

**程序设计实践**

**概 要 设 计 （详细设计）报 告**



**题目: BUPT HUB论坛**

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | **学号** | **学院** | **班级** |
| **刘兆宏** | **2021213579** | **人工智能学院** | **2021219113** |
| **沈尉林** | **2021213595** | **人工智能学院** | **2021219113** |
| **李小芊** | **2021213633** | **人工智能学院** | **2021219113** |

**2023年4月**

目录

[1、系统概述 3](#_Toc24366)

[1.1、系统简介 3](#_Toc1247)

[1.2、系统目标 3](#_Toc19998)

[1.3、系统运行环境 3](#_Toc23001)

[1.4、开发环境 3](#_Toc31985)

[2、总体结构设计 4](#_Toc2748)

[2.1、软件结构 4](#_Toc24149)

[2.2、设计思想 4](#_Toc1555)

[3、模块设计 5](#_Toc6892)

[3.1、用户模块 5](#_Toc7445)

[3.2、帖子模块 6](#_Toc26034)

[3.3、评论模块 8](#_Toc11008)

[3.4、关注模块 9](#_Toc6227)

[3.5、点赞模块 9](#_Toc2199)

[3.6、通知模块 10](#_Toc32577)

[3.7、管理员模块 11](#_Toc13396)

[4、数据库与数据结构设计 12](#_Toc15256)

[4.1、 数据库及数据表 12](#_Toc7489)

[4.2、 数据结构设计 13](#_Toc4373)

[4.3、 数据存储设计 15](#_Toc11680)

[5、接口设计 17](#_Toc26778)

[5.1、外部接口 17](#_Toc23495)

[5.2、内部接口 17](#_Toc21540)

[6、其他设计 18](#_Toc31445)

# 1、系统概述

**1.1、系统简介**

1. 基本情况和背景：

平台的建立基于学生社区的交流需求，使同学们拥有可以讨论课程内容、分享学习资源、解决问题的专属空间，为同学们提供一个安全、友好的学术交流环境。

本系统是一个小巧轻量、集发布和讨论于一体的论坛平台；基于北邮人论坛的使用和周围同学的反馈体验，让用户体验更加简约流畅。

2.术语介绍：

|  |  |  |
| --- | --- | --- |
| **序号** | **术语或缩略语** | **说明性定义** |
| 1 | Vue.js | 一款用于构建用户界面的 JavaScript 框架 |
| 2 | Vue Route | Vue.js 的官方路由 |
| 3 | Naïve UI , Element Plus | 基于 Vue 3，面向设计师和开发者的组件库 |
| 4 | Less | Leaner Style Sheets是一门向后兼容的 CSS 扩展语言 |
| 5 | Axios | 一个基于 promise 的网络请求库 |
| 6 | FastAPI | 一个使用 Python 3.6+ 用于构建 API 的 web 框架 |
| 7 | SQLAlchemy | 一个python语言实现的针对关系型数据库的ORM库 |
| 8 | ORM | 对象关系映射（Object Relational Mapping，简称ORM）是解决面向对象与关系数据库互不匹配的技术模式 |

**1.2、系统目标**

1. 功能目标：

论坛用户可以注册和登录，并能够发布帖子、查看用户消息、关注其他用户

论坛用户可以对论坛上的帖子进行评论、点赞和搜索

管理端可以对用户发送系统消息和禁用账户

管理端可以对论坛上的帖子和评论进行审核和删除

1. 性能目标：

精度：用户上传的文字和图片信息能够被系统正确识别、处理和输出

时间特性：用户能够快速访问和使用系统，系统能在短时间内快速响应用户的请求

灵活性：系统能够适应不同用户和不同场景下的使用需求、技术和环境的变化

故障处理：对于网络、数据库故障和安全漏洞能够及时发现和维护

**1.3、系统运行环境**

1. 硬件平台：Legion Y7000P 2019、Dell G15 5511、MacBook Air(M1)

