LangChain 3주차 정리

category	GDGoC
sub category	LangChain Study
상태	분류대기
완료	☐

LangChain 출력 파서(Output Parser) 시스템의 구조적 이해와 활용

1. 개요

LangChain의 Output Parser는 LLM이 생성한 자연어 텍스트를 **정형화된 데이터 구조로 변환하는 후처리 계층(Post-processing layer)**이다.

LLM은 본질적으로 비정형 텍스트를 출력하지만, 실용적인 AI 응용에서는 **데이터 분석, API 응답, 인터페이스 통신** 등 구조화된 형태가 필요하다.

따라서 Output Parser는 단순히 텍스트를 파싱하는 역할을 넘어, LLM 기반 시스템의 **데이터 일관성·신뢰성·자동화 수준**을 결정하는 핵심 모듈이다.

LangChain은 이 계층을 체계적으로 모듈화하여 다양한 형태의 파서를 제공한다.

각 파서는 데이터 형식(JSON, 리스트, 날짜, Enum 등)에 따라 다른 전략을 취하지만, **공통적으로 PromptTemplate → LLM → OutputParser**라는 흐름을 따른다.

2. Output Parser의 아키텍처와 동작 원리

2.1 핵심 개념

LangChain의 Chain은 보통 다음 세 단계로 구성된다.

1. **PromptTemplate**: LLM에게 필요한 형식을 안내 (Input formatting)
2. **LLM 호출**: 언어적 출력을 생성
3. **Output Parser**: 출력된 문자열을 구조화된 객체로 변환

Output Parser는 이 세 번째 단계에 위치하며, LLM의 응답을 *데이터 모델 수준에서 검증 및 변환*한다.

즉, 텍스트 → 데이터 객체(예: Pydantic 모델, 리스트, dict, datetime 등)의 변환기를 자동화한다.

2.2 내부 구조

모든 Output Parser는 공통적으로 두 가지 핵심 메서드를 구현한다.

- `get_format_instructions()`

  모델이 어떤 형식으로 출력을 내야 하는지를 설명하는 텍스트 지침을 반환한다.

  이 문자열은 PromptTemplate 내부에 `{format_instructions}` 변수로 주입되어 LLM의 출력 형식을 통제한다.

- `parse(text: str)`

  LLM이 생성한 문자열을 받아서 구조화된 Python 객체로 변환한다.

  예외 발생 시 유효성 검사를 수행하거나, OutputFixingParser가 후속 보정 단계로 개입한다.

2.3 Chain에서의 연결 방식

Output Parser는 LangChain의 `|` 연산자로 LLM 뒤에 연결된다.

```
chain = prompt | llm | parser
```

이때 각 컴포넌트는 "데이터 흐름"으로 연결되며,

입력 → 프롬프트 포맷 → 모델 출력 → 파서 변환 → 최종 결과의 일관성을 보장한다.

3. PydanticOutputParser — 스키마 기반의 강력한 구조화

3.1 개요

`PydanticOutputParser`는 Pydantic의 데이터 유효성 검증 기능을 활용하여 LLM의 출력을 **엄격한 스키마에 맞게 변환**한다.

이는 LLM 출력의 불확실성을 제어하는 가장 강력한 방법이다.

3.2 동작 원리

1. 사용자는 `BaseModel`을 상속한 데이터 모델을 정의한다.
2. 파서는 `get_format_instructions()`를 통해 JSON 스키마를 생성한다.
3. 프롬프트에 이 지침을 삽입하여 모델이 해당 스키마에 맞는 출력을 생성하도록 유도한다.
4. `parse()` 단계에서 결과를 실제 Pydantic 객체로 변환하며, 유효성 검증을 수행한다.

3.3 예시

```
class EmailSummary(BaseModel):
    person: str
    email: str
    subject: str
    summary: str
    date: str
```

프롬프트에 삽입된 format instruction은 "위의 필드 구조에 맞게 JSON을 생성하라"는 명시적 지침이 된다.

