

**《人工智能大模型应用基础》课程**

**基于大模型的问答系统  
代码分析与设计说明**

姓 名： 杜勃锋

学 号： 22407020315

班 级： 通信工程2303班

学 院： 通信与信息工程学院

设计说明

本项目是设计一个基于langchian框架，使用All-miniLM-L6-v2作为文本词向量嵌入模型和ChatGLM-4-Flash作为文本解析大模型的问答系统，它使用了多个库和组件来构建一个能够理解和回答用户问题的智能助手。

代码分析

依赖库和组件说明

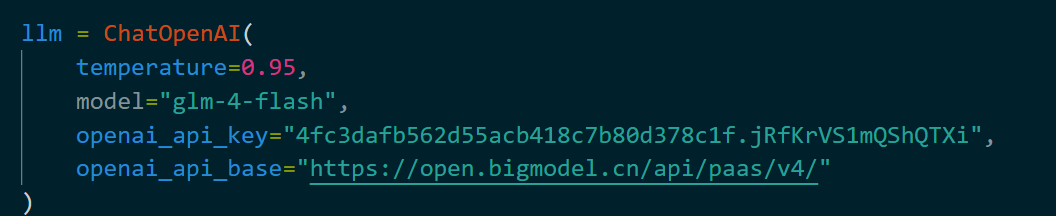
* langchain.chains.create\_retrieval\_chain：创建一个检索链，用于整合问答和文档检索功能。
* langchain.chains.combine\_documents.create\_stuff\_documents\_chain：构建一个文档处理链，允许将多个文档合并为一个回答。
* langchain\_community.document\_loaders.WebBaseLoader：从网络加载文档的工具，便于获取网页内容。
* langchain\_core.output\_parsers.StrOutputParser：用于解析字符串格式的输出，通常用于处理文本结果。
* langchain\_core.prompts.ChatPromptTemplate：构建聊天提示模板，以便为模型生成特定格式的输入。
* langchain\_core.runnables.RunnablePassthrough：一个可运行的对象，允许将输入直接传递到输出，通常用于调试或简单处理。
* langchain\_core.vectorstores.InMemoryVectorStore：在内存中存储向量，实现快速检索和相似性搜索。
* langchain\_community.embeddings.HuggingFaceInferenceAPIEmbeddings：使用 Hugging Face 的推理 API 生成文本嵌入，方便进行语义检索。
* langchain\_text\_splitters.RecursiveCharacterTextSplitter：用于将文本分割成较小块的工具，便于处理和分析。
* langchain.prompts.ChatPromptTemplate（重复）: 同上。
* langchain.prompts.MessagesPlaceholder：在聊天提示中使用的占位符，用于动态插入消息。
* langchain.prompts.SystemMessagePromptTemplate：定义系统消息提示的模板，帮助设定对话背景。
* langchain.prompts.HumanMessagePromptTemplate：定义用户消息提示的模板，用于生成用户输入的格式。
* langchain\_openai.ChatOpenAI：提供与 OpenAI 聊天模型的接口，用于生成基于自然语言的响应。
* langchain.chains.LLMChain：用于将大型语言模型（LLM）与其他组件连接，形成完整的处理链。
* langchain.memory.ConversationBufferMemory：保持对话上下文的工具，用于跟踪会话历史。
* langchain\_huggingface.HuggingFaceEmbeddings：提供与 Hugging Face 嵌入模型的接口，生成文本的向量表示。

