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

Advisors

- Entità in grado di intercettare
  - la *request* dal `ChatClient` al LLM
  - la *response* del LLM prima che arrivi all'utente
- Utilizzi *advisors*
  - Pre-/post-processamento dei dati al/dal LLM
  - Validazione/filtraggio customizzato
  - Creare flussi di processamento puliti e sequenziali
- Linee guida di buon utilizzo
  - Evitare comportamenti *session-scoped*
  - Creare più *advisors* in catena piuttosto che un unico *advisor* complesso
  - Evitare comportamenti che intacchino la logica del sistema (es. no modifiche al *prompt*)

- *Advisors built-in*
  - `SimpleLoggerAdvisor` riporta informazione dettagliata delle strutture di *request* e di *response*
  - `SafeGuardAdvisor` valida la *request* utente relativamente ad una *blacklist*
  - `PromptChatMemoryAdvisor` recupera e copia la memoria nel *prompt* come contesto di sistema
  - ...
- *Advisor* utente
  - Devono implementare `CallAdvisor` e/o `StreamAdvisor`

Configurazione statica

```
chatClientBuilder
    .defaultAdvisors(new SimpleLoggerAdvisor(1), new SafeGuardAdvisor(0))
    .build();
```

Configurazione dinamica

```
chatClient
    .prompt()
    .advisors(new SimpleLoggerAdvisor(1), new SafeGuardAdvisor(0))
    .user(message)
    .call()
    .content();
```

- *Advisors* per `ChatClient` Ollama
  1. Modifica `applicaytion.yml` per gestione *logging*
  2. Modifica configurazioni di `ChatClient` per Ollama
  3. Creazione modello per gestore prezzi
  4. Creazione *advisor* per calcolo risparmio economico Ollama
  5. *Test* delle funzionalità con Postman/Insomnia

Configurazione *logging*

```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://172.19.0.2:11434
            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
logging:
    level:
        org:
            springframework:
                ai:
                    chat:
```

Configurazione Gemini + Ollama

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

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

import it.venis.ai.spring.demo.advisors.OllamaCostSavingsAdvisor;

@Configuration
public class ChatClientConfig {

    @Bean
    public ChatClient geminiChatClient(OpenAiChatModel geminiChatClient) {

        return ChatClient.create(geminiChatClient);

        /*
         * or:
         * ChatClient.Builder chatClientBuilder = ChatClient.builder(geminiChatClient);
         * return chatClientBulder.build();
         */

    }

    @Bean
    public ChatClient ollamaChatClient(OllamaChatModel ollamaChatModel) {

        ChatClient.Builder chatClientBuilder = ChatClient.builder(ollamaChatModel);

        return chatClientBuilder
                .defaultAdvisors(new SimpleLoggerAdvisor(), new OllamaCostSavingsAdvisor())
                .defaultSystem(
                        """
```

Modello di gestione costi

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

public record ModelPricing(Float inputPrice, Float outputPrice) {

}
```

Advisor per risparmio Ollama

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

import java.util.HashMap;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.ai.chat.client.ChatClientRequest;
import org.springframework.ai.chat.client.ChatClientResponse;
import org.springframework.ai.chat.client.advisor.api.CallAdvisor;
import org.springframework.ai.chat.client.advisor.api.CallAdvisorChain;
import org.springframework.ai.chat.metadata.Usage;
import org.springframework.ai.chat.model.ChatResponse;

import it.venis.ai.spring.demo.model.ModelPricing;

public class OllamaCostSavingsAdvisor implements CallAdvisor {

    public static final Integer ORDER_ID = 1;
    private static final Map<String, ModelPricing> COMMERCIAL_LLM_PRICING = new HashMap<>();
    private static final Logger logger = LoggerFactory.getLogger(OllamaCostSavingsAdvisor.class);

    static {
        /*
         * Commercial API usage costs, updated 2025/10/22, jtlyk.
         */

        /*
         * OpenAI GPT-5
         */
        COMMERCIAL_LLM_PRICING.put("gpt-5-pro", new ModelPricing(15.0f, 120.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5", new ModelPricing(1.25f, 10.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5-mini", new ModelPricing(0.25f, 2.0f));
        COMMERCIAL_LLM_PRICING.put("gpt-5-nano", new ModelPricing(0.05f, 0.4f));

        /*
         * Anthropic Claude
```

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

Branch: `3-spring-ai-gemini-ollama-advisors`