

대화 메모리 관리




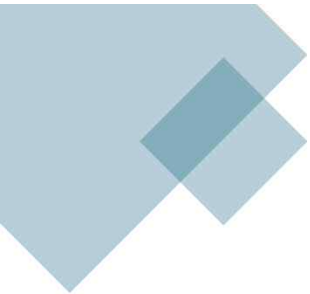
1. 대화 메모리 관리 개요



대화 메모리 관리 (Conversation Memory Management)

◆ 메모리의 개념

- ****대화 메모리(MessageHistory)****는 AI Agent가 이전 대화 내용을 메시지 단위로 저장하고, 이를 바탕으로 ****맥락(Context)****을 유지하여 자연스러운 대화를 이어가기 위한 구조이다.
 - 단순한 “대화 이력 저장”을 넘어서, ****대화 요약(Summarization)****과 **의도 추적(Context Tracking)** 기능을 포함한다.
- 



◆ 메모리의 필요성

구분	설명	예시
맥락 유지	세션별 대화를 기억하여 연결된 대화 가능	“내가 어제 말한 계획 기억하지?”
정보 재활용	과거의 답변이나 사실을 다시 참조 가능	“그때 추천한 식당 이름이 뭐였지?”
개인화 응답	사용자 이름, 선호, 스타일 반영	“다음에도 영어로 번역해줄게요.”
대화 품질 향상	반복 질문 감소, 응답의 자연스러움 증가	대화 흐름이 인간처럼 이어짐



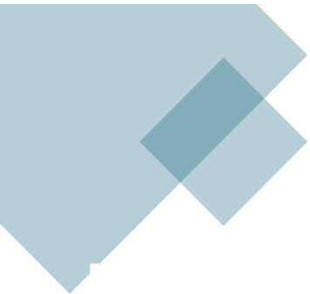
메모리의 주요 유형

(구버전 클래스: LangChain \geq 0.2.7 신버전에서 Deprectaed)

유형	특징	예시 클래스 (LangChain)
Buffer Memory	모든 대화 이력을 그대로 저장	<code>ConversationBufferMemory</code>
Buffer Window Memory	최근 n개의 대화만 저장	<code>ConversationBufferWindowMemory</code>
Summary Memory	이전 대화를 요약 형태로 저장	<code>ConversationSummaryMemory</code>
Entity Memory	인물·객체별 정보 기억	<code>ConversationEntityMemory</code>
Combined Memory	위 여러 방식을 조합	커스텀 메모리 구현 가능


◆ 구버전과 신버전 차이 (LangChain ≥ 0.2.7)

항목	구버전 (≤0.2.6)	신버전 (≥0.2.7)
구조	ConversationChain + Memory	RunnableWithMessageHistory + MessageHistory
저장 단위	단순 문자열	메시지 객체 리스트
세션 구분	직접 관리	<code>session_id</code> 기반 자동 관리
확장성	제한적	고확장성 (멀티 세션 + 외부 저장)
상태	Deprecated	✅ 표준 방식



◆ 최신 메모리 관리 구조

항목	설명
핵심 클래스	<code>RunnableWithMessageHistory</code>
대화 이력 저장소	<code>InMemoryChatMessageHistory</code> , <code>FileChatMessageHistory</code> , <code>RedisChatMessageHistory</code> , <code>SQLChatMessageHistory</code>
세션 관리	<code>session_id</code> 기반으로 사용자별 대화 분리
외부 저장 연계	Redis, SQL DB 등을 사용해 장기 저장 가능





2. RunnableWithMessageHistory





RunnableWithMessageHistory란?

