랭체인 Multi-Chain 구현

1.1 Multi-Chain

1.1 Multi-Chain

🔗 랭체인 Multi-Chain 구현 개념

LangChain에서 **Chain**은 하나의 입력을 받아 특정 작업을 수행하고 출력을 내는 모듈이다.
Multi-Chain은 여러 Chain을 **순차**, **조건 분기**, 또는 **병렬**로 연결하여 복잡한 작업 흐름(Workflow)을 구현하는 방식이다.
이를 활용하면 단순한 질의응답을 넘어, **다단계 reasoning**이나 **복합 데이터 처리 파이프라인**을 만들 수 있다.

1.1 Multi-Chain

랭체인 멀티-체인 구현 방식

랭체인에서 멀티-체인을 구현하는 주요 방식은 **라우팅(Routing)**과 **에이전트(Agent)**를 사용하는 것입니다.

- **라우팅**: 여러 체인 중 특정 조건에 맞는 체인을 선택해 요청을 보내는 방식입니다. 예를 들어, 사용자의 질의가 특정 도메인(예: 법률, 의료)과 관련될 경우, 해당 도메인에 특화된 체인으로 요청을 전달합니다. 이를 통해 각 체인이 특정 분야에 대해 더 깊이 있는 지식을 갖도록 할 수 있습니다.
- **에이전트**: 에이전트는 사용자의 질의를 분석해 어떤 도구(Tool)를 사용해야 할지 결정합니다. 여기서 각 체인은 **'도구'**가 될 수 있습니다. 에이전트가 "A"라는 체인에 요청을 보낼지, "B"라는 체인에 요청을 보낼지 스스로 판단하고 실행합니다. 이 방식은 보다 동적인 상황에서 유연하게 체인을 선택할 수 있게 합니다.

1.1 Multi-Chain

🔗 Multi-Chain의 주요 유형 (신버전)

1. 순차 연결 (Pipeline with `|` Operator)

- 여러 **Runnable** 컴포넌트를 **앞→뒤 순서**로 연결하여, 이전 단계의 출력을 다음 단계의 입력으로 자동 전달.
- `SequentialChain`을 따로 만들 필요 없이 `프롬프트 | LLM | 프롬프트 | LLM` 형태로 한 줄로 표현 가능.
- 예시:
  - 1단계: 긴 텍스트 요약
  - 2단계: 요약 결과 영어 번역
  - 3단계: 번역 결과 감정 분석
- 장점:
  - 코드가 간결하고 가독성이 높음
  - 입력/출력 변수 이름을 명시할 필요가 없음
  - 디버깅 시 단계별 결과 확인 가능 (`verbose=True` 옵션)

💡 핵심 포인트

- **파이프 연산자(`|`)** = 자동 순차 연결
- **Output Parser** 연결 시 결과를 문자열·JSON 등 원하는 형식으로 자동 변환 가능
- **재사용성 높음** → 각 단계(프롬프트, LLM)를 모듈처럼 조립 가능

1.1 Multi-Chain

2. 조건 분기 (Router / Multi-Prompt Chain, 신버전)

- **개념**

  입력된 질문이나 요청을 먼저 분석하여 **어떤 체인(또는 프롬프트)**으로 보낼지 결정하는 라우터 역할.

  최근 버전에서는 `RouterChain` 또는 `RunnableBranch`를 활용해 구현.

- **동작 방식**
  1. 입력을 받아 질문의 **유형·의도** 분류
  2. 의도에 맞는 체인으로 **라우팅**
  3. 선택된 체인에서 **결과 생성 후 반환**
- **예시 시나리오**
  - 계산 문제 → 계산 체인 실행
  - 번역 요청 → 번역 체인 실행
  - 일반 대화 → 기본 대화 체인 실행

1.1 Multi-Chain

- **장점**
  - 하나의 시스템에서 다양한 요청을 처리할 수 있어 **확장성**이 높음
  - 새로운 요청 유형이 생기면 라우터에 체인을 추가하는 것만으로 확장 가능
  - 모듈화된 구조라 유지보수가 쉬움

💡 최신 구현 포인트

- **RouterChain + LLM 기반 라우팅**

  → LLM이 직접 "이 입력은 번역 vs 수학 vs 대화 중 어디로 가야 하는가?"를 판단
- **RunnableBranch** 사용 시 코드가 더 직관적

  → if/else 대신 브랜치별 조건과 체인을 깔끔하게 정의 가능

1.1 Multi-Chain

3. 병렬 실행 (Parallel Chain, 신버전)

- **개념**

  하나의 입력을 여러 체인에 **동시에 전달**하여 각각 결과를 생성하고, 이후 결과를 모아 최종 응답을 만든다.

  최근 버전에서는 `RunnableParallel` 이나 `RunnableMap` 을 활용해 구현.

