，出'chu﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽﷽2017届硕士研究生学位论文

分 类 号： 学校代码： 10269

密 级： 学 号： 51151500127



**基于CAN-Ethernet的汽车混合网关的研究与实现**

院 系： 计算机科学与软件工程学院

专 业： 软件工程

研究方向： 嵌入式系统设计与开发

指导教师： 王江涛 高级工程师

学位申请人： 唐益军

2017年9月

2017 Master Thesis School Code: 10269

Student Number: 51151500127



**String-Oriented Test Input Gereration**

**for Android Apps**

Department: School of Computer Science and Software Engineering

Major: Software Engineering

Subject: Embedded system design and development

Tutors: Jiangtao Wang. Senior Engineer

Author: Yijun Tang

2017.9

**摘 要**

教育是民族振兴的基石，是提高人民综合素质、促进人的全面发展的重要途径。我国作为一个人口大国，竞争压力之大不言而喻，为了应对学生对教育资源的更高需求，以及合理利用闲散教育资源，家教这一行业也应运而生。而随着我国对教育的重视，大学教育的普及程度越来越高，更多的人可以接受大学教育，并且大学生自由时间相对较多，也可以通过家教来锻炼自己，所以大学生也成为家教老师的主要来源，这其中以师范类大学更为明显。

伴随着互联网的高速发展，教育也需要与时俱进，摆脱之前的家长到家教机构登记，家教机构帮忙寻找家教资源的繁琐的线下模式。转而通过线上的智能推荐、双向互选来高效的实现教育资源的分配。

本文的主要研究目标是实现一个基于微信公众号的供需对接平台，并以家教匹配为例。通过微服务架构实现多个业务模块之间的协同运作，并将各个微服务部署在Docker容器中。同时解决微服务层面如何提供权限控制，服务发现以及服务分发，并且实现基于用户和基于内容的协同过滤推荐算法，并对其进行优化，从而为用户有针对性的进行推荐，以此来提高我们系统的匹配效率和用户体验。

系统整体采用微服务架构，各个微服务应用子模块采用Spring Boot框架进行开发，并将其部署到不同的Docker容器中，各个模块启动后首先到配置中心获取自己的配置文件，然后到注册中心进行服务注册，最后通过Gateway进行各微服务的路由映射，统一调度。同时采用Oauth2实现用户身份认证和权限校验，通过ActiveMQ消息队列来进行各模块之间的通信，实现各模块之间的消息传递，服务间的解耦合。推荐算法的优化也是本文研究的重点，实现协同过滤推荐算法并结合项目实际进行参数调整，从而提高匹配成功率。

**关键词：**微服务架构；推荐算法；Spring Boot；Docker；家教匹配

**Abstract**

Education is the cornerstone of national revitalization, is to improve the overall quality of the people, to promote the comprehensive development of an important way. China as a large population, competitive pressure is self-evident, in order to cope with the higher demand for educational resources, as well as the rational use of idle educational resources, tutor this industry also came into being. With the importance of education in our country, the popularity of university education is getting higher and higher, more people can accept university education, and college students are relatively free time, but also through tutoring to exercise their own, so college students have become tutor The main source, which is more obvious to the normal university.

With the rapid development of the Internet, education also need to advance with the times, get rid of the parents to the tutor before the registration, tutoring institutions to help find the tedious resources of the tricks of the subtle model. Turn through the intelligent recommendation on the line, two-way mutual election to achieve the efficient allocation of educational resources.

The main research goal of this paper is to realize a supply and demand docking platform based on WeChat public number, and to match the tutor as an example. Through the micro-service architecture to achieve multiple business modules between the co-operation, and the various micro-services deployed in the Docker container. It also addresses how the micro-service level provides access control, service discovery, and service distribution, and enables user-based and content-based collaborative filtering algorithms to be optimized and optimized for users. System matching efficiency and user experience.

The system uses the micro service architecture as a whole. Each sub-service application sub-module is developed by using the Spring Boot framework and deployed to different Docker containers. After the modules are started, they will first get their configuration files to the configuration center and then go to the registration center Service registration, and finally through the Gateway for each micro-service routing mapping, unified scheduling. At the same time using Oauth2 to achieve user authentication and permissions verification, through the ActiveMQ message queue to communicate between the modules to achieve the message between the modules, the decoupling between services. The optimization of the proposed algorithm is also the focus of this paper, to achieve the collaborative filtering; recommendation algorithm and adjust the parameters with the actual project to improve the matching success rate.

**Key words:**  Microservice architecture ; Recommended algorithm ; Spring Boot;

Docker; Tutor match

目 录

[**摘 要** 2](#_Toc494051513)

[**Abstract** 3](#_Toc494051514)

[**1.** **绪论** 7](#_Toc494051515)

[1.1 选题的背景及意义 7](#_Toc494051516)

[1.2 国内外研究现状 8](#_Toc494051517)

[1.2.1 微服务架构的研究现状 8](#_Toc494051518)

[1.2.2 推荐算法的研究现状 9](#_Toc494051519)

[1.3 论文的主要研究内容 10](#_Toc494051520)

[1.4 论文的组织结构 11](#_Toc494051521)

[**2.** **相关技术理论** 13](#_Toc494051522)

[2.1 前端开发相关技术 13](#_Toc494051523)

[2.1.1 AngularJS框架介绍 13](#_Toc494051525)

[2.1.2 React技术介绍 15](#_Toc494051526)

[2.2 Spring Boot框架简介 16](#_Toc494051527)

[2.3 微服务架构简介 18](#_Toc494051528)

[2.4 推荐系统相关介绍 21](#_Toc494051529)

[2.4.1 推荐系统定义 21](#_Toc494051530)

[2.4.2 推荐算法 22](#_Toc494051531)

[2.5 Docker技术简介 25](#_Toc494051532)

[2.6 本章小结 27](#_Toc494051533)

[**3** **需求分析** 28](#_Toc494051534)

[3.1 功能性需求分析 28](#_Toc494051535)

[3.1.1 客户端功能性需求分析 29](#_Toc494051536)

[3.1.2 Web管理端功能性需求分析 35](#_Toc494051537)

[3.2 非功能性需求分析 38](#_Toc494051538)

[3.3 系统环境要求 39](#_Toc494051539)

[3.4 本章小结 39](#_Toc494051540)

[**4** **基于协同过滤的推荐算法** 40](#_Toc494051541)

[4.1. 用户偏好的收集与表示 41](#_Toc494051542)

[4.2. 相似度计算 43](#_Toc494051543)

[4.3. 寻找相似近邻 46](#_Toc494051544)

[4.4. 生成推荐结果 47](#_Toc494051545)

[4.5. 本章小结 48](#_Toc494051546)

[**5** **微服务架构系统设计与实现** 49](#_Toc494051547)

[5.1. 系统设计 49](#_Toc494051548)

[5.1.1 系统总体结构设计 49](#_Toc494051549)

[5.1.2 系统详细设计 51](#_Toc494051550)

[5.2. 系统总体实现 54](#_Toc494051551)

[5.2.1 Docker环境搭建 54](#_Toc494051552)

[5.2.2 服务配置中心搭建 55](#_Toc494051553)

[5.2.3 服务注册与服务发现 57](#_Toc494051554)

[5.2.4 服务网关的实现 58](#_Toc494051555)

[5.3. 具体业务服务模块实现 59](#_Toc494051556)

[5.3.1 业务基本数据库设计 59](#_Toc494051557)

[5.3.2 业务主要接口设计 60](#_Toc494051558)

[5.3.3 功能实现 61](#_Toc494051559)

[5.4. 本章小结 62](#_Toc494051560)

[**6** **系统测试与结论** 63](#_Toc494051561)

[6.1 系统测试 63](#_Toc494051562)

[6.1.1 功能测试 63](#_Toc494051563)

[6.1.2 用户界面测试 64](#_Toc494051564)

[6.1.3 安全性测试 65](#_Toc494051565)

[6.2 系统测试结论 66](#_Toc494051566)

[**7** **总结与展望** 67](#_Toc494051567)

[7.1. 总结 67](#_Toc494051568)

[7.2. 展望 67](#_Toc494051569)

1. **绪论**

* 1. 选题的背景及意义

教育是提高人民综合素质、促进人的全面发展的重要途径，是民族振兴、社会进步的重要基石。随着互联网技术的不断发展，社会文明的日益进步，人们对于教育的重视程度也不断提高。“百年大计，教育为本”，教育在社会发展中的重要作用也是不言而喻的。而我国，作为一个人口大国，竞争日趋激烈，如何将其转变为人力资源强国，成为社会经济发展的推动力量，教育正是其中的关键。只有把教育搞上去，才能化人口压力为人才优势，才能从根本上提高我国的综合国力，才能在激烈的竞争中取得战略主动地位。

当前社会，互联网技术飞速发展，科学技术日新月异，知识更新节奏越来越快，本领恐慌体现在生活中的方方面面，因此对国民素质的要求越来越高。在这个信息现代化、经济全球化的新世纪阶段，如果没有较强的学习能力，就跟不上新形势，导致出现一批新的英语盲、电脑盲从而不能更好的适应这个社会。只有学习新知，探索未知才能提高人的现代化素质和能力，成为适应现代化社会的人。而学习是从小开始，并保持持续不断地终身学习。但是一些孩子在学校不爱学习，学不到新知识，而学校大多是少数的教师负责大多数的学生，教师限于精力不能全面照顾到所有的学生，这时候就需要家教的出现，可以对学生进行更多时间的有针对性的指导。也存在很大部分的一些家长不愿让自己孩子输在起跑线上，除了正常的学校教育学习，也希望自己的孩子可以利用额外的时间学习新的东西，掌握不同的技能，以增强自己的竞争优势。这种情况下家教的重要性就不言而喻了。

近些年来随着我国高等教育的大力扶持，越来越多的学生有机会接受大学教育，使得大学生这个群体日益壮大。大学生群体除了自己的学业之外，自由支配时间相对而言较为充足，而大学生家教则是对这一资源的合理充分利用。相对于学校教师对学生的教育，大学生家教属于精品班教育，大多是一对一或一对二教育，家教老师有足够的精力因材施教，指导学生学习方式上的不足，帮助他们走出学习上的误区，让学生更加的热爱学习。同时因为大学生家教和学生年龄差距不大，不存在交流上的代沟，学习之外也可以帮助学生解决生活上的疑惑，他们的建议也通常更容易被接受。大学生家教可以帮助学生身心健康的全面发展，在提高学习成绩的基础上找到自己的良师益友，提高学习的积极性。而大学生也可以通过做家教锻炼自己，提高自己的与人沟通表达能力，为毕业后适应社会打下基础。

通过对上海几所高校的大学生家教现状调查发现，现在的大学生从事家教主要通过大学生社团或者高校的勤工助学部门，由在校学生登记注册表明自己的家教意愿，社团或勤工助学部门面向社会收集需要家教的信息，然后推荐登记注册的大学生去进行家教。这样的供需对接模式显得较为原始，需要耗费较多的人力，并且家教匹配度也得不到保障，经常会出现家教老师上门试讲，并不是家长所需要的类型，导致家教对接的失败。同时这种供需对接也存在需求方没有实名认证，大学生家教得不到安全保障，大学生家教经历没有较好的记录，导致家教群体能力参差不齐，不能较好的区分等多种不足。这种供需对接模式也与我们当前的互联网思维显得格格不入。

本文所讲的基于推荐算法的微服务架构供需对接平台就是用来解决这一问题。整个系统采用微服务架构，多个功能模块分离开发，并部署到不同的Docker容器，通过gatway进行各模块的路由映射，统一调度。利用微服务架构中的服务发现和服务分发理念，使各个模块协同工作，并对已有推荐算法进行优化。该系统分为用户使用的移动客户端和管理人员使用的web管理端，用户通过应用注册登录并实名认证，家长可通过客户端发布家教信息，后台管理人员审核通过后予以发布，大学生家教老师可通过客户端浏览所有的家教信息，根据信息描述选择最适合自己的家教。每份家教可有多名家教老师竞聘，家长通过查看家教老师信息挑选最适合自己孩子的家教老师从而实现双向互选。展示给家教老师的信息是系统后台通过推荐算法得到的和该教师匹配度最高的家教信息，从而帮家教老师减少无效信息的干扰。当家长的某份订单长期没人竞聘时，系统会主动像家长推荐比较适合的并且当前状态为可聘请状态的家教老师，避免订单因长期处于不活跃状态而被家教老师查看到的可能性降低。

* 1. 国内外研究现状
  2. 微服务架构的研究现状

微服务架构是一种新的架构模式，在其出现之前，网站服务、接口服务等都是“一站式”的。应用所涉及到的所有模块和接口都包含在同一个服务应用中，所有的功能、接口统一设计，分层实现然后逐层的开发。传统的开发模式中，一个完整的系统应用至少包含表示层、业务层、服务层、数据访问层和数据库这5层。这5层中的所有模块相互耦合、互相依赖构成一个完整的整体，若有任何一个模块服务发生宕机或者需要重新编译则会导致应用整体不能使用，影响整个站点的部署。

针对上述提到的问题，20世纪90年代末，SOA（Service Oriented Architecture）面向服务架构被提出，首次提出了在软件架构设计的时候，使用耦合度较低的面向服务思想。SOA阐述了“对于复杂的企业IT系统，应按照不同的、可重用的粒度划分，将功能相关的一组功能提供者组织在一起为消费者服务”，系统核心是业务逻辑，由定义服务、域对象和事件的模块完成。围绕的核心是与外界进行数据交换的适配器。尽管也是模块化逻辑，但它最终还是会被打包并部署成单体式应用，具体格式依赖于应用的框架和语言。单体式应用越大，启动时间越久，如果开发者需要经常重启应用的话，那无疑会造成大量时间的浪费，使得开发效率降低，影响开发者心情。这样的应用虽然易于调试和部署，但是随着应用的不断完善，新功能的迭代开发，整个应用会变得越来越庞大，以致于单个开发者不可能完全搞懂它，修正bug和添加新功能将会变得非常困难，敏捷开发和部署也会变得举步维艰。同时在扩展性方面，由于传统应用大多是单一部署，而应用内部有的模块是I/O密集型，有的模块是CPU密集型，但他们都被部署在同一硬件服务器上，很难用单一硬件来满足所有模块对自己所需资源的要求。这样会导致应用无法应对多用户、高并发的使用场景。

现在很多公司，比如Amazon、eBay和NetFlix,通过采用微服务架构的模式来解决现有应用架构对大型且需要不断迭代开发的应用所造成的弊端。其思路不是开发一个完整的巨大的单体式应用，而是将应用分解为小的、互相连接的微服务。一个微服务一般用来完成某个特定的功能，而每一个微服务也都满足传统得六角形架构，都有自己的业务逻辑和适配器。其中一些服务还会暴露自己的API给其他微服务模块或应用客户端使用。

