dgroove
Tecnologia emozionale.

ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

Novembre 2025

🔧 Setup progetto Spring AI

Multi-configurazione

Applicazioni business
Perché approccio multi configurazione?

- Selezione modello LLM in base alla *task*
  - Ragionamento complesso richiede modelli più potenti
  - Richieste semplici affidate a modelli più contenuti
  - Creazione *pipeline* con LLM specializzati in sotto-*task*
- Strategia di *fallback*
  - Molteplici configurazioni permettono *switch* automatici a modelli secondari se il primario non risponde
- *Test* comparativi
  - In base ad accuratezza, latenza, costi, …
- Preferenze utente
  - Approccio *user-centric* all’interazione con i modelli

- Inizializzare un progetto Spring con le seguenti caratteristiche:
  - Maven come *build tool*
  - Spring Boot alla versione più recente non SNAPSHOT
  - Linguaggio Java 21
  - *Group* `it.venis.ai.spring`
  - *Artifact* `demo`
  - `jar` *packaging*
  - `Spring Web`, `OpenAI` o `Ollama` come dipendenze
- ⚠ Riportare variabili di ambiente `env` in `launch.json` e `settings.json`

Progetto Spring AI
Applicazione e passaggi

- Stub di progetto Spring AI multi-configurazione (Gemini + Ollama)
  1. Creazione `application.yml` per applicativo multi LLM
  2. Creazione `docker-compose.yml` per servizio Docker Ollama
  3. Creazione *file* variabili di ambiente per servizio Ollama
  4. Creazione *script* per *start*, *stop* ed eliminazione servizi Docker
  5. Creazione configurazione multi-LLM
  6. Creazione modelli per domanda e risposta
  7. Creazione interfaccia ed implementazione del servizio di richiesta
  8. Creazione del controllore MVC
  9. *Test* delle funzionalità con Postman/Insomnia

Progetto Spring AI
Configurazione applicativo

File `application.yml`

```yaml
spring:
    application:
        name: demo
    ai:
        chat:
            client:
                enabled: false # Spring AI auto-configures a single ChatClient.Builder bean by default.
                               # Disabling ChatClient.Builder auto-configuration allows to manually
                               # configure multiple bean and inject them where needed.
        ollama:
            base-url: http://172.17.0.1:11434
            init:
                pull-model-strategy: when_missing
                timeout: 15m
                max-retries: 3
            chat:
              options:
                model: VitoF/llama-3.1-8b-italian
                temperature: 0.2
                top-k: 40
                top-p: 0.9
                repeat-penalty: 1.1
                presence-penalty: 1.0
        openai:
            api-key: ${GOOGLE_AI_API_KEY}
            base-url: https://generativelanguage.googleapis.com/v1beta/openai
            chat:
                completions-path: /chat/completions
                options:
                    model: gemini-2.0-flash-lite
                    temperature: 2.0
```

Progetto Spring AI
Servizio Docker Ollama - I

File `docker-compose.yml`

```yaml
services:
  spring-ai-llm-gpu:
    image: ollama/ollama:${OLLAMA_VERSION:-latest}
    hostname: spring-ai-llm
    container_name: spring_ai_llm
    environment:
      OLLAMA_DEBUG: ${OLLAMA_DEBUG:-false}
      OLLAMA_FLASH_ATTENTION: ${OLLAMA_FLASH_ATTENTION:-false}
      OLLAMA_KEEP_ALIVE: ${OLLAMA_KEEP_ALIVE:-"5m"}
      OLLAMA_MAX_LOADED_MODELS: ${OLLAMA_MAX_LOADED_MODELS:-1}
      OLLAMA_NUM_PARALLEL: ${OLLAMA_NUM_PARALLEL:-1}
    ports:
      - ${OLLAMA_PORT:-11434}:11434
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
    volumes:
      - spring_ai_llm:/root/.ollama
    restart: unless-stopped
    profiles: [llm-gpu]

...
```

Progetto Spring AI
Servizio Docker Ollama - II

File `docker-compose.yml`

```yaml
...

  spring-ai-llm-cpu:
    image: ollama/ollama:${OLLAMA_VERSION:-latest}
    hostname: spring-ai-llm
    container_name: spring_ai_llm
    environment:
      OLLAMA_DEBUG: ${OLLAMA_DEBUG:-false}
      OLLAMA_FLASH_ATTENTION: ${OLLAMA_FLASH_ATTENTION:-false}
      OLLAMA_KEEP_ALIVE: ${OLLAMA_KEEP_ALIVE:-"5m"}
      OLLAMA_MAX_LOADED_MODELS: ${OLLAMA_MAX_LOADED_MODELS:-1}
      OLLAMA_NUM_PARALLEL: ${OLLAMA_NUM_PARALLEL:-1}
    ports:
      - ${OLLAMA_PORT:-11434}:11434
    volumes:
      - spring_ai_llm:/root/.ollama
    restart: unless-stopped
    profiles: [llm-cpu]

volumes:
  spring_ai_llm:
    name: spring_ai_llm
```

