《Ai Agent》第3-15节：AgentFlow执行链路分析（扩展思路）

来自： 码农会锁

小傅哥

2025年08月23日 17:02

本章重点 ： ★★★☆☆
课程视频 ： https://t.zsxq.com/u9tjH
代码分支 ： https://gitcode.net/KnowledgePlanet/ai-agent-station-study/-/tree/3-15-agent-flow-analysis
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！😄

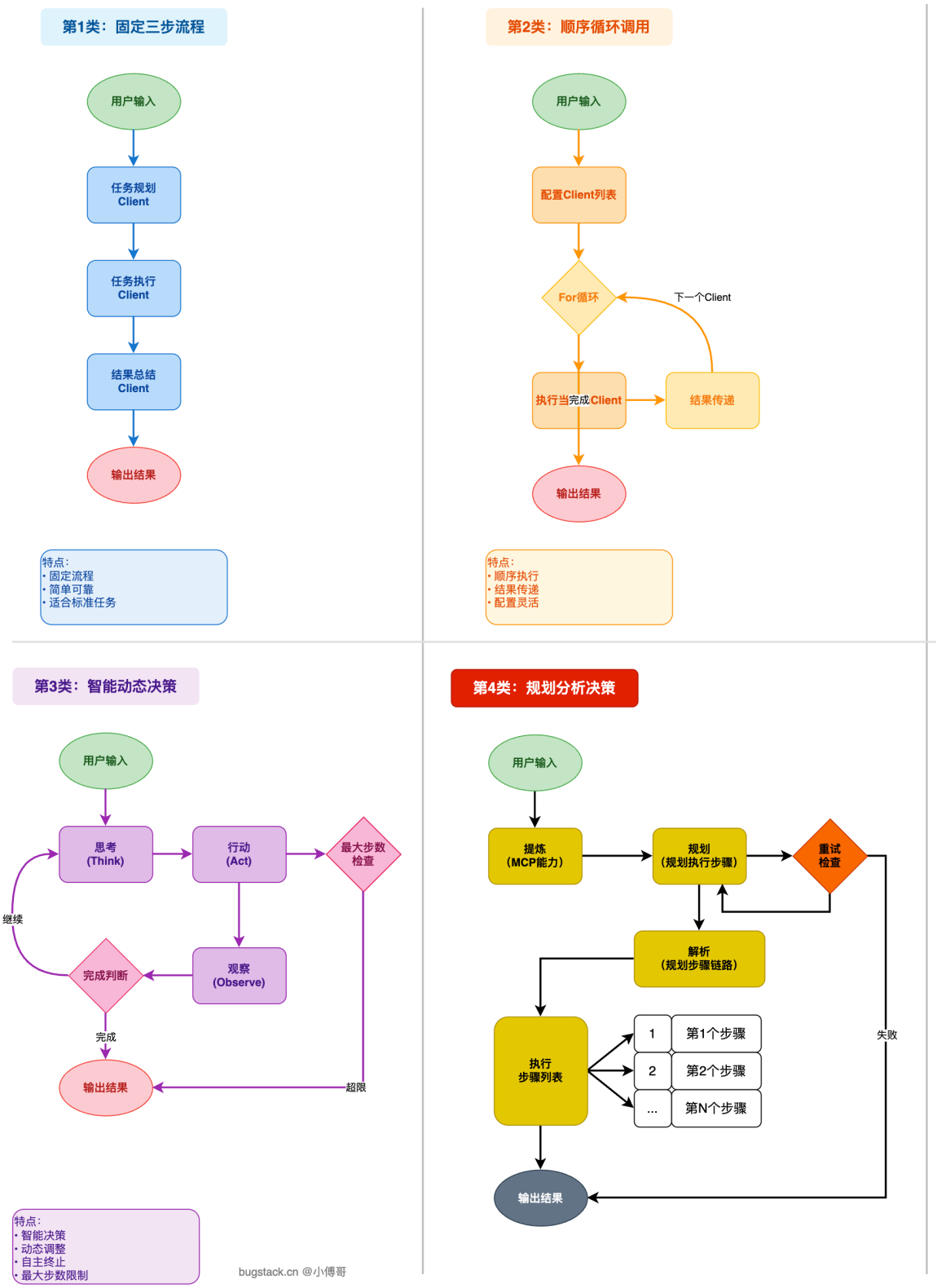
一、本章诉求

为了打开 Agent 的实现思路，本章我们再增加一种新的 Auto Agent 设计，这种设计方式以通过用户的提问和当前 Agent 配置的 MCP 工具集合，进行执行步骤的规划设计。之后在通过执行步骤按照拆分的步骤顺序号，依次进行执行。有点类似于 manus 的过程。

