# 基于iOS的智能手环配合系统的设计与实现

## 【原文对照报告-大学生版】

报告编号：39da0f137dc55b52　　检测时间：2019-06-03 20:34:49　　检测字数：15,886字

作者名称：夏海泉　　所属单位：华南理工大学继续教育学院

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2019-06-03

检测结论：

| 全文总相似比 | = | 复写率 | + | 他引率 | + | 自引率 | + | 专业术语 |
|---|---|---|---|---|---|---|---|---|
| 17.47% | | 17.47% | | 0.00% | | 0.00% | | 0.00% |

其他指标：

自写率：82.53%

专业用语：0.00%

高频词：数据,系统,开发,产品,设计

典型相似性：　无

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，请正确标注引用

自引率：引用自己已发表部分占全文的比重，请正确标注引用

自写率：原创内容占全文的比重

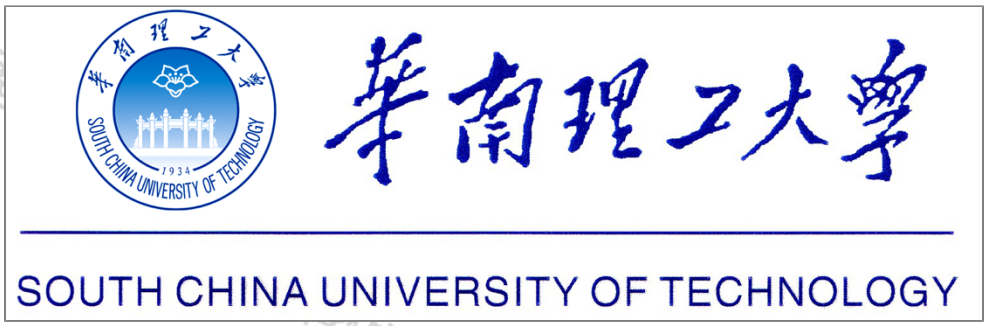
专业用语：公式定理、法律条文、行业用语等占全文的比重

典型相似性：相似或疑似重复内容占互联网资源库的比重，超过60%可以访问

总相似片段：181

期刊：19 博硕：45 外文：1 综合：0 自建库：0 互联网：116

学 号 201707192013018




毕业论文

# 基于iOS的智能手环配合系统的设计与实现

| | |
|---|---|
| 学生姓名 | 夏海泉 |
| 专业名称 | 计算机科学与技术 |
| 指导教师 | 伍尤发 |

网络教育学院

2019年 5 月 24

目录

摘 要

为满足使用公司手环产品的用户，随时随地跨移动端掌握自身运动状况的需求,研究并设计基于iOS的智能手环配合系统的设计与实现。手环将采集到的数据使用蓝牙4.0通过蓝牙传输协议传输至APP, APP在接收到从蓝牙发来的数据后，本地处理并缓冲，同时页面展示，数据收集并存储到本地数据库后上传至云端服务器。APP可实时显示手环检测到的数据，也可查看历史数据，同时对相关数据作出建议供用户参考，及其一整套有效方案。 系统测试结果表明,系统较准确显示出测量到运动步数、心率、消耗卡路里等值。并有判断心率是否正常等显示，该系统实时采集用户运动数据,能有效监测运动状况,并给出相关建议，还能将数据同步至云端。系统具有实时性强、可靠性高和人机界面友好等优点,和好地满足了用户需求，可在公司内普遍使用。

最后，总结了本文所做的研究工作，回顾了整个系统开发过程。

关键词 智能硬件 实时数据 数据采集 数据存储 数据同步

Abstract

In order to meet the needs of users who use the company's bracelet products, they can research and design the design and implementation of the iOS-based smart bracelet coordination system anytime and anywhere to grasp the needs of their own sports conditions. The data collected by the bracelet is transmitted to the APP through the Bluetooth transmission protocol using Bluetooth 4.0. After receiving the data sent from the Bluetooth, the APP processes and buffers the data locally, and the page is displayed, and the data is collected and stored in the local database and uploaded to the cloud. server. The APP can display the data detected by the bracelet in real time, and can also view the historical data, and make recommendations for relevant data for the user's reference, and a set of effective solutions. The system test results show that the system accurately displays the measured number of steps, heart rate, and calories burned. And to determine whether the heart rate is normal, etc., the system collects user motion data in real time, can effectively monitor the movement status, give relevant suggestions, and synchronize the data to the cloud. The system has the advantages of strong real-time performance, high reliability and friendly human-machine interface, and satisfies the user's needs well, and can be widely used in the company.

Finally, the research work done in this paper is summarized and the whole system development process is reviewed.

Key words intelligent hardware real-time data data acquisition data storage data synchronization

第一章 绪 论

1.%2 系统的概述

在本章中，首先介绍了基于iOS的智能手环配合系统的设计与实现

课题的实际意义和课题研究背景，接着介绍了本课题的开发方法，最后介绍了本课题的来源和主要研究范围以及应用的方向。

1.1.%3 本课题的背景

现公司的手环型号较多，且又各个手环功能都不一样，有的只是功能验证性，有的是正在研发，还有的是量产后期维护迭代的改进版本。数据的获取和处理，存在的重复且繁琐的工作，不利于内部的数据共享、分析，没有达到高效化、协同化的办公理念。

本课题来源公司于实际需求，结合公司手环特征，建设以公司为单位所有手环数据处理的系统，经过内部会议讨论后立项。在此基础上提出本课题基于iOS的智能手环配合系统的设计与实现。

1.2.%3 本课题的研究意义

借助现代无线通信技术及移动客户端开发技术，将手环数据实时传输到APP，数据处理后缓存至本地，并依据数据同步逻辑，适时将数据上传服务器。做到数据集中处理分析，合理利用内部资源加强信息共享，提高工作效率。

1.3.%3 本课题的目的

本课题的目的在于对系统方案的可行性及风险从各方面进行分析，从而对基于iOS的智能手环配合系统的设计与实现的开发过程中的保证系统的可靠性、稳定性。

1.%2 数据管理系统概述

1.1.%3 手环产品背景

1982年，Seiko通过其收购的公司推出了可编程手环，这可称为智能手环的起源。 手环产品的诞生时间很早，并且在技术可以实现的情况下，它们是早期的尝试。 在技术可以满足手表产品的时代，手环成为智能手表硬件形式链的垂直。 在市场上，安全手表生产和移动健康手表产品也在智能手表的垂直细分领域。 存在这些形式是为了能够满足垂直消费的相应需求。。

从手环智能的产品用户来看，该产品的潜在客户是200万中国老年人或1500万中国老年人。 这是一个庞大的用户市场，有达到3500亿美元的规模，如数据显示在2011年全球市场吸引了1400万人，未来的用户人数难于想象。 可穿戴的智能设备是电子商务行业必不可少的东西之一。

1.2.%3 系统开发方法介绍

我们采用了敏捷开发方法，用户需求是核心，我们采用迭代和逐步的方法进行软件开发。 在敏捷开发中，软件项目在构建开始时分为多个子项目，每个子项目的结果都通过可见性、完整性和操作使用进行测试和特点。 也就是说，大型项目被分成多个彼此相关的较小项目，但可以独立运行或单独运行。