- LangChain의 **Runnable** 인터페이스를 확장한 객체
 - LLM(예: ChatOpenAI)과 ****대화 기록(MessageHistory)****을 결합해서 **사용자와의 멀티턴 대화**를 관리한다.
 - 기존 **Memory 시스템(ConversationBufferMemory 등)**을 대체하며, LangChain 1.0에서는 **표준 방식**으로 자리잡는다.
- 



주요 특징

특징	설명
대화 히스토리 자동 관리	이전 사용자 입력과 모델 응답을 자동으로 기록
세션 구분	<code>session_id</code> 로 사용자별 대화 이력 분리 가능
저장소 선택 가능	InMemory, File, Redis, SQL 등 다양한 히스토리 백엔드 지원
Runnable 호환	모든 LangChain Runnable 체인과 동일한 방식으로 작동
Deprecated 대체	<code>ConversationChain</code> + <code>ConversationBufferMemory</code> 를 완전히 대체

📦 구성 요소

1. LLM 모델

- 예: `ChatOpenAI(model="gpt-4o-mini")`

2. `get_session_history` 함수

- 세션 ID별로 어떤 히스토리 저장소(`InMemoryChatMessageHistory`, `RedisChatMessageHistory` 등)를 사용할지 결정하는 콜백 함수

3. `MessageHistory` 클래스


- 실제 대화 이력을 저장하는 객체 (InMemory / File / Redis / SQL 지원)

4. `Configurable Session`

- `config = {"configurable": {"session_id": "chat-001"}}`
- 세션별 기록을 자동으로 불러오고 저장할 수 있음




기존 ConversationChain과 비교

항목	ConversationChain (구버전)	RunnableWithMessageHistory (신버전) 
핵심 클래스	ConversationChain	RunnableWithMessageHistory
메모리 관리	ConversationBufferMemory	ChatMessageHistory
세션 관리	직접 구현 필요	session_id 자동 지원
저장소 확장성	제한적	InMemory, File, Redis, SQL 지원
상태	Deprecated	✅ 최신 표준

3. InMemoryChatMessageHistory




🧠 InMemoryChatMessageHistory란?

- 대화 기록(Chat Messages)을 메모리(RAM)에 저장하는 기본 클래스
 - 한 세션(session_id) 동안 사용자 메시지와 AI 응답을 순차적으로 리스트 형태로 보관
 - 프로그램이 종료되면 기록이 사라지는 **휘발성(Volatile)** 구조
 - `RunnableWithMessageHistory` 와 함께 사용해 멀티턴 대화 관리에 활용됨
- 



🔑 주요 특징

특징	설명
휘발성 저장소	데이터를 메모리에만 저장 → 프로그램 종료 시 삭제됨
간단하고 빠름	추가 설정 불필요, 바로 사용 가능
실습/테스트에 적합	챗봇 프로토타입, 실습 코드에서 많이 사용
메시지 기반 저장	<code>HumanMessage</code> , <code>AIMessage</code> , <code>SystemMessage</code> 등 LangChain 메시지 객체 리스트로 저장
세션 구분 없음 (단독 사용시)	<code>RunnableWithMessageHistory</code> 와 함께 쓰면 <code>session_id</code> 별 기록 관리 가능



📦 기본 사용법

```
from langchain_core.chat_history import InMemoryChatMessageHistory
from langchain_core.messages import HumanMessage, AIMessage

# 1) 메모리 객체 생성
history = InMemoryChatMessageHistory()

# 2) 사용자 메시지 추가
history.add_message(HumanMessage(content="안녕?"))

# 3) AI 응답 추가
history.add_message(AIMessage(content="안녕하세요! 만나서 반가워요."))

# 4) 기록 확인
for msg in history.messages:
    role = "👤 사용자" if msg.type == "human" else "🤖 AI"
    print(f"{role}: {msg.content}")
```

🧩 실행 예시

👤 사용자: 안녕?

🤖 AI: 안녕하세요! 만나서 반가워요.



🏛️ 다른 MessageHistory 클래스와 비교

클래스	저장 위치	특징	사용 사례
InMemoryChatMessageHistory	메모리(RAM)	휘발성, 빠름	프로토타입, 실습
FileChatMessageHistory	로컬 파일(JSON)	지속성 있음	간단한 로컬 앱
RedisChatMessageHistory	Redis 서버	네트워크 공유 가능	멀티유저 챗봇, 서버
SQLChatMessageHistory	DB(SQLite/MySQL 등)	영속적 저장	서비스 운영 환경




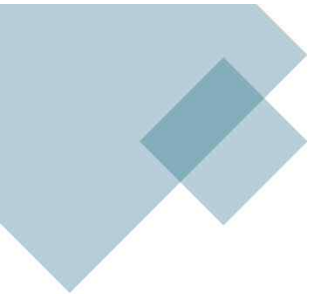
4. FileChatMessageHistory




FileChatMessageHistory란?