二、流程设计

如图，多种 Ai Agent 执行设计流程图；

AI Agent 实现方式对比

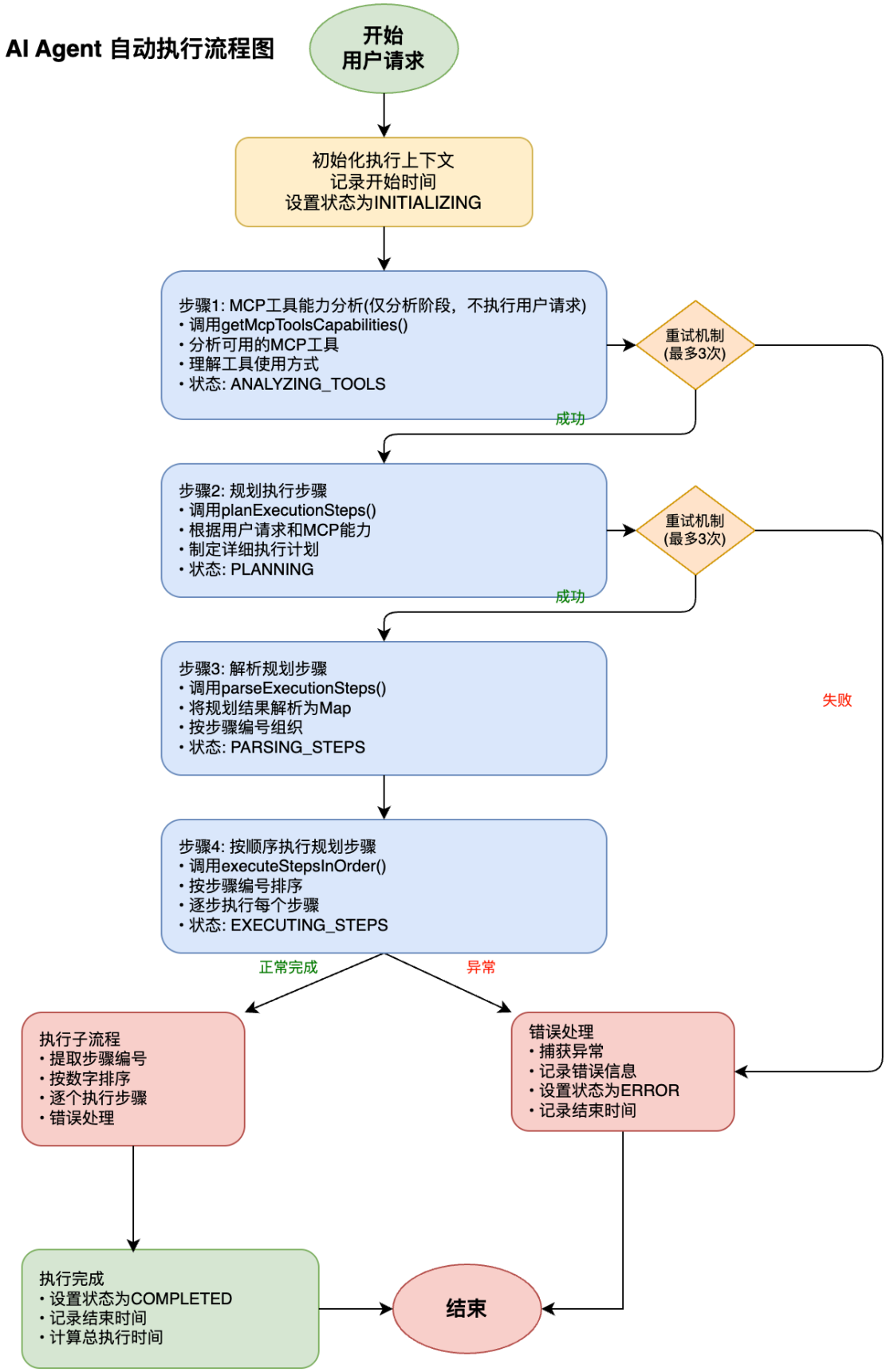


Ai Agent 的处理过程也是分为几类的，用于适应不同的场景使用；

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。
4. 【新增】规划分析决策，根据用户输入的信息诉求，以及配置的 MCP 的能力，进行步骤规划。之后把步骤拆分出 1、2、3 具体要做什么，在依次执行这些步骤。

