内部股票行情数据分析系统

设计说明书

**2025年07月19日**

**资料版本：** V0.5

**日 期： 2025**年07月

**密 级：** 🞎公开资料 🗹内部资料 🞎保密资料 🞎机密资料

**状 态：** 🗹初稿 🞎讨论稿 🞎发布

**修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 作者 | 版本 | 修改记录 |
| 2025-07-19 | 王学武 | 0.5 | 创建 |
| 2025-08-11 | 王学武 | 0.6 | 完成系统架构、数据采集系统等内容。 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**审阅**

|  |  |  |
| --- | --- | --- |
| 姓名 | 时间 | 职位 |
|  |  |  |
|  |  |  |
|  |  |  |

**目录**

[1 概述 7](#_Toc205970039)

[*1.1.* 目的 7](#_Toc205970040)

[*1.2.* 核心系统架构范围 7](#_Toc205970041)

[*1.3.* 系统组件图 8](#_Toc205970042)

[*1.4.* 数据流管道图 10](#_Toc205970043)

[*1.5.* 关键组件 11](#_Toc205970044)

[1.5.1. 数据采集基础设施 11](#_Toc205970045)

[1.5.2. 后端服务 11](#_Toc205970046)

[1.5.3. 前端应用程序 11](#_Toc205970047)

[*1.6.* 数据库架构概述 11](#_Toc205970048)

[*1.7.* 开发路线图 12](#_Toc205970049)

[*1.8.* 系统入口点 12](#_Toc205970050)

[2 系统架构 12](#_Toc205970051)

[*2.1.* 目的和范围 12](#_Toc205970052)

[*2.2.* 顶层架构概述 13](#_Toc205970053)

[3 数据采集系统 15](#_Toc205970054)

[*3.1.* 采集调度 15](#_Toc205970055)

[3.1.1. 目的和范围 15](#_Toc205970056)

[3.1.2. 调度器架构 15](#_Toc205970057)

[3.1.3. 系统启动流程 17](#_Toc205970058)

[3.1.4. 采集任务计划 18](#_Toc205970059)

[3.1.5. 数据采集执行流程 19](#_Toc205970060)

[3.1.6. 数据流图 20](#_Toc205970061)

[*3.2.* Akshare数据采集器 20](#_Toc205970062)

[3.2.1. 架构概述 20](#_Toc205970063)

[*3.2.1.1.* 类层次结构 20](#_Toc205970064)

[*3.2.1.2.* 数据流架构 21](#_Toc205970065)

[3.2.2. 采集器基类 22](#_Toc205970066)

[3.2.3. 实时数据采集器 23](#_Toc205970067)

[*3.2.3.1.* 实时行情报价采集器 23](#_Toc205970068)

[*3.2.3.2.* 行业板块行情采集器 24](#_Toc205970069)

[*3.2.3.3.* 指数行情采集器 24](#_Toc205970070)

[*3.2.3.4.* 新闻公告研报采集器 25](#_Toc205970071)

[3.2.4. 历史行情数据采集器 26](#_Toc205970072)

[3.2.5. 错误处理策略 28](#_Toc205970073)

[3.2.6. 数据存储模式 28](#_Toc205970074)

[*3.3.* Tushare数据采集器 29](#_Toc205970075)

[3.3.1. 架构概述 29](#_Toc205970076)

[*3.3.1.1.* 类层次结构 29](#_Toc205970077)

[*3.3.1.2.* 数据库集成 30](#_Toc205970078)

[3.3.2. 历史行情数据采集器 30](#_Toc205970079)

[*3.3.2.1.* 主要特性 30](#_Toc205970080)

[*3.3.2.2.* 数据采集流程 31](#_Toc205970081)

[*3.3.2.3.* 数据库模式操作 31](#_Toc205970082)

[3.3.3. 实时行情数据采集器 32](#_Toc205970083)

[*3.3.3.1.* 采集流程 32](#_Toc205970084)

[3.3.4. 指数行情数据采集器 33](#_Toc205970085)

[*3.3.4.1.* 实现 33](#_Toc205970086)

[3.3.5. 配置和操作 33](#_Toc205970087)

[*3.3.5.1.* CLI界面 33](#_Toc205970088)

[*3.3.5.2.* 错误处理策略 34](#_Toc205970089)

[3.3.6. 数据质量和验证 35](#_Toc205970090)

[*3.3.6.1.* 价值处理管道 35](#_Toc205970091)

[*3.4.* 关注股票列表数据采集 35](#_Toc205970092)

[3.4.1. 目的和范围 35](#_Toc205970093)

[3.4.2. 概述 35](#_Toc205970094)

[3.4.3. 采集工作流程 36](#_Toc205970095)

[3.4.4. 主要功能 37](#_Toc205970096)

[3.4.5. 数据处理管道 37](#_Toc205970097)

[*3.4.5.1.* 历史数据插入流程 37](#_Toc205970098)

[3.4.6. 数据库集成 37](#_Toc205970099)

[3.4.7. 调度集成 39](#_Toc205970100)

[4 后端API服务 39](#_Toc205970101)

[*4.1.* 系统概述 39](#_Toc205970102)

[4.1.1. FastAPI应用程序架构 39](#_Toc205970103)

[4.1.2. 身份验证和安全框架 40](#_Toc205970104)

[4.1.3. 请求处理管道 41](#_Toc205970105)

[4.1.4. API端点类别 43](#_Toc205970106)

[*4.1.4.1.* 股票管理API（stock\_router） 43](#_Toc205970107)

[*4.1.4.2.* 历史数据API（history\_router） 43](#_Toc205970108)

[*4.1.4.3.* 股票分析API（stock\_analysis\_router） 44](#_Toc205970109)

[*4.1.4.4.* 系统管理API（多个router） 44](#_Toc205970110)

[*4.1.4.5.* 市场数据API（market\_router） 45](#_Toc205970111)

[4.1.5. 数据库集成 46](#_Toc205970112)

[4.1.6. 错误处理和响应格式 47](#_Toc205970113)

[*4.2.* FastAPI 应用程序核心 48](#_Toc205970114)

[4.2.1. FastAPI应用程序配置 48](#_Toc205970115)

[4.2.2. 中间件堆栈配置 49](#_Toc205970116)

[*4.2.2.1.* 请求日志中间件 49](#_Toc205970117)

[*4.2.2.2.* CORS配置 49](#_Toc205970118)

[4.2.3. 路由器集成架构 49](#_Toc205970119)

[*4.3.* 股票管理API 51](#_Toc205970120)

[4.3.1. 目的和范围 51](#_Toc205970121)

[4.3.2. 历史数据API 51](#_Toc205970122)

[4.3.3. 实时报价管理 51](#_Toc205970123)

[4.3.4. 批量报价检索 51](#_Toc205970124)

[4.3.5. 个股实时数据 51](#_Toc205970125)

[4.3.6. 财务数据集成 51](#_Toc205970126)

[4.3.7. 股票排名和筛选 51](#_Toc205970127)

[*4.3.7.1.* 行情板块实现 51](#_Toc205970128)

[4.3.8. 前端集成 52](#_Toc205970129)

[*4.4.* 历史数据API 52](#_Toc205970130)

[4.4.1. API端点概述 52](#_Toc205970131)

[4.4.2. 数据检索API 52](#_Toc205970132)

[*4.4.2.1.* 端点规范 52](#_Toc205970133)

[*4.4.2.2.* 查询实现 52](#_Toc205970134)

[*4.4.2.3.* 响应结构 52](#_Toc205970135)

[4.4.3. 导出API 52](#_Toc205970136)

[*4.4.3.1.* Excel生成管道 52](#_Toc205970137)

[*4.4.3.2.* Excel样式功能 53](#_Toc205970138)

[4.4.4. 数据过滤 53](#_Toc205970139)

[4.4.5. 分页 53](#_Toc205970140)

[4.4.6. 前端集成 53](#_Toc205970141)

[*4.4.6.1.* 前端数据处理 53](#_Toc205970142)

[4.4.7. 数据流架构 53](#_Toc205970143)

[*4.5.* 市场数据API 53](#_Toc205970144)

[4.5.1. 概述 53](#_Toc205970145)

[4.5.2. 市场指数端点 53](#_Toc205970146)

[4.5.3. 行业板块端点 54](#_Toc205970147)

[4.5.4. 数据库集成 54](#_Toc205970148)

[4.5.5. 数据转换管道 54](#_Toc205970149)

[4.5.6. 与数据采集系统集成 54](#_Toc205970150)

[*4.6.* 新闻与研究API 54](#_Toc205970151)

[4.6.1. 目的和范围 54](#_Toc205970152)

[4.6.2. API端点概述 54](#_Toc205970153)

[4.6.3. 数据源和集成 54](#_Toc205970154)

[4.6.4. 数据处理管道 55](#_Toc205970155)

[4.6.5. PDF处理 55](#_Toc205970156)

[4.6.6. 数据库交互 55](#_Toc205970157)

[4.6.7. 错误处理和监控 55](#_Toc205970158)

[*4.7.* 身份验证和管理API 55](#_Toc205970159)

[4.7.1. 身份验证架构 55](#_Toc205970160)

[4.7.2. 管理仪表板API 55](#_Toc205970161)

[4.7.3. 用户管理API 55](#_Toc205970162)

[4.7.4. 报价数据管理API 56](#_Toc205970163)

[4.7.5. 安全与授权 56](#_Toc205970164)

[5 前端应用程序 56](#_Toc205970165)

[*5.1.* 概述 56](#_Toc205970166)

[5.1.1. 系统架构概述 56](#_Toc205970167)

[5.1.2. 应用程序通信流程 56](#_Toc205970168)

[5.1.3. 前端应用程序架构 56](#_Toc205970169)

[*5.2.* 股票详情界面 56](#_Toc205970170)

[*5.3.* 市场概览 56](#_Toc205970171)

[*5.4.* 关注列表管理 57](#_Toc205970172)

[*5.5.* 历史数据接口 57](#_Toc205970173)

[*5.6.* 常见前端实用程序 57](#_Toc205970174)

[6 数据库和存储 57](#_Toc205970175)

[*6.1.* 数据库架构 57](#_Toc205970176)

[6.1.1. 数据库技术栈 57](#_Toc205970177)

[6.1.2. Sqlalchemy模型架构 57](#_Toc205970178)

[*6.2.* 核心数据模型 57](#_Toc205970179)

[*6.3.* 表关系和约束 57](#_Toc205970180)

[*6.4.* 常见存储模式 57](#_Toc205970181)

[7 配置和部署 57](#_Toc205970182)

[*7.1.* 系统配置管理 58](#_Toc205970183)

[7.1.1. 核心配置结构 58](#_Toc205970184)

[7.1.2. 数据库配置 58](#_Toc205970185)

[7.1.3. 身份验证配置 58](#_Toc205970186)

[7.1.4. 数据采集配置 58](#_Toc205970187)

[*7.2.* 开发环境设置 58](#_Toc205970188)

[7.2.1. 依赖管理和启动过程 58](#_Toc205970189)

[7.2.2. 开发服务器配置 58](#_Toc205970190)

[7.2.3. 日志配置 58](#_Toc205970191)

[*7.3.* 部署架构 58](#_Toc205970192)

[7.3.1. 生产环境结构 58](#_Toc205970193)

[7.3.2. 服务启动和流程管理 58](#_Toc205970194)

[7.3.3. 环境变量管理 59](#_Toc205970195)

[7.3.4. 安全配置 59](#_Toc205970196)

# 概述

本文档对中国股市分析系统进行了高层次的介绍，涵盖其架构、核心组件和开发路线图。该系统旨在全面收集、分析和可视化中国股市数据，并具备实时监控功能。

有关特定子系统的详细信息，请参阅[系统架构](https://deepwiki.com/wangj9601/stock_quote_analayze/2-system-architecture)了解整体设计模式、[数据收集系统](https://deepwiki.com/wangj9601/stock_quote_analayze/3-data-collection-system)了解自动数据收集、[后端 API 服务](https://deepwiki.com/wangj9601/stock_quote_analayze/4-backend-api-services)了解 REST 服务实现、[前端应用程序](https://deepwiki.com/wangj9601/stock_quote_analayze/5-frontend-applications)了解用户界面以及[数据库和存储](https://deepwiki.com/wangj9601/stock_quote_analayze/6-database-and-storage)了解数据持久性。

## 目的和范围

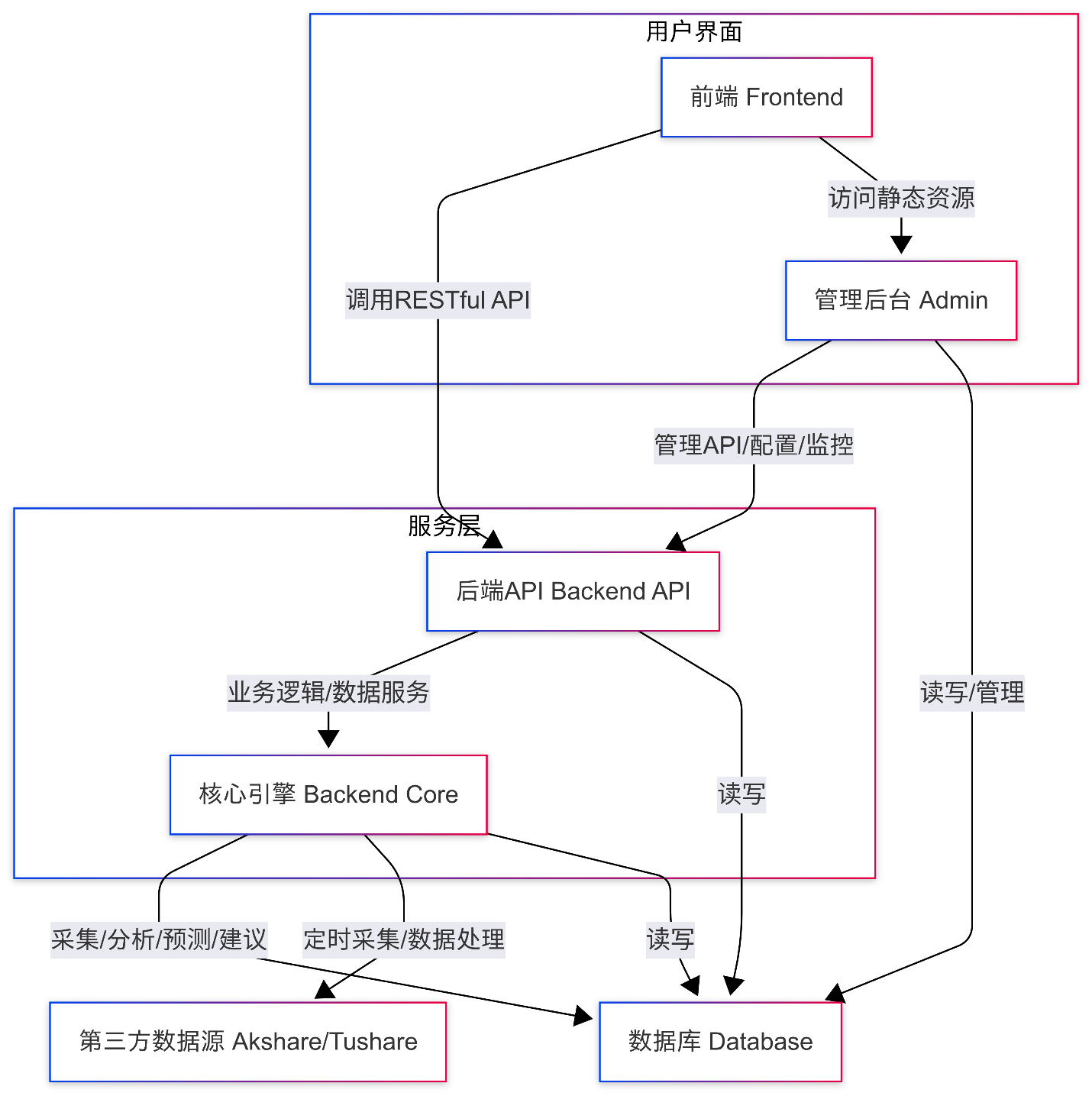
股票分析系统是针对中国股市数据分析的综合平台，提供：

* **自动数据收集：**通过 Akshare 和 Tushare API 定期收集实时报价、历史数据、新闻和财务指标
* **REST API 服务：**基于 FastAPI 的后端提供数据访问、用户管理和业务逻辑
* **Web 界面：**用于股票分析、投资组合跟踪和市场监控的交互式前端
* **数据存储：**具有结构化模式的 PostgreSQL 数据库，可实现高效的数据检索和分析

## 核心系统架构范围

本系统遵循分层架构模式，数据收集、存储、API 服务和表示层之间有明确的分离。。

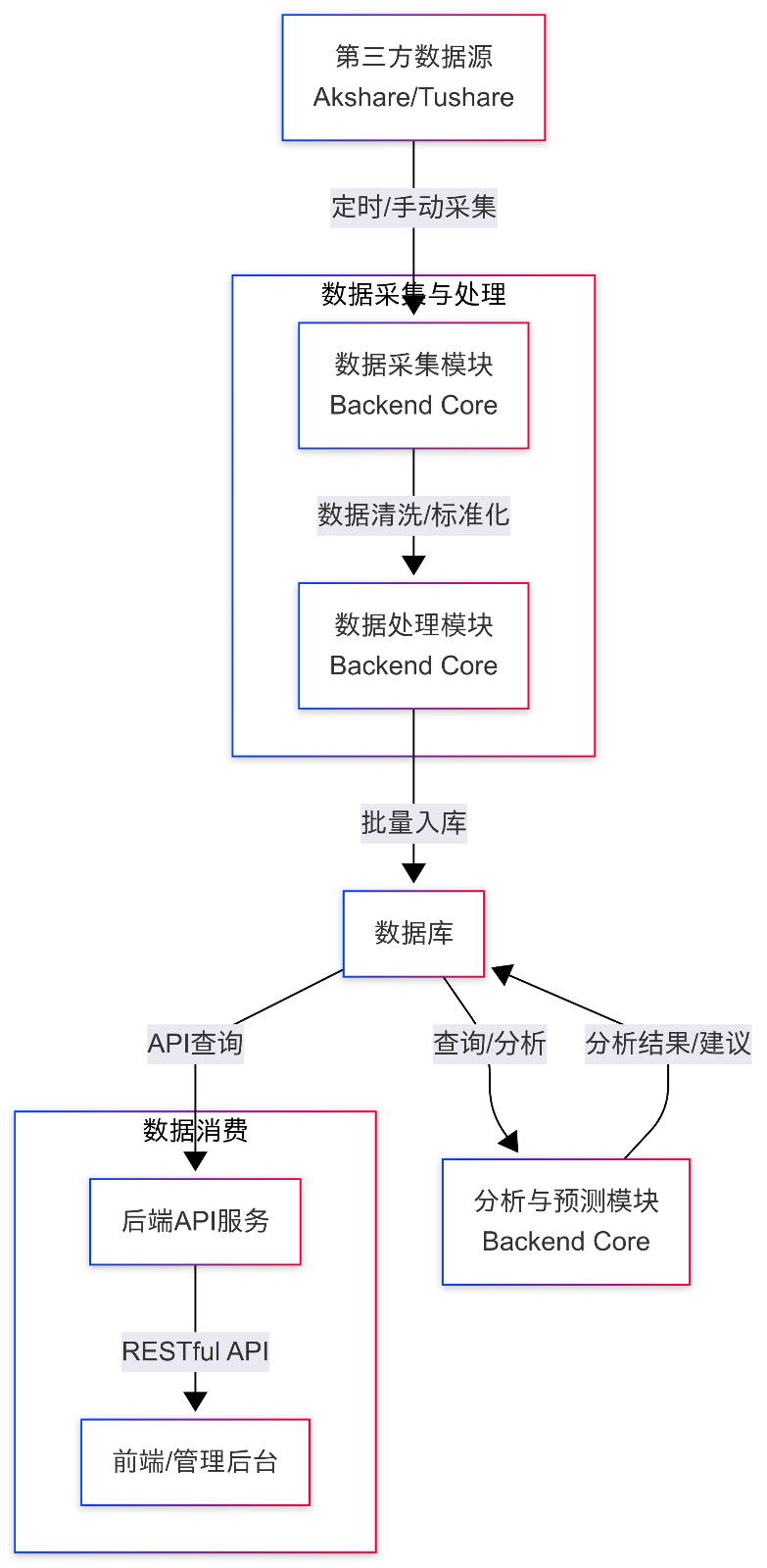
## 系统组件图

****

股票分析系统的主要组件及其关系如下：

* 前端（Frontend）：为普通用户提供行情、分析、选股、资讯、自选股等页面，通过RESTful API与后端交互。
* 管理后台（Admin）：为管理员提供用户、数据、采集、系统配置、日志、模型等管理功能，也通过 API与后端交互。
* 后端API（Backend API）：基于 FastAPI，负责用户认证、数据服务、业务逻辑处理、系统管理等，向前端和管理后台提供统一接口。
* 核心引擎（Backend Core）：负责数据采集（akshare/tushare）、数据清洗、分析计算、模型预测、交易建议等核心功能，定时采集并处理第三方数据。
* 数据库（Database）：存储行情、用户、日志、配置、分析结果等所有结构化数据。
* 第三方数据源（Akshare/Tushare）：为系统提供实时及历史行情、财务、新闻等原始数据。

## 数据流管道图

****

**说明：**

* 数据从Akshare/Tushare 等第三方数据源采集，经采集模块进入数据处理模块进行清洗和标准化。
* 处理后的数据批量写入数据库。
* 分析与预测模块对数据库中的数据进行分析、建模和生成建议，结果也存回数据库。
* 后端API服务从数据库读取数据，向前端和管理后台提供RESTful API接口。
* 前端和管理后台通过API获取所需数据进行展示和管理。

## 关键组件

### 数据采集基础设施

|  |  |  |
| --- | --- | --- |
| **组件** | **用途** | **关键类** |
| **主程序** | 集中数据采集协调 | backend\_core.data\_collectors.main |
| **Akshare 采集器** | 实时报价、行业板块、指数行情、新闻公告等 | AkshareRealtimeQuoteCollector, RealtimeIndexSpotAkCollector |
| **Tushare 采集器** | 历史数据、财务指标等 | HistoricalQuoteCollector, RealtimeQuoteCollector |
| **调度器** | 自动采集调度 | APScheduler.BlockingScheduler |

