# 目录

[目录 I](#_Toc451943841)

[图目录 V](#_Toc451943842)

[表目录 VI](#_Toc451943843)

[第一章 引言 1](#_Toc451943844)

[（一） 项目背景 1](#_Toc451943846)

[（二） 国内外可穿戴设备技术现状 1](#_Toc451943847)

[（三） 国内智能手环市场现状 2](#_Toc451943848)

[（四） 设计的主要工作与组织结构 3](#_Toc451943849)

[第二章 技术简介及原型平台 5](#_Toc451943850)

[（一） MEMS技术与微传感器 5](#_Toc451943852)

[（二） LinkIt RTOS SDK 5](#_Toc451943853)

[（三） LinkIt ONE 6](#_Toc451943854)

[1. Arduino 6](#_Toc451943855)

[2. LinkIt ONE在Arduino基础上的革新 6](#_Toc451943856)

[（四） 本章小结 7](#_Toc451943857)

[第三章 智能手环的需求分析与概要设计 8](#_Toc451943858)

[（一） 运动数据采集与分析平台整体概述 8](#_Toc451943860)

[1. 需求分析 8](#_Toc451943861)

[2. 概要设计 10](#_Toc451943862)

[(1) 服务器端 10](#_Toc451943863)

[(2) 智能手机端 10](#_Toc451943864)

[（二） 智能手环的需求分析 10](#_Toc451943865)

[1. 功能需求 10](#_Toc451943866)

[2. 非功能需求 13](#_Toc451943867)

[(1) 性能需求 14](#_Toc451943868)

[(2) 质量属性与经济约束 15](#_Toc451943869)

[（三） 智能手环的概要设计 16](#_Toc451943870)

[（四） 本章小结 19](#_Toc451943871)

[第四章 智能手环的详细设计与实现 20](#_Toc451943872)

[（一） 硬件原型的构建 20](#_Toc451943874)

[1. 硬件选型分析 20](#_Toc451943875)

[(1) ARM7EJ-S或ARM926EJ-S 20](#_Toc451943876)

[(2) ADXL345加速度传感器 21](#_Toc451943877)

[2. 电气连线 21](#_Toc451943878)

[（二） 手环嵌入式软件的详细设计 23](#_Toc451943879)

[1. 蓝牙通信 23](#_Toc451943880)

[(1) 蓝牙连接的保持 23](#_Toc451943881)

[(2) 消息的发送 24](#_Toc451943882)

[(3) 消息的接收 24](#_Toc451943883)

[(4) 数据交换格式 24](#_Toc451943884)

[2. 传感器数据的读取 27](#_Toc451943885)

[3. 传感器数据分析 27](#_Toc451943886)

[4. 用户提醒 27](#_Toc451943887)

[5. 线程间的同步与互斥机制 28](#_Toc451943888)

[(1) 蓝牙操作许可 28](#_Toc451943889)

[(2) 传感器共享数据区的读写互斥 28](#_Toc451943890)

[(3) 收到命令后的解释与执行 30](#_Toc451943891)

[6. 使用场景中的典型时序 30](#_Toc451943892)

[(1) 蓝牙连接的建立 30](#_Toc451943893)

[(2) 命令的接收与执行 32](#_Toc451943894)

[(3) 运动状态的分析与统计 33](#_Toc451943895)

[(4) 跌倒报警 34](#_Toc451943896)

[（三） 手环嵌入式软件的实现 36](#_Toc451943897)

[1. 运行流程 36](#_Toc451943898)

[2. 重要数据结构 37](#_Toc451943899)

[(1) 环形队列 37](#_Toc451943900)

[(2) 控制命令堆 39](#_Toc451943901)

[3. 重要逻辑 43](#_Toc451943902)

[(1) 计步算法 44](#_Toc451943903)

[(2) 睡眠质量分析算法 46](#_Toc451943904)

[(3) 用户运动状态的判定 46](#_Toc451943905)

[（四） 本章小结 49](#_Toc451943906)

[第五章 总结与展望 50](#_Toc451943907)

[（一） 总结 50](#_Toc451943909)

[（二） 展望 50](#_Toc451943910)

[1. 当前智能手环原型的不足 50](#_Toc451943911)

[2. 平台未来产品化的方向 51](#_Toc451943912)

[参考文献 52](#_Toc451943913)

[致谢 55](#_Toc451943914)

# 图目录

[图3‑1 运动数据采集分析平台整体用例图 9](#_Toc451812798)

[图3‑2 智能手环包图 17](#_Toc451812799)

[图4‑1 智能手环原型的硬件连线 22](#_Toc451812800)

[图4‑2 运动健康统计数据格式示例 25](#_Toc451812801)

[图4‑3 指令数据格式示例 26](#_Toc451812802)

[图4‑4 传感器共享数据区互斥问题的伪代码示意 29](#_Toc451812803)

[图4‑5 建立蓝牙连接相关的活动图 31](#_Toc451812804)

[图4‑6 命令接收与执行的活动图 32](#_Toc451812805)

[图4‑7 运动状态分析与统计的活动图 34](#_Toc451812806)

[图4‑8 跌倒的检测逻辑 35](#_Toc451812807)

[图4‑9 嵌入式软件运行流程的简化示意 36](#_Toc451812808)

[图4‑10 环形队列的定义 38](#_Toc451812809)

[图4‑11 环形队列插入与移除元素的关键实现 39](#_Toc451812810)

[图4‑12 小根堆的定义 42](#_Toc451812811)

[图4‑13 小根堆的关键操作实现 43](#_Toc451812812)

[图4‑14 计步算法的伪代码示意 45](#_Toc451812813)

[图4‑15 运动状态判定的有限状态机 48](#_Toc451812814)

# 表目录

[表3‑1 睡眠质量记录用例说明 11](#_Toc451812849)

[表3‑2 运动计步用例说明 11](#_Toc451812850)

[表3‑3 跌倒报警用例说明 12](#_Toc451812851)

[表3‑4 定时提醒用例说明 12](#_Toc451812852)

[表3‑5 绑定智能手环用例说明 13](#_Toc451812853)

[表3‑6 解绑智能手环用例说明 13](#_Toc451812854)

[表4‑1 跌倒判定算法真值表 35](#_Toc451812855)

# 引言

## 章节1

### 项目背景

随着社会与经济的全面发展，我国居民对健康生活的关注程度逐步提高。以南京市居民为例，平日参加体育锻炼的居民已对体育锻炼强身健体、消除疲劳、放松心情等功能产生了充分的认可[1]。可以预见，今后将有越来越多的居民参与到体育锻炼中来；同时，不可避免地，我国也正加速步入老龄化社会。与年轻人相比，老年人拥有更加充裕的体育活动时间，但身体条件往往会稍差一些，甚至患有某些慢性病。在这些条件的共同作用下，老年人的体育锻炼更加需要确保自身的安全。

强身健体与人身安全并重的主张同样适用于患有慢性病或正在参加康复训练的人群。对此类人而言，身体运动机能受到或多或少的限制，但运动也是必不可少的。如今，大数据在互联网行业已经得到广泛的应用。如果能够借助物联网与可穿戴设备引领的潮流，采集此类人群的运动数据与身体状况，并进行大数据分析，医师将能够更加有针对性地调整运动处方或康复训练的质与量，从而提高体育锻炼对患者的作用与价值，并最终推广到运动人群的全体中来，为全民锻炼与保健提供个性化的科学指导。

基于以上条件，使用可穿戴设备收集个人的运动与健康数据是一个相对简单的方案。在众多可穿戴设备中，目前又以智能手环最为大众所接受。因此，以智能手环作为最接近人的设备进行数据的采集，并以此为基础建立一个运动数据的分析平台，将会具有良好的前景。

### 国内外可穿戴设备技术现状

可穿戴设备不同于植入人体内、用于医疗用途的嵌入式设备。它由用户佩戴在身上或整合在服装外部，只对用户的各项指标进行获取与分析。无论是硬件方面还是软件方面，可穿戴设备都不属于生命攸关产品。换而言之，可穿戴设备没有严苛的实时性要求，偶然的数据采集超时或数据分析超时并不会对宏观分析结果的准确性产生不可接受的影响，故通常将其定位为软实时系统。

硬件方面，国内外可穿戴设备的技术研究着眼设备的轻质化与在充分满足计算性能的前提下，提升可穿戴设备的续航能力。以为可穿戴设备供电的电池为例，材料和工艺上层出不穷的新技术使得电池的形状与体积打破了传统锂离子电池与干电池在体积和形状上的限制[2]，并且电池的小型化丝毫没有影响电池必要的输出功率[3]。

国内外在软件方面的研究重点在于提高数据分析算法的准确性和效率。以跌倒检测算法为例，有研究表明，在只使用佩戴于腰部的三轴加速度计的情况下，用户在跌倒时，会产生垂直速度以及与竖直方向夹角的特征性变化，并且各方向加速度的变化也满足一定的特征模式。如果将这些条件进行适当的组合逻辑判定，对用户跌倒动作与其它状态各种日常活动的分辨准确度可以达到100%[4]。再如计步算法，通过对人体运动过程中加速度变化加以适当程度的分析与处理，计步的精确度完全可以达到令人满意的程度，实现相对准确的计步[5]。

综上所述，以目前的技术水平，可穿戴设备无论在软件还是硬件上都具备充分的可行性。未来的可穿戴设备将以进一步的小型化、智能化与更强的续航能力为发展方向，存在着巨大的发展与革新空间。

### 国内智能手环市场现状

中国信息通信研究院发布的《可穿戴设备研究报告》显示，2015年，中国智能可穿戴设备市场规模为125.8亿元，增速高达471.8%[6]。基于前文所述的技术支撑，未来几年内，可穿戴设备的市场规模仍然能够保持相当的增速。

然而，可穿戴设备，尤其是智能手环，在广阔的市场前景背后潜藏着巨大的风险。2009年，Fitbit问世。该产品将互联网技术与传统的计步器相结合，开启了可穿戴运动健身类设备的时代。随后两年内，NIKE+FUELBAND和Jawbone up相继发布，可穿戴式运动产品的热潮迅速被掀起。各互联网行业、体育行业公司以及各家创业公司都将目光投向这个领域，试图在这个新兴市场挖掘更大商机[7]。在数年前，智能手环的售价整体较高，产品定位中高端用户。但2014年8月16日，小米手环发布时79元的搅局式定价将智能手环的目标市场群体迅速向下拓展，同时也催化出更多智能手环厂家，并发布了各自的手环产品。目前，市场上存在着大量不同厂家发布的不同型号的智能手环。纵观这些智能手环，它们大多具有计步、睡眠监测、振动提醒等功能，但除去基础功能，各种产品的特有功能基本都与其它类型的产品产生了重叠；没有更多创新性功能的手环产品，则几乎只在外观上有所区别——换而言之，市面上的智能手环产品同质化现象相当严重。

我国市场上现存的智能手环产品还存在着夸大宣传的问题。基于所述的原因，各智能手环的广告宣传都大打“健康”牌，从而吸引更多潜在客户。但事实上，通过包括手环在内的智能可穿戴设备，用户只能获取数据，而不能直接获取健康。智能手环受体积、成本等多方面因素的限制，能够获取的数据极其有限。基于有限数据进行分析而得到的监测指标，其科学性与准确性本身就无法完全保证，存在可质疑之处；另一方面，随着智能设备数量规模的爆炸式增长，与其依靠单个设备通过预置的各项指标与阈值进行离线的健康监测，不如依托云平台和大数据背景，将智能手环作为健康监测设备来单纯地采集数据，由专业的健康管理工作者在专业理论的指导下妥善利用这些海量数据，使其正确、合理地转化为具体的健康管理建议。因此，除了可穿戴设备核心技术上的突破，更需要尽快构建完整的生态系统，满足消费者的切实需求[8]。因此，构建一个运动数据分析平台十分必要，并且相当迫切。

### 设计的主要工作与组织结构

本文主要介绍了运动分析平台中，作为用户终端之一的智能手环这一嵌入式产品的开发原型的设计与实现。这里将从硬件与软件两个方面分别介绍产品设计中的各项重大决策及其原因。

第一章：引言部分，主要介绍了项目背景和当前可穿戴设备，特别是智能手环的技术与市场现状，并描述了本文的主要工作。

第二章：技术简介部分，主要介绍了运动类智能产品中的重要概念，以及本设计基于的原型平台的各项技术与性能指标。

第三章：系统需求分析与概要设计部分，简单地介绍了整个系统需要实现的功能性需求以及必须满足的非功能性需求，并有选择性地详细描述了手环原型这一子系统的概要设计结果。

第四章：详细设计与实现部分，主要介绍了硬件选型时的各项考虑因素、各待选方案的比较、最终的硬件选择结果以及手环原型子系统的详细设计结果与其内部嵌入式软件关键机制的细节实现。

第五章：总结与展望部分，主要介绍了此次设计的产品原型在未来商业化道路上的演化方向，并对现有设计中遗留的问题进行了解释和说明，有望成为今后重构的参考与指南。

# 技术简介及原型平台

## 章节2

### MEMS技术与微传感器

MEMS（Microelectromechanical System，微机电系统）是在微电子技术的基础上发展起来的，融合了硅、非硅微加工和精密机械加工技术制作的，包括微传感器、微执行器、微能源等微机械基本部分以及高性能的电子集成线路组成的微机电器件与装置[9]。构建一个MEMS，通常是为了获得特定的工程性功能，这样的功能往往通过机电或电化学的方法实现。MEMS可以被大批量地制造[10]。

微传感器在全体MEMS产品中占有重要的地位。传统的传感器技术与MEMS结合之后，智能化水平得到了有效提高，并同时具备了小体积、低重量、低能耗、高性能、低成本的特点，便于集成化和多功能化[9]。

### LinkIt RTOS SDK

LinkIt RTOS开发平台[[1]](#footnote-1)由联发科技推出，是为构建智能家居产品、智能工具及物联网设备等产品而具备强大功能的开发平台。其包含的SDK（Software Development Kit，软件开发工具包）以实时操作系统FreeRTOS为基础，并囊括了TCP/IP、HTTP、XML、JSON等大量物联网设备频繁使用的组件。这套SDK支持各种以ARM Cortex-M4架构为基础的SoC（System on Chip，片上系统），并且部分基本功能也支持ARM7系列的MCU（Micro Controller Unit，微控制单元）。LinkIt RTOS SDK完全开源，代码可以由开发者任意更改[11]。值得注意的是，虽然LinkIt RTOS SDK提供了在线版本的API（Application Interface，应用程序接口）参考指南与大量代码示例，但由于SDK的结构尚未完全稳定下来，仍在频繁更改，而大量示例与API说明却没有同步更新，导致它们并不对应最新版本的SDK。这可能会对后续嵌入式软件的开发造成一定的阻碍。

