摘 要

【目的/意义】本文将用严格的软件工程学的方式，在实践中深入的探讨一个完整的B/S端的，知识分享社区应用的设计，搭建，构成和测试；深入讨论背后的技术和原理，框架和选型，以及成熟的项目生命周期流程和设计方案。以学术化的思考方式和衡量准则去测定工业化的软件生产的生命周期，将有助于我们更好的设计出更优异的软件。

关键词：互联网知识共享；知乎；B/S结构；Vue；Nodejs;非关系型数据库；

# ABSTRACT

[purpose/significance] in this paper, the design, construction, composition and testing of a complete B/S end, knowledge sharing community application will be deeply discussed in practice in a strict way of software engineering.In-depth discussion of the underlying technology and principles, framework and selection, as well as mature project life cycle processes and design options.Measuring the life cycle of industrial software production with academic thinking and measurement criteria will help us better design software with excellence.

Key words: Internet knowledge sharing; Zhihu; B/S structure; Vue.Nodejs; Non-relational database;

目 录

[第一章 绪论 1](#_Toc33719118)

[1.1研究背景与意义 1](#_Toc33719119)

[1.2国内外研究现状 1](#_Toc33719120)

[1.2.1国内研究现状 1](#_Toc33719121)

[1.2.2国外研究现状 1](#_Toc33719122)

[1,3本文主要工作与内容安排 1](#_Toc33719123)

[第二章 vue和nodejs的相关技术 1](#_Toc33719124)

[2.1 MongoDB相关技术 1](#_Toc33719125)

[2.1.1 关系型与非关系型数据库 1](#_Toc33719126)

[2.1.2 为什么要使用MongoDB 1](#_Toc33719127)

[2.2 B/S结构 1](#_Toc33719128)

[2.2.1 B/S与C/S 1](#_Toc33719129)

[2.2.2 为何选择B/S架构 1](#_Toc33719130)

[2.3 统一建模语言UML 1](#_Toc33719131)

[2.3.1 什么是UML 1](#_Toc33719132)

[2.3.2 UML工具processon 2](#_Toc33719133)

[2.4 express相关技术 2](#_Toc33719134)

[2.4.1 什么是express 2](#_Toc33719135)

[2.4.2 为何要使用express 2](#_Toc33719136)

[2.4.3 session技术和热更新技术 2](#_Toc33719136)

[2.5 前端框架相关技术 2](#_Toc33719137)

[2.5.1 什么是渐进式JavaScript 框架 2](#_Toc33719138)

[2.5.2 为什么要使用渐进式JavaScript 框架 2](#_Toc33719139)

[2.5.3 为什么不使用react或者agular 2](#_Toc33719140)

[2.5.4 富文本编辑器——wangEditor 2](#_Toc33719140)

[2.6 UI框架相关技术 2](#_Toc33719141)

[2.6.1 什么是UI框架 2](#_Toc33719142)

[2.6.2 什么是element UI 2](#_Toc33719143)

[2.6.3 为什么选择element UI 2](#_Toc33719144)

[2.7 小结 2](#_Toc33719145)

[第三章 知识分享社区论坛的分析 2](#_Toc33719146)

[3.1 系统需求分析 2](#_Toc33719147)

[3.1.1 目标用户 3](#_Toc33719148)

[3.1.2 目标用户行为和需求分析 3](#_Toc33719149)

[3.2 系统功能分析 3](#_Toc33719151)

[3.3 主要数据流图 3](#_Toc33719152)

[3.4 系统整体业务流程分析 3](#_Toc33719153)

[3.5 小结 3](#_Toc33719154)

[4.2 所需工具和依赖版本 3](#_Toc33719157)

[4.3 数据库设计 3](#_Toc33719158)

[4.4 路由设计 3](#_Toc33719159)

[4.5小结 3](#_Toc33719160)

[第四章 知识分享社区论坛的的设计 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719155)

[4.1 结构设计 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719156)

[4.2 所需工具和依赖版本 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719157)

[4.3 数据库设计 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719158)

[4.4 路由设计 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719159)

[4.5小结 3](file:///C:\Users\admin\Downloads\毕业论文.docx#_Toc33719160)

[第五章 知识分享社区论坛的实现 3](#_Toc33719161)

[5.1 登陆、注册模块的实现 3](#_Toc33719162)

[5.2 用户信息模块的实现 4](#_Toc33719163)

[5.3 提问模块的实现 4](#_Toc33719164)

[5.4 回答模块的实现 4](#_Toc33719165)

[5.5 评论模块的实现 4](#_Toc33719166)

[5.6 点赞、关注、收藏模块的实现 4](#_Toc33719167)

[5.7 系统推荐的实现（略） 4](#_Toc33719168)

[5.8 富文本编辑功能的实现 4](#_Toc33719171)

[5.9 小结 4](#_Toc33719172)

[第六章 知识分享社区论坛的测试 4](#_Toc33719173)

[6.1 软件测试的目的和原则 4](#_Toc33719174)

[6.2 测试内容 4](#_Toc33719175)

[6.3 测试用例的编写 4](#_Toc33719176)

[6.4 系统运行测试结果 4](#_Toc33719177)

[6.5 小结 5](#_Toc33719178)

[总 结 5](#_Toc33719179)

[致 谢 5](#_Toc33719180)

[参考文献 5](#_Toc33719181)

# 绪 论

## 研究背景与意义

当今，被互联网高速冲刷的全球网络环境，已经比过去的十年变的更加更加的成熟。互联网从最开始的服务于必要的信息交流，现在仍然在传播信息这一基础载体上，开出了各式各样的鲜花。从基础信息通信到格式各样的内容载体的信息推荐，有娱乐，新闻，教育，知识，等等；有信息找人、人找信息、人找人和推荐信息[[1]](#footnote-1)，等等。各种各样的内容与形式，构成了现在错综复杂的信息网络，其中，知识分享社区，是不可或缺的一个新媒体信息载体形式。

研究新型的知识分享社区媒体，可以有助于我们了解知识在互联网中如何有效真实的流动到它所需要的人那里去，可以有助于科普和基本的学科知识流入到惠普大众之中去。本文作者能力有限，但术业有专攻，谨以软件工程这一学科的角度，来探讨一个简单的以知乎为原型的知识分享社区是如何科学的合理的实现。在谈系统的社会效应，信息学效应时，其前提还是如何合理的实现。

