

基于微信小程序的微服务化健康助手系统开发

摘 要

当下，互联网产品层出不穷，人们也变得异常忙碌，关乎个人健康的意识也似乎正在被忙碌的人群所唤醒。此刻需要一款同社会同类产品不同的软件产品。去解决已有系统的遗留问题，去解决用户的使用诉求。因此本项目致力打造一套关乎个人健康的微信小程序，并通过运维界面提升运维人员简单运维的目的方便运维人员进行服务弹性扩容。项目采用微服务架构思想，力争在完成项目的同时为用户提供更加完美的体验和健壮的服务。

在技术层面：项目采用微服务架构思想，将原有的庞大单体应用拆分成多个服务模块，方便服务模块动态扩容以提升节点负载能力；采用网关对流量合法性进行验证以及节点压力监控；项目采用中心化配置，项目中较为多变的配置发布在 Github 中，采用 Config 实时监控配置变化并实时推送给对应的服务模块。避免重新打包上线。

关键词 微服务；运维；微信小程序；健康

Development of micro service health assistant system based on wechat applet

Abstract

At present, Internet products emerge in an endless stream, people are also very busy, and the awareness of personal health seems to be awakened by busy people. Now we need a software product different from the same kind of social product. To solve the legacy problems of the existing system, to solve the user's use demands. Therefore, the project is committed to creating a set of wechat small programs related to personal health, and improving the simple operation and maintenance of operation and maintenance personnel through the operation and maintenance interface to facilitate the flexible expansion of service for operation and maintenance personnel. The project adopts the idea of microservice architecture, striving to provide users with more perfect experience and robust services while completing the project.

At the technical level: the project adopts the micro service architecture idea to split the original huge single application into multiple service modules, which is convenient for the dynamic expansion of service modules to improve the node load capacity; the gateway is used to verify the legitimacy of traffic and monitor the node pressure; the project adopts the centralized configuration, and the more changeable configuration in the project is published in GitHub, and the config real-time monitoring is used Configuration changes are pushed to the corresponding service module in real time. Avoid repacking to go live.

Keywords microservice; operation and maintenance; WeChat applet; health

目 录

第 1 章 绪论	1
1.1 项目背景	1
1.2 现状分析	1
1.3 主要工作	2
1.4 目的和意义	2
1.5 研究内容	3
1.6 章节介绍	3
1.7 小结	4
第 2 章 系统架构	5
2.1 语言选型及原因	5
2.1.1 项目采用 Java 语言原因如下	5
2.1.2 项目采用 VueJs 的原因如下	5
2.2 框架选型	6
2.3 微服务	7
2.4 总体架构	8
2.5 具体技术	9
2.5.1 GateWay 流量网关	9
2.5.2 Eureka 注册发现中心	10
2.5.3 Kafka 消息队列	11
2.5.4 Druid 数据库连接池	11
2.5.5 Hystrix 熔断与 Dashborad 仪表盘	12
2.5.6 配置文件动态配置	12
2.5.7 服务 RPC 与 Http	13
2.5.8 定时任务框架	13
2.6 项目难点	14
2.7 技术官网文档汇总	14
2.8 硬、软件开发环境	14
2.9 小结	15
第 3 章 概要设计	16
3.1 需求分析	16
3.2 用例分析	17
3.2.1 小程序端用例分析	17
3.2.2 运营 PC 端	18
3.3 数据库设计	19
3.4 界面设计	21
3.4.1 小程序界面设计	21
3.4.2 网页 PC 端	21
3.5 总体功能模块	21
3.6 小结	23
第 4 章 详细设计	24
4.1 简述	24
4.2 小程序免密登录	24
4.3 小程序请求封装	27
4.4 Gateway 流量鉴权	28

4.5 Kafka 记录流量	29
4.6 邮件告警	30
4.7 SpringBoot 统一配置	32
4.8 节点健康状态	34
4.9 小结	36
第5章 项目测试	37
5.1 简述	37
5.2 小程序免密登录测试	37
5.3 阅读后端发布文章	38
5.4 悄悄话模块	39
5.5 基本信息	40
5.6 每日目标	41
5.7 健康周报	42
5.8 划任务	42
5.9 系统帮助	43
5.10 节点健康检查	44
5.11 仪表盘	45
5.12 Druid 统计	45
5.13 Eureka 中心	45
5.14 食物重量定制化	46
5.15 小结	46
第6章 项目部署	47
6.1 简述	47
6.2 配置 JDK	47
6.3 配置 Zookeeper	47
6.4 配置 Kafka	47
6.5 部署业务模块	48
6.6 小结	49
第7章 总结	50
致 谢	51
参 考 文 献	52
附录一 译文	53
附录二 外文原文	62

第1章 绪论

1.1 项目背景

逢毕设项目编写之际，想利用用个人所学结合在校期间学习的 Java 编程、Web 编程、数据库设计，设计编写出高可用、性能良好、分区容错性高的软件项目。

毛主席曾经说过：“一个人要想做成一件事，必须拥有多方面的素质，但所有这些都必须依赖于一个前提条件：健康的体魄”。现如今，新一代年轻人越来越多的走向办公室，日常的办公都在相对狭隘的空间下完成无限的脑力活动。越来越少的年轻人靠体力劳动去实现自身价值。

社会正在飞速的发展，城市的生活节奏也变得非常的快，自然要求年轻一代花费更多的时间与精力投入到工作当中，熬夜工作，不吃早餐为了赶时间等情况比比皆是。

与之相比最可怕的是心理疾病。现如今，抑郁症等相关心理疾病也在逐渐控制着越来越多的人。

再此背景之下，真正做出一款 UI 良好、用户体验感良好、内容科学的软件去解决上述所提出的问题迫在眉睫。

1.2 现状分析

出于上述的背景之下。在开发之前还需要进行现有类似软件的调研，去研究现有软件的优点及缺点对于后期的项目设计与开发非常有帮助。

打开手机商城，搜索“健康”类别的软件比比皆是。

分别从项目 UI、项目易用性、体验感、内容科学性等多个方面试用市面上已存在的项目。UI 和体验感方面都非常的好，但基本都未为用户生成类似于用户报表的相关功能模块，这就导致用户在使用该软件的时候无法及时形象的收到个人相关反馈。比如某用户跟着软件进行健康生活的作息但一段时间之后并未收到个人训练的成果报表，这很可能让使用者慢慢的丧失对健康兴趣。

从功能全面性来看也有不少相关软件功能性缺失。比如只推荐健康饮食但是没有根据用户的健康状况去推荐。再比如，在健康标准中只选择了某一种算法，而实际上在国际上不同的人种的健康标准其实都不一样，健康的算法自然也就不一样。

因此，总结上述的问题，列出几个问题如下：

- （1） 健康算法种类不全，计算不严谨；
- （2） 没有及时反馈用户成果，使得用户丧失信息；
- （3） 文章内容没有及时的更新，部分内容不具有科学性；
- （4） 平台运维迭代工作较低。

1.3 主要工作

出于上述背景，决定做出一款健康类别的软件，包括饮食健康类别的软件，这其中包括身体健康和心理健康。尽可能的在该主题中满足大多数用户的需求，与此同时尽可能解决市面上已有类似项目的弊端。

微信的用户量巨大涉及到各个行业，不论是工作还是休闲生活，微信的用户群里只会越来越大。因此项目依托微信小程序进行开发，使得项目成立之初就具有大批量用户的潜能。

项目采用了微服务架构，所用技术多为目前主流、新颖的技术。在开发过程中，在项目中添加了大量关于运维的功能点。比如 SQL 记录、SQL 黑名单拦截；服务节点健康状况查询；Hystrix Dashborad 服务压力展示等相关技术。

在项目完全开发完毕之后进行一系列科学测试，在覆盖项目所有功能模块之外尽可能的覆盖项目所有分支，以保证项目的正确性。在测试优化项目之前编写每个功能对应的测试用例。项目使用百盒测试、黑盒测试、冒烟测试等相关测试技术手段进行全方位测试。

最终交付一篇论文用于对项目设计、开发、技术选型以及相关技术的特点、技术架构进行详细说明。

本次毕设将交付出一款“健康”类别的小程序及相关的运维平台，并命名为“惜己”小程序。

1.4 目的和意义

出于上述的背景分析，以及目前现状的分析，做出一款真正意义上的“健康”小程序与 PC 界面运维端。从用户需求入手，从方便运维方向进行入手，力争做出一款方便使用者进行健康事宜活动。使得使用者方便的使用项目，方便的查看个人的身体健康状态以及个人每日完成任务的折线图。

在运维方向，可以方便的看到各个服务节点的状况以及压力情况，方便运维人员

选择性的对相关服务模块进行扩容，通过 Eureka 进行注册，从而达成负载均衡效果，以减轻对应服务的负载压力。

1.5 研究内容

本毕业设计以微服务架构思想结合 Spring “全家桶”、“VueJs”、“微信小程序”一起做出一套带有用户端、运维端的健壮性良好的平台。

在开发中学习并将 SpringCloud 相关组件的学习及时的运用在项目中。

项目着重于微服务架构的设计思路，分别从流量合法性验证、流量网关、流量路由、注册发现中心、服务健壮性仪表盘、SQL 执行分析与统计、流量存储与统计（借助 Kafka 中间件）等来进行对项目健壮性的加固以及流量的实时分析以追求良好的用户体验以及良好的运维体验。

分别对上述提到的技术进行简短说明，在本论文的详细设计会有对该项技术的详细介绍：

- （1） Zuul Gateway 网关：用来做流量统一入口合法性鉴权并路由用户请求；
- （2） Eureka 注册中心：用来协调各个微服务节点；
- （3） Druid 数据源：开源于阿里巴巴，是最快且功能最全的数据源，内置的 Web 平台可以方便运维；
- （4） Kafka 消息队列：存储流式数据，某些场景下并不需要将数据持久化进入数据库，这样的场景往往需要保存在消息队列即可；
- （5） Hystrix Dashborad：用于服务压力展示以及服务熔断，方便运维人员及时发现可能宕机的服务并及时进行扩容以达到高压节点负载均衡的效果。

1.6 章节介绍

本论文一共分为七个章节，下面分别对着七个章节内容进行

项目的编写与测试，项目最终的打包公网的部署流程以及使用项目核心功能的展示与使用说明详见第六章。最后关于项目的编写感受，所领悟的知识，项目展望详见第七章。

- （1） 第一章：针对需求分析、主要工作、现状分析、研究内容、章节介绍来进行项目背景介绍；
- （2） 第二章：从系统架构详细描述系统的技术选型，该章节是技术相关核心内

容；

- （3） 第三章：概要设计方面描述数据库设计、UI 设计、用例图、系统功能模块等进行说明；
- （4） 第四章：详细的描述系统相关功能是如何开发，描述技术代码是如何实现；
- （5） 第五章：项目测试，测试各个功能模块功能能否正确无误的运行；
- （6） 第六章：项目部署，因为项目为多结点项目，因此有必要进行节点部署、服务环境搭建的说明；
- （7） 第七章：总结，包括项目的编写新的包括对以后的展望和认识现在的不足。

1.7 小结

在本章节中，从需求方面以及目前现状两个方面分析了该毕设项目的意义与重要性。从技术方面规划了项目研究内容，之后的开发围绕着这些技术点进行开发。在本章节的最后对本论文章节进行简短说明。

第2章 系统架构

2.1 语言选型及原因

项目采用 Java 后端开发，VueJs 以及微信小程序开发作为前端开发语言。

2.1.1 项目采用 Java 语言原因如下

1.Java 语言的普遍性，社区群体庞大，网上解决方案很多；

2.Java 的顺风车可以直接对接 Spring 全家桶，可以开箱即用 Spring 提供的很多解决方案，这里当然包括 SpringCloud 微服务的一站式解决方案；

3.Java 语言的灵活性：Java 不仅可以编译加载已经编写好的 Java 代码，更可以在运行期间动态的创建对象并调用其中的方法，这项技术就是 Java 提供的强大反射机制，是一种从 Class 文件反射成 Java 文件的技术；

4.Spring 提供的 AOP 技术可以方便快捷的补充 Java 的面向对象思想，AOP 是一种面向切面编程。其 AOP 的流行也得益于 Java 的动态代理技术；

5.一次编写到处运行，写好的程序打包之后不仅可以在 Windows OS 运行，也可以在 Linux OS 运行，这得益于 Java 的 JVM 技术，因为 JVM 的存在使得 Java 代码可以翻译成不同操作系统可识别的指令集合。

2.1.2 项目采用 VueJs 的原因如下

在前端的发展历程中，经历了几次改革，原生 JS 操作的复杂，譬如原生 JS 中操作 DOM 元素需要大量的代码，前端页面代码会显得特别臃肿与代码膨胀，在这之后有各大厂商基于 JS 封装自己的库，这其中 JQuery 就是封装的比较完美的 JS 库，JQuery 虽然开发起来简洁很多，但使用 JQuery 的开发效率也并不算高，再加上 JQuery 封装了大量的 API，需要开发者有较强的记忆性。VUEJS 的出现几乎解决了上述的问题，VUEJS 基于观察者模式的 MVVM 模型使得页面几乎不需要操作任何的 DOM 元素而是简单的使用模板语言就可以解决。

上述是从开发者角度而言，下面从不同脚本语言浏览器渲染性能进行对比：

在了解浏览器渲染之前首先需要知道浏览器渲染页面的流程，如下图，是浏览器渲染 HTML 文件的过程如图 2-1 所示：

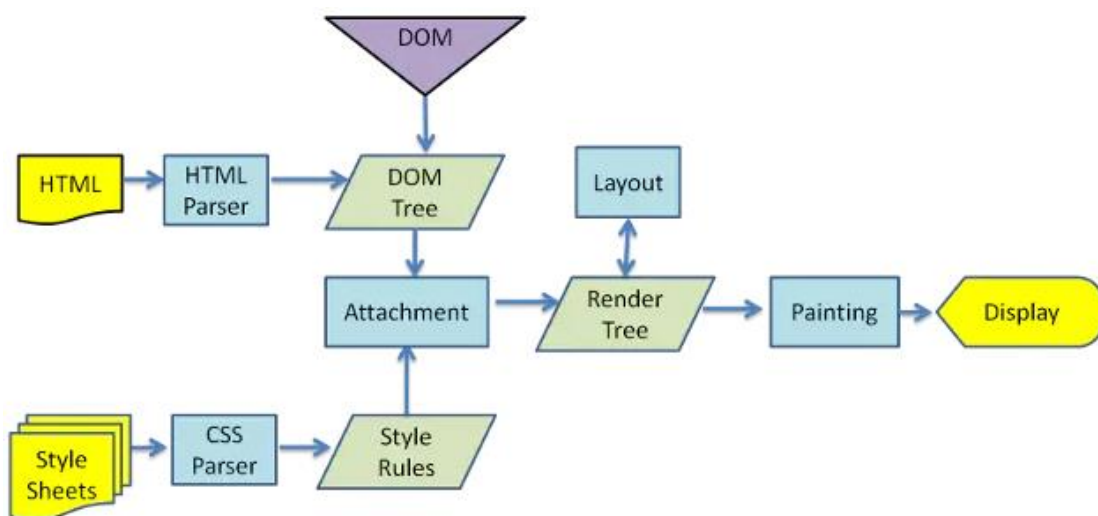


图 2-1 webkit 渲染流程

对上图简述为五个步骤：①根据 HTML 文件分析标签构造一颗 DOM 树；②CSS 分析器分析 CSS 文件和 HTML 标记中的 Inline 样式（css 内联样式）并根据优先级原则进行取舍，最终生成页面样式表；③将 DOM 树和样式表进行匹配关联生成一颗 Render 树；④浏览器根据生成的 Render 树进行唯一确定元素在浏览器页面的准确坐标轴；⑤根据元素坐标轴进行 Paint 绘制。经过上述步骤最终渲染出整个页面。

在明白上述的渲染原理之后还需要理解两个概念：重排（ReFlow）、重绘（Repaint）。其中 ReFlow 更加的影响页面的性能。在如下情况会导致页面发生重排：①加、减 DOM 元素；②元素的位置发生改变影响了原局部 DOM 树；③DOM 尺寸发生变化；④页面初始化；⑤窗口大小的改变。

明白了浏览器渲染原理与 ReFlow、Repaint 原理之后回顾 JS 与 JQuery 代码中经常有大量 DOM 树的操作。这样一来页面的 ReFlow 次数将会非常多，页面性能开销巨大显得网页卡顿现象严重。但 VUEJS 内部已经对其进行了优化，在收到 DOM 元素更新时，其会先维护在一个队列中并对重排的 CSS 和新的页面唯一进行合并，最终进行一次重排，这样大大节省了页面的性能开销。

2.2 框架选型

项目后端使用基于 Spring 框架。具体而言后端使用了 SpringBoot 框架开发，SpringBoot 相较于其他的 Java 后端框架有配置少、内置 Tomcat 等优势，使得开发后端变得容易。除此之外项目使用 SpringCloud 作为微服务一站式解决方案。

项目前端使用 IViewUi、ColorUi、Vant 组件库或 CSS 库，方便前端快速开发，

可以在短时间内完成界面风格统一等优势。

2.3 微服务

现代应用中，为了更好的应用健壮性，微服务的思想也逐渐成为了流行的架构思路，因此有必要花合适的篇幅对微服务思想进行描述。

微服务思想由 Martin Fowler 提出，概述其提出的微服务为一句话：将原来庞大的单体应用拆分成一组小型服务，在服务和服务之间采用 HTTP 协议的 RestFul API 进行通信，最终为用户创造价值。

微服务是一种开发思想并不是落地的某项技术，微服务的诞生完全颠覆了以往的单体应用的开发模式。

下图展示了软件架构的演变过程，如图 2-2：

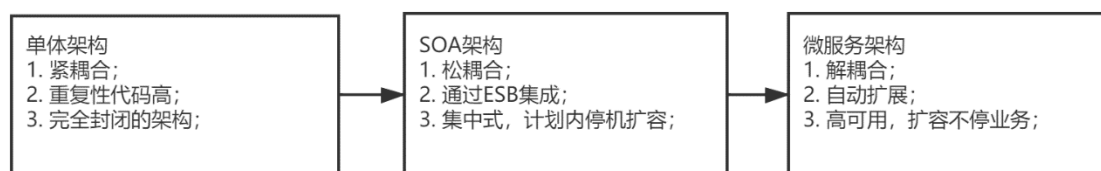


图 2-2 服务架构演变

因为微服务的特性，其优势也不言而喻：①避免出现庞大、复杂的单体“怪兽”；②专门的团队做专业的服务（框架设计、数据库的选型不会被局限为一种）；③拆分出的每个微服务独立部署互不影响，相互之前通过远程任务调度框架协调工作共同完成用户的请求。这里列出几个常见的 RPC、Http 框架：Alibaba 厂商的 Dubbo、SpringCloud 封装的 FeignClient、大众点评的 XXL-RPC 等等；④使得上线项目的弹性扩容成为可能。

在此可以做一个简单的假设，可以更加直观的了解微服务的优势：

假设有一个后端服务，其对外提供着成百上千个接口，假设突然有一天服务器宕机，这可能导致所有的接口无法对外提供快速、优质的服务，且造成大量请求数据丢失，那么后果将不堪设想。

