

TURING

需要整本电子书，联系我QQ：2667271557



第一行代码

Android

第2版

郭霖◎著

人民邮电出版社

需要整本电子书，^{北京}联系我QQ：2667271557

需要整本电子书，联系我QQ：2667271557

图书在版编目（CIP）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

需要整本电子书，联系我QQ：2667271557

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，

需要整本电子书，联系我QQ：2667271557

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，内容将前所未有的充实。

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/booksources>。

需要整本电子书，联系我QQ：2667271557

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，你都一直默默的理解和支持我。

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	第5章 全局大喇叭——详解广播机制170
3.4.1 引入布局.....109	5.1 广播机制简介.....170
3.4.2 创建自定义控件.....111	5.2 接收系统广播.....171
3.5 最常用和最难用的控件——ListView.....113	5.2.1 动态注册监听网络变化.....171
3.5.1 ListView的简单用法.....114	5.2.2 静态注册实现开机启动.....174
3.5.2 定制ListView的界面.....115	5.3 发送自定义广播.....177
3.5.3 提升ListView的运行效率.....119	5.3.1 发送标准广播.....177
3.5.4 ListView的点击事件.....120	5.3.2 发送有序广播.....179
3.6 更强大的滚动控件——RecyclerView.....122	5.4 使用本地广播.....183
3.6.1 RecyclerView的基本用法.....122	5.5 广播的最佳实践——实现强制下线功能.....185
3.6.2 实现横向滚动和瀑布流布局.....125	5.6 Git时间——初识版本控制工具.....192
3.6.3 RecyclerView的点击事件.....130	5.6.1 安装Git.....192
3.7 编写界面的最佳实践.....132	5.6.2 创建代码仓库.....193
3.7.1 制作Nine-Patch图片.....132	5.6.3 提交本地代码.....195
3.7.2 编写精美的聊天界面.....135	5.7 小结与点评.....195
3.8 小结与点评.....141	

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

内容提供者	244
7.1 内容提供者简介	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	254
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者	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	281
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	312
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用法	402
9.6 小结与点评	338	11.5.1 分支的用法	403
第 10 章 后台默默的劳动者——探究服务	339	11.5.2 与远程版本库协作	404
10.1 服务是什么	339	11.6 小结与点评	406
10.2 Android 多线程编程	340	第 12 章 最佳的 UI 体验——Material Design 实战	407
10.2.1 线程的基本用法	340	12.1 什么是 Material Design	407
10.2.2 在子线程中更新 UI	341	12.2 Toolbar	408
10.2.3 解析异步消息处理机制	345	12.3 滑动菜单	415
10.2.4 使用 AsyncTask	347	12.3.1 DrawerLayout	415
10.3 服务的基本用法	349	12.3.2 NavigationView	418
10.3.1 定义一个服务	349	12.4 悬浮按钮和可交互提示	423
10.3.2 启动和停止服务	352	12.4.1 FloatingActionButton	424
10.3.3 活动和服务进行通信	355	12.4.2 Snackbar	427
10.4 服务的生命周期	359	12.4.3 CoordinatorLayout	428
10.5 服务的更多技巧	359	12.5 卡片式布局	430
10.5.1 使用前台服务	359	12.5.1 CardView	431
10.5.2 使用 IntentService	361	12.5.2 AppBarLayout	437
10.6 服务的最佳实践——完整版的下载示例	365	12.6 下拉刷新	440
10.7 小结与点评	378	12.7 可折叠式标题栏	443
第 11 章 Android 特色开发——基于位置的服务	379	12.7.1 CollapsingToolbarLayout	443
11.1 基于位置的服务简介	379	12.7.2 充分利用系统状态栏空间	453
11.2 申请 API Key	380	12.8 小结与点评	456
11.3 使用百度定位	384	第 13 章 继续进阶——你还应该掌握的高级技巧	457
11.3.1 准备 LBS SDK	384	13.1 全局获取 Context 的技巧	457
11.3.2 确定自己位置的经纬度	386	13.2 使用 Intent 传递对象	461
11.3.3 选择定位模式	391	13.2.1 Serializable 方式	461
11.3.4 看得懂的位置信息	393	13.2.2 Parcelable 方式	463

13.3	定制自己的日志工具	464	14.5.4	获取必应每日一图	526
13.4	调试 Android 程序	466	14.6	手动更新天气和切换城市	532
13.5	创建定时任务	469	14.6.1	手动更新天气	532
13.5.1	Alarm 机制	469	14.6.2	切换城市	535
13.5.2	Doze 模式	471	14.7	后台自动更新天气	540
13.6	多窗口模式编程	472	14.8	修改图标和名称	542
13.6.1	进入多窗口模式	473	14.9	你还可以做的事情	543
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					
14.1	功能需求及技术可行性分析	486	第 15 章 最后一步——将应用发布到 360 应用商店		
14.2	Git 时间——将代码托管到 GitHub 上	489	15.1	生成正式签名的 APK 文件	545
14.3	创建数据库和表	494	15.1.1	使用 Android Studio 生成	546
14.4	遍历全国省市县数据	499	15.1.2	使用 Gradle 生成	548
14.5	显示天气信息	509	15.1.3	生成多渠道 APK 文件	551
14.5.1	定义 GSON 实体类	509	15.2	申请 360 开发者账号	554
14.5.2	编写天气界面	514	15.3	发布应用程序	556
14.5.3	将天气显示到界面上	520	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。

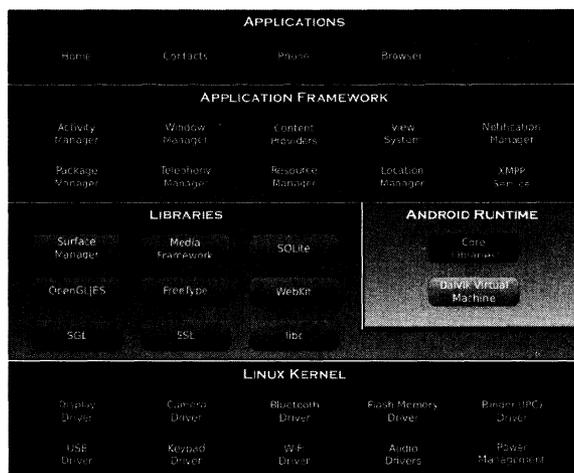


图 1.1 Android 系统架构

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来看一看，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) 和内容提供者 (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。内容提供者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者来实现。

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 手把手带你搭建开发环境

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搭建开发环境搭建起来。

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 1 版仍然选用的 Eclipse 来作用开发工具。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l。）

