

Hands-On Spring AI

Build Intelligent Java Applications with RAG, MCP and AI Agents



Fu Cheng

Hands-on Spring AI

Build Intelligent Java Applications with RAG, MCP and AI Agents

Fu Cheng

This book is available at https://leanpub.com/spring-ai

This version was published on 2026-10-16



This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2026 Fu Cheng

Also By Fu Cheng

Exploring Java 25
Text-to-SQL, Spring AI Implementation with RAG
Understanding Java Virtual Threads
From Java 21 to Java 25
From Java 17 to Java 21
Build Native Java Apps with GraalVM
From Java 11 to Java 17
ES6 Generators
A Practical Guide for Java 8 Lambdas and Streams
Lodash 4 Cookbook
JUnit 5 Cookbook

Contents

Spring AI Basics 1

Getting Started **2**
- Prerequisites 2
- Spring Boot Application 5
- Use Model Service 8
- Consolidate Local and Production Environment 8
- Summary 13

Chat Completion **14**
- Model 14
- ChatModel 14
- Create Prompts 15
- Chat Response 15
- ChatClient 15
- Use Custom Chat Options 16
- Prompt Template 16
- Advisor 17

Conversation and chat memory **19**
- From single requests to conversations 19
- ChatMemory 19
- MessageChatMemoryAdvisor 19
- A conversational REST API 19
- Persistent memory 20
- Failures and concurrent requests 20
- Summary 20

Tool **21**
- Basic Tool Usage 21
- ToolCallback 21
- ToolCallbackProvider 22

Streaming Chat Completion **23**
StreamingChatModel 23
Streaming Web Application 23
Streaming JSON Events 23

Structured Output Converter **24**
StructuredOutputConverter 24
ListOutputConverter 24
MapOutputConverter 24
BeanOutputConverter 24
Use ChatClient 24

Multimodal Input **25**
Media 25
Image Understanding 25

Retrieval-Augmented Generation 26

RAG Introduction **27**
Reduce Hallucinations 27
Naive RAG 29

Embedding Model **32**
EmbeddingModel 32
EmbeddingOptions 32
BatchingStrategy 32
Use EmbeddingModel 32

Documents **33**
Create Documents 33
Document Reader 33
Document Transformer 33
Document Writer 33

Vector Store **34**
VectorStore 34
Create `VectorStore` 34
Add Documents 34
Delete Documents 34
Similarity Search 34

VectorStore REST API . . . 34
SimpleVectorStore . . . 35
Pgvector . . . 35
Vector Store Cloud Services . . . 35

RAG . . . 37
Simple RAG . . . 37

Modular RAG . . . 38
Query . . . 38
Pre-Retrieval . . . 38
Retrieval . . . 39
Post-Retrieval . . . 39
Generation . . . 40
RetrievalAugmentationAdvisor . . . 40

RAG Examples . . . 41
Text-to-SQL . . . 41
PDF Q&A . . . 42

MCP . . . 44

MCP Introduction . . . 45

Quick Start . . . 46
Java Development Basics . . . 46
Sample MCP Server . . . 47
stdio Server . . . 48
HTTP SSE Server . . . 54
MCP Client . . . 57
Spring Integration . . . 60
Spring AI Integration . . . 63

MCP Server . . . 69
Shared Models . . . 69
ServerExchange . . . 70
Prompt Templates . . . 70
Resources . . . 71
Tools . . . 72
Completions . . . 72

Logging . . . 72
Pagination . . . 72

MCP Client . . . 74
Roots . . . 74
Sampling . . . 74

MCP Annotations . . . 75
Server Annotations . . . 75
Client Annotations . . . 75
Special Parameters . . . 76

MCP Examples . . . 78
File System MCP Server . . . 78
Database MCP Server . . . 78

Agent . . . 80

Agent Introduction . . . 81
Cooking Suggestion Agent . . . 82

Agent Components . . . 84
Profile . . . 84
Persona . . . 84
Tools . . . 84
Knowledge and memory . . . 84
Reasoning and evaluation . . . 84
Planning and feedback . . . 84

Agentic Patterns . . . 86
Task Execution . . . 86
Evaluator-Optimizer . . . 87
Parallelization Workflow . . . 88
Routing Workflow . . . 88
Chain Workflow . . . 89
Agent as Tool . . . 89
Tool as Agent . . . 90
Orchestrator-Workers Workflow . . . 90

Agent Development . . . 91

Persona . . . 91
Knowledge . . . 91
Tools . . . 91
Reasoning . . . 91
Task . . . 91

Agent Skills . . . **92**
Progressive Disclosure . . . 92
Skills and Tools . . . 92
SkillsJars . . . 92
Configure the Application . . . 92
Create the Agent . . . 92
Publish the Agent as an API . . . 92
Designing Reliable Skill-Based Agents . . . 93

AI Application Infrastructure . . . 94

Observability . . . **95**
OpenTelemetry . . . 95
OpenTelemetry Collector . . . 96
Spring AI App . . . 97
Langfuse . . . 98
Test . . . 98
MLflow . . . 99

AG-UI . . . **101**
Create Agent . . . 101
AG-UI API . . . 101
Frontend . . . 101

Source Code and Materials . . . 102

Spring AI Basics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Getting Started

Let's start the journey with Spring AI from a simple AI application.

Prerequisites

Before writing Spring AI applications, we need to prepare the local development environment. Obviously, we need to have Java installed and configured. We also need to have a large language model (LLM) ready for development and testing.

Java

Spring AI requires a minimal Java version of 17. It's recommended to use Java 21 or Java 25 LTS, so we can leverage the power of virtual threads.



Source code of this book is tested using Java 25 with virtual threads enabled.

Spring AI

This book uses Spring AI `2.0.0`. Example applications in this book use Maven to manage dependencies.



Spring AI `2.0.0` requires Spring Boot 4.x and Spring 7.x. If your project still uses Spring Boot 3.x and Spring 6.x, you need to either upgrade Spring first, or stick to Spring AI 1.x.

To simplify dependency management of Spring AI related modules, the `spring-ai-bom` dependency can be imported to set versions of Spring AI dependencies. In the code below, set the property `spring-ai.version` to the version of Spring AI.

Figure 1. Spring AI dependency management using BOM

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.ai</groupId>
      <artifactId>spring-ai-bom</artifactId>
      <version>${spring-ai.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Language Model

A language model is required for development, testing and production deployments. This language model can run locally or on the cloud, as long as it provides an API endpoint to access its service. Spring AI provides built-in support for a large number of language model services.

- To run a model locally, there are many options available, including Ollama, vLLM, and LM Studio.
- To use a cloud-based model service, you need to open an account on the cloud service and get an API key.

Here let's start from using Ollama.

Ollama is a tool to run large language models locally. You can simply download Ollama and install it on your local machine. After installation, you can open a terminal window and use Ollama CLI command `ollama` to work with it.

There are many models available for use with Ollama, see Ollama's models page for a full list.

We can use `ollama pull` to pull a model. Here we are using Qwen3.

Figure 2. Use Ollama to pull a model

```
ollama pull qwen3:0.6b
```



The size of `qwen3:0.6b` is only 523MB. It's good for local development and testing.

After the model is pulled, it can be run using `ollama run`.

Figure 3. Use Ollama to run a model

```
ollama run qwen3:0.6b
```



`ollama run` command pulls non-existing models automatically.

`ollama run` starts a command-line session with the LLM. You can simply type any text to receive completions from LLM.

```
alexcheng@Fus-Mac-mini ~ % ollama run qwen3:0.6b
>>> hello
Thinking...
Okay, the user said "hello /think" and I need to respond. First, I should acknowledge their greeting.
Maybe say hello and ask how I can help. Keep it friendly and open-ended. Let me make sure the response is
welcoming and encourages them to ask more questions. Avoid any technical terms and keep the language
simple.
...done thinking.

Hello! How can I assist you today? 😊

>>> Send a message (/? for help)
```

Figure 4. ollama run

Ollama also provides a GUI window to send messages to a model.

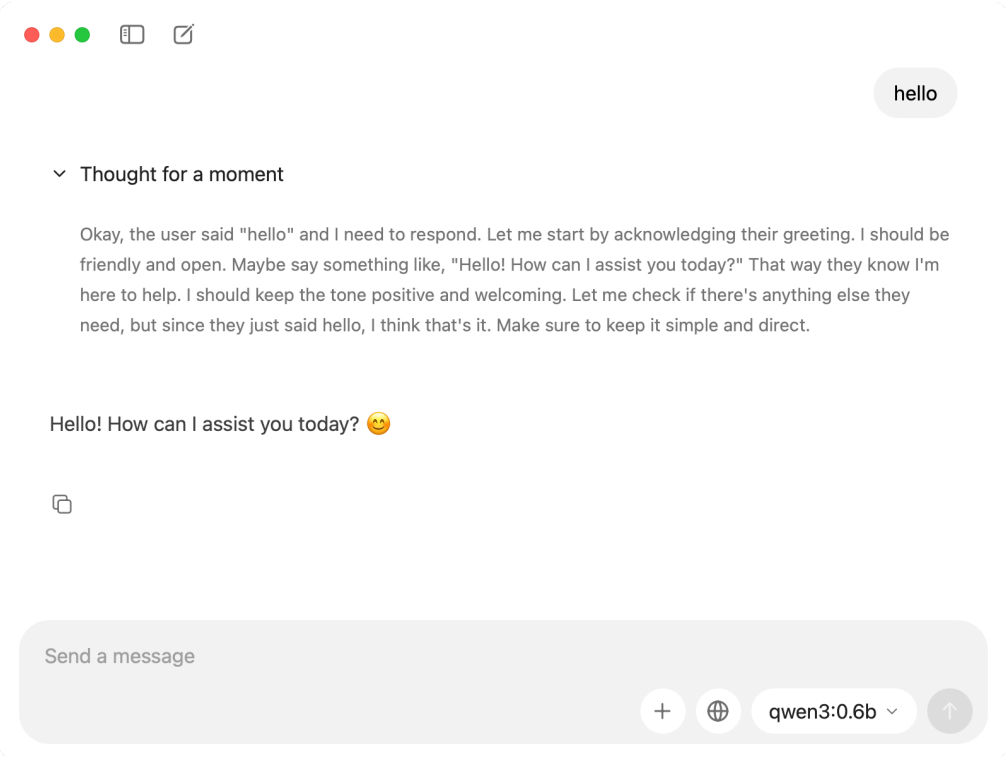

Figure 5. Ollama GUI

By default, Ollama provides its API endpoint at port `11434`.

