Broadview®
www.broadview.com.cn

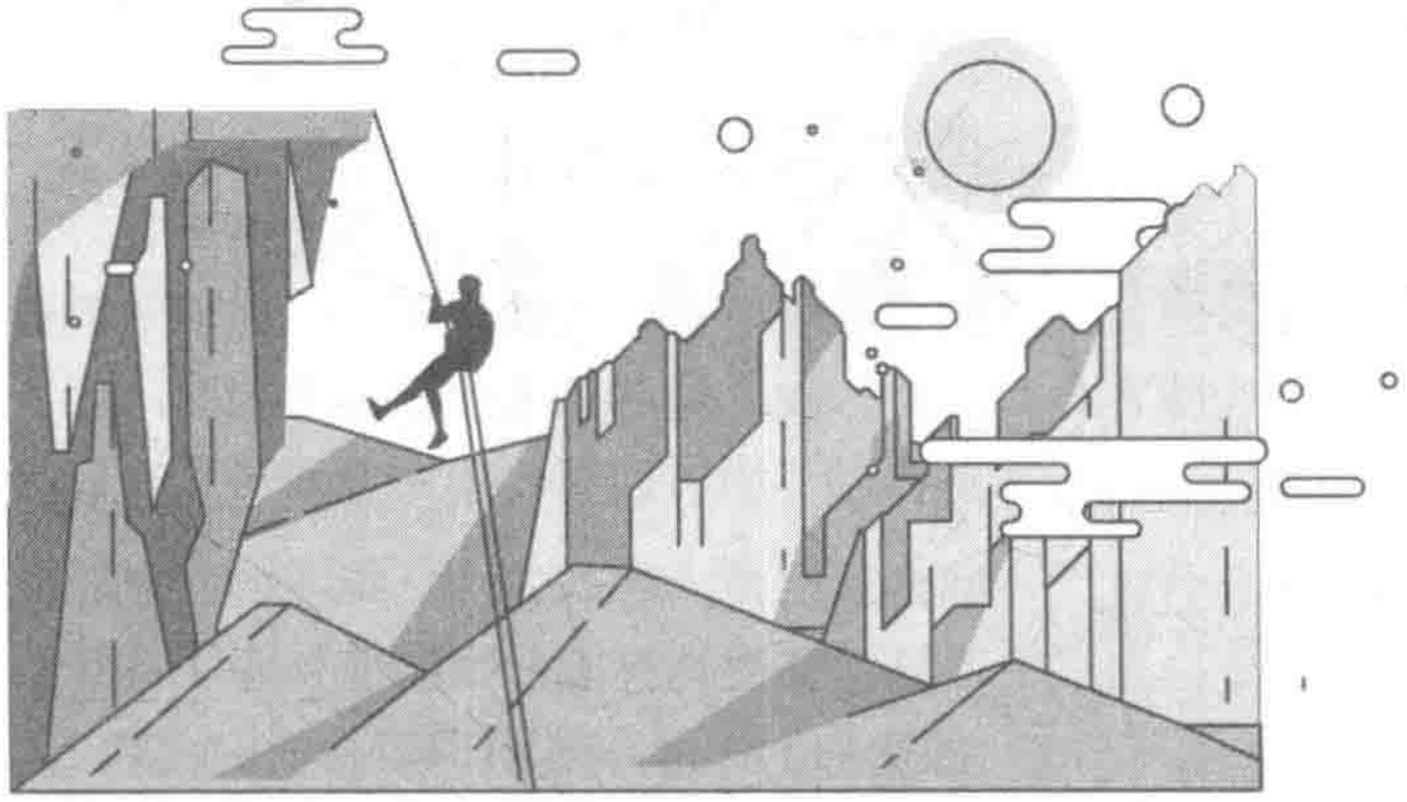
高级Android开发强化实战

王辰龙 编著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

高级Android开发强化实战

王辰龙 编著



電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书收集了约20个关于高级Android开发的进阶实例，这些实例都是对在日常开发中遇到的问题的抽象，涉及整个Android开发的各个技术栈。本书从常见的问题入手，引导读者逐步地掌握进阶的各个实例，同时提供分析和解决问题的思考过程，寻求最优方案。本书的内容分为以下几个部分。

