

TURING

图灵原創



第一行代码

Android

第 2 版

郭霖 ◎ 著

人民邮电出版社
北京

图书在版编目 (C I P) 数据

第一行代码——Android / 郭霖著. -- 2版. -- 北京 : 人民邮电出版社, 2016.12
(图灵原创)
ISBN 978-7-115-43978-9

I. ①第… II. ①郭… III. ①移动终端—应用程序—程序设计 IV. ①TN929.53

中国版本图书馆CIP数据核字(2016)第267736号

内 容 提 要

本书被广大 Android 开发者誉为“Android 学习第一书”。全书系统全面、循序渐进地介绍了 Android 软件开发的必备知识、经验和技巧。

第 2 版基于 Android 7.0 对第 1 版进行了全面更新，将所有知识点都在最新的 Android 系统上进行重新适配，使用全新的 Android Studio 开发工具代替之前的 Eclipse，并添加了对 Material Design、运行时权限、Gradle、RecyclerView、百分比布局、OkHttp、Lambda 表达式等全新知识点的详细讲解。

本书内容通俗易懂，由浅入深，既是 Android 初学者的入门必备，也是 Android 开发者的进阶首选。

-
- ◆ 著 郭霖
 - 责任编辑 王军花
 - 执行编辑 张霞
 - 责任印制 彭志环
 - 封面插画 巫俊武
 - 封面设计 潘建永 陈冰
 - ◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 <http://www.ptpress.com.cn>
北京鑫正大印刷有限公司印刷
 - ◆ 开本：800×1000 1/16
印张：36.25
字数：856千字 2016年12月第2版
印数：89 001~99 000册 2016年12月北京第2次印刷
-

定价：79.00元

读者服务热线：(010)51095186转600 印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广字第 8052 号

前　　言

虽然我从事Android开发工作已经很多年了，但是之前从来没有想过自己要去写一本Android技术相关的书。在我看来，写一本书可以算是一个很庞大的工程，写一本好书的难度并不亚于开发一款好的应用程序。

由于我长期坚持在CSDN上发表技术博文，因而得到了大量网友的认可，也积累了一定的名气。很荣幸的是，人民邮电出版社图灵公司的前副总编辑陈冰老师联系上了我，希望我可以写一本关于Android开发技术的书，这着实让我受宠若惊。

在写本书第1版的时候，我可以说是费了相当大的心思。写书和写博客最大的区别在于，书的内容不能像博客那样散乱，不能想到哪里写到哪里，而是一定要系统化，要循序渐进，基本上在写第1章的时候就应该把全书的内容都确定下来了。

令我非常欣慰的是，本书的第1版在推出之后获得了广大读者的强烈认可，在短短两年时间内，已经成为了国内最畅销的Android技术书。各大书店、图书馆都能看到《第一行代码》的身影，许多学校和培训机构也纷纷将《第一行代码》选为Android课程的教材。

不过，在科技高速发展的今天，各种技术的发展都是日新月异的。在两年的时间里，Android操作系统经历了5.0、6.0、7.0的飞速升级。不可否认的是，本书第1版中的不少知识点都已经过时，而且这两年间出现的很多新知识，第1版中也没有涵盖。因此，这让我坚定了写作本书第2版的想法。

刚开始写的时候，我以为只是小修小补，但事实上并没有我想象得那么轻松。除了介绍新知识点以外，书中之前的所有项目都需要重新编写和测试，以保证代码在新老系统上的兼容性。另外，由于Android从5.0系统开始，UI风格变化很大，因此第1版中所有的截图都需要更新。毫不夸张地说，我几乎重写了整本书。

而现在，你手中捧着的正是全新版的《第一行代码》，同时这也是国内第一本基于Android 7.0系统写作的技术书。我真诚地希望你可以用心去阅读这本书，因为每多掌握一份知识，你就会多一份喜悦。Enjoy it!

第2版的变化

由于第2版修改内容繁多，因此这里我只列举出最主要的变化。首先是开发工具上的改变，本书第1版使用的开发工具是Eclipse，而第2版使用了目前最新的Android Studio 2.2版本。另外，

本书第1版是基于Android 4.x系统的，而第2版是基于Android 7.0系统的，其中囊括了新系统中的诸多知识点，包括Android 5.0系统中引入的Material Design、Android 6.0系统中引入的运行时权限和Doze模式、Android 7.0系统中引入的多窗口模式等。

除此之外，第2版还加入了Gradle、RecyclerView、百分比布局、OkHttp、Lambda表达式等全新知识点的讲解，内容将前所未有地充实。

读者对象

本书内容通俗易懂，由浅入深，既适合初学者阅读，也同样适合专业人员。学习本书内容之前，你并不需要有任何的Android基础，但是你需要有一定的Java基础，因为Android开发都是使用Java语言的，而本书并不会去专门介绍Java方面的知识。

阅读本书时，你可以根据自身的情况来决定如何阅读。如果你是初学者的话，建议你从第1章开始循序渐进地阅读，这样理解起来就不会感到吃力。而如果你已经有了一定的Android基础，那么就可以选择某些你感兴趣的章节进行跳跃式的阅读。但请记住，很多章最后的最佳实践部分一定是你不想错过的。

本书内容

正如前面所说，本书的内容是非常系统化的，不仅全面介绍了那些你必须掌握的知识，而且保证了各章的难度都是梯度式上升的。全书一共分为15章，涵盖了四大组件、UI、碎片、数据存储、多媒体、网络、定位服务等方方面面的知识。为了让你在学完所有内容之后还可以有综合运用的能力，本书的尾声部分还会带你一起开发一个天气预报程序，并教会你如何将程序发布到应用商店，以及如何在程序中嵌入广告盈利。

除此之外，本书的第5章、第7章、第11章、第14章中都穿插有对Git的讲解，如果想要掌握它的用法，这几章的内容是绝对不能错过的。

本书中各个章节的内容都相对比较独立，因此除了可以循序渐进地学习之外，你还可以把它当成一本参考手册，随时查阅。

源码下载

首先，我建议你在学习本书的时候将所有项目的源码都亲手敲上一遍，因为只有这样才能加深你对代码的理解。不过为了方便于你的学习，我还是提供了书中所有项目的源码，请仅在需要的时候再去参考（如下载项目中的图片资源）。切勿直接将源码复制粘贴就当成自己的东西了，只有亲手敲过的代码才真正是你自己的。

源码下载地址：<https://github.com/guolindev/booksource>。

致 谢

在这近一年的时间里，我又完成了一项浩大的工程。和写作本书第1版时的感觉类似，当全书完稿之后，回顾整本书，我仍然不敢相信这所有的内容竟然是我一字字地敲出来的。

如今这已经是我写的第二本书了，和写第一本书时的情况不同，现在我有了更广的人脉和资源，有了更多的人愿意帮助和支持我来完成一本更好的技术书。因此，我要在这里对很多人表示感谢。

首先我要感谢我的父母，感谢你们将我抚养长大，感谢你们的付出，让我从小不用为生计、上学而发愁，可以一直做我自己想做的事情，也感谢你们指引我走上了技术这条路。

其次我要感谢我的妻子，感谢你每天为我准备好一日三餐，感谢你对我永远的包容，不管是平日的加班还是没日没夜的写书，你都一直默默地理解和支持我。

我还非常感谢本书第1版的编辑陈冰老师，如果没有你当初在CSDN上找到我，并邀请我写书，就不会有现在的《第一行代码》。另外，你也是当时唯一一个坚信这本书一定会大卖的人，甚至连我自己当时都没有如此的眼光。

我也非常感谢本书第2版的编辑张霞，你全程负责了第2版的出版工作，并且完成得非常出色。你对文字的把控能力让我敬佩，感谢你对书中每一章节的尽心审阅，才能让这本书更趋近于完美。

另外我还要特别感谢一部分人，你们对本书的试读、内容建议、勘误检查、代码纠错，甚至是对我个人的支持等都作出了卓越的贡献。有了你们的帮助，才会有这样一本更加出色的书呈现在所有人面前，这本书上也理应有你们的名字（按姓氏拼音排序，排名不分先后）：

陈建林 陈俊杰 陈雷 陈龙 陈琪 陈秀相 陈逸鸣 代云蛟 董霖轩 段郭森
高鹤泉 高太稳 关爱民 何以诚 胡恩泽 黄楠 赖帆 李济州 李建友 李沛明
李潭 李永鹏 李志云 林火荣 刘萌 刘明渊 刘治国 陆德俊 罗亚超 吕国鑫
马文杰 覃文斌 孙建飞 王柏强 王光东 王杰 王龙 王路路 王鹏 王荣宗
王善昌 韦振南 吴波 吴宏权 吴绍志 徐阳 轩仲宽 杨辉 易静杰 查童
张鸿洋 张英祥 赵翠龙 赵庆元 赵迎超 郑传书 郑敏馨 庄育锋 周苏 朱海丰

目 录

第1章 开始启程——你的第一行

Android 代码	1
1.1 了解全貌——Android 王国简介	2
1.1.1 Android 系统架构	2
1.1.2 Android 已发布的版本	3
1.1.3 Android 应用开发特色	4
1.2 手把手带你搭建开发环境	5
1.2.1 准备所需要的工具	5
1.2.2 搭建开发环境	5
1.3 创建你的第一个 Android 项目	9
1.3.1 创建 HelloWorld 项目	9
1.3.2 启动模拟器	12
1.3.3 运行 HelloWorld	15
1.3.4 分析你的第一个 Android 程序	16
1.3.5 详解项目中的资源	22
1.3.6 详解 build.gradle 文件	23
1.4 前行必备——掌握日志工具的使用	26
1.4.1 使用 Android 的日志工具 Log	26
1.4.2 为什么使用 Log 而不使用 System.out	27
1.5 小结与点评	29

第2章 先从看得到的入手——探究

活动	30
2.1 活动是什么	30
2.2 活动的基本用法	30
2.2.1 手动创建活动	31
2.2.2 创建和加载布局	32

2.2.3 在 AndroidManifest 文件中 注册	35
2.2.4 在活动中使用 Toast	37
2.2.5 在活动中使用 Menu	38
2.2.6 销毁一个活动	40
2.3 使用 Intent 在活动之间穿梭	41
2.3.1 使用显式 Intent	41
2.3.2 使用隐式 Intent	44
2.3.3 更多隐式 Intent 的用法	46
2.3.4 向下一个活动传递数据	50
2.3.5 返回数据给上一个活动	51
2.4 活动的生命周期	53
2.4.1 返回栈	53
2.4.2 活动状态	54
2.4.3 活动的生存期	55
2.4.4 体验活动的生命周期	56
2.4.5 活动被回收了怎么办	62
2.5 活动的启动模式	63
2.5.1 standard	64
2.5.2 singleTop	65
2.5.3 singleTask	67
2.5.4 singleInstance	68
2.6 活动的最佳实践	71
2.6.1 知晓当前是在哪一个活动	71
2.6.2 随时随地退出程序	72
2.6.3 启动活动的最佳写法	74
2.7 小结与点评	75

第3章 软件也要拼脸蛋——UI 开发的点点滴滴	76	第4章 手机平板要兼顾——探究碎片	142
3.1 如何编写程序界面	76	4.1 碎片是什么	142
3.2 常用控件的使用方法	77	4.2 碎片的使用方式	144
3.2.1 TextView	77	4.2.1 碎片的简单用法	144
3.2.2 Button	80	4.2.2 动态添加碎片	147
3.2.3 EditText	82	4.2.3 在碎片中模拟返回栈	150
3.2.4 ImageView	86	4.2.4 碎片和活动之间进行通信	151
3.2.5 ProgressBar	88	4.3 碎片的生命周期	151
3.2.6 AlertDialog	91	4.3.1 碎片的状态和回调	151
3.2.7 ProgressDialog	93	4.3.2 体验碎片的生命周期	153
3.3 详解 4 种基本布局	94	4.4 动态加载布局的技巧	156
3.3.1 线性布局	94	4.4.1 使用限定符	156
3.3.2 相对布局	100	4.4.2 使用最小宽度限定符	159
3.3.3 帧布局	103	4.5 碎片的最佳实践——一个简易版的新闻应用	160
3.3.4 百分比布局	105	4.6 小结与点评	169
3.4 系统控件不够用？创建自定义控件	108		
3.4.1 引入布局	109		
3.4.2 创建自定义控件	111		
3.5 最常用和最难用的控件——ListView	113		
3.5.1 ListView 的简单用法	114		
3.5.2 定制 ListView 的界面	115		
3.5.3 提升 ListView 的运行效率	119		
3.5.4 ListView 的点击事件	120		
3.6 更强大的滚动控件——RecyclerView	122		
3.6.1 RecyclerView 的基本用法	122		
3.6.2 实现横向滚动和瀑布流布局	125		
3.6.3 RecyclerView 的点击事件	130		
3.7 编写界面的最佳实践	132		
3.7.1 制作 Nine-Patch 图片	132		
3.7.2 编写精美的聊天界面	135		
3.8 小结与点评	141		
第5章 全局大喇叭——详解广播机制	170		
5.1 广播机制简介	170		
5.2 接收系统广播	171		
5.2.1 动态注册监听网络变化	171		
5.2.2 静态注册实现开机启动	174		
5.3 发送自定义广播	177		
5.3.1 发送标准广播	177		
5.3.2 发送有序广播	179		
5.4 使用本地广播	183		
5.5 广播的最佳实践——实现强制下线功能	185		
5.6 Git 时间——初识版本控制工具	192		
5.6.1 安装 Git	192		
5.6.2 创建代码仓库	193		
5.6.3 提交本地代码	195		
5.7 小结与点评	195		

第6章 数据存储全方案——详解

持久化技术	196
6.1 持久化技术简介	196
6.2 文件存储	197
6.2.1 将数据存储到文件中	197
6.2.2 从文件中读取数据	201
6.3 SharedPreferences 存储	203
6.3.1 将数据存储到 SharedPreferences 中	203
6.3.2 从 SharedPreferences 中读取数据	206
6.3.3 实现记住密码功能	208
6.4 SQLite 数据库存储	211
6.4.1 创建数据库	211
6.4.2 升级数据库	216
6.4.3 添加数据	219
6.4.4 更新数据	222
6.4.5 删除数据	224
6.4.6 查询数据	225
6.4.7 使用 SQL 操作数据库	228
6.5 使用 LitePal 操作数据库	229
6.5.1 LitePal 简介	229
6.5.2 配置 LitePal	230
6.5.3 创建和升级数据库	231
6.5.4 使用 LitePal 添加数据	236
6.5.5 使用 LitePal 更新数据	237
6.5.6 使用 LitePal 删除数据	240
6.5.7 使用 LitePal 查询数据	241
6.6 小结与点评	243

第7章 跨程序共享数据——探究

内容提供器	244
7.1 内容提供器简介	244
7.2 运行时权限	245
7.2.1 Android 权限机制详解	245
7.2.2 在程序运行时申请权限	249

7.3 访问其他程序中的数据

7.3.1 ContentResolver 的基本用法	254
7.3.2 读取系统联系人	256
7.4 创建自己的内容提供器	260
7.4.1 创建内容提供器的步骤	261
7.4.2 实现跨程序数据共享	265
7.5 Git 时间——版本控制工具进阶	275
7.5.1 忽略文件	275
7.5.2 查看修改内容	276
7.5.3 撤销未提交的修改	278
7.5.4 查看提交记录	279
7.6 小结与点评	280

第8章 丰富你的程序——运用手机**多媒体**

8.1 将程序运行到手机上	281
8.2 使用通知	283
8.2.1 通知的基本用法	283
8.2.2 通知的进阶技巧	289
8.2.3 通知的高级功能	291
8.3 调用摄像头和相册	293
8.3.1 调用摄像头拍照	294
8.3.2 从相册中选择照片	298
8.4 播放多媒体文件	303
8.4.1 播放音频	303
8.4.2 播放视频	307
8.5 小结与点评	311

第9章 看看精彩的世界——使用**网络技术**

9.1 WebView 的用法	312
9.2 使用 HTTP 协议访问网络	314
9.2.1 使用 HttpURLConnection	315
9.2.2 使用 OkHttp	319
9.3 解析 XML 格式数据	321
9.3.1 Pull 解析方式	324

9.3.2 SAX 解析方式.....	326	11.4 使用百度地图.....	395
9.4 解析 JSON 格式数据.....	329	11.4.1 让地图显示出来.....	395
9.4.1 使用 JSONObject	330	11.4.2 移动到我的位置.....	397
9.4.2 使用 GSON	331	11.4.3 让“我”显示在地图上	400
9.5 网络编程的最佳实践.....	334	11.5 Git 时间——版本控制工具的高级	
9.6 小结与点评.....	338	用法	402
第 10 章 后台默默的劳动者——探究服务	339	11.5.1 分支的用法	403
10.1 服务是什么	339	11.5.2 与远程版本库协作	404
10.2 Android 多线程编程	340	11.6 小结与点评	406
10.2.1 线程的基本用法	340		
10.2.2 在子线程中更新 UI.....	341		
10.2.3 解析异步消息处理机制	345		
10.2.4 使用 AsyncTask	347		
10.3 服务的基本用法	349		
10.3.1 定义一个服务	349		
10.3.2 启动和停止服务	352		
10.3.3 活动和服务进行通信	355		
10.4 服务的生命周期	359		
10.5 服务的更多技巧	359		
10.5.1 使用前台服务	359		
10.5.2 使用 IntentService	361		
10.6 服务的最佳实践——完整版的下载示例	365		
10.7 小结与点评	378		
第 11 章 Android 特色开发——基于位置的服务	379		
11.1 基于位置的服务简介	379		
11.2 申请 API Key	380		
11.3 使用百度定位	384		
11.3.1 准备 LBS SDK	384		
11.3.2 确定自己位置的经纬度	386		
11.3.3 选择定位模式	391		
11.3.4 看得懂的位置信息	393		
第 12 章 最佳的 UI 体验——Material Design 实战	407		
12.1 什么是 Material Design	407		
12.2 Toolbar	408		
12.3 滑动菜单	415		
12.3.1 DrawerLayout	415		
12.3.2 NavigationView	418		
12.4 悬浮按钮和可交互提示	423		
12.4.1 FloatingActionButton	424		
12.4.2 Snackbar	427		
12.4.3 CoordinatorLayout	428		
12.5 卡片式布局	430		
12.5.1 CardView	431		
12.5.2 AppBarLayout	437		
12.6 下拉刷新	440		
12.7 可折叠式标题栏	443		
12.7.1 CollapsingToolbarLayout	443		
12.7.2 充分利用系统状态栏空间	453		
12.8 小结与点评	456		
第 13 章 继续进阶——你还应该掌握的高级技巧	457		
13.1 全局获取 Context 的技巧	457		
13.2 使用 Intent 传递对象	461		
13.2.1 Serializable 方式	461		
13.2.2 Parcelable 方式	463		

13.3	定制自己的日志工具	464
13.4	调试 Android 程序	466
13.5	创建定时任务	469
13.5.1	Alarm 机制	469
13.5.2	Doze 模式	471
13.6	多窗口模式编程	472
13.6.1	进入多窗口模式	473
13.6.2	多窗口模式下的生命周期	475
13.6.3	禁用多窗口模式	479
13.7	Lambda 表达式	481
13.8	总结	485
第 14 章 进入实战——开发酷欧天气 486		
14.1	功能需求及技术可行性分析	486
14.2	Git 时间——将代码托管到 GitHub 上	489
14.3	创建数据库和表	494
14.4	遍历全国省市县数据	499
14.5	显示天气信息	509
14.5.1	定义 GSON 实体类	509
14.5.2	编写天气界面	514
14.5.3	将天气显示到界面上	520
14.5.4	获取必应每日一图	526
14.6	手动更新天气和切换城市	532
14.6.1	手动更新天气	532
14.6.2	切换城市	535
14.7	后台自动更新天气	540
14.8	修改图标和名称	542
14.9	你还可以做的事情	543
第 15 章 最后一步——将应用发布到 360 应用商店 545		
15.1	生成正式签名的 APK 文件	545
15.1.1	使用 Android Studio 生成	546
15.1.2	使用 Gradle 生成	548
15.1.3	生成多渠道 APK 文件	551
15.2	申请 360 开发者账号	554
15.3	发布应用程序	556
15.4	嵌入广告进行盈利	560
15.4.1	注册腾讯广告联盟账号	560
15.4.2	新建媒体和广告位	562
15.4.3	接入广告 SDK	564
15.4.4	重新发布应用程序	569
15.5	结束语	570

第 1 章

开始启程——你的第一行 Android 代码

欢迎你来到 Android 世界！Android 系统是目前世界上市场占有率最高的移动操作系统，不管你在哪里，都可以看到 Android 手机几乎无处不在。今天的 Android 世界可谓欣欣向荣，可是你知道它的过去是什么样的吗？我们一起来看一看它的发展史吧。

2003 年 10 月，Andy Rubin 等人一起创办了 Android 公司。2005 年 8 月谷歌收购了这家仅仅成立了 22 个月的公司，并让 Andy Rubin 继续负责 Android 项目。在经过了数年的研发之后，谷歌终于在 2008 年推出了 Android 系统的第一个版本。但自那之后，Android 的发展就一直受到重重阻挠。乔布斯自始至终认为 Android 是一个抄袭 iPhone 的产品，里面剽窃了诸多 iPhone 的创意，并声称一定要毁掉 Android。而本身就是基于 Linux 开发的 Android 操作系统，在 2010 年被 Linux 团队从 Linux 内核主线中除名。又由于 Android 中的应用程序都是使用 Java 开发的，甲骨文则针对 Android 侵犯 Java 知识产权一事对谷歌提起了诉讼……

可是，似乎再多的困难也阻挡不了 Android 快速前进的步伐。由于谷歌的开放政策，任何手机厂商和个人都能免费获取到 Android 操作系统的源码，并且可以自由地使用和定制。三星、HTC、摩托罗拉、索爱等公司都推出了各自系列的 Android 手机，Android 市场上百花齐放。仅仅推出两年后，Android 就超过了已经霸占市场逾十年的诺基亚 Symbian，成为了全球第一大智能手机操作系统，并且每天都还会有数百万台新的 Android 设备被激活。而近几年，国内的手机厂商也是大放异彩，小米、华为、魅族等新兴品牌都推出了相当不错的 Android 手机，并且也获得了市场的广泛认可，目前 Android 已经占据了全球智能手机操作系统 70% 以上的份额。

说了这些，想必你已经体会到 Android 系统炙手可热的程度，并且迫不及待地想要加入到 Android 开发者的行列当中了吧。试想一下，十个人中有七个人的手机都可以运行你编写的应用程序，还有什么能比这个更诱人的呢？那么从今天起，我就带你踏上学习 Android 的旅途，一步步地引导你成为一名出色的 Android 开发者。

好了，现在我们就来一起初窥一下 Android 世界吧。

1.1 了解全貌——Android 王国简介

Android 从面世以来到现在已经发布了二十几个版本了。在这几年的发展过程中，谷歌为 Android 王国建立了一个完整的生态系统。手机厂商、开发者、用户之间相互依存，共同推进着 Android 的蓬勃发展。开发者在其中扮演着不可或缺的角色，因为如果没有开发者来制作丰富的应用程序，那么不管多么优秀的操作系统，也是难以得到大众用户喜爱的，相信没有多少人能够忍受没有 QQ、微信的手机吧。而且，谷歌推出的 Google Play 更是给开发者带来了大量的机遇，只要你能制作出优秀的产品，在 Google Play 上获得了用户的认可，你就完全可以得到不错的经济回报，从而成为一名独立开发者，甚至是成功创业！

那我们现在就以一个开发者的角度，去了解一下这个操作系统吧。纯理论型的东西也比较无聊，怕你看睡着了，因此我只挑重点介绍，这些东西跟你以后的开发工作都是息息相关的。

1.1.1 Android 系统架构

为了让你能够更好地理解 Android 系统是怎么工作的，我们先来看一下它的系统架构。Android 大致可以分为四层架构：Linux 内核层、系统运行库层、应用框架层和应用层。

1. Linux 内核层

Android 系统是基于 Linux 内核的，这一层为 Android 设备的各种硬件提供了底层的驱动，如显示驱动、音频驱动、照相机驱动、蓝牙驱动、Wi-Fi 驱动、电源管理等。

2. 系统运行库层

这一层通过一些 C/C++ 库来为 Android 系统提供了主要的特性支持。如 SQLite 库提供了数据库的支持，OpenGL|ES 库提供了 3D 绘图的支持，Webkit 库提供了浏览器内核的支持等。

同样在这一层还有 Android 运行时库，它主要提供了一些核心库，能够允许开发者使用 Java 语言来编写 Android 应用。另外，Android 运行时库中还包含了 Dalvik 虚拟机（5.0 系统之后改为 ART 运行环境），它使得每一个 Android 应用都能运行在独立的进程当中，并且拥有一个自己的 Dalvik 虚拟机实例。相较于 Java 虚拟机，Dalvik 是专门为移动设备定制的，它针对手机内存、CPU 性能有限等情况做了优化处理。

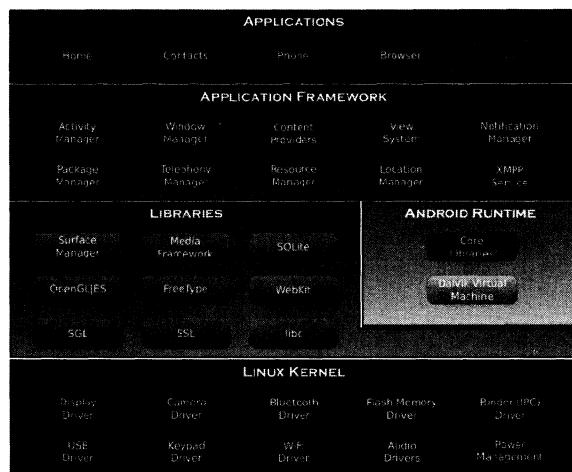
3. 应用框架层

这一层主要提供了构建应用程序时可能用到的各种 API，Android 自带的一些核心应用就是使用这些 API 完成的，开发者也可以通过使用这些 API 来构建自己的应用程序。

4. 应用层

所有安装在手机上的应用程序都是属于这一层的，比如系统自带的联系人、短信等程序，或者是你从 Google Play 上下载的小游戏，当然还包括你自己开发的程序。

结合图 1.1 你将会理解得更加深刻，图片源自维基百科。



1.1.2 Android 已发布的版本

2008年9月，谷歌正式发布了Android 1.0系统，这也是Android系统最早的版本。随后的几年，谷歌以惊人的速度不断地更新Android系统，2.1、2.2、2.3系统的推出使Android占据了大量的市场。2011年2月，谷歌发布了Android 3.0系统，这个系统版本是专门为平板电脑设计的，但也是Android为数不多的比较失败的版本，推出之后一直不见什么起色，市场份额也少得可怜。不过很快，在同年的10月，谷歌又发布了Android 4.0系统，这个版本不再对手机和平板进行差异化区分，既可以应用在手机上，也可以应用在平板上。2014年Google I/O大会上，谷歌推出了号称史上版本改动最大的Android 5.0系统，其中使用ART运行环境替代了Dalvik虚拟机，大大提升了应用的运行速度，还提出了Material Design的概念来优化应用的界面设计。除此之外，还推出了Android Wear、Android Auto、Android TV系统，从而进军可穿戴设备、汽车、电视等全新领域。之后Android的更新速度更加迅速，2015年Google I/O大会上推出了Android 6.0系统，加入运行时权限功能，2016年Google I/O大会上推出了Android 7.0系统，加入多窗口模式功能，这也是目前最新的Android系统版本。

下表中列出了目前市场上主要的Android系统版本及其详细信息。你看到这张表格时，数据很可能已经发生了变化，查看最新的数据可以访问 <http://developer.android.com/about/dashboards/>。

版本号	系统代号	API	市场占有率
2.2	Froyo	8	0.1%
2.3.3 – 2.3.7	Gingerbread	10	1.5%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	1.3%
4.1.x		16	5.6%
4.2.x	Jelly Bean	17	7.7%
4.3		18	2.3%

(续)

版本号	系统代号	API	市场占有率
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%
7.0	Nougat	24	0.1%

从上表中可以看出，目前4.0以上的系统已经占据了超过98%的Android市场份额，因此我们本书中开发的程序也只面向4.0以上的系统，2.x的系统就不再去兼容了。

1.1.3 Android应用开发特色

预告一下，你马上就要开始真正的Android开发旅程了。不过先别急，在开始之前我们再一起看一看，Android系统到底提供了哪些东西，可供我们开发出优秀的应用程序。

1. 四大组件

Android系统四大组件分别是活动（Activity）、服务（Service）、广播接收器（Broadcast Receiver）和内容提供器（Content Provider）。其中活动是所有Android应用程序的门面，凡是在应用中你看得到的东西，都是放在活动中的。而服务就比较低调了，你无法看到它，但它会一直在后台默默地运行，即使用户退出了应用，服务仍然是可以继续运行的。广播接收器允许你的应用接收来自各处的广播消息，比如电话、短信等，当然你的应用同样也可以向外发出广播消息。内容提供器则为应用程序之间共享数据提供了可能，比如你想要读取系统电话簿中的联系人，就需要通过内容提供器来实现。

2. 丰富的系统控件

Android系统为开发者提供了丰富的系统控件，使得我们可以很轻松地编写出漂亮的界面。当然如果你品位比较高，不满足于系统自带的控件效果，也完全可以定制属于自己的控件。

3. SQLite数据库

Android系统还自带了这种轻量级、运算速度极快的嵌入式关系型数据库。它不仅支持标准的SQL语法，还可以通过Android封装好的API进行操作，让存储和读取数据变得非常方便。

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，等等，这一切你都可以在程序中通过代码进行控制，让你的应用变得更加丰富多彩。

5. 地理位置定位

移动设备和PC相比起来，地理位置定位功能应该可以算是很大的一个亮点。现在的Android手机都内置有GPS，走到哪儿都可以定位到自己的位置，发挥你的想象就可以做出创意十足的应

用，如果再结合功能强大的地图功能，LBS 这一领域潜力无限。

既然有 Android 这样出色的系统给我们提供了这么丰富的工具，你还用担心做不出优秀的应用吗？好了，纯理论的东西就介绍到这里，我知道你已经迫不及待想要开始真正的开发之旅了，那我们就开始启程吧！

1.2 手把手带你搭建环境

俗话说得好，“工欲善其事，必先利其器”，开着记事本就想去开发 Android 程序显然不是明智之举，选择一个好的 IDE 可以极大地提高你的开发效率，因此本节我就将手把手带着你把开发环境搭建起来。

1.2.1 准备所需要的工具

我现在对你了解还并不多，但我希望你已经是一个颇有经验的 Java 程序员，这样你理解本书的内容时将会轻而易举，因为 Android 程序都是使用 Java 语言编写的。如果你对 Java 只是略有了解，那阅读本书应该会有一点困难，不过一边阅读一边补充 Java 知识也是可以的。但如果你对 Java 完全没有了解，那么我建议你可以暂时将本书放下，先买本介绍 Java 基础知识的书学上两个星期，把 Java 的基本语法和特性都学会了，再来继续阅读这本书。

好了，既然你已经阅读到这里，说明你已经掌握 Java 的基本用法了，下面我们就来看一看开发 Android 程序需要准备哪些工具。

- **JDK**。JDK 是 Java 语言的软件开发工具包，它包含了 Java 的运行环境、工具集合、基础类库等内容。需要注意的是，本书中的 Android 程序必须要使用 JDK 8 或以上版本才能进行开发。
- **Android SDK**。Android SDK 是谷歌提供的 Android 开发工具包，在开发 Android 程序时，我们需要通过引入该工具包，来使用 Android 相关的 API。
- **Android Studio**。在很早之前，Android 项目都是用 Eclipse 来开发的，相信所有 Java 开发者都一定会对这个工具非常熟悉，它是 Java 开发神器，安装 ADT 插件后就可以用来开发 Android 程序了。而在 2013 年的时候，谷歌推出了一款官方的 IDE 工具 Android Studio，由于不再是以插件的形式存在，Android Studio 在开发 Android 程序方面要远比 Eclipse 强大和方便得多。不过由于 Android Studio 早期的测试版本并不是非常稳定，所以本书的第一版仍然选用的 Eclipse 来作为开发工具。而如今，Android Studio 已经推出了 2.2 版本，稳定性完全不再是问题，普及程度方面也远超 Eclipse，没有比现在更适合的时机来换用 Android Studio 了，因此本书中所有的代码都将在 Android Studio 上进行开发。

1.2.2 搭建开发环境

当然，上述软件并不需要你去一个个地下载，因为谷歌为了简化搭建开发环境的过程，将所

有需要用到的工具都帮我们集成好了，到Android官网就可以下载最新的开发工具，下载地址是：<https://developer.android.com/studio/index.html>。不过，Android官网通常都需要科学上网才能访问，如果你无法访问的话，也可以直接到我的百度网盘去下载，下载地址是：<https://pan.baidu.com/s/1nuABMDb>。（注意网址中是阿拉伯数字1，而不是英文字母1。）

你下载下来的将是一个安装包，安装的过程也很简单，一直点击Next就可以了。其中选择安装组件时建议全部勾上，如图1.2所示。

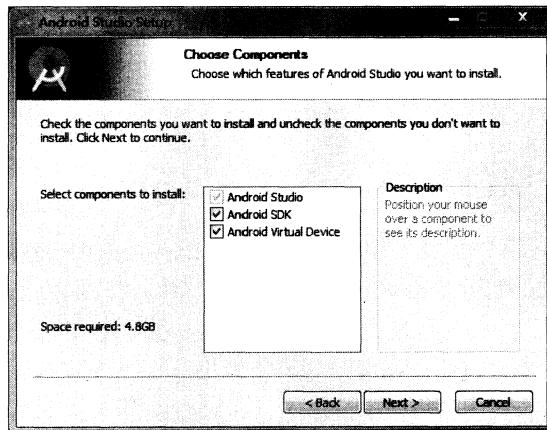


图1.2 选择安装组件

接下来还会让你选择Android Studio的安装地址以及Android SDK的安装地址，这些根据你自己电脑的实际情况选择就行了，不想改动的话就保持默认，如图1.3所示。

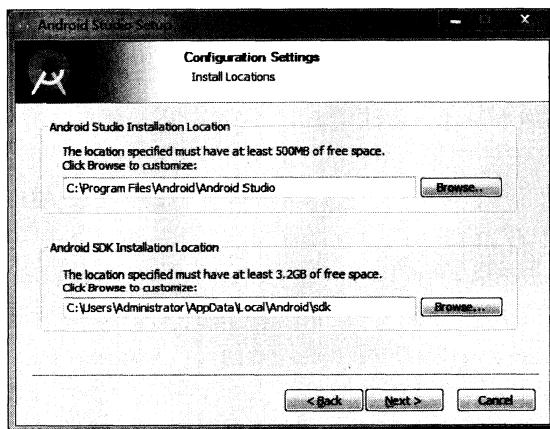


图1.3 选择安装地址

后面就没什么需要注意的了，全部保持默认项，一直点击Next即可完成安装，如图1.4所示。



图 1.4 安装完成

现在点击 Finish 按钮来启动 Android Studio，一开始会让你选择是否导入之前 Android Studio 版本的配置，由于这是我们首次安装，这里选择不导入就可以了，如图 1.5 所示。

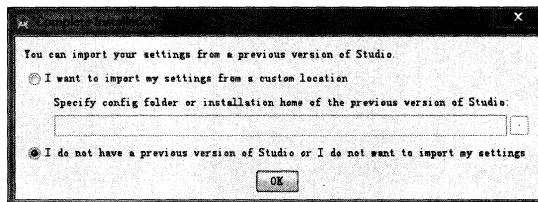


图 1.5 选择不导入配置

点击 OK 按钮会进入到 Android Studio 的配置界面，如图 1.6 所示。



图 1.6 Android Studio 的配置界面

然后点击 Next 开始进行具体的配置，如图 1.7 所示。

这里我们可以选择Android Studio的安装类型，有Standard和Custom两种。Standard表示一切都使用默认的配置，比较方便；Custom则可以根据用户的特殊需求进行自定义。简单起见，这里我们就选择Standard类型了，继续点击Next完成配置工作，如图1.8所示。



图1.7 选择安装类型



图1.8 完成Android Studio配置

现在点击Finish按钮，配置工作就全部完成了。然后Android Studio会尝试联网下载一些更新，等待更新完成后再点击Finish按钮就会进入Android Studio的欢迎界面，如图1.9所示。

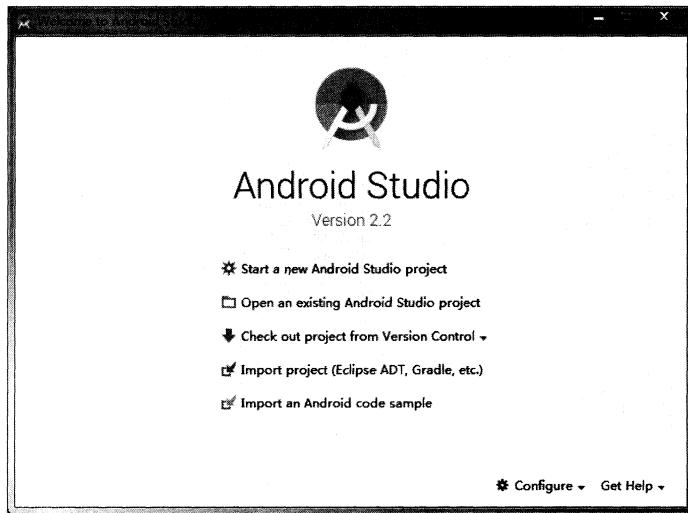


图1.9 Android Studio的欢迎界面

目前为止，Android开发环境就已经全部搭建完成了。那现在应该做什么？当然是写下你的第一行Android代码了，让我们快点开始吧。

1.3 创建你的第一个 Android 项目

任何一个编程语言写出的第一个程序毫无疑问都会是 Hello World，这已经是自 20 世纪 70 年代一直流传下来的传统，在编程界已成为永恒的经典，那我们当然也不会搞例外了。

1.3.1 创建 HelloWorld 项目

在 Android Studio 的欢迎界面点击 Start a new Android Studio project，会打开一个创建新项目的界面，如图 1.10 所示。



图 1.10 创建新项目

其中 Application name 表示应用名称，此应用安装到手机之后会在手机上显示该名称，这里我们填入 HelloWorld。Company Domain 表示公司域名，如果是个人开发者，没有公司域名的话，那么就像我一样填 example.com 就可以了。Package name 表示项目的包名，Android 系统就是通过包名来区分不同应用程序的，因此包名一定要具有唯一性。Android Studio 会根据应用名称和公司域名来自动帮我们生成合适的包名，如果你不想使用默认生成的包名，也可以点击右侧的 Edit 按钮自行修改。最后，Project location 表示项目代码存放的位置，如果没有特殊要求的话，这里也保持默认就可以了。

接下来点击 Next 可以对项目的最低兼容版本进行设置，如图 1.11 所示。



图 1.11 设置项目的最低兼容版本

前面已经说过，Android 4.0 以上的系统已经占据了超过 98% 的 Android 市场份额，因此这里我们将 Minimum SDK 指定成 API 15 就可以了。另外，Wear、TV 和 Android Auto 这几个选项分别是用于开发可穿戴设备、电视和汽车程序的，目前这几个领域在国内还没有普及，我们暂时就先忽略吧。接着点击 Next 会跳转到创建活动界面，这里我们可以选择一种模板，如图 1.12 所示。

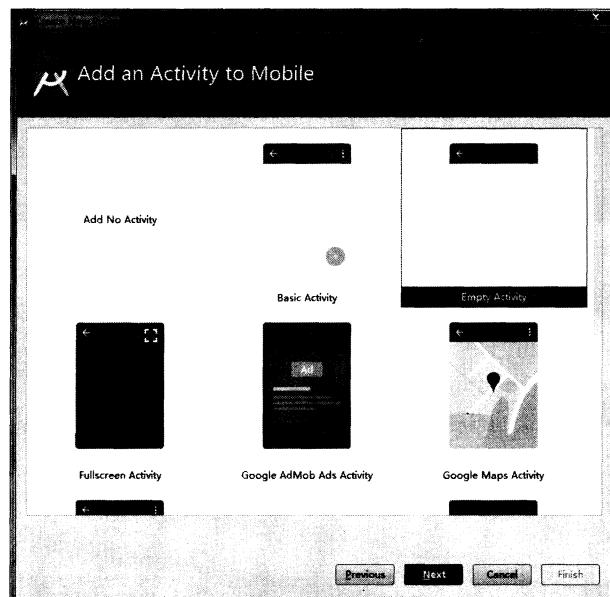


图 1.12 选择模板

可以看到，Android Studio 提供了很多种内置模板，不过由于我们才刚刚开始学习，用不着这么多复杂的模板，这里直接选择 Empty Activity 来创建一个空的活动就可以了。

继续点击 Next，可以给创建的活动和布局命名，如图 1.13 所示。



图 1.13 给活动和布局命名

其中，Activity Name 表示活动的名字，这里填入 HelloWorldActivity，Layout Name 表示布局的命名，这里填入 hello_world_layout。然后点击 Finish 按钮，并耐心等待一会儿，项目就会创建成功了，如图 1.14 所示。

