个人博客系统

软件（结构）设计说明（SAD）

教师：余仲星

队员：戴梦琪 张政航 许嘉诺

郑春英 姚佳悦

日期：2025/5/30

目录

1 引言 4
1.1 标识 4
1.2 系统概述 4
1.3 文档概述 5
1.4 基线 5
2 引用文件 6
3 CSCI 级设计决策 6
3.1 关于用户权限的设计决策： 6
3.2 关于管理员权限的设计决策： 6
4 CSCI 体系结构设计 7
4.1 体系结构 7
4.1.1 程序(模块)划分 7
4.1.2 程序(模块)层次结构关系 8
4.2 全局数据结构说明 9
4.2.1 常量 9
4.2.2 变量 9
4.2.3 数据结构 12
4.3 CSCI 部件 14
4.3.1 软件配置项及用途 14
4.3.2 软件配置项之间的静态关系 15
4.3.3 软件配置项分配的 CSCI 需求和 CSCI 级设计决策 15
4.3.4 软件配置项的开发状态/类型 16
4.3.5 CSCI 计划使用的计算机硬件资源 16
4.3.6 软件配置项之间的动态关系 17
4.3.7 软件配置项的质量要求 17
4.4 执行概念 18
4.5 接口设计 20
4.5.1 软件内部接口 20
4.5.2 软件外部接口 21
4.5.3 用户接口 25
4.5.4 接口约束与协议 25
5 CSCI 详细设计 27
5.1 用户管理模块 27
5.1.1 概述 27
5.1.2 软件配置项设计中的约束、限制或非常规特征 27
5.1.3 编程语言和使用理由 27
5.1.4 过程式命令列表 28
5.1.5 逻辑说明 28
5.2 文章管理模块 29

5.2.1 概述 29
5.2.2 软件配置项设计中的约束、限制或非常规特征 29
5.2.3 编程语言和使用理由 30
5.2.4 过程式命令列表 30
5.2.5 逻辑说明 30
5.3 文章管理员管理模块 31
5.3.1 概述 32
5.3.2 软件配置项设计中的约束、限制或非常规特征 32
5.3.3 编程语言和使用理由 32
5.3.4 过程式命令列表 32
5.3.5 逻辑说明 33
5.4 查询与统计 34
5.4.1 概述 34
5.4.2 软件配置项设计中的约束、限制或非常规特征 34
5.4.3 编程语言和使用理由 35
5.4.4 过程式命令列表 35
5.4.5 逻辑说明 35
6 需求的可追踪性 36
6.1 CSCI 需求可追踪性 36
6.2 CSCI 需求到它被分配给的软件配置项的可追踪性 37
7 注解 37
7.1 背景信息 37
7.2 术语和定义 38
7.3 缩略语列表 38

1 引言

1.1 标识

名称：新迹（NexTecht）

标识号：BT 1.0

版本号：Version 1.0

发行号：XX-XX-XX-XX

适用系统：具有 web 的系统

1.2 系统概述

（1）适用系统：具有 web 的系统。

（2）用途：本个人博客系统旨在为用户提供一个便捷、高效的博客创作与管理平台。系统面向普通用户和管理员，支持用户进行内容发布、评论互动、个人信息维护等操作，同时为管理员提供后台管理工具，用于内容审核、用户管理及系统维护等。该系统具有如下主要功能和特点：

① 支持丰富文本博客创作与发布；

② 提供评论、点赞、收藏等互动功能，提升用户参与度；

③ 具备用户注册、登录与个人资料管理模块；

④ 后台管理系统支持文章管理、用户管理与数据监控；

⑤ 系统采用模块化设计，具备良好的可扩展性和可维护性；

⑥ 适用于个人博客搭建、小型内容发布平台或学习实验项目。

（3）项目于 2025 年 3 月启动，由软件大作战项目小组成员共同参与需求分析、系统设计与功能开发等全过程。目前系统仍处于开发阶段，正在逐步实现博客内容发布、用户注册登录、评论互动、管理员后台管理等核心功能。

（4）本系统采用前后端分离架构，前端使用 Vue 框架进行开发，后端采用 Flask 框架构建 RESTful API，并结合 MySQL 实现数据持久化存储。系统开发过程大致分为以下几个阶段：

① 需求分析阶段：明确系统的目标用户与功能需求，制定整体开发计划；

② 设计阶段：完成系统架构、数据库模型、接口规范及前端界面原型的

设计；

③ 开发阶段：各模块正在开发中，前端与后端协同进行接口联调；

④ 测试与部署阶段（计划中）：完成主要功能后将进行系统测试与优化，并部署到本地或云服务器运行。

（5）相关方

投资方/需方：软件大作战团队；

用户：普通用户、自媒体人；

开发方：软件大作战团队。

运行现场：当前本系统主要在开发者本地环境中运行，前端使用 Vue 运于浏览器环境中，后端基于 Flask 框架运行于本地 Python 环境，MySQL 数据库部署在本地数据库服务器。当前开发和测试均在 Windows 操作系统下进行。