2. 操作系统：Windows、Mac OS、Linux

3. 前端框架：Vue.js + Vue Router + Naïve UI + Element Plus + Less + Axios

4. 后端框架：FastAPI+SQLAlchemy

5. 数据库：Sqlite

6. 编程平台：VSCode

7. 网络协议：TCP/IP

**1.4、开发环境**

1. 工程工具：

集成开发环境：VSCode、PyCharm

版本控制工具：Git

自动化构建工具：Vite

调试工具：Volar、Unicorn、Chrome浏览器

截止到项目发布时的最新版

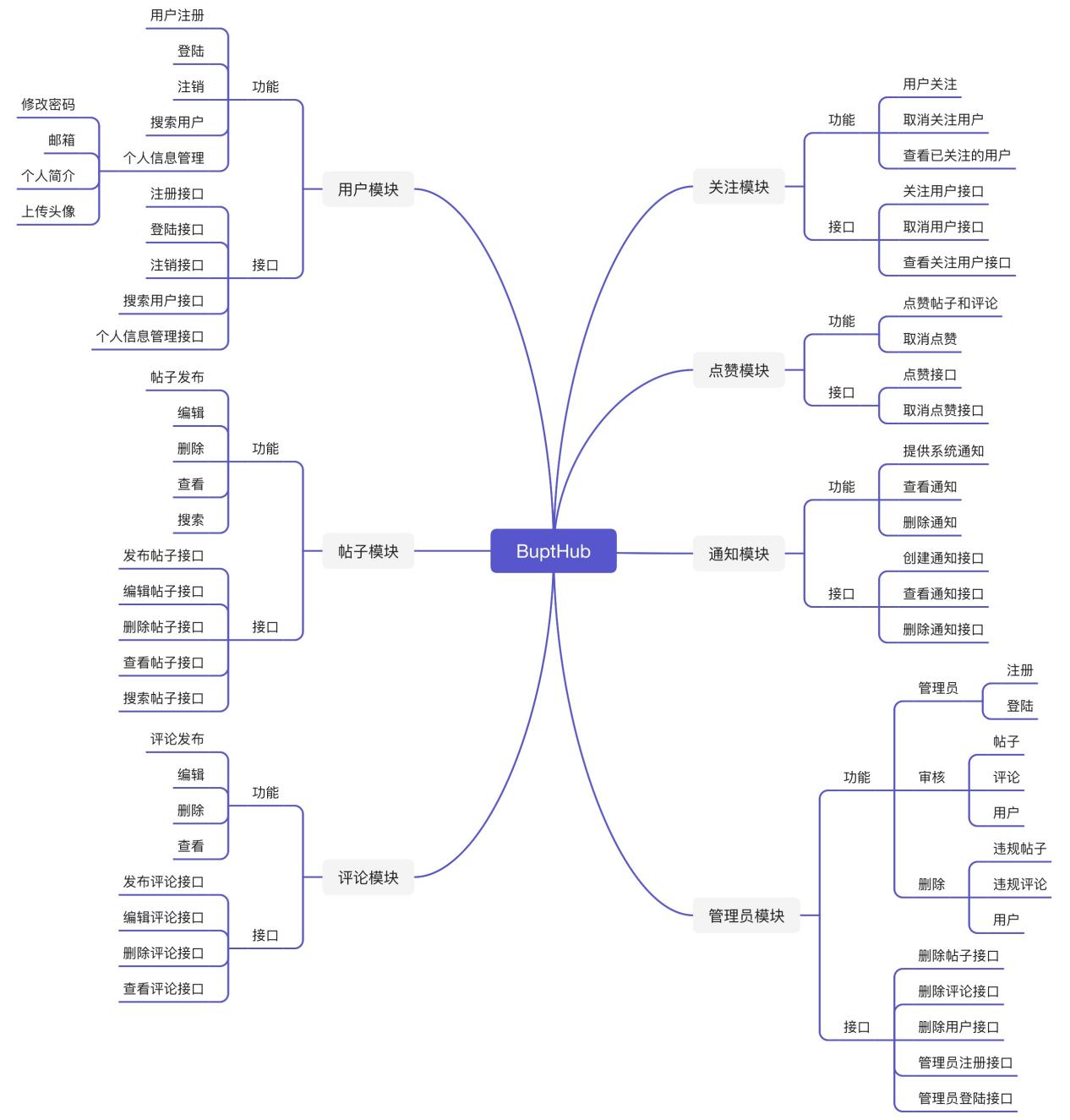
2. 开发语言：

Vue：3.2.47

Python：3.9.12

# 2、总体结构设计

**2.1、软件结构**



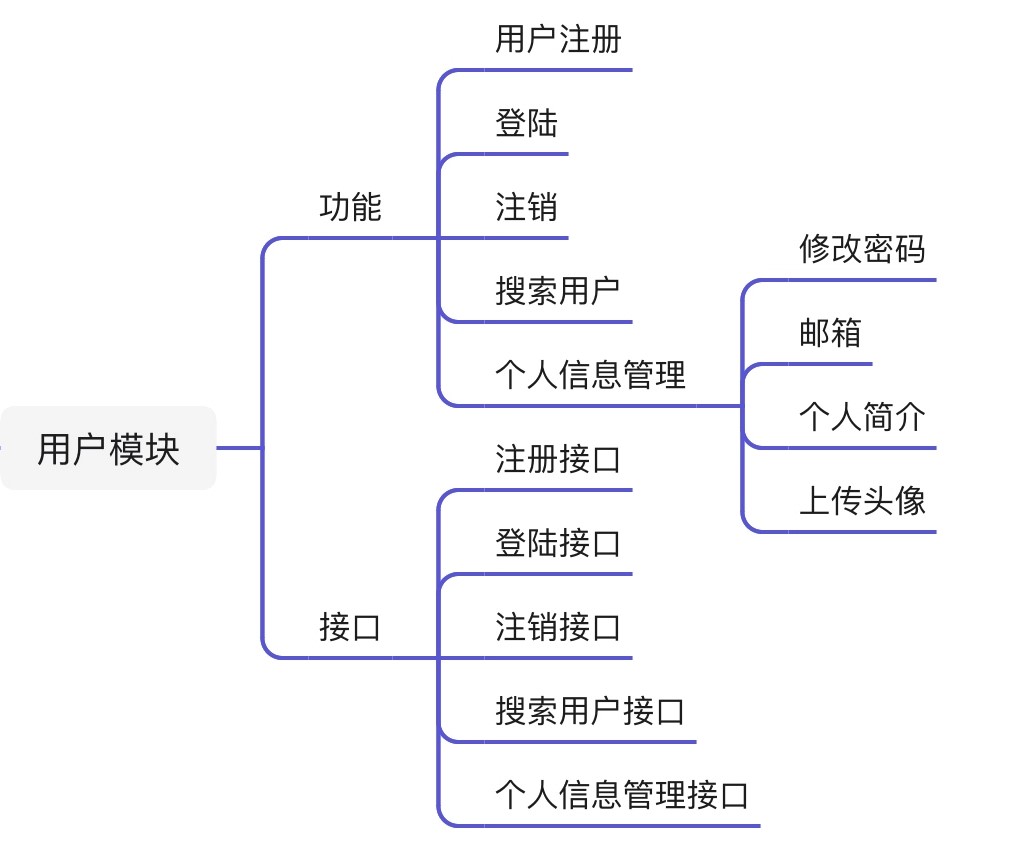
**2.2、设计思想**

本论坛项目的基本设计思想和理念是建立一个用户友好、功能丰富的论坛平台，使用户能够方便地进行帖子发布、交流和互动。

# 3、模块设计

**3.1、用户模块**

3.1.1 功能描述



3.1.2 接口描述

1. 输入数据：

- 注册接口：

- 需要用户提供注册所需的信息，包括用户名、密码、邮箱等。输入数据需要进行有效性检验，例如检查用户名是否已被使用、用户名是否符合要求、密码的复杂度要求、邮箱是否已被使用、邮箱是否存在等。

