目　录

题目：AI 健身系统

1. 概述

1.1. 研究背景

健康，无论是对于个人还是对于国家而言，都是绕不开、躲不过的重要话题。

立于国家的层面，习近平总书记在党的十九大报告中指出：“广泛开展全民健身活动，加快推进体育强国建设”“人民健康是民族昌盛和国家富强的重要标志”等重要论断，为“体育强国梦”和“健康中国梦”指明了方向。

站在人民的角度，在如今这样一个互联网与公民生活深度融合的时代背景下，人们也越来越青睐使用线上健身类 APP 辅助健身。知名研究机构 IHS 发布的报告显示，健身类 App 下载量从 2012 年的 1.56 亿次增加到 2017 年的 2.48 亿次，增幅为 63%[1]。由此可见，健身类 App 俨然已经了新的运动模式和生活方式。

1.2. 研究意义

目前，传统健身 APP 发展日趋成熟。以“Keep”为例，它可以做到根据用户多样化的需求，有针对性地提供丰富多彩的健身课程，包括“减脂”、“塑形”、“增肌”等等，帮助用户可持续、分阶段地完成健身挑战。传统健身 APP 大多以播放固定长度的预录视频为主要方式，辅以健康穿戴设备用于监控体征。

但是，用户日趋追求优良的用户体验与新科技赋能带来的新特性。仅通过机械的语音播报提醒用户做出正确有效的健身姿势是十分困难的。而且，教学者的标准节拍或许并不适用于所有健身人群，导致 APP 记录完成的动作数、消耗卡路里数并不准确，给用户带来不佳体验。

旨在尽可能地削弱这一“用户痛点”，本设计将做一个小尝试：结合计算机视觉、人工智能、卷积神经网络与深度学习的前沿科技，使健身软件可以最大限度地利用手机摄像头，识别并计数用户的标准动作。

[1] [1]王苗. "运动类 app 传播效果研究." 科技传播 5(2015):2.

2. 系统设计

2.1. 关键技术

2.1.1. 综述

2.1.2 ~ 2.1.8：Android X 新特性新技术

2.1.9 ~ 2.1.10：计算机视觉与图像处理的相关技术，包括 Camera X 库

2.1.11 ~ 2.1.14：本例使用到的其他重要技术及外部依赖

2.1.2. JetPack 概览

API 等级和 Android 版本在不断更新。2018 年，谷歌 I/O 开发者大会发布了最新的 JetPack 应用架构工具库，AndroidX 是 JetPack 的命名空间。

它的存在原则之一是“不应该使用组件存储数据和状态”。换而言之，我们需要采用 MVVM 模式，通过数据模型驱动界面，分离界面与数据。

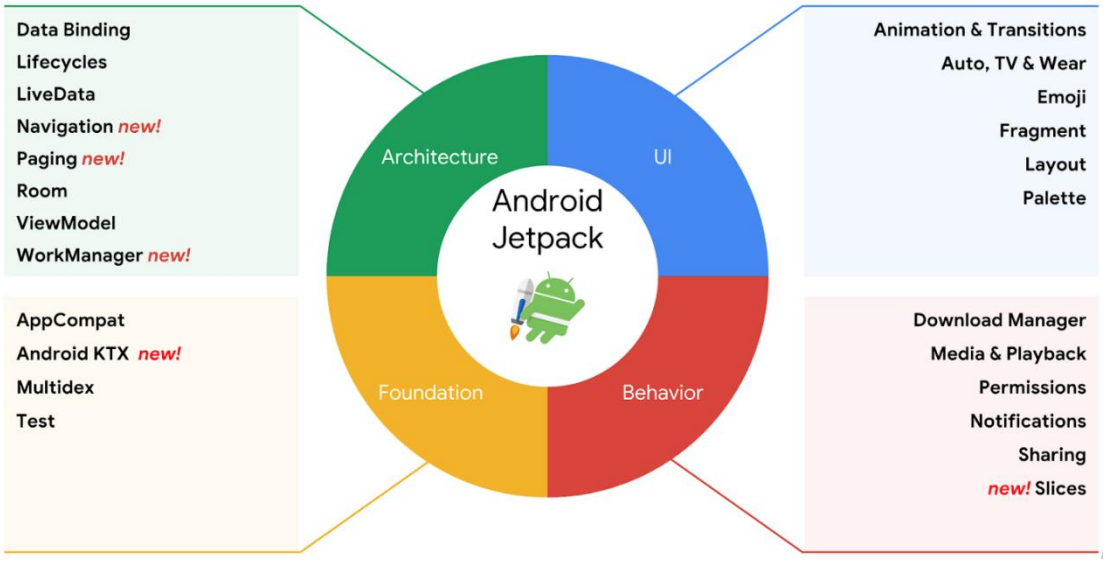


图 2.1　JetPack 新特性

如上图 2.1 所示，我们发现，JetPack 中有许多工具是课堂上的“老熟人”，例如，Room 数据持久化技术；同时其中也有许多内容是“新面孔”，例如，Navigation 导航、View Model、Lifecycle 与 Date Binding 技术。囿于篇幅所限，应用于本 APP，但课堂无暇提及的最新技术，我们将在接下来几个小节重点介绍。

2.1.3. JetPack 之 Fragment 与 Navigation

在不同屏幕和应用之间“切换”是用户体验的核心组成部分，如果将每一个屏幕视作一个“目的地”(Destination)，“切换”便可以被视作“导航”(Navigation)。“导航状态”的逻辑结构是堆栈；其顶部为当前屏幕，底部为启动屏幕。

这里所谓的导航机制和 Intent 机制不同，它支持碎片（Fragment）穿梭跳转

可以说，Fragment 组件介于 Activity 和 View 之间。

一方面，它未继承 Context,无法独立存在，必须作为宿主视图的一部分；可另一方面，它又可以定义管理自己的布局，具有自己的生命周期（Lifecycle）。

导航编辑器为可视化操作，可自定义界面逻辑流向图、屏幕切换的可能路径、动画效果、弹栈操作等。值得一提的是，动画或许没有想象中那么复杂：它不过是个 XML 文件，有透明度（alpha）和偏移（translate）两种简单定义方式。