## 国内外研究现状

### 国内研究现状

国内主要的知识分享社区以知乎为主。自从知乎兴起“知识付费”的概念之后，各大主流媒体平台纷纷效仿，微博，豆瓣，简书，等等以内容为主的网站，都交出了大同小异的答卷。也有不少人去研究相关话题，比如《消解与重构：知识型自媒体思想政治教育功能的回归[[2]](#footnote-2)》、《基于网络问答社区的老年科技公众关注热点及情感分析》[[3]](#footnote-3)等等，都是纵向深度非常深，且专业化强的新研究。

### 1.2.2国外研究现状

国外的类似知乎的网站最主要的就是Quora。Quora是一个问答SNS网站，由Facebook前雇员查理·切沃（Charlie Cheever）和亚当·安捷罗（Adam D' Angelo）于2009年6月创办。在Quora上你既可以订阅主题，也可以订阅具体问题的答案或是某些人的活动。当有人关注你时，当你关注的主题中有新的问题出现，或当有新的人回答你关注的问题时，你就会收到提醒。Quora系统是真的很聪明。首先，它有类似DIGG那样的投票机制，你可以根据某个回答的质量选择“顶”或“踩”的投票动作。如果你的目标是在一个答案堆栈中靠近顶部（这样所有关注这个主题的人就都可以读到的回答），那么你就需要给出高质量的答案。你还要赶早作答，因为当一个问题已经出现了一阵子后，重要人士不太可能回来重新阅读它（所以如果你太晚作答，他们就看不到你的回答，也不会投票给你的答案了）。[[4]](#footnote-4)

由此可见国外的知识分享社区的思路跟国内也是大同小异。

## 1.3本文主要工作与内容安排

本文主要围绕着如何从头搭建一个B/S端的知识分享社区为线索，在着手实现时，对齐实现技术背后的原理以及应用，框架的搭建思路和优势，做出相应的分析阐述。同时将会把现有成果与市面上已有的同类产品对齐，寻找自身的不足，并在力所能及的范围内加以修正。从软件工程的角度出发，对系统的需求分析，系统的框架采用、搭建、架构设计，前端界面的设计，后端接口的设计，数据库的设计，系统运行和测试的方方面面，进行详细到位的剖析阐述，本文设计实现的社区管理系统，对于互联网行业下未来的该类社区的发展，有着相应的意义，和美好的愿景。

# 第二章 vue和nodejs的相关技术

## 2.1 MongoDB相关技术

### 2.1.1关系型与非关系型数据库

我们常见的数据库分为SQL和NoSQL，NoSQL可以理解为Not a Sql，也可以理解为Not Only SQL，不仅仅是一个SQL数据库。在现代的服务器架构上，每天都会产生海量的信息数据，这些数据有很大一部分是由关系数据库管理系统（RDBMS）来处理。1970年 E.F.Codd's提出的关系模型的论文 "A relational model of data for large shared data banks"，这使得数据建模和应用程序编程更加简单。

通过应用实践证明，关系模型是非常适合于客户服务器编程，远远超出预期的利益，今天它是结构化数据存储在网络和商务应用的主导技术。

NoSQL 是一项全新的数据库革命性运动，早期就有人提出，发展至2009年趋势越发高涨。NoSQL的拥护者们提倡运用非关系型的数据存储，相对于铺天盖地的关系型数据库运用，这一概念无疑是一种全新的思维的注入。[[5]](#footnote-5)

### 2.1.2为什么要使用MongoDB

MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。[[6]](#footnote-6)

个人而言，使用MongoDB的理由非常的多。它拥有着丰富的社区资源和诸多的SDK支持，在以nodejs端为主要平台的各大厂商的“中后台”之中，MongoDB渐渐的成为了主流。其特性和性能也成为了当前处理庞大数据的重要角色。在做知识分享社区这样的一个系统时，庞大的用户信息，问答信息等，将会成为传统SQL数据库处理性能的一个瓶颈。今天我们可以通过第三方平台（如：Google,Facebook等）可以很容易的访问和抓取数据。用户的个人信息，社交网络，地理位置，用户生成的数据和用户操作日志已经成倍的增加。我们如果要对这些用户数据进行挖掘，那SQL数据库已经不适合这些应用了, NoSQL 数据库的发展却能很好的处理这些大的数据。[[7]](#footnote-7)

## 2.2 B/S结构

### 2.2.1 B/S与C/S

B/S的由来，是英文的Browser和Server的缩写，意为前端（客户端）为浏览器，而C/S与其的差别就是其客户端为Client，即定制开发的前端系统。Browser具有跨平台的特性，无论是在移动端，电脑端，不同的操作系统上，都有Browser的身影。而Client作为定制的前台，在不同的操作系统上，可能需要定制化的重新开发，在付出额外的开发代价时，客户端往往能拥有比浏览器更多的系统权限和功能，给与用户更好的体验。

### 2.2.2为何选择B/S架构

以知乎为例，知乎最开始兴起时，是没有Client端的。最开始的是以网页的形式展现在大家面前。同理，从它的老前辈天涯社区，到后面同样大获成功的微博，一开始的载体首先就是浏览器。本次的社区开发，也准备遵循同样的发展规律，优先选择Browser端进行开发。另外，在日新月异的今天，Web前端的技术早已更新换代，其表现能力在大部分场景下早已可以和Client相提并论。

## 2.3 统一建模语言UML

### 2.3.1 什么是UML

统一建模语言(Unified Modeling Language，UML)是一种为面向对象系统的产品进行说明、可视化和编制文档的一种标准语言，是非专利的第三代建模和规约语言。UML使用面向对象设计的的建模工具，但独立于任何具体程序设计语言。[[8]](#footnote-8)

### 2.3.2 UML工具processon

Processon是一款在线的浏览器端的UML工具，对中文用户友好，可视化拖拽界面，效率非常的高。之后本文许多图表的处理都会采用processon。

## 2.4 express相关技术

### 2.4.1 什么是express

基于 Node.js 平台，快速、开放、极简的 Web 开发框架。[[9]](#footnote-9)

Express是当下nodejs端最主流的后端框架之一。它可以在保证其架构灵活性的前提下，让项目体积也变得非常轻巧，不会因为冗余的配置项而臃肿。无论是在构建大型还是小型Web应用上，express都有他的一席之地。之前笔者曾考虑过使用其孪生兄弟—Koa.js，但是考虑到其与MongoDB的兼容性，暂时放弃了。

### 2.4.2 为何要使用express

除了以上提到的体积大小-灵活性的比值、数据库的兼容等，express还拥有丰富的可插拔的中间件，可以灵活选择各种HTTP实用工具以及中间件，根据需求快速定制出最符合需要的后端载体。在性能方面，express 提供精简又基本的 Web 应用程序功能，来保证nodejs上的原生功能仍然可以顺利的使用。

