Spring AI

作者： -- 天气预报

日期： 2026-07-18

01. Deepseek

> https://platform.deepseek.com

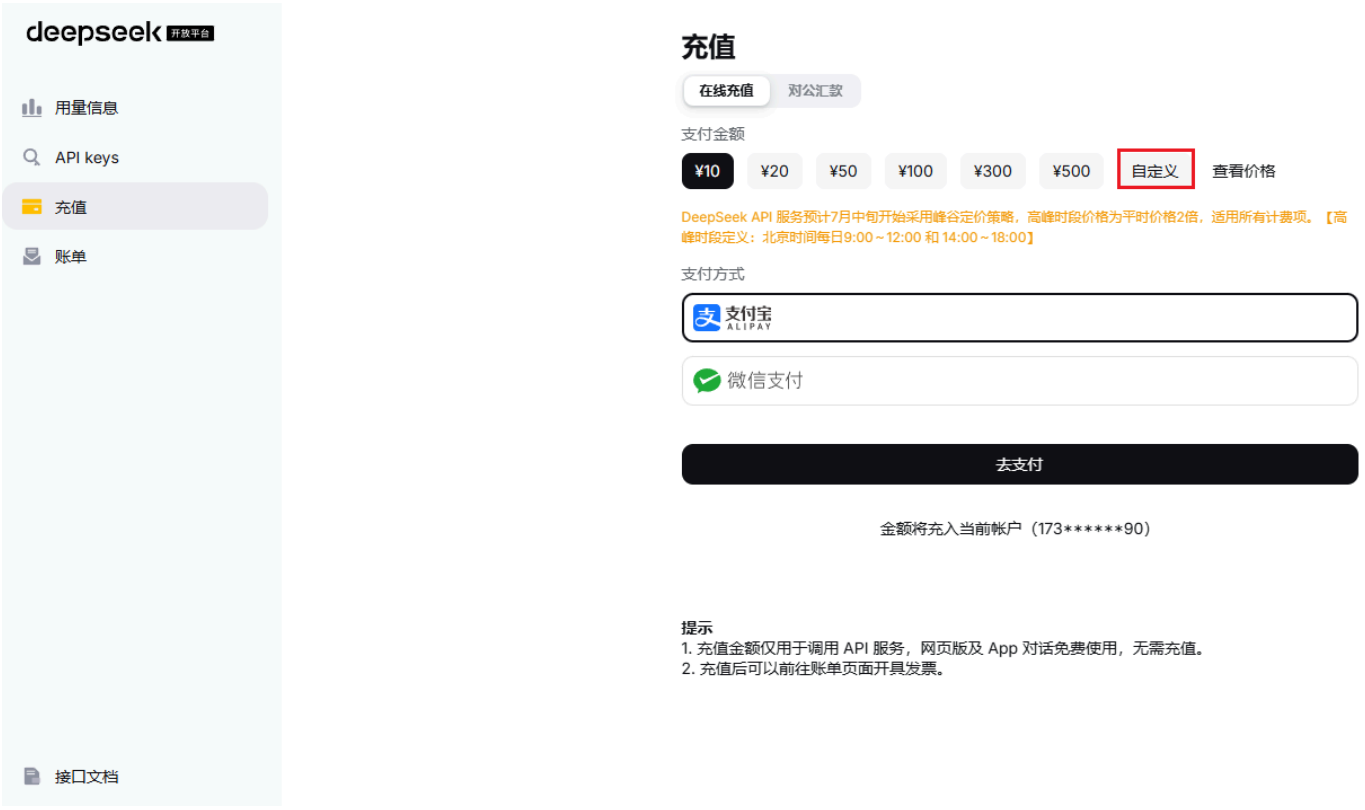
1.1 登录

- ui

1.2 充值

> 自行充值

- ui



1.3 申请 Api key

> 直接创建 API key 后复制 key 并自行存储

- ui

1.4 Api

> 请求地址 | 令牌 | 请求使用模型

- baseUrl : https://api.deepseek.com/chat/completions [http]
- baseUrl : https://api.deepseek.com [openai 标准]
- apiKey : sk-f568b*******ed4c
- model   : deepseek-v4-pro | deepseek-v4-flash | deepseek-chat ...

02. 阿里云百炼

> https://bailian.console.aliyun.com/cn-beijing [大量模型可白嫖]

- ui

2.1 申请 Api key

> 可支付宝、淘宝扫码登录、登录后即可创建

- ui

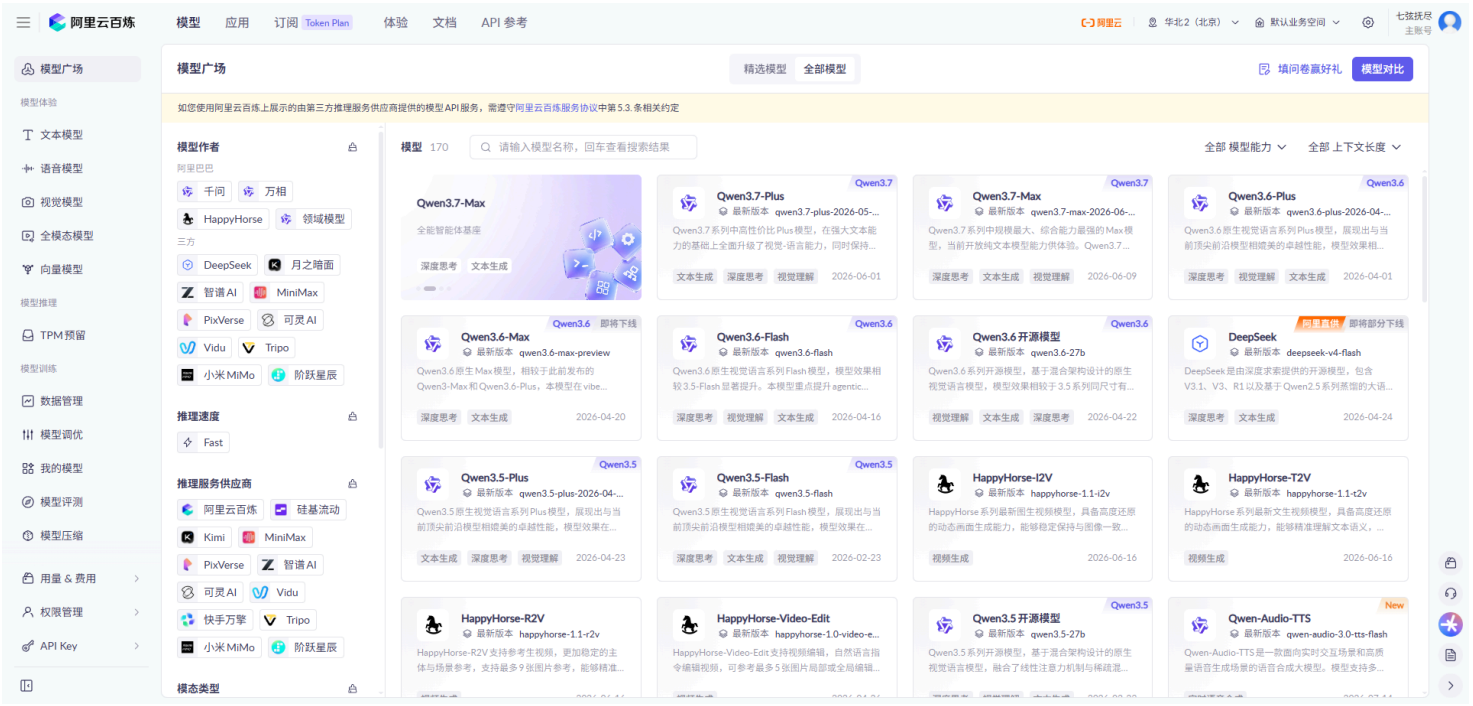
> 请保存创建信息

- ui

2.2 模型

> https://bailian.console.aliyun.com/cn-beijing?tab=model#/model-market/all

- ui



2.3 Api

> 请求地址 | 令牌 | 请求使用模型

- baseUrl : https://ws-h4qjfke1wt127m1g.cn-beijing.maas.aliyuncs.com/compatible-mode/v1
- apiKey : sk-ws-H.EDXERRM.ma5i.MEY...
- model : text-embedding-v4 | qwen3.7-text-embedding ...

03. OpenAI

> https://developers.openai.com/api/reference/java

- pom

```
<dependency>
    <groupId>com.openai</groupId>
    <artifactId>openai-java</artifactId>
    <version>4.43.0</version>
</dependency>

<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.5.38</version>
    <scope>compile</scope>
</dependency>
```

3.1 对话

> OpenAIClient 配置请求信息 | ChatCompletionCreateParams 配置模型信息

☐ LogLevel.DEBUG: 该级别可查看请求体/及响应体全部信息

- OpenAi.java

```
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.LogLevel;
import com.openai.models.chat.completions.ChatCompletion;
import com.openai.models.chat.completions.ChatCompletionCreateParams;

import java.util.Optional;

public class OpenAi {
```

```
    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams params = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                .addUserMessage("你好")
                .build();

        ChatCompletion completion = client.chat().completions().create(params);

        Optional<String> content = completion.choices().getFirst().message().content();

        System.out.println(content.orElse(""));

        client.close();
    }

}
```

- post

```
{
    "messages": [
        {
            "content": "你是一个简洁的中文助手",
            "role": "system"
        },
        {
            "content": "你好",
            "role": "user"
        }
    ],
    "model": "deepseek-v4-pro"
}
```

- body


```
{
    "id": "e5a14065-df0e-4550-a655-e1a1313c6e74",
    "object": "chat.completion",
    "created": 1784369628,
    "model": "deepseek-v4-pro",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "你好！有什么可以帮你的？",
                "reasoning_content": "嗯，用户发来简单的问候“你好”。..."
            },
            "logprobs": null,
            "finish_reason": "stop"
        }
    ],
    "usage": {
        "prompt_tokens": 11,
        "completion_tokens": 69,
        "total_tokens": 80,
        "prompt_tokens_details": {
            "cached_tokens": 0
        },
        "completion_tokens_details": {
            "reasoning_tokens": 61
        },
        "prompt_cache_hit_tokens": 0,
        "prompt_cache_miss_tokens": 11
    },
    "system_fingerprint": "fp_9954b31ca7_prod0820_fp8_kvcache_20260402"
}
```


3.2 流式响应

> client.chat().completions().createStreaming(params)

- OpenAi.java

```java
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.LogLevel;
import com.openai.core.http.StreamResponse;
import com.openai.models.chat.completions.ChatCompletion;
import com.openai.models.chat.completions.ChatCompletionChunk;
import com.openai.models.chat.completions.ChatCompletionCreateParams;

import java.util.Optional;

public class OpenAi {

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams params = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                .addUserMessage("你好")
                .build();

        StreamResponse<ChatCompletionChunk> response =
            client.chat().completions().createStreaming(params);

        response.stream()
                .flatMap(chunk -> chunk.choices().stream())
                .flatMap(choice -> choice.delta().content().stream())
                .forEach(System.out::print);

        System.out.println();

        client.close();
    }

}
```

- body

```
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
data: {"id":"c55a0938-f579-48ba-8742-abf65b550b42","object":"chat.completion.chunk", ...
```

- data

```
data: {
    "id": "c55a0938-f579-48ba-8742-abf65b550b42",
    "object": "chat.completion.chunk",
    "created": 1784369902,
    "model": "deepseek-v4-pro",
    "system_fingerprint": "fp_9954b31ca7_prod0820_fp8_kvcache_20260402",
    "choices": [
        {
            "index": 0,
            "delta": {
                "role": "assistant",
                "content": null,
                "reasoning_content": ""
            },
            "logprobs": null,
            "finish_reason": null
        }
    ]
}
```

3.3 结构化输出

> ═ 提示词方式不保证结构一定符合需求 | `response_format` 可强制指定模型输出格式

3.3.1 系统提示词

> 可通过系统提示词指定格式 addSystemMessage()