1.3.%3 开发本系统的重要理念

创造实际使用的产品，通过早期低成本的原型设计来控制产品风险。

品质不过关决不发布产品，即便是落后于既定的发布日期。

通过产品发布后持续地调整优化，来确保产品从发布时就表现优异，直至最后得到稳定、可靠的产品

从系统创意到形成产品4个阶段

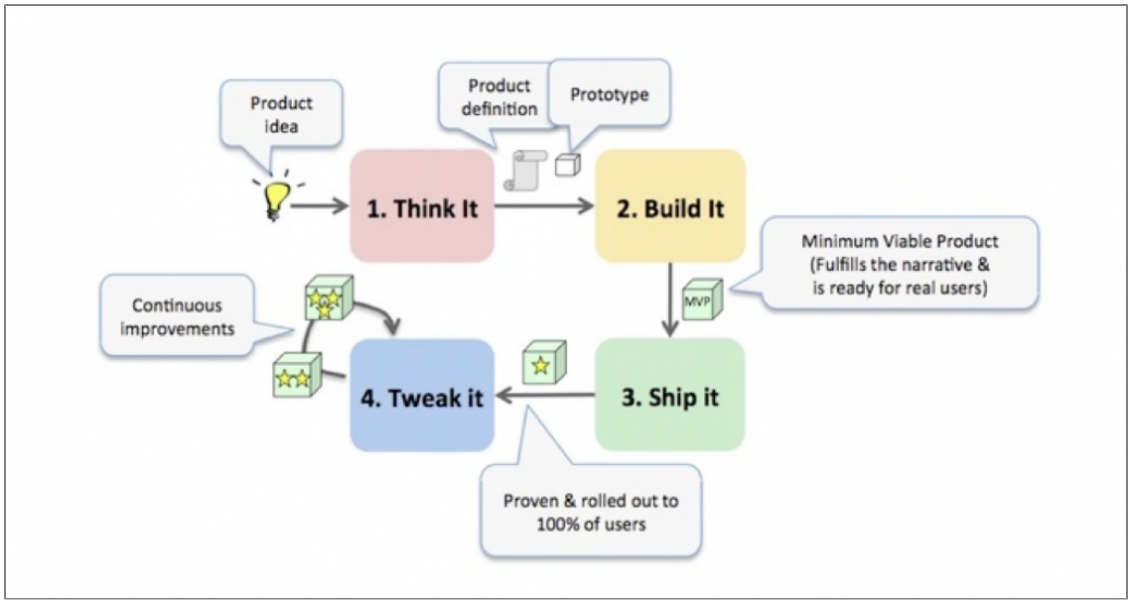


图1-1 从系统创意到产品的4个阶段示意图

系统风险控制

1、 产品开发最大的风险构建一个错误的产品。

2、 思考阶段：以较低的成本，大幅度降低产品风险。

3、 开发阶段：高运营成本难以降低产品风险，因此应尽可能短。。

4、 Release阶段：随着产品的发布和客户使用，产品风险持续降低。

5、 调整阶段：随着时间的推移，产品逐渐完善，运作成本持续下降，小队成员们可以开始逐渐去做其他事情。

。


图1-2 风险示意图

阶段之Think it

以下是工作流程

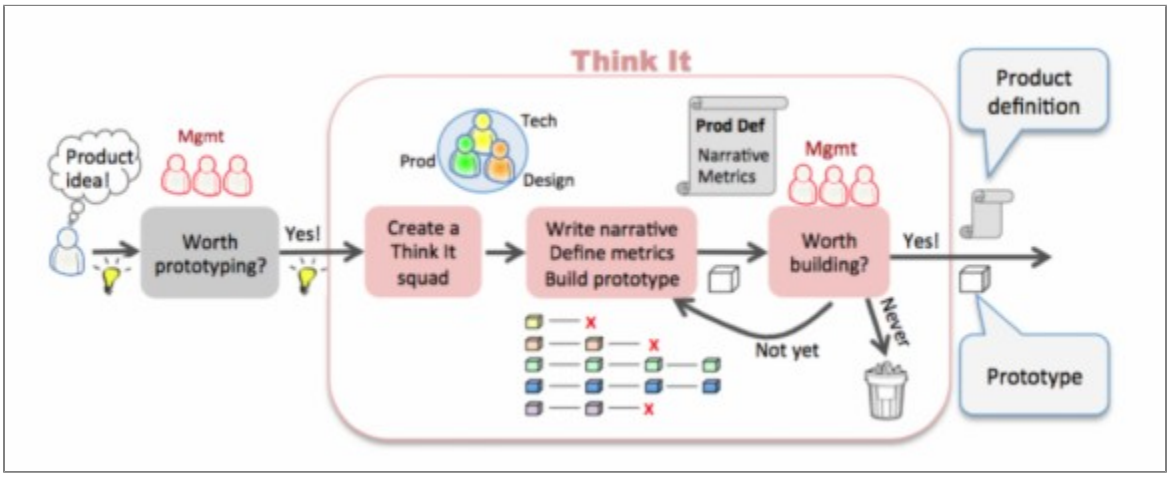



图1-3 Think it 阶段示意图

过程说明

目标：提出一个可以传达它的可运行的原型和一个非常吸引人的故事描述。

输入：产品创新

过程：

1. 形成一个Think it小团队。
2. 编写故事描述，定义要达到的指标。
3. 原型搭建。
4. 确认是不是有建立MVP的必要。

定义完成：

1. 经理和Think it团队同意该产品值得建设（或者该产品永远不值得建设并且应该被丢弃）。

2. 说明：描述：这是一个主观决定，硬数据不支持。 在Ship It阶段之前它不会是稳固的，因此尽快到达Ship It阶段是我们所希望的。

故事描述（narrative）

故事描述，这是一个很短的文档描述，对如下问题作做出了回答：

1. 为什么我们要开发这个产品？谁能从中受益？如何受益？
2. 我们期望可以从这个产品提升哪些关键指标？诸如：会增加多少音乐播放量，增加多少下载量，增加多少登录量等等。
3. 我们该怎样才能判断这个产品是不是成功的？预期又是什么的？
4. 能不能是我们的产品上一个台阶呢？（其中指的是，我们预期这个产品在既定指标上会带来至少双倍的提升。如果只是在度量指标上略有提升，最好有个更强有力的理由，例如战略方面的原因）