这里提前说明一下，虽然依靠express原有的HTTP功能就可以搭建出一个完整的后台，但是对于日常开发，效率作为不可忽略的重要指标时，还是需要依靠第三方插件，来简化HTTP的交互请求，从而节约时间并提升稳定性。

### 2.4.3 session技术和热更新技术

说到后端，除了我们最关注的数据库和路由，session技术也是非常非常重要的一环。通过session，我们可以放心的将常用数据存在node端中，不必再冗余的去请求数据库。同事也杜绝了客户端伪造信息的安全隐患。

另外，在express开发中，没有一个热更新的中间件是非常非常头疼的。这意味着你每一次改动，都需要去重启服务，非常的费时费力。幸好我们有nodemon这样的一个基于node的中间件，使得我们在改动文件时，就可以自动热更新整个node项目。

## 2.5 前端框架相关技术

### 2.5.1 什么是渐进式JavaScript 框架

渐进式在英文中的意思是Progressive，其意思是循序渐进，一步一步，不是说你做一个前端，你就得把所有前端东西全部用上，而是需要什么，用什么。

### 2.5.2 为什么要使用渐进式JavaScript 框架

使用渐进式框架的好处就是，不会使得前端框架变得过于冗余庞大，追加了太多不必要的功能去拖累前端的性能。这一点在前端上尤为重要。

浏览器端是作用于客户机上，有别于服务端。现代的服务器为了满足业务需求，性能在设计时就会留有余地。而在千变万化的用户场景中，Browser的性能完完全全取决于客户机的性能。虽然身处21世纪，可是无论是在一些政府单位中，还是在一些普通家庭中，用于连接互联网的设备可能已经与社会平均性能落后了几年甚至十年。考虑到这些因素，为了使几年前的电脑（一般一块新的设备上市，也得等2-3年才能在整个市场占有一定的持有率）也能流畅的浏览我们的页面，一个注重性能的前端框架是必不可少的。

### 2.5.3 为什么不使用react或者agular

对于构建本文所计划构建的项目，vue无疑是最好的选择。

本文所构建的项目因为资源的限制，注定将会是一个小型的项目，而对于快速构建小型项目，vue可以做到最少的配置，最好的优化。

react等框架在构建大型应用上无疑有更好的表现。更多的可配置项，就等于更深度的优化。vue属于不需要手动的去做过多的优化，在小型项目上的表现就已经不错，综合实际情况，本文采用的前端框架将会是vue。

### 2.5.4 富文本编辑器——wangEditor

对于本系统而言，最最重要的功能、内容、载体，都是文字。如何使得枯燥的文字能以更加生动形象的方式展现出来，这个也是本文需要研究的重点。

在技术选型时，笔者注意到了这个目前比较小众的富文本编辑器: wangEditor。用纯JavaScripts编写而成，内置css因而不需要引入。轻量、简洁、易用、开源且免费[[10]](#footnote-10)。虽然官网上给的例子，是在传统html页面上，通过js文件或者CDN引入的方式来进行使用，但是笔者这里将会尝试在vue的大体框架下去运行这一款插件。

## 2.6 UI框架相关技术

### 2.6.1 什么是UI框架

很多人可能会把前端框架和UI框架混为一谈。其实这是一个包含的关系，UI框架属于前端框架，是其一个子集。前端作为一个新兴领域，作为数据绑定和模板解析等功能的载体，产生了许许多多的框架，这些专注于数据的框架称之为前端框架，而关注UI的，基于前端框架之上的，便是UI框架。

### 2.6.2 什么是element UI

element UI是一款网站快速原型的，基于Vue2.0的桌面端组件库。其提供丰富的组件，可以大大简化前端的工作，用很短的时间就可以快速生成软件原型。

### 2.6.3 为什么选择element UI

在vue的技术栈领域，经常将两者绑定在一起，是一个主流的做法。Element+Vue的组合可以说是业界比较常见的组合，类似于Ant-design与React结合。是UI框架与前端框架融合比较成功的例子，其稳定性和可配置型都非常优异。

## 2.7 小结

在本章的最后，针对与以上的技术选项。除了这些框架本身的优秀以外，更多的还是说这些框架自身的优点适合于本文将要构建的项目。另外，笔者本身也对这些技术比较熟悉或者青睐。笔者在公司实习时，曾被人这样教导过：

“不要一味的去追求新的技术。在做产品时，往往采用的，都是被反复使用并验证的技术。一项被很多人使用的技术，恰恰证明了其自身的品质和稳定。”

所以其实在选型的时候，其实也考虑过使用React+Ant-design作为前端框架，后台使用Koa框架（Koa的特性主要是其“洋葱模型”，其路由配置可以按照洋葱模型进行配置，对于消息处理和其他一些全局处理，可以通过它的“路由中间件”轻松的搞定），数据库采用redis。可是这些技术要么就是暂未获得相应的框架支持（比如Koa与redis的相性可能不是那么的好），要么就是自身就不符合需求（redis作为企业级的轻量级非关系型数据库，其实本身做一做日志缓存或者token存储之类的还行，或者说需要应用分布式的场景也合适。但是真正要作为大型数据库去存储数据，这种纯非关系型数据库就显得有些麻烦，比如其表分开是通过命名约定俗成去进行，在设计数据库时设计名字就要设计许久。MongoDB拥有文档去规范这一行为，使得我们在能享用非关系数据库的便利时，也能像关系型数据库一样去拥有一定的约束）。

# 第三章 知识分享社区论坛的分析

## 3.1 系统需求分析

### 3.1.1 目标用户

本系统的目标用户主要为35岁以下的年轻人。且80%以上为本科以上的学历。较之其他平台，例如微博来讲，年龄定位比较年轻，学历定位较其他平台有明显差距。

### 3.1.2 目标用户行为和需求分析

对于35岁以下的，本科以上学历的人才来讲，处在一个人生事业的起跑线和关键点上。往往这群人的求知欲，对真理的探寻欲望比较高。相较于百度知道这种问牛答马数不胜数的名存实亡平台，这群用户所希望的是更高水平和更专业的提问和回答。

同样，在这群年轻的用户上，也有着强烈的自我认同的需求。需要一个平台，能向别人展示自己的专业知识，能合理的表达自己的见解与不同看法。通过平台的点赞和收藏系统，能让用户得到被认同的愉悦感，也会给用户学习到了知识的快乐。问答的主题将不仅限于专业知识，热门社会事件、影视书刊、电竞体育等等，都是可以作为交谈的主题。