Progetto Spring AI
Variabili di ambiente servizio Docker Ollama

File `spring-ai.env`

```
# MODE:
# "llm-cpu" --> Large Language Model in cpu mode
# "llm-gpu" --> Large Language Model in gpu mode
MODE=llm-cpu
COMPOSE_PROFILES=${MODE}
LOG_LEVEL=WARNING                                                  # default: WARNING

# Ollama configuration
#OLLAMA_VERSION=0.1.39                                              # default: latest
OLLAMA_PORT=11434                                                   # default: 11434
OLLAMA_DEBUG=false                                                  # default: false
OLLAMA_FLASH_ATTENTION=false                                        # default: false
OLLAMA_KEEP_ALIVE="5m"                                               # default: "5m"
OLLAMA_MAX_LOADED_MODELS=2                                             # default: 1
OLLAMA_NUM_PARALLEL=1                                                  # default: 1
```

Progetto Spring AI
Script servizi Docker

File `start_spring_ai_services.sh`

```bash
#!/bin/bash

stack=spring-ai-demo-services
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile up --build --remove-orphans --force-recreate --detach
```

Progetto Spring AI
Script servizi Docker

File `stop_spring_ai_services.sh`

```
#!/bin/bash

stack=spring-ai-demo-services
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile down --rmi $rmi
```

Progetto Spring AI
Script servizi Docker

File `erase_spring_ai_services.sh`

```bash
#!/bin/bash

stack=spring-ai-demo
rmi=local
file=docker-compose.yml
envfile=spring-ai.env

if [ -z ${1+x} ];
  then rmi=local;
  else rmi=$1;
fi

## --rmi flag must be one of: all, local
docker compose -f $file -p $stack --env-file $envfile down --rmi $rmi --volumes
```

Configurazione Gemini + Ollama

```java
package it.venis.ai.spring.demo.config;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.ollama.OllamaChatModel;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ChatClientConfig {

    @Bean
    public ChatClient geminiChatClient(OpenAiChatModel geminiChatClient) {
        return ChatClient.create(geminiChatClient);
        /*
         * or:
         * ChatClient.Builder chatClientBulder = ChatClient.builder(geminiChatClient);
         * return chatClientBulder.build();
         */
    }

    @Bean
    public ChatClient ollamaChatClient(OllamaChatModel ollamaChatModel) {
        ChatClient.Builder chatClientBulder = ChatClient.builder(ollamaChatModel);
        return chatClientBulder.build();
        /*
         * or:
         * return ChatClient.create(ollamaChatModel);
         */
    }

}
```

Modello per domanda

```java
package it.venis.ai.spring.demo.model;

import java.util.UUID;

public record Question(UUID id, String question) {

    public Question(String question) {

        this(UUID.randomUUID(), question);

    }
}
```

Modello per risposta

```java
package it.venis.ai.spring.demo.model;

public record Answer(String answer) {
}
```

Progetto Spring AI
Servizio multi LLM

Interfaccia servizio

```java
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.Question;

public interface QuestionService {

    Answer getGeminiAnswer(Question question);

    Answer getOllamaAnswer(Question question);

}
```

Implementazione servizio

```java
package it.venis.ai.spring.demo.services;

...

@Service
@Configuration
public class QuestionServiceImpl implements QuestionService {

    private final ChatClient geminiChatClient;
    private final ChatClient ollamaChatClient;

    public QuestionServiceImpl(@Qualifier("geminiChatClient") ChatClient geminiChatClient,
                               @Qualifier("ollamaChatClient") ChatClient ollamaChatClient) {
        this.geminiChatClient = geminiChatClient;
        this.ollamaChatClient = ollamaChatClient;
    }

    @Override
    public Answer getGeminiAnswer(Question question) {
        return new Answer(this.geminiChatClient.prompt()
                .user(question.question())
                .call()
                .content());
    }

    @Override
    public Answer getOllamaAnswer(Question question) {
        return new Answer(this.ollamaChatClient.prompt()
                .user(question.question())
                .call()
                .content());
    }

}
```

Implementazione controllore REST

```
package it.venis.ai.spring.demo.controllers;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.Question;
import it.venis.ai.spring.demo.services.QuestionService;

@RestController
public class QuestionController {

    private final QuestionService service;

    public QuestionController(QuestionService service) {
        this.service = service;
    }

    @PostMapping("/gemini/ask")
    public Answer geminiAskQuestion(@RequestBody Question question) {
        return this.service.getGeminiAnswer(question);
    }

    @PostMapping("/ollama/ask")
    public Answer ollamaAskQuestion(@RequestBody Question question) {
        return this.service.getOllamaAnswer(question);
    }

}
```

Codice
Branch di riferimento

https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git

Branch: `1-spring-ai-gemini-ollama-configuration`