《Ai Agent》第3-7节：动态实例化客户端API

来自： 码农会锁

小傅哥

2025年07月05日 10:44

本章重点： ★★★☆☆
课程视频：https://t.zsxq.com/4r5r4
代码分支：https://gitcode.net/KnowledgePlanet/ai-agent-station-study/-/tree/3-7-register-bean-client-api
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！😄

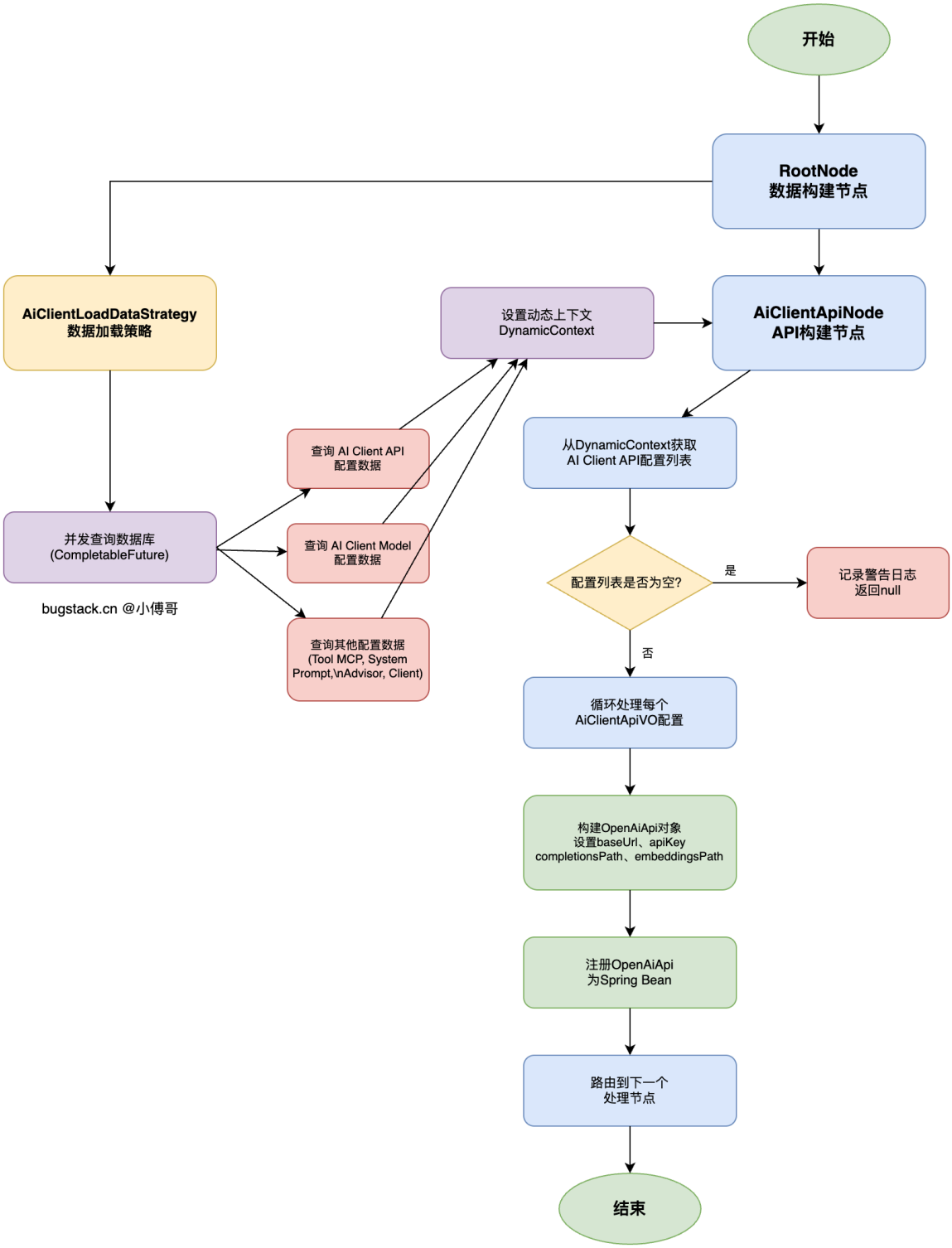
一、本章诉求

完善数据加载操作，动态实例化客户端API（ai_client_api）并注册到 Spring 容器。

这是整个 armory 动态装配 Ai Agent 节点的第一步，涉及到了数据的获取，对象的创建和 Spring 容器的 Bean 对象注册。能看懂本节的操作，基本后续一直到整个 Ai Agent 构建也就都可以看懂了。

