



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

- ➔ 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)

| 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 avendo 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' <i>app client</i> di eseguire il <i>tool</i> | Recupera solo documenti e continua a generare la risposta |

Classe di definizione dei 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) {
        ...
    }
}
```

- ➔ Solitamente istanziati come `@Component`
- ➔ Ogni metodo a disposizione del LLM come *tool* deve essere annotato come `@Tool`
 - ➔ se `name` non specificato, Spring AI popola il parametro con il nome del metodo
 - ⚠ `name` come **identificativo univoco** per Spring AI!
 - ➔ `description` fornisce il **contesto** al LLM per determinare se utilizzare il *tool* in base alla richiesta utente!
 - ➔ `@ToolParam` fornisce ulteriore contesto al LLM per iniettare parametri al metodo

Classe di definizione dei 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) {
        ...
    }
}
```

👤 "D: Che ore sono a New York?"

>_ (thinking) — Scansiono i tool a disposizione...—

>_ (thinking) — Vedo che ho dei tool che gestiscono il recupero dell'ora...—

>_ (thinking) — Il primo tool è relativo alla posizione dell'utente...—

>_ (thinking) — Il secondo tool è più astratto e determina la posizione attraverso parametrizzazione del fuso orario...—

>_ (thinking) — ...Il fuso orario di New York è America/New York...—

>_ "R: Utilizza il tool `getCurrentTime('America/New York')`."

Processo di *method as tool*

ChatClient potenziato con *tool*

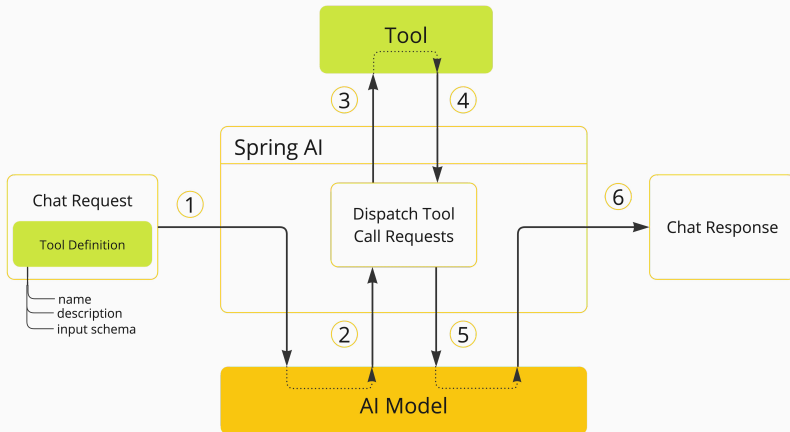
```
@Bean
public ChatClient ollamaChatClient(OllamaChatModel ollamaChatModel, TimeTools timeTools) {

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

    return chatClientBuilder
        .defaultTools(timeTools)
        ...
        .build();
}
```

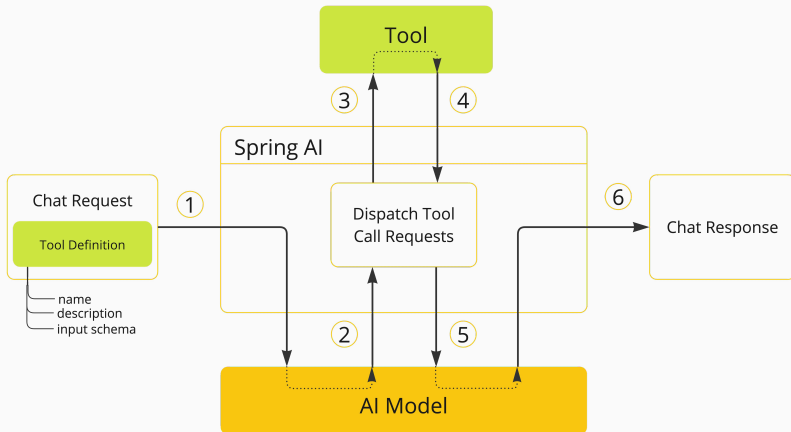
➔ Disponibilità dei *tool* al ChatClient attraverso *injection*

- 1 L'utente invia la richiesta al LLM attraverso `ChatClient/ChatModel` riferendo al LLM i *tool* a sua disposizione



