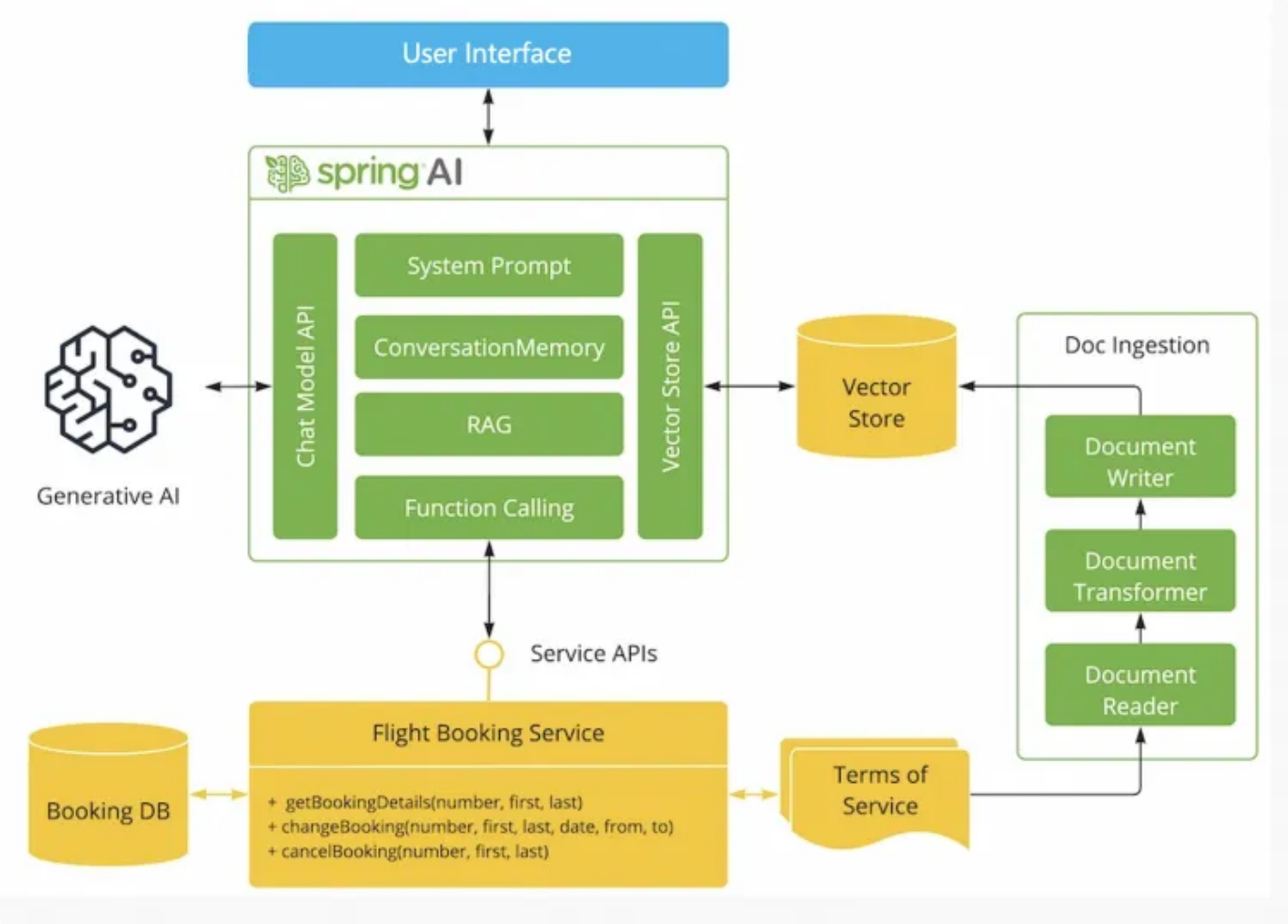
Spring AI 智能航空助手项目

文档嵌入

文档查询

完整架构图



环境准备

- Java 17
- Node.js 18+
- 获取API Key

api-key 配置

在正式开始体验之前，需要申请到模型的 api-key。申请地址：https://help.aliyun.com/zh/dashscope/developer-reference/activate-dashscope-and-create-an-api-key