“대화 이력을 로컬 파일(JSON)로 저장하는 클래스”

- `InMemoryChatMessageHistory` 가 ****RAM(메모리)****에 저장하는 반면, `FileChatMessageHistory` 는 ****파일(디스크)****에 기록합니다.
 - 프로그램이 종료되어도 기록이 유지되므로 ****영속성(Persistence)****이 필요한 상황에 적합합니다.
 - `RunnableWithMessageHistory` 와 함께 사용하면 **세션별 대화 로그 자동 저장 챗봇**을 쉽게 만들 수 있습니다.
- 



주요 특징

항목	설명	
저장 위치	로컬 파일(JSON)	
지속성	프로그램 종료 후에도 대화 기록 유지	
저장 형식	<pre>[{"type": "human", "content": "..."}, {"type": "ai", "content": "..."}]</pre>	
세션 구분	파일 경로를 세션별로 다르게 지정 가능	
활용 예시	챗봇 로그 기록, 사용자별 대화 저장, 학습 데이터 수집	



📦 기본 사용법

```
from langchain_community.chat_message_histories import FileChatMessageHistory
from langchain_core.messages import HumanMessage, AIMessage

# 1) FileChatMessageHistory 생성
history = FileChatMessageHistory("chat_history.json")

# 2) 메시지 추가
history.add_message(HumanMessage(content="안녕?"))
history.add_message(AIMessage(content="안녕하세요! 만나서 반가워요."))

# 3) 저장 내용 확인
for msg in history.messages:
    role = "👤 사용자" if msg.type == "human" else "🤖 AI"
    print(f"{role}: {msg.content}")
```



결과 예시

👤 사용자: 안녕?

🤖 AI: 안녕하세요! 만나서 반가워요.

그리고 로컬 디렉터리에 **chat_history.json** 파일이 생성됩니다 👉

```
[  
  {"type": "human", "content": "안녕?"},  
  {"type": "ai", "content": "안녕하세요! 만나서 반가워요."}  
]
```

⚖️ 다른 MessageHistory 클래스와 비교

클래스	저장 위치	지속성	특징	용도
InMemoryChatMessageHistory	메모리(RAM)	❌ 휘발성	빠르고 간단	테스트, 단기 세션
FileChatMessageHistory	파일(JSON)	✅ 영속적	자동 저장/불러오기	개인 챗봇, 로그 관리
RedisChatMessageHistory	Redis 서버	✅ 영속 + 공유	네트워크 공유 가능	다중 사용자 환경
SQLChatMessageHistory	DB(SQLite, MySQL 등)	✅ 영속 + 질의 가능	대규모 서비스	상용 환경

5.RedisChatMessageHistory



Redis란? 메모리(RAM)에 데이터를 저장하는 초고속 데이터베이스

항목	설명
정식 명칭	REmote DIctionary Server
종류	인메모리(In-Memory) 데이터베이스
저장 위치	디스크가 아닌 메모리(RAM)
특징	매우 빠른 읽기/쓰기 속도
형태	key-value 구조의 NoSQL 데이터베이스

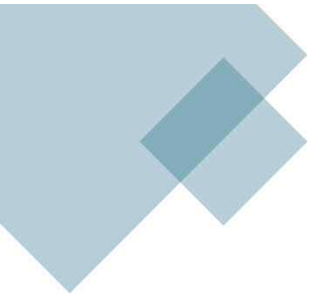




Redis의 장점

장점	설명
 빠름	디스크가 아닌 메모리에 저장되어 읽기/쓰기 속도가 매우 빠름
 세션 유지	로그인, 채팅, 대화 이력 등 빠르게 갱신되는 데이터에 적합
 데이터 공유 용이	여러 프로그램/사용자가 동시에 접근 가능
 구조 단순	key-value 구조라 사용하기 쉬움
 옵션으로 영구 저장도 가능	RDB, AOF 설정으로 디스크 백업 지원





Redis를 어디에 쓰나?