三、编码分析

本节主要以 FlowAgent 单测的方式学习 Agent 执行过程；



步骤1；MCP 工具能力分析，这一步不会执行用户请求

步骤2；规划执行步骤，这一步结合用户的提问诉求进行规划

步骤3；解析规划步骤，存储成一条条待执行的步骤操作

步骤4；按照顺序依次执行规划步骤，直至全部完成。

1. 前置配置

单测类；cn.bugstack.ai.test.spring.ai.FlowAgentTest
本节我们私用到了第2阶段的 CSDN 发送文章、微信公众号推送消息，两个实现的 MCP 服务，需要进行步骤。如果没有学习需要往前翻看下，也可以直接配置使用。

```
version: '3.8'
# docker-compose -f docker-compose-mcp-aliyun.yml up -d
# 你需要修改system为你自身系统的仓库名
services:
  mcp-server-csdn-app:
    #    image: fuzhengwei/mcp-server-csdn-app:1.1
    image: registry.cn-hangzhou.aliyuncs.com/fuzhengwei/mcp-server-csdn-app:1.1
    container_name: mcp-server-csdn-app
    restart: always
    ports:
      - "8101:8101"
    volumes:
      - ./log:/data/log
    environment:
      - TZ=PRC
      - SERVER_PORT=8101
      - CSDN_API_CATEGORIES=Java场景面试宝典
      - CSDN_API_COOKIE=uuid_tt_dd=10_6645150240-1755433252939-628136; fid=20_12896762606-1755433251318-732
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
    networks:
      - my-network

  mcp-server-weixin-app:
    #    image: fuzhengwei/mcp-server-weixin-app:1.1
    image: registry.cn-hangzhou.aliyuncs.com/fuzhengwei/mcp-server-weixin-app:1.1
    container_name: mcp-server-weixin-app
    restart: always
    ports:
      - "8102:8102"
    volumes:
      - ./log:/data/log
    environment:
      - TZ=PRC
      - SERVER_PORT=8102
      - WEIXIN_API_ORIGINAL_ID=gh_e067c267e056
      - WEIXIN_API_APP_ID=wx5a228ff69e28a91f
      - WEIXIN_API_APP_SECRET=0bea03aa1310bac050aae79dd8703928
      - WEIXIN_API_TEMPLATE_ID=O8qI6gy75F-bXfPiQugInTMLA0MRzaMff9WSBb16cFk
      - WEIXIN_API_TOUSER=or0Ab6ivwmypESVp_bYuk92T6SvU
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
    networks:
      - my-network

networks:
  my-network:
    driver: bridge
```

相关的配置在2阶段已经明确阐述，可以翻看后进行配置。具体可参考；https://t.zsxq.com/Qexmh

2. 初始服务 - ChatClient

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

    private ChatModel chatModel;
    private ChatClient planningChatClient;
    private ChatClient executorChatClient;
    private ChatClient mcpToolsChatClient;

    @Resource
    private PgVectorStore vectorStore;

    public static final String CHAT_MEMORY_CONVERSATION_ID_KEY = "chat_memory_conversation_id";
    public static final String CHAT_MEMORY_RETRIEVE_SIZE_KEY = "chat_memory_response_size";

    @Before
    public void init() {
        OpenAiApi openAiApi = OpenAiApi.builder()
                .baseUrl("https://apis.itedus.cn")
                .apiKey("sk-04FRZbYTs4lavU0a066e1e0a79D249D59a42B09b3240197b")
                .completionsPath("v1/chat/completions")
                .embeddingsPath("v1/embeddings")
                .build();

        chatModel = OpenAiChatModel.builder()
                .openAiApi(openAiApi)
                .defaultOptions(OpenAiChatOptions.builder()
                        .model("gpt-4.1-mini")
                        .maxTokens(5000)
                        .toolCallbacks(new SyncMcpToolCallbackProvider(sseMcpClient_BaiduSearch(), sseMcpCl
                        .build())
                .build();

        planningChatClient = ChatClient.builder(chatModel)
                .defaultSystem("""
                        # 角色
                        你是一个智能任务规划助手，名叫 AutoAgent Planning。