关于故事描述的说明：

1. 故事描述不是所谓的产品愿景规划。它不包括特性清单、预算、资源计划。更像是一个用数据说话（数据驱动）的意愿陈述。

2. 讲一个生动的而又能贴切产品的故事，是这其中最重要的故事。

举例：一个新手环新产品发布上线了，它是如何人们解决运动中的数据同步的痛点的，这是一个很新颖的方法，我们都应该探索它。

原型的搭建

1. Think It小队将制作许许多多不同的产品原型来传达产品的感官体验。

2. 保真率很低的一个纸面的模型。

3. 高保真的可运行的原型（上面跑伪数据源之类）。

4. 将使用若干内部焦点小组进行识别哪一个原型能最好地传达了它的产品精神（那个故事性描述）。

5. 直到我们不断缩小范围，最后只剩下几个胜出的原型。

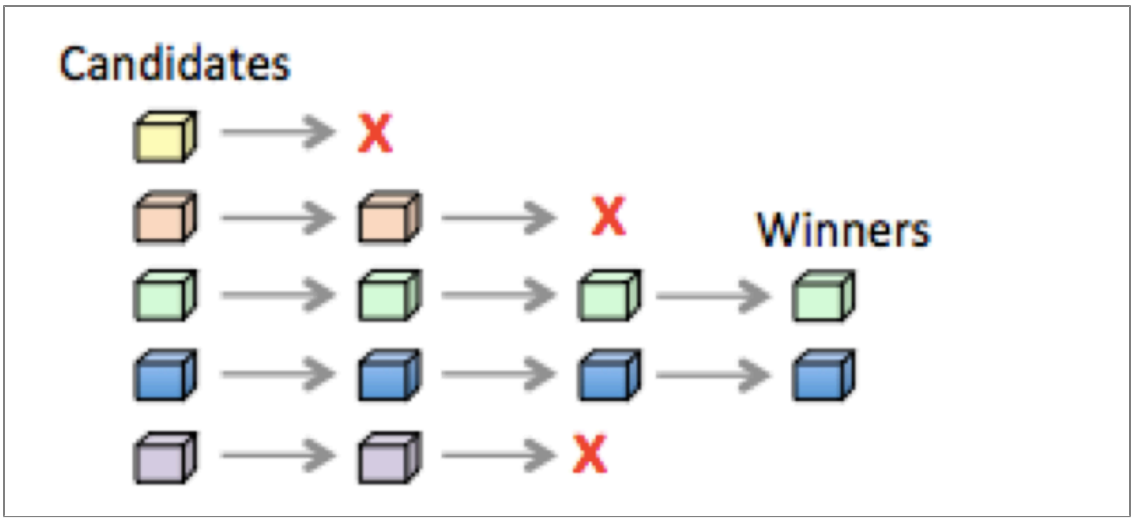


图1-4 故事原形示意图

Think it 阶段何时可以结束

1. 这是一个没有截止日期的迭代过程，我们无法决定这个产品前期会花去多少时间。

2. 当我们能够提出足够吸引人的故事和可以传达它的工作原型时，构建产品是值得的。

阶段之Build it

以下是工作流程

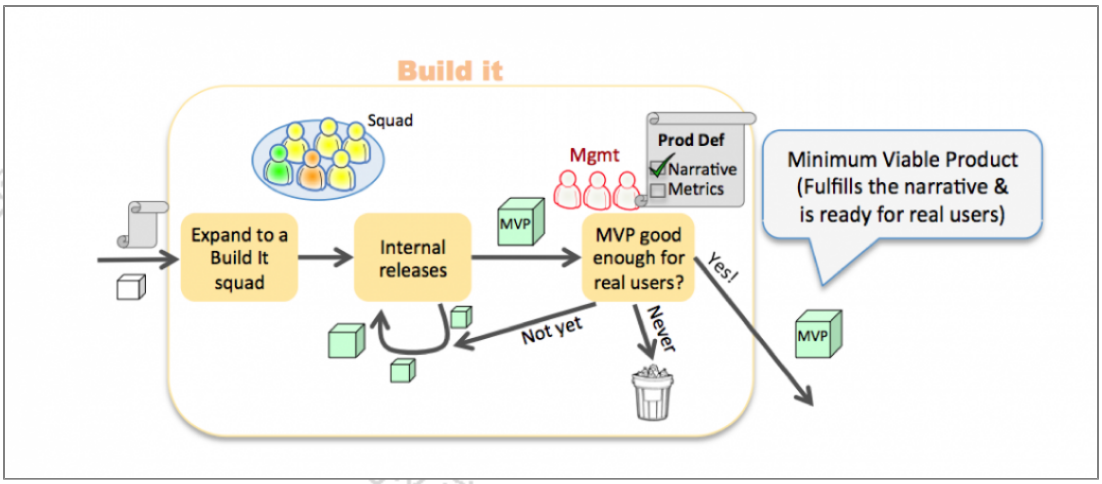



图1-5 Build it阶段示意图


图1-6 Build it 示意图

过程说明

目标：构建出能够向真实用户充分传达产品理念的MVP（最小可行产品）。

过程：

1. 扩建小队。

2. 构建和内部发布。

3. 确认MVP是否足够好。

注意事项：

1.尽量快：不希望在发布产品前构建一个十分完备的产品，因为这个过程会延迟我们获取客户使用数据的时间。

2.不希望产出无用的或令人沮丧的产品。要写产品级的代码并且保障质量。

3.（MVP也可以称为：最早令人喜爱产品。）

完成的定义：

管理者和小队共同认为：

1.目前这个MVP已经实现了基本的故事描述。

2.当前产品已经很好了，可以开始向Release用户发布该产品。

1. 阶段之 Ship it

以下是工作流程

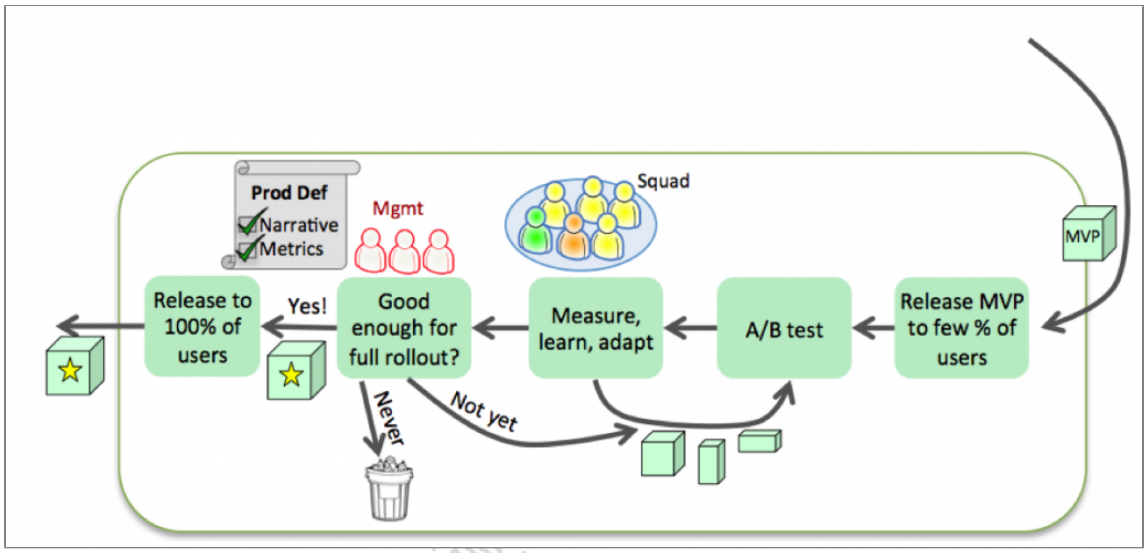



图1-7 Ship it 示意图

过程说明

1) 目标：逐渐将产品扩散到所有用户，同时进行度量和分析，确保产品在真实环境下，也能够达成它的设计初衷。

过程：

1. 小范围发布。

2. A/B测试。

3. 度量和分析、学习、迭代提升。

4. 逐步扩散到所有用户，或者抛弃。

完成的定义：

1. 产品扩散到所有用户。

2. 注意点：

1）这时并不意味着产品已经功能齐全（feature complete），完成了Ship It阶段只是意味着产品（MVP+必要的改进）已经被100%铺开而已。

2）没有全功能声明，因为即使在Ship It阶段之后产品仍将继续优化。

阶段之Tweak It

以下是工作流程

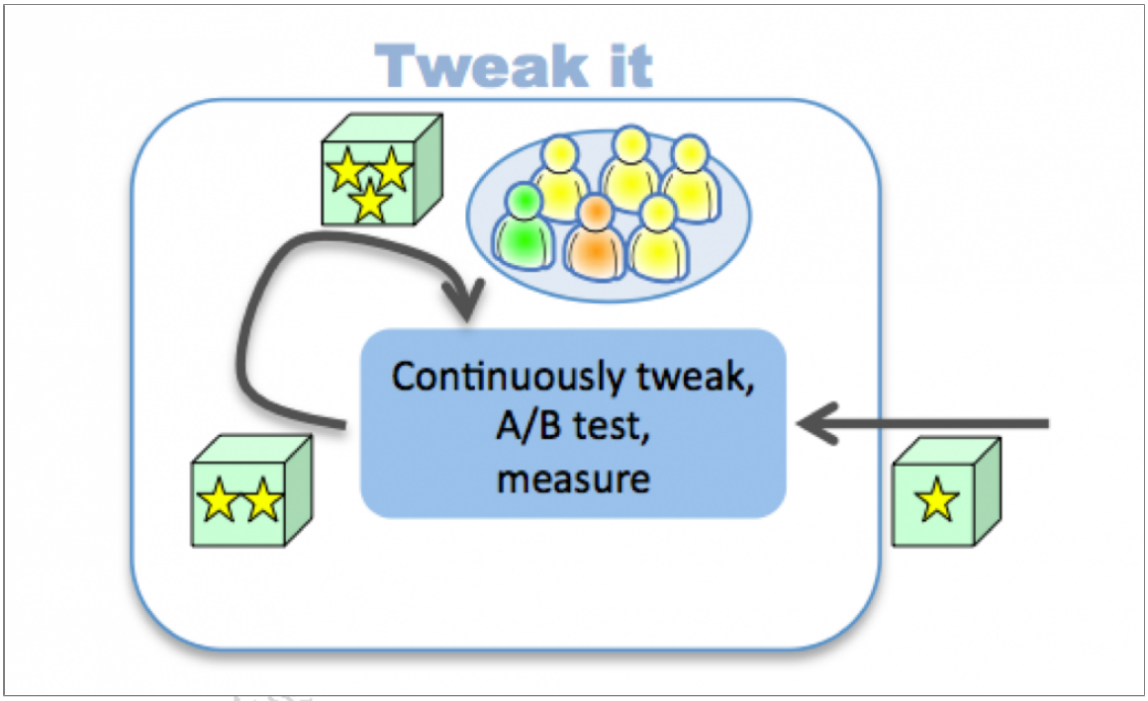



图1-8 Teak it 示意图

过程说明：

1. 在这个阶段，小队持续优化、A/B测试、度量和分析。

2. 直到有一天，所有重要的改进都已经完成，新的改进已经无法带来吸引人的收益，指标数据已经很难有进一步的提升，产品已经趋近于极致。

3. 这时候，小队会逐渐转向新的工作或者重构下一代产品（回到Think it 阶段）

1、 总结

产品开发的各个阶段说明列举如下，四个阶段包含其中：

阶段之Think it：做出一个可以准确传达它的可运行原型和想出一个非常吸引人的故事描述和。

阶段之Build it：构建一个MVP（最小的可行产品），将产品创意完全传达给真实用户。

阶段之Ship it：逐渐将产品扩散到所有用户，同时进行度量和分析，确保产品在真实环境下，也能够达成它的设计初衷。

阶段之Tweak it：在这个阶段，小队持续优化、A/B测试、度量和分析。

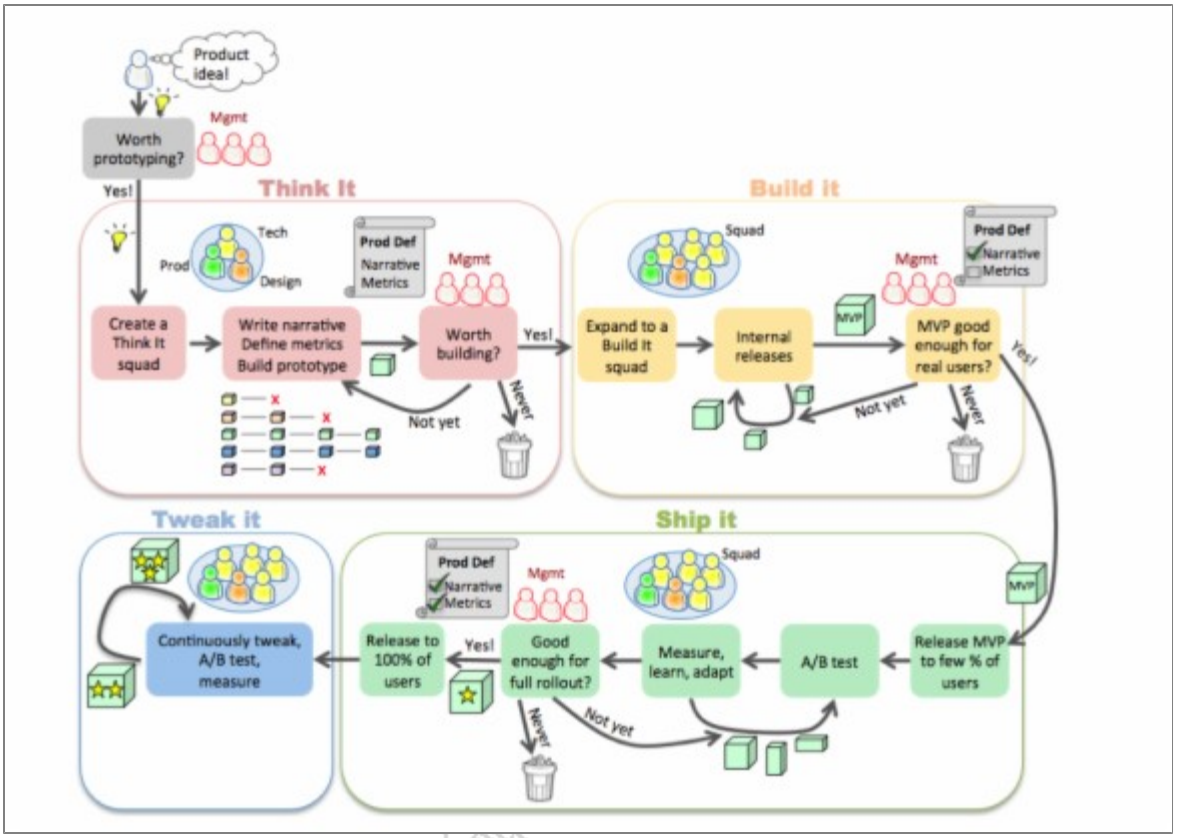



图1-9 阶段总结示意图

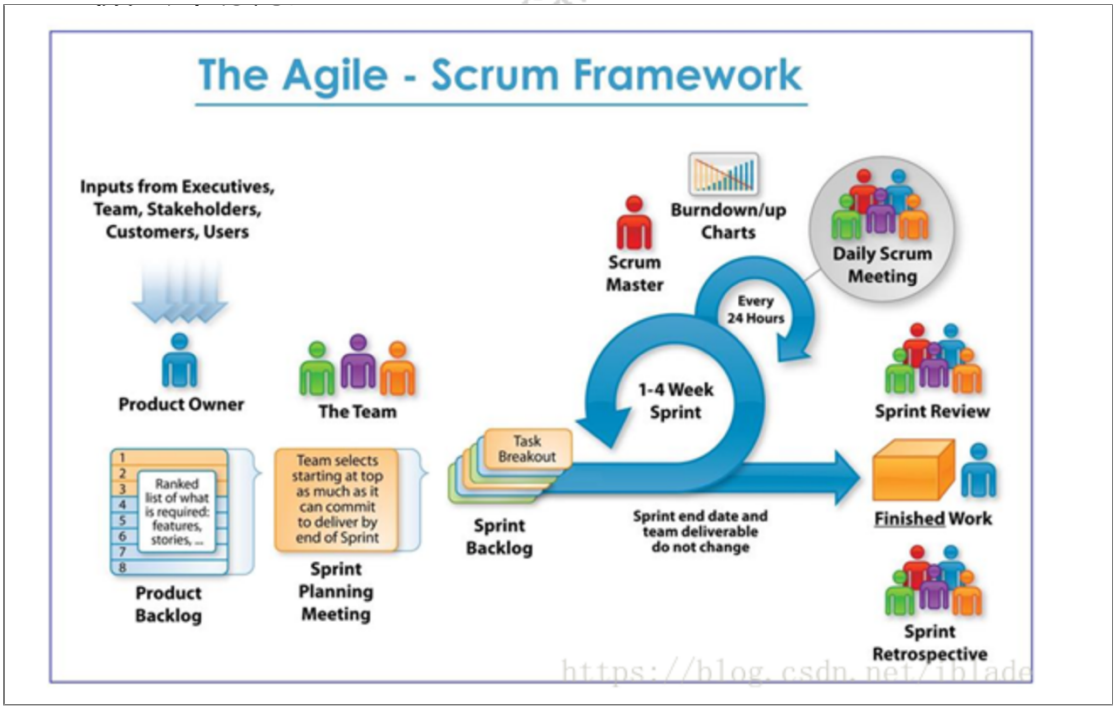



图1-10阶段总结示意图

根据以上的准则，制定出如下工作计划，具体进度安排如下：

%2）1-2周，明确需求、收集开发资料，了解产品的背景，梳理并作出系统系统可行性报告和需求分析报告；

%2）3-5周，系统架构设计、逻辑梳理和程序开发。；

%2）6-7周，整理材料，撰写论文，系统验收；

%2）8-9周，装订论文，编写答辩PPT,评阅、答辩。

1.%2 本章小结

本章详细说明了开发该数据管理系统的必要性和开发方法，论文将主要讨论应用系统软件功能部分的需求、设计和应用状况。从以下两个方便为侧重点。

1. 采用敏捷开发方式，开发以管理公司所有手环数据的管理系统。

2. 模块化系统中的各个子系统、功能，并形成相关文档。

第二章 规划并设计系统

1.%2 系统的初步调查

系统设计规划阶段的第一项活动就是系统的初步调查，初步调查的一个主要目的是掌握系统有那些功能和需求。

进行调研在正式开发该系统之前的必要条件，以下几个方面是其必要性的归结要点：

1、 与公司手环硬件组同事、其他组同事沟通了解，进行需求调研会议，了解系统需求以及需要实现的功能，还有确定项目的开发周期、实施时间的要求。

2、 根据会议讨论提出的需求和时间，进行系统的可行性分析，确认数据系统开发的可行之处。

3、 总的来说，我们必须对系统进行详细的调查，系统的要求必须是明确的，确保所开发系统的功能与会议讨论的要求相一致，避免浪费公司资源，以便在最短的时间内开发出最合适的系统。

1.%2 系统开发的必要性分析

我司手环品类繁杂，搜集、处理、同步数据工作量较大， 为实现数据收集集中化、规范化。提高工作效率。实现数据共享。按照内部会议讨论，结合各部门实际情况，开发基于iOS的智能手环配合系统的设计与实现

1.%2 分析系统开发的可行性

软件系统可行性严谨分析是通过对项目关键设备的选择、用户实际的需求、和系统使用范围的研究，从经济、技术和工程等多的角度对项目进行调查、分析和比较，并在项目完成后取得成果和结果，对科学预计的影响，为项目的最终决策提供公正、可靠以及科学的软件系统开发建议。主要从经济、技术、用户体验等多个方面多维度分析解决方案是不是可行的。只有当解决方案可行且具有一定的经济和/或者是使用价值的时候，才开始开发真正的开发工作。