분야

예시

AI 챗봇

대화 이력 저장 (`RedisChatMessageHistory`)

웹 서비스

로그인 세션, 쿠키, 사용자 상태 저장

실시간 시스템

실시간 순위, 채팅방, 게임 서버

캐시(Cache)

자주 쓰는 데이터를 DB 대신 메모리에 저장





Windows 환경에서 Redis 설치 방법

<https://github.com/microsoftarchive/redis/releases>

공식 Redis는 리눅스용이지만,
Microsoft가 빌드한 Windows용 포트 버전이 존재합니다.

1 설치 링크

<https://github.com/microsoftarchive/redis/releases>

2 예: "Redis-x64-3.2.100.msi" 다운로드 후 설치

3 설치 후 실행

```
bash
```

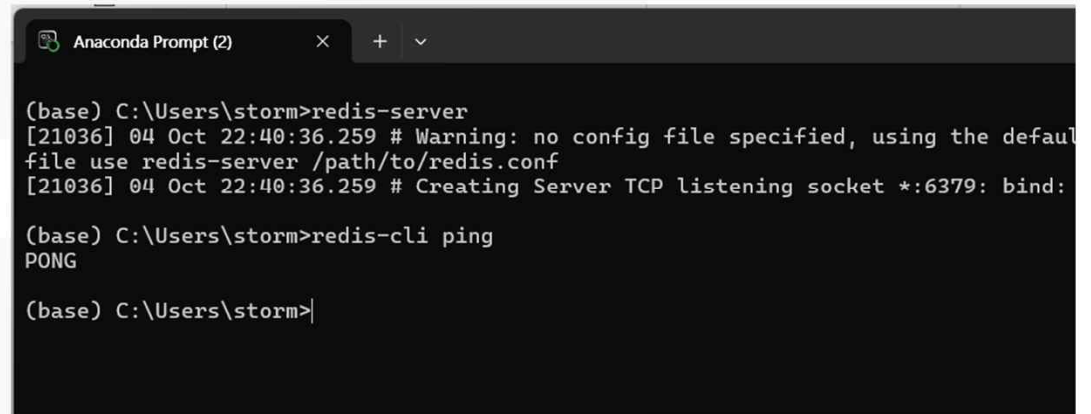
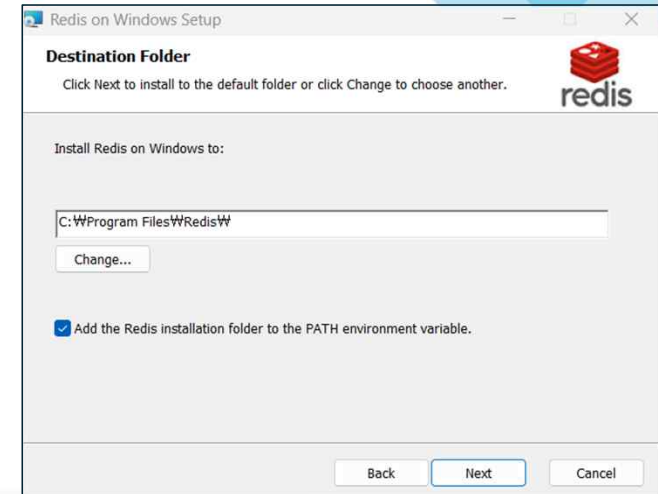
```
redis-server
```

4 접속 테스트

```
bash
```

```
redis-cli ping
```

```
# PONG
```



RedisChatMessageHistory란?