- [ ] 注意提示词严谨、确保数据结构包含不匹配数据、且命令模型不允许胡编乱造

- OpenAi.java

```
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.LogLevel;
import com.openai.models.ResponseFormatJsonObject;
import com.openai.models.chat.completions.ChatCompletion;
import com.openai.models.chat.completions.ChatCompletionCreateParams;

import java.util.Optional;

public class OpenAi {

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams params = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage(
                        """
    你是信息提取助手，只输出 JSON，格式：
    {
        "matched": true/false,      // 输入是否包含人物信息
```

```
        "data": {"name": ..., "age": ..., "city": ...} 或 null,
        "reason": 不匹配时说明原因，匹配时为 null
    }

    提取不到的字段填 null，不要编造

    示例1：输入"张三25岁住上海"\s
    {"matched": true, "data": {"name":"张三","age":25,"city":"上海"}, "reason": null}

    示例2：输入"今天天气怎么样"\s
    {"matched": false, "data": null, "reason": "输入与人物信息无关"}
                        """
                )
                .addUserMessage("张三今年25岁，住在上海")
                .responseFormat(ResponseFormatJsonObject.builder().build())
                .build();

        ChatCompletion.Choice choice = client.chat().completions().create(params)
            .choices().getFirst();

        Optional<String> content = choice.message().content();

        System.out.println(content.orElse(""));

        client.close();
    }

}
```

- body

```
{
    "id": "7fd4e904-cf58-4fdd-b849-bd7410024002",
    "object": "chat.completion",
    "created": 1784372226,
    "model": "deepseek-v4-pro",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "{"matched": true, "data":
                {"name": "张三", "age": 25, "city": "上海"}, "reason": null}",

                "reasoning_content": "我们被要求输出JSON。输入是张三今年25岁，住在上海 ..."
            },
            "logprobs": null,
            "finish_reason": "stop"
```

```
        }
    ]
}
```

3.3.2 response_format

> response_format 强制指定模型输出结构 | deepseek 占不支持

- OpenAi.java

```java
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.JsonValue;
import com.openai.core.LogLevel;
import com.openai.models.ResponseFormatJsonObject;
import com.openai.models.ResponseFormatJsonSchema;
import com.openai.models.chat.completions.ChatCompletion;
import com.openai.models.chat.completions.ChatCompletionCreateParams;

import java.util.List;
import java.util.Map;
import java.util.Optional;

public class OpenAi {

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        ResponseFormatJsonSchema.JsonSchema.Schema schema =
                ResponseFormatJsonSchema.JsonSchema.Schema.builder()
                        .putAdditionalProperty("type", JsonValue.from("object"))
                        .putAdditionalProperty("properties", JsonValue.from(Map.of(
                                "name", Map.of("type", "string"),
                                "age", Map.of("type", "int"),
                                "city", Map.of("type", "string")
                        )))
                        .putAdditionalProperty("required",
```

```
                                                    JsonValue.from(List.of("name", "age",
"city")))
                        .putAdditionalProperty("additionalProperties",
JsonValue.from(false))
                        .build();

        ResponseFormatJsonSchema.JsonSchema jsonSchema =
            ResponseFormatJsonSchema.JsonSchema.builder()
                .name("person")
                .schema(schema)
                .strict(true)
                .build();

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams params = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                .addUserMessage("张三今年25岁，住在上海")

.responseFormat(ResponseFormatJsonSchema.builder().jsonSchema(jsonSchema).build())
                .build();

        client.chat().completions().create(params);

        client.close();
    }

}
```

- post

```
{
    "messages": [
        {
            "content": "你是一个简洁的中文助手",
            "role": "system"
        },
        {
            "content": "张三今年25岁，住在上海",
            "role": "user"
        }
```

```
    ],
    "model": "deepseek-v4-pro",
    "response_format": {
        "json_schema": {
            "name": "person",
            "schema": {
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string"
                    },
                    "age": {
                        "type": "int"
                    },
                    "name": {
                        "type": "string"
                    }
                },
                "required": [
                    "name",
                    "age",
                    "city"
                ],
                "additionalProperties": false
            },
            "strict": true
        },
        "type": "json_schema"
    }
}
```

3.3.3 @JsonPropertyDescription

> openai sdk 内部可将注解解析为 schema 并附带在请求上

- OpenAi.java

```
package org.example;

import com.fasterxml.jackson.annotation.JsonPropertyDescription;
import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
```

```java
import com.openai.core.LogLevel;
import com.openai.models.chat.completions.ChatCompletionCreateParams;
import com.openai.models.chat.completions.StructuredChatCompletionCreateParams;

public class OpenAi {

    public static class Person {
        @JsonPropertyDescription("姓名")
        public String name;

        @JsonPropertyDescription("年龄")
        public int age;

        @JsonPropertyDescription("居住城市")
        public String city;
    }

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");


        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        StructuredChatCompletionCreateParams<Person> params = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                .addUserMessage("张三今年25岁，住在上海")
                .responseFormat(Person.class)v
                .build();

        client.chat().completions().create(params);

        client.close();
    }

}
```

3.4 上下文维护

> 大模型本身只具备推理能力、不具备记忆能力、上下文须不断累积对话

- OpenAi.java

```java
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.JsonValue;
import com.openai.core.LogLevel;
import com.openai.models.ResponseFormatJsonObject;
import com.openai.models.ResponseFormatJsonSchema;
import com.openai.models.chat.completions.ChatCompletion;
import com.openai.models.chat.completions.ChatCompletionCreateParams;
import com.openai.models.chat.completions.ChatCompletionMessage;

import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Scanner;

public class OpenAi {

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams.Builder history = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手");

        Scanner scanner = new Scanner(System.in);

        while (true) {
```

```
            System.out.println("你: ");
            String prompt = scanner.nextLine();

            if (prompt.isBlank() || prompt.equalsIgnoreCase("quit")) {
                return;
            }

            history.addUserMessage(prompt);

            ChatCompletionMessage message = client.chat().completions()
                .create(history.build())
                .choices().getFirst().message();

            System.out.println(message.content().orElse(""));

            // 累积模型回答信息
            history.addMessage(message);

        }

    }

}
```

> messages: [不断累积对话] [role: system | user | assistant]

- body

```
{
    "messages": [
        {
            "content": "你是一个简洁的中文助手",
            "role": "system"
        },
        {
            "content": "我是小童",
            "role": "user"
        },
        {
            "role": "assistant",
            "content": "你好，小童！有什么可以帮你的吗？"
        },
        {
            "content": "我是谁呀？",
```

```
            "role": "user"
        }
    ],
    "model": "deepseek-v4-pro"
}
```

3.5 工具

> 请求时全量发送工具、模型响应后、须循环处理所有工具调用

3.5.1 定义调用

- GetWeather.java

```java
package org.example.tools;

import com.fasterxml.jackson.annotation.JsonClassDescription;
import com.fasterxml.jackson.annotation.JsonPropertyDescription;
import com.fasterxml.jackson.annotation.JsonTypeName;

@JsonTypeName("get_weather")
@JsonClassDescription("查询指定城市当前的实时天气,包括温度、天气状况和风力")
public class GetWeather {

    @JsonPropertyDescription("城市名称,使用中文,不带省份和'市'字,例如:北京、上海、杭州")
    public String city;

    public String execute() {
        return city + " 28°C, 晴, 微风";
    }
```

```
}
```

- OpenAi.java

```
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.LogLevel;
import com.openai.models.chat.completions.*;
import org.example.tools.GetWeather;

import java.util.List;
import java.util.Optional;

public class OpenAi {

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams.Builder history = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                .addTool(GetWeather.class)
                .addUserMessage("广州天气怎么样");

        ChatCompletionMessage message = client
                .chat()
                .completions()
                .create(history.build())
                .choices()
                .getFirst()
                .message();

        // 模型响应回来的信息
        history.addMessage(message);
```

```
        List<ChatCompletionMessageToolCall> toolCalls =
message.toolCalls().orElse(List.of());

        for (ChatCompletionMessageToolCall toolCall : toolCalls) {

            ChatCompletionMessageFunctionToolCall fn = toolCall.asFunction();
            String id = fn.id();
            String name = fn.function().name();
            if (name.equals("get_weather")) {
                String weather = fn.function().arguments(GetWeather.class).execute();
                ChatCompletionToolMessageParam toolParam = ChatCompletionToolMessageParam
                        .builder()
                        .toolCallId(id)
                        .content(weather)
                        .build();
                // 将工具调用信息放入上下文
                history.addMessage(toolParam);
            }

        }

        ChatCompletion completion = client.chat().completions().create(history.build());
        Optional<String> content = completion.choices().getFirst().message().content();
        System.out.println(content.orElse(""));

        client.close();
    }

}
```

- post

```
{
    "id": "afab522b-e526-41e1-9dc2-bbbbb73f5883",
    "object": "chat.completion",
    "created": 1784380755,
    "model": "deepseek-v4-pro",
    "choices": [
        {
            "index": 0,
            "message": {
                "role": "assistant",
                "content": "",
                "reasoning_content": "用户想知道广州的天气，我需要调get_weather函数，城市参数应该
是广州",
                "tool_calls": [
                    {
                        "index": 0,
```

```
                        "id": "call_00_gREhmuKCC5NyM5y3BTP09112",
                        "type": "function",
                        "function": {
                            "name": "get_weather",
                            "arguments": "{\"city\": \"广州\"}"
                        }
                    }
                ]
            },
            "logprobs": null,
            "finish_reason": "tool_calls"
        }
    ]
}
```

- body

```
{
    "messages": [
        {
            "content": "你是一个简洁的中文助手",
            "role": "system"
        },
        {
            "content": "广州天气怎么样",
            "role": "user"
        },
        {
            "role": "assistant",
            "content": "",
            "tool_calls": [
                {
                    "id": "call_00_gREhmuKCC5NyM5y3BTP09112",
                    "function": {
                        "arguments": "{\"city\": \"广州\"}",
                        "name": "get_weather"
                    },
                    "type": "function",
                    "index": 0
                }
            ]
        },
        {
            "content": "广州 28°C，晴，微风",
            "role": "tool",
            "tool_call_id": "call_00_gREhmuKCC5NyM5y3BTP09112"
        }
    ],
```

```
    "model": "deepseek-v4-pro",
    "tools": [
        {
            "function": {
                "name": "get_weather",
                "description": "查询指定城市当前的实时天气,包括温度、天气状况和风力",
                "parameters": {
                    "$schema": "https://json-schema.org/draft/2020-12/schema",
                    "type": "object",
                    "properties": {
                        "city": {
                            "type": "string",
                            "description": "城市名称,使用中文,不带省份和'市'字,例如:北京、上海、
杭州"
                        }
                    },
                    "required": [
                        "city"
                    ],
                    "additionalProperties": false
                },
                "strict": true
            },
            "type": "function"
        }
    ]
}
```

3.5.2 工具编排

> 多工具循环逻辑须自行处理、避免对某个工具出现硬编码调用

- Tool.java

```
package org.example.tools;

public interface Tool {
    Object execute();
}
```

- GetWeather.java

```java
package org.example.tools;

import com.fasterxml.jackson.annotation.JsonClassDescription;
import com.fasterxml.jackson.annotation.JsonPropertyDescription;
import com.fasterxml.jackson.annotation.JsonTypeName;

@JsonTypeName("get_weather")
@JsonClassDescription("查询指定城市当前的实时天气,包括温度、天气状况和风力")
public class GetWeather implements Tool {

    @JsonPropertyDescription("城市名称,使用中文,不带省份和'市'字,例如:北京、上海、杭州")
    public String city;

    public String execute() {
        return city + " 28°C, 晴, 微风";
    }

}
```

- GetCurrentTime.java

```java
package org.example.tools;

import com.fasterxml.jackson.annotation.JsonClassDescription;
import com.fasterxml.jackson.annotation.JsonTypeName;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@JsonTypeName("get_current_time")
@JsonClassDescription("获取当前的系统日期和时间,格式为 yyyy-MM-dd HH:mm:ss")
public class GetCurrentTime implements Tool{

    static DateTimeFormatter FORMATTER = DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss");

    public String execute() {
        return LocalDateTime.now().format(FORMATTER);
    }

}
```

