

chain

2023년 10월 30일 월요일 오전 7:18

[How to | □□ Langchain](#)
[Interface | □□ Langchain](#)

document에는 chain사용법 나오지만 동시에 LCEL을 권장하니까 여기서는 LCEL사용법 정리 하겠다
chain은 LCEL로 표현할수있다 chain object를 tool로 통합 할수도 있다 -> agent에서 사용한다

define

Runnable : A unit of work that can be invoked ([langchain.schema.runnable.base Runnable — □□ LangChain 0.0.325](#))

stream : stream back chunks of the response

invoke : call the chain -> custom chain에서 __call__부분
=> chain.invoke(dicttype)

batch : call the chain on a list of inputs
=> chain.batch([])

-> 위에꺼 Asyn method방식도 있다

input type

prompt : dict

LLM,Retriever : single str

chatLLM : list of chat message or a PromptValue

tool : single str, or dict or depending tool

outputparser :

schema

확인자 and debugging 할때 좋음

input_schema : input pydantic model -> runnable구조에서 자동 생성됨.

- chain.input_schema.schema()
- prompt.input_schema.schema()
- model.input_schema.schema()

output_schema : output pydantic model -> runnable구조에서 자동 생성됨

- chain.output_schema.schema()

- Async method & Parallelism아래에서 확인

[Interface | □□ Langchain](#)

- bind를 통해 출력으로 나온것 중에서 중간에 나온 과정을 생략하거나 조작을 할수있다 다른 모델에서 작동안되는것도 있다고 한다(bind를 통해 prompt에서 input variable로 있는 arg를 조작할수있다
예) stop] ([Bind runtime args | □□ Langchain](#))

- Configuration -> runnable object의 argument를 조정하는 것 크게 두가지 방식있다 필요할때 찾아보면 될것 같다([Configuration | □□ Langchain](#))

how to configure – alternatives

```
llm = ChatGooglePalm(temperature=0).configurable_alternatives(  
    # This gives this field an id  
    # When configuring the end runnable, we can then use this id to configure this field  
    ConfigurableField(id="llm"),  
    # This sets a default_key.  
    # If we specify this key, the default LLM (ChatAnthropic initialized above) will be used  
    default_key="google-palm",  
    # This adds a new option, with name 'openai' that is equal to 'ChatOpenAI()'`  
    openai=ChatOpenAI(),  
    # This adds a new option, with name 'gpt4' that is equal to 'ChatOpenAI(model="gpt-4")`  
    gpt4=ChatOpenAI(model="gpt-4"),  
    # You can add more configuration options here  
)
```

how to configure - alternatives

```
prompt = PromptTemplate.from_template("Tell me a joke about {topic}").configurable_alternatives(
    # This gives this field an id
    # When configuring the end runnable, we can then use this id to configure this field
    ConfigurableField(id="prompt"),
    # This sets a default_key.
    # If we specify this key, the default LLM (ChatAnthropic initialized above) will be used
    default_key="joke",
    # This adds a new option, with name `poem`
    poem=PromptTemplate.from_template("Write a short poem about {topic}"),
    # You can add more configuration options here
)
chain = {'topic': RunnablePassthrough()} | prompt | llm

• chain.invoke('bears')
• chain.with_config(configurable={'llm':'openai', 'prompt':'poem'}).invoke('bears')
```

다음과 같이 ConfigurableField(id)를 통해 invoke할때 .with_config(configurable=dic)으로 해주면 프로그램 디버깅할때도 편하고 조금더 간결하게 짤수있다
이건 설정을 하면 좋을것 같다 다른 prompt template또는 다른 모델도 적용할수있다 - 간결한 코드를 위해 configuration은 매번하는게 좋을것 같다

- debug시 유용할것 같다 다른 llm모델 사용했을때 지원하는지는 모르겠다 ([Add fallbacks](#) | [Langchain](#))

```
from unittest.mock import patch
from openai.error import RateLimitError

# Note that we set max_retries = 0 to avoid retrying on
openai_llm = ChatOpenAI(max_retries=0)
google_llm = ChatGooglePalm()
llm = openai_llm.with_fallbacks([google_llm])
```