2014年作为标准化、可移植容器的Docker出现了，Docker因其可移植性，以及高速的容器交付速度使其短时间内流行起来，所有开发者几乎都在学习Docker相关的知识。而微服务架构的一个重点就是所有的微服务模块都需要部署在单独的容器中，同一台服务器上部署在虚拟机中的开销太大，Docker的出现无疑弥补了为微服务的这一不足，每一个微服务都部署在单独的Docker容器中。目前微服务架构只在中大型企业中得到实践，因为微服务架构带来的种种优势，相信不久之后也会被中小型企业接受，会给相关行业带来不只是软件开发管理上的影响。

* 1. 推荐算法的研究现状

随着互联网的高速发展，社交媒体越来越流行，在从中获得大量信息的同时，也给我们带来了严重的“信息过载”的问题。近年来，物联网、云计算、大数据和社会网络等新名词频繁的出现在我们的生活中，随着这些技术的迅猛发展，网络中所蕴含的信息量也成指数级增长。据国际数据公司IDC(Int’l data corporation)2012年报告显示：预计到2020年，全球数据总量将达到35.2ZB，会是2011年的22倍。

信息化社会，用户对信息的获取主要分为两个模式。第一种称之为“拉”模式，比较典型的就是搜索引擎，用户提交查询，系统返回搜索结果，这种模式的前提是用户对自己想要获取的信息有一个明确的认识，定向的获取；第二种称之为“推”模式，比较典型的就是类似于今日头条等新闻推荐APP，用户不需要提交任何查询信息，系统会自动化向用户进行信息推送，如何推送用户想要的信息，推荐算法在其中就起着至关重要的作用，这样也会给用户更好的应用体验。推荐系统也正是在这样的背景下提出来的。

推荐系统是一个非常大的领域，在各种系统中得到了广泛的应用，包括电商推荐、内容推荐（内容还分为很多的类别），而推荐算法则是其中最核心的技术点。目前，推荐系统主要应用于在线电子商务（如淘宝、京东、Amazon等），新闻热点推荐（如网易新闻、今日头条、豆瓣等），生活服务（如旅游、外卖、大众点评等）等各个领域。针对不同的推荐系统选择不同的推荐算法也是至关重要的。目前已经有很多种的推荐算法可供选择，每一种推荐算法都有其优点和缺点，当然也有自己的限制条件和其最适应的使用环境。最常见的推荐算法可以大致的分为三大类：基于内容的推荐算法、协同过滤算法以及混合推荐算法。这些传统推荐算法主要考虑的是用户和内容之间的关系，根据不同的标准的评分等级来进行推荐。

近年来因为机器学习技术的日益成熟，以及对各种复杂特征的利用方式也不断完善。现在国内比较厉害的以推荐为主要技术的公司都在广泛使用机器学习来构建推荐系统，传统的协同过滤等基于规则或者群体智能的推荐方法在逐渐被淘汰。随着用户对个性化推荐的要求逐渐增高，深度学习也进入了发展的黄金期，并在文本领域、图像领域、语音领域等取得了重要进展，深度学习其本质上是对数据特征进行深层次的抽象挖掘，通过大规模数据来学习有效的特征表示以及复杂映射机制，从而建立有效的数据模型。深度学习相较于协同过滤是对数据进行一个更深层次的分析，从而计算出的推荐目标也将更有针对性。像Google这类的公司现在已经把所有涉及到机器学习方面的应用从浅层模型转换到用深度学习重新实现，这也是未来几年的主要发展趋势。

* 1. 论文的主要研究内容

本文的主要研究目的是将互联网思维应用于大学生家教供需对接，并结合当前比较热门的微服务架构思想，改变已有系统架构模式，并结合推荐算法来实现一个各模块松耦合、可独立开发、独立部署的家教供需对接平台。本文将对当前比较主流的一些前端框架进行对比，选择比较适合系统的框架进行前端开发，良好的前端交互对提高用户体验是非常重要的，也是一个完整系统所必不可少的一环。也会详细论述微服务架构对大型系统应用所带来的优势，并完整介绍微服务架构搭建的全部过程，对相关模块进行详细分析与解读。最后还会对当前常见推荐算法进行介绍，并对最适合本系统的推荐算法结合供需对接需求进行优化，实现对用户的最优推荐，提高用户使用体验。

论文的主要研究内容包括以下几个方面：

1. 分析家教对接的主要功能需求，设计合理的用户交互流程。
2. 前端技术研究，主流框架的对比，选择最合适的前端开发框架。
3. 划分系统主要功能模块，并进行各模块数据库设计。
4. 对微服务架构和Docker相关技术进行分析和研究，并应用到系统中。
5. 对推荐算法相关技术进行分析和研究，并针对供需对接进行优化。
6. 对系统整体功能进行测试，投入使用并不断完善既有功能。
   1. 论文的组织结构

本论文总共分为七章，具体组织结构安排如下。

第一章：绪论。主要介绍了本文选题的背景以及研究意义，并对国内外研究现状进行分析，其中主要包括本论文的两个核心点即微服务架构和推荐算法的研究现状介绍。接着对本论文的主要研究内容进行一个总体介绍，包括整个研究过程中不同阶段的不同着重点。并给出了本论文各个章节的组织结构。

第二章：相关技术理论。主要介绍了了本论文中所涉及到的所有相关技术。包括前端主流框架如Framework7、AngularJs和React的核心技术以及优缺点介绍。后端开发用到的Spring Boot框架技术的介绍，以及微服务架构和微服务架构部署所用到的Docker相关技术的介绍。最后则是对常见推荐算法的介绍，并分析其所适合的不同场景。

第三章：系统需求分析。主要对该供需对接系统进行需求分析，包括系统的可行性分析，用户的具体需求分析，以及功能需求和相关数据库设计分析。即本章是对系统整体框架的一个了解。

第四章：推荐算法的模型构建及优化。本章主要对本系统所要用的推荐算法进行模型搭建，并通过虚拟数据对算法模型进行测试，根据实验结果不断对模型进行优化改进。

第五章：微服务架构系统设计。本章主要是微服务架构系统的总体设计，包括系统的总体功能结构，系统开发测试环境，以及微服务对各个模块的统一协调管理。具体的服务注册、服务发现等。以及对各个功能模块的详细介绍。

第六章：系统测试与性能分析。本章首先介绍系统测试的目的和意义以及系统测试的完整过程，进而对系统整体的一个测试，对系统性能进行分析，检测是否达到设计要求。

第七章：总结与展望。本章主要是对前面章节所完成的工作进行一个总结，对所实现的成果给予肯定以及存在的问题进行分析和研究。最后对系统以及系统所用到的技术的发展进行展望。

1. **相关技术理论**
   1. 前端开发相关技术

本文所述系统是基于微服务架构的以“互联网+教育”为理念的家教对接平台。用户主体分为三类，包括家教的需求方即需要家教服务的家长用户，家教的提供方即有能力有时间可以提供家教服务的在校大学生以及管理整个系统的运营方。

系统通过微信公众号提供统一入口，采用H5页面嵌入形式。用户不需要下载专门的应用，只需要关注微信公众号便可进入使用。通过对平台的需求进行分析，确定该供需对接平台主要分为两部分，即提供给用户使用的移动端应用，使得用户可以在移动端进行身份认证，需求发布以及匹配需求和供运营方使用的web后台管理端，管理人员可通过web管理端进行用户、需求的审核、发布、匹配等一系列流程，以及用户、需求信息的管理。要实现整个应用，在根据需求进行原型设计后，选取一个合适的前端框架进行页面开发，变成了首要任务。下面我们就针对当前比较主流的的一些前端框架进行简单介绍并对比评测其优缺点，从而选取最适合我们的开发框架。

1. * 1. AngularJS框架介绍

AngularJS是一个基于JavaScript的开源MVC框架，致力于MVC代码解耦，采用model、controller以及view方式去组织代码，由Google在2009年首次发布。最适合开发客户端的单页面应用，它不是一个功能库，而是用来开发动态网页的框架，专注于扩张HTML的功能，提供动态数据绑定，让我们能更清楚、简洁地构建我们的应用组件。

AngularJS是一个非常适合编写大型应用的JS框架，它采用单页面动态应用程序开发模式其目的就是为了简化开发过程以及测试的复杂度。开发人员在web应用的开发过程中可以实时的看到代码所产生的效果，这样使得大规模开发时程序依然可以扩展、维护和测试。AngularJS之所以使得程序开发变得更灵活、便捷，主要得益于这个框架的创造理念和所带来的四大核心特性：MVC、Module、指令系统、数据双向绑定。

MVC (Model View Controller) 程序设计模式是19世纪70年代诞生的，最初在后台开发中经常会涉及到MVC开发模式，而AngularJS将后台的MVC模式写入前端语言中，为前端技术带来新的发展方向。Model即数据模型层，被存储在对象的属性中；View即展示给用户的视图层，也就是前端中所提到的DOM；Controller即业务逻辑和控制逻辑层，主要指JavaScript类。其核心思想就是将model, view, controller的代码分开，从而提高程序的可移植性，使代码逻辑清晰易懂。

Module (模块化) 是提供特殊服务的功能块。在前端的开发中，我们需要尽量避免定义全局变量和全局函数，而AngularJS则将这种靠编程习惯上的约束直接变成新的框架特性，方便大家的开发。在AngularJS的模块化概念中，模块可以是对象、方法，每个模块的代码独立拥有自己的作用域，model, controller等。模块之间互不影响，服务在相应的模块内部，当我们需要某个服务的时候，需要先把模块实例化，然后再调用模块的方法。对于AngularJS而言，一切都是从模块开始的，所有东西都是建立在模块的基础上，只有定义了模块才能调用其他方法。

指令系统是AngularJS最有特色、最吸引人的一个特性，强大的directive可以将很多功能封装成HTML的tag, 属性或者注释等，这大大美化了HTML的结构，增强了可阅读性。AngularJS中所用到的ng-app, ng-bind都是框架的内置指令，相应的我们也可以自定义自己需要的指令。指令是绑定在DOM元素上的函数，它的作用就是帮助定义应用程序的视图，可以调用方法、操作文档那个对象模型以及绑定controller和scope对象。比如，加载angularJS时找到ng-app指令，确定应用的边界；编译时遍历DOM，找到所有的指令，根据代码template等生成DOM结构，并且执行指令的compile函数。

数据双向绑定是AngularJS区别于其他主流框架的一个主要创新之处。它的工作原理是：HTML模板被浏览器解析到DOM中后，DOM结构也就相应的成为了AngularJS编译器的输入。AngularJS将会遍历整个DOM结构，来生成相应的ng指令，生成的所有指令都针对view来设置数据绑定。也就是，界面上的操作可以直接反映到数据，数据的变更也能实时的展现到界面。其原理主要是使用发布者-订阅者模式(PubSub模式)，框架使用数据特性来为HTML代码进行绑定，所有被绑定在一起的JavaScript对象和DOM元素都会订阅一个PubSub对象。只要JavaScript对象或者一个HTML输入元素监听到数据变化时，就会触发绑定到PubSub对象上的事件，从而其他绑定的对象和元素都会做出相应的变化。Angular的数据监测并不存在定时脏检测，而是对常用的DOM事件，XHR事件等做了封装，在里面触发进入Angular的digest流程。在digest流程里，会从RootScope开始遍历，检查所有的watcher。同样，因为对所有watcher的监测，当数据量较大时也会带来性能上的降低。

AngularJS除了它的这四大核心特性之外，不得不提的还有它的依赖注入思想。上面讲到AngularJS的模块特性，Angular的模块只保留服务的声明，服务的实例化是由服务注入器完成的，实例化之后服务就留在了服务注入器中。依赖注入使得你可以将自己需要的东西通过调用Angular来注入而不需要去创建那些它所依赖的东西，它是最少知识原则设计模式的一种体现，我们可以直接把scope对象注入到相应的Controller中进行数据的处理，这一特性极大的提高系统的可测试性。在开发过程中，对象的依赖注入，很大程度上加速了动态web应用的开发速度。这种将后端语言的设计模式赋予前端代码，意味着前端代码可以提高重用性和灵活性，未来也可以将更多的操作放在客户端，服务端只需提供数据来源和其他客户端无法完成的操作，这也大大开拓了前端开发的丰富性。

* + 1. React技术介绍

React是Facebook开发的一款JS库，最初来自Facebook内部的广告系统项目，随着项目的不断迭代开发，代码变的越来越臃肿且混乱不堪，不易于维护。开发工程师们认为MVC无法满足他们的开发需求，每当需要添加一项新的功能或特性的时候，系统的复杂度就成级数的增长。特别是当系统中有很多的模型和对应的视图的时候，由于模型和视图之间可能存在双向数据流动，系统会变得非常难于理解和调试。所以他们决定舍弃很多所谓的“最佳实践”，重新对前端框架进行构建，也就是现在的React.

React不是一个MVC框架，也不使用HTML模板，采用非常简单的响应式更新。它可以简单的表述任意时间点你的应用的表现状态，当数据发生变化时会自动的管理UI界面进行更新。并且React构造可重用组件，通过封装使得组件代码复用、测试以及关注点分离。只需要进行组件的构建和组建，对页面进行更新时实际上是只更新了变化的那一部分组件。React带来了很多开创性的思路来构建前端界面，以简单直观、符合习惯的方式去编程，让代码更容易被理解，从而易于维护和扩展。React通过组件机制来定义界面，组件之间以及组件与模型之间则是通过单向数据流来进行通信。这样避免了渲染大数据集时，监听所有数据变化所带来的性能消耗。使用React构建应用程序，其编程思想非常简单，首先将UI拆解到组件层次结构中，然后用React构建一个各个组件的静态版本，确定UI state的最小但完整的表示，最后确定state的位置并添加反向数据流，这样一个完整的React应用程序就构建成功了。

React的核心机制之一就是虚拟DOM，即可以在内存中创建的虚拟DOM元素，虚拟DOM和真是DOM元素一样也可以通过JavaScript来创建。React利用操作虚拟DOM元素以减少对实际DOM的操作来提升性能。为了使通过JavaScript构建出来的DOM树比较美观，React采用JSX语法，即通过HTML来创建虚拟DOM树，增加代码的可读性。JSX语法看上去和模板比较类似，但其本质是通过代码来构建界面，也可以理解成一种新的语法糖。

React的虚拟DOM机制主要服务于其高效的Diff算法。传统的web界面由DOM树构成，当某一部分发生变化时，其实就是对应的某个DOM节点发生变化，前端需要实时更改相应的DOM节点，而DOM操作带来的性能开销较大。所以React采用虚拟DOM，将所有的DOM变化在虚拟DOM中进行更新，并通过Diff算法对比虚拟DOM和真是DOM树之间的差异，以最小的转换步骤对真实DOM进行更新。在React的Diff算法中认为，两个相同组件产生类似的DOM结构，不同的组件产生不同的DOM结构。React的Diff算法会对DOM树逐层进行节点比较，对于同一层次的一组节点，它们可以通过唯一的id进行区分。当在树的同一位置前后输出了不同类型的节点，React直接删除前面的节点，然后创建并插入新的节点。在React中则以组件为比较单位，当React在同一位置遇到不同的组建时，也是简单的销毁第一个组件，把新创建的组件替换上去。正是这样的一种Diff比较算法，将原本需要进行O(n^3)时间复杂度的标准Diff算法优化至时间复杂度为O(n)。