Spring Boot Application

The easiest way to create a new Spring AI application is using Spring Initializr. When adding the project's dependencies, `Ollama` should be selected. This enables Spring AI to interact with Ollama. `Spring Web` is also added to create a simple REST API.

Below is the screenshot of Spring Initializr.

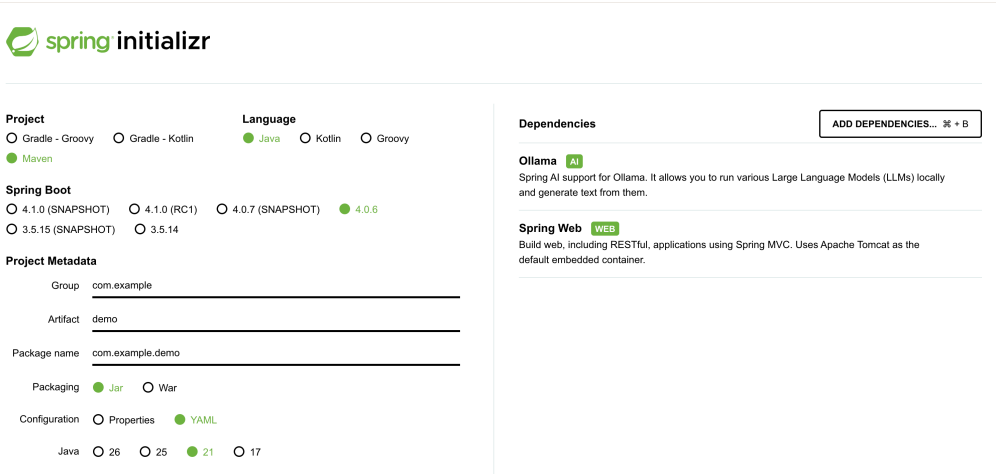

Figure 6. Spring Initializr UI

Now we can download the created application and open it using IntelliJ IDEA or other IDEs.

Adding the `Ollama` dependency actually includes the `spring-ai-starter-model-ollama` to the Maven project. This Spring Boot starter will create necessary beans to work with Spring AI.

Figure 7. Spring Boot Ollama starter

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-ollama</artifactId>
</dependency>
```

Here we need to add an `application.yaml` file to configure the Spring Boot application. This is because `qwen3` model should be used. By default, Ollama uses Mistral model. The property to configure the Ollama model is `spring.ai.ollama.chat.options.model`.

Figure 8. Spring Boot configuration for Spring AI using Ollama

```
spring:
  ai:
    ollama:
      chat:
        options:
          model: "qwen3:0.6b"
```

Now we add a REST endpoint to chat with an LLM. The `ChatClient.Builder` instance is injected into the REST controller to create `ChatClient` instances. This `ChatClient.Builder` instance is provided by Ollama Spring Boot starter. A `ChatClient` is created from this `ChatClient.Builder` using the `build` method. The code `chatClient.prompt().user(message).call().content()` sends a request to Ollama API endpoint and receives the output.

Figure 9. REST Controller

```
@RestController
public class ChatController {

  private final ChatClient chatClient;

  public ChatController(ChatClient.Builder builder,
      LoggingAdvisor loggingAdvisor) {
    this.chatClient = builder.defaultAdvisors(loggingAdvisor).build();
  }

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

Now we can start the Spring Boot application. Once the application is started, we can use any REST client tool to interact with the REST API.

Here we use SpringDoc to expose an OpenAPI endpoint and Swagger UI to test the API.

Figure 10. SpringDoc dependency

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>3.0.2</version>
</dependency>
```

We can open a browser window and navigate to `http://localhost:8080/swagger-ui/`, then use Swagger UI to test the API.

Below is the result of testing the API using Swagger UI.

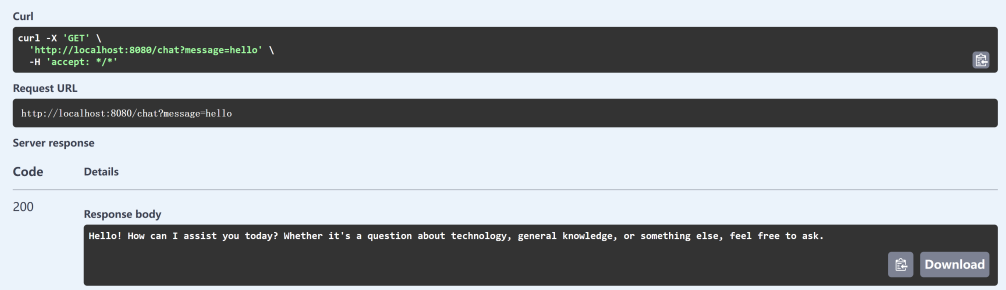

Figure 11. Use Swagger UI

Use Model Service

While Ollama is great for local development and testing, we usually use cloud-based model services for production. All major cloud platforms provide AI models as services, including Google, Amazon, and Microsoft. Spring AI supports major AI model services. Here OpenAI is used as an example.

For Spring Boot, the easiest way is adding the Spring Boot starter dependency. For OpenAI support, the dependency is `spring-ai-starter-model-openai`.

Figure 12. OpenAI Spring Boot starter dependency

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-openai</artifactId>
</dependency>
```

An OpenAI API key is required to use OpenAI services. In the configuration below, the OpenAI API key is read from environment variable `OPENAI_API_KEY`.

Figure 13. Set OpenAI API key

```
spring:
  ai:
    openai:
      apiKey: ${OPENAI_API_KEY}
```

Consolidate Local and Production Environment

If we use Ollama for local development and OpenAI for production, we need to have add both model dependencies in Spring AI. These two dependencies will conflict with each other. We should consolidate these two models. We can use only OpenAI model but different API endpoints in development and production.

Many model services provide an API which is compatible with OpenAI. Ollama also has this API. After Ollama is started, this API can be accessed from base URL `http://localhost:11434/v1/`.



OpenAI compatibility of Ollama is experimental and is subject to major adjustments including breaking changes. Only parts of OpenAI API are supported.

We can use Spring profiles to apply configurations for different environments. For the development profile, `spring.ai.openai.baseUrl` is configured to `http://localhost:11434/v1`. The API key is required for configuration, but its value will be ignored, so the value can be anything.

Figure 14. Use Ollama OpenAI compatible API in development profile

```
spring:
  ai:
    openai:
      baseUrl: http://localhost:11434/v1
      apiKey: ollama
```

In the production profile `application-prod.yaml`, `spring.ai.openai.baseUrl` is configured to `https://api.openai.com/v1`, which is the endpoint of OpenAI API.

Figure 15. Use OpenAI in production profile

```
spring:
  ai:
    openai:
      baseUrl: https://api.openai.com/v1
      apiKey: ${OPENAI_API_KEY}
```

Profiles can be switched using the option `-Dspring.profiles.active`, e.g. `-Dspring.profiles.active=prod` to use the `prod` profile.

Depends on whether you want to run models locally, there are two recommendations about setup of development environment.

Cloud-based Model Services

Cloud-based model services are actually cheap to use. One option is to simply use model services for both development and production. Spring AI provides integration modules for popular model service platforms. We only need to include the Spring AI module and configure it.

Let's use Anthropic Claude as an example. In a Spring Boot application, we can add the dependency of `spring-ai-starter-model-anthropic` module.

Figure 16. Anthropic module dependency

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-model-anthropic</artifactId>
</dependency>
```

Then we can configure Anthropic Claude. The prefix of configuration properties is `spring.ai.anthropic`. An API key is required to be configured as environment variable `ANTHROPIC_API_KEY`. The model `claude-opus-4-0` is used.

Figure 17. Configure Anthropic Claude

```
spring:
  ai:
    anthropic:
      apiKey: ${ANTHROPIC_API_KEY}
      chat:
        options:
          model: claude-opus-4-0
```

Use Container

If you want to run models locally, It's recommended to run models in a container. Container tools like Docker and Podman have already been used

extensively in development. You may already use containers to run databases, message brokers, and other tools. Running models in a container means that you don't need to install other tools.

llama.cpp

A popular choice is using llama.cpp to run models. `llama.cpp` provides an OpenAI compatible API to interact with the model. Model files can be downloaded from Hugging Face.

In the Docker compose file below, the model file of `Qwen3-0.6B` is downloaded from Hugging Face, then `llama.cpp` is started to serve this model.

Figure 18. Docker compose file to run models using llama.cpp

```yaml
services:
  model-runner:
    image: ghcr.io/ggml-org/llama.cpp:server
    volumes:
      - model-files:/models
    command:
      - "--host"
      - "0.0.0.0"
      - "--port"
      - "8080"
      - "-n"
      - "512"
      - "-m"
      - "/models/Qwen3-0.6B-Q8_0.gguf"
    ports:
      - "8180:8080"
    depends_on:
      model-downloader:
        condition: service_completed_successfully

  model-downloader:
    image: ghcr.io/alexcheng1982/model-downloader
    restart: "no"
    volumes:
      - model-files:/models
    command:
      - "hf"
      - "download"
      - "unsloth/Qwen3-0.6B-GGUF"
      - "Qwen3-0.6B-Q8_0.gguf"
      - "--local-dir"
      - "/models"

```

```
volumes:
  model-files:
```

After the container is started, the model API can be accessed from `http://localhost:8180`. In Spring AI, we can create a new profile which sets the configuration key `spring.ai.openai.baseUrl` to `http://localhost:8180`. The `apiKey` can be set to anything.

Figure 19. Use OpenAI compatible API running in the container

```
spring:
  ai:
    openai:
      baseUrl: http://localhost:8180
      apiKey: demo
```



llama.cpp provides a web UI to interact with the model. You can access this UI at `http://localhost:8180` using a browser.

Ollama

Ollama can also run in a container, which means we don't need to install Ollama on local machine.

In the Docker compose file below, Ollama is started in a container. Another container is used to pull the `qwen3:0.6b` model.

Figure 20. Docker compose file to run models using Ollama

```
services:
  ollama:
    image: ollama/ollama
    container_name: ollama
    ports:
      - "11434:11434"
    volumes:
      - ollama:/root/.ollama
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:11434"]
      interval: 30s
      timeout: 10s
```

