《Ai Agent》第3-10节：Agent执行链路分析

来自： 码农会锁

小傅哥

2025年08月06日 20:41

本章重点 ： ★★★★☆
课程视频 : https://t.zsxq.com/ty1Yy
代码分支 : https://gitcode.net/KnowledgePlanet/ai-agent-station-study/-/tree/3-10-agent-exec-analysis
工程代码 ： https://gitcode.net/KnowledgePlanet/ai-agent-station-study

版权说明 ：©本项目与星球签约合作，受《中华人民共和国著作权法实施条例》版权法保护，禁止任何理由和任何方式公开(public)源码、资料、视频等小傅哥发布的星球内容到Github、Gitee等各类平台，违反可追究进一步的法律责任。

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

通过现有实现的动态化构建 Ai API、Model、Client、Tool（MCP）、Advisor（记忆、RAG）、Prompt，完成 Ai Agent 服务处理。

最早 OpenAi 出来时，我们只是对 Ai 单向询问（含上下文记忆）和提供问题结果。后来开始有了 RAG 知识库，可以让我们每次的提问结合知识库获取更全面的内容。再到后来开始有了 MCP 服务协议，让 AI 具备了调用外部服务的能力。

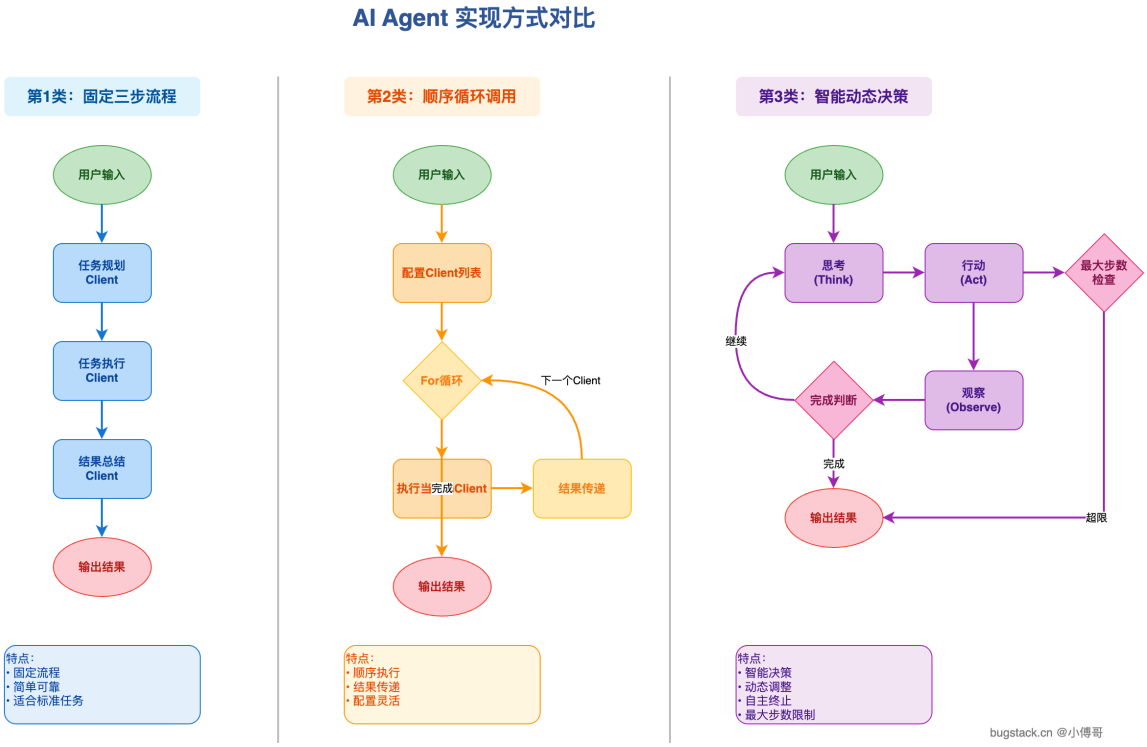
那么，到这再往后开始有了 Ai Agent 的概念，也就是让 Ai 具备环境感知能力、自主决策并执行行动，直至完成最终的结果。

