
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

8. `RunnableWithMessageHistory`

1) `메시지 기록` *(메모리)* `추가하기`

- `RunnableWithMessageHistory`
  - `대화형 애플리케이션` or `복잡한 데이터 처리 작업`을 구현할 때 `이전 메시지`의 `맥락`을 `유지`해야 할 필요가 있을 때 중요
  - 메시지 기록 관리 → 개발자는 **`애플리케이션`의 `흐름`을 더 잘 `제어`하고, `사용자`의 `이전 요청`에 따라 `적절`하게 `응답` 가능**

- `실제 활용 예시`
  - **`대화형 챗봇 개발`**: 사용자와의 `대화 내역`을 기반으로 챗봇의 응답 조정 가능
  - **`복잡한 데이터 처리`**: 데이터 처리 과정에서 `이전 단계`의 `결과`를 `참조`하여 다음 단계의 로직 결정 가능
  - **`상태 관리`가 `필요`한 `애플리케이션`**: `사용자`의 `이전 선택`을 `기억`하고 그에 따라 다음 화면이나 정보 제공 가능

- `RunnableWithMessageHistory` 유용성
  - 애플리케이션의 상태 유지
  - 사용자 경험 향상
  - 더 정교한 응답 메커니즘 구현알 수 있게 해주는 강력한 도구

- **환경 설정**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv

# API 키 정보 로드
load_dotenv()                           # True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
```

```
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

---

2) 메모리 저장 방법

- **필요한 주요 요소**
  - ① **Runnable**: 주로 `Retriever`, `Chain`과 같이 `BaseChatMessageHistory`와 상호작용하는 `runnable` 객체
  - ② **BaseChatMessageHistory의 인스턴스를 반환하는 호출 가능한 객체(`callable`)**: 메시지 기록을 관리하기 위한 객체
    - 메시지 기록을 저장, 검색, 업데이트하는 데 사용
    - 대화의 맥락 유지, 사용자의 이전 입력에 기반한 응답을 생성하는 데 필요
- *참고: memory integrations*

- 코드 예시

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_openai import ChatOpenAI

model = ChatOpenAI()

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 {ability} 에 능숙한 어시스턴트입니다. 20자 이내로 응답하세요",
        ),
        # 대화 기록을 변수로 사용, history 가 MessageHistory 의 key 가 됨
        MessagesPlaceholder(variable_name="history"),
        ("human", "{input}"),       # 사용자 입력을 변수로 사용
    ]
)

runnable = prompt | model           # 프롬프트와 모델을 연결하여 runnable 객체 생성
```

- **주요 방법**
  - ① **인메모리 `ChatMemoryHistory` 사용**
    - 메모리 내에서 메시지 기록을 관리
    - *주로 개발 단계 or 간단한 애플리케이션에서 사용됨*
    - 장점: 빠른 접근 속도 제공
    - 단점: 애플리케이션 재시작 시 메시지 기록 사라짐
  - ② **`RedisChatMessageHistory` → 영구적인 저장소 활용**
    - 높은 성능을 제공하는 오픈 소스 인메모리 데이터 구조 저장소
    - 분산 환경에서도 안정적으로 메시지 기록 관리 가능
    - 복잡한 애플리케이션 or 장기간 운영되는 서비스에 적합

*

- *방법 선택 기준: 애플리케이션의 요구사항, 예상되는 트래픽 양, 메시지 데이터의 중요성 및 보존 기간 등*
  - ① *인메모리*: 구현이 간단하고 빠름
  - ② *영구저장소*: 데이터의 영구성이 요구되는 경우

---

3) 휘발성 대화기록 : `In-Memory`

- **`RunnableWithMessageHistory`** 설정 매개 변수
  - **`runnable`**
  - **`BaseChatMessageHistory`** 이거나 `상속받은 객체`
    - *예시:* *`ChatMessageHistory`*
  - **`input_messages_key`** : `chain.invoke()` 할 때 → 사용자 쿼리 `입력`으로 지정하는 `key`
  - **`history_messages_key`** : `대화 기록`으로 `지정`하는 `key`

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import chain
from langchain_google_genai import ChatGoogleGenerativeAI

from dotenv import load_dotenv
import os

# LLM 초기화
# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 생성하기
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0,
)
```