进阶基础：通过管中窥豹，剖析Activity和View的一些基本概念，展示源码分析的常见方法；高阶控件：讲解MD的两个复合布局和约束布局，介绍高级控件的开发流程；项目架构：架构是项目的骨骼，该部分介绍主流的MVP系列和Flux架构；响应式编程：解析响应式编程三剑客RxJava+Dagger+Retrofit的不同特性组合使用方法；功能与动画：列举若干实际开发中的经典实例，包含功能定制和页面动画等；Kotlin与SVG：讲解Kotlin编程语言和SVG图像技术的若干开发技巧；测试与优化：介绍自动化测试框架的设计方法，以及优化应用的常用工具。

通过对本书的学习，读者可以极大地提高Android开发的工程能力，从而成为一名合格的高级Android工程师，不仅在理论上有所提升，在实践中也能直接应用。高级Android工程师通过对本书的学习也能完善知识体系和技术栈。

图书在版编目（CIP）数据

高级 Android 开发强化实战 / 王辰龙编著. —北京：电子工业出版社，2018.7
ISBN 978-7-121-34298-1

Ⅰ. ①高… Ⅱ. ①王… 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（2018）第 111022 号

策划编辑：张国霞
责任编辑：宋亚东
印　　刷：三河市兴达印务有限公司
装　　订：三河市兴达印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　印张：18.5　字数：413 千字
版　　次：2018 年 7 月第 1 版
印　　次：2018 年 7 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前　言

在编程之余，有时候我就在想，什么样的程序员属于高级程序员呢？或者说，高级程序员有哪些特性呢？工作年限一定不是一个关键的指标，许多工作多年的程序员依然写不出优雅的程序。无论是在 Android 开发还是其他领域，高级程序员一定是勤奋的，可以快速地掌握大量的新技术、新框架，不仅懂得原理，还能把新的技术落地到公司的产品中去。这是衡量程序员工作能力的一个重要标准，那么怎样才能将技术运用自如呢？唯有实践。基于此，我想把自己在日常实践中的一些经典案例，编著成一本成体系的书，以便为想要进步的 Android 程序员增加更多的实战经验，这也是编写本书的核心目的所在。

编写本书的另外一个目的，是帮助程序员建立产品的思想，对于技术而言，孤立的存在是没有任何意义的，技术只有与需求相结合，才能具有自身的价值。技术人员在开发的过程中，要时刻了解所完成的功能可以为公司带来哪些价值，是提升用户的访问兴趣，还是提升用户的使用流畅度，抑或是其他。当以产品思维去思考技术的时候，就会有动力、有目的地学习更多有价值的技术，而不是哗众取宠地学一些“看似有用”的新技术。

除此之外，还有理解架构的本质。一些技术人员经常会问：“为什么要使用架构？这只会增加额外的代码量，而且并不会对功能或性能有所提升，只需要完成必要的开发任务即可。”这种想法是非常浅显的，因为任何一个应用都不是一次成型的，需要不断地迭代，不断地扩展，同时需要不断地修改已有的业务逻辑，这就会涉及系统兼容性的问题。如何修改新的业务逻辑而不影响旧的业务逻辑？如何最大限度地复用已有的业务逻辑？架构就是解决这类问题的钥匙，一个优秀且合适的项目架构可以保证系统的稳定性，当开发新的功能或者修改旧的功能时，不至于破坏已有的业务体系。

本书的实例都是经典实战实例，每一个例子都代表一类在开发中需要掌握的技巧。本书由浅入深地逐个讲解需要掌握的开发理论与实践，共分为七部分。

◎ 进阶基础：偏重于源码的解析和理解，介绍阅读源码的技巧，以 Activity 和 View 为例，管中窥豹地分析其中的基础知识。读者也可依据此类方法，分析其他系统

组件的源码。

◎ 高阶控件：介绍若干复杂的控件布局，即 AppBarLayout、CoordinatorLayout 和 ConstraintLayout。通过实例，让读者理解在复杂控件布局中子视图是如何组合和相互关联的。

