



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



CONCETTI AVANZATI

- ➔ Come tenere traccia di diverse conversazioni?
 - ➔ ChatMemory definita secondo una logica **multi-conversazione**
 - ⚠ **CONVERSATION_ID** chiave che il sistema ricerca nel contesto per CRUD storico messaggistica
 - ⚠ **DEFAULT_CONVERSATION_ID** valore di *default* fornito alla chiave se non popolata programmaticamente

Interfaccia Chat Memory

```
public interface ChatMemory {  
    String DEFAULT_CONVERSATION_ID = "default";  
    /**  
     * The key to retrieve the chat memory conversation id from the context.  
     */  
    String CONVERSATION_ID = "chat_memory_conversation_id";  
    ...  
}
```

- ➔ Come sovrascrivere il valore di *default*?
- ➔ Agire a livello di *chat memory advisor*
- ➔ Utilizzo delle *advisor specifications* di `ChatClient` (`ChatClient$AdvisorSpec`)

Interfaccia per ogni *chat memory advisor*

```
public interface BaseChatMemoryAdvisor extends BaseAdvisor {  
    /**  
     * Retrieve the conversation ID from the given context or return the default conversation ID when not found.  
     */  
    default String getConversationId(Map<String, Object> context, String defaultConversationId) {  
        Assert.notNull(context, "context cannot be null");  
        Assert.noNullElements(context.keySet().toArray(), "context cannot contain null keys");  
        Assert.hasText(defaultConversationId, "defaultConversationId cannot be null or empty");  
        return context.containsKey(ChatMemory.CONVERSATION_ID) ? context.get(ChatMemory.CONVERSATION_ID).toString()  
            : defaultConversationId;  
    }  
}
```

Customizzazione *id* conversazione

```
String conversation_id = "custom_conversation_id";  
  
ChatClient  
    .prompt()  
    .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID, conversation_id))  
    .build();
```

- ➔ Configurazione *per-user chat memory* per ChatClient Ollama
 - 1 Creazione modello per richiesta domanda con *username*
 - 2 Modifica interfaccia e implementazione del servizio di risposta
 - 3 Modifica controllore MVC
 - 4 Test delle funzionalità con Postman/Insomnia

Modello richiesta con *username*

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

import java.util.Objects;
import java.util.UUID;

public record QuestionRequest(UUID id, String username, Question body) {

    public QuestionRequest(String username, Question body) {
        this(UUID.randomUUID(), username, body);
    }

    @Override
    public String username() {
        return Objects.requireNonNullElse(this.username, "default");
    }
}
```

Interfaccia servizio

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

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

public interface QuestionService {

    Answer getGeminiAnswer(Question question);

    Answer getOllamaAnswer(Question question);

    Answer getOllamaDefaultAnswer(Question question);

    Answer getOllamaMemoryAwareAnswer(Question question);

    Answer getOllamaPerUserMemoryAwareAnswer(QuestionRequest request);

}
```


Implementazione servizio

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

...

@Service
@Configuration
public class QuestionServiceImpl implements QuestionService {
    private final ChatClient geminiChatClient;
    private final ChatClient ollamaChatClient;
    private final ChatClient ollamaMemoryChatClient;

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

    ...

    @Override
    public Answer getOllamaPerUserMemoryAwareAnswer(QuestionRequest request) {
        return new Answer(this.ollamaMemoryChatClient
            .prompt()
            .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID, request.username()))
            .user(request.body().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.model.QuestionRequest;
import it.venis.ai.spring.demo.services.QuestionService;

@RestController
public class QuestionController {

    ...

