

1 Perplexica

1.1 相关地址

1.2 其他修改

1.2.1 更改 dockerfile

1.2.2 更改yaml文件

1.2.3 更改图标

1.2.4 更改后端配置文件

1.3 增加配置项(config.toml)

1.3.1 更改 src/config.ts

1.3.2 更改 routes/config.ts

1.3.3 更改 SettingsDialog.tsx

1.4 增加 models

1.4.1 修改 oneapi 的 baseurl

1.4.2 增加模型(以增加 qwen-long 为例子)

1.4.3 增加oneapi中的模型使用

1.5 替换搜索部分

1.5.1 搜索

1.5.2 直接使用接口

1.5.3 使用 python 启动的接口

1.5.4 更改 docker-compose.yaml 文件

1.6 接口相关

1.6.1 图片视频接口返回

1.6.2 python 服务

1.6.3 Perplexica 自带的后端接口

1 Perplexica

1.1 相关地址

- 原 `github` 地址

```
https://github.com/ltzCrazyKns/Perplexica
```

- 更改后的（自己的）

```
https://github.com/qinchihongye/Perplexica
```

- 一直刷新处理(issue)

```
https://github.com/ltzCrazyKns/Perplexica/issues/180
```

- 新增 oneapi（功能不完整）

```
https://github.com/ltzCrazyKns/Perplexica/issues/367
```

1.2 其他修改

1.2.1 更改 dockerfile

- 更改后端 dockerfile

```
backend.dockerfile
```

增加

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

1.2.2 更改yaml文件

- 更改IP 地址

```
1  args:
2    - NEXT_PUBLIC_API_URL=http://127.0.0.1:3001/api
3    - NEXT_PUBLIC_WS_URL=ws://127.0.0.1:3001
```

将 `127.0.0.1` 替换为运行 `Perplexica` 服务器的 IP 地址

- 更改前端端口

```
1     ports:
2       - 3000:3000
3
4     ports:
5       - 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）

```
1  src={`https://s2.googleusercontent.com/s2/favicons?
   domain_url=${source.metadata.url}`}
2
3  src='./favicon.ico'
```

1.2.4 更改后端配置文件

- 配置文件地址

```
/config.toml
```

```
1  [API_KEYS]
2  OPENAI = "sk-PJrYIIj1QwodFT4n00Df8956B465411389C44dF11eDc07Da"
3  GROQ = ""
4  ANTHROPIC = ""
5  GEMINI = ""
6
7  [API_ENDPOINTS]
8  OLLAMA = "http://host.docker.internal:11434"
9  OPENAIBASEURL = "http://38.54.50.54:8025/v1"
10 SELFSEARCHURL = "http://host.docker.internal"
11 SEARXNG = "http://localhost:32768"
12
```