◎ 项目架构：分析主流架构的设计思想，即 Google 推荐的 MVP 和 MVVM，还有 Facebook 的 Flux。理解这些架构是如何组织和管理大型项目的，以及它们的优点和缺点各有哪些。

◎ 响应式编程：响应式是一种编程思想，在处理网络请求和功能测试时，基于响应式框架的项目拥有更好的可扩展性和可维护性，响应式编程三剑客即 RxJava、Dagger 和 Retrofit。

◎ 酷炫功能与精美动画：实现两个稍复杂的功能，分别是基于第三方控件和基于系统控件的扩展；并实现两个动画效果，分别用于页面展开和页面切换。对于功能和动画，不同的需求或样式层出不穷，本部分侧重于开发思路的分享。

◎ Kotlin 与 SVG：Kotlin 是用于替代 Java 的高阶编程语言，SVG 是用于替代 PNG 格式的图像技术，本书着重介绍一些基础概念，提纲挈领，以便于读者后续进行自主学习。

◎ 测试与优化：分享一个主流的自动化测试框架，以及优化内存与电量的一些常见方法。产品的性能与功能同样重要，应用的高品质也会提升产品的用户体验。

这七部分几乎已经包含高级 Android 编程的全部内容，本书的每个部分都会通过多个实例，从不同的角度引领读者亲身实战，真正地掌握高级编程的核心开发技巧。但是，实例的数量终究有限，希望读者更多地关注于实战中的开发思想，而不是具体的代码逻辑，代码总会不断地更迭，解决问题的思维却历久弥新。本书中的实例更多的是以点带面，读者可以一边阅读和思考，一边编写代码，相信读完本书，一定受益匪浅；同时，通过本书的实例可以解决一些常见的开发需求。衷心希望每位读者在阅读完本书之后，都“不虚此行”！

将本书送给我正在怀孕的妻子，感谢你在生活和工作中给予我的支持和帮助。

王辰龙

2018 年 5 月于北京海淀

目　录

第 1 章

进阶基础

1.1 深入剖析 Activity 的生命周期

在 Android 系统中，所有应用都是由系统提供的四大组件构成的，即 Activity（活动）、Service（服务）、ContentProvider（内容提供者）和 BroadcastReceiver（广播接收器），而它们之间通过 Intent（意图）进行通信并相互关联。其中，Activity 作为与用户打交道最为频繁的组件，承担起显示的重任，其重要性不言而喻。因此，在开发中，对于 Activity 的全部工作细节，需要熟练掌握。

在 Android 系统中使用生命周期（Lifecycle）描述组件在不同状态之间的切换。通过管理组件的生命周期实现资源的获取与释放，在不同阶段的生命周期中触发相应的回调函数，在回调函数的上层方法中，系统控制组件的创建与销毁。同样，Activity 作为组件之一，也有着一套有序的生命周期，管理 Activity 的全部行为。