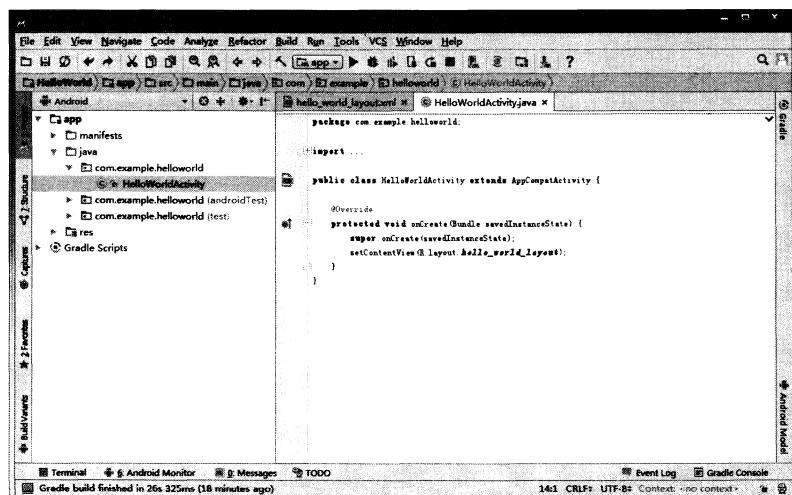


图 1.14 项目创建成功

1.3.2 启动模拟器

由于Android Studio自动为我们生成了很多东西，你现在不需要编写任何代码，HelloWorld项目就已经可以运行了。但是在此之前还必须要有一个运行的载体，可以是一部Android手机，也可以是Android模拟器。这里我们暂时先使用模拟器来运行程序，如果你想立刻就将程序运行到手机上的话，可以参考8.1节的内容。

那么我们现在就来创建一个Android模拟器，观察Android Studio顶部工具栏中的图标，如图1.15所示。



图1.15 顶部工具栏中的图标

其中，最左边的按钮就是用于创建和启动模拟器的，点击该按钮，会弹出如图1.16所示的窗口。

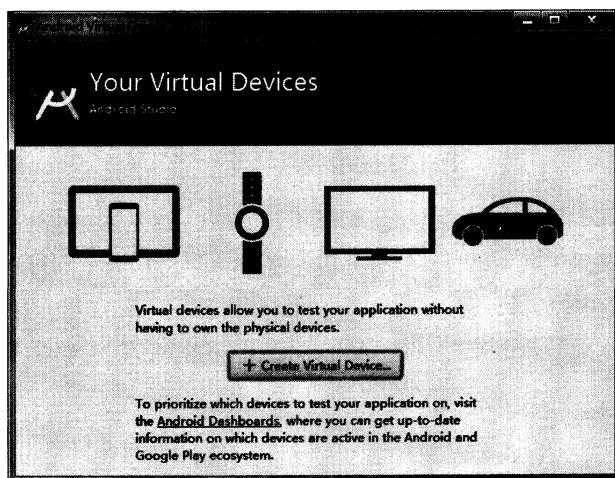


图1.16 创建模拟器

可以看到，目前我们的模拟器列表中还是空的，点击Create Virtual Device按钮就可以立刻开始创建了，如图1.17所示。



图 1.17 选择要创建的模拟器设备

这里有很多种设备可供我们选择，不仅能创建手机模拟器，还可以创建平板、手表、电视等模拟器。

那么我就选择创建 Nexus 5X 这台设备的模拟器了，点击 Next，如图 1.18 所示。

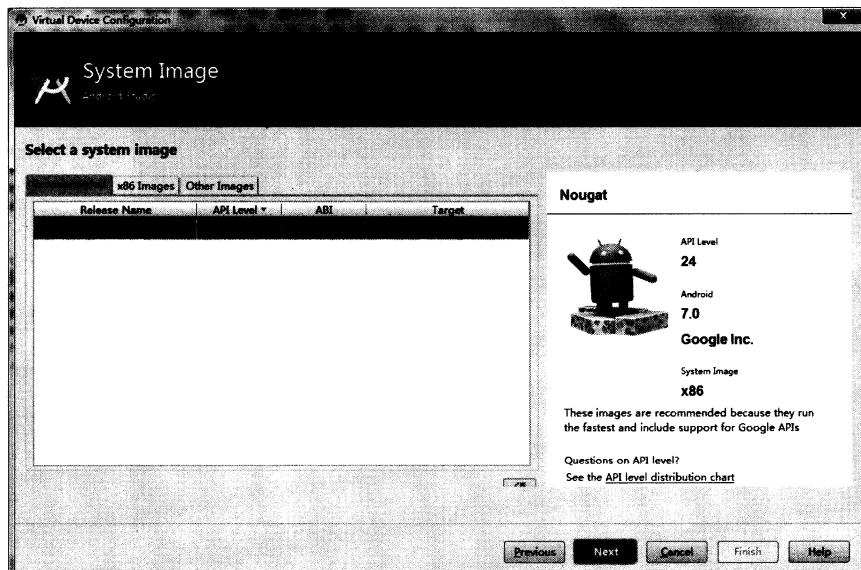


图 1.18 选择模拟器的操作系统版本

这里可以选择模拟器所使用的操作系统版本，毫无疑问，我们肯定要选择最新的Android 7.0系统。继续点击Next，如图1.19所示。

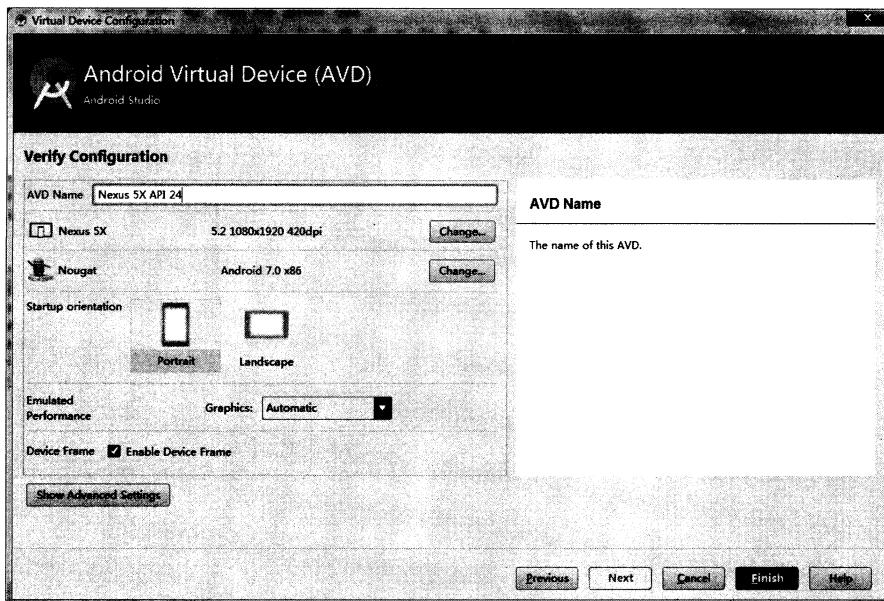


图1.19 确认模拟器配置

在这里我们可以对模拟器的一些配置进行确认，比如说指定模拟器的名字、分辨率、横竖屏等信息，如果没有特殊需求的话，全部保持默认就可以了。点击Finish完成模拟器的创建，然后会弹出如图1.20所示的窗口。

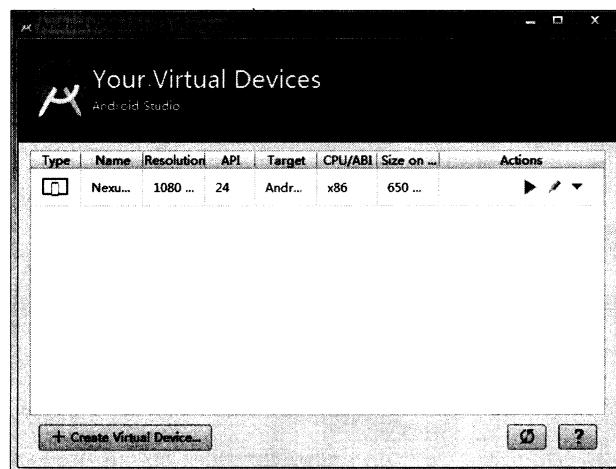


图1.20 模拟器列表

可以看到，现在模拟器列表中已经存在一个创建好的模拟器设备了，点击 Actions 栏目中最左边的三角形按钮即可启动模拟器。模拟器会像手机一样，有一个开机过程，启动完成之后的界面如图 1.21 所示。

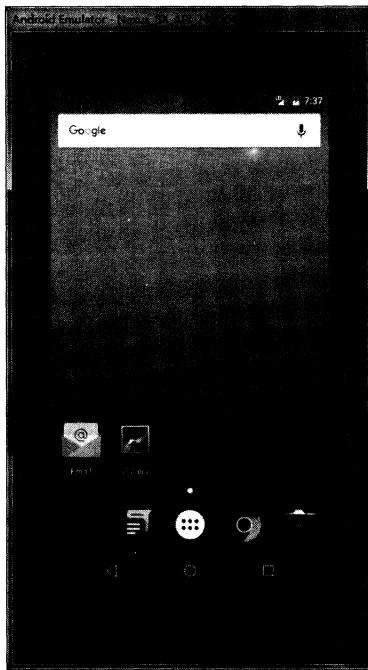


图 1.21 启动后的模拟器界面

很清新的 Android 界面出来了！看上去还挺不错吧，你几乎可以像使用手机一样使用它，Android 模拟器对手机的模仿度非常高，快去体验一下吧。

1.3.3 运行 HelloWorld

现在模拟器已经启动起来了，那么下面我们就开始将 HelloWorld 项目运行到模拟器上。观察 Android Studio 顶部工具栏中的图标，如图 1.22 所示。其中左边的锤子按钮是用来编译项目的，中间的下拉列表是用来选择运行哪一个项目的，通常 app 就是当前的主项目，右边的三角形按钮是用来运行项目的。

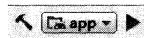


图 1.22 顶部工具栏中的图标

现在点击右边的运行按钮，会弹出一个选择运行设备的对话框，如图 1.23 所示。



图 1.23 选择运行设备对话框

可以看到，我们刚刚创建的模拟器现在是在线的，点击 OK 按钮，稍微等待一会儿，HelloWorld 项目就会运行到模拟器上了，结果应该和图 1.24 中显示的是一样的。

HelloWorld 项目运行成功！并且你会发现，模拟器上已经安装上 HelloWorld 这个应用了。打开启动器列表，如图 1.25 所示。



图 1.24 运行 HelloWorld 项目



图 1.25 查看启动器列表

这个时候你可能会说我坑你了，说好的第一行代码呢？怎么一行还没写，项目就已经运行起来了？这个只能说是因为 Android Studio 太智能了，已经帮我们把一些简单内容都自动生成了。你也别心急，后面写代码的机会多着呢，我们先来分析一下 HelloWorld 这个项目吧。

1.3.4 分析你的第一个 Android 程序

回到 Android Studio 当中，首先展开 HelloWorld 项目，你会看到如图 1.26 所示的项目结构。

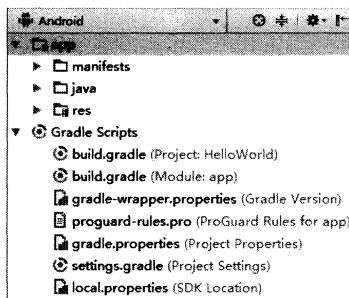


图 1.26 Android 模式的项目结构

任何一个新建的项目都会默认使用 Android 模式的项目结构，但这并不是项目真实的目录结构，而是被 Android Studio 转换过的。这种项目结构简洁明了，适合进行快速开发，但是对于新手来说可能并不易于理解。点击图 1.26 当中的 Android 区域可以切换项目结构模式，如图 1.27 所示。

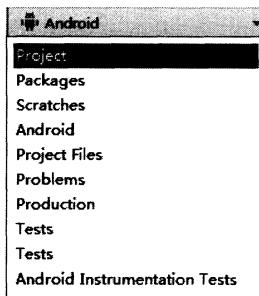


图 1.27 切换项目结构模式

这里我们将项目结构模式切换成 Project，这就是项目真实的目录结构了，如图 1.28 所示。

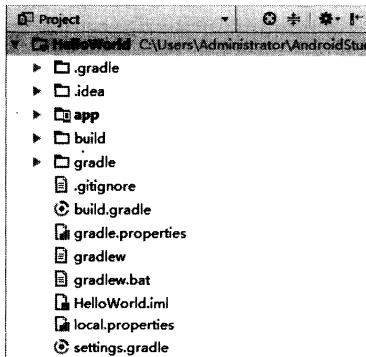


图 1.28 Project 模式的项目结构

一开始看到这么多陌生的东西，你一定会感到有点头晕吧。别担心，我现在就对图 1.28 中的内容进行一一讲解，之后你再看这张图就不会感到那么吃力了。

1. .gradle 和.idea

这两个目录下放置的都是 Android Studio 自动生成的一些文件，我们无须关心，也不要去做手动编辑。

2. app

项目中的代码、资源等内容几乎都是放置在这个目录下的，我们后面的开发工作也基本都是在这个目录下进行的，待会儿还会对这个目录单独展开进行讲解。

3. build

这个目录你也不需要过多关心，它主要包含了一些在编译时自动生成的文件。

4. gradle

这个目录下包含了 gradle wrapper 的配置文件，使用 gradle wrapper 的方式不需要提前将 gradle 下载好，而是会自动根据本地的缓存情况决定是否需要联网下载 gradle。Android Studio 默认没有启用 gradle wrapper 的方式，如果需要打开，可以点击 Android Studio 导航栏→File→Settings→Build, Execution, Deployment→Gradle，进行配置更改。

5. .gitignore

这个文件是用来将指定的目录或文件排除在版本控制之外的，关于版本控制我们将在第 5 章中开始正式的学习。

6. build.gradle

这是项目全局的 gradle 构建脚本，通常这个文件中的内容是不需要修改的。稍后我们将会详细分析 gradle 构建脚本中的具体内容。

7. gradle.properties

这个文件是全局的 gradle 配置文件，在这里配置的属性将会影响到项目中所有的 gradle 编译脚本。

8. gradlew 和 gradlew.bat

这两个文件是用来在命令行界面中执行 gradle 命令的，其中 gradlew 是在 Linux 或 Mac 系统中使用的，gradlew.bat 是在 Windows 系统中使用的。

9. HelloWorld.iml

iml 文件是所有 IntelliJ IDEA 项目都会自动生成的一个文件（Android Studio 是基于 IntelliJ IDEA 开发的），用于标识这是一个 IntelliJ IDEA 项目，我们不需要修改这个文件中的任何内容。

10. local.properties

这个文件用于指定本机中的 Android SDK 路径，通常内容都是自动生成的，我们并不需要修改。除非你本机中的 Android SDK 位置发生了变化，那么就将这个文件中的路径改成新的位置即可。

11. settings.gradle

这个文件用于指定项目中所有引入的模块。由于 HelloWorld 项目中就只有一个 app 模块，因此该文件中也就只引入了 app 这一个模块。通常情况下模块的引入都是自动完成的，需要我们手动去修改这个文件的场景可能比较少。

现在整个项目的外层目录结构已经介绍完了。你会发现，除了 app 目录之外，大多数的文件和目录都是自动生成的，我们并不需要进行修改。想必你已经猜到了，app 目录下的内容才是我们以后的工作重点，展开之后结构如图 1.29 所示。

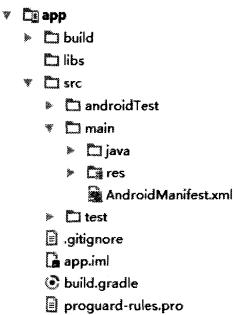


图 1.29 app 目录下的结构

那么下面我们就来对 app 目录下的内容进行更为详细的分析。

1. build

这个目录和外层的 build 目录类似，主要也是包含了一些在编译时自动生成的文件，不过它里面的内容会更多更杂，我们不需要过多关心。

2. libs

如果你的项目中使用到了第三方 jar 包，就需要把这些 jar 包都放在 libs 目录下，放在这个目录下的 jar 包都会被自动添加到构建路径里去。

3. androidTest

此处是用来编写 Android Test 测试用例的，可以对项目进行一些自动化测试。

4. java

毫无疑问，java 目录是放置我们所有 Java 代码的地方，展开该目录，你将看到我们刚才创建的 HelloWorldActivity 文件就在里面。

5. res

这个目录下的内容就有点多了。简单点说，就是你在项目中使用到的所有图片、布局、字符串等资源都要存放在这个目录下。当然这个目录下还有很多子目录，图片放在 drawable 目录下，布局放在 layout 目录下，字符串放在 values 目录下，所以你不用担心会把整个 res 目录弄得乱糟糟的。

6. AndroidManifest.xml

这是你整个Android项目的配置文件，你在程序中定义的所有四大组件都需要在这个文件里注册，另外还可以在这个文件中给应用程序添加权限声明。由于这个文件以后会经常用到，我们用到的时候再做详细说明。

7. test

此处是用来编写Unit Test测试用例的，是对项目进行自动化测试的另一种方式。

8. .gitignore

这个文件用于将app模块内的指定的目录或文件排除在版本控制之外，作用和外层的.gitignore文件类似。

9. app.iml

IntelliJ IDEA项目自动生成的文件，我们不需要关心或修改这个文件中的内容。

10. build.gradle

这是app模块的gradle构建脚本，这个文件中会指定很多项目构建相关的配置，我们稍后将会详细分析gradle构建脚本中的具体内容。

11. proguard-rules.pro

这个文件用于指定项目代码的混淆规则，当代码开发完成后打成安装包文件，如果不希望代码被别人破解，通常会将代码进行混淆，从而让破解者难以阅读。

这样整个项目的目录结构就都介绍完了，如果你还不能完全理解的话也很正常，毕竟里面有太多的东西你都还没接触过。不过不用担心，这并不会影响到你后面的学习。等你学完整本书再回来看这个目录结构图时，你会觉得特别地清晰和简单。

接下来我们一起分析一下HelloWorld项目究竟是怎么运行起来的吧。首先打开AndroidManifest.xml文件，从中可以找到如下代码：

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

这段代码表示对HelloWorldActivity这个活动进行注册，没有在AndroidManifest.xml里注册的活动是不能使用的。其中intent-filter里的两行代码非常重要，<action android:name="android.intent.action.MAIN" />和<category android:name="android.intent.category.LAUNCHER" />表示HelloWorldActivity是这个项目的主活动，在手机上点击应用图标，首先启动的就是这个活动。

那HelloWorldActivity具体又有什么作用呢？我在介绍Android四大组件的时候说过，活动

是 Android 应用程序的门面，凡是在应用中你看得到的东西，都是放在活动中的。因此你在图 1.24 中看到的界面，其实就是 HelloWorldActivity 这个活动。那我们快去看一下它的代码吧，打开 HelloWorldActivity，代码如下所示：

```
public class HelloWorldActivity extends AppCompatActivity {

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

}
```

首先我们可以看到，HelloWorldActivity 是继承自 AppCompatActivity 的，这是一种向下兼容的 Activity，可以将 Activity 在各个系统版本中增加的特性和功能最低兼容到 Android 2.1 系统。Activity 是 Android 系统提供的一个活动基类，我们项目中所有的活动都必须继承它或者它的子类才能拥有活动的特性（AppCompatActivity 是 Activity 的子类）。然后可以看到 HelloWorldActivity 中有一个 onCreate() 方法，这个方法是一个活动被创建时必定要执行的方法，其中只有两行代码，并且没有 Hello World! 的字样。那么图 1.24 中显示的 Hello World! 是在哪里定义的呢？

其实 Android 程序的设计讲究逻辑和视图分离，因此是不推荐在活动中直接编写界面的，更加通用的一种做法是，在布局文件中编写界面，然后在活动中引入进来。可以看到，在 onCreate() 方法的第二行调用了 setContentView() 方法，就是这个方法给当前的活动引入了一个 hello_world_layout 布局，那 Hello World! 一定就是在里面定义的了！我们快打开这个文件看一看。

布局文件都是定义在 res/layout 目录下的，当你展开 layout 目录，你会看到 hello_world_layout.xml 这个文件。打开该文件并切换到 Text 视图，代码如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/hello_world_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.helloworld.HelloWorldActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

现在还看不懂？没关系，后面我会对布局进行详细讲解的，你现在只需要看到上面代码中有

一个 TextView，这是 Android 系统提供的一个控件，用于在布局中显示文字的。然后你终于在 TextView 中看到了 Hello World! 的字样！哈哈！终于找到了，原来就是通过 `android:text="Hello World!"` 这句代码定义的。

这样我们就将 HelloWorld 项目的目录结构以及基本的执行过程都分析完了，相信你对 Android 项目已经有了一个初步的认识，下一小节中我们就来学习一下项目中所包含的资源。

1.3.5 详解项目中的资源

如果你展开 res 目录看一下，其实里面的东西还是挺多的，很容易让人看得眼花缭乱，如图 1.30 所示。



图 1.30 res 目录下的结构

看到这么多的文件夹也不用害怕，其实归纳一下，res 目录就变得非常简单了。所有以 `drawable` 开头的文件夹都是用来放图片的，所有以 `mipmap` 开头的文件夹都是用来放应用图标的，所有以 `values` 开头的文件夹都是用来放字符串、样式、颜色等配置的，`layout` 文件夹是用来放布局文件的。怎么样，是不是突然感觉清晰了很多？

之所以有这么多 `mipmap` 开头的文件夹，其实主要是为了让程序能够更好地兼容各种设备。`drawable` 文件夹也是相同的道理，虽然 Android Studio 没有帮我们自动生成，但是我们应该自己创建 `drawable-hdpi`、`drawable-xhdpi`、`drawable-xxhdpi` 等文件夹。在制作程序的时候最好能够给同一张图片提供几个不同分辨率的版本，分别放在这些文件夹下，然后当程序运行的时候，会自动根据当前运行设备分辨率的高低选择加载哪个文件夹下的图片。当然这只是理想情况，更多的时候美工只会提供给我们一份图片，这时你就把所有图片都放在 `drawable-xxhdpi` 文件夹下就好了。

知道了 res 目录下每个文件夹的含义，我们再来看一下如何去使用这些资源吧。打开 `res/values/strings.xml` 文件，内容如下所示：

```

<resources>
    <string name="app_name">HelloWorld</string>
</resources>
  
```

可以看到，这里定义了一个应用程序名的字符串，我们有以下两种方式来引用它。

- 在代码中通过 `R.string.hello_world` 可以获得该字符串的引用。
- 在 XML 中通过 `@string/hello_world` 可以获得该字符串的引用。

基本的语法就是上面这两种方式，其中 `string` 部分是可以替换的，如果是引用的图片资源就可以替换成 `drawable`，如果是引用的应用图标就可以替换成 `mipmap`，如果是引用的布局文件就可以替换成 `layout`，以此类推。

下面举一个简单的例子来帮助你理解，打开 `AndroidManifest.xml` 文件，找到如下代码：

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    ...
</application>
```

其中，`HelloWorld` 项目应用图标就是通过 `android:icon` 属性来指定的，应用的名称则是通过 `android:label` 属性指定的。可以看到，这里对资源引用的方式正是我们刚刚学过的在 XML 中引用资源的语法。

经过本小节的学习，如果你想修改应用的图标或者名称，相信已经知道该怎么办了吧。

1.3.6 详解 `build.gradle` 文件

不同于 `Eclipse`，`Android Studio` 是采用 `Gradle` 来构建项目的。`Gradle` 是一个非常先进的项目构建工具，它使用了一种基于 `Groovy` 的领域特定语言（DSL）来声明项目设置，摒弃了传统基于 `XML`（如 `Ant` 和 `Maven`）的各种烦琐配置。

在 1.3.4 小节中我们已经看到，`HelloWorld` 项目中有两个 `build.gradle` 文件，一个是在最外层目录下的，一个是在 `app` 目录下的。这两个文件对构建 `Android Studio` 项目都起到了至关重要的作用，下面我们就来对这两个文件中的内容进行详细的分析。

先来看一下最外层目录下的 `build.gradle` 文件，代码如下所示：

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.0'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

这些代码都是自动生成的，虽然语法结构看上去可能有点难以理解，但是如果我们忽略语法结构，只看最关键的部分，其实还是很好懂的。

首先，两处 `repositories` 的闭包中都声明了 `jcenter()` 这行配置，那么这个 `jcenter` 是什么意思呢？其实它是一个代码托管仓库，很多 Android 开源项目都会选择将代码托管到 `jcenter` 上，声明了这行配置之后，我们就可以在项目中轻松引用任何 `jcenter` 上的开源项目了。

接下来，`dependencies` 闭包中使用 `classpath` 声明了一个 Gradle 插件。为什么要声明这个插件呢？因为 Gradle 并不是专门为构建 Android 项目而开发的，Java、C++ 等很多种项目都可以使用 Gradle 来构建。因此如果我们要想使用它来构建 Android 项目，则需要声明 `com.android.tools.build:gradle:2.2.0` 这个插件。其中，最后面的部分是插件的版本号，我在写作本书时最新的插件版本是 2.2.0。

这样我们就将最外层目录下的 `build.gradle` 文件分析完了，通常情况下你并不需要修改这个文件中的内容，除非你想添加一些全局的项目构建配置。

下面我们再来看一下 `app` 目录下的 `build.gradle` 文件，代码如下所示：

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.example.helloworld"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

这个文件中的内容就要相对复杂一些了，下面我们一行行地进行分析。首先第一行应用了一个插件，一般有两种值可选：`com.android.application` 表示这是一个应用程序模块，`com.android.library` 表示这是一个库模块。应用程序模块和库模块的最大区别在于，一个是可以直接运行的，一个只能作为代码库依附于别的应用程序模块来运行。

接下来是一个大的 `android` 闭包，在这个闭包中我们可以配置项目构建的各种属性。其中，`compileSdkVersion` 用于指定项目的编译版本，这里指定成 24 表示使用 Android 7.0 系统的 SDK 编译。`buildToolsVersion` 用于指定项目构建工具的版本，目前最新的版本就是 24.0.2，如果

有更新的版本时，Android Studio 会进行提示。

然后我们看到，这里在 `android` 闭包中又嵌套了一个 `defaultConfig` 闭包，`defaultConfig` 闭包中可以对项目的更多细节进行配置。其中，`applicationId` 用于指定项目的包名，前面我们在创建项目的时候其实已经指定过包名了，如果你想在后面对其进行修改，那么就是在这里修改的。`minSdkVersion` 用于指定项目最低兼容的 Android 系统版本，这里指定成 15 表示最低兼容到 Android 4.0 系统。`targetSdkVersion` 指定的值表示你在该目标版本上已经做过了充分的测试，系统将会为你的应用程序启用一些最新的功能和特性。比如说 Android 6.0 系统中引入了运行时权限这个功能，如果你将 `targetSdkVersion` 指定成 23 或者更高，那么系统就会为你的程序启用运行时权限功能，而如果你将 `targetSdkVersion` 指定成 22，那么就说明你的程序最高只在 Android 5.1 系统上做过充分的测试，Android 6.0 系统中引入的新功能自然就不会启用了。剩下的两个属性都比较简单，`versionCode` 用于指定项目的版本号，`versionName` 用于指定项目的版本名，这两个属性在生成安装文件的时候非常重要，我们在后面都会学到。

分析完了 `defaultConfig` 闭包，接下来我们看一下 `buildTypes` 闭包。`buildTypes` 闭包中用于指定生成安装文件的相关配置，通常只会有两个子闭包，一个是 `debug`，一个是 `release`。`debug` 闭包用于指定生成测试版安装文件的配置，`release` 闭包用于指定生成正式版安装文件的配置。另外，`debug` 闭包是可以忽略不写的，因此我们看到上面的代码中就只有一个 `release` 闭包。下面来看一下 `release` 闭包中的具体内容吧，`minifyEnabled` 用于指定是否对项目的代码进行混淆，`true` 表示混淆，`false` 表示不混淆。`proguardFiles` 用于指定混淆时使用的规则文件，这里指定了两个文件，第一个 `proguard-android.txt` 是在 Android SDK 目录下的，里面是所有项目通用的混淆规则，第二个 `proguard-rules.pro` 是在当前项目的根目录下的，里面可以编写当前项目特有的混淆规则。需要注意的是，通过 Android Studio 直接运行项目生成的都是测试版安装文件，关于如何生成正式版安装文件我们将会在第 15 章中学习。

这样整个 `android` 闭包中的内容就都分析完了，接下来还剩一个 `dependencies` 闭包。这个闭包的功能非常强大，它可以指定当前项目所有的依赖关系。通常 Android Studio 项目一共有 3 种依赖方式：本地依赖、库依赖和远程依赖。本地依赖可以对本地的 Jar 包或目录添加依赖关系，库依赖可以对项目中的库模块添加依赖关系，远程依赖则可以对 jcenter 库上的开源项目添加依赖关系。观察一下 `dependencies` 闭包中的配置，第一行的 `compile fileTree` 就是一个本地依赖声明，它表示将 `libs` 目录下所有 `.jar` 后缀的文件都添加到项目的构建路径当中。而第二行的 `compile` 则是远程依赖声明，`com.android.support:appcompat-v7:24.2.1` 就是一个标准的远程依赖库格式，其中 `com.android.support` 是域名部分，用于和其他公司的库做区分；`appcompat-v7` 是组名称，用于和同一个公司中不同的库做区分；`24.2.1` 是版本号，用于和同一个库不同的版本做区分。加上这句声明后，Gradle 在构建项目时会首先检查一下本地是否已经有这个库的缓存，如果没有的话则会去自动联网下载，然后再添加到项目的构建路径当中。至于库依赖声明这里没有用到，它的基本格式是 `compile project` 后面加上要依赖的库名称，比如说有一个库模块的名字叫 `helper`，那么添加这个库的依赖关系只需要加入 `compile project(':helper')` 这句声明即可。另外剩下

的一句 `testCompile` 是用于声明测试用例库的，这个我们暂时用不到，先忽略它就可以了。

1.4 前行必备——掌握日志工具的使用

通过上一节的学习，你已经成功创建了你的第一个Android程序，并且对Android项目的目录结构和运行流程都有了一定的了解。现在本应该是你继续前行的时候，不过我想在这里给你穿插一点内容，讲解一下Android中日志工具的使用方法，这对你以后的Android开发之旅会有极大的帮助。

1.4.1 使用Android的日志工具Log

Android中的日志工具类是 `Log` (`android.util.Log`)，这个类中提供了如下5个方法来供我们打印日志。

- `Log.v()`。用于打印那些最为琐碎的、意义最小的日志信息。对应级别 `verbose`，是Android日志里面级别最低的一种。
- `Log.d()`。用于打印一些调试信息，这些信息对你调试程序和分析问题应该是有帮助的。对应级别 `debug`，比 `verbose` 高一级。
- `Log.i()`。用于打印一些比较重要的数据，这些数据应该是你非常想看到的、可以帮你分析用户行为数据。对应级别 `info`，比 `debug` 高一级。
- `Log.w()`。用于打印一些警告信息，提示程序在这个地方可能会有潜在的风险，最好去修复一下这些出现警告的地方。对应级别 `warn`，比 `info` 高一级。
- `Log.e()`。用于打印程序中的错误信息，比如程序进入到了 `catch` 语句当中。当有错误信息打印出来的时候，一般都代表你的程序出现严重问题了，必须尽快修复。对应级别 `error`，比 `warn` 高一级。

其实很简单，一共就5个方法，当然每个方法还会有不同的重载，但那对你来说肯定不是什么难理解的地方了。我们现在就在HelloWorld项目中试一试日志工具好不好用吧。

打开HelloWorldActivity，在`onCreate()`方法中添加一行打印日志的语句，如下所示：

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.hello_world_layout);  
    Log.d("HelloWorldActivity", "onCreate execute");  
}
```

`Log.d()`方法中传入了两个参数：第一个参数是 `tag`，一般传入当前的类名就好，主要用于对打印信息进行过滤；第二个参数是 `msg`，即想要打印的具体的内容。

现在可以重新运行一下HelloWorld这个项目了，点击顶部工具栏上的运行按钮，或者使用快捷键 Shift + F10 (Mac系统是 control + R)，等程序运行完毕，点击Android Studio底部工具栏的Android Monitor，在logcat中就可以看到打印信息了，如图1.31所示。



图 1.31 logcat 中的打印信息

其中，你不仅可以看到打印日志的内容和 tag 名，就连程序的包名、打印的时间以及应用程序的进程号都可以看到。

另外，不知道你有没有注意到，你的第一行代码已经在不知不觉中写出来了，我也总算是交差了。

1.4.2 为什么使用 Log 而不使用 System.out

我相信很多的 Java 新手都非常喜欢使用 `System.out.println()` 方法来打印日志，不知道你是不是也喜欢这么做。不过在真正的项目开发中，是极度不建议使用 `System.out.println()` 方法的！如果你在公司的项目中经常使用这个方法，就很有可能要挨骂了。

为什么 `System.out.println()` 方法会这么遭大家唾弃呢？经过我仔细分析之后，发现这个方法除了使用方便一点之外，其他就一无是处了。方便在哪儿呢？在 Eclipse 中你只需要输入 `sys`，然后按下代码提示键，这个方法就会自动出来了，相信这也是很多 Java 新手对它钟情的原因，不过遗憾的是，Android Studio 中已经不支持这种快捷输入了。那缺点又在哪儿了呢？这个就太多了，比如日志打印不可控制、打印时间无法确定、不能添加过滤器、日志没有级别区分……

听我说了这些，你可能已经不太想用 `System.out.println()` 方法了，那么 Log 就把上面所说的缺点全部都改好了吗？虽然谈不上全部，但我觉得 Log 已经做得相当不错了。我现在就来带你看看 Log 和 logcat 配合的强大之处。

首先刚才提到的快捷输入，在 Android Studio 当中也是有的，比如你想打印一条 `debug` 级别的日志，那么只需要输入 `logd`，然后按下 Tab 键，就会帮你自动补全一条完整的打印语句。输入 `logi`，然后按下 Tab 键，会自动补全一条 `info` 级别的打印日志。输入 `logw`，按下 Tab 键，会自动补全一条 `warn` 级别的打印日志，以此类推。另外，由于 Log 的所有打印方法都要求传入一个 tag 参数，每次写一遍显然太过麻烦。这里还有一个小技巧，我们在 `onCreate()` 方法的外面输入 `logt`，然后按下 Tab 键，这时就会以当前的类名作为值自动生成一个 TAG 常量，如下所示：

```
public class HelloWorldActivity extends AppCompatActivity {

    private static final String TAG = "HelloWorldActivity";

    ...
}
```

除了快捷输入之外，logcat中还能很轻松地添加过滤器，你可以在图1.32中看到我们目前所有的过滤器。

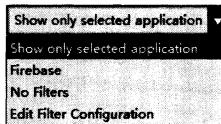


图1.32 logcat中的过滤器

目前只有3个过滤器，Show only selected application表示只显示当前选中程序的日志，Firebase是谷歌提供的一个分析工具，我们可以不用管它，No Filters相当于没有过滤器，会把所有的日志都显示出来。那可不可以自定义过滤器呢？当然可以，我们现在就来添加一个过滤器试试。

点击图1.32中的Edit Filter Configuration，会弹出一个过滤器配置界面。我们给过滤器起名叫data，并且让它对名为data的tag进行过滤，如图1.33所示。

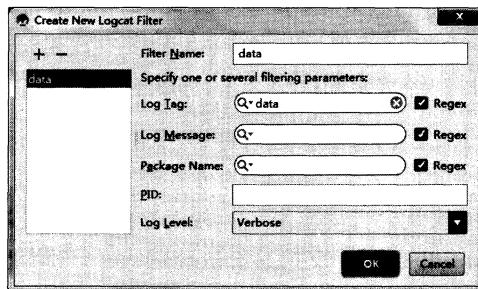


图1.33 过滤器配置界面

点击OK，你就会发现你已经多出了一个data过滤器。当你点击这个过滤器的时候，你会发现刚才在onCreate()方法里打印的日志没了，这是因为data这个过滤器只会显示tag名称为data的日志。你可以尝试在onCreate()方法中把打印日志的语句改成Log.d("data", "onCreate execute")，然后再次运行程序，你就会在data过滤器下看到这行日志了。

不知道你有没有体会到使用过滤器的好处，可能现在还没有吧。不过当你的程序打印出成百上千行日志的时候，你就会迫切地需要过滤器了。

看完了过滤器，再来看一下logcat中的日志级别控制吧。logcat中主要有5个级别，分别对应着上一节介绍的5个方法，如图1.34所示。

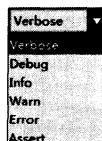


图1.34 logcat中的日志级别

当前我们选中的级别是 `verbose`，也就是最低等级。这意味着不管我们使用哪一个方法打印日志，这条日志都一定会显示出来。而如果我们将级别选中为 `debug`，这时只有我们使用 `debug` 及以上级别方法打印的日志才会显示出来，以此类推。你可以做一下试验，当你把 `logcat` 中的级别选中为 `info`、`warn` 或者 `error` 时，我们在 `onCreate()` 方法中打印的语句是不会显示的，因为我们打印日志时使用的是 `Log.d()` 方法。

日志级别控制的好处就是，你可以很快地找到你所关心的那些日志。相信如果让你从上千行日志中查找一条崩溃信息，你一定会抓狂的吧。而现在你只需要将日志级别选中为 `error`，那些不相干的琐碎信息就不会再干扰你的视线了。

最后我们再来看一下关键字过滤。如果使用过滤器加日志级别控制还是不能锁定到你想查看的日志内容的话，那么还可以通过关键字进行进一步的过滤，如图 1.35 所示。



图 1.35 关键字输入框

我们可以在输入框里输入关键字的内容，这样只有符合关键字条件的日志才会显示出来，从而能够快速定位到任何你想查看的日志。另外还有一点需要注意，关键字过滤是支持正则表达式的，有了这个特性，我们就可以构建出更加丰富的过滤条件。

关于 Android 中日志工具的使用我就准备讲到这里，`logcat` 中其他的一些使用技巧就要靠你自己去摸索了。今天你已经学到了足够多的东西，我们来总结和梳理一下吧。

1.5 小结与点评

你现在一定会觉得很充实，甚至有点沾沾自喜。确实应该如此，因为你已经成为一名真正的 Android 开发者了。通过本章的学习，你首先对 Android 系统有了更加充足的认识，然后成功将 Android 开发环境搭建了起来，接着创建了你自己的第一个 Android 项目，并对 Android 项目的目录结构和执行过程有了一定的认识，在本章的最后还学习了 Android 日志工具的使用，这难道还不够充实吗？

不过你也别太过于满足，相信你很清楚，Android 开发者和出色的 Android 开发者还是有很大的区别的，你还需要付出更多的努力才行。即使你目前在 Java 领域已经有了不错的成绩，我也希望在 Android 的世界你可以放下身段，以一只萌级小菜鸟的身份起飞，在后面的旅途中你会不断地成长。

现在你可以非常安心地休息一段时间，因为今天你已经做得非常不错了。储备好能量，准备进入到下一章的旅程当中。

第 2 章

先从看得到的入手——探究活动

通过上一章的学习，你已经成功创建了你的第一个 Android 项目。不过仅仅满足于此显然是不够的，是时候学点新的东西了。作为你的导师，我有义务帮你制定好后面的学习路线，那么今天我们应该从哪儿入手呢？现在你可以想象一下，假如你已经写出了一个非常优秀的应用程序，然后推荐给你的第一个用户，你会从哪里开始介绍呢？毫无疑问，当然是从界面开始介绍了！因为即使你的程序算法再高效，架构再出色，用户根本不在乎这些，他们一开始只会对看得到的东西感兴趣，那么我们今天的主题自然也要从看得到的入手了。

2.1 活动是什么

活动（Activity）是最容易吸引用户的地方，它是一种可以包含用户界面的组件，主要用于和用户进行交互。一个应用程序中可以包含零个或多个活动，但不包含任何活动的应用程序很少见，谁也不想让自己的应用永远无法被用户看到吧？

其实在上一章中，你已经和活动打过交道了，并且对活动也有了初步的认识。不过上一章我们的重点是创建你的第一个 Android 项目，对活动的介绍并不多，在本章中我将对活动进行详细的介绍。

2.2 活动的基本用法

到现在为止，你还没有手动创建过活动呢，因为上一章中的 `HelloWorldActivity` 是 Android Studio 帮我们自动创建的。手动创建活动可以加深我们的理解，因此现在是时候应该自己动手了。

由于 Android Studio 在一个工作区间内只允许打开一个项目，因此首先你需要将当前的项目关闭，点击导航栏 `File`→`Close Project`。然后再新建一个 Android 项目，项目名可以叫作 `ActivityTest`，包名我们就使用默认值 `com.example.activitytest`。新建项目的步骤你已经在上一章学习过了，不过图 1.12 中的那一步需要稍做修改，我们不再选择 `Empty Activity` 这个选项，而是选择 `Add No`

Activity，因为这次我们准备手动创建活动，如图 2.1 所示。

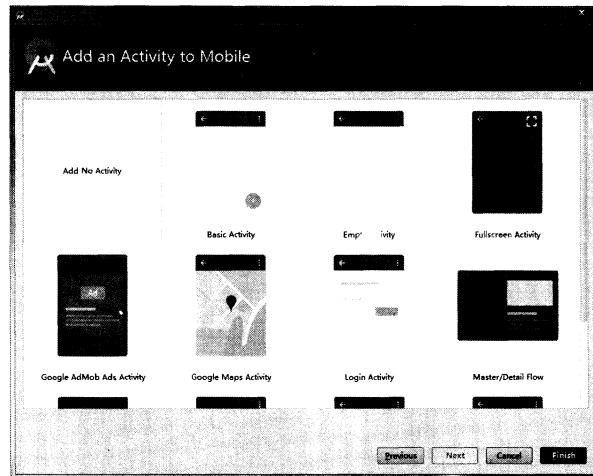


图 2.1 选择不添加活动

点击 Finish，等待 Gradle 构建完成后，项目就创建成功了。

2.2.1 手动创建活动

项目创建成功后，仍然会默认使用 Android 模式的项目结构，这里我们手动改成 Project 模式，本书中后面的所有项目都要这样修改，以后就不再赘述了。目前 ActivityTest 项目中虽然还是会自动生成很多文件，但是 app/src/main/java/com.example.activitytest 目录应该是空的了，如图 2.2 所示。



图 2.2 初始项目结构

现在右击 com.example.activitytest 包 → New → Activity → Empty Activity，会弹出一个创建活动的对话框，我们将活动命名为 FirstActivity，并且不要勾选 Generate Layout File 和 Launcher Activity 这两个选项，如图 2.3 所示。