```
      retries: 5
    command: ["/bin/ollama", "serve"]

  ollama-pull-qwen3:
    image: ollama/ollama
    container_name: ollama-pull-qwen3
    volumes:
      - ollama:/root/.ollama
    depends_on:
      ollama:
        condition: service_healthy
    command: ["/bin/ollama", "pull", "qwen3:0.6b"]

volumes:
  ollama:
    driver: local
```

Summary

The chapter introduces Spring AI basics by showing how to set up Java, manage Spring AI dependencies with the BOM, and run a local model using Ollama. It then walks through creating a Spring Boot app (via Spring Initializr) with an Ollama starter, configuring the model in application.yaml, and exposing a chat REST endpoint tested through Swagger UI. Finally, it explains production-oriented setup by using OpenAI-compatible APIs and Spring profiles to switch endpoints (local vs cloud), plus alternatives like Anthropic services and containerized local model runtimes (llama.cpp or Ollama).

Chat Completion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatModel

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Message

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatOptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatResponseMetadata

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RateLimit

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

PromptMetadata

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Chat Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatClient

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Build Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Call Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Handle Response

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatClient Builder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use Custom Chat Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Global Default Chat Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Request Level Chat Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prompt Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

PromptTemplate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Builder of PromptTemplate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Custom `TemplateRenderer`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Advisor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

`Advisor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

`CallAdvisor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

`StreamAdvisor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

`BaseAdvisor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Update `ChatClientRequest`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

`LoggingAdvisor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use `AdvisorSpec`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Enable `Advisors`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Advisors Order

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Recursive Advisor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Conversation and chat memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

From single requests to conversations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ChatMemory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Memory strategy and repository

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Window size and context budget

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MessageChatMemoryAdvisor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

A conversational REST API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Test a follow-up question

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Clear a conversation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Persistent memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Memory and complete chat history

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Failures and concurrent requests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Basic Tool Usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ToolCallback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ToolDefinition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MethodToolCallback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

FunctionToolCallback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ToolCallbackProvider

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Streaming Chat Completion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

StreamingChatModel

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Streaming Web Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Requests and session isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Completion, cancellation and cleanup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Browser client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Streaming JSON Events

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Structured Output Converter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

StructuredOutputConverter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ListOutputConverter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MapOutputConverter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

BeanOutputConverter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use ChatClient

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Multimodal Input

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Media

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Image Understanding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

OpenAI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Retrieval-Augmented Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RAG Introduction

In this part, we'll start the journey of building Retrieval-Augmented Generation (RAG) applications using Spring AI. RAG applications have already gained a lot of attentions from industry. It's a typical area where AI technology has practical usage.

Reduce Hallucinations

The main goal of using RAG is to reduce hallucinations when using language models for chat completions. Language models are trained using material from various sources. Once a language model is trained, its parametric knowledge is frozen. When the prompt to a language model contains information not seen in the training materials, the output will hallucinate.

For example, when asking a language model (Llama 3) the following question: *Who won the gold medal in men's 100 meters at 2020 Olympic Games?*. The language model can provide the correct result.

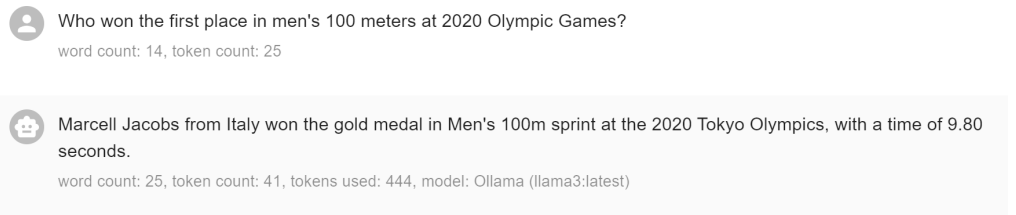

Figure 21. QA using model internal knowledge

If we change the question to *Who won the gold medal in men's 100 meters at 2024 Olympic Games?*, the language model cannot provide a meaningful answer.

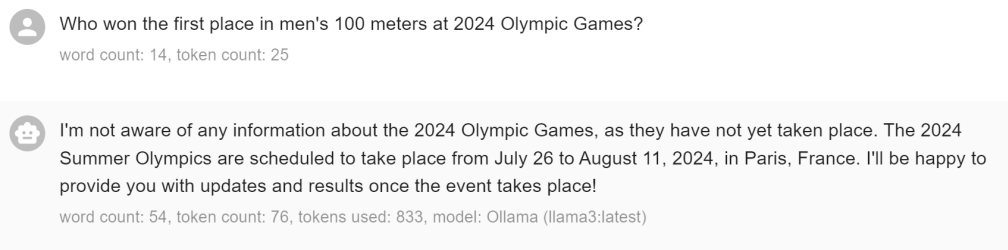

Figure 22. QA without model internal knowledge

There are three approaches to reduce hallucinations, model fine-tuning, tools, and RAG.

- **Model fine-tuning** works by fine-tuning the model to include extra materials.
- **Tools** allow a model to interact with external systems to provide necessary information.
- **RAG** works by augmenting the original prompts with retrieved content for a model to generate output.

The idea behind RAG is quite simple. A frozen language model itself lacks information to generate output for certain prompts. If we augment the prompt to include content from external sources, the model can leverage the provided content to generate the output. The included content is retrieved from an external system and must be semantically similar with the original prompt.

Let's go back to the example above. If we augment the original prompt with the content from a web page, the model can generate meaningful output.

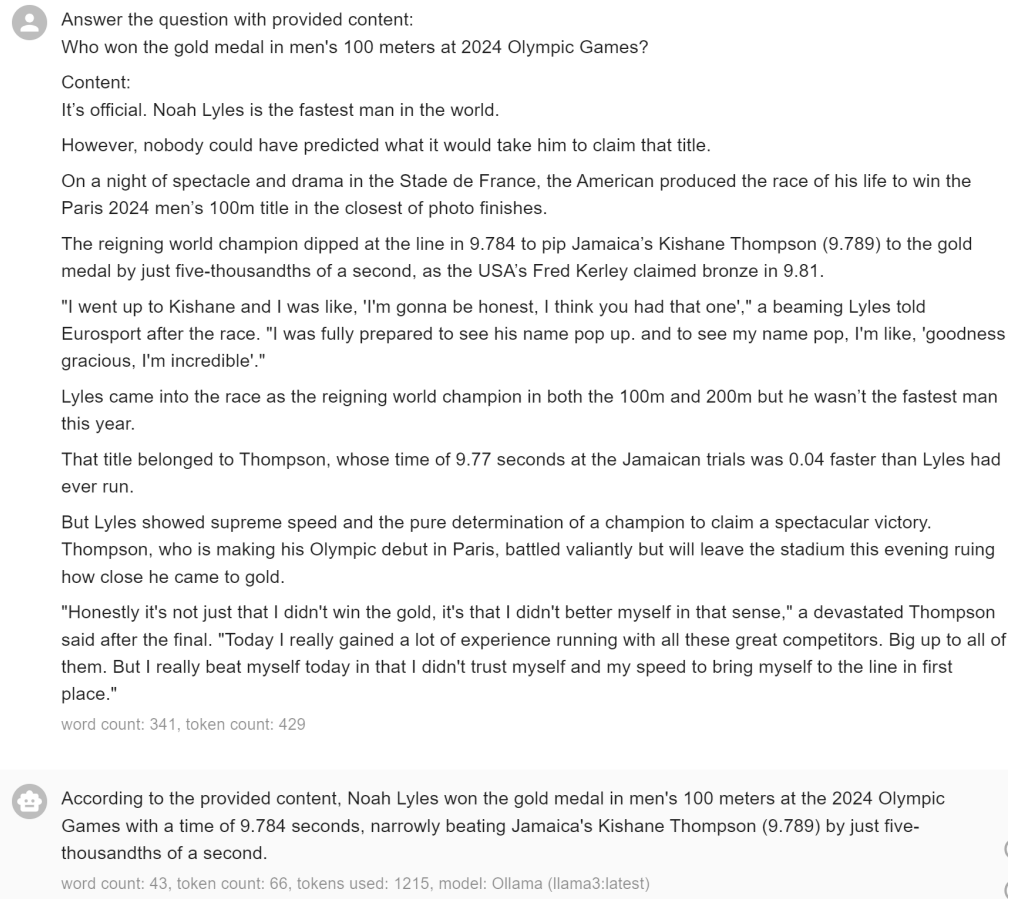

Figure 23. QA with provided content

Naive RAG

Naive RAG is a simple way to implement RAG. As the name suggests, naive RAG may not provide best results. However, naive RAG is a good starting point to leverage RAG.

The diagram below shows the architecture of naive RAG. Given a user prompt, similar documents related the prompt are retrieved from a vector database. Similar documents and the original prompt are combined together to form the final input to a language model to generate the output.

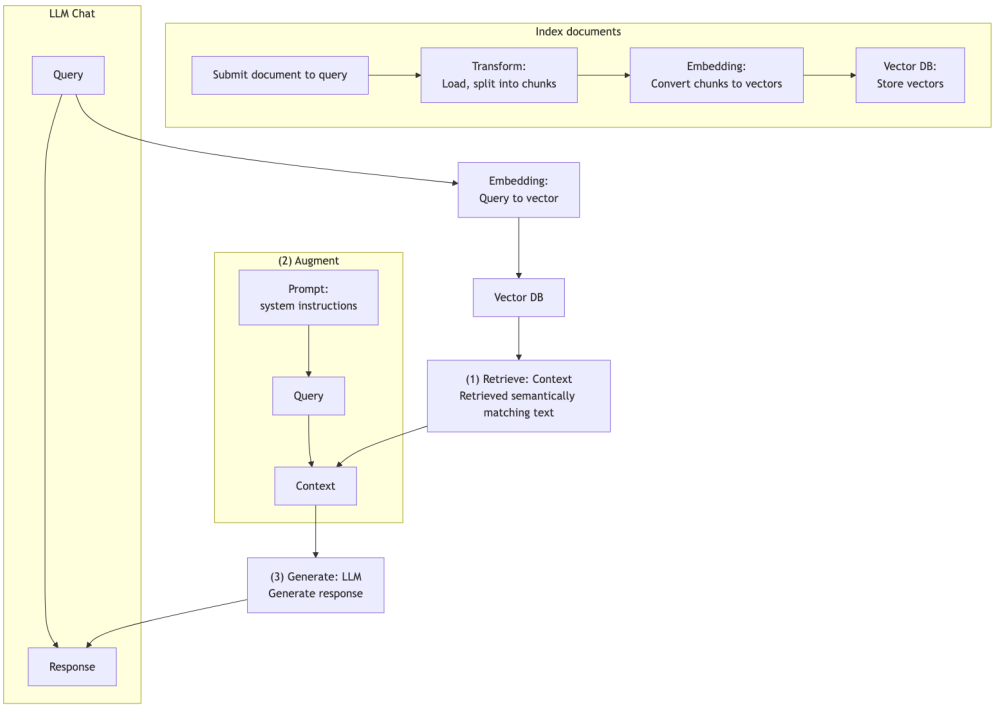

Figure 24. Naive RAG

Vector databases play an important role in RAG. Reference documents are stored in vector databases. Text content is converted to a vector (an array of floating point numbers) using a text embedding model. For a given document, its content and embedding vector are both stored in a vector database. For a prompt, its content is also converted to a vector using the same embedding model. Similarity between the vector of prompt and vectors of reference documents are calculated using vector similarity algorithms. By doing this, similarity between documents is converted to similarity between vectors.

After retrieving similar documents from the vector database, these documents are combined with the original prompt. The combination is usually done with a predefined template. In the template below, we instruct the language model to answer the question using provided content.

Figure 25. Prompt template