2.1.4. JetPack 之 Lifecycle

“朝菌不知晦朔，蟪蛄不知春秋”生死轮回，周而复始，是生物的生命周期。课堂上，我们研讨过活动的生命周期（Lifecycle）的概念，如下图 2.3。

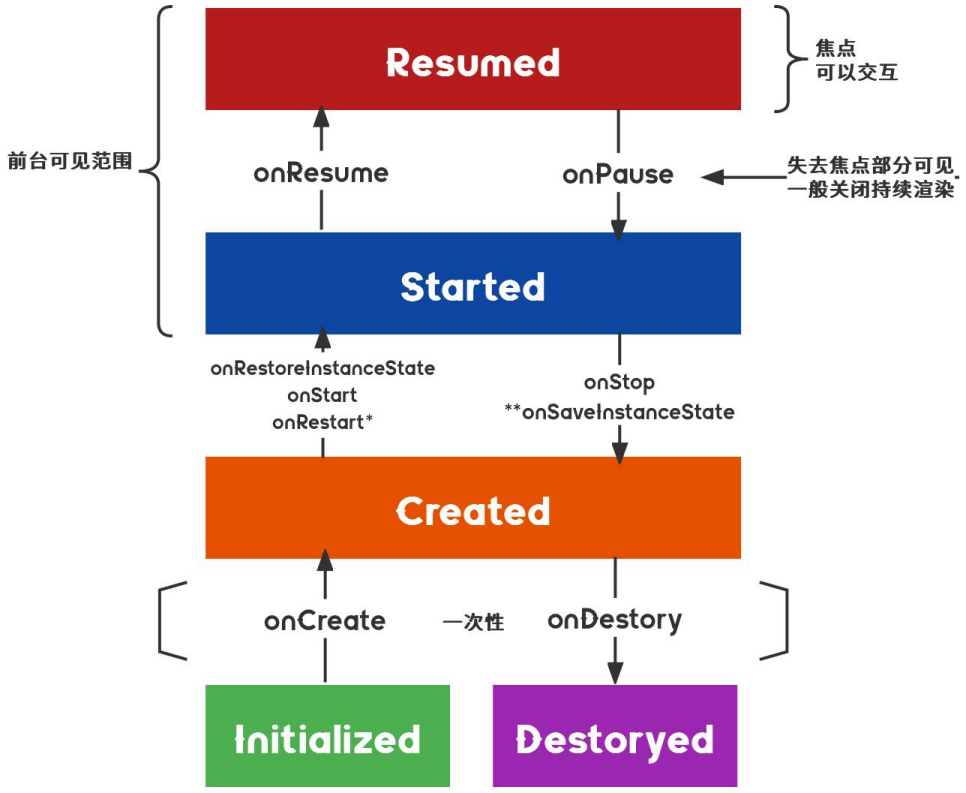


图 2.2　生命周期所处状态与回调函数

生命周期不只是一个概念，更是 JetPack 的核心组件。理解并使用生命周期的重要性自不必多言：一方面，动画渲染、后台线程、配置更改、进程终结，甚至不少系统错误、数据异常都与其有关；另一方面，在性能有限的情况下，无论是系统自动还是编程人工，都希望把资源优先掉配给最活跃最紧急的事务。

例如，我们希望某一个 APP，在处于前台可见状态时（onStart），统计时长，渲染动画，播放音乐，调用手机摄像头；当 APP 处于后台不可见状态时 (onStop)，它便停止计时，停止渲染，停止播放，不再获取摄像头数据。

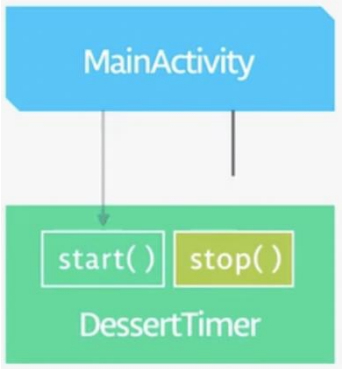


图 2.3　传统高耦合模式

显然，这种依赖 Activity 类回调函数中控制其他对象生命周期的方式，不仅不符合面向对象的编程思想，也会增加出错风险和维护成本。

2017 年，谷歌 I/O 开发者大会首次提出：Lifecycle 类、LifecycleOwner 和 LifecycleObserver 接口。简单来说，生命周期感知型组件（LifecycleObserver）可以观察遵循 Activity 或 Fragment 状态，“自主”调整行为。这与传统“命令式”的思维模式正好相反，可以降低耦合（Coupling），使类变得更加“纯粹”。

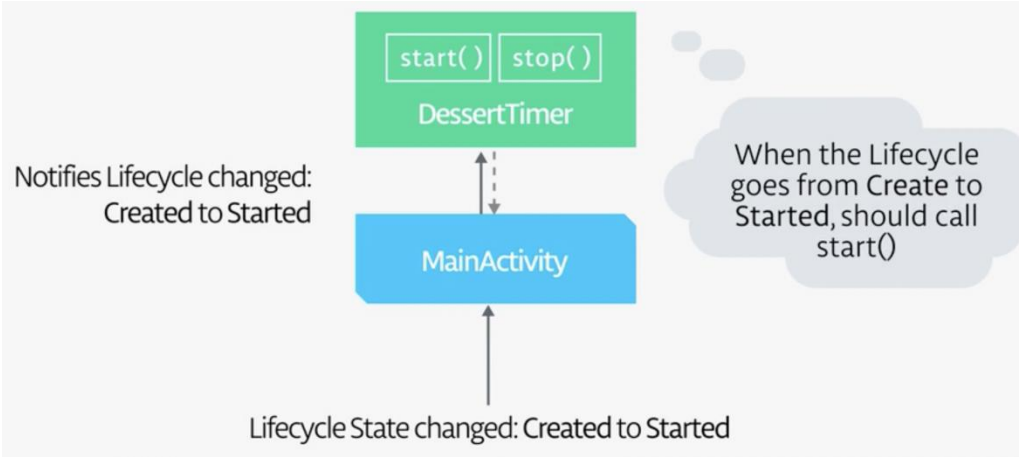


图 2.4　生命周期感知模式

2.1.5. JetPack 之 DataBinding

XML 文件中的视图（Views）在构建项目、注入活动（Layout Inflation）时，系统将在内存自动生成一个视图对象的层次结构（View Object Hierarchy Tree）。同时由于资源类构建了 ID 与对象引用的映射，用户可以委托 findViewById(R.id.*)搜索对象。课堂上，我们惯例将视图对象作为 Activity 类的成员变量（或字段，Field）在 onCreate()一次性初始化，也是因为重复搜索可能导致响应延时。

尽管如此，运行时（Runtime）进行层次遍历仍然是不小的开销。同时，为了消除类型不兼容的安全隐患，早在 2015 年，谷歌便推出了“视图绑定”。系统会在编译时为每一个以<layout>为根组件的 XML 文件生成一个绑定类。绑定类实例包含具有 ID 值所有视图的直接引用；绑定类名为 XML 文件名转换为驼峰式大小写，并在末尾添加“Binding”。

图 2.5　数据绑定示例

而“数据绑定”不仅可以使 Activity 类获取视图对象，顾名思义，还可以使得 XML 文件获取 Bean 类数据，简化代码，甚至，单击监听事件也可以直接在 XML 属性+@表达式进行设置。

2.1.6. JetPack 之 ViewModel 与 LiveData

配置更改（例如旋转屏幕）时，UI 数据可能丢失，是因为界面被系统自动销毁重绘（和会输销毁整个程序不同），同时，界面类的成员变量也被回收。一个符合直觉的解决方法是，将数据的所有权和界面交互逻辑分离开；如此，系统在回收 Activity 类中的对数据类的“引用”，并不会影响数据类本身。

