1 Perplexica

1.1 相关地址

- 原 `github` 地址

  > https://github.com/ItzCrazyKns/Perplexica

- 更改后的（自己的）

  > https://github.com/qinchihongye/Perplexica-pa

- 一直刷新处理(issue)

  > https://github.com/ItzCrazyKns/Perplexica/issues/180

- 新增 oneapi（功能不完整）

  > https://github.com/ItzCrazyKns/Perplexica/issues/367

1.2 其他修改

1.2.1 更改 dockerfile

- 更改后端 dockerfile

  > backend.dockerfile

  增加

  ```
  RUN apt-get update
  RUN apt-get install -y vim
  ```

- 更改前端 dockerfile

  > app.dockerfile

  增加前端用到的环境变量(后续启动 python 服务的地址和端口)

```
ARG NEXT_PUBLIC_PY_API=http://host.docker.internal
ARG NEXT_PUBLIC_PY_PORT=8013

ENV NEXT_PUBLIC_PY_API=${NEXT_PUBLIC_PY_API}
ENV NEXT_PUBLIC_PY_PORT=${NEXT_PUBLIC_PY_PORT}
```

1.2.2 更改 docker-compose.yaml文件

- 增加前端用到的环境变量

  perplexica-frontend 下的 args 中增加

```
      - NEXT_PUBLIC_PY_API=http://host.docker.internal
      - NEXT_PUBLIC_PY_PORT=8013
```

- 更改IP 地址

```
    args:
      - NEXT_PUBLIC_API_URL=http://127.0.0.1:3001/api
      - NEXT_PUBLIC_WS_URL=ws://127.0.0.1:3001
      - NEXT_PUBLIC_PY_API=http://host.docker.internal
      - NEXT_PUBLIC_PY_PORT=8013
```

  将 `127.0.0.1` 替换为运行 `Perplexica` 服务器的 IP地址，host.docker.internal 也一样更改

- 更改前端端口

```
  ports:
    - 3000:3000

  ports:
    - 8345:3000
```

  将`perplexica-frontend:` 下的 ports 改为 `8345:3000`

1.2.3 更改图标

- 更改展示网页来源时候的图标

  先找个图标，命名为 `favicon.ico`

  图标放置位置

> /ui/components/favicon.ico

更改文件

> /ui/components/MessageSources.tsx

替换下面第一行的为第二行（共三处）

```
src={`https://s2.googleusercontent.com/s2/favicons?domain_url=${source.metadata.url}`}

src='./favicon.ico'
```

1.2.4 更改后端配置文件

- 配置文件地址

> /cofig.toml

```
[API_KEYS]
OPENAI = "sk-PJrYIIj1QwodFT4n00Df8956B465411389C44dF11eDc07Da"
GROQ = ""
ANTHROPIC = ""
GEMINI = ""

[API_ENDPOINTS]
OLLAMA = "http://host.docker.internal:11434"
OPENAIBASEURL = "http://38.54.50.54:8025/v1"
SELFSEARCHURL = "http://host.docker.internal"
SEARXNG = "http://localhost:32768"

[GENERAL]
PORT = 3_001
SIMILARITY_MEASURE = "cosine"
KEEP_ALIVE = "5m"
PYPORT = 8_013
LLM_NAME = "qwen2.5-72b-instruct"
TIMEOUT = 50
RETRIEVALURL = "http://js1.blockelite.cn:14875/search"
VERIFYTOKEN = "d27121560bdb4d9d8a780cb24bf9a399"
```

配置项中 `OPENAIBASEURL`、`SELFSEARCHURL`、`PYPORT` 是 自己后加的。

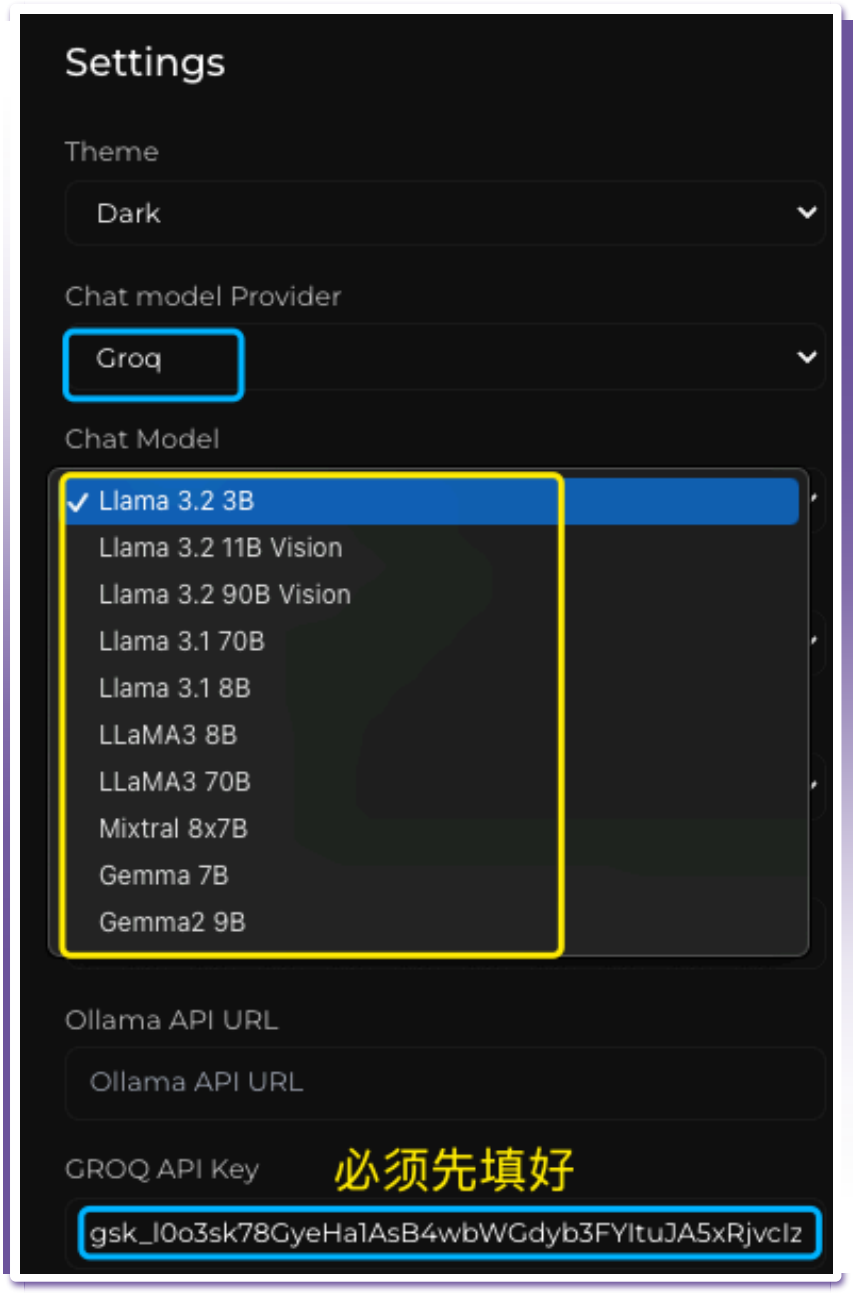
说明如下：

1. `OPENAI`：可以使用 OpenAI 的 API-Key，也可以使用支持 OpenAI 接口的其他 API-Key。使用时需与 OPENAIBASEURL 配合,增删模型在如下地址设置。