二、功能流程

如图，客户端API实例化过程设计；

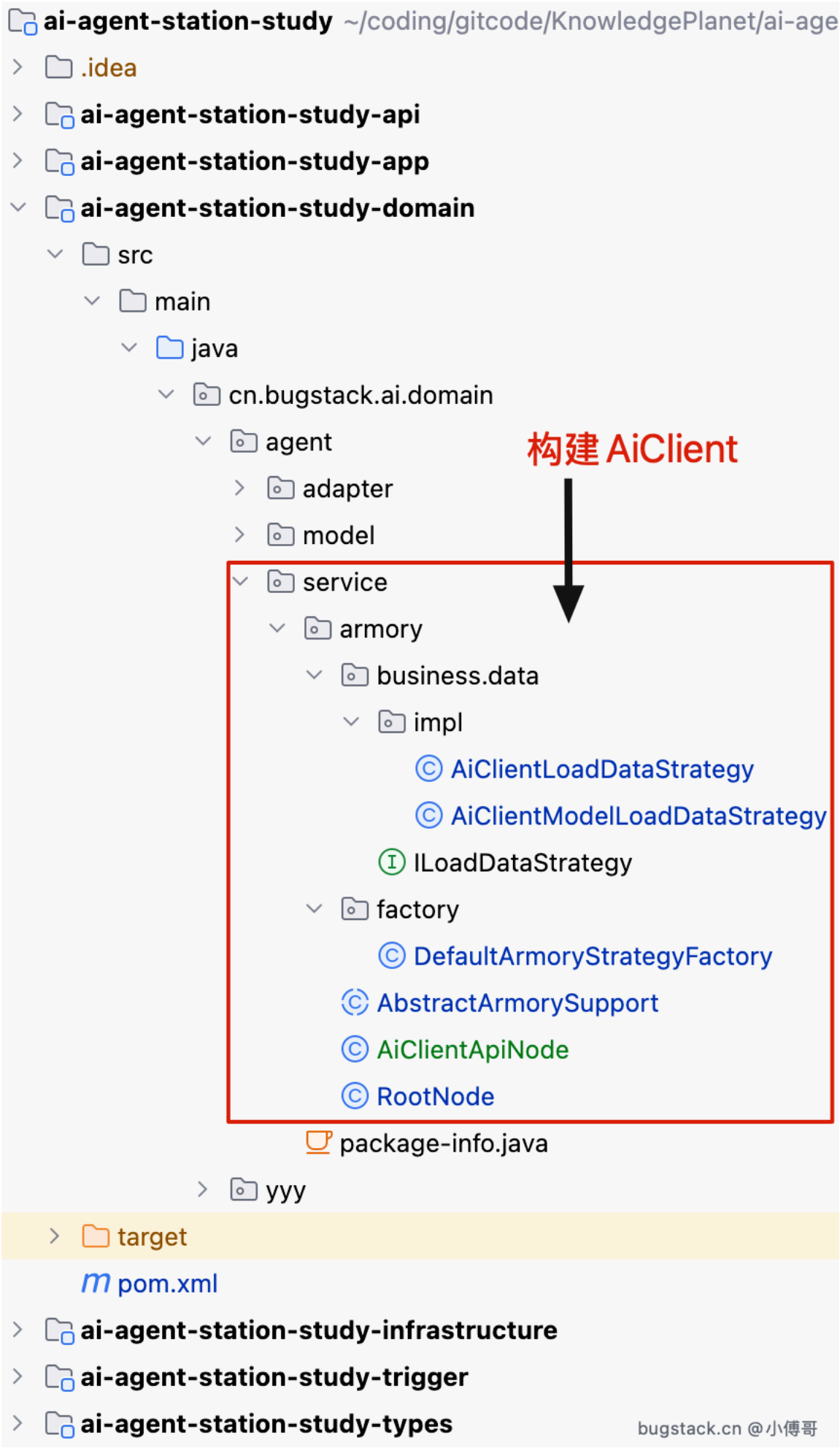


首先，整个 AI Agent 的实例化过程，就是各项组件的创建和组装的过程。为了让整体的实现代码更易于维护，我们把这样的创建过程，通过 规则树的方式 进行串联实现。这种设计模式的优势在于：模块化设计、易于扩展、代码复用度高。