### 后端服务

|  |  |  |
| --- | --- | --- |
| **服务层** | **端点** | **主要功能** |
| **股票 API** | /api/stock/\* | 实时行情、K线数据、财务指标等 |
| **市场 API** | /api/market/\* | 市场指数、行业表现等 |
| **授权认证 API** | /api/autht/\* | 用户认证、JWT token管理等 |
| **管理 API** | /api/admin/\* | 系统管理、用户管理等 |

### 前端应用程序

|  |  |  |
| --- | --- | --- |
| **界面** | **文件** | **主要特性** |
| **主页** | index.html | 市场概览、指数、关注列表预览、新闻等 |
| **股票分析** | stock.html | 交互式图表、财务数据、新闻、公告等 |
| **关注列表管理** | watchlist.html | 投资组合跟踪、股票分组等 |
| **市场行情探索** | markets.html | 行业分析、股票排名等 |

## 数据库架构概述

系统使用 PostgreSQL 数据库名称stock\_analysis，并包含以下核心表：

* **股票数据：**stock\_realtime\_quote、historical\_quotes、stock\_basic\_info…
* **用户管理：**users、watchlist…
* **市场信息：**industry\_board\_realtime\_quotes，stock\_news…
* **系统操作：**operation\_logs…

## 开发路线图

**近期开发**

* 个股数据采集与分析
* 交易详情，包括时间、价格、交易量和金额数据
* 开发环境搭建、应用和数据库服务器部署

**中期目标**

* 高级分析模型与算法
* 股票选择和筛选能力

**远期愿景**

* 交易策略管理系统
* 策略回测框架

## 系统入口点

主系统可以通过几个入口点启动：

* **数据采集主程序：**python -m backend\_core.data\_collectors.main
* **API：**端口 5000 上的 FastAPI 应用程序
* **数据库访问：**使用管理员凭据访问 PostgreSQL 服务器 5446 端口
* **前端：**8000端口访问股票数据分析前端
* **管理端：**8001端口访问后台管理端

# 系统架构

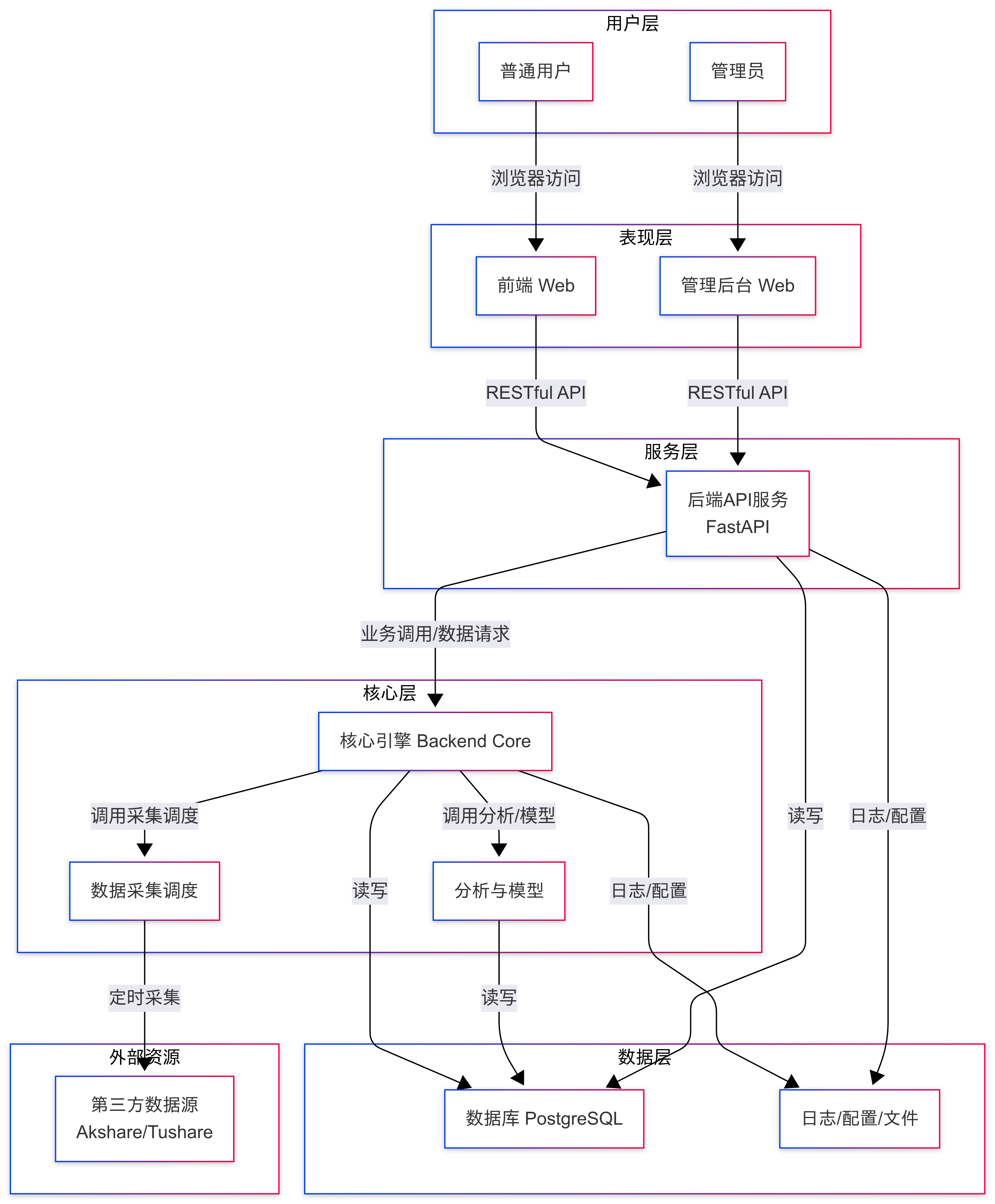
## 目的和范围

本章节全面概述股票分析系统的架构，涵盖整体系统设计、数据流模式以及整个平台的组件关系。文档描述主要子系统的交互方式，以及实现自动数据收集、实时市场分析和用户投资组合管理的关键架构决策。

有关特定子系统的详细信息，请参阅[**数据采集系统**](https://deepwiki.com/wangj9601/stock_quote_analayze/3-data-collection-system)**、**[**后端 API 服务**](https://deepwiki.com/wangj9601/stock_quote_analayze/4-backend-api-services)**、**[**前端应用程序**](https://deepwiki.com/wangj9601/stock_quote_analayze/5-frontend-applications)和[**数据库与存储**](https://deepwiki.com/wangj9601/stock_quote_analayze/6-database-and-storage)。

## 顶层架构概述

本系统采用分层架构模式，数据收集、API 服务和用户界面之间清晰划分。该架构支持实时数据处理、定时批量操作和多用户 Web 访问。



1. **用户层**

* **普通用户**：通过浏览器访问前端页面，获取行情、分析、资讯、自选股等服务。
* **管理员**：通过浏览器访问管理后台，进行用户、数据、采集、系统配置、日志、模型等管理操作。

1. **表现层**

* **前端 Web：**为普通用户提供股票行情、分析、图表、资讯等可视化界面，所有数据均通过后端API获取。
* **管理后台 Web：**为管理员提供系统管理界面，包括用户管理、数据源配置、采集任务、日志监控、模型管理等。

1. **服务层**

* **后端API服务（FastAPI）：**统一对外提供RESTful API，负责用户认证、数据服务、业务逻辑处理、系统管理等。前端和管理后台均通过API与后端交互。

1. **核心层**

* **核心引擎（Backend Core）：**负责数据采集、清洗、标准化、分析、预测、交易建议等核心业务逻辑。
* **分析与模型：**实现技术分析、基本面分析、机器学习/深度学习模型预测等功能，为用户和管理端提供智能分析结果。
* **数据采集调度：**定时或手动从第三方数据源采集行情、财务、新闻等数据，保证数据的实时性和完整性。

1. **数据层**

* **数据库（PostgreSQL）：**存储所有结构化数据，包括行情、用户、日志、配置、分析结果等，支持高效查询和批量写入。
* **日志/配置/文件：**存储系统运行日志、采集日志、配置文件、导出文件等，便于系统监控和维护。

1. **外部资源**

* **第三方数据源（Akshare/Tushare）：**为系统提供A股、ETF、指数、财务、新闻等原始数据，是数据采集的基础。

**主要交互说明**

* 用户和管理员均通过浏览器访问各自界面，所有数据和操作均通过后端API完成。
* 后端API服务作为唯一的数据和业务逻辑入口，既服务前端，也服务管理后台。
* 核心引擎负责所有数据的采集、处理、分析和预测，并与数据库进行读写交互。
* 数据采集调度模块定时从第三方数据源拉取最新数据，经过清洗和标准化后入库。
* 日志和配置文件为系统运维和监控提供支持，所有关键操作均有日志记录。

# 数据采集系统

## 采集调度

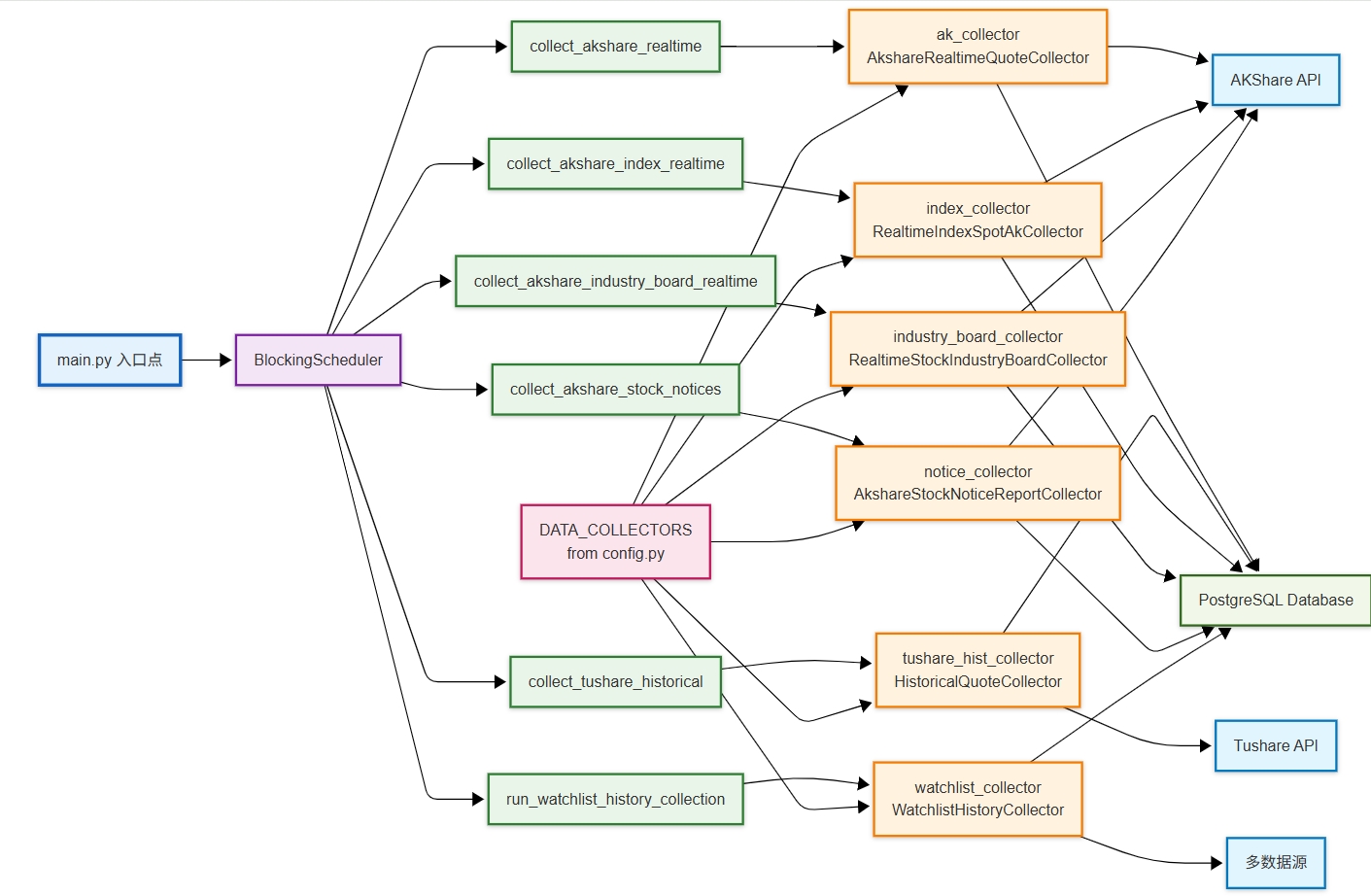
### 目的和范围

数据采集调度模块为股票分析平台中的所有自动化数据收集任务提供集中调度和协调。该模块使用 APSchedulerBlockingScheduler来管理来自 Akshare 和 Tushare 等多个外部 API 的数据收集器的定时执行。

本章节涵盖了主要的调度器实现、任务配置以及协调逻辑。关于具体的数据收集器，请参阅[**Akshare 数据收集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.2-akshare-data-collectors)、[**Tushare 数据收集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.3-tushare-data-collectors)和[**Watchlist 数据收集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.4-watchlist-data-collection)等。

### 调度器架构

编排系统围绕一个中央实例构建BlockingScheduler，该实例协调多个专用数据收集器。每个收集器只需初始化一次，即可在计划执行期间重复使用。



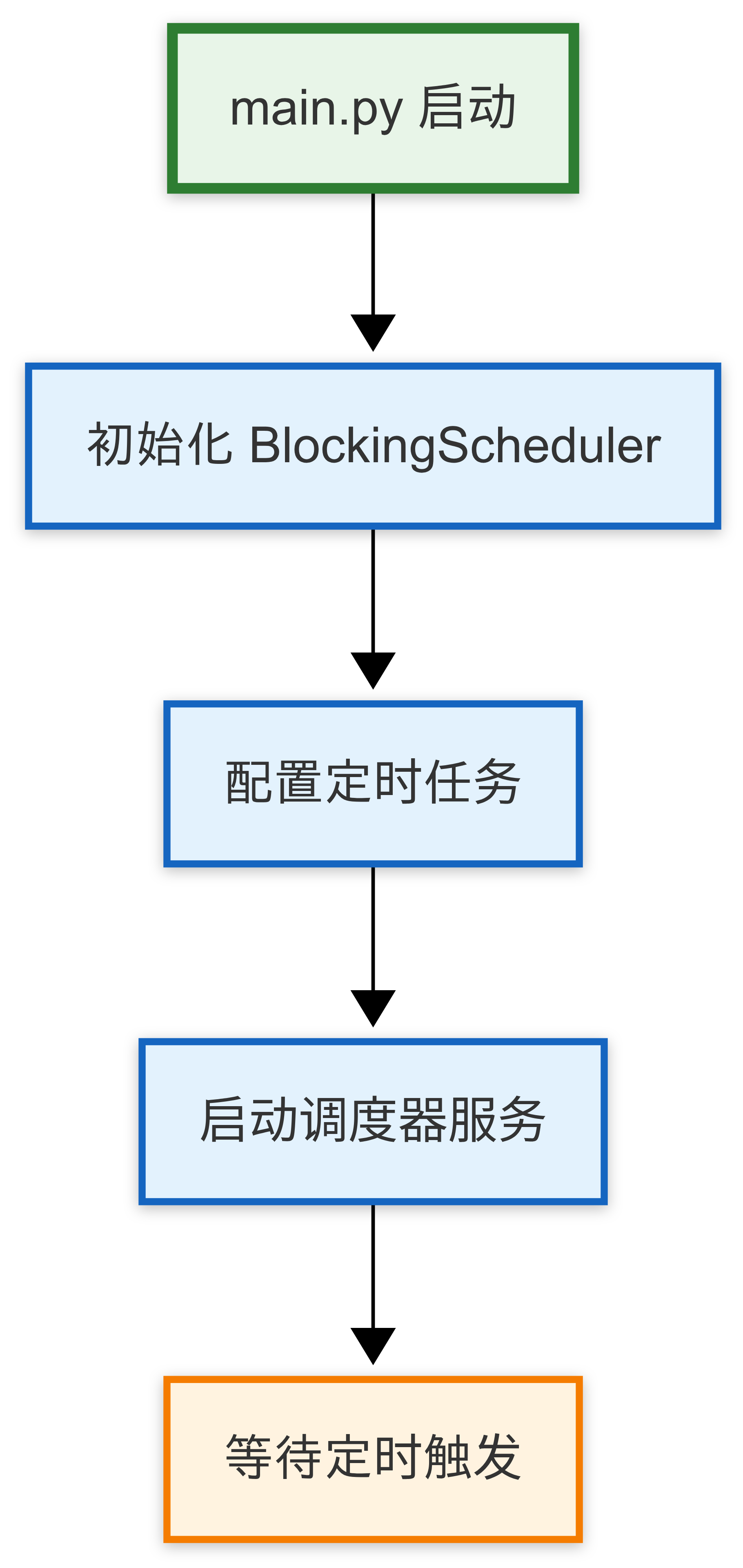
**关键组件说明**

* **编排层(Orchestration Layer)**
* main.py入口点
* BlockingScheduler调度器
* **包装函数层(Wrapper Functions Layer)**

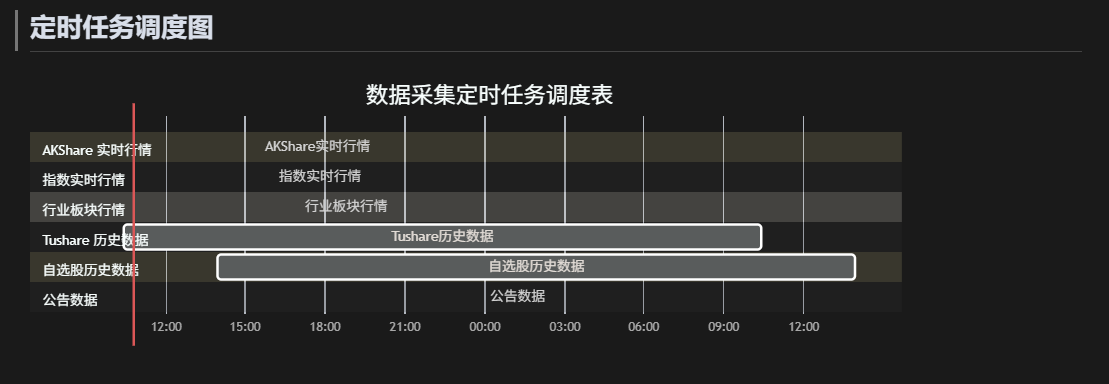
6个主要的包装函数，对应不同的数据采集任务

* collect\_akshare\_realtime(): AKShare实时行情采集
* collect\_tushare\_historical(): Tushare历史数据采集
* collect\_akshare\_index\_realtime(): 指数实时行情采集
* collect\_akshare\_industry\_board\_realtime(): 行业板块行情采集
* collect\_akshare\_stock\_notices(): 公告数据采集
* run\_watchlist\_history\_collection(): 自选股历史数据采集
* **采集器实例层(Collector Instances Layer)**
* Tushare采集器：历史数据、实时数据
* ak\_collector: AKShare实时行情采集器
* tushare\_hist\_collector: Tushare历史数据采集器
* index\_collector: 指数实时行情采集器
* industry\_board\_collector: 行业板块行情采集器
* notice\_collector: 公告数据采集器
* watchlist\_collector: 自选股历史数据采集器
* **配置层(Configuration Layer)**
* DATA\_COLLECTORS统一配置管理