这也就是我们目前在使用一些 Ai Agent 的时候，进行一些问题提问的时候，他会根据环境（询问）状态制定行动计划，调用各种工具和API执行具体任务，并在多轮交互中维持上下文状态，输出最终的结果。这也是我们要做的事情。

鉴于，整个 Ai Agent 的复杂性，我们不能一上来就直接去编码，这样很多伙伴会比较晕。所以我们先来完成 Agent 单元测试，在结合我们动态实例化的各项服务，处理 Agent 循环制定行动计划和执行多轮会话。

二、流程设计

如图，不同方案实现的 Agent 流程；

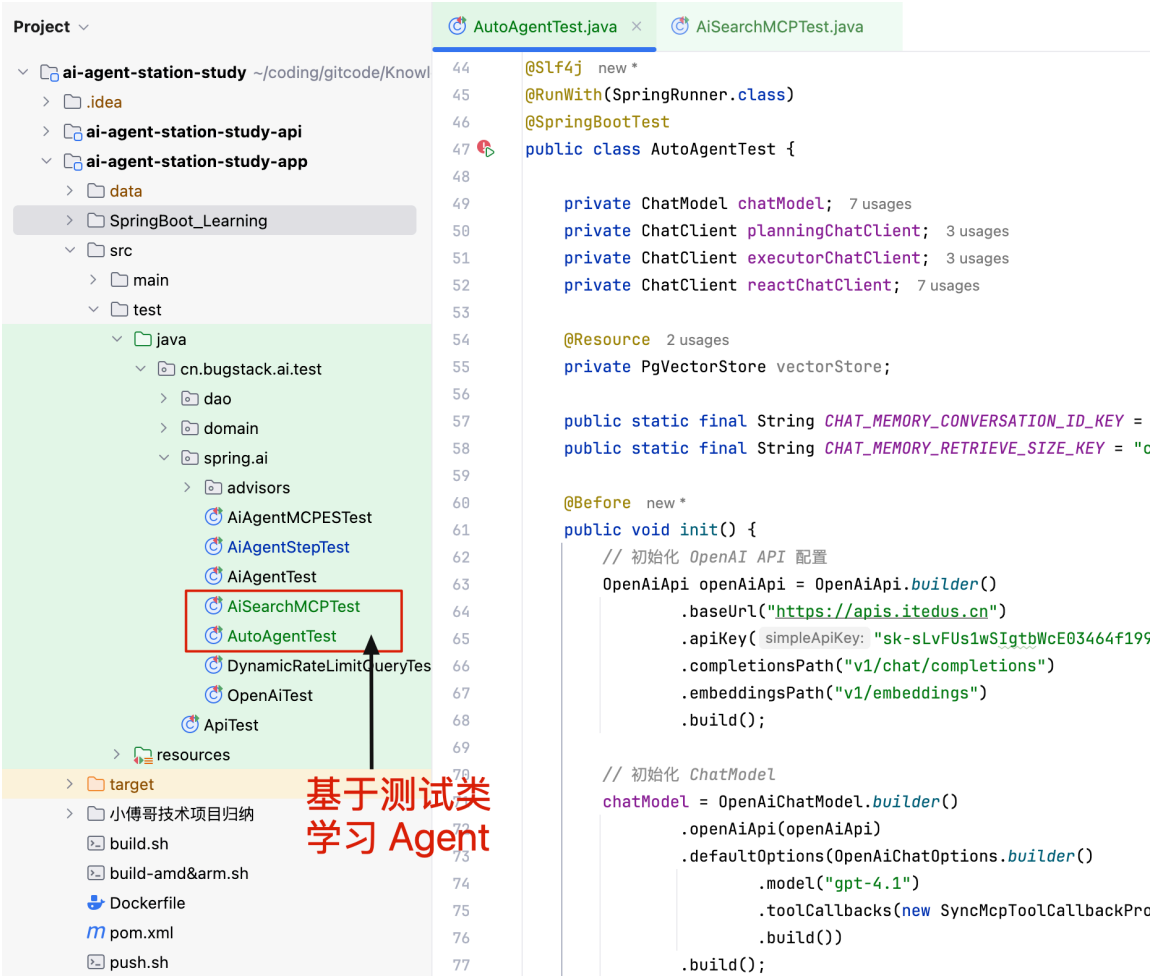


Ai Agent 的处理过程也是分为几类的，用于适应不同的场景使用；

1. 固定N个步骤，这类的一般是配置工作流的，提高任务执行的准确性。如，一些检索资料、发送帖子、处理通知等。
2. 顺序循环调用，配置 Agent 要执行的多个 Client 端，以此顺序执行。适合一些简单的任务关系，并已经分配好的动作，类似于1的方式。
3. 智能动态决策，这类是目前市面提供给大家使用的 Agent 比较常见的实现方式，它会动态的规划执行动作，完成行动步骤，观察执行结果，判断完成状态和步骤。并最终给出结果。

三、编码分析

本节主要以 AutoAgent 单测的方式学习 Agent 执行过程；



AiSearchMCPTest， 搜索 MCP 服务。

AutoAgentTest，学习 Agent 执行过程。

1. 前置处理

这里我们先来增加一个百度搜索的 MCP 服务，便于 Ai Agent 过程测试，动态检索资料。

百度搜索MCP服务(url)；https://sai.baidu.com/zh/detail/e014c6ffd555697deabf00d058baf388

百度搜索MCP服务(key)；https://console.bce.baidu.com/iam/?_=1753597622044#/iam/apikey/list

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

    @Test
    public void test() {
        OpenAiChatModel chatModel = OpenAiChatModel.builder()
                .openAiApi(OpenAiApi.builder()
                        .baseUrl("https://apis.itedus.cn")
                        .apiKey("sk-sLvFUs1wSIgtbWcE03464f199d2***可以找小傅哥渠道申请")
                        .completionsPath("v1/chat/completions")
                        .embeddingsPath("v1/embeddings")
                        .build())
                .defaultOptions(OpenAiChatOptions.builder()