- ToolTable.java

```java
package org.example.tools;

import com.openai.core.JsonSchemaLocalValidation;
import com.openai.models.chat.completions.ChatCompletionCreateParams;

import java.util.Collection;
import java.util.Map;

public final class ToolTable {

    public static Map<String, Class<?extends Tool>> TOOLS = Map.of(
            "get_weather", GetWeather.class,
            "get_current_time", GetCurrentTime.class
    );

    public static void register(ChatCompletionCreateParams.Builder builder) {
        Collection<Class<? extends Tool>> toolClasses = TOOLS.values();
        for (Class<? extends Tool> toolClass : toolClasses) {
            // 关闭本地 工具函数格式校验
            builder.addTool(toolClass, JsonSchemaLocalValidation.NO);
        }
    }

}
```

- OpenAi.java

```java
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.core.LogLevel;
import com.openai.models.chat.completions.*;
import org.example.tools.Tool;
import org.example.tools.ToolTable;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.List;
import java.util.Optional;

public class OpenAi {
```

```
    static Logger logger = LoggerFactory.getLogger(OpenAi.class);

    static void main() {

        String baseUrl = "https://api.deepseek.com";
        String apiKey = System.getenv("DEEPSEEK-API-KEY");

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(apiKey)
                .logLevel(LogLevel.DEBUG)
                .build();

        ChatCompletionCreateParams.Builder history = ChatCompletionCreateParams
                .builder()
                .model("deepseek-v4-pro")
                .addSystemMessage("你是一个简洁的中文助手")
                // .addUserMessage("广州天气怎么样");
                .addUserMessage("明天是几号？");

        ToolTable.register(history);

        ChatCompletionMessage message = client
                .chat()
                .completions()
                .create(history.build())
                .choices()
                .getFirst()
                .message();

        history.addMessage(message);

        List<ChatCompletionMessageToolCall> toolCalls =
message.toolCalls().orElse(List.of());

        if (toolCalls.isEmpty()) {
            System.out.println(message.content().orElse(""));
            return;
        }

        logger.info("模型请求调用 {} 个工具: {}",
                toolCalls.size(),
                toolCalls.stream()
                        .map(tc -> tc.asFunction().function().name())
                        .toList());

        for (ChatCompletionMessageToolCall toolCall : toolCalls) {
            var fn = toolCall.asFunction();
            Object toolCallResult = call(fn);

            var param = ChatCompletionToolMessageParam
                    .builder()
                    .toolCallId(fn.id())
```

```
                    .contentAsJson(toolCallResult)
                    .build();

            history.addMessage(param);

        }


        ChatCompletion completion = client.chat().completions().create(history.build());
        Optional<String> content = completion.choices().getFirst().message().content();
        System.out.println(content.orElse(""));

        client.close();
    }

    private static Object call(ChatCompletionMessageFunctionToolCall fn) {

        var function = fn.function();

        Class<? extends Tool> toolClass = ToolTable.TOOLS.get(function.name());

        // 模型幻觉
        if (toolClass == null) {
            logger.warn("未注册的工具: {}, 参数: {}", function.name(),
function.arguments());
            return "错误:工具 " + function.name() + " 不存在";
        }

        Object execute = function.arguments(toolClass).execute();

        logger.info("工具执行: {} | 参数: {} | 结果: {}",
                function.name(), function.arguments(), execute);

        return execute;
    }

}
```

3.6 Response Api

```
Response response = client.responses().create(params)
```

- [ ] openai 主推新 API / ChatCompletion API 主要兼容第三方大模型

- OpenAi.java

```java
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.models.ChatModel;
import com.openai.models.responses.Response;
import com.openai.models.responses.ResponseCreateParams;
import com.openai.models.responses.ResponseOutputText;

public class OpenAi {

    static String baseUrl = "";
    static String apiKey = "";

    static OpenAIClient client = OpenAIOkHttpClient
            .builder()
            .baseUrl(baseUrl)
            .apiKey(apiKey)
            .build();

    static void main() {

        ResponseCreateParams params = ResponseCreateParams
                .builder()
                .model(ChatModel.GPT_5_6_LUNA)
                .instructions("你是一个简洁的中文助手")
                .input("你好")
                .build();

        Response response = client.responses().create(params);

        System.out.println(textOf(response));

    }

    static String textOf(Response response){

        return
        response.output().stream()
                .flatMap(item -> item.message().stream())
                .flatMap(message -> message.content().stream())
                .flatMap(content -> content.outputText().stream())
                .map(ResponseOutputText::text)
                .reduce("", String::concat);

    }
```

```
}
```

3.7 Jtokkit

> Token 统计

- pom

```
<dependency>
    <groupId>com.knuddels</groupId>
    <artifactId>jtokkit</artifactId>
    <version>1.1.0</version>
</dependency>
```

- CountTokens.java

```
package org.example;

import com.knuddels.jtokkit.Encodings;
import com.knuddels.jtokkit.api.Encoding;
import com.knuddels.jtokkit.api.EncodingType;

public class CountTokens {
    static void main() {

        // cl100k_base
        // 直观感受：英文常见单词大多是 1 个 token、一个汉字通常是 1~2 个 token

        Encoding encoding = Encodings
                .newDefaultEncodingRegistry()
                .getEncoding(EncodingType.CL100K_BASE);

        int counted = encoding.countTokens("千里之行始于足下");
        // 10
```

```
            System.out.println(counted);

            // 2
            System.out.println(encoding.countTokens("hello world"));
        }
    }
```

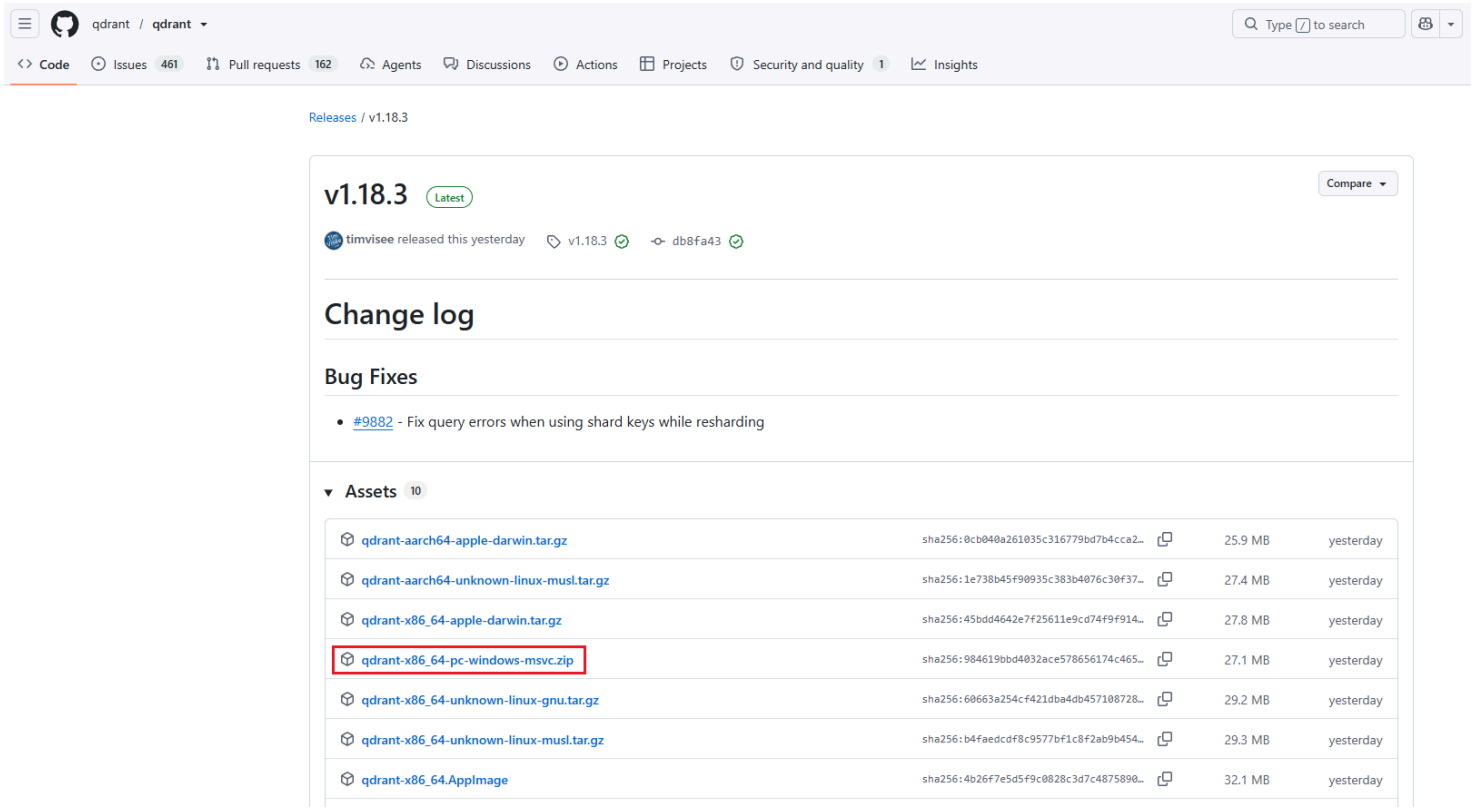
04. Qdrant

> https://github.com/qdrant/qdrant [向量数据库]

4.1 下载运行

> https://github.com/qdrant/qdrant/releases/tag/v1.18.3

- ui



> qdrant.exe 双击即可运行

- ui

- console

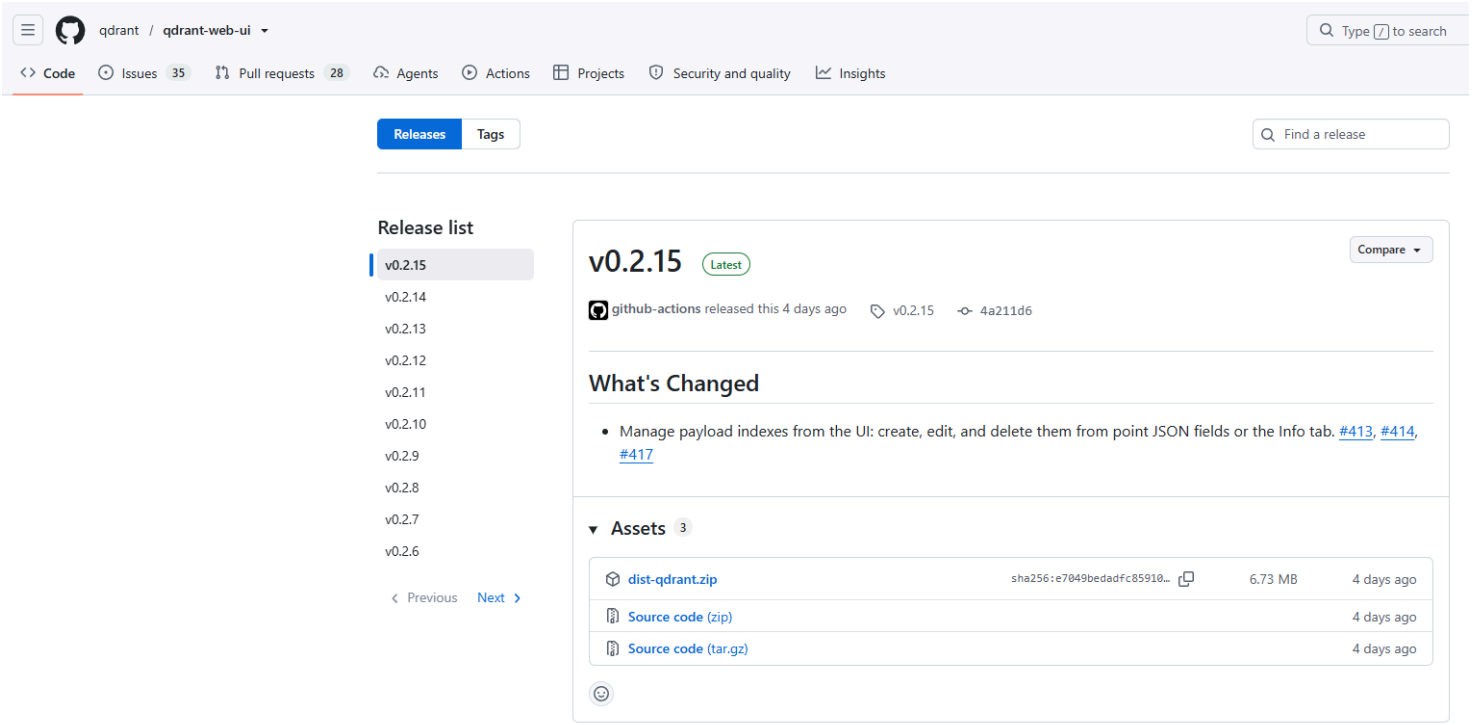
```
·[1;31m           _                 _     ·[0m
·[1;31m  __ _  __| |_ __ __ _ _ __ | |_  ·[0m
·[1;31m / _` |/ _` | '__/ _` | '_ \| __| ·[0m
·[1;31m| (_| | (_| | | | (_| | | | | |_  ·[0m
·[1;31m \__, |\__,_|_|  \__,_|_| |_|\__| ·[0m
·[1;31m    |_|                          ·[0m

·[32mVersion:·[0m ·[1;34m1.18.2·[0m, ·[32mbuild:·[0m ·[1;34m44ad62f8·[0m
·[32mAccess web UI at·[0m ·[1;4;34mhttp://localhost:6333/dashboard·[0m
·[32mAgentic Skills:·[0m ·[1;4;34mhttps://skills.qdrant.tech/·[0m

·[2m2026-07-18T16:12:35.049092Z·[0m ·[33m WARN·[0m ·[2mqdrant::settings·[0m·[2m:·[0m
Config file not found: config/config

...

·[2m2026-07-18T16:12:35.068434Z·[0m ·[32m INFO·[0m ·[2mqdrant::actix·[0m·[2m:·[0m Qdrant
HTTP listening on 6333
·[2m2026-07-18T16:12:35.068539Z·[0m ·[32m INFO·[0m ·[2mactix_server::builder·[0m·[2m:·[0m
starting 15 workers
·[2m2026-07-18T16:12:35.068684Z·[0m ·[32m INFO·[0m ·[2mactix_server::server·[0m·[2m:·[0m
Actix runtime found; starting in Actix runtime
```

```
•[2m2026-07-18T16:12:35.068866Z•[0m •[32m INFO•[0m •[2mactix_server::server•[0m•[2m:•[0m
starting service: "actix-web-service-0.0.0.0:6333", workers: 15, listening on:
0.0.0.0:6333
•[2m2026-07-18T16:12:35.074075Z•[0m •[32m INFO•[0m •[2mqdrant::tonic•[0m•[2m:•[0m Qdrant
gRPC listening on 6334
•[2m2026-07-18T16:12:35.074169Z•[0m •[32m INFO•[0m •[2mqdrant::tonic•[0m•[2m:•[0m TLS
disabled for gRPC API
```

