10195101428-刘议临-面向移动终端的可穿戴动态心电图的智能监测应用

【原文对照报告-大学生版】

报告编号：9ff7013886148e31　　检测时间：2023-04-14 09:53:41　　检测字符数：60832

作者姓名：刘议临　　所属单位：华东师范大学

检测结论：

全文总相似比 **2.77%** = 复写率 1.42% + 他引率 1.35% + 自引率 0.0% + 专业术语 0.0%

其他指标：

自写率：97.23%

高频词：应用,心电,使用,数据,进行

典型相似文章：　无

疑似文字图片：0

指标说明：

复写率：相似或疑似重复内容占全文的比重　　他引率：引用他人的部分占全文的比重

自引率：引用自己已发表部分占全文的比重　　自写率：原创内容占全文的比重

典型相似性：相似或疑似重复内容占全文总相似比超过30%　　专业术语：公式定理、法律条文、行业用语等占全文的比重

相似片段：

总相似片段 50

期刊：2　　博硕：21　　综合：1

外文：0　　自建库：0　　互联网：26

检测范围：

中文科技期刊论文全文数据库　　中文主要报纸全文数据库　　中国专利特色数据库

博士/硕士学位论文全文数据库　　中国主要会议论文特色数据库　　港澳台文献资源

外文特色文献数据全库　　维普优先出版论文全文数据库　　互联网数据资源/互联网文档资源

高校自建资源库　　图书资源　　古籍文献资源

个人自建资源库　　年鉴资源　　IPUB原创作品

时间范围：1989-01-01至2023-04-14

原文对照

颜色标注说明：

- 自写片段
- 复写片段（相似或疑似重复）
- 引用片段（引用）
- 专业术语（公式定理、法律条文、行业用语等）

事•盛啤范犬擘
East China Normal University
本科生毕业论文
而向移动终端的可穿戴动态心电图
的智能监测应用
Intelligent Monitoring Application
of Wearable Dynamic
Electrocardiogram for Mobile
Terminals
姓名：刘议临
学 号： 10195101428
学 院： 软件工程学院
专 业： 软件工程
指导教师：王丽苹
职称：副教授
2023年4月
目录

面向移动终端的可穿戴动态心电图的智能监测应用

摘要：

近年来，心血管疾病的患病率持续上升，已成为居民死亡的首位原因。动态心电监测是尽早发现心血管疾病的重要手段。随着小型化的可穿戴式动态心电监测设备的流行和移动设备的普及，移动端心电监测应用的需求愈发明显。另一方面，动态心电图的大量数据对心电信号自动分析算法的需求也更高，加之深度学习模型的快速发展，涌现出了许多用于动态心电分析的智能算法。

本项目基于已有的动态心电智能分析算法，结合移动应用开发技术，设计并实现了一款面向移动终端的可穿戴动态心电图的智能监测应用。文中首先对项目的背景与意义进行了介绍，然后分析了相关技术并确定了技术选型，接着完成了应用的需求分析，确定了心电图、分析报告、设备管理等需求。之后，按照Flutter特有的Riverpod架构模式划分了应用的各个模块。在开发阶段，基于LibTorch框架对已有的模型进行了移植，然后基于Flutter框架进行了跨平台开发，使应用支持Android和iOS两大主流移动操作系统。最后进行了应用测试，确认应用功能清晰完备，界面简洁美观，易于患者使用。

关键词：移动终端，可穿戴设备，动态心电图，深度学习

Intelligent Monitoring Application of Wearable Dynamic Electrocardiogram for Mobile Terminals

Abstract:

Heart disease has overtaken all other causes of death in the population in recent years due to an increase in prevalence. A crucial tool for the early diagnosis of cardiovascular disorders is ambulatory electrocardiogram (ECG) monitoring. The demand for mobile ECG monitoring applications is increasing significantly as a result of the acceptance of smaller wearable Holter monitoring devices and the proliferation of mobile devices. On the other hand, the abundance of ambulatory ECG data has increased the need for automatic ECG analysis algorithms, and numerous artificial intelligence ambulatory ECG analysis algorithms have evolved as a result of the quick development of deep learning models.

Based on the existing dynamic ECG artificial intelligence analysis algorithms and mobile application development technologies, this project designed and implemented a wearable dynamic ECG intelligent monitoring application for mobile terminals. The context and relevance of the project are first discussed, followed by an analysis of relevant technologies and a choice of technologies. The requirement analysis of the application is then finished, and the requirements for ECG, analysis reports, and device management are determined. Then, the application's modules were split up using Flutter's Riverpod architecture pattern. The existing model was ported with the LibTorch framework during the development phase, and then cross-platform development using the Flutter framework was done to make the application compatible with the two most popular mobile operating systems, Android and iOS. Last but not least, testing of the program was done to ensure that it is comprehensive and functional, with a user-friendly interface for patients.

Keywords: mobile terminal, wearable device, dynamic electrocardiogram, deep learning

1、绪论

1.1可穿戴动态心电监测应用的背景与意义

1.1.1国内外心血管疾病现状

根据中国国家心血管病中心出版的《中国心血管健康与疾病报告2021》[1],目前中国人口心血管疾病的发病率和患病率均处于持续上升阶段，心血管疾病已成为居民死亡的首位原因；如图1-1所示，2019年，中国农村、城市心血管疾病分别占疾病死因的46.74%和44.26%。心血管疾病给社会和居民带来的经济负担日益加重，且杀伤力越来越强。

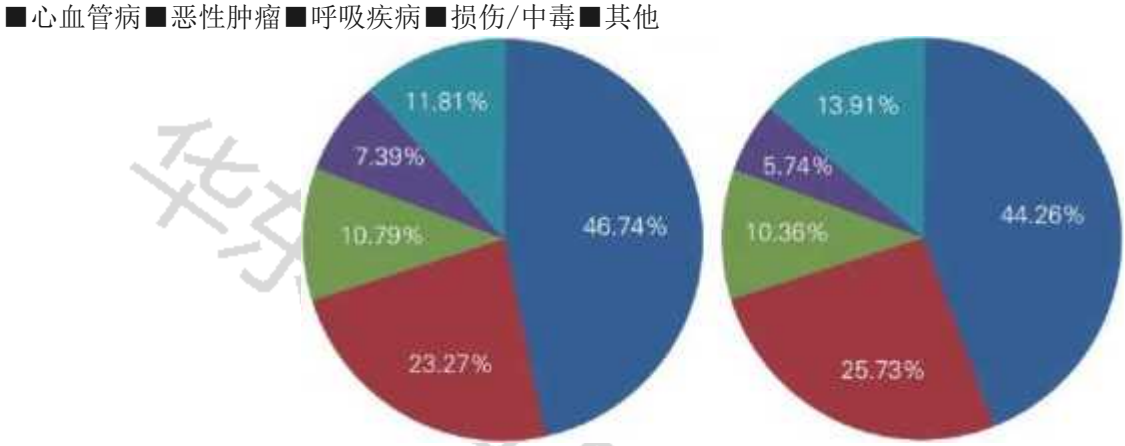


图1-12019年中国城乡居民主要疾病死因构成比［1］

Figure 1-1 Proportion of major causes of death among rural and urban in China in 2019

不仅如此，根据世界卫生组织的统计[2]，心血管疾病也是全球的头号死因，每年死于心血管疾病的人数多于任何其它死因；2019年，估计有1790万人死于心血管疾病，占全球死亡总数的32%。

同时，近年来新型冠状病毒（COVID-19）的爆发也加剧了心血管疾病的危害。证据显示，既往合并心血管疾病的患者更容易在新型冠状病毒感染后发展为重症患者，死亡风险更高，体征和症状更容易恶化［3］。

1.1.2心电监测的用途与原理

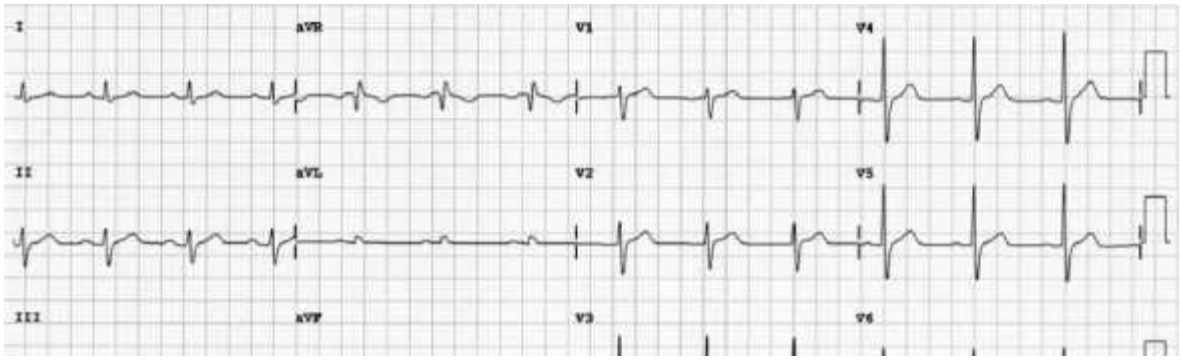
尽早发现心血管疾病非常重要，这样就可以及时采取措施，减少疾病对患者健康的威胁(2)。在各种心血管疾病的诊断方法中，心电图（Electrocardiogram，缩写为ECG）是最常用、最基本、最重要的一种，具有非侵入性、诊断快速、成本低廉、广泛可用等诸多优点［4］。

心电图是对心电信号的记录。通过放置在皮肤上的电极可以检测到心脏的电活动，将电压与时间的关系绘制为二维图像就得到了心电图。

1.1.3常规心电图与动态心电图

心电图有两种主要类型：常规心电图(StandardECG)和动态心电图(HolterECG、DynamicECG或AmbulatoryECG)。

常规心电图，也被称为静息心电图(RestingECG)，是在患者保持静止状态时记录的心电图。常规心电图的记录需要在专业的医疗机构中进行，患者通常处于平躺状态，在四肢和胸部表面安装电极，电极检测心脏产生的电信号并将其传输到心电图仪器。常规心电图通常包含12个导联气一次记录10秒，并打印在心电图纸上，如图1-2所示是一张10秒12导联的常规心电图。



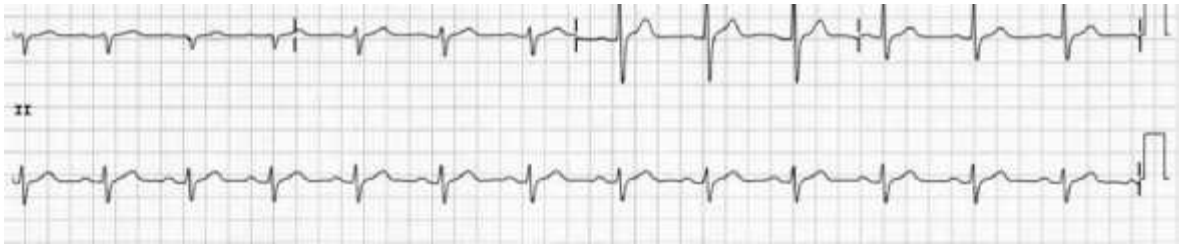


图1-2一张10秒12导联的常规心电图

Figure 1-2 A 10-second 12-lead standard ECG

动态心电图则是在患者进行日常生活活动时记录的心电图。动态心电图的记录可以在几乎任何环境进行，只需要患者佩戴动态心电记录仪(Holtermonitor)即可。动态心电记录仪是一种便携式可穿戴设备，可在较长时间内(24小时以上)记录心脏的电活动，有助于检出非持续性心律失常。统计显示，动态心电图在临床诊断中对心肌缺血及各种心律失常事件的检出率都明显高于常规心电图［5］。1*

1.1.4可穿戴动态心电监测应用的意义

动态心电图的数据量远远超过常规心电图，在显著提升了诊断精度的同时，也大幅提高了人工诊断的难度。人工进行大量数据的诊断识别，不仅使得医生过于疲劳、容易出错，也难以进行实时监测。于是，可以进行动态心电自动分析的可穿戴动态心电监测应用的需求就显而易见了。应用可以基于自动化的心电分析算法快速、准确地处理心电信号，为专业医疗人员节省时间，提高医疗服务的效率，降低医疗成本。使用可穿戴动态心电监测应用也更方便对患者进行实时监测，通过对心电数据进行实时分析可以及时发现患者的异常状况，为患者提供随时随地的医疗监护，降低心血管疾病对患者健康的威胁。

1.2相关技术与应用现状

1.2.1动态心电自动分析技术现状

动态心电自动分析是指在采集到的动态心电信号的基础上，通过对其处理提取表征心脏状态的波形信息和特征参数，获取心脏工作状态的相关信息，然后利用这些特征信息分析、判别心电信号类型及所对应的疾病类型或健康水平，进而对心脏状态和健康状况进行预测［6］。

为了进行动态心电信号的自动分析，已经提出过许多相关算法。比如最著名的Pan-Tompkins算法［7］可以实时提取出心拍的位置，被广泛用于实时心率检测。U-Net模型［8］作为一种生物医学图像分割模型也可以用于心拍提取，虽然无法实时给出结果，但在准确度等方面相较Pan-Tompkins等传统算法表现更好。基于ResNet［9］则可以训练心拍分类模型，用于自动识别每个心拍是否正常以及异常类型。

1.2.2国内外可穿戴动态心电监测应用现状

当前国内外已有不少可穿戴动态心电监测应用，其中一些使用了人工编写的传统心电分析算法［10-15］，包括自适应双阈值法［12］、Pan-Tompkins算法［14］以及一些原创的或基于已有算法改进的未命名的算法等；另一些可穿戴动态心电监测应用则使用了深度学习模型［16-21］。

使用了深度学习模型的这些可穿戴动态心电监测应用按其整体架构可以大致分为两类：一类是在移动端只对数据进行简单统计处理，完整的算法模型则部署于服务端［16-17］，患者如果想获取完整的分析报告，则需要将数据上传至服务端后，等待服务端分析完成；这种应用架构的主要缺点是，患者能在移动端立即获取的结果较少，而完整分析结果可能因为网络质量差、服务端算力不足等原因而有较大延迟；另外，虽然患者不需要在自己的设备上运行相关算法，节省了一定开销，但上传与下载数据消耗的电量与流量成本不可忽视。另一类应用则采用了边缘计算架构，将深度学习模型部署在移动端，数据分析直接在移动端进行［18-21］，这样可以缩短延迟，节省带宽，并节省了高算力服务器的成本；然而，已有的少数此类应用都只实现了极简单的功能，旨在以演示应用验证模型正确性，并没有进行完整的应用开发，如图1-3所示；此外，这些简单的演示应用均只进行了单一操作系统的开发。

(a)陈桂琛的应用［18］(b)刘荟的应用[19] (c)ZhanpengJin等的HeartToGo[21]

图1-3一些仅为验证模型而开发的简单演示应用

Figure 1-3 Some demo apps only for model validation

1.3本项目的主要工作

本项目与1.2.2节中所述的最后一类相似，也就是采用边缘计算架构，将深度学习模型部署在移动端，数据分析直接在移动端进行。本项目与上文提及的已有的同类应用的主要差别在于其选题来源于导师的心电图课题的子项目，是在已经基于PyTorch完成了相关算法模型［22］的情况下进行的，在项目之初就已有较好的人工智能算法模型的技术支撑。于是，相比于偏重算法研究而在实际的应用开发方面较为欠缺的其他项目，本项目的主要工作在于将已有的算法模型与移动应用开发技术进行整合，以实现一个较为完整的可穿戴动态心电监测应用，目标是最终投入到实际的生产环境中供患者使用。

在项目过程中，首先对应用的需求进行了分析，明确了应用的功能性需求和非功能性需求。然后为了满足需求进行了技术选型，明确了项目要使用的技术栈。之后对应用的整体架构和各个模块进行了设计。在开发阶段，完成了已有算法的移植与优化，以及应用程序本身的开发。最后，对应用进行了测试，以确保其可以满足需求。

1.4论文组织结构

本文分为7个章节，各章节的主要内容如下：

1、 绪论介绍了本项目的背景与意义、相关技术与应用现状、本项目的主要工作、本文的组织结构。

2、 相关技术介绍对本项目使用的相关技术进行了介绍，包括Flutter相关的移动应用开发技术和算法相关的一些技术。

3、 可穿戴动态心电监测应用的需求分析对应用进行了需求分析，明确了应用的目标用户、功能性需求、非功能性需求、项目可行性。

4、 可穿戴动态心电监测应用的设计对应用的整体架构和各个模块以及数据库进行了设计。

5、 可穿戴动态心电监测应用的开发介绍了应用的开发过程，包括开发环境与工具、算法的实现与移植、应用程序各个模块的实现等。

6、 可穿戴动态心电监测应用的测试包含项目的测试环境、测试数据、测试方法、测试用例等。

7、 总结与展望总结了本项目的主要工作，对未来的工作进行了展望。

2、相关技术介绍

2.1Flutter跨平台应用程序开发框架

2.1.1使用跨平台应用程序开发框架的意义

要开发一个面向移动终端的可穿戴动态心电监测应用，首先要考虑的是该应用支持哪些移动平台。根据2023年3月的数据［23］，在中国的移动平台操作系统占有率中，Android系统以76.15%名列第一，iOS系统以23.28%位居第二，这两个最主流的移动平台操作系统占据了绝大部分市场份额。因此，本应用开发的目标平台选定为Android和iOS。

Android和iOS开发的各种技术可以分为两大方向：原生应用程序开发和跨平台应用程序开发。构建传统的原生

应用程序需要维护两个不同的代码库，分别为Android和iOS平台编写代码，通常意味着需要分别使用Kotlin/Java和Swift/Objective-C来编写两份高度相似的代码。而跨平台应用程序开发框架则可以通过一套代码库同时为Android和iOS平台构建应用程序，降低了开发人员的学习成本，缩减了开发时间，提高了开发效率。

2.1.2.1.2Flutter 简介

Flutter［24］是一个由Google开源的跨平台应用程序开发框架，仅通过一套代码库就能构建精美的、原生平台编译的跨平台应用。

Flutter对Android和iOS平台均有良好支持，此外还支持Windows,Linux、macOS、Web等平台［25］。基于Flutter框架进行开发不仅可以避免为Android和iOS平台分别编写大量相似的代码，还可以方便未来将部分不依赖于移动端特性的功能（比如历史心电展示等）的相关代码在其他平台（比如Web端）复用。

2.1.3Flutter与其他框架的对比

Flutter并不是唯一一个支持Android和iOS平台的跨平台应用程序开发框架。在本项目的选型过程中也考虑了其他几个经常被与Flutter进行比较的同类框架，包括ReactNative［26］、Xamarin［27］、IonicCreator［28］。关于这几个框架，最先被对比的是其热度。以近一年的Google趋势作为标准，对比结果如图2-1所示。

从图中可以看出，Flutter和ReactNative的热度远远高于Xamarin和IonicCreatoro热度高意味着其社区更加活跃，生态更为优秀，更容易找到相关的资料和技术支持，也更容易利用社区已经开发过的包，这些对于项目开发来说都是非常重要的。因此，

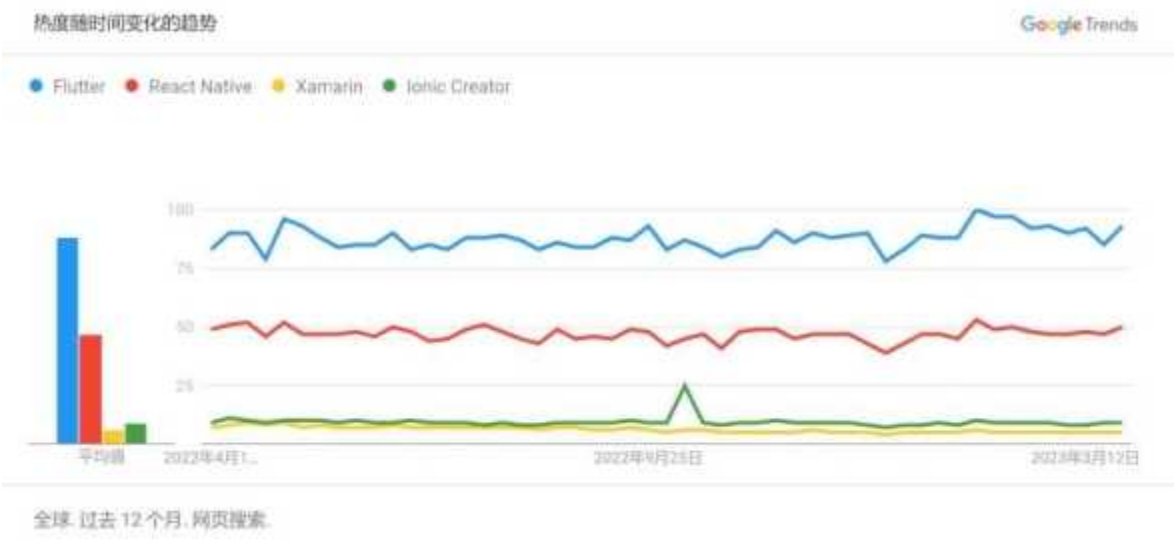


图2-1Flutter与其他框架的Google趋势

Figure 2-1 Google Trends for Flutter and other frameworks

Xamarin和IonicCreator在本项目的选型过程中最先被排除。

之后需要对比的是Flutter和ReactNativeo从热度上来看，两者的热度都比较平稳，Flutter长期保持在ReactNative的两倍左右，这意味着Flutter应当被优先考虑，但还不足以作为决定性理由。因此，需要进一步对比这两个框架的特性。

ReactNative是由Meta（前身为Facebook）于2015年开源的跨平台应用程序开发框架。正如其名称所暗示的，ReactNative和React（一个流行的Web框架）的关系十分密切，这既是优点也是缺点。从优点的方面来说，ReactNative非常适合已有React开发经验但没有移动平台开发经验的开发者快速上手，也适合将已有的基于React的Web项目迁移至移动端，但这两点对于本项目来说都没有明显意义。另一方面，由于和React的紧密联系，基于ReactNative的应用需要使用JavaScript和CSS编写，代码会在JavaScript引擎下解释执行，并通过序列化消息与本机代码桥接通信以渲染原生组件。JavaScript本来就不是一种以性能见长的语言，额外的桥接翻译层更使得其性能较之原生应用程序相去甚远。

作为对比，Flutter是Google于两年后（2017年）开源的，其设计上明显借鉴了ReactNative，与其一样使用响应式风格的界面编写方式。主要差别在于Flutter是编译成原生代码运行，直接接管并控制屏幕上的每一个像素，由此可以避免使用JavaScript桥接导致的性能问题。

除了上述的性能优势之外，Flutter还在热重载、IDE集成、内置调试工具等方面相较其他框架更为优秀。最终，基

于以上所有考虑，本项目选择了Flutter作为开发框架。

2.1.4 Dart 语言

Dart[29]是由Google开源的编程语言，专门针对用户界面（Userinterface，缩写为UI）的创造进行优化，可编译为移动平台、桌面平台的二进制文件和Web平台的Javascript代码。Dart最初于2011年发布，原本是打算捆绑于Chrome浏览器来替代JavaScript，但后来由于各种因素而未能成功。在Flutter开发团队选择了Dart作为开发语言之后，两者的关系便愈加紧密，Dart也因此得到了更多的关注，并顺应Flutter框架的需求进行了许多更新迭代。时至今日，Dart与Flutter可以说是处于完全捆绑的状态，基于Flutter框架进行开发几乎是学习并使用Dart语言的唯一目的，使用Dart语言也是基于Flutter框架进行开发的必要条件。