可行性分析一般可定义为：对项目前期的调查和评价，对拟建项目进行全面、全面的技术经济能力调查，考察是不是可行的。

以建立系统的初步设想，进行系统的可行性研究、技术可行性和经济可行性，完成了可行性研究报告。

如下结论是通过分析本系统得到的：

1、 管理可行性

该系统开发完成后，以系统为中心，只要手环满足系统的数据传输协议，都可以使用该系统。

2、 技术可行性

在公司现有技术下，已有成功上线了类似系统，该技术成熟、稳定、可靠。

3、 经济可行性

用户已经有硬件设备，无须购置其他设备。公司内部是刚需。

综上所述,该系统的开发是切实可行的。

1.%2 系统开发目标

为了避免开发系统不全面,结合公司实际需求，有效管理公司内部手环，提高工作效率。按照规划，利用现代信息技术和网络技术，搭建公司内部的手环管理系统。

1.%2 关键技术研究

本章对本项目涉及到的主要技术进行介绍，包括：iOS系统中的蓝牙开发的技术要点、软件设计模式MVVM的实现，数据库SQLite的设

计、数据同步策略。在此项目中，这些技术将作为应用软件系统的重要开发技术。

1.1.%3 在iOS系统中使用蓝牙开发的技术核心点

蓝牙：一种短程无线通信技术，能够在固定设备、移动设备和构建的个人区域网络（ISM波段2.4-2.485GHz的超高频无线电波）之间进行短程数据交换。

iOS蓝牙框架概述

使用核心蓝牙框架允许iOS和Mac应用程序与蓝牙低能耗设备进行通信，这将变得可能， 例如，利用该蓝牙框架，应用程序可以检测，搜索和与低能量外围设备交互，例如数字恒温器，心率监测器、智能可穿戴设备，甚至其他iOS设备。

该框架是蓝牙4.0规范的抽象，用于低能耗设备。

有两个主要的参与者参与所有蓝牙低能量通信：中央和外围设备。基于某种传统的客户机-服务器体系结构，外设通常具有其他设备所需的数据。中心通常使用外围设备提供的信息来完成某些特定的任务。例如，如图所示，心率监视器可能具有Mac或iOS应用程序可能需要的有用信息，以便以用户友好的方式显示用户的心率。

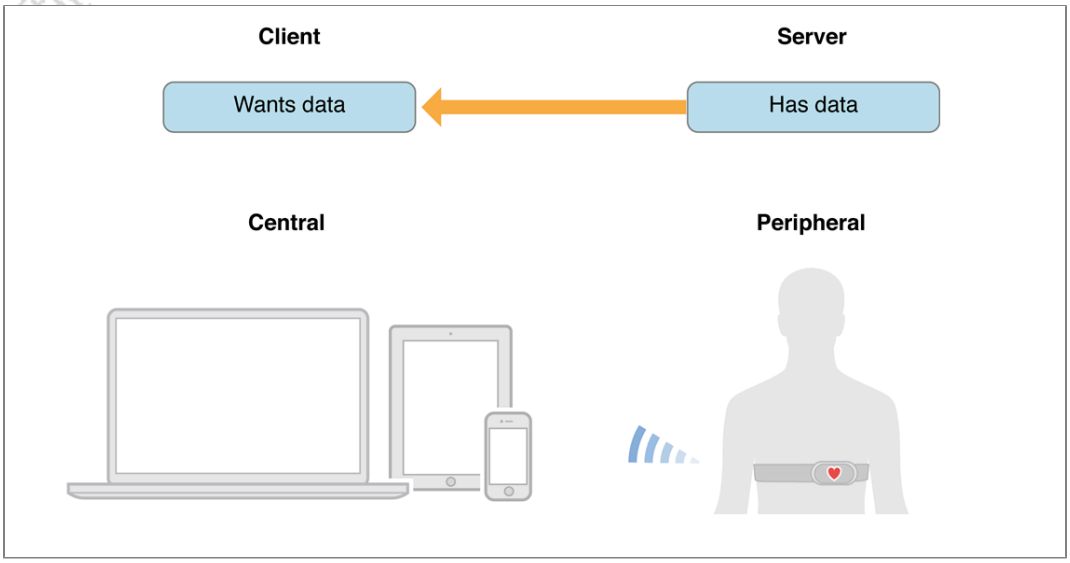



图2-1 蓝牙交互概述示意图

iOS系统中的蓝牙开发框架CoreBluetooth是位于蓝牙低功耗协议栈上。当我们使用它开发时，我们只使用框架CoreBluetooth，通过它我们可以很容易地开发外围设备或中央设备。

。

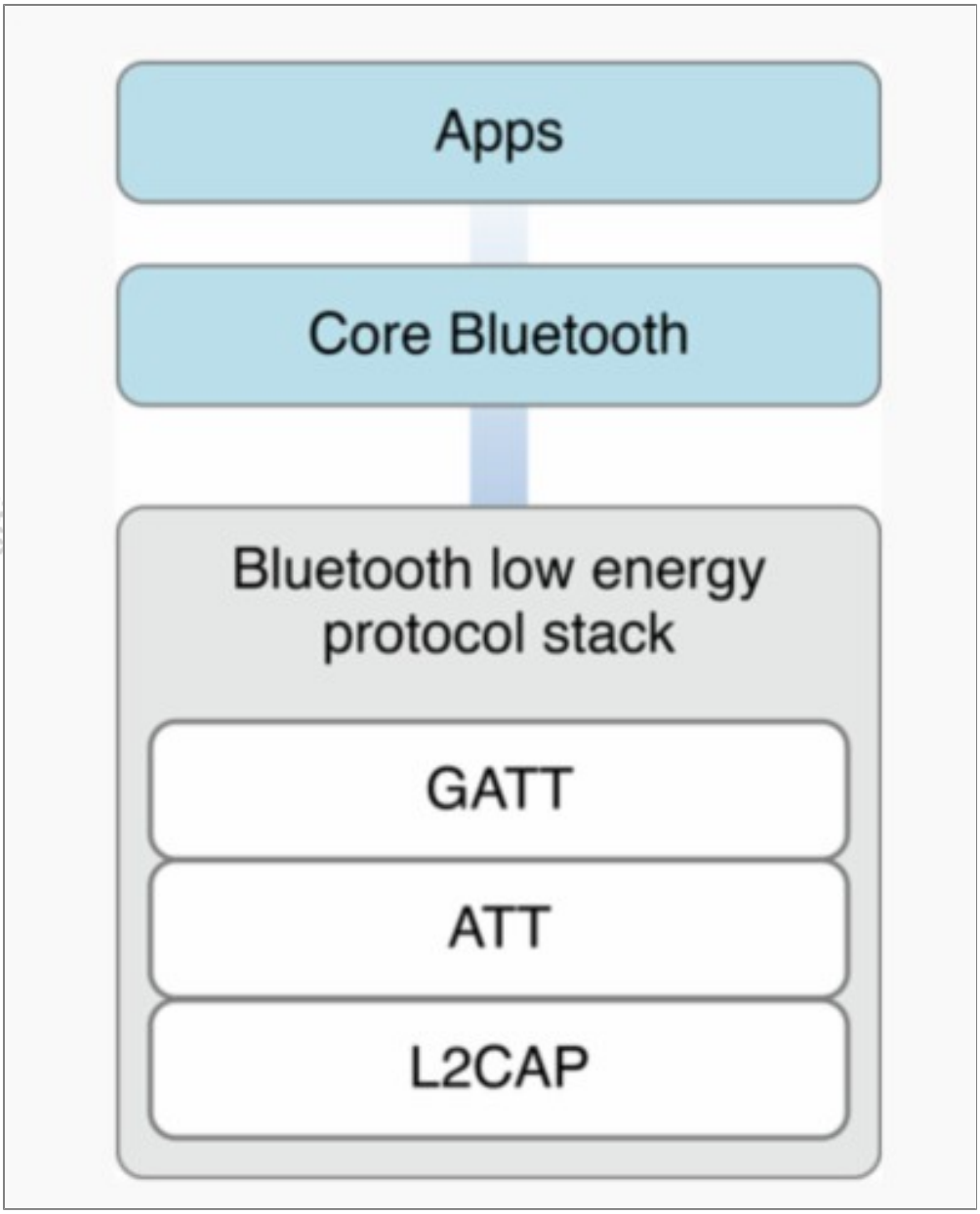



图2-2 CoreBluetooth框架示意图

1.2.%3 基于MVVM设计模式的应用

整个应用架构从上到下是分为三层的，独立于应用的一般层、一般业务层和业务层。业务层用于处理上层业务。业务层可以独立于应用程序依赖于一般业务层和一般层，这种依赖是单向的，自上而下，不能依赖于上层。

大环境软件开发技术的不断提高，也随着前端技术的日新月异的发展，成为应用方案的主流方案是三层结构或多层结构。这不仅有益与开发人员，对整个系统有很多好处，具体有如下几点：

独立的数据处理和接口，以提高界面灵活度。

可承载负载、良率、工作量。

分布式的环境，负载可以共享，可以构成非常强大的计算环境。

组件化可以做准备进行业务模块之间的解耦

本系统采用的是MVVM框架结构。 使用浏览器/服务器体系结构和三层体系结构，系统的整体框架如下图所式。

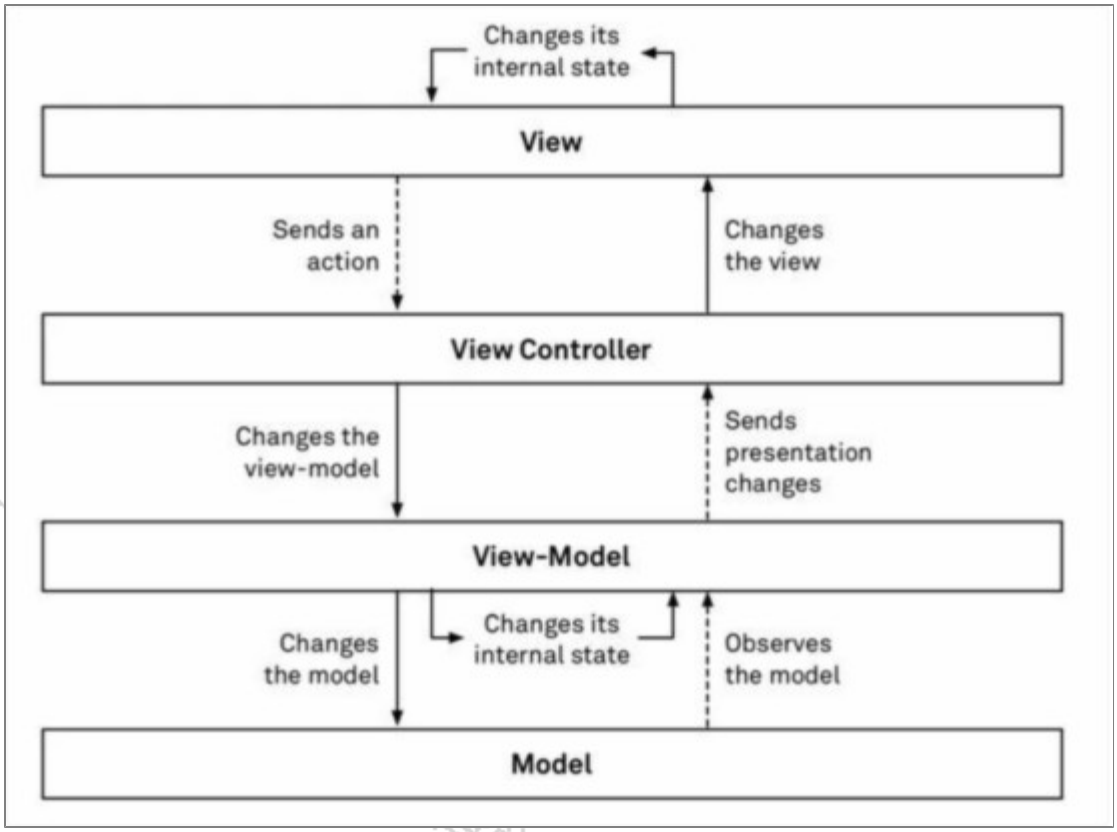



图2-3 MVVM架构示意图

1.1.1.1 处理UI的层

UI层是这三层体系结构中最复杂的层，包括但不限于界面布局、屏幕适应、文本大小、图像资源、用户事件处理、提示信息和动画等。UI层也是变化最频繁的一个层面，每天改得最多的就是UI了。因此，UI层也是最容易变得混乱不堪的一个层。一个良好的UI层，应该有较好的可读性、健壮性、维护性、扩展性。

