CH01 LangChain 시작하기

🦜🔗 랭체인 LangChain이란?

: 언어 모델을 활용해 다양한 애플리케이션을 개발할 수 있는 프레임워크

02. OpenAI API 키 발급 및 테스트

1. API 키 발급

1. https://platform.openai.com/account/api-keys 접속
2. **Create new secret key** 클릭
3. 생성된 키를 `.env` 파일에 저장

```
OPENAI_API_KEY=sk-xxxxxxxxxxxxxxxx
```

2. API 테스트 코드

```
from langchain_openai import ChatOpenAI

# 모델 초기화
llm = ChatOpenAI(
    temperature=0.1,  # 창의성 정도 조절
    model_name="gpt-4o-mini"  # 사용할 모델 지정
)
```

```
# 질문 입력
question = "대한민국의 수도는 어디인가요?"

# 모델 호출 및 출력
print(llm.invoke(question))
```

3. 실행 결과

```
AIMessage(content='대한민국의 수도는 서울입니다.')
```

LangChain이 OpenAI 모델과 연결되어 정상적으로 작동하는지를 확인하는 첫 단계

즉, **API 키가 올바르게 설정되었는지**, **LangChain이 모델과 대화 가능한지** 테스트하는 과정

03. LangSmith 추적 설정

> LLM 애플리케이션 개발, 모니터링 및 테스트를 위한 플랫폼

LangChain으로 실행한 모든 코드(Chain, LLM 호출 등)의

입력 → 처리 → 출력 흐름을 시각적으로 추적할 수 있음.