Dart是一种多范式的语言，支持函数式、命令式、面向对象、反射式的写法，语法风格近似于C++与Java的混合，支持抽象类、泛型、类型推断、自动内存管理、空安全等其他语言中较为常见的特性，也有Mixin、Extension方法等其他语言中不太常见的特性。因篇幅限制，本文不便对Dart的诸多特性进行完整介绍，这里仅对后文必须涉及，且与其他语言有明显差异的部分特性进行简单说明。

2.1.4.1外部函数接口

外部函数接口（Foreignfunctioninterface，缩写为FFI）是一种机制，指用一种编程语言编写的程序可以调用另一种编程语言编写的功能。比如在C++中可以指定extern"C"来要求编译器在调用约定、名字重整等方面遵循C语言的约定，从而实现与C语言之间的双向FFI。这一特性在C++中也被称为语言链接（Languagelinkage）。

Dart作为一种专精于UI开发的语言，在其他方面的生态较为薄弱。作为一种弥补方式，Dart实现了与许多其他语言之间的FFI，比如在Web上可以与JavaScript互相调用，在macOS和iOS上可以与Swift或Objective-C互相调用，在Android、Windows,macOS、Linux上可以与Kotlin或Java互相调用，以及对本项目来说最重要的在除Web以外的所有平台可以与C语言直接进行交互，而无须像ReactNative等框架那样通过Java等中间层进行间接调用，这使得Dart与C语言之间的FFI十分高效。由于C++代码也可以通过指定extern"C"来编译为与C语言格式相同的接口，这意味着本应用的部分功能可以直接使用C++编写，从而充分利用C++的性能、生态、跨平台性等优势。

在项目的开发过程中，使用C++编写了两部分代码供DartFFI调用，分别是Pan-Tompkins算法和基于LibTorch的人工智能算法，在后文会详细介绍。

2.1.4.2.1.4.2Isolate

Isolate是Dart中的一种介于线程与进程之间的概念。由于该术语没有常用的中文译名，故本文中直接使用英文原名。在Dart程序中，所有的代码都在isolate中运行。每一个isolate都有独立的运行线程和独立的堆内存，从而确保isolate之间互相隔离，无法互相访问状态。相比于常规的线程，这样的实现并不会共享内存，所以也不需要担心互斥锁和其他锁等问题。相对地，由于无法与其他isolate共享可变对象，isolate之间的通信必须使用消息机制。

Dart程序有一个默认的主isolate，在Flutter应用中也称为UIisolate，因为UI的更新都是在主isolate中进行的。在本应用中，数据库查询等可能较为耗时的操作都是放在其他isolate中异步执行的，以避免阻塞UIisolate导致UI卡顿。

2.1.5项目使用的Flutter包

尽管FlutterSDK本身已经包含了大量的基础组件，但是在实际开发中，还是经常需要额外使用官方或社区开发的Flutter包来实现一些特定的功能。本项目所使用的Flutter包总计有上百个，其中一些包的使用比较简单，比如URLLauncher包只是用来在各个平台提供在浏览器中打开指定的URL的功能；而另一些包则在项目开发的各个阶段有着广泛的影响，比如Isar包的特性直接影响了本项目数据库的设计、实现与测试。

由于项目使用的Flutter包非常多，而且作用分散在项目的各个阶段、各个模块，在此处一一介绍会使得本节过于冗长且难以理解，本文在后续章节涉及到相关Flutter包时再对各个包分别进行说明。另外，所有包的相关文档均可在https://pub.dev/上找到，由于数目众多，本文不分别列出其链接。

2.2项目使用的Python包与C++中的对应替代

2.2.1将Python代码迁移至C++的原因

本项目使用的用于心电信号分析的人工智能算法最初是使用Python编写的。为了在面向移动终端的应用中使用这些算法，考虑了各种可能的技术方案。

2.2.1.1移动端直接执行Python代码的方案

最简单的一个方案是设法在移动端直接执行Python代码。这种方式在移动应用开发中很少被使用，主要是基于

时间上和空间上这两个方面的顾虑。从时间上来说，Python语言的执行效率较低，在设备性能相对较差的移动端表现得更为明显；不过本应用中人工智能算法的执行时间占比并不大，仅仅是每10分钟调用一次，一次执行数秒而已，所以这一缺点也并非不可接受。从空间上来说，将Python语言的运行环境打包进应用会导致应用的安装包体积更大；但此算法所需要的Torch模型文件本身就已有60多MB，相较之下Python的运行环境所额外占据的大小并不是很明显。因此，常见的关于性能或安装包大小的顾虑并不能构成否定本方案的充分依据。之后考虑了在移动端执行Python代码的各种方式。

PyQtDeploy[30]、QPython[31]、Termux[32]等工具可以用于在移动端执行Python代码，但未提供可供Java、Dart等语言调用的接口，因此无法在本项目中使用。

Kivy[33]提供了Python与Java、Objective-C的交互，但是缺少对于PyTorch的支持，而这一依赖难以替代，所以Kivy也无法在本项目中使用。

Chaquopy[34]提供了PyTorch的支持。尽管最高只支持1.8.1版本，但算法原本所依赖的版本甚至比这更旧，所以PyTorch的使用没有问题。但是，Chaquopy仅支持Android平台，而不支持iOS平台，所以无法完全满足本项目的需求。

BeeWare[35]在Chaquopy之上增加了对iOS平台的支持，但是需要将整个应用都使用Python编写，这样就无法通过仅在少数部分调用Python来避开本小节最初提到的性能问题，因此也不是可行的方案。

综合以上调查结果，可以认定移动端直接执行Python代码的方案并不可行。因此，必须将原本使用Python编写的算法迁移至其他语言。

2.2.1.2使用Dart实现相关算法的方案

由于应用主体使用Dart编写，下一个考虑的方案是使用Dart实现相关算法。社区已经编写了PyTorchMobile、PyTorchLite等包来帮助将PyTorch模型迁移至Flutter框架，但这些包都仅支持对模型的直接调用，而本项目所使用的算法除了直接调用模型的部分之外还包含一些额外的处理步骤，这些步骤在缺少相关支持的Dart中较难实现，因此这一方案也不可行。

2.2.1.3使用C++实现相关算法的方案

本项目最终采用的方案是将已有的使用Python编写的算法迁移至C++。使用C++编写的内容可以通过DartFFI在移动端使用，而且C++代码具有跨平台性，这两点完全契合本项目的需要。在迁移之前对Python代码进行了大幅度精简，消去了许多非必要的依赖项，但是仍然有一些关键的依赖项无法消除，因此需要在C++中找到对应的替代。

2.2.2.2.2PyTorch 与 LibTorch

PyTorch[3]是一个开源的Python机器学习库，也是算法中最核心的依赖项，必须找到可能的替代。

一个可能的选择是使用PyTorch官方推出的PyTorchMobileoPyTorchMobile目前尚处于Beta测试阶段，但已经被广泛验证可用，并且同时适用于Android和iOS平台。然而PyTorchMobile为Android和iOS平台提供的接口并不相同，使得在多平台使用时需要编写两套不同的代码。由于算法在Python中的实现较为繁琐，单次迁移的工作量已经很大，因此分别针对Android和iOS平台进行两次迁移的方案的工作量难以接受。综上所述，PyTorchMobile本身并不是一个可行的替代方案。不过，由于PyTorchMobile在实现上是为LibTorch封装了Java和Swift接口，其作为依赖项的存在仍然为本项目提供了很大帮助，在后文介绍具体实现时会详细说明。

其他的替代方案包括PyTorch官方提供的LibTorch、阿里巴巴的MNN、腾讯的NCNN等。MNN和NCNN提供了对于PyTorch模型迁移的支持，但是这两个框架的接口与PyTorch不同，需要对算法进行较大的重构。相比之下，LibTorch本身是PyTorch的C++前端气其设计理念即是Python前端很棒，应该尽可能使用。所以，LibTorch在编写时就考虑了在设计、命名、约定和功能方面紧密模拟PyTorch。虽然两个前端之间偶尔可能存在差异，但官方保证将Python算法移植到C++的努力应该完全在于翻译语言功能，不修改功能或行为。因此，经过综合考虑，本项目最终选择了LibTorch作为PyTorch的替代。

2.2.3NumPy 与 NumCpp

NumPy[37]是Python的一个很常用的科学计算库，算法中使用其进行了许多矩阵操作。严格来说，NumPy这一依赖项并非完全不可能消除，其许多功能在Py-Torch/LibTorch中也有提供，剩下的一些则可以通过手动编写相关算法来完成。但是，由于存在NumCpp[38]这一完美的替代，因此在算法重构过程中保留了NumPy的依赖，这也使得算法实现的过程更加顺畅。

NumCpp，顾名思义，是NumPy的C++实现，接口设计紧密贴合NumPy（类似LibTorch之于PyTorch）。NumCpp提供了其矩阵与LibTorch张量之间的双向转换，这使得其可以在本项目中作为NumPy的C++替代。

2.2.4pytest 与 Catch2

pytest1[39]是一个Python的测试框架。原始版本的算法并未包含单元测试，在重构过程中，为了保证不破坏算法的正确性，基于pytest编写了简单的测试用例。

Catch2[40]是C++中的一个轻量的测试框架。在算法迁移至C++的过程中，为了保证两边的实现一致，在C++中编写了与Python中几乎相同的测试用例。由于本项目的C++部分没有编写复杂的测试用例的需求，所以没有选择GoogleTest、BoostTest等功能更强大但依赖也更重的测试框架。此外，本项目的Pan-Tompkins算法部分也基于Catch2进行了测试。

2.2.5nlohmann::json

本项目中使用了多种编程语言，为了方便不同语言中对数据的读写，数据交换格式统一为了JSON。在Python中，标准库中已经提供了JSON的支持，因此不需要额外的依赖。而在C++中，标准库中并没有提供JSON的支持，因此需要选定使用哪一个第三方JSON库。由于项目中对JSON的需求仅在测试阶段，只是用于读取少量测试数据，因此性能不是一个重要的考量因素。最终，出于接口设计简洁易用的优势，本项目选择了nlohmann::json[41]这一JSON库作为C++中的JSON解析器。

’pytest的官方名称即为全小写形式，即使在位于句首的情况下也是如此。

3、可穿戴动态心电监测应用的需求分析

3.1应用面向的用户群体

本应用的目标用户是佩戴可穿戴动态心电信号监测设备的院外患者。由于心血管疾病的发病率随着年龄增长而增加［1］，有动态心电监测需求的患者也以中老年人为主。除了常规的功能性需求分析之外，本项目也结合用户群体的特征进行了额外的非功能性需求分析以及相关的设计和实现。

3.2应用的功能性需求分析

本应用的整体用例情况如图3-1所示。为了避免图像过于复杂，图中进行了一定程度的简化。

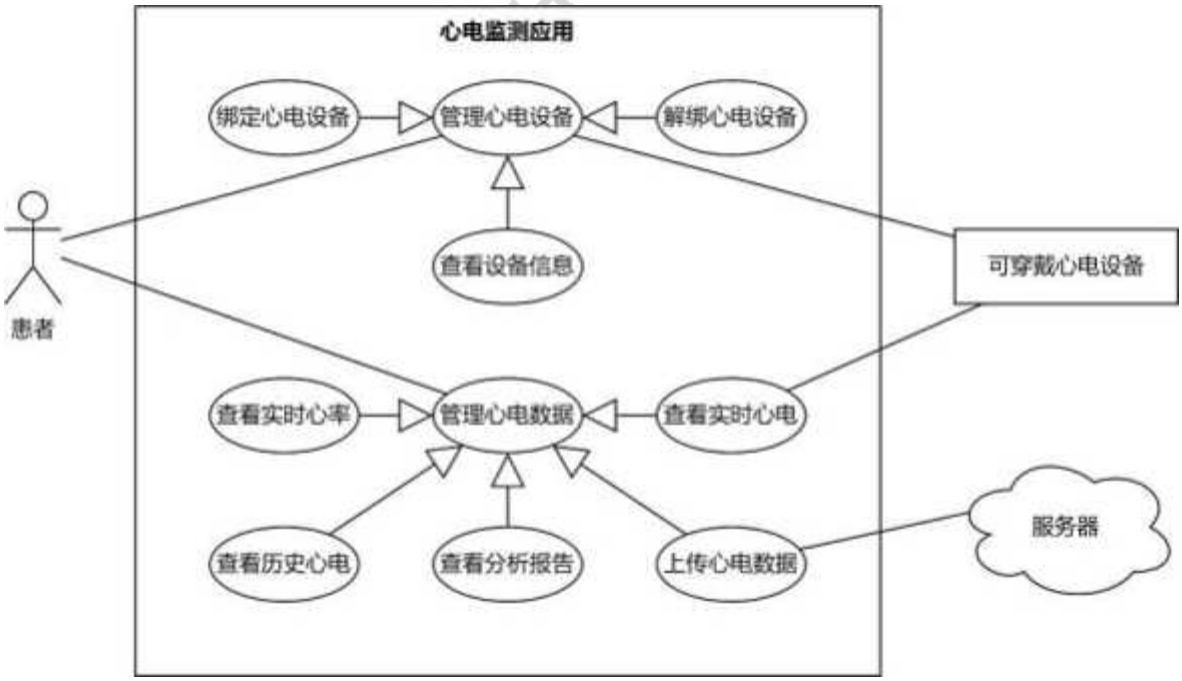


图3-1应用的用例图

Figure 3-1 Use case diagram of the app

3.2.1心电设备相关需求分析

3.2.1.1绑定心电设备

通常而言，新用户使用本应用的第一步是绑定其预先购买的动态心电信号监测设备（以下简称心电设备）。绑定心电设备的用例详情如表3-1所示。

3.2.1.2解绑心电设备

患者在绑定心电设备后，可能会因为各种原因需要解绑心电设备。解绑心电设备的用例详情如表3-2所示。

表3-1绑定心电设备的用例详情

Table 3-1 Use case of binding a device

名称	绑定心电设备
主要参与者	患者
次要参与者	心电设备
前置条件	未绑定心电设备
后置条件（成功时）	已绑定心电设备
触发条件	患者尝试绑定心电设备
主要事件流	1.应用列出可绑定的心电设备，并定时刷新； 2.患者指定要绑定的心电设备； 3.应用绑定心电设备。
扩展事件流	1a.没有可绑定的心电设备：应用提示患者没有可绑定的心 电设备，并定时刷新。
异常事件流	3a .绑定心电设备失败：应用提示患者绑定心电设备失败。

表3-2解绑心电设备的用例详情

Table 3-2 Use case of unbinding the device

名称	解绑心电设备
主要参与者	患者
前置条件	已绑定心电设备
后置条件	未绑定心电设备
触发条件	患者尝试解绑心电设备
主要事件流	1.应用解除心电设备的绑定。

解绑心电设备是应用程序单向的动作，不需要心电设备的参与或者同意，患者完全可以在心电设备已无法正常工作的情况下成功执行此用例。因此，该用例也没有异常事件流，应用保证其必定成功。

3.2.1.3查看心电设备信息

患者在绑定心电设备后，可以查看其相关信息。查看心电设备信息的用例详情如表3-3所示。

表3-3查看心电设备信息的用例详情

Table 3-3 Use case of viewing the device information

名称	查看心电设备信息
主要参与者	患者
次要参与者	心电设备
前置条件	已绑定心电设备
触发条件	患者尝试查看心电设备信息
主要事件流	1.应用展示心电设备的名称、型号、信号强度、电量等信息。
扩展事件流	1a.设备已断开连接：应用提示患者设备已断开连接，并仅展示心 电设备的名称、型号等已知信息，不展示电量等无法获知的信 息。

可以展示的心电设备的信息是根据心电设备的通信协议中能提供的信息，以及这些信息是否对于一般患者有意

义来决定的。例如，心电设备的信号强度、电量等信息对于一般患者来说是有意义的，而心电设备的内部序列号等信息则对于一般患者来说是无意义的，即使心电设备的通信协议中提供了这些信息，应用也不应该展示，以保证界面的简洁易用。

3.2.2实时心电数据相关需求分析

3.2.2.1查看实时心率

在连接了心电设备后，应用可以展示患者当前的心率。查看实时心率的用例详情如表3-4所示。

3.2.2.2查看实时心电图

作为一个动态心电图监测应用，查看实时的动态心电图是应用最核心的功能之一。查看实时心电图的用例详情如表3-5所示。

心电图在易用性、性能等非功能性需求方面要求较高，但在功能性需求的方面是相对简单的。

表3-4查看实时心率的用例详情

Table 3-4 Use case of viewing the real-time heart rate

名称	查看实时心率
主要参与者	患者
次要参与者	心电设备
前置条件	已连接心电设备
触发条件	患者尝试查看实时心率
主要事件流	1.应用展示患者当前的心率。
扩展事件流	1a. 心电设备未连接：应用提示患者心电设备未连接。 1b. 心率信息未就绪：应用提示患者心率正在检测中。

表3-5查看实时心电图的用例详情

Table 3-5 Use case of viewing the real-time ECG

名称	查看实时心电图
主要参与者	患者
次要参与者	心电设备
前置条件	已连接心电设备
触发条件	患者尝试查看实时心电图
主要事件流	1.应用展示患者的实时心电图。
扩展事件流	1a.心电设备未连接：应用提示患者心电设备未连接。

3.2.3历史心电数据相关需求分析

3.2.3.1查看历史心电图

患者可以查看过去某个时间点的心电图。查看历史心电图的用例详情如表3-6所示。

患者尝试查看历史心电图有两种情况：其一是患者直接指定某个时间点进行查看，这种情况相对较少；其二是患者看到分析报告中提及的某个时间点，然后查看该

表3-6查看历史心电图的用例详情

Table 3-6 Use case of viewing the historical ECG

名称	查看历史心电图
主要参与者	患者
触发条件	患者尝试查看历史心电图
主要事件流	1.应用展示患者的实时心电图。
	1a. 当前时间点无数据：应用提示患者当前所选的时间点无历史心 电数据。

扩展事件流	1b. 患者指定另一个时间点：应用展示患者指定的时间点的历史心 电图。

时间点的心电图，这种情况较为常见。
另外，在1b中，患者可能指定绝对时间点（如10:24）或相对时间点（如当前查询点的5秒前）。
3.2.3.2查看心电分析报告
应用内置心电分析算法模型，可以自动在本地对历史心电数据进行分析。患者可以查看过去某一时间段的分析报告。
表3-7查看心电分析报告的用例详情
Table 3-7 Use case of viewing the ECG analysis report

名称	查看心电分析报告
主要参与者	患者
触发条件	患者尝试查看心电分析报告
主要事件流	1.应用展示患者的心电分析报告。
扩展事件流	1a.患者指定另一个时间段：应用展示患者指定的时间段的分析报 告。

3.2.3.3自动上传心电数据
在患者已绑定医疗账号的情况下，应用会自动上传心电数据至服务器。一方面是为了备份，另一方面也方便患者对接的医生进一步分析诊断。自动上传心电数据的用例详情如表3-8所示。
表3-8自动上传心电数据的用例详情
Table 3-8 Use case of automatically uploading the ECG data

名称	自动上传心电数据
主要参与者	定时任务管理器
次要参与者	服务器、患者
前置条件	患者已绑定医疗账号
后置条件（成功时）	数据已上传至服务器
触发条件	定时触发
主要事件流	1.应用上传本地未上传的历史数据至服务器。 2.应用通知患者已完成上传。
异常事件流	1a.上传失败：通知患者上传失败，患者可以选择重试或忽 略（不进行选择则默认忽略）。重试则重新执行1；忽 略则用例终止。

3.2.4其他需求分析
除上述需求外，还有登录账号、调整应用设置、查看应用版本等其他需求。这些需求在许多应用中通用，在需求分析、设计等方面都没有值得特别说明的地方，因此不再赘述。
3.3应用的非功能性需求分析
3.3.1应用的易用性需求分析
因为大部分中老年的患者没有丰富的移动设备使用经验，所以设计一个简单直观、易于使用的应用程序很重要。界面设计应避免使用复杂的元素，以尽量简洁清晰为目标。
此外，由于一般患者通常不会具备专业的医学知识，所以应用程序内应该尽可能地避免使用过于晦涩难懂的专业术语。同时，对于应用内无法避免使用的部分术语应该提供明显且易于理解的解释，以便患者理解其含义。
3.3.2应用的无障碍需求分析
应用需要考虑到患者在视力等方面的障碍，尽可能地保证患者能够在不同的环境下使用应用程序。例如，应用程序应该提供对大字号、粗体字、高对比度、深浅主题等内容的支持。
3.3.3应用的性能需求分析

由于动态心电监测的需要，本应用会长期驻留在系统后台运行。因此，应用程序的性能对于患者体验至关重要。应用程序应该尽可能地保持较低的内存占用和较低的电量消耗，以保证患者的使用体验。

3.3.4应用的兼容性需求分析

应用程序需要能够在各种主流移动设备上运行。这包括对于Android和iOS两大移动操作系统的支持，以及对于各种型号的移动设备的支持。在应用的设计与实现中，需要充分考虑到不同环境可以为应用提供的可用功能有所差别，以保证应用程序在不同环境下的兼容性。

3.4项目的可行性分析

3.4.1项目的技术可行性分析

本项目所需的技术主要包括移动应用开发和动态心电智能分析两大部分。关于移动应用开发，本项目预期使用Flutter框架，开发者此前已有相关的经验，具备技术可行性。关于心电智能分析，项目所属课题此前已有相关的算法成果［22］，具备技术可行性。

3.4.2项目的经济可行性分析

本项目在经济方面的成本主要在于动态心电记录仪的获取。由于课题已与相关医疗公司达成合作，获取设备的成本较低，具备经济可行性。

3.4.3项目的法律与道德可行性

本项目在法律和道德方面的考量主要在于用于测试的心电数据的获取。由于互联网上可以找到许多经志愿者同意的公开的心电数据，因此本项目可以使用这些数据进行测试，具备法律和道德可行性。

3.5应用开发的重点与难点

3.5.1动态心电智能检测算法的移植

本应用所使用的动态心电智能检测算法虽然已经开发完成，但其编写时仅简单验证了算法的可行性与正确性，并未以方便调用为目的进行合适的封装，也没有考虑19

在移动终端运行。在项目的开发过程中，需要对算法的实现进行重构以方便应用调用，并且需要将算法及其依赖项迁移至移动平台环境。算法的实现较为复杂，且外部依赖较多，因此对其的移植是开发的一个重难点。