这符合谷歌开发者文档的软件架构设计原则：关注点分离、数据驱动界面、单一数据源和单向数据流。

ViewModel 与 2.1.4 相得益彰，充分体现“观察者模式”的优越性。例如，当 ViewModel 中存在一个 LiveData 容器, 可以指定当前数据的观察者 (Observer)，即每当 LiveData 中存放的数据发生变化时，观察者可以全自动地执行操作，例如，改变 UI 界面中的数据。相比起定时检测更新和通知模式，观察者，如同一个有主观能动性的血肉之躯，具有强大生命力和无比优越性。

甚至，可以设置“双观察”模式，即 ViewModel 可以观察 Activity 的生命周期变化，而 Activity 可以观察 ViewModel 中的数据变化。

2.1.7. JetPack 之 Room Database

2.1.5 提到的 View Model 和 Live Data 仍然是一种运行时数据，如果要想 Android 操作系统销毁整个应用，数据依然“留存”下来。就不得不提到，数据持久化技术（Persistence）。Room 持久性库在 SQLite 上提供了一个抽象层，以便在充分利用 SQLite 的强大功能的同时，能够流畅地访问数据库。

2.1.8. JetPack 之 RecyclerView

RecyclerView 可以轻松高效地显示大量数据，当提供数据并定义每个列表项的外观后，RecyclerView 库会根据需要动态创建元素。

顾名思义，RecyclerView 会回收这些单个的元素。当列表项滚动出屏幕时，RecyclerView 不会销毁其视图。相反，RecyclerView 会对屏幕上滚动的新列表项重用该视图。这种重用可以显著提高性能，改善应用响应能力并降低功耗。

2.1.9. JetPack 之 Camera X 架构

虽然，课堂上我们使用 registerForActivityResult 可以预览图片，但那只是打开了系统的相机应用，没有触及到摄像头的独立部分，或者说，宏观层面而言，我们希望做到的是，调用手机的摄像头，相机程序在底层运行，健身应用在表层运行，也就涉及到了并发编程的相关技术。如果不使用并发编程，长时运行的任务，又将阻塞主线程并导致应用无响应。

不过在这里，我们不纠结概念区分，也不深入探讨并发底层机理。“没有什么是真的，一切可能都有问题。”《Java 编程思想》作者 Bruce Eckel 如是写到。可见，并发编程是一个颇具难度的议题。简单来说，并发性是一系列性能技术，专注于减少等待（Concurrency is a collection of performance techniques focused on the reduction of waiting）。与早期 Java 版本不同，如今被鼓励的做法是使用线程池（thread pool）。Runnable 和 Callable 封装一个异步运行的任务，Future 保存异步计算的结果。FutureTask 实现了 Future 和 Runnable 接口。线程池会在方便的时候尽早执行提交（submit）的任务，调用 submit 时，会得到一个 Future 对象，可用来得到结果或取消任务。

再来提 CameraX 架构。之所以它可以专注相机的核心功能，支持自动生命周期管理，不需要考虑底层细节与兼容性问题，逻辑简单，很大程度上是因为它提出了一个新概念——场景用例（UseCase）。如此以来，上层逻辑上，我们只需要考虑，以下几个问题：

1. 用哪个镜头？例如，前置镜头。
2. 用它干什么？例如，预览分析。
3. 谁指导开关？例如，感知某一个活动的生命周期。

2.1.10. Google ML Kit & Mediapipe

Google Machine Learning Kit （简称 ML Kit）的 Pose Detection API 可以使得终端设备实时检测人体 33 个骨架关键点，并提取其视觉信息——三维坐标。正是关键点坐标的提取，给姿态分类、运动计数等应用提供了有效的抓手。

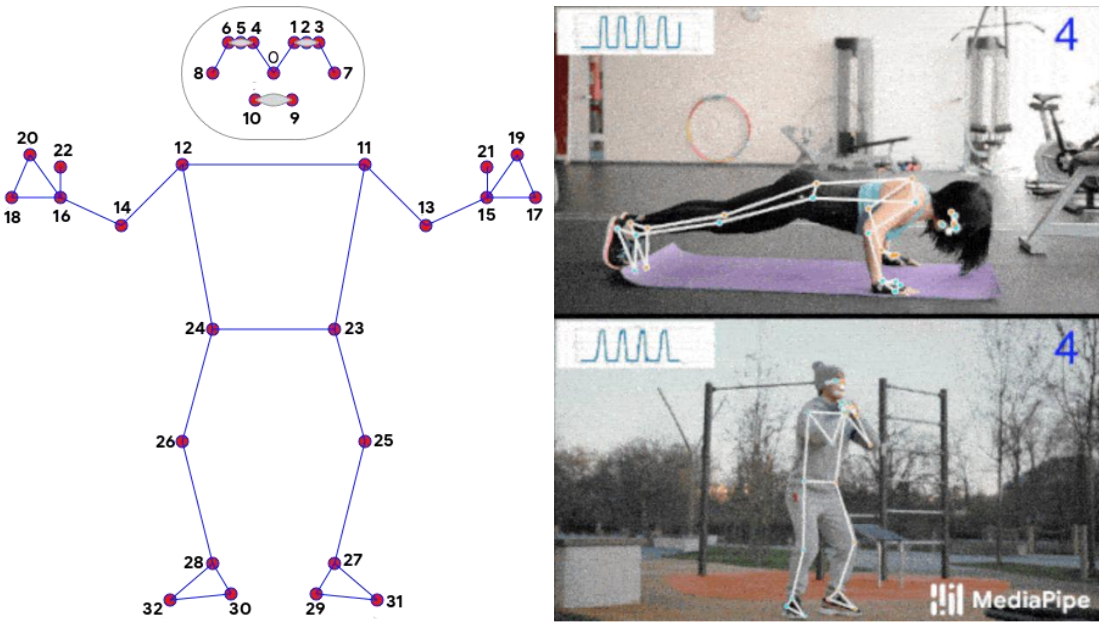


图 2.6　33 个人体关键点（左）与运动计数应用（右）

与传统可以硬编码解决的算法问题不同，计算机处理图像分类的一个突出问题是，“没有确定的简单的数值特征”：人工几乎无法构造或推算出某个确定的计算方式或达标阈值作为分类条件。

而如今，在机器学习的加持下，数据驱动不可或缺：计算机在学习海量数据、提取可以被数值化的信息之后，我们就可以利用已经训练学习过的模型辅助分类。具体而言，可以使用简单的 **K-最近邻分类算法 (K-nearest neighbors algorithm)**。这种算法的思想异常直觉：在训练集中找 K 个和当前样本数值特征最相似的，它们大多数属于哪一类别，待测样本就属于哪一类。

2.1.11. 其他技术之 Canvas

Canvas 意为“油画布”、“画板”，与之对应的是画笔（Paint），通常用于 Android 系统 2D 绘图。在诸多模块，例如人体关键点检测结果与图像分析时延帧率信息显示，需要根据统计数据构建图像或图表，便可以借助 Canvas 来构建自定义类，以便自定义显示可视化信息。

2.1.12. 其他技术之 EazeGraph[2]