```
13  [GENERAL]
14  PORT = 3_001
15  SIMILARITY_MEASURE = "cosine"
16  KEEP_ALIVE = "5m"
17  PYPORT = 8_013
18  LLM_NAME = "qwen2.5-72b-instruct"
19  TIMEOUT = 5
20  RETRIEVALURL = "http://js1.blockelite.cn:19575/search"
```

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

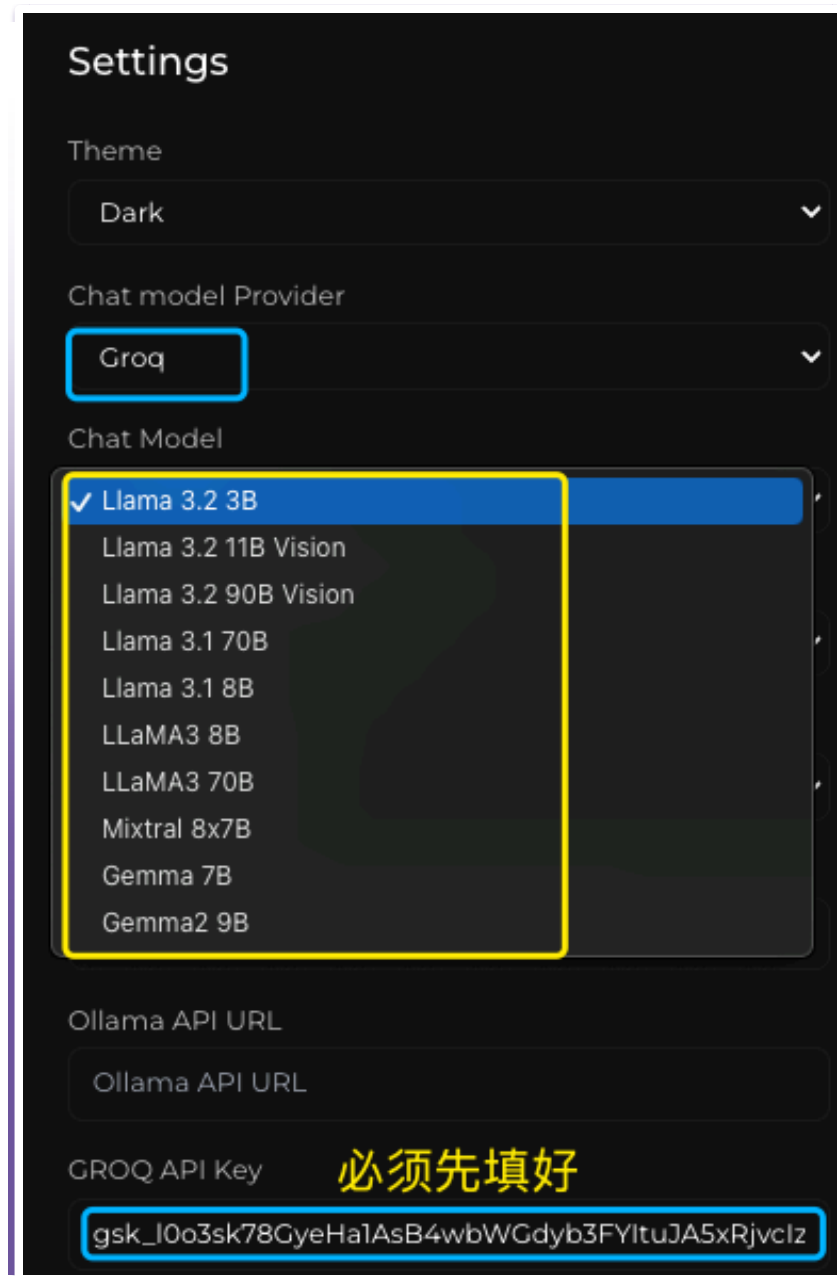
说明如下：

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
-

1.3 增加配置项(config.toml)

- 以增加 `OPENAIBASEURL` 、 `SELFSEARCHURL` 、 `PYPORT` 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 =
```

- 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。
 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

```
1 OPENAIBASEURL: string;
2 SELFSEARCHURL: string;
```

GENERAL下增加

```
1 PYPORT: number;
```

- 增加导出函数

```
1 export const getOpenaiBaseUrl = () =>
  loadConfig().API_ENDPOINTS.OPENAIBASEURL;
2
3 export const getSelfSearchUrl = () =>
  loadConfig().API_ENDPOINTS.SELFSEARCHURL;
4
5 export const getPyPort = () => loadConfig().GENERAL.PYPORT;
```

1.3.2 更改 routes/config.ts

- 地址

```
/src/routes/config.ts
```

- 新增

```
1 import {getOpenaiBaseUrl} from '../config';
2 import {getSelfSearchUrl} from '../config';
3
4 //在 60 行加入
5 config['openaiBaseUrl'] = getOpenaiBaseUrl();
6
7 //在 61 行加入
8 config['selfSearchUrl'] = getSelfSearchUrl();
9
10 //在 updatedConfig 中API_ENDPOINTS下加入
11 OPENAIBASEURL: config.openaiBaseUrl,
12 SELFSEARCHURL: config.selfSearchUrl,
```

1.3.3 更改 SettingsDialog.tsx

- 地址

```
/ui/components/SettingsDialog.tsx
```

- SettingsType 类型中增加

```

1  openaiBaseUrl: string;
2
3  selfSearchUrl: string;