LangSmith가 하는 일

> LangChain을 통해 실행된 AI 코드의 “실행 로그(Log Dashboard)”를 자동으로 기록

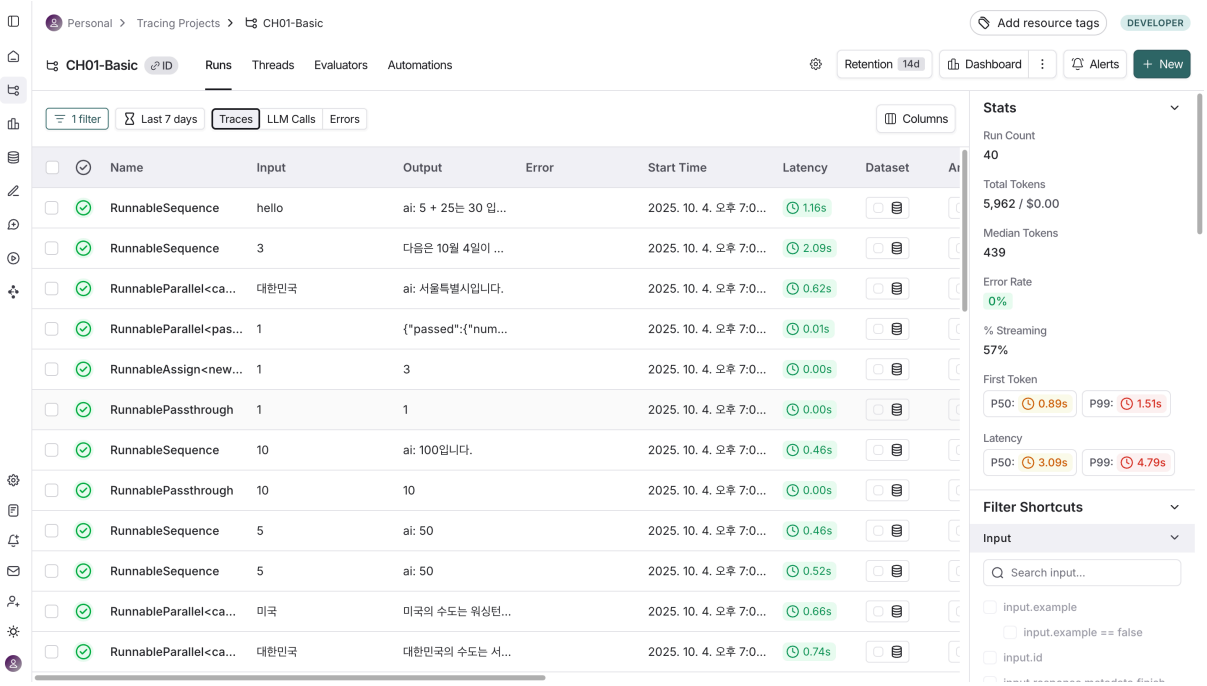
추적 기능

- 예상치 못한 최종 결과
- 에이전트가 루핑되는 이유
- 체인이 예상보다 느린 이유
- 에이전트가 각 단계에서 사용한 토큰 수

🔗 LangChain과 LangSmith 관계

역할	설명
LangChain	LLM 실행(프롬프트, 체인, 에이전트 등)
LangSmith	실행 내역 기록 및 분석 (로그 대시보드)

[실행 화면 해석 (Dashboard)]



→ 위 화면은 LangSmith가 자동으로 기록한 **LangChain 실행 히스토리의 대시보드**

즉,

LangSmith가 코드 실행 내역을 전부 **자동으로 수집**해서,

어떤 입력 → 어떤 출력 → 얼마나 걸렸는가를 한눈에 보여주는 요약 페이지임.

실행 화면 해석 (Dashboard)

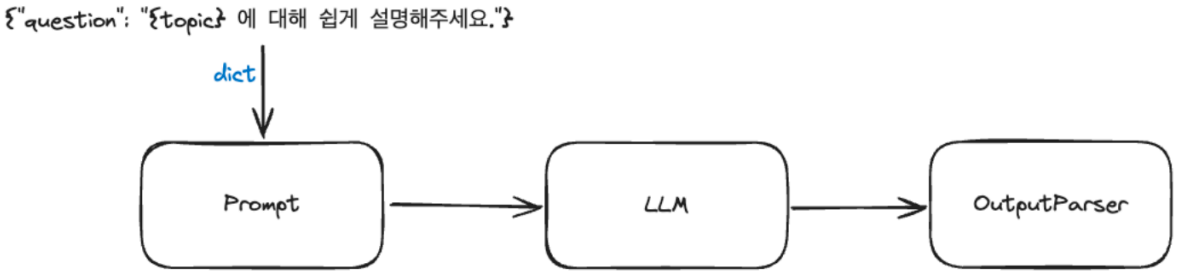
위치	내용
왼쪽 리스트	내가 돌린 실행 목록 (Chain, Runnable, LLM 등)
오른쪽 패널	실행 통계 (횟수, 속도, 토큰 수, 에러율 등)
중앙 영역	각 실행의 Input, Output, Latency, Model 정보

05. LangChain Expression Language(LCEL)

> AI의 "입력 → 처리 → 출력" 과정을 연결(Chain)하는 언어

> *사용자의 질문을 받음 → LLM이 답변을 생성 → 그 결과를 파싱(정리)* 하는 데이터 흐름을 한 줄 코드로 표현할 수 있게 해주는 언어!

아래 사진이 하나의 **Chain**이 됨!



예를 들어,

"인공지능의 원리를 쉽게 설명해줘!" 라고 입력하면:

1. **Prompt**

- "{topic}에 대해 쉽게 설명해 주세요." 라는 템플릿 존재
- 여기 `{topic}` 자리에 `"인공지능의 원리"` 가 들어감.

2. **LLM(모델)**

- OpenAI의 GPT 모델이 그 프롬프트를 받아서 실제 답변을 만듦.

3. **OutputParser**

- 생성된 텍스트를 정리하거나 필요한 형식(JSON 등)으로 바꿔서 최종 출력함.

06. LCEL 인터페이스 & Runnable

LCEL의 핵심 아이디어

> AI 구성 요소를 하나의 체인으로 연결해 재사용성과 가독성을 높이는 문법

실제 실행 과정 (invoke 호출)

1단계. 프롬프트와 모델 정의

```
from langchain.prompts import PromptTemplate
from langchain_openai import ChatOpenAI

prompt = PromptTemplate.from_template("{topic}에 대해 쉽게 설명해주세요.")
model = ChatOpenAI()
```

2단계. 체인 생성

```
chain = prompt | model
```

3단계. 입력값 전달 및 실행

```
input = {"topic": "인공지능의 학습 원리"}
response = chain.invoke(input)
print(response.content)
```

`invoke()` 는 LCEL에서 **"실행 버튼"** 같은 역할을 함.