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选上，如图 1.2 所示。

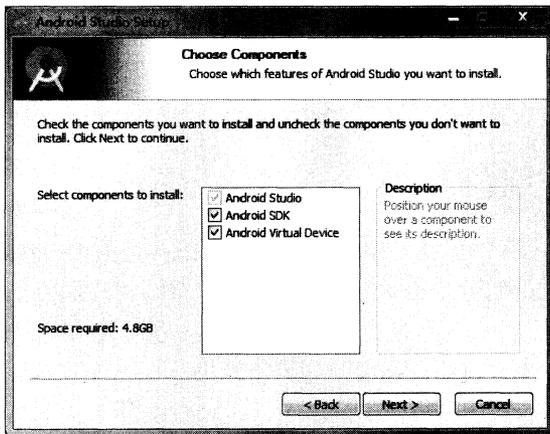


图 1.2 选择安装组件

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。

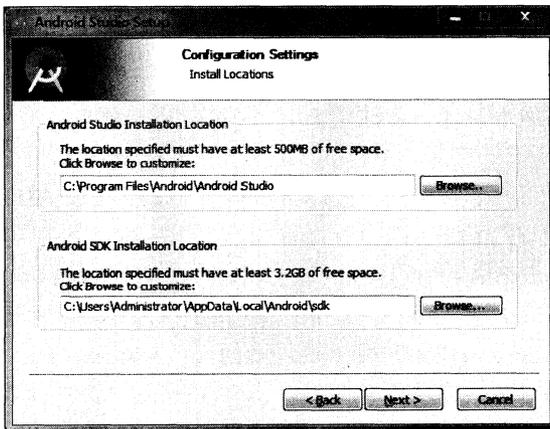


图 1.3 选择安装地址

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。

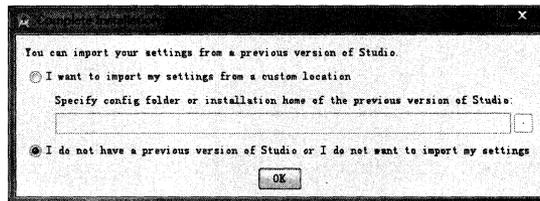


图 1.5 选择不导入配置

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

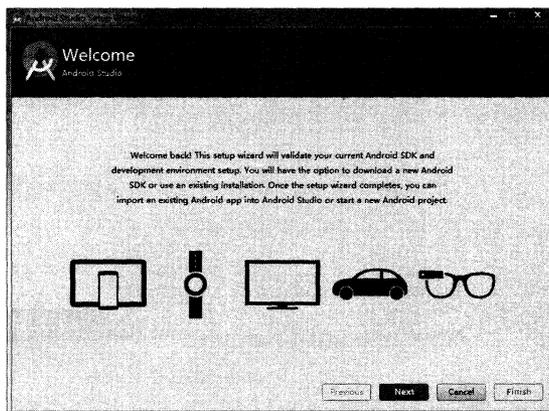


图 1.6 Android Studio 的配置界面

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。

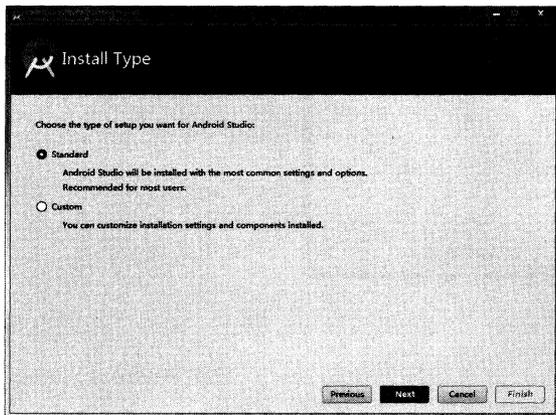


图 1.7 选择安装类型

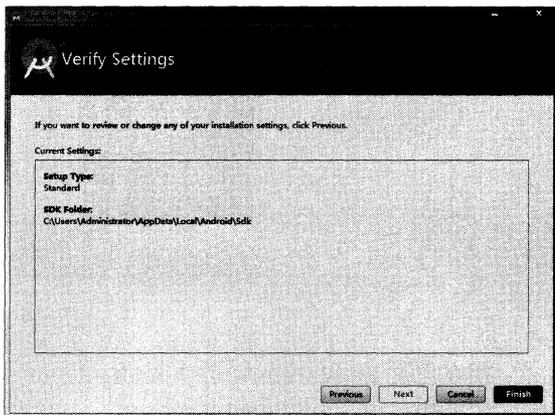


图 1.8 完成 Android Studio 配置

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。

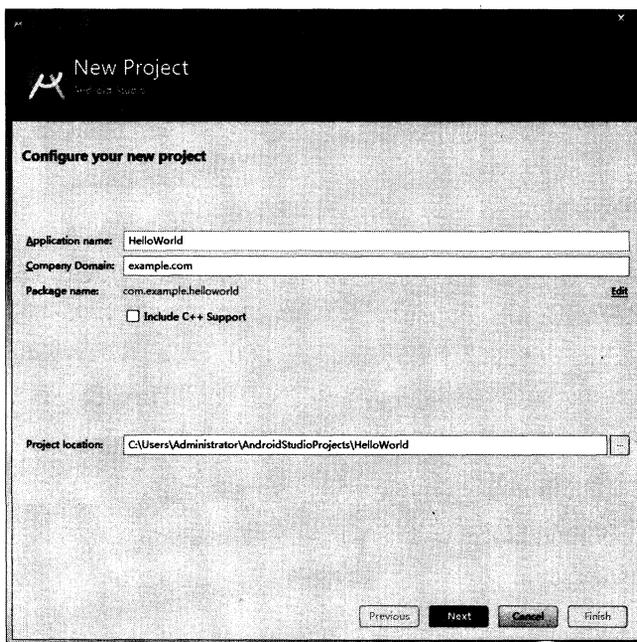


图 1.10 创建新项目

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。

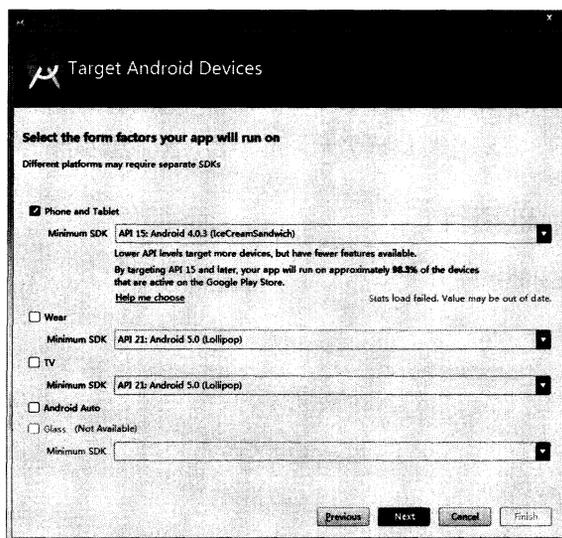


图 1.11 设置项目的最低兼容版本

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。

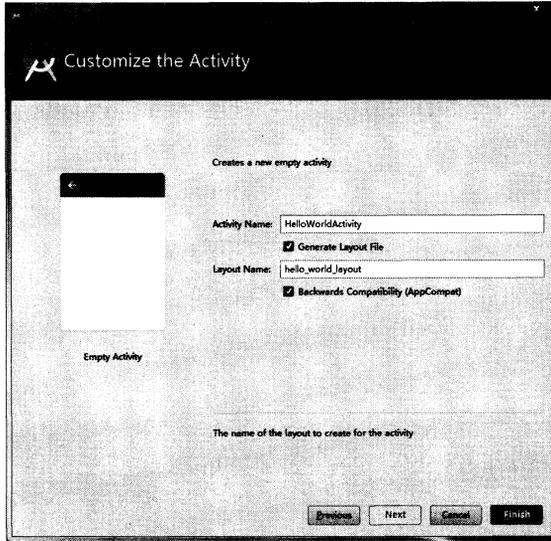


图 1.13 给活动和布局命名

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。

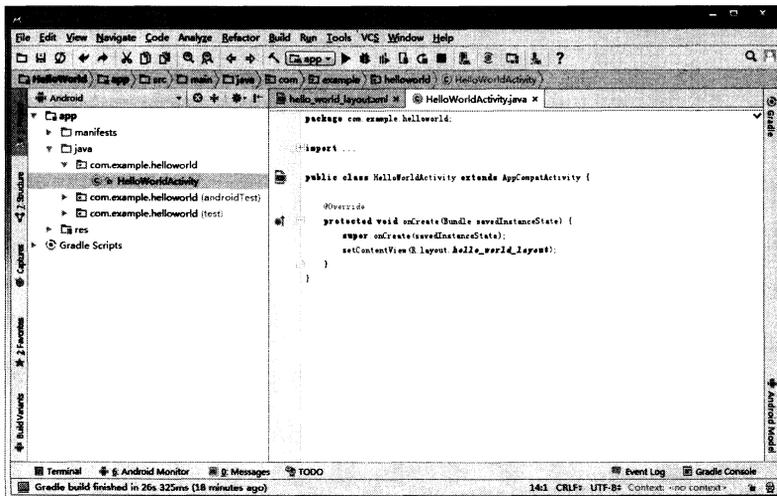


图 1.14 项目创建成功

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。

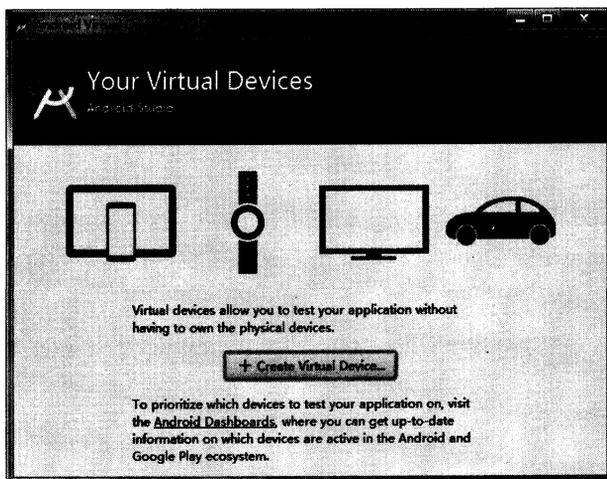


图 1.16 创建模拟器

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。

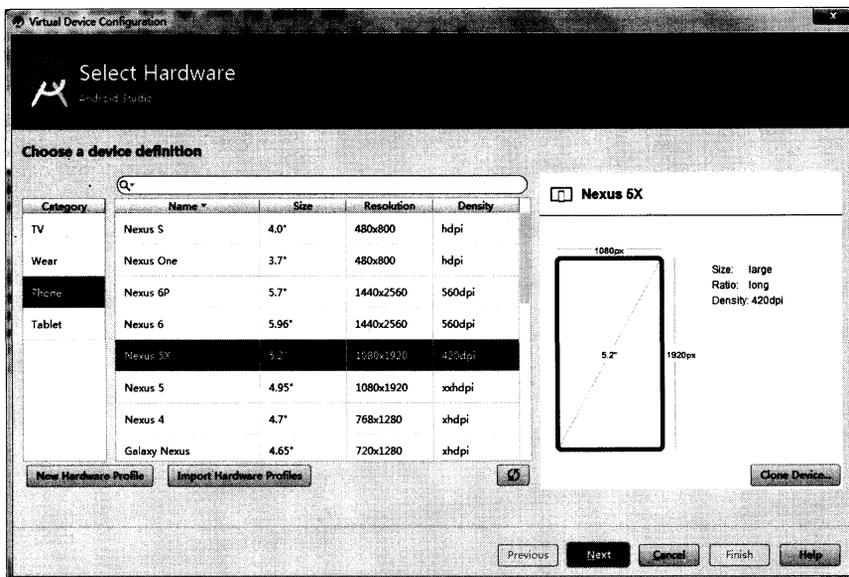


图 1.17 选择要创建的模拟器设备

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。

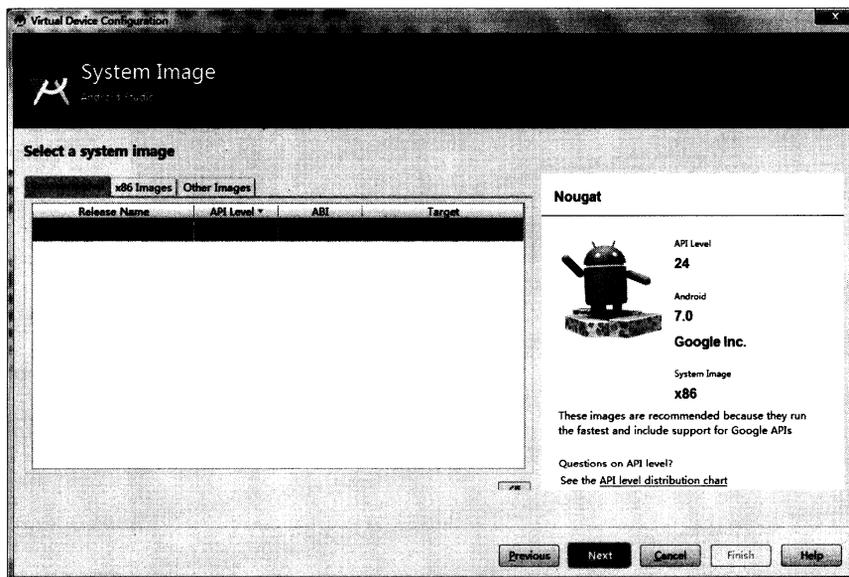


图 1.18 选择模拟器的操作系统版本

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。

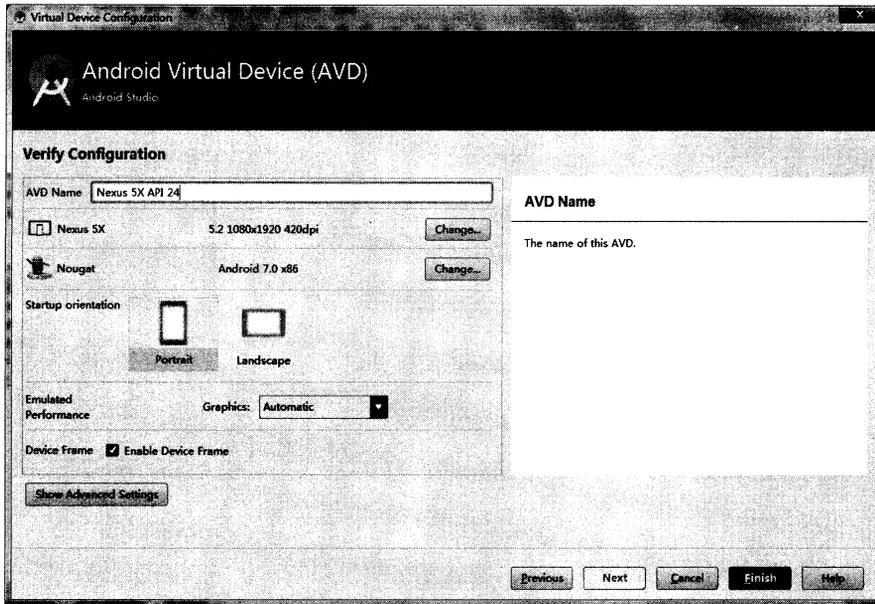


图 1.19 确认模拟器配置

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。

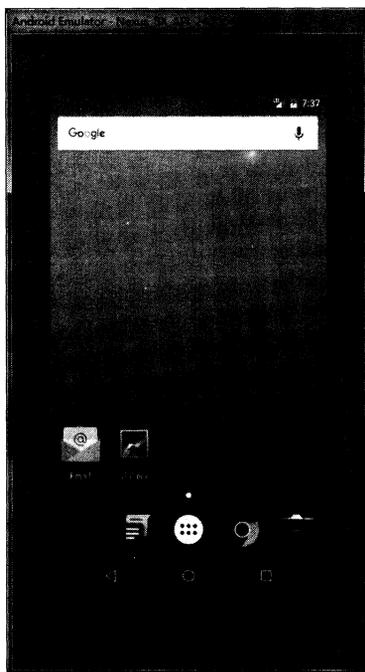


图 1.21 启动后的模拟器界面

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。

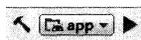


图 1.22 顶部工具栏中的图标

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。

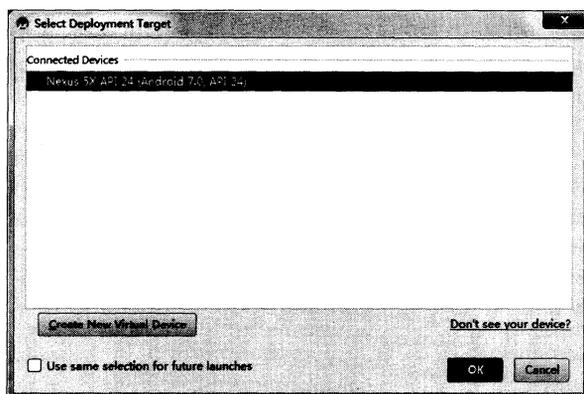


图 1.23 选择运行设备对话框

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。

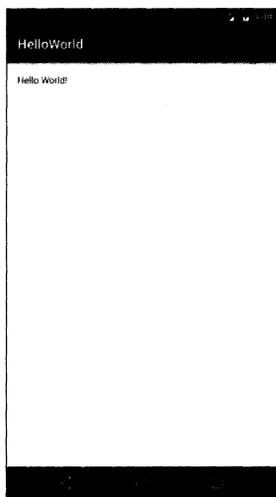


图 1.24 运行 HelloWorld 项目



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。

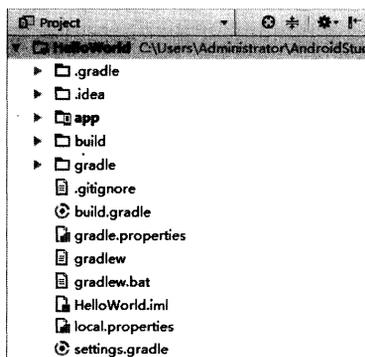


图 1.28 Project 模式的项目结构

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，也不要动手编辑。

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。

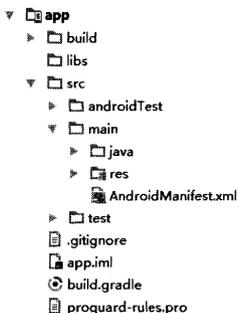


图 1.29 app 目录下的结构

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! 一定就是在这里定义的了！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` 项目的应用图标就是通过 `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，这些数据应该不是你非常想看到的、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。

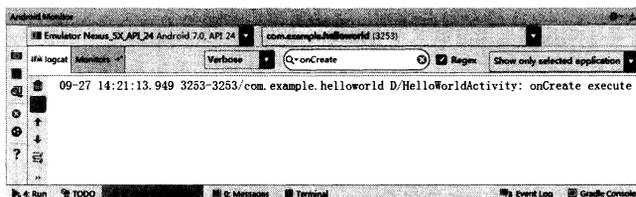


图 1.31 logcat 中的打印信息

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 `syso`，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。

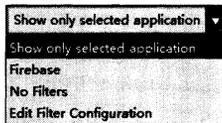


图 1.32 logcat 中的过滤器

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。

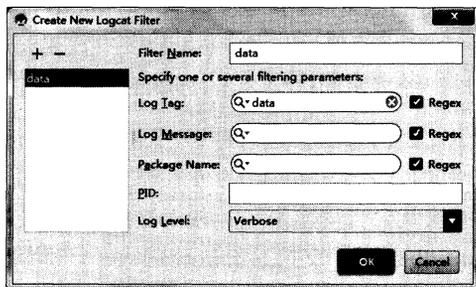


图 1.33 过滤器配置界面

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，你将会在 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。

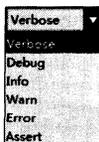


图 1.34 logcat 中的日志级别

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，假如你已经写出了是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，用户根本不会在乎这些，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。

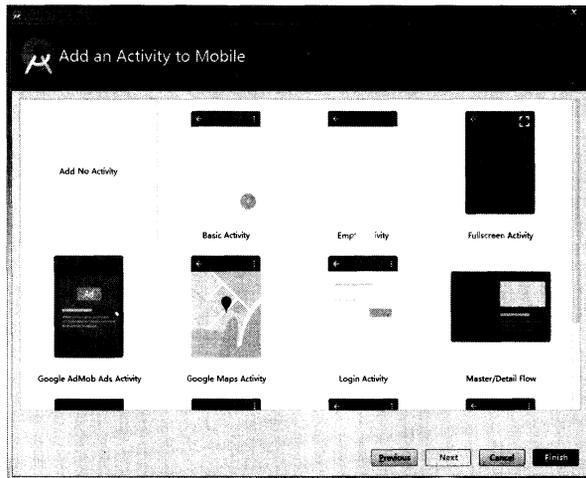


图 2.1 选择不添加活动

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。

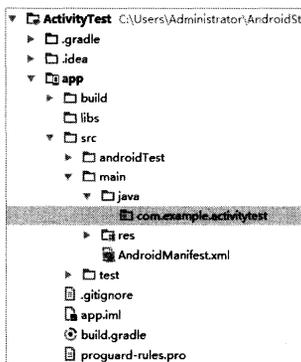


图 2.2 初始项目结构

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。

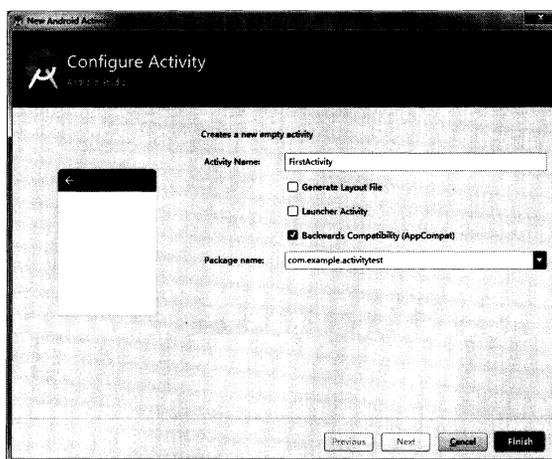


图 2.3 新建活动对话框

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,下面我们将会一个个手动来完成。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。

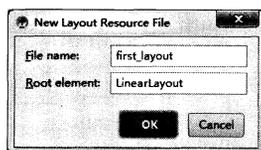


图 2.4 新建布局资源文件

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。

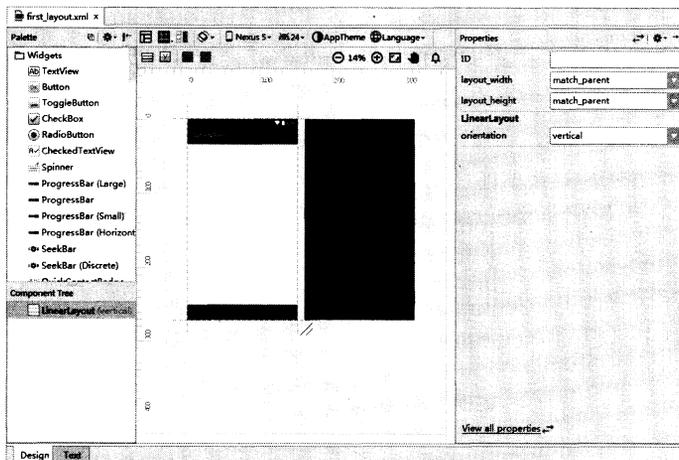


图 2.5 布局编辑器

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。

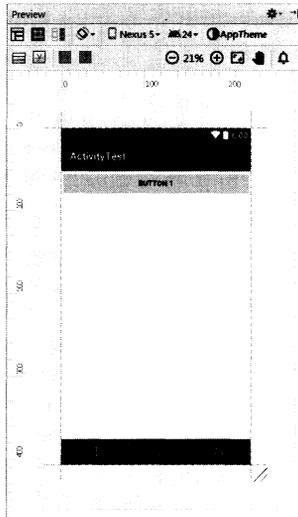


图 2.6 预览当前布局

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 `AndroidManifest.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。

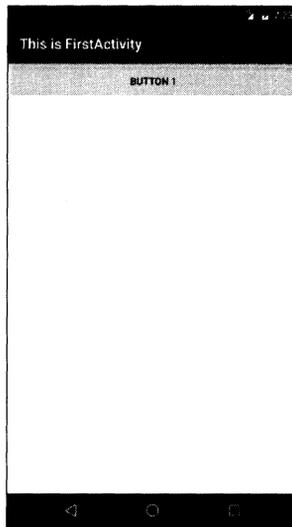


图 2.7 首次运行结果

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些什么事情吧。

2.2.4 在活动中使用 Toast

Toast 是 Android 系统提供了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。

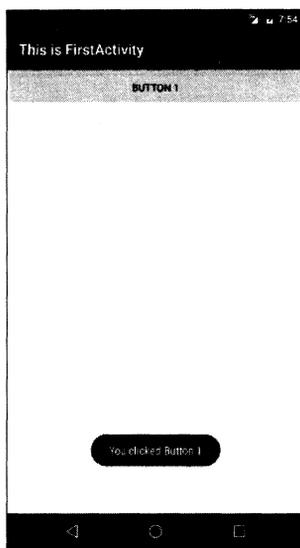


图 2.8 Toast 运行效果

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。

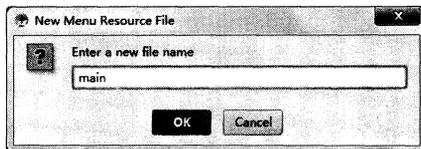


图 2.9 新建 Menu 资源文件

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 MenuInflater 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。

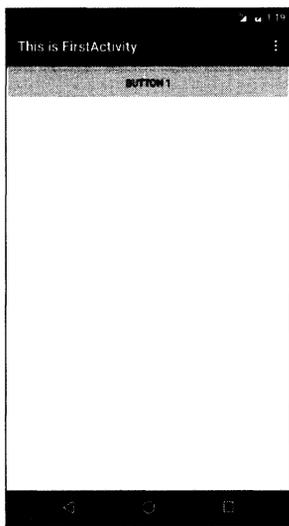


图 2.11 带菜单按钮的活动

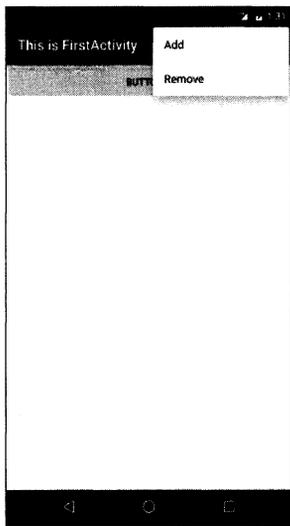


图 2.12 弹出菜单项的界面

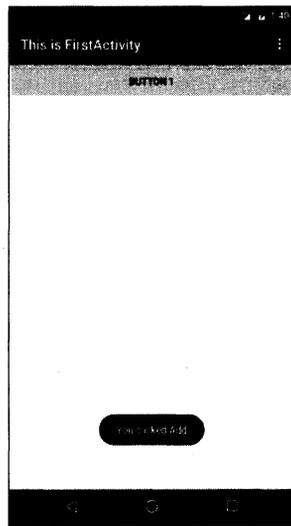


图 2.13 点击了 Add 菜单项

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。

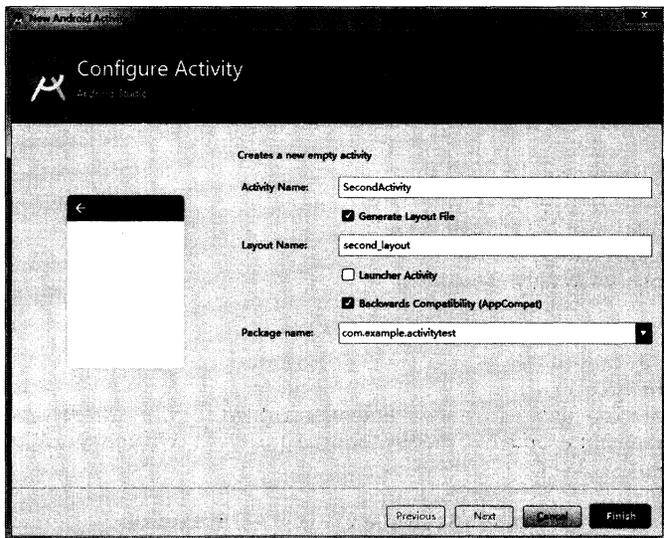


图 2.14 创建 SecondActivity

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 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”就非常明显了，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。

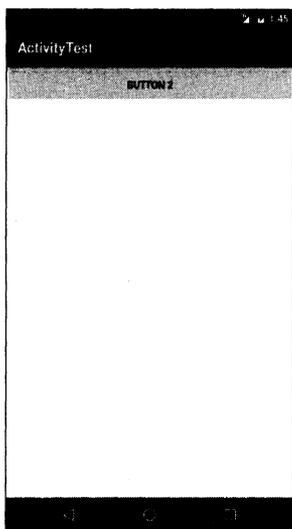


图 2.15 SecondActivity 界面

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。

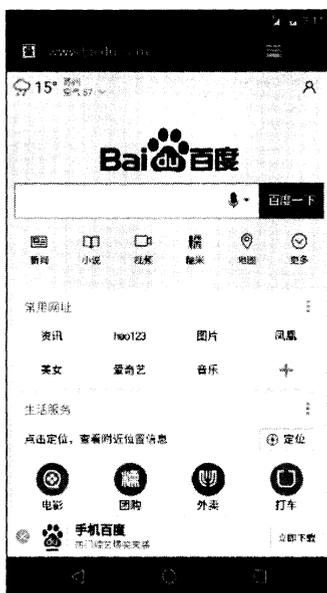


图 2.17 系统浏览器界面

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。

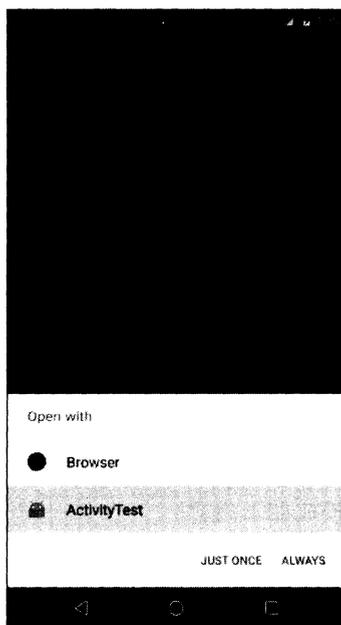


图 2.18 选择响应 Intent 的程序

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。

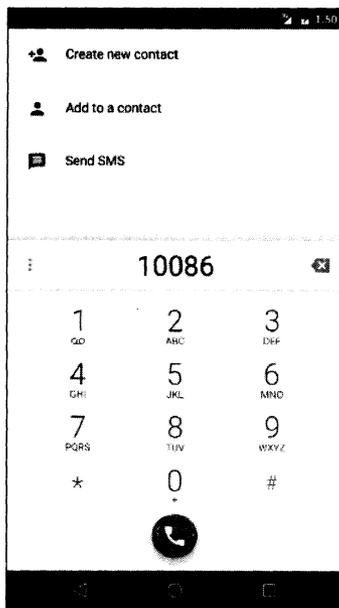


图 2.19 系统拨号界面

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，下面我们来一起看一下。

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，可以把我们想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。

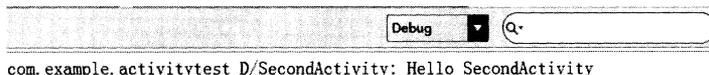


图 2.20 SecondActivity 中的打印信息

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，这时前一个入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。

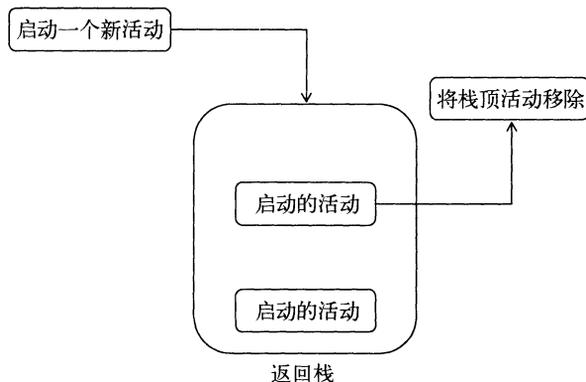


图 2.22 返回栈工作示意图

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着的，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的状态和成员变量，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。

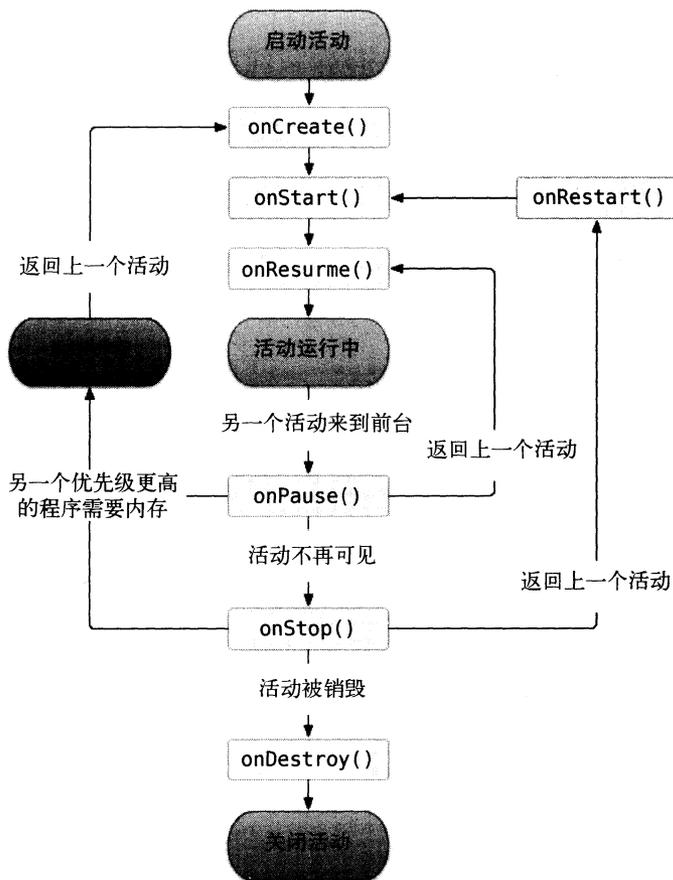


图 2.23 活动的生命周期

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 Dialog-Activity，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。