UI层三原则：

1. 维护的规范性：定义规范开发、包括规范编写、规范常用的逻辑处理机制、规范命名、规范注释等，并严格按照规范执行；

2. 保持统一性：布局只做布局的事情，内容只做内容的事情，分开每个要做的事情，做到每个方法，每个类，只有一个东西；

3. 保持简单性：保持代码和结构简洁。 不要在每个方法，每个类，每个文件中放入太多的代码或资源。如果有多个部分，则需要将其拆分。

1.1.1.1 处理业务的层

如果业务层向下处理逻辑的，那它就是和与数据层产生交互；如果业务层向上处理逻辑的，那么与产生交互的是UI层。

与数据层的交互仅调用数据层的接口以获取数据，并且与表示层的交互需要提供到表示层的接口。 因为服务处理通常是相对耗时的操作，主要是因为底层网络请求是耗时的，所以应该以异步方式提供提供给表示层的接口数据。 因此，接口需要提供一个回调参数来返回服务。 处理Server端返回的结果。 业务层就是处理核心业务逻辑的数据处理场。 实际上，只要清楚了业务层的职责，业务层的具体实现就不难了。

1.1.1.1 处理数据的层

数据层是三层体系结构中的最低的层，它是负责数据管理工作的。以下任务是其主要任务：

调用后端提供网络API请求数据；根据应用逻辑可在本地缓存数据；在将数据传递到上一层，也就是UI层

依据这三个划分的任务，数据层还可以把粒度放小，具体可以分为三层：网络请求层，本地缓存数据层和数据的传递层。

1.3.%3 设计数据库

结合本系统的实际使用场景，采用SQLite数据库数据处理方案，搭建数据库，确定数据库各表字段名及其代表含义。根据设备端的功能需求以及结合 Server 端数据存储的格式，使移动移动 APP 端在数据库设计上存在一致性以及稳定性。

1.1.1.1 数据库命名

数据库以用户ID命名，没有用户用户一个单独的数据库

1.1.1.1 数据库中的主键

因为项目定义是以用户为维度，设备是从属关系，所以除了信息表，每张表使用userId、macAddress或timestamp作为联合主键

1.4.%3 数据同步逻辑

根据应用的实际需求自定义数据同步策略，依据不同功能模块的特性，设置不同 的数据同步策略，同时也能够有效的控制用户流量开销

1.1.1.1 登录成功后

首次登录，卸载重装登录，没有数据库

概要数据:概要数据以天为单位统计的各项数据，从云端拉取属于登录用户的所有数据， 失败需尝试再次拉取。

明细数据:明细数据以分钟为单位的各项数据，考虑数据量较大，先拉取小部分时间内 的数据，从当前时间起往前推一个星期七天内的数据，剩下的数据，UI 操作到那个时间的 数据再拉回。

设备列表:登录成功后拉取，失败不再尝试拉取，APP 手动添加新的手环设备。

退出再登录，数据库里面有数据

概要数据:数据库最新数据记录时间，拉取该时间到今天为止的数据。

明细数据:原则上拉取数据库最新时间到今天为止的数据，如果时间大于七天则拉取七 天，剩余的 UI 操作时策略拉回。

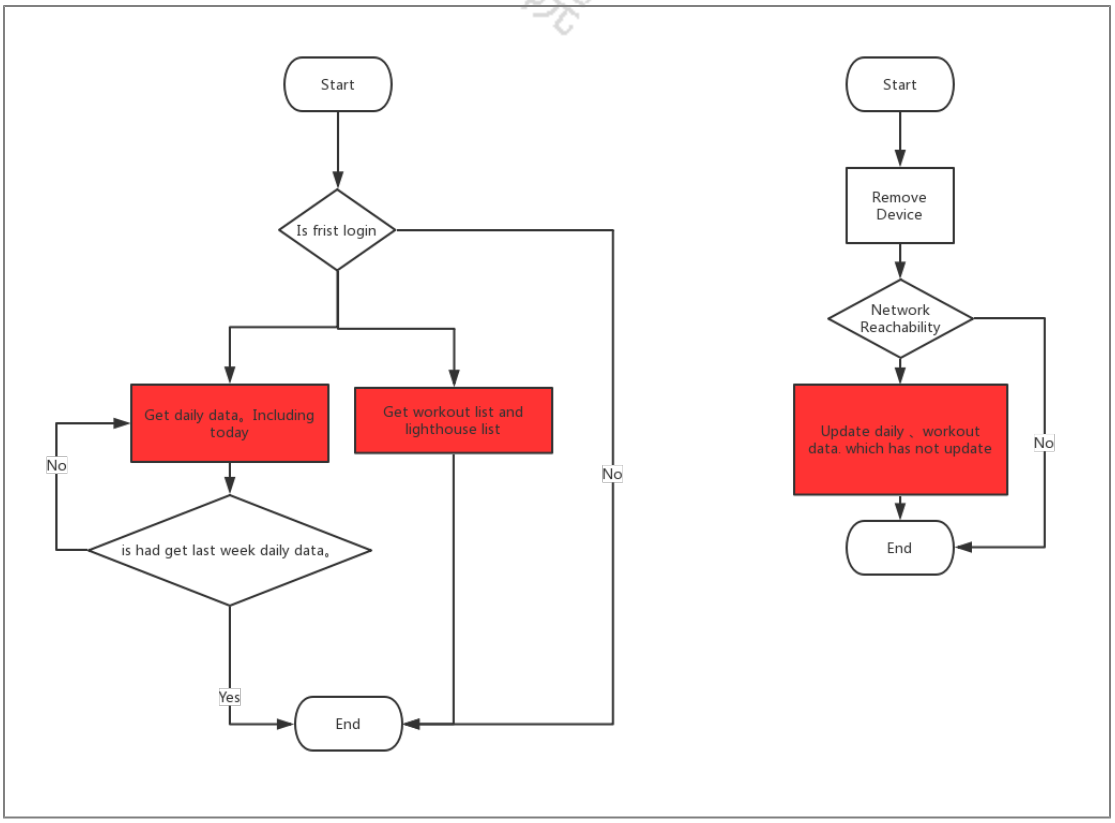



图2-4 登录拉取数据流程图

1.1.1.1 登出

登出需提交未提交、待提交的数据

1.1.1.1 同步

APP 与手环同步数据后，上传当天的概要和明细，按标识上传，上传过的不再上传，明 细距离上一次同步上传需要间隔大于三分钟上传，概要间隔大于三分钟策略上传。需注意每。。当 APP 与设备同步后，需汇总今天明细数据为今天的概要，并且同时，需要查询没汇总的数 据，汇总并上传。其中睡眠的明细数据支持片段上传策略，上传以天为单位的明细数据。

