

CHAPTER 04. LCEL 인터페이스

1. LangChain Expression Language(LCEL) 인터페이스

LCEL 문법은 완성된 체인을 만드는 것이라고 할 수 있습니다.

Langchain에서는 사용자 정의 체인을 가능한 한 쉽게 만들 수 있도록, "Runnable" 프로토콜을 구현했습니다.

chain을 구성할 때 llm, output parser를 묶어서 구성하는데 이런 llm이나 output parser가 Runnable입니다. 사용자 정의 체인은 이렇게 묶어서 만들어진 chain입니다.

chain을 묶을 때 구성되는 llm, output parser와 같은 각각의 모듈은 모두 Runnable 프로토콜을 가지고 있습니다. "실행 가능한 객체다."라고 해서 Runnable이라는 표현을 사용합니다.

Runnable은 prompt가 될 수도 있고, llm 객체가 될 수도 있고, output parser가 될 수도 있습니다.

이렇게 Runnable(llm, output parser.. 등)들을 묶어서 chain을 구성합니다.

묶어서 만들어진 chain도 Runnable(실행 가능한 객체)입니다.

invoke()를 호출할 수 있으면 모두 Runnable입니다.

예를 들어 묶어놓은 chain1, chain2이 있다면 chain1도 Runnable이고 chain2도 Runnable입니다.

두 개의 chain을 병렬적으로 실행 가능하게 만들어 주는 것이 RunnableParallel입니다.

Runnable 프로토콜은 대부분의 컴포넌트에 구현되어 있습니다.

이는 표준 인터페이스로, 사용자 정의 체인을 정의하고 표준 방식으로 호출하는 것을 쉽게 만듭니다. 표준 인터페이스에는 다음이 포함됩니다.

- **stream**: 응답의 청크를 스트리밍
- **invoke**: 입력에 대해 체인을 호출
- **batch**: 입력 목록에 대해 체인을 호출

- **astream**: 비동기적으로 응답의 청크를 스트리밍
- **ainvoke**: 비동기적으로 입력에 대해 체인을 호출
- **abatch**: 비동기적으로 입력 목록에 대해 체인을 호출

2. stream: 실시간 출력

예제 - stream

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

prompt = PromptTemplate.from_template("{product}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

answer = chain.stream({"product": "갤럭시폰"})

# chain.stream 메소드를 사용하여 '갤럭시폰' product에 대한 스트림을 생성하고 반복합니다.
for token in answer:
    # 스트림 에서 받은 데이터의 내용을 출력.
    # 줄 바꿈을 없애고 이어서 출력하면서, 버퍼를 즉시 비운다.
    print(token, end="", flush=True)
```

실행결과

갤럭시폰은 삼성전자가 제조한 스마트폰 시리즈로, 최신 기술과 혁신적인 디자인을 특징으로 합니다. 다양한 모델이 제공되어 사용자의 필요에 맞춘 선택이 가능하며, 고화질 카메라와 강력한 성능을 자랑합니다. 또한, 안드로이드 운영체제를 기반으로 하여 다양한 앱과 서비스를 지원합니다.

3. invoke: 호출

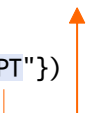
예제 - invoke

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)
prompt = PromptTemplate.from_template("{pruduct}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()
answer = chain.invoke({"pruduct": "ChatGPT"})
print(answer)
```



실행결과

'ChatGPT는 OpenAI에서 개발한 대화형 인공지능 모델로, 자연어 처리 기술을 기반으로 합니다. 사용자의 질문이나 요청에 대해 이해하고 적절한 답변을 생성하여 대화할 수 있는 능력을 가지고 있습니다. 다양한 주제에 대해 정보를 제공하고, 창의적인 글쓰기, 문제 해결 등 여러 용도로 활용될 수 있습니다.'

4. batch: 배치 (단위 실행)

함수 `chain.batch`는 여러 개의 딕셔너리를 포함하는 리스트를 인자로 받아, 각 딕셔너리에 있는 `product` 키의 값을 사용하여 일괄 처리를 수행합니다. 여러 개가 동시에 처리.