目标用户将会以分享知识，分享自我观点为主，在其他用户之中，通过平台的社交功能，去挖掘与自己观点相似的人，去拓展自己的人际网，获得社会认同感。

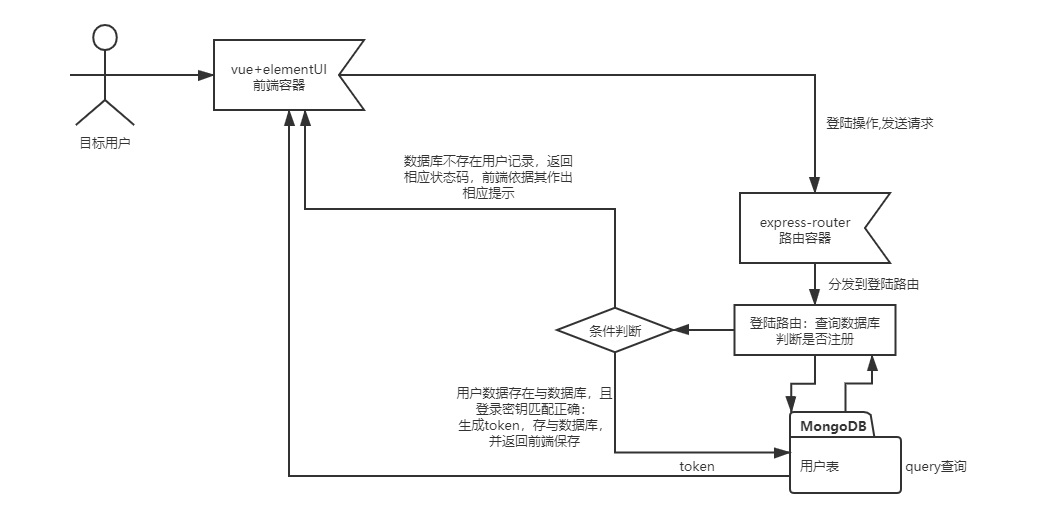
另一类目标用户，将会以学习为主，关注他们所认为的大牛或者达人，从这些能人上学习到自己所欠缺的东西，或者寻找知己。在不断完善自己的过程中去拓展自己的未来。主要以评论的方式，在别人的看法下，表达出自己或许一样，或许又不一样的看法，让这一类用户同样的获得社会认同感。

## 3.2系统功能分析

综上，本系统需要有以下的功能支撑，才成对应以上用户需求：

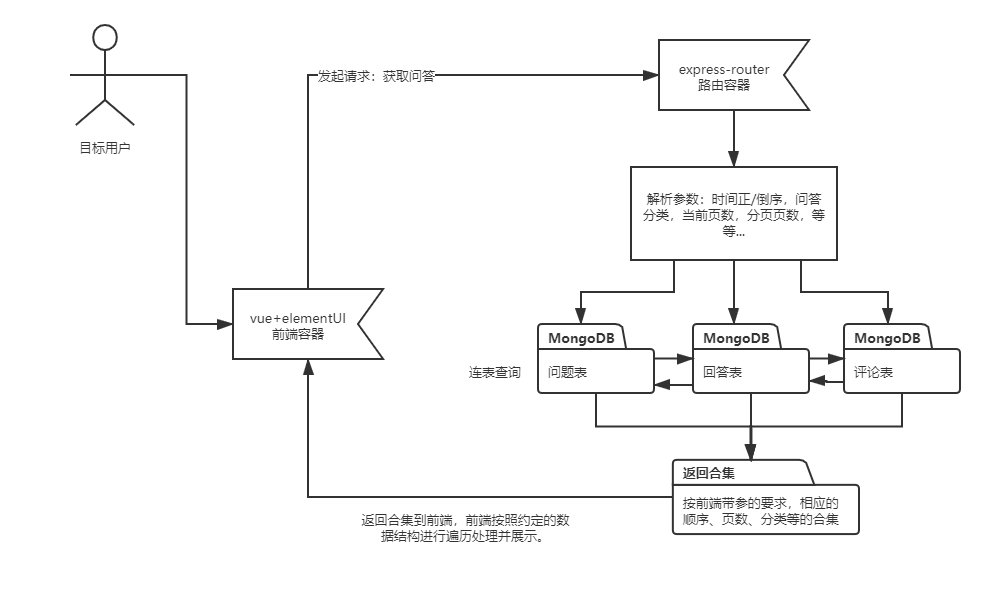
1. 注册、登陆系统，对应的是用户表，存放用户相应信息。
2. 问答系统。包括问题表，回答表，以及回复表。存放的将是相应的问答信息，且需要有相应的键作为指针去对应这些数据之间的联系。
3. 推荐系统。推荐系统分为两类。首先要做问题的分类，拥有分类问答索引，才能作为指向去驱动机器学习对应的方法，作出相关主题的推荐系统。本文由于篇幅限制，机器学习这块将不做涉及。
4. 富文本交互。问答系统主要以文字和图片的显示进行交互展示。但并不是所有的回答都会有答主精心设置的图片去装饰。大部分回答可能只是千篇一律，密密麻麻而又单调的文字。考验一个问答系统能否有人去耐心的看，不会一开始就劝退，就需要富文本来加以装饰。

