

第 1 章 Android 简介与环境搭建



学习任务

本章主要完成以下学习目标：

- (1) 了解 Android 发展历史。
- (2) 了解 Android 架构特点。
- (3) 掌握 JDK 环境的安装和配置。
- (4) 掌握 Android Studio 安装和配置。
- (5) 掌握 Android 模拟器的使用。
- (6) 掌握 Android 应用程序的部署。



本章介绍

本章简单介绍了 Android 的诞生及其发展史和 Android 系统的整体架构。通过本章的学习，需系统且熟练地掌握 Android 开发环境的搭建，了解 Android 项目开发流程，并学会调试 Android 程序。



1.1 了解 Android

1.1.1 Android 简介

Android 系统是 Google 公司于 2007 年 11 月 5 日发布的基于 Linux 内核的移动平台，该平台由操作系统、中间件、用户界面和应用软件组成，是一个真正开放的移动开发平台。Android 作为目前世界上最成功的操作系统之一，实际上已经超过 Windows 操作系统，覆盖了包括手机、平板、电视、汽车、智能手表等在内的多种产品线，拥有超过了 20 亿台活跃设备，是如今全球最大的操作系统。Android 一词最早出现于法国作家维里耶德利尔·亚当在 1986 年发表的科幻小说《未来夏娃》中，作者将外表像人类的机器起名为 Android。

2010 年，Ascender 公司的伊琳娜·布洛克（Irina Blok）设计了 Android 的 logo，如图 1-1 所示。Android 的 logo 是一个全身绿色的机器人，它是 Android 操作系统品牌形象，时而会以纯文本的形式展示。传闻其设计灵感来源于男女厕所门口的图形符号，Android logo 设计理念为体现产品精神、开源角度以及直接面向对象为开发者，达到与 Linux 的企鹅 logo 类似的效果。

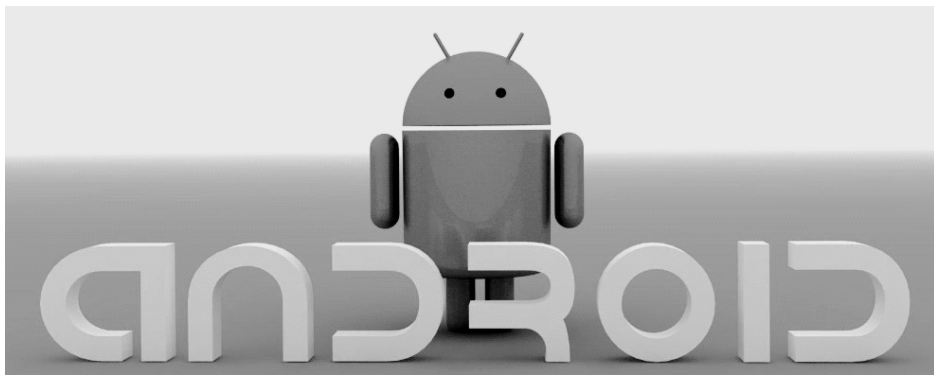


图 1-1 Android logo

1.1.2 Android 系统架构

Android 的系统架构和其他操作系统一样，采用分层的架构，一共分为四个主要功



能层。如图 1-2 所示，Android 系统架构从上到下分别是应用程序层（Applications）、应用框架层（Application Framework）、系统运行库层（Libraries 和 Android Runtime）和 Linux 内核层（Linux Kernel）。

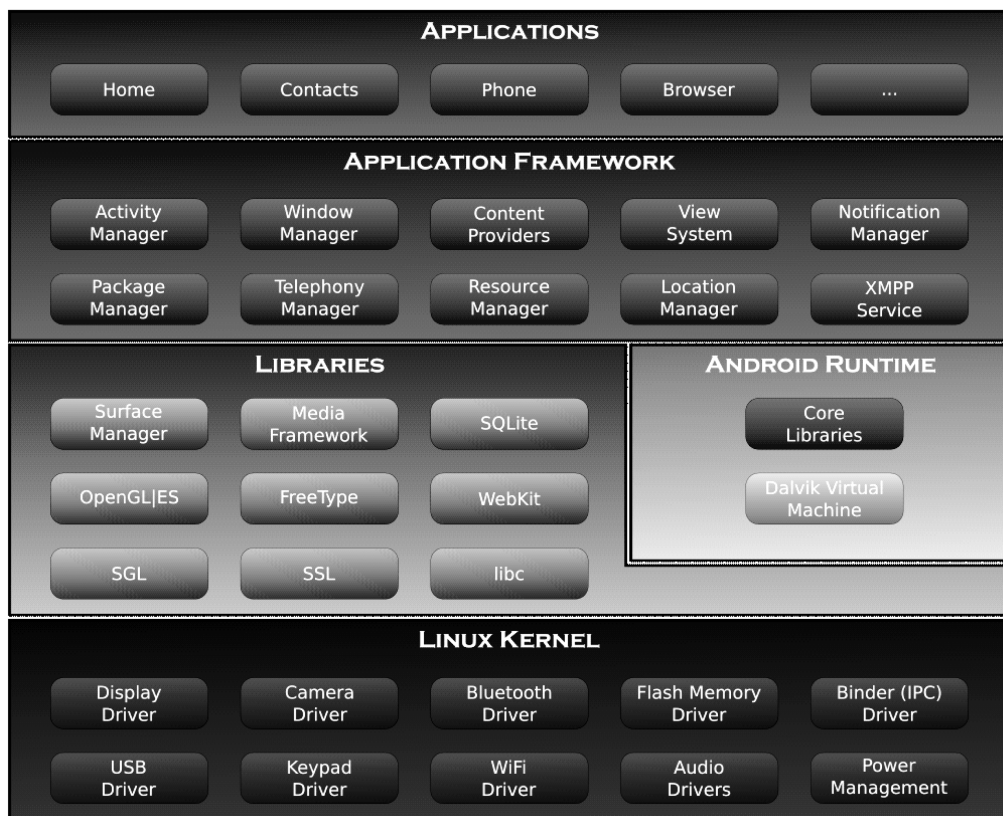


图 1-2 Android 系统架构

1. 应用程序层

Android 的应用程序层包含许多应用程序，其直接与用户交互。例如 Android 智能机上常见的电话、联系人、日历、浏览器等基本功能程序。应用程序一般使用 Java 语言，这也是 Android 开发者需要重点掌握的知识。

2. 应用框架层

应用框架层为 Android 开发人员提供了开发应用程序所需要的 API，Framework 可简单理解为一些 API 的库房，开发者将一些已实现的基本功能，通过接口提供给上层调用，并可以重复调用。应用框架层包括活动管理器（Activity Manager）、窗口管理器（Window Manager）、内容提供者（Content Providers）、视图系统（View System）、通知



管理器 (Notification Manager)、包管理器 (Package Manager)、电话管理器 (Telephony Manager)、资源管理器 (Resource Manager)、位置管理器 (Location Manager)、XMPP 服务等多种服务, 即多个功能模块的组合。Framework 层是由 java 语言编写的。

3. 系统运行库层

系统运行库层分为系统库和 Android 运行时环境两部分。系统库包括系统 C 库、媒体库、Surface Manage、LibWebCore、SGL、3D libraries、FreeType、SQLite 等。Android 运行时的环境由核心库和 Dalvik 虚拟机组成, 继 Android 5.0, Dalvik 虚拟机被 ART 取代。

4. Linux 内核层

Android 系统基于 Linux 内核构建, Linux 内核为 Android 设备的各种硬件提供底层驱动, 实现系统与各种硬件的通信, 例如显示屏、摄像头、内存、键盘、无线网络、音频设备、电源等组件。在 Linux 内核的基础上添加了部分 Android 专用的驱动。Android 的 Linux 内核层包括安全管理、内存管理、进程管理、网络协议栈、驱动程序模型和电源管理等。

1.1.3 Android 的发展史

2003 年 10 月, Android 之父 Andy Rubin 和其他创始人共同成立了 Android 公司, 希望要设计一款可对所有软件开发者开放的移动平台。2005 年 Android 以 4000 万美元被谷歌公司收购, Andy Rubin 成为 Google 公司工程部副总裁, 继续负责 Android 项目。

2007 年 11 月 5 日, 谷歌公司正式发布 Android 系统的测试版本, 并且成立了开放手持设备联盟 (Open Handset Alliance, OHA), 该组织由 34 家手机制造商、软件开发商、电信运营商以及芯片制造商共同组成。这一联盟将支持谷歌发布的手机操作系统以及应用软件, 将共同开发 Android 系统的开放源代码。11 月 12 日, 软件开发工具包 (SDK) 发布。

2008 年, 在 Google I/O 大会上, 谷歌提出了 Android HAL 架构图; 8 月 18 号, Android 获得了美国联邦通信委员会 (FCC) 的批准; 9 月 23 日, 谷歌正式发布了 Android 1.0 系统, 这也是 Android 系统的第一个商业版本。同年 10 月, 世界上第一款真正意义上的 Android 手机产品 HTC Dream 诞生, HTC Dream 是由美国运营商 T-Mobile USA 定制, 中国台湾 HTC (宏达电) 代工生产。2009 年 2 月 9 日, Android 1.1 的更新发布最



初只是为了 HTC Dream。Android 1.1 在内部被称为 Petit Four，虽然这个名字没有被正式使用。

2009 年 4 月 27 日，Android 1.5 版本正式发布，Android 1.5 命名为 Cupcake（纸杯蛋糕）。这是第一次正式发布基于甜点项目的代号，并且谷歌从此开始将 Android 的版本以甜点的名字命名，该系统与 Android 1.0 相比有了很大的改进。9 月，Android 1.6 版本发布，并且推出了搭载 Android 1.6 正式版的手机 HTC Hero（G3），凭借着出色的外观设计以及全新的 Android 1.6 操作系统，HTC Hero（G3）成为当时全球最受欢迎的手机。Android 1.6 也有一个有趣的甜品名称：Donut（甜甜圈）。

自 Android 系统首次发布以来，已经走过十年时间。十年来，Android 各版本及其发布时间如表 1-1 所示。Android 2.2 版本整体性能有较大幅度的提升，Android 3.0 为平板设计的 Android 分支，Android 4.0 直接适配不同设备，无论手机还是平板均可以使用。Android 4.4 主要是优化性能，并没有特别重大的功能更新，它可以让 512MB RAM 的手机也能畅快运行，由此使得大量低端机型上市。Android 5.0 系统使用一种新的 Material Design 设计风格，此外，谷歌对平台进行了内部的更改，ART 正式取代 Dalvik 以提高应用性能，以及旨在改进和优化电池使用的举措 Project Volta 项目。Android 6.0 带来了指纹识别、USB-C 接口、Android Pay、以及运行时权限等功能。Android 7.0 功能进步明显，包括针对大屏手机的分屏模式、更优的多任务模式、Vulkan 3D 渲染 API 等内核改进。Android 8.0 的聚焦重点是电池续航能力、速度和安全，让用户更好地控制各种应用程序，谷歌正慢慢让安卓系统向竞争对手苹果的 iOS 靠拢，加大了对 APP 在后台操作时的限制。目前 Android 9.0 正式版尚未发布，但是预计 Android 9.0 的新功能将包括谷歌统一推送升级、深度集成 Project Treble 模式、更加封闭、原生支持通话录音等。

表 1-1 Android 历史版本

Android 系统代号	版本号	发布时间	API Level
Petit Four（花式小蛋糕）	1.1	2009.2.9	2
Cupcake（纸杯蛋糕）	1.5	2009.4.27	3
Donut（甜甜圈）	1.6	2009.9.15	4
Eclair（松饼）	2.0 - 2.1	2009.10.26	5 - 7
Froyo（冻酸奶）	2.2 - 2.2.3	2010.5.20	8
Gingerbread（姜饼）	2.3 - 2.3.7	2010.12.6	9 - 10



续表

Android 系统代号	版本号	发布时间	API Level
Honeycomb (蜂巢)	3.0 - 3.2.6	2011. 2. 22	11 - 13
Ice Cream Sandwich (冰激凌三明治)	4.0 - 4.0.4	2011. 10. 18	14 - 15
Jelly Bean (果冻豆)	4.1 - 4.3.1	2012. 6. 9	16 - 18
KitKat (奇巧)	4.4 - 4.4.4	2013. 10. 31	19 - 20
Lollipop (棒棒糖)	5.0 - 5.1.1	2014. 11. 12	21 - 22
Marshmallow (棉花糖)	6.0 - 6.0.1	2015. 10. 5	23
Nougat (牛轧糖)	7.0 - 7.1.2	2016. 8. 22	24 - 25
Oreo (奥利奥)	8.0 - 8.1	2017. 8. 21	26 - 27
Pistachio Ice Cream (开心果冰淇淋)	9	2018. 7 (beta 3)	28

1.1.4 Android 开发特色

1. 四大组件

Android 系统的四大组件分别是活动 (Activity)、服务 (Service)、广播接收者 (Broadcast Receiver) 和内容提供者 (Content Provider)。Activity 负责应用程序和用户的所有交互, Activity 是用户接触的最直观的 Android 组件, 用户看到的所有界面基本均是 Activity。Service 是在后台运行, 没有界面展示, 比较常用的 APP 中, 音乐播放器在后台播放时即使用了 Service。在 Android 中, 广播是一种在应用程序之间广泛运用的传输信息的机制, Broadcast Receiver 允许 APP 接收来自各处的广播消息, 可以监听 Android 设备状态变化, 接收、过滤和响应外部事件。Content Provider 是 Android 提供的第三方应用数据的访问方案, 为应用程序之间共享数据提供了可能, 比如开发者想要读取系统电话簿中的联系人, 就需要内容提供者来实现。

2. 丰富的系统控件

Android 系统提供了丰富的系统控件, 开发者可以使用这些控件轻松地写出漂亮的界面, 为用户提供良好的交互。当然, 在实际开发项目中, 开发者还可以根据自己的需求自定义控件, 写出更精彩的控件。

3. SQLite 数据库

Android 系统内置 SQLite 数据库, SQLite 是一款轻型的且遵守 ACID 的关系型数据



库管理系统。Android 系统具有封装完成的 API 用来操作 SQLite 数据库，较方便使用，它也支持标准的 SQL 语法。

4. 强大的多媒体

Android 系统还提供了丰富的多媒体服务，如音乐、视频、录音、拍照、闹铃等，这一切都可以在程序中通过代码进行控制，使用户的应用变得更加丰富多彩。

5. 传感器

Android 系统内置有多种传感器，如加速度传感器、磁力传感器、方向传感器、陀螺仪传感器、光线感应传感器、压力传感器、温度传感器、接近传感器、重力传感器、线性加速度传感器和旋转矢量传感器等。通过灵活地使用这些传感器，开发者可以开发出多种不同作用的应用。

1.2 搭建开发环境

1.2.1 安装配置 JDK

Android 开发工具有很多，目前比较常用的有 Eclipse、IntelliJ IDEA、Android Studio 等。本书推荐使用 Android Studio，Android Studio 是谷歌官方推出的一个基于 IntelliJ IDEA 的 Android 开发工具。Java 作为 Android 开发的主要编程语言，依赖于 JDK 环境。

JDK (Java Development Kit) 软件开发工具包，是整个 Java 开发的核心，包括了 Java 运行环境 (Java Runtime Environment, JRE)、Java 基础类库和一些 Java 工具。首先，在 Oracle 官网 (<http://www.oracle.com>) 上下载 JDK 安装包，JDK 有多重版本，适配各种操作系统，本书默认使用 Windows 64 位操作系统和 JDK 1.8 版本。准备完成 JDK 安装包之后，下面开始安装 JDK。

(1) 双击 JDK 安装包，弹出如图 1-3 所示的界面。

(2) 设置 JDK 安装路径，如图 1-4 所示。单击“更改”按钮可以修改 JDK 的安装路径，建议直接使用默认安装路径，单击“下一步”按钮，开始安装 JDK。



图 1-3 开始安装 JDK



图 1-4 设置 JDK 安装路径

(3) 安装 JRE，如图 1-5 所示。单击“更改”按钮，可以修改 JRE 安装路径，同样建议使用默认路径。单击“下一步”按钮，安装 JRE。



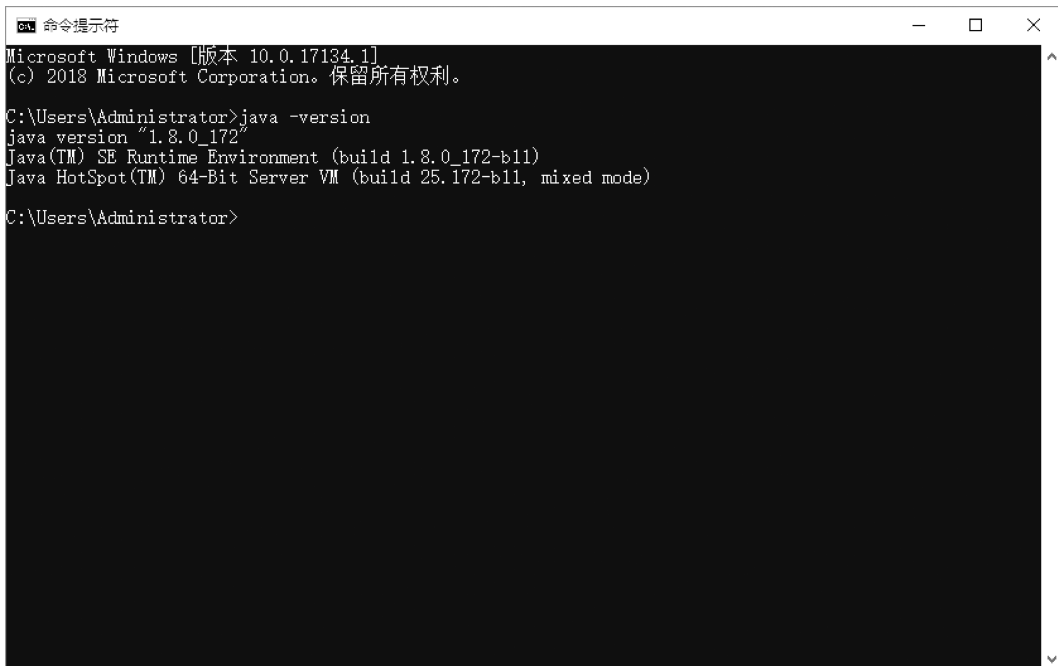
图 1-5 安装 JRE

(4) 安装完成，单击“关闭”按钮，开始配置环境变量。右击桌面计算机图标，并执行“属性”→“高级系统设置”→“环境变量”命令，在系统变量中添加如表 1-2 所示的环境变量。JAVA_ HOME 变量的值是 JDK 的安装路径。

表 1-2 JDK 环境变量

变量名	变量值
JAVA_ HOME	C: \ Program Files \ Java \ jdk1. 8. 0_ 172
Path	;% JAVA_ HOME% \ bin;% JAVA_ HOME% \ jre \ bin;
CLASSPATH	. ;% JAVA_ HOME% \ lib;

(5) JDK 的环境变量已经配置完成，在 Windows 的命令窗口中执行 java -version 命令进行验证，如图 1-6 所示。



```
命令提示符
Microsoft Windows [版本 10.0.17134.1]
(c) 2018 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>java -version
java version "1.8.0_172"
Java(TM) SE Runtime Environment (build 1.8.0_172-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.172-b11, mixed mode)

C:\Users\Administrator>
```

图 1-6 JDK 配置完成

1.2.2 安装配置 Android Studio

2013 年 5 月 16 日，在 Google I/O 大会上，谷歌推出了一款新的 Android 开发环境，即 Android Studio。毫无疑问，Android Studio 是谷歌为 Android 的开发量身定做的开发环境，是谷歌大力支持的一款基于 IntelliJ IDEA 改造的集成开发环境（Integrated Development Environment，IDE）。Android Studio 集成了 Gradle 构建工具，而 Gradle 是一个基于 Ant 和 Maven 的项目自动化构建开源工具，更便于 Android 开发者配置、编译和打包操作。

经过五年的发展，Android Studio 更新了很多个版本，到目前为止，Android Studio 已经正式发布了 3.1.3 版本。本书将以 Android Studio 3.1.2 版本为例详细介绍 Android Studio 在 Windows 平台上的安装与配置。首先，可以在 Android 中国官网（<https://developer.android.google.cn>）上下载 Android Studio 的安装文件，也可以访问 Android Studio 中文社区（<http://www.android-studio.org/>）获取 Android Studio 的安装文件。

如图 1-7 所示是 Android Studio 中文社区官网，展示了谷歌公司针对 Windows、Linux 和 Mac 等不同操作平台发布的不同版本的 Android Studio 安装文件。本书默认使用 Windows 64 位操作系统，可以看到该平台下有两种不同的安装文件，如果是 EXE 可

执行文件，双击“开始安装”向导即可；如果是 ZIP 压缩文件，将其解压即可。下面本书将针对 ZIP 压缩文件进行详细讲解。

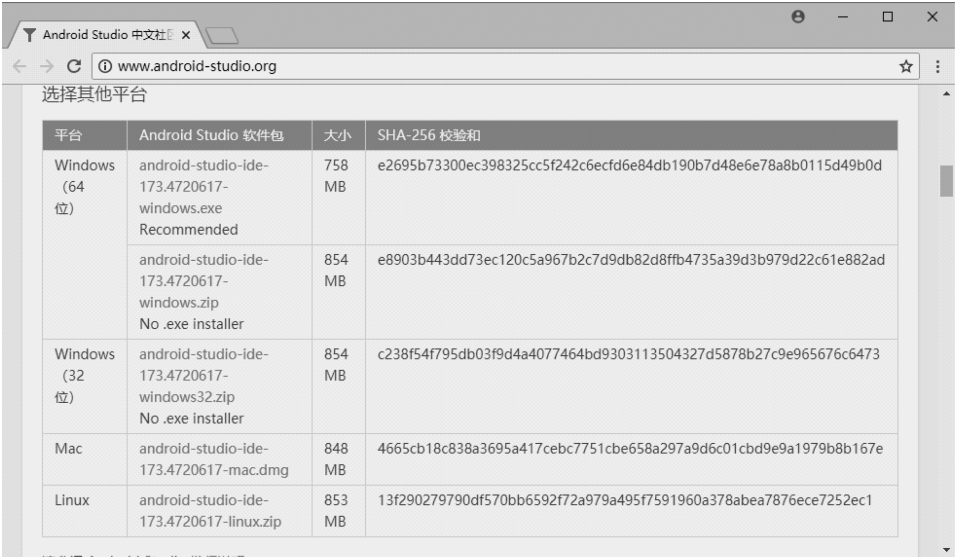


图 1-7 Android Studio 中文社区 AS 下载

SDK（Software Development Kit）软件开发工具包，是软件工程师为特定的操作系统、硬件平台、软件框架和软件包等建立应用程序时的开发工具集合。Android SDK 是 Android 专属的软件开发工具包。由于目前的 Android Studio 安装包中没有包含 Android SDK，因此需要先安装 Android SDK。同理，可以在 Android Studio 中文社区下载 SDK 的 ZIP 压缩文件，见图 1-8 所示，安装步骤如下：

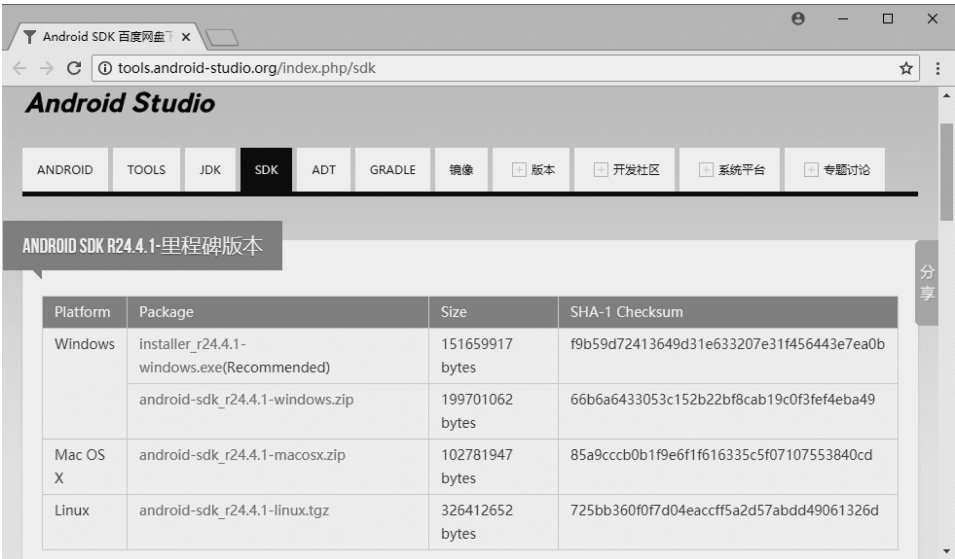


图 1-8 Android Studio 中文社区 SDK 下载



(1) 安装 Android SDK，在 C 盘新建“Android”文件夹，将“android-sdk_r24.4.1-windows.zip”文件解压到 C 盘 Android 目录下。

(2) 安装 Android Studio。解压完成后，进入 Android Studio 的“bin”目录下，找到“studio64.exe”文件（见图 1-9），双击该文件，启动 Android Studio。为了方便使用，可以在桌面为该文件创建快捷方式。

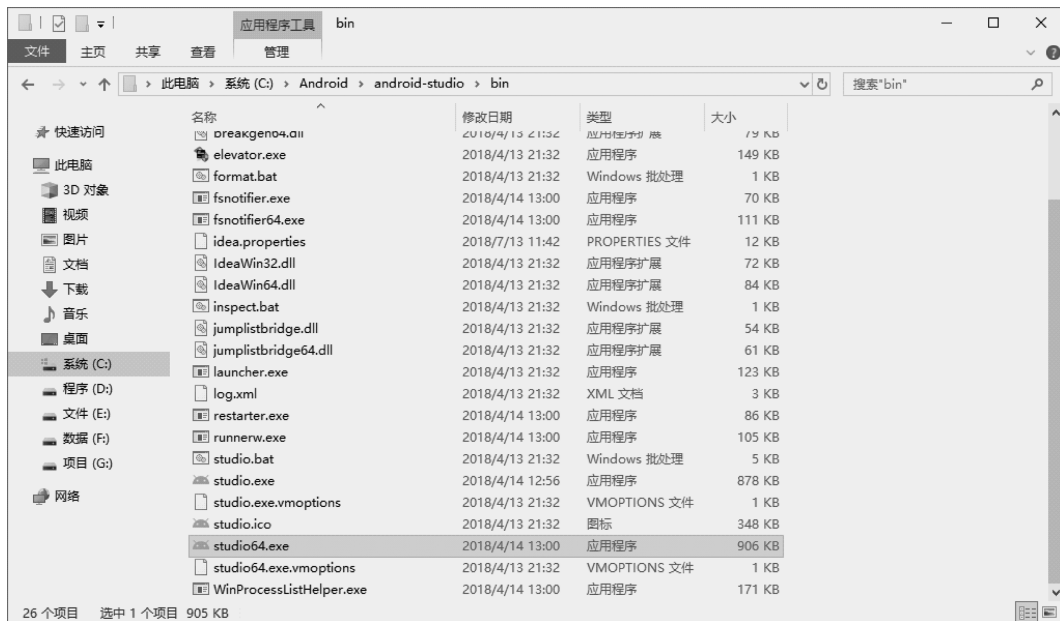


图 1-9 Android Studio 启动文件

(3) 导入 Android Studio 的设置，如图 1-10 所示。首次安装 Android Studio，没有配置文件，选择“Do not import settings”选项，单击“OK”按钮，进入下一步。弹出如图 1-10 所示窗口，单击“Cancel”按钮进入下一步。

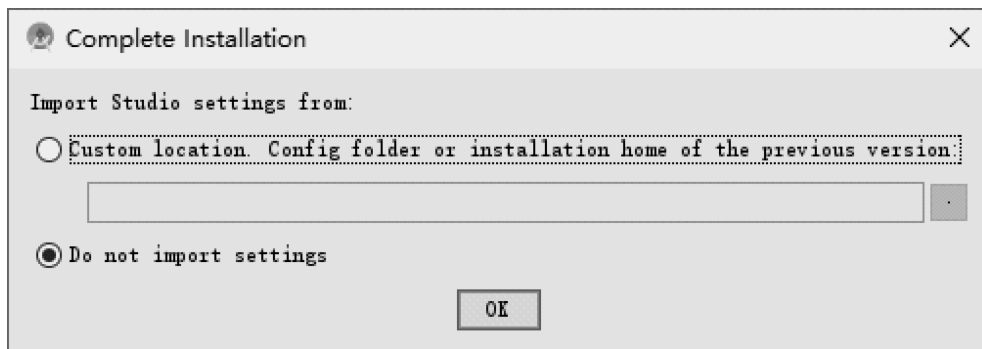


图 1-10 导入 Android Studio 设置

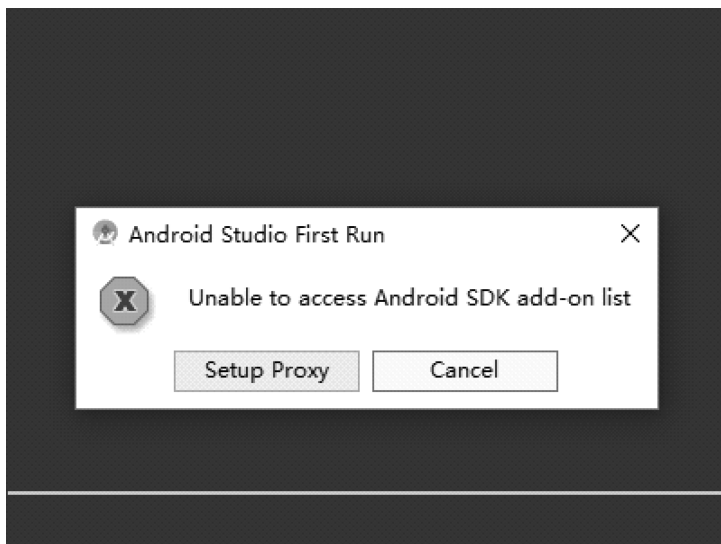


图 1-11 首次运行 Android Studio 弹出框

(4) 单击“Next”按钮，选择安装类型，如图 1-12 所示。选择“Standard”选项，单击“Next”按钮。

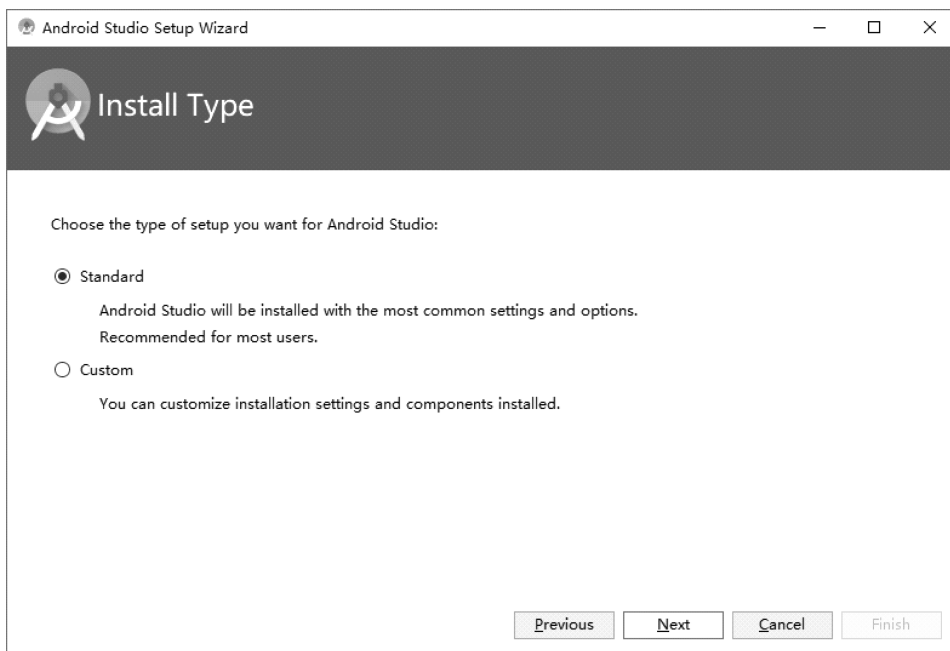


图 1-12 Android Studio 安装类型

(5) 选择 Android Studio 的 UI 主题，如图 1-13 所示。本书选择“Darcula”主题，单击“Next”按钮。

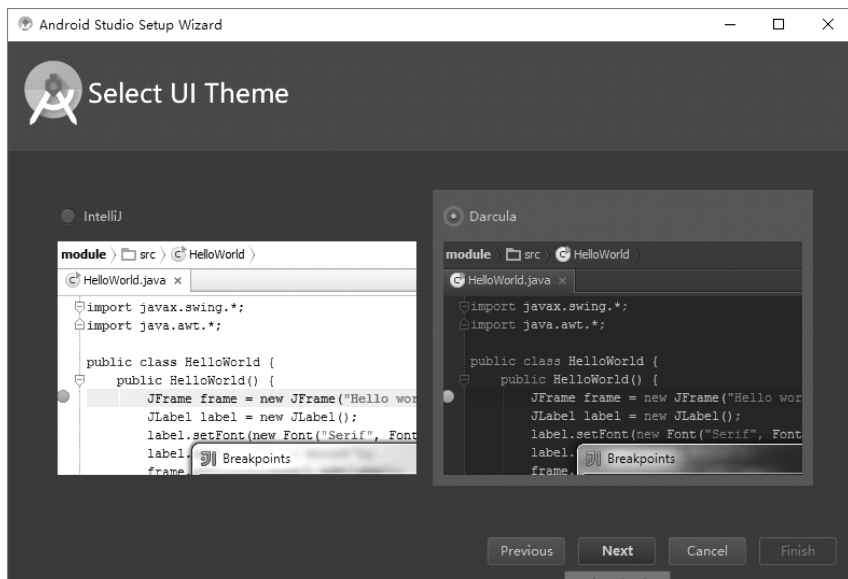


图 1-13 选择 UI 主题

(6) 选择 SDK 安装路径，如图 1-14 所示。在第一步中已安装 SDK，这里直接将“Android SDK Location”设置为之前 SDK 的目录。单击“Next”按钮进入下一步，单击“Finish”按钮，开始下载组件，此过程可能需要一点时间，请保持网络畅通。下载完成后，单击“Finish”按钮，完成 Android Studio 的安装配置，进入 Android Studio 的启动页面，（见图 1-15），现在就可以创建 Android 项目并开始开发。

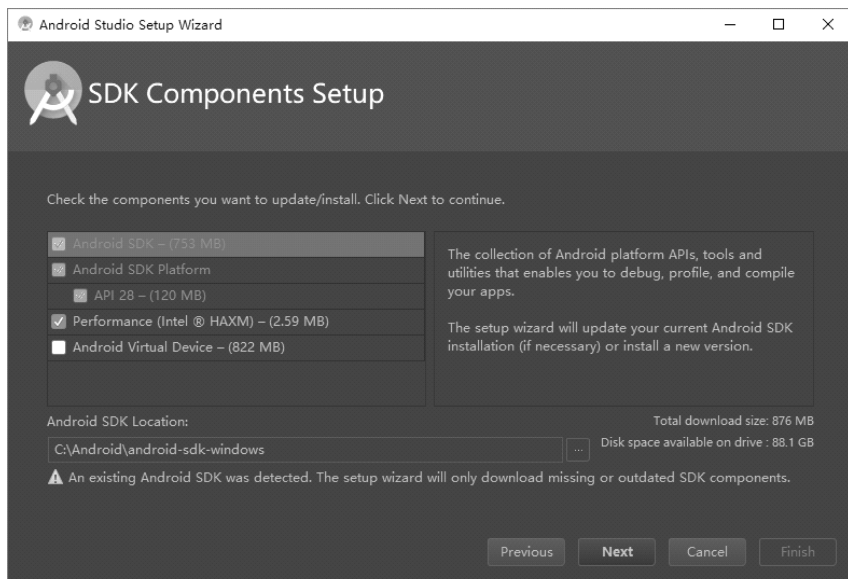


图 1-14 SDK 安装

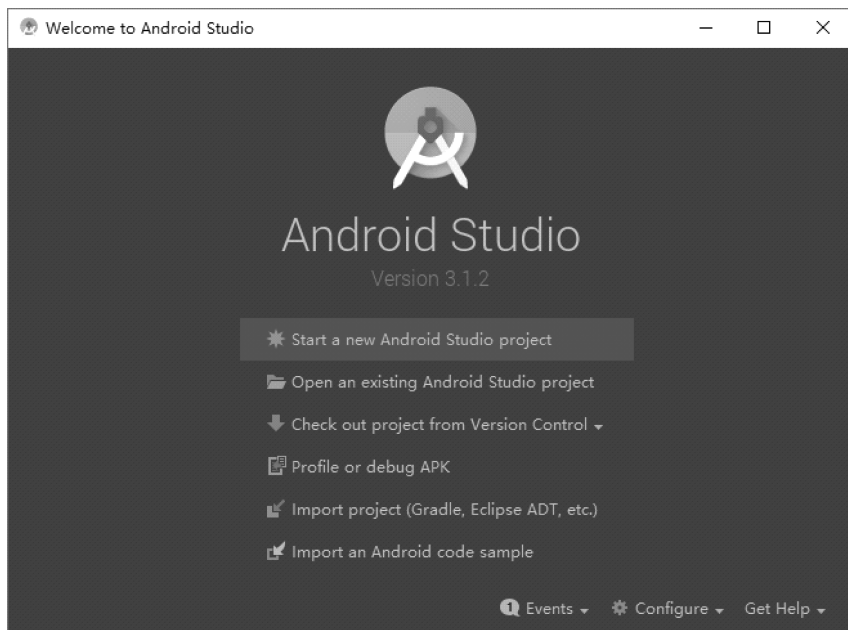


图 1-15 Android Studio 的欢迎页面

1.3 创建第一个项目

1.3.1 创建 HelloWorld 项目

在 1.2 节中，已经完成了 Android 开发的环境搭建。下面，开始创建第一个 Android 项目。

(1) 启动 Android Studio，出现 Android Studio 的欢迎页面，单击“Start a new Android Studio project”按钮，进入如图 1-16 所示界面，配置项目名（Application name）为“HelloWorld”，公司域名（Company domain）一般以“.com”结尾，项目存放的路径（Project location）为“G://androidproject/HelloWorld”，单击“Edit”按钮可以修改 Android 项目的包名（Package name）。编辑完成之后，单击“Next”按钮进入下一个页面。