（6）其他有关的文档

《可行性分析（研究）报告》（FAR）1.0

1.3 文档概述

（1）用途：评估项目的技术、经济、操作和法律可行性，为后续开发提供决策依据。

（2）内容：包括背景、技术选择、成本估算、风险分析等多方面分析，确定项目的可行性。

（3）保密性：本文档仅限项目团队及投资方内部使用，禁止外传。

1.4 基线

编写本系统设计说明书所依据的设计基线为：

《可行性分析（研究）报告》（FAR）1.0

2 引用文件

（1）《MySQL 8.0 参考手册》

编号：DOC-1 版本：8.0 发行日期：2022-11-18

（2）《中华人民共和国个人信息保护法》

编号：DOC-2 版本：1.0 发行日期：2021-11-01

（3）《中华人民共和国网络安全法》

编号：DOC-3 版本：1.0 发行日期：2016-11-07

3 CSCI 级设计决策

系统分为用户（发博客的作者）视角和管理员（管理博客网站的人员）视角，为两种身份分别设置不同的权限。

3.1 关于用户权限的设计决策：

支持注册/登录，编辑个人信息，发表文章，与他人的博客文章进行点赞评论等互动。

（1）设计信息录入接口

与用户接口：提供登录窗口，用户可以选择注册账号或登录已有账号；提供发表文章页面，用户可以撰写文章并发表在我们的博客网站。

与数据库接口：设计合适的数据库表结构和存储过程，以便存储用户信息（包括其个人介绍、发表过的文章等）

（2）设计信息修改接口

与用户接口：提供修改功能按钮，用户可对个人信息，已发布的文章内容进行重新编辑

与数据库接口：实现数据库的更新操作，确保修改后的信息立即生效。

3.2 关于管理员权限的设计决策：

设置两种级别的管理员，普通管理员和超级管理员，普通管理员可以管理博客网站上的用户，包括管理他们的用户状态（正常、封禁）、发布权限、评

论权限等；超级管理员可管理博客网站上的用户（与普通管理员的权限一致）和普通管理员，如对普通管理员的增加、删除、查找。

与普通管理员接口：建立用户表，包含用户状态、发布权限、评论权限，普通管理员可编辑这些表的内容，以确定当前用户的权限。

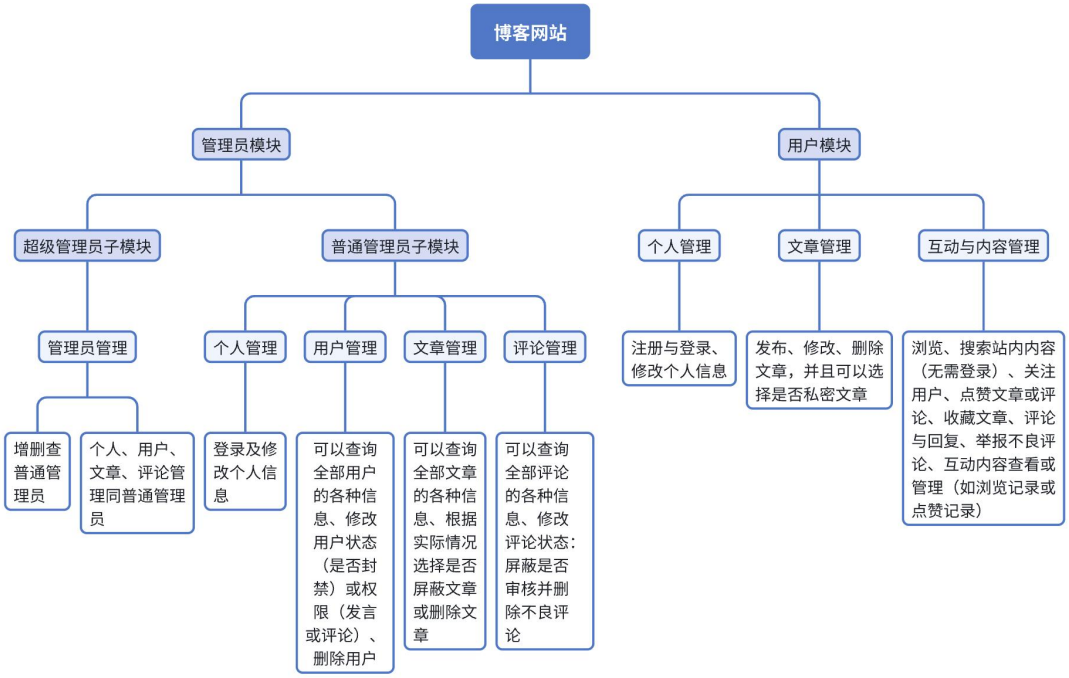
与超级管理员接口：建立管理员表，记录管理员信息，超级管理员可编辑用户表及管理员表，可增加、删除、查找普通管理员。

4 CSCI 体系结构设计

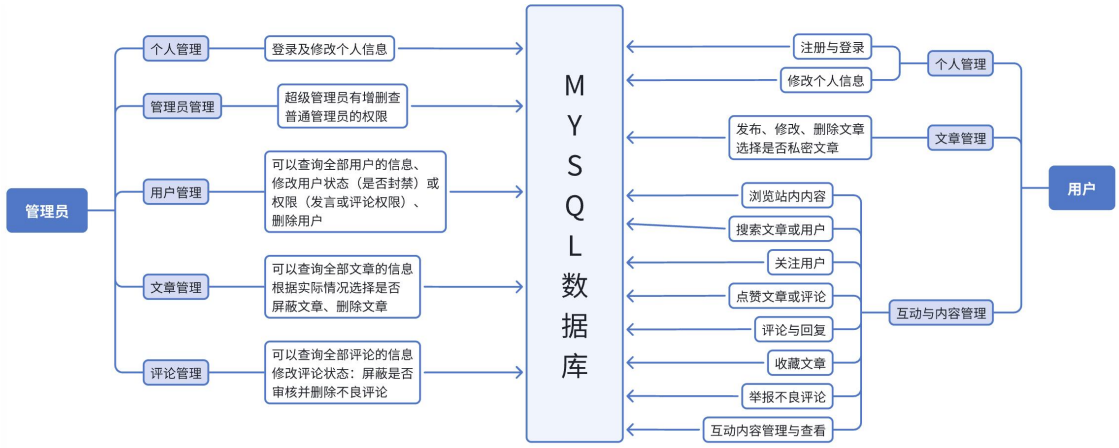
4.1 体系结构

4.1.1 程序(模块)划分

本博客系统的程序模块划分如下图所示：



4.1.2 程序(模块)层次结构关系



本 CSCI 内的每个程序(包括每个模块和子程序)之间的层次结构与调用关系如上图所示。

MYSQL 数据库：博客系统的核心，负责存储所有管理员和用户的相关数据，比如用户的个人信息、博客内容、浏览记录、互动记录等等。

数据库的设计确保了数据的一致性、完整性和安全性。

数据表信息如下：

名称	字段	作用
用户表	用户 ID、用户名、性别、昵称、头像、个人介绍、手机号、邮箱、密码、注册时间、用户状态、权限状态、登录状态、最后登录时间	记录用户的信息
管理员表	管理员 ID、用户名、昵称、手机号、邮箱、密码、性别、头像、注册时间	记录管理员信息
文章表	文章 ID、作者、标题、内容、创建时间、更新时间、私密状态、文章状态、标签分类、阅读量、点赞数、收藏数、评论数	记录文章信息
评论表	评论 ID、内容、评论时间、用户、状态、点赞数、回复数、评论的文章	记录评论信息
文章点赞表	点赞 ID、用户、点赞的文章、点赞时间	记录点赞信息
评论点赞表	点赞 ID、用户、点赞的评论、点赞时间	记录点赞

		信息
文章收藏表	收藏 ID、用户、收藏的文章、收藏时间	记录收藏信息
关注表	关注 ID、关注用户、被关注用户、时间	记录关注信息
浏览记录表	记录 ID、用户、浏览的文章、浏览时间	记录浏览记录