当然，微服务架构并不是没有缺点：①原有的庞大单体应用拆分成多个服务，因此拆分的粒度不易掌握，业界也没有明确的参考论证；②微服务应用是分布式系统，其有固有的开发复杂性，服务和服务之间需要通过 RPC 调度，而不像 Java 代码中的进程调度；③数据库的一致性设计（尤其的数据一致性要求很高的系统中，如证券、

交易平台）对于开发者而言将是更大的挑战；④如果服务之间没有处理得当，很容易发生可怕的【服务雪崩】效应。

此处，对服务雪崩效应做出简单说明（前提开发者未对服务做特殊处理）：服务相互之间调用，当某个服务突然变得不可用时，那么依赖他的服务都会出现请求等待状态指导服务超时被销毁，这就是请求积压，但需要知道的是，服务在请求等待状态时依然会消耗 CPU 资源。假设有一万个请求都依赖于某个已经坏掉的服务，那么这一万个请求会进入请求等待状态并消耗着服务器资源，久而久之导致 Java 出现可怕的 OutOfMemory Exception.最终导致该服务坏掉，同样的道理这样的恶性现象一旦出现会逐渐让整个微服务都变得不可用，这是一个渐变的过程。这就是可怕的【服务雪崩效应】。

2.4 总体架构

1. 项目采用目前主流的前后端分离模式开发，该模式有很多好处，譬如：前后端完全解耦、前端工程化开发完全不依赖于后端、升级版本可减少对方停机损失... ...可以说前后端分离是传统软件开发漫漫长路上具有划时代的意义的大事件。

2. 项目采用微服务的架构。

下图为项目架构图：

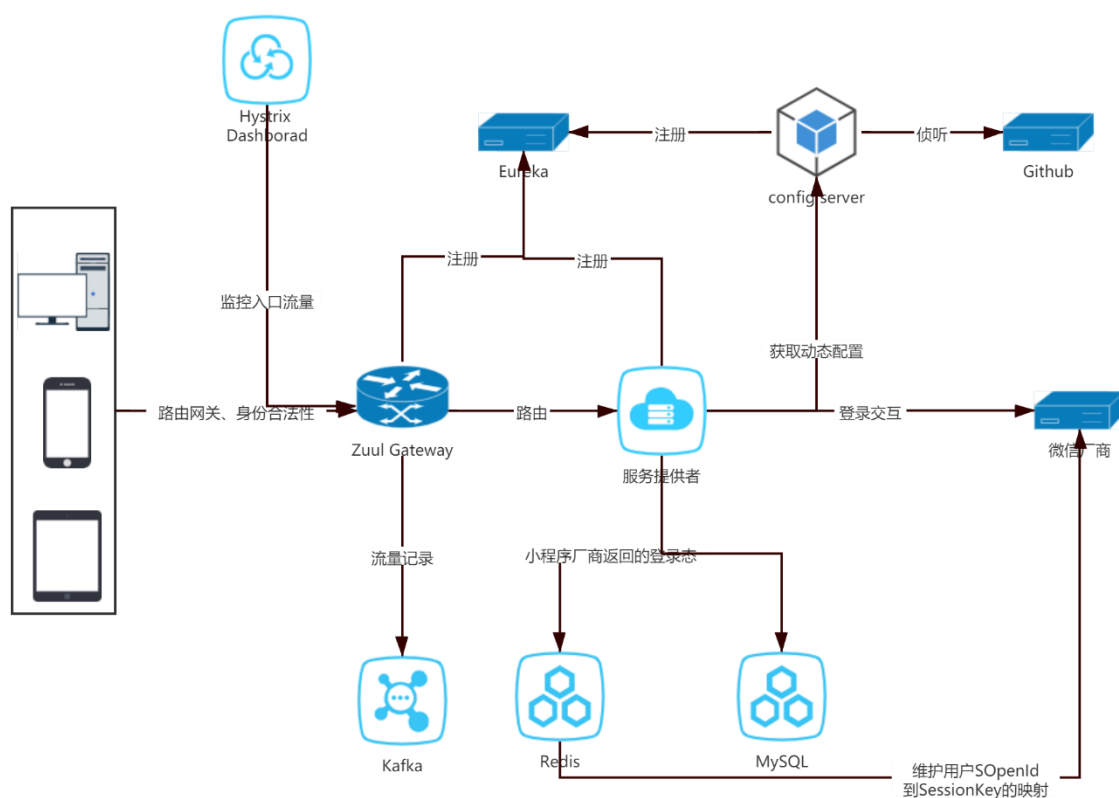


图 2-3 项目架构图

使用文字对上述框架进行简述：

所有的流量统一从 Gateway 进入，对用户的身份进行合法性认证，并使用 Kafka 消息中间件对流量进行记录以及 Hystrix Dashborad 对入口流量监控。若身份合法就会进入服务提供者，服务提供者根据获取到的用户信息去 Redis 中查询该用户是否已经免密登录微信服务，若是，则直接提供服务，若不是则返回登录页面。因为服务和提供者之间需要相互调度，这里采用 Eureka 服务注册发现中心维护各个服务的关系表。

2.5 具体技术

2.5.1 GateWay 流量网关

Zuul 图标：



图 2-4 Zuul 图标

Zuul 在业界中一直被称为“门户神兽”，其提供着各类型的 Filter 为系统的安全

性和系统容错性提供了强大的支撑，并且 Zuul 内聚整合了 Hystrix 提供了强大的总入口的服务降维处理。在 SpringCloud 整合的 Zuul 中还包括了 Ribbon 的负载均衡，Ribbon 提供的重试机制以及配置的超时时间与 Hystrix 配置的熔断时间相结合，可以使得项目在满足负载均衡的同时阻止服务雪崩的现象。

GateWay 会根据流量请求的地址去 Eureka（微服务下的注册发现中心）中查找服务的真实地址并引导流量进行路由到真实的服务提供者，当然这些流程对用户而言都是无感知的，是透明的。

2.5.2 Eureka 注册发现中心

Eureka 图标：



图 2-5 Eureka 图标

Eureka 是常见的服务注册发现中心，它的作用在于分布式中，当多个服务提供者不在同一个服务中，那么相互之间的通信如果全部依赖于 IP 地址加端口加服务路径那么将变得非常麻烦，如果将来服务迁移到了其他服务器，那么依赖它的服务都需要需要手动的修改 IP 地址以及端口号，这样就导致需要经常弹性扩容和服务迁移变得非常的麻烦。因此希望我们能够给服务取一个别名，然后让这个别名和服务的 Ip 地址进行动态的绑定并且需要动态的感知到 IP 地址的变化，这样的需求就需要 Eureka 的支持。

可以讲 Eureka 简单的理解为网络中的 DNS。

当然，满足该需求的除了 Eureka 技术，还有 Zookeeper 技术。

Eureka VS Zookeeper

在著名的分布式架构思想中著名的 CAP 原则，其中 C 表示强一致性，A 表示高可用性，P 表示分区容错性。在这三个字母中，不论是什么样的技术也只能实现其中的两个，要么是 CP，亦或是 AP。对于 Zookeeper 而言其选择的是 CP，它需要保证高强度的一致性，这就导致，当 Zookeeper 正在选择 Leader 或者超过半数以上的 Zookeeper 宕机时，为了保证整个项目的高强度一致性，Zookeeper 不会对外提供服务，

那么整个项目都会变得不可用。与 Zookeeper 不同的是, Eureka 选择了 AP, 在 Eureka 的世界里, 项目的服务可用性远比不可用要强的多。项目考虑到服务的高可用性选择放弃了“强一致性”但不代表放弃了“一致性”原则。因此选择了 Eureka 作为服务注册中心。

2.5.3 Kafka 消息队列

图标:



图 2-6 Kafka 图标

在项目中使用了 Kafka 消息中间件, 消息中间件有很多比如此处的 Kafka 还有 ActiveMQ... .. 在这其中被使用最多的就是 Kafka 了, 从源码开源的 GitHub 上来看, Kafka 的 Star 数为 15.7k 远远比其他 MQ 消息队列要多得多。

Kafka 主要应对与数据流式处理的场景。比如日志、流量统计等场景。对于这些数据而言, 并不适合将数据落盘到关系型数据库或者是非关系型数据库, 因为这些数据并不重要到浪费磁盘空间或者浪费内存。这些数据只适合于保存一段时间 (Kafka 对某条数据默认保存 168 小时), 消费者可以对最近一段时间的数据进行消费统计分析。

2.5.4 Druid 数据库连接池

图标: 暂无图标

在项目数据源连接池技术选型中, 需要先明白连接池的作用。

数据源连接池的概念可以先类比到线程池的概念, 这些概念都可以统称为“池”类技术, “池”类技术的诞生都源于性能消耗的考虑, 比如创建一个线程、创建一个数据源对象都是非常消耗性能的一件事, 但是整个项目不可能就只有一个线程一个数据源连接, 因为这样是无法应对高并发的场景。因此, 项目中可以创建一个“池”, 可以在初始化阶段为这些池创建很多对应的实例, 这样下面需要使用到这些实例的时候直接从对应的线程池中去获取即可。而这些实例的个数往往大于 1 个, 这样又可以满足高并发的基础需求, 也不至于在今后的每次使用之前都需要手动的创建实例。既

省开销又满足一定的并发需求。

在所有的数据源连接池中（比如：C3P0、DBCP、Druid、），Druid 是兼顾性能与功能的完美结合体，Druid 内置的 Filter，比如 WallFilter 用于 SQL 防注入，StatFilter 用于监控统计，LogFilter 用于日志记录或者慢查询记录。甚至可以自定义 Filter 用于操作。

其中 StatFilter 非常强大，Druid 基于它可以统计各种 SQL 查询报表并内置运维界面，使得运维人员可以非常清晰的了解到在一段时间中执行了哪些 SQL，以及分布等情况。

也正是出于上述的考虑，项目使用了 Druid 数据源。

2.5.5 Hystrix 熔断与 Dashborad 仪表盘

图标：



图 2-7 Hystrix 图标

在本论文第一章中，详细的阐述了服务雪崩的效应，Hystrix 就是能阻止服务雪崩的利器。他能感知一个请求的请求时间，不论如何配置了 Ribbon 负载均衡，为了阻止服务雪崩，Hystrix 可以让超时的请求迅速退出并进行降维处理，这样可以非常友好的给出服务超时的回应与用户体验。

对于 HystrixDashborad 来说，Hystrix 可以量化的展示出服务的压力。为了开发的便利性，把 HystrixDashborad 对 Gateway 网关进行了监控，因为 Gateway 是流量压力的总入口，总入口的压力大就表明该项目的压力大，因为当前项目中为了节省开发的成本，服务提供者只有一个。

2.5.6 配置文件动态配置

图标：暂无

在 SpringBoot 开发中，某些配置文件我们希望可以动态的去配置并且理解生效，

比如项目中的 Log 日志级别... ..

而不希望由于修改了某些配置而需要重新启动项目。这个时候可以考虑 SpringCloud 提供的 SpringCloud-Config 技术。

该技术的使用简述如下：开启一个针对 Github 的服务，动态的感知保存在 Github 中的配置文件，然后推送给需要它的服务模块并动态加载配置。

2.5.7 服务 RPC 与 Http

图标：暂无

在分布式的架构中，不可避免的出现服务和不在同一个项目中，甚至不在同一个服务器中，这就会导致服务和之间的交互将会变得困难。

SpringCloud 封装了 FeignClient，其定位就是为了替代 Java 原始的 HttpURLConnection 和 HttpClient，因为 Java 原生的方式代码量非常的多且不易维护，在 IO 流的处理上还需要开发者有一定的经验。而 FeignClient 使得不同服务之间的调用就像调用接口一样的方便。

与此对应的是 Apache 的 Dubbo（以前由 Alibaba 进行维护，最后成为 Apache 的顶级项目）。Dubbo 也是基于接口，但他需要有 Zookeeper 的服务注册中心。他的特点是调用方像使用本项目中的类一样的去使用远程服务。

Dubbo 和 FeignClient 最大的区别，前者是基于 RPC 调度，FeignClient 是基于 Http 方式调度。

为了节省开发复杂和控制成本，项目不会再去假设 Zookeeper，因此直接采用 FeignClient 开发。

2.5.8 定时任务框架

图标：



图 2-8 Quartz 图标

项目中存在定时邮件告警，目前暂定于每天的整点时间进行将数据库记录的 ERROR 级别的日志进行发送到指定的邮箱。服务中除了需要有邮件发送的功能外，还需要有定时任务调度框架，并且该框架可以支持在线 CORN 表达式的修改，以此

来动态的修改定时任务。

考虑到复杂性，项目今后的升级会考虑将 Quartz 更换成大众点评的 XXL-JOB，因为其能够分布式的任务执行并不会有数据库的侵入性。

2.6 项目难点

项目因为是微服务的架构，因为微服务的世界里，每个服务都可以有其独立的数据库。那么就需要考虑数据库数据的一致性，这点是非常复杂的。

因为项目业务的功能性并不多且并不复杂，因此使用线上的一个数据库就可以完美的规避数据一致性的问题。

微服务的项目必然是多模块化，因为所有的服务模块都是基于 SpringBoot 开发，那么在整合的时候可能出现 SpringBoot 配置文件冲突或者未被加载，这一点随着功能的增加会变得尤为的复杂。需要开发者对每个配置文件的熟悉，可以在每个配置文件中添加响应的注释以往忘记。

2.7 技术官网文档汇总

- 1.SpringBoot: <https://spring.io/projects/spring-boot>
- 2.SpringCloud: <https://spring.io/projects/spring-cloud>
- 3.Gateway: <https://spring.io/projects/spring-cloud-gateway>
- 4.Kafka: <http://kafka.apache.org/>
- 5.Hystrix: <https://github.com/Netflix/Hystrix>
- 6.MySQL: <https://www.mysql.com/>
- 7.Quartz: <http://www.quartz-scheduler.org/>

2.8 硬、软件开发环境

- 1.JDK 版本: 1.8.0_131
- 2.WebStorm 版本: 2018.2.4
- 3.IntelliJ IDEA 版本: 2018.2.4
- 4.OS 版本: Windows 10 64Bit
- 5.MySQL 版本: 5.5
- 6.Redis 版本: 2.8
- 7.NodeJs 版本: v10.16.0

8.VueJs 版本: v2.9.6

2.9 小结

本章节中，从项目语言选型开始，阐述了项目后端采用 Java 语言，项目前端使用微信小程序以及 VueJs 的背景及原因。后来分析了软件行业中架构演变的过程，分析了以往单一架构的缺点，以及现在微服务架构的优点及缺点。之后分析了项目中所使用的新技术，逐一对新技术的介绍以及使用场景。

第3章 概要设计

3.1 需求分析

从宏观来看，项目分为一个后端项目，两个前端项目，两个前端分别为：小程序端和 PC 运营端。

小程序端：

①需要对用户的健康状况进行收集，在收集的过程中需要使用 BMI 等身体指数计算公式进行对用户身体状况进行计算；

②根据小程序使用者填写的身体参数并根据中国饮食膳食标准提供定制化的饮食推荐；

③用户可以实时的填写今天已经完成了哪些定制化推荐的任务，并填写实时的百分比；

④系统根据用户填写的定制化需求的百分比进行实时的计算当天各个任务完成的任务量占据总任务量的百分比，为了更加直观的展示给用户，这些数据都将使用百度 Echarts 产品进行数据展示，方便用户直观的对比每天完成任务量的趋势图以及每一项的完成占比；

⑤平台提供“悄悄话”功能，打破平台用户与用户之间交际壁垒并，在此功能上在不违法的情况下可以进行畅所欲言，释放压力或者分享收获的喜悦亦或者诉说内心的委屈，用户也可对其进行评论或私信，私信将以站内消息的形式发送给文章编写者。

运维 PC 端：

①运维端需要对项目中饮食种类以及定制化进行数据添加、删除、修改、查询；

②运维界面需要对接微信厂商获取用户访问趋势图以及页面访问量等，方便运维者进行及时调整项目功能；

③运维平台需要展示页面 SQL 执行情况、执行时间分布、由哪些 URL 触发的 SQL、SQL 执行黑名单、白名单展示，以及 SQL 执行的时间，这些数据可以非常方便告知开发者，哪些 SQL 需要进行优化；

④运维平台需要展示服务压力，并且以图表形式直观展示，这样可以方便告知运维者需要将哪些节点弹性扩容以释放并发压力；

⑤平台运营者需要编写关键健康类的文章以提供小程序首页的支持

后端：

①为小程序端和 PC 端提供接口支持；

②支持邮件发送功能并整合 Quartz 进行项目错误日志发送邮箱进行错误邮箱告警，并且日志错误级别是可配置的。

3.2 用例分析

3.2.1 小程序端用例分析

从小程序使用者出发，用户使用微信小程序进行免密登录，登录进去之后小程序大体分为三个模块，分别是：“首页”、“悄悄话”、“个人中心”三大模块。首页对接运营者写的健康类别的文章；悄悄话模块，用户可看到别人说的“悄悄话”，可以对其追加评论或者私信作者引发共鸣；“个人中心”中可以查看个人编写记录个人的身体参数系统会生成每日目标，用户填写每日定制化任务的完成度，系统以此来进行生成健康日报。如下图，为系统用例图：