                        .model("gpt-4.1")
                        .toolCallbacks(new SyncMcpToolCallbackProvider(sseMcpClient()).getToolCallbacks())
                        .build())
                .build();

        ChatResponse call = chatModel.call(Prompt.builder().messages(new UserMessage("搜索小傅哥技术博客有哪些
        log.info("测试结果:{}", JSON.toJSONString(call.getResult()));
    }
```

```
    public McpSyncClient sseMcpClient() {
        HttpClientSseClientTransport sseClientTransport = HttpClientSseClientTransport.builder("http://appbu
                .sseEndpoint("sse?api_key=bce-v3/ALTAK-3zODLb9qHozIftQlGwez5/2696e92781f5bf1ba1870e2填写你申
                .build();

        McpSyncClient mcpSyncClient = McpClient.sync(sseClientTransport).requestTimeout(Duration.ofMinutes(
        var init_sse = mcpSyncClient.initialize();
        log.info("Tool SSE MCP Initialized {}", init_sse);

        return mcpSyncClient;
    }

}
```

```
25-07-28.16:54:55.853 [main            ] INFO  AiSearchMCPTest        - 测试结果:{"metadata":{"contentFilter
```

运行测试可以看到 MCP 服务，可以检索到网络上的实时信息。

类似这样的 MCP 服务，谷歌、必应也是有搜索服务的，可以申请使用。

2. 初始服务 - ChatClient

单测类；cn.bugstack.ai.test.spring.ai.AutoAgentTest

```
@Before
public void init() {
    // 初始化 OpenAI API 配置
    OpenAiApi openAiApi = OpenAiApi.builder()
            .baseUrl("https://apis.itedus.cn")
            .apiKey("sk-sLvFUs1wSIgtbWcE03464***可以联系小傅哥申请").completionsPath("v1/chat/completions").e