(2) 选择 Android 设备，如图 1-17 所示。一般选择手机和平板（Phone and Tablet），最低兼容版本目前默认是 Android 4.0.3，即 API 15。还有其他的 Android 设备，例如智能手表（Wear）、智能电视（TV）、车载系统（Android Auto）或者其他嵌入式



设备（Android Things）。单击“Next”按钮进入下一步。

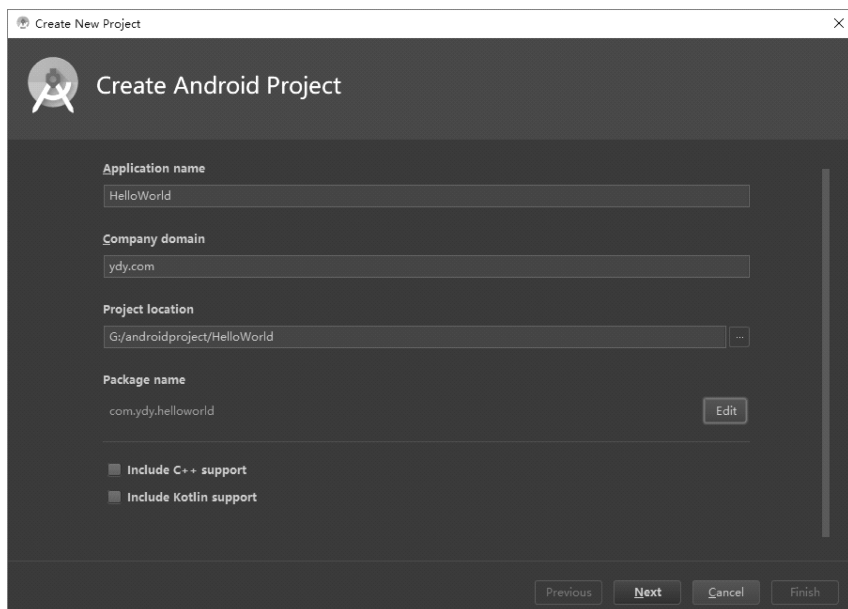


图 1-16 创建 Android 项目

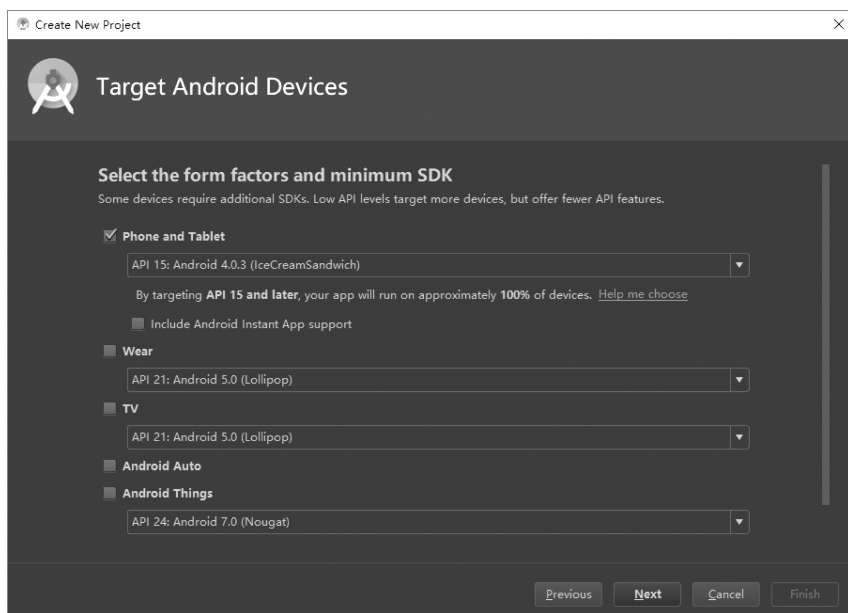


图 1-17 选择 Android 设备

(3) 创建一个活动，如图 1-18 所示。Android Studio 提供了很多模版，选择“Basic Activity”选项，单击“Next”按钮，进入下一个界面。可以修改 Activity 的名字，默认不修改，直接单击“Next”按钮，Android Studio 将会下载一些组件，请保持网络良好。



下载完成之后，单击“Finish”按钮，Android Studio 就会开始下载 Gradle 构建项目。第一次构建一个项目，需要较长的时间。构建完成之后，第一个 Android 应用程序 HelloWorld 即创建完成。

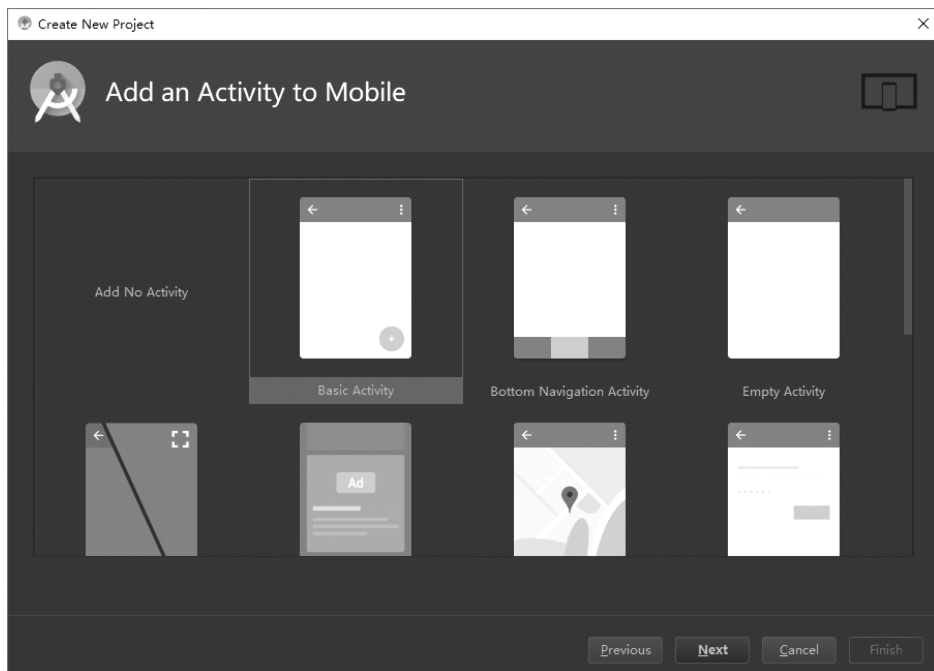


图 1-18 创建 Activity

(4) HelloWorld 项目创建完成之后，进一步了解项目结构。所有的项目创建完成后均是 Android 模式的项目结构，单击“Android”按钮切换到 Project 模式，如图 1-19 所示。

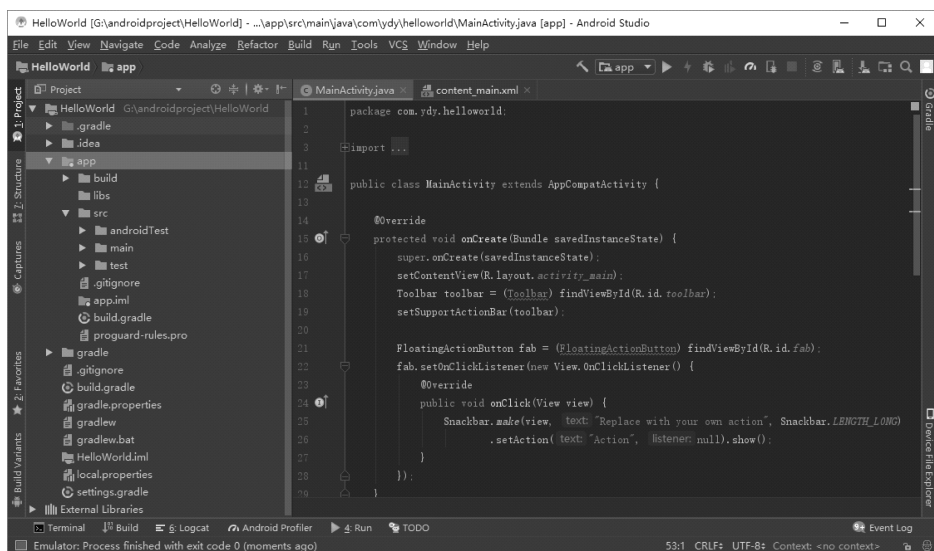


图 1-19 Android Studio 项目



. gradle/. idea：是 Android Studio 自动生成的目录。在版本控制中，这些都是需要被忽略的文件。

gradle：是 Gradle 配置文件，一般不做修改。

app：是整个项目的核心 module 模块，Android 程序中所有的代码、资源文件等全都包含在 app 目录中。

build：项目编译产生的文件。

libs：存放于第三方 jar 包中。

1.3.2 安装部署第一个项目

在上一节中，创建了第一个 Android 项目——HelloWorld 项目。下面将此项目部署到 Android 模拟器上。

(1) 创建 Android 模拟器。单击如图 1-3-4 所示中的 Android Studio 工具栏上的“AVD Manager”按钮，弹出“Android Virtual Device Manager”会话框，单击“Create Virtual Device...”按钮，弹出“Virtual Device Configuration”对话框（见图 1-20），选择一个喜欢的手机模型，单击“Next”按钮进入下一步。下载一个系统镜像，单击“Next”按钮，所有设置均可以使用默认设置，单击“Finish”按钮，回到“Android Virtual Device Manager”会话框（见图 1-21），单击“Launch this AVD in the emulator”按钮，即右

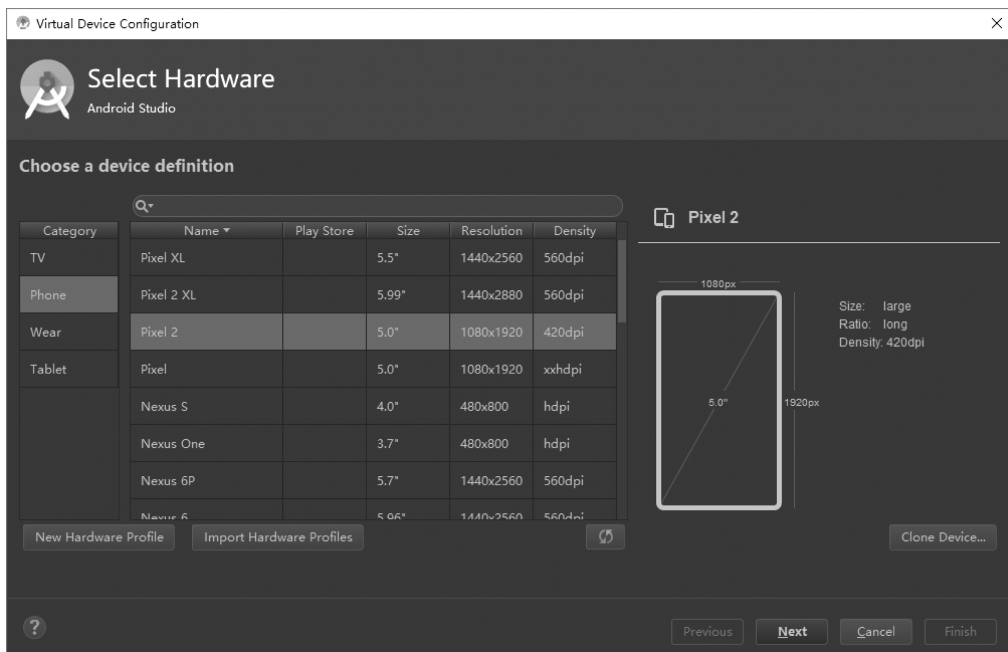


图 1-20 选择手机模型

侧运行图标▶启动模拟器。如果在运行过程中遇到“Intel HAXM is required to run this AVD, VT-x is disabled in BIOS”这样的错误提示信息，需要进入 BIOS 界面，在“configurations”中找到“Intel Virtualization Technology”，（Launch this AVD in the emulator 当光标放到右上角小三角形上时会显示出来）将其设置成“Enable”即可。

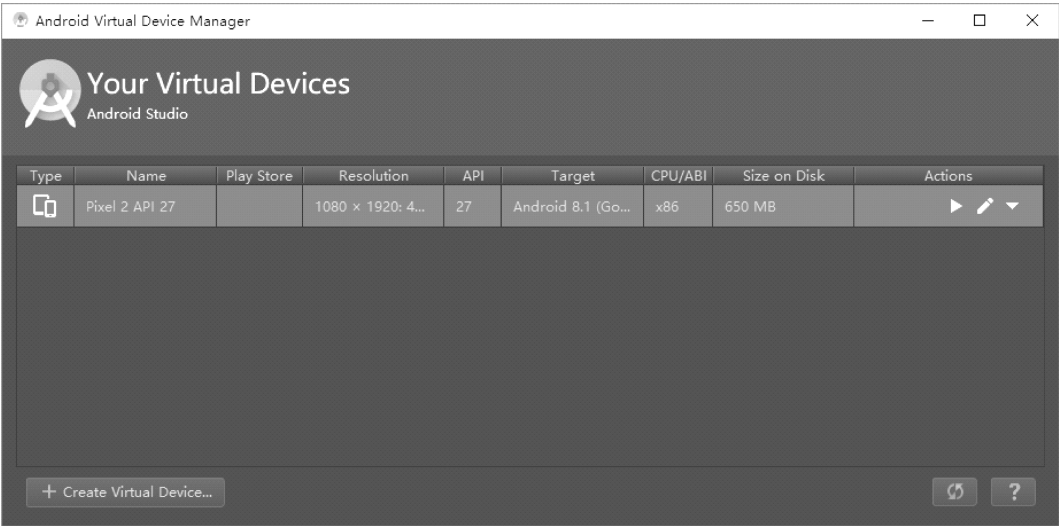


图 1-21 启动模拟器

（2）启动模拟器之后，关闭“Android Virtual Device Manager”对话框，单击 Android Studio 工具栏上的“Run ‘app’”按钮，弹出如图 1-22 所示对话框，选中刚新建的模拟器，单击“OK”按钮，将 HelloWorld 项目部署到模拟器上。部署 Android 项目也需要一些时间，部署完成之后，出现如图 1-23 所示的界面。

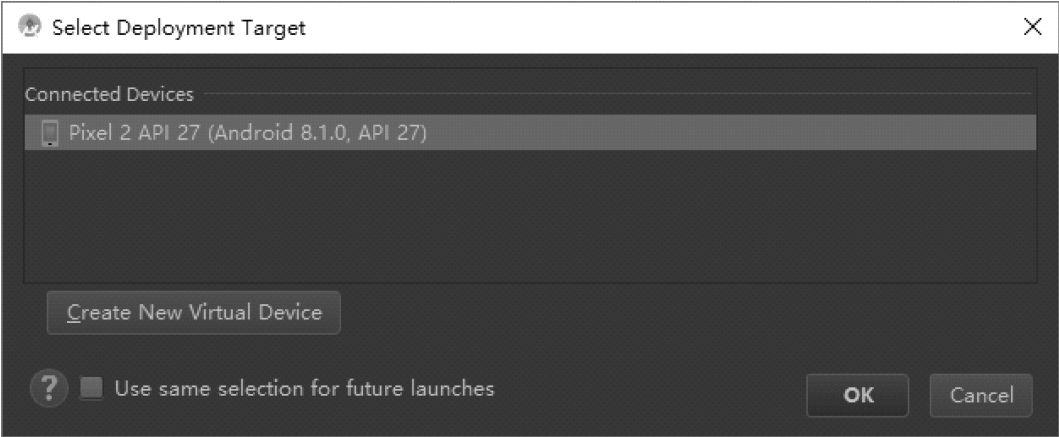


图 1-22 部署 Android 项目

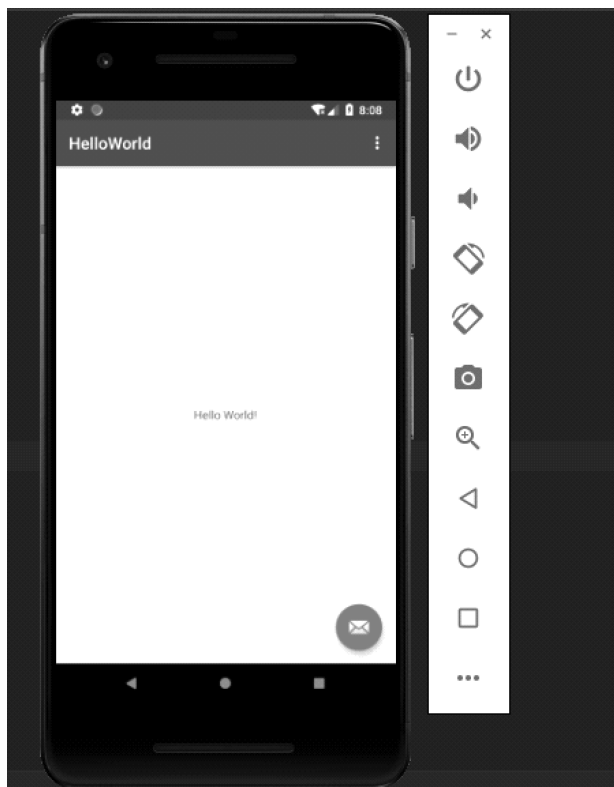


图 1-23 Android 模拟器

1.3.3 日志工具使用

通过前面的学习，相信读者已经能独立完成 Android 开发环境的搭建，创建一个简单的 App 并将其部署到模拟器上。

1. Log 类

Java 是 Android 开发的基础，很多人已经习惯使用 `System.out.printf()` 方法来打印日志，但是在 Android 开发中并不推荐使用此方法。Android 中有专门的日志工具类 `Log` (`android.util.Log`)。

`Log` 类主要提供了 5 个方法输出日志：

- `Log.v()`：输出 Verbose 级别的日志，是 Android 系统中最低级别的日志。
- `Log.d()`：输出 Debug 级别的日志，打印一些调试信息。
- `Log.i()`：输出 Info 级别的日志，打印一些比较重要的信息。



- `Log.w()`: 输出 Warn 级别的日志, 打印警告信息。
- `Log.e()`: 输出 Error 级别的日志, 打印错误信息。

在前面创建的 HelloWorld 项目的 “MainActivity.java” 文件中, 使用 `onCreate` 方法添加如下代码:

```
Log.i("tag", "onCreate:");
```

`Log.i()` 方法有两个字符串类型的参数, 第一个参数是 `tag`, 需要自定义, 用来对打印的日志进行过滤。第二个参数是需要打印的具体内容。运行 app, 可以在控制台看到这条打印日志, 如图 1-24 所示。

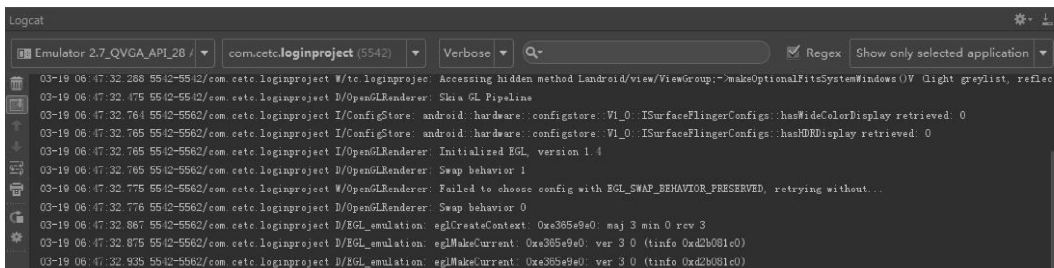


图 1-24 Logcat 输出日志

2. Logcat

Logcat 是 Android Studio 的日志查看工具, 是 Android Studio 的重要组件。在上一节的案例中, 可以了解到 Logcat 中输出的日志信息包含日志打印时间、应用程序进程号、应用程序的包名、日志级别、tag 名 (用来给日志做过滤的标志) 和日志内容。

Android Studio 的 Logcat 功能非常强大, 整个 Logcat 有 5 个过滤设置。第一个设置是 Android 设备选择, 当开发者同时使用多个 Android 设备调试程序时, 通过此设置可以选择当前调试的设备, 过滤掉其他 Android 设备的日志信息。第二个是应用程序包名过滤, 通常一个 Android 设备中都有多个应用程序同时运行, 这个时候可以设置包名过滤器, 让开发者在 Logcat 中只看到当前调试的应用程序的日志。第三个是日志级别的选择, 较常用的是 Verbose、Debug、Info、Warn、Error 这 5 个级别的日志, 与 Android 的 Log 类输出日志级别一一对应, 因此开发者可以利用此特性更方便调试程序。第四个是 tag 过滤, 就是开发者在 Log 函数中传递的 tag 参数, 这是为了方便开发者能精准定位, 在输入框中输入自定义的 tag, Logcat 即能过滤掉其他日志, 只显示匹配当前 tag 的日志, 使用非常方便。第五个是 Filter, 单击右边下拉框, 出现如图 1-25 所示的选择框, Android Studio 的 Logcat 的 Filter 有 4 个选项: “Show only selected application” 表示



只显示当前选择的应用程序的日志；“Firebase”是谷歌提供方的一个分析工具；“No Filters”表示没有过滤器，Logcat 会打印出所有的日志信息；“Edit Filter Configuration”表示开发者开源自定义过滤器。

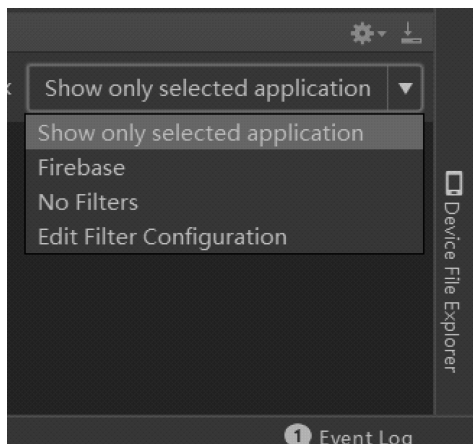


图 1-25 Logcat Filter 选择

接来自定义一个过滤器。单击图 1-26 中的“Edit Filter Configuration”，弹出如图 1-26 所示的对话框。可以看出自定义 Filter 基本包含了整个 Logcat 的功能，在这里只编辑“Log Tag”和“Log Level”，即日志 tag 和日志级别，过滤器的名字（Filter Name）直接命名为“过滤规则”，这样比较方便使用。单击“OK”按钮，自定义过滤器就创建完成。

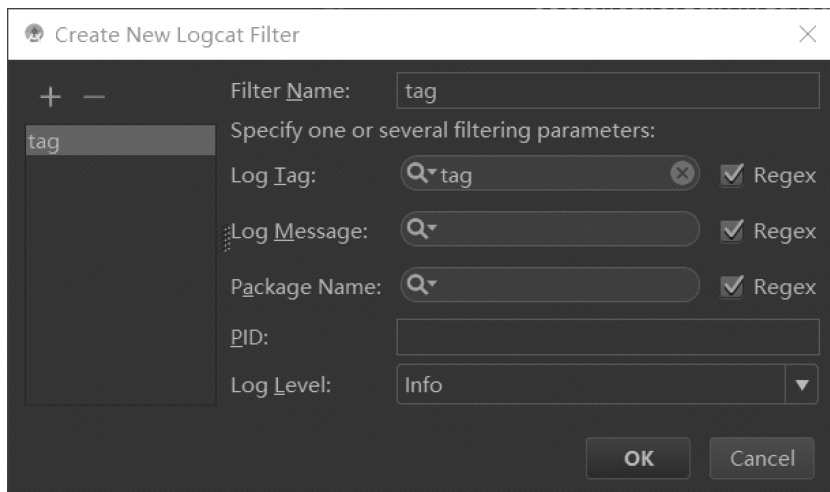


图 1-26 自定义过滤器

Logcat 输出的 info 级别日志为白色，如果没有 tag，很难在 Logcat 中找到这一条日志，由于除了 Error 级别的日志为红色，其他日志均为白色，这对开发者而言处理不方

便。为了方便使用，可以将不同日志级别改成不同的颜色。首先进入 Android Logcat 执行“File”→“Settings”→“Editor”→“Color Scheme”→“Android Logcat”命令，如图 1-27 所示。选择日志级别，再选择“Foreground”选项，单击其后的颜色方框，弹出“Select Color”会话框，选择或者编辑自己喜欢的颜色，单击“Choose”按钮，该级别的日志颜色就设置完成。将所有日志颜色设置完成之后，单击“OK”按钮。

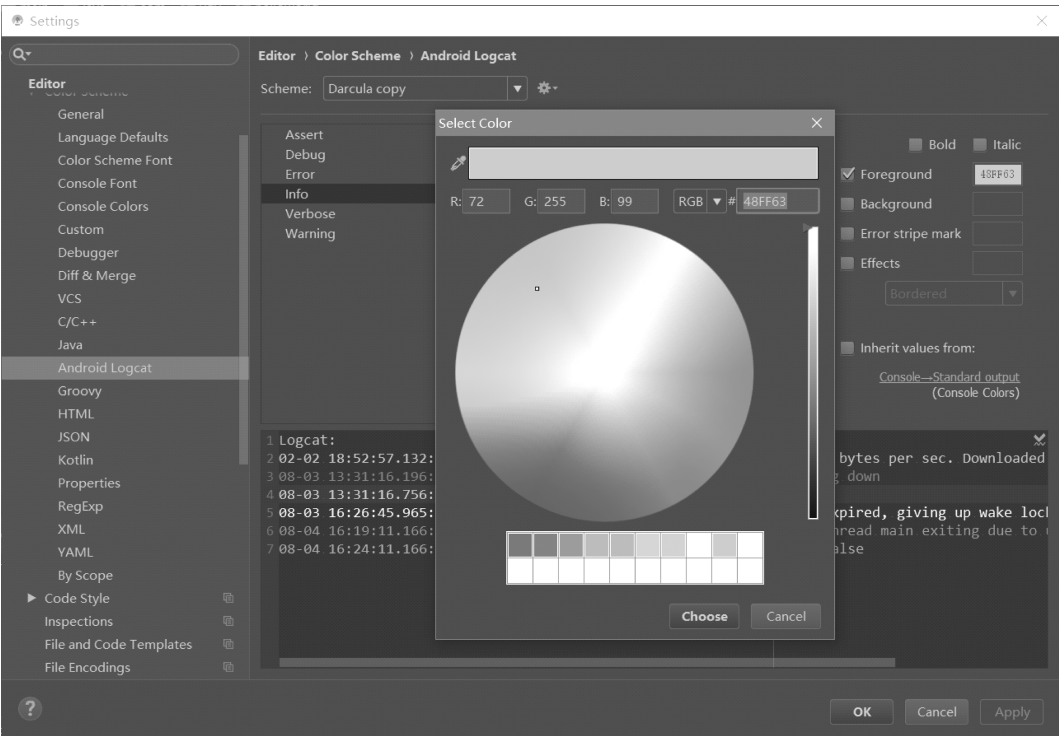


图 1-27 Logcat 颜色设置

Logcat 的颜色已经设置完成，下面进行验证设置。运行 HelloWorld 程序，打印日志（见图 1-28），info 级别的日志颜色变为绿色，过滤器是之前自定义的“tag”过滤器。

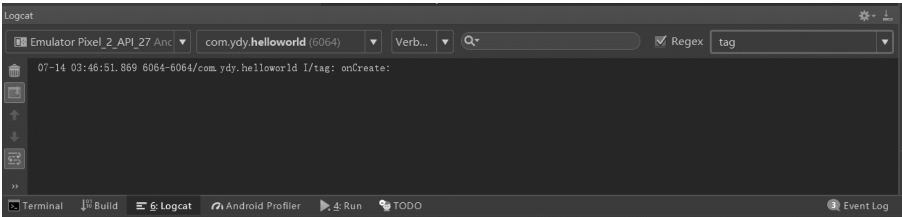


图 1-28 Logcat 输出日志



练习题

- (1) Android 下一个发行版本的英文系统代号以 R 开头对吗?
- (2) 不支持 VT-x 的 CPU 还能正常使用 Android 模拟器吗?
- (3) Android 应用程序最终打包为_____运行在智能设备中。
- (4) Android studio 是借助于_____实现项目构建与管理的。
- (5) Android 设备的调试与开发是借助与 SDK 中的_____工具进行操作的。

第2章 探究 Activity



学习任务

本章主要完成以下学习目标：

- (1) 掌握 Activity 的创建和加载。
- (2) 掌握 Intent 的使用方法。
- (3) 掌握 Activity 的生命周期。
- (4) 掌握 Activity 的启动模式。



本章介绍

前一章已学习了 Android 开发的环境搭建，本章开始实践 Android 的简单开发。Activity 是 Android 开发最常用的组件，通过本章的学习，读者需要掌握 Activity 的基本用法、Activity 之间的相互启动、Activity 之间数据的传递、Activity 的生命周期，这是 Android 开发的基础。还要学会根据不同的应用场景，使用 Activity 的不同启动模式。



2.1 Activity 的基本用法

2.1.1 了解 Activity

Activity 表示具有用户界面的单一屏幕。Activity 是 Android 的四大组件之一，用户可以通过 Activity 提供的屏幕进行交互，进而使用拨打电话、发送信息、查看联系人或拍摄照片等功能。每个 Activity 均会有一个用于绘制用户界面的窗口，通常会占据整个 Android 设备的屏幕，其大小和位置可以改变，可以小于屏幕，或者浮动于其他应用窗口之上。

一个应用程序一般会包含多个 Activity，指定其中一个 Activity 为主，Activity 作为整个程序的入口，当用户首次启动应用程序时，这个主 Activity 就会出现在屏幕上。所有 Activity 之间相互启动，完成与用户的不同交互功能，组成整个应用程序。在上一章中创建的 HelloWorld 程序中，只包含一个 Activity，即 MainActivity，它作为主 Activity，是整个 HelloWorld 程序的入口，将 HelloWorld 程序部署到 Android 模拟器上时，用户即可看到 hello world 界面，即为一个 Activity。

2.1.2 创建 Activity

在 HelloWorld 程序中创建 Activity 项目的时候，若选择“Basic Activity”，Android Studio 则会自动创建一个 MainActivity，同时它也是程序的入口，将程序部署到模拟器上，即可看到 hello world 界面。下面手动创建一个名为“FirstActivity”的 Activity。

将 HelloWorld 项目切换至 Project 模式，展开 app 目录，如图 2-1 所示。HelloWorld 项目的包名为“com.ydy.helloworld”，在 helloworld 目录下为“MainActivity”文件，“helloworld”目录是所有 Java 文件的根目录。为了方便管理，创建一个“activity”包来存放所有的 Activity 文件。右击执行“helloworld”→“New”→“Package”命令，出现 New Package 会话框，在输入框中填写“activity”，单击“OK”按钮，“activity”包就创建在“helloworld”目录内。右击执行“activity”→“New”→“Java Class”命令，弹出 Create New Class 会话框（见图 2-1），在“Name”中填入“FirstActivity”，所有的 Activity 均是继承自 Android 的 Activity 类，因此可以在“Superclass”中填入“Activi-



ty”，其他选项均保持默认，单击“OK”按钮完成设置。

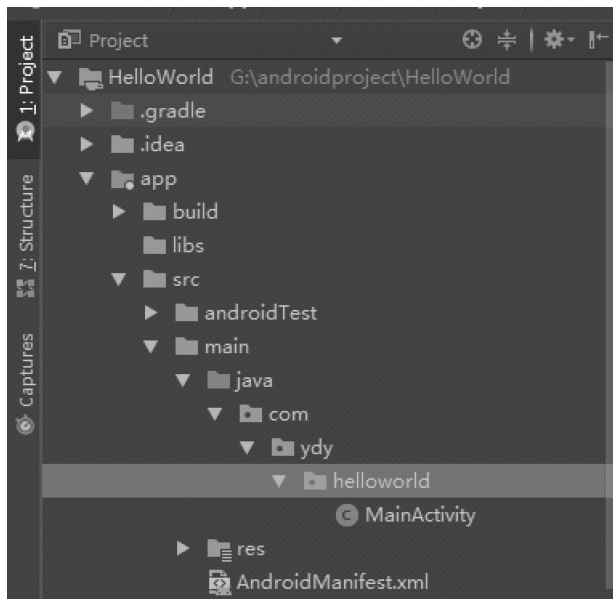


图 2-1 HelloWorld 目录结构

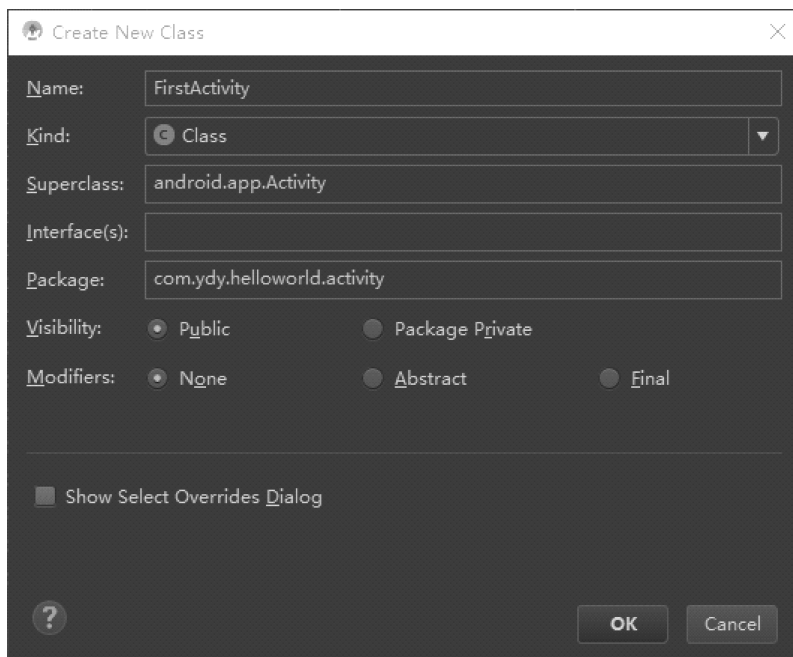


图 2-2 创建 FirstActivity

打开“FirstActivity”文件，在 FirstActivity 类中右击并执行“Generate”命令，在 Generate 弹出框中选择“Override Methods”选项，弹出 Select Methods to Override/Imple-



ment 对话框，在“android.app.Activity”选项目录下选择“onCreate (savedInstanceState: Bundle): void”，单击“OK”按钮（见图 2-3），FirstActivity 文件创建完成。

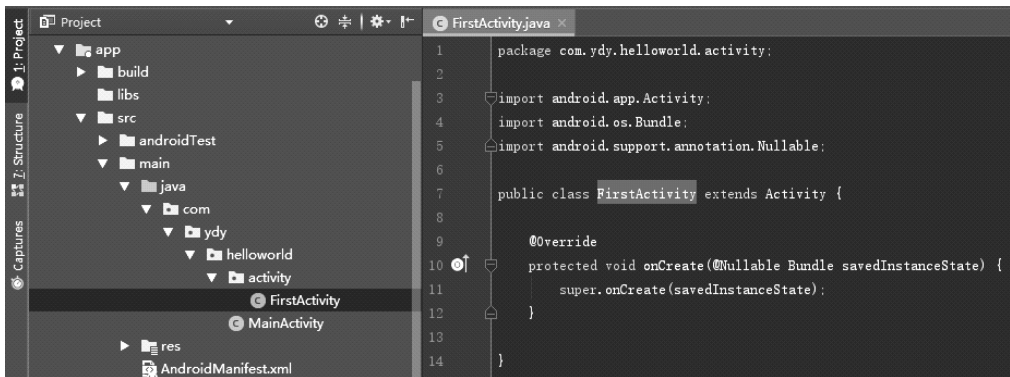


图 2-3 FirstActivity 文件

2.1.3 创建和加载布局

Android 应用的用户界面包含用户可查看并与之交互的所有内容，Android 采用 XML 语言编写 Activity 的布局文件。Android 用户界面是由层级式视图提供，层级式视图均是由 Android 的 View 类衍生而来，每个视图都控制 Activity 窗口内的特定矩形空间，可对用户交互做出响应。下面进行创建 FirstActivity 的布局。