```

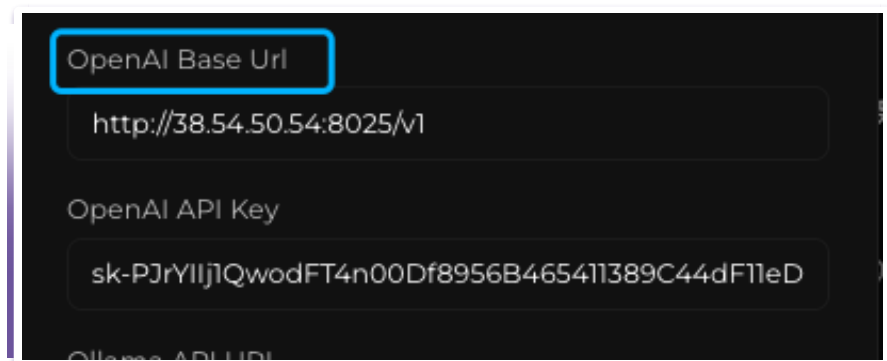
- **Return** 中增加 (对应网页的设置页面, 找个合适的位置)

1. 放在 OpenAI API Key 上面

```

1          <div className="flex flex-col space-y-1">
2              <p className="text-black/70 dark:text-white/70
text-sm">
3                  OpenAI Base Url
4              </p>
5              <Input
6                  type="text"
7                  placeholder="OpenAI Base Url"
8                  defaultValue={config.openaiBaseUrl}
9                  onChange={(e) =>
10                      setConfig({
11                          ...config,
12                          openaiBaseUrl: e.target.value,
13                      })
14              </>
15          </div>
16

```



2. 放在 Gemini API Key 下面

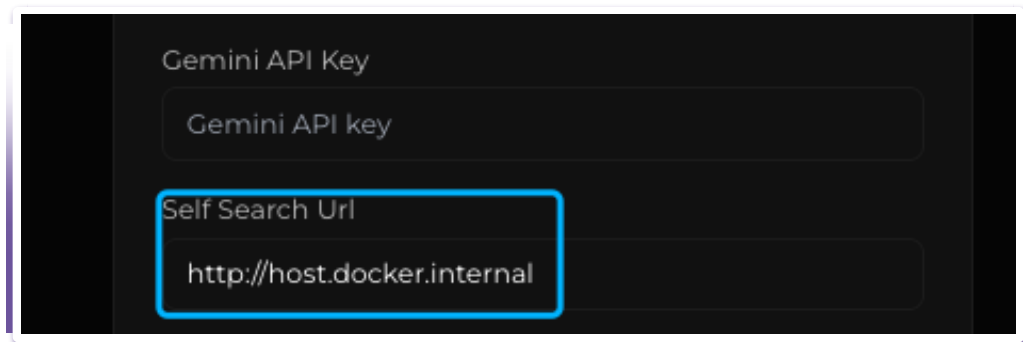
```

1          <div className="flex flex-col space-y-1">
2              <p className="text-black/70 dark:text-white/70
text-sm">
3                  Self Search Url
4              </p>
5              <Input
6                  type="text"
7                  placeholder="Self Search Url"
8                  defaultValue={config.selfSearchUrl}
9                  onChange={(e) =>
10                      setConfig({
11                          ...config,
12                          selfSearchUrl: e.target.value,
13                      })

```



```
14         }
15     }
16 </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

