ABCoder Java 扩展和应用落地

ABCoder Java extensions and application implementation

演讲人: 马跃伟

SPEAKER: YUEWEI MA(Mars)

目录 | Contents

01

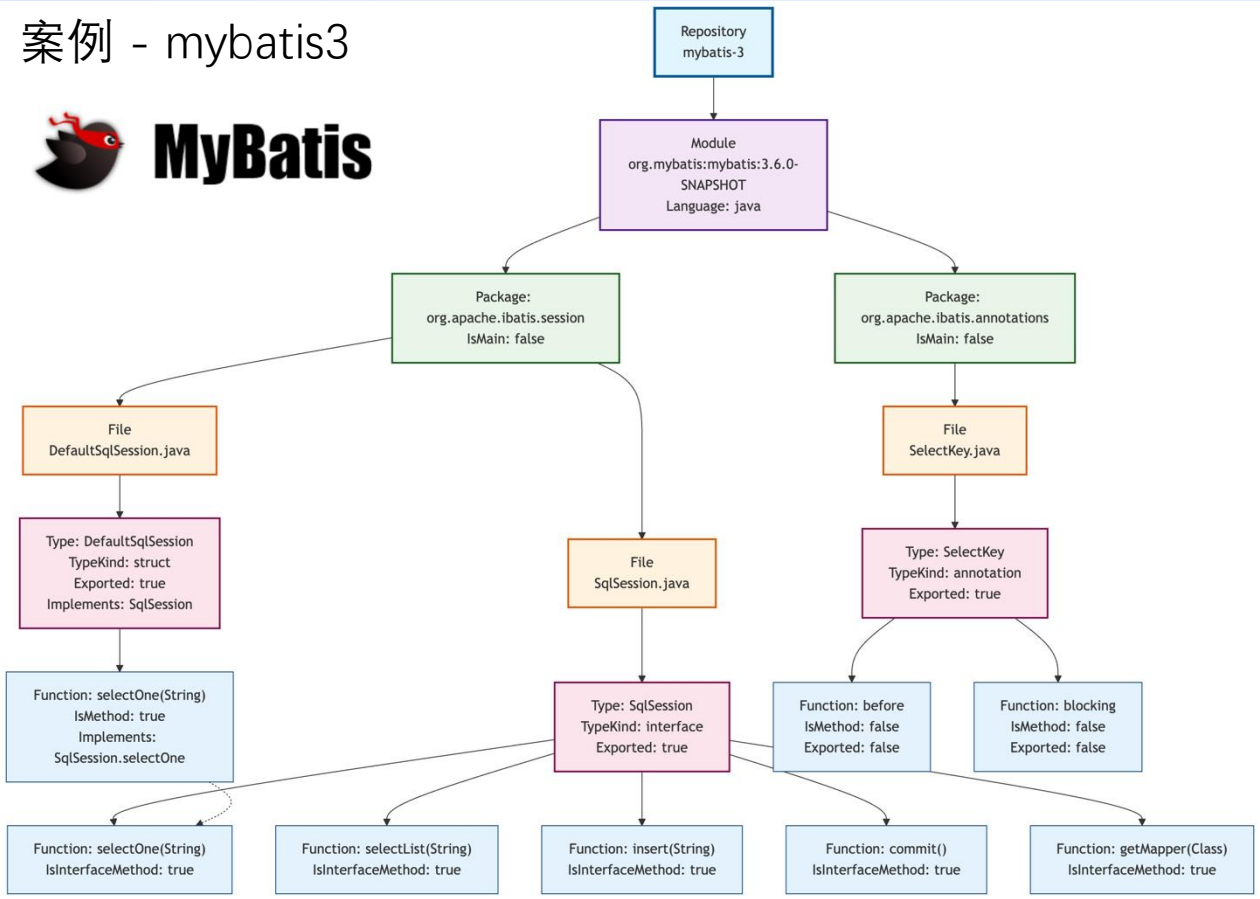
ABCoder 介绍

AI-Based Coder : index 方式的 codebase context 实现

ABCoder 的介绍和应用

index 方式的 codebase context 实现

- 代码理解和分析的复杂性问题
- Code To 统一的抽象语法树（UniAST）
- 结合大语言模型的推理能力，实现深度的代码理解和分析




```
"Modules": {
  "org.mybatis:mybatis:3.6.0-SNAPSHOT": {
    "Packages": {
      "org.apache.ibatis.session.defaults": {
        "Functions": {
          "DefaultSqlSession.selectOne": {
            "Exported": true,
            "IsMethod": true,
            "IsInterfaceMethod": false,
            "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
            "PkgPath": "org.apache.ibatis.session.defaults",
            "Name": "DefaultSqlSession.selectOne",
            "File": "src/main/java/org/apache/ibatis/session/defaults/DefaultSqlSession.java",
            "Line": 71,
            "StartOffset": 2448,
            "EndOffset": 2934,
            "Content": "@Override\n  public \u003cT\u003e T selectOne(String statement, Object parameter) {\n    // Popular vote was to return null on 0 results
             and throw exception on too many.\n    List\u003cT\u003e list = this.selectList(statement, parameter);\n    if (list.size() == 1) {\n      return
             list.get(0);\n    }\n    if (list.size() \u003e 1) {\n      throw new TooManyResultsException(\n          \"Expected one result (or null) to be
             returned by selectOne(), but found: \" + list.size());\n    } else {\n      return null;\n    }\n  }",
            "Signature": "@Override\n  public \u003cT\u003e T selectOne(String statement, Object parameter)",
            "Receiver": {
              "IsPointer": false,
              "Type": {
                "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
                "PkgPath": "org.apache.ibatis.session.defaults",
                "Name": "DefaultSqlSession"
              }
            },
            "Params": [
              {
                "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
                "PkgPath": "org.apache.ibatis.session.defaults",
                "Name": "String",
                "File": "src/main/java/org/apache/ibatis/session/defaults/DefaultSqlSession.java",
                "Line": 72,
                "StartOffset": 2483,
                "EndOffset": 2489
              },
              {
                "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
                "PkgPath": "org.apache.ibatis.session.defaults",
                "Name": "Object",
                "File": "src/main/java/org/apache/ibatis/session/defaults/DefaultSqlSession.java",
                "Line": 72,
                "StartOffset": 2501,
                "EndOffset": 2507
              }
            ],
            "Results": [
              {
                "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
                "PkgPath": "org.apache.ibatis.session.defaults",
                "Name": "T",
                "File": "src/main/java/org/apache/ibatis/session/defaults/DefaultSqlSession.java",
                "Line": 72,
                "StartOffset": 2471,
                "EndOffset": 2472
              }
            ],
            "MethodCalls": [
              {
                "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT",
                "PkgPath": "org.apache.ibatis.session.defaults",
                "Name": "selectList",
                "File": "src/main/java/org/apache/ibatis/session/defaults/DefaultSqlSession.java",
                "Line": 74,
                "StartOffset": 2630,
                "EndOffset": 2640
              }
            ]
          },
          "DefaultSqlSession.update(String)": {
            "Exported": true,
            "IsMethod": true,
            "IsInterfaceMethod": false,
            "ModPath": "org.mybatis:mybatis:3.6.0-SNAPSHOT".
```