## 3.3 主要数据流图



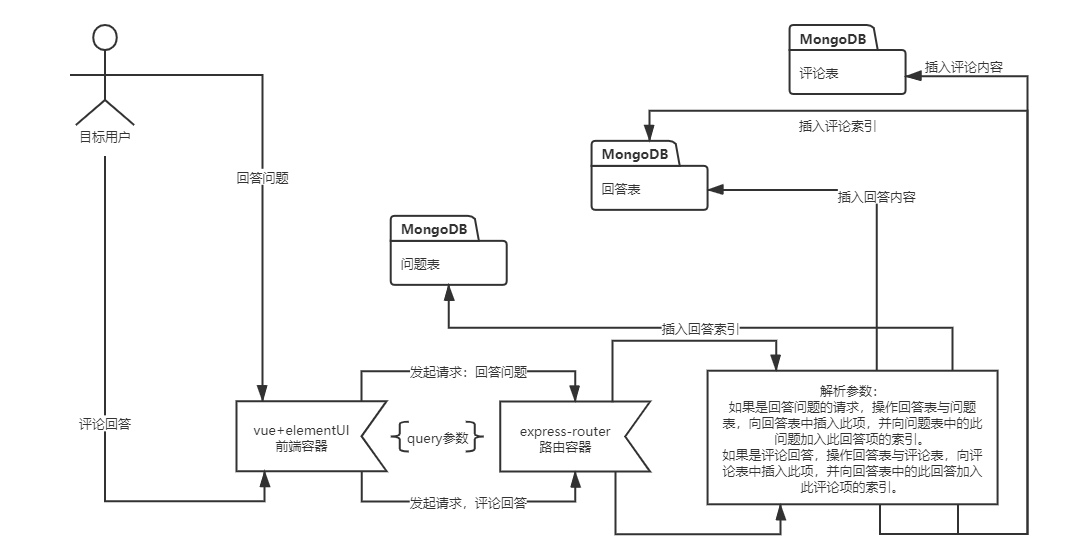
登陆流程图

登陆流程图如上图所示，笔者在公司实习时，系统里无一例外都是采用了token限制。本系统由于篇幅限制，没有加入token。但流程图上仍将其清晰的展示出来。



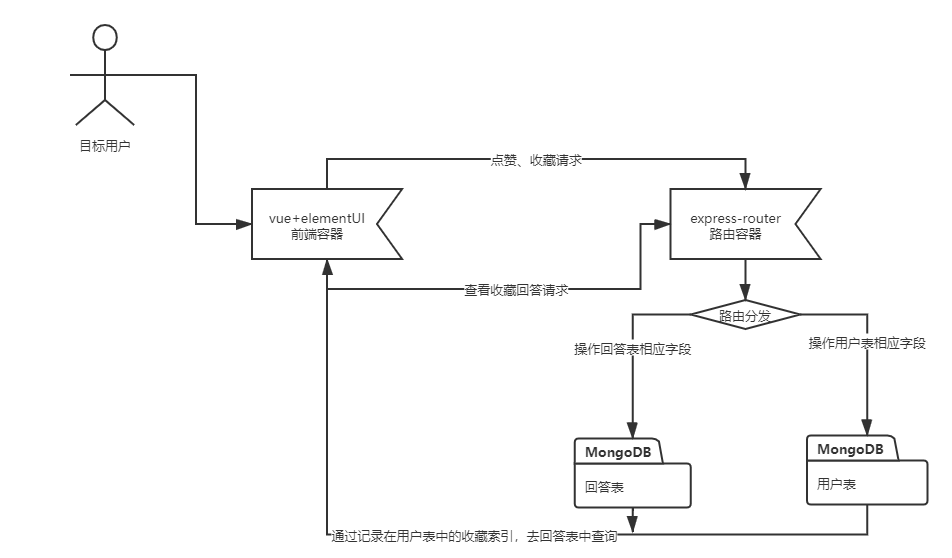
获取问答概览

这里之所以说是概括，是因为这张图融合了查询操作，统一划分为了“连表查询”。MongoDB作为非关系型数据库，是没有连表这一说的。另外，这里获取数据也不是一次性获取到。数据除了分页的约束区间外，还分层级。比如，在首页，看到的仅仅是问题表，要点进去某一个问题，才会依据问题表的依赖去查询回答表，以此类推。详细设计实现流程将会放在后续的实现章节里。



评论与回答流程图

评论与回答流程图如上图所示。可以看出三张表其实存在层级关系。在逻辑上次该层级关系能映射到问答社区里的实体。当然这里的流程并不完整，其实这些表与用户表还有关联，此处只是概览。



点赞、收藏流程图

点赞与收藏流程图如上图所示。其实可以看出，问答表以及评论表都和用户表是静静相连的。另外需要考虑的是将收藏表储存与用户表上，当用户收藏的索引很多时，根据此索引去查询相应回答，在大型数据环境下的效率问题，这是考验数据库选项和配置的一个比较关键的地方。

## 3.4 系统整体业务流程分析

以上主要流程图当时不是整个系统的全部流程。而且仅仅只是在前后端-数据库这一宏观层面去描述如何做这样一个事情。具体的实现会复杂一些流程图里完全没有去关注前端是如何在前端管理相应数据。甚至在之前章节中介绍Vue技术时，也没有说到Vuex这项技术。所以本文在整体架构规划上，几乎没有对前端的数据流动做分析，这其实也是因为前后端自身的特性所导致。UML图一般都是用于描述后端调度和服务逻辑，前端真要讲，就会显得啰嗦而繁琐。所以更细致且进一步的设计和实现，将会留到之后的两章。

本章主要就几个主要场景宏观的数据流动和逻辑控制做了一个大体的设计，按照笔者在实习时学到的，前后端分离的模式进行数据通信和规划设计。通信的载体为每次请求的query带参，具体类型视请求情况去决定。但几乎表单类型的提交，例如登录，都会设置成POST请求；浏览信息的请求几乎都是GET；设计到各个数据实例的更新和删除（问答实体，用户实体等等）就使用PUT和DELETE等。这些设计属于路由设计的部分，将会在下一章节详细描述。

## 3.5 小结

本章的需求分析主要参考了Micheal\_Li的《知乎产品分析报告[[11]](#footnote-11)》。其余部分的设计，均为宏观概要设计。但这也是软件工程生命周期不可缺少的部分。往往在这种宏观的交互设计中，就能发现系统可能潜在的巨大问题和具体矛盾。比如笔者在进行此部分设计时，就会去思考到三层问答评论表的连接关系，考虑到为了优化查询速度，不同字段的数据的存储类型应该设置成什么样子，以及每一个字段应该跟哪一个表放在一起，才会减少跨表查询的次数，等等。系统的致命缺陷和重大问题，往往是在这个部分提出了，以至于现有技术框架如果不能实现或者满足需求，可能会重新回到技术选型的那一步，从头计议。

# 第四章 知识分享社区论坛的的设计

## 4.1 工程文件结构设计

搭建这样的一个工程，前端后端分离的情况下，无疑是分散成两个工程。我们先从后端工程开始分析：

.

│ .gitignore

│ app.js

│ package-lock.json

│ package.json

│

├─bin

│ www

│

├─db

│ index.js

│ user.js

│

├─node\_modules（此处不展开）

│

├─public

│ ├─images

│ ├─javascripts

│ └─stylesheets

│ style.css

│

├─routes

│ index.js

│ users.js

│

└─views

error.ejs

error.jade

index.ejs

index.jade

layout.jade

此时的项目结构是我刚开始初始化后端项目的结构。使用了git-init、npm-init等方法。之后使用了express-e进行初始化。之后安装了mongooes插件。

值得注意的是，此处为了有序的安放mongodb的Schema，我在db文件夹下将其进行了拆分。

db文件夹的入口文件安装nodejs的规范成为index，我在index里将会一个个的引入相应的Schema文件，目前只有一个user，但后续将会有很多。

const user = require('./user') //引入

module.exports = {

user: mongoose.model('User', new mongoose.Schema(user))

};

向外暴露接口时，就可通过db.user的方式，分门别类的获取到要操作的表，进行相应操作。使用这种文件的划分方式，可以有效的保证数据库表之间的分离，使得在后期Schema过多时不会混乱。

在路由控制上，使用了express-session中间件。具体的好处，前文选型环节有讲过，这里不再赘述。

另外后端使用了一个express的热更新工具，nodemon，非常重要。可以大大提升开发效率。这里也要做一点小配置，express脚手架搭建项目时，将scrip下的start命令改为nodemon ./bin/www，这样就可以实现热更新了。

接下来是前端工程：

.