> src/lib/providers/openai.ts

2. `GROQ`： GROQ 的 api-key,超低延迟推理，只有设置后在前端页面才会在"Chat model Provider"选项中出现，并自动在"Chat Model"选项框中呈现支持的模型, 新模型可在 以下地址配置。

> /src/lib/providers/groq.ts



3. `ANTHROPIC`: 同上，新模型可在以下地址设置

> src/lib/providers/anthropic.ts

4. `GEMINI`:同上，新模型可在以下地址设置

> src/lib/providers/gemini.ts

5. `OLLAMA` : ollama 服务的启动地址，若是 docker启动的填

> **http://host.docker.internal:11434**

6. `SEARXNG` : SEARXNG服务地址。
7. `OPENAIBASEURL`： 用于指定基础 URL，可以是 OneAPI 的 baseurl，或者是符合 OpenAI 接口格式的 URL。
8. `SELFSEARCHURL`： 搜索接口地址（/src/lbi/searxng.ts 中使用的）。
9. `PORT`： 后端服务的端口
10. `SIMILARITY_MEASURE`： 要使用的相似性度
11. `KEEP_ALIVE`： 等待ollma 模型加载的时间
12. `PYPORT`： python 服务启动的端口。
13. `LLM_NAME` : python 服务中 query改写的模型
14. `TIMEOUT` : python 服务中检索服务的超时限制、
15. `RETRIEVALURL` : 由美芝提供的检索服务地址
16. `verfiy_token` : 后端鉴权 token

1.3 增加配置项(config.toml)

- 以增加 `OPENAIBASEURL` 、 `SELFSEARCHURL` 、 `PYPORT` 、 `VERIFYTOKEN` 为例

  将 `OPENAIBASEURL` 和 `SELFSEARCHURL` 增加在 config.toml 的[API_ENDPOINTS]下

  ```
  OPENAIBASEURL = ""
  SELFSEARCHURL = ""
  ```

  将 `PYPORT` 增加在[GENERAL]下面

  ```
  PYPORT =
  VERIFYTOKEN =
  ```

- 备注

1. 如果不需要在前端页面中更改，则只需要更改第一步`更改 src/config.ts`即可,后续直接在代码中 import,比如PYPORT和 VERIFYTOKEN。
2. 记得如果在容器中更改，改完要编译

> npm run build

1.3.1 更改 src/config.ts

- 地址

> /src/config.ts

- `Config` 类型中的 `API_ENDPOINTS` 下 增加

```
OPENAIBASEURL: string;
SELFSEARCHURL: string;
```

GENERAL下增加

```
PYPORT: number;
VERIFYTOKEN: string;
```

- 增加导出函数

```
export const getOpenaiBaseUrl = () =>
loadConfig().API_ENDPOINTS.OPENAIBASEURL;

export const getSelfSearchUrl = () =>
loadConfig().API_ENDPOINTS.SELFSEARCHURL;

export const getPyPort = () => loadConfig().GENERAL.PYPORT;

export const getVerifyToken = () =>
loadConfig().GENERAL.VERIFYTOKEN;
```

1.3.2 更改 routes/config.ts

- 地址

> /src/routes/config.ts

- 新增

```
import {getOpenaiBaseUrl} from '../config';
import {getSelfSearchUrl} from '../config';

//在 60 行加入
config['openaiBaseUrl'] = getOpenaiBaseUrl();

//在 61 行加入
config['selfSearchUrl'] = getSelfSearchUrl();

//在 updatedConfig 中API_ENDPOINTS下加入
OPENAIBASEURL: config.openaiBaseUrl,
SELFSEARCHURL: config.selfSearchUrl,
```

1.3.3 更改 SettingsDialog.tsx

- 地址

> /ui/components/SettingsDialog.tsx

- `SettingsType` 类型中增加

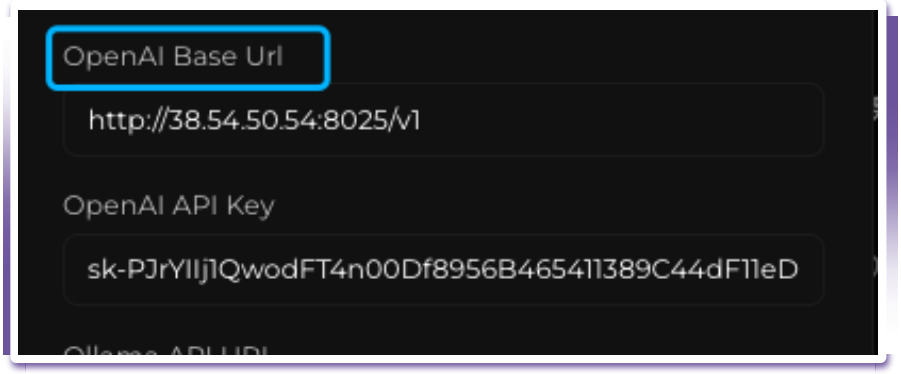
```
openaiBaseUrl: string;

selfSearchUrl: string;
```

- `Return` 中增加 (对应网页的设置页面，找个合适的位置)

1. 放在`OpenAI API Key`上面

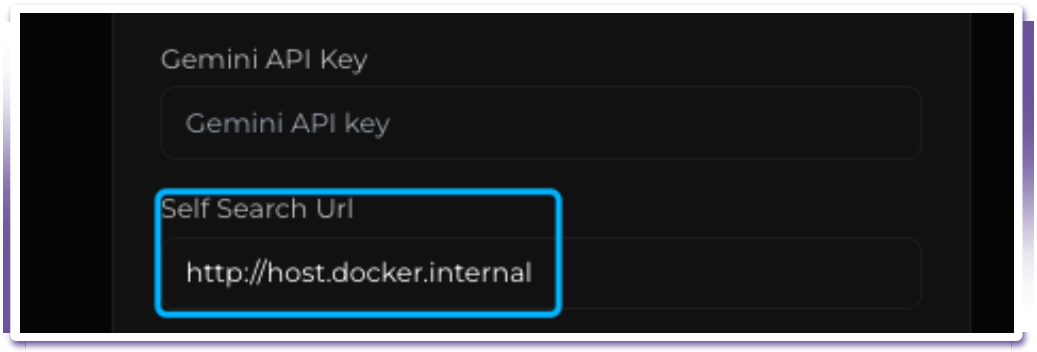
```
                    <div className="flex flex-col space-y-1">
                      <p className="text-black/70 dark:text-
white/70 text-sm">
                        OpenAI Base Url
                      </p>
                      <Input
                        type="text"
                        placeholder="OpenAI Base Url"
                        defaultValue={config.openaiBaseUrl}
                        onChange={(e) =>
                          setConfig({
                            ...config,
                            openaiBaseUrl: e.target.value,
                          })
                        }
```

```
                        />
                      </div>
```



2. 放在 `Gemini API Key` 下面

```
                    <div className="flex flex-col space-y-1">
                      <p className="text-black/70 dark:text-white/70 text-sm">
                        Self Search Url
                      </p>
                      <Input
                        type="text"
                        placeholder="Self Search Url"
                        defaultValue={config.selfSearchUrl}
                        onChange={(e) =>
                          setConfig({
                            ...config,
                            selfSearchUrl: e.target.value,
                          })
                        }
                        />
                    </div>
```



1.4 增加 models

1.4.1 修改 oneapi 的 baseurl

- 文件地址

> /src/lib/providers/openai.ts

- 增加代码

```
1   import { getOpenaiBaseUrl } from '../../config';
```

- 在 chatModels 下更改模型的默认 base_url

  原

```
1         'gpt-3.5-turbo': {
2           displayName: 'GPT-3.5 Turbo',
3           model: new ChatOpenAI({
4             openAIApiKey,
5             modelName: 'gpt-3.5-turbo',
6             temperature: 0.7,
7           }
8         },
```