### 系统启动流程

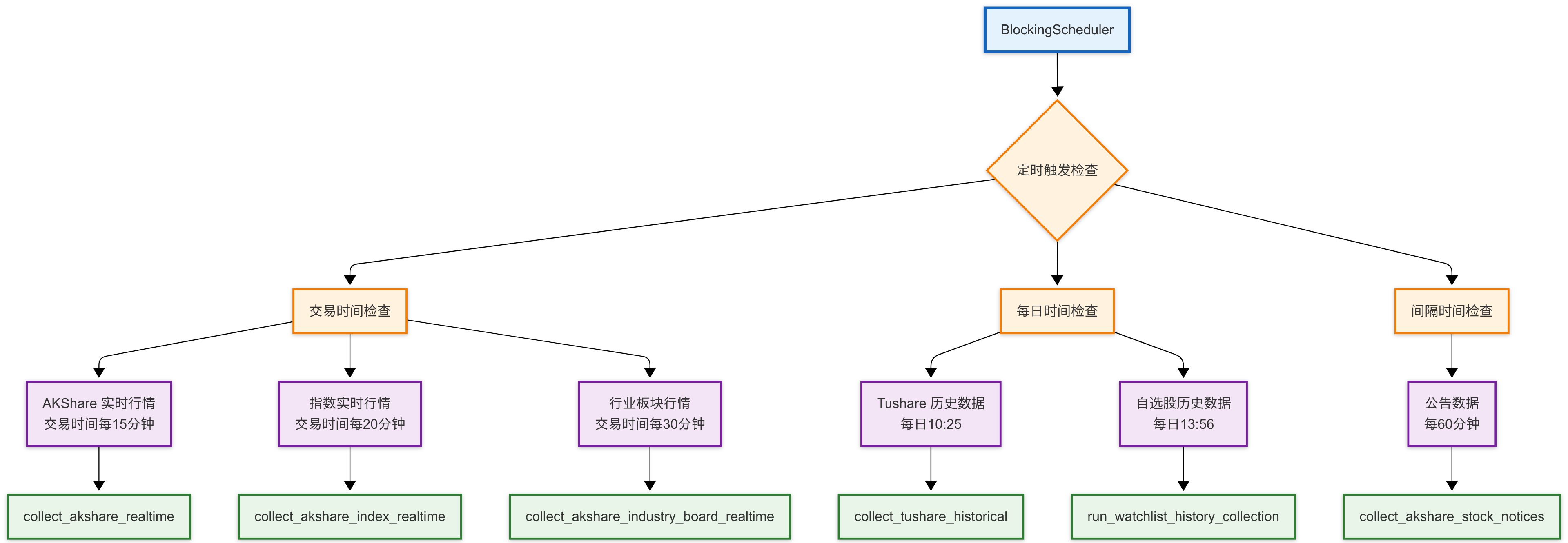


### 采集任务计划

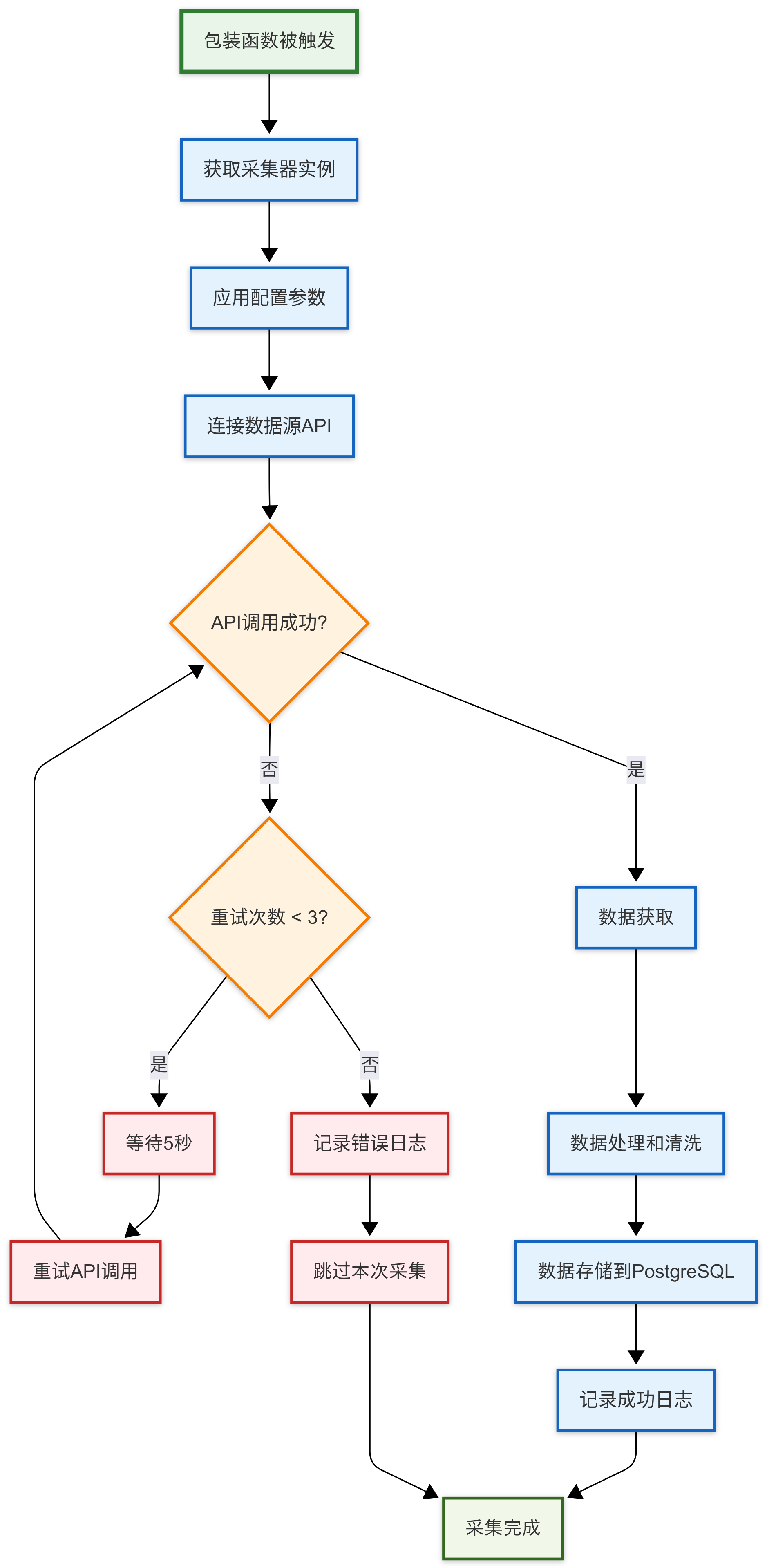


|  |  |  |  |
| --- | --- | --- | --- |
| **任务名称** | **调度策略** | **包装函数** | **采集器实例** |
| **AKShare 实时行情** | 交易时间每15分钟 | collect\_akshare\_realtime() | ak\_ collector |
| **Tushare 历史数据** | 每日10:25 | collect\_tushare\_historical() | tushare\_hist\_collector |
| **指数实时行情** | 交易时间每20分钟 | collect\_akshare\_index\_realtime() | index\_collector |
| **行业板块行情** | 交易时间每30分钟 | collect\_akshare\_industry\_board\_realtime() | industry\_board\_collector |
| **公告数据** | 每60分钟 | collect\_akshare\_stock\_notices() | notice\_collector |
| **自选股历史数据** | 每日13:56 | run\_watchlist\_history\_collection() | watchlist\_collector |

**定时任务调度逻辑**



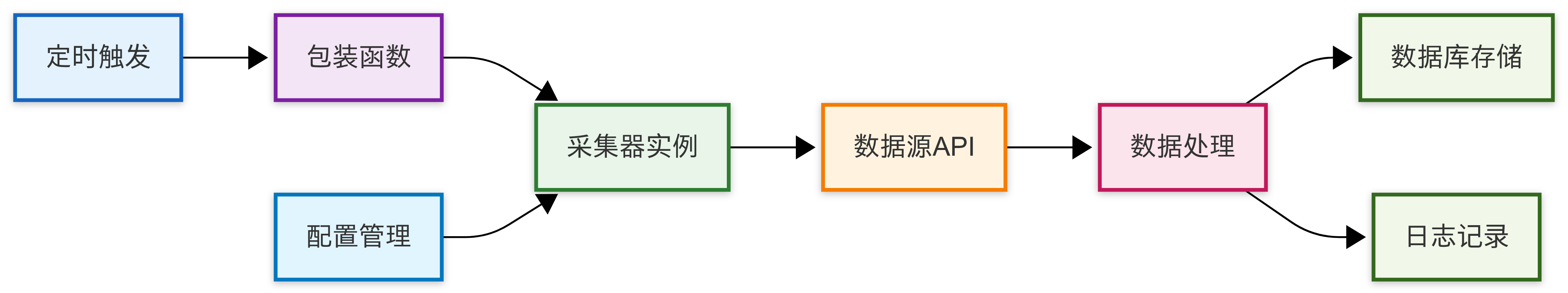
### 数据采集执行流程



**错误处理和重试机制**

* **重试配置**
* 最大重试次数: 3次
* 重试延迟: 5秒
* 请求超时: 30秒
* 最大连接错误: 10次

### 数据流图



## Akshare数据采集器

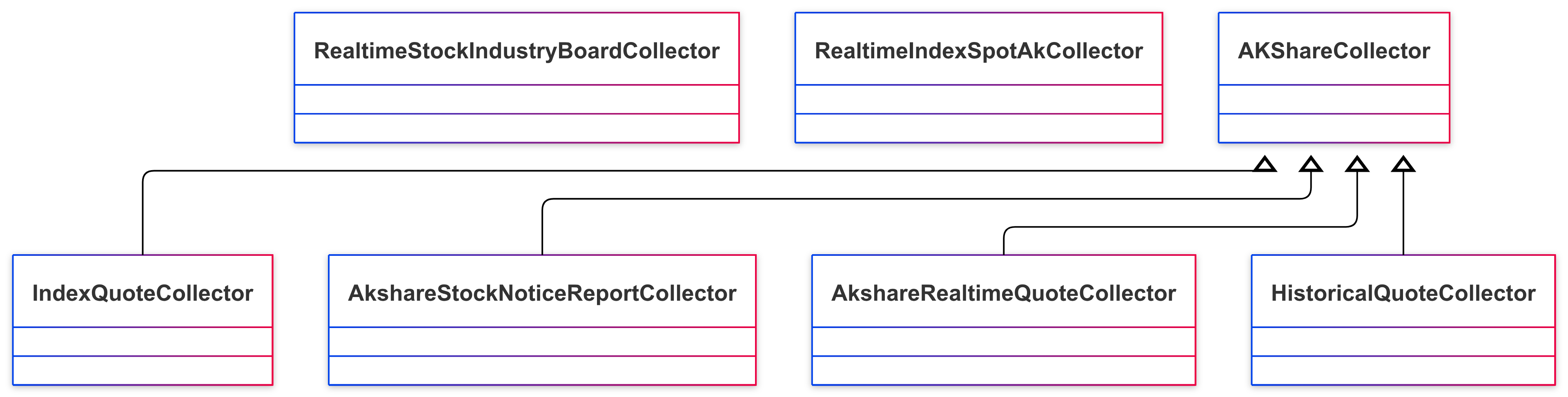
本章节介绍 Akshare 数据收集子系统，该系统负责通过 Akshare API 自动收集中国股市数据。该系统收集实时行情、行业板块表现、股票公告、指数数据和历史交易数据，并将其存储在 PostgreSQL 数据库表中。

整体数据采集编排和调度请参见[**采集调度**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.1-collection-orchestration)。 Tushare API 的历史数据采集请参见[**Tushare 数据采集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.3-tushare-data-collectors)。

### 架构概述

Akshare 数据收集器采用分层架构，并具有通用基类，为 API 调用、错误处理和数据库操作提供共享功能。每个收集器都专门处理 Akshare API 中的特定数据类型。

#### 类层次结构



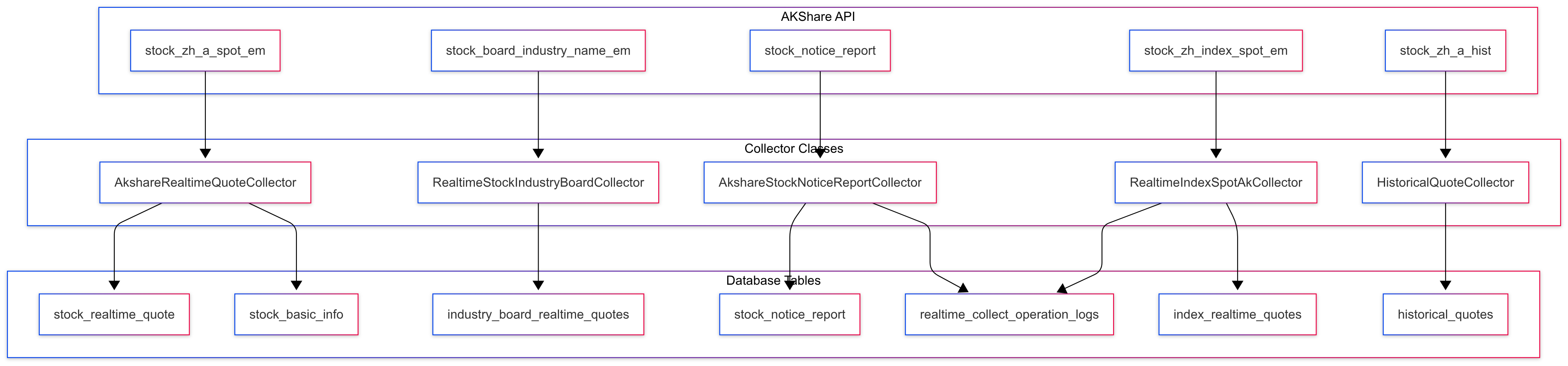
说明

* 核心基类：AKShareCollector，其子类负责不同类型的数据采集。
* 部分采集器（如行业板块、指数实时采集器）未继承基类，但实现方式类似，属于同一采集体系。
* 采集器之间通过配置、数据库操作等实现复用和扩展。

代码路径：

* [backend\_core/data\_collectors/akshare/base.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/base.py#L13-L17)
* [backend\_core/data\_collectors/akshare/realtime.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/realtime.py#L18-L19)
* [backend\_core/data\_collectors/akshare/historical.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/historical.py#L21-L22)
* [backend\_core/data\_collectors/akshare/realtime\_stock\_notice\_report\_ak.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/realtime_stock_notice_report_ak.py#L31-L32)
* [backend\_core/data\_collectors/akshare/realtime\_stock\_industry\_board\_ak.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/realtime_stock_industry_board_ak.py#L10-L14)
* [后端核心/数据收集器/akshare/realtime\_index\_spot\_ak.py](https://github.com/wangj9601/stock_quote_analayze/blob/f38b222c/backend_core/data_collectors/akshare/realtime_index_spot_ak.py#L10-L15)

#### 数据流架构



**Akshare 数据采集器数据流图架构说明**

* AKShare API 层

**作用**：作为数据采集的源头，负责从互联网获取原始金融数据。

**主要接口**：

* stock\_zh\_a\_spot\_em：获取A股实时行情数据。
* stock\_board\_industry\_name\_em：获取行业板块实时数据。
* stock\_notice\_report：获取股票公告数据。
* stock\_zh\_index\_spot\_em：获取指数实时行情数据。
* stock\_zh\_a\_hist：获取A股历史行情数据。
* Collector Classes层

**作用：**采集器类负责调度、调用API、处理数据，并将数据写入数据库。

**主要采集器：**

* AkshareRealtimeQuoteCollector：采集A股实时行情，写入stock\_realtime\_quote和stock\_basic\_info表。
* RealtimeStockIndustryBoardCollector：采集行业板块实时数据，写入industry\_board\_realtime\_quotes表。
* AkshareStockNoticeReportCollector：采集公告数据，写入stock\_notice\_report和realtime\_collect\_operation\_logs表。
* RealtimeIndexSpotAkCollector：采集指数实时行情，写入index\_realtime\_quotes和realtime\_collect\_operation\_logs表。
* HistoricalQuoteCollector：采集A股历史行情，写入historical\_quotes表。
* Database Tables 层

作用**：存储采集到的结构化数据，供后续分析、查询和展示使用。**

主要表：

* stock\_realtime\_quote：A股实时行情数据。
* stock\_basic\_info：股票基础信息。
* industry\_board\_realtime\_quotes：行业板块实时行情。
* stock\_notice\_report：股票公告数据。
* realtime\_collect\_operation\_logs：采集操作日志，记录采集过程中的操作和异常。
* index\_realtime\_quotes：指数实时行情数据。
* historical\_quotes：A股历史行情数据。
* 数据流动说明
* 每个API接口的数据流向对应的采集器类。
* 采集器类对数据进行处理后，写入一到多个数据库表。
* 采集器在采集和写入过程中，会将操作日志写入realtime\_collect\_operation\_logs表，便于追踪和监控。

### 采集器基类

基类AKShareCollector提供所有 Akshare 数据收集器使用的通用功能，包括配置管理、日志设置、重试机制和基本数据操作。

**核心功能**：

|  |  |  |
| --- | --- | --- |
| **特性** | **实现** | **用途** |
| **配置** | DATA\_COLLECTORS['akshare'] | 集中收集器设置 |
| **日志记录** | logging.getLogger()带有文件输出 | 运行监控和调试 |
| **重试逻辑** | \_retry\_on\_failure()采用指数退避算法 | API 可靠性和错误恢复 |
| **数据验证** | \_safe\_value()方法 | 清理数字数据转换 |

### 实时数据采集器

#### 实时行情报价采集器

该系统AkshareRealtimeQuoteCollector通过 API 端点收集所有 A 股股票的实时行情stock\_zh\_a\_spot\_em。它维护两个包含全面市场数据的主要数据库表。

**数据库模式**：

|  |  |  |
| --- | --- | --- |
| **表** | **主键** | **用途** |
| **stock\_basic\_info** | code | A股代码基础信息 |
| **stock\_realtime\_quote** | code | A股实时行情数据和价格 |

采集器使用 PostgreSQL UPSERT 操作实现复杂的冲突解决：

**INSERT** **INTO** stock\_realtime\_quote (...)

**VALUES** (...)

**ON** CONFLICT (code) DO UPDATE **SET**

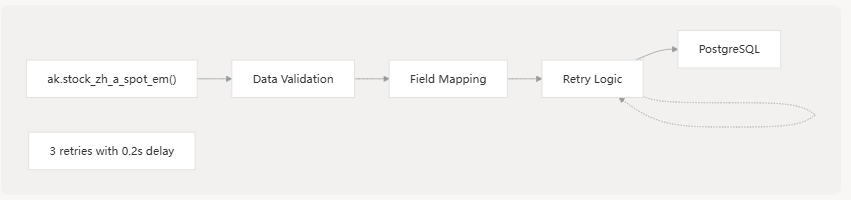
current\_price = EXCLUDED.current\_price,

change\_percent = EXCLUDED.change\_percent,

volume = EXCLUDED.volume,

...

**数据处理管道**：

采集器处理带有中文字段映射的数据：

* 代码→code
* 名称→name
* 最新价→current\_price
* 涨跌幅→change\_percent
* 成交量→volume
* …

#### 行业板块行情采集器

RealtimeStockIndustryBoardCollector,采集行业板块数据，stock\_board\_industry\_name\_em接口为行业分类提供汇总的市场数据。

**主要特性**：

* **完整数据替换：**插入新记录之前清除现有数据
* **综合指标：**价格、成交量、营业额、领涨股等
* **多字段映射：**中英文12个数据字段

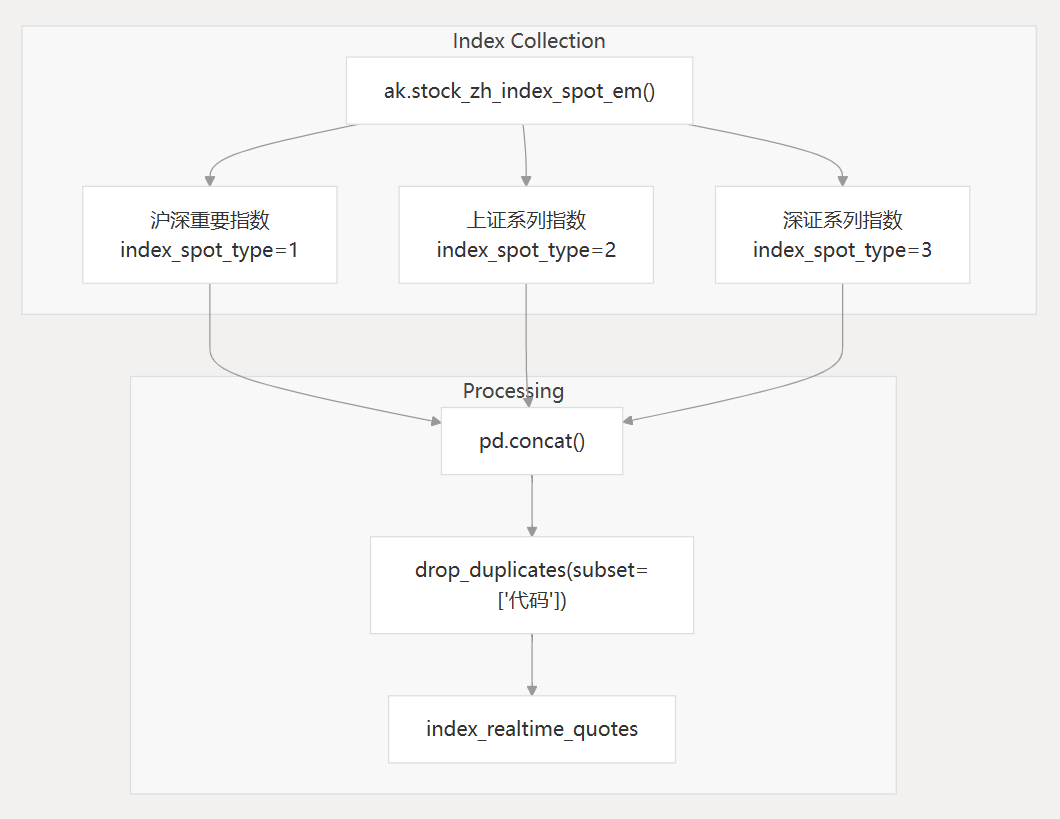
采集器系统字段映射说明：

|  |  |  |
| --- | --- | --- |
| **中文字段** | **英文字段** | **数据类型** |
| **板块代码** | board\_code | TEXT |
| **板块名称** | board\_name | TEXT |
| **最新价** | latest\_price | REAL |
| **涨跌额** | change\_amount | REAL |
| **领涨股** | leading\_stock\_name | TEXT |

#### 指数行情采集器

RealtimeIndexSpotAkCollector采集国内主要股票指数的实时数据，分为重要指数、上证指数、深证指数等。

**指数类别**：



#### 新闻公告研报采集器

使用AkshareStockNoticeReportCollector采集公司公告和监管文件stock\_notice\_report。它支持多种公告类别，并提供全面的搜索功能。

**公告类别**：

|  |  |
| --- | --- |
| **类别** | **用途** |
| **全部** | 所有公告类型 |
| **重大事项** | 大型企业活动 |
| **财务报告** | 财务报告 |
| **融资公告** | 融资公告 |
| **风险提示** | 风险警告 |
| **资产重组** | 资产重组 |
| **信息变更** | 信息变更 |
| **持股变动** | 持股变动 |

采集器提供单个分类和批量两种采集方式：

**def** **collect\_multiple\_types**(self, date\_str: **Optional**[str] = None) -> bool:

notice\_types = ["全部", "重大事项", "财务报告", "融资公告",

"风险提示", "资产重组", "信息变更", "持股变动"]

**for** notice\_type **in** notice\_types:

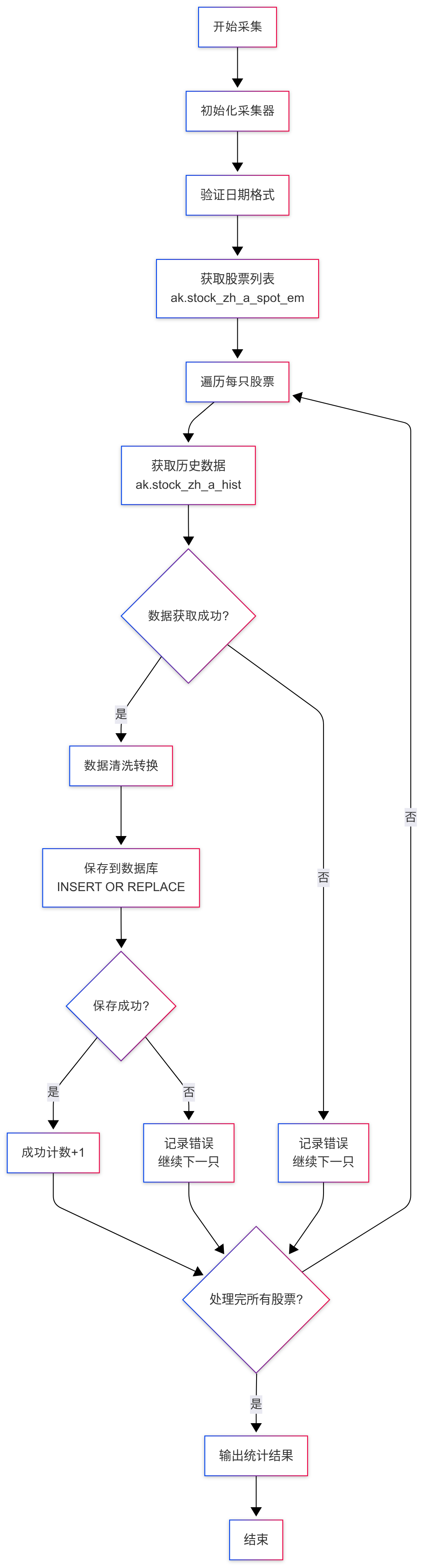
self.collect\_stock\_notices(symbol=notice\_type, date\_str=date\_str)

time.sleep(2) *# Rate limiting*

### 历史行情数据采集器

使用HistoricalQuoteCollector采集A股历史交易数据。

**采集流程**：



### 错误处理策略

采集器实现了多种错误恢复机制：

|  |  |  |
| --- | --- | --- |
| **错误类型** | **策略** | **临界点** |
| 连接错误 | 延迟5秒重试 | 最多错误10次 |
| API故障 | 跳过，继续 | 记录并继续 |
| 信号中断 | 优雅退出 | 即时 |
| 数据错误 | 安全值转换 | 继续处理 |