您可以通过 DashScope 提供的方式配置 api-key，SCA AI 完全兼容 DashScope 环境变量配置 key 的方式：https://help.aliyun.com/zh/dashscope/developer-reference/api-key-settings

实战开始

前端：

前端环境安装：https://note.youdao.com/s/AXcWXq3v

1. 直接下载我的代码
2. cd app/chatgpt-demo
3. 执行安装依赖命令： `npm i`
4. 运行: `npm run dev`

后端

依赖：

```java
<dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud.ai</groupId>
            <artifactId>spring-ai-alibaba-starter</artifactId>
            <version>1.0.0-M2.1</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>


    </dependencies>
```

配置：

```java
spring.ai.dashscope.api-key=${TONGYI_AI_KEY}
spring.ai.dashscope.chat.options.model=qwen-max
```

创建Controller

配置ChatClient

```java

@RestController
public class OpenAiController {
    private final ChatClient chatClient;

    public OpenAiController(ChatClient.Builder chatClientBuilder, VectorStore vectorStore, ChatMemory chatMemory) {
        this.chatClient = chatClientBuilder.build();
    }
}
```

流式对话

```java
@CrossOrigin
@GetMapping(value = "/ai/generateStreamAsString", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
public Flux<String> generateStreamAsString(@RequestParam(value = "message", defaultValue = "讲个笑话") String message) {
    //Prompt prompt = new Prompt(new UserMessage(message));
    //return chatClient.stream(prompt);
    Flux<String> content = chatClient.prompt()
    .user(message)
    .stream()
    .content();

    return  content;

}
```

解决sse长连接重复请求问题：

```java
 @CrossOrigin
    @GetMapping(value = "/ai/generateStreamAsString", produces = MediaType.TEXT_EVENT_STREAM_VALUE)
    public Flux<String> generateStreamAsString(@RequestParam(value = "message", defaultValue = "讲个笑话") String message) {
        //Prompt prompt = new Prompt(new UserMessage(message));
        //return chatClient.stream(prompt);
        Flux<String> content = chatClient.prompt()
                .user(message)
                .stream()
                .content();

        return  content
                .doOnNext(System.out::println)
                .concatWith(Flux.just("[complete]"));

    }
```

加入打印日志拦截器

1. 加入Advisor

```java
public class LoggingAdvisor implements RequestResponseAdvisor {

    @Override
    public AdvisedRequest adviseRequest(AdvisedRequest request, Map<String, Object> context) {
        System.out.println("Request: " + request);
        return request;
    }
}
```

2. 使用chatClientBuilder的 `defaultAdvisors()` 方法注册Advisor

```java
public OpenAiController(ChatClient.Builder chatClientBuilder, VectorStore vectorStore, ChatMemory chatMemory) {
        this.chatClient = chatClientBuilder.defaultAdvisors(new LoggingAdvisor()).build();
    }
```

预设角色

我们做的是一个智能机票助手， 他主要的职责是给用户退票、改票、买票。 必须职责分明

```java
public OpenAiController(ChatClient.Builder chatClientBuilder, VectorStore vectorStore, ChatMemory chatMemory) {
        this.chatClient = chatClientBuilder
                .defaultSystem("""
                        您是“XuShu”航空公司的客户聊天支持代理。请以友好、乐于助人且愉快的方式来回复。
                       您正在通过在线聊天系统与客户互动。
                       请讲中文。
                       今天的日期是 {current_date}.
                    """)
                .defaultAdvisors(new LoggingAdvisor()).build();
    }