图 1-1 展示的是 Activity 的完整生命周期（图片来源于 Android 官网）。

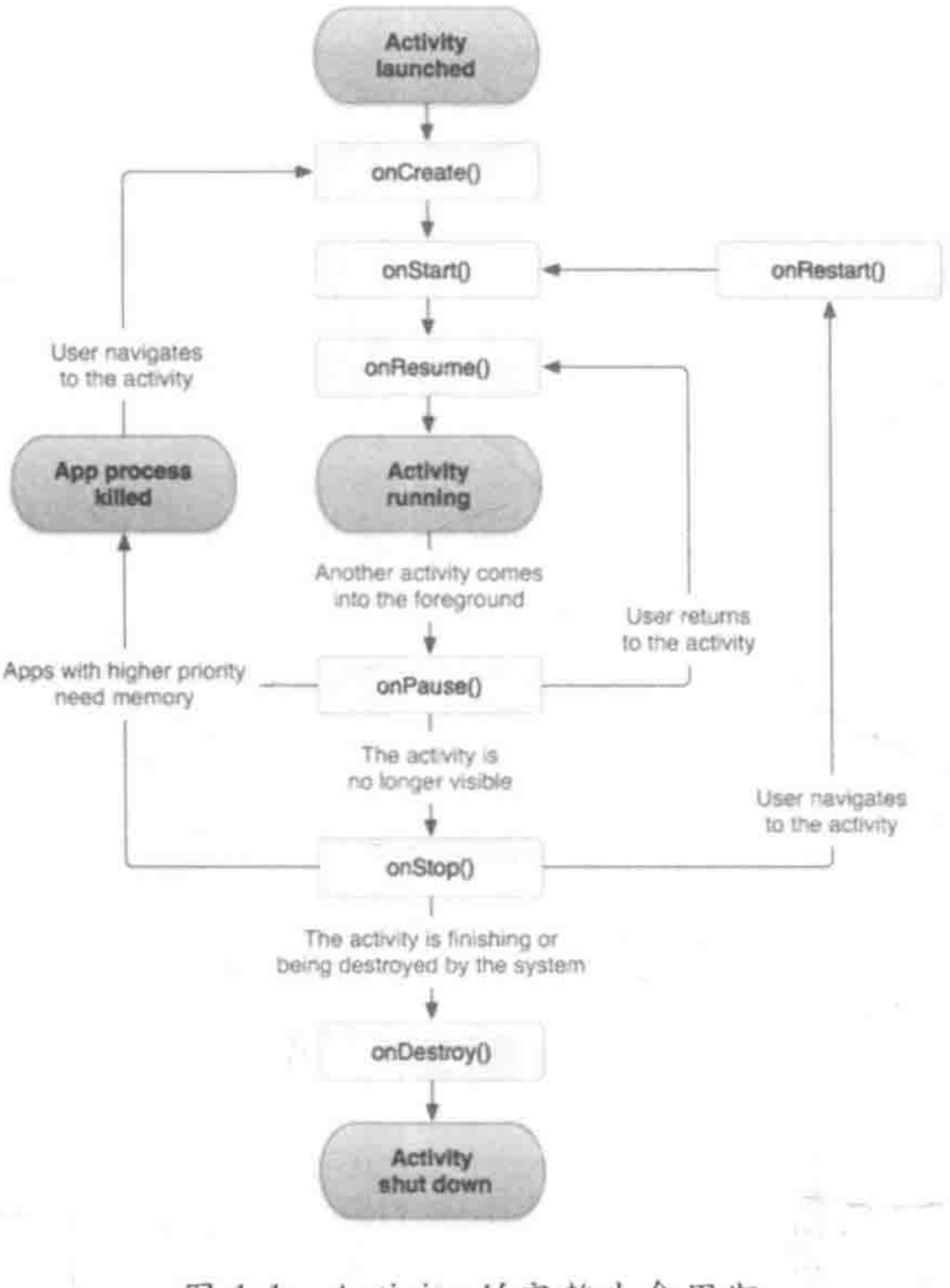


图 1-1　Activity 的完整生命周期

在系统中，有两种情况可以触发生命周期的改变，第 1 种是有用户参与的生命周期的改变，第 2 种是因为系统回收或配置修改而导致的生命周期的改变。为了更加直观地展示这两种改变生命周期的行为，本书编写了一个相关实例。通过这个实例，可以更加清晰地理解 Activity 的生命周期的变换形态。实例分为三个部分，将有用户参与的生命周期的变化分为因硬件导致的生命周期变化、在页面切换时生命周期的变化及由系统原因导致的生命周期的变化。

该实例的完整代码的下载地址为 https://github.com/SpikeKing/wcl-activity-lifecycle-demo。

1.1.1　Activity 的生命周期的各种状态

在 Activity 中，生命周期状态主要包含 6 种，即 onCreate()、onStart()、onResume()、onPause()、onStop()和 onDestroy()。每种生命周期都代表着 Activity 处于不同的状态：onCreate()表示 Activity 的创建，onStart()表示 Activity 的启动，onResume()表示 Activity 的恢复，onPause()表示 Activity 的暂停，onStop()表示 Activity 的停止，onDestroy()表示 Activity 的销毁。

```
// Activity组件
public class Activity extends ApplicationContext {
```

```
    // Activity 的创建
    protected void onCreate(Bundle savedInstanceState);

    // Activity 的启动
    protected void onStart();

    // Activity 的恢复
    protected void onResume();

    // Activity 的暂停
    protected void onPause();

    // Activity 的停止
    protected void onStop();

    // Activity 的销毁
    protected void onDestroy();
}
```