4.2 Dashboard

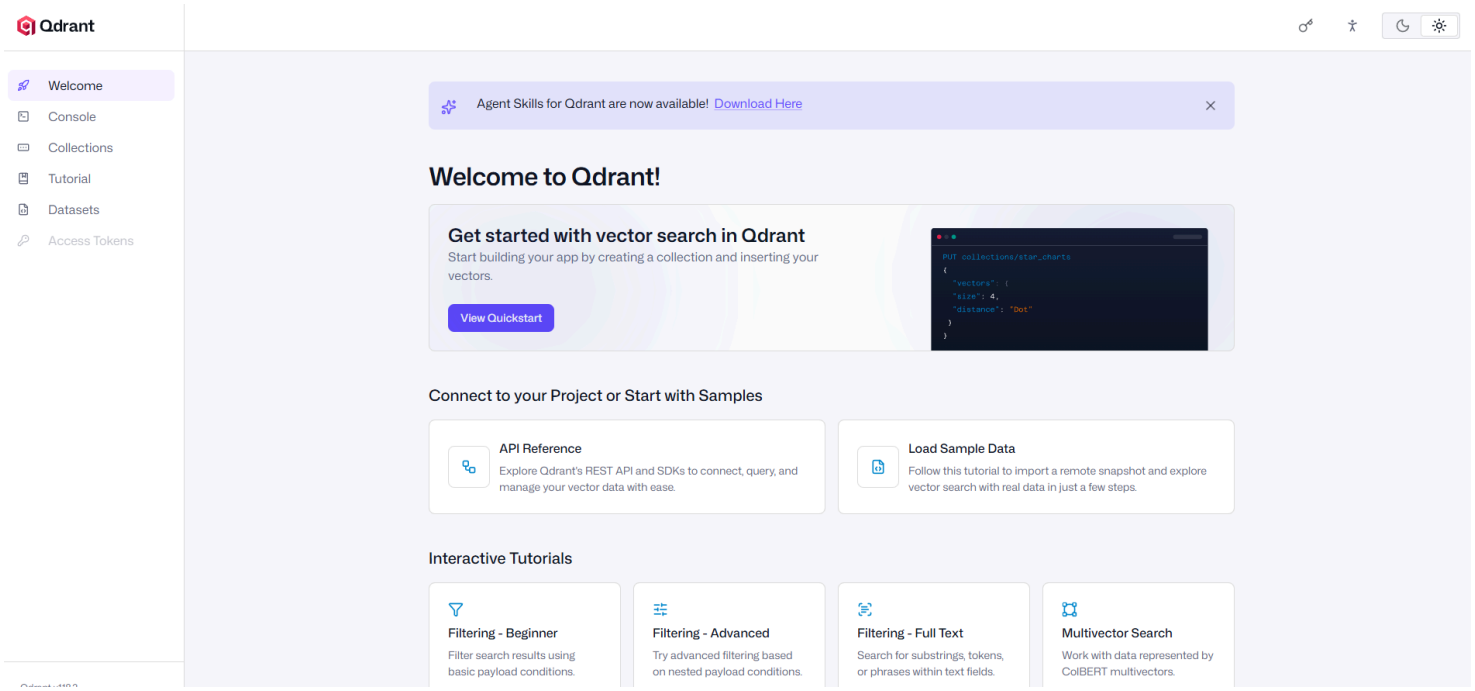
> https://github.com/qdrant/qdrant-web-ui/releases

- [ ] 解压后将 dist 目录中的资源 移动到 qdrant/static/ 下 [没有 static 就手动创建]
- ui



> http://localhost:6333/dashboard#/welcome

- ui



4.3 Dsl

> Domain-Specific Language

4.3.1 集合操作

```
PUT collections/ai
{
    "vectors": { "size": 4, "distance": "Cosine" }
}

DELETE collections/ai
```

4.3.2 插入数据

```
PUT collections/ai/points
{
    "points":
    [
        {
            "id": 1,
            "vector": [0.1, 0.2, 0.3, 0.4],
            "payload": { "doc_content": "千里之行始于足下", "source": "手册.md" }
        },
        {
            "id": 2,
            "vector": [0.2, 0.1, 0.9, 0.7],
            "payload": { "doc_content": "万事开头难", "source": "手册.md" }
        },
        {
            "id": 3,
            "vector": [0.9, 0.8, 0.1, 0.2],
            "payload": { "doc_content": "今天股市大跌", "source": "新闻.md" }
        }
    ]
}
```

4.3.3 检索全部数据

```
POST collections/ai/points/scroll
{
    "limit": 10,
    "with_payload": true
}
```

4.3.4 ID操作

```
POST collections/ai/points
{
    "ids": [1, 2],
    "with_payload": true,
    "with_vector": true
}

// ID删除

POST collections/ai/points/delete
{ "points": [3] }
```

4.3.5 相似度检索

```
POST collections/ai/points/search
{
    "vector": [0.1, 0.2, 0.3, 0.5],
    "limit": 2,
    "with_payload": true
}
```

4.3.6 条件检索

```
POST collections/ai/points/search
{
    "vector": [0.1, 0.2, 0.3, 0.5],
    "filter": { "must": [{ "key": "source", "match": { "value": "手册.md" } }] },
    "limit": 2,
    "with_payload": true
}
```

4.3.7 统计

```
POST collections/ai/points/count
```

4.3.8 余弦相似度

> 直观感受：两个向量从原点发起的箭头不论长度、夹角越小越相似

☐ 夹角为 0° 极其相似 | 90° 内容毫无关系 | 180° 语义相反

```
******************** 计算规则 ********************

A = [3, 4]
B = [4, 3]

A     ⟹ 3×4 = 12
B     ⟹ 4×3 = 12

A + B ⟹ 12 + 12 = 24

A     ⟹ 3×3 + 4×4 = 9+16 = 25 → sqrt 5
B     ⟹ 4×4 + 3×3 = 16+9 = 25 → sqrt 5

24 ÷ (5×5) = 0.96 → [0, 1] 越接近 1 越相似

******************** 计算规则 ********************
```

4.4 qdrant/java-client

> https://github.com/qdrant/java-client

- pom

```
<dependency>
    <groupId>io.qdrant</groupId>
    <artifactId>client</artifactId>
    <version>1.18.3</version>
</dependency>
```

4.4.1 QdrantClient

> 6333 是 HTTP 端口 | 6334 是 grpc 端口

- QdrantUsage.java

```
package org.example;

import io.qdrant.client.QdrantClient;
import io.qdrant.client.QdrantGrpcClient;
import io.qdrant.client.grpc.Collections;

import java.util.concurrent.ExecutionException;

import static io.qdrant.client.grpc.Collections.VectorParams;

public class QdrantUsage {
    static void main() throws ExecutionException, InterruptedException {
        QdrantClient client = new QdrantClient(QdrantGrpcClient
                .newBuilder("127.0.0.1", 6334, false)
                .build()
```

```
        );

        Collections.CollectionOperationResponse response =
            client.createCollectionAsync("ai", VectorParams
                        .newBuilder()
                        .setSize(4)
                        .setDistance(Collections.Distance.Cosine)
                        .build())
                .get();
        System.out.println(response.getResult());

    }
}
```

4.4.2 upsertAsync

> io.qdrant.client.grpc.Points.PointStruct 构建插入数据结构

- QdrantUsage.java

```
package org.example;

import io.qdrant.client.QdrantClient;
import io.qdrant.client.QdrantGrpcClient;

import java.util.List;
import java.util.Map;

import static io.qdrant.client.PointIdFactory.id;
import static io.qdrant.client.ValueFactory.value;
import static io.qdrant.client.VectorsFactory.vectors;
import static io.qdrant.client.grpc.Points.PointStruct;

public class QdrantUsage {
    static void main() {
        QdrantClient client = new QdrantClient(QdrantGrpcClient
                .newBuilder("127.0.0.1", 6334, false)
                .build()
        );

        client.upsertAsync("ai", List.of(
                PointStruct.newBuilder()
```

```
                        .setId(id(1))
                        .setVectors(vectors(0.1f, 0.2f, 0.3f, 0.4f))
                        .putAllPayload(Map.of(
                                "doc_content", value("千里之行始于足下"),
                                "source", value("手册.md")
                        ))
                        .build(),
                PointStruct.newBuilder()
                        .setId(id(2))
                        .setVectors(vectors(0.2f, 0.1f, 0.9f, 0.7f))
                        .putAllPayload(Map.of(
                                "doc_content", value("万事开头难"),
                                "source", value("手册.md")
                        ))
                        .build()
        ));
    }
}
```

4.4.3 nearest()

> io.qdrant.client.QueryFactory.nearest 相似度检索

- [ ] 检索向量数据库不需要操作向量数组、也无需将向量数组返回到应用程序

- QdrantUsage.java

```
package org.example;

import io.qdrant.client.QdrantClient;
import io.qdrant.client.QdrantGrpcClient;
import io.qdrant.client.grpc.Points;

import java.util.List;
import java.util.concurrent.ExecutionException;

import static io.qdrant.client.QueryFactory.nearest;
import static io.qdrant.client.WithPayloadSelectorFactory.enable;

public class QdrantUsage {
```

```
    static void main() throws ExecutionException, InterruptedException {
        QdrantClient client = new QdrantClient(QdrantGrpcClient
                .newBuilder("127.0.0.1", 6334, false)
                .build()
        );

        List<Points.ScoredPoint> scoredPoints =
client.queryAsync(Points.QueryPoints.newBuilder()
                .setCollectionName("ai")
                .setQuery(nearest(0.1f, 0.2f, 0.3f, 0.4f))
                .setLimit(5)
                .setWithPayload(enable(true))
                .build()
        ).get();

        for (Points.ScoredPoint p : scoredPoints) {
            System.out.printf("id=%d score=%.4f payload=%s%n",
                    p.getId().getNum(),
                    p.getScore(),
                    p.getPayloadMap().get("doc_content").getStringValue());
        }

    }
}
```