│ .gitignore

│ babel.config.js

│ package-lock.json

│ package.json

│ README.md

│

├─public

│ favicon.ico

│ index.html

│ login.html

│

├─api

│ axios.js

│

└─src

│ App.vue

│ main.js

│ routes.js

│ store.js

│

├─assets

│ └─img

│ ad\_1.jpeg

│ ad\_2.jpeg

│ avatar.jpg

│ close.svg

│ loginBG.png

│ loginLogo.png

│ logo.png

│

└─components

│ index.vue

│

├─login

│ login.vue

│

├─main

│ │ main.vue

│ │

│ ├─content

│ │ │ content.vue

│ │ │

│ │ ├─attention

│ │ │ attention.vue

│ │ │

│ │ ├─hotsearch

│ │ │ hotsearch.vue

│ │ │

│ │ └─recommend

│ │ recommend.vue

│ │

│ └─sticky

│ card.vue

│ creator.vue

│ globalwrite.vue

│ sidebar.vue

│ sticky.vue

│

└─nav

nav.vue

personal.vue

search.vue

同样，这也是前端初期的工程文件结构。这里看似文件夹很多，是因为把前端工程划分的非常非常细，在编写过程中，一个nav导航栏拆分成了3个，主要页面左边拆出一个主要content区域，用于展示问答，右侧也拆成了5份，主要做复用。

以上做法，只是遵循vue组件越细分，越复用，效果越好的思想。在前端项目构建初期，最主要使用的构建工具还是vue-cli。

在页面布局上，不太寻常的做法是，本项目的vue-router将login页面也划分了进去。一般来说因为登陆页与主页的结构不同，这两个页面应该是独立存在的，vue-router的主要作用范围应该是主页的content区域。但是因为知识分享社区不是类似于管理系统的布局规划，其content区域相对复用较少，所以就采用了另一种方法：在每个子页面重新引入nav组件（几乎是唯一的公共区域）。本身由于前端工程的划分细粒度较高，所以这样做也是没有问题的。

网络通信方面，前端使用了axios来发送请求，axios之前在主要技术选型上没有讲，但是并不代表这个技术不重要。axios可以大大的简化前端请求的步骤，提升速度，给予更好的请求头封装和提高安全性。

## 4.2 所需工具和依赖版本

代码工具：

Git: 2.25.1（尽量使用最新版）

Npm: 6.13.4（尽量使用最新版）

Nodejs: 12.14.1（尽量使用最新版）

前端：

axios:0.19.2(及以上)

core-js:2.6.5(及以上)

element-ui:2.12.0(及以上)

vue:2.6.10(及以上)

vue-cookies:1.7.0(及以上)

vue-router:3.1.3(及以上)

vuex:3.1.1(及以上)

wangeditor:3.1.1(及以上)

后端：

body-parser:1.19.0(及以上)

cookie-parser:1.4.4(及以上)

debug:2.6.9(及以上)

express:4.16.1(及以上)

http-errors:1.6.3(及以上)

jade:1.11.0(及以上)

mongoose:5.9.4(及以上)

morgan:1.9.1(及以上)

express-session:1.17.0(及以上)

nodemon:2.0.2(及以上)

浏览器：Chrome: 版本 80.0.3987.132（正式版本） （64 位）

数据库：Mongodb: 3.0.6(及以上)

集成开发环境：WebStorm 2019.3.1 Build #WS-193.5662.54, built on December 18, 2019 Licensed to Wenxing Ding

## 4.3数据库设计

用户表(user):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | \_id | Account | Password | Name | Gender | Intro | Area | Industry | Education | Collect | Foucs | stars | visit | creatTime |
| 类型 | ObjectId | String | String | String | Number | String | String | String | String | Array | Array | Number | Number | Date |
| 必要 | 自动生成 | 是 | 是 | 自动生成 | 否 | 否 | 否 | 否 | 否 | 否 | 自动生成 | 自动生成 | 自动生成 | 自动生成 |

用户表初期的主要设计如上，\_id是mongodb自动去生成，而createTime是在执行时，笔者调用了new Date()方法生成的。Name则是默认的‘知乎用户’加上注册时的账户

在用户注册时，不会要求填写除了必要的账户和密码以外的信息，这些信息可以进一步的到nav栏中的设置里去填写。

问题表(question):

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | \_ id | Classify | Title | Content | answerList | Creator | createTime | answerCount | startsCount | commentCount | visitCount |
| 类型 | ObjectId | String | String | String | Array | ObjectId | Date | Number | Number | Number | Number |
| 必要 | 自动生成 | 是 | 是 | 是 | 自动生成 | 自动生成 | 自动生成 | 自动生成 | 自动生成 | 自动生成 | 自动生成 |

问题表初期的主要设计如上，通过对比我们可以看出，作为一个系统的内部表，它不像用户表一样，有许多的属性需要用户去手动填写。许多关键，重要的信息，都是系统自动去生成的。有些是像createTime一样，写在Schema内，自动生成。另外一些，是在后端开发的过程，编写相应的逻辑，去执行相应的代码来管控生成的。

回答表(question)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | \_ id | targetQuestion | content | creator | creatorName | creatorJob | commentList | commentCount | startsCount | creatTime |
| 类型 | ObjectId | ObjectId | String | ObjectId | String | String | Array | Number | Number | Date |
| 必要 | 自动生成 | 是 | 是 | 是 | 是 | 是 | 自动生成 | 自动生成 | 自动生成 | 自动生成 |

回答表的主要设计如上表所示。同样的作为系统内部表，不需要用户去填写太多的信息。在界面中，需要填写的其实只要content，即回答内容。其余所有的内容，要么是后端自动生成，要么是前端在生成请求时自动获取信息生成。表里有一个targetQuestion字段。其实这个字段是最开始是没有的。当然在系统的进一步设计中发现，要保持表与表之间的强关联性，这样的双向绑定还是必不可少的。