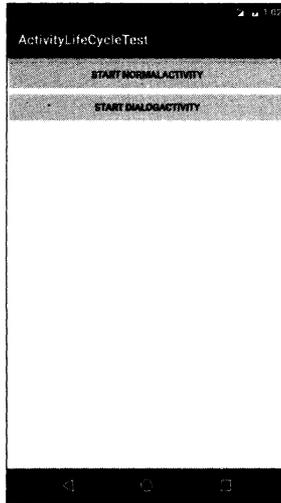


图 2.24 MainActivity 界面

这时观察 logcat 中的打印日志，如图 2.25 所示。

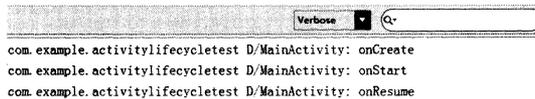


图 2.25 启动程序时的打印日志

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。

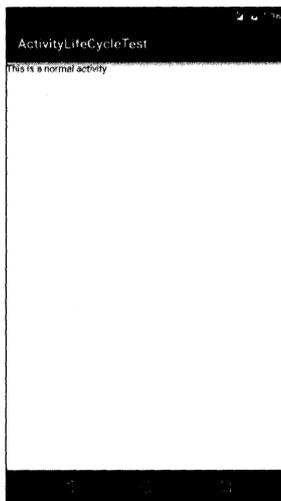
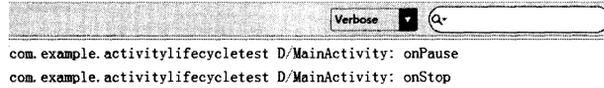


图 2.26 NormalActivity 界面

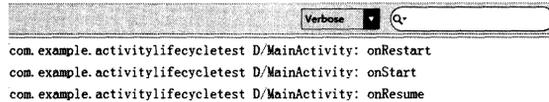
此时的打印信息如图 2.27 所示。



```
com.example.activitylifecyletest D/MainActivity: onPause
com.example.activitylifecyletest D/MainActivity: onStop
```

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。



```
com.example.activitylifecyletest D/MainActivity: onRestart
com.example.activitylifecyletest D/MainActivity: onStart
com.example.activitylifecyletest D/MainActivity: onResume
```

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。



```
com.example.activitylifecyletest D/MainActivity: onPause
```

图 2.30 打开 DialogActivity 时的打印日志

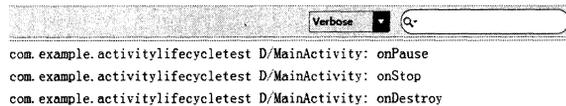
可以看到，只有 `onPause()` 方法得到了执行，`onStop()` 方法并没有执行，这是因为 `DialogActivity` 并没有完全遮挡住 `MainActivity`，此时 `MainActivity` 只是进入了暂停状态，并没有进入停止状态。相应地，按下 `Back` 键返回 `MainActivity` 也应该只有 `onResume()` 方法会得到执行，如图 2.31 所示。



```
com.example.activitylifecycletest D/MainActivity: onResume
```

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。



```
com.example.activitylifecycletest D/MainActivity: onPause
com.example.activitylifecycletest D/MainActivity: onStop
com.example.activitylifecycletest D/MainActivity: onDestroy
```

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。下面我们来逐个进行学习。

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，系统不会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。

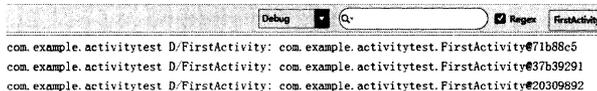


图 2.33 standard 模式下的打印日志

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，因此你需要连接 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。

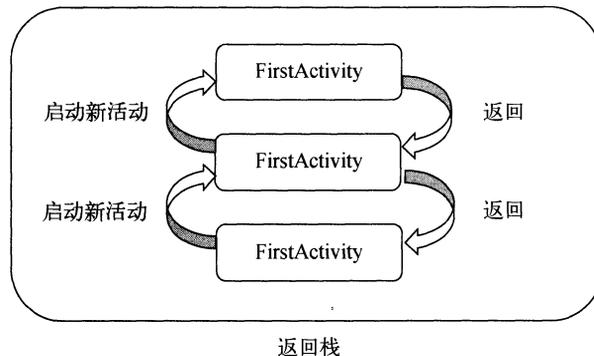


图 2.34 standard 模式示意图

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。

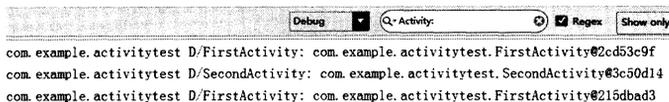


图 2.36 singleTop 模式下的打印日志

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。

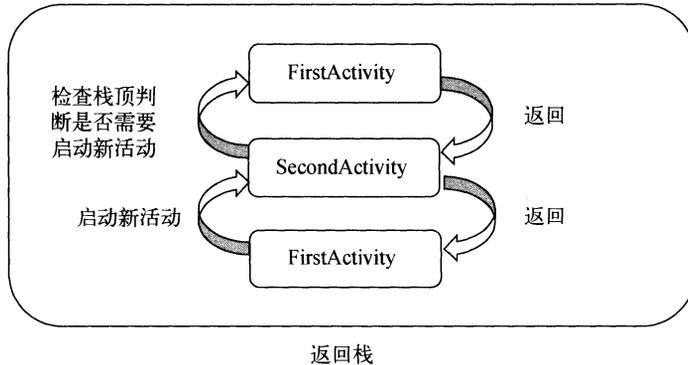


图 2.37 singleTop 模式示意图

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。

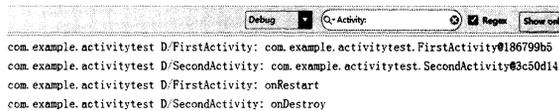


图 2.38 singleTask 模式下的打印日志

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。

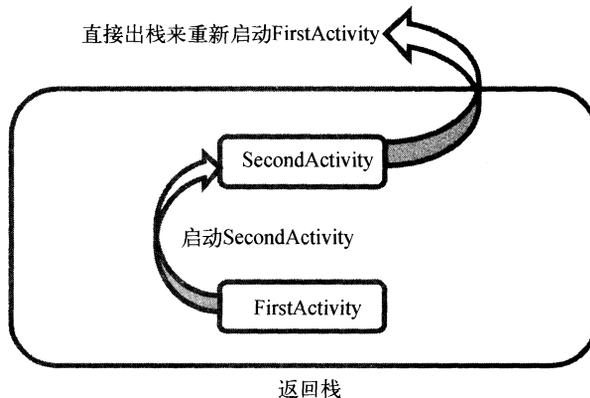


图 2.39 singleTask 模式示意图

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。

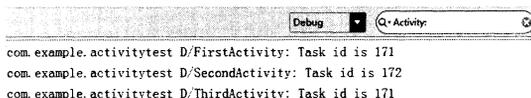


图 2.40 singleInstance 模式下的打印日志

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。

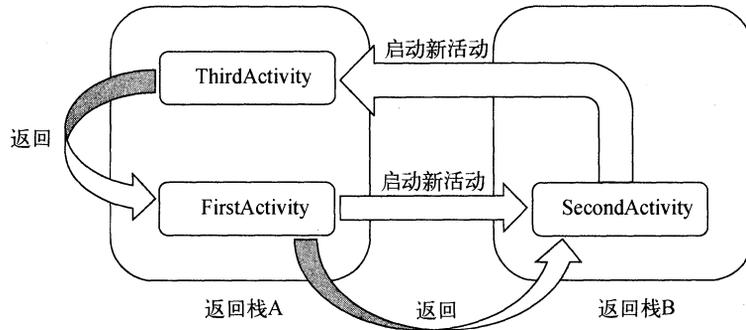


图 2.41 singleInstance 模式示意图

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。

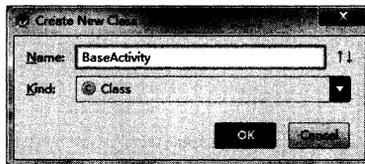


图 2.42 创建 BaseActivity 类

注意这里 BaseActivity 和普通活动的创建方式并不一样，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。

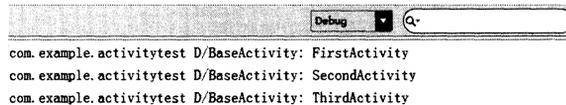


图 2.43 BaseActivity 中的打印日志

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，这样我们就可以时时刻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，需要连接 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，表明将当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東西都涉及了，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。

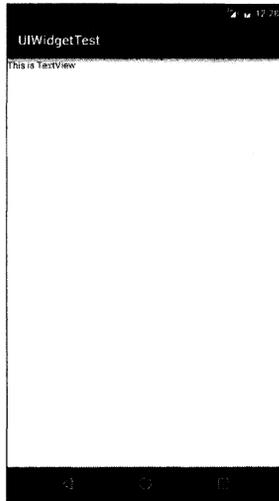


图 3.1 TextView 运行效果

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。

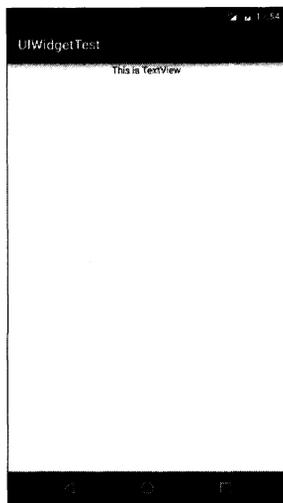


图 3.2 TextView 居中效果

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。

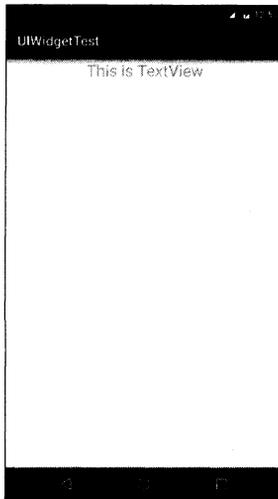


图 3.3 改变 TextView 文字大小和颜色的效果

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法

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。

```
        for (int i = 0; i < allCounties.length(); i++) {
            JSONObject countyObject = allCounties.getJSONObject(i);
            County county = new County();
            county.setCountyName(countyObject.getString("name"));
            county.setWeatherId(countyObject.getString("weather_id"));
            county.setCityId(cityId);
            county.save();
        }
        return true;
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
return false;
}
}
```

可以看到，我们提供了 `handleProvincesResponse()`、`handleCitiesResponse()`、`handleCountiesResponse()` 这 3 个方法，分别用于解析和处理服务器返回的省级、市级和县级数据。处理的方式都是类似的，先使用 `JSONArray` 和 `JSONObject` 将数据解析出来，然后组装成实体类对象，再调用 `save()` 方法将数据存储到数据库当中。由于这里的 JSON 数据结构比较简单，我们就不使用 GSON 来进行解析了。

需要准备的工具类就这么多，现在可以开始写界面了。由于遍历全国省市县的功能我们在后面还会复用，因此就不写在活动里面了，而是写在碎片里面，这样需要复用的时候直接在布局里面引用碎片就可以了。

在 `res/layout` 目录中新建 `choose_area.xml` 布局，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="#fff">

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="?attr/colorPrimary">

        <TextView
            android:id="@+id/title_text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:textColor="#fff"
            android:textSize="20sp"/>

    <Button
```

```
        android:id="@+id/back_button"
        android:layout_width="25dp"
        android:layout_height="25dp"
        android:layout_marginLeft="10dp"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:background="@drawable/ic_back"/>
</RelativeLayout>

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

</LinearLayout>
```

布局文件中的内容并不复杂，我们先是定义了一个头布局来作为标题栏，将布局高度设置为 actionBar 的高度，背景色设置为 colorPrimary。然后在头布局中放置了一个 TextView 用于显示标题内容，放置了一个 Button 用于执行返回操作，注意我已经提前准备好了一张 ic_back.png 图片用于作为按钮的背景图。这里之所以要自己定义标题栏，是因为碎片中最好不要直接使用 ActionBar 或 Toolbar，不然在复用的时候可能会出现一些你不想看到的效果。

接下来在头布局的下面定义了一个 ListView，省市县的数据就将显示在这里。之所以这次使用了 ListView，是因为它会自动给每个子项之间添加一条分隔线，而如果使用 RecyclerView 想实现同样的功能则会比较麻烦，这里我们总是选择最优的实现方案。

接下来也是最关键的一步，我们需要编写用于遍历省市县数据的碎片了。新建 ChooseAreaFragment 继承自 Fragment，代码如下所示：

```
public class ChooseAreaFragment extends Fragment {

    public static final int LEVEL_PROVINCE = 0;

    public static final int LEVEL_CITY = 1;

    public static final int LEVEL_COUNTY = 2;

    private ProgressDialog progressDialog;

    private TextView titleText;

    private Button backButton;

    private ListView listView;

    private ArrayAdapter<String> adapter;

    private List<String> dataList = new ArrayList<>();

    /**
     * 省列表
```

```
    */
    private List<Province> provinceList;

    /**
     * 市列表
     */
    private List<City> cityList;

    /**
     * 县列表
     */
    private List<County> countyList;

    /**
     * 选中的省份
     */
    private Province selectedProvince;

    /**
     * 选中的城市
     */
    private City selectedCity;

    /**
     * 当前选中的级别
     */
    private int currentLevel;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.choose_area, container, false);
        titleText = (TextView) view.findViewById(R.id.title_text);
        backButton = (Button) view.findViewById(R.id.back_button);
        listView = (ListView) view.findViewById(R.id.list_view);
        adapter = new ArrayAdapter<>(getContext(), android.R.layout.simple_list_
            item_1, dataList);
        listView.setAdapter(adapter);
        return view;
    }

    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                long id) {
                if (currentLevel == LEVEL_PROVINCE) {
                    selectedProvince = provinceList.get(position);
                    queryCities();
                } else if (currentLevel == LEVEL_CITY) {
                    selectedCity = cityList.get(position);
                    queryCounties();
                }
            }
        });
    }
}
```

```
        }
    }
});
backButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (currentLevel == LEVEL_COUNTY) {
            queryCities();
        } else if (currentLevel == LEVEL_CITY) {
            queryProvinces();
        }
    }
});
queryProvinces();
}

/**
 * 查询全国所有的省，优先从数据库查询，如果没有查询到再去服务器上查询
 */
private void queryProvinces() {
    titleText.setText("中国");
    backButton.setVisibility(View.GONE);
    provinceList = DataSupport.findAll(Province.class);
    if (provinceList.size() > 0) {
        dataList.clear();
        for (Province province : provinceList) {
            dataList.add(province.getProvinceName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        currentLevel = LEVEL_PROVINCE;
    } else {
        String address = "http://guolin.tech/api/china";
        queryFromServer(address, "province");
    }
}

/**
 * 查询选中省内所有的市，优先从数据库查询，如果没有查询到再去服务器上查询
 */
private void queryCities() {
    titleText.setText(selectedProvince.getProvinceName());
    backButton.setVisibility(View.VISIBLE);
    cityList = DataSupport.where("provinceid = ?", String.valueOf(selectedProvince.getId())).find(City.class);
    if (cityList.size() > 0) {
        dataList.clear();
        for (City city : cityList) {
            dataList.add(city.getCityName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        currentLevel = LEVEL_CITY;
    } else {
        int provinceCode = selectedProvince.getProvinceCode();
```

```
String address = "http://guolin.tech/api/china/" + provinceCode;
queryFromServer(address, "city");
    }
}

/**
 * 查询选中市内所有的县，优先从数据库查询，如果没有查询到再去服务器上查询
 */
private void queryCounties() {
    titleText.setText(selectedCity.getCityName());
    backButton.setVisibility(View.VISIBLE);
    countyList = DataSupport.where("cityid = ?", String.valueOf(selectedCity.getId())).find(County.class);
    if (countyList.size() > 0) {
        dataList.clear();
        for (County county : countyList) {
            dataList.add(county.getCountyName());
        }
        adapter.notifyDataSetChanged();
        listView.setSelection(0);
        currentLevel = LEVEL_COUNTY;
    } else {
        int provinceCode = selectedProvince.getProvinceCode();
        int cityCode = selectedCity.getCityCode();
        String address = "http://guolin.tech/api/china/" + provinceCode + "/" + cityCode;
        queryFromServer(address, "county");
    }
}

/**
 * 根据传入的地址和类型从服务器上查询省市县数据
 */
private void queryFromServer(String address, final String type) {
    showProgressDialog();
    HttpUtil.sendOkHttpRequest(address, new Callback() {
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            String responseText = response.body().string();
            boolean result = false;
            if ("province".equals(type)) {
                result = Utility.handleProvinceResponse(responseText);
            } else if ("city".equals(type)) {
                result = Utility.handleCityResponse(responseText, selectedProvince.getId());
            } else if ("county".equals(type)) {
                result = Utility.handleCountyResponse(responseText, selectedCity.getId());
            }
            if (result) {
                getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        closeProgressDialog();
                        if ("province".equals(type)) {

```

```
        queryProvinces();
    } else if ("city".equals(type)) {
        queryCities();
    } else if ("county".equals(type)) {
        queryCounties();
    }
    }
    });
}

@Override
public void onFailure(Call call, IOException e) {
    // 通过 runOnUiThread()方法回到主线程处理逻辑
    getActivity().runOnUiThread(new Runnable() {
        @Override
        public void run() {
            closeProgressDialog();
            Toast.makeText(getContext(), "加载失败", Toast.LENGTH_SHORT).
                show();
        }
    });
}

/**
 * 显示进度对话框
 */
private void showProgressDialog() {
    if (progressDialog == null) {
        progressDialog = new ProgressDialog(getActivity());
        progressDialog.setMessage("正在加载...");
        progressDialog.setCanceledOnTouchOutside(false);
    }
    progressDialog.show();
}

/**
 * 关闭进度对话框
 */
private void closeProgressDialog() {
    if (progressDialog != null) {
        progressDialog.dismiss();
    }
}
}
```

这个类里的代码虽然非常多，可是逻辑却不复杂，我们来慢慢理一下。在 `onCreateView()` 方法中先是获取到了一些控件的实例，然后去初始化了 `ArrayAdapter`，并将它设置为 `ListView` 的适配器。接着在 `onActivityCreated()` 方法中给 `ListView` 和 `Button` 设置了点击事件，到这里我们的初始化工作就算是完成了。

在 `onActivityCreated()` 方法的最后，调用了 `queryProvinces()` 方法，也就是从这里开始加载省级数据的。`queryProvinces()` 方法中首先会将头布局的标题设置成中国，将返回按钮隐藏起来，因为省级列表已经不能再返回了。然后调用 LitePal 的查询接口来从数据库中读取省级数据，如果读取到了就直接将数据显示到界面上，如果没有读取到就按照 14.1 节讲述的接口组装出一个请求地址，然后调用 `queryFromServer()` 方法来从服务器上查询数据。

`queryFromServer()` 方法中会调用 `HttpUtil` 的 `sendOkHttpRequest()` 方法来向服务器发送请求，响应的数据会回调到 `onResponse()` 方法中，然后我们在这里去调用 `Utility` 的 `handleProvincesResponse()` 方法来解析和处理服务器返回的数据，并存储到数据库中。接下来的一步很关键，在解析和处理完数据之后，我们再次调用了 `queryProvinces()` 方法来重新加载省级数据，由于 `queryProvinces()` 方法牵扯到了 UI 操作，因此必须要在主线程中调用，这里借助了 `runOnUiThread()` 方法来实现从子线程切换到主线程。现在数据库中已经存在了数据，因此调用 `queryProvinces()` 就会直接将数据显示到界面上了。

当你点击了某个省的时候会进入到 `ListView` 的 `onItemClick()` 方法中，这个时候会根据当前的级别来判断是去调用 `queryCities()` 方法还是 `queryCounties()` 方法，`queryCities()` 方法是去查询市级数据，而 `queryCounties()` 方法是去查询县级数据，这两个方法内部的流程和 `queryProvinces()` 方法基本相同，这里就不重复讲解了。

另外还有一点需要注意，在返回按钮的点击事件里，会对当前 `ListView` 的列表级别进行判断。如果当前是县级列表，那么就返回到市级列表，如果当前是市级列表，那么就返回到省级列表。当返回到省级列表时，返回按钮会自动隐藏，从而也就不需要再做进一步的处理了。

这样我们就把遍历全国省市县的功能完成了，可是碎片是不能直接显示在界面上的，因此我们还需要把它添加到活动里才行。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">

    <fragment
        android:id="@+id/choose_area_fragment"
        android:name="com.coolweather.android.ChooseAreaFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

布局文件很简单，只是定义了一个 `FrameLayout`，然后将 `ChooseAreaFragment` 添加进来，并让它充满整个布局。

另外，我们刚才在碎片的布局里面已经自定义了一个标题栏，因此就不再需要原生的 `ActionBar` 了，修改 `res/values/styles.xml` 中的代码，如下所示：