```

1      'qwen-long': {
2          displayName: 'Qwen-Long',
3          model: new ChatOpenAI({
4              openAIApiKey,
5              modelName: 'qwen-long',
6              temperature: 0.7,
7          }, {
8              baseUrl: getOpenaiBaseUrl()
9          }),
10     },

```

其中

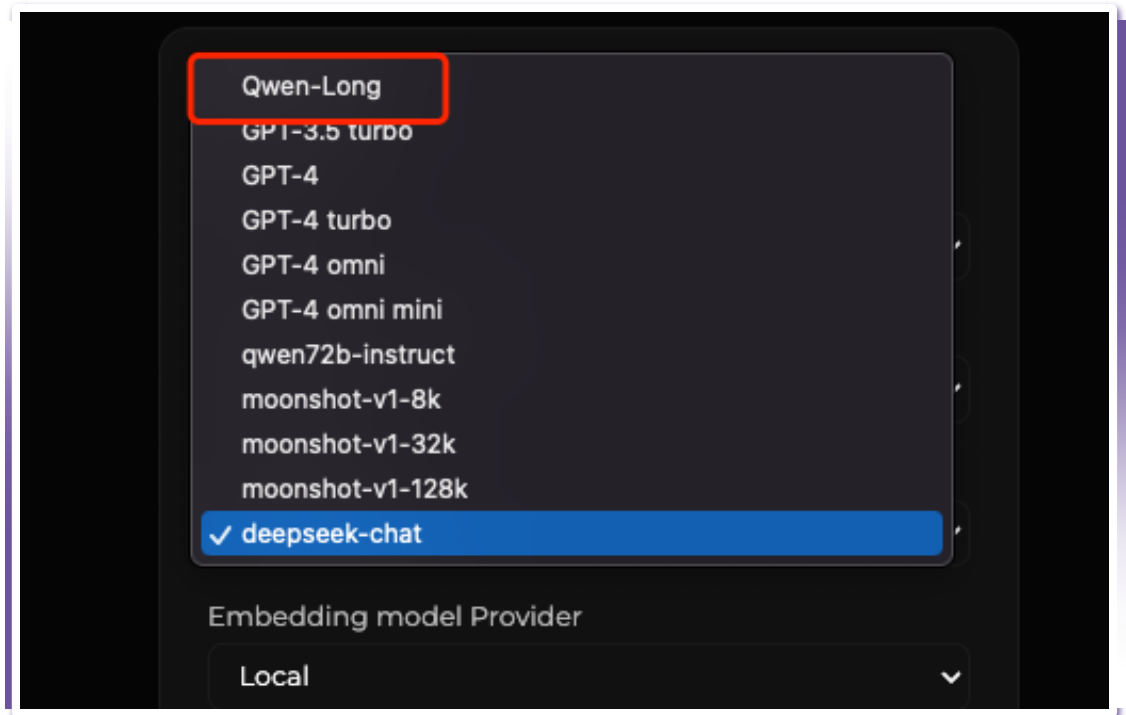
1. `modelName`：为 `oneapi` 中模型的名称；
2. `displayName`：为显示在前端中的模型名称；
3. `baseUrl`：如果没有指定则默认为 `openai` 自己的 url，且这时 `api-key` 必须是原生的 `openai api key`。

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-PJrYllj1QwodFT4n00Df8956B465411389C44dF11eDc07Da
```

自行增减 模型

更新令牌信息

名称

Perplexica专用

模型范围

Qwen/Qwen2-72B-Instruct ✕ gpt-3.5-turbo ✕ gpt-4 ✕ gpt-4-turbo ✕ gpt-4o ✕ gpt-4o-mini ✕ moonshot-v1-8k ✕ moonshot-v1-32k ✕ moonshot-v1-128k ✕ deepseek-chat ✕ text-embedding-ada-002 ✕ qwen-long ✕ qwen2.5-72b-instruct ✕

- `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
```

```
1      'qwen-long': {
2        displayName: 'Qwen-Long',
3        model: new ChatOpenAI({
4          openAIApiKey,
5          modelName: 'qwen-long',
6          temperature: 0.7,
7        }, {
8          baseUrl: getOpenaiBaseUrl()
9        }),
10     },
11     'gpt-3.5-turbo': {
12       displayName: 'GPT-3.5 Turbo',
13       model: new ChatOpenAI({
14         openAIApiKey,
15         modelName: 'gpt-3.5-turbo',
16         temperature: 0.7,
17       }, {
18         baseUrl: getOpenaiBaseUrl()
19       }),
20     },
21     'gpt-4': {
22       displayName: 'GPT-4',
23       model: new ChatOpenAI({
24         openAIApiKey,
25         modelName: 'gpt-4',
26         temperature: 0.7,
27       }, {
28         baseUrl: getOpenaiBaseUrl()
29       }),
30     },
31     'gpt-4-turbo': {
32       displayName: 'GPT-4 turbo',
33       model: new ChatOpenAI({
```