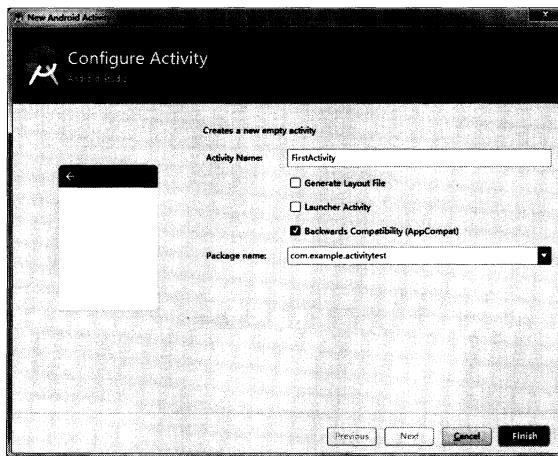


图 2.3 新建活动对话框

勾选 Generate Layout File 表示会自动为 FirstActivity 创建一个对应的布局文件，勾选 Launcher Activity 表示会自动将 FirstActivity 设置为当前项目的主活动，这里由于你是第一次手动创建活动，这些自动生成的东西暂时都不要勾选，下面我们将一个个手动来完成。勾选 Backwards Compatibility 表示会为项目启用向下兼容的模式，这个选项要勾上。点击 Finish 完成创建。

你需要知道，项目中的任何活动都应该重写 Activity 的 `onCreate()` 方法，而目前我们的 FirstActivity 中已经重写了这个方法，这是由 Android Studio 自动帮我们完成的，代码如下所示：

```
public class FirstActivity extends AppCompatActivity {

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

可以看到，`onCreate()` 方法非常简单，就是调用了父类的 `onCreate()` 方法。当然这只是默认的实现，后面我们还需要在里面加入很多自己的逻辑。

2.2.2 创建和加载布局

前面我们说过，Android 程序的设计讲究逻辑和视图分离，最好每一个活动都能对应一个布局，布局就是用来显示界面内容的，因此我们现在就来手动创建一个布局文件。

右击 `app/src/main/res` 目录 → New → Directory，会弹出一个新建目录的窗口，这里先创建一个名为 `layout` 的目录。然后对着 `layout` 目录右键 → Layout resource file，又会弹出一个新建布局资源文件的窗口，我们将这个布局文件命名为 `first_layout`，根元素就默认选择为 `LinearLayout`，如图 2.4 所示。



图 2.4 新建布局资源文件

点击 OK 完成布局的创建，这时候你会看到如图 2.5 所示的布局编辑器。



图 2.5 布局编辑器

这是 Android Studio 为我们提供的可视化布局编辑器，你可以在屏幕的中央区域预览当前的布局。在窗口的最下方有两个切换卡，左边是 Design，右边是 Text。Design 是当前的可视化布局编辑器，在这里你不仅可以预览当前的布局，还可以通过拖放的方式编辑布局。而 Text 则是通过 XML 文件的方式来编辑布局的，现在点击一下 Text 切换卡，可以看到如下代码：

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

</LinearLayout>
```

由于我们刚才在创建布局文件时选择了 LinearLayout 作为根元素，因此现在布局文件中已经有一个 LinearLayout 元素了。那我们现在对这个布局稍做编辑，添加一个按钮，如下所示：

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

<Button
    android:id="@+id/button_1"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button 1"
/>
</LinearLayout>
```

这里添加了一个 Button 元素，并在 Button 元素的内部增加了几个属性。`android:id` 是给当前的元素定义一个唯一标识符，之后可以在代码中对这个元素进行操作。你可能会对`@+id/button_1` 这种语法感到陌生，但如果把加号去掉，变成`@id/button_1`，这样你就会觉得有些熟悉了吧，这不就是在 XML 中引用资源的语法吗？只不过是把 `string` 替换成了 `id`。是的，如果你需要在 XML 中引用一个 `id`，就使用`@id/id_name` 这种语法，而如果你需要在 XML 中定义一个 `id`，则要使用`@+id/id_name` 这种语法。随后 `android:layout_width` 指定了当前元素的宽度，这里使用 `match_parent` 表示让当前元素和父元素一样宽。`android:layout_height` 指定了当前元素的高度，这里使用 `wrap_content` 表示当前元素的高度只要能刚好包含里面的内容就行。`android:text` 指定了元素中显示的文字内容。如果你还不能完全看明白，没有关系，关于编写布局的详细内容我会在下一章中重点讲解，本章只是先简单涉及一些。现在按钮已经添加完了，你可以通过右侧工具栏的 Preview 来预览一下当前布局，如图 2.6 所示。

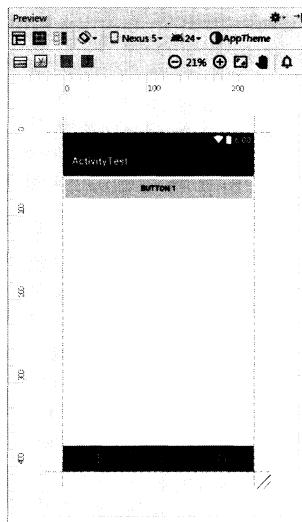


图 2.6 预览当前布局

可以看到，按钮已经成功显示出来了，这样一个简单的布局就编写完成了。那么接下来我们要做的，就是在活动中加载这个布局。

重新回到 FirstActivity，在 `onCreate()` 方法中加入如下代码：

```
public class FirstActivity extends AppCompatActivity {
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.first_layout);
}

```

可以看到，这里调用了 `setContentView()` 方法来给当前的活动加载一个布局，而在 `setContentView()` 方法中，我们一般都会传入一个布局文件的 id。在第 1 章介绍项目资源的时候我曾提到过，项目中添加的任何资源都会在 R 文件中生成一个相应的资源 id，因此我们刚才创建的 `first_layout.xml` 布局的 id 现在应该是已经添加到 R 文件中了。在代码中去引用布局文件的方法你也已经学过了，只需要调用 `R.layout.first_layout` 就可以得到 `first_layout.xml` 布局的 id，然后将这个值传入 `setContentView()` 方法即可。

2.2.3 在 AndroidManifest 文件中注册

别忘了在上一章我们学过，所有的活动都要在 `AndroidManifest.xml` 中进行注册才能生效，而实际上 `FirstActivity` 已经在 `AndroidManifest.xml` 中注册过了，我们打开 `app/src/main/AndroidManifest.xml` 文件瞧一瞧，代码如下所示：

```

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitytest">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".FirstActivity"></activity>
    </application>
</manifest>

```

可以看到，活动的注册声明要放在 `<application>` 标签内，这里是通过 `<activity>` 标签来对活动进行注册的。那么又是谁帮我们自动完成了对 `FirstActivity` 的注册呢？当然是 Android Studio 了，之前在使用 Eclipse 创建活动或其他系统组件时，很多人都会忘记要去 `Android Manifest.xml` 中注册一下，从而导致程序运行崩溃，很显然 Android Studio 在这方面做得更加人性化。

在 `<activity>` 标签中我们使用了 `android:name` 来指定具体注册哪一个活动，那么这里填入的 `.FirstActivity` 是什么意思呢？其实这不过就是 `com.example.activitytest.FirstActivity` 的缩写而已。由于在最外层的 `<manifest>` 标签中已经通过 `package` 属性指定了程序的包名是 `com.example.activitytest`，因此在注册活动时这一部分就可以省略了，直接使用 `.FirstActivity` 就足够了。

不过，仅仅是这样注册了活动，我们的程序仍然是不能运行的，因为还没有为程序配置主活动，也就是说，当程序运行起来的时候，不知道要首先启动哪个活动。配置主活动的方法其实在第 1 章中已经介绍过了，就是在 `<activity>` 标签的内部加入 `<intent-filter>` 标签，并在这个

标签里添加<action android:name="android.intent.action.MAIN"/>和<category android:name="android.intent.category.LAUNCHER" />这两句声明即可。

除此之外，我们还可以使用 android:label 指定活动中标题栏的内容，标题栏是显示在活动最顶部的，待会儿运行的时候你就会看到。需要注意的是，给主活动指定的 label 不仅会成为标题栏中的内容，还会成为启动器（Launcher）中应用程序显示的名称。

修改后的 AndroidManifest.xml 文件，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.activitytest">
    <application
        ...
        <activity android:name=".FirstActivity"
            android:label="This is FirstActivity"
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

这样的话，FirstActivity 就成为我们这个程序的主活动了，即点击桌面应用程序图标时首先打开的就是这个活动。另外需要注意，如果你的应用程序中没有声明任何一个活动作为主活动，这个程序仍然是可以正常安装的，只是你无法在启动器中看到或者打开这个程序。这种程序一般都是作为第三方服务供其他应用在内部进行调用的，如支付宝快捷支付服务。

好了，现在一切都已准备就绪，让我们来运行一下程序吧，结果如图 2.7 所示。

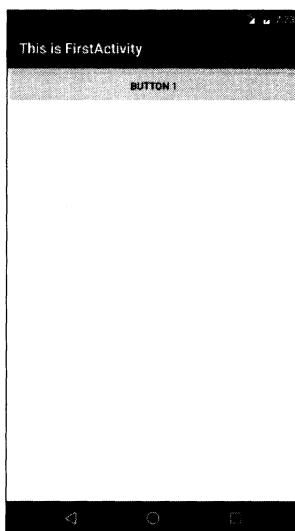


图 2.7 首次运行结果

在界面的最顶部是一个标题栏，里面显示着我们刚才在注册活动时指定的内容。标题栏的下面就是在布局文件 `first_layout.xml` 中编写的界面，可以看到我们刚刚定义的按钮。现在你已经成功掌握了手动创建活动的方法，下面让我们继续看一看你在活动中还能做哪些事情吧。

2.2.4 在活动中使用 Toast

Toast 是 Android 系统提供的一种非常好的提醒方式，在程序中可以使用它将一些短小的信息通知给用户，这些信息会在一段时间后自动消失，并且不会占用任何屏幕空间，我们现在就尝试一下如何在活动中使用 Toast。

首先需要定义一个弹出 Toast 的触发点，正好界面上有个按钮，那我们就让点击这个按钮的时候弹出一个 Toast 吧。在 `onCreate()` 方法中添加如下代码：

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(FirstActivity.this, "You clicked Button 1",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

在活动中，可以通过 `findViewById()` 方法获取到在布局文件中定义的元素，这里我们传入 `R.id.button_1`，来得到按钮的实例，这个值是刚才在 `first_layout.xml` 中通过 `android:id` 属性指定的。`findViewById()` 方法返回的是一个 `View` 对象，我们需要向下转型将它转成 `Button` 对象。得到按钮的实例之后，我们通过调用 `setOnClickListener()` 方法为按钮注册一个监听器，点击按钮时就会执行监听器中的 `onClick()` 方法。因此，弹出 Toast 的功能当然是要在 `onClick()` 方法中编写了。

Toast 的用法非常简单，通过静态方法 `makeText()` 创建出一个 `Toast` 对象，然后调用 `show()` 将 `Toast` 显示出来就可以了。这里需要注意的是，`makeText()` 方法需要传入 3 个参数。第一个参数是 `Context`，也就是 `Toast` 要求的上下文，由于活动本身就是一个 `Context` 对象，因此这里直接传入 `FirstActivity.this` 即可。第二个参数是 `Toast` 显示的文本内容，第三个参数是 `Toast` 显示的时长，有两个内置常量可以选择 `Toast.LENGTH_SHORT` 和 `Toast.LENGTH_LONG`。

现在重新运行程序，并点击一下按钮，效果如图 2.8 所示。



图 2.8 Toast 运行效果

2.2.5 在活动中使用 Menu

手机毕竟和电脑不同，它的屏幕空间非常有限，因此充分地利用屏幕空间在手机界面设计中就显得非常重要了。如果你的活动中有大量的菜单需要显示，这个时候界面设计就会比较尴尬，因为仅这些菜单就可能占用屏幕将近三分之一的空间，这该怎么办呢？不用担心，Android 给我们提供了一种方式，可以让菜单都能得到展示的同时，还能不占用任何屏幕空间。

首先在 res 目录下新建一个 menu 文件夹，右击 res 目录→New→Directory，输入文件夹名 menu，点击 OK。接着在这个文件夹下再新建一个名叫 main 的菜单文件，右击 menu 文件夹→New→Menu resource file，如图 2.9 所示。

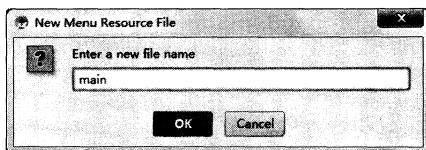


图 2.9 新建 Menu 资源文件

文件名输入 main，点击 OK 完成创建。然后在 main.xml 中添加如下代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/add_item"
        android:title="Add"/>
    <item
        android:id="@+id/remove_item"
```

```
        android:title="Remove"/>
</menu>
```

这里我们创建了两个菜单项，其中`<item>`标签就是用来创建具体的某一个菜单项，然后通过`android:id`给这个菜单项指定一个唯一的标识符，通过`android:title`给这个菜单项指定一个名称。

接着重新回到`FirstActivity`中来重写`onCreateOptionsMenu()`方法，重写方法可以使用`Ctrl + O`快捷键（Mac系统是`control + O`），如图2.10所示。

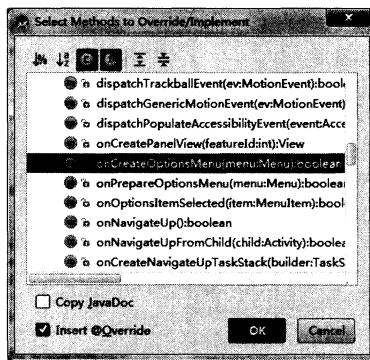


图2.10 重写`onCreateOptionsMenu()`方法

然后在`onCreateOptionsMenu()`方法中编写如下代码：

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

通过`getMenuInflater()`方法能够得到`MenuItemInflater`对象，再调用它的`inflate()`方法就可以给当前活动创建菜单了。`inflate()`方法接收两个参数，第一个参数用于指定我们通过哪一个资源文件来创建菜单，这里当然传入`R.menu.main`。第二个参数用于指定我们的菜单项将添加到哪一个`Menu`对象当中，这里直接使用`onCreateOptionsMenu()`方法中传入的`menu`参数。然后给这个方法返回`true`，表示允许创建的菜单显示出来，如果返回了`false`，创建的菜单将无法显示。

当然，仅仅让菜单显示出来是不够的，我们定义菜单不仅是为了看的，关键是要菜单真正可用才行，因此还要再定义菜单响应事件。在`FirstActivity`中重写`onOptionsItemSelected()`方法：

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.add_item:
            Toast.makeText(this, "You clicked Add", Toast.LENGTH_SHORT).show();
            break;
        case R.id.remove_item:
            Toast.makeText(this, "You clicked Remove", Toast.LENGTH_SHORT).show();
    }
}
```

```

        break;
    default:
    }
    return true;
}

```

在 `onOptionsItemSelected()` 方法中，通过调用 `item.getItemId()` 来判断我们点击的是哪一个菜单项，然后给每个菜单项加入自己的逻辑处理，这里我们就活学活用，弹出一个刚刚学会的 `Toast`。

重新运行程序，你会发现在标题栏的右侧多了一个三点的符号，这个就是菜单按钮了，如图 2.11 所示。

可以看到，菜单里的菜单项默认是不会显示出来的，只有点击一下菜单按钮才会弹出里面具体的内容，因此它不会占用任何活动的空间，如图 2.12 所示。

然后如果你点击了 Add 菜单项就会弹出 You clicked Add 提示（如图 2.13 所示），如果点击了 Remove 菜单项就会弹出 You clicked Remove 提示。

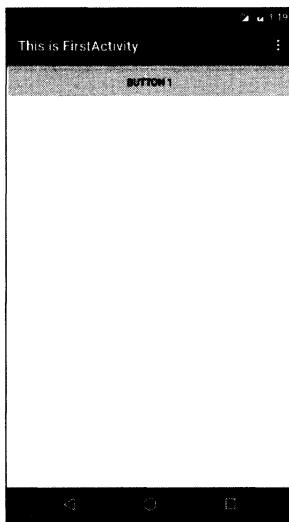


图 2.11 带菜单按钮的活动

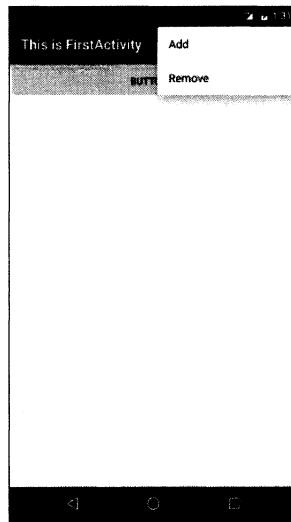


图 2.12 弹出菜单项的界面

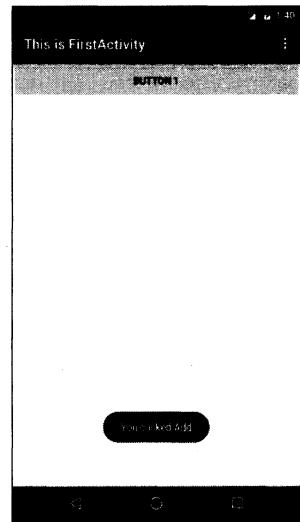


图 2.13 点击了 Add 菜单项

2.2.6 销毁一个活动

通过上一节的学习，你已经掌握了手动创建活动的方法，并学会了如何在活动中创建 `Toast` 和创建菜单。或许你现在心中会有个疑惑，如何销毁一个活动呢？

其实答案非常简单，只要按一下 `Back` 键就可以销毁当前的活动了。不过如果你不想通过按键的方式，而是希望在程序中通过代码来销毁活动，当然也可以，`Activity` 类提供了一个 `finish()` 方法，我们在活动中调用一下这个方法就可以销毁当前活动了。

修改按钮监听器中的代码，如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        finish();  
    }  
});
```

重新运行程序，这时点击一下按钮，当前的活动就被成功销毁了，效果和按下 Back 键是一样的。

2.3 使用 Intent 在活动之间穿梭

只有一个活动的应用也太简单了吧？没错，你的追求应该更高一点。不管你想创建多少个活动，方法都和上一节中介绍的是一样的。唯一的问题在于，你在启动器中点击应用的图标只会进入到该应用的主活动，那么怎样才能由主活动跳转到其他活动呢？我们现在就来一起看一看。

2.3.1 使用显式 Intent

你应该已经对创建活动的流程比较熟悉了，那我们现在快速地在 ActivityTest 项目中再创建一个活动。

仍然还是右击 com.example.activitytest 包→New→Activity→Empty Activity，会弹出一个创建活动的对话框，我们这次将活动命名为 SecondActivity，并勾选 Generate Layout File，给布局文件起名为 second_layout，但不要勾选 Launcher Activity 选项，如图 2.14 所示。

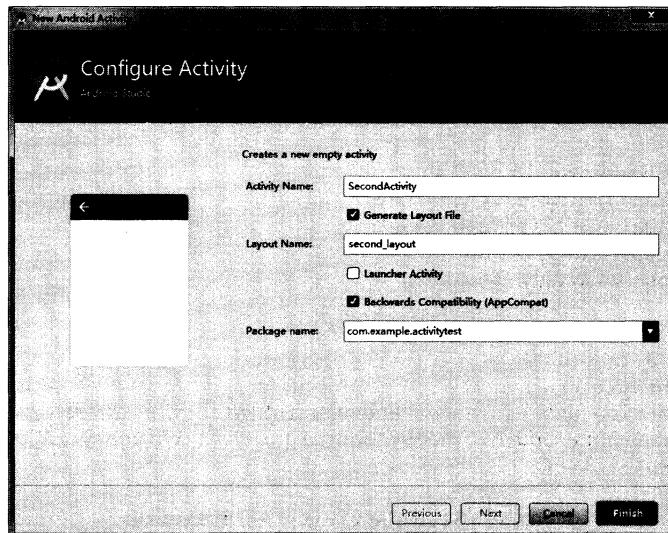


图 2.14 创建 SecondActivity

点击 Finish 完成创建, Android Studio 会为我们自动生成 SecondActivity.java 和 second_layout.xml 这两个文件。不过自动生成的布局代码目前对你来说可能有些复杂, 这里我们仍然还是使用最熟悉的 LinearLayout, 编辑 second_layout.xml, 将里面的代码替换成如下内容:

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

    <Button
        android:id="@+id/button_2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
    />

</LinearLayout>
```

我们还是定义了一个按钮, 按钮上显示 Button 2。

然后 SecondActivity 中的代码已经自动生成了一部分, 我们保持默认不变就好, 如下所示:

```
public class SecondActivity extends AppCompatActivity {

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

另外不要忘记, 任何一个活动都是需要在 AndroidManifest.xml 中注册的, 不过幸运的是, Android Studio 已经帮我们自动完成了, 你可以打开 AndroidManifest.xml 瞧一瞧:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
        android:name=".FirstActivity"
        android:label="This is FirstActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".SecondActivity"></activity>
</application>
```

由于 SecondActivity 不是主活动，因此不需要配置<intent-filter>标签里的内容，注册活动的代码也简单了许多。现在第二个活动已经创建完成，剩下的问题就是如何去启动这第二个活动了，这里我们需要引入一个新的概念：Intent。

Intent 是 Android 程序中各组件之间进行交互的一种重要方式，它不仅可以指明当前组件想要执行的动作，还可以在不同组件之间传递数据。Intent 一般可被用于启动活动、启动服务以及发送广播等场景，由于服务、广播等概念你暂时还未涉及，那么本章我们的目光无疑就锁定在了启动活动上面。

Intent 大致可以分为两种：显式 Intent 和隐式 Intent，我们先来看一下显式 Intent 如何使用。

Intent 有多个构造函数的重载，其中一个是 Intent(Context packageContext, Class<?> cls)。这个构造函数接收两个参数，第一个参数 Context 要求提供一个启动活动的上下文，第二个参数 Class 则是指定想要启动的目标活动，通过这个构造函数就可以构建出 Intent 的“意图”。然后我们应该怎么使用这个 Intent 呢？Activity 类中提供了一个 startActivity()方法，这个方法是专门用于启动活动的，它接收一个 Intent 参数，这里我们将构建好的 Intent 传入 startActivity()方法就可以启动目标活动了。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        startActivity(intent);
    }
});
```

我们首先构建出了一个 Intent，传入 FirstActivity.this 作为上下文，传入 SecondActivity.class 作为目标活动，这样我们的“意图”就非常明确了，即在 FirstActivity 这个活动的基础上打开 SecondActivity 这个活动。然后通过 startActivity()方法来执行这个 Intent。

重新运行程序，在 FirstActivity 的界面点击一下按钮，结果如图 2.15 所示。

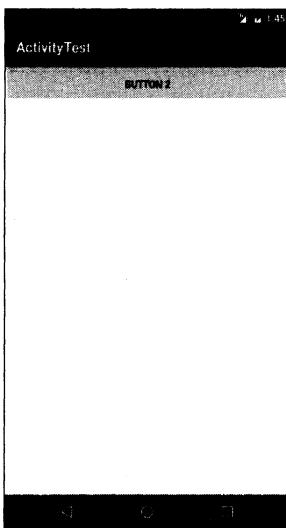


图 2.15 SecondActivity 界面

可以看到，我们已经成功启动 SecondActivity 这个活动了。如果你想要回到上一个活动怎么办呢？很简单，按下 Back 键就可以销毁当前活动，从而回到上一个活动了。

使用这种方式来启动活动，Intent 的“意图”非常明显，因此我们称之为显式 Intent。

2.3.2 使用隐式 Intent

相比于显式 Intent，隐式 Intent 则含蓄了许多，它并不明确指出我们想要启动哪一个活动，而是指定了一系列更为抽象的 action 和 category 等信息，然后交由系统去分析这个 Intent，并帮我们找出合适的活动去启动。

什么叫作合适的活动呢？简单来说就是可以响应我们这个隐式 Intent 的活动，那么目前 SecondActivity 可以响应什么样的隐式 Intent 呢？额，现在好像还什么都响应不了，不过很快就会有了。

通过在<activity>标签下配置<intent-filter>的内容，可以指定当前活动能够响应的 action 和 category，打开 AndroidManifest.xml，添加如下代码：

```
<activity android:name=".SecondActivity" >
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

在<action>标签中我们指明了当前活动可以响应 com.example.activitytest.ACTION_START 这个 action，而<category>标签则包含了一些附加信息，更精确地指明了当前的活动能

够响应的 Intent 中还可能带有的 category。只有<action>和<category>中的内容同时能够匹配上 Intent 中指定的 action 和 category 时，这个活动才能响应该 Intent。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent("com.example.activitytest.ACTION_START");
        startActivity(intent);
    }
});
```

可以看到，我们使用了 Intent 的另一个构造函数，直接将 action 的字符串传了进去，表明我们想要启动能够响应 com.example.activitytest.ACTION_START 这个 action 的活动。那前面不是说要<action>和<category>同时匹配上才能响应的吗？怎么没看到哪里有指定 category 呢？这是因为 android.intent.category.DEFAULT 是一种默认的 category，在调用 startActivity()方法的时候会自动将这个 category 添加到 Intent 中。

重新运行程序，在 FirstActivity 的界面点击一下按钮，你同样成功启动 SecondActivity 了。不同的是，这次你是使用了隐式 Intent 的方式来启动的，说明我们在<activity>标签下配置的 action 和 category 的内容已经生效了！

每个 Intent 中只能指定一个 action，但却能指定多个 category。目前我们的 Intent 中只有一个默认的 category，那么现在再来增加一个吧。

修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent("com.example.activitytest.ACTION_START");
        intent.addCategory("com.example.activitytest.MY_CATEGORY");
        startActivity(intent);
    }
});
```

可以调用 Intent 中的 addCategory()方法来添加一个 category，这里我们指定了一个自定义的 category，值为 com.example.activitytest.MY_CATEGORY。

现在重新运行程序，在 FirstActivity 的界面点击一下按钮，你会发现，程序崩溃了！这是你第一次遇到程序崩溃，可能会有些束手无策。别紧张，其实大多数的崩溃问题都是很好解决的，只要你善于分析。在 logcat 界面查看错误日志，你会看到如图 2.16 所示的错误信息。

```
Process: com.example.activitytest, PID: 24027
android.content.ActivityNotFoundException: No Activity found to handle Intent {
    act=com.example.activitytest.ACTION_START cat=[com.example.activitytest.MY_CATEGORY] }
```

图 2.16 错误信息

错误信息中提醒我们，没有任何一个活动可以响应我们的 Intent，为什么呢？这是因为我们刚刚在 Intent 中新增了一个 category，而 SecondActivity 的<intent-filter>标签中并没有声明可以响应这个 category，所以就出现了没有任何活动可以响应该 Intent 的情况。现在我们在<intent-filter>中再添加一个 category 的声明，如下所示：

```
<activity android:name=".SecondActivity" >
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="com.example.activitytest.MY_CATEGORY"/>
    </intent-filter>
</activity>
```

再次重新运行程序，你就会发现一切都正常了。

2.3.3 更多隐式 Intent 的用法

上一节中，你掌握了通过隐式 Intent 来启动活动的方法，但实际上隐式 Intent 还有更多的内容需要你去了解，本节我们就来展开介绍一下。

使用隐式 Intent，我们不仅可以启动自己程序内的活动，还可以启动其他程序的活动，这使得 Android 多个应用程序之间的功能共享成为了可能。比如说你的应用程序中需要展示一个网页，这时你没有必要自己去实现一个浏览器（事实上也不太可能），而是只需要调用系统的浏览器来打开这个网页就行了。

修改 FirstActivity 中按钮点击事件的代码，如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://www.baidu.com"));
        startActivity(intent);
    }
});
```

这里我们首先指定了 Intent 的 action 是 Intent.ACTION_VIEW，这是一个 Android 系统内置的动作，其常量值为 android.intent.action.VIEW。然后通过 Uri.parse()方法，将一个网址字符串解析成一个 Uri 对象，再调用 Intent 的 setData()方法将这个 Uri 对象传递进去。

重新运行程序，在 FirstActivity 界面点击按钮就可以看到打开了系统浏览器，如图 2.17 所示。

在上述代码中，可能你会对 setData()部分感觉到陌生，这是我们前面没有讲到的。这个方法其实并不复杂，它接收一个 Uri 对象，主要用于指定当前 Intent 正在操作的数据，而这些数据通常都是以字符串的形式传入到 Uri.parse()方法中解析产生的。



图 2.17 系统浏览器界面

与此对应，我们还可以在<intent-filter>标签中再配置一个<data>标签，用于更精确地指定当前活动能够响应什么类型的数据。<data>标签中主要可以配置以下内容。

- **android:scheme**。用于指定数据的协议部分，如上例中的 http 部分。
- **android:host**。用于指定数据的主机名部分，如上例中的 www.baidu.com 部分。
- **android:port**。用于指定数据的端口部分，一般紧随在主机名之后。
- **android:path**。用于指定主机名和端口之后的部分，如一段网址中跟在域名之后的内容。
- **android:mimeType**。用于指定可以处理的数据类型，允许使用通配符的方式进行指定。

只有<data>标签中指定的内容和 Intent 中携带的 Data 完全一致时，当前活动才能够响应该 Intent。不过一般在<data>标签中都不会指定过多的内容，如上面浏览器示例中，其实只需要指定 android:scheme 为 http，就可以响应所有的 http 协议的 Intent 了。

为了让你能够更加直观地理解，我们来自己建立一个活动，让它也能响应打开网页的 Intent。

右击 com.example.activitytest 包 → New → Activity → Empty Activity，新建 ThirdActivity，并勾选 Generate Layout File，给布局文件起名为 third_layout，点击 Finish 完成创建。然后编辑 third_layout.xml，将里面的代码替换成如下内容：

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

    <Button
```

```
    android:id="@+id/button_3"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button 3"
/>

```

```
</LinearLayout>
```

ThirdActivity 中的代码保持不变就可以了，最后在 AndroidManifest.xml 中修改 ThirdActivity 的注册信息：

```
<activity android:name=".ThirdActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
    </intent-filter>
</activity>
```

我们在 ThirdActivity 的<intent-filter>中配置了当前活动能够响应的 action 是 Intent.ACTION_VIEW 的常量值，而 category 则毫无疑问指定了默认的 category 值，另外在<data>标签中我们通过 android:scheme 指定了数据的协议必须是 http 协议，这样 ThirdActivity 应该就和浏览器一样，能够响应一个打开网页的 Intent 了。让我们运行一下程序试试吧，在 FirstActivity 的界面点击一下按钮，结果如图 2.18 所示。

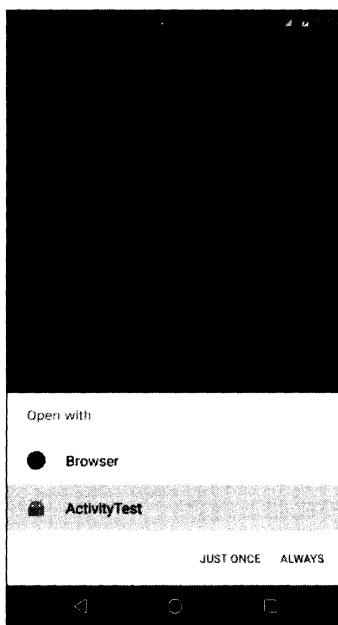


图 2.18 选择响应 Intent 的程序

可以看到，系统自动弹出了一个列表，显示了目前能够响应这个 Intent 的所有程序。选择 Browser 还会像之前一样打开浏览器，并显示百度的主页，而如果选择了 ActivityTest，则会启动 ThirdActivity。JUST ONCE 表示只是这次使用选择的程序打开，ALWAYS 则表示以后一直都使用这次选择的程序打开。需要注意的是，虽然我们声明了 ThirdActivity 是可以响应打开网页的 Intent 的，但实际上这个活动并没有加载并显示网页的功能，所以在真正的项目中尽量不要出现这种有可能误导用户的行为，不然会让用户对我们的应用产生负面的印象。

除了 http 协议外，我们还可以指定很多其他协议，比如 geo 表示显示地理位置、tel 表示拨打电话。下面的代码展示了如何在我们的程序中调用系统拨号界面。

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(Intent.ACTION_DIAL);
        intent.setData(Uri.parse("tel:10086"));
        startActivity(intent);
    }
});
```

首先指定了 Intent 的 action 是 Intent.ACTION_DIAL，这又是一个 Android 系统的内置动作。然后在 data 部分指定了协议是 tel，号码是 10086。重新运行一下程序，在 FirstActivity 的界面上点击一下按钮，结果如图 2.19 所示。

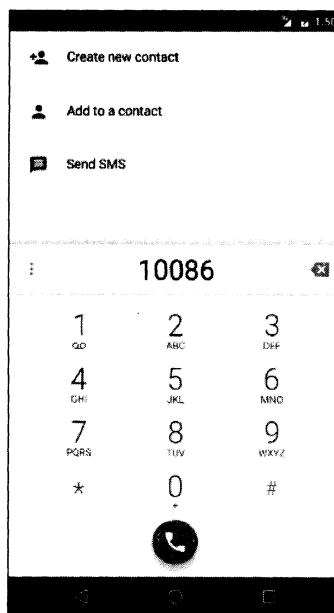


图 2.19 系统拨号界面

2.3.4 向下一个活动传递数据

经过前面几节的学习，你已经对 Intent 有了一定的了解。不过到目前为止，我们都只是简单地使用 Intent 来启动一个活动，其实 Intent 还可以在启动活动的时候传递数据，下面一起来看一下。

在启动活动时传递数据的思路很简单，Intent 中提供了一系列 `putExtra()` 方法的重载，可以把我们要传递的数据暂存在 Intent 中，启动了另一个活动后，只需要把这些数据再从 Intent 中取出就可以了。比如说 `FirstActivity` 中有一个字符串，现在想把这个字符串传递到 `SecondActivity` 中，你就可以这样编写：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String data = "Hello SecondActivity";
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        intent.putExtra("extra_data", data);
        startActivity(intent);
    }
});
```

这里我们还是使用显式 Intent 的方式来启动 `SecondActivity`，并通过 `putExtra()` 方法传递了一个字符串。注意这里 `putExtra()` 方法接收两个参数，第一个参数是键，用于后面从 Intent 中取值，第二个参数才是真正要传递的数据。

然后我们在 `SecondActivity` 中将传递的数据取出，并打印出来，代码如下所示：

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_layout);
        Intent intent = getIntent();
        String data = intent.getStringExtra("extra_data");
        Log.d("SecondActivity", data);
    }
}
```

首先可以通过 `getIntent()` 方法获取到用于启动 `SecondActivity` 的 Intent，然后调用 `getStringExtra()` 方法，传入相应的键值，就可以得到传递的数据了。这里由于我们传递的是字符串，所以使用 `getStringExtra()` 方法来获取传递的数据。如果传递的是整型数据，则使用 `getIntExtra()` 方法；如果传递的是布尔型数据，则使用 `getBooleanExtra()` 方法，以此类推。

重新运行程序，在 `FirstActivity` 的界面点击一下按钮会跳转到 `SecondActivity`，查看 logcat

打印信息，如图 2.20 所示。

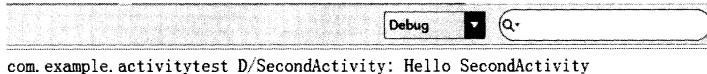


图 2.20 SecondActivity 中的打印信息

可以看到，我们在 SecondActivity 中成功得到了从 FirstActivity 传递过来的数据。

2.3.5 返回数据给上一个活动

既然可以传递数据给下一个活动，那么能不能够返回数据给上一个活动呢？答案是肯定的。不过不同的是，返回上一个活动只需要按一下 Back 键就可以了，并没有一个用于启动活动 Intent 来传递数据。通过查阅文档你会发现，Activity 中还有一个 `startActivityForResult()` 方法也是用于启动活动的，但这个方法期望在活动销毁的时候能够返回一个结果给上一个活动。毫无疑问，这就是我们所需要的。

`startActivityForResult()` 方法接收两个参数，第一个参数还是 Intent，第二个参数是请求码，用于在之后的回调中判断数据的来源。我们还是来实战一下，修改 FirstActivity 中按钮的点击事件，代码如下所示：

```
button1.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
        startActivityForResult(intent, 1);
    }
});
```

这里我们使用了 `startActivityForResult()` 方法来启动 SecondActivity，请求码只要是一个唯一值就可以了，这里传入了 1。接下来我们在 SecondActivity 中给按钮注册点击事件，并在点击事件中添加返回数据的逻辑，代码如下所示：

```
public class SecondActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second_layout);
        Button button2 = (Button) findViewById(R.id.button_2);
        button2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.putExtra("data_return", "Hello FirstActivity");
                setResult(RESULT_OK, intent);
                finish();
            }
        });
    }
}
```

```

        }
    });
}
}

```

可以看到，我们还是构建了一个 Intent，只不过这个 Intent 仅仅是用于传递数据而已，它没有指定任何的“意图”。紧接着把要传递的数据存放在 Intent 中，然后调用了 `setResult()` 方法。这个方法非常重要，是专门用于向上一个活动返回数据的。 `setResult()` 方法接收两个参数，第一个参数用于向上一个活动返回处理结果，一般只使用 `RESULT_OK` 或 `RESULT_CANCELED` 这两个值，第二个参数则把带有数据的 Intent 传递回去，然后调用了 `finish()` 方法来销毁当前活动。

由于我们是使用 `startActivityForResult()` 方法来启动 `SecondActivity` 的，在 `SecondActivity` 被销毁之后会回调上一个活动的 `onActivityResult()` 方法，因此我们需要在 `FirstActivity` 中重写这个方法来得到返回的数据，如下所示：

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case 1:
            if (resultCode == RESULT_OK) {
                String returnedData = data.getStringExtra("data_return");
                Log.d("FirstActivity", returnedData);
            }
            break;
        default:
    }
}

```

`onActivityResult()` 方法带有三个参数，第一个参数 `requestCode`，即我们在启动活动时传入的请求码。第二个参数 `resultCode`，即我们在返回数据时传入的处理结果。第三个参数 `data`，即携带着返回数据的 Intent。由于在一个活动中有可能调用 `startActivityForResult()` 方法去启动很多不同的活动，每一个活动返回的数据都会回调到 `onActivityResult()` 这个方法中，因此我们首先要做的就是通过检查 `requestCode` 的值来判断数据来源。确定数据是从 `SecondActivity` 返回的之后，我们再通过 `resultCode` 的值来判断处理结果是否成功。最后从 `data` 中取值并打印出来，这样就完成了向上一个活动返回数据的工作。

重新运行程序，在 `FirstActivity` 的界面点击按钮会打开 `SecondActivity`，然后在 `SecondActivity` 界面点击 `Button 2` 按钮会回到 `FirstActivity`，这时查看 `logcat` 的打印信息，如图 2.21 所示。

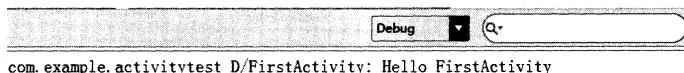


图 2.21 FirstActivity 中的打印信息

可以看到，SecondActivity 已经成功返回数据给 FirstActivity 了。

这时候你可能会问，如果用户在 SecondActivity 中并不是通过点击按钮，而是通过按下 Back 键回到 FirstActivity，这样数据不就没法返回了吗？没错，不过这种情况还是很好处理的，我们可以通过在 SecondActivity 中重写 `onBackPressed()` 方法来解决这个问题，代码如下所示：

