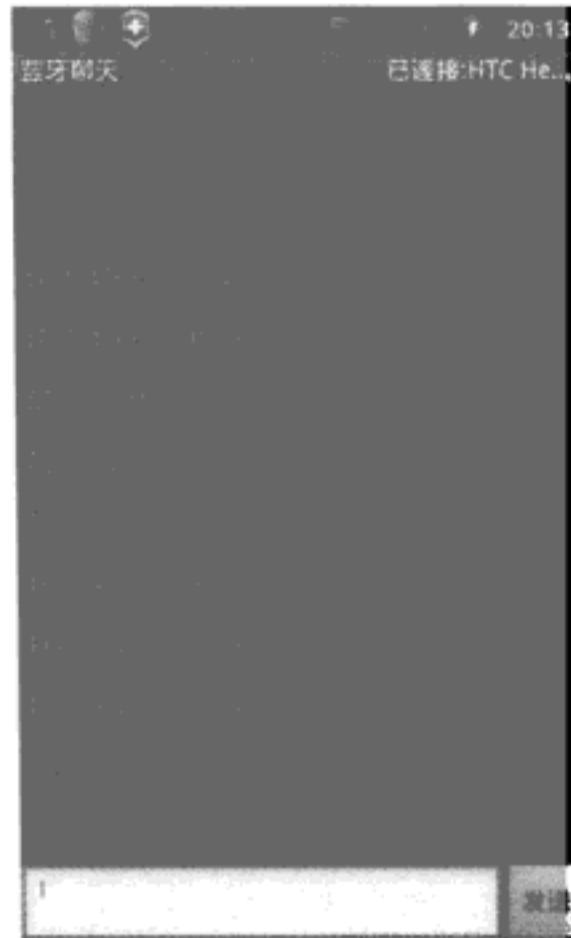
```

```
        buffer).sendToTarget();  
    }  
    catch (IOException e)  
    {  
    }  
}  
public void cancel()  
{  
    try  
    {  
        mmSocket.close();  
    }  
    catch (IOException e)  
    {  
    }  
}  
}
```

在两部手机上运行蓝牙聊天程序，并进行连接，聊天效果如图 23.9 和图 23.10 所示。



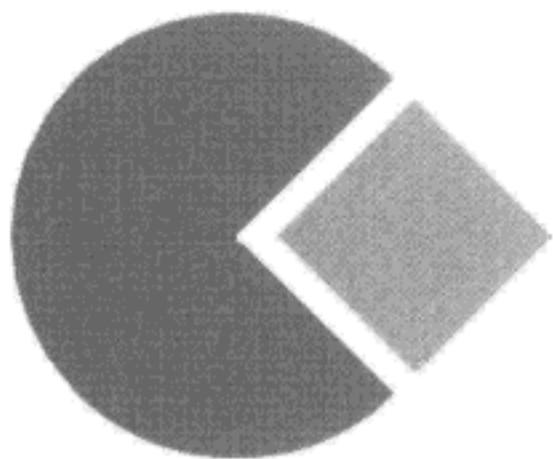
▲ 图 23.9 G3 上的聊天效果



▲ 图 23.10 Nexus S 上的聊天效果

## 23.6 小结

本章介绍了如何利用蓝牙技术实现一个完整的聊天程序。使用蓝牙设备之前需要先连接蓝牙设备。如果想使其他的蓝牙设备可以搜索到自己的手机，需要打开被发现功能（只在一定时间内有效），然后就像网络 Socket 一样，服务端通过 accept 方法获得 BluetoothSocket 对象，并获得 InputStream 和 OutputStream 来接收和发送数据，客户端则使用 OutputStream 向服务端发送数据。



## 第 24 章 Android 综合案例二—— 月球登陆（游戏）

本章将实现一个月球登陆的程序，一架宇宙飞船飞临月球，并在指定的地点着陆。由于月球重力的原因，需要在适当的时候通过点火来减缓着陆的速度，否则飞船将坠毁。为了不偏离着陆点，飞船需要来回飞行以便准确地落在着陆点。本例的工程目录是 src\demo\lunar\_lander。

### 24.1 游戏的玩法

本节先介绍一下月球登陆游戏的玩法。运行程序后，会显示图 24.1 所示的游戏界面，按“Up”键或单击图 24.2 所示的“开始”菜单项开始玩游戏。



▲ 图 24.1 游戏主界面



▲ 图 24.2 主界面的选项菜单

开始游戏后，在屏幕上方会出现一艘飞船，如图 24.3 所示。飞船会不断向下运动，在适当时候，按“确定”键喷火以降低降落速度，效果如图 24.4 所示。

如果飞船坠毁或远离着陆点，显示的效果如图 24.5 所示，这时可以通过按“Up”键或“开始”菜单项重新开始游戏。



▲ 图 24.3 飞船开始降落



▲ 图 24.4 飞船喷火以降低降落速度



▲ 图 24.5 飞船坠毁

## 24.2 实现游戏界面

游戏主界面有一个选项菜单，我们可以使用下面的代码为选项菜单添加菜单项。

```
public boolean onCreateOptionsMenu(Menu menu)
{
    super.onCreateOptionsMenu(menu);
    // “开始” 菜单项
    menu.add(0, MENU_START, 0, R.string.menu_start);
    // “停止” 菜单项
    menu.add(0, MENU_STOP, 0, R.string.menu_stop);
    // “暂停” 菜单项
    menu.add(0, MENU_PAUSE, 0, R.string.menu_pause);
    // “继续” 菜单项
    menu.add(0, MENU_RESUME, 0, R.string.menu_resume);
    // “易” 菜单项
    menu.add(0, MENU_EASY, 0, R.string.menu_easy);
    // “中” 菜单项
    menu.add(0, MENU_MEDIUM, 0, R.string.menu_medium);
    // “难” 菜单项
    menu.add(0, MENU_HARD, 0, R.string.menu_hard);
    return true;
}
```

LunarView 类负责绘制游戏背景图以及飞船、喷火等效果，包括背景图在内的各种图像资源都需要在 LunarView 类的构造方法中初始化，代码如下：

```
public LunarThread(SurfaceHolder surfaceHolder, Context context,
    Handler handler) {
    // 保存一些重要的对象
    mSurfaceHolder = surfaceHolder;
```

```

mHandler = handler;
mContext = context;

Resources res = context.getResources();
// 缓冲游戏图像元素
mLanderImage = context.getResources().getDrawable(
    R.drawable.lander_plain);
mFiringImage = context.getResources().getDrawable(
    R.drawable.lander_firing);
mCrashedImage = context.getResources().getDrawable(
    R.drawable.lander_crashed);

// 装载背景图
mBackgroundImage = BitmapFactory.decodeResource(res,
    R.drawable.earthrise);

// 获得飞船的宽度和高度
mLanderWidth = mLanderImage.getIntrinsicWidth();
mLanderHeight = mLanderImage.getIntrinsicHeight();

// 初始化计速器
mLinePaint = new Paint();
mLinePaint.setAntiAlias(true);
mLinePaint.setARGB(255, 0, 255, 0);

mLinePaintBad = new Paint();
mLinePaintBad.setAntiAlias(true);
mLinePaintBad.setARGB(255, 120, 180, 0);

mScratchRect = new RectF(0, 0, 0, 0);

mWinsInARow = 0;
mDifficulty = DIFFICULTY_MEDIUM;

// 初始化显示飞船 (还没有开始游戏)
mX = mLanderWidth;
mY = mLanderHeight * 2;
mFuel = PHYS_FUEL_INIT;
mDX = 0;
mDY = 0;
mHeading = 0;
mEngineFiring = true;
}

```

在 LunarView 类中有一个 onDraw 方法，用于绘制背景图及其他图像，在该方法中使用如下代码绘制了背景图。

```
canvas.drawBitmap(mBackgroundImage, 0, 0, null);
```

## 24.3 设置游戏难度

月球登陆游戏可以设置 3 种难度：易、中和难，其中难度“中”是默认值。通过游戏主界面的



选项菜单可以设置不同的游戏难度，设置游戏难度的代码如下：

```
// 将游戏难度设置为“易”
mLunarThread.setDifficulty(LunarThread.DIFFICULTY_EASY);
// 将游戏难度设置为“中”
mLunarThread.setDifficulty(LunarThread.DIFFICULTY_MEDIUM);
// 将游戏难度设置为“难”
mLunarThread.setDifficulty(LunarThread.DIFFICULTY_HARD);
```

实际上，游戏越难，就是指飞船的燃料越少（速度和旋转角度也会有不同），如果燃料不够，飞船就无法升空，也就是说，可能会坠毁。下面的代码设置了“易”和“难”的游戏难度。

```
if (mDifficulty == DIFFICULTY_EASY)
{
    // “易”级别时，燃料比“中”级别的多50%
    mFuel = mFuel * 3 / 2;
    mGoalWidth = mGoalWidth * 4 / 3;
    mGoalSpeed = mGoalSpeed * 3 / 2;
    mGoalAngle = mGoalAngle * 4 / 3;
    speedInit = speedInit * 3 / 4;
}
else if (mDifficulty == DIFFICULTY_HARD)
{
    // “难”级别时，燃料是“中”级别的7/8
    mFuel = mFuel * 7 / 8;
    mGoalWidth = mGoalWidth * 3 / 4;
    mGoalSpeed = mGoalSpeed * 7 / 8;
    speedInit = speedInit * 4 / 3;
}
```

## 24.4 开始游戏

当按“Up”键或单击选项菜单中的“开始”菜单项，游戏就会开始。在 LunarView 类中有一个 doStart 方法，该方法用于开始游戏，并设置一些游戏中使用的参数，代码如下：

```
public void doStart()
{
    synchronized (mSurfaceHolder)
    {
        // 首先设置游戏为中等难度
        mFuel = PHYS_FUEL_INIT;
        mEngineFiring = false;
        mGoalWidth = (int) (mLanderWidth * TARGET_WIDTH);
        mGoalSpeed = TARGET_SPEED;
        mGoalAngle = TARGET_ANGLE;
        int speedInit = PHYS_SPEED_INIT;

        // 调整游戏难度为“易”或“难”
        if (mDifficulty == DIFFICULTY_EASY)
        {
            mFuel = mFuel * 3 / 2;
```



## Android 开发权威指南

```

mGoalWidth = mGoalWidth * 4 / 3;
mGoalSpeed = mGoalSpeed * 3 / 2;
mGoalAngle = mGoalAngle * 4 / 3;
speedInit = speedInit * 3 / 4;
}
else if (mDifficulty == DIFFICULTY_HARD)
{
    mFuel = mFuel * 7 / 8;
    mGoalWidth = mGoalWidth * 3 / 4;
    mGoalSpeed = mGoalSpeed * 7 / 8;
    speedInit = speedInit * 4 / 3;
}

// 设置飞船的开始位置
mX = mCanvasWidth / 2;
mY = mCanvasHeight - mLanderHeight / 2;

// 随机设置飞船的动作
mDY = Math.random() * -speedInit;
mDX = Math.random() * 2 * speedInit - speedInit;
mHeading = 0;