```
<resources>

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

</resources>
```

现在第二阶段的开发工作也完成得差不多了，我们可以运行一下来看看效果。不过在运行之前还有一件事没有做，那就是声明程序所需要的权限。修改 `AndroidManifest.xml` 中的代码，如下所示：

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

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

    ...

</manifest>
```

由于我们是通过网络接口来获取全国省市县数据的，因此必须要添加访问网络的权限才行。现在可以运行一下程序了，结果如图 14.18 所示。

可以看到，全国所有省级数据都显示出来了。我们还可以继续查看市级数据，比如点击江苏省，结果如图 14.19 所示。

这个时候标题栏上会出现一个返回按钮，用于返回上一级列表。

然后再点击苏州市查看县级数据，结果如图 14.20 所示。

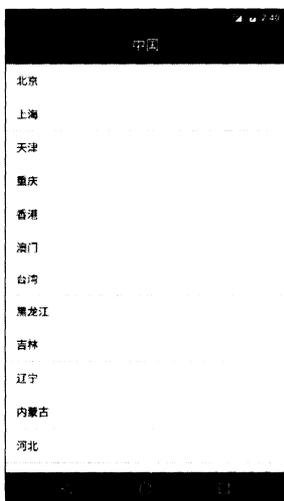


图 14.18 显示省级数据

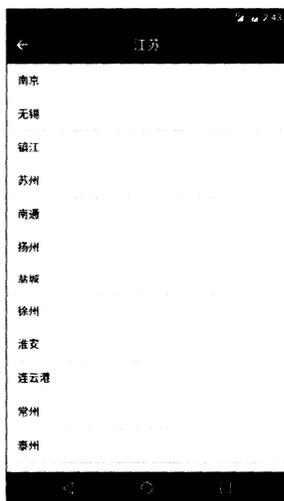


图 14.19 显示市级数据

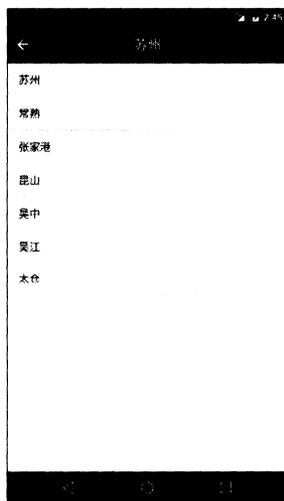


图 14.20 显示县级数据

好了，这样第二阶段的开发工作也都完成了，我们仍然要把代码提交一下。

```
git add .
git commit -m "完成遍历省市县三级列表的功能。"
git push origin master
```

到目前为止进度算是相当不错啊，那么我们就趁热打铁，来进行第三阶段的开发工作。

14.5 显示天气信息

在第三阶段中，我们就要开始去查询天气，并且把天气信息显示出来了。由于和风天气返回的 JSON 数据结构非常复杂，如果还使用 JSONObject 来解析就会很麻烦，这里我们就准备借助 GSON 来对天气信息进行解析了。

14.5.1 定义 GSON 实体类

GSON 的用法很简单，解析数据只需要一行代码就能完成了，但前提是要先将数据对应的实体类创建好。由于和风天气返回的数据内容非常多，这里我们不可能将所有的内容都利用起来，因此我筛选了一些比较重要的数据来进行解析。

首先我们回顾一下返回数据的大致格式：

```
{
  "HeWeather": [
    {
      "status": "ok",
      "basic": {},
      "aqi": {},
      "now": {},
      "suggestion": {},
      "daily_forecast": []
    }
  ]
}
```

其中，basic、aqi、now、suggestion 和 daily_forecast 的内部又都会有具体的内容，那么我们就可以将这 5 个部分定义成 5 个实体类。

下面开始来一个个看，basic 中具体内容如下所示：

```
"basic":{
  "city":"苏州",
  "id":"CN101190401",
  "update":{
    "loc":"2016-08-08 21:58"
  }
}
```

其中，city 表示城市名，id 表示城市对应的天气 id，update 中的 loc 表示天气的更新时

间。我们按照此结构就可以在 gson 包下建立一个 Basic 类，代码如下所示：

```
public class Basic {  
  
    @SerializedName("city")  
    public String cityName;  
  
    @SerializedName("id")  
    public String weatherId;  
  
    public Update update;  
  
    public class Update {  
  
        @SerializedName("loc")  
        public String updateTime;  
  
    }  
  
}
```

由于 JSON 中的一些字段可能不太适合直接作为 Java 字段来命名，因此这里使用了 @SerializedName 注解的方式来让 JSON 字段和 Java 字段之间建立映射关系。

这样我们就将 Basic 类定义好了，还是挺容易理解的吧？其余的几个实体类也是类似的，我们使用同样的方式来定义就可以了。比如 api 中的具体内容如下所示：

```
"aqi":{  
  "city":{  
    "aqi":"44",  
    "pm25":"13"  
  }  
}
```

那么，在 gson 包下新建一个 AQI 类，代码如下所示：

```
public class AQI {  
  
    public AQICity city;  
  
    public class AQICity {  
  
        public String aqi;  
  
        public String pm25;  
  
    }  
  
}
```

now 中的具体内容如下所示：

```
"now":{
```

```
"tmp": "29",  
"cond": {  
  "txt": "降雨"  
}  
}
```

那么，在 gson 包下新建一个 Now 类，代码如下所示：

```
public class Now {  
  
  @SerializedName("tmp")  
  public String temperature;  
  
  @SerializedName("cond")  
  public More more;  
  
  public class More {  
  
    @SerializedName("txt")  
    public String info;  
  
  }  
  
}
```

suggestion 中的具体内容如下所示：

```
"suggestion": {  
  "comf": {  
    "txt": "白天天气较热，虽然有雨，但仍然无法削弱较高气温给人们带来的暑意，  
    这种天气会让您感到不很舒适。"  
  },  
  "cw": {  
    "txt": "不宜洗车，未来 24 小时内有雨，如果在此期间洗车，雨水和路上的泥水  
    可能会再次弄脏您的爱车。"  
  },  
  "sport": {  
    "txt": "有降水，且风力较强，推荐您在室内进行低强度运动；若坚持户外运动，  
    请选择避雨防风的地点。"  
  }  
}
```

那么，在 gson 包下新建一个 Suggestion 类，代码如下所示：

```
public class Suggestion {  
  
  @SerializedName("comf")  
  public Comfort comfort;  
  
  @SerializedName("cw")  
  public CarWash carWash;  
  
  public Sport sport;  
  
}
```

```
public class Comfort {  
    @SerializedName("txt")  
    public String info;  
}  
  
public class CarWash {  
    @SerializedName("txt")  
    public String info;  
}  
  
public class Sport {  
    @SerializedName("txt")  
    public String info;  
}  
}
```

到目前为止都还比较简单，不过接下来的一项数据就有点特殊了，`daily_forecast` 中的具体内容如下所示：

```
"daily_forecast": [  
  {  
    "date": "2016-08-08",  
    "cond": {  
      "txt_d": "阵雨"  
    },  
    "tmp": {  
      "max": "34",  
      "min": "27"  
    }  
  },  
  {  
    "date": "2016-08-09",  
    "cond": {  
      "txt_d": "多云"  
    },  
    "tmp": {  
      "max": "35",  
      "min": "29"  
    }  
  },  
  ...  
]
```

可以看到，`daily_forecast` 中包含的是一个数组，数组中的每一项都代表着未来一天的天气信息。针对于这种情况，我们只需要定义出单日天气的实体类就可以了，然后在声明实体类引用的时候使用集合类型来进行声明。

那么在 gson 包下新建一个 Forecast 类，代码如下所示：

```
public class Forecast {  
  
    public String date;  
  
    @SerializedName("tmp")  
    public Temperature temperature;  
  
    @SerializedName("cond")  
    public More more;  
  
    public class Temperature {  
  
        public String max;  
  
        public String min;  
  
    }  
  
    public class More {  
  
        @SerializedName("txt_d")  
        public String info;  
  
    }  
  
}
```

这样我们就把 basic、aqi、now、suggestion 和 daily_forecast 对应的实体类全部都创建好了，接下来还需要再创建一个总的实例类来引用刚刚创建的各个实体类。在 gson 包下新建一个 Weather 类，代码如下所示：

```
public class Weather {  
  
    public String status;  
  
    public Basic basic;  
  
    public AQI aqi;  
  
    public Now now;  
  
    public Suggestion suggestion;  
  
    @SerializedName("daily_forecast")  
    public List<Forecast> forecastList;  
  
}
```

在 Weather 类中，我们对 Basic、AQI、Now、Suggestion 和 Forecast 类进行了引用。其中，由于 daily_forecast 中包含的是一个数组，因此这里使用了 List 集合来引用 Forecast 类。

另外，返回的天气数据中还会包含一项 status 数据，成功返回 ok，失败则会返回具体的原因，那么这里也需要添加一个对应的 status 字段。

现在所有的 GSON 实体类都定义好了，接下来我们开始编写天气界面。

14.5.2 编写天气界面

首先创建一个用于显示天气信息的活动。右击 com.coolweather.android 包→New→Activity→Empty Activity，创建一个 WeatherActivity，并将布局名指定成 activity_weather.xml。

由于所有的天气信息都将在同一个界面上显示，因此 activity_weather.xml 会是一个很长的布局文件。那么为了让里面的代码不至于混乱不堪，这里我准备使用 3.4.1 小节学过的引入布局技术，即将界面的不同部分写在不同的布局文件里面，再通过引入布局的方式集成到 activity_weather.xml 中，这样整个布局文件就会显得非常工整。

右击 res/layout→New→Layout resource file，新建一个 title.xml 作为头布局，代码如下所示：

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

    <TextView
        android:id="@+id/title_city"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"
        android:textColor="#fff"
        android:textSize="20sp" />

    <TextView
        android:id="@+id/title_update_time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="10dp"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:textColor="#fff"
        android:textSize="16sp"/>

</RelativeLayout>
```

这段代码还是比较简单的，头布局中放置了两个 TextView，一个居中显示城市名，一个居右显示更新时间。

然后新建一个 now.xml 作为当前天气信息的布局，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="15dp">

<TextView
    android:id="@+id/degree_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end"
    android:textColor="#fff"
    android:textSize="60sp" />

<TextView
    android:id="@+id/weather_info_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end"
    android:textColor="#fff"
    android:textSize="20sp" />

</LinearLayout>
```

当前天气信息的布局中也是放置了两个 TextView，一个用于显示当前气温，一个用于显示天气概况。

然后新建 forecast.xml 作为未来几天天气信息的布局，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="15dp"
    android:background="#8000">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="15dp"
        android:text="预报"
        android:textColor="#fff"
        android:textSize="20sp"/>

    <LinearLayout
        android:id="@+id/forecast_layout"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    </LinearLayout>

</LinearLayout>
```

这里最外层使用 LinearLayout 定义了一个半透明的背景，然后使用 TextView 定义了一个标

题，接着又使用一个 `LinearLayout` 定义了一个用于显示未来几天天气信息的布局。不过这个布局中并没有放入任何内容，因为这是要根据服务器返回的数据在代码中动态添加的。

为此，我们还需要再定义一个未来天气信息的子项布局，创建 `forecast_item.xml` 文件，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:layout_margin="15dp">

    <TextView
        android:id="@+id/date_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="2"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/info_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_weight="1"
        android:gravity="center"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/max_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="right"
        android:textColor="#fff"/>

    <TextView
        android:id="@+id/min_text"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_weight="1"
        android:gravity="right"
        android:textColor="#fff"/>

</LinearLayout>
```

子项布局中放置了 4 个 `TextView`，一个用于显示天气预报日期，一个用于显示天气概况，另外两个分别用于显示当天的最高温度和最低温度。

然后新建 `aqi.xml` 作为空气质量信息的布局，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="15dp"
    android:background="#8000">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="15dp"
        android:text="空气质量"
        android:textColor="#fff"
        android:textSize="20sp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="15dp">

        <RelativeLayout
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1">

            <LinearLayout
                android:orientation="vertical"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true">

                <TextView
                    android:id="@+id/aqi_text"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:textColor="#fff"
                    android:textSize="40sp"
                    />

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="center"
                    android:text="AQI 指数"
                    android:textColor="#fff"/>

            </LinearLayout>

        </RelativeLayout>

    <RelativeLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
```

```
        android:layout_weight="1">
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true">
            <TextView
                android:id="@+id/pm25_text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:textColor="#fff"
                android:textSize="40sp"
            />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center"
                android:text="PM2.5 指数"
                android:textColor="#fff"
            />
        </LinearLayout>
    </RelativeLayout>
</LinearLayout>
</LinearLayout>
```

这个布局中的代码虽然看上去有点长，但是并不复杂。首先前面都是一样的，使用 `LinearLayout` 定义了一个半透明的背景，然后使用 `TextView` 定义了一个标题。接下来，这里使用 `LinearLayout` 和 `RelativeLayout` 嵌套的方式实现了一个左右平分并且居中对齐的布局，分别用于显示 AQI 指数和 PM 2.5 指数。相信你只要仔细看一看，这个布局还是很好理解的。

然后新建 `suggestion.xml` 作为生活建议信息的布局，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="15dp"
    android:background="#8000">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginTop="15dp"
        android:text="生活建议"
```

```
        android:textColor="#fff"
        android:textSize="20sp"/>

<TextView
    android:id="@+id/comfort_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="15dp"
    android:textColor="#fff" />

<TextView
    android:id="@+id/car_wash_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="15dp"
    android:textColor="#fff" />

<TextView
    android:id="@+id/sport_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="15dp"
    android:textColor="#fff" />

</LinearLayout>
```

这里同样也是先定义了一个半透明的背景和一个标题，然后下面使用了 3 个 `TextView` 分别用于显示舒适度、洗车指数和运动建议的相关数据。

这样我们就把天气界面上每个部分的布局文件都编写好了，接下来的工作就是将它们引入到 `activity_weather.xml` 当中，如下所示：

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

    <ScrollView
        android:id="@+id/weather_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="none"
        android:overScrollMode="never">

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

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

            <include layout="@layout/now" />
```

```
<include layout="@layout/forecast" />

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

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

</LinearLayout>

</ScrollView>

</FrameLayout>
```

可以看到，首先最外层布局使用了一个 `FrameLayout`，并将它的背景色设置成 `colorPrimary`。然后在 `FrameLayout` 中嵌套了一个 `ScrollView`，这是因为天气界面中的内容比较多，使用 `ScrollView` 可以允许我们通过滚动的方式查看屏幕以外的内容。

由于 `ScrollView` 的内部只允许存在一个直接子布局，因此这里又嵌套了一个垂直方向的 `LinearLayout`，然后在 `LinearLayout` 中将刚才定义的所有布局逐个引入。

这样我们就将天气界面编写完成了，接下来开始编写业务逻辑，将天气显示到界面上。

14.5.3 将天气显示到界面上

首先需要在 `Utility` 类中添加一个用于解析天气 JSON 数据的方法，如下所示：

```
public class Utility {

    ...

    /**
     * 将返回的 JSON 数据解析成 Weather 实体类
     */
    public static Weather handleWeatherResponse(String response) {
        try {
            JSONObject jsonObject = new JSONObject(response);
            JSONArray jsonArray = jsonObject.getJSONArray("HeWeather");
            String weatherContent = jsonArray.getJSONObject(0).toString();
            return new Gson().fromJson(weatherContent, Weather.class);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

可以看到，`handleWeatherResponse()`方法中先是通过 `JSONObject` 和 `JSONArray` 将天气数据中的主体内容解析出来，即如下内容：

```
{
  "status": "ok",
  "basic": {},
  "aqi": {},
```

```
"now": {},  
"suggestion": {},  
"daily_forecast": []  
}
```

然后由于我们之前已经按照上面的数据格式定义过相应的 GSON 实体类，因此只需要通过调用 `fromJson()` 方法就能直接将 JSON 数据转换成 `Weather` 对象了。

接下来的工作是我们如何在活动中去请求天气数据，以及将数据展示到界面上。修改 `WeatherActivity` 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {  
  
    private ScrollView weatherLayout;  
  
    private TextView titleCity;  
  
    private TextView titleUpdateTime;  
  
    private TextView degreeText;  
  
    private TextView weatherInfoText;  
  
    private LinearLayout forecastLayout;  
  
    private TextView aqiText;  
  
    private TextView pm25Text;  
  
    private TextView comfortText;  
  
    private TextView carWashText;  
  
    private TextView sportText;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_weather);  
        // 初始化各控件  
        weatherLayout = (ScrollView) findViewById(R.id.weather_layout);  
        titleCity = (TextView) findViewById(R.id.title_city);  
        titleUpdateTime = (TextView) findViewById(R.id.title_update_time);  
        degreeText = (TextView) findViewById(R.id.degree_text);  
        weatherInfoText = (TextView) findViewById(R.id.weather_info_text);  
        forecastLayout = (LinearLayout) findViewById(R.id.forecast_layout);  
        aqiText = (TextView) findViewById(R.id.aqi_text);  
        pm25Text = (TextView) findViewById(R.id.pm25_text);  
        comfortText = (TextView) findViewById(R.id.comfort_text);  
        carWashText = (TextView) findViewById(R.id.car_wash_text);  
        sportText = (TextView) findViewById(R.id.sport_text);  
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences  
            (this);  
        String weatherString = prefs.getString("weather", null);  
        if (weatherString != null) {
```

```
// 有缓存时直接解析天气数据
Weather weather = Utility.handleWeatherResponse(weatherString);
showWeatherInfo(weather);
} else {
    // 无缓存时去服务器查询天气
    String weatherId = getIntent().getStringExtra("weather_id");
    weatherLayout.setVisibility(View.INVISIBLE);
    requestWeather(weatherId);
}
}

/**
 * 根据天气 id 请求城市天气信息
 */
public void requestWeather(final String weatherId) {

    String weatherUrl = "http://guolin.tech/api/weather?cityid=" +
        weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";
    HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
        @Override
        public void onResponse(Call call, Response response) throws IOException {
            final String responseText = response.body().string();
            final Weather weather = Utility.handleWeatherResponse(responseText);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    if (weather != null && "ok".equals(weather.status)) {
                        SharedPreferences.Editor editor = PreferenceManager.
                            getDefaultSharedPreferences(WeatherActivity.this).
                                edit();
                        editor.putString("weather", responseText);
                        editor.apply();
                        showWeatherInfo(weather);
                    } else {
                        Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                            Toast.LENGTH_SHORT).show();
                    }
                }
            });
        }
    });
}

@Override
public void onFailure(Call call, IOException e) {
    e.printStackTrace();
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                Toast.LENGTH_SHORT).show();
        }
    });
}
});
}
});
}
```

```
/**
 * 处理并展示 Weather 实体类中的数据
 */
private void showWeatherInfo(Weather weather) {
    String cityName = weather.basic.cityName;
    String updateTime = weather.basic.update.updateTime.split(" ")[1];
    String degree = weather.now.temperature + "°C";
    String weatherInfo = weather.now.more.info;
    titleCity.setText(cityName);
    titleUpdateTime.setText(updateTime);
    degreeText.setText(degree);
    weatherInfoText.setText(weatherInfo);
    forecastLayout.removeAllViews();
    for (Forecast forecast : weather.forecastList) {
        View view = LayoutInflater.from(this).inflate(R.layout.forecast_
            item, forecastLayout, false);
        TextView dateText = (TextView) view.findViewById(R.id.date_text);
        TextView infoText = (TextView) view.findViewById(R.id.info_text);
        TextView maxText = (TextView) view.findViewById(R.id.max_text);
        TextView minText = (TextView) view.findViewById(R.id.min_text);
        dateText.setText(forecast.date);
        infoText.setText(forecast.more.info);
        maxText.setText(forecast.temperature.max);
        minText.setText(forecast.temperature.min);
        forecastLayout.addView(view);
    }
    if (weather.aqi != null) {
        aqiText.setText(weather.aqi.city.aqi);
        pm25Text.setText(weather.aqi.city.pm25);
    }
    String comfort = "舒适度：" + weather.suggestion.comfort.info;
    String carWash = "洗车指数：" + weather.suggestion.carWash.info;
    String sport = "运动建议：" + weather.suggestion.sport.info;
    comfortText.setText(comfort);
    carWashText.setText(carWash);
    sportText.setText(sport);
    weatherLayout.setVisibility(View.VISIBLE);
}
}
```

这个活动中的代码也比较长，我们还是一步步梳理下。在 `onCreate()` 方法中仍然先去获取一些控件的实例，然后会尝试从本地缓存中读取天气数据。那么第一次肯定是没有缓存的，因此就会从 `Intent` 中取出天气 `id`，并调用 `requestWeather()` 方法来从服务器请求天气数据。注意，请求数据的时候先将 `ScrollView` 进行隐藏，不然空数据的界面看上去会很奇怪。

`requestWeather()` 方法中先是使用了参数中传入的天气 `id` 和我们之前申请好的 `API Key` 拼装出一个接口地址，接着调用 `HttpUtil.sendOkHttpRequest()` 方法来向该地址发出请求，服务器会将相应城市的天气信息以 `JSON` 格式返回。然后我们在 `onResponse()` 回调中先调用 `Utility.handleWeatherResponse()` 方法将返回的 `JSON` 数据转换成 `Weather` 对象，再将当前线程切换到主线程。然后进行判断，如果服务器返回的 `status` 状态是 `ok`，就说明请求天气成功了，此时将返

回的数据缓存到 SharedPreferences 当中，并调用 showWeatherInfo() 方法来进行内容显示。

showWeatherInfo() 方法中的逻辑就比较简单了，其实就是从 Weather 对象中获取数据，然后显示到相应的控件上。注意在未来几天天气预报的部分我们使用了一个 for 循环来处理每天的天气信息，在循环中动态加载 forecast_item.xml 布局并设置相应的数据，然后添加到父布局当中。设置完了所有数据之后，记得要将 ScrollView 重新变成可见。

这样我们就将首次进入 WeatherActivity 时的逻辑全部梳理完了，那么当下一次再进入 WeatherActivity 时，由于缓存已经存在了，因此会直接解析并显示天气数据，而不会再次发起网络请求了。

处理完了 WeatherActivity 中的逻辑，接下来我们要做的，就是如何从省市县列表界面跳转到天气界面了，修改 ChooseAreaFragment 中的代码，如下所示：

```
public class ChooseAreaFragment extends Fragment {  
  
    ...  
  