展开如图 2-1 所示的 res 目录，右击执行“layout”→“New”→“Layout Resource File”命令，弹出 New Layout Resource File 对话框，（见图 2-4），在“File name”中填写“activity_first”，在“Root element”中填写“LinearLayout”，单击“OK”按钮。

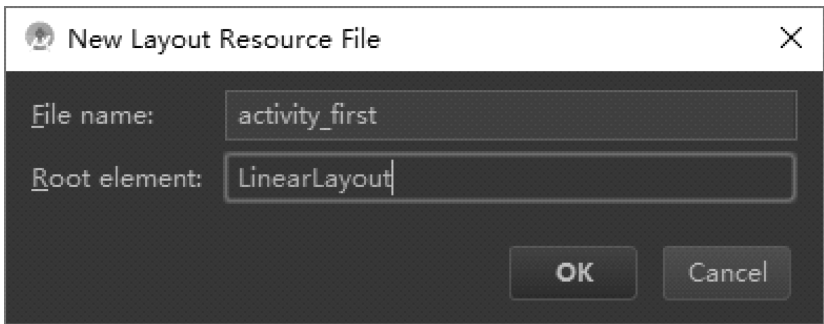


图 2-4 创建 FirstActivity 的布局文件

创建完成之后，出现如图 2-5 所示的布局编辑器，这是 Android Studio 提供的可视化布局编辑器。在编辑器的左下方有两个切换卡：“Design”和“Text”。“Design”即是当前的布局编辑器，可以选择左边的控件，拖动到中间手机屏幕上编辑布局。



“Text” 界面即是 XML 编辑器。

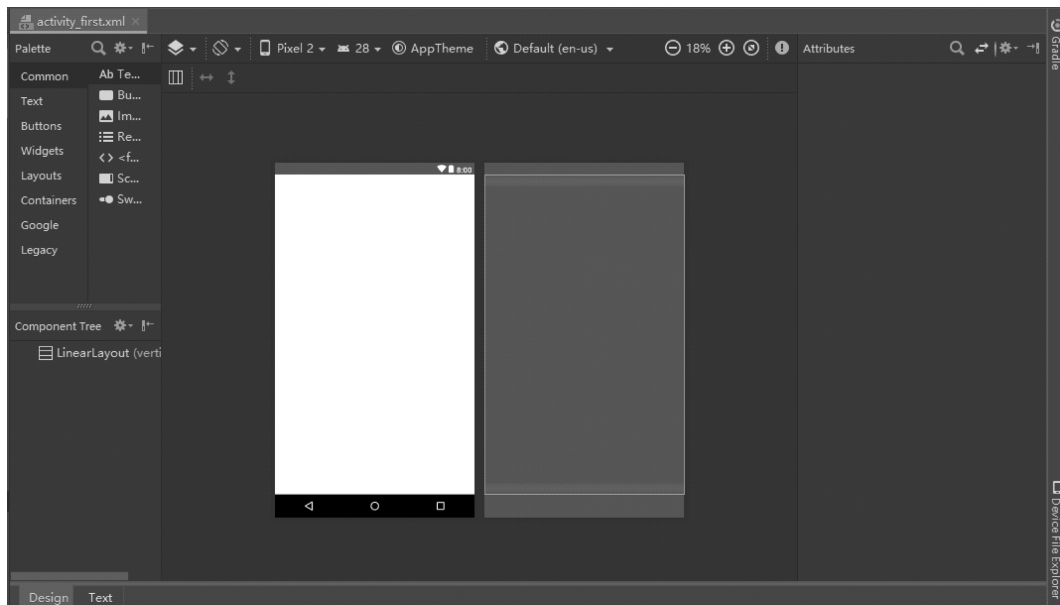


图 2-5 布局编辑器

切换到“Text”界面，可以看到如下代码：

```
<? xml version = " 1.0" encoding = " utf-8" ? >
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: orientation = " vertical"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent" >
</LinearLayout >
```

若在创建该布局文件的时候选择“LinearLayout”作为根节点，则所有的控件均要放在根节点下。现在在该布局文件中添加一个 Button 控件，代码如下：

```
< Button
    android: id = " @ + id/activity_ first_ bt_ first"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: text = " First" / >
```

Button 控件中添加了 4 个属性：id 是当前控件的唯一标识符，layout_ width 是指定控件的宽度，layout_ height 是指定控件的高度，text 是控件展示的文字内容。layout_



width 和 layout_height 是所有控件都必须指定的属性。单击编辑器右边的“preview”按钮，预览当前布局，如图 2-6 所示。

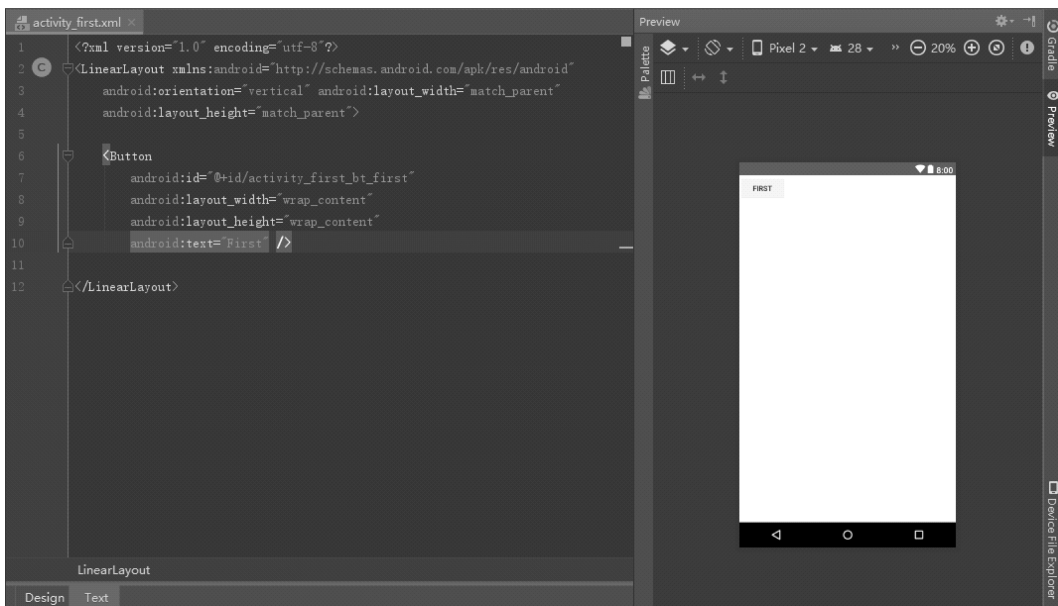


图 2-6 activity_first.xml 编辑预览

注意：

本书使用的是 Android Studio 3.1.2 版本，新建项目的布局文件无法进行预览，这可能是 Android Studio 的一个 bug。如果用户也遇到此问题，请在“res”目录下，展开“values”目录，打开“styles.xml”文件，将主题样式加上 Base 前缀，即在“Theme.AppCompat.Light.DarkActionBar”前加上“Base.”，代码如下：

```
<!-- Base application theme. -->
<style name = " AppTheme"
    parent = " Base. Theme.AppCompat.Light.DarkActionBar" >

</style>
```

即可预览。

布局文件编写完成之后，回到 Activity 中加载该布局文件。在 FirstActivity 类的 onCreate（）方法中调用 setContentView（）方法加载布局。代码如下：

```
public class FirstActivity extends Activity {
    @Override
    protected void onCreate（@Nullable Bundle savedInstanceState）{
```



```
super.onCreate ( savedInstanceState );  
    setContentView ( R.layout.activity_ first );  
}  
}
```

Android 项目中的所有资源均会在 R 文件中生成一个相应的资源 id，在此处将 “activity_ first.xml” 文件的资源 id 传给 setContentView ()，并将布局加载到 FirstActivity 中。

2.1.4 在清单文件中注册

AndroidManifest.xml 是 Android 的清单文件，Android 的四大组件都需要在 AndroidManifest.xml 中注册声明，这样 Android 系统才能访问。每个 Android 应用均包含一个 AndroidManifest.xml 文件，为应用的 Java 软件包命名，描述应用的各个组件，确定托管应用组件的进程，声明应用必须具备的权限，列出 Instrumentation 类，声明应用所需的最低 Android API 级别，列出应用必须链接到的库。下面将 FirstActivity 注册到 Android 的清单文件中。

打开 “app/src/main/AndroidManifest.xml” 路径下文件，如图 2-7 所示。<manifest> 和 <application> 是必需的元素，其他大部分元素可以出现多次或者从不出现。Ac-

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    package="com.ydy.helloworld">  
  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="@string/app_name"  
        android:roundIcon="@mipmap/ic_launcher_round"  
        android:supportsRtl="true"  
        android:theme="@style/AppTheme">  
        <activity android:name=".MainActivity">  
            <intent-filter>  
                <action android:name="android.intent.action.MAIN" />  
  
                <category android:name="android.intent.category.LAUNCHER" />  
            </intent-filter>  
        </activity>  
    </application>  
</manifest>
```

图 2-7 AndroidManifest.xml



tivity 声明在 < application > 节点下。< activity > 元素声明 MainActivity，这是已创建的 Activity，< intent-filter > 元素及其包含的内容声明应用程序的入口，因此在模拟器上打开 HelloWorld 程序就出现了 hello world 界面。

现在将 FirstActivity 在清单文件中声明，并指定 FirstActivity 为 HelloWorld 程序的入口。添加如下代码：

```
<manifest ... >
    <application ... >
        <activity android:name=".MainActivity" ... >
        </activity>
        <activity android:name=".activity.FirstActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

添加完成，运行 HelloWorld 程序。部署完成之后（见图 2-8），HelloWorld 程序的入口已经变为 FirstActivity。

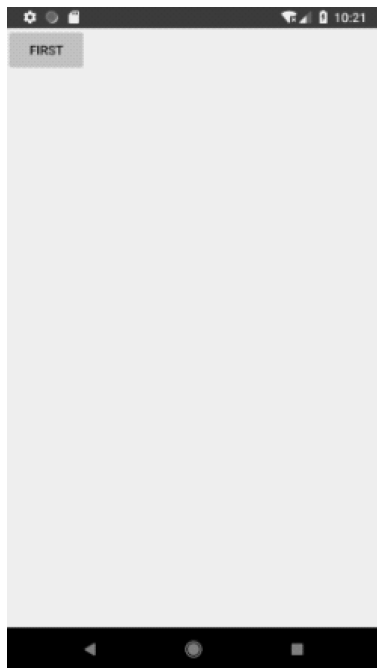


图 2-8 FirstActivity 界面



2.1.5 销毁 Activity

在上一节中，已经创建完成 FirstActivity，并且部署到模拟器上，现在需要销毁 FirstActivity。

最简单的方法为按“Back”键，FirstActivity 即被销毁。也可以使用代码的方式销毁 FirstActivity，代码如下：

```
protected void onCreate (@Nullable Bundle savedInstanceState) {  
    super.onCreate ( savedInstanceState );  
    setContentView ( R.layout.activity_ first );  
  
    findViewById ( R.id.activity _ first _ bt _ first ) .setOnClickListener ( new  
View.OnClickListener () {  
        @Override  
        public void onClick ( View view ) {  
            finish ();  
        }  
    } );  
}
```

给控件 Button 添加一个监听单击事件，单击“Button”按钮时，使用 Activity 类提供的 finish () 方法销毁 FirstActivity。

重新运行 HelloWorld 程序，将程序部署到模拟器上，部署完成后，屏幕上将出现 FirstActivity 界面，单击“FIRST”按钮，FirstActivity 销毁，模拟器重新回到 Home 界面。

2.2 使用 Intent

2.2.1 显式 Intent

Intent 是一个消息传递对象，不仅应用程序内各大组件（Content Provider 除外）之间的通信可以使用 Intent，不同应用程序之间也可以使用 Intent 来通信，Intent 会在运行



时将各个组件相互绑定。Intent 定义的消息用于启动特定组件或者特定类型的组件，Intent 可以是显式的，也可以是隐式的。

显式 Intent 指的是使用完全限定类名来指定要启动的组件。由于开发者知道需要启动的 Activity 或者 Service 的类名，应用程序内部各组件之间的通信通常会使用显式 Intent。创建显式 Intent 启动应用组件时，Android 系统将立即启动该 Intent 指定的对象。

在 HelloWorld 程序中，已经创建完成一个 FirstActivity，现在再新建一个 SecondActivity，同样在 SecondActivity 中添加一个 Button，布局如图 2-9 所示。

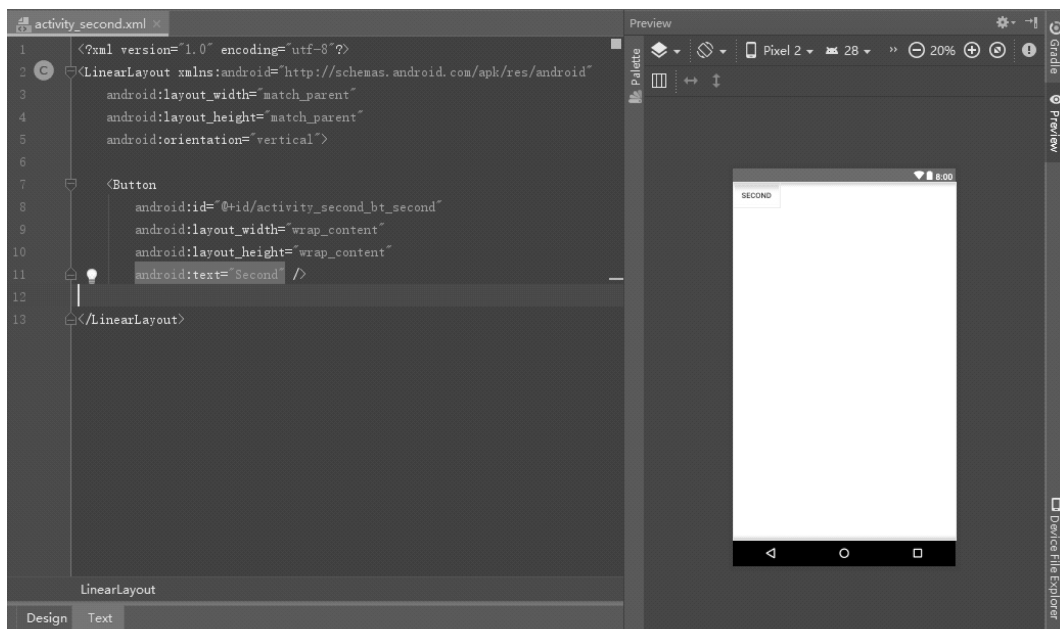


图 2-9 SecondActivity 布局界面

“HelloWorld”程序已有主 Activity，SecondActivity 作为一个普通 Activity 注册到清单中，不需要添加 <intent-filter> 标签中的内容。将 SecondActivity 注册到清单中，代码如下：

```
<manifest ... >
    <application ... >
        <activity android: name = " . MainActivity" ... > </activity>
        <activity android: name = " . activity. FirstActivity" >
            <intent-filter>
                <action android: name = " android. intent. action. MAIN" />
```




```
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

</activity>

<activity android:name=".activity.SecondActivity" ></activity>

</application>

</manifest>
```

在 2.1.4 节中，已设置“FIRST”按钮的单点击事件，当用户单点击该按钮的时候，销毁 FirstActivity，现在监听“FIRST”按钮，当用户单点击该按钮的时候，启动 SecondActivity。在 FirstActivity 中添加如下代码：

```
public class FirstActivity extends Activity {
    @Override
    protected void onCreate (@Nullable Bundle savedInstanceState) {
        super.onCreate ( savedInstanceState );
        setContentView ( R.layout.activity_first );
        findViewById ( R.id.activity_first_bt_first ).setOnClickListener ( new
View.OnClickListener () {
            @Override
            public void onClick ( View view ) {
                Intent intent = new Intent ();
                intent.setClass ( FirstActivity.this, SecondActivity.class );
                startActivity ( intent );
            }
        } );
    }
}
```

创建一个 Intent 的实例，使用 Intent 的 setClass () 方法传递两个参数：第一个参数表示当前 Activity，第二个参数是需要启动的 Activity，在此 Intent 明确指定 SecondActivity 即为需要启动的目标 Activity。再使用 Activity 的 startActivity () 方法，将 Intent 实例作为参数传递给该方法，启动 SecondActivity。重新运行 HelloWorld 程序，部署完成后，单击“FIRST”按钮，页面跳转至 SecondActivity，界面如图 2-10 所示。

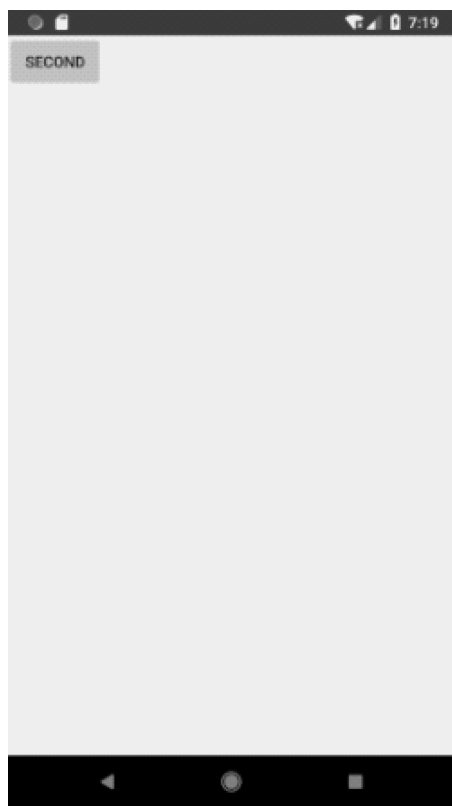


图 2-10 SecondActivity 部署界面

2.2.2 隐式 Intent

隐式 Intent 不会明确指定需要启动的目标组件，而是通过声明指出需要执行的常规操作，从而允许其他应用中的组件进行处理。创建隐式 Intent 时，Android 系统通过将 Intent 的内容与在设备上其他应用的清单文件中声明的 Intent 过滤器进行比较，从而找到需要启动的相应组件。如果 Intent 与 Intent 过滤器匹配，则系统将启动该组件，并向其传递 Intent 对象。如果多个 Intent 过滤器兼容，则系统会显示一个对话框，支持用户选取需要使用的应用。

Intent 过滤器是应用清单文件中的一个表达式，指定了该过滤器的组件能够接收的 Intent 类型。例如，在 HelloWorld 程序中，SecondActivity 暂时没有声明任何 Intent 过滤器，所以只能使用显式 Intent 来启动 SecondActivity。下面给 SecondActivity 声明一个 Intent 过滤器，然后在 FirstActivity 中使用隐式 Intent 进行启动。

在 HelloWorld 程序的清单文件中给 SecondActivity 声明 Intent 过滤器，代码如下：



```
<manifest ... " >
    <application ... >
        <activity android: name = " . MainActivity" ... > </activity >
        <activity android: name = " . activity. FirstActivity" >
            <intent-filter >
                <action android: name = " android.intent.action.MAIN" / >
                <category android: name = " android.intent.category.LAUNCHER" / >
            </intent-filter >
        </activity >
        <activity android: name = " . activity. SecondActivity" >
            <intent-filter >
                <action android: name = " com.ydy.helloworld.start.second" / >
                <category
android: name = " android.intent.category.DEFAULT" / >
            </intent-filter >
        </activity >
    </application >
</manifest >
```

在 `<action>` 标签中声明了 `SecondActivity` 能够响应的 `action`，`<category>` 标签中声明了一种默认的 `category`。修改 `FirstActivity` 中的“FIRST”按钮的单击事件，代码如下：

```
findViewById (R.id.activity_ first_ bt_ first) .setOnClickListener
( new View.OnClickListener () {
    @Override
    public void onClick ( View view) {
        Intent intent = new Intent ();
        intent.setAction ( " com.ydy.helloworld.start.second");
        startActivity ( intent);
    }
});
```

重新运行 `HelloWorld` 程序，部署完成后，单击“FIRST”按钮，页面跳转至 `SecondActivity`。



2.2.3 向下一个 Activity 传递数据

在前面两节中，已经学习了 Activity 之间的启动，现在将学习在启动 Activity 的同时，向目标 Activity 传递数据。

在 HelloWorld 程序中，从 FirstActivity 启动 SecondActivity，并向 SecondActivity 传递一个字符串：this is our first app。在 FirstActivity 中添加如下代码：

```
findViewById (R.id.activity_ first_ bt_ first) .setOnClickListener (
new View.OnClickListener () {
    @Override
    public void onClick (View view) {
        Intent intent
            = new Intent (FirstActivity.this, SecondActivity.class);
        intent.putExtra (" data", " this is our first app");
        startActivity (intent);
    }
});
```

Intent 有多个构造函数的重载，这里使用的构造函数包含两个参数：第一个为当前 Activity，第二个参数为目标 Activity。Intent 提供了一系列 putExtra () 方法的重载，将需要传递的数据通过该方法暂存在 Intent 中并传递给下一个 Activity。putExtra () 方法有两个参数：第一个参数是传递数据的键，第二个参数才是需要传递数据的值。FirstActivity 把字符串传递给 SecondActivity，SecondActivity 接收该字符串，并且将其打印出来。在 SecondActivity 中添加如下代码：

```
public class SecondActivity extends Activity {
    @Override
    protected void onCreate (@Nullable Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setContentView (R.layout.activity_ second);
        Intent intent = getIntent ();
        String data = intent.getStringExtra (" data");
        Log.i (" SecondActivity", " onCreate: " + data);
    }
}
```



在 SecondActivity 中，使用 getIntent（）拿到启动 SecondActivity 的 Intent，然后调用 Intent 的 getStringExtra（）方法获取数据，该方法有一个参数，需要传递 FirstActivity 传递的数据的键值，最后用 Android 的 Log 类将 SecondActivity 接收到的数据打印出来。重新运行 HelloWorld 程序，部署完成之后，单击“FIRST”按钮，页面跳转至 SecondActivity，打开 Logcat 查看日志信息（见图 2-11），SecondActivity 接收到的数据与 FirstActivity 传递的数据一致。

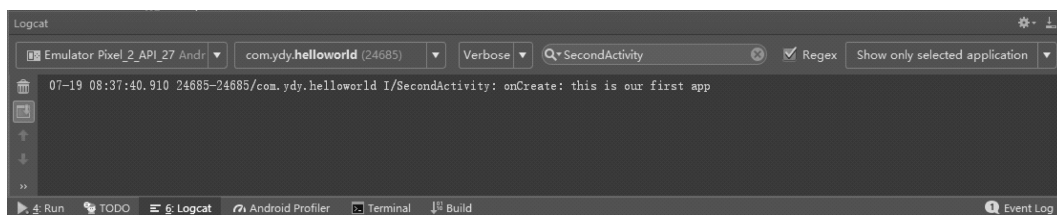


图 2-11 SecondActivity 接收到的数据

2.2.4 向上一个 Activity 返回数据

在实际应用中，可能会遇到将当前 Activity 中的数据回传给启动当前 Activity 的上一个 Activity。下面进行详细讲解。

从 FirstActivity 跳转至 SecondActivity，在 FirstActivity 中编写如下代码：

```
public class FirstActivity extends Activity {  
    public static final int REQUEST_ CODE = 0x101;  
    @Override  
    protected void onCreate (@Nullable Bundle savedInstanceState) {  
        super.onCreate ( savedInstanceState );  
        setContentView ( R.layout.activity_ first );  
        findViewById ( R.id.activity _ first _ bt _ first ) .setOnClickListener ( new  
View.OnClickListener () {  
            @Override  
            public void onClick ( View view ) {  
                Intent intent = new Intent ( FirstActivity.this, SecondActivity.class );  
                startActivityForResult ( intent, REQUEST_ CODE );  
            }  
        } );  
    }  
}
```



```
@Override
protected void onActivityResult (int requestCode, int resultCode, Intent data) {
    super.onActivityResult (requestCode, resultCode, data);
    if (data != null) {
        switch (requestCode) {
            case REQUEST_CODE:
                switch (resultCode) {
                    case RESULT_OK:
                        Log.i (" FirstActivity", " onActivityResult: " + data.getStringExtra (" data"));
                        break;
                    case RESULT_CANCELED:
                        break;
                    default:
                        break;
                }
            break;
        default:
            break;
        }
    }
}
```

监听“FIRST”按钮，当单击该按钮的时候，使用 `startActivityForResult()` 跳转至 `SecondActivity`，`startActivityForResult()` 将接收两个参数：第一个参数为 `Intent`，指定目标启动 `Activity`；第二个参数为请求码，此处自定义一个 `requestCode`。`onActivityResult()` 方法是接收 `SecondActivity` 返回的数据，`onActivityResult()` 方法含三个参数：第一个参数是在 `startActivityForResult()` 方法中传递的请求码；第二参数是 `SecondActivity` 返回数据时传入的处理结果；第三个参数是 `Intent`，包含返回的数据。接收到返回数据之后用 `Log` 类打印出来。在 `SecondActivity` 中添加如下代码：



```
public class SecondActivity extends Activity {  
    @Override  
    protected void onCreate (@Nullable Bundle savedInstanceState) {  
        super.onCreate ( savedInstanceState );  
        setContentView ( R.layout.activity_ second );  
        findViewById ( R.id.activity_ second _ bt _ second ) .setOnClickListener ( new  
View.OnClickListener () {  
            @Override  
            public void onClick ( View view ) {  
                Intent intent = new Intent ( );  
                intent.putExtra ( " data", " hello world" );  
                setResult ( RESULT_ OK, intent );  
                finish ( );  
            }  
        } );  
    }  
}
```

监听 SecondActivity 中的“SECOND”按钮，当单击该按钮时，通过 setResult () 方法回传数据，setResult () 方法中有两个参数；第一个参数表示处理结果，第二个参数是 Intent，构建一个 Intent 用于传递数据。最后销毁 SecondActivity，页面返回 FirstActivity。重新运行 HelloWorld，部署完成之后，单击“FIRST”按钮，页面跳转至 SecondActivity，单击“SECOND”按钮，SecondActivity 销毁，页面返回 FirstActivity，打开 Logcat 查看日志（见图 2-12），FirstActivity 接收到的数据与 SecondActivity 返回的数据一致。

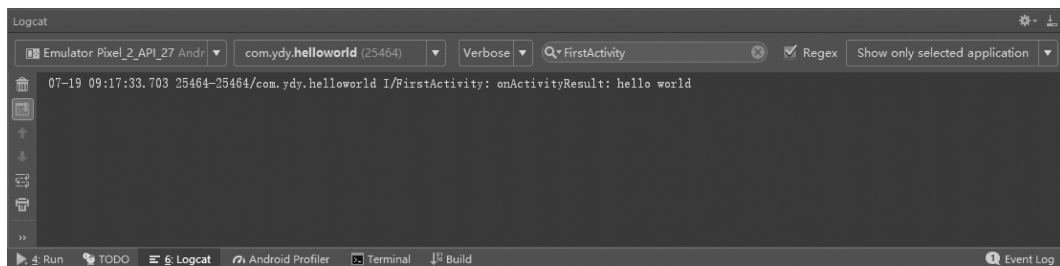


图 2-12 FirstActivity 接收到的数据



2.3 Activity 生命周期

2.3.1 返回栈

任务（tasks）是指执行特定工作时与用户交互的一系列 Activity 的集合，这些 Activity 按照它们各自打开的顺序排列在堆栈，即返回栈（Back Stack）中。栈是一种运算受限的线性表，仅允许在表的一端进行插入和删除运算，即栈顶，另一端为栈底。

Android 使用任务来管理 Activity。当 Android 系统启动一个新的 Activity，它会被压入栈中，处于栈顶，通常处于栈顶的 Activity 就是与用户交互的 Activity。如果从当前 Activity 启动另一个新的 Activity，那么新的 Activity 又会被压入栈中处于栈顶，先前的 Activity 保留在栈中，处于栈底。当用户单击“Back”键的时候，新的 Activity 出栈，Activity 被销毁，先前的 Activity 恢复，获得焦点，处于栈顶。堆栈遵循“后进先出”原则。

Android 设备的 Home 界面是大多数任务开始的位置，当用户单击一个应用程序的图标时程序启动，该应用的任务则被转移到前台。如图 2-13 所示，程序启动后，该程序的主 Activity（Activity 1）压入 Back Stack 中，处于栈顶，Activity 1 处于前台，可以与用户交互。从 Activity 1 启动 Activity 2，Activity 2 压入 Back Stack 中，此时 Activity 2 处于栈顶，获得焦点与用户交互，Activity 1 在栈底，处于后台无法交互。从 Activity 2 启动 Activity 3，Activity 3 压入 Back Stack，处于栈顶获得焦点，在前台能与用户交互，而 Activity 1 和 Activity 2 都保留在栈底，处于后台，不能与用户交互。然后用户单击 Back 键，Activity 3 销毁，Activity 3 出栈，Activity 2 处在栈顶的位置，此时 Activity 2 处于前台与用户交互，Activity 1 还在栈底，处于后台。一个窗口可以有多个任务。

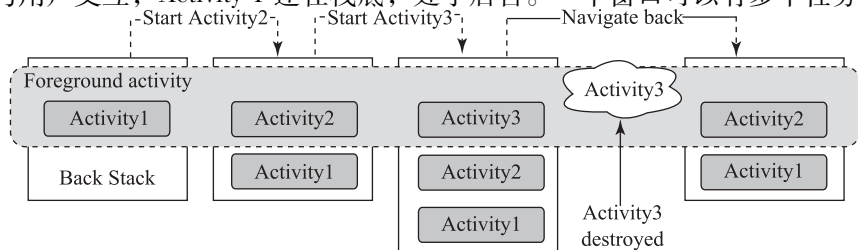


图 2-13 Activity 管理



2.3.2 Activity 状态

Activity 在整个生命周期中，最多会经历四种状态：运行状态、暂停状态、停止状态和销毁状态。

1. 运行状态

当启动一个新的 Activity 时，Activity 被推入栈中处于栈顶，在 Android 系统的前台，此时的 Activity 是可见的，可以与用户交互。而且系统也会尽可能地保持它的活动状态，当资源不够时会杀死其他的 Activity 来确保当前 Activity 有足够资源使用。

2. 暂停状态

假设当前处于前台的 Activity 为 Activity 1，当 Activity 1 启动一个对话框样式的 Activity 时，Activity 1 仍然可见，但是已经失去焦点，无法与用户交互，此时 Activity 1 则处于暂停状态，但依然与窗口管理器保持连接。

3. 停止状态

当一个 Activity 不处于栈顶，且不可见时，该 Activity 就处在停止状态。此时系统仍然会为处于该状态的 Activity 保存其状态和成员变量，但是如果系统资源不足时，就会销毁处于这种状态的 Activity，回收资源。

4. 销毁状态

当一个 Activity 从 Back Stack 中移除之后，该 Activity 就处于销毁状态，系统会回收处于这种状态的 Activity，保证 Android 设备的资源充足。

2.3.3 Activity 的生命周期

如图 2-14 所示，展示了 Activity 的生命周期。在 Activity 的整个生命周期中，Activity 类提供了 onCreate ()、onStart ()、onResume ()、onPause ()、onStop ()、onDestroy ()、onRestart () 七种回调方法。当 Activity 状态改变时，就会回调对应的回调方法。

(1) onCreate ()。此方法是开发者使用最多的方法，在 HelloWorld 程序中较常见此方法。启动一个新的 Activity，系统就会调用 onCreate () 方法开始创建 Activity，开

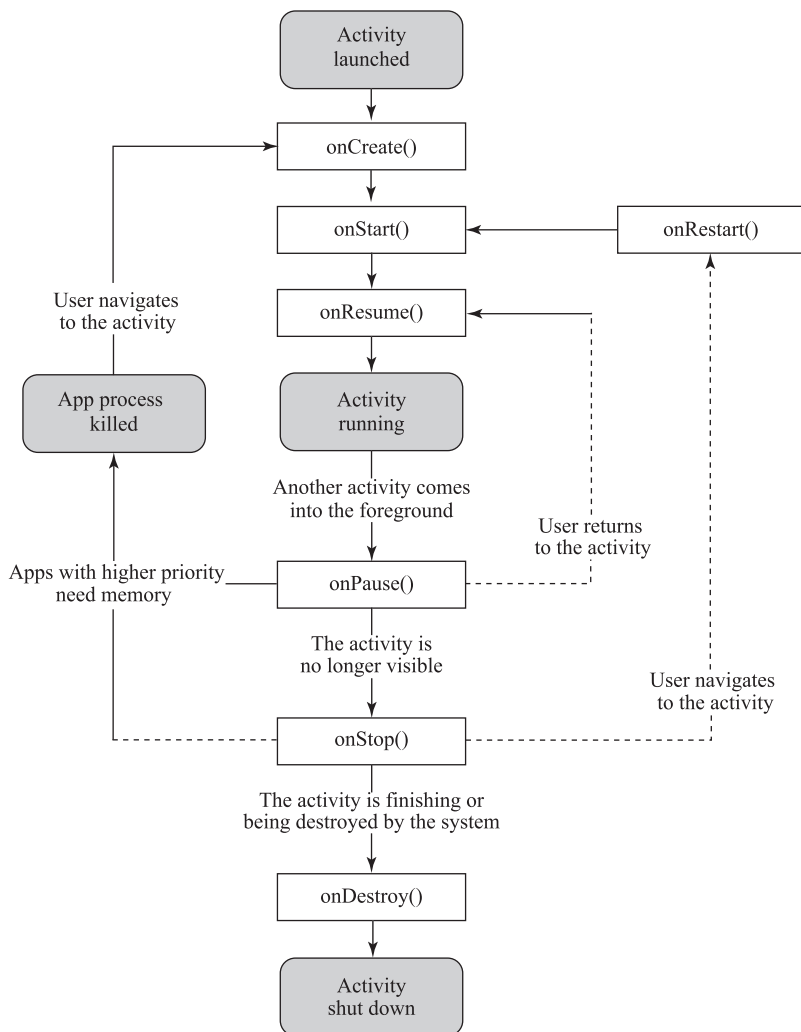


图 2-14 Activity 的生命周期

发者可以在该方法中实现基本的应用程序启动逻辑。每个 Activity 都必须实现该回调方法，该方法在 Activity 的整个生命周期中只执行一次。此时的 Activity 尚不可见。

(2) `onStart()`。在 Activity 由不可见变为可见时系统会调用此方法。开发者可能会在一些应用场景中重写该方法，实现自己的逻辑。

(3) `onResume()`。当 Activity 可见时，系统会调用此方法。此时 Activity 在 Back Stack 的最顶端，处于 Android 的前台，获得焦点，可以与用户进行交互。Activity 在可交互和不可交互之间切换时均会调用该方法。

(4) `onPause()`。Activity 由前台变为后台，失去焦点时会调用该方法。例如 Activity 1 弹出一个对话框样式的 Activity，虽然 Activity 1 仍然部分可见，但其已经失去焦



点，不能与用户进行交互，此时系统仍然会调用该方法。

(5) `onStop()`。当 Activity 转入后台对于用户而言不可见时，该 Activity 进入停止状态，此时系统会调用该方法。

(6) `onDestroy()`。当 Activity 销毁时系统会调用该方法，`onDestroy()` 方法执行完成后，Activity 即被销毁，进入销毁状态。开发者可能会在该方法中释放一些应用程序的资源，让系统回收。

(7) `onRestart()`。在 Activity 由停止状态变成运行状态之前系统会调用该方法。

这七个方法除了 `onRestart()` 方法，其他方法都是一一对应的。

2.4 Activity 启动模式

2.4.1 standard

Activity 包含 `standard`、`singleTop`、`singleTask` 和 `singleInstance` 4 种启动模式，其中 `standard` 是 Activity 的默认启动模式。开发者可以使用清单文件或者使用 Intent 标志来定义 Activity 的启动模式，只要没有显式定义 Activity 的启动模式，都是使用 `standard` 模式启动 Activity。

在 HelloWorld 程序中，所有的 Activity 都是使用 `standard` 模式启动的。在此种模式下，每当启动一个新的 Activity，该 Activity 就会被压入 Back Stack 中，处于栈顶，并可以与用户交互。Activity 可以有多个实例，在默认模式下，每次启动 Activity 都会创建一个新的实例。