开源代码，EazeGraph 是安卓系统上的图表库。支持生成四种图表：条形图表，堆积柱形图，饼状图，折线图，便携易上手。

2.1.13. 其他技术之 DatePicker

DatePicker 继承自 FrameLayout 类，日期选择控件的主要功能是向用户提供包含年、月、日的日期数据并允许用户对其修改。如果要捕获用户修改日期选择控件中的数据事件，需要为 DatePicker 添加 OnDateChangedListener 监听器。

2.1.14. 其他技术之 RulerView[3]

开源代码，模拟刻度尺，可以滑动选择身高、体重。实现思路：

1. 初始化画笔，以及其他需要的参数
2. 重写 onMeasuer()确定尺子的大小
3. 重写 onDraw()绘画出静态尺子，并且将一些滑动时需要改变的参数设置为变量，绘制时只绘制当前屏幕可见区域，滑动尺子时，只刷新当前屏幕模拟滑动并不是真正的滑动
4. 重写 onTouchEvent()处理滑动，增加滑动速率监听 VelocityTracker 以及惯性滑动以及抬起手指时指针落在刻度上面需要的属性动画 ValueAnimator

[2] https://github.com/paulroehr/EazeGraph
[3] https://github.com/superSp/RulerView

2.1.15. 其他技术之 SharedPreferences

SharedPreferences 是 Android 平台上一个轻量级的存储类，用来保存应用的一些常用配置。原理是生成 xml 文件保存到：/data/data/包名/shared_prefs 目录下，类似键值对的方式来存储数据。用于相机默认设置，用户记住密码。

2.1.16. 其他技术之 TTS

TTS 是语音合成应用的一种，它将储存于电脑中的文件，如帮助文件或者网页，转换成自然语音输出。用于健身计数时播报。

2.2. 系统设计

2.2.1. 软件设计图

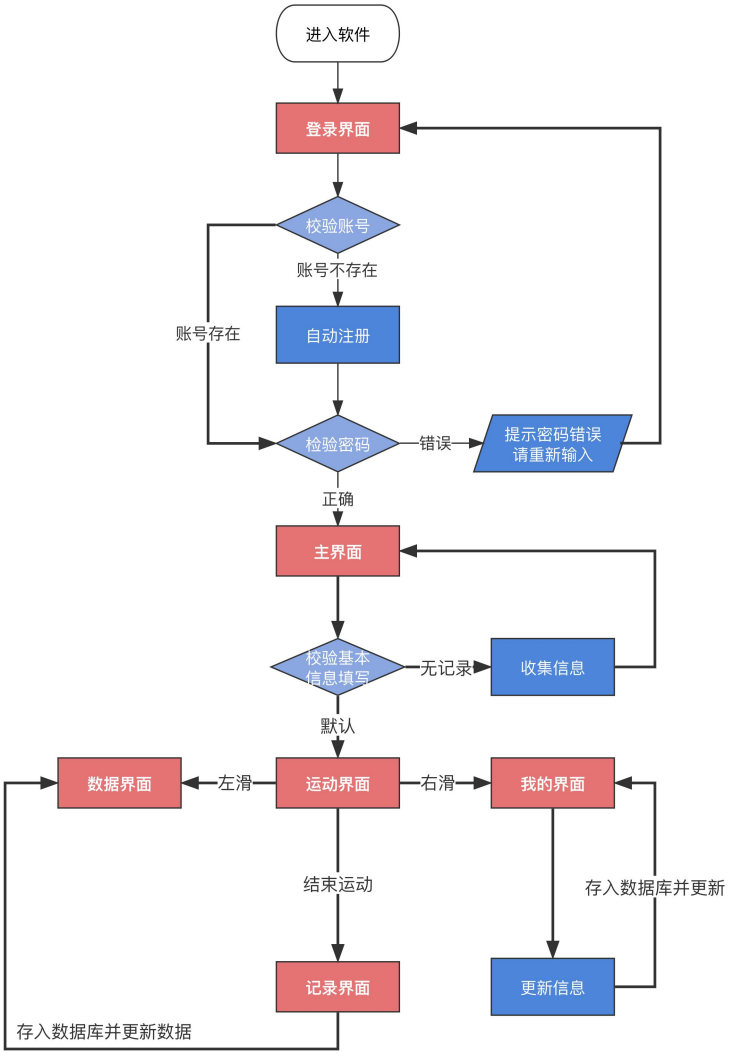


图 2.7 软件设计简图

2.2.2. 功能结构图

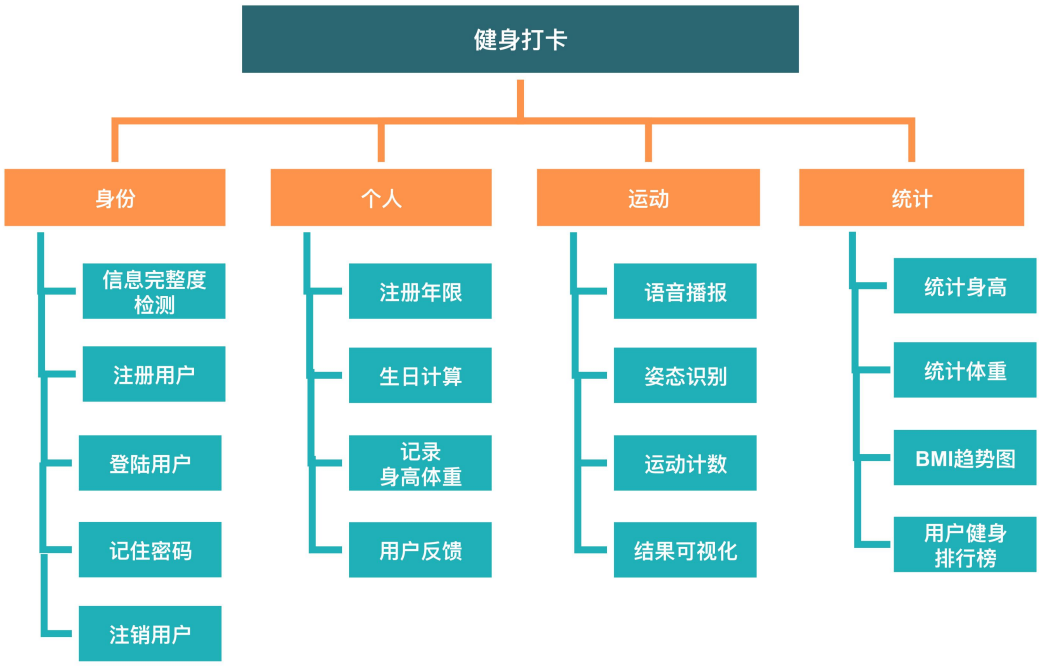


图 2.7 功能结构简图

2.2.3. 数据库设计

表 1 性别表

属性名	含义	类型	说明
sex_id	性别代码	Long	主键，0/1/2
sex_name	性别名称	String	未知/男/女

表 2 状态表

属性名	含义	类型	说明
state_id	状态代码	Long	主键，0/1
state_name	状态名称	String	信息不完整/完整

表 3 运动表

属性名	含义	类型	说明
sports_id	运动代码	Long	主键，自增
sports_name	运动名称	String	
sports_intro	运动简介	String	
sports_csv	运动计数数据集路径	String	