    @PostMapping("/ollama/ask/memory")
    public Answer getOllamaMemoryAwareAnswer(@RequestBody Question question) {

        return this.service.getOllamaMemoryAwareAnswer(question);

    }

    @PostMapping("/ollama/ask/memory/user")
    public Answer getOllamaPerUserMemoryAwareAnswer(@RequestBody QuestionRequest request) {

        return this.service.getOllamaPerUserMemoryAwareAnswer(request);

    }

}
```

<https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git>
Branch: 5-spring-ai-gemini-ollama-per-user-chat-memory

- ⚠ Storico della conversazione valida per singola sessione
 - ➔ Opzione valida per demo/applicazioni a bassa criticità
 - ➔ Possibile **persistere** le informazioni di storico su DB
 - ➔ PostgreSQL
 - ➔ MySQL/MariaDB
 - ➔ SQL Server
 - ➔ HSQLDB

Dipendenza di progetto

```
<dependency>  
  <groupId>org.springframework.ai</groupId>  
  <artifactId>spring-ai-starter-model-chat-memory-repository-jdbc</artifactId>  
</dependency>
```

Definizione bean

```
@Autowired
JdbcChatMemoryRepository chatMemoryRepository;

ChatMemory chatMemory = MessageWindowChatMemory.builder().chatMemoryRepository(chatMemoryRepository)
    .maxMessages(10)
    .build();
```

- 1 Il sistema inferisce la tipologia di DB dalle proprietà dell'applicativo

File application.yml

```
spring:
  datasource:
    url: jdbc:postgresql:...
```

- 2 Il sistema applica lo schema relativo al DB inferito

File schema-postgresql.sql

```
CREATE TABLE IF NOT EXISTS SPRING_AI_CHAT_MEMORY (
  conversation_id VARCHAR(36) NOT NULL,
  content TEXT NOT NULL,
  type VARCHAR(10) NOT NULL CHECK (type IN ('USER', 'ASSISTANT', 'SYSTEM', 'TOOL')),
  "timestamp" TIMESTAMP NOT NULL
);

CREATE INDEX IF NOT EXISTS SPRING_AI_CHAT_MEMORY_CONVERSATION_ID_TIMESTAMP_IDX
ON SPRING_AI_CHAT_MEMORY(conversation_id, "timestamp");
```

1 Definire un nuovo schema per il nuovo connettore

File ./resources/schema/schema-h2.sql

```
CREATE TABLE SPRING_AI_CHAT_MEMORY (  
  conversation_id VARCHAR(36) NOT NULL,  
  content LONGVARCHAR NOT NULL,  
  type VARCHAR(10) NOT NULL,  
  "timestamp" TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL  
);  
  
CREATE INDEX SPRING_AI_CHAT_MEMORY_CONVERSATION_ID_TIMESTAMP_IDX  
ON SPRING_AI_CHAT_MEMORY(conversation_id, timestamp DESC);  
  
ALTER TABLE SPRING_AI_CHAT_MEMORY ADD CONSTRAINT TYPE_CHECK CHECK (type IN ('USER', 'ASSISTANT', 'SYSTEM', 'TOOL'));
```

2 Impostare i corretti parametri dell'applicativo

File application.yml

```
spring:
  ai:
    chat:
      ...
      memory:
        repository:
          jdbc:
            initialize-schema: always # always, never, embedded (default)
            schema: classpath:/schema/schema-h2.sql
      ...
    datasource:
      url: jdbc:h2:file:./db/chatmemory
      driver-class-name: org.h2.Driver
      username: user
      password: pwd
    h2:
      console:
        enabled: true
    ...
```

File pom.xml

<pre><dependency> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-devtools</artifactId> <optional>true</optional> </dependency></pre>	<pre><dependency> <groupId>com.h2database</groupId> <artifactId>h2</artifactId> <scope>runtime</scope> </dependency></pre>
---	--

- ➔ Configurazione *JDBC H2 chat memory* per `ChatClient` Ollama
 - 1 Modifica al `pom.xml` per dipendenze Spring AI JDBC, H2 e *devtools*
 - 2 Modifica ad `application.yml`
 - 3 Creazione schema H2
 - 4 Modifica ai *bean* `ChatClient` per Ollama
 - 5 Test delle funzionalità con Postman/Insomnia

Dipendenze di sistema aggiuntive

```
...  
<dependency>  
  <groupId>org.springframework.ai</groupId>  
  <artifactId>spring-ai-starter-model-chat-memory-repository-jdbc</artifactId>  
</dependency>  
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <scope>runtime</scope>  
</dependency>  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-devtools</artifactId>  
  <optional>true</optional>  
</dependency>  
...
```

Configurazione applicativo

