前端文档

项目文件结构

- components/
    - NotLoggedInView.vue 登录前主页面
    - NotLoggedInMenu.vue 登录前的侧边菜单
    - PrivacyPolicy.vue 隐私政策页面
    - headBar.vue 头部
    - LoginView.vue 登录页面
    - RegisterView.vue 注册页面
    - HomeView.vue 登录后主页面
    - Menu.vue 登录后侧边菜单
    - Chat.vue 写作的主页面
    - ChatSelect.vue 写作左面的选择项目列表
    - Project.vue 项目的主页面
    - Dialog.vue 单个对话
    - Paragraph.vue 单个段落
    - Papers.vue 论文库的主页面
    - Search.vue 搜索的主页面
    - SearchResultCard.vue 搜索结果卡片
    - Setting.vue 设置的主页面
    - endpoints.js 用于定义 API 地址
    - tauth.js 用于处理用户认证
- css/
    - homestyle.css 用于处理登录后页面的样式
    - style.css 用于处理整体的样式
    - loginstyle.css 用于处理登录页面的样式
    - vscode-darkplus.css 代码高亮样式
    - docstyle.css 隐私政策样式

技术栈

- 使用 Vue3 作为前端框架
- 使用 Webpack 作为构建及调试工具
- 使用 Element Plus 作为 UI 组件库
- 使用 Babel 作为编译器
- 使用 marked 作为 Markdown 解析器
- 使用 highlight.js 作为代码高亮工具

页面结构

总地来说，路径一共有 5 个：

- 登录前页面
- 登录页面
- 注册页面
- 条款页面
- 登录后页面（主页面）

其中在主页面内的组件实现了功能切换，但并不会改变路径。

可能切换的组件有：

- 设置
- 论文库
- 搜索
- 写作

复用的组件有：

- 头部，通过 Slot 传入不同的标题
- 侧边菜单
- 对话（即一个聊天）文字过长时自动折叠（会自动识别单词，不会在单词中间折叠）
- 段落
- 搜索结果卡片

菜单

登录

注册

InfiniDoc

与任意 API 结合使用[1]

您可以选择您喜欢的大语言模型接入本系统，以提供定制化和个性化的服务。

多端同步的使用体验

您的信息将被安全地存放在云端[2]，供您在任何位置和设备随时使用。

论文导入方式

支持 PDF 导入[3]。

自动的引用联想

当您提出要求时，如果相关文本涉及引用到的文献，且该文献已加入论文库，大模型将考虑被引用文献的内容。

流式生成

将实时生成的文本流式传输到您的设备。

InfiniDoc 隐私政策

更新日期：2025年3月5日

生效日期：2025年3月5日

《InfiniDoc隐私政策》（以下称“本政策”）适用于InfiniDoc人工智能基础技术研究有限公司及其关联公司（以下简称“我们”或“InfiniDoc”）推出的InfiniDoc人工智能服务（以下简称“InfiniDoc”或“本服务”）。您在使用本服务前，请仔细阅读本政策，以了解我们处理您个人信息的详情。

除非本隐私政策另有说明，本隐私政策适用于我们提供的InfiniDoc网页、应用程序、小程序、供第三方网站和应用程序使用的软件开发工具包（SDK）和应用程序编程接口（API）以及随着技术发展出现的创新形态方式提供的产品和服务。如您使用我们或关联公司向您提供的InfiniDoc的某项或某几项服务，有其单独的隐私政策的，则相应服务或产品适用相应隐私政策；**如未设立单独隐私政策的，则本政策同样适用于该部分产品或服务。**

请您知悉，终端用户访问开发者使用我们的开放平台服务开发的下游系统或应用时，其被收集的个人信息的处理规则不在本隐私政策的说明范围内，运营该应用的开发者作为该个人信息处理活动的处理者应当向终端用户披露相关个人信息保护规则。

当您使用或开启相关功能或使用服务时，为实现功能、服务所必需，我们会收集、使用相关信息。除非是为实现业务功能或根据法律法规要求所必需的必要信息，您均可以拒绝提供且不影响其他功能或服务。我们将在隐私政策中逐项说明所要收集的信息。

相机权限均不会默认开启，只有经过您的明示授权才会在为实现特定功能或服务时使用，您也可以撤回授权。**特别需要指出的是，即使经过您的授权，我们获得了这些敏感权限，也不会在相关功能或服务不需要时而收集您的信息。**

