FastAPI 中文 API 接口使用说明书

此文档为前端开发者提供了如何与后端 API 进行对接的使用说明。该 API 提供了获取不同数据类型的接口，支持 GET 和 POST 方法，并且最后一个接口是一个异步聊天机器人接口，支持流式输出。

基本信息

- 基础 URL: `http://<your-server>:1551`
- 所有接口都需要提供 `api_key` 作为查询参数进行身份验证。

API 接口列表

1. 获取新的 Acknowledge 数据

GET /v1/api/new

- **描述**: 获取新的 Acknowledge 数据。
- **方法**: GET
- **参数**:
  - `date` (可选): 用于根据日期过滤结果的数据。
  - `api_key` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `date` 参数的数据列表。如果未提供 `date` 参数，则返回所有数据。

POST /v1/api/new

- **描述**: 获取新的 Acknowledge 数据。
- **方法**: POST
- **参数**:
  - 请求体包含:
    - `date` (可选): 用于根据日期过滤结果的数据。
    - `api` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `date` 参数的数据列表。如果未提供 `date` 参数，则返回所有数据。

示例

```
// GET 请求
fetch('http://<your-server>:1551/v1/api/new?date=2023-07-
01&api_key=your_api_key')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// POST 请求
fetch('http://<your-server>:1551/v1/api/new', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
```

```
  body: JSON.stringify({
    date: '2023-07-01',
    api: 'your_api_key'
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

2. 获取对话 Acknowledge 数据

GET /v1/api/conversation

- **描述**: 获取对话 Acknowledge 数据。
- **方法**: GET
- **参数**:
  - `id` (可选): 用于根据 ID 过滤结果的数据。
  - `api_key` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `id` 参数的数据列表。如果未提供 `id` 参数，则返回所有数据。

POST /v1/api/conversation

- **描述**: 获取对话 Acknowledge 数据。
- **方法**: POST
- **参数**:
  - 请求体包含:
    - `id` (可选): 用于根据 ID 过滤结果的数据。
    - `api` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `id` 参数的数据列表。如果未提供 `id` 参数，则返回所有数据。

示例

```
// GET 请求
fetch('http://<your-server>:1551/v1/api/conversation?
id=123&api_key=your_api_key')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// POST 请求
fetch('http://<your-server>:1551/v1/api/conversation', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    id: '123',
    api: 'your_api_key'
  })
})
  .then(response => response.json())
```

```
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

3. 获取黑客 Acknowledge 数据

GET /v1/api/hacker

- **描述**: 获取黑客 Acknowledge 数据。
- **方法**: GET
- **参数**:
  - `id` (可选): 用于根据 ID 过滤结果的数据。
  - `api_key` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `id` 参数的数据列表。如果未提供 `id` 参数，则返回所有数据。

POST /v1/api/hacker

- **描述**: 获取黑客 Acknowledge 数据。
- **方法**: POST
- **参数**:
  - 请求体包含:
    - `id` (可选): 用于根据 ID 过滤结果的数据。
    - `api` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `id` 参数的数据列表。如果未提供 `id` 参数，则返回所有数据。

示例

```
// GET 请求
fetch('http://<your-server>:1551/v1/api/hacker?id=123&api_key=your_api_key')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// POST 请求
fetch('http://<your-server>:1551/v1/api/hacker', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    id: '123',
    api: 'your_api_key'
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

4. 获取 AI 工具信息

GET /v1/api/tools_info

- **描述**: 获取 AI 工具信息。
- **方法**: GET
- **参数**:
  - `title` (可选): 用于根据标题过滤结果的数据。
  - `api_key` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `title` 参数的数据列表。如果未提供 `title` 参数，则返回所有数据。

POST /v1/api/tools_info

- **描述**: 获取 AI 工具信息。
- **方法**: POST
- **参数**:
  - 请求体包含:
    - `title` (可选): 用于根据标题过滤结果的数据。
    - `api` (必选): 身份验证的 API 密钥。
- **返回值**: 匹配 `title` 参数的数据列表。如果未提供 `title` 参数，则返回所有数据。

示例

```
// GET 请求
fetch('http://<your-server>:1551/v1/api/tools_info?title=Some
Tool&api_key=your_api_key')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

// POST 请求
fetch('http://<your-server>:1551/v1/api/tools_info', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    title: 'Some Tool',
    api: 'your_api_key'
  })
})
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

5. 聊天机器人接口

POST /v1/chat/completions

- **描述**: 使用聊天机器人生成回复。
- **方法**: POST
- **参数**:
  - 请求体包含:
    - `query` (必选): 用户的查询内容。
- **返回值**: 流式输出机器人回复。

示例

```
<!DOCTYPE html>
<html>
<head>
  <title>Chat Bot</title>
</head>
<body>
  <h1>Chat Bot</h1>
  <div id="chat-box"></div>
  <input type="text" id="user-input" placeholder="Type your message here...">
  <button onclick="sendMessage()">Send</button>

  <script>
    function appendMessage(message, sender) {
      const chatBox = document.getElementById('chat-box');
      const messageElement = document.createElement('p');
      messageElement.textContent = `${sender}: ${message}`;
      chatBox.appendChild(messageElement);
    }

    function sendMessage() {
      const userInput = document.getElementById('user-input').value;
      const apiUrl = 'http://<your-server>:1551/v1/chat/completions';
      const apiKey = 'your_api_key';

      fetch(apiUrl, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${apiKey}`
        },
        body: JSON.stringify({ query: userInput })
      })
      .then(response => {
        const reader = response.body.getReader();
        const decoder = new TextDecoder('utf-8');

        function readStream() {
          reader.read().then(({ done, value }) => {
            if (done) {
              return;
```

```
            }
            const message = decoder.decode(value);
            appendMessage(message, 'Bot');
            readStream();
          });
        }

        readStream();
      })
      .catch(error => console.error('Error:', error));

      appendMessage(userInput, 'User');
      document.getElementById('user-input').value = '';
    }
  </script>
</body>


</html>
```

中间件

API 使用了两个中间件：

1. **log_requests**: 记录每个请求的日志，包括请求处理时间。
2. **log_crawlers**: 检测并记录爬虫的访问行为。

CORS 配置

API 配置了 CORS 中间件，允许所有来源的请求，支持所有方法和头部。

文件挂载

- `/news_images`: 挂载目录 `utils/new_images` 以提供静态文件服务。
- `/tools_images`: 挂载目录 `utils/tools_images` 以提供静态文件服务。