这些状态之间两两配对、有始有终，构成三组生命周期，相互嵌套。这三组生命周期分别为完整的生命周期（Entire Lifetime）、可视的生命周期（Visible Lifetime）及前台的生命周期（Foreground Lifetime）。

1）完整的生命周期：表示 Activity 组件从创建到销毁的全部过程，是最外层的生命周期。生命周期发生在调用 onCreate()方法与调用 onDestory()方法之间。在 onCreate()的方法中，系统会创建 Activity 的全局状态，例如填充布局等；在 onDestroy()方法中，系统会释放 Activity 所保存的资源。需要注意的是，onDestroy()不能保证被调用的时机，例如 Activity 在 Activity 栈中，在系统内存不足时，就可能触发强制销毁调用 onDestroy()方法，而在其他情况下，只有当 Activity 出栈时，才会调用 onDestroy()方法。

2）可视的生命周期：表示 Activity 组件从用户可视到离开用户视线的全过程。在这期间，用户可以在屏幕上看见当前的 Activity。生命周期发生在调用 onStart()方法与 onStop()方法之间。在 onStart()方法和 onStop()方法中，执行与用户可视相关的逻辑。因为 onDestroy()方法并不一定会被调用，所以在 onStop()方法中，一般会执行保持当前 Activity 状态的逻辑。在 Activity 的全部存活时间中，onStart()方法和 onStop()方法可能会被多次调用，这是因为 Activity 可能交替地由可视状态到不可视状态，再恢复到可视状态。

3）前台的生命周期：表示 Activity 组件显示于其他 Activity 组件之前，即位于 Activity 任务栈的栈顶，拥有最高优先级的资源使用权限，同时可以获得用户的输入焦点与用户进行交互。在可视的生命周期中，Activity 组件可能位于全透明或部分透明的 Activity 下面，即属于可视状态。而前台状态必须位于全部的 Activity 之上。生命周期发生在调用 onResume()方法与 onPause()方法之间。在 onResume()方法和 onPause()方法中，执行与交互相关的逻辑。同样，在 Activity

的全部存活时间中，onResume()和 onPause()也可能会被多次调用，即 Activity 可能会频繁地获取与失去焦点。

这三类生命周期循环嵌套，完整的生命周期包含可视的生命周期，可视的生命周期又包含前台的生命周期，它们共同构成 Activity 的生命周期体系。除此之外，还有三个特殊的生命周期状态，即 onRestart()、onRestoreInstanceState()、onSaveInstanceState()。当页面从 Activity 栈内调至栈顶时，会调用 onRestart()方法，初次创建时不会调用；onSaveInstanceState()用于储存 Activity 的状态信息；onRestoreInstanceState()用于恢复 Activity 的状态信息，仅用于系统导致的页面重建，而用户导致的页面重建需要在 onCreate()中由开发者自主恢复状态信息。

1.1.2 实例：准备

下面通过实例，让我们更好地理解生命周期的状态切换。实例分为三个部分：因硬件导致的生命周期变化、在页面切换时导致的生命周期变化、由系统原因导致的生命周期变化。在讲解实例之前，先展示本实例的运行环境，以便读者更好地理解与执行代码，本书其他实例的运行环境与此实例类似。动手实践是学习编程的不二法门，也是成为高级程序员的必要条件之一。

1．硬件环境

实验真机为 Redmi Note 4X，即红米 Note 4X；操作系统为 Android 6.0。

实验模拟器：操作系统为 Android 7.1.1，如图 1-2 所示。

图 1-2 模拟器（开发电脑：MacBook Pro 10.11.6）

IDE（Integrated Development Environment）：Android Studio 2.3.2，如图 1-3 所示。

图 1-3　IDE

2．软件环境

Android Tools 的 Gradle 版本为 2.2.3，位于根目录下的 build.grale 文件中

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:2.2.3'
// Android Tools 的 Gradle 版本

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
```

Android 的 Gradle 构建版本为 2.14.1，位于根目录下的