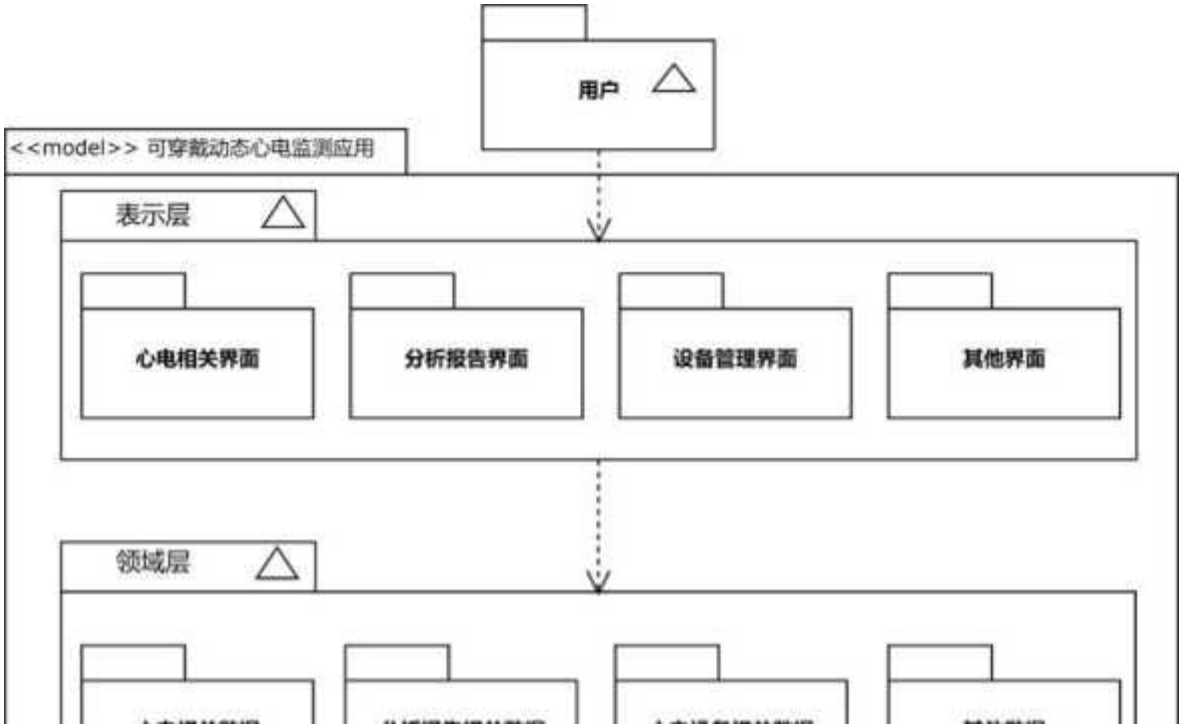
3.5.2应用的用户体验设计

本应用的目标用户是佩戴动态心电监测设备的院外患者，以中老年人为主。因此，应用的用户体验设计上需要力求简单易用，不能仅以实现基本功能为目标。如何将患者所需的功能以合适的形式进行展示，是设计与开发时需要特别考虑的一个问题。

4、可穿戴动态心电监测应用的设计

4.1应用的整体架构设计

本应用的整体架构如图4-1所示。因空间有限，此处对部分模块进行了合并与省略，后续小节会展开介绍。



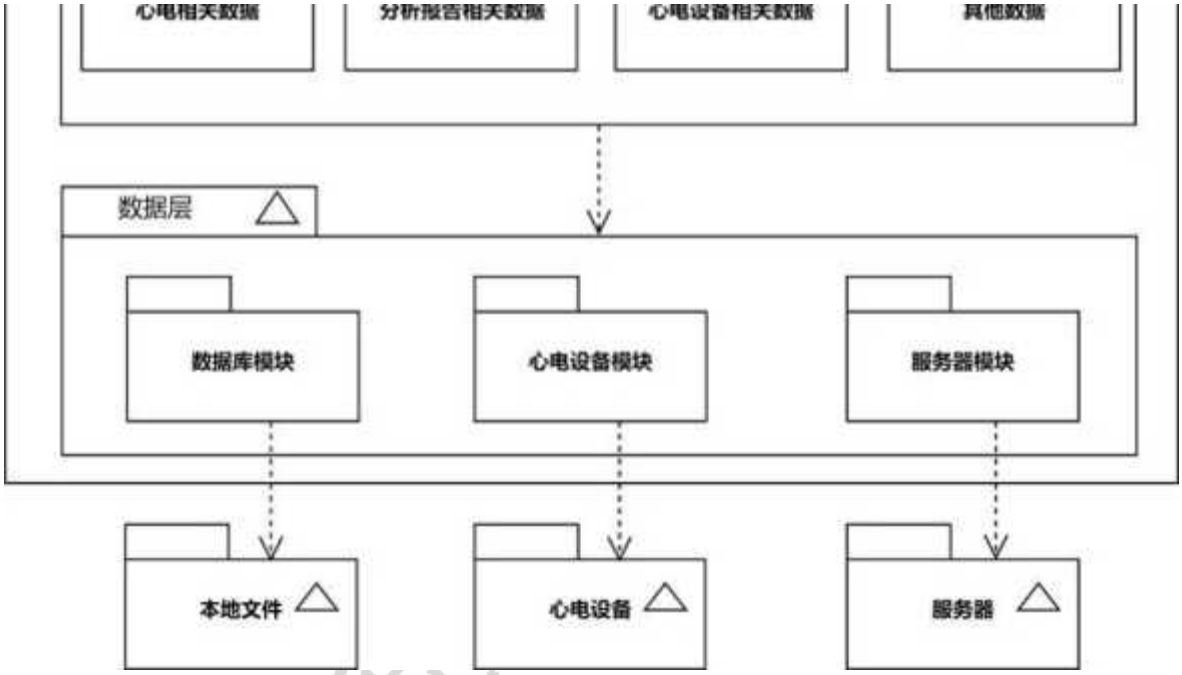


图4-1应用的整体架构

Figure 4-1 Architecture of the app

本应用使用了Riverpod架构模式［42］，其与MVC等传统模式有较大差别，且由于提出较晚而知名度不高，因此本节顺带对该模式进行简单介绍。

在设计应用程序时选择正确的应用架构至关重要，过于简陋的设计会使得代码整体缺乏组织性，过于繁复的设计则会阻碍代码的修改迭代。自最经典的MVC架构以来，已有许多流行的应用架构被相继提出。其中一些对原始的MVC架构进行改动后仍沿用其名称，导致MVC这一术语的含义愈加模糊；另一些变体则被命名为MVP、MVVM、MVC+S等MV*,或是如CleanArchitecture等另起炉灶。这些经典的架构模式很难原样照搬至Flutter框架，即使强行在Flutter中进行实现也只会使得代码结构不伦不类。在探索Flutter适用的应用架构的过程中，社区已经提出了许多新模式，如BLoC、Stacked等。而在本应用的架构设计中，采用的则是Andrea于2022年初提出的Riverpod架构模式。

Flutter曾经有一个流行的状态管理包Provider，该包也是上述的BLoC、Stacked等架构模式的基础。后来，由于Provider包的设计逐渐暴露出一些难以解决的问题,其开发者将该包进行了大幅重写，并因其与旧版本不兼容而改名为Riverpod（对Provider中字母的重组）重新发布，Andrea提出的该架构模式因基于Riverpod包而命名为Riverpod架构模式。

该架构模式由三层或四层组成，从上至下分别是表示（Presentation）层、可选的应用（Application）层、领域（Domain）层、数据（Data）层。

4.1.1表示层

表示层类似于MVC中的View和Controller，或是MVVM中的View和View-Model，以及Android应用架构指南[4]中的UI层。该层包含用户可见的UI组件以及相关的特定于某个组件的状态与交互逻辑。由于Flutter采用了响应式的设计思想，UI本身与其对应的状态的关系极为密切，因此在Riverpod架构中，将MV*中的Model以外的两层进行了合并。在本应用中，该层包含实时心电界面、历史心电界面、分析报告界面等内容。

4.1.2应用层

应用层类似Android应用架构指南中的领域层（两边对领域层这一术语的使用不一致），并且与其一样是可选的。这是因为并非所有应用都具有复杂的业务逻辑，也并非所有业务逻辑都需要被提取出来以便重用。在Riverpod架构中，如果不需要应用层，则可以直接省略这一层。在本应用中，由于应用的业务逻辑较为简单，因此并未使用该层。

4.1.3领域层

领域层类似于MV*中的Model。该层包含领域模型，即应用程序中的数据及其相关的方法。由于领域层在各种架构模式中被广泛使用（尽管命名可能不同），此处不作详细介绍。在本应用中，该层包括心电数据、心拍数据、应用设置数据等模型。

4.1.4数据层

数据层类似Android应用架构指南中的数据层。该层包含与外部数据源通信的相关代码，负责将领域模型与底层的数据源的实现细节进行隔离，将从数据源获取的原始格式的数据（如JSON等）转换为领域层中的领域对象（应用中自定义的数据类），有时也执行数据缓存等操作。在本应用中，该层包含与数据库、心电监测设备、服务器进行通信的相关代码。

4.2应用各个模块的设计

4.2.1心电模块的设计

心电模块的整体架构如图4-2所示。因Riverpod架构模式未指明外部函数接口和后台定时任务属于哪一层，本图将其绘制于已有层次之外。

心电模块可以大致划分为实时心电模块和历史心电模块这两个子模块，不过两者在各个层级的重合部分较多，在最终的实现中也有不少共享的代码，因此一并归于心电模块。

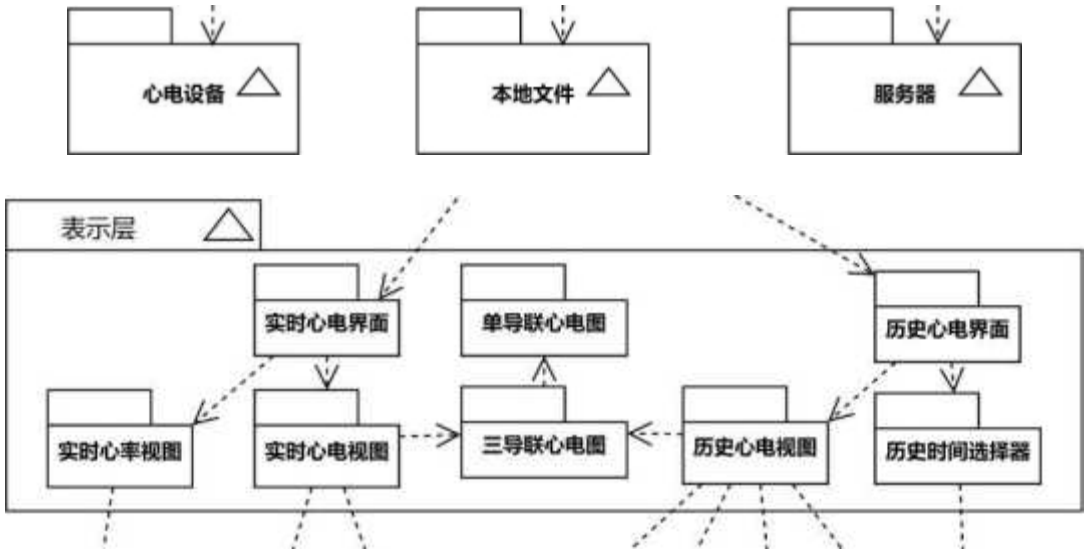
4.2.1.1表示层

心电模块的表示层中为用户提供了两个心电相关的界面，分别是实时心电界面和历史心电界面。实时心电界面包含实时心电视图和实时心率视图，历史心电界面则包含历史心电视图和历史时间选择器。实时心电视图与历史心电视图均为心电图的显示，因此提取其共享部分作为三导联心电图UI组件。然后，由于三个导联的心电图之间也高度相似，在三导联心电图之中再提取出单导联心电图UI组件，以免编写重复代码。

4.2.1.2领域层

心电模块的领域层中，实时心率数据由实时心率视图读取；实时心电数据由实时心电视图读取，也被用于心率数据的获取，并被存入数据库；心电图相关设置数据由实时心电视图和历史心电视图读取（实际上有不同的设置项，但在该设计层次无须细分）；历史心电数据和历史心拍数据由历史心电视图读取；历史时间由历史时间选择用户△

<<model>>心电模块



城ˆ一/ \ //: ’’、、«ˆ»
’ ■/ / I \ ,
/ \ 丁7 * \

实时心率数据	实时心电数据	心电图相关设置	历史心电数据	历史心拍数据	历史时间

Pan-T算法	1 、 % 1 / \ ’土 \ : : \ <<写>> \ 1 ’	1	1
	1卖>> 、、 V <读>> v <读>> V <1卖>>		

数据层	\ ''、、 \ /	数据自动清理	数据自动上传

心电设备模块	1	1为G L 数据库模块	<<H>> 、：、 了5读>> < <^]^> > V 服务器模块

图4-2心电模块的架构

Figure 4-2 Architecture of the ECG module

器进行修改，由历史心电视图读取，并被用于对历史心电数据和历史心拍数据的查询。

4.2.1.3其他层

Pan-Tompkins算法（图中简写为Pan-T算法）模块以C++语言实现，通过Dart

FFI被调用，用于通过实时心电数据得到心拍位置，进而得到心率数据；另外，Pan-Tompkins算法所给出的心拍位置也会作为未知类型的心拍存入数据库，以供历史时间选择器使用（跳转至上一个或下一个心拍的时间），为了保证图像清晰而并未在图中标出。数据自动清理模块和数据自动上传模块都是后台定时运行的自动任务，对用户而言并不可见；数据自动清理模块负责清理较旧的数据，避免应用占用空间过多;数据自动上传模块负责将数据库中的数据上传至服务器作为备份。

4.2.1.4数据层

心电模块需要用到数据层中的所有三个子模块。心电设备模块用于提供实时心电数据；数据库模块用于实时心电数据的写入、心电相关设置与历史心电数据的读取、来源于Pan-Tompkins算法的心拍位置的写入、历史心拍数据的读取；服务器模块用于数据上传。

4.2.2分析报告模块的设计

分析报告模块的整体架构如图4-3所示。

4.2.2.1表示层

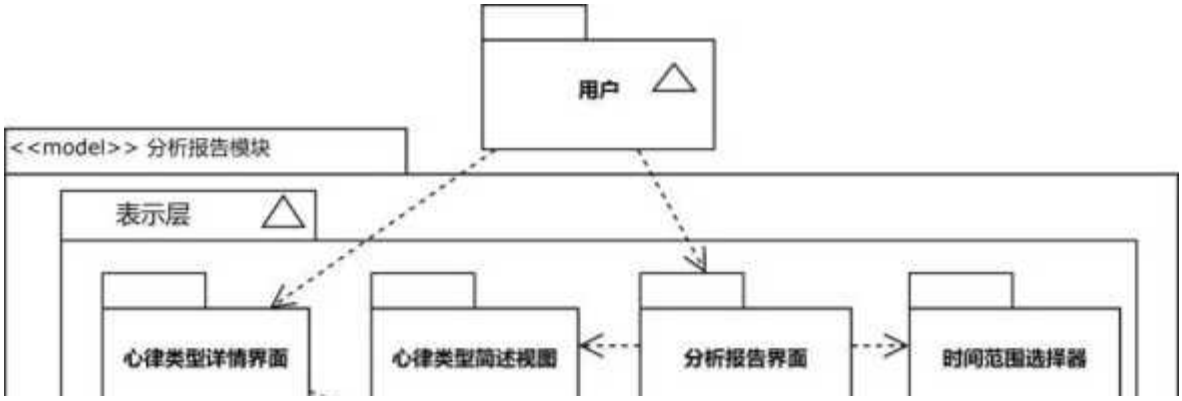
分析报告模块的表示层包含两个界面，分别是分析报告界面和心律类型详情界面。分析报告界面用于展示对各个心律类型的分析结果的总结，包含心律类型简述视图和用于控制分析时间段的时间范围选择器。点击某个心律类型简述视图所在区域后，会打开心律类型详情界面，详细说明该心律类型的情况，包括该心律类型每次出现的时间，点击时间可以跳转至历史心电界面的对应时间。

4.2.2.2领域层

分析报告模块的领域层包含三个数据模型，分别是分析报告数据、历史心拍数据和历史时间范围。三个数据都同时被心律类型简述视图和心律类型详情界面所读取，因为前者只是后者的折叠形式。历史时间范围还被时间范围选择器展示和设置。历史时间范围即当前所查看的分析报告包含的时间范围，被读取后用于历史心拍数据的查询参数。历史心拍数据即过去的每个心拍的位置和对应的心律类型。分析报告数据是基于历史心拍数据的分析结果，比如平均心室率等。

4.2.2.3其他层

数据自动清理模块和数据自动上传模块与历史心电中的作用相同，不进行重复说明。数据自动分析模块每10分钟自动运行一次（这是所用模型允许的最短输入数据的时长），读取过去10分钟的历史心电数据，输入智能检测算法，将得到的心拍分类结果写入数据库。该算法在后台自动运行，而非延迟到用户查询分析报告时再运



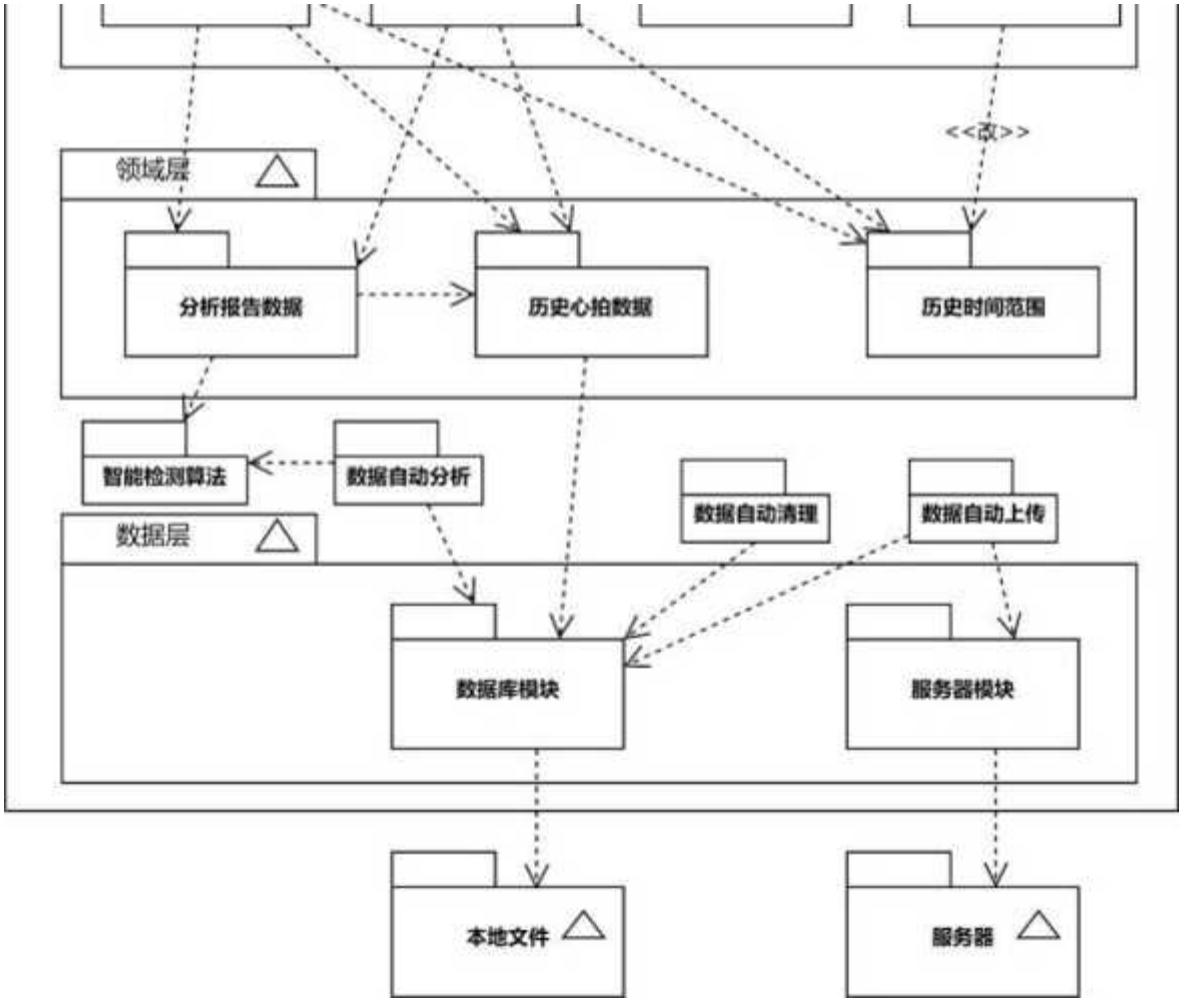


图4-3分析报告模块的架构

Figure 4-3 Architecture of the analytics module

行，原因之一是为了能将分析结果及时上传至服务器；原因之二是分析每10分钟的数据需要数秒时间，虽然不长，但用户选取较长范围时仍需要一定时间的等待，提前分析并存入数据库可以作为缓存；原因之三是历史心拍数据比较容易裁切和拼接，方便分段存入数据库和进行任意时间段的查询。而基于历史心拍数据分析得到的分析报告数据则因为计算量小而没有必要预先缓存，且较难进行多时间段的拼接和分离，因此在用户查询分析报告时才进行分析。

4.2.2.4数据层

分析报告模块需要用到数据层中的数据库模块和服务器模块。数据库模块用于历史心电数据的查询、历史心拍数据的写入与查询。服务器模块用于数据上传。心电设备模块在分析报告模块中并不需要，用户甚至可以在已经断开心电监测设备的连接时查看分析报告，并不会对本模块的功能产生影响。

4.2.3设备管理模块的设计

设备管理模块的整体架构如图4-4所示。

4.2.3.1表示层

设备管理模块相比上述模块较为简单。虽然还有其他更简单的模块（如日志模块等），但因为在设计阶段几乎没有相关工作，故其他模块不在此章介绍。设备管理模块的表示层只包含一个面向用户的界面，即设备管理界面。设备管理界面实际上由两个不同的界面组成，在未连接设备的情况下是连接新设备界面，已连接设备的情况下则是已连接设备界面。由于两个界面的出现条件对立，因此将其合并为设备管理界面。在已连接设备界面中，点击解绑设备则会断开连接，切换至连接新设备界面。在连接新设备界面中，选择要连接的新设备则会进行连接，切换至已连接设备界面。

4.2.3.2领域层

设备管理模块的领域层包含三个数据模型，即可连接设备列表、当前心电设备ID、当前心电设备信息。当前心

电设备ID为字符串或null，用于标识应用当前连接的设备，在应用程序启动时会从数据库读取，改变时也会写入数据库，设备管理界面在连接新设备界面和已连接设备界面之间的切换判定也是基于当前心电设备ID是否为null。当前心电设备信息仅在心电设备已连接，即当前心电设备ID不为null时有效，由已连接设备界面使用，其内容包括当前设备的信号强度、剩余电量等信息。可连接设备列表仅在连接新设备界面中有使用，为其提供可连接设备的名称与ID等信息，在连接后也会将所连接设备的ID更新至当前心电设备ID。

