CH03 출력 파서(Output Parsers)

출력파서: LLM이 낸 텍스트 답변을 **JSON**이나 **파이썬 객체** 같은 **정해진 형식**으로 변환해주는 역할. LLM 기반 애플리케이션 개발에 있어 중요한 도구.

파서의 **주요 특징**

1. 많은 종류의 출력 파서
2. 대부분 **스트리밍 지원**
3. 확장성이 좋다= **쉽게 추가/교체할 수 있다**

<총 정리?>

prompt template으로 **prompt 생성** → prompt.partial()으로 **프롬프트 내부 채우기**

→ **LLM** 객체 생성 → **Parse**r (결과물을 내가 원하는 형태로 가공해줌) 생성

→ prompt = prompt.partial(format_instructions=parser.get_format_instructions())
{**답변 형태(instruction)을 parser에서 쪽 빼내서 프롬프트에 주입**}

→ chain = prompt | model | parser **체인** 만들기

→ chain.invoke({"question": question})
프롬프트 '질문'란에 질문 넣고 chain 쭉 돌아 **결과물 받기**

1. Pydantic OuputParser

언어 모델의 출력을 **특정 데이터 모델에 맞게 변환**

`PydanticOutputParser` (이는 대부분의 OutputParser 에 해당되기도 합니다) 에는 주로 **두 가지 핵심 메서드가 구현** 되어야 합니다.

- `get_format_instructions()` : 언어 모델이 출력해야 할 정보의 형식을 정의
- `parse()` : 언어 모델의 출력(문자열로 가정)을 받아들여 이를 특정 구조로 분석하고 변환합니다. Pydantic와 같은 도구를 사용하여, 입력된 문자열을 사전 정의된 스키마에 따라 검증하고, 해당 스키마를 따르는 데이터 구조로 변환합니다.

2. CommaSeparatedList OutputParser

3. StructuredOutputParser

LLM에 대한 답변을 dict 형식으로 구성하고 key/value 쌍으로 갖는 여러 필드를 반환

Pydantic/JSON 파서가 더 강력하지만, 이는 덜 강력한 모델(parameter 수가 낮은 모델)에 유

- **ResponseSchema**: **어떤 출력**이 필요한지 **규칙**을 정의한다.
  - **쉽게 말해:** "답변은 `answer` 칸에, 참고 자료는 `source` 칸에 넣어줘" 하고 칸을 만들어 주는 것과 같아요.
  - **핵심 역할:** 사용자의 질문에 대한 `답변` 필드와 사용된 **`참고 자료(웹사이트 등)`*에 대한 필드를 **필수적으로 포함**하도록 **출력 형식(Schema)**을 정의합니다.
- **StructuredOutputParser**: 그 **규칙에 맞춰** LLM의 출력을 **정리**한다.

4. JSON 출력 파서(Json OutputParser)

LLM이 원하는복잡한 형식(JSON)에 맞춰 출력하려면, 그 모델의 지능(용량)이 충분히 커야 함.

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import JsonOutputParser
from pydantic import BaseModel, Field
# from langchain_openai import ChatOpenAI

# OpenAI 객체를 생성합니다.
model = ChatOpenAI(temperature=0, model_name="gpt-4o")

# 원하는 데이터 구조를 정의합니다.
class Topic(BaseModel):
    description: str = Field(description="주제에 대한 간결한 설명")
    hashtags: str = Field(description="해시태그 형식의 키워드(2개 이상)")
```

```
# 질의 작성
question = "지구 온난화의 심각성 대해 알려주세요."

# 파서를 설정하고 프롬프트 템플릿에 지시사항을 주입합니다.
```

```
parser = JsonOutputParser(pydantic_object=Topic)

prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "당신은 친절한 AI 어시스턴트 입니다. 질문에 간결하게 답변하세
요."),
        ("user", "#Format: {format_instructions}\n\n#Question: {question}"),
    ]
)

prompt = prompt.partial(format_instructions=parser.get_format_instruction
s())

chain = prompt | model | parser  # 체인을 구성합니다.

chain.invoke({"question": question})  # 체인을 호출하여 쿼리 실행
```

Pydantic 사용 안하는 버전 존재
이 경우 JSON을 반환하도록 요청하지만, 스키마가 어떻게 되어야 하는지에 대한 구체적인 정보는 제공하지 않습니다.

