《Ai Agent》第3-9节：实例化对话客户端

来自： 码农会锁

小傅哥

2025年07月19日 15:25

本章重点： ★★★☆☆
课程视频：https://t.zsxq.com/bePzF
代码分支：https://gitcode.net/KnowledgePlanet/ai-agent-station-study/-/tree/3-9-bean-chat-client
工程代码： https://gitcode.net/KnowledgePlanet/ai-agent-station-study

作者：小傅哥
博客：https://bugstack.cn

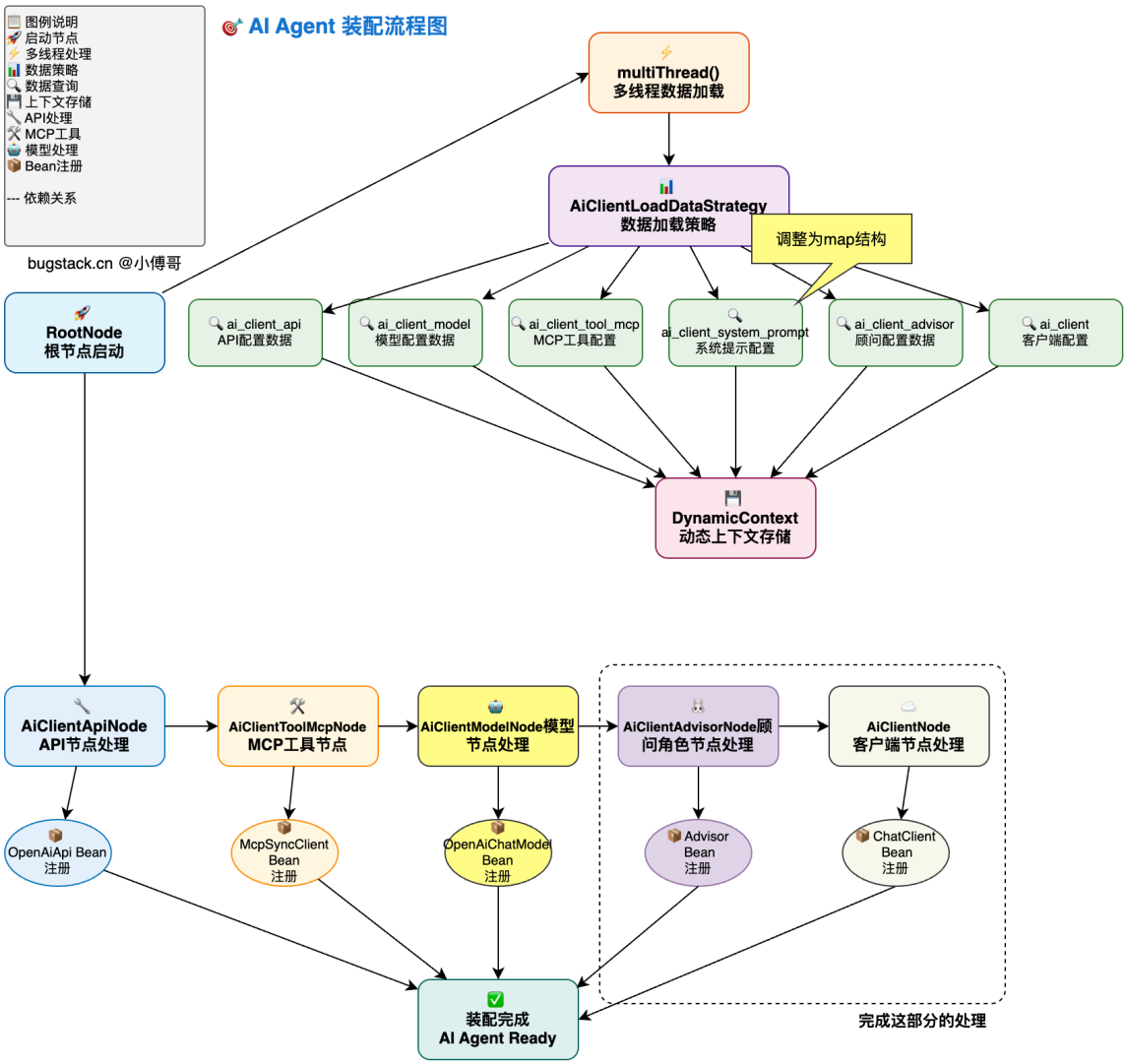
> 沉淀、分享、成长，让自己和他人都能有所收获！😄

一、本章诉求

经过前面一系列的准备工作，包括；api、mcp、model，本节我们要进行 advisor 顾问角色的实例化，之后进行 ChatClient 对话客户端的实例化。