// 计算着陆点
while (true)
{
    mGoalX = (int) (Math.random() * (mCanvasWidth - mGoalWidth));
    if (Math.abs(mGoalX - (mX - mLanderWidth / 2)) > mCanvasHeight / 6)
        break;
}
mLastTime = System.currentTimeMillis() + 100;
setState(STATE_RUNNING);
}
}

```

通过线程不断绘制各种状态的游戏界面。在线程中通过 SurfaceHolder.lockCanvas 方法获得 Canvas 对象，并在 Canvas 上绘制各种图像。线程中 run 方法的代码如下：

```

public void run()
{
    while (mRun)
    {
        Canvas c = null;
        try
        {
            // 获得 Canvas 对象
            c = mSurfaceHolder.lockCanvas(null);
            synchronized (mSurfaceHolder)
            {
                if (mMode == STATE_RUNNING)
                {
                    // 如果游戏正在运行，不断更新游戏中各种角色的状态
                    updatePhysics();
                }
                // 绘制游戏的图像
            }
        }
    }
}

```

```
        doDraw(c);
    }
}
finally
{
    if (c != null)
    {
        mSurfaceHolder.unlockCanvasAndPost(c);
    }
}
}
}
```

上面代码中涉及一个 `onDraw` 方法，该方法是游戏的核心，用于绘制背景、飞船、喷火等图像，代码如下：

```
private void doDraw(Canvas canvas)
{
    // 绘制背景图
    canvas.drawBitmap(mBackgroundImage, 0, 0, null);

    int yTop = mCanvasHeight - ((int) mY + mLanderHeight / 2);
    int xLeft = (int) mX - mLanderWidth / 2;

    // 绘制燃料指示器
    int fuelWidth = (int) (UI_BAR * mFuel / PHYS_FUEL_MAX);
    mScratchRect.set(4, 4, 4 + fuelWidth, 4 + UI_BAR_HEIGHT);
    canvas.drawRect(mScratchRect, mLinePaint);

    // 绘制计速器
    double speed = Math.sqrt(mDX * mDX + mDY * mDY);
    int speedWidth = (int) (UI_BAR * speed / PHYS_SPEED_MAX);

    if (speed <= mGoalSpeed)
    {
        mScratchRect.set(4 + UI_BAR + 4, 4,
                         4 + UI_BAR + 4 + speedWidth, 4 + UI_BAR_HEIGHT);
        canvas.drawRect(mScratchRect, mLinePaint);
    }
    else
    {

        mScratchRect.set(4 + UI_BAR + 4, 4,
                         4 + UI_BAR + 4 + speedWidth, 4 + UI_BAR_HEIGHT);
        canvas.drawRect(mScratchRect, mLinePaintBad);
        int goalWidth = (UI_BAR * mGoalSpeed / PHYS_SPEED_MAX);
        mScratchRect.set(4 + UI_BAR + 4, 4, 4 + UI_BAR + 4 + goalWidth,
                         4 + UI_BAR_HEIGHT);
        canvas.drawRect(mScratchRect, mLinePaint);
    }

    // 绘制着陆点
    canvas.drawLine(mGoalX, 1 + mCanvasHeight - TARGET_PAD_HEIGHT,
                    mGoalX + mGoalWidth, 1 + mCanvasHeight - TARGET_PAD_HEIGHT,
```

```

        mLinePaint);

    canvas.save();
    // 设置飞船的绘制角度
    canvas.rotate((float) mHeading, (float) mX, mCanvasHeight
        - (float) mY);
    if (mMode == STATE_LOSE)
    {
        mCrashedImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop
            + mLanderHeight);
        mCrashedImage.draw(canvas);
    }
    // 控制飞船喷火
    else if (mEngineFiring)
    {
        mFiringImage.setBounds(xLeft, yTop, xLeft + mLanderwidth, yTop
            + mLanderHeight);
        // 绘制喷火图像
        mFiringImage.draw(canvas);
    }
    else
    {
        mLanderImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop
            + mLanderHeight);
        mLanderImage.draw(canvas);
    }
    canvas.restore();
}

```

## 24.5 控制飞船喷火

喷火由两个变量(mMode 和 mEngineFiring)控制。如果 mEngineFiring 为 true，并且游戏处于运行状态 (mMode == STATE\_RUNNING)，就要通过 doDraw 方法在飞船下方绘制喷火图像。设置喷火状态的代码如下：

```

public void setFiring(boolean firing)
{
    synchronized (mSurfaceHolder)
    {
        // 如果 firing 参数值为 true，表示飞船开始喷火
        mEngineFiring = firing;
    }
}

```

## 24.6 控制飞船改变飞行方向

游戏通过 mRotating 变量控制飞船的方向。如果 mRotating 的值为-1，飞船向左转；如果 mRotating 的值为 1，飞船向右转。在 updatePhysics 中根据 mRotating 的值计算出飞船具体的旋转角度，代码如下：

```

if (mRotating != 0)
{
    // 计算旋转角度
    mHeading += mRotating * (PHYS_SLEW_SEC * elapsed);

    // 将旋转角度控制在 0 至 360 度之间
    if (mHeading < 0)
        mHeading += 360;
    else if (mHeading >= 360)
        mHeading -= 360;
}

```

在计算完飞船的旋转角度后，就可以在 doDraw 方法中使用下面的代码按旋转角度绘制飞船了。

```

canvas.rotate((float) mHeading, (float) mX, mCanvasHeight - (float) mY);
mLanderImage.setBounds(xLeft, yTop, xLeft + mLanderWidth, yTop + mLanderHeight);
mLanderImage.draw(canvas);

```

## 24.7 判断飞船是否成功着陆

在 updatePhysics 方法中判断了飞船是否成功着陆，并根据着陆状态设置当前模式（mMode），代码如下：

```

double yLowerBound = TARGET_PAD_HEIGHT + mLanderHeight / 2
    - TARGET_BOTTOM_PADDING;
if (mY <= yLowerBound)
{
    mY = yLowerBound;

    int result = STATE_LOSE;
    CharSequence message = "";
    Resources res = mContext.getResources();
    double speed = Math.sqrt(mDX * mDX + mDY * mDY);
    // 计算飞船着陆点是否正确
    boolean onGoal = (mGoalX <= mX - mLanderWidth / 2 && mX
        + mLanderWidth / 2 <= mGoalX + mGoalWidth);

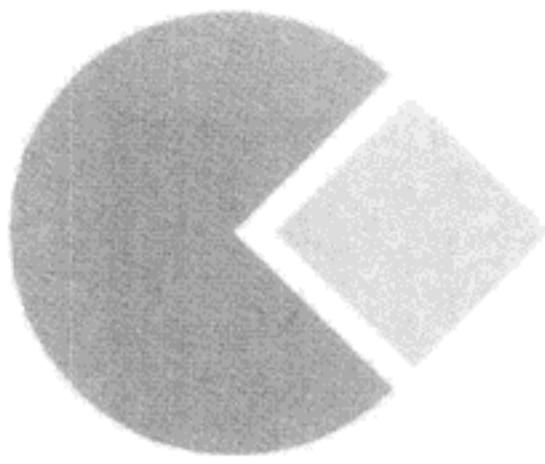
    // 赢得了游戏，重新设置游戏当前状态
    if (onGoal && Math.abs(mHeading - 180) < mGoalAngle
        && speed > PHYS_SPEED_HYPERSPACE)
    {
        result = STATE_WIN;
        mWinsInARow++;
        doStart();
        return;
    }
    else if (!onGoal)
    {
        message = res.getText(R.string.message_off_pad);
    }
    else if (!(mHeading <= mGoalAngle || mHeading >= 360 - mGoalAngle))
    {

```

```
        message = res.getText(R.string.message_bad_angle);
    }
    else if (speed > mGoalSpeed)
    {
        message = res.getText(R.string.message_too_fast);
    }
    else
    {
        // 赢得游戏，重新设置游戏状态
        result = STATE_WIN;
        mWinsInARow++;
    }
    setState(result, message);
}
```

## 24.8 小结

本章介绍了如何使用 SurfaceView 实现一个简单的月球登陆游戏。该游戏利用手机上的物理键控制飞船飞行的方向、是否喷火等动作，并根据最终的着陆点和下降速度判断是否成功着陆。虽然这个游戏很简单，但麻雀虽小，五脏俱全，游戏涉及的基本元素都包含在该游戏中，例如，背景图、游戏角色、角色的各种姿势的变化等。在后面的几章读者将会学习到更复杂的游戏的实现。



## 第 25 章 Android 综合案例三—— 全键盘输入法（应用）

本章将实现一个完整的输入法程序，该输入法包含了小写字母软键盘和数字软键盘，会根据 EditText 控件相应属性值自动切换到小写字母或数字软键盘。通过小写字母软键盘也可手工切换到数字软键盘和大写字母软键盘。本例的工程目录是 src\demo\input\_method。

### 25.1 安装输入法

安装输入法后，进入“语言和键盘”设置页面，选择“我的输入法”后面的复选框，这时会弹出图 25.1 所示的对话框，单击“确定”按钮使用该输入法。

随便找一个带文本输入框的程序，长按文本输入框，在弹出的菜单项中选择“输入法”，然后选择“我的输入法”菜单项，如图 25.2 所示。这样在文本输入框获得焦点时就可以显示输入法软键盘了。



▲ 图 25.1 使用输入法



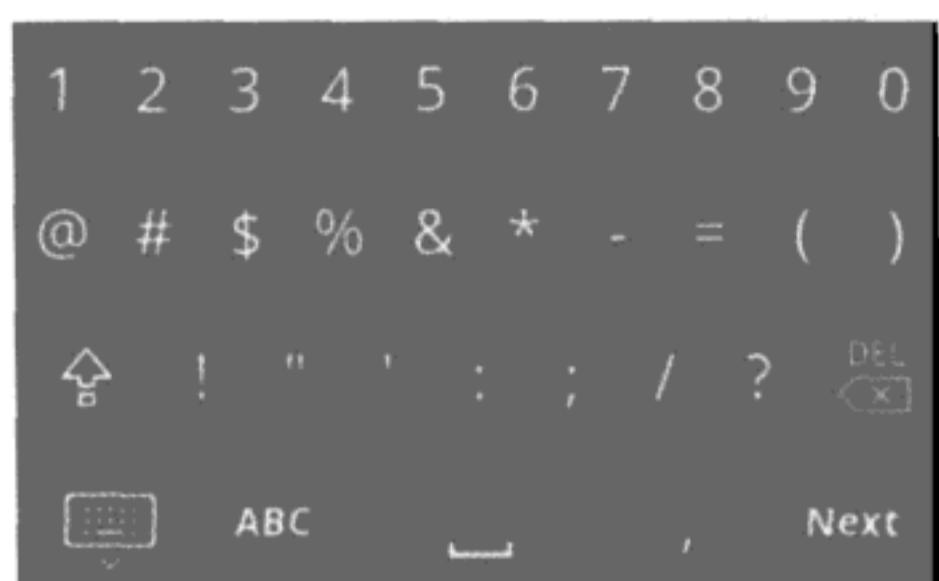
▲ 图 25.2 选择“我的输入法”菜单项

如果当前文本输入框要求输入任意的字符，就会默认弹出小写字母软键盘，如图 25.3 所示。

如果文本输入框只要求输入数字，默认会显示如图 25.4 所示的数字软键盘。



▲ 图 25.3 小写字母软键盘



▲ 图 25.4 数字软键盘

## 25.2 输入法的初始化工作

输入法程序必须有一个服务类，该类必须继承自 `InputMethodService`，本例的服务类是 `SoftKeyboard`。该类的配置代码如下：