现在通过代码进一步直观地认识 `standard` 模式。回到 HelloWorld 程序中，在 `FirstActivity` 中添加如下代码：

```
public class FirstActivity extends Activity {  
    private static final String TAG = " FirstActivity";  
    @Override  
    protected void onCreate (@Nullable Bundle savedInstanceState) {  
        super.onCreate (savedInstanceState);  
        setContentView (R.layout.activity_ first);  
    }  
}
```



```
Log.i (TAG, " onCreate: ");  
findViewById ( R.id.activity _ first _ bt _ first ) .setOnClickListener ( new  
View.OnClickListener () {  
    @Override  
    public void onClick (View view) {  
        Intent intent = new Intent (FirstActivity.this, FirstActivity.class);  
        startActivity (intent);  
    }  
});  
}
```

监听“FIRST”按钮，当单击该按钮的时候，启动一个 FirstActivity，定义一个 tag，作为日志输出的过滤标志，每调用一次 onCreate（）方法会打印一次日志，即每创建一个 FirstActivity 会有一次日志输出。运行 HelloWorld 程序，部署完成后，打开 Logcat 查看日志，发现有一条 onCreate 的日志，单击“FIRST”按钮，发现模拟器屏幕有闪烁，查看 Logcat 日志（见图 2-15），有两条 onCreate 的日志，说明已创建两个 FirstActivity。按“Back”键，销毁一个 Activity，模拟器屏幕有闪烁，但是模拟器屏幕上仍是 FirstActivity 的界面，再按一次“Back”键，FirstActivity 销毁，模拟器回到 Home 界面，确定已创建两个 FirstActivity 实例。

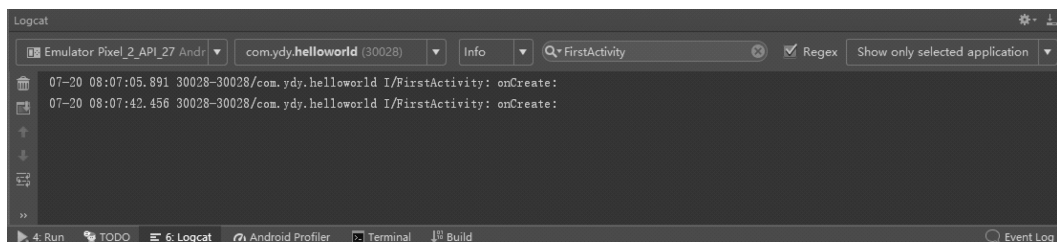


图 2-15 onCreate 日志

2.4.2 singleTop

指定启动模式为 singleTop 模式的 Activity，如果该 Activity 已经处于 Back Stack 的栈顶，此时再去启动它，则不会再创建该 Activity 的实例。如果该 Activity 没有处于 Back Stack 的栈顶，此时再启动它，就会创建该 Activity 的实例，与 standard 模式相同。

回到 HelloWorld 程序，修改 FirstActivity 的启动模式，在清单文件中添加如下代码：



```
<activity
    android: name = " . activity. FirstActivity"
    android: launchMode = " singleTop" >
    <intent-filter >
        <action android: name = " android. intent. action. MAIN" / >
        <category android: name = " android. intent. category. LAUNCHER" / >
    </intent-filter >
</activity >
```

将 FirstActivity 的启动模式修改为 singleTop，重新运行 HelloWorld 程序，部署完成后，打开 Logcat 查看日志，会发现有一条 onCreate 的日志，单击“FIRST”按钮，模拟器屏幕没有闪烁，Logcat 中也没有新的 onCreate 日志输出，重复单击“FIRST”按钮，Logcat 中仍没有新的 onCreate 日志输出，按“Back”键，模拟器回到 Home 界面。说明只创建了一个 FirstActivity，FirstActivity 一直处于栈顶。

如果 FirstActivity 不是在 Back Stack 的栈顶，则会创建 FirstActivity 的实例。

2.4.3 singleTask

使用 singleTask 模式为启动模式的 Activity 在整个 Back Stack 中只有一个实例。当该 Activity 已经处于 Back Stack 的栈顶时，如果去启动它，系统不会创建该 Activity 新的实例，与 singleTop 模式相同；当该 Activity 已经存在但并没有处于 Back Stack 的栈顶时，如果去启动它，那么系统会把 Back Stack 中该 Activity 之上的所有 Activity 全部弹出栈，使该 Activity 处于栈顶。当然，如果 Back Stack 中没有该 Activity 的实例，若去启动它，则会正常启动。

将 FirstActivity 的启动模式改为 singleTask 模式，代码如下：

```
<activity
    android: name = " . activity. FirstActivity"
    android: launchMode = " singleTask" >
    <intent-filter >
        <action android: name = " android. intent. action. MAIN" / >
        <category android: name = " android. intent. category. LAUNCHER" / >
    </intent-filter >
</activity >
```



修改 FirstActivity 中 onCreate () 方法中 “FIRST” 按钮的单击事件，代码如下：

```
@Override
public void onClick ( View view ) {
    Intent intent = new Intent ( FirstActivity. this, SecondActivity. class );
    startActivity ( intent );
}
```

添加 onRestart () 方法，代码如下：

```
@Override
protected void onRestart () {
    super. onRestart ();
    Log. i ( TAG, " onRestart: " );
}
```

修改 SecondActivity 的代码如下：

```
@Override
protected void onCreate ( @Nullable Bundle savedInstanceState ) {
    super. onCreate ( savedInstanceState );
    setContentView ( R. layout. activity_ second );
    Log. i ( " SecondActivity", " onCreate: " );
    findViewById ( R. id. activity _ second _ bt _ second ) . setOnClickListener ( new
View. OnClickListener () {
        @Override
        public void onClick ( View view ) {
            Intent intent = new Intent ( SecondActivity. this, FirstActivity. class );
            startActivity ( intent );
        }
    } );
}

@Override
protected void onDestroy () {
    super. onDestroy ();
    Log. i ( " SecondActivity", " onDestroy: " );
}
```



重新运行 HelloWorld 程序，部署完成后，查看 Logcat 日志，可以看到 FirstActivity 的 onCreate（）已被执行，单击“FIRST”按钮，页面跳转至 SecondActivity，在 Logcat 中可以看到 SecondActivity 的 onCreate（）日志，单击“SECOND”按钮，页面跳转至 FirstActivity，在 Logcat 中有 FirstActivity 的 onRestart（）执行的日志和 SecondActivity 的 onDestroy（）方法执行的日志，按“Back”键，模拟器回到 Home 界面，说明 Back Stack 中仅有一个 FirstActivity 实例。

2.4.4 singleInstance

使用 singleInstance 模式为启动模式的 Activity 将会新建一个 Back Stack 来管理该 Activity，而且新的 Back Stack 中仅有这一个 Activity。singleInstance 模式并不常用，通常用于多个应用共享某一个 Activity 的实例的情况。

再次回到 HelloWorld 程序，修改 FirstActivity 中 onCreate（）方法中日志打印的信息，代码如下：

```
Log.i (" FirstActivity", " onCreate: " + getTaskId ());
```

在清单文件中修改 SecondActivity 的启动模式，代码如下：

```
<activity android: name = " . activity. SecondActivity"
    android: launchMode = " singleInstance" >
...
</activity >
```

修改 SecondActivity 中 onCreate（）方法中日志打印的信息，代码如下：

```
Log.i (" SecondActivity", " onCreate: " + getTaskId ());
```

重新运行 HelloWorld 程序，部署完成后，单击“FIRST”按钮，页面跳转至 SecondActivity，打开 Logcat（见图 2-16），可以发现 FirstActivity 和 SecondActivity 的 task id 不同，说明 SecondActivity 存放在一个单独的 Back Stack 中。

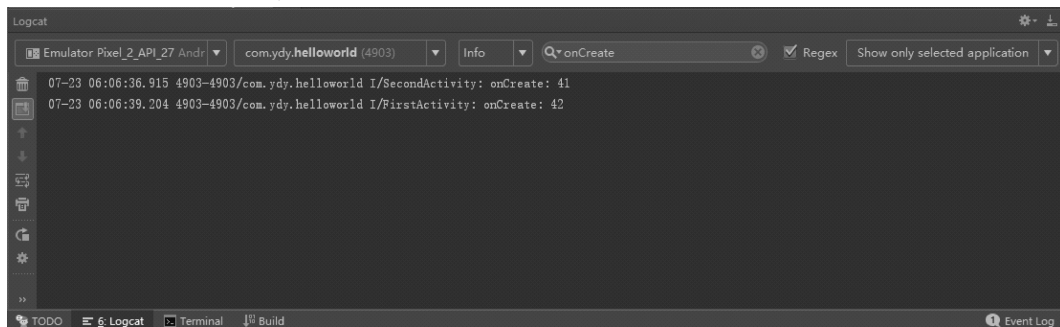


图 2-16 Logcat Task id



练习题

- (1) 创建 Activity 时，默认实现的方法为_____。
- (2) 某 Activity 被某个弹出框覆盖掉一部分时，会调用的方法为_____。
- (3) 创建的 Activity 都必须在 AndroidManifest.xml 文件中进行注册吗？
- (4) 简述 Activity 生命周期中有哪几种状态。
- (5) Android 项目启动时，如何决定最先加载的 Activity？

第3章 UI 界面开发



学习任务

本章主要完成以下学习目标：

- (1) 掌握 Android 常用控件的使用方法。
- (2) 掌握 Android 开发中的常用布局。
- (3) 完成登录注册页面的布局及跳转逻辑。



本章介绍

在学习了 Android 最常用的 Activity 组件之后，需要学习 Activity 的布局文件的编写。本章通过介绍 TextView、Button、EditText 和 ImageView 等元素的使用，让读者对 Android 开发中 UI 控件的使用有一个整体的认识。本章还将学习最重要的布局编写，所有的 Activity 界面都需要使用布局，具有特色的界面是 Android 应用程序吸引用户和增加用户粘度的重要因素。最后通过一个 Activity 界面布局实例让读者实际演练，检验读者对所学知识的掌握程度。



3.1 常用控件

3.1.1 TextView

Android 提供了较多 UI 控件来帮助开发者开发应用程序，本节将介绍一些常用的 Android 控件。Android Studio 提供的可视化编辑器允许开发者使用拖放控件的方式来布局，但是在实际开发中并不提倡此种做法，推荐使用最基本的方式即通过 XML 语言来完成界面的布局。

TextView 控件继承自 View，主要用于不可编辑的文字显示。回到 HelloWorld 程序，新建 UIActivity，并在清单文件中注册为程序入口，代码如下：

```
<activity android:name=".activity.UIActivity" >
    <intent-filter >
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter >
</activity >
```

在 UIActivity 的布局文件 “activity_ui.xml” 文件中添加如下代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/activity_ui_tv"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#FF4081"
        android:text="我是一个 TextView" />
</LinearLayout >
```



除根布局（LinearLayout）之外，在 UIActivity 的布局文件中添加一个 TextView 控件，并给定多个属性，在第 2 章的 2.1.3 节中已经使用过此类属性。id 是唯一标识符，在 Activity 中需要使用 id 来获取该控件的实例。layout_ width 和 layout_ height 分别表示控件的宽度和高度，其值有 3 个：fill_ parent、match_ parent 和 wrap_ content。fill_ parent 和 match_ parent 均表示该控件撑满父控件，即父布局的大小决定了当前控件的大小，不过 fill_ parent 已废弃，官方推荐使用 match_ parent；wrap_ content 表示当前控件刚好包裹住该控件包含的内容。Background 表示该控件的背景颜色，这里给出一个颜色值。text 表示该控件显示的内容。因此上述的代码表示该 TextView 控件的宽度与父布局一样，即屏幕宽度，而高度则是足够包含 text 的内容即可。运行 HelloWorld 程序，部署完成后，（见图 3-1），与预期的效果相同。

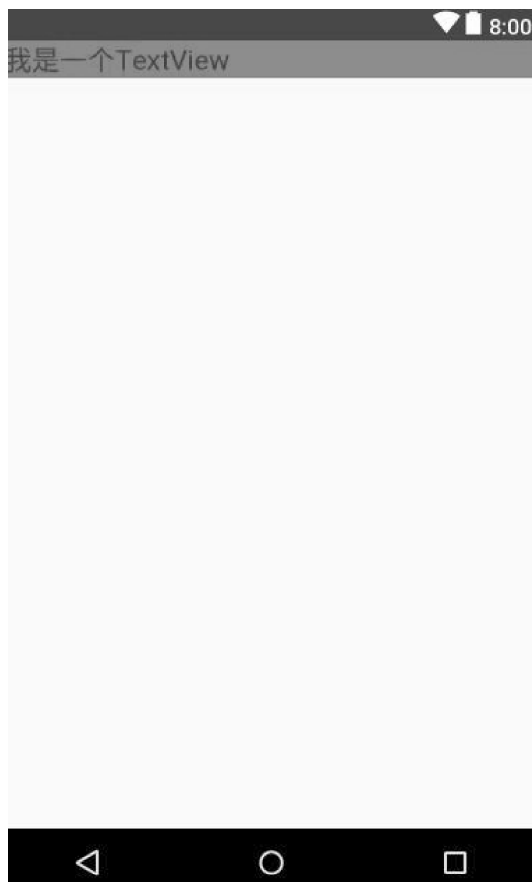


图 3-1 TextView

修改“activity_ ui. xml”文件中的代码如下：



<TextView

```
android: id = " @ + id/activity_ ui_ tv"
android: layout_ width = " match_ parent"
android: layout_ height = " wrap_ content"
android: background = " #FF4081"
android: gravity = " center"
android: text = " 我是一个 TextView"
android: textColor = " #3F51B5"
android: textSize = " 20sp" / >
```

此处新增三个属性：gravity 指定控件内容处于控件中的位置，gravity 常用的值包括 bottom（下）、center（中间）、center_horizontal（横向中间）、center_vertical（竖向中间）、left（左）、right（右）和 top（上）；textColor 属性指定字体颜色，此处已给定一个颜色值；textSize 属性指定字体大小。因此上述的代码表示该 TextView 控件中显示的字体在中间位置，颜色是#3F51B5，字体大小是 20 sp。重新运行 HelloWorld 程序，部署完成后（见图 3-2），文字内容居于控件中间，其大小明显大于默认大小。

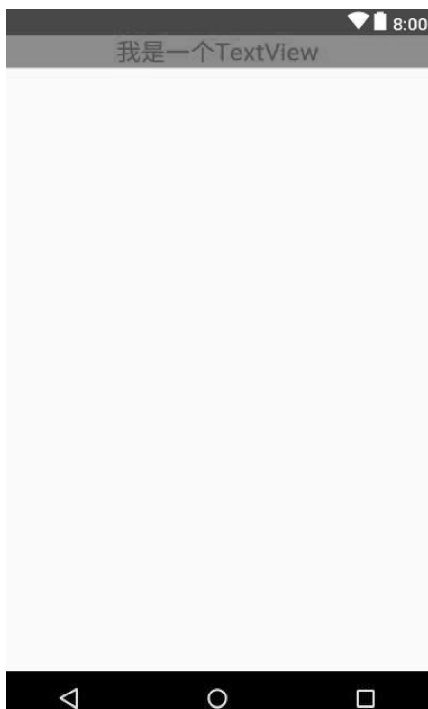


图 3-2 TextView 常用属性设置



上述内容为 TextView 的常用属性，更多属性可以参考 Android 的 API 文档，也可以访问 <https://developer.android.google.cn/reference/packages> 查看。

3.1.2 Button

Button 继承自 TextView，拥有 TextView 的所有属性，主要用来监听用户的点击交互。在 “activity_ ui. xml” 文件中添加如下代码：

```
<Button
    android: id = " @ + id/activity_ ui_ bt"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ gravity = " center_ horizontal"
    android: background = " #76EE00"
    android: text = " 按钮" />
```

此处使用到一个新的属性：layout_ gravity，其值与 gravity 的值相同，只是 layout_ gravity 表示的是该控件在父布局中的位置。在 UIActivity 中添加如下代码，监听按钮的单击事件：

```
Button button = findViewById (R. id. activity_ ui_ bt);
button. setOnClickListener (new View. OnClickListener () {
    @ Override
    public void onClick (View view) {
        Toast. makeText (UIActivity. this, " 你单击了按钮", Toast. LENGTH_ SHORT) . show
        ();
    }
});
```

根据 Button 的 id 获取该控件的实例，监听它的点击事件，当用户单击该按钮时，会使用 Toast 弹出“你单击了按钮”的提示信息。运行 HelloWorld 程序，部署完成后（见图 3-3），Button 处于整个横向屏幕的中间位置，Button 有一个默认的风格，在实际开发中，开发者往往需要自定义它的风格，单击按钮，Toast 弹出提示。

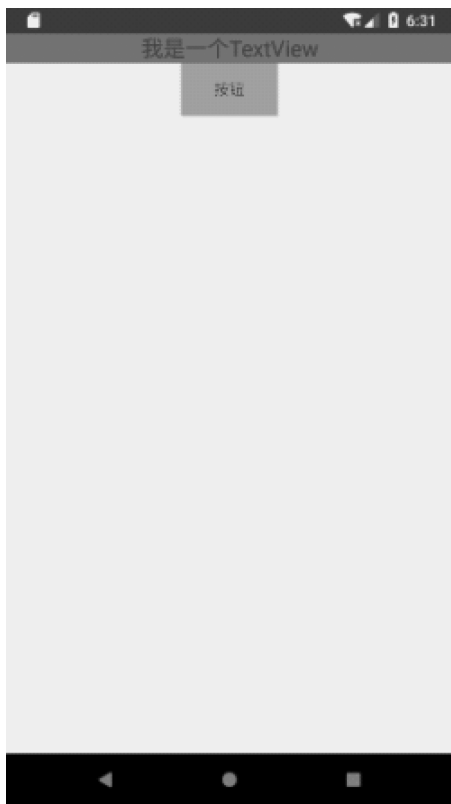


图 3-3 Button

3.1.3 EditText

EditText 继承自 TextView，EditText 最大的特点是可以输入和修改文字内容，是为用户交互的重要控件。在“activity_ui.xml”中添加如下代码：

```
<EditText
    android:id="@+id/activity_ui_et"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="我是一个输入框" />
```

hint 属性是给用户的一个提示，当 EditText 中输入内容后，它将不会再显示，如果输入框中没有内容，则会显示给定的提示。运行程序，部署完成后如图 3-4 所示，输入框中包含一个“我是一个输入框”的信息提示，在输入框中输入文字内容，提示消失，显示的为输入的内容。由于此处指定的 EditText 的高度是 wrap_content，所以当输入内容超过一行的时候，控件就会向下拉长，影响整个页面布局，一般在开发中会使用



maxLines 来指定 EditText 的最大行数，这样就不会影响到页面的美观。

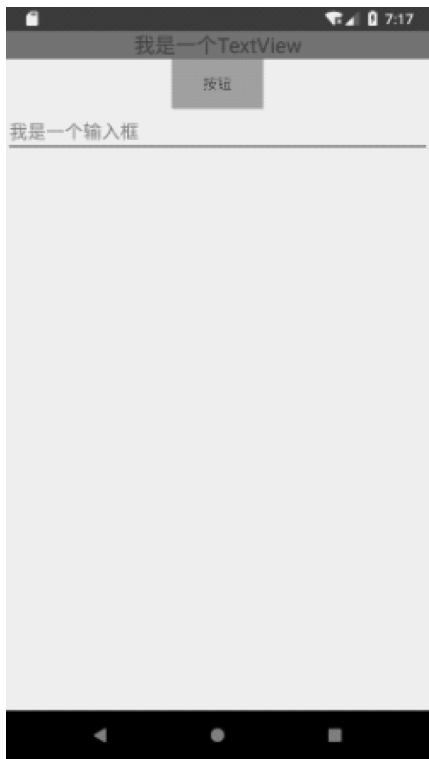


图 3-4 EditText

修改 UIActivity 中的代码如下：

```
Button button = findViewById (R.id.activity_ ui_ bt);  
final EditText editText = findViewById (R.id.activity_ ui_ et);  
button.setOnClickListener (new View.OnClickListener () {  
    @Override  
    public void onClick (View view) {  
        String result = editText.getText () . toString () . trim ();  
        Log.i (" UIActivity", " onClick: " + result);  
    }  
});
```

获取 EditText 的实例，使用 getText () 方法获取输入框中的内容，监听按钮的点击事件，当用户单击按钮时，使用 Log 将 EditText 中的内容打印到 Logcat 中。重新运行程序，部署完成后在输入框中输入 “hello world”，单击按钮，打开 Logcat 查看日志，如图 3-5 所示。

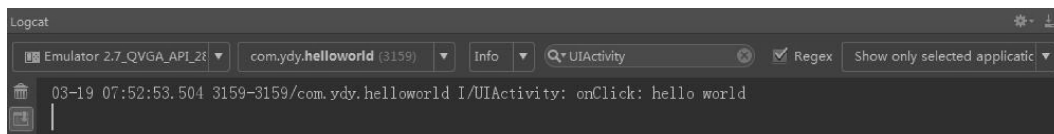


图 3-5 Logcat

3.1.4 ImageView

ImageView 控件继承自 View，主要用于展示图片资源。在“activity_ ui. xml”文件中添加如下代码：

```
< ImageView
    android: id = " @ + id/activity_ ui_ iv"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: src = " @ mipmap/ic_ launcher" / >
```

在其中可以发现，所有的控件使用方法大致相同，均是指定一个 id，然后给定控件的宽度和高度，再加上控件的特有属性即可。此处的 src 属性指定 ImageView 加载的图片资源。在 Android Studio 中，所有图片资源都放在 mipmap 目录下。运行程序，部署完成后如图 3-6 所示。



图 3-6 ImageView



3.1.5 AlertDialog

AlertDialog 作为一个弹出框，可以在当前 Activity 上弹出一个对话框，浮于整个界面之上。AlertDialog 一般用来给用户提示重要信息。在 UIActivity 类中添加如下代码：

```
private void showDialog () {  
    AlertDialog.Builder builder = new AlertDialog.Builder (this);  
    builder.setTitle (" 对话框标题")  
        .setMessage (" 对话框内容")  
        .setCancelable (false)  
        .setNegativeButton (" 取消", new DialogInterface.OnClickListener () {  
            @Override  
            public void onClick (DialogInterface dialogInterface, int i) {  
                //点击取消按钮执行的逻辑  
            }  
        }) .setPositiveButton (" 确定", new DialogInterface.OnClickListener () {  
            @Override  
            public void onClick (DialogInterface dialogInterface, int i) {  
                //点击确定按钮执行的逻辑  
            }  
        }) .show ();  
}
```

在这个方法中创建一个 AlertDialog，给该对话框设置一个标题以及一个内容信息。setCancelable (false) 表示单击对话框外的空白处或者按“Back”键，对话框不会消失，setNegativeButton () 方法给该对话框设置了一个“取消”按钮，并监听这个按钮的单击事件，setPositiveButton () 方法给该对话框设置了一个“确定”按钮，并监听这个按钮的单击事件。

修改在第3章 3.1.2 节中创建的 Button 的监听事件，按钮的单击事件中调用 showDialog () 方法。运行程序，部署完成后，单击按钮，弹出对话框（见图 3-7），单击对话框外的空白处或者按“Back”键，对话框不消失，单击“取消”或者“确定”按钮，对话框消失。



图 3-7 AlertDialog

3.2 常用布局

3.2.1 线性布局

布局定义用户界面的视觉结构，布局的方式通常包含两种：一是直接在 XML 文件中声明 UI 元素；二是运行时实例化布局元素。在实际应用中，用户界面通常是由多个控件组成的复杂布局。

LinearLayout 即线性布局，是 Android 中最常用的布局之一。线性布局将其所包含的控件在水平或垂直方向上依次堆叠。在前面的学习中，已经多次使用 LinearLayout，orientation 属性指定 LinearLayout 的排列方向，其包含的两个值 vertical 表示垂直方向，horizontal 表示水平方向。

打开 HelloWorld 程序新建 LinearActivity，并将其设置为程序入口。在 LinearActivity 的布局文件 “activity_linear.xml” 中添加如下代码：



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:layout_width="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>
```

此处 LinearLayout 使用垂直布局，orientation 属性的值为 vertical。在根节点下放置三个 Button，依次从上到下排列，它们的宽度和高度均是 wrap_content。运行程序，部署完成后界面如图 3-8 所示，与预期结果一致。需要注意的是，垂直线性布局中，放置在前面的控件的高度不能设置为 match_parent，否则前面的控件已经占满整个屏幕，后面的控件则被排挤到屏幕之外，用户将无法在屏幕上看到它们。

修改“activity_linear.xml”文件中根节点的 orientation 的值，代码如下所示：

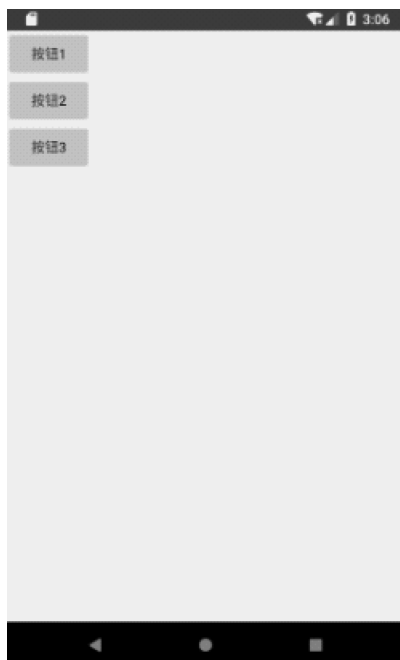


图 3-8 LinearLayout 垂直布局



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

</LinearLayout>
```

现将 LinearLayout 改成水平线性布局，这三个 Button 则变为横向水平排列在屏幕上。同理，水平线性布局中，前面的控件的宽度不能设置为 match_parent，以免将其后的控件排挤出屏幕。运行程序，部署完成后的布局效果，如图 3-9 所示。

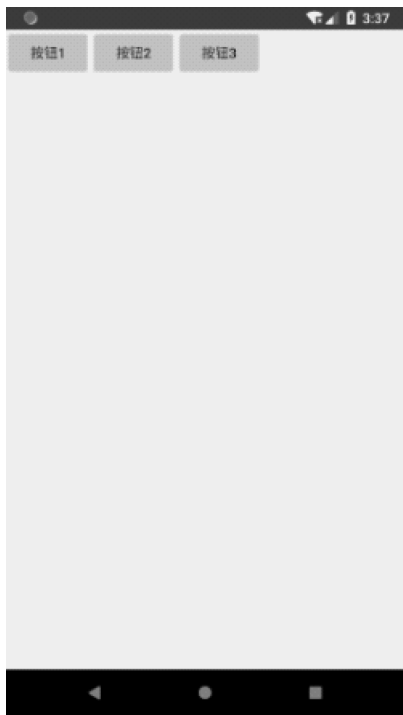


图 3-9 LinearLayout 水平布局

线性布局的另一个重要的属性为布局权重 layout_weight。LinearLayout 支持使用 layout_weight 属性为其包含的子控件分配权重，修改“activity_linear.xml”文件中的代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
```



```
< Button
    android:layout_width = " 0dp"
    android:layout_height = " wrap_ content"
    android:layout_weight = " 1"
    android: text = " 按钮 1" />
< Button
    android:layout_width = " 0dp"
    android:layout_height = " wrap_ content"
    android:layout_weight = " 1"
    android: text = " 按钮 2" />
< Button
    android:layout_width = " 0dp"
    android:layout_height = " wrap_ content"
    android:layout_weight = " 1"
    android: text = " 按钮 3" />
</LinearLayout>
```

现将三个 Button 的宽度设为 0 dp，添加 layout_weight 为 1，表示水平方向上三个 Button 控件平分整个屏幕。运行程序，部署完整后的布局效果，如图 3-10 所示。

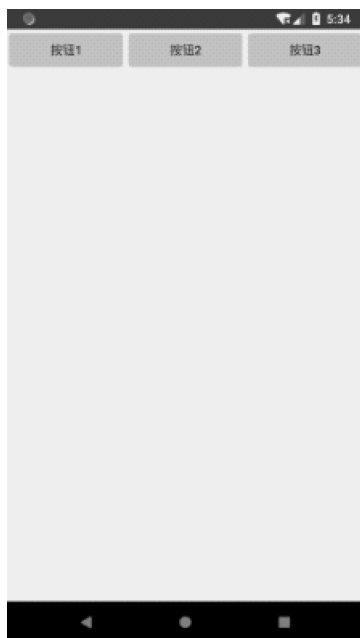


图 3-10 LinearLayout 权重布局



3.2.2 相对布局

RelativeLayout 即相对布局，也是 Android 开发中常使用的布局。相较于 LinearLayout，RelativeLayout 的使用更加灵活，在实际开发应用中，布局较复杂的界面通常会使用 RelativeLayout。RelativeLayout 具有多种属性，下面具体介绍。

返回至 HelloWorld 程序，新建 RelativeLayout，并将其设置为程序入口，在 RelativeLayout 的布局文件 “activity_relative.xml” 中添加如下代码：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        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:layout_alignParentRight="true"
        android:text="按钮 3" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:text="按钮 4" />
```



```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentBottom="true"  
    android:layout_alignParentRight="true"  
    android:text="按钮 5" />  
</RelativeLayout>
```

layout_centerInParent 属性指定控件是否处于父布局的中心位置, layout_alignParentLeft 指定控件是否处于父布局的左边, layout_alignParentRight 指定控件是否处于父布局的右边, layout_alignParentBottom 指定控件是否处于父布局的下边。它们的值均是一个布尔值。运行程序, 部署成功后布局如图 3-11 所示。

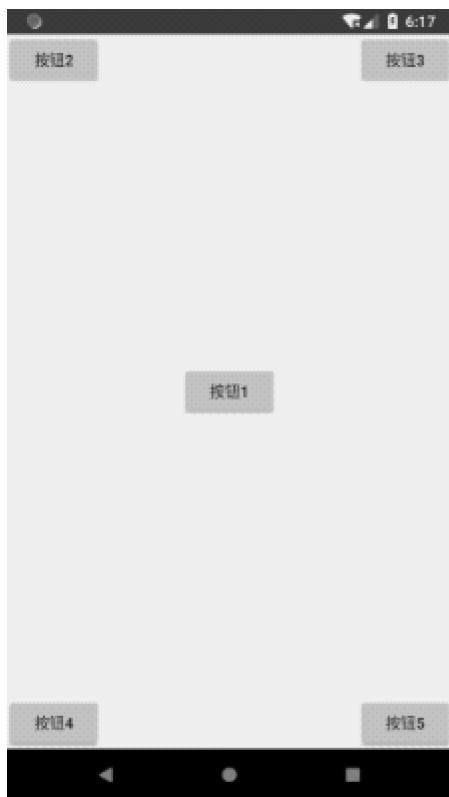


图 3-11 RelativeLayout

修改 activity_relative.xml 中的代码如下:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"
```



```
android: layout_ height = " match_ parent" >
< Button
    android: id = " @ + id/activity_ relative_ bt_ one"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ centerInParent = " true"
    android: text = " 按钮 1" />
< Button
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ above = " @ + id/activity_ relative_ bt_ one"
    android: layout_ toLeftOf = " @ + id/activity_ relative_ bt_ one"
    android: text = " 按钮 2" />
< Button
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ above = " @ + id/activity_ relative_ bt_ one"
    android: layout_ toRightOf = " @ + id/activity_ relative_ bt_ one"
    android: text = " 按钮 3" />
< Button
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ below = " @ + id/activity_ relative_ bt_ one"
    android: layout_ toLeftOf = " @ + id/activity_ relative_ bt_ one"
    android: text = " 按钮 4" />
< Button
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ below = " @ + id/activity_ relative_ bt_ one"
    android: layout_ toRightOf = " @ + id/activity_ relative_ bt_ one"
    android: text = " 按钮 5" />
</RelativeLayout>
```




`layout_ above` 指定该控件在某个控件之上, `layout_ below` 指定该控件在某个控件之下, `layout_ toLeftOf` 指定该控件在某个控件左边, `layout_ toRightOf` 指定该控件在某个控件右边。这几个属性的值均是某个控件的 `id`。运行程序, 部署完成后的处理如图 3-12 所示。

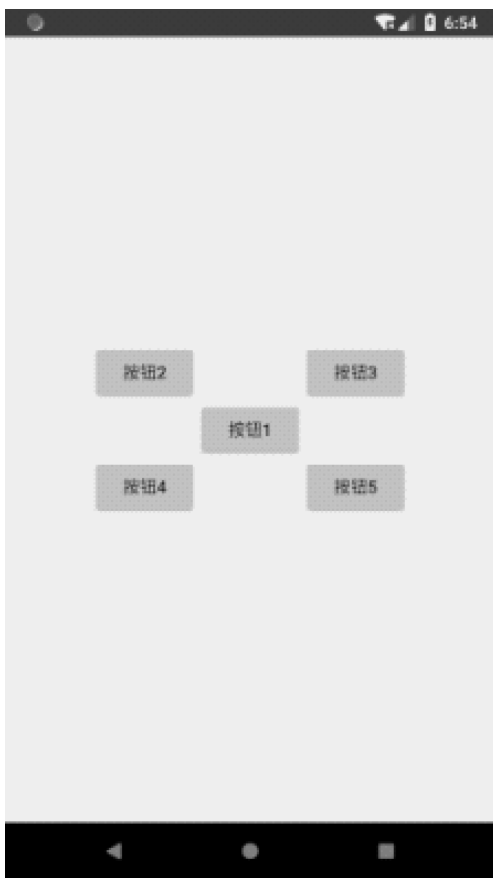


图 3-12 RelativeLayout 布局

3.2.3 帧布局

FrameLayout 即帧布局, 在实际开发中使用较少, 使用方法也较为简单。在 `HelloWorld` 程序中, 新建 `FrameActivity`, 并将其设置为程序入口, 在 `FrameActivity` 的布局文件 “`activity_ frame. xml`” 中添加如下代码:

```
<FrameLayout xmlns: android = " http://schemas. android. com/apk/res/ android"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent" >
```



```
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="20dp"
    android:background="#FF0000" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="50dp"
    android:background="#00FF00" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="80dp"
    android:background="#0000FF" />
<TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="120dp"
    android:background="#000000" />
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:src="@mipmap/ic_launcher" />
</FrameLayout>
```

layout_margin 指定控件的外间距，在帧布局中依次叠加红色、绿色、蓝色和黑色四个背景颜色的 TextView，使用不同的外间距来展示效果，最后添加一个 ImageView 放在屏幕中间，运行程序，效果如图 3-13 所示，可以发现 FrameLayout 中在后面放置的控件会覆盖前面的控件。

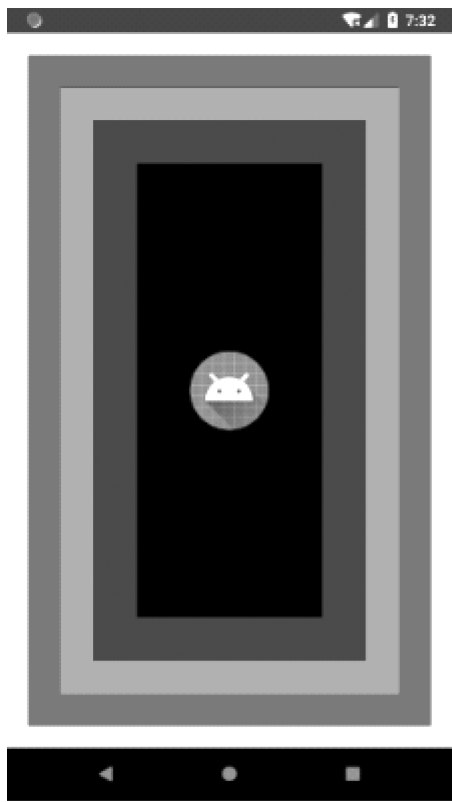


图 3-13 帧布局

3.2.4 约束布局

ConstraintLayout 即约束布局，是新版 Android Studio 中默认的布局方式。前面介绍的布局虽然同样支持使用可视化的方式编写 Android 界面，但是有较大的局限性，本书不推荐此种方式编写布局，通过使用 XML 代码来完成布局，而 ConstraintLayout 是为了解决传统 Android 开发中通过编写 XML 代码完成布局的现状而设计，它较适合在可视化编辑器中通过拖拽控件来完成界面布局，当然读者也可以使用 XML 代码来实现布局。

在 ConstraintLayout 出现之前，Android 开发者编写复杂的布局时常伴随着多重嵌套，影响程序性能，而 ConstraintLayout 可有效解决布局嵌套过多的问题，ConstraintLayout 使用约束的方式来指定各个控件的位置和关系，实现完全扁平的层次结构布局，而不必嵌套，提高绘制效率，与 RelativeLayout 相似，但比 RelativeLayout 功能强大。下面介绍 ConstraintLayout 的使用。