- 登录接口：

- 需要用户提供登录所需的信息，包括用户名和密码。输入数据需要进行有效性检验，检查用户名和密码是否匹配。

- 注销接口：

- 无需输入数据，直接执行注销操作。

- 个人信息管理接口：

- 根据用户需求，可能需要输入修改后的个人信息，例如新密码、新邮箱、新个人简介、新头像等。输入数据需要进行有效性检验，例如检查密码强度、邮箱格式等。

- 搜索用户接口：

- 要搜索的用户名的关键词

2.输出数据：

- 注册接口的输出数据：

- 注册成功的提示信息

- 注册失败的错误信息

- 登录接口的输出数据：

- 登录成功的身份验证信息，生成的 JWT token

- 登录失败的错误信息

- 注销接口的输出数据：

- 注销成功的提示信息

- 注销失败的错误信息

- 个人信息管理接口的输出数据：

- 修改个人信息成功的提示信息

- 修改个人信息失败的错误信息

- 搜索用户接口的输出数据：

- 根据关键词匹配的用户列表

3.1.3 数据结构描述：

用户模块涉及的局部数据结构包括用户信息、注册信息、登录信息等。

- 用户信息：包括用户的唯一标识（ID）、用户名、密码、邮箱、个人简介、头像等。

- 注册信息：包括用户注册时提供的用户名、密码、邮箱等。

- 登录信息：包括用户登录时提供的用户名和密码。

3.1.4 实现思路：

1. 注册流程：

- 接收用户提供的注册信息。

- 验证注册信息的有效性。

- 如果验证通过，创建新的用户记录，并保存到数据库中。

- 返回注册成功的提示信息。

1. 登录流程：

- 接收用户提供的登录信息。

- 验证登录信息的有效性。

- 如果验证通过，生成登录凭证（如JWT token）作为身份验证标识。

- 返回登录成功的身份验证信息。

1. 注销流程：

- 销毁当前登录的用户凭证，使其失效。

- 返回注销成功的提示信息。

1. 个人信息管理流程：

- 根据用户需求，接收修改后的个人信息。

- 验证修改后的个人信息的有效性。

- 如果验证通过，更新用户记录中对应的信息。

- 返回修改成功的提示信息。

5. 搜索用户流程：

- 接收用户提供的搜索条件，用户名关键词。

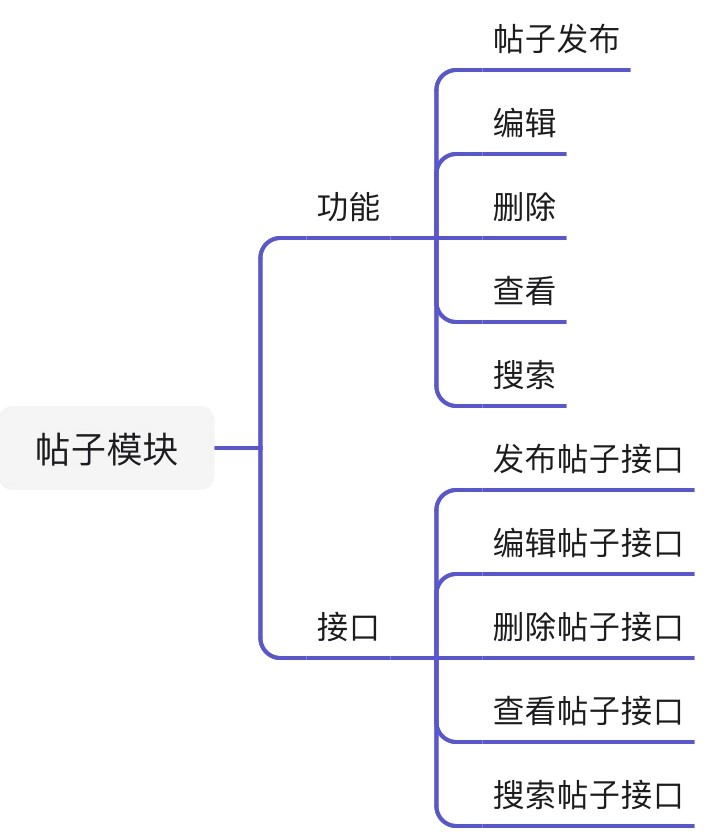
- 验证搜索条件的有效性。

- 如果验证通过，执行搜索操作，根据搜索条件查询符合条件的用户。

- 返回搜索结果，包含符合条件的用户列表或相应的提示信息

**3.2、帖子模块**

3.2.1 功能描述：



3.2.2 接口描述：

1. 输入数据：

- 帖子ID（用于获取帖子信息、更新帖子、删除帖子）。

- 帖子标题、内容、分类等信息（用于创建帖子、更新帖子）。

- 关键字（用于搜索帖子）。

2. 输出数据：

- 帖子的详细信息，包括帖子ID、标题、内容、分类等。

3.2.3 数据结构描述

1. 帖子模块的数据结构包括：

- 帖子信息：包括标题、内容、分类等。

2. 相关数据库表：

- posts：存储帖子的数据表，包括帖子ID、标题、内容、分类等字段。

3.2.4 实现思路

1. 发布帖子：

- 验证用户身份，确保用户已登录。