LinkIt RTOS SDK使用的开源协议中，明确了只要开发的程序不使用其它基于GPL协议的开源代码，最终的产品代码就不必开源。这样一来，LinkIt RTOS SDK就成为了对非自由软件开发十分友好的开源平台。即使要开发商业用途的嵌入式软件，在最终的软件产品中也可以无所顾忌地使用LinkIt RTOS SDK。

### LinkIt ONE

#### Arduino

Arduino是一个源自意大利的开源电子原型平台系列。从硬件的本质上讲，大部分型号的Arduino都只是一只AVR单片机加上必要的外围电路，因此成本极其低廉。使用Arduino作为产品开发的平台具备显而易见的优势，包括其跨平台开发的特性、简单清晰的开发过程、从软件代码到硬件设计全面而充分的开放性以及来自社区与第三方的广泛支持等[12]。

Arduino具有多种不同型号的产品，如Arduino Uno、Arduino Mega、Arduino Mini等。[[2]](#footnote-2)除物理尺寸与电气接口以外，这些型号的Arduino使用不同的主控芯片，因此在工作频率、存储空间、I/O端口数量、外部中断数量等方面的技术参数上有所不同；[[3]](#footnote-3)在某些特殊型号的Arduino上，引脚的操作电压也有变化。不同型号的Arduino成本也不同，为了降低产品成本，在开发产品原型时，应当根据实际产品的需求，选择性能等技术指标最符合应用场景需要的型号。

#### LinkIt ONE在Arduino基础上的革新

Arduino是开源的硬件平台，任何单位和个人都可以在Arduino硬件设计的基础上，利用其它硬件设备构建类似Arduino的开源兼容型硬件平台。联发科技于2014年9月22日发布的LinkIt ONE也是Arduino的衍生控制器之一。LinkIt ONE不再使用AVR单片机作为平台的主控，转而使用基于ARM7架构的32位处理器，故而计算性能有明显的提升。同时，LinkIt ONE的可用内存也由使用AVR作为主控的Arduino系列产品中最大的96KB提升到了4MB，从而能够应对更加繁重、复杂的计算任务。

LinkIt ONE除实现了Arduino的所有基本功能以外，还板载了蓝牙、GSM/GPRS、音视频解码等大量功能性模块，SDK中也提供了对这些模块各项功能的API封装，让可穿戴设备与物联网设备的原型与应用得以更加快速、简单地被构建出来[13]。

LinkIt ONE采用了与Arduino Uno逻辑兼容的引脚布局，同时移植了 Arduino的原生API，因此大多数Arduino代码都能够不加调整地直接移植到LinkIt ONE上运行；但是处理器的变化也对其兼容性造成了影响与限制，如I/O引脚的驱动电流过小、引脚操作电压与Arduino Uno不同等等[14]。

### 本章小结

本章介绍了开发智能手环原型所涉及的主要技术，以及采用的原型平台的各项技术特征及各自的优缺点。通过这些描述与说明，可以确认构建目标原型所需的技术条件已经初步具备，因此构建相应原型的设想是可行的。与此同时，原型平台或多或少存在一些缺点与限制。明确这些缺陷与不足，能为之后的软硬件设计与开发工作有效地规避项目初期可能面临的重大风险，保障项目启动后的顺利推进。

# 智能手环的需求分析与概要设计

## 章节3

### 运动数据采集与分析平台整体概述

本次开发的运动数据采集与分析平台包括智能设备、移动客户端与服务端三部分。其中，智能设备，即下文所述的智能手环，起到收集运动数据和通过振动、发声等方式向用户发出提醒或通知的功能；由于智能手机计算性能相比手环更强，移动客户端会被用于收集智能设备提供的运动数据，对这些数据进行进一步的分析与统计，并上传到服务器。此外，移动客户端还被用于发送紧急求助信息，并在用户同时在多个设备上使用移动客户端，或在不同时间使用多个手环的情况下进行历史运动健康数据的同步；服务端则主要管理用户个人基本信息与历史运动数据。这些历史数据通过专业健康工作者的分析，产生健康建议与提醒，再由服务器分发到移动客户端，向个人用户提供反馈以改进自身的运动、生活方式。

在大多数实际使用场景中，服务端对用户而言是透明的。这里的透明有两层含义：一方面，用户感知不到服务端的实际存在，因为如计步、运动提醒、跌倒报警等最为核心的功能并不以服务端的参与为功能实现的必须条件；另一方面，即使移动客户端与服务端失去联系，除了来自服务器的推送，其它功能都应当能够被正常使用，即使相应的数据没能完成与服务器的同步，在暂时的离线状态下，所有由用户自行产生的数据仍然能正常采集并完成初步分析，用户自行发起的任务请求，如设定定时提醒等，也都能够被正确地响应。这些在离线状态下产生的所有数据，在移动客户端恢复与服务端的连接后，将会自动地与服务端同步，保证服务端与客户端数据的相对完整性。

类似地，智能手环也能在一定程度下脱离与移动客户端的连接而独立工作。所有连接断开前完成了设置的定时提醒与通知，在指定的时间仍然会触发，离线期间产生的运动数据，也会在设备存储性能允许的前提下尽可能多地保留下来，在重新与移动客户端建立连接后再发送出去。

#### 需求分析

本平台的三大部分都存在着各自的功能性需求与非功能性需求。受本文主题的影响，本文将主要分析智能手环部分的需求。本系统的用户有两类，分别是普通用户与健康工作者。除去用户注册、登录等极其基本的用例，他们各自对应的用例可以归纳为如下的用例图：

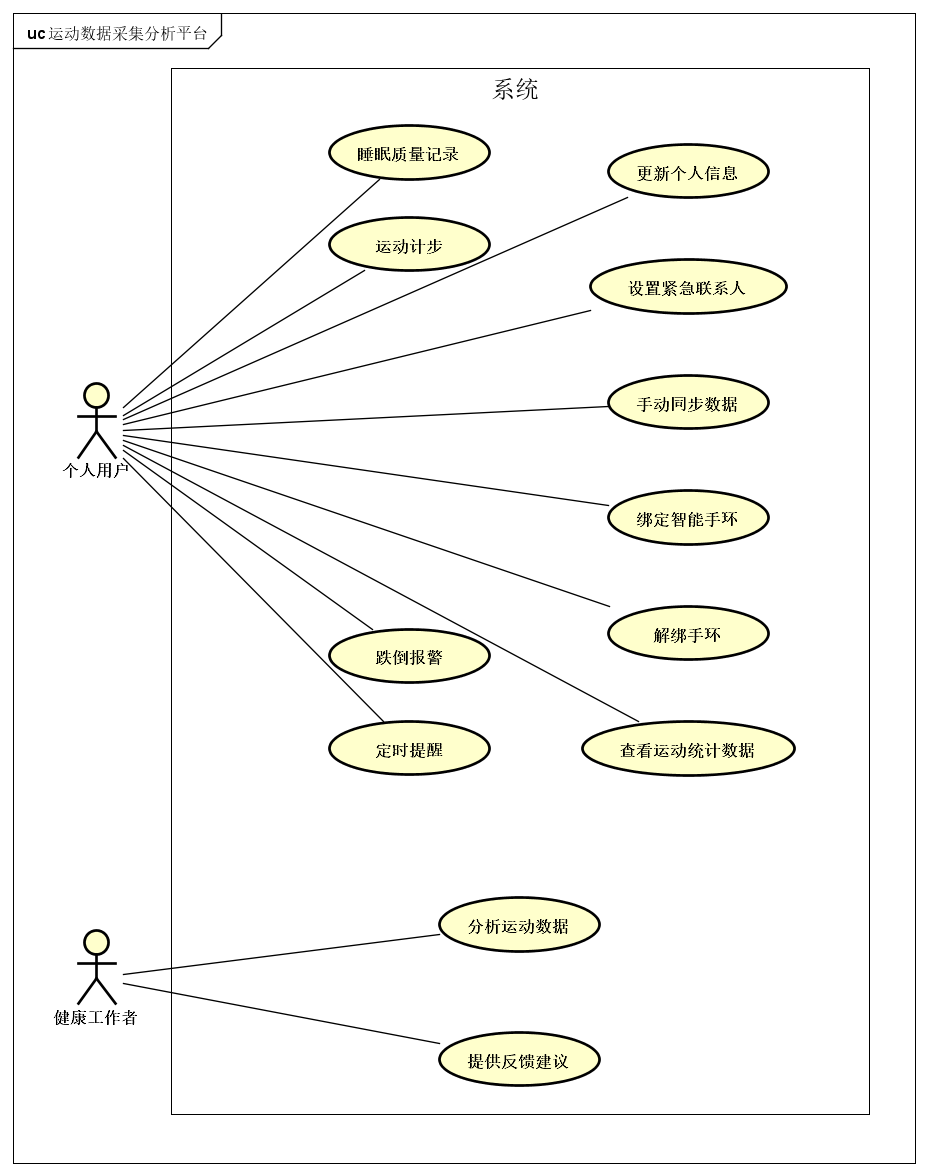


图3‑1 运动数据采集分析平台整体用例图

#### 概要设计

根据上文的需求分析，整个平台三大部分的概要设计已经可以明确，下面将分别给出。由于嵌入式设备端智能手环的设计与实现是本文的重点，这里只对另外两部分的概要设计做简略的描述；智能手环部分的概要设计将在后面的章节中详细展开。

##### 服务器端

服务器端运行一个Web服务器和一个数据库。数据库中存储用户的个人信息和各用户的健康、运动数据。服务端通过RESTful的Web服务提供对外接口，所有用户认证、数据同步等对数据库的增、删、改、查操作通过无状态的HTTP请求完成。这样的服务提供方式降低了服务器处理请求时的资源开销，也简化了移动客户端与服务器端的通信复杂度，适应产品原型的快速开发。此外，这样的对外接口易于扩充，便于今后功能的进一步扩展。

##### 智能手机端

运行于智能手机上的移动客户端是一个基于图形界面的应用。无论是用户的输入还是对用户的结果呈现，交互方式都是基于可视化条件的。移动客户端因此承担在可视化数据与业务逻辑的抽象数据互相转换的职责，并按需主动地与手环或服务端进行通信，主动地控制数据的同步过程。

### 智能手环的需求分析

#### 功能需求

的用例都能直接转换成本次开发的平台的功能需求。其中，睡眠质量记录、运动计步、跌倒报警、定时提醒、绑定智能手环、解绑智能手环六个用例与智能手环直接关联。这些用例的详细说明如下：

表3‑1 睡眠质量记录用例说明

| 优先级 | 高 |
| --- | --- |
| 用例描述 | 在用户佩戴手环进入睡眠状态后，手环自动地收集睡眠状态信息，并进行相关分析判别睡眠质量。每次产生的睡眠质量统计结果在用户醒来后同步到移动客户端。 |
| 刺激/响应序列 | 刺激：用户开始睡眠  响应：手环侦测到睡眠特征后开始睡眠数据记录  刺激：用户在睡眠中产生翻身等不自主动作  响应：手环记录这些动作的特征信号，并据此进行睡眠质量分析  刺激：用户醒来  响应：手环结束睡眠数据的记录，分析睡眠质量并发送到移动客户端 |

表3‑2 运动计步用例说明

| 优先级 | 高 |
| --- | --- |
| 用例描述 | 用户佩戴手环开始走、跑等运动动作后，手环自动地进入计步状态。依据手环上的传感器收集到的运动数据，记录用户前进的单步[[4]](#footnote-4)步数。每次的计步结果在用户停止前进，进入平静状态后同步到移动客户端。 |
| 刺激/响应序列 | 刺激：用户开始前进  响应：手环进入计步状态，收集传感器的运动数据  刺激：用户停止前进，保持静止一段时间  响应：手环结束对此段运动的记录，统计步数结果并发送到移动客户端 |

表3‑3 跌倒报警用例说明

| 优先级 | 高 |
| --- | --- |
| 用例描述 | 在用户跌倒后，手环侦测到跌倒动作并提醒用户。用户如果在30秒内对提醒做出了回应，手环回到原先的工作状态；否则，手环向预先设定的紧急联系人发出求救信息。 |
| 刺激/响应序列 | 刺激：用户跌倒  响应：手环提醒用户发生了跌倒，向用户请求取消发送求救信号的许可  刺激：用户响应手环发出的请求  响应：手环取消发送求救信号，回到原先的工作状态  刺激：用户没有响应手环发出的请求  响应：手环向预先设定的紧急联系人发出求救信息，请求援助 |

表3‑4 定时提醒用例说明

| 优先级 | 高 |
| --- | --- |
| 用例描述 | 用户在移动客户端上预先设定提醒的类型（如服药、运动等）、时间与提醒方式（仅振动、仅发声、振动并发声）。设定的提醒同步到手环后，手环在设定的时间到达后向用户发出相应的提醒。 |
| 刺激/响应序列 | 刺激：用户在移动客户端上事先设定提醒  响应：移动客户端将提醒信息同步到手环，手环保存移动客户端发来的提醒信息  刺激：提醒中设定的时间到  响应：手环按事先设定的提醒类型向用户发出提醒，在用户确认后结束提醒动作 |

表3‑5 绑定智能手环用例说明

| 优先级 | 中 |
| --- | --- |
| 用例描述 | 用户在移动客户端上选择将当前连接到移动端的一个手环进行绑定，这样，在执行解绑操作之前，指定的手环只能与这台智能手机连接并通信。 |
| 刺激/响应序列 | 刺激：用户在移动客户端上要求与智能手环绑定  响应：移动客户端向用户选择的智能手环发出绑定指令，手环记录当前移动客户端的标识符并保存，用于下次连接时的身份识别 |