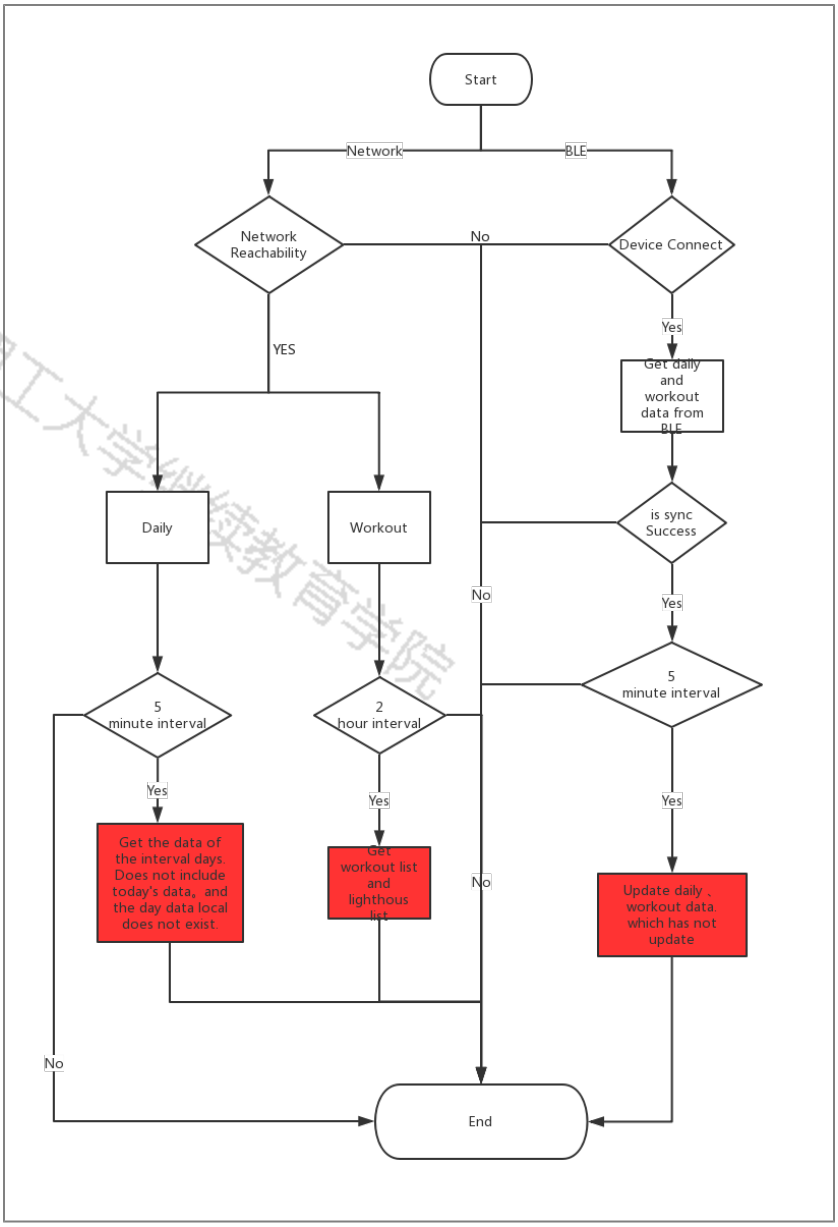





图2-5同步数据流程图

1.1.1.1 后台数据更新

满足三分钟半小时策略情况下，提交数据。

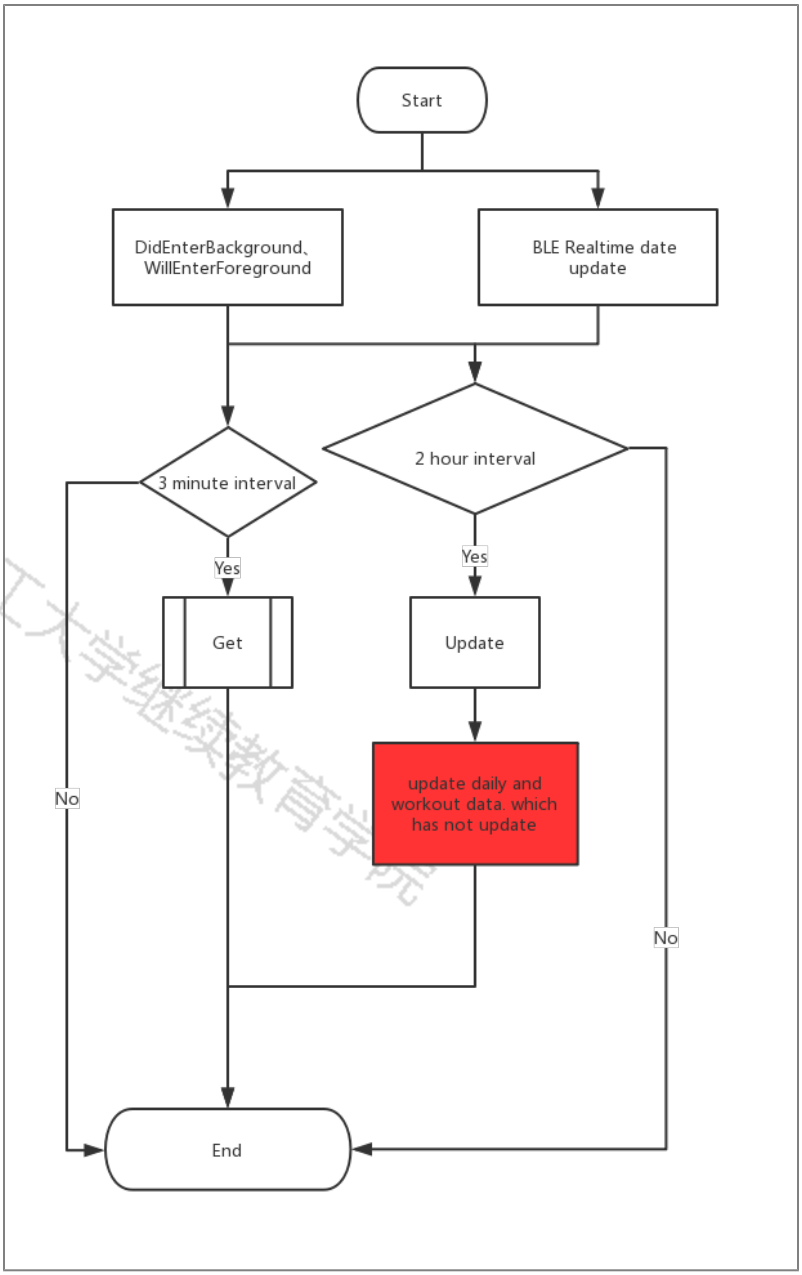





图2-6 后台数据更新流程图

1.1.1.1 个人信息修改项

修改项优先保存保证本地数据库，然后提交到云端，遇修改提交失败， 可依据策略再次提交，可加入三分钟策略扫描再次提交，如有再次下拉用户的信息的操作，那么 需考虑本地标识是否有修改未提交，以本地高优先，本地有修改待提交，那么不能覆盖，以 本地为最新数据。因身高体重值跟运动数据等紧密相关，优先考虑用户信息与手环同步。

1.%2 本章小结

本章详细介绍了基于iOS的智能手环配合系统的设计与实现开发的前期准备工作和系统可行性的分析工作，然后介绍了系统的要达到的开发目标，最后还详细的分析了关键技术的研究。

第三章 系统需求的分析

1.%2 引言

1.1.%3 编写系统需求目的

本系统的需求分析是主要是建立在项目组调研的会议纪要上，从而编写的，它详细的描述了基于iOS的智能手环配合系统的设计与实

现的需求情况，通过用户使用需求确定了系统的使用范围和大致的用户量，在这同时也有助与所有与该项目的相关人员，帮助参与项目所有相关人员充分理解该管理系统，

1.2.%3 系统需求参考文档

编写本报告时所参考的文档资料，还有开发此系统软件时用到的软件开发规范。

GB8566-88 《计算机软件开发规范》

App Developer Documentation 点击查看

App Human Interface Guidelines 点击查看文档

1.%2 目标概述

1.1.%3 管理系统目标

收集公司内所有手环型号生成的数据，系统进行数据处理后缓存本地并同步到云端，用户可在系统中查看其所拥有手环的所有数据。

1.%2 系统性能需求

1.1.%3 安全性和稳定性

数据是企业的重要财产，它的安全影响到系统的正常运作。因此，系统必须要有一套有效的安全机制，保证系统级数据的安全。

在系统的安全性管理方面，本地数据有做加密处理。使用Https与Server数据同步，对数据都有做加密处理。

本系统做了各种逻辑、数据异常的容错处理，系统稳定性高。

1.2.%3 高效性和易操作性

本系统采用的是MVVM架构编程技术，使系统在响应时间，更新处理时间，数据传输、转换时间，计算时间等方面具有较高的性能。

1.3.%3 扩充性和前瞻性

因为未来业务的不断发展，很好的扩展性是系统必须要具备的，因为系统是在不断的完善中的。同时系统的并不是仅仅满足当前的需求，它还要适应于未来业务的不断发展。所以系统会有很长的生命周期。

1.%2 需求分析

根据实际业务操作流程，并结合系统设计，系统功能分为今天、锻炼、历史、我的这四个部分。今天的页面如下，如下图3-1

1.1.%3 今天

图3-1 今天页面效果图

简要说明

1. 实时查看当前连接手环的数据，以及今天的详情数据。

2. 查看当前手环的连接状态。

事件流

1. 界面实时显示手环数据。

2. 依据数据同步策略，将数据库的数据上传云端。

数据流

1. 在已连接的情况下，手环数据才能实时显示。

2. 接收手环的数据，处理数据，缓存数据。

3. 在个人信息设置中，用户更改的运动目标，该页面数据都要重算、刷新显示。

1.2.%3 锻炼


图3-2 锻炼页面效果图

简要说明

1. 用户可查看历史锻炼数据，能设置单次的锻炼目标。可以手动开始结束Workout。

事件流

1. 用户点击目标（默认运动距离是5千米）按钮，可设置单次的运动目标，也可设置运动时间，可设置的运动时间范围是5分钟到2小时。

2. 用户可以点击开始按钮，开始Workout,此时页面会实时显示运动数据，结束workout后，页面会显示最近的一次运动数据。

3. 用户点击页面的任意一个Cell， 可查看所有的运动记录，以月为单位汇总显示。

数据流

1. 开始一个Workout必须是在设备已连接的情况下进行，且设备数据生成完成后，才能从设备端请求数据。

1.3.%3 历史


图3-3 历史页面效果图

简要说明

1. 用户可以查看所有记录。运动数据（包括步数、距离、卡路里、睡眠），时间线为查询基线， 可按日、月、年查询。心率、血糖只能以天为单位查询。其中有日历功能，可查询指定天的数据。

2. 查看已连接过手环的状态、

事件流

1. 点击顶部的标签可以按日、月、年为单位查询数据。

2. 左右滑动可查询已选择的时间查询单位的下一页、上一页数据。

3. 点击下方Cell以项目（步数、距离、卡路里）为单位，在当前选择的时间单位下查询数据。

4. 点击右上角日历图标，调出日历页面，可查询指定天的数据。

数据流

1. 有历史数据才显示，无历史数据、没有到达的日期都不显示。

1.4.%3 我的

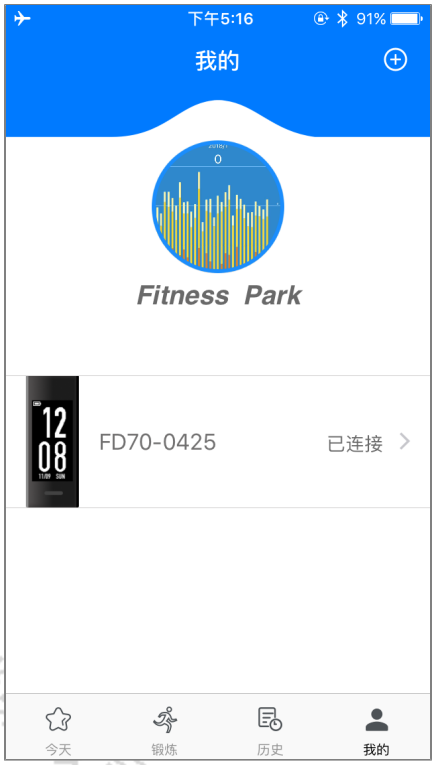





图3-4 我的页面效果图

简要说明

1. 可查看、修改用户信息，

2. 可连接设备，查看当前连接的设备，可对设置进行偏好设置，并检测手环固件版本，对手环进行OTA（Over－the－Air Technology）。

事件流

1. 点击右上角添加设备按钮，可搜索附件设备并连接。

2. 用户头像可修改用户信息等操作。

3. 如果有已连接的设备，可修改设备信息等操作。

数据流

1. 修改用户信息，在没有网络无法同步到Server的情况下，先将数据保存到本地，有网络的情况下，再将数据上传。

1.%2 系统开发环境

1.1.%3 开发平台

Mac OS操作系统简介：

Mac OS是在Apple的Macintosh系列计算机上运行的一种操作系统。 在商业领域第一个成功的图形用户界面的操作系统是Mac OS。版本号macOS 10.14.4是当前最新的系统，并且网络上还有各种各样可以运行的Mac系统，称为Mac PC。

Mac系统是基于Unix内核的图形操作系统;它是一般不能安装在普通PC上的操作系统。由Apple自己开发。 Mac的操作系统已经到达OS 10，代号为MAC OS X，这是自MAC计算机诞生以来15年来最大的变化。它的许多服务和各种各样的功能都体现了Apple的理念。新系统非常可靠，非常地受用户喜爱。

与此同时，在被病毒感染方面，各种各样的的计算机病毒大部分都是针对Windows系统的。因为MAC系统的体系结构与Windows系统的体系不同，所有受到病毒的攻击是非常少的。 操作系统界面非常独特MAC OSX是一大特点，它着重突出了图标和人机对话。 Apple它不仅开发了专属于自己的系统，它还擅长硬件的开发。

在2011年的7月20日，Apple在WWDC （苹果开发者大会）上， 苹果将OS X变为了Mac OS X的新名字了.V 10.14.2是最新的版本号，它发布于2018年的9月25日凌晨。

在2018年的3月30日，Apple推出了macOS High Sierra 10.13.4的官方版本。新版本增强了对外部eGPU的支持，并添加了iMac Pro的原始墨水云壁纸。

在2018年的9月25日，苹果更新了macOS 系统，新系统的版本号是Mojave 10.14，深色模式是它的特点，Safari 浏览器也到了更新，Mac App Store ，访达，桌面，股市，语音备忘录，家庭App等。

Xcode

Xcode是在Apple Inc.开发的操作系统Mac OS X上运行的集成开发工具（IDE）.Xcode是开发macOS和iOS应用程序的最快方式。 Xcode具有统一的用户界面设计，编码，测试，调试都在一个简洁的窗口中完成。