예제 - batch

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

prompt = PromptTemplate.from_template("{product}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

# product 리스트를 batch 처리하는 함수 호출. 묶어서 그룹 단위로 실행
answer = chain.batch([{"product": "ChatGPT"}, {"product": "Instagram"}])
print(answer)
print(answer[0])
```

실행결과

['ChatGPT는 OpenAI에서 개발한 인공지능 언어 모델로, 자연어 처리를 통해 사람과 대화할 수 있는 능력을 가지고 있습니다. 이 모델은 대량의 텍스트 데이터를 학습하여 다양한 주제에 대해 질문에 답하거나 정보를 제공할 수 있습니다. 사용자는 ChatGPT와의 상호작용을 통해 정보 검색, 창작, 문제 해결 등 다양한 작업을 수행할 수 있습니다.', '인스타그램은 사용자들이 사진과 동영상을 공유할 수 있는 소셜 미디어 플랫폼입니다. 다양한 필터와 편집 도구를 제공하여 사용자가 자신의 콘텐츠를 창의적으로 표현할 수 있도록 돕습니다. 또한, 친구와의 소통, 해시태그를 통한 관심사 탐색, 그리고 브랜드와의 연결을 통해 활발한 커뮤니티를 형성하고 있습니다.']

ChatGPT는 OpenAI에서 개발한 인공지능 언어 모델로, 자연어 처리를 통해 사람과 대화할 수 있는 능력을 가지고 있습니다. 이 모델은 대량의 텍스트 데이터를 학습하여 다양한 주제에 대해 질문에 답하거나 정보를 제공할 수 있습니다. 사용자는 ChatGPT와의 상호작용을 통해 정보 검색, 창작, 문제 해결 등 다양한 작업을 수행할 수 있습니다.

config 딕셔너리는 max_concurrency 키를 통해 동시에 처리할 수 있는 최대 작업 수를 설정합니다.
max_concurrency 매개변수를 사용하여 **동시 요청 수를 설정**할 수 있습니다

예제 - batch

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

prompt = PromptTemplate.from_template("{pruduct}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

answer = chain.batch(
    [
        {"pruduct": "ChatGPT"},
        {"pruduct": "Instagram"},
        {"pruduct": "멀티모달"},
        {"pruduct": "프로그래밍"},
        {"pruduct": "머신러닝"},
    ],
    config={"max_concurrency": 3},
)

print(answer)
```



실행결과

['ChatGPT는 OpenAI에서 개발한 대화형 인공지능 모델로, 자연어 처리 기술을 기반으로 합니다. 사용자의 질문이나 요청에 대해 자연스럽게 유창한 대화를 생성할 수 있으며, 다양한 주제에 대한 정보를 제공합니다. 이 모델은 학습된 데이터를 바탕으로 사용자와의 상호작용을 통해 지속적으로 개선됩니다.'], '인스타그램은 사용자가 사진과 동영상을 공유할 수 있는 소셜 미디어 플랫폼입니다. 다양한 필터와 편집 도구를 제공하여 사용자들이 창의적으로 콘텐츠를 꾸밀 수 있도록 돕습니다. 또한, 친구와의 소통, 유명인 및 브랜드와의 연결을 통해 글로벌 커뮤니티를 형성하는 데 기여합니다.'], '멀티모달은 다양한 형태의 데이터를 동시에 처리하고 분석하는 접근 방식을 의미합니다. 예를 들어, 텍스트, 이미지, 오디오 등 서로 다른 유형의 정보를 결합하여 더 풍부하고 정확한 인사이트를 도출할 수 있습니다. 이러한 기술은 인공지능, 머신러닝, 자연어 처리 등 여러 분야에서 활용되고 있습니다.'], '프로그래밍은 컴퓨터가 수행할 작업을 정의하는 과정으로, 특정 언어를 사용하여 코드로 작성됩니다. 이 과정에서는 문제를 해결하기 위한 알고리즘을 설계하고, 이를 구현하여 소프트웨어나 애플리케이션을 개발합니다. 프로그래밍은 창의성과 논리적 사고를 결합하여 다양한 분야에서 활용되는 중요한 기술입니다.'], '머신러닝은 컴퓨터가 데이터에서 패턴을 학습하고 예측을 수행할 수 있도록 하는 인공지능의 한 분야입니다. 이 과정은 주로 알고리즘을 사용하여 데이터를 분석하고, 이를 기반으로 모델을 생성하여 새로운 데이터에 대한 결정을 내리게 합니다. 머신러닝은 이미지 인식, 자연어 처리, 추천 시스템 등 다양한 응용 분야에서 활용되고 있습니다.']