```
<service android:name=".SoftKeyboard"
    android:permission="android.permission.BIND_INPUT_METHOD">
    <intent-filter>
        <action android:name="android.view.InputMethod" />
    </intent-filter>
    <meta-data android:name="android.view.im" android:resource="@xml/method" />
</service>
```

输入法服务类的核心方法是 `onCreateInputView`，该方法返回一个 `View` 对象，该对象实际上就是软键盘的界面。下面先看一下 `onCreateInputView` 方法的代码。

```
public View onCreateInputView()
{
    // 装载软键盘布局文件
    mInputView = (KeyboardView) getLayoutInflater().inflate(R.layout.input, null);
    mInputView.setOnKeyboardActionListener(this);
    mInputView.setKeyboard(mQwertyKeyboard);
    return mInputView;
}
```

在 `onCreateInputView` 方法中通过装载一个布局文件（`input.xml`）来创建一个 `View` 对象，这个布局文件就是软键盘的界面。`input.xml` 文件的代码如下：

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

<mobile.android.demo.input.method.LatinKeyboardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/keyboard"
    android:layout_alignParentBottom="true"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

上面的代码只定义了一个控件：LatinKeyboardView，该控件用于显示软键盘的界面，代码如下：

```
package mobile.android.demo.input.method;

import android.content.Context;
import android.inputmethodservice.KeyboardView;
import android.util.AttributeSet;

public class LatinKeyboardView extends KeyboardView
{
    static final int KEYCODE_OPTIONS = -100;
    public LatinKeyboardView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
    }
    public LatinKeyboardView(Context context, AttributeSet attrs, int defStyle)
    {
        super(context, attrs, defStyle);
    }
}
```

LatingKeyboardView 类并没有实际的代码，只是一个继承自 KeyboardView 的类。KeyboardView 是一个拥有标准键盘的类，如果只想显示标准的软键盘，直接从 KeyboardView 类继承即可。

### 25.3 响应键盘操作

软键盘有两种类型的按键：输入按键（用于输入数字、符号和字母）和功能按键，要想处理这两种按键的动作，输入法服务类需要实现 KeyboardView.OnKeyboardActionListener 接口。处理按键的方法是 onKey，代码如下：

```
public void onKey(int primaryCode, int[] keyCodes)
{
    // 处理符号按键
    if (isWordSeparator(primaryCode))
    {
        if (mComposing.length() > 0)
        {
            commitTyped(getApplicationContext());
        }
        sendKey(primaryCode);
        updateShiftKeyState(getApplicationContext());
    }
    // 处理删除按键
    else if (primaryCode == Keyboard.KEYCODE_DELETE)
    {
        handleBackspace();
    }
    // 处理 Shift 按键
    else if (primaryCode == Keyboard.KEYCODE_SHIFT)
```

```

    {
        handleShift();
    }
    // 处理 Cancel 按键
    else if (primaryCode == Keyboard.KEYCODE_CANCEL)
    {
        handleClose();
        return;
    }
    else
    {
        // 处理输入按键
        handleCharacter(primaryCode, keyCodes);
    }
}

```

上面代码中涉及一些处理功能按键的方法，例如，`handleBackspace`（处理退格动作）、`handleShift`（处理 Shift 键）、`handleClose`（用于关闭软键盘）。下面看看这几个方法的实现。

### handleBackspace 方法

```

private void handleBackspace()
{
    final int length = mComposing.length();
    // 如果文本输入框中已经输入至少一个字符，将最后一个字符删除
    if (length > 1)
    {
        mComposing.delete(length - 1, length);
        getCurrentInputConnection().setComposingText(mComposing, 1);
        updateCandidates();
    }
    else if (length > 0)
    {
        // 如果文本输入框中还没有输入任何字符，直接清空文本输入框
        mComposing.setLength(0);
        getCurrentInputConnection().commitText("", 0);
        updateCandidates();
    }
    else
    {
        keyDownUp(KeyEvent.KEYCODE_DEL);
    }
    updateShiftKeyState(getCurrentInputEditorInfo());
}

```

### handleShift 方法

```

private void handleShift()
{
    if (mInputView == null)
    {
        return;
    }
    Keyboard currentKeyboard = mInputView.getKeyboard();
    if (mQwertyKeyboard == currentKeyboard)

```

```

{
    // 切换到字母键盘
    checkToggleCapsLock();
    mInputView.setShifted(mCapsLock || !mInputView.isShifted());
}
else if (currentKeyboard == mSymbolsKeyboard)
{
    // 切换到符号键盘
    mSymbolsKeyboard.setShifted(true);
    mInputView.setKeyboard(mSymbolsShiftedKeyboard);
    mSymbolsShiftedKeyboard.setShifted(true);
}
else if (currentKeyboard == mSymbolsShiftedKeyboard)
{
    mSymbolsShiftedKeyboard.setShifted(false);
    mInputView.setKeyboard(mSymbolsKeyboard);
    mSymbolsKeyboard.setShifted(false);
}
}
}

```

通过 handleShift 方法，还可以切换到图 25.5 所示的符号键盘。



▲ 图 25.5 符号键盘

### handleClose 方法

```

private void handleClose()
{
    commitTyped(getApplicationContext());
    requestHideSelf(0);
    // 关闭软键盘
    mInputView.closing();
}

```

## 25.4 根据 EditText 控件的属性显示不同的软键盘

如果将<EditText>标签的 android:inputType 属性设为“number”，当 EditText 控件处于焦点时，会显示数字键盘；否则，会显示输入字母的键盘。当 EditText 控件获得焦点时，会调用输入法服务的 onStartInput 方法，在该方法中会根据当前文本输入框的输入条件显示不同的软键盘。onStartInput 方法的代码如下。

## Android 开发权威指南

```
public void onStartInput(EditorInfo attribute, boolean restarting)
{
    super.onStartInput(attribute, restarting);

    // 清空输入缓冲区
    mComposing.setLength(0);
    updateCandidates();

    if (!restarting)
    {
        // 清空 Shift 状态
        mMetaState = 0;
    }

    mPredictionOn = false;
    mCompletionOn = false;
    mCompletions = null;

    switch (attribute.inputType & EditorInfo.TYPE_MASK_CLASS)
    {
        case EditorInfo.TYPE_CLASS_NUMBER:
        case EditorInfo.TYPE_CLASS_DATETIME:
            // 当文本输入框要求输入数字或日期时，显示数字软键盘
            mCurKeyboard = mSymbolsKeyboard;
            break;

        case EditorInfo.TYPE_CLASS_PHONE:
            // 显示符号键盘
            mCurKeyboard = mSymbolsKeyboard;
            break;

        case EditorInfo.TYPE_CLASS_TEXT:
            // 显示输入字母的软键盘
            mCurKeyboard = mQwertyKeyboard;
            mPredictionOn = true;

            int variation = attribute.inputType
                & EditorInfo.TYPE_MASK_VARIATION;
            if (variation == EditorInfo.TYPE_TEXT_VARIATION_PASSWORD
                || variation == EditorInfo.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD)
            {
                mPredictionOn = false;
            }

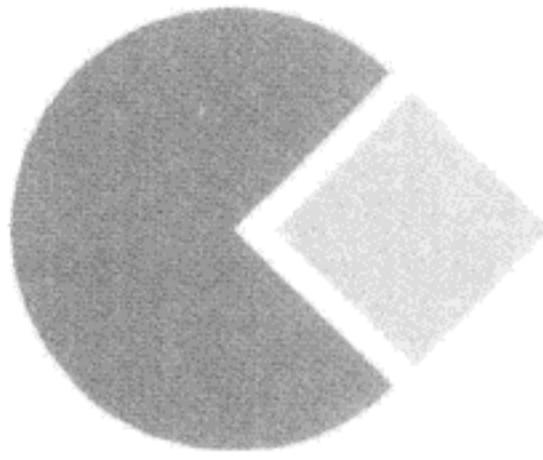
            if (variation == EditorInfo.TYPE_TEXT_VARIATION_EMAIL_ADDRESS
                || variation == EditorInfo.TYPE_TEXT_VARIATION_URI
                || variation == EditorInfo.TYPE_TEXT_VARIATION_FILTER)
            {
                mPredictionOn = false;
            }

            if ((attribute.inputType & EditorInfo.TYPE_TEXT_FLAG_AUTO_COMPLETE) != 0)
            {
                mPredictionOn = false;
            }
    }
}
```

```
mCompletionOn = isFullscreenMode();  
}  
  
// 更新 Shift 状态  
updateShiftKeyState(attribute);  
break;  
  
default:  
    // 对于所有未知类型，显示字母输入键盘  
    mCurKeyboard = mQwertyKeyboard;  
    updateShiftKeyState(attribute);  
}  
  
mCurKeyboard.setImeOptions(getResources(), attribute.imeOptions);  
}
```

## 25.5 小结

本章介绍了一个完整的全键盘输入法的实现。由于本程序的代码较多，因此，只给出了部分的核心代码，完整的实现请参阅随书光盘的源代码。这个输入法程序可以根据 EditText 控件的设置显示字母软键盘或数字软键盘，还可切换到符号软键盘。



## 第 26 章 Android 综合案例四—— 贪吃蛇（游戏）

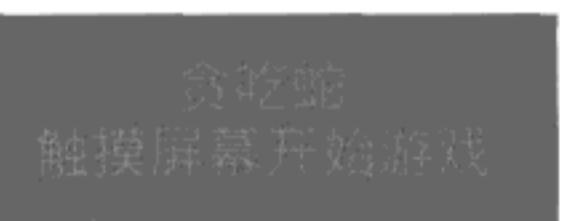
贪吃蛇是一款很经典的小游戏，在PC及很多手机平台都有实现，本章将介绍如何实现Android版的贪吃蛇游戏。由于有的手机没有物理键盘，所以本章实现的贪吃蛇游戏完全采用了触摸屏幕的方式。本例的工程目录是src\demo\snake。

### 26.1 游戏玩法

运行游戏程序，在屏幕中心会显示如图 26.1 所示的信息。触摸屏幕的任意位置后开始游戏。

游戏开始后，会显示如图 26.2 所示的主界面，其中屏幕中间由 6 个小点组成的是小蛇，其他两个是蛇的食物。

小蛇每吃掉一块食物，会积一分，如果小蛇没有吃到食物而撞到了四周的墙壁上，游戏就结束了，并会显示本次游戏的积分，如图 26.3 所示，触摸屏幕游戏会再次开始。通过触摸屏幕的 4 个区域（上、下、左、右），小蛇就会转向这个方向。



▲ 图 26.1 开始游戏



▲ 图 26.2 游戏主界面



▲ 图 26.3 游戏结束

## 26.2 游戏主界面设计

Snake类是游戏的主类。在该类的onCreate方法中初始化了游戏主界面，代码如下：