* 1. Spring Boot框架简介

Spring Boot是 Pivotal团队于2013年提出的全新框架，其设计目的是用来简化新Spring应用的初始搭建和开发过程。从根本上来讲，Spring Boot就是一些库的集合，能够被用来构建任意的系统。Spring Boot继承了原有Spring框架的优秀基因，简化了使用Spring的过程，能够快速启动一个web容器，因为这些优点它可以快速的帮助开发者搭建一个Spring框架。Spring Boot是伴随着Spring 4.0诞生的，还集成了很多我们常用的基础框架，如内嵌容器Tomcat、Jettty等，日志框架、Jms框架、缓存框架、持久性框架以及流行的NOSQL数据库如MongoDB等，如图2-1所示：



图2-1 Spring Boot的诞生

Spring Boot的出现使得我们编码变得更加简单易懂，声明一项依赖只需要:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

实现一个方法只需要：

@Configuration

@EnableAutoConfiguration

@ComponentScan

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

Spring框架的配置也极为繁琐，用户需要自己编写XML文件并在Maven中增加相应的依赖包，在相应的代码中添加功能代码，才能使用。并且当引入多个依赖包的时候，经常需要处理依赖包之间的版本冲突，重复且费时费力，大大降低开发效率。而使用Spring Boot进行开发的时候，只需要在Maven的配置文件中加入少量的配置就可以使用所需要的框架，也可以轻松的解决依赖包的版本冲突问题，大大的简化了配置，具体如图2-2所示：



图2-2 Spring Boot配置

Spring Boot也使得部署变得非常简单，一键启动，不需要预部署应用服务器，并且对运行环境的基本要求也大大降低，只需环境变量中有JDK即可全量更新。同时也可通过HTTP GET请求对应用进行监控，查看配置属性、线程工作状态、环境变量、JVM性能指标以及RESTful Path与服务类的映射关系等十余项性能参数。Boot对Spring应用的开发提供了模块化方式导入依赖的能力，强调了开发RESTful Web服务的功能并提供了生成可运行Jar的能力，这也表明Spring Boot在微服务部署方面也有着强大的优势。同时Spring Boot具有自动配置Spring Data以访问数据库的能力，这也使得它在数据库集成方面也变得非常简单。只需要在工程中将spring-boot-starter-data-jpa包含进来，框架的自动配置引擎就能探测到你的工程需要数据访问功能，并且会在工程上下文中创建必要的Bean。微服务所提供的数据并不一定全部适合关系型结构，针对这些，Spring Boot也专门提供一些模块供开发人员使用，包括Spring Data的MongoDB和Redis项目。因为Spring Boot和Spring一脉相承，所以对于广大开发者而言，学习成本几乎为零，可以较快上手，并大大提高开发效率，综合其种种优势Spring Boot也可以看做是微服务框架的起点。

* 1. 微服务架构简介

微服务架构在过去的几年成为软件架构中的热门，微服务是指开发一个单一、小型但有业务功能的服务，每个服务有自己的处理和轻量通讯机制，并且可以部署在单个或多个服务器上。它也是一种松耦合的、有一定有界上下文的面向服务的架构。也就是说，如果每个服务都需要同时更改那么他就不算是微服务，因为它们紧紧的耦合在一起，牵一发而动全身，一个服务的宕机可能导致整个应用的不可用。

通过与传统得WEB开发模式相对比我们可以更好的了解微服务架构。通常我们的开发也是模块化逻辑，程序完成后会打包并部署为一个个具体的应用。每个具体的格式依赖于相应的开发语言和框架。所有的功能打包到一个WAR包里，基本没有外部依赖（除了容器），部署在一个JEE容器里，包含了DAO、Service、UI等所有逻辑，具体结构如图2-3所示：



图2-3 传统架构模式

这种架构模式，开发简单直接，集中式管理，基本不会重复开发，功能都在本地，没有分布式的管理开销和调用开销。同样它的缺点也比较明显，这样的应用会随着功能的不断添加，代码结构越来越臃肿，所有开发都在一个项目改功能，提交代码相互等待，代码冲突不断，部署不灵活，稳定性不高等问题。

微服务的目的则是有效的拆分应用，实现敏捷开发和部署。简单地说，微服务架构模式就是将整个WEB应用组织为一系列小的WEB服务。这些小的WEB服务可以独立的进行编译和部署，并通过各自暴露的API接口相互通讯。它们彼此相互协作，作为一个整体为用户提供功能，却可以独立的进行扩容。在微服务架构模式中，我们需要更高一层的分割，在尝试将需要实现的功能分成一系列的组件之前，应该首要要考虑怎样将需要实现的功能交给彼此相互独立的一系列服务来完成，并且每种服务有自己的侧重点。在微服务架构模式中还需要一系列的公共服务来辅助整个应用的运行，包括用于整个应用的权限管理服务，以及监控各个服务实例服务状态的模块。这些服务在各个子服务的服务实例之间共享。

微服务在现在的开发中越来越多的被使用，许多SASS公司正在摆脱单一产品架构模式，而转向微服务架构。在微服务的概念中，WEB服务器或微服务器都是单元组件，这些单元组件提供完整的业务功能，也可以依靠其他微服务完成。由于单独构建组件和单独运行它们意味着集中控制和数据管理的概念消失了。传统架构，是围绕一系列的架构、技术和框架来构建的，关键的架构将决定使用什么技术，关键的DBA将决定使用哪些数据库。而在微服务架构中，每一个组件模块都完成一个完整的业务功能，所以关键架构和关键DBA的集中控制也随之消失，进而采用分散式控制，具体变化如下图2-4所示：



图2-4 微服务架构分散式控制

因为所有的微服务都是独立的Java进程跑在独立的虚拟机上，所以服务间的通信就是IPC(inter process communication),这个已经有很多成熟的方案。现在最常用的一般有两种，同步调用（REST、RPC）和异步消息调用（Kafka、Notify）。一般同步调用比较简单，一致性较强，但是容易出现问题，体验较差，特别是调用的层次较多时。一般REST基于HTTP，比较容易实现也比较容易接受，服务端的实现也更加灵活，支持多种语言，同时能跨客户端。所以相对而言，使用的比较广一些。

在微服务架构中，一般每一个服务都有多个拷贝，来做负载均衡。服务之间的感知一般都是通过ZooKeeper等类似技术做服务注册信息的分布式管理。当服务上线的时候，服务提供者将自己的服务信息注册到ZooKeeper等类似框架，通过心跳维持长链接，实时更新链接信息，服务调用者通过注册框架寻址，根据可定制算法找到相应的服务。

微服务架构相对于传统架构模式优势也较为明显。

1. 复杂度可控：微服务架构将应用分解后，每个服务只专注于单一的功能，避免了单一应用复杂度无限积累的问题。由于每个微服务体积小，复杂度低可由一个小规模的开发团队完全掌控，易于保持较高的可维护性和开发效率。
2. 独立部署：微服务的单一服务具有独立进程，可以进行单独部署。当某个微服务发生变化时不需要重新编译、部署整个应用。使得发布更加高效，同时降低生产环境给应用部署带来的未知风险。
3. 技术选型灵活：微服务架构是去中心化的，只需通过暴露的API与其他服务进行交流，每个服务可以根据自己开发团队的需要选择适合自己的技术栈。大大提高团队的开发效率。
4. 容错：当某一服务发生故障时，故障会被隔离在自己的单一服务中，只会造成自己当前服务的不可用，不会影响应用整体。
5. 功能特定：一个微服务一般完成一个特定的功能需求，比如消息管理、用户管理等。每一个微服务也都有自己的应用逻辑和适配器，运行时，每一个实例可能是一个云VM或者是Docker容器。
   1. 推荐系统相关介绍
6. 推荐系统定义

随着互联网的高速发展人们已经从信息匮乏的时代走入了信息过载时代。如何才能从大量的信息中快速、高效的找到我们想要的，是现在面临的主要挑战。而推荐系统则是解决这一问题的重要工具。推荐系统的主要目标就是构建支持用户在线决策的系统，这个目标强调要为大规模用户提供便捷访问的高质量推荐，即推荐用户支付得起、个性化、匹配度高的产品。推荐系统作为一门技术，广泛的应用在电子商务、大规模零售业以及新闻知识信息推送，并且深入人工智能、数据挖掘等各个高新技术，视图寻求为自己提供有力的指导和改进。比如当你想要看一部电影但又不知道看具体哪一部的时候，你打开视频网站面对无数的电影资源会显得手足无措，这个时候就是你遇到信息过载的时候，此时你需要一个过滤工具来帮你筛选最适合你看的、你最想看的电影来给你一些选择建议，缩小可选择范围。这个工具可以根据你的历史兴趣，以及点评记录，从庞大的电影片库中选择几部符合你兴趣的电影，这个工具我们就称之为推荐系统。

众所周知的帕洛阿尔托研究中心的Tapestry系统（Goldberg et al.1992）引入了协同过滤的思想和概念，展示了用户如何利用数据进行过滤；Grouplens系统针对Usernet新闻消息进行自动协同过滤；贝尔通讯研究中心的视频推荐系统针对影片进行推荐。这些系统都是通过算法进行计算，识别出有相同爱好的用户，然后综合他们的个性化评分进行加权平均值计算，简单的“K最近邻”算法则从中脱颖而出，并取得良好的效果。最基本的推荐系统模型如下图2-6所示：



图2-6 推荐系统模型

1. 推荐算法

推荐算法是推荐系统的核心，由上一小节中推荐系统模型可知，推荐系统收集用户信息，获取用户偏好，将所有数据传给推荐算法，由推荐算法通过计算为用户提供个性化推荐。由此可知，一个好的推荐算法就类似于推荐系统的大脑，直接决定着整个推荐系统的质量。同样，同一个推荐算法，在不同的场景下，得到的推荐结果也可能会有很大的差异。所以在设计推荐系统时，应该充分的了解各个推荐算法的不同特性，结合系统需求，适当调整推荐算法的推荐参数，也可将多个推荐算法混合使用，将推荐算法的优势发挥到最大。常见的推荐算法主要有，基于内容的推荐、基于关联规则的推荐、基于协同过滤（用户、商品、模型）的推荐以及混合推荐。

1）基于内容的推荐是在推荐系统出现之初使用最为广泛的推荐算法，它的核心思想是根据推荐物品或内容的元数据，来计算物品或内容的相关性，然后根据用户以往的喜好记录，推荐跟已购买物品相类似的产品。



图2-7 基于内容的推荐

例如，如上图2-7所示，在对电影的推荐中，已知用户A喜欢电影A，根据电影的定位以及用户评价可知电影A的标签为爱情、浪漫类型的。用户B和用户C都喜欢电影B，从类型标签可知电影B的类型为恐怖、惊悚系列。这时有新电影C上线，该影片的类型标签为浪漫、爱情，此时将电影C推荐给用户A则明显比推荐给用户B和C被接受的概率要高得多，当然也不排除用户B和用户C也喜欢爱情、浪漫类型的电影，所以基于内容的推荐在有大量数据集的支撑下，推荐会更加精准。这种推荐算法易于实现，不存在用户数据系数和冷启动的问题，同时，因为是基于物品本身特征进行的推荐，也不存在过渡推荐热门的问题。主要缺点是为了保证物品定位、分类的实际意义，需要人工维护tag，从而保证tag的准确性。

2）基于关联规则的推荐，更常见于电子商务推荐系统中。这种推荐算法的前提要素即要挖掘出关联规则，也就是那些同时被很多用户购买的商品集合，然后在集合内进行推荐。常见的基于关联规则挖掘算法大多都是从Apriori和FP-Growth演变而来的。基于关联规则的推荐算法转化率一般较高，当用户购买规则集合内物品后，购买集合内其他物品的可能性一般会更高。但也存在一些缺点，因为需要采集大量用户数据，所以不可避免的会存在冷启动和稀疏性的问题，同时计算量也比较大，并且热门物品可能会存在被过渡推荐。

3）协同过滤算法（Collaborative Filtering, CF）是目前在推荐系统中使用最为广泛的一种推荐算法。协同过滤的思想非常简单，它认为兴趣相似的用户可能也会喜欢相似的东西，或者对相似的东西表示出相似的偏好程度。从该算法中的协同二字可以看出，该算法主要利用的是集体智慧，通过相似用户对相似物品的评分进行个性化推荐。协同过滤算法可以分为两大类，基于记忆的协同过滤算法和基于模型的协同过滤算法。

基于记忆的协同过滤算法主要又可分为基于用户的协同过滤算法（User-based Collaborative Filtering, UBCF）和基于物品的协同过滤算法（Items-based Collaborative Filtering, IBCF）两种。这种算法前者首先计算用户和其他用户之间的相似度，然后选择近邻进行推荐。而后者与之相近即首先计算物品之间的相似度，然后选择近邻进而生成推荐。其主要思想如下图2-8所示：



图2-8 基于用户的协同过滤推荐

基于记忆的协同过滤虽然取得较大的成功，但当用户数和物品数巨大时无法保证推荐的实时性，所以进而提出了基于模型的协同过滤算法。基于模型的协同过滤算法是通过对用户或对物品的评分记录，运用统计学或者机器学习的一些思想进行模型构建，在离线的情况下构建好用户的偏好模型，在线使用已构建好的偏好模型，根据用户的实时信息进行预测从而生成推荐，以此做到快速响应的效果。建模过程中主要用到的技术包括聚类技术、奇异值分解技术、贝叶斯网络技术回归分析等。还有目前比较新的支持向量机技术也被用于构建模型。

总体而言协同过滤算法就是根据相似度的高低找出k个与你最相近的邻居，在这些邻居喜欢的物品中，根据邻居与你的远近程度计算出每一件物品的推荐度，最后根据每一件物品的推荐度为你生成推荐。协同过滤算法的核心是基于历史数据的，所以对于新物品和新用户都有“冷启动”的问题，推荐的效果也依赖于用户历史偏好数据的多少和准确度。

4）混合推荐算法。实际应用中其实很少有直接用某种单一的算法来做推荐系统的。在一些大的网站如Netflix等，通常会融合几种甚至数十种推荐算法，通过给不同算法生成的推荐结果加权重来综合结果，或者是在不同的计算环节中运用不同的推荐算法来混合，以达到生成更贴合自己实际业务的推荐结果。

* 1. Docker技术简介

Docker是一个开源的容器引擎，可以轻松的为任意一个应用创建一个轻量级的、可移植的、自给自足的容器。让开发者可以打包他们的应用以及依赖包到该容器中，然后发布到任何流行的Linux机器上，也可以实现虚拟化。并且进一步封装了容器的一些操作接口，开发者可直接对容器进行管理和使用，而无需去了解Linux底层的一些运行机制。其核心思想即“构建一次，到处运行”，这也体现了Docker在基础设施层的便捷部署能力。Docker应该是2014年最火爆的技术之一，自开源后受到广泛的关注和讨论，其目标就是实现轻量级的操作系统虚拟化解决方案。