- 기본 `LLM` 생성하기 (`gemini_lc`) - `gemini-2.5.flash-lite`

```
E0000 00:00:1760007057.287445 2240258 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 {ability} 에 능숙한 어시스턴트입니다. 20자 이내로 응답하세요",
        ),
        # 대화 기록을 변수로 사용, history 가 MessageHistory 의 key 가 됨
        MessagesPlaceholder(variable_name="history"),
        ("human", "{input}"),                       # 사용자 입력을 변수로 사용
    ]
)

runnable = prompt | gemini_lc                       # 프롬프트와 모델을 연결하여 runnable 객체 생성
```

```
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

# 세션 기록을 저장할 딕셔너리
store = {}


# 세션 ID를 기반으로 세션 기록을 가져오는 함수
def get_session_history(session_ids: str) -> BaseChatMessageHistory:
    print(session_ids)
    if session_ids not in store:                    # 세션 ID가 store에 없는 경우
        # 새로운 ChatMessageHistory 객체를 생성하여 store에 저장
        store[session_ids] = ChatMessageHistory()
    return store[session_ids]                       # 해당 세션 ID에 대한 세션 기록 반환


with_message_history = (
    RunnableWithMessageHistory(                     # RunnableWithMessageHistory 객체 생성
        runnable,                                   # 실행할 Runnable 객체
        get_session_history,                        # 세션 기록을 가져오는 함수 (직전에 정의한 함수)
        input_messages_key="input",                 # 입력 메시지의 키
        history_messages_key="history",             # 기록 메시지의 키
    )
)
```

- **`input_message_key`** = 최신 입력 메시지로 처리될 키를 지정함
- **`history_message_key`** = 이전 메시지를 추가할 키를 지정함

- **`RunnableWithMessageHistory`**
  - 초기값: **`session_id`** **키** = `Default`
  - **`대화 스레드별 관리`** = **`RunnableWithMessageHistory`** 가 대화 스레드를 **`session_id`** 로 관리함

```
    # 참고 코드

    if history_factory_config:
        _config_specs = history_factory_config

    else:
        # If not provided, then we'll use the default session_id field
        _config_specs = [
            ConfigurableFieldSpec(
                id="session_id",
                annotation=str,
                name="Session ID",
                description="Unique identifier for a session.",
                default="",
                is_shared=True,
            ),
        ]
```

- **`invoke()`** → **`config={"configurable": {"session_id": "세션ID입력"}}`** 코드 반드시 지정하기

```
# query_1
with_message_history.invoke(
    # 수학 관련 질문, "코사인의 의미는 무엇인가요?"를 입력으로 전달
    {"ability": "math", "input": "What does cosine mean?"},        # "What does cosine mean?"
    # 설정 정보로 세션 ID "abc123"을 전달
    config={"configurable": {"session_id": "abc123"}},
)
```

- query_1 - (`1.1s`)
- `abc123`

```
AIMessage(content='Ratio of adjacent to hypotenuse.', additional_kwargs={}, response_metadata={'prompt_feedback':
```

- **같은 `session_id` 입력 → `이전 대화 스레드`의 `내용`을 가져옴 → 이어서 대화 가능**

```
# 메시지 기록 포함해 호출하기 (query_2)

with_message_history.invoke(
    # ability, input을 설정하기
    {"ability": "math", "input": "이전의 내용을 한글로 답변해 주세요."},
    # 설정 옵션 지정하기
    config={"configurable": {"session_id": "abc123"}},          # 같은 session_id
)
```

