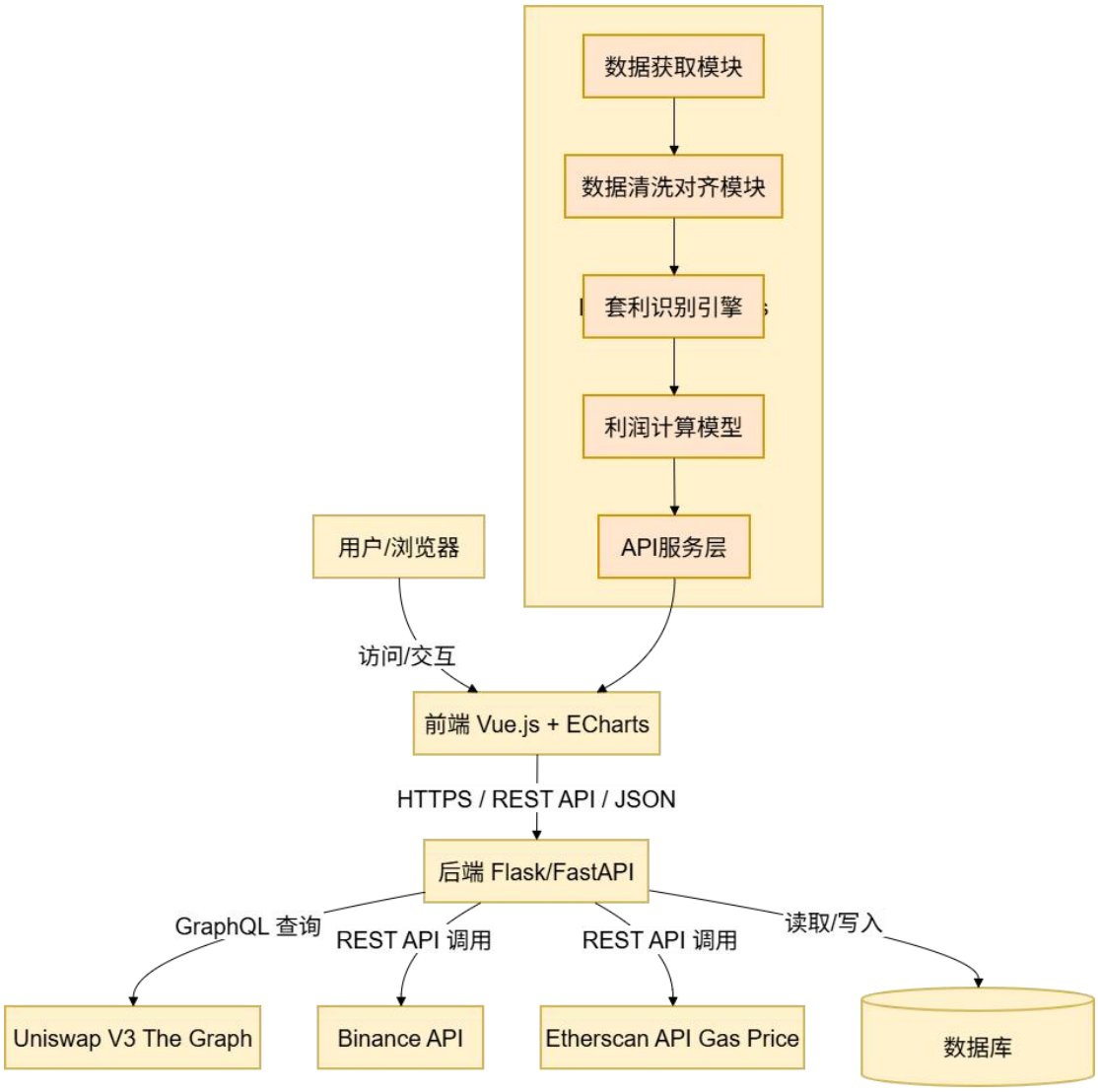
软件设计文档

1 总体设计

1.1 系统架构

本系统采用 B/S 架构，前后端分离开发。系统旨在整合与处理来自异构数据源（DEX 与 CEX）的价格数据，通过核心算法识别非原子套利机会，并通过 Web 界面将结果直观地呈现给用户。以下是系统架构图：