가. 비동기 메소드

비동기 메소드는 프로그램이 특정 작업을 수행하고 있을 때, 그 작업이 완료될 때까지 프로그램 전체를 멈추지 않고 다른 작업을 계속 수행할 수 있게 해주는 방법입니다. 즉, **작업의 완료를 기다리는 동안 다른 코드의 실행이 가능**합니다.

나. 일상의 예: 커피숍

비동기 메소드를 커피숍에서의 상황으로 비유해보겠습니다. 카페에 가서 커피를 주문했다고 상상해 봅시다.

1) 동기 방식(Synchronous)

커피를 주문한 후, 커피가 나올 때까지 그 자리에서 기다립니다.

기다리는 동안 다른 어떠한 일을 하지 못하고 오로지 커피만을 기다리는 상태입니다.

2) 비동기 방식(Asynchronous)

커피를 주문한 후, 기다리는 동안에 다른 일을 하는 거예요. (예: 책 읽기, 친구와 대화하기)을 합니다.

커피가 준비되면 번호가 호출되고, 이때 커피를 받으러 갑니다.

다. 프로그래밍에서의 예

웹 서버를 개발할 때 비동기 메소드는 매우 유용하게 사용됩니다.

예를 들어, 사용자가 데이터베이스에서 정보를 요청하는 상황을 생각해 볼 수 있습니다.

1) 동기 방식(Synchronous)

서버는 데이터베이스의 응답을 기다리면서 다른 요청을 처리하지 못하고 멈춰 있습니다.

이는 효율성이 떨어질 수 있습니다.

2) 비동기 방식(Asynchronous)

서버는 데이터를 요청하고, 그 응답을 기다리는 동안 다른 사용자의 요청을 계속 처리할 수 있습니다.

데이터베이스에서 응답이 오면, 그때 해당 작업을 완료합니다.

6. async stream: 비동기 스트림

chain.astream() 함수는 비동기 스트림을 생성하며, 주어진 product에 대한 메시지를 비동기적으로 처리합니다.

비동기 for 루프(async for)를 사용하여 **스트림에서 메시지를 순차 적으로 받아오고**, **print()** 함수를 통해 **메시지의 내용(s.content)**을 즉시 출력합니다.

end=""는 출력 후 줄 바꿈을 하지 않도록 설정하며, **flush=True**는 출력 버퍼를 강제로 비워 바로 출력합니다.

예제 - 비동기 스트림

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)
prompt = PromptTemplate.from_template("{product}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