```
@Override
public void onBackPressed() {
    Intent intent = new Intent();
    intent.putExtra("data_return", "Hello FirstActivity");
    setResult(RESULT_OK, intent);
    finish();
}
```

这样的话，当用户按下 Back 键，就会去执行 `onBackPressed()` 方法中的代码，我们在这里添加返回数据的逻辑就行了。

2.4 活动的生命周期

掌握活动的生命周期对任何 Android 开发者来说都非常重要，当你深入理解活动的生命周期之后，就可以写出更加连贯流畅的程序，并在如何合理管理应用资源方面发挥得游刃有余。你的应用程序将会拥有更好的用户体验。

2.4.1 返回栈

经过前面几节的学习，我相信你已经发现了这一点，Android 中的活动是可以层叠的。我们每启动一个新的活动，就会覆盖在原活动之上，然后点击 Back 键会销毁最上面的活动，下面的一个活动就会重新显示出来。

其实 Android 是使用任务（Task）来管理活动的，一个任务就是一组存放在栈里的活动的集合，这个栈也被称作返回栈（Back Stack）。栈是一种后进先出的数据结构，在默认情况下，每当我们启动了一个新的活动，它会在返回栈中入栈，并处于栈顶的位置。而每当我们按下 Back 键或调用 `finish()` 方法去销毁一个活动时，处于栈顶的活动会出栈，这时前一个人栈的活动就会重新处于栈顶的位置。系统总是会显示处于栈顶的活动给用户。

示意图 2.22 展示了返回栈是如何管理活动入栈出栈操作的。



图 2.22 返回栈工作示意图

2.4.2 活动状态

每个活动在其生命周期中最多可能会有 4 种状态。

1. 运行状态

当一个活动位于返回栈的栈顶时，这时活动就处于运行状态。系统最不愿意回收的就是处于运行状态的活动，因为这会带来非常差的用户体验。

2. 暂停状态

当一个活动不再处于栈顶位置，但仍然可见时，这时活动就进入了暂停状态。你可能会觉得既然活动已经不在栈顶了，还怎么会可见呢？这是因为并不是每一个活动都会占满整个屏幕的，比如对话框形式的活动只会占用屏幕中间的部分区域，你很快就会在后面看到这种活动。处于暂停状态的活动仍然是完全存活的，系统也不愿意去回收这种活动（因为它还是可见的，回收可见的东西都会在用户体验方面有不好的影响），只有在内存极低的情况下，系统才会去考虑回收这种活动。

3. 停止状态

当一个活动不再处于栈顶位置，并且完全不可见的时候，就进入了停止状态。系统仍然会为这种活动保存相应状态和成员变量，但是这并不是完全可靠的，当其他地方需要内存时，处于停止状态的活动有可能会被系统回收。

4. 销毁状态

当一个活动从返回栈中移除后就变成了销毁状态。系统会最倾向于回收处于这种状态的活动，从而保证手机的内存充足。

2.4.3 活动的生存期

Activity 类中定义了 7 个回调方法，覆盖了活动生命周期的每一个环节，下面就来一一介绍这 7 个方法。

- ❑ **onCreate()**。这个方法你已经看到过很多次了，每个活动中我们都重写了这个方法，它会在活动第一次被创建的时候调用。你应该在这个方法中完成活动的初始化操作，比如说加载布局、绑定事件等。
- ❑ **onStart()**。这个方法在活动由不可见变为可见的时候调用。
- ❑ **onResume()**。这个方法在活动准备好和用户进行交互的时候调用。此时的活动一定位于返回栈的栈顶，并且处于运行状态。
- ❑ **onPause()**。这个方法在系统准备去启动或者恢复另一个活动的时候调用。我们通常会在这个方法中将一些消耗 CPU 的资源释放掉，以及保存一些关键数据，但这个方法的执行速度一定要快，不然会影响到新的栈顶活动的使用。
- ❑ **onStop()**。这个方法在活动完全不可见的时候调用。它和 **onPause()**方法的主要区别在于，如果启动的新活动是一个对话框式的活动，那么 **onPause()**方法会得到执行，而 **onStop()**方法并不会执行。
- ❑ **onDestroy()**。这个方法在活动被销毁之前调用，之后活动的状态将变为销毁状态。
- ❑ **onRestart()**。这个方法在活动由停止状态变为运行状态之前调用，也就是活动被重新启动了。

以上 7 个方法中除了 **onRestart()**方法，其他都是两两相对的，从而又可以将活动分为 3 种生存期。

- ❑ **完整生存期**。活动在 **onCreate()**方法和 **onDestroy()**方法之间所经历的，就是完整生存期。一般情况下，一个活动会在 **onCreate()**方法中完成各种初始化操作，而在 **onDestroy()**方法中完成释放内存的操作。
- ❑ **可见生存期**。活动在 **onStart()**方法和 **onStop()**方法之间所经历的，就是可见生存期。在可见生存期内，活动对于用户总是可见的，即便有可能无法和用户进行交互。我们可以通过这两个方法，合理地管理那些对用户可见的资源。比如在 **onStart()**方法中对资源进行加载，而在 **onStop()**方法中对资源进行释放，从而保证处于停止状态的活动不会占用过多内存。
- ❑ **前台生存期**。活动在 **onResume()**方法和 **onPause()**方法之间所经历的就是前台生存期。在前台生存期内，活动总是处于运行状态的，此时的活动是可以和用户进行交互的，我们平时看到和接触最多的也就是这个状态下的活动。

为了帮助你能够更好地理解，Android 官方提供了一张活动生命周期的示意图，如图 2.23 所示。



图 2.23 活动的生命周期

2.4.4 体验活动的生命周期

讲了这么多理论知识，也是时候该实战一下了，下面我们将通过一个实例，让你可以更加直观地体验活动的生命周期。

这次我们不准备在 `ActivityTest` 这个项目的基础上修改了，而是新建一个项目。因此，首先关闭 `ActivityTest` 项目，点击导航栏 `File→Close Project`。然后再新建一个 `ActivityLifecycleTest` 项目，新建项目的过程你应该已经非常清楚了，不需要我再进行赘述，这次我们允许 `Android Studio` 帮我们自动创建活动和布局，并且勾选 `Launcher Activity` 来将创建的活动设置为主活动，这样可以省去不少工作，创建的活动名和布局名都使用默认值。

这样主活动就创建完成了，我们还需要分别再创建两个子活动——`NormalActivity` 和 `DialogActivity`，下面一步步来实现。

右击 com.example.activitylifecycletest 包→New→Activity→Empty Activity，新建 NormalActivity，布局起名为 normal_layout。然后使用同样的方式创建 DialogActivity，布局起名为 dialog_layout。

现在编辑 normal_layout.xml 文件，将里面的代码替换成如下内容：

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a normal activity"
    />

</LinearLayout>
```

这个布局中我们就非常简单地使用了一个 TextView，用于显示一行文字，在下一章中你将会学到更多关于 TextView 的用法。

然后再编辑 dialog_layout.xml 文件，将里面的代码替换成如下内容：

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a dialog activity"
    />

</LinearLayout>
```

两个布局文件的代码几乎没有区别，只是显示的文字不同而已。

NormalActivity 和 DialogActivity 中的代码我们保持默认就好，不需要改动。

其实从名字上你就可以看出，这两个活动一个是普通的活动，一个是对话框式的活动。可是我们并没有修改活动的任何代码，两个活动的代码应该几乎是一模一样的，在哪里有体现出将活动设成对话框式的呢？别着急，下面我们马上开始设置。修改 AndroidManifest.xml 的<activity>标签的配置，如下所示：

```
<activity android:name=".NormalActivity">
</activity>
<activity android:name=".DialogActivity"
    android:theme="@android:style/Theme.Dialog">
</activity>
```

这里是两个活动的注册代码，但是 DialogActivity 的代码有些不同，我们给它使用了一个 android:theme 属性，这是用于给当前活动指定主题的，Android 系统内置有很多主题可以选择，当然我们也可以定制自己的主题，而这里@android:style/Theme.Dialog 则毫无疑问是让 DialogActivity 使用对话框式的主题。

接下来我们修改 activity_main.xml，重新定制主活动的布局，将里面的代码替换成如下内容：

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

    <Button
        android:id="@+id/start_normal_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start NormalActivity" />

    <Button
        android:id="@+id/start_dialog_activity"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Start DialogActivity" />

</LinearLayout>
```

可以看到，我们在 LinearLayout 中加入了两个按钮，一个用于启动 NormalActivity，一个用于启动 DialogActivity。

最后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    public static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button startNormalActivity = (Button) findViewById(R.id.start_normal_activity);
        Button startDialogActivity = (Button) findViewById(R.id.start_dialog_activity);
        startNormalActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, NormalActivity.class);
                startActivity(intent);
            }
        });
        startDialogActivity.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
        Intent intent = new Intent(MainActivity.this, DialogActivity.class);
        startActivity(intent);
    }
});

@Override
protected void onStart() {
    super.onStart();
    Log.d(TAG, "onStart");
}

@Override
protected void onResume() {
    super.onResume();
    Log.d(TAG, "onResume");
}

@Override
protected void onPause() {
    super.onPause();
    Log.d(TAG, "onPause");
}

@Override
protected void onStop() {
    super.onStop();
    Log.d(TAG, "onStop");
}

@Override
protected void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy");
}

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

}
```

在 `onCreate()` 方法中，我们分别为两个按钮注册了点击事件，点击第一个按钮会启动 `NormalActivity`，点击第二个按钮会启动 `DialogActivity`。然后在 `Activity` 的 7 个回调方法中分别打印了一句话，这样就可以通过观察日志的方式来更直观地理解活动的生命周期。

现在运行程序，效果如图 2.24 所示。



图 2.24 MainActivity 界面

这时观察 logcat 中的打印日志，如图 2.25 所示。



图 2.25 启动程序时的打印日志

可以看到，当 MainActivity 第一次被创建时会依次执行 `onCreate()`、`onStart()` 和 `onResume()` 方法。然后点击第一个按钮，启动 NormalActivity，如图 2.26 所示。



图 2.26 NormalActivity 界面

此时的打印信息如图 2.27 所示。

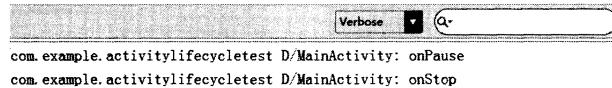


图 2.27 打开 NormalActivity 时的打印日志

由于 NormalActivity 已经把 MainActivity 完全遮挡住，因此 onPause() 和 onStop() 方法都会得到执行。然后按下 Back 键返回 MainActivity，打印信息如图 2.28 所示。

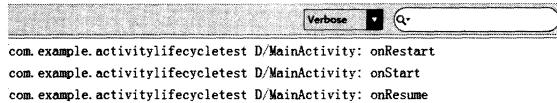


图 2.28 返回 MainActivity 的打印日志

由于之前 MainActivity 已经进入了停止状态，所以 onRestart() 方法会得到执行，之后又会依次执行 onStart() 和 onResume() 方法。注意此时 onCreate() 方法不会执行，因为 MainActivity 并没有重新创建。

然后再点击第二个按钮，启动 DialogActivity，如图 2.29 所示。



图 2.29 DialogActivity 界面

此时观察打印信息，如图 2.30 所示。

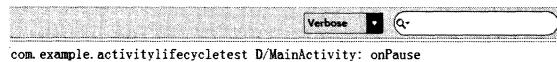


图 2.30 打开 DialogActivity 时的打印日志

可以看到，只有 `onPause()` 方法得到了执行，`onStop()` 方法并没有执行，这是因为 `DialogActivity` 并没有完全遮挡住 `MainActivity`，此时 `MainActivity` 只是进入了暂停状态，并没有进入停止状态。相应地，按下 Back 键返回 `MainActivity` 也应该只有 `onResume()` 方法会得到执行，如图 2.31 所示。

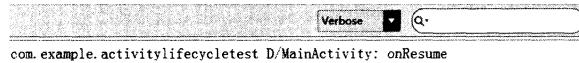


图 2.31 再次返回 `MainActivity` 的打印日志

最后在 `MainActivity` 按下 Back 键退出程序，打印信息如图 2.32 所示。

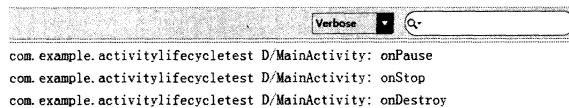


图 2.32 退出程序时的打印日志

依次会执行 `onPause()`、`onStop()` 和 `onDestroy()` 方法，最终销毁 `MainActivity`。

这样活动完整的生命周期你已经体验了一遍，是不是理解得更加深刻了？

2.4.5 活动被回收了怎么办

前面我们已经说过，当一个活动进入到了停止状态，是有可能被系统回收的。那么想象以下场景：应用中有一个活动 A，用户在活动 A 的基础上启动了活动 B，活动 A 就进入了停止状态，这个时候由于系统内存不足，将活动 A 回收掉了，然后用户按下 Back 键返回活动 A，会出现什么情况呢？其实还是会正常显示活动 A 的，只不过这时并不会执行 `onRestart()` 方法，而是会执行活动 A 的 `onCreate()` 方法，因为活动 A 在这种情况下会被重新创建一次。

这样看上去好像一切正常，可是别忽略了一个重要问题，活动 A 中是可能存在临时数据和状态的。打个比方，`MainActivity` 中有一个文本输入框，现在你输入了一段文字，然后启动 `NormalActivity`，这时 `MainActivity` 由于系统内存不足被回收掉，过了一会你又点击了 Back 键回到 `MainActivity`，你会发现刚刚输入的文字全部都没了，因为 `MainActivity` 被重新创建了。

如果我们的应用出现了这种情况，是会严重影响用户体验的，所以必须要想想办法解决这个问题。查阅文档可以看出，`Activity` 中还提供了一个 `onSaveInstanceState()` 回调方法，这个方法可以保证在活动被回收之前一定会被调用，因此我们可以通过这个方法来解决活动被回收时临时数据得不到保存的问题。

`onSaveInstanceState()` 方法会携带一个 `Bundle` 类型的参数，`Bundle` 提供了一系列的方法用于保存数据，比如可以使用 `putString()` 方法保存字符串，使用 `.putInt()` 方法保存整型数据，以此类推。每个保存方法需要传入两个参数，第一个参数是键，用于后面从 `Bundle` 中取值，

第二个参数是真正要保存的内容。

在 MainActivity 中添加如下代码就可以将临时数据进行保存：

```
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    String tempData = "Something you just typed";  
    outState.putString("data_key", tempData);  
}
```

数据是已经保存下来了，那么我们应该在哪里进行恢复呢？细心的你也许早就发现，我们一直使用的 `onCreate()` 方法其实也有一个 `Bundle` 类型的参数。这个参数在一般情况下都是 `null`，但是如果在活动被系统回收之前有通过 `onSaveInstanceState()` 方法来保存数据的话，这个参数就会带有之前所保存的全部数据，我们只需要再通过相应的取值方法将数据取出即可。

修改 MainActivity 的 `onCreate()` 方法，如下所示：

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    Log.d(TAG, "onCreate");  
    setContentView(R.layout.activity_main);  
    if (savedInstanceState != null) {  
        String tempData = savedInstanceState.getString("data_key");  
        Log.d(TAG, tempData);  
    }  
    ...  
}
```

取出值之后再做相应的恢复操作就可以了，比如说将文本内容重新赋值到文本输入框上，这里我们只是简单地打印一下。

不知道你有没有察觉，使用 `Bundle` 来保存和取出数据是不是有些似曾相识呢？没错！我们在使用 `Intent` 传递数据时也是用的类似的方法。这里跟你提醒一点，`Intent` 还可以结合 `Bundle` 一起用于传递数据，首先可以把需要传递的数据都保存在 `Bundle` 对象中，然后再将 `Bundle` 对象存放在 `Intent` 里。到了目标活动之后先从 `Intent` 中取出 `Bundle`，再从 `Bundle` 中一一取出数据。具体的代码我就不写了，要学会举一反三哦。

2.5 活动的启动模式

活动的启动模式对你来说应该是个全新的概念，在实际项目中我们应该根据特定的需求为每个活动指定恰当的启动模式。启动模式一共有 4 种，分别是 `standard`、`singleTop`、`singleTask` 和 `singleInstance`，可以在 `AndroidManifest.xml` 中通过给 `<activity>` 标签指定 `android:launchMode` 属性来选择启动模式。下面我们就逐个进行学习。

2.5.1 standard

standard 是活动默认的启动模式，在不进行显式指定的情况下，所有活动都会自动使用这种启动模式。因此，到目前为止我们写过的所有活动都是使用的 standard 模式。经过上一节的学习，你已经知道了 Android 是使用返回栈来管理活动的，在 standard 模式（即默认情况）下，每当启动一个新的活动，它就会在返回栈中入栈，并处于栈顶的位置。对于使用 standard 模式的活动，系统不在乎这个活动是否已经在返回栈中存在，每次启动都会创建该活动的一个新的实例。

我们现在通过实践来体会一下 standard 模式，这次还是准备在 ActivityTest 项目的基础上修改，首先关闭 ActivityLifeCycleTest 项目，打开 ActivityTest 项目。

修改 FirstActivity 中 onCreate() 方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", this.toString());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, FirstActivity.class);
            startActivity(intent);
        }
    });
}
```

代码看起来有些奇怪吧，在 FirstActivity 的基础上启动 FirstActivity。从逻辑上来讲这确实没什么意义，不过我们的重点在于研究 standard 模式，因此不必在意这段代码有什么实际用途。另外我们还在 onCreate() 方法中添加了一行打印信息，用于打印当前活动的实例。

现在重新运行程序，然后在 FirstActivity 界面连续点击两次按钮，可以看到 logcat 中打印信息如图 2.33 所示。

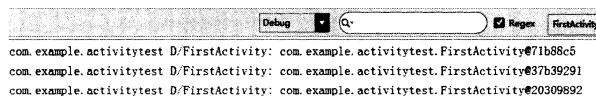


图 2.33 standard 模式下的打印日志

从打印信息中我们就可以看出，每点击一次按钮就会创建出一个新的 FirstActivity 实例。此时返回栈中也会存在 3 个 FirstActivity 的实例，因此你需要按压 3 次 Back 键才能退出程序。

standard 模式的原理示意图，如图 2.34 所示。

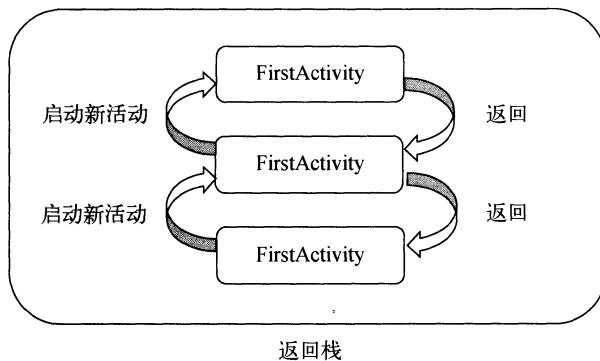


图 2.34 standard 模式示意图

2.5.2 singleTop

可能在有些情况下，你会觉得 standard 模式不太合理。活动明明已经在栈顶了，为什么再次启动的时候还要创建一个新的活动实例呢？别着急，这只是系统默认的一种启动模式而已，你完全可以根据自己的需要进行修改，比如说使用 singleTop 模式。当活动的启动模式指定为 singleTop，在启动活动时如果发现返回栈的栈顶已经是该活动，则认为可以直接使用它，不会再创建新的活动实例。

我们还是通过实践来体会一下，修改 AndroidManifest.xml 中 FirstActivity 的启动模式，如下所示：

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

然后重新运行程序，查看 logcat 会看到已经创建了一个 FirstActivity 的实例，如图 2.35 所示。

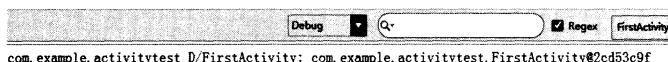


图 2.35 singleTop 模式下的打印日志

但是之后不管你点击多少次按钮都不会再有新的打印信息出现，因为目前 FirstActivity 已经处于返回栈的栈顶，每当想要再启动一个 FirstActivity 时都会直接使用栈顶的活动，因此 FirstActivity 也只会有一个实例，仅按一次 Back 键就可以退出程序。

不过当 FirstActivity 并未处于栈顶位置时，这时再启动 FirstActivity，还是会创建新的实例的。

下面我们来实验一下，修改 FirstActivity 中 onCreate() 方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", this.toString());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
            startActivity(intent);
        }
    });
}
```

这次我们点击按钮后启动的是 SecondActivity。然后修改 SecondActivity 中 onCreate() 方法的代码，如下所示：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("SecondActivity", this.toString());
    setContentView(R.layout.second_layout);
    Button button2 = (Button) findViewById(R.id.button_2);
    button2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(SecondActivity.this, FirstActivity.class);
            startActivity(intent);
        }
    });
}
```

我们在 SecondActivity 中的按钮点击事件里又加入了启动 FirstActivity 的代码。现在重新运行程序，在 FirstActivity 界面点击按钮进入到 SecondActivity，然后在 SecondActivity 界面点击按钮，又会重新进入到 FirstActivity。

查看 logcat 中的打印信息，如图 2.36 所示。



图 2.36 singleTop 模式下的打印日志

可以看到系统创建了两个不同的 FirstActivity 实例，这是由于在 SecondActivity 中再次启动 FirstActivity 时，栈顶活动已经变成了 SecondActivity，因此会创建一个新的 FirstActivity 实例。现在按下 Back 键会返回到 SecondActivity，再次按下 Back 键又会回到 FirstActivity，再按一次

Back 键才会退出程序。

singleTop 模式的原理示意图，如图 2.37 所示。

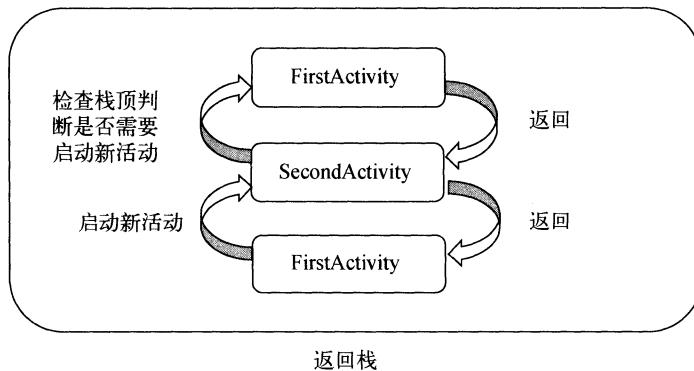


图 2.37 singleTop 模式示意图

2.5.3 singleTask

使用 singleTop 模式可以很好地解决重复创建栈顶活动的问题，但是正如你在上一节所看到的，如果该活动并没有处于栈顶的位置，还是可能会创建多个活动实例的。那么有没有什么办法可以让某个活动在整个应用程序的上下文中只存在一个实例呢？这就要借助 singleTask 模式来实现了。当活动的启动模式指定为 singleTask，每次启动该活动时系统首先会在返回栈中检查是否存在该活动的实例，如果发现已经存在则直接使用该实例，并把在这个活动之上的所有活动统统出栈，如果没有发现就会创建一个新的活动实例。

我们还是通过代码来更加直观地理解一下。修改 AndroidManifest.xml 中 FirstActivity 的启动模式：

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

然后在 FirstActivity 中添加 onRestart() 方法，并打印日志：

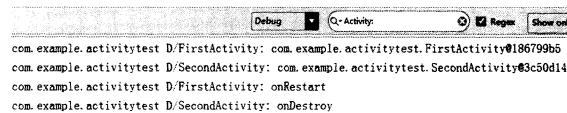
```
@Override
protected void onRestart() {
    super.onRestart();
    Log.d("FirstActivity", "onRestart");
}
```

最后在 SecondActivity 中添加 `onDestroy()` 方法，并打印日志：

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

现在重新运行程序，在 FirstActivity 界面点击按钮进入到 SecondActivity，然后在 SecondActivity 界面点击按钮，又会重新进入到 FirstActivity。

查看 logcat 中的打印信息，如图 2.38 所示。



```
com.example.activitytest D/FirstActivity: com.example.activitytest.FirstActivity@186799b5
com.example.activitytest D/SecondActivity: com.example.activitytest.SecondActivity@3c50d14
com.example.activitytest D/FirstActivity: onRestart
com.example.activitytest D/SecondActivity: onDestroy
```

图 2.38 singleTask 模式下的打印日志

其实从打印信息中就可以明显看出了，在 SecondActivity 中启动 FirstActivity 时，会发现返回栈中已经存在一个 FirstActivity 的实例，并且是在 SecondActivity 的下面，于是 SecondActivity 会从返回栈中出栈，而 FirstActivity 重新成为了栈顶活动，因此 FirstActivity 的 `onRestart()` 方法和 SecondActivity 的 `onDestroy()` 方法会得到执行。现在返回栈中应该只剩下一个 FirstActivity 的实例了，按一下 Back 键就可以退出程序。

singleTask 模式的原理示意图，如图 2.39 所示。

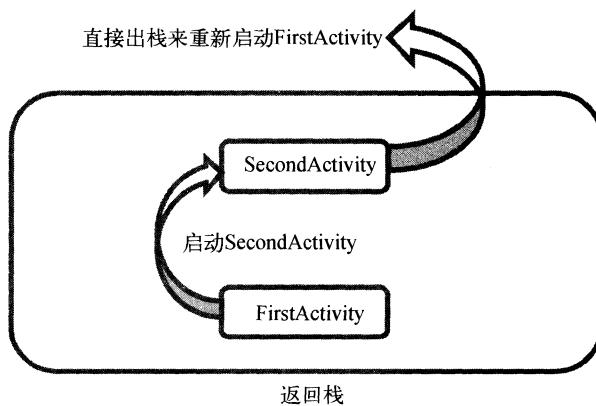


图 2.39 singleTask 模式示意图

2.5.4 singleInstance

singleInstance 模式应该算是 4 种启动模式中最特殊也最复杂的一个了，你也需要多花点功夫来理解这个模式。不同于以上 3 种启动模式，指定为 singleInstance 模式的活动会启用一个新的返

回栈来管理这个活动（其实如果 singleTask 模式指定了不同的 taskAffinity，也会启动一个新的返回栈）。那么这样做有什么意义呢？想象以下场景，假设我们的程序中有一个活动是允许其他程序调用的，如果我们想实现其他程序和我们的程序可以共享这个活动的实例，应该如何实现呢？使用前面 3 种启动模式肯定是做不到的，因为每个应用程序都会有自己的返回栈，同一个活动在不同的返回栈中入栈时必然是创建了新的实例。而使用 singleInstance 模式就可以解决这个问题，在这种模式下会有一个单独的返回栈来管理这个活动，不管是哪个应用程序来访问这个活动，都共用的同一个返回栈，也就解决了共享活动实例的问题。

为了帮助你更好地理解这种启动模式，我们还是来实践一下。修改 AndroidManifest.xml 中 SecondActivity 的启动模式：

```
<activity android:name=".SecondActivity"
    android:launchMode="singleInstance">
    <intent-filter>
        <action android:name="com.example.activitytest.ACTION_START" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="com.example.activitytest.MY_CATEGORY" />
    </intent-filter>
</activity>
```

我们先将 SecondActivity 的启动模式指定为 singleInstance，然后修改 FirstActivity 中 onCreate() 方法的代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("FirstActivity", "Task id is " + getTaskId());
    setContentView(R.layout.first_layout);
    Button button1 = (Button) findViewById(R.id.button_1);
    button1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
            startActivity(intent);
        }
    });
}
```

在 onCreate() 方法中打印了当前返回栈的 id。然后修改 SecondActivity 中 onCreate() 方法的代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("SecondActivity", "Task id is " + getTaskId());
    setContentView(R.layout.second_layout);
    Button button2 = (Button) findViewById(R.id.button_2);
    button2.setOnClickListener(new View.OnClickListener() {
        @Override
```

```

        public void onClick(View v) {
            Intent intent = new Intent(SecondActivity.this, ThirdActivity.class);
            startActivity(intent);
        }
    });
}

```

同样在 `onCreate()` 方法中打印了当前返回栈的 id，然后又修改了按钮点击事件的代码，用于启动 `ThirdActivity`。最后修改 `ThirdActivity` 中 `onCreate()` 方法的代码：

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("ThirdActivity", "Task id is " + getTaskId());
    setContentView(R.layout.third_layout);
}

```

仍然是在 `onCreate()` 方法中打印了当前返回栈的 id。现在重新运行程序，在 `FirstActivity` 界面点击按钮进入到 `SecondActivity`，然后在 `SecondActivity` 界面点击按钮进入到 `ThirdActivity`。

查看 logcat 中的打印信息，如图 2.40 所示。

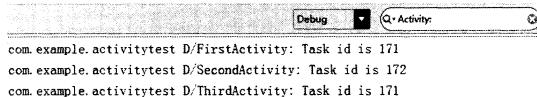


图 2.40 singleInstance 模式下的打印日志

可以看到，`SecondActivity` 的 `Task id` 不同于 `FirstActivity` 和 `ThirdActivity`，这说明 `SecondActivity` 确实是存放在一个单独的返回栈里的，而且这个栈中只有 `SecondActivity` 这一个活动。

然后我们按下 Back 键进行返回，你会发现 `ThirdActivity` 竟然直接返回到了 `FirstActivity`，再按下 Back 键又会返回到 `SecondActivity`，再按下 Back 键才会退出程序，这是为什么呢？其实原理很简单，由于 `FirstActivity` 和 `ThirdActivity` 是存放在同一个返回栈里的，当在 `ThirdActivity` 的界面按下 Back 键，`ThirdActivity` 会从返回栈中出栈，那么 `FirstActivity` 就成为了栈顶活动显示在界面上，因此也就出现了从 `ThirdActivity` 直接返回到 `FirstActivity` 的情况。然后在 `FirstActivity` 界面再次按下 Back 键，这时当前的返回栈已经空了，于是就显示了另一个返回栈的栈顶活动，即 `SecondActivity`。最后再次按下 Back 键，这时所有返回栈都已经空了，也就自然退出了程序。

`singleInstance` 模式的原理示意图，如图 2.41 所示。

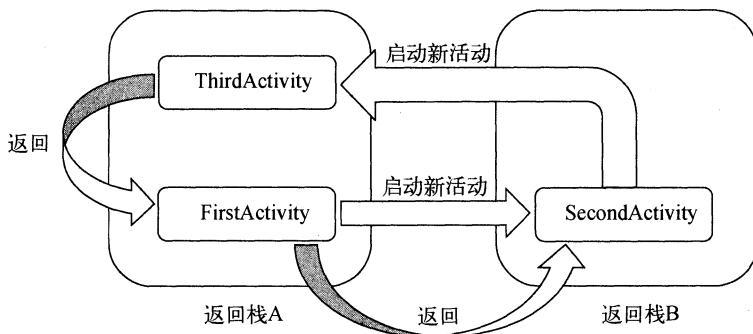


图 2.41 singleInstance 模式示意图

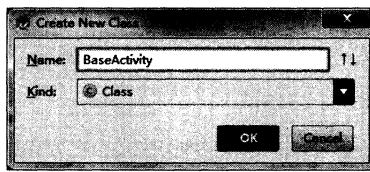
2.6 活动的最佳实践

你已经掌握了关于活动非常多的知识，不过恐怕离能够完全灵活运用还有一段距离。虽然知识点只有这么多，但运用的技巧却是多种多样。所以，在这里我准备教你几种关于活动的最佳实践技巧，这些技巧在你以后的开发工作当中将会非常受用。

2.6.1 知晓当前是在哪一个活动

这个技巧将教会你如何根据程序当前的界面就能判断出这是哪一个活动。可能你会觉得挺纳闷的，我自己写的代码怎么会不知道这是哪一个活动呢？很不幸的是，在你真正进入到企业之后，更有可能的是接手一份别人写的代码，因为你刚进公司就正好有一个新项目启动的概率并不高。阅读别人的代码时有一个很头疼的问题，就是当你需要在某个界面上修改一些非常简单的东西时，却半天找不到这个界面对应的活动是哪一个。学会了本节的技巧之后，这对你来说就再也不是难题了。

我们还是在 ActivityTest 项目的基础上修改，首先需要新建一个 `BaseActivity` 类。右击 com.example.activitytest 包 → New → Java Class，在弹出的窗口中输入 `BaseActivity`，如图 2.42 所示。

图 2.42 创建 `BaseActivity` 类

注意这里 `BaseActivity` 和普通活动的创建方式不一样，因为我们不需要让 `BaseActivity` 在 `AndroidManifest.xml` 中注册，所以选择创建一个普通的 Java 类就可以了。然后让 `BaseActivity` 继承自 `AppCompatActivity`，并重写 `onCreate()` 方法，如下所示：

```

public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("BaseActivity", getClass().getSimpleName());
    }
}

```

我们在 `onCreate()` 方法中获取了当前实例的类名，并通过 `Log` 打印了出来。

接下来我们需要让 `BaseActivity` 成为 `ActivityTest` 项目中所有活动的父类。修改 `FirstActivity`、`SecondActivity` 和 `ThirdActivity` 的继承结构，让它们不再继承自 `AppCompatActivity`，而是继承自 `BaseActivity`。而由于 `BaseActivity` 又是继承自 `AppCompatActivity` 的，所以项目中所有活动的现有功能并不受影响，它们仍然完全继承了 `Activity` 中的所有特性。

现在重新运行程序，然后通过点击按钮分别进入到 `FirstActivity`、`SecondActivity` 和 `ThirdActivity` 的界面，这时观察 `logcat` 中的打印信息，如图 2.43 所示。

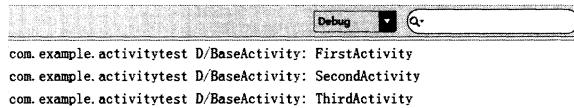


图 2.43 `BaseActivity` 中的打印日志

现在每当我们进入到一个活动的界面，该活动的类名就会被打印出来，这样我们就可以时刻知晓当前界面对应的是哪一个活动了。

2.6.2 随时随地退出程序

如果目前你手机的界面还停留在 `ThirdActivity`，你会发现当前想退出程序是非常不方便的，需要按压 3 次 `Back` 键才行。按 `Home` 键只是把程序挂起，并没有退出程序。其实这个问题就足以引起你的思考，如果我们的程序需要一个注销或者退出的功能该怎么办呢？必须要有一个随时随地都能退出程序的方案才行。

其实解决思路也很简单，只需要用一个专门的集合类对所有的活动进行管理就可以了，下面我们就来实现一下。

新建一个 `ActivityCollector` 类作为活动管理器，代码如下所示：

```

public class ActivityCollector {

    public static List<Activity> activities = new ArrayList<>();

    public static void addActivity(Activity activity) {
        activities.add(activity);
    }
}

```

```

public static void removeActivity(Activity activity) {
    activities.remove(activity);
}

public static void finishAll() {
    for (Activity activity : activities) {
        if (!activity.isFinishing()) {
            activity.finish();
        }
    }
}
}

```

在活动管理器中，我们通过一个 List 来暂存活动，然后提供了一个 `addActivity()` 方法用于向 List 中添加一个活动，提供了一个 `removeActivity()` 方法用于从 List 中移除活动，最后提供了一个 `finishAll()` 方法用于将 List 中存储的活动全部销毁掉。

接下来修改 `BaseActivity` 中的代码，如下所示：

```

public class BaseActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d("BaseActivity", getClass().getSimpleName());
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        ActivityCollector.removeActivity(this);
    }
}

```

在 `BaseActivity` 的 `onCreate()` 方法中调用了 `ActivityCollector` 的 `addActivity()` 方法，表明将目前正在创建的活动添加到活动管理器里。然后在 `BaseActivity` 中重写 `onDestroy()` 方法，并调用了 `ActivityCollector` 的 `removeActivity()` 方法，表明将一个马上要销毁的活动从活动管理器里移除。

从此以后，不管你想在什么地方退出程序，只需要调用 `ActivityCollector.finishAll()` 方法就可以了。例如在 `ThirdActivity` 界面想通过点击按钮直接退出程序，只需将代码改成如下所示：

```

public class ThirdActivity extends BaseActivity {

    @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d("ThirdActivity", "Task id is " + getTaskId());
    setContentView(R.layout.third_layout);
    Button button3 = (Button) findViewById(R.id.button_3);
    button3.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ActivityCollector.finishAll();
        }
    });
}

}

```

当然你还可以在销毁所有活动的代码后面再加上杀掉当前进程的代码，以保证程序完全退出，杀掉进程的代码如下所示：

```
android.os.Process.killProcess(android.os.Process.myPid());
```

其中，`killProcess()`方法用于杀掉一个进程，它接收一个进程 `id` 参数，我们可以通过 `myPid()` 方法来获得当前程序的进程 `id`。需要注意的是，`killProcess()` 方法只能用于杀掉当前程序的进程，我们不能使用这个方法去杀掉其他程序。

2.6.3 启动活动的最佳写法

启动活动的方法相信你已经非常熟悉了，首先通过 `Intent` 构建出当前的“意图”，然后调用 `startActivity()` 或 `startActivityForResult()` 方法将活动启动起来，如果有数据需要从一个活动传递到另一个活动，也可以借助 `Intent` 来完成。

假设 `SecondActivity` 中需要用到两个非常重要的字符串参数，在启动 `SecondActivity` 的时候必须要传递过来，那么我们很容易会写出如下代码：

```

Intent intent = new Intent(FirstActivity.this, SecondActivity.class);
intent.putExtra("param1", "data1");
intent.putExtra("param2", "data2");
startActivity(intent);

```

这样写是完全正确的，不管是从语法上还是规范上，只是在真正的项目开发中经常会有对接的问题出现。比如 `SecondActivity` 并不是由你开发的，但现在你负责的部分需要有启动 `SecondActivity` 这个功能，而你却不清楚启动这个活动需要传递哪些数据。这时无非就有两种办法，一个是你自己去阅读 `SecondActivity` 中的代码，二是询问负责编写 `SecondActivity` 的同事。你会不会觉得很麻烦呢？其实只需要换一种写法，就可以轻松解决掉上面的窘境。

修改 `SecondActivity` 中的代码，如下所示：

```
public class SecondActivity extends BaseActivity {
```

```
public static void actionStart(Context context, String data1, String data2) {  
    Intent intent = new Intent(context, SecondActivity.class);  
    intent.putExtra("param1", data1);  
    intent.putExtra("param2", data2);  
    context.startActivity(intent);  
}  
...  
}
```

我们在 SecondActivity 中添加了一个 `actionStart()` 方法，在这个方法中完成了 Intent 的构建，另外所有 SecondActivity 中需要的数据都是通过 `actionStart()` 方法的参数传递过来的，然后把它们存储到 Intent 中，最后调用 `startActivity()` 方法启动 SecondActivity。

这样写的好处在哪里呢？最重要的一点就是一目了然，SecondActivity 所需要的数据在方法参数中全部体现出来了，这样即使不用阅读 SecondActivity 中的代码，不去询问负责编写 SecondActivity 的同事，你也可以非常清晰地知道启动 SecondActivity 需要传递哪些数据。另外，这样写还简化了启动活动的代码，现在只需要一行代码就可以启动 SecondActivity，如下所示：

```
button1.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        SecondActivity.actionStart(FirstActivity.this, "data1", "data2");  
    }  
});
```

养成一个良好的习惯，给你编写的每个活动都添加类似的启动方法，这样不仅可以让启动活动变得非常简单，还可以节省不少有同事过来询问你的时间。

2.7 小结与点评

真是好疲惫啊！没错，学习了这么多的东西不疲惫才怪呢。但是，你内心那种掌握了知识的喜悦感相信也是无法掩盖的。本章的收获非常多啊，不管是理论型还是实践型的东西都涉及了，从活动的基本用法，到启动活动和传递数据的方式，再到活动的生命周期，以及活动的启动模式，你几乎已经学会了关于活动所有重要的知识点。另外在本章的最后，还学习了几种可以应用在活动中的最佳实践技巧，毫不夸张地说，你在 Android 活动方面已经算是一个小高手了。

不过你的 Android 旅途才刚刚开始呢，后面需要学习的东西还很多，也许会比现在还累，一定要做好心理准备哦。总体来说，我给你现在的状态打满分，毕竟你已经学会了那么多的东西，也是时候放松一下了。自己适当控制一下休息的时间，然后我们继续前进吧！

第3章

软件也要拼脸蛋——UI 开发的点点滴滴

我一直都认为程序员在软件的审美方面普遍都比较差，至少我个人就是如此。如果说要追究其根本原因，我觉得这是由程序员的工作性质所导致的。每当我们看到一个软件时，不会像普通用户那样仅仅是关注一下它的界面和功能，而是会不自觉地思考这些功能是如何实现的。很多在普通用户看来理所应当的功能，背后可能却需要非常复杂的逻辑来完成，以至于当别人唾骂一句“这软件做得真丑”的时候，我们还可能赞叹一句“这功能做得好牛啊”！

不过缺乏审美观毕竟不是一件值得炫耀的事情，在软件开发过程中，界面设计和功能开发同样重要。界面美观的应用程序不仅可以大大增加用户粘性，还能帮我们吸引到更多的新用户。而 Android 也给我们提供了大量的 UI 开发工具，只要合理地使用它们，就可以编写出各种各样漂亮的界面。

在这里，我无法教会你如何提升自己的审美观，但我可以教会你怎样使用 Android 提供的 UI 开发工具来编写程序界面。你在上一章中反反复复地使用那几个按钮，想必都快要吐了吧，本章我们就来学习更多的 UI 开发方面的知识。

3.1 如何编写程序界面

Android 中有多种编写程序界面的方式可供选择。Android Studio 和 Eclipse 中都提供了相应的可视化编辑器，允许使用拖放控件的方式来编写布局，并能在视图上直接修改控件的属性。不过我并不推荐你使用这种方式来编写界面，因为可视化编辑工具并不利于你去真正了解界面背后的实现原理。通过这种方式制作出的界面通常不具有很好的屏幕适配性，而且当需要编写较为复杂的界面时，可视化编辑工具将很难胜任。因此本书中所有的界面都将通过最基本的方式去实现，即编写 XML 代码。等你完全掌握了使用 XML 来编写界面的方法之后，不管是进行高复杂度的界面实现，还是分析和修改当前现有界面，对你来说都将是手到擒来。

讲了这么多理论的东西，也是时候学习一下到底如何编写程序界面了，下面我们就从 Android 中几种常见的控件开始吧。

3.2 常用控件的使用方法

Android 提供了大量的 UI 控件，合理地使用这些控件就可以非常轻松地编写出相当不错的界面，下面我们就挑选几种常用的控件，详细介绍一下它们的使用方法。

首先新建一个 UIWidgetTest 项目，简单起见，我们还是允许 Android Studio 自动创建活动，活动名和布局名都使用默认值。

3.2.1 TextView

TextView 可以说是 Android 中最简单的一个控件了，你在前面其实已经和它打过一些交道了。它主要用于在界面上显示一段文本信息，比如你在第 1 章看到的“Hello world!”。下面我们就来看一看关于 TextView 的更多用法。

修改 activity_main.xml 中的代码，如下所示：

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

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is TextView" />

</LinearLayout>
```

外面的 LinearLayout 先忽略不看，在 TextView 中我们使用 `android:id` 给当前控件定义了一个唯一标识符，这个属性在上一章中已经讲解过了。然后使用 `android:layout_width` 和 `android:layout_height` 指定了控件的宽度和高度。Android 中所有的控件都具有这两个属性，可选值有 3 种：`match_parent`、`fill_parent` 和 `wrap_content`。其中 `match_parent` 和 `fill_parent` 的意义相同，现在官方更加推荐使用 `match_parent`。`match_parent` 表示让当前控件的大小和父布局的大小一样，也就是由父布局来决定当前控件的大小。`wrap_content` 表示让当前控件的大小能够刚好包含住里面的内容，也就是由控件内容决定当前控件的大小。所以上面的代码就表示让 TextView 的宽度和父布局一样宽，也就是手机屏幕的宽度，让 TextView 的高度足够包含住里面的内容就行。当然除了使用上述值，你也可以对控件的宽和高指定一个固定的大小，但是这样做有时会在不同手机屏幕的适配方面出现问题。

接下来我们通过 `android:text` 指定 TextView 中显示的文本内容，现在运行程序，效果如图 3.1 所示。



图 3.1 TextView 运行效果

虽然指定的文本内容正常显示了，不过我们好像没看出来 TextView 的宽度是和屏幕一样宽的。其实这是由于 TextView 中的文字默认是居左上角对齐的，虽然 TextView 的宽度充满了整个屏幕，可是由于文字内容不够长，所以从效果上完全看不出来。现在我们修改 TextView 的文字对齐方式，如下所示：

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

    <TextView
        android:id="@+id/text_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="This is TextView" />

</LinearLayout>
```

我们使用 `android:gravity` 来指定文字的对齐方式，可选值有 `top`、`bottom`、`left`、`right`、`center` 等，可以用“|”来同时指定多个值，这里我们指定的 `center`，效果等同于 `center_vertical|center_horizontal`，表示文字在垂直和水平方向都居中对齐。现在重新运行程序，效果如图 3.2 所示。

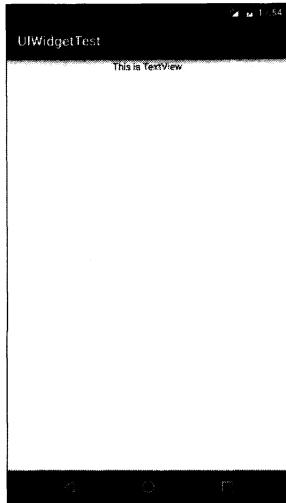


图 3.2 TextView 居中效果

这也说明了 TextView 的宽度确实是和屏幕宽度一样的。

另外我们还可以对 TextView 中文字的大小和颜色进行修改，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    <TextView  
        android:id="@+id/text_view"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:gravity="center"  
        android:textSize="24sp"  
        android:textColor="#00ff00"  
        android:text="This is TextView" />  
  
</LinearLayout>
```

通过 android:textSize 属性可以指定文字的大小，通过 android:textColor 属性可以指定文字的颜色，在 Android 中字体大小使用 sp 作为单位。重新运行程序，效果如图 3.3 所示。



图 3.3 改变 TextView 文字大小和颜色的效果

当然 TextView 中还有很多其他的属性，这里就不再一一介绍了，用到的时候去查阅文档就可以了。

3.2.2 Button

Button 是程序用于和用户进行交互的一个重要控件，相信你对这个控件已经非常熟悉了，因为我们在上一章用了太多次 Button。它可配置的属性和 TextView 是差不多的，我们可以在 activity_main.xml 中这样加入 Button：

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

    ...

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

</LinearLayout>
```

加入 Button 之后的界面如图 3.4 所示。



图 3.4 Button 运行效果

细心的你可能会留意到，我们在布局文件里面设置的文字是“Button”，但最终的显示结果却是“BUTTON”。这是由于系统会对 Button 中的所有英文字母自动进行大写转换，如果这不是你想要的效果，可以使用如下配置来禁用这一默认特性：

```
<Button
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button"
    android:textAllCaps="false" />
```

接下来我们可以在 MainActivity 中为 Button 的点击事件注册一个监听器，如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // 在此处添加逻辑
            }
        });
    }
}
```

这样每当点击按钮时，就会执行监听器中的 onClick() 方法，我们只需要在这个方法中加入待处理的逻辑就行了。如果你不喜欢使用匿名类的方式来注册监听器，也可以使用实现接口的方

式来进行注册，代码如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                // 在此处添加逻辑
                break;
            default:
                break;
        }
    }
}
```

这两种写法都可以实现对按钮点击事件的监听，至于使用哪一种就全凭你的喜好了。

3.2.3 EditText

EditText 是程序用于和用户进行交互的另一个重要控件，它允许用户在控件里输入和编辑内容，并可以在程序中对这些内容进行处理。EditText 的应用场景非常普遍，在进行发短信、发微博、聊 QQ 等操作时，你不得不使用 EditText。那我们来看一看如何在界面上加入 EditText 吧，修改 activity_main.xml 中的代码，如下所示：

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

    ...