  更改后

```
1         'gpt-3.5-turbo': {
2           displayName: 'GPT-3.5 Turbo',
3           model: new ChatOpenAI({
4             openAIApiKey,
5             modelName: 'gpt-3.5-turbo',
6             temperature: 0.7,
7           },{
8             baseURL: getOpenaiBaseUrl()
9           }),
10        }
```

  因使用 `customer openai` 指定 `baseurl` 和 `apikey` 会无法使用完整功能，所以直接在 openai.ts 中更改，增加 `baseURL: getOpenaiBaseUrl()`

  备注： 新版中 `customer openai` 应该也是可以使用完整功能，但是因为 `需要自己填` model_name,所以还是直接在 `openai.ts` 中配置 `baseURL`，这样可以直接在前端下拉框界面选择已经配置好的模型。

1.4.2 增加模型(以增加 qwen-long 为例子)

- 使用 `oneapi` 中转，需提前配置好模型(其他途径类似)
- 文件地址

> /src/lib/providers/openai.ts

- 代码

  在`chatModels`下增加

```
        'qwen-long': {
          displayName: 'Qwen-Long',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'qwen-long',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
```

  其中

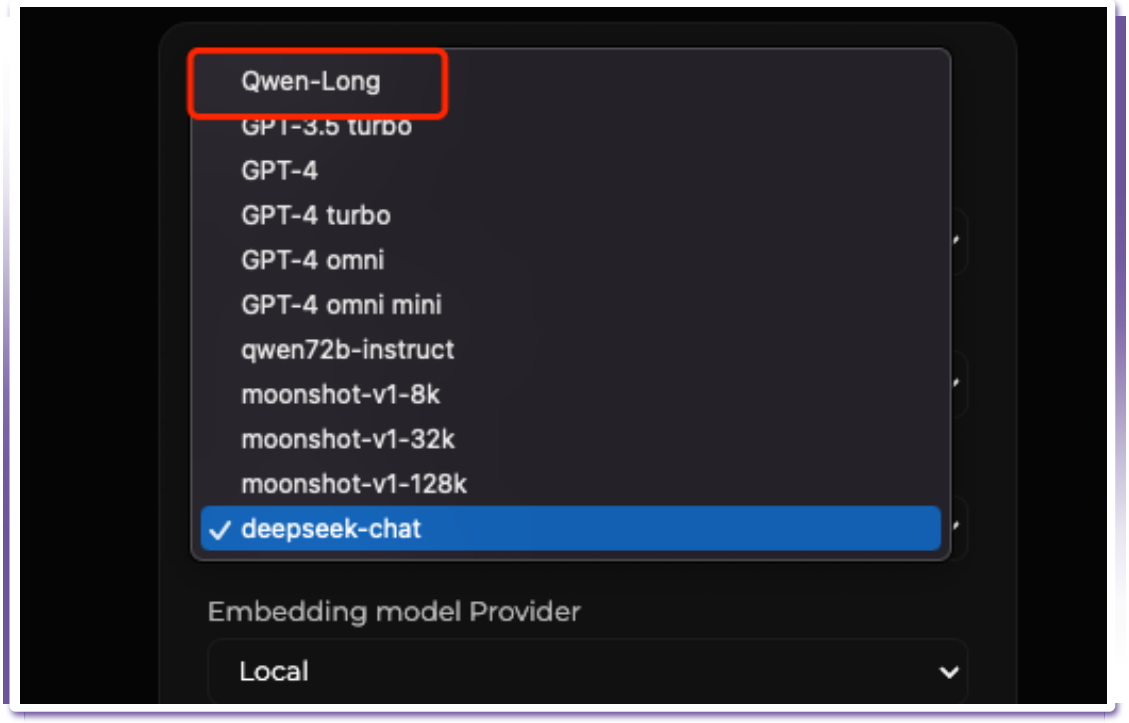
  1. `modelName`：为 oneapi 中模型的名称；
  2. `displayName`：为显示在前端中的模型名称；
  3. `baseURL`: 如果没有指定则默认为 openai 自己的 url，且这时 api-key 必须是原生的 openai api key。

  如果是在容器中更改，需要编译

> npm run build

  编译后需要重启

> docker-compose restart



1.4.3 增加oneapi中的模型使用

- **东京机 oneapi**

> http://38.54.50.54:8025/

`令牌名称`： Perplexica专用

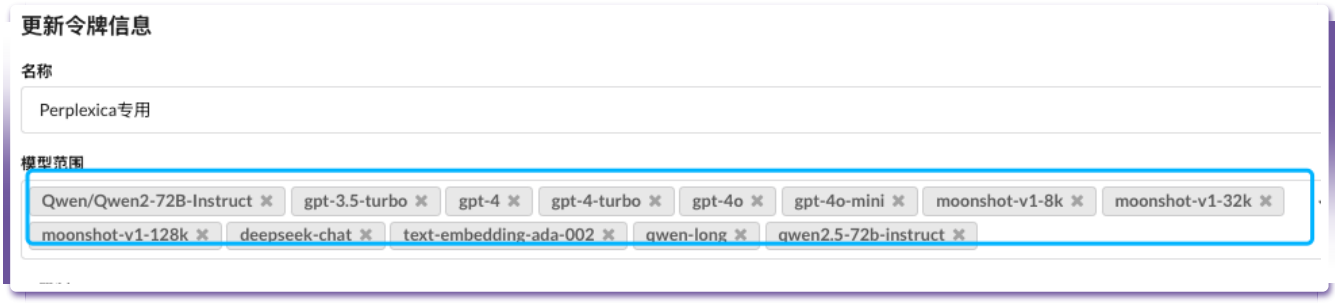
`base_url`

> http://38.54.50.54:8025/v1

`api-key`

> sk-PJrYIlj1QwodFT4n00Df8956B465411389C44dF11eDc07Da

自行增减 模型



- `openai.ts` 中配置 oneapi 的模型

> /src/lib/providers/openai.ts

在`chatModels`下增加

注意如果是在容器中修改完成后记得编译完成后重启

> npm run build

```
        'qwen-long': {
          displayName: 'Qwen-Long',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'qwen-long',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-3.5-turbo': {
          displayName: 'GPT-3.5 Turbo',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-3.5-turbo',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4': {
          displayName: 'GPT-4',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-4',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'gpt-4-turbo': {
          displayName: 'GPT-4 turbo',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'gpt-4-turbo',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
```

```
      }),
    },
    'gpt-4o': {
      displayName: 'GPT-4 omni',
      model: new ChatOpenAI({
        openAIApiKey,
        modelName: 'gpt-4o',
        temperature: 0.7,
      },{
        baseURL: getOpenaiBaseUrl()
      }),
    },
    'gpt-4o-mini': {
      displayName: 'GPT-4 omni mini',
      model: new ChatOpenAI({
        openAIApiKey,
        modelName: 'gpt-4o-mini',
        temperature: 0.7,
      },{
        baseURL: getOpenaiBaseUrl()
      }),
    },
    'qwen2.5-72b-instruct': {
      displayName: 'qwen2.5-72b-instruct',
      model: new ChatOpenAI({
        openAIApiKey,
        modelName: 'qwen2.5-72b-instruct',
        temperature: 0.7,
      },{
        baseURL: getOpenaiBaseUrl()
      }),
    },
    'moonshot-v1-8k': {
      displayName: 'moonshot-v1-8k',
      model: new ChatOpenAI({
        openAIApiKey,
        modelName: 'moonshot-v1-8k',
        temperature: 0.7,
      },{
        baseURL: getOpenaiBaseUrl()
      }),
    },
    'moonshot-v1-32k': {
      displayName: 'moonshot-v1-32k',
      model: new ChatOpenAI({
        openAIApiKey,
        modelName: 'moonshot-v1-32k',
```

```
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'moonshot-v1-128k': {
          displayName: 'moonshot-v1-128k',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'moonshot-v1-128k',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
        'deepseek-chat': {
          displayName: 'deepseek-chat',
          model: new ChatOpenAI({
            openAIApiKey,
            modelName: 'deepseek-chat',
            temperature: 0.7,
          },{
            baseURL: getOpenaiBaseUrl()
          }),
        },
```