1.2 模块划分

前端模块

视图层:

Overview.vue: 系统总览，展示关键指标。

MarketCompare.vue: 市场比价，展示 CEX 与 DEX 的价格走势及价差。

ArbRadar.vue: 套利雷达，实时监控套利机会。

Backtest.vue: 回测分析，展示历史策略表现。

工具层:

CsvLoader.js: 负责异步加载和解析 CSV 数据文件。

WorkerManager.js: 管理 Web Workers，将繁重的数据计算任务移出主线程。

组件层:

ChartCard.vue: 封装 ECharts 图表，提供统一的图表展示容器。

后端模块

接口服务: api.py，定义对外接口（目前主要用于调试和未来扩展）。

核心业务: service.py，包含套利识别算法、价差计算逻辑。

配置管理: config.py，管理数据库连接、算法参数等配置。

1.3 技术路线

前端: Vue 2.6.14 + Axios + ECharts 5.4.3，结合 Web Worker 处理体量较大的 CSV/JSON 数据。

后端: Python 3.x + Flask + Pandas/NumPy。通过 service.py 和 api.py 进行核心算法实现与数据处理。

数据存储: CSV 文件 + JSON 文件 + MySQL 数据库，用于存储清洗后的数据与套利信号。

数据获取: 脚本获取（get_data.py 从 Etherscan/Binance API）与本地数据处理（processedDataLoader.js, csvLoader.js）。

