Capítulo 18 – A Virada de Chave (Bem-vindos ao Spring AI)

Aléssio Miranda Jr.

Outubro - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

A Inteligência Artificial na indústria corporativa não roda em scripts Python descartáveis. Ela roda em aplicações Java robustas, com injeção de dependências, tratamento de exceções, APIs REST e pipelines CI/CD. Neste capítulo, apresentaremos o **Spring AI** — a extensão oficial do Spring Framework para integração com LLMs —, e construiremos nossa primeira aplicação Java que consome a API do Gemini/OpenAI através de uma arquitetura MVC profissional.

1 Por que Java? Por que Spring?

No Arco 2 e início do Arco 3, utilizamos Python para toda a cadeia (Embeddings, ChromaDB, chamadas à API). Python é imbatível para prototipagem rápida, mas em ambientes corporativos brasileiros (bancos, seguradoras, indústria), o **Java** domina o *back-end* por motivos estruturais:

- **Tipagem estática:** Erros de tipo são detectados em tempo de compilação, não em produção.
- **Ecossistema maduro:** Injeção de dependências, Spring Security, Spring Data JPA, filas de mensagens (Kafka, RabbitMQ).
- **Escalabilidade:** A JVM (Java Virtual Machine) é otimizada para servidores que rodam 24/7 com milhares de requisições concorrentes.

- **Mercado de trabalho:** Java é a linguagem mais demandada para *back-end* no Brasil (segundo o StackOverflow Survey e dados do LinkedIn).

O **Spring Boot** é o framework Java mais popular do mundo para construção de APIs REST e microsserviços. O **Spring AI** é o módulo oficial (lançado em 2024) que estende o Spring Boot com integrações nativas para LLMs, Embeddings e Vector Stores.

2 A Abstração do ChatClient

O Spring AI introduz a interface `ChatClient` — uma abstração que unifica a comunicação com **qualquer** provedor de LLM sob a mesma API Java:

• Teoria: title=Agnóstico de Provedor

Com o `ChatClient`, seu código Java chama o mesmo método independentemente de estar usando OpenAI, Google Gemini, Anthropic Claude, ou um modelo local via Ollama. Para trocar de provedor, basta alterar **uma linha no** `application.properties`. Zero alteração no código de negócio.

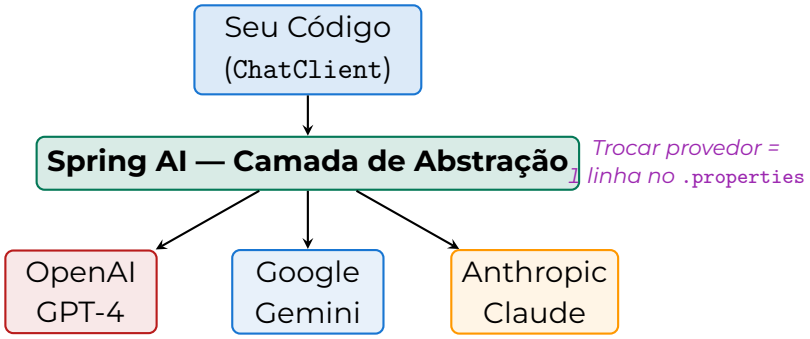

Figura 1: O Spring AI atua como camada de abstração: seu código chama sempre o mesmo `ChatClient`, independente do provedor na nuvem.

Tabela 1: Starters disponíveis no Spring AI por provedor.