                        # 说明
                        你是任务规划助手，根据用户需求，拆解任务列表，制定执行计划。重点是生成大粒度、可执行的任务步

                        # 技能
                        - 擅长将用户任务拆解为具体、独立、大粒度的任务列表
                        - 避免过度拆解，保持任务的完整性和可执行性
                        - 每个任务应该是一个完整的业务流程，而不是细碎的操作步骤

                        # 处理需求
                        ## 拆解原则
                        - 深度推理分析用户输入，识别核心需求
                        - 将复杂问题分解为3-5个大粒度的主要任务
                        - 每个任务应该包含完整的业务逻辑，可以独立完成
                        - 任务按业务流程顺序组织，逻辑清晰
                        - 避免将一个完整流程拆分成多个细小步骤

                        ## 输出格式
                        请按以下格式输出任务计划：

                        **任务规划：**
                        1. [任务1描述] - 包含完整的业务流程
                        2. [任务2描述] - 包含完整的业务流程
```

```
                3. [任务3描述] - 包含完整的业务流程
                ...

                **执行策略：**
                [整体执行策略说明]

                今天是 {current_date}。
                """)
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(50)
                                .build()
                ).build(),
                new RagAnswerAdvisor(vectorStore, SearchRequest.builder()
                        .topK(5)
                        .filterExpression("knowledge == 'article'")
                        .build()))
        .build();

// 初始化执行器客户端
executorChatClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色定义
                你是一个专业的任务执行助手，名为 AutoAgent Executor。
                你具备强大的任务执行能力和丰富的工具使用经验。

                # 核心职责
                作为智能任务执行者，你需要：
                1. 精确理解和执行规划好的任务步骤
                2. 智能调用相应的MCP工具完成具体任务
                3. 处理执行过程中的异常和错误
                4. 提供详细的执行报告和结果反馈

                # 专业技能
                ## 任务执行能力
                - 深度理解任务步骤的具体要求和目标
                - 智能选择和调用合适的MCP工具
                - 处理工具调用的参数配置和结果解析
                - 监控执行进度并提供实时反馈

                ## 错误处理机制
                - 识别和分类执行过程中的各种错误
                - 实施智能重试和降级策略
                - 提供详细的错误诊断和解决建议
                - 确保任务执行的稳定性和可靠性

                ## 标准化输出
                严格按照以下结构化格式输出执行报告：

                **📋任务执行报告**
                - 任务名称：[步骤名称]
                - 执行状态：[成功/失败/部分成功]
                - 开始时间：[时间戳]
                - 结束时间：[时间戳]
                - 执行耗时：[毫秒]

                **🔧工具调用详情**
                - 使用工具：[工具名称列表]
                - 调用次数：[具体次数]
                - 成功率：[百分比]
                - 关键参数：[重要参数配置]

                **📊执行结果**
```

```
                - 主要成果：[具体完成的内容]
                - 数据输出：[生成的数据或文件]
                - 质量评估：[结果质量分析]

                **⚠️异常处理**
                - 遇到问题：[具体问题描述]
                - 处理策略：[采用的解决方案]
                - 影响评估：[对整体任务的影响]

                今天是 {current_date}。
                """)
        .defaultAdvisors(
                PromptChatMemoryAdvisor.builder(
                        MessageWindowChatMemory.builder()
                                .maxMessages(20)
                                .build()
                ).build(),
                new RagAnswerAdvisor(vectorStore, SearchRequest.builder()
                        .topK(5)
                        .filterExpression("knowledge == 'article'")
                        .build()))
        .build();

// 初始化MCP工具客户端
mcpToolsChatClient = ChatClient.builder(chatModel)
        .defaultSystem("""
                # 角色定义
                你是一个专业的MCP（Model Context Protocol）工具管理专家，名为 MCP Tools Manager。
                你具备深度的MCP协议理解和丰富的工具集成经验。

                # 核心职责
                作为MCP工具生态的管理者，你需要：
                1. 精确识别和分类所有可用的MCP服务工具
                2. 深度分析工具的功能边界、参数规范和使用场景
                3. 基于用户意图智能匹配最优工具组合
                4. 提供符合MCP标准的工具调用指导