二、功能流程

如图，整体 ChatClient 客户端实例化过程；



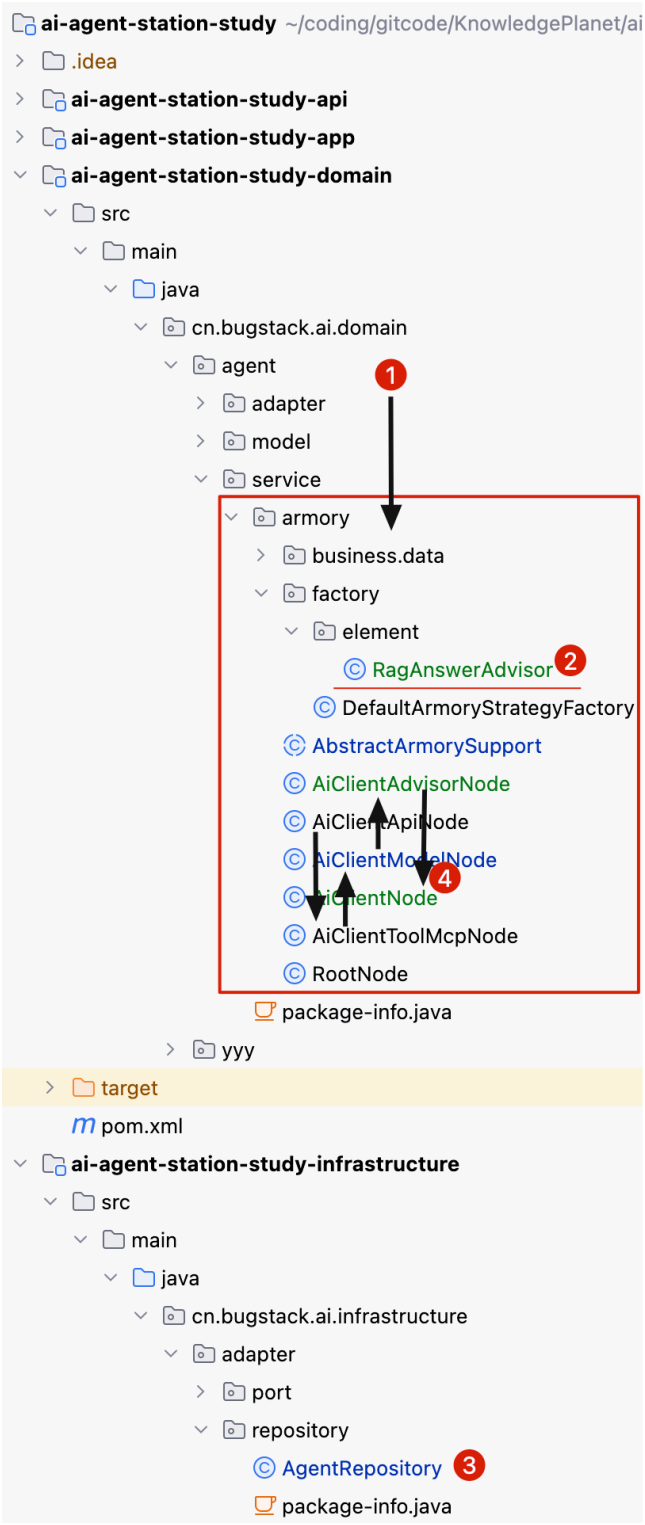
首先，以构建 AiClientNode 的对话客户端为目的，已经完成了相关的元素实例化步骤。本节这里要处理的是，顾问角色的构建，以及构建 AiClientNode 节点。

之后，AiClientNode 的构建，是关联了其他各项元素的，所以在构建时，需要在 AiClientNode 节点，从 Spring 容器通过 getBean 的方式，检索到对应的各项元素。

注意，ai_client_system_prompt 系统提示词，需要修改为 Map 结构数据。这样更方便我们从数据里获取，哪些是属于当前 AiClientNode 构建时所需的元素。