4.2 全局数据结构说明

4.2.1 常量

（1）User（用户表）

U_status：用户状态，0：正常（默认），1：禁止用户的所有权限

is_publish：发布权限，1：可以（默认），0：不可以

is_comment：评论权限，1：可以（默认），0：不可以

（2）Article（文章表）

permission：文章权限，0 表示公开，1 表示屏蔽

status：文章状态，0 表示已发布，1 表示已删除，2 表示被举报

（3）Comment（评论表）

status：评论状态，0 表示正常，1 表示已删除，2 表示屏蔽，3 表示被举报

is_approved：评论审核状态，0 表示待审核，1 表示已通过，2 表示未通过

depth：评论深度，0 表示顶级，大于 0 表示嵌套

4.2.2 变量

（1）User（用户表）

id：用户主键，自增生成

username：用户名，唯一，用户注册时输入

password_hash：加密后的密码

gender：用户性别

nickname：昵称

intro：个人简介，选填

avatar：头像 URL，选填

create_at：注册时间，默认当前时间

phone：手机号，唯一，选填

email：邮箱地址，唯一

verification_code_expiry：验证码过期时间

last_login_at：用户最后登录时间

is_online：是否在线状态

（2）Manager（管理员表）

mng_id：管理员主键，自增生成

mng_name：管理员用户名，唯一

mng_phone：手机号，唯一

mng_email：邮箱地址，唯一

mng_password_hash：加密后的密码

mng_gender：性别

mng_nickname：昵称

mng_avatar：头像 URL，选填

mng_create_at：注册时间，默认当前时间

（3）Article（文章表）

id：文章主键，自增生成

title：文章标题

content：文章正文内容

create_time：创建时间，系统默认当前时间

update_time：更新时间，自动更新

user_id：关联用户 ID（外键）

read_count：阅读数

image_path：文章封面图片路径

tag：文章标签分类

like_count：文章点赞数

favorite_count：文章收藏数

（4）Comment（评论表）

id：评论主键，自增生成

content：评论内容

create_time：评论创建时间

update_time：评论更新时间

like_count：评论点赞数

reply_count：评论回复数

user_id：评论所属用户 ID

article_id：评论所属文章 ID

parent_id：父评论 ID（用于嵌套）

（5）Alike（文章点赞表）

id：点赞记录主键，自增生成

user_id：点赞用户 ID

article_id：被点赞的文章 ID

create_time：点赞时间

（6）CommentLike（评论点赞表）

id：评论点赞记录主键，自增生成

user_id：点赞用户 ID

comment_id：被点赞的评论 ID

created_at：点赞时间

（7）ArticleFavorite（文章收藏表）

id：收藏记录主键，自增生成

user_id：收藏文章的用户 ID，用户行为触发，外键

article_id：被收藏文章的 ID，用户行为触发，外键

create_time：收藏时间，由系统记录，随用户操作变化

（8）Follow（关注表）

id：关注记录主键，自增生成

follower_id：发起关注的用户 ID，用户行为产生，外键

followed_id：被关注的用户 ID，用户行为产生，外键

timestamp：关注操作发生的时间，由系统生成

（9）UserBrowseRecord（浏览记录表）

id：浏览记录主键，自增生成

user_id：发起浏览的用户 ID

article_id：被浏览的文章 ID

browse_time：浏览时间，由系统自动记录

4.2.3 数据结构

（1）用户类

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # 自增的主键
    username = db.Column(db.String(80), unique=True, nullable=False)  # 用户名，唯一
    password_hash = db.Column(db.String(512), nullable=False)  # 存储加密后的密码
    gender = db.Column(db.String(10), nullable=False)  # 性别
    nickname = db.Column(db.String(80), nullable=False)  # 昵称
    intro = db.Column(db.Text, nullable=True)  # 个人介绍，选填
    avatar = db.Column(db.String(256), nullable=True)  # 头像 URL，选填
    create_at = db.Column(db.DateTime, default=datetime.utcnow)  # 注册时间，默认当前时间
    phone = db.Column(db.String(20), unique=True, nullable=True)  # 手机号，选填
    email = db.Column(db.String(120), unique=True, nullable=False)  # 邮箱，唯一
    verification_code = db.Column(db.String(10), nullable=True)
    verification_code_expiry = db.Column(db.DateTime, nullable=True)  # 存储验证码的过期时间
    u_status = db.Column(db.Integer, default=0)  # 用户状态，0：正常（默认），1：禁止用户的所有权限
    is_publish = db.Column(db.Integer, default=1)  # 发布文章权限,1:可以（默认），0:不可以
    is_comment = db.Column(db.Integer, default=1)  # 评论权限,1:可以（默认），0:不可以
    last_login_at = db.Column(db.DateTime)  # 最后登录时间
    is_online = db.Column(db.Boolean, default=False)  # 是否在线
```

（2）管理员类

```
class Manager(db.Model):
    mng_id = db.Column(db.Integer, primary_key=True)  # 自增的主键  1
    mng_name = db.Column(db.String(40), unique=True, nullable=False)  # 用户名，唯一  2
    mng_phone = db.Column(db.String(20), unique=True, nullable=False)  # 手机号，选填,唯一 3
    mng_email = db.Column(db.String(120), unique=True, nullable=False)  # 邮箱，唯一 4
    mng_password_hash = db.Column(db.String(512), nullable=False)  # 存储加密后的密码 5
    mng_gender = db.Column(db.String(10), nullable=False)  # 性别 6
    mng_nickname = db.Column(db.String(80), nullable=False)  # 昵称 7
    mng_avatar = db.Column(db.String(256), nullable=True)  # 头像 URL，选填 8
    mng_create_at = db.Column(db.DateTime, default=datetime.utcnow)  # 注册时间，默认当前时间 9
```

（3）文章类

```python
class Article(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # 自增的主键，文章 ID
    title = db.Column(db.String(256), nullable=False)  # 文章标题
    content = db.Column(db.Text, nullable=False)  # 文章内容
    create_time = db.Column(db.DateTime, default=datetime.utcnow)  # 文章创建时间，默认为当前时间
    update_time = db.Column(db.DateTime, onupdate=datetime.utcnow)  # 文章更新时间
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  # 关联用户 ID，外键
    user = db.relationship('User', backref=db.backref('articles', lazy=True))  # 建立与用户模型的关联

    permission = db.Column(db.Integer, default=0)  # 文章权限位，0表示公开，1表示屏蔽
    status = db.Column(db.Integer, default=0)  # 文章状态位，0表示已发布，1表示已删除，2表示被举报
    read_count = db.Column(db.Integer, default=0)
    image_path = db.Column(db.String(256))  # 图片路径字段
    tag = db.Column(db.String(128))  # 文章分类标签字段
    like_count = db.Column(db.Integer, default=0)  # 文章点赞数
    favorite_count = db.Column(db.Integer, default=0)  # 文章收藏数
```

（4）评论类

```python
class Comment(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # 自增的主键，评论 ID
    content = db.Column(db.Text, nullable=False)  # 评论内容
    create_time = db.Column(db.DateTime, default=datetime.utcnow)  # 评论创建时间，默认为当前时间
    update_time = db.Column(db.DateTime, onupdate=datetime.utcnow)  # 评论更新时间
    status = db.Column(db.Integer, default=0)  # 评论状态，0: 正常，1: 已删除，2: 屏蔽，3: 举报
    is_approved = db.Column(db.Integer, default=0)  # 审核状态，0: 待审核，1: 已通过，2: 未通过
    like_count = db.Column(db.Integer, default=0)  # 点赞数
    reply_count = db.Column(db.Integer, default=0)  # 回复数
    depth = db.Column(db.Integer, default=0)  # 评论深度，用于区分顶级评论和回复

    # 关联字段
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  # 关联用户 ID，外键
    user = db.relationship('User', backref=db.backref('comments', lazy=True))  # 与用户模型的关联
    article_id = db.Column(db.Integer, db.ForeignKey('article.id'), nullable=False)  # 关联文章 ID，外键
    article = db.relationship('Article', backref=db.backref('comments', lazy=True))  # 与文章模型的关联
    parent_id = db.Column(db.Integer, db.ForeignKey('comment.id'), nullable=True)  # 父评论 ID，用于回复
    parent_comment = db.relationship('Comment', backref=db.backref('replies', lazy=True), remote_side=[id])  # 与父评论的关联
```

（5）文章点赞类

```python
class Alike(db.Model):
    id = db.Column(db.Integer, primary_key=True)  # 自增的主键，点赞 ID
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  # 外键，指向 User 表
    article_id = db.Column(db.Integer, db.ForeignKey('article.id'), nullable=False)  # 外键，指向 Article 表
    create_time = db.Column(db.DateTime, default=datetime.utcnow)  # 点赞时间

    user = db.relationship('User', backref=db.backref('likes', lazy=True))  # 与 User 的关系
    article = db.relationship('Article', backref=db.backref('likes', lazy=True))  # 与 Article 的关系
```

（6）评论点赞类

```python
class CommentLike(db.Model):
    __tablename__ = 'comment_like'
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    comment_id = db.Column(db.Integer, db.ForeignKey('comment.id'), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    __table_args__ = (db.UniqueConstraint('user_id', 'comment_id', name='_user_comment_uc'),)

    user = db.relationship('User', backref=db.backref('comment_likes', lazy='dynamic'))
    comment = db.relationship('Comment', backref=db.backref('comment_likes', lazy='dynamic'))
```

（7）文章收藏类