Docker的基础是Linux容器（LXC）等技术，并在此基础上进行了进一步的封装，使得操作更为简便。与传统的虚拟化技术相比，Docker容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统的虚拟化技术则是在硬件的基础上，虚拟出自己的系统，再在系统上部署相关的服务。传统虚拟化方式虽然也可以为虚拟机模拟出一套完整的硬件环境，同样这也会造成对资源的过度封装从而对系统资源需求过大，并且额外需要的虚拟机管理层也会耗费大量的系统资源，这也是传统方式中一台物理机上只能运行极少数量的虚拟机的原因。相对而言，Docker容器的这种直接在宿主机操作系统上利用LXC技术进行资源的管理，几乎不会消耗额外的系统资源。正是由于这种机制，在实际生产环境中我们可以再一台物理机上同时部署运行上千个Docker容器。这两种机制的详细对比见下图2-5：



图2-5 Docker容器和传统虚拟机技术的不同

Docker虚拟化中主要涉及镜像、容器和仓库三个概念。其中，Docker镜像其实就是模板，类似于常见的IOS镜像，是一个样板；容器是指使用镜像常见的应用或者系统；仓库则是存放镜像的地方，并且分为公开仓库和私有仓库两种形式。和传统虚拟化技术相比，Docker容器具有以下几种优势：

1. 操作启动快。Docker基于宿主操作系统进行操作，使得运行时的性能可以获得较大的提升，管理操作（启动，停止，开始，重启等等）都是以秒或毫秒为单位的。这大大提高开发效率，减少无效的等待时间。
2. 轻量级虚拟化。Docker容器高效的资源利用率，因其不消耗过多的系统资源，你可以拥有足够多的“操作系统”，仅需添加或减少镜像就好。
3. 开源免费。Docker虚拟化技术是开源的、免费的、低成本的。由现代Linux内核支持并驱动。
4. 跨平台。因其只需拥有Linux内核的操作系统即可，所以Docker容器可以运行在虚拟机、物理机以及各种主流的云平台。这种特性也使得移植变得更加简单便捷。
5. 快速的软件交付周期。Docker容器因其资源隔离特性，使得开发环境和生产环境都可以相对独立的操作，进而缩短了代码开发到代码部署上线的整个周期。

目前为止Docker技术通常用于如下场景：

* Web应用的自动化打包和发布；
* 自动化测试和持续集成、发布；
* 在服务型环境中部署和调整数据库或其他应用；
* 从头编译或者扩展现有Cloud Foundry平台来搭建自己的Pass环境；
  1. 本章小结

本章主要介绍了基于推荐算法和微服务架构的供需对接平台中所用到的相关技术，主要分为五个小节。

第一节主要介绍了系统客户端以及web后台管理端前端开发所用到的技术框架，包括Framework7、AngularJS以及React，对其核心技术、主要设计思想以及优劣势进行分析介绍。

第二节主要介绍了系统后端开发所用到的Spring Boot框架，包括其诞生、针对Spring所做的一些优化以及应用到开发中所带来的一些优势。

第三、四节主要介绍了本系统的核心框架微服务架构相关技术和与微服务架构部署相辅相成的Docker容器相关技术。包括这两项技术诞生背景，核心设计思想介绍以及和传统架构（传统虚拟机技术）相比有有哪些优势。

第五节则是主要介绍了推荐系统的定义，整体结构以及其核心部分推荐算法相关。并根据分类分别介绍了基于内容的推荐算法、基于关联规则的推荐算法和基于协同过滤的推荐算法的核心思想。

1. **需求分析**
   1. 功能性需求分析

通常我们想要设计并实现一个系统时，首先要对系统预计实现功能进行分析，然后根据功能将系统分为多个子系统，并且每个子系统下面又可以包含多个模块，每个模块实现特有功能。系统功能的清晰、合理划分会使得我们后面的开发条理清楚，水到渠成。

本文所设计并实现的系统，是一个以微服务架构为基础以推荐算法为驱动的供需对接系统。系统实现根据用户特征以及历史数据信息主动推荐合适订单，用户双方互选以完成在线匹配。该系统包括用户直接使用的客户端和管理人员使用的web管理端，用户通过客户端发布需求、查看需求以及完成匹配。管理端实现对注册用户和发布的需求进行审核，符合要求的用户信息和需求信息才会被通过，以此保障平台信息数据的可靠性，并且还提供对用户信息、用户星级、需求信息以及交易记录进行管理。通过上面的分析可知，系统的总体业务流程如下图3-1所示：



图3-1 系统总体业务流程图

下文将具体分为客户端和管理端分别进行分析，并结合流程图和UML示例图进行详尽的分析描述。

1. 客户端功能性需求分析

对于系统客户端来讲，首先从使用者来划分，系统的使用者主要分为两类：一类是需要为自己找家教的用户，这类用户主要是为自己孩子找家教老师的家长，当然也不排除一些成年人想要培养工作之余兴趣爱好而找辅导老师的情况，为了统一区分我们暂时把这类用户称为家长类用户；而另一类则是有时间有能力提供家教服务的在校大学生，当然也不排除在职教师的情况，我们统称这类用户为教员。

本系统家长类用户和教员类用户共用一个客户端，首次注册时需手动选择身份，根据选择的不同身份进入相应的注册流程，注册完成后进入系统。之后每次进入系统，系统会根据用户的相应身份进入不同的页面。主要流程如下图3-2所示：



图3-2 进入系统流程图

1、家长类用户从功能需求上来讲，主要包括个人信息管理模块、学生信息管理模块、发布新的订单需求模块、订单匹配管理模块以及评价教员模块，用例图如下图3-3所示：



图3-3 家长用例图

（1）个人信息管理模块

家长对个人信息的管理主要包括，姓名、手机号、家庭住址以及座机号，后台管理人员会根据家长座机号核实家长真实身份。只有身份审核通过的家长才可以通过系统发布需求订单。因家教需要上门服务所以家长在注册家庭住址信息时需要详细到门牌号，同时也为了保护家长个人隐私和信息安全，系统需要对家长的详细地址信息进行部分处理。

（2）学生信息管理模块

每个家长账号下面最多可注册5个学生信息，家长可对学生信息进行增加、删除、修改等操作。学生基本信息包括学生姓名、性别、年龄、当前就读学校以及家长对学生的一些简单介绍，这些信息可以为学生生成相应的标签，方便推荐系统在为家教老师推荐需求时进行参考，信息越详细、真实则推荐的匹配度越高。学生信息中还有一项是有效期，账号下每位学生每年交取一定额的年费，则系统会为其提供有效期内无限次的家教需求匹配。系统需记录学生的有效期信息并在超出有效期后禁止为其发布家教订单。

（3）发布订单需求模块

发布家教需求是家长用户的主要操作，首先系统会判定家长是否实名制认

证，如果没有实名制认证则不提供发布订单入口。家长发布订单时因为需要提交的信息过多暂定分为三个步骤：首先选择学生信息，如果学生未在有效期内则提醒需先交取年费才可以创建订单，并跳转至相应界面，同时选择家教的年级、科目、上课时长等信息，系统会根据年级科目信息自动生成相应价格，家长确认后进入下一步；第二步则是选取教师需求，包括家教老师的性别、星级、之后的上课时间安排以及一些额外的需求；第三步则是订单中家长的地址信息，该部分会根据家长个人信息中的地址信息预填，家长也可以根据自己的需要进行修改。

三部分都完善后，家长便可以提交订单，具体的操作流程如下图3-4所示：



图3-4 创建订单流程图

（4）订单匹配管理模块

家长将订单发布后，订单会初始化为COMMITED(审核中)状态,后台管理人员会对订单进行人工审核，如果订单不符合要求（比如地点太偏僻，时间点太晚或者详细说明中包含联系方式等信息）会被退回修改并更新状态为INITIALIZED（待处理），这两种状态的订单不会出现在教员的推荐列表中。订单后台审核通过后，订单状态会更新为PUBLISHED(招聘中)，此时的订单将会加入推荐系统中，根据相应的推荐算法推送给相应的用户，教员用户可以查看并进行竞聘。“招聘中”状态的订单，家长可以进行取消操作，订单状态更新为CANCELED（已取消），如果此时此时已经有教员竞聘，则应给相应教员发送短信提醒，告知该订单已取消。同时每份订单最多可有三位教员竞聘，家长则需对合适的教员进行聘请操作，聘请过后订单状态更新为MATCHED(匹配成功)，订单则从匹配系统中排除，其他教员不可见，若家长对所有竞聘教员都不满意则可淘汰掉部分教员，此时可有新的教员进行竞聘。匹配成功的订单，只会出现在家长和相应教员的订单列表中，等待其他操作。订单匹配成功后，家教老师上门服务，如果通过实际授课家长并不满意家教老师的教学质量可以单方面发起“中止订单”操作，此时订单状态修改为ENDING（中止处理中）状态，如果教员端同意中止申请，订单状态更新为ABORTED(已中止)，否则则需要后台管理人员进行协商解决。与中止订单相对应的还有结束订单操作，当家教老师完成一学段的家教工作时，需由任一方发起结束订单操作，此时订单状态更新为FINISHING(结束中)，等待另一方确认，对方确认后订单状态更新为FINISHED(已结束)，表示该订单已结束。

根据上面对订单的分析可知订单有多种不同状态，具体状态描述如下表3-1所示：

表3-1 订单各状态的描述

|  |  |
| --- | --- |
| 状态项 | 说明 |
| COMMITED(审核中) | 订单已发布，待审核。 |
| INITIALIZED（待处理） | 订单已退回，待修改。 |
| PUBLISHED（招聘中） | 订单审核通过，已进入推荐系统。 |
| CANCELED（已取消） | 订单已取消，成为历史订单，不可进一步处理。 |
| MATCHED（招聘完成） | 订单已匹配成功，不可继续竞聘，可进行中止结束操作。 |
| ENDING（中止处理中） | 订单发起中止操作，等待对方处理。 |
| ABORTED（已中止） | 订单已经协商中止。 |
| FINISHING（结束中） | 订单结束，发起结束操作，等待对方处理。 |
| FINISHED（已结束） | 订单已成功结束。 |

并且可以得出各状态之间根据不同操作而发生变化的UML状态机图，如下图3-5所示：



图3-5 订单状态改变流程图

（5）对教员评价模块

订单结束后，家长需对家教老师本次课程质量予以评价，该模块跟教员类用户的星级管理模块直接相关，评价模块以五分制评分的形式，根据教学态度、是否准时以及备课质量等多个方面进行评分，最后根据权重加成为用户得出一个综合得分。

2、教员类用户从功能需求上来讲，主要包括个人信息管理模块、订单需求管理模块、定向搜索模块、上岗证获取模块以及星级管理模块，用例图如下图3-6所示：



图3-6 教员用例图

（1）个人信息管理模块

教员个人信息管理模块与家长个人信息管理模块略有不同，除了最基本的姓名、性别、手机号等基本信息外，省去了详细家庭住址和学生管理两部分，额外的需要进行学校认证，以保证家教老师的教学质量。本系统中通过公共数据库账号、密码来对教师进行身份认证，为了更好的根据已有信息来对教员构建模型，方便推荐系统使用，还需要教员用户完善自己的学院、专业信息以及个人兴趣爱好、历史教课记录信息等。个人具体信息如下图3-7所示：



图3-7 教员个人信息结构图

（2）订单需求管理模块

教员端订单管理模块主要操作类似于家长用户端，没有了家长端的订单创建功能，但是增加了发现页，在这部分将会展示所有系统根据匹配度为教员推荐的家教订单，教员从中选择适合自己的订单进行竞聘，极大的减少了冗余信息对教员的干扰。教员最多同时进行五份家教订单，并且最多可同时竞聘三份订单，若家长长时间未进行聘请操作，或者教员自己想要放弃该份订单的竞聘可以直接通过“取消竞聘”操作，退出此次竞聘。同时教员也可以通过定向搜索订单模块通过指定筛选条件，搜索自己想要看的订单。教员客户端“我的订单”模块显示所有跟教员相关的订单，并可以通过tab切换，查看不同状态类别的订单，清晰明了的对相关订单进行分类管理。

（3）上岗证考试模块

上岗证考试模块是一个独立开发的在线考试模块，教员用户通过系统提供的入口跳转至在线考试模块进行在线答题，考试通过则给与上岗证号，教员只有拥有上岗证号才可以进行竞聘操作。

（4）星级管理模块

星级管理模块是对教员的评价激励机制，不同等级的教员每小时的课时费不一样，这也是对教员教学能力的一个肯定。教员可以通过完善自己的个人信息或者获取上岗证来获取相应的积分，但这些积分属于一次性的，增值积分则是需要靠家教过程中家长的评价，根据评价得分的高低来获取相应的积分，当积分达到一定值并且满足该星级的教学年限则可自动升星，当星级达到三星以后，每次升级需要通过系统登记申请，管理人员实际审核过后予以升星，五星最高。家教老师有违规操作或者不良记录的情况下，管理人员也可以通过扣除积分进行降星处罚。

1. Web管理端功能性需求分析

对于web管理端来讲，首先从权限管理来划分，主要分为超级管理员和普通管理员两类。普通管理员即通常所讲的后台管理人员，他们的主要功能需求包括：用户管理、订单管理和星级管理。其中用户管理模块又包含家长类用户管理和教员类用户管理两个模块。超级管理员则包括普通管理员的所有功能，并额外包括账号管理和费用明细两个功能模块。由此可得管理端的功能需求结构如下图3-8所示：



图3-8 管理端系统功能用例图

各个模块主要功能需求描述如下：

1. 用户管理模块

Web管理端的用户管理模块主要包括对用户基本信息的增删改查操作，以及相应的备注操作。家长类用户除了基本信息以外还有额外的学生信息管理，一个家长用户可以关联多个学生信息，每个学生可以单独设置有效期。在用户管理中可以对指定用户进行禁用操作，用户账号被禁用后，用户将不能登录客户端，推荐系统也不会向该用户进行推送，但后台依然可以查询到该用户信息。禁用操作常用于对违规用户的惩罚处理和已毕业教员的暂停家教资格处理。

1. 订单管理模块

订单管理模块是后台管理的重中之重，主要包括对家长发布订单的审核，管理员根据相关要求审核订单，不符合要求的订单“退回”处理，符合要求的订单点击“发布”操作，订单则进入推荐系统流程。对于招聘中的订单管理员可以帮助家长后台对已竞聘教员进行淘汰或聘请操作，订单匹配成功后，管理员通过对订单的“派遣”操作，打印带有家长详细地址的派遣单给教员，教员交取相应的匹配费用持派遣单上门进行家教。后台管理员还可对“中止处理中”状态的订单进行协商处理，并更新订单状态。综上可知订单管理模块的功能用例如下图3-9所示：