```
Answer the question using provided content:

Question: {question}

Provided content: {content}

Answer:
```

Embedding Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

EmbeddingModel

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

EmbeddingOptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

BatchingStrategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use EmbeddingModel

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Document Reader

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

TextReader

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

JsonReader

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Document Transformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Document Writer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Vector Store

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

VectorStore

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create `VectorStore`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Add Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Delete Documents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Similarity Search

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

VectorStore REST API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

SimpleVectorStore

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Save and Load

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Pgvector

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Installation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Spring AI Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Vector Store Cloud Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MongoDB Atlas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RAG

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Simple RAG

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Modular RAG

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Query

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Pre-Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

QueryTransformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RewriteQueryTransformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

TranslationQueryTransformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

CompressionQueryTransformer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

QueryExpander

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MultiQueryExpander

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

DocumentRetriever

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

VectorStoreDocumentRetriever

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

DocumentJoiner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ConcatenationDocumentJoiner

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Post-Retrieval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

DocumentSelector

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

DocumentRanker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

DocumentCompressor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Generation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

QueryAugmenter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ContextualQueryAugmenter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RetrievalAugmentationAdvisor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

RAG Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Text-to-SQL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Database Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use Advisor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Execute SQL Statements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

REST Controller

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

PDF Q&A

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prerequisites

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Load PDF

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Q&A

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Ollama

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

OpenAI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MCP

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MCP Introduction

MCP (Model Context Protocol) is currently a hot topic related to AI development. MCP provides a standard protocol to provide context data for AI models. MCP defines two roles: *server* and *client*. The server and the client interact with each other using a standard protocol. MCP server is responsible for providing knowledge, including prompt templates, resources, and tools. MCP client interacts with the server to obtain these knowledge, including creating prompts from templates, retrieving the content of resources, and invoking tools. A JSON-RPC based protocol is used between the server and the client. In this way, the provider of knowledge is decoupled from the consumer. The most important benefit of decoupling is to facilitate sharing and reuse.

For an AI application, only one MCP client needs to be integrated to use knowledge from different MCP servers. There is no restriction on how the MCP servers themselves are implemented. A Java-based AI application that integrates a Java MCP client can use knowledge from MCP servers written in JavaScript or Python.

The greatest value of MCP is that it facilitates the sharing of knowledge. A large number of reusable MCP servers can already be found publicly, see mcp.so. Due to the importance of MCP, the development of AI applications nowadays cannot be done without a good understanding of MCP.

Quick Start

The details of MCP protocol is out of scope of this book. The MCP protocol itself is not complicated. We just need to understand the methods that client and the server can use. Each method has predefined structures for parameters and return value.

Java Development Basics

MCP provides SDKs that can be used in different programming languages. These SDKs encapsulates the implementation details of the underlying protocol, which simplifies the development of servers and clients.

When developing an MCP server or client in Java, we can use the official SDKs. There are currently two SDKs available for Java and Kotlin.

When using MCP with Spring AI, the SDK version required by Spring AI should be used. This book uses the Java SDK of version `2.0.0`.

In order to add the MCP SDK dependencies to the project, we can import the `mcp-bom` dependency first. This removes the need to declare the version numbers of the different modules in MCP SDK.

Figure 26. mcp-bom dependency

```
<dependency>
  <groupId>io.modelcontextprotocol.sdk</groupId>
  <artifactId>mcp-bom</artifactId>
  <version>2.0.0</version>
  <type>pom</type>
  <scope>import</scope>
</dependency>
```

The MCP SDK contains several modules, and in most cases we will only need to use the `mcp` module. This module already contains the MCP server and client implementations, as well as the basic implementation of stdio and HTTP transports.

Figure 27. mcp dependency

```
<dependency>
  <groupId>io.modelcontextprotocol.sdk</groupId>
  <artifactId>mcp</artifactId>
</dependency>
```

The SDK supports both synchronous and asynchronous programming modes. There is no functional difference between synchronous and asynchronous programming modes. The only difference is that in asynchronous programming mode, objects such as `Flux` and `Mono` from the Reactor project are used as the return values of these methods. If you have used Spring WebFlux before, you should be familiar with the use of `Flux` and `Mono`. If you're not familiar with `Flux` and `Mono`, you'll have no problem using the synchronous programming model. In order to reduce complexity, the code in this book focuses on the synchronous programming model and does not require familiarity with `Flux` and `Mono`.

Sample MCP Server

There are four different implementations for each of the MCP servers and clients, taking into account the transport and programming modes. For example, a stdio server implemented in synchronous programming mode, or an HTTP client implemented in asynchronous programming mode. Synchronous or asynchronous mode is only an internal implementation detail and does not affect the external use.

In total, there are 4 different development scenarios. These are stdio and HTTP transport for MCP servers, and stdio and HTTP transport for MCP clients.

A simple MCP server will be created as an example. The server provides a simple tool that the client can call.

The functionality of this tool is very simple. When using the HTTP Basic authentication method, given a username and password, they are encoded using base64 format. The encode value is used as the value of the HTTP `Authorization` header.

For example, given the username `admin` and password `password`, value of the header is the base64 encoding of `admin:password`.

The goal of this tool is to calculate this value. The tool takes two parameters, `username` and `password`, which represent the username and password, respectively. The return value of the tool contains only one value, which represents the value of the `Authorization` header.

stdio Server

Let's start from the *stdio* transport of MCP server and client. In this approach, the server is packaged into a standalone JAR file. The client runs this JAR file to start the server and then interacts with the standard input and output streams of the server process. The client writes messages to the server's input stream and reads messages from the server's output stream. The basic process is similar for both the server and the client. Inside a loop, messages are first read from the input, then processed, and finally written to the output.

Create Server

The first step is to create a server that uses *stdio* transport. The MCP server is represented by the `McpServer` interface, which has two static methods, `sync` and `async`, to create a server in synchronous and asynchronous modes, respectively. The parameters of these methods are the `McpServerTransportProvider` interface, which indicates the transport mode of the server. *stdio* transport mode uses `StdioServerTransportProvider`.

The return value of the `sync` method of `McpServer` interface is `SyncSpecification`, which is a builder used to create the server. We should use methods in `SyncSpecification` to configure the server, then use the `build` method to finish building the server.

The following describes the methods to configure the server.

- The `serverInfo` method sets the basic information of the server, including the server name and version number.
- The `capabilities` method sets the capabilities of the server, represented by `ServerCapabilities`.

`ServerCapabilities` is a record type with attributes corresponding to the different capabilities of the server in the MCP protocol. Each capability has its

own corresponding record type. If a capability has sub-capabilities, the corresponding record type contains attributes to set the sub-capabilities. For example, `ToolCapabilities`, which represents the `tools` capability, contains the sub-capability `listChanged`, while `LoggingCapabilities`, which represents the `logging` capability, does not contain any attributes. `ServerCapabilities` provides a builder. It is a common practice to first create a builder using the `builder` method, then use the `build` method to get a `ServerCapabilities` object after the configuration is complete.

The example MCP server only declares support for the `tools` capability. The parameter `false` indicates that the sub-capability `listChanged` is not supported.

After setting the `ServerCapabilities`, we can use the `build` method to build the server. At this point, the server only supports the `tools` capability, and no tools have been added.

Figure 28. Create a MCP server using stdio transport

```
public class SimpleStdioServer {

  public void start() {
    var provider = new StdioServerTransportProvider();
    McpServer.sync(provider)
        .serverInfo("Sample", "1.0.0")
        .capabilities(ServerCapabilities.builder()
            .tools(false)
            .build())
        .build();
  }

  public static void main(String[] args) {
    new SimpleStdioServer().start();
  }
}
```

Add Tool

The next step is to add the previously mentioned tool that generates the HTTP basic authentication header. Tools can be added during the build phase or dynamically after the server has been started. Tools are also divided into synchronous and asynchronous types. Tools are referred to as `SyncToolSpecification` and `AsyncToolSpecification`, respectively. Synchronous

servers can only use synchronous tools, while asynchronous servers can only use asynchronous tools.

`SyncToolSpecification` is a record type that contains the attributes `tool` and `call`. `tool` represents the metadata of the tool. `call` represents the code implementation of the tool call.

The attribute `tool` is of type `Tool`. There are several attributes of `Tool`. Here we only discuss three attributes, `name`, `description` and `inputSchema`, which represent the name of the tool, the description and the JSON schema of the input parameters, respectively. The JSON schema of the input parameters can be represented either as an object or as a JSON string, which is automatically deserialized to an object.

The attribute `call` is of type `BiFunction<McpSyncServerExchange, CallToolRequest, CallToolResult>` and represents the code implementation of the tool with two parameters and a return value. The first parameter, `McpSyncServerExchange`, is used to interact with the client, where methods can call client methods.

A parameter of type `CallToolRequest` represents the request for the method call. It has the `arguments` attribute of `Map<String, Object>`, which contains the actual tool call parameters. The value of the input parameter is retrieved from this map.

The return value of a tool call is `CallToolResult`. `CallToolResult` has attributes, `content`, `isError`, and `structuredContent`.

- `content` represents a list of contents of the tool's output. `Content` is an interface that represents different types of content.
- `isError` indicates whether or not the result of the tool call was successful.
- `structuredContent` is an optional JSON object that represents the structured result of the tool call.

The following section describes how to implement the previously mentioned tool related to HTTP Basic authentication. The first part is the description of the tool. The tool's input parameters contain two values, `username` and `password`, and the `Map<String, Object>` object is used to describe the method's input parameters using JSON schema spec. When using a `Map<String, Object>` to describe the schema, the map content should match the JSON schema spec.

The implementation of the tool is shown below. In the definition of the tool, the JSON schema of the tool's input parameters has been defined. Values of input parameters can be retrieved directly from the parameter map based on their names. After obtaining the values of `username` and `password`, the value of the `Authorization` header is generated according to Basic authentication specification.

`SyncToolSpecification` and `Tool` both have builders to create objects. Once we have the definition and implementation of the tool, we can create the `SyncToolSpecification` object corresponding to the tool and use the `tools` method of `SyncSpecification` to add the tool.

Figure 29. Tool implementation