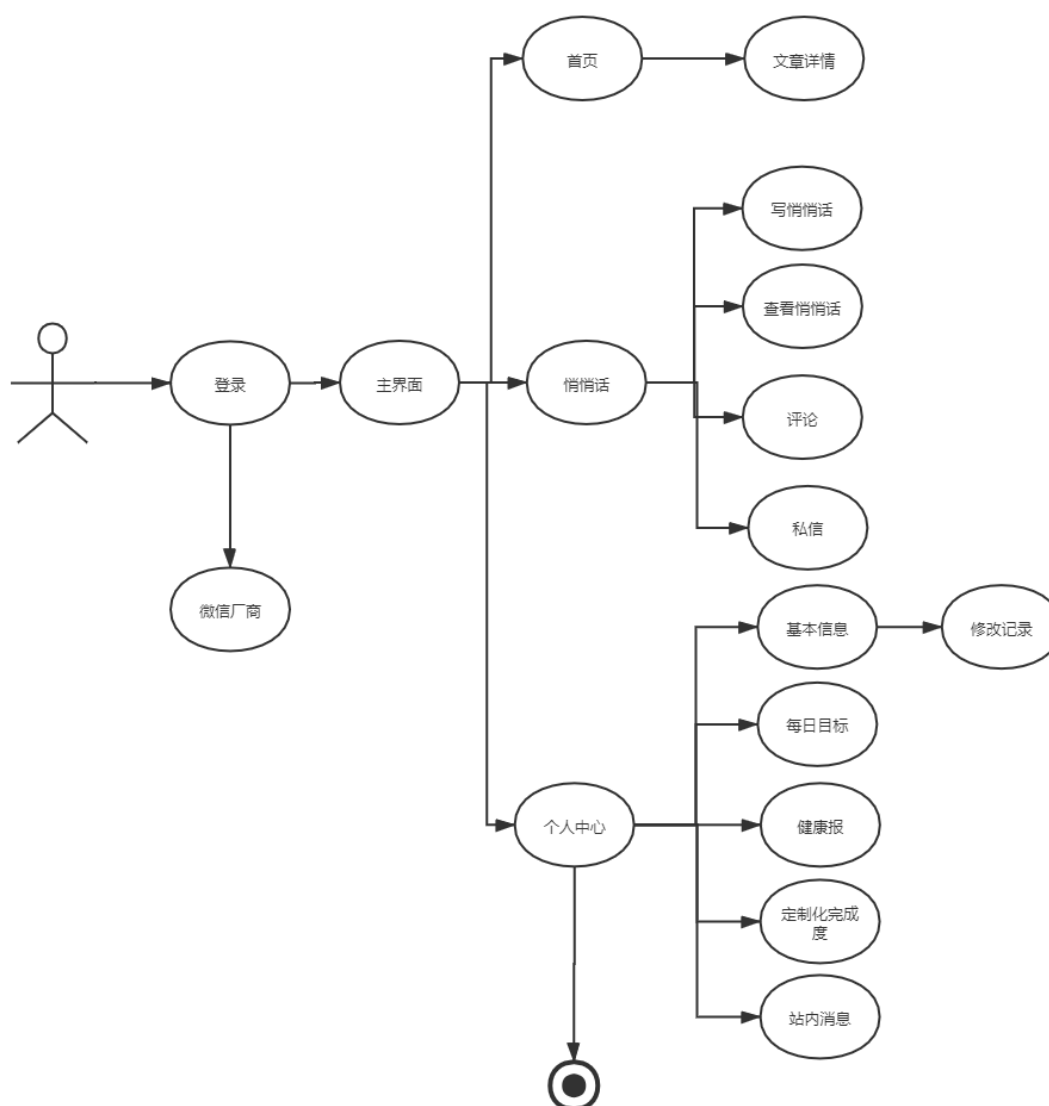


图 3-1 小程序端用例图

3.2.2 运营 PC 端

运营这登录运营系统之后，进入主界面，主界面左侧有很多一级菜单而每个一级菜单都表示一个功能点，分别有“文章编写”、“服务仪表盘”、“节点健康状态”、“Druid 统计”、“菜品管理”。如下图，为运维 PC 端用例图：

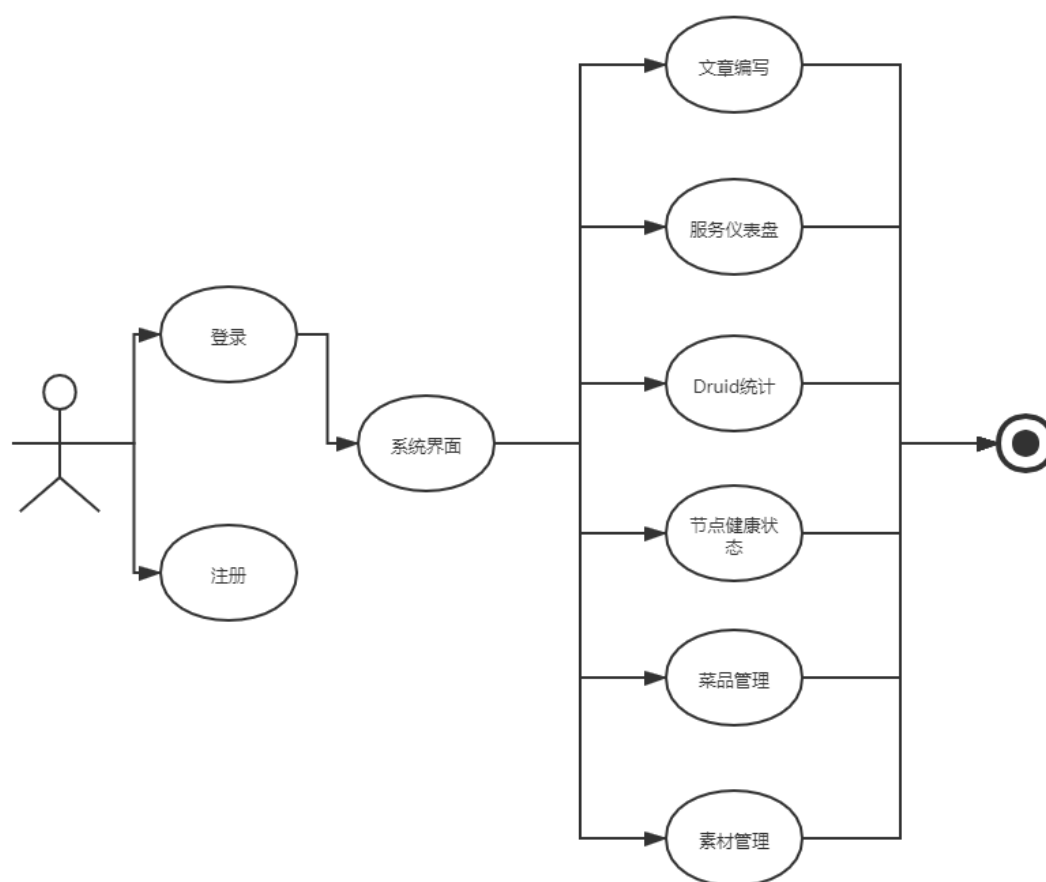


图 3-2 运营 PC 端用例图

3.3 数据库设计

概要：在数据库设计中，根据企业实习的经验来看，数据库设计使用 Innodb 数据库引擎，通过锁的机制来保证数据库事务，进而保证数据的可靠性和一致性。

主外键关联：在数据库的数据一致性中而言，数据的主外键的存在也在一定的程度上保证了数据的可靠性和正确性。但这在一定的程度上添加了开发的维护难度，在开发初期，“子”数据的删除必须要保证“父”数据已经删除，很显然在开发过程中带来了开发成本，在每一次的数据删除之前都需要对数据进行是否“可删除”的逻辑判断。

索引方面：数据库中的索引，可以大大的提高数据查询的能力。以 MySQL 关系型数据库而言，索引分为：普通索引、唯一索引、主键索引和聚集索引。项目对于每一张表而言，都使用了主键索引并且根据整形递增的方式进行保存索引。

表数目：数据库中存在表数为 26 张表，其中 11 张表为定时任务相关表，剩下的 15 张表为业务相关表。

多表查询：在数据库中多表关联查询的方式有很多种，项目中多采用左外连接的方式（Left Join）。

以下为项目数据库 ER 图，如下图：

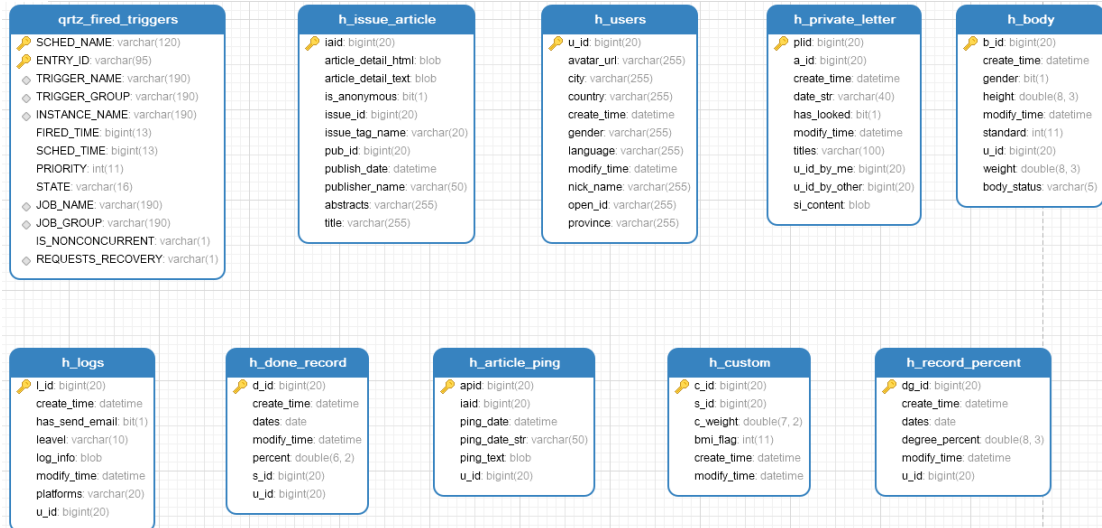


图 3-3 ER 图

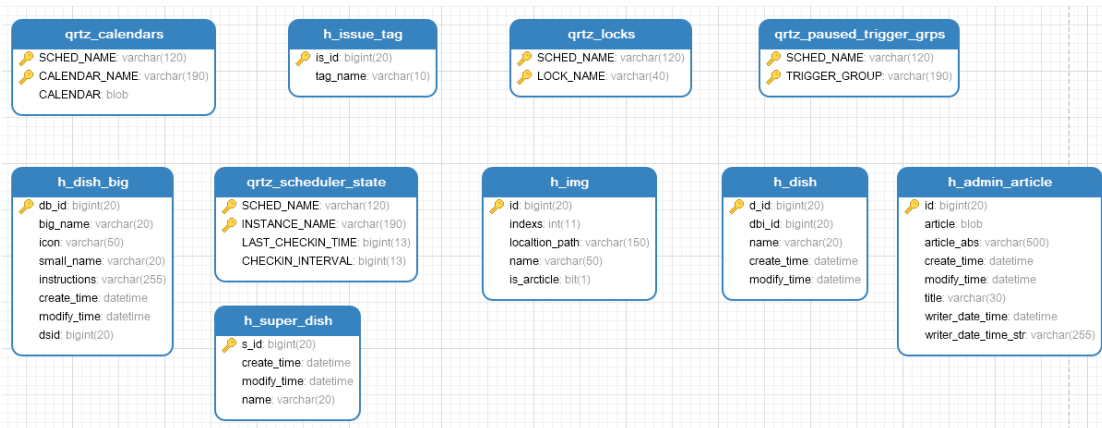


图 3-4 ER 图

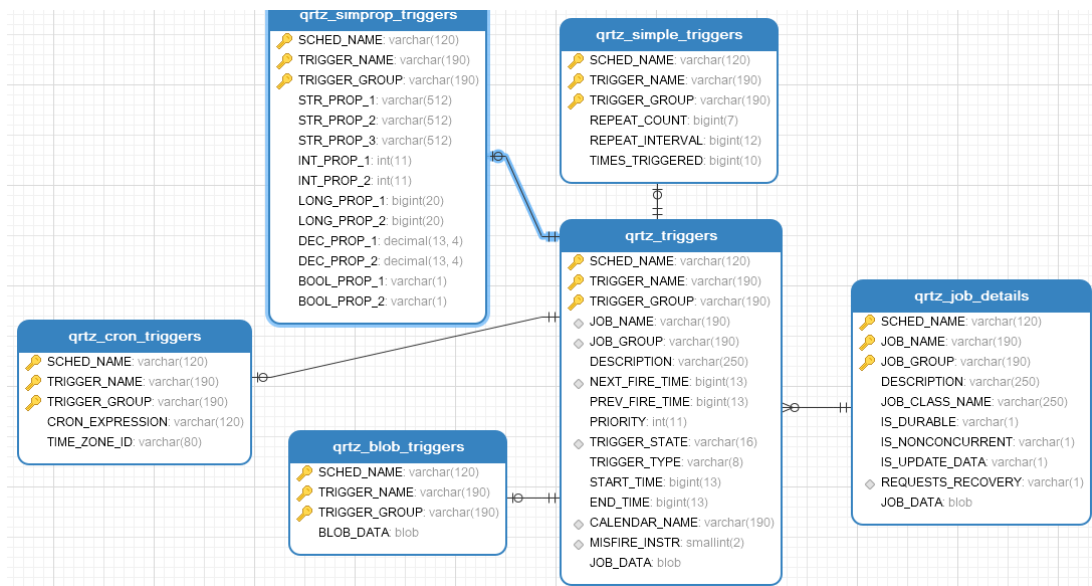


图 3-5 ER 图

3.4 界面设计

3.4.1 小程序界面设计

主题色十六进制：#2d355b，蓝色系；

图标：图标采用 Iconfont 的多彩扁平化图标；

辅助性文字颜色为#657180；

3.4.2 网页 PC 端

主题色十六进制：#515a6e；

网页控件背景色：#2b85e4（中性主题色）、#ff9900（警告主题色）、#ed4014（错误主题色）、#808695（辅助性文字）；

3.5 总体功能模块

简述：在本项目中分为三个开发项目，分别为微信小程序端、运维 PC 端，Java 后端，其中前两者统称为项目前端，最后者称为项目后端。由于图片排版问题，将这三者的功能图分开说明：

小程序端总体功能模块，小程序端主要给普通用户所使用，如下图：

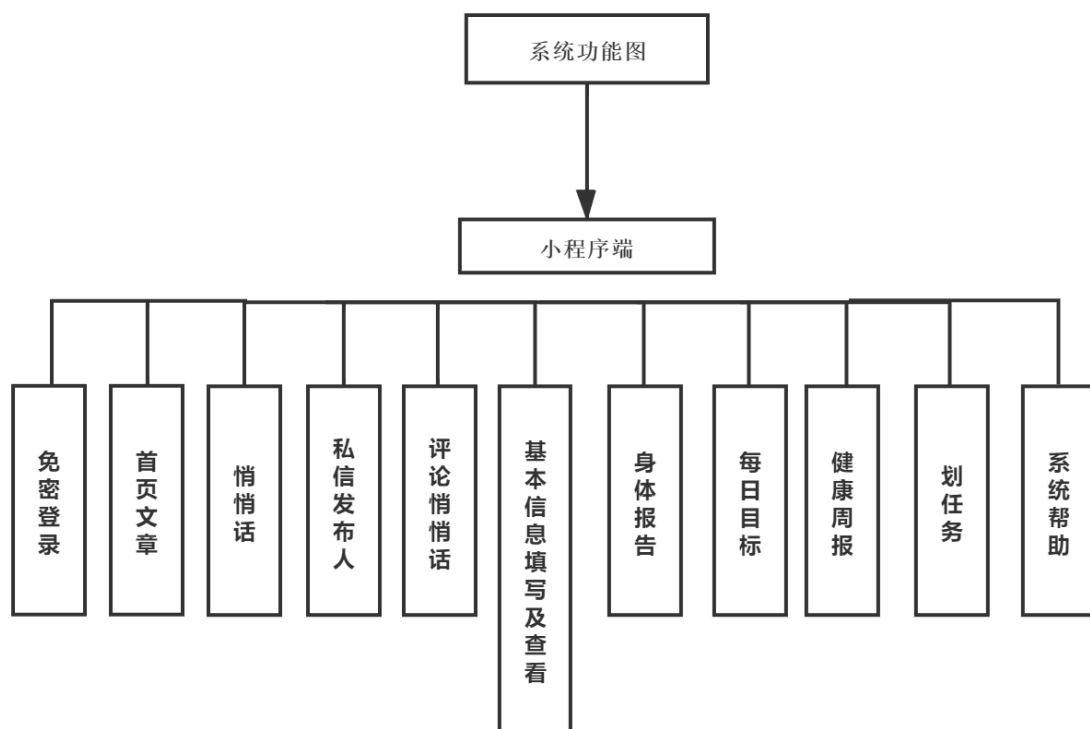


图 3-6 小程序端总体功能模块图

运维 PC 端总体功能模块，主要给运维者使用，如下图：

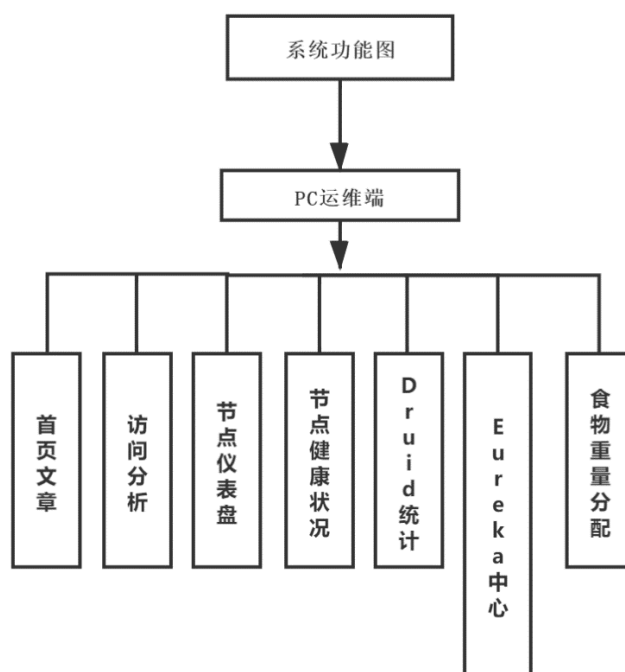


图 3-7 运维 PC 端总体功能模块图

Java 后端功能，主要为整个项目提供技术与接口支持，如下图：

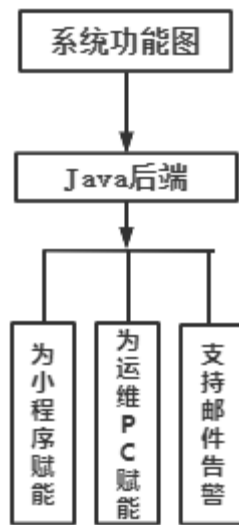


图 3-8 Java 后台总体功能模块图

3.6 小结

本章节开头部分分析了需求，从需求入手进行拆分出一个个功能模块。之后根据需求和功能画出小程序端和 PC 端的用例分析图。功能性需求分析完毕之后进行对数据库表的设计，以及前端的 UI 设计。本章节最后给出小程序、运营 PC 端、Java 后端的总体功能图。

第4章 详细设计

4.1 简述

本章节为项目的详细设计，由于项目角色多，功能较多，由于篇幅的问题，本章节将会选取重点功能和核心功能进行从需求分析，思路分析、生命周期图方法、核心代码说明等方面进行说明。

4.2 小程序免密登录

简述：项目使用微信小程序作为用户的使用平台是出于微信庞大群体生态圈的考虑，既然使用了微信小程序，那么用户免密登录而使用微信自然的登录接口是必然的选择。因为微信小程序的登录逻辑较为复杂，因此有必要在“系统详细设计”中进行详细说明：