```
# 질의 작성
question = "지구 온난화에 대해 알려주세요.
온난화에 대한 설명은 `description`에, 관련 키워드는 `hashtags`에 담아주세요."

# JSON 출력 파서 초기화
parser = JsonOutputParser()

# 프롬프트 템플릿을 설정합니다.
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "당신은 친절한 AI 어시스턴트 입니다. 질문에 간결하게 답변하세
요."),
        ("user", "#Format: {format_instructions}\n\n#Question: {question}"),
    ]
)

# 지시사항을 프롬프트에 주입합니다.
```

```
prompt = prompt.partial(format_instructions=parser.get_format_instruction
s())

# 프롬프트, 모델, 파서를 연결하는 체인 생성
chain = prompt | model | parser

# 체인을 호출하여 쿼리 실행
response = chain.invoke({"question": question})

# 출력을 확인합니다.
print(response)
```

5. 데이터프레임 출력 파서 PandasDataFrame OutputParser)

Pandas DataFrame: Python 프로그래밍 언어에서 널리 사용되는 데이터 구조로, 데이터 조작 및 분석을 위해 흔히 사용됩니다.

이 기능은 LLM에게 **데이터 분석가** 역할을 맡기는 것과 같아요.

1. **DataFrame 지정**: 사용자가 분석하고 싶은 **판다스 데이터(표 형태의 데이터)**를 LLM에게 **제공**합니다.
2. **데이터 추출 요청**: LLM에게 "이 데이터에서 특정 정보를 **찾아줘**"라고 **요청**합니다.
3. **사전(Dictionary) 형태로 응답**: LLM은 찾은 데이터를 **깔끔하고 사용하기 편한 형식(Dictionary)**으로 정리해서 **돌려줍니다**.

- **쉽게 말해:** 이 **출력 파서**를 이용하면, LLM에게 **"이 데이터프레임에서 필요한 정보만 찾아서, 프로그래밍에 쓰기 좋은 딕셔너리 형태로 딱 정리해 줘"**라고 시킬 수 있습니다.

```
# 출력 목적으로만 사용됩니다.
def format_parser_output(parser_output: Dict[str, Any]) -> None:
    # 파서 출력의 키들을 순회합니다.
    for key in parser_output.keys():
        # 각 키의 값을 딕셔너리로 변환합니다.
        parser_output[key] = parser_output[key].to_dict()
    # 예쁘게 출력합니다.
    return pprint.PrettyPrinter(width=4, compact=True).pprint(parser_output)
```

`format_parser_output` 함수는 파서 출력을 사전 형식(key, value)으로 변환하고 출력 형식을 지정하는 데 사용됩니다.

```
import pprint
from typing import Any, Dict

import pandas as pd
from langchain.output_parsers import PandasDataFrameOutputParser
# from langchain_core.prompts import PromptTemplate
# from langchain_openai import ChatOpenAI

# ChatOpenAI 모델 초기화 (gpt-3.5-turbo 모델 사용을 권장합니다)
model = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo")

# 파서를 설정하고 프롬프트 템플릿에 지시사항을 주입합니다.
parser = PandasDataFrameOutputParser(dataframe=df)

# 열 작업 예시입니다.
df_query = "Age column 을 조회해 주세요."
# 쿼리를 엉망으로 적어도 LLM이 잘알아듣고 정확한 파이썬 코드를 생성하여 실행!!

# 프롬프트 템플릿을 설정합니다.
prompt = PromptTemplate(
    template="Answer the user query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],  # 입력 변수 설정
    partial_variables={
        "format_instructions": parser.get_format_instructions() #아까 df 구조에
서 추출한 지시
    },  # 부분 변수 설정
)

