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

---

5. `RunnableParallel`

1) `입력 및 출력 조작`

- 개념
  - `시퀀스` 내에서 하나의 `Runnable` 의 출력 → `Runnable` 의 **`입력 형식`에 `맞게` `조작`하는 데 유용하게 사용**
  - `prompt`에 대한 입력 = **`"context"`**, **`"question"`** `키`를 가진 `map` 형태로 예상
    - `사용자 입력` = 단순한 질문 내용 → **`retriever`** 사용 → 컨텍스트 가져오기
    - 사용자 입력 = **`"question"`** **`키 아래`**에 **`전달`해야 함**

- **`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}')")
    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.
```

```
from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
from langchain_huggingface import HuggingFaceEmbeddings
```

```
from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv
import os                                           # 3.4s
```

```
# 임베딩 형성

import warnings                                             # 경고 무시

warnings.filterwarnings("ignore")

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

embeddings = embeddings                                     # 12.7s
```

```
# 텍스트로부터 FAISS 벡터 저장소 생성하기
vectorstore = FAISS.from_texts(
    ["A is an AI engineer who loves programming!"],
    embedding=embeddings
)                                               # 0.8s
```

```
# 벡터 저장소를 검색기로 사용하기
retriever = vectorstore.as_retriever()
```

```
# 템플릿 정의하기

template = """Answer the question based only on the following context:
{context}

Question: {question}
"""
```

```
# 템플릿으로부터 채팅 프롬프트 생성하기

prompt = ChatPromptTemplate.from_template(template)
```

```
# 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",
)
```

- `gemini-2.5-flash-lite` 생성

```
E0000 00:00:1759979087.687909 1736881 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
# 검색 체인 구성하기
retrieval_chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | gemini_lc
    | StrOutputParser()
)
```

```
# 검색 체인 실행 → 질문에 대한 답변 얻기
import warnings                                             # 경고 무시
warnings.filterwarnings("ignore")

retrieval_chain.invoke("What is A's occupation?")
```

- 검색 체인 실행 (`0.9s`)

```
'A is an AI engineer.'
```

- **유의하기**
  - 다른 `Runnable`과 함께 `RunnableParell` 구성 시 **유형 변환이 자동 처리** → **`RunnableParallel`** 클래스에서 입력으로 주입되는 **`dict`** 입력을 **별도 래핑할 필요 없음**

- 아래 3가지 방식은 모두 동일하게 처리됨

```
# 자체 RunnableParallel 로 래핑됨
# 1번 방법
{"context": retriever, "question": RunnablePassthrough()}

# 2번 방법
RunnableParallel({"context": retriever, "question": RunnablePassthrough()})

# 3번 방법
RunnableParallel(context=retriever, question=RunnablePassthrough())
```

---

2) `itemgetter`를 단축어로 사용하기

- **`itemgetter` 단축어 사용** → `map`에서 데이터 추출 → `RunnableParellel`과 결합 가능
  - *참고: [Python Documentation – itemgetter]*

```
from operator import itemgetter

from langchain_community.vectorstores import FAISS
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_huggingface import HuggingFaceEmbeddings

from langchain_google_genai import ChatGoogleGenerativeAI
from dotenv import load_dotenv
import os
```

- `LLM` 초기화

```
# 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
model = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
)
```

- `gemini-2.5-flash-lite` → `model`이라는 이름으로 다시 초기화하기

```
E0000 00:00:1759979698.219470 1736881 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnableParallel
```

```
# 수도를 묻는 질문에 대한 체인 정의하기

capital_chain = (
    ChatPromptTemplate.from_template("{country} 의 수도는 어디입니까?")
    | model
```

```
    | StrOutputParser()
)
```

```
# 면적을 묻는 질문에 대한 체인 정의하기

area_chain = (
    ChatPromptTemplate.from_template("{country} 의 면적은 얼마입니까?")
    | model
    | StrOutputParser()
)
```

```
# capital_chain, area_chain 을 병렬로 실행할 수 있는 RunnableParallel 객체 생성하기

map_chain = RunnableParallel(capital=capital_chain, area=area_chain)
```

```
# map_chain 호출 → 대한민국의 수도와 면적 묻기

map_chain.invoke({"country": "대한민국"})
```

- `map_chain` (`"country": "대한민국"`) - (`1.4s`)

```
{'capital': '대한민국의 수도는 **서울**입니다.',
'area': '대한민국(남한)의 면적은 약 **100,403 제곱킬로미터(km²)**입니다.\n\n이는 세계 국가 면적 순위에서 약 107~109위 정도에 해당하며, 우
```

- `chain`별로 **`입력 템플릿의 변수가 달라도`** 상관없이 실행 가능

```
# 수도를 묻는 질문에 대한 체인 재정의

capital_chain2 = (
    ChatPromptTemplate.from_template("{country1} 의 수도는 어디입니까?")
    | model
    | StrOutputParser()
)
```

```
# 면적을 묻는 질문에 대한 체인 재정의

area_chain2 = (
    ChatPromptTemplate.from_template("{country2} 의 면적은 얼마입니까?")
    | model
    | StrOutputParser()
)
```

```
# capital_chain, area_chain을 병렬로 실행할 수 있는 RunnableParallel 객체 재생성

map_chain2 = RunnableParallel(capital=capital_chain2, area=area_chain2)
```

```
# map_chain 호출 → 이때 각각의 key에 대한 value를 전달함

map_chain2.invoke({"country1": "대한민국", "country2": "미국"})
```

- `map_chain2` (`"country1": "대한민국", "country2": "미국"`) - (`0.9s`)

```
{'capital': '대한민국의 수도는 **서울**입니다.',
'area': '미국의 총 면적은 약 **9,833,520 제곱 킬로미터 (km²)** 또는 **3,796,742 제곱 마일 (mi²)** 입니다.\n\n이 면적에는 육지와 내륙
```

---

3) `병렬 처리`

- **`map`** 에 있는 각 `Runnable`이 **`병렬`**로 **`실행`** → **`독립적인 프로세스`**를 **`병렬`**로 **`실행`**하는 데에도 **`유용`**
  - `area_chain`, `capital_chain`, `map_chain` = `map_chain`이 다른 두 체인을 모두 실행함에도 불구하고 **`거의 동일한 실행 시간`**을 가지는 것을 확인할 수 있음

```
%%timeit
```

```
# 면적을 묻는 체인 호출 → 실행 시간 측정하기
area_chain.invoke({"country": "대한민국"})
```

- 면적을 묻는 체인에 걸린 시간 (`6.9s`)

```
828 ms ± 244 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
# 수도를 묻는 체인 호출 → 실행 시간 측정하기
capital_chain.invoke({"country": "대한민국"})
```

- 수도를 묻는 체인에 걸린 시간 (`5.0`)

```
630 ms ± 100 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
# Parallel 구성 체인 호출 → 실행 시간 측정하기
map_chain.invoke({"country": "대한민국"})
```

- Parallel 구성 체인에 걸린 시간 (`7.0`)

```
844 ms ± 212 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- next: ***`06. 동적 속성 지정 (configurable_fields, configurable_alternatives)`***