信号处理允许在长时间运行的操作期间正常关闭：

**def** **\_setup\_signal\_handlers**(self):

signal.signal(signal.SIGINT, self.\_signal\_handler)

signal.signal(signal.SIGTERM, self.\_signal\_handler)

**def** **\_signal\_handler**(self, signum, frame):

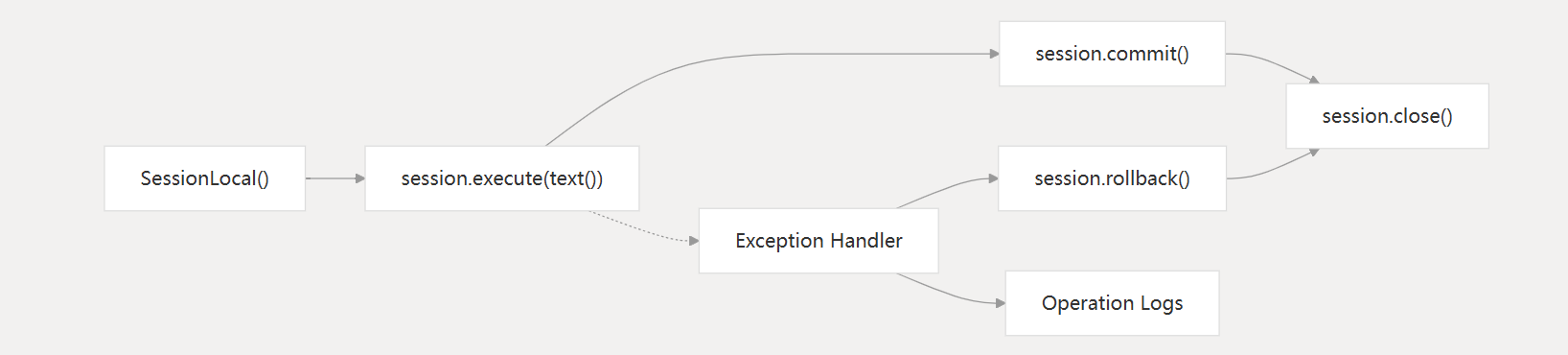
self.logger.info("接收到中断信号，正在安全退出...")

self.should\_stop = True

### 数据存储模式

所有 Akshare 收集器都为数据库操作、错误日志记录和数据完整性实现一致的模式。

**数据库连接管理**：



**操作日志**：

realtime\_collect\_operation\_logs所有采集器都使用标准化字段维护操作日志：

|  |  |
| --- | --- |
| **字段** | **说明** |
| **operation\_type** | 采集操作类型 |
| **operation\_desc** | 采集操作描述 |
| **affected\_rows** | 处理的记录数 |
| **status** | **success**或者**error** |
| **error\_message** | 详细错误信息 |
| **created\_at** | 时间戳 |

**重试和冲突解决**：

**ON CONFLICT**收集器使用 PostgreSQL 的 upsert子句实现数据库级别的冲突解决**DO UPDATE SET**。这确保了并发操作期间的数据一致性，并优雅地处理重复键的情况。

## Tushare数据采集器

Tushare 数据采集子系统通过 Tushare API 提供专门的金融市场数据采集功能。该模块处理历史股票报价、实时市场数据和指数信息采集。

对于基于 Akshare 的数据收集，请参阅[**Akshare 数据采集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.2-akshare-data-collectors)。对于整体采集编排，请参阅**采集调度**。对于特定于自定义股票列表的数据采集，请参阅**自定义股票**[**列表数据采集**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.4-watchlist-data-collection)。

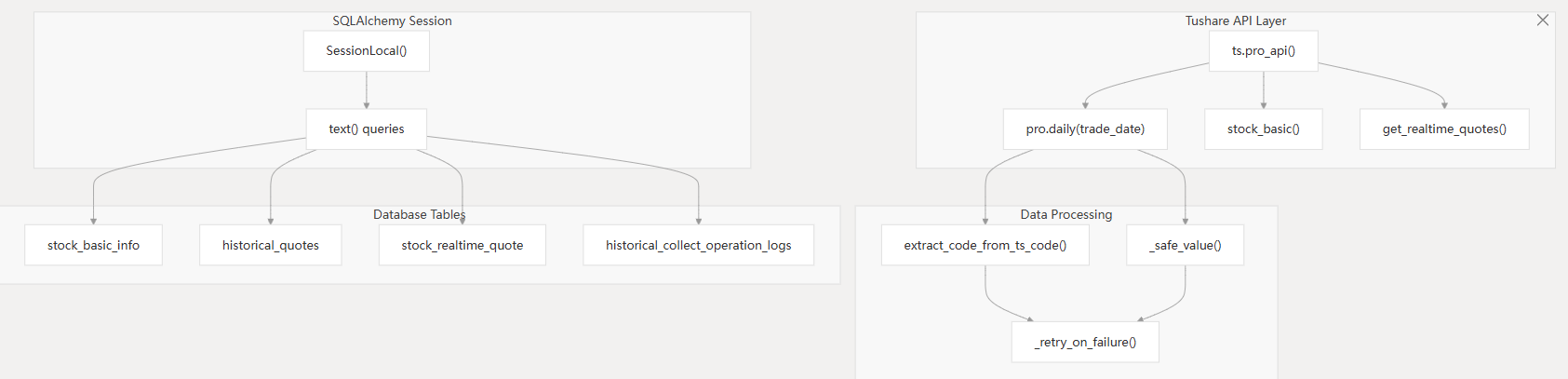
### 架构概述

Tushare 数据采集器遵循分层架构，具有共享基本功能和针对不同数据类型的专门实现。

#### 类层次结构



#### 数据库集成



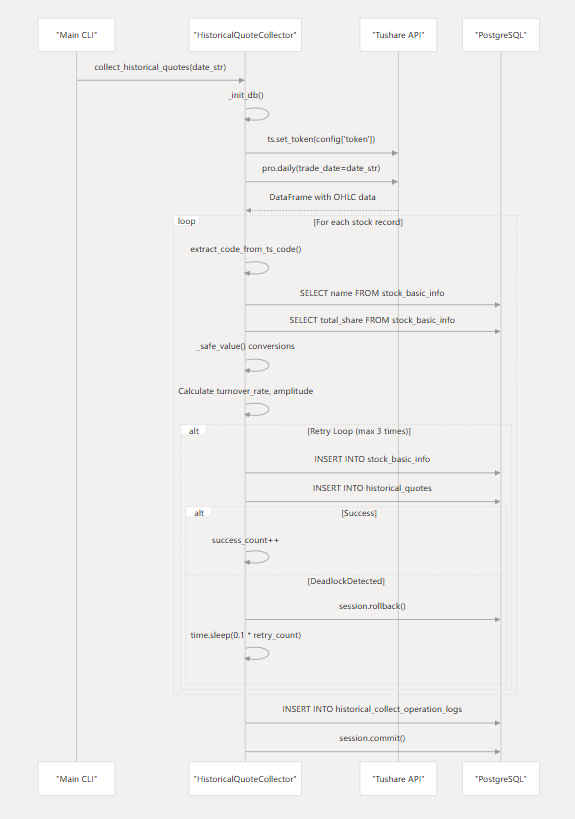
### 历史行情数据采集器

通过**HistoricalQuoteCollector**采集指定日期的A股历史交易数据。

#### 主要特性

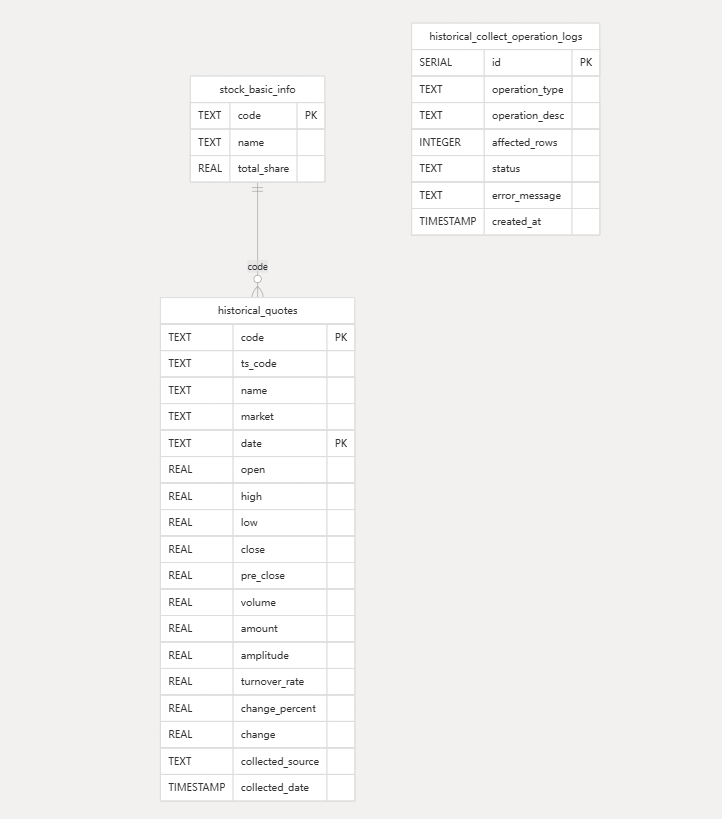
|  |  |  |
| --- | --- | --- |
| **特性** | **实现** | **目的** |
| 信号处理 | **\_setup\_signal\_handlers()** | SIGINT/SIGTERM 时正常关闭 |
| 死锁恢复 | 采用指数退避算法的重试机制 | 数据库事务弹性 |
| 数据验证 | **\_safe\_value()**用于数字转换 | 干净数据处理 |
| 股票代码解析 | **extract\_code\_from\_ts\_code()** | Tushare格式规范化 |
| 批处理 | 每100条记录提交一次 | 性能优化 |

#### 数据采集流程



#### 数据库模式操作

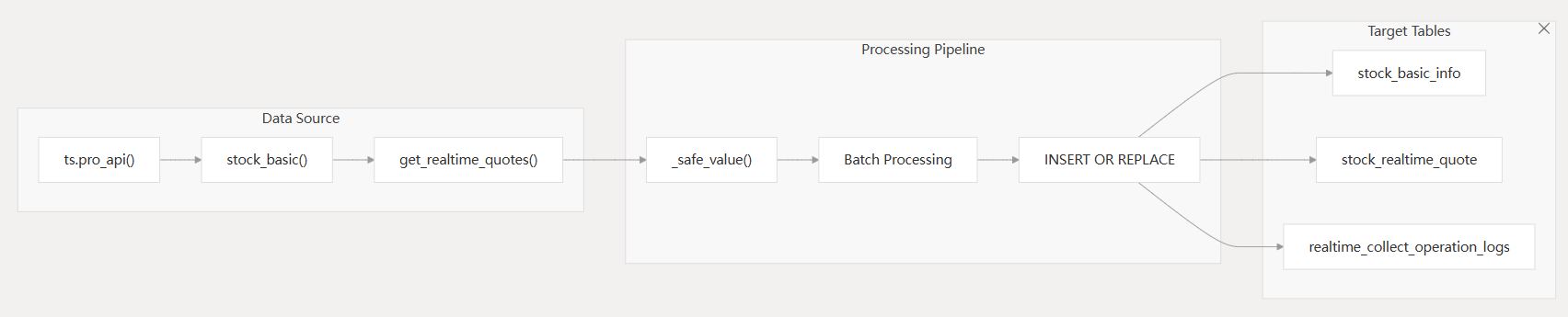
历史采集器管理三个主要相关表:



### 实时行情数据采集器

通过**RealtimeQuoteCollector**在交易时间内提供实时市场数据收集，并优化性能以实现频繁更新。

#### 采集流程



### 指数行情数据采集器

通过**IndexQuoteCollector**使用 Tushare 专门的**pro\_bar**接口采集市场指数数据。

#### 实现

采集器采用一种简化的方法，重点关注主要市场指数：

*# Core collection method*

**def** **collect\_index\_quotes**(self):

df = ts.pro\_bar(ts\_code='000001.SH', asset='I')

self.logger.info(f"采集到 {len(df)} 条指数行情数据")

### 配置和操作

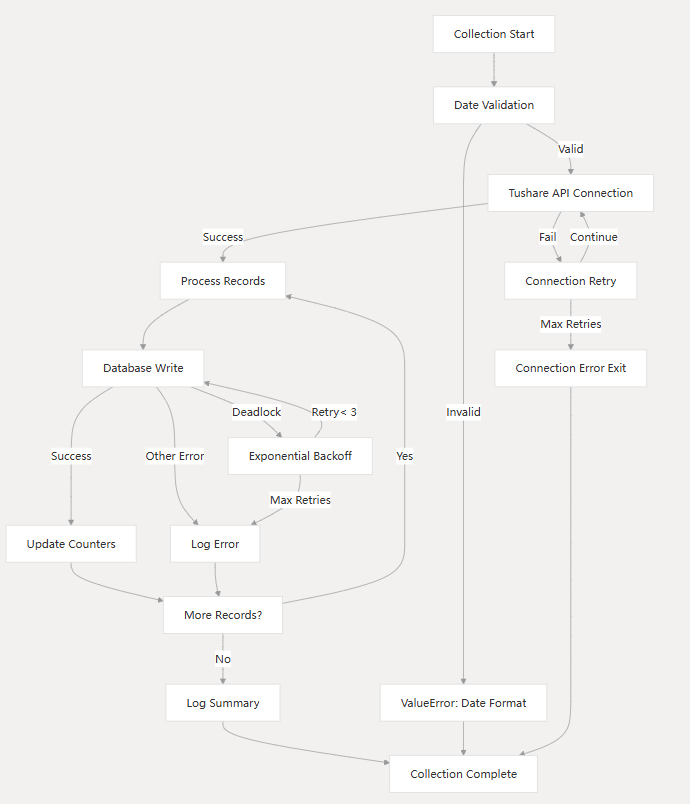
通过**IndexQuoteCollector**使用 Tushare 专门的**pro\_bar**接口采集市场指数数据。

#### CLI界面

主入口点提供对所有采集器的命令行访问：

|  |  |  |
| --- | --- | --- |
| **范围** | **选型** | **目的** |
| **--type** | **realtime，historical，index** | 采集器选择 |
| **--date** | **YYYYMMDD** 格式 | 历史数据日期（历史必需） |

#### 错误处理策略



### 数据质量和验证

#### 价值处理管道

收集器通过**\_safe\_value()**等方法实现强大的数据清理。

## 关注股票列表数据采集

### 目的和范围

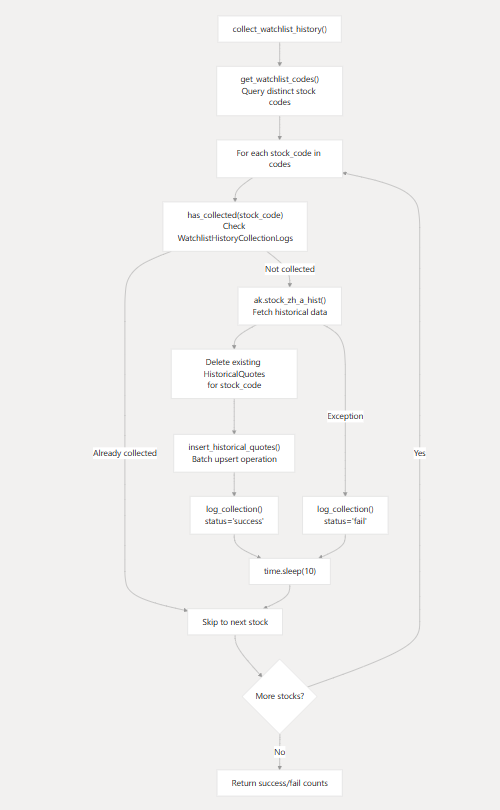
关注列表数据采集系统是数据采集层的一个专用组件，用于检索用户已添加到个人关注列表的股票的历史数据。与[**Tushare 数据采集器**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.3-tushare-data-collectors)中涵盖的全面历史数据采集器不同，该系统针对特定股票子集按需运行，以优化数据存储和采集效率。

该系统以计划任务的方式运行，确保用户当前跟踪的所有股票都有历史价格数据，支持**历史**[**数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.2-historical-data-apis)和[**历史数据接口**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.4-historical-data-interface)中描述的历史数据分析功能。

### 概述

关注列表数据采集充当用户管理的关注列表和历史数据存储之间的桥梁，仅采集用户正在主动关注的股票的全面历史数据。

### 采集工作流程



### 主要功能

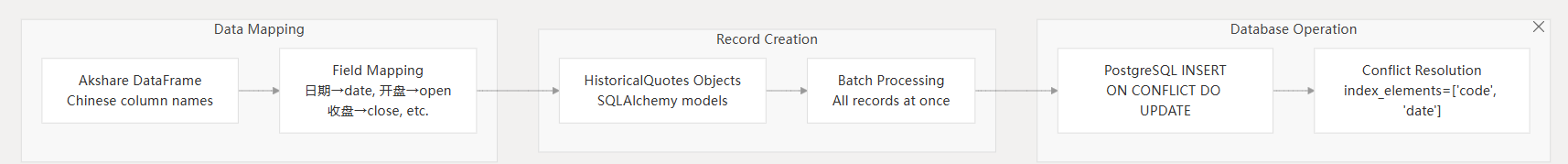
|  |  |  |
| --- | --- | --- |
| **功能** | **目的** | **返回类型** |
| **get\_watchlist\_codes()** | 从用户关注列表中检索唯一股票代码 | **List[str]** |
| **has\_collected()** | 检查股票的历史数据采集是否完成 | **Bool** |
| **insert\_historical\_quotes()** | 执行历史价格数据的批量更新 | **Int** |
| **log\_collection()** | 记录采集尝试的结果和错误 | **None** |
| **collect\_watchlist\_history()** | 采集流程的主要编排功能 | **Dict[str, int]** |

### 数据处理管道

该模块实施复杂的数据处理，以处理大量历史数据，并进行适当的错误处理和重复预防。

#### 历史数据插入流程

该**insert\_historical\_quotes()**函数使用 PostgreSQL 的子句执行复杂的 upsert 操作，ON CONFLICT以优雅地处理重复数据。



upsert 操作处理以下字段：

open,close,high,low,volume，amount,amplitude，change\_percent,changeturnover\_rate,

code，name,date

### 数据库集成

关注股票列表数据采集与多个数据库表集成，以保持数据的一致性和跟踪。

**表关系：**



**采集状态跟踪：**

系统通过以下**WatchlistHistoryCollectionLogs**模型维护全面的日志。

* 成功追踪：记录成功收集affected\_rows次数
* 错误处理：捕获失败集合的异常消息
* 重复预防：用于has\_collected()避免重复处理
* 审计跟踪：所有收集尝试的时间戳

### 调度集成

该采集器每天凌晨 2:00 作为计划任务运行，并与[**采集调度**](https://deepwiki.com/wangj9601/stock_quote_analayze/3.1-collection-orchestration)中描述的主要采集调度编排系统集成。

**采集参数：**

|  |  |  |
| --- | --- | --- |
| **参数** | **参数值** | **目的** |
| **开始日期** | **19940101** | 采集最近30年历史数据 |
| **结束日期** | 前一交易日 | 不包括当天 |
| **复权方式** | **Qfq(**前复权) |  |
| **休眠间隔** | 10秒 |  |

# 后端API服务

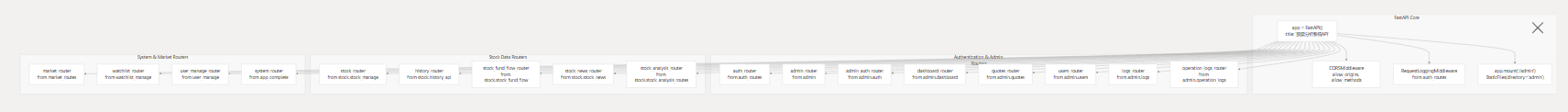
## 系统概述

本文档介绍基于 FastAPI 的 REST API 后端，该后端为股票分析系统提供数据访问、用户管理和业务逻辑服务。后端作为前端应用程序和数据库之间的主要接口，公开股票数据检索、用户身份验证、关注列表管理和其他管理功能的端点。

有关输入到这些 API 的数据采集流程，请参阅[**数据采集系统**](https://deepwiki.com/wangj9601/stock_quote_analayze/3-data-collection-system)。有关使用这些 API 的前端应用程序，请参阅[**前端应用程序**](https://deepwiki.com/wangj9601/stock_quote_analayze/5-frontend-applications)。有关数据库模型和架构的详细信息，请参阅[**数据库和存储**](https://deepwiki.com/wangj9601/stock_quote_analayze/6-database-and-storage)。

### FastAPI应用程序架构

后端 API 的结构是一个模块化的 FastAPI 应用程序，**backend\_api/main.py**它聚合了多个专用路由器。app实例充当编排层，配置中间件并通过统一服务公开所有 API 端点。