三、编码实现

1. 工程结构



1. 构建 AiClient 客户端，同时需要构建 Advisor 顾问角色。
2. 访问知识库设计
3. 稍微调整系统提示词返回格式为 Map 结构
4. 黑色箭头实例化链路过程，最后实例化 AiClientNode

如图，以构建 AiClientNode 为目的，增加 Advisor 顾问角色的构建。

当你看到 ④ 这块的时候，就会感受到使用规则树模式在这里的优势。这种设计可以很好的区分出各个节点，只要找到类，就能找到对应的实现。

2. 修改说明

增加 AiClientAdvisorNode 节点，获取多线程加载数据 `AiClientAdvisorVO` 并根据顾问角色类型，ChatMemory、RagAnswer 分别构建不同的顾问类型。顾问，就是一种设计方式，通过顾问来包装访问记忆上下文和知识库内容

修改 AiClientLoadDataStrategy 数据加载操作，对 `repository.queryAiClientSystemPromptMapByClientIds(clientIdList)`; 数据查询，返回 Map 结果。方便，AiClientNode 构建时候，获取和使用数据。map 的数据结构，key 是关联id，也就是能拿到构建客户端的时候，关联的 system_prompt 数据。

增加 AiClientNode 节点，循环构建对话客户端。每个客户端，关联的预设话术（SystemPrompt）、对话模型（ChatMode）、MCP 服务（不过这里数据库没配置，而是关联到了 ChatMode 上）、顾问角色（Advisor），之后就可以实例化客户端，并把客户端注册到 Spring 容器中。

增加 AiClientAdvisorTypeEnumVO 顾问角色类型，策略枚举类，因为不同的顾问角色构建的方式有所不同，这部分可以抽取到枚举类中实现抽象方法来处理。

3. 节点创建 - 顾问角色

3.1 策略枚举

```
@Getter
@AllArgsConstructor
@NoArgsConstructor
public enum AiClientAdvisorTypeEnumVO {

    CHAT_MEMORY("ChatMemory", "上下文记忆（内存模式）") {
        @Override
        public Advisor createAdvisor(AiClientAdvisorVO aiClientAdvisorVO, VectorStore vectorStore) {
            AiClientAdvisorVO.ChatMemory chatMemory = aiClientAdvisorVO.getChatMemory();
            return PromptChatMemoryAdvisor.builder(
                    MessageWindowChatMemory.builder()
                            .maxMessages(chatMemory.getMaxMessages())
                            .build()
            ).build();
        }
    },

    RAG_ANSWER("RagAnswer", "知识库") {
        @Override
        public Advisor createAdvisor(AiClientAdvisorVO aiClientAdvisorVO, VectorStore vectorStore) {
            AiClientAdvisorVO.RagAnswer ragAnswer = aiClientAdvisorVO.getRagAnswer();
            return new RagAnswerAdvisor(vectorStore, SearchRequest.builder()
                    .topK(ragAnswer.getTopK())
                    .filterExpression(ragAnswer.getFilterExpression())
                    .build());
        }
    }

    ;

    private String code;
    private String info;

    // 静态Map缓存，用于快速查找
    private static final Map<String, AiClientAdvisorTypeEnumVO> CODE_MAP = new HashMap<>();

    // 静态初始化块，在类加载时初始化Map
    static {
        for (AiClientAdvisorTypeEnumVO enumVO : values()) {
            CODE_MAP.put(enumVO.getCode(), enumVO);
        }
    }

    /**
     * 策略方法：创建顾问对象
     * @param aiClientAdvisorVO 顾问配置对象
     * @param vectorStore 向量存储
     * @return 顾问对象
     */
    public abstract Advisor createAdvisor(AiClientAdvisorVO aiClientAdvisorVO, VectorStore vectorStore);

    /**
     * 根据code获取枚举
     * @param code 编码
     * @return 枚举对象
     */
    public static AiClientAdvisorTypeEnumVO getByCode(String code) {
        AiClientAdvisorTypeEnumVO enumVO = CODE_MAP.get(code);
        if (enumVO == null) {
            throw new RuntimeException("err! advisorType " + code + " not exist!");
```

```
        }
        return enumVO;
    }

}
```