```

对话记忆

```java
@Bean
public ChatMemory chatMemory() {
    return new InMemoryChatMemory();
}
```

```java
.defaultAdvisors(new PromptChatMemoryAdvisor(chatMemory))
```

实现 退订 、 更改预定

所有退订 、 更改 逻辑 都在该类中已准备好`FlightBookingService` ，没有操作数据库，这个不是重点。

退订业务：

超过2天无法退订

退订后状态改为“取消”

#	Name	Date	From	To	Status
101	云小宝	2024-11-11	南京	南京	✅
102	李千问	2024-11-13	青岛	成都	✅
103	张百炼	2024-11-15	青岛	深圳	✅
104	王通义	2024-11-17	广州	南京	✅
105	刘魔搭	2024-11-19	成都	南京	✅

难点

1. 需要获取当前航班信息（ 需要用户提供 姓名，航班号）

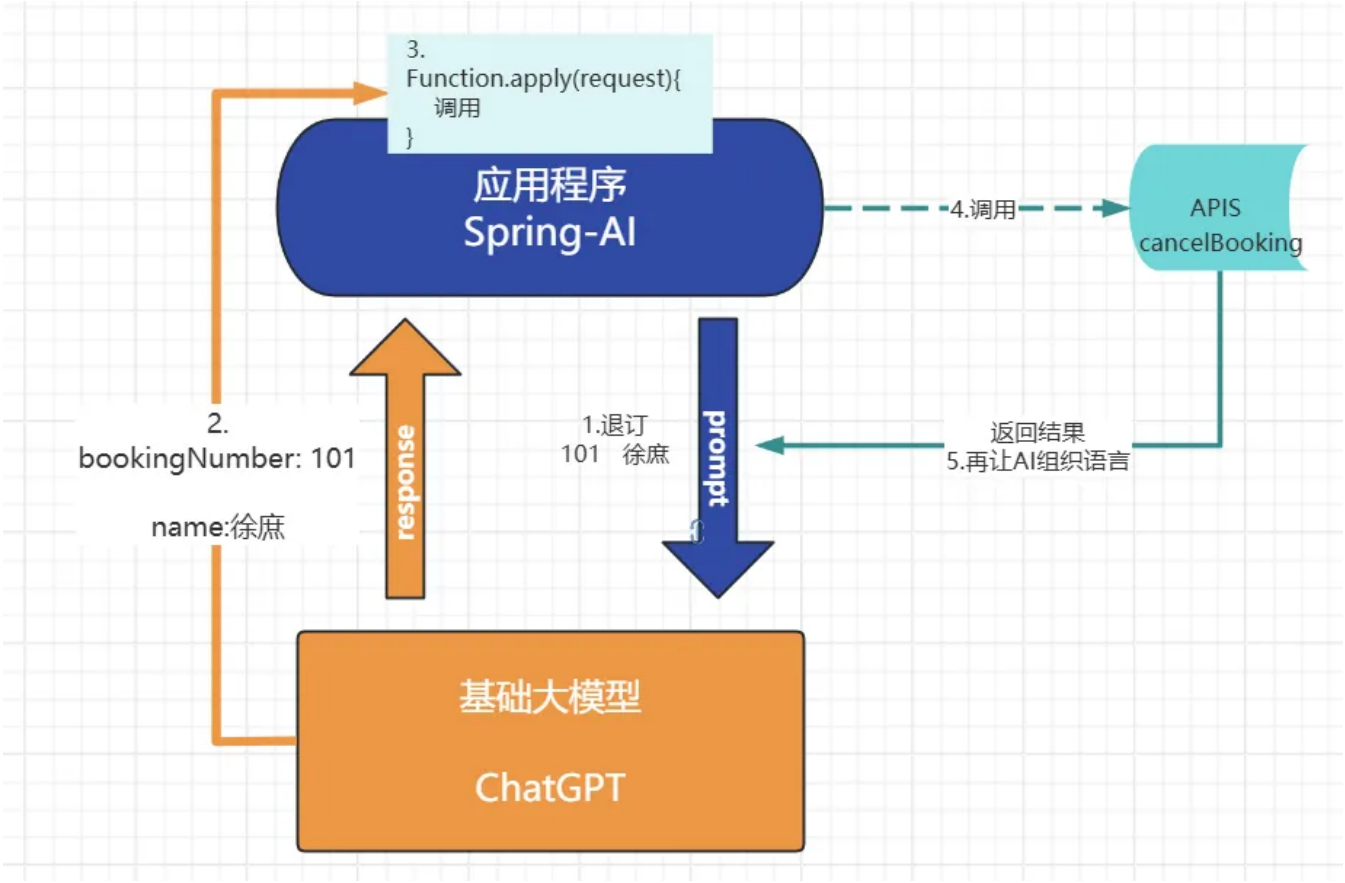
解决： 加入提示词

> 在进行有关预订或取消预订的信息之前，您必须始终从用户处获取以下信息：预订号、客户姓名。
>
> 在询问用户之前，请检查消息历史记录以获取此信息。

```java
public OpenAiController(ChatClient.Builder chatClientBuilder, VectorStore vectorStore, ChatMemory chatMemory) {
        this.chatClient = chatClientBuilder
                .defaultSystem("""
                        您是“XuShu”航空公司的客户聊天支持代理。请以友好、乐于助人且愉快的方式来回复。
                        您正在通过在线聊天系统与客户互动。
                        在提供有关预订或取消预订的信息之前，您必须始终
                        从用户处获取以下信息：预订号、客户姓名。
                        在询问用户之前，请检查消息历史记录以获取此信息。
                        请讲中文。
                        今天的日期是 {current_date}.
                    """)
                .defaultAdvisors(
                        new PromptChatMemoryAdvisor(chatMemory),
                        new QuestionAnswerAdvisor(vectorStore, SearchRequest.defaults()), // RAG
                        new LoggingAdvisor()).build();
    }

```

通过function-call 进行方法回调



1. 需要告诉大模型：回调哪个方法
    a. 提供实现了Function接口Bean(调用apply)
2. 需要告诉大模型：什么对话才回调
    a. 配置Function作用（处理退订）
3. 需要告诉大模型：提取对话的什么关键字
    a. Function的第一个泛型去指定提取关键字的变量名

————Function 来解决

通过function-call 进行退订