4.2.3.3数据层

设备管理模块需要用到数据层中的数据库模块和心电设备模块。数据库模块用于连接的心电设备的ID的读写。心电设备模块用于与心电设备的通信和可供连接的设备的扫描。

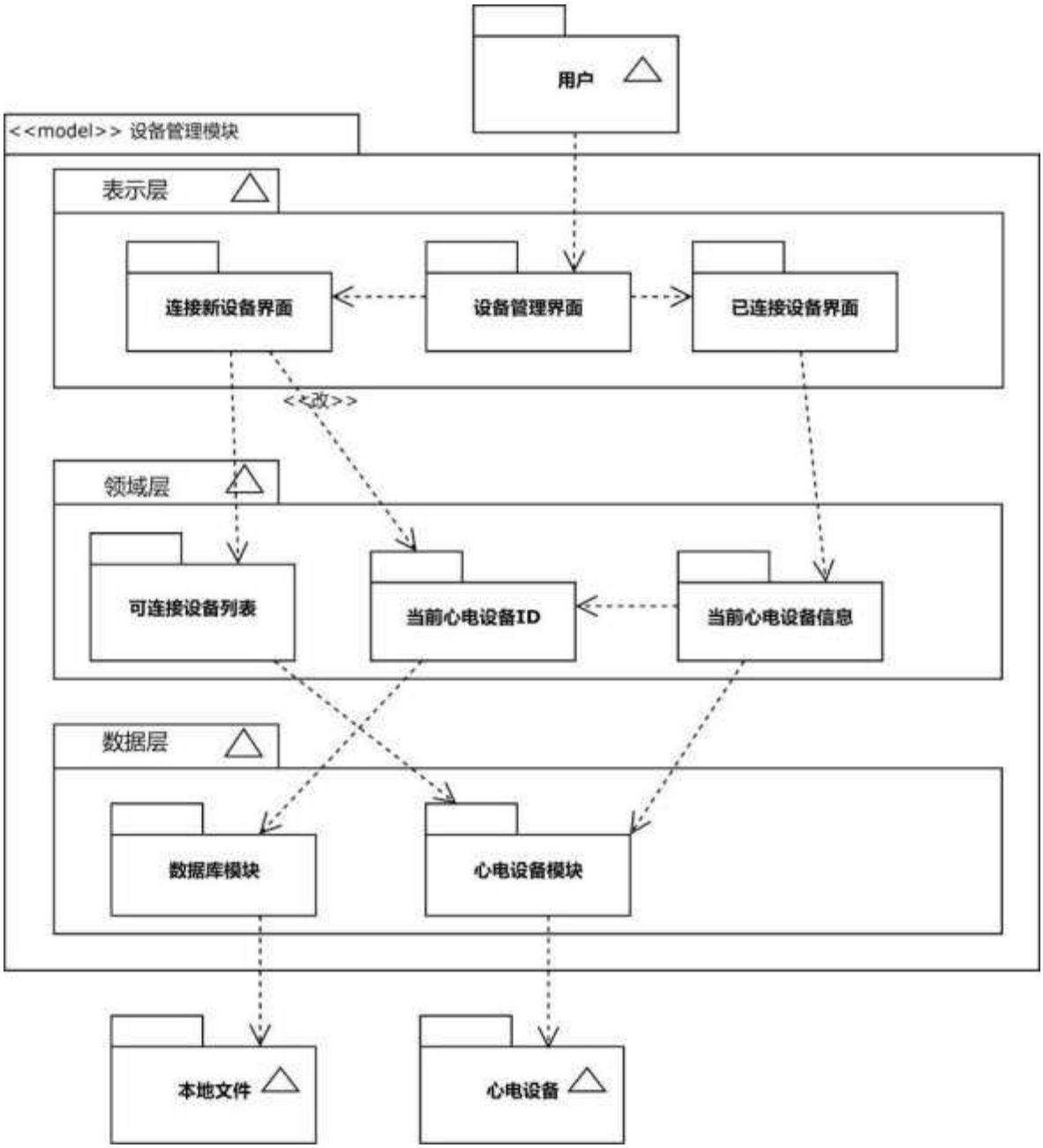


图4-4设备管理模块的架构

Figure 4-4 Architecture of the device module

4.3应用的数据库设计

应用的数据库分为两部分，存储简单数据的SharedPreferences数据库和存储复杂数据的Isar数据库。前者主要用于存储应用的设置，后者主要用于存储历史心电数据与分析报告结果。

4.3.4.3.1SharedPreferences 数据库的设计

4.3.1.4.3.1.1SharedPreferences 数据库介绍

SharedPreferences是Flutter的一个包，提供了简单键值对的存储功能，可以视为一个键值型的非关系型数据库。其在Android平台上基于同名的SharedPreferences功能，在iOS上则基于NSUserDefaults功能。

SharedPreferences仅支持少数几种数据类型，即int、double、bool、string以及List<String>。键值型数据库的读写非常简单，不需要进行特别的设计。本项目对该数据库的设计主要在于将各种需要存储的类型映射为其支持的类型。

4.3.1.2Duration类型的存储

Duration类型表示时间长度。由于应用内对时间的各种操作最多只需要毫秒精度，所以应用在需要将Duration类型的数据存入SharedPreferences时会将其转换为毫秒数以整数格式进行存储。在读取时，应用会将毫秒数转换为Duration类型。

4.3.1.3Color类型的存储

Color类型表示颜色。应用在需要将Color类型的数据存入SharedPreferences时,会将其编码为32位整数以整数格式进行存储。在读取时，应用会将32位整数转换为Color类型。具体的编码方式是将颜色的ARGB值分别存储在32位整数的高8位、次高8位、次低8位和低8位中。

4.3.1.4枚举类型的存储

应用中使用了各种枚举类型，有时会需要将枚举类型存入SharedPreferenceso应用在存储枚举类型时会将其转换为索引值，以整数形式存储。枚举类型常见的存储方式还包括按其名称以字符串形式存储、为每个枚举值赋予自定义值然后按自定义值的类型存储。相比其他方法，直接按索引值存储的优点在于其简单高效；缺点在于已经存在的枚举值不能轻易改动，否则会破坏已有的数据。由于应用内的枚举值设计基本不变，所以该缺点并不明显，可以接受。

4.3.2Isar数据库的设计

4.3.2.1Isar数据库介绍

Isar是Flutter的一个包，提供了一个跨平台数据库。Isar属于非关系型数据库,不过提供了组合索引、ACID语义、事务等功能，可以视为一个关系型数据库的子集。相比常规的关系型数据库，Isar有一些额外的限制（或者说缺少一些功能）影响了本应用对数据库的设计。

当使用Isar来存储数据时，需要对Collection进行操作。Collection可理解为Isar数据库中的表，其包含的数据只能为同一类Dart对象，每个对象则代表了对应数据表中的一行数据。该类对象所对应的类则是这张表的Schema，其中每个字段对应数据库中的一列。与一般的关系型数据库相同的是，Schema中必须要有主键；不同的是，Isar中主键的类型被限制为64位整数值，这一限制导致了本应用对数据库的设计中的一些不同之处。

在查询时，Isar并不像一般的关系型数据库那样在编写SQL语句后自动生成最优查询策略，而是需要手动指定查询方式。Isar中的查询过滤方式分为两种，分别称为Filter和Where，前者执行遍历过滤，后者依靠索引表进行过滤。多个Where子句的过滤结果直接只能进行并集运算，无法进行交集运算。如果需要按多个属性进行过滤，并且还希望使用索引加快速度，就必须使用组合索引。组合索引有一个特殊的限制：主键不能包含在组合索引之中。这一限制也影响了本应用对数据库的设计。

此外，Isar对外键、Join等功能的支持也比较有限，不过这些功能在本应用的数据库设计中本来也不需要，所以不进行过多说明。

4.3.2.2心拍数据的存储

在分析报告数据中，只有心拍数据被存入数据库，基于心拍数据产生的进一步分析结果则只在查询时动态生成。这一方面是因为算法的大部分时间开销在于分割分类模型的运行上，其余部分开销较小。另一方面是因为心拍之外的结果的格式较为复杂，不方便进行合适的数据库设计。心拍数据则格式简单，且存储、查询时可以对多段数据进行拼接、裁剪而不失去太多准确度，所以适合存入数据库之中。

心拍数据的Schema包含3个字段和2个索引。字段中的ID是由Isar管理的自增整数，没有特别用途。另外两个字段分别是DateTime类型的心拍时间和枚举类型的心拍类型标签。对心拍时间进行了单列索引，以方便在查看历史心电时快速检索指定时间范围内的所有心拍。对心拍类型和心拍时间进行了组合索引，以心拍类型为主索引，这个组合索引用于查询指定标签在指定时间范围内的数量等信息。

4.3.2.3心电数据的存储

对心电数据的查询需求较为简单，只有在历史心电界面中需要对指定时间范围内的心电数据进行查询。因此，

也只需要在心电数据的时间上进行索引。

由于没有组合索引的需求，所以可以直接把时间作为主键，以节省额外的索引表的开销，并避免浪费主键所占的空间。由于主键只能为64位整数，所以需要对时间进行编码。考虑到时间只需要毫秒精度，将其编码为了自Unix纪元（UTC时间的1970年1月1日00:00:00）以来经过的毫秒数。这样，心电数据的Schema中不包含任何额外定义的索引，查询时只使用主键自带的索引功能。

除了作为主键的时间之外，一条心电数据中还包含各导联的电压数据。由于导联I、II、III被定义为左臂、右臂、左腿三个点之间的电位差，这三个导联的数据实际上只包含两个差值的信息量，所以只需要存储两个值即可。

5、可穿戴动态心电监测应用的开发

5.1项目的开发环境与开发工具

本项目包含了多种语言的代码的编写，不同语言所使用的开发环境与工具有重叠部分也有不同部分，并且在本地环境与持续集成环境中的配置也有一定差异。

5.1.1通用的环境与工具

项目中的所有代码都使用了Git进行版本控制，项目的源代码托管在GitHub上，并且使用了GitHubActions作为持续集成工具，开发过程中也遵循了GitHub推荐并支持的pullrequest,release等工作流程。Git的提交信息遵循约定式提交(ConventionalCommits)规范，版本号则按照语义化版本(SemanticVersioning)规范进行管理。新版本的发布及版本号的更新基于Google的ReleasePlease工具半自动地进行，该工具同时用于更新日志的自动生成。为了方便使用GitHub,在本地安装并使用了GitHubCLI与GitHubDesktopo

项目的各部分代码均使用Codecov进行测试覆盖率的统计追踪，使用Renovate进行依赖项的自动更新，使用Restyled自动格式化代码，使用CodeFactor进行代码质量的检查，并使用Sentry来自动收集并上报错误信息。另外，各部分代码的编写都借助了GitHubCopilot来提供更智能的代码补全。

5.1.2Dart及Flutter的环境与工具

项目开发基于最新的稳定版的FlutterSDK,在开发过程中其版本进行过几次更新，截止本文撰写时使用的是Flutter3.7.10。DartSDK使用的是捆绑于FlutterSDK中的对应版本，并未单独配置。

开发过程中，在本地使用了IntelliJIDEA作为Flutter开发的IDE,安装了Flutter插件与Dart插件来提供相应的支持，并额外安装了FlutterFreezedSnippets,FlutterIntl等插件来提供更多相关功能。为了获得对移动平台开发的支持，在一台Windows设备和一台macOS设备上分别额外安装了AndroidStudio和Xcode及其对应移动平台设备的模拟器。

在持续集成环境中，基于subosito开发的flutter-action配置了Android和iOS平台的应用安装包的自动构建发布。除FlutterSDK自带的静态分析、代码测试等工具外，额外使用了社区开发的DependencyValidator包来检查依赖项是否有缺少或冗余。

5.1.5.1.3LTeX的环境与工具

在本地安装了TeXLive2023,并使用安装了TEXiFy插件的IntelliJIDEA作为IDE。在持续集成环境中，使用了xu-cheng提供的最新版TEXLive的Docker镜像进行文档的编译，并配置了相应的自动发布。

5.1.4C++的环境与工具

项目中对于Pan-Tompkins算法和基于LibTorch的算法的开发使用了不同的环境。由于开发时主要使用的是Windows系统，所以在Pan-Tompkins算法的开发过程中使用了MSVC工具链。而基于LibTorch的算法则因为Windows平台的LibTorch分发版区分了调试与发布版本而较难使用，所以使用了安装在WSL2中的GCC工具链。两者都使用CLion作为IDE，一个直接安装在Windows系统中，另一个安装在WSL2中然后通过JetBrainsGateway连接。

在持续集成环境中，使用aminya提供的setup-cpp工具来安装必要的依赖。两个仓库中的代码均在最新的Ubuntu环境下使用GCC与CMake工具链进行编译，然后基于Gcovr进行代码覆盖率的统计。

5.1.5Python的环境与工具

对于Python环境的管理使用了Mamba，Conda的一个更优秀的替代工具。在本地安装了Mambaforge用于依赖管理，持续集成环境中则使用provision-with-micromamba工具。Python代码的测试与覆盖率生成基于pytest和pytest-cov工具。本地开发的IDE选择了PyCharm。

5.2项目文件的整体组织结构

5.2.1Git子模块介绍

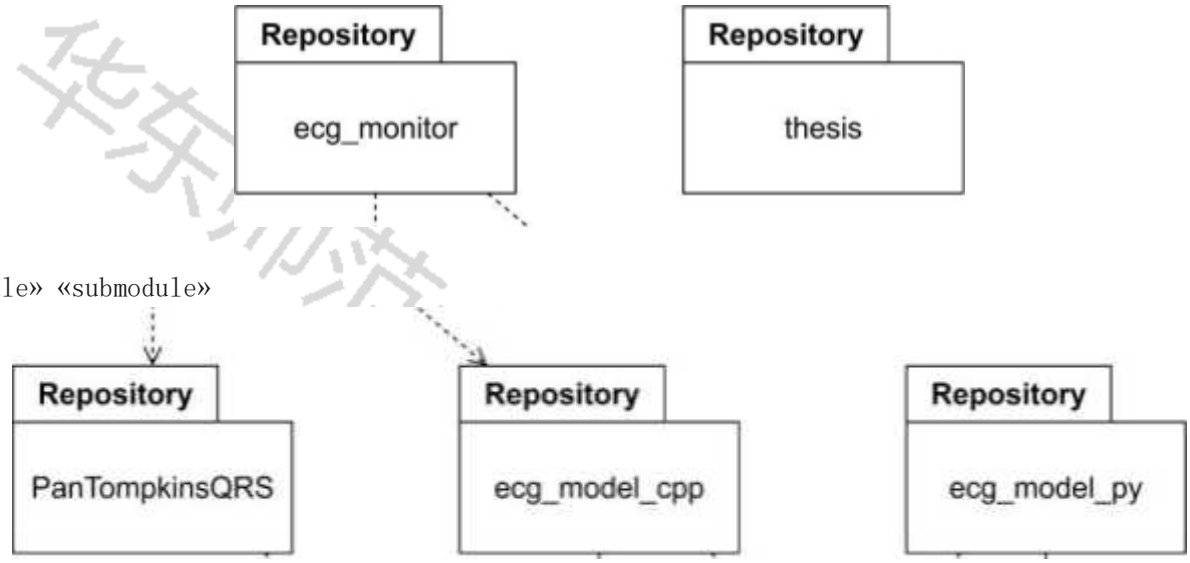
本项目的相关文件基于Git的子模块（submodule）功能进行组织，在此先对Git子模块进行简单介绍。

Git子模块是一个Git中自带的工具，允许将一个Git仓库作为另一个Git仓库的子目录。为了方便项目管理，本项目的内容被拆分到了多个Git仓库，各个仓库之间的一些共享的部分使用了Git子模块来实现复用与同步。

与Git子模块功能类似的工具还有Git子树（subtree）等，但其他工具对于在主仓库修改之后向子仓库进行推送的这一需求都没有像Git子模块这样的良好支持，再考虑到整个项目架构的一致性，本项目在内部各个仓库之间的依赖方式上都选择了Git子模块作为解决方案（对外部仓库的依赖方式则有所不同）。

5.2.2项目包含的Git仓库

本项目在整体上一共由7个Git仓库组成，这些仓库的名称以及仓库之间的子模块关系如图5-1所示。



、、、 ： «submodule» / :
、、、 ■ 、、、«submodtile» ■
«submodule» «submbdule» 、、 ，/ «submbdule»

’4 ▽ / ▽
Repository Repository
*二
图5-1项目包含的Git仓库
Figure 5-1 Git repositories of the project

5.2.2.1 thesis

这是本篇论文所在的仓库。论文本身作为项目文档，也被视为本项目的一部分。该仓库包含论文的LTEX源码，以及论文中引用的图片等资源文件，还有自动格式化、依赖版本管理、自动构建测试、自动发布新版本等持续集成配置。

论文与应用本身的代码是分离的，所以论文的仓库与应用的仓库之间没有任何子模块关系。此外，论文仓库以Git子树的形式包含了论文模板1，没有使用Git子模块是因为基于模板进行编写后只需要直接提交至主仓库，不需要向子仓库推送，这与5.2.1节所述的情况不同。

5.2.2.5.2.2.2ecg_monitor

这是应用的主体仓库。该仓库包含基于Flutter框架编写的Dart代码，以及相关的各种配置文件等。仓库的命名与应用的包名一致，因此按照Dart对包名的相关限制使用了小写下划线的命名风格，而非Git仓库常用的小写连字符。为了保持项目各个仓库之间的一致性，其他仓库（除PanTompkinsQRS外）也都使用了小写下划线的命名风格。

为了方便调用C++代码，该仓库以Git子模块的形式包含了PanTompkinsQRS和ecg_model_cpp两个子仓库，这两个子仓库分别提供了Pan-Tompkins算法和基于LibTorch的心电图智能分析算法的实现。

5.2.2.3PanTompkinsQRS

这是应用中用于实现Pan-Tompkins算法的仓库。该仓库是rafaelmmoreira编写的Pan-Tompkins算法的C语言实现1的复刻（fork），在原始版本的基础上根据项目需要进行了一些修改。出于对原作者的尊重，该仓库的名称与原作者的命名保持一致，因此与项目内其他仓库的命名风格不同。

该仓库被ecg_monitor所包含，并且为了方便测试而将ecg_data包含为了子仓库,后者提供了一些测试用的心电图数据，用作测试时的输入。

5.2.2.4ecg_model_cpp

这是应用中用于实现基于LibTorch的心电图智能分析算法的仓库。该仓库是ecg_model_py仓库的C++实现，使用了LibTorch作为框架。

该仓库被ecg_monitor所包含，且包含了ecg_data作为测试输入数据，还包含了ecg_models作为与ecg_model_py共享的模型文件以及测试预期输出。

5.2.2.5ecg_model_py

这是基于PyTorch实现的心电图智能分析算法的所在仓库。该仓库以算法的原始版本作为初始提交，在此基础上根据项目需要进行了修改。

Python版本的算法并不直接被应用调用，因此该仓库不是其他仓库的子模块。该仓库包含了ecg_data作为测试输入数据，将与ecg_model_cpp共享的模型文件保存在ecg_models,并将PyTorch版本的算法输出也写入ecg_models子模块以便与ecg_model_cpp共享。

’https://github.com/rafaelmmoreira/PanTompkinsQRS

5.2.2.6ecg_data

该仓库存储了各算法仓库用作测试输入的心电图数据，以及从公开数据库下载心电数据并转换格式的相关代码。该仓库中的数据由仓库内的代码写入，被上述三个算法仓库所读取。

5.2.2.7ecg_models

该仓库存储了TorchScript模型文件，以及模型在测试输入下的输出结果。该仓库的内容由ecg_model_py写入，被ecg_model_cpp读取。因为模型文件较大（60多MB），且不被Pan-Tompkins算法所需要，所以该仓库与ecg_data进行了分离。

5.3 Pan-Tompkins算法的实现

本应用为用户提供了实时的心率显示。因为所使用的基于LibTorch的算法无法实时给出心电分割结果，所以有必要另外使用一种实时在线算法来进行心率的统计。

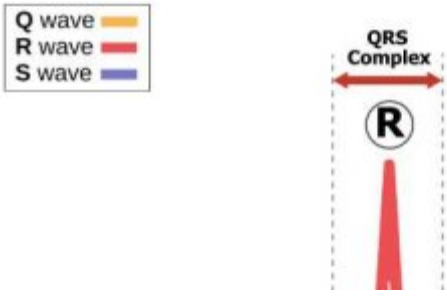
Pan-Tompkins算法［7］可以用于检测心电图中的QRS波群。一次正常的窦性心律的组成如图5-2所示，其中的QRS波群位于心电图中最明显的尖峰处。此功能使得Pan-Tompkins算法非常适合用作心率测量。

该算法是心电分析中最经典的算法之一。于1985年被提出后，有许多人对其进行了各种各样的改进。由于算法的原始版本已经有很高的准确率（原作者给出的统计结果为99.3%）,同时本应用仅将其用作实时心率检测，没有很高的准确率要求，所以出于实现较为简单的优势而直接使用了原始版本的算法，而非其他人的改版。

由于Pan-Tompkins算法的应用非常广泛，多年以来，已经有大量开发者用各种语言各种方式对其进行了实现。为了不进行无意义的额外工作，本项目在Pan-Tompkins算法的实现过程中并非从零开始编写，而是基于已有的实现进行了修改。经过检索对比，发现rafaelmmoreira为该算法编写的C语言实现1的代码质量较高，以MIT协议开源，并提供了充分的注释文档以方便理解，因此本项目以该版本实现为基础按项目需要进行了一些修改。

首先，将该实现由C语言迁移至了C++。除了将在C++中无法通过编译的特性（比如动态数组大小）进行了修改外，还将一些内容的写法改为了C++中惯用的写法，包括将使用宏定义的常量改为constexpr、将fopen相关的写法替换为std::ifstream等。

之后，对该实现的输入输出方式进行了修改。在原始版本的算法中，程序开始运



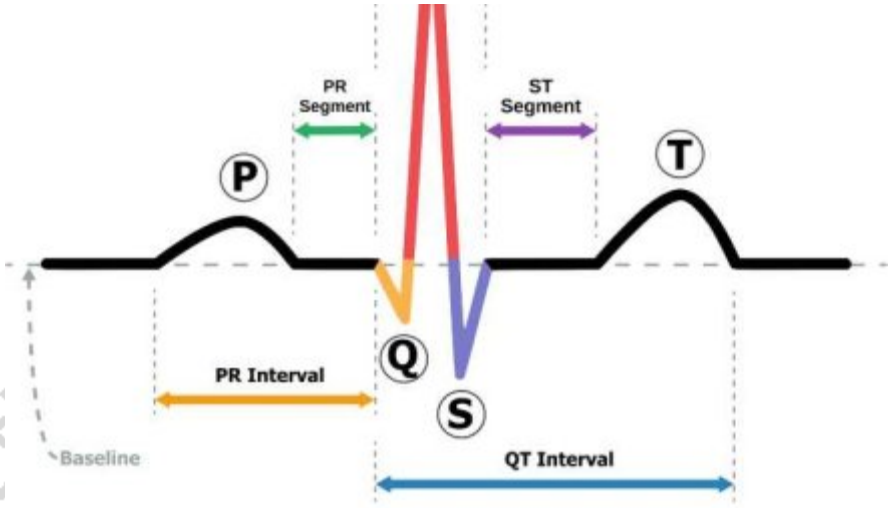


图5-2正常窦性心律的心电图

Figure 5-2 ECG of a heart in normal sinus rhythm

行时会打开两个文件作为输入输出。为了方便Dart调用，需要将输入输出方式改为直接通过参数和返回值进行。原作者已经考虑到需要修改输入输出方式的需求，并从算法中提取出了dataTypeinput()和voidoutput(intout)两个函数。原始情况下，算法会在需要读入新数据时调用input，在分析结果就绪后调用output，两者之间并不同步，而是使用了异步回调。跨语言实现异步回调虽然并非完全不可行,但相比简单的调用后直接返回的流程更为繁琐，且没有太大收益。经过分析，发现该算法实现的主体部分是一个无限循环，在循环的开头会调用input获取采样点数据,在循环中间的多处会调用output后进入到下一次循环。为了将其改为单次调用后直接返回的方式，将该循环的内容提取为了另一个函数，并将需要在循环之间保留的状态由函数内的局部变量暂时改为了全局变量。同时，将输入与输出进行同步，以略微降低算法精度为代价获得了更及时的检测结果。