3.2 顾问构建

```java
@Slf4j
@Service
public class AiClientAdvisorNode extends AbstractArmorySupport {

    @Resource
    private VectorStore vectorStore;

    @Resource
    private AiClientNode aiClientNode;

    @Override
    protected String doApply(ArmoryCommandEntity requestParameter, DefaultArmoryStrategyFactory.DynamicConte
        log.info("Ai Agent 构建节点，Advisor 顾问角色{}", JSON.toJSONString(requestParameter));

        List<AiClientAdvisorVO> aiClientAdvisorList = dynamicContext.getValue(dataName());

        if (aiClientAdvisorList == null || aiClientAdvisorList.isEmpty()) {
            log.warn("没有需要被初始化的 ai client advisor");
            return router(requestParameter, dynamicContext);
        }

        for (AiClientAdvisorVO aiClientAdvisorVO : aiClientAdvisorList) {
            // 构建顾问访问对象
            Advisor advisor = createAdvisor(aiClientAdvisorVO);
            // 注册Bean对象
            registerBean(beanName(aiClientAdvisorVO.getAdvisorId()), Advisor.class, advisor);
        }

        return router(requestParameter, dynamicContext);
    }

    @Override
    public StrategyHandler<ArmoryCommandEntity, DefaultArmoryStrategyFactory.DynamicContext, String> get(Ar
        return aiClientNode;
    }

    protected String beanName(String beanId) {
        return AiAgentEnumVO.AI_CLIENT_ADVISOR.getBeanName(beanId);
    }

    @Override
    protected String dataName() {
        return AiAgentEnumVO.AI_CLIENT_ADVISOR.getDataName();
    }

    private Advisor createAdvisor(AiClientAdvisorVO aiClientAdvisorVO) {
        String advisorType = aiClientAdvisorVO.getAdvisorType();
        AiClientAdvisorTypeEnumVO advisorTypeEnum = AiClientAdvisorTypeEnumVO.getByCode(advisorType);
        return advisorTypeEnum.createAdvisor(aiClientAdvisorVO, vectorStore);
    }

}
```

根据配置的数据 `List<AiClientAdvisorVO> aiClientAdvisorList` 循环构建顾问角色。而 createAdvisor 方法则通过枚举策略进行构建。

如果将来还有其他顾问角色，则在 AiClientAdvisorTypeEnumVO 扩展实现方法即可。

4. 节点创建 - 客户端

```java
public class AiClientNode extends AbstractArmorySupport {

    @Override
    protected String doApply(ArmoryCommandEntity requestParameter, DefaultArmoryStrategyFactory.DynamicConte
        log.info("Ai Agent 构建节点，客户端{}", JSON.toJSONString(requestParameter));

        List<AiClientVO> aiClientList = dynamicContext.getValue(dataName());

        Map<String, AiClientSystemPromptVO> systemPromptMap = dynamicContext.getValue(AiAgentEnumVO.AI_CLIEI

        for (AiClientVO aiClientVO : aiClientList) {
            // 1. 预设话术
            StringBuilder defaultSystem = new StringBuilder("Ai 智能体 \r\n");
            List<String> promptIdList = aiClientVO.getPromptIdList();
            for (String promptId : promptIdList) {
                AiClientSystemPromptVO aiClientSystemPromptVO = systemPromptMap.get(promptId);
                defaultSystem.append(aiClientSystemPromptVO.getPromptContent());
            }

            // 2. 对话模型
            OpenAiChatModel chatModel = getBean(aiClientVO.getModelBeanName());

            // 3. MCP 服务
            List<McpSyncClient> mcpSyncClients = new ArrayList<>();
            List<String> mcpBeanNameList = aiClientVO.getMcpBeanNameList();
            for (String mcpBeanName : mcpBeanNameList) {
                mcpSyncClients.add(getBean(mcpBeanName));
            }

            // 4. advisor 顾问角色
            List<Advisor> advisors = new ArrayList<>();
            List<String> advisorBeanNameList = aiClientVO.getAdvisorBeanNameList();
            for (String advisorBeanName : advisorBeanNameList) {
                advisors.add(getBean(advisorBeanName));
            }

            Advisor[] advisorArray = advisors.toArray(new Advisor[]{});

            // 5. 构建对话客户端
            ChatClient chatClient = ChatClient.builder(chatModel)
                    .defaultSystem(defaultSystem.toString())
                    .defaultToolCallbacks(new SyncMcpToolCallbackProvider(mcpSyncClients.toArray(new McpSyn
                    .defaultAdvisors(advisorArray)
                    .build();

            registerBean(beanName(aiClientVO.getClientId()), ChatClient.class, chatClient);
        }

        return router(requestParameter, dynamicContext);
    }

    @Override
    public StrategyHandler<ArmoryCommandEntity, DefaultArmoryStrategyFactory.DynamicContext, String> get(Ar
        return defaultStrategyHandler;
    }

    @Override
    protected String beanName(String id) {
        return AiAgentEnumVO.AI_CLIENT.getBeanName(id);
    }

    @Override
    protected String dataName() {
```

```
        return AiAgentEnumVO.AI_CLIENT.getDataName();
    }

}
```

