1. **代码结构**
2. 文件组织：将相关功能代码模块化，每个功能模块放在独立的文件中
3. 目录结构：

* app/：应用主目录，包含Flask应用的核心代码
* templates/：HTML模板文件存放目录
* static/：静态文件（如CSS、JavaScript、图片等）存放目录

1. **命名规范**
2. **变量与函数名**：代码中使用了小写字母加下划线的方式命名变量和函数
3. **类名**：类名采用了帕斯卡命名法，这种命名方式符合PEP 8推荐的标准，即每个单词首字母大写，不使用下划线
4. **常量**：对于常量，如SQLAlchemy配置中的字符串值SQLALCHEMY\_DATABASE\_URI，使用了全部大写的字母，并用下划线分隔单词。
5. **数据库表和列名**：数据库模型类中的表名和列名通常采用小写并以单数形式表示
6. **属性和方法**：在类内部定义的方法和属性也遵循了小写加下划线的规则，此外，对于关联关系，使用了描述性的名称来指代相关联的对象
7. 代码风格
8. **缩进**：整个代码段使用了4个空格作为缩进单位，有助于提高代码可读性
9. **空行**：函数之间以及类定义之后有适当的空行，用于逻辑上的分离，使代码更加清晰易懂
10. **导入声明**：所有的导入语句都放在文件顶部，并且按照标准库、第三方库和个人模块的顺序排列，同时避免了不必要的导入

from flask import Flask, render\_template, request, redirect, url\_for, flash, session,jsonify

from flask\_sqlalchemy import SQLAlchemy

import secrets

from flask\_cors import CORS

import numpy as np

from werkzeug.security import generate\_password\_hash

from werkzeug.security import check\_password\_hash

from datetime import datetime

import uuid

import os

from sqlalchemy import or\_, text,func

from sqlalchemy.orm import joinedload

from functools import wraps

from flask import request, redirect, url\_for

from werkzeug.utils import secure\_filename

from urllib.parse import quote\_plus

1. **注释**：函数、类和复杂逻辑应添加必要的注释，说明其功能、参数和返回值

# 发送消息功能

@app.route('/send\_message', methods=['POST'])

def send\_message():

1. 错误处理
2. **异常处理**：使用try...except结构捕获并处理可能的异常，避免程序因未处理的异常而崩溃
3. **错误返回**：使用JSON格式返回错误信息。例如：

        if user and check\_password\_hash(user.password, password):  # 验证密码

            session['user\_id'] = user.id  # 将用户 ID 存储在会话中

            flash('登录成功！', 'success')

            return redirect(url\_for('main\_page'))  # 登录成功后重定向到主页

        else:

            flash('手机号或密码错误！', 'danger')

1. 数据库操作
2. **数据库连接和配置**：代码的开始部分，配置了Flask应用与SQLAlchemy的连接，以及数据库URI的配置

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql://root:123456@localhost/mydate'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db = S**QLAlchemy(app)**

1. **定义数据库模型**：定义了多个数据库模型，包括用户(User)、用户信息(UserInfo)、好友关系(Friend)、聊天记录(ChatHistory)、消息(Message)、问题(Question)、回答(Answer)和纸条盲盒(BlindBoxEntry)等。

class UserInfo(db.Model):

    \_tablename\_ = 'user\_info'

1. **关系映射**‌：通过ORM（对象关系映射）技术，定义了模型之间的关系，如User和UserInfo之间的一对一关系。

class UserInfo(db.Model):

  user = db.relationship('User', back\_populates='user\_info',lazy=True, foreign\_keys=[id])

1. **安全规范**
2. **输入验证**

* ‌**用户输入检查**‌：在处理用户输入（如注册、登录、发送消息等）时，代码会检查必要的参数是否存在且有效。例如，在添加好友或发送消息的功能中，会检查接收者ID和消息内容是否为空。
* ‌**防止SQL注入**‌：由于使用了ORM（对象关系映射）如Flask-SQLAlchemy，SQL语句的生成由框架自动处理，从而减少了SQL注入的风险。此外，代码中未直接出现拼接SQL语句的情况，这也是防止SQL注入的一种做法。

1. 测试规范
2. 登录：测试用户登录功能，验证输入错误的手机号或密码时的提示
3. 发帖：测试发帖功能，验证帖子是否正确显示在首页
4. 评论：测试评论功能，验证评论是否正确显示在帖子详情页
5. 筛选：测试筛选功能，验证筛选是否正确显示对应条件的用户
6. 盲盒交友：测试加入盲盒和抽取盲盒功能，验证信息的正确性