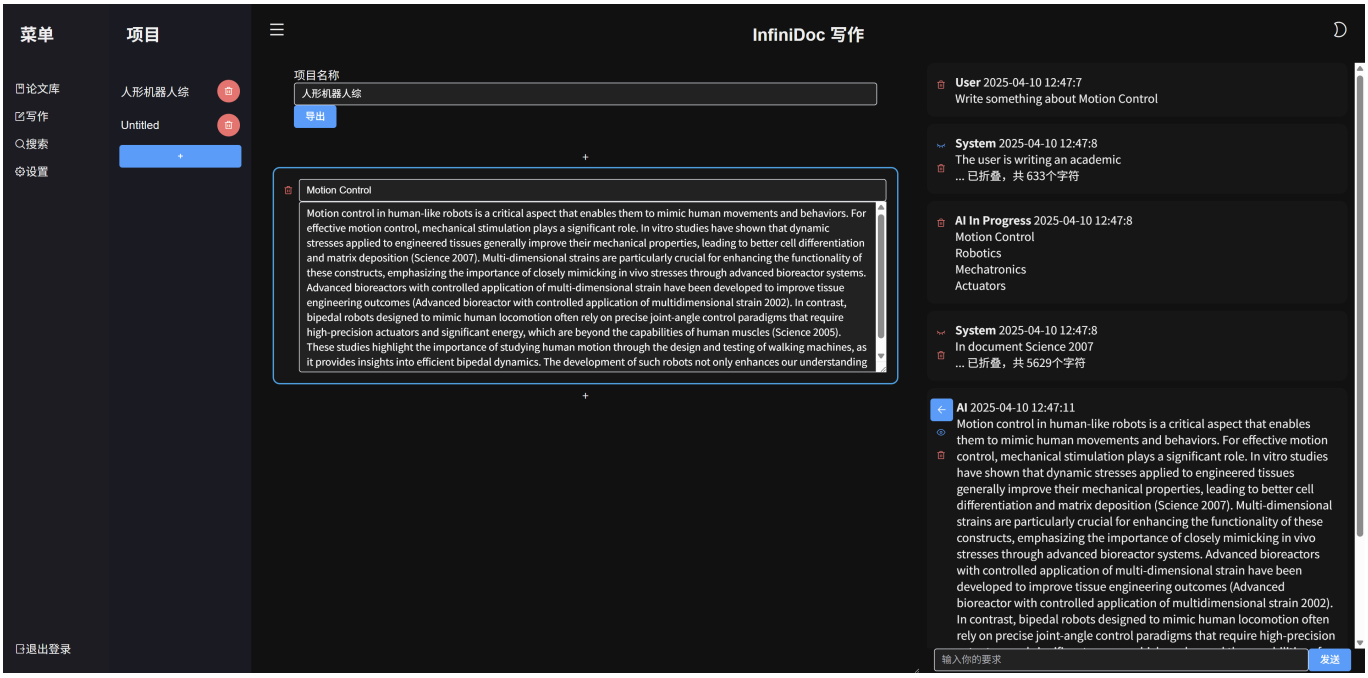
下文将帮您详细了解我们如何收集、使用、存储、转移（如适用）与保护个人信息；帮您了解查询、复制、删除、更正、撤回授权个人信息的方式。本政策与您使用我们的服务关系密切，我们建议您仔细阅读并理解本政策全部内容，作出您认为适当的选择，**有关您个人信息权益的条款重要内容我们已用加粗形式提示，请特别关注。**