```java
@Bean
@Description("取消机票预定")
public Function<CancelBookingRequest, String> cancelBooking() {
    return request -> {
        flightBookingService.cancelBooking(request.bookingNumber(), request.name());
        return "";
    };
}

public record CancelBookingRequest(String bookingNumber, String name) {
}
```

退订前确认信息是否存在

解决： 加入提示词

> 如果更改需要收费，您必须在继续之前征得用户同意。

或者

> 在更改或退订之前，请先获取预订信息并且用户确定信息。

通过function-call 获取预定信息

```java
@Bean
    @Description("获取机票预定详细信息")
    public Function<BookingDetailsRequest, BookingDetails> getBookingDetails() {
        return request -> {
            try {
                return flightBookingService.getBookingDetails(request.bookingNumber(), request.name());
            }
            catch (Exception e) {
                logger.warn("Booking details: {}", NestedExceptionUtils.getMostSpecificCause(e).getMessage());
                return new BookingDetails(request.bookingNumber(), request.name(), null, null, null, null, null);
            }
        };
    }


    public record BookingDetailsRequest(String bookingNumber, String name) {
    }


    public record BookingDetails(String bookingNumber, String name, LocalDate date, BookingStatus bookingStatus,
            String from, String to, String bookingClass) {
    }
```

通过function-call 修改预定信息

```java
    @Bean
    @Description("修改机票预定日期")
    public Function<ChangeBookingDatesRequest, String> changeBooking() {
        return request -> {
            flightBookingService.changeBooking(request.bookingNumber(), request.name(), request.date(), request.from(),
                    request.to());
            return "";
        };
    }