之后，从开始节点看，依次执行，数据构建节点、API构建节点。在 API 构建的过程中，会检查上下文中是否存在已经从数据库获取的数据，之后依次循环构建并注册到 Spring 容器。

三、编码实现

1. 工程结构



如图，先来构建第一个 Ai Agent 使用到的 API 节点。AiClientApiNode 的构建会，使用到 RootNode 节点加载的数据。

AiClientApiNode 的构建完成后，会使用到 AbstractArmorySupport 中提供的注册 Bean 到 Spring 容器的方法。这部分是对 Spring 源码的一个使用。

2. Spring Bean 容器

类：AbstractArmorySupport

```
protected synchronized <T> void registerBean(String beanName, Class<T> beanClass, T beanInstance) {
    DefaultListableBeanFactory beanFactory = (DefaultListableBeanFactory) applicationContext.getAutowireCapa
    // 注册Bean
    BeanDefinitionBuilder beanDefinitionBuilder = BeanDefinitionBuilder.genericBeanDefinition(beanClass, ()
    BeanDefinition beanDefinition = beanDefinitionBuilder.getRawBeanDefinition();
    beanDefinition.setScope(BeanDefinition.SCOPE_SINGLETON);
    // 如果Bean已存在，先移除
    if (beanFactory.containsBeanDefinition(beanName)) {
        beanFactory.removeBeanDefinition(beanName);
    }
    // 注册新的Bean
    beanFactory.registerBeanDefinition(beanName, beanDefinition);
    log.info("成功注册Bean: {}", beanName);
}

protected <T> T getBean(String beanName) {
    return (T) applicationContext.getBean(beanName);
}
```

第一步：获取Bean工厂，DefaultListableBeanFactory 是Spring容器的核心实现类。通过它可以动态管理Bean的生命周期。

第二步：构建Bean定义，使用 BeanDefinitionBuilder 创建Bean定义。genericBeanDefinition(beanClass, () -> beanInstance) 指定Bean类型和实例供应商。设置作用域为单例模式（ SCOPE_SINGLETON ）

第三步：处理Bean冲突，检查是否已存在同名Bean，如果存在，先移除旧的Bean定义，确保新Bean能够正确注册。

第四步：注册新Bean，将新的Bean定义注册到Spring容器。

3. 节点构建(AiClientApiNode)

类：AiClientApiNode

```
@Slf4j
@Service
public class AiClientApiNode extends AbstractArmorySupport {

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

        List<AiClientApiVO> aiClientApiList = dynamicContext.getValue(AiAgentEnumVO.AI_CLIENT_API.getDataNa

        if (aiClientApiList == null || aiClientApiList.isEmpty()) {
            log.warn("没有需要被初始化的 ai client api");
            return null;
        }

        for (AiClientApiVO aiClientApiVO : aiClientApiList) {
            // 构建 OpenAiApi
            OpenAiApi openAiApi = OpenAiApi.builder()
                    .baseUrl(aiClientApiVO.getBaseUrl())
                    .apiKey(aiClientApiVO.getApiKey())
                    .completionsPath(aiClientApiVO.getCompletionsPath())
                    .embeddingsPath(aiClientApiVO.getEmbeddingsPath())
                    .build();
```

```
            // 注册 OpenAiApi Bean 对象
            registerBean(AiAgentEnumVO.AI_CLIENT_API.getBeanName(aiClientApiVO.getApiId()), OpenAiApi.class
        }

        return router(requestParameter, dynamicContext);
    }

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

}
```

doApply 是处理业务流程的方法区，在这里首先通过上下文获取加载的 AIClientApi 数据。如果数据为空则返回为null。后续这块也可以处理为路由到下一个节点继续处理其他节点实例化。