```
public static final SyncToolSpecification CALCULATE_HTTP_BASIC_AUTH_HEADER =
    SyncToolSpecification.builder()
        .tool(
            Tool.builder(
                    "calculateHttpBasicAuthHeader",
                    Map.of(
                        "type",
                        "object",
                        "properties",
                        Map.of(
                            "username", Map.of("type", "string"),
                            "password", Map.of("type", "string")),
                        "required",
                        List.of("username", "password"),
                        "additionalProperties",
                        false))
                .title("Calculate HTTP basic auth header")
                .description(
                    "Calculate value of HTTP Authorization header for basic a\
uthentication")
                .outputSchema(
                    Map.of(
                        "type",
                        "object",
                        "properties",
                        Map.of("value", Map.of("type", "string"))))
                .annotations(
                    ToolAnnotations.builder()
                        .readOnlyHint(true)
                        .destructiveHint(false)
                        .idempotentHint(true)
                        .openWorldHint(false)
                        .returnDirect(false)
                        .build())
```

```
                .build())
        .callHandler(
            (exchange, request) -> {
              var args = request.arguments();
              String username = (String) args.get("username");
              String password = (String) args.get("password");
              String header =
                  "Basic "
                      + Base64.getEncoder()
                          .encodeToString(
                              (username + ":" + password).getBytes(StandardCh\
arsets.UTF_8));
              var result = Map.of("value", header);
              return CallToolResult.builder()
                  .structuredContent(result)
                  .addTextContent(objectMapper.writeValueAsString(result))
                  .build();
            })
        .build();
```

In the case of an asynchronous tool implementation, the corresponding type is `AsyncToolSpecification`. `AsyncToolSpecification` is also a record type. This record type also contains two attributes, `tool` and `call`. `tool` is also of type `Tool`. `call` is of type `BiFunction<McpAsyncServerExchange, CallToolRequest, Mono<CallToolResult>>`, which differs from the `call` attribute of `SyncToolSpecification` in that the return value of the tool call is of type `Mono<CallToolResult>`, indicating that this is an asynchronous tool call.

Once the `build` method of `SyncSpecification` has been called to build the server, the server is already running. By encapsulating the method call that starts the server in the `main` method, we have a Java program that can run as a MCP server.

Build Runnable Server

To simplify starting of the server, we package the Java program into a single executable JAR file. The Maven assembly plugin, which comes with its own `jar-with-dependencies` descriptor, is used here. We need to specify the name of the Java class to start.

Figure 30. Use Maven assembly plugin to package MCP server

```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.7.1</version>
    <executions>
        <execution>
        <phase>package</phase>
        <goals>
            <goal>single</goal>
        </goals>
        </execution>
    </executions>
    <configuration>
        <archive>
        <manifest>
            <addClasspath>true</addClasspath>
            <mainClass>com.javaaidev.mcp.server.SimpleStdioServer</mainClass>
        </manifest>
        </archive>
        <descriptorRefs>
        <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
</plugin>
```

After packaging with the `mvn package` command, the corresponding JAR file is generated and can be run directly with the `java` command.

Test Server

The final step is to test the MCP server. A separate tool MCP Inspector is used here.

To run this tool, we need NodeJS support, and use the following command to run it.

Figure 31. Run MCP Inspector

```
npx @modelcontextprotocol/inspector node build/index.js
```

Once it is running, open a browser and access port `6274` on localhost to see the web interface.

During the test, we need to select the connection method of the MCP server, and since we are using stdio, we need to specify the way to start the server.

Here we use the `java` command, and provide relevant command line parameters, including `-jar` and the path to the generated JAR file.

Click the *Connect* button to connect to the server. After connecting to the server, switch to the *Tools* tab and click *List Tools* to list all the tools. We can see the tool used as an example. Select the tool and we will see the input boxes for the parameters username and password. Enter the values for these two parameters and click *Run Tool* to invoke the tool and see the results of the tool invocation.

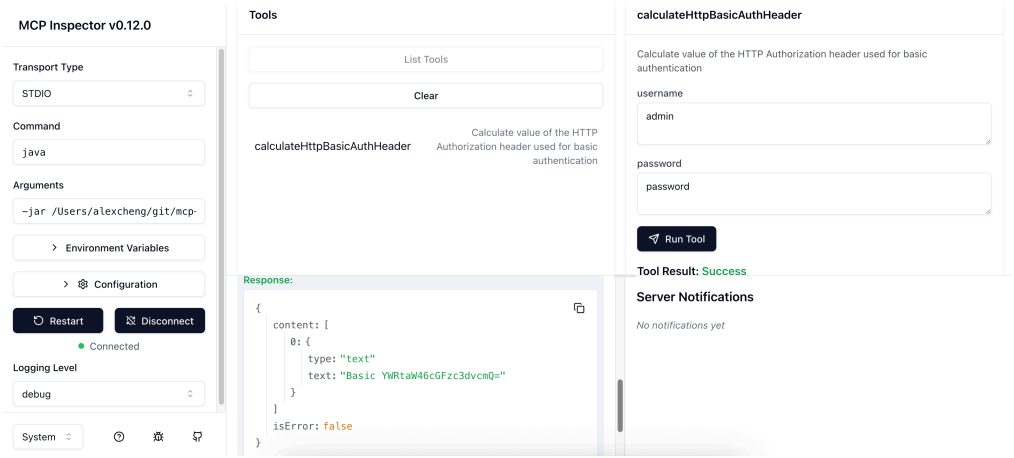

Figure 32. Test stdio server

This completes the development and testing of a MCP server using stdio.

HTTP SSE Server

After introduction of stdio transport, HTTP transport is described below. The HTTP transport has been changed between version `2025-03-26` and `2024-11-05`.The MCP Java SDK `2.0.0` supports the HTTP SSE transport in version `2024-11-05`. The choice of transport doesn't affect functionalities of MCP server. We can easily switch to HTTP transport from stdio transport.



HTTP SSE transport has been deprecated, streamable HTTP should be used.

Servers using the HTTP transport method are created in the same way as servers using stdio, only the transport method is different. The implementation class for the HTTP transport method is `HttpServletSseServerTransportProvider`, which is also an implementation of the `McpServerTransportProvider` interface. `HttpServletSseServerTransportProvider` provides a builder, and there are several configuration properties that can be set.

- `baseUrl`, the base url to access the HTTP server, defaults to an empty string.
- `messageEndpoint`, the path to the HTTP endpoint for receiving messages.
- `sseEndpoint`, the path to the SSE endpoint, default is `/sse`.
- `jsonMapper`, the `McpJsonMapper` used for JSON serialization, default value is created from `McpJsonDefaults.getMapper()`.
- `contextExtractor`, the `McpTransportContextExtractor` used for extracting metadata from the request.
- `keepAliveInterval`, the interval for keep-alive pings.
- `securityValidator`, the `ServerTransportSecurityValidator` for validating HTTP requests.

Of the above parameters, only `messageEndpoint` is required.

The following code creates a new `HttpServletSseServerTransportProvider` by specifying the value of `messageEndpoint`.

Figure 33. Create HttpServletSseServerTransportProvider

```
1  var provider = HttpServletSseServerTransportProvider.builder()
2      .messageEndpoint("/messages")
3      .build();
```

The `sync` method of `McpServer` is called with the provider as an argument, and the rest of the configuration is the same as the stdio transport. The complete code is shown below.

Figure 34. Create MCP server

```
var provider = HttpServletSseServerTransportProvider.builder()
    .messageEndpoint("/messages")
    .build();

McpServer.sync(provider)
    .serverInfo("Sample", "1.0.0")
    .capabilities(ServerCapabilities.builder()
        .logging()
        .tools(false)
        .build())
    .tools(Tools.CALCULATE_HTTP_BASIC_AUTH_HEADER)
    .build();
```

Unlike the stdio transport, the `HttpServletSseServerTransportProvider` cannot run independently because it is only a servlet implementation and requires runtime support from the servlet container. There are three different scenarios depending on how the MCP server is used.

The first is a minimalist deployment approach that does not rely on other web frameworks and only runs the MCP server's servlet. In this approach, an embedded servlet container such as Tomcat or Jetty needs to be started to run as a separate process.

The second is to use Spring WebMVC or WebFlux. The Java SDK for MCP provides integration modules with Spring that can be used directly. If you need to extend MCP functionality for an existing Spring web project, It's recommended to use this approach to provide a new MCP access endpoint on top of the existing HTTP service.

The third is to deploy to a shared web server , such as IBM WebSphere or RedHat JBoss. The MCP server is typically packaged as a WAR file to deploy.

Here I introduce the first way which uses an embedded Jetty to start the MCP server. The code is shown below. The overall logic is relatively simple. It creates a Jetty server, adds the `HttpServletSseServerTransportProvider` servlet, and finally starts Jetty server.

Figure 35. Use Jetty

```
var server = new Server(9000);
var context = new ServletContextHandler();
context.setContextPath("/");
context.addServlet(provider, "/*");
server.setHandler(context);
server.start();
server.join();
```

The implementation of this server is also packaged into a JAR file to run independently.

The tests also use MCP Interceptor. When connecting to the server, select the HTTP transport method and specify the hostname, port and context path of the HTTP server. Once connected, we can test the server in the same way as the stdio transport.

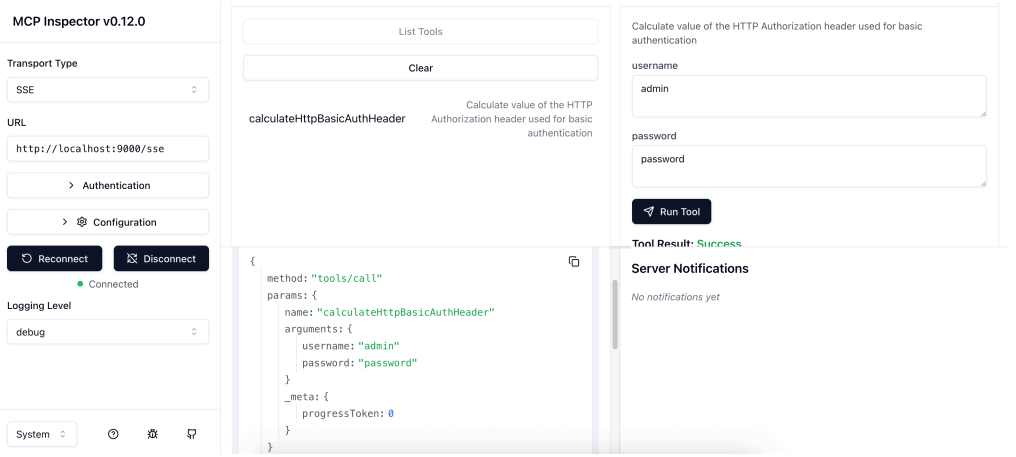

Figure 36. Test HTTP server

MCP Client

Like servers, MCP clients can use both stdio and HTTP transports. Clients are created in a similar way to servers.

The static methods `sync` and `async` in the interface `McpClient` create synchronous and asynchronous clients, respectively. The arguments to these

methods are the `McpClientTransport` interface, which represents the transport used by the client. The `sync` method returns a `SyncSpec`, which is a builder that provides methods to set different properties, and finally calls the `build` method to complete the creation.

The stdio transport uses `StdioClientTransport`.

`StdioClientTransport` needs to provide `ServerParameters` when it is created.`ServerParameters` specifies the way to start the MCP server. We need to provide the command to start the server, the parameter values, and the values of the environment variables. `ServerParameters` provides a builder to create objects.

The `ServerParameters` value is used to start an external process to the server, using `ProcessBuilder` from the Java Standard Library.

As an example client, we can use the `java -jar` command to start the MCP server described earlier. In the following code, `jarFilePath` is the path to the JAR file.

Figure 37. Create StdioClientTransport