- 根据输入的帖子标题、内容、分类等信息创建新的帖子。

- 将帖子信息存储到数据库。

- 返回发布的帖子信息。

2. 更新帖子：

- 验证用户身份，确保用户已登录。

- 根据输入的帖子ID查询数据库，获取帖子的详细信息。

- 如果帖子不存在或当前用户不是帖子的发布者，则抛出HTTP异常，返回相应的错误信息。

- 根据输入的更新字段，更新帖子的标题、内容等信息。

- 将更新后的帖子信息存储到数据库。

- 返回更新后的帖子信息。

3. 删除帖子：

- 验证用户身份，确保用户已登录。

- 根据输入的帖子ID查询数据库，获取帖子的详细信息。

- 如果帖子不存在或当前用户不是帖子的发布者，则抛出HTTP异常，返回相应的错误信息。

- 删除帖子及相关的评论等数据。

- 返回删除成功的提示信息。

4. 查看帖子信息：

- 根据输入的帖子ID查询数据库，获取帖子的详细信息。

- 如果帖子不存在，则抛出HTTP异常，返回相应的错误信息。

- 如果帖子存在，则返回帖子的详细信息。

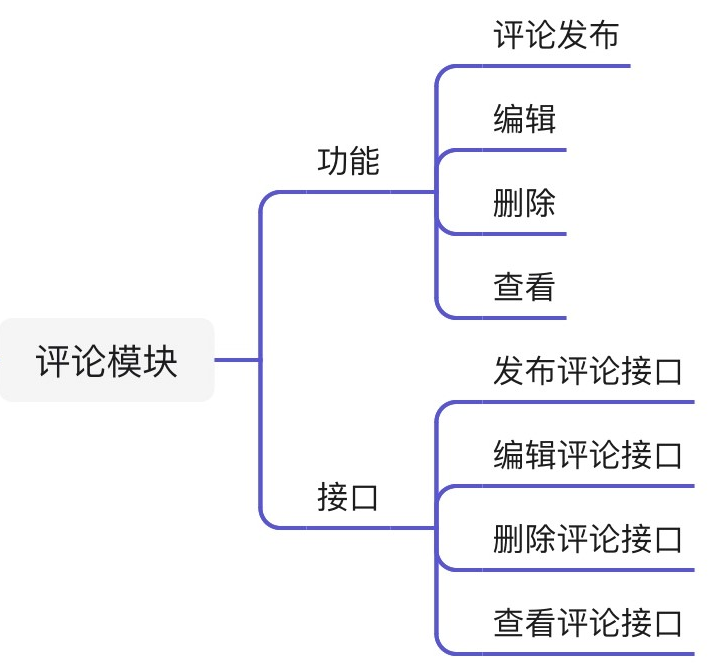
5. 搜索帖子：

- 根据输入的关键字查询数据库，获取符合条件的帖子列表。

- 返回符合条件的帖子列表。

**3.3、评论模块**

3.3.1 功能描述



3.3.2 接口描述

1. 输入数据：用户输入的评论内容，包括评论文本、所评论的帖子或评论的ID等。

2. 输出数据：评论信息，包括评论的ID、评论内容、评论所属帖子或评论的ID、评论者的用户ID等。

3.3.3 数据结构描述

评论信息：包括评论的ID、评论内容、评论所属帖子或评论的ID、评论者的用户ID等。

3.3.4 实现思路

1. 发布评论：

- 接收用户输入的评论内容和所评论的帖子或评论的ID。

- 验证评论内容的有效性。

- 创建评论对象并存储到数据库中。

2. 编辑评论：

- 接收用户输入的评论ID和新的评论内容。

- 验证评论ID的有效性。

- 更新评论对象的评论内容。

3. 删除评论：

- 接收用户输入的评论ID。

- 验证评论ID的有效性。

- 从数据库中删除评论对象。

4. 查看评论：

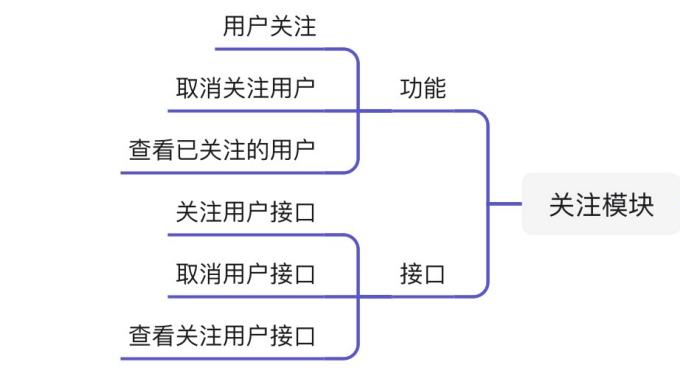
- 接收用户输入的评论ID。

- 验证评论ID的有效性。

- 根据评论ID从数据库中获取评论对象并返回给用户。

**3.4、关注模块**

3.4.1、功能描述



3.4.2、接口描述

1. 输入数据：

- 关注用户接口：当前用户ID和要关注的用户ID。

- 取消关注接口：当前用户ID和要取消关注的用户ID。

- 查看关注用户接口：要查看的用户ID

2. 输出数据：关注模块的输出数据包括执行操作后的成功消息或错误消息。

3.4.3、数据结构描述

用户关注关系：存储用户之间的关注关系，包括关注者ID和被关注者ID。

3.4.4、实现思路

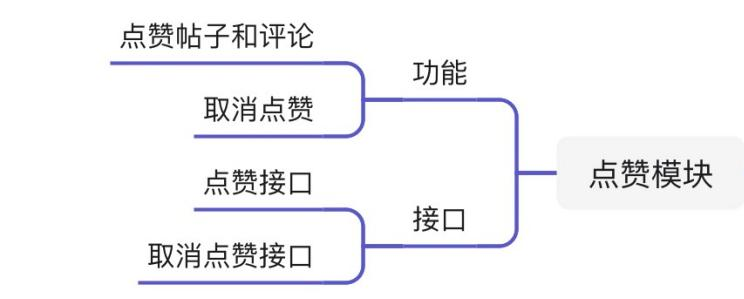
1. 关注用户：验证用户ID的有效性，检查关注关系是否已存在，如果不存在，则在用户关注关系表中创建新的关注关系。