```

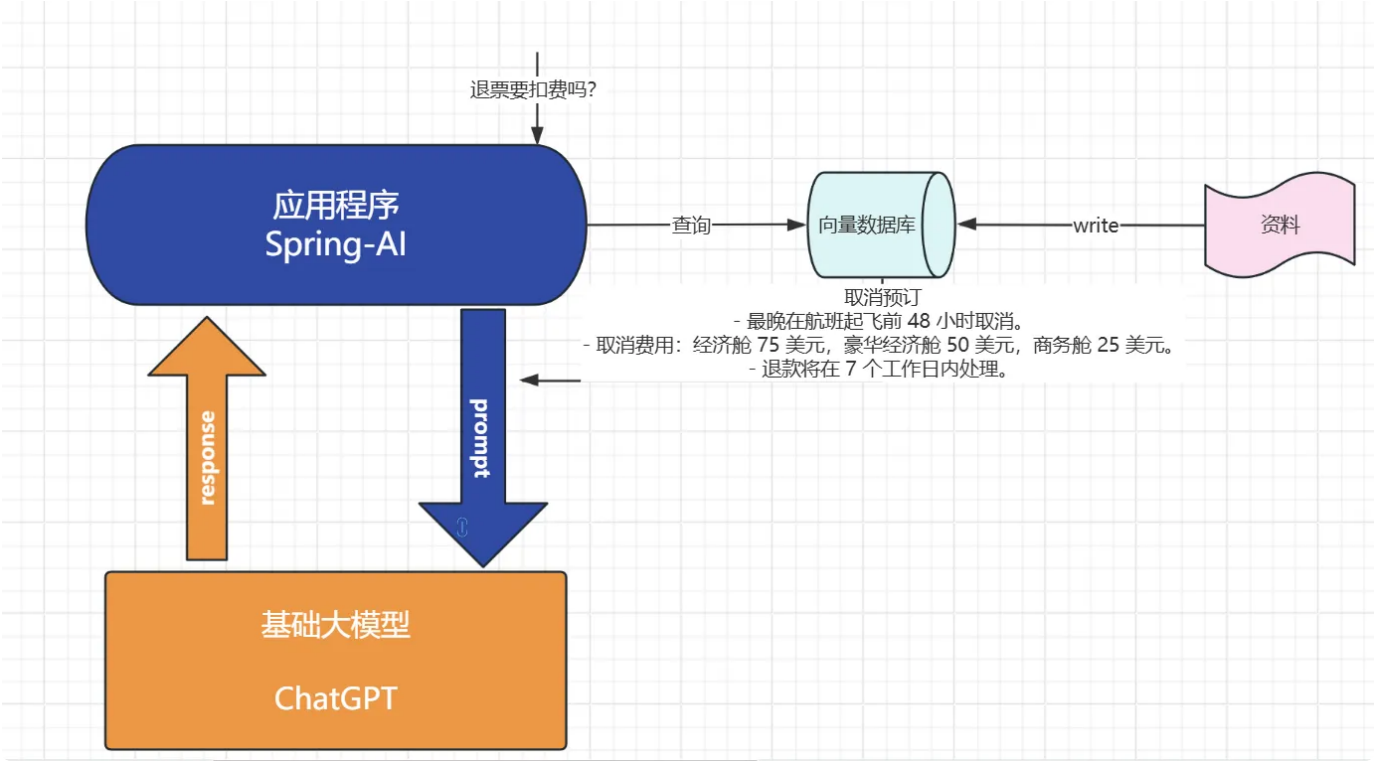
ChangeBookingDatesRequest：

```java

```

通过RAG(检索增强生成)，外挂一个知识库

向量数据库：**相似性检索**，大数据领域（推荐、你喜欢的）



1. 配置向量数据库
2. 写入数据（Embedding）
3. 查询

📎terms-of-service.txt

Plain Text

```
本服务条款适用于您对 Funnair 的体验。预订航班，即表示您同意这些条款。
1. 预订航班
- 通过我们的网站或移动应用程序预订。
- 预订时需要全额付款。
- 确保个人信息（姓名、ID 等）的准确性，因为更正可能会产生 25 的费用。
2. 更改预订
- 允许在航班起飞前 24 小时更改。
- 通过在线更改或联系我们的支持人员。
- 改签费：经济舱 50，豪华经济舱 30，商务舱免费。
3. 取消预订
- 最晚在航班起飞前 48 小时取消。
- 取消费用：经济舱 75 美元，豪华经济舱 50 美元，商务舱 25 美元。
- 退款将在 7 个工作日内处理。
```

向量数据库

redis es ...

```java
    @Bean
    public VectorStore vectorStore(EmbeddingModel embeddingModel) {
        return new SimpleVectorStore(embeddingModel);
    }

```

写入向量数据库


```java
@Bean
CommandLineRunner ingestTermOfServiceToVectorStore(EmbeddingModel embeddingModel, VectorStore vectorStore,
                                                   @Value("classpath:rag/terms-of-service.txt") Resource termsOfServiceDocs) {

    return args -> {
        vectorStore.write(                                  // 3.写入
            new TokenTextSplitter().transform(          // 2.转换
                new TextReader(termsOfServiceDocs).read())  // 1.读取
        );

    };
}
```

配置Advisor：

new QuestionAnswerAdvisor(vectorStore, SearchRequest.defaults()), // RAG

`QuestionAnswerAdvisor` 可以在用户发起的提问时，先向数据库查询相关的文档，再把相关的文档拼接到用户的提问中，再让模型生成答案。那就是 `RAG` 的实现了。

```java
 this.chatClient = chatClientBuilder
                .defaultSystem("""
                        您是“Funnair”航空公司的客户聊天支持代理。请以友好、乐于助人且愉快的方式来回复。
                        您正在通过在线聊天系统与客户互动。
                        在提供有关预订或取消预订的信息之前，您必须始终
