Spring AI上手开发

1. 概述

SpringAI是一个旨在简化大模型应用开发过程的框架，支持对符合OpenAI形式的API服务以及阿里云、谷歌等其他大模型厂商API服务的调用，功能上基本对标LangChain，提供了上下文管理、持久化消息存储、RAG、MCP、工具调用、提示词工程等常见功能。

在我看来，SpringAI是一个功能强大、渐进式且灵活的框架，它可以通过简单的配置实现大部分常见的功能，同时支持实现接口和继承类来实现自定义部分组件。

2. 环境信息

JDK 17

SpringBoot：3.4.6

SpringAI：1.0.0

MySQL：8.0

3. 快速开始

3.1 导入依赖

可参考Github文件https://github.com/mahoushoujyo-eee/LearningSpringAI/blob/master/pom.xml

3.2 完成配置

填入如下配置（默认使用支持OpenAI形式的API）

Spring配置

```
# OpenAI 配置
spring.ai.openai.api-key=你的API密钥
spring.ai.openai.base-url=你的API地址
spring.ai.openai.chat.options.model=模型名称
spring.ai.openai.chat.options.temperature=0.7

# 数据库配置
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/spring_ai?useSSL=false&serverTimezone=UTC&characterEncoding=utf-8
spring.datasource.username=root
spring.datasource.password=你的密码
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# MyBatis 配置
mybatis.mapper-locations=classpath:mapper/*.xml
mybatis.type-aliases-package=com.atguigu.learningspringai.entity
mybatis.configuration.map-underscore-to-camel-case=true
```

3.3 完成一个基础的Controller

ChatController

```
@RestController
public class ChatController {

    //注入刚才配置的openai的api服务模型
    @Autowired
    private OpenAiChatModel chatModel;

    //提供http服务封装的api服务
    @GetMapping("/ai/generate")
    public String chatModelTest(@RequestParam(value = "message", defaultValue = "hello")
                                String message)
    {
        String response = this.chatModel.call(message);
        System.out.println("response : "+response);
        return response;
    }
}
```

这样就完成了一个基础的API调用服务

4. 概念介绍

SpringAI中有几个常用的类，在框架中起着核心作用，下面一一介绍。

4.1 ChatModel

在SpringAI中，ChatModel就是对应了我们调用的模型。它包含了api-key，base-url， temperature等模型基础信息。这些可以通过完成配置文件后自动装配也可以在程序内进行实例化。

下面以openai形式的api服务为例进行演示。

4.1.1 使用配置文件

在配置文件中完成如下配置：

配置

```
# OpenAI 配置
spring.ai.openai.api-key=你的API密钥
spring.ai.openai.base-url=你的API地址
spring.ai.openai.chat.options.model=模型名称
spring.ai.openai.chat.options.temperature=0.7
```

完成配置后直接自动装配即可

ChatController

```
@RestController
public class ChatController {

    //注入刚才配置的openai的api服务模型
    @Autowired
    private OpenAiChatModel chatModel;

    @GetMapping("/ai/generate")
    public String chatModelTest(@RequestParam(value = "message", defaultValue
= "hello")
                                String message)
    {
        String response = this.chatModel.call(message);
        System.out.println("response : "+response);
        return response;
    }
}
```

4.1.2 在程序中配置

在程序中可以直接实例化一个ChatModel，然后通过Api类配置api的相关信息，通过ChatOptions配置模型相关信息，最后填入ChatModel实例化。

配置示例

```
@GetMapping("/chat-model/chat")
public String chatModelTest(@RequestParam(value = "message", defaultValue =
"hello")
```

```
                        String message)
{
    OpenAiApi openAiApi = OpenAiApi.builder()
            .baseUrl("https://api.qnaigc.com")
            .apiKey("sk-xxx")
            .build();
    OpenAiChatOptions options = OpenAiChatOptions.builder()
            .model("deepseek-v3")
            .temperature(0.8)
            .build();
    OpenAiChatModel chatModel = OpenAiChatModel.builder()
            .defaultOptions(options)
            .openAiApi(openAiApi).build();


    String response = chatModel.call(message);
    System.out.println("response : "+response);
    return response;
}
```