图3-9 订单管理功能用例图

1. 星级管理模块

本系统中积分系统由用户客户端记录，根据家长评价用户累计积分，并且1-3星级为系统自动升星，无需管理员后台操作。当用户达到三星级后，若达到升星要求时可通过客户端提交升星申请，后台星级管理系统中会生成相应记录。记录包括申星申请发起人姓名、邮箱、手机号、当前星级、当前积分以及上次升星时间。后台管理员登录星级系统，发现有新的升星申请时，对用户实际情况进行调查，以及对该教员历史订单进行回访，经管理老师讨论通过后，予以升星操作，若讨论不通过，则拒绝升星并通过系统给教员回复邮件说明理由并设置下次发起申请的最短时间间隔。

1. 账号管理模块

账号管理模块是仅对超级管理员可见的模块，该模块实现添加管理员账号并设置初始密码。超级管理员可以对已有管理员进行禁用或删除操作，并且可以对不同的管理员设置不同的功能权限，比如：A管理员只负责用户管理，则订单管理等其他模块均对其不可见；B管理员只负责订单审核，则出订单管理模块外其他模块也都对其不可见，实现专人专能，分工明确。

1. 费用明细模块

费用明细模块也是一个仅对超级管理员可见的功能模块，在这个模块中，可查看最近的所有收支明细，具体到每笔费用的支付人、关联订单以及支付时间等所有相关信息。可以分类别查看教员匹配费和家长服务费的收入，以及因为家教试讲不通过，退还匹配费的所有记录。该模块内同时提供对最近几个月收支统计的图表信息，使得所有数据一目了然。

* 1. 非功能性需求分析

系统的功能性需求分析主要侧重于系统所要实现的功能，以及系统的整个使用流程，功能性需求分析透彻有助于整个系统框架的搭建。但仅仅这样还不足以完全的描述实现一个系统的所有需求，“非功能性需求”也应是我们的重点分析对象。

所谓的非功能性需求，就是指一个好的系统在实现所有功能性需求的基础上，还应该具有的一些特性。比如易用性、安全性、可扩展性等。下面就分别对这三项进行分析。

1. 易用性要求

一个好的系统除了可以实现用户所需的所有功能之外，还应有较好的用户体验，以用户为中心适应用户的使用习惯。系统想要有好的用户体验首先应该保持良好的页面视觉效果，包括整体配色风格，菜单栏布局，空间大小样式统一等；也需要有合理的功能布局，尽量减少功能的深层次嵌套，最常用的功能放在最显眼的地方；清晰的逻辑操作流程，明确的让用户知道执行一项操作需要哪些步骤，以及用户目前所处的位置；常用功能的便捷入口，尽可能的减少无效的跳转操作。

针对以上要求我们可以采取如下措施：

（1）专业的UI设计，配色简单大气，功能控件统一设计，保持风格一致；菜单、工具栏等符合主流用户的使用习惯，可适当使用图标，但图标要简单直观，做到见图知意；提示、帮助信息固定合理显示位置。

（2）系统主体采用底部tab栏切换，放置常用的订单模块、个人管理模块以及首页的快捷切换；功能菜单纵深一般控制在7个以内，层级一般控制在3层以内，横向菜单尽量单排排列并且不要超出屏幕宽度。

（3）用户所处位置定位，最基本要求用户所在功能区标签或图标高亮显示；因用户创建订单时需提交大量信息所以分三页显示，在完善信息的每个页面中应标明该操作共分几步，当前处于哪一步，这样业务流程清晰明了，用户可以感受到完整高效的操作系统。

（4）无论用户移动客户端还是后台web管理端都应提供首页模块，首页模块集成系统常用功能的快捷入口，以及web端首页的快捷搜索实现对用户管理、订单管理的直接操作，实现“界面内容可见即可用”且无需额外的多余操作，尽量简化用户操作，提高用户体验。

1. 安全性要求

系统安全性是一个系统在实现功能需求后的首要考虑因素。本系统中用户登录采用Oauth2.0的授权登录验证方式，该方式是一种不需要加密的认证方式，同时签名机制也大大简化，不需要特殊的解析处理、编码、和对参数的排序。因其各种特性使用户登录的安全性得到很大的保障。客户端用户通过微信公众号提供的入口进入，首先会经过微信 service的校验，用户合法后才可进入，这也保证了微信账号和系统账号的唯一绑定，同时也大大提升了系统使用的便捷性。

系统数据库采用MongoDB数据库,在数据库部署的时候启用auth配置，当网络收到攻击时，它可以提供“深层防御”。启用SSL可以有效防止数据传输中被恶意的攻击。同时系统也会定时备份数据库数据以保证数据安全。

1. 可扩展性要求

可扩展性是系统设计的重要指标，现在的系统功能大多属于敏捷开发模式，首先会有最小可用版本，之后不断迭代开发新的功能，所以好的可扩展性是至关重要的。本系统拟采用微服务架构模式，各个模块独立开发、单独部署，极大的减小了各个模块之间的耦合性。

* 1. 系统环境要求

1. 硬件环境

系统硬件环境包括：可正常联网的PC设备、CPU为1GHz或以上、内存为2G或以上、硬盘容量为50G或以上；可联网且装有微信APP的移动设备。

1. 软件环境

系统的软件环境包括：操作系统：windows操作系统；数据库：MongoDB数据库；开发工具：Spring Tool Suite、Visual Studio Code; 虚拟容器：Docker。

* 1. 本章小结

本章主要介绍了系统的功能性需求和非功能性需求。功能性需求分别从用户客户端和web管理端两个部分来进行介绍。首先按照用户分类将客户端用户分为家长类用户和教员类用户，并依次介绍了家长用户的个人信息管理模块需求、学生信息管理模块、订单信息管理模块、对教员进行评价的模块，以及教员类用户的个人信息管理模块、订单信息管理模块、上岗证考试模块和星级管理模块等需求。接着对web管理端进行功能性需求分析，包括管理人员对用户和订单的管理模块，以及超级管理员对普通管理员的管理和费用明细统计等需求。非功能性需求分析小节分别从易用性、安全性和可扩展性三方面进行需求分析，最后也说明了系统对软、硬件环境方面的需求。

1. **基于协同过滤的推荐算法**

一个完整的推荐系统，至少需要包含用户的行为/偏好信息和项目资源信息，以及相适应的推荐算法，在我们的系统中教员用户和家教需求订单则分别代表着算法中提到的用户和项目，下文均以用户和项目代指，具体结构如下图4-1所示：



图4-1 推荐系统结构图

对于推荐系统而言在已有文献中已经提出了其形式化定义：设用户集合为

,资源项目集合为,推荐度函数为Res(),推荐度函数即指根据项目i的已有信息并结合用户u的行为/偏好计算出的一个匹配度指数, 是一定范围内的全序非负实数。推荐度的计算则是推荐算法的核心，系统可以根据计算出的推荐度从大到小对资源项目进行排序，并返回前N个资源项目组成的集合Iu给相应的用户，从而生成推荐结果。如下公式4-1所示：

（公式 4-1）

本系统中采用的推荐算法是协同过滤，协同过滤算法的核心思想非常简单，它认为有相似偏好的用户可能会喜欢相似的项目或者对相似的项目表现相似的偏好程度。算法中所提到的用户协同过滤算法包括基于用户的协同过滤和基于项目的协同过滤，使用协同过滤算法首先就需要对用户的偏好或者项目的相关信息进行收集，并将其公式化表示，然后分别计算用户间相似度和项目间相似度，最后根据相似度寻找相似近邻从而产生推荐结果。下面将分别针对这几个步骤对协同过滤算法做进一步的介绍。

* 1. 用户偏好的收集与表示

协同过滤算法中用户的偏好信息是指用户对项目进行的评分，一个评分就是一个用户-项目对，表示用户对该项目偏好程度或者用户与该项目的相关度。而用户对项目的评分又主要分为两种类型，一种是显式评分，一种是隐式评分。

1. 显式评分

显式评分是指用户对项目的直接打分，用户直接表示自己对该项目的偏好程度。在我们的供需对接系统中，用户之间某些属性的直接相关，比如：教员个人信息为大四软件学院在校生，而订单要求中正好有要求家教老师必须是软件学院学生，那么在这个属性值方面两者完全匹配，则可理解为用户为项目的直接打分。用户对项目的直接打分一般分为两种形式，一种是二进制评分，这种评分标准非此即彼，例：赞同/反对，爱/不爱等；另一种是分值形式的打分，常见的有5分制、10分制，用户对项目打不同的分制表示对该项目的偏好程度，分值越高表示用户对这个项目的偏好程度越高，即表示两者匹配度更高。在我们系统中，因为教员用户和订单都有多种属性，所以需要计算这多种属性之间的匹配度或者用户对某个订单属性的偏好度，最后根据不同属性的不同权重来计算出用户对该项目的总体评分。

显示评分的优点在于简单、直接、评分可靠性高，比较容易实现，但是这种评分方式除了硬性属性之间的关联，其他情况都需要用户主动操作，增加了用户额外操作，会影响用户用完即走的用户体验。并且大多数用户不会去参与打分这样会使得可参考评分样本较少，导致偏好信息获取不准确。

1. 隐式评分

隐式评分则是从用户行为间接的来推测用户对项目的评分，系统主要分析用户的历史完成订单、订单浏览历史（点击订单查看详情的才算）和收藏订单列表等行为信息来完成用户对该项目的评分。例如：用户曾多次查看小学数学相关的订单，则系统会认为用户对所有小学数学类型的家教订单感兴趣，从而提高用户对该类订单的偏好程度。

隐式评分的优点在于不需要用户的参与，全程自动化的计算用户对项目的偏好程度，从而减少偏好信息数据稀疏的问题。但其也存在一些缺点，这种方式需要对大量数据进行分析，并且分析的来的结果并不能完全代表用户的主观意愿，可能会产生误导作用。例如：用户因为某些误操作查看了某些订单详情，会被系统误以为用户对此类项目感兴趣，从而产生错误的评分预测。

正常情况下，显示评分的可信度是高于隐式评分的，所以通常综合评分的时候，显示评分所占的权重也应高于隐式评分。这样可以更真实的反应用户对项目的偏好程度。下表4-1给出了我们系统中常用的显式评分和隐式评分的示例，系统可以根据具体应用的环境添加对应的用户行为，并根据这些示例收集、分析用户的偏好。

表4-1 用户偏好程度收集

|  |  |  |  |
| --- | --- | --- | --- |
| 行为 | 类型 | 特征 | 作用 |
| 属性 | 显式 | 用户与订单属性强关联 | 根据属性值是否匹配，精确得到用户-项目匹配度 |
| 教学经历 | 显式 | 用户完善教学经历，生成相应标签 | 通过分析标签得到用户之间相似度 |
| 评分 | 显式 | 数字式分值（0-5分），显示匹配程度 | 家长对教员评分越高，说明教员和订单之间的匹配度越高 |
| 历史订单 | 隐式 | 布尔化偏好，取值0/1 | 通过历史订单，精确得到用户偏好 |
| 浏览历史 | 隐式 | 布尔化偏好，取值0/1 | 通过查看历史订单，不精确但一定几率上得到用户偏好 |
| 收藏列表 | 隐式 | 布尔化偏好，取值0/1 | 通过订单收藏列表，不精确但一定几率上得到用户偏好 |

用户集合为,项目集合为,收集到用户偏好信息后，通过对数据进行处理，可得到用户对项目的偏好程度为，进而得到一个的用户-项目偏好矩阵,如下表4-2所示：

表4-2 用户-项目偏好矩阵

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Item1 | … | Itemi | … | Itemn |
| User1 | r11 | … | r1i | … | r1n |
| … | … | … | … | … | … |
| Useri | ri1 | … | rii | … | rin |
| … | … | … | … | … | … |
| Userm | rm1 | … | rmi | … | rmn |

系统对已有用户、项目进行偏好收集之后，则需要根据这些已有数据对新加入的用户或者项目进行预测评分，从而根据评分对候选项目进行推荐度预测。此时被预测评分的项目i则称为待预测项目（To-be-predicted Item）或目标项目（Target Item）,与之计算匹配度的用户u则称为活动用户（Active User）或目标用户（Target User）。最后根据预测推荐度产生推荐列表。

* 1. 相似度计算

协同过滤算法主要分为两类即基于用户的协同过滤和基于项目的协同过滤，这两者之间的主要差别就是前者是根据用户偏好信息计算目标用户和其他用户之间的相似度，然后根据相似度为目标用户选择相似近邻，最后可将近邻项目的推荐结果也推荐给目标用户；而后者则是根据评分矩阵计算项目间的相似度，然后根据相似度为目标项目选择相似近邻，最后根据目标用户对近邻项目的评分来预测目标用户对自己的评分，从而生成推荐结果。下面分别介绍一下用户间相似度和项目间相似度的计算方式。

1. 用户间相似度计算

计算用户间相似度是为目标用户寻找相似近邻的基础，主要运用了相似统计的方法。从上文的用户偏好评分可以得到一个的用户-项目评分矩阵，其中用户u对所有项目的评分可以看成是一个评分向量,用户之间相似度是通过计算两个用户之间的评分向量得到的。我们常用的根据评分矩阵计算相似度矩阵的方法有余弦相似度（Cosine Similarity）和Pearson相关系数（Pearson Correlation Coefficient）两种。在评分矩阵中横纵坐标分别代表用户和项目，而在相似度矩阵中，横纵坐标都是用户，两个用户的交叉处则代表用户u和用户v之间的相似度，对角线上元素表示用户和自身的相似度，因为用户自身不参加与自己的相似度比较，所以值全部默认为0，具体关联如下图4-2所示：

