Spring AI 기초

학습 목표

아키텍처 이해

Spring AI의 핵심 구성 요소인 Advisor, OutputConverter의 동작 원리를 파악하고, LLM 추론 성능을 극대화하는 프롬프트 전략을 설명할 수 있습니다.

실전 구현

ChatClient의 Fluent API로 실시간 스트리밍 서비스를 구현하고, 다양한 저장소를 활용한 대화 기억 기능을 프로젝트에 적용할 수 있습니다.

보안과 윤리

세이프가드 Advisor를 통해 AI 서비스의 보안성과 윤리적 책임을 준수하며, 최신 AI 생태계 변화에 능동적으로 대응하는 자세를 갖출 수 있습니다.

AI 애플리케이션 개발의 도전과 해결

개발의 어려움

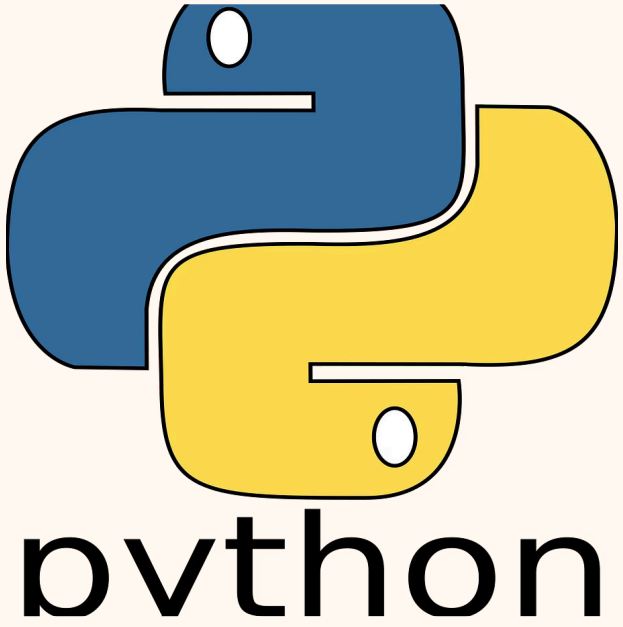
- 복잡한 데이터 흐름 관리
- 유지보수의 어려움
- 표준화된 프레임워크 필요

Spring AI의 강점

Python의 LangChain과 동등한 수준의 고도화된 기능을 제공하며, 자바 개발자에게 친숙한 Spring 생태계와 완벽하게 통합됩니다.

다양한 LLM 지원, 엔터프라이즈급 연동, RAG와 도구 호출은 물론 MCP 서버 개발까지 지원합니다.

LangChain vs Spring AI



LangChain (Python)

프롬프트-응답 단계를 체인으로 연결하는 Python/Node.js 기반 프레임워크입니다.

Spring AI (Java)

스프링 빈으로 자동 주입된 API로 프롬프트-응답을 메소드 호출로 처리하는 Java 기반 프레임워크입니다.

두 프레임워크 모두 대화 기억, 구조화된 출력, 벡터 저장소, 도구 호출, 멀티모달, 비동기/스트리밍, RAG를 지원하지만, Spring AI는 MCP Server 개발까지 지원합니다.

개발환경 구축

01

OpenAI API Key 발급

platform.openai.com에서 가입 후 Settings → API keys에서 새 키를 생성하고 환경 변수로 등록합니다. 최소 $5 이상 크레딧 충전이 필요합니다.

02

프로젝트 생성

Spring Boot 3.5.x, Java 25, Gradle을 선택하고 Spring Web, Spring Reactive Web, OpenAI, Thymeleaf, Lombok 의존성을 추가합니다.

03

설정 파일 구성

application.yaml에 OpenAI API 키와 파일 업로드, 정적 리소스 캐시 설정을 추가합니다.

Chat Model API 핵심 개념

1

ChatModel

동기 방식으로 전체 문장이 완성된 후 한 번에 반환하며, 데이터 분석과 백엔드 로직 처리에 적합합니다.

2

StreamingChatModel

비동기 스트림 방식으로 첫 토큰 생성 즉시 순차적으로 반환하며, 챗봇 UI와 실시간 텍스트 생성에 최적화되어 있습니다.

Spring AI 1.0 이후 ChatModel이 StreamingChatModel을 직접 확장하여 하나의 Bean 주입으로 두 방식을 모두 사용할 수 있습니다.

Message 인터페이스와 데이터 흐름

1. **SystemMessage**

   AI의 역할, 말투, 제약 사항 등 기본 지침을 설정합니다.

2. **UserMessage**

   사용자가 입력한 질문이나 명령을 전달하며 멀티모달을 지원합니다.

3. **AssistantMessage**

   AI의 응답으로 대화 기억 유지에 사용되어 일관된 대화를 가능하게 합니다.

4. **ToolResponseMessage**

   외부 도구 실행 후 전달되는 결과 값을 담습니다.

ChatClient로 어린왕자 챗봇 구현

구현 특징

- Fluent API로 간결한 코드
- 동기/스트리밍 모두 지원
- 페르소나 기반 응답
- 실시간 대화 경험

ChatClient는 복잡한 객체 생성 없이 메서드 체이닝으로 시스템 메시지, 사용자 질문, 옵션 설정을 직관적으로 구성할 수 있습니다.

system() 메서드로 어린왕자의 페르소나를 주입하고, user() 메서드로 질문을 전달하며, call() 또는 stream()으로 실행 방식을 선택합니다.

프롬프트 엔지니어링 전략

제로-샷 프롬프트

예시 없이 지시사항만으로 작업을 수행하게 하는 가장 간단한 방식입니다.