一、我们如何收集和使用您的个人信息

实现功能

深色模式

实现了深色和浅色两套配色，进入页面时根据用户的系统设置自动选择深色或浅色模式，用户可以在页面右上角手动切换深色和浅色模式。



画面过渡

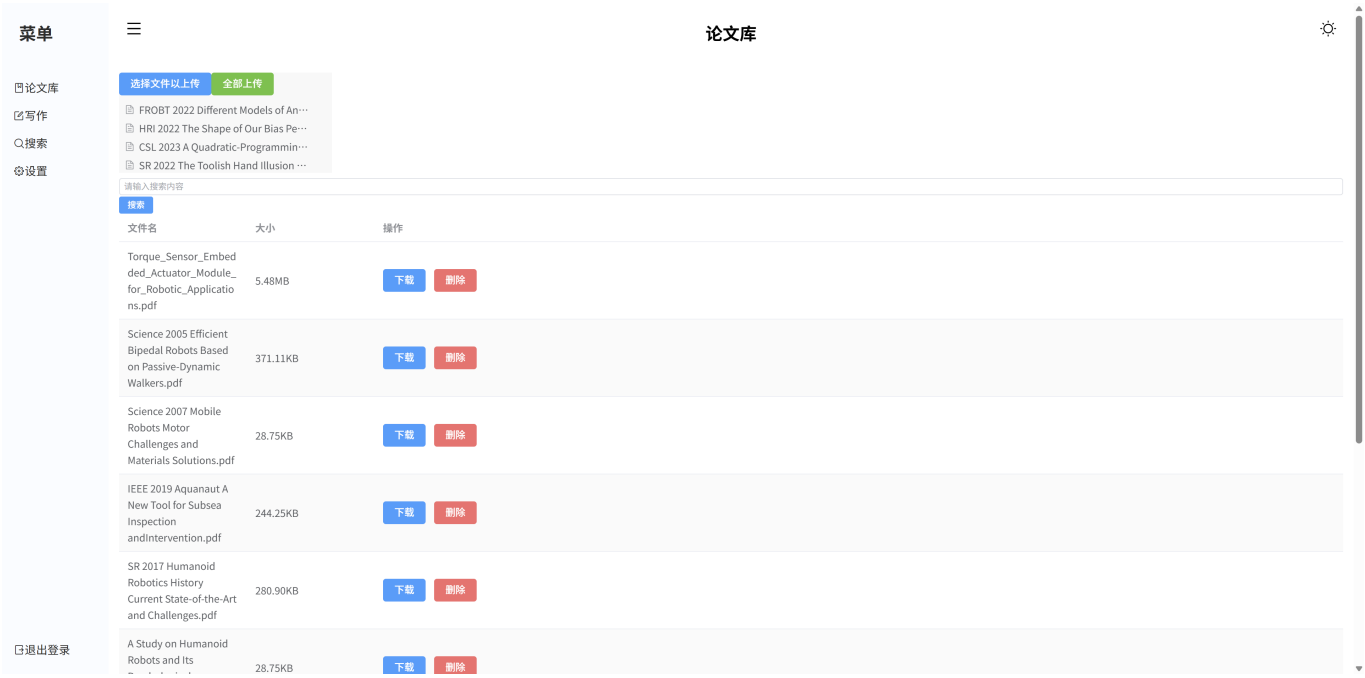
使用 Vue 的过渡组件实现了页面切换时的过渡效果，使用 CSS 的 transition 属性实现了元素的过渡效果。

