CHAPTER 5. 출력 파서(Output Parser)

출력 파서(Output parser) 는 언어 모델(LLM)의 출력을 더 유용하고 구조화된 형태로 변환하는 컴포넌트입니다.
파서는 해석하다 라는 의미를 가지고 있는데, LLM이 준 답변을 구조화된 형태로 변환해 주는 것이 파서의 역할입니다.

답변을 받을 스키마를 파서에 정의하고 LLM에 붙여서 사용합니다.
LLM의 출력을 받아서 구조화된 데이터 형식으로 변환 해주는 역할을 합니다.

LangChain의 출력 파서는 LLM 기반 애플리케이션 개발에 있어 중요한 도구입니다.
이를 통해 개발자는 LLM의 출력을 보다 효과적으로 활용하고, 구조화된 데이터로 쉽게 변환할 수 있습니다

1. PydanticOutputParser

PydanticOutputParser 는 언어 모델의 출력을 더 구조화된 정보로 변환하는 데 도움이 되는 클래스입니다. 단순 텍스트 형태의 응답 대신, 사용자가 필요로 하는 정보를 명확하고 체계적인 형태로 제공할 수 있습니다.

PydanticOutputParser 에는 주로 두 가지 핵심 메소드가 구현되어야 합니다.

. **get_format_instructions()**:

언어 모델이 출력해야 할 정보의 형식을 정의하는 지침(instruction)을 제공합니다.
예를 들어, 언어 모델이 출력해야 할 데이터의 필드와 그 형태를 설명하는 지침을 문자열로 반환할 수 있습니다.
이때 설정하는 지침(instruction)의 역할이 매우 중요합니다.
이 지침에 따라 언어 모델은 출력을 구조화하고, 이를 특정 데이터 모델에 맞게 변환할 수 있습니다.

. **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. 구조화된 객체로 변환해 줍니다.

가. 출력 파서를 사용하지 않는 경우

예제 - 파서를 사용하지 않은 경우

```
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
```

```
email_conversation = """From: 김철수 (chulsoo.kim@bikecorporation.me)
To: 이은채 (eunchae@teddyinternational.me)
Subject: "ZENESIS" 자전거 유통 협력 및 미팅 일정 제안

안녕하세요, 이은채 대리님,

저는 바이크코퍼레이션의 김철수 상무입니다. 최근 보도자료를 통해 귀사의 신규 자전거 "ZENESIS"에 대해 알게 되었습니다. 바이크코퍼레이션은 자전거 제조 및 유통 분야에서 혁신과 품질을 선도하는 기업으로, 이 분야에서의 장기적인 경험과 전문성을 가지고 있습니다.

ZENESIS 모델에 대한 상세한 브로슈어를 요청드립니다. 특히 기술 사양, 배터리 성능, 그리고 디자인 측면에 대한 정보가 필요합니다. 이를 통해 저희가 제안할 유통 전략과 마케팅 계획을 보다 구체화할 수 있을 것입니다.

또한, 협력 가능성을 더 깊이 논의하기 위해 다음 주 화요일(1월 15일) 오전 10시에 미팅을 제안합니다. 귀사 사무실에서 만나 이야기를 나눌 수 있을까요?

감사합니다.

김철수
상무이사
바이크코퍼레이션
"""

prompt = PromptTemplate.from_template(
    "다음의 이메일 내용중 중요한 내용을 추출해 주세요.\n\n{email_conversation}"
)

llm = ChatOpenAI(api_key=key, model='gpt-4o-mini')

chain = prompt | llm

answer = chain.invoke({"email_conversation": email_conversation})
print(answer.content)
```

실행결과

```
중요 내용 추출:

- 발신자: 김철수 (바이크코퍼레이션 상무)
- 수신자: 이은채 (Teddy International)
- 주제: "ZENESIS" 자전거 유통 협력 및 미팅 일정 제안
- 요청사항: ZENESIS 모델에 대한 상세 브로슈어 (기술 사양, 배터리 성능, 디자인 정보)
- 미팅 제안: 1월 15일 (다음 주 화요일) 오전 10시, 귀사 사무실에서 만나기 희망
- 목적: 협력 가능성 논의 및 유통 전략과 마케팅 계획 구체화
```