- query_2 - (`0.7s`)
- `abc123`

```
AIMessage(content='직각삼각형의 빗변 대비 인접변 길이', additional_kwargs={}, response_metadata={'prompt_feedback': {'blo
```

- **다른 `session_id` 지정** → 대화 기록 ❌ → 답변 제대로 수행 ❌
  - *아래의 예시:* `session_id` : `def234` → *존재* ❌ → *엉뚱한 답변*

```
# query_3
with_message_history.invoke(
    {"ability": "math", "input": "이전의 내용을 한글로 답변해 주세요"},
    # 새로운 session_id 설정하기
    # 새로운 session_id로 인해 이전 대화 내용을 기억하지 못함
    config={"configurable": {"session_id": "def234"}},
)
```

- `query_3` - (`0.6s`)
- `def234`

```
AIMessage(content='네, 이전 내용을 한글로 답변해 드리겠습니다.', additional_kwargs={}, response_metadata={'prompt_feedback'
```

- **`ConfigurableFieldSpec`** 객체 리스트 → **`history_factory_config`** 매개변수로 전달 → 사용자 정의 가능
  - **`history_factory_config` 새로 설정 → 기존의 `session_id` 설정을 덮어쓰게 됨**

```
from langchain_core.runnables import ConfigurableFieldSpec

store = {}                                                  # 빈 딕셔너리 초기화


def get_session_history(user_id: str, conversation_id: str) -> BaseChatMessageHistory:
    # 주어진 user_id와 conversation_id에 해당하는 세션 기록 반환하기
    if (user_id, conversation_id) not in store:
        # 해당 키가 store에 없으면 새로운 ChatMessageHistory를 생성하여 저장하기
        store[(user_id, conversation_id)] = ChatMessageHistory()
    return store[(user_id, conversation_id)]


# RunnableWithMessageHistory 객체 생성
with_message_history = RunnableWithMessageHistory(
    runnable,                                               # 실행할 Runnable 객체
    get_session_history,                                    # 세션 기록을 가져오는 함수 (이전 생성)
    input_messages_key="input",                             # 입력 메시지의 키
    history_messages_key="history",                         # 기록 메시지의 키

    # 기존의 "session_id" 설정 대체하기
    history_factory_config=[
        # user_id
        ConfigurableFieldSpec(
            id="user_id",                                   # get_session_history 함수의 첫 번째 인자로 사용됨
            annotation=str,
            name="User ID",
            description="사용자의 고유 식별자입니다.",
            default="",
            is_shared=True,
        ),

        # conversation_id
        ConfigurableFieldSpec(
            id="conversation_id",                           # get_session_history 함수의 두 번째 인자로 사용됨
            annotation=str,
            name="Conversation ID",
            description="대화의 고유 식별자입니다.",
            default="",
            is_shared=True,
        ),
    ],
)
```

```
# query_4
with_message_history.invoke(
    {"ability": "math", "input": "Hello"},
    config={"configurable": {"user_id": "123", "conversation_id": "1"}},
)
```

- `query_4` - (`0.7s`)

```
AIMessage(content='안녕하세요!', additional_kwargs={}, response_metadata={'prompt_feedback': {'block_reason': 0, 'sa
```