    @Override  
    public void onActivityCreated(Bundle savedInstanceState) {  
        super.onActivityCreated(savedInstanceState);  
        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> parent, View view, int position,  
                long id) {  
                if (currentLevel == LEVEL_PROVINCE) {  
                    selectedProvince = provinceList.get(position);  
                    queryCities();  
                } else if (currentLevel == LEVEL_CITY) {  
                    selectedCity = cityList.get(position);  
                    queryCounties();  
                } else if (currentLevel == LEVEL_COUNTY) {  
                    String weatherId = countyList.get(position).getWeatherId();  
                    Intent intent = new Intent(getActivity(), WeatherActivity.  
                        class);  
                    intent.putExtra("weather_id", weatherId);  
                    startActivity(intent);  
                    getActivity().finish();  
                }  
            }  
        });  
        ...  
    }  
  
    ...  
  
}
```

非常简单，这里在 onItemClick() 方法中加入了一个 if 判断，如果当前级别是 LEVEL_COUNTY，就启动 WeatherActivity，并把当前选中县的天气 id 传递过去。

另外，我们还需要在 MainActivity 中加入一个缓存数据的判断才行。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);  
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences  
            (this);  
        if (prefs.getString("weather", null) != null) {  
            Intent intent = new Intent(this, WeatherActivity.class);  
            startActivity(intent);  
            finish();  
        }  
    }  
}
```

可以看到，这里在 onCreate() 方法的一开始先从 SharedPreferences 文件中读取缓存数据，如果不为 null 就说明之前已经请求过天气数据了，那么就没必要让用户再次选择城市，而是直接跳转到 WeatherActivity 即可。

好了，现在重新运行一下程序，然后选择江苏→苏州→昆山，结果如图 14.21 所示。

然后我们还可以向下滑动查看更多天气信息，如图 14.22 所示。

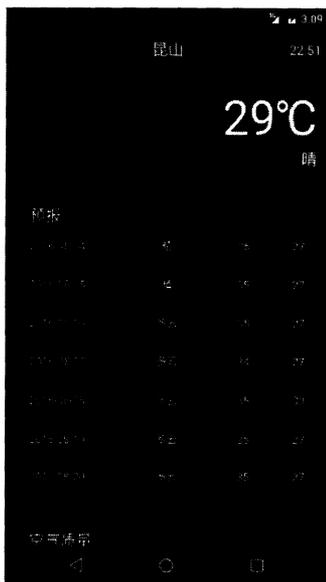


图 14.21 显示天气信息



图 14.22 查看更多天气信息

14.5.4 获取必应每日一图

虽说现在我们已经把天气界面编写得非常不错了，不过和市场上的一些天气软件的界面相比，仍然还是有一定差距的。出色的天气软件不会像我们现在这样使用一个固定的背景色，而是会根据不同的城市或者天气情况展示不同的背景图片。

当然实现这个功能并不复杂，最重要的是需要有服务器的接口支持。不过我实在是没有精力去准备这样一套完善的服务器接口，那么为了不让我们的天气界面过于单调，这里我准备使用一个巧妙的办法。

必应想必你肯定不会陌生，这是一个由微软开发的搜索引擎网站。这个网站除了提供强大的搜索功能之外，还有一个非常有特色的地方，就是它每天都会在首页展示一张精美的背景图片，如图 14.23 所示。



图 14.23 必应的首页

由于这些图片都是由必应精挑细选出来的，并且每天都会变化，如果我们使用它们来作为天气界面的背景图，不仅可以使界面变得更加美观，而且解决了界面一成不变、过于单调的问题。

为此我专门准备了一个获取必应每日一图的接口：http://guolin.tech/api/bing_pic。

访问这个接口，服务器会返回今日的必应背景图链接：

http://cn.bing.com/az/hprichbg/rb/ChicagoHarborLH_ZH-CN9974330969_1920x1080.jpg。

然后我们再使用 Glide 去加载这张图片就可以了。

总体思路就是这么简单，下面开始来动手实现吧。首先修改 `activity_weather.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"
android:background="@color/colorPrimary">

<ImageView
    android:id="@+id/bing_pic_img"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop" />

<ScrollView
    android:id="@+id/weather_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scrollbars="none"
    android:overScrollMode="never">

    ...

</ScrollView>

</FrameLayout>
```

这里我们在 `FrameLayout` 中添加了一个 `ImageView`，并且将它的宽和高都设置成 `match_parent`。由于 `FrameLayout` 默认情况下会将控件都放置在左上角，因此 `ScrollView` 会完全覆盖住 `ImageView`，从而 `ImageView` 也就成为背景图片了。

接着修改 `WeatherActivity` 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {

    ...

    private ImageView bingPicImg;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_weather);
        // 初始化各控件
        bingPicImg = (ImageView) findViewById(R.id.bing_pic_img);
        ...
        String bingPic = prefs.getString("bing_pic", null);
        if (bingPic != null) {
            Glide.with(this).load(bingPic).into(bingPicImg);
        } else {
            loadBingPic();
        }
    }

    /**
     * 根据天气 id 请求城市天气信息
     */
    public void requestWeather(final String weatherId) {
```

```
        ...
        loadBingPic();
    }

    /**
     * 加载必应每日一图
     */
    private void loadBingPic() {
        String requestBingPic = "http://guolin.tech/api/bing_pic";
        HttpUtil.sendOkHttpRequest(requestBingPic, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                final String bingPic = response.body().string();
                SharedPreferences.Editor editor = PreferenceManager.
                    getDefaultSharedPreferences(WeatherActivity.this).edit();
                editor.putString("bing_pic", bingPic);
                editor.apply();
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Glide.with(WeatherActivity.this).load(bingPic).into
                            (bingPicImg);
                    }
                });
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }
        });
    }

    ...
}
```

可以看到，首先在 `onCreate()` 方法中获取了新增控件 `ImageView` 的实例，然后尝试从 `SharedPreferences` 中读取缓存的背景图片。如果有缓存的话就直接使用 `Glide` 来加载这张图片，如果没有的话就调用 `loadBingPic()` 方法去请求今日的必应背景图。

`loadBingPic()` 方法中的逻辑就非常简单了，先是调用了 `HttpUtil.sendOkHttpRequest()` 方法获取到必应背景图的链接，然后将这个链接缓存在 `SharedPreferences` 当中，再将当前线程切换到主线程，最后使用 `Glide` 来加载这张图片就可以了。另外需要注意，在 `requestWeather()` 方法的最后也需要调用一下 `loadBingPic()` 方法，这样在每次请求天气信息的时候同时也会刷新背景图片。现在重新运行一下程序，效果如图 14.24 所示。

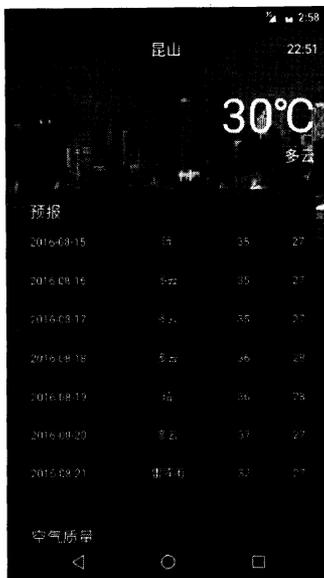


图 14.24 在天气界面显示必应背景图

怎么样？虽说只是换了一张背景图而已，但是整个界面的视觉体验就完全不一样了，瞬间提升了好几个档次。而且我们的背景图并不是一成不变的，每天都会是不同的图片，永远给人一种耳目一新的感觉。

不过如果你仔细观察图 14.24，你会发现背景图并没有和状态栏融合到一起，这样的话视觉体验就还是没有达到最佳的效果。虽说我们在 12.7.2 小节已经学习过如何将背景图和状态栏融合到一起，但当时是借助 Design Support 库完成的，而我们这个项目中并没有引入 Design Support 库。

当然如果还是模仿 12.7.2 小节的做法，引入 Design Support 库，然后嵌套 CoordinatorLayout、AppBarLayout、CollapsingToolbarLayout 等布局，也能实现背景图和状态栏融合到一起的效果，不过这样做就过于麻烦了，这里我准备教你另外一种更简单的实现方式。修改 WeatherActivity 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {  
  
    ...  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        if (Build.VERSION.SDK_INT >= 21) {  
            View decorView = getWindow().getDecorView();  
            decorView.setSystemUiVisibility(  
                View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN  
                | View.SYSTEM_UI_FLAG_LAYOUT_STABLE);  
        }  
    }  
}
```

```
        getWindow().setStatusBarColor(Color.TRANSPARENT);
    }
    setContentView(R.layout.activity_weather);
    ...
}
...
}
```

由于这个功能是 Android 5.0 及以上的系统才支持的，因此我们先在代码中做了一个系统版本的判断，只有当版本号大于或等于 21，也就是 5.0 及以上系统时才会执行后面的代码。

接着我们调用了 `getWindow().getDecorView()` 方法拿到当前活动的 `DecorView`，再调用它的 `setSystemUiVisibility()` 方法来改变系统 UI 的显示，这里传入 `View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN` 和 `View.SYSTEM_UI_FLAG_LAYOUT_STABLE` 就表示活动的布局会显示在状态栏上面，最后调用一下 `setStatusBarColor()` 方法将状态栏设置成透明色。

仅仅这些代码就可以实现让背景图和状态栏融合到一起的效果了。不过，如果运行一下程序，你会发现还是有些问题，天气界面的头布局几乎和系统状态栏紧贴到一起了，如图 14.25 所示。

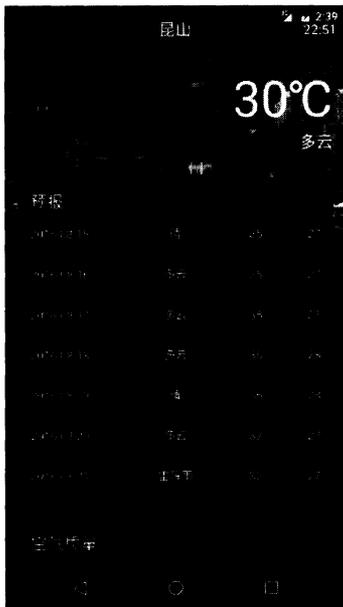


图 14.25 头布局 and 状态栏紧贴在一起

这是由于系统状态栏已经成为我们布局的一部分，因此没有单独为它留出空间。当然，这个问题也是非常好解决的，借助 `android:fitsSystemWindows` 属性就可以了。修改 `activity_weather.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"
    android:background="@color/colorPrimary">

    ...

    <ScrollView
        android:id="@+id/weather_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scrollbars="none"
        android:overScrollMode="never">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fitsSystemWindows="true">

            ...

        </LinearLayout>

    </ScrollView>

</FrameLayout>
```

这里在 `ScrollView` 的 `LinearLayout` 中增加了 `android:fitsSystemWindows` 属性，设置成 `true` 就表示会为系统状态栏留出空间。现在重新运行一下代码，效果如图 14.26 所示。



图 14.26 为系统状态栏留出空间

OK，这样第三阶段的开发工作也都完成了，我们把代码提交一下。

```
git add .
git commit -m "加入显示天气信息的功能。"
git push origin master
```

14.6 手动更新天气和切换城市

经过第三阶段的开发，现在酷欧天气的主体功能已经有了，不过你会发现目前存在着一个比较严重的 bug，就是当你选中了某一个城市之后，就没法再去查看其他城市的天气了，即使退出程序，下次进来的时候还会直接跳转到 WeatherActivity。

因此，在第四阶段中我们要加入切换城市的功能，并且为了能够实时获取到最新的天气，我们还会加入手动更新天气的功能。

14.6.1 手动更新天气

先来实现一下手动更新天气的功能。由于我们在上一节中对天气信息进行了缓存，目前每次展示的都是缓存中的数据，因此现在非常需要一种方式能够让用户手动更新天气信息。

至于如何触发更新事件呢？这里我准备采用下拉刷新的方式，正好我们之前也学过下拉刷新的用法，实现起来会比较简单。

首先修改 activity_weather.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"
    android:background="@color/colorPrimary">
    ...
    <android.support.v4.widget.SwipeRefreshLayout
        android:id="@+id/swipe_refresh"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <ScrollView
            android:id="@+id/weather_layout"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:scrollbars="none"
            android:overScrollMode="never">
            ...
        </ScrollView>
```

```
</android.support.v4.widget.SwipeRefreshLayout>
```

```
</FrameLayout>
```

可以看到，这里在 ScrollView 的外面又嵌套了一层 SwipeRefreshLayout，这样 ScrollView 就自动拥有下拉刷新功能了。

然后修改 WeatherActivity 中的代码，加入更新天气的处理逻辑，如下所示：

```
public class WeatherActivity extends AppCompatActivity {  
  
    public SwipeRefreshLayout swipeRefresh;  
  
    ...  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        swipeRefresh = (SwipeRefreshLayout) findViewById(R.id.swipe_refresh);  
        swipeRefresh.setColorSchemeResources(R.color.colorPrimary);  
        SharedPreferences prefs = PreferenceManager.  
            getDefaultSharedPreferences(this);  
        String weatherString = prefs.getString("weather", null);  
        final String weatherId;  
        if (weatherString != null) {  
            // 有缓存时直接解析天气数据  
            Weather weather = Utility.handleWeatherResponse(weatherString);  
            weatherId = weather.basic.weatherId;  
            showWeatherInfo(weather);  
        } else {  
            // 无缓存时去服务器查询天气  
            weatherId = getIntent().getStringExtra("weather_id");  
            weatherLayout.setVisibility(View.INVISIBLE);  
            requestWeather(weatherId);  
        }  
        swipeRefresh.setOnRefreshListener(new SwipeRefreshLayout.  
            OnRefreshListener() {  
                @Override  
                public void onRefresh() {  
                    requestWeather(weatherId);  
                }  
            });  
        ...  
    }  
  
    /**  
     * 根据天气 id 请求城市天气信息  
     */  
    public void requestWeather(final String weatherId) {  
  
        String weatherUrl = "http://guolin.tech/api/weather?cityid=" +  
            weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";  
    }  
}
```

```
HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
    @Override
    public void onResponse(Call call, Response response) throws IOException {
        ...
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (weather != null && "ok".equals(weather.status)) {
                    SharedPreferences.Editor editor = PreferenceManager.
                        getDefaultSharedPreferences(WeatherActivity.
                            this).edit();
                    editor.putString("weather", responseText);
                    editor.apply();
                    showWeatherInfo(weather);
                } else {
                    Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                        Toast.LENGTH_SHORT).show();
                }
                swipeRefresh.setRefreshing(false);
            }
        });
    }

    @Override
    public void onFailure(Call call, IOException e) {
        e.printStackTrace();
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(WeatherActivity.this, "获取天气信息失败",
                    Toast.LENGTH_SHORT).show();
                swipeRefresh.setRefreshing(false);
            }
        });
    }
});
loadBingPic();
}
...
}
```

修改的代码并不算多，首先在 `onCreate()` 方法中获取到了 `SwipeRefreshLayout` 的实例，然后调用 `setColorSchemeResources()` 方法来设置下拉刷新进度条的颜色，这里我们就使用主题中的 `colorPrimary` 作为进度条的颜色了。接着定义了一个 `weatherId` 变量，用于记录城市的天气 id，然后调用 `setOnRefreshListener()` 方法来设置一个下拉刷新的监听器，当触发了下拉刷新操作的时候，就会回调这个监听器的 `onRefresh()` 方法，我们在这里去调用 `requestWeather()` 方法请求天气信息就可以了。

另外不要忘记，当请求结束后，还需要调用 `SwipeRefreshLayout` 的 `setRefreshing()` 方法

并传入 `false`，用于表示刷新事件结束，并隐藏刷新进度条。

现在重新运行一下程序，并在屏幕的主界面向下拖动，效果如图 14.27 所示。

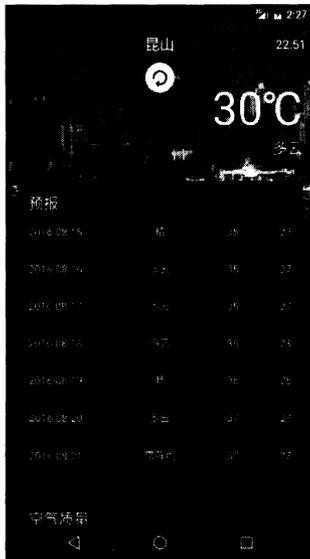


图 14.27 手动更新天气

更新完天气信息之后，下拉进度条会自动消失。

14.6.2 切换城市

完成了手动更新天气的功能，接下来我们继续实现切换城市功能。

既然是要切换城市，那么就肯定需要遍历全国省市县的数据，而这个功能我们早在 14.4 节就已经完成了，并且当时考虑为了方便后面的复用，特意选择了在碎片当中实现。因此，我们其实只需要在天气界面的布局中引入这个碎片，就可以快速集成切换城市功能了。

虽说实现原理很简单，但是显然我们也不可能让引入的碎片把天气界面遮挡住，这又该怎么办呢？还记得 12.3 节学过的滑动菜单功能吗？将碎片放入到滑动菜单中真是再合适不过了，正常情况下它不占据主界面的任何空间，想要切换城市的时候只需要通过滑动的方式将菜单显示出来就可以了。

下面我们就按照这种思路来实现。首先按照 Material Design 的建议，我们需要在头布局中加入一个切换城市的按钮，不然的话用户可能根本就不知道屏幕的左侧边缘是可以拖动的。修改 `title.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="?attr/actionBarSize">
```

```
<Button
    android:id="@+id/nav_button"
    android:layout_width="30dp"
    android:layout_height="30dp"
    android:layout_marginLeft="10dp"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    android:background="@drawable/ic_home" />
```

```
...
```

```
</RelativeLayout>
```

这里添加了一个 Button 作为切换城市的按钮，并且让它居左显示。另外，我提前准备好了一张图片来作为按钮的背景图。

接着修改 activity_weather.xml 布局来加入滑动菜单功能，如下所示：

```
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorPrimary">
    ...
    <android.support.v4.widget.DrawerLayout
        android:id="@+id/drawer_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <android.support.v4.widget.SwipeRefreshLayout
            android:id="@+id/swipe_refresh"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            ...
        </android.support.v4.widget.SwipeRefreshLayout>

        <fragment
            android:id="@+id/choose_area_fragment"
            android:name="com.coolweather.android.ChooseAreaFragment"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            />

    </android.support.v4.widget.DrawerLayout>
</FrameLayout>
```

可以看到，我们在 `SwipeRefreshLayout` 的外面又嵌套了一层 `DrawerLayout`。`DrawerLayout` 中的第一个子控件用于作为主屏幕中显示的内容，第二个子控件用于作为滑动菜单中显示的内容，因此这里我们在第二个子控件的位置添加了用于遍历省市县数据的碎片。

接下来需要在 `WeatherActivity` 中加入滑动菜单的逻辑处理，修改 `WeatherActivity` 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {  
  
    public DrawerLayout drawerLayout;  
  
    private Button navButton;  
  
    ...  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        ...  
        drawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);  
        navButton = (Button) findViewById(R.id.nav_button);  
        ...  
        navButton.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                drawerLayout.openDrawer(GravityCompat.START);  
            }  
        });  
    }  
  
    ...  
  