微信官网给出了对接微信厂商进行用户免密登录的生命周期图。如下图：

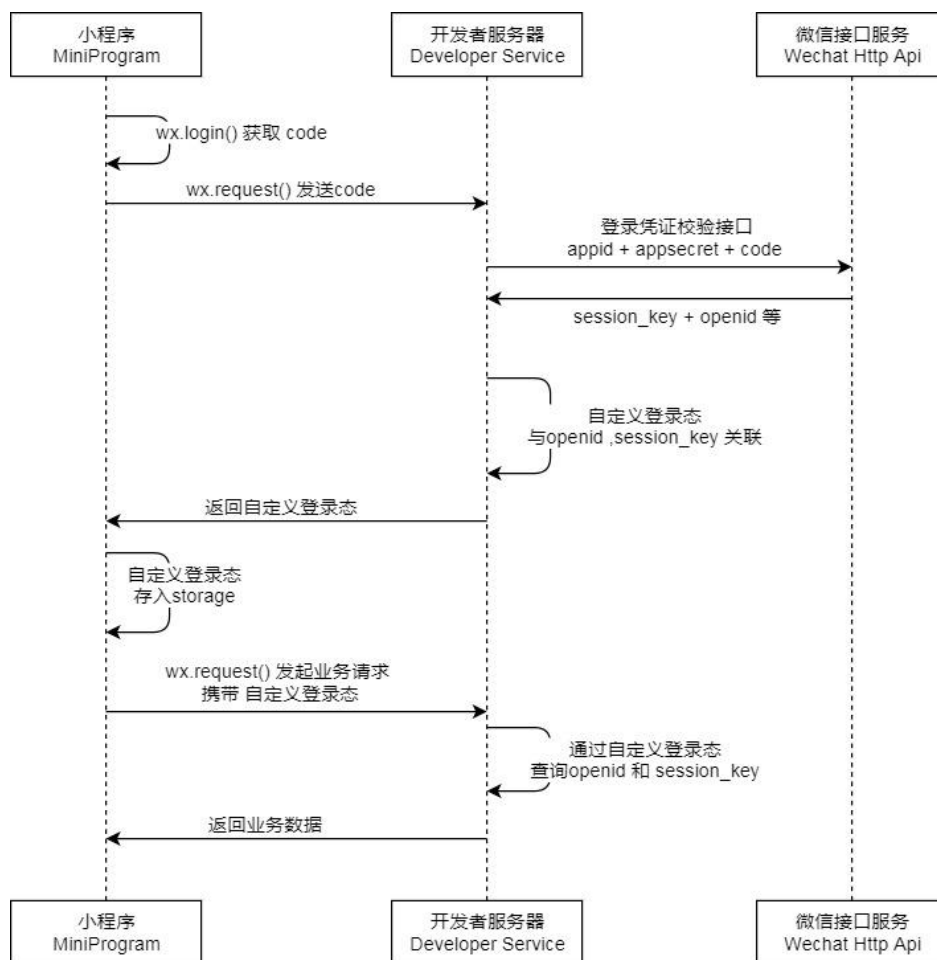


图 4-1 Java 后台总体功能模块图

文字描述如下：

② 小程序端触发 `wx.login()` 获取一次性登录凭证 `Code`，并传递到后端服务中；

② 开发者服务携带收到的 `Code` 请求微信厂商的 `auth,code2Session` 进行换取该用户的 `OpenId` 和 `SessionKey`，需要注意的是 `SessionKey` 的存活时间并不是一成不变；

③ 开发者服务器根据获取的 `OpenId`、`SessionKey` 自行组织系统凭证；

注：在微信登录流程中有必要对 `SessionKey` 和 `OpenId` 进行了解，

SessionKey：微信官方开发文档给出了 `SessionKey` 以及其生命周期的详细解释，大概意思为：用户 `SessionKey` 的过期时间对于开发者而言完全透明，某个用户在某次使用中，使用的越频繁 `SessionKey` 的生命周期就会越长。

OpenId：`OpenId` 是一个用户对一个微信小程序的唯一认证，简单而言类似于该用户在该小程序用户表中的主键 ID。

了解了登录流程以及必要字段的说明，下面对上述步骤二进行详细说明，因为步骤二直接关系系统合法性用户的判断方案。

① 平台不会直接使用 `OpenId` 作为用户的主键，引用微信官方文档是：`OpenId`、`SessionKey` 都是较为隐私的数据；

② 当用户发起登录流程时，首先经过项目的路由网关（网关的白名单决定不对登录接口进行认证检查）并放行到服务提供者，服务提供者会拿着用户传递来的 `Code` 请求去请求微信厂商并获得 `OpenId` 和 `SessionKey`；

③ 维护 `OpenId->SessionKey` 的映射关系并保存在 `Redis` 缓存中；

④ 使用 `JWT` 技术，将 `OpenId` 和用户的 ID 放入 `JWT` 令牌中并返回小程序端，往后的请求只需在请求头 `Header` 中携带 `JWT` 令牌即可。后端收到请求之后会对用户令牌进行认证，认证通过即可进入服务，验证不通过返回登录页面；

⑤ 微信厂商的某些接口需要使用用户的 `SessionKey`，若用到 `SessionKey`，即根据 `OpenId` 去缓存中换取 `SessionKey` 即可。

小程序端核心代码如下：

```
wx.login({
  success: res => {
    request.sendHttp('POST', '/user/login', {
      code: res.code
    }).then(res => {
```

```

let result = res.data.result;
if (!result.users) {
  this.setData({
    btn_disable: false
  })
} else {
  app.globalData.userInfo = result.users;
  util.go.switchTab('/pages/index/index', null);
}
app.globalData.token = result.jwt;
}).catch(err => {
  console.log(err);
});
},
fail: function(res) {},
complete: function(res) {
  wx.hideLoading();
}
})

```

后端核心代码

```

public Map<String, Object> login(Map<String, String> map) {
  Map<String, Object> result = new HashMap<>();
  Map<String, Object> jwtMap = new HashMap<>();
  String response = auth.getOpenIdAndSessionKey(appId, secret,
  map.get("code"), "authorization_code");
  try {
    map = objectMapper.readValue(response, Map.class);
    Users usersFromData =
  userDao.findByOpenId(String.valueOf(map.get("openid")));
    if (usersFromData != null) {
      jwtMap.put("uId", usersFromData.getUId());
      result.put("users", usersFromData.createUserVO());
      redisUtils.putSessionKey(map.get("openid"), map.get("session_key"));
    }
    jwtMap.put("openId", map.get("openid"));
    result.put("jwt", genJWTUtil.creatToken(jwtMap));
  } catch (JsonProcessingException e) {
    e.printStackTrace();
  } catch (JOSEException e) {
    e.printStackTrace();
  }
  return result;
}

```

4.3 小程序请求封装

简述：在小程序开发中，小程序提供请求 API（`wx.request`）进行前后端分离异步请求，而本项目就是采用原生小程序的请求方式。

在开发的过程中常常需要进行请求代码的编写用于调用后端提供的 API 接口，但这需要编写大量的重复性代码导致开发效率低下，因此说，在前端开发过程中，首先进行请求代码的封装是关键步骤，这直接影响着前端开发效率。

本项目中使用使用 `wx.request` 和 `Promise` 异步方案进行封装，对 `Content-type` 和请求数据也进行封装。

前端核心代码如下：

```
function sendHttp(method, url, data, header = {
  "Content-Type": "application/json",
  "Token": app.globalData.token
}) {
  return new Promise((resolve, reject) => {
    wx.request({
      url: server_api + url,
      method: method,
      data: data,
      header: header,
      success: res => {
        if (res.data.code === 1) {
          resolve(res);
        } else {
          util.toast.showToast(res.data.result, 'none');
          resolve(res);
        }
      },
      fail: err => {
        console.log(err);
        wx.showToast({
          title: 'NETERROR',
          icon: 'loading',
          duration: 2000
        })
        reject(err);
      }
    })
  })
}
```

对上述核心代码进行说明：

使用了 ES6 函数参数默认值的写法：

在 ES6 中支持了函数参数默认值填充，它存在的最大意义在于，对于一些不太容易变化的参数，在函数参数状态就进行初始化赋值，若调用者没有对其数据进行传参填充那么就使用默认值，若调用者在调用的同时对参数进行赋值，那么调用者的参数值将会覆盖方法的默认值，从而方便开发，一般而已，该技术点常常用于某个参数几乎固定不变的情况下。

使用了 ES6 的 Promise 的语法糖：

Promise 的“诞生”直接影响了整个前端开发人员，在早期，前后端分离的异步请求需要采用 JS 构建出 XMLHttpRequest 对象进行数据的填充，但这开发起来代码量非常大，后来 Ajax 的出现解决了开发复杂性和代码膨胀的问题。但 Ajax 有他致命的弱点：“回调炼狱”（回调简单理解为提供方法给对方调用）。回调中代码出现大量的垂直结构，如下为简单的伪代码体现“回调炼狱”：`res->{res->{res 操作代码.....}}`。而 Promise 的出现会将回调变成扁平状态，于是上面的“回调炼狱”伪代码近乎斜撑：`res->{{}.res->{{}.res->{{}.....`

为了解决“回调炼狱”的问题，项目还采用了 Async、Await 的方式。

4.4 Gateway 流量鉴权

简述：项目中对于流量鉴权非常的重要，哪些流量是正常的，哪些流量是异常的，系统需要进行严格的判断，进而减少服务器的 DDS 攻击，从而浪费服务器资源。

由于项目为微服务架构思想，因此后期的多个服务模块若每个模块都进行鉴权操作，那么代码会非常的冗余，且在架构中给路由也代码一定的困难，因此项目使用 Zuul 网关进行统一流量入口，并对用户请求进行统一鉴权。

总体思路：构建网关服务，所有的请求统一请求到 Zuul 网关，网关的前置过滤器（Pre）拦截请求获取 RequestUrl、Header、Token，网关配置白名单（登录、注册、访问图片资源等相关路径）。若 RequestUrl 根据匹配原则匹配到了白名单，那么对该请求直接放行。若 RequestUrl 不属于白名单范围，就会验证请求头中的 Token 信息，Token 信息有 JWT 进行构造，若 JWT 合法则进行路由，若不合法则退出。

总体设计如下：

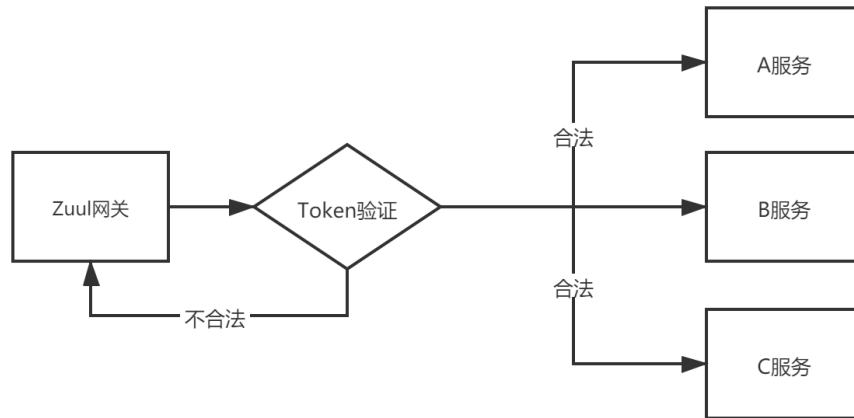


图 4-2 Zuul 验证

生成 Token 核心代码如下：

```

public String creatToken(Map<String, Object> payloadMap) throws JOSEException {
    JWSHeader jwsHeader = new JWSHeader(JWSAlgorithm.HS256);
    Payload payload = new Payload(new JSONObject(payloadMap));
    JWSSignature jwsObject = new JWSSignature(jwsHeader, payload);
    JWSSigner jwsSigner = new MACSigner(jwtSecret.getBytes());
    jwsObject.sign(jwsSigner);
    return jwsObject.serialize();
}
  
```

核心代码说明：

在 Jwt 中由三个部分组成，分别是头部，载荷，签名。其中头部采用了 HS256 的加密算法，载荷为放在 JWT 中的自定义信息，尾部为头部和载荷的压缩签名，尾部用于验证 JWT 是否有限是否合法是否被篡改。

4.5 Kafka 记录流量

简述：Kafka 为流式数据的短暂存储（Kafka 的数据保留一般设置为 7 天）。Kafka 一般用于存储日志信息或者流量记录统计，这些数据并不适合持久化在数据库中，而是暂时保存的一个地方，若后续需要可以对其进行数据消费。而这就是消息队列技术。

在本项目中消息队列用于存储系统流量。即在用户访问之后将用户的 ID 信息，访问 URL 等敏感信息进行记录。以防知道项目每日 PV 量以及访问者信息等。

为了统一进行记录，Kakfa 用于 Zuul Gateway 处，因为该模块是访问流量总入口。

总体设计如下：

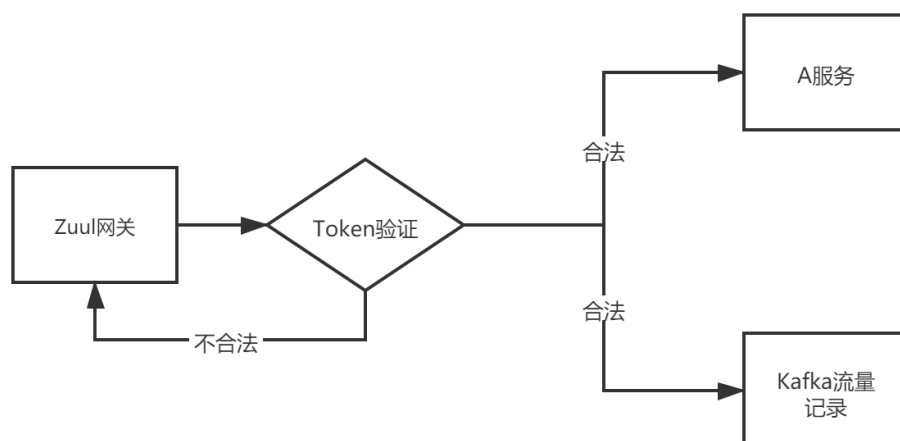


图 4-3 流量记录

核心代码如下：

```
public void sendToTopic(String topic, String jsonMsg) {
    kafkaTemplate.send(Contanst._FLOWTOPIC, jsonMsg);
    ListenableFuture<SendResult<String, String>> future =
    kafkaTemplate.send(topic, jsonMsg);
    future.addCallback(success -> {
        String sendTopicName = success.getProducerRecord().topic();
        Integer partition = success.getProducerRecord().partition();
        Long offset = success.getRecordMetadata().offset();
        log.info("\n Success:\nTopName: {}、Partition: {}、Offset: {}",
        sendTopicName, partition, offset);
    },
    fail -> log.error("\n ERROR \n {}", fail.getMessage()));
}
```

核心代码说明：

为了提供向消息队列中发送消息，项目中发送方法采用默认的异步方式进行消息发送。

但平台中想将发送信息是否成功以及发送完毕之后消息在消息队列中的位移（Offset）的显示，因此编写了发送消息的回调。在发送消息的成功回调中进行消息的位移记录等。

4.6 邮件告警

简述：在项目中邮件告警尤为的重要。尤其是互联网相关的软件，除非出现 Error 级别的大问题导致项目运行崩溃，否则开发人员一般无法感知到线上产品近期运行是

否健康的问题，除非开发人员去翻看项目日志，查看是否有 Error 级别的错误日志来判断，这种通过“人肉”查看日志的情形即费时费力还非常的低效。

因此项目再小程序端或者是后端都采用了异常统一处理的机制，当小程序端发现错误，将会统一收集并将错误信息发送给后端，由后端记录数据库，而对于 Java 后端，采用了 AOP 的原理进行异常捕获并记录在数据库中。

数据库中是否存在一个是否已经发送邮件的字段，项目中集成了 Quartz 定时调度框架，该框架通过设定的 CRON 表达式进行任务触发，而在一个任务中会收集数据库中的异常且未被发送过邮件的错误信息并整合起来，通过邮件的形式发送给内置设定好的开发人员。在邮件中设置了错误信息回调的功能，当开发人员收到邮件之后可点击异常详情则可进行错误信息详细查看。

总体设计如下：

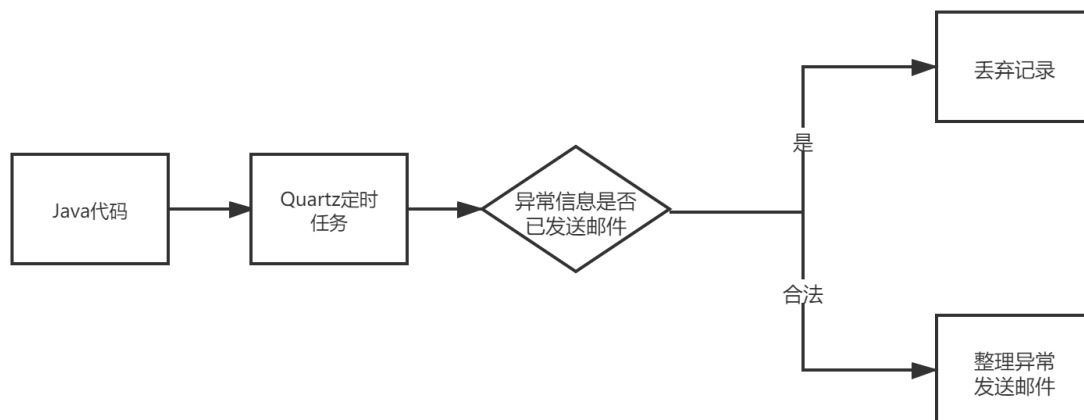


图 4-4 邮件告警

核心代码如下：

```

private void appendSegment(StringBuffer stringBuffer, boolean isAhref, Object data,
Long dataId) {
    stringBuffer.append("<td style=\"border:solid #add9c0; border-width:0px 1px 1px 0px;\">");
    if (isAhref) {
        stringBuffer.append("<span><a target='_blank' href=\"" + callBackServer +
"/log/by/id/" + dataId + "\">" + data + "</span></a>");
    } else {
        stringBuffer.append("<span>" + (data == null ? "暂无" : data) + "</span>");
    }
    stringBuffer.append("</td>");
}

```

上述代码说明：在上述代码中主要用来构建 HTML Segment，因为在本项目中所有的邮件告警采用 HTML 的方式，这样可以更加美观和直观的查看 Error 信息。

核心代码：

```
private boolean sendMail(StringBuffer htmlMessage) {
    if (StringUtils.isEmpty(htmlMessage)) {
        return false;
    }
    MimeMessage mimeMailMessage = javaMailSender.createMimeMessage();
    try {
        MimeMessageHelper mimeMessageHelper = new
MimeMessageHelper(mimeMailMessage, true);
        mimeMessageHelper.setFrom(sendMail);
        mimeMessageHelper.setTo(sendMail);
        mimeMessageHelper.setSubject("Healthy 平台日志运维，日志级别 > " +
emailLogLevel);
        mimeMessageHelper.setText(htmlMessage.toString(), true);
        javaMailSender.send(mimeMailMessage);
        return true;
    } catch (MessagingException e) {
        e.printStackTrace();
        return false;
    }
}
```