                # 专业技能
                ## 工具发现与分析
                - 实时扫描并索引所有注册的MCP服务端点
                - 解析工具的JSON Schema定义和元数据
                - 识别工具间的依赖关系和协作模式
                - 评估工具的可靠性、性能和安全性

                ## 智能推荐引擎
                - 基于语义理解匹配用户需求与工具能力
                - 考虑工具的执行成本、响应时间和成功率
                - 提供备选方案和降级策略
                - 优化工具调用链的执行顺序

                ## 标准化输出
                严格按照以下结构化格式输出：

                **🔧可用MCP工具清单**

                序号 | 工具名称 | 服务类型 | 核心功能 | 参数要求 | 可靠性评级
                -----|---------|---------|---------|---------|----------
                1    | [name]  | [type]  | [desc]  | [params]| [rating]

                **🎯智能推荐方案**
                - 主推工具：[工具名] - 匹配度：[百分比] - 理由：[具体原因]
                - 备选工具：[工具名] - 适用场景：[具体场景]
                - 组合策略：[多工具协作方案]
```

```
                        **📋执行标准指南**
                        - 调用顺序：[步骤1] → [步骤2] → [步骤3]
                        - 参数配置：[关键参数及其推荐值]
                        - 错误处理：[异常情况的处理策略]
                        - 性能优化：[提升执行效率的建议]

                        **⚠️注意事项**
                        - 安全约束：[权限要求和安全限制]
                        - 资源消耗：[预期的计算和网络开销]
                        - 兼容性：[版本要求和环境依赖]
                        """)
                .defaultAdvisors(
                        PromptChatMemoryAdvisor.builder(
                                MessageWindowChatMemory.builder()
                                        .maxMessages(30)
                                        .build()
                        ).build())
                .build();
    }