表3‑6 解绑智能手环用例说明

| 优先级 | 中 |
| --- | --- |
| 用例描述 | 用户在移动客户端上或智能手环上选择解除已经设定好的绑定关系。此后，这个手环又可以自由地与其它移动客户端进行连接并通信。 |
| 刺激/响应序列 | 刺激：用户在移动客户端上要求解绑  响应：移动客户端向手环发出解绑指令，手环解除当前的绑定关系  刺激：用户在手环上直接要求解绑  响应：手环解除当前的绑定关系 |

中的其它用例由于与智能手环没有直接关联，这里不再展开进行详细的描述。

#### 非功能需求

对智能手环的开发而言，非功能需求主要包含性能需求与质量属性两部分。非功能需求还包含对外接口与约束等内容，但由于智能手环只与移动客户端进行直接的通信与数据交换，并且此次的原型开发在技术上的限制条件相当宽松，因此这两部分的非功能需求不需要特别进行列举，这里只会描述未来产品化时必须纳入考量的成本约束。智能手环与移动客户端的数据交换格式将推后至详细设计阶段再进行明确的定义。

##### 性能需求

智能手环显然是一个嵌入式设备，对实时性具备一定的要求。虽然智能手环不要求强实时性，但弱实时性仍然对周期性任务的超时频率做出了“偶发”的条件限制。因此，有必要明确地定义产品的性能的需求。

**对于传感器产生的数据，99%以上的部分要在20ms内完成接收与处理。**20ms的时间限制来自传感器采样频率的最低要求。在本开发周期中，对传感器的采样频率要求最高的是计步功能。因此，计步功能所要求的传感器频率就是智能手环搭载的传感器采样频率的下限。有资料表明，当代职业田径运动员，在百米跑时的步频最大能达到4.95步/s，两腿最快的交换频率也在3.1-4.85次/s[15]。很容易得到推论，普通用户在跑步时的步频不可能比这个值更高。在健步走的情况下，普通成年人的步频最高水平在148步/min，合2.47步/s[16]。由于计步需要同时兼顾走、跑两种基本运动类型，这里取较快的步频4.95步/s。Nyquist取样定理指出，只有在采样频率大于连续信号中最高频率的2倍时，采样后的数字信号才能完整保留原始信号中的信息。显然，用户在运动过程中产生的传感器数据可以认为是连续的，并且其中包含各种频率的连续信号。人因走、跑产生的运动信号具备一定的周期性，在所有系统感兴趣的信号中，这部分信号的频率是最高的，因此，运动信号中的最高频率不妨直接取步频。经验上，实际应用中只有保证采样频率为最高频率的2.56倍～4倍，才能获得理想的结果。因此，在取的情况下，传感器的采样频率应当保证20Hz的下限。因此，每个采样周期的时间间隔为50ms；然而，智能手环上尚有其它任务，如定时提醒、蓝牙通信、同步等在运行，并且这些任务也相当消耗时间。考虑到今后智能手环功能扩展的可能，传感器数据的处理很可能会增加更多的后续流程，因此，当前的原型产品内对传感器数据的获取与处理占用的CPU时间不能过大，再另外考虑到加速度计的数据刷新率未必能够设定在20Hz，每个数据周期消耗的时间不超过20ms的限定相对稳妥一些。

**手环至少能离线存储6小时内产生的全部运动数据。**用户虽然很有可能会随时随身携带智能手机，但未必会随时开启蓝牙连接，与手环进行数据同步。为了保证用户运动数据与健康数据的记录完整性，智能手环必须具备缓存一部分蓝牙未连接期间产生的数据的能力。目前，马拉松可以被认为是人类持续运动的时间极限。参照近几年国内举办的普及性马拉松赛事对业余选手设定的6小时15分的关门时间[17]，认定6小时为最长的可能持续运动时间具备足够的合理性。

**紧急求救信号要在与移动客户端建立连接后1秒内发送到客户端。**紧急求救信号的“紧急”主要体现在一旦发生意外，就需要以尽可能快的速度向外界发送上。出于成本的考虑，智能手环并不直接通过移动网络发送求救信号，而需要通过智能手机上的客户端中转。如果需要发送求救信号时与客户端没有建立起连接，那么手环应当主动地发起连接。由于等待建立连接的时间与移动端真正通过移动网络发送求救信号消耗的时间不是产品所能控制的，手环与移动客户端间的通信速度这一唯一可控因素就必须加以控制，尽最大的可能缩短求救信号从产生到真正发送出去的时间间隔。

**提醒的前后时间误差不能超过5秒。**在移动客户端的规划中，用户设定的定时提醒时间可以精确到分钟。智能手环上可以配备RTC（Real-Time Clock，实时时钟），但RTC的校准仍然要依靠移动端来完成。由于手环已经占据了用户手腕的位置，用户不太可能在佩戴智能手环的同时再戴手表，因此此时用户对时间的判断基准只能取自随身携带的手机上的显示时间，而不是来自手表等精密计时器的精确时间。受本平台的使用场景影响，用户不会产生在手环上设置对时间准确性具有极高要求的任务的倾向，因此5秒的误差是能够满足用户在运动与健康意义上对“准时”的理解的。

##### 质量属性与经济约束

可靠性方面，存在以下需求：

* 没有数据正在传输时，如果与移动客户端的连接断开，手环不能停止工作；
* 与移动客户端的连接断开后，手环应当在1秒内检测到连接断开，并重新发起尝试连接；
* 如果连接未能成功建立，等待5秒后重新进行尝试；
* 等待连接过程中，手环的运动记录相关功能要正常工作；

可用性方面，存在以下需求：

* 如果手环发生软件故障，应当自动重启而非进入宕机状态失去响应；
* 在电池充满电到完全耗尽的一个循环周期内，手环因故障而重启的次数不超过2次。

安全性方面，存在以下需求：

* 手环不能被除智能手机以外的设备连接；
* 如果进行了绑定，手环不能被非绑定智能手机连接；
* 如果手环的硬件被破坏，内部保存的运动数据与个人信息要被擦除；
* 每个用户只能查看自己的个人信息与运动健康数据。

可维护性方面，存在以下需求：

* 如果增加新的运动记录功能，系统现有的体系结构不能发生变化；
* 如果增加新的运动记录功能，在相关算法完备的情况下，要能在1人日内完成。

经济约束方面，存在以下需求：

* 嵌入式软件中不能使用任何收费的软件授权；
* 手环硬件产品附带的嵌入式软件不能被强制开放源代码；
* 在满足计算与存储性能的前提下，硬件成本要尽可能地低；
* 产品不搭载任何非必须的外围设备。

### 智能手环的概要设计

智能手环部分的嵌入式软件可划分为初始化模块、数据格式化器模块、RTOS服务模块、提醒模块、全局共享数据模块、运动健康模块及外部通信模块7个部分。每个模块内部还可继续细分，各模块之间的依赖关系见。

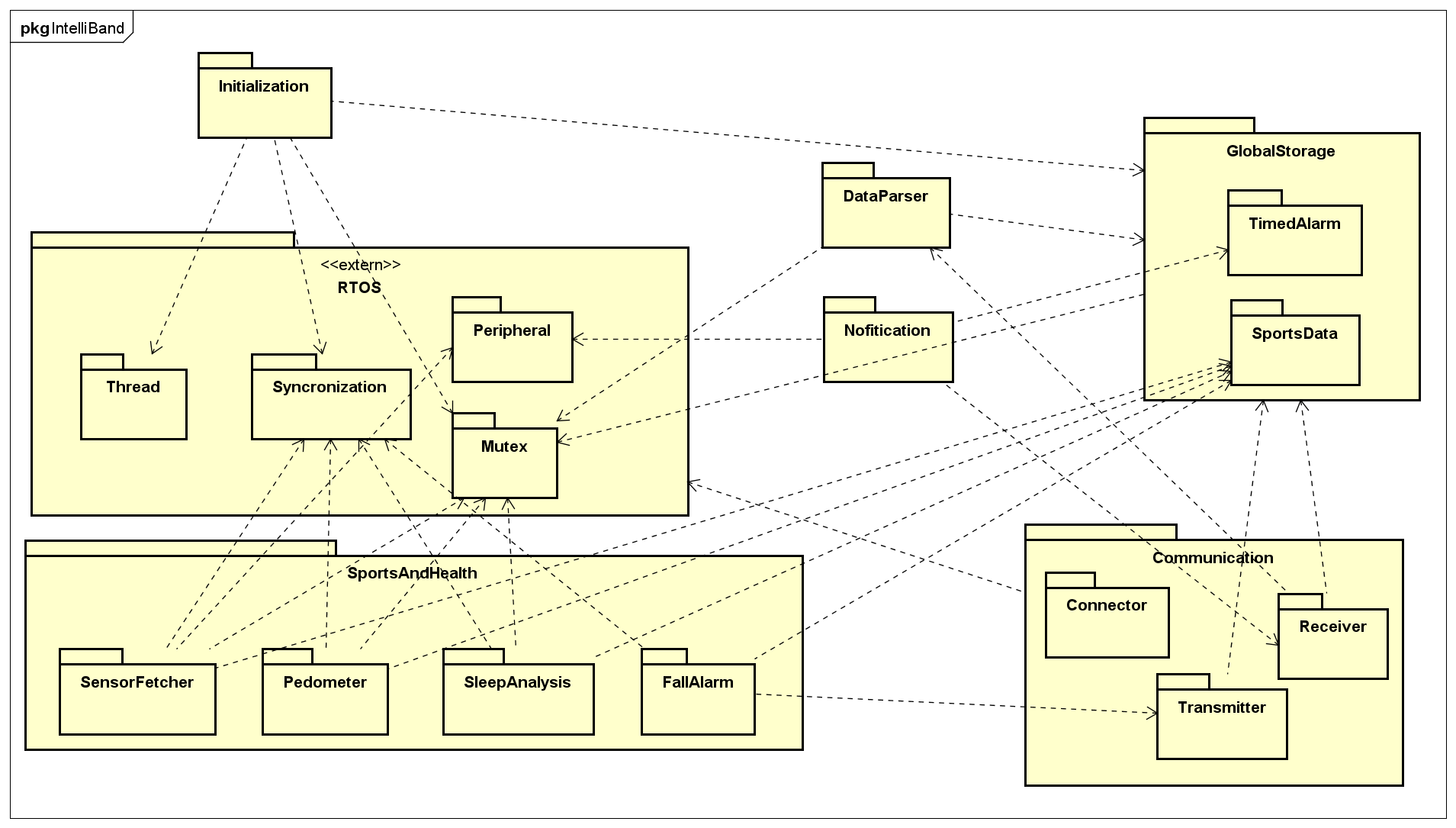


图3‑2 智能手环包图

初始化模块在手环上电时被执行一次，进行软硬件的各项初始化工作，并在初始化完成后建立更多的工作线程，分配给其它模块使用。运动健康模块包含两个工作线程，一个线程从传感器按一定频率取数据，将其存入全局共享数据中，供在另一个线程中运行的计步、睡眠检测、跌倒报警等子模块作为输入数据进行运动统计。在时序上，从传感器读到数据之后才能进行后续的运动统计。但是，之所以不把两部分工作合并在同一个线程内完成，是因为计步、睡眠检测与跌倒报警的执行优先级并不均等。项目背景中已经提到，老年人在进行运动时更要保障人身安全，一旦发生诸如摔倒这样的意外，跌倒报警模块需要排他地优先完成紧急信号的发送。那么，在紧急信号发布期间，计步等普通统计任务很可能不能得到充足的计算资源完成一个新周期传感器数据的分析与统计。基于手环弱实时系统的特性，简单地忽略这一周期的数据不会对统计结果宏观上的准确性造成很大的影响。并且在业界习惯上，将数据的产生与消耗分隔开也是约定俗成的惯例。因此最终决定在运动健康模块上采用这样的设计方案。也正是因此，产生了多个读者与单个写者对同一块共享内存空间的操作，需要利用实时操作系统提供的信号与同步机制来保证数据的完整性与正确性。

外部通信模块承担了与移动客户端直接进行数据交换的职责。这个模块运行三个线程，一个线程用以保持连接，另外两个线程分别完成从移动端到手环与相反方向的数据通信。因为可能存在没有连接的情形，不能直接由运动健康模块调用数据收发的相关接口，这里还是采用共享内存的方式实现线程间的数据传递与共享。

数据格式化器模块没有线程实体，只提供接口给其它模块调用。此模块负责将运动与健康的统计信息以相同的格式封装成数据包，便于移动端在收到数据后以统一的方式进行处理；同时也负责解释收到的各项控制指令或是定时提醒，将其转换成合适的结构暂存在手环中，以便之后的利用。

提醒模块的职责相对简单，只需要定期地检查现有的定时任务，如果到达了任务的预定时间，就产生预定的行为，操纵外围设备发声和（或）振动即可。

### 本章小结

本章阐述了智能手环的概要设计方案，从背景和相关因素的考量两方面解释了做出各项设计决策的原因。概要设计产生后，模块的划分与数据共享方式也就确定了下来。

需要说明的是，在完成概要设计时，对智能手环嵌入式软件的开发过程中使用的相关开发技术的选择也已经隐式地完成了。对这个手环而言，除去多态机制的C++语言是开发语言的最终选择。一方面，嵌入式平台的软件开发对运行效率有较高的要求，另一方面，多态作为相对重量级的运行时机制，对嵌入式硬件性能提出了较高的要求。在这两者之间进行折衷选择，利用C++作为开发语言，但不使用多态理所当然地成为了最终的决策。

# 智能手环的详细设计与实现

## 章节4

### 硬件原型的构建

嵌入式系统是以应用为中心，以计算机技术为基础，软硬件可配置，对功能、可靠性、成本、体积、功耗有严格约束的专用计算机系统[18]。智能手环毫不例外地属于这一范畴。作为可穿戴设备，其对体积与功耗的要求相比其它类型的嵌入式设备更为严格。在通用嵌入式处理器及各类集成度极高的片上系统得到长足发展的今天，嵌入式产品的功能与可靠性主要由软件来决定。因此，硬件选型主要针对成本、体积、功耗三个方面进行对比与选择，而不过多地考虑硬件的工作稳定性。

#### 硬件选型分析

##### ARM7EJ-S或ARM926EJ-S

就智能手环承担的任务负荷而言，入门级嵌入式处理器的性能已经能够满足需求。各种架构的入门级处理器，在扩展功能接近的情况下，成本上的差距并不大，考虑市场占有率及技术资料与开发资源的丰富程度后，决定选择ARM处理器。