                        从用户处获取以下信息：预订号、客户姓名。
                        在询问用户之前，请检查消息历史记录以获取此信息。
                        在更改预订之前，您必须确保条款允许这样做。
                        如果更改需要收费，您必须在继续之前征得用户同意。
                        使用提供的功能获取预订详细信息、更改预订和取消预订。
                        如果需要，可以调用相应函数调用完成辅助动作。
                        请讲中文。
                        今天的日期是 {current_date}.
                    """)
                .defaultAdvisors(
                        new PromptChatMemoryAdvisor(chatMemory),
                            new QuestionAnswerAdvisor(vectorStore, SearchRequest.defaults()), // RAG
                        new LoggingAdvisor())
                .defaultFunctions("getBookingDetails", "changeBooking", "cancelBooking") // FUNCTION CALLING
                .build();
```

redis向量数据库

安装RedisStack

需要先禁用掉自己原本的redis，防止端口冲突。访问 `localhost:8001` 查看数据库的信息。用户名： `default` ,密码： `123456` 。

```bash
docker run -d --name redis-stack --restart=always  -v redis-data:/data -p 6379:6379 -p 8001:8001 -e REDIS_ARGS="--requirepass 123456" redis/redis-stack:latest
```

引入依赖

```xml
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-redis-store</artifactId>
</dependency>
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-tika-document-reader</artifactId>
</dependency>
```

配置连接

```yaml
spring:
  data:
    redis:
      database: 0
      timeout: 10s
      lettuce:
        pool:
          # 连接池最大连接数
          max-active: 200
          # 连接池最大阻塞等待时间（使用负值表示没有限制）
          max-wait: -1ms
          # 连接池中的最大空闲连接
          max-idle: 10
          # 连接池中的最小空闲连接
          min-idle: 0
      repositories:
        enabled: false
      password: 123456
```