Xcode包含开发人员为iPhone ，Mac，Apple TV，iPad，和Apple Watch创建出色应用所需的一切。 Xcode为开发人员提供统一的编码、测试用户界面设计和Debug工作流程。 Xcode还集合了Swift开发环境，这使开发出来应用程序比以往开发的应用响应更快速、体积更小。

Swift

苹果公司在2014年WWDC（Apple开发者大会）上发布的新开发语言Swift与Mac OS和iOS平台上的Objective-C *协同工作，构建基于Apple平台的应用程序。

Swift是一种很好掌握的编程语言，它具有与脚本语言相同的表现力和乐趣性，并且是具有这一特性的第一种系统编程语言，安全性是Swift的设计考虑的第一点，它有避免各种常见的编程错误的能力，这会让开发人员更容易把代码写对，而不容易把代码写错

Swift的发展已经存在了很长时间。为了为Swift奠定基础，Apple改进了编译器，调试器和框架。使用了自动引用计数（ARC）来简化内存管理，这相对于手动引用计数（MRC）来说进步很多。我们基于Foundation和Cocoa构建和标准化框架堆栈。 Objective-C本身支持块，集合语法和模块，因此框架可以轻松支持现代编程语言技术。由于这些基础工作，我们现在可以为将来的Apple软件开发发布这样一种新语言。

SQLite数据库

SQLite是一个非常轻量级的数据库，是一个符合ACID的关系数据库管理系统，一个相对较小的C库包含其中。 这是由D. Richard Hipp建立的公共领域项目。 嵌入式是其的开始设计目标，并且已经在许多嵌入式产品中使用。 它占用的资源非常少。它可能只需要几百千字节的内存，在嵌入式设备中。 它可以支持Linux / Windows / Unix等主流操作系统，并且可以与许多编程语言结合，如PHP，C#，Tcl， Java等等各种编程语言，以及ODBC接口，也比较Mysql，PostgreSQL两个来的开放些 来源在世界着名的数据库管理系统的情况下，SQLite的处理速度是最快的。 在2000年5月SQLite的第一个Alpha版本诞生了。SQLite发布的SQLite 3版布，自2015年以来已经有15年了。

SQLite工作原理：与通常的客户端 - 服务器范例不同，SQLite引擎不是与程序通信的单独进程，而是作为程序的主要部分连接到程序。 因此主要的通信协议是编程语言中直接API调用就可以了。 这产生的积极影响有延迟时间、总消耗和整体简单性。 整个数据库（定义，表，索引和数据本身）存储在单个文件中的单个主机上。 启动事务时锁定整个数据文件，是它的简单设计要点。

1.2.%3 限制的条件

1. 本系统必须使用公司生产的手环进行连接，同步数据。
2. 手环与APP数据的数据通信依照数据同步协议文档进行。
3. 数据同步服务器的对接按照数据同步文档进行 。
4. 系统运行在iOS系统8及以上版本。

1.3.%3 开发条件依赖

1. 软件框架和软件技术确定后在项目中有效，并且不会轻易作变更。
2. 流程管理严格管理需求变更计划。

3. 项目开发进度不会被需求的变更影响到。

4. UI变更需要按照UI变更管理流程执行。

1.4.%3 需求优先级

| 优先级 | 功能需求/使用用例 |
|---|---|
| 高 | iOS8及以上版本可正常安装、使用本系统。 |
| | 数据同步至网络。 |
| | 手环数据实时显示。 |
| 中 | 不影响功能的情况下，系统性能稍有欠缺。 |
| | 允许数据同步缓慢、稍微卡顿。 |
| 低 | 各模块的UI优化， |
| | 部分同步策略的不完善。 |

对高、中、低的解释如下：

| 优先级 | 解释 |
|---|---|
| 高 | 项目中要实现的关键功能。 |
| 中 | 为实现项目目标需要的一般功能，一些性能的优化。 |
| 低 | 属于优化性质的功能、UI提高。 |



1.%2 本章小结

本章中详细介绍了基于iOS的智能手环配合系统的设计与实现的功能需求，并分析梳理了各个功能模块的功能点，系统的开发环境也作了较为详细介绍了。最后介绍了需求完成的优先级。

第四章 系统详细设计与实现

1.%2 简介

系统功能分为今天、锻炼、历史、我的四个部分。今天、锻炼、历史侧重数据的显示，其中所有图表显示的目标值在 我的模块中设置，也就是说目标值是数据显示的核心要素。

1.%2 系统设计

1.1.%3 系统软件架构设计

应用程序开发中普遍存在的不良做法是开始构建应用程序而不遵循任何应用程序架构模式。虽然它们并非绝对需要，但基于架构设计模式的开发可以帮助我们为应用程序构建真正坚实的基础。

。

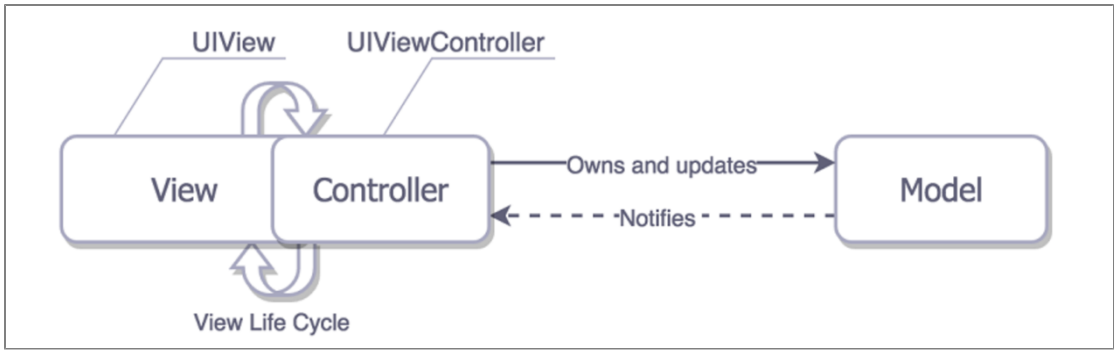



图4-1 系统软件架构示意图

忽略体系结构模式还会带来集群、杂乱的代码和组织不严密的模块。这违背了我们通常遵循的关注点分离原则。

不遵循架构设计模式的应用程序紧密耦合，难以更改，并且更容易出错。

通过本文，我们将对其作较为详细的介绍。

1.2.%3 使用MVVM在iOS的使用

MVC的改进版本称为MVVM设计模式。在MVVM中，视图是由视图和控制器结合在一起组成的，另外还引入了ViewModel，ViewModel现在处理表示逻辑。

在Model中引入了ViewModel，View绑定了ViewModel,ViewModel更改并按事件逻辑更新Model。此外，因为ViewModel与Model进行了强绑定，这样有便于Model和ViewModel的通信。

这种分离使得ViewControllers非常地简洁，由于更明确的关注点分离，它引入了更多的可读性和可维护性。

MVVM中的数据绑定。MVVM中出现的一个常见问题是，如果ViewModel和View都不包含任何引用，那么UI如何在MVVM中更新？

这个问题的答案是数据绑定，它将UI元素连接到ViewModel的可观察属性。虽然默认情况下iOS中的MVVM不支持数据绑定，但是有很多第三方库有支持，如RXSwift.