代码解读

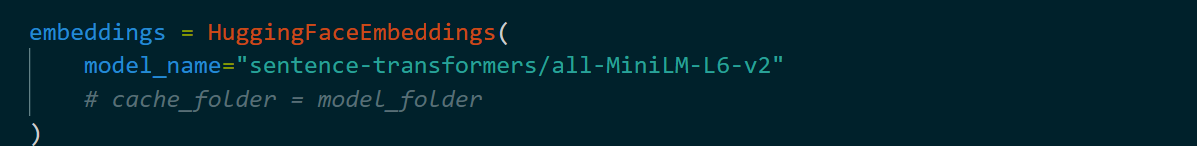
首先，尝试导入在使用langchain框架构建问答系统所使用的依赖库。



接下来我们创建一个使用ChatOpenAI类作为构造器的对象。它负责验证和读取使用大模型所需的api\_key



接下来，定义一个使用HuggingfaceEmbeddings类作为构造器的对象。它负责读取一个本地存储的（或者从外部服务器下载的）embedding模型。



接下来，我们导入langchain定义RAG问答程序的chains所需要的依赖库



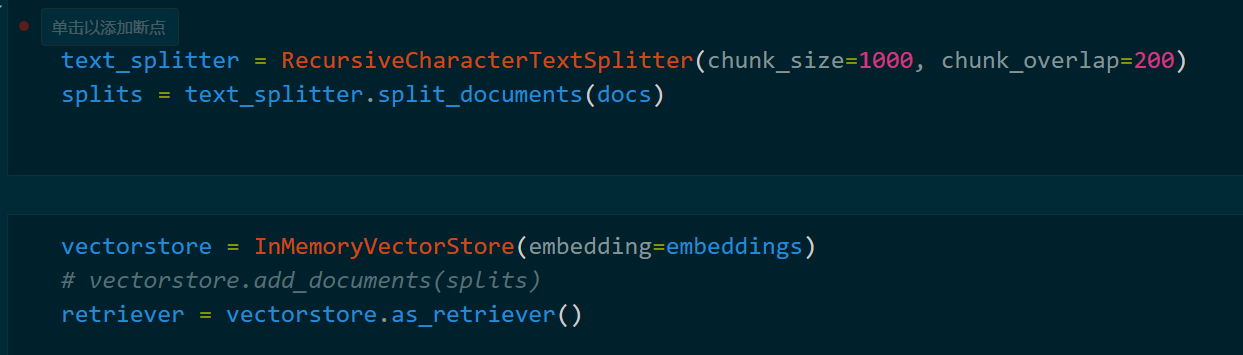
我们创建了基于WebBaseLoader构造器的对象。它的作用是读取web\_path所指向的网页的内容并加载。



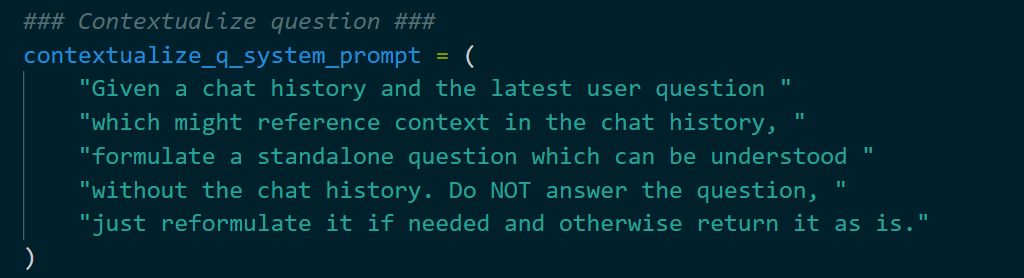
创建了一个 RecursiveCharacterTextSplitter 对象，指定了每个文本块的大小为 1000 字符，并且设置了相邻块之间的重叠为 200 字符。接着，使用 split\_documents 方法对文档 docs 进行分割，结果存储在 splits 变量中。这意味着文档会被拆分成多个较小的部分，便于后续处理。

创建一个名为 vectorstore 的 InMemoryVectorStore 实例，使用提供的 embeddings 作为嵌入表示。

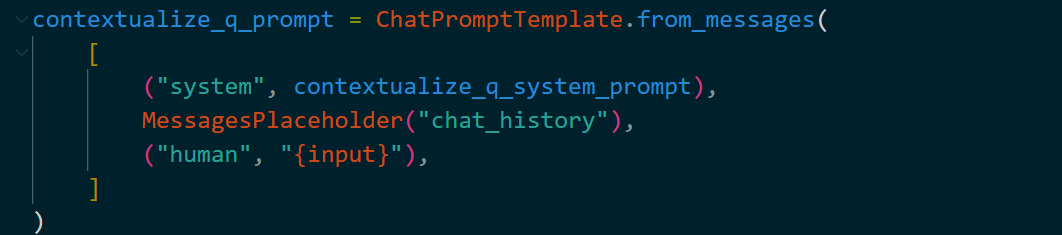
最后，通过调用 as\_retriever() 方法，将这个向量存储转换为一个检索器，便于后续根据嵌入进行信息检索。



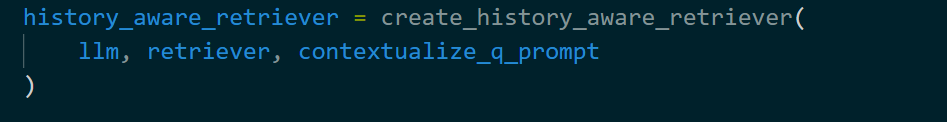
这段代码定义了一个系统提示，用于给模型提供上下文信息。提示指示模型在给定聊天历史和最新用户问题时，如何格式化一个独立的问题，而不是回答该问题。



