这是整个项目文件的整体树形结构：

root:[.]

+--.idea

| +--agent v3.iml

| +--inspectionProfiles

| | +--profiles\_settings.xml

| +--misc.xml

| +--modules.xml

| +--workspace.xml

+--backend

| +--.env

| +--.idea

| | +--.name

| | +--inspectionProfiles

| | | +--profiles\_settings.xml

| | +--misc.xml

| | +--modules.xml

| | +--my-project.iml

| | +--workspace.xml

| +--agent

| | +--agent\_chain.py

| | +--grading\_agent.py

| | +--questionnaire\_agent.py

| | +--\_\_pycache\_\_

| | | +--agent\_chain.cpython-310.pyc

| | | +--grading\_agent.cpython-310.pyc

| | | +--questionnaire\_agent.cpython-310.pyc

| +--app.log

| +--app.py

| +--frontend

| | +--index.html

| +--requirements.txt

| +--\_\_pycache\_\_

| | +--questionnaire.cpython-310.pyc

+--README.md

+--red\_corpus.txt

+--test.txt

+--输出项目树形结构.py

这是所有的py文件内容：

这是agent\_chain.py:

# backend/agent/agent\_chain.py  
  
from agent.questionnaire\_agent import QuestionnaireAgent  
from agent.grading\_agent import GradingAgent  
  
class AgentChain:  
 def \_\_init\_\_(self, q\_agent: QuestionnaireAgent, g\_agent: GradingAgent):  
 self.q\_agent = q\_agent  
 self.g\_agent = g\_agent  
  
 def process\_long\_text(self, long\_text: str, num\_questions\_per\_chunk: int, question\_type: str = "普通问答题"):  
 *"""  
 用问卷Agent生成题目  
 """* return self.q\_agent.generate\_questionnaire(  
 long\_text,  
 num\_questions\_per\_chunk=num\_questions\_per\_chunk,  
 question\_type=question\_type  
 )  
  
 def grade\_answers(self, questions: list, answers: list, question\_type: str):  
 *"""  
 依次调用GradingAgent对每道题进行打分  
 """* feedback = []  
 total\_score = 0  
 for question, answer in zip(questions, answers):  
 result = self.g\_agent.grade\_question(question, answer, question\_type)  
 score = result['score']  
 total\_score += score  
 feedback.append({  
 'question': question,  
 'answer': answer,  
 'score': score,  
 'feedback': result['feedback']  
 })  
 return feedback, total\_score

这是grading\_agent.py:

import openai  
import json  
import re  
import logging  
  
# 设置日志  
logger = logging.getLogger()  
  
class GradingAgent:  
 def \_\_init\_\_(self, api\_key: str, base\_url: str = None, model: str = "gpt-3.5-turbo"):  
 *"""  
 使用官方 openai 库进行初始化。  
 """* openai.api\_key = api\_key  
 if base\_url:  
 openai.api\_base = base\_url  
 self.model = model  
  
 def grade\_question(self, question: str, answer: str, question\_type: str) -> dict:  
 *"""  
 输入题目和答案，根据题目类型返回 {'score': int, 'feedback': str}  
 """* try:  
 # 对于普通问答题  
 if question\_type == "普通问答题":  
 prompt = (  
 f"题目：{question}\n"  
 f"答案：{answer}\n\n"  
 "请根据以下评分标准为此题打分，满分10分：\n"  
 "1. 内容准确性（4 分）：\n"  
 " - 4 分：答案内容完全准确，涵盖了所有关键点。\n"  
 " - 3 分：答案内容大部分准确，但有少数遗漏。\n"  
 " - 2 分：答案存在较大误差，缺少关键点。\n"  
 " - 1 分：答案内容严重偏离正确答案。\n"  
 "2. 逻辑清晰度（3 分）：\n"  
 " - 3 分：答案条理清晰，逻辑严谨，易于理解。\n"  
 " - 2 分：答案结构较清晰，逻辑大致合适，但部分地方表达不清。\n"  
 " - 1 分：答案缺乏清晰的结构，逻辑混乱。\n"  
 "3. 语言表达（2 分）：\n"  
 " - 2 分：语言流畅，表达清晰，几乎没有语法错误。\n"  
 " - 1 分：语言基本流畅，存在一些语法错误，但不影响理解。\n"  
 " - 0 分：语言表达较差，语法错误多，影响理解。\n"  
 "4. 完整性（1 分）：\n"  
 " - 1 分：答案涵盖了所有要求的要点，完整回答问题。\n"  
 " - 0 分：答案不完整，缺少重要信息。\n\n"  
 "请根据以上标准对这个答案进行评分，并给出详细的反馈。务必按以下格式返回：\n"  
 "score: X分\n"  
 "feedback: <这里写出详细评价，基于上述评分标准给出反馈>\n"  
 "不要输出其他内容。\n"  
 )  
 else:  
 # 对于选择题和判断题，使用简单评分和反馈  
 prompt = (  
 f"题目：{question}\n"  
 f"答案：{answer}\n\n"  
 "请为此题进行评分（满分10分）并给出简短的反馈。\n"  
 "只能给出10分或0分两种分数\n"  
 "返回格式：\n"  
 "score: X分\n"  
 "feedback: <简短的反馈说明>\n"  
 "不要输出其他内容。\n"  
 )  
  