ABCoder 的介绍和应用

index 方式的 codebase context 实现

搭建知识库

序号	档 / 文档ID	ChunkType string	Title 索引 string	NodeID string	StartLine int64	EndLine int64	Content 索引 string	操作
#2	udwego_hertz_project_doc_chunks.csv sto_gen_doc_id-18032982281810955643	h2_section	2. 项目概述		7	10	## 2. 项目概述	
#3	udwego_hertz_project_doc_chunks.csv sto_gen_doc_id-18032982281810955643	package_details	4. github.com/cloudwego/hertz/internal/bytestr + 4.1 Package 描述 + 4.2 Structure		469	579		
#4	udwego_hertz_project_doc_chunks.csv sto_gen_doc_id-18032982281810955643	type_definition	12.3.1 JSONMarshaler	github.com/cloudwego/hertz/pkg/app/server/render#JSONMarshaler	3843	3859		
#5	udwego_hertz_project_doc_chunks.csv sto_gen_doc_id-18032982281810955643	h5_section	34.3.12.20 Request.Method	github.com/cloudwego/hertz/pkg/protocol#Request.Method	37614	37651		
#6	udwego_hertz_project_doc_chunks.csv sto_gen_doc_id-18032982281810955643	h5_section	34.3.12.21 Request.PostArgString	github.com/cloudwego/hertz/pkg/protocol#Request.PostArgString	37652	37690	##### 34.3.12.21 Request.PostArgString 编辑 删除 描述: Request.PostArgStrin...	

34.3.12.21 Request.PostArgString

描述: Request.PostArgString 是一个方法，用于获取 HTTP 请求中 POST 参数的查询字符串格式的字节切片。它通过调用接收者 Request 结构体中的 postArgs 字段的 QueryString 方法来实现这一功能。

该函数的主要功能和用途：
- 提供便捷方式访问请求体中的 POST 参数
- 返回已序列化为查询字符串格式的 POST 参数

该函数的具体实现过程：
1. 直接调用接收者 req 的 postArgs 字段的 QueryString 方法
2. 返回该方法生成的查询字符串字节切片

入参:
- Null

出参:
- []byte 类型，包含序列化后的 POST 参数查询字符串

指导技术方案和代码生成

代码块 YAML 取消自动换行 复制

```yaml
- task: "为Flight数据模型添加comfortInfo字段"
  file: "models/flight.go"
  action: "modify"

- task: "创建ComfortInfo数据源的客户端"
  file: "clients/comfort_client.go"
  action: "create"

- task: "在FlightService中增加调用comfort_client的逻辑"
  file: "services/flight_service.go"
  action: "modify"

- task: "更新GetFlightDetails API的返回值"
  file: "controllers/flight_controller.go"
  action: "modify"

- task: "为新增逻辑编写单元测试"
  file: "services/flight_service_test.go"
  action: "create_or_modify"
```

跨语言重构-半空 Go2Rust

Golang 原始实现	「半空」意译效果

```go
// customizeRegister registers customize routers.
func customizedRegister(r *server.Hertz) {
    r.GET( relativePath: "/ping", hertz_handler.Ping)

    // your code ...
    r.NoRoute(func(ctx context.Context, c *app.RequestContext) { // used for HTTP 404
        demoapi.SendResponse(c, errno.ServiceErr, data: nil)
    })
    r.NoMethod(func(ctx context.Context, c *app.RequestContext) { // used for HTTP 405
        demoapi.SendResponse(c, errno.ServiceErr, data: nil)
    })
}
```

customizedRegister()