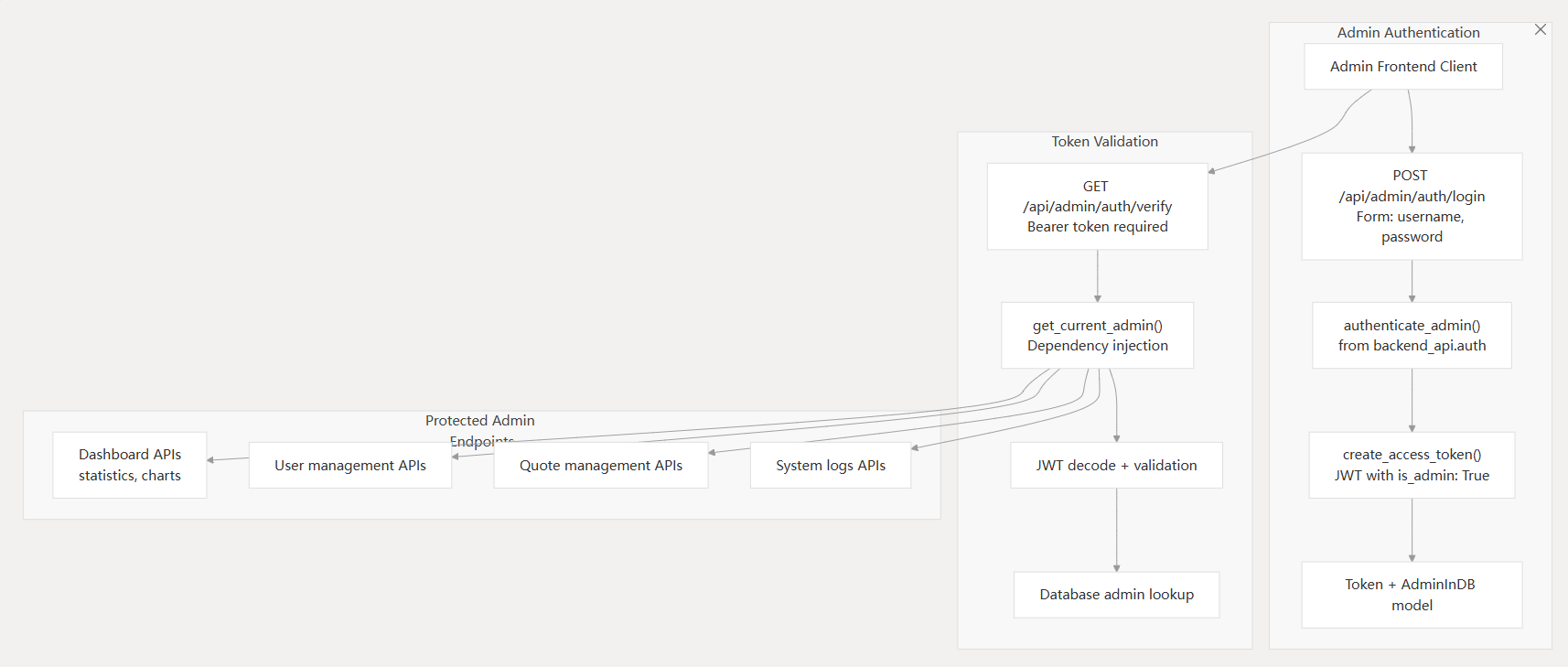


**路由器注册摘要**



### 身份验证和安全框架

该 API 通过专用身份验证路由器实现基于 JWT 的身份验证。管理员身份验证与常规用户身份验证分开处理，具有不同的端点和令牌管理。



**身份验证端点**



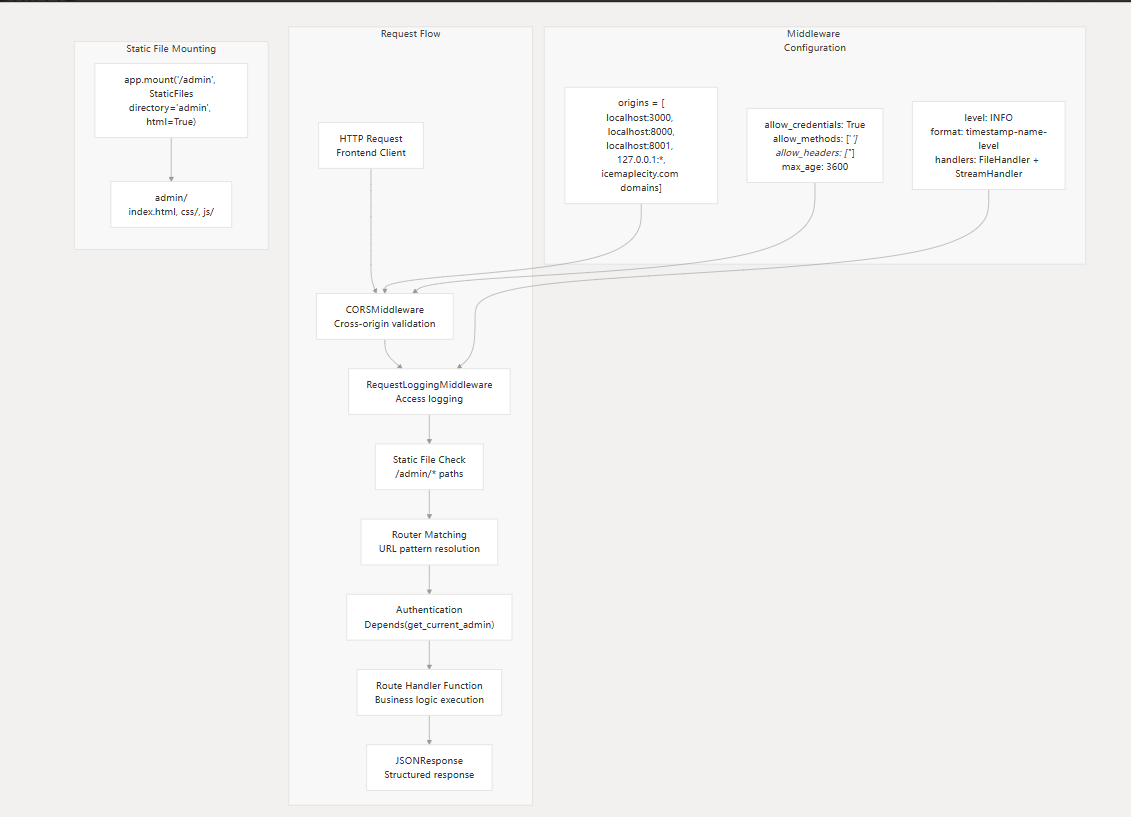
**令牌配置**

* **JWT 算法**：**HS256**具有可配置的密钥
* **令牌到期**：可通过以下方式配置**ACCESS\_TOKEN\_EXPIRE\_MINUTES**
* **令牌类型：**授权标头中的承载令牌
* **Admin Model：AdminInDB**用于序列化的 Pydantic 模型

资料来源：[backend\_api/admin/auth.py27-81](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L27-L81)

### 请求处理管道

FastAPI 应用程序在到达路由处理程序之前，会通过分层中间件堆栈处理请求。该管道包括 CORS 处理、请求日志记录、静态文件服务和身份验证。



**CORS配置详情**

该应用程序支持多个前端源，包括开发服务器和生产域：



**静态文件服务**

该/admin路径被安装用于直接从目录为管理界面提供服务，admin/并具有 HTML 后备支持。

资料来源：[backend\_api/main.py51-78](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L51-L78) [backend\_api/main.py81](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L81-L81) [backend\_api/main.py15-24](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L15-L24)

### API端点类别

后端会跨功能类别公开 API，这些类别映射到特定的路由器模块。每个类别都提供具有标准化响应格式的不同端点组。

#### 股票管理API（stock\_router）

通过 AKShare API 进行实时市场数据集成的核心股票数据操作。

**关键端点：**

|  |  |  |
| --- | --- | --- |
| **端点** | **方法** | **目的** |
| **/api/stock/quote** | Post | 批量实时报价检索 |
| **/api/stock/list** | Get | 使用查询/限制参数进行股票搜索 |
| **/api/stock/quote\_board** | Get | 涨幅最大的股票排名 |
| **/api/stock/quote\_board\_list** | Get | 按类型分页的股票排名 |
| **/api/stock/realtime\_quote\_by\_code** | Get | 个股实时报价 |
| **/api/stock/minute\_data\_by\_code** | Get | 日内分钟级数据 |
| **/api/stock/kline\_hist** | Get | 历史K线数据 |
| **/api/stock/kline\_min\_hist** | Get | 历史分钟K线数据 |
| **/api/stock/latest\_financial** | Get | 最新财务指标 |

**数据处理功能：**

* **safe\_float()**数据清理实用程序
* **DataFrameCache**用于 600 秒响应缓存
* **ak.stock\_bid\_ask\_em()**来自和的实时数据**ak.stock\_intraday\_em()**
* 非交易时间数据库回退

#### 历史数据API（history\_router）

具有导出功能的时间序列股票数据专用端点操作。

**主要特点：**

* **start\_date**使用**/end\_date**参数进行日期范围过滤
* 多个周期：每日、每周、每月、分钟间隔
* 数据调整类型：（qfq向前）、hfq（向后）、None
* CSV/Excel 导出功能
* 大型数据集的分页

#### 股票分析API（stock\_analysis\_router）

股票价格技术分析和预测服务。

**分析组件：**

* 技术指标：RSI、MACD、KDJ、布林带
* 具有置信区间的价格预测模型
* 支撑/阻力位识别
* 交易信号生成与风险评估
* 进入/退出点的关键水平分析

#### 系统管理API（多个router）

通过专用路由器模块进行后端管理。

**管理员身份验证**（**admin\_auth\_router**）

* **POST /api/admin/auth/login** - 管理员使用表单数据登录
* **GET /api/admin/auth/verify** - 令牌验证
* **GET /api/admin/auth/me** - 当前管理员资料
* **POST /api/admin/auth/logout** - 会话终止

**管理员身份验证**（**admin\_auth\_router**）

* **POST /api/admin/auth/login** - 管理员使用表单数据登录
* **GET /api/admin/auth/verify** - 令牌验证
* **GET /api/admin/auth/me** - 当前管理员资料
* **POST /api/admin/auth/logout** - 会话终止

**仪表板API**（**dashboard\_router**）

* 系统统计信息汇总
* 用户数量和活动指标
* 股票数据量统计
* 系统健康监控

**用户管理**（**users\_router**）

* 用户 CRUD 操作
* 账户状态管理
* 用户活动追踪
* 权限管理

#### 市场数据API（market\_router）

市场范围的数据聚合和指数跟踪。

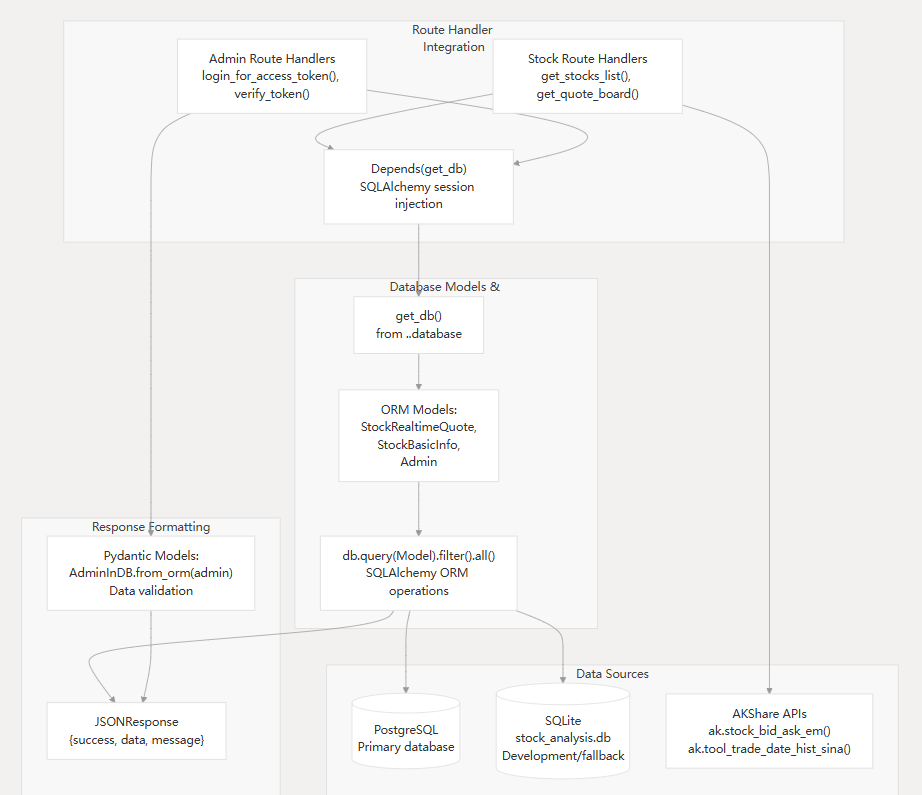
**覆盖范围：**

* 用户 CRUD 操作
* 账户状态管理
* 用户活动追踪
* 权限管理
* 上证综合指数
* 深证成指
* 行业板块表现
* 市值排名
* 交易量统计
* 热门股票识别

资料来源：[backend\_api/stock/stock\_manage.py36-44](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L36-L44) [backend\_api/stock/stock\_manage.py46-105](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L46-L105) [backend\_api/admin/auth.py29-81](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L29-L81)

### 数据库集成

API 服务通过 SQLAlchemy ORM 模型和依赖注入与数据库集成，以实现会话管理。数据库访问模式因端点需求而异。



**数据库访问模式**

|  |  |  |
| --- | --- | --- |
| **操作类型** | **执行** | **示例用法** |
| ORM 查询 | **db.query(StockRealtimeQuote)** | 股票报价检索 |
| 过滤 | **.filter(Model.field == value)** | 股票代码查询 |
| 排序 | **.order\_by(Model.field.desc())** | 涨幅最大的股票排名 |
| 分页 | **.limit(limit).all()** | 分页股票清单 |
| 外部API | **ak.stock\_bid\_ask\_em(symbol=code)** | 实时市场数据 |

**模型集成示例**

* **StockRealtimeQuote** - 来自数据库的实时报价数据
* **StockBasicInfo** - 用于搜索功能的股票元数据
* **Admin** - 管理员用户身份验证和个人资料数据
* **AdminInDB** - 用于管理员响应序列化的 Pydantic 模型

资料来源：[backend\_api/stock/stock\_manage.py4-14](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L4-L14) [backend\_api/stock/stock\_manage.py109-128](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L109-L128) [backend\_api/admin/auth.py9-23](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L9-L23)

### 错误处理和响应格式

该 API 在所有端点上实现了标准化的错误处理和响应格式。HTTP 状态代码遵循 REST 约定，并带有详细的错误消息，方便调试和用户反馈。

**标准响应格式：**



**模型集成示例**

* **401 Unauthorized：**JWT 令牌无效或过期
* **403 Forbidden：**权限不足
* **422 Unprocessable Entity：**请求数据格式无效

身份验证系统针对凭证验证失败和令牌过期场景提供特定的错误响应，使前端应用程序能够适当地处理身份验证状态。

资料来源：[backend\_api/admin/\_\_init\_\_.py77-94](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/__init__.py#L77-L94) [backend\_api/admin/\_\_init\_\_.py102-122](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/__init__.py#L102-L122)

## FastAPI 应用程序核心

本文档介绍了作为股票分析系统后端 API 服务主要入口点的中央 FastAPI 应用程序设置。它涵盖应用程序配置、中间件设置、路由器集成、静态文件挂载以及应用程序生命周期管理。

有关特定 API 端点和功能的信息，请参阅[**股票管理 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.2-stock-management-apis)**、**[**历史数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.3-historical-data-apis)**、**[**市场数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.4-market-data-apis)**、**[**股票分析和新闻 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.5-stock-analysis-and-news-apis)以及[**身份验证和管理 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.6-authentication-and-admin-apis)。

### FastAPI应用程序配置

主 FastAPI 应用程序已配置**backend\_api/main.py**全面的元数据和日志设置。该应用程序采用结构化的配置方法，并为 API 文档提供中文支持。

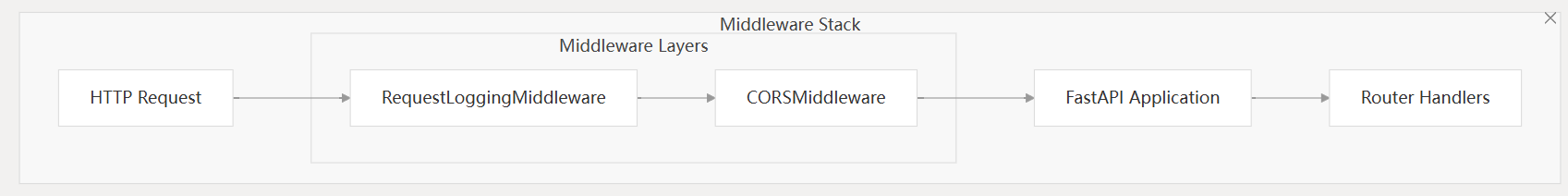


该应用程序配置了日志记录，可写入文件（app.log）和控制台输出，并支持中文字符的 UTF-8 编码和包括文件名和行号在内的详细格式。

资料来源：[backend\_api/main.py44-49](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L44-L49)

### 中间件堆栈配置

该应用程序实现了分层中间件架构，用于横切关注点，包括请求日志记录和 CORS 处理。



请求处理流程。

#### 请求日志中间件

RequestLoggingMiddleware从 auth\_routes 模块导入，并提供全面的请求日志记录功能以用于监控和调试目的。

#### CORS配置

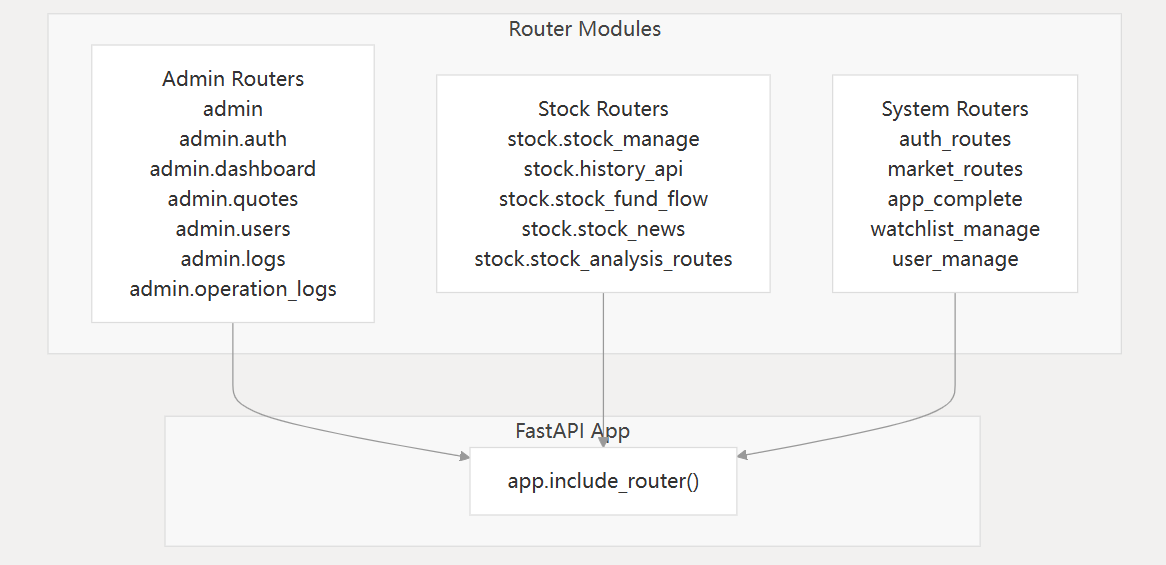
CORS 中间件配置了对开发和生产环境的广泛来源支持：

|  |  |
| --- | --- |
| **环境** | **来源** |
| 开发环境 | localhost:3000, localhost:8000, localhost:8001 |
| 本地网络 | 192.168.3.60:8000，192.168.3.60:5000。。。 |
| 生产环境 | icemaplecity.com，www.icemaplecity.com（HTTP/HTTPS） |

CORS 配置允许所有方法、标头和凭证，最长使用期限为 3600 秒。

### 路由器集成架构

该应用程序通过全面的导入和注册系统集成了多个专用路由器。



**路由器注册模式**

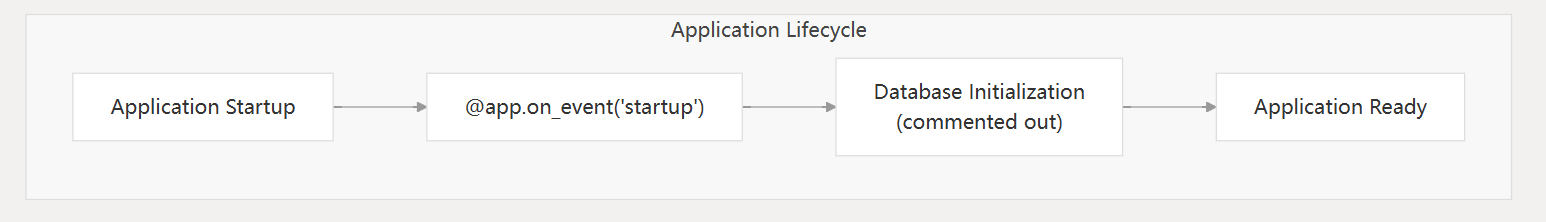
应用程序遵循一致的路由器注册模式：

1. **导入阶段：**所有路由器都使用描述性别名导入
2. 注册阶段：每个路由器都使用app.include\_router()

|  |  |  |
| --- | --- | --- |
| **路由器类别** | **路由器名称** | **目的** |
| 验证 | **auth\_router** | 用户身份验证和 JWT 处理 |
| 管理核心 | **admin\_router** | 基本管理功能 |
| 管理员授权 | **admin\_auth\_router** | 管理员身份验证端点 |
| 行政管理 | **dashboard\_router，quotes\_router，users\_router** | 管理面板功能 |
| 管理员日志 | **logs\_router，operation\_logs\_router** | 系统日志接口 |
| 股票管理核心 | **stock\_router** | 核心股票数据管理 |
| 股票分析 | **stock\_analysis\_router，stock\_news\_router** | 高级股票功能 |
| 市场数据 | **market\_router** | 市场指数和行业数据 |
| 用户功能 | **watchlist\_router，user\_manage\_router** | 用户特定功能 |