항목	설명
역할	Redis 서버를 이용해 대화 메시지를 저장하고 불러옴
특징	네트워크 기반이므로 다중 사용자, 서버 환경에 적합
저장 형태	Redis key-value 쌍으로 JSON 직렬화된 메시지 리스트
필요 조건	Redis 서버가 실행 중이어야 함 (<code>redis-server</code> 프로세스 실행 필요)

메모리에 휘발되는 InMemoryChatMessageHistory 대신
Redis 서버에 대화 내용을 저장해서 프로그램을 꺼도,
혹은 여러 사용자가 동시에 접속해도 이전 대화가 유지되는 구조.

🗨 기본 예제 — RedisChatMessageHistory 단독 사용

```
from langchain_community.chat_message_histories import RedisChatMessageHistory
from langchain_core.messages import HumanMessage, AIMessage

# 1) Redis 서버 연결 설정
# host, port, session_id는 자유롭게 변경 가능
history = RedisChatMessageHistory(
    url="redis://localhost:6379/0", # Redis 서버 주소
    session_id="user-001"         # 세션 구분용 ID
)

# 2) 메시지 추가
history.add_message(HumanMessage(content="안녕?"))
history.add_message(AIMessage(content="안녕하세요! 반가워요 😊"))

# 3) 현재 대화 이력 확인
print("=== Redis 저장된 대화 이력 ===")
for msg in history.messages:
    role = "👤 사용자" if msg.type == "human" else "🤖 AI"
    print(f"{role}: {msg.content}")
```

Redis 내부 저장 예시

Redis에 다음과 같이 저장됩니다 📌

```
KEY: chat_history:user-001
```

```
VALUE: [
```

```
  {"type": "human", "content": "안녕?"},
```

```
  {"type": "ai", "content": "안녕하세요! 반가워요 😊"}]
```

Redis 데이터 삭제

```
import redis

# Redis 연결
r = redis.Redis(host='localhost', port=6379, db=0)

# 전체 삭제
r.flushall() # 모든 데이터 삭제

# 현재 DB만 삭제
r.flushdb() # 기본적으로 db=0이므로 동일

# 특정 세션만 삭제
r.delete("chat_history:user-001")

print("Redis 데이터 삭제 완료!")
```


Redis 메모리 내용 삭제 방법 요약

목적	명령	설명
① 전체 데이터 삭제	<code>FLUSHALL</code>	Redis 전체 DB의 모든 키 삭제 (⚠️ 완전 초기화)
② 현재 DB만 삭제	<code>FLUSHDB</code>	선택된 DB(보통 0번 DB)의 데이터만 삭제
③ 특정 키만 삭제	<code>DEL <키 이름></code>	지정된 키만 제거
④ LangChain 세션만 삭제	<code>DEL chat_history:<session_id></code>	특정 세션 대화 기록만 삭제

```
chat_history:user-001
chat_history:chat-redis-001
```

6. SQLAlchemyMessageHistory

SQLChatMessageHistory란?

대화(Message) 기록을 SQL 데이터베이스에 저장하는 LangChain의 메모리 클래스

항목	설명
역할	대화 이력을 SQLite, MySQL, PostgreSQL 등 SQL DB에 저장
소속 모듈	<code>langchain_community.chat_message_histories</code>
데이터 구조	<code>session_id</code> , <code>message_type</code> , <code>content</code> , <code>timestamp</code>
특징	휘발성 아님 → 프로그램 꺼져도 기록 유지
활용 예시	챗봇 로그 저장, 사용자별 대화 분석, 장기 세션 관리

🌿 기본 예제 — SQLite 기반

python


```
from langchain_community.chat_message_histories import SQLChatMessageHistory
from langchain_core.messages import HumanMessage, AIMessage

# 1 데이터베이스 파일 지정
db_url = "sqlite:///chat_history.db" # 로컬 SQLite 파일
session_id = "user-001" # 세션별로 구분 가능

# 2 SQLChatMessageHistory 생성
history = SQLChatMessageHistory(session_id=session_id, connection_string=db_url)

# 3 메시지 추가
history.add_message(HumanMessage(content="안녕?"))
history.add_message(AIMessage(content="안녕하세요! 만나서 반가워요 😊"))

# 4 저장된 대화 이력 확인
print("=== SQL 저장된 대화 이력 ===")
for msg in history.messages:
    role = "👤 사용자" if msg.type == "human" else "🤖 AI"
    print(f"{role}: {msg.content}")
```



