1. 基于SpringMVC框架的Chat GPT提示词系统 开发文档
1.1 系统基本思路
1.2 前端
1.2.1 通过FastAPI获取提示词
1.2.2 HTTP请求响应
1.2.3 实现AI"记忆"功能
1.2.4 实现删除功能
1.3 后端
1.3.1 调用AI接口：chat_with_gpt
1.3.2 提示词模板
2. 成果展示

1. 基于SpringMVC框架的Chat GPT提示词系统 开发文档

1.1 系统基本思路

用户于前端输入提示词（提问），后端通过接口FastAPI获取提示词，并与chat_with_gpt函数进行交互获取返回内容。同时，该系统还拥有记忆功能，即通过保存之前的提示词并同时和当前提示词共同输入进chat_with_gpt，以此实现"阅读"前后文进行AI回答。最后将chat_with_gpt的输出内容返回至前端页面。

1.2 前端

1.2.1 通过FastAPI获取提示词

```
app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/chat/{query}")
def chat(query: str):  #  query :查询
    return chat_with_gpt(query, os.getenv("OPENAI_API_KEY"))
```

1.2.2 HTTP请求响应

通过RequestMapping函数对前端请求进行响应

```
@RequestMapping(value = "gener", produces = "text/html;charset=utf-8")
    public String getGenerChat(String theme, Model model) {
        String themeo="";    // 存放 经过getBytes 转换后的theme内容
```

```
        try {
            themeo = new String(theme.getBytes("ISO-8859-1"), "UTF-8"); // 转换编码    编码转换：将输入的主题字符串从 ISO-8859-1 编码转换为 UTF-8 编码，以确保正确处理中文字符。
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }

        String str = demoService.generChat(themeo, memory.getChatHistory());  // 联系上下文  结合历史记录和当下主题生成所需的结果str

        // 打印字符串
        System.out.println("API 响应: [" + str + "]");
        Answer answer = null;
        try {
            answer = JSON.parseObject(str, Answer.class);
        } catch (JSONException e) {
            System.err.println("JSON 解析错误: " + e.getMessage());
            System.err.println("解析失败的字符串: [" + str + "]"); // 打印出解析失败的字符串
            // 返回一个默认的 Answer 对象或者处理错误
        }
        //Answer answer = JSON.parseObject(str, Answer.class);
        memory.addEntry(new ChatEntry("用户", themeo));
        memory.addEntry(new ChatEntry("GPT", answer.getMessage()));

        // 添加模型属性：将聊天历史和回答添加到模型中，以便在视图中使用
        model.addAttribute("chatHistory", memory.getChatHistory());
        model.addAttribute("answer", answer);
        return "gener.jsp";  // 返回视图名称 gener.jsp，用于渲染响应。
    }
```

1.2.3 实现AI"记忆"功能

将聊天历史和回答添加到模型中，以便在视图中使用

```
memory.addEntry(new ChatEntry("用户", themeo));
        memory.addEntry(new ChatEntry("GPT", answer.getMessage()));

        model.addAttribute("chatHistory", memory.getChatHistory());
        model.addAttribute("answer", answer);
```

1.2.4 实现删除功能

将保存在列表内的对象内容进行清空

```
@RequestMapping(value = "clearMemory", produces = "text/html;charset=utf-8")
    public String clearMemory(Model model) {
        memory.clearHistory();
        model.addAttribute("chatHistory", memory.getChatHistory());
        return "gener.jsp";
    }
```

```java
package nj.ng.entity;

import java.util.ArrayList;
import java.util.List;

public class Memory {
    private List<ChatEntry> chatHistory = new ArrayList<>();

    public List<ChatEntry> getChatHistory() {
        return chatHistory;
    }

    public void addEntry(ChatEntry entry) {
        chatHistory.add(entry);
    }

    public void clearHistory() {
        chatHistory.clear();
    }
}
```

1.3 后端

1.3.1 调用AI接口：`chat_with_gpt`

```python
def chat_with_gpt(query, openai_api_key):
    prompt = ChatPromptTemplate.from_messages([
        ("system", system_template_text),
        ("user", user_template_text)
    ])
    model = ChatOpenAI(model="gpt-3.5-turbo", api_key=openai_api_key,
openai_api_base="https://api.aigc369.com/v1")
    output_parser = PydanticOutputParser(pydantic_object=ChatResponse)  #创建一个用于解析输出的对象（PydanticOutputParser），并指定要解析的输出格式（ChatResponse）

    chain = prompt | model | output_parser   #将上述步骤连接成一个处理链（chain），包括对话模板、模型和输出解析器。

    result = chain.invoke({   # 调用链的invoke方法，传入解析器的格式指令和用户的查询，从而触发整个处理过程
        "parser_instructions": output_parser.get_format_instructions(),
        "query": query
    })
    return result
```

调用AI接口，将提示词输入其中，再将结果`result`返回至前端页面。

1.3.2 提示词模板

为了方便用户进行阅读、理解AI回答的内容，故设定提示词回复模板

```
system_template_text = """你是一个全能的GPT聊天助手，专门设计用于与用户进行对话，回答他们的问题，提供建议和信息。请遵循以下指导原则：

### 1. 理解用户需求
- **分析用户问题**：仔细理解用户提出的问题，确定其核心需求。
- **提供相关建议**：根据用户的需求，提供针对性的建议或信息。

### 2. 交流风格
- **保持专业和友好**：使用友好的语气，让用户感到舒适和被重视。
- **鼓励互动**：询问更多细节以明确用户意图，鼓励用户继续提问。

### 3. 回答结构
- **简洁明了**：确保回答清晰、简洁，避免冗长的解释。
- **有效信息**：提供有用的信息，确保回答具有实质性，避免重复。

### 4. 提供示例
- **举例说明**：在合适的情况下，提供相关的例子以增强理解。

### 5. 适应性
- **灵活应对**：能够适应不同的话题和用户风格，调整回答的复杂度和形式。

请确保你的回答帮助用户有效解决他们的问题，并在需要时提供相关的资源和链接，以促进进一步的了解。

{parser_instructions}
"""

user_template_text = "{query}"
```

2. 成果展示


GPT聊天
大模型
人工智能是一种模拟人类智能的技术，旨在使计算机系统能够执行需要人类智力的任务。这包括学习、推理、问题解决和自主感知等功能。人工智能的应用领域包括机器学习、自然语言处理、计算机视觉和专家系统等。
我和你进行了几次对话
您和我进行了两次对话。在第一次对话中，您询问了关于人工智能的问题，我解释了人工智能是一种模拟人类智能的技术。在第二次对话中，您提到了大模型，这也是人工智能领域中的一个重要话题。如果您有任何更多问题或需要进一步了解，请随时提出。
输入你的消息...
发送