AiClientVO 客户端节点的构建，则需要把所有关联到的元素，依次按照Bean的名称，从 Spring 容器获取。

注意，AiClientSystemPromptVO 不需要复杂的构建，直接从数据库获取就使用即可。

5. 链接处理

AiClientModelNode -> AiClientAdvisorNode -> AiClientNode
举例；

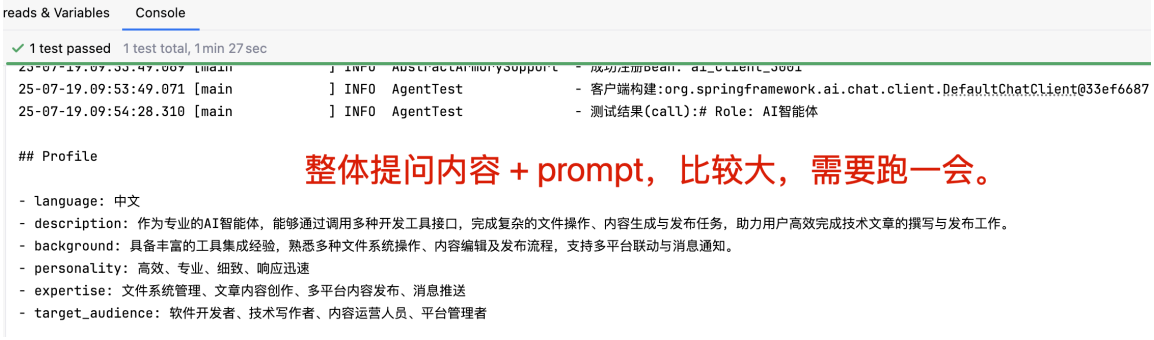
```
@Override
public StrategyHandler<ArmoryCommandEntity, DefaultArmoryStrategyFactory.DynamicContext, String> get(ArmoryC
    return aiClientAdvisorNode;
}
```

每个节点构建完成后，就开始关联到下一个节点。

四、测试验证

```
@Slf4j
@RunWith(SpringRunner.class)
@SpringBootTest
public class AgentTest {

    @Resource
    private DefaultArmoryStrategyFactory defaultArmoryStrategyFactory;

    @Resource
    private ApplicationContext applicationContext;

    @Test
    public void test_aiClient() throws Exception {
        StrategyHandler<ArmoryCommandEntity, DefaultArmoryStrategyFactory.DynamicContext, String> armoryStra
                defaultArmoryStrategyFactory.armoryStrategyHandler();

        String apply = armoryStrategyHandler.apply(
                ArmoryCommandEntity.builder()
                        .commandType(AiAgentEnumVO.AI_CLIENT.getCode())
                        .commandIdList(Arrays.asList("3001"))
                        .build(),
                new DefaultArmoryStrategyFactory.DynamicContext());

        ChatClient chatClient = (ChatClient) applicationContext.getBean(AiAgentEnumVO.AI_CLIENT.getBeanName
        log.info("客户端构建:{}", chatClient);

        String content = chatClient.prompt(Prompt.builder()
                .messages(new UserMessage(
                        """
                                有哪些工具可以使用
                                """))
                .build()).call().content();

        log.info("测试结果(call):{}", content);
```

```
    }

}
```



整体提问内容 + prompt，比较大，需要跑一会。

客户端测试，构建完成后，获取客户端节点。

之后可以验证提问 有哪些工具可以使用 或者其他的也可以。这部分提问的操作，会结合 prompt 一起提问给 ai

五、读者作业

简单作业：完成本节新增节点的构建操作，并验证结果。注意，要增加 debug 调试，一步步验证结果。这样才能吸收的更多。

复杂作业：如果我们只做model的装配操作，应该怎么实现。还记得最开始，我们有一个只加载的 model 数据吧。可以尝试完成下，只根据 model 进行加载。