1.5 替换搜索部分

1.5.1 搜索

- 由美芝老师提供的知乎数据的搜索

`url`

> http://js1.blockelite.cn:19575/search

`请求体`

```
{
    "query":query,
    "datasetIds":["661f9c02e50601c600e26eb9"],
    "topN": 9,
    "simThreshold": 0.0
}
```

- python连接 示例代码

```python
import requests

"""知乎数据的相关搜索接口"""
url = 'http://js1.blockelite.cn:19575/search'
body = {
    "query":query,
    "datasetIds":["661f9c02e50601c600e26eb9"],
    "topN": 2,
    "simThreshold": 0.0
}
res = requests.post(url,json=body).json()
conv_links,content_list = res['conv_links'],res['content_list']
```

- typescript 连接示例代码

```typescript
import axios from 'axios';

const url = "http://js1.blockelite.cn:19575/search";
const payload = {
    query: "什么是保险",
    datasetIds: ["661f9c02e50601c600e26eb9"],
    topN: 2,
    simThreshold: 0
};

axios.post(url, payload)
  .then(response => {
    // 处理响应数据
    console.log("conv_links:",response.data.conv_links);
    console.log("content_list:",response.data.content_list);
  })
  .catch(error => {
    // 处理错误情况
    console.error(error);
  });
```

1.5.2 直接使用接口

- 直接使用美芝老师提供的接口，query 是 string 类型

1.5.2.1 测试用代码

- 文件名

search_use_zhihu.ts

```
npm install axios

tsc search_use_zhihu.ts

node search_use_zhihu.js
```

- 代码(topN =2 测试)

```
import axios from 'axios';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}

export const searchSearxng = async (
  query: string,
  opts?: SearxngSearchOptions,
) => {
  const url = "http://js1.blockelite.cn:19575/search";
  console.log('url', url);
  const payload = {
    query: query,
    datasetIds: ["661f9c02e50601c600e26eb9"],
    topN: 2,
    simThreshold: 0
  };

  try {
    const response = await axios.post(url, payload);
    const contentList = response.data.content_list;
```

```
    const convLinks = response.data.conv_links;

    const results: SearxngSearchResult[] = [];
    const suggestions: any[] = [];

    contentList.forEach((val, idx) => {
      const dataDict = {
        url: val.url,
        title: convLinks[idx].title,
        content: val.content,
        engine: 'bing',
        template: 'default.html',
        engines:['bing'],
        positions: [idx + 1],
        score: val.score,
        category:'general'
      };
      results.push(dataDict);
    });
    console.log('这里是结果:convLinks,', convLinks)
    return { results, suggestions };
  } catch (error) {
    console.error('这里出现错误:', error);
    throw error;
  }
};


// 调用 searchSearxng 函数
async function performSearch() {
  const query = '中国平安2023年业绩'; // 你的搜索查询
  const options: SearxngSearchOptions = {
      categories: ['general', 'news'], // 可选的搜索类别
      engines: ['bing', 'google'],       // 可选的搜索引擎
      language: 'zh-CN',       // 可选的搜索语言
      pageno: 1                // 可选的页码
  };

  try {
      const response = await searchSearxng(query, options);
      console.log('搜索结果:', response.results);
      console.log('response 的 type',typeof response);
      console.log('results 的 type',typeof response.results);
      console.log('results 的 type333',typeof response.suggestions);
  } catch (error) {
      console.error('搜索出错:', error);
  }
```

```
84  }
85
86
87  // 执行搜索
88  performSearch();
```

1.5.2.2 更改searxng.ts

- 如果使用了以下代码修改，则 config.toml 文件中的 SELFSEARCHURL填写

> SELFSEARCHURL = "http://js1.blockelite.cn:19575/search"

- 地址

> /src/lib/searxng.ts

原代码

```
1   import axios from 'axios';
2   import { getSearxngApiEndpoint } from '../config';
3
4   interface SearxngSearchOptions {
5     categories?: string[];
6     engines?: string[];
7     language?: string;
8     pageno?: number;
9   }
10
11  interface SearxngSearchResult {
12    title: string;
13    url: string;
14    img_src?: string;
15    thumbnail_src?: string;
16    thumbnail?: string;
17    content?: string;
18    author?: string;
19    iframe_src?: string;
20  }
21
22  export const searchSearxng = async (
23    query: string,
24    opts?: SearxngSearchOptions,
25  ) => {
26    const searxngURL = getSearxngApiEndpoint();
27
```

```
    const url = new URL(`${searxngURL}/search?format=json`);
    url.searchParams.append('q', query);

    if (opts) {
      Object.keys(opts).forEach((key) => {
        if (Array.isArray(opts[key])) {
          url.searchParams.append(key, opts[key].join(','));
          return;
        }
        url.searchParams.append(key, opts[key]);
      });
    }

    const res = await axios.get(url.toString());

    const results: SearxngSearchResult[] = res.data.results;
    const suggestions: string[] = res.data.suggestions;

    return { results, suggestions };
  };
```

修改后代码

```
  import axios from 'axios';
  import logger from '../utils/logger';
  import { getSelfSearchUrl, getPyPort } from '../config';

  interface SearxngSearchOptions {
    categories?: string[];
    engines?: string[];
    language?: string;
    pageno?: number;
  }

  interface SearxngSearchResult {
    title: string;
    url: string;
    img_src?: string;
    thumbnail_src?: string;
    thumbnail?: string;
    content?: string;
    author?: string;
    iframe_src?: string;
  }

  export const searchSearxng = async (
    query: string | string[],
```

```
  opts?: SearxngSearchOptions,
) => {
  if (typeof query === 'string') {
    // 如果 query 是字符串类型，将其转换为单元素数组
    if (query.startsWith('[') && query.endsWith(']')) query = JSON.parse(query);
    else query = [query];
  } else if (!Array.isArray(query)) {
    // 如果 query 既不是字符串也不是字符串数组，抛出一个错误。
    throw new Error('query必须是 【字符串 】或者 【字符串数组】！！！！！');
  }

  // const url = "http://localhost:8013/retrieval";
  const urlroot = getSelfSearchUrl();
  const pyport = getPyPort();
  const url = urlroot + ':' + pyport + '/retrieval';

  logger.info(`python 服务检索的url: ${url}`);
  const payload = {
    query_list: query,
    opts: opts
  };
  logger.info(`python 服务检索的请求体: ${JSON.stringify(payload, null, 2)}`);

  try {
    const response = await axios.post(url, payload);
    const results = response.data.results;
    const suggestions = response.data.suggestions;
    const queryList = query;
    logger.info(`检索: ${suggestions}`);
    return { results, suggestions, queryList };
  } catch (error) {
    logger.error('这里出现错误:', error);
    throw error;
  }
};

// // 调用 searchSearxng 函数
// async function performSearch() {
//   const query = '中国平安2023年业绩'; // 你的搜索查询
//   const options: SearxngSearchOptions = {
//       categories: ['general', 'news'], // 可选的搜索类别
//       engines: ['bing', 'google'],       // 可选的搜索引擎
//       language: 'zh-CN',       // 可选的搜索语言
//       pageno: 1                // 可选的页码
```

```
//   };

//   try {
//       const response = await searchSearxng(query, options);
//       console.log('搜索结果:', response.results);
//       console.log('response 的 type',typeof response);
//       console.log('results 的 type',typeof response.results);
//       console.log('results 的 type333',typeof
    response.suggestions);
//   } catch (error) {
//       console.error('搜索出错:', error);
//   }
// }

// // 执行搜索
// performSearch();