2. 取消关注：验证用户ID的有效性，检查关注关系是否存在，如果存在，则在用户关注关系表中删除关注关系。

3. 查看关注用户：根据当前用户ID在用户关注关系表中检索并返回关注的用户列表。

**3.5、点赞模块**

3.5.1 功能描述



3.5.2 接口描述

1. 输入数据：

- 用户ID：标识进行点赞操作的用户。

- 帖子ID或评论ID：标识用户对哪个评论或帖子进行点赞。

2. 输出数据：操作成功或失败的消息。

3.5.3 数据结构描述

局部数据结构即点赞信息：用于存储用户的点赞信息，包括点赞ID、用户ID、帖子ID或评论ID等。

3.5.4 实现思路

1. 用户发送点赞请求，提供用户ID和内容ID作为输入数据。

2. 系统验证用户身份和权限。

3. 系统检查点赞表中是否已存在该用户对该内容的点赞记录。

- 如果存在，表示用户已经点过赞，返回操作失败的消息。

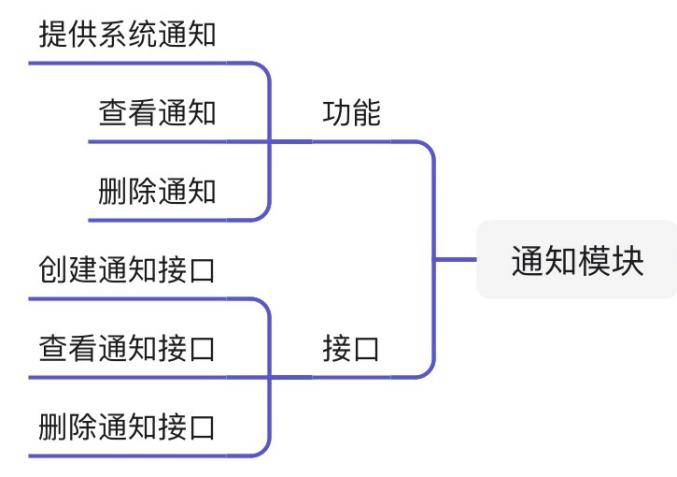
- 如果不存在，执行下一步。

4. 系统在点赞表中创建一条新的点赞记录，记录用户ID和内容ID。

5. 返回操作成功的消息。

**3.6、通知模块**

3.6.1、功能描述：



3.6.2、接口描述：

1. 输入数据：系统发送通知的请求数据，包括通知内容、接收用户等。

2. 输出数据：

- 发送通知的结果，如成功或失败的信息。

- 接收到的通知内容和相关信息。

3.6.3、数据结构描述：

局部数据结构:

- 通知内容: 包含通知标题、内容、发送时间等。

- 用户信息: 包含用户ID、用户名等。

3.6.4、实现思路：

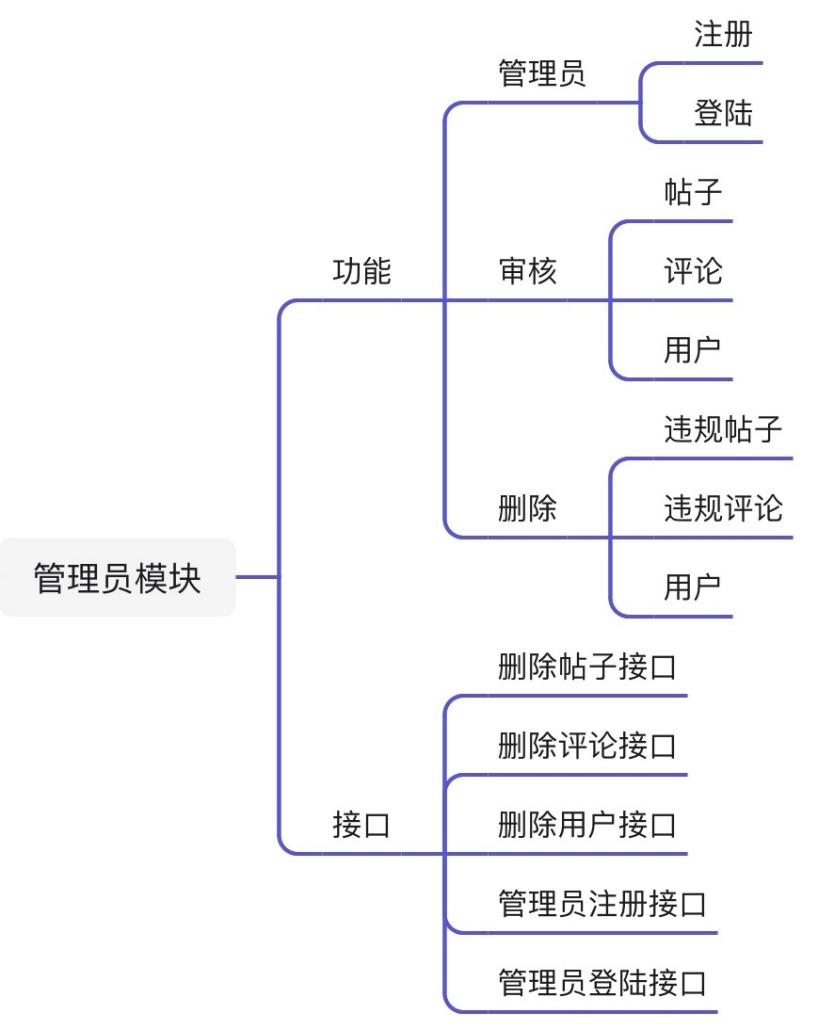
1. 发送通知: 当用户发送通知的请求到达时，根据请求数据创建通知记录，并将通知存储到数据库中。