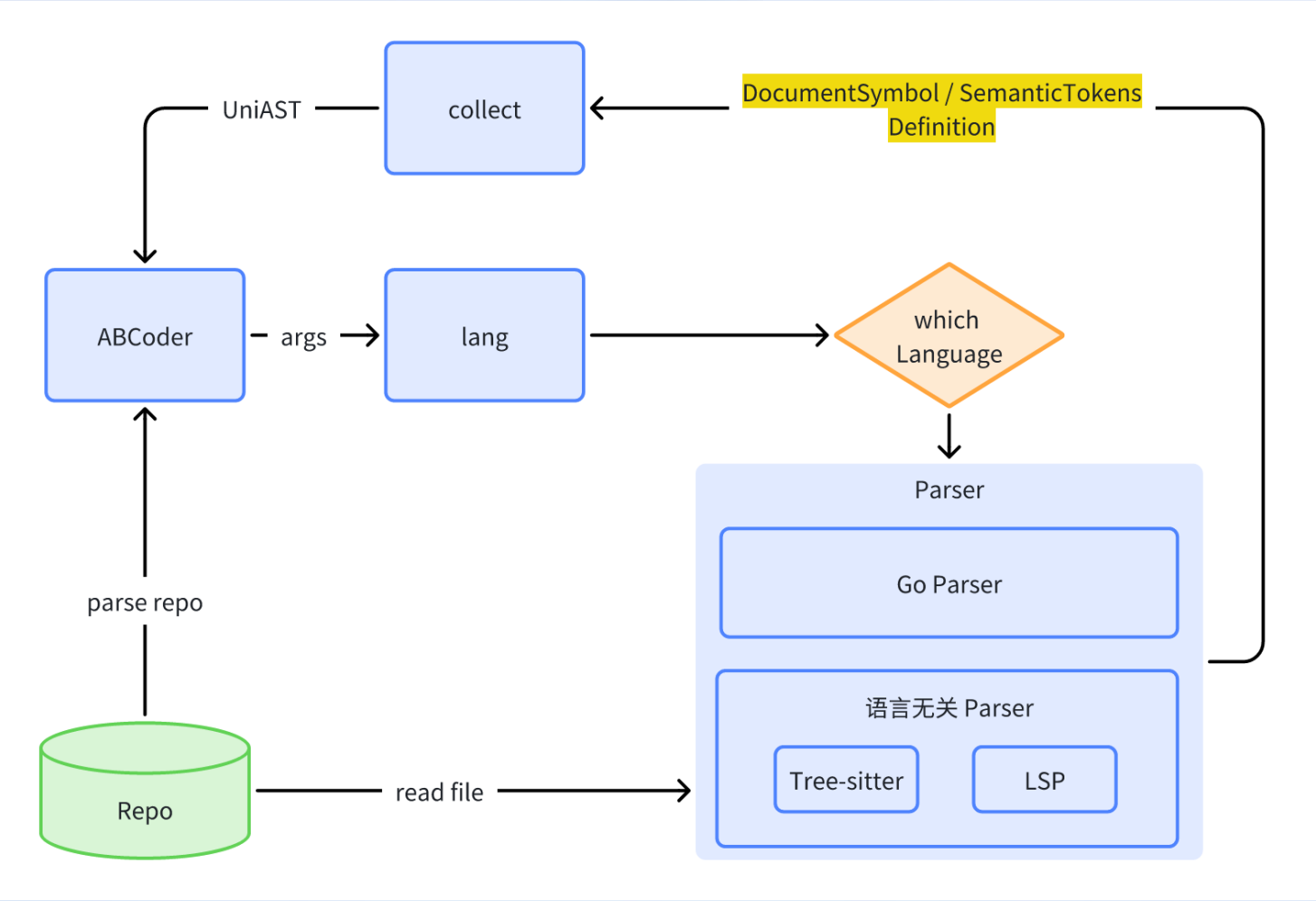
```rust
// customized_register registers custom routers.
pub fn customized_register() -> Router {
    Router::new() Router
        // 1. 注册 GET 请求的 '/ping' 路由，并指定处理函数
        .route(uri: "/ping", method_router: get_service(service_fn(hertz_handler::ping))) Router
        // 2. 为未匹配到任何路由的请求注册处理函数
        .fallback(handler: handle_uri_not_found) Router
        // 3. 为使用了不被允许的方法的请求注册处理函数
        .fallback(handler: handle_method_not_allowed)
}

// fallback handler for METHOD NOT ALLOWED
async fn handle_method_not_allowed() -> (StatusCode, &'static str) {
    demoapi::send_response(errno::SERVICE_ERR, _msg: "Method Not Allowed")
}
```

customized_register()

CR、bug修复、知识问答...