4.2 ChatClient

ChatClient是构建在ChatModel之上的组件，把ChatModel封装为了一个功能更丰富、使用更方便的一个API，正如类的名字，它是一个API调用服务的客户端。这是SpringAI中相当核心的一个类，其余的许多功能都通过它配置起来。它有两种常用配置方式，一种是使用配置类，另一种是直接在程序中实例化。

4.2.1 使用配置类

这里需要注意，Bean方法中有一个Builder参数，它使用了自动装配的ChatModel。如果是手动配置的需要我们自己在程序中填入对应Builder。

代码块

```
@Configuration
public class ChatClientConfig {

    //利用一个配置类来完成ChatClient的配置，包括上下文记忆、提示词、RAG等
    @Bean
    public ChatClient chatClient(ChatClient.Builder builder) {

        return builder
                //配置Advisor，后面会讲
                .defaultAdvisors()
                //配置系统默认提示词
                .defaultSystem()
```

```
                //配置模型信息，前面提到过的Options
                .defaultOptions()
                //配置工具调用
                .defaultToolCallbacks()
                .build();
    }
}
```

正如所用的链式方法的名字所说，这些配置都是默认配置，我们在后面进行使用的时候还可以再重新配置。

4.2.2 程序中进行配置

其实跟配置类的配置方式基本一样，这里只重点解释调用，也就是return后面的部分。

每次chatClient.call都是一次调用，而前面的配置就是临时性的配置，看名字可以基本看出来各个方法的作用，下面一一解释：

1. user/system：用户/系统提示词
2. toolCallbacks：工具调用配置
3. advisors：顾问的参数配置

代码块

```
@GetMapping("/chat-client/chat")
public String chatClientTest(@RequestParam(value = "message", defaultValue = "hello")
                        String message)
{
    //使用配置好的ChatModel
    ChatClient chatClient = ChatClient.builder(chatModel)
            .defaultOptions()
            .defaultSystem()
            .build();

    return chatClient.prompt()
            .user(message)
            .toolCallbacks()
            .advisors()
            .system()
            .call()
            .content();
}
```

4.3 Advisor

顾问(Advisor)是模型拓展功能依赖的核心类，大部分提示词相关功能都需要通过配置顾问完成。

SpringAI预置了几个常用的Advisor。

常见的有MessageChatMemoryAdvisor（对话信息处理）、QuestionAnswerAdvisor、RetrievalAugmentationAdvisor（RAG相关）

想要使用Advisor也很简单，如果使用预先定义好的Advisor可以直接根据参数使用Builder方法，大部分Builder方法的参数都是所讲的常用组件。builder构建后直接根据前面所讲配置到ChatClient中就可以了。

代码块

```
ChatClient chatClient = ChatClient.builder(chatModel)
        .defaultAdvisors(
            MessageChatMemoryAdvisor.builder(chatMemory).build()
         )
        .build();
```

4.4 ChatMemory

ChatMemory是处理多轮对话功能的核心组件，它负责消息存储（数据库/内存/其他）、上下文管理等功能。

ChatMemory本身只是一个接口，我们根据自己需要去使用预先配置的类或是自定义的类，SpringAI提供了一个InMemoryChatMemory的实现类，顾名思义，就是一个直接把对话消息存到内存中管理的类，对于简单的测试是没有问题的，但是不建议使用。

对于有持久化需求的场景，我们更建议使用MessageWindowChatMemory，后续讲解持久化消息存储的时候会讲解如何使用。

这里是构建MessageWindowChatMemory的方式，后面会详细讲解。

代码块

```
MessageWindowChatMemory.builder()
        .chatMemoryRepository(jdbcChatMemoryRepository)
        .maxMessages(10) // 最大消息记录条数
        .build();
```

关于使用则是按照如下方式填入Advisor

代码块

```
@Autowired
ChatMomery chatMomery

ChatClient chatClient = ChatClient.builder(chatModel)
```

```
        .defaultAdvisors(
            MessageChatMemoryAdvisor.builder(chatMemory).build()
         )
        .build();
```

4.5 ChatMemoryRepository

ChatMemoryRepository是一个在底层负责管理ChatMemory消息存储方式的接口，这里有几个预置的实现类：InMemoryChatMemoryRepository、JdbcChatMemoryRepository（使用关系型数据库存储消息）、CassandraChatMemoryRepository（使用Apache Cassandra存储聊天消息）、Neo4jChatMemoryRepository（使用Neo4j图数据库存储聊天消息）。他们都有各自的持久化配置方式，后面我们会重点介绍JDBC存储配置。