 completion = openai.ChatCompletion.create(  
 model=self.model,  
 messages=[  
 {'role': 'system', 'content': 'You are an expert in grading essay-type questions.'},  
 {'role': 'user', 'content': prompt}  
 ]  
 )  
 content = completion.choices[0].message.content.strip()  
  
 score\_match = re.search(r"score:\s\*(\d+)\s\*分", content)  
 feedback\_match = re.search(r"feedback:\s\*(.\*)", content, re.DOTALL)  
  
 score = 0  
 feedback = ""  
 if score\_match:  
 score = int(score\_match.group(1))  
  
 if feedback\_match:  
 feedback = feedback\_match.group(1).strip()  
  
 logger.info(f"评分：{score} 分，反馈：{feedback}")  
 return {'score': score, 'feedback': feedback}  
  
 except Exception as e:  
 logger.exception("评分时发生错误")  
 return {'score': 0, 'feedback': '评分出错'}

这是questionnaire\_agent.py:

# backend/agent/questionnaire\_agent.py  
  
import openai  
import re  
import json  
import tiktoken  
  
class QuestionnaireAgent:  
 def \_\_init\_\_(self, api\_key: str, base\_url: str = None, model: str = "qwen-plus"):  
 *"""  
 使用官方 openai 库进行初始化。  
 如果需要自定义 base\_url(例如兼容其他服务端), 可以赋值给 openai.api\_base。  
 """* openai.api\_key = api\_key  
 if base\_url:  
 openai.api\_base = base\_url  
 self.model = model  
  
 def split\_text\_by\_tokens(self, text: str, max\_tokens=2000) -> list:  
 *"""  
 使用 tiktoken 对文本进行 token 化，根据 max\_tokens 切分成多个 chunk。  
 每个 chunk decode 回原文本，以保证能够投喂到模型里。  
 """* # 根据self.model获取对应的tokenizer, 若不支持qwen-plus，可换成"gpt-3.5-turbo"的encoding  
 # 或者使用 tiktoken.get\_encoding("cl100k\_base")  
 try:  
 encoding = tiktoken.encoding\_for\_model(self.model)  
 except:  
 # 若tiktoken不支持该模型，可以fallback到一个通用编码  
 encoding = tiktoken.get\_encoding("cl100k\_base")  
  
 tokens = encoding.encode(text)  
 chunks = []  
 start = 0  
 while start < len(tokens):  
 end = start + max\_tokens  
 chunk\_tokens = tokens[start:end]  
 chunk\_text = encoding.decode(chunk\_tokens)  
 chunks.append(chunk\_text)  
 start = end  
 return chunks  
  
  
 def generate\_questions\_from\_chunk(self, chunk: str, num\_questions=5, question\_type="普通问答题") -> str:  
 *"""  
 给定文本 chunk，生成若干问题。  
 question\_type: 前端传入的"普通问答题"、"选择题"、"判断题"等  
 """* try:  
 prompt = (  
 f"根据以下内容生成{num\_questions}个'{question\_type}'问题：\n\n{chunk}\n\n"  
 f"要求：\n"  
 f"1. 不要输出多余的话，也不要给出答案。\n"  
 f"2. 如果是选择题，把选项和题目写在同一行内；如果是判断题，注意只生成判断内容。\n"  
 f"3. 输出时每个问题独立占一行。\n"  
 f"4. 只出与核心思想和关键事件相关的问题，不要出无关数据或生僻数字作为考题，除非这个数字确实是非常重要的历史节点。\n"  
 )  
  
 completion = openai.ChatCompletion.create(  
 model=self.model,  
 messages=[  
 {'role': 'system', 'content': 'You are an expert in the history of the Communist Party of China.'},  
 {'role': 'user', 'content': prompt}  
 ],  
 )  
 # 官方库的结构  
 content = completion.choices[0].message.content  
  