4.5 text-embedding-v4

> 向量模型是将数据转为向量数组的过程、向量维度在使用时通知模型固定

- TexEmbeddingV4.java

```
package org.example;

import com.openai.client.OpenAIClient;
import com.openai.client.okhttp.OpenAIOkHttpClient;
import com.openai.models.embeddings.CreateEmbeddingResponse;
import com.openai.models.embeddings.EmbeddingCreateParams;

import java.util.List;
```

```
public class TexEmbeddingV4 {
    static void main() {

        String baseUrl = "https://ws-h4qjfke1wt127m1g.cn-
beijing.maas.aliyuncs.com/compatible-mode/v1";

        OpenAIClient client = OpenAIOkHttpClient
                .builder()
                .baseUrl(baseUrl)
                .apiKey(System.getenv("TEXT-EMBEDDING-V4-API-KEY"))
                .build();

        EmbeddingCreateParams params = EmbeddingCreateParams
                .builder()
                .model("text-embedding-v4")
                .encodingFormat(EmbeddingCreateParams.EncodingFormat.FLOAT)
                .dimensions(1024)
                .input("千里之行始于足下")
                .build();

        CreateEmbeddingResponse response = client.embeddings().create(params);
        List<Float> embedding = response.data().getFirst().embedding();
        System.out.println(embedding);
    }
}

// 备注：最终存储 { id、vector、payload } [vector 通过将内容拿去请求向量模型得到]
```

4.6 spring-ai-reader

> spring-ai-x-document-reader

4.6.1 ai-markdown

> spring-ai-markdown-document-reader

- pom

```xml
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-markdown-document-reader</artifactId>
    <version>2.0.0</version>
    <scope>compile</scope>
</dependency>
```

- MarkdownReader.java

```java
package org.example;

import org.springframework.ai.document.Document;
import org.springframework.ai.reader.markdown.MarkdownDocumentReader;
import org.springframework.ai.reader.markdown.config.MarkdownDocumentReaderConfig;
import org.springframework.ai.transformer.splitter.TokenTextSplitter;

import java.util.List;

public class MarkdownReader {
    static void main() {
        MarkdownDocumentReaderConfig config = MarkdownDocumentReaderConfig
                .builder()
                // 分割线中的内容独立为一个文档
                .withHorizontalRuleCreateDocument(true)
                // 代码块独立为一个文档
                .withIncludeCodeBlock(false)
                // 引用块独立为一个文档
                .withIncludeBlockquote(false)
                // 附加元信息
                .withAdditionalMetadata("filename", "doc.md")
                .build();
        MarkdownDocumentReader reader =
                new MarkdownDocumentReader("classpath:handbook.md", config);

        List<Document> documents = reader.read();

        // for (Document document : documents) {
        //     System.out.println(document.getText());
        //     System.out.println();
        // }

        TokenTextSplitter splitter =
```

```
                TokenTextSplitter.builder().withChunkSize(800).build();

        // 按照语义切块 同上 结构切块不冲突
        List<Document> transform = splitter.transform(documents);

    }
}
```

4.6.2 ai-pdf

> spring-ai-pdf-document-reader

- pom

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-pdf-document-reader</artifactId>
    <version>2.0.0</version>
    <scope>compile</scope>
</dependency>
```

- PDFReader.java

```
package org.example;

import org.springframework.ai.document.Document;
import org.springframework.ai.reader.ExtractedTextFormatter;
import org.springframework.ai.reader.pdf.PagePdfDocumentReader;
import org.springframework.ai.reader.pdf.ParagraphPdfDocumentReader;
import org.springframework.ai.reader.pdf.config.PdfDocumentReaderConfig;

import java.util.List;
```

```
public class PDFReader {
    static void main() {
        PdfDocumentReaderConfig config = PdfDocumentReaderConfig
                .builder()
                // 一页一个文档
                .withPagesPerDocument(1)
                .withPageExtractedTextFormatter(ExtractedTextFormatter.builder()
                        .withNumberOfTopTextLinesToDelete(0)  // 可去页眉页脚行
                        .build())
                .build();

        PagePdfDocumentReader reader =
            new PagePdfDocumentReader("classpath:spring-ai.pdf", config);

        List<Document> documents = reader.read();

        // for (Document document : documents) {
        //     System.out.println(document.getText());
        // }

        // 按目录章节
        ParagraphPdfDocumentReader pdfReader =
                new ParagraphPdfDocumentReader("classpath:spring-ai.pdf");

        List<Document> documentList = pdfReader.read();
        for (Document document : documentList) {
            System.out.println(document.getText());
        }

    }
}
```

4.6.3 ai-tika

> PDF | DOC/DOCX | PPT/PPTX | HTML

- pom

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tika-document-reader</artifactId>
    <version>2.0.0</version>
    <scope>compile</scope>
</dependency>
```

- TikaReader.java

```
package org.example;

import org.springframework.ai.document.Document;
import org.springframework.ai.reader.tika.TikaDocumentReader;

import java.util.List;

public class TikaReader {
    static void main() {
        TikaDocumentReader reader = new TikaDocumentReader("classpath:tika.docx");

        List<Document> documents = reader.read();

        for (Document document : documents) {
            System.out.println(document.getText());
        }
    }
}
```

05. Spring AI Api

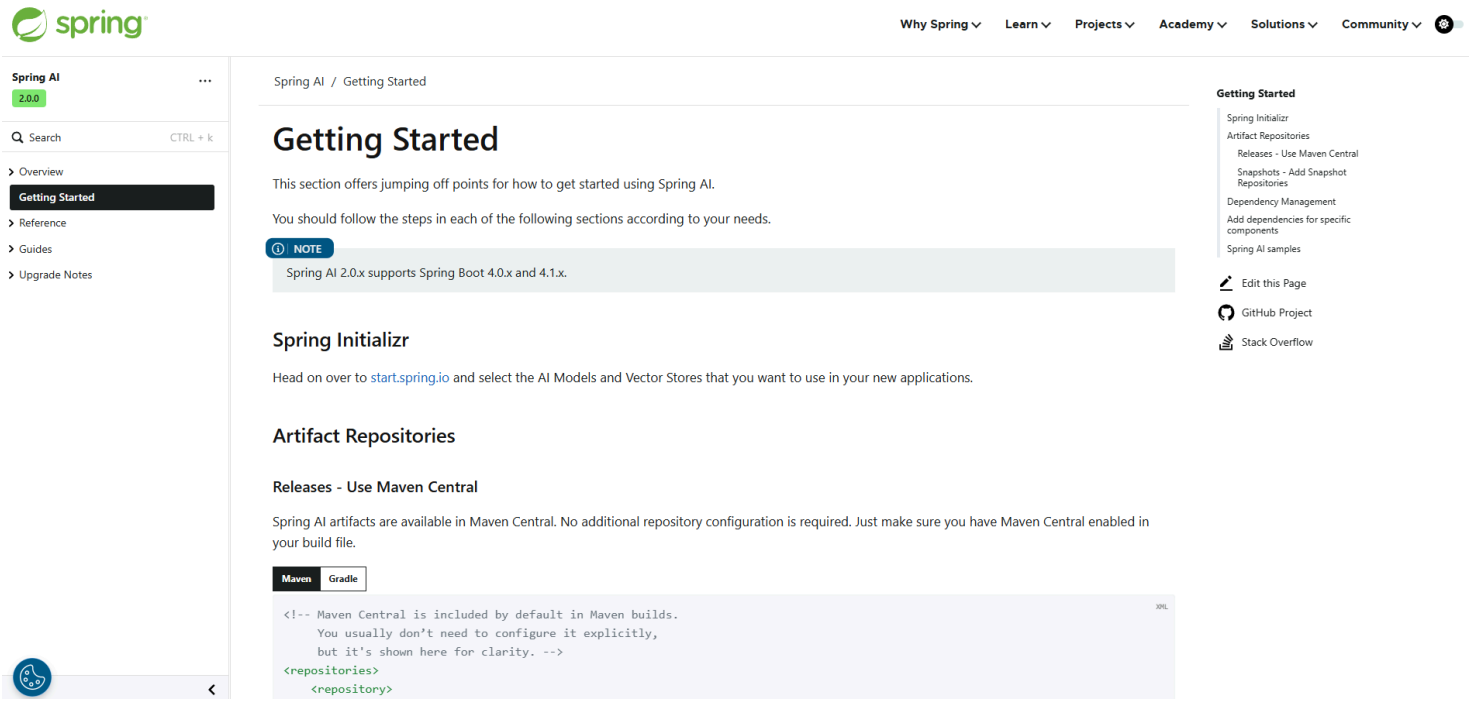
> https://spring.io/projects/spring-ai#learn

- [ ] Spring AI 2.0.x supports Spring Boot 4.0.x and 4.1.x

5.1 官方文档

> https://docs.spring.io/spring-ai/reference/index.html

- ui



5.2 pom

> https://docs.spring.io/spring-ai/reference/getting-started.html

- pom

```
<dependencyManagement>
    <dependencies>
        <dependency>
```

```
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-bom</artifactId>
            <version>2.0.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>


<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-commons</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-template-st</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-model</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-vector-store</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-rag</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-vector-store-advisor</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tool-search-advisor</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tool-search-tool-advisor</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tool-search-tool</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-client-chat</artifactId>
</dependency>

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

5.3 会话

> OpenAiChatModel | ChatClient

- pom

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-bom</artifactId>
            <version>2.0.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-openai</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-client-chat</artifactId>
</dependency>

<dependency>
    <groupId>tools.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>3.2.1</version>
```

```
    <scope>compile</scope>
</dependency>
```

5.3.1 OpenAiChatModel

> org.springframework.ai.openai.OpenAiChatModel 封装对应模型

- SpringAi.java

```
package org.example;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;

import java.util.Scanner;

public class SpringAi {
    static void main() {

        String baseUrl = "https://api.deepseek.com";
        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl(baseUrl)
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .model("deepseek-v4-flash")
                        .temperature(0.7)
                        .build()
                )
                .build();

        String systemPrompt = "你是一个简洁的中文助手";

        ChatClient client = ChatClient
```

```
                .builder(model)
                .defaultSystem(systemPrompt)
                .build();

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("你: ");
            String prompt = scanner.nextLine();
            if (prompt.isBlank() || prompt.equals("quit")) {
                return;
            }

            String content = client.prompt()
                    .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID, "some-
id"))
                    .user(prompt)
                    .call()
                    .content();

            System.out.print("AI: ");
            System.out.println(content);

        }

    }
}
```

5.3.2 提示词模版

> PromptTemplate 可构建提示词模版 也可创建 Prompt 对象

- SpringAiPromptTemplate.java

```
package org.example;

import org.springframework.ai.chat.prompt.Prompt;
import org.springframework.ai.chat.prompt.PromptTemplate;
import org.springframework.ai.template.st.StTemplateRenderer;

import java.util.Map;
```

```java
public class SpringAiPromptTemplate {
    static void main() {

        // {}
        PromptTemplate template = new PromptTemplate("编写一首{topic}唐诗");
        String render = template.render(Map.of("topic", "怀古"));
        System.out.println(render);

        // <>
        PromptTemplate promptTemplate = PromptTemplate
                .builder()
                .renderer(StTemplateRenderer
                        .builder()
                        .startDelimiterToken('<')
                        .endDelimiterToken('>')
                        .build())
                .template("编写一首<topic>唐诗")
                .build();

        System.out.println(promptTemplate.render(Map.of("topic", "悲伤")));

        Prompt prompt = promptTemplate.create(Map.of("topic", "悲伤"));


        System.out.println(build.getContents());

    }
}
```

```
=== client.prompt().user(spec -> spec.text("提示词模版"))
```