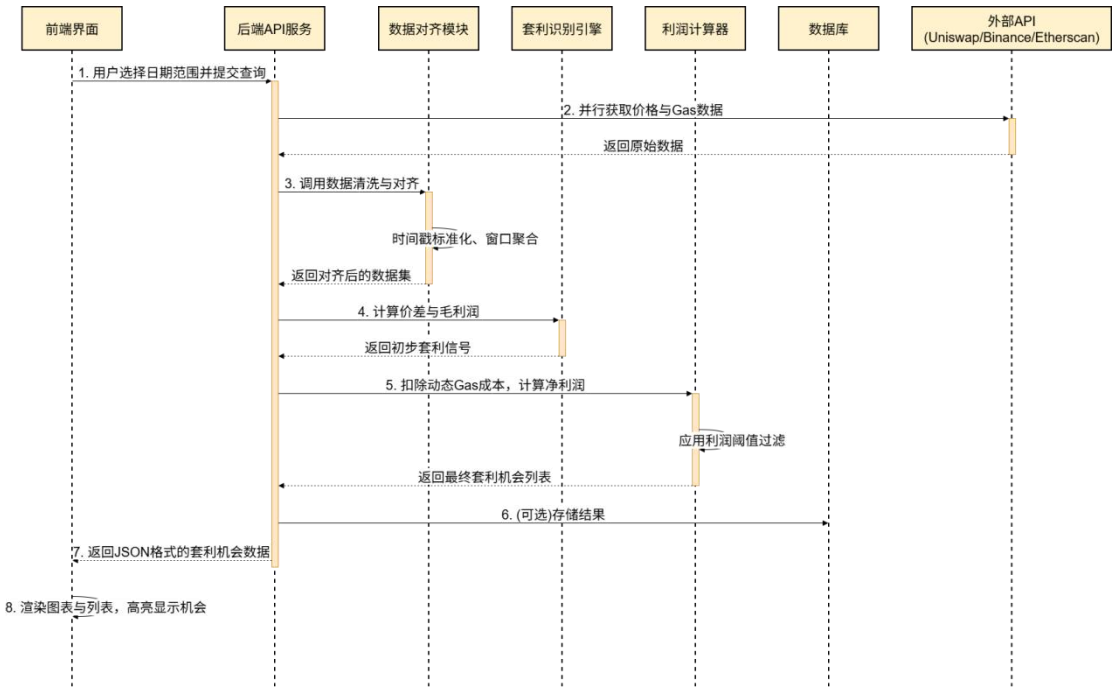
开发与部署: Git（版本控制）。

接口定义: 基于文件结构与数据字段定义进行前后端协作（Flask REST API + 本地文件加载）。

2 详细设计

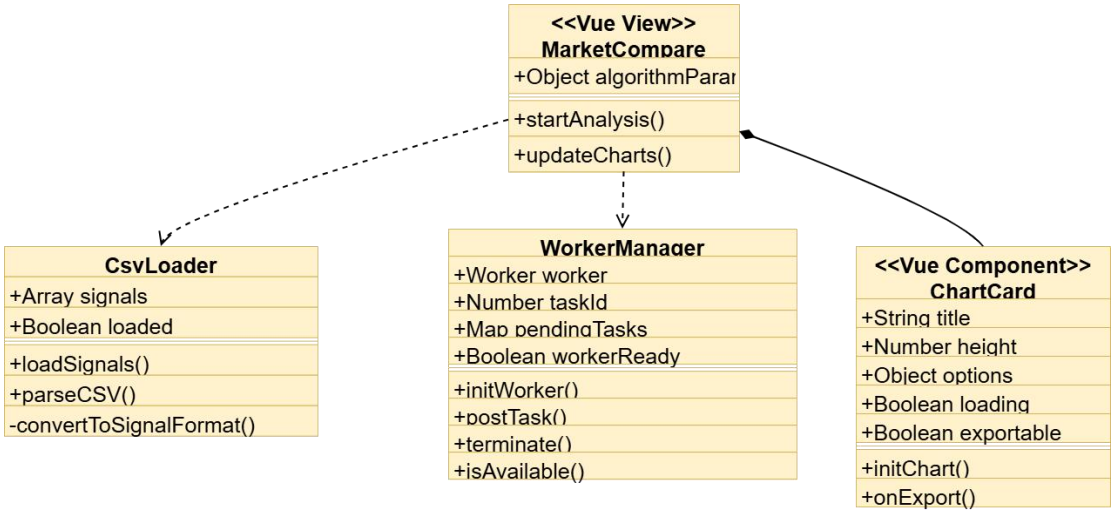
2.1 核心流程

非原子套利机会识别与展示时序图



2.2 类图

已实现前端核心类图展示：



2.3 接口说明

虽然目前主要通过文件交互，但定义了明确的数据结构规范：

字段名	类型	说明
`id`	String	信号唯一标识
`time`	Number	时间戳
`direction`	String	交易方向 (e.g., "CEX→DEX")
`spread`	Number	价差值
`spreadPct`	Number	价差百分比
`zScore`	Number	Z-Score 值（用于统计套利）
`netProfit`	Number	预期净利润
`confidence`	Number	置信度

3 关键决策

(1) 明确采用前后端分离的开发模式，并通过设计良好的 RESTful API 进行通信。理由:允许前端团队专注于用户体验和交互逻辑，后端团队专注于数据处理和核心算法，并行开发提高效率，使得前端技术栈（Vue）和后端技术栈（Python）可以独立选择和升级。

(2) 不直接使用原始时间戳进行计算，而是开发一个中间件来专门处理 Uniswap 和 Binance 数据的时间对齐问题。理由:直接计算会导致“虚假价差”，严重影响算法准确性。这样做可以确保数据质量，通过向下取整和聚合，为后续的价差计算提供了干净、可比对的数据基础，是算法可靠性的前提。

(3) 在 MVP 阶段后期，放弃使用固定 Gas 价格，改为通过 Etherscan API 查询历史交易发生时的真实 Gas 价格。理由:提升利润计算真实性: 固定 Gas 费无法反映链上拥堵情况，会导致利润估算严重偏离实际（如高估利润）。动态模型是使“非原子套利”分析具有实际参考价值的关键。团队采取了分步集成策略，先实现固定成本模型确保流程跑通，再升级为动态模型，降低了开发复杂度。

(4) 采用 MySQL 存储清洗后的历史数据和计算结果。理由:价格数据、套利机会都是结构化的记录，适合用关系型数据库存储和查询。对于按时间范围查询、批量插入等操作，关系型数据库性能优异，能满足前端图表和列表的数据请求。