核心代码说明：

上述代码是将组织好的 HTML Segment 进行邮件发送功能。上述代码核心类为：MimeMessage，该类有所使用的 SpringBoot 邮件功能提供。

4.7 SpringBoot 统一配置

简述：在多服务模块中，所有的 SpringBoot 单个节点服务的启动都需要依赖自身定制化的项目配置文件。在开发过程中，经常需要该服务模块的功能而去定制化的配

置相关文件。但如果很多的模块以及打包部署发布在了公网上，再需要修改配置，然后停止项目运行重新启动，那会带来太多的麻烦且带来了很不好的用户体验。

为了能够在线修改配置文件，且让项目自动的感知配置文件的变化，最终要的是不需要重新运行项目是一个必须的需求。

因此项目采用 SpringCloudConfig 组件进行统一配置。

总体设计如下：

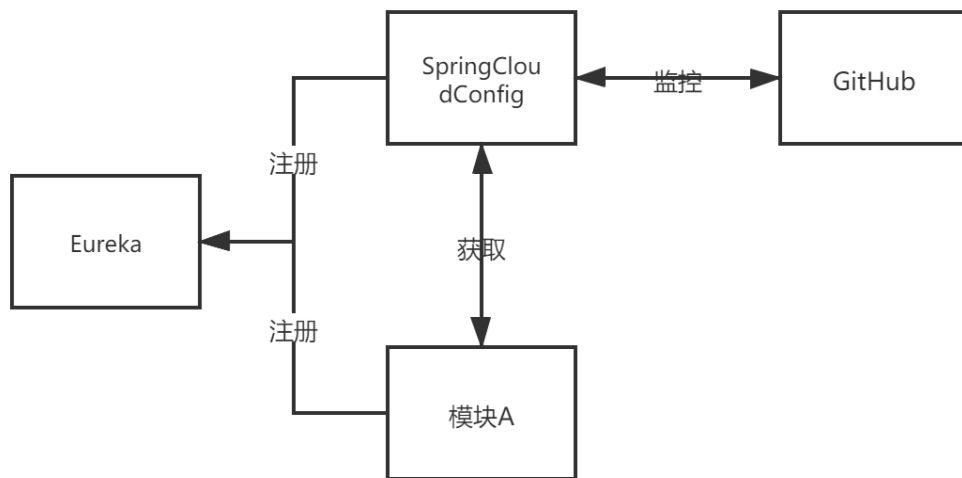


图 4-5 统一配置

核心配置：

```
spring:
  application:
    name: config-single-server
  cloud:
    config:
      server:
        git:
          uri: https://github.com/xxx/config
          username: xxx
          password: xxx
          default-label: master
          search-paths: config
```

核心配置说明：

在 SpringCloudConfig 模块中，需要配置 Github 地址，并填写远程仓库地址以及仓库分支。

仓库配置文件截图如下：

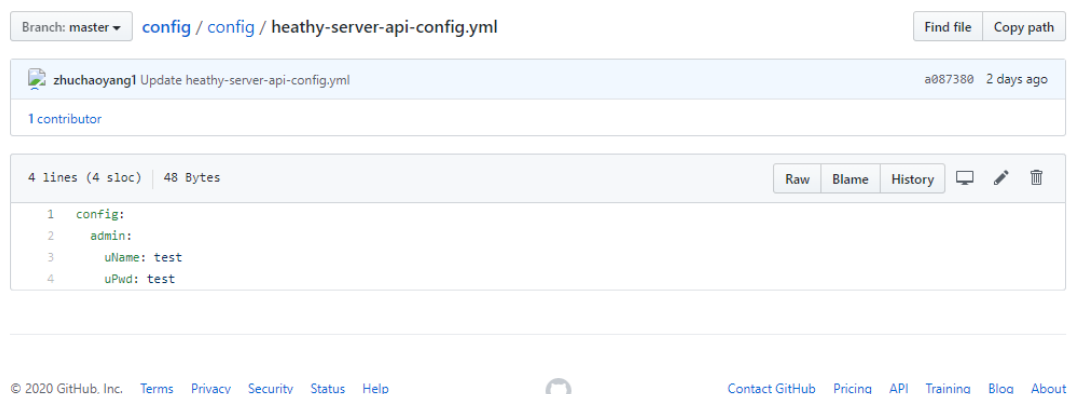


图 4-6 GitHub 截图

需要使用动态配置项目的配置：

```
spring:
  cloud:
    config:
      name: heathy-server-api # 如果不配置 name 则默认为当前服务 API, 则 Github
config: API-xxx
      label: master
      profile: config
      discovery:
        enabled: true # 开启 Eureka 路由
        service-id: config-single-server # config-server Application ID
```

上述代码需要配置 SpringCloudConfig 的服务模块，上述配置中尤其需要注意 name 的属性值，name 的属性值加上下述的 Profile 为 Github 的配置文件名称。

4.8 节点健康状态

简述：在项目运维过程中，开发人员实时的查看项目各个节点的健康状况是非常重要的，本项目中采用 Acutor 进行节点健康检查，并采用注册中心客户端获取所有的节点信息以及对应的健康信息，最终展现在运维平台中。

总体设计如下：

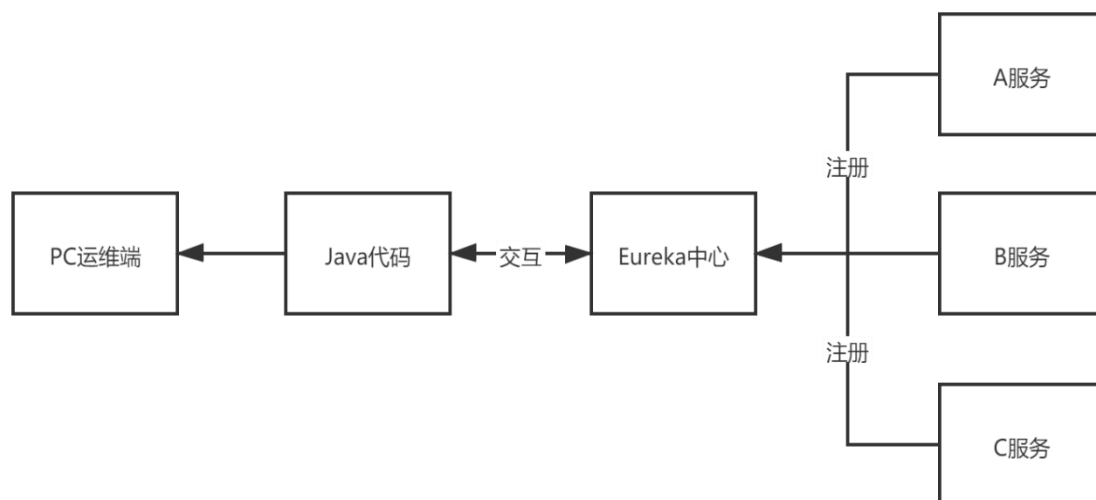


图 4-7 健康状态

核心代码：

```

@GetMapping("/get/application")
public ResultHttp getServicesList() {
    List<ServiceInstance> servicesList = new ArrayList<>();
    List<String> serviceNames = discoveryClient.getServices();
    for (String serviceName : serviceNames) { // 可能一个 appname 对应多个服务
        做负载均衡
        List<ServiceInstance> serviceInstances =
            discoveryClient.getInstances(serviceName);
        servicesList.addAll(serviceInstances);
    }
    return ResultHttp.builder().code(1).result(servicesList).build();
}
  
```

核心代码说明：

通过 Eureka 客户端 `DiscoveryClient` 获取所有的 Service, 并迭代获得的 Services, 并按照名称进行获取服务实例。至于为什么每一个 Service 都需要再使用 For 循环迭代, 这是因为存在一个 ServiceName 对应多个实例的情况, 这种情况就是负载均衡。

这样项目可以非常方便的通过注册中心获取服务实例以及对应状态, 从而避免开发服务器脚本的编写。

但是上述代码仍需要改进, 因为 Eureka 有“自我保护”机制, 即 Eureka 一段时间没有收到服务的心跳 (可能已经宕机), 那么这时候拿到服务节点的状态信息可能不是最新的, 该问题可以关闭 Eureka “自我保护”功能。

4.9 小结

本章节详细介绍了项目中的核心功能点。分别从功能点所使用的技术点、核心代码说明、功能流程图等方面进行详细阐述。其中主要对如下功能进行说明：小程序免密登录、小程序请求封装、Gateway 流量鉴权、Kafka 记录流量、邮件告警、SpringBoot 统一配置、节点健康状态。

第5章 项目测试

5.1 简述

本论文中前四章分别从设计和实际开发维度进行说明，本章对项目的若干功能选取典型功能进行测试。主要测试在一些极端的使用情况下，项目是否会出现错误。项目所有的功能能否正确走通并不报错。

首先需要声明的是在本章中所有的小程序截图右下角都有一个绿色的 Console 按钮，该按钮为小程序真机调试的按钮并非项目中因业务需要而设置。

5.2 小程序免密登录测试

用户打开小程序即进行免密登录，如果用户为第一次使用，首页按钮会提醒用户授权登录，若该用户为老用户，那么首页的按钮就为灰化无法点击的状态而直接保持为登录状态。登录之后即可看见小程序的主界面，如下图 5-1，登录之后进入主界面为图 5-2。

惜已

欢迎来到惜已小程序！

微信授权一键登录



图 5-1 免密登录

图 5-2 首页

5.3 阅读后端发布文章

在本项目中，平台运维者可以通过运维界面定期发布健康或心理的好文章，发布之后，小程序端可以立即看到系统推送的“好文”，并进行文章阅览。图 5-3 为运维界面添加文章，图 5-4 为小程序用户查阅系统推送的文章。



图 5-3 运维界面添加文章



图 5-4 文章阅读

5.4 悄悄话模块

用户登录之后选择底部“悄悄话 TabBar”，然后就可以进行悄悄话编写。悄悄话的种类很多，用户可以查看对应种类的悄悄话，且可以进行评论，亦可以进行编写，或者私信等相关功能。

下图 5-5 为编写悄悄话，图 5-6 为查看悄悄话列表，图 5-7 为查看悄悄话，图 5-8 为私信悄悄话，图 5-9 为查收私信消息，图 5-10 查看私信。

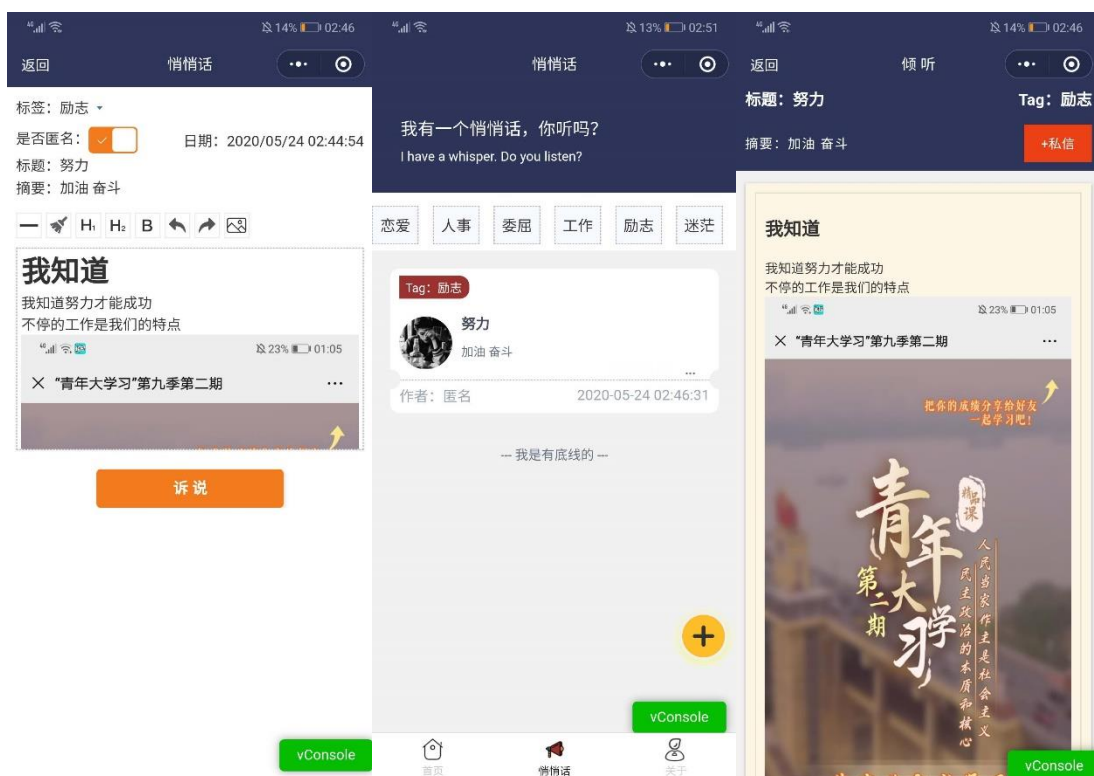


图 5-5 为编写悄悄话

图 5-6 悄悄话列表

图 5-7 查看悄悄话

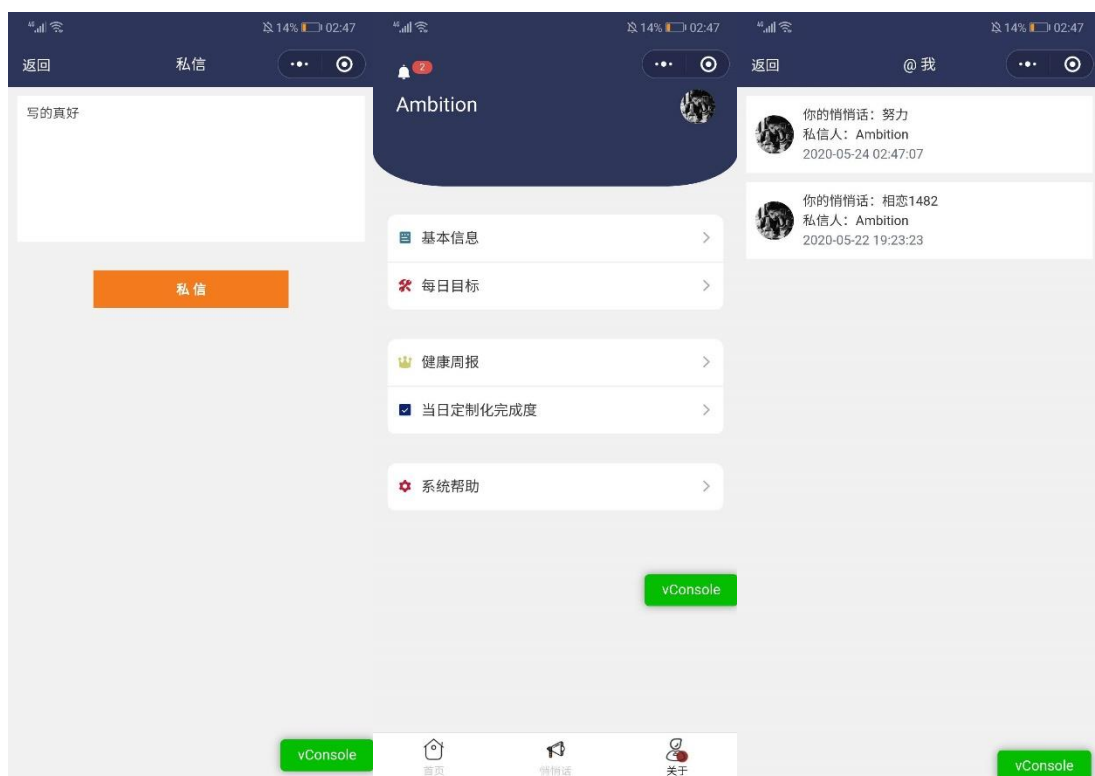


图 5-8 私信

图 5-9 查收私信消息

图 5-10 查看私信

5.5 基本信息

用户登录之后，在“我的”页面中找到“个人信息”，并可以按照不同的标准进行选择。在填写完身体报告之后就可以查看身体报告，身体报告分别从“标准体重”和“BMI”指数进行说明。除此之外还可以查看修改记录。下图 5-11 为信息参数填写，图 5-12 为身体报告查看，图 5-13 为修改记录查询。

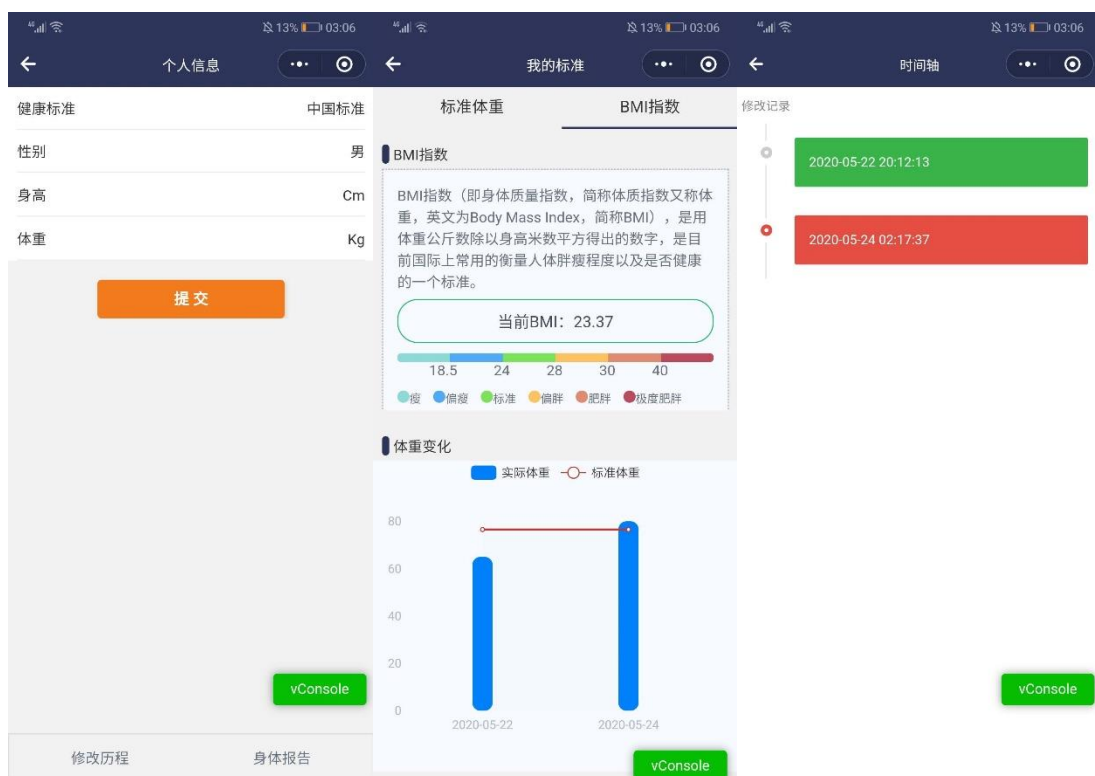


图 5-11 信息参数填写

图 5-12 身体报告查看

图 5-13 修改记录查询

5.6 每日目标

用户登录之后，打开“我的”，点击每日目标，系统会根据用户填写的最新身体参数定制化的推荐健康饮食，如下图 5-14：



图 5-14 每日任务

5.7 健康周报

用户每日填写任务进度之后，系统会根据用户所填写的进度进行百分比计算，分别以折线图和饼图的形式展示。折线图用于展示用户近几日完成任务百分比的对比。饼状图是某一天各项任务完成在总任务的占比，如下图 5-15 折线图，5-16 饼图。