```

34         openAIApiKey,
35         modelName: 'gpt-4-turbo',
36         temperature: 0.7,
37     }, {
38         baseUrl: getOpenaiBaseUrl()
39     } ),
40 },
41 'gpt-4o': {
42     displayName: 'GPT-4 omni',
43     model: new ChatOpenAI({
44         openAIApiKey,
45         modelName: 'gpt-4o',
46         temperature: 0.7,
47     }, {
48         baseUrl: getOpenaiBaseUrl()
49     } ),
50 },
51 'gpt-4o-mini': {
52     displayName: 'GPT-4 omni mini',
53     model: new ChatOpenAI({
54         openAIApiKey,
55         modelName: 'gpt-4o-mini',
56         temperature: 0.7,
57     }, {
58         baseUrl: getOpenaiBaseUrl()
59     } ),
60 },
61 'Qwen/Qwen2-72B-Instruct': {
62     displayName: 'qwen72b-instruct',
63     model: new ChatOpenAI({
64         openAIApiKey,
65         modelName: 'Qwen/Qwen2-72B-Instruct',
66         temperature: 0.7,
67     }, {
68         baseUrl: getOpenaiBaseUrl()
69     } ),
70 },
71 'moonshot-v1-8k': {
72     displayName: 'moonshot-v1-8k',
73     model: new ChatOpenAI({
74         openAIApiKey,
75         modelName: 'moonshot-v1-8k',
76         temperature: 0.7,
77     }, {
78         baseUrl: getOpenaiBaseUrl()
79     } ),
80 },
81 'moonshot-v1-32k': {
82     displayName: 'moonshot-v1-32k',
83     model: new ChatOpenAI({
84         openAIApiKey,
85         modelName: 'moonshot-v1-32k',

```

```

86         temperature: 0.7,
87     }, {
88         baseUrl: getOpenaiBaseUrl()
89     } ),
90 },
91 'moonshot-v1-128k': {
92     displayName: 'moonshot-v1-128k',
93     model: new ChatOpenAI({
94         openAIApiKey,
95         modelName: 'moonshot-v1-128k',
96         temperature: 0.7,
97     }, {
98         baseUrl: getOpenaiBaseUrl()
99     } ),
100 },
101 'deepseek-chat': {
102     displayName: 'deepseek-chat',
103     model: new ChatOpenAI({
104         openAIApiKey,
105         modelName: 'deepseek-chat',
106         temperature: 0.7,
107     }, {
108         baseUrl: getOpenaiBaseUrl()
109     } ),
110 },

```

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

```

1  {
2      "query": query,
3      "datasetIds": ["661f9c02e50601c600e26eb9"],
4      "topN": 9,
5      "simThreshold": 0.0
6  }

```

- python连接 示例代码

```

1  import requests
2
3  """知乎数据的相关搜索接口"""
4  url = 'http://js1.blockelite.cn:19575/search'
5  body = {
6      "query":query,
7      "datasetIds":["661f9c02e50601c600e26eb9"],
8      "topN": 2,
9      "simThreshold": 0.0
10 }
11 res = requests.post(url,json=body).json()
12 conv_links,content_list = res['conv_links'],res['content_list']

```

- typescript 连接示例代码

```

1  import axios from 'axios';
2
3  const url = "http://js1.blockelite.cn:19575/search";
4  const payload = {
5      query: "什么是保险",
6      datasetIds: ["661f9c02e50601c600e26eb9"],
7      topN: 2,
8      simThreshold: 0
9  };
10
11  axios.post(url, payload)
12      .then(response => {
13          // 处理响应数据
14          console.log("conv_links:",response.data.conv_links);
15          console.log("content_list:",response.data.content_list);
16      })
17      .catch(error => {
18          // 处理错误情况
19          console.error(error);
20      });

```

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
```

```
1  npm install axios
2
3  tsc search_use_zhihu.ts
4
5  node search_use_zhihu.js
```