2. 接收通知: 当用户请求接收通知时，从数据库中检索相关的通知记录，并返回给用户。

3. 管理通知: 用户可以对通知进行管理，包括删除通知、标记通知状态等操作。

**3.7、管理员模块**

3.7.1 功能描述



3.7.2 接口描述

1. 输入数据：管理员的操作指令和相关的数据，例如管理员密码、要删除的用户ID或帖子ID等。

2. 输出数据：根据管理员操作的不同，可能返回相应的提示信息、操作结果或相关数据。

3.7.3 数据结构描述

管理员模块的接口可能涉及以下数据结构：

用户信息：包括用户ID、用户名、邮箱、是否拥有管理员权限等。

帖子信息：包括帖子ID、标题、内容等。

评论信息：包括评论的ID、评论内容、评论所属帖子或评论的ID、评论者的用户ID等。

3.7.4 实现思路

1. 管理员注册流程：

- 接收用户提供的注册信息。

- 验证注册信息的有效性。

- 如果验证通过，将该用户设置为管理员，并保存到数据库中。

- 返回注册成功的提示信息。

1. 管理员登录流程：

- 接收用户提供的登录信息。

- 验证登录信息的有效性。

- 如果验证通过，生成登录凭证（如JWT token）作为身份验证标识。

- 返回登录成功的身份验证信息。

3. 删除用户：

- 管理员可以执行删除用户操作。

- 接收管理员输入的删除指令和相关的用户ID。

- 验证用户ID的有效性。

- 根据操作指令执行相应的删除用户操作，返回操作结果或相应的提示信息。

4. 删除帖子：

- 管理员可以执行删除帖子操作。

- 接收管理员输入的删除指令和相关的帖子ID。

- 验证帖子ID的有效性。

- 根据操作指令执行相应的删除帖子操作，返回操作结果或相应的提示信息。

3. 删除评论：

- 管理员可以执行删除评论操作。

- 接收管理员输入的删除指令和相关的评论ID。

- 验证用户ID的有效性。

- 根据操作指令执行相应的删除用户操作，返回操作结果或相应的提示信息。

# 4、数据库与数据结构设计

**4.1、 数据库及数据表**

A. 数据库系统：Sqlite

B. 数据库和数据表设计：

1. 验证码表（codes）

id: 验证码ID，主键，自增长

code: 验证码，唯一

email: 电子邮箱，唯一

1. 用户表（users）:

id：用户ID，主键

username：用户名，唯一

email：电子邮件，唯一

password\_hash：密码哈希值

avatar\_url：头像URL

introduction：个人介绍

create\_time：注册时间

update\_time：最后一次更新时间

is\_admin：是否为管理员，布尔类型

is\_active：是否已被激活，布尔类型

1. 帖子表（posts）

id：帖子ID，主键

title：帖子标题

content：帖子内容

create\_time：帖子创建时间

update\_time：最后一次更新时间

is\_deleted：是否已被删除，布尔类型

user\_id：发帖用户ID，外键

1. 评论表（comments）

id：评论ID，主键

content：评论内容

create\_time：评论创建时间

update\_time：最后一次更新时间

reply\_user\_id：回复用户ID，默认为0

user\_id：评论用户ID，外键

post\_id：所属帖子ID，外键

parent\_id：父评论ID，外键，可以为空

1. 关注表（follows）

id：关注ID，主键

follower\_id：关注者ID，外键

followed\_id：被关注者ID，外键

1. 点赞表（likes）

id：点赞ID，主键

user\_id：点赞用户ID，外键

post\_id：被点赞帖子ID，外键

comment\_id：被点赞评论ID，外键，可以为空

1. 通知表（notifications）

id：通知ID，主键

content：通知内容

create\_time：通知创建时间

is\_read：是否已读，布尔类型

user\_id：接收通知的用户ID，外键

**4.2、 数据结构设计**

1. 验证码表（codes）

标识符：codes

数据项：

id: 验证码ID，整数类型，主键，自增长

code: 验证码，字符串类型，唯一

email: 电子邮箱，字符串类型，唯一

1. 用户表（users）

标识符：users

数据项：

id: 用户ID，整数类型，主键，自增长

username: 用户名，字符串类型，唯一，长度为20

email: 电子邮件，字符串类型，唯一，长度为64

password\_hash: 密码哈希值，字符串类型

avatar\_url: 头像URL，字符串类型

introduction: 个人介绍，字符串类型

create\_time: 注册时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

is\_admin: 是否为管理员，布尔类型

is\_active: 是否已被激活，布尔类型

1. 帖子表（posts）

标识符：posts

数据项：

id: 帖子ID，整数类型，主键，自增长

title: 帖子标题，字符串类型

content: 帖子内容，字符串类型

create\_time: 帖子创建时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

is\_deleted: 是否已被删除，布尔类型

user\_id: 发帖用户ID，整数类型，外键，关联用户表的id字段

1. 评论表（comments）

标识符：comments

数据项：

id: 评论ID，整数类型，主键，自增长

content: 评论内容，字符串类型

create\_time: 评论创建时间，日期时间类型

update\_time: 最后一次更新时间，日期时间类型

reply\_user\_id: 回复用户ID，整数类型，默认为0

user\_id: 评论用户ID，整数类型，外键，关联用户表的id字段

post\_id: 所属帖子ID，整数类型，外键，关联帖子表的id字段

parent\_id: 父评论ID，整数类型，外键，可以为空，关联评论表的id字段

1. 关注表（follows）

标识符：follows

数据项：

id: 关注ID，整数类型，主键，自增长

follower\_id: 关注者ID，整数类型，外键，关联用户表的id字段

followed\_id: 被关注者ID，整数类型，外键，关联用户表的id字段

1. 点赞表（likes）

标识符：likes

数据项：

id: 点赞ID，整数类型，主键，自增长

user\_id: 点赞用户ID，整数类型，外键，关联用户表的id字段

post\_id: 被点赞帖子ID，整数类型，外键，关联帖子表的id字段

comment\_id: 被点赞评论ID，整数类型，外键，可以为空，关联评论表的id字段

1. 通知表（notifications）

标识符：notifications

数据项：

id: 通知ID，整数类型，主键，自增长

content: 通知内容，字符串类型

create\_time: 通知创建时间，日期时间类型

is\_read: 是否已读，布尔类型

user\_id: 接收通知的用户ID，整数类型，外键，关联用户表的id字段

- 这些表之间的关系如下：

1. 用户表（users）和帖子表（posts）是一对多的关系，即一个用户可以创建多个帖子，但一个帖子只能由一个用户创建。
2. 帖子表（posts）和评论表（comments）也是一对多的关系，即一个帖子可以有多个评论，但一个评论只能属于一个帖子。
3. 评论表（comments）和自身之间存在一种多级关系，即一个评论可以有多个子评论（即回复），也可以没有子评论。
4. 用户表（users）和关注表（follows）之间是多对多的关系，即一个用户可以关注多个其他用户，也可以被多个用户关注。
5. 用户表（users）和点赞表（likes）之间也是多对多的关系，即一个用户可以给多个帖子或评论点赞，一个帖子或评论也可以被多个用户点赞。
6. 用户表（users）和通知表（notifications）是一对多的关系，即一个用户可以收到多条通知，但一条通知只能发送给一个用户。