Provedor	Starter (artifactId)	Modelos
OpenAI	`spring-ai-openai-*`	GPT-4o, GPT-3.5
Google	`spring-ai-vertex-ai-gemini-*`	Gemini Pro, Flash
Anthropic	`spring-ai-anthropic-*`	Claude 3.5/3
Ollama (Local)	`spring-ai-ollama-*`	LLaMA, Mistral

▶ Prática: title=Primeira Chamada ao LLM com Spring AI

```java
// ChatController.java
@RestController
public class ChatController {

    private final ChatClient chatClient;

    // Injecao de dependencia automatica
    public ChatController(ChatClient.Builder builder) {
        this.chatClient = builder.build();
    }

    @GetMapping("/chat")
    public String chat(@RequestParam String pergunta) {
        return chatClient.prompt()
            .user(pergunta)
            .call()
            .content();
    }
}
```

No application.properties:

```
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.options.model=gpt-4o
spring.ai.openai.chat.options.temperature=0.7
```

3 Outputs Estruturados (BeanOutputParser)

Uma das maiores vantagens do Spring AI sobre scripts Python é a capacidade de mapear a saída do LLM diretamente para um **objeto Java tipado**:

```java
// Definir o POJO (Plain Old Java Object)
public record Entidade(
    String nome,
    String empresa,
    String cargo
) {}

// No controller:
Entidade entidade = chatClient.prompt()
    .user("Extraia: Maria Silva, engenheira da Petrobras")
    .call()
    .entity(Entidade.class);

System.out.println(entidade.nome()); // "Maria Silva"
```

O Spring AI utiliza internamente o `BeanOutputParser`, que injeta no Prompt as instruções de formatação JSON e, na resposta, desserializa automaticamente o JSON para o `Record` Java. O desenvolvedor nunca toca em regex nem em parsing manual.

4 Configuração e Dependências

Para iniciar um projeto Spring AI, basta adicionar a dependência no `pom.xml` (Maven) ou `build.gradle`:

```
<!-- pom.xml (Maven) -->
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-openai-spring-boot-starter</artifactId>
</dependency>
```

◇ Informação

O Spring AI está em evolução rápida (milestone releases). Consulte sempre a documentação oficial em `docs.spring.io/spring-ai` para a versão mais recente dos starters e das APIs disponíveis.

5 Conclusão

O salto do Python para o Spring AI não é apenas uma troca de linguagem — é uma mudança de **paradigma arquitetural**. Com injeção de dependências, tipagem estática, outputs estruturados e agnóstica de provedores, o Spring AI transforma o LLM em um componente de software tão confiável quanto um repositório JPA ou um serviço REST. Na próxima aula, combinaremos o Spring AI com o Vector Store para construir o **RAG Corporativo** completo.

✔ Takeaways

- O **Spring AI** é o módulo oficial do Spring Framework para integração com LLMs, lançado em 2024, que abstrai provedores sob uma API Java unificada.
- O `ChatClient` é agnóstico de provedor: trocar de OpenAI para Gemini requer alterar **uma linha** no `application.properties`, sem modificar código de negócio.
- A transição de Python para Java/Spring não é luxo — é uma exigência do mercado corporativo brasileiro (bancos, seguradoras, indústria), onde Java domina o back-end.
- O `BeanOutputParser` mapeia a saída do LLM diretamente para **objetos Java tipados** (Records), eliminando parsing manual e regex.
- A **tipagem estática** do Java detecta erros em tempo de compilação, não em produção — uma vantagem crítica para sistemas que operam 24/7.
- O ecossistema Spring (Security, Data JPA, Cloud) integra-se nativamente com o Spring AI, formando um stack corporativo completo.

✎ Questões: Autoavaliação

1 **[Direta]** Explique como o Spring AI abstrai os provedores de LLM. Quais classes e configurações são necessárias para trocar de OpenAI para Google Gemini?

2 **[Direta]** Descreva o fluxo do `BeanOutputParser`: como ele transforma a saída textual do LLM em um objeto Java tipado? Quais são as vantagens sobre parsing manual em Python?

3 **[Direta]** Cite três motivos estruturais pelos quais Java domina o back-end corporativo brasileiro e explique como cada um se traduz em vantagem prática para sistemas de IA.

4 **[Reflexiva]** Compare criticamente a experiência de desenvolver um RAG em Python puro (Aula 17) versus Spring AI. Em que cenários cada abordagem é mais adequada?

5 **[Reflexiva]** O Spring AI está em "milestone releases" (evolução rápida). Quais riscos isso traz para projetos corporativos de longo prazo e como a equipe de engenharia pode mitigar esses riscos?