在该实现的原始版本中，采样率是以常量的形式硬编码在算法之中的。但是，实际调用该算法进行分析时，可能会以不同采样率的数据进行输入。一种可能的方式是

将算法复制多份修改常量后分别编译，但比较繁琐。另一种方式是将该常量改为算法的参数，从外部传入。由于算法中有很多常量和变量的值与采样率有关，因此为了方便将其作为参数，将上一个步骤中提取出的全局变量与函数封装成为了类，并将采样率作为该类的构造函数的参数。这一改动也解释了之前为什么选择将C语言的实现迁移至C++而非直接使用C语言进行重构，这样可以利用C++的强大特性来简化代码的编写难度。

在最终版本的算法中，对外提供了两个函数作为接口，分别是voidinit(intfs)和boolpanTompkins(floatsample)o包含相关状态与方法的类被简单地命名为PanTompkins。程序中维护了一■个std::optional<PanTompkins>类型的全局变量，用于存储算法的状态。在调用init时，如果全局变量目前为空或者存储的算法的采样率与传入的参数不同，则会创建一个新的算法实例并存储在全局变量中；如果全局变量目前不为空且存储的算法的采样率与传入的参数相同，则不会创建新的算法实例，而是直接复用旧的实例，以减少不必要的开销。由于用户通常不会频繁更换使用不同采样率的设备，此优化的效果是比较明显的。在调用panTompkins时，会断言当前全局变量不为空，也就是要求调用者必须先调用过init再调用此方法；之后，会将传入的采样点数据转发至算法实例的对应方法，该方法会返回一个布尔值，表示是否检测到了新的QRS波群。

在算法的测试过程中，发现其偶尔会对于同一个心拍输出两次或更多相近甚至相邻的true，导致计算出的心率可能会突然提升至每分钟上千次。由于难以确定是在算法的哪一处出现了问题，所以在算法之外对于其输出结果设计了两层额外修正，而没有修改算法内部的执行逻辑。第一层修正是为算法输出的QRS波群设置最小间隔，该间隔的计算方式如下：

, 八 60 „

leastsamplesBetweenQrs = x fs

maxHeartRate

其中，leastsamplesBetweenQrs表示连续两个QRS波群之间至少需要间隔的采样点数。当算法返回了true时，会

将当前的采样点与上一次检测到QRS波群的采样点进行比较，如果两者之间的采样点数目小于leastsamplesBetweenQrs，则会忽略该检测结果，否则会将当前的采样点作为最后一次检测到QRS波群的采样点。60表示一分钟的秒数，maxHeartRate表示最大心率(单位bpm)，fs表示采样率(单位Hz)。关于人类的心率上限，最流行的说法是220减去年龄。考虑到患者应该不会在心率超出此上限的情况下还有使用本应用观察自己心率的需求，同时为了不需要获取患者的实际年龄就可以运行算法，在实现中将maxHeartRate设置为了常量220。经过测试，该修正可以有效地解决算法输出的QRS波群之间间隔过小导致心率上千的问题。

此外，在最终的心率显示部分也增加了一层修正。该层修正是在前述过滤方式未能完全排除算法的错误输出的情况下的缓冲。考虑到人类的心率的变化应该是相对平滑的，不太可能在相邻几个心拍之间产生非常大的变动，应用在最终将心率计算结果展示给用户的时候会先与之前的结果进行比较，如果相差超过10bpm，贝U会将显示结果改为上一次结果加上或减去10bpm（取决于心率是突然上升还是突然下降），但不改变底层数据的实际值。通过这种方式，可以有效地避免因为算法的错误输出导致心率突然上升或下降的情况，同时在心率真的在短时间内发生了较大的变化时，也能够在数次心拍后将所显示的心率恢复到正确的值。

应用对于心率的计算是根据最后两次检测到的心拍的间隔时间，所以另一个考虑过的修正方案是使用较多心拍之间间隔的平均值来计算心率。这种方案的效果与原理和上一段说明的方式类似，都是对心率变化进行平滑处理，滤去突变。相比起来，这种方案需要将过去的更多心拍纳入计算，但相比之前的方法在效果上没有明显的优势，所以在实现中没有采用。

5.4智能检测算法的移植与优化

Pan-Tompkins算法只能给出心拍的位置，但无法确定心拍的类型。同时，其作为在线算法，虽然有能实时输出结果的优点，但准确率相较离线算法也更低一些。由于用户对于分析报告的查看没有实时性非常高的需求，因此应用使用基于人工智能的另一种算法来对心电数据进行分析。

所用算法的原始版本使用Python实现。对于将其迁移至C++的过程，考虑过两种方案。其一是先将Python代码原样翻译成C++，再对C++代码进行重构以满足项目需求；这种方案的优点在于比较容易按C++的风格对代码进行重构，而缺点在于需要翻译的代码量比较大故容易出错，而且其中包含很多最终不需要使用的代码以及难以在C++中找到对应替代的依赖项。因此，本项目采用了与之相对的另一种方案，即先对Python代码进行重构，再将重构后的代码原样翻译成C++；这种方案可以最小化翻译代码步骤的工作量，并可以事先编写单元测试来保证重构过程不会破坏代码的正确性；其主要缺点则在于需要在重构过程中注意避免使用难以在C++中实现的特性，比如对同一变量进行多次不同类型的赋值[5]等。

首先执行的步骤是对代码的格式进行统一。在原始版本的代码中，命名、空格等格式均混合使用了多种风格，不便阅读。因此，首先对代码使用Black工具进行了格式化，使得代码的空格、换行等风格更加统一。然后利用PyCharm的重构功能对代码中变量、函数等的命名进行了修改，使其遵守Python官方的PEP-8规范中对命名的相关要求。此外，在该步骤中也移除了PyCharm能够直接检测出的一些未被使用的导入项；自动优化导入后发现代码无法继续正常运行，排查发现代码中使用类似反射的方式使用了部分导入的内容，因此未被PyCharm检测到；为该类实际上被使用的导入项添加了注释，以抑制PyCharm的误报。

之后进行的是对算法的输入输出方式的优化。在原始版本中，算法的每个步骤都是从磁盘文件读取数据作为输入，之后把其结果写入另一个磁盘文件。这种方式虽然编写简单，但会导致数据在各个步骤之间的流向非常不清晰，并且在性能上有所损耗。最关键的是，这种输入输出方式使得难以为各个步骤编写单元测试。因此，优先调整了代码中数据的传递方式，使之仅在主函数中从磁盘文件读入，之后各个步骤均通过参数与返回值直接传递，最后再在主函数中将最终结果写入磁盘文件。这样以来，就可以通过替代主函数来为算法编写单元测试，方便后续重构。该步骤完成之后，对比了最终的磁盘文件与原始版本的输出，发现了一些不一致，经研究讨论后发现是出于原始版本的一些bug，修复后可以得到一致的输出。据此可以判定该步骤并未破坏算法的正确性，反而修正了原本存在的缺陷，而此后的步骤都可以通过持续集成中自动运行的单元测试而非手动对比来保证算法在相同输入下的输出并未改变。

原始项目中包含非常多的代码文件，但是经过覆盖率检测，发现在算法执行的过程中大部分代码都没有被执行到。经过仔细测试、分析，发现有许多代码是多余的。于是删去了无意义的代码，这包括原项目中的数个目录、一些文件、以及某些文件中的部分代码行。之后，发现有部分模型等资源文件也未被读取过，将这些文件也进行了清理。完成对于未使用文件的清理后，对项目中各个文件的位置进行了调整，将代码文件与资源文件等分别移至单独的目录中，方便后续继续重构。

该算法原本使用了一段长达24小时的动态心电数据进行测试，这可以较为充分地测试算法对于各种类型的心电数据的表现，但也导致测试时间很长，单次执行需要数分钟，在性能较差的持续集成环境下甚至需要更久。由于代码每次进行更改时

都要重新运行测试以保证正确性，为了减少在测试步骤的等待时间，提高开发效率，将数据截取了前10分钟来进行更快速的测试。

下一个步骤是进行更细粒度的重构。由于原项目代码的组织结构较为混乱，为了方便后续处理，首先将所有代码文件合并至单个文件之中，去除了多个代码文件之间的导入。在合并的过程中以及合并完成后，发现各个文件中存在部分逻辑相似甚至完全相同的代码，对这些重复的代码进行了合并，提取了共用的方法。并且，对一些未使用的参数进行了删除；对在所有调用中均传递了相同值的参数（比如重采样的目标采样率被固定为250Hz）进行了简化，仅作为全局常量，而不在层层调用中进行传递。在该步骤中，也对方法中一些不符合正常编码习惯的写法进行了替换，比如将flag==False改为notflag（C++中为！flag）、手动打印错误消息改为抛出异常等。此外，对一些与最终结果无关的调试代码，比如基于tqdm打印算法执行进度的代码，进行了删除，并消去了相关依赖项。最后，将简化完毕的代码重新按执行步骤分割为多个模块，分别存储于单独的代码文件供主函数调用，并将仅在某一步骤使用的函数设为模块级别的私有函数，以提高代码的内聚性。

将代码迁移至C++前的最后一个重构步骤是为参数、变量、返回值等添加类型标注。虽然Python是动态类型语言，但通过类型标注，仍然可以获得静态类型检查的优势。在该步骤中，使用Mypy工具进行了严格的检查，固定了每个变量、参数的类型，对同变量多次赋值为不同类型的情况进行了变量拆分。在这一步骤中也借助类型检查发现了原始代码中的一处笔误，该笔误所处代码所代表的情况刚好在测试数据的执行过程中未出现过，所以之前未被发现。之后，对错误进行了修正。

得益于上述的重构，以及LibTorch、NumCpp等库的接口设计的友好性，迁移过程的工作基本都在于翻译语言功能，很少涉及代码逻辑的修改。然而，在将代码原样翻译成C++后，发现在原本在Python中可以正常使用torch.load加载的模型文件无法在C++中使用torch::load读取。检索并查阅相关资料后，发现需要先将模型转换为TorchScript格式以消除Python依赖。在转换过程中，为了适应TorchScript的限制，对模型代码进行了微小重构，如将for循环进行展开等。

在迁移过程中编写的LibTorch相关代码是在Linux环境下进行调试的，这是考虑到在移动端调试C++代码较为不便，并且C++代码具有跨平台性，在Linux下能够正常运行应该就意味着可以在其他平台也正常运行。不过，在将于Linux平台下调试完成的代码移植至移动平台时还是遇到了麻烦，其原因在于C++的跨平台性仅在源代码级别成立，而编译后的二进制文件并不能跨平台使用。而在该部分算法的各种41

依赖项中，LibTorch的分发方式与其他依赖不同，虽然官方也提供了开源代码，但主要的安装方式是下载并解压预先编译好的静态链接库以及相关的头文件等。但是在官网的下载界面[6]中，只提供了三个桌面端系统的LibTorch下载，而没有提供移动端的LibTorch。

为了获得能在移动端使用的LibTorch，考虑过两个方案，一是从源代码自行编译，二是寻找已经编译好的版本。由于LibTorch项目规模庞大，自行编译的方式繁琐且耗时，所以优先考虑了使用已经被编译好的版本。在经过了一些寻找与尝试后，发现PyTorch官方提供的PyTorchMobile中存在LibTorch的相关文件；事后想来这是很自然的，因为PyTorch的各种版本都是为其底层的C++实现提供其他语言的接口，那么LibTorch作为C++版本的接口也很可能会在其中使用到。在移动端的构建过程中，首先将PyTorchMobile作为依赖项正常下载，然后并不直接使用，而是将其解压并将其中的LibTorch头文件和链接库复制至构建目录中，在C++的编译过程中对相关的头文件目录进行包含并将其链接库与算法代码进行链接。

5.5心电模块的实现

5.5.1实时心电界面的实现

实时心电界面的整体外观如图5-3所示。

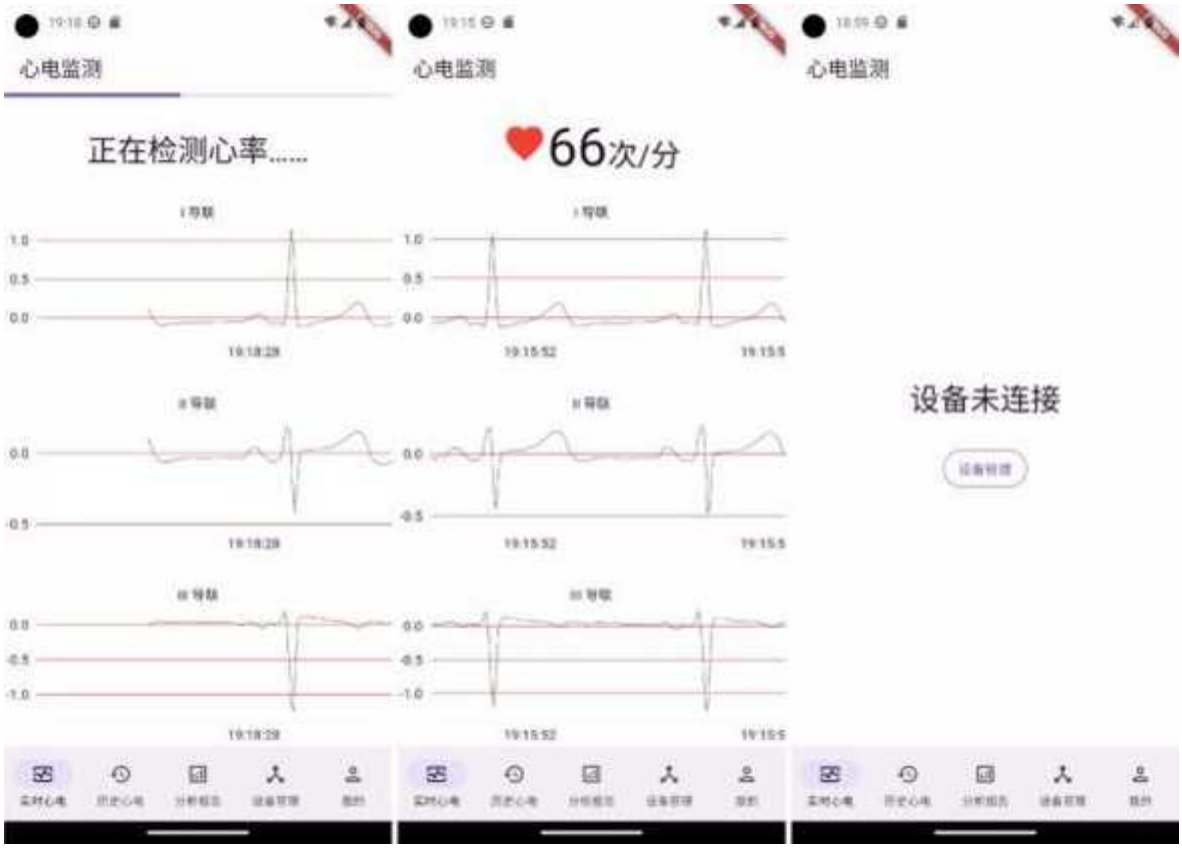
实时心电界面的主体可以分为心率和心电图两大部分。另外，在整个应用的最上方显示了应用的名称，最下方显示了应用的导航栏，导航栏提供了应用中最主要的几个界面的入口。

5.5.1.1心率部分的界面实现

实时心电界面的上方显示了用户当前的心率。

当用户刚刚佩戴上或连接上心电监测设备的时候，由于缺少数据，心率无法立即获得。在这种状态下，心率部分会显示“正在检测心率……”作为占位符，并且上方会显示一个线状的进度条。Material规范中规定了两种类型

的线状进度条，分别表示确定的和不确定的进度。因为心率检测所需的时间可以大致确定，所以此处使用了确定进度的版本，进度条的已填充长度占比表示心率检测的预测进度。进度条的使用可以向用户提供检测进度的视觉反馈，虽然并不能在实际上加快检测速度，但由于减少了不确定性，仍然可以减少用户在等待过程中的焦虑感与沮丧感，起到了与加快检测速度类似的作用。反之，如果在系统正在工作的过程中不为用户提供指示，



(a)心率检测中(b)正常状态(c)设备未连接
图5-3实时心电界面的截图
Figure 5-3 Screenshots of the real-time ECG page

比如某些网站在上传作业文件时的实现，即使是在只需要等待数秒的情况下，用户也会感到焦虑与沮丧，在更长时间后甚至会由于缺乏对任务最终可以完成的信心而感到挫折并放弃使用。进度条虽然只是个简单的组件，但对于软件的易用性和用户体验来说却是个非常重要的细节。

心率检测的结果可用之后，该区域则会显示心率数据。显示内容由横向排布的三个组件组成：图标、数值和单位。图标使用了一个红色的心形标志，主要起到两个作用：一是指示该区域显示的内容是心率，由于应用的性质与该界面其他元素的暗示，用户很容易理解到所示数值表示心率，所以不需要使用文本等方式进行明确说明，只使用简单的图标已经足够；二是加强该区域的视觉强调效果，由于心率区域在整个屏幕上的面积占比较少，但重要性并不显著弱于下方的心电图等元素，所以使用了这个颜色较为鲜艳的图标来将用户的注意力吸引至该区域。数值和单位使用了不同大小的字体，数值较大而单位较小。这是因为心率数值是该区域的主要内容，且不断变动，需要较为强调。而心率的单位是固定的，不会变动，而且每分钟的次数(bpm)作为最常用的心率单位已经为用户所习惯，不需要特别强调，所以使用较小的字体弱化了单位的视觉效果，以突出其他部分。三个组件在横向与纵向上整体居中，内部按文本基线对齐（即靠下对齐）。

此外，整个心率显示区域在实时心电界面上的高度占比是固定的，以免在检测完成前后以及数值变动时引起下方心电图位置的上下移动。

5.5.1.2心电图部分的界面实现

心电图部分的界面外观参考了医学上常用的心电图纸的样式。心电图纸的整体外观在绪论部分的图1-2中已经展示过，这里对其具体布局方式进行一些补充介绍。

心电图纸包含横纵交错的坐标线，通常为红色或浅红色，如图5-4所示。每1mm一小格（用细线分隔）每5mm一大

格（用粗线分隔）横轴一小格表示40ms，一大格表示200ms。纵轴一小格表示0.1mV，一大格表示0.5mV。心电图的边缘有1mV（10mm高）的参考波形，并标注该条图像的导联名称。

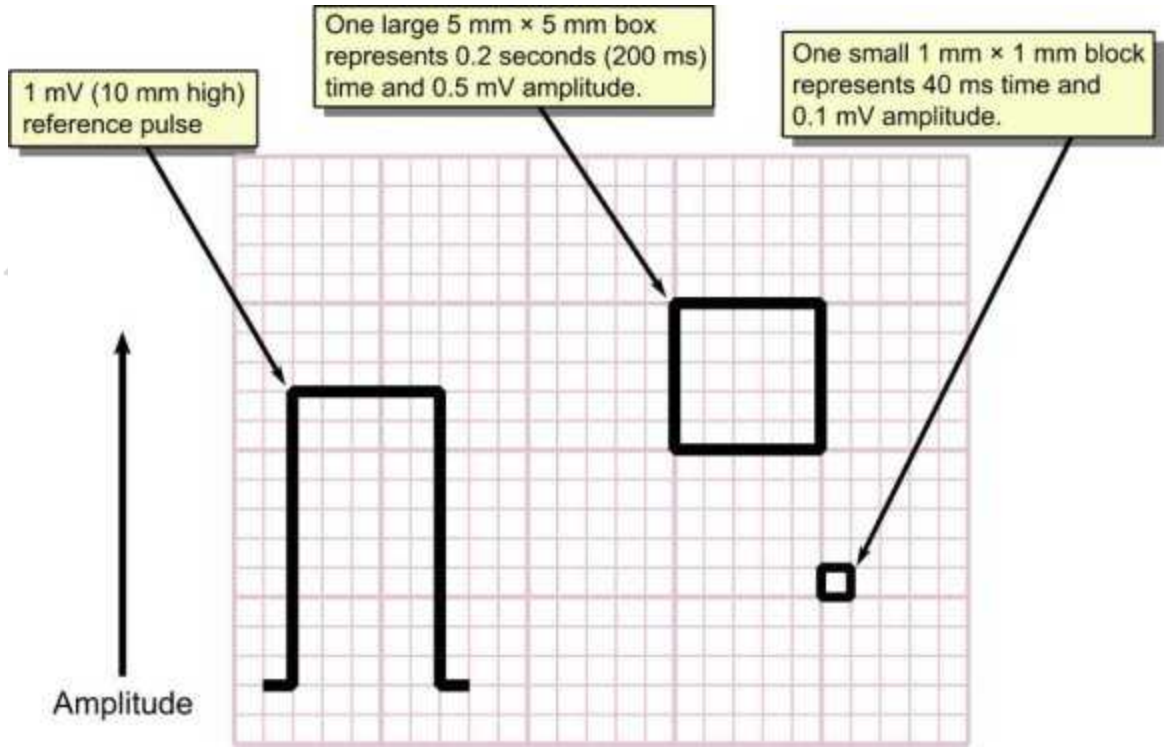


图5-4心电图纸的布局

Figure 5-4 Layout of ECG paper

在应用内实时心电图的界面实现过程中，对心电图纸的元素进行了一些精简与修改。首先，由于移动设备的屏幕尺寸较小，将心电数据的显示范围进行了缩减，纵轴范围根据当前图中显示的心电数据自适应变动，以最大值加0.1mV作为上限，最小值减0.1mV作为下限。横轴范围则按照心电数据的常见电压范围进行了等比例调整，使其与纵轴比例可以保持近似对应，并在应用设置内增加了相关的选项供用户按需自行调整。之后，由于纵向坐标线在实时心电图的快速移动的过程中会产生视觉上的严重干扰，所以将其去除，改为在下方显示数据对应的时间；为了保持视觉上的协调，横轴坐标线去除了划分小格的细线，仅保留了0.5mV一条的粗线。最后，考虑到一般用户对心电图纸的参考波形的了解程度较低，加之移动设备的显示空间有限,所以将参考波形去除，改为在左侧显示电压数值，并相应地将导联名称移至心电图的正上方中央。

心电图部分在竖屏和横屏状态下有不同的显示方式。竖屏状态下，如之前所示,三个导联的心电图从上到下竖向排布在该区域。由于移动设备的屏幕宽度较窄，所以竖屏状态下可以显示的时长也较短，屏幕中每个导联通常只能显示一到两个心拍。虽然可以在应用设置中对显示时长进行调整，但在显示区域的宽度无法改变的情况下，显示更长时间的数据只能是以降低细节的可见程度为代价的。为了提供一种可以在不提高显示密度的情况下查看更长时间的数据的方式，应用在横屏状态下会改为仅显示单个导联的数据，如图5-5所示。