나. 출력 파서를 사용한 경우

예제 – PydanticOutputParser

```
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.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
from langchain_core.prompts import PromptTemplate

email_conversation = """From: 김철수 (chulsoo.kim@bikecorporation.me)
To: 이은채 (eunchae@teddyinternational.me)
Subject: "ZENESIS" 자전거 유통 협력 및 미팅 일정 제안

안녕하세요, 이은채 대리님,

저는 바이크코퍼레이션의 김철수 상무입니다. 최근 보도자료를 통해 귀사의 신규 자전거 "ZENESIS"에 대해 알게 되었습니다. 바이크코퍼레이션은 자전거 제조 및 유통 분야에서 혁신과 품질을 선도하는 기업으로, 이 분야에서의 장기적인 경험과 전문성을 가지고 있습니다.

ZENESIS 모델에 대한 상세한 브로슈어를 요청드립니다. 특히 기술 사양, 배터리 성능, 그리고 디자인 측면에 대한 정보가 필요합니다. 이를 통해 저희가 제안할 유통 전략과 마케팅 계획을 보다 구체화할 수 있을 것입니다.

또한, 협력 가능성을 더 깊이 논의하기 위해 다음 주 화요일(1월 15일) 오전 10시에 미팅을 제안합니다. 귀사 사무실에서 만나 이야기를 나눌 수 있을까요?

감사합니다.

김철수
상무이사
바이크코퍼레이션
"""

class EmailSummary(BaseModel):
    person: str = Field(description="메일을 보낸 사람")
    email: str = Field(description="메일을 보낸 사람의 이메일 주소")
    subject: str = Field(description="메일 제목")
    summary: str = Field(description="메일 본문을 요약한 텍스트")
    date: str = Field(description="메일 본문에 언급된 미팅 날짜와 시간")


# PydanticOutputParser 생성
parser = PydanticOutputParser(pydantic_object=EmailSummary)
print(parser.get_format_instructions())
```

```
'''
프롬프트를 정의합니다.
 1.question: 유저의 질문을 받습니다.
 2.email_conversation: 이메일 본문의 내용을 입력합니다.
 3.format: 형식을 지정합니다.
'''

prompt = PromptTemplate.from_template(
    """
You are a helpful assistant. Please answer the following questions in KOREAN.

QUESTION:
{question}

EMAIL CONVERSATION:
{email_conversation}

FORMAT:
{format}
"""
)

# format 에 PydanticOutputParser의 부분 포맷팅(partial) 추가
prompt = prompt.partial(format=parser.get_format_instructions())

llm = ChatOpenAI(api_key=key, model='gpt-4o-mini')
chain = prompt | llm

# chain 을 실행하고 결과를 출력합니다.
response = chain.invoke(
    {
        "email_conversation": email_conversation,
        "question": "이메일 내용중 주요 내용을 추출해 주세요.",
    }
)

print(response.content)    # str이다

structured_output = parser.parse(response.content)  # str을 객체로 만들기
print(structured_output)
```

실행결과

```json
{  "person": "김철수",
  "email": "chulsoo.kim@bikecorporation.me",
  "subject": "\"ZENESIS\" 자전거 유통 협력 및 미팅 일정 제안",
  "summary": "바이크코퍼레이션의 김철수가 이은채 대리에게 ZENESIS 자전거의 브로슈어 요청 및 협력 논의를 위한 미팅을 제안함.",
  "date": "2024-01-15"
}```

person='김철수' email='chulsoo.kim@bikecorporation.me' subject='"ZENESIS" 자전거 유통 협력 및 미팅 일정 제안' summary='바이크코퍼레이션의 김철수가 이은채 대리에게 ZENESIS 자전거의 브로슈어 요청 및 협력 논의를 위한 미팅을 제안함.' date='2024-01-15'
```

다. 출력 파서 사용한 경우

예제 – PydanticOutputParser

```
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.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
from langchain_core.prompts import PromptTemplate

email_conversation = """From: 김철수 (chulsoo.kim@bikecorporation.me)
To: 이은채 (eunchae@teddyinternational.me)
Subject: "ZENESIS" 자전거 유통 협력 및 미팅 일정 제안

안녕하세요, 이은채 대리님,

저는 바이크코퍼레이션의 김철수 상무입니다. 최근 보도자료를 통해 귀사의 신규 자전거 "ZENESIS"에 대해 알게 되었습니다. 바이크코퍼레이션은 자전거 제조 및 유통 분야에서 혁신과 품질을 선도하는 기업으로, 이 분야에서의 장기적인 경험과 전문성을 가지고 있습니다.

ZENESIS 모델에 대한 상세한 브로슈어를 요청드립니다. 특히 기술 사양, 배터리 성능, 그리고 디자인 측면에 대한 정보가 필요합니다. 이를 통해 저희가 제안할 유통 전략과 마케팅 계획을 보다 구체화할 수 있을 것입니다.

또한, 협력 가능성을 더 깊이 논의하기 위해 다음 주 화요일(1월 15일) 오전 10시에 미팅을 제안합니다. 귀사 사무실에서 만나 이야기를 나눌 수 있을까요?

감사합니다.

김철수
상무이사
바이크코퍼레이션
"""

class EmailSummary(BaseModel):
    person: str = Field(description="메일을 보낸 사람")
    email: str = Field(description="메일을 보낸 사람의 이메일 주소")
    subject: str = Field(description="메일 제목")
    summary: str = Field(description="메일 본문을 요약한 텍스트")
    date: str = Field(description="메일 본문에 언급된 미팅 날짜와 시간")


# PydanticOutputParser 생성
parser = PydanticOutputParser(pydantic_object=EmailSummary)
print(parser.get_format_instructions())
```

```
prompt = PromptTemplate.from_template(
    """
You are a helpful assistant. Please answer the following questions in KOREAN.

QUESTION:
{question}

EMAIL CONVERSATION:
{email_conversation}

FORMAT:
{format}
"""
)

# format 에 PydanticOutputParser의 부분 포맷팅(partial) 추가
prompt = prompt.partial(format=parser.get_format_instructions())

llm = ChatOpenAI(api_key=key, model='gpt-4o-mini')

chain = prompt | llm | parser

# chain 을 실행하고 결과를 출력합니다.
response = chain.invoke(
    {
        "email_conversation": email_conversation,
        "question": "이메일 내용중 주요 내용을 추출해 주세요.",
    }
)

print(response)
```