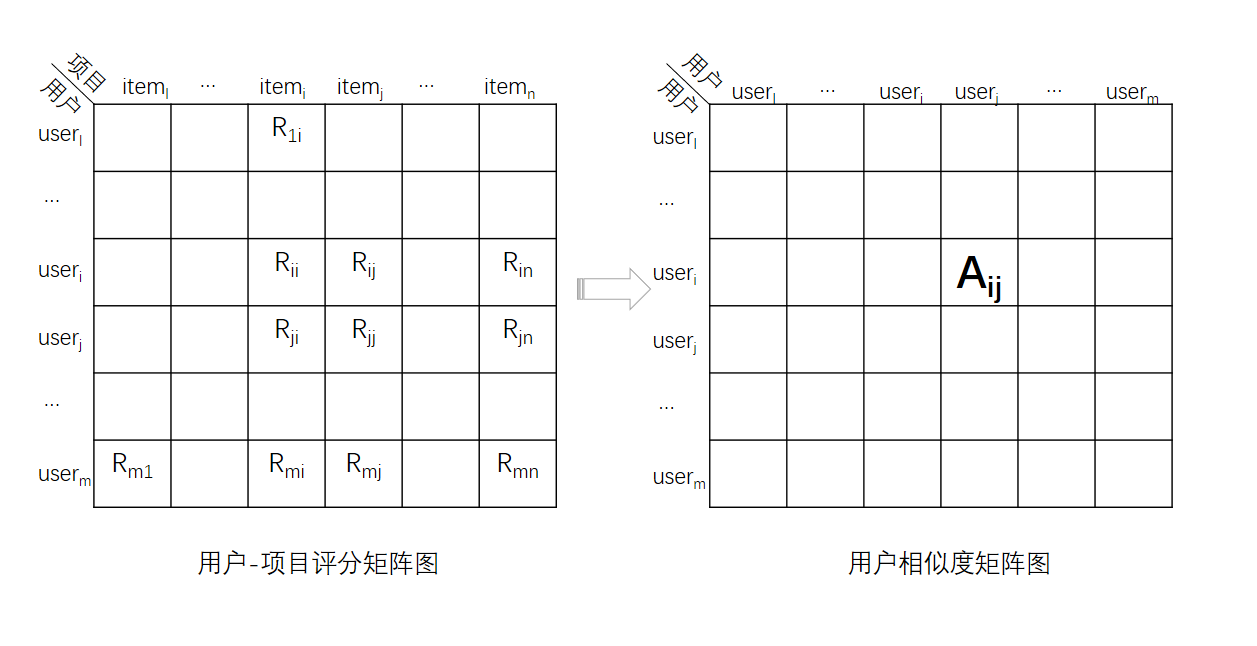


图4-2 用户相似度矩阵图

余弦相似度：将用户对所有项目的评分看成一个维向量，用户之间相似度是通过两个向量间的夹角余弦计算出来的，夹角越小用户间的相似度越高，向量余弦计算公式如下：

(公式 4-2)

上面公式中，表示用户u和用户v之间的相似度，向量分别表示用户u和用户v对所有项目的评分向量，分别表示用户u和用户v评分向量的模，分别表示用户u和用户v对项目i的评分，则分别表示用户u和用户v的已评分项目集以及两个用户的共同评分项目集。

Pearson相关系数计算两个用户评分向量之间的相关系数，公式如下：

（公式 4-3）

上面公式中，分别代表用户u和用户v的平均评分，其他表达式和余弦相似度计算公式中的一致。

对于同一个评分矩阵不同的相似度计算公式计算出来的结果也不一定相同，有时甚至会出现相反的情况，如下表4-3所示，用户1和用户2的相似度通过余弦相似度公式计算，两者的相似度只有0.24，但是通过Pearson相关系数公式计算的话两者完全相似。并且相似度是不可以传递的，比如用户1和用户2相似，用户2和用户3相似，我们并不能得出用户1和用户3也相似的结论，可能会因为其他因素导致相似度较低。

表4-3 用户-项目评分表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 项目1 | 项目2 | 项目3 | 项目4 | 项目5 | 项目6 |
| 用户1 | 0 | 4 | 2 | 0 | 0 | 4 |
| 用户2 | 2 | 0 | 3 | 0 | 2 | 0 |
| 用户3 | 0 | 5 | 2 | 4 | 0 | 4 |
| 用户4 | 3 | 4 | 0 | 5 | 2 | 0 |
| 用户5 | 0 | 4 | 4 | 0 | 3 | 0 |

1. 项目间相似度计算

项目间相似度的计算与用户间相似度的计算类似，项目间相似度是将所有m个用户对同一个项目i的评分看成一个评分向量，两个项目之间的相似度是通过两个项目的评分向量计算得来的。计算方式同用户间相似度计算，主要有Pearson相关系数、余弦相似度计算以及修正余弦相似度三种。项目相似度矩阵图是一个横纵坐标都是项目的矩阵图，表示项目i和项目j之间的相似度，具体如下图4-3所示：

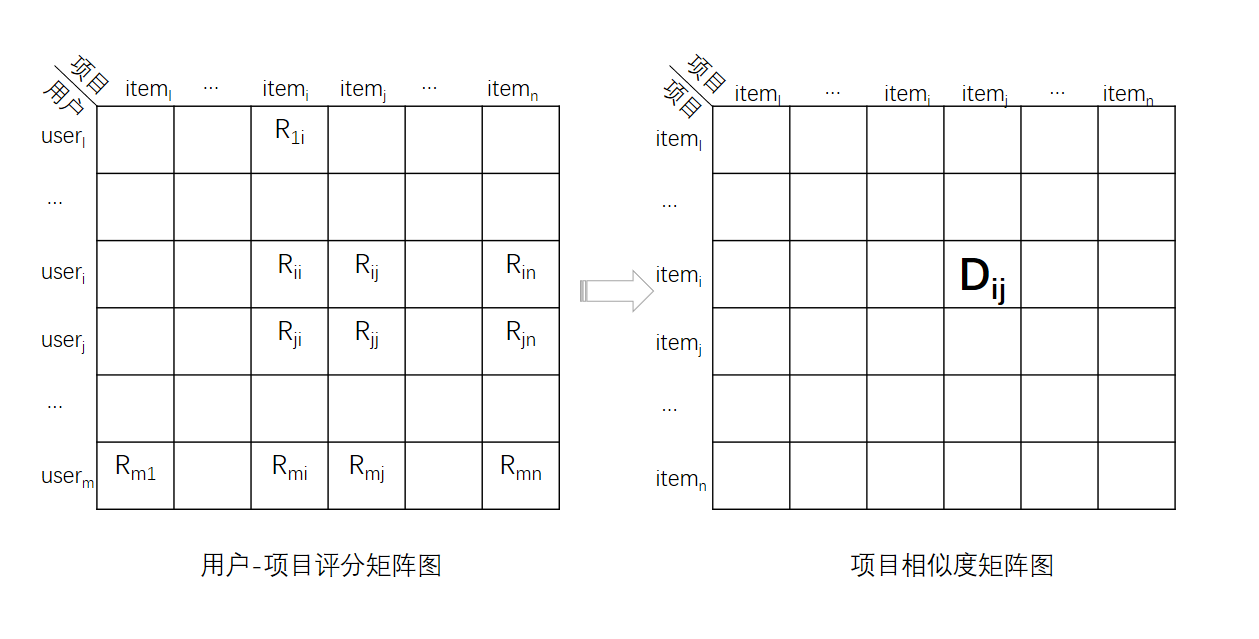


图4-3 项目相似度矩阵图

余弦向量：与用户余弦相似度计算不同的是，将所有用户对项目的评分看成维向量，计算公式如下：

（公式 4-4）

上面公式中，代表项目i和项目j之间的相似度，表示既对项目i进行过评分又对项目j进行过评分的所有用户的集合，和则分别表示用户u对项目i和项目j的评分，其他公式类似于计算用户间相似度所代表的含义。

Pearson相关系数计算两个项目之间相似度的公式如下所示：

（公式 4-5）

修正余弦向量：有的情况下认为我们在余弦向量相似度的计算中没有考虑不同用户对同一项目的评分标准问题，有的用户打分比较随意，而有的用户打分则要求比较严格，所以为了提高相似度的稳定性，提出了一种新的修正余弦向量方法。在修正余弦向量中，我们首先把所有用户的评分减去平均分，然后再进行余弦向量相似度计算，具体公式如下所示：

（公式 4-6）

项目间相似度和用户间相似度有着同样的性质，即同样的评分矩阵根据不同的相似度计算方法得出的相似度可能会有很大差别，同样相似度也不具有传递性，具体实例同用户间相似度的描述，就不单独举例说明。

* 1. 寻找相似近邻

得到用户间相似度或项目间相似度后，需要根据相似度系数为目标用户或目标项目找到多个相似近邻。用户的相似近邻和项目的相似近邻求法基本一致，因为本系统中是主动将订单推送给教员，所以本节以项目的相似近邻为例进行算法介绍。求相似近邻最常见的方法有以下两种：

1. K最临近法

这种方法就是不论项目与目标项目的相似度是多少，只选择与目标项目v相似度最大的前k个项目作为他的相似近邻，而不考虑相似度的具体大小。如下图4-4所示，要为项目1找寻4个相似近邻，则直接根据所有项目与项目1之间的距离从近到远依次选取4个而不考虑他们与项目1之间相似度的具体大小，可得近邻项目为项目2、项目4、项目5和项目6。

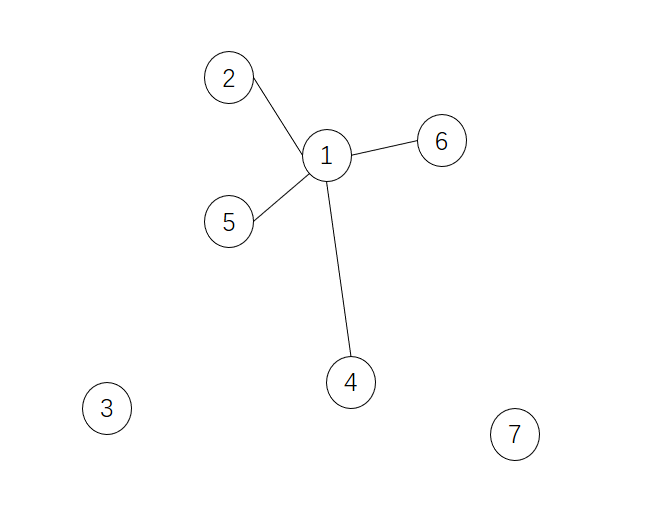


图4-4 K临近法

1. 设定阈值法

设置阈值法和K临近法有所不同，这种方法会设置一个相似度的阈值，如果项目i和目标项目之间的相似度超过阈值的话，则项目i被选定为目标项目的相似近邻，否则则不是目标项目的相似近邻。实现方法是以目标项目i为中心半径为的圆内的所有项目都是该目标项目的相似近邻。这种方法获得的相似近邻数量不确定，但他们与目标项目之间的相似度一般不会有较大的差别，阈值的选取则需要结合大量数据、多次试验后慎重确定，当阈值过小时可能会出现找不到相似近邻的情况，这种情况下需要相应的调整阈值的大小，以扩大搜索范围。如下图4-5所示，项目之间相似度分布同上一示例，现在将邻近算法改为阈值法，设定阈值为，则可得到项目1的相似近邻为项目2、项目5和项目6，没有了项目4。

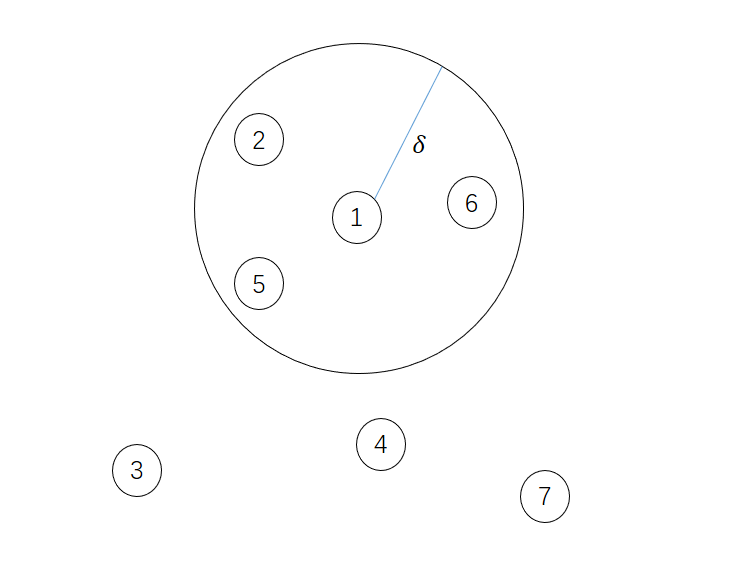


图4-5 阈值法求相似近邻

* 1. 生成推荐结果

找到用户的相似近邻后，根据相似近邻的评分信息为目标用户进行推荐，需要根据目标用户对近邻项目的评分，预测目标用户对待预测项目评分。常用的预测方式有，直接对近邻的评分加权平均或者先计算近邻评分的增量，然后再加权平均，分别用到如下两个公式：

（公式 4-7）

（公式 4-8）

上面公式中，表示用户u对项目i的预测评分，N指的是项目i的临近项目集，是用户u对项目j的评分，是项目i和项目j的相似度，, 分别表示项目i和项目j的平均评分。

得到用户u对项目i的预测评分后，根据预测评分生成Top-N的推荐列表。假设目标用户u的已评分项目集合为，为每个项目 找到p个近邻，组成近邻集，目标用户u的候选项目集为，对,预测用户u对该项目的评分，然后根据从大到小排序选取前N个项目生成用户u的Top-N推荐集。

* 1. 本章小结

本章主要介绍了基于协同过滤算法在本系统中的实际应用。通过分析生成推荐的整个流程，详细对每一个环节进行介绍。第一节介绍了用户对项目的偏好收集以及表示形式，介绍了偏好收集的显式评分与隐式评两种形式，以及最后得到的偏好矩阵。第二节主要介绍了根据用户对项目的评分矩阵计算用户间相似度和项目间相似度的两种方法，即余弦相似度方法和Pearson相关系数方法，并且在项目相似度计算中增加了修正余弦向量的相似度计算方法。第三节则介绍了如何根据用户间相似度以及项目间相似度采用K临近法或阈值法计算相似近邻。最后一节则讲解如何根据相似近邻预测待评测项目得分，从而生成推荐结果。整章内每一小节环环相扣，并且结合项目实际情况对协同过滤推荐算法的实现进行介绍。

1. **微服务架构系统设计与实现**
   1. 系统设计
2. 系统总体结构设计

所有的架构设计都是基于一定原则的，微服务架构也不例外。为了充分发挥微服务架构的优势，在设计开发时尽量要遵循以下四个原则：

1. 在开发之前对数据进行拆解
2. 在原型基础上逐渐迭代扩大
3. 关注内部服务之间的通信细节
4. 确保开发技能满足架构需要

从第三章的系统功能性需求分析，我们可以清楚的了解一个完整的系统需要包含哪些模块，每个模块都有自己的数据主体。因为系统需求分析做的比较充分，我们可以把当前所需实现功能当做第一版原型进行开发，之后有新的功能需求，便在此基础上进行迭代开发。所以本系统的设计应该着重于服务之间的通信细节，以及在当前技术可实现的范围内进行设计。根据第三章的功能需求分析我们可以将系统整体拆分为如下微服务应用：

1. 基础服务

UI-Server Service（静态资源服务）

Auth-Server Service（认证服务）

Config-Server Service（配置服务）

Service-registry（服务注册模块）

API-Gateway Service（网关服务）

1. 业务服务

User Service（用户服务，包括家长和教员）

VerifyCode Service（验证码服务）

Requirement Service（家教订单服务）

Star Service（星级服务）

Evaluation Service（评价服务）

Dispatch Service（订单匹配服务，推荐算法实现模块）

Push Service（消息推送服务）

Payment Service（收支明细服务）

在上面划分的微服务应用中，基础服务是微服务架构必不可少的，UI-Server Service用来提供一切静态资源主要以前端资源为主；Auth-Server Service主要负责用户身份认证，是保证系统安全性必不可少的，也为系统提供了权限控制；Config-Server Service用来提供所有服务的配置信息，也是每个服务启动之初需要访问的服务，需要最先部署；Service-registry则是为所有已部署服务提供注册功能，只有在这里注册的服务才能通过被其他服务发现进而进行通信；API-Gateway Service（网关服务）顾名思义则是接收所有的请求然后根据不同服务的注册id将请求分发至指定服务，是整个微服务架构的调度中心。业务服务则是根据不同的系统功能需要，可以进行扩展、修改、删除。