表 4 用户表

属性名	含义	类型	说明
user_id	手机号	Long	主键
user_passport	密码	String	
user_sex_id	性别代码	Long	外键, 引用性别表 id 默认值为 0L
user_birth	生日	Date	
user_state_id	状态代码	Long	外键, 引用状态表 id 默认值为 0L
user_register	注册时间	Date	默认为系统时钟
user_avatar	头像	String	内部存储地址
user_name	昵称	String	
user_sign	个性签名	String	

表 5 指标记录表

属性名	含义	类型	说明
record_id	更新代码	Long	主键
record_date	更新日期	Data	
record_user_id	用户手机号	Long	外键，引用用户表
record_index	指标代码	Long	外键，引用用户表

表 6 运动记录表

属性名	含义	类型	说明
live_id	记录代码	Long	主键，自增
live_user_id	用户代码	Long	外键，引用用户表
live_sports_id	运动代码	Long	外键，引用运动表
live_time_start	开始时间	Date	默认值为系统时钟
live_time_end	结束时间	Date	默认值为系统时钟
live_sports_num	计数	Long	
live_rate	数字评分	Integer	0～10，默认值为-1

2.2.4. UI 设计与 Material Design

样式设计采用经典的“Theme-Style-View Attribute”金字塔结构，尽可能地减少因硬编码带来的繁琐修改，试图给用户带来精致一致的视觉体验 (Consistent Styling) 。同时，借助行业惯例 Material Design[4] 设计打磨，动画和颜色。尤其值得一提的是，用户界面全面改用最新的 Material 3 组件完成，是它使得我对于设计产生了浓厚的兴趣与深厚的敬畏，也是它使得本 APP 的界面如此美丽。

[4] https://material.io/

2.2.5. 应用设计架构

谷歌在安卓开发者文档中强调，“应用架构定义了应用的各个部分之间的界限以及每个部分应承担的职责”，推荐按照下列原则设计应用架构：

1. 关注点分离。基于界面的类（Activity 或 Fragment）只应该包含处理界面和系统交互的逻辑。这些类应尽可能保持精简，以避免产生与组件生命周期相关的问题，并提高可测试性。毕竟，操作系统可能会根据内存不足等系统条件随时销毁它们，所以，最好避免在 Activity 或 Fragment 编写所有代码。

2. 数据驱动界面。数据模型应独立于界面或其他组件。持久性模型是理想之选，因为如果 Android 操作系统销毁应用以释放资源，用户也不会丢失数据。

3. 单一数据源和单向数据流（Unidirectional Data Flow）。一定条件下，只有唯一的数据所有者（Signle Source Of Truth）可以修改数据。在离线优先应用中，应用数据的单一数据源通常是数据库。