```
class ArticleFavorite(db.Model):
    __tablename__ = 'article_favorite'  # 指定表名

    id = db.Column(db.Integer, primary_key=True)  # 自增的主键，收藏记录 ID
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  # 收藏用户的 ID，外键
    user = db.relationship('User', backref=db.backref('article_favorites', lazy=True))  # 与用户模型的关联
    article_id = db.Column(db.Integer, db.ForeignKey('article.id'), nullable=False)  # 被收藏的文章 ID，外键
    article = db.relationship('Article', backref=db.backref('favorites', lazy=True))  # 与文章模型的关联
    create_time = db.Column(db.DateTime, default=datetime.utcnow)  # 收藏时间，默认为当前时间
```

（8）关注类

```
class Follow(db.Model):
    id = db.Column(db.Integer, primary_key=True) #关注ID
    follower_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) #关注者
    followed_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) #被关注者
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)  # 关注的时间戳
```

（9）浏览记录类

```
class UserBrowseRecord(db.Model):
    __tablename__ = 'user_browse_record'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    article_id = db.Column(db.Integer, db.ForeignKey('article.id'), nullable=False)
    browse_time = db.Column(db.DateTime, default=datetime.utcnow)

    user = db.relationship('User', backref=db.backref('records', lazy=True))  # 与 User 的关系
    article = db.relationship('Article', backref=db.backref('records', lazy=True))  # 与 Article 的关系
```

4.3 CSCI 部件

4.3.1 软件配置项及用途

（1）用户管理模块（User Management Module）

负责用户注册、登录、身份验证和个人信息管理功能。

（2）文章管理模块（Article Management Module）

处理文章的发布、编辑、删除、查看和搜索功能。

（3）首页展示模块（Homepage Display Module）

负责首页内容展示，包括热榜、关注用户动态和个性化推荐。

（4）互动功能模块（Interaction Module）

处理用户间的互动功能，包括点赞、收藏、评论和关注。

（5）个人中心模块（Personal Center Module）

管理用户个人页面，展示个人文章、收藏、浏览历史等信息。

（6）管理员控制模块（Admin Control Module）

提供管理员后台功能，包括用户管理、内容审核和系统监控。

（7）数据访问层模块（Data Access Layer Module）

封装数据库操作，提供统一的数据访问接口。

（8）安全认证模块（Security Authentication Module）

处理系统安全相关功能，包括权限控制和数据加密。

4.3.2 软件配置项之间的静态关系

（1）分层架构关系

包括表示层，业务逻辑层，数据访问层。其中，表示层包括用户管理模块、首页展示模块、个人中心模块和管理员控制模块，业务逻辑层包括文章管理模块、互动功能模块和安全认证模块，数据访问层包括数据访问层模块。

（2）模块依赖关系

所有业务模块依赖于安全认证模块进行权限验证，首页展示模块依赖于文章管理模块和用户管理模块获取数据，互动功能模块依赖于用户管理模块和文章管理模块，管理员控制模块依赖于所有其他模块进行管理操作，所有模块依赖于数据访问层模块进行数据持久化。

（3）接口关系

各模块通过定义的 API 接口进行通信，同时采用标准接口方便统一管理和调试。

4.3.3 软件配置项分配的 CSCI 需求和 CSCI 级设计决策

（1）用户管理模块需求分配

功能需求：用户注册登录、信息修改、密码管理等。

设计决策：采用 JWT 令牌进行会话管理，使用哈希进行密码加密。

（2）文章管理模块需求分配

功能需求：文章创建、读取、更新、管理操作、分类管理、搜索功能等。

设计决策：采用搜索算法实现全文搜索。

（3）首页展示模块需求分配

功能需求：热榜展示、关注动态、个性化推荐等。

设计决策：采用算法排序实现热榜。

（4）互动功能模块需求分配

功能需求：点赞、收藏、评论、关注功能等。

设计决策：采用异步处理提高响应速度，使用消息队列处理高并发操作

（5）管理员控制模块需求分配

功能需求：内容审核、用户管理、系统监控等。

设计决策：基于角色的权限控制，提供可视化管理界面

4.3.4 软件配置项的开发状态/类型

（1）新开发组件

用户管理模块：自主开发，实现项目特定的用户管理需求

文章管理模块：自主开发，满足博客系统的核心功能

首页展示模块：自主开发，提供个性化的用户体验

（2）框架集成组件

数据访问层模块：基于 MySQL 框架开发，提供标准化数据操作

（3）第三方集成组件

搜索功能：搜素算法，提供高效的全文搜索，匹配对应的字符串。

缓存系统：提升系统性能。

（4）可复用组件

通用工具类：日期处理、字符串操作、图片上传等。

通用 UI 组件：分页组件、表格组件、表单验证组件。

4.3.5 CSCI 计划使用的计算机硬件资源

（1）服务器资源需求

CPU：4 核心及以上，支持高并发处理。

内存：8GB 及以上，确保系统流畅运行。

存储：SSD 硬盘 500GB 及以上，提供快速的数据读写。

网络：100Mbps 及以上带宽，保证用户访问体验。

（2）数据库服务器资源

CPU：2 核心及以上，处理数据库查询操作。

内存：4GB 及以上，支持数据缓存和索引。

存储：SSD 硬盘 200GB 及以上，存储用户和文章数据。

（3）缓存服务器资源

CPU：2 核心，处理缓存读写操作。

内存：2GB 及以上，存储热点数据。

存储：50GB，持久化缓存数据。

4.3.6 软件配置项之间的动态关系

（1）用户操作流程

用户通过用户管理模块进行登录认证；

安全认证模块验证用户身份并生成访问令牌；

首页展示模块调用文章管理模块获取内容数据；

用户进行互动操作时，互动功能模块记录行为数据；

数据访问层模块将所有操作持久化到数据库；

（2）管理员操作流程

管理员通过管理员登陆模块进行身份验证；

确认管理员权限；

管理员控制模块调用各业务模块进行管理操作；

审核操作触发相应模块的状态更新；

系统日志记录所有管理操作；

（3）数据流动关系

用户行为数据从互动功能模块流向首页展示模块用于推荐算法；

文章数据从文章管理模块流向搜索索引和缓存系统；

审核结果从管理员控制模块流向相关内容模块更新状态；

4.3.7 软件配置项的质量要求

（1）性能要求

系统响应时间：页面加载时间不超过 3 秒；

并发处理能力：支持 1000 个并发用户同时在线；

数据库查询效率：单次查询响应时间不超过 100 毫秒；

文件上传速度：支持 10MB 以内文件的快速上传；

（2）可靠性要求

系统可用性：99.5%以上的系统正常运行时间；

错误处理：提供完善的异常处理和错误恢复机制；

数据完整性：确保用户数据和文章内容的完整性和一致性；