- 代码(topN =2 测试)

```
1  import axios from 'axios';
2
3  interface SearxngSearchOptions {
4      categories?: string[];
5      engines?: string[];
6      language?: string;
7      pageno?: number;
8  }
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 export const searchSearxng = async (
22     query: string,
23     opts?: SearxngSearchOptions,
24 ) => {
25     const url = "http://js1.blockelite.cn:19575/search";
26     console.log('url', url);
27     const payload = {
28         query: query,
29         datasetIds: ["661f9c02e50601c600e26eb9"],
30         topN: 2,
31         simThreshold: 0
32     };
33
34     try {
35         const response = await axios.post(url, payload);
36         const contentList = response.data.content_list;
37         const convLinks = response.data.conv_links;
38
39         const results: SearxngSearchResult[] = [];
40         const suggestions: any[] = [];
41
42         contentList.forEach((val, idx) => {
43             const dataDict = {
44                 url: val.url,
```



```

45         title: convLinks[idx].title,
46         content: val.content,
47         engine: 'bing',
48         template: 'default.html',
49         engines: ['bing'],
50         positions: [idx + 1],
51         score: val.score,
52         category: 'general'
53     };
54     results.push(dataDict);
55 });
56 console.log('这里是结果:convLinks,', convLinks)
57 return { results, suggestions };
58 } catch (error) {
59     console.error('这里出现错误:', error);
60     throw error;
61 }
62 };
63
64
65 // 调用 searchSearxng 函数
66 async function performSearch() {
67     const query = '中国平安2023年业绩'; // 你的搜索查询
68     const options: SearxngSearchOptions = {
69         categories: ['general', 'news'], // 可选的搜索类别
70         engines: ['bing', 'google'], // 可选的搜索引擎
71         language: 'zh-CN', // 可选的搜索语言
72         pageno: 1 // 可选的页码
73     };
74
75     try {
76         const response = await searchSearxng(query, options);
77         console.log('搜索结果:', response.results);
78         console.log('response 的 type', typeof response);
79         console.log('results 的 type', typeof response.results);
80         console.log('results 的 type333', typeof response.suggestions);
81     } catch (error) {
82         console.error('搜索出错:', error);
83     }
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
28   const url = new URL(`${searxngURL}/search?format=json`);
29   url.searchParams.append('q', query);
30
31   if (opts) {
32     Object.keys(opts).forEach((key) => {
33       if (Array.isArray(opts[key])) {
34         url.searchParams.append(key, opts[key].join(','));
35         return;
36       }
37       url.searchParams.append(key, opts[key]);
38     });
39   }
40
41   const res = await axios.get(url.toString());
42
43   const results: SearxngSearchResult[] = res.data.results;
44   const suggestions: string[] = res.data.suggestions;
45 }
```

```
46     return { results, suggestions };
47   };
```

修改后代码

```
1   import axios from 'axios';
2   import { getSelfSearchUrl } 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 url = "http://js1.blockelite.cn:19575/search";
27    const url = getSelfSearchUrl();
28    console.log('url', url);
29    const payload = {
30      query: query,
31      datasetIds: ["661f9c02e50601c600e26eb9"],
32      topN: 10,
33      simThreshold: 0
34    };
35
36    try {
37      const response = await axios.post(url, payload);
38      const contentList = response.data.content_list;
39      const convLinks = response.data.conv_links;
40
41      const results: SearxngSearchResult[] = [];
42      const suggestions: any[] = [];
43
44      contentList.forEach((val, idx) => {
45        const dataDict = {
46          url: val.url,
47          title: convLinks[idx].title,
```

```

48         content: val.content,
49         engine: 'bing',
50         template: 'default.html',
51         engines: ['bing'],
52         positions: [idx + 1],
53         score: val.score,
54         category: 'general'
55     };
56     results.push(dataDict);
57 });
58 console.log('这里是结果:convLinks,', convLinks)
59 return { results, suggestions };
60 } catch (error) {
61     console.error('这里出现错误:', error);
62     throw error;
63 }
64 };
65
66
67 // // 调用 searchSearxng 函数
68 // async function performSearch() {
69 //     const query = '中国平安2023年业绩'; // 你的搜索查询
70 //     const options: SearxngSearchOptions = {
71 //         categories: ['general', 'news'], // 可选的搜索类别
72 //         engines: ['bing', 'google'], // 可选的搜索引擎
73 //         language: 'zh-CN', // 可选的搜索语言
74 //         pageno: 1 // 可选的页码
75 //     };
76
77 //     try {
78 //         const response = await searchSearxng(query, options);
79 //         console.log('搜索结果:', response.results);
80 //         console.log('response 的 type', typeof response);
81 //         console.log('results 的 type', typeof response.results);
82 //         console.log('results 的 type333', typeof response.suggestions);
83 //     } catch (error) {
84 //         console.error('搜索出错:', error);
85 //     }
86 // }
87
88
89 // // 执行搜索
90 // 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
```

```
1  npm install axios
2
3  tsc search_use_python.ts
4
5  node search_use_python.js
```