还有一处设计是在creatorName与creatorJob字段。为什么我们明明存了creator的id，还需要去存这两个字段呢？笔者在设计时认识到，强关联表之间一定会有频繁的相互查功能。然而，在MongoDB中，虽然他的普通搜索效率远超mysql，但是其连表查询的效率，确要低得多。因为MongoDB并不是一个完全的关系型数据库，去做这样关系型数据库拿手的查询操作，的确是不太理想。所以这里，笔者把连表时需要频繁查询的字段直接插入到目标表中去。这里的一个典型例子就是问题表需要同时展示问题是由谁提出的，以及其简略信息。这里直接的插入就可以略掉相应的连用户表查询。而且对于问题list持续增长的情况，对于每一项都需要去做这样的一个查询操作，那样效率会非常非常的低下。

总而言之，这里是典型的用空间换取时间的置换操作。

评论表(comment)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 字段 | \_ id | targetAnswer | content | creator | creatorName | replierName | creatTime | startsCount |
| 类型 | ObjectId | ObjectId | String | ObjectId | String | String | Date | Number |
| 必要 | 自动生成 | 是 | 是 | 是 | 是 | 否 | 自动生成 | 自动生成 |

最后是评论表。评论区域因为要展示和存储的信息相对较少，所以表的维度也相比其他表要少一些。另外，为了不把评论表做成递归表，这里的设计形式是设计成的一维表。即没有递归，递归层数为0。replierName字段将会决定此条评论是在评论问题，还是在回复评论。targetAnswer自不用说，也是为了保持两个表之间的强关系性。而creatorName也是作为了减少连表查询而做的存储。

## 4.4路由设计

后端部分接口路由设计:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 路由 | 类型 | 参数 | 用途 | 响应 |
| / | get |  | 检查后台状态 | 后台状态 |
| /login | post | 登陆要素 | 登陆 | 成功与否 |
| /user | get | userID | 获取用户信息 | 用户信息 |
| /user | post | 用户信息 | 更新用户信息 | 成功与否，新用户信息 |
| /user/collect | post | userID，问题id | 关注问题 | 成功与否 |
| /question | get |  | 获取问题列表 | 问题列表 |
| /question | post | 问题信息 | 添加问题 | 成功与否，新问题列表 |
| /question/startsCount | put | questionID | 点赞、踩问题 | 成功与否 |
| /question/visitCount | put | questionID | 记录浏览数 | 成功与否 |
| /answer | get | questionID | 查询回答 | 回答列表 |
| /answer | post | questionID，回答信息 | 添加回答 | 成功与否，新回答列表 |
| /answer/startsCount | put | answerID | 点赞、踩回答 | 成功与否 |
| /comment | get | answerID | 获取评论 | 评论列表 |
| /comment | post | answerID | 添加评论 | 成功与否，新评论列表 |
| /comment/startsCount | put | commentID | 点赞、踩评论 | 成功与否 |

后端路由设计比较规整，但也没有那么规整。目前的接口都还没有删除的功能。

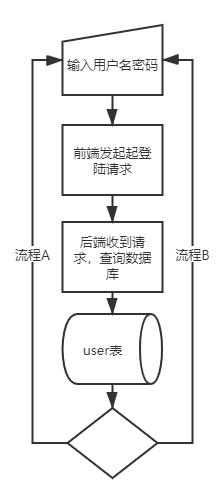
## 4.5小结

项目还没写完，删除接口还没写，先不写小结。

# 第五章 知识分享社区论坛的实现

## 5.1 登陆、注册模块的实现

因为流程比较复杂，大的流程笔者这里直接使用流程图：



这里有一处比较特别的功能是自动注册功能。

后台查询数据库时，首先查询的即是这个用户名是否存在。如果存在，那就是图中的流程A，随后将传入的用户名与密码进行匹配，匹配成功，返回登陆成功的状态，并存入session。

但是如果查询用户名时即显示不存在。就会变成自动注册状态，将此时传入的用户名密码作为注册的参数，在后头进行自动注册。这就是流程B。

前端在接受到后端返回的数据之后，会将常用的信息直接存储在本地的cookies上，以免反复请求后台，增加网络通信负重。

响应体的大致结构如下所示：

{

message: "获取成功!"

userData:

account: "testaccount"

gender: 1

intro: "本系统第一个用户。"

area: "中国 成都"

industry: "前端开发"

job: "北京捷泰天域信息技术有限公司"

education: "加州大学伯克利分校 (UC Berkeley)·生命科学"

collect: []

focus: []

stars: 0

visit: 0

creatTime: "2020-03-18T09:51:39.644Z"

\_id: "5e71ef95a7622577e420d246"

name: "知乎小透明"

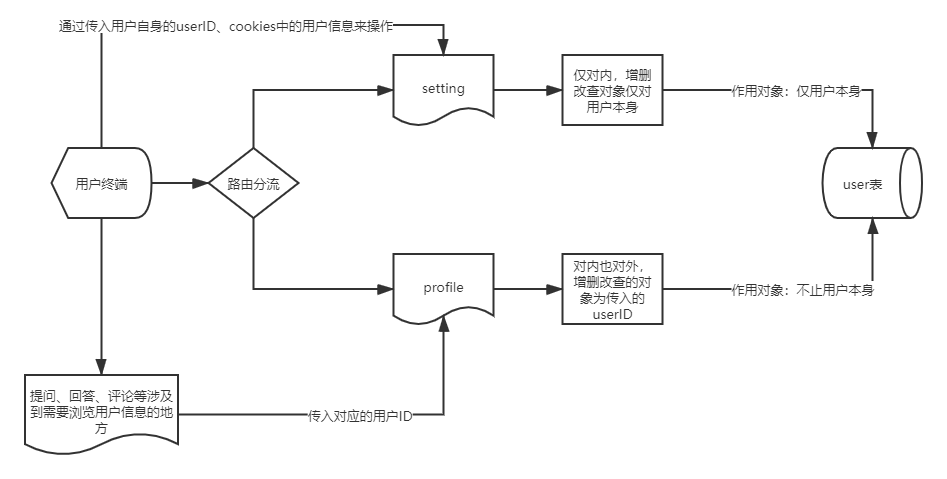
\_\_v: 0

}

最初注册的响应体并不是这个样子，而是只有拥有最基本的账户信息，名字也会被初始化成统一的“知乎用户xxx”。xxx即为账号。后续的详细信息都是登陆之后在个人详情设置页面手动填入。

## 5.2 用户信息模块的实现

用户信息模块分为两大块。一块是用于填写设置用户信息的页面，对应的前端路由为/setting。此页面属于用户页面，只有用户自身能够访问并使用。另一块是既可以对内，也可以对外的个人主页页面，路由为/profile。这个页面不仅可以预览自己的对外信息情况，也可以用来浏览系统中其他用户的相关信息。具体的情况下面先用一张图来展示：



