

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

5. 대화내용을 기억하는 RAG 체인

1) 이전 대화를 기억하는 Chain 생성 방법

- 필요 사전 지식: [RunnableWithMessageHistory](#)

• 환경설정

```
# 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}')
```

• 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) 일반 Chain에 대화기록만 추가

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.chat_history import BaseChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.output_parsers import StrOutputParser
```

```

from dotenv import load_dotenv
import os

load_dotenv()

# 프롬프트 정의
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 Question-Answering 챗봇입니다. 주어진 질문에 대한 답변을 제공해주세요.",
        ),
        # 대화기록용 key 인 chat_history 는 가급적 변경 없이 사용하세요!
        MessagesPlaceholder(variable_name="chat_history"),
        ("human", "#Question:\n{question}"), # 사용자 입력을 변수로 사용
    ]
)

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

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0, # temperature = 0으로 설정
    max_output_tokens=4096,
)

# 일반 Chain 생성
chain = prompt | gemini_lc | StrOutputParser()

```

- 셀 출력

```
E0000 00:00:1759821431.490766 2567372 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted
```

- 대화를 기록하는 체인 생성 (**chain_with_history**)

```

# 세션 기록을 저장할 딕셔너리
store = {}

# 세션 ID를 기반으로 세션 기록을 가져오는 함수
def get_session_history(session_ids):
    print(f"[대화 세션ID]: {session_ids}")
    if session_ids not in store:
        # 새로운 ChatMessageHistory 객체를 생성하여 store에 저장
        store[session_ids] = ChatMessageHistory()
    return store[session_ids]

chain_with_history = RunnableWithMessageHistory(
    chain,
    get_session_history,
    input_messages_key="question",
    history_messages_key="chat_history",
)

```

세션 ID가 store에 없는 경우

해당 세션 ID에 대한 세션 기록 반환

세션 기록을 가져오는 함수

사용자의 질문이 템플릿 변수에 들어갈 key

기록 메시지의 키

- **첫 번째 질문 실행**

```

# query_1

chain_with_history.invoke(
    {"question": "나의 이름은 앨리스입니다."},
    config={"configurable": {"session_id": "abc123"}},
)

```

질문 입력

세션 ID 기준으로 대화 기록

- **query_1 (1.4s)**

```
[대화 세션ID]: abc123

'#Answer:\n안녕하세요, 앨리스님! 만나서 반갑습니다.'
```

- **이어서 질문 실행**

```
# query_2
```

```
chain_with_history.invoke(
    {"question": "내 이름이 뭐라고?"},
    config={"configurable": {"session_id": "abc123"}},
)
```

```
# 질문 입력
# 세션 ID 기준으로 대화 기록
```

- query_2 (0.8s)

```
[대화 세션ID]: abc123
```

```
'#Answer:\n앨리스님이라고 하셨습니다.'
```

2) RAG + RunnableWithMessageHistory

- 일반 RAG Chain 생성하기
 - 단, 6단계에 prompt 에 {chat_history} 를 꼭 추가

```
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import PDFPlumberLoader
from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import PromptTemplate
from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_core.output_parsers import StrOutputParser
from operator import itemgetter
from langchain_huggingface import HuggingFaceEmbeddings
import warnings
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv
import os
```

```
# 단계 1: 문서 로드(Load Documents)
loader = PDFPlumberLoader("../12_RAG/data/SPRI_AI_Brief_2023년12월호_F.pdf")
docs = loader.load()
```

```
# 단계 2: 문서 분할(Split Documents)
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=50)
split_documents = text_splitter.split_documents(docs)
```

```
# 단계 3: 임베딩(Embedding) 생성
```

```
from langchain_huggingface import HuggingFaceEmbeddings
import warnings
```

```
# 경고 무시
```

```
warnings.filterwarnings("ignore")
```

```
# HuggingFace Embeddings 사용
```

```
embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'}, \
    encode_kwargs={'normalize_embeddings': True})
```