1.3.%3 系统的总体结构


图4-2 系统总体结构图

1.%2 数据库结构设计

1.1.%3 数据库中的表结构

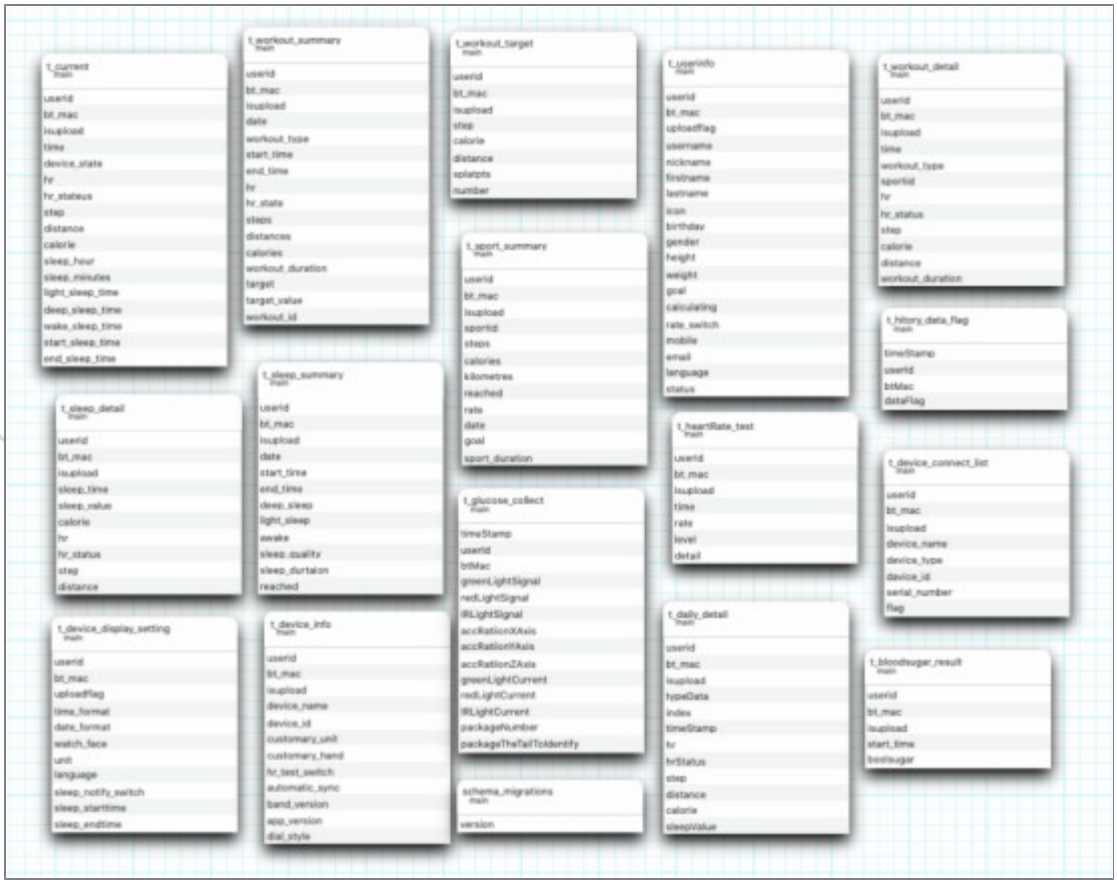


图4-3 数据库表示意图

1.2.%3 子模块Workout历史功能实现示例

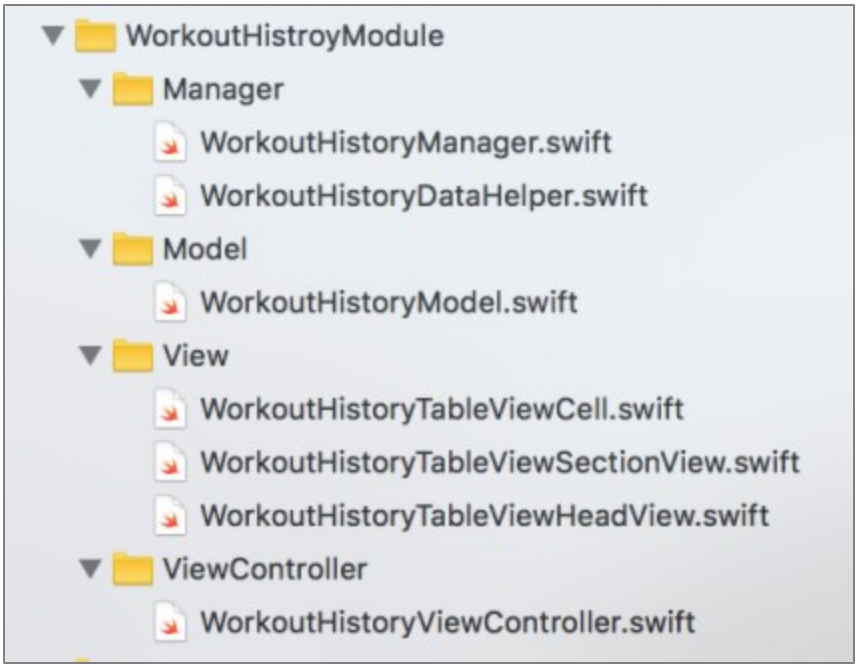



图4-4 Workout文件目录图

功能模块内分为Manger、View、ViewController层

Manger层为数据处理层

```
mport UIKit
import SQLite
class WorkoutHistoryDataHelper: DataHelperProtocol {

static let TABLE_NAME = "t_workout_summary"

static let userid = Expression<String>("userid")
static let bt_mac = Expression<String>("bt_mac")
static let isupload = Expression<String>("isupload")

static let workout_id = Expression<String>("workout_id")
static let workout_type = Expression<String>("workout_type")
static let start_time = Expression<String>("start_time")
static let end_time = Expression<String>("end_time")

static let hr = Expression<String>("hr")
static let hr_state = Expression<String>("hr_state")
static let steps = Expression<String>("steps")
static let distances = Expression<String>("distances")
static let calories = Expression<String>("calories")

static let workout_duration = Expression<String>("workout_duration")
static let target = Expression<String>("target")
static let target_value = Expression<String>("target_value")
static let date = Expression<String>("date")

static let table = Table(TABLE_NAME)

typealias T = WorkoutSummaryModel

//协议方法
static func createTable() throws {
guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}
do {
_ = try connection.run( table.create(ifNotExists: true) {t in
t.column(userid)
```

```
t.column(bt_mac)
t.column(isupload)

t.column(date)
t.column(workout_type)
t.column(start_time)
t.column(end_time)

t.column(hr)
t.column(hr_state)
t.column(steps)
t.column(distances)
t.column(calories)

t.column(workout_duration)
t.column(target)
t.column(target_value)
t.column(workout_id)

t.primaryKey(userid, bt_mac, start_time)
})
} catch _ {
throw DataAccessError.datastoreConnectionError
}
}
}
// MARK: - 自定义方法
extension WorkoutHistoryDataHelper {

static func insertOrUpdate(items: [T], complete:() -> Void) throws {
guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}

try connection.transaction(.deferred, block: {
for (index, item) in items.enumerated() {
if try connection.run(table.filter(item.userid == userid).filter(item.bt_mac == bt_mac).filter(item.date ==
date).update(item)) > 0 {
print("updated")
```

```
} else {
let _ = try connection.run(table.insert(item))
print("insert")
}

if index == items.count - 1 {
complete()
}
}
})

}


static func update(item: T) throws -> Bool {

guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}
let alice = table.filter(item.userid == userid).filter(item.bt_mac == bt_mac).filter(item.date == date)
if try connection.run(alice.update(item)) > 0 {
return true
} else {
return false
}

}

static func find(userID: String) throws -> [T] {
guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}

let items = try connection.prepare(table.filter(userid == userID))
var retArray = [T]()
for item in items {
retArray.append(T.init(userid: item[userid],
bt_mac: item[bt_mac],
isupload: item[isupload],
```

```
workout_id: item[workout_id],
workout_type: item[workout_type],
start_time: item[start_time],
end_time: item[end_time],
hr: item[hr],
hr_state: item[hr_state],
steps: item[steps],
distances: item[distances],
calories: item[calories],
workout_duration: item[workout_duration],
target: item[target],
target_value: item[target_value],
date: item[date]))
}

return retArray

}

static func findNotUpdateData() throws -> [T]? {

guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}

let items = try connection.prepare(table.filter(isupload == "0"))
var retArray = [T]()
for item in items {
retArray.append(T.init(userid: item[userid],
bt_mac: item[bt_mac],
isupload: item[isupload],
workout_id: item[workout_id],
workout_type: item[workout_type],
start_time: item[start_time],
end_time: item[end_time],
hr: item[hr],
hr_state: item[hr_state],
steps: item[steps],
distances: item[distances],
```

```
calories: item[calories],
workout_duration: item[workout_duration],
target: item[target],
target_value: item[target_value],
date: item[date]))
}
if retArray.isEmpty {
return nil
}else {
return retArray
}
}

static func FindLatestWorkoutTarget(userID: String, macAddress: String) throws -> T {
guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}


let itemOptional = try connection.pluck(table.filter(userID == userid).filter(macAddress ==
bt_mac).order(date.desc))

guard let item = itemOptional else {
var defaultModel = WorkoutSummaryModel()
//0 距离 1 时长
defaultModel.target = "0"
defaultModel.target_value = "5"

defaultModel.userid = userID
defaultModel.bt_mac = macAddress

try! insertOrUpdate(items: [defaultModel], complete: {})

return defaultModel
}

return T.init(userid: item[userid],
bt_mac: item[bt_mac],
isupload: item[isupload],
```

```
workout_id: item[workout_id],
workout_type: item[workout_type],
start_time: item[start_time],
end_time: item[end_time],
hr: item[hr],
hr_state: item[hr_state],
steps: item[steps],
distances: item[distances],
calories: item[calories],
workout_duration: item[workout_duration],
target: item[target],
target_value: item[target_value],
date: item[date])

}

static func find(_ userId: String, _ macAddress: String) throws -> [WorkoutHistoryModel] {

guard let connection = SQLiteDataStore.sharedInstance!.connection else {
throw DataAccessError.datastoreConnectionError
}

let queryMonthDataSQLiteStr = "select sum(distances) as distanceTotal,
strftime('%Y/%m',start_time,'unixepoch','localtime')as 'yearMonth' from t_workout_summary group by yearMonth"

var array = [String: String]()

let stm = try connection.prepare(queryMonthDataSQLiteStr)

for row in stm {
var resultDic = [String: String]()
for (index, name) in stm.columnNames.enumerated() {

if let value = row[index] as? String {
resultDic[name] = value
}else if let value = row[index] as? Double {
resultDic[name] = value.description
}else if let value = row[index] as? Int64 {
resultDic[name] = value.description
```

```
}
}
array.append(resultDic)
}



var dataSource = [WorkoutHistoryModel]()
for resultDict in array {
let distanceTotal = resultDict["distanceTotal"]

let yearMonth = resultDict["yearMonth"] ?? ""

var historyModel2 = WorkoutHistoryModel()

var sumModel2 = WorkoutHistorySummaryModel()
sumModel2.monthDate = yearMonth
sumModel2.summaryDistance = distanceTotal ?? ""

var detailModels = [WorkoutHistoryDetailModel]()



print("+++++++++\(yearMonth)")
let queryDayDataSQLiteStr = "select * from t_workout_summary where strftime('%Y/%m',
start_time,'unixepoch','localtime') = '\(yearMonth)'"

let stm = try connection.prepare(queryDayDataSQLiteStr)

for row in stm {

for (index, name) in stm.columnNames.enumerated() {
var detailModel5 = WorkoutHistoryDetailModel()
if name == "workout_type" {
detailModel5.type = row[index] as! String
}else if name == "distances" {

detailModel5.distance = row[index] as! String
}else if name == "workout_duration" {
```

```
detailModel5.schedule = row[index] as! String
}else if name == "steps" {
detailModel5.step = row[index] as! String

}else if name == "calories" {
detailModel5.calorie = row[index] as! String

}else if name == "date" {

detailModel5.dayDate = row[index] as! String
}

detailModels.append(detailModel5)
}
}



historyModel2.summaryModel = sumModel2
historyModel2.detailModels = detailModels

dataSource.append(historyModel2)
}

return dataSource

}

}
```

1.%2 本章小结

本章梳理了系统的开发架构，列举了部分功能的详细实现方法。

第五章 结论

1.%2 全文总结

这次毕业设计让我收获到很多的知识，它使我能够把这几年来所学的软件开发知识进行综合地应用与实践，而且还是和实际的项目结合起来的。经过三个多月的构思与开发，特别是在毕业指导老师的精心指导和安排下，我的基于iOS配合手环系统的设计与实现已经设计完毕。

其功能符合用户的实际需求，具有很好适用性，但是由于时间仓促，加上本人软件设计经验的不足。因此，在分析问题、解决问题和程序设计中的某些细节把握地还不够好，这还需要在以后的版本迭代工作中不断地改进和提高。

做完毕业设计后，使我真切的认识到软件开发的不容易。它涉及到很多方面的知识，开发它必须要有具有严密的思维、非常全面的专业知识、认真的工作态度以及较高发现问题、解决问题的能力，能够攻坚克难。而我在一些方面做的还是不够好，这需要我不断地学

习和实践来提高自己的开发水平。

参考文献

1. GB/T11457-89 《软件工程术语》

2. GB8567-88 《计算机软件开发文件编制指南》

3. GB/T 12505 《计算机软件配置管理计划规范》

4. GB8566-88 《计算机软件开发规范》

5. GB/T 12504 《计算机软件质量保证计划规范》

6. GB9385-88 《计算机软件需求分析说明编制指南》

致 谢

感谢伍老师对我完成毕业论文的亲切指导和帮助，使我得到不少的启发和提高，伍老师富有启发式的建议和各个方面的大力支持使得本文的研究、开发工作得到了顺利进行。

感谢华南理工大学网络学院的领导和老师们，在学习过程中，给我创造了良好的学习环境，使我在思想、学习各方面都得到不断的进步。感谢我所有的同学，感谢我所在工作单位的所有同事，感谢他们从各方面给我的关心和鼓励，给了我不少的启发。

感谢我的父母及家人。感谢在我成长过程中教导过我的所有的老师，是他们给予我无微不至的关心和教诲，使我懂得了做人的道理，学到了不少的科学文化知识。帮助我在人生的道路上一步步向前迈进。



附录

1.%2 在Swift中，为什么访问struct比class快?

堆和栈并不是数据结构上的Heap跟Stack，而是程序运行中的不同内存空间。栈是程序启动的时候，系统事先分配的，使用过程中，系统不干预；堆是用的时候才向系统申请的，用完了需要交还，这个申请和交还的过程开销相对就比较大了。栈是编译时分配空间，而堆是动态分配（运行时分配空间），所以栈的速度快。

从两方面来考虑：

1. 分配和释放：堆在分配和释放时都要调用函数（MALLOC,FREE），比如分配时会到堆空间去寻找足够大小的空间（因为多次分配释放后会造成空洞），这些都会花费一定的时间，而栈却不需要这些。

2. 访问时间：访问堆的一个具体单元，需要两次访问内存，第一次得取得指针，第二次才是真正得数据，而栈只需访问一次。

---

关注微信公众号