```
var transport = new StdioClientTransport(
    ServerParameters.builder("java")
        .args("-jar", jarFilePath)
        .build());
```

After completing the creation of the client, we can connect to the server and invoke the tool according to the client-server interaction process. This process is divided into 3 steps.

- The first step is initialization, which is done by calling the client's `initialize` method. From the result of the method call, we can view information about the server and the capabilities it provides.
- The second step is the normal interaction when the client calls different methods. For example, the `callTool` method calls a tool. The parameters of the call are the name of the method and the values of the input parameters. The return value is the result of the tool call.
- The last step is to close the client. The client implements Java's `AutoClosable` interface, which can be placed in a try-with-resource to enable automatic closing of the client. It can also be closed manually by calling the `close` method. In addition to the `close` method, there is also a

`closeGracefully` method to gracefully close the client. `closeGracefully` waits for the operation currently in progress to finish processing.

The following is the complete sample code for the MCP client. It first creates a `StdioClientTransport` and starts the MCP server with the `java` command. The `sync` method of `McpClient` is used to create a synchronous client. The `initialize` method is called to complete the initialization. After initialization, the `callTool` method is used to call the tools provided by the server. Finally the client is automatically closed by using a try-with-resource.

Figure 38. Use stdio client

```
public class SimpleStdioClient {

  private static final Logger LOGGER =
      LoggerFactory.getLogger(SimpleStdioClient.class);

  public void connect(String jarFilePath) {
    var transport = new StdioClientTransport(
        ServerParameters.builder("java").args("-jar", jarFilePath).build(),
        new JacksonMcpJsonMapperSupplier().get());
    try (var client = McpClient.sync(transport).build()) {
      var initializeResult = client.initialize();
      LOGGER.info("Client initialized: {}", initializeResult);
      var callToolResult = client
          .callTool(CallToolRequest.builder("calculateHttpBasicAuthHeader")
              .arguments(Map.of("username", "admin", "password", "password"))
              .build());
      LOGGER.info("Tool call callToolResult: {}", callToolResult);
    }
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("JAR file is required");
      return;
    }
    var client = new SimpleStdioClient();
    client.connect(args[0]);
  }
}
```

The HTTP client is used in much the same way as stdio. The only difference is that when calling the `sync` or `async` method of `McpClient`, the

argument provided is a transport of type `HttpClientSseClientTransport`. `HttpClientSseClientTransport` provides a builder to create the object. When creating a `HttpClientSseClientTransport`, the `baseUri` is required. `baseUri` represents the address of the HTTP server to which the connection is made. Once the client is created, the interaction with the server is the same as the stdio transport described previously.

The complete code for using an HTTP client is shown below. The rest of the code is identical to the stdio client except for the creation of the transport.

Figure 39. Use HTTP client

```
public class SimpleHttpClient {

  private static final Logger LOGGER =
      LoggerFactory.getLogger(SimpleHttpClient.class);

  public void connect(String baseUri) {
    var transport = HttpClientSseClientTransport.builder(baseUri).build();
    try (var client = McpClient.sync(transport).build()) {
      var initializeResult = client.initialize();
      LOGGER.info("Client initialized: {}", initializeResult);
      var callToolResult = client
          .callTool(CallToolRequest.builder("calculateHttpBasicAuthHeader")
              .arguments(Map.of("username", "admin", "password", "password"))
              .build());
      LOGGER.info("Tool call callToolResult: {}", callToolResult);
    }
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Base url is required");
      return;
    }
    var client = new SimpleHttpClient();
    client.connect(args[0]);
  }
}
```

Spring Integration

If you are using the Spring framework to develop an MCP server or client, you can use the integration modules provided by Spring to further simplify

development.

There are two Spring modules for the HTTP transport, including Spring WebMVC and Spring WebFlux. These two modules can be used directly if Spring is already used in the project.

These two modules use `org.springframework.ai` as the group id, see Maven dependency below.

Figure 40. Spring integration module

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>mcp-spring-webmvc</artifactId>
</dependency>
```

Spring WebMVC

The Spring WebMVC implementation is included in the module `mcp-spring-webmvc`, which contains the HTTP server transport `WebMvcSseServerTransportProvider`. The `WebMvcSseServerTransportProvider` is created using the builder, which provides the the configuration properties the same as for the servlet transport, namely `jsonMapper`, `baseUrl`, `messageEndpoint`, `sseEndpoint`, `keepAliveInterval`, `contextExtractor`, and `securityValidator`.

After creating the `WebMvcSseServerTransportProvider`, we can call the `getRouterFunction` method to get a `RouterFunction` object. This `RouterFunction` is added as a Spring bean to enable the MCP HTTP server.

Figure 41. Configuration of Spring WebMVC integration

```
@Configuration
public class AppConfiguration {

  @Bean
  public RouterFunction<ServerResponse> mcpServerRouterFunction() {
    var provider = WebMvcSseServerTransportProvider.builder()
        .messageEndpoint("/message").build();
    McpServer.sync(provider)
        .serverInfo("spring-webmvc-server", "1.0.0")
        .capabilities(ServerCapabilities.builder().tools(false).build())
        .tools(Tools.CALCULATE_HTTP_BASIC_AUTH_HEADER)
        .build();
```

```
    return provider.getRouterFunction();
  }
}
```

Spring WebFlux

The Spring WebFlux implementation is included in the module `mcp-spring-webflux`, which contains both the server and client implementations of the HTTP transport. The server implementation of the HTTP transport is the `WebFluxSseServerTransportProvider`, which can be created using the builder and provides the same configuration properties as the `WebMvcSseServerTransportProvider`.

`WebFluxSseServerTransportProvider` also has a `getRouterFunction` method to get the `RouterFunction` object, which needs to be added as a Spring bean.

Figure 42. Configuration of Spring WebFlux integration

```
@Configuration
public class AppConfiguration {

  @Bean
  public RouterFunction<?> mcpServerRouterFunction() {
    var provider =
        WebFluxSseServerTransportProvider.builder()
            .messageEndpoint("/message")
            .build();
    McpServer.sync(provider)
        .serverInfo("spring-webflux-server", "1.0.0")
        .capabilities(ServerCapabilities.builder().tools(false).build())
        .tools(Tools.CALCULATE_HTTP_BASIC_AUTH_HEADER)
        .build();
    return provider.getRouterFunction();
  }
}
```

Client

The implementation of the client-side HTTP transport, `WebFluxSseClient-Transport`, is also created using the builder. The builder has the following configuration properties.

- `WebClientBuilder` of type `WebClient.Builder`, which is used to create the WebClient that sends out the HTTP requests.
- `McpJsonMapper` with a default value retrieved from `McpJsonDefaults.getMapper()`.
- `sseEndpoint` with a default value of `/sse`.

By using the implementation of the client-side HTTP transport provided by Spring, we can connect to a server that also uses HTTP. The complete code is shown below. The implementation is similar to the previously described HTTP client.

Figure 43. Use Spring WebFlux HTTP client

```
public class SimpleHttpClient {

  private static final Logger LOGGER =
      LoggerFactory.getLogger(SimpleHttpClient.class);

  public void connect(String baseUri) {
    var transport = HttpClientSseClientTransport.builder(baseUri).build();
    try (var client = McpClient.sync(transport).build()) {
      var initializeResult = client.initialize();
      LOGGER.info("Client initialized: {}", initializeResult);
      var callToolResult = client
          .callTool(CallToolRequest.builder("calculateHttpBasicAuthHeader")
              .arguments(Map.of("username", "admin", "password", "password"))
              .build());
      LOGGER.info("Tool call callToolResult: {}", callToolResult);
    }
  }

  public static void main(String[] args) {
    if (args.length < 1) {
      System.out.println("Base url is required");
      return;
    }
    var client = new SimpleHttpClient();
    client.connect(args[0]);
  }
}
```

Spring AI Integration

Spring AI provides deeper integration with MCP and can leverage the auto-configuration capabilities of Spring Boot, further simplifying the development of MCP servers and clients.

Server

To quickly create an MCP server, we can use Spring Boot and add the Spring Boot starter `spring-ai-starter-mcp-server`. We need to add the relevant dependencies based on the transport. For the HTTP transport, we should add the Spring Boot web starter and the Spring integration module in the MCP SDK.

Below is the Maven dependencies used for Spring AI MCP server with WebMVC.