실행결과

```
person='김철수' email='chulsoo.kim@bikecorporation.me' subject='"ZENESIS" 자전거 유통 협력 및 미팅 일정 제안' summary="김철수 상무가 이은채 대리님에게 'ZENESIS' 자전거의 브로슈어를 요청하고, 협력 가능성을 논의하기 위한 미팅을 제안함." date='2024-01-15T10:00:00'
```

라. 체인에 구조화된 출력 추가

예제 - with_structured_output()

```
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.output_parsers import PydanticOutputParser
from pydantic import BaseModel, Field
from langchain_core.prompts import PromptTemplate

class EmailSummary(BaseModel):
    person: str = Field(description="메일을 보낸 사람")
    email: str = Field(description="메일을 보낸 사람의 이메일 주소")
    subject: str = Field(description="메일 제목")
    summary: str = Field(description="메일 본문을 요약한 텍스트")
    date: str = Field(description="메일 본문에 언급된 미팅 날짜와 시간")

llm_with_structered = ChatOpenAI(
    temperature=0, model_name="gpt-4o-mini"
).with_structured_output(EmailSummary)

email_conversation = """From: 김철수 (chulsoo.kim@bikecorporation.me)
To: 이은채 (eunchae@teddyinternational.me)
Subject: "ZENESIS" 자전거 유통 협력 및 미팅 일정 제안

안녕하세요, 이은채 대리님,

저는 바이크코퍼레이션의 김철수 상무입니다. 최근 보도자료를 통해 귀사의 신규 자전거 "ZENESIS"에 대해 알게 되었습니다. 바이크코퍼레이션은 자전거 제조 및 유통 분야에서 혁신과 품질을 선도하는 기업으로, 이 분야에서의 장기적인 경험과 전문성을 가지고 있습니다.

ZENESIS 모델에 대한 상세한 브로슈어를 요청드립니다. 특히 기술 사양, 배터리 성능, 그리고 디자인 측면에 대한 정보가 필요합니다. 이를 통해 저희가 제안할 유통 전략과 마케팅 계획을 보다 구체화할 수 있을 것입니다.
또한, 협력 가능성을 더 깊이 논의하기 위해 다음 주 화요일(1월 15일) 오전 10시에 미팅을 제안합니다. 귀사 사무실에서 만나 이야기를 나눌 수 있을까요?

감사합니다.

김철수
상무이사
바이크코퍼레이션
"""
answer = llm_with_structered.invoke(email_conversation)
print(answer)
```

실행결과

```
person='김철수' email='chulsoo.kim@bikecorporation.me' subject='ZENESIS 자전거 유통 협력 및 미팅 일정 제안' summary="김철수 상무가 이은채 대리님에게 바이크코퍼레이션의 자전거 유통 협력 제안을 하며, 'ZENESIS' 모델에 대한 브로슈어 요청과 미팅 일정을 제안하는 내용입니다." date='2024-01-15T10:00:00'
```

2. CommaSeparatedListOutputParser

CommaSeparatedListOutputParser 는 쉼표로 구분된 항목 목록을 반환할 필요가 있을 때 유용합니다.

이 출력 파서를 사용하면, 사용자가 입력한 데이터나 요청한 정보를 쉼표로 구분하여 명확하고 간결한 목록 형태로 제공받을 수 있습니다. 예를 들어, 여러 개의 데이터 포인트, 이름, 항목 또는 다른 종류의 값들을 나열할 때 이를 통해 효과적으로 정보를 정리하고 사용자에게 전달할 수 있습니다.

