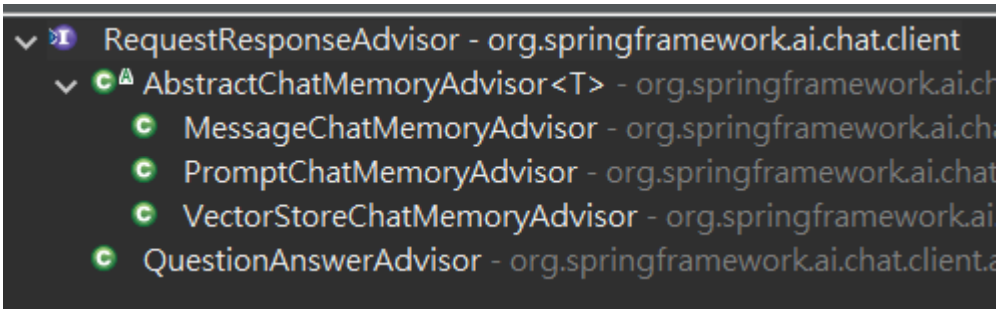
Day19 - Spring AI 的 AOP 應用

Spring 框架有兩個很重要的觀念，一個是 IoC，另一個則是 AOP，只是 AOP 大多整合進應用中，開發人員已越來越少直接撰寫了，AOP 的概念如下

AOP 可以攔截指定的方法並且對方法增強，而且無需侵入到業務代碼中，使業務與非業務處理邏輯分離，例如 Spring 的事務，透過事務的註解配置，Spring 會自動在業務方法中開啟、提交業務，並且在業務處理失敗時，執行對應的回滾策略。另一個常見的應用是 log，使用 AOP 可以在不更動原程式的情況下在執行前後加上 log 紀錄

在 AOP 中找到切入點後要執行的程式就稱為 Advice，還記得 Day 15 最後提到的 `advisors` 嗎？這正是要放入 Advice 程式，Spring AI 提供一個 `RequestResponseAdvisor` 介面，只要實作就能在　ChatModel 呼叫 call() 或是 stream() 的前後調用，Spring AI 提供的實作類別有以下四個



- MessageChatMemoryAdvisor: 將訊息送出前加入 ChatMemory，之後將所有 ChatMemory 資料已 List 方式送給 AI 處理
- PromptChatMemoryAdvisor: 將所有 ChatMemory 訊息加入 SystemMassage，並將當前訊息加入 ChatMemory 中，最後與 SystemMessage 一同送給 AI 處理
- VectorStoreChatMemoryAdvisor: 前面兩個都是處理短期聊天資訊，一旦系統關閉就會失去所有聊天內容，而此類別則可在訊息送出前加入向量資料庫，並從向量資料庫中取出相似聊天內容送給 AI 處理
- QuestionAnswerAdvisor: 此類別即是 RAG 的調用實作，凱文大叔會在後面詳細說明

程式碼實作

今天先來測試短期記憶，凱文大叔將昨天的 Service 進行改寫，原本手動加入 InMemoryChatMemory 以及取得歷史聊天訊息的部分都不用自己動手了

```
@RequiredArgsConstructor
@Service
public class ChatService {
    private final ChatClient chatClient;
    private ChatMemory chatMemory = new InMemoryChatMemory();

    public String chat(String chatId, String userMessage) {
        return this.chatClient.prompt()
        .advisors(new MessageChatMemoryAdvisor(chatMemory, ch
        .user(userMessage)
        .call().content()
    }
}
```

`.advisors(new MessageChatMemoryAdvisor(chatMemory, chatId, 30))`

這一段與

`.advisors(new PromptChatMemoryAdvisor(chatMemory, chatId, 30))`

效果基本差不多，大家也可以自行測試結果是否有差異，參數分別是

`chatMemory`:用來存對話訊息的 ChatMemory

`chatId`:區分對話的代號

`30`:送給 AI 的歷史聊天數量

驗收成果

> 可以看出效果跟昨天一模一樣，MessageChatMemoryAdvisor 與 PromptChatMemoryAdvisor 只差在歷史訊息送出的方式

回顧

今天學到了甚麼?

- AOP 基本概念
- Spring AI 提供的 Advisor 程式
- 如何在 ChatClient 設定 Advisor 程式

> 程式看起來與昨天差別不大，不過 `.advisors()` 是可放入多個 Advisor 程式的，除了 Spring AI 提供的以外，開發人員也可以自行實作 `RequestResponseAdvisor`，例如用來記錄 log的 Advisor，可一起加入`.advisors()`，或是有多個程式需依序調用，可以寫多個 Advisor 程式形成 Advisor 鏈