```

1.5.3 使用 python 启动的接口

- 使用 python 启动的接口，也是调用美芝老师的接口，不同的是 query 是 string或者list 类型，其中 query 进入后会做判断，如果是 string 类型，则转换为 list 类型

1.5.3.1 测试用代码

- 文件名

> search_use_python.ts

```
npm install axios

tsc search_use_python.ts

node search_use_python.js
```

- 代码

```
import axios from 'axios';

interface SearxngSearchOptions {
  categories?: string[];
  engines?: string[];
  language?: string;
  pageno?: number;
}
```

```
9
10  interface SearxngSearchResult {
11    title: string;
12    url: string;
13    img_src?: string;
14    thumbnail_src?: string;
15    thumbnail?: string;
16    content?: string;
17    author?: string;
18    iframe_src?: string;
19  }
20
21
22  export const searchSearxng = async (
23    query: string | string[],
24    opts?: SearxngSearchOptions,
25  ) => {
26    if (typeof query === "string") { // 如果 query 是字符串类型，将其转换
    为单元素数组
27      query = [query];
28    } else if (!Array.isArray(query)) { // 如果 query 既不是字符串也不是
    字符串数组，抛出一个错误。
29      throw new Error("query必须是 【字符串 】或者 【字符串数
    组】！！！！！ ");
30    }
31    const urlroot = "http://localhost:8013"
32    const url = urlroot + "/retrieval";
33
34    console.log('url', url);
35    const payload = {
36      query_list:query
37    };
38
39    try {
40      const response = await axios.post(url, payload);
41      const results = response.data.results;
42      const suggestions = response.data.suggestions;
43      return { results, suggestions };
44    } catch (error) {
45      console.error('这里出现错误:', error);
46      throw error;
47    }
48  };
49
50  // 调用 searchSearxng 函数
51  async function performSearch() {
52    const query = '中国平安2023年业绩'; // 你的搜索查询
```

```
    const options: SearxngSearchOptions = {
        categories: ['general', 'news'], // 可选的搜索类别
        engines: ['bing', 'google'],        // 可选的搜索引擎
        language: 'zh-CN',         // 可选的搜索语言
        pageno: 1                  // 可选的页码
    };

    try {
        const response = await searchSearxng(query, options);
        console.log('搜索结果:', response.results);
        console.log('response 的 type',typeof response);
        console.log('results 的 type',typeof response.results);
        console.log('results 的 type333',typeof response.suggestions);
    } catch (error) {
        console.error('搜索出错:', error);
    }
}


// 执行搜索
performSearch();
```

1.5.3.2 更改searxng.ts

- 如果使用了以下代码修改，则 config.toml 文件中的 SELFSEARCHURL和 PYPORT填写

> SELFSEARCHURL = 服务器 ip
>
> PYPORT = python 后端服务的地址

既使用 python 启动的后端服务 的接口，后续在代码中会转换成

"http://${SELFSEARCHURL}:${PYPORT} /retrieval"的检索接口

- 地址

> /src/lib/searxng.ts

原代码:同上

修改后

```
import axios from 'axios';
import { getSelfSearchUrl } from '../config';

interface SearxngSearchOptions {
  categories?: string[];
```

```
  engines?: string[];
  language?: string;
  pageno?: number;
}

interface SearxngSearchResult {
  title: string;
  url: string;
  img_src?: string;
  thumbnail_src?: string;
  thumbnail?: string;
  content?: string;
  author?: string;
  iframe_src?: string;
}


export const searchSearxng = async (
  query: string | string[],
  opts?: SearxngSearchOptions,
) => {
  if (typeof query === "string") { // 如果 query 是字符串类型，将其转换为单元素数组
    query = [query];
  } else if (!Array.isArray(query)) { // 如果 query 既不是字符串也不是字符串数组，抛出一个错误。
    throw new Error("query必须是 【字符串 】或者 【字符串数组】！！！！！");
  }

  // const url = "http://localhost:8013/retrieval";
  const urlroot = getSelfSearchUrl();
  const url = urlroot + "/retrieval";

  console.log('python 服务检索的url: ', url);
  const payload = {
    query_list:query
  };

  try {
    const response = await axios.post(url, payload);
    const results = response.data.results;
    const suggestions = response.data.suggestions;
    return { results, suggestions };
  } catch (error) {
    console.error('这里出现错误:', error);
    throw error;
```

```
  }
};

// // 调用 searchSearxng 函数
// async function performSearch() {
//   const query = '中国平安2023年业绩'; // 你的搜索查询
//   const options: SearxngSearchOptions = {
//       categories: ['general', 'news'], // 可选的搜索类别
//       engines: ['bing', 'google'],       // 可选的搜索引擎
//       language: 'zh-CN',       // 可选的搜索语言
//       pageno: 1                // 可选的页码
//   };

//   try {
//       const response = await searchSearxng(query, options);
//       console.log('搜索结果:', response.results);
//       console.log('response 的 type',typeof response);
//       console.log('results 的 type',typeof response.results);
//       console.log('results 的 type333',typeof
response.suggestions);
//   } catch (error) {
//       console.error('搜索出错:', error);
//   }
// }


// // 执行搜索
// performSearch();
```

1.5.4 更改 docker-compose.yaml 文件

1.5.4.1 删除 Searxng

- 原 Perplexica 搜索部分是 SearXNG，这个需要启动一个独立的容器，现在将搜索部分替换后，直接将 SearXNG 部分给删除（注释掉）

```
  # searxng:
  #   image: docker.io/searxng/searxng:latest
  #   volumes:
  #     - ./searxng:/etc/searxng:rw
  #   ports:
  #     - 4000:8080
```

```
#   networks:
#     - perplexica-network
#   restart: unless-stopped



    # depends_on:
  #   - searxng
```

1.6 增加python服务

1.6.1 代码

- 地址

> /python_src

- build 镜像

> docker build -t perplexica-rag:1.0 . -f py.dockerfile

- 启动测试

> docker run -d -p 8012:8012 --name perplexica-rag perplexica-rag:1.0

1.6.2 增加到docker-compose.yaml

- 启动一个独立的python容器

```
perplexica-rag:
  build:
    context: .
    dockerfile: py.dockerfile
  ports:
    - 8013:8013
  volumes:
    - backend-dbstore:/home/perplexica/py_config.toml
    - ./config.toml:/home/perplexica/config.toml
  networks:
    - perplexica-network
  restart: unless-stopped
```