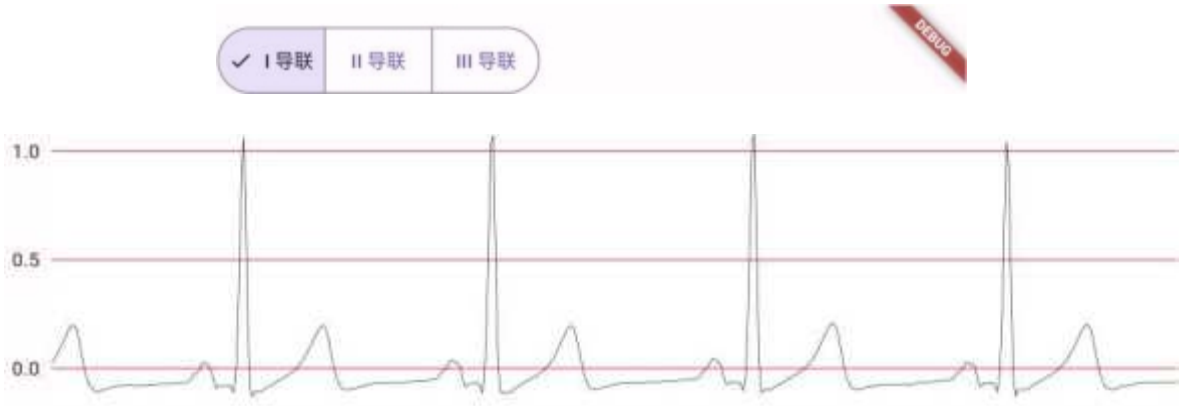


图5-5横屏状态下的心电图截图

Figure 5-5 Screenshot of the ECG in landscape mode

在横屏状态下，应用会进入全屏状态，隐去系统自带的状态栏等元素。在应用界面内部，顶端的标题栏以及屏幕上部的心率显示区域也会隐藏，以提供更多纵向空间。心电图区域会由同时显示三个导联变为仅显示单个导联，并将心电图上方的导联名称替换为分段按钮，以便用户可以在不同的导联之间进行切换。在横屏状态下，应用在心电图之外仍然为上方的导联切换按钮与下方的导航栏保留了足够的空间。这一方面是因为Material规范要求可点击的元素必须保留足够的大小，以便用户可以轻松点击；另一方面是因为如果要让心电图在保持横纵比例不变的情况下显示更长的时间，纵向空间就需要进行相应缩减；同时，这也保证了在横屏状态下仍然可以正常使用应用的基本功能，使用户不需要在进行切换界面等操作之前先将设备恢复为竖屏状态。

5.5.1.3设备未连接状态的界面实现

在设备未连接的情况下，由于无从获取实时心电数据，实时心电界面没有可供查看的实际内容。如果仅显示没有数据的空心电图，可能会使用户产生疑惑。因此，在设备未连接的情况下，应用会将实时心电界面替换为一个特殊的错误提示界面。该界面非常简洁，仅在显示区域的中央包含两个元素，竖向排布并留有一定间隔。上方以较大字体显示“设备未连接”的提示信息，下方以按钮形式提供前往设备管理界面的入口，提醒用户检查设备的状态。由于用户可能会在应用保持处于该界面的情况下调试心电设备，所以即使在心电设备尚未连接的情况下，应用也不会自动重定向至设备管理页，而是在用户手动点击按钮后才会跳转。

Material规范提供了各种外观风格的按钮。在该提示界面中，比较适用的几种风格的按钮如图5-6所示。

设备未连接 设备管理	设备未连接	设备未连接 设备管理	设备未连接 (设备管理)	设备未连接 设备管理
(a)抬升按钮	(b)填充按钮	(c)填充色调按钮	(d)轮廓按钮	(e)文本按钮

图5-6各种风格的按钮对比

Figure 5-6 Comparison of different button styles

图中从左到右分别为抬升按钮(Elevatedbutton).填充按钮(Filledbutton).填充色调按钮(Filledtonalbutton)、轮廓按钮(Outlinedbutton)和文本按钮(Textbutton)。其中，抬升按钮因其阴影效果较易与其他元素之间产生不协调感，被规定为仅在绝对必要的时候，比如需要从图案背景中进行视觉分离时等特殊情况下才应使用，显然此界面并不属于应添加阴影的特殊情况。其余四种按钮的强调程度由强至弱，应当按需使用。由于该界面元素较少，所以不需要对该按钮进行较为强烈的强调，故未使用两种填充按钮。如果使用文本按钮，则视觉效果过于微弱，甚至使用户有些难以察觉到这是一个按钮。最终，选定了轮廓按钮的风格，其强调程度充足但不过度，与其他元素最为协调。

5.5.1.4应用中图标的选择

实时心电界面在导航栏中的图标使用了常见的心电图的标志，这是根据该界面的主要内容决定的。

Material规范对于图标的设计与使用有一些相关要求。本应用为了遵循规范，使用的均是Google官方设计并开源供免费使用的图标。Google为其图标提供了各种风格的样式，常见的几种如图5-7所示。

(a)轮廓(b)圆润 (c)尖锐(d)常规(旧)(e)填充(旧)

图5-7Material图标的各种风格

Figure 5-7 Different styles of Material icons

图中的前三种风格是Materials新增的，通常应该比旧版优先使用。具体使用哪一种风格应该根据应用程序的整体界面风格来确定，比如圆润的图标使用了较多圆角，与使用较重的排版、弯曲徽标或圆形元素来表达其风格的品牌搭配得很好；尖锐的图标则带有较多直角，体现出清晰锐利的风格。本应用内的所有图标都使用了轮廓风格，这和整个应用的轻盈、干净的界面风格保持一致。

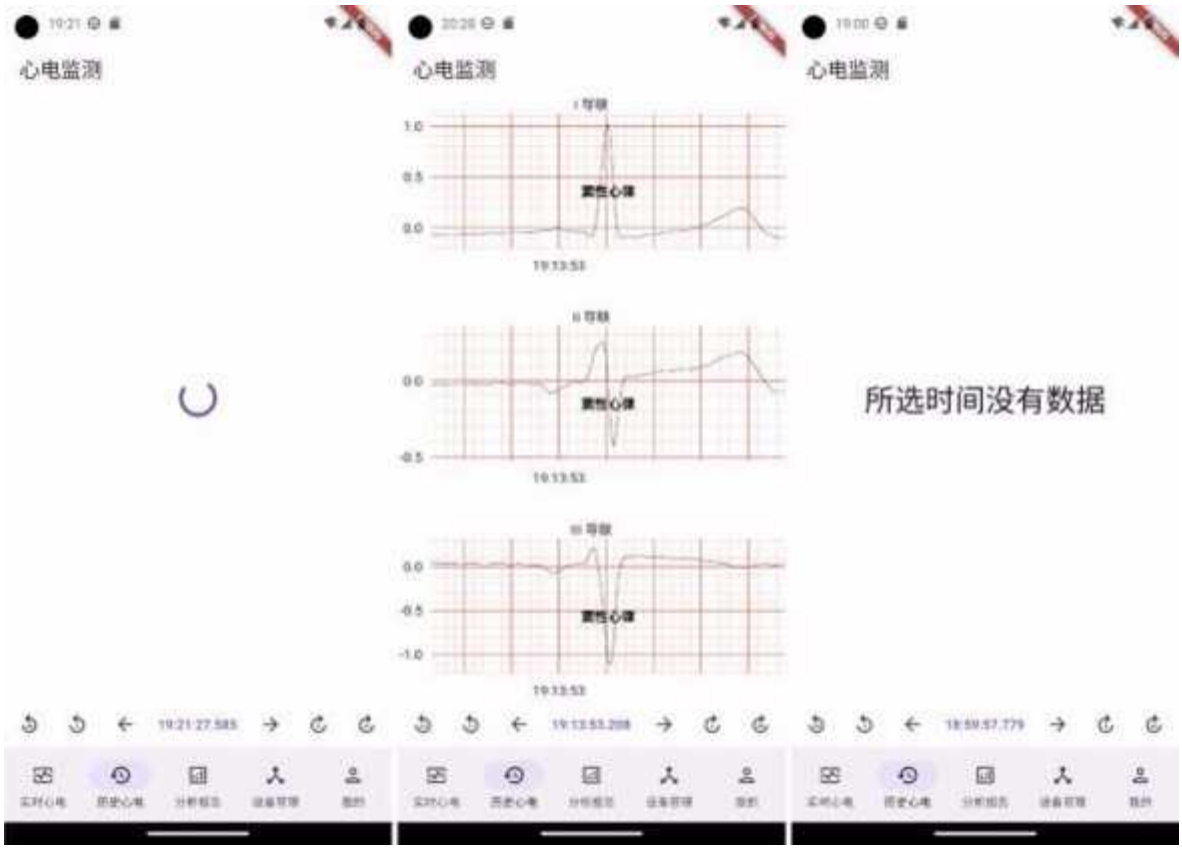
5.5.2历史心电界面的实现

历史心电界面的整体外观如图5-8所示。

该界面由两部分组成，分别是上方的心电图部分和下方的时间选择器部分。为了与实时心电进行区分，导航栏中的图标使用了表示历史记录的符号。

5.5.2.1心电图部分的界面实现

历史心电图与实时心电图的总体界面实现方式相同，重复部分不须赘述。相比实时心电图，历史心电图因其性质而在界面外观上具有一些不同点。



(a)加载中(b)正常状态(c)无数据

图5-8历史心电界面的截图

Figure 5-8 Screenshots of the history ECG page

首先，关于心电图背景的网格，在实时心电图中完全隐去了纵线、部分显示了横线，如上文所述是因为心电图在不停横向移动。但是在历史心电图中，图中显示的数据是静态的，在用户不调整所选时间的情况下不会发生变动。因此，没有必要在历史心电图中隐去或部分隐去背景的坐标网格。在历史心电图的界面外观中，背景的网格线与实际的心电图纸保持一致，以粗线分割大格，每大格里再以细线(实际显示为颜色较浅的线)分割为5小格。这样的界面可以使得心电图上各点的数值更加清晰，方便用户对历史心电数据进行仔细查看。

另一个显著的不同点在于历史心电图中对每个心拍的类型进行了标注。由于所使用的智能检测算法是离线算法，无法实时给出心拍类型，因此该标注在实时心电图中无法实现，仅可在历史心电图中进行查看。心拍的类型以文本形式直接显示在每个导联的心电图中对应位置的中央。一个被考虑过的替代方案是在R波(心电图中幅度最大的波)的波峰所在的点进行标注，但这需要心电图中在上方保留一些额外的空间，不太适合移动设备的较小的屏幕，而且各个导联中的波峰也并非完全同步。另一个可能的替代方案是使用颜色而非文本对心拍类型进行标注，但这种方案要么需要在屏幕上的某处显示各种颜色对应的标签而不适合小屏幕设备，要么需要用户在某个说明界面中查看并记忆各种标签的颜色而严重影响了应用的易用性，因此也没有采用。

此外，该部分区域存在两种特殊状态，分别是加载中和无数据的状态。经过对数据库的优化之后，加载中状态持续的时间很短，仅在设备性能较差、数据库索引未完成、缓存也未命中等情况同时发生的状态下才会有较长时间的加载；因此，加载中界面仅简单使用了一个圆形的不确定进度的加载动画来指示应用并未失去响应。无数据的状态则是在用户选择的时间前后内没有心电数据的情况下出现的，可能是由于用户在该时间并未佩戴监测设备，或是

应用后台进程因各种原因被终止而缺失该时间的数据，也可能是用户刚刚开始使用该应用但试图查询几小时前的数据。在没有可用数据的情况下，该区域并不会显示只有背景的空心电图，而是会直接显示所选时间没有数据的提示。

5.5.2.2时间选择器部分的界面实现

时间选择器部分由7个横向排开的按钮组成，按钮之间等间距，左右两侧不留额外空间。由于按钮在外观和功能上都采用了左右对称的设计，因此只对左起的4个按钮进行说明。

前两个按钮分别使用了在倒退符号中包含数字30和数字5的图标，按钮功能分别为将所选时间调整为30秒前和5秒前。

第三个按钮使用了向左的箭头作为图标，其作用在不同情况下有所差别。当用户点击该按钮后，如果当前时间之前可以找到另一个心拍，则会跳转至该心拍所在的时间；如果当前时间之前已经没有心拍，比如当前时间所示的是记录内的第一个心拍，或者用户手动跳转到了有数据记录之前的区域，或者算法在数据边界出现了偶然的故障而没有正确识别出心拍，则会跳转到1秒之前，保证该按钮无论如何都不会毫无作用。

第四个按钮，即中间的按钮，其作用相较前几个按钮更多。首先，该按钮作为文本按钮，起到了普通文本的显示作用。其指示的当前查询时间是心电图正中间的时间，并且在切换横竖屏状态时仍然保持该时间在正中间。由于时间选择器的上一个与下一个心拍按钮以及分析报告中跳转心拍所在时间的功能，位于图像中间的通常刚好是某个心拍的R波，这保证了视觉效果最明显的R波恰好会位于用户视觉焦点上。其次，该按钮也提供了相比其他几个按钮更大粒度的时间跳转。点击按钮后，会弹出如图5-9所示的Material风格的时间选择器，用户可以在其中以表盘模式或数字模式输入要跳转到的时间，点击确定后，时间选择器会自动关闭并将历史心电图跳转至所选时间。由于该自由跳转功能的使用频率远低于其他几个按钮，所以只使用了强调效果最弱的文本按钮，以避免不必要地干扰用户的注意力。

(a)表盘模式输入(b)数字模式输入

图5-9历史心电时间选择器的截图

Figure 5-9 Screenshots of the history ECG time picker

从图中也可以看出，当可用显示区域由于键盘的弹出而缩小时，应用的界面布局仍然可以保持基本可用。这也体现了应用的界面可以适应不同尺寸、不同屏幕比例的设备，并且对分屏等比例特殊的情况也有支持。

5.5.3心电图的细节处理

5.5.3.1坐标标签文本的间隔

坐标标签文本的间隔在默认设置下为每秒一个标签，但是当用户自行调整心电数据显示时长时，尤其是调整至较长时间时，需要一个算法来自动确定合适的间隔。经过研究，给出了以下公式：

intervals econds = \duration.inSeconds/intervalCount~\

其中,duration是用户自行设定的时间,intervalCount是坐标标签文本的数量上限，在横屏下为5,竖屏下为10。

5.5.3.2坐标线粗细判定

smalllnterval2心电图中每5小格（细线分隔）为1大格（粗线分隔），为了基于坐标线所位于的值来判定其应使用细线还是粗线，首先基于其横纵获取了相应的largelnterval（200或0.5）和smalllnterval（40或0.1），然后考虑如下不等式是否成立：

(\value\ + 10-6) mod largelnterval <

若成立则使用粗线，否则使用细线。该算法可以适用于正数或负数的值，并考虑了浮点数的相关误差。

5.5.4实时心电图的性能优化

实时心电图由于数据量大、更新频繁，对性能要求较高，且对电量等资源的消耗也较高。为了降低实时心电图的渲染等开销，进行了两方面的优化。

5.5.4.1刷新率的降低

在应用设置中提供了实时心电图的刷新率设置，默认为30Hz。在实时心电图的实现中，首先将其取倒数转换为最小刷新时间间隔，然后维护一个上次刷新时间的变量，在每次为底层数据添加了新的点时判断当前时间与上次刷新时间的差值是否大于最小刷新时间间隔，如果是则进行刷新，否则跳过。

5.5.4.2点数的减少

每秒钟上百个点对于快速移动的实时心电图来说过于精细了，可以考虑通过各种算法减少图上的采样点的数量来降低渲染开销。

一个最简单的方法是每隔若干个点取一个点，但是这样会导致图像的连续性变差，尤其是如果刚好跳过了一个波峰或波谷，那么图像的形状会有明显的变更，即使是在快速移动中也足够明显。因此，需要考虑一种算法来保证图像的连续性。

在心电信号的重采样领域，已经存在许多算法，普遍都是使用各种方式进行插值。为了在大量计算时保证效率，本应用中使用了一种更简单的算法，每当一个点到达时，会比较其与前一个点的标准化的距离：

b.x — a.x

ax =

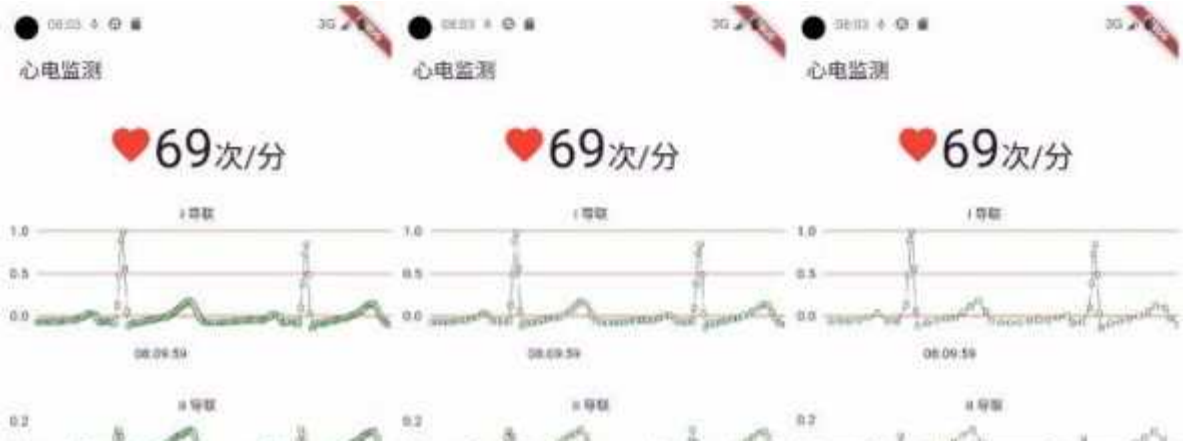
smallX Interval

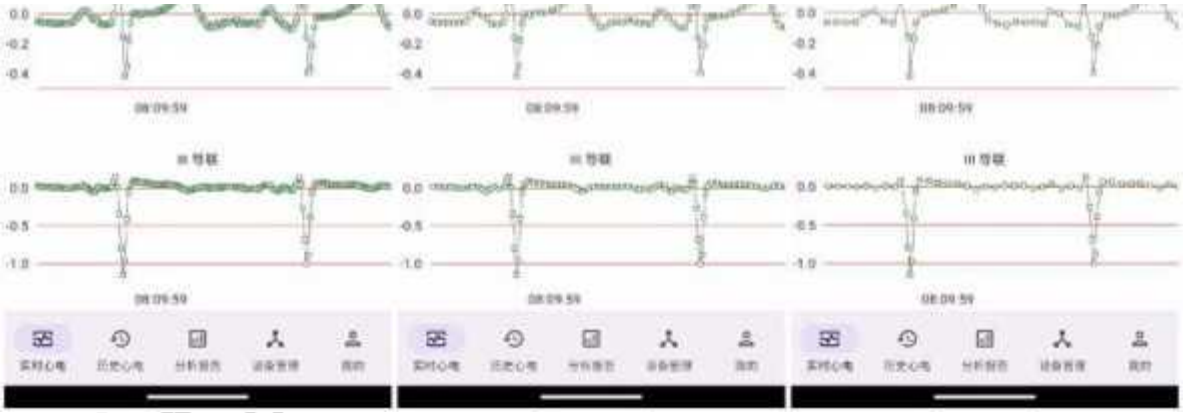
ay = b, —时

smallY Interval

dis = ^ dx2 + dy2

其中，smallXInterval为横坐标的小格间距（40ms）,smallYInterval为纵坐标的小格间距（0.1mV）,a为前一个点，b为当前点。如果dis大于可在设置中调整的一个阈值，则保留当前点，否则丢弃当前点。图5-10为不同阈值的效果对比。本应用在默认设置下会使用0.5作为阈值，该阈值可以明显降低屏幕上需要渲染的点数，但不会使得图像过于失真。从图中也可以看出，本算法主要过滤掉的是较为稠密的区域的点，而对于波峰等较为稀疏的点则保留原样。





(a)0(b) 0.5(c)1

图5-10不同阈值的效果对比

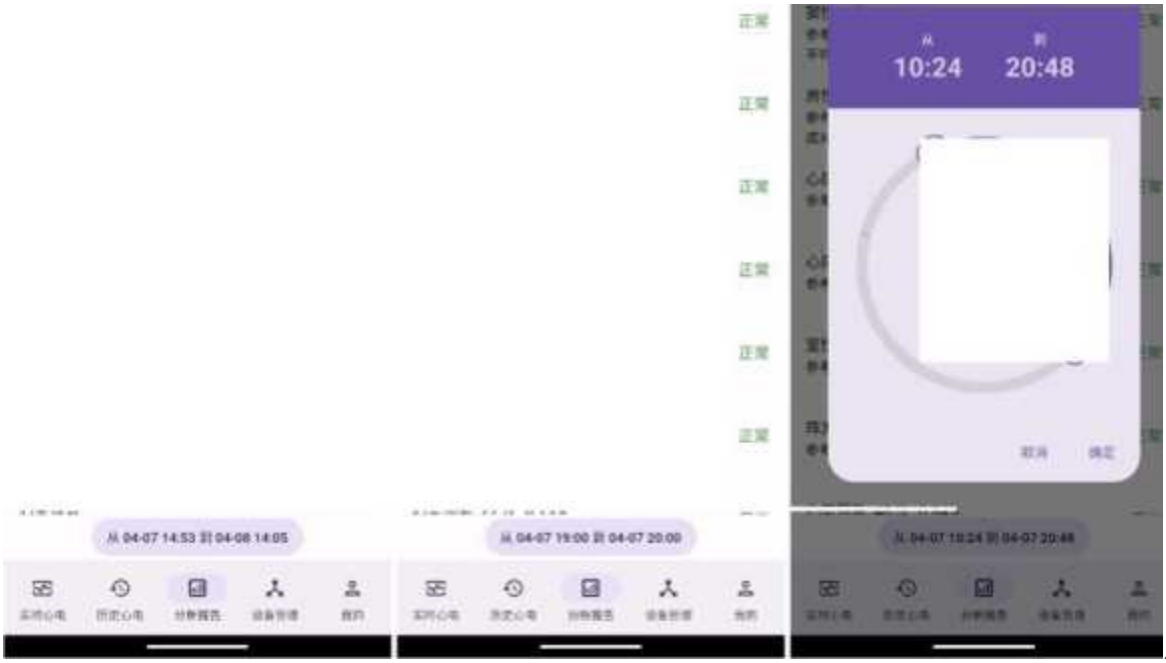
Figure 5-10 Comparison of different thresholds

5.6分析报告模块的实现

5.6.1分析报告界面的实现

分析报告界面的整体外观如图5-11所示。在导航栏中为分析报告使用了表示数据分析的图标。

.15:55金fi心电监测 .15:570fi心电监测 ■心电监测



总体分析结果报告仅供参考，有胸痛、胸闷等不适感请及时就医。总体分析结果 小报告仅供参考，有胸痛、胸闷等不适感请及时就医。止吊总体分析结果 -报告仅供参考，有胸痛、胸闷等不适感请及时就医。止吊,|、\室褊渤77冲072%.室褊’救40次nn、、宝新:知室性早搏参考值：无心房颤动参考值：无心房扑动参考值：无心房扑动无参考值：无心房颤动无参考值：无房性早搏参考值：占比＜10%窦性心律3571次96.33%参考值：占比＞90%,平均心室率60-100,无窦…平均心室率80,最快130,最慢54；无窦性停搏房性早搏107次2.89%参考值：占比＜10%成对房早13次，房早二联律15次，房早三联律2阵发性室上性心动过速无参考值：无窦性心律参考值：占比＞9。%,平均心室率60-100,无窦性停搏阵发性室

上性心动过速参考值：无室性早搏无参考值：无(a)加载中(b)正常状态 (c)时间范围选择
图5-11分析报告界面的截图
Figure 5-11 Screenshots of the analytics page
该界面可以划分为分析报告展示与时间范围选择器两部分，另外点击心律类型可以展开心律类型详情界面。
5.6.1.1分析报告展示部分的界面实现