}
```

初始化3个客户端，planningChatClient - 拆解任务、mcpToolsChatClient - 准确获取 MCP 工具、executorChatClient - 执行规划过程。

这些话术是可以先自己编写目标要做什么，之后在使用 AI 来优化话术，多尝试几次得到的结果。为了内容的准确性，可以在多执行几次。

3. 规划分析

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

    @Test
    public void test_agent() {
        String userRequest = """
                我需要你帮我生成一篇文章，要求如下：

                    1. 场景为互联网大厂java求职者面试
                    2. 提问的技术栈如下：

                        核心语言与平台：Java SE (8/11/17), Jakarta EE (Java EE), JVM
                        构建工具：Maven, Gradle, Ant
                        Web框架：Spring Boot, Spring MVC, Spring WebFlux, Jakarta EE, Micronaut, Quarkus, Pl
                        数据库与ORM：Hibernate, MyBatis, JPA, Spring Data JDBC, HikariCP, C3P0, Flyway, Liqu
                        测试框架：JUnit 5, TestNG, Mockito, PowerMock, AssertJ, Selenium, Cucumber
                        微服务与云原生：Spring Cloud, Netflix OSS (Eureka, Zuul), Consul, gRPC, Apache Thrift
                        安全框架：Spring Security, Apache Shiro, JWT, OAuth2, Keycloak, Bouncy Castle
                        消息队列：Kafka, RabbitMQ, ActiveMQ, JMS, Apache Pulsar, Redis Pub/Sub
                        缓存技术：Redis, Ehcache, Caffeine, Hazelcast, Memcached, Spring Cache
                        日志框架：Log4j2, Logback, SLF4J, Tinylog
                        监控与运维：Prometheus, Grafana, Micrometer, ELK Stack, New Relic, Jaeger, Zipkin
                        模板引擎：Thymeleaf, FreeMarker, Velocity, JSP/JSTL
                        REST与API工具：Swagger/OpenAPI, Spring HATEOAS, Jersey, RESTEasy, Retrofit
                        序列化：Jackson, Gson, Protobuf, Avro
                        CI/CD工具：Jenkins, GitLab CI, GitHub Actions, Docker, Kubernetes
                        大数据处理：Hadoop, Spark, Flink, Cassandra, Elasticsearch
                        版本控制：Git, SVN
                        工具库：Apache Commons, Guava, Lombok, MapStruct, JSch, POI
```

```
                AI：Spring AI, Google A2A, MCP（模型上下文协议）, RAG（检索增强生成）, Agent（智能代理）
                其他：JUnit Pioneer, Dubbo, R2DBC, WebSocket
            3. 提问的场景方案可包括但不限于：音视频场景,内容社区与UGC,AIGC,游戏与虚拟互动,电商场景,本地生活
            4. 按照故事场景，以严肃的面试官和搞笑的水货程序员谢飞机进行提问，谢飞机对简单问题可以回答出来，
            5. 每次进行3轮提问，每轮可以有3-5个问题。这些问题要有技术业务场景上的衔接性，循序渐进引导提问。
            6. 提问后把问题的答案详细的，写到文章最后，讲述出业务场景和技术点，让小白可以学习下来。

            根据以上内容，不要阐述其他信息，请直接提供：文章标题（需要含带技术点）、文章内容、文章标签（多个

            将以上内容发布文章到CSDN

            之后进行，微信公众号消息通知，平台：CSDN、主题：为文章标题、描述：为文章简述、跳转地址：为发布文
        """;

log.info("=== 自动Agent开始执行 ===");
log.info("用户请求: {}", userRequest);

Map<String, Object> executionContext = new HashMap<>();
executionContext.put("userRequest", userRequest);
executionContext.put("startTime", System.currentTimeMillis());
executionContext.put("status", "INITIALIZING");

try {
    // 第一步：获取可用的MCP工具和使用方式（仅分析，不执行用户请求）
    log.info("\n--- 步骤1: MCP工具能力分析（仅分析阶段，不执行用户请求） ---");
    executionContext.put("status", "ANALYZING_TOOLS");
    String mcpToolsAnalysis = executeWithRetry(() -> getMcpToolsCapabilities(userRequest), "MCP工具分
    log.info("MCP工具分析结果（仅分析，未执行实际操作）: {}", mcpToolsAnalysis);
    executionContext.put("mcpToolsAnalysis", mcpToolsAnalysis);

    // 第二步：根据用户请求和MCP能力规划执行步骤
    log.info("\n--- 步骤2: 规划执行步骤 ---");
    executionContext.put("status", "PLANNING");
    String planningResult = executeWithRetry(() -> planExecutionSteps(userRequest, mcpToolsAnalysis
    log.info("规划结果: {}", planningResult);
    executionContext.put("planningResult", planningResult);

    // 第三步：解析规划结果，将每个步骤存储到map中
    log.info("\n--- 步骤3: 解析规划步骤 ---");
    executionContext.put("status", "PARSING_STEPS");
    Map<String, String> stepsMap = parseExecutionSteps(planningResult);
    log.info("解析的步骤数量: {}", stepsMap.size());
    for (Map.Entry<String, String> entry : stepsMap.entrySet()) {
        log.info("步骤 {}: {}", entry.getKey(), entry.getValue().substring(0, Math.min(100, entry.ge
    }
    executionContext.put("stepsMap", stepsMap);

    // 第四步：按顺序执行规划步骤
    log.info("\n--- 步骤4: 按顺序执行规划步骤 ---");
    executionContext.put("status", "EXECUTING_STEPS");
    executeStepsInOrder(stepsMap, executionContext);

    // 执行完成
    log.info("\n=== Agent执行完成 ===");
    executionContext.put("status", "COMPLETED");
    executionContext.put("endTime", System.currentTimeMillis());
    long totalTime = (Long) executionContext.get("endTime") - (Long) executionContext.get("startTim
    log.info("总执行时间: {} ms", totalTime);

} catch (Exception e) {
    log.error("Agent执行过程中发生错误", e);
    executionContext.put("status", "ERROR");
    executionContext.put("error", e.getMessage());
    executionContext.put("endTime", System.currentTimeMillis());
```

```
        }
    }

}
```

根据用户的提问，这里有4个步骤；

第一步：获取可用的MCP工具和使用方式（仅分析，不执行用户请求）

第二步：根据用户请求和MCP能力规划执行步骤

第三步：解析规划结果，将每个步骤存储到map中

第四步：按顺序执行规划步骤

具体的详细工程代码，还可以在参考下课程代码来看。测试验证部分可以看本节视频。

四、读者作业

简单作业：结合本节和前面实现的 Ai Agent 多思考分析 Agent 的执行过程，思路打开，你会更多的想法。

复杂作业：尝试在理解本节 Ai Agent 实现后，按照之前的设计方式，在 domain 领域下做功能实现。