```
/gradle/wrapper/gradle -wrapper.properties 中:
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
    zipStorePath=wrapper/dists
    distributionUrl=https\://services.gradle.org/distributions/gradle-2.14.1-all
.zip
```

Android SDK 版本为 24，位于项目目录下的/app/build.gradle 中：

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

android {
    compileSdkVersion 24
```

```
    buildToolsVersion '24.0.0'

    defaultConfig {
        applicationId "org.wangchenlong.xxx"
        minSdkVersion 14
        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'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:24.0.0'
}
```

全部实例代码都已经通过测试，从网络下载后，运行项目即可使用。读者在阅读本节时，需要提前下载源码，并调试通过，根据所讲述的内容运行代码，通过实战更好地学习本节知识。

1.1.3 实例：因硬件导致的生命周期变化

我们在使用应用的过程中，除在应用内的交互外，通过手机的硬件按键，也能触发生命周期的改变。对于应用内的交互会导致页面的切换，则触发常规的生命周期状态的改变，从 onCreate()到 onDestroy()依次调用，这是基本的生命周期变化。除此之外，还需重点关注一些特定的硬件操作，比如用户按手机的菜单（Home）键和后退（Back）键。

对于一款基本的 Android 手机而言，手机底部都会包含两个基础的按钮，一个是位于中部的菜单键，另一个是位于右侧的后退键。因此，除了在应用内提供的交互场景，用户最常用的两种操作就是按菜单键或后退键。那么这两种常用的操作又会给 Activity 带来哪些生命周期的变化呢？下面通过实例来讲解。