---

4) `다양한 Key를 사용한 Runnable을 사용한 예시`

- ① `Messages 객체를 입력`, `dict 형태의 출력`
  - 메시지를 입력으로 받고 딕셔너리로 출력으로 반환하는 경우
  - *중요!:* `input_messages_key` = `"input"` *생략 → 입력으로* `Message` *객체를 넣도록 설정하게 됨*

```
from langchain_core.messages import HumanMessage
from langchain_core.runnables import RunnableParallel

# chain 생성하기
chain = RunnableParallel({"output_message": gemini_lc})
```

```
def get_session_history(session_id: str) -> BaseChatMessageHistory:
    # 세션 ID에 해당하는 대화 기록이 저장소에 없으면 새로운 ChatMessageHistory 생성하기
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]                    # 세션 ID에 해당하는 대화 기록 반환하기
```

```
# 체인에 대화 기록 기능을 추가한 RunnableWithMessageHistory 객체 생성하기

with_message_history = RunnableWithMessageHistory(
    chain,
    get_session_history,
    # 입력 메시지의 키를 "input"으로 설정(생략시 Message 객체로 입력)
    # input_messages_key="input",
    # 출력 메시지의 키를 "output_message"로 설정 (생략시 Message 객체로 출력)
    output_messages_key="output_message",
)
```

```
# query_5
# 주어진 메시지와 설정으로 체인 실행하기
with_message_history.invoke(
    # 혹은 "what is the definition of cosine?" 도 가능
    [HumanMessage(content="what is the definition of cosine?")],
    config={"configurable": {"session_id": "abc123"}},
)
```

- `query_5` - (`2.6s`)

```
{'output_message': AIMessage(content="The **cosine** of an angle in a right-angled triangle is defined as the rat
```

```
# query_6
with_message_history.invoke(
    # 이전의 답변에 대하여 한글로 답변을 재요청하기
    [HumanMessage(content="이전의 내용을 한글로 답변해 주세요!")],
    # 설정 옵션을 딕셔너리 형태로 전달하기
    config={"configurable": {"session_id": "abc123"}},
)
```

- `query_6` - (`2.7s`)

```
{'output_message': AIMessage(content='이전 내용을 한국어로 답변해 드리겠습니다.\n\n**코사인(cosine)의 정의**\n\n직각삼각형에서 코사
```

- ② `Messages 객체를 입력`, `Messages 객체를 출력`
  - *중요!:* `output_messages_key` = `output_message` *생략 → 출력으로* `Message` *객체를 반환함*

```
with_message_history = RunnableWithMessageHistory(
    gemini_lc,                                          # gemini_lc로 gemini 모델 사용하기
```

```
    get_session_history,                                # 대화 세션 기록을 가져오는 함수 지정하기
    # 입력 메시지의 키를 "input"으로 설정(생략시 Message 객체로 입력)
    # input_messages_key="input",
    # 출력 메시지의 키를 "output_message"로 설정 (생략시 Message 객체로 출력)
    # output_messages_key="output_message",
)
```

```
# query_7
with_message_history.invoke(
    [HumanMessage(content="코사인의 의미는 무엇인가요?")],
    config={"configurable": {"session_id": "def123"}},
)
```

- `query_7` - (`3.8s`)

```
AIMessage(content='코사인(cosine)은 삼각함수의 한 종류로, **직각삼각형에서 특정 각도에 대한 변의 길이 비율**을 나타냅니다. 좀 더 구체적으로
```

- ③ **모든 메시지 입력과 출력을 위한 단일 키를 가진 Dict**
  - 모든 입력 메시지와 출력 메시지에 대해 단일 키를 사용하는 방식
  - **`itemgetter("input_messages")`** → 입력 메시지 추출

```
from operator import itemgetter

with_message_history = RunnableWithMessageHistory(
    # "input_messages" 키 사용 → 입력 메시지 가져옴 → gemini 모델에 전달함
    itemgetter("input_messages") | gemini_lc,
    get_session_history,                                # 세션 기록을 가져오는 함수
    input_messages_key="input_messages",                # 입력 메시지의 키 지정하기
)
```

```
# query_8

with_message_history.invoke(
    {"input_messages": "코사인의 의미는 무엇인가요?"},
    config={"configurable": {"session_id": "xyz123"}},  # 설정 옵션 = 딕셔너리 형태
)
```

- `query_8` - (`4.2s`)

```
AIMessage(content='코사인(cosine)은 삼각함수의 한 종류로, **직각삼각형에서 특정 각도에 대한 변의 길이 비율**을 나타냅니다. 좀 더 구체적으로
```

---

6) 영구 저장소 (Persistent storage)