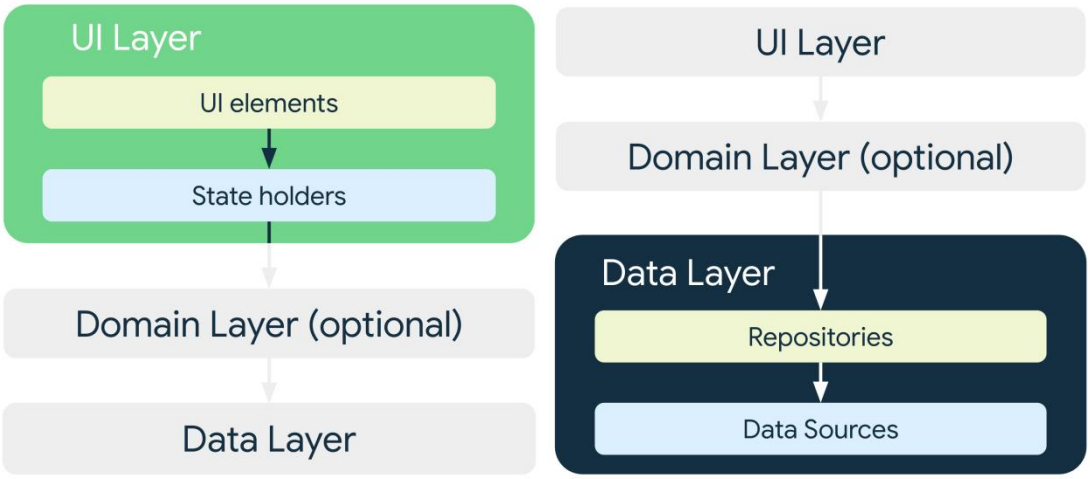


图 2.8　谷歌推荐的应用架构

基于上述架构原则，每个应用应至少有两个层：界面层和数据层。

界面层负责展示数据，更新数据，一般对应 Activity 或 Fragment 类。

数据层包含业务逻辑，包含应用创建、存储和更改数据的规则，向应用的其余部分公开数据，一般对应一个数据类。

网域层是位于界面与数据层之间的可选层。网域层负责封装复杂的业务逻辑，一般该层中的类通常称为用例（UseCase），每个用例都应仅负责单个功能。

3. 系统实现

3.1. 身份模块

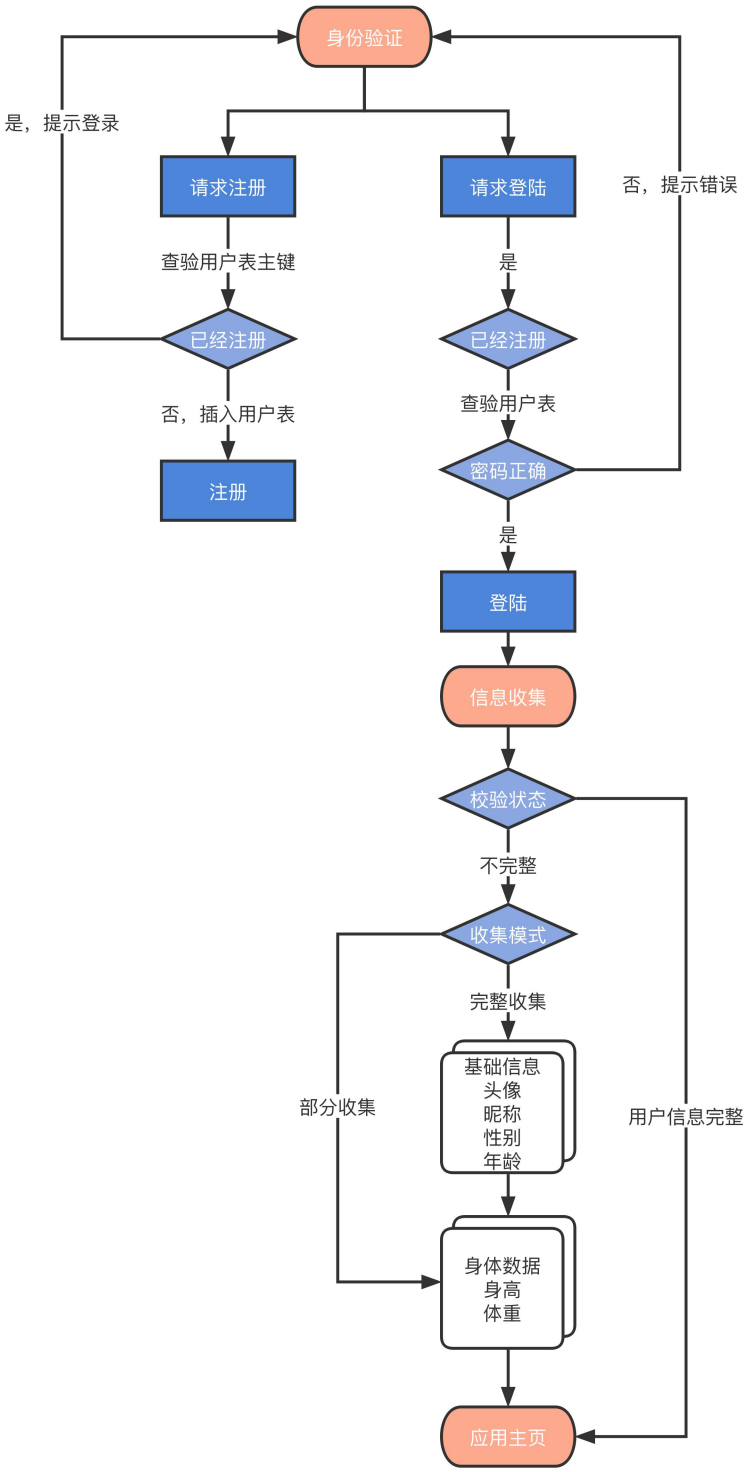


图 3.1　身份验证（逻辑）

3.2. 运动模块

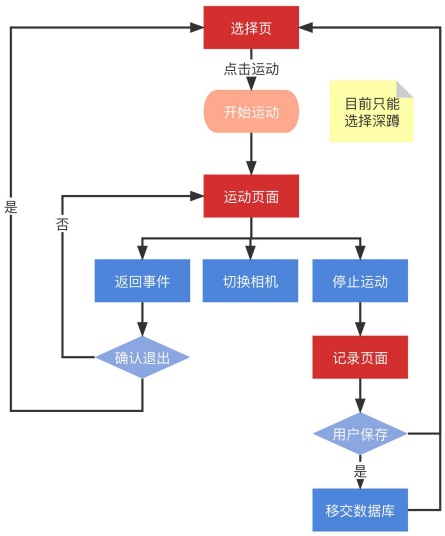
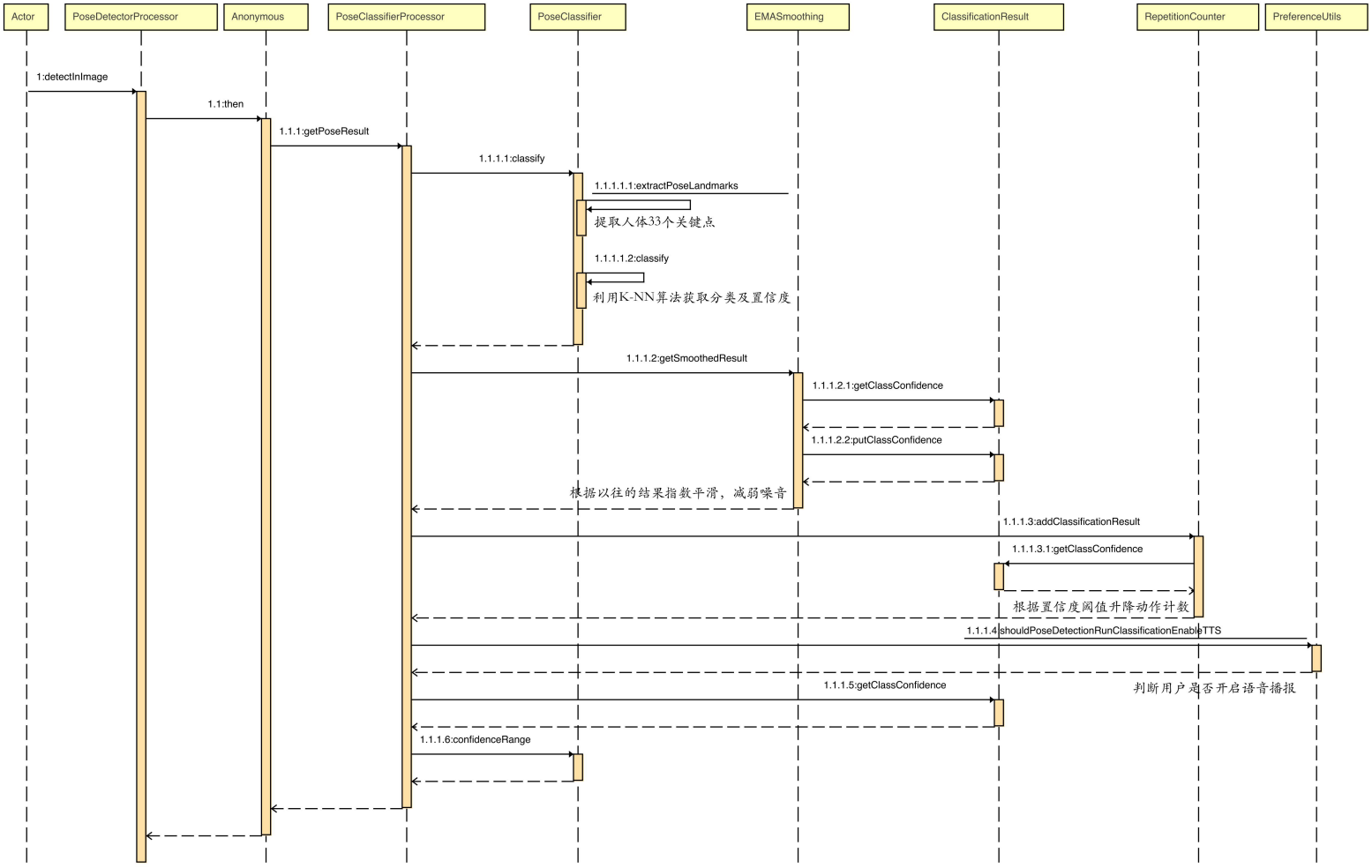


图 3.2　运动模块（逻辑）

正如 2.1 所述，计数的图像来源是 CameraX，核心算法是 K-NN，可视化利用 Graphic 与 Canvas。下面给出图像分析获取分类结果的简易时序图 3.3：



3.3. 统计与我的模块

统计模块主要利用数据库查询语句。

例如，BMI 趋势统计，查询最近 8 次当前用户的健康数据

```
@Query("SELECT * FROM index_record_table
            WHERE record_user_id=:user_id
            ORDER BY record_date DESC LIMIT 8")
```

例如，用户健身排行榜，就是分组查询每个用户最好的成绩，结果降序

```
SELECT user_name, user_avatar, MAX(live_sports_num)
FROM user_table,live_record_table
WHERE user_id=live_user_id
GROUP BY user_id
ORDER BY live_sports_num DESC
```