4.6 Message

Message是代表了我们聊天对话中的每个消息个体，它被SpringAI很好的封装在了底层，我们基本不会使用。对于一个Message，常见的信息要素有：text（内容）、type（类型：System/User/Assistant）

5. 常用功能实现示例

5.1 流式响应

把call()换为stream()就实现了流式响应。

流式响应注意引入对应Flux依赖，produces的参数形式决定了流式响应的格式，常见的是event-stream，但是也有其他如text/html的响应方式，注意设置编码格式，不然可能会出现乱码。

代码块

```
@GetMapping(value = "/ai/stream/chat", produces = "text/event-stream;charset=UTF-8")
public Flux<String> streamChat(@RequestParam(value = "message",defaultValue = "给我讲个笑话")
                               String message) {

    return this.chatClient.prompt()
            //用户输入的信息
            .user(message)
            //请求大模型
            .stream()
            //返回文本
            .content();
}
```

5.2 消息持久化

消息持久化主要是对ChatMemory进行一定配置，我们先从上往下看。

我们需要这样构建一个ChatMemory实现类，顾名思义，这里实现持久化的核心参数是jdbcChatMemoryRepository

代码块

```
MessageWindowChatMemory.builder()
        .chatMemoryRepository(jdbcChatMemoryRepository)
        .maxMessages(10) // Configure the maximum number of messages to retain
        .build();
```

这其实是ChatMemoryRepository接口的一个实现类，下面介绍它的配置方式。

5.2.1 首先它默认会直接使用Spring的Datasource配置来做持久化的数据库

代码块

```
spring.datasource.url=jdbc:mysql://localhost:3306/spring_ai?
useSSL=false&serverTimezone=UTC&characterEncoding=utf-8
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

5.2.2 随后需要配置持久化的建表

第一条有never、always、embedded三个参数，分别要求从不初始化表、总是初始化表和初始化向量化数据库。

第二条则是表示初始化数据库的sql语句的位置。

代码块

```
spring.ai.chat.memory.repository.jdbc.initialize-schema=never
spring.ai.chat.memory.repository.jdbc.schema=classpath:chatmemory/schema.sql
```

如果第一个为never则不要使用第二个参数，否则会出错。

所需的SQL建表语句如下：

代码块

```
CREATE TABLE `SPRING_AI_CHAT_MEMORY` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `conversation_id` varchar(255) NOT NULL,
```

```
  `type` varchar(255) NOT NULL,
  `content` text NOT NULL,
  `timestamp` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=470 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

5.2.3 直接使用自动装配好的JdbcTemplate

接下来使用配置类完成Repository和ChatMemory的配置。

为JdbcChatMemoryRepository配置jdbcTemplate和数据库相关信息，随后直接注入ChatMemory配置方法，完成ChatMemory配置，最后直接在Advisor中使用即可（关于Advisor如何配置可以回去看前面）。

代码块

```
@Configuration
public class ChatMemoryConfig {

    @Bean
    public JdbcChatMemoryRepository jdbcChatMemoryRepository(JdbcTemplate
jdbcTemplate) {
        return JdbcChatMemoryRepository.builder()
                .jdbcTemplate(jdbcTemplate)
                //这里是选择数据库类型，还有其他的数据库可用，可以自行查看相关API
                .dialect(new MysqlChatMemoryRepositoryDialect())
                .build();
    }

    @Bean
    public ChatMemory chatMemory(JdbcChatMemoryRepository
jdbcChatMemoryRepository) {

        //return MessageWindowChatMemory.builder().build();
        return MessageWindowChatMemory.builder()
                .chatMemoryRepository(jdbcChatMemoryRepository)
                .maxMessages(10) // Configure the maximum number of messages
to retain
                .build();
    }
}
```