이 방법은 정보를 구조화하고, 가독성을 높이며, 특히 데이터를 다루거나 리스트 형태의 결과를 요구하는 경우에 매우 유용합니다.

예제 - CommaSeparatedListOutputParser

```
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.output_parsers import CommaSeparatedListOutputParser
from langchain_core.prompts import PromptTemplate

# 콤마로 구분된 리스트 출력 파서 초기화
output_parser = CommaSeparatedListOutputParser()

# 출력 형식 지침 가져오기
format_instructions = output_parser.get_format_instructions()

# 프롬프트 템플릿 설정
prompt = PromptTemplate(
    # 주제에 대한 다섯 가지를 나열하라는 템플릿
    template="List five {subject}.\n{format_instructions}",
    input_variables=["subject"],  # 입력 변수로 'subject' 사용
    # 부분 변수로 형식 지침 사용
    partial_variables={"format_instructions": format_instructions},
)

llm = ChatOpenAI(api_key=key, model='gpt-4o-mini')

chain = prompt | llm | output_parser

answer = chain.invoke({"subject": "대한민국 관광명소"})
print(answer)
```

실행결과

```
['경복궁', 'N서울타워', '제주도', '부산 해운대', '경주 불국사']
```

3. JSON 출력 파서(JsonOutputParser)

이 출력 파서는 사용자가 원하는 JSON 스키마를 지정할 수 있게 해주며, 그 스키마에 맞게 LLM에서 데이터를 조회하여 결과를 도출해줍니다.

JSON (JavaScript Object Notation) 은 데이터를 저장하고 구조적으로 전달하기 위해 사용되는 경량의 데이터 교환 포맷입니다. 웹 개발에서 매우 중요한 역할을 하며, 서버와 클라이언트 간의 통신을 위해 널리 사용됩니다. JSON은 읽기 쉽고, 기계가 파싱하고 생성하기 쉬운 텍스트를 기반으로 합니다.

JSON의 기본 구조 JSON 데이터는 이름(키)과 값의 쌍으로 이루어져 있습니다. 여기서 "이름"은 문자열이고, "값"은 다양한 데이터 유형일 수 있습니다. JSON은 두 가지 기본 구조를 가집니다:

- 객체: 중괄호 {}로 둘러싸인 키-값 쌍의 집합입니다. 각 키는 콜론 :을 사용하여 해당하는 값과 연결되며, 여러 키-값 쌍은 쉼표 ,로 구분됩니다.

- 배열: 대괄호 []로 둘러싸인 값의 순서 있는 목록입니다. 배열 내의 값은 쉼표 , 로 구분됩니다.

```
{
  "name": "John Doe",
  "age": 30,
  "is_student": false,
  "skills": ["Java", "Python", "JavaScript"],
  "address": {
      "street": "123 Main St",
      "city": "Anytown"
  }
}
```

예제 – jsonOutputParser

```
from dotenv import load_dotenv
import os

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

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

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

```
# 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)
print(parser.get_format_instructions())

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

prompt = prompt.partial(format_instructions=parser.get_format_instructions())

chain = prompt | llm | parser

answer = chain.invoke({"question": question})

print(answer)
print(answer['description'])
```

실행결과

```
{'description': '지구 온난화는 기후 변화의 주요 원인으로, 지구의 평균 기온이 상승하여 극단적인
                 기상 현상, 해수면 상승, 생태계 파괴 등을 초래하고 있습니다.',
 'hashtags': '#지구온난화 #기후변화'}
```

Pydantic 없이도 이 기능을 사용할 수 있습니다. 이 경우 JSON을 반환하도록 요청하지만, 스키마가 어떻게 되어야 하는지에 대한 구체적인 정보는 제공하지 않습니다.

예제 – jsonOutputParser (Pydantic 없이 사용)

```
from dotenv import load_dotenv
import os

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

from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import JsonOutputParser
from langchain_openai import ChatOpenAI

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

# 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 답변하세요."),
        ("user", "#Format: {format_instructions}\n\n#Question: {question}"),
    ]
)

# 지시사항을 프롬프트에 주입합니다.
prompt = prompt.partial(format_instructions=parser.get_format_instructions())

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

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

# 출력을 확인합니다.
print(response)
```

실행결과

```
{'description': '지구 온난화는 지구의 평균 기온이 상승하는 현상으로, 주로 온실가스의 증가로 인해 발생합니다. 이는 기후 변화, 해수면 상승, 극단적인 기상 현상 등을 초래할 수 있습니다.', 'hashtags': ['#지구온난화', '#기후변화', '#온실가스', '#환경문제', '#지속가능성']}
```

- 출처 : 랭체인LangChain 노트 (https://wikidocs.net/233343)