```
spring:
  ai:
    chat:
      ...
      memory:
        repository:
          jdbc:
            initialize-schema: always # always, never, embedded (default)
            schema: classpath:/schema/schema-h2.sql
      ...
    datasource:
      url: jdbc:h2:file:./db/chatmemory
      driver-class-name: org.h2.Driver
      username: user
      password: pwd
    h2:
      console:
        enabled: true
  ...
```

Schema H2

```
CREATE TABLE SPRING_AI_CHAT_MEMORY (  
    conversation_id VARCHAR(36) NOT NULL,  
    content LONGVARCHAR NOT NULL,  
    type VARCHAR(10) NOT NULL,  
    "timestamp" TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL  
);  
  
CREATE INDEX SPRING_AI_CHAT_MEMORY_CONVERSATION_ID_TIMESTAMP_IDX  
ON SPRING_AI_CHAT_MEMORY(conversation_id, timestamp DESC);  
  
ALTER TABLE SPRING_AI_CHAT_MEMORY ADD CONSTRAINT TYPE_CHECK CHECK (type IN ('USER', 'ASSISTANT', 'SYSTEM', 'TOOL'));
```

Configurazione Gemini + Ollama

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

...

@Configuration
public class MemoryChatClientConfig {

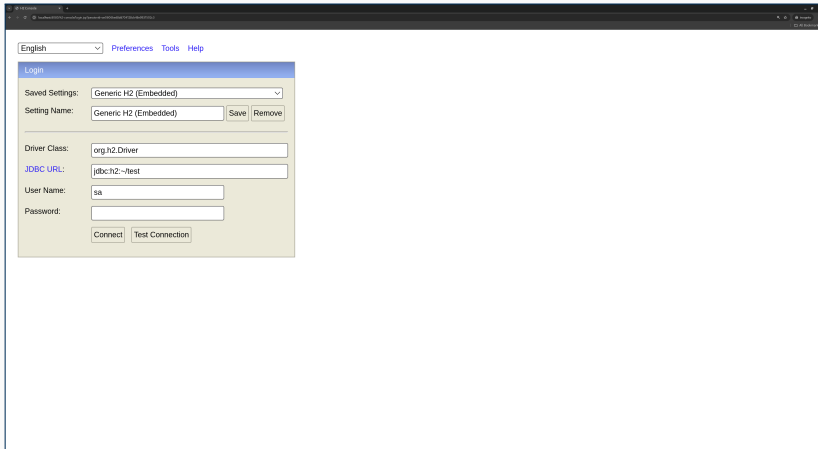
    ...

    /**
     * Bean to store messages into relational DBs. This is done
     * under the hood when the corresponding .pom dependency is added.
     * The system infers the type of DB through properties in application
     * .properties/.yml file.
     */
    @Autowired
    JdbcChatMemoryRepository jdbcChatMemoryRepository;

    /**
     * Custom bean for chat memory, based on the (auto-)configured chat memory repository.
     * Done under-the-hood via Spring auto-configuration with maxMessages = 20.
     */
    @Bean
    public ChatMemory chatMemory(@Qualifier("jdbcChatMemoryRepository") ChatMemoryRepository chatMemoryRepository) {
        return MessageWindowChatMemory.builder()
            .chatMemoryRepository(chatMemoryRepository)
            .maxMessages(20)
            .build();
    }

    ...
}
```

 Verificare la *dashboard* H2 (<http://localhost:8080/h2-console>)



The screenshot shows the H2 console web interface in a browser window. The interface has a dark header bar with the title "H2 Console" and a menu bar with "English", "Preferences", "Tools", and "Help". Below the header is a "Login" section with a blue header. The "Login" section contains the following fields and buttons:

- Saved Settings:** A dropdown menu showing "Generic H2 (Embedded)".
- Setting Name:** A text input field containing "Generic H2 (Embedded)". To its right are "Save" and "Remove" buttons.
- Driver Class:** A text input field containing "org.h2.Driver".
- JDBC URL:** A text input field containing "jdbc:h2:~test".
- User Name:** A text input field containing "sa".
- Password:** A text input field.
- At the bottom of the login section are "Connect" and "Test Connection" buttons.

<https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git>
Branch: 6-spring-ai-gemini-ollama-jdbc-chat-memory