- 代码

```
1  import axios from 'axios';
2
3  interface SearxngSearchOptions {
4    categories?: string[];
5    engines?: string[];
6    language?: string;
7    pageno?: number;
8  }
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
53     const options: SearxngSearchOptions = {
54         categories: ['general', 'news'], // 可选的搜索类别
55         engines: ['bing', 'google'], // 可选的搜索引擎
56         language: 'zh-CN', // 可选的搜索语言
57         pageno: 1 // 可选的页码
58     };
59
60     try {
61         const response = await searchSearxng(query, options);
62         console.log('搜索结果:', response.results);
63         console.log('response 的 type', typeof response);
64         console.log('results 的 type', typeof response.results);
65         console.log('results 的 type333', typeof response.suggestions);
66     } catch (error) {
67         console.error('搜索出错:', error);
68     }
69 }
70
71
72 // 执行搜索
73 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

原代码:同上

修改后

```
1  import axios from 'axios';
2  import { getSelfSearchUrl } 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
23 export const searchSearxng = async (
24   query: string | string[],
25   opts?: SearxngSearchOptions,
26 ) => {
27   if (typeof query === "string") { // 如果 query 是字符串类型, 将其转换为单元
    素数组
28     query = [query];
29   } 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。
30     throw new Error("query必须是 【字符串】 或者 【字符串数组】 ! ! ! ! !");
31   }
32
33   // const url = "http://localhost:8013/retrieval";
34   const urlroot = getSelfSearchUrl();
35   const url = urlroot + "/retrieval";
36
37   console.log('python 服务检索的url: ', url);
38   const payload = {
39     query_list: query
40   };
41
42   try {
43     const response = await axios.post(url, payload);
```

```

44     const results = response.data.results;
45     const suggestions = response.data.suggestions;
46     return { results, suggestions };
47   } catch (error) {
48     console.error('这里出现错误:', error);
49     throw error;
50   }
51 };
52
53 // // 调用 searchSearxng 函数
54 // async function performSearch() {
55 //   const query = '中国平安2023年业绩'; // 你的搜索查询
56 //   const options: SearxngSearchOptions = {
57 //     categories: ['general', 'news'], // 可选的搜索类别
58 //     engines: ['bing', 'google'], // 可选的搜索引擎
59 //     language: 'zh-CN', // 可选的搜索语言
60 //     pageno: 1 // 可选的页码
61 //   };
62
63 //   try {
64 //     const response = await searchSearxng(query, options);
65 //     console.log('搜索结果:', response.results);
66 //     console.log('response 的 type', typeof response);
67 //     console.log('results 的 type', typeof response.results);
68 //     console.log('results 的 type333', typeof response.suggestions);
69 //   } catch (error) {
70 //     console.error('搜索出错:', error);
71 //   }
72 // }
73
74
75 // // 执行搜索
76 // 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）

```

1     # searxng:
2     #   image: docker.io/searxng/searxng:latest
3     #   volumes:
4     #     - ./searxng:/etc/searxng:rw
5     #   ports:
6     #     - 4000:8080