ABCoder 的架构和实现

多语言 parser



Go Parser 的痛点

- 语言强耦合
- 环境依赖性
- 性能问题
- 扩展性差

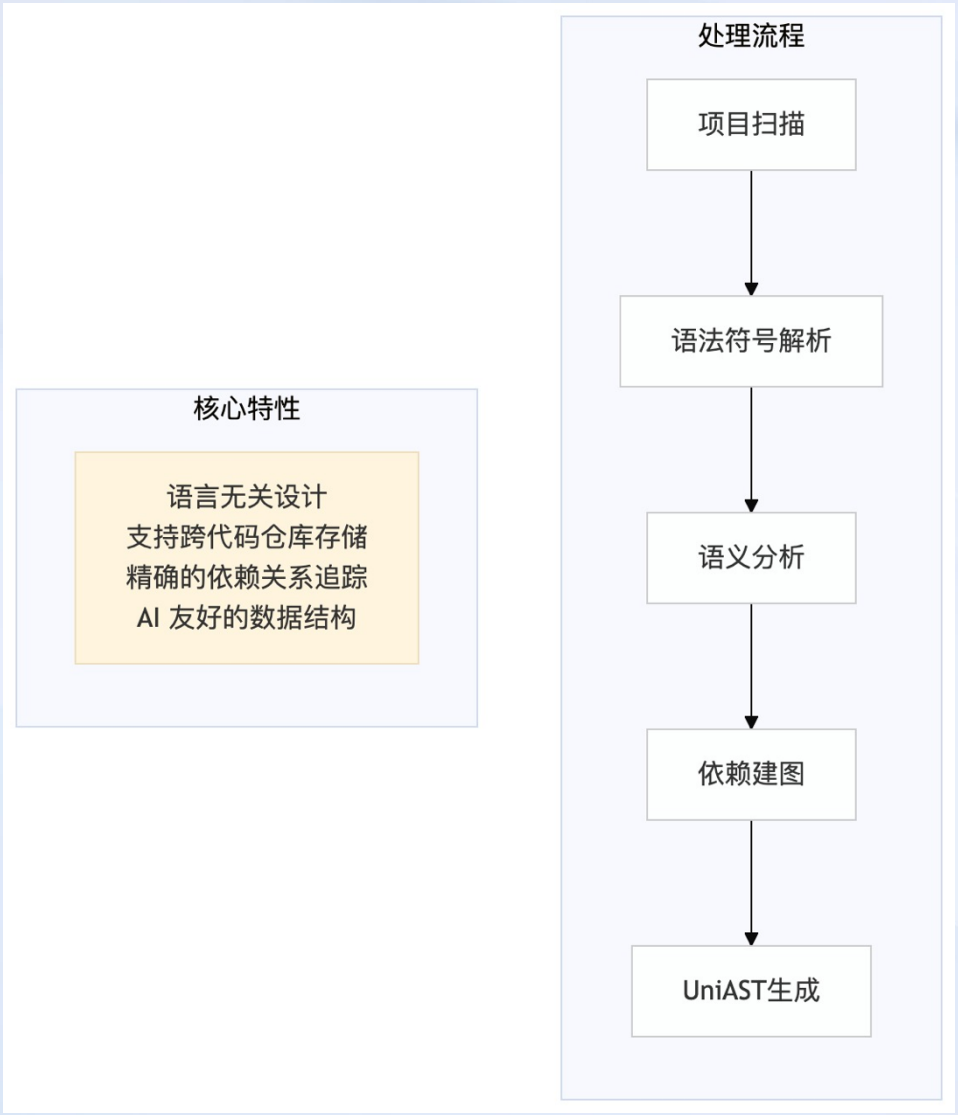
Tree-sitter + LSP 实现 优点

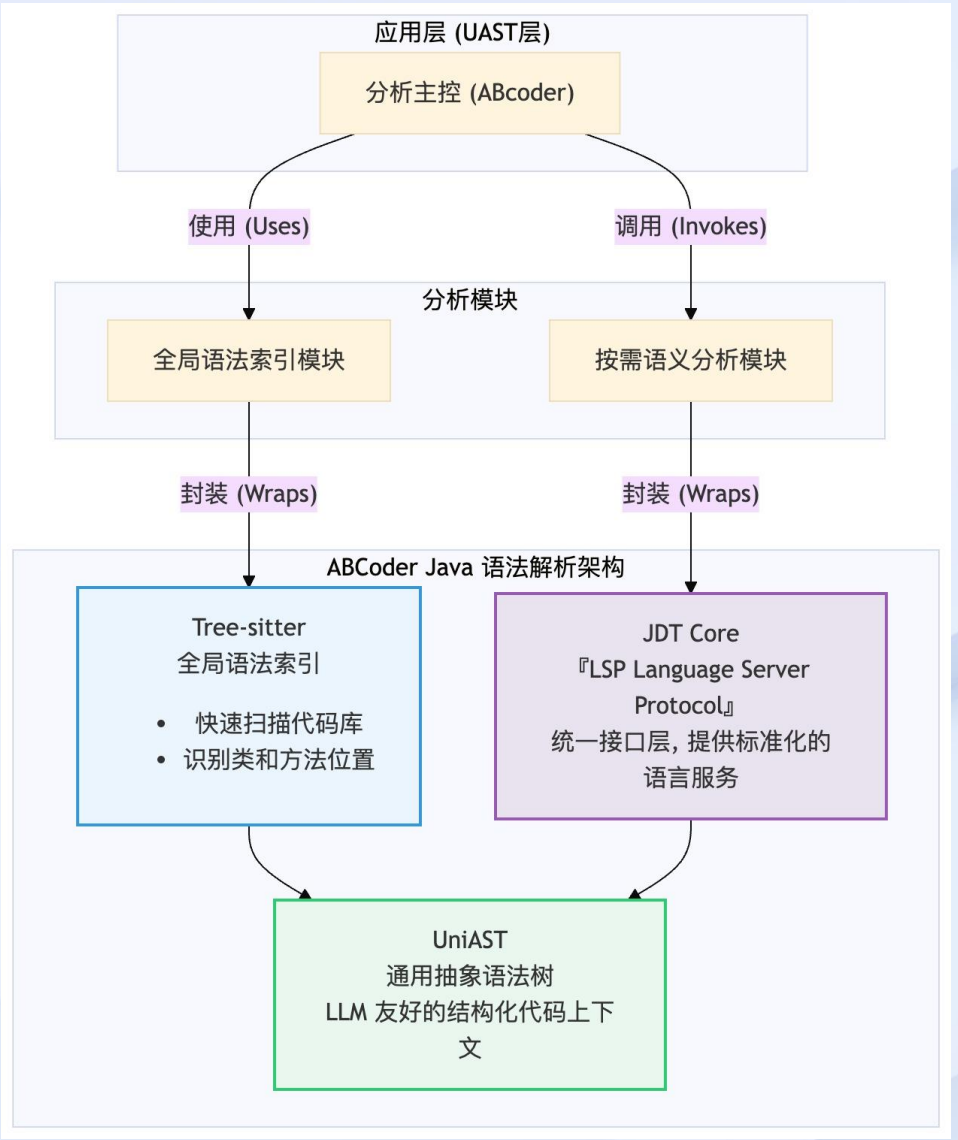
- 架构设计统一
- 仅需 LSP 语言服务器
- 性能优势
- 插件化语言支持

