dgroove
Tecnologia emozionale.

ICT Training Center

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

Note

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

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

Novembre 2025

Note

- Tecnica di mitigazione della *knowledge cut-off*…
- …ma non solo
  - Ricerca di informazioni non presenti nella sua base di conoscenza (es. “Che tempo fa a Venezia?”)
  - ⚠ Quindi, differenza con RAG *web search*...?!
  - Messa in atto di ciò che comprende
    - “Prenota un biglietto”
    - “Invia un’email”
    - “Crea un nuovo *ticket* di segnalazione guasto”
  - Primo passaggio nella transizione da LLM a ALM (Action Language Model)

Note

Tool Calling	RAG
Chiamare un idraulico per riparare una perdita	Leggere un manuale fai-da-te per risolverla da solo
“In base a quello che mi hai chiesto, ecco come far agire il sistema con i mezzi a disposizione”	“In base a quello che mi hai chiesto e avedno letto alcuni documenti, ti spiego”
Esegue azioni dal vivo o recupera dati dal vivo	Si concentra sulla generazione delle risposte utilizzando contenuti recuperati
Richiede all’*app client* di eseguire il tool	Recupera solo documenti e continua a generare la risposta

Note

Classe di definizione del *tool*

```
@Component
public class TimeTools {
    @Tool(name="getCurrentLocalTime", description = "Ottieni l'ora corrente nel fuso orario dell'utente")
    String getCurrentLocalTime() { }

    @Tool(name = "getCurrentTime", description = "Ottieni l'ora corrente nel fuso orario specificato.")
    public String getCurrentTime(@ToolParam(
        description = "Valore che rappresenta il fuso orario") String timeZone) { }
}
```

Due metodi: Uno usa la locale corrente dell'utente e l'altro accetta una stringa di fuso orario (come "Asia/Kolkata"). Pensa a questi come metodi di utilità che l'AI può chiamare.

Note

2. Il ChatClient Consapevole dei Tool – ToolsChatClientConfig

```java
chatClientBuilder .defaultTools(timeTools)
```

Stiamo iniettando il tool nel ChatClient così il modello ne è a conoscenza

3. Il Controller – ToolsCallingController

```java
@GetMapping("/local-time") public ResponseEntity<String> localTime (...)
```

Usa chatClient.prompt() per interagire con l'AI

Note

Come i Tool vengono chiamati dall'LLM?

Può sembrare che Spring AI esponga il tuo tool come endpoint per OpenAI da chiamare, ma non è quello che succede. Invece, avviene una conversazione dietro le quinte tra la tua app e il modello ogni volta che un tool è coinvolto.

Note

Come i Tool vengono chiamati dall'LLM?

Note

- Gli LLM non “eseguono” realmente i tools. Sono come un project manager—delegano.
- L’applicazione client riceve la richiesta, chiama il tool/API, e invia il risultato al modello.

Come Aiuta Spring AI:

- Puoi definire tools come semplici bean Java o metodi annotati.
- Spring AI:
  - Rileva le richieste di chiamata tool dal modello
  - Mappa automaticamente gli argomenti di input
  - Esegue il metodo tool giusto
  - Invia i risultati al modello come un assistente esperto di tecnologia!

Note

“Se il RAG è la tua AI che legge un libro per rispondere a una domanda, il Tool Calling è la tua AI che chiama un amico per far fare le cose!”

Note

Tool Context – Condivisione di Informazioni Extra

ToolContext è un meccanismo in Spring AI che ti permette di iniettare dati extra—non parte del prompt principale o degli argomenti del tool—nel flusso di esecuzione del tool. È utile per scenari in cui un tool ha bisogno di accedere a metadati specifici dell'utente/sessione come username, userId, o organizationId.

1. Passare Dati Contestuali
Questo contesto non è direttamente visibile al modello AI. È destinato puramente alla logica backend durante l'invocazione del tool.

java

```
String answer = chatClient.prompt() .advisors(a ->
a.param(CONVERSATION_ID, username)).user(message).tools(helpDeskTools).toolContext(Map.of("username", us
```


Note

2. Accedere ai Dati Contestuali nel Tool
Il ToolContext viene iniettato automaticamente quando il tool viene chiamato. toolContext.getContext().get("username"): Recupera il valore di contesto (username) che è stato passato in precedenza.

java

```
@Tool(description = "Recupera lo stato dei ticket aperti basato su un dato username")
List<HelpDeskTicket> getTicketStatus(ToolContext toolContext)  String username = (String)
toolContext.getContext().get("username"); return service.getTicketsByUsername(username);
```


Note

L'esecuzione del tool implica invocare un tool con gli argomenti di input dati e restituire il risultato. Questo processo è gestito dall'interfaccia ToolCallingManager, che supervisiona l'intero ciclo di vita dell'esecuzione.

Esso:

- Invoca il tool
- Passa gli argomenti
- Coordina tutto

DefaultToolCallingManager è l'implementazione auto-configurata dell'interfaccia ToolCallingManager. Per personalizzare il comportamento di esecuzione del tool, puoi definire e registrare il tuo bean ToolCallingManager.

Note

Il risultato della chiamata al tool viene convertito in una String usando l'interfaccia ToolCallResultConverter e inviato al modello AI. Per impostazione predefinita, è serializzato in JSON con Jackson tramite DefaultToolCallResultConverter, ma puoi personalizzarlo fornendo la tua implementazione.

Note

Return Direct – Salta la Risposta del Modello

Per impostazione predefinita, il risultato del Tool viene inviato all'LLM così può ragionare e continuare a chattare.

Ma a volte, vuoi:

- Restituire il risultato del tool direttamente all'utente
- Saltare l'elaborazione extra da parte del modello

Il flag returnDirect dell'annotazione @Tools dice a Spring AI di non inviare l'output del tool al modello AI per ulteriore elaborazione. Invece, restituisce immediatamente il risultato del tool direttamente come risposta finale all'utente (o al chiamante).

java

```
@Tool(description = "Crea il Ticket di Supporto", returnDirect = true) String createTicket(
@ToolParam(required = true, description = "Dettagli per creare un Ticket di Supporto")
TicketRequest ticketRequest, ToolContext toolContext)  String username = (String)
toolContext.getContext().get("username"); HelpDeskTicket savedTicket =
service.createTicket(ticketRequest, username); return "Ticket " + savedTicket.getId() + " creato con
successo per l'utente " + savedTicket.getUsername();
```


Note

Usalo quando:

- Il tool genera un messaggio finale leggibile dall'uomo.
- Non vuoi che il modello alteri, riassuma o continui il ragionamento dopo l'esecuzione del tool.
- Vuoi ridurre la latenza saltando il passaggio di post-elaborazione.

Note

Gestione Errori con Grazia – ToolExecutionException

Se il tuo tool genera un errore, Spring lo avvolge in una ToolExecutionException

Per impostazione predefinita, il messaggio viene inviato al modello. Invece possiamo generare l'eccezione all'app client,

java

```
@Bean ToolExecutionExceptionProcessor processor()
return new
DefaultToolExecutionExceptionProcessor(true);
```

Se stai usando Spring AI Boot Starters, DefaultToolExecutionExceptionProcessor è il gestore predefinito per gli errori di esecuzione del tool.

Per impostazione predefinita, invia messaggi di errore al modello.

Puoi impostare il flag alwaysThrow nel suo costruttore su true per generare eccezioni invece di restituire messaggi di errore.

Note