图 5-16 饼状图

5.8 划任务

用户登录之后，在用户已经填写完身体参数信息之后，系统会每天为用户推荐近一阶段的健康饮食，用户每日可以对任务完成度进行反馈，如图 5-17：



图 5-17 划任务

5.9 系统帮助

在小程序端，提供“系统帮助”功能，为了帮助用户更好的了解如何使用系统，用户可以进行查看，如图：

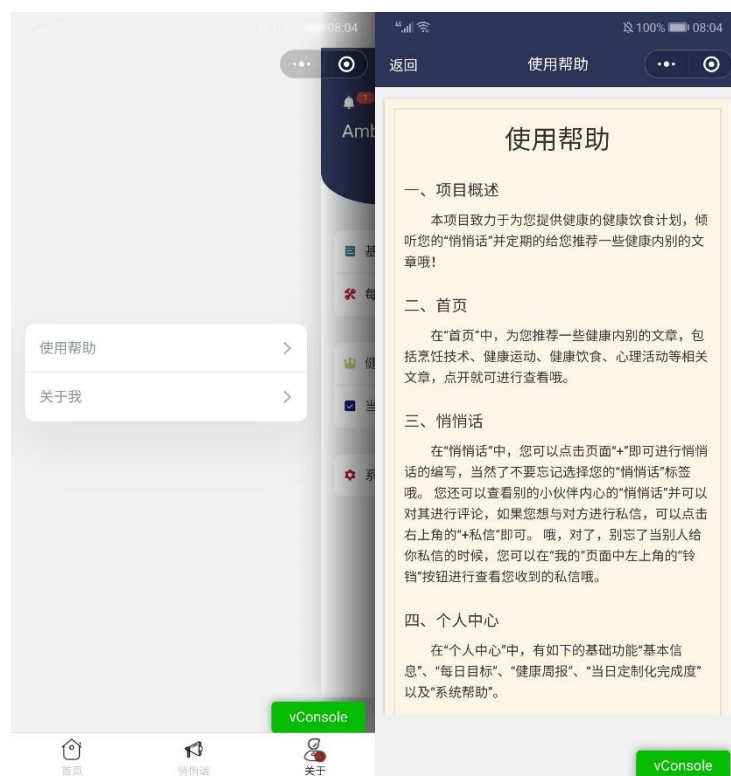


图 5-18 系统帮助

5.10 节点健康检查

项目部署之后，运维人员可以时刻查看各个节点的“健康”状况以及当前节点所在的服务器地址、端口等信息，如下图：

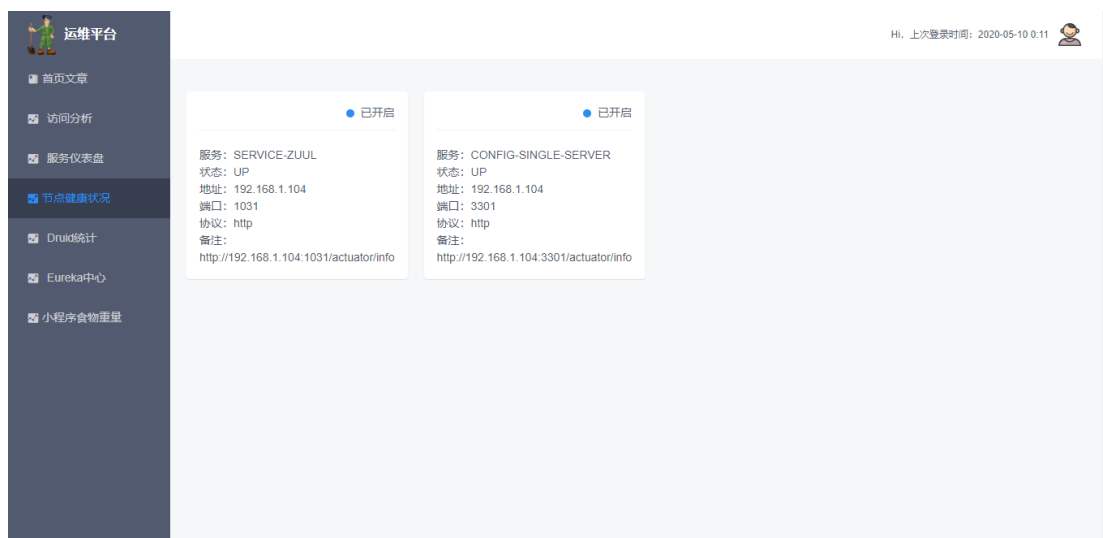


图 5-19 节点健康状况

5.11 仪表盘

项目仪表盘为 Hystrix DashBorad 技术。运维人员可以根据该技术进行查看服务器压力，如下图：

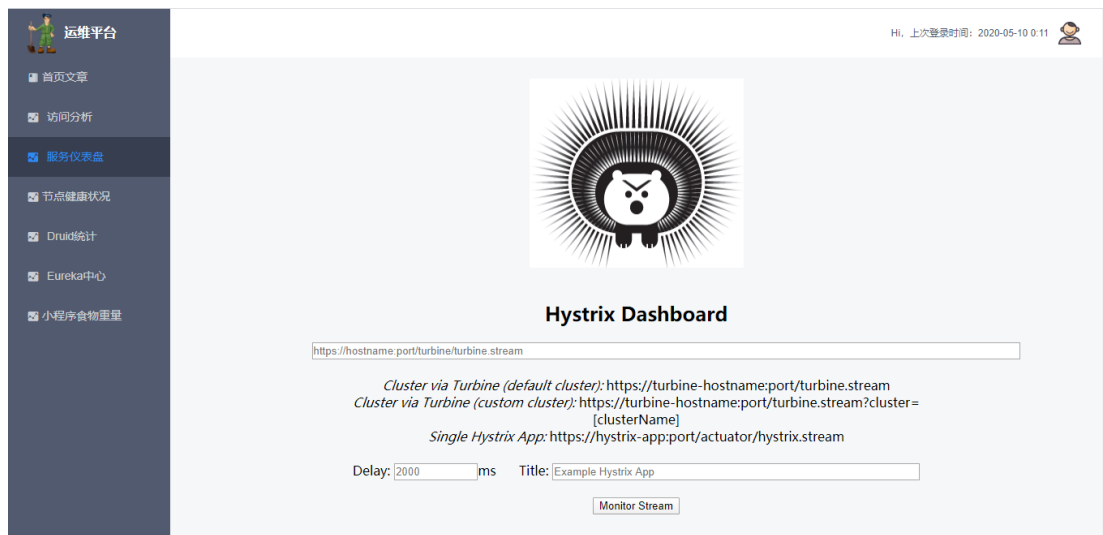


图 5-20 服务器压力仪表盘

5.12 Druid 统计

项目中使用 Druid 数据源，该数据源提供 Web 控制台，里面包括 SQL 执行的时间，以及是哪个 URL 触发了 SQL 记录，如下图：

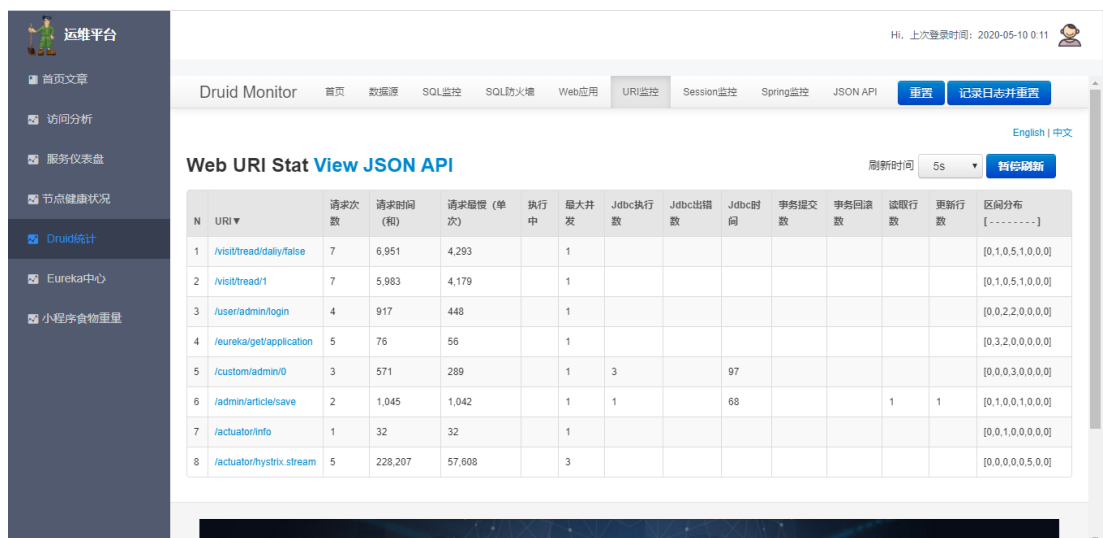


图 5-21 Druid 控制台

5.13 Eureka 中心

项目中，使用 Eureka 作为注册中心。Eureka 注册中心自带控制台，用于显示所有注册到 Eureka 的服务节点，如下图：

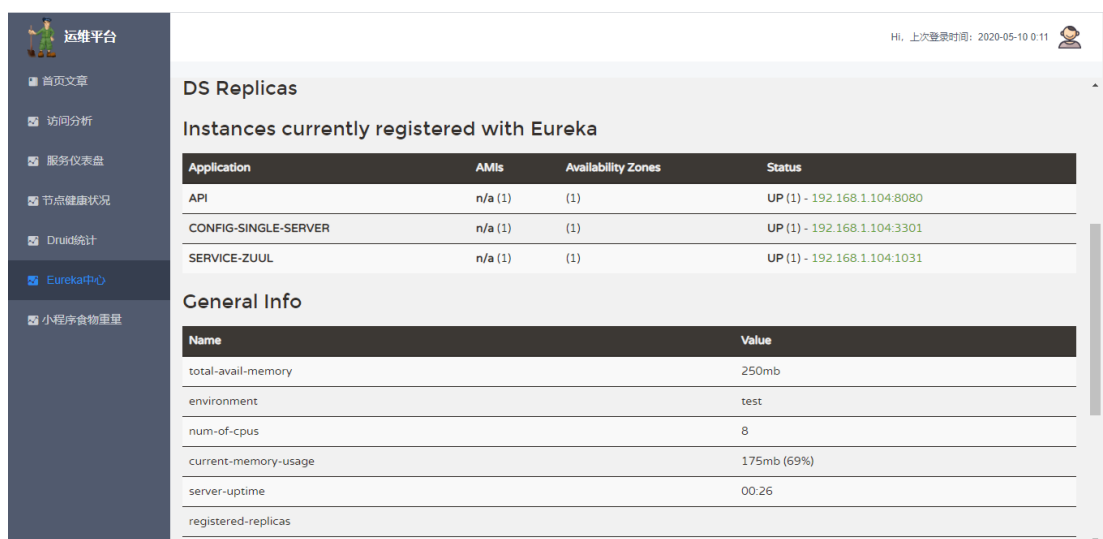


图 5-22 Eureka 控制台

5.14 食物重量定制化

项目中，需要定制化的保存不同的 BMI 体型然后进行保存，小程序端会根据用户的 BMI 指数进行动态的推荐饮食，如下图：

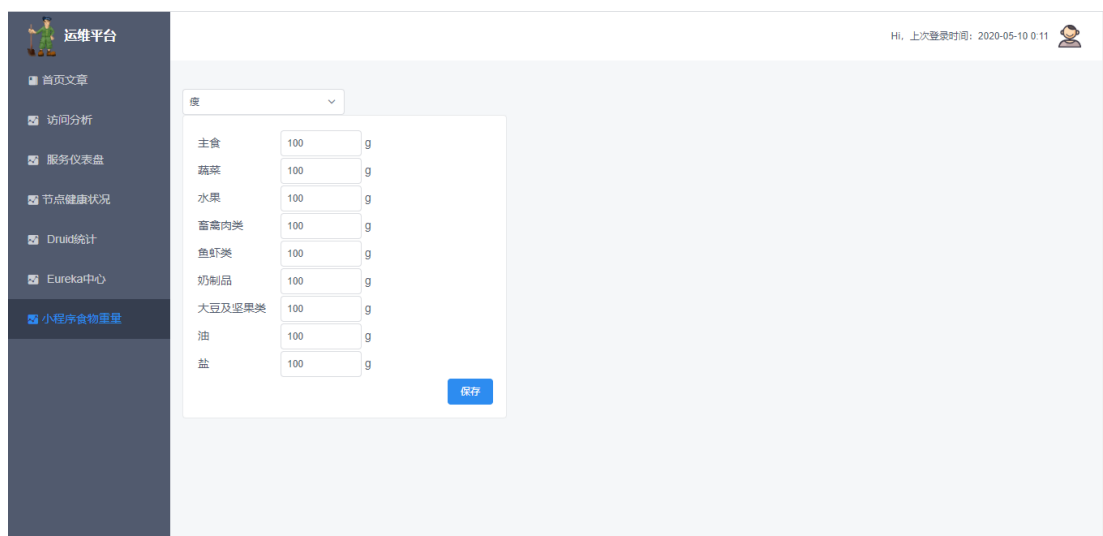


图 5-23 饮食推荐

5.15 小结

本章节主要讲述了项目测试，尽可能的覆盖项目中所有的测试功能。本章节主要对如下功能进行测试：小程序免密登录、发布文章、悄悄话、基本信息维护、每日目标、健康周报、划任务、系统帮助展示、系统节点健康检查展示、项目服务节点仪表盘、Druid SQL 统计、Eureka 微服务注册中心、食物重量定制化等功能。

第6章 项目部署

6.1 简述

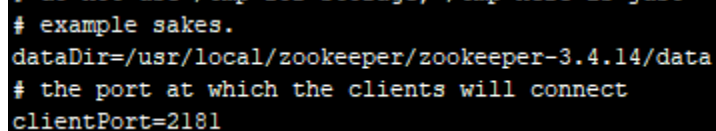
项目需要发布在服务器中，目前项目有三个角色，分别为小程序、运维 PC 端、Java 后台，其中 Java 后端的服务分别部署在两台服务器：106.14.160.110 以及 47.101.216.18 服务器。

6.2 配置 JDK

每台服务器需要搭建 JDK 环境，并配置环境变量。

6.3 配置 Zookeeper

在服务器 106.14.160.110 下载安装 Zookeeper，并修改 Zookeeper 目录下的 Zoo.cfg 信息，修改如下图：



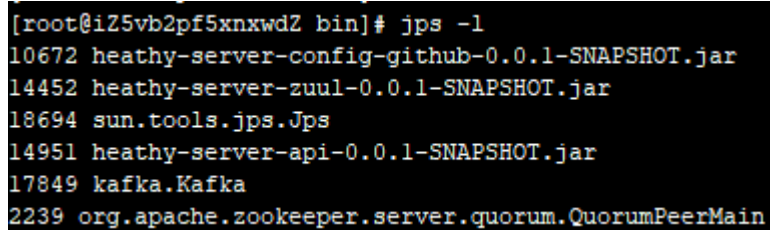
```
# example sakes.
dataDir=/usr/local/zookeeper/zookeeper-3.4.14/data
# the port at which the clients will connect
clientPort=2181
```

图 6-1 Zookeeper 配置文件修改

配置完毕之后，使用如下命令进行启动 Zookeeper（以守护进程方式）：

```
nohup ./zkServer.sh > zookeeper.out &
```

使用 Jps -l 命令验证 Zookeeper 是否启动成功，如图：



```
[root@iZ5vb2pf5xnxwdZ bin]# jps -l
10672 heathy-server-config-github-0.0.1-SNAPSHOT.jar
14452 heathy-server-zuul-0.0.1-SNAPSHOT.jar
18694 sun.tools.jps.Jps
14951 heathy-server-api-0.0.1-SNAPSHOT.jar
17849 kafka.Kafka
2239 org.apache.zookeeper.server.quorum.QuorumPeerMain
```

图 6-2 验证 Zookeeper

进程 2239 即为 Zookeeper 进程，说明启动成功。

6.4 配置 Kafka

前提：在 106 服务器确保 Zookeeper 安装且正确的运行才能保证 Kafka 的运行。

下载 Kafka，修改 Kafka 如下配置，如下图：

```
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

##### Socket Server Settings #####

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#   listeners = listener_name://host_name:port
#   EXAMPLE:
#   listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://172.19.6.253:9092
```

图 6-3 配置 Kafka

```
# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=10000
```

图 6-4 配置 Kafka

配置完毕之后，使用如下命令进行启动 Kafka（以守护进程方式）：

```
nohup ../bin/kafka-server-start.sh ./server.properties &
```

使用 Jps -l 命令验证 Kafka 是否启动成功，如图：

```
[root@iZ5vb2pf5xnxdZ bin]# jps -l
10672 heathy-server-config-github-0.0.1-SNAPSHOT.jar
14452 heathy-server-zuul-0.0.1-SNAPSHOT.jar
18694 sun.tools.jps.Jps
14951 heathy-server-api-0.0.1-SNAPSHOT.jar
17849 kafka.Kafka
2239 org.apache.zookeeper.server.quorum.QuorumPeerMain
```

图 6-5 验证 Kafka

进程 17849 即为 Kafka 进程，说明启动成功。

6.5 部署业务模块

项目使用 IDEA Maven 进行各个 SpringBoot 服务模块打成 Jar 包，并部署在两台服务器中运行。

如下为项目中服务模块的 Jar 包：

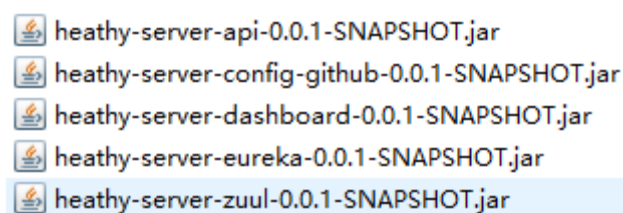


图 6-6 服务 Jar 包

其中 heathy-server-api-0.0.1-SNAPSHOT.jar、
heathy-server-config-github-0.0.1-SNAPSHOT.jar、
heathy-server-zuul-0.0.1-SNAPSHOT.jar 部署在 106.14.160.110 服务器中，启动模式格式均为如下方式：

```
nohup java -jar xxxx.jar > xxx.out &
```

剩下的服务：heathy-server-dashboard-0.0.1-SNAPSHOT.jar、
heathy-server-eureka-0.0.1-SNAPSHOT.jar 均部署在 47.101.216.18 服务器中，且启动方式与上相同。

6.6 小结

在前述所有章节中，分别对项目需求分析、技术选型、详细开发以及最后的软件测试进行全方位的说明。在系统开发完毕之后就需要对项目进行部署上线。本章节首先对配置 Java 基础环境：JDK 进行说明，由于项目中使用了 Kafka 而其依赖于 Zookeeper，因此先进行 Zookeeper 安装与启动随后对 Kafka 进行安装启动。最后在所有的基础环境得到保证之后开始部署项目业务服务。在最后进行验证。

第7章 总结

本项目采用微服务架构方式，采用 SpringCloud 一站式解决方案，所有的服务模块均采用 SpringBoot 作为服务单元进行开发。以最大程度的靠近微服务的架构思想。微服务架构的灵活性避免了传统软件的开发模式。

在最大限度上进行服务模块之间的相互解耦。也方便后期的服务扩容和负载。

项目中使用 MySQL 关系型数据库。并采用 Innodb 引擎。

在项目业务层，项目采用前后端分离的开发方式，两者之间的交互通过 Axios 或者 Wx.Request 进行交互，交互数据采用 JSON 格式数据，因为 JSON 有着轻便解析轻便等良好特点。