# 체인 생성
chain = prompt | model | parser

# 체인 실행
parser_output = chain.invoke({"query": df_query})
```

```
# 출력: 위에서 만든 예쁜 출력 def 사용
format_parser_output(parser_output)
```

※ `parser = PandasDataFrameOutputParser( dataframe=df )` 방식으로 DataFrame 객체를 전달해서 초기화

초기화 시 제공된 DataFrame의 구조(컬럼 이름 등)를 기반으로 언어 모델에게 어떤 컬럼을 사용하고 어떤 연산을 수행할 수 있는지 알려주는 instructions 생성

6. 날짜 형식 출력 파서(Datetime OutputParser)

▼ 날짜 **형식 코드**

형식 코드	설명	예시
%Y	4자리 연도	2024
%y	2자리 연도	24
%m	2자리 월	07
%d	2자리 일	04
%H	24시간제 시간	14
%I	12시간제 시간	02
%p	AM 또는 PM	PM
%M	2자리 분	45
%S	2자리 초	08
%f	마이크로초 (6자리)	000123
%z	UTC 오프셋	+0900
%Z	시간대 이름	KST
%a	요일 약어	Thu
%A	요일 전체	Thursday
%b	월 약어	Jul
%B	월 전체	July
%c	전체 날짜와 시간	Thu Jul 4 14:45:08 2024
%x	전체 날짜	07/04/24
%X	전체 시간	14:45:08

```
from langchain.output_parsers import DatetimeOutputParser
# from langchain.prompts import PromptTemplate
```

```
# from langchain_openai import ChatOpenAI

# 날짜 및 시간 출력 파서
output_parser = DatetimeOutputParser()
output_parser.format = "%Y-%m-%d"

# 사용자 질문에 대한 답변 템플릿
template = """Answer the users question:\n\n#Format Instructions: \n{form
at_instructions}\n\n#Question: \n{question}\n\n#Answer:"""

prompt = PromptTemplate.from_template(
    template,
    partial_variables={
        "format_instructions": output_parser.get_format_instructions()
    },  # 지침을 템플릿에 적용
)
```

프롬프트 내용을 출력

`PromptTemplate(input_variables=['question'], input_types={}, partial_variables={' format_instructions ': " Write a datetime string that matches the following pattern: '%Y-%m-%d' .\n\nExamples: 2025-11-08, 2024-11-08, 2025-11-07\n\nReturn ONLY this string, no other words!"}, template='Answer the users question:\n\n#Format Instructions: \n{format_instructions}\n\n#Question: \n{question}\n\n#Answer:')`

```
# Chain 을 생성합니다.
chain = prompt | ChatOpenAI() | output_parser
# 체인을 호출하여 질문에 대한 답변을 받습니다.
output = chain.invoke({"question": "Google 이 창업한 연도는?"})
# 결과를 문자열로 변환
output.strftime("%Y-%m-%d")
```

output → `datetime.datetime(1998, 9, 4, 0, 0)`

7. 열거형 출력 파서(Enum OutputParser)

```
from langchain.output_parsers.enum import EnumOutputParser

from enum import Enum
# Enum을 상속받은 Colors 클래스
class Colors(Enum):
```

```
    RED = "빨간색"
    GREEN = "초록색"
    BLUE = "파란색"

# EnumOutputParser 인스턴스 생성
parser = EnumOutputParser(enum=Colors)

# from langchain_core.prompts import PromptTemplate
# from langchain_openai import ChatOpenAI

# 프롬프트 템플릿을 생성합니다.
prompt = PromptTemplate.from_template(
    """다음의 물체는 어떤 색깔인가요?
Object: {object}
Instructions: {instructions}
""" # 파서에서 지시사항 형식을 가져와 부분적으로 적용합니다.
).partial(instructions=parser.get_format_instructions())
# 프롬프트와 ChatOpenAI, 파서를 연결합니다.
chain = prompt | ChatOpenAI() | parser

response = chain.invoke({"object": "하늘"})  # "하늘" 에 대한 체인 호출 실행
print(response)
```

📖 prompt 생성하면서 바로 `.partial` 가능한 거랑 chain 내부에 바로 `ChatOpenAI()` 넣는 거

8. 출력 수정 파서(Output Fixing Parser)

출력 파싱 과정에서 발생할 수 있는 **오류를 자동으로 수정**하는 기능

스키마를 준수하지 않는 결과가 나올 경우, `OutputFixingParser` 가 자동으로 형식이 잘못된 출력을 인식하고, 이를 수정하기 위한 새로운 명령어와 함께 모델에 다시 제출한다는 것입니다. 이 과정에서, 수정을 위한 명령어는 오류를 정확히 지적하고, 올바른 형식으로 데이터를 재구성할 수 있도록 구체적인 지시를 포함해야 합니다.