- **영구 저장소**
  - 개념: 프로그램 종료 or 시스템 재부팅 → 데이터 유지하는 저장 메커니즘
  - 데이터베이스, 파일 시스템, 기타 비휘발성 저장 장치 → 구현 가능
  - 필요성
    - 애플리케이션의 상태 저장, 사용자 설정 유지, **장기간 데이터 보존** 에 필수적
    - 이전 실행에서 중단된 지점부터 프로그램 다시 시작 가능 → **사용자는 데이터 손실 없이 계속 작업 가능**
  - 예제
    - **`RunnableWithMessageHistory`** = **`get_session_history`** 호출 가능 객체가 채팅 메시지 기록을 어떻게 검색하는지에 대해서 독립적
    - *로컬 파일 시스템 사용 예제*
    - *memory integrations*

- **① Redis 설치**
  - 먼저 `VS Code` 터미널에 설치할 것

```
pip install -qU redis
```

- **② Redis 서버 구동**
  - 기존에 연결할 `Redis` 배포가 없는 경우, 로컬 `Redis Stack` 서버 시작하기
  - *아래는 `Docker`로 `Redis` 서버 구동하는 명령어*

```
docker run -d -p 6379:6379 -p 8001:8001 redis/redis-stack:latest
```

- **`REDIS_URL`** 변수 = `Redis` 데이터베이스 연결 `URL` 할당하기

```
# Redis 서버의 URL은 아래와 같이 설정되어 있음
REDIS_URL = "redis://localhost:6379/0"
```

- **`LangSmith` 추적 설정** *optional*

```
from dotenv import load_dotenv
import os

load_dotenv()


os.environ["LANGCHAIN_TRACING_V2"] = "true"         # LANGCHAIN_TRACING_V2 환경 변수 = "true"로 설정
os.environ["LANGCHAIN_PROJECT"] = "RunnableWithMessageHistory"            # LANGCHAIN_PROJECT 설정
```

- 새로운 호출 가능한 객체 정의 = 메시지 기록 구현 업데이트 위함
- `RedisChatMessageHistory`의 인스턴스 반환

```
from langchain_community.chat_message_histories import RedisChatMessageHistory


def get_message_history(session_id: str) -> RedisChatMessageHistory:
    # 세션 ID를 기반으로 RedisChatMessageHistory 객체 반환하기
    return RedisChatMessageHistory(session_id, url=REDIS_URL)


with_message_history = RunnableWithMessageHistory(
    runnable,                                                    # 실행 가능한 객체
    get_message_history,                                         # 메시지 기록을 가져오는 함수
    input_messages_key="input",                                  # 입력 메시지의 키
    history_messages_key="history",                              # 기록 메시지의 키
)
```

- 이전과 동일한 방식으로 호출 가능

```
# query_9
with_message_history.invoke(
    {"ability": "math", "input": "What does cosine mean?"},
    config={"configurable": {"session_id": "redis123"}},
)
```

- **동일한 `session_id` ⟶ 두 번째 호출 수행**

```
# query_10
with_message_history.invoke(
    # 이전 답변에 대한 한글 번역 요청하기
```

```
    {"ability": "math", "input": "이전의 답변을 한글로 번역해 주세요."},
    # 설정 값으로 세션 ID를 "foobar"로 지정하기
    config={"configurable": {"session_id": "redis123"}},
)
```

- **다른 `session_id` → 질문하기**
  - 마지막 답변은 이전 대화 기록이 없음 → 제대로 된 답변을 받을 수 없음

```
# query_11
with_message_history.invoke(
    # 이전 답변에 대한 한글 번역 요청하기
    {"ability": "math", "input": "이전의 답변을 한글로 번역해 주세요."},
    # 설정 값으로 세션 ID를 "redis456"로 지정하기
    config={"configurable": {"session_id": "redis456"}},        # 다른 세션 id
)
```

- next: ***`09. 사용자 정의 제네레이터 (generator)`***