02 构建AI专用的代码地图

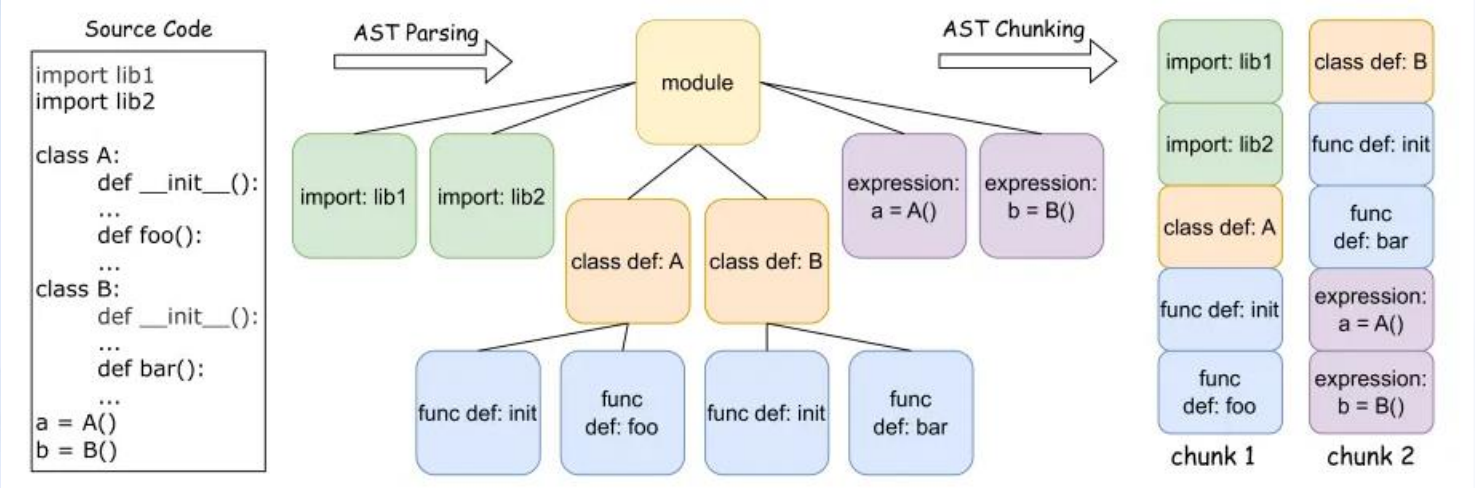
Java 为例的 parser 实现

ABCoder的 多语言 parser 实现思路





使用Tree-sitter进行快速文件解析，理解代码的基本符号



语法精准解析

Tree-sitter基于词法解析技术，精准识别Java语法结构，确保代码骨架提取无误。

语言结构提取

提取类、方法、变量等关键节点，构建清晰的代码逻辑视图。

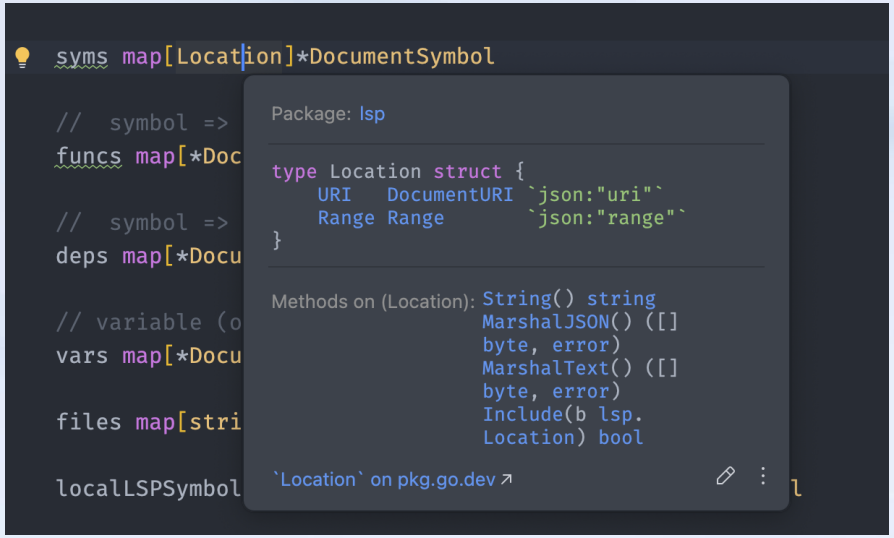
高效性能表现

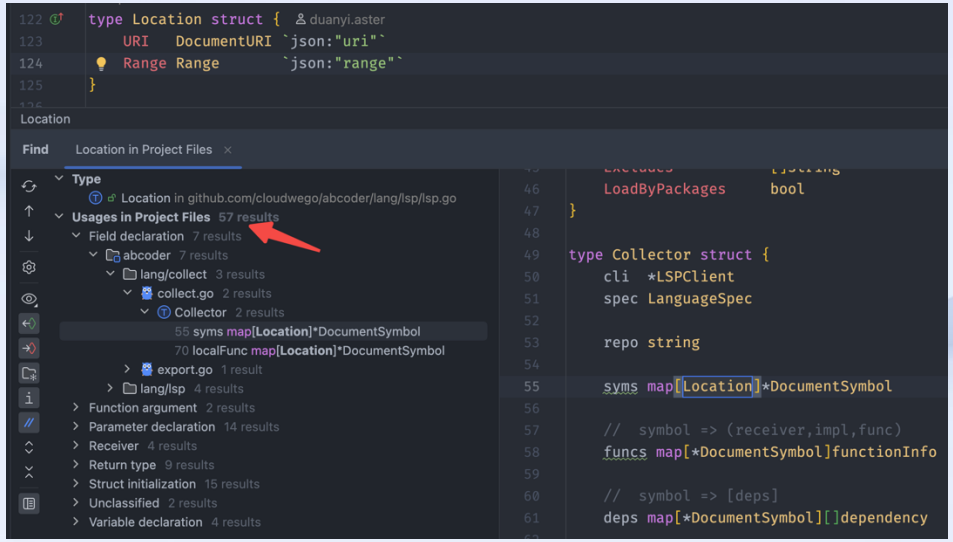
支持快速遍历大型文件，毫秒级响应，满足实时分析需求。

AI友好输出

生成结构化AST数据，便于后续处理与跨文件关联分析。

通过LSP关联所有符号，了解跨文件的关系网





符号全局索引

利用LSP建立跨文件符号引用关系，实现类、方法、变量的精准追踪。

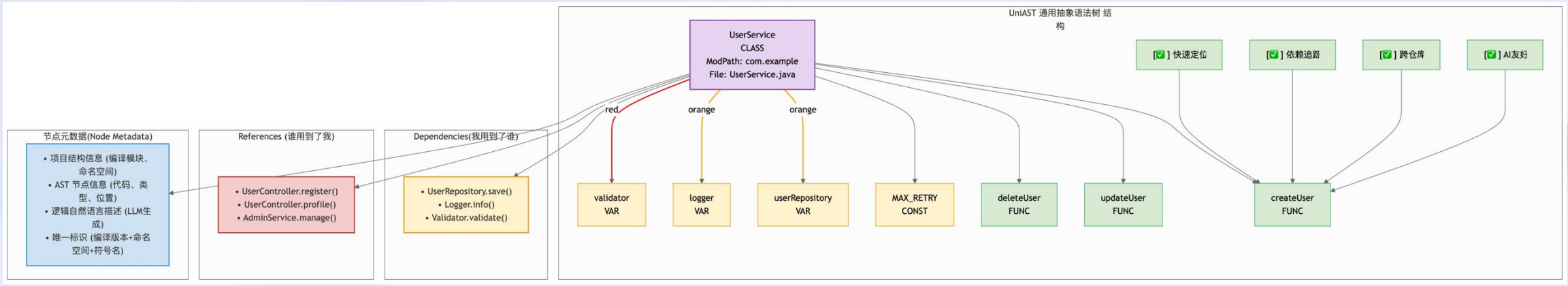
调用链路解析

自动识别函数调用路径，还原分布式场景下的复杂依赖结构。

语义关系网络

整合类型继承、接口实现等语义信息，构建完整的代码知识图谱。

UniAST: 结合前两步构建出的最终AI友好地图



统一语法树

融合Tree-sitter与LSP信息，构建跨文件、语义完整的统一抽象语法树。

AI可读结构