# 비동기 스트림을 사용하여 '삼성 갤럭시 폰' product의 메시지를 처리합니다.
async for token in chain.astream({"product": "삼성 갤럭시폰"}):
    print(token, end="", flush=True)
```

실행결과

삼성 갤럭시 폰은 삼성전자가 제조한 스마트폰 시리즈로, 다양한 모델과 가격대가 있어 사용자 선택의 폭이 넓습니다. 이 시리즈는 고급스러운 디자인과 뛰어난 카메라 성능, 그리고 최신 기술이 적용된 디스플레이로 유명합니다. 또한, 안드로이드 운영체제를 기반으로 하여 다양한 앱과 기능을 지원하며, 사용자 맞춤형 경험을 제공합니다.

7. async invoke: 비동기 호출

chain 객체의 ainvoke 메소드는 비동기적으로 주어진 인자를 사용하여 작업을 수행합니다. 여기서는 **product**라는 키와 **삼성 갤럭시 폰**이라는 값을 가진 딕셔너리를 인자로 전달하고 있습니다. 이 메소드는 특정 product에 대한 처리를 비동기적으로 요청하는 데 사용될 수 있습니다.

예제 - 비동기 호출

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

prompt = PromptTemplate.from_template("{product}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

# 비동기 체인 객체의 'ainvoke' 메서드를 호출하여 '삼성 갤럭시 폰' product를 처리합니다.
my_process = chain.ainvoke({"product": "삼성 갤럭시 폰"})

# 비동기로 처리되는 프로세스가 완료될 때까지 기다립니다.
# my_process를 가지고 있다가 await라는 키워드를 붙여서 호출하게 되면 그때 답변을 받는다.
print(await my_process)
```

실행결과

삼성 갤럭시 폰은 삼성전자가 제조한 스마트폰 시리즈로, 다양한 모델과 가격대가 있어 소비자 선택의 폭이 넓습니다. 최신 기술이 적용된 고성능 카메라, AMOLED 디스플레이, 그리고 강력한 배터리 성능을 자랑하며, 사용자 경험을 최적화하기 위한 다양한 소프트웨어 기능도 포함되어 있습니다. 또한, 갤럭시 생태계와의 호환성 덕분에 스마트워치, 태블릿 등 다른 삼성 기기와의 연동이 용이합니다.

8. async batch: 비동기 배치

함수 abatch는 비동기적으로 일련의 작업을 일괄처리합니다.

다음 예제는 chain 객체의 abatch 메소드를 사용하여 product에 대한 작업을 비동기적으로 처리하고 있습니다. **await** 키워드는 **해당 비동기 작업이 완료될 때까지 기다리는 데** 사용됩니다.

예제 - 비동기 배치

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

prompt = PromptTemplate.from_template("{product}에 대해서 3문장으로 설명해줘.")
chain = prompt | llm | StrOutputParser()

# 주어진 토픽에 대해 비동기적으로 일괄 처리를 수행합니다.
my_abatch_process = chain.abatch(
    [{"product": "삼성 갤럭시 폰"}, {"product": "삼성 노트북"}, {"product": "인스타그램"}]
)