### 应用程序生命周期管理

FastAPI 应用程序实现生命周期挂钩以进行正确的初始化和清理。



## 股票管理API

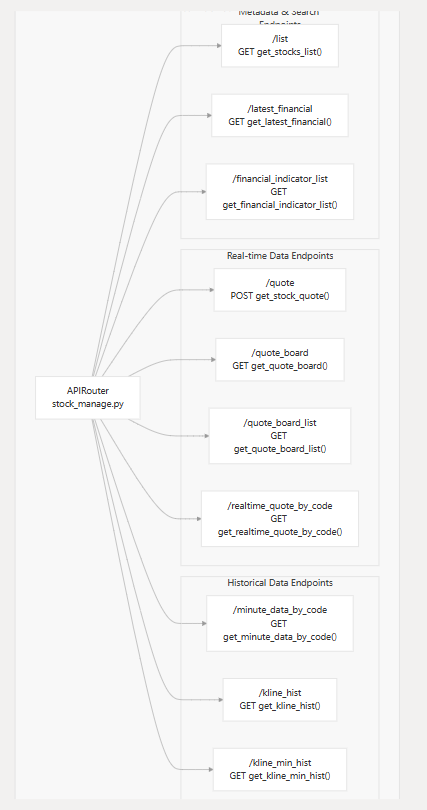
### 目的和范围

本文档涵盖 backend\_api 系统中的核心股票数据管理 API，特别是**/api/stock**路由器中定义的端点。这些 API 为前端应用程序提供实时和历史股市数据、金融指标和交易信息。这些 API 主要从 AKShare 等外部提供商获取数据，并维护本地缓存以优化性能。

有关股票分析和预测 API 的信息，请参阅[**股票分析和新闻 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.5-stock-analysis-and-news-apis)。有关市场范围的数据和指数，请参阅[**市场数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.4-market-data-apis)。有关历史数据导出功能，请参阅[**历史数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.3-historical-data-apis)。

### API端点架构

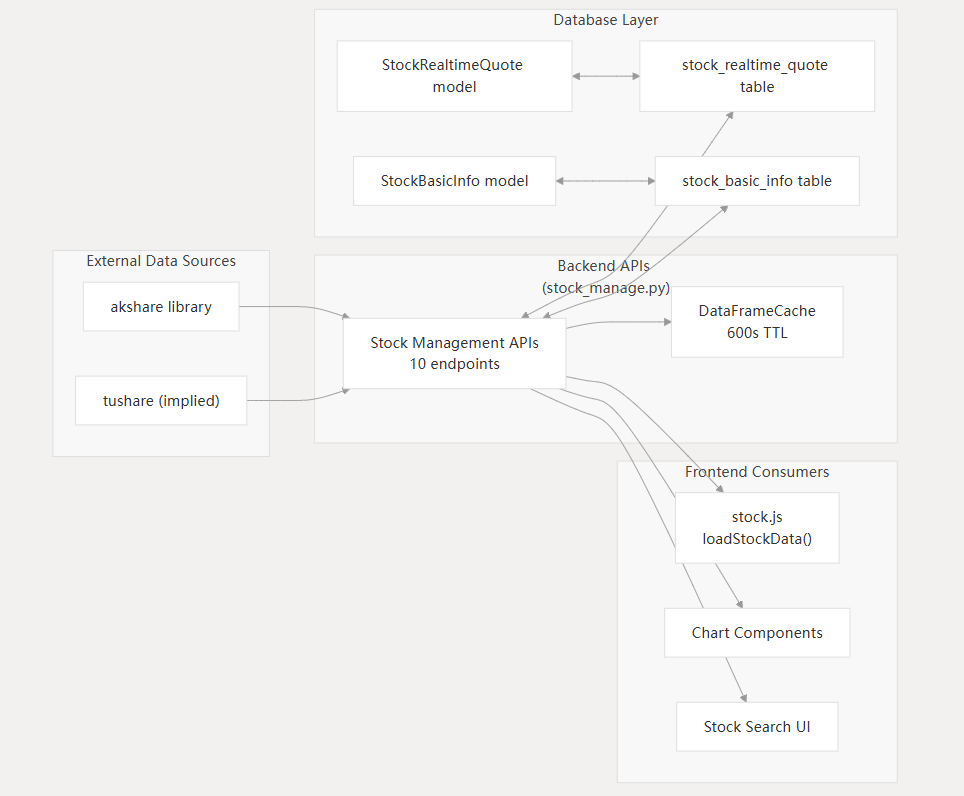
股票管理 API 实现为 FastAPI 路由器，具有集中式结构，可处理多种数据类型和来源。



资料来源：[backend\_api/stock/stock\_manage.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L36-L36)

### 核心数据流架构

股票管理 API 实现了一种多层数据访问模式，结合了外部 API 调用、数据库缓存和内存缓存。



资料来源：[backend\_api/stock/stock\_manage.py17-34](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L17-L34) [backend\_api/stock/stock\_manage.py14](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L14-L14) [前端/js/stock.js596-635](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L596-L635)

### 实时报价API

#### 批量报价检索 (POST /quote)

通过单个请求检索多只股票的实时报价。支持周末回退到数据库数据。

**功能：**get\_stock\_quote()  
**输入：**{"codes": ["000001", "600519", ...]}  
**输出：**包含当前价格、变化、交易量等的股票报价对象数组。

**逻辑流程：**

1. 检查当前日期是否为周末
2. StockRealtimeQuote如果是周末：从数据库查询表
3. 若为工作日：调用ak.stock\_bid\_ask\_em()每个股票代码
4. 返回标准化的报价格式

资料来源：[backend\_api/stock/stock\_manage.py 46-105](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L46-L105)

#### 个股报价(GET /realtime\_quote\_by\_code)

使用平均价格等计算字段检索单个股票的详细实时报价。

**功能：**get\_realtime\_quote\_by\_code() **参数：（**code股票代码**）  
特殊功能：**

* **average\_price**如果未提供，则根据交易量和换手率计算
* 将所有数字字段格式化为 2 位小数
* 包括市盈率和周转率等扩展字段

**资料来源：**[backend\_api/stock/stock\_manage.py 269-316](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L269-L316)

#### 行业板块报价API

##### 行业板块简易报价（GET /quote\_board）

从数据库中按成交量涨幅百分比返回前 N 只股票。

**功能：**get\_quote\_board() **查询：**直接在数据库中stock\_realtime\_quote查询ORDER BY change\_percent DESC

##### 行业板块高级报价（GET /quote\_board\_list）

提供具有分页和市场过滤功能的综合排名功能。

**功能：**get\_quote\_board\_list() **参数：**直接在数据库中stock\_realtime\_quote查询ORDER BY change\_percent DESC

* **ranking\_type：**上涨/下跌/成交量/周转率
* **market：**all/sh/sz/bj/cy（上海/深圳/北京/创业板）
* **page,page\_size：**分页支持

**实现细节：**

* 使用 Pandas 对数据库转储进行 DataFrame 操作
* 按股票代码前缀筛选市场（6=上海、0/3=深圳、8/4=北京）
* 计算价格变化量等派生字段
* clean\_nan()使用实用程序处理 NaN 值

资料来源：[backend\_api/stock/stock\_manage.py131-161](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L131-L161) [backend\_api/stock/stock\_manage.py164-266](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L164-L266)

### K线实时/历史数据API

#### 分钟级数据（GET /minute\_data\_by\_code）

提供带有交易日检测的日内逐分钟交易数据。

**功能：**get\_minute\_data\_by\_code()

**主要特点：**

* ak.tool\_trade\_date\_hist\_sina()交易日历的用途
* 非交易日回落至前一交易日
* 返回交易日和非交易日的不同数据结构：
* 交易日：ak.stock\_intraday\_em()- 简单价格/交易量
* 非交易日：ak.stock\_zh\_a\_hist\_pre\_min\_em()- 完整的 OHLC 数据

资料来源：[backend\_api/stock/stock\_manage.py319-396](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L319-L396)

#### K线历史数据（GET /kline\_hist）

每日 K 线（蜡烛图）数据，具有可调整的日期范围和调整类型。

**功能：get\_kline\_hist()  
参数：**

* period：“每日”（固定）
* start\_date,end\_date: YYYY-MM-DD 格式
* adjust：“qfq”（前复权）默认

**数据来源：**ak.stock\_zh\_a\_hist()

资料来源：[backend\_api/stock/stock\_manage.py398-453](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L398-L453)

#### 分钟K线历史数据（GET /kline\_min\_hist）

较短时间范围内的日内分钟级 K 线数据。

**功能：**get\_kline\_min\_hist()

**参数：**

* period：1/5/15/30/60分钟
* start\_datetime,end\_datetime：完整的日期时间字符串
* adjust：调整类型（1分钟数据禁用）

**数据来源：**ak.stock\_zh\_a\_hist\_min\_em()

**资料来源：**[backend\_api/stock/stock\_manage.py 456-514](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L456-L514)

### 股票搜索和元数据API

##### 股票清单搜索（GET /list）

使用数据库查询提供股票代码和名称的搜索功能。

**功能：get\_stocks\_list()  
实现：**

* 使用StockBasicInfo模型进行数据库查询
* 支持代码和名称字段的部分匹配
* like()使用带有通配符模式的SQLAlchemy运算符
* 可配置结果限制（默认 15）

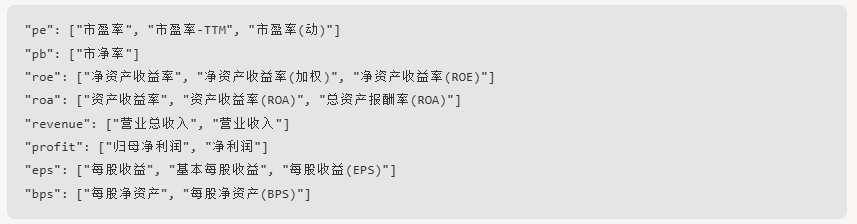
资料来源：[backend\_api/stock/stock\_manage.py 108-128](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L108-L128)

### 财务数据API

#### 最新财务指标（GET / latest\_financial）

检索最近报告期的关键财务指标。

**功能：**get\_latest\_financial()  
**数据源：**ak.stock\_financial\_abstract()  
**指标映射：**



资料来源：[backend\_api/stock/stock\_manage.py516-588](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L516-L588)

#### 财务指标时间序列（GET /financial\_indicator\_list）

提供跨多个报告期的历史财务指标。

**功能：**get\_financial\_indicator\_list()  
**数据来源：**ak.stock\_financial\_abstract\_ths()  
**报告类型：**“按报告期”、“按年度”、“按单季度”

资料来源：[backend\_api/stock/stock\_manage.py590-631](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L590-L631)

### 缓存和性能优化

**DataFrameCache 实现**

系统为 DataFrame 对象实现了线程安全的内存缓存。



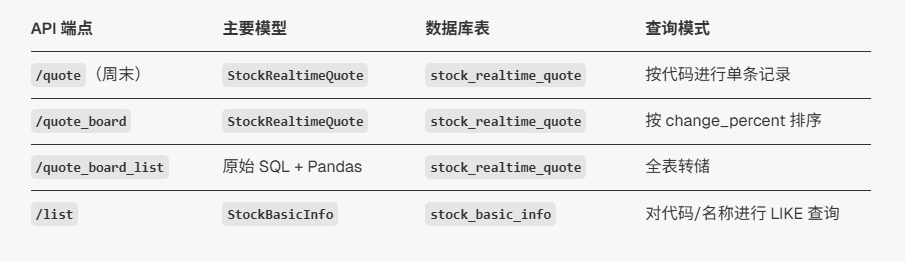
**线程安全**：用于threading.Lock并发访问保护 **TTL 管理**：基于time.time()比较的自动过期 **使用模式**：当前已实例化但未在代码库中主动使用

资料来源：[backend\_api/stock/stock\_manage.py17-34](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L17-L34)

### 数据库集成模式

#### 模型使用

API 与 SQLAlchemy ORM 模型集成以进行数据库操作：



#### 数据库会话管理

* next(get\_db())手动会话创建的用途
* 明确db.close()要求清理
* Depends(get\_db)搜索端点内的依赖注入

资料来源：[backend\_api/stock/stock\_manage.py63-78](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L63-L78)

### 错误处理和数据验证

#### 实用函数

**safe\_float(value)：**将值转换为浮点数并进行空处理

* 将 None、空字符串、破折号（“-”）视为 None
* 转换错误时返回 None

**clean\_nan(obj)：**递归 NaN 和无穷大移除

* 处理嵌套字典和列表
* math.isnan()将值转换math.isinf()为 None

#### 响应模式

所有端点都遵循一致的 JSON 响应格式：



资料来源：[backend\_api/stock/stock\_manage.py38-44](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L38-L44)

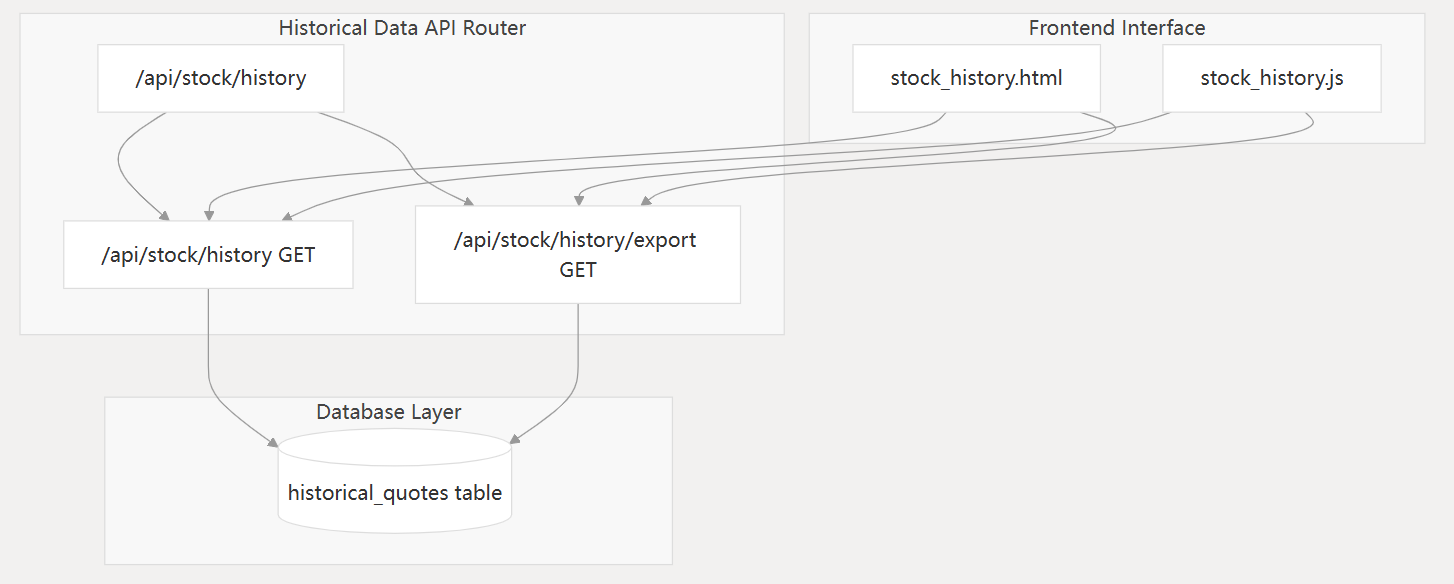
## 历史数据API

本文档介绍历史数据 API，这些 API 提供对历史股价数据的访问，并支持分页、日期筛选和 Excel 导出功能。这些 API 作为历史数据接口的后端基础，使用户能够检索、筛选和导出历史股市信息。

实时股票数据接口请参见[股票管理接口](https://deepwiki.com/wangj9601/stock_quote_analayze/4.2-stock-management-apis)。行情概览及指数数据接口请参见[行情数据接口](https://deepwiki.com/wangj9601/stock_quote_analayze/4.4-market-data-apis)。

### API端点概述

历史数据 API 实现为 FastAPI 路由器，具有两个用于数据检索和导出功能的主要端点。



资料来源：[backend\_api/stock/history\_api.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L17-L17)

### 数据检索端点

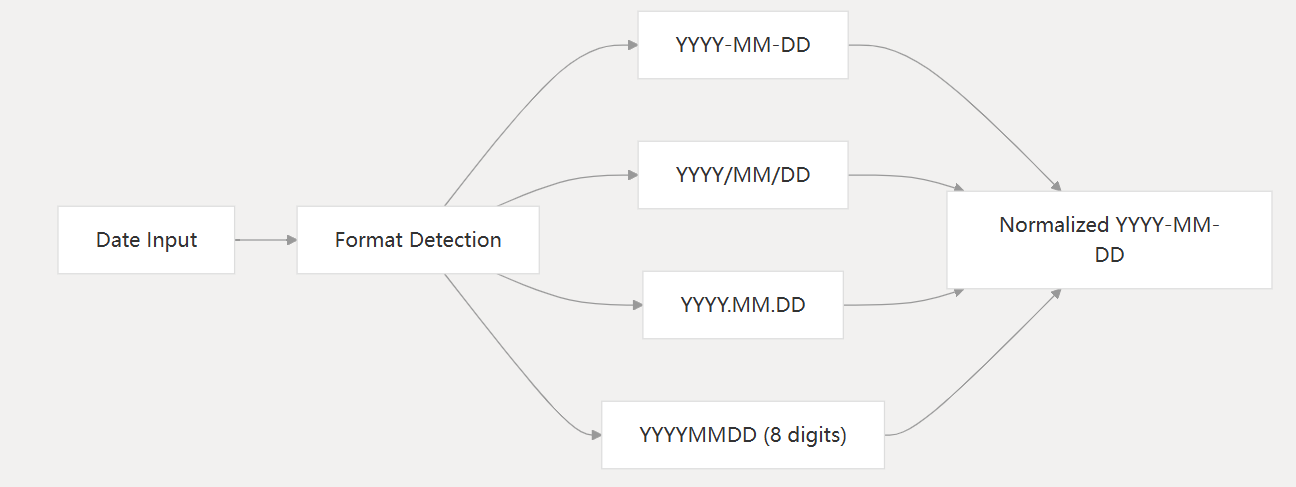
主要端点GET /api/stock/history提供对历史股票数据的分页访问，并具有过滤功能。

#### 请求参数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **范围** | **类型** | **是否必需** | **默认值** | **描述** |
| **code** | **String** | **是** | **-** | 股票代码 |
| **start\_date** | **String** | **否** | **无** | 过滤开始日期 (YYYY-MM-DD) |
| **end\_date** | **String** | **否** | **无** | 过滤结束日期 (YYYY-MM-DD) |
| **page** | **String** | **否** | **1** | 页码（≥1） |
| **size** | **String** | **否** | **20** | 页面大小（1-N） |

#### 日期格式处理

系统通过format\_date\_yyyymmdd函数支持多种日期格式：



资料来源：[backend\_api/stock/history\_api.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L19-L34)

#### 数据库查询结构

API 根据提供的参数构建动态 SQL 查询：

* 基本查询：SELECT code,name,date, open, close, high, low, volume,amount,change\_percent,change FROM historical\_quotes WHERE code = :code
* 日期过滤：AND date >= :start\_date附加AND date <= :end\_date条件
* 排序：ORDER BY date DESC首先按最新数据排序
* 分页：LIMIT :limit OFFSET :offset用于基于页面的结果

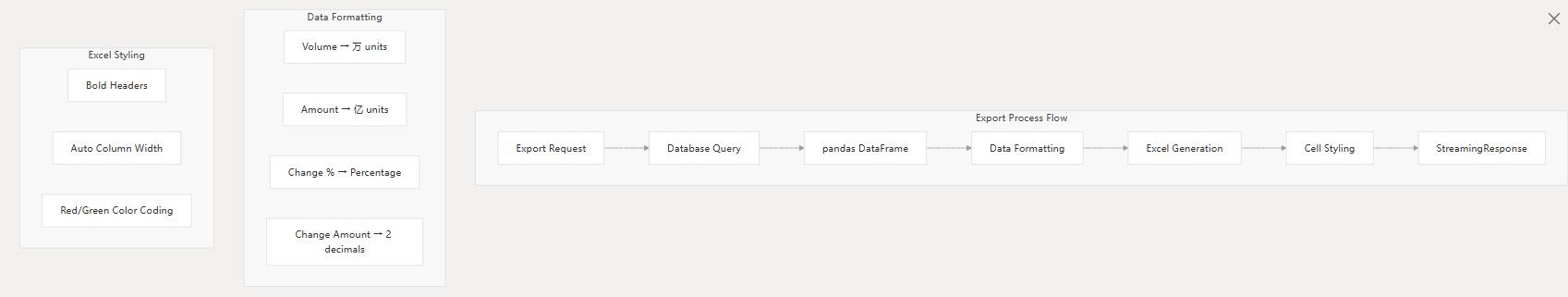
#### 响应格式

端点返回带有分页元数据的 JSON 对象：



### Excel导出功能

该GET /api/stock/history/export端点提供具有高级格式化功能的 Excel 文件生成。



#### 数据格式化功能

导出过程包括复杂的数据格式化：

* 成交量格式：将原始体积转换为“万”（10,000）个单位
* 金额格式：将原始金额转换为“亿”（1亿）单位
* 涨跌额百分比：格式为百分比，小数点后 2 位
* 找零金额：四舍五入至小数点后2位

#### Excel样式功能

生成的Excel文件包括：

* 列标题：中文标题，加粗格式
* 自动调整大小：列宽根据内容自动调整
* 颜色编码：正向变化为红色（#FF0000），负向变化为绿色（#009933）
* 字体样式：更改值的粗体格式

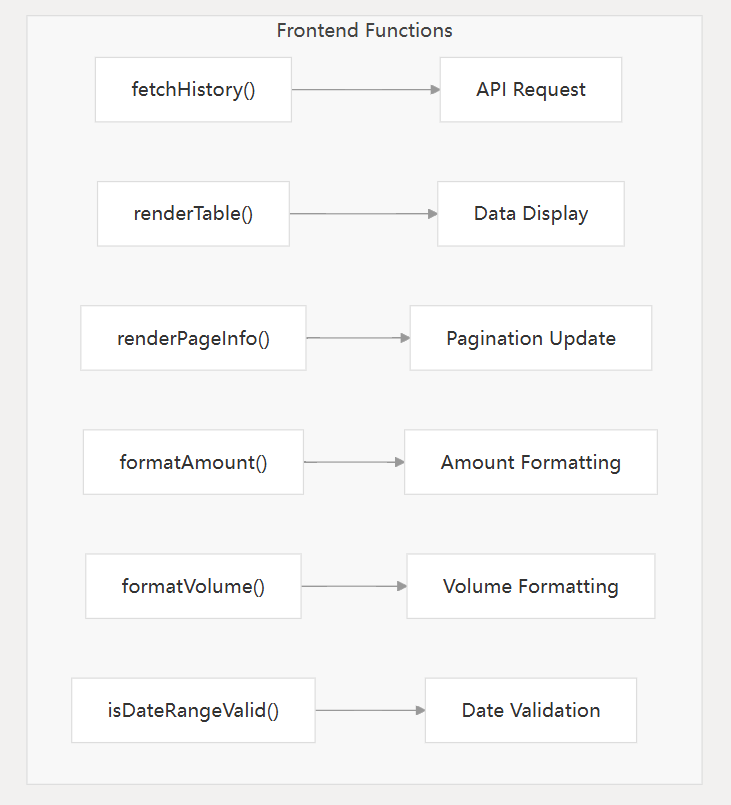
### 前端界面集成

历史数据界面提供了一种用户友好的方式，通过专用网页与 API 进行交互。

#### 核心组件



#### 关键JavaScript函数



资料来源：[前端/js/stock\_history.js](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock_history.js#L46-L63)