1.7 接口相关

1.7.1 图片视频接口返回

1.7.1.1 输入值

- 用户 query

1.7.1.2 返回值

- **results**：由多个字典组成的列表

  每个字典需要包含的字段原代码

```
    title: string;
    url: string;
    img_src: string;
    thumbnail_src: string;
    thumbnail: string;
    content: string;
    author: string;
    iframe_src: string;
```

- **suggestions**:默认返回空列表即可
- 备注：

  图片接口的请求参数中的 `engines: ['bing images', 'google images']`

  视频接口的请求参数中的 `engines: ['youtube']`

1.7.1.3 图片接口

- 原代码

```
      RunnableLambda.from(async (input: string) => {
        const res = await searchSearxng(input, {
          engines: ['bing images', 'google images'],
        });//使用 searchSearxng 函数执行搜索，指定搜索引擎为 'bing images' 和 'google images'。

        const images = [];

        res.results.forEach((result) => {//遍历搜索结果，筛选出包含图片源 (img_src)、链接 (url) 和标题 (title) 的结果。
          if (result.img_src && result.url && result.title) {
```

```
            images.push({
              img_src: result.img_src,
              url: result.url,
              title: result.title,
            });
          }
        });

        return images.slice(0, 10);// 返回前 10个结果
      }),
```

results 中需要包含的字段示例(两条)：

```
[
  {
    "template":"images.html",
    "url": "https://www.baogaoting.com/info/494035",
    "thumbnail_src": "https://tse1.mm.bing.net/th?id=OIP.xWvO2Z4yyAQg2rNYt3QAkAHaEK&pid=15.1",
    "img_src": "https://imgcdn.baogaoting.com/PDFImage/2023/08/31/cbba8ccc6b91467f9add74be3ca5fbfd.jpg",
    "title": "中国平安2023年中期业绩报告-48页_报告-报告厅",
    "engine": "bing images",
    "engines": [
      "bing images"
    ],
    "positions": [
      1
    ],
    "score": 1.0,
    "category": "images"
  },
  {
    "template": "images.html",
    "url": "https://www.deviantart.com/blinchikart/art/xiaomi-mira-968479829",
    "thumbnail_src": "https://222.png",
    "img_src": "https://222.png",
```

```
    "title": "xiaomi mira by BlinchikART, visual art",
    "engines": [
      "deviantart"
    ],
    "positions": [
      1
    ],
    "score": 1.0,
    "category": "images"
  },
]
```


1.7.1.4 视频接口

- 原代码

```
    RunnableLambda.from(async (input: string) => {
      const res = await searchSearxng(input, {
        engines: ['youtube'],
      });

      const videos = [];

      res.results.forEach((result) => {
        if (
          result.thumbnail &&
          result.url &&
          result.title &&
          result.iframe_src
        ) {
          videos.push({
            img_src: result.thumbnail,
            url: result.url,
            title: result.title,
            iframe_src: result.iframe_src,
          });
        }
      });

      return videos.slice(0, 10);
    })
```

results 中需要包含的字段示例(两条)：(必须包含 url,title,img_src,iframe_src)


```
[
    {
      "template": "videos.html",
      "url": "https://www.youtube.com/watch?v=np2SOHeGpCA",
      "title": "中國歷史24個朝代，每個朝代滅亡原因是什麼？",
      "content": "",
      "author": "歷史面面觀",
      "length": "30:42",
      "iframe_src": "https://www.youtube-nocookie.com/embed/np2SOHeGpCA",
      "engine": "youtube",
      "engines": [
        "youtube"
      ],
      "positions": [
        1
      ],
      "score": 1.0,
      "category": "videos"
    },
    {
      "url": "https://www.youtube.com/watch?v=mF3I8nKlPFc",
      "title": "快速有趣地看完中国历史，不同朝代和著名人物 | 中国历史时间轴 | 简懂笔记",
      "content": "",
      "author": "简懂笔记",
      "length": "9:15",
      "template": "videos.html",
      "iframe_src": "https://www.youtube-nocookie.com/embed/mF3I8nKlPFc",
      "thumbnail": "https://i.ytimg.com/vi/mF3I8nKlPFc/hqdefault.jpg",
      "engine": "youtube",
      "engines": [
```

```
            "youtube"
        ],
        "positions": [
            2
        ],
        "score": 0.5,
        "category": "videos"
    }]
```

1.7.1.5 东京机上的 searxng 测试(服务不一定开了)

- curl

```
curl --location --request POST 'http://38.54.50.54:4000/search?format=json&q=中国的历史朝代&categories=images&engines=bing%20images&language=zh-CN&pageno=1'
```

1.7.2 python 服务接口

- helath

```
curl --location 'http://localhost:8013/health'
```

- 改写接口

```
curl --location 'http://localhost:8013/rewrite_query' \
--header 'Content-Type: application/json' \
--data '{
    "query":"24年的大学生就业情况怎么样"
}'
```

- 检索接口（）

```
curl --location 'http://localhost:8013/retrieval' \
--header 'Content-Type: application/json' \
--data '{
    "query_list": ["24年的大学生就业市场是什么样的", "24年大学生就业的主要行业和岗位有哪些", "如何提高24年大学生的就业竞争力"]
}
'
```

- 改写检索 websocket 接口（这个接口是将改写和检索合并，并用 websocket 写的）

```
ws://localhost:8013/rewrite_retrieval


{
    "query":"什么是保险"
}
```

测试：

1. 安装 wscat

   ```
   sudo npm install -g wscat
   ```

2. 测试

   使用 `wscat` 连接到 WebSocket 服务器

   ```
   ws://localhost:8013/rewrite_retrieval
   ```

   连接成功后，你可以直接输入 JSON 消息并发送：

   ```
   {
       "query":"什么是保险"
   }
   ```

1.7.3 Perplexica 自带的后端接口

- health

```
curl --location 'http://127.0.0.1:3001/api'
```

- 获取 config（包含 config.toml 中的内容以及 mode列表）

```
curl --location 'http://127.0.0.1:3001/api/config'
```

- 更改 config (可更改 config.tom 中的内容)

```
curl --location 'http://127.0.0.1:3001/api/config' \
--header 'Content-Type: application/json' \
--data '{
  "groqApiKey": "11111"
}'
```

- 获取所有聊天

```
curl --location 'http://127.0.0.1:3001/api/chats'
```

- 获取已经配置的 model 和 embedding model

```
curl --location 'http://127.0.0.1:3001/api/models'
```

- 根据 chatid 获取 chat (每个页面的所有聊天),chatid 可从获取所有聊天中找到

```
curl --location 'http://127.0.0.1:3001/api/chats/${chatid}'
```

- 根据聊天历史生成建议，聊天历史可从上面找到

```
curl --location 'http://127.0.0.1:3001/api/suggestions' \
--header 'Content-Type: application/json' \
--data '{
    "chatHistory": [
        {
            "id": 16,
            "content": "什么是保险",
            "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",
            "messageId": "033eee2ea15ddc",
            "role": "user",
            "metadata": "{\"createdAt\":\"2024-11-28T01:26:35.194Z\"}"
        },
        {
            "id": 17,
            "content": "保险是一种金融工具，旨在为个人或企业提供财务保护，以应对意外损失。以下是关于保险的详细解释：",
            "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",
            "messageId": "9266de787dab70",
            "role": "assistant",
            "metadata": ""
        }
    ],
    "chatModel": { "provider": "openai", "model": "qwen-long" }
    }'
```

- discover 页面

```
curl --location 'http://127.0.0.1:3001/api/discover'
```

1.7.4 修改后，搜索服务的请求体

- 普通搜索

```
{
  "query_list": ["什么是保险"],
  "opts": {
    "language": "en"
  }
}
```