弹性布局

使用 CSS 的 Flexbox 实现了页面的弹性布局。

上传页面

使用 Element Plus 的 Upload 组件实现了文件上传功能，支持拖拽上传和点击选择文件上传。

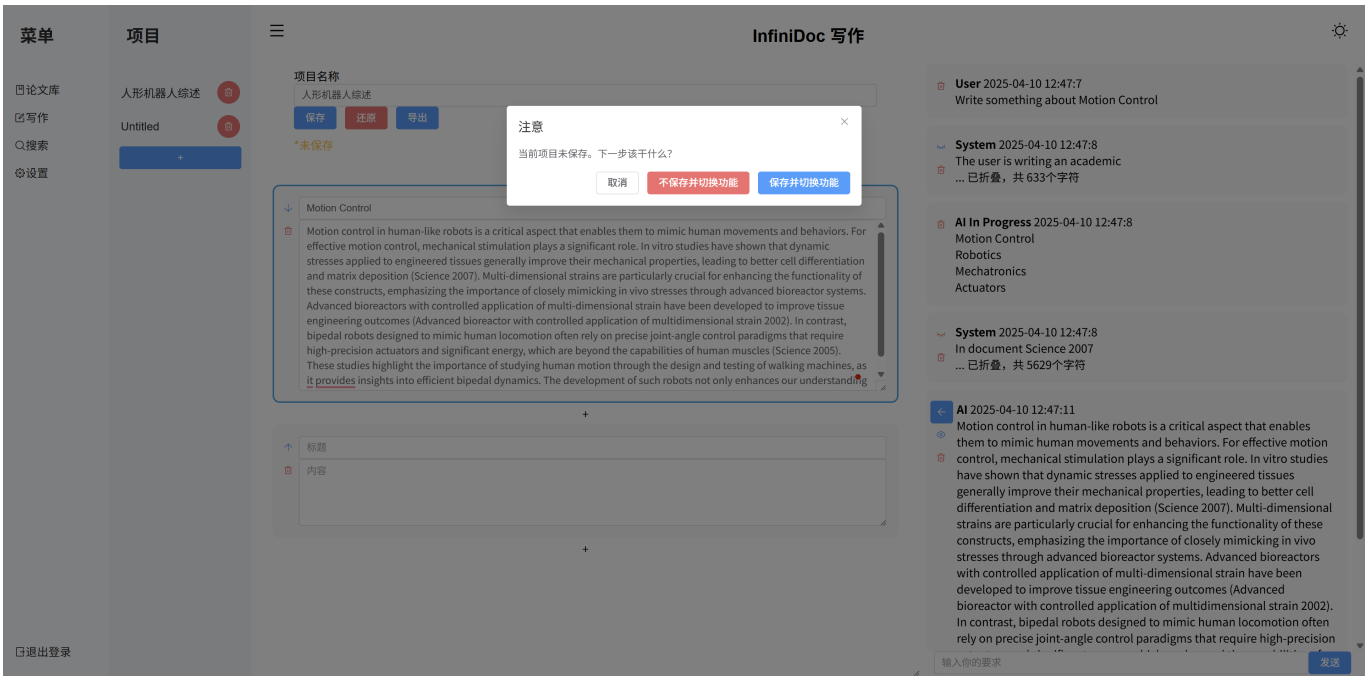


Websocket 实现 LLM 的流式推理

使用 Websocket 实现了 LLM 的流式推理，使用了 Vue 的响应式系统实现了数据的双向绑定。当接收到 Websocket 数据包时，将数据包中的数据解析并更新到页面上。

打断切换

当用户有改动未保存时，点击切换组件会弹出提示框，询问用户是否保存改动，也会 hook 浏览器关闭事件，询问用户是否确定离开。



API 调用

在 localStorage 中存储了 Token，在每次请求时从 localStorage 中获取 Token，并将 Token 添加到请求头中。

后端技术实现文档

该文档介绍后端的具体实现。

项目文件结构

- server.py 用于启动 uvicorn 服务器
- main.py 定义服务器，提供 router，引入其他 router，包括 Websocket 的处理
- schemas.py 定义请求体
- tauthAuthenticator.py 对接第三方认证系统的支持库

- authenticate.py 认证系统的实现
- filel.py 文件上传与解析的实现，包括删除和更改操作
- projectManager.py 项目管理的实现，包括保存和加载操作
- largeModel.py LLM 交互相关的实现，包括核心的对话逻辑
- route_file.py 文件上传与解析的路由
- route_project.py 项目管理的路由
- route_settings.py 设置与自定义
- route_user.py 用户管理的路由，包括注册、登录、注销等操作
- dbpassword.py(.default) 数据库密码

用户管理

用户相关数据库表设计

主要涉及以下两个表：

```
CREATE TABLE `tokens` (
  `token` varchar(64) NOT NULL,
  `time_accessed` bigint NOT NULL,
  `UNIQUE_ID` tinytext NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE `users` (
  `username` varchar(32) NOT NULL,
  `password` varchar(64) NOT NULL,
  `salt` varchar(32) NOT NULL,
  `id` int NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
ALTER TABLE `tokens`
  ADD UNIQUE KEY `token` (`token`);
ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD UNIQUE KEY `username` (`username`);
```

用户名不应当重复。

注意到有大量检索 token 的需求，因此我们在 `tokens` 表中添加了索引。

```
ALTER TABLE `users`
  MODIFY `id` int NOT NULL AUTO_INCREMENT;
```

`tokens` 表用于存储用户的 token 信息，这是前端调用后端 API 的凭证。`token` 在登录时被生成，存储在数据库中。`time_accessed` 用于记录 token 的最后访问时间，便于后续的 token 过期处理。`UNIQUE_ID` 用于标识用户的唯一身份。这用于唯一标识用户，便于后续的权限管理和数据隔离。