}
```

很简单，首先在 `onCreate()` 方法中获取到新增的 `DrawerLayout` 和 `Button` 的实例，然后在 `Button` 的点击事件中调用 `DrawerLayout` 的 `openDrawer()` 方法来打开滑动菜单就可以了。

不过现在还没有结束，因为这仅仅是打开了滑动菜单而已，我们还需要处理切换城市后的逻辑才行。这个工作就必须要在 `ChooseAreaFragment` 中进行了，因为之前选中了某个城市后是跳转到 `WeatherActivity` 的，而现在由于我们本来就是在 `WeatherActivity` 当中的，因此并不需要跳转，只是去请求新选择城市的天气信息就可以了。

那么很显然这里我们需要根据 `ChooseAreaFragment` 的不同状态来进行不同的逻辑处理，修改 `ChooseAreaFragment` 中的代码，如下所示：

```
public class ChooseAreaFragment extends Fragment {  
  
    ...  
  
    @Override
```

```
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
            long id) {
            if (currentLevel == LEVEL_PROVINCE) {
                selectedProvince = provinceList.get(position);
                queryCities();
            } else if (currentLevel == LEVEL_CITY) {
                selectedCity = cityList.get(position);
                queryCounties();
            } else if (currentLevel == LEVEL_COUNTY) {
                String weatherId = countyList.get(position).getWeatherId();
                if (getActivity() instanceof MainActivity) {
                    Intent intent = new Intent(getActivity(), WeatherActivity.
                        class);
                    intent.putExtra("weather_id", weatherId);
                    startActivity(intent);
                    getActivity().finish();
                } else if (getActivity() instanceof WeatherActivity) {
                    WeatherActivity activity = (WeatherActivity) getActivity();
                    activity.drawerLayout.closeDrawers();
                    activity.swipeRefresh.setRefreshing(true);
                    activity.requestWeather(weatherId);
                }
            }
        }
    });
    ...
}
...
}
```

这里我使用了一个 Java 中的小技巧，`instanceof` 关键字可以用来判断一个对象是否属于某个类的实例。我们在碎片中调用 `getActivity()` 方法，然后配合 `instanceof` 关键字，就能轻松判断出该碎片是在 `MainActivity` 当中，还是在 `WeatherActivity` 当中。如果是在 `MainActivity` 当中，那么处理逻辑不变。如果是在 `WeatherActivity` 当中，那么就关闭滑动菜单，显示下拉刷新进度条，然后请求新城市的天气信息。

这样我们就把切换城市的功能全部完成了，现在可以重新运行一下程序，效果如图 14.28 所示。

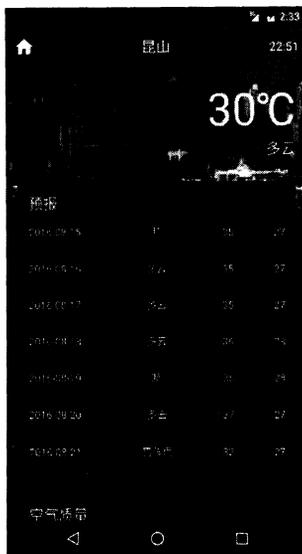


图 14.28 拥有切换城市按钮的天气界面

可以看到，标题栏上多出了一个用于切换城市的按钮。点击该按钮，或者在屏幕的左侧边缘进行拖动，就能让滑动菜单界面显示出来了，如图 14.29 所示。

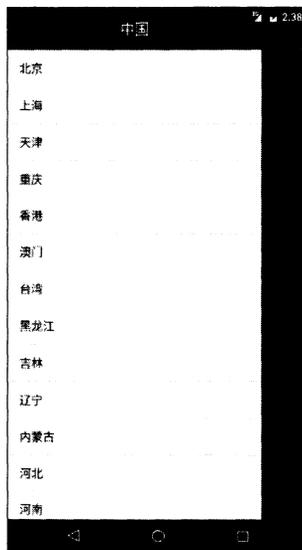


图 14.29 显示滑动菜单界面

然后我们就可以在这里切换其他城市了。选中城市之后滑动菜单会自动关闭，并且主界面上的天气信息也会更新成你选择的那个城市。

这样，第四阶段的开发任务也完成了。当然，仍然不要忘记提交代码。

```
git add .
git commit -m "新增切换城市和手动更新天气的功能。"
git push origin master
```

14.7 后台自动更新天气

为了要让酷欧天气更加智能，在第五阶段我们准备加入后台自动更新天气的功能，这样就可以尽可能地保证用户每次打开软件时看到的都是最新的天气信息。

要想实现上述功能，就需要创建一个长期在后台运行的定时任务，这个功能肯定是难不倒你的，因为我们在 13.5 节中就已经学习过了。

首先在 service 包下新建一个服务，右击 com.coolweather.android.service→New→Service→Service，创建一个 AutoUpdateService，并将 Exported 和 Enabled 这两个属性都勾中。然后修改 AutoUpdateService 中的代码，如下所示：

```
public class AutoUpdateService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        updateWeather();
        updateBingPic();
        AlarmManager manager = (AlarmManager) getSystemService(ALARM_SERVICE);
        int anHour = 8 * 60 * 60 * 1000; // 这是8小时的毫秒数
        long triggerAtTime = SystemClock.elapsedRealtime() + anHour;
        Intent i = new Intent(this, AutoUpdateService.class);
        PendingIntent pi = PendingIntent.getService(this, 0, i, 0);
        manager.cancel(pi);
        manager.set(AlarmManager.ELAPSED_REALTIME_WAKEUP, triggerAtTime, pi);
        return super.onStartCommand(intent, flags, startId);
    }

    /**
     * 更新天气信息
     */
    private void updateWeather(){
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
        String weatherString = prefs.getString("weather", null);
        if (weatherString != null) {
            // 有缓存时直接解析天气数据
            Weather weather = Utility.handleWeatherResponse(weatherString);
            String weatherId = weather.basic.weatherId;

            String weatherUrl = "http://guolin.tech/api/weather?cityid=" +
```

```
        weatherId + "&key=bc0418b57b2d4918819d3974ac1285d9";
        HttpUtil.sendOkHttpRequest(weatherUrl, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws
                IOException {
                String responseText = response.body().string();
                Weather weather = Utility.handleWeatherResponse(responseText);
                if (weather != null && "ok".equals(weather.status)) {
                    SharedPreferences.Editor editor = PreferenceManager.
                        getDefaultSharedPreferences(AutoUpdateService.this).
                            edit();
                    editor.putString("weather", responseText);
                    editor.apply();
                }
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }
        });
    }

    /**
     * 更新必应每日一图
     */
    private void updateBingPic() {
        String requestBingPic = "http://guolin.tech/api/bing_pic";
        HttpUtil.sendOkHttpRequest(requestBingPic, new Callback() {
            @Override
            public void onResponse(Call call, Response response) throws IOException {
                String bingPic = response.body().string();
                SharedPreferences.Editor editor = PreferenceManager.getDefault
                    SharedPreferences(AutoUpdateService.this).edit();
                editor.putString("bing_pic", bingPic);
                editor.apply();
            }

            @Override
            public void onFailure(Call call, IOException e) {
                e.printStackTrace();
            }
        });
    }
}
```

可以看到，在 `onStartCommand()` 方法中先是调用了 `updateWeather()` 方法来更新天气，然后调用了 `updateBingPic()` 方法来更新背景图片。这里我们将更新后的数据直接存储到 `SharedPreferences` 文件中就可以了，因为打开 `WeatherActivity` 的时候都会优先从 `SharedPreferences` 缓存中读取数据。

之后就是我们学习过的创建定时任务的技巧了，为了保证软件不会消耗过多的流量，这里将时间间隔设置为 8 小时，8 小时后 `AutoUpdateReceiver` 的 `onStartCommand()` 方法就会重新执行，这样也就实现后台定时更新的功能了。

不过，我们还需要在代码某处去激活 `AutoUpdateService` 这个服务才行。修改 `WeatherActivity` 中的代码，如下所示：

```
public class WeatherActivity extends AppCompatActivity {  
  
    ...  
  
    /**  
     * 处理并展示 Weather 实体类中的数据。  
     */  
    private void showWeatherInfo(Weather weather) {  
        if (weather != null && "ok".equals(weather.status)) {  
            ...  
            Intent intent = new Intent(this, AutoUpdateService.class);  
            startService(intent);  
        } else {  
            Toast.makeText(WeatherActivity.this, "获取天气信息失败", Toast.LENGTH_ SHORT).show();  
        }  
    }  
}
```

可以看到，这里在 `showWeather()` 方法的最后加入启动 `AutoUpdateService` 这个服务的代码，这样只要一旦选中了某个城市并成功更新天气之后，`AutoUpdateService` 就会一直在后台运行，并保证每 8 小时更新一次天气。

现在可以再提交一下代码：

```
git add .  
git commit -m "增加后台自动更新天气的功能。"  
git push origin master
```

14.8 修改图标和名称

目前的酷欧天气看起来还不太像是一个正式的软件，为什么呢？因为都还没有一个像样的图标呢。一直使用 `Android Studio` 自动生成的图标确实不太合适，是时候需要换一下了。

这里我事先准备好了一张图片来作为软件图标，由于我也不是搞美术的，因此图标设计得非常简单，如图 14.30 所示。



图 14.30 酷欧天气的图标

理论上讲，我们应该给这个图标提供几种不同分辨率的版本，然后分别放入到相应分辨率的 `mipmap` 目录下，这里简单起见，我就都使用同一张图了。将这张图片命名成 `logo.png`，放入到所有以 `mipmap` 开头的目录下，然后修改 `AndroidManifest.xml` 中的代码，如下所示：

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

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

    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        ...
    </application>

</manifest>
```

这里将 `<application>` 标签的 `android:icon` 属性指定成 `@mipmap/logo` 就可以修改程序图标了。接下来我们还需要修改一下程序的名称，打开 `res/values/string.xml` 文件，其中 `app_name` 对应的就是程序名称，将它修改成酷欧天气即可，如下所示：

```
<resources>
    <string name="app_name">酷欧天气</string>
</resources>
```

现在重新运行一遍程序，这时观察酷欧天气的桌面图标，如图 14.31 所示。



图 14.31 手机桌面图标

养成良好的习惯，仍然不要忘记提交代码。

```
git add .
git commit -m "修改程序图标和名称。"
git push origin master
```

这样我们就终于大功告成了！

14.9 你还可以做的事情

经过五个阶段的开发，酷欧天气已经是一个完善、成熟的软件了吗？嘿嘿，还差得远呢！现在的酷欧天气只能说是具备了一些最基本的功能，和那些商用的天气软件比起来还有很大的差

距，因此你仍然还有非常巨大的发挥空间来对它进行完善。

比如说以下功能是你可以考虑加入到酷欧天气中的。

- 增加设置选项，让用户选择是否允许后台自动更新天气，以及设定更新的频率。
- 优化软件界面，提供多套与天气对应的图片，让程序可以根据不同的天气自动切换背景图。
- 允许选择多个城市，可以同时观察多个城市的天气信息，不用来回切换。
- 提供更加完整的天气信息，目前我们只使用了和风天气返回的一小部分数据而已。

另外，由于酷欧天气的源码已经托管在了 GitHub 上面，如果你想在现有代码的基础上继续对这个项目进行完善，就可以使用 GitHub 的 Fork 功能。

首先登录你自己的 GitHub 账号，然后打开酷欧天气版本库的主页：<https://github.com/guolindev/coolweather>，这时在页面头部的最右侧会有一个 Fork 按钮，如图 14.32 所示。



图 14.32 GitHub Fork 按钮

点击一下 Fork 按钮就可以将酷欧天气这个项目复制一份到你的账号下，再使用 `git clone` 命令将它克隆到本地，然后你就可以在现有代码的基础上随心所欲地添加任何功能并提交了。

第 15 章

最后一步——将应用发布到 360 应用商店

应用已经开发出来了，下一步我们需要思考推广方面的工作。那么如何才能让更多的用户知道并使用我们的应用程序呢？在手机领域，最常见的做法就是将程序发布到某个应用商店中，这样用户就可以通过商店找到我们的应用程序，然后轻松地进行下载和安装。

说到应用商店，在 Android 领域真的可以称得上是百家争鸣，除了谷歌官方推出的 Google Play 之外，在中国还有像 360、豌豆荚、百度、应用宝等知名的应用商店。当然，这些商店所提供的功能都是比较类似的，发布应用的方法也大同小异，因此这里我们就只学习如何将应用发布到 360 应用商店，其他应用商店的发布方法相信你完全可以自己摸索出来。

15.1 生成正式签名的 APK 文件

之前我们一直都是通过 Android Studio 来将程序安装到手机上的，而它背后实际的工作流程是，Android Studio 会将程序代码打包成一个 APK 文件，然后将这个文件传输到手机上，最后再执行安装操作。Android 系统会将所有的 APK 文件识别为应用程序的安装包，类似于 Windows 系统上的 EXE 文件。

但并不是所有的 APK 文件都能成功安装到手机上，Android 系统要求只有签名后的 APK 文件才可以安装，因此我们还需要对生成的 APK 文件进行签名才行。那么你可能会有疑问了，直接通过 Android Studio 来运行程序的时候好像并没有进行过签名操作啊，为什么还能将程序安装到手机上呢？这是因为 Android Studio 使用了一个默认的 keystore 文件帮我们自动进行了签名。点击 Android Studio 右侧工具栏的 Gradle→项目名→:app→Tasks→android，双击 signingReport，结果如图 15.1 所示。

```
Variant: debug
Config: debug
Store: C:\Users\Administrator\.android\debug.keystore
```

图 15.1 查看默认的 keystore 文件

也就是说，我们所有通过 Android Studio 来运行的程序都是使用了这个 debug.keystore 文件来进行签名的。不过这仅仅适用于开发阶段而已，现在酷欧天气已经快要发布了，要使用一个正式的 keystore 文件来进行签名才行。下面我们就来学习一下，如何生成一个带有正式签名的 APK 文件。

15.1.1 使用 Android Studio 生成

先学习一下如何使用 Android Studio 来生成正式签名的 APK 文件。点击 Android Studio 导航栏上的 Build→Generate Signed APK，首次点击可能会提示让我们输入操作系统的密码，如图 15.2 所示。

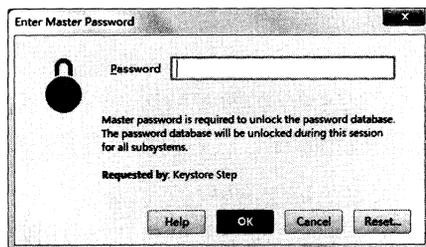


图 15.2 输入操作系统密码提示框

输入密码之后点击 OK，则会弹出如图 15.3 所示的创建签名 APK 对话框。

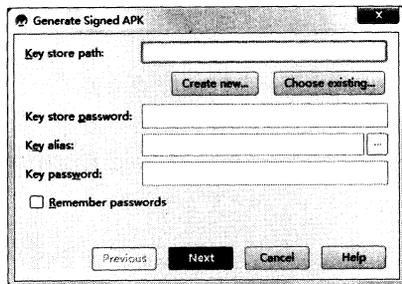


图 15.3 创建签名 APK 对话框

由于目前我们还没有一个正式的 keystore 文件，所以应该点击 Create new 按钮，然后会弹出一个新的对话框来让我们填写创建 keystore 文件所必要的信息。根据自己的实际情况进行填写就行了，如图 15.4 所示。

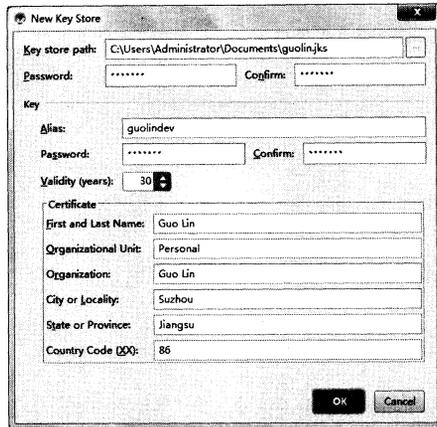


图 15.4 填写 keystore 文件信息

这里需要注意，在 Validity 那一栏填写的是 keystore 文件的有效时长，单位是年，一般建议时间可以填得长一些，比如我填了 30 年。然后点击 OK，这时我们刚才填写的信息会自动填充到创建签名 APK 对话框当中，如图 15.5 所示。

如果你希望以后都不用再输 keystore 的密码了，可以将 Remember passwords 选项勾上。然后点击 Next，这时就要选择 APK 文件的输出地址了，如图 15.6 所示。



图 15.5 信息自动填充完整

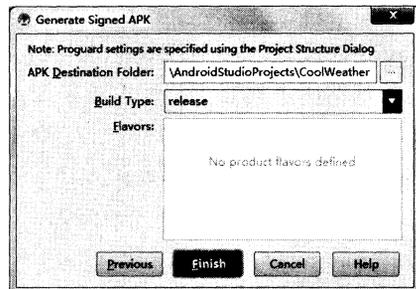


图 15.6 选择 APK 文件的输出地址

这里默认是将 APK 文件生成到项目的根目录下，我就不做修改了。现在点击 Finish，然后稍等一段时间，APK 文件就都会生成好了，并且会在右上角弹出一个如图 15.7 所示的提示。

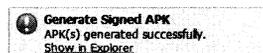


图 15.7 提示 APK 文件生成成功

我们点击提示上的 Show in Explorer 可以立刻查看生成的 APK 文件，如图 15.8 所示。



图 15.8 查看生成的 APK 文件

这里的 `app-release.apk` 就是带有正式签名的 APK 文件了。

15.1.2 使用 Gradle 生成

上一小节中我们使用了 Android Studio 提供的可视化工具来生成带有正式签名的 APK 文件，除此之外，Android Studio 其实还提供了另外一种方式——使用 Gradle 生成，下面我们就来学习一下。

Gradle 是一个非常先进的项目构建工具，在 Android Studio 中开发的所有项目都是使用它来构建的。在之前的项目中，我们也体验过了 Gradle 带来的很多便利之处，比如说当需要添加依赖库的时候不需要自己再去手动下载了，而是直接在 `dependencies` 闭包中添加一句引用声明就可以了。