- 图片

```
 {
  "query_list": ["保险是什么"],
  "opts": {
    "engines": ["bing images","google images"]
  }
}
```

- 视频

```
{
  "query_list": ["Rephrased question: 什么是保险? 如何运作? "],
  "opts": {
    "engines": ["youtube"]
  }
}
```

- discovery

```
{
  "query_list": ["site:yahoo.com tech"],
  "opts": {
    "engines": ["bing news"],
    "pageno": 1
  }
}
```

1.8 其他

1.8.1 urlencoder特殊例子（已经解决）

```
# 第一个例子，编码前 File notfound,编码后可以
http://js1.blockelite.cn:19574/display/保险营销_外行人怎么买保险不被坑?_外行人买保险防坑指南：如何高效选择与自学_1566578444.json
```

```
http://js1.blockelite.cn:19574/display/%E4%BF%9D%E9%99%A9%E8%90%A5%E9%94%80_%E5%A4%96%E8%A1%8C%E4%BA%BA%E6%80%8E%E4%B9%88%E4%B9%B0%E4%BF%9D%E9%99%A9%E4%B8%8D%E8%A2%AB%E5%9D%91%3F_%E5%A4%96%E8%A1%8C%E4%BA%BA%E4%B9%B0%E4%BF%9D%E9%99%A9%E9%98%B2%E5%9D%91%E6%8C%87%E5%8D%97%EF%BC%9A%E5%A6%82%E4%BD%95%E9%AB%98%E6%95%88%E9%80%89%E6%8B%A9%E4%B8%8E%E8%87%AA%E5%AD%A6_1566578444.json

from urllib.parse import quote
s = "http://js1.blockelite.cn:19574/display/保险营销_外行人怎么买保险不被坑?_外行人买保险防坑指南：如何高效选择与自学_1566578444.json"
encoded_url =  quote(s,safe = '/:') # quote(s, safe='/:?=&')


# 第二个例子，编码前 File notfound,编码后可以
http://js1.blockelite.cn:19574/display/奥巴马医改\n_美国通过的医改法（奥巴马医改方案）是什么_奥巴马医改方案：全面解析与两党分歧_14753021.json
http://js1.blockelite.cn:19574/display/%E5%A5%A5%E5%B7%B4%E9%A9%AC%E5%8C%BB%E6%94%B9%0A_%E7%BE%8E%E5%9B%BD%E9%80%9A%E8%BF%87%E7%9A%84%E5%8C%BB%E6%94%B9%E6%B3%95%EF%BC%88%E5%A5%A5%E5%B7%B4%E9%A9%AC%E5%8C%BB%E6%94%B9%E6%96%B9%E6%A1%88%EF%BC%89%E6%98%AF%E4%BB%80%E4%B9%88_%E5%A5%A5%E5%B7%B4%E9%A9%AC%E5%8C%BB%E6%94%B9%E6%96%B9%E6%A1%88%EF%BC%9A%E5%85%A8%E9%9D%A2%E8%A7%A3%E6%9E%90%E4%B8%8E%E4%B8%A4%E5%85%9A%E5%88%86%E6%AD%A7_14753021.json

from urllib.parse import quote
s = "http://js1.blockelite.cn:19574/display/奥巴马医改\n_美国通过的医改法（奥巴马医改方案）是什么_奥巴马医改方案：全面解析与两党分歧_14753021.json"
encoded_url =  quote(s,safe = '/:') # quote(s, safe='/:?=&')


# 第三个例子，
http://js1.blockelite.cn:19574/display/保险理赔\n_看过了19年的54份理赔报告，我知道为什_揭秘保险行业真相：重疾险保额不足，消费者如何自保？_128021819.json


from urllib.parse import quote
s = "http://js1.blockelite.cn:19574/display/保险理赔\n_看过了19年的54份理赔报告，我知道为什_揭秘保险行业真相：重疾险保额不足，消费者如何自保?_128021819.json"
encoded_url =  quote(s,safe = '/:') # quote(s, safe='/:?=&')

```

1.8.2 阿里云上 yaml 文件

- yaml 文件

```
services:
```

```
  perplexica-backend:
    image: itzcrazykns1337/perplexica-backend:latest
    ports:
      - "8012:3001"
    volumes:
      - backend-dbstore:/home/perplexica/data
      - uploads:/home/perplexica/uploads
      - ./config.toml:/home/perplexica/config.toml
    extra_hosts:
      - "host.docker.internal:host-gateway"
    networks:
      - perplexica-network
    restart: unless-stopped

  perplexica-frontend:
    image: itzcrazykns1337/perplexica-frontend:latest
    environment:
      - NEXT_PUBLIC_API_URL=http://121.41.60.137:8012/api
      - NEXT_PUBLIC_WS_URL=ws://121.41.60.137:8012
      - NEXT_PUBLIC_PY_API=http://121.41.60.137
      - NEXT_PUBLIC_PYWS_API=ws://121.41.60.137
      - NEXT_PUBLIC_PY_PORT=8013
      - NEXT_PUBLIC_ADMIN=user
      - NEXT_PUBLIC_PASSWORD=Pafc?2025
      - NEXT_PUBLIC_VERIFYTOKEN=d27121560bdb4d9d8a780cb24bf9a399
    depends_on:
      - perplexica-backend
    ports:
      - "8345:3000"
    networks:
      - perplexica-network
    restart: unless-stopped

  perplexica-rag:
    image: itzcrazykns1337/perplexica-rag:latest
    ports:
      - "8013:8013"
    volumes:
      - ./config.toml:/home/perplexica/config.toml
    networks:
      - perplexica-network
    restart: unless-stopped

networks:
  perplexica-network:

volumes:
```

```
49      backend-dbstore:
50      uploads:
51
```

1.8.3 阿里云上容器中修改后提交

- 代码

  如有需要，重新提交

```
1   docker commit
    e28052ce34dfbe824e8cf4a9ad6886af9674f9b9a1f06b1a88a091cc685cd501
    itzcrazykns1337/perplexica-rag:latest
2
3   docker commit
    18c26e0d218f7447d504a5e6eaf01119776bb4158d176ab8f7c652131460593a
    itzcrazykns1337/perplexica-backend:latest
4
5   docker commit
    cc330b3deb2ba6503681d279995942a2f97ed18e8193e3eff259b4f5ba9edb2b
    itzcrazykns1337/perplexica-frontend:latest
```

2 后续的修改记录

2.1 suggestions

- 调用路径
  1. 后端请求路径

     src/agents/suggestionGeneratorAgent.ts（export generateSuggestions) --->src/routes/suggestions.ts(export router)

     使用，访问地址 ip+ /api/suggestions

  2. 前端请求

     ui/lib/actions.ts (chatHistory 、getSuggestions) --> ui/components/ChatWindow.tsx (getSuggstions)

     注意，ChatWindow.tsx 中messagesRef.current 记录的是当前所有的详细聊天信息

- 更改为最近两轮

  ui/lib/actions.ts

```
    //定义一个变量并将 chatHisory 赋值给它
    const history = chatHisory;
    console.log('原始聊天历史:', JSON.stringify(history, null, 2));
    //调用 removeSourcesField 函数，删除每个 Message 对象的 sources 字段
    const source_deleted_history = removeSourcesField(history);
    console.log('删除 sources 字段后的聊天历史:', JSON.stringify(source_deleted_history, null, 2));
    //截取最后4条消息(最近两轮)
    const lastTwoHistory = source_deleted_history.slice(-4);
    console.log('最近两轮的聊天历史:', JSON.stringify(lastTwoHistory, null, 2));
```