    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

其实看到这里，我估计你已经总结出 Android 控件的使用规律了，用法基本上都很相似：给控件定义一个 id，再指定控件的宽度和高度，然后再适当加入一些控件特有的属性就差不多了。

所以使用 XML 来编写界面其实一点都不难，完全可以不用借助任何可视化工具来实现。现在重新运行一下程序，EditText 就已经在界面上显示出来了，并且我们是可以在里面输入内容的，如图 3.5 所示。

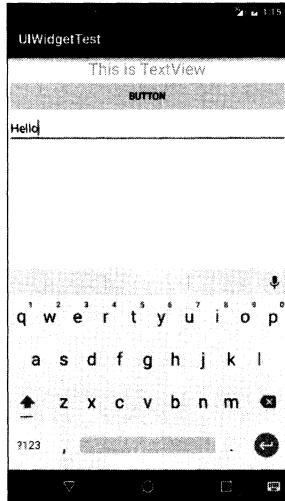


图 3.5 EditText 运行效果

细心的你平时应该会留意到，一些做得比较人性化的软件会在输入框里显示一些提示性的文字，然后一旦用户输入了任何内容，这些提示性的文字就会消失。这种提示功能在 Android 里是非常容易实现的，我们甚至不需要做任何的逻辑控制，因为系统已经帮我们都处理好了。修改 activity_main.xml，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    ...  
  
    <EditText  
        android:id="@+id/edit_text"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:hint="Type something here"  
    />  
  
</LinearLayout>
```

这里使用 `android:hint` 属性指定了一段提示性的文本，然后重新运行程序，效果如图 3.6 所示。



图 3.6 EditText 设置 hint 效果

可以看到，EditText 中显示了一段提示性文本，然后当我们输入任何内容时，这段文本就会自动消失。

不过，随着输入的内容不断增多，EditText 会被不断地拉长。这时由于 EditText 的高度指定的是 `wrap_content`，因此它总能包含住里面的内容，但是当输入的内容过多时，界面就会变得非常难看。我们可以使用 `android:maxLines` 属性来解决这个问题，修改 `activity_main.xml`，如下所示：

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

    ...
    <EditText
        android:id="@+id/edit_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Type something here"
        android:maxLines="2"
        />
</LinearLayout>
```

这里通过 `android:maxLines` 指定了 EditText 的最大行数为两行，这样当输入的内容超过两行时，文本就会向上滚动，而 EditText 则不会再继续拉伸，如图 3.7 所示。



图 3.7 EditText 设置 maxLines 效果

我们还可以结合使用 `EditText` 与 `Button` 来完成一些功能，比如通过点击按钮来获取 `EditText` 中输入的内容。修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
  
    private EditText editText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button button = (Button) findViewById(R.id.button);  
        editText = (EditText) findViewById(R.id.edit_text);  
        button.setOnClickListener(this);  
    }  
  
    @Override  
    public void onClick(View v) {  
        switch (v.getId()) {  
            case R.id.button:  
                String inputText = editText.getText().toString();  
                Toast.makeText(MainActivity.this, inputText,  
                Toast.LENGTH_SHORT).show();  
                break;  
            default:  
                break;  
        }  
    }  
}
```

首先通过 `findViewById()` 方法得到 `EditText` 的实例，然后在按钮的点击事件里调用 `EditText` 的 `getText()` 方法获取到输入的内容，再调用 `toString()` 方法转换成字符串，最后还是老方法，使用 `Toast` 将输入的内容显示出来。

重新运行程序，在 `EditText` 中输入一段内容，然后点击按钮，效果如图 3.8 所示。

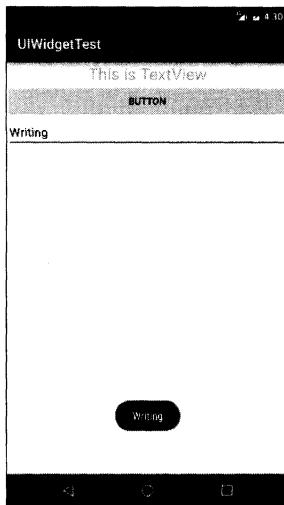


图 3.8 获取 `EditText` 中输入的内容

3.2.4 ImageView

`ImageView` 是用于在界面上展示图片的一个控件，它可以让我们的程序界面变得更加丰富多彩。学习这个控件需要提前准备好一些图片，图片通常都是放在以“`drawable`”开头的目录下的。目前我们的项目中有一个空的 `drawable` 目录，不过由于这个目录没有指定具体的分辨率，所以一般不使用它来放置图片。这里我们在 `res` 目录下新建一个 `drawable-xhdpi` 目录，然后将事先准备好的两张图片 `img_1.png` 和 `img_2.png` 复制到该目录当中。

接下来修改 `activity_main.xml`，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
    ...  
  
<ImageView  
    android:id="@+id/image_view"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:src="@drawable/img_1 "
```

```
/>
</LinearLayout>
```

可以看到，这里使用 `android:src` 属性给 `ImageView` 指定了一张图片。由于图片的宽和高都是未知的，所以将 `ImageView` 的宽和高都设定为 `wrap_content`，这样就保证了不管图片的尺寸是多少，图片都可以完整地展示出来。重新运行程序，效果如图 3.9 所示。

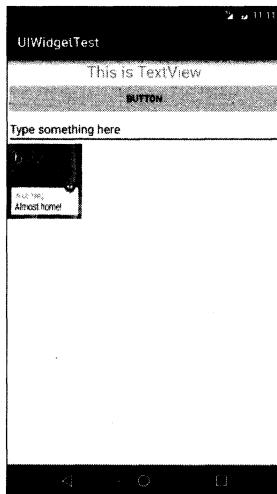


图 3.9 ImageView 运行效果

我们还可以在程序中通过代码动态地更改 `ImageView` 中的图片，然后修改 `MainActivity` 的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private EditText editText;
    private ImageView imageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView = (ImageView) findViewById(R.id.image_view);
        button.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                imageView.setImageResource(R.drawable.img_2);
        }
    }
}
```

```

        break;
    default:
        break;
    }
}

}

```

在按钮的点击事件里，通过调用 ImageView 的 `setImageResource()` 方法将显示的图片改成 `img_2`，现在重新运行程序，然后点击一下按钮，就可以看到 ImageView 中显示的图片改变了，如图 3.10 所示。

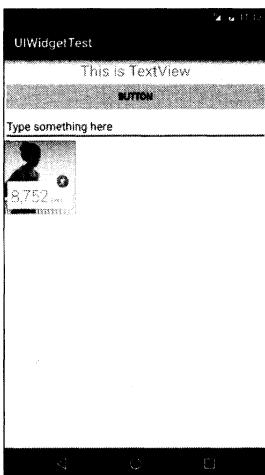


图 3.10 动态更改 ImageView 中的图片

3.2.5 ProgressBar

ProgressBar 用于在界面上显示一个进度条，表示我们的程序正在加载一些数据。它的用法也非常简单，修改 `activity_main.xml` 中的代码，如下所示：

```

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

    ...

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        />

</LinearLayout>

```

重新运行程序，会看到屏幕中有一个圆形进度条正在旋转，如图 3.11 所示。

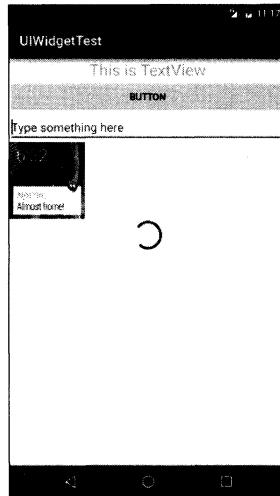


图 3.11 ProgressBar 运行效果

这时你可能会问，旋转的进度条表明我们的程序正在加载数据，那数据总会有加载完的时候吧？如何才能让进度条在数据加载完成时消失呢？这里我们就需要用到一个新的知识点：Android 控件的可见属性。所有的 Android 控件都具有这个属性，可以通过 `android:visibility` 进行指定，可选值有 3 种：`visible`、`invisible` 和 `gone`。`visible` 表示控件是可见的，这个值是默认值，不指定 `android:visibility` 时，控件都是可见的。`invisible` 表示控件不可见，但是它仍然占据着原来的位置和大小，可以理解成控件变成透明状态了。`gone` 则表示控件不仅不可见，而且不再占用任何屏幕空间。我们还可以通过代码来设置控件的可见性，使用的是 `setVisibility()` 方法，可以传入 `View.VISIBLE`、`View.INVISIBLE` 和 `View.GONE` 这 3 种值。

接下来我们就来尝试实现，点击一下按钮让进度条消失，再点击一下按钮让进度条出现的这种效果。修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    private EditText editText;
    private ImageView imageView;
    private ProgressBar progressBar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        editText = (EditText) findViewById(R.id.edit_text);
        imageView = (ImageView) findViewById(R.id.image_view);
    }
}
```

```

    progressBar = (ProgressBar) findViewById(R.id.progress_bar);
    button.setOnClickListener(this);
}

@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button:
            if (progressBar.getVisibility() == View.GONE) {
                progressBar.setVisibility(View.VISIBLE);
            } else {
                progressBar.setVisibility(View.GONE);
            }
            break;
        default:
            break;
    }
}
}

```

在按钮的点击事件中，我们通过 `getVisibility()` 方法来判断 `ProgressBar` 是否可见，如果可见就将 `ProgressBar` 隐藏掉，如果不可见就将 `ProgressBar` 显示出来。重新运行程序，然后不断地点击按钮，你就会看到进度条会在显示与隐藏之间来回切换。

另外，我们还可以给 `ProgressBar` 指定不同的样式，刚刚是圆形进度条，通过 `style` 属性可以将它指定成水平进度条，修改 `activity_main.xml` 中的代码，如下所示：

```

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

    ...

    <ProgressBar
        android:id="@+id/progress_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="?android:attr/progressBarStyleHorizontal"
        android:max="100"
    />

</LinearLayout>

```

指定成水平进度条后，我们还可以通过 `android:max` 属性给进度条设置一个最大值，然后在代码中动态地更改进度条的进度。修改 `MainActivity` 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...
    @Override

```

```

public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button:
            int progress = progressBar.getProgress();
            progress = progress + 10;
            progressBar.setProgress(progress);
            break;
        default:
            break;
    }
}

```

每点击一次按钮，我们就获取进度条的当前进度，然后在现有的进度上加 10 作为更新后的进度。重新运行程序，点击数次按钮后，效果如图 3.12 所示。



图 3.12 ProgressBar 水平样式效果

ProgressBar 还有几种其他的样式，你可以自己去尝试一下。

3.2.6 AlertDialog

AlertDialog 可以在当前的界面弹出一个对话框，这个对话框是置顶于所有界面元素之上的，能够屏蔽掉其他控件的交互能力，因此 AlertDialog 一般都是用于提示一些非常重要的内容或者警告信息。比如为了防止用户误删重要内容，在删除前弹出一个确认对话框。下面我们来学习一下它的用法，修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...
    @Override

```

```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.button:  
            AlertDialog.Builder dialog = new AlertDialog.Builder (MainActivity.  
                this);  
            dialog.setTitle("This is Dialog");  
            dialog.setMessage("Something important.");  
            dialog.setCancelable(false);  
            dialog.setPositiveButton("OK", new DialogInterface.  
                OnClickListener() {  
                    @Override  
                    public void onClick(DialogInterface dialog, int which) {  
                    }  
                });  
            dialog.setNegativeButton("Cancel", new DialogInterface.  
                OnClickListener() {  
                    @Override  
                    public void onClick(DialogInterface dialog, int which) {  
                    }  
                });  
            dialog.show();  
            break;  
        default:  
            break;  
    }  
}
```

首先通过 `AlertDialog.Builder` 创建一个 `AlertDialog` 的实例, 然后可以为这个对话框设置标题、内容、可否取消等属性, 接下来调用 `setPositiveButton()` 方法为对话框设置确定按钮的点击事件, 调用 `setNegativeButton()` 方法设置取消按钮的点击事件, 最后调用 `show()` 方法将对话框显示出来。重新运行程序, 点击按钮后, 效果如图 3.13 所示。



图 3.13 `AlertDialog` 运行效果

3.2.7 ProgressDialog

ProgressDialog 和 AlertDialog 有点类似，都可以在界面上弹出一个对话框，都能够屏蔽掉其他控件的交互能力。不同的是，ProgressDialog 会在对话框中显示一个进度条，一般用于表示当前操作比较耗时，让用户耐心地等待。它的用法和 AlertDialog 也比较相似，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {  
    ...  
    @Override  
    public void onClick(View v) {  
        switch (v.getId()) {  
            case R.id.button:  
                ProgressDialog progressDialog = new ProgressDialog(MainActivity.this);  
                progressDialog.setTitle("This is ProgressDialog");  
                progressDialog.setMessage("Loading...");  
                progressDialog.setCancelable(true);  
                progressDialog.show();  
                break;  
            default:  
                break;  
        }  
    }  
}
```

可以看到，这里也是先构建出一个 ProgressDialog 对象，然后同样可以设置标题、内容、可否取消等属性，最后也是通过调用 show() 方法将 ProgressDialog 显示出来。重新运行程序，点击按钮后，效果如图 3.14 所示。



图 3.14 ProgressDialog 运行效果

注意，如果在 `setCancelable()` 中传入了 `false`，表示 `ProgressDialog` 是不能通过 Back 键取消掉的，这时你就一定要在代码中做好控制，当数据加载完成后必须要调用 `ProgressDialog` 的 `dismiss()` 方法来关闭对话框，否则 `ProgressDialog` 将会一直存在。

好了，关于 Android 常用控件的使用，我要讲的就只有这么多。一节内容就想覆盖 Android 控件所有的相关知识不太现实，同样一口气就想学会所有 Android 控件的使用方法也不太现实。本节所讲的内容对于你来说只是起到了一个引导的作用，你还需要在以后的学习和工作中不断地摸索，通过查阅文档以及网上搜索的方式学习更多控件的更多用法。当然，当本书后面涉及一些我们前面没学过的控件和相关用法时，我仍然会在相应的章节做详细的讲解。

3.3 详解 4 种基本布局

一个丰富的界面总是要由很多个控件组成的，那我们如何才能让各个控件都有条不紊地摆放在界面上，而不是乱糟糟的呢？这就需要借助布局来实现了。布局是一种可用于放置很多控件的容器，它可以按照一定的规律调整内部控件的位置，从而编写出精美的界面。当然，布局的内部除了放置控件外，也可以放置布局，通过多层布局的嵌套，我们就能够完成一些比较复杂的界面实现，图 3.15 很好地展示了它们之间的关系。

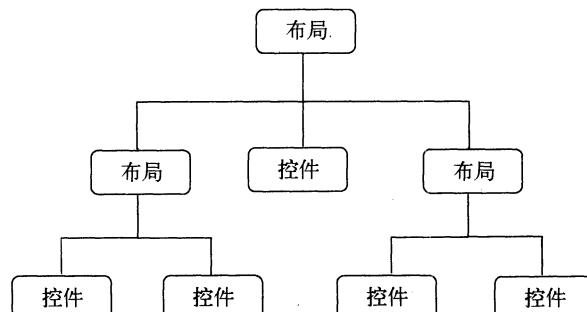


图 3.15 布局和控件的关系

下面我们来详细讲解下 Android 中 4 种最基本的布局。先做好准备工作，新建一个 `UILayoutTest` 项目，并让 Android Studio 自动帮我们创建好活动，活动名和布局名都使用默认值。

3.3.1 线性布局

`LinearLayout` 又称作线性布局，是一种非常常用的布局。正如它的名字所描述的一样，这个布局会将它所包含的控件在线性方向上依次排列。相信你之前也已经注意到了，我们在上一节中学习控件用法时，所有的控件就都是放在 `LinearLayout` 布局里的，因此上一节中的控件也确实在垂直方向上线性排列的。

既然是线性排列，肯定就不仅只有一个方向，那为什么上一节中的控件都是在垂直方向排列

的呢？这是由于我们通过 `android:orientation` 属性指定了排列方向是 `vertical`，如果指定的是 `horizontal`，控件就会在水平方向上排列了。下面我们通过实战来体会一下，修改 `activity_main.xml` 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    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="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3" />

</LinearLayout>
```

我们在 `LinearLayout` 中添加了 3 个 `Button`，每个 `Button` 的长和宽都是 `wrap_content`，并指定了排列方向是 `vertical`。现在运行一下程序，效果如图 3.16 所示。



图 3.16 `LinearLayout` 垂直排列

然后我们修改一下 LinearLayout 的排列方向，如下所示：

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

将 `android:orientation` 属性的值改成了 `horizontal`，这就意味着要让 `LinearLayout` 中的控件在水平方向上依次排列。当然如果不指定 `android:orientation` 属性的值，默认的排列方向就是 `horizontal`。重新运行一下程序，效果如图 3.17 所示。

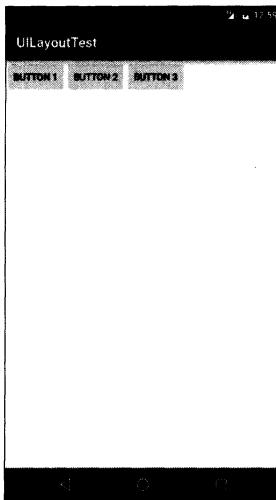


图 3.17 LinearLayout 水平排列

这里需要注意，如果 `LinearLayout` 的排列方向是 `horizontal`，内部的控件就绝对不能将宽度指定为 `match_parent`，因为这样的话，单独一个控件就会将整个水平方向占满，其他的控件就没有可放置的位置了。同样的道理，如果 `LinearLayout` 的排列方向是 `vertical`，内部的控件就不能将高度指定为 `match_parent`。

首先来看 `android:layout_gravity` 属性，它和我们上一节中学到的 `android:gravity` 属性看起来有些相似，这两个属性有什么区别呢？其实从名字就可以看出，`android:gravity` 用于指定文字在控件中的对齐方式，而 `android:layout_gravity` 用于指定控件在布局中的对齐方式。`android:layout_gravity` 的可选值和 `android:gravity` 差不多，但是需要注意，当 `LinearLayout` 的排列方向是 `horizontal` 时，只有垂直方向上的对齐方式才会生效，因为此时水平方向上的长度是不固定的，每添加一个控件，水平方向上的长度都会改变，因而无法指定该方向上的对齐方式。同样的道理，当 `LinearLayout` 的排列方向是 `vertical` 时，只有水平方向上的对齐方

式才会生效。修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    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:layout_gravity="top"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:text="Button 3" />

</LinearLayout>
```

由于目前 LinearLayout 的排列方向是 horizontal，因此我们只能指定垂直方向上的排列方向，将第一个 Button 的对齐方式指定为 top，第二个 Button 的对齐方式指定为 center_vertical，第三个 Button 的对齐方式指定为 bottom。重新运行程序，效果如图 3.18 所示。

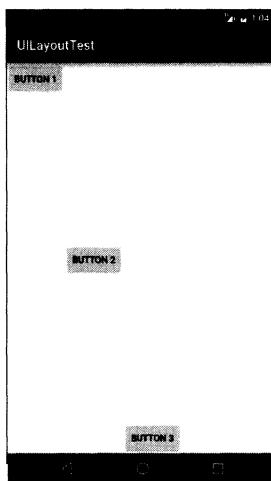


图 3.18 指定 layout_gravity 的效果

接下来我们学习下 LinearLayout 中的另一个重要属性——`android:layout_weight`。这个属性允许我们使用比例的方式来指定控件的大小，它在手机屏幕的适配性方面可以起到非常重要的作用。比如我们正在编写一个消息发送界面，需要一个文本编辑框和一个发送按钮，修改 activity_main.xml 中的代码，如下所示：

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

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
        />

    <Button
        android:id="@+id/send"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Send"
        />

</LinearLayout>
```

你会发现，这里竟然将 EditText 和 Button 的宽度都指定成了 0dp，这样文本编辑框和按钮还能显示出来吗？不用担心，由于我们使用了 `android:layout_weight` 属性，此时控件的宽度就不应该再由 `android:layout_width` 来决定，这里指定成 0dp 是一种比较规范的写法。另外，dp 是 Android 中用于指定控件大小、间距等属性的单位，后面我们还会经常用到它。

然后在 EditText 和 Button 里都将 `android:layout_weight` 属性的值指定为 1，这表示 EditText 和 Button 将在水平方向平分宽度。

为什么将 `android:layout_weight` 属性的值同时指定为 1 就会平分屏幕宽度呢？其实原理也很简单，系统会先把 LinearLayout 下所有控件指定的 `layout_weight` 值相加，得到一个总值，然后每个控件所占大小的比例就是用该控件的 `layout_weight` 值除以刚才算出的总值。因此如果想让 EditText 占据屏幕宽度的 3/5，Button 占据屏幕宽度的 2/5，只需要将 EditText 的 `layout_weight` 改成 3，Button 的 `layout_weight` 改成 2 就可以了。

重新运行程序，你会看到如图 3.19 所示的效果。

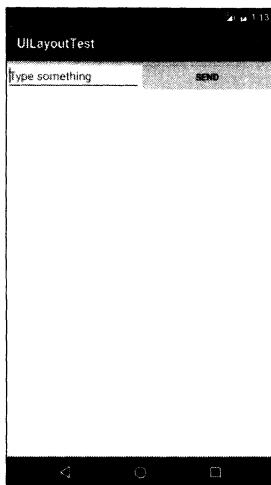


图 3.19 指定 `layout_weight` 的效果

我们还可以通过指定部分控件的 `layout_weight` 值来实现更好的效果。修改 `activity_main.xml` 中的代码，如下所示：

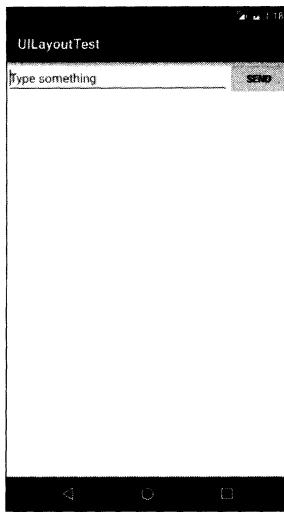
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/input_message"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something"
    />

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

</LinearLayout>
```

这里我们仅指定了 `EditText` 的 `android:layout_weight` 属性，并将 `Button` 的宽度改回 `wrap_content`。这表示 `Button` 的宽度仍然按照 `wrap_content` 来计算，而 `EditText` 则会占满屏幕所有的剩余空间。使用这种方式编写的界面，不仅在各种屏幕的适配方面会非常好，而且看起来也更加舒服。重新运行程序，效果如图 3.20 所示。

图 3.20 使用 `layout_weight` 实现宽度自适配效果

3.3.2 相对布局

RelativeLayout 又称作相对布局，也是一种非常常用的布局。和 LinearLayout 的排列规则不同，RelativeLayout 显得更加随意一些，它可以通过相对定位的方式让控件出现在布局的任何位置。也正因为如此，RelativeLayout 中的属性非常多，不过这些属性都是有规律可循的，其实并不难理解和记忆。我们还是通过实践来体会一下，修改 `activity_main.xml` 中的代码，如下所示：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    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:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="Button 2" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="Button 3" />

<Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:text="Button 4" />

<Button
    android:id="@+id/button5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:text="Button 5" />

</RelativeLayout>
```

我想以上代码已经不需要我再做过多解释了，因为实在是太好理解了。我们让 Button 1 和父布局的左上角对齐，Button 2 和父布局的右上角对齐，Button 3 居中显示，Button 4 和父布局的左下角对齐，Button 5 和父布局的右下角对齐。虽然 `android:layout_alignParentLeft`、`android:layout_alignParentTop`、`android:layout_alignParentRight`、`android:layout_alignParentBottom`、`android:layout_centerInParent` 这几个属性我们之前都没接触过，可是它们的名字已经完全说明了它们的作用。重新运行程序，效果如图 3.21 所示。



图 3.21 相对于父布局定位的效果

上面例子中的每个控件都是相对于父布局进行定位的，那控件可不可以相对于控件进行定位呢？当然是可以的，修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:text="Button 3" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 2" />

    <Button
        android:id="@+id/button4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toLeftOf="@id/button3"
        android:text="Button 4" />

    <Button
        android:id="@+id/button5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/button3"
        android:layout_toRightOf="@id/button3"
        android:text="Button 5" />

</RelativeLayout>
```

这次的代码稍微复杂一点，不过仍然是有规律可循的。`android:layout_above` 属性可以让一个控件位于另一个控件的上方，需要为这个属性指定相对控件 id 的引用，这里我们填入了 `@id/button3`，表示让该控件位于 Button 3 的上方。其他的属性也都是相似的，`android:layout_below` 表示让一个控件位于另一个控件的下方，`android:layout_toLeftOf` 表示让一

一个控件位于另一个控件的左侧，`android:layout_toRightOf` 表示让一个控件位于另一个控件的右侧。注意，当一个控件去引用另一个控件的 id 时，该控件一定要定义在引用控件的后面，不然会出现找不到 id 的情况。重新运行程序，效果如图 3.22 所示。



图 3.22 相对于控件定位的效果

`RelativeLayout` 中还有另外一组相对于控件进行定位的属性，`android:layout_alignLeft` 表示让一个控件的左边缘和另一个控件的左边缘对齐，`android:layout_alignRight` 表示让一个控件的右边缘和另一个控件的右边缘对齐。此外，还有 `android:layout_alignTop` 和 `android:layout_alignBottom`，道理都是一样的，我就不再多说，这几个属性就留给你自己去尝试吧。

好了，正如我前面所说，`RelativeLayout` 中的属性虽然多，但都是有规律可循的，所以学起来一点都不觉得吃力吧？

3.3.3 帧布局

`FrameLayout` 又称作帧布局，它相比于前面两种布局就简单太多了，因此它的应用场景也少了很多。这种布局没有方便的定位方式，所有的控件都会默认摆放在布局的左上角。让我们通过例子来看一看吧，修改 `activity_main.xml` 中的代码，如下所示：

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

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="This is TextView"
  />

<ImageView
  android:id="@+id/image_view"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:src="@mipmap/ic_launcher"
/>
</FrameLayout>
```

FrameLayout 中只是放置了一个 TextView 和一个 ImageView。需要注意的是，当前项目我们没有准备任何图片，所以这里 ImageView 直接使用了@mipmap 来访问 ic_launcher 这张图，虽说这种用法的场景可能非常少，但我还是要告诉你，这是完全可行的。重新运行程序，效果如图 3.23 所示。



图 3.23 FrameLayout 运行效果

可以看到，文字和图片都是位于布局的左上角。由于 ImageView 是在 TextView 之后添加的，因此图片压在了文字的上面。

当然除了这种默认效果之外，我们还可以使用 `layout_gravity` 属性来指定控件在布局中的对齐方式，这和 LinearLayout 中的用法是相似的。修改 `activity_main.xml` 中的代码，如下所示：

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

  <TextView
    android:id="@+id/text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="top|left"
    android:text="This is TextView"/>
  <ImageView
    android:id="@+id/image_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@mipmap/ic_launcher"/>
</FrameLayout>
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="This is TextView"
    />

<ImageView
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="right"
    android:src="@mipmap/ic_launcher"
    />

</FrameLayout>
```

我们指定 TextView 在 FrameLayout 中居左对齐，指定 ImageView 在 FrameLayout 中居右对齐，然后重新运行程序，效果如图 3.24 所示。



图 3.24 指定 layout_gravity 的效果

总体来讲，FrameLayout 由于定位方式的欠缺，导致它的应用场景也比较少，不过在下一章中介绍碎片的时候我们还是可以用到它的。

3.3.4 百分比布局

前面介绍的 3 种布局都是从 Android 1.0 版本中就开始支持了，一直沿用到现在，可以说是满足了绝大多数场景的界面设计需求。不过细心的你会发现，只有 LinearLayout 支持使用 layout_weight 属性来实现按比例指定控件大小的功能，其他两种布局都不支持。比如说，如果想用 RelativeLayout 来实现让两个按钮平分布局宽度的效果，则是比较困难的。

为此，Android 引入了一种全新的布局方式来解决此问题——百分比布局。在这种布局中，我们可以不再使用 `wrap_content`、`match_parent` 等方式来指定控件的大小，而是允许直接指定控件在布局中所占的百分比，这样的话就可以轻松实现平分布局甚至是任意比例分割布局的效果了。

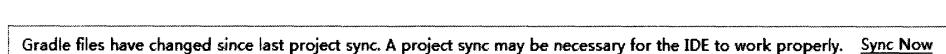
由于 `LinearLayout` 本身已经支持按比例指定控件的大小了，因此百分比布局只为 `FrameLayout` 和 `RelativeLayout` 进行了功能扩展，提供了 `PercentFrameLayout` 和 `PercentRelativeLayout` 这两个全新的布局，下面我们就来具体学习一下。

不同于前 3 种布局，百分比布局属于新增布局，那么怎么才能做到让新增布局在所有 Android 版本上都能使用呢？为此，Android 团队将百分比布局定义在了 `support` 库当中，我们只需要在项目的 `build.gradle` 中添加百分比布局库的依赖，就能保证百分比布局在 Android 所有系统版本上的兼容性了。

打开 `app/build.gradle` 文件，在 `dependencies` 闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:percent:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

需要注意的是，每当修改了任何 `gradle` 文件时，Android Studio 都会弹出一个如图 3.25 所示的提示。



Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly. [Sync Now](#)

图 3.25 `gradle` 文件修改后的提示

这个提示告诉我们，`gradle` 文件自上次同步之后又发生了变化，需要再次同步才能使项目正常工作。这里只需要点击 `Sync Now` 就可以了，然后 `gradle` 会开始进行同步，把我们新添加的百分比布局库引入到项目当中。

接下来修改 `activity_main.xml` 中的代码，如下所示：

```
<android.support.percent.PercentFrameLayout
    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:text="Button 1"
        android:layout_gravity="left|top"
        app:layout_widthPercent="50%"
        app:layout_heightPercent="50%">
```

```

/>

<Button
    android:id="@+id/button2"
    android:text="Button 2"
    android:layout_gravity="right|top"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

<Button
    android:id="@+id/button3"
    android:text="Button 3"
    android:layout_gravity="left|bottom"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

<Button
    android:id="@+id/button4"
    android:text="Button 4"
    android:layout_gravity="right|bottom"
    app:layout_widthPercent="50%"
    app:layout_heightPercent="50%"
/>

</android.support.percent.PercentFrameLayout>

```

最外层我们使用了 PercentFrameLayout，由于百分比布局并不是内置在系统 SDK 当中的，所以需要把完整的包路径写出来。然后还必须定义一个 app 的命名空间，这样才能使用百分比布局的自定义属性。

在 PercentFrameLayout 中我们定义了 4 个按钮，使用 app:layout_widthPercent 属性将各按钮的宽度指定为布局的 50%，使用 app:layout_heightPercent 属性将各按钮的高度指定为布局的 50%。这里之所以能使用 app 前缀的属性就是因为刚才定义了 app 的命名空间，当然我们一直能使用 android 前缀的属性也是同样的道理。

不过 PercentFrameLayout 还是会继承 FrameLayout 的特性，即所有的控件默认都是摆放在布局的左上角。那么为了让这 4 个按钮不会重叠，这里还是借助了 layout_gravity 来分别将这 4 个按钮放置在布局的左上、右上、左下、右下 4 个位置。

现在我们已经可以重新运行程序了，不过如果你使用的是老版本的 Android Studio，可能会在 activity_main.xml 中看到一些如图 3.26 所示的错误提示。

'layout_height' attribute should be defined more... (Ctrl+F1)
 'layout_width' attribute should be defined more... (Ctrl+F1)

图 3.26 activity_main.xml 中错误提示

这是因为老版本的Android Studio中内置了布局的检查机制，认为每一个控件都应该通过`android:layout_width`和`android:layout_height`属性指定宽高才是合法的。而其实我们是通过`app:layout_widthPercent`和`app:layout_heightPercent`属性来指定宽高的，所以Android Studio没检测到。不过这个错误提示并不影响程序运行，我们直接忽视就可以了。当然最新的Android Studio 2.2版本中已经修复了这个问题，因此你可能并不会看到上述的错误提示。

现在重新运行程序，效果如图3.27所示。



图3.27 PercentFrameLayout运行效果

可以看到，每一个按钮的宽和高都占据了布局的50%，这样我们就轻松实现了4个按钮平分屏幕的效果。

PercentFrameLayout的用法就介绍到这里，另外一个PercentRelativeLayout的用法也是非常相似的，它继承了RelativeLayout中的所有属性，并且可以使用`app:layout_widthPercent`和`app:layout_heightPercent`来按百分比指定控件的宽高，相信聪明的你一定可以举一反三了。

这样我们就把最常用的几种布局都讲解完了，其实Android中还有AbsoluteLayout、TableLayout等布局，不过由于使用得实在是太少了，就不在本书中进行讲解了。

3.4 系统控件不够用？创建自定义控件

在前面两节我们已经学习了Android中的一些常用控件以及基本布局的用法，不过当时我们并没有关注这些控件和布局的继承结构，现在是时候来看一下了，如图3.28所示。



图 3.28 常用控件和布局的继承结构

可以看到，我们所用的所有控件都是直接或间接继承自 View 的，所用的所有布局都是直接或间接继承自 ViewGroup 的。View 是 Android 中最基本的一种 UI 组件，它可以在屏幕上绘制一块矩形区域，并能响应这块区域的各种事件，因此，我们使用的各种控件其实就是在 View 的基础之上又添加了各自特有的功能。而 ViewGroup 则是一种特殊的 View，它可以包含很多子 View 和子 ViewGroup，是一个用于放置控件和布局的容器。

这个时候我们就可以思考一下，当系统自带的控件并不能满足我们的需求时，可不可以利用上面的继承结构来创建自定义控件呢？答案是肯定的，下面我们就来学习一下创建自定义控件的两种简单方法。先将准备工作做好，创建一个 UICustomViews 项目。

3.4.1 引入布局

如果你用过 iPhone 应该会知道，几乎每一个 iPhone 应用的界面顶部都会有一个标题栏，标题栏上会有一到两个按钮可用于返回或其他操作（iPhone 没有实体返回键）。现在很多 Android 程序也都喜欢模仿 iPhone 的风格，在界面的顶部放置一个标题栏。虽然 Android 系统已经给每个活动提供了标题栏功能，但这里我们决定先不使用它，而是创建一个自定义的标题栏。

经过前面两节的学习，相信创建一个标题栏布局对你来说已经不是什么困难的事情了，只需要加入两个 Button 和一个 TextView，然后在布局中摆放好就可以了。可是这样做却存在着一个问题，一般我们的程序中可能有很多个活动都需要这样的标题栏，如果在每个活动的布局中都编写一遍同样的标题栏代码，明显就会导致代码的大量重复。这个时候我们就可以使用引入布局的方式来解决这个问题，新建一个布局 title.xml，代码如下所示：

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/title_bg">

    <Button
        android:id="@+id/title_back"
        android:layout_width="wrap_content"
  
```

```

    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="5dp"
    android:background="@drawable/back_bg"
    android:text="Back"
    android:textColor="#fff" />

    <TextView
        android:id="@+id/title_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="center"
        android:text="Title Text"
        android:textColor="#fff"
        android:textSize="24sp" />

    <Button
        android:id="@+id/title_edit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="5dp"
        android:background="@drawable/edit_bg"
        android:text="Edit"
        android:textColor="#fff" />

</LinearLayout>

```

可以看到，我们在 `LinearLayout` 中分别加入了两个 `Button` 和一个 `TextView`，左边的 `Button` 可用于返回，右边的 `Button` 可用于编辑，中间的 `TextView` 则可以显示一段标题文本。上面代码中的大多数属性都是你已经见过的，下面我来说明一下几个之前没有讲过的属性。`android:background` 用于为布局或控件指定一个背景，可以使用颜色或图片来进行填充，这里我提前准备好了 3 张图片——`title_bg.png`、`back_bg.png` 和 `edit_bg.png`，分别用于作为标题栏、返回按钮和编辑按钮的背景。另外，在两个 `Button` 中我们都使用了 `android:layout_margin` 这个属性，它可以指定控件在上下左右方向上偏移的距离，当然也可以使用 `android:layout_marginLeft` 或 `android:layout_marginTop` 等属性来单独指定控件在某个方向上偏移的距离。

现在标题栏布局已经编写完成了，剩下的就是如何在程序中使用这个标题栏了，修改 `activity_main.xml` 中的代码，如下所示：

```

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

    <include layout="@layout/title" />

</LinearLayout>

```

没错！我们只需要通过一行 `include` 语句将标题栏布局引入进来就可以了。

最后别忘了在 MainActivity 中将系统自带的标题栏隐藏掉，代码如下所示：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ActionBar actionBar = getSupportActionBar();
        if (actionBar != null) {
            actionBar.hide();
        }
    }
}
```

这里我们调用了 `getSupportActionBar()` 方法来获得 `ActionBar` 的实例，然后再调用 `ActionBar` 的 `hide()` 方法将标题栏隐藏起来。关于 `ActionBar` 的更多用法我们将会在第 12 章中讲解，现在你只需要知道可以通过这种写法来隐藏标题栏就足够了。现在运行一下程序，效果如图 3.29 所示。

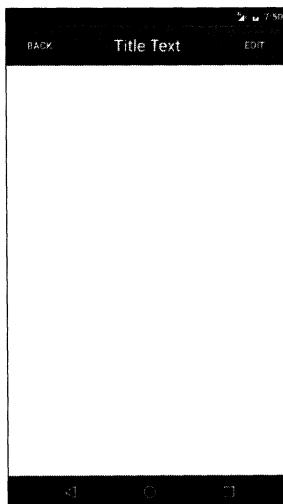


图 3.29 引入标题栏布局的效果

使用这种方式，不管有多少布局需要添加标题栏，只需一行 `include` 语句就可以了。

3.4.2 创建自定义控件

引入布局的技巧确实解决了重复编写布局代码的问题，但是如果布局中有一些控件要求能够响应事件，我们还是需要在每个活动中为这些控件单独编写一次事件注册的代码。比如说标题栏中的返回按钮，其实不管是在哪一个活动中，这个按钮的功能都是相同的，即销毁当前活动。而

如果在每一个活动中都需要重新注册一遍返回按钮的点击事件，无疑会增加很多重复代码，这种情况最好是使用自定义控件的方式来解决。

新建 TitleLayout 继承自 LinearLayout，让它成为我们自定义的标题栏控件，代码如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
    }

}
```