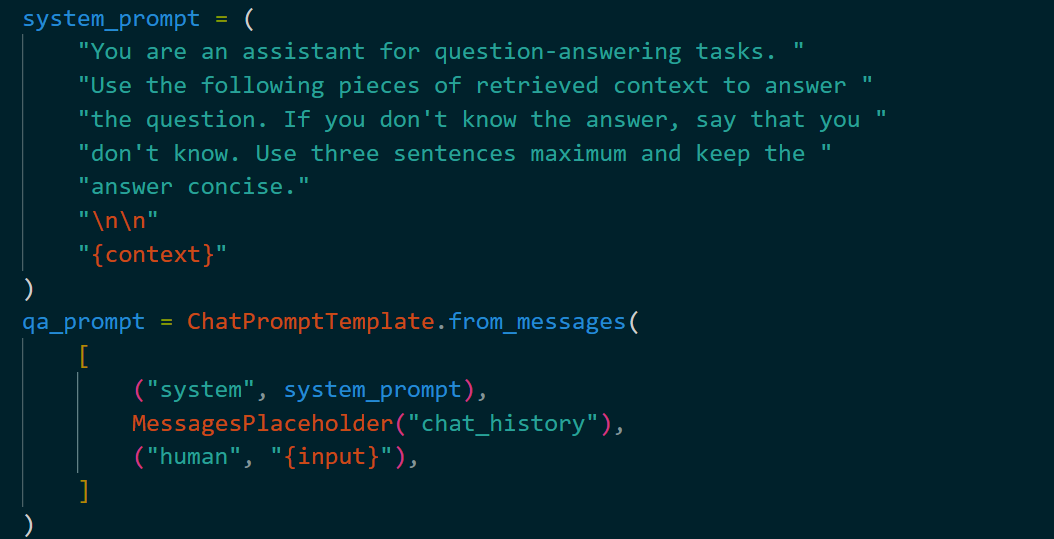
使用 ChatPromptTemplate 类创建了一个聊天提示模板。这个模板包含了系统提示、聊天历史的占位符和用户输入的占位符。这里 MessagesPlaceholder("chat\_history") 表示将要输入的聊天历史信息。



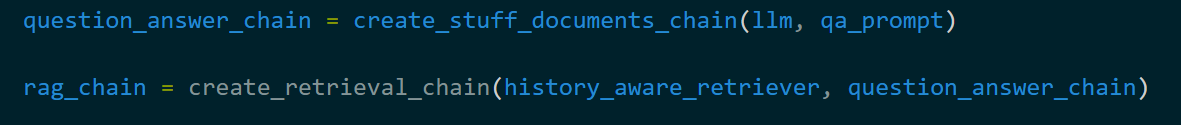
最后，这段代码调用 create\_history\_aware\_retriever 函数，创建一个历史感知的检索器。它使用了指定的语言模型（llm）、检索器（retriever）和之前定义的上下文化提示模板（contextualize\_q\_prompt）。



创建一个问题回答助手的模板，以便在处理用户查询时提供明确的指导。首先，定义了一个 system\_prompt 字符串，为助手设定了角色和操作的框架，指示它在回答问题时需要利用检索到的上下文信息，明确如果无法提供答案，则应该表示“不知道”。此外，系统提示强调回答应简练，限制在最多三句话内。接着，通过 ChatPromptTemplate.from\_messages 方法，构建一个交互式的聊天模板 qa\_prompt，这个模板包含了系统角色的提示、聊天历史的占位符以及用户输入的占位符，确保最后生成的提示能够整合上下文和用户的具体提问，从而实现高效的问答交互。这种设置优化了回答过程，使得助手在提供信息时更具连贯性和相关性。



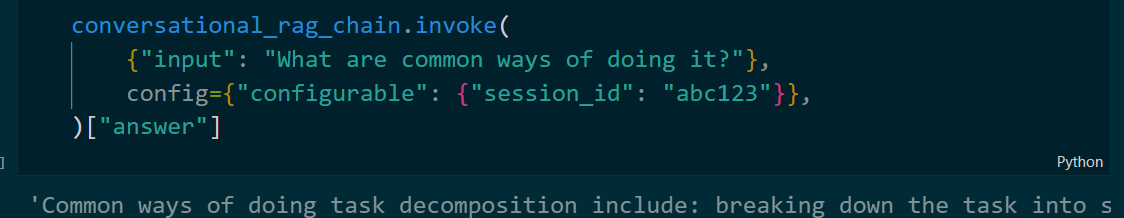
构建一个多阶段的问答系统链。首先，通过 create\_stuff\_documents\_chain 函数，基于指定的语言模型（llm）和问答提示（qa\_prompt），创建了一个处理问题和上下文信息的问答链 question\_answer\_chain。接着，调用 create\_retrieval\_chain 函数，结合历史感知检索器（history\_aware\_retriever）和之前定义的问答链，从而生成一个名为 rag\_chain 的检索链。这一流程的目的是将用户的查询与历史数据相结合，更高效地检索和回答问题。



定义了一个名为 get\_session\_history 的函数，该函数根据会话 ID 检查是否已经存在对应的聊天历史，如果不存在，则为该会话初始化一个新的 ChatMessageHistory 实例，并将其存储在 store 字典中，最后返回对应的历史记录。接着，通过 RunnableWithMessageHistory 设置了输入消息、历史消息和输出消息的键，使得该对象能够在交互过程中自动管理和使用聊天历史，提高问答的上下文相关性和用户体验。



调用 conversational\_rag\_chain 对象的 invoke 方法，以处理一个用户输入的问题并返回相应的答案。具体来说，它传入了一个字典，其中包含用户的提问“有什么常见的实现方式？”以及一个配置字典，指定了会话的 ID 为“abc123”。该方法通过查找历史消息和上下文信息，生成并返回相关的答案，最终通过索引 "answer" 获取相应的结果。



接下来，改动“input”参数的值，并运行代码，输出结果就会显示经由大模型分析给出的结果，从而实现了问答效果。