`users` 表用于存储用户的基本信息，包括用户名、密码和盐值。密码经过加密存储，盐值用于增强密码的安全性。`id` 是用户的唯一标识符，用于在系统中区分不同的用户。请注意，这里只维护在本站点注册的用户的信息，第三方登录不储存在其中。此外，这里的 `id` 和上述的 `unique_id` 不同，但存在一一对应关系（但存在 `unique_id` 的用户不一定存在 `id`）。

注册与登录

用户注册与登录的功能主要通过 `POST` 请求实现。详见 API 文档。

密码存储：使用了 哈希(哈希(密码)+盐值) 的方式存储密码，即防止被彩虹表攻击。

注意，由于默认启用 https，可认为使用 POST 传输明文密码是安全的。但即便如此，我们仍然在前端计算第一层的哈希值。

校验用户名格式：应当为大小写字母、数字和`-_` 组成，长度为 4-32 位。

> 我们默认用户应该对其密码安全性负责，因此不对密码的复杂性进行校验。

第三方登录

用户先向第三方平台申请一个令牌，然后后端将向第三方平台发送请求，以验证该令牌的有效性并获取用户的标识符。

如果验证成功，后端将生成一个 token 返回给前端。前端可以使用该 token 进行后续的 API 调用。

Token

用户在访问不显式涉及 Token 的 API 时，后端会自动从请求头中提取 Token。

设置与自定义 LLM

设置与自定义 LLM 的数据库表设计

```
CREATE TABLE `user_settings` (
  `UNIQUE_ID` varchar(32) NOT NULL,
  `setting` varchar(32) NOT NULL,
  `value` text NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
ALTER TABLE `user_settings`
  ADD PRIMARY KEY (`UNIQUE_ID`,`setting`);
```

用户 ID 和设置项作为联合主键。 `setting` 用于标识设置项的名称，`value` 用于存储设置项的值。

自定义 LLM 的模型获取

```python
def get_models(endpoint, api_key):
    try:
        client = OpenAI(
            api_key=api_key,
            base_url=endpoint
        ) if api_key != "" else OpenAI(
            base_url=endpoint
        )
        models = client.models.list()
        return models
    except Exception as e:
        return []
```

可以直接使用 OpenAI 的 API 获取模型列表。

自定义 LLM 的测试

这实际上是一个非常基础的对话调用。但在此我们一并讲述异步调用的实现。

```python
async def get_ai_response(message: str, client: AsyncOpenAI, model: str) ->
AsyncGenerator[str, None]:
    response = await client.chat.completions.create(
        model=model,
        messages=[
            {
                "role": "system",
                "content": (
                    "You are a helpful assistant, skilled in explaining "
                    "complex concepts in simple terms."
                ),
            },
            {
                "role": "user",
                "content": message,
            },
        ],
        stream=True,
    )

    all_content = ""
    async for chunk in response:
        content = chunk.choices[0].delta.content
        if content:
            all_content += content
            yield content
```

这可以当大模型流式返回一些内容时，逐步返回给前端。每 `yield` 一次，前端就会收到一次数据。

`WebSocket` 是一个长连接，它主要是解决“后端不能主动推送数据给前端”的问题。解决了轮询带来的性能损失。

论文库管理

论文库相关数据库表设计

```
CREATE TABLE `files` (
  `id` int NOT NULL,
  `size` int NOT NULL,
  `sha256` varchar(64) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE `user_files` (
  `UNIQUE_ID` varchar(32) NOT NULL,
  `fileid` int NOT NULL,
  `name` text NOT NULL,
  `seq` int NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
ALTER TABLE `files`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `user_files`
  ADD PRIMARY KEY (`seq`),
  ADD KEY `UNIQUE_ID` (`UNIQUE_ID`);
```