**4.3、 数据存储设计**

1. 用户模块数据结构：

- 数据项：用户ID、用户名、密码、个人信息

- 存储要求：用户数据可以存储在数据库或持久化存储中。

- 访问方法：可以通过用户ID或用户名进行用户数据的访问和管理。

- 存取单位：每个用户作为一个记录进行存取。

- 存取的物理关系：用户数据可以根据用户ID或用户名进行索引，存储在用户数据表中。

- 设计考虑和保密条件：对用户密码进行安全加密存储，确保个人信息的隐私和安全，实施适当的访问控制机制。

2. 帖子模块数据结构：

- 数据项：帖子ID、发布者用户ID、帖子内容、发布时间、点赞数

- 存储要求：帖子数据可以存储在数据库或持久化存储中。

- 访问方法：可以通过帖子ID或发布者用户ID进行帖子数据的访问和管理。

- 存取单位：每个帖子作为一个记录进行存取。

- 存取的物理关系：帖子数据可以根据帖子ID或发布者用户ID进行索引，存储在帖子数据表中。

- 设计考虑和保密条件：确保只有授权用户可以发布、编辑或删除帖子，对帖子内容进行适当的访问控制和保密措施。

3. 评论模块数据结构：

- 数据项：评论ID、帖子ID、评论者用户ID、评论内容、评论时间、点赞数

- 存储要求：评论数据可以存储在数据库或持久化存储中。

- 访问方法：可以通过帖子ID或评论者用户ID进行评论数据的访问和管理。

- 存取单位：每个评论作为一个记录进行存取。

- 存取的物理关系：评论数据可以根据帖子ID或评论者用户ID进行索引，存储在评论数据表中。

- 设计考虑和保密条件：确保只有授权用户可以发表、编辑或删除评论，对评论内容进行适当的访问控制和保密措施。

4. 关注模块数据结构：

- 数据项：关注者用户ID、被关注者用户ID、关注时间

- 存储要求：关注关系需要存储在数据库或持久化存储中。

- 访问方法：可以通过用户ID快速访问用户的关注列表。

- 存取单位：每个关注关系作为一个记录进行存取。

- 存取的物理关系：关注关系可以根据用户ID进行索引，存储在用户数据表或独立的关注表中。

- 设计考虑和保密条件：确保只有授权用户可以访问关注关系数据，实施适当的身份验证和权限控制机制。

5. 点赞模块数据结构：

- 数据项：用户ID、帖子或评论ID、点赞时间

- 存储要求：点赞记录可以存储在数据库或持久化存储中。

- 访问方法：可以通过帖子或评论ID快速访问相关的点赞记录。

- 存取单位：每个点赞记录作为一个记录进行存取。

- 存取的物理关系：点赞记录可以根据帖子或评论ID进行索引，存储在帖子或评论数据表中。

- 设计考虑和保密条件：确保只有授权用户可以进行点赞操作，防止恶意的刷赞行为，对点赞记录进行安全保密，仅授权用户可见。

6. 通知模块数据结构：

- 数据项：通知ID、接收者用户ID、通知内容

- 存储要求：通知数据可以存储在数据库或持久化存储中。

- 访问方法：可以通过用户ID访问用户的通知列表，并根据通知ID获取特定的通知信息。

- 存取单位：每个通知作为一个记录进行存取。

- 存取的物理关系：通知数据可以根据接收者用户ID进行索引，存储在用户数据表中。

- 设计考虑和保密条件：确保只有接收者用户可以访问和管理自己的通知，对通知内容进行保密处理，实施合适的安全控制措施。

7. 管理员模块数据结构：

- 数据项：用户ID、管理权限、登录密码

- 存储要求：管理员数据可以存储在数据库或持久化存储中。

- 访问方法：可以通过管理权限或登录密码验证管理员的身份和权限。

- 存取单位：每个管理员作为一个记录进行存取。

- 存取的物理关系：管理员数据可以根据管理员ID进行索引，存储在用户数据表中。

- 设计考虑和保密条件：对管理员身份和权限进行严格控制，确保只有授权的管理员可以进行管理操作，保护管理员的登录凭证和密码不被泄露。

# 5、接口设计

**5.1、外部接口**

平台与外界的交互接口：

1. Web应用程序接口：用户通过浏览器访问本系统，需要通过Web应用程序接口进行通信。Web应用程序接口使用HTTP协议，支持GET、POST、DELETE和PUT等请求方式。系统对外提供的接口包括用户模块接口、帖子模块接口、评论模块接口、关注模块接口、点赞模块接口、通知模块接口等。
2. 数据库接口：本系统需要与数据库进行交互，使用关系型数据库管理系统Sqlite。
3. 邮件发送接口：本系统需要向用户发送邮件，包括注册确认、密码重置等功能。系统使用QQ邮箱发送邮件，支持SMTP协议。

**5.2、内部接口**

1. 用户模块：

- 注册接口调用方式：发送POST请求到/register，传递用户注册信息作为请求参数。

- 登录接口调用方式：发送POST请求到/login，传递用户登录信息作为请求参数。

- 注销接口调用方式：发送POST请求到/user，不需要传递参数。

- 个人信息管理接口调用方式：发送PUT请求到/user/update，传递要修改的用户信息作为请求参数。

- 搜索用户接口调用方式：发送POST请求到/user/search，传递要查询的用户名关键词作为请求参数。

1. 帖子模块：

- 创建帖子接口调用方式：发送POST请求到/post/create，传递帖子内容作为请求参数。

- 编辑帖子接口调用方式：发送PUT请求到/post/update，传递帖子ID和要修改的内容作为请求参数。

- 删除帖子接口调用方式：发送DELETE请求到/posts/delete，传递帖子ID作为请求参数。

- 查看帖子接口调用方式：发送GET请求到/posts/{post\_id}，传递帖子ID作为请求参数。

- 搜索帖子接口调用方式：发送POST请求到/posts/search，传递要查询的关键词作为请求参数。

1. 评论模块：

- 发布评论接口调用方式：发送POST请求到/comment/create，传递评论内容和所属帖子ID作为请求参数。

- 删除评论接口调用方式：发送DELETE请求到/comment/delete，传递评论ID作为请求参数。

- 查看评论接口调用方式：发送GET请求到/comments/{comment\_id.id}，传递评论ID作为请求参数。

1. 关注模块：

- 关注用户接口调用方式：发送POST请求到/follow/create，传递被关注用户的ID作为请求参数。

- 取消关注接口调用方式：发送DELETE请求到/follow/delete，传递被取消关注用户的ID作为请求参数。

- 查看用户的关注接口调用方式：发送GET请求到/follow/get\_followed/{user\_id}，传递要查看的用户ID作为请求参数。

- 查看用户的粉丝接口调用方式：发送GET请求到/follow/get\_follower/{user\_id}，传递要查看的用户ID作为请求参数。

1. 点赞模块：

- 点赞接口调用方式：发送POST请求到/like/create，传递帖子或评论的ID作为请求参数。

- 取消点赞接口调用方式：发送DELETE请求到/like/delete，传递帖子或评论的ID作为请求参数。

1. 通知模块：

- 创建通知接口调用方式：发送POST请求到/notification/create，传递通知内容和接收用户ID作为请求参数。

- 查看通知接口调用方式：发送GET请求到/all，不需要传递参数。

- 删除通知接口调用方式：发送DELETE请求到/notification/delete，传递通知ID作为请求参数。

7. 管理员模块：

- 管理员注册调用方式：发送POST请求到/admin/register，传递管理员注册信息作为请求参数。

- 管理员登录接口调用方式：发送POST请求到/admin，传递用户登录信息作为请求参数。

- 删除用户接口调用方式：发送DELETE请求到/admin/users/delete，传递要删除用户的ID作为请求参数。

- 删除帖子接口调用方式：发送DELETE请求到/admin/posts/delete，传递要删除帖子的ID作为请求参数。

- 删除评论接口调用方式：发送DELETE请求到/admin/comment/delete，传递要删除评论的ID作为请求参数。

# 6、其他设计

1. 并行处理：

- 利用并行处理技术来提高系统的性能和响应能力。例如，可以使用多线程或分布式架构来并行处理用户请求、数据存取和业务逻辑等，以实现高效的并发处理。

1. 安全设计：

- 利用OAuth2 实现密码哈希与 Bearer JWT 令牌验证。

- 实施适当的身份验证和权限控制机制，以确保只有授权用户可以进行敏感操作。

- 对用户密码进行加密存储。

- 可使用HTTPS协议进行数据传输加密，实施访问日志记录和监测。

1. 内存管理：

- 优化系统的性能和资源利用。

- 使用合适的数据结构和算法来减少内存占用，定期进行内存清理和释放。

- 尝试使用内存缓存技术来加速数据访问，减少对数据库的频繁访问。

4. 扩展性和可维护性：

- 设计有良好扩展性和可维护性的系统架构，来适应平台不断增长的用户和数据量。

- 进行模块化设计，清晰定义和规范接口，代码结构易于扩展和修改，系统监测和错误处理机制合理。