项目后端封装了统一应答（Response）模块，方便后端开发。后端多采用 Java 容器技术，如 Map，ArrayList,LinkedList 等相关容器。在容器之间，也做过比较，若需要对链表进行删除、查询操作那么项目中会多使用双向链表的容器模式：LinkedList，因为该中容器采用链表的方式进行扩容，而 ArrayList 本质为数据实现，当需要动态的添加或删除时，ArrayList 需要将原数据重新转移到新的内存空间。该中做法相当的耗时和浪费内存。

在整个项目开发完成之后，对微服务有了更深的认识，发现有些业务场景不能为了微服务而去微服务，因为微服务会直接导致开发成本提高，比如服务节点的内存需要更大一些，宽带需要更好一些，这样才能承载一个项目包分成多个包多个端口同时运行，相互之间还需要通过注册中心进行相互调度。且在微服务中，模块之间的错误率也会提升，若一个服务宕机可能会导致整个项目的宕机。

比如在本项目中，若网关宕机，那么会导致整个项目无法使用。

因此开发过程中必须进行取舍，对于业务场景不复杂的情况下可以简单的使用业务处理。为了微服务而微服务不是明智之举。

致 谢

首先，感谢此处的毕设项目的参与机会。能够让我在做毕设的过程之中能够学习更多，感触更多。

感谢本次毕设的指导老师：傅启明老师。在本项目开发过程中，傅老师一直孜孜不倦，不吝教导我去从多方面考虑技术问题，参与了整个毕设项目的全方位指导。在傅老师的带领下，学习到了更多的开发技巧，更重要的是开发思路。我想只有开发思路明确，目标明确，做法明确，才能非常的高效开发出毕业设计出来。

感谢大学期间所有任课的教师，在他们的引路之下，我才能迅速的在软件开发的世界中迅速的找到自我，并使我对软件开发产生了非常浓烈的兴趣爱好，我也在这爱好之中在技术中不断徜徉。再次，谢谢各位老师。

感谢电子学院为我提供了良好的学习环境和非常浓烈的学习氛围。班级同学的浓烈氛围使得我能够在安静平和的环境之下不断的提升自我，电子学院计算机实验室的计算机设备为我提供了物质需求，我可以利用这些计算机设备提高自己的 Linux 运维等相关能力，谢谢电子学院。

再次也要感谢我的大学舍友，我们就像是家人一般，在困难面前我们互帮互助，上课的问题我们课后在宿舍及时的讨论，谈天论地，聊着各种社会热门的话题，热门的技术。再此，我感谢你们。谢谢你们大学时光的陪伴。

参 考 文 献

- [1] 陈宇收,饶宏博,王英明,谷国栋,胡进贤.基于 JWT 的前后端分离程序设计研究[J].电脑编程技巧与维护,2019(09):11-12.
- [2] 韩菊茹,杨秩,纪兆轩,马存庆.基于微信小程序的文件加密系统设计与实现[J].信息网络安全,2019(09):81-85.
- [3] 曹郁.基于 Docker 容器的微服务研究与实现[J].科学技术创新,2019(28):97-98.
- [4] 李苗,李志豪,徐芸,付宝君.微信平台下的个性化学习研究[J].中国教育信息化,2019(17):38-41.
- [5] 欧阳宏基,杨铎.基于微服务架构的学位论文写作辅助平台[J].计算机与现代化,2019(10):34-39.
- [6] 万书鹏,易强,张凯,彭晖,王毅,杨明.新一代调度控制系统基于微服务架构的服务编排技术[J].电力系统自动化,2019(09):10-30.
- [7] 田子兰.浅谈 linux 系统的安全[J].电脑知识与技术,2019(30):1-2.
- [8] 夏松竹.高校本科毕业设计一体化教学管理系统的设计与实现[J].计算机教育,2008(11):61-63.
- [9] 陈亚莉.基于计算机 Web 3D 和 Java 程序开发的农机信息系统[J].农机化研究,2020,42(12):200-204.
- [10] 曾裕涵,杨立敏,刘小西.基于分布式技术的数字档案信息管理系统设计[J].计算机时代,2020(04):58-60.
- [11] 云岩.基于微服务的校园应用门户网站的设计与实现[D].黑龙江大学,2018.

附录一 译文

第 11 届 CIRP 工业产品服务系统会议 基于微服务架构的中小企业制 造云平台应用研究

周毅^{a, *}, 王梅林^a 陈仁元^b, 王阳帅^b,
王娇^b

a 广东工业大学信息工程学院
b 广东工业大学信息工程学院

*相应作者。 电话。 *+8618318881572; 电子邮件地址: 1763562637@qq.com

摘要

对于中小企业来说,智能制造执行是完善 PSS 的重要组成部分。中小企业的业务需求差异很大,系统的开发和维护面临着巨大的挑战。然而,现有的制造执行系统无法快速响应业务重构的要求。针对当前中小企业制造执行系统的要求,随着企业在云制造中的发展,本文提出了一种基于微服务架构的中小企业制造云平台应用研究. . 在本研究中,我们介绍了平台微服务体系结构的设计和基于平台的业务系统的实现。它可以解决传统单片架构的巨大应用造成的业务扩展和用户需求无法快速响应的问题。为业务系统的开发提供了有效的支持。

©2019 年作者。 由 Elsevier B.V. 出版。

由第十一届工业产品服务系统 CIRP 会议科学委员会负责的同行评审

关键词: 微服务框架; 云平台; MES; 中小企业; 云制造

1. 引言

智能制造是提高中小企业工业 PSS 的重要组成部分。制造执行系统是智能制造生产现场管理解决方案.. 制造执行系统(MES) 是中小企业智能制造升级改造到“中国制造 2025”的重要组成部分。目前, 制造执行系统技术研究需要大量的技术人员维护, 同时对信息化建设成本要求高, 成本昂贵.. 因此, 它只适用于一些大型制造企业, 而 SMME 难以实施。此外, 市场上适合中小企业的 MES 系统平台很少。

在 SSME 中实施 MES 的主要问题概述如下:

1. 不同企业之间 MES 业务模式存在较大差异;
2. 企业内部业务流程动态配置不良;
3. 企业信息技术人才的不足;

为了解决现阶段 SSME 信息系统建设中的上述问题, 本文采用微服务框架技术实现 MES。

(1) 微服务技术采用统一管理的服务注册、服务发现、消息总线、负载均衡等技术, 可以快速实现 MES 系统业务模块的解耦和组合。将高度耦合的集成系统应用程序划分为粒度较小的微服务业务模块.. 对于不同的企业业务系统, 可以通过微服务系统组装解耦的微服务业务模块, 使可以快速实现业务的实现和扩展。(2) 微服务技术构建的小粒度业务模块具有特定的功能和有限的影响范围。因此, 根据 SSME 的业务需求, 可以快速重构, 大大降低了业务重组的复杂性。(3) 微型服务业务模块可作为 Docker 映像部署。快速部署 Docker[9]可以实现云端的快速安装和实现, 可以解决制造业技术人员短缺的问题..

2. MES 研究现状

2.1 云制造, 微服务架构

中国制造信息化知名专家李伯虎提出了一种网络化服务制造新模式, 将现有的网络化制造、云计算、服务型技术、高性能计算、物联网等相关技术结合起来。他把这个模型命名为云制造。[3]随着与2012年提出的软件隔离思想相似的微服务体系结构的概念, 微服务理论不断完善, 微服务开发人员的数量不断增加。姜勇提出了基于微服务架构的基础设计[4]谭一鸣进行了基于微服务架构[5] 的高耦合平台服务框架的设计与实现.. 目前, 微服务技术在各个领域的应用需求不断增加, 出现

了一些企业项目，如制造业、电子商务等。[6]。

2.2 SOA，可配置MES

近年来，中国重庆大学刘伟宁制造实验室的研究团队对制造业执行系统领域引入服务型技术进行了深入研究.. [1]。西北工业大学制造技术团队张英峰通过结合新兴的物联网技术，提出了一种基于物联网技术的制造执行系统。 [2][7]针对制造执行系统应用的多变性和复杂性，我国华中科技大学饶云清和周伟研究了装配车间可配置MES系统，并通过工作流引擎[8]对过程模型的执行进行了指导和控制。

3. MES 微服务框架

3.1 框架系统

本文建立了一个基于微服务框架的制造执行系统。制造执行业务分为生产订单服务、过程管理服务、质量管理服务和生产过程服务。利用微服务开发模型开发各种微服务模块..不同企业可以根据生产服务的要求，配置和部署相应的微服务模块，实现企业制造业务。通过这种方式，企业可以实现个性化的生产要求。系统总体架构如图所示：

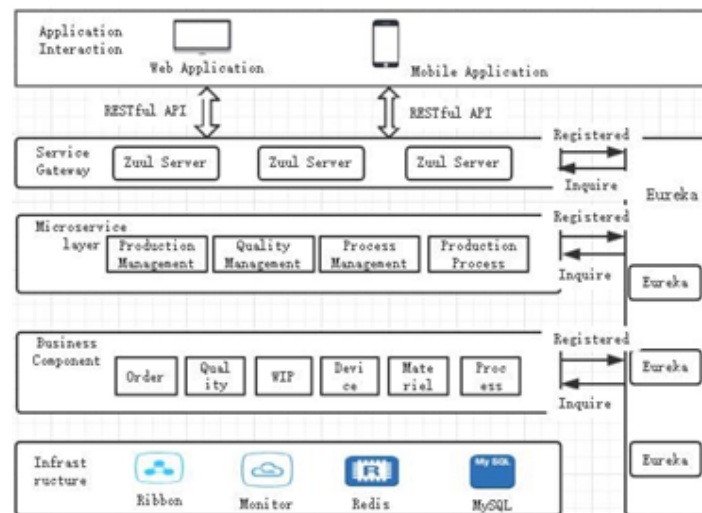


图 1. 系统框架

●应用交互层

前后端分离技术用于实现制造执行系统在Web和移动客户端上的业务操作。通过微服务层的RESTful接口，可以相互调用服务..

●微服务层.

微服务层由一些制造执行系统功能微服务组成，包括 生产管理微、质量管理、过程管理和生产过程微服务。 内置微服务技术，微服务层可以快速形成新型的微服务业务系统..

●原子业务层

原子业务组件层部署MES系统的业务逻辑组件，如物 料，质量，设备，工艺管理组件中的工作等.. 微服务启动时，根据功能模式获取原子业务组件实现，通过各种原子服务完成微服务的组合..

●基础设施层

该层主要为上层服务提供基础设施服务，包括微服务体系结构技术的一些功能组件， 以实现对微服务、Ribbon、Redis和数据库底层资源的监控。

3.2 网关实现制造业务.

在微服务框架中，采用微服务网关技术实现制造执行 系统的微服务功能，微服务网关为客户端提供统一的请求入口，负责请求路由、服务组成和协议转换。 网关实现流程图如图所示 2

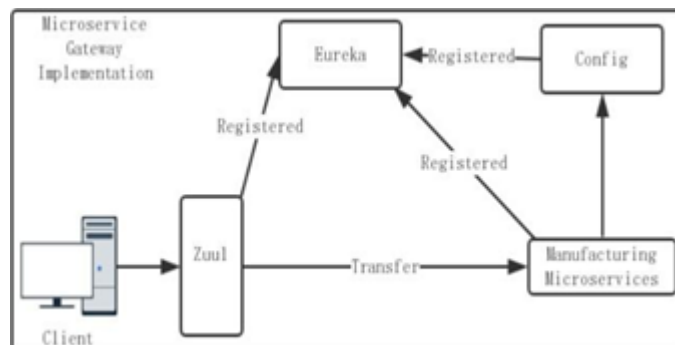


图 2 微服务网关实现过程

服务网关是系统的统一门户.. 注册中心提供统一的底层微服务API录入和注册管理.. 封装内部系统体系结构，并向客户端提供RESTAPI。 在启动过程中，微服务首先通过服务注册将其信息注册到服务注册中心.. 微服务客户端发送请求时，请求服务网关，然后服务网关读取请求数 据，从服务注册表中获取.. 相应的信息。最后，服务网关调用相应的微服务。

4. 基于微服务架构的 MES.

4.1 服务呼叫协议

在微服务架构下的制造执行系统中，每个微服务通过HTTP协议进行通信.. 生产订单服务，工程过程服务，订单调度服务，质量控制服务均通过请求参数实现信息传递.. 这种消息传递方法难以保证整个制造执行系统中各微服务之间服务请求格式的一致性.. 指定基于HTTP协议的服务调用协议来封装服务请求信息，可以统一整个制造执行.. 系统中的每个微服务请求调用格式实现了服务使用者和服务发布者消息格式的统一。

当服务调用请求发生时，通过统一的协议将请求参数 承载的消息进行封装，实现服务之间请求调用的一致性， 这是保证服务调用一致性的关键.. 本节针对这个问题提出了一个服务请求协议(SRP)。 SRP描述目标服务和目标服务所需的参数信息。

通过不同的服务名称属性和参数属性，这些属性由JSON 数据表示。 使用SRP协议，可以向服务使用者提供封装目标服务和目标服务所需参数的请求实体。

该SRP协议是统一的服务请求和服务响应协议，是为实现服务请求信息和服务响应信息格式的一致性而定制的服务通信协议.. 主要由目标服务信息和目标服务所需的参数信息组成.. 这两个部分共同构成服务请求或响应的数据协议包。 为了实现与通用数据协议格式的一致性，SRP协议还包括头和主体。 该SRP协议的各种属性如下所示： .

Element	Parent element	Type
ServiceName	Header	String
ServicePath	Header	String
OperationType	Header	String
OperationPath	Header	String
Token	Header	String
Entity	Body	Object
RequestEntity	Entity	List
ParamName	RequestEntity	String
ParamType	RequestEntity	Object
ResponseEntity	Entity	List
ResponseName	ResponseEntity	String
ResponseType	ResponseEntity	Object

图 3 SRP协议

4.2 生产业务流程实现.

在制造企业生产的总体过程中，主要的过程是接收采 购订单信息->订单发布->工艺路线->调度生产->生产执行->产品检验->产品报告。 在这个过程中，需要调用不同的服务模块来执行.. 基于微服务架构的MES系统是一个以服务为中心的车间制造信息管理系统，并根据MES的各个功能服务进行业务流程设计和优化。 例如，流程

图金属制造执行系统的生产过程如下：

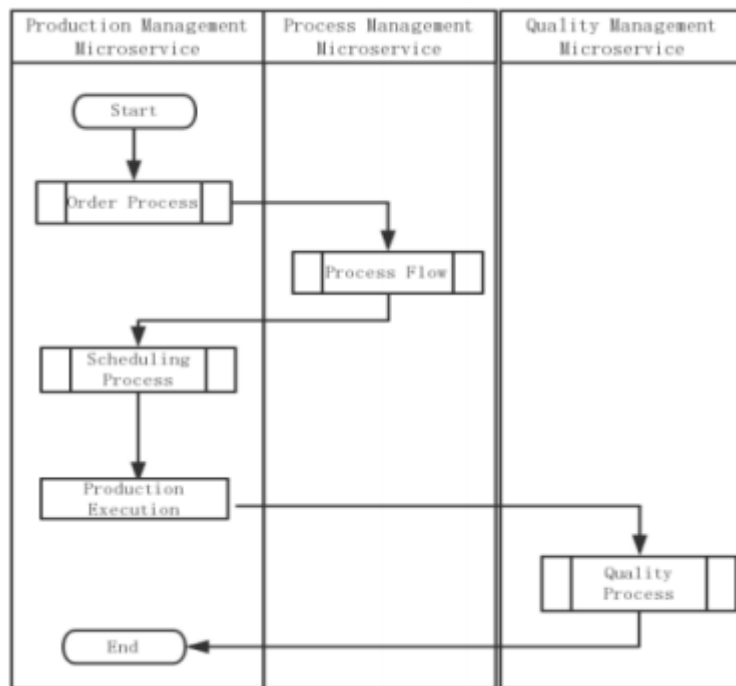


图 4 生产工艺业务流程图

生产管理是一种聚合服务。它是一个暴露于客户端的边界服务。它主要通过聚合订单服务来管理整个生产过程，流程服务、调度服务和质量检验服务。前端订单请求到达GateAPI后，调用生产管理微服务..的生产管理服务通过根据订单号调用订单服务提供的API接口生成生产订单。生产管理服务调用流程服务，根据生产订单号和订单类型生成车间工艺..生成的店铺楼层流程修改确认后，可调用调度服务发布订单作业..车间操作员通过调度服务获得具体目标操作。加工完成后，通过生产管理汇总上报质检信息。

●订购服务

订单服务主要负责生产制作订单的编辑。生产订单批次是企业订单生成后自动生成的吗从ERP系统中获得。之前的拆分操作作业在生产订单批中被释放。每一个批量生产的订单对应于一个独特的车间并对车间质量检验模板、及车间地板进行了分析车间质量检验模板对应的批次可以在生产指令作业之前编辑生产指令被释放。整个订单的接口信息服务内容如下：

表1. 订购服务

API	Parameter Type	Business Description
/order/create	OrderInfo	Create an order based on a business order
/order/craft/{id}	Integer	Get process information based on order ID
/order/craft	PlantCraftInfo	Edit workshop process information
/order/batch/{id}	Integer、BatchList	Split production orders based on batch information

- 流程服务

在流程管理业务中，首先从生产管理微应用中选择流程管理微服务和生产管理微服务来选择生产订单，通过标准流程库选择标准流程作为车间流程，在流程管理中选择每个订单批次。 制定工艺卡.. 工艺卡设置生产零件所需的一组工艺，包括所有可能的工艺。

表2 流程微服务接口业务描述

API	Parameter Type	Business Description
/craft/order/{id}	Integer	Get process information based on order ID
/craft/CraftType	PlantCraftTypeInfo	Workshop process type information
/craft/CraftEdit	PlantCraftEditInfo	Edit workshop process information
/order/CraftCheckStatus	PlantCraftCheckStatusInfo	Review process templates based on orders

- 优质服务

质量管理微服务按照标准对质量和业务流程进行控制，为生产数据处理提供平台统计，分析和查询相关信息，通过微服务架构API网关将质量报告信息反馈给生产管理微服务。

表3 高质量微服务接口业务描述

API	Parameter Type	Business Description
/Job/create{id}	VARCHAR	Get business job order information
/order/ReportQualifiedCount	Integer	Reporting on products according to quality inspection services
/order/ReportBadCount	PlantReportBadCountInfo	Handling inconsistent information

5. 结论

本文研究并实施了应用研究基于微服务架构的SMME制造云平台，划分了SMME的业务功能将制造执行系统转化为各种微服务。建立制造执行系统业务通过微服务框架，每个业务模块都是一个可以独立部署和行的单元。和明确的模块之间的边界以消息驱动RESTAPI。为使中小企业业务多元化，微服务系统是用来装配解耦的微服务实现了服务模块的快速组装服务。对于内部业务流程的可变性，小粒度业务模块组件是由microservice技术。每个小粒度组件都有影响范围小，可实现快速内部业务流程的组合。的微观服务模块被部署为一个Docker镜像来快速实现云部署和微服务安装解决中小企业技术人才短缺。

鸣谢

这项工作得到了中国国家自然科学基金（编号）的支持。U1701266，没有。第61372173号。61671163)，广东省(2014B050502014，2017KCXTD011，501130144,gdj2016004，2017b090901056，2018b030322016)，香港(编号。S/E/070/17)，广州(编号：201802020007)

参考文献

- [1] 大山 E SA, 霍斯尼 T, 萨利姆 ABM.引用该报告.MRI 脑图像分类的混合智能技术[J].[]数字信号处理, 2010, 20 (2): 433-441.
- [2] 引用该报告.洛克伍德·SA, 米勒·AJ, 克罗米·M.准备未来生物学系: 研究生高级专业发展计划[J], 美国生物学教师, 2014 年.
- [3] 格雷戈里·JK, LachmanN, C 营地 L, 等.引用该报告.核心能力基础科学课程的重组: 解剖学教学的一个例子[J].医学教师, 2009, 31 (9): 855-861.
- [4] 黄国, 洪普, 陈, 等.引用该报告.Mindtool-Assisted In-Field Learning(MAIL): 台湾先进的泛素化学习项目[J].. 教育技术与社会, 2014 年, 17 (2): 4-16.
- [5] 引用该报告.脚手架学生生成的问题: 可定制在线学习系统的设计与开发[J].计算机在人类行为中的作用, 2009, 25 (5): 1129-1138.
- [6] 乌穆拉 K, InagakiM, 郑 C, 等.引用该报告.利用中心静脉压和组织多普勒三尖瓣/二尖瓣环速预测肺毛细血管楔压的新技术[J].心脏和血管, 2014 年.