在成本接近的情况下，体积与功耗需要进一步考量。市场上的智能手环，除去腕带部分，大小与体积都相当接近。在相对固定的空间限制下，如果硬件外设部分占用的体积能够尽可能减小，电池的体积就可以做得更大，从而提供更好的续航性能。联发科技专为可穿戴设备设计的SoC产品MT2502A很好地满足了以上的要求，在SoC封装尺寸减小到5.4mm×6.2mm的同时，仍然板载了蓝牙的专用处理器[19]，恰好能满足手环与智能手机移动端进行通信的方式，因为目前市售的智能手机几乎都支持蓝牙通信的功能。在这样的考虑下，MT2502A就成为了绝佳的选择。这样，MCU部分自然也就选用了MT2502A搭载的ARM7EJ-S。

然而，不得不承认ARM7EJ-S有些过于古老。ARM已经不建议新的产品设计继续使用ARM7系列处理器[20]，而MT2502也更像是联发科技将2G时代移动电话的单芯片解决方案简单重新包装后发布的产物。但考虑到MT2502A优异的封装尺寸，并且其性能依然能满足需求，继续使用ARM7EJ-S也没有明显的工程性缺陷。ARM已经给出了从ARM7EJ-S平滑升级到ARM926EJ-S的路线图，如果今后发布了封装尺寸达到MT2502A同等水平的基于ARM926EJ-S的SoC解决方案，可以再着手更改硬件选型，并移植现有的嵌入式软件。

##### ADXL345加速度传感器

市售的加速度传感器分模拟和数字两类。MT2502A能够提供ADC（Analog-to-Digital Converter，模/数转换器）功能，但精度有限，因此要获得较高的传感器数据的精确性，选择数字式的加速度传感器更加合适。智能手环一旦上电，加速度传感器就要持续以指定的数据刷新率获取加速度数据。因此，加速度传感器的功耗是硬件选型时一项十分重要的参考因素。ADXL345是Analog Devices当前还在量产的数字式三轴加速度传感器之一。它可以通过I2C或SPI总线与主设备进行通信，最高提供13位分辨率。在以25Hz的频率刷新传感器数据时，最大工作电流仅为60μA，功耗相当理想。此外，ADXL345的封装尺寸为3mm×5mm×1mm，体积也足够小，符合选型条件中对硬件体积的要求。

值得注意的是，Analog Devices的新型号三轴加速度计ADXL344即将上市。ADXL344在传感器性能与接口规格上与ADXL345没有差异，但封装尺寸进一步减小，仅为3mm×3mm×1mm[22]。此外，相比ADXL345，ADXL344的售价更低，目前官方报价为1.15美元。ADXL344大批量生产上市后，完全可以考虑用它来更换现用的ADXL345。

#### 电气连线

由于当前进行的是硬件原型开发，所有元件都选用已经集成到PCB上的模块化插接件，通过面包板和杜邦线完成部件之间的连接。由于杜邦线的连接并不完全牢靠，测试跌倒检测相关功能时，搭建的硬件原型中的一个通过焊接的方式固化了连线，模块化插接件也在万用板上完成了初步布局并以盾板的形式与LinkIt ONE初步集成，便于佩戴后数据的采集与实地实验。

受原型平台LinkIt ONE的数字I/O端口驱动电流过小的影响，振动电机与蜂鸣器选用了三线制模块，电源正负极直接连接振动电机与蜂鸣器的正负极，数字I/O端口提供元件的触发信号。ADXL345模块通过I2C总线与LinkIt ONE相连，电源也直接引自LinkIt ONE的电源输出脚。I2C总线上同一时刻只能有一个设备与主设备进行通信，因此在挂接多个I2C设备时，需要给正在通信的设备上的CS（Chip Select，片选）信号引脚置高电平以明确通信的对象。此次的硬件原型只有一个I2C设备，简单起见，直接将ADXL345模块的片选引脚接在了电源正极上。

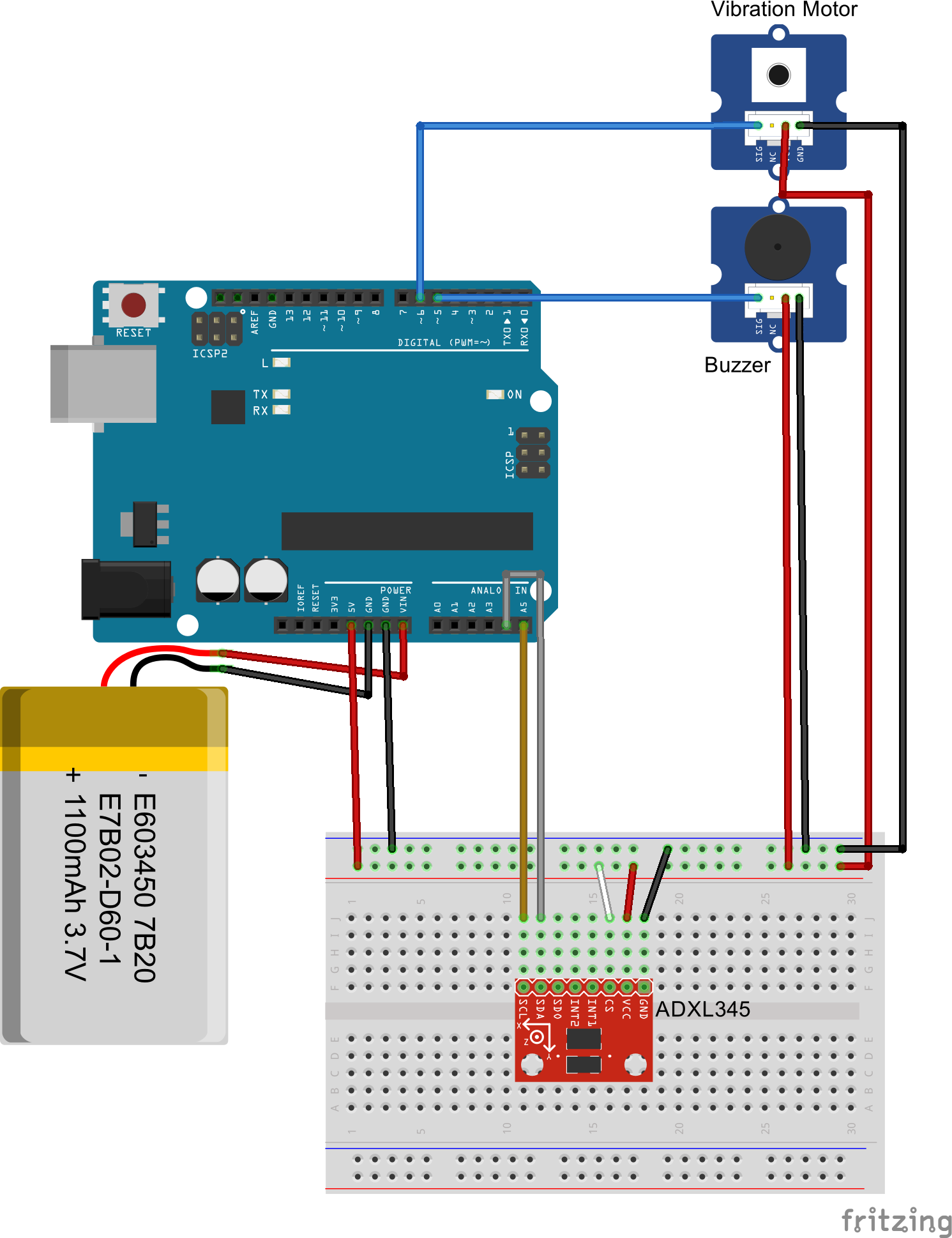


图4‑1 智能手环原型的硬件连线

为了模拟智能手环工作时的能耗情况，LinkIt ONE连接了一块锂离子电池用以供电。整个手环原型的电气连线示意如所示。

之前已经提及，LinkIt ONE的引脚布局与Arduino Uno完全兼容，因此在绘制示意图时使用了Arduino作为LinkIt ONE的替代性示意。这样的做法并不会使得实际产品的物理连线与示意图产生任何冲突或混淆，也不会妨碍对连线方式的正确理解。

### 手环嵌入式软件的详细设计

#### 蓝牙通信

LinkIt RTOS SDK提供了蓝牙连接操作的API。这套API允许LinkIt ONE作为蓝牙主设备开启SPP（Serial Port Profile，串行端口配置文件）服务，与连接到主设备上的从设备以串行通信的方式完成数据交换。由于正常工作状况下，智能手环与移动客户端之间的数据交换量并不是很大，SPP的通信速率能够满足数据交换的要求。

##### 蓝牙连接的保持

在LinkIt ONE上启动蓝牙功能时，LinkIt ONE只能作为主设备工作。受此限制，嵌入式设备上的蓝牙必须始终保持开启状态，等待移动客户端的连接。因此，负责保持蓝牙连接的线程要定期检查蓝牙连接状态，如果连接中断，就要重新发起连接，等待移动客户端的再次连接。

具体到检查蓝牙连接状态的机制上。考虑到蓝牙连接的不稳定性，在当前存在蓝牙连接的条件下，并不能得到这个连接还会保持很长时间的推论。如果在连接中断时尝试对蓝牙串口进行读写操作，嵌入式软件将会发生运行时错误，产生不可预期的行为。因此，存在连接时，对连接状态的确认要尽量频繁一些；相反，在尚未建立连接的情况下，如果连续几次发起的连接都没有得到响应，那么可以预期短时间内不会再有智能手机尝试与手环建立连接。因此，如果连续几次建立连接的尝试没有成功，那么就完全可以拉长下一次尝试建立连接的时间间隔，降低对智能手环计算资源与能源的消耗。

##### 消息的发送

承担数据发送职责的线程在待发送的数据准备好了的情况下，检查蓝牙连接状态，如果连接正常，就向蓝牙串口写入数据，完成消息的发送。如果暂时没有连接，那么这部分数据应当继续暂存，等待之后连接恢复时再次发送。

具体到实现时，因为对蓝牙串口的读写原本就是消耗资源较多的操作，考虑到效率问题，如果由其它线程将需要发送的数据通过线程间消息传递的方式传递过来，再将存储的任务分配给此线程，势必进一步加大此线程对各类资源的占用。因此，运动健康模块在生成要发送的信息后直接存储，再由发送消息的线程读取后进行发送会是更合适的解决方案。

##### 消息的接收

由于移动客户端向智能手环发送控制指令的时间不固定，承担数据接收职责的线程在连接建立的情况下，需要定期检查蓝牙串口上是否有发来的数据。如果有，就将这些数据接收下来，调用数据格式化模块将数据转换成可识别、存储空间要求低的数据格式并保存，供未来执行。

移动客户端会向智能手环发送的指令分两种。一种是定时指令，用户设置定时提醒后，移动端将其同步到手环，要求手环在指定的时间触发相应的提醒动作；另一种是即时指令，在发生特殊情况时，平台主动地向用户发出提醒。前者符合上文所述的处理方式，后者则应当在接收到指令后立即执行。结合下文中用户提醒部分的设计方案，这里将两种指令以相同的方式处理，即将两种指令全部保存下来。因为基于用户提醒部分提出的设计方案，这样的选择不会造成即时指令在实时性上的问题。

##### 数据交换格式

在智能手环和移动客户端之间需要进行交换的数据只有两种类型。一种是运动健康统计数据，由智能手环产生，只向移动客户端发送；另一种是指令，由移动客户端产生，只向智能手环发送。可以确认，在蓝牙串口上传送的数据，收、发的格式分别相对一致，因此可以进行一些格式化的定义。

由于单次交换的数据量不大，轻量级数据交换格式成为了良好的选择。JSON的数据格式完全独立于编程使用的高级语言，但几乎各种高级语言都能高效地生成和解析JSON。与二进制数据相比，数据的长度虽然有所增加，但仍在可接受的范围内。更重要的是，JSON是人类不借助计算机帮助即可读写的数据格式。基于以上原因，选择了JSON作为在智能手环和移动客户端间进行数据交换的格式标准。

运动健康统计数据与指令的数据格式如下：

|  |
| --- |
| {  "type":"pedometer",  "startTime":1459382700,  "endTime":1459383000,  "statistic":  {  "sporting":true,  "steps":100  },  "raw":  [  {"x":0.001,"y":0.0001,"z":0.8726},  {"x":0.0012,"y":0.0323,"z":1.0754},  ...  ],  "comment":"这是一条由可穿戴设备向移动端发送的，带有原始数据的，2016年3月31日8:05~8:10的计步器消息"  } |

图4‑2 运动健康统计数据格式示例

其中，type对应不同的统计类型，如计步、睡眠等，分别使用不同的字符串作为标识符。startTime与endTime用于标记此段统计数据对应的开始时间与结束类型，使用Unix时间戳，精确到秒。statistic字段是一个嵌套的JSON对象，由于各类型的统计数据需要传递不同的统计信息，这部分内容需要根据不同的统计类型，各自生成。为了便于调试，raw字段可以以JSON数组的形式保存来自传感器的原始数据，便于开发者在调试阶段检查与排除故障，实际产品中此字段应当留空。comment字段起到了文字注记的作用，由开发者在其中加入自由格式的文本，同样是为了方便调试，实际产品中，这个字段同样留空。

由于时间戳中不包含时区信息，使用Unix时间戳可能会产生与时区相关的问题。这个问题的解决方案是时间戳表示的时间所在的时区由移动客户端来进行相应的控制。由于RTC的校准依赖移动客户端，消息中附带的时间戳也由移动客户端提供，那么移动客户端只要保证用于校正的时间和提醒时间的时间戳对应同一时区即可。

|  |
| --- |
| {  "type":"medicine",  "time":1458639000,  "vibration":true,  "beep":false  } |

图4‑3 指令数据格式示例

指令中的type用来标记指令的类型。time字段用时间戳标记提醒的触发时间，vibration与beep字段使用布尔值定义提醒触发时需要完成的动作。对于即时指令，将time字段设为-1即可将其与定时指令区分开来。