```
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // 装载游戏主界面的布局文件
    setContentView(R.layout.snake_layout);
    // 获得显示游戏界面的View对象
    mSnakeView = (SnakeView) findViewById(R.id.snake);
    mSnakeView.setTextView((TextView) findViewById(R.id.text));

    if (savedInstanceState == null)
    {
        // 准备新游戏
        mSnakeView.setMode(SnakeView.READY);
    }
    else
    {
        Bundle map = savedInstanceState.getBundle(ICICLE_KEY);
        if (map != null)
        {
            mSnakeView.restoreState(map);
        }
        else
        {
            mSnakeView.setMode(SnakeView.PAUSE);
        }
    }
}
```

上面的代码装载了一个snake\_layout.xml布局文件，该布局文件是游戏主界面的布局，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <!-- 游戏画布 -->
    <mobile.android.demo.snake.SnakeView
        android:id="@+id/snake" android:layout_width="match_parent"
        android:layout_height="match_parent" tileSize="24" />
    <RelativeLayout android:layout_width="match_parent"
        android:layout_height="match_parent">
        <!-- 用于显示游戏状态信息 -->
        <TextView android:id="@+id/text" android:text="@string/snake_layout_text_text"
            android:visibility="visible" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:layout_centerInParent="true"
            android:gravity="center_horizontal" android:textColor="#ff8888ff"
            android:textSize="24sp" />
    </RelativeLayout>
</FrameLayout>
```

`snake_layout.xml` 文件中有两个控件，其中 `SnakeView` 是绘制游戏图像的控件，而 `TextView` 用于显示游戏中的各种状态信息，如游戏开始、游戏结束等，图 26.1 和图 26.3 的信息就是由该控件显示的。由于这两个控件都放在了`<FrameLayout>`标签中，而且 `TextView` 控件在 `SnakeView` 控件上面，因此可以通过隐藏和显示 `TextView` 控件来显示和隐藏状态信息。

在 `SnakeView` 类中通过 `initNewGame` 方法来设置小蛇和食物的初始位置，代码如下：

```
private void initNewGame()
{
    mSnakeTrail.clear();
    mAppleList.clear();
    // 初始化组成小蛇的每个小图像的位置
    mSnakeTrail.add(new Coordinate(7, 7));
    mSnakeTrail.add(new Coordinate(6, 7));
    mSnakeTrail.add(new Coordinate(5, 7));
    mSnakeTrail.add(new Coordinate(4, 7));
    mSnakeTrail.add(new Coordinate(3, 7));
    mSnakeTrail.add(new Coordinate(2, 7));
    // 小蛇默认的移动方向（北）
    mNextDirection = NORTH;

    // 随机设置两块食物的位置
    addRandomApple();
    addRandomApple();
    // 600 毫秒后开始移动
    mMoveDelay = 600;
    // 初始积分为 0
    mScore = 0;
}
```

`SnakeView` 的父类是 `TileView`，在 `TileView` 类的 `onDraw` 方法中绘制了游戏界面四周的围墙，代码如下：

```
public void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    for (int x = 0; x < mXTileCount; x += 1)
    {
        for (int y = 0; y < mYTileCount; y += 1)
        {
            if (mTileGrid[x][y] > 0)
            {
                // 绘制每一个组成围墙的瓦片
                canvas.drawBitmap(mTileArray[mTileGrid[x][y]], mXOffset + x
                    * mTileSize, mYOffset + y * mTileSize, mPaint);
            }
        }
    }
}
```

## 26.3 控制小蛇的移动

游戏中通过触摸屏幕的4个区域（上、下、左、右）控制小蛇的移动方向。处理触摸动作的onTouch方法的代码如下：

```
public boolean onTouchEvent(MotionEvent event)
{
    mWidth = getMeasuredWidth();
    mHeight = getMeasuredHeight();
    if (mMode == RUNNING)
    {
        // 如果触摸位置在下方区域，小蛇向下（南）移动
        if (event.getX() > mWidth / 4 && event.getX() < mWidth * 3 / 4
            && event.getY() < mHeight / 2)
        {
            if (mDirection != SOUTH)
            {
                // 改变移动方向
                mNextDirection = NORTH;
            }
        }
        // 如果触摸位置在上方区域，小蛇向上（北）移动
        else if (event.getX() > mWidth / 4 && event.getX() < mWidth * 3 / 4
                  && event.getY() > mHeight / 2)
        {
            if (mDirection != NORTH)
            {
                // 改变移动方向
                mNextDirection = SOUTH;
            }
        }
        // 如果触摸位置在左侧区域，小蛇向左（西）移动
        else if (event.getX() < mWidth / 4)
        {
            if (mDirection != EAST)
            {
                // 改变移动方向
                mNextDirection = WEST;
            }
        }
        // 如果触摸位置在右侧区域，小蛇向右（东）移动
        else if (event.getX() > (mWidth * 3 / 4))
        {
            if (mDirection != WEST)
            {
                // 改变移动方向
                mNextDirection = EAST;
            }
        }
    }
}
```

## Android 开发权威指南

```

    }
else
{
    // 新开始游戏执行下面的代码

    // 初始化游戏
    initNewGame();
    // 设置游戏模式为正在运行
    setMode(RUNNING);
    // 开始移动小蛇
    update();
    if (mDirection != SOUTH)
    {
        // 设置小蛇的初始移动方向
        mNextDirection = NORTH;
    }
}
return super.onTouchEvent(event);
}

```

上面代码中涉及一个重要的 `update` 方法，该方法负责更新小蛇的移动轨迹，以及绘制其他的图像。该方法的代码如下：

```

public void update()
{
    if (mMode == RUNNING)
    {
        long now = System.currentTimeMillis();
        if (now - mLastMove > mMoveDelay)
        {
            clearTiles();
            updateWalls();
            // 更新小蛇的位置
            updateSnake();
            updateApples();
            mLastMove = now;
        }
        // 产生动画效果
        mRedrawHandler.sleep(mMoveDelay);
    }
}

```

上面代码中的 `updateSnake` 和 `sleep` 方法很重要，其中 `updateSnake` 方法用于更新小蛇移动所需要的信息，而 `sleep` 方法会产生小蛇移动的动作效果。`updateSnake` 方法的实现代码如下：

```

private void updateSnake()
{
    boolean growSnake = false;

    // 获得小蛇的头
    Coordinate head = mSnakeTrail.get(0);
    Coordinate newHead = new Coordinate(1, 1);

    mDirection = mNextDirection;

```

```
// 根据小蛇的移动方向重新确定蛇头
switch (mDirection)
{
    case EAST:
    {
        newHead = new Coordinate(head.x + 1, head.y);
        break;
    }
    case WEST:
    {
        newHead = new Coordinate(head.x - 1, head.y);
        break;
    }
    case NORTH:
    {
        newHead = new Coordinate(head.x, head.y - 1);
        break;
    }
    case SOUTH:
    {
        newHead = new Coordinate(head.x, head.y + 1);
        break;
    }
}

// 检测小蛇是否撞到了四周的墙
if ((newHead.x < 1) || (newHead.y < 1) || (newHead.x > mXTileCount - 2)
    || (newHead.y > mYTileCount - 2))
{
    setMode(LOSE);
    return;
}

// 检测小蛇是否撞到了自己
int snakelength = mSnakeTrail.size();
for (int snakeindex = 0; snakeindex < snakelength; snakeindex++)
{
    Coordinate c = mSnakeTrail.get(snakeindex);
    if (c.equals(newHead))
    {
        setMode(LOSE);
        return;
    }
}

// 检测小蛇是否吃到了食物
int applecount = mAppleList.size();
for (int appleindex = 0; appleindex < applecount; appleindex++)
{
    Coordinate c = mAppleList.get(appleindex);
    if (c.equals(newHead))
    {
        mAppleList.remove(c);
        addRandomApple();
        mScore++;
        mMoveDelay *= 0.9;
        growSnake = true;
    }
}
```

```

        }
    }

    // 为了保持蛇的长度不变，改变蛇头的位置
    if (!growSnake)
    {
        mSnakeTrail.remove(mSnakeTrail.size() - 1);
    }

    int index = 0;
    for (Coordinate c : mSnakeTrail)
    {
        if (index == 0)
        {
            setTile(YELLOW_STAR, c.x, c.y);
        }
        else
        {
            setTile(RED_STAR, c.x, c.y);
        }
        index++;
    }
}

```

`sleep` 方法在 `RefreshHandler` 类中，该类继承自 `Handler`。通过发送消息使当前的 `View` 不断刷新，从而产生动作效果。`RefreshHandler` 类的代码如下：

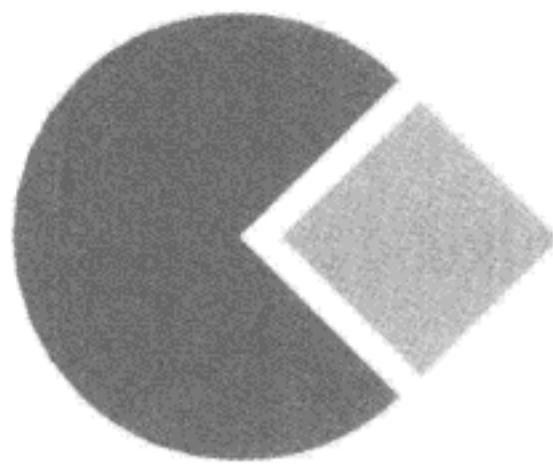
```

class RefreshHandler extends Handler
{
    @Override
    public void handleMessage(Message msg)
    {
        // 更新小蛇的移动状态
        SnakeView.this.update();
        // 刷新当前 View
        SnakeView.this.invalidate();
    }
    public void sleep(long delayMillis)
    {
        this.removeMessages(0);
        // 发送消息
        sendMessageDelayed(obtainMessage(0), delayMillis);
    }
};

```

## 26.4 小结

本章介绍了一个简单的贪吃蛇的游戏。该游戏使用 6 个连在一起的小图像表示一条小蛇，使用另外两个小图像分别表示小蛇要吃的两块食物。如果小蛇吃了某块食物，会在另一个地方随机增加一块食物，也就是说，屏幕上始终保持两块食物。当小蛇撞到了周围的墙壁，游戏结束，并显示本次游戏的积分。



## 第 27 章 Android 综合案例五—— 新浪微博客户端（应用）

微博是最近几年比较火的 Web 应用，这个源于 Twitter 的应用目前在各个国家已经有了众多的模仿者。国内的新浪微博是最早的实践者，新浪微博在一开始推出的时候就开放了 API 接口，因此，也涌现了大量的第三方应用。本章将结合新浪开放 API 实现一个 Android 版的微博客户端。本例的工程目录是 src\demo\sina\_microblog。

### 27.1 新微博简介

新浪微博的首页是 <http://t.sina.com.cn> 或 [weibo.com](http://weibo.com)。本节将介绍新浪微博手机客户端以及微博开放 API。读者可以通过本节的介绍基本了解新浪微博客户端的使用方法及开放 API 的细节。