值得注意的是，我的模块中修改信息、增加健康数据和身份模块中的信息收集运用的是同一个活动，这是由于 InfoActivity 利用导航 Navigation 技术有选择性的在 Fragment 之间切换跳转；可以实现代码重用。例如，当用户需要修改信息时，可以直接跳转至该活动的开始碎片；当用户只需要记录身高体重时，可以直接跳转至该活动身高碎片，并依照顺序依次执行。

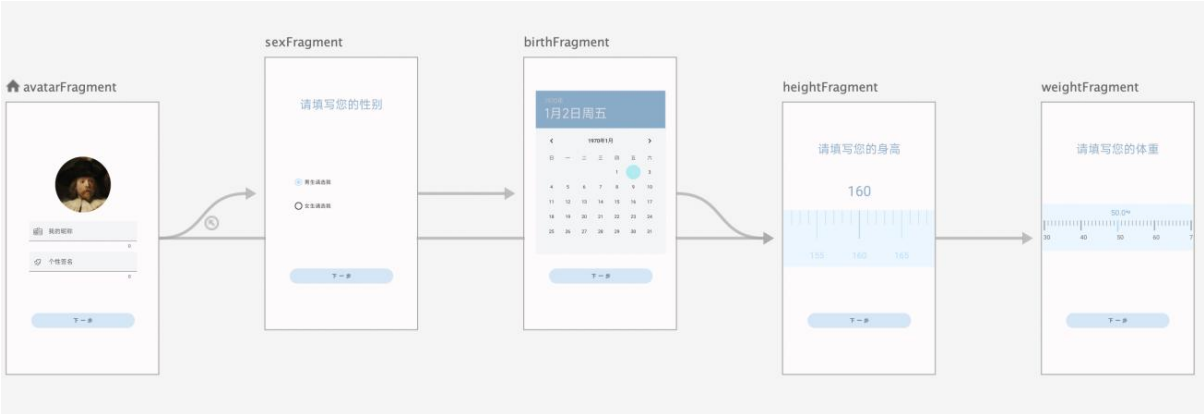


图 3.4　信息收集导航图

信息收集后，用户的状态 ID 将切换为 1L（0L 需要收集）。手机的信息能够在我的主页和数据界面自动更新显示，也利用了 2.1 介绍的观察者模式。数据模块中的 BMI 作为导出属性由下列公式计算得到，超过目标范围将为红色：