附录二 外文原文

11th CIRP Conference on Industrial
Product-Service Systems

Research on Application of SME Manufacturing Cloud Platform Based on Micro Service Architecture

Zhou Yi ^{a,*}, Wang Meilin^a, Chen RenYuan^b, Wang YangShuai^b, Wang Jiao^b

^aSchool of Information Engineering,
Guangdong University of Technology,
China

^bSchool of Information Engineering,
Guangdong University of Technology,
China

* Corresponding author. Tel.: +8618318881572; E-mail address: 1763562637@qq.com

Abstract

For SMEs, intelligent manufacturing execution is an important part of perfecting PSS. the business requirements of small and medium-sized manufacturing enterprises vary greatly, and system development and maintenance are facing enormous challenges. However, the existing manufacturing execution system can not quickly respond to the requirements of business reconfiguration. Aiming at the current requirements of manufacturing execution system of SME, and with the development of enterprise in cloud manufacturing, this paper proposes a manufacturing cloud platform application research for small and medium-sized manufacturing enterprises based on micro-service architecture. In this research we have introduced the design of platform's micro-service architecture and the implementation of platform-based business system. It can solve the problems of business expansion and user requirement cannot be quickly respond caused by the huge application of traditional monolithic architecture. It provides an effective support for the development of business systems.

© 2019 The Authors. Published by Elsevier B.V.

Peer-review under responsibility of the scientific committee of the 11th CIRP Conference on Industrial Product-Service Systems

Keywords: Microservices Framework; Cloud Platform; MES ; SME ; Cloud Manufacturing

1.Introduction

Intelligent manufacturing is an important part of improving industrial PSS for SMEs. The manufacturing execution system is an intelligent manufacturing production site management solution. Manufacturing Execution System(MES) is an important part in the process of smart manufacturing upgrading and transformation of Small and Medium-sized Manufacturing Enterprises(SMME) to "made in China 2025". At present, the manufacturing execution system technology research needs a large number of technical personnel maintain, at the same time it requires high and expensive information construction costs. So it only applies to some large manufacturing enterprises, and difficult for SMME to implement. Also, there are few MES system platforms suitable for small and medium-sized enterprises in the market.

the main problems in the implementation of MES in SSME are summarized as follows:

- There is a big difference in MES business models between different enterprises.
- Poor dynamic configuration of business processes within the enterprise.
- The shortage of enterprise information technology talents.

In order to solve the above problems in the construction of information systems for SSME at this stage, this paper adopts the micro-service framework technology to implement MES.

(1) The micro-service technology adopts unified management of service registration, service discovery, message bus, load balancing and other technologies, which can realize decoupling and combination of the business modules of MES system quickly. The highly coupled integrated system applications are divided into micro-service business modules with small granularity. For different enterprise business systems, the decoupled micro-service business modules can be assembled through the micro-service system, so that the implementation and expansion of the business can be realized quickly.

(2) The small-grained business module built by micro-service technology has the specific function and limited influence scope. Therefore, according to the business requirements of SSME it can be rapidly reconstructed, which greatly reducing the complexity of business restructuring.

(3) The micro service business module can be deployed as a Docker

image. The rapid deployment of Docker [9] can realize the rapid installation and implementation of the cloud, which can solve the problem of shortage of technical personnel in the manufacturing industry.

2.MES research status

2.1 Cloud manufacturing, microservice architecture

Li Bohu, a well-known Chinese expert on manufacturing informatization, proposes a new model of networked-oriented service manufacturing, which integrates existing networked manufacturing, cloud computing, service-oriented technology, high-performance computing, Internet of things and other related technologies. He named this model as cloud manufacturing. [3] With the concept of microservices architecture which is similar to the idea of software isolation proposed in 2012, the theory of micro services is constantly improving, and the number of developers of micro services is constantly increasing. Jiang Yong proposed the basic design based on micro-service architecture [4] Tan Yiming has carried out the design and implementation of high coupling platform service framework based on micro-service architecture [5]. At present, the application requirements of micro-service technology in various fields are increasing and some enterprise projects have emerged, such as manufacturing, e-commerce, etc. [6].

2.2 SOA, configurable MES

In recent years, the research team of Liu Weining's manufacturing laboratory at Chongqing University in China conducted in-depth research on the introduction of service- oriented technology in the field of manufacturing execution systems. [1]. Zhang Yingfeng, a manufacturing technology team of Northwestern Polytechnical University, proposed a manufacturing execution system based on the Internet of things technology by combining the emerging Internet of things technology. [2][7] Aiming at the variability and complexity of manufacturing execution system application, Rao Yunqing and Zhou Wei of Huazhong University of Science and Technology in China have studied the configurable MES system for assembly workshops, and guided and controlled the execution of process models through workflow engine [8].

3.MES microservice framework

3.1 Framework system

A manufacturing execution system based on microservice framework is built in this

paper. Manufacturing execution

business is divided into production order service, process management service, quality management service and production process service. The micro-service development model is used to develop various micro-service modules. According to the requirements of production services, different enterprises can configure and deploy corresponding micro-service modules to implement the enterprise manufacturing business. By this way, the enterprise can achieve individualized production requirements. The overall architecture of the system is shown in the Fig.1

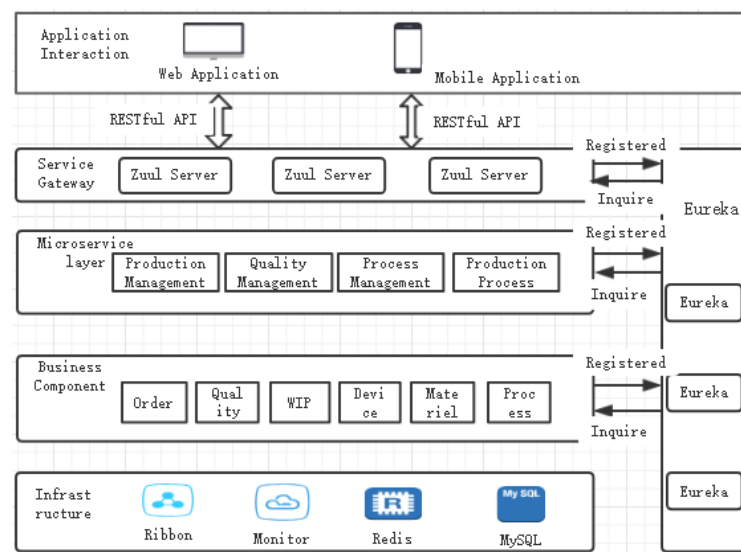


Fig. 1. System Framework

- Application interaction layer

The technology of separating the front and back ends is used to implement the business operations of the manufacturing execution system on the Web and mobile clients. Through the RESTful interface of the micro-service layer, the services can be called from each other.

- Microservice layer

The micro-service layer consists of some manufacturing execution system function micro-services, including production management micro, quality management, process management, and production process micro-services. Built with micro-service technology, the micro-service layer can quickly form a new micro-service business system.

- Atomic Business layer

The atomic business component layer deploys the business logic components of the

MES system, such as materials, quality, equipment, work in process management components, and so on. When the microservice starts, the atomic business component implementation is obtained according to the functional mode, and the combination of the microservices is completed by various atomic services.

- Infrastructure layer

This layer mainly provides infrastructure services for upper-layer services, including some functional components of micro-service architecture technology to implement monitoring of micro-service services, Ribbon, Redis, and database underlying resources.

3.2 Gateway implements manufacturing business

In the micro-service framework, the micro-service function of the manufacturing execution system is implemented by adopting the micro-service gateway technology, and the micro-service gateway provides a unified request entry for the client and is responsible for request routing, service composition, and protocol conversion. The gateway implementation flow chart is shown in Fig.2

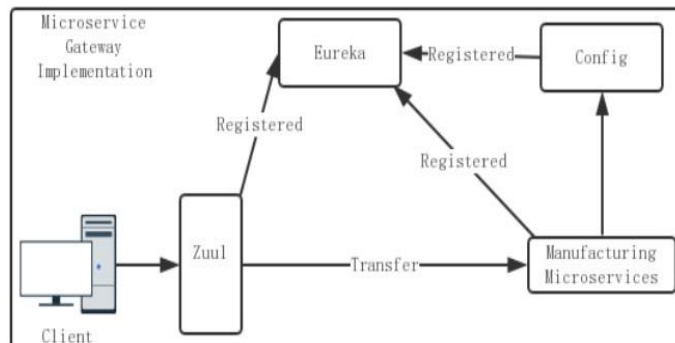


Fig. 2. Microservice Gateway Implementation Process

The service gateway is the unified portal of the system. The registration center provides a unified underlying microservice API entry and registration management. Encapsulates the internal system architecture and provides REST APIs to clients. During the startup process, the microservice first registers its information to the service registry through service registration. When the microservice client sends a request, it requests the service gateway, and then the service gateway reads the request data and obtains it from the service registry. Corresponding information. Finally, the service gateway calls the corresponding microservice.

4.MES based on microservice architecture

4.1 Service call protocol

In the manufacturing execution system under the micro- service architecture, each micro-service communicates through the HTTP protocol. The production order service, the engineering process service, the order scheduling service, and the quality control service all pass the request parameters to realize the information transmission. This kind of message delivery method is difficult to ensure the uniformity of the service request format between the microservices in the entire manufacturing execution system. A service call protocol based on the HTTP protocol is specified to encapsulate the service request information, which can unify the entire manufacturing execution. Each micro-service request invocation format in the system realizes the unification of the message format of the service consumer and the service publisher.

When the service invocation request occurs, the message carried by the request parameter is encapsulated by a unified protocol to achieve the consistency of the request call between the services, which is the key to ensure the consistency of the service call. This section proposes a Service Request Protocol (SRP) for this problem. The SRP describes the target service and the required parameter information of the target service through different service name attributes and parameter attributes, which are represented by JSON data. With the SRP protocol, a request entity that encapsulates the target service and the parameters required by the target service can be provided to the service consumer.

The SRP protocol is a unified service request and service response protocol, and is a service communication protocol customized to realize the consistency of service request information and service response information format. It mainly consists of the target service information and the parameter information required by the target service. These two parts together form a data protocol package for a service request or response. To achieve consistency with the common data protocol format, the SRP protocol also consists of a header and a body. The various attributes of the SRP protocol are shown in the following table:

Table 1. SRP Protocol Attribute ⁴⁾

Element	Parent element	Type ⁴⁾
<u>ServiceName</u>	Header	String ⁴⁾
<u>ServicePath</u>	Header	String ⁴⁾
<u>OperationType</u>	Header	String ⁴⁾
<u>OperationPath</u>	Header	String ⁴⁾
Token	Header	String ⁴⁾
Entity	Body	Object ⁴⁾
<u>RequestEntity</u>	Entity	List ⁴⁾
<u>ParamName</u>	<u>RequestEntity</u>	String ⁴⁾
<u>ParamType</u>	<u>RequestEntity</u>	Object
<u>ResponseEntity</u>	Entity	List
<u>ResponseName</u>	<u>ResponseEntity</u>	String ⁴⁾
<u>ResponseType</u>	<u>ResponseEntity</u>	Object

4.2 Production business process realization

In the overall process of production in a manufacturing enterprise, the main process is that receive the acquisition order information -> order release -> process route -> scheduling production -> production execution -> product inspection -> product report. In this process, different service modules need to be called to execute. MES system based on the micro-service architecture is a service-centered workshop manufacturing information management system, and the business process is designed and optimized according to each functional service of the MES. For example, the flow chart of the production process of metal MES is as follows:

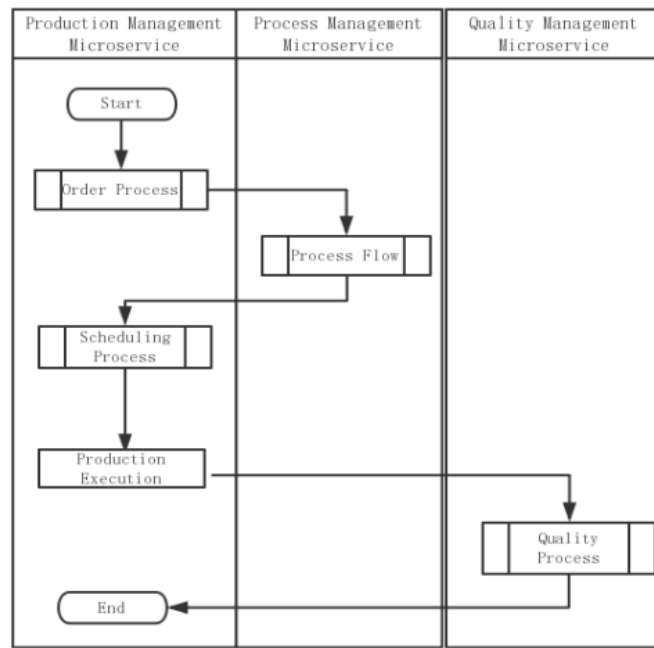


Fig. 3. Production process business flow chart

Production management is an aggregation service. It is a boundary service exposed to the client. It mainly manages the entire production process by aggregating order service, process service, scheduling service, and quality inspection service. After the front-end order request arrives at GateAPI, the production management micro-service is called. The production management service generates a production order by calling the API interface provided by the order service according to the order number. The production management service calls the process service to generate the shop floor process according to the production order number and the order type. After the generated shop floor process is modified and confirmed, the scheduling service can be called to release the order job. The workshop operator obtains the specific target operation through the scheduling service. After the processing is completed, the quality inspection information is reported through the production management aggregation service.

- Order Service

The order service is mainly responsible for the production and editing of the production order. A production order batch is automatically generated after the enterprise order is obtained from the ERP system. The split operation before the job is released is split in the production order batch. Each batch of production orders corresponds to a unique shop floor and shop quality inspection template, and the shop floor and shop quality inspection templates corresponding to the batch production order can be edited before the production

order job is released. The interface information of the entire order service is as follows:

Table 2. Order micro service interface business description

API	Parameter Type	Business Description
/order/create	OrderInfo	Create an order based on a business order
/order/craft/{id}	Integer	Get process information based on order ID
/order/craft	PlantCraftInfo	Edit workshop process information
/order/batch/{id}	Integer、BatchList	Split production orders based on batch information

• Process Service

In the process management business, the process management micro-services and production management micro-services are first selected from the production management micro-applications to select the production orders, and the standard process is selected as the workshop process through the standard process library, and each order batch is selected in the process management. Develop a craft card. The process card sets the set of processes required to produce the part, including all possible processes.

Table 3. Process microservice interface business description

API	Parameter Type	Business Description
/craft/order/{id}	Integer	Get process information based on order ID
/craft/CraftType	PlantCraftTypeInfo	Workshop process type information
/craft/CraftEdit	PlantCraftEditInfo	Edit workshop process information
/order/CraftCheckStatus	PlantCraftCheckStatusInfo	Review process templates based on orders

• Quality service

The quality management micro-service controls the quality and business process according to the criteria to provide platform statistics, analysis and query related information for

production data processing, and feeds the quality report information to the production management micro-service through the micro-service architecture API Gateway.

Table 4. Quality microservice interface business description

API	Parameter Type	Business Description
/Job/create{id}	VARCHAR	Get business job order information
/order/ReportQualifiedCount	Integer	Reporting on products according to quality inspection services
/order/ReportBadCount	PlantReportBadCountInfo	Handling inconsistent information

5.Conclusion

This paper studies and implements the application research of manufacturing cloud platform of SMME based on micro- service architecture, which divides the business functions of manufacturing execution system into various micro-services. Building the manufacturing execution system business through micro-service framework, each business module is a unit that can be deployed and run independently. And Clear boundaries are defined between modules in the form of message-driven RESTAPI . For the diversity of SME business, the microservice system is used to assemble the decoupled microservice service modules to achieve rapid assembly of services. For the variability of internal business processes, small - grained business module components are built by microservice technology. Each small - grained component has a small scope of influence and can realize the rapid combination of internal business processes. The micro-service module is deployed as a Docker image to quickly implement cloud deployment and installation of micro-services to solve the shortage of technical talents in SMEs.

Acknowledgements

This work is supported by the National Nature Science Foundation of China (no. U1701266, no. 61372173 and no. 61671163), the Guangdong Province (2014B050502014, 2017KCXTD011, 501130144, GDJ2016004,2017B090901056, 2018B030322016), Hong Kong (no.S/E/070/17), Guangzhou (No. 201802020007).

References

- [1] El-Dahshan E S A, Hosny T, Salem A B M. Hybrid intelligent techniques for MRI brain images

classification[J]. Digital Signal Processing, 2010, 20(2): 433-441.

[2] Lockwood S A, Miller A J, Cromie M M. Preparing Future Biology Faculty: An Advanced Professional Development Program for Graduate Students[J]. American Biology Teacher, 2014.

[3] Gregory J K, Lachman N, Camp C L, et al. Restructuring a basic science course for core competencies: An example from anatomy teaching[J]. Medical teacher, 2009, 31(9): 855-861.

[4] Hwang G, Hung P, Chen N, et al. Mindtool-Assisted In-Field Learning (MAIL): An Advanced Ubiquitous Learning Project in Taiwan[J]. Educational Technology & Society, 2014, 17(2):4-16.

[5] Yu F Y. Scaffolding student-generated questions: Design and development of a customizable online learning system[J]. Computers in Human Behavior, 2009, 25(5): 1129-1138.

[6] Uemura K, Inagaki M, Zheng C, et al. A novel technique to predict pulmonary capillary wedge pressure utilizing central venous pressure and tissue Doppler tricuspid/mitral annular velocities[J]. Heart & Vessels, 2014.