该部分区域整体上是一个可以上下滑动的列表。由于分析报告内容过多，在一个屏幕内完全展示会过于拥挤，所以使用了可滑动列表的形式。

最上方的一小条区域是线状进度指示器。由于难以确定生成分析报告所需的具体时长，所以该进度指示器是不确定进度的样式。另外，由于该部分有很多界面元素可以在获取到实际数据之前就确认其内容与位置，所以没有像历史心电界面那样使用屏幕中央的圆形加载进度指示器，而只是在最上方使用了线状进度指示器，并在加载时提前显示部分可以确定的元素，以减少加载前后的界面变化，提升用户对界面内容的确定感。在加载完成后，该区域并不会直接消失，而是会替换为不可见的与进度指示器高度相同的空白元素，以保证界面布局的稳定性，避免下方内容在加载完成后突然上移。

在进度指示器之下显示了所选时间范围的分析结果的总结，以右侧以带颜色的字体显示对分析结果的简要概括“正常”“心动过速”等），并在下方给出稍详细一些的解释。

总结区域的下方放置了一条分割线，用以分割总体分析结果与具体的心律类型分析结果，并在视觉上对总体结果加以适当强调。

在分割线之下，每个心律类型的分析结果都会显示在一个独立的区域中。首先以稍大的字体显示心律类型的名称、次数、占比，然后以小一些的字体给出该心律类型相关数据的正常范围作为参考值，第三行显示部分心律类型会具有的额外信息，如成对房早次数等；后两行的内容各自被限制在一行之内，溢出的内容会被以省略号替代，以提示用户可以点击该区域查看更多信息。此外，每个心律类型的右侧也和总结一样给出了对该心律类型分析结果的简要概括。

各个心律类型的区域都是可以点击展开查看详细信息的。通常而言，可点击区域应当使用按钮等样式加以指示，但当屏幕上的可点击区域过多时应该避免过多的装饰，以免造成视觉上的混乱。对于列表项的可点击性的指示，一种常见的做法是在右侧显示一个箭头，但在本界面中右侧已经被占据，不便添加更多元素。因此，本界面使用了其他方式来提醒用户心律类型可以点击。除上文所述的省略号外，该界而利用下方被截断的列表项指示该区域可以滑动查看更多内容，并在用户滑过心律区域时展示了水墨扩散的效果，如图5-12所示，效果会从点击位置向四周扩散。水墨扩散效果在Material设计中用于各种可点击区域的反馈，可以提醒用户心律展示区域点击后能触发额外动作。

室性早搏无 正常
参考值：无
图5-12心律类型区域的水墨扩散效果
Figure 5-12 Ink effect of the heart rhythm area
5.6.1.2时间范围选择器部分的界面实现

时间范围选择器部分仅在中间包含一个填充色调按钮。因为需要明确提醒用户所示分析报告的时间范围，所以没有使用强调效果较弱的轮廓按钮和文本按钮。按钮中的文本指示了当前选择的时间范围。点击按钮后，会弹出时间范围选择对话框。

时间范围选择对话框的主体是一个圆环，用户可以拖动圆环上的两个点来选定分析范围的起止时间。为了不引起日期的混淆，当前时间(在圆环上显示为红色)不被允许包含在选择部分之中，这样可以保证用户所选的时间范围总是可以解释为过去24小时之内的某个时间段。
5.6.2心律类型详情界面的实现

心律类型详情界面的整体外观的如图5-13所示。除上方的心律名称和返回按钮外，界面由可滑动查看的列表构成。列表内容包括该心律类型的说明文本，以及心律出现的具体时间。为了保证每个时间都可以较容易点击到，时间之间留有适当的间隔。点击时间后，会跳转至历史心电的对应时间。

s 房性早搏

房性早搏共124次，占比2.61%,在正常范围(<10%)内。

成对房早13次，房早二联律15次，房早三联律2次，短阵房速8阵，短阵房速最快心室率112次/分。

房性早搏可在许多健康人中出现，极少引起症状。饮用咖啡、茶或酒精或者使用有些治疗感冒、花粉热或哮喘的药物都可能会引起房性早搏。

以下是房性早搏出现的具体时间，可以点击查看对应时间的心电图。

2023-04-07 19:00:59.858
2023-04-07 19:01:56.141
2023-04-07 19:02:15.350
2023-04-07 19:02:18.116
2023-04-07 19:02:23.616
2023-04-07 19:02:29.083
2023-04-07 19:03:06.020
2023-04-07 19:03:18.195
2023-04-07 19:03:54.758

2023-04-07 19:05:38.554

图5-13心律类型详情界面的截图

Figure 5-13 Screenshot of the heart rhythm details page

5.6.3分析报告模块的性能优化

在从数据库查询历史心拍数据时，将多个类型的心拍数据进行了并发查询，以提升性能。不过后来发现提升效果并不明显，因为九成以上的心拍集中于窦性心律类型。

5.6.5.6.4将 TimeRange 转换为 DateTime 的算法

当用户提交所选的查询范围时，可以获取的数据类型是TimeRange，其内部是起止时间的TimeOfDay，也就是只包含小时和分钟数据，不包含是哪一天。为了将其转换为完整的DateTime以供查询，首先将结束时间转换为过去的最近一次小时和分钟与TimeOfDay相同的DateTime，然后以同样方式将开始时间转换为结束时间前的最近一个DateTime，这样就完成了时间范围的转换。

5.6.5LibTorch模块相关的处理技巧

由于LibTorch只提供了从文件加载模型的接口，而没有允许从内存直接加载模型，因此传递给C++的参数被设计为文件路径字符串。然而，Flutter中的assets没有直接提供路径，于是在应用启动时的初始化步骤中，会使用rootBundle.load将模型文件先从assets加载到内存，然后再将其保存到临时文件中，最后将临时文件的路径传递给C++。此外，将传递字符串参数的时候，实际的实现为动态分配的字符数组，需要注意在使用完毕后手动释放内存，在将心电数据作为数组传递时也是一样。5.7设备管理模块的实现

5.7.1设备管理界面的实现

设备管理界面的整体外观如图5-14所示。设备管理在导航栏中的图标是对心电监测设备所使用的电极片的简化表示。

该界面相比上述其他界面较为简单，分为已绑定设备和未绑定设备两种状态。

在已绑定设备的状态下，该界面会以列表形式展示设备的名称、型号、信号强度、剩余电量与充电状态等信息，并提供解绑设备的按钮。该按钮没有使用额外的视觉效果来表示其可以点击，因为“解绑设备”这一文本本身已经足够明确地指示了该区域是可以点击的。

在未绑定设备的状态下，该界面会起到搜索并绑定设备的作用。界面上方展示“正在搜索蓝牙设备”的提示并显示了不确定进度的圆形进度指示器，之后有一条分割线，下方则展示了搜索到的设备列表。列表中的每一项都表示了一个搜索到的蓝牙设备（由于Android和iOS模拟器均不支持蓝牙功能，截图中仅显示了一个模拟设

(b)连接新设备(c)无可用设备(a)已连接FakeDeviceFakeDevice蓝牙已连接信号强度：-42dBm电量100%解绑设备正在搜索蓝牙设备FakeDeviceFakeDevice

图5-14设备管理界面的截图

Figure 5-14 Screenshots of the device page

备)，并以带阴影的卡片的样式来显示其可以直接点击，而没有额外放置绑定按钮，以使应用界面更加简洁。

5.7.2设备的相关状态

设备类上定义了其状态的获取方法。其中，ID、名称、型号、采样率因为不会变动而直接定义为Stringget或intget。连接状态、信号强度、剩余电量和最关键的心电数据则因为需要被持续监听而定义为Stream。之后，基于Riverpod包为这些Stream提供了分别的StreamProvider以方便监听。为了编码方便，这些StreamProvider在设备未连接时也会提供一些默认值，多数是Stream.empty()，但连接状态是Stream.value(false)。

5.8其他功能的实现

应用中包含一些不直接面向用户的辅助功能，或者虽然用户不常主动使用但仍然应该提供的功能，一些相关界面的外观如图5-15所示。

.23:08金fi心电监测、测试用F测男21岁ID:testIZI反馈0帮助◎设置0关于玄Q回实时心电 历史心电 分析报告侦刀.17:27金<设置实时心电S 显示风格□ 竖屏数据范围口 横屏数据范围◎ 背景颜色折线颜色H 网格颜色水平网格线垂直网格线(2) 刷新率30HzV 最小点间距0.5历史心电人 2设备管理 我的 企自动上传1500ms 4000 ms. 23:10 © i< 许可心电监测0.35.2+57 Debug BuildPowered by FlutterStackWalker2份许可_fe_analyzer_shared1份许可abseil-cpp3份许可accessibility15份许可aFileChooser1份许可all_lint_rules_community1份许可analyzer1份许可analyzer_plugin1份许可angle(a)我的(b)设置(c)许可

图5-15部分其他界面的截图

Figure 5-15 Screenshots of the some other pages

5.8.1我的界面的实现

我的界面提供了其他所有界面的入口，在导航栏中的图标是用户头像的抽象表示。该界面以列表形式展示其内容。用户项目显示了当前用户的头像、姓名、性别、年龄、ID等信息，由于可以复用Web端(同课题下的另一个子项目)的用户管理相关的内容，所以并未对用户界面进行专门设计。反馈、帮助按钮会在点击后打开对应的网页。设置按钮会在点击后打开设置界面。关于按钮会在点击后显示包含应用自身的版本、构建模式等信息的对话框，并提供更新日志与许可信息的入口。

5.8.2应用设置模块的实现

应用设置界面为常见的列表形式，划分为多个部分，每部分有一个小标题。设置项目均使用了图标、标题、输入组件的形式来展示，有部分如滑动条等的设置项目因不方便在输入组件直接展示当前状态而添加了副标题。而颜色选择器等所需面积较大的输入组件则有缩小形式与完整形式，缩小形式仅指示当前状态，点击后会在弹出的对话框中显示完整形式。

5.8.3许可界面的实现

通常而言，用户并不会有查看应用程序所使用的各个开源包的许可协议的需求。但是，由于多数许可协议要求在应用程序中提供许可协议的副本，因此应用程序中必须提供该界面的入口。该界面以列表形式展示其内容。许可界面的每一项都是一个开源包，点击后会在弹出的界面中显示该开源包的名称、作者、许可协议等信息。此外，该界面的上方也显示了本应用的名称、版本和构建模式的信息，并显示了“PoweredbyFlutter”的标识。

5.8.3.1应用自身信息的获取

许可界面也显示了本应用自身的版本号和构建模式信息。版本号可以使用PackageInfoPlus包直接获取，而构建模式则是通过kReleaseMode、kProfileMode等常量来判断。

5.8.4界面过渡动画的实现

界面过渡动画是连接单个元素或应用程序全屏视图的短动画，是出色的用户体验的基础，可以帮助用户感知到应用程序的界面设计逻辑。由于在无法插入动态图片或视频的情况下较难充分展示界面过渡动画的实际效果，因此本文只作简短说明。

应用在心律类型详情界面的开启与关闭过程中使用了容器转换。当用户点击心律类型区域后，该区域会逐渐扩展至全屏，并将内容渐变为心律类型详情内容，关闭时则反之。这使得各个界面之间的关系清晰明确，加强了分析报告界面与心律类型详情界面的关联感。

在历史心电的所选时间改变后，使用了水平滑动效果，以提醒用户心电图是发生了水平移动。由于各个心拍之间的图像可能较为相似，如果没有该动画，用户可能会产生心电图发生了微小变形而非水平移动了一个心拍的错觉。

点击导航栏跳转至对应界面时使用了顶级(Toplevel)转换效果。该效果专门用于在应用程序的顶级目的地之间导航时，比如点击导航栏中的按钮时。该效果会将旧屏幕快速淡出，然后新屏幕快速淡入。由于顶级目的地的内容不一定相关，因此该动画效果有意不使用分组或持久元素来在屏幕之间创建牢固的关系。

5.8.5数据自动上传与自动清理的实现

应用每天会自动上传历史心电数据与分析报告结果至服务器(服务器端属于同课题的另一个子项目)。由于历史心电数据量较大而分析报告数据量较小，应用在设置中为两者的上传提供了分别的开关，以便流量有限的用户可以选择仅上传分析报
告。

因为移动设备存储空间较为有限，所以应用会自动清理较旧的数据。应用删除数据的范围是距离当前时间超过两天的数据，而非一天，这是出于两点考虑：其一是因为应用内各界面最远提供了一天前的数据查询功能，如果在查询过程中进行删除，需要进行额外的加锁等考虑，而且可能会导致查询结果不准确；其二是因为应用在自动上传数据时可能会因为网络等原因失败，将数据额外保留一天可以提供一些重试的机会。

5.8.6应用的无障碍功能实现

图5-16展示了应用的一些无障碍功能。

•21:58Gf1心电监测 心电监测21:58efi  总体分析结果报告仅供参考，有胸痛、胸闷等不适感请及时就医。正常总体分析结果 I报告仅供参考，有胸痛、胸闷等不适感谓及时就医。止吊窦性心律3571次96.33%参考值：占比〉90%,平…平均心室率80,最快130...房性早搏107次2.89%参考值：占比＜10%成对房早13次，房早二…心房扑动无参考值：无心房颤动无会玄估.ZJZ正常正常正常正常窦性心律4586次96.51%参考值：占比＞90%,平均心室

率60-100,无窦…平均心室率80,最快130,最慢54；无窦性停搏房性早搏124次2.61%参考值：占比＜10%成对房早13次，房早二联律15次，房早三联律2...心房扑动无心房颤动无参考值：无室性早搏无阵发性室上性心动过速无从04-

0719:00到04-0720:00丑。叵］人2实时心电历史心电分析报告设备管理我的[redacted]心室？
Bi渤4(1次nR4%从04-0121:58到04-0821:58正常(a)界面放大(b)粗体高对比度(c)深色模式
图5-16应用的无障碍功能
Figure 5-16 Accessibility of the app

当用户开启了界面放大后，应用内一些原本在一行之内显示的文本会无法在一行内完整显示。对于有必要完全显示出来的内容，应用会将其进行换行，在多行之中显示。对于没有必要完全显示的内容，应用会将其折叠，以省略号指示内容未显示完全。通过这些处理，应用在界面放大的状态下虽然美观程度略有下降，但功能都可以保证正常使用。界面放大的一个常用替代方式是将文本加粗加黑，应用在这种情况下也可以正常提供功能。

为了支持深色模式与浅色模式的切换，应用在界面实现中尽量避免了使用固定的颜色，而是使用系统上下文提供的颜色。这样，应用在不同的主题下自然都可以随之自动切换颜色，从而正常显示内容。

5.8.7平台自适应UI的实现

5.8.7.1应用UI遵循的规范

本应用在UI方面与其他同类应用的一大差别在于其遵循专业的规范对UI元素的使用进行了精细的考虑。

应用整体上遵循Material3规范［43］。Material3（也称为MaterialYou）是Google在2021年5月的GoogleI/O大会上宣布的新一代Material规范，和前代规范相比具有从用户的壁纸自动生成动态主题颜色、按钮更大、界面动画更多等新特性，并在其他一些细节上有所优化。目前Material3规范已被广泛应用于Google的各种应用程序，并成为Android应用的推荐UI规范。

尽管Material3按照Google的设计是可以在任何平台（包括iOS）上使用的，但从事实上来说，iOS应用确实较少使用Material风格的UI，而较多按照苹果的推荐遵循Cupertino[7]规范。

5.8.7.2应用在不同平台的UI对比

为了符合iOS平台的用户习惯，本应用基于FlutterPlatformWidgets包在iOS平台上将部分UI组件替换为了iOS应用常用的Cupertino风格的组件。由于Cupertino并未像Material一样设计为全平台可用，所以应用的UI风格仍然以Material为主，在Android平台上完全使用Material3风格的UI，而在iOS平台上则使用Material3风格与Cupertino风格的混合，如图5-17所示。

从图中可以看出应用在Android平台与iOS平台上的UI实现上的一些差异，包括标题的对齐方式（Android居左，iOS居中）、左上角返回按钮的图标（Android使用完整箭头，iOS使用仅有头部的简化箭头）、滑动条与切换按钮的外观（按照各自规范中的相关要求）、文本的字体（iOS的字体更细）等。

.17:27企fi 设置实时心电E 显示风格□ 竖屏数据范围口 横屏数据范围© 背景颜色折线颜色H 网格颜色W 水平

网格线垂直网格线（2） 刷新率30HzN 最小点间距0.5（C简洁曲专业*自定 4000 ms

[redacted]历史心电(a)Android [redacted] < 设置1500ms实时心电

（2） 刷新率30HzN 最小点间距0.5 历史心电(b)iOS

图5-17应用在Android和iOS平台上的UI对比

Figure 5-17 Comparison of the app’s UI on Android and iOS

5.8.7.3 Flutter Platform Widgets 包在实现中的作用

FlutterPlatformWidgets包在平台自适应UI的实现中起到的作用主要在于基于共识消除了不必要的样板代码。比如为了在Android平台和iOS平台使用不同的滑动条外观，原本需要在if条件中通过判断Platform.isAndroid或Platform.isIOS来决定使用Slider还是CupertinoSlidero由于Slider和CupertinoSlider是公认的不同风格设计对同一种UI的实现，相关的代码并没有提供任何额外的信息量，只是无意义的冗余。使用FlutterPlatformWidgets包后，只需要使用PlatformSlider即可，其内部会根据平台自动选择使用Slider还是CupertinoSlidero这不仅避免了大量关于平台判断的if语句，也避免了将相同的参数分别传递至两个平台对应的类的构造函数之中，有助于使代码更加简洁。

5.8.8数据库的性能优化

在测试过程中，发现数据库相关查询的耗时较长。经分析后，发现数据库大部分时间忙于采样点数据的存入；在设备性能较差时，其存入速度甚至低于数据产生的速度，导致永远无法完成。考虑到采样点数据的产生频率过高，且在异常情况下丢失极少量数据的影响不大，为采样点的写入增加了缓存。在向数据库模块写入采样点数据时，会先将数据写入缓存；若缓存已满，才会将缓存内容实际存入数据库并清空缓存。经测试，缓存大小设为100已经能显著缓解数据库的压力。在该缓存大小下，最多可能在应用意外退出时丢失不足1秒的数据，完全可以接受。完成写入速度的优化后，观察到数据库的查询性能也得到了显著提升。

5.8.9开发者工具的实现

在开发过程中，为了方便调试应用，实现了一些开发者工具。这些工具的入口在默认情况下不会向用户展示，也不会加载相关资源，因此不会对正常的用户体验产生影响。可以通过在设置界面中手动启用开发者工具来开启相关功能。本节介绍一些较为重要的开发者工具。

5.8.9.1模拟设备的实现

因为Android和iOS模拟器都不支持连接外部的蓝牙设备，所以为了方便调试，在应用中实现了一个模拟设备。该模拟设备对于电量、信号等数据均会给出固定值，而关键的心电数据则会循环输出一段样例数据。在曾经的旧版本实现中，模拟设备所使用的样例数据文件是作为资源文件打包在应用程序之中的，这样会使终端用户承担一些不必要的成本。当前的新版实现中，在开发者工具中实现了加载模拟设备所用心电数据的功能，在开启模拟设备之后需要选择本地的心电数据文件进行加载。通过这一改动，不仅减小了应用体积，也更方便切换不同数据进行调试。

5.8.9.2日志功能的实现

应用的日志功能基于Logging包来实现，该包默认情况下只会将日志打印至控制台。为了方便在不连接控制台的情况下进行调试，基于FlutterUME包实现了应用内控制台。该包原本的实现并不支持将Logging包对接其提供的控制台，为了方便使用，对该包添加了相关的支持，并已经将该改动合并回了该包的开源仓库。

Logging包除ALL和OFF夕卜，提供了8个日志等级，从高到低分别为SHOUT、SEVERE、WARNING、INFO、CONFIG、FINE、FINER、FINEST。但是，该包并未指明在

何种情况下使用哪个等级，而是交由开发者自行判断。在本应用中，结合实际情况，为日志等级的使用制定了如表5-1所示的规则。

表5-1日志等级

Table 5-1 Log levels

等级	描述	示例
SHOUT	逻辑上不可能存在的情况，可能 由一些笔误引发，需要修正程序代 码。	kReleaseMode、 kProfileMode、 kDebugMode均不成立。
SEVERE	严重故障，应用已经不能正常运 行，可能由不兼容的应用升级引 发，需要清除应用数据或卸载重 装。	数据库格式异常，无法读取。
	潜在问题，应用主要功能仍然正 常，可能需要用户处	网络故障，无法连接到服务器，导 致未能完成

WARNING	理，但并不紧 急。	数据上传。
INFO	信息消息，确认应用按预期运行。	完成了一段数据的上传。
CONFIG	配置信息，帮助调试可能与特定配 置相关的问题。	内存大小、深色模式开关。
FINE	细节信息，用于Debug，并且仅在 主动执行特定操作时会触发有限 次数。	NavigationBar （应用下方常驻的 导航栏）的selectedIndex变化， 切换到另一个界面。
FINER	更细节的信息，同样用于Debug， 会自动循环无限触发，但频率不 高，是正常人类可以实时检查的程 度。	检测到一个新的QRS波群（每秒 1次左右）。
FINEST	最细节的信息，用于临时Debug， 会自动循环无限触发，且频率过 高，只能运行一段时间后检查历史 记录。	接收到一个新的采样点信息（每秒 上百次）。

其中，INFO及以上级别的消息也需要在UI上展示给用户。CONFIG及以下级别的消息对一般用户没有意义，不在UI上展示。但应用内的开发者工具不受此约束，因为开发者工具相关的UI并不是面向一般用户的。

6、可穿戴动态心电监测应用的测试

6.1项目的测试环境

6.1.1测试的硬件环境

测试共使用了一台Windows电脑、一台macOS电脑、一部Android手机、一个动态心电记录仪。另有部分测试运行于GitHub提供的虚拟环境中，其硬件情况不确定。由于模型可以在CPU上运行，测试时没有显卡方面的需求。Windows电脑是ThinkBook16G4+IAP型号，为主要的开发用设备，也顺便执行了多数测试。macOS电脑为MacminiM2Pro型号，用于运行iOS模拟器以进行iOS平台相关测试,；苹果不允许Windows或Linux系统运行其设备模拟器，虽然也有办法绕过限制，但较为不便。Android手机为RedmiNote8Pro型号，用于执行蓝牙相关测试，因为Android和iOS模拟器均不支持蓝牙。动态心电记录仪为数创医疗的HA301B型号，用于测试设备连接。GitHubActions环境用于持续集成中自动执行测试。

6.1.2测试的软件环境

本地测试环境中，不需要移动设备的测试都在Windows电脑上进行，其系统版本为Windows11专业版22H2，测试时使用最新的stable频道的FlutterSDK，最新的PythonSDK等；另通过WSL2配置了Linux环境，安装了最新的GCC等工具。需要移动设备的测试多数在模拟器中进行，Android模拟器使用了Pixel6ProAPI33型号，iOS模拟器使用了iPhone14ProMax型号；需要真实蓝牙连接的测试在Android手机上执行，为Android11版本。持续集成环境中各种工具也均使用最新稳定版本。

6.2测试数据的来源

测试使用的部分数据是继承至原算法项目的，包括输入数据和模型的预期输出结果。由于其数据量较大，且为异常心电数据，另外从公开数据库中下载了一些正常数据以供测试。具体而言，从PhysioNet下载了CEBSDB数据集[8]，并提取了其中m001号数据的I、II导联的前10分钟数据，再重采样为125Hz并转换为项目中使用的JSON格式。进行重采样一方面是因为原始数据为5000Hz，文件过大；另一方便是因为HA301B型号心电设备的采样率为125Hz，对数据进行重采样后可以更贴近其情况。