$$\mathrm{BMI} = \frac{\mathrm{weight}}{\mathrm{height}^2}$$

4. 系统测试

4.1. 身份模块

图 4.1　验证账号是否存在，密码是否正确

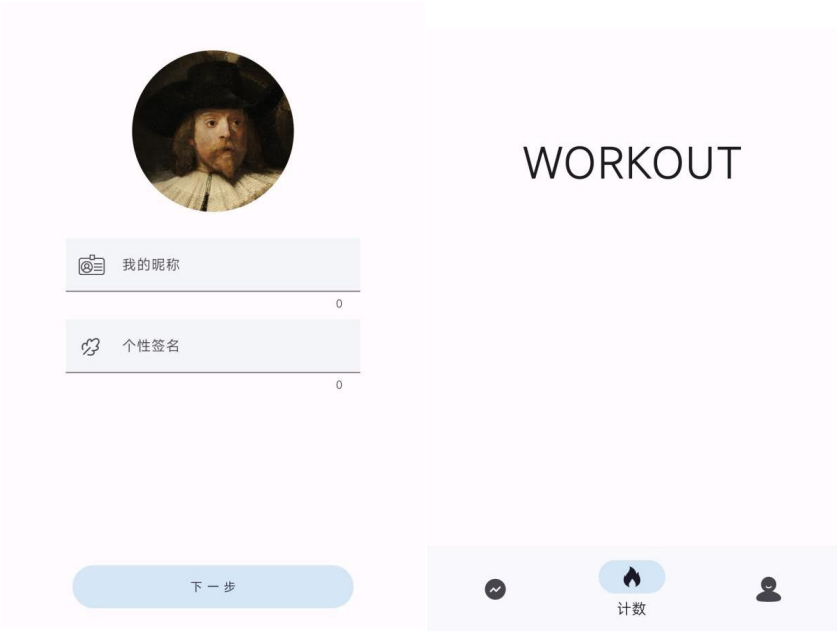


图 4.2　验证账信息，是否需要填写基本信息

4.2. 个人与统计模块

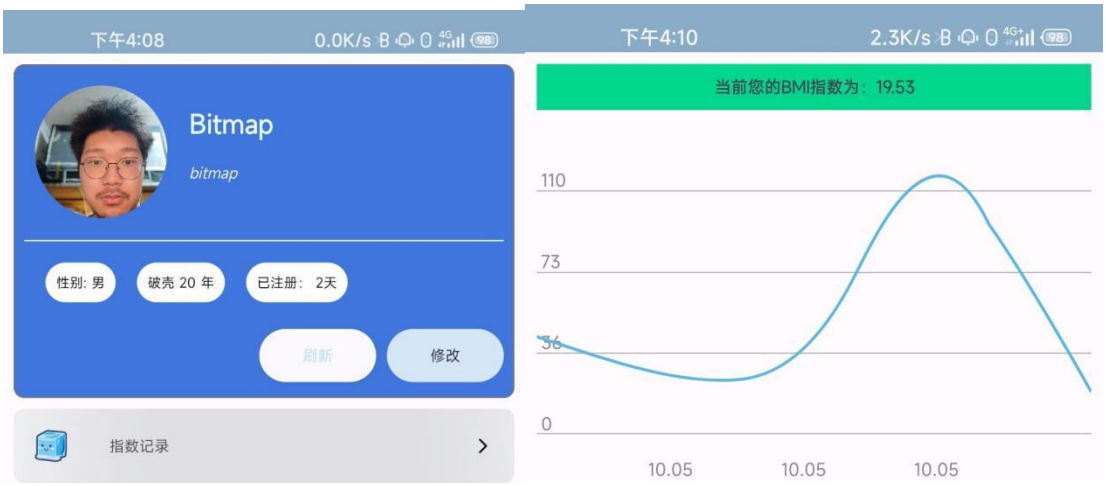


图 4.3　数据自动更新并计算

4.3. 运动模块

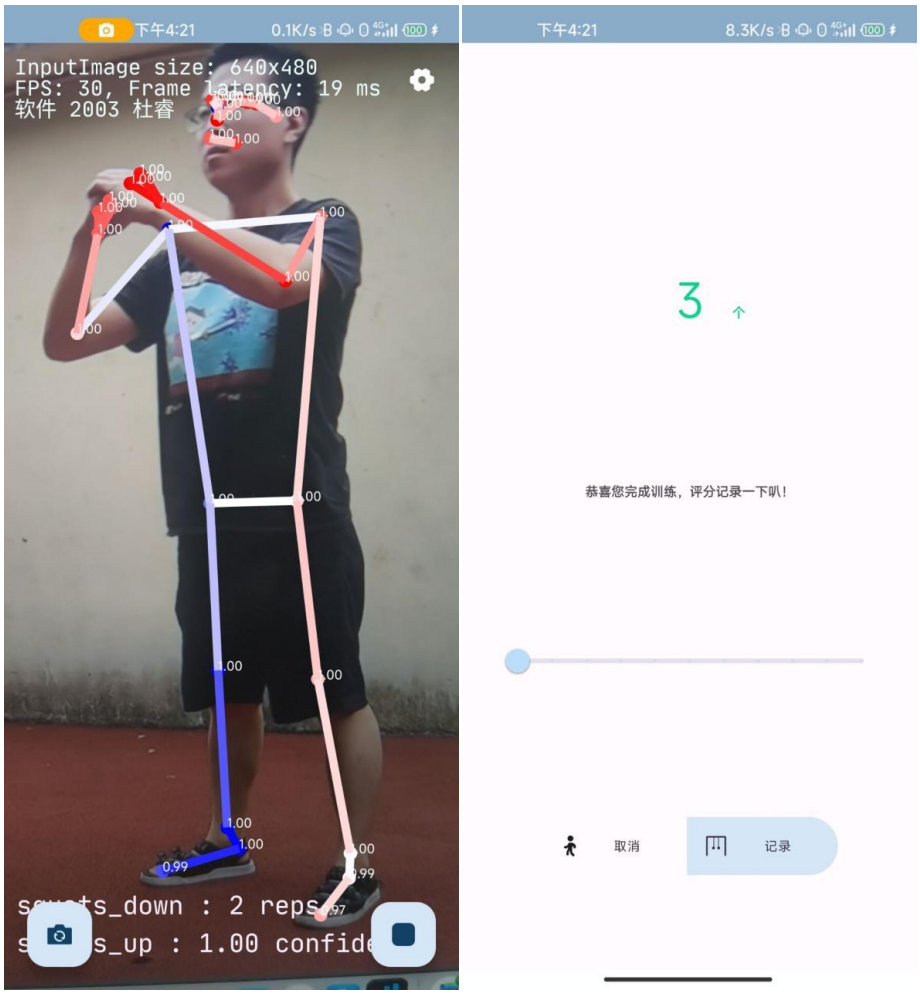


图 4.4　用户点击结束运动后记录

5. 设计总结

5.1. 问题回顾

虽然本 APP 设计距离一流的交互、精致的设计相去甚远，数据库，但是，要完成这样的软件也绝非一日之功。过程艰辛，遇到了诸多问题，下面列举一二：

1. 并发编程。例如，获取相机资源并非在 UI 线程中，因为这样会使界面不响应用户时间，所以在 ViewModel 内部提供了 ListenableFuture 异步计算与 LiveData 观察者模式。

2．Python 与机器学习。使用 K-NN 算法对比获取姿态检测结果时，样本理应是使用 Colab 与 MLKit 对（大量）图像进行识别。然而，囿于自身水平所限，没有很好的完成这一点。

3. ***on a null object reference*** 可以说是开发过程中最容易犯的错误了，尤其是在并发编程的背景下，许多过程并不是想象中顺序执行的，所以在获取对象值之前，一定要判断是否为 null 以及是否为缺省值，例如字符串空串。

4. Gradle 无缘无故开始报错，例如找不到 R 文件，有时候需要删除自动生成文件，有时候需要重建项目等等。

5.2. 心得体会

虽然本设计我而言是一个相当大的挑战；不过同时，它也使我在巩固学习的过程中对移动应用开发有了新的认知。这其中，有一部分是此前只是略有耳闻的核心思想，也有一部分是在课堂学习从未注意到的设计细节。例如，CameraX 的操作逻辑，函数式接口与 Lambda 表达式，配置更改与生命周期，OOP 思想在移动应用平台开发的应用等等。甚至包括观察者模式、适配器模式、MVVM 架构等诸多概念（设计原则），都在本例中运用得淋漓尽致。

当我在翻阅《软件设计与体系结构》时，可以用一句“初闻不知曲中意，再闻已是曲中人”表达我震撼又喜悦的心情。同时，写程序时，经常，为了一个功能更改，代码必须改来改去，也是《软件工程》中管理不善的反面教材。

还有许多知识随着时间的演变，了解愈发深入。司空见惯的东西，在不断使用的过程中，逐渐变得陌生又迷人。例如，APK 构建工具 Gradle 对我们既熟悉又陌生。在第一次下载 Google 或其他开源代码时，很容易遭遇无法编译的情况，常常是因为 Android SDK Platforms 或 Android SDK Tools 版本不一致或插件冲突造成的。对这类型导致的标红报错，我深有体会。此外，还有一些依赖库和最新

特性需要手动修改 Gradle。例如，要想使 APP 兼容可绘制矢量图形（Vector Drawable)，以减小 APP 因转化 jpg 文件而消耗的内存占用，就可以在 build.gradle 添加：

```
vectorDrawables.useSupportLibrary = true
```

又如，XML 代码区有一处常常因为开发工具自动补全而被忽视的细节——命名空间。例如，对于同一个属性“android:src”，当冠以不同的命名空间“tools:src”，则不会依托安卓核心框架编译，仅用作设计预览使用。而注明命名空间，则是在 xmlns（XML Namespaces）。

再如，XML 文件中的布局视图（Views）在构建项目、注入活动（Layout Inflation）时，系统将在内存自动生成一个视图对象的层次结构（View Object Hierarchy Tree）。资源类（R class）构建视图标识码与视图对象的映射；用户可以委托 findViewById()搜索视图对象。课堂上，我们惯例将视图对象作为 Activity 类的成员变量（或称为字段，Field）并在 onCreate()中初始化，就是因为重复搜索可能导致软件响应延时。

同时，也有从陌生，到熟悉的过程。当我初次使用 Databinding 时，觉得有一些怪诞；但是，在亲手实践、时间检验中，我愈发觉得它有优越性。一方面，Android Studio 的自动补全给予了莫大的帮助；另一方面，可能由于团队合作或其他因素，id 值可能重名，这就导致编译失败。而在 Databinding 中，一个 binding 对应一个布局，就避免了这个问题。起初，我也不习惯 ConstraintLayout。我钟爱线性布局（LinearLayout），因为几乎只有它可以用权重，设计优雅规整的适应性布局。但是，构建并渲染层次过于复杂的布局会使系统加载缓慢。在深入了解约束布局，尤其是基准（Baseline）、重心偏移（Bias）、比率（Ratio）、连锁（Chain）后，我开始对它“刮目相看”。

另外，在课程设计的推动下，我的学习方式也除课堂教学、书本学习外，慢慢变得越来越开阔。例如，Google Dev 谷歌开发者文档，Udacity 谷歌官方合作的 Kotlin 实验教学视频，机器学习论文阅读，GitHub 各种依赖与开发库。优秀的插件与依赖库不胜枚举，其中，Timber、DebugDB、superSp/RulerView、EazeGraph 是我印象最为深刻的，救人于水火之中的。

最后的最后，感谢各位老师，也感谢谷歌以及各位开发者前辈。