결과는 다음처럼 구조화된다:

```
{
  "person": "김철수",
  "email": "chulsoo.kim@bikecorporation.me",
  "subject": "\"ZENESIS\" 자전거 유통 협력 및 미팅 일정 제안",
  "summary": "...",
  "date": "1월 15일 화요일 오전 10시"
}
```

3.4 특징

- 완전한 스키마 검증 제공
- `with_structured_output()`을 통해 LLM 객체에 직접 내장 가능
- 오류 발생 시 OutputFixingParser와 결합 가능
- 스트리밍은 지원하지 않음

3.5 활용 맥락

- 이메일 요약, 문서 요약, 보고서 생성 등 **정보 추출형 응용**
- LLM 출력을 DB나 API 응답으로 변환해야 하는 **엔터프라이즈 시스템**

4. CommaSeparatedListOutputParser — 간단한 항목 목록 반환

4.1 개요

쉼표(,)로 구분된 리스트를 반환하도록 유도하는 단순 파서.

LLM이 여러 개의 항목을 나열해야 하는 상황에서 사용된다.

4.2 구조

`get_format_instructions()`는 “출력을 쉼표로 구분하라”는 간단한 지시를 생성한다.

`parse()`는 문자열을 `str.split(",")` 방식으로 리스트로 변환한다.

4.3 사용 예시

```
prompt = PromptTemplate(..., partial_variables={"format_instructions": output_parser.get_for
mat_instructions()})
chain = prompt | ChatOpenAI() | output_parser
```

입력: “List five Korean landmarks”

출력: `['경복궁', '인사동', '해운대해수욕장', '제주도', '남산타워']`

4.4 특징

- Lightweight 파서로 속도 빠름
- 단순 키워드, 개념 나열에 적합

- 모델의 일관된 항목 분리 훈련에 유용

5. StructuredOutputParser — 간단한 Key-Value 구조

5.1 개요

`ResponseSchema` 객체로 스키마를 정의하고, 이를 바탕으로 LLM의 출력을 key-value 쌍으로 정리한다.

Pydantic보다 유연하지만, 검증 수준은 낮다.

5.2 예시

```
response_schemas = [
    ResponseSchema(name="answer", description="사용자의 질문에 대한 답변"),
    ResponseSchema(name="source", description="참고한 웹사이트 주소")
]
parser = StructuredOutputParser.from_response_schemas(response_schemas)
```

출력 예:

```
{"answer": "서울", "source": "https://ko.wikipedia.org/wiki/서울"}
```

5.3 특징

- 로컬/경량 모델에서도 동작
- 간단한 QA 시스템에 적합
- Pydantic과 달리 엄격한 타입 검증은 수행하지 않음

6. JsonOutputParser — JSON 기반 정형 출력

6.1 개요

LLM이 JSON 구조를 직접 생성하도록 유도하는 파서.

특정 필드 이름과 구조를 지정할 수 있어 REST API, 데이터 교환에 적합하다.

6.2 두 가지 사용 방식

- **Pydantic 결합형**: JSON을 Pydantic 모델로 직접 변환
- **비-Pydantic형**: 스키마 없이 JSON 형식만 요청

6.3 예시

```
class Topic(BaseModel):
    description: str
```

```
    hashtags: str
parser = JsonOutputParser(pydantic_object=Topic)
```

결과 예:

```
{"description": "지구 온난화는 ...", "hashtags": "#지구온난화 #기후변화 #온실가스"}
```

6.4 특징

- 표준 JSON 직렬화 지원
- API 서버, 대시보드, 백엔드 연동에 유용
- 모델의 문법적 안정성이 필수

7. PandasDataFrameOutputParser — 데이터 프레임 질의 파서

7.1 개요

DataFrame 객체를 입력으로 주고, LLM이 자연어로 질의하면

이를 Pandas 명령어로 해석해 결과를 반환하는 파서.

7.2 내부 동작

1. `get_format_instructions()`에서 가능한 연산 유형(column, row, mean 등)을 규칙으로 제시
2. LLM이 이를 기반으로 명령 문자열을 생성
3. 파서가 해당 명령을 실제 DataFrame 연산으로 실행
4. 결과를 dict 형식으로 반환

7.3 예시

입력: “Retrieve the average of the Ages from row 0 to 4.”

출력: `{'mean': 31.2}`

7.4 응용

- 데이터 분석 LLM Assistant
- 자연어 기반 DataFrame 인터페이스

8. DatetimeOutputParser — 날짜·시간 파싱

8.1 개요

날짜·시간 문자열을 Python의 `datetime` 객체로 변환한다.