- **동작 방식**
  1. 동일 입력을 여러 체인에 동시에 보냄
  2. 각 체인이 독립적으로 결과 생성
  3. 모든 결과를 합쳐 최종 응답 반환
- **예시 시나리오**
  - 같은 텍스트를 **요약 + 감정 분석 + 키워드 추출**을 동시에 실행
  - 여러 모델에서 동시에 답변 생성 → 성능 비교 후 최적 응답 선택

1.1 Multi-Chain

- **장점**
  - 응답 시간을 단축할 수 있음 (비동기 실행)
  - 다양한 분석 결과를 종합해 더 풍부하고 신뢰성 있는 답변 제공
- **신버전 구현 포인트**
  - `RunnableParallel` 로 여러 체인을 한 번에 실행
  - 결과는 `dict` 형태로 반환되어 후처리하기 편리
  - `asyncio` 기반으로 비동기 실행 지원 → 속도 향상

1.1 Multi-Chain

🎯 Multi-Chain의 활용 예시

- **사내 지식 검색 에이전트**: 질문 → 관련 문서 검색 → 요약 → 답변 생성
- **고객 상담 챗봇**: 입력 분석 → FAQ 검색 Chain / 상담 예약 Chain / 일반 대화 Chain으로 분기
- **데이터 분석 자동화**: 데이터 로딩 → 통계 계산 → 시각화 보고서 생성

1.2 출력 파서 (Output Parser)

1.2 출력 파서 (Output Parser)

📜 2. 출력 파서 (Output Parser)

✅ 개념

- LLM의 출력은 대부분 **자연어 텍스트**이지만, 실제 프로그램에서는 구조화된 데이터가 필요할 때가 많다.
- **Output Parser**는 LLM의 출력 결과를 **딕셔너리, 리스트, JSON, Pydantic 모델** 등으로 변환하는 도구이다.

🔧 주요 기능

- **구조화**: 자유 텍스트 → 키-값 쌍으로 파싱
- **검증**: 응답 형식이 지정한 스키마와 맞는지 검사
- **자동 재요청**: 응답 형식이 잘못된 경우 LLM에 다시 요청 (re-ask)

1.2 출력 파서 (Output Parser)

🎯 사용 예시

- "이 텍스트에서 키워드 3개 뽑아줘" → `["키워드1", "키워드2", "키워드3"]` 형태로 변환
- 채팅 응답에서 `"answer": "...", "confidence": 0.92` 형태로 정리
- JSON 형식으로 응답을 강제 → 이후 코드에서 바로 활용 가능

1.2 출력 파서 (Output Parser)

출력 파서(output parser) 사용하기

```python
from langchain_core.output_parsers import StrOutputParser

parser = StrOutputParser()    # 텍스트만 추출하여 반환하는 파서 객체 생성

result = model.invoke(messages)
parser.invoke(result)  # 문자열(content)만 출력
```

1.3 파이프 연산자("|")

1.3 파이프 연산자("|")

🔗 3. 파이프 연산자 ("|")

✅ 개념

- LangChain 0.2 이후, 여러 컴포넌트를 **함수형 스타일**로 연결할 때 `|` **연산자**를 사용.
- **데이터 흐름**을 직관적으로 표현할 수 있고, 읽기 쉬운 코드 작성 가능.

🔧 특징

- 왼쪽 → 오른쪽으로 데이터가 흐름
- 프롬프트 → LLM → 출력 파서 순서로 연결 가능
- 여러 단계의 Chain을 단순하게 표현

🎯 사용 예시 (개념 설명)

- `프롬프트 | 모델 | 파서`
- "이 텍스트 요약 → 모델에 입력 → 결과를 JSON으로 파싱" 과 같은 형태로 한 줄에 표현 가능
- 기존의 `SequentialChain` 보다 직관적이며, 간단한 연결에 적합

1.3 파이프 연산자("|")

파이프 연산자("|") 사용하기

```
1 chain = model | parser
2 chain.invoke(messages)
```

1.4 프롬프트 템플릿

1.4 프롬프트 템플릿

📝 4. 프롬프트 템플릿 (PromptTemplate)

✅ 개념

- **PromptTemplate**은 프롬프트 안의 변수를 `{}`로 정의해 **동적으로 프롬프트를 생성**하는 도구이다.
- 코드에서 프롬프트를 하드코딩하지 않고, 매번 다른 입력 값을 넣어 재사용 가능.

🔧 주요 기능

- **변수 치환**: `{product}`, `{language}` 같은 변수에 실제 값 대입
- **템플릿 관리**: 여러 프롬프트를 체계적으로 관리 가능
- **LLM 프롬프트 엔지니어링**의 기초

🎯 사용 예시

- 프롬프트: `"다음 제품 설명을 {language}로 번역해줘:\n{description}"`
- 실행 시 변수 대입: `language="영어", description="이 제품은 휴대용 배터리입니다."`
- 최종 생성 프롬프트:

```
다음 제품 설명을 영어로 번역해줘:
이 제품은 휴대용 배터리입니다.
```