梳理一下，基本的配置流程是：

SQL -> JDBC -> Repository -> ChatMemory -> Advisor -> ChatClient

5.3 工具调用/函数调用

函数调用已在新版本废弃，被工具调用完全替代，这里只讲解工具调用。

工具调用的核心就是方法上的@Tool注解，有如下几个参数

1. description：最主要的参数，输入给模型这个工具的描述。
2. required：表示参数是否为必须的，默认为true。

代码块

```
public class ToolTest
{
    @Tool(description = "测试工具调用的接口")
    public String test(String message)
    {
        log.info("测试工具调用成功！" + message);
        return "测试工具调用成功！" + message;
    }

    @Tool(description = "那一天的忧郁，忧郁起来")
    public String responseDirectly()
    {
        log.info("测试工具调用成功！");
        return "那一天的寂寞，寂寞起来";
    }
}
```

使用则是根据前面提到的ChatClient示例中，直接把这个类传入tool相关方法。

代码块

```
chatClient.prompt()
        .user(message)
        .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID,
conversationId))
        .tools(new ToolTest())
        .call()
        .content();
```

5.4 MCP

配置MCP前，首先要了解MCP，这里不多赘述，可以自己去找一些教程。

这里重点讲解的是MCP Client的配置方式。

配置MCP的方式也有两种，一种是直接写到配置文件中，一种是写到配置类中。

5.4.1 配置文件中

使用配置文件还有两种配置方式，一种是直接写在配置文件里，一种是写在Anthropic规定的JSON文件中。

1. 写在JSON文件中：用如下方式指定路径（stdio/sse根据自己实际调用方式自行切换，但是测试中JSON文件如果使用sse会出现报错）

```
代码块
spring.ai.mcp.client.stdio.servers-configuration=classpath:mcp/fetch-web.json
```

随后在指定JSON文件中完成配置（一般的MCP Server会提供相关信息）

```
fetch-web.json
{
  "mcpServers": {
    "fetch": {
      "type": "sse",
      "url": "https://mcp.api-inference.modelscope.net/a30b4276b9cc40/sse"
    }
  }
}
```

2. 使用配置文件：因为MCP配置格式特殊且复杂，建议使用yaml格式进行配置，这里就参考MCP Server提供的JSON信息进行配置。

```
代码块
spring:
  ai:
    mcp:
      client:
        enabled: true
        name: my-mcp-client
        version: 1.0.0
        request-timeout: 30s
        type: SYNC  # or ASYNC for reactive applications
        sse:
          connections:
            server1:
```

```yaml
            url: http://localhost:8080
          server2:
            url: http://otherserver:8081
      stdio:
        root-change-notification: false
        connections:
          server1:
            command: /path/to/server
            args:
              - --port=8080
              - --mode=production
            env:
              API_KEY: your-api-key
              DEBUG: "true"
```

5.4.2 使用配置类

这里以高德地图为例

首先需要配置好通用信息，随后使用配置类注入容器

代码块

```
spring.ai.mcp.client.request-timeout=60s
spring.ai.mcp.client.type=ASYNC
```

代码块

```java
//注意这里使用的不是SpringAI包下的
import io.modelcontextprotocol.spec.McpClientTransport;

@Configuration
public class OneMCPConfig
{
    //高德地图的MCP服务
    @Bean
    public List<NamedClientMcpTransport> mcpClientTransport() {
        McpClientTransport transport = HttpClientSseClientTransport
                //base_url
                .builder("https://mcp.amap.com")
                //sse
                .sseEndpoint("/sse?key=xxxx")
                //传输消息的序列化/反序列化
                .objectMapper(new ObjectMapper())
                .build();

```

```
        return Collections.singletonList(new NamedClientMcpTransport("amap", transport));
    }
}
```

配置完成后，可以直接把MCP的各种工具看作前面所讲的工具调用中的Tools，他们会被自动装配到AsyncMcpToolCallbackProvider和List<McpAsyncClient>中（同步则是xxxSyncxxx），可以直接把一个ToolCallbackProvider填入ChatClient中，就可以直接使用。

代码块

```
@Autowired
private List<McpAsyncClient> mcpAsyncClients;
@Autowired
private ChatClient chatClient;
@Autowired
private AsyncMcpToolCallbackProvider toolCallbackProvider;

String  response = chatClient.prompt()
        .user(message)
        //直接填入
        .toolCallbacks(toolCallbackProvider)
        .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID, conversationId))
        .call()
        .content();
```

5.5 RAG