不过这里我要提醒你一句，如果你想将 Gradle 完全精通的话，这个难度就比较大了。Gradle 的用法极为丰富，想要完全掌握它的用法，其复杂程度并不亚于学习一门新的语言（Gradle 是使用 Groovy 语言编写的）。而 Android 中主要只是使用 Gradle 来构建项目而已，因此这里我们掌握一些它的基本用法就好了，重点还是要放在功能开发上面，不要本末倒置了。当然，如果你对 Gradle 非常感兴趣，也可以到网上去查询它的更多用法。

下面我们开始学习如何使用 Gradle 来生成带有正式签名的 APK 文件。编辑 `app/build.gradle` 文件，在 `android` 闭包中添加如下内容：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
    signingConfigs {
        config {
            storeFile file('C:/Users/Administrator/Documents/guolin.jks')
            storePassword '1234567'
            keyAlias 'guolindev'
            keyPassword '1234567'
        }
    }
}
```

```
    }  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'),  
                'proguard-rules.pro'  
        }  
    }  
}
```

可以看到，这里在 android 闭包中添加了一个 signingConfigs 闭包，然后在 signingConfigs 闭包中又添加了一个 config 闭包。接着在 config 闭包中配置 keystore 文件的各种信息，storeFile 用于指定 keystore 文件的位置，storePassword 用于指定密码，keyAlias 用于指定别名，keyPassword 用于指定别名密码。

将签名信息都配置好了之后，接下来只需要在生成正式版 APK 的时候去应用这个配置就可以了。继续编辑 app/build.gradle 文件，如下所示：

```
android {  
    ...  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android.txt'),  
                'proguard-rules.pro'  
            signingConfig signingConfigs.config  
        }  
    }  
}
```

这里我们在 buildTypes 下面的 release 闭包中应用了刚才添加的签名配置，这样当生成正式版 APK 文件的时候就会自动使用我们刚才配置的签名信息来进行签名了。

现在 build.gradle 文件已经配置完成，那么我们如何才能生成 APK 文件呢？其实非常简单，Android Studio 中内置了很多的 Gradle Tasks，其中就包括了生成 APK 文件的 Task。点击右侧工具栏的 Gradle→项目名→:app→Tasks→build，如图 15.9 所示。

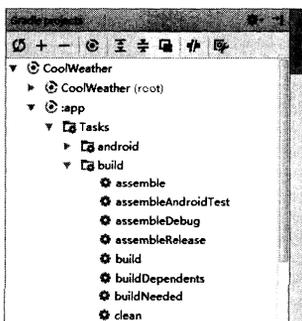


图 15.9 查看内置 Gradle Tasks

其中 `assembleDebug` 用于生成测试版的 APK 文件，`assembleRelease` 用于生成正式版的 APK 文件，`assemble` 用于同时生成测试版和正式版的 APK 文件。在生成 APK 之前，先要双击 `clean` 这个 Task 来清理一下当前项目，然后双击 `assembleRelease`，结果如图 15.10 所示。

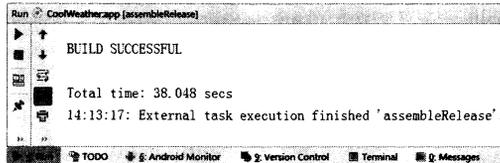


图 15.10 assembleRelease 执行成功

可以看到，这里提示我们 `BUILD SUCCESSFUL`，说明 `assembleRelease` 执行成功了。APK 文件会自动生成在 `app/build/outputs/apk` 目录下，如图 15.11 所示。

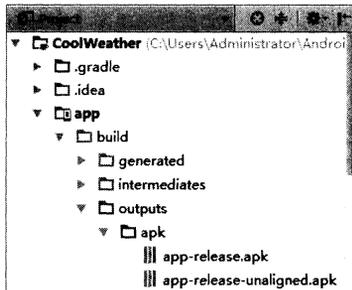


图 15.11 查看生成的 APK 文件

其中，`app-release.apk` 就是带有正式签名的 APK 文件了。另外还有一个 `app-release-unaligned.apk`，这是一个没有经过对齐的正式版 APK 文件，我们直接忽略它就可以了。

虽说现在 APK 文件已经成功生成了，不过还有一个小细节需要注意一下。目前 keystore 文件的所有信息都是以明文的形式直接配置在 `build.gradle` 中的，这样就不太安全。Android 推荐的做法是将这类敏感数据配置在一个独立的文件里面，然后再在 `build.gradle` 中去读取这些数据。

下面我们来按照这种方式实现。Android Studio 项目的根目录下有一个 `gradle.properties` 文件，它是专门用来配置全局键值对数据的，我们在 `gradle.properties` 文件中添加如下内容：

```
KEY_PATH=C:/Users/Administrator/Documents/guolin.jks
KEY_PASS=1234567
ALIAS_NAME=guolindev
ALIAS_PASS=1234567
```

可以看到，这里将 keystore 文件的各种信息以键值对的形式进行了配置，然后我们在 `build.gradle` 中去读取这些数据就可以了。编辑 `app/build.gradle` 文件，如下所示：

```
android {
    ...
}
```

```
signingConfigs {
    config {
        storeFile file(KEY_PATH)
        storePassword KEY_PASS
        keyAlias ALIAS_NAME
        keyPassword ALIAS_PASS
    }
}
...
}
```

这里只需要将原来的明文配置改成相应的键值，一切就完工了。这样直接查看 `build.gradle` 文件是无法看到 `keystore` 文件的各种信息的，只有查看 `gradle.properties` 文件才能看得到。然后我们只需要将 `gradle.properties` 文件保护好就行了，比如说将它从 `Git` 版本控制中排除。这样 `gradle.properties` 文件就只会保留在本地，从而也就不担心 `keystore` 文件的信息会泄漏了。

15.1.3 生成多渠道 APK 文件

现在你已经掌握了两种生成带有正式签名的 APK 文件的方式，从简易程度上来讲，两种方式差不多，基本都还是比较简单的，选择使用哪一种全凭你自己的喜好。

现在 APK 文件已经生成好了，可能在大多数情况下，我们都只需要一个 APK 文件就足够了，不过本小节中我们再来讨论一种比较特殊的情况——生成多渠道 APK 文件。

在本章的开头就已经提到过，目前 `Android` 领域的应用商店非常多，不像苹果只有一个 `App Store`。当然我们完全可以使用同一个 `APK` 文件来上架不同的应用商店，但是如果你有一些特殊需求的话，比如说针对不同的应用商店渠道来定制不同的界面，这就比较头疼了。

传统情况下，开发这种差异性需求非常痛苦，通常需要维护多份代码版本，然后逐个打成相应渠道的 `APK` 文件。一旦有任何功能变更就苦不堪言，因为每份代码版本里面都需要逐个修改一遍。

幸运的是，现在 `Android Studio` 提供了一种非常方便的方法来应对这种差异性需求，极大地解决了之前版本维护困难的问题，下面我们就来学习一下。

比如说这里我们准备生成 `360` 和 `百度` 两个渠道的 `APK` 文件，那么修改 `app/build.gradle` 文件，如下所示：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
}
```

```
    }  
    productFlavors {  
        qihoo {  
            applicationId "com.coolweather.android.qihoo"  
        }  
        baidu {  
            applicationId "com.coolweather.android.baidu"  
        }  
    }  
    ...  
}
```

可以看到，这里添加了一个 `productFlavors` 闭包，然后在该闭包中添加所有的渠道配置就可以了。注意 Gradle 中的配置规定不能以数字开头，因此这里我将 360 的渠道名配置成了 `qihoo`。渠道名的闭包中可以覆写 `defaultConfig` 中的任何一个属性，比如说这里将 `applicationId` 属性进行了覆写，那么最终生成的各渠道 APK 文件的包名也将各不相同。

接下来我们开始针对不同渠道编写差异性需求。在 `app/src` 目录下（`main` 的平级目录）新建一个 `baidu` 目录，然后在 `baidu` 目录下再新建 `java` 和 `res` 这两个目录，如图 15.12 所示。

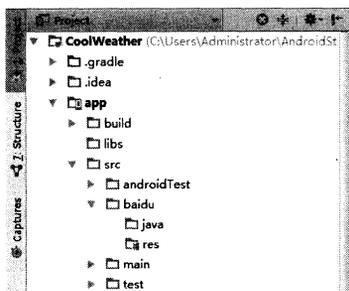


图 15.12 创建渠道专属目录

这样我们就可以在这里编写百度渠道特有的功能了，`java` 目录用于存放代码，`res` 目录用于存放资源，如果需要覆写 `AndroidManifest` 文件中的内容，还可以在 `baidu` 目录下再新建一个 `AndroidManifest.xml` 文件。

当然，实际上我们并没有什么渠道差异性的需求，因此这里也只是为了演示一下，我们就给不同渠道的 APK 起一个不同的应用名吧。

应用名之前是定义在 `main/res/values/string.xml` 文件中的，那么我们在 `baidu` 目录下也建立一个相同的目录结构，然后将 `baidu/res/values/string.xml` 中的内容进行如下修改：

```
<resources>  
    <string name="app_name">酷欧百度版</string>  
</resources>
```

这样百度渠道的 APK 就会使用 `baidu/res/values/string.xml` 中定义的应用名来覆盖原有的应用名。同样的道理，我们再新建一个 `qihoo` 目录，然后在 `qihoo` 目录下也建立相同的目录结构，并

将 string.xml 中的内容进行如下修改：

```
<resources>  
    <string name="app_name">酷欧 360 版</string>  
</resources>
```

这样我们就以一个简单的示例实现渠道差异性需求了，下面开始来生成多渠道的 APK 文件。观察右侧工具栏的 Gradle Tasks 列表，你会发现里面多出了几个新的 Task，如图 15.13 所示。

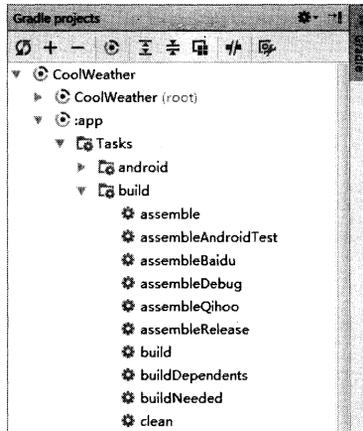


图 15.13 查看新的 Task

其中，如果你只想生成百度渠道的 APK 文件，那么就执行 assembleBaidu；如果你只想生成 360 渠道的 APK 文件，那么就执行 assembleQihoo；如果你想一次性生成所有渠道的 APK 文件，那么就还是执行 assembleRelease。

除了使用 Gradle 的方式生成之外，使用 Android Studio 提供的可视化工具也是能生成多渠道 APK 文件的，如图 15.14 所示。

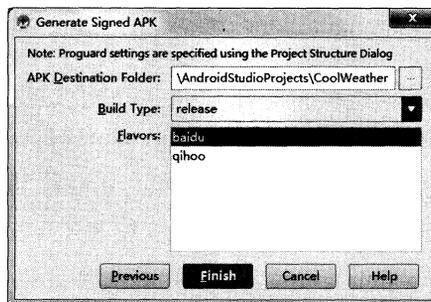


图 15.14 使用可视化工具生成多渠道 APK

这里我们可以选择是生成百度渠道的 APK 文件，还是生成 360 渠道的 APK 文件，如果你想

一次性生成多个渠道的 APK 文件，按住 CTRL 键就可以进行多选了。

接下来我们可以通过 `adb install` 命令将生成好的 APK 文件安装到模拟器上，如图 15.15 所示。



图 15.15 将生成的 APK 安装到模拟器上

`adb install` 命令的后面加上 APK 文件的路径，就可以将该 APK 文件安装到模拟器上了。我们使用同样的方法将百度和 360 这两个渠道的 APK 文件都安装到模拟器上，结果如图 15.16 所示。



图 15.16 模拟器上的安装结果

可以看到，目前模拟器上有 3 个版本的酷欧天气，这是由于之前我们在 `productFlavors` 中覆盖了各渠道的 `applicationId` 属性，保证每个 APK 文件的包名都不相同，因而它们才能安装到同一个设备上。另外，从应用名上来看，渠道差异性开发工作也顺利完成了。

不过，上面的例子只是为了演示生成多渠道 APK 功能而特意编写的，实际上我们并没有这个需求。现在将 `productFlavors` 闭包删除，恢复成之前的 APK 文件，我们准备进行上架操作。

15.2 申请 360 开发者账号

目前，酷欧天气的 APK 安装包已经准备好了，但如果想要把它发布到 360 应用商店，还需要去申请一个 360 开发者账号才行，申请地址是：<http://dev.360.cn>。

打开该网页，在页面顶部有登录和注册按钮。如果你还没有 360 账号，则需要在这里注册一个新的账号，如果你之前已经有 360 账号了，那么直接登录就可以了。

登录成功之后打开 <http://dev.360.cn/mod/developer> 这个网址，来申请成为开发者，如图 15.17 所示。

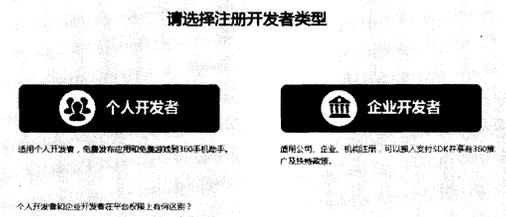


图 15.17 选择注册开发者类型

这里可以选择是申请成为个人开发者还是企业开发者。很显然，我们是以个人的身份来发布应用的，那么点击个人开发者就可以了。

接下来需要填写一些基本信息和联系方式，如图 15.18 所示。

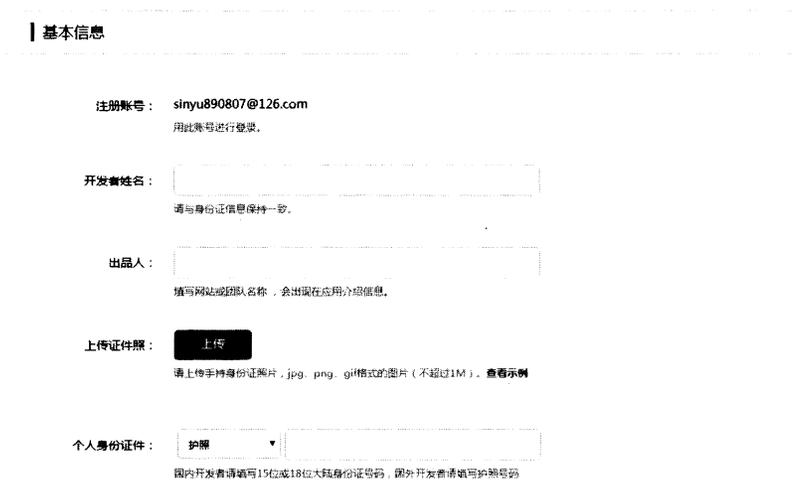


图 15.18 填基本信息和联系方式

填写完基本信息之后向下滚动继续填写联系方式，全部填写完成之后，点击屏幕最下方的“同意并注册开发者”按钮来完成注册，如图 15.19 所示。

我已阅读并同意《360移动开放平台服务条款》

同意并注册开发者

图 15.19 完成开发者注册

这样你就成功成为一名 360 开发者了！

15.3 发布应用程序

接下来我们开始发布酷欧天气这个应用，还是在浏览器访问地址：<http://dev.360.cn>，你会在界面上看到如图 15.20 所示的内容。

然后点击软件发布，就会显示如图 15.21 所示的界面。



图 15.20 软件发布和游戏发布

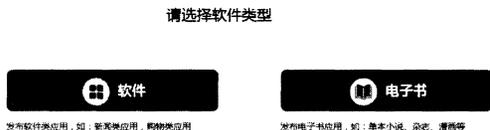


图 15.21 选择软件类型

我们需要选择是发布软件类应用还是电子书类应用，这里点击软件。接下来会弹出一个新的界面让我们上传 APK 以及填写应用信息。首先来上传 APK 吧，点击上传按钮，选择带有正式签名的 APK 文件，然后就会自动开始上传了，上传完成之后会显示如图 15.22 所示的界面。

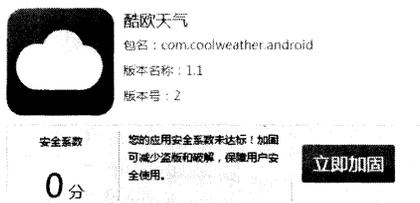


图 15.22 上传 APK 完成

这个界面提醒我们，目前应用的安全系数较低，建议对 APK 进行加固。实际上这个是 360 应用商店的特殊需求，并不是所有应用商店都要求进行加固的。但是我们还是得按照它的要求来修改，不然审核可能会不通过。

这里点击立即加固按钮，360 会帮忙我们将原 APK 文件进行加固，并生成一个新的 APK 文件，如图 15.23 所示。

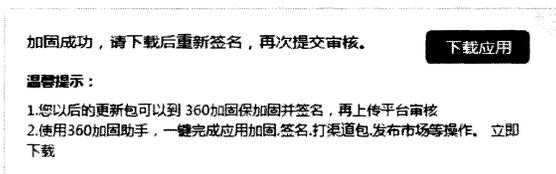


图 15.23 加固成功提示

不过这个加固后的 APK 文件是没有经过签名的，也就是说我们还需要将它下载下来，然后手动进行签名才行。

点击下载应用按钮，先将加固后的 APK 文件下载下来。接下来的工作就有点烦琐了，因为 Android Studio 中并没有提供对一个未签名的 APK 直接进行签名的功能，因此我们只能通过最原始的方式，使用 `jarsigner` 命令来进行签名。