저장 구조 (SQLite 예시)

id	session_id	message_type	content	timestamp
1	user-001	human	안녕?	2025-10-04 13:00:22
2	user-001	ai	안녕하세요! 만나서 반가워요 😊	2025-10-04 13:00:23





다양한 DB 지원 예시

DB 종류

connection_string 예시

SQLite

```
sqlite:///chat_history.db
```

MySQL

```
mysql+pymysql://user:password@localhost:3306/chat_db
```


PostgreSQL

```
postgresql+psycopg2://user:password@localhost:5432/chat_db
```




⚖️ 다른 MessageHistory 클래스와 비교

클래스	저장 위치	지속성	특징	사용 예시
InMemoryChatMessageHistory	RAM	❌ 휘발성	빠름	테스트용
FileChatMessageHistory	로컬 파일(JSON)	✅ 영구	간단	개인 챗봇
RedisChatMessageHistory	Redis 서버	✅ 영구+공유	네트워크 공유	웹 챗봇
SQLChatMessageHistory	SQL DB	✅ 영구+쿼리 가능	데이터 분석, 로그	상용 서비스



✓ 정리 요약

항목	내용
클래스명	SQLChatMessageHistory
저장 위치	SQLite/MySQL/PostgreSQL 등 DB
특징	영구 저장 + SQL 쿼리 가능
장점	안정성, 확장성, 분석 용이
단점	I/O 속도 느림 (InMemory보다)
추천 용도	챗봇 로그, 사용자별 히스토리 관리, 장기 세션 서비스



🧠 1 특정 세션 대화만 삭제

`SQLChatMessageHistory` 객체에는 내부적으로 `session_id` 로 대화를 구분합니다.
따라서, 특정 세션의 모든 메시지를 지우려면 다음과 같이 하면 됩니다 🗑️

python

```
from langchain_community.chat_message_histories import SQLChatMessageHistory
from sqlalchemy import create_engine

# 1) DB 연결
engine = create_engine("sqlite:///chat_history.db")

# 2) 세션 ID 지정
session_id = "chat-sql-001"

# 3) 해당 세션의 히스토리 불러오기
history = SQLChatMessageHistory(session_id=session_id, connection=engine)

# 4) 대화 전체 삭제
history.clear() # ✅ 이 한 줄로 세션 내 모든 메시지 삭제

print(f"✅ 세션 {session_id}의 대화 내용이 모두 삭제되었습니다.")
```

이 명령은 SQL 테이블에서 해당 `session_id` 의 메시지 행(row)을 전부 삭제합니다.

🔧 2 데이터베이스 전체 초기화 (모든 세션 삭제)

만약 모든 세션의 대화를 완전히 지우고 싶다면,
SQLAlchemy를 이용해서 테이블 전체를 비우면 됩니다.

python

```
from sqlalchemy import create_engine, text

# 1) DB 연결
engine = create_engine("sqlite:///chat_history.db")

# 2) 모든 메시지 삭제 (테이블 전체 비움)
with engine.connect() as conn:
    conn.execute(text("DELETE FROM message_store")) # ✅ 모든 메시지 행 삭제
    conn.commit()

print("✅ DB의 모든 대화 내용이 삭제되었습니다.")
```

💡 `message_store` 는 LangChain이 자동 생성하는 테이블 이름입니다.
(DB를 처음 실행했을 때 자동으로 만들어집니다.)

🔧 3 테이블 자체를 완전히 제거 (드롭)

DB 파일을 완전히 초기화하고 싶다면 테이블을 아예 삭제(drop) 할 수도 있습니다 🙋

python

```
from sqlalchemy import create_engine, text

engine = create_engine("sqlite:///chat_history.db")

with engine.connect() as conn:
    conn.execute(text("DROP TABLE IF EXISTS message_store"))
    conn.commit()

print("✅ message_store 테이블이 완전히 삭제되었습니다.")
```



감사합니다