#### 27.1.1 新微博客户端

目前已经有非常多的新浪微博客户端。新浪官方有一个 Android 版本的客户端，如图 27.1 所示是官方微博客户端的主界面。



▲ 图 27.1 新微博客户端主界面

Android 版的客户端可以从下面的地址下载。

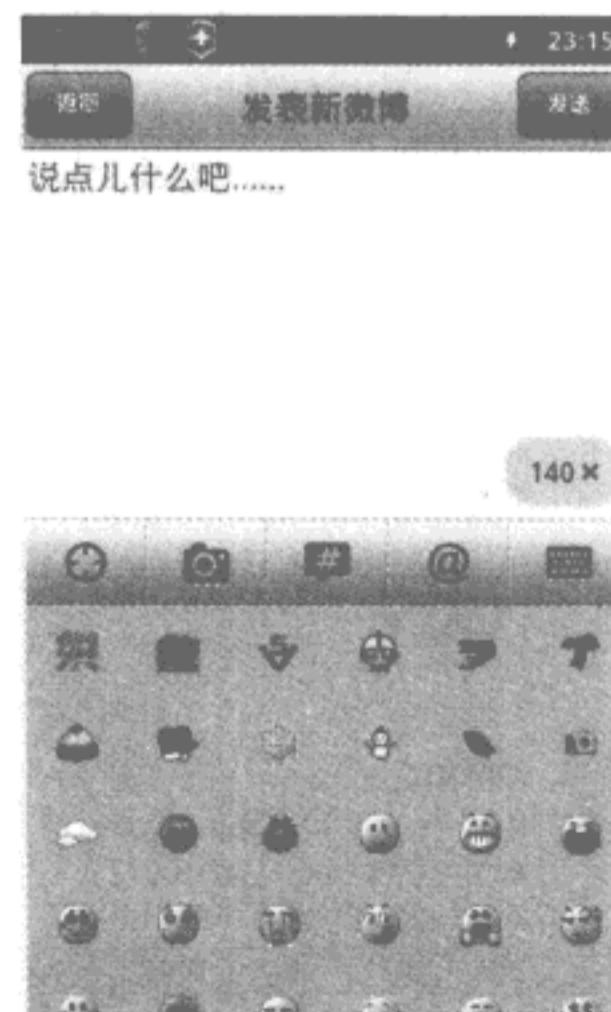
| <http://t.sina.com.cn/mobile/android.php>

单击主界面右上角的按钮可以刷新当前的微博，单击主界面左上角的按钮可以写微博，如图 27.2 所示。



▲ 图 27.2 写微博

在写微博界面中间有一排按钮，分别是通过 GPS 获得当前位置、获得照片（通过手机摄像头或手机相册获得照片）、插入话题、插入@符号和插入笑脸表情。单击插入表情按钮，会显示如图 27.3 所示的表情列表。



▲ 图 27.3 插入表情

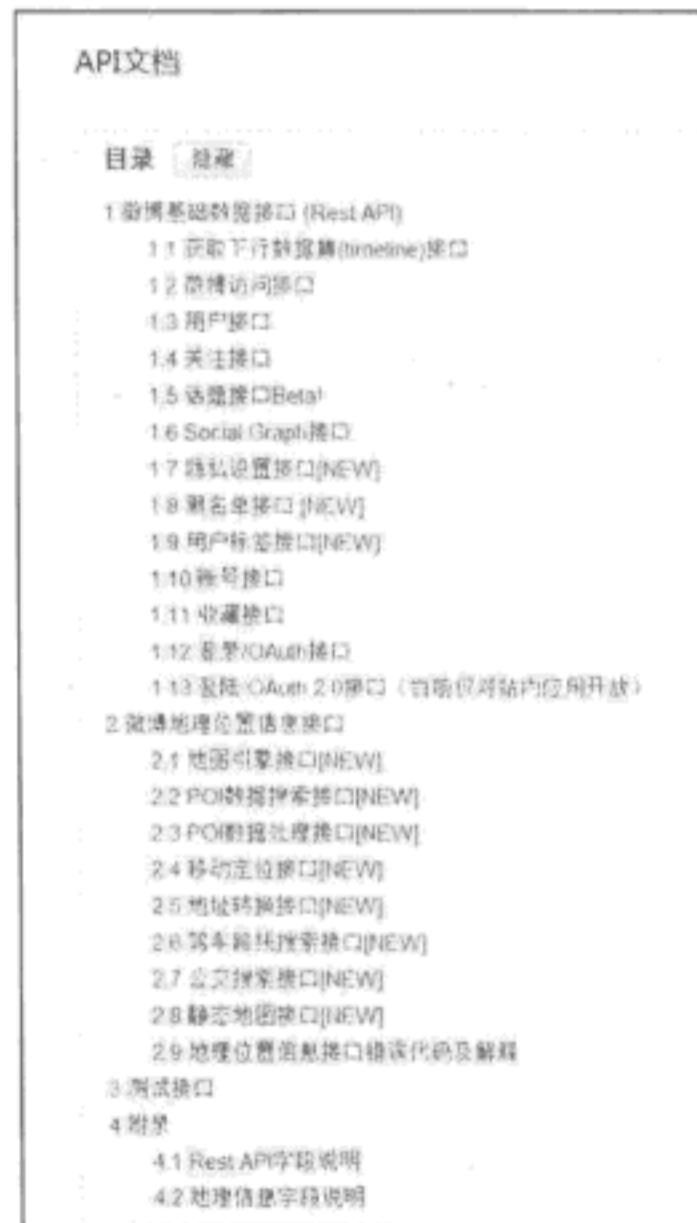
本章的例子将实现类似官方微博客户端的主要功能，包括浏览微博首页、我的微博等内容，以及发微博、评论微博等。

### 27.1.2 新浪微博开放 API

新浪微博 API 规范可以从下面地址查看。

<http://open.t.sina.com.cn/wiki/index.php/API%E6%96%87%E6%A1%A3>

进入开放 API 规范页面，会看到微博包含的各种 API 目录，主要使用的 API 包括“微博访问 API”、“用户 API”、“私信 API”、“账号 API”等。完整的 API 目录列表如图 27.4 所示。



▲ 图 27.4 完整的新浪微博开放 API 目录列表

开放 API 通过 HTTP 访问，数据格式分为两种：json 和 xml。我们可以自己通过 HTTP 访问这些 API，也可以利用新浪提供的 SDK 访问这些 API。SDK 将在 27.2 节介绍。

## 27.2 使用新浪微博开发 API

新浪提供了超过 10 种 SDK，可以适应各个操作系统平台，我们可以选择适合自己的 SDK。对于 Android，可以选择 Android 或 Java 平台的 SDK。

在使用新浪微博 API 之前，需要先到如下地址免费创建一个应用。

<http://open.t.sina.com.cn/apps>

在成功创建应用后，会生成与应用程序绑定的 App Key 和 App Secret。将这两个值替换 SDK

中的 Weibo.java 文件中的 CONSUMER\_KEY 和 CONSUMER\_SECRET 常量。

下面使用 Java 控制台程序来测试一下新浪微博 SDK。

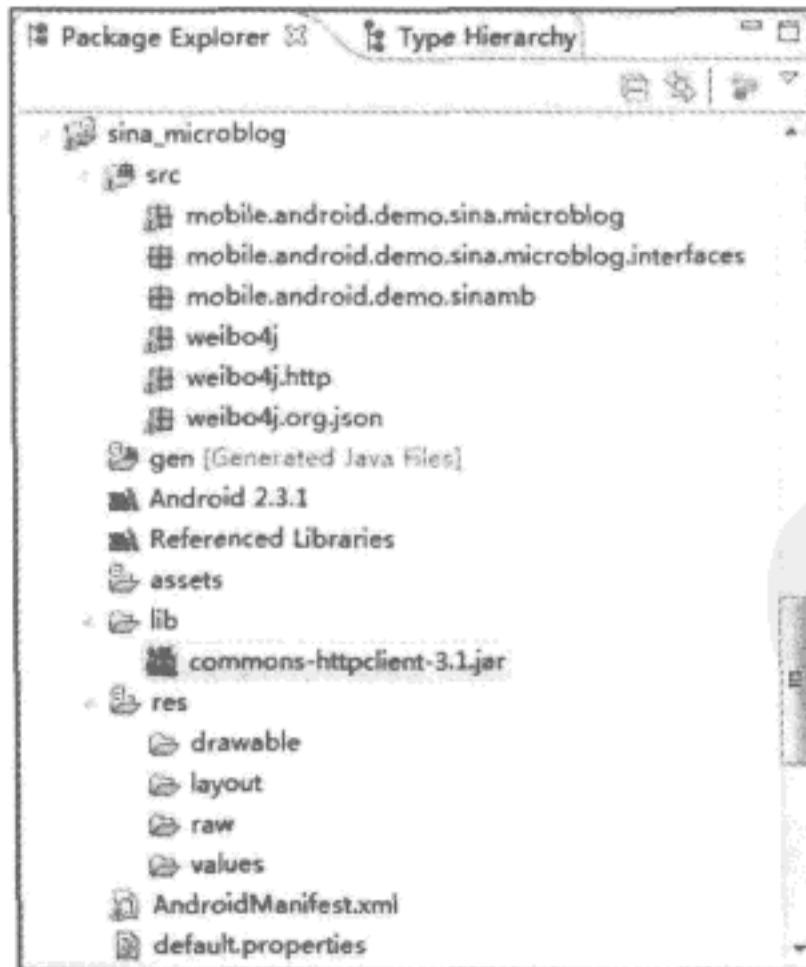
```
Weibo weibo = new Weibo(account, password);
// 获得首页微博信息
List<Status> statusList = weibo.getHomeTimeline();

for (Status status : statusList)
{
    // 输出当前微博用户名、内容、ID 和发布时间
    System.out.print(status.getUser().getName() + ":"
        + status.getText() + ":" + status.getId() + " time:" + status.getCreatedAt());
    System.out.println();
}
```