将原始代码转化为AI易于理解的标准化、结构化中间表示形式。

关系全覆盖

整合符号定义、引用与调用关系，形成全局代码知识网络。

精准导航图

为AI提供包含上下文与依赖的高精度代码导航与修改指引。

03

ABCoder加持的AI 实践

实际案例-在复杂系统中 PRD 到技术方案 和 代码

AI coding 的美好愿景与现实挑战

理想的 vibe AI 编程

智能协作者

AI深度理解业务与代码，像资深工程师一样思考，主动提出合理方案。

零误操作

所有修改精准可控，绝不引入意外变更，确保系统稳定性。

无缝协作流

与开发者自然配合，按需提供信息、生成代码、解释逻辑，提升效率。

现实挑战

架构复杂性高

分布式系统涉及多服务、多节点交互，AI难以全面理解整体架构与依赖关系。

分布式困境

数据与状态跨服务分布，AI易遗漏关键上下文，导致生成代码逻辑不一致。

影响难预估

局部修改可能引发全局副作用，AI缺乏对系统级影响的准确判断能力。

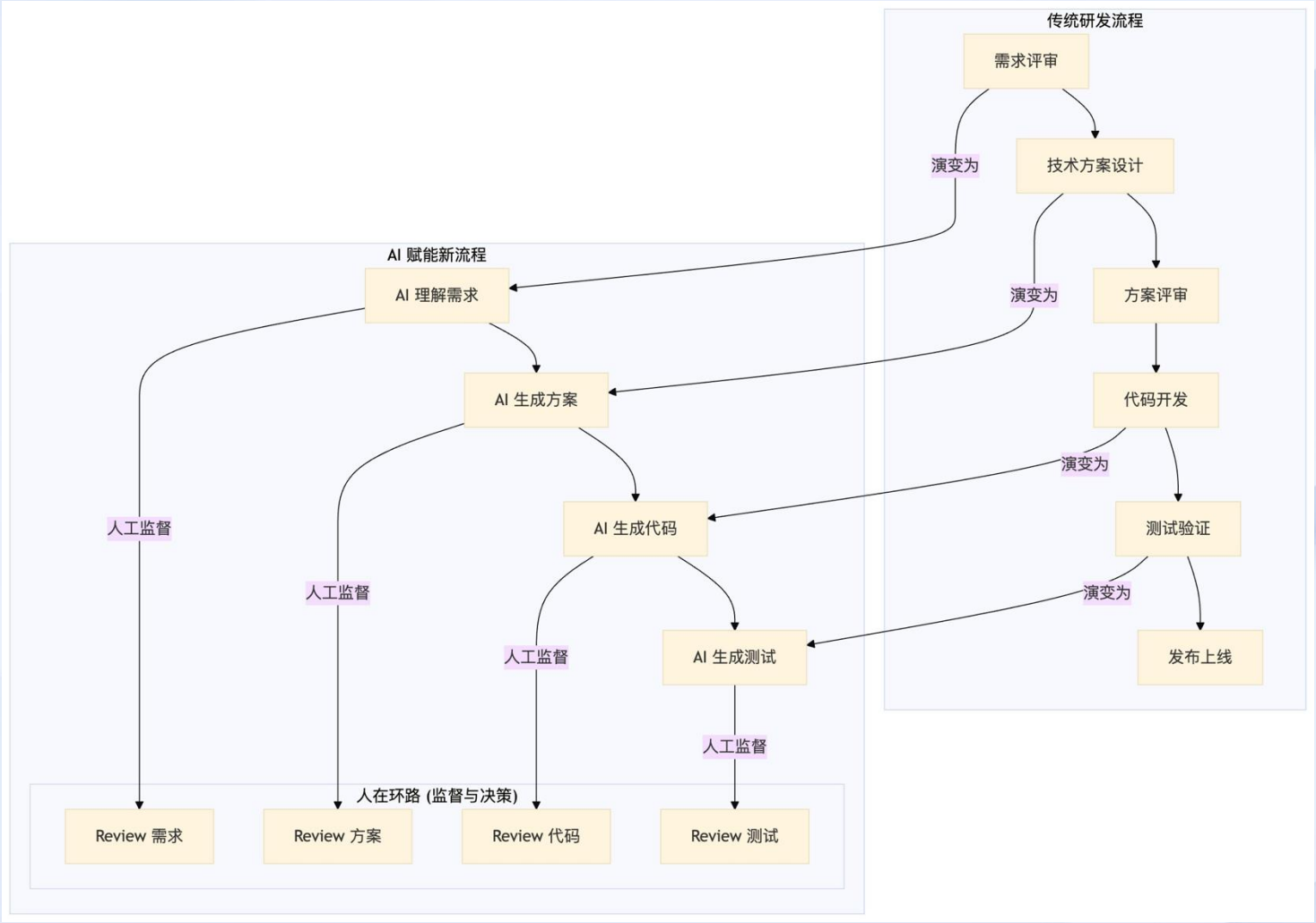
解决方案: 拆分步骤+人在回路，确保每一步都由人来监督和确认

核心目标：高效产出可控、可理解的代码

分步解耦任务
将开发流程拆解为可控阶段，
降低AI出错概率。

人在回路监督
每步输出均需人工审核确认
确保方向正确。

闭环反馈迭代
结合反馈动态调整AI行为
提升结果可靠性。



分析现有仓库-改动点

分析过程

接收PRD输入