- SpringAiPromptTemplate.java

```java
package org.example;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;

public class SpringAiPromptTemplate {
    static void main() {

        OpenAiChatModel model = OpenAiChatModel
                .builder()
```

```
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl("https://api.deepseek.com")
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .temperature(0.7)
                        .model("deepseek-v4-flash")
                        .build())
                .build();

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem("你是一个简洁的中文助手")
                .build();

        String content = client
                .prompt()
                .user(spec -> spec.text("创建一首{topic}诗词").param("topic", "悲伤"))
                .call()
                .content();

        System.out.println(content);
    }
}
```

5.3.3 上下文

> MessageWindowChatMemory 自动维护会话历史

- SpringAi.java

```
package org.example;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.MessageChatMemoryAdvisor;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.ai.chat.memory.MessageWindowChatMemory;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;
```

```
import java.util.Scanner;

public class SpringAi {
    static void main() {

        String baseUrl = "https://api.deepseek.com";

        MessageWindowChatMemory memory = MessageWindowChatMemory
                .builder()
                .maxMessages(20)
                // .chatMemoryRepository(jdbc | redis | mongodb)
                .build();

        OpenAiChatModel model = ...

        String systemPrompt = "你是一个简洁的中文助手";

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem(systemPrompt)
                .defaultAdvisors(MessageChatMemoryAdvisor.builder(memory).build())
                .build();

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("你: ");
            String prompt = scanner.nextLine();
            if (prompt.isBlank() || prompt.equals("quit")) {
                return;
            }

            String content = client.prompt()
                    .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID, "some-
id"))
                    .user(prompt)
                    .call()
                    .content();

            System.out.print("AI: ");
            System.out.println(content);

        }

    }
}
```

5.3.4 流式响应

```
package org.example;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;

public class SpringAi {
    static void main() {

        String baseUrl = "https://api.deepseek.com";


        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl(baseUrl)
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .model("deepseek-v4-flash")
                        .temperature(0.7)
                        .build()
                )
                .build();

        String systemPrompt = "你是一个简洁的中文助手";

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem(systemPrompt)
                .build();


        client.prompt()
                .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID, "some-
id"))
                .user("详细介绍一下 spring ai")
                .stream()
                .content()
                .doOnNext(System.out::print)
                .blockLast();
    }
}
```

5.3.5 结构化输出

> AdvisorParams.ENABLE_NATIVE_STRUCTURED_OUTPUT 强制开启 response_format

☐ spring ai 会将 entity 信息解析为 schema 附加在请求提示词上

- SpringAi.java

```
package org.example;

import org.example.entity.User;
import org.springframework.ai.chat.client.AdvisorParams;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.api.Advisor;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;
import org.springframework.core.ParameterizedTypeReference;

import java.util.List;

public class SpringAi {
    static void main() {

        String baseUrl = "https://api.deepseek.com";

        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl(baseUrl)
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .model("deepseek-v4-flash")
                        .temperature(0.7)
                        .build()
                )
                .build();

        String systemPrompt = "你是一个简洁的中文助手";

        ChatClient client = ChatClient
                .builder(model)
                // .defaultAdvisors(AdvisorParams.ENABLE_NATIVE_STRUCTURED_OUTPUT)
                .defaultSystem(systemPrompt)
```

```
                .build();


        // User user = client.prompt()
        //         .advisors(advisor -> advisor.param(ChatMemory.CONVERSATION_ID, "some-id"))
        //         .user("张三23岁了居住在广州")
        //         .call()
        //         .entity(User.class);
        //
        // System.out.println(user);

        List<User> userList = client.prompt()
                .advisors(advisor → advisor.param(ChatMemory.CONVERSATION_ID, "some-
id"))
                .user("张三23岁了居住在广州，李四在北京已经生活了35年了")
                .call()
                .entity(new ParameterizedTypeReference<List<User>>() {
                });

        userList.forEach(System.out::println);
    }
}
```

5.3.6 工具

> == @Tool 描述工具函数 | @ToolParam 描述工具函数参数

- DatetimeTools.java

```
package org.example.tools;

import org.springframework.ai.tool.annotation.Tool;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class DatetimeTools {

    static final DateTimeFormatter FORMATTER =
            DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    @Tool(
```

```
            description = "获取当前的系统日期和时间,格式为 yyyy-MM-dd HH:mm:ss",
            name = "get_current_time"
    )
    String getCurrentTime() {
        return LocalDateTime.now().format(FORMATTER);
    }

}
```

- WeatherTools.java

```
package org.example.tools;

import org.springframework.ai.tool.annotation.Tool;
import org.springframework.ai.tool.annotation.ToolParam;

public class WeatherTools {

    @Tool(
            description = "查询指定城市当前的实时天气,包括温度、天气状况和风力",
            name = "get_weather",
            returnDirect = true)
    String getWeather(@ToolParam(description = "城市名称,使用中文,不带省份和'市'字,例如:北京、上海")
                      String city) {
        return city + "28°C, 晴, 微风";
    }
}
```

- SpringAi.java

```
package org.example;

import org.example.tools.DatetimeTools;
import org.example.tools.WeatherTools;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;

public class SpringAi {
```

```
    static void main() {

        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl("https://api.deepseek.com")
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .temperature(0.7)
                        .model("deepseek-v4-flash")
                        .build())
                .build();

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem("你是一个简洁的中文助手")
                .defaultTools(new DatetimeTools(), new WeatherTools())
                .build();

        String content = client.prompt().user("广州天气").call().content();

        System.out.println(content);

    }
}
```

5.3.7 工具拦截

> org.springframework.ai.tool.ToolCallback

- InterceptingToolCallback.java

```
package org.example.tools;

import org.springframework.ai.tool.ToolCallback;
import org.springframework.ai.tool.definition.ToolDefinition;

public class InterceptingToolCallback implements ToolCallback {

    private final ToolCallback delegate;

    public InterceptingToolCallback(ToolCallback delegate) {
```

```
        this.delegate = delegate;
    }

    @Override
    public ToolDefinition getToolDefinition() {
        return this.delegate.getToolDefinition();
    }

    @Override
    public String call(String toolInput) {

        // boolean returnDirect = delegate.getToolMetadata().returnDirect();
        ToolDefinition toolDefinition = delegate.getToolDefinition();

        String toolName = toolDefinition.name();

        System.out.println("toolName = " + toolName);

        System.out.println("before ...");

        String called = delegate.call(toolInput);

        System.out.println("after ...");

        return called;
    }
}
```

- SpringAi.java

```
package org.example;

import org.example.tools.DatetimeTools;
import org.example.tools.InterceptingToolCallback;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;
import org.springframework.ai.support.ToolCallbacks;

public class SpringAi {
    static void main() {

        InterceptingToolCallback callback =
                new InterceptingToolCallback(ToolCallbacks.from(new DatetimeTools())[0]);

        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
```

```
                        .builder()
                        .baseUrl("https://api.deepseek.com")
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .temperature(0.7)
                        .model("deepseek-v4-flash")
                        .build())
                .build();

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem("你是一个简洁的中文助手")
                .defaultTools(callback)
                .build();

        String content = client.prompt().user("几点了").call().content();

        System.out.println(content);

    }
}
```

5.3.8 OpenAiEmbeddingModel

> org.springframework.ai.openai.OpenAiEmbeddingModel

- SpringAi.java

```
package org.example;

import org.springframework.ai.openai.OpenAiEmbeddingModel;
import org.springframework.ai.openai.OpenAiEmbeddingOptions;

import java.util.Arrays;

public class SpringAi {
    static void main() {

        String baseUrl =
```

```
            "https://ws-h4qjfke1wt127m1g.cn-beijing.maas.aliyuncs.com/compatible-
mode/v1";

        OpenAiEmbeddingModel model = OpenAiEmbeddingModel
                .builder()
                .options(OpenAiEmbeddingOptions
                        .builder()
                        .baseUrl(baseUrl)
                        .apiKey(System.getenv("TEXT-EMBEDDING-V4-API-KEY"))
                        .model("text-embedding-v4")
                        .dimensions(1024)
                        .encodingFormat(OpenAiEmbeddingOptions.EncodingFormat.FLOAT)
                        .build()
                )
                .build();

        float[] embed = model.embed("千里之行始于足下");

        System.out.println(Arrays.toString(embed));
    }
}
```

5.3.9 SimpleLoggerAdvisor

> org.springframework.ai.chat.client.advisor 请求链路调用

- pom

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.5.38</version>
    <scope>compile</scope>
</dependency>
```

- src/main/resources/logback.xml

```xml
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="org.springframework.ai.chat.client.advisor" level="DEBUG"/>

    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
    </root>
</configuration>
```

- SpringAi.java

```java
package org.example;

import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.SimpleLoggerAdvisor;
import org.springframework.ai.openai.OpenAiChatModel;
import org.springframework.ai.openai.OpenAiChatOptions;

public class SpringAi {
    static void main() {

        OpenAiChatModel model = OpenAiChatModel
                .builder()
                .options(OpenAiChatOptions
                        .builder()
                        .baseUrl("https://api.deepseek.com")
                        .apiKey(System.getenv("DEEPSEEK-API-KEY"))
                        .temperature(0.7)
                        .model("deepseek-v4-flash")
                        .build())
                .build();

        ChatClient client = ChatClient
                .builder(model)
                .defaultSystem("你是一个简洁的中文助手")
                .defaultAdvisors(new SimpleLoggerAdvisor())
                .build();
```

```
        String content = client.prompt().user("广州天气怎么样？").call().content();

        System.out.println(content);

