Ch 05. Memory

상태	완료
최종 편집 일시	@2025년 11월 22일 오후 7:33

https://wikidocs.net/233801

01. 대화 버퍼 메모리(ConversationBufferMemory)
02. 대화 버퍼 윈도우 메모리(ConversationBufferWindowMemory)
03. 대화 토큰 버퍼 메모리(ConversationTokenBufferMemory)
04. 대화 엔티티 메모리(ConversationEntityMemory)
05. 대화 지식그래프 메모리 (ConversationKGMemory)
06. 대화 요약 메모리(ConversationSummaryMemory)
07. 벡터저장소 검색 메모리(VectorStoreRetrieverMemory)
08. LCEL Chain 에 메모리 추가
09. SQLite 에 대화내용 저장
10. RunnableWithMessageHistory에 ChatMessageHistory추가

01. 대화 버퍼 메모리 (ConversationBufferMemory)

> 지금까지 나누었던 대화를 기록하고 관리할 수 있다.

`save_context(inputs, outputs)` 메서드를 사용하여 대화 기록을 저장할 수 있습니다.

- 이 메서드는 `inputs`와 `outputs` 두 개의 인자를 받습니다.
- `inputs`는 사용자의 입력을, `outputs`는 AI의 출력을 저장합니다.
- 이 메서드를 사용하면 대화 기록이 `history` 키에 저장됩니다.
- 이후 `load_memory_variables` 메서드를 사용하여 저장된 대화 기록을 확인할 수 있습니다.

---

02. 대화 버퍼 윈도우 메모리 (ConversationBufferWindowMemory)

> Window를 활용해 전체 상호작용이 아닌 최근의 상호작용만으로 한정지어 관리할 수 있다

`ConversationBufferWindowMemory` 는 시간이 지남에 따라 대화의 상호작용 목록을 유지합니다.

이때, `ConversationBufferWindowMemory` 는 모든 대화내용을 활용하는 것이 아닌 **최근 K개** 의 상호작용만 사용합니다.

이는 버퍼가 너무 커지지 않도록 가장 최근 상호작용의 슬라이딩 창을 유지하는 데 유용할 수 있습니다.

```
memory = ConversationBufferWindowMemory(k=2, return_messages=True)
```

---

03. 대화 토큰 버퍼 메모리 (ConversationTokenBufferMemory)

`ConversationTokenBufferMemory` 는 최근 대화의 히스토리를 버퍼를 메모리에 보관하고, 대화의 개수가 아닌 **토큰 길이** 를 사용하여 대화내용을 플러시(flush)할 시기를 결정합니다.

> Token 단위의 관리가 진행되기 때문에 저장 관리 및 출력 용량에 대한 세심한 관리가 가능해보인다.

---

04. 대화 엔티티 메모리 (ConversationEntityMemory)

엔티티 메모리는 대화에서 특정 엔티티에 대한 주어진 사실을 기억합니다.

엔티티 메모리는 엔티티에 대한 정보를 추출하고(LLM 사용) 시간이 지남에 따라 해당 엔티티에 대한 지식을 축적합니다(역시 LLM 사용).

> 따라서 엔티티에 대한 정보를 추출하는 과정이 추가되었기에 실행시 발생하는 컴퓨팅 자원 소모는 앞선 세 메모리 방식과는 다르게 커지겠지만, 실제 저장 단계에서는 사용자의 의도에 따라 특정 주제에 대한 정보만을 골라 저장할 수 있다는 점에서 의의가 있는 것 같다.

대화 엔티티 메모리를 사용하는 프롬프트 본문

You are an assistant to a human, powered by a large language model trained by OpenAI.You are designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discussions on a wide range of topics. As a language model, you are able to generate human-like text based on the input you receive, allowing you to engage in natural-sounding conversations and provide responses that are coherent and relevant to the topic at hand.

You are constantly learning and improving, and your capabilities are constantly evolving. You are able to process and understand large amounts of text, and can use this knowledge to provide accurate and informative responses to a wide range of questions. You have access to some personalized information provided by the human in the Context section below. Additionally, you are able to generate your own text based on the input you receive, allowing you to engage in discussions and provide explanations and descriptions on a wide range of topics.

Overall, you are a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range of topics. Whether the human needs help with a specific question or just wants to have a conversation about a particular topic, you are here to assist.

Context:
{entities}

Current conversation:
{history}
Last line:
Human: {input}
You:

---

05. 대화 지식그래프 메모리 (ConversationKGMemory)

> KG는 Knowledge Graph의 약자처럼 보인다

> `ConversationEntityMemory` **는** 엔티티 그 자체에 대한 정보를 수집한 뒤 저장했다면, KGMemory는 엔티티간의 관계에 초점을 맞춰 저장을 하는 방식이다. 엔티티 자체에 대한 저장의 상세성은 떨어져보이지만 관계성에 대해 파악할 때는 유용하게 쓰일 것 같다.

---

06. 대화 요약 메모리 (ConversationSummaryMemory)

> 이 메모리는 대화를 그대로 저장하면 발생할 수 있는 과도한 메모리사용에 초점을 맞춘 듯하다. 따라서 어떤 특정 주제나 관계성 보다는 대화 자체의 요약본을 저장하는데 초점을 맞추었다

`ConversationSummaryBufferMemory` 는 요약본을 저장하는 전략과 최근의 내용은 그대로 저장함으로써 버퍼의 사용 관리와 최근 대화의 상세성을 동시에 유지하려한 모습을 보인다. 최근의 기준을 정하기 위해서 상호작용의 개수가 아닌 **토큰 길이**를 사용한다.

---

07. 벡터저장소 검색 메모리 (VectorStoreRetrieverMemory)

`VectorStoreRetrieverMemory` 는 벡터 스토어에 메모리를 저장하고 호출될 때마다 가장 '눈에 띄는' 상위 K개의 문서를 쿼리합니다.

이는 대화내용의 순서를 **명시적으로 추적하지 않는다는 점** 에서 다른 대부분의 메모리 클래스와 다릅니다.

> 엔티티메모리처럼 특정 기준을 저장 과정 자체에 도입한다라는 점에서는 공통분모를 가지고 있지만 이 메모리는 벡터 스토어에 저장한다는 점, 호출에도 가장 의미가 있는 상위의 문서를 쿼리한다는 점에서 의의가 있다

→ 하지만 연관성에 대한 검증이 있는 만큼 다른 메모리와는 출력 속도에서 조금 떨어지는 모습ㅂ을 보인다.

---

08. LCEL Chain 에 메모리 추가

> 실제로 사용하는 chain에 메모리를 추가하는 방식으로 구성해볼 수도 있다.

대화내용을 저장할 메모리인 `ConversationBufferMemory` 생성하고 `return_messages` 매개변수를 `True`로 설정하여, 생성된 인스턴스가 메시지를 반환하도록 합니다.

- `memory_key` 설정: 추후 Chain 의 `prompt` 안에 대입될 key 입니다. 변경하여 사용할 수 있습니다.

> 메모리를 붙였기 때문에 체인으로 invoke되는 내역은 자동으로 저장이 된다. 또한 이 저장 과정에서 임의로 커스텀을 통해 조건문 컨트롤이 가능해보인다.

---

09. SQLite 에 대화내용 저장

> 꽤나 다양한 플랫폼에서 사용가능한 SQLite를 활용해 데이터 베이스에 채팅 기록을 저장하는 방식이다.

---

10. RunnableWithMessageHistory에 ChatMessageHistory추가