首先我们重写了 LinearLayout 中带有两个参数的构造函数，在布局中引入 TitleLayout 控件就会调用这个构造函数。然后在构造函数中需要对标题栏布局进行动态加载，这就要借助 LayoutInflater 来实现了。通过 LayoutInflater 的 from() 方法可以构建出一个 LayoutInflater 对象，然后调用 inflate() 方法就可以动态加载一个布局文件，inflate() 方法接收两个参数，第一个参数是要加载的布局文件的 id，这里我们传入 R.layout.title，第二个参数是给加载好的布局再添加一个父布局，这里我们想要指定为 TitleLayout，于是直接传入 this。

现在自定义控件已经创建好了，然后我们需要在布局文件中添加这个自定义控件，修改 activity_main.xml 中的代码，如下所示：

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

    <com.example.uicustomviews.TitleLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

添加自定义控件和添加普通控件的方式基本是一样的，只不过在添加自定义控件的时候，我们需要指明控件的完整类名，包名在这里是不可以省略的。

重新运行程序，你会发现此时效果和使用引入布局方式的效果是一样的。

下面我们尝试为标题栏中的按钮注册点击事件，修改 TitleLayout 中的代码，如下所示：

```
public class TitleLayout extends LinearLayout {

    public TitleLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
        LayoutInflater.from(context).inflate(R.layout.title, this);
        Button titleBack = (Button) findViewById(R.id.title_back);
        Button titleEdit = (Button) findViewById(R.id.title_edit);
        titleBack.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
        ((Activity) getContext()).finish();
    }
});
titleEdit.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(getContext(), "You clicked Edit button",
                Toast.LENGTH_SHORT).show();
    }
});
}
}
```

首先还是通过 `findViewById()` 方法得到按钮的实例，然后分别调用 `setOnClickListener()` 方法给两个按钮注册了点击事件，当点击返回按钮时销毁掉当前的活动，当点击编辑按钮时弹出一段文本。重新运行程序，点击一下编辑按钮，效果如图 3.30 所示。



图 3.30 点击编辑按钮的效果

这样的话，每当我们在一个布局中引入 `TitleLayout` 时，返回按钮和编辑按钮的点击事件就已经自动实现好了，这就省去了很多编写重复代码的工作。

3.5 最常用和最难用的控件——ListView

`ListView` 绝对可以称得上是 Android 中最常用的控件之一，几乎所有的应用程序都会用到它。由于手机屏幕空间都比较有限，能够一次性在屏幕上显示的内容并不多，当我们的程序中有大量的数据需要展示的时候，就可以借助 `ListView` 来实现。`ListView` 允许用户通过手指上下滑动的方式将屏幕外的数据滚动到屏幕内，同时屏幕上原有的数据则会滚动出屏幕。相信你其实每天都在使用这个控件，比如查看 QQ 聊天记录，翻阅微博最新消息，等等。

不过比起前面介绍的几种控件，ListView 的用法也相对复杂了很多，因此我们就单独使用一节内容来对 ListView 进行非常详细的讲解。

3.5.1 ListView 的简单用法

首先新建一个 ListViewTest 项目，并让 Android Studio 自动帮我们创建好活动。然后修改 activity_main.xml 中的代码，如下所示：

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

    <ListView
        android:id="@+id/list_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

在布局中加入 ListView 控件还算非常简单，先为 ListView 指定一个 id，然后将宽度和高度都设置为 `match_parent`，这样 ListView 也就占满了整个布局的空间。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private String[] data = { "Apple", "Banana", "Orange", "Watermelon",
        "Pear", "Grape", "Pineapple", "Strawberry", "Cherry", "Mango",
        "Apple", "Banana", "Orange", "Watermelon", "Pear", "Grape",
        "Pineapple", "Strawberry", "Cherry", "Mango" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            MainActivity.this, android.R.layout.simple_list_item_1, data);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
    }
}
```

既然 ListView 是用于展示大量数据的，那我们就应该先将数据提供好。这些数据可以是从网上下载的，也可以是从数据库中读取的，应该视具体的应用程序场景而定。这里我们就简单使用了一个 `data` 数组来测试，里面包含了很多水果的名称。

不过，数组中的数据是无法直接传递给 ListView 的，我们还需要借助适配器来完成。Android 中提供了很多适配器的实现类，其中我认为最好用的就是 `ArrayAdapter`。它可以通过泛型来指定要适配的数据类型，然后在构造函数中把要适配的数据传入。`ArrayAdapter` 有多个构造函数的重

载，你应该根据实际情况选择最合适的一种。这里由于我们提供的数据都是字符串，因此将 ArrayAdapter 的泛型指定为 `String`，然后在 ArrayAdapter 的构造函数中依次传入当前上下文、ListView 子项布局的 id，以及要适配的数据。注意，我们使用了 `android.R.layout.simple_list_item_1` 作为 ListView 子项布局的 id，这是一个 Android 内置的布局文件，里面只有一个 `TextView`，可用于简单地显示一段文本。这样适配器对象就构建好了。

最后，还需要调用 ListView 的 `setAdapter()` 方法，将构建好的适配器对象传递进去，这样 ListView 和数据之间的关联就建立完成了。

现在运行一下程序，效果如图 3.31 所示。可以通过滚动的方式来查看屏幕外的数据。



图 3.31 ListView 运行效果

3.5.2 定制 ListView 的界面

只能显示一段文本的 ListView 实在是太单调了，我们现在就来对 ListView 的界面进行定制，让它可以显示更加丰富的内容。

首先需要准备好一组图片，分别对应上面提供的每一种水果，待会我们要让这些水果名称的旁边都有一个图样。

接着定义一个实体类，作为 ListView 适配器的适配类型。新建类 `Fruit`，代码如下所示：

```
public class Fruit {  
    private String name;  
    private int imageId;
```

```

public Fruit(String name, int imageId) {
    this.name = name;
    this.imageId = imageId;
}

public String getName() {
    return name;
}

public int getImageId() {
    return imageId;
}

}

```

Fruit 类中只有两个字段，name 表示水果的名字，imageId 表示水果对应图片的资源 id。

然后需要为 ListView 的子项指定一个我们自定义的布局，在 layout 目录下新建 fruit_item.xml，代码如下所示：

```

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

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginLeft="10dp" />

</LinearLayout>

```

在这个布局中，我们定义了一个 ImageView 用于显示水果的图片，又定义了一个 TextView 用于显示水果的名称，并让 TextView 在垂直方向上居中显示。

接下来需要创建一个自定义的适配器，这个适配器继承自 ArrayAdapter，并将泛型指定为 Fruit 类。新建类 FruitAdapter，代码如下所示：

```

public class FruitAdapter extends ArrayAdapter<Fruit> {

    private int resourceId;

    public FruitAdapter(Context context, int textViewResourceId,
                       List<Fruit> objects) {
        super(context, textViewResourceId, objects);
        resourceId = textViewResourceId;
    }
}

```

```

@Override
public View getView(int position, View convertView, ViewGroup parent) {
    Fruit fruit = getItem(position); // 获取当前项的 Fruit 实例
    View view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
        false);
    ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
    TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
    fruitImage.setImageResource(fruit.getImageId());
    fruitName.setText(fruit.getName());
    return view;
}

```

`FruitAdapter` 重写了父类的一组构造函数，用于将上下文、`ListView`子项布局的 id 和数据都传递进来。另外又重写了 `getView()`方法，这个方法在每个子项被滚动到屏幕内的时候会被调用。在 `getView()`方法中，首先通过 `getItem()`方法得到当前项的 `Fruit` 实例，然后使用 `LayoutInflater` 来为这个子项加载我们传入的布局。

这里 `LayoutInflater` 的 `inflate()`方法接收 3 个参数，前两个参数我们已经知道是什么意思了，第三个参数指定成 `false`，表示只让我们在父布局中声明的 `layout` 属性生效，但不为这个 `View` 添加父布局，因为一旦 `View` 有了父布局之后，它就不能再添加到 `ListView` 中了。如果你现在还不能理解这段话的含义也没关系，只需要知道这是 `ListView` 中的标准写法就可以了，当你以后对 `View` 理解得更加深刻的时候，再来读这段话就没有问题了。

我们继续往下看，接下来调用 `View` 的 `findViewById()`方法分别获取到 `ImageView` 和 `TextView` 的实例，并分别调用它们的 `setImageResource()`和 `setText()`方法来设置显示的图片和文字，最后将布局返回，这样我们自定义的适配器就完成了。

下面修改 `MainActivity` 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        FruitAdapter adapter = new FruitAdapter(MainActivity.this,
            R.layout.fruit_item, fruitList);
        ListView listView = (ListView) findViewById(R.id.list_view);
        listView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit("Apple", R.drawable.apple_pic);
            fruitList.add(apple);
        }
    }
}

```

```
        Fruit banana = new Fruit("Banana", R.drawable.banana_pic);
        fruitList.add(banana);
        Fruit orange = new Fruit("Orange", R.drawable.orange_pic);
        fruitList.add(orange);
        Fruit watermelon = new Fruit("Watermelon", R.drawable.watermelon_pic);
        fruitList.add(watermelon);
        Fruit pear = new Fruit("Pear", R.drawable.pear_pic);
        fruitList.add(pear);
        Fruit grape = new Fruit("Grape", R.drawable.grape_pic);
        fruitList.add(grape);
        Fruit pineapple = new Fruit("Pineapple", R.drawable.pineapple_pic);
        fruitList.add(pineapple);
        Fruit strawberry = new Fruit("Strawberry", R.drawable.strawberry_pic);
        fruitList.add(strawberry);
        Fruit cherry = new Fruit("Cherry", R.drawable.cherry_pic);
        fruitList.add(cherry);
        Fruit mango = new Fruit("Mango", R.drawable.mango_pic);
        fruitList.add(mango);
    }
}
```

可以看到，这里添加了一个 `initFruits()`方法，用于初始化所有的水果数据。在 `Fruit` 类的构造函数中将水果的名字和对应的图片 `id` 传入，然后把创建好的对象添加到水果列表中。另外我们使用了一个 `for` 循环将所有的水果数据添加了两遍，这是因为如果只添加一遍的话，数据量还不足以充满整个屏幕。接着在 `onCreate()` 方法中创建了 `FruitAdapter` 对象，并将 `FruitAdapter` 作为适配器传递给 `ListView`，这样定制 `ListView` 界面的任务就完成了。

现在重新运行程序，效果如图 3.32 所示。

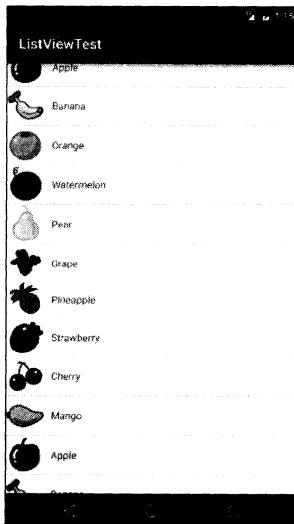


图 3.32 定制界面的 ListView 运行效果

虽然目前我们定制的界面还很简单，但是相信聪明的你已经领悟到了诀窍，只要修改 fruit_item.xml 中的内容，就可以定制出各种复杂的界面了。

3.5.3 提升 ListView 的运行效率

之所以说 ListView 这个控件很难用，就是因为它有很多细节可以优化，其中运行效率就是很重要的一点。目前我们 ListView 的运行效率是很低的，因为在 FruitAdapter 的 getView() 方法中，每次都将布局重新加载了一遍，当 ListView 快速滚动的时候，这就会成为性能的瓶颈。

仔细观察会发现，getView()方法中还有一个 convertView 参数，这个参数用于将之前加载好的布局进行缓存，以便之后可以进行重用。修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    ...

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Fruit fruit = getItem(position);
        View view;
        if (convertView == null) {
            view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
                false);
        } else {
            view = convertView;
        }
        ImageView fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        TextView fruitName = (TextView) view.findViewById(R.id.fruit_name);
        fruitImage.setImageResource(fruit.getImageId());
        fruitName.setText(fruit.getName());
        return view;
    }
}
```

可以看到，现在我们在 getView()方法中进行了判断，如果 convertView 为 null，则使用 LayoutInflater 去加载布局，如果不为 null 则直接对 convertView 进行重用。这样就大大提高了 ListView 的运行效率，在快速滚动的时候也可以表现出更好的性能。

不过，目前我们的这份代码还是可以继续优化的，虽然现在已经不会再重复去加载布局，但是每次在 getView()方法中还是会调用 View 的 findViewById()方法来获取一次控件的实例。我们可以借助一个 ViewHolder 来对这部分性能进行优化，修改 FruitAdapter 中的代码，如下所示：

```
public class FruitAdapter extends ArrayAdapter<Fruit> {

    ...

    @Override
```

```

public View getView(int position, View convertView, ViewGroup parent) {
    Fruit fruit = getItem(position);
    View view;
    ViewHolder viewHolder;
    if (convertView == null) {
        view = LayoutInflater.from(getContext()).inflate(resourceId, parent,
            false);
        viewHolder = new ViewHolder();
        viewHolder.fruitImage = (ImageView) view.findViewById (R.id.fruit_image);
        viewHolder.fruitName = (TextView) view.findViewById (R.id.fruit_name);
        view.setTag(viewHolder); // 将 ViewHolder 存储在 View 中
    } else {
        view = convertView;
        viewHolder = (ViewHolder) view.getTag(); // 重新获取 ViewHolder
    }
    viewHolder.fruitImage.setImageResource(fruit.getImageId());
    viewHolder.fruitName.setText(fruit.getName());
    return view;
}

class ViewHolder {
    ImageView fruitImage;

    TextView fruitName;
}
}

```

我们新增了一个内部类 ViewHolder，用于对控件的实例进行缓存。当 convertView 为 null 的时候，创建一个 ViewHolder 对象，并将控件的实例都存放在 ViewHolder 里，然后调用 View 的 setTag()方法，将 ViewHolder 对象存储在 View 中。当 convertView 不为 null 的时候，则调用 View 的 getTag()方法，把 ViewHolder 重新取出。这样所有控件的实例都缓存在了 ViewHolder 里，就没有必要每次都通过 findViewById()方法来获取控件实例了。

通过这两步优化之后，我们 ListView 的运行效率就已经非常不错了。

3.5.4 ListView 的点击事件

话说回来，ListView 的滚动毕竟只是满足了我们视觉上的效果，可是如果 ListView 中的子项不能点击的话，这个控件就没有什么实际的用途了。因此，本小节我们就来学习一下 ListView 如何才能响应用户的点击事件。

修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

```

```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    initFruits();  
    FruitAdapter adapter = new FruitAdapter(MainActivity.this, R.layout.  
        fruit_item, fruitList);  
    ListView listView = (ListView) findViewById(R.id.list_view);  
    listView.setAdapter(adapter);  
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
        @Override  
        public void onItemClick(AdapterView<?> parent, View view,  
            int position, long id) {  
            Fruit fruit = fruitList.get(position);  
            Toast.makeText(MainActivity.this, fruit.getName(),  
                Toast.LENGTH_SHORT).show();  
        }  
    });  
}  
  
...  
}
```

可以看到，我们使用 `setOnItemClickListener()` 方法为 `ListView` 注册了一个监听器，当用户点击了 `ListView` 中的任何一个子项时，就会回调 `onItemClick()` 方法。在这个方法中可以通过 `position` 参数判断出用户点击的是哪一个子项，然后获取到相应的水果，并通过 `Toast` 将水果的名字显示出来。

重新运行程序，并点击一下橘子，效果如图 3.33 所示。



图 3.33 点击 ListView 的效果

3.6 更强大的滚动控件——RecyclerView

ListView由于其强大的功能，在过去的Android开发当中可以说是贡献卓越，直到今天仍然还有不计其数的程序在继续使用着ListView。不过ListView并不是完全没有缺点的，比如说如果我们不使用一些技巧来提升它的运行效率，那么ListView的性能就会非常差。还有，ListView的扩展性也不够好，它只能实现数据纵向滚动的效果，如果我们想实现横向滚动的话，ListView是做不到的。

为此，Android提供了一个更强大的滚动控件——RecyclerView。它可以说是一个增强版的ListView，不仅可以轻松实现和ListView同样的效果，还优化了ListView中存在的各种不足之处。目前Android官方更加推荐使用RecyclerView，未来也会有更多的程序逐渐从ListView转向RecyclerView，那么本节我们就来详细讲解一下RecyclerView的用法。

首先新建一个RecyclerViewTest项目，并让Android Studio自动帮我们创建好活动。

3.6.1 RecyclerView的基本用法

和百分比布局类似，RecyclerView也属于新增的控件，为了让RecyclerView在所有Android版本上都能使用，Android团队采取了同样的方式，将RecyclerView定义在了support库当中。因此，想要使用RecyclerView这个控件，首先需要在项目的build.gradle中添加相应的依赖库才行。

打开app/build.gradle文件，在dependencies闭包中添加如下内容：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

添加完之后记得要点击一下Sync Now来进行同步。然后修改activity_main.xml中的代码，如下所示：

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

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

在布局中加入RecyclerView控件也是非常简单的，先为RecyclerView指定一个id，然后将宽度和高度都设置为match_parent，这样RecyclerView也就占满了整个布局的空间。需要注意的是，由于RecyclerView并不是内置在系统SDK当中的，所以需要把完整的包路径写出来。

这里我们想要使用 RecyclerView 来实现和 ListView 相同的效果，因此就需要准备一份同样的水果图片。简单起见，我们就直接从 ListViewTest 项目中把图片复制过来就可以了，另外顺便将 Fruit 类和 fruit_item.xml 也复制过来，省得将同样的代码再写一遍。

接下来需要为 RecyclerView 准备一个适配器，新建 FruitAdapter 类，让这个适配器继承自 RecyclerView.Adapter，并将泛型指定为 FruitAdapter.ViewHolder。其中，ViewHolder 是我们在 FruitAdapter 中定义的一个内部类，代码如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {

    private List<Fruit> mFruitList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        ImageView fruitImage;
        TextView fruitName;

        public ViewHolder(View view) {
            super(view);
            fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
            fruitName = (TextView) view.findViewById(R.id.fruit_name);
        }
    }

    public FruitAdapter(List<Fruit> fruitList) {
        mFruitList = fruitList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.fruit_item, parent, false);
        ViewHolder holder = new ViewHolder(view);
        return holder;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Fruit fruit = mFruitList.get(position);
        holder.fruitImage.setImageResource(fruit.getImageId());
        holder.fruitName.setText(fruit.getName());
    }

    @Override
    public int getItemCount() {
        return mFruitList.size();
    }
}
```

虽然这段代码看上去好像有点长，但其实它比 ListView 的适配器要更容易理解。这里我们首先定义了一个内部类 ViewHolder，ViewHolder 要继承自 RecyclerView.ViewHolder。然后 ViewHolder 的构造函数中要传入一个 View 参数，这个参数通常就是 RecyclerView 子项的最外

层布局，那么我们就可以通过 `findViewById()` 方法来获取到布局中的 `ImageView` 和 `TextView` 的实例了。

接着往下看，`FruitAdapter` 中也有一个构造函数，这个方法用于把要展示的数据源传进来，并赋值给一个全局变量 `mFruitList`，我们后续的操作都将在这个数据源的基础上进行。

继续往下看，由于 `FruitAdapter` 是继承自 `RecyclerView.Adapter` 的，那么就必须重写 `onCreateViewHolder()`、`onBindViewHolder()` 和 `getItemCount()` 这 3 个方法。`onCreateViewHolder()` 方法是用于创建 `ViewHolder` 实例的，我们在这个方法中将 `fruit_item` 布局加载进来，然后创建一个 `ViewHolder` 实例，并把加载出来的布局传入到构造函数当中，最后将 `ViewHolder` 的实例返回。`onBindViewHolder()` 方法是用于对 `RecyclerView` 子项的数据进行赋值的，会在每个子项被滚动到屏幕内的时候执行，这里我们通过 `position` 参数得到当前项的 `Fruit` 实例，然后再将数据设置到 `ViewHolder` 的 `ImageView` 和 `TextView` 当中即可。`getCount()` 方法就非常简单了，它用于告诉 `RecyclerView` 一共有多少子项，直接返回数据源的长度就可以了。

适配器准备好了之后，我们就可以开始使用 `RecyclerView` 了，修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits(); // 初始化水果数据
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit("Apple", R.drawable.apple_pic);
            fruitList.add(apple);
            Fruit banana = new Fruit("Banana", R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit("Orange", R.drawable.orange_pic);
            fruitList.add(orange);
            Fruit watermelon = new Fruit("Watermelon", R.drawable.watermelon_pic);
            fruitList.add(watermelon);
            Fruit pear = new Fruit("Pear", R.drawable.pear_pic);
            fruitList.add(pear);
            Fruit grape = new Fruit("Grape", R.drawable.grape_pic);
            fruitList.add(grape);
            Fruit pineapple = new Fruit("Pineapple", R.drawable.pineapple_pic);
            fruitList.add(pineapple);
        }
    }
}
```

```
        Fruit strawberry = new Fruit("Strawberry", R.drawable.strawberry_pic);
        fruitList.add(strawberry);
        Fruit cherry = new Fruit("Cherry", R.drawable.cherry_pic);
        fruitList.add(cherry);
        Fruit mango = new Fruit("Mango", R.drawable.mango_pic);
        fruitList.add(mango);
    }
}
```

可以看到，这里使用了一个同样的 `initFruits()`方法，用于初始化所有的水果数据。接着在 `onCreate()`方法中我们先获取到 `RecyclerView` 的实例，然后创建了一个 `LinearLayoutManager` 对象，并将它设置到 `RecyclerView` 当中。`LayoutManager` 用于指定 `RecyclerView` 的布局方式，这里使用的 `LinearLayoutManager` 是线性布局的意思，可以实现和 `ListView` 类似的效果。接下来我们创建了 `FruitAdapter` 的实例，并将水果数据传入到 `FruitAdapter` 的构造函数中，最后调用 `RecyclerView` 的 `setAdapter()` 方法来完成适配器设置，这样 `RecyclerView` 和数据之间的关联就建立完成了。

现在可以运行一下程序了，效果如图 3.34 所示。

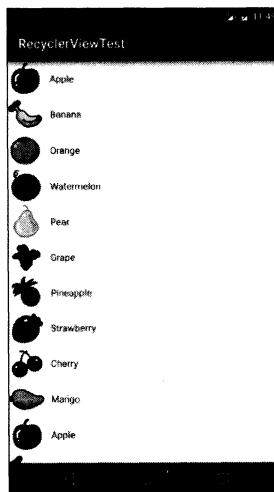


图 3.34 RecyclerView 运行效果

可以看到，我们使用 RecyclerView 实现了和 ListView 几乎一模一样的效果，虽说在代码量方面并没有明显地减少，但是逻辑变得更加清晰了。当然这只是 RecyclerView 的基本用法而已，接下来我们就看一看 RecyclerView 还能实现哪些 ListView 实现不了的效果。

3.6.2 实现横向滚动和瀑布流布局

我们已经知道，ListView 的扩展性并不好，它只能实现纵向滚动的效果，如果想进行横向滚动

动的话，ListView 就做不到了。那么 RecyclerView 就能做得到吗？当然可以，不仅能做到，还非常简单，那么接下来我们就尝试实现一下横向滚动的效果。

首先要对 fruit_item 布局进行修改，因为目前这个布局里面的元素是水平排列的，适用于纵向滚动的场景，而如果我们要实现横向滚动的话，应该把 fruit_item 里的元素改成垂直排列才比较合理。修改 fruit_item.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="100dp"
    android:layout_height="wrap_content" >

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:layout_marginTop="10dp" />

</LinearLayout>
```

可以看到，我们将 LinearLayout 改成垂直方向排列，并把宽度设为 100dp。这里将宽度指定为固定值是因为每种水果的文字长度不一致，如果用 wrap_content 的话，RecyclerView 的子项就会有长有短，非常不美观；而如果用 match_parent 的话，就会导致宽度过长，一个子项占满整个屏幕。

然后我们将 ImageView 和 TextView 都设置成了在布局中水平居中，并且使用 layout_marginTop 属性让文字和图片之间保持一些距离。

接下来修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        layoutManager.setOrientation(LinearLayoutManager.HORIZONTAL);
        recyclerView.setLayoutManager(layoutManager);
```

```

        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }
    ...
}

```

MainActivity 中只加入了一行代码，调用 LinearLayoutManager 的 setOrientation()方法来设置布局的排列方向，默认是纵向排列的，我们传入 LinearLayoutManager.HORIZONTAL 表示让布局横向排列，这样 RecyclerView 就可以横向滚动了。

重新运行一下程序，效果如图 3.35 所示。



图 3.35 横向 RecyclerView 效果

你可以用手指在水平方向上滑动来查看屏幕外的数据。

为什么 ListView 很难或者根本无法实现的效果在 RecyclerView 上这么轻松就能实现了呢？这主要得益于 RecyclerView 出色的设计。ListView 的布局排列是由自身去管理的，而 RecyclerView 则将这个工作交给了 LayoutManager，LayoutManager 中制定了一套可扩展的布局排列接口，子类只要按照接口的规范来实现，就能定制出各种不同排列方式的布局了。

除了 LinearLayoutManager 之外，RecyclerView 还给我们提供了 GridLayoutManager 和 StaggeredGridLayoutManager 这两种内置的布局排列方式。GridLayoutManager 可以用于实现网格布局，StaggeredGridLayoutManager 可以用于实现瀑布流布局。这里我们来实现一下效果更加炫酷的瀑布流布局，网格布局就作为课后习题，交给你自己来研究了。

首先还是来修改一下 fruit_item.xml 中的代码，如下所示：

```

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

```

```

    android:layout_height="wrap_content"
    android:layout_margin="5dp" >

    <ImageView
        android:id="@+id/fruit_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:id="@+id/fruit_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left"
        android:layout_marginTop="10dp" />

</LinearLayout>

```

这里做了几处小的调整，首先将 LinearLayout 的宽度由 100dp 改成了 match_parent，因为瀑布流布局的宽度应该是根据布局的列数来自动适配的，而不是一个固定值。另外我们使用了 layout_margin 属性来让子项之间互留一点间距，这样就不至于所有子项都紧贴在一起。还有就是将 TextView 的对齐属性改成了居左对齐，因为待会我们会将文字的长度变长，如果还是居中显示就会感觉怪怪的。

接着修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity {

    private List<Fruit> fruitList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initFruits();
        RecyclerView recyclerView = (RecyclerView) findViewById(R.id.recycler_view);
        StaggeredGridLayoutManager layoutManager = new
        StaggeredGridLayoutManager(3, StaggeredGridLayoutManager.VERTICAL);
        recyclerView.setLayoutManager(layoutManager);
        FruitAdapter adapter = new FruitAdapter(fruitList);
        recyclerView.setAdapter(adapter);
    }

    private void initFruits() {
        for (int i = 0; i < 2; i++) {
            Fruit apple = new Fruit(
                getRandomLengthName("Apple"), R.drawable.apple_pic);
            fruitList.add(apple);
            Fruit banana = new Fruit(
                getRandomLengthName("Banana"), R.drawable.banana_pic);
            fruitList.add(banana);
            Fruit orange = new Fruit(

```

```

        getRandomLengthName("Orange"), R.drawable.orange_pic);
fruitList.add(orange);
Fruit watermelon = new Fruit(
        getRandomLengthName("Watermelon"), R.drawable.watermelon_pic);
fruitList.add(watermelon);
Fruit pear = new Fruit(
        getRandomLengthName("Pear"), R.drawable.pear_pic);
fruitList.add(pear);
Fruit grape = new Fruit(
        getRandomLengthName("Grape"), R.drawable.grape_pic);
fruitList.add(grape);
Fruit pineapple = new Fruit(
        getRandomLengthName("Pineapple"), R.drawable.pineapple_pic);
fruitList.add(pineapple);
Fruit strawberry = new Fruit(
        getRandomLengthName("Strawberry"), R.drawable.strawberry_pic);
fruitList.add(strawberry);
Fruit cherry = new Fruit(
        getRandomLengthName("Cherry"), R.drawable.cherry_pic);
fruitList.add(cherry);
Fruit mango = new Fruit(
        getRandomLengthName("Mango"), R.drawable.mango_pic);
fruitList.add(mango);
}
}

private String getRandomLengthName(String name) {
    Random random = new Random();
    int length = random.nextInt(20) + 1;
    StringBuilder builder = new StringBuilder();
    for (int i = 0; i < length; i++) {
        builder.append(name);
    }
    return builder.toString();
}
}

```

首先，在`onCreate()`方法中，我们创建了一个`StaggeredGridLayoutManager`的实例。`StaggeredGridLayoutManager`的构造函数接收两个参数，第一个参数用于指定布局的列数，传入3表示会把布局分为3列；第二个参数用于指定布局的排列方向，传入`StaggeredLayoutManager.VERTICAL`表示会让布局纵向排列，最后再把创建好的实例设置到`RecyclerView`当中就可以了，就是这么简单！

没错，仅仅修改了一行代码，我们就已经成功实现瀑布流布局的效果了。不过由于瀑布流布局需要各个子项的高度不一致才能看出明显的效果，为此我又使用了一个小技巧。这里我们把目光聚焦在`getRandomLengthName()`这个方法上，这个方法使用了`Random`对象来创造一个1到20之间的随机数，然后将参数中传入的字符串重复随机遍。在`initFruits()`方法中，每个水果的名字都改成调用`getRandomLengthName()`这个方法来生成，这样就能保证各水果名字的长短

差距都比较大，子项的高度也就各不相同了。

现在重新运行一下程序，效果如图 3.36 所示。

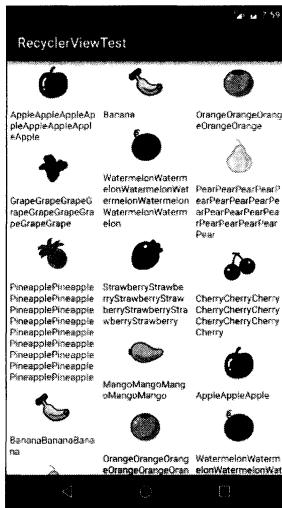


图 3.36 瀑布流布局效果

当然由于水果名字的长度每次都是随机生成的，你运行时的效果肯定和图中还是不一样的。

3.6.3 RecyclerView 的点击事件

和 ListView 一样，RecyclerView 也必须要能影响点击事件才可以，不然的话就没什么实际用途了。不过不同于 ListView 的是，RecyclerView 并没有提供类似于 `setOnItemClickListener()` 这样的注册监听器方法，而是需要我们自己给子项具体的 View 去注册点击事件，相比于 ListView 来说，实现起来要复杂一些。

那么你可能就有疑问了，为什么 RecyclerView 在各方面的设计都要优于 ListView，偏偏在点击事件上却没有处理得非常好呢？其实不是这样的，ListView 在点击事件上的处理并不人性化，`setOnItemClickListener()` 方法注册的是子项的点击事件，但如果我想点击的是子项里具体的某一个按钮呢？虽然 ListView 也是能做到的，但是实现起来就相对比较麻烦了。为此，RecyclerView 干脆直接摒弃了子项点击事件的监听器，所有的点击事件都由具体的 View 去注册，就再没有这个困扰了。

下面我们来具体学习一下如何在 RecyclerView 中注册点击事件，修改 `FruitAdapter` 中的代码，如下所示：

```
public class FruitAdapter extends RecyclerView.Adapter<FruitAdapter.ViewHolder> {
    private List<Fruit> mFruitList;
```

```

static class ViewHolder extends RecyclerView.ViewHolder {
    View fruitView;
    ImageView fruitImage;
    TextView fruitName;

    public ViewHolder(View view) {
        super(view);
        fruitView = view;
        fruitImage = (ImageView) view.findViewById(R.id.fruit_image);
        fruitName = (TextView) view.findViewById(R.id.fruit_name);
    }
}

public FruitAdapter(List<Fruit> fruitList) {
    mFruitList = fruitList;
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.
        fruit_item, parent, false);
    final ViewHolder holder = new ViewHolder(view);
    holder.fruitView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = holder.getAdapterPosition();
            Fruit fruit = mFruitList.get(position);
            Toast.makeText(v.getContext(), "you clicked view " + fruit.getName(),
                Toast.LENGTH_SHORT).show();
        }
    });
    holder.fruitImage.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            int position = holder.getAdapterPosition();
            Fruit fruit = mFruitList.get(position);
            Toast.makeText(v.getContext(), "you clicked image " + fruit.getName(),
                Toast.LENGTH_SHORT).show();
        }
    });
    return holder;
}

...
}

```

我们先是修改了 `ViewHolder`, 在 `ViewHolder` 中添加了 `fruitView` 变量来保存子项最外层布局的实例, 然后在 `onCreateViewHolder()` 方法中注册点击事件就可以了。这里分别为最外层布局和 `ImageView` 都注册了点击事件, `RecyclerView` 的强大之处也在这里, 它可以轻松实现子项中任意控件或布局的点击事件。我们在两个点击事件中先获取了用户点击的 `position`, 然后通过 `position` 拿到相应的 `Fruit` 实例, 再使用 `Toast` 分别弹出两种不同的内容以示区别。

现在重新运行代码，并点击香蕉的图片部分，效果如图 3.37 所示。可以看到，这时触发了 ImageView 的点击事件。

然后再点击菠萝的文字部分，由于 TextView 并没有注册点击事件，因此点击文字这个事件会被子项的最外层布局捕获到，效果如图 3.38 所示。

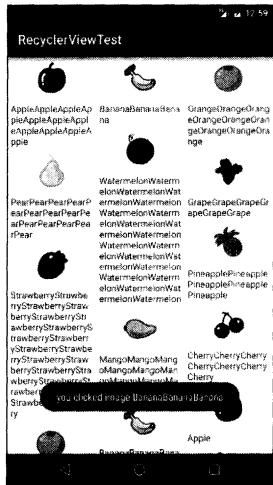


图 3.37 点击香蕉的图片部分

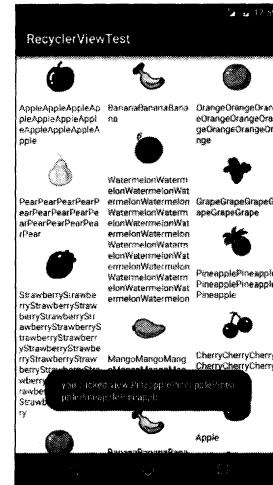


图 3.38 点击菠萝的文字部分

3.7 编写界面的最佳实践

既然已经学习了那么多 UI 开发的知识，也是时候实战一下了。这次我们要综合运用前面所学的大量内容来编写出一个较为复杂且相当美观的聊天界面，你准备好了吗？要先创建一个 `UIBestPractice` 项目才算准备好了哦。

3.7.1 制作 Nine-Patch 图片

在实战正式开始之前，我们还需要先学习一下如何制作 Nine-Patch 图片。你可能之前还没有听说过这个名词，它是一种被特殊处理过的 png 图片，能够指定哪些区域可以被拉伸、哪些区域不可以。

那么 Nine-Patch 图片到底有什么实际作用呢？我们还是通过一个例子来看一下吧。比如说项目中有一张气泡样式的图片 `message_left.png`，如图 3.39 所示。



图 3.39 气泡样式图片

我们将这张图片设置为 LinearLayout 的背景图片，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/message_left"
    >
</LinearLayout>
```

将 LinearLayout 的宽度指定为 `match_parent`，然后将它的背景图设置为 `message_left`，现在运行程序，效果如图 3.40 所示。

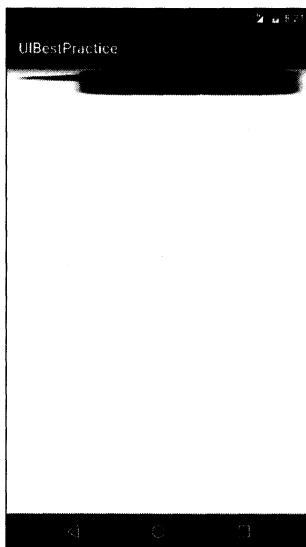


图 3.40 气泡被均匀拉伸的效果

可以看到，由于 `message_left` 的宽度不足以填满整个屏幕的宽度，整张图片被均匀地拉伸了！这种效果非常差，用户肯定是不能容忍的，这时我们就可以使用 Nine-Patch 图片来进行改善。

在 Android sdk 目录下有一个 tools 文件夹，在这个文件夹中找到 `draw9patch.bat` 文件，我们就是使用它来制作 Nine-Patch 图片的。不过，要打开这个文件，必须先将 JDK 的 bin 目录配置到环境变量当中才行，比如你使用的是 Android Studio 内置的 jdk，那么要配置的路径就是`<Android Studio 安装目录>/jre/bin`。如果你还不知道该如何配置环境变量，可以先去参考 6.4.1 小节的内容。

双击打开 `draw9patch.bat` 文件，在导航栏点击 `File→Open 9-patch` 将 `message_left.png` 加载进来，如图 3.41 所示。

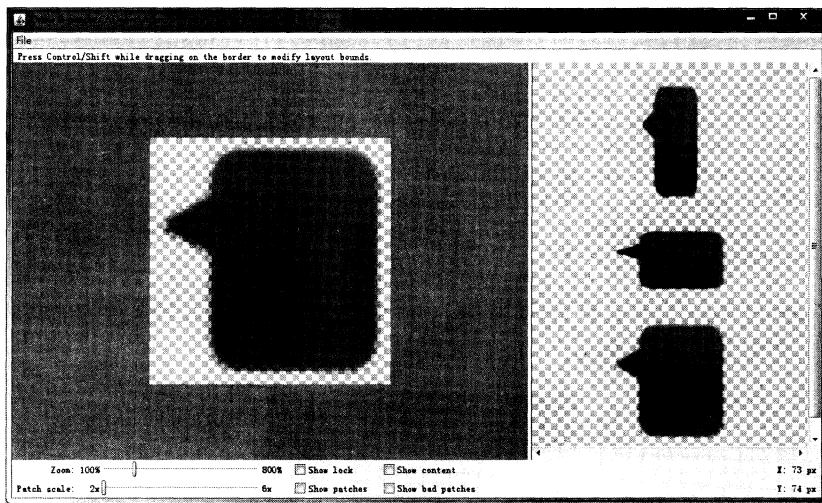


图 3.41 使用 draw9patch 编辑 message_left 图片

我们可以在图片的四个边框绘制一个个的小黑点，在上边框和左边框绘制的部分表示当图片需要拉伸时就拉伸黑点标记的区域，在下边框和右边框绘制的部分表示内容会被放置的区域。使用鼠标在图片的边缘拖动就可以进行绘制了，按住 Shift 键拖动可以进行擦除。绘制完成后效果如图 3.42 所示。

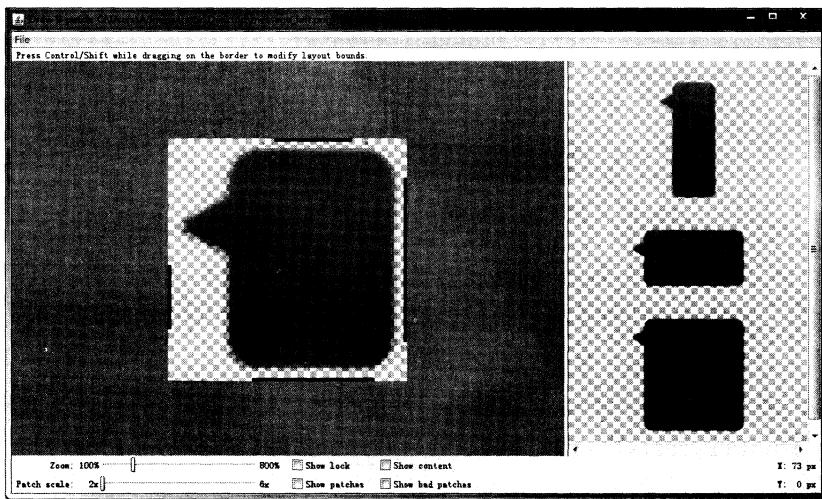


图 3.42 绘制完成后的 message_left 图片

最后点击导航栏 File→Save 9-patch 把绘制好的图片进行保存，此时的文件名就是 message_left.9.png。使用这张图片替换掉之前的 message_left.png 图片，重新运行程序，效果如图 3.43 所示。

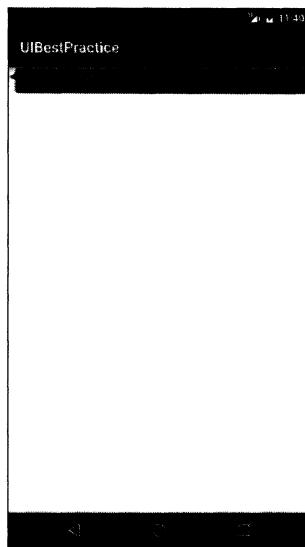


图 3.43 气泡只拉伸绘制区域的效果

这样当图片需要拉伸的时候，就可以只拉伸指定的区域，程序在外观上也有了很大的改进。有了这个知识储备之后，我们就可以进入实战环节了。

3.7.2 编写精美的聊天界面

既然是要编写一个聊天界面，那就肯定要有收到的消息和发出的消息。上一节中我们制作的 message_left.9.png 可以作为收到消息的背景图，那么毫无疑问你还需要再制作一张 message_right.9.png 作为发出消息的背景图。

图片都提供好了之后就可以开始编码了。由于待会我们会用到 RecyclerView，因此首先需要在 app/build.gradle 当中添加依赖库，如下所示：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:recyclerview-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

接下来开始编写主界面，修改 activity_main.xml 中的代码，如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#d8e0e8" >

    <android.support.v7.widget.RecyclerView
```

```
    android:id="@+id/msg_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1" />

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

    <EditText
        android:id="@+id/input_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="Type something here"
        android:maxLines="2" />

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

</LinearLayout>

</LinearLayout>
```

我们在主界面中放置了一个 RecyclerView 用于显示聊天的消息内容，又放置了一个 EditText 用于输入消息，还放置了一个 Button 用于发送消息。这里用到的所有属性都是我们之前学过的，相信你理解起来应该不费力。

然后定义消息的实体类，新建 Msg，代码如下所示：