형식 문자열(`%Y-%m-%d`, `%H:%M`, `%p` 등)을 기반으로 정확한 변환 수행.

8.2 예시

```
parser = DatetimeOutputParser()
parser.format = "%Y-%m-%d"
chain = prompt | ChatOpenAI() | parser
chain.invoke({"question": "Google이 창업된 연도는?"})
```

출력: `datetime(1998, 9, 4)`

8.3 특징

- 시간 정보 자동 정규화
- 일정·이벤트 추출 시스템에 활용

9. EnumOutputParser — 선택형 범주 출력

9.1 개요

Enum 클래스를 기반으로 LLM 출력을 제한된 범주 안으로 정규화한다.

색상, 감정, 상태 등 고정된 분류 체계에서 유용하다.

9.2 예시

```
class Colors(Enum):
    RED = "빨간색"
    GREEN = "초록색"
    BLUE = "파란색"
parser = EnumOutputParser(enum=Colors)
```

입력: “하늘의 색깔은?” → 출력: `Colors.BLUE`

9.3 특징

- 분류(Classification) 태스크의 안정성 향상
- 텍스트 내 모호성 제거

10. OutputFixingParser — 오류 자동 수정 계층

10.1 개요

다른 파서를 감싸고, 형식 오류 발생 시 LLM을 재호출하여 자동으로 수정한다.

즉, PydanticOutputParser 등에서 ValidationError가 발생하면 이를 LLM이 스스로 고치도록 유도한다.

10.2 예시

```
parser = PydanticOutputParser(pydantic_object=Actor)
fixing_parser = OutputFixingParser.from_llm(parser=parser, llm=ChatOpenAI())
```

```
actor = fixing_parser.parse("{'name': 'Tom Hanks', 'film_names': ['Forrest Gump']}")
```

→ 결과: `Actor(name='Tom Hanks', film_names=['Forrest Gump'])`

10.3 특징

- 자동 보정 메커니즘 내장
- 복잡한 스키마 기반 태스크에서 실용적
- 재호출로 인한 비용 증가 가능성 존재

11. LangChain의 파서 시스템의 철학적 설계 의도

LangChain의 Output Parser 계층은 단순히 출력 포맷팅을 위한 유틸리티가 아니라,

LLM의 언어적 창의성(Naturalness)과 **시스템적 일관성(Structure)** 사이의 균형을 설계한 구조이다.

1. **입력의 구조화 (PromptTemplate)** — 모델에게 명확한 맥락 제공
2. **출력의 구조화 (OutputParser)** — 모델의 결과를 신뢰 가능한 데이터로 변환
3. **체인 내 통합 (| Operator)** — LLM을 함수형 데이터 파이프라인처럼 연결

이러한 구조는 LLM이 단순히 문장을 "생성"하는 존재가 아니라,

데이터를 해석하고 가공하는 연산 단위로 동작하게 만든다.

12. 비교 요약

파서명	구조화 수준	검증 강도	주요 활용	비고
PydanticOutputParser	매우 높음	강력	정형 데이터, API 응답	Validation, schema 기반
StructuredOutputParser	중간	보통	간단한 QA	경량 대안
JsonOutputParser	높음	강함	JSON API, 데이터 교환	포맷 유연
CommaSeparatedListOutputParser	낮음	약함	목록, 키워드	간결한 리스트 반환
PandasDataFrameOutputParser	특수	중간	데이터 질의	자연어 →DataFrame 연산
DatetimeOutputParser	특수	강함	일정/날짜 인식	datetime 변환
EnumOutputParser	중간	강함	분류 태스크	카테고리 안정화
OutputFixingParser	보조	매우 강함	자동 보정	LLM 재호출 기반

결론

LangChain의 Output Parser 시스템은 LLM 응용의 “출력 계층(Output Layer)”을 체계화한 설계로,

텍스트 생성 중심의 언어 모델을 **데이터 중심 시스템으로 전환**시킨다.

이 계층 덕분에 LangChain은 단순한 대화형 프레임워크를 넘어

구조적 신뢰성·데이터 검증·자동 오류 복원이 가능한 완전한 파이프라인을 구성할 수 있다.

즉, Output Parser는 LangChain이 “언어 모델”에서 “데이터 엔진”으로 발전하는 과정의 핵심 축이라 할 수 있다.