在 HelloWorld 程序中新建 ConstraintActivity，并将其设置为程序入口，在 ConstraintActivity 的布局文件 “activity_constraint.xml” 中添加如下代码：

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="中间"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="右上"
        app:layout_constraintBottom_toTopOf="@+id/button1"
        app:layout_constraintStart_toEndOf="@+id/button1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="左上"
        app:layout_constraintBottom_toTopOf="@+id/button1"
        app:layout_constraintEnd_toStartOf="@+id/button1" />
```



```
<Button
    android: id = " @ + id/button4"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: text = " 左下"
    app: layout_ constraintEnd_ toStartOf = " @ + id/button1"
    app: layout_ constraintTop_ toBottomOf = " @ + id/button1" />

<Button
    android: id = " @ + id/button5"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: text = " 右下"
    app: layout_ constraintStart_ toEndOf = " @ + id/button1"
    app: layout_ constraintTop_ toBottomOf = " @ + id/button1" />
</android.support.constraint.ConstraintLayout>
```

在界面中放置了 5 个按钮，以一个按钮为中心，其他四个按钮环绕在其四周。ConstraintLayout 中 layout_constraintX_ toYOf 格式的属性，表示将当前控件的方向 X 放在参照物控件的方向 Y 处，属性值为参照物控件的 id，如果参照物是父布局（即 ConstraintLayout），则用“parent”来表示。运行程序，部署成功后的效果如图 3-14 所示。

ConstraintLayout 的功能强大，除类似 RelativeLayout 的布局，还可以类似于 LinearLayout 使用 weight 控制控件占比。修改“activity_constraint.xml”文件中的代码如下：



图 3-14 ConstraintLayout 示例



```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/button1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="按钮 1"
        app:layout_constraintHorizontal_chainStyle="spread_inside"
        app:layout_constraintHorizontal_weight="1"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@+id/button2"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="按钮 2"
        app:layout_constraintHorizontal_chainStyle="spread_inside"
        app:layout_constraintHorizontal_weight="1"
        app:layout_constraintLeft_toRightOf="@+id/button1"
        app:layout_constraintRight_toLeftOf="@+id/button3"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/button3"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="按钮 3"
```



```
app:layout_constraintHorizontal_chainStyle="spread_inside"
app:layout_constraintHorizontal_weight="1"
app:layout_constraintLeft_toRightOf="@+id/button2"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

在界面上方放置三个按钮，将 `constraintHorizontal_chainStyle` 属性设为 `spread_inside`，将 `constraintHorizontal_weight` 属性设为 1，让三个按钮横向平分屏幕。运行程序，部署完成后的效果如图 3-15 所示。

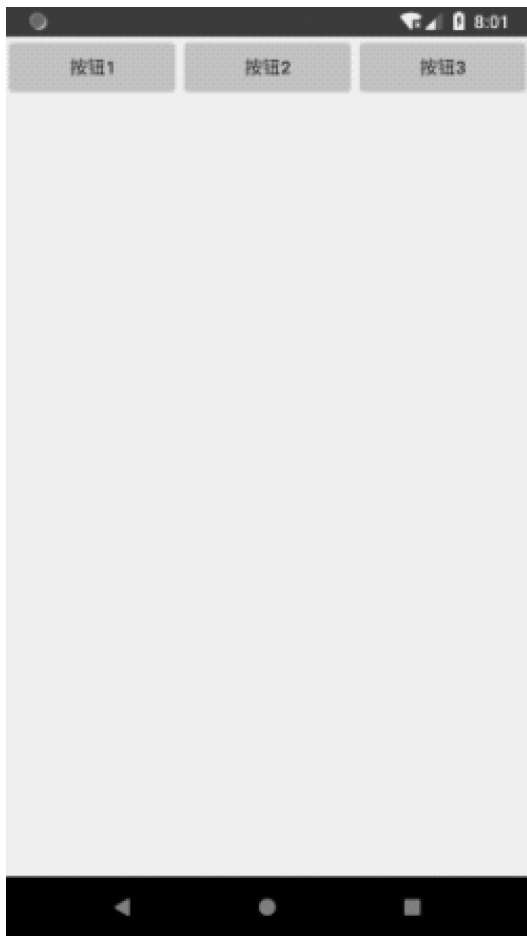


图 3-15 ConstraintLayout

ConstraintLayout 几乎包含现有布局所有的功能，随着 Android 的愈发完善，ConstraintLayout 必将成为 Android 开发中的主流布局样式。



3.3 编写登录注册界面

3.3.1 编写登录界面

创建一个新的项目，取名为“ChonqingApp”，创建登录页面（LoginActivity），设为程序入口，单击“登录”按钮进入程序主界面（MainActivity），单击“注册”按钮进入注册页面（RegisterActivity），如图 3-16 所示。



图 3-16 登录界面

代码如下：



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:background="#303F9F" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:text="登录"
            android:textColor="#FFFFFF"
            android:textSize="20dp" />

    </RelativeLayout>

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:scaleType="center"
        android:src="@mipmap/logo" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:orientation="horizontal"
        android:paddingLeft="20dp"
        android:paddingRight="20dp" >

        <TextView
```



```
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:text="用户名" />

<EditText
    android:id="@+id/activity_login_et_username"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:hint="请输入用户名"
    android:inputType="text" />

</LinearLayout>

    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:orientation="horizontal"
    android:paddingLeft="20dp"
    android:paddingRight="20dp" >

    <TextView
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:text="密码" />

    <EditText
        android:id="@+id/activity_login_et_password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:hint="请输入密码"
        android:inputType="textPassword" />

</LinearLayout>

<RelativeLayout
    android:layout_width="match_parent"
```



```
android:layout_height="wrap_content"
android:layout_marginLeft="20dp"
android:layout_marginRight="20dp"
android:layout_marginTop="10dp" >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingBottom="5dp"
    android:paddingRight="15dp"
    android:paddingTop="5dp"
    android:text="忘记密码" />

<TextView
    android:id="@+id/activity_login_tv_register"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:paddingBottom="5dp"
    android:paddingLeft="15dp"
    android:paddingTop="5dp"
    android:text="注册" />

</RelativeLayout>

<Button
    android:id="@+id/activity_login_bt_login"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="100dp"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="20dp"
    android:text="登录" />
```

```
</LinearLayout>
```



3.3.2 编写注册界面

编写如图 3-17 所示的注册界面，代码如下：



图 3-17 注册界面

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:background="#303F9F" >

        <ImageView
            android:id="@+id/activity_register_iv_back"
```



```
android:layout_width="40dp"
android:layout_height="match_parent"
android:paddingBottom="5dp"
android:paddingTop="5dp"
android:src="@mipmap/back" />
```

```
<TextView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
android:text="注册"
android:textColor="#FFFFFF"
android:textSize="20dp" />
```

```
</RelativeLayout>
```

```
<LinearLayout
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="50dp"
android:orientation="horizontal"
android:paddingLeft="20dp"
android:paddingRight="20dp" >
```

```
<TextView
```

```
android:layout_width="60dp"
android:layout_height="wrap_content"
android:text="用户名" />
```

```
<EditText
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="10dp"
android:inputType="text"
android:hint="请输入用户名" />
```



```
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:orientation="horizontal"
    android:paddingLeft="20dp"
    android:paddingRight="20dp" >

    <TextView
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:text="密码" />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:inputType="textPassword"
        android:hint="请输入密码" />

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:orientation="horizontal"
    android:paddingLeft="20dp"
    android:paddingRight="20dp" >

    <TextView
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:text="确认密码" />
```



```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:inputType="textPassword"
    android:hint="请确认密码" />

</LinearLayout>

<Button
    android:id="@+id/activity_register_bt_register"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"
    android:layout_marginTop="20dp"
    android:text="注册" />

</LinearLayout>
```

3.3.3 实现跳转逻辑

布局页面代码编写完成后，在 Activity 中实现逻辑的跳转。在 LoginActivity 中添加如下代码：

```
public class LoginActivity extends Activity {

    //用户名 密码
    private EditText mEtName, mEtPwd;
    //注册
    private TextView mTvRegister;
    //登录
    private Button mBtLogin;

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



```
setContentView ( R.layout.activity_login );
mEtName = findViewById ( R.id.activity_login_et_username );
mEtPwd = findViewById ( R.id.activity_login_et_password );
mTvRegister = findViewById ( R.id.activity_login_tv_register );
mBtLogin = findViewById ( R.id.activity_login_bt_login );

mTvRegister.setOnClickListener ( clickListener );
mBtLogin.setOnClickListener ( clickListener );

}

View.OnClickListener clickListener = new View.OnClickListener () {
@ Override
public void onClick ( View v ) {
    switch ( v.getId () ) {
        //注册
        case R.id.activity_login_tv_register:
            startActivity ( new Intent ( LoginActivity.this, RegisterActivity.class ) );
            break;
        //登录
        case R.id.activity_login_bt_login:
            startActivity ( new Intent ( LoginActivity.this, MainActivity.class ) );
            break;
        default:
            break;
    }
}
};
}
```

上述代码为监听注册和登录的单击事件，当单击“注册”按钮时，页面跳转至注册页面；当单击“登录”按钮时，页面跳转至主页。

在 RegisterActivity 中添加如下代码：

```
public class RegisterActivity extends Activity implements View.OnClickListener {
//返回
```




```
private ImageView mIvBack;
//注册
private Button mBtRegister;

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

    mIvBack = findViewById ( R.id.activity_register_iv_back );
    mBtRegister = findViewById ( R.id.activity_register_bt_register );

    mIvBack.setOnClickListener ( this );
    mBtRegister.setOnClickListener ( this );
}

@Override
public void onClick ( View v ) {
    switch ( v.getId () ) {
        //返回
        case R.id.activity_register_iv_back:
            finish ();
            break;

        //注册
        case R.id.activity_register_bt_register:
            finish ();
            break;

        default:
            break;
    }
}
```

上述代码为监听返回键和“登录”按钮的单击事件，当单击这两处按钮时，页面返回至登录页面。



练习题

- (1) _____属性指定了 LinearLayout 布局的子控件排列方向。
- (2) 使用 ConstraintLayout 约束布局时，为控件指定一个约束就可以确定其位置吗？
- (3) 创建 Activity 时，可以继承其他基类创建吗？
- (4) 创建 Activity 时，必须创建对应的 layout 布局吗？
- (5) 布局 layout 文件中，控件 id 属性值可以一致吗？

第4章 资源

学习任务

本章主要完成以下学习目标：

- (1) 了解 Android 编程规范。
- (2) 掌握 Android 开发中各种资源的使用。
- (3) 掌握 Drawable 资源在实际开发中的应用。

本章介绍

在软件开发中，较多开发者为方便开发会直接使用硬编码，但实际上这是一种极其不好的编码习惯。在 Android 开发中，Android 为开发者提供了各种资源文件来分别存放开发中所需要用到的资源，以方便维护。本章将介绍在实际开发中使用的 Android 资源文件，提高代码规范程序和编程质量。

4.1 简介

4.1.1 Res 目录下的资源

Android 推荐用户把应用开发中需要用到各种资源（例如字符串资源、颜色资源、图片资源等）放在 res 目录的特定子目录下。此做法便于在程序中直接使用这些资源。



源。表 4-1 所示为 res 目录下支持的资源目录。在 Java 代码中可以使用 R 文件来访问这些资源，在 XML 文件中需要使用“@”字符访问此类资源。

表 4-1-1 res 目录下支持的资源目录

目录	资源类型及访问
animator	用于定义属性动画的 XML 文件，通过 R. animator 类访问
anim	定义渐变动画的 XML 文件，通过 R. anim 类访问
color	用于定义颜色状态列表的 XML 文件，通过 R. color 类访问
drawable	位图文件（.png、.9.png、.jpg、.gif）或编译为以下可绘制对象资源子类型的 XML 文件： • 位图文件 • 九宫格（可调整大小的位图） • 状态列表 • 形状动画可绘制对象 • 其他可绘制对象 通过 R. drawable 类访问
mipmap	适用于不同启动器图标密度的可绘制对象文件，通过 R. mipmap 类访问
layout	用于定义用户界面布局的 XML 文件，通过 R. layout 类访问
menu	用于定义应用菜单的 XML 文件，通过 R. menu 类访问
raw	要以原始形式保存的任意文件，要使用原始 InputStream 打开这些资源，请使用资源 ID（即 R. raw. filename）调用 Resources.openRawResource（）
values	包含字符串、整型数和颜色等简单值的 XML 文件 • arrays.xml：用于资源数组（类型化数组），通过 R. array 类访问 • colors.xml：颜色值，通过 R. color 类访问 • dimens.xml：尺寸值，通过 R. dimen 类访问 • strings.xml：字符串值，通过 R. string 类访问 • styles.xml：样式，通过 R. style 类访问
xml	可以在运行时通过调用 Resources.getXML（）读取的任意 XML 文件

4.1.2 Assets 目录下的资源

除了 res 目录下的资源，assets 目录也可以存放资源。Assets 目录下的文件和 res/raw 目录下的文件一样，它们会被原封不动的拷贝到 APK 中，而不会被编译成二进制的形式。raw 目录下的所有资源文件都会在 R 文件中生成一个 id，可以直接使用 R 访



问，而 assets 目录下的资源文件不会在 R 文件中生成 id，只能通过 AssetManager 来访问此类文件。

4.2 Drawable Resources

4.2.1 使用资源

在前面的项目中，编程均使用硬编码。本节在学习资源管理之后，在项目中需要习惯将资源文件放入其对应的目录下，使用 Android 的资源管理来访问这些资源。

新建一个项目，取名为“ResApp”，新建 MainActivity，在“activity_main.xml”文件中添加如下代码：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:background="@color/black"
        android:drawablePadding="@dimen/length_10"
        android:drawableTop="@mipmap/ic_launcher"
        android:gravity="center"
        android:padding="@dimen/length_10"
        android:text="@string/button_text"
        android:textColor="@color/white"
        android:textSize="@dimen/text_size_18" />
</RelativeLayout>
```



这里使用相对布局，添加一个 Button 在布局中心位置。Button 控件设置一个背景色 (background)，引用颜色资源。textColor 属性指定 Button 中文字的颜色，同样引用一个颜色资源。颜色资源在 “res/values/colors.xml” 路径下的文件中，代码如下：

```
<resources>

    <color name="black">#000000</color>

    <color name="white">#FFFFFF</color>

</resources>
```

颜色资源使用 < color > 标签在 “colors.xml” 文件中声明，在 Java 代码中用 “R.color.name” 访问，在 XML 文件中用 “@color/name” 访问。

Button 中 drawableTop 属性指定一张图片处于 Button 中的文字的上方，此处引用图片 (mipmap) 资源，在 “res/mipmap” 目录下，图片格式可以采用 png、jpg 等，ic_launcher.png 是 Android Studio 新建 Android 项目时自带的加载图标，在实际开发中需要开发者自行替换。drawablePadding 属性指定图片与文字之间的间距，引用一个尺寸值。padding 属性是内间距，指定 Button 中的内容距离 Button 边框的长度。textSize 属性指定 Button 中文字的字体大小，同样引用一个尺寸值。尺寸值资源位于 “res/values/dimens.xml” 路径下的文件中，同理，尺寸值资源需要使用 < dimen > 标签声明，代码如下：

```
<resources>

    <dimen name="length_10">10dp</dimen>

    <dimen name="text_size_18">18sp</dimen>

</resources>
```

Button 控件的 text 属性使用了一个字符串资源，字符串资源位于 “res/values/strings.xml” 路径下的文件中，字符串资源用 < string > 标签声明，代码如下：

```
<resources>

    <string name="button_text">我是一个按钮</string>

</resources>
```

综上所述，MainActivity 的界面是一个位于屏幕中心的黑色的 Button，Button 上有一段 “我是一个按钮” 大小为 18sp 的白色的文字，文字上有一张 Android 默认的加载图片，图片和文字之间的距离是 10 dp，整个 Button 的内容与 Button 边框间距是 10 dp，运行 ResApp 项目，部署完成后，效果如图 4-1 所示，与预期结果相符。



图 4-1 矩形按钮

4.2.2 ShapeDrawable

Drawable 资源是 Android 中应用较广泛的资源，它可以直接使用 .png、.jpg、.gif 和 .9.png（使用九宫格切分的方法，使该类型图片在 android 环境下可自适应展示）等图片资源，还可以使用 XML 资源。ShapeDrawable 资源是 Android 开发中常用的 Drawable 资源。

在 ResApp 项目中，MainActivity 中添加一个黑色背景的矩形按钮，现在需将按钮变成圆形，代码如下：

```
< Button
    android:layout_width = " wrap_ content"
    android:layout_height = " wrap_ content"
    android:layout_centerInParent = " true"
    android: background = " @ drawable/button_ background_ black"
    android:drawablePadding = " @ dimen/length_ 10"
    android:drawableTop = " @ mipmap/ic_ launcher"
```



```
android: gravity = " center"  
android: padding = " @ dimen/length_ 30"  
android: text = " @ string/button_ text"  
android: textColor = " @ color/white"  
android: textSize = " @ dimen/text_ size_ 18" />
```

将 background 属性的值改成一个 Drawable 资源。在“res/drawable”目录下新建“button_ background_ black. xml”文件，添加代码如下：

```
<shape xmlns: android = " http: //schemas. android. com/apk/res/android"  
    android: shape = " oval" >  
    <solid android: color = " @ color/black" />  
</shape>
```

在 XML 文件中使用 < shape > 标签创建 ShapeDrawable 资源。shape 属性的值有 4 个，分别是 oval（椭圆形）、rectangle（矩形）、line（线形）和 ring（环形），shape 的值默认为 rectangle。运行程序，部署完成后的效果如图 4-2 所示，按钮形状变为圆形。



图 4-2 圆形按钮



定义 ShapeDrawable 的完整代码如下：

```
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" | "oval" | "line" | "ring" >
    <!-- 定义角度 -->
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <!-- 定义渐变色 -->
    <gradient
        android:angle="integer"
        android:centerX="float"
        android:centerY="float"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type="linear" | "radial" | "sweep"
        android:useLevel="true" | "false" />
    <!-- 定义内边距 -->
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <!-- 定义大小 -->
    <size
        android:width="integer"
```



```
        android: height = " integer" />
    <!-- 定义填充色 -->
    <solid
        android: color = " color" />
    <!-- 定义边框绘制 -->
    <stroke
        android: width = " integer"
        android: color = " color"
        android: dashWidth = " integer"
        android: dashGap = " integer" />
</shape>
```

4.2.3 StateListDrawable

StateListDrawable 是在 XML 中定义的可绘制对象，它可根据对象的状态，使用多个不同的图像来表示同一个图形。修改“activity_main.xml”文件中的代码如下：

```
<RelativeLayout xmlns: android = " http://schemas.android.com/apk/res/android"
    xmlns: tools = " http://schemas.android.com/tools"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    tools: context = " . MainActivity" >
    <Button
        android: id = " @ + id/activity_main_bt"
        android: layout_ width = " wrap_ content"
        android: layout_ height = " wrap_ content"
        android: layout_ centerInParent = " true"
        android: background = " @ drawable/selector_button"
        android: gravity = " center"
        android: paddingRight = " @ dimen/length_30"
        android: paddingLeft = " @ dimen/length_30"
        android: text = " @ string/button_text"
        android: textColor = " @ color/selector_color"
```



```
        android:textSize="@dimen/text_size_18" />
</RelativeLayout>
```

在主界面中添加一个矩形的黑色按钮，单击按钮，按钮变成红色，字体颜色也由白色变为黑色，松开按钮，按钮颜色恢复黑色，字体颜色恢复白色。Button 控件中 background 属性引用了一个新的 StateListDrawable 资源文件，在“res/drawable”目录下，新建“selector_button.xml”文件，添加代码如下：

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/rectangle_black" android:state_pressed="false" />
    <item android:drawable="@drawable/rectangle_red" android:state_pressed="true" />
</selector>
```

StateListDrawable 资源用 <selector> 标签声明，文件中两个 <item> 标签分别表示按钮不同状态下的背景。当单击按钮的时候，将背景设为“red”，“rectangle_red.xml”文件中的代码如下：

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="@dimen/length_15" />
    <solid android:color="@color/red" />
</shape>
```

当按钮未被单击时，将按钮背景设为“black”，“rectangle_black.xml”文件中的代码如下：

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <corners android:radius="@dimen/length_15" />
    <solid android:color="@color/black" />
</shape>
```

给按钮的文字颜色设置不同的单击状态有不同的颜色，在 textColor 属性中引用 selector_color 颜色资源，在 res 目录下新建 color 目录，然后在“res/color”目录下新建“selector_color.xml”文件，在其中添加代码如下：



```
<selector xmlns:android="http://schemas.android.com/apk/res/android">  
    <item android:color="@color/white" android:state_pressed="false"/>  
    <item android:color="@color/black" android:state_pressed="true"/>  
</selector>
```

运行 ResApp 项目，部署完成后，页面效果如图 4-3 所示，当前按钮的颜色为黑色，文字颜色为白色，单击按钮，按钮颜色变为红色，文字颜色变为黑色，松开按钮，按钮颜色恢复黑色，文字颜色恢复白色，与预期结果一致。



图 4-2-3 StateListDrawable

定义 StateListDrawable 的完整代码如下：

```
<selector xmlns:android="http://schemas.android.com/apk/res/android"  
    android:constantSize="true" | "false"  
    android:dither="true" | "false"  
    android:variablePadding="true" | "false">  
    <item  
        <!-- 图片资源 -->  
        android:drawable="@[package:]drawable/drawable_resource"
```



```
<!-- 是否按下 -->
    android:state_pressed = ["true" | "false"]
<!-- 是否获得焦点 -->
    android:state_focused = ["true" | "false"]
<!-- 是否指针（鼠标）位置 -->
    android:state_hovered = ["true" | "false"]
<!-- 是否选中 -->
    android:state_selected = ["true" | "false"]
<!-- 是否可勾选 -->
    android:state_checkable = ["true" | "false"]
<!-- 是否已勾选 -->
    android:state_checked = ["true" | "false"]
<!-- 是否可用 -->
    android:state_enabled = ["true" | "false"]
<!-- 是否激活 -->
    android:state_activated = ["true" | "false"]
<!-- 窗口是否获得焦点 -->
    android:state_window_focused = ["true" | "false"/>
</selector>
```

4.3 优化登录注册界面

4.3.1 优化登录界面

在第3章中新建的一个名为 ChongqingApp 的 Android 项目，在这个程序中已完成登录界面和注册界面，但采用的全为硬编码的方式，现在已学习资源文件的使用，可将界面优化，使代码更规范。

回到 ChongqingApp 程序中，将登录页面优化成如图 4-4 所示页面。将页面中输入框的提示语和输入文字颜色做出区分，为按钮添加更恰当的背景，并实现按钮的单击



效果。



图 4-4 登录界面优化

在登录页面的布局文件“activity_login.xml”中添加如下代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/page_default"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="@dimen/length_40"
        android:background="@color/colorPrimaryDark" >

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```



```
        android:layout_centerInParent="true"
        android:text="@string/login"
        android:textColor="@color/white"
        android:textSize="@dimen/text_size_20" />
</RelativeLayout>

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_weight="1"
    android:scaleType="center"
    android:src="@mipmap/logo" />

<EditText
    android:id="@+id/activity_login_et_username"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="@dimen/length_20"
    android:layout_marginRight="@dimen/length_20"
    android:layout_marginTop="@dimen/length_20"
    android:background="@drawable/rectangle_et_back"
    android:gravity="start"
    android:hint="@string/login_et_hint"
    android:inputType="text"
    android:paddingBottom="@dimen/length_10"
    android:paddingLeft="@dimen/length_20"
    android:paddingRight="@dimen/length_20"
    android:paddingTop="@dimen/length_10"
    android:textColor="@color/black"
    android:textColorHint="@color/et_hint" />

<EditText
    android:id="@+id/activity_login_et_password"
```



```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="@dimen/length_20"
android:layout_marginRight="@dimen/length_20"
android:layout_marginTop="@dimen/length_10"
android:background="@drawable/rectangle_et_back"
android:hint="@string/pwd_et_hint"
android:inputType="textPassword"
android:paddingBottom="@dimen/length_10"
android:paddingLeft="@dimen/length_20"
android:paddingRight="@dimen/length_20"
android:paddingTop="@dimen/length_10"
android:textColor="@color/black"
android:textColorHint="@color/et_hint" />
```

<RelativeLayout

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="@dimen/length_20"
android:layout_marginRight="@dimen/length_20"
android:layout_marginTop="@dimen/length_10" >
```

<TextView

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:paddingBottom="@dimen/length_5"
android:paddingRight="@dimen/length_15"
android:paddingTop="@dimen/length_5"
android:text="@string/pwd_forget" />
```

<TextView

```
android:id="@+id/activity_login_tv_register"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```




```

        android:layout_alignParentRight="true"
        android:paddingBottom="@dimen/length_5"
        android:paddingLeft="@dimen/length_15"
        android:paddingTop="@dimen/length_5"
        android:text="@string/register" />

</RelativeLayout>

<Button
    android:id="@+id/activity_login_bt_login"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="@dimen/length_100"
    android:layout_marginLeft="@dimen/length_20"
    android:layout_marginRight="@dimen/length_20"
    android:layout_marginTop="@dimen/length_20"
    android:background="@drawable/selector_bt"
    android:text="@string/login"
    android:textColor="@color/selector_bt_text_color"
    android:textSize="@dimen/text_size_20" />

</LinearLayout>

```

布局中使用的颜色资源在“res/values/colors.xml”路径文件中的代码如下：

```

<resources>

    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="page_default">#F4F4F4</color>
    <color name="white">#FFFFFF</color>
    <color name="black">#000000</color>
    <color name="et_hint">#999999</color>
    <color name="blue">#0000FF</color>
    <color name="blue_violet">#8A2BE2</color>

```



```
< color name = " red" > #FF0000 </color >
</resources >
```

“res/color/ selector_ bt_ text_ color. xml” 路径下的文件中的代码如下:

```
< selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    < item android: color = " @ color/white" android: state_ pressed = " false" / >
    < item android: color = " @ color/black" android: state_ pressed = " true" / >
</selector >
```

尺寸资源 “res/values/dimens. xml” 路径下的文件中代码如下:

```
< resources >
    < dimen name = " length_ 5" > 5dp </dimen >
    < dimen name = " length_ 10" > 10dp </dimen >
    < dimen name = " length_ 15" > 15dp </dimen >
    < dimen name = " length_ 20" > 20dp </dimen >
    < dimen name = " length_ 40" > 40dp </dimen >
    < dimen name = " length_ 50" > 50dp </dimen >
    < dimen name = " length_ 100" > 100dp </dimen >
    < dimen name = " text_ size_ 20" > 20sp </dimen >
</resources >
```

字符串资源 “res/values/strings. xml” 路径下的文件中代码如下:

```
< resources >
    < string name = " app_ name" > ChonqingApp </string >
    < string name = " login" > 登录 </string >
    < string name = " login_ et_ hint" > 请输入用户名 </string >
    < string name = " pwd_ et_ hint" > 请输入密码 </string >
    < string name = " pwd_ forget" > 忘记密码 </string >
    < string name = " register" > 注册 </string >
    < string name = " pwd_ certain_ et_ hint" > 请确认密码 </string >
    < string name = " toast_ login" > 用户名或密码不能为空 </string >
    < string name = " toast_ different_ pwd" > 两次输入的密码不同 </string >
</resources >
```



Drawable 资源 “res/drawable/ rectangle_ et_ back.xml” 路径下的文件中代码如下：

```
<shape xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: shape = " rectangle" >
    <solid android: color = " @ color/white" />
    <corners android: radius = " @ dimen/length_ 10" />
</shape>
```

“res/drawable/ rectangle_ bt_ no_ press.xml” 路径下的文件中代码如下：

```
<shape xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: shape = " rectangle" >
    <solid android: color = " @ color/blue_ violet" />
    <corners android: radius = " @ dimen/length_ 10" />
</shape>
```

“res/drabable/ rectangle_ bt_ press.xml” 路径下的文件中代码如下：

```
<shape xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: shape = " rectangle" >
    <solid android: color = " @ color/blue" />
    <corners android: radius = " @ dimen/length_ 10" />
</shape>
```

“res/drawable/ selector_ bt.xml” 路径下的文件中代码如下所示：

```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: state_ pressed = " true" android: drawable = " @ drawable/rectangle_
bt_ no_ press" />
    <item android: state_ pressed = " false" android: drawable = " @ drawable/rectangle_
bt_ press" />
</selector>
```

4.3.2 优化注册界面

如图 4-5 所示，优化注册页面。添加输入框的背景、输入文字颜色、提示语颜色以及增加按钮单击效果和背景颜色。



图 4-5 注册页面优化

注册页面布局“activity_register.xml”文件中的修改代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="@dimen/length_40"
        android:background="@color/colorPrimaryDark" >

        <ImageView
            android:id="@+id/activity_register_iv_back"
            android:layout_width="@dimen/length_40"
            android:layout_height="match_parent"
            android:paddingBottom="@dimen/length_5"
```



```
        android: paddingTop = " @ dimen/length_ 5"
        android: src = " @ mipmap/back" />

<TextView
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ centerInParent = " true"
    android: text = " @ string/register"
    android: textColor = " @ color/white"
    android: textSize = " @ dimen/text_ size_ 20" />

</RelativeLayout>

<EditText
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content"
    android: layout_ marginLeft = " @ dimen/length_ 20"
    android: layout_ marginRight = " @ dimen/length_ 20"
    android: layout_ marginTop = " @ dimen/length_ 50"
    android: background = " @ drawable/rectangle_ et_ back"
    android: hint = " @ string/login_ et_ hint"
    android: inputType = " text"
    android: paddingBottom = " @ dimen/length_ 10"
    android: paddingLeft = " @ dimen/length_ 20"
    android: paddingRight = " @ dimen/length_ 20"
    android: paddingTop = " @ dimen/length_ 10"
    android: textColor = " @ color/black"
    android: textColorHint = " @ color/et_ hint" />

<EditText
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content"
    android: layout_ marginLeft = " @ dimen/length_ 20"
    android: layout_ marginRight = " @ dimen/length_ 20"
```



```
android:layout_marginTop="@dimen/length_10"
android:background="@drawable/rectangle_et_back"
android:hint="@string/pwd_et_hint"
android:inputType="textPassword"
android:paddingBottom="@dimen/length_10"
android:paddingLeft="@dimen/length_20"
android:paddingRight="@dimen/length_20"
android:paddingTop="@dimen/length_10"
android:textColor="@color/black"
android:textColorHint="@color/et_hint" />
```

< EditText

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="@dimen/length_20"
android:layout_marginRight="@dimen/length_20"
android:layout_marginTop="@dimen/length_10"
android:background="@drawable/rectangle_et_back"
android:hint="@string/pwd_certain_et_hint"
android:inputType="textPassword"
android:paddingBottom="@dimen/length_10"
android:paddingLeft="@dimen/length_20"
android:paddingRight="@dimen/length_20"
android:paddingTop="@dimen/length_10"
android:textColor="@color/black"
android:textColorHint="@color/et_hint" />
```

< Button

```
android:id="@+id/activity_register_bt_register"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginLeft="@dimen/length_20"
android:layout_marginRight="@dimen/length_20"
```



```
        android:layout_marginTop = " @ dimen/length_ 20"
        android:background = " @ drawable/selector_ bt"
        android:text = " @ string/register"
        android:textColor = " @ color/selector_ bt_ text_ color"
        android:textSize = " @ dimen/text_ size_ 20" />
</LinearLayout>
```

布局中所需资源文件在上一节中已进行介绍，此处不再赘述。

4.3.3 优化逻辑跳转

修改 LoginActivity 中的代码如下：

```
public class LoginActivity extends Activity {

    //用户名 密码
    private EditText mEtName, mEtPwd;
    //注册
    private TextView mTvRegister;
    //登录
    private Button mBtLogin;

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

        mEtName = findViewById ( R.id.activity_ login_ et_ username );
        mEtPwd = findViewById ( R.id.activity_ login_ et_ password );
        mTvRegister = findViewById ( R.id.activity_ login_ tv_ register );
        mBtLogin = findViewById ( R.id.activity_ login_ bt_ login );

        mTvRegister.setOnClickListener ( clickListener );
        mBtLogin.setOnClickListener ( clickListener );
    }

    View.OnClickListener clickListener = new View.OnClickListener () {
```



```
@Override
public void onClick ( View v ) {
    switch ( v.getId () ) {
        //注册
        case R.id.activity_login_tv_register:
            startActivity ( new Intent ( LoginActivity.this,
RegisterActivity.class ) );
            break;
        //登录
        case R.id.activity_login_bt_login:
            login ();
            break;
        default:
            break;
    }
}

};

private void login () {

    String name = mEtName.getText () . toString () . trim ();
    String pwd = mEtPwd.getText () . toString () . trim ();

    if (TextUtils.isEmpty ( name ) || TextUtils.isEmpty ( pwd ) ) {
        Toast.makeText ( this, getText ( R.string.toast_login ), Toast.LENGTH_
SHORT) . show ();
    } else {
        startActivity ( new Intent ( LoginActivity.this, MainActivity.class ) );
        finish ();
    }
}
}
```

单击“登录”按钮，判断输入框中是否含有输入内容，如果没有，弹出 Toast 提示信



息“用户名或密码不能为空”，如果已输入内容，则跳转至主界面，并销毁 LoginActivity。

修改 RegisterActivity 中的代码如下：

```
public class RegisterActivity extends Activity implements View.OnClickListener {

    //用户名 密码 确认密码
    private EditText mName, mPwd, mPsd;
    //返回
    private ImageView mIvBack;
    //注册
    private Button mBtRegister;

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

        mName = findViewById ( R.id.activity_register_et_name );
        mPwd = findViewById ( R.id.activity_register_et_pwd );
        mPsd = findViewById ( R.id.activity_register_et_psd );
        mIvBack = findViewById ( R.id.activity_register_iv_back );
        mBtRegister = findViewById ( R.id.activity_register_bt_register );

        mIvBack.setOnClickListener ( this );
        mBtRegister.setOnClickListener ( this );
    }

    @Override
    public void onClick ( View v ) {
        switch ( v.getId () ) {
            //返回
            case R.id.activity_register_iv_back:
                finish ();
                break;
            //注册
            case R.id.activity_register_bt_register:
```



```
        register ();
        break;
    default:
        break;
    }
}

private void register () {
    String name = mName.getText (). toString (). trim ();
    String pwd = mPwd.getText (). toString (). trim ();
    String psd = mPsd.getText (). toString (). trim ();

    if ( TextUtils.isEmpty ( name ) || TextUtils.isEmpty ( pwd ) || TextU-
tils.isEmpty ( psd )) {
        Toast.makeText ( this, getText ( R.string.toast _ login ), Toast.LENGTH _
SHORT) . show ();
    } else if ( ! pwd.equals ( psd )) {
        Toast.makeText ( this, getText ( R.string.toast _ different _ pwd ),
Toast.LENGTH _ SHORT) . show ();
    } else {
        finish ();
    }
}
```

单击“注册”按钮，首先判断输入框中是否含有输入内容，如果没有输入内容，则弹出提示信息“用户名或密码不能为空”；如果已有输入内容，则进行判断两次输入的密码是否相同，若不同，则弹出提示信息“两次输入的密码不同”，若相同，则返回 LoginActivity，并销毁 RegisterActivity。

练习题

- (1) Android 项目中，图片资源推荐放置目录为_____。
- (2) Android 项目在所有资源编译完成后，会自动生成_____文件提供访问。



- (3) 对资源文件进行命名时，应注意哪些事项？
- (4) 资源目录下可以有相同名称，但格式不同的两个文件存在吗？
- (5) 使用 `StateListDrawable` 可以实现点击按钮时，背景颜色切换效果和文字大小变化效果吗？

第 5 章 ListView



学习任务

本章主要完成以下学习目标：

- (1) 了解控件 ListView。
- (2) 掌握 ArrayAdapter 适配器使用。
- (3) 掌握 SimpleAdapter 适配器使用。
- (4) 掌握自定义 BaseAdapter 适配器使用。
- (5) 掌握适配器的优化。



本章介绍

在手机 UI 界面显示大量数据时，在遵循 MVC 设计架构的基础上，使用适配器是最主流的做法。通过适配器，将大量数据遵循特定的格式显示在 UI 界面上，使其具有良好的用户体验。通过本章的学习，读者可以深入了解适配器的作用及内部实现机制，提高项目开发效率。



5.1 ListView 简介

5.1.1 常用属性

列表视图 (ListView) 是 Android 内部已定义的控件, 如 `< ImageView >`、`< TextView >` 等常用控件。在 ListView 上填充的数据在指定区域内可以上下滑动显示, 如手机通讯录列表、QQ 联系人列表等。在布局文件 layout 中一般常用的属性有:

- (1) `android: id`, 唯一标识此控件。
- (2) `android: layout_ width`, 控件占用屏幕的宽度值。
- (3) `android: layout_ height`, 控件占用屏幕的高度值。
- (4) `android: divider`, 列表各项间分割线颜色。
- (5) `android: dividerHeight`, 列表各项间分割线高度。
- (6) `android: scrollbars`, 是否显示右侧纵向滚动条。
- (7) `android: listSelector`, 列表 item 选中时背景颜色。
- (8) `android: cacheColorHint`, 指定列表绘制背景, 指定透明色可优化绘制过程。
- (9) `android: fastScrollEnabled`, 当有较多列表项时决定是否出现快速滚动的按钮。

5.1.2 布局控件实例声明

常用属性设置的布局控件实例声明如下:

```
< ListView
    android: id = " @ + id/list"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: cacheColorHint = " #00000000"
    android: divider = " @ null"
    android: fastScrollEnabled = " false"
    android: listSelector = " @ android: color/transparent"
    android: scrollbars = " none" />
```



在布局文件 layout 声明后，在需要与该布局文件关联的视图类（如 Activity、Fragment、Dialog 等）引入该布局文件后，即可通过 findViewById（）方法准确查找到该 ListView 对象。

5.2 适配器

5.2.1 ArrayAdapter

ListView 控件只是显示数据的容器，而显示数据的方式由适配器决定。ArrayAdapter 是 Android 系统内部已实现的适配器，适用于类似数组、集合等存放相同的单一类型的数据。实现原理是将数据集中的每条数据分发到每一条“模板”布局中，根据数据的个数创建“模板”布局的个数。

因此应首先确定列表项“模板”布局。值得一提的是，Android 系统已为用户实现多种针对 ArrayAdapter 的“模板”布局，例如：

- (1) android.R.layout.simple_list_item_1。
- (2) android.R.layout.simple_list_item_2。
- (3) android.R.layout.simple_expandable_list_item_1。
- (4) android.R.layout.simple_expandable_list_item_2。
- (5) android.R.layout.simple_list_item_activated_1。
- (6) android.R.layout.simple_list_item_activated_2。

上述列举的各种系统布局中，内部均至少含有 1 个 <TextView>，其 id 值为 android.R.id.text1。以此更加深入说明 ArrayAdapter 的作用，即将数据集中的数据分发到指定“模板”内部的某个 <TextView> 控件上进行显示。

ArrayAdapter 的构造函数如下：

```
• ArrayAdapter (Context context, int resource, int textViewResourceId, T [] objects)
• ArrayAdapter (Context context, int resource, int textViewResourceId, List<T> objects)
```

第一个参数为当前 ArrayAdapter 使用的应用上下文，第二个参数为模板布局文件的资源地址，第三个参数为模板布局文件中要显示数据的某个 <TextView> 资源 id 标识，第四



个参数为数据源（可以为 List 集合或者数组）。以系统模板文件 “android. R. layout. simple_list_item_1” 为例，声明适配器 ArrayAdapter，代码如下：

```
// 声明数据源
List<String> data = new ArrayList<> ();

// 初始化数据源
for (int i=0; i<100; i++) {
    data.add (" this is item " +i);
}

// 构造一个 ArrayAdapter
ArrayAdapter<String> adapter = new ArrayAdapter<> (this, android. R. layout. simple_list_item_1, android. R. id. text1, data);

// 为 ListView 控件绑定适配器
listview.setAdapter (adapter);
```

运行效果如图 5-1 所示。

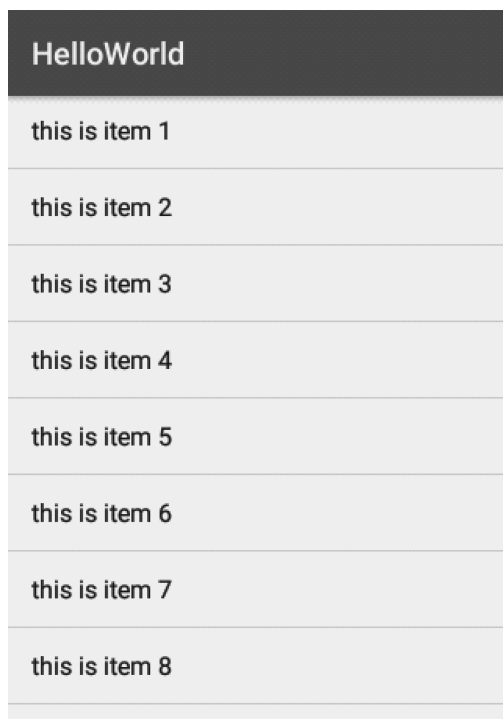


图 5-1 ArrayAdapter 效果图



当然，系统已定义的“模板”文件可能不符合用户项目的实际需要，此时用户需要重新定义所需要的“模板”文件。如果用户需要设置列表单项文本颜色为红色，且靠右显示一个复选框，则“模板”文件应做如下处理：

(1) 在 layout 布局资源文件夹中声明 xml 文件，名称设为“test.xml”。

(2) 在上述“test.xml”文件中写入控件内容等，代码如下：

```
<!-- 列表 item 高度自适应显示，横向子布局 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TextView
        android:id="@+id/text"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:padding="10dp"
        android:textColor="@color/colorAccent" />

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="match_parent" />
</LinearLayout>
```

(3) 构造 ArrayAdapter，使用自定义的“模板”布局，代码如下：

```
ListView listView = this.findViewById ( R.id.listview );

List<String> data = new ArrayList<> ();
for ( int i=0; i<100; i++ ) {
    data.add ( " this is item " +i );
}

ArrayAdapter<String> adapter = new ArrayAdapter<> ( this, R.layout.test,
R.id.text, data );

listView.setAdapter ( adapter );
```


运行程序，最终显示效果如图 5-2 所示，可满足用户需求。

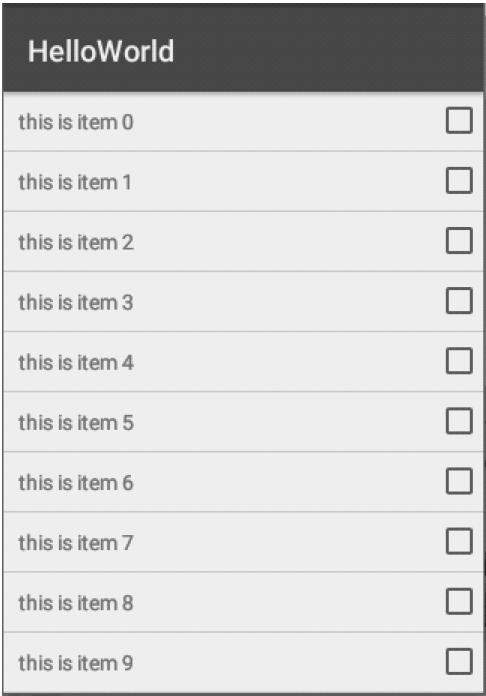


图 5-2 自定义列表单项布局

5.2.2 SimpleAdapter

通过 ArrayAdapter 的学习，读者可能会发现一个较重要的问题，那就是 ArrayAdapter 只能在 1 个“模板”文件中填充显示 1 个数据。那么如果“模板”文件较为复杂，仅仅填充显示 1 个数据显然无法满足实际需要。而 SimpleAdapter 的出现弥补了 ArrayAdapter 的不足。

此时数据源的格式为 List < Map > 结构，即数据源中的每一个 Map 数据分发到“模板”上的多个控件。Map 数据结构与多个控件需要进行顺序对应，将 Map 的 key 与控件的 id 进行顺序对应即可。

由于系统没有为用户预定义适用于 SimpleAdapter 的“模板”文件，因此需要用户自行实现。例如用户要实现列表单项有索引、文本、复选框 3 部分内容。此时处理逻辑顺序为：

（1）在 layout 布局资源文件夹下新建 XML 文件，命名为“test2.xml”，其中代码如下：



```
<LinearLayout
    xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: orientation = " horizontal"
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content" >

    <TextView
        android: id = " @ + id/index"
        android: layout_ width = " wrap_ content"
        android: layout_ height = " match_ parent"
        android: padding = " 10dp" />

    <TextView
        android: id = " @ + id/text"
        android: layout_ width = " 0dp"
        android: layout_ height = " match_ parent"
        android: layout_ weight = " 1"
        android: padding = " 10dp"
        android: textColor = " @ color/colorAccent" />

    <CheckBox
        android: id = " @ + id/checkbox"
        android: layout_ width = " wrap_ content"
        android: layout_ height = " match_ parent" />

</LinearLayout>
```

(2) 在视图文件（如 Activity、Fragment）中声明并初始化数据源 List < Map >，代码如下：

```
ListView listView = this. findViewById ( R. id. listview);
List < Map < String, Object > > data = new ArrayList < > ();
for ( int i = 0; i < 100; i + + ) {
    Map < String, Object > map = new HashMap < > ();
    map. put ( " index", i + 1);
    map. put ( " text", " This is item " + i);
```



```
map.put (" checked", i%4 == 0); //列表行为 4 的倍数时, 复选框  
勾选  
  
data.add (map);  
}
```

(3) 关联数据源中 Map 与“模板”文件中控件 id, 代码如下:

```
//map 数据结构的 key 都为字符串类型  
String [] from = {" index", " text", " checked"};  
  
//资源控件 id 都是 int 类型  
int [] to = {R.id.index, R.id.text, R.id.checkbox};
```

(4) 构造 SimpleAdapter, 并绑定适配器, 代码如下:

```
SimpleAdapter adapter = new SimpleAdapter (this, data, R.layout.test, from, to);  
listView.setAdapter (adapter);
```

完成后运行程序, 最终效果如图 5-3 所示。

7	This is item 6	☐
8	This is item 7	☐
9	This is item 8	☒
10	This is item 9	☐
11	This is item 10	☐
12	This is item 11	☐
13	This is item 12	☒
14	This is item 13	☐
15	This is item 14	☐
16	This is item 15	☐

图 5-3 SimpleAdapter 运行图



5.2.3 BaseAdapter

ArrayAdapter 和 SimpleAdapter 均是在 BaseAdapter 基础之上为用户封装功能，这两种适配器已经满足大多数用户的项目需求，但仍需要自定义 BaseAdapter 由于 ArrayAdapter 和 SimpleAdapter 虽然可以满足界面显示需要，但是对于深入的用户操作需求却难以满足，诸如列表项 item 的按钮点击事件处理，复选框选中状态事件处理等。此时需要自定义 BaseAdapter。

以实现 SimpleAdapter 相同的界面显示效果为例，自定义 BaseAdapter 的逻辑步骤如下：

(1) 首先定义“模板”文件，包括索引、文本和复选框 3 部分，可参考上节知识。

(2) 在项目源文件中，自定义类继承自 BaseAdapter，并重写基类的抽象方法，代码如下：

```
public class TestBaseAdapter extends BaseAdapter {

    public List < Map < String, Object > > data;
    public LayoutInflater inflater;

    //自定义构造函数
    public TestBaseAdapter ( Context context, List < Map < String, Object > > data ) {
        this. data = data;
        inflater = LayoutInflater. from ( context );
    }

    @ Override
    public int getCount () {
        return data == null? 0: data. size ();
    }

    @ Override
    public Object getItem ( int i ) {
        return data == null? null: data. get ( i );
    }

    @ Override
    public long getItemId ( int i ) {
```



```

        return i;
    }

    @Override
    public View getView (int i, View view, ViewGroup viewGroup) {
        View view = inflater.inflate (R.layout.test, viewGroup, false);
        Map <String, Object> map = data.get (i);
        TextView index = view.findViewById (R.id.index);
        TextView content = view.findViewById (R.id.text);
        CheckBox checkBox = view.findViewById (R.id.checkbox);
        index.setText (map.get ("index").toString ());
        content.setText (map.get ("text").toString ());
        checkBox.setChecked ((Boolean) map.get ("checked"));
        return view;
    }
}

```

(3) 在 Activity 或者 Fragment 中初始化数据集。注意此时 map 中的 key 应和自定义 BaseAdapter 中使用的 key 保持一致，代码如下：

```

ListView listView = this.findViewById (R.id.listview);
List <Map <String, Object>> data = new ArrayList <> ();
for (int i = 0; i < 100; i++) {
    Map <String, Object> map = new HashMap <> ();
    map.put ("index", i + 1);
    map.put ("text", "This is item " + i);
    map.put ("checked", i % 4 == 0);
    data.add (map);
}

```

(4) 构造自定义适配器，并使 ListView 绑定适配器，代码如下：

```

TestBaseAdapter adapter = new TestBaseAdapter (this, data);
listView.setAdapter (adapter);

```

运行程序，其实现效果如图 5-4 所示，可以发现结果与 SimpleAdapter 效果图一致。

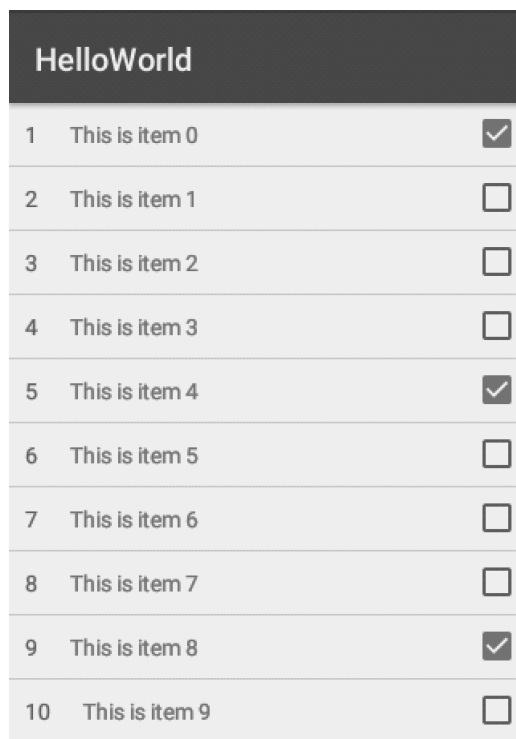


图 5-4 自定义 BaseAdapter

若要为列表单项的复选框添加选择事件时，可以在自定义 BaseAdapter 中加入如下代码：

```
checkBox.setOnCheckedChangeListener ( new  
CompoundButton. OnCheckedChangeListener () {  
    @ Override  
    public void onCheckedChanged ( CompoundButton compoundButton, boolean  
b) {  
        if ( b) {  
            Toast. makeText ( context, " 你选择了第" + ( i + 1) + " 个 item  
项", Toast. LENGTH_ SHORT) . show () ;  
        }  
    }  
});
```

运行程序，此时选择某一项列表中的复选框，效果如图 5-5 所示。

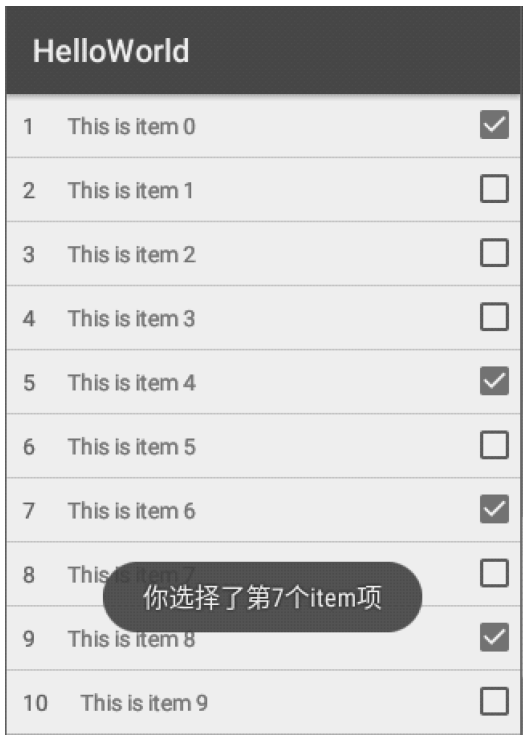


图 5-5 自定义 BaseAdapter 列表单项复选框选择事件

5.3 自定义适配器的优化

5.3.1 缓存优化实例

假设数据集中共有 40 个数据项，但由于手机界面的限制，若一次只能显示 7 个数据项，那么另外 33 个数据项为隐藏状态。而显示这 7 个数据项就需要 7 个 View 实例，即需要实例化 7 个 View 显示内容，此时系统含有 7 个 View 实例占据内存空间。若用户往下滑动屏幕，顶端的数据项会被隐藏，底端的数据项显示出来，此时需要 new 一个新的 View 实例。系统内存空间已包含 8 个 View 实例，可是只需要 7 个 View 实例就可以在手机屏幕完全显示，顶端被隐藏实例占用的内存空间是不需要的。随着手机屏幕向下滑动，创建更多 View 实例，与手机内存空间最多 7 个 View 实例的需求背道而驰。



利用缓存机制，在数据列表向下滑动时，最新进入屏幕的列表布局重复利用最新退出屏幕的列表 view 实例，保持内存中创建最初的 view 实例的最大显示个数，如图 5-6 所示。

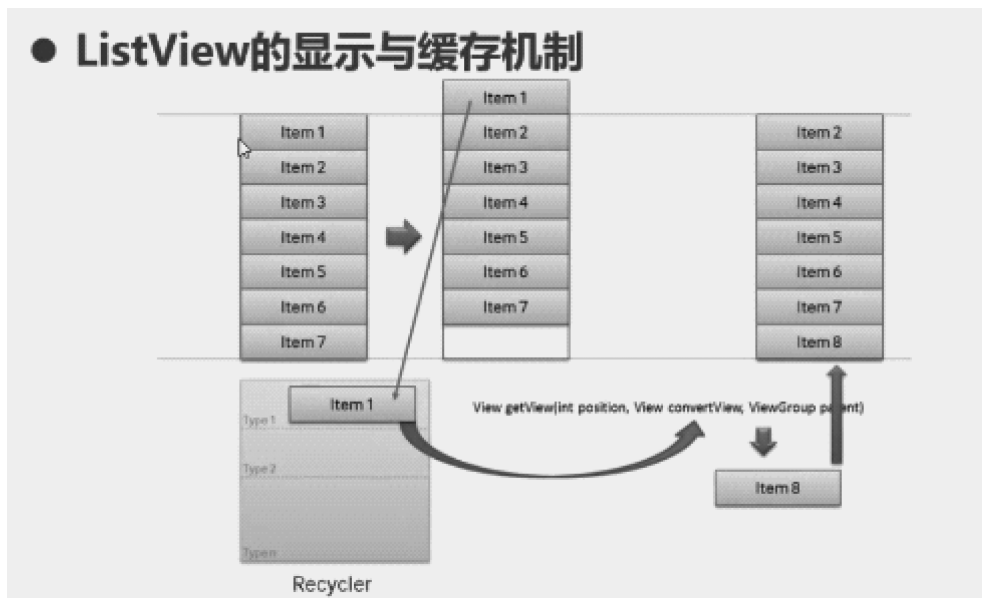


图 5-6 ListView 缓存优化

5.3.2 代码实现逻辑

此种优化方式代码实现逻辑如下：

(1) 在自定义 BaseAdapter 中声明内部类，内部类中成员属性为“模板”布局文件中要填充数据项的控件对象，代码如下：

```
public class ViewTag {  
    private TextView indexView;  
    private TextView contentView;  
    private CheckBox checkBox;  
}
```

(2) 在自定义 BaseAdapter 中 getView () 方法中实例化缓存，代码如下：



```
@Override
public View getView (final int i, View view, ViewGroup viewGroup) {

    ViewTag viewTag = null;
    if (view == null) {
        viewTag = new ViewTag ();
        viewTag.indexView = view.findViewById (R.id.index);
        viewTag.contentView = view.findViewById (R.id.text);
        viewTag.checkBox = view.findViewById (R.id.checkbox);
        view = inflater.inflate (R.layout.test, viewGroup, false);
        view.setTag (viewTag);
    } else {
        viewTag = (ViewTag) view.getTag ();
    }

    Map<String, Object> map = data.get (i);
    viewTag.indexView.setText (map.get ("index").toString ());
    viewTag.contentView.setText (map.get ("text").toString ());
    viewTag.checkBox.setChecked ((Boolean) map.get ("checked"));

    viewTag.checkBox.setOnCheckedChangeListener (new
CompoundButton.OnCheckedChangeListener () {

        @Override
        public void onCheckedChanged (CompoundButton compoundButton, boolean
b) {

            if (b) {
                Toast.makeText (context, "你选择了第" + (i + 1) + " 个 item
项", Toast.LENGTH_SHORT).show ();
            }
        }
    });

    return view;
}
```

(3) 声明并初始化自定义适配器，ListView 绑定适配器，如上节所述。



5.4 聊天机器人案例

5.4.1 聊天机器人基本效果

本章要求实现单机聊天功能。用户通过输入消息栏将文本发送后，将有聊天机器人随机发送消息进行单机聊天功能，基本效果如图 5-7 所示。



图 5-7 聊天机器人效果

5.4.2 单机聊天项目需求

接下来，按如下逻辑完成该项目需求：

(1) 新建 Android 项目 ChatProject，修改 “res/layout/activity_ main. xml” 路径下文件中的代码如下：

```
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
```



```
        android:orientation="vertical" >

<!-- 聊天内容所在的列表容器 -->
    <ListView
        android:id="@+id/lv"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:divider="@color/colorAccent"
        android:dividerHeight="1dp" />

<!-- 底部的横向线性布局，包括 1 个输入框和 1 个发送按钮 -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <EditText
            android:id="@+id/et"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:hint="请输入消息" />

        <Button
            android:id="@+id/send"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="点我发送" />

    </LinearLayout >
</LinearLayout >
```

该布局包含 1 个列表 ListView、1 个输入框和 1 个发送按钮。

(2) 新建“res/layout/chat_item.xml”路径文件作为列表单项 item 布局，其中代码如下：



```
<LinearLayout
    xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: orientation = " vertical"
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content" >

<!-- 左边机器人消息框 -->
<TextView
    android: id = " @ +id/left"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ gravity = " left"
    android: gravity = " center_ vertical"
    android: background = " @drawable/chatleft" />

<!-- 右边用户已发送消息显示框 -->
<TextView
    android: id = " @ +id/right"
    android: layout_ width = " wrap_ content"
    android: layout_ height = " wrap_ content"
    android: layout_ gravity = " right"
    android: gravity = " right | center_ vertical"
    android: background = " @drawable/chatright" />

</LinearLayout>
```

其包括左边机器人消息文本框和用户发送的文本消息框。此处使用 2 个 “.9” 格式的图片文件，以此保证背景中文字显示区域不被拉伸变形。

(3) 自定义 BaseAdapter 类，完成消息列表 view 实例的缓存及数据显示处理。当左侧消息内容为空时，则隐藏背景消息框。新建 “ChatBaseAdapter.java” 文件，其中代码如下：

```
public class ChatBaseAdapter extends BaseAdapter {
    private List<Map<String, String>> data; //数据集
    private Context context;
    private LayoutInflater inflater;
```



```
public ChatBaseAdapter (List<Map<String, String>> left, Context context) {
    this.data = left;
    this.context = context;
    this.inflater = LayoutInflater.from (context);
}

@Override
public int getCount () {
    return data == null? 0: data.size ();
}

@Override
public Object getItem (int i) {
    return data == null? null: data.get (i);
}

@Override
public long getItemId (int i) {
    return i;
}

@Override
public View getView (int i, View view, ViewGroup viewGroup) {
    ViewTag viewHolder = null;
//判断一下当前加载的 View 是否已加载
    if (view == null) {
//实例化 ViewHolder 对象
        viewHolder = new ViewTag ();
//设置 view 加载布局 (当前 Item 布局, 当前 Item 的父类布局)
        view = inflater.inflate (R.layout.chat_item, viewGroup, false);
//ViewHolder 对象和 Item 布局对象一一绑定
        viewHolder.leftTv = (TextView) view.findViewById (R.id.left);
        viewHolder.rightTv = (TextView) view.findViewById (R.id.right);
//设置 view 的 Tag (标签): 给 view 添加一组额外的数据
```



```
        view.setTag ( viewHolder );
    } else {
        viewHolder = ( ViewTag ) view.getTag ( ); // 获取 viewHolder
    }

    Map < String, String > itemData = data.get ( i );
    viewHolder.setText ( itemData );
    return view;
}

class ViewTag {
    private TextView leftTv;
    private TextView rightTv;

    void setText ( Map < String, String > data ) {
        String left = data.get ( " left" );
        if ( TextUtils.isEmpty ( left ) ) {
            leftTv.setVisibility ( View.GONE );
        } else {
            leftTv.setVisibility ( View.VISIBLE );
            leftTv.setText ( data.get ( " left" ) );
        }
        rightTv.setText ( data.get ( " right" ) );
    }
}
}
```

(4) 在 MainActivity 中绑定 ListView、EditText 和 Button 实例，监听发送按钮的单击监听事件，代码如下：

```
public class MainActivity extends AppCompatActivity {

    private ListView listView;
    private EditText sendText;
    private Button sendBtn;
    private List < Map < String, String > > data;
    private ChatBaseAdapter adapter;
```



```
@Override
protected void onCreate (Bundle savedInstanceState) {
    super.onCreate ( savedInstanceState );
    setContentView ( R.layout.activity_ main );
    listView = findViewById ( R.id.lv );
    sendText = findViewById ( R.id.et );
    sendBtn = findViewById ( R.id.send );

    //初始化适配器，并加载空的列表数据集
    data = new ArrayList < > ( );
    adapter = new ChatBaseAdapter ( data, this );
    listView.setAdapter ( adapter );

    //发送按钮的单击监听事件
    sendBtn.setOnClickListener ( new View.OnClickListener () {
        @Override
        public void onClick ( View view ) {
            String sendMsg = sendText.getText ( ) .toString ( );
            if (TextUtils.isEmpty ( sendMsg ) ) {
                Toast.makeText ( MainActivity.this, " 输入 不能 为空 ",
Toast.LENGTH_ SHORT ) .show ( );
                return;
            }
            Map <String, String > item = new HashMap <String, String > ( );

            //当用户输入消息内容中包含 empty 关键字时，则将左侧机器人消息赋值为空
            if ( sendMsg.contains ( " empty" ) ) {
                item.put ( " left", "" );
            } else {
                item.put ( " left", " 这是测试" );
            }
            item.put ( " right", sendMsg );
        }
    })
}
```



```
        data.add ( item );  
  
//数据源有更新之后，通知界面刷新  
        adapter.notifyDataSetChanged ();  
  
//当前让列表最后一项选中（可见）  
        listView.setSelection ( data.size ( ) );  
  
//清空输入框中内容  
        sendText.setText ( null );  
    }  
    }  
}
```

至此，已经完成整个案例设计。运行 ChatProject 程序，在输入框中输入文字，单击“发送”按钮，即可实现单机聊天功能。

练习题

- (1) 列表的显示需要 ListView、适配器和_____这 3 个元素。
- (2) ArrayAdapter 适配器的数据格式可以是 List < Map < String, T > > 吗？
- (3) 系统提供的 ListView 组件，支持横向滑动吗？
- (4) 无论使用何种适配器，都会创建屏幕可见数量的 item 布局实例吗？
- (5) 简述自定义适配器需要使用哪些方法？作用分别是什么？

第6章 探究 Fragment



学习任务

本章主要完成以下学习目标：

- (1) 了解 Fragment。
- (2) 掌握 Fragment 的使用。
- (3) 掌握 Fragment 与 Activity 的通信。
- (4) 掌握 Fragment 的生命周期。
- (5) 完成 Fragment 的滑动切换与导航的联动。



本章介绍

本章主要介绍 Fragment 的使用，Fragment 与 Activity 有较多关联之处，它们的使用方法和生命周期都有较多相似之处。通过本章的学习，希望读者能掌握 Fragment 的使用。Fragment 与 Activity 之间的通信在实际 Android 开发中是较常用的重要技能。最后，在本章的末尾，需要完成一个 Fragment 的使用实例。



6.1 Fragment 的基本知识

6.1.1 Fragment 介绍

Android 3.0 (API 11) 开始引入 Fragment, 以便为大屏幕移动设备 (主要是为了适配平板电脑) 提供更加动态、灵活的 UI 设计。Fragment 嵌入 Activity 用户界面, 可以将多个 Fragment 组合在同一个 Activity 中, 也可以在多个 Activity 中重复使用一个 Fragment。

Fragment 必须嵌入 Activity 使用, 可以视 Fragment 为 Activity 的模块化组成部分。Fragment 具有生命周期, 能接收自身的交互事件, 但也受到它嵌入的 Activity 的生命周期的影响。例如, 当 Activity 暂停时, 其中的所有片段将会暂停; 当 Activity 被销毁时, 所有片段也将被销毁。Activity 还可以在运行时动态添加或移除 Fragment。

6.1.2 Fragment 的基本用法

下面通过实际使用 Fragment, 让读者具有一个更直观的印象。首先新建一个 “FragmentApp” 项目, 打开该项目, 在 “Java 包” 根目录下新建 “fragment/FragmentA.java”, 在 FragmentA 中添加如下代码:

```
public class FragmentA extends Fragment {  
  
    @ Nullable  
    @ Override  
    public View onCreateView (@ NonNull LayoutInflater inflater, @ Nullable ViewGroup container, @ Nullable Bundle savedInstanceState) {  
        return inflater.inflate (R.layout.fragment_ a, container, false);  
    }  
}
```

类似于 Activity, 用户创建的所有 Fragment 都需继承 Android 的 Fragment 类或其子类, 再重写 onCreateView () 方法, return 一个 View 作为 FragmentA 的视图。使用 LayoutInflater 类的 inflate () 方法将一个布局文件转为 View, inflate () 方法传递三个参



数：第一个参数为需被转为 View 的 XML 布局文件资源 id，第二个参数为扩展布局父项的 ViewGroup，第三个参数指示是否在扩展期间将扩展布局附加至 ViewGroup（第二个参数）的布尔值。

在 FragmentA 中使用一个尚未创建的布局文件，现在在“res/layout”路径目录下创建“fragment_ a. xml”文件，再在“fragment_ a. xml”文件中添加代码如下：

```
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: orientation = " vertical" >
    <TextView
        android: layout_ width = " wrap_ content"
        android: layout_ height = " wrap_ content"
        android: text = " 我是 FragmentA" />
</LinearLayout>
```

类似于 Activity 的布局文件，在这里使用硬编码，是为了方便读者阅读，在此处使用硬编码，而在实际开发中不推荐此做法。

Fragment 必须嵌入 Activity 才可使用，因此在 MainActivity 的布局“activity_ main. xml”文件中添加如下所示代码：

```
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    xmlns: tools = " http: //schemas. android. com/tools"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: orientation = " vertical"
    tools: context = " . MainActivity" >
    <fragment
        android: id = " @ + id/activity_ main_ container"
        android: name = " com. ydy. fragmentapp. fragment. FragmentA"
        android: layout_ width = " match_ parent"
        android: layout_ height = " match_ parent" />
</LinearLayout>
```

在布局文件中使用 <fragment> 访问 FragmentA，以此把 FragmentA 嵌入 MainActivity 中。需要注意的是，此处必须加上 id 属性。以上便创建了一个 FragmentA，并把其嵌入



MainActivity，运行 FragmentApp 程序，部署完成后的效果如图 6-1 所示。

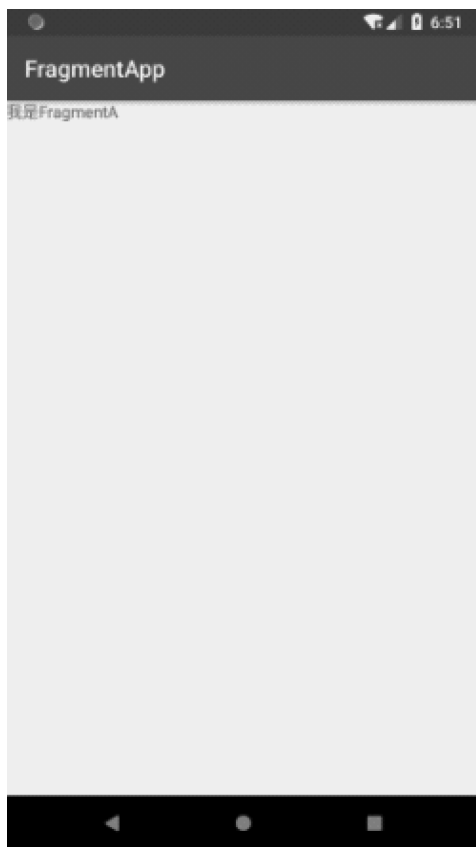


图 6-1 FragmentA

6.2 Fragment 的使用

6.2.1 动态添加 Fragment

在上一节中的 Activity 的布局文件中使用 `<fragment>` 标签来添加 Fragment，但在实际开发中较少使用，Android 中的控件均可动态添加到 Activity 中，Fragment 同样可使用 Java 代码动态添加到 Activity 中。

再新建一个 FragmentB，在 FragmentB 的布局 “fragment_b.xml” 文件中添加如下代码：



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorAccent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="我是 FragmentB"
        android:textColor="#FFFFFF" />
</LinearLayout>
```

为 FragmentB 添加一个红色的背景色。修改 “activity_main.xml” 文件中的代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >
    <fragment
        android:id="@+id/activity_main_container"
        android:name="com.ydy.fragmentapp.fragment.FragmentA"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />
    <FrameLayout
        android:id="@+id/activity_main_container_frame"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >
        <Button
            android:layout_width="wrap_content"
```



```
        android:layout_height="wrap_content"
        android:text=" 添加 FragmentB"
        android:onClick=" addFragmentB"
        android:layout_gravity="center"
        android:textAllCaps="false" />
    </FrameLayout>
</LinearLayout>
```

在 Activity 的布局中添加一个 FrameLayout 作为 FragmentB 的容器，由此 MainActivity 中包含两个 Fragment，上下平分 MainActivity 的屏幕。在 FrameLayout 中放置一个 Button 控件，使用 onClick 属性添加监听，在 MainActivity 中需要实现对应的 addFragmentB 方法，当单击按钮的时候，即可添加一个 FragmentB。

在 MainActivity 中添加如下代码：

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate ( savedInstanceState );
        setContentView ( R.layout.activity_main );
    }

    public void addFragmentB (View v) {

        FragmentManager manager = getSupportFragmentManager ();
        FragmentTransaction transaction = manager.beginTransaction ();

        FragmentB fragmentB = new FragmentB ();
        transaction.add ( R.id.activity_main_container_frame, fragmentB );
        transaction.commit ();
    }
}
```

在 MainActivity 的布局文件中绑定监听事件，实现 addFragmentB () 方法即可监听按钮的单击事件，需要注意的是 addFragmentB () 必须按照 OnClickListener 的回调方法格式编写，代码如下：



```
public void onClick (View view) {  
    //点击事件处理  
}
```

在 Activity 中动态添加 Fragment 需要用到 `FragmentManager` 类，获取它的实例。获取 `FragmentB` 的实例，使用 `FragmentManager` 的 `add()` 方法添加 `FragmentB` 至 `Activity` 中，`add()` 方法传递两个参数：第一个参数指定 `Activity` 中放置 `Fragment` 的容器；第二个参数就是需要添加的 `Fragment`。最后还需要使用 `FragmentManager` 的 `commit()` 方法提交事务，使代码生效。

运行 `FragmentApp` 程序，部署完成后，单击按钮，结果如图 6-2 所示，可以看出 `MainActivity` 的上半部分是 `FragmentA`，下半部分红色背景的是 `FragmentB`，其平分整个 `MainActivity`。

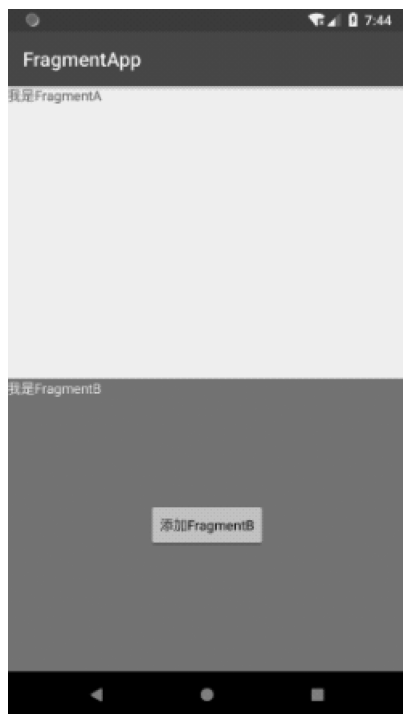


图 6-2 FragmentB

6.2.2 Fragment 与 Activity 之间的通信

`Fragment` 依赖于 `Activity` 使用，而 `Fragment` 之间相对独立。为了方便 `Fragment` 和 `Activity` 之间的通信，`Android` 提供一系列的方法可供使用。



在 Activity 中，可以通过 FragmentManager 类提供的 findFragmentById () 或者 findFragmentByTag () 方法获取 Fragment 的实例，具体代码如下：