 # 去除\*\*符号  
 content = re.sub(r'\\*\\*', '', content)  
  
 return content.strip()  
 except Exception as e:  
 print(f"Error generating questions for chunk: {e}")  
 return ""  
  
 def generate\_questionnaire(self, long\_text: str, num\_questions\_per\_chunk=5, question\_type="普通问答题") -> list:  
 *"""  
 主函数：先用token计数切分文本，再对每个chunk调用 generate\_questions\_from\_chunk。  
 question\_type 由前端传入，默认为"普通问答题"。  
 """* chunks = self.split\_text\_by\_tokens(long\_text, max\_tokens=2000)  
 all\_questions = []  
  
 for idx, chunk in enumerate(chunks):  
 print(f"Processing chunk {idx + 1}/{len(chunks)}")  
 questions = self.generate\_questions\_from\_chunk(chunk,  
 num\_questions=num\_questions\_per\_chunk,  
 question\_type=question\_type)  
 if questions:  
 # 按行拆分  
 question\_lines = re.split(r'\n+', questions)  
 for line in question\_lines:  
 clean\_line = re.sub(r'^\d+\.?\s\*', '', line).strip()  
 if clean\_line:  
 all\_questions.append(clean\_line)  
  
 # 去重  
 unique\_questions = list(dict.fromkeys(all\_questions))  
 return unique\_questions

这是index.html:

<!-- frontend/index.html -->  
<!DOCTYPE html>  
<html lang="zh-CN">  
<head>  
 <meta charset="UTF-8">  
 <title>问卷生成器</title>  
 <!-- 引入 Bootstrap CSS -->  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">  
 <style>  
 body {  
 background-color: #f8f9fa;  
 padding-top: 50px;  
 }  
 .container {  
 max-width: 800px;  
 }  
 .questions, .history {  
 margin-top: 30px;  
 }  
 .questionnaire {  
 background-color: #ffffff;  
 padding: 15px;  
 border-radius: 5px;  
 margin-bottom: 10px;  
 box-shadow: 0 2px 5px rgba(0,0,0,0.1);  
 }  
 .loading-spinner {  
 display: none;  
 margin-top: 20px;  
 text-align: center;  
 }  
 </style>  
</head>  
<body>  
<div class="container">  
 <h1 class="mb-4 text-center">AI 问卷生成器</h1>  
 <form id="questionnaire-form">  
 <div class="mb-3">  
 <label for="long\_text" class="form-label">请输入长文本内容或上传文本文件：</label>  
 <textarea class="form-control" id="long\_text" name="long\_text" style="display: none;" placeholder="直接输入文本内容..."></textarea>  
 <div class="input-group">  
 <input class="form-control" type="file" id="file\_input" accept=".txt">  
 <button type="button" class="btn btn-secondary" id="toggle\_text\_input">直接输入文本</button>  
 </div>  
 </div>  
  
 <!-- 新增：题型选择 -->  
 <div class="mb-3">  
 <label for="question\_type" class="form-label">题型选择：</label>  
 <select class="form-select" id="question\_type" name="question\_type">  
 <option value="普通问答题">普通问答题</option>  
 <option value="选择题">选择题</option>  
 <option value="判断题">判断题</option>  
 </select>  
 </div>  
 <!-- 新增结束 -->  
  
 <div class="mb-3">  
 <label for="num\_questions" class="form-label">每2000字生成问题数量：</label>  
 <input type="number" class="form-control" id="num\_questions" name="num\_questions" value="5" min="1" required>  
 </div>  
 <button type="submit" class="btn btn-primary w-100">生成问卷</button>  
 </form>  
  
 <div class="loading-spinner" id="loading-spinner">  
 <div class="spinner-border text-primary" role="status">  
 <span class="visually-hidden">生成中...</span>  
 </div>  
 <p>生成中，请稍候...</p>  
 </div>  
  
 <div class="questions" id="questions"></div>  
 <div class="history" id="history"></div>  
</div>  
  