#### 用户界面功能

该界面提供了全面的数据访问控制：

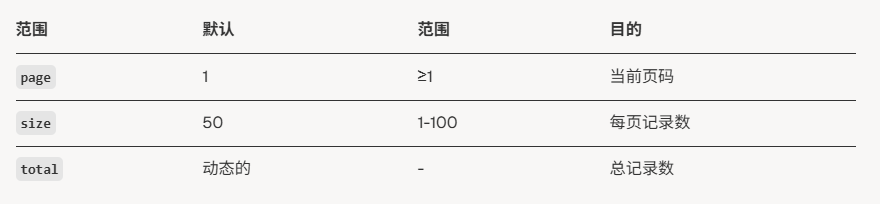
* 日期范围选择：带有验证的开始和结束日期选择器
* 实时查询：搜索按钮触发即时数据刷新
* 导出功能：使用当前过滤器直接下载 Excel
* 分页控制：第一页、上一页、下一页、最后一页导航
* 响应式设计：针对移动设备优化的布局，具有自适应尺寸

资料来源：[前端/stock\_history.html 140-145](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock_history.html#L140-L145)

### 分页和数据管理

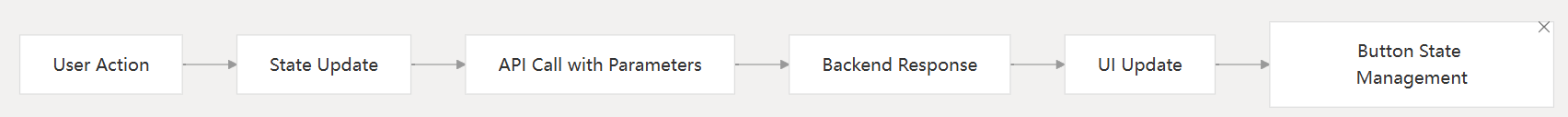
该系统实现了强大的分页功能，并在前端和后端组件之间实现了一致的状态管理。

#### 分页参数



#### 状态同步

前端维护分页状态并通过 URL 参数和 JavaScript 变量与后端同步：



### 错误处理和验证

该系统包括对数据完整性和用户输入的全面验证。

#### 日期验证

该**isDateRangeValid**函数确保逻辑日期范围：

* 接受多种日期格式（YYYY-MM-DD、YYYY/MM/DD）
* 验证结束日期不早于开始日期
* 为无效范围提供用户反馈

#### 后端数据处理

该format\_date\_yyyymmdd函数提供强大的日期解析功能，并针对各种输入格式进行回退处理，确保无论输入格式如何变化，数据库查询都保持一致。

资料来源：[前端/js/stock\_history.js](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock_history.js#L12-L18)；[backend\_api/stock/history\_api.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L19-L34)

## 市场数据API

本文档涵盖股票分析系统后端 API 层提供的市场数据 API。这些 API 为前端应用程序提供涵盖整个市场的数据，包括指数、行业板块和板块表现指标。有关单个股票数据 API，请参阅[**股票管理 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.2-stock-management-apis)。有关历史数据 API，请参阅[**历史数据 API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.3-historical-data-apis)。

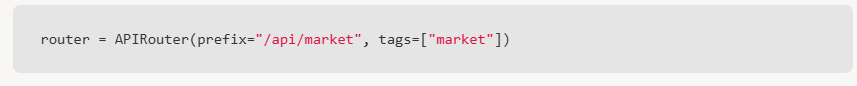
### 概述

市场数据 API 在基于 FastAPI 的后端实现，并提供用于检索实时和汇总市场数据的 REST 端点。这些 API 充当 PostgreSQL 数据库（由数据收集系统填充）与显示市场概览和行业分析的前端应用程序之间的接口。

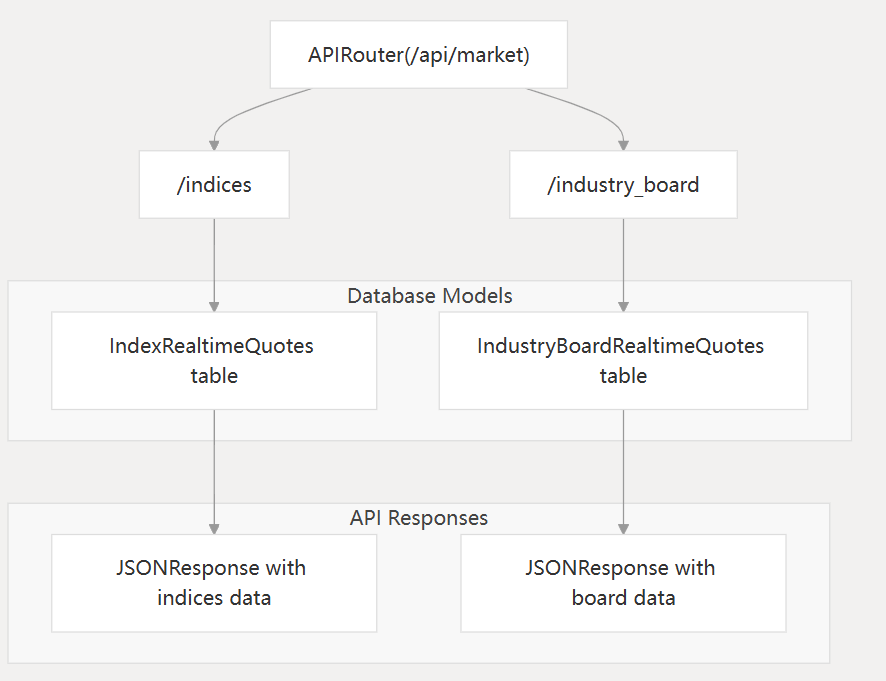
资料来源：[backend\_api/market\_routes.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/market_routes.py#L18-L18)

### API路由结构

市场数据功能由专用 FastAPI 路由器组织，其配置如下：

所有市场数据端点都以“market”标签为前缀并分组。

### 市场数据API端点



### 市场指数API

端点/api/market/indices从数据库表中检索主要市场指数的实时数据IndexRealtimeQuotes。

#### 端点详细信息

**方法：**GET  
**路径：**/api/market/indices  
**功能：**get\_market\_indices()

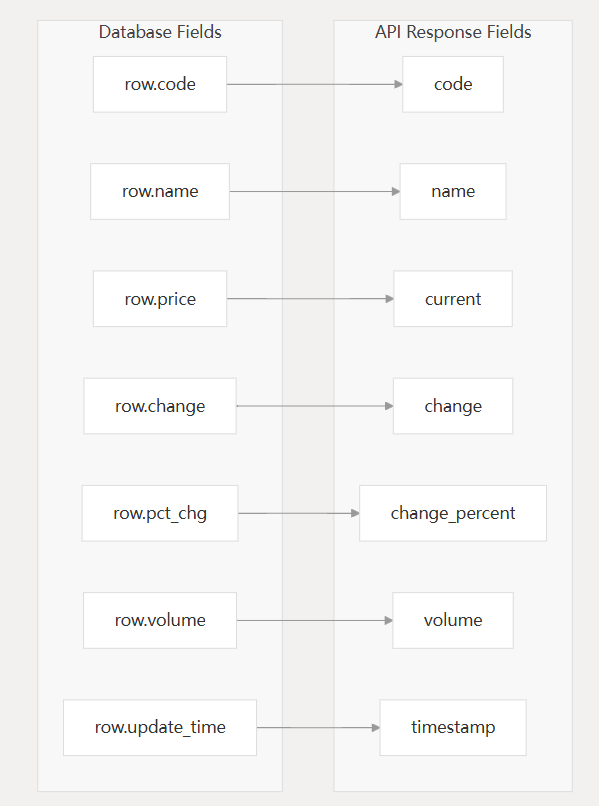
#### 支持的指数

API 检索以下主要市场指数的数据：



#### 数据转换

API 使用map\_index\_fields()以下函数将数据库字段映射到响应字段：



#### 前端集成

前端通过loadMarketIndices()更新索引显示卡的函数使用此 API：

资料来源：[前端/js/home.js50-74](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/home.js#L50-L74)

### 行业板块实时行情API

端点 /api/market/industry\_board提供实时行业板块表现数据，按涨跌幅变化百分比降序排列。

#### 端点详细信息

**方法：**GET  
**路径：**/api/market/industry\_board  
**功能：**get\_industry\_board()

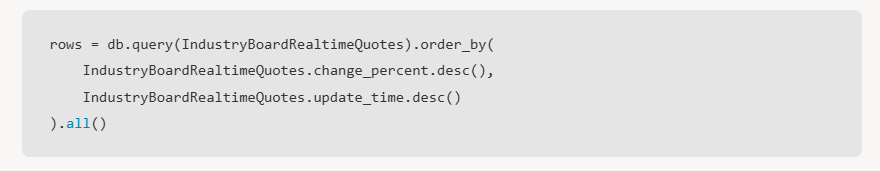
#### 响应数据结构

API 返回全面的板数据，包括：

* 行业板块标识（board\_code,board\_name）
* 价格指标（latest\_price,,）change\_amount,change\_percent
* 成交量指标（volume、amount、turnover\_rate）
* 市值（total\_market\_value）
* 领涨股票信息（leading\_stock\_name、leading\_stock\_code、leading\_stock\_change\_percent）

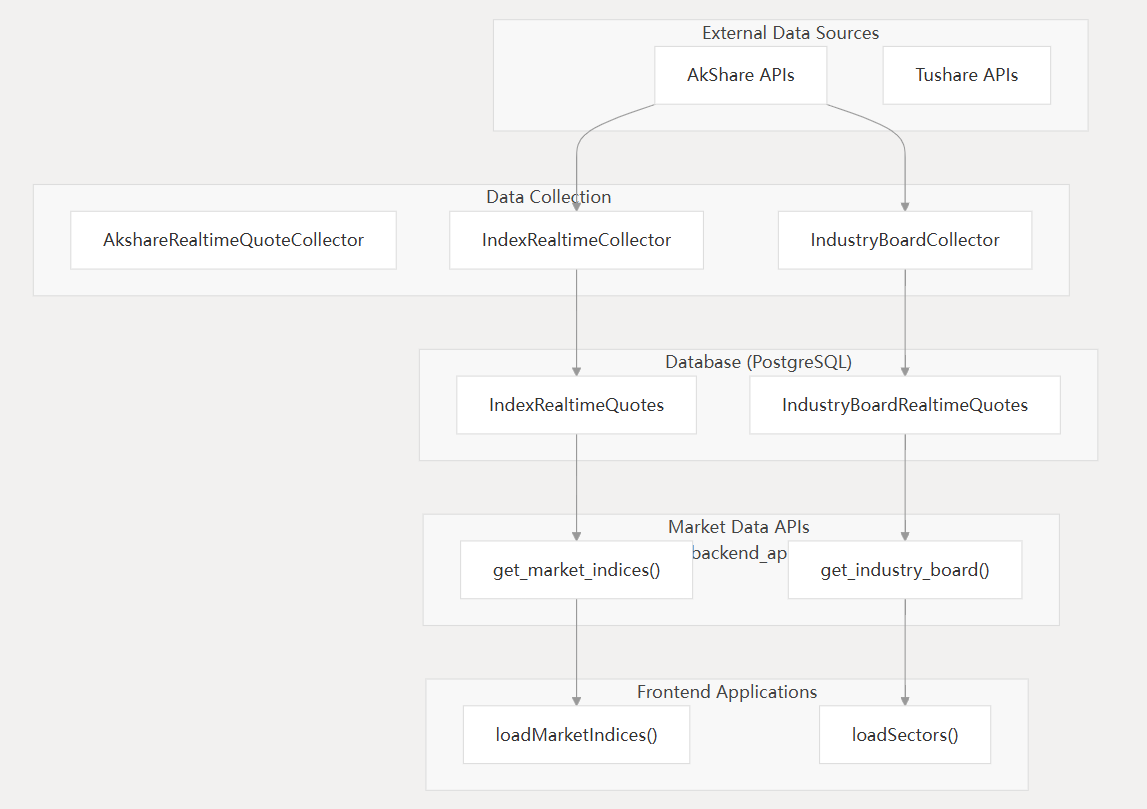
#### 数据库查询

端点按涨跌幅变化百分比排序查询IndustryBoardRealtimeQuotes表：



### 数据流架构

下图显示了市场数据如何从外部来源通过收集系统流向市场数据 API：

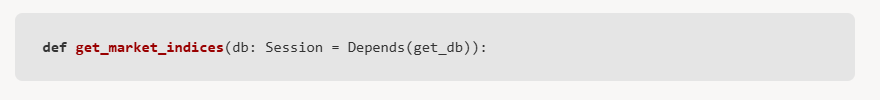


### 数据库集成

市场数据 API 通过 SQLAlchemy ORM 模型和依赖注入与 PostgreSQL 数据库集成。

#### 数据库会话管理

API 使用get\_db()依赖项来获取数据库会话：



#### ORM模型

API 与两个主要数据库模型交互：

* IndexRealtimeQuotes- 对于市场指数实时行情数据
* IndustryBoardRealtimeQuotes- 针对行业板块实时行情数据

#### 数据转换

该row\_to\_dict()实用函数将 SQLAlchemy 模型实例转换为适合 JSON 响应的字典格式，并对日期时间字段进行特殊处理。

资料来源：[backend\_api/market\_routes.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/market_routes.py#L12-L13)

### 实用函数

**安全数据转换**

该safe\_float()函数提供了对数字数据转换的强大处理：  


这可确保无效或缺失的数值不会导致 API 失败。

### 错误处理

市场数据 API 实现了全面的错误处理并提供了详细的错误响应：

**标准错误响应格式**  


**异常处理模式**

API 捕获异常并使用 HTTP 状态代码返回结构化的错误响应：



## 股票分析与新闻API

本文档涵盖股票分析系统提供的高级分析和新闻 API。这些 API 提供智能股票分析功能，包括技术指标、价格预测、交易建议、新闻聚合和研究报告。

股票基础数据及实时行情，请参见[**股票管理API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.2-stock-management-apis)。市场层面数据及指数，请参见[**市场数据API**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.4-market-data-apis)。

### 概述

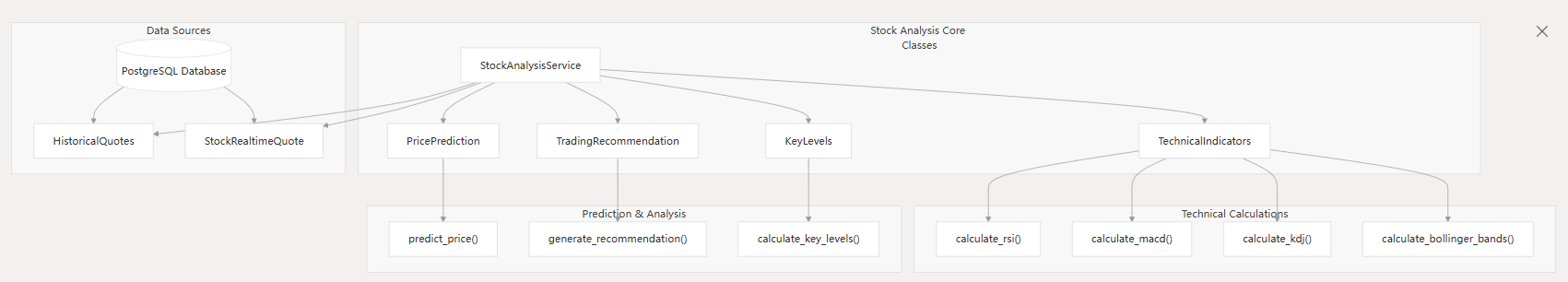
股票分析和新闻 API 提供两大类功能：

1. 股票分析 API - 技术分析、价格预测、交易建议和关键水平识别
2. 新闻与研究 API - 新闻聚合、研究报告收集和内容过滤

这些API支持股票详情界面显示的智能分析功能，为投资决策提供全面的信息。

### 股票分析服务架构

股票分析功能围绕模块化架构构建，具有针对不同类型分析的专门类：

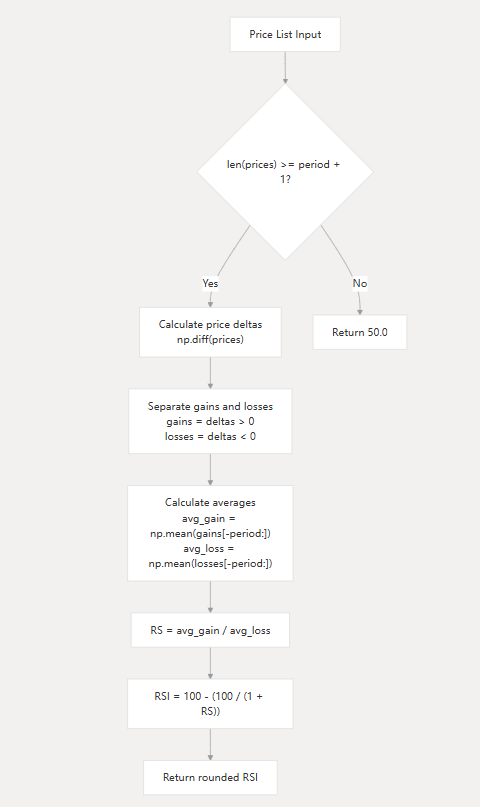


### 技术指标实现

该类TechnicalIndicators提供了计算常见技术分析指标的静态方法：



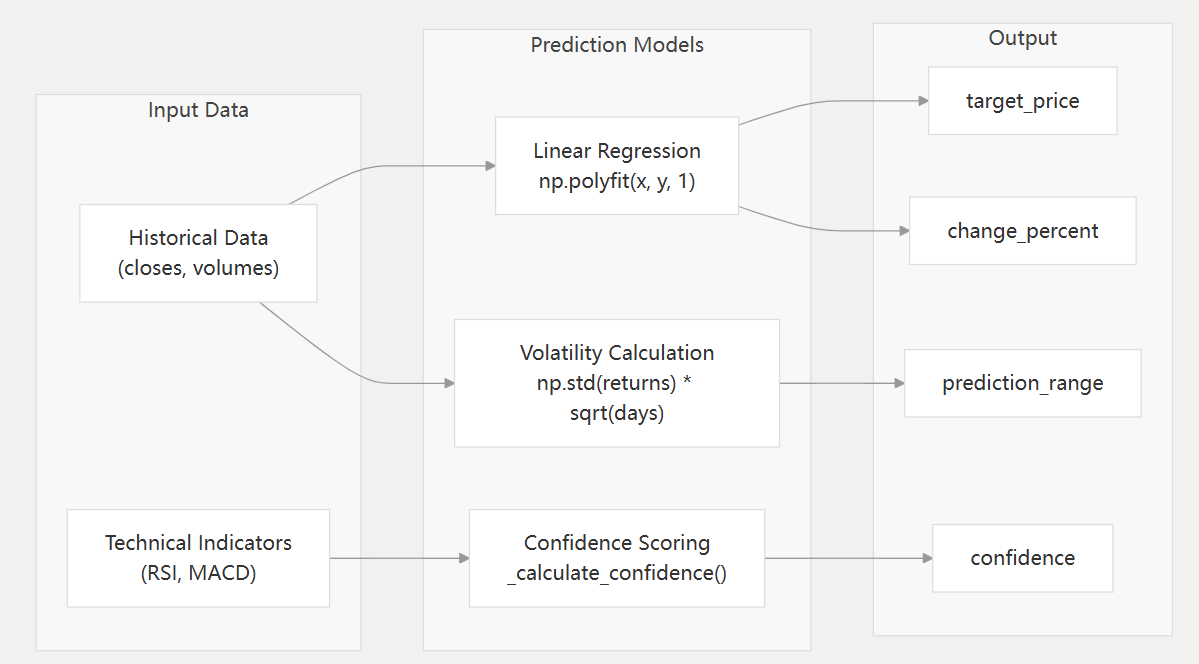
**RSI计算流程**



资料来源：[backend\_api/stock/stock\_analysis.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L16-L34)

### 价格预测系统

该类**PricePrediction**实现了多因素预测模型：



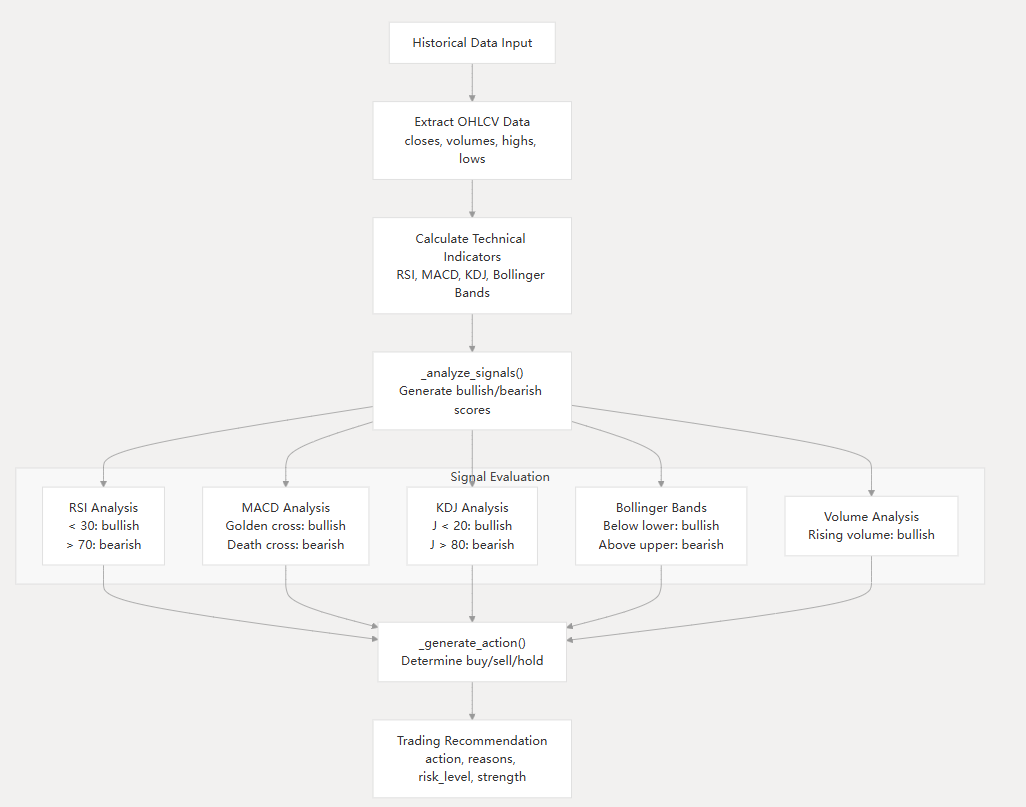
预测算法结合了：

* 使用**线性回归趋势分析np.polyfit(x, y, 1)**  [backend\_api/stock/stock\_analysis.py144](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L144-L144)
* 根据历史收益计算出的**基于波动率的预测范围**[backend\_api/stock/stock\_analysis.py152-158](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L152-L158)
* 基于 RSI、MACD 和趋势一致性的**信心评分**[backend\_api/stock/stock\_analysis.py161](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L161-L161)