系统接入产品需求文档

自动提取核心功能描述与变更意图。

语义关键词提取

基于LLM识别业务关键词，

如订单、支付、用户权限等关键实体。

映射代码模块

结合项目领域模型，将业务术语匹配到

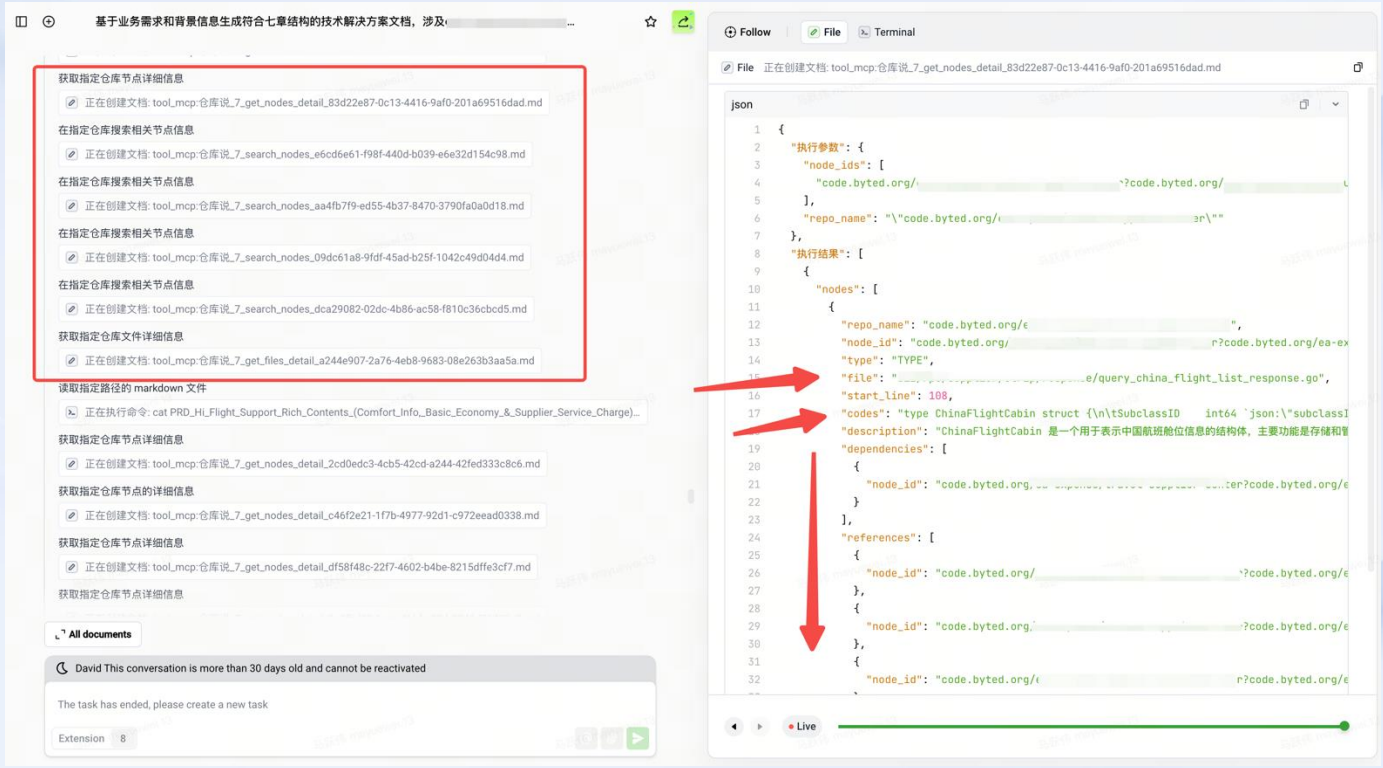
对应的微服务与包结构。

定位逻辑边界

初步划定受影响的业务层、

控制器及数据访问组件范围。

分析结果



文件路径	变更类型	类/结构体	方法/字段	变更原因及核心考量
`biz/converter/flight/flight_converter.go`	**MODIFY**	`FlightConverter`	`ToDomain...`	**MODIFY (核心)**: 1. **解析舒适度**: 将供应商返回的复杂结构转换为内部 `ComfortInfo` 模型。 2. **识别基础经济舱**: 遍历 `BrandAttributeList`，根据 `refNum` 常量值设置 `IsBasicEconomy` 标志。 3. **提取服务费**: 将 `consolidatorFee` 映射到 `Policy` 模型。**可维护性**: 所有转换逻辑集中于此，便于未来维护和扩展。
`biz/domain/flight/flight.go`	MODIFY	`Flight`	`+ ComfortInfo ComfortInfo`	**ADD**: 在航班主实体中增加结构体，用于聚合所有舒适度信息，使领域模型更内聚。