以上格式示例中的缩进与空白只起到便于阅读的作用。在实际的数据传输过程中，这些与具体内容无关的空白字符与换行符可以也应当全部除去，以便压缩JSON的数据长度，提高通信效率；当然，这个建议不是强制性的，为了保证对未来不同来源JSON数据的兼容性，含有空白字符与换行符的JSON数据也应当能够被正确地解析。同样地，由于JSON的数据格式相对自由，惯用的换行符与空白字符并不能用以分隔两条独立的数据。因此，在进行数据传输时，两条信息之间应当插入一个在数据内容中不会出现的字符作为每条消息的结束符。这里选择了ASCII码二进制值为0x1F的单元分隔符“␟”[[5]](#footnote-5)充当消息的结束符。

#### 传感器数据的读取

对ADXL345加速度数据的读取是一项周期性任务。理想情况下，传感器每个周期产生一份新数据，每份数据都会被读取用以后续分析。实际上，这样的周期性任务可能会被其它突发任务中断，从而丢失一些周期的数据。因此，读取传感器数据的线程要能够应对这样的情况。实际实现时，在每次通过I2C总线读取数据之前，记录当前精确到毫秒的系统级时间，完成传感器数据的存储后，通知数据分析的相关线程开始工作，随后再记录当前时间，根据完成整个流程消耗的时间，决定等待新数据产生需要的休眠时间。休眠时间长度的设置基于这样的原则：一次读取、存储操作消耗的时间与当次数据休眠时间之和为传感器数据周期的整数倍。

#### 传感器数据分析

前文已经提到，计步、睡眠检测与跌倒报警的执行优先级并不均等。在三项任务中，跌倒报警的优先级最高，计步与睡眠检测两者在一个周期内只会选择其中一个执行。显然，跌倒报警需要在数据分析的最初被执行。确定运行计步还是睡眠检测，需要根据当前用户的状态进行判定，而这一判定又需要一些历史传感器数据进行支撑。因此，传感器数据分析需要一块存储区域，存储一定数量的历史传感器数据。这些传感器数据按时间顺序存储，存储区满后，最老的数据被覆盖。

#### 用户提醒

如前文所述，所有接收到的指令都被保存了下来。用户提醒线程需要定期检查所有指令，并执行到达指定时间的指令。由于提醒的时间误差不能超过5秒，这样的检查过程也必须每5秒被执行一次。对于这样的执行频率，指令的检查就需要进行有针对性的效率优化。下文所述的控制命令堆就是优化的方案。具体细节将在之后阐述。

#### 线程间的同步与互斥机制

##### 蓝牙操作许可

LinkIt RTOS SDK提供了对蓝牙操作的相关API，但在LinkIt ONE上受到限制，检测蓝牙连接状态的相关API只能在发起连接的线程上运行。因此，承担数据收发功能的线程不能直接判断连接的状态，并依此决定后续的行为。

原型的嵌入式软件中使用了替代方案，对蓝牙串口进行读写操作的线程使用信号机制进行同步。保持蓝牙连接的线程在成功建立连接，或连接已经建立的情况下激活此信号，对蓝牙串口进行读写操作的线程在每次操作前等待此信号；如果这个信号始终不被激活，两个线程将一直保持休眠状态，不会试图对蓝牙串口进行读写操作。

这样的设计不会造成同步和实时性问题。一方面，LinkIt RTOS SDK实现串口读写时，读与写的内容使用了不同的缓冲区；另一方面，写蓝牙串口的线程在等待信号之前会先等待另一个来自运动健康模块的、表示要发送的数据已经准备好的信号，这样，只有存在需要发送的消息时，这个进程才会响应保持蓝牙连接的线程发出的蓝牙操作许可信号，这样此进程不会被蓝牙操作信号许可频繁地唤醒，虚耗处理器时间。按照系统需求，运动统计数据的实时性没有很强的要求，因此经过此处处理而产生的延时再发送，不会导致性能与实时性方面的需求被违反。

##### 传感器共享数据区的读写互斥

这个问题可以抽象为有一个写者，多个读者，且读者优先的读写者问题。这里的写者对应读取传感器数据的线程，读者对应计步、运动检测、跌倒检测等多项任务。之所以定义读者优先的条件，是因为如果要对一条数据进行分析处理，那么这条数据所有的分析都要被完成，不能出现只完成了一部分分析的情形。并且，有几条传感器数据没有得到处理，实际上并不会严重影响系统的实时性与统计数据的准确性。

此类读写者问题的实现方式已经得到了归纳与总结，需要三个互斥量及一个辅助变量。相应PV操作的伪代码示意如下：

|  |
| --- |
| semaphore mutexSensorDataWrite=1,mutexReaderCount=1,mutexSensor=1;  int readerCount=0;  void dataWriter()  {  while(true)  {  读传感器数据到临时变量;  P(mutexSensor);  P(mutexSensorDataWrite);  复制数据到共享变量;  V(mutexSensorDataWrite);  V(mutexSensor);  休眠一段时间;  }  }  //多个读者全部遵照此模式对共享数据区的数据进行操作  void dataReader\_i()  {  while(true)  {  P(mutexSensor);  P(mutexReaderCount);  readerCount++;  if(readerCount==1)  P(mutexSensorDataWrite);  V(mutexReaderCount);  V(mutexSensor);  读共享变量到本地变量;  P(mutexReaderCount);  readerCount--;  if(readerCount==0)  V(mutexSensorDataWrite);  V(mutexReaderCount);  后续处理;  休眠一段时间;  }  } |

图4‑4 传感器共享数据区互斥问题的伪代码示意

其中，readCount变量用于控制多个读者同时读数据的情形。只要有数据的读者存在，允许写数据的信号量mutexSensorDataWrite就必须保持P操作的状态，直到所有读者都完成任务，才能重新释放，允许写者写数据。

##### 收到命令后的解释与执行

由于数据交换格式已经确定下来，并且与发送的运动健康统计数据相比，数据格式上几乎完全统一，因此直接使用合适的JSON库进行解析，就能获知命令的确切含义。

cJSON[[6]](#footnote-6)是一个用C语言编码，对JSON数据格式进行解析与编码的第三方库，自然也能在C++语言中使用。cJSON采用Public Domain授权，并且完全使用ANSI C编码，移植到嵌入式设备上不需要任何源代码的改动。在读蓝牙串口，获取JSON字符串后，简单的函数调用就可以将其转换成JSON对象，并按照层次与键值对的名称访问到需要的元素。

对于收到的指令，执行时只需根据指令中vibration与beep字段相应的布尔值操作对应的数字I/O端口，即可实现对外设的控制。之后等待一段时间，或在用户响应此次提醒后，再将外设的由工作状态复位到常态。

#### 使用场景中的典型时序

##### 蓝牙连接的建立

建立蓝牙连接前后相关动作的时序如这幅活动图所示。

BluetoothConnector、BluetoothReceiver与BluetoothTransmitter三个线程在逻辑结构上都是一个无限循环，每个线程的每次循环都执行相同的操作指令。所有的与蓝牙连接相关的活动都以BluetoothConnector发出的信号为基准。这一线程每次先检查当前的蓝牙连接状态，如果蓝牙连接正常，就直接向另外两个线程发送允许进行蓝牙操作的信号；如果连接中断，就尝试重新建立连接。重新建立连接的尝试有两种结果，一种是成功建立了新的连接，另一种是因超时而失败。对前者，同样发送蓝牙操作许可的信号；对后者，由于蓝牙连接没有成功建立，此时就不应发送蓝牙操作的许可信号，停止两者读、写蓝牙串口的两个线程继续操作，并且直接进入休眠状态，直到下一次蓝牙连接状态的检查。

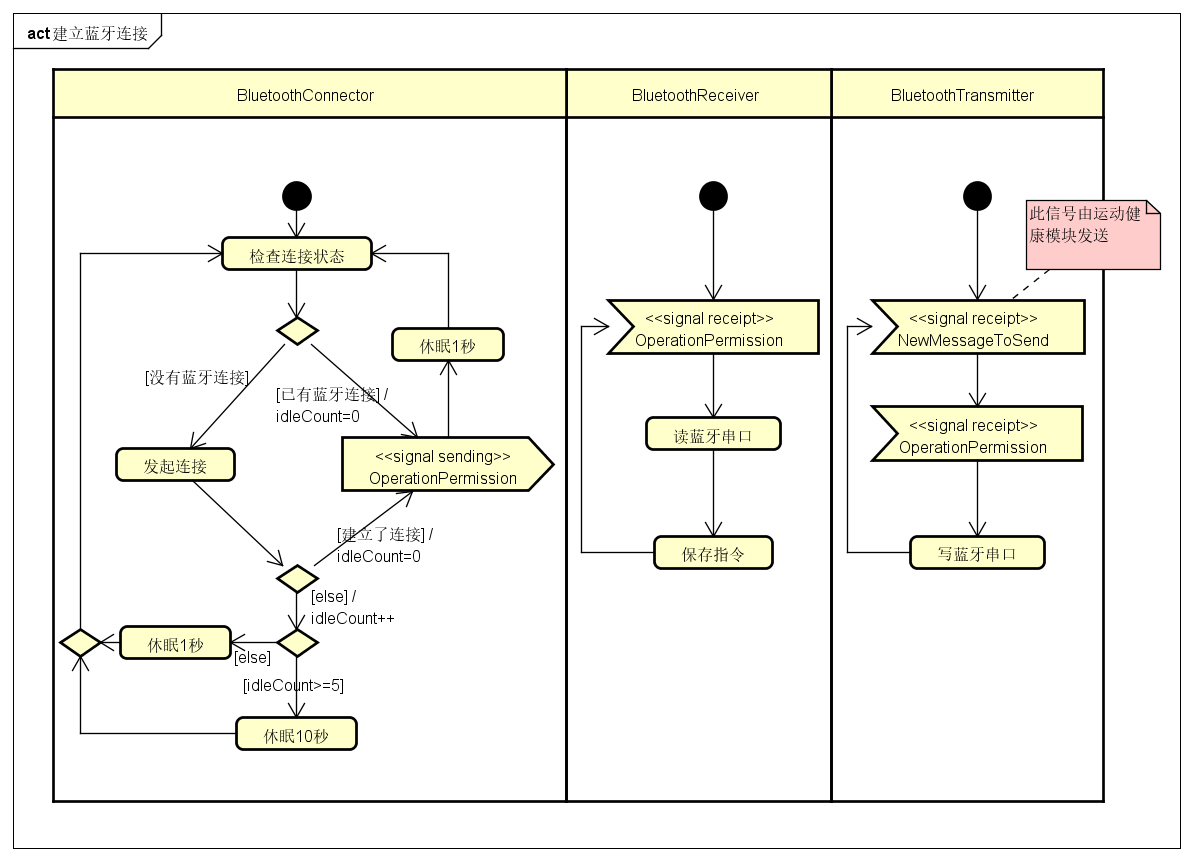


图4‑5 建立蓝牙连接相关的活动图

无论是否发出了蓝牙操作许可的信号，BluetoothConnector都需要等待一段时间后再进行下一次连接状态的检查。等待的时间依据之前几次发起连接或检查的结果做出决策。如果最近一段时间内有过连接，那么短时间的休眠之后即可进行下一个循环，否则，基于之后很长一段时间可能不会再有设备连接的预测，休眠的时间可以被拉长，进一步降低功耗。

##### 命令的接收与执行

命令接收与执行过程中的相关时序如所示：

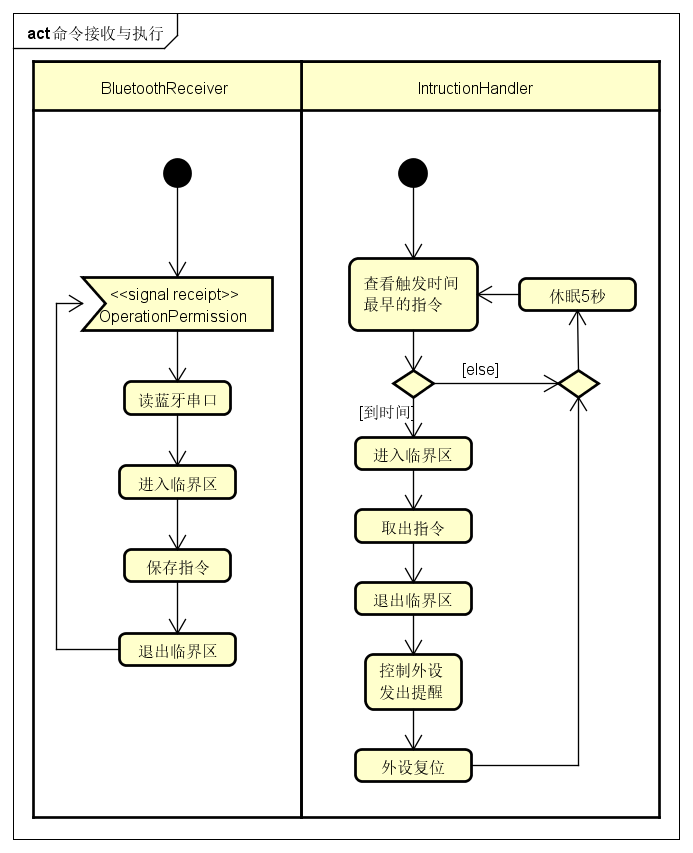


图4‑6 命令接收与执行的活动图

从蓝牙串口读命令的线程与检查时间最早的命令的线程在工作时没有任何需要进行时序同步的地方。两个线程唯一的交叉点在于对指令的读写操作目标是同一块内存区域。与传感器数据的读写互斥问题相比，因为这里的读者与写者都是唯一的，使用一个信号量即可确保同时只能有一个线程对存储指令的内存区域进行访问。

在自然语言层面上，“触发时间最早”的判断是无法适用于需要立即执行的即时指令的。由于之间已经规定过，即时指令中包含的触发时间戳为-1，比任何有效的Unix时间戳数值都要小，因此，在这样的设定下，使用时间戳数值的大小关系来定义两个指令执行时间的先后顺序是可行的，即所有即时指令都会在现有的定时指令之前被执行。