Lab. 18: Olá Mundo com Spring AI

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

Migraremos nosso escopo do Python para o ecossistema corporativo Java. O objetivo é inicializar um projeto Spring Boot e utilizar o Spring AI para executar um prompt simples, atestando a injeção de dependências e a conectividade com o modelo de nuvem via Java.

6 Atividade Prática

1. Inicializando o Projeto

1 Acesse `https://start.spring.io/` (Spring Initializr).

2 Configure as opções do Projeto:

- **Project:** Maven
- **Language:** Java (versão 17 ou superior)
- **Spring Boot:** Versão 3.2.x ou superior
- **Group:** `br.edu.instituicao`
- **Artifact:** `spring-ai-demo`

3 Clique em **Add Dependencies** e busque por:

- **Spring Web** (Para construir REST APIs futuramente)
- **OpenAI** (Na seção de AI, ou Gemini, conforme escolha).

4 Gere o projeto (botão GENERATE), extraia o arquivo `.zip` e abra na sua IDE (IntelliJ, Eclipse ou VSCode).

2. Configuração das Credenciais

No diretório `src/main/resources/`, abra o arquivo `application.properties` e adicione as configurações de conexão:

```
# Desabilitando a necessidade de uma chave na compilação do Spring (Fallback)
spring.ai.openai.api-key=${OPENAI_API_KEY}
spring.ai.openai.chat.options.model=gpt-3.5-turbo
```

Nota: Exporte a variável `OPENAI_API_KEY` *no seu terminal ou na configuração de Run da sua IDE antes de executar a aplicação.*

3. O Código: O Controlador REST

No mesmo pacote da sua classe principal (a que possui `@SpringBootApplication`), crie a classe `ChatController.java`:

```
package br.edu.instituicao.springaidemo;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ChatController {

    private final ChatClient chatClient;

    // A mágica da Injeção de Dependência do Spring
    public ChatController(ChatClient.Builder chatClientBuilder) {
        this.chatClient = chatClientBuilder.build();
    }

    @GetMapping("/api/pergunta")
    public String fazerPergunta(@RequestParam(value = "msg", defaultValue = "Me diga u
        return chatClient.prompt()
                .user(msg)
                .call()
                .content();
    }
}
```

4. Execução e Entrega via Git

1. Rode a aplicação (no IntelliJ, execute o método `main` da classe base).
2. Abra o navegador ou o Postman na URL: `http://localhost:8080/api/pergunta?msg=Como a internet funciona`

3 Você deverá receber o texto gerado pela IA no navegador.

4 Copie a pasta do projeto para o seu repositório Git local, realize o rastreamento (`git add .`) assegurando que os arquivos compilados (`target/`) estão no `.gitignore`.

5 Salve e empurre para a nuvem: `git commit -m "Cria esqueleto Spring AI e expõe endpoint generativo"`.

7 Critério de Avaliação

Presença do código Java válido no GitLab, onde se constate o uso nativo do `ChatClient` para orquestrar a chamada generativa, encapsulado adequadamente num `@RestController`.

✔ Takeaways

- Você construiu seu primeiro **endpoint REST com IA generativa** em Java — o Spring Boot inicializou, injetou o `ChatClient` e expôs a rota `/api/pergunta`.
- A **Injeção de Dependências** do Spring fez a mágica: o `ChatClient.Builder` é automaticamente configurado pela dependência do starter, sem setup manual.
- A chave da API é lida de variável de ambiente (`${OPENAI_API_KEY}`) — padrão corporativo que separa credenciais de código.
- O endpoint pode ser testado via navegador ou Postman — exatamente como qualquer API REST que você já conhece.

◇ Informação

Gancho para a Próxima Aula: O endpoint atual responde com base no conhecimento geral do LLM. Na próxima aula, adicionaremos o **VectorStore** e o **QuestionAnswerAdvisor** para construir o RAG corporativo completo em Spring AI — o LLM passará a responder usando **seus documentos**.