所有服务拆分好之后，也就是明确了每个服务自己所负责实现的功能，接下来就是各服务之间如何通信，良好的通信机制可以使各个服务构成一个完整的整体。每个服务通信时都会携带token并通过通过访问OAuth2 Authorization Server来验证请求是否合法、是否满足权限要求。用户请求发送至API-Gateway Service，在网关服务模块我们通过Zuul来做负载均衡和路由管理。

因为不同的服务是部署到不同的Docker容器中的，由于Docker容器的隔离机制，我们可以为每个服务选择适合自己的数据库，User Service，Requirement Service这类的属性信息存储我们可以使用MongoDB数据库，Evaluation Service，Payment Service这类数据之间关联的存储我们可以选择MySQL这类的关系型数据库。

各服务部署也要遵循相应的顺序，首先要部署的是Config-server服务，该服务包含所有服务的配置信息，其他服务启动时都需要先来这里获取自己的配置；接着依次部署Service-registry（注册服务）、API-Gateway（网关服务）和其他业务服务。所有服务（包括最先启动的Config-server）都需要在Service-registry服务注册，只有注册过的服务才会被API-Gateway服务，进而分发相应的请求至对应的服务，各服务模块之间的数据流图如下图5-2所示：

根据以上分析设计我们可得到系统的整体架构设计如下图5-1所示：

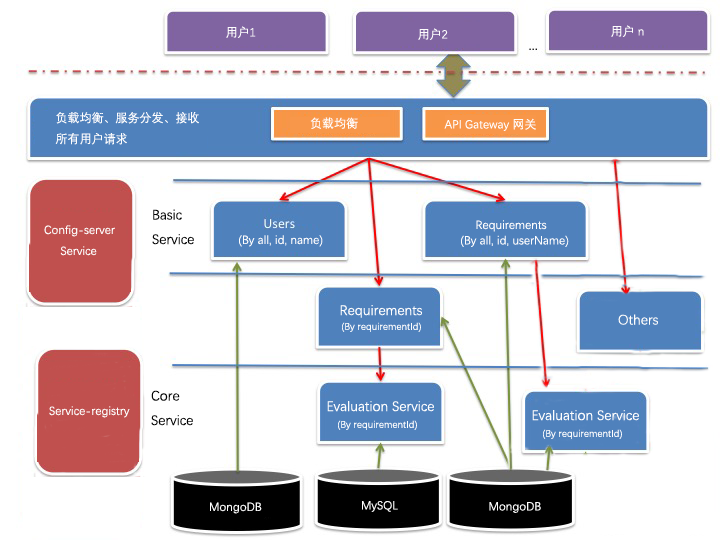


图5-1 系统整体架构设计图

1. 系统详细设计

根据系统整体架构设计，我们按照先基础后业务的流程依次进行个服务模块的设计，具体分析设计如下：

1. Docker环境搭建设计

为了方便开发和运维可以快速的了解服务所需的运行环境，我们通过使用Dockerfile来构建Docker镜像，通过Dockerfile可以快速在生产环境中进行部署。我们也可以通过Docker Compose更方便的配置和管理多个容器，在配置文件中配置你所需要的服务，之后就可以通过命令快速的对已配置服务进行创建、启动等操作。

1. 配置服务设计

一个软件的完整生命周期包括开发、测试、生产，每个产品阶段也都有对应的环境配置，所以我们需要一种分布式的配置集中管理方式。配置服务器是一个外部配置，属于配置的中心，各个服务的不同阶段所需环境配置都可以统一存放在这里。其他服务在部署或迁移的时候，可以直接来配置中心拉取自己所需的配置信息，快速的进行环境搭建，从而可以方便的管理开发、测试以及生产环境的配置。

本系统中将采用git存储配置信息，这也是配置服务器最常用的一种形式，git repo也是免费开源的，并且简单易操作。Git上配置信息的定义规则如下：

* /{application}/{profile}[/{label}]
* /{application}-{profile}.yml
* /{label}/{application}-{profile}.yml
* /{application}-{profile}.properties
* /{label}/{application}-{profile}.properties

配置中心存放了各个微服务的配置，对于每个服务来说，resource文件夹下的bootstrap和application文件是本地的配置文件，并且bootstrap的优先级高于application，所以一般在bootstrap.yml中指定配置中心的地址，通过编写代码可以让服务自动从配置中心获取配置。

例：想要获取mircoservice-config-requirement-pro.properties的配置信息，可直接通过http:// ${address}/microservice-config-requirement-pro.properties请求直接获取查看。

1. 服务注册与服务发现

本系统中每一个服务都是一个应用实例，随着系统版本的迭代开发，实例数量并不是一成不变的，每个服务实例在Docker容器中部署后都会动态分配一个IP以及端口号，所以系统在启动时需要告诉其他服务，该系统有多少个实例，应该如何去与之通信。

通过对服务发现技术的一些调研，我们决定采用Eureka客户端发现模式。Eureka模式分为Eureka Server和Eureka Client两种，前者是一个服务注册中心，当服务启动时需要向服务注册中心注册服务，获取中心注册表副本；服务注册成功后，Eureka客户端会定时的通过发送心跳来刷新自己的状态；服务停止后，Eureka客户端需要去服务注册中心注销该服务。其主要流程如下图5-2所示：

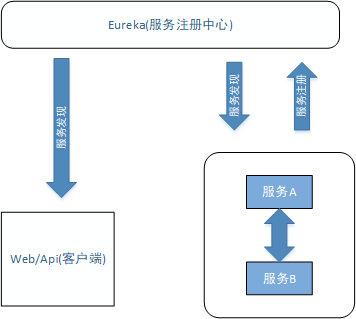


图5-2 Eureka服务注册发现

1. 服务网关

在一个系统应用中包含如此之多的服务实例，客户端可以向每一个有访问权限的的服务发送请求，而随着开发的不断迭代，我们可能需要合并服务、拆分服务、进而改变系统整体的服务划分，客户端和服务直接交互的话意味着一方改变另一方也需要随之改变，这样不利用系统业务的重构。

因此我们需要一个API Gateway(服务网关)来将外部公共API和内部微服务API分开，这样可以在不对外界客户端产生影响的前提下进行内部服务API的调整，同时为我们的微服务提供单一入口点，对客户端隐藏服务发现等详细信息。客户端根据需要定制API接口，所有请求信息都通过API Gateway与内部服务通信，也可以将其看成系统的唯一入口。我们也可以方便的在这唯一入口提供系统的授权验证、负载均衡、静态响应和缓存等功能。

通过API Gateway我们可以暴露一个粗粒度的API给客户端，客户端发送一个聚合的url过来，网关将url进行拆分并去多个服务模块获取数据。最后将涉及到的所有服务返回数据全部返回给客户端，这些对客户端都是不可见的，客户端也无需知道是如何实现。

1. 系统安全性设计

传统系统架构中，系统在接受到客户端请求，经过负载均衡后，到达auth层，进行身份验证和权限批准。验证通过之后分发到相应的Controller中执行逻辑操作，最终返回数据给客户端。这种模式下后端拥有所有的身份验证，只要入侵某个请求就有可能获得后端所有的信息，应用的每次验证都需要跟数据库交互，造成数据库负载较高。

所以在本系统中，应独立划分每个微服务的权限范围，请求只能访问权限内的功能和资源。并且通过OAuth2.0来进行权限验证，其主要思想是基于token的验证。用户登录后，发送包含用户名和密码的credential请求，系统接收到请求，通过Auth-server服务生成带有有效期的token字符串返回给客户端。拿到token后客户端再发请求获取资源，后端转发带token的请求到Auth-server，如果通过校验，就从资源服务器获取资源返回给客户端，否则返回无授权信息。

我们在项目中使用cloud-security和security-oauth2这两个依赖，为我们提供了非常大的便利。OAuth2.0规范定义了4种授权方式来获取令牌，项目里使用的是Authorization Code授权码类型。

* 1. 系统总体实现

1. Docker环境搭建

Docker容器的使用是我们整个系统中所有服务部署的基础，每个微服务都需要部署到自己的Docker容器中。下面就以部署系统中订单服务所需的MongoDB容器为例进行详细使用说明：

1. 获取镜像

Docker Hub是一个docker镜像库，首先我们可以通过docker pull来下载docker镜像 sudo docker pull centos,因为Docker Hub是国外服务器，如果下载过慢可以通过国内第三方库进行下载，运行命令：

docker pull docker.io/frodenas/monodb

你也可以直接采用 $docker run + 镜像名 的方式运行某个镜像，docker首先会在本地查找是否有该镜像，如果不存在的话docker会自动从Hub上获取。也可以通过 $docker images 查看自己当前拥有的所有镜像。

1. 启动容器

获取到镜像后，通过执行以下命令，在这个镜像上启动一个容器。

* 首先将镜像绑定到27017端口：

docker run -d --name mongodb -p 27017:27017 docker.io/frodenas/mongodb

* 使用docker ps命令查看是否启动，这个镜像启动的时候会创建一个默认的mongoDB用户和密码，以及一个默认的数据库，可以通过docker logs查看详细信息。
* 现在，一个在容器作为守护进程运行的 mongoDB 容器已经成功运行，并且容器的监听端口27017绑定到了宿主机的27017端口。进入容器： docker exec -it [id] /bin/bash

在容器内执行命令连接数据库。

* 可通过 use 命令切换指定数据库，如果该数据库不存在，则创建。

1. 使用Dockerfile制作镜像

本地开发时创建了docker镜像，在将服务部署到服务器上时，如果还需要重新将创建镜像的操作执行一遍，无疑会显得效率极其低下并浪费大量时间精力，所以我们需要通过Dockerfile来将对镜像的每一步操作包括安装、修改、构建等全部记录下来，当迁移到其他环境的时候只需一键执行脚本就可以定制镜像了。

定制镜像的前提是已有镜像基础，如下图5-3所示是requirement（订单）业务服务的dockerfile文件：

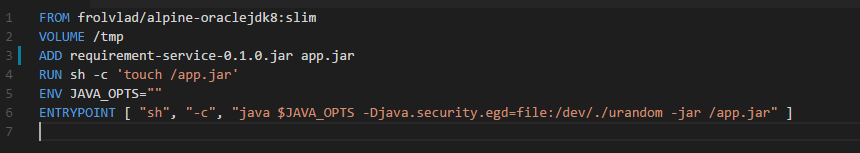


图5-3 Dockerfile相关设置

其中FROM是基于已有镜像的基础，且这条命令必须是镜像文件的第一条指令。VOLUME可以将本地文件夹或者其他容器的文件夹挂载到容器中。ADD 指令会自动把压缩文件解压到目标路径。ENTRYPOINT 目的是指定容器启动程序及参数。执行 docker build -t [镜像名]，构建好之后执行docker run –p [端口1]:[端口2] [镜像名]，访问http://${address}:${端口2}就可以访问服务了。

1. 服务配置中心搭建

（1）配置中心是所有微服务运行的基础，每个微服务启动时都需要通过配置中心来获取自己服务所需的所有配置文件。

* 首先新建config-server项目，引用相关依赖资源。
* 在git上新建radar-config，所有微服务的配置文件全部放在这里。配置文件的命名规范为“[服务名称].yml”,项目名称如果配置错误的话，将会找不到相应的配置文件。
* 在启动类中使用@EnableConfigServer 注解，以达到激活配置服务的目的。

配置config-rver的application.yml文件，具体配置如下图5-4所示，其中uri是配置文件的仓库地址：

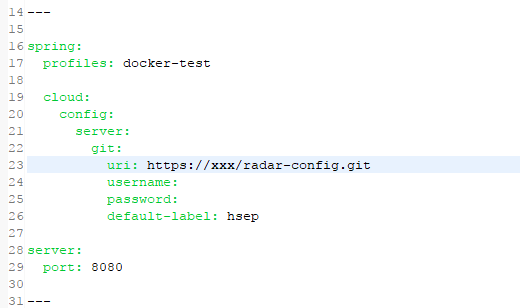


图5-4 config-server相关配置

（2）对于其他业务服务而言，启动时都需要访问config-server，所以在其bootstrap.yml的配置文件中都应进行如下图5-5所示配置：

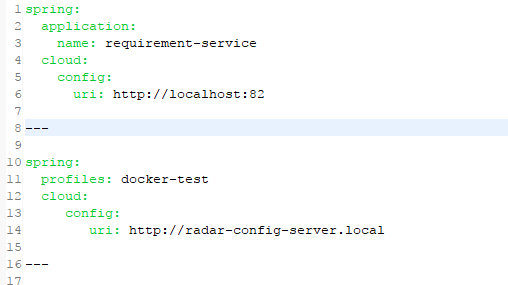


图5-5 其他微服务配置文件

Uri所示即为配置中心所在地址，并且spring.application.name要和radar-config 中的配置文件名字一致。

1. 服务注册与服务发现
2. 创建服务注册中心

首先创建一个基础的Spring Boot项目，命名为Service-registry，并在pom.xml中引入必须的依赖内容如下图5-6所示：



图5-6 注册服务依赖示例代码

通过@EnableEurekaServer注解启动一个服务注册中心，提供给其他应用进行对话。默认配置下该注册中心也会将自己作为客户端进行注册，我们需要在application.properties配置文件中通过增加eureka.client.register-with-eureka=false设置来禁止此行为。

1. 创建服务提供方

创建提供服务的客户端即我们之前设计中提到的基础服务和业务服务，并向注册中心注册自己。首先创建一个Spring Boot项目，这里以评价服务为例，pom.xml文件中引入必须的依赖，具体引用如下图5-7所示：

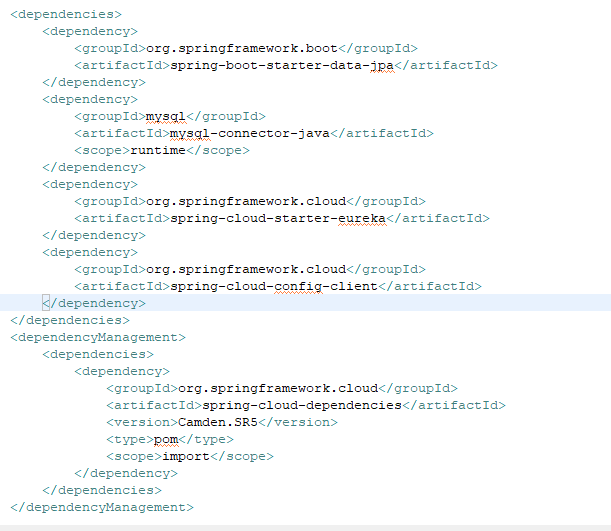


图5-7 服务提供方依赖示例代码

在具体的微服务主类中通过加上@EnableDiscoveryClient注解，就可以激活Eureka中的DiscoveryClient实现，这样就可以实现Controller中对服务信息的输出。