<!-- 引入 Bootstrap JS 和依赖 -->  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>  
<script>  
 // 切换显示文本输入框  
 document.getElementById('toggle\_text\_input').addEventListener('click', function() {  
 const textArea = document.getElementById('long\_text');  
 const fileInput = document.getElementById('file\_input');  
 if (textArea.style.display === 'none') {  
 textArea.style.display = 'block';  
 fileInput.style.display = 'none';  
 this.innerText = '上传文本文件';  
 } else {  
 textArea.style.display = 'none';  
 fileInput.style.display = 'block';  
 this.innerText = '直接输入文本';  
 }  
 });  
  
 // 处理表单提交  
 document.getElementById('questionnaire-form').addEventListener('submit', function(e) {  
 e.preventDefault();  
  
 const num\_questions = parseInt(document.getElementById('num\_questions').value);  
 const fileInput = document.getElementById('file\_input');  
 let long\_text = document.getElementById('long\_text').value.trim();  
  
 const loadingSpinner = document.getElementById('loading-spinner');  
 const questionsDiv = document.getElementById('questions');  
 const historyDiv = document.getElementById('history');  
 questionsDiv.innerHTML = '';  
 historyDiv.innerHTML = '';  
 loadingSpinner.style.display = 'block';  
  
 // 读取选中的题型  
 const questionType = document.getElementById('question\_type').value;  
  
 if (fileInput.files.length > 0) {  
 const file = fileInput.files[0];  
 const reader = new FileReader();  
 reader.onload = function(event) {  
 long\_text = event.target.result.trim();  
 sendRequest(long\_text, num\_questions, questionType);  
 };  
 reader.onerror = function() {  
 loadingSpinner.style.display = 'none';  
 alert('读取文件时出错，请重试。');  
 };  
 reader.readAsText(file, 'UTF-8');  
 } else if (long\_text) {  
 sendRequest(long\_text, num\_questions, questionType);  
 } else {  
 loadingSpinner.style.display = 'none';  
 alert('请输入文本内容或上传文本文件。');  
 }  
 });  
  
 function sendRequest(long\_text, num\_questions, questionType) {  
 fetch('/api/generate\_questionnaire', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 long\_text: long\_text,  
 num\_questions\_per\_chunk: num\_questions,  
 question\_type: questionType  
 }),  
 })  
 .then(response => response.json())  
 .then(data => {  
 const loadingSpinner = document.getElementById('loading-spinner');  
 loadingSpinner.style.display = 'none';  
 if (data.error) {  
 alert(data.error);  
 return;  
 }  
 const questionsDiv = document.getElementById('questions');  
 questionsDiv.innerHTML = '<h2>生成的问卷问题如下：</h2>';  
 const answers = [];  
 data.questions.forEach((q, index) => {  
 const div = document.createElement('div');  
 div.className = 'questionnaire';  
 div.innerHTML = `<strong>${index + 1}.</strong> ${q}<br>  
 <textarea class="form-control answer" rows="3" placeholder="请在这里输入您的答案"></textarea>`;  
 questionsDiv.appendChild(div);  
 answers.push(null); // 初始化答案数组  
 });  
  
 // 提交按钮  
 const submitButton = document.createElement('button');  
 submitButton.textContent = '提交答案';  
 submitButton.className = 'btn btn-primary w-100 mt-3';  
 submitButton.onclick = function() {  
 const answerTexts = document.querySelectorAll('.answer');  
 answerTexts.forEach((textarea, i) => {  
 answers[i] = textarea.value.trim();  
 });  
 submitAnswers(data.questions, answers);  
 };  
 questionsDiv.appendChild(submitButton);  
 })  
 .catch((error) => {  
 console.error('Error:', error);  
 const loadingSpinner = document.getElementById('loading-spinner');  
 loadingSpinner.style.display = 'none';  
 alert('生成问卷时出错，请稍后再试。');  
 });  
 }  
  
 function submitAnswers(questions, answers) {  
 fetch('/api/submit\_answers', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({  
 questions: questions,  
 answers: answers,  
 }),  
 })  
 .then(response => response.json())  
 .then(data => {  
 if (data.feedback) {  
 const feedbackDiv = document.createElement('div');  
 feedbackDiv.innerHTML = '<h2>批改结果</h2>';  
 data.feedback.forEach(item => {  
 const feedbackItem = document.createElement('div');  
 feedbackItem.className = 'questionnaire';  
 feedbackItem.innerHTML = `<strong>问题：${item.question}</strong><br>  
 答案：${item.answer}<br>  
 得分：${item.score}分<br>  
 反馈：${item.feedback}`;  
 feedbackDiv.appendChild(feedbackItem);  
 });  
 const totalScoreDiv = document.createElement('div');  
 totalScoreDiv.innerHTML = `<h3>总分：${data.total\_score}分</h3>`;  
 feedbackDiv.appendChild(totalScoreDiv);  
 document.getElementById('history').innerHTML = '';  
 document.getElementById('history').appendChild(feedbackDiv);  
 }  
 })  
 .catch((error) => {  
 console.error('Error:', error);  
 alert('提交答案时出错，请稍后再试。');  
 });  
 }  
</script>  
</body>  
</html>

