# 5.1 记忆组件概述

想一想，我们为什么需要记忆 ？

大型语言模型本质上是无记忆的。当我们与其交互时，它仅根据提供的提示生成相应的输出，而无法存储或记住过去的交互内容。这一特性，使得大型语言模型在实现聊天机器人或聊天代理时，难以满足人们的期望。

人们期待聊天机器人具有人的品质和回应能力。当他们意识到机器人只是进行一次性的调用和响应，而无法记住以往的交流内容时，可能会感到沮丧。在现实的聊天环境中，人们的对话充满了缩写和含蓄表达，他们会引用过去的对话内容，并期待对方能够理解和回应。例如，如果聊天历史中只在一开始提到了某人的名字，随后仅用代词指代，那么他们就期望聊天机器人能够理解和记住这个指代关系。

我们期待的是聊天机器人能够跨越整个对话，理解和记住我们的交流内容。但是，要实现这个目标，我们需要赋予大型语言模型一些“记忆”。

#### LangChain 记忆组件

为此，LangChain 提供了两种记忆组件。首先，LangChain 提供了用于管理和操作之前聊天消息的辅助工具。这些工具设计得模块化，可以灵活地适应各种使用场景。其次，LangChain 提供了简单的方式将这些工具整合到模型中。

这种记忆机制的引入，使得大型语言模型能够“回顾”以往的对话，理解并确定他们正在谈论的是谁，以及当前的对话主题是什么。这无疑是对聊天机器人的一大提升，让它更接近人类的交流方式，更好地满足人们的期望。

记忆是在用户与语言模型的互动过程中保留状态概念的过程。用户与语言模型的交互被封装在聊天信息（ChatMessages）这个概念中，因此记忆的关键在于如何摄取、捕捉、转化以及从一系列的聊天信息中提取知识。为此，我们有多种方式可以实现，每种方式都有其对应的记忆类型。

对于每种记忆，我们通常有两种理解方式。一种是独立的函数，从一系列信息中提取信息；另一种则是在链中使用这种记忆类型的方式。

记忆可以返回多种信息（例如，最近的N条信息和所有先前信息的摘要）。返回的信息可以是字符串或是一列信息。

我们会在后面分别介绍这些记忆类型。我们将展示如何在此使用模块化的工具函数，然后展示如何在链中使用它。

#### 记忆组件的核心类

记忆组件的核心实用类之一，是支持大多数（如果不是所有）记忆组件的ChatMessageHistory类。这是一个超轻量级的封装器，它提供了便利的方法，可以保存人类与AI的消息，并取出所有的消息。

如果你在链之外管理记忆，你可能会直接使用这个类。

ChatMessageHistory类以非常方便的方式封装了人类和AI的对话消息。它提供了一种简单的添加方法，使得我们可以将新的聊天信息加入到历史聊天记录中。这个类有一个重要的特性，就是它可以以字符串或消息列表的形式提取信息。如果你的程序在链调用中有特定的记忆对象类型，那么这个基类将在链调用不存在时发挥作用。

以下是ChatMessageHistory类的示例代码：

from langchain.memory import ChatMessageHistory  
  
history = ChatMessageHistory()  
  
history.add\_user\_message("hi!")  
  
history.add\_ai\_message("whats up?")

这个记忆还可以继续添加，我们打印看看 history.messages 的结果：

[HumanMessage(content='hi!', additional\_kwargs={}),  
 AIMessage(content='whats up?', additional\_kwargs={})]

#### 与数据库程序集成

ChatMessageHistory类可以与所有跨平台的数据库（包括MongoDB，Redis 等主流数据库）进行集成。

langchain提供了一个集成包装器，你可以利用它调用ChatMessageHistory类的方法。值得一提的是，你不必清楚所有数据库都被包装成了什么样的类，因为它们都基于ChatMessageHistory类。你可以直接使用这个类的方法，就像调用ChatMessageHistory类的方法一样简单。

下面的代码是与Redis 数据库程序集成的记忆组件的调用方式。

from langchain.memory import RedisChatMessageHistory  
  
history = RedisChatMessageHistory("foo")  
  
history.add\_user\_message("hi!")  
  
history.add\_ai\_message("whats up?")

从代码中我们可以看到跟ChatMessageHistory类添加记忆的方式完全一样。只是“ ChatMessageHistory ”类名加上了“ Redis ”。

#### 模型记忆遇到的问题

记忆组件的工作原理相当直观：它接收聊天消息，将这些消息作为提示，然后传递给语言模型。随后，模型将根据这些提示的约束，生成相应的输出。

你可能会注意到，所有这些信息都会被存储在“提示”中。然而，这种方法在某些情况下会出现问题，那就是当我们进行长期的对话时，语言模型可接受的令牌范围内可能无法容纳所有的信息。

当前，OpenAI的最新模型在一次处理中可以使用大约16k个令牌。尽管这是一个巨大的信息量，但它仍然无法保存完整的对话内容。这就引发了一系列问题，我们该如何解决这个问题呢？

比如对话摘要记忆组件就提供了一个解决方案。这种记忆模式会在对话进行的过程中实时总结对话内容，并将当前的摘要存储在记忆中。然后，这些记忆可以被用于将迄今为止的对话摘要注入到提示或链中。这种记忆方式对于长期的对话来说尤其有用，因为如果直接在提示中保留过去的消息历史，可能会占用太多的令牌。

这就是记忆组件在LangChain中的运作方式。

在接下来的部分中，我们主要会对比和探讨几种不同的记忆类型，但这会涉及到链和代理的应用。如果你对链和代理的概念不熟悉，你可能会觉得难以理解。因此，建议你在深入了解这两个概念之后再继续阅读下文。