1.4 프롬프트 템플릿

```python
from langchain_core.prompts import ChatPromptTemplate

system_template = "너는 {story}에 나오는 {character_a} 역할이다. 그 캐릭터에 맞게 사용자와 대화하라."
human_template = "안녕? 나는 {character_b}입니다. 오늘 시간 괜찮으시면 {activity}에 같이 갈까요?"

prompt_template = ChatPromptTemplate([
    ("system", system_template),
    ("user", human_template),
])

result = prompt_template.invoke({
    "story": "춘향전",
    "character_a": "성춘향",
    "character_b": "변사또",
    "activity": "잔치"
})

print(result)
```

1.4 프롬프트 템플릿

```
chain = prompt_template | model | parser

chain.invoke({
    "story": "춘향전",
    "character_a": "성춘향",
    "character_b": "변사또",
    "activity": "잔치"
})
```

1.5 Runnable 클래스 종류

1.5 Runnable 클래스 종류

Runnable은 기본적으로 Runnable 프로토콜을 구현한 객체들이고, 서로 파이프(|)로 연결하거나 병렬, 분기 등으로 조합 가능하다.

1. 기본 Runnable

이들은 가장 자주 쓰이는 Runnable 클래스들이다.

Runnable	설명
RunnableLambda	파이썬 함수나 람다를 Runnable로 감싸서 실행 가능하게 만듦
RunnableMap	여러 Runnable을 병렬로 실행하고 결과를 딕셔너리로 반환
RunnableSequence	여러 Runnable을 순차적으로 실행 (파이프라인)
RunnableBranch	조건 분기용 Runnable – 입력값에 따라 특정 Runnable 실행
RunnableParallel	여러 Runnable을 동시에 실행하고 결과 합침

1.5 Runnable 클래스 종류

2. 입출력 변환용 Runnable

데이터 변환을 담당하는 Runnable들.

Runnable	설명
RunnablePassthrough	입력을 그대로 다음 단계로 전달
RunnableAssign	상태(State)에 특정 key-value를 추가 (새 필드 생성)
RunnablePick	입력에서 특정 key만 골라서 전달
RunnableEach	리스트 형태 입력을 받아 각 요소에 같은 Runnable 적용

1.5 Runnable 클래스 종류

3. 모델/프롬프트 관련 Runnable

LangChain에서 모델 호출, 프롬프트 구성 등을 감싼 Runnable들.

Runnable	설명
RunnableBinding	Runnable에 설정(config)이나 options을 미리 바인딩
RunnableParallel	여러 모델/프롬프트를 동시에 실행
RunnableWithMessageHistory	대화 이력을 유지하며 Runnable 실행
ChatPromptTemplate (Runnable 역할)	입력을 LLM Prompt로 변환

1.5 Runnable 클래스 종류

4. 흐름 제어 및 유틸리티

보다 복잡한 워크플로우 구현 시 쓰이는 것들.

Runnable	설명
RunnableRetry	실패 시 재시도 로직 추가
RunnableWithFallbacks	실패 시 대체 Runnable 실행
RunnableAssign	중간 단계에서 데이터 가공 후 상태에 추가
RunnablePick	딕셔너리에서 특정 key만 선택

1.5 Runnable 클래스 종류

5. LangGraph와의 연결

LangGraph 노드에 들어가는 것도 사실 전부 Runnable이다.
즉, 노드의 동작 자체가 Runnable로 정의되기 때문에, 위 모든 종류를 그대로 활용할 수 있다.

---

6. 간단 예시

python

```python
from langchain_core.runnables import RunnableLambda, RunnableMap, RunnableBranch

# 1) RunnableLambda
to_upper = RunnableLambda(lambda x: x.upper())

# 2) RunnableMap - 병렬 실행
multi = RunnableMap({"upper": to_upper, "length": RunnableLambda(lambda x: len(x))})
```

1.5 Runnable 클래스 종류

```
# 3) RunnableBranch - 조건 분기
branch = RunnableBranch(
    branches=[
        (lambda x: "HELLO" in x, RunnableLambda(lambda x: "인사 감지됨")),
        (lambda x: True, RunnableLambda(lambda x: "그 외 입력"))
    ]
)

print(to_upper.invoke("hello"))
print(multi.invoke("hello"))
print(branch.invoke("HELLO WORLD"))
```

출력 예:

```
bash

HELLO
{'upper': 'HELLO', 'length': 5}
인사 감지됨
```

1.5 Runnable 클래스 종류

🔑 핵심 포인트

1. **데이터 변환 계열**
   - `RunnableLambda`, `RunnableAssign`, `RunnablePick`, `RunnablePassthrough`
   - 데이터를 가공하거나 필요한 key만 추출하는 데 유용
2. **흐름 제어 계열**
   - `RunnableSequence`, `RunnableBranch`, `RunnableParallel`, `RunnableEach`
   - 직렬, 병렬, 조건 분기, 반복 실행을 조합해 워크플로우 구축 가능
3. **에러/옵션 처리 계열**
   - `RunnableRetry`, `RunnableWithFallbacks`, `RunnableBinding`
   - 안정성 높이고 재시도·대체 경로 지원
4. **대화형 사용**
   - `RunnableWithMessageHistory` → LangGraph와 연결해 stateful chatbot 구현 시 필수

감사합니다