##### 运动状态的分析与统计

运动状态分析与统计的相关时序如所示。

正如详细设计中的描述，跌倒报警、计步与睡眠监测虽然被安排在同一线程内工作，但三者实际上是存在优先级差异的。因此，在获取到新的传感器数据后，首先要进行跌倒的判断。判断跌倒的具体流程将在下一节详细介绍。如果经过判断，跌倒确实发生，那么就进入等待用户响应的流程。如果此时用户还能回应来自手环的提醒，那么可以认为用户这次的跌倒没有造成很大的损伤，用户可以自行爬起甚至继续运动。在这种情况下，此次跌倒应当被忽略；如果用户没能在一定的时限内回应手环的提醒，那么用户有很大的可能已经受伤，此时发出紧急求助信号就是必须的。

在没有跌倒，或发生了跌倒但并不需要发送求助信号的情况下，就可以进行后续的运动状态分析了。通过对睡眠、静止、运动三种状态时传感器特征数据的比较与匹配，手环确定用户当前的运动状态，并与分析上一个传感器数据时的运动状态进行比较。如果两个状态不同，那么上一个运动状态就已经结束，可以对上一个状态的运动与健康数据进行统计，并准备发送给移动客户端。否则，这样的状态还将继续下去，根据当前的运动状态，将数据派发给计步模块或睡眠检测模块进行相应的分析与处理即可。整个流程处理结束后，这一线程即可休眠，等待下一组新的传感器数据到达后再开始一个新的循环周期。

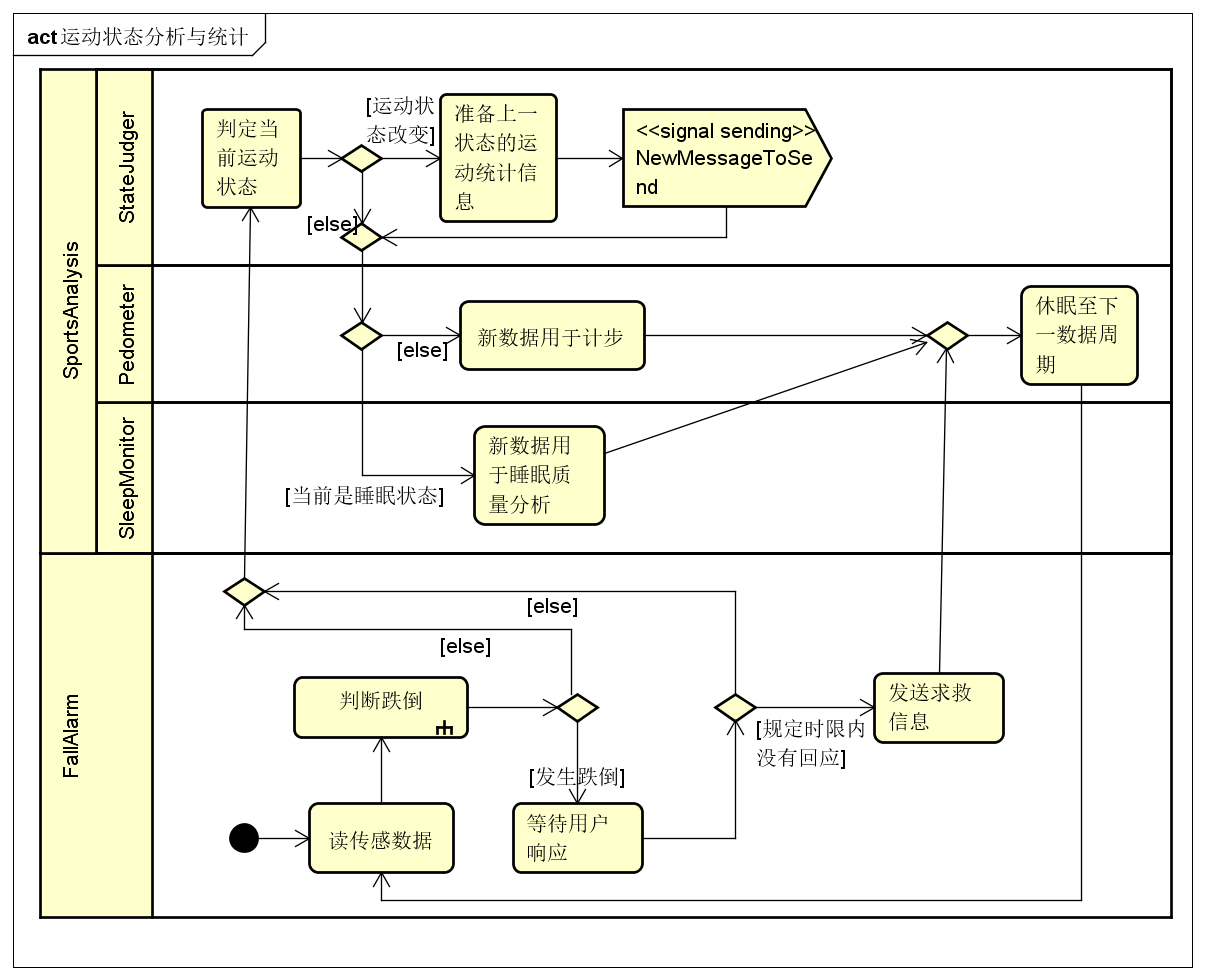


图4‑7 运动状态分析与统计的活动图

##### 跌倒报警

用户是否跌倒的判定完全是基于现有研究与算法的简单实现[4]。进行判定时，需要分别计算用户的瞬时速度、在加速度阈值范围内的停留时间和与竖直方向的夹角大小。由于用户佩戴手环的状况不固定，与竖直方向的夹角大小也无法通过计算就精确地获得，但在跌倒状态下，用户的手臂通常都与地面平行，因此由手环佩戴状况造成的夹角计算误差不会对判定造成很大的影响。理论上来讲，这三个步骤可以如所示的形式一样并行执行，但这三个步骤的计算资源开销都还没有大到并行程序能获得明显收益的程度，因此实际代码中仍然使用了串行编程的方式，将三个步骤的计算结果通过逻辑与运算进行组合，得到最终的结果。

此算法的实现，在布尔逻辑上是完备的，任何一组输入都能得到确定的输出结果。这样的对应关系参见真值表。

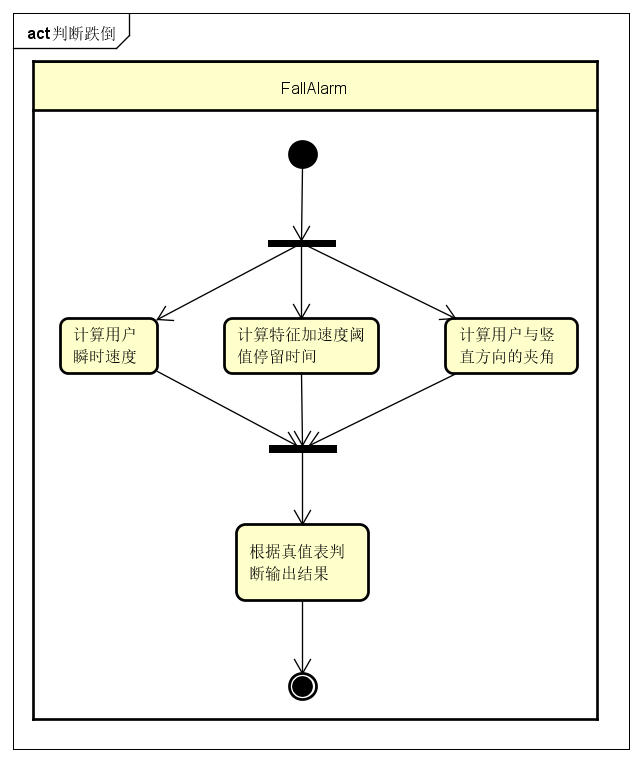


图4‑8 跌倒的检测逻辑

表4‑1 跌倒判定算法真值表

|  |  |  |  |
| --- | --- | --- | --- |
| （m/s） | （ms） | （°） | 结果 |
| ＞0.7 | ＜350 | ＞60 | T |
| ＞0.7 | ＜350 | ≤60 | F |
| ＞0.7 | ≥350 | ＞60 | F |
| ＞0.7 | ≥350 | ≤60 | F |
| ≤0.7 | ＜350 | ＞60 | F |
| ≤0.7 | ＜350 | ≤60 | F |
| ≤0.7 | ≥350 | ＞60 | F |
| ≤0.7 | ≥350 | ≤60 | F |

### 手环嵌入式软件的实现

#### 运行流程

LinkIt ONE SDK已经提供了基于实时操作系统的平台机制，因此整个嵌入式软件都基于这个实时操作系统提供的各项系统服务来完成需求分析中定义的各项功能。由于各项功能被分配在不同的线程中，基于这样的结构描述系统整体的运行流程涉及到大量的线程间通信，复杂度相当之高。如果将整个系统简化成一个按顺序执行的任务，在假定蓝牙连接不会中断，并且从设备加电起就始终保持连接，那么嵌入式软件的运行流程可以通过以下的流程图进行简化的描述：

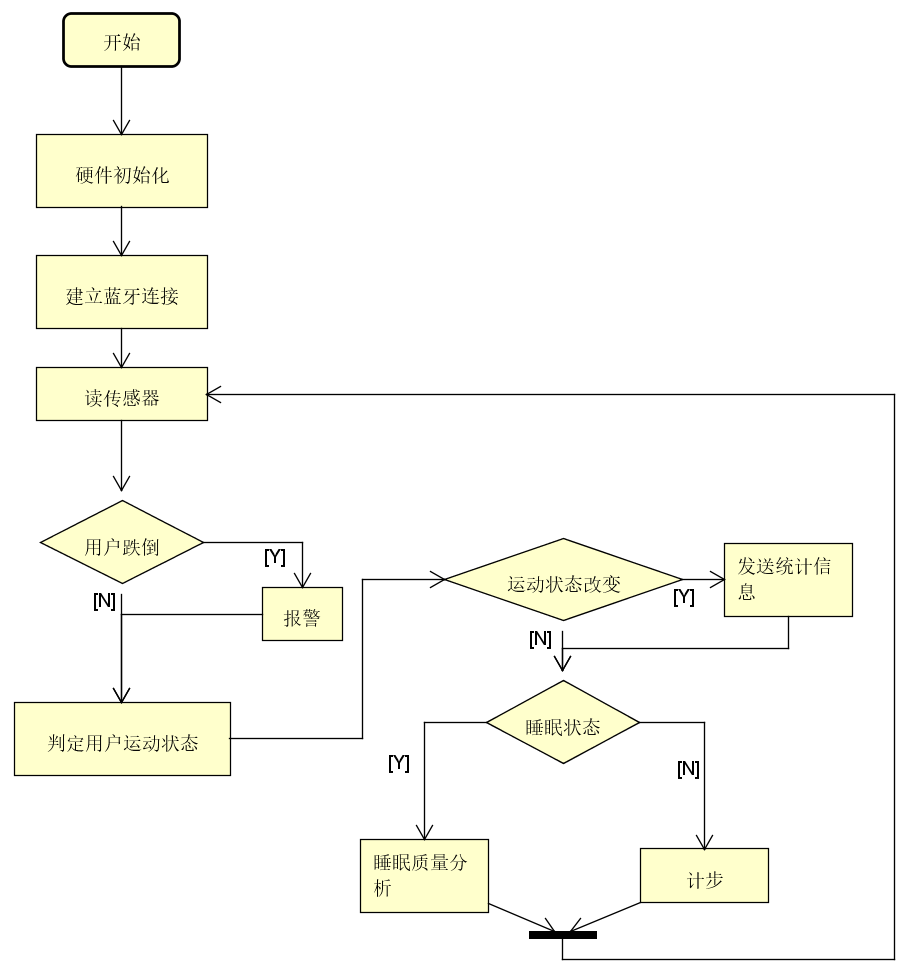


图4‑9 嵌入式软件运行流程的简化示意

#### 重要数据结构

##### 环形队列

健康统计信息的产生基于用户运动状态的切换。用户同一时刻只可能处于一种状态，切换到的下一个状态也是可以根据传感器数据唯一确定的，那么同一时刻最多只会产生一条运动健康统计信息。智能手环可能会处于没有蓝牙连接的工作状态。此时如果有运动健康统计信息产生，就无法立即发送出去，必须先暂存起来，等待网络连接恢复后再进行发送。另一方面，受存储空间的限制，待发送的运动健康统计信息不能无限制地存储，如果超出存储容量限制，就需要舍弃一些旧的信息，以存储更新的信息。

基于以上分析，存储运动健康统计信息的数据结构要满足以下条件：

* 存储空间的大小固定；
* 现有统计信息的存储空间满后，后续的存储操作要以一定的逻辑将之前的信息覆盖；
* 从这样的数据结构中取出信息时，要先取出最早的信息，因为它在未来被新产生的信息覆盖的可能性最大。

这些条件自然而然地将解决方案指向了环形队列。

队列是一种FIFO（First In First Out，先进先出）的线性数据结构。在队列中，所有数据元素只能从一端插入，从另一端移出。环形队列则在一般队列线性存储空间的基础上，臆造出一个环状的存储空间，使得存储空间的首尾相连，并另外使用一组标志位标记队列的首尾元素[23]。

在此次开发的智能手环对队列的实际应用中，只存在两种操作：向队列中插入一个新元素和从队列中取出一个元素。因此，在代码的实现中，环形队列结构中常见的清空队列、批量遍历处理等操作都没有实现，这样既减少了开发工作量，也缩小了二进制代码的规模。程序中，对环形队列的实现如下所示。

|  |
| --- |
| namespace Statistic  {  const int QUEUE\_SIZE=32;  struct MessageItem  {  unsigned int startTime;  unsigned int endTime;  StatisticType type;  cJSON \*statisticInfo;  };  class SendMessageQueue  {  private:  MessageItem messages[QUEUE\_SIZE];  int currentCursor;  int count;  public:  SendMessageQueue();  ~SendMessageQueue();  bool addMessage(unsigned int startTime,  unsigned int endTime,  StatisticType type,  cJSON \*statisticInfo);  char \*popOne();  };  } |