Figure 44. Maven dependencies

```
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>spring-ai-starter-mcp-server</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.ai</groupId>
  <artifactId>mcp-spring-webmvc</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

The advantage of using Spring Boot is that we can define the behavior of the MCP server through configuration without writing code.

The prefix for MCP server-related configuration properties is `spring.ai.mcp.server`. Supported configuration properties are listed as below.

- `enabled`: Whether to enable the server. The default value is `true`.
- `stdio`: Whether to enable the stdio transport. The default value is `false`.
- `name`: The name of the MCP server. The default value is `mcp-server`.
- `version`: The version of the MCP server. The default value is `1.0.0`.

- `instructions`: Additional instructions provided by the MCP server to the client. The default value is `null`.
- `baseUrl`: The context path for the HTTP transport. The default value is an empty string.
- `sseEndpoint`: The context path for SSE. The default value is `/sse`.
- `sseMessageEndpoint`: The context path for SSE message handling. This is valid only for WebMVC and WebFlux. The default value is `/mcp/message`.
- `type`: The server type. The possible values are `SYNC` and `ASYNC`, indicating asynchronous and synchronous modes, respectively. The default value is `SYNC`.
- `promptChangeNotification`: Whether to enable notifications when the list of prompt templates changes. The default value is `true`.
- `resourceChangeNotification`: Whether to enable notifications when the list of resources changes. The default value is `true`.
- `toolChangeNotification`: Whether to enable notifications when the list of tools changes. The default value is `true`.
- `toolResponseMimeType`: A `Map<String, String>` type that specifies the MIME type for each tool's return value. The map key is the tool name, and the corresponding value is the MIME type.

The prompt templates, resources, and tools included in the MCP server are created as Spring beans. We can simply define beans of type `SyncPromptSpecification`, `SyncResourceSpecification`, and `SyncToolSpecification` in Spring configuration. These entities will be automatically added to the MCP server. If the server type is `ASYNC`, the added beans will be of the corresponding asynchronous type.

To customize the capabilities of the MCP server, we can add a bean of type `ServerCapabilities.Builder`. Spring Boot automatically configures a `ServerCapabilities.Builder` object, which determines whether to enable corresponding capabilities based on whether it contains beans of certain types. For example, if a bean of type `SyncToolSpecification` or `AsyncToolSpecification` is found, the `tools` capability is enabled. If you dynamically add prompt templates, resources, or tools to the server, you typically need to explicitly enable the corresponding capabilities by adding a bean of type `ServerCapabilities.Builder`.

Spring provides two MCP HTTP servers: WebMVC and WebFlux.

- To enable WebMVC, add the `mcp-spring-webmvc` and `spring-boot-starter-web` modules as dependencies. Set the value of `spring.ai.mcp.server.transport` property to `WEBMVC`.
- To enable WebFlux, add the `mcp-spring-webflux` and `spring-boot-starter-webflux` modules as dependencies. Set the value of `spring.ai.mcp.server.transport` property to `WEBFLUX`.

Below is a sample configuration of MCP server.

Figure 45. Server configuration

```
spring:
  ai:
    mcp:
      server:
        name: sample-server
        version: 1.0.0
        transport: WEBMVC
```

Client

To integrate the MCP client, add the `spring-ai-starter-mcp-client` Spring Boot starter to Spring Boot project.

We can configure the created MCP client. Configuration properties are prefixed with `spring.ai.mcp.client`. Supported configuration properties are listed as below.

- `enabled`: Whether the client is enabled. The default value is `true`.
- `name`: The name of the MCP client. The default value is `spring-ai-mcp-client`.
- `version`: The version of the MCP client. The default value is `1.0.0`.
- `initialized`: Whether to initialize the client. The default value is `true`.
- `requestTimeout`: The timeout for client requests. The default value is `20` seconds.
- `type`: The client type. Optional values are `SYNC` and `ASYNC`. The default value is `SYNC`.
- `rootChangeNotification`: Whether to enable notifications when the root path list changes. The default value is `true`.

- `toolcallback.enabled`: Whether to automatically integrate the tools provided by the MCP server. The default value is `false`.

The prefix of configuration properties for connecting to the stdio server is `spring.ai.mcp.client.stdio`.

The MCP client can connect to multiple stdio servers. The server configuration is contained in `connections`, which is of type `Map<String, Parameters>`. `Parameters` represents the parameters for starting the MCP server. The configuration properties are listed as below.

- `command`: The command to start the server.
- `args`: The command line arguments used to start the server.
- `env`: The environment variables used when starting the server. The type is `Map<String, String>`.

The prefix of configuration properties for connecting to an HTTP server is `spring.ai.mcp.client.sse`.

We can also connect to multiple HTTP SSE servers. Server configuration is contained in `connections`, which is a `Map<String, SseParameters>` object. `SseParameters` represents the parameters for connecting to the HTTP SSE server. The only configuration property is `url`, which represents the server URL.

Below is a sample configuration of MCP client.

Figure 46. MCP client configuration

```yaml
spring:
  ai:
    mcp.client:
      type: SYNC
      toolcallback:
        enabled: true
      sse:
        connections:
          defaultServer:
            url: http://localhost:9100
```

After the MCP client connects to different MCP servers, tools found from the MCP server can be automatically registered with Spring AI. When using Spring AI's `ChatClient` to send requests to a large language model, we can specify the

name of the tool found from the MCP server. When Spring AI calls a tool, the tool call is sent from the MCP client to the MCP server.

There are two points to note here. First, Spring AI's integration with the tools provided by the MCP server is disabled by default. To do this, we need to set the configuration property `toolcallback.enabled` to `true`. Second, Spring AI adds a prefix to the tool names found from the MCP server to prevent name conflicts. This prefix is the name of the MCP client. When sending requests using `ChatClient`, you must provide the prefixed tool name; otherwise, you will receive a tool not found error.

MCP Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Shared Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Role

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Content

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

TextContent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ImageContent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

AudioContent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ResourceLink

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

EmbeddedResource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

ServerExchange

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prompt Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

GetPromptResult

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

AsyncPromptSpecification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Add Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use Prompts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Resource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Resource Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Adding Resources and Resource Templates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Client Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Usage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Completions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Pagination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MCP Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Roots

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Sampling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MCP Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Server Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MetaProvider

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpTool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpResource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpPrompt

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpComplete

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Client Annotations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpSampling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpElicitation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpLogging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

McpProgress

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

List Changed Handlers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Special Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Metadata

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

MCP Examples

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

File System MCP Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Resources

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Completions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Directory boundary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Database MCP Server

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Resource

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prompt Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Completion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Spring AI Support

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Test

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agent Introduction

Agent is a hot topic in AI. Many tools, libraries, services, platforms claim that they can build AI agents. Agent is a broad concept. An agent can be a chatbot to provide cooking suggestions. It can also provide comprehensive results of a research topic.

It's hard to give *agent* a clear definition in the context of AI. The Merriam-Webster Dictionary has several definitions of agent. Below are definitions related to *agent* in AI.

- one that acts or exerts power
- something that produces or is capable of producing an effect
- a means or instrument by which a guiding intelligence achieves a result
- a computer application designed to automate certain tasks (such as gathering information online)

There are different ways to create agents. Many agent platforms don't require coding skills to build agents. In this book, I'll focus on building agents using Java.

An agent have five components.

- *Profile*, description of an agent. It may include elements such as background and demographics.
- *Tools*, tools used to complete tasks or acquire information.
- *Knowledge and memory*, annotates context with most relevant information.
- *Reasoning and evaluation*, enables self-reflection and internal reasoning for the completion of a task.
- *Planning and feedback*, organizes tasks to achieve high-level goals.

Not all agents have all these five components mentioned above. Having more components will make an agent more powerful, but harder to implement. When building an agent, the components to add depend on the intended usage scenarios. For the agent to provide cooking suggestions, adding a

profile is enough. For the agent to do a research, it may require all those five components.

Let's start from building the agent for cooking suggestions.

Cooking Suggestion Agent

This agent provides cooking suggestions and it only use the profile component. The profile of an agent is set in the system prompt when interacting with the LLM.

This agent is a Spring AI application. The code below shows the REST controller. The key point is the system text, which describes the persona of this agent.

Figure 47. Cooking agent REST controller

```
@RestController
@RequestMapping("/cooking")
public class CookingAgentController {

  private static final String SYSTEM_TEXT = """
      You are a chef who is proficient in various cuisines.
      Please answer users' questions about cooking.
      For other unrelated inputs, simply tell the user that you don't know.
      """;

  private final ChatClient chatClient;

  public CookingAgentController(ChatClient.Builder builder) {
    chatClient = builder.build();
  }

  @PostMapping
  public AgentResponse chat(@RequestBody AgentRequest request) {
    return new AgentResponse(
        chatClient.prompt().system(SYSTEM_TEXT)
            .user(request.input())
            .call().content());
  }
}
```

After starting the agent, we can test the agent using Swagger UI. The screenshot below shows the response from the agent. The input is *cook fish*. The agent provides suggestions about how to cook fish.

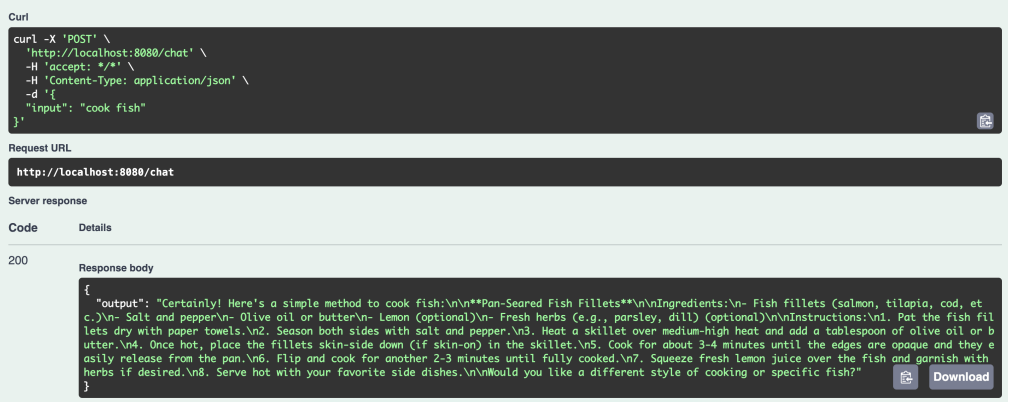

Figure 48. Test agent using Swagger UI

In the system text, we instruct the agent to ignore unrelated inputs. This is a common practice to avoid abuse of an agent. The screenshot below shows the response from the agent with a bad input. The input is *write quick sort algorithm in java*, which is unrelated to cooking. So the agent refuses to give output.

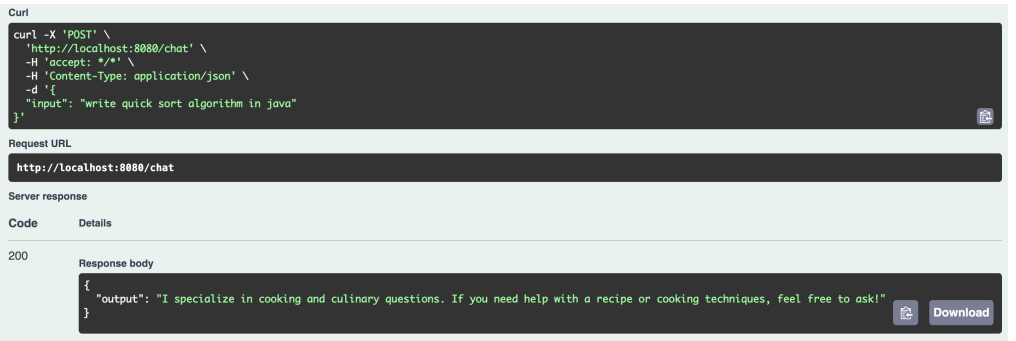

Figure 49. Bad input for the agent

Agent Components

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Profile

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Persona

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Knowledge and memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Reasoning and evaluation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Planning and feedback

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agentic Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Task Execution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Scope of Objective

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

LLM Options

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Prompt Template

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Evaluator-Optimizer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Use Different Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Max Number of Evaluations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Evaluation Results

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Generate Initial Result

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Evaluate

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Optimize

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Parallelization Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Result Types of Subtasks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Assembling Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Rate Limits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Routing Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Route Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Chain Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agent as Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tool as Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Orchestrator-Workers Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Implementation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agent Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Persona

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Knowledge

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Food Nutrition Facts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Reasoning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Agent Skills

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Progressive Disclosure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Skills and Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

SkillsJars

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Configure the Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create the Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Publish the Agent as an API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Designing Reliable Skill-Based Agents

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

AI Application Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Observability

Spring AI Observability provides built-in instrumentation that helps developers understand how their AI applications behave in production. Built on top of Micrometer Observability and integrated with OpenTelemetry, it automatically captures metrics, traces, and structured attributes for common AI operations such as model calls, tool execution, and agent workflows.

With observability enabled, Spring AI can emit traces that describe key aspects of an AI interaction, including the model being used, prompts and LLM responses, tool invocation details, latency, and token usage. These traces can then be exported through the OpenTelemetry ecosystem and visualized in observability platforms such as Grafana Tempo, Jaeger, or Langfuse.

This capability is particularly valuable for modern AI applications that involve multiple components, including LLM calls, retrieval pipelines, and external tools. Because it provides end-to-end visibility into how a request flows through the system. By correlating traces, metrics, and logs, Spring AI observability helps developers diagnose failures, analyze model performance, monitor token consumption, and better understand the behavior of AI agents in real-world environments.

Here I provide an example of enabling observability for Spring AI with OpenTelemetry and Langfuse.



Complete source code of this chapter can be found on GitHub.

OpenTelemetry

To enable OpenTelemetry, the following dependencies should be added to the Spring Boot project.

Figure 50. OpenTelemetry dependencies

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-opentelemetry</artifactId>
</dependency>
```