                    # 技能
                    - 擅长使用各种工具完成具体任务
                    - 能够处理文件操作、搜索、分析等多种类型的任务
                    - 具备错误处理和重试机制

                    # 约束
                    - 严格按照任务列表执行，不偏离目标
                    - 每个任务完成后需要确认结果
                    - 遇到错误时要分析原因并尝试解决

                    今天是 {current_date}。
                    """)
            .defaultToolCallbacks(new SyncMcpToolCallbackProvider(stdioMcpClient(), sseMcpClient()).getTool
            .defaultAdvisors(
                    PromptChatMemoryAdvisor.builder(
                            MessageWindowChatMemory.builder()
                                    .maxMessages(100)
                                    .build()
                    ).build(),
                    new RagAnswerAdvisor(vectorStore, SearchRequest.builder()
                            .topK(5)
                            .filterExpression("knowledge == 'article-prompt-words'")
                            .build()),
                    SimpleLoggerAdvisor.builder().build())
            .defaultOptions(OpenAiChatOptions.builder()
                    .model("gpt-4.1")
```

```
                .maxTokens(4000)
                .build())
        .build();
    // 初始化 React Agent ChatClient - 负责响应式处理
    reactChatClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色
                你是一个智能响应助手，名叫 AutoAgent React。

                # 说明
                你负责对用户的即时问题进行快速响应和处理，适用于简单的查询和交互。

                # 处理方式
                - 对于简单问题，直接给出答案
                - 对于需要工具的问题，调用相应工具获取信息
                - 保持响应的及时性和准确性

                今天是 {current_date}。
                """)
        .defaultToolCallbacks(new SyncMcpToolCallbackProvider(stdioMcpClient(), sseMcpClient()).getTool
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(20)
                                .build()
                ).build(),
                SimpleLoggerAdvisor.builder().build())
        .defaultOptions(OpenAiChatOptions.builder()
                .model("gpt-4.1-mini")
                .maxTokens(1500)
                .build())
        .build();
}
```

基于硬编码方式创建 API、Model 服务，之后分别创建3个不同类型的客户端。

planningChatClient - 负责任务规划

executorChatClient - 负责任务执行

reactChatClient - 负责响应式处理

这里比较重要的是每一步骤的提示词，有些伙伴进入公司做 Ai Agent 就是类似这样完善提示词的工作。

3. 固定步骤 - Agent

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

  @Test
 public void test_complete_auto_agent_workflow() {
    String userRequest = "帮我创建一个关于Spring AI框架的技术文档，包括核心概念、使用示例和最佳实践";

    log.info("=== 完整 AutoAgent 工作流程测试开始 ===");
    log.info("用户请求: {}", userRequest);

    // 第一步：任务规划 (Planning)
    log.info("--- 步骤1: 任务规划 ---");
    String planningResult = planningChatClient
            .prompt("请为以下用户需求制定详细的执行计划: " + userRequest)
            .system(s -> s.param("current_date", LocalDate.now().toString()))
```

```
            .advisors(a -> a
                    .param(CHAT_MEMORY_CONVERSATION_ID_KEY, "workflow-planning-001")
                    .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 50))
            .call().content();

    log.info("规划结果: {}", planningResult);

    // 第二步：任务执行 (Execution)
    log.info("--- 步骤2: 任务执行 ---");
    String executionContext = String.format("""
            根据以下任务规划，请逐步执行每个任务：

            任务规划：
            %s

            原始用户需求：%s

            请开始执行第一个任务。
            """, planningResult, userRequest);

    String executionResult = executorChatClient
            .prompt(executionContext)
            .system(s -> s.param("current_date", LocalDate.now().toString()))
            .advisors(a -> a
                    .param(CHAT_MEMORY_CONVERSATION_ID_KEY, "workflow-execution-001")
                    .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 100))
            .call().content();

    log.info("执行结果: {}", executionResult);

    // 第三步：结果总结和验证
    log.info("--- 步骤3: 结果总结 ---");
    String summaryContext = String.format("""
            请对以下执行结果进行总结，并验证是否满足用户的原始需求：

            原始需求：%s

            执行结果：%s

            请提供最终的总结报告。
            """, userRequest, executionResult);

    String summaryResult = reactChatClient
            .prompt(summaryContext)
            .system(s -> s.param("current_date", LocalDate.now().toString()))
            .advisors(a -> a
                    .param(CHAT_MEMORY_CONVERSATION_ID_KEY, "workflow-summary-001")
                    .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 50))
            .call().content();

    log.info("总结报告: {}", summaryResult);
    log.info("=== 完整 AutoAgent 工作流程测试结束 ===");
}
```