在运行上面代码之前，需要将 account 和 password 改成读者的新浪微博账号和密码。如果正常输出微博信息，说明微博 SDK 已可以正常使用了。

## 27.3 创建和配置新浪微博客户端工程

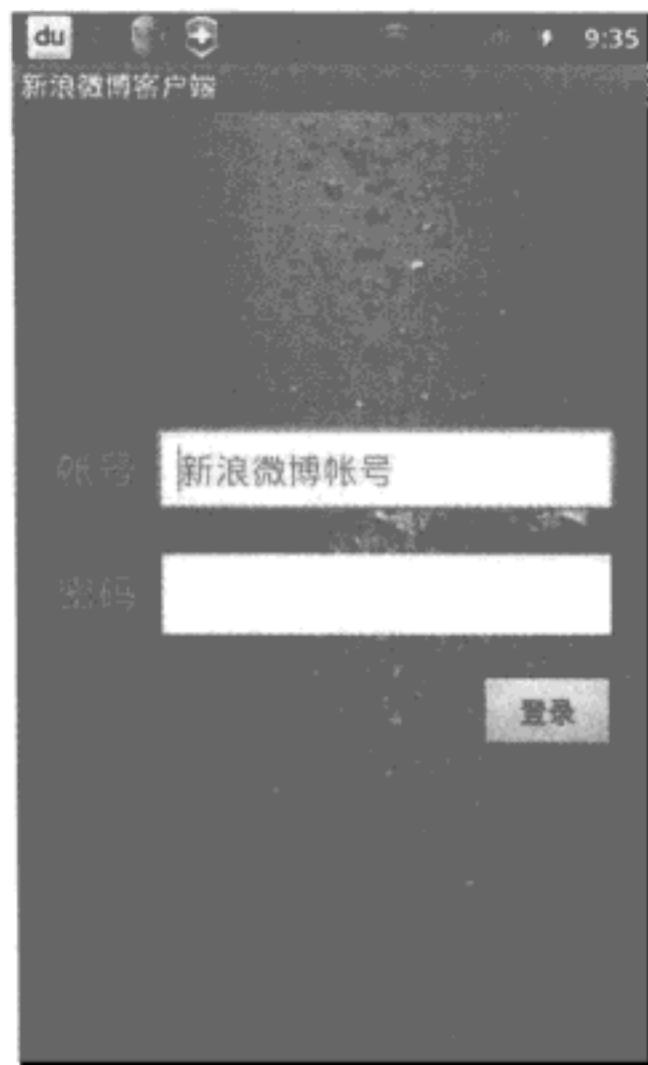
现在新建一个 Android 工程 (sina\_microblog)，并将微博 SDK 的源代码复制到 Android 工程的 src 目录中，最后引用一个 commons-httpclient-3.1.jar 包，sina\_microblog 工程的目录结构如图 27.5 所示（已包括完整的包和类）。



▲ 图 27.5 sina\_microblog 工程的目录结构

## 27.4 登录新浪微博

从本节开始编写新浪微博客户端，先来实现用户登录功能、登录界面如图 27.6 所示。



▲ 图 27.6 登录界面

如图 27.6 所示的界面很简单，与用户交互的控件只有两个 EditText 控件和一个按钮控件，布局文件如下：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:gravity="center"
    android:background="@drawable/login_background">
    <LinearLayout android:id="@+id/l1Account"
        android:orientation="horizontal" android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="@string/account"
            android:textSize="@dimen/login_textsize"
            android:layout_marginLeft="@dimen/login_textview_marginleft" />
        <!-- 输入账号的 EditText 控件 -->
        <EditText android:id="@+id/etAccount" android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:layout_marginLeft="@dimen/
            login_edittext_marginleft"
            android:layout_marginRight="@dimen/login_edittext_marginright"
            android:singleLine="true" android:hint="新浪微博账号"/>
    </LinearLayout>
    <LinearLayout android:id="@+id/l1Password"
        android:orientation="horizontal" android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="@dimen/login_account_password_margintop"
        android:layout_below="@+id/l1Account">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="@string/password"
            android:textSize="@dimen/login_textsize"
            android:layout_marginLeft="@dimen/login_textview_marginleft" />
```

```

<!-- 输入密码的 EditText 控件 -->
<EditText android:id="@+id/etPassword" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:layout_marginLeft="@dimen/
    login_edittext_marginleft"
    android:layout_marginRight="@dimen/login_edittext_marginright"
    android:password="true" android:singleLine="true" />
</LinearLayout>
<LinearLayout android:orientation="horizontal"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/login_account_password_margintop"
    android:layout_below="@+id/l1Password" android:gravity="right">
<!--登录按钮 -->
<Button android:id="@+id/btnLogin" android:layout_width="@dimen/login_
    loginbutton_width"
    android:layout_height="@dimen/login_loginbutton_height" android:text="登录"
    android:layout_marginRight="@dimen/login_loginbutton_marginright" />
</LinearLayout>
</RelativeLayout>

```

当输入账号和密码后，单击“登录”按钮，会执行如下的代码来登录新浪微博。

```

// 显示登录状态
mProgressDialog = mUtil.showSpinnerProgressDialog("正在登录..."); 
// 登录服务器
handleLoadSinaMicroBlog();

```

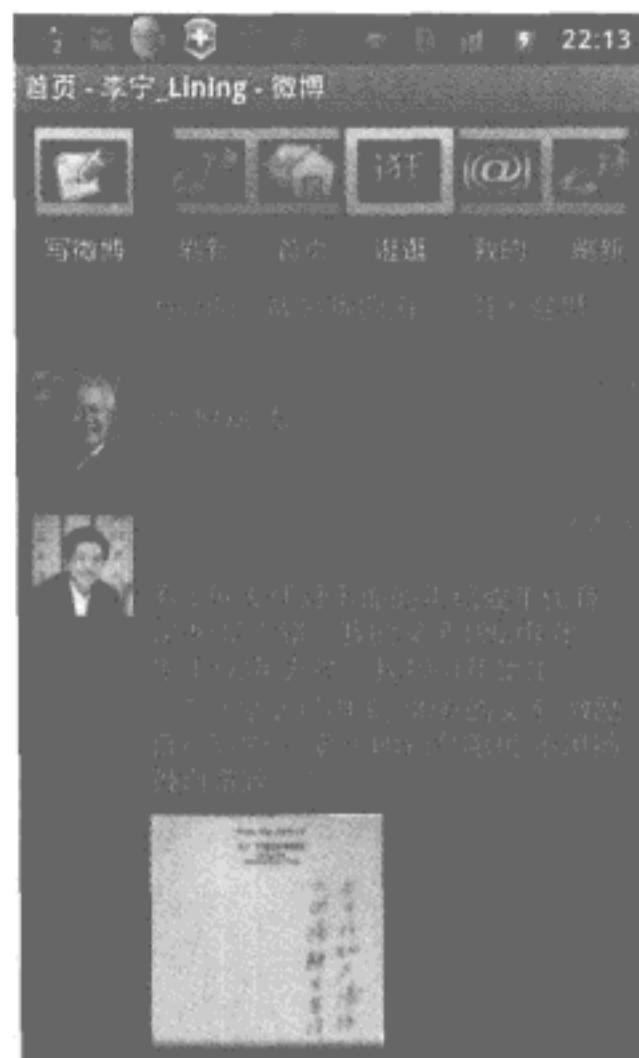
其中 handleLoadSinaMicroBlog 方法处理登录动作，代码如下：

```

private void handleLoadSinaMicroBlog()
{
    try
    {
        // 获得账号
        mAccount = metAccount.getText().toString();
        // 获得密码
        mPassword = metPassword.getText().toString();
        // 创建 MyAsyncWeibo 对象用于登录服务器
        mWeibo = new MyAsyncWeibo(mAccount, mPassword);
        // 验证用户
        mUser = mWeibo.verifyCredentials();
        // 如果成功登录，保存用户名和密码
        mUtil.saveString(ACCOUNT, mAccount);
        mUtil.saveString(PASSWORD, mPassword);
        mUtil.saveString(USER_NAME, mUser.getName());
        // 如果成功登录，切换到我的首页，将在后面详细介绍该方法
        loadSinaMicroBlogView(null);
    }
    catch (Exception e)
    {
    }
}

```

如果输入的账号的密码正确，会显示如图 27.7 所示的“我的首页”微博。



▲ 图 27.7 “我的首页”微博信息

## 27.5 功能按钮

从如图 27.7 所示的界面可以看出，最上面是一排功能按钮，这些功能按钮包括“写微博”、“我的首页”、“刷新”、“随便看看（逛逛）”、“提到我的”等功能。单击这些按钮，就会执行相应的动作，系统默认会显示“我的微博”中的微博信息，包括自己发表的微博和所关注者发表的微博。

实际上，这些功能按钮是一个 `Gallery` 控件，通过 `FunImageAdapter` 对象获得相应的图像和数据，并可无限循环显示（详见 5.9.4 节的内容）。`FunImageAdapter` 类的代码如下：

```
package mobile.android.demo.sina.microblog;

import android.content.Context;
import android.content.res.TypedArray;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class FunImageAdapter extends BaseAdapter
{
    private LayoutInflater mLayoutInflater;
    private int mGalleryItemBackground;
    // 定义每个功能按钮的图像资源 ID
    private int[] mFunImageId = new int[]
    { R.drawable.home, R.drawable.stroll, R.drawable.me, R.drawable.refresh };
    ...
}
```

## Android 开发权威指南

```
// 定义每个功能按钮的文本
private String[] mFunText = new String[]
{ "首页", "逛逛", "我的", "刷新", };
public FunImageAdapter(Context context)
{
    mLayoutInflater = (LayoutInflater) context
        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    // 初始化 Gallery
    TypedArray typedArray = context
        .obtainStyledAttributes(R.styleable.Gallery);
    mGalleryItemBackground = typedArray.getResourceId(
        R.styleable.Gallery_android_galleryItemBackground, 0);
}

@Override
public int getCount()
{
    return Integer.MAX_VALUE;
}
public int getFunCount()
{
    return mFunImageId.length;
}

public int getIndex(int position)
{
    return mFunImageId[position % mFunImageId.length];
}
@Override
public Object getItem(int position)
{
    return null;
}
@Override
public long getItemId(int position)
{
    return 0;
}
@Override
public View getView(int position, View convertView, ViewGroup parent)
{
    // 装载每一个菜单项的布局文件
    View view = mLayoutInflater.inflate(R.layout.fun_item, null);
    ImageView ivFunction = (ImageView) view.findViewById(R.id.ivFunction);
    TextView tvFunItemText = (TextView) view
        .findViewById(R.id.tvFunItemText);
    // 设置菜单项的图像
    ivFunction.setImageResource(mFunImageId [position
        % mFunImageId.length]);
    // 设置菜单项的文本
    tvFunItemText.setText(mFunText [position % mFunImageId.length]);
    ivFunction.setScaleType(ImageView.ScaleType.FIT_XY);
    // 设置菜单项的背景
    ivFunction.setBackgroundResource(mGalleryItemBackground);
```

```
    return view;
}
```

## 27.6 显示“我的首页”的微博

登录成功后，系统会立刻显示“我的首页”的微博内容，这个功能需要通过 Weibo.getTimelineAsync 方法异步来完成。我们先看看 getTimelineAsync 方法是如何实现的。

```
public void getTimelineAsync(final Paging paging,
final GetTimelineListener listener, final int showType)
throws Exception
{
    Thread thread = new Thread(new Runnable()
    {

        @Override
        public void run()
        {
            try
            {
                List<Status> statusList = null;
                switch (showType)
                {
                    case Const.SHOW_TYPE_HOME:
                        // 获得“我的首页”微博信息
                        statusList = getHomeTimeline(paging);
                        break;
                    case Const.SHOW_TYPE_STROLL:
                        // 获得公共微博信息
                        statusList = getPublicTimeline();
                        break;
                    case Const.SHOW_TYPE_ME:
                        // 获得提到我的微博信息
                        statusList = getMentions(paging);
                        break;
                    case Const.SHOW_TYPE_ME_BLOG:
                        // 获得自己发布的微博信息
                        statusList = getUserTimeline(paging);
                        break;
                }
                // 成功获得微博信息后，在该事件方法中处理获得的微博
                listener.ok(statusList);
            }
            catch (Exception e)
            {
                listener.error(e);
            }
        }
    });
    thread.start();
}
```

}

从 `getTimelineAsync` 的代码可以看出，该方法根据 `showType` 参数值的不同获得了 4 种不同的微博信息。所谓异步，就是将同步获得微博的方法放到一个线程中，当成功获得微博后，通过事件（`ok` 方法）返回所获得的数据。

在获得微博数据后，会使用 `MicroBlogAdapter` 对象将微博数据装载到 `ListView` 控件中，代码如下：