接下来，for循环的过程，就是不断的创建 OpenAiApi 对象，之后注册到 Spring 容器中。完成后，执行 router 路由到下一个节点。

router 执行后，会走到 get 方法，目前设置的是 defaultStrategyHandler，也就是不执行下一个节点。后续会随着功能开发来修改。

4. 节点路由(RootNode)

```
@Slf4j
@Service
public class RootNode extends AbstractArmorySupport {

    private final Map<String, ILoadDataStrategy> loadDataStrategyMap;

    @Resource
    private AiClientApiNode aiClientApiNode;

    public RootNode(Map<String, ILoadDataStrategy> loadDataStrategyMap) {
        this.loadDataStrategyMap = loadDataStrategyMap;
    }

    @Override
    protected void multiThread(ArmoryCommandEntity requestParameter, DefaultArmoryStrategyFactory.DynamicCo
        // 获取命令：不同的命令类型，对应不同的数据加载策略
        String commandType = requestParameter.getCommandType();

        // 获取策略
        AiAgentEnumVO aiAgentEnumVO = AiAgentEnumVO.getByCode(commandType);
        String loadDataStrategyKey = aiAgentEnumVO.getLoadDataStrategy();

        // 加载数据
        ILoadDataStrategy loadDataStrategy = loadDataStrategyMap.get(loadDataStrategyKey);
        loadDataStrategy.loadData(requestParameter, dynamicContext);
    }

    @Override
    protected String doApply(ArmoryCommandEntity requestParameter, DefaultArmoryStrategyFactory.DynamicCont
        log.info("Ai Agent 构建，数据加载节点 {}", JSON.toJSONString(requestParameter));
        return router(requestParameter, dynamicContext);
    }

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

```
}
```

注意，RootNode 节点的要修改 get 方法，路由到 aiClientApiNode。也就是数据加载完成后，要走到下一个节点进行 api 构建操作。

router 的操作是在设计模式模板类里，可以到上一层的方法中查看具体的路由操作。这类的代码，都可以通过 debug 方式，一步步调试学习，调试后也就可以知道这些流程都是什么样的了。而且调试也是程序员必须学习的东西。

四、功能测试

类：ai-agent-station-study-app/cn.bugstack.ai.test.domain.AgentTest

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

    @Resource
    private DefaultArmoryStrategyFactory defaultArmoryStrategyFactory;

    @Resource
    private ApplicationContext applicationContext;

    @Test
    public void test_aiClientApiNode() 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());

        OpenAiApi openAiApi = (OpenAiApi) applicationContext.getBean(AiAgentEnumVO.AI_CLIENT_API.getBeanNam

        log.info("测试结果：{}", openAiApi);
    }

}
```

测试结果

```
25-07-05.07:52:24.536 [pool-2-thread-1 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client_api) [3001]
25-07-05.07:52:24.536 [pool-2-thread-2 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client_model) [300
25-07-05.07:52:24.536 [pool-2-thread-3 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client_tool_mcp) [
25-07-05.07:52:24.537 [pool-2-thread-4 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client_system_promp
25-07-05.07:52:24.537 [pool-2-thread-5 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client_advisor) [3
25-07-05.07:52:24.537 [pool-2-thread-6 ] INFO  AiClientLoadDataStrategy - 查询配置数据(ai_client) [3001]
25-07-05.07:52:24.548 [pool-2-thread-6 ] INFO  HikariDataSource       - MainHikariPool - Starting...
25-07-05.07:52:24.640 [pool-2-thread-6 ] INFO  HikariPool             - MainHikariPool - Added connection c
25-07-05.07:52:24.641 [pool-2-thread-6 ] INFO  HikariDataSource       - MainHikariPool - Start completed.
25-07-05.07:52:24.715 [main            ] INFO  RootNode               - Ai Agent 构建，数据加载节点 {"command
25-07-05.07:52:24.715 [main            ] INFO  AiClientApiNode        - Ai Agent 构建，API 构建节点 {"commanc
25-07-05.07:52:34.234 [main            ] INFO  AbstractArmorySupport  - 成功注册Bean: ai_client_api_1001
25-07-05.07:52:40.241 [main            ] INFO  AgentTest              - 测试结果：org.springframework.ai.ope
```

测试结果可以看到，OpenAiApi 已经构建完成。

五、读者作业

简单作业：了解本节的目的和实现的过程，可以自己复刻出来。并运行出结果。

复杂作业：尝试完成下一个节点 mode 的构建，并从api构建节点路由到下一个节点。