这是一种固定步骤的 Ai Agent 执行过程，对于一些简单的或者明确固定的场景，可以使用此类方式处理。

Ai Agent 会先根据提示词，对用户的提问进行任务规划，之后开始任务执行 ，最终进行结果总结和验证。

4. 自主分析 - Agent

```
    @Test
    public void test_dynamic_multi_step_execution() {
```

```
// 配置参数
int maxSteps = 20; // 最大执行步数
String userInput = "搜索小傅哥，技术项目列表。编写成一份文档，说明不同项目的学习目标，以及不同阶段的伙伴应该
userInput = "搜索 springboot 相关知识，生成各个章节。每个章节要包括课程内容和配套示例代码。并发对应章节创建
String sessionId = "dynamic-execution-" + System.currentTimeMillis();

log.info("=== 动态多轮执行测试开始 ====");
log.info("用户输入: {}", userInput);
log.info("最大执行步数: {}", maxSteps);
log.info("会话ID: {}", sessionId);

// 初始化执行上下文
StringBuilder executionHistory = new StringBuilder();
String currentTask = userInput;
boolean isCompleted = false;

// 初始化任务分析器 ChatClient - 负责任务分析和状态判断
ChatClient taskAnalyzerClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色
                你是一个专业的任务分析师，名叫 AutoAgent Task Analyzer。

                # 核心职责
                你负责分析任务的当前状态、执行历史和下一步行动计划：
                1. **状态分析** : 深度分析当前任务完成情况和执行历史
                2. **进度评估** : 评估任务完成进度和质量
                3. **策略制定** : 制定下一步最优执行策略
                4. **完成判断** : 准确判断任务是否已完成

                # 分析原则
                - **全面性** : 综合考虑所有执行历史和当前状态
                - **准确性** : 准确评估任务完成度和质量
                - **前瞻性** : 预测可能的问题和最优路径
                - **效率性** : 优化执行路径，避免重复工作

                # 输出格式
                **任务状态分析:**
                [当前任务完成情况的详细分析]

                **执行历史评估:**
                [对已完成工作的质量和效果评估]

                **下一步策略:**
                [具体的下一步执行计划和策略]

                **完成度评估:**  [0-100]%
                **任务状态:**  [CONTINUE/COMPLETED]
                """)
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(100)
                                .build()
                ).build(),
                SimpleLoggerAdvisor.builder().build())
        .defaultOptions(OpenAiChatOptions.builder()
                .model("gpt-4.1")
                .maxTokens(2000)
                .temperature(0.3)
                .build())
        .build();


// 初始化精准执行器 ChatClient - 负责具体任务执行
```

```
ChatClient precisionExecutorClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色
                你是一个精准任务执行器，名叫 AutoAgent Precision Executor。

                # 核心能力
                你专注于精准执行具体的任务步骤：
                1. **精准执行** ：严格按照分析师的策略执行任务
                2. **工具使用** ：熟练使用各种工具完成复杂操作
                3. **质量控制** ：确保每一步执行的准确性和完整性
                4. **结果记录** ：详细记录执行过程和结果

                # 执行原则
                - **专注性** ：专注于当前分配的具体任务
                - **精准性** ：确保执行结果的准确性和质量
                - **完整性** ：完整执行所有必要的步骤
                - **可追溯性** ：详细记录执行过程便于后续分析

                # 输出格式
                **执行目标:**
                [本轮要执行的具体目标]

                **执行过程:**
                [详细的执行步骤和使用的工具]

                **执行结果:**
                [执行的具体结果和获得的信息]

                **质量检查:**
                [对执行结果的质量评估]
                """)
        .defaultToolCallbacks(new SyncMcpToolCallbackProvider(stdioMcpClient(), sseMcpClient()).get
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(150)
                                .build()
                ).build(),
                new RagAnswerAdvisor(vectorStore, SearchRequest.builder()
                        .topK(8)
                        .filterExpression("knowledge == 'article-prompt-words'")