```
public class Msg {

    public static final int TYPE_RECEIVED = 0;

    public static final int TYPE_SENT = 1;

    private String content;

    private int type;

    public Msg(String content, int type) {
        this.content = content;
        this.type = type;
    }

    public String getContent() {
        return content;
    }

    public int getType() {
```

```
    return type;  
}  
  
}
```

Msg 类中只有两个字段，content 表示消息的内容，type 表示消息的类型。其中消息类型有两个值可选，TYPE_RECEIVED 表示这是一条收到的消息，TYPE_SENT 表示这是一条发出的消息。

接着来编写 RecyclerView 子项的布局，新建 msg_item.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:padding="10dp" >  
  
    <LinearLayout  
        android:id="@+id/left_layout"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="left"  
        android:background="@drawable/message_left" >  
  
        <TextView  
            android:id="@+id/left_msg"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:layout_gravity="center"  
            android:layout_margin="10dp"  
            android:textColor="#fff" />  
  
    </LinearLayout>  
  
    <LinearLayout  
        android:id="@+id/right_layout"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="right"  
        android:background="@drawable/message_right" >  
  
        <TextView  
            android:id="@+id/right_msg"  
            android:layout_width="wrap_content"  
            android:layout_height="wrap_content"  
            android:layout_gravity="center"  
            android:layout_margin="10dp" />  
  
    </LinearLayout>  
  
</LinearLayout>
```

这里我们让收到的消息居左对齐，发出的消息居右对齐，并且分别使用 message_left.9.png 和

message_right.9.png 作为背景图。你可能会有些疑虑，怎么能让收到的消息和发出的消息都放在同一个布局里呢？不用担心，还记得我们前面学过的可见属性吗？只要稍后在代码中根据消息的类型来决定隐藏和显示哪种消息就可以了。

接下来需要创建 RecyclerView 的适配器类，新建类 MsgAdapter，代码如下所示：

```
public class MsgAdapter extends RecyclerView.Adapter<MsgAdapter.ViewHolder> {

    private List<Msg> mMsgList;

    static class ViewHolder extends RecyclerView.ViewHolder {
        LinearLayout leftLayout;
        LinearLayout rightLayout;
        TextView leftMsg;
        TextView rightMsg;

        public ViewHolder(View view) {
            super(view);
            leftLayout = (LinearLayout) view.findViewById(R.id.left_layout);
            rightLayout = (LinearLayout) view.findViewById(R.id.right_layout);
            leftMsg = (TextView) view.findViewById(R.id.left_msg);
            rightMsg = (TextView) view.findViewById(R.id.right_msg);
        }
    }

    public MsgAdapter(List<Msg> msgList) {
        mMsgList = msgList;
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate
            (R.layout.msg_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        Msg msg = mMsgList.get(position);
        if (msg.getType() == Msg.TYPE_RECEIVED) {
            // 如果是收到的消息，则显示左边的消息布局，将右边的消息布局隐藏
            holder.leftLayout.setVisibility(View.VISIBLE);
            holder.rightLayout.setVisibility(View.GONE);
            holder.leftMsg.setText(msg.getContent());
        } else if (msg.getType() == Msg.TYPE_SENT) {
            // 如果是发出的消息，则显示右边的消息布局，将左边的消息布局隐藏
            holder.rightLayout.setVisibility(View.VISIBLE);
            holder.leftLayout.setVisibility(View.GONE);
            holder.rightMsg.setText(msg.getContent());
        }
    }
}
```

```

        }

    }

    @Override
    public int getItemCount() {
        return mMsgList.size();
    }

}

```

以上代码你应该非常熟悉了，和我们学习 RecyclerView 那一节的代码基本是一样的，只不过在 `onBindViewHolder()` 方法中增加了对消息类型的判断。如果这条消息是收到的，则显示左边的消息布局，如果这条消息是发出的，则显示右边的消息布局。

最后修改 MainActivity 中的代码，来为 RecyclerView 初始化一些数据，并给发送按钮加入事件响应，代码如下所示：

```

public class MainActivity extends AppCompatActivity {

    private List<Msg> msgList = new ArrayList<>();

    private EditText inputText;

    private Button send;

    private RecyclerView msgRecyclerView;

    private MsgAdapter adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        initMsgs(); // 初始化消息数据
        inputText = (EditText) findViewById(R.id.input_text);
        send = (Button) findViewById(R.id.send);
        msgRecyclerView = (RecyclerView) findViewById(R.id.msg_recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(this);
        msgRecyclerView.setLayoutManager(layoutManager);
        adapter = new MsgAdapter(msgList);
        msgRecyclerView.setAdapter(adapter);
        send.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String content = inputText.getText().toString();
                if (!"".equals(content)) {
                    Msg msg = new Msg(content, Msg.TYPE_SENT);
                    msgList.add(msg);
                    adapter.notifyItemInserted(msgList.size() - 1); // 当有新消息时，刷新 ListView 中的显示
                    msgRecyclerView.scrollToPosition(msgList.size() - 1); // 将 ListView 定位到最后一行
                    inputText.setText(""); // 清空输入框中的内容
                }
            }
        });
    }
}

```

```

        }
    });
}

private void initMsgs() {
    Msg msg1 = new Msg("Hello guy.", Msg.TYPE_RECEIVED);
    msgList.add(msg1);
    Msg msg2 = new Msg("Hello. Who is that?", Msg.TYPE_SENT);
    msgList.add(msg2);
    Msg msg3 = new Msg("This is Tom. Nice talking to you. ", Msg.TYPE_RECEIVED);
    msgList.add(msg3);
}
}

```

在 `initMsgs()` 方法中我们先初始化了几条数据用于在 RecyclerView 中显示。然后在发送按钮的点击事件里获取了 EditText 中的内容，如果内容不为 null 则创建出一个新的 Msg 对象，并把它添加到 msgList 列表中去。之后又调用了适配器的 `notifyItemInserted()` 方法，用于通知列表有新的数据插入，这样新增的一条消息才能够在 RecyclerView 中显示。接着调用 RecyclerView 的 `scrollToPosition()` 方法将显示的数据定位到最后一条，以保证一定可以看到最后发出的一条消息。最后调用 EditText 的 `setText()` 方法将输入的内容清空。

这样所有的工作就都完成了，终于可以检验一下我们的成果了，运行程序之后你将会看到非常美观的聊天界面，并且可以输入和发送消息，如图 3.44 所示。



图 3.44 精美的聊天界面

相信这个例子的实战过程不仅加深了你对本章中所学 UI 知识的理解，还让你有了如何灵活运用这些知识来设计出优秀界面的思路。这一章也是学了不少东西，让我们来总结一下吧。

3.8 小结与点评

虽然本章的内容很多，但我觉得学习起来应该还是挺愉快的吧。不同于上一章中我们来来回回使用那几个按钮，本章可以说是使用了各种各样的控件，制作出了丰富多彩的界面。尤其是在实战环节，编写出了那么精美的聊天界面，你的满足感应该比上一章还要强吧？

本章从 Android 中的一些常见控件开始入手，依次介绍了基本布局的用法、自定义控件的方法、ListView 的详细用法以及 RecyclerView 的使用，基本已经将重要的 UI 知识点全部覆盖了。想想在开始的时候我说不推荐使用可视化的编辑工具，而是应该全部使用 XML 的方式来编写界面，现在你是不是已经感觉使用 XML 非常简单了呢？以后不管面对多么复杂的界面，我希望你都能够自信满满，因为真正理解了界面编写的原理之后，是没有什么能够难得倒你的。

不过到目前为止，我们还只是学习了 Android 手机方面的开发技巧，下一章将会涉及一些 Android 平板方面的知识点，能够同时兼容手机和平板也是自 Android 4.0 系统开始就支持的特性。适当地放松和休息一段时间后，我们再来继续前行吧！

第 4 章

手机平板要兼顾——探究碎片

当今是移动设备发展非常迅速的时代，不仅手机已经成为了生活必需品，就连平板电脑也变得越来越普及。平板电脑和手机最大的区别就在于屏幕的大小，一般手机屏幕的大小会在 3 英寸到 6 英寸之间，而一般平板电脑屏幕的大小会在 7 英寸到 10 英寸之间。屏幕大小差距过大有可能会让同样的界面在视觉效果上有较大的差异，比如一些界面在手机上看起来非常美观，但在平板电脑上看起来就可能会有控件被过分拉长、元素之间空隙过大等情况。

作为一名专业的 Android 开发人员，能够同时兼顾手机和平板的开发是我们必须做到的事情。Android 自 3.0 版本开始引入了碎片的概念，它可以让界面在平板上更好地展示，下面我们就来一起学习一下。

4.1 碎片是什么

碎片（Fragment）是一种可以嵌入在活动当中的 UI 片段，它能让程序更加合理和充分地利用大屏幕的空间，因而在平板上应用得非常广泛。虽然碎片对你来说应该是个全新的概念，但我相信你学习起来应该毫不费力，因为它和活动实在是太像了，同样都能包含布局，同样都有自己的生命周期。你甚至可以将碎片理解成一个迷你型的活动，虽然这个迷你型的活动有可能和普通的活动是一样大的。

那么究竟要如何使用碎片才能充分地利用平板屏幕的空间呢？想象我们正在开发一个新闻应用，其中一个界面使用 RecyclerView 展示了一组新闻的标题，当点击了其中一个标题时，就打开另一个界面显示新闻的详细内容。如果是在手机中设计，我们可以将新闻标题列表放在一个活动中，将新闻的详细内容放在另一个活动中，如图 4.1 所示。

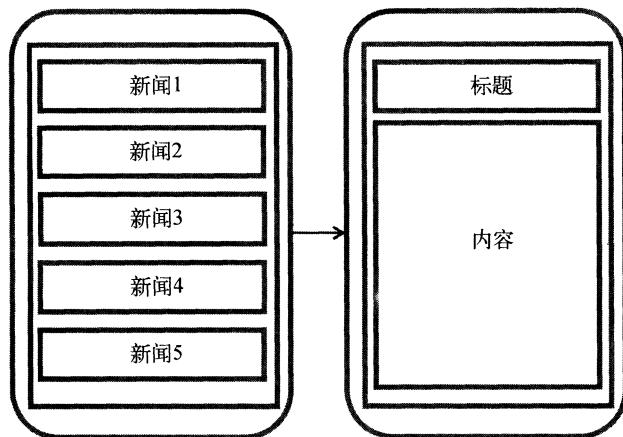


图 4.1 手机的设计方案

可是如果在平板上也这么设计，那么新闻标题列表将会被拉长至填充满整个平板的屏幕，而新闻的标题一般都不会太长，这样将会导致界面上有大量的空白区域，如图 4.2 所示。

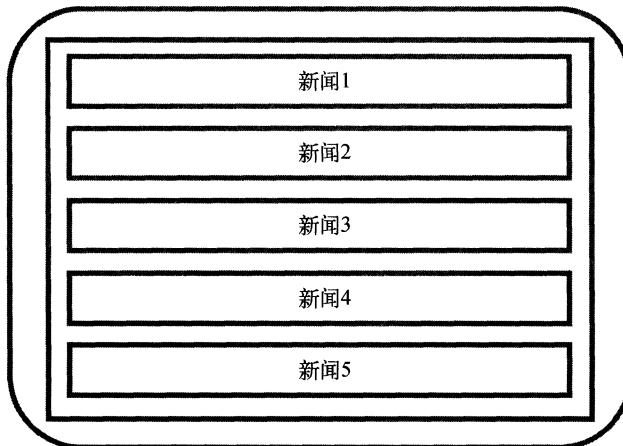


图 4.2 平板的新闻列表

因此，更好的设计方案是将新闻标题列表界面和新闻详细内容界面分别放在两个碎片中，然后在同一个活动里引入这两个碎片，这样就可以将屏幕空间充分地利用起来了，如图 4.3 所示。

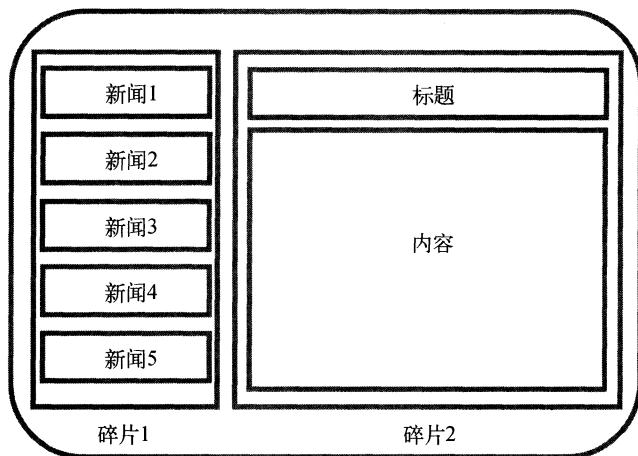


图 4.3 平板的双页设计

4.2 碎片的使用方式

介绍了这么多抽象的东西，也是时候学习一下碎片的具体用法了。你已经知道，碎片通常都是在平板开发中使用的，因此我们首先要做的就是创建一个平板模拟器。创建模拟器的方法我们在第1章已经学过了，创建完成后启动平板模拟器，效果如图4.4所示。

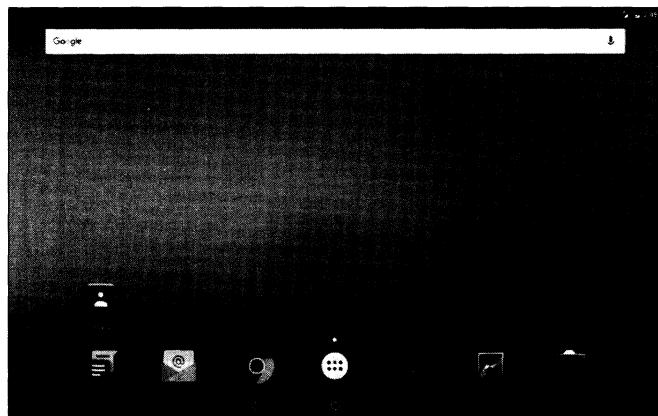


图 4.4 平板模拟器的运行效果

好了，准备工作都完成了，接着新建一个 FragmentTest 项目，然后开始我们的碎片探索之旅吧。

4.2.1 碎片的简单用法

这里我们准备先写一个最简单的碎片示例来练练手，在一个活动中添加两个碎片，并让这

两个碎片平分活动空间。

新建一个左侧碎片布局 left_fragment.xml，代码如下所示：

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

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Button"
    />

</LinearLayout>
```

这个布局非常简单，只放置了一个按钮，并让它水平居中显示。然后新建右侧碎片布局 right_fragment.xml，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#00ff00"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
        android:text="This is right fragment"
    />

</LinearLayout>
```

可以看到，我们将这个布局的背景色设置成了绿色，并放置了一个 TextView 用于显示一段文本。

接着新建一个 `LeftFragment` 类，并让它继承自 `Fragment`。注意，这里可能会有两个不同包下的 `Fragment` 供你选择，一个是系统内置的 `android.app.Fragment`，一个是 `support-v4` 库中的 `android.support.v4.app.Fragment`。这里我强烈建议你使用 `support-v4` 库中的 `Fragment`，因为它可以让碎片在所有 Android 系统版本中保持功能一致性。比如说在 `Fragment` 中嵌套使用 `Fragment`，这个功能是在 Android 4.2 系统中才开始支持的，如果你使用的是系统内置的 `Fragment`，那么很遗憾，4.2 系统之前的设备运行你的程序就会崩溃。而使用 `support-v4` 库中的 `Fragment` 就不会出现这个问题，只要你保证使用的是最新的 `support-v4` 库就可以了。另外，我们并不需要在 `build.gradle` 文件中添加 `support-v4` 库的依赖，因为 `build.gradle` 文件中已经添加了 `appcompat-v7`

库的依赖，而这个库会将 support-v4 库也一起引入进来。

现在编写一下 `LeftFragment` 中的代码，如下所示：

```
public class LeftFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.left_fragment, container, false);
        return view;
    }

}
```

这里仅仅是重写了 `Fragment` 的 `onCreateView()` 方法，然后在这个方法中通过 `LayoutInflater` 的 `inflate()` 方法将刚才定义的 `left_fragment` 布局动态加载进来，整个方法简单明了。接着我们用同样的方法再新建一个 `RightFragment`，代码如下所示：

```
public class RightFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }

}
```

基本上代码都是相同的，相信已经没有必要再做什么解释了。接下来修改 `activity_main.xml` 中的代码，如下所示：

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

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

</LinearLayout>
```

可以看到，我们使用了`<fragment>`标签在布局中添加碎片，其中指定的大多数属性都是你熟悉的，只不过这里还需要通过`android:name`属性来显式指明要添加的碎片类名，注意一定要将类的包名也加上。

这样最简单的碎片示例就已经写好了，现在运行一下程序，效果如图 4.5 所示。



图 4.5 碎片的简单运行效果

正如我们所期待的一样，两个碎片平分了整个活动的布局。不过这个例子实在是太简单了，在真正的项目中很难有什么实际的作用，因此我们马上来看一看，关于碎片更加高级的使用技巧。

4.2.2 动态添加碎片

在上一节当中，你已经学会了在布局文件中添加碎片的方法，不过碎片真正的强大之处在于，它可以在程序运行时动态地添加到活动当中。根据具体情况来动态地添加碎片，你就可以将程序界面定制得更加多样化。

我们还是在上一节代码的基础上继续完善，新建`another_right_fragment.xml`，代码如下所示：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:background="#ffff00"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp"
```

```

        android:text="This is another right fragment"
    />

</LinearLayout>

```

这个布局文件的代码和 right_fragment.xml 中的代码基本相同，只是将背景色改成了黄色，并将显示的文字改了改。然后新建 AnotherRightFragment 作为另一个右侧碎片，代码如下所示：

```

public class AnotherRightFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.another_right_fragment, container,
                                     false);
        return view;
    }

}

```

代码同样非常简单，在 onCreateView() 方法中加载了刚刚创建的 another_right_fragment 布局。这样我们就准备好了另一个碎片，接下来看一下如何将它动态地添加到活动当中。修改 activity_main.xml，代码如下所示：

```

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

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/right_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" >
    </FrameLayout>

</LinearLayout>

```

可以看到，现在将右侧碎片替换成了一个 FrameLayout 中，还记得这个布局吗？在上一章中我们学过，这是 Android 中最简单的一种布局，所有的控件默认都会摆放在布局的左上角。由于这里仅需要在布局里放入一个碎片，不需要任何定位，因此非常适合使用 FrameLayout。

下面我们将向 FrameLayout 里添加内容，从而实现动态添加碎片的功能。修改 MainActivity 中的代码，如下所示：

```

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(this);
        replaceFragment(new RightFragment());
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button:
                replaceFragment(new AnotherRightFragment());
                break;
            default:
                break;
        }
    }

    private void replaceFragment(Fragment fragment) {
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        transaction.replace(R.id.right_layout, fragment);
        transaction.commit();
    }
}

```

可以看到，首先我们给左侧碎片中的按钮注册了一个点击事件，然后调用 `replaceFragment()` 方法动态添加了 `RightFragment` 这个碎片。当点击左侧碎片中的按钮时，又会调用 `replaceFragment()` 方法将右侧碎片替换成 `AnotherRightFragment`。结合 `replaceFragment()` 方法中的代码可以看出，动态添加碎片主要分为 5 步。

- (1) 创建待添加的碎片实例。
- (2) 获取 `FragmentManager`，在活动中可以直接通过调用 `getSupportFragmentManager()` 方法得到。
- (3) 开启一个事务，通过调用 `beginTransaction()` 方法开启。
- (4) 向容器内添加或替换碎片，一般使用 `replace()` 方法实现，需要传入容器的 id 和待添加的碎片实例。
- (5) 提交事务，调用 `commit()` 方法来完成。

这样就完成了在活动中动态添加碎片的功能，重新运行程序，可以看到和之前相同的界面，然后点击一下按钮，效果如图 4.6 所示。

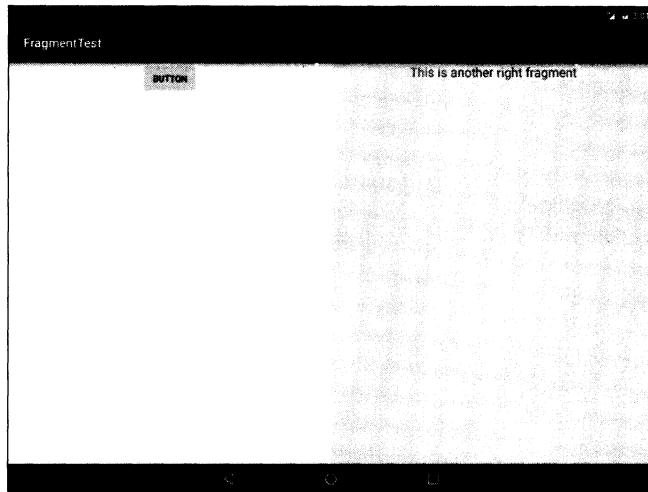


图 4.6 动态添加碎片的效果

4.2.3 在碎片中模拟返回栈

在上一小节中，我们成功实现了向活动中动态添加碎片的功能，不过你尝试一下就会发现，通过点击按钮添加了一个碎片之后，这时按下 Back 键程序就会直接退出。如果这里我们想模仿类似于返回栈的效果，按下 Back 键可以回到上一个碎片，该如何实现呢？

其实很简单，`FragmentTransaction` 中提供了一个 `addToBackStack()` 方法，可以用于将一个事务添加到返回栈中，修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    ...

    private void replaceFragment(Fragment fragment) {
        FragmentManager fragmentManager = getSupportFragmentManager();
        FragmentTransaction transaction = fragmentManager.beginTransaction();
        transaction.replace(R.id.right_layout, fragment);
        transaction.addToBackStack(null);
        transaction.commit();
    }
}
```

这里我们在事务提交之前调用了 `FragmentTransaction` 的 `addToBackStack()` 方法，它可以接收一个名字用于描述返回栈的状态，一般传入 `null` 即可。现在重新运行程序，并点击按钮将 `AnotherRightFragment` 添加到活动中，然后按下 Back 键，你会发现程序并没有退出，而是回到了 `RightFragment` 界面，继续按下 Back 键，`RightFragment` 界面也会消失，再次按下 Back 键，程序才会退出。

4.2.4 碎片和活动之间进行通信

虽然碎片都是嵌入在活动中显示的，可是实际上它们的关系并没有那么亲密。你可以看出，碎片和活动都是各自存在于一个独立的类当中的，它们之间并没有那么明显的方式来直接进行通信。如果想要在活动中调用碎片里的方法，或者在碎片中调用活动里的方法，应该如何实现呢？

为了方便碎片和活动之间进行通信，`FragmentManager` 提供了一个类似于 `findViewById()` 的方法，专门用于从布局文件中获取碎片的实例，代码如下所示：

```
RightFragment rightFragment = (RightFragment) getFragmentManager()
    .findFragmentById(R.id.right_fragment);
```

调用 `FragmentManager` 的 `findFragmentById()` 方法，可以在活动中得到相应碎片的实例，然后就能轻松地调用碎片里的方法了。

掌握了如何在活动中调用碎片里的方法，那在碎片中又该怎样调用活动里的方法呢？其实这就更简单了，在每个碎片中都可以通过调用 `getActivity()` 方法来得到和当前碎片相关联的活动实例，代码如下所示：

```
MainActivity activity = (MainActivity) getActivity();
```

有了活动实例之后，在碎片中调用活动里的方法就变得轻而易举了。另外当碎片中需要使用 `Context` 对象时，也可以使用 `getActivity()` 方法，因为获取到的活动本身就是一个 `Context` 对象。

这时不知道你心中会不会产生一个疑问：既然碎片和活动之间的通信问题已经解决了，那么碎片和碎片之间可不可以进行通信呢？

说实在的，这个问题并没有看上去那么复杂，它的基本思路非常简单，首先在一个碎片中可以得到与它相关联的活动，然后再通过这个活动去获取另外一个碎片的实例，这样也就实现了不同碎片之间的通信功能，因此这里我们的答案是肯定的。

4.3 碎片的生命周期

和活动一样，碎片也有自己的生命周期，并且它和活动的生命周期实在是太像了，我相信你很快就能学会，下面我们马上就来看一下。

4.3.1 碎片的状态和回调

还记得每个活动在其生命周期内可能会有哪几种状态吗？没错，一共有运行状态、暂停状态、停止状态和销毁状态这 4 种。类似地，每个碎片在其生命周期内也可能会经历这几种状态，只不过在一些细小的地方会有部分区别。

1. 运行状态

当一个碎片是可见的，并且它所关联的活动正处于运行状态时，该碎片也处于运行状态。

2. 暂停状态

当一个活动进入暂停状态时（由于另一个未占满屏幕的活动被添加到了栈顶），与它相关联的可见碎片就会进入到暂停状态。

3. 停止状态

当一个活动进入停止状态时，与它相关联的碎片就会进入到停止状态，或者通过调用 FragmentTransaction 的 `remove()`、`replace()` 方法将碎片从活动中移除，但如果在事务提交之前调用 `addToBackStack()` 方法，这时的碎片也会进入到停止状态。总的来说，进入停止状态的碎片对用户来说是完全不可见的，有可能会被系统回收。

4. 销毁状态

碎片总是依附于活动而存在的，因此当活动被销毁时，与它相关联的碎片就会进入到销毁状态。或者通过调用 FragmentTransaction 的 `remove()`、`replace()` 方法将碎片从活动中移除，但在事务提交之前并没有调用 `addToBackStack()` 方法，这时的碎片也会进入到销毁状态。

结合之前的活动状态，相信你理解起来应该毫不费力吧。同样地，Fragment 类中也提供了一系列的回调方法，以覆盖碎片生命周期的每个环节。其中，活动中有的回调方法，碎片中几乎都有，不过碎片还提供了一些附加的回调方法，那我们就重点看一下这几个回调。

- `onAttach()`。当碎片和活动建立关联的时候调用。
- `onCreateView()`。为碎片创建视图（加载布局）时调用。
- `onActivityCreated()`。确保与碎片相关联的活动一定已经创建完毕的时候调用。
- `onDestroyView()`。当与碎片关联的视图被移除的时候调用。
- `onDetach()`。当碎片和活动解除关联的时候调用。

碎片完整的生命周期示意图可参考图 4.7，图片源自 Android 官网。

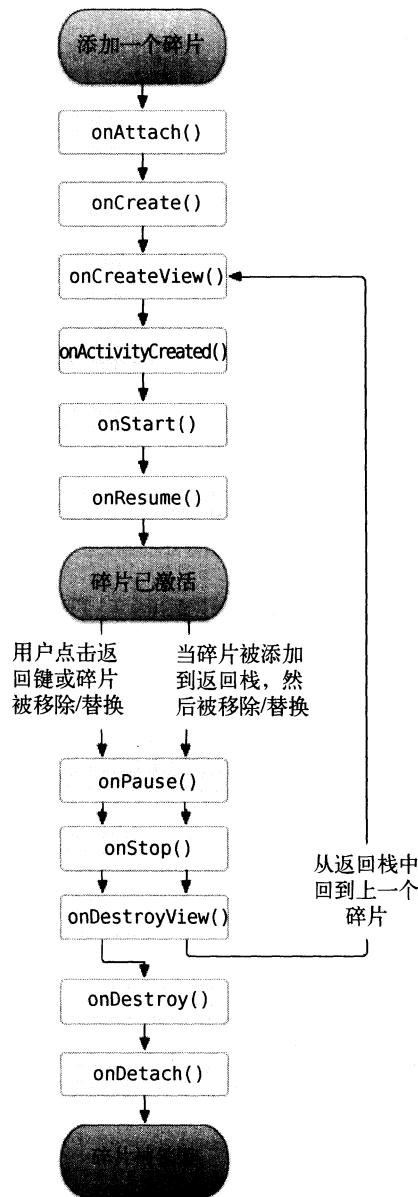


图 4.7 碎片的生命周期

4.3.2 体验碎片的生命周期

为了让你能够更加直观地体验碎片的生命周期，我们还是通过一个例子来实践一下。例子很简单，仍然是在 FragmentTest 项目的基础上改动的。

修改 RightFragment 中的代码，如下所示：

```
public class RightFragment extends Fragment {

    public static final String TAG = "RightFragment";

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        Log.d(TAG, "onAttach");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        Log.d(TAG, "onCreateView");
        View view = inflater.inflate(R.layout.right_fragment, container, false);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        Log.d(TAG, "onActivityCreated");
    }

    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop");
    }
}
```

```

    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        Log.d(TAG, "onDestroyView");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy");
    }

    @Override
    public void onDetach() {
        super.onDetach();
        Log.d(TAG, "onDetach");
    }

}

```

我们在 RightFragment 中的每一个回调方法里都加入了打印日志的代码，然后重新运行程序，这时观察 logcat 中的打印信息，如图 4.8 所示。



图 4.8 启动程序时的打印日志

可以看到，当 RightFragment 第一次被加载到屏幕上时，会依次执行 `onAttach()`、`onCreate()`、`onCreateView()`、`onActivityCreated()`、`onStart()` 和 `onResume()` 方法。然后点击 LeftFragment 中的按钮，此时打印信息如图 4.9 所示。



图 4.9 替换成 AnotherRightFragment 时的打印日志

由于 AnotherRightFragment 替换了 RightFragment，此时的 RightFragment 进入了停止状态，因此 `onPause()`、`onStop()` 和 `onDestroyView()` 方法会得到执行。当然如果在替换的时候没有调用 `addToBackStack()` 方法，此时的 RightFragment 就会进入销毁状态，`onDestroy()` 和 `onDetach()` 方法就会得到执行。

接着按下 Back 键，RightFragment 会重新回到屏幕，打印信息如图 4.10 所示。

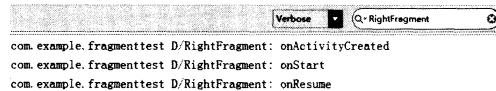


图 4.10 返回 RightFragment 时的打印日志

由于 RightFragment 重新回到了运行状态，因此 `onActivityCreated()`、`onStart()` 和 `onResume()` 方法会得到执行。注意此时 `onCreate()` 和 `onCreateView()` 方法并不会执行，因为我们借助了 `addToBackStack()` 方法使得 RightFragment 和它的视图并没有销毁。

再次按下 Back 键退出程序，打印信息如图 4.11 所示。



图 4.11 退出程序时的打印日志

依次会执行 `onPause()`、`onStop()`、`onDestroyView()`、`onDestroy()` 和 `onDetach()` 方法，最终将活动和碎片一起销毁。这样碎片完整的生命周期你也体验了一遍，是不是理解得更加深刻了？

另外值得一提的是，在碎片中你也是可以通过 `onSaveInstanceState()` 方法来保存数据的，因为进入停止状态的碎片有可能在系统内存不足的时候被回收。保存下来的数据在 `onCreate()`、`onCreateView()` 和 `onActivityCreated()` 这 3 个方法中你都可以重新得到，它们都含有一个 `Bundle` 类型的 `savedInstanceState` 参数。具体的代码我就不在这里给出了，如果你忘记了该如何编写，可以参考 2.4.5 小节。

4.4 动态加载布局的技巧

虽然动态添加碎片的功能很强大，可以解决很多实际开发中的问题，但是它毕竟只是在一个布局文件中进行一些添加和替换操作。如果程序能够根据设备的分辨率或屏幕大小在运行时来决定加载哪个布局，那我们可发挥的空间就更多了。因此本节我们就来探讨一下 Android 中动态加载布局的技巧。

4.4.1 使用限定符

如果你经常使用平板电脑，应该会发现现在很多的平板应用都采用的是双页模式（程序会在左侧的面板上显示一个包含子项的列表，在右侧的面板上显示内容），因为平板电脑的屏幕足够大，完全可以同时显示下两页的内容，但手机的屏幕一次就只能显示一页的内容，因此两个页面需要分开显示。

那么怎样才能在运行时判断程序应该是使用双页模式还是单页模式呢？这就需要借助限定符（Qualifiers）来实现了。下面我们通过一个例子来学习一下它的用法，修改 FragmentTest 项目中的 activity_main.xml 文件，代码如下所示：

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

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

这里将多余的代码都删掉，只留下一个左侧碎片，并让它充满整个父布局。接着在 res 目录下新建 layout-large 文件夹，在这个文件夹下新建一个布局，也叫作 activity_main.xml，代码如下所示：

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

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

可以看到，layout/activity_main 布局只包含了一个碎片，即单页模式，而 layout-large/activity_main 布局包含了两个碎片，即双页模式。其中 large 就是一个限定符，那些屏幕被认为是 large 的设备就会自动加载 layout-large 文件夹下的布局，而小屏幕的设备则还是会加载 layout 文件夹下的布局。

然后将 MainActivity 中 replaceFragment() 方法里的代码注释掉，并在平板模拟器上重新运行程序，效果如图 4.12 所示。

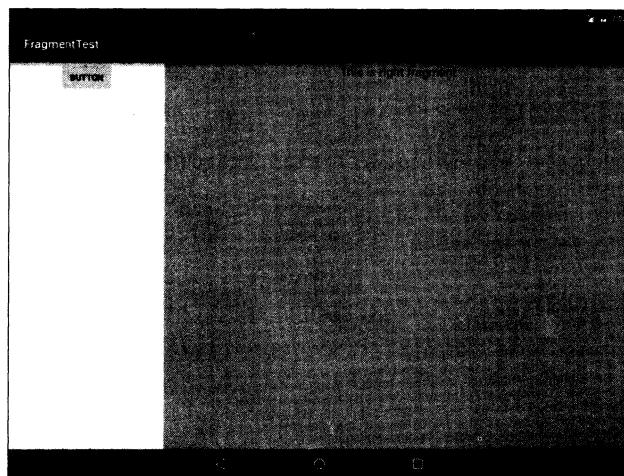


图 4.12 双页模式运行效果

再启动一个手机模拟器，并在这个模拟器上重新运行程序，效果如图 4.13 所示。



图 4.13 单页模式运行效果

这样我们就实现了在程序运行时动态加载布局的功能。

Android 中一些常见的限定符可以参考下表。

屏幕特征	限定符	描述
大小	small	提供给小屏幕设备的资源
	normal	提供给中等屏幕设备的资源
	large	提供给大屏幕设备的资源
	xlarge	提供给超大屏幕设备的资源

(续)

屏幕特征	限定符	描述
分辨率	ldpi	提供给低分辨率设备的资源（120dpi以下）
	mdpi	提供给中等分辨率设备的资源（120dpi~160dpi）
	hdpi	提供给高分辨率设备的资源（160dpi~240dpi）
	xhdpi	提供给超高分辨率设备的资源（240dpi~320dpi）
	xxhdpi	提供给超超高分辨率设备的资源（320dpi~480dpi）
方向	land	提供给横屏设备的资源
	port	提供给竖屏设备的资源

4.4.2 使用最小宽度限定符

在上一小节中我们使用 `large` 限定符成功解决了单页双页的判断问题，不过很快又有一个新的问题出现了，`large` 到底是指多大呢？有的时候我们希望可以更加灵活地为不同设备加载布局，不管它们是不是被系统认定为 `large`，这时就可以使用最小宽度限定符（Smallest-width Qualifier）了。

最小宽度限定符允许我们对屏幕的宽度指定一个最小值（以 `dp` 为单位），然后以这个最小值为临界点，屏幕宽度大于这个值的设备就加载一个布局，屏幕宽度小于这个值的设备就加载另一个布局。

在 `res` 目录下新建 `layout-sw600dp` 文件夹，然后在这个文件夹下新建 `activity_main.xml` 布局，代码如下所示：

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

    <fragment
        android:id="@+id/left_fragment"
        android:name="com.example.fragmenttest.LeftFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <fragment
        android:id="@+id/right_fragment"
        android:name="com.example.fragmenttest.RightFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" />

</LinearLayout>
```

这就意味着，当程序运行在屏幕宽度大于 `600dp` 的设备上时，会加载 `layout-sw600dp/activity_main` 布局，当程序运行在屏幕宽度小于 `600dp` 的设备上时，则仍然加载默认的 `layout/activity_main` 布局。

4.5 碎片的最佳实践——一个简易版的新闻应用

现在你已经将关于碎片的重要知识点都掌握得差不多了，不过在灵活运用方面可能还有些欠缺，因此下面该进入我们本章的最佳实践环节了。

前面有提到过，碎片很多时候都是在平板开发当中使用的，主要是为了解决屏幕空间不能充分利用的问题。那是不是就表明，我们开发的程序都需要提供一个手机版和一个 Pad 版呢？确实有不少公司都是这么做的，但是这样会浪费很多的人力物力。因为维护两个版本的代码成本很高，每当增加什么新功能时，需要在两份代码里各写一遍，每当发现一个 bug 时，需要在两份代码里各修改一次。因此今天我们最佳实践的内容就是，教你如何编写同时兼容手机和平板的应用程序。

还记得在本章开始的时候提到过的一个新闻应用吗？现在我们就将运用本章中所学的知识来编写一个简易版的新闻应用，并且要求它是可以同时兼容手机和平板的。新建好一个 FragmentBestPractice 项目，然后开始动手吧！

由于待会在编写新闻列表时会使用到 RecyclerView，因此首先需要在 app/build.gradle 当中添加依赖库，如下所示：

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:24.2.1'  
    testCompile 'junit:junit:4.12'  
    compile 'com.android.support:recyclerview-v7:24.2.1'  
}
```

接下来我们要准备好一个新闻的实体类，新建类 News，代码如下所示：

```
public class News {  
  
    private String title;  
  
    private String content;  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public String getContent() {  
        return content;  
    }  
  
    public void setContent(String content) {  
        this.content = content;  
    }  
}
```

`News` 类的代码还是比较简单的，`title` 字段表示新闻标题，`content` 字段表示新闻内容。接着新建布局文件 `news_content_frag.xml`，用于作为新闻内容的布局：

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

    <LinearLayout
        android:id="@+id/visibility_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:visibility="invisible" >

        <TextView
            android:id="@+id/news_title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:padding="10dp"
            android:textSize="20sp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="1dp"
            android:background="#000" />

        <TextView
            android:id="@+id/news_content"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:padding="15dp"
            android:textSize="18sp" />

    </LinearLayout>

    <View
        android:layout_width="1dp"
        android:layout_height="match_parent"
        android:layout_alignParentLeft="true"
        android:background="#000" />

</RelativeLayout>
```

新闻内容的布局主要可以分为两个部分，头部部分显示新闻标题，正文部分显示新闻内容，中间使用一条细线分隔开。这里的细线是利用 `View` 来实现的，将 `View` 的宽或高设置为 `1dp`，再通过 `background` 属性给细线设置一下颜色就可以了。这里我们把细线设置成黑色。

然后再新建一个 `NewsContentFragment` 类，继承自 `Fragment`，代码如下所示：

```
public class NewsContentFragment extends Fragment {
```

```

private View view;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    view = inflater.inflate(R.layout.news_content_frag, container, false);
    return view;
}

public void refresh(String newsTitle, String newsContent) {
    View visibilityLayout = view.findViewById(R.id.visibility_layout);
    visibilityLayout.setVisibility(View.VISIBLE);
    TextView newsTitleText = (TextView) view.findViewById(R.id.news_title);
    TextView newsContentText = (TextView) view.findViewById(R.id.news_content);
    newsTitleText.setText(newsTitle); // 刷新新闻的标题
    newsContentText.setText(newsContent); // 刷新新闻的内容
}
}

```

首先在 `onCreateView()` 方法里加载了我们刚刚创建的 `news_content_frag` 布局，这个没什么好解释的。接下来又提供了一个 `refresh()` 方法，这个方法就是用于将新闻的标题和内容显示在界面上的。可以看到，这里通过 `findViewById()` 方法分别获取到新闻标题和内容的控件，然后将方法传递进来的参数设置进去。

这样我们就把新闻内容的碎片和布局都创建好了，但是它们都是在双页模式中使用的，如果想在单页模式中使用的话，我们还需要再创建一个活动。右击 `com.example.fragmentbestpractice` 包 → New → Activity → Empty Activity，新建一个 `NewsContentActivity`，并将布局名指定成 `news_content`，然后修改 `news_content.xml` 中的代码，如下所示：

```

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

    <fragment
        android:id="@+id/news_content_fragment"
        android:name="com.example.fragmentbestpractice.NewsContentFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</LinearLayout>

```

这里我们充分发挥了代码的复用性，直接在布局中引入了 `NewsContentFragment`，这样也就相当于把 `news_content_frag` 布局的内容自动加了进来。

然后修改 `NewsContentActivity` 中的代码，如下所示：

```
public class NewsContentActivity extends AppCompatActivity {
```

```

public static void actionStart(Context context, String newsTitle, String
newsContent) {
    Intent intent = new Intent(context, NewsContentActivity.class);
    intent.putExtra("news_title", newsTitle);
    intent.putExtra("news_content", newsContent);
    context.startActivity(intent);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.news_content);
    String newsTitle = getIntent().getStringExtra("news_title"); // 获取传入的新
   闻标题
    String newsContent = getIntent().getStringExtra("news_content"); // 获取传入的新闻内
    容
    NewsContentFragment newsContentFragment = (NewsContentFragment)
    getSupportFragmentManager().findFragmentById(R.id.news_content_fragment);
    newsContentFragment.refresh(newsTitle, newsContent); // 刷新 NewsContent-
    Fragment 界面
}

}

```

可以看到，在 `onCreate()` 方法中我们通过 `Intent` 获取到了传入的新闻标题和新闻内容，然后调用 `FragmentManager` 的 `findFragmentById()` 方法得到了 `NewsContentFragment` 的实例，接着调用它的 `refresh()` 方法，并将新闻的标题和内容传入，就可以把这些数据显示出来了。注意这里我们还提供了一个 `actionStart()` 方法，还记得它的作用吗？如果忘记的话就再去阅读一遍 2.6.3 小节吧。