备份机制：实现自动数据备份和快速恢复功能；

（3）安全性要求

身份认证：采用多层次的用户身份验证机制；

数据加密：敏感数据采用 AES 加密算法保护；

权限控制：实现细粒度的用户权限管理；

防护措施：具备 SQL 注入、XSS 攻击等常见安全威胁的防护能力；

（4）可维护性要求

代码规范：遵循统一的编码规范和注释标准；

模块化设计：采用松耦合、高内聚的模块化架构；

日志记录：提供详细的系统运行日志和错误日志；

文档完整性：提供完整的技术文档和用户手册；

（5）可扩展性要求

架构灵活性：支持功能模块的独立扩展和升级；

数据库扩展：支持数据库的水平和垂直扩展；

负载均衡：支持多服务器部署和负载均衡配置；

接口标准化：提供标准化的 API 接口便于第三方集成；

4.4 执行概念

博客网站系统采用分层设计，各个功能模块在运行时通过相互调用来完成用户的各种操作。系统主要基于用户请求和响应的方式工作，当用户在网页上进行操作时，系统会按照预定的流程来处理这些请求并返回结果。

当用户打开博客网站进行登录时，首先会触发用户管理模块的登录验证过程。系统收到用户输入的用户名和密码后，用户管理模块会调用安全认证模块来检查用户信息是否正确。安全认证模块通过数据访问层模块查询数据库中的用户信息，如果验证成功就会生成一个登录凭证返回给用户。这个过程是按顺序执行的，确保用户身份验证的准确性。验证成功后，系统会记住用户的登录状态，用户在后续操作中不需要重复登录。

首页内容的加载展现了系统如何同时处理多个任务。当用户访问首页时，首页展示模块需要同时获取多种不同的内容，包括热门文章、用户关注的人发布的文章，以及系统推荐的文章。为了提高效率，系统会同时向文章管理模块请求热门文章数据，向用户管理模块请求关注信息，这些任务可以并行执行而不互相等待。所有数据收集完成后，首页展示模块会将这些内容整合在一起显示给用户。系统使用了多线程技术来处理这些并发操作，确保在用户较多时网站仍能正常运行。

文章的发布和阅读操作体现了系统的数据处理流程。当用户发布新文章时，文章管理模块首先会检查文章内容是否符合要求，然后将文章信息保存到数据库中。为了支持文章搜索功能，系统还会将文章内容添加到搜索系统中，这个过程在后台执行，不会影响用户的操作体验。文章发布成功后，系统会更新相关的缓存数据，以便其他用户能够及时看到新发布的内容。

用户的互动操作如点赞、收藏、评论等具有频繁和快速的特点。当用户点击点赞按钮时，互动功能模块会立即响应用户的操作，让用户感觉到操作成功了。但实际的数据更新工作会放在后台队列中处理，这样可以避免用户界面卡顿。系统维护了一个快速的计数系统，用来记录每篇文章的点赞数、收藏数等信息，当用户查看这些数据时能够快速显示最新的统计结果。

管理员的后台操作有着特殊的执行权限和处理方式。当管理员登录后台系统时，管理员控制模块会首先验证用户是否具有管理员权限。管理员进行文章审核或用户管理操作时，系统会记录这些操作的详细信息，包括操作时间、操作内容等。对于批量处理操作，比如审核多篇文章，系统会分批处理这些请求，避免一次性处理太多数据导致系统响应缓慢。所有管理操作都会在系统中留下记录，方便后续查询和追踪。

系统具有完善的错误处理机制来应对各种异常情况。当某个模块出现问题时，系统不会完全停止工作，而是会记录错误信息并尝试恢复正常状态。对于网络连接中断、数据库访问失败等临时性问题，系统会自动重试几次；对于严重的系统错误，系统会启动备用方案，确保网站的基本功能仍然可用。用户在使用过程中如果遇到错误，会看到友好的错误提示信息而不是复杂的技术错误代码。

时间控制在系统运行中发挥重要作用。用户的登录状态有一定的有效期，系统会定期检查并清理过期的登录信息。系统中的缓存数据也有时效性，过期的缓存会被自动清理并更新为最新数据。系统还会在固定时间执行一些维护任务，比如清理临时文件、计算统计数据、备份重要信息等。

各个功能模块之间通过标准的接口进行通信，每个模块都有明确的职责分工。重要的业务流程都有明确的步骤和状态，系统会跟踪这些流程的执行情况，确保每个步骤都能正确完成。这种设计使得系统易于理解和维护，当需要添加新功能或修改现有功能时，可以在不影响其他模块的情况下进行修改。

4.5 接口设计

本节描述个人博客系统中各软件配置项之间的接口特性，以及系统与外部实体（如数据库、前端界面、第三方平台等）之间的接口。接口设计旨在确保系统模块之间的有效通信，增强系统的可扩展性、可维护性和可测试性。

个人博客系统主要包含前端界面、后端服务、数据库以及可选的第三方登录或评论系统等组件。各模块之间通过 API 或数据库接口进行通信，前后端通过 HTTP 协议交换数据，采用 JSON 和 FORM 格式传输信息。

4.5.1 软件内部接口

功能模块	接口名称	接口类型	接口用途
用户管理	set_password()	函数调用	加密用户密码
	check_password()	函数调用	验证密码是否匹配
	create_access_token()	函数调用	创建令牌
	generate_verification_code()	函数调用	生成邮箱验证码
	send_email_verification_code()	函数调用	发送邮箱验证码
	set_verification_code_expiry()	函数调用	设置验证码有效时长
	set_password()	函数调用	加密用户密码

管理员管理	check_password()	函数调用	验证密码是否匹配
	create_access_token()	函数调用	创建令牌
	mng_dict()	函数调用	将数据封装成 json 形式
文章管理	update_article()	函数调用	更新文章内容
	delete_article()	函数调用	删除文章
	get_articles_by_status()	函数调用	获取文章内容
	set_permission()	函数调用	设置文章权限
	set_status()	函数调用	设置文章状态
评论管理	update_comment()	函数调用	更新评论
	delete_comment()	函数调用	删除评论
	report_comment()	函数调用	反驳评论
	like_comment()	函数调用	点赞评论

4.5.2 软件外部接口

（1）前后端接口

功能模块	函数名称	URL 路径	接口用途
用户管理	register_user	/user/register	添加新用户到系统
	login_user	/user/login	从系统中登录指定用户
	send_verification_code	/user/sendEmailCode	发送验证码接口
	verify_code	/user/verifyEmailCode	验证邮箱验证码
	update_profile	/user/updateProfile	更新用户信息
	logout_user	/user/logout	用户退出登录

管理员管理	manager_login	/manager/login	管理员登录
	add_manager	/manager/add_mng	增加管理员
	delete_manager	/manager/delete_mng	删掉管理员
	list_managers	/manager/manager_list	列出所有管理员
	get_manager_profile	/manager/profile	管理员查看个人信息接口
	update_manager_profile	/manager/update_profile	管理员修改个人信息接口
	list_users	/manager/user_list	管理员查看用户列表接口
	update_user_status	/manager/update_user_status	管理员修改用户状态接口
	update_user_permission	/manager/update_user_permission	管理员修改用户权限接口
文章管理	get_all_articles	/manager/article_list	获取所有文章列表（管理员专用）
	get_article_detail	/manager/article_detail/<int:article_id>	获取特定文章详情（管理员专用）
	change_article_status_by_admin	/article/manager/update/<int:article_id>/status	修改文章状态（管理员专用）
	change_article_permission_by_admin	/article/manager/update/<int:article_id>/permission	修改文章权限（管理员专用）