注意我们并不禁止用户将同一个文件上传多次。它对应不同的 `seq` 值。

文件上传与解析

文件管理系统分离为两个部分：硬盘上文件与数据库中文件的对应关系 与 用户与数据库中文件的对应关系。

将计算用户上传文件的 `sha256` 值，若数据库中文件已有该值，则只需要维护数据库中文件与用户的对应关系。

否则，需要将该文件保存至对应目录（以 `sha256` 的前两位为目录），并在数据库中插入一条记录。

文件删除

注意到，文件删除不仅应该删除用户与数据库中文件的对应关系，还应当删除在向量数据库中的数据。

注意，我们设计即使用户完全删除了文件，即不存在任何该文件与用户的对应关系，但磁盘上仍然保留该文件。这是监管合规的要求。

然后通知向量数据库（算法层）删除该文件对应的记录。

分页查询

只需要使用 `LIMIT` 和 `OFFSET` 即可实现，注意 OFFSET 的值应当为 `page * batch_size`，其中 `page` 从 0 开始。

```
mysql_cursor.execute(
        "SELECT `name`,(SELECT size from `files` where `id`=`fileid`),`seq` FROM
`user_files` WHERE `UNIQUE_ID`=%s ORDER BY `seq` DESC LIMIT %s OFFSET %s;",
(unique_id, batch_size, offset))
```

项目管理

项目相关数据库表设计

```
CREATE TABLE `user_projects` (
  `UNIQUE_ID` varchar(32) NOT NULL,
  `project_id` int NOT NULL,
  `project_name` text NOT NULL,
  `paragraphs` mediumtext NOT NULL,
  `deleted` tinyint(1) NOT NULL DEFAULT '0'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
ALTER TABLE `user_projects`
  ADD PRIMARY KEY (`project_id`);
ALTER TABLE `user_projects`
  MODIFY `project_id` int NOT NULL AUTO_INCREMENT;
```

其中 paragraphs 保存的是 JSON 格式的字符串，表示该项目的段落信息。

保存与加载

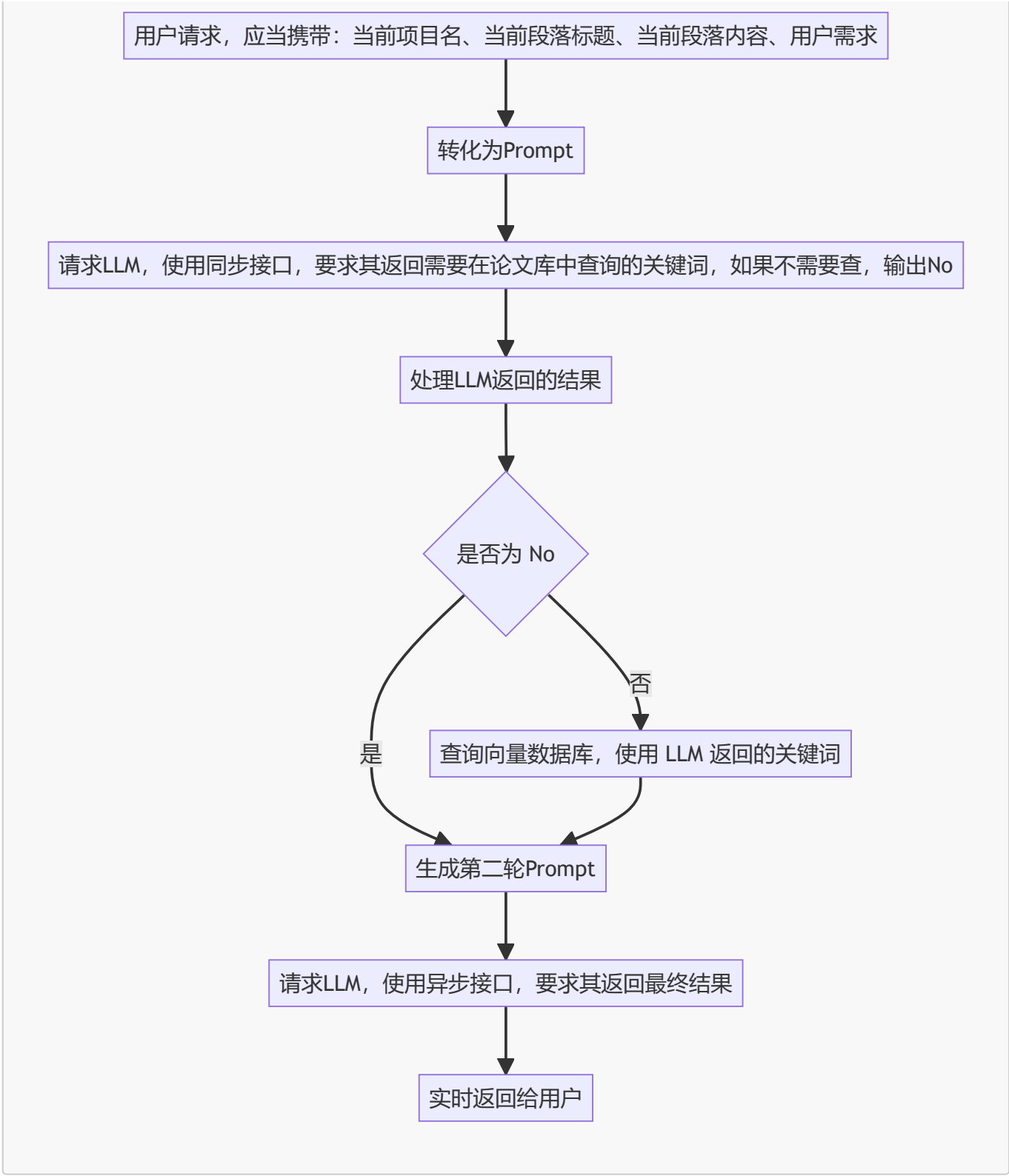
这里其实值得讨论的是：应该向数据库塞一整个数据包（对应一个项目），还是将项目的各个段落拆开，标题拆开，分别存储？我认为各有优劣，但主流的还是存整个数据包。具体原因在于：前端实际上相当于一个 Web App，考虑我们日常编写 Word 等文档的习惯，都是将整个文档存储为一个文件，而不是将每一段文字都存储为一个文件。另外，这样做相当于把打包和解包的工作分散到前端。