```
FragmentA fragmentA = (FragmentA) getSupportFragmentManager () .  
    findFragmentById (R.id.activity_ main_ container);  
FragmentA fragmentA = (FragmentA) getSupportFragmentManager () .  
    findFragmentByTag (" fragment_ a");
```

得到 Fragment 的实例，即可轻松调用 Fragment 中的方法。在 Fragment 中可通过调用 getActivity () 方法获取 Activity 的实例，代码如下所示：

```
MainActivity mainActivity = (MainActivity) getActivity ();
```

获得 Activity 的实例后，开发者即可访问 Activity 中的方法。需要注意的是，一个 Fragment 可以在多个 Activity 中使用，但 Fragment 通过 getActivity () 方法获取的 Activity 实例只能是与其关联的当前 Activity 的实例。

除此之外，一些应用场景中可能需要通过 Fragment 和 Activity 共享事件，在这种情况下，可以在 Fragment 中定义一个回调接口，再在与其绑定的 Activity 中实现该接口。

6.3 Fragment 生命周期

6.3.1 回调方法

类似于 Activity，Fragment 也具有生命周期。图 6-3 为 Fragment 的生命周期示意，在 Fragment 的生命周期中，如下方法将会回调：

- (1) onAttach ()：当 Fragment 添加到 Activity 中时回调。
- (2) onCreate ()：创建 Fragment 时回调，只调用一次。
- (3) onCreateView ()：每次绘制该 Fragment 的 View 时都会回调该方法。
- (4) onActivityCreated ()：当与 Fragment 绑定的 Activity 启动完成后回调。
- (5) onStart ()：启动 Fragment 时回调。
- (6) onResume ()：类似于 Activity，在 Fragment 可见时回调。
- (7) onPause ()：暂停时回调该方法。
- (8) onStop ()：停止时回调。



(9) `onDestroyView()`: 销毁 Fragment 的 View 时回调。

(10) `onDestroy()`: 销毁 Fragment 时回调。

(11) `onDetach()`: 与 Activity 解绑时回调。

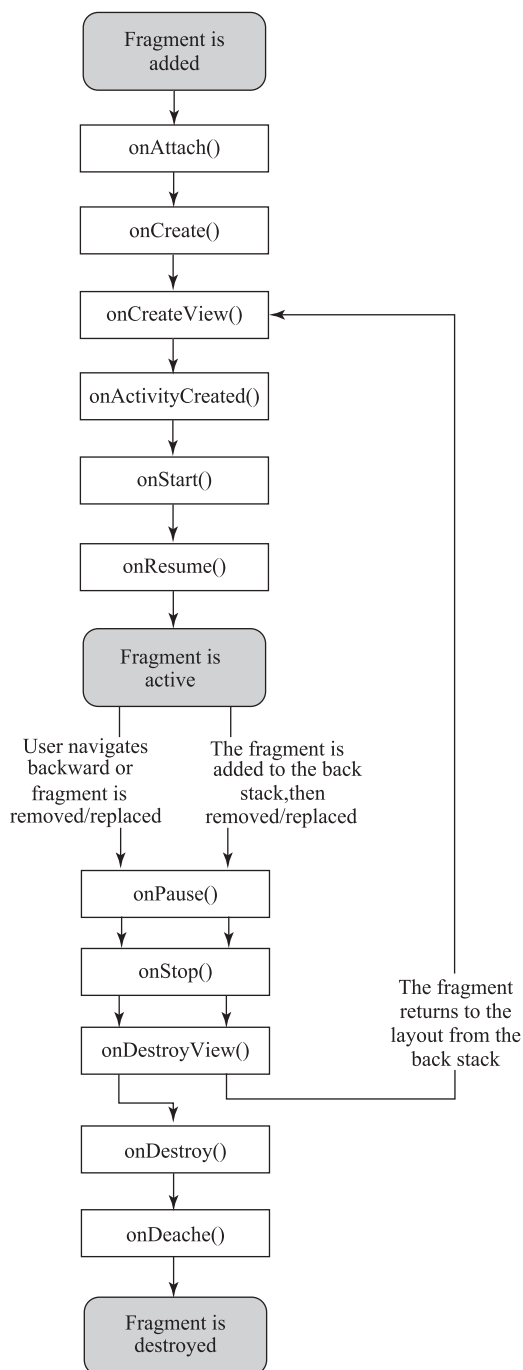


图 6-3 Fragment 的生命周期



6.3.2 Fragment 的状态

类似于 Activity，Fragment 具有如下几种状态：

- (1) 运行状态：Fragment 在 Activity 中可见，获得焦点可与用户进行交互。
- (2) 暂停状态：Fragment 仍然可见，但是已经失去焦点。
- (3) 停止状态：Fragment 已不可见，与其绑定的 Activity 停止时，或者 Fragment 被移除但已添加到 Back Stack 中。
- (4) 销毁状态：Activity 销毁时，或者 Fragment 被移除且没有被添加到 Back Stack 中。

6.4 仿微信主页界面切换

6.4.1 创建 Fragment

如图 6-4 所示，在本书中将完成仿微信主界面的布局及滑动切换类型的功能。具体功能可以参考微信，当单击界面下边的导航时，Fragment 会根据所选中的 Tab 不同而进行切换，并且当左右滑动上方的内容时，下方的导航也会跟随进行切换，实现 Fragment 和导航的联动。



图 6-4 布局



新建一个 Android 项目，取名为 “Wechat”，打开该项目，在 Java 的根目录下新建 “fragment” 包，方便管理。微信的主界面有 4 个 Tab 选项，分别是 “微信” “通讯录” “发现” 和 “我”，因此需先创建四个 Fragment。

在 “fragment” 包下新建 “ChatFragment” 作为微信 Tab，ChatFragment 中的代码如下：

```
public class ChatFragment extends Fragment {  
  
    @ Nullable  
    @ Override  
    public View onCreateView (@ NonNull LayoutInflater inflater, @ Nullable ViewGroup container, @ Nullable Bundle savedInstanceState) {  
        return inflater.inflate (R.layout.fragment_ chat, container, false);  
    }  
}
```

在 “res/layout” 目录下新建 ChatFragment 的布局 “fragment_ chat.xml” 文件，其中代码如下：

```
<RelativeLayout xmlns: android = " http: //schemas. android. com/apk/res/android"  
    android: layout_ width = " match_ parent"  
    android: layout_ height = " match_ parent" >  
    <TextView  
        android: layout_ width = " wrap_ content"  
        android: layout_ height = " wrap_ content"  
        android: layout_ centerInParent = " true"  
        android: text = " @ string/wechat"  
        android: textColor = " @ color/colorAccent"  
        android: textSize = " @ dimen/text_ size" />  
</RelativeLayout >
```

在 “fragment” 包下新建 ContactsFragment 作为通讯录 Tab，ContactsFragment 中的代码如下：

```
public class ContactsFragment extends Fragment {  
  
    @ Nullable  
    @ Override
```



```
public View onCreateView (@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
    return inflater.inflate (R.layout.fragment_contacts, container, false);  
}  
}
```

在“res/layout”目录下新建 ContactsFragment 的“fragment_contacts.xml”布局文件，其中代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:background="@color/green" >  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_centerInParent="true"  
        android:text="@string/contact"  
        android:textColor="@color/colorAccent"  
        android:textSize="@dimen/text_size" />  
</RelativeLayout>
```

在“fragment”包下新建 ChatFragment 作为“微信”Tab 选项卡对应的 Fragment，ChatFragment 中的代码如下：

```
public class FindFragment extends Fragment {  
    @Nullable  
    @Override  
    public View onCreateView (@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
        return inflater.inflate (R.layout.fragment_find, container, false);  
    }  
}
```

在“res/layout”目录下新建 ChatFragment 对应的 fragment_chat.xml 布局文件，其中代码如下：



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="@string/find"
        android:textColor="@color/colorAccent"
        android:textSize="@dimen/text_size" />
</RelativeLayout>
```

在“fragment”包下新建 MyFragment 作为“我的”Tab 选项卡对应的 Fragment，MyFragment 中的代码如下：

```
public class MyFragment extends Fragment {

    @Nullable
    @Override
    public View onCreateView (@NonNull LayoutInflater inflater, @Nullable ViewGroup
container, @Nullable Bundle savedInstanceState) {
        return inflater.inflate (R.layout.fragment_my, container, false);
    }
}
```

在“res/layout”目录下新建 MyFragment 对应的 fragment_my.xml 布局文件，其中代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/green" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
```



```
        android:text="@string/my"

        android:textColor="@color/colorAccent"

        android:textSize="@dimen/text_size" />
</RelativeLayout>
```

已创建完成的 4 个 Fragment，也可创建一个 Fragment 用于复用。Wechat 项目中所用资源将在后述内容中给出。

字符串资源在“res/values/strings.xml”路径下文件中，其中代码如下：

```
<resources>

    <string name="app_name">Wechat</string>

    <string name="wechat">微信</string>

    <string name="contact">通讯录</string>

    <string name="find">发现</string>

    <string name="my">我</string>

</resources>
```

尺寸资源在“res/values/dimens.xml”路径下文件中的代码如下：

```
<resources>

    <dimen name="length_5">5dp</dimen>

    <dimen name="length_60">60dp</dimen>

    <dimen name="text_size">30sp</dimen>

</resources>
```

颜色资源在“res/values/colors.xml”路径下文件中的代码如下：

```
<resources>

    <color name="colorPrimary">#3F51B5</color>

    <color name="colorPrimaryDark">#303F9F</color>

    <color name="colorAccent">#FF4081</color>

    <color name="page_default">#F0F0F0</color>

    <color name="grey">#999999</color>

    <color name="green">#46C01B</color>

</resources>
```

“res/color/selector_text_color_main.xml”路径下文件中的代码如下：



```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: color = " @ color/grey" android: state_ checked = " false" />
    <item android: color = " @ color/green" android: state_ checked = " true" />
</selector>
```

Drawable 资源在 “res/drawable/selector_ chat. xml” 路径文件中代码如下:

```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: drawable = " @ mipmap/weixin_ normal" android: state_ checked = "
false" />
    <item android: drawable = " @ mipmap/weixin_ pressed" android: state_ checked
= " true" />
</selector>
```

“res/drawable/ selector_ contacts. xml” 路径下文件中代码如下:

```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: drawable = " @ mipmap/contact_ list_ normal" android: state _
checked = " false" />
    <item android: drawable = " @ mipmap/contact_ list_ pressed" android: state _
checked = " true" />
</selector>
```

“res/drawable/ selector_ find. xml” 路径下文件中的代码如下:

```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: drawable = " @ mipmap/find_ normal" android: state_ checked = "
false" />
    <item android: drawable = " @ mipmap/find_ pressed" android: state_ checked = "
true" />
</selector>
```

“res/drawable/ selector_ my. xml” 路径下文件中代码如下:

```
<selector xmlns: android = " http: //schemas. android. com/apk/res/android" >
    <item android: drawable = " @ mipmap/profile_ normal" android: state_ checked = "
false" />
    <item android: drawable = " @ mipmap/profile_ pressed" android: state_ checked
= " true" />
</selector>
```



在程序中用到的图片资源请自行插入。

6.4.2 将 Fragment 嵌入 Activity

创建完成 Fragment，接着就将其嵌入 Activity。可以直接将创建好的 Fragment 嵌入 MainActivity。

MainActivity 的 “activity_ main.xml” 布局文件中代码如下：

```
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: orientation = " vertical" >

    <android. support. v4. view. ViewPager
        android: id = " @ + id/activity_ main_ vp"
        android: layout_ width = " match_ parent"
        android: layout_ height = " 0dp"
        android: layout_ weight = " 1" >

    </android. support. v4. view. ViewPager >

    <RadioGroup
        android: id = " @ + id/activity_ main_ rg"
        android: layout_ width = " match_ parent"
        android: layout_ height = " @ dimen/length_ 60"
        android: background = " @ color/page_ default"
        android: orientation = " horizontal" >

        <RadioButton
            android: id = " @ + id/activity_ main_ rb_ one"
            android: layout_ width = " 0dp"
            android: layout_ height = " match_ parent"
            android: layout_ weight = " 1"
            android: button = " @ null"
            android: checked = " true"
            android: drawablePadding = " @ dimen/length_ 5"
```




```
android:drawableTop="@drawable/selector_chat"
android:gravity="center"
android:padding="@dimen/length_5"
android:text="@string/wechat"
android:textColor="@color/selector_text_color_main" />
```

< RadioButton

```
android:id="@+id/activity_main_rb_two"
android:layout_width="0dp"
android:layout_height="match_parent"
android:layout_weight="1"
android:button="@null"
android:drawablePadding="@dimen/length_5"
android:drawableTop="@drawable/selector_contacts"
android:gravity="center"
android:padding="@dimen/length_5"
android:text="@string/contact"
android:textColor="@color/selector_text_color_main" />
```

< RadioButton

```
android:id="@+id/activity_main_rb_three"
android:layout_width="0dp"
android:layout_height="match_parent"
android:layout_weight="1"
android:button="@null"
android:drawablePadding="@dimen/length_5"
android:drawableTop="@drawable/selector_find"
android:gravity="center"
android:padding="@dimen/length_5"
android:text="@string/find"
android:textColor="@color/selector_text_color_main" />
```

< RadioButton

```
android:id="@+id/activity_main_rb_four"
```



```
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:button="@null"
        android:drawablePadding="@dimen/length_5"
        android:drawableTop="@drawable/selector_my"
        android:gravity="center"
        android:padding="@dimen/length_5"
        android:text="@string/my"
        android:textColor="@color/selector_text_color_main" />

</RadioGroup>

</LinearLayout>
```

在主界面布局中，用户使用 ViewPager 实现 Fragment 的滑动，用 RadioGroup 作为主界面下的 Tab 栏。

MainActivity 中的代码如下：

```
public class MainActivity extends AppCompatActivity {

    private ViewPager mVp;
    private RadioGroup mRg;

    private List<Fragment> list;
    private MyAdapter adapter;

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

        mVp = findViewById (R.id.activity_main_vp);
        mRg = findViewById (R.id.activity_main_rg);

        list = new ArrayList<> ();

        list.add (new ChatFragment ());
        list.add (new ContactsFragment ());
    }
}
```



```
list.add (new FindFragment ());  
list.add (new MyFragment ());  
  
adapter = new MyAdapter (getSupportFragmentManager ());  
mVp.setAdapter (adapter);  
  
mVp.addOnPageChangeListener (new ViewPager.OnPageChangeListener () {  
    @Override  
    public void onPageScrolled (int i, float v, int i1) {  
  
    }  
  
    @Override  
    public void onPageSelected (int page) {  
  
        switch (page) {  
            case 0:  
                mRg.check (R.id.activity_ main_ rb_ one);  
                break;  
  
            case 1:  
                mRg.check (R.id.activity_ main_ rb_ two);  
                break;  
  
            case 2:  
                mRg.check (R.id.activity_ main_ rb_ three);  
                break;  
  
            case 3:  
                mRg.check (R.id.activity_ main_ rb_ four);  
                break;  
  
            default:  
                break;  
        }  
    }  
}  
  
@Override  
public void onPageScrollStateChanged (int i) {  
  
}
```



```
    });

    mRg.setOnCheckedChangeListener ( new RadioGroup. OnCheckedChangeListener
() {

    @ Override

    public void onCheckedChanged ( RadioGroup radioGroup, int checkedId) {

        switch ( checkedId) {

            case R. id. activity_ main_ rb_ one:

                mVp. setCurrentItem ( 0 );

                break;

            case R. id. activity_ main_ rb_ two:

                mVp. setCurrentItem ( 1 );

                break;

            case R. id. activity_ main_ rb_ three:

                mVp. setCurrentItem ( 2 );

                break;

            case R. id. activity_ main_ rb_ four:

                mVp. setCurrentItem ( 3 );

                break;

            default:

                break;

        }

    }

});

}

class MyAdapter extends FragmentPagerAdapter {

    public MyAdapter ( FragmentManager fm) {

        super ( fm );

    }

    @ Override

    public Fragment getItem ( int position) {
```



```
        return list.get (position);  
    }  
  
    @Override  
    public int getCount () {  
        return list.size ();  
    }  
}
```

监听 ViewPager 和 RadioGroup 的事件, 当这些事件被触动的时候, 添加逻辑处理, 实现 Tab 栏和 Fragment 滑动的联动。MainActivity 中所使用的资源在上一节中已有介绍。

完成上面的任务后, 基本已经实现 Fragment 滑动切换联动导航的功能, 运行 Wechat 程序, 部署完成后的效果如图 6-5 所示, 当单击通讯录 Tab 时, 界面如图 6-6 所示。



图 6-5 微信界面



图 6-6 通讯录界面



练习题

- (1) Fragment 的使用，必须借助于 Activity 实现吗？
- (2) Fragment 视图上弹窗出现时，会调用生命周期的_____方法。
- (3) FragmentManager 对 Fragment 的管理是借助于_____实现的。
- (4) 创建 Fragment 时，系统默认实现的方法为_____。
- (5) Android 项目中使用到的 Fragment 类都必须在 AndroidManifest.xml 文件中注册吗？

第7章 网络编程



学习任务

本章主要完成以下学习目标：

- (1) 了解 HTTP 协议 GET 请求，POST 请求的特点。
- (2) 掌握 HTTPURLConnection 中 GET 请求，POST 请求实现。
- (3) 理解 Android 系统多线程交互机制。
- (4) 掌握 JSON 数据格式的传递与解析。



本章介绍

在手机日益智能化的时代，手机客户端需要显示加载各种各样的网络数据，此时就需要深入掌握 Android 网络编程知识。考虑到 Android 系统的多线程机制，子线程的处理优化对于网络数据的加载尤为重要。通过本章的学习，读者可以完成基本的网络数据获取与加载，深入理解数据流的处理操作。



7.1 访问网络

7.1.1 HTTP 协议请求

HTTP 定义了与服务器交互的不同方法，最基本的方法有 4 种，分别是 GET、POST、PUT、DELETE。可理解为：一个 URL 地址用于描述一个网络上的资源，而 HTTP 中的 GET、POST、PUT、DELETE 则对应着对这个资源的查、改、增、删 4 个操作。GET 一般用于获取/查询资源信息，而 POST 一般用于更新资源信息。

GET 请求的特点是在请求 URL 时，传入的参数会依附在 URL 地址后面，多个参数之间以 & 符号来进行连接，例如“http://url 地址? param1 = value1¶m2 = value2&...”。细心的读者可以发现，进行浏览器搜索时，URL 地址栏明显显示的是 GET 请求方式处理后结果。

POST 请求的特点是在请求 URL 时，传入的参数体会以数据流的格式请求，意味着在 URL 地址后面无法看到请求参数内容。例如常见的网页账户登录注册等模块，具体的输入信息 URL 地址无法体现。使用此种方式更加安全，保护用户隐私，也是目前主流推荐的请求方式。

7.1.2 网络编程实现

Android 系统内部已封装实现两个网络请求框架类，即 HttpClient 和 HttpURLConnection。其中 HttpClient 已在 Android6.0（即 API 版本为 23）之后被官方移除，旨在引导用户使用 HttpURLConnection，因此本书以 HttpURLConnection 为例讲解。

对于 GET 请求，一般的操作代码如下：

```
try {  
    //设置服务器项目连接地址  
    URL url =
```




```
new URL ( " http: //localhost: 8080/xxxxxxx? param1 = value1&param2 = value2" );  
  
//连接服务器  
URLConnection connection = url.openConnection ( );  
  
//强转为 HTTP 类型的 URLConnection  
HttpURLConnection httpURLConnection = ( HttpURLConnection ) connection;  
  
//设置请求方式  
httpURLConnection.setRequestMethod ( " GET" );  
  
//验证是否请求成功。200 代表成功，404 代表服务器项目资源找不到  
if ( 200 == httpURLConnection.getResponseCode ( ) ) {  
  
    //服务器返回 IO 数据流  
    InputStream inputStream = httpURLConnection.getInputStream ( );  
  
    //将服务器返回的 IO 流数据解析存放到字节输出流  
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream ( );  
  
        //循环读取 IO 流数据，每次读取 1 kb 数据  
        int length;  
    byte [ ] array = new byte [ 1024 ];  
        while ( ( length = inputStream.read ( array ) ) != -1 ) {  
        outputStream.write ( array, 0, length );  
        outputStream.flush ( );  
    }  
  
    //将拷贝数据后的输出流 outputStream 转换为需要的输出格式，如字符串、文件、图  
    片等  
    outputStream.close ( );  
    inputStream.close ( );  
    }  
    } catch ( IOException e ) {  
    e.printStackTrace ( );  
    }
```



对于 POST 请求，一般的操作代码如下：

```
try {  
    //设置服务器项目连接地址  
    URL url = new URL (" http: //localhost: 8080/xxxxxxx" );  
  
    //连接服务器  
    URLConnection connection = url.openConnection ();  
  
    //强转为 HTTP 类型的 URLConnection  
    HttpURLConnection httpURLConnection = (HttpURLConnection) connection;  
  
    //设置请求方式  
    httpURLConnection.setRequestMethod (" POST" );  
  
    //设置接收请求体及输出响应体  
    httpURLConnection.setDoOutput ( true );  
    httpURLConnection.setDoInput ( true );  
  
    //传入请求体  
    OutputStream outwritestream = conn.getOutputStream ();  
    outwritestream.write ( json.getBytes () );  
    outwritestream.flush ();  
    outwritestream.close (); //一定要 close  
  
    //验证是否请求成功。200 代表成功，404 代表服务器项目资源找不到  
    if ( 200 == httpURLConnection.getResponseCode () ) {  
  
        //服务器返回 IO 数据流  
        InputStream inputStream = httpURLConnection.getInputStream ();  
  
        //将服务器返回的 IO 流数据解析存放到字节输出流，最后转化为需要的字符串即可  
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream ();  
  
        //循环读取 IO 流数据，每次读取 1 kb 数据内容  
        int length;  
        byte [] array = new byte [ 1024 ];  
        while ( ( length = inputStream.read ( array ) ) != -1 ) {  
            outputStream.write ( array, 0, length );  
        }  
    }  
}
```



```
        outputStream.flush ();
    }

//这里我们拿到了已拷贝数据后的输出流 outputStream，转换为需要的输出格式，//如
//字符串、文件、图片等
outputStream.close ();
inputStream.close ();
}

} catch (IOException e) {
e.printStackTrace ();
}
```

由于网络编程需要对网络进行访问，因此 Android 项目需要在 AndroidManifest.xml 文件中配置网络权限，代码如下：

```
<manifest ... >

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

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

通过上述 GET 与 POST 请求编码方式的对比，读者会发现两种请求方式在代码层面基本一致，主要区别在于对请求参数的处理，表现在以下两点：

- (1) 请求 URL 地址时，GET 请求会把所需参数拼接起来。
- (2) 在与服务器进行连接通信前，POST 请求会在请求数据流中加入请求参数体。

7.1.3 多线程与 Handler

在一个 Android 程序开始运行时，将会单独启动一个进程。默认的情况下，所有这个程序中的组件（如 Activity、Service、BroadcastReceiver、ContentProvider）都会运行在这个进程中。一个 Android 程序默认情况下也只有一个进程，但一个进程下却可以有多个线程。在多个线程当中，有一个被称为 UI 线程，其他均为子线程。UI 线程在 Android 程序运行时已被创建，作为进程当中的主线程，主要负责控制 UI 界面的显示、更新和控件交互。而子线程则是负责后台的处理，如下载文件，上传文件，获取网络数据等。在 Android 程序创建之初，一个进程呈现的是单线程模型，所有的任务均在一个线程中运行。因此，UI Thread 所执行的每一个函数，所花费的时间均应该是越短越优。



而其他比较费时的的工作（如访问网络，下载数据，查询数据库等）均应该交由子线程去执行，以免阻塞主线程。同时应注意的是，子线程不能与主线程直接进行交互。

考虑到上述网络请求获取数据会受到当前网络状态、数据流读写速度等影响而造成耗时过长的的问题，因此应把网络请求获取数据的行为放在子线程中完成。最简单的做法为开启一个子线程即可，代码如下：

```
new Thread ( new Runnable () {  
    @ Override  
    public void run () {  
//网络请求操作  
    }  
} ) . start () ;
```

至此已有 1 个 UI 主线程和 1 个网络请求子线程，若直接在子线程中执行 UI 操作逻辑（如弹出 Dialog 窗口，弹出吐司，修改控件 TextView 等显示内容），可以在上述已拿到拷贝数据后的输出流后，进行如下 Toast 操作测试：

```
String result = new String ( outputStream. toByteArray () , " utf-8" );  
Toast. makeText ( MainActivity. this ,  
    " 服务器返回的数据为：" + result , Toast. LENGTH_ SHORT ) . show () ;
```

可以发现，程序已崩溃，爆出的错误问题如图 7-1-1 所示。

```
08-06 07:44:42.299 5751-5778/com.example.liupengzhan.yundemo E/AndroidRuntime: FATAL EXCEPTION: Thread-4  
Process: com.example.liupengzhan.yundemo, PID: 5751  
java.lang.RuntimeException: Can't toast on a thread that has not called Looper.prepare()  
    at android.widget.Toast$TN.<init>(Toast.java:390)  
    at android.widget.Toast.<init>(Toast.java:114)  
    at android.widget.Toast.makeText(Toast.java:277)  
    at android.widget.Toast.makeText(Toast.java:267)  
    at com.example.liupengzhan.yundemo.HomeFragment.doGetNoParameter(HomeFragment.java:251)  
    at com.example.liupengzhan.yundemo.HomeFragment.access$500(HomeFragment.java:44)  
    at com.example.liupengzhan.yundemo.HomeFragment$4$1.run(HomeFragment.java:157)  
    at java.lang.Thread.run(Thread.java:764)
```

图 7-1-1 子线程 UI 操作问题

爆出的问题是“Can't toast on a thread that has not called Looper.prepare()?”，这里提到的 Looper 是指子线程与 UI 主线程交互的消息循环机制中的关键对象。子线程与主线程之间交互机制如图 7-1-2 所示，一共包含如下三个过程：

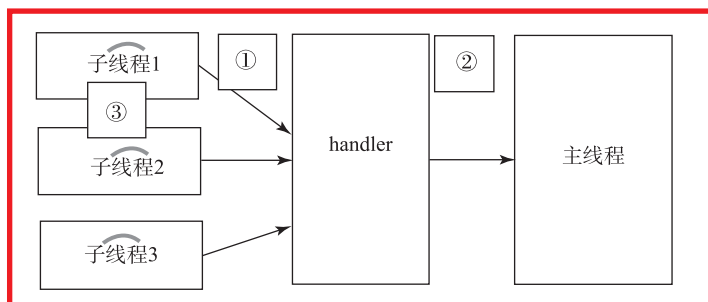


图 7-1-2 子线程与主线程交互机制

(1) 过程代表子线程执行任务完成后，将执行完成的信息数据通知 handler。消息封装用 message 实现，包括以下两个部分：

①what：消息标识符。

②obj：消息具体内容。

声明 1 个 message 并发送给 handler 的代码如下：

```
Message message = handler.obtainMessage ();
message.what = 1;
message.obj = result;
handler.sendMessage (message);
```

(2) 过程代表 handler 通知主线程做出处理。如显示数据、弹出框等。声明 1 个 handler 并做出业务处理的代码如下：

```
//定义 1 个 handler
private Handler handler = new Handler () {
//handler 通知主线程更新 UI (textview 内容更新)
    @Override
    public void handleMessage (Message msg) {
        super.handleMessage (msg);
        switch (msg.what) {
            case 1:
//当 message 中 what 为 1 时的业务逻辑
                break;
            case 2:
//当 message 中 what 为 2 时的业务逻辑
```



```
        break;
//此处省略其他 message 中 what 值情况
        default:
//当 message 中 what 上述都不满足时的业务逻辑
        break;
    }
}
};
```

(3) 过程 `Looper` 运行在子线程里，循环检测是否有新任务执行成功。如果成功之后，加入消息队列里面，后续会推送给 `handler`。

综上所述，`handler` 的主要作用有以下两点：

- (1) 在子线程中封装数据到 `message` 并发送。
- (2) 在主线程中接收 `message` 数据并解析处理数据。

`handler` 的创建和使用的一般过程如下：

- (1) 在主线程创建 `handler` 对象，重写 `handlerMessage()` 方法，用于随时接收数据。
- (2) 在子线程调用主线程创建的 `handler` 对象，为主线程发送信息。
- (3) 主线程接收到子线程发送的信息并进行处理，且可以直接显示在 UI 界面上。

7.1.4 AsyncTask 使用

在 `handler` 异步实现时，涉及 `handler`、`looper`、`message`、`thread` 四个对象，实现异步的流程是主线程启动 `thread`（子线程），`Thread`（子线程）运行并生成 `message`，`looper` 获取 `message` 并传递给 `handler`。`handler` 逐个获取 `looper` 中的 `message`，并进行 UI 变更。在单个后台异步处理时，会显得代码过多，结构过于复杂。Android 系统提供 `AsyncTask` 异步抽象类，用于弥补 `handler` 机制的不足之处。

`AsyncTask` 是 Android 提供的轻量级的异步类，可直接继承 `AsyncTask` 在类中实现异步操作，它本身已是 `handler` 的封装实现。`AsyncTask` 定义 3 种泛型类型：`params`、`progress` 和 `result`。

- (1) `params`：表示启动任务执行的输入参数，如 HTTP 请求的 URL。
- (2) `progress`：表示后台任务执行的百分比。
- (3) `result`：后台执行任务最终返回的结果，如 `String`。



继承 AsyncTask 类至少需要实现 2 个抽象方法：

(1) doInBackground (params...) 后台执行，比较耗时的操作均可以放在此处。注意此处不可直接进行操作 UI。此方法在后台线程执行，完成任务的主要工作，通常需要较长的时间。

(2) onPostExecute (result) 相当于 handler 处理 UI 的方式，在其内可使用在 doInBackground 处得到的结果处理操作 UI。此方法在主线程执行，任务执行的结果作为此方法的参数传入。

用 AsyncTask 类来实现网络请求的操作步骤如下：

(1) 自定义类继承自 AsyncTask，并实现其抽象方法，代码如下：

```
public class GetAsyncTask extends AsyncTask <String, Void, String> {

    @Override
    protected String doInBackground (String... strings) {
        String result = null;
        try {
            //设置服务器项目连接地址
            URL url = new URL (strings [0]);
            //连接服务器
            URLConnection connection = url.openConnection ();
            //强转为 HTTP 类型的 URLConnection
            HttpURLConnection httpURLConnection = (HttpURLConnection) connection;
            //设置请求方式
            httpURLConnection.setRequestMethod (" GET" );
            //验证是否请求成功。200 代表成功，404 代表服务器项目资源找不到
            if (200 == httpURLConnection.getResponseCode ()) {
                //服务器返回 IO 数据流
                InputStream inputStream = httpURLConnection.getInputStream ();
                //将服务器返回的 IO 流数据解析存放到字节输出流，最后转化为需要的字符串即可
                ByteArrayOutputStream outputStream = new ByteArrayOutputStream ();
```



```
//循环读取 IO 流数据，每次读取 1 kb 数据
        int length;
        byte [] array = new byte [1024];
        while ( (length = inputStream.read (array))! = -1) {
            outputStream.write (array, 0, length);
            outputStream.flush ();
        }
        result = new String (outputStream.toByteArray (), " utf-8");
        outputStream.close ();
        inputStream.close ();
    }
    } catch (IOException e) {
        e.printStackTrace ();
    }
    return result;
}

@Override
protected void onPostExecute (String s) {
    Toast.makeText ( getActivity () , " 获取结果为:" + result,
Toast.LENGTH_ SHORT) . show ();
}
}
```

(2) 在主线程中实例化该自定义类，并开启任务，代码如下：

```
new GetAsyncTask () . execute (" http: //192.168.1.172: 8080/basic/get/test");
```

综上所述，AsyncTask 网络编程使得业务逻辑更加简单，用户不再担心子线程与主线程由于交互问题而产生的各种问题，极大地提高开发效率。正是由于其封装的简单性，使得用户不会深入探究网络编程的内部实现机制，但对于宏观系统的学习反而是不利的。因此希望读者在理解 handler 机制的基础上再去学习使用 AsyncTask。



7.2 JSON 数据处理

JSON 是一种轻量级的数据交换格式，由于解析 XML 比较复杂，且需要编写大段的代码，因此客户端和服务器的数据交换格式往往通过 JSON 进行交换。

7.2.1 JSON 对象

对象表示为键值对，数据由逗号分隔，花括号保存对象。使用双引号包裹键名，使用冒号分隔键和值。如果值是 String 类型且含有双引号或冒号，需要使用“\”转义，如：

```
{ " name": " jadyli", " gender": " male" ... }
```

7.2.2 JSON 数组

由多个 JSON 对象组合而成，使用中括号外裹，格式如下：

```
[  
    { " name": " jadyli", " gender": " male" },  
    { " name": " sanqi", " gender": " male" }  
    ...  
]
```

针对上述两种数据格式，Android 系统内部已封装实现对应的数据类，即 JSONObject 和 JSONArray。

JSONObject 的数据封装和解析如下：

(1) 创建 JSON 数据，代码如下：

```
JSONObject jsonObject = new JSONObject ();  
jsonObject.put ( " name", " jadyli" );  
jsonObject.put ( " gender", " male" );  
jsonObject.put ( " age", 18 );  
System.out.println ( jsonObject.toString ( ));
```

输出结果如下：

```
{ "name": " jadyli", " gender": " male", " age": 18 }
```

(2) 解析 JSON 数据（从服务器返回的数据），代码如下：



//假如 json 为服务器返回的 JSON 字符串

```
JSONObject jsonObject = new JSONObject (json);  
System.out.println (" 姓名:" + jsonObject.getString (" name"));  
System.out.println (" 性别:" + jsonObject.getString (" gender"));  
System.out.println (" 年龄:" + jsonObject.getInt (" age"));
```

而对于 JSONArray，其数据封装和解析如下。

(1) 创建 JSON 数据，代码如下：

```
String [] names = {" jadyli", " Juliet"};  
String [] genders = {" male", " female"};  
int [] ages = {18, 20};  
JSONArray students = new JSONArray ();  
for (int i = 0; i < names.length; i++) {  
    JSONObject student = new JSONObject ();  
    student.put (" name", names [i]);  
    student.put (" gender", genders [i]);  
    student.put (" age", ages [i]);  
    students.put (student);  
}  
System.out.println (students.toString ());
```

输出结果如下：

```
[  
  { "name": " jadyli", " gender": " male", " age": 18 },  
  { "name": " Juliet", " gender": " female", " age": 20 }  
]
```

(2) 解析 JSON 数据（从服务器返回的 JSON 数据），代码如下：

```
//假如 json 为服务器返回的 JSON 字符串  
JSONArray students = new JSONArray (json);  
for (int i = 0; i < students.length (); i++) {  
    JSONObject student = students.getJSONObject (i);  
    System.out.println (" 姓名:" + student.getString (" name"));  
    System.out.println (" 性别:" + student.getString (" gender"));  
    System.out.println (" 年龄:" + student.getInt (" age") + " \n");  
}
```



练习题

- (1) Android 程序运行的主线程也称为_____线程。
- (2) 网络请求在主线程中执行会出现_____异常。
- (3) 使用 AsyncTask 时，必须实现的两个方法为_____和_____。
- (4) 在子线程中输出日志，会产生错误吗？
- (5) 错误的使用 Handler 为什么会存在内存泄露的现象？

第 8 章 数据存储

学习任务

本章主要完成以下学习目标：

- (1) 了解数据存储。
- (2) 掌握文件存储。

本章介绍

本章主要介绍 Android 下的数据存储技术，并通过实践操作掌握常见的数据存储技术。

8.1 数据存储介绍

8.1.1 数据存储概念

数据存储是指将内存中的瞬时数据保存在存储设备中，保证在手机或电脑关机的情况下，这些数据依然不会丢失。保存在内存中的数据处于瞬时状态，而保存在存储设备中的数据处于持久状态，数据存储则提供一种机制可以让数据在瞬时状态和持久状态之间进行转换。



8.1.2 实现数据存储的方式

Android 系统中主要提供 3 种方式用于简单实现数据存储功能，即文件存储、SharedPreferences 存储以及数据库存储。还可以把数据保存在手机 SD 卡中，但使用文件、SharedPreferences 或数据库来保存数据会相对更简单、更安全。

8.2 文件存储

8.2.1 将数据存储到文件中

文件存储是 Android 中一种基本的数据存储方式，它不对存储的内容进行格式化处理，所有的数据均原封不动保存在文件中，因此它适合用于存储一些简单的文本数据。若需其保存一些复杂的文件数据，就需要定义一套格式，方便以后将数据解析出来。

在 Context 类中的 `openFileOutput()` 方法，可以将数据存储到指定的文件中。该方法需要传入两个参数：第一个参数是文件名，在文件创建时使用的名称，需要注意的是这里指定的文件名不能包含路径，因为使用此方法保存的文件都是默认存储到“`data/data/<packagename>/files`”目录下。第二个参数是文件的操作模式，主要含有两种模式：`MODE_PRIVATE` 和 `MODE_APPEND`。`MODE_PRIVATE` 是默认的操作模式，表示当指定相同的文件名时，所写入的内容将会直接覆盖文件之前的内容；`MODE_APPEND` 模式则是在原文件内容中追加新的内容，若指定的文件不存在，两种模式均会创建新文件。

`OpenFileOutput()` 方法返回的是 `FileOutputStream` 对象，得到此对象后即可通过 Java 流的形式将数据写入文件中。以下是一段示例代码：