	delete_article_physically	/article/manager/del/<int:article_id	物理删除文章（管理员专用）

		>	
	soft_delete_article	/article/manager/de2/<int:article_id>	软删除文章（管理员专用）
	create_article	/article/create	创建文章（用户）
	get_articles	/article/list	获取特定用户文章列表（管理员和用户均可）
	update_article_by_user	/article/update/<int:article_id>	更新文章（用户）
	delete_article_by_user	/article/delete/<int:article_id>	删除文章（用户）
	like_article	/articles/<int:article_id>/like	点赞文章（用户）
	unlike_article	/articles/<int:article_id>/unlike	取消点赞（用户）
管理评论	create_comment	/comment/<int:article_id>/create	用户创建评论，已经实现父评论功能
	update_comment	/comment/update/<int:comment_id>	用户更新自己的评论
	delete_comment	/comment/delete/<int:comment_id>	用户删除自己的评论
	report_comment	/comment/report/<int:comment_id>	用户举报评论
	get_user_comments	/comment/listall	用户获取自己发布的所有评论
	get_user_reported_comments	/comment/reported_comments	用户获取自己所有被举报的评论
管理收藏	favorite_artic	/article/favorite/	收藏

	le	<int:article_id>	
	unfavorite_article	/article/unfavorite/<int:article_id>	取消收藏
	get_user_favorites	/user/favorites/<int:user_id>	获取某个用户的所有收藏
	get_article_favorites_count	/article/favorites/count/<int:article_id>	获取某个文章的收藏数
管理文章点赞	like_article	/alike/like/<int:article_id>	点赞
	unlike_article	/alike/unlike/<int:article_id>	取消点赞
	get_like_count	/alike/count/<int:article_id>	获取文章的点赞数
	get_likers	/alike/list/<int:article_id>	获取文章的点赞用户
	get_user_likes	/alike/user/records	获取用户的所有点赞记录
管理评论点赞	like_comment	/comment/like/<int:comment_id>	用户评论点赞
	unlike_comment	/comment/unlike/<int:comment_id>	用户取消点赞
	get_comment_likes_count	/comment/likes/count/<int:comment_id>	用户获得某个评论的所有点赞数
	get_comment_likes_users	/comment/likes/users/<int:comment_id>	用户获得某个评论所有点赞的用户

（2）数据库接口

使用 SQLAlchemy 或原生 SQL 操作以下核心表：

用户表（user）：存储用户名、密码哈希、注册时间等；

文章表（article）：标题、正文、作者、标签、创建时间等；

评论表（comment）：内容、所属文章、评论时间等；

文章收藏表（ArticleFavorite）：用于文章分类。

接口包括：

User.query.filter_by(username='...');

article.query.order_by(Post.created_at.desc()).limit(10);

db.session.add(comment);

User.query.filter;

db.session.commit() 等。

4.5.3 用户接口

用户通过浏览器与前端页面交互，输入信息由表单提交至后端：

注册页面：表单输入用户名、密码、邮箱等

写博客页面：输入标题、正文、标签，点击“发布”即触发/article/create 接口

评论区：填写评论后点击提交按钮，前端调用/comment/<int:article_id>/create 响应信息通过弹窗或状态提示组件反馈，包括成功提示或错误信息。

4.5.4 接口约束与协议

（1）通信协议

为保证个人博客系统各模块之间通信的统一性、安全性和可维护性，系统接口在通信协议、认证方式和错误处理方面采用了标准化设计，具体如下：

系统所有 API 接口均基于标准的 **HTTP/HTTPS 协议**，推荐部署环境中启用 HTTPS 以保证数据传输的加密性和安全性。

① 编码方式：统一采用 UTF-8 编码，支持多语言文本处理，避免字符编码问题；

② 数据格式：所有请求与响应数据默认采用 JSON 格式，轻量且便于前后端交互；

③ 请求方式：API 接口遵循 RESTful 风格，使用不同的 HTTP 方法来表示不同的操作语义：GET：查询资源、POST：创建资源、PUT：更新资源、DELETE：删除资源

（2）认证机制

① 为保护用户隐私与系统数据安全，系统采用基于 JWT 的身份认证机制。

② 登录注册接口开放：用户首次注册（/api/register）和登录（/api/login）时无需认证；

③ Token 获取与使用：

登录成功后，后端生成签名后的 JWT 并返回给前端； 前端需在后续请求的 HTTP Header 中加入如下字段用于身份验证：

Authorization: Bearer <token>

（3）错误处理规范

状态码	含义	触发场景
200	OK	请求成功，返回正常数据
201	Created	成功创建新资源
400	Bad Request	参数错误、缺失必要字段等
401	Unauthorized	用户未登录或 Token 无效/过期
403	Forbidden	权限不足
404	Not Found	请求资源不存在
500	Internal Server Error	后端代码异常、数据库错误等

（4）接口变更管理策略

① 所有接口版本通过 URL 进行区分，例如 /api/v1/posts

② 新接口向后兼容旧版接口，必要时保留旧接口路径

③ 接口更新需同步更新接口文档，并通知前端开发人员进行适配

5 CSCI 详细设计

5.1 用户管理模块

5.1.1 概述

该模块负责管理平台上的用户信息，包括用户的注册、登录、登出、信息修改、权限管理、身份验证等功能。用户管理模块是系统访问控制和个性化服务的基础，确保用户身份的合法性与数据交互的安全性，同时支持后台对用户行为的监管与分析。

5.1.2 软件配置项设计中的约束、限制或非常规特征

用户身份验证与权限控制：需要确保所有用户操作在认证的前提下进行，并根据用户角色（如普通用户、管理员）分配不同的访问权限。

密码加密与安全存储：用户密码采用安全哈希算法进行加密存储，防止信息泄露。

登录状态管理：通过 JWT 实现无状态的登录认证机制，支持登录过期时间控制与刷新机制。

用户唯一性校验：用户名、邮箱等信息必须具备唯一性，防止重复注册。
数据隐私保护：对用户个人信息（如邮箱、手机号等）严格管控，仅授权范围内可见。

5.1.3 编程语言和使用理由

用户管理模块采用 Python 语言进行开发，原因如下：

开发效率高：Python 提供简洁易读的语法，有助于快速构建与调试用户认证相关逻辑；

安全机制成熟：结合 Flask-JWT-Extended、Werkzeug 等库可实现成熟的身份认证与安全管理；

集成方便：与其他模块（文章、评论、推荐等）易于集成并共享用户信息；

社区资源丰富：在用户认证、权限控制、密码加密等方面有大量开源工具和文档支持。

5.1.4 过程式命令列表