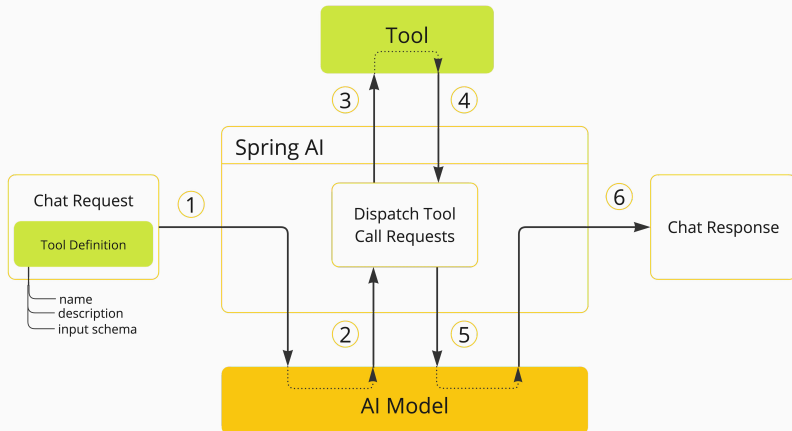
©Spring AI

- 2** Se il LLM decide di chiamare un *tool*, invia una *response* con nome del *tool* e parametri istanziati, seguendo la schema definito da `@Tool` e `@ToolParam`



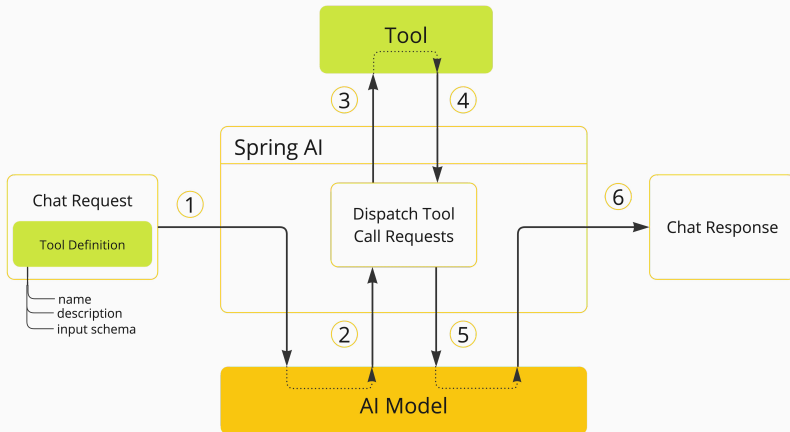
©Spring AI

- 3 L'applicativo interroga il `@Bean` corrispondente alla classe che contiene il metodo con i parametri istanziati suggeriti dal LLM



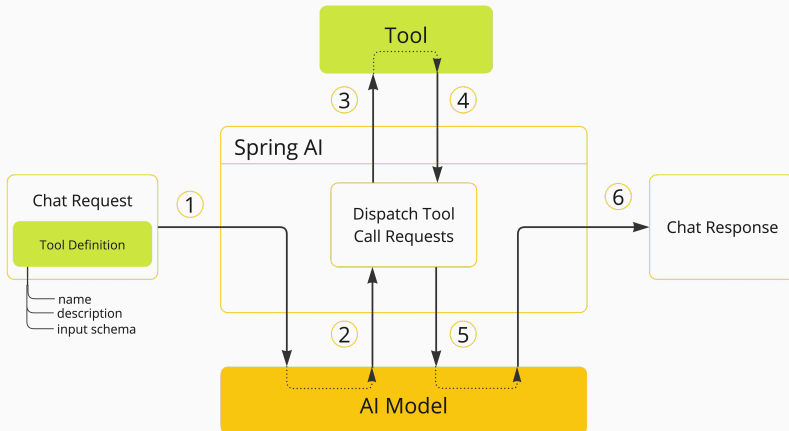
©Spring AI

- 4 Il @Bean invoca il metodo con i parametri restituiti dal LLM e restituisce il risultato al *container* Spring



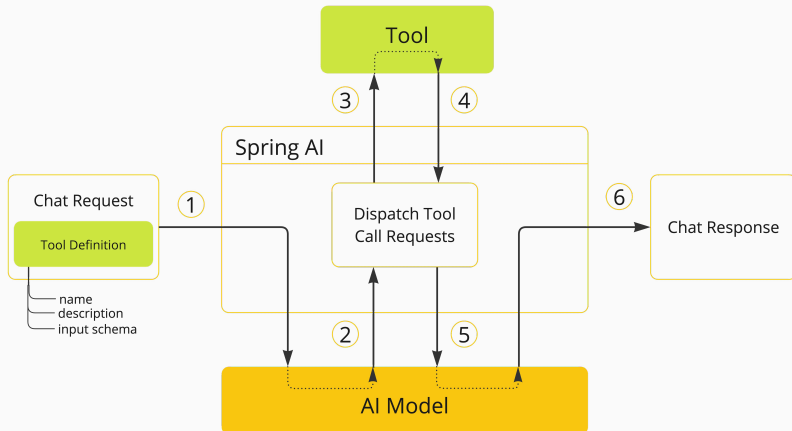
©Spring AI

- 5 Il sistema invoca nuovamente il LLM con la richiesta utente precedente ma ulteriormente contestualizzata dall'*output* del *tool*



©Spring AI

- 6** Il LLM restituisce la risposta finale al ChatClient/ChatModel sfruttando generazione di testo e contesto



©Spring AI