다른 llm을 적용하기 위해서 openai llm선언하고 with_fallbacks인자로 다른 llm모델 넘겼다

```
# Now let's try with fallbacks to GooglePalm
with patch('openai.ChatCompletion.create', side_effect=RateLimitError()):
    try:
        print(llm.invoke("Why did the the chicken cross the road?"))
    except:
        print("Hit error")
```

- Runnable은 동적으로 invoke로 작동되면 나온 결과물을 동적으로 실행하기 위해 사용하는 개념 같다[**Not implemented for all component**]. 예를 들어 RunnableLambda는 invoke dic에서 넘어온 인자를 내가 만든 함수에 적용할수있게 한다 ([Run arbitrary functions](#) | [Langchain](#))

```

from langchain.schema.runnable import RunnableLambda
def length_function(text):
    return len(text)

def _multiple_length_function(text1, text2):
    return len(text1) * len(text2)


def multiple_length_function(_dict):
    return _multiple_length_function(_dict["text1"], _dict["text2"])

prompt = ChatPromptTemplate.from_template("what is {a} + {b}")
model = ChatOpenAI()

chain = {
    "a": itemgetter('foo') | RunnableLambda(length_function),
    "b": {"text1": itemgetter("foo"), "text2": itemgetter("bar")} | RunnableLambda(multiple_length_function)
} | prompt | model

> chain.invoke({'foo': 'hello world!', 'bar': 'what is bar?'})

```



화면 캡처: 2023-12-15 오전 9:07

- custom outputparser가 필요할때 또는 streaming capabilities는 유지하면서 이전 결과를 바꾸고 싶을때 사용 ([Custom generator functions](#) | [Langchain](#))

- yield, generator
 - generator : iterator를 생성해주는 함수이다
 - 모든 값을 메모리에 담고 있지 않다 필요할때마다 생성해서 반환한다
 - 호출할때마다 한 개의 값을 리턴한다

```

def generator():
    yield "a"
    print("aa")
    yield "b"
    print("bb")
    yield "c"

gen = generator()
gen
# <generator object generator at 0x7f5c74734350>
next(gen) # a
next(gen) # aa , b
next(gen) # bb , c

```

화면 캡처: 2023-11-02 오전 11:48

- To handle multiple Runables in parallel. prompt가 다르게 적용된 chain에 한번에 입력 또는 같은 주제에 대해서 한번에 입력 가능하다 ([Use RunnableParallel/RunnableMap](#) | [Langchain](#))
 - prompt안에서도 input variable이 여러 개 있을수있는데 이것도 invoke할때 dict형태로 넣으면 가능하다

- ([Route between multiple Runables](#) | [Langchain](#))

- Routing : 이전 결과물로 인해 다음이 결정되는 chain을 만드는것이다
 - interactions with LLMs
 - two ways to perform routing
 1. RunnableBranch
 2. writing custom factory function : 반드시 return runnable이어야 한다
- RunnableBranch
 - 초기화방법 initialized -> list (condition,runnable) pair , default runnable
 - input의 condition을 넘긴다
 - (T/F, 실행함수)
 - 만약 true이면 실행함수 실행(chain등등 포함이다)
 - 약간 if else문을 langchain에 적용한 썸띵스 느낌

지금까지 정리

출력 나온거 중에 prompt input variable을 출력중에 생략하거등 조작 가능 -bind
 pytorch argument설정하듯이 langchain이도 arg설정 가능 - 필요할때 찾아보면 될것 같다
 debug시 유용할것 같은 fallback 확인 -> other llms model가능?
 chain invoke시 나온 결과를 동적으로 조작 할수있게 하는 runnable
 parlle에서 여러 prompt input variable dic으로 넣을수있다
 routing을 이용해서 if else같이 조건에 맞는 chain등등을 적용할수있는것 같다

LCEL cookbook ([Cookbook](#) | [Langchain](#))