After adding the dependencies, Spring Boot creates necessary beans to work with OpenTelemetry.

OpenTelemetry Collector

In the Spring AI application, we don't send data directly to any observation platforms. Instead, the data is sent to OpenTelemetry collector. The collector can transform data and send to different upstream platforms.

OpenTelemetry collector is also started using Docker Compose.

Below is the configuration file of OpenTelemetry collector. It defines two exporters.

- `otlp_http/langfuse`: This exporter sends data to the OpenTelemetry API endpoint provided by Langfuse. This endpoint is available at path `/api/public/otel`. The `Authorization` header is required with the value of Langfuse API key.
- `prometheus`: This exporter exports data in the Prometheus format, which allows it to be scraped by a Prometheus server.

Figure 51. Configuration of OpenTelemetry collector

```
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: "0.0.0.0:4317"
      http:
        endpoint: "0.0.0.0:4318"
processors:
  batch:
exporters:
  otlp_http/langfuse:
    endpoint: "http://langfuse-web:3000/api/public/otel"
    headers:
      Authorization: "Basic ${LANGFUSE_AUTH_STRING}"
    tls:
      insecure: true
  prometheus:
    endpoint: "otel-collector:10010"
service:
  pipelines:
    metrics:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ prometheus ]
    traces:
      receivers: [ otlp ]
      processors: [ batch ]
      exporters: [ otlp_http/langfuse ]
```

Spring AI App

In the configuration below, metrics and tracing data is sent to `localhost:4318`, which is the port of OpenTelemetry collector. `management.tracing.sampling.probability` is set to `1.0`, so all tracing data will be collected and sent to OpenTelemetry collector.

Figure 52. Spring Boot configuration for observation

```yaml
management:
  otlp:
    metrics:
      export:
        enabled: true
        url: http://localhost:4318/v1/metrics
    traces:
      export:
        enabled: true
        url: http://localhost:4318/v1/traces
  observations:
    annotations:
      enabled: true
  tracing:
    enabled: true
    sampling:
      probability: 1.0
otel:
  logs:
    exporter: none
```

Langfuse

Langfuse is an open-source observability platform designed for applications built with large language models (LLMs). It allows developers to monitor and analyze AI interactions by capturing structured traces that include prompts, responses, tool calls, latency, and token usage.

Langfuse integrates with standards such as OpenTelemetry, enabling applications to export telemetry data and visualize complete AI workflows in a dedicated dashboard. This helps developers debug model behavior, track performance in production, and continuously improve the quality of AI-powered applications.

Langfuse is started using Docker Compose. See the docker-compose.yaml for the complete Docker Compose file to start Langfuse and its required dependencies.

Test

After starting the Spring AI app, we can interact with it by send some messages. Open Langfuse UI to view traces. Below is the screenshot of Langfuse UI.

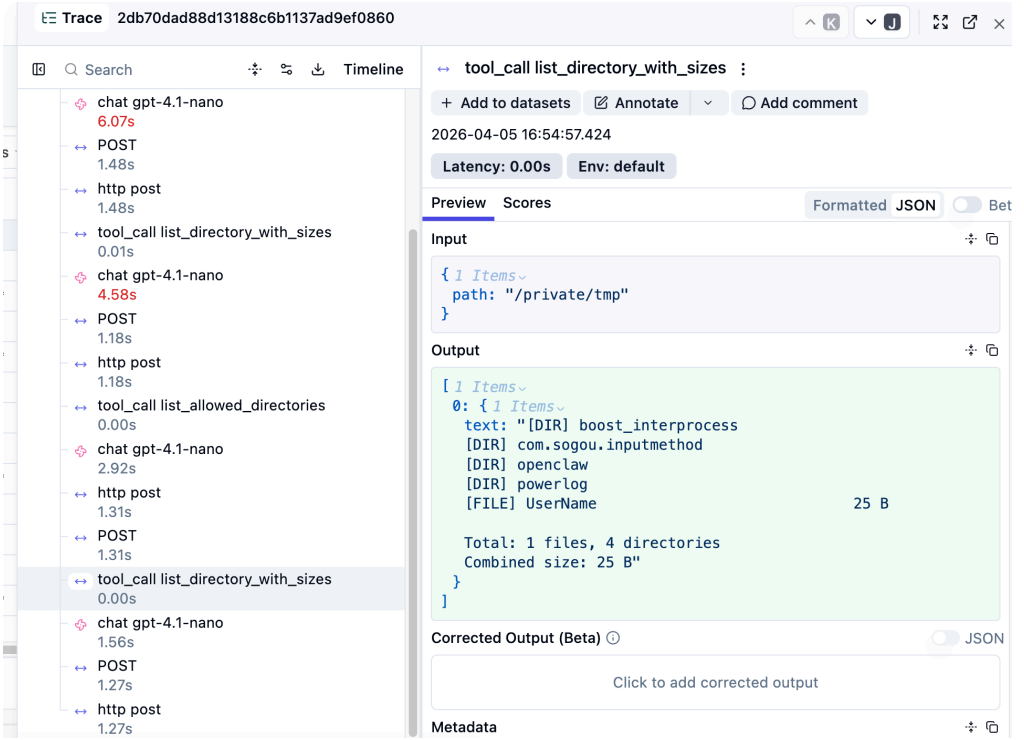

Figure 53. Langfuse UI

MLflow

MLflow is an open-source MLOps platform for managing the end-to-end machine learning lifecycle. It enables teams to track experiments, log parameters and metrics, package models, manage model versions, and deploy models across different environments. Designed to be framework-agnostic, MLflow integrates with popular machine learning libraries such as Scikit-learn, PyTorch, TensorFlow, XGBoost, and Hugging Face.

By providing a unified interface for experiment tracking, model registry, and deployment, MLflow helps improve reproducibility, collaboration, and oper-

ational efficiency in machine learning projects. Whether you're developing traditional machine learning models or modern generative AI applications, MLflow simplifies the process of managing models from experimentation to production.

MLflow is started using Docker Compose. See the docker-compose.yaml for the complete Docker Compose file to start MLflow.

We can configure the OpenTelemetry collector to send tracing data to MLflow. The `x-mlflow-experiment-id` header specifies the experiment ID in MLflow. This header is required, otherwise the request will be rejected with `400` errors.

Figure 54. MLflow exporter in OpenTelemetry collector config

```yaml
exporters:
  otlp_http/mlflow:
    endpoint: "http://mlflow:5000"
    headers:
      x-mlflow-experiment-id: "0"
    tls:
      insecure: true
```

Below is the screenshot of MLflow UI.

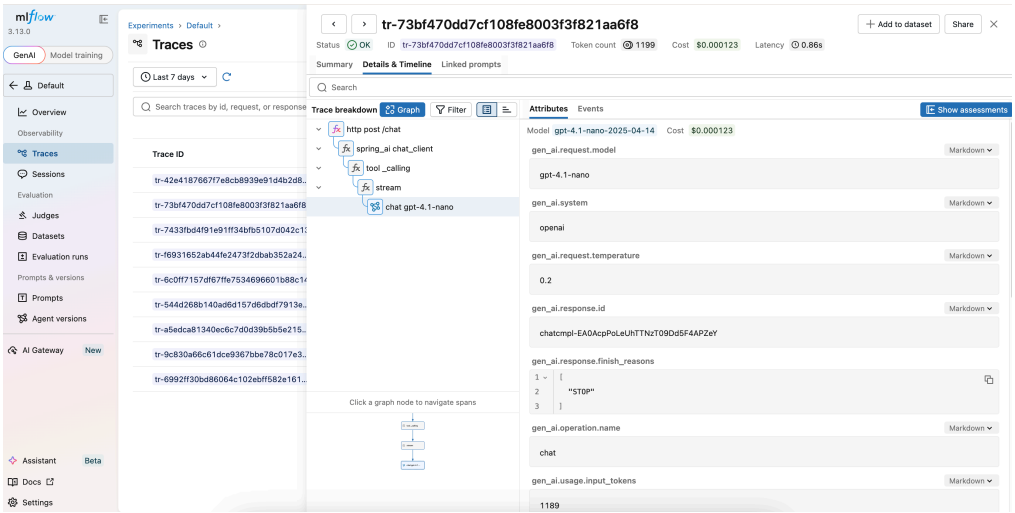

Figure 55. MLflow UI

AG-UI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Create Agent

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

AG-UI API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Frontend

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.

Source Code and Materials

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/spring-ai.