    }
}
```

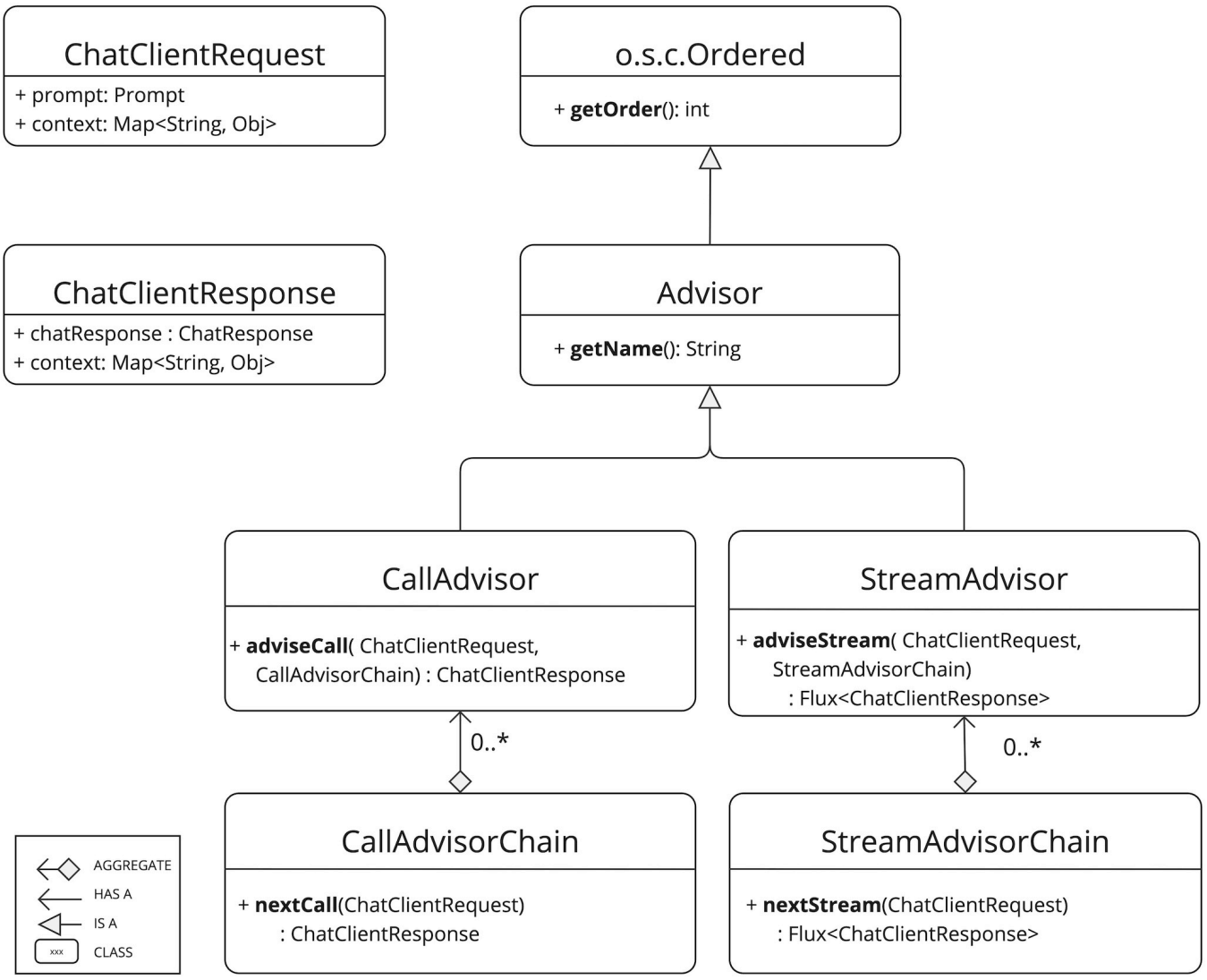
5.4 Advisors

> 拦截整个请求：从提示词到达大模型 → 大模型响应内容

- [ ] 整个拦截链被 order 排序、值越小、则 advisor 优先级越高
- [ ] 低于 ToolCallingAdvisor 的优先级、将被 for [原生 sdk 工具调用机制]

5.4.1 核心组件

- ui



5.4.2 CallAdvisor

自定义实现示例

- ChainInspectorAdvisor.java

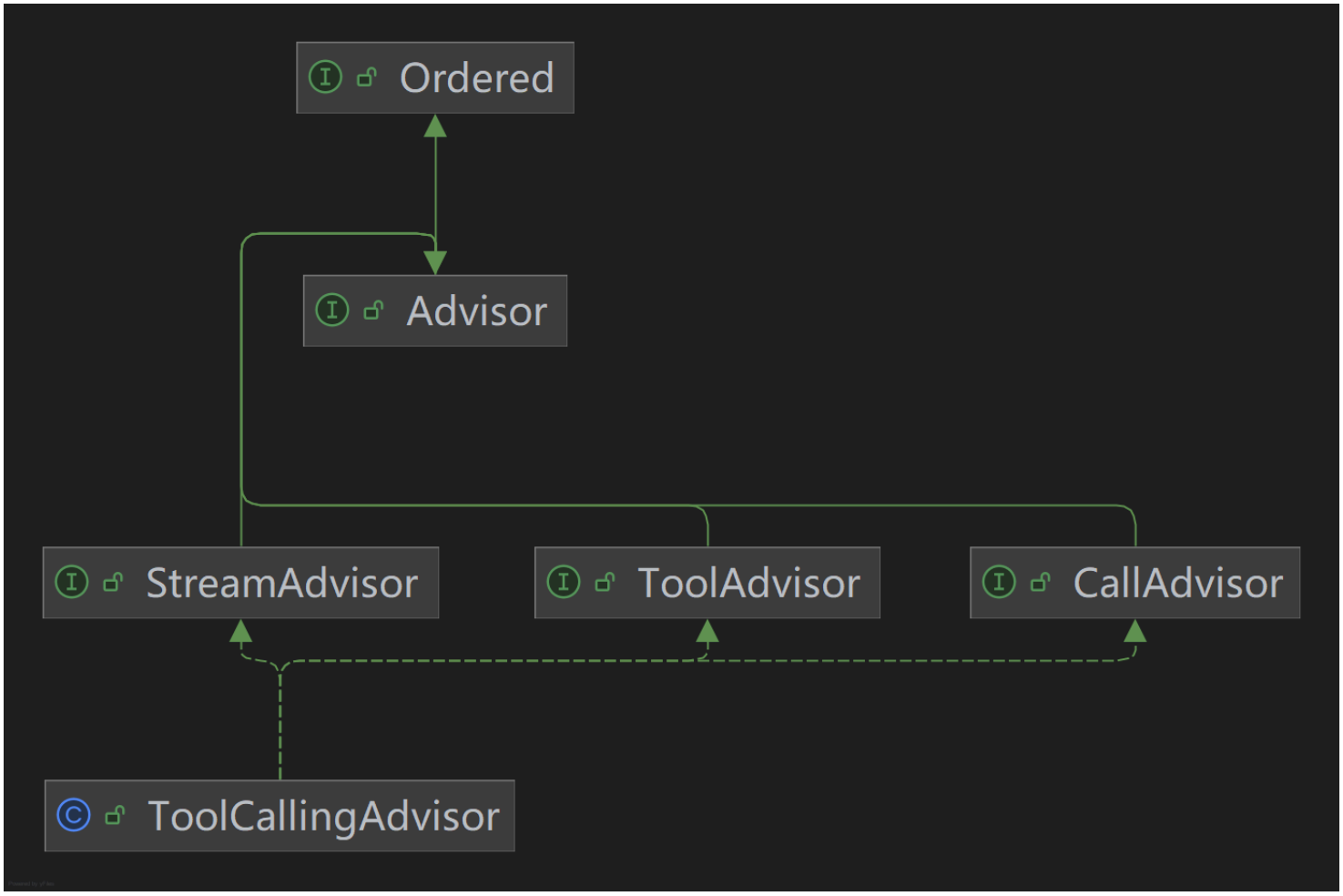
```java
package org.example.advisors;

import org.springframework.ai.chat.client.ChatClientRequest;
import org.springframework.ai.chat.client.ChatClientResponse;
import org.springframework.ai.chat.client.advisor.api.CallAdvisor;
```

```java
import org.springframework.ai.chat.client.advisor.api.CallAdvisorChain;
import org.springframework.core.Ordered;

import java.util.List;

public class ChainInspectorAdvisor implements CallAdvisor {

    @Override
    public ChatClientResponse adviseCall(ChatClientRequest chatClientRequest,
                                         CallAdvisorChain callAdvisorChain) {

        System.out.println("====================================");

        List<CallAdvisor> callAdvisors = callAdvisorChain.getCallAdvisors();
        for (CallAdvisor callAdvisor : callAdvisors) {
            System.out.println(callAdvisor.getName() + ", "
                               + callAdvisor.getClass().getName());
        }

        System.out.println("====================================");

        return callAdvisorChain.nextCall(chatClientRequest);
    }

    @Override
    public String getName() {
        return this.getClass().getSimpleName();
    }

    @Override
    public int getOrder() {
        return Ordered.HIGHEST_PRECEDENCE;
        // return 0;
    }
}
```

5.4.3 ToolCallingAdvisor

> 负责工具循环调用、自定义提供则使内部默认无效、工具循环只能有一个实现

- ui



- 执行流程

```
A_Advisor (order MIN+100)                          ── 循环外,执行 1 次
   │ req ①
   │ next()
   ▼
B_Advisor (order MIN+200)                          ── 循环外,执行 1 次
   │ req ②
   │ next()
   ▼
ToolCallingAdvisor (order MIN+300)
   │ ┌──────────────── 循环开始 ────────────────┐
   │ │ next()                                   │
   │ │   ▼                                      │
   │ │ C_Advisor (order 0)                      │ ── 循环内,每轮执行
   │ │    │ req ③                               │
   │ │    │ next()                              │
   │ │    │   ▼                                 │
   │ │    │ ChatModelCallAdvisor → 模型 ④       │
   │ │    │ res ⑤                               │
   │ │    ▲                                     │
   │ │ 响应有 tool call?                        │
   │ │    ├─ 有: 执行工具,拼历史,回到循环头 ─────┘
   │ │    └─ 没有: 带最终响应退出循环
```

```
  ▼
B_Advisor res ⑥                                  ── 只看到最终结果
  ▼
A_Advisor res ⑦
  ▼
调用者
```

- java

```java
class MyToolAdvisor implements ToolAdvisor {

    // 不是该你实现的

}
```

06. SpringBoot

> base starter: spring-ai-starter-model-openai

- 常用自动装配列表

```
spring-ai-autoconfigure-model-chat-client:2.0.0
spring-ai-autoconfigure-model-chat-memory:2.0.0
spring-ai-autoconfigure-model-chat-observation:2.0.0
spring-ai-autoconfigure-model-embedding-observation:2.0.0
spring-ai-autoconfigure-model-image-observation:2.0.0
spring-ai-autoconfigure-model-openai:2.0.0
spring-ai-autoconfigure-model-tool:2.0.0
spring-ai-autoconfigure-retry:2.0.0
```

6.1 会话

> spring-ai-starter-model-openai

- pom

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-model-openai</artifactId>
</dependency>
```

- application.properties

```
#spring.ai.chat.client.enabled=true
#spring.ai.chat.client.tool-calling.enabled=true
spring.ai.chat.client.tool-calling.advisor-order=100
spring.ai.openai.base-url=https://api.deepseek.com
spring.ai.openai.api-key=${DEEPSEEK-API-KEY}
spring.ai.openai.chat.model=deepseek-v4-flash
spring.ai.openai.chat.temperature=0.7
logging.level.org.springframework.ai.chat.client.advisor=DEBUG
```

6.1.1 ChatClient.Builder

> ChatClient.Builder | ChatMemory 已自动装配到容器

- ChatController.java

```java
package org.example.ai.controller;


import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.MessageChatMemoryAdvisor;
import org.springframework.ai.chat.client.advisor.SimpleLoggerAdvisor;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.core.Ordered;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ChatController {

    final ChatClient client;

    public ChatController(ChatClient.Builder builder, ChatMemory memory) {

        System.out.println(memory.getClass());

        MessageChatMemoryAdvisor advisor =
MessageChatMemoryAdvisor.builder(memory).build();

        this.client = builder
                .defaultSystem("你是一个简洁的中文助手")
                .defaultAdvisors(new SimpleLoggerAdvisor(50), advisor)
                // .defaultTools()
                .build();
    }

    @GetMapping("/ai")
    public String chat(String prompt, String username) {

        // username from in HttpSession or Token

        return client
                .prompt()
                .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID,
username))
                .user(prompt)
                .call()
                .content();
    }

}
```

spring-ai-autoconfigure-model-openai-2.0.0.jar

- [ ] spring-ai-autoconfigure-model-openai-2.0.0.jar [openai-model]
- [ ] spring-ai-autoconfigure-model-chat-client-2.0.0.jar [openai-chat-client]

- ChatMemoryAutoConfiguration.java

```java
package org.springframework.ai.model.chat.memory.autoconfigure;

import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.ai.chat.memory.ChatMemoryRepository;
import org.springframework.ai.chat.memory.InMemoryChatMemoryRepository;
import org.springframework.ai.chat.memory.MessageWindowChatMemory;
import org.springframework.boot.autoconfigure.AutoConfiguration;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;

@AutoConfiguration
@ConditionalOnClass({ChatMemory.class, ChatMemoryRepository.class})
public class ChatMemoryAutoConfiguration {
    @Bean
    @ConditionalOnMissingBean
    ChatMemoryRepository chatMemoryRepository() {
        return new InMemoryChatMemoryRepository();
    }

    @Bean
    @ConditionalOnMissingBean
    ChatMemory chatMemory(ChatMemoryRepository chatMemoryRepository) {
        return MessageWindowChatMemory
                    .builder()
                    .chatMemoryRepository(chatMemoryRepository)
                    .build();
    }
}
```

6.1.2 上下文持久化

> org.springframework.ai.chat.memory.repository.jdbc.JdbcChatMemoryRepository

- pom

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

<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

- application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/ai
spring.datasource.username=root
spring.datasource.password=root
spring.ai.chat.memory.repository.jdbc.initialize-schema=always
```

> spring-ai-autoconfigure-model-chat-memory-repository-jdbc-2.0.0.jar

- [ ] 核心逻辑: @AutoConfiguration(before = {ChatMemoryAutoConfiguration.class})

- JdbcChatMemoryRepositoryAutoConfiguration.java