通过spring.application.name属性，我们可以设置微服务的名称，在之后的调用时只需要使用该名称就可以进行服务的访问。eureka.client.serviceUrl.defaultZone属性对应服务注册中心的配置内容，指定服务注册中心的位置。为了在本机上测试区分服务提供方和服务注册中心，使用server.port属性设置不同的端口。启动服务后，如果可以在注册列表中看到我们的服务，就说明我们定义的服务被成功注册了。服务注册后，Eureka Client会定时的发送心跳来刷新服务的最新状态。

1. 服务网关的实现

通过服务网关（API Gateway）可以统一向外部服务提供RESTful API，可以自己控制API的粗细粒度，进一步将内部实现与外部服务可见接口隔离。API Gateway服务的构成就比较简单明了，只需要有启动类而不需多余的业务逻辑。

* 相关pom.xml依赖

API Gateway的依赖需要包含spring-cloud-starter-zuul，具体如下所示：

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-zuul</artifactId>

</dependency>

启动类需要加上@EnaleZuulProxy注解来激活zuul。

* application.yml相关配置

配置文件中需要指明端口号和注册中心位置。也可以自定义路径，外部通过访问配置的path路径就可以访问指定的service。

* 1. 具体业务服务模块实现

在微服务架构中，不论是提供服务的还是消费服务的，每个微服务都是一个完整的Java Spring Boot项目，有自己的配置文件、依赖资源等，并且遵循自己的实体层（Entity）、逻辑层（Service）、存储层（repository）和控制层（Controller）的项目结构。对于构建每个微服务来说其总体流程是相似的，需要设计好数据存储的表结构、API接口类、具体的业务逻辑实现以及OAuth对接口进行的安全设计。下面以评价服务（Evaluation Service）为例，从以上几点对一个Spring Boot项目的实现进行详细介绍。

1. 业务基本数据库设计

在评价服务（Evaluation Service）中，主要是记录什么人在什么时间对谁做了怎么样的评价。因此我们应该选择关系型数据库来对这一系列数据进行存储。在我们系统中主要是对用户或者家教订单的评价，而用户和订单的数据模型在他们相应的服务中都有定义，就不过多叙述。这里主要是设计一个Evaluation对象来存储评价记录，并设计一个Target对象来存储当前被评价对象。具体表结构如下表5-1，表5-2所示：

表5-1 评价记录Evaluation数据结构表

|  |  |  |
| --- | --- | --- |
| id | long | 记录id |
| target | Object | 评价的是什么（人/问题/其他）包括type和id |
| type | string | 评价类型 |
| score | int | 评分,1-5分 |
| comment | string | 预留的评论 |
| time | long timestamp | 什么时候评的 |
| author | string | 谁评的(user id) |

表5-2 评价对象Target数据结构表

|  |  |  |
| --- | --- | --- |
| target\_id | long | 被评价对象的id |
| target\_type | string | 被评价对象的类型（家长、教员、订单） |
| target\_name | string | 被评价对象的name(用户名、或者订单名) |

1. 业务主要接口设计

该微服务应用评价服务提供的是评价类服务，包括评价记录的创建、删除、修改以及查看获取。根据基本功能可得该服务的接口设计类图如下图5-8所示：

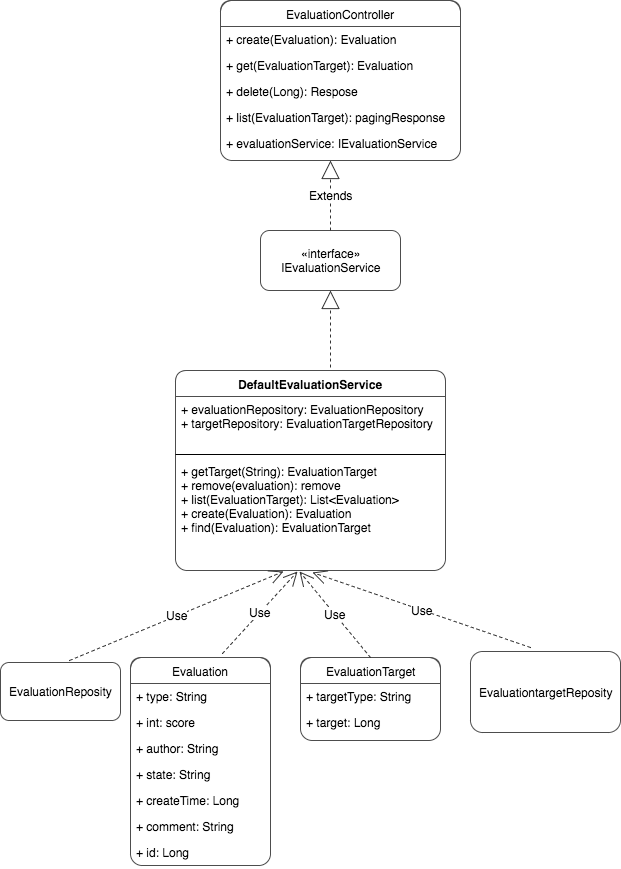


图5-8 评价服务接口API类图

1. 功能实现

项目框架搭建好后，开始功能开发。开发过程不再论述，强调开发过程中需要注意的几个要点：

* 启动类需要加上@SpringBootApplication和@EnableEurekaClient注解，后者的作用是在服务启动时把自己注册到服务注册与发现中心。
* Entity层按照设计实现evaluation和evaluationTarget类，Repository层结合jpaRepository实现按照id获取评价。
* Controller层实现根据evaluation id对evaluation记录进行增删改查，查询返回的评价列表数据支持分页处理。
  1. 本章小结

本章主要介绍了微服务架构系统的设计与实现。以及单一的Spring Boot微服务应用的设计与实现。第一节主要从微服务架构设计需要遵循的几个要点逐条的整体设计进行阐述，包括根据需求分析如何对微服务进行拆分。最后对微服务架构实现流程分别要实现的模块逐个进行设计。第二节主要介绍了微服务系统的整体搭建实现，包括Docker环境的搭建、微服务配置中心的实现、服务的注册与发现以及服务网关的实现。第三节则主要介绍了具体微服务应用的设计与实现。包括单一Spring Boot项目实现过程中的数据库设计、接口类设计以及最后业务逻辑实现的介绍。全章从设计和实现两个方面对微服务架构进行了一个全面且透彻的分析，并成功的在我们系统中得以应用。

1. **系统测试与结论**
   1. 系统测试
2. 功能测试

系统功能测试，主要包括开发过程中的单元测试和项目完成后对系统进行实际操作的测试。单元测试在软件开发过程中是非常重要的，它可以帮助我们尽早的发现程序中的bug和不足，有研究指出在软件开发中bug隐藏的时间越久，修改它时所付出的代价就越大。本系统因为是采用的是基于Java语言的项目开发，所以我们采用Junit4来进行单元测试。在我们的开发工具Spring Tool Suite 中集成有Junit4组件，我们可以方便的进行测试用例的编写。

单元测试即是对系统涉及到的主要功能进行测试，根据需求所应达到的目标和我们实现后所返回的结果进行对比，进而判断我们的开发是否存在bug。

下面以订单管理的单元测试为例，介绍如何在开发过程中进行单元测试，测试用例实例如下表6-1所示：

表6-1 订单管理的单元测试实例

|  |  |  |
| --- | --- | --- |
| 编号 | 功能点 | 测试用例 |
| 1 | 新建订单 | 1. 输入所有属性字段，期望返回新建成功 2. 只输入必填项属性字段，期望返回新建成功 3. 必填项为空，期望返回新建失败 4. 输入信息中属性字段不符合要求，期望返回新建失败 5. 插入空对象，期望返回新建失败 |
| 2 | 删除订单 | 1. 传入已有订单id，期望返回删除成功 2. 传入不存在订单id,期望返回找不到订单 3. 传入已删除订单id，期望返回订单已删除不能重复操作 |
| 3 | 修改订单 | 1. 传入正确订单id,数据字段为空，期望返回修改成功，订单无变化 2. 传入正确订单id,以及可修改字段值（与原本值一直或不一致），期望返回修改成功，字段新值是传入值 3. 传入正确订单id,修改字段为不可修改状态（已完成匹配状态订单修改至待处理状态），期望返回修改失败，提示该状态不可修改 4. 传入错误订单id，期望返回修改失败，找不到该订单 |
| 4 | 获取指定订单 | 1. 输入正确订单id，返回期望订单数据 2. 输入错误订单id，返回该订单不存在 |
| 5 | 获取订单列表 | 1. 传入多属性参数，期望返回所有符合这些参数的订单 2. 传入参数中包含未知参数，期望返回符合已有参数的所有订单 |

1. 用户界面测试

用户界面测试包括用户使用的移动客户端和管理员使用的web管理端两部分的测试。其主要目的是检测用户界面和相关元素组件的显示情况、是否存在设备/浏览器兼容情况以及该有的功能入口是否可见等。用户界面的测试，主要是为了检测在需求分析时对用户体验和交互性所做的设计是否都有体现，是产品交付的前提。

根据系统的设计要求，给出以下界面测试内容，列出的只是部分并不仅限于此，详细介绍如下表6-2所示：

表6-2 用户界面测试内容

|  |  |
| --- | --- |
| 编号 | 检查项 |
| 1 | 各界面元素文字是否正常（标题栏、正文等字体格式、大小、颜色等） |
| 2 | 按钮排列是否合理，是否易于点击 |
| 3 | 输入框点击是否正常，输入内容超出预期是显示是否有好 |
| 4 | 下拉框、选择框等是否正常（无效、选中等状态），数据回显是否正常 |
| 5 | 确认、删除等操作是否有二次确认提示，是否容易误操作 |
| 6 | 弹出对话框内文字意义是否明确 |
| 7 | 提示信息（非法输入或操作返回）是否规范 |
| 8 | 功能入口是否都比较容易发现、放置是否合理 |
| 9 | 窗口在不同分辨率下是否能够全部显示 |
| 10 | 快捷入口是否有效 |
| 11 | 屏幕内显示不下时是否有增加滚动条确保显示完整 |

1. 安全性测试

系统的安全性测试对于系统是否可以投入生产环境有着极大的意义。如果系统安全不过关，导致非法用户的入侵或者数据库的数据泄露，将会造成极大的损失。所以我们本节的安全性测试主要包括用户认证安全测试和数据库安全测试两个方面。主要是考察系统在受到安全性攻击时的抗风险能力。所以针对以上两点给出如下测试内容，详细介绍如下表6-3所示：

表6-3 系统安全性测试内容

|  |  |
| --- | --- |
| 安全测试分类 | 检查内容 |
| 用户  认证  安全  测试 | 1. 用户名、密码的校验是否准确 2. 是否可以正常获取到授权码 3. 是否可以正常获取到access\_token 4. access\_token过期或超过有效期是否可以使用 5. 请求头中没有access\_token是否可以访问资源 6. 系统中用户权限是否区分明确 7. 是否可以通过其他途径登录系统 8. 不通过登录页面直接访问子页面url是否会跳转权限校验 9. 系统中是否会出现用户冲突 |
| 数据库  安全测试 | 1. 数据库身份认证（检测用户身份是否合法） 2. 系统数据是否机密（数据内容是否有加密处理） 3. 系统数据是否完整（数据库是否备份并且可恢复） 4. 数据库操作日志记录是否正常工作 |

* 1. 系统测试结论

根据上述具体测试用例和检测内容对系统从功能、界面以及安全性三方面进行系统性的测试，测试结果表明：

1. 系统中各功能模块运行正常满足系统的所有业务需求，数据显示准确，运行流畅，响应较快。
2. 系统用户界面布局合理，功能布局符合操作习惯，有着很好的交互体验，各种提示信息也比较准确完善，页面切换流畅，刷新快捷有着良好的用户体验。
3. 系统安全性方面，有着较为完善的身份认证和权限校验体系，登录安全，并且数据库定期备份数据可以有效的保护用户数据安全。

综上所述，该系统的设计实现，完全符合用户需求，并且有着较好的用户体验、安全性和可扩展性。

1. **总结与展望**
2. 总结

随着互联网的高速发展，教育也要与时俱进，本文根据对当下教育资源供需对接现状的分析，并结合微服务架构、Docker等最新技术在实际项目中的应用，设计了这样一个基于微服务架构的家教在线匹配系统。并且对基于协同过滤的推荐算法进行研究，结合项目实际，实现推荐算法在系统中的应用。成功的将原本的线下对接模式转到线上，更高效的为家教服务方推荐匹配度最高的家教需求，实现需求双方在线匹配、双向互选。本文主要分为六章，主要完成的研究工作有：

1. 对家教需求在线匹配的研究意义以及微服务架构、推荐算法等技术的研究现状进行分析。
2. 对系统中所用到相关技术进行分析和研究，寻找最佳实践，主要包括微服务架构实现相关技术和常见推荐算法的研究。
3. 从功能性、非功能性和系统环境三个方面对系统的需求进行分析，并确定系统的总体流程。
4. 对协同过滤算法的实现进行研究，并结合项目具体情况调整部分参数，有效的提高算法的匹配成功率。
5. 对微服务架构系统进行设计与实现，对总体架构进行设计，各功能服务进行拆分并逐个介绍各个服务的具体实现。
6. 对系统进行测试与总结分析，并投入正式生产环境进行试运行。

在实际的开发中会遇到各种各样的问题，这也是一个完整系统开发中所不可避免的。微服务架构的实现需要不断地对基础服务进行完善，业务服务的拆分与整合，寻找最合适的拆分粒度，持续改进。微服务对项目的部署与测试也有着更高的要求，需要通过向后兼容的开发方式，尽量避免迭代开发过程中所造成的版本冲突。

1. 展望

本文所实现的在线家教匹配系统已经在实际环境中投入使用，并且发挥了积极地作用，极大地提高了家教匹配的效率，减少了人力成本。但有些部分还不是非常完善，需要在之后做进一步的改进。功能方面，虽然现在系统已经实现了用户需求的大部分功能，但之后我们还可以继续添加在线支付等其他功能模块，使整个项目不断迭代开发、逐步完善。并且在这种敏捷开发的项目中，我们也可以增加测试驱动和持续集成，从而缩短项目开发周期，提前交付，并且有助于项目中开发数据的收集。现如今，人工智能和深度学习也变得越来越热门，之后我们可以将我们的推荐算法结合人工智能，进一步的提高我们算法的推荐准确性，提高系统匹配效率。微服务架构以及Docker容器等新技术保持着良好的发展势头，社区也比较繁荣，相信随着越来越多企业的应用，未来一定会越来越完善，被更多的开发人员所接受。