图4‑10 环形队列的定义

|  |
| --- |
| bool Statistic::SendMessageQueue::addMessage(unsigned int startTime,unsigned int endTime,StatisticType type,cJSON \*statisticsInfo)  {  bool ret=true;  int posToWrite=(currentCursor+count)%QUEUE\_SIZE;  vm\_mutex\_lock(&mutexMessage);  //此位置还有未写到蓝牙串口的数据，需要覆盖  if(messages[posToWrite].statisticInfo!=NULL)  {  cJSON\_Delete(messages[posToWrite].statisticInfo);  messages[posToWrite].statisticInfo=NULL;  }  messages[posToWrite].startTime=startTime;  messages[posToWrite].endTime=endTime;  messages[posToWrite].type=type;  messages[posToWrite].statisticInfo=statisticsInfo;  ret=true;  ++count;  if(count>QUEUE\_SIZE)  {  int delta=count-QUEUE\_SIZE;  count-=delta;  currentCursor=(currentCursor+delta)%QUEUE\_SIZE;  }  vm\_mutex\_unlock(&mutexMessage);  return ret;  }  char \*Statistic::SendMessageQueue::popOne()  {  char \*ret;  vm\_mutex\_lock(&mutexMessage);  if(count>0)  {  ...  ret=整理后的JSON字符串;  currentCursor=(currentCursor+1)%QUEUE\_SIZE;  count--;  }  else  ret=NULL;  vm\_mutex\_unlock(&mutexMessage);  return ret;  } |

图4‑11 环形队列插入与移除元素的关键实现

##### 控制命令堆

为了改进智能手环在定时提醒的过程中查找最早触发提醒的效率，控制命令的存储使用了小根堆的结构。堆通常被组织成完全二叉树的形式。如果将完全二叉树存储在一个一维数组内，那么小根堆可以以如下的方式被定义：

含有个元素的序列被称作小根堆，当且仅当



使用小根堆组织控制命令，与直接将控制命令朴素地存储在线性数组内，再使用线性查找的方式取得最早触发提醒的方式相比，在时间复杂度上有明显的优势。下面将通过对时间复杂度的计算来比较两者的效率。

使用线性数组时，每次增加新的控制命令只需朴素地将命令放置在数组末尾，时间复杂度为；取出最早触发的提醒时，一方面要通过逐个查找的方式确定最早的提醒位置，另一方面又要移动剩余命令的位置，时间复杂度为。对于条控制命令的加入删除队列，时间开销最大的操作序列显然是将这些命令按触发时间顺序加入，然后再依次取出。在实际应用中，真正将命令取出并移动其它命令的操作并不很多，更多的操作是在发现下一条最早触发的命令还未到达触发时间，然后不进行任何后续的操作。设一条命令平均被查看次，在这样的操作情况下，条控制命令总的时间复杂度为：



其中，为将条命令加入队列消耗的全部时间；至各项为按触发时间顺序依次取出所有命令时，每条指令消耗的时间；为每次查找操作消耗的时间，对命令全体而言，这样的操作一共被执行了次。

这样，对每条命令进行时间复杂度的均摊分析，不难得到每条命令的均摊时间复杂度为。由于上述的操作序列已经是操作复杂度的最坏情况，因此，这样的时间消耗即为时间复杂度的均摊分析上界，记为。

如果使用小根堆，增加新的控制命令时，将新的命令放置在小根堆最底层，然后逐层调整，直到新的完全二叉树满足小根堆的性质；取出最早触发的提醒时，用底层最靠“右”的元素取代被取出的元素，再由上而下地进行调整，直到新的完全二叉树满足小根堆的性质。对于条控制命令的加入删除队列，时间开销最大的操作序列显然是将这些命令按触发时间倒序加入，然后再依次取出。同样地设一条命令平均被查看次，在这样的操作情况下，条控制命令总的时间复杂度为：



每次插入与取出操作消耗的时间与堆的深度有关，对一个深度为的堆，每次插入或取出操作的时间复杂度为，即消耗的时间最多为。在上式所假定的情形下，插入第条命令前，堆的深度为，插入操作消耗的时间即为；由于对任意正整数，始终有，上式中不添加下取整符号将不会低估时间复杂度。取出命令时，消耗的时间同样为，上式直接将插入与取出操作的合并了起来。

使用对线性数组类似的方式进行复杂度分析，可得到每条命令的均摊时间复杂度为。使用哪种方式时间复杂度更优一目了然。

相应的小根堆定义与关键实现如下。

|  |
| --- |
| namespace Command  {  struct CommandItem  {  unsigned int timeStamp;  bool vibration,beep;  };  class CommandHeap  {  public:  CommandHeap();  ~CommandHeap();  bool empty();  bool push(const CommandItem \*newCommand);  bool peak(CommandItem\* const dest);  bool pop(CommandItem\* const dest=NULL);  private:  int capacity;  int size;  CommandItem \*root;  };  } |

图4‑12 小根堆的定义

|  |
| --- |
| bool Command::CommandHeap::push(const CommandItem \*newCommand)  {  if(size>=capacity)  return false;  else  {  size++;  int j=size-1;  int i=(j-1)/2;  while(j>0)  {  if(root[i].timeStamp<newCommand->timeStamp)  break;  else  {  root[j]=root[i];  j=i;  i=(j-1)/2;  }  }  root[j]=\*newCommand;  return true;  }  }  bool Command::CommandHeap::peak(CommandItem\* const dest)  {  if(!empty())  {  \*dest=root[0];  return true;  }  else  return false;  }  bool Command::CommandHeap::pop(CommandItem\* const dest)  {  if(!empty())  {  if(dest!=NULL)  \*dest=root[0];  const CommandItem tmp=root[size-1];  size--;  int i=0;  for(int j=2+i+1;j<size;j=2\*j+1)  {  if(j<size-1&&root[j].timeStamp>root[j+1].timeStamp)  j++;  if(tmp.timeStamp<=root[j].timeStamp)  break;  else  {  root[i]=root[j];  i=j;  }  }  root[i]=tmp;  return true;  }  else  return false;  } |

图4‑13 小根堆的关键操作实现

#### 重要逻辑

本智能手环产品的软件中，计步算法、睡眠质量分析算法与用户的运动状态判定算法是重要的逻辑。本节将介绍上述三种算法的主要思想与基本原理；由于算法具体的实现细节不是本文的关注点，不会对算法的技术细节进行详细描述。

##### 计步算法

无论是走或跑，人都要迈开双脚前进。在前进的过程中，随着双脚的周期性运动，各方向的瞬时加速度也会产生周期性变化。只要能通过加速度传感器收集到连续的瞬时加速度，通过适当的低通滤波器过滤高频噪声信号，每一步的加速度特征变化就能被检测出来，进而实现计步[5]。

在此次开发的原型中实现计步算法时，考虑到用户佩戴手环位置的不确定性，使用合加速度大小作为输入的数据；考虑到嵌入式设备上的效率问题，原始传感器数据的滤波器采用了简单的均值滤波方法。针对不同用户的步行习惯，实现的计步算法以自适应的方式动态调整判定每步波峰信号的阈值，以瞬时加速度向上突破阈值作为计一步的基准。

计步算法以每一个数据周期加速度传感器产生的合加速度大小为输入，并依据之前几个周期的传感器数据，综合地判定当前时刻的加速度变化特征是否对应迈出了一步。如果产生了一步，算法输出逻辑真；否则，输出逻辑假。

整个算法可以简单地分为三个步骤，分别是滤波、阈值调整和步伐特征检测。计步算法的执行流程描述如下：

对新的传感器数据进行计数，计数戳的值记为；

检查传感器数据的有效性，如果不是有效的浮点数，直接退出；

对原始传感器数据进行16-窗口大小的均值滤波，得到滤波后的传感器数据。其中，；

如果存储的数据数量达到预定的标准，计算新的动态阈值；

判定加速度大小是否出现了1-邻域内的极大值；

如果这个极大值超过动态阈值，继续以下步骤；否则不计步；

如果与上一个最大值的计数戳间隔在合理范围内，计一步，并更新上一个最大值的数据计数，令；否则不计步，只更新上一个最大值的数据计数；

返回计步结果。

计步算法的伪代码示意如所示。

|  |
| --- |
| bool judgeFootstep(double acceleration)  {  tick++;  if(isnan(acceleration))  return false;  bool ret=false;  static bool rising;  filteredAcceleration=经均值滤波后的加速度大小;  //存入滤波后的数据  accelerationRecord[count++]=filteredAcceleration;  if(count==150u)  {  count=0;  重新计算阈值;  }  if(filteredAcceleration>=lastAcceleration)  rising=true;  else  {  if(lastAcceleration>=threshold&&rising)  {  unsigned long currentTick=tick;  unsigned long deltaTick=currentTick-lastPeakTick;  if(deltaTick>FOOTSTEP\_TIME\_LOWER\_BOUND&&deltaTick<FOOTSTEP\_TIME\_UPPER\_BOUND)  ret=true;  lastPeakTick=currentTick;  }  rising=false;  }  lastAcceleration=filteredAcceleration;  return ret;  } |

图4‑14 计步算法的伪代码示意

其中，阈值的产生方法是计算最近150组经过滤波后的加速度大小最大值与最小值的平均值，即：



步伐特征检测有三项判断标准：上次产生的加速度数值成为一个极大值、此极大值大于之前计算出的阈值、这一极大值与产生上一个极大值的时间间隔落在一定范围内。只有这三个条件同时被满足，上述的计步算法才会产生一个计步信号。由于这个计步算法在被连续地调用，如果用户长时间处于静止状态，加速度大小的变化将不会很明显，产生的动态阈值将不足以区分由于传感器偏差与偶然误差产生的数据噪声与真实的步伐特征。为此，阈值除了根据传感器数据动态产生以外，还需要为其确定一个下界，阈值在计算结果与这个下界中取较大的一个。在目前智能手环原型的嵌入式软件中，这一阈值取。

##### 睡眠质量分析算法

医学上，通常使用对EMG（electromyography，肌电图）的研判来分析患者的睡眠质量。但此方法需要在用户体表设置大量的EMG传感器收集数据，日常生活场景下几乎无法实现。有研究表明，深睡期与浅睡期的转换过程中会产生肢体活动，那么这些特定的肢体活动就可以用于区分不同的睡眠阶段，进而用以评估睡眠质量[24]。

在此次开发的原型中实现睡眠质量分析算法时，参考了现有的算法设计，以包括前后各3分钟在内共计7分钟的腕动计数作为输入，利用线性评估的方式分析用户在当前这一分钟内的睡眠状态[25]。睡眠质量分析算法的调用方式与计步算法类似，以每个传感器周期的各轴向的加速度数据分别作为输入参数，提供给算法进行相关的分析。在判定用户结束一段完整的睡眠后，调用此算法的另一接口，获取此段睡眠周期内每一分钟的睡眠质量分析结果，再将获得的结果发送到移动端。

##### 用户运动状态的判定

用户运动状态的判定关系到完成跌倒检测后，将这份数据派发给哪个分析函数进行处理的问题。此过程具有排他性，并且需要即时地产生判定结果，不能基于任何未来的传感器数据做出决策，状态的判定速度又要尽可能地快，避免挤占运动健康信息统计的计算时间。在这些条件的约束下，准确度必须为性能让位而成为次要的目标。因此，智能手环的嵌入式软件中使用的运动状态判定方法不会追求准确性。通过使用一些简单的、可以作为运动状态变更必要条件使用的判断标准，这一方法在快速判定状态的同时，差错率也维持在了可以被容忍的水平。

目前，手环中定义了三种运动状态：睡眠状态、空闲状态与运动状态。这三种状态之间的转换条件与动作可以以下面的有限状态机表示。由于在每种不同状态下，向其它状态跳转的条件（加速度大小、夹角大小等）各不相同，因此，这样的有限状态机是一个Mealy状态机。基于此有限状态机实现运动状态的判定算法，在状态的转换过程中可以同时完成源状态临时数据的清理工作（如运动分析数据的生成与存储）与目标状态的初始化。

运动状态判定的执行流程描述如下：

计算合加速度大小与当前加速度与竖直向下方向的夹角；

依据有限状态机的定义，在当前状态与计数器数值的条件下，基于加速度大小与夹角大小确定下一个状态，；

如果，生成上一状态的统计数据并将相关计数器恢复到初始状态；否则，相应地修改计数器的值；

用新的状态覆盖当前状态；

返回新的状态。

其中，合加速度的大小为：



当前加速度方向与竖直向下方向的夹角为：



状态判定使用的有限状态机如所示：

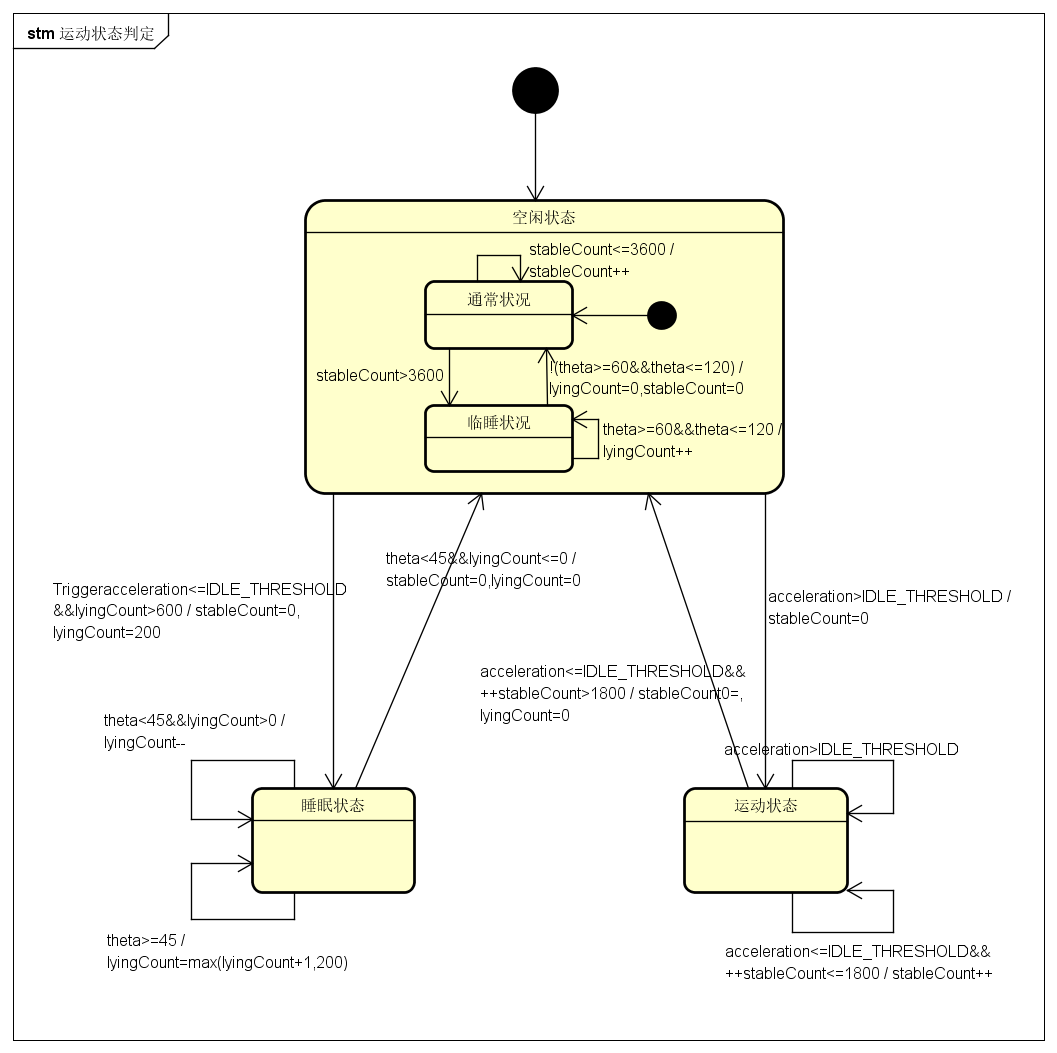


图4‑15 运动状态判定的有限状态机

确定了有限状态机的各个状态与状态之间转换的条件与动作之后，代码上的实现就很容易了。用户运动状态的判断同样以各数据周期的加速度数据作为输入参数。对于每一组传感器数据，算法再依据内部保持的上一个状态判定此时状态是否发生了变化，以及会变化到哪个新的状态。如果确实发生了状态转换，这个算法还将执行有限状态机上规定的状态转换时触发的行为。