在命令行界面按照以下格式输入签名命令：

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore [keystore 文件路径]
-storepass [keystore 文件密码] [待签名 APK 路径] [keystore 文件别名]
```

将[]中的描述替换成 keystore 文件的具体信息就能签名成功了，注意[]符号是不需要的。接着我们将签名后的 APK 文件重新上传就可以了。

APK 上传成功之后，接下来需要选择应用的分类，如图 15.24 所示。

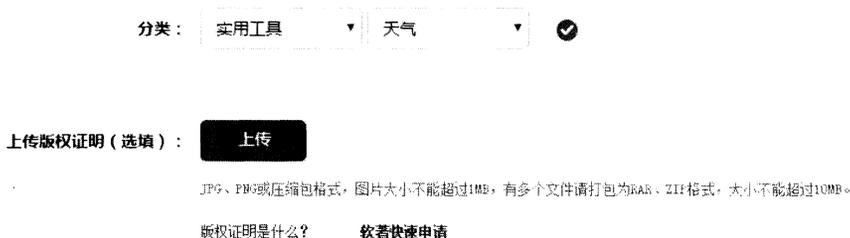


图 15.24 选择应用分类

这里我们将应用分类选择成实用工具→天气。下面还有一个上传版权证明的选项，这是一个选填项，我们直接忽略就可以了。

接着向下滚动网页，设置支持的语言以及资费类型，如图 15.25 所示。



图 15.25 设置支持语言和资费类型

继续滚动网页，下面需要填写应用简介以及当前版本介绍，如图 15.26 所示。

应用简介： 酷欧天气是一款基于Android端开源的天气预报软件，具备查看全国的省市县、查询任意城市天气、自由切换城市、手动更新天气、后台自动更新天气等功能。酷欧天气中的天气数据由和风天气提供，背景图片由必应提供，代码遵循Apache v2 License开源协议。本软件主要作为学习和交流使用。

50~1500字，请向用户介绍一下你的应用。

当前版本介绍： 酷欧天气是一款基于Android端开源的天气预报软件，具备查看全国的省市县、查询任意城市天气、自由切换城市、手动更新天气、后台自动更新天气等功能。

50~400字符，请向用户介绍当前应用版本及更新内容。

图 15.26 填写应用简介和当前版本介绍

在版本介绍的下面，360 还要求填写一项隐私权限说明，由于酷欧天气只申请了一个网络权限，因此没什么需要说明的，我们直接忽略这一项就可以了。

继续向下滚动网页，接下来需要上传一张高分辨率的应用图标，图标要求是 512 × 512 像素的 PNG 格式图片，如图 15.27 所示。

应用图标： 要求与安装包中图标一致。尺寸：512×512PX，圆角半径范围：70PX，图片格式：PNG。 [请按照右侧示例上传。](#)

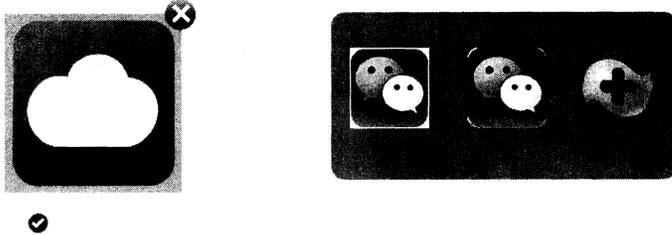


图 15.27 上传高分辨率的应用图标

上传好了图标，我们还需要提供 5 张酷欧天气的屏幕截图，点击上传截图按钮，然后选择准备好的图片即可，如图 15.28 所示。

应用截图： 请上传4~5张截图（尺寸保持一致），支持JPG、PNG格式。截图尺寸要求：不小于800×480（480×800），单张图片不能超过3M。请去除截图中的顶部

通知栏。 [查看示例](#)

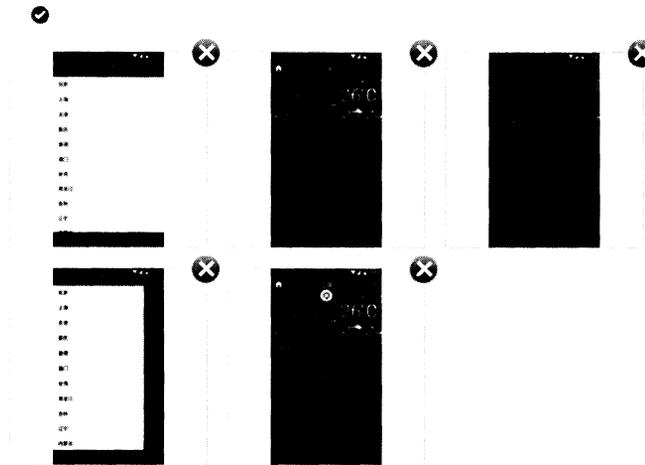


图 15.28 上传屏幕截图

继续向下滚动，还有一个审核辅助说明的选填项，我们也直接忽略就可以了。最后就是一些额外的定制选项，如图 15.29 所示。

是否进行云测试： 是 否

由 [Testin云测](#) 提供专业的应用机型测试，确保客户端自动为您的应用程序进行测试，并发送测试报告。

发布时间： 审核后立即发布 定时发布

提交审核

图 15.29 额外的辅助选项

这里我们选择不进行云测试，并在审核后立即发布。

激动人心的时刻终于到了，现在点击一下提交审核按钮就可以将酷欧天气发布到 360 应用商店了，这时会显示如图 15.30 所示的提示。

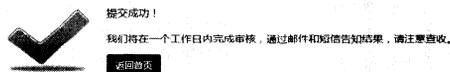


图 15.30 提交成功提示

由于 360 会对我们的应用程序进行审核，接下来又进入了等待当中。不过还好，根据提示来看，这次也许不需要等太久。

果不其然，过了几个小时之后在 360 手机助手上搜索酷欧天气关键字，就可以看到这个应用已经成功上线了，如图 15.31 所示。

点击进去可以查看应用的详情，如图 15.32 所示。



图 15.31 搜索酷欧天气关键字



图 15.32 查看应用详情

到了这里，我们就将应用程序的发布工作全部完成了，之后你应该尽可能地多为你的应用进行宣传，因为用户越多，你能得到的回报就越大。那么如何才能从我们辛辛苦苦编写的程序中得到回报呢？方式有很多种，其中较为常见的做法就是通过广告来进行盈利，因此下一节我们就学习一下，如何在应用程序中嵌入广告。

15.4 嵌入广告进行盈利

谷歌充分考虑到了可以在 Android 应用程序中嵌入广告来让开发者获得收入，因此早早地就收购了 AdMob 公司。AdMob 创立于 2006 年，是全球最早致力于在移动设备上提供广告服务的公司之一，如今成为了谷歌的子公司，AdMob 的广告更加适合在 Android 系统以及 Google Play 上面进行投放。

不过对于国内开发者来说，AdMob 可能就不是那么适合了。因为 AdMob 平台上的广告大多都是英文的，中文广告数有限，并且将 AdMob 账户中的钱提取到银行账户中也比较麻烦，因此这里我们就不准备使用 AdMob 了，而是将眼光放在一些国内的移动广告平台上面。在国内的这一领域，做得比较好的移动广告平台也不少，其中我个人认为腾讯广告联盟（原广点通）特别专业，因此我们就选择它来为酷欧天气提供广告服务吧。

15.4.1 注册腾讯广告联盟账号

下面开始动手，首先第一步我们需要注册一个腾讯广告联盟的账号，注册地址为：<http://e.qq>。

com/dev/index.html。

打开该网页，选择使用QQ号登录，然后就会自动跳转到腾讯广告联盟的注册界面，如图 15.33 所示。图 15.33 中的所有内容都是必填项，我们按照实际情况来填写就可以了，填写完成之后点击下一步，如图 15.34 所示。

会员类型: 个人 请选择正确的会员类型, 否则无法获取收益

姓名:

身份证号码:

联系地址: 北京市 东城

手机号码:

电子邮箱:

电子邮箱验证码: 获取邮箱验证码

注册来源:

下一步 取消

图 15.33 填写个人信息

收款方: 郭霖

开户银行: 请选择

开户行所在地: 北京市 东城

支行名称:

银行账号:

账号确认:

银行卡正面照片: 支持格式仅限jpg,png,大小2M以内

未选择任何文件

上一步 下一步

图 15.34 填写银行卡信息

由于腾讯广告联盟涉及提现服务，因此我们还需要填写银行卡信息，并上传银行卡照片。填写完成之后继续点击下一步，如图 15.35 所示。

最后，将你的身份证正反面照片上传，点击提交按钮，就能提交审核了，如图 15.36 所示。

身份证正面照片: 未选择任何文件

身份证反面照片: 未选择任何文件

支持格式仅限jpg,png,大小2M以内

同意 《腾讯广告通移动联盟开发者协议》

上一步 提交

图 15.35 上传身份证照片

会员注册

您的会员注册已提交成功, 我们会在1个工作日内完成审核, 请耐心等待。

关闭

图 15.36 提交审核

只要你前面填写的内容都是真实有效的，审核一般都会很快通过，这里我们只需要耐心等待几个小时就好了。

15.4.2 新建媒体和广告位

审核通过之后，我们就可以进入到腾讯广告联盟的后台，开始给酷欧天气添加广告了。首先需要进入媒体管理界面，点击新建媒体按钮，这时会显示一个页面来让你填写应用的相关信息，我们根据提示一一填好即可，如图 15.37 所示。

系统平台： Android程序 iOS程序

媒体名称：

关键词：

媒体类别：

媒体简介：

程序主包名：

媒体状态： 已上架 未上架

详情页地址：

图 15.37 填写应用的相关信息

注意这里需要填写一个详情页地址，也就是酷欧天气在 360 应用商店上的详情页地址。打开 <http://zhushou.360.cn>，在搜索框上输入“酷欧天气”，就能找到该地址了。

填写完成之后点击下一步，接下来需要下载 SDK，如图 15.38 所示。

点击 Android SDK 下载按钮，先把 SDK 下载下来，我们稍后就会进行接入。继续点击下一步，如图 15.39 所示。

媒体名称：

媒体类型： Android程序

应用ID：

图 15.38 下载 SDK

媒体名称：

媒体类型： Android程序

应用ID：

应用程序： 上传 输入下载URL

图 15.39 完成媒体创建

这里要求填入一个 APK 的下载的地址，我们直接就填入图 15.37 当中的详情页地址就可以

了。点击完成按钮，现在又会进入到审核等待当中。

为什么新建媒体也需要进行审核呢？这是因为腾讯为了防止某些开发者在垃圾软件上面投放广告，因此要求开发者必须提交应用程序的 APK 文件进行审核，只有审核通过的应用才允许进行广告投放。那么我们只能继续等待。审核通过之后，在媒体管理界面查看新建媒体的状态，如图 15.40 所示。

应用ID	媒体名称	联盟开通状态	业务状态	系统平台	操作
1105585573	酷欧天气	已开通	正常	Android	修改 新建广告位

图 15.40 查看新建媒体的状态

可以看到，联盟开通状态显示已开通，业务状态显示正常，说明新建的媒体已经通过审核了。注意这里还自动生成了一个应用 ID，我们稍后就会用到。

现在点击新建广告位，就可以来创建一个广告位了，如图 15.41 所示。

新建广告位 ✕

媒体选择:

广告位名称:

广告位类型: Banner广告 应用墙 插屏广告 开屏广告

屏蔽行业: 成人用品类 医疗科室类 减肥类 心理健康类 星座算命类 P2P网贷平台

如果对于某些敏感行业素材(例如成人用品)过于敏感，也可以进行行业广告屏蔽

图 15.41 新建广告位

首先要输入广告位的名称，然后选择广告位的类型。腾讯广告联盟支持 Banner、应用墙、插屏和开屏这 4 种广告类型，具体每种广告类型的区别你可以通过查阅文档进行了解，这里我们选择开屏广告。接下来还可以对一些敏感行业的广告进行屏蔽，选择完成之后点击创建按钮完成广告位创建。

现在进入到广告位管理界面，就能查看到我们刚刚新建的广告位了，如图 15.42 所示。

名称&ID	广告位类型	所属应用&ID	业务状态	广告位状态	操作
启动广告 4010212448179536	开屏	酷欧天气 1105585573	正常	<input checked="" type="checkbox"/> 启用中	修改

图 15.42 查看新建广告位

其中，4010212448179536 是广告位 ID，1105585573 是应用 ID。有了这两个数据之后，我们就可以开始接入广告 SDK 了。

15.4.3 接入广告 SDK

首先将刚才下载的广告 SDK 压缩包解压，里面的内容非常简单，如图 15.43 所示。

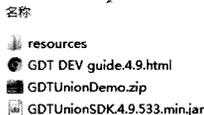


图 15.43 广告 SDK 压缩包中的内容

其中 resources 文件夹中放的是一些资源图片，我们使用不到。GDT DEV guide.4.9.html 是广告 SDK 的对接文档，GDTUnionDemo.zip 是广告 SDK 的对接示例，GDTUnionSDK.4.9.533.min.jar 则是广告 SDK 中最主要的一个 Jar 包文件了。

由于腾讯广告 SDK 中的功能还是挺多的，这里不可能面面俱到，将每一个功能都进行详细地讲解。因此我准备只讲解开屏广告这一种类型的广告用法，剩下的其他功能你可以通过阅读文档来进行学习。

我们先将 GDTUnionSDK.4.9.533.min.jar 复制到 app/libs 目录当中，并点击一下 Android Studio 顶部工具栏中的 Sync 按钮完成同步。

接着在 AndroidManifest.xml 中声明以下权限，其中网络访问权限是之前声明过的，不需要声明两遍。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

注意，其中 READ_PHONE_STATE、ACCESS_COARSE_LOCATION 和 WRITE_EXTERNAL_STORAGE 这 3 个权限是危险权限，因此我们待会还需要进行运行时权限处理。

接下来在<application>标签中添加如下内容：

```
<activity
    android:name="com.qq.e.ads.ADAActivity"
    android:configChanges="keyboard|keyboardHidden|orientation|screenSize" />
<service
    android:name="com.qq.e.comm.DownloadService"
    android:exported="false" />
```

这样就将配置工作完成了。

然后我们还需要创建一个用于显示开屏广告的活动，右击 com.coolweather.android 包→New→Activity→Empty Activity，创建一个 SplashActivity，并将布局名指定成 activity_splash.xml。修改 activity_splash.xml 中的代码，如下所示：

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

</RelativeLayout>
```

这里只有一个空的 `RelativeLayout`，我们并不需要在 `RelativeLayout` 当中放入什么内容，但是必须给它定义一个 `id`。

接着修改 `SplashActivity` 中的代码，如下所示：

```
public class SplashActivity extends AppCompatActivity {

    private RelativeLayout container;

    /**
     * 用于判断是否可以跳过广告，进入 MainActivity
     */
    private boolean canJump;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        container = (RelativeLayout) findViewById(R.id.container);
        // 进行运行时权限处理
        List<String> permissionList = new ArrayList<>();
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_PHONE_
            STATE) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.READ_PHONE_STATE);
        }
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_
            COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.ACCESS_COARSE_LOCATION);
        }
        if (ContextCompat.checkSelfPermission(this, Manifest.permission.WRITE_
            EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            permissionList.add(Manifest.permission.WRITE_EXTERNAL_STORAGE);
        }
        if (!permissionList.isEmpty()) {
            String [] permissions = permissionList.toArray(new String[permissionList.
                size()]);
            ActivityCompat.requestPermissions(this, permissions, 1);
        } else {
            requestAds();
        }
    }

    /**
     * 请求开屏广告
     */
    private void requestAds() {
        String appId = "1105585573";
```

```
String adId = "4010212448179536";
new SplashAD(this, container, appId, adId, new SplashADListener() {
    @Override
    public void onADDismissed() {
        // 广告显示完毕
        forward();
    }

    @Override
    public void onNoAD(int i) {
        // 广告加载失败
        forward();
    }

    @Override
    public void onADPresent() {
        // 广告加载成功
    }

    @Override
    public void onADClicked() {
        // 广告被点击
    }
});

@Override
protected void onPause() {
    super.onPause();
    canJump = false;
}

@Override
protected void onResume() {
    super.onResume();
    if (canJump) {
        forward();
    }
    canJump = true;
}

private void forward() {
    if (canJump) {
        // 跳转到 MainActivity
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
        finish();
    } else {
        canJump = true;
    }
}

@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
    int[] grantResults) {
```

```
switch (requestCode) {
    case 1:
        if (grantResults.length > 0) {
            for (int result : grantResults) {
                if (result != PackageManager.PERMISSION_GRANTED) {
                    Toast.makeText(this, "必须同意所有权限才能使用本程序",
                        Toast.LENGTH_SHORT).show();
                    finish();
                    return;
                }
            }
            requestAds();
        } else {
            Toast.makeText(this, "发生未知错误", Toast.LENGTH_SHORT).show();
            finish();
        }
        break;
    default:
}
}
```

可以看到，在 `onCreate()` 方法中，我们先是获取到了 `RelativeLayout` 的实例，紧接着就开始进行运行时权限处理。由于这里也是需要运行时一次性申请多个权限，因此采用了和 11.3.2 小节同样的写法，相信你一定不会陌生吧。

当用户同意了所有的权限申请之后，就会调用 `requestAds()` 方法来请求广告数据。在 `requestAds()` 方法中我们先是定义了 `appId` 和 `adId` 这两个变量，它们的值就是在腾讯广告联盟后台生成的应用 ID 和广告位 ID，然后创建 `SplashAD` 的实例来获取广告数据。`SplashAD` 的构造函数接收 5 个参数，第 1 个参数是当前活动的实例，第 2 个参数是 `RelativeLayout` 的实例，第 3 个参数是应用 ID，第 4 个参数是广告位 ID，相信前 4 个参数都没什么需要解释的。第 5 个参数是一个 `SplashADListener` 的实例，用于监听广告数据的回调。其中 `onADDismissed()` 方法会在广告显示完毕时回调，`onNoAD()` 方法会在广告加载失败时回调，`onADPresent()` 方法会在广告加载成功时回调，`onADClicked()` 方法会在广告被点击时回调。当广告显示完毕或者广告加载失败时，我们调用 `forward()` 方法跳转到 `MainActivity`，并将当前活动关闭即可。

另外注意这里还使用了一个 `canJump` 变量用于对活动跳转进行控制。这是因为如果用户点击了广告，会启动一个新的活动来展示广告的详细内容，这个时候即使回调了 `onADDismissed()` 方法，显然也不应该启动 `MainActivity`，因此我们在 `onPause()` 方法中将 `canJump` 设置成了 `false`。然后在 `forward()` 方法中发现 `canJump` 是 `false`，因此不会进行跳转，但是会将 `canJump` 设置成 `true`。最后，当用户看完了广告回到 `SplashActivity` 时，`onResume()` 方法将会执行，这个时候发现 `canJump` 是 `true`，因此就会调用 `forward()` 方法来启动 `MainActivity`。

整体流程大概就是这个样子了，接下来我们还需要将主活动设置成 `SplashActivity` 而不再是 `MainActivity`，否则广告界面将无法得到展示。修改 `AndroidManifest.xml` 中的代码，如下所示：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.coolweather.android">
    ...
    <application
        android:name="org.litepal.LitePalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/logo"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
        </activity>
        <activity android:name=".SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        ...
    </application>
</manifest>
```

这样我们就将广告 SDK 全部对接完成了，现在可以重新运行一下程序来看一看效果。不过需要注意，广告在模拟器上是不会显示的，我们要用真正的手机测试才行。程序启动后首先会弹出运行时权限的申请对话框，全部都点击允许之后就能看到广告数据了，如图 15.44 所示。



图 15.44 显示广告数据

开屏广告会持续 5 秒钟时间，然后就会自动跳转到 MainActivity 中，后面的流程就和之前是完全一样的了。

15.4.4 重新发布应用程序

现在我们已经成功在酷欧天气中接入了广告功能，那么是时候将这个新版本更新到 360 应用商店上了。

由于即将发布的会是新一版的酷欧天气，因此在生成安装包之前还需要修改一下应用程序的版本号信息。编辑 `app/build.gradle` 文件，如下所示：

```
android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"
    defaultConfig {
        applicationId "com.coolweather.android"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 2
        versionName "1.1"
    }
    ...
}
```

可以看到，这里将 `versionCode` 改成了 2，`versionName` 改成了 1.1。需要注意的是，每个版本的 `versionCode` 和 `versionName` 都不能和其他版本相同，且新版应用的版本号必须大于老版应用的版本号。

接下来我们就可以使用在 15.1 节学习的技术来生成新的 APK 文件，具体的步骤就不再重复介绍了，最终会生成一个新的 `app-release.apk` 文件，下面我们将它重新发布到 360 应用商店。

打开 360 开发者后台的管理中心页面，然后点击酷欧天气的更新管理按钮，如图 15.45 所示。



图 15.45 更新酷欧天气

点击编辑与更新按钮，就能上传新的 APK 文件了。注意上传之后仍然会提醒应用的安全系数较低，我们只需要使用和之前同样的方式进行加固就可以了。

另外，由于新版的酷欧天气中增加了一些敏感隐私权限，因此我们还需要在这一项上面做出说明，如图 15.46 所示。

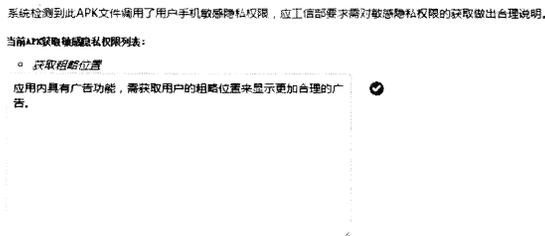


图 15.46 对敏感隐私权限进行说明

现在只需要点击页面最下方的提交审核按钮，新版本的酷欧天气就发布成功了，当然还需要通过 360 的审核才行。以后每当有用户观看或点击了应用程序中的广告时，我们就能真正地得到收益。在腾讯广告联盟的后台管理界面可以查看到每天的收益情况，如图 15.47 所示。

昨日预估收益	昨日收益	本月累计收益	撰写总收益
0.01	0.02	0.02	0.02

图 15.47 查看广告收益情况

你的应用越成功，所获得的广告收益也会越多，因此赶快去编写更多优秀的应用程序来赚更多的钱吧，相信通过整本书的学习，你已经有足够的能力做到了！

15.5 结束语

就这样，本书所有的内容你都学完了，现在你已经成功毕业，并且成为了一名合格的 Android 开发者。但是如果想要成为一名出色的 Android 开发者，光靠本书中的这些理论知识以及少量的实践还是不够的，你需要真正步入到工作岗位当中，通过更多的项目实战来不断地历练和提升自己。

唠叨了整本书的话，但是到了最后却不知道该说点什么好，我不想说我能教你的就只有这些了，因为实际上我想教你或者和你一起探讨的内容还有很多很多，不过限于篇幅的原因，本书的内容就只能到此为止了。但我会长期在博客和微信公众号上面分享更多 Android 相关的技术文章，如果感兴趣的话，可以到我的博客和公众号中继续学习。当然，如果对本书中的内容有疑问，也可以到博客或公众号中给我留言，我的博客地址如下：

<http://guolin.tech>

扫一扫下方二维码即可关注我的公众号：



好了，就到这里吧，祝你未来的 Android 之旅都能愉快。