# 비동기로 처리되는 일괄 처리 프로세스가 완료될 때까지 기다립니다.
# my_abatch_process를 가지고 있다가 await라는 키워드를 붙여서 호출하게 되면 그때 답변을 받는다.
print(await my_abatch_process)
```

실행결과

['삼성 갤럭시 폰은 삼성전자가 제조한 스마트폰 시리즈로, 다양한 모델과 가격대에서 제공됩니다. 이 시리즈는 고급스러운 디자인과 뛰어난 카메라 성능, 그리고 사용자 친화적인 인터페이스인 One UI로 유명합니다. 또한, 최신 기술을 적용하여 5G 지원, 고해상도 디스플레이, 강력한 배터리 수명 등을 특징으로 합니다.'],

'삼성 노트북은 뛰어난 성능과 세련된 디자인으로 유명하며, 다양한 모델이 있어 사용자 요구에 맞춰 선택할 수 있습니다. 특히, AMOLED 디스플레이와 긴 배터리 수명으로 고화질 영상 감상과 이동 중 작업에 적합합니다. 또한, 삼성의 소프트웨어와 하드웨어 통합으로 원활한 사용자 경험을 제공합니다.'],

'인스타그램은 사용자들이 사진과 동영상을 공유하고 소통할 수 있는 소셜 미디어 플랫폼입니다. 다양한 필터와 편집 도구를 제공하여 사용자가 자신의 콘텐츠를 창의적으로 표현할 수 있도록 돕습니다. 또한, 해시태그와 스토리 기능을 통해 관심 있는 주제나 사람들과 쉽게 연결될 수 있습니다.']

9. Parallel: 병렬성 - 병렬체인 구성(RunnableParallel)

RunnableParallel을 사용할 때(자주 디렉터리 형태로 작성됨), 각 요소를 병렬로 실행합니다.

langchain_core.runnables 모듈의 RunnableParallel 클래스를 사용하여 두 가지 작업을 병렬로 실행하는 예시입니다.

- (1) ChatPromptTemplate.**from_template()** 메소드를 사용하여 주어진 country에 대한 수도 와 면적을 구하는 **두 개의 체인(chain1, chain2)**을 만듭니다.
- (2) 이 체인들은 각각 model과 파이프(|) 연산자를 통해 연결됩니다.
- (3) RunnableParallel 클래스를 사용하여 이 두 개의 체인을 **capital과 area라는 키로 결합하여 동시에 실행할 수 있는 combined 객체**를 생성합니다.

예제 - 병렬체인

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnableParallel

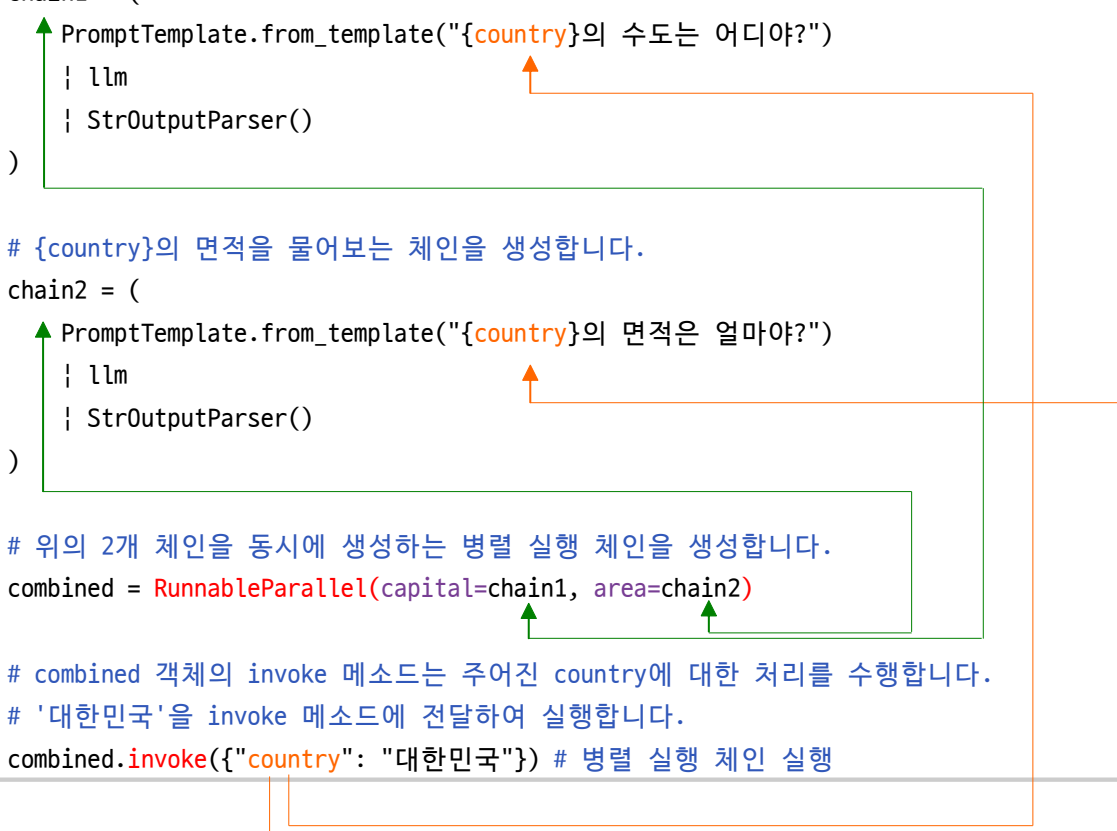
llm = ChatOpenAI(
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

# {country}의 수도를 물어보는 체인을 생성합니다.
chain1 = (
    PromptTemplate.from_template("{country}의 수도는 어디야?")
    | llm
    | StrOutputParser()
)

# {country}의 면적을 물어보는 체인을 생성합니다.
chain2 = (
    PromptTemplate.from_template("{country}의 면적은 얼마야?")
    | llm
    | StrOutputParser()
)

# 위의 2개 체인을 동시에 생성하는 병렬 실행 체인을 생성합니다.
combined = RunnableParallel(capital=chain1, area=chain2)

# combined 객체의 invoke 메소드는 주어진 country에 대한 처리를 수행합니다.
# '대한민국'을 invoke 메소드에 전달하여 실행합니다.
combined.invoke({"country": "대한민국"}) # 병렬 실행 체인 실행
```



