@

Thursday, April 24, 2025 14:00

App.get("/answer")
Def answer_question(seed_account_ids: list = Query(...) , question: str = Query(...) , mode: str = 'normal'):

接收账户ID 列表、问题字符串和模式参数

调用talk2casefile.get_answer 处理问题并生成解释
返回包含HTML格式答案的json响应

原始数据断点
@app.get("/raw_data/")
Def get_raw_data(seed_account_ids: list = Query(...) , question: str = Query(...) , mode: str = 'normal'):
与问答端点接收相同参数
调用talk2casefile.directly_get_data 获取原始数据
返回不带处理的json数据

对话端点
@app.post(""/conversation/")
Def conversation(seed_account_ids: list = Body(...), conversation_history: list=Body(...))
接收账户ID列表和对话历史
首先处理对话历史，判断是否需要回答完整问题

根据不同情况返回不同格式的响应

预定义问题端点

接收账户id列表和预定义问题列表

为每个问题提供固定的示例答案

使用batchformattingsummarytool格式化问答对

带解释的问答端点
@app.get("/answer_with_explanation")
Def answer_with_explanation(question:str, mode: str-'normal'):

接收问题和模式参数
调用talk2casefile.get_answer获取结果 解释 描述和图表
返回结构化的json响应

整体架构思路

1. 分层设计： fastAPI 处理http请求，talk2casefile 模块处理业务逻辑
2. 多种交互方式：支持直接问答 对话式交互 和批量预定义问题处理
3. 响应格式多样化：可以返回html 格式的解释 json数据 图表等
4. 性能优化：使用缓存机制避免重复计算相同问题的答案
5. 模式切换：通过mode参数支持normal和debug模式，便于调试

这个api设计适合构建一个交互式的案例分析工具，用户可以通过问答形式了解账户相关信息，系统能够提供格式化的解释和可视化数据

这个api设计适合构建一个交互式的案例分析工具，用户可以通过问答形式了解账户相关信息，系统能够提供格