删除

使用 `deleted` 字段来标记项目是否被删除，而不是直接删除该条记录。这可以实现回收站操作，以及确保合规性审查。

对话与写作

每一次段落写作其实可以认为是一个工作流。



注意到，我们实际上并没有也不需要实现上下文的保存，只是提供了当前段落的内容。这就可以完成诸如“请帮我改成学术论文的语气”之类的需求。

与算法层交互

使用 RESTFul API 的方式与算法层交互。

算法层技术实现文档

该文档介绍算法层的实现。

技术栈

该部分主要涉及论文库的解析与 ChromaDB 的使用。

用户分割

对于不同用户之间的论文相互隔离，不同用户使用不同的 collection 来存储论文。查询时限定在该用户的 collection 中进行。

论文解析

对于每一个 PDF 文件，使用 PyPDF 读取 PDF 文件，提取文本。提取文本后，根据固定长度的字符串进行分割，以灵活查询论文的不同片段，也规避了 ChromaDB 的 token 限制。在读取后，根据文件的 sha256 码和第几个片段创建一个唯一 id，另外将原来的文件 ID 作为标签，以便后续分类查询与删除。

关键词检索

实际上是在 ChromaDB 中进行的检索。注意，制作了一些精细化处理，例如：检索到的两个最匹配片段分别位于论文 A 的第 10 段和论文 B 的第 10 段。他会自动将前后的段落（+-1）也查询出来，考虑到文本的连贯性。

另外，对于一些情况做了合并，例如查询到的片段是 A 的 10 和 12 段，则理论上需要以下片段

```
9 10 11
11 12 13
```

系统会自动合并相邻的片段，输出会类似于

```
chunk 9 - 13: 9-13段的内容
```

多关键词检索

检索多次，然后一齐合并。

算法层代码

较为简单，都放在 `main.py` 中了。包括上述的功能，以及 pydantic 的请求体。

API 文档

该文档介绍前后端交互接口

RESTful API

Verify Token （验证 Token）

方法	URL
POST	/login/verifyToken

请求体

Field	类型	详细	必须
token	string		必须

响应 (200)

Field	类型	详细
success	boolean	是否为有效 Token

Login With Tauth （使用 Tauth 登录）

方法	URL
POST	/login/tauth

请求体

Field	类型	详细	必须
token	string	TAuth（第三方平台）提供的 Token	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功登录
token	string	登录后返回的 token

Register （注册）

方法	URL
POST	/register/native

参数

Name	In	详细	必须

请求体

Field	类型	详细	必须
username	string	用户名	必须
password	string	密码（已经进行一次 sha256）	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功注册
token	string	返回的 token