```

try
{
    mHandler.post(new Runnable()
    {
        @Override
        public void run()
        {
            // 装载微博数据到 ListView 控件中
            mMicrBlogAdapter = new MicroBlogAdapter(
                MicroBlog.this, statusList, mWeibo,
                mCache);
        }
    });
    mCache.saveHomeTimeline(statusList);
    mMicrBlogAdapter
        .downloadCacheProfileImageForHomeTimeline();

    mHandler.sendEmptyMessage(Const.PROCESS_TYPE_LOAD_MICROBLOG_VIEW);
    mHandler.sendEmptyMessage(Const.SHOW_TYPE_HOME);
}
catch (Exception e)
{
}

```

在 `MicroBlogAdapter` 类中通过 `getView` 方法显示微博内容、评论数、转发数、发布时间以及用户头像，效果如图 27.7 所示。除了用户头像，其他的内容都是在获得微博信息时获得的，而用户头像在微博信息中只有一个 URL，需要浏览到当前微博时通过这个 URL 异步从网络下载图像。异步下载图像的代码如下：

```

private void loadBitmapFromNetAsync(final long id, final String url,
    final ImageView imageView,
    final OnLoadBitmapFromNetListener listener)
{
    Thread thread = new Thread(new Runnable()
    {
        @Override
        public void run()
        {
            try
            {
                Bitmap bitmap;
                // 将 Url 转换成 hash 码

```

```
int hashCode = url.toString().hashCode();
// 如果这个图像以前未获得过，继续通过网络获得该图像
bitmap = mImageMap.get(hashCode);
if (bitmap == null)
{
    // 开始从网络上装载图像
    InputStream is = Util.getNetInputStream(url.toString());
    bitmap = BitmapFactory.decodeStream(is);
    is.close();
    mImageMap.put(hashCode, bitmap);
    mIdHashCodeMap.put(id, hashCode);

}
if (listener != null)
    listener.onObtainBitmap(bitmap, imageView);
final Bitmap bitmap1 = bitmap;
// 下载完图像后，将该图像显示在 ImageView 控件中
mHandler.post(new Runnable()
{
    @Override
    public void run()
    {
        if (bitmap1 == null)
        {
            imageView.setImageResource(R.drawable.nophoto);
        }
        else if (listener != null)
        {
            listener.onObtainBitmap(bitmap1, imageView);
        }
    }
});
}
catch (Exception e)
{
    mHandler.post(new Runnable()
    {
        @Override
        public void run()
        {
            imageView.setImageResource(R.drawable.nophoto);
            Util.showMsg(mContext, "下载失败");
        }
    });
}
});
thread.start();
}
```

微博头像可能会重复，如果是重复的头像，就没有必要再次下载了，因此，上面的代码将已经成功下载的图像 URL 以 hashCode 的形式放到一个 Map 中（mImageMap），下次再遇到这个 URL，就直接从 Map 中获得该图像了。

## 27.7 评论微博

长按微博列表，会弹出一个菜单，单击“评论”菜单项，会弹出一个如图 27.8 所示的评论当前微博的对话框。



▲ 图 27.8 评论当前微博的对话框

单击“发布”按钮，会执行下面的代码来评论当前的微博。

```
public void onClick(View view)
{
    String msg = metComment.getText().toString().trim();
    if ("".equals(msg))
        return;
    mProgressDialog = Util.showSpinnerProgressDialog(this, "正在发布评论...");

    // 异步发表评论
    MicroBlog.mWeibo.updateCommentAsync(msg, mStatusId, null,
        new OnUpdateCommentListener()
    {
        @Override
        public void onSuccess(Comment comment)
        {
            // 成功发表评论，显示发布成功提示信息框
            mHandler.sendMessage(mHandler.obtainMessage());
        }

        @Override
        public void onException(Exception e)
        {
            Message message = new Message();
            message.obj = e.getMessage();
            message.what = 1;
            mHandler.sendMessage(message);
        }
    });
}
```

上面代码中使用了 `updateCommentAsync` 方法异步发布了评论，该方法的基本原理也是利用多线程执行同步评论的方法，代码如下：

```
| public void updateCommentAsync(final String msg, final String id,
```

```

    final String cid, final OnUpdateCommentListener listener)
{
    Thread thread = new Thread(new Runnable()
    {

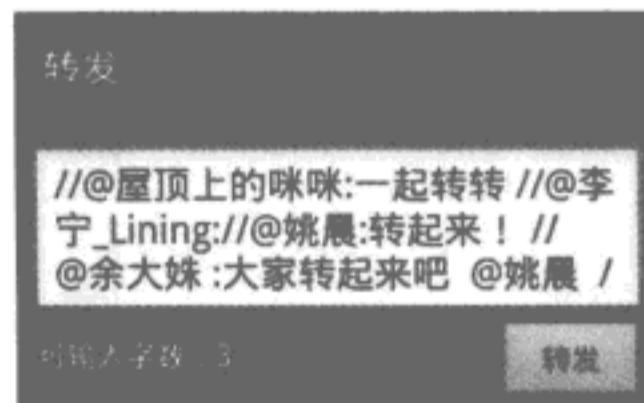
        @Override
        public void run()
        {
            try
            {
                // 执行同步的发布评论
                Comment comment = updateComment(msg, id, cid);
                listener.onSuccess(comment);

            }
            catch (Exception e)
            {
                listener.onException(e);
            }
        }
    });
    thread.start();
}

```

## 27.8 转发微博

微博转发与微博评论类似，只是如果当前微博已经有很多人转发了，一般需要将这些转发的内容先显示在 EditText 控件中，如图 27.9 所示。



▲ 图 27.9 转发微博

下面的代码用于装载当前微博已被转发的内容。

```

// 从主界面传入的当前微博内容
mText = getIntent().getExtras().getString("text");
mStatusUserName = getIntent().getExtras().getString("username");
metRepost.setText("//@" + mStatusUserName + ":" + mText);

```

单击“转发”按钮，会将当前微博作为自己的微博发布，在“我的首页”可以看到该微博，代码如下：

```
public void onClick(View view)
```

```

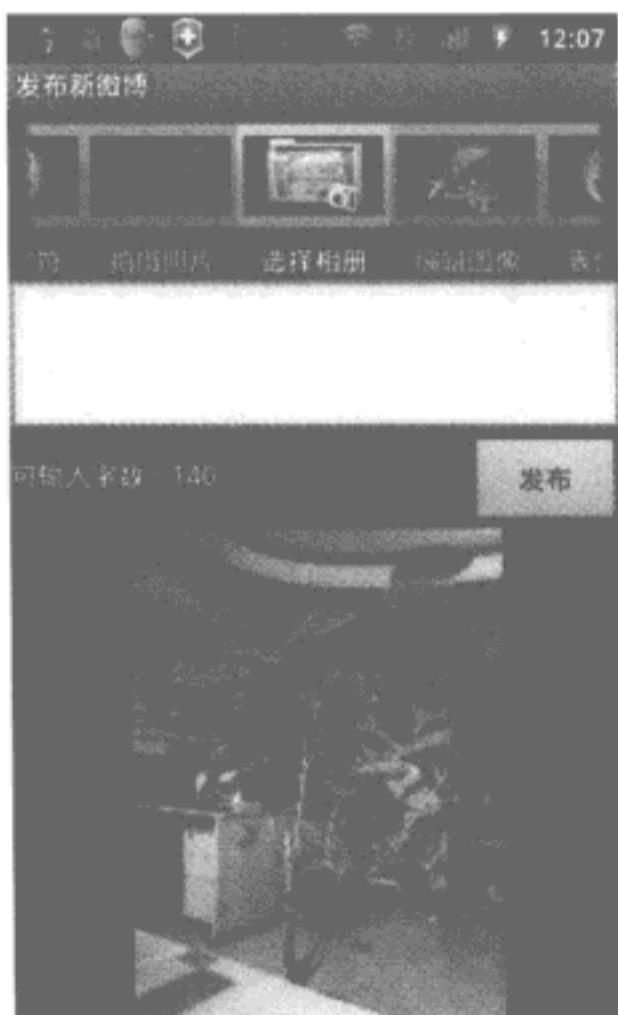
{
    String msg = metRepost.getText().toString().trim();
    if ("".equals(msg))
        msg = "转发微博";

    mProgressDialog = Util.showSpinnerProgressDialog(this, "正在转发...");
    // 异步转发微博
    MicroBlog.mWeibo.updateStatusAsync(msg, mStatusId,
        new OnStatusListener()
    {
        @Override
        public void onSuccess(Status status)
        {
            // 成功转发，显示“转发成功”信息框
            mHandler.sendEmptyMessage(0);
        }
        @Override
        public void onException(Exception e)
        {
            Message message = new Message();
            message.obj = e.getMessage();
            message.what = 1;
            mHandler.sendMessage(message);
        }
    });
}

```

## 27.9 写微博

单击主界面左上角的按钮，会显示当前写微博的界面。在该界面中除了可以写微博外，还可以选择相册中的照片、拍照、选择表情等，拍照效果如图 27.10 所示，选择表情效果如图 27.11 所示。



▲ 图 27.10 “写微博”主界面



▲ 图 27.11 选择表情

单击如图 27.10 所示界面的“发布”按钮后，会执行下面的代码来发布微博。在发布微博时会考虑到用户是否选择了图像或拍了照。如果包含了图像，会连同图像一起作为微博发布。

```
public void onClick(View view)
{
    switch (view.getId())
    {
        case R.id.btnPostNewMicroBlog:
            String content = metWriteMicroBlog.getText().toString().trim();
            // 如果包含图像，但未输入微博内容，设置默认微博文本
            if (mCameraBitmap != null && "".equals(content))
            {
                content = "分享图片。";
            }
            if (!"".equals(content))
            {
                mProgressDialog = Util.showSpinnerProgressDialog(this,
                        "正在发布微博...");

                try
                {
                    if (mCameraBitmap == null)
                    {
                        // 异步发布微博（不含图像）
                        MicroBlog.mWeibo.updateStatusAsync(content,
                                new OnStatusListener()
                                {
                                    @Override
                                    public void onSuccess(Status status)
                                    {
                                        mHandler.sendMessage(1);
                                    }

                                    @Override
                                    public void onException(Exception e)
                                    {