register_user：注册新用户，写入数据库并返回初始用户信息。

login_user：用户登录，校验身份并生成 JWT Token。

logout_user：用户登出，触发后台 Token 黑名单。

get_user_profile：获取指定用户的详细资料。

update_user_profile：修改用户基本信息（如昵称、头像、简介等）。

change_user_password：用户修改密码，需验证旧密码。

5.1.5 逻辑说明

（1）该软件配置项执行启动时，其内部起作用的条件：

系统接收到用户相关的 API 请求（如注册、登录、获取资料）；

请求中附带合法的 JWT Token（除注册与登录外）；

后台已初始化用户数据库表结构与索引。

（2）把控制交给其他软件配置项的条件：

用户登录成功后，Token 交由前端保存，用于后续模块（如文章、评论）鉴权；

用户更新信息后，通知缓存模块刷新数据；

（3）对每个输入的响应及响应时间，包括数据转换、重命名和数据传送操作：

输入包括用户名、密码、邮箱等，需进行格式验证、唯一性校验与加密处理；

数据写入与查询涉及数据库操作，响应时间一般在 1 秒内；

响应结果为标准 JSON，附带状态码与信息字段。

（4）该软件配置项运行期间的操作序列和动态控制序列，包括：

① 序列控制方法：

判断请求类型（注册、登录、更新等）并调用对应逻辑；

输入校验 → 权限检查 → 数据处理（如加密、查询）→ 数据库交互 → 返回结果。

② 逻辑与输入条件控制

表单字段必须完整且符合格式要求（如邮箱正则表达式）；

登录失败超过指定次数可触发账户暂时锁定机制；

用户角色优先级按管理员 > 普通用户排序控制操作权限。

③ 数据在内存中的进出

请求体数据解析为用户数据对象，存入内存中；

查询用户资料时将缓存命中优先级置前，缓存未命中则从数据库读取；

Token、Session 等认证信息可存于内存缓存中。

④ 离散输入信号与中断操作时序

模块持续监听注册/登录/登出/修改资料等用户交互信号；

用户在操作过程中可随时取消提交或退出流程；

若操作失败（如登录失败、Token 过期），需立即中断并提示前端。

5.2 文章管理模块

5.2.1 概述

该模块负责管理平台上的文章信息，包括文章的创建、更新、删除、搜索、推荐、热点等功能。它确保文章的管理操作准确无误，并能够根据用户需求提供个性化的文章推荐和搜索结果。

5.2.2 软件配置项设计中的约束、限制或非常规特征

① 文章权限与状态管理：需要灵活设计文章的权限和状态管理，以适应不同的文章可见性和管理需求，例如公开、私密、已删除、被举报等状态。

② 搜索功能的高效性：需要考虑多种搜索条件，如标题、内容、作者、标签等，并确保搜索结果的准确性和响应速度。

③ 推荐算法的个性化：文章推荐需要根据用户的浏览历史、兴趣偏好、点击量等进行个性化推荐，同时要确保推荐结果的多样性和相关性。

④ 数据安全与隐私保护：文章管理模块需要遵循相关的法律法规，确保用户数据的安全和隐私。

5.2.3 编程语言和使用理由

文章管理模块采用 Python 语言进行开发，原因如下：

① 良好的可读性和可维护性：Python 代码简洁，易于理解和修改，适合快速开发和迭代。

② 丰富的第三方库和框架：Python 提供了丰富的 Web 开发框架（如 Flask）、数据库操作库（如 SQLAlchemy）和自然语言处理库（如 NLTK、jieba），可快速构建高效的文章管理系统。

③ 支持多种应用场景：Python 适用于 Web 开发、数据处理和自然语言处理等多种场景，能够满足文章管理模块的多样化需求。

④ 强大的社区支持：Python 拥有庞大的开发者社区，遇到问题时可以快速获取解决方案。

5.2.4 过程式命令列表

create_article：该命令用于创建新文章。该命令调用文章创建模块和数据库写入模块来完成该任务。

update_article：该命令用于更新已有的文章。该命令调用文章更新模块和数据库更新模块来完成该任务。

delete_article：该命令用于删除文章（物理删除或软删除）。该命令调用文章删除模块和数据库更新模块来完成该任务。

search_articles：该命令用于搜索文章。该命令调用文章搜索模块和数据库查询模块来完成该任务。

recommend_articles：该命令用于推荐文章。该命令调用文章推荐模块和数据库查询模块来完成该任务。

get_article_detail：该命令用于获取特定文章的详细信息。该命令调用文章详情查询模块和数据库查询模块来完成该任务。

get_articles_by_user：该命令用于获取特定用户的文章列表。该命令调用用户文章查询模块和数据库查询模块来完成该任务。

5.2.5 逻辑说明

（1）该软件配置项执行启动时，其内部起作用的条件：

系统接收到文章创建、更新、删除、搜索、推荐或获取文章详情的请求。

系统已经通过了用户身份验证和访问权限检查。

（2）把控制交给其他软件配置项的条件：

当文章创建或更新完成后，将结果传递给文章列表显示模块。

当文章删除完成后，将删除记录传递给文章记录模块。

当搜索或推荐完成后，将结果传递给前端展示模块。

（3）对每个输入的响应及响应时间，包括数据转换、重命名和数据传送操作：

输入为文章相关的数据，需要进行数据验证和格式转换以确保数据质量。

响应时间取决于数据库操作的速度和网络延迟，通常在数秒内完成。

（4）该软件配置项运行期间的操作序列和动态控制序列，包括：

① 序列控制方法：

根据用户请求类型（文章创建、更新、删除、搜索、推荐等）执行相应的操作流程。

对输入数据进行验证，确保数据的完整性和准确性。

执行数据库操作，如插入、更新、删除或查询记录。

返回操作结果给用户界面。

② 该方法的逻辑与输入条件，如计时偏差、优先级赋值：

数据验证和格式转换是实时进行的，以确保数据的一致性。

数据库操作根据操作类型和系统负载动态调整优先级，以优化性能。

③ 数据在内存中的进出：

输入的文章数据首先存储在内存中的数据结构中，如对象或数据模型。

处理后的数据通过数据库接口进出内存，与数据库进行交互。

④ 离散输入信号的感知，以及在软件配置项内中断操作之间的时序关系：

软件配置项内部监听用户操作和系统事件，如用户提交表单或定时任务触发。

中断操作可能包括用户取消操作或系统错误，需要适当处理这些情况以保持系统稳定。

5.3 文章管理员管理模块

5.3.1 概述

该模块负责管理文章平台的管理员信息，包括管理员的登录、注册、信息管理、权限分配等功能。它确保管理员的操作符合系统要求，并能够高效地管理平台的日常运营。通过该模块，管理员可以对用户、文章、评论等进行管理和监控。

5.3.2 软件配置项设计中的约束、限制或非常规特征

管理员权限的分层设计：需要进行灵活的权限设计，以适应不同管理员的职责和权限范围，例如超级管理员和普通管理员。

管理员操作的安全性：需要确保管理员登录、权限修改等操作的安全性，防止未授权访问和数据泄露。

数据完整性和一致性：管理员模块需要确保数据的完整性和一致性，尤其是在管理员信息更新和权限修改时。

遵循法律法规：管理员模块需要遵循相关的法律法规，确保用户数据的安全和隐私保护。

5.3.3 编程语言和使用理由

文章管理员管理模块采用 Python 语言进行开发，原因如下：

良好的可读性和可维护性：Python 代码简洁，易于理解和修改，适合快速开发和迭代。

丰富的第三方库和框架：Python 提供了丰富的 Web 开发框架（如 Flask）、数据库操作库（如 SQLAlchemy）以及用户认证库（如 Flask-JWT-Extended），可快速构建高效的安全管理功能。

支持多种应用场景：Python 适用于 Web 开发、数据处理和安全控制等多种场景，能够满足管理员模块的多样化需求。

强大的社区支持：Python 拥有庞大的开发者社区，遇到问题时可以快速获取解决方案。

5.3.4 过程式命令列表

admin_login：该命令用于管理员登录。该命令调用管理员登录界面和数据库验证模块来完成该任务。

add_manager：该命令用于添加新的管理员。该命令调用管理员注册模块和数据库写入模块来完成该任务。

delete_manager：该命令用于删除管理员。该命令调用管理员删除模块和数据库更新模块来完成该任务。

update_manager_profile：该命令用于更新管理员的个人信息。该命令调用管理员信息更新模块和数据库更新模块来完成该任务。

list_managers：该命令用于列出所有管理员信息。该命令调用管理员列表查询模块和数据库查询模块来完成该任务。