[LangChain cookbook](#) | [Langchain](#)

prompt + LLM +outputparser

prompt 만들때 도움 되는 사이트 : [AI Prompt Generator](#) | [Taskade](#)
<https://www.feedough.com/ai-prompt-generator/>

StrOutParser(),JsonOutputFunctionsParser, JsonKeyOutputFunctionsParser(key_name) : json파일에 key name 설정하면 특정 key내용만 가져올수있다

```
from langchain.output_parsers.openai_functions import JsonKeyOutputFunctionsParser

chain = (
    prompt
    | model.bind(function_call={"name": "joke"}, functions=functions)
    | JsonKeyOutputFunctionsParser(key_name="setup")
)
```

화면 캡처: 2023-11-02 오후 2:28

```
# simplifying input - 3
prompt = ChatPromptTemplate.from_template('Tell me a joke about {topic} and {topic-2}')

simple_input_chain = (
    {'topic': RunnablePassthrough(), 'topic-2':RunnablePassthrough()}
    | prompt
    | model.with_config({'llm':'palm'})
    | StrOutputParser())

print(simple_input_chain.invoke(['dog', 'boots']))
```

화면 캡처: 2023-11-02 오후 2:41

RAG

FAISS나 chrom db를 활용해서 retrival을 만들었다면 다음과 같이 LCEL적용할수있다

```
chain = (
    {"context": retriever, "question": RunnablePassthrough()}
    | prompt
    | model
    | StrOutputParser()
)
```

화면 캡처: 2023-11-02 오후 2:43

[미완]

multiple chains

[Multiple chains](#) | [Langchain](#)

chain1의 item_variable을 chain2에서 | 로 묶었을때 넘길수있다

서로 다른 prompt라도 chain1에서나온결과를 chain2에 chain2에 나온결과를 chain3의 prompt에 넘길수있다

Branching and Merging

여러 개의 Branch를 combine할수있다

위에서 언급한것같이 branch는 | 로 여러 개 묶인거다

이런 branch들을 combine할수있는 예시를 제공하고 있다 -> 어떤 주제에 대한 장점 프롬프트적용한 branch + 단점프롬프트 적용한 brach 를 combine해서 하나의 결과로 출력한다

Querying a SQL DB

[Querying a SQL DB | □□ Langchain](#)

[미완] 라마인덱스와 같은 선상에서 작성하면 좋을것 같다

Agent

[Agents | □□ Langchain](#)

agent파트에는 개념과 예제 중심으로 설명하고 여기서는 LCEL에서 agent가 어떻게 적용되는지 알아본다 [Runnable into an agent]

embedding을 사용해서 질문과 유사도 높은 프롬프트 적용할수도 있다 - Routing by semantic similarity

Evaluation

[Evaluation | □□ Langchain](#)

[Evaluating Large Language Models: Methods, Best Practices & Tools | Lakera – Protecting AI teams that disrupt the world.](#)

신뢰 할 수 있는 결과물을 만들기 위해 고려 해야하는 부분이다

string evaluators

[String Evaluators | □□ Langchain](#)

- criteria evaluation (표준평가) :
- embedding distance : semantic similarity를 측정하기 위해 vector distance사용 : score낮을수록 더 유사하다

trajectory evaluators

comparison evaluators

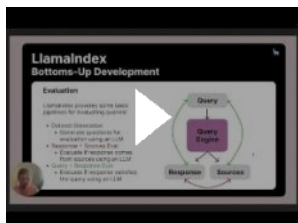
[Evaluation | □□ Langchain](#)

[Evaluating RAG pipelines with Ragas + LangSmith \(langchain.dev\)](#)

Benchmark langchain : [Evaluate LLMs and RAG a practical example using Langchain and Hugging Face \(philschmid.de\)](#)

: 사람이 직접 판단하거나 llm을 사용해서 판단 하는 방법이 있는데 llm으로 평가 하는 방식 참고를 위에 사이트에서 하면 될것 같아요

[Discover LlamaIndex: Bottoms-Up Development with LLMs \(Part 3, Evaluation\)](#)



[langchain/docs/docs/langsmith/walkthrough.ipynb at master · langchain-ai/langchain \(github.com\)](#)