2.2 多轮对话

- 调用路径
  1. 前端： ui/components/ChatWindow.tsx

     sendMessage 中通过 websocket传播

  2. 后端：

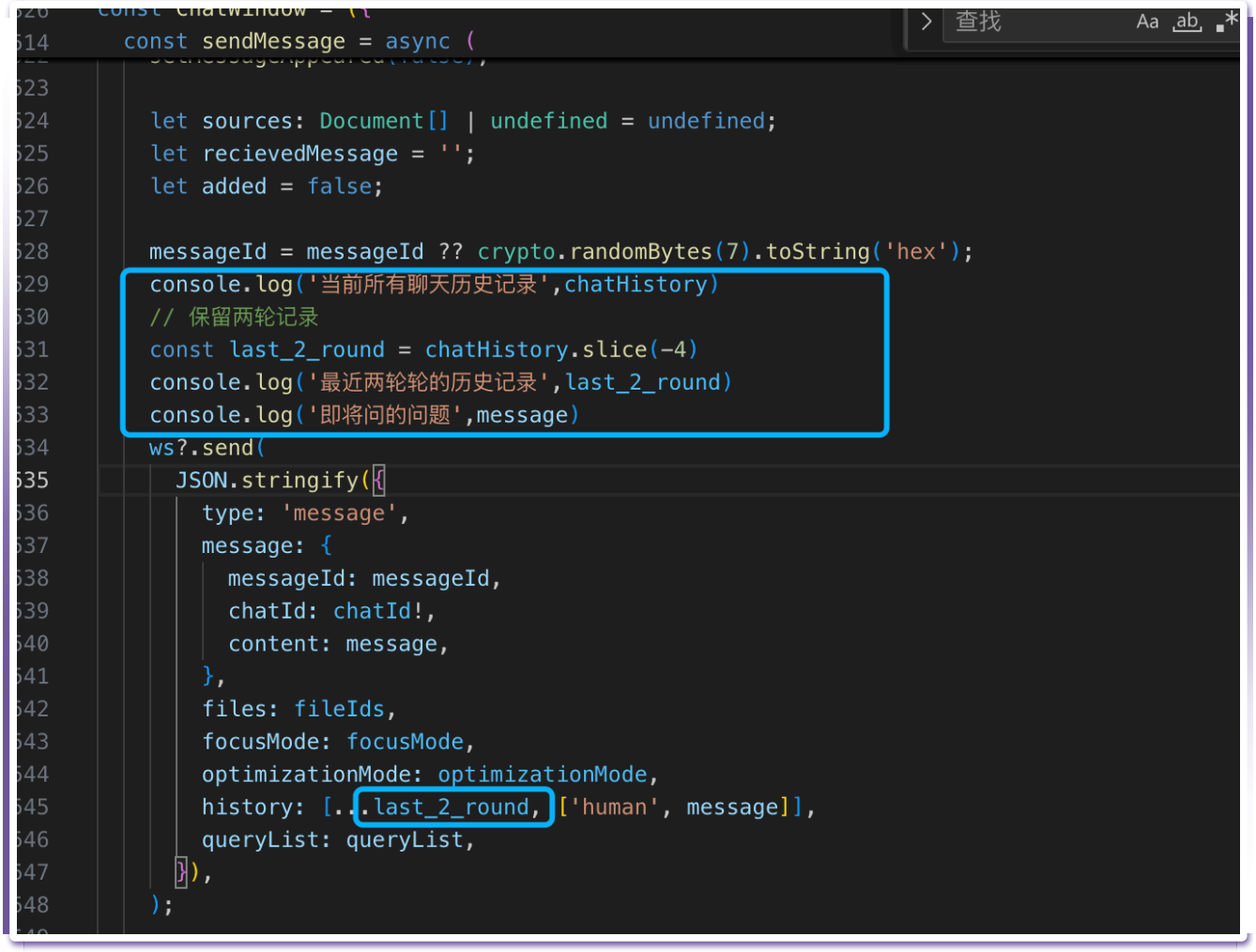
     src/websocket/messageHandler.ts

接收前端 ws 的消息并处理

- 更改为只用最近两轮对话

  ui/components/ChatWindow.tsx

```
      messageId = messageId ?? crypto.randomBytes(7).toString('hex');
      console.log('当前所有聊天历史记录',chatHistory)
      // 保留两轮记录
      const last_2_round = chatHistory.slice(-4)
      console.log('最近两轮轮的历史记录',last_2_round)
      console.log('即将问的问题',message)
      ws?.send(
        JSON.stringify({
          type: 'message',
          message: {
            messageId: messageId,
            chatId: chatId!,
            content: message,
          },
          files: fileIds,
          focusMode: focusMode,
          optimizationMode: optimizationMode,
          history: [...last_2_round, ['human', message]],
          queryList: queryList,
        }),
      );
```



```
const sendMessage = async (

  let sources: Document[] | undefined = undefined;
  let recievedMessage = '';
  let added = false;

  messageId = messageId ?? crypto.randomBytes(7).toString('hex');
  console.log('当前所有聊天历史记录',chatHistory)
  // 保留两轮记录
  const last_2_round = chatHistory.slice(-4)
  console.log('最近两轮轮的历史记录',last_2_round)
  console.log('即将问的问题',message)
  ws?.send(
    JSON.stringify({
      type: 'message',
      message: {
        messageId: messageId,
        chatId: chatId!,
        content: message,
      },
      files: fileIds,
      focusMode: focusMode,
      optimizationMode: optimizationMode,
      history: [...last_2_round, ['human', message]],
      queryList: queryList,
    }),
  );
```

2.3 修改记录

- 2025-01-13
  1. `鉴权`：
     1. 前端：新增登录页面
     2. 后端：实现api鉴权功能。
  2. `优化前端页面`：
     1. **页面加载速度优化**：
        - 优化前：平均等待时长为 10 秒。
        - 优化后：平均等待时长缩短至 3 秒。
     2. **问题展示**：
        - 将 3 个改写的问题按先后顺序优先展示。
     3. **页面锁定**：
        - 在模型生成过程中，页面不再自动下拉，提升用户体验。

3. `优化传入历史问题`

    1. **问题建议**：

        - **问题**：传入后端时包含 `source` 字段（针对当前问题的所有检索记录），但该字段实际并未被使用，导致在第三轮聊天后（一问一答为一轮），请求体超过100KB，无法生成后续问题建议。
        - **修改**：去除 `source` 字段，仅保留最近两轮聊天记录。(在模型 `max_token` 允许的条件下，最多可支持 20 轮左右的聊天)

    2. **多轮聊天**：

        问题：多轮聊天传入所有聊天历史，随着轮次增加，可能导致以下问题： token 消耗过大、聊天历史被截断、模型性能下降、模型产生幻觉甚至直接报错等问题。

        修改：仅保留最近两轮聊天记录。

4. `文档上传`

    原代码中当前文档上传功能做了三件事

    1. **文档加载与保存**：

        - 根据文件类型（PDF、DOCX、TXT）加载文档内容，并保存到固定文件夹（`uploads`）中。

    2. **文档分块处理**：

        - 对原文档进行分块处理，分块后保存为 JSON 文件，并存放在 `uploads` 文件夹中。

    3. **生成嵌入（Embedding）**：

        - 使用嵌入模型为分块后的文本生成嵌入（Embedding），并将结果保存为 JSON 文件，存放在 `uploads` 文件夹中。

    暂时未看到对这些文档的使用。