先来看看应用的主页 - MainActivity.java，通过源码来分析这个 Activity 页面是如何组成的，如图 1-4 所示。

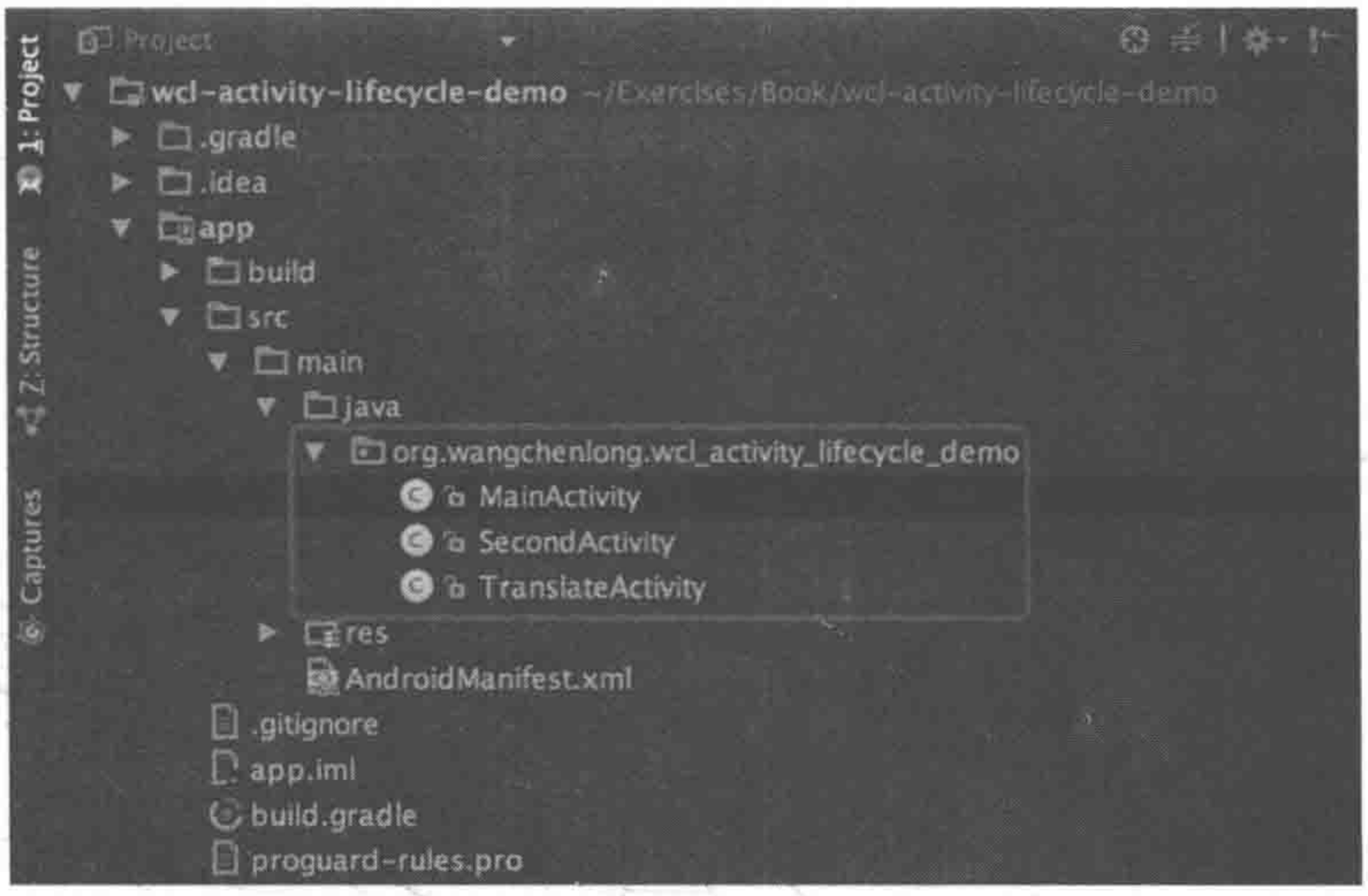


图 1-4　MainActivity

首先，声明 TAG 标签，作为 Log 输出的标签，用于在 Logcat 中过滤信息；使用 ButterKnife 插件，声明两个按钮，用于页面的跳转；声明 EXTRA_TEXT 标签，用于在 Activity 中模拟状态的存储。对于字符串静态常量，一般使用大写字母加下画线的形式表示。

代码如下：

```
    // Log 输出的标签，用于在 logcat 中过滤
    private static final String TAG = "DEBUG-WCL: " + MainActivity.class.
getSimpleName();
```

```
    @Bind(R.id.main_b_translucent)
    Button mBTranslucent; // 透明页面
    @Bind(R.id.main_b_second)
    Button mBSecond; // 第二个页面

    private static final String EXTRA_TEXT = "extra_text";
    // 用于在 Activity 中模拟状态的存储
```

注意：ButterKnife 是由 Jake Wharton 编写的一个用于声明布局中控件的插件，代替调用 findViewById()方法，节省了大量的开发时间。在项目目录中，添加 ButterKnife 的依赖即可使用。具体原理与使用方式可参考 ButterKnife 的官方网址 https://github.com/JakeWharton/butterknife。

其次，onCreate()方法作为创建 Activity 的第一个调用函数，负责初始化 Activity 的全部行为，我们一般重点编写代码逻辑的部分。例如 setContentView()，设置页面布局；ButterKnife.bind()，负责绑定 ButterKnife 并初始化，只有在初始化 ButterKnife 之后，才能调用 Bind()标注进行布局

ID 的绑定；设置 mBTranslucent 按钮和 mBSecond 按钮的绑定事件；获取 EXTRA_TEXT 的状态存储信息及输出 onCreate()方法的调用提示。在每个生命周期方法中，都添加了相应的调用提示，方便在 Logcat 中观察当 Activity 页面改变时，生命周期变化的整个过程。

代码如下：

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

    ButterKnife.bind(this); // 绑定 ButterKnife，并初始化

    mBTranslucent.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(MainActivity.this, TranslateActivity.
class));
        }
    });

    mBSecond.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(MainActivity.this, SecondActivity.class));
        }
    });

    // 获取 EXTRA_TEXT 的状态存储信息
    if (savedInstanceState != null) {
        String txt = savedInstanceState.getString(EXTRA_TEXT);
        Log.e(TAG, "[onCreate]savedInstanceState: " + txt);
    }

    Log.e(TAG, "onCreate"); // 调用提示
}
```

再次，在 Activity 的启动过程中，先调用 onCreate()方法，然后依次调用 onRestart()、onStart()、onRestoreInstanceState()、onResume()，其中 onRestart()与 onRestoreInstanceState()并不是每个生命周期都会调用的。

代码如下：

```
@Override
protected void onRestart() {
    super.onRestart();

    Log.e(TAG, "onRestart");
```