퓨-샷 프롬프트

예시 데이터를 제공하여 문맥 내에서 학습하게 하는 방식으로 정확도를 높입니다.

역할 부여

특정 페르소나를 주입하여 전문성과 톤을 조절하는 방식입니다.

생각의 사슬

중간 논리 과정을 나열하게 하여 복잡한 문제의 정확도를 높이는 방식입니다.

스텝-백

구체적 질문에서 추상적 배경 원리로 후퇴하여 오류를 방지하는 방식입니다.

자기 일관성

동일 질문을 여러 번 요청하여 다수결로 정확도를 높이는 방식입니다.

구조화된 출력 (Structured Output)

OutputConverter 종류

ListOutputConverter

List<String> 형태로 변환

BeanOutputConverter

Java 객체(POJO)로 변환

MapOutputConverter

Map<String, Object>로 변환

LLM의 텍스트 응답을 애플리케이션에서 사용 가능한 데이터 구조로 자동 변환합니다.

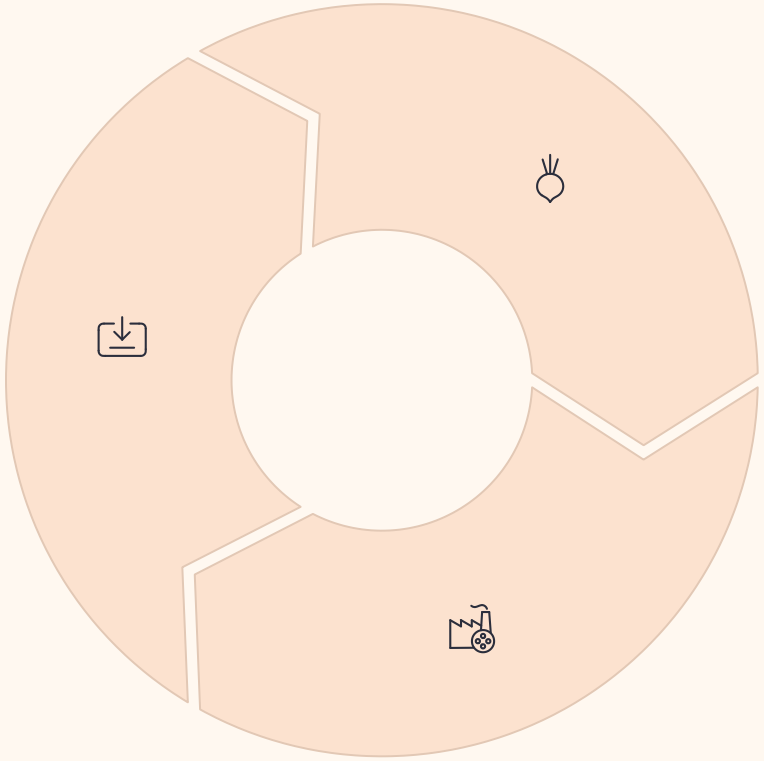
형식 지침 생성, 데이터 추출, 객체 변환의 세 단계를 StructuredOutputConverter 인터페이스가 표준화합니다.

고수준 API인 entity() 메서드를 사용하면 변환 과정이 자동으로 처리됩니다.

Advisor: AI 파이프라인의 인터셉터

전처리

프롬프트 강화, 컨텍스트 추가, RAG 데이터 주입



LLM 호출

가공된 프롬프트로 모델 실행

후처리

응답 검증, 변환, 기록 저장

Advisor는 Spring의 AOP 패턴을 AI 파이프라인에 적용하여 공통 로직을 비즈니스 로직에서 분리합니다. 여러 Advisor를 체인으로 구성하여 순차적으로 실행할 수 있으며, Ordered 인터페이스로 우선순위를 제어합니다.

내장 Advisor 활용

SimpleLoggerAdvisor

요청과 응답 내용을 로깅하여 디버깅과 모니터링을 지원합니다.

SafeGuardAdvisor

민감한 단어가 포함된 질문을 차단하여 보안과 윤리를 준수합니다.

MessageChatMemoryAdvisor

대화 기억을 메시지 객체로 프롬프트에 추가하여 문맥을 유지합니다.

QuestionAnswerAdvisor

벡터 저장소에서 관련 내용을 조회하여 RAG를 구현합니다.

대화 기억 저장소 비교

InMemory 애플리케이션 메모리에 저장하는 가장 빠른 방식이지만 재시작 시 데이터가 소실됩니다.	**RDBMS (JDBC)** PostgreSQL, MySQL 등에 영구 저장하며 스키마 자동 초기화를 지원합니다.
VectorStore 유사성 기반 검색으로 관련성 높은 대화만 선별하여 효율적인 컨텍스트 관리가 가능합니다.	**Cassandra** 분산 NoSQL로 대규모 데이터를 처리하며 TTL 자동 삭제 기능을 제공합니다.

대화 기억 구현 패턴

MessageChatMemoryAdvisor

대화 이력을 독립적인 메시지 객체들의 묶음으로 처리하여 구조가 명확히 유지됩니다.

LLM이 '누가 어떤 말을 했는지' 객체 단위로 파악할 수 있어 표준적인 대화형 인터페이스에 적합합니다.

PromptChatMemoryAdvisor

과거 대화 내용을 하나의 긴 텍스트로 변환하여 시스템 텍스트 내에 포함시킵니다.

시스템 지침과 대화 이력을 단일 텍스트 흐름 안에서 관리하고 싶을 때 유용합니다.

두 Advisor 모두 전처리에서 과거 기록을 조회하고, 후처리에서 새로운 질문과 답변을 자동으로 저장합니다.