结合 改动点和技术方案模板 LLM 生成技术方案

构建输入模板

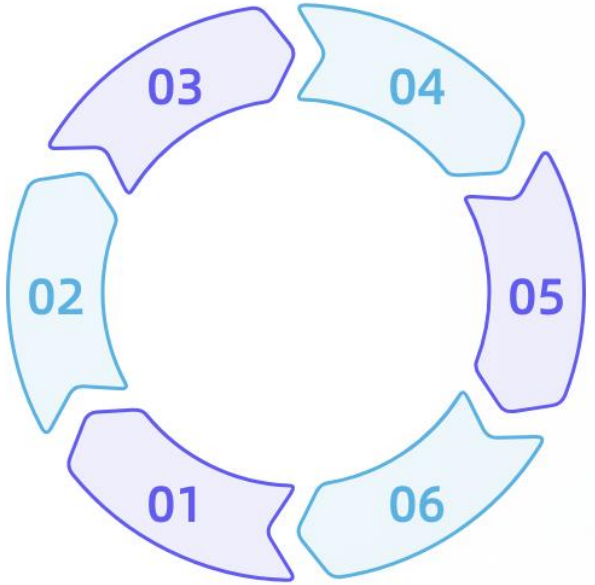
设计结构化输入格式，规范问题描述与请求参数。提升LLM理解准确性，降低歧义风险。

明确改动点

识别本次迭代的具体变更范围，界定新增或修改功能。避免范围蔓延，确保开发与设计对齐。

整合PRD摘要

提取产品需求文档核心内容，明确背景与目标。聚焦关键功能点，减少冗余信息干扰。为后续技术方案生成提供清晰上下文。



引导LLM生成

在限定上下文中引导大模型输出，确保逻辑连贯性。聚焦技术实现细节，避免泛化回答。

统一输出格式

设定标准化响应结构，保证结果可读与可比性。便于团队评审、归档与后续自动化处理。

提升评审效率

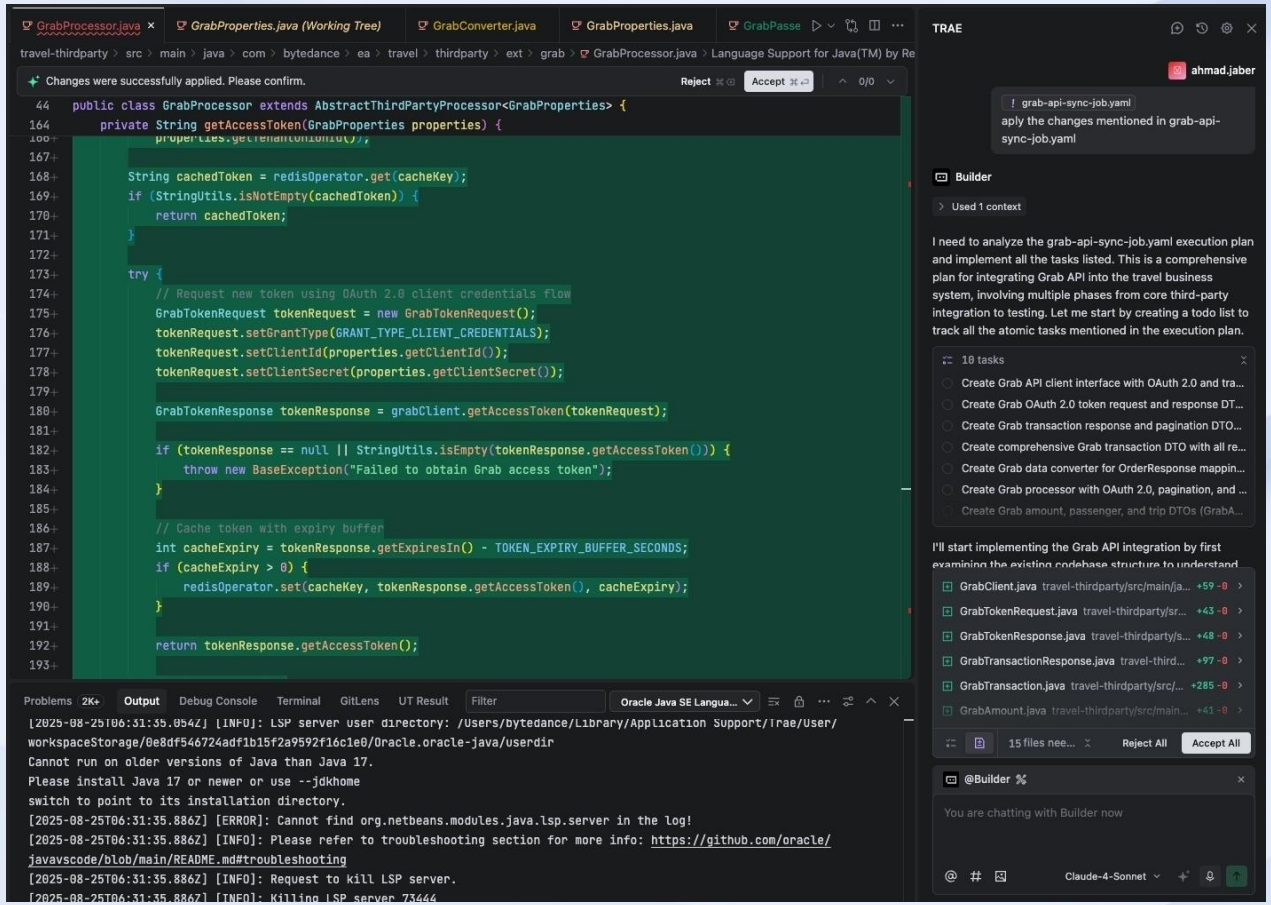
通过一致性与精准性优化，缩短反馈周期。降低沟通成本，加快迭代进度。

Code 生成-基于ABCoder 的执行计划,Trae IDE Agent 执行

底层逻辑同技术方案 一致，都依赖于 ABCoder 提供的 仓库 context

代码块 YAML 取消自动换行 复制

```yaml
- task: "为Flight数据模型添加comfortInfo字段"
  file: "models/flight.go"
  action: "modify"

- task: "创建ComfortInfo数据源的客户端"
  file: "clients/comfort_client.go"
  action: "create"

- task: "在FlightService中增加调用comfort_client的逻辑"
  file: "services/flight_service.go"
  action: "modify"

- task: "更新GetFlightDetails API的返回值"
  file: "controllers/flight_controller.go"
  action: "modify"

- task: "为新增逻辑编写单元测试"
  file: "services/flight_service_test.go"
  action: "create_or_modify"
```



Q&A

THANKS