配置向量数据库

如果你的项目里面有用到redis，需要先禁用`RedisVectorStoreAutoConfiguration`。这是SpringAI自动配置RedisStack的向量数据库连接，会导致Redis的连接配置冲突。

`VectorStore`对象需要提供`EmbeddingModel`，这个案例提供的是阿里灵积的`EmbeddingModel`。可以切换换成其他厂家的EmbeddingModel。

```java
@Configuration
// 禁用SpringAI提供的RedisStack向量数据库的自动配置，会和Redis的配置冲突。
@EnableAutoConfiguration(exclude = {RedisVectorStoreAutoConfiguration.class})
// 读取RedisStack的配置信息
@EnableConfigurationProperties({RedisVectorStoreProperties.class})
@AllArgsConstructor
public class RedisVectorConfig {

    /**
     * 创建RedisStack向量数据库
     *
     * @param embeddingModel 嵌入模型
     * @param properties     redis-stack的配置信息
     * @return vectorStore 向量数据库
     */
    @Bean
    public VectorStore vectorStore(EmbeddingModel embeddingModel,
                                   RedisVectorStoreProperties properties,
                                   RedisConnectionDetails redisConnectionDetails) {
        RedisVectorStore.RedisVectorStoreConfig config = RedisVectorStore.RedisVectorStoreConfig.builder().withIndexName(properties.getIndex()).withPrefix(properties.getPrefix()).build();
        return new RedisVectorStore(config, embeddingModel,
                new JedisPooled(redisConnectionDetails.getStandalone().getHost(),
                        redisConnectionDetails.getStandalone().getPort()
                        , redisConnectionDetails.getUsername(),
                        redisConnectionDetails.getPassword()),
                properties.isInitializeSchema());
    }
}
```

文档嵌入

在上面的 `VectorStore` 配置中我们提供了 `EmbeddingModel`，调用 `vectorStore.add(splitDocuments)` 底层会把文档给 `EmbeddingModel` 把文本变成向量然后再存入向量数据库。

```java
private final VectorStore vectorStore;
    /**
     * 嵌入文件
     *
     * @param file 待嵌入的文件
     * @return 是否成功
     */
    @SneakyThrows
    @PostMapping("embedding")
    public Boolean embedding(@RequestParam MultipartFile file) {
        // 从IO流中读取文件
        TikaDocumentReader tikaDocumentReader = new TikaDocumentReader(new InputStreamResource(file.getInputStream()));
        // 将文本内容划分成更小的块
        List<Document> splitDocuments = new TokenTextSplitter()
                .apply(tikaDocumentReader.read());
        // 存入向量数据库，这个过程会自动调用embeddingModel,将文本变成向量再存入。
        vectorStore.add(splitDocuments);
        return true;
    }
```

文档查询

调用 `vectorStore.similaritySearch(query)` 时同样会先把用户的提问给 `EmbeddingModel`，将提问变成向量，然后与向量数据库中的文档向量进行相似度计算（cosine值）。

要注意：此时向量数据库不会回答用户的提问。要回答用户的提问需要指定advisor

```java
/**
     * 查询向量数据库
     *
     * @param query 用户的提问
     * @return 匹配到的文档
     */

    @GetMapping("query")
    public List<Document> query(@RequestParam String query) {
        return vectorStore.similaritySearch(query);
    }
```

指定advisor

```java
return chatClient.prompt()
                .user(prompt)
                // 2. QuestionAnswerAdvisor会在运行时替换模板中的占位符`question_answer_context`，替换成向量数据库中查询到的文档。此时的query=用户的提问+替换完的提示词模板;
                .advisors(new QuestionAnswerAdvisor(vectorStore, prompt))
                .stream()
                // 3. query发送给大模型得到答案
                .content()
                .map(chatResponse -> ServerSentEvent.builder(chatResponse)
                        .event("message")
                        .build());
```