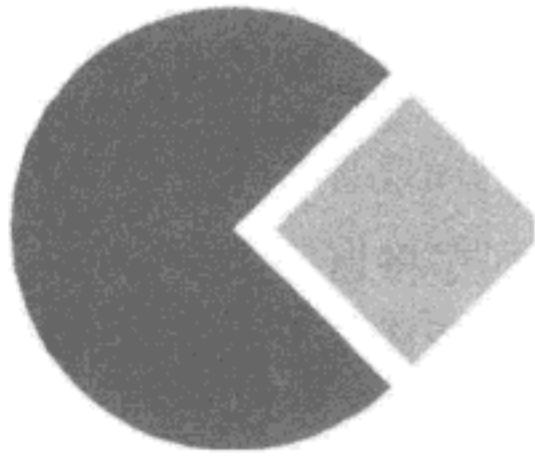
                                        mHandler.sendMessage(10);
                                    }
                                });
                }
                else
                {
                    byte[] buffer = Util
                            .bitmapToByteArray(mCameraBitmap);
                    ImageItem pic = new ImageItem("pic", buffer);
                    // 异步发布微博（包含图像）
                    MicroBlog.mWeibo.uploadStatusAsync(content,
                            pic, new OnStatusListener()
                            {
                                @Override
                                public void onSuccess(Status status)
                                {
                                    mHandler.sendMessage(1);
                                }
                            });
                }
            }
        }
    }
}
```

```
        @Override
        public void onException(Exception e)
        {
            mHandler.sendEmptyMessage(10);
        }
    });

}
catch (Exception e)
{
    mHandler.sendEmptyMessage(10);
}
break;
default:
    break;
}
}
```

## 27.10 小结

本章利用新浪微博 SDK 实现了 Android 版的微博客户端。这个系统拥有微博客户端的基本功能，包括浏览微博信息、发布微博、评论微博和转发微博，还包括一些辅助功能，如选择相册中的照片、拍照、选择表情等。由于本系统代码过多，本章只给出了重要功能的核心代码，完整的实现请读者参阅随书光盘中的源代码。



## 第 28 章 Android 综合案例六—— 笑脸连连看（游戏）

本章将实现一个连连看游戏。连连看是一款非常经典的游戏，通过显示  $n \times m$  个图像，然后按照游戏规则点击两个相同的图像，这两个图像就会消失。直到所有的图像都消失后，游戏结束。如果有时间限制，在指定的时间内未使所有的图像消失，游戏就失败。本例的工程目录是 src\demo\smile\_llk。

### 28.1 游戏玩法

运行游戏后，单击“开始”按钮，会随机生成 42 ( $7 \times 6$ ) 个笑脸图像，如图 28.1 所示。这 42 个图像分为 21 对，随机排列，每一对是相同的两个图像。连连看选相同图像的规则是两个图像之间的连线不能多于 2 个拐角。按此规则，图 28.2 中的两个笑脸（用线段连接的两个笑脸）可以认为是相同的，先后单击两个笑脸后，这两个图像都会消失，图 28.3 中的两个笑脸（用线段连接的两个笑脸）也是两个相同的图像。



▲ 图 28.1 笑脸连连看主界面



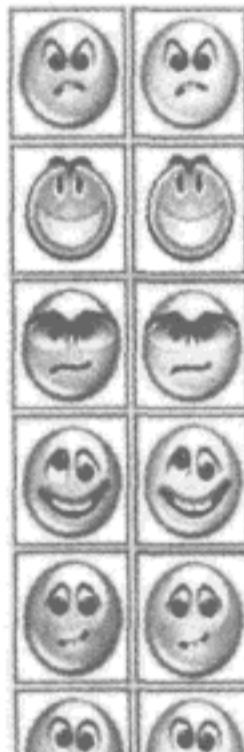
▲ 图 28.2 相同图像 1



▲ 图 28.3 相同图像 2

## 28.2 准备图像素材

连连看的素材主要是游戏的主角：笑脸。随机生成笑脸的方法很多，本程序将这些笑脸都放在了一个图像中，每个笑脸有彩色和灰度两种版本。如果选中图像，该图像就会变成灰度。笑脸的部分图像如图 28.4 所示。



▲ 图 28.4 部分笑脸图像

## 28.3 实现主界面

连连看的核心类是 GameView，该类的构造方法初始化了在游戏中要使用的变量，代码如下：

```
public GameView(Context context)
{
    super(context);
    displayMoreInfo = false;
    mousept = new Point(0, 0);
    lastpt = new Point(0, 0);
    paint = new Paint();
    paintLine = new Paint();
    paintLine.setARGB(255, 255, 0, 0);
    paintLine1 = new Paint();
    paintLine1.setARGB(255, 0, 0, 255);
    paintText = new Paint();
    paintText.setARGB(255, 0, 0, 0);
    paintText.setTextSize(13);
    paintText.setFlags(Paint.ANTI_ALIAS_FLAG);
    paintPross1 = new Paint();
    paintPross1.setARGB(255, 210, 210, 210);
    paintPross = new Paint();
    paintPross.setARGB(255, 0, 0, 210);
    clientRectPaint = new Paint();
    clientRectPaint.setARGB(255, 255, 255, 255);
```

```

hintpt1 = new Point(-1, -1);
hintpt2 = new Point(-1, -1);

res = context.getResources();
isInited = false;
setFocusable(true);
}

```

在 onDraw 方法中绘制了游戏的主界面（还没有生成 42 个随机笑脸图像）。

```

protected void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    canvas.drawBitmap(buff, 1, 1, paint);
}

```

初始化界面有一个核心方法 drawText，该方法绘制了按钮文件以及一些状态信息，代码如下：

```

private void drawtext()
{
    // 绘制主界面图像
    buffCanvas.drawBitmap(bitmapButton, btnRect1.left, btnRect1.top, paint);
    if (!isstart)
    {
        // 如果游戏未开始，绘制“开始”按钮的文本
        buffCanvas.drawText("开始", btnRect1.left + 14, btnRect1.top + 14,
                           paintText);
    }
    else
    {
        if (ispause)
        {
            // 如果游戏正处于当前状态，设置按钮的文本为“继续”
            buffCanvas.drawText("继续", btnRect1.left + 14,
                               btnRect1.top + 14, paintText);
            buffCanvas.drawText(pauseStr, clientLeft, clientTop + 20,
                               paintText);
        }
        else
        {
            buffCanvas.drawText("暂停", btnRect1.left + 14,
                               btnRect1.top + 14, paintText);
        }
    }
    buffCanvas.drawRect(tRect, clientRectPaint);
    buffCanvas.drawRect(tRect1, clientRectPaint);
    // 绘制状态信息
    buffCanvas.drawText("级数：" + String.valueOf(flevel), tRect.left,
                       tRect.bottom, paintText);
    buffCanvas.drawText("分数：" + String.valueOf(ffen), tRect1.left,
                       tRect1.bottom, paintText);
}

```

## 28.4 随机生成连连看图像

通过 `drawPoint` 方法可以生成 42 个笑脸连连看图像，代码如下：

```
private void drawPoint()
{
    Rect dstRect;
    Rect src_r = new Rect(0, 0, 0, 0);
    // 横向绘制笑脸连连看图像
    for (int i = 1; i <= fpointlist.fxcount; i++)
    {
        // 纵向绘制笑脸连连看图像
        for (int j = 1; j <= fpointlist.fycount; j++)
        {
            if (fpointlist.getstat(i, j) < 0)
                continue;
            if (fpointlist.getimgidx(i, j) < 0)
                continue;
            dstRect = fpointlist.getrect(i, j);
            getbitmapPointRect(fpointlist.getimgidx(i, j),
                fpointlist.getstat(i, j), src_r);
            // 绘制每一个图像
            buffCanvas.drawBitmap(bitmapPoint, src_r, dstRect, paint);
            if (((hintpt1.x == i) && (hintpt1.y == j))
                || ((hintpt2.x == i) && (hintpt2.y == j)))
            {
                buffCanvas.drawBitmap(bitmapHint, dstRect.left + 10,
                    dstRect.bottom - 40, paint);
            }
        }
    }
}
```

## 28.5 选中两个相同图像后消失

触摸图像时，需要通过触摸点的坐标判断哪个图像被触摸，并根据是否有与当前触摸图像相同的图像先被触摸决定是否通过红色线段连接后同时消失。

```
public boolean onTouchEvent(MotionEvent event)
{
    if (isOnmousedown)
        return super.onTouchEvent(event);

    isOnTimering = true;
    isOnmousedown = true;

    if (event.getAction() == MotionEvent.ACTION_DOWN)
    {
        int ax = (int) event.getX();
        int ay = (int) event.getY();
```

```

    if (btnRectClose.contains(ax, ay))
    {
        android.os.Process.killProcess(android.os.Process.myPid());
    }
    if (btnRect1.contains(ax, ay))
    {
        dobtn_1();
    }
    // 满足两个图像相同的条件，图像消失
    if (clientRect.contains(ax, ay))
    {
        fpointlist.mouseToxy(ax, ay, mousept);
        if (mousept.x >= 1)
        {
            dopoint();
        }
    }
}
isonmousedown = false;
isontimering = false;
return super.onTouchEvent(event);
}

```

## 28.6 用定时器限制游戏时间

游戏通过一个定时器每 300 毫秒检测一次剩余游戏时间，游戏主界面上方会显示剩余游戏时间。定时器的代码如下：

```

public void ontimer()
{
    if (isontimering)
    {
        return;
    }
    if (isonmousedown)
    {
        return;
    }
    isontimering = true;

    if ((isstart) && (!ispause))
    {
        if (timecount > 0)
        {
            // 减少游戏时间
            timecount = timecount - 1;
        }
        else
        {
            beginlevel(flevel);
        }
    }
}

```

```
        }
        if (ontimercount > 0)
        {
            ontimercount = ontimercount - 1;
        }

        if (ontimercount <= 0)
        {
            drawall();
            // 重绘游戏界面
            invalidate();
            ontimercount = 10;
        }
        isontimering = false;
    }
```

## 28.7 小结

本章实现了一个笑脸连连看的程序。在一定时间内，玩家必须找到所有的相同图像，并单击它们使其消失，否则游戏就会 Game Over，并重新开始游戏。通过本章的学习，读者可以了解游戏的基本开发方法。虽然提供的游戏并不大，但认真学习这些游戏程序也会对我们学习游戏编程的基本方法起到很大的帮助。

Android  
SDK  
2.3

# Android 开发权威指南

本书内容全面，不仅详细讲解了Android框架、Android控件、用户界面开发、游戏开发、数据存储和网络开发等，还深入阐述了传感器、语音识别、桌面组件开发、多媒体开发、OpenGL ES、HTML5、Android NDK编程、Android平台测试等高级知识。

本书注重对实际动手能力的指导，在容易产生错误、不易理解的环节都配以了详实的开发情景截图，并将重要的知识点、开发技巧以“多学一招”、“扩展学习”、“技巧点拨”等活泼的形式呈现给读者。

体现创新的6大综合案例，如新浪微博客户端、蓝牙聊天、全键盘输入法、月球登陆游戏、贪吃蛇游戏、笑脸连连看游戏。

## 本书支持社区



51CTO.com  
技术成就梦想

ZDNet China  
至顶网



博客园  
cnblogs.com

封面设计：胡萍丽

分类建议：计算机/程序设计/移动开发  
人民邮电出版社网址：[www.ptpress.com.cn](http://www.ptpress.com.cn)

ISBN 978-7-115-25714-7



9 787115 257147 >

ISBN 978-7-115-25714-7  
定价：79.00 元（附光盘）