学术文献管理系统 – 技术文档

概述

学术文献管理系统是一个基于Streamlit的Web应用，集成了学术文献搜索、收藏管理、引用关系可视化等功能。系统采用模块化设计，具有良好的可扩展性和维护性。

技术栈

前端技术

- Streamlit: 主要UI框架，提供Web界面和用户交互
- st-link-analysis: 自定义Streamlit组件，用于可视化引用关系图谱
- HTML/CSS: 界面样式和布局

后端技术

- Python 3.8+: 主要编程语言
- SQLite: 轻量级数据库，用于数据持久化
- Requests: HTTP客户端库，用于API调用

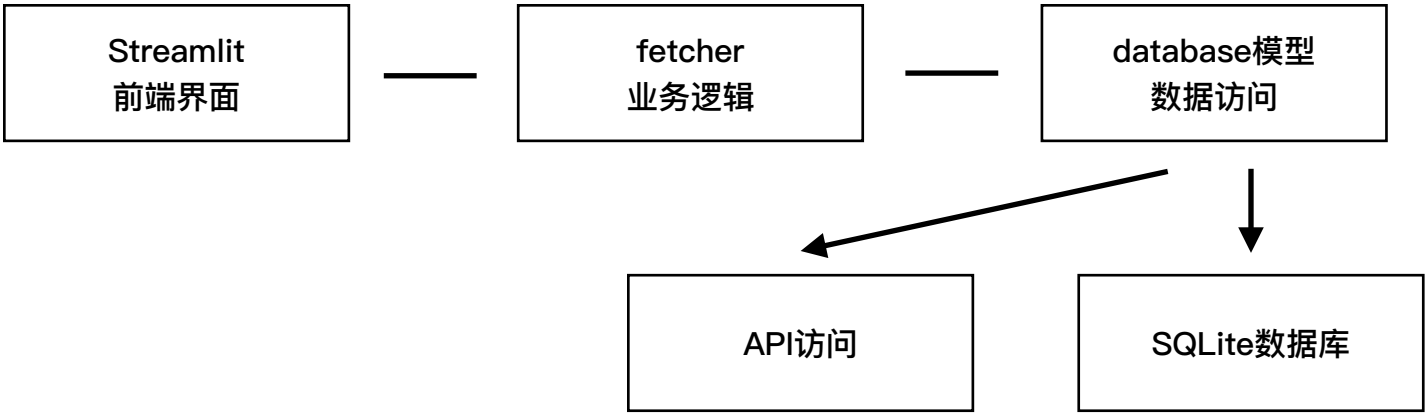
第三方服务

- Omini AI API: 学术文献数据源，提供文献搜索和引用关系数据

数据模型

- Pydantic/Dataclasses: 数据验证和模型定义

系统架构



模块设计

- 数据模型模块 (models/paper.py)：主要用于定义核心数据结构

```
class Paper:
paper_id: str          # 论文唯一标识
title: str             # 论文标题
```

authors: list # 作者列表
year: int # 发表年份
abstract: str # 摘要
citation_count: int # 被引用次数
reference_count: int # 参考文献数量
url: str = "" # PDF链接

- 数据访问层 (engine/database.py)：数据库操作封装，提供数据持久化功能
  数据表设计:
    paper表 – 用户收藏的论文
      - id (TEXT PRIMARY KEY): 论文ID
      - title (TEXT): 论文标题
      - authors (TEXT): JSON序列化的作者列表
      - year (INTEGER): 发表年份
      - abstract (TEXT): 摘要
      - citation_count (INTEGER): 引用次数
      - reference_count (INTEGER): 参考文献数量
      - url (TEXT): PDF链接
      - added_time (TIMESTAMP): 添加时间
  关键方法:
  - get_all_papers(): 获取所有收藏论文
  - paper_exists(): 检查论文是否存在
  - add_paper()/remove_paper(): 添加/移除收藏
  - update_relations(): 更新引用关系缓存
  - get_relations(): 获取引用关系
- API服务层 (engine/fetcher.py): 与外部API交互，获取文献数据
  核心函数:
  - fetch_papers(keyword: str): 根据关键词搜索文献
  - get_paper_relations(paper_id: str): 获取文献的引用关系
  - get_simple_references()/get_simple_citations(): 获取简化版引用关系

  API集成特点:
  - 使用Bearer Token认证
  - 支持字段选择，减少不必要的数据传输
  - 错误处理和空结果处理
  - 与数据库缓存层协同工作
- 页面模块
  搜索页面 (search.py)：文献搜索和结果展示
  实现特点:
  - URL参数管理，支持深链接和状态保持
  - 动态过滤和分页功能
  - 响应式界面设计

核心组件:
- 搜索表单
- 年份范围滑块
- 作者多选过滤器
- 分页控制器
- 论文卡片组件

收藏页面 (favorites.py)：收藏文献管理和浏览

实现特点:
- 复用搜索页面的过滤和展示逻辑
- 独立的收藏管理功能
- 与搜索页面共享详情查看功能

- 可视化组件 (st_link_analysis)：引用关系图谱可视化

技术特点:
- 基于Cytoscape.js的Streamlit封装
- 支持节点双击事件处理
- 可定制的节点和边样式
- 动态布局算法

交互设计:
- 节点颜色编码（当前文献、参考文献、引用文献）
- 标题截断优化显示
- 双击节点跳转到对应文献详情

核心实现细节

- 状态管理

系统使用多种状态管理机制:
- URL参数: 用于页面导航和状态持久化
- Session State: 用于临时数据存储（如搜索结果）
- 数据库: 用于持久化数据存储（如收藏列表）

- 错误处理
  - API调用异常处理
  - 数据库操作异常处理
  - 用户输入验证
  - 空结果友好提示
- 性能优化
  - 分页加载，减少单次渲染数据量
  - 数据库索引优化
  - API字段选择，减少网络传输

部署考虑

环境要求

- Python 3.8+
- SQLite3
- 网络访问权限（用于API调用）

配置管理

- API密钥管理
- 数据库路径配置
- 缓存策略配置

未来扩展方向

1. 多数据源支持: 集成多个学术数据库API
2. 高级搜索: 支持复杂搜索条件和布尔运算
3. 批量操作: 支持批量导出、导入文献数据
4. 协作功能: 支持用户间共享收藏和注释
5. 离线功能: 支持离线访问已缓存文献
6. AI推荐: 基于用户行为推荐相关文献

这个技术架构提供了良好的基础，既满足了当前功能需求，也为未来扩展留下了充足的空间。