```



```
7      # networks:
8      #   - perplexica-network
9      # restart: unless-stopped
10
11
12
13      # depends_on:
14      #   - searxng
```

1.6 接口相关

1.6.1 图片视频接口返回

1.6.1.1 输入值

- 用户 query

1.6.1.2 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

```
1      title: string;
2      url: string;
3      img_src: string;
4      thumbnail_src: string;
5      thumbnail: string;
6      content: string;
7      author: string;
8      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']`

1.6.1.3 图片接口

- 原代码

```
1      RunnableLambda.from(async (input: string) => {
2          const res = await searchSearxng(input, {
3              engines: ['bing images', 'google images'],
```

```

4      }); //使用 searchSearxng 函数执行搜索, 指定搜索引擎为 'bing images' 和
      'google images'。
5
6      const images = [];
7
8      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。
9          if (result.img_src && result.url && result.title) {
10             images.push({
11                 img_src: result.img_src,
12                 url: result.url,
13                 title: result.title,
14             });
15         }
16     });
17
18     return images.slice(0, 10); // 返回前 10个结果
19 },

```

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"
},
]

```

1.6.1.4 视频接口

- 原代码

```

1  RunnableLambda.from(async (input: string) => {
2    const res = await searchSearxng(input, {
3      engines: ['youtube'],
4    });
5
6    const videos = [];
7
8    res.results.forEach((result) => {
9      if (
10        result.thumbnail &&
11        result.url &&
12        result.title &&
13        result.iframe_src
14      ) {
15        videos.push({
16          img_src: result.thumbnail,
17          url: result.url,
18          title: result.title,
19          iframe_src: result.iframe_src,
20        });
21      }
22    });
23
24    return videos.slice(0, 10);
25  })

```

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=mF3l8nKlPFc ",
    "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/mF3l8nKlPFc ",
    "thumbnail": " https://i.ytimg.com/vi/mF3l8nKlPFc/hqdefault.jpg ",
    "engine": "youtube",
    "engines": [
      "youtube"
    ],
    "positions": [
```

```
    2
  ],
  "score": 0.5,
  "category": "videos"
}]
```

1.6.1.5 东京机上的 searxng 测试(服务不一定开了)

- curl

```
1 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'
```

1.6.2 python 服务

- build 镜像

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

- 启动

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

- health

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

- 改写接口

```
1 curl --location 'http://localhost:8013/rewrite_query' \
2 --header 'Content-Type: application/json' \
3 --data '{
4   "query": "24年的大学生就业情况怎么样"
5 }'
```

- 检索接口

```
1 curl --location 'http://localhost:8013/retrieval' \
2 --header 'Content-Type: application/json' \
3 --data '{
4   "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"]
5 }
6 '
```

1.6.3 Perplexica 自带的后端接口

- health

```
1 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)

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

- 更改 config (可更改 config.toml 中的内容)

```
1 curl --location 'http://127.0.0.1:3001/api/config' \  
2 --header 'Content-Type: application/json' \  
3 --data '{  
4     "groqApiKey": "11111"  
5 }'
```

- 获取所有聊天

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

- 获取已经配置的 model 和 embedding model

```
1 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

```
1 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

```
1 curl --location 'http://127.0.0.1:3001/api/suggestions' \  
2 --header 'Content-Type: application/json' \  
3 --data '{  
4     "chatHistory": [  
5         {  
6             "id": 16,  
7             "content": "什么是保险",  
8             "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",  
9             "messageId": "033eee2ea15ddc",  
10            "role": "user",  
11            "metadata": "{\"createdAt\":\"2024-11-28T01:26:35.194Z\"}"  
12        },  
13        {  
14            "id": 17,  
15            "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:",  
16            "chatId": "c539389ea79786145734b6289b9f0095e14b8b32",  
17            "messageId": "9266de787dab70",  
18            "role": "assistant",  
19            "metadata": ""  
20        }  
21    ]  
22 }'
```

```
20         }
21     ],
22     "chatModel": { "provider": "openai", "model": "qwen-long" }
23 }
```

- discover 页面

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