智能手环的实际软件实现中采用了较为简单的switch-case语句，而没有使用状态模式等其它的方法来实现有限状态机。一方面，这个有限状态机并不包含很多状态，虽然存在子状态，但嵌套的深度与数量仍然在可控范围内，并且空闲这一复合状态相对稳定，今后如果要增加更多状态，也不会再产生更深层次的子状态嵌套；另一方面，基于对运行效率的考虑，在概要设计阶段就已经确定了不会使用C++语言的多态机制，而这是实现状态模式所必备的条件之一，因而状态模式受选定的开发技术限制，无法被应用到嵌入式软件中来。

### 本章小结

本章介绍了手环两大关键部件MCU与加速度传感器的硬件选型以及智能手环嵌入式软件的详细设计与实现。

选型过程中，以成本与体积作为重要的参考依据，最终选定了ARM7EJ-S与ADXL345作为产品原型中使用的硬件。这样的硬件选型结果是在各方面因素产生冲突时，依据实际情况进行折衷与取舍的工程思维的体现。选型结果中产生了选择老旧型号产品的决策，这可能对今后产品的演化与升级造成一些障碍，因此还对今后硬件升级时重新选型的思路与方向提出了工程上的建议。

详细设计部分从智能手环的各个功能作为切入点，详细地解释了软件各部分组件的行为以及关键部分的业务逻辑细节。通过对各线程之间的交互与协作的描述，给出了软件产品的运行时行为层面上的详细设计方案。实现部分重点描述了详细设计中涉及到的重要数据结构的实现，并从性能方面与其它候选方案进行对比，给出了选择当前数据结构的原因。最后对嵌入式软件产品中涉及的核心算法做出了描述，并给出了算法相应的伪代码示意。

# 总结与展望

## 章节五

### 总结

在本文完成之前，整个运动数据采集与分析平台的原型软硬件都已经顺利结束了相关开发，开始了小范围的应用性测试。目前，整套平台的服务端架设在校园网内；搭载于运行Android操作系统的智能手机上的移动客户端实现了各项计划中的功能，除用户界面外，核心业务逻辑已经满足了实际应用的需要；运行在iOS上的移动客户端正在开发；多个智能手环的硬件原型被组装起来，分发给了项目组内的开发人员进行实际的数据采集与日常体验。在整个项目的所有工作中，除睡眠分析算法由组内其他成员实现外，我承担了智能手环从需求分析到编码测试整个过程的大部分工作。当然，相比整个大项目的工作量，我完成的工作仍然是微不足道的，并且无论是硬件上还是软件上，都还存在很多有待改进之处。

### 展望

#### 当前智能手环原型的不足

手环目前基本实现了计步算法与睡眠质量分析算法，准确度与市售产品相比水平接近，但与相关文献中提及的理想准确度还有一定差距。科学、合理的运动与健康分析的前提是数据的准确性，因此，今后对相关算法的探索与研究还要不断深入下去。

类似地，用户当前运动状态的快速判定也应当成为算法的研究方向之一。尽管由于对效率的严苛要求，完全正确的算法在工程角度并不适用，但这并不妨碍继续进行各运动状态的特征研究，进而演化出在效率可以被接受的范围内精确性更优的近似算法。

在手环与移动客户端的通信方面，运动与健康统计数据是由手环在一个运动状态结束时，主动发送给移动客户端的，客户端只能被动地接收。想象以下场景：用户在健身跑的同时希望查看当前已经跑的步数。在目前的实现下，除非用户停下，否则用户看到的只能是开始运动前的计数0。这样的系统行为对用户而言并不友好，会影响用户体验。为此，手环与移动客户端间的通信机制需要进行一些调整，以适应客户端主动请求统计数据的情形。

目前，硬件原型在LinkIt ONE上搭建，快速开发的意图得到了充分的满足。然而，当前的硬件原型与智能手环在外观上的差距还比较大，LCD显示屏等智能手环上的常见外设也还没有被搭载。后续的硬件演化，可以考虑逐步抛弃LinkIt ONE的整体框架，在沿用MT2502A SoC与其它外设的同时，逐渐从PCB（Printed Circuit Board，印刷电路板）的层次开始设计，摒弃现有的模块化插接件，转而将相应元件与外围电路直接焊在PCB上，从而逐渐消弭硬件原型与最终产品的外观差异，这也更有利于今后更大范围的测试活动。

#### 平台未来产品化的方向

目前的原型化产品只完成了普通用户使用本平台过程中所需要的智能手环、移动客户端与服务端三个部分。这部分组件可以完成运动健康数据的采集与自动化的简单分析，但分析的结果与健康工作者所能提供的专业化建议还有较大的差距，整个系统预想功能的完整实现还差最后一步。要改变这一局面，供健康工作者使用的数据分析应用需要尽快着手开发。这样，运动健康数据平台的生态系统中才能够出现真正意义上的数据消费者，保证这个新兴生态系统的健康发展。

在产品化的道路上，智能手环自身也还有很多工作要做。一方面，原型毕竟只是原型，要将其投放到市场成为真正的产品，外观需要进行专门的工业设计；另一方面，智能手环不能步市场上的同类产品的后尘，走上同质化的道路——相反，今后应当继续开发如步态分析等更多的专业化功能。相关的研究已经有所进展[26][27]，如果能将MEMS与嵌入式技术结合，小型设备在医疗、康复领域将有望发挥更大的作用。

# 参考文献

1. 栾世超，雍明.南京市居民体育锻炼现状调查研究[J].体育时空,2015,第11期:28.
2. 聂翠蓉.新型电池能助可穿戴设备融入大众生活[J].前沿科学,2015,第3期:91-92.
3. 日本松下公司研发供可穿戴设备使用的微型针形锂电池[J].电源技术,2014,第11期:1986-1987.
4. Bourke, AK，van de Ven, P，Gamble, M，O'Connor, R，Murphy, K，Bogan, E，McQuade, E，Finucane, P，OLaighin, G，Nelson, J. Evaluation of waist-mounted tri-axial accelerometer based fall-detection algorithms during scripted and continuous unscripted activities[J]. Journal of Biomechanics,2010,43:3051-3057.
5. 陈国良，李飞，张言哲.一种基于自适应波峰检测的MEMS计步算法[J].中国惯性技术学报,2015,第3期:315-321.
6. 吴茂林.2016年智能可穿戴式设备的那些事儿[J].通信世界,2016,第1期:60-61.
7. 高一乐.智能运动手环的发展现状分析[J].当代体育科技,2015,第33期:202,204.
8. 陈念昭，谢敏.从智能手环的热潮看现代信息化健康管理[A].浙江省医学会健康管理学分会第七次学术年会[C],2014:317-319.
9. 董永贵.微型传感器[M].北京：清华大学出版社,2007:6-7.
10. （美）徐泰然（Tai-Ran Hsu）著；王晓浩等译. MEMS和微系统-设计与制造[M].北京：机械工业出版社,2004:1-2.
11. 联发科技创意实验室.什么是联发科技 LinkIt™ RTOS 开发平台?[EB/OL]. http://labs.mediatek.com/site/znch/developer\_tools/mediatek\_linkit\_rtos/what\_is\_linkit\_rtos/index.gsp, 2016-03-07/2016-03-26.
12. 陈吕洲.ARDUINO程序设计基础 第2版[M].北京：北京航空航天大学出版社,2015:4-6.
13. 联发科技创意实验室.什么是联发科LinkIt™ ONE 开发平台[EB/OL]. http://labs.mediatek.com/site/znch/developer\_tools/mediatek\_linkit/whatis\_linkit\_one/index.gsp, 2016-03-01/2016-03-26.
14. 联发科技.联发科技 LinkIt ONE 开发指南[J/OL],v1.3:2016-03-01/2016-03-21.
15. 杨文学.中外男子优秀百米运动员步幅、步频的对比分析[A].中国体育科学学会运动训练学分会第六届全国田径运动发展研究成果交流会[C],2013.
16. 练艺影，王正珍，李雪梅，王娟，米欢，李萌.20～59岁年龄段普通成年人健步走推荐速度及步频的研究[J].北京体育大学学报,2012,第7期:49-51:57.
17. 厦门国际马拉松赛组委会.2016建发厦门国际马拉松赛竞赛规程[EB/OL]. http://www.xmim.org/cn/eventtopic.asp, 2015-10-26/2016-04-30.
18. 桑楠.嵌入式系统原理及应用开发技术 第2版[M].北京：高等教育出版社,2008:8.
19. MediaTek Inc. MT2502A SOC Processor Technical Brief[J/OL], v1.0: 2014-09-08/2016-03-27.
20. ARM Ltd. ARM7 处理器系列[EB/OL]. http://www.arm.com/zh/ products /processors/classic/arm7/index.php, 2016-03-27.
21. Analog Devices. ADXL345[J/OL]. http://www.analog.com/media/cn/tec hnical-documentation/data-sheets/ADXL345\_cn.pdf,Rev.D:2013-02/2016-01-13.
22. Analog Devices. ADXL344[J/OL]. http://www.analog.com/media/en/tec hnical-documentation/data-sheets/ADXL344.pdf,Rev.0:2012-04/2016-05-06.
23. 吴伟民，严蔚敏.数据结构 C语言版[M].北京：清华大学出版社,2009:58-65.
24. De Koninck J，Gagnon P，Lallier S. Sleep positions in the young adult and their relationship with the subjective quality of sleep[J]. Sleep,1983,6(1):52.
25. 冯晓明.基于腕动信号的睡眠质量监测装置设计[D].华南理工大学,2014.
26. 孙嘉利，唐丹，钟世镇.三维步态分析的研究与应用[J].中国组织工程研究与临床康复,2007,第5期:944-948.
27. 岳雨珊，俞君，张文毅，谢斌，朱毅.三维步态分析的研究进展[J].中华生物医学工程杂志,2011,第4期:372-376.

# 致谢

这份设计，寥寥数十页纸，背后蕴藏着很多人给我提供的帮助和支持。在这里，向各位我背后的支柱道一声诚挚的感谢。

感谢刘海涛老师在论文撰写与智能手环软件体系结构设计方面给我的指导。为了支持本次原型的开发，刘老师在硬件设备上对我提供了极大的支持，准备了大量元器件供我选用。

感谢周坚石同学对睡眠检测算法进行的深入研究，并实现了这个算法。初次接触LinkIt ONE时，我对它的特性掌握得并不十分完善。凭借我们共同的单片机开发经验，两人共同摸索，明确了一些技术手册中没有标注的技术特性。在编码阶段，原先的方案因受API的限制无法直接利用时，他与我一同尝试寻找替代方案。

感谢王思议同学为开发移动客户端付出的巨大贡献。在制定手环与移动客户端之间的数据交换格式时，我们进行了深入的探讨，最终确定下了现行的方案。测试阶段，他不厌其烦地与我一同反复测试各种环境下的蓝牙通信，有效地保证了智能手环内嵌入式软件的质量水准。

感谢项目组内的其他同学。整个平台现今的规模、功能与质量是团队内所有成员共同努力、高效合作的结果。

感谢LinkIt RTOS SDK与cJSON的开发者与开发团队。他们提供的高质量库，有效地帮助我快速完成了整个软件原型的开发工作。

最后，衷心地感谢身边其他的同学以及学弟学妹们。他们在我的研究遇到瓶颈，项目进展缓慢的时候与我进行了推心置腹的沟通，并对我的情绪进行了有效的疏导。没有他们，很难想象我在这几个月里如何才能始终以积极乐观的心理状态投入到本次开发的项目中来。

1. 关于LinkIt RTOS HDK（Hardware Development Kit，硬件开发工具包）的信息，参见http://labs.mediatek.com/site/znch/developer\_tools/mediatek\_linkit\_rtos/hdk\_intro/index.gsp [↑](#footnote-ref-1)
2. Arduino全部型号的列表，参见https://www.arduino.cc/en/Main/Products [↑](#footnote-ref-2)
3. 关于各型号Arduino的技术参数，参见https://www.arduino.cc/en/Products/Compare [↑](#footnote-ref-3)
4. 人前进时足部动作的一个周期，包括两个腾空周期和两个支撑周期，被称为一个复步。每个复步又包含两个单步。 [↑](#footnote-ref-4)
5. 单元分隔符US不是一个可显示字符。在现代计算机系统中，此ASCII码对应的字符如果出现在文本中，通常会以空白字符代替显示。 [↑](#footnote-ref-5)
6. 更多关于cJSON的信息，参见https://github.com/DaveGamble/cJSON [↑](#footnote-ref-6)