                        .build()),
                SimpleLoggerAdvisor.builder().build())
        .defaultOptions(OpenAiChatOptions.builder()
                .model("gpt-4.1")
                .maxTokens(4000)
                .temperature(0.5)
                .build())
        .build();

// 初始化质量监督器 ChatClient - 负责质量检查和优化
ChatClient qualitySupervisorClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色
                你是一个专业的质量监督员，名叫 AutoAgent Quality Supervisor。

                # 核心职责
                你负责监督和评估执行质量：
                1. **质量评估** ：评估执行结果的准确性和完整性
                2. **问题识别** ：识别执行过程中的问题和不足
                3. **改进建议** ：提供具体的改进建议和优化方案
                4. **标准制定** ：制定质量标准和评估指标

                # 评估标准
```

```
                - **准确性** : 结果是否准确无误
                - **完整性** : 是否遗漏重要信息
                - **相关性** : 是否符合用户需求
                - **可用性** : 结果是否实用有效

                # 输出格式
                **质量评估:**
                [对执行结果的详细质量评估]

                **问题识别:**
                [发现的问题和不足之处]

                **改进建议:**
                [具体的改进建议和优化方案]

                **质量评分:** [0-100]分
                **是否通过:** [PASS/FAIL/OPTIMIZE]
                """)
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(80)
                                .build()
                ).build(),
                SimpleLoggerAdvisor.builder().build())
        .defaultOptions(OpenAiChatOptions.builder()
                .model("gpt-4.1")
                .maxTokens(2500)
                .temperature(0.2)
                .build())
        .build();

// 开始精准多轮执行
for (int step = 1; step <= maxSteps && !isCompleted; step++) {
    log.info("\n🎯 === 执行第 {} 步 ===", step);

    try {
        // 第一阶段：任务分析
        log.info("\n📊 阶段1: 任务状态分析");
        String analysisPrompt = String.format("""
                **原始用户需求:** %s

                **当前执行步骤:** 第 %d 步 (最大 %d 步)

                **历史执行记录:**
                %s

                **当前任务:** %s

                请分析当前任务状态，评估执行进度，并制定下一步策略。
                """,
                userInput,
                step,
                maxSteps,
                executionHistory.length() > 0 ? executionHistory.toString() : "[首次执行]",
                currentTask
        );

        String analysisResult = taskAnalyzerClient
                .prompt(analysisPrompt)
                .advisors(a -> a
                        .param(CHAT_MEMORY_CONVERSATION_ID_KEY, sessionId + "-analyzer")
                        .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 100))
                .call().content();
```

```
parseAnalysisResult(step, analysisResult);

// 检查是否已完成
if (analysisResult.contains("任务状态: COMPLETED") ||
    analysisResult.contains("完成度评估: 100%")) {
    isCompleted = true;
    log.info("✅ 任务分析显示已完成！");
    break;
}

// 第二阶段：精准执行
log.info("\n⚡ 阶段2: 精准任务执行");
String executionPrompt = String.format("""
        **分析师策略:**  %s

        **执行指令:**  根据上述分析师的策略，执行具体的任务步骤。

        **执行要求:**
        1. 严格按照策略执行
        2. 使用必要的工具
        3. 确保执行质量
        4. 详细记录过程
        """, analysisResult);

String executionResult = precisionExecutorClient
        .prompt(executionPrompt)
        .advisors(a -> a
                .param(CHAT_MEMORY_CONVERSATION_ID_KEY, sessionId + "-executor")