```
public void saveData (String data) {  
    FileOutputStream fos;  
    BufferedWriter bw = null;  
    try {  
        fos = openFileOutput (" data.txt", MODE_PRIVATE);  
        bw = new BufferedWriter (new OutputStreamWriter (fos));  
        bw.write (data);  
    }  
}
```



```
    } catch (IOException e) {  
        e.printStackTrace ();  
    } finally {  
        if (null != bw) {  
            try {  
                bw.close ();  
            } catch (IOException e) {  
                e.printStackTrace ();  
            }  
        }  
    }  
}
```

这里通过 `openFileOutput()` 方法得到一个 `FileOutputStream` 对象，再通过它构建一个 `OutputStreamWriter` 对象，接着通过 `OutputStreamWriter` 对象构建出一个 `BufferedWriter` 对象，这样即可通过 `BufferedWriter` 对象将我们需要保存的内容写入到文件中了。

接下来通过实例学习如何在 Android 中使用文件存储。程序的主要功能是在进入程序时可在输入栏中输入需要保存的内容，退出程序时，自动保存输入的数据。

首先创建一个“FileStore”项目，并修改“activity_main.xml”中的代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
    tools:context=".MainActivity" >  
    <EditText  
        android:id="@+id/edit"  
        android:layout_width="match_parent"  
        android:layout_height="40dp"  
        android:hint="请输入需要保存的数据" />  
</LinearLayout>
```

在布局中加入一个 `EditText`，用于输入需要存储的内容。



MainActivity 的代码如下：

```
public class MainActivity extends AppCompatActivity {  
    EditText editText;  
  
    @Override  
    protected void onCreate ( Bundle savedInstanceState ) {  
        super.onCreate ( savedInstanceState );  
        setContentView ( R.layout.activity_ main );  
        editText = findViewById ( R.id.edit );  
    }  
  
    @Override  
    protected void onDestroy ( ) {  
        super.onDestroy ( );  
        String data = editText.getText ( ) . toString ( );  
        if ( !TextUtils.isEmpty ( data ) )  
            saveData ( data );  
    }  
  
    public void saveData ( String data ) {  
        FileOutputStream fos;  
        BufferedWriter bw = null;  
        try {  
            fos = openFileOutput ( " data.txt", MODE_ PRIVATE );  
            bw = new BufferedWriter ( new OutputStreamWriter ( fos ) );  
            bw.write ( data );  
        } catch ( IOException e ) {  
            e.printStackTrace ( );  
        } finally {  
            if ( null ! = bw ) {  
                try {  
                    bw.close ( );  
                }
```



```
        } catch (IOException e) {  
            e.printStackTrace ();  
        }  
    }  
}  
}
```

从代码中可以看到，先在 onCreate（）方法中获取到 EditText，然后重写 onDestroy（）方法，以此保证在 Activity 销毁前必须调用保存数据的方法。在 onDestroy（）方法中获取到 EditText 中输入的数据，若数据不为空，则调用 saveData（）方法，把输入的内容存储在文件中，文件命名为“data.txt”。运行程序，并输入需保存的内容，如图 8-1 所示。

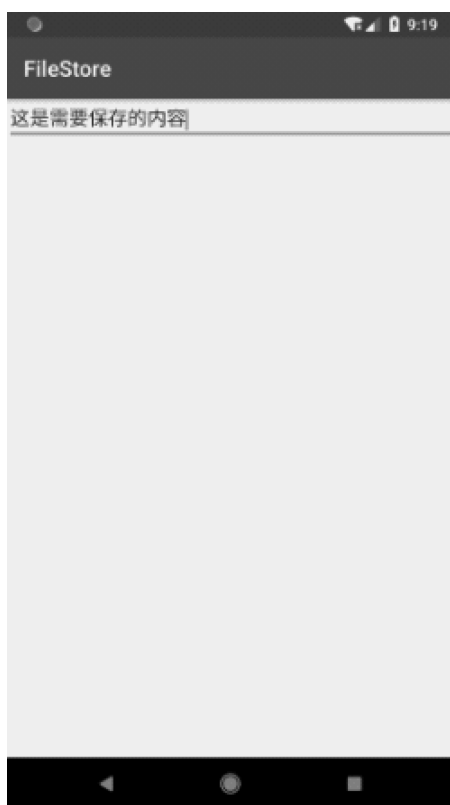


图 8-1 在 EditText 中输入内容

再按返回键关闭程序，此时用户输入的内容已保存在文件中。为验证输入数据已正确保存，可以借助 Android Studio 中的 Device File Explorer 工具查看。该工具默认在

Android Studio 右下方，点开“DeviceFileExplorer”即可看到连接的设备 and 目录结构，如图 8-2 所示。

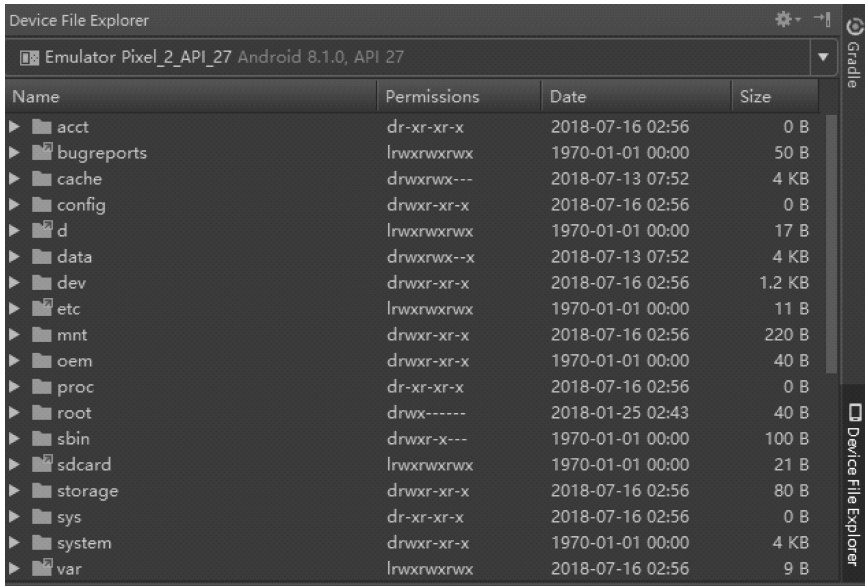


图 8-2 浏览设备目录

进入“data”目录下面的“data”目录，找到程序的包名单击进入，在“files”目录下面即可找到保存的文件“data. txt”。可以右击选择“Save As”将文件保存到电脑上，也可以双击直接打开查看，如图 8-3 所示。

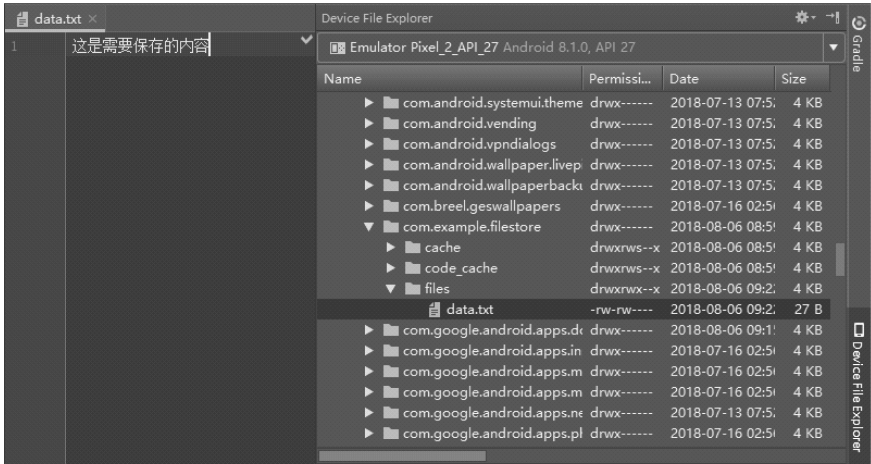


图 8-3 查看保存内容

可以看到文件中的内容和输入的内容一致，证实输入的内容已经保存在文件。接下来还需将存储在文件中的数据读取出来并显示在程序中。



8.2.2 从文件中读取数据

类似于存储数据在文件中，Context 类中还提供一个 `openFileInput()` 方法，用于从文件中读取数据。此方法与 `openFileOutput()` 相比较为简单，它只接收一个参数，即要读取的文件名，然后会自动到 “data/data/<package name>/files” 目录下寻找并加载此文件，若此文件存在，则返回一个 `FileInputStream` 对象，再通过 Java 流的方式读取数据。以下为一段示例代码：

```
public String loadData () {  
    FileInputStream fis;  
    BufferedReader br = null;  
    StringBuilder sb = new StringBuilder ();  
    try {  
        fis = openFileInput (" data.txt");  
        br = new BufferedReader (new InputStreamReader (fis));  
        String line;  
        while ( (line = br.readLine ()) != null) {  
            sb.append (line);  
        }  
    } catch (IOException e) {  
        e.printStackTrace ();  
    } finally {  
        if (null != br) {  
            try {  
                br.close ();  
            } catch (IOException e) {  
                e.printStackTrace ();  
            }  
        }  
    }  
    return sb.toString ();  
}
```



在这段代码中，通过 `openFileInput()` 方法获取 `FileInputStream` 对象，再借助它构建出一个 `InputStreamReader` 对象，接着再构建出一个 `BufferedReader` 对象，把文件中的所有数据都读取出来，最后再返回读取到的内容。

接下来继续完善上一小节的例子，启动程序时，显示之前存储的数据。修改 `MainActivity` 中的代码如下：

```
public class MainActivity extends AppCompatActivity {  
    EditText editText;  
  
    @Override  
    protected void onCreate (Bundle savedInstanceState) {  
        super.onCreate (savedInstanceState);  
        setContentView (R.layout.activity_main);  
        editText = findViewById (R.id.edit);  
        String data = loadData ();  
        if (!TextUtils.isEmpty (data)) {  
            editText.setText (data);  
            editText.setSelection (data.length ());  
        }  
    }  
  
    public String loadData () {  
        FileInputStream fis;  
        BufferedReader br = null;  
        StringBuilder sb = new StringBuilder ();  
        try {  
            fis = openFileInput ("data.txt");  
            br = new BufferedReader (new InputStreamReader (fis));  
            String line;  
            while ( (line = br.readLine ()) != null) {  
                sb.append (line);  
            }  
        } catch (IOException e) {  
            e.printStackTrace ();  
        }  
    }  
}
```



```
        } finally {
            if (null != br) {
                try {
                    br.close ();
                } catch (IOException e) {
                    e.printStackTrace ();
                }
            }
        }

        return sb.toString ();
    }

    @Override
    protected void onDestroy () {
        super.onDestroy ();
        String data = editText.getText ().toString ();
        if (!TextUtils.isEmpty (data))
            saveData (data);
    }

    public void saveData (String data) {
        FileOutputStream fos;
        BufferedWriter bw = null;
        try {
            fos = openFileOutput (" data.txt", MODE_PRIVATE);
            bw = new BufferedWriter (new OutputStreamWriter (fos));
            bw.write (data);
        } catch (IOException e) {
            e.printStackTrace ();
        } finally {
            if (null != bw) {
                try {
                    bw.close ();
                } catch (IOException e) {
                    e.printStackTrace ();
                }
            }
        }
    }
}
```



```
        } catch (IOException e) {  
            e.printStackTrace ();  
        }  
    }  
}  
}
```

此处思路较为简单，在 `onCreate ()` 方法中调用 `load ()` 方法读取文件中存储内容，若读取的内容不为空，则把内容填充在 `EditText` 中，并调用 `setSelection ()` 方法，将输入光标移动到文本的末尾以便继续输入。

上述代码在对字符串进行非空判断时使用 `TextUtils.isEmpty ()` 方法，它可以一次性进行两种空值的判断。当传入的字符串等于 `null` 或者等于空字符串时，均会返回 `true`，从而不需要判断此两种空值再使用逻辑运算符连接起来。重新运行程序，把保存的内容显示在 `EditText` 中，如图 8-4 所示。



图 8-4 显示存储的内容



对于 `MODE_ APPEND` 模式，此处不再深入讲解，读者可以自行实验体会，对比两种模式的区别。而文件存储的方式并不适合保存一些较为复杂的文本数据，因此，下面学习 Android 中另一种存储的方式，它比文件存储更加简单易用，且可以方便地对某已指定数据进行操作。

8.3 SharedPreferences 存储

8.3.1 将数据存储到 SharedPreferences

不同于文件存储的方式，SharedPreferences 是使用键值对的方式存储数据。用户在保存一条数据时，需要给这条数据提供一个对应的键，这样在读取数据时即可通过这个键把对应的值读取出。且 SharedPreferences 还支持多种不同的数据类型存储，如果存储的数据类型是整型，那么读取出的数据也为整型；如果存储的数据是字符串，那么读取出的数据也为字符串。

若要使用 SharedPreferences 存储数据，首先需要获取 SharedPreferences 对象。Android 中主要提供 3 种方法用于获取 SharedPreferences 对象。

1. Context 类中的 getSharedPreferences () 方法

此方法接收两个参数，第一个参数用于指定 SharedPreferences 文件的名称，如果指定的文件不存在则会创建一个，SharedPreferences 文件都是存放在 “data/data/ <package_name>/shared_ prefs” 目录下。第二个参数用于指定操作模式，目前只有 `MODE_ PRIVATE` 可选，表示只有当前的应用程序才能对这个 SharedPreferences 文件进行操作，其他的几种模式已被抛弃。

2. Activity 类中的 getPreferences () 方法

此方法与 Context 的 getSharedPreferences () 方法类似，但它只接收一个操作模式参数，它使用的文件名即为当前 Activity 的类名。

3. PreferenceManager 类中的 getDefaultSharedPreferences () 方法

这是一个静态方法，它只接收一个 Context 参数，以当前程序的包名作为前缀来命名 SharedPreferences 文件。

得到 SharedPreferences 对象后，即可继续操作。在 SharedPreferences 文件中存储数



据的步骤如下：

- (1) 调用 SharedPreferences 对象的 edit () 方法获取一个 Editor 对象。
- (2) 向 Editor 对象中添加数据，比如添加一个字符串使用 putString () 方法。
- (3) 调用 apply () 方法提交添加的数据，从而完成数据存储的操作。

接下来创建一个 SharedPreferences 项目，深入了解 SharedPreferences 存储的用法。

修改 “activity_ main.xml” 中的代码如下：

```
<LinearLayout xmlns: android = " http: //schemas. android. com/apk/res/android"
    xmlns: app = " http: //schemas. android. com/apk/res-auto"
    xmlns: tools = " http: //schemas. android. com/tools"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: orientation = " vertical"
    tools: context = " . MainActivity" >
    <Button
        android: id = " @ + id/button"
        android: layout_ width = " match_ parent"
        android: layout_ height = " wrap_ content"
        android: text = " 保存数据" />
</LinearLayout>
```

此处只放置一个按钮，用于把指定的数据存储到 SharedPreferences 文件中。

修改 MainActivity 中的代码如下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate ( Bundle savedInstanceState ) {
        super. onCreate ( savedInstanceState );
        setContentView ( R. layout. activity_ main );
        Button button = findViewById ( R. id. button );
        button. setOnClickListener ( new View. OnClickListener ( ) {
            @Override
            public void onClick ( View view ) {
```



```
SharedPreferences sp = getSharedPreferences (" data", MODE_ PRI-  
VATE);  
  
SharedPreferences.Editor editor = sp.edit ();  
editor.putString (" name", " 张三");  
editor.putInt (" 年龄", 20);  
editor.putBoolean (" 婚否", false);  
editor.apply ();  
  
}  
  
});  
  
}
```

通过这段代码，可以发现这里首先给按钮注册一个单击监听事件，再在监听中通过 `getSharedPreferences ()` 方法指定文件名为 `data`，同时调用 `edit ()` 方法得到 `SharedPreferences.Editor` 对象。再向此对象中添加 3 条不同类型的数据，最后再调用 `apply ()` 方法提交操作，从而完成数据存储的操作。

接下来查看 `SharedPreferences` 文件中的内容。在 Android Studio 中打开 `DeviceFileExplorer`，在 “`data/data/ <packageName >/shared_ prefs`” 目录下含有一个 “`data.xml`” 文件，右击保存到计算机或者直接双击打开查看，如图 8-5 所示。

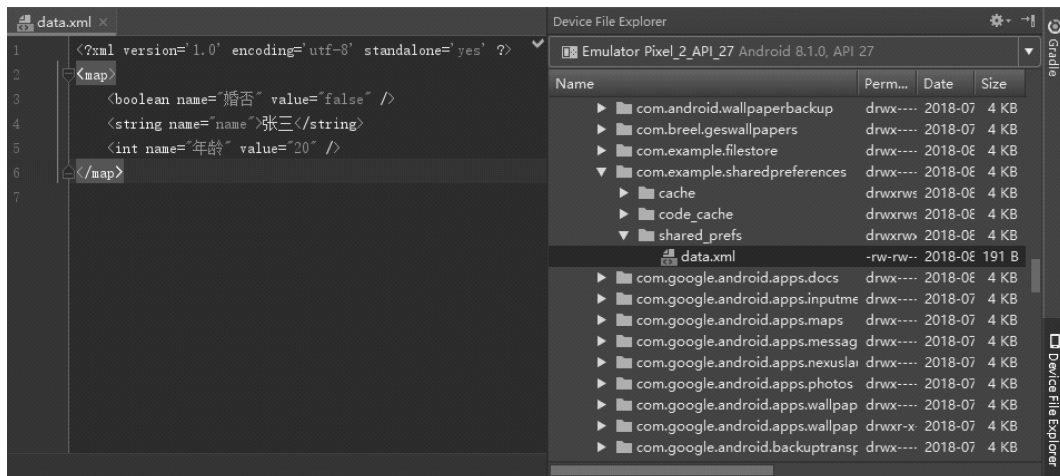


图 8-5 查看 `SharedPreferences` 内容

可发现在按钮的单击事件中添加的所有数据均已成功保存，并且 `SharedPreferences` 文件是以 XML 格式来对数据进行管理。



8.3.2 从 SharedPreferences 中读取数据

使用 SharedPreferences 存储数据较为简单，而从 SharedPreferences 文件中读取数据会更加简单。SharedPreferences 对象中提供一系列的 get 方法，用于读取存储数据，例如读取一个字符串可使用 getString（）方法。此类 get 方法需接收两个参数，第一个为键，传入存储数据时使用的键即可得到对应的值；第二个参数为默认值，当传入的键无法找到对应的值时就会以指定的默认值返回。

使用上一小节的“SharedPreferences”项目，修改“activity_main.xml”中的代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="保存数据" />

    <Button
        android:id="@+id/button_read_data"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="读取数据" />

</LinearLayout>
```

此处增加一个按钮，用于读取 SharedPreferences 文件中数据。

修改 MainActivity 中的代码如下：



```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate (savedInstanceState);
        setContentView (R.layout.activity_main);
        Button button = findViewById (R.id.button);
        button.setOnClickListener (new View.OnClickListener () {
            @Override
            public void onClick (View view) {
                SharedPreferences sp = getSharedPreferences("data", MODE_PRIVATE);
                SharedPreferences.Editor editor = sp.edit ();
                editor.putString (" name", " 张三");
                editor.putInt (" 年龄", 20);
                editor.putBoolean (" 婚否", false);
                editor.apply ();
            }
        });

        Button btReadData = findViewById (R.id.button_read_data);
        btReadData.setOnClickListener (new View.OnClickListener () {
            @Override
            public void onClick (View view) {
                SharedPreferences sp = getSharedPreferences("data", MODE_PRIVATE);
                String name = sp.getString (" name", " 未获取到数据");
                int age = sp.getInt (" 年龄", 0);
                boolean married = sp.getBoolean (" 婚否", false);
                Log.i (" MainActivity", " name is:" + name);
                Log.i (" MainActivity", " 年龄 is:" + age);
                Log.i (" MainActivity", " 婚否 is:" + married);
            }
        });
    }
}
```



可以看到，在还原数据按钮的点击事件中先得到 SharedPreferences 对象，再通过调用 getString ()、getInt ()、getBoolean () 方法获取对应存储的内容，再通过 Log 日志打印出来。重新运行程序，单击界面上的“读取数据”按钮，再在 logcat 中查看打印的信息，如图 8-6 所示。

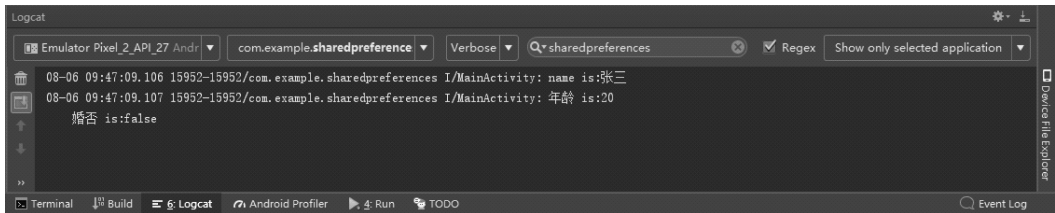


图 8-6 读取 SharedPreferences 数据

通过前面的学习，已经掌握如何使用 SharedPreferences 存储和读取数据。相比之下，SharedPreferences 存储比文本存储较为简单方便，应用场景，例如较多应用程序中的偏好设置功能均是使用 SharedPreferences 技术。但想要充分熟悉这些使用方法，需大量的练习和实践。

8.4 记事本开发

8.4.1 记事本开发要求

通过前面的学习，已经学会文件存储和 SharedPreferences 存储技术，为了更好地体验和应用这两种存储技术，接下来通过写一个简易的记事本，进一步了解和掌握这两种方式。这个记事本主要包含用户登录、记住密码、文本编辑保存、加载文本等功能。

8.4.2 实现过程

新建一个项目 MyNotePad，自动生成一个 MainActivity，再新建一个用户登录的页面 LoginActivity，同时将其作为启动页面，即修改“AndroidManifest.xml”的代码如下：

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

    <application
        android:allowBackup="true"
```



```
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportRtl="true"
android:theme="@style/AppTheme" >
<activity android:name=".MainActivity" />
<activity android:name=".LoginActivity" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
</application>
</manifest>
```

在用户登录页面中，需要添加输入用户名和密码的 EditText、是否记住密码的 CheckBox 和登录按钮，修改 “activity_login” 的代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="20dp"
    tools:context=".LoginActivity" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <TextView
            android:id="@+id/textView"
            android:layout_width="40dp"
            android:layout_height="wrap_content"
```



```
        android:layout_weight = " 1"
        android:gravity = " right"
        android:text = " 用户名" />

<EditText
    android:id = " @ + id/et_ name"
    android:layout_width = " wrap_ content"
    android:layout_height = " wrap_ content"
    android:layout_weight = " 1"
    android:ems = " 10"
    android:hint = " 请输入用户名"
    android:inputType = " textPersonName" />
</LinearLayout>

<LinearLayout
    android:layout_width = " match_ parent"
    android:layout_height = " wrap_ content"
    android:orientation = " horizontal" >

    <TextView
        android:id = " @ + id/textView2"
        android:layout_width = " 40dp"
        android:layout_height = " wrap_ content"
        android:layout_weight = " 1"
        android:gravity = " right"
        android:text = " 密码" />

    <EditText
        android:id = " @ + id/et_ pwd"
        android:layout_width = " wrap_ content"
        android:layout_height = " wrap_ content"
        android:layout_weight = " 1"
        android:ems = " 10"
        android:hint = " 请输入密码"
        android:inputType = " textPassword" />
</LinearLayout>
```



```
< CheckBox
```

```
    android: id = " @ + id/cb_ save_ pwd"
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content"
    android: layout_ marginStart = " 50dp"
    android: text = " 记住密码" />
```

```
< Button
```

```
    android: id = " @ + id/btn_ login"
    android: layout_ width = " match_ parent"
    android: layout_ height = " wrap_ content"
    android: onClick = " doClick"
    android: text = " 登录" />
```

```
</LinearLayout>
```

运行后的页面效果如图 8-7 所示。



图 8-7 LoginActivity 布局效果



接下来要实现 LoginActivity 中的功能，包括记住密码和登录跳转，修改 LoginActivity 中的代码如下：

```
public class LoginActivity extends AppCompatActivity {  
    Button btLogin;  
    EditText etName, etPwd;  
    CheckBox cbSavePwd;  
    SharedPreferences sp;  
    boolean isSavePwd;  
  
    @Override  
    protected void onCreate (Bundle savedInstanceState) {  
        super.onCreate (savedInstanceState);  
        setContentView (R.layout.activity_login);  
        initView ();  
        loadData ();  
    }  
  
    /**  
     * 加载数据，加载历史用户名和是否保存密码信息  
     */  
    private void loadData () {  
        sp = getSharedPreferences ("userInfo", MODE_PRIVATE);  
        isSavePwd = sp.getBoolean ("isSavePwd", false);  
        cbSavePwd.setChecked (isSavePwd);  
        String saveName = sp.getString ("name", "");  
        String savePwd = sp.getString ("pwd", "");  
        if (isSavePwd) {  
            etName.setText (saveName);  
            etPwd.setText (savePwd);  
        } else  
            etName.setText (saveName);  
  
        cbSavePwd.setOnCheckedChangeListener (new CompoundBut-  
ton.OnCheckedChangeListener () {
```



```
@Override
    public void onCheckedChanged (CompoundButton compoundButton, boolean b) {
        isSavePwd = b;
    }
});
}

/* *
 * 初始化控件
 */
private void initView () {
    btLogin = findViewById (R.id. btn_ login);
    etName = findViewById (R.id. et_ name);
    etPwd = findViewById (R.id. et_ pwd);
    cbSavePwd = findViewById (R.id. cb_ save_ pwd);
}

/* *
 * 按钮单击事件
 * 如果用户信息正确，则跳转到下一个页面，并保存用户信息，结束当前页面
 *
 * @param view 点击的按钮
 */
public void doClick (View view) {
    String name = etName.getText () . toString ();
    String pwd = etPwd.getText () . toString ();
    if (checkUserInfo (name, pwd)) {
        startActivity (new Intent (this, MainActivity. class));
        saveUserInfo (name, pwd);
        finish ();
    } else
        Toast.makeText (this, " 登录失败，请核对您的用户名和密码!",
Toast.LENGTH_ SHORT) . show ();
```




```
}

/* *
 * 保存用户登录信息
 * 如果用户勾选保存密码，则保存用户名和密码信息
 * 否则只保存用户名
 *
 * @param name 用户名
 * @param pwd 密码
 */
private void saveUserInfo (String name, String pwd) {
    if (isSavePwd)
        sp.edit () .putBoolean (" isSavePwd", true)
            .putString (" name", name)
            .putString (" pwd", pwd)
            .apply ();
    else
        sp.edit () .putBoolean (" isSavePwd", false)
            .putString (" name", name)
            .remove (" pwd")
            .apply ();
}

/* *
 * 检查用户信息
 * 如果用户为输入信息则提示用户名或密码不能为空
 * 如果用户名和密码都匹配，则通过检测（这里默认用户名：admin，默认密码：
admin)
 *
 * @param name 输入的用户名
 * @param pwd 输入的密码
 * @return 用户信息是否正确
 */
```



```
private boolean checkUserInfo (String name, String pwd) {  
    if (TextUtils.isEmpty (name) || TextUtils.isEmpty (pwd)) {  
        Toast.makeText (this, " 用户名或密码不能为空", Toast.LENGTH_   
SHORT) . show ();  
        return false;  
    }  
    return " admin" . equals (name) && " admin" . equals (pwd);  
}
```

通过以上代码，可以了解到在 onCreate () 方法中调用 initView () 方法来初始化所需要用的控件，然后再调用 loadData () 方法加载数据。在 loadData () 方法中，首先获取 SharedPreferences 对象，然后获取里面存入的信息，包括是否保存密码、用户名、密码的信息，再体现在控件中，同时还需为是否记住密码的 checkbox 添加监听，用于判断是否记住密码。

此处登录按钮的单击事件直接在 “activity_ login. xml” 布局文件中进行绑定，在 Button 的属性中含有一个 onClick 属性绑定 doClick 方法，也可在代码中为其添加单击监

听事件。在 doClick () 方法中，获取到用户输入的用户名和密码，再调用 checkUserInfo () 方法判断输入信息的正确性，若信息正确，则跳转到下一个页面，同时调用 saveUserInfo () 方法保存用户信息，再结束当前页面，避免返回时再次返回到登录页；若信息不正确，则给予提示。

上面提及的 checkUserInfo () 方法，主要判断用户输入的内容的规范性以及用户信息正确性。如果输入的内容为空，则给予对应的提示，并返回 “false”。如果用户信息正确则返回 “true”，否则返回 “false”。此处默认的用户名和密码都是 “admin”。在 saveUserInfo () 方法中，需要判断当前是否记住用户密码，如果选择 “是”，则保存用户名和密码；如果选择 “不是”，则保存用户名并移除密码。重新运行程序，并记住密码，效果如图 8-8 所示。



图 8-8 记住密码

登录成功后，将会跳转到 MainActivity 页面，对



应的布局文件“activity_ main.xml”代码修改如下：

```
<LinearLayout xmlns: android = http: //schemas. android. com/apk/res/android
    xmlns: app = " http: //schemas. android. com/apk/res-auto"
    xmlns: tools = " http: //schemas. android. com/tools"
    android: layout_ width = " match_ parent"
    android: layout_ height = " match_ parent"
    android: orientation = " vertical"
    android: padding = " 20dp"
    tools: context = " . MainActivity" >

    <EditText
        android: id = " @ + id/editText"
        android: layout_ width = " match_ parent"
        android: layout_ height = " 0dp"
        android: layout_ weight = " 1"
        android: ems = " 10"
        android: gravity = " top"
        android: minLines = " 1" />

</LinearLayout>
```

从此段代码可以看到，此处只添加一个 EditText，用于显示和编辑内容。同时需要把内容居上显示，最小行数设为 1，这样在输入时可自动换行。

为实现需要的功能，修改 MainActivity 的代码如下：

```
public class MainActivity extends AppCompatActivity {
    EditText et;

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



```
/* *
 * 初始化控件
 */
private void initView () {
    et = findViewById (R. id. editText);
}

/* *
 * 加载读取到的数据到 EditText 中
 */
private void loadData () {
    String data = readData ();
    if (! TextUtils. isEmpty (data)) {
        et. setText (data);
        et. setSelection (data. length ());
    }
}

/* *
 * 读取 myNotePad. txt 文件中的数据
 *
 * @return 读取到的内容
 */
private String readData () {
    FileInputStream fis;
    BufferedReader br = null;
    StringBuilder sb = new StringBuilder ();
    try {
        fis = openFileInput (" myNotePad. txt");
        br = new BufferedReader (new InputStreamReader (fis));
        String line;
        while ( (line = br. readLine ()) != null) {
            sb. append (line);
        }
    }
```



```
        } catch (IOException e) {
            e.printStackTrace ();
        } finally {
            if (null != br) {
                try {
                    br.close ();
                } catch (IOException e) {
                    e.printStackTrace ();
                }
            }
        }
        return sb.toString ();
    }

    /**
     * 保存数据到文件中，这里默认的文件名为 myNotePad.txt
     *
     * @param data 需要保存的数据
     */
    public void saveData (String data) {
        FileOutputStream fos;
        BufferedWriter bw = null;
        try {
            fos = openFileOutput (" myNotePad.txt", MODE_ PRIVATE);
            bw = new BufferedWriter (new OutputStreamWriter (fos));
            bw.write (data);
        } catch (IOException e) {
            e.printStackTrace ();
        } finally {
            if (null != bw) {
                try {
                    bw.close ();
                } catch (IOException e) {
                    e.printStackTrace ();
                }
            }
        }
    }
}
```



```
        } catch (IOException e) {  
            e.printStackTrace ();  
        }  
    }  
}  
  
/**  
 * 退出前，保存已输入的内容  
 */  
@Override  
protected void onDestroy () {  
    super.onDestroy ();  
    String data = et.getText ().toString ();  
    if (!TextUtils.isEmpty (data))  
        saveData (data);  
}
```

通过以上代码可以发现，在 MainActivity 创建时，会首先调用 initView () 方法初始化控件，然后调用 loadData () 方法加载已输入的内容在 EditText 中显示。loadData () 方法调用 readData () 方法读取存放在文件名为“myNotePad.txt”中的数据。在 MainActivity 销毁时，调用 saveData () 方法，自动保存用户输入的内容。saveData () 方法和 readData () 方法中操作的文件是同一个，因此文件名必须保持一致，在这段代码中，文件名为“myNotePad.txt”。重新运行程序，输入内容进行查看，如图 8-9 所示。

也可以打开 Android Studio 中的 DeviceFileExplorer 查看文件存储的文件“myNotePad.txt”和 SharedPreferences 存储的文件“userInfo.xml”，如图 8-10 所示。



图 8-9 记事本输入内容

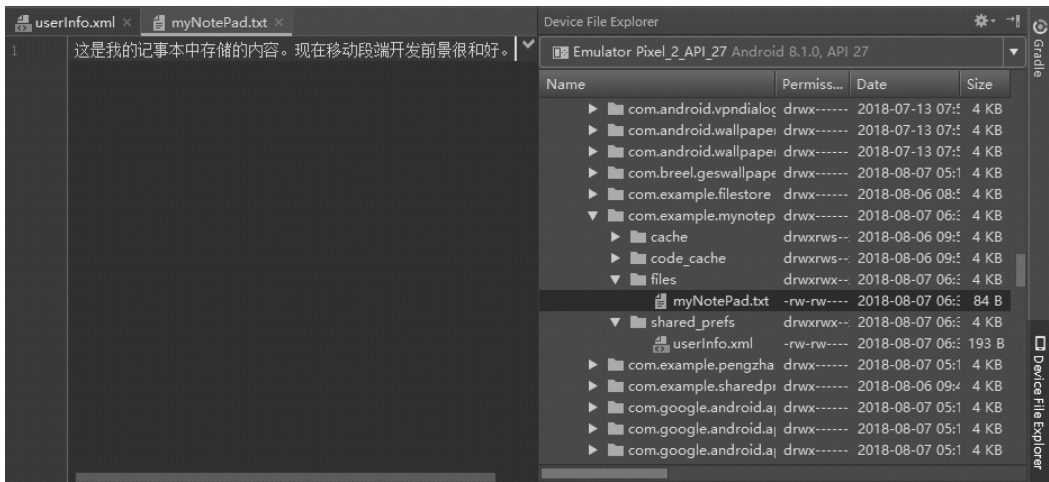


图 8-10 查看存储文件

已实现记事本的基本功能，但有不少地方可以改进，例如自动登录、用户注册、把存储的文件放到 sdcard 目录下（非 data 分区），读者可自行思考如何完善。

本章主要介绍文件存储和 SharedPreferences 存储的两种方式，这是两种简单且实用的存储方式，尤其是 SharedPreferences 方法使用频率较高，如果涉及其他的存储方式，读者可以参考其它资料，根据项目的实际需求选择合适存储方式，这也是读者在未来需要探索的内容。

练习题

- (1) Android 中数据存储主要有_____、_____和_____这 3 种。
- (2) SharedPreferences 是以_____格式存储数据的。
- (3) File 文件存储数据时，通过 Context 类的_____方法获取 IO 输出流。
- (4) SharedPreferences 存储适用场景？使用时需要注意什么？
- (5) 在读取数据量较大的文件（比如图片、视频、音频）等，可以有哪些优化方法？

参考文献

- [1] 郭霖. 第一行代码 [M]. 北京: 人民邮电出版社, 2016.
- [2] 李鸥. 实战 Android 应用开发 [M]. 北京: 清华大学出版社, 2012.
- [3] 郑凤. 移动互联网技术架构及其发展 [M]. 北京: 人民邮电出版社, 2013.
- [4] 吴亚峰. Android 应用案例开发大全 [M]. 北京: 人民邮电出版社, 2013.
- [5] 黄隽实. Android 和 PHP 开发最佳实践 [M]. 北京: 机械工业出版社, 2013.
- [6] 杨青平. 深入理解 Android [M]. 北京: 机械工业出版社, 2013.
- [7] 李桂珍. 职业技能大赛对推动《Android 应用开发》课程项目化改革的探索 [J]. 价值工程, 2017, 36 (35): 71-72.