在前端的路由控制中，进入profile页面有两种方式。第一章是通过点击nav导航栏上的下拉菜单，点击我的主页，来进行页面的跳转。

笔者在这里有一处稍显精妙的设计。在vue的template之中，下拉菜单的大致骨架是这样的 ：

<el-dropdown @command="handleCommand">

<img class="avatar" :src="avatar"/>

<el-dropdown-menu slot="dropdown">

<el-dropdown-item command="myhome">我的主页</el-dropdown-item>

<el-dropdown-item command="myset">设置</el-dropdown-item>

<el-dropdown-item command="logout">退出</el-dropdown-item>

</el-dropdown-menu>

</el-dropdown>

系统的前端使用了elementUI的ui框架库，所以这里的标签是el-dropdown的形式。dropdown标签上绑定了一个特有的command事件。一般来说事件绑定都是click事件或者其他时间，command事件是一个非常好用的分流事件。笔者在下方的methods这样写道：

handleCommand(command){

this[command]()

},

这个函数其实可以简化成这样：

handleCommand: command => this[command]()

这个其实就是JavaScripts的其中的一种理念，化繁为简，ECMAScript6标准大费周章的精心设计了许多语法糖，为的就是把各个这种函数浓缩成这一行。

但是笔者为了代码风格统一还是没有这样写。毕竟水平有限，不是每个函数的都可以设计成这样精简。

挂载在dropdown上的command钩子，会给函数传入一个参数叫command，类型是字符串，内容就是用户触发的某一项item的command属性上的字符串内容。通过这个内容，我们可以在methods里设置相应的方法，让方法名与之对应，这样就可以通过这个函数来实现函数的自动调用。如果害怕重名或者受限于命名规范，直接修改command属性值或者是在函数调用处做字符串拼接，也是可行的。

例如下面就是其中跳转profile页面的函数：

myhome(){

this.$router.push('/profile')

},

我们再来看看从其他地方进入profile页面时，调用函数的样子：

goDetail(id){

this.$router.push({

name: 'profile',

params: {

tagerId: id

}

})

},

Vue实现路由带参有许多中方式，这里笔者为了隐式的带参，采用了使用name的方式指定组件，然后再通过params带参传入。而在profile的mounted方法里，写有如下的逻辑：

if (this.$route.params.tagerId){

this.others = true

this.$axios.get('/user',

{\_id: this.$route.params.tagerId}, res => {

this.userData = res.data.userData

})

} else if (this.$cookies.get('userData')){

this.others = false

this.userData = this.$cookies.get('userData').userData

console.log(this.$cookies.get('userData'))

} else {

this.$router.push('/login')

}

这段代码分为三个理解块。第一个if，如果路由转跳profile时有传入了params参数，那么就会调用获取用户信息的接口，获取相应的用户信息，填入页面中。另外，页面里也有一些人称字段，通过other属性来控制切换是第一人称还是第三人称。第二个if，就是没有传入id参数的情况下，去检查cookies是否有保持现有的用户数据。如果没有的话证明当前系统cookies丢失，或者不在登陆状态，那么就要调回login页面进行登陆。

## 5.3 提问模块s的实现

提问模块分为展示和提问。展示就是系统的首页，因为系统的最核心功能就是围绕着问答展开的。展示部分的实现相对简答（逻辑实现简答，但是前端的工作量可不小），在打开首页时，向后台请求问题的列表。问题列表的每一项大致结构如下：

{

classify: ["["社会学","形而上学","心理学","神经科学","语言学"]"]

content: "<p>如题，语言本身不是一个很方便的东西。不仅存在在空间上的差异，在时间上也会存在差异甚至于消失。如何能通过一种方式，存在一种统一的语言表达方式，可以直面人与人的心理，不会产生信息传递的差异与丢失，让歧义消失，让心灵更近？&nbsp;&nbsp;<br></p>"

answerList:["5e7f500f5f68985f44414424","5e7f506a7cf4065444317a55", "5e7f53087cf4065444317a56",…]

answerCount: 6

startsCount: 3

commentCount: 0

visitCount: 77

\_id: "5e7c79aa9594e014ed4fdcec"

title: "如何跨越语言的瓶颈，清楚的表达自己真正想要表达的东西？"

creator: "5e71ef95a7622577e420d246"

creatTime: "2020-03-26T09:45:14.413Z"

\_\_v: 0

}

这里的数据是测试时的数据，为了更好的表示数据结构没有把里面的数据全部清除干净再展示。在comtent里的html形式的富文本,这一部分后面讲到富文本实现时会说明。

## 5.4 回答模块的实现

## 5.5 评论模块的实现

## 5.6 点赞、关注、收藏模块的实现

## 5.7 系统推荐的实现（略）

## 5.8 富文本编辑功能的实现

## 5.9 小结

1. Webxyc.互联网中信息传播方式的转变与发展[DB/OL]. https://wenku.baidu.com/view/d802f687a0116c175f0e48e7.html [↑](#footnote-ref-1)
2. 曹银忠,闫兴昌.消解与重构：知识型自媒体思想政治教育功能的回归——以知乎APP平台为例[J].电子科技大学学报(社科版),2020,22(01):95-100. [↑](#footnote-ref-2)
3. 娄岩,杨嘉林,黄鲁成,苗红.基于网络问答社区的老年科技公众关注热点及情感分析——以“知乎”为例[J/OL].情报杂志:1-8[2020-03-02].http://kns.cnki.net/kcms/detail/61.1167.G3.20200113.1723.006.html. [↑](#footnote-ref-3)
4. 百科词条.Quora [DB/OL]. https://baike.baidu.com/item/Quora/9144430?fr=aladdin#5 [↑](#footnote-ref-4)
5. RUNOOB.NoSQL 简介 [DB/OL].https://www.runoob.com/mongodb/nosql.html [↑](#footnote-ref-5)
6. RUNOOB.MongoDB 教程 [DB/OL].https://www.runoob.com/mongodb/mongodb-tutorial.html [↑](#footnote-ref-6)
7. [↑](#footnote-ref-7)
8. 百科词条.统一建模语言[DB/OL]. https://baike.baidu.com/item/%E7%BB%9F%E4%B8%80%E5%BB%BA%E6%A8%A1%E8%AF%AD%E8%A8%80/3160571?fromtitle=UML&fromid=446747&fr=aladdin [↑](#footnote-ref-8)
9. Express 中文网.express 4.17.1[DB/OL].https://www.expressjs.com.cn/ [↑](#footnote-ref-9)
10. wangEditor.wangEditor[DB/OL].http://www.wangeditor.com/index.html [↑](#footnote-ref-10)
11. Micheal\_Li.知乎产品分析报告[DB/OL].https://www.jianshu.com/p/de203b7b8f20 [↑](#footnote-ref-11)