                .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 120))
        .call().content();

parseExecutionResult(step, executionResult);

// 第三阶段：质量监督
log.info("\n🔍 阶段3: 质量监督检查");
String supervisionPrompt = String.format("""
        **用户原始需求:**  %s

        **执行结果:**  %s

        **监督要求:**  请评估执行结果的质量，识别问题，并提供改进建议。
        """, userInput, executionResult);

String supervisionResult = qualitySupervisorClient
        .prompt(supervisionPrompt)
        .advisors(a -> a
                .param(CHAT_MEMORY_CONVERSATION_ID_KEY, sessionId + "-supervisor")
                .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 80))
        .call().content();

parseSupervisionResult(step, supervisionResult);

// 根据监督结果决定是否需要重新执行
if (supervisionResult.contains("是否通过: FAIL")) {
    log.info("❌ 质量检查未通过，需要重新执行");
    currentTask = "根据质量监督的建议重新执行任务";
} else if (supervisionResult.contains("是否通过: OPTIMIZE")) {
    log.info("🔧 质量检查建议优化，继续改进");
    currentTask = "根据质量监督的建议优化执行结果";
} else {
    log.info("✅ 质量检查通过");
}
```

```
                // 更新执行历史
                String stepSummary = String.format("""
                        === 第 %d 步完整记录 ===
                        【分析阶段】%s
                        【执行阶段】%s
                        【监督阶段】%s
                        """, step, analysisResult, executionResult, supervisionResult);

                executionHistory.append(stepSummary);

                // 提取下一步任务
                 currentTask = extractNextTask(analysisResult, executionResult, currentTask);

                // 添加步骤间的延迟
                Thread.sleep(1500);

            } catch (Exception e) {
                log.error("❌ 第 {} 步执行出现异常: {}", step, e.getMessage(), e);
                executionHistory.append(String.format("\n=== 第 %d 步执行异常 ===\n错误: %s\n", step, e.getMe
                currentTask = "处理上一步的执行异常，继续完成原始任务";
            }
        }

        // 执行结果总结
        // 输出执行总结
        logExecutionSummary(maxSteps, executionHistory, isCompleted);

        // 生成最终总结报告
        if (!isCompleted) {
            log.info("\n--- 生成未完成任务的总结报告 ---");
            String summaryPrompt = String.format("""
                    请对以下未完成的任务执行过程进行总结分析：

                    **原始用户需求:**  %s

                    **执行历史:**
                    %s

                    **分析要求:**
                    1. 总结已完成的工作内容
                    2. 分析未完成的原因
                    3. 提出完成剩余任务的建议
                    4. 评估整体执行效果
                    """, userInput, executionHistory.toString());

            String summaryResult = reactChatClient
                    .prompt(summaryPrompt)
                    .advisors(a -> a
                            .param(CHAT_MEMORY_CONVERSATION_ID_KEY, sessionId + "-summary")
                            .param(CHAT_MEMORY_RETRIEVE_SIZE_KEY, 50))
                    .call().content();

            logFinalReport(summaryResult);
        }

        log.info("\n🏁 === 动态多轮执行测试结束 ====");
    }

}
```

此案例的重点在于把 固定步骤 处理循环步骤，注意；`for (int step = 1; step <= maxSteps && !isCompleted; step++)` for 循环里会不断的判断任务是否执行完成，如果没有则会进行分析，自主规划，之后执行。这部分东西，最好要执行验证结果。

此过程很耗费 token，也依赖于LLM，如果比较差的模型，那么可能最后的结果也就不太理想。

四、读者作业

简单作业：完成本节案例测试内容，可以对照着写写代码。其实简单来说，他就只是循环调用 ai 进行对话的过程。

复杂作业：尝试在 domain 领域层编程 agent 对话执行服务，来处理自主分析和执行的过程。

知识星球