list_users：该命令用于列出所有用户信息。该命令调用用户列表查询模块和数据库查询模块来完成该任务。

update_user_status：该命令用于更新用户的账户状态。该命令调用用户状态更新模块和数据库更新模块来完成该任务。

update_user_permission：该命令用于更新用户的权限。该命令调用用户权限更新模块和数据库更新模块来完成该任务。

5.3.5 逻辑说明

（1）该软件配置项执行启动时，其内部起作用的条件：

系统接收到管理员登录、注册、信息更新或权限修改的请求；

系统已经通过了用户身份验证和访问权限检查。

（2）把控制交给其他软件配置项的条件：

当管理员登录成功后，将生成的令牌传递给用户界面；

当管理员信息更新完成后，将更新结果传递给管理员信息展示模块；

当用户状态或权限更新完成后，将更新结果传递给用户管理模块。

（3）对每个输入的响应及响应时间，包括数据转换、重命名和数据传送操作：

输入为管理员或用户相关的数据，需要进行数据验证和格式转换以确保数据质量；

响应时间取决于数据库操作的速度和网络延迟，通常在数秒内完成。

（4）该软件配置项运行期间的操作序列和动态控制序列，包括：

① 序列控制方法：

根据用户请求类型（管理员登录、注册、信息更新、权限修改等）执行相应的操作流程。

对输入数据进行验证，确保数据的完整性和准确性。

执行数据库操作，如插入、更新、删除或查询记录。

返回操作结果给用户界面。

② 该方法的逻辑与输入条件，如计时偏差、优先级赋值：

数据验证和格式转换是实时进行的，以确保数据的一致性。

数据库操作根据操作类型和系统负载动态调整优先级，以优化性能。

③ 数据在内存中的进出：

输入的管理员或用户数据首先存储在内存中的数据结构中，如对象或数据模型。

处理后的数据通过数据库接口进出内存，与数据库进行交互。

④ 离散输入信号的感知，以及在软件配置项内中断操作之间的时序关系：

软件配置项内部监听用户操作和系统事件，如用户提交表单或定时任务触发。

中断操作可能包括用户取消操作或系统错误，需要适当处理这些情况以保持系统稳定。

5.4 查询与统计

5.4.1 概述

该模块提供对文章信息的查询和统计分析功能，以便管理员和用户能够快速获取所需信息并了解文章的发布、浏览、点赞等情况。它包括多条件查询、信息汇总和报表生成等功能，能够帮助用户更好地管理和分析文章数据。

5.4.2 软件配置项设计中的约束、限制或非常规特征

高效的查询优化：需要进行高效的数据库设计和查询优化，以提高查询速度和响应时间，尤其是在处理大量文章数据时。

灵活的统计方法：需要实现灵活的统计方法，支持按时间、作者、分类标签、阅读量、点赞数等多种维度进行统计。

数据安全与隐私保护：查询与统计模块需要考虑数据安全和隐私保护，确保只有授权用户能够访问敏感信息，例如私密文章的统计数据。

报表生成与导出：支持将统计结果生成报表，并提供多种格式（如 CSV、Excel、PDF）的导出功能，以便用户进行进一步分析或存档。

5.4.3 编程语言和使用理由

Python 是该模块所使用的编程语言，主要是因为其强大的数据处理和数据分析库（如 Pandas、NumPy）以及丰富的可视化工具（如 Matplotlib、Seaborn），适合开发高效的查询与统计功能。此外，Python 的 Web 开发框架（如 Flask）能够方便地与前端交互，实现动态的查询和报表展示。

5.4.4 过程式命令列表

PerformArticleQuery：该命令用于根据用户输入的条件执行文章查询操作。该命令调用文章查询界面和数据库查询模块来完成该任务。

GenerateArticleReport：该命令用于生成文章统计报表。该命令调用文章统计模块和报表生成模块来完成该任务。

ExportArticleData：该命令用于将查询结果或统计报表导出为特定格式的文件。该命令调用数据导出模块来完成该任务。

5.4.5 逻辑说明

（1）该软件配置项执行启动时，其内部起作用的条件：

系统接收到文章查询或统计的请求；

系统已经通过了用户身份验证和访问权限检查。

（2）把控制交给其他软件配置项的条件：

当查询或统计操作完成后，将结果传递给用户界面或报表生成模块;

对每个输入的响应及响应时间，包括数据转换、重命名和数据传送操作；

（3）输入为查询条件或统计参数，需要进行数据验证和格式转换以确保数据质量：

响应时间取决于数据库操作的速度和网络延迟，通常在数秒内完成。

（4）该软件配置项运行期间的操作序列和动态控制序列，包括：

① 序列控制方法：

根据用户请求类型（查询、统计等）执行相应的操作流程;

对输入数据进行验证，确保数据的完整性和准确性;

执行数据库操作，如查询、汇总或生成报表;

返回操作结果给用户界面。

② 该方法的逻辑与输入条件，如计时偏差、优先级赋值：

数据验证和格式转换是实时进行的，以确保数据的一致性;

数据库操作根据操作类型和系统负载动态调整优先级，以优化性能。

③ 数据在内存中的进出：

输入的查询条件或统计参数首先存储在内存中的数据结构中，如对象或数据模型;

处理后的数据通过数据库接口进出内存，与数据库进行交互。

④ 离散输入信号的感知，以及在软件配置项内中断操作之间的时序关系：

软件配置项内部监听用户操作和系统事件，如用户提交查询请求或定时任务触发;

中断操作可能包括用户取消操作或系统错误，需要适当处理这些情况以保持系统稳定。

6 需求的可追踪性

6.1 CSCI 需求可追踪性

CSCI 模块	主要能力需求	所追踪的系统需求 / 设计目标
前端应用程序	响应式界面、用户交互（浏览、评论、点赞等）	系统功能需求：支持多平台访问，提供良好用户体验
后端应用程序	用户认证、评论管理、数据处理等	系统功能需求：处理用户请求、保障数据正确性和业务逻辑完整性
数据库管理系统	数据存储、查询、更新、一致性保障	系统功能需求：支持海量数据的持久化存储和快速访问
用户认证系统	支持多种登录注册方式、账号验证	系统安全性需求：身份验证、访问控制
内容管理系统	管理文章、评论、点赞收藏等	系统功能需求：内容创建与互动管理

6.2 CSCI 需求到它被分配给的软件配置项的可追踪性

系统需求编号	系统需求内容	所涉及的 CSCI 及其需求编号（简写）
SYS-01	系统应支持多种登录方式	用户认证系统（3.4.4）
SYS-02	支持用户注册、登录、资料管理	前端应用程序（3.4.1）、后端应用程序（3.4.2）、用户认证系统
SYS-03	用户可浏览、发表、编辑、删除文章	内容管理系统（3.4.5）、前端、后端
SYS-04	支持用户评论、点赞、收藏功能	内容管理系统、前端、后端
SYS-05	数据必须安全、可追溯、可恢复	数据库管理系统（3.4.3）
SYS-06	所有模块需通过接口协调通信	所有 CSCI（3.6 内部接口）
SYS-07	支持移动设备与浏览器访问	前端应用程序、硬件接口（3.5）
SYS-08	系统应可扩展并支持高并发	后端应用程序、数据库、服务器接口（3.5.2）

7 注解

7.1 背景信息

本系统为一个面向公众的博客网站，旨在提供一个用户可以注册登录、发布文章、评论、点赞、收藏的互动平台。系统采用前后端分离的架构设计，前端使用 Vue 框架实现响应式页面交互，后端采用 Python 的 Flask 框架处理业务逻辑，数据库使用 MySQL 存储各类数据。

7.2 术语和定义

术语	定义
CSCI	计算机软件配置项，表示系统中的独立软件单元
API	应用程序编程接口，用于前后端或系统之间的通信
RESTful	一种基于 HTTP 协议的 Web 服务设计架构风格
JSON	一种轻量级数据交换格式
ORM	对象关系映射，后端用于操作数据库的技术
Token	用户登录后的身份验证令牌，用于保持会话状态

7.3 缩略语列表

缩略语	全称	含义
API	Application Programming Interface	应用程序接口
CSCI	Computer Software Configuration Item	软件配置项
DB	Database	数据库
JSON	JavaScript Object Notation	一种数据交换格式
ORM	Object-Relational Mapping	对象关系映射
REST	Representational State Transfer	一种 Web 服务架构风格
SDK	Software Development Kit	软件开发工具包
UI	User Interface	用户界面
UX	User Experience	用户体验