Login （登录）

方法	URL
POST	/login/native

请求体

Field	类型	详细	必须
username	string	用户名	必须
password	string	密码	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功登录
token	string	返回的 token

Convert File （转换文件）

方法	URL
POST	/convert

请求体

Field	类型	详细	必须
markdown	string	源文档 Markdown 格式	必须
target	string	目标格式	必须

响应 (200)

一个 Octet Stream，表示转换后的文件

Upload File （上传文件）

方法	URL
POST	/upload

参数

Name	In	详细	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
file	file	上传的文件	必须

响应 (200)

即表示上传成功

File List （文件列表）

方法	URL
GET	/fileList

参数

Name	In	详细	必须
limit	query	最多返回的个数	必须
offset	query	从哪一个开始查询	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
files	array	文件列表
totalfiles	integer	文件总数

Download File （下载文件）

方法	URL
GET	/download

参数

Name	In	详细	必须
seq	query	哪一个文件	必须
infiniDocToken	header		必须

响应 (200)

一个 Octet Stream，表示下载的文件

Delete File （删除文件）

方法	URL
GET	/delete

参数

Name	In	详细	必须
seq	query		必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
success	boolean	是否成功删除

Search In Files （在文件中搜索）

方法	URL
GET	/search

参数

Name	In	详细	必须
keyword	query	关键词	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
result	object	搜索结果

Get Projects （获取项目列表）

方法	URL
GET	/project/get

参数

Name	In	详细	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
projects	array	项目列表

Create Project （创建项目）

方法	URL
POST	/project/create

参数

Name	In	详细	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
project_name	string	项目名称	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功创建项目
id	integer	项目 ID

Delete Project （删除项目）

方法	URL
POST	/project/delete

参数

Name	In	详细	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
project_id	integer	项目 ID	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功删除项目

Rename Project （重命名项目）

方法	URL
POST	/project/rename/{project_id}

参数

Name	In	详细	必须
project_id	path	项目 ID	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
new_name	string	新名称	必须

响应 (200)

Field	类型	详细

Field	类型	详细
success	boolean	是否成功重命名项目

Get Paragraphs （获取项目段落）

方法	URL
GET	/project/getparagraphs/{project_id}

参数

Name	In	详细	必须
project_id	path	项目 ID	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
paragraphs	string	段落 json

Get Project Name （获取项目名称）

方法	URL
GET	/project/name/{project_id}

参数

Name	In	详细	必须
project_id	path	项目 ID	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
project_name	string	项目名称

Save Project

方法	URL
POST	/project/save/{project_id}

参数

Name	In	详细	必须
project_id	path	项目 ID	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
paragraphs	string	段落 json	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功保存项目

Get Unique Id （获取唯一 ID）

方法	URL
POST	/settings/getUniqueID

参数

Name	In	详细	必须

请求体

Field	类型	详细	必须
token	string		必须

响应 (200)

Field	类型	详细
unique_id	string	唯一 ID

Get User Settings （获取用户设置）

方法	URL
GET	/user/settings

参数

Name	In	详细	必须
infiniDocToken	header		必须

响应 (200)

Field	类型	详细
success	boolean	是否成功获取设置
settings	object	用户设置

Set User Settings （设置用户设置）

方法	URL
POST	/user/settings/set

参数

Name	In	详细	必须
infiniDocToken	header		必须

请求体

Field	类型	详细	必须
payload	object	设置列表	必须

响应 (200)

Field	类型	详细
success	boolean	是否成功设置

Get Models （获取模型列表）

方法	URL
POST	/llm/getModels

参数

Name	In	详细	必须

请求体

Field	类型	详细	必须
endpoint	string		必须
api_key	string		必须

响应 (200)

Field	类型	详细
（根）	array	模型列表

WebSocket API

test （测试对话）

请求体

Field	类型	详细	必须
type	string	为 test	必须
message	string	用户的信息	必须
endpoint	string		必须
model	string		必须
key	string		必须

project （项目对话）

请求体

Field	类型	详细	必须
type	string	为 project	必须
project_name	string	项目名称	必须
paragraph_title	string	段落标题	必须
paragraph_current_content	string	段落当前内容	必须
user_prompt	string	用户要求	必须