실행결과

```
{'capital': '대한민국의 수도는 서울입니다.',
 'area': '대한민국의 면적은 약 100,210 평방킬로미터(km²)입니다. 이는 한반도의 남쪽 부분에 해당하며,
 다양한 지형과 기후를 가지고 있습니다.'}
```

예제 - 체인을 따로 실행시켰을 때

```
...
# chain1.invoke() 함수는 chain1 객체의 invoke 메서드를 호출합니다.
# 이때, country이라는 키에 대한민국라는 값을 가진 딕셔너리를 인자로 전달합니다.
chain1.invoke({"country": "대한민국"})    # chain1 실행
...
```

실행 결과

'대한민국의 수도는 서울입니다.'

10. 배치에서 병렬 처리

병렬 처리는 다른 실행 가능한 코드와 결합 될 수 있습니다.

chain.batch() :

여러 개의 딕셔너리를 포함하는 리스트를 인자로 받아, 각 딕셔너리에 있는 "topic" 키에 해당하는 값을 처리합니다. 이 예시에서는 "대한민국"과 "미국"이라는 두 개의 토픽을 배치 처리하고 있습니다.

예제 - 배치에서 병렬처리

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnableParallel

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

# {country} 의 수도를 물어보는 체인을 생성합니다.
chain1 = (
    PromptTemplate.from_template("{country}의 수도는 어디야?")
    | llm
    | StrOutputParser()
)
```

```

# {country} 의 면적을 물어보는 체인을 생성합니다.
chain2 = (
    PromptTemplate.from_template("{country}의 면적은 얼마야?")
    | llm
    | StrOutputParser()
)

# 배치 처리를 수행합니다.
# chain1.batch([{"country": "대한민국"}, {"country": "미국"}])

# 배치 처리를 수행합니다.
# chain2.batch([{"country": "대한민국"}, {"country": "미국"}])

# 위의 2개 체인을 동시에 생성하는 병렬 실행 체인을 생성합니다.
combined = RunnableParallel(capital=chain1, area=chain2)

# combined.batch 함수는 주어진 데이터를 배치로 처리하는 데 사용됩니다.
# 이 예시에서는 두 개의 딕셔너리 객체를 포함하는 리스트를 인자로 받아 각각 대한민국과 미국
# 두 나라에 대한 데이터를 배치 처리합니다.

# 주어진 데이터를 배치로 처리합니다.
combined.batch([{"country": "대한민국"}, {"country": "미국"}])

```

실행결과

```

[{'capital': '대한민국의 수도는 서울입니다.',
  'area': '대한민국의 면적은 약 100,210 평방킬로미터(km²)입니다. 이는 한반도의 남쪽 부분에 해당하
며, 북한과 함께 한반도를 구성하고 있습니다.'},
 {'capital': '미국의 수도는 워싱턴 D.C.입니다.',
  'area': '미국의 면적은 약 9,830,000 평방킬로미터(약 3,796,000 평방마일)입니다. 이는 세계에서 세
번째로 큰 나라로, 러시아와 캐나다에 이어 가장 큰 면적을 가지고 있습니다.'}]

```