6.3测试用例

应用开发过程中，在编写功能代码的同时，也为其编写了许多对应的单元测试用例，本节简单记录这些测试用例的情况。另外，对应用也以演示形式进行了集成测试，但没有明确遵循固定步骤，故不在此列出。

6.3.1Flutter部分的测试

Flutter部分的测试基于该框架自带的测试功能。其写法与主流测试库相近，主要差别在于对FlutterUI的测试有特别支持。在测试中，可以使用pumpWidget方法只执行某一UI组件相关的代码，方便对各部分分别进行测试。其中的pump一词难以翻译，故以下描述中直接使用原文。

6.3.1.1冒烟测试

这是最基本的一个测试用例。该用例在开发过程中有少数几次未通过的情况，基本都是类似提交了不完整的代码导致无法通过编译，或者持续集成环境配置出错等在实际运行前就报错的场景。该用例的通过可以确认项目的基本测试环境正常，而其失败则意味着其他用例的执行情况没有详细查看的意义。

1.进行必要的初始化，如数据库初始化等；基本与主函数相同，但数据库文件路径等使用测试用空路径，以模拟应用首次运行的情况。

2. pump应用主体组件AppCore。
3. 断言屏幕上可以找到一个realTime字符串（在中文环境下为实时心电）。
6.3.1.2标签间隔测试

这是对5.5.3.1节中确定坐标标签文本的间隔的算法的测试。该算法用于确定在心电图上显示坐标标签的间隔，以避免标签过于密集。该算法的输入为心电图的显示时长，输出为标签间隔的时间长度。
由于横屏与竖屏下的测试类似，仅在表6-1中列出了竖屏下的测试用例。
6.3.1.3标准化距离测试

这是对5.5.4.2节所述的标准化距离算法的测试。该算法用于计算两点在心电图上的按小格计算的距离，而非屏幕上实际显示的距离。尽管测试私有方法通常不被推荐，但由于通过公开的UI组件测试这种细节较为繁琐，因此本应用的测试中仍然使用了一■些visibleForTesting。
表6-1标签间隔测试
Table 6-1 Label interval test

显示时长	预期标签间隔
1秒	1秒
5秒	1秒
6秒	2秒
10秒	2秒

测试时定义了一些点作为常量，如表6-2所示。其x坐标含义为以毫秒为单位的时间，y坐标含义为以毫伏为单位的电压。
表6-2标准化距离测试中使用的点
Table 6-2 Spots used in normalized distance test

点	x	y
a	0	0
b	0	0.1
c	40	0
d	40	0.1

这些点之间的标准化距离的预期结果如表6-3所示。
表6-3标准化距离测试的预期结果
Table 6-3 Expected results of normalized distance test

参数1	参数2	预期结果
a	b	1
a	c	1
a	d	显

6.3.1.4三导联心电图测试

该测试用于判定三导联心电图组件的外观基本正常。易用性等方面的测试难以通过自动化测试来完成，仅通过肉眼检查来保证。
1. pump三导联心电图组件Chart3Lead,传入一些假数据。
2. 调整屏幕尺寸为横向1000、纵向2000。
3. 断言屏幕上可以找到三个单导联心电图组件。
4. 断言屏幕上可以找到lead1、lead11、lead111三个字符串。
5. 调整屏幕尺寸为横向2000、纵向1000。
6. 断言屏幕上可以找到一个单导联心电图组件。
7. 断言屏幕上可以找到一个分段按钮。
8. 断言该分段按钮的已选中项仅有一个，为第一项。（默认情况）

9.点击分段按钮的第二项。
10.断言该分段按钮的已选中项仅有一个，为第二项。
11.恢复默认屏幕尺寸。
6.3.1.5心率就绪判定测试

心率组件的底层数据为一个HeartRateData类型的变量，其有intrate和doubleprogress两个字段，默认值均为0。在创建该状态时只会提供其中最多一个参数，rate的正常值为正数，progress的正常值为0到1的浮点数。该测试用于判定代码中对于心率数据是否就绪，即构造时传入了哪个参数的判断是否正确。测试输入及预期结果如表6-4所示。

表6-4心率就绪判定测试

Table 6-4 Heart rate ready test

rate	progress	预期
60	默认	就绪
默认	默认	未就绪
默认	0	未就绪
默认	0.5	未就绪

6.3.1.6时间转文本测试

应用程序中有多处需要将时间以文本形式显示，遗憾的是，Flutter中没有提供合适的方法。因为转换规则较为简单，在开发时自行实现了相关方法，并编写了如表6-5所示的测试。

表6-5时间转文本测试

Table 6-5 Time to text test

时间	显示毫秒	预期结果
1970-01-01 00:00:00.000	否	00:00:00
1970-01-01 00:00:00.000	是	00:00:00.000
1970-01-01 00:00:00.001	否	00:00:00
1970-01-01 00:00:00.001	是	00:00:00.001
1970-01-01 00:00:00.999	否	00:00:00
1970-01-01 00:00:00.999	是	00:00:00.999
1970-01-01 00:00:01.000	否	00:00:01
1970-01-01 00:00:01.000	是	00:00:01.000

6.3.1.7分析报告界面测试

该测试用于确保分析报告界面的外观基本正常。

1.执行必要的数据库初始化等步骤。
2.pump分析报告界面Analytics。
3.断言可以找到所有心律类型的名称，包括屏幕外部分(因为该界面需要滑动才能看到完整内容)。

6.3.1.8数据库测试

该测试用于保证数据库模块的各种读写功能正常。

1.初始化数据库(初始时数据库中没有数据)。
2.查询在DateTime(2022)和DateTime(2024)之间的所有窦性心律类型的心拍总数，断言结果为0。
3.查询在DateTime(2022)和DateTime(2024)之间的所有窦性心律类型的心拍的出现时间，断言结果列表为空。
4.写入时间为DateTime(2023)的窦性心律类型的心拍。
5.查询在DateTime(2022)和DateTime(2024)之间的所有窦性心律类型的心拍总数，断言结果为1。
6.查询在DateTime(2022)和DateTime(2024)之间的所有窦性心律类型的心拍的出现时间，断言结果列表中只有一个元素，且与之前插入的数据一致。

7. 查询在DateTime(2024)之前的最近一个心拍的时间，断言结果与之前插入的数据的时间一致。
8. 查询在DateTime(2022)之后的最近一个心拍的时间，断言结果与之前插入的数据的时间一致。
9. 查询在DateTime(2024)之后的最近一个心拍的时间，断言结果为空。
10. 查询在DateTime(2022)之前的最近一个心拍的时间，断言结果为空。
11. 查询在DateTime(2022)和DateTime(2024)之间的所有采样点数据，断言结果为空。
12. 写入时间为DateTime(2023)，导联I为1mV，导联II为2mV的采样点数据。
13. 查询在DateTime(2022)和DateTime(2024)之间的所有采样点数据，断言结果为空。因为如5.8.8节所述，存在写入缓存。
14. 以之前写入的采样点的时间为起始，写入1000个采样点数据。
15. 查询在DateTime(2022)和DateTime(2024)之间的所有采样点数据，断言结果列表不为空。这里没有断言结果的具体长度，因为缓存大小这一实现细节不在测试范围，只要保证不会过大导致延迟过长即可。
16. 清空数据库。

6.3.2Python部分的测试

Python部分的测试基于pytest框架。

6.3.2.1动态心电智能检测算法的测试

该部分为对智能检测算法的Python版本的测试，用于确保在重构过程中没有改变算法的表现。

测试基于的一个假设是，该算法的原始版本是正确的。在后来的重构过程中发现其实并非如此，但没有找到更好的测试方法。因此，该测试无法保证算法可以正确对心电数据进行分析，只能保证分析结果与算法的原版本或经过确认的修正版本一致。

1. 设置模型文件路径。该路径在Python版本的实现中可以作为常量，但仍然将设置函数提取了出来，因为最终在Dart中调用C++时会需要设置该路径。
2. 读取用作测试输入的心电数据文件。
3. 获取算法的预测结果。
4. 读取存储预期结果的文件。
5. 断言实际的预测结果与预期一致。

6.3.2.2心电数据获取与转换的测试

该部分为对6.2中提到的从公开数据库下载数据并转换格式的测试。测试目标是保证仓库中的数据文件是代码实际生成的文件，未作改动，也没有因代码变动时忘记重新获取而过时。另外，该测试也保证数据满足一些基本约束。

1. 下载原始数据。
2. 断言原始数据的采样率不低于输出结果的目标采样率。（在当前版本中，采样率是从5000Hz降低到125Hz。）
3. 断言原始数据的时长不低于输出结果的目标时长。输出结果是裁剪了开头一段，所以原始数据的时长至少要不比输出结果的时长更短。
4. 断言原始数据中前两段数据的标签为I和II。原始数据中包含除心电外的更多数据，但本项目只需要心电数据，因此不对其他部分进行断言。
5. 转换为输出数据。
6. 从文件读取输出数据。
7. 断言转换的输出数据与从文件读取的输出数据一致。

在持续集成环境中进行测试时，对下载的文件进行了缓存，以免对原始数据库造成过大压力。此外，缓存也加快了测试的执行速度，因为GitHub内部的数据传输速度远远快于从公开数据库下载数据的速度。

6.3.3C++部分的测试

C++部分的测试基于Catch2框架。

6.3.3.1 Pan-Tompkins 算法的测试

该算法的结果的正确性是由人工检查的，不在自动化测试中。自动化测试仅保证算法可以正常执行，不会报错，且输出结果与存储于文件供检查的结果一致。

1. 打开作为测试输入的心电数据文件。
2. 将心电数据文件内容作为JSON进行解析。

3.打开作为预期输出的结果文件。
4.以125田的采样率（测试输入的采样率）初始化算法。
5.对于测试输入中的每个点，调用算法获取实际结果，从预期输出文件中读取预期结果，断言二者一致。
6.忽略输出文件中的1个换行。
7.断言输出文件已经到达末尾。
6.3.3.2动态心电智能检测算法的测试
该部分为对智能检测算法的C++版本的测试，用于确保在迁移过程中没有改变算法的表现。
测试代码与算法代码一样是根据Python版本原样翻译的，只是将相关依赖库进行了替换。因此，测试流程与6.3.2.1中的Python版本的测试流程一致，不再重复说明。
6.4测试结果分析
在对实现进行过各种修正后，各部分单元测试均可通过。在集成测试中也未发现明显问题。因此，可以认为应用基本满足了需求。
另外，对各部分代码的覆盖率基于Codecov进行了统计。其中，Flutter部分的测试覆盖率仅有54%，这是因为代码中的UI部分的测试编写较为繁琐，只对一些核心方法进行了测试。Pan-Tompkins算法的测试覆盖率为78%，一部分未覆盖部分是用于输入输出的主函数，另一部分是算法原实现中有部分代码在测试时未被执行全部分支；由于未充分理解其实现意图，对该部分保留原样，未作改动。其余几个仓库中的代码的覆盖率均为100%，不过手动排除了不可能被正常覆盖的if__name__==″__main__″部分。从测试覆盖率来看，项目的容易出错的部分代码均已经得到了充分的测试，可以保证其基本正确。
7、总结与展望
7.1总结
本文描述了一款面向移动终端的可穿戴动态心电图的智能监测应用。
首先，回顾了国内外心血管疾病现状，介绍了心电监测的用途与原理，以及常规心电图与动态心电图的区别。分析了相关技术与应用的现状，介绍了本项目的主要工作，以及与已有项目的差别。
然后，对项目可能使用的各种技术进行了分析对比，解释了项目的选型理由，并简要介绍了最终使用的各种相关技术。
之后进行了项目的需求分析，明确了目标用户，分析了功能性需求与非功能性需求，并进行了简单的可行性分析。
项目的设计基于Riverpod架构模式，这是一种特定于Flutter框架的架构模式，在各层次模块的划分上与传统模式有一定差异。
在项目的开发阶段，实现了Pan-Tompkins算法，对已有的动态心电图分析算法进行了移植，实现了心电图、心率的显示、分析报告的生成、设备的管理等功能。
最后，对项目的各模块进行了测试，确认了项目的可用性。
7.2展望
本应用在iOS平台上混合使用了Material风格与Cupertino风格的UI。这是一个受开发时间等因素影响的决策，在大幅降低了工作量的同时，也使得iOS平台上的应用界面相较Android平台上的界面在美观的方面有些欠缺。在未来的开发中，希望能够将更多的界面元素在iOS平台上进行贴近于原生的设计，以提高界面的美观度。
由于缺乏医学专业知识，本应用在分析报告中仅对算法的输出结果进行了简单展示，没有进行更专业的分析。希望未来可以与专业人员进行合作，设计一套对普通患者更友好的分析报告生成方式，以提高应用的实用性。
在心电数据的存储方式上，本应用只是将其当作一般数据直接存入数据库。如果结合心电数据的规律专门设计一套压缩存储方式，应该可以进一步减少存储空间的占用，提高应用的性能。
参考文献
[1]中国心血管健康与疾病报告2021[M].科学出版社，2022.GoogleBooks:dtdNzwEACAAJ.217pp.
[2]Cardiovascular Diseases (CVDs)[EB/OL]. [2023-04-01]. https://www.who.int/news-room/fact-s heets/detail/cardiovascular-diseases-(cvds).
[3]张梅，李闯.新型冠状病毒肺炎与心血管疾病[J/OL].武警医学,2020, 31(2): 93-96 [2023-0401]. https://kns.cnki.net/kcms2/article/abstract?v=3uoqIhG8C44YLTlOAiTRKibYlV5Vjs7i8oRR1PAr7RxjuAJk4dHXoiwpCkQQaDHiVuqE4ibqoRq7MTtEodU77lhtrxx96 - zb&uniplatform =NZKPT. DOI: 10.14010/j.cnki.wjyx.20200215.001.

[4]心血管疾病诊断流程与治疗策略[M].科学出版社，2007.GoogleBooks:SmvjPgAACAAJ.617pp.
[5]郑易，岑镇波,胡海雷.动态心电图与常规心电图诊断冠心病患者心肌缺血及心律失常的临 床效果比较[J/OL].现代实用医学,2011, 23(6): 639-640 [2023-04-02]. https://kns.cnki.net/kc ms2/article/abstract?v=3uoqIhG8C44YLTlOAiTRKgchrJ08w1e7tvjWANqNvp-4SNMxa0G_To MEGVGSr_6oPCxpNoM1x0b6GCZOgns7XDOJtOS-WXLP&uniplatform=NZKPT.
[6]季虎.心电信号自动分析关键技术研究[D/OL].国防科学技术大学.2006[2023-04-02]. https ://kns.cnki.net/kcms2/article/abstract?v=3uoqIhG8C447WN1SO36whBaOoOkzJ23ELn_-3AA gJ5enmUaXDTPHrK1dJ7ouyOHRjOllT1Xt93bMTnSixVHYOGAoxiDaUn3V&uniplatform=NZ KPT.
[7]PAN J, TOMPKINS W J. A Real-Time QRS Detection Algorithm[J]. IEEE Transactions on Biomedical Engineering, 1985, BME-32(3): 230-236. DOI: 10.1109/TBME.1985.325532.
[8]RONNEBERGER O, FISCHER P, BROX T. U-Net: Convolutional Networks for Biomedical Image Segmentation [C]. in: NAVAB N, HORNEGGER J, WELLS W M, et al. Lecture Notes in Computer Science: Medical Image Computing and Computer-Assisted Intervention -MICCAI 2015. Cham: Springer International Publishing, 2015: 234-241. DOI: 10.1007/978-3-319-24574-4_28.
[9]HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[EB/OL]. (201512-10) [2023-04-10]. http://arxiv.org/abs/1512.03385. arXiv: arXiv:1512.03385. preprint.
[10]郑贤娴.基于可穿戴设备的移动监护APP[J/OL].电子技术与软件工程，2019(23): 34-35 [2023-03-31]. https://kns.cnki.net/kcms/detail/detail.aspx?dbname=cjfd2019&filename=dzr u201923021&dbcode=cjfq.
[11]吴敏.移动心电监测系统的研究与实现[D/OL].广东工业大学.2018 [2023-03-31]. https://kns .cnki.net/KCMS/detail/detail.aspx?filename=1018866799.nh&dbname=CMFDTEMP.
[12]陈耿铎.移动心电信息监护系统及心电监测算法的研究[D/OL].南方医科大学.2018 [202303-31]. https://kns.cnki.net/KCMS/detail/detail.aspx?filename=1018276660.nh&dbname=CMF DTEMP.
[13]贺其，张鹏，赵安华,等.基于移动平台的心电监测医疗系统的实现[J/OL].山东师范大学学 报(自然科学版),2017, 32(01): 49-53. https://kns.cnki.net/kcms/detail/detail.aspx?dbcode=CJF D&dbname=CJFDLAST2017&filename=SDZK201701009&uniplatform=NZKPT&v=_Ff55Q WuexZwI7yxFvcsmT5z3mSAez3W3-2AxlIytwOqblTi-wsWVWiI4COyjbvg.
[14]GRADL S, KUGLER P, LOHMULLER C, et al. Real-Time ECG Monitoring and Arrhythmia Detection Using Android-based Mobile Devices[C]. in: 2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 2012: 2452-2455. DOI: 10.1109/EMBC .2012.6346460.
[15]WEN C, YEH M F, CHANG K C, et al. Real-Time ECG Telemonitoring System Design with Mobile Phone Platform[J/OL]. Measurement, 2008, 41(4): 463-470(2008-05-01) [2023-03-31]. h ttps://www.sciencedirect.com/science/article/pii/S0263224107000036. DOI: 10.1016/j.measurem ent.2006.12.006.
[16]汪祖民，左长青，秦静，等.基于深度学习的移动远程心电监测系统设计与实现[J/OL].大连 大学学报，2020,41(6): 84-88+98 [2023-03-31]. https://kns.cnki.net/kcms/detail/detail.aspx2dbname=cjfd2020&filename=dali202006015&dbcode=cjfq.
[17]SINGH S N, BHUSHAN M. Smart ECG Monitoring and Analysis System Using Machine Learning [C]. in: 2022 IEEE VLSI Device Circuit and System (VLSI DCS). 2022: 304-309. DOI: 10.1109 /VLSIDCS53788.2022.9811433.
[18]陈桂琛.基于深度学习的心电分析模型的设计与优化[D/OL].北京邮电大学.2021 [2023-0331]. https://kns.cnki.net/KCMS/detail/detail.aspx?filename=1021130914.nh&dbname=CMFDT EMP.
[19]刘荟.基于移动终端分析的可穿戴柔性心电监测系统[D/OL].东南大学.2021 [2023-03-31]. https://kns.cnki.net/KCMS/detail/detail.aspx?filename=1022468120.nh&dbname=CMFDTEMP
.
[20]WANG X, GUI Q, LIU B, et al. Enabling Smart Personalized Healthcare: A Hybrid Mobile-Cloud Approach for ECG Telemonitoring[J]. IEEE Journal of Biomedical and Health Informatics, 2014, 18(3): 739-745. DOI: 10.1109/JBHI.2013.2286157.
[21]JIN Z, SUN Y, CHENG A C. Predicting Cardiovascular Disease from Real-Time Electrocardiographic

Monitoring: An Adaptive Machine Learning Approach on a Cell Phone[C]. in: 2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. 2009: 6889-6892. DOI: 10.1109/IEMBS.2009.5333610.
[22]宋培东.动态心电图的智能检测算法研究与应用[D].华东师范大学,2022.
[23]Mobile Operating System Market Share China[EB/OL]. StatCounter Global Stats. [2023-04-04]. https://gs.statcounter.com/os-market-share/mobile/china.
[24]Flutter/Flutter[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/flutter/flutter.
[25]Supported Deployment Platforms[EB/OL]. [2023-04-04]. https://docs.flutter.dev/reference/supported-platforms.
[26]React Native[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/facebook/react-native.
[27]Xamarin.Forms[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/xamarin/Xamarin.Form s.
[28]Ionic[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/ionic-team/ionic-framework.
[29]Dart Language Evolution[CP/OL]. (2023-04-03) [2023-04-06]. https://github.com/dart-lang/language.
[30]Pyqtdeploy: PyQt Application Deployment Tool[CP/OL].3.3.0. [2023-04-06]. https://www.riverbankcomputing.com/software/pyqtdeploy/.
[31]QPython[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/qpython-android/qpython.
[32]Termux Application[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/termux/termux- ap p.
[33]Kivy[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/kivy/kivy.
[34]Chaquopy: The Python SDK for Android[CP/OL]. (2023-04-05) [2023-04-06]. https://github.co m/chaquo/chaquopy.
[35]Beeware/Toga[CP/OL]. (2023-04-06) [2023-04-06]. https://github.com/beeware/toga.
[36]PASZKE A, GROSS S, MASSA F, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library[C/OL]. in: WALLACH H, LAROCHELLE H, BEYGELZIMER A, et al. Advances in Neural Information Processing Systems 32. Curran Associates, Inc., 2019: 8024-8035. h ttp://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning -library.pdf.
[37]HARRIS C R, MILLMAN K J, van der WALT S J, et al. Array Programming with NumPy[J/OL]. Nature, 2020, 585(7825): 357-362(2020-09-17) [2023-04-06]. https://www.nature.com/articles/s4 1586-020-2649-2. DOI: 10.1038/s41586-020-2649-2.
[38]PILGER D. NumCpp: A Templatized Header Only C++ Implementation of the Python NumPy Library[CP/OL]. (2023-04-05) [2023-04-06]. https://github.com/dpilger26/NumCpp.
[39]KREKEL H, OLIVEIRA B, PFANNSCHMIDT R, et al. pytest[CP/OL]. (2023-04-06) [2023-0406]. https://github.com/pytest-dev/pytest.
[40]Catchorg/Catch2[CP/OL]. (2023-04-05) [2023-04-06]. https://github.com/catchorg/Catch2.
[41]LOHMANN N. JSON for Modern C++[CP/OL]. 3.11.2. 2022 [2023-04-06]. https://github.com /nlohmann.
[42]BIZZOTTO A. Flutter App Architecture with Riverpod: An Introduction[EB/OL]. Code With Andrea. [2023-04-11]. https://codewithandrea.com/articles/flutter-app-architecture-riverpod-intr oduction/.
Material Design[EB/OL]. Material Design. [2023-04-07]. https://m3.material.io/.
致谢
（应盲审要求，隐去致谢内容。

脚注
[1]这里的一个导联是指一对电极之间的电势差，比如□导联是指右臂电极和腿电极之间的电势差。另外，连接电
[2]极与心电图仪器的电缆也常被称为导联。本文中无特别说明的情况下，导联均指电势差而非电缆。
[3]1考虑到PyTorch的底层是使用C++实现的，某种程度上也可以说PyTorch是在LibTorch之上封装了Python接口。
[4]https://developer.android.com/jetpack/guide#recommended- app- arch
[5]严格来说，这可以通过std::variant或union等方式实现，但会失去一部分静态类型检查的好处。

[6] https://pytorch.org/get- started/locally/
[7]苹果官方并未给自己的UI规范进行像Material这样的明确命名，Cupertino这个名字是Flutter在其文档中使用的，可能是基于苹果总部位于美国加利福尼亚州丘珀蒂诺（Cupertino）市的事实。
[8]https://physionet.org/content/cebsdb/1.0.0/