```
print("✅ hugging-face 임베딩 모델 로딩 완료!")
```

- ✅ hugging-face 임베딩 모델 로딩 완료! (11.0s)

```
# 단계 4: DB 생성(Create DB) 및 저장
# 벡터스토어 생성하기
```

```
vectorstore = FAISS.from_documents(
    documents=split_documents,
    embedding=embeddings
)
```

```
print("✅ 벡터스토어 생성 완료!")
```

- ✅ 벡터스토어 생성 완료! (2.5s)

```
# 단계 5: 검색기(Retriever) 생성
# 문서에 포함되어 있는 정보를 검색하고 생성하기
retriever = vectorstore.as_retriever()
```

```
# 단계 6: 프롬프트 생성(Create Prompt)
# 프롬프트 생성하기

prompt = PromptTemplate.from_template(
    """You are an assistant for question-answering tasks.
    Use the following pieces of retrieved context to answer the question.
    If you don't know the answer, just say that you don't know.
    Answer in Korean.

    #Previous Chat History:
    {chat_history}

    #Question:
    {question}

    #Context:
    {context}

    #Answer:"""
)
```

```
print(type(prompt))
print(prompt.template)
```

- 셀 출력

```
<class 'langchain_core.prompts.prompt.PromptTemplate'>
```

```
You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question.
If you don't know the answer, just say that you don't know.
Answer in Korean.
```

```
#Previous Chat History:
{chat_history}
```

```
#Question:
{question}
```

```
#Context:
{context}
```

```
#Answer:
```

```
# 단계 7: 언어모델(LLM) 생성
# 모델(LLM) 생성하기

# LLM 초기화
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0,
    max_output_tokens=4096,
)
# temperature = 0으로 설정
```

- 셀 출력

```
E0000 00:00:1759822385.103469 2567372 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untrusted
```

```
# 단계 8: 체인(Chain) 생성
```

```
chain = (
    {
        "context": itemgetter("question") | retriever,
        "question": itemgetter("question"),
        "chat_history": itemgetter("chat_history"),
```

```

    }
    | prompt
    | gemini_lc
    | StrOutputParser()
)

```

- **대화를 저장할 수 있는 함수 정의하기**

```

# 세션 기록을 저장할 딕셔너리
store = {}

# 세션 ID를 기반으로 세션 기록을 가져오는 함수
def get_session_history(session_ids):
    print(f"[대화 세션ID]: {session_ids}")
    if session_ids not in store:
        store[session_ids] = ChatMessageHistory()
    return store[session_ids]

# 대화를 기록하는 RAG 체인 생성
rag_with_history = RunnableWithMessageHistory(
    chain,
    get_session_history,
    input_messages_key="question",
    history_messages_key="chat_history",
)

```

세션 ID가 store에 없는 경우
새로운 ChatMessageHistory 객체를 생성하여 store에 저장
해당 세션 ID에 대한 세션 기록 반환

세션 기록을 가져오는 함수
사용자의 질문이 템플릿 변수에 들어갈 key
기록 메시지의 키

- **첫 번째 질문 실행**

```

# query_1

rag_with_history.invoke(
    # 질문 입력
    {"question": "삼성전자가 만든 생성형 AI 이름은?"},
    # 세션 ID 기준으로 대화를 기록합니다.
    config={"configurable": {"session_id": "rag123"}},
)

```

- **query_1 (1.4s)**

```

[대화 세션ID]: rag123

"삼성전자가 만든 생성형 AI 이름은 '삼성 가우스'입니다."

```

- **이어진 질문 실행**

```

# query_2

rag_with_history.invoke(
    # 질문 입력
    {"question": "이전 답변을 영어로 번역해주세요."},
    # 세션 ID 기준으로 대화를 기록합니다.
    config={"configurable": {"session_id": "rag123"}},
)

```

- **query_2 (1.1s)**

```

[대화 세션ID]: rag123

"The generative AI created by Samsung Electronics is named 'Samsung Gauss'."

```

- next: **CH13 LangChain Expression Language (LCEL)**