```java
package org.springframework.ai.model.chat.memory.repository.jdbc.autoconfigure;

import javax.sql.DataSource;
import org.springframework.ai.chat.memory.repository.jdbc.JdbcChatMemoryRepository;
import
org.springframework.ai.chat.memory.repository.jdbc.JdbcChatMemoryRepositoryDialect;
import
org.springframework.ai.model.chat.memory.autoconfigure.ChatMemoryAutoConfiguration;
import org.springframework.boot.autoconfigure.AutoConfiguration;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.boot.sql.autoconfigure.init.OnDatabaseInitializationCondition;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Conditional;
import org.springframework.jdbc.core.JdbcTemplate;

@AutoConfiguration(
    before = {ChatMemoryAutoConfiguration.class}
)
@ConditionalOnClass({JdbcChatMemoryRepository.class,
                     DataSource.class,
                     JdbcTemplate.class})

@EnableConfigurationProperties({JdbcChatMemoryRepositoryProperties.class})
public class JdbcChatMemoryRepositoryAutoConfiguration {
    @Bean
    @ConditionalOnMissingBean
    JdbcChatMemoryRepository jdbcChatMemoryRepository(JdbcTemplate
                                                      jdbcTemplate,
                                                      DataSource dataSource) {

        JdbcChatMemoryRepositoryDialect dialect =
            JdbcChatMemoryRepositoryDialect.from(dataSource);

        return JdbcChatMemoryRepository
                    .builder()
                    .jdbcTemplate(jdbcTemplate)
                    .dialect(dialect)
                    .build();
    }

    @Bean
    @ConditionalOnMissingBean
    @Conditional({OnJdbcChatMemoryRepositoryDatasourceInitializationCondition.class})
    JdbcChatMemoryRepositorySchemaInitializer jdbcChatMemoryScriptDatabaseInitializer(
        DataSource dataSource,
        JdbcChatMemoryRepositoryProperties properties) {

        return new JdbcChatMemoryRepositorySchemaInitializer(dataSource, properties);
```

```
    }

    static class OnJdbcChatMemoryRepositoryDatasourceInitializationCondition
        extends OnDatabaseInitializationCondition {

        OnJdbcChatMemoryRepositoryDatasourceInitializationCondition() {
            super("Jdbc Chat Memory Repository",
                  new String[]
                  {"spring.ai.chat.memory.repository.jdbc.initialize-schema"}
                 );
        }
    }
}
```

6.2 工具

> DateTimeTools | WeatherTools | or more ...

- DateTimeTools.java

```
package org.example.ai.tools;

import org.springframework.ai.tool.annotation.Tool;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@Component
public class DateTimeTools {

    static final DateTimeFormatter FORMATTER =
            DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    @Tool(
            description = "获取当前的系统日期和时间,格式为 yyyy-MM-dd HH:mm:ss",
            name = "get_current_time"
```

```
    )
    String getCurrentTime() {
        return LocalDateTime.now().format(FORMATTER);
    }

}
```

- GeoHelper.java

```
package org.example.ai.tools.weather;

import org.springframework.web.client.RestClient;

import java.util.List;

public final class GeoHelper {

    // https://geocoding-api.open-meteo.com/v1/search?
    // name=Shanghai&count=1&format=json&language=zh
    public static Geocoding getGeocodingByName(String city) {

        RestClient client = RestClient
                .builder()
                .baseUrl("https://geocoding-api.open-meteo.com")
                .build();

        GeocodingBody body = client.get()
                .uri(uriBuilder -> uriBuilder.path("/v1/search")
                        .queryParam("name", city)
                        .queryParam("count", 1)
                        .queryParam("language", "zh")
                        .queryParam("format", "json")
                        .build()
                )
                .retrieve()
                .body(GeocodingBody.class);

        assert body != null;
        if (body.results() != null && !body.results().isEmpty()) {
            return body.results().getFirst();
        }

        return null;

    }

    // 地理
    public static record Geocoding(String name,
```

```
                                    double latitude,
                                    double longitude) {}

    public static record GeocodingBody(List<Geocoding> results) {}

}
```

- WeatherHelper.java

```
package org.example.ai.tools.weather;

import com.fasterxml.jackson.annotation.JsonProperty;
import org.springframework.web.client.RestClient;

public final class WeatherHelper {

    // https://api.open-meteo.com/v1/forecast?
    // latitude=31.22222&longitude=121.45806&current=temperature_2m,wind_speed_10m
    public static Current getOpenWeather(double latitude, double longitude) {

        RestClient client = RestClient.builder().baseUrl("https://api.open-
meteo.com").build();

        WeatherResponse response = client.get()
                .uri(uriBuilder -> uriBuilder
                        .path("/v1/forecast")
                        .queryParam("latitude", latitude)
                        .queryParam("longitude", latitude)
                        .queryParam("current", "temperature_2m,wind_speed_10m")
                        .build()
                )
                .retrieve()
                .body(WeatherResponse.class);

        return response.current();

    }

    public static record WeatherResponse(double latitude,
                                         double longitude,
                                         @JsonProperty("current")
                                         Current current) {}

    public static record Current(@JsonProperty("temperature_2m")
                                 double temperature,
                                 @JsonProperty("wind_speed_10m")
                                 double windSpeed) {}
```

```
}
```

- WeatherTools.java

```java
package org.example.ai.tools;


import org.example.ai.tools.weather.GeoHelper;
import org.example.ai.tools.weather.WeatherHelper;
import org.springframework.ai.tool.annotation.Tool;
import org.springframework.ai.tool.annotation.ToolParam;
import org.springframework.stereotype.Component;

@Component
public class WeatherTools {

    @Tool(description = "查询指定城市当前的实时天气,包括温度、天气状况和风力",
          name = "get_weather",
          returnDirect = false)

    String getWeather(
        @ToolParam(description = "城市名称,使用中文,不带省份和'市'字,例如:北京、上海")
        String city) {

        GeoHelper.Geocoding geocoding = GeoHelper.getGeocodingByName(city);

        if (geocoding == null) {
            return "暂无该城市天气天气信息";
        }

        WeatherHelper.Current current =
            WeatherHelper.getOpenWeather(geocoding.latitude(),
                                         geocoding.longitude());


        if (current == null) {
            return "暂无该城市天气天气信息";
        }

        return current.toString();
    }
}
```

- ChatController.java

```java
package org.example.ai.controller;


import org.example.ai.tools.DateTimeTools;
// import org.example.ai.tools.WeatherTools;
import org.example.ai.tools.WeatherTools;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.MessageChatMemoryAdvisor;
import org.springframework.ai.chat.client.advisor.SimpleLoggerAdvisor;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ChatController {

    final ChatClient client;

    public ChatController(ChatClient.Builder builder,
                          ChatMemory memory) {

        MessageChatMemoryAdvisor advisor =
MessageChatMemoryAdvisor.builder(memory).build();

        this.client = builder
                .defaultSystem("你是一个简洁的中文助手")
                // .defaultAdvisors(new SimpleLoggerAdvisor(50), advisor)
                .defaultAdvisors(advisor)
                .build();
    }

    DateTimeTools dateTimeTools;

    @Autowired
    public void setDateTimeTools(DateTimeTools dateTimeTools) {
        this.dateTimeTools = dateTimeTools;
    }

    WeatherTools weatherTools;

    @Autowired
    public void setWeatherTools(WeatherTools weatherTools) {
        this.weatherTools = weatherTools;
    }

    @GetMapping("/ai")
    public String chat(String prompt, String username) {

        return client
```

```
                .prompt()
                .advisors(advisorSpec ->
                          advisorSpec.param(ChatMemory.CONVERSATION_ID, username))

                .tools(dateTimeTools, weatherTools)
                .user(prompt)
                .call()
                .content();
    }

}
```

6.3 RAG

> spring-ai-starter-vector-store-qdrant

6.3.1 环境配置

- pom

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-vector-store-qdrant</artifactId>
</dependency>
```

- application.properties

```
#qdrant
spring.ai.vectorstore.qdrant.host=localhost
spring.ai.vectorstore.qdrant.port=6334
spring.ai.vectorstore.qdrant.collection-name=ai
spring.ai.vectorstore.qdrant.initialize-schema=true
#spring.ai.vectorstore.qdrant.use-tls=false
#spring.ai.vectorstore.qdrant.api-key=
#spring.ai.vectorstore.qdrant.content-field-name=
spring.ai.openai.embedding.base-url=https://aliyuncs.com/compatible-mode/v1
spring.ai.openai.embedding.api-key=${TEXT-EMBEDDING-V4-API-KEY}
spring.ai.openai.embedding.model=text-embedding-v4
spring.ai.openai.embedding.dimensions=1024
```

6.3.2 文档写入存储

- RagDocumentInitializer.java

```
package org.example.ai.runner;


import org.springframework.ai.document.Document;
import org.springframework.ai.reader.markdown.MarkdownDocumentReader;
import org.springframework.ai.reader.markdown.config.MarkdownDocumentReaderConfig;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class RagDocumentInitializer implements ApplicationRunner {

    final VectorStore vectorStore;

    public RagDocumentInitializer(VectorStore vectorStore) {
        this.vectorStore = vectorStore;
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        MarkdownDocumentReaderConfig config = MarkdownDocumentReaderConfig
                .builder()
```

```
                .withHorizontalRuleCreateDocument(true)
                .withIncludeCodeBlock(false)
                .withIncludeBlockquote(false)
                .build();
        MarkdownDocumentReader reader =
                new MarkdownDocumentReader("classpath:company.md", config);

        List<Document> documents = reader.read();

        vectorStore.add(documents);

    }
}
```

6.3.3 工具

- KnowledgeTools.java

```
package org.example.ai.tools;

import org.springframework.ai.document.Document;
import org.springframework.ai.tool.annotation.Tool;
import org.springframework.ai.vectorstore.SearchRequest;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.stream.Collectors;

@Component
public class KnowledgeTools {

    final VectorStore vectorStore;

    public KnowledgeTools(VectorStore vectorStore) {
        this.vectorStore = vectorStore;
    }

    @Tool(description  = "检索内部知识库，获取与用户问题相关的信息。"
            + "当用户询问公司政策、产品文档，或其他非通用常识的内部资料时使用此工具。",
            name = "search_knowledge")
    public String searchKnowledge(String prompt){

        List<Document> documents = vectorStore.similaritySearch(SearchRequest
                .builder()
                .query(prompt)
```

```
                .topK(5)
                .build());

        if (documents.isEmpty()) {
            return "告诉用户没有找到相关信息";
        }

        return
documents.stream().map(Document::toString).collect(Collectors.joining(","));

    }

}
```

- ChatController.java

```
package org.example.ai.controller;


import org.example.ai.tools.DateTimeTools;
// import org.example.ai.tools.WeatherTools;
import org.example.ai.tools.KnowledgeTools;
import org.example.ai.tools.WeatherTools;
import org.springframework.ai.chat.client.ChatClient;
import org.springframework.ai.chat.client.advisor.MessageChatMemoryAdvisor;
import org.springframework.ai.chat.client.advisor.SimpleLoggerAdvisor;
import org.springframework.ai.chat.memory.ChatMemory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Flux;

@RestController
public class ChatController {

    final ChatClient client;

    public ChatController(ChatClient.Builder builder,
                          ChatMemory memory) {

        MessageChatMemoryAdvisor advisor =
MessageChatMemoryAdvisor.builder(memory).build();

        this.client = builder
                .defaultSystem("你是一个简洁的中文助手")
                // .defaultAdvisors(new SimpleLoggerAdvisor(50), advisor)
                .defaultAdvisors(advisor)
```

```
                .build();
    }


    KnowledgeTools knowledgeTools;

    @Autowired
    public void setKnowledgeTools(KnowledgeTools knowledgeTools) {
        this.knowledgeTools = knowledgeTools;
    }

    @GetMapping(value="/ai", produces = MediaType.TEXT_PLAIN_VALUE + ";charset=UTF-8")
    public Flux<String> chat(String prompt, String username) {

        return client
                .prompt()
                .advisors(advisorSpec → advisorSpec.param(ChatMemory.CONVERSATION_ID,
username))
                .tools(dateTimeTools, weatherTools, knowledgeTools)
                .user(prompt)
                .stream()
                .content();
    }

}
```

- ui

6.4 React 前端实现

6.5 MCP