这是app.py:

import os  
import json  
import logging  
from flask import Flask, request, jsonify, send\_from\_directory  
from dotenv import load\_dotenv  
  
# 引入Agent与AgentChain  
from agent.questionnaire\_agent import QuestionnaireAgent  
from agent.grading\_agent import GradingAgent  
from agent.agent\_chain import AgentChain  
  
# 设置日志记录  
logging.basicConfig(level=logging.DEBUG,  
 format='%(asctime)s - %(levelname)s - %(message)s',  
 handlers=[  
 logging.StreamHandler(), # 输出到控制台  
 logging.FileHandler("app.log", mode='w') # 输出到文件  
 ])  
logger = logging.getLogger()  
  
load\_dotenv() # 加载环境变量  
  
app = Flask(\_\_name\_\_, static\_folder='frontend')  
  
# 读取 KEY 和 BASE\_URL  
API\_KEY = os.getenv("OPENAI\_API\_KEY")  
BASE\_URL = os.getenv("DASHSCOPE\_BASE\_URL") # 不设置则默认为官方地址  
  
# 初始化Agent  
q\_agent = QuestionnaireAgent(api\_key=API\_KEY, base\_url=BASE\_URL, model="qwen-plus")  
g\_agent = GradingAgent(api\_key=API\_KEY, base\_url=BASE\_URL, model="qwen-plus")  
  
# 组合到 AgentChain  
chain = AgentChain(q\_agent, g\_agent)  
  
# 用于存储question\_type的全局字典  
question\_types\_storage = {}  
  
@app.route('/')  
def home():  
 return send\_from\_directory(app.static\_folder, 'index.html')  
  
@app.route('/favicon.ico')  
def favicon():  
 return '', 204  
  
@app.route('/api/generate\_questionnaire', methods=['POST'])  
def api\_generate\_questionnaire():  
 *"""  
 生成完整的问卷。  
 """* try:  
 data = request.get\_json()  
 long\_text = data.get('long\_text', '')  
 num\_questions\_per\_chunk = data.get('num\_questions\_per\_chunk', 5)  
 question\_type = data.get('question\_type') # 从前端接收题型  
  
 if not long\_text:  
 logger.error('缺少 long\_text 参数')  
 return jsonify({'error': '缺少 long\_text 参数'}), 400  
  
 # 存储question\_type，以便后续使用  
 question\_types\_storage['question\_type'] = question\_type  
  
 # 调用 AgentChain 来生成问卷  
 questions = chain.process\_long\_text(long\_text, num\_questions\_per\_chunk, question\_type)  
 logger.info(f"成功生成问卷：{len(questions)}个问题")  
 return jsonify({'questions': questions})  
 except Exception as e:  
 logger.exception("生成问卷时发生错误")  
 return jsonify({'error': '生成问卷时发生错误'}), 500  
  
@app.route('/api/submit\_answers', methods=['POST'])  
def submit\_answers():  
 *"""  
 处理用户提交的问卷答案。  
 """* try:  
 data = request.get\_json()  
 answers = data.get('answers', [])  
 questions = data.get('questions', [])  
  
 if not answers or not questions:  
 logger.error('缺少答案数据或题目数据')  
 return jsonify({'error': '缺少答案数据或题目数据'}), 400  
  
 # 获取之前存储的 question\_type  
 question\_type = question\_types\_storage.get('question\_type', '普通问答题')  
  
 # 调用 AgentChain 来批改答案  
 feedback, total\_score = chain.grade\_answers(questions, answers, question\_type)  
  
 logger.info(f"成功批改问卷，得分：{total\_score}")  
 return jsonify({  
 'feedback': feedback,  
 'total\_score': total\_score  
 })  
 except Exception as e:  
 logger.exception("批改答案时发生错误")  
 return jsonify({'error': '批改答案时发生错误'}), 500  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 logger.info("启动 Flask 应用")  
 app.run(debug=True, port=5000)