딕셔너리 형태의 입력을 받아 체인을 순서대로 실행해 최종 결과를 반환함.

결과

```
"인공지능의 학습 원리는 데이터를 이용해 패턴을 학습하는 것입니다..."
```

**** 확장된 사용 예 - 체인 조합**

두 개의 프롬프트를 **병렬 실행**하거나, 출력값을 다시 다음 입력으로 넘길 수 있음.

```
from langchain.schema.runnable import RunnableParallel

chain = RunnableParallel({
"capital": prompt1 | model,
"population": prompt2 | model
})
```

→ 두 개의 질문을 동시에 GPT에 던지고 결과를 병렬로 받아오는 구조.

Runnable이란?

> LangChain의 모든 실행 가능한 객체의 공통 인터페이스

Runnable 종류 (Runnable의 '타입')

> Runnable 인터페이스를 **상속받은 실제 실행 객체들**이야. 데이터를 전달하거나 변형하거나 병렬 실행하는 역할을 함.

Runnable 종류	역할	예시
RunnablePassthrough	입력을 그대로 전달하거나 추가 키를 붙임	`assign()` 사용 가능
RunnableParallel	여러 Runnable을 동시에 실행	"수도" + "면적" 병렬
RunnableLambda	직접 만든 함수를 Runnable처럼 연결	`get_today()`, 계산 함수 등
itemgetter	딕셔너리에서 특정 키 추출	`"word1"`, `"word2"` 값만 꺼내기

LCEL에서 Prompt, Model, Chain 모두 **Runnable** 형태로 구현되어 있음.

즉, 공통적으로 다음 **네 가지 실행 메서드**를 쓸 수 있음.

메서드	설명	주요 상황
`.invoke(input)`	한 번 실행	일반적인 단일 입력 실행
`.batch(inputs)`	여러 입력 동시 실행	반복 작업 / 병렬 처리
`.stream(input)`	실시간 스트리밍 출력	ChatGPT 같은 답변 실시간 표시
`.ainvoke(input)`	비동기 실행	async 환경에서 (await 사용)

예시

```
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate

prompt = PromptTemplate.from_template("{topic}에 대해 한 문장으로 설명해 줘.")
model = ChatOpenAI()

chain = prompt | model
```

① invoke() – 단일 실행

```
response = chain.invoke({"topic": "인공지능"})
print(response.content)
```

→ 한 번 입력을 넣고 결과를 바로 받는 기본 실행 방식.

② batch() – 여러 입력 실행

```
inputs = [
{"topic": "인공지능"},
{"topic": "머신러닝"},
{"topic": "딥러닝"}
]
responses = chain.batch(inputs)
```

```
for res in responses:
print(res.content)
```

→ 여러 입력값을 한 번에 실행 (병렬 처리)

시간 효율을 높이는 방식으로, 대량 데이터 요약 등에 유용함.

③ stream() – 스트리밍 출력

```
for chunk in chain.stream({"topic": "인공지능"}):
print(chunk.content, end="")
```

→ ChatGPT처럼 **단어 단위로 출력** 되는 효과.

실시간 인터페이스 만들 때 사용됨.

④ ainvoke() – 비동기 실행

```
import asyncio

async def main():
response = await chain.ainvoke({"topic": "인공지능"})
print(response.content)

asyncio.run(main())
```

→ 여러 체인을 **동시에 비동기 실행** 가능.

웹서비스나 봇 같은 환경에서 자주 사용됨.

> **Runnable**은 **LCEL**에서 실행 가능한 모든 구성요소를 추상화한 인터페이스이며,
> **Passthrough·Parallel·Lambda** 등을 통해 데이터 흐름을 제어하거나
> 체인을 병렬·동적으로 확장할 수 있다.