接下来还需要再创建一个用于显示新闻列表的布局，新建 `news_title_frag.xml`，代码如下所示：

```

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

    <android.support.v7.widget.RecyclerView
        android:id="@+id/news_title_recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</LinearLayout>

```

这个布局的代码就非常简单了，里面只有一个用于显示新闻列表的 `RecyclerView`。既然要用到 `RecyclerView`，那么就必定少不了子项的布局。新建 `news_item.xml` 作为 `RecyclerView` 子项的布局，代码如下所示：

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

```

    android:id="@+id/news_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:ellipsize="end"
    android:textSize="18sp"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:paddingTop="15dp"
    android:paddingBottom="15dp" />

```

子项的布局也非常简单，只有一个 TextView。仔细观察 TextView，你会发现其中有几个属性是我们之前没有学过的。`android:padding` 表示给控件的周围加上补白，这样不至于让文本内容会紧靠在边缘上。`android:singleLine` 设置为 `true` 表示让这个 TextView 只能单行显示。`android:ellipsize` 用于设定当文本内容超出控件宽度时，文本的缩略方式，这里指定成 `end` 表示在尾部进行缩略。

既然新闻列表和子项的布局都已经创建好了，那么接下来我们就需要一个用于展示新闻列表的地方。这里新建 NewsTitleFragment 作为展示新闻列表的碎片，代码如下所示：

```

public class NewsTitleFragment extends Fragment {

    private boolean isTwoPane;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.news_title_frag, container, false);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        if (getActivity().findViewById(R.id.news_content_layout) != null) {
            isTwoPane = true; // 可以找到 news_content_layout 布局时，为双页模式
        } else {
            isTwoPane = false; // 找不到 news_content_layout 布局时，为单页模式
        }
    }
}

```

可以看到，NewsTitleFragment 中并没有多少代码，在 `onCreateView()` 方法中加载了 `news_title_frag` 布局，这个没什么好说的。我们注意看一下 `onActivityCreated()` 方法，这个方法通过在活动中能否找到一个 id 为 `news_content_layout` 的 View 来判断当前是双页模式还是单页模式，因此我们需要让这个 id 为 `news_content_layout` 的 View 只在双页模式中才会出现。

那么怎样才能实现这个功能呢？其实并不复杂，只需要借助我们刚刚学过的限定符就可以

了。首先修改 activity_main.xml 中的代码，如下所示：

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/news_title_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
    />

</FrameLayout>
```

上述代码表示，在单页模式下，只会加载一个新闻标题的碎片。

然后新建 layout-sw600dp 文件夹，在这个文件夹下再新建一个 activity_main.xml 文件，代码如下所示：

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

    <fragment
        android:id="@+id/news_title_fragment"
        android:name="com.example.fragmentbestpractice.NewsTitleFragment"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1" />

    <FrameLayout
        android:id="@+id/news_content_layout"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3" >

        <fragment
            android:id="@+id/news_content_fragment"
            android:name="com.example.fragmentbestpractice.NewsContentFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />
    </FrameLayout>

</LinearLayout>
```

可以看出，在双页模式下我们同时引入了两个碎片，并将新闻内容的碎片放在了一个 FrameLayout 布局下，而这个布局的 id 正是 news_content_layout。因此，能够找到这个 id 的时候就是双页模式，否则就是单面模式。

现在我们已经将绝大部分的工作都完成了，但还剩下至关重要的一点，就是在 NewsTitle-

Fragment 中通过 RecyclerView 将新闻列表展示出来。我们在 NewsTitleFragment 中新建一个内部类 NewsAdapter 来作为 RecyclerView 的适配器，如下所示：

```
public class NewsTitleFragment extends Fragment {

    private boolean isTwoPane;

    ...

    class NewsAdapter extends RecyclerView.Adapter<NewsAdapter.ViewHolder> {

        private List<News> mNewsList;

        class ViewHolder extends RecyclerView.ViewHolder {

            TextView newsTitleText;

            public ViewHolder(View view) {
                super(view);
                newsTitleText = (TextView) view.findViewById(R.id.news_title);
            }
        }

        public NewsAdapter(List<News> newsList) {
            mNewsList = newsList;
        }

        @Override
        public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.news_item, parent, false);
            final ViewHolder holder = new ViewHolder(view);
            view.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    News news = mNewsList.get(holder.getAdapterPosition());
                    if (isTwoPane) {
                        // 如果是双页模式，则刷新 NewsContentFragment 中的内容
                        NewsContentFragment newsContentFragment =
                            (NewsContentFragment) getFragmentManager()
                                .findFragmentById(R.id.news_content_fragment);
                        newsContentFragment.refresh(news.getTitle(),
                            news.getContent());
                    } else {
                        // 如果是单页模式，则直接启动 NewsContentActivity
                        NewsContentActivity.actionStart(getActivity(),
                            news.getTitle(), news.getContent());
                    }
                }
            });
            return holder;
        }
    }
}
```

```

@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    News news = mNewsList.get(position);
    holder.newsTitleText.setText(news.getTitle());
}

@Override
public int getItemCount() {
    return mNewsList.size();
}

}

```

RecyclerView 的用法你已经相当熟练了，因此这个适配器的代码对你来说应该没有什么难度吧？需要注意的是，之前我们都是将适配器写成一个独立的类，其实也是可以写成内部类的，这里写成内部类的好处就是可以直接访问 NewsTitleFragment 的变量，比如 isTwoPane。

观察一下 onCreateViewHolder()方法中注册的点击事件，首先获取到了点击项的 News 实例，然后通过 isTwoPane 变量来判断当前是单页还是双页模式，如果是单页模式，就启动一个新的活动去显示新闻内容，如果是双页模式，就更新新闻内容碎片里的数据。

现在还剩最后一步收尾工作，就是向 RecyclerView 中填充数据了。修改 NewsTitleFragment 中的代码，如下所示：

```

public class NewsTitleFragment extends Fragment {

    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.news_title_frag, container, false);
        RecyclerView newsTitleRecyclerView = (RecyclerView) view.findViewById
            (R.id.news_title_recycler_view);
        LinearLayoutManager layoutManager = new LinearLayoutManager(getActivity());
        newsTitleRecyclerView.setLayoutManager(layoutManager);
        NewsAdapter adapter = new NewsAdapter(getNews());
        newsTitleRecyclerView.setAdapter(adapter);
        return view;
    }

    private List<News> getNews() {
        List<News> newsList = new ArrayList<>();
        for (int i = 1; i <= 50; i++) {
            News news = new News();
            news.setTitle("This is news title " + i);
            news.setContent(getRandomLengthContent("This is news content " + i + "."));
            newsList.add(news);
        }
    }
}

```

```

        return newsList;
    }

    private String getRandomLengthContent(String content) {
        Random random = new Random();
        int length = random.nextInt(20) + 1;
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < length; i++) {
            builder.append(content);
        }
        return builder.toString();
    }

    ...
}

}

```

可以看到，`onCreateView()`方法中添加了`RecyclerView`标准的使用方法，在碎片中使用`RecyclerView`和在活动中使用几乎是一模一样的，相信没有什么需要解释的。另外，这里调用了`getNews()`方法来初始化50条模拟新闻数据，同样使用了一个`getRandomLengthContent()`方法来随机生成新闻内容的长度，以保证每条新闻的内容差距比较大，相信你对这个方法肯定不会陌生了。

这样我们所有的编写工作就已经完成了，赶快来运行一下吧！首先在手机模拟器上运行，效果如图4.14所示。

可以看到许多条新闻的标题，然后点击第一条新闻，会启动一个新的活动来显示新闻的内容，效果如图4.15所示。



图4.14 单页模式的新闻列表界面

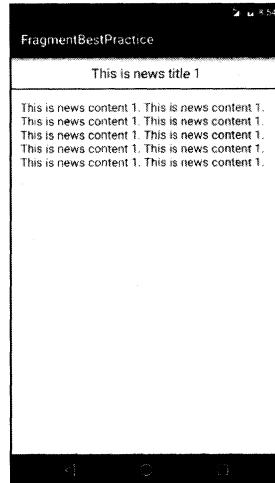


图4.15 单页模式的新闻内容界面

接下来将程序在平板模拟器上运行，同样点击第一条新闻，效果如图4.16所示。



图 4.16 双页模式的新闻标题和内容界面

怎么样？同样的一份代码，在手机和平板上运行却分别是两种完全不同的效果，说明我们程序的兼容性已经相当不错了。通过这个例子，我相信你对碎片的理解一定又加深了很多，现在就让我们一起来总结一下吧。

4.6 小结与点评

你应该可以感觉到，上一节中我们开发的新闻应用，代码复杂度还是有点高的，比起只需要兼容一个终端的应用，我们要考虑的东西多了很多。不过在开发的过程中多付出一些，在以后的代码维护中就可以轻松很多。因此，有时候提前的付出还是很值得的。

我们再来看看本章所学的内容吧，首先你了解了碎片的基本概念以及使用场景，接着通过几个实例掌握了碎片的常见用法，随后又学习了碎片生命周期的相关内容以及动态加载布局的技巧，最后在本章的最佳实践部分将前面所学的内容综合运用了一遍，相信你已经将碎片相关知识点都牢记在心，并可以较为熟练地应用了。

本章其实是具有一个里程碑式的纪念意义的，因为到这里为止，我们已经基本将 Android UI 相关的重要知识点都讲完了。后面在很长一段时间内都不会再系统性地介绍 UI 方面的知识，而是将结合前面所学的 UI 知识来更好地讲解相应章节的内容。那么我们下一章将要学习什么呢？还记得在第 1 章里介绍过的 Android 四大组件吧？目前我们只掌握了活动这一个组件，那么下一章就来学习广播接收器吧。跟上脚步，准备继续前进！

第 5 章

全局大喇叭——详解广播机制

记得在我上学的时候，每个班级的教室里都会装有一个喇叭，这些喇叭都是接入到学校的广播室的，一旦有什么重要的通知，就会播放一条广播来告知全校的师生。类似的工作机制其实在计算机领域也有很广泛的应用，如果你了解网络通信原理应该会知道，在一个 IP 网络范围中，最大的 IP 地址是被保留作为广播地址来使用的。比如某个网络的 IP 范围是 192.168.0.XXX，子网掩码是 255.255.255.0，那么这个网络的广播地址就是 192.168.0.255。广播数据包会被发送到同一网络上的所有端口，这样在该网络中的每台主机都将会收到这条广播。

为了便于进行系统级别的消息通知，Android 也引入了一套类似的广播消息机制。相比于我前面举出的两个例子，Android 中的广播机制会显得更加灵活，本章就将对这一机制的方方面面进行详细的讲解。

5.1 广播机制简介

为什么说 Android 中的广播机制更加灵活呢？这是因为 Android 中的每个应用程序都可以对自己感兴趣的广播进行注册，这样该程序就只会接收到自己所关心的广播内容，这些广播可能是来自于系统的，也可能是来自于其他应用程序的。Android 提供了一套完整的 API，允许应用程序自由地发送和接收广播。发送广播的方法其实之前稍微提到过，如果你记性好的话可能还会有印象，就是借助我们第 2 章学过的 Intent。而接收广播的方法则需要引入一个新的概念——广播接收器（Broadcast Receiver）。

广播接收器的具体用法将会在下一节中做介绍，这里我们先来了解一下广播的类型。Android 中的广播主要可以分为两种类型：标准广播和有序广播。

- **标准广播**（Normal broadcasts）是一种完全异步执行的广播，在广播发出之后，所有的广播接收器几乎都会在同一时刻接收到这条广播消息，因此它们之间没有任何先后顺序可言。这种广播的效率会比较高，但同时也意味着它是无法被截断的。标准广播的工作流程如图 5.1 所示。



图 5.1 标准广播工作示意图

□ **有序广播 (Ordered broadcasts)** 则是一种同步执行的广播，在广播发出之后，同一时刻只会有一个广播接收器能够收到这条广播消息，当这个广播接收器中的逻辑执行完毕后，广播才会继续传递。所以此时的广播接收器是有先后顺序的，优先级高的广播接收器就可以先收到广播消息，并且前面的广播接收器还可以截断正在传递的广播，这样后面的广播接收器就无法收到广播消息了。有序广播的工作流程如图 5.2 所示。

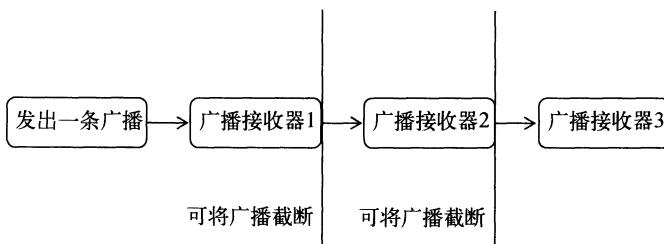


图 5.2 有序广播工作示意图

掌握了这些基本概念后，我们就可以来尝试一下广播的用法了，首先就从接收系统广播开始吧。

5.2 接收系统广播

Android 内置了很多系统级别的广播，我们可以在应用程序中通过监听这些广播来得到各种系统的状态信息。比如手机开机完成后会发出一条广播，电池的电量发生变化会发出一条广播，时间或时区发生改变也会发出一条广播，等等。如果想要接收到这些广播，就需要使用广播接收器，下面我们就来看一下它的具体用法。

5.2.1 动态注册监听网络变化

广播接收器可以自由地对自己感兴趣的广播进行注册，这样当有相应的广播发出时，广播接收器就能够收到该广播，并在内部处理相应的逻辑。注册广播的方式一般有两种，在代码中注册和在 `AndroidManifest.xml` 中注册，其中前者也被称为动态注册，后者也被称为静态注册。

那么该如何创建一个广播接收器呢？其实只需要新建一个类，让它继承自 `BroadcastReceiver`，并重写父类的 `onReceive()` 方法就行了。这样当有广播到来时，`onReceive()` 方法

就会得到执行，具体的逻辑就可以在这个方法中处理。

那我们就先通过动态注册的方式编写一个能够监听网络变化的程序，借此学习一下广播接收器的基本用法吧。新建一个 BroadcastTest 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private IntentFilter intentFilter;

    private NetworkChangeReceiver networkChangeReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        intentFilter = new IntentFilter();
        intentFilter.addAction("android.net.conn.CONNECTIVITY_CHANGE");
        networkChangeReceiver = new NetworkChangeReceiver();
        registerReceiver(networkChangeReceiver, intentFilter);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(networkChangeReceiver);
    }

    class NetworkChangeReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(context, "network changes", Toast.LENGTH_SHORT).show();
        }
    }
}
```

可以看到，我们在 MainActivity 中定义了一个内部类 NetworkChangeReceiver，这个类是继承自 BroadcastReceiver 的，并重写了父类的 onReceive()方法。这样每当网络状态发生变化时，onReceive()方法就会得到执行，这里只是简单地使用 Toast 提示了一段文本信息。

然后观察 onCreate()方法，首先我们创建了一个 IntentFilter 的实例，并给它添加了一个值为 android.net.conn.CONNECTIVITY_CHANGE 的 action，为什么要添加这个值呢？因为当网络状态发生变化时，系统发出的正是一条值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就是说我们的广播接收器想要监听什么广播，就在这里添加相应的 action。接下来创建了一个 NetworkChangeReceiver 的实例，然后调用 registerReceiver()方法进行注册，将 NetworkChangeReceiver 的实例和 IntentFilter 的实例都传了进去，这样 NetworkChangeReceiver 就会收到所有值为 android.net.conn.CONNECTIVITY_CHANGE 的广播，也就实现了

监听网络变化的功能。

最后要记得，动态注册的广播接收器一定都要取消注册才行，这里我们是在 `onDestroy()` 方法中通过调用 `unregisterReceiver()` 方法来实现的。

整体来说，代码还是非常简单的，现在运行一下程序。首先你会在注册完成的时候收到一条广播，然后按下 Home 键回到主界面（注意不能按 Back 键，否则 `onDestroy()` 方法会执行），接着打开 Settings 程序→Data usage 进入到数据使用详情界面，然后尝试着开关 Cellular data 按钮来启动和禁用网络，你就会看到有 Toast 提醒你网络发生了变化。

不过，只是提醒网络发生了变化还不够人性化，最好是能准确地告诉用户当前是有网络还是没有网络，因此我们还需要对上面的代码进行进一步的优化。修改 `MainActivity` 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {
    ...
    class NetworkChangeReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            ConnectivityManager connectionManager = (ConnectivityManager)
                getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connectionManager.getActiveNetworkInfo();
            if (networkInfo != null && networkInfo.isAvailable()) {
                Toast.makeText(context, "network is available",
                    Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(context, "network is unavailable",
                    Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

在 `onReceive()` 方法中，首先通过 `getSystemService()` 方法得到了 `ConnectivityManager` 的实例，这是一个系统服务类，专门用于管理网络连接的。然后调用它的 `getActiveNetworkInfo()` 方法可以得到 `NetworkInfo` 的实例，接着调用 `NetworkInfo` 的 `isAvailable()` 方法，就可以判断出当前是否有网络了，最后我们还是通过 `Toast` 的方式对用户进行提示。

另外，这里有非常重要的一点需要说明，Android 系统为了保护用户设备的安全和隐私，做了严格的规定：如果程序需要进行一些对用户来说比较敏感的操作，就必须在配置文件中声明权限才可以，否则程序将会直接崩溃。比如这里访问系统的网络状态就是需要声明权限的。打开 `AndroidManifest.xml` 文件，在里面加入如下权限就可以访问系统网络状态了：

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

```

package="com.example.broadcasttest"

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

```

这是你第一次遇到权限的问题,其实Android中有许多操作都是需要声明权限才可以进行的,后面我们还会不断使用新的权限。不过目前这个访问系统网络状态的权限还是比较简单的,只需要在AndroidManifest.xml文件中声明一下就可以了,而Android 6.0系统中引入了更加严格的运行时权限,从而能够更好地保证用户设备的安全和隐私,关于这部分内容我们将在第7章中学习。

现在重新运行程序,然后按下Home键→Settings→Data usage,进入到数据使用详情界面,关闭Cellular data会弹出无网络可用的提示,如图5.3所示。

然后重新打开Cellular data又会弹出网络可用的提示,如图5.4所示。



图5.3 禁用系统网络

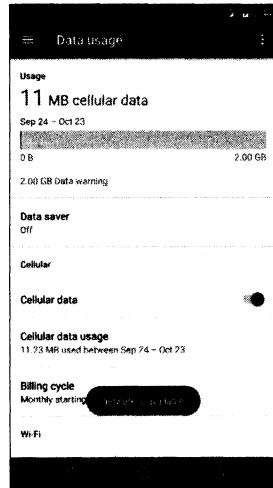


图5.4 启用系统网络

5.2.2 静态注册实现开机启动

动态注册的广播接收器可以自由地控制注册与注销,在灵活性方面有很大的优势,但是它也存在着一个缺点,即必须要在程序启动之后才能接收到广播,因为注册的逻辑是写在onCreate()方法中的。那么有没有什么办法可以让程序在未启动的情况下就能接收到广播呢?这就需要使用静态注册的方式了。

这里我们准备让程序接收一条开机广播,当收到这条广播时就可以在onReceive()方法里执行相应的逻辑,从而实现开机启动的功能。可以使用Android Studio提供的快捷方式来创建一个广播接收器,右击com.example.broadcasttest包→New→Other→Broadcast Receiver,会弹出如

图 5.5 所示的窗口。

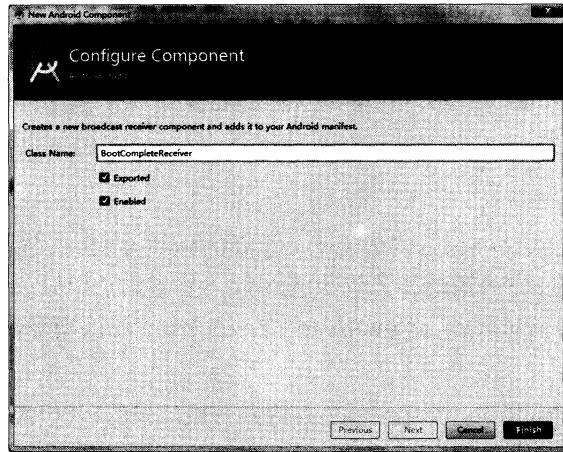


图 5.5 创建广播接收器的窗口

可以看到，这里我们将广播接收器命名为 BootCompleteReceiver， Exported 属性表示是否允许这个广播接收器接收本程序以外的广播，Enabled 属性表示是否启用这个广播接收器。勾选这两个属性，点击 Finish 完成创建。

然后修改 BootCompleteReceiver 中的代码，如下所示：

```
public class BootCompleteReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Boot Complete", Toast.LENGTH_LONG).show();
    }
}
```

代码非常简单，我们只是在 onReceive() 方法中使用 Toast 弹出一段提示信息。

另外，静态的广播接收器一定要在 AndroidManifest.xml 文件中注册才可以使用，不过由于我们是使用 Android Studio 的快捷方式创建的广播接收器，因此注册这一步已经被自动完成了。打开 AndroidManifest.xml 文件瞧一瞧，代码如下所示：

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
```

```

    android:theme="@style/AppTheme">
    ...
<receiver
    android:name=".BootCompleteReceiver"
    android:enabled="true"
    android:exported="true">
</receiver>
</application>

</manifest>

```

可以看到，`<application>`标签内出现了一个新的标签`<receiver>`，所有静态的广播接收器都是在这里进行注册的。它的用法其实和`<activity>`标签非常相似，也是通过`android:name`来指定具体注册哪一个广播接收器，而`enabled`和`exported`属性则是根据我们刚才勾选的状态自动生成的。

不过目前`BootCompleteReceiver`还是不能接收到开机广播的，我们还需要对`AndroidManifest.xml`文件进行修改才行，如下所示：

```

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

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".BootCompleteReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>

</manifest>

```

由于Android系统启动完成后会发出一条值为`android.intent.action.BOOT_COMPLETED`的广播，因此我们在`<intent-filter>`标签里添加了相应的action。另外，监听系统开机广播也是需要声明权限的，可以看到，我们使用`<uses-permission>`标签又加入了一条`android.permission.RECEIVE_BOOT_COMPLETED`权限。

现在重新运行程序后，我们的程序就已经可以接收开机广播了。将模拟器关闭并重新启动，在启动完成之后就会收到开机广播，如图5.6所示。



图 5.6 接收系统开机广播

到目前为止，我们在广播接收器的 `onReceive()` 方法中都只是简单地使用 `Toast` 提示了一段文本信息，当你真正在项目中使用到它的时候，就可以在里面编写自己的逻辑。需要注意的是，不要在 `onReceive()` 方法中添加过多的逻辑或者进行任何的耗时操作，因为在广播接收器中是不允许开启线程的，当 `onReceive()` 方法运行了较长时间而没有结束时，程序就会报错。因此广播接收器更多的是扮演一种打开程序其他组件的角色，比如创建一条状态栏通知，或者启动一个服务等，这几个概念我们会在后面的章节中学到。

5.3 发送自定义广播

现在你已经学会了通过广播接收器来接收系统广播，接下来我们就要学习一下如何在应用程序中发送自定义的广播。前面已经介绍过了，广播主要分为两种类型：标准广播和有序广播，在本节中我们就将通过实践的方式来看一下这两种广播具体的区别。

5.3.1 发送标准广播

在发送广播之前，我们还是需要先定义一个广播接收器来准备接收此广播才行，不然发出去也是白发。因此新建一个 `MyBroadcastReceiver`，代码如下所示：

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceiver", Toast.LENGTH_SHORT).show();
    }
}
```

这里当 MyBroadcastReceiver 收到自定义的广播时，就会弹出“received in MyBroadcastReceiver”的提示。然后在 AndroidManifest.xml 中对这个广播接收器进行修改：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest">
    ...
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".MyBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

可以看到，这里让 MyBroadcastReceiver 接收一条值为 com.example.broadcasttest.MY_BROADCAST 的广播，因此待会儿在发送广播的时候，我们就需要发出这样的一条广播。

接下来修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send Broadcast"
        />

</LinearLayout>
```

这里在布局文件中定义了一个按钮，用于作为发送广播的触发点。然后修改 MainActivity 中的代码，如下所示：

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

```
Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new
        Intent("com.example.broadcasttest.MY_BROADCAST");
        sendBroadcast(intent);
    }
});
...
}

}
```

可以看到，我们在按钮的点击事件里面加入了发送自定义广播的逻辑。首先构建出了一个 Intent 对象，并把要发送的广播的值传入，然后调用了 Context 的 sendBroadcast()方法将广播发送出去，这样所有监听 com.example.broadcasttest.MY_BROADCAST 这条广播的广播接收器就会收到消息。此时发出去的广播就是一条标准广播。

重新运行程序，并点击一下 Send Broadcast 按钮，效果如图 5.7 所示。

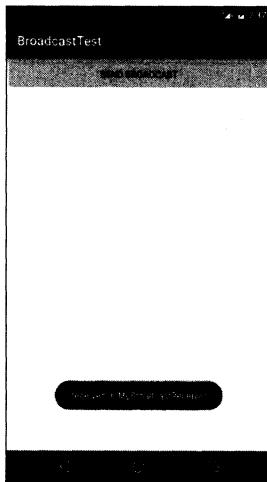


图 5.7 接收到自定义广播

这样我们就成功完成了发送自定义广播的功能。另外，由于广播是使用 Intent 进行传递的，因此你还可以在 Intent 中携带一些数据传递给广播接收器。

5.3.2 发送有序广播

广播是一种可以跨进程的通信方式，这一点从前面接收系统广播的时候就可以看出来了。因此在我们应用程序内发出的广播，其他的应用程序应该也是可以收到的。为了验证这一点，我们

需要再新建一个 BroadcastTest2 项目，点击 Android Studio 导航栏→File→New→New Project 进行创建。

将项目创建好之后，还需要在这个项目下定义一个广播接收器，用于接收上一小节中的自定义广播。新建 AnotherBroadcastReceiver，代码如下所示：

```
public class AnotherBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in AnotherBroadcastReceiver",
                      Toast.LENGTH_SHORT).show();
    }
}
```

这里仍然是在广播接收器的 `onReceive()` 方法中弹出了一段文本信息。然后在 `AndroidManifest.xml` 中对这个广播接收器进行修改，代码如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".AnotherBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="com.example.broadcasttest.MY_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

可以看到，`AnotherBroadcastReceiver` 同样接收的是 `com.example.broadcasttest.MY_BROADCAST` 这条广播。现在运行 `BroadcastTest2` 项目将这个程序安装到模拟器上，然后重新回到 `BroadcastTest` 项目的主界面，并点击一下 `Send Broadcast` 按钮，就会分别弹出两次提示信息，如图 5.8 所示。



图 5.8 两个程序中都接收到自定义广播

这样就强有力地证明了，我们的应用程序发出的广播是可以被其他的应用程序接收到的。

不过到目前为止，程序里发出的都还是标准广播，现在我们来尝试一下发送有序广播。重新回到 BroadcastTest 项目，然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new
                    Intent("com.example.broadcasttest.MY_BROADCAST");
                sendOrderedBroadcast(intent, null);
            }
        });
        ...
    }
}
```

可以看到，发送有序广播只需要改动一行代码，即将 `sendBroadcast()`方法改成 `sendOrderedBroadcast()`方法。`sendOrderedBroadcast()`方法接收两个参数，第一个参数仍然是

`Intent`, 第二个参数是一个与权限相关的字符串, 这里传入 `null` 就行了。现在重新运行程序, 并点击 Send Broadcast 按钮, 你会发现, 两个应用程序仍然都可以接收到这条广播。

看上去好像和标准广播没什么区别嘛, 不过别忘了, 这个时候的广播接收器是有先后顺序的, 而且前面的广播接收器还可以将广播截断, 以阻止其继续传播。

那么该如何设定广播接收器的先后顺序呢? 当然是在注册的时候进行设定的了, 修改 `AndroidManifest.xml` 中的代码, 如下所示:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.broadcasttest2">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
        <receiver
            android:name=".AnotherBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter android:priority="100">
                <action android:name="com.example.broadcasttest.MY_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```

可以看到, 我们通过 `android:priority` 属性给广播接收器设置了优先级, 优先级比较高的广播接收器就可以先收到广播。这里将 `MyBroadcastReceiver` 的优先级设成了 100, 以保证它一定会在 `AnotherBroadcastReceiver` 之前收到广播。

既然已经获得了接收广播的优先权, 那么 `MyBroadcastReceiver` 就可以选择是否允许广播继续传递了。修改 `MyBroadcastReceiver` 中的代码, 如下所示:

```
public class MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received in MyBroadcastReceiver",
                      Toast.LENGTH_SHORT).show();
        abortBroadcast();
    }
}
```

如果在 `onReceive()` 方法中调用了 `abortBroadcast()` 方法, 就表示将这条广播截断, 后面

的广播接收器将无法再接收到这条广播。现在重新运行程序，并点击一下 Send Broadcast 按钮，你会发现，只有 MyBroadcastReceiver 中的 Toast 信息能够弹出，说明这条广播经过 MyBroadcastReceiver 之后确实是终止传递了。

5.4 使用本地广播

前面我们发送和接收的广播全部属于系统全局广播，即发出的广播可以被其他任何应用程序接收到，并且我们也可以接收来自于其他任何应用程序的广播。这样就很容易引起安全性的问题，比如说我们发送的一些携带关键性数据的广播有可能被其他的应用程序截获，或者其他的应用程序不停地向我们的广播接收器里发送各种垃圾广播。

为了能够简单地解决广播的安全性问题，Android 引入了一套本地广播机制，使用这个机制发出的广播只能够在应用程序的内部进行传递，并且广播接收器也只能接收来自本应用程序发出的广播，这样所有的安全性问题就都不存在了。

本地广播的用法并不复杂，主要就是使用了一个 LocalBroadcastManager 来对广播进行管理，并提供了发送广播和注册广播接收器的方法。下面我们就通过具体的实例来尝试一下它的用法，修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends AppCompatActivity {

    private IntentFilter intentFilter;
    private LocalReceiver localReceiver;
    private LocalBroadcastManager localBroadcastManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        localBroadcastManager = LocalBroadcastManager.getInstance(this); // 获取实例
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent("com.example.broadcasttest.LOCAL_BROADCAST");
                localBroadcastManager.sendBroadcast(intent); // 发送本地广播
            }
        });
        intentFilter = new IntentFilter();
        intentFilter.addAction("com.example.broadcasttest.LOCAL_BROADCAST");
        localReceiver = new LocalReceiver();
        localBroadcastManager.registerReceiver(localReceiver, intentFilter); // 注册本地广播监听器
    }
}
```

```

@Override
protected void onDestroy() {
    super.onDestroy();
    localBroadcastManager.unregisterReceiver(localReceiver);
}

class LocalReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "received local broadcast", Toast.LENGTH_SHORT).
            show();
    }
}
}

```

有没有感觉这些代码很熟悉？没错，其实这基本上就和我们前面所学的动态注册广播接收器以及发送广播的代码是一样的。只不过现在首先是通过 LocalBroadcastManager 的 getInstance() 方法得到了它的一个实例，然后在注册广播接收器的时候调用的是 LocalBroadcastManager 的 registerReceiver() 方法，在发送广播的时候调用的是 LocalBroadcastManager 的 sendBroadcast() 方法，仅此而已。这里我们在按钮的点击事件里面发出了一条 com.example.broadcasttest.LOCAL_BROADCAST 广播，然后在 LocalReceiver 里去接收这条广播。重新运行程序，并点击 Send Broadcast 按钮，效果如图 5.9 所示。

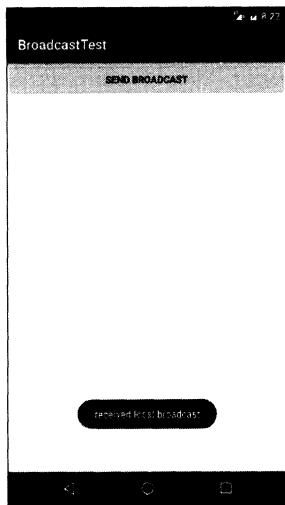


图 5.9 接收到本地广播

可以看到，LocalReceiver 成功接收到了这条本地广播，并通过 Toast 提示了出来。如果你还有兴趣进行实验，可以尝试在 BroadcastTest2 中也去接收 com.example.broadcasttest.

`LOCAL_BROADCAST` 这条广播。答案是显而易见的，肯定无法收到，因为这条广播只会在 `BroadcastTest` 程序内传播。

另外还有一点需要说明，本地广播是无法通过静态注册的方式来接收的。其实这也完全可以理解，因为静态注册主要就是为了让程序在未启动的情况下也能收到广播，而发送本地广播时，我们的程序肯定是已经启动了，因此也完全不需要使用静态注册的功能。

最后我们再来盘点一下使用本地广播的几点优势吧。

- 可以明确地知道正在发送的广播不会离开我们的程序，因此不必担心机密数据泄漏。
- 其他的程序无法将广播发送到我们程序的内部，因此不需要担心会有安全漏洞的隐患。
- 发送本地广播比发送系统全局广播将会更加高效。

5.5 广播的最佳实践——实现强制下线功能

本章的内容不是非常多，因此相信你也一定学得很轻松吧。现在我们就准备通过一个完整例子的实践，来综合运用一下本章中所学到的知识。

强制下线功能应该算是比较常见的了，很多的应用程序都具备这个功能，比如你的 QQ 号在别处登录了，就会将你强制挤下线。其实实现强制下线功能的思路也比较简单，只需要在界面上弹出一个对话框，让用户无法进行任何其他操作，必须要点击对话框中的确定按钮，然后回到登录界面即可。可是这样就存在着一个问题，因为当我们被通知需要强制下线时可能正处于任何一个界面，难道需要在每个界面上都编写一个弹出对话框的逻辑？如果你真的这么想，那思维就偏远了，我们完全可以借助本章中所学的广播知识，来非常轻松地实现这一功能。新建一个 `BroadcastBestPractice` 项目，然后开始动手吧。

强制下线功能需要先关闭掉所有的活动，然后回到登录界面。如果你的反应足够快的话，应该会想到我们在第 2 章的最佳实践部分早就已经实现过关闭所有活动的功能了，因此这里只需要使用同样的方案即可。先创建一个 `ActivityCollector` 类用于管理所有的活动，代码如下所示：

```
public class ActivityCollector {  
  
    public static List<Activity> activities = new ArrayList<>();  
  
    public static void addActivity(Activity activity) {  
        activities.add(activity);  
    }  
  
    public static void removeActivity(Activity activity) {  
        activities.remove(activity);  
    }  
  
    public static void finishAll() {  
        for (Activity activity : activities) {  
            if (!activity.isFinishing()) {  
                activity.finish();  
            }  
        }  
    }  
}
```

```
}
```

然后创建 `BaseActivity` 类作为所有活动的父类，代码如下所示：

```
public class BaseActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ActivityCollector.addActivity(this);  
    }  
  
    @Override  
    protected void onDestroy() {  
        super.onDestroy();  
        ActivityCollector.removeActivity(this);  
    }  
}
```

以上代码都是直接拿之前写好的内容，非常开心。不过从这里开始，就要靠我们自己去动手实现了。首先需要创建一个登录界面的活动，新建 LoginActivity，并让 Android Studio 帮我们自动生成相应的布局文件。然后编辑布局文件 activity_login.xml，代码如下所示：

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

    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="60dp">
        <TextView
            android:layout_width="90dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textSize="18sp"
            android:text="Account:" />

        <EditText
            android:id="@+id/account"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_gravity="center_vertical" />
    </LinearLayout>

    <LinearLayout
        android:orientation="horizontal"
```

```

    android:layout_width="match_parent"
    android:layout_height="60dp">
        <TextView
            android:layout_width="90dp"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:textSize="18sp"
            android:text="Password:" />

        <EditText
            android:id="@+id/password"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:layout_gravity="center_vertical"
            android:inputType="textPassword" />
    </LinearLayout>

    <Button
        android:id="@+id/login"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Login" />
</LinearLayout>

```

这里我们使用 `LinearLayout` 编写出了一个登录布局，最外层是一个纵向的 `LinearLayout`，里面包含了 3 行直接子元素。第一行是一个横向 `LinearLayout`，用于输入账号信息；第二行也是一个横向的 `LinearLayout`，用于输入密码信息；第三行是一个登录按钮。这个布局文件里面用到的全部都是我们之前学过的内容，相信你理解起来应该不会费劲。

接下来修改 `LoginActivity` 中的代码，如下所示：

```

public class LoginActivity extends BaseActivity {

    private EditText accountEdit;
    private EditText passwordEdit;
    private Button login;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        accountEdit = (EditText) findViewById(R.id.account);
        passwordEdit = (EditText) findViewById(R.id.password);
        login = (Button) findViewById(R.id.login);
        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String account = accountEdit.getText().toString();

```

```
        String password = passwordEdit.getText().toString();
        // 如果账号是 admin 且密码是 123456, 就认为登录成功
        if (account.equals("admin") && password.equals("123456")) {
            Intent intent = new Intent(LoginActivity.this, MainActivity.class);
            startActivity(intent);
            finish();
        } else {
            Toast.makeText(LoginActivity.this, "account or password is invalid", Toast.LENGTH_SHORT).show();
        }
    });
}
```

这里我们模拟了一个非常简单的登录功能。首先要将 LoginActivity 的继承结构改成继承自 BaseActivity，然后调用 `findViewById()` 方法分别获取到账号输入框、密码输入框以及登录按钮的实例。接着在登录按钮的点击事件里面对输入的账号和密码进行判断，如果账号是 admin 并且密码是 123456，就认为登录成功并跳转到 MainActivity，否则就提示用户账号或密码错误。

因此，你就可以将 MainActivity 理解成是登录成功后进入的程序主界面了，这里我们并不需要在主界面里提供什么花哨的功能，只需要加入强制下线功能就可以了，修改 activity_main.xml 中的代码，如下所示：

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

    <Button
        android:id="@+id/force_offline"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send force offline broadcast" />

</LinearLayout>
```

非常简单，只有一个按钮而已，用于触发强制下线功能。然后修改 MainActivity 中的代码，如下所示：

```
public class MainActivity extends BaseActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button forceOffline = (Button) findViewById(R.id.force_offline);  
        forceOffline.setOnClickListener(new View.OnClickListener() {  
            @Override
```

```

        public void onClick(View v) {
            Intent intent = new Intent("com.example.broadcastbestpractice.
                FORCE_OFFLINE");
            sendBroadcast(intent);
        }
    });
}

```

同样非常简单，不过这里有个重点，我们在按钮的点击事件里面发送了一条广播，广播的值为 `com.example.broadcastbestpractice.FORCE_OFFLINE`，这条广播就是用于通知程序强制用户下线的。也就是说强制用户下线的逻辑并不是写在 `MainActivity` 里的，而是应该写在接收这条广播的广播接收器里面，这样强制下线的功能就不会依附于任何的界面，不管是在程序的任何地方，只需要发出这样一条广播，就可以完成强制下线的操作了。

那么毫无疑问，接下来我们就需要创建一个广播接收器来接收这条强制下线广播，唯一的问题就是，应该在哪里创建呢？由于广播接收器里面需要弹出一个对话框来阻塞用户的正常操作，但如果创建的是一个静态注册的广播接收器，是没有办法在 `onReceive()` 方法里弹出对话框这样的 UI 控件的，而我们显然也不可能在每个活动中都去注册一个动态的广播接收器。

那么到底应该怎么办呢？答案其实很明显，只需要在 `BaseActivity` 中动态注册一个广播接收器就可以了，因为所有的活动都是继承自 `BaseActivity` 的。

修改 `BaseActivity` 中的代码，如下所示：

```

public class BaseActivity extends AppCompatActivity {

    private ForceOfflineReceiver receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ActivityCollector.addActivity(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction("com.example.broadcastbestpractice.FORCE_OFFLINE");
        receiver = new ForceOfflineReceiver();
        registerReceiver(receiver, intentFilter);
    }

    @Override
    protected void onPause() {
        super.onPause();
        if (receiver != null) {
            unregisterReceiver(receiver);
        }
    }
}

```

```

        receiver = null;
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    ActivityCollector.removeActivity(this);
}

class ForceOfflineReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(final Context context, Intent intent) {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setTitle("Warning");
        builder.setMessage("You are forced to be offline. Please try to login again.");
        builder.setCancelable(false);
        builder.setPositiveButton("OK", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                ActivityCollector.finishAll(); // 销毁所有活动
                Intent intent = new Intent(context, LoginActivity.class);
                context.startActivity(intent); // 重新启动 LoginActivity
            }
        });
        builder.show();
    }
}
}

```

先来看一下 ForceOfflineReceiver 中的代码，这次 `onReceive()`方法里可不再是仅仅弹出一个 `Toast` 了，而是加入了较多的代码，那我们就来仔细地看看吧。首先肯定是使用 `AlertDialog.Builder` 来构建一个对话框，注意这里一定要调用 `setCancelable()`方法将对话框设为不可取消，否则用户按一下 `Back` 键就可以关闭对话框继续使用程序了。然后使用 `setPositiveButton()`方法来给对话框注册确定按钮，当用户点击了确定按钮时，就调用 `ActivityCollector` 的 `finishAll()`方法来销毁掉所有活动，并重新启动 `LoginActivity` 这个活动。

再来看一下我们是怎么注册 `ForceOfflineReceiver` 这个广播接收器的，可以看到，这里重写了 `onResume()` 和 `onPause()` 这两个生命周期函数，然后分别在这两个方法里注册和取消注册了 `ForceOfflineReceiver`。

那么为什么要这样写呢？之前不都是在 `onCreate()` 和 `onDestroy()` 方法里来注册和取消注册广播接收器的么？这是因为我们始终需要保证只有处于栈顶的活动才能接收到这条强制下线广播，非栈顶的活动不应该也没有必要去接收这条广播，所以写在 `onResume()` 和 `onPause()` 方法里就可以很好地解决这个问题，当一个活动失去栈顶位置时就会自动取消广播接收器的注册。