### 交易引擎推荐

该**TradingRecommendation**类分析多种技术信号以生成交易建议：

**信号分析过程**



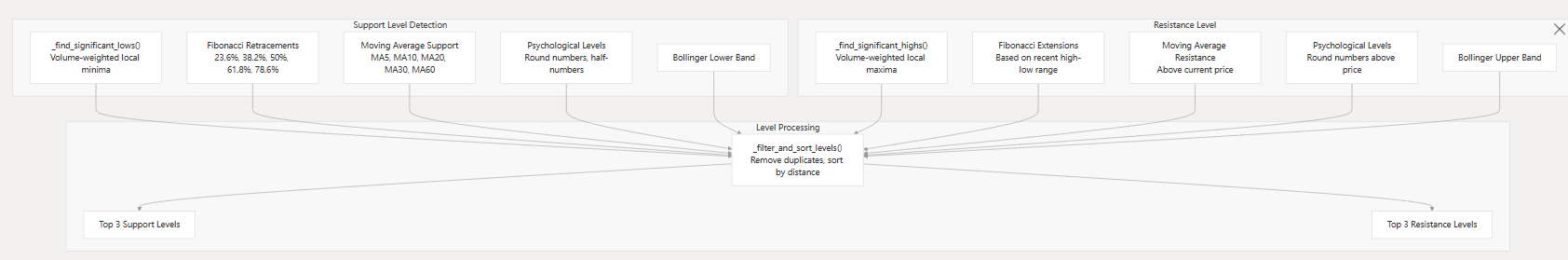
推荐系统评估来自多个指标的信号并生成：

* 行动：根据信号共识买入、卖出或持有[backend\_api/stock/stock\_analysis.py294-302](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L294-L302)
* 理由：支持技术因素列表[backend\_api/stock/stock\_analysis.py314](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L314-L314)
* 风险等级：根据信号强度，分为低、中、高三个等级[backend\_api/stock/stock\_analysis.py305-310](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L305-L310)
* 强度：数值置信度得分（0-100）[backend\_api/stock/stock\_analysis.py296-302](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L296-L302)

### 关键水平识别

该KeyLevels类使用多种方法识别支撑位和阻力位：

**支撑和阻力检测**



系统使用以下方法识别关键级别：

* 成交量确认的**技术极限**[backend\_api/stock/stock\_analysis.py419-464](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L419-L464)
* 标准比率的**斐波那契回撤**[backend\_api/stock/stock\_analysis.py467-491](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L467-L491)
* **移动平均线**作为动态支撑/阻力[backend\_api/stock/stock\_analysis.py494-529](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L494-L529)
* **心理价格水平**处于整数水平[backend\_api/stock/stock\_analysis.py532-563](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L532-L563)

### 新闻和研报API

该新闻系统通过专用 API 端点提供全面的新闻聚合和研究报告收集：

#### 新闻和研报API端点



#### 新闻处理管道



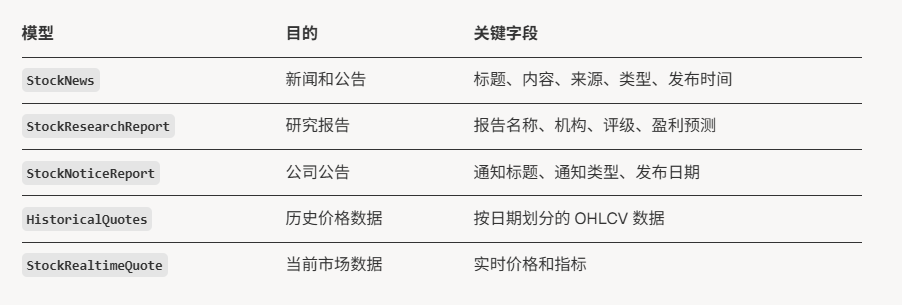
新闻处理系统实现：

* 使用股票名称/代码匹配进行**内容相关性过滤**[backend\_api/stock/stock\_news.py30-124](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_news.py#L30-L124)
* 使用相似度评分**进行重复检测**[backend\_api/stock/stock\_news.py126-175](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_news.py#L126-L175)
* 来自 AkShare、数据库和研究 API 的**多源聚合**[backend\_api/stock/stock\_news.py324-473](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_news.py#L324-L473)

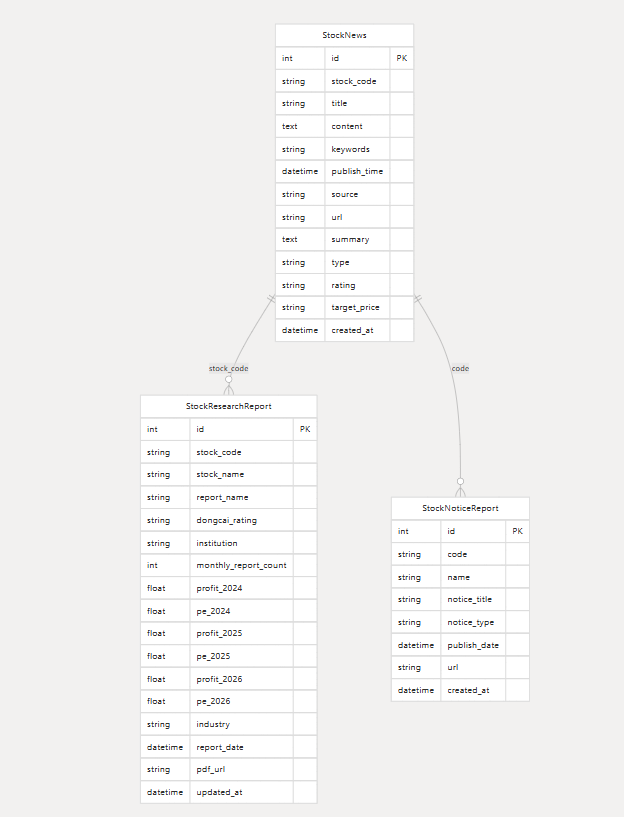
资料来源：[backend\_api/stock/stock\_news.py1-987](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_news.py#L1-L987)

### 数据库模型

系统使用几种数据库模型来存储分析和新闻数据：



**新闻数据模型**



资料来源：[backend\_api/models.py280-322](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/models.py#L280-L322)

### 前端集成

股票分析和新闻API通过股票详情界面与前端集成：

#### 股票分析显示

前端在“智能分析”标签页展示分析结果：

* 根据目标价格和信心**进行价格预测** [前端/stock.html132-146](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L132-L146)
* 带有行动和理由的**交易建议** [前端/stock.html148-163](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L148-L163)
* 带信号的**技术指标** [前端/stock.html165-189](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L165-L189)
* **关键**支撑位和阻力位 [前端/stock.html191-224](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L191-L224)

#### 新闻与研报展示

新闻内容显示在专用选项卡中：

* **新闻标签**，可按类型（全部、公告、新闻、研究）进行过滤 [前端/stock.html269-307](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L269-L307)
* 包含分析师报告和评级的**研报标签** [前端/stock.html309-344](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L309-L344)

前端使用 JavaScript 调用 API 获取数据并根据所选股票代码动态更新显示。

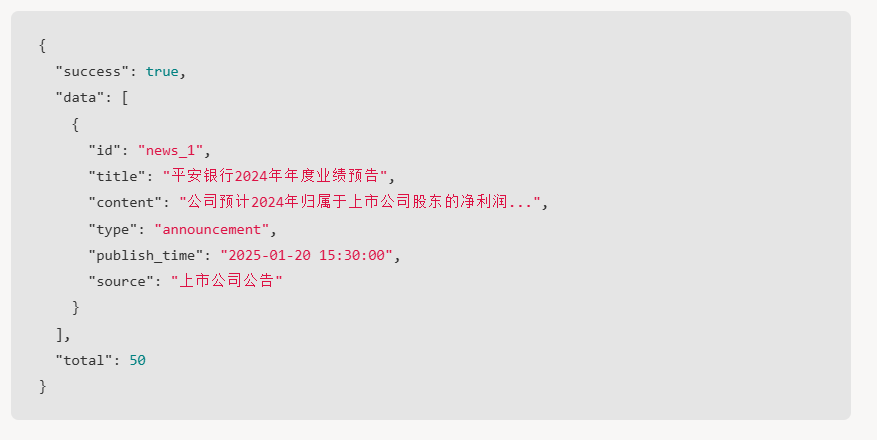
资料来源：[前端/stock.html116-376](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock.html#L116-L376)

### API响应格式

#### 分析API响应



#### 新闻API响应



资料来源：[backend\_api/stock/stock\_analysis.py669-679](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_analysis.py#L669-L679) [backend\_api/stock/stock\_news.py467-468](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_news.py#L467-L468)

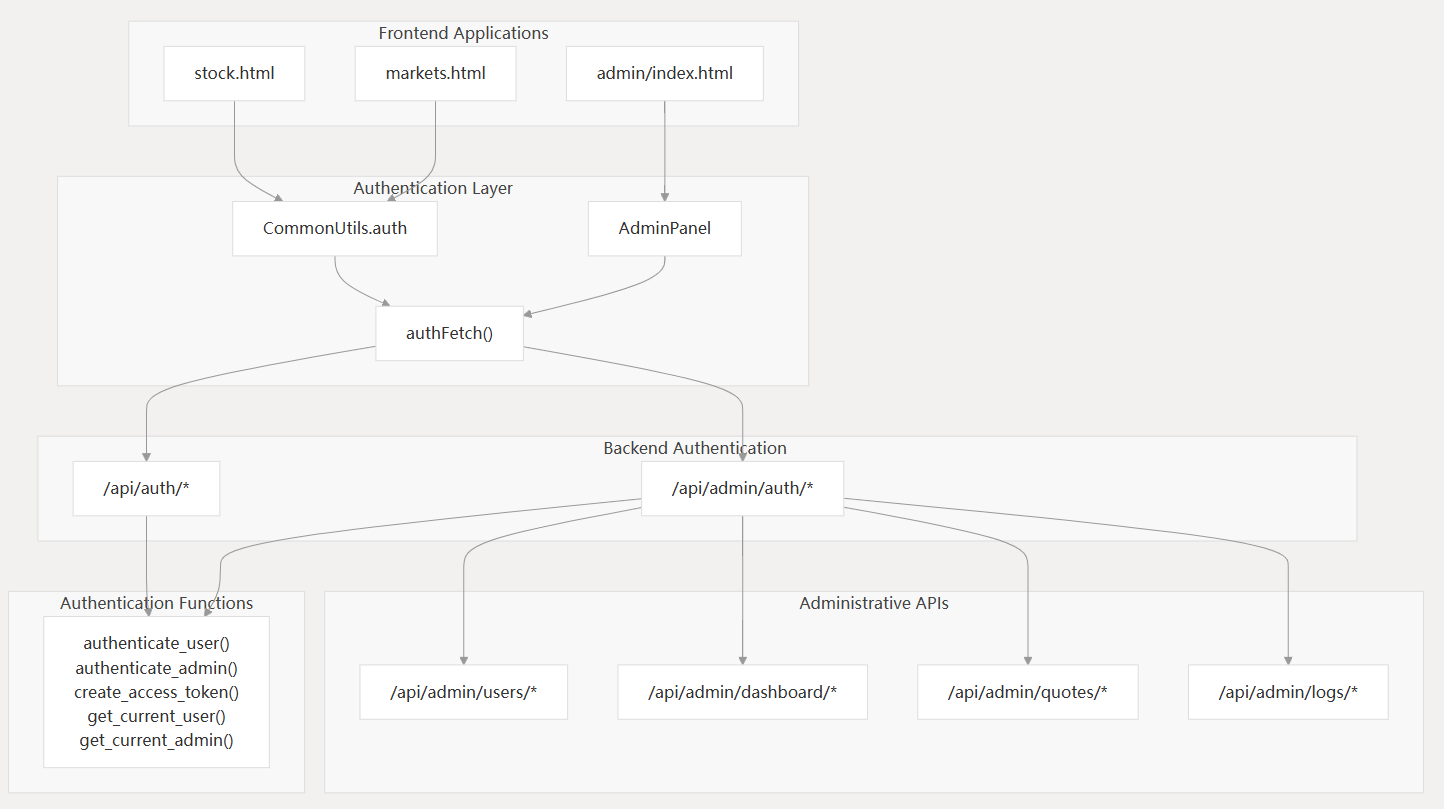
## 身份验证和管理API

本文档涵盖股票分析系统的身份验证机制和管理 API 端点。它涵盖了前端应用程序的用户身份验证、管理界面的管理员身份验证，以及用于系统管理的完整管理 API 集。

有关前端身份验证实用程序和用户界面集成，请参阅[**常用前端实用程序**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.6-common-frontend-utilities)。有关管理面板用户界面的详细信息，请参阅[**新版管理界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.4-legacy-admin-interface)。

### 身份验证架构概述

系统采用双重身份验证架构，为普通用户和管理员提供独立的身份验证流程。两个系统均使用基于 JWT（JSON Web Token）的身份验证，并具有不同的令牌范围和访问级别。

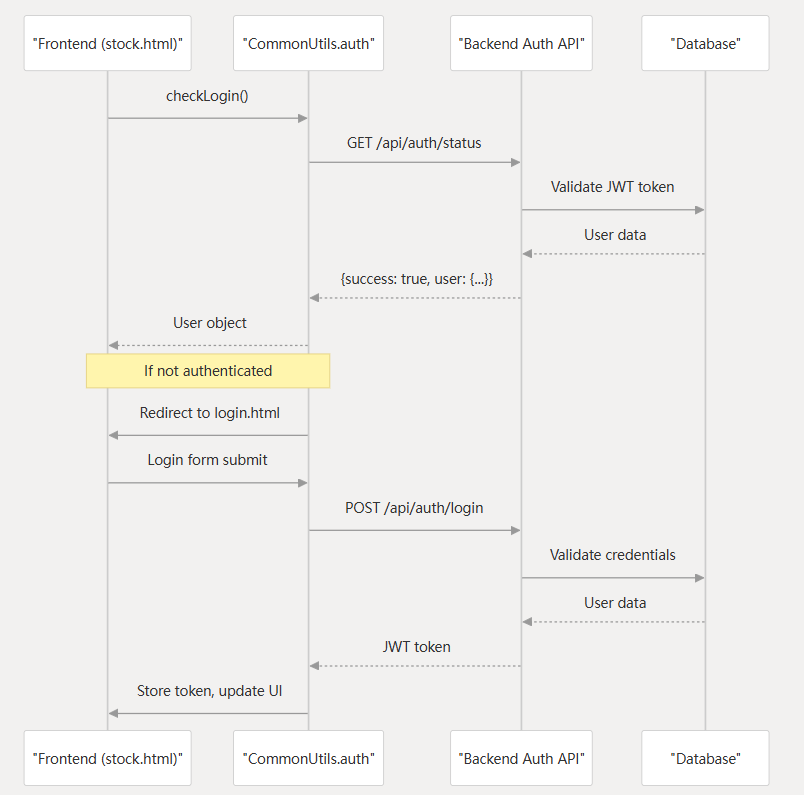


资料来源：[backend\_api/main.py84-100](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L84-L100) [前端/js/common.js4-13](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/common.js#L4-L13) [管理员/js/admin.js4-9](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/admin/js/admin.js#L4-L9) [backend\_api/admin/auth.py27](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L27-L27)

### 用户认证

用户认证系统为访问股票分析功能的前端应用程序提供基于JWT的认证。

#### 身份验证流程



资料来源：[前端/js/common.js20-44](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/common.js#L20-L44) [前端/js/common.js52-77](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/common.js#L52-L77)

#### 前端身份验证实用程序

该CommonUtils.auth模块为前端应用程序提供集中式身份验证管理：



该authFetch()函数自动在 API 请求中包含 JWT 令牌：



资料来源：[前端/js/common.js5-12](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/common.js#L5-L12) [前端/js/common.js16-136](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/common.js#L16-L136)

### 管理员身份验证

管理员身份验证系统为具有提升权限的管理功能提供单独的身份验证。

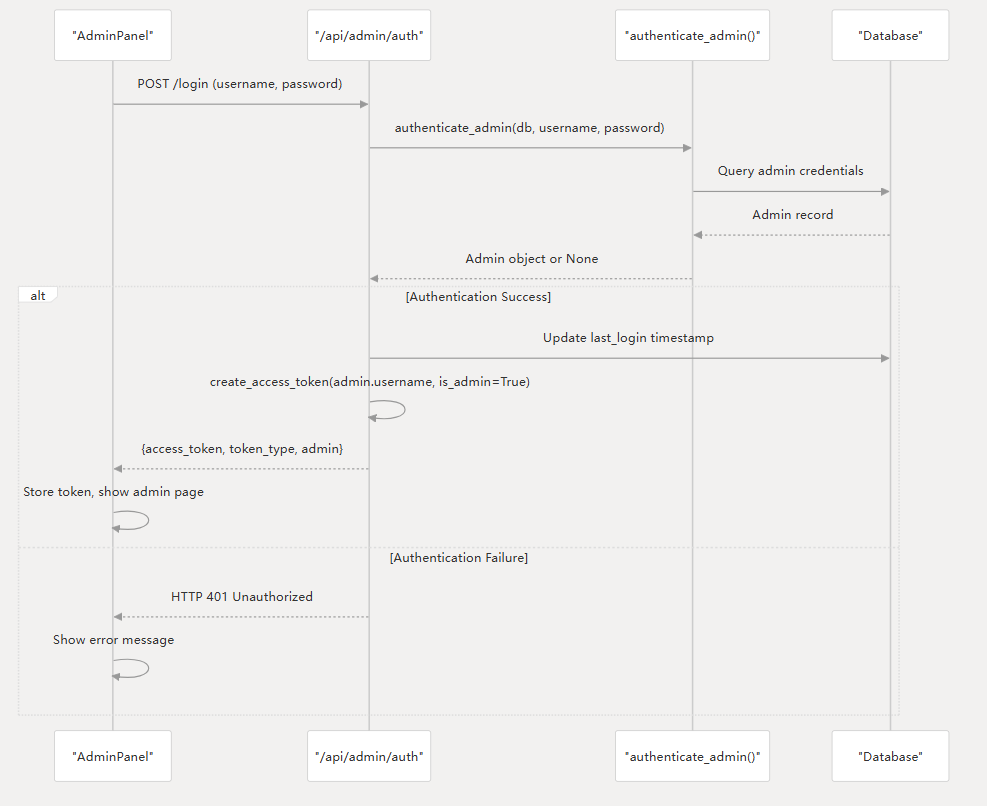
#### 管理员身份验证路由

backend\_api/admin/auth.py管理员身份验证系统通过以下端点实现：



资料来源：[backend\_api/admin/auth.py29-81](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L29-L81)

#### 管理员身份验证流程

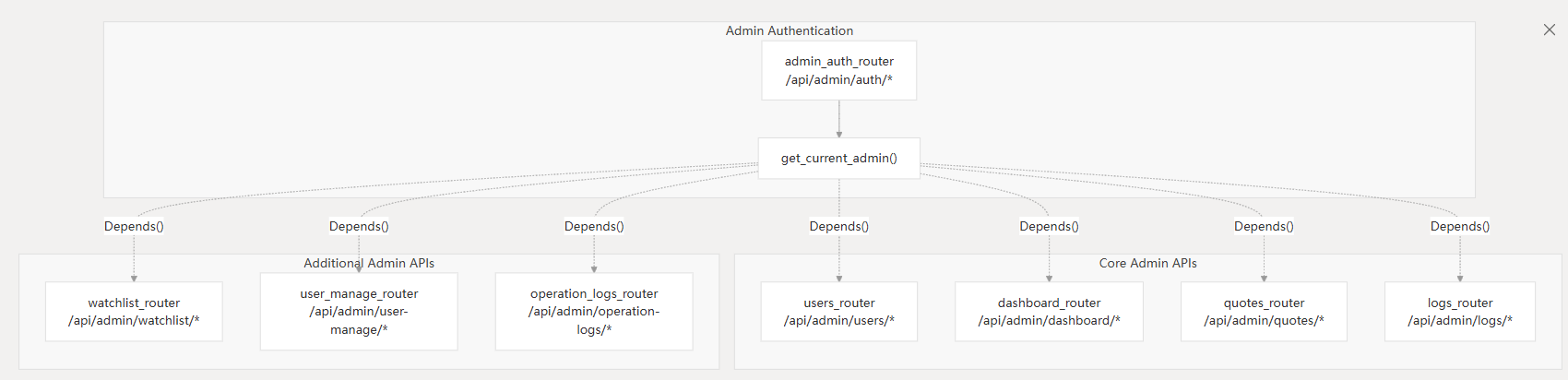


资料来源：[backend\_api/admin/auth.py29-59](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/auth.py#L29-L59) [管理员/js/admin.js78-119](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/admin/js/admin.js#L78-L119)

### 管理员管理API

系统提供了全面的管理API，用于管理股票分析系统的各个方面。

#### 管理API架构



资料来源：[backend\_api/main.py84-100](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/main.py#L84-L100)

#### 用户管理API

用户管理系统为用户账户提供了完整的CRUD操作：



资料来源：[backend\_api/admin/users.py27-166](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/users.py#L27-L166)

#### 仪表板API

仪表板 API 为管理界面提供统计数据和系统指标：



资料来源：[backend\_api/admin/dashboard.py45-187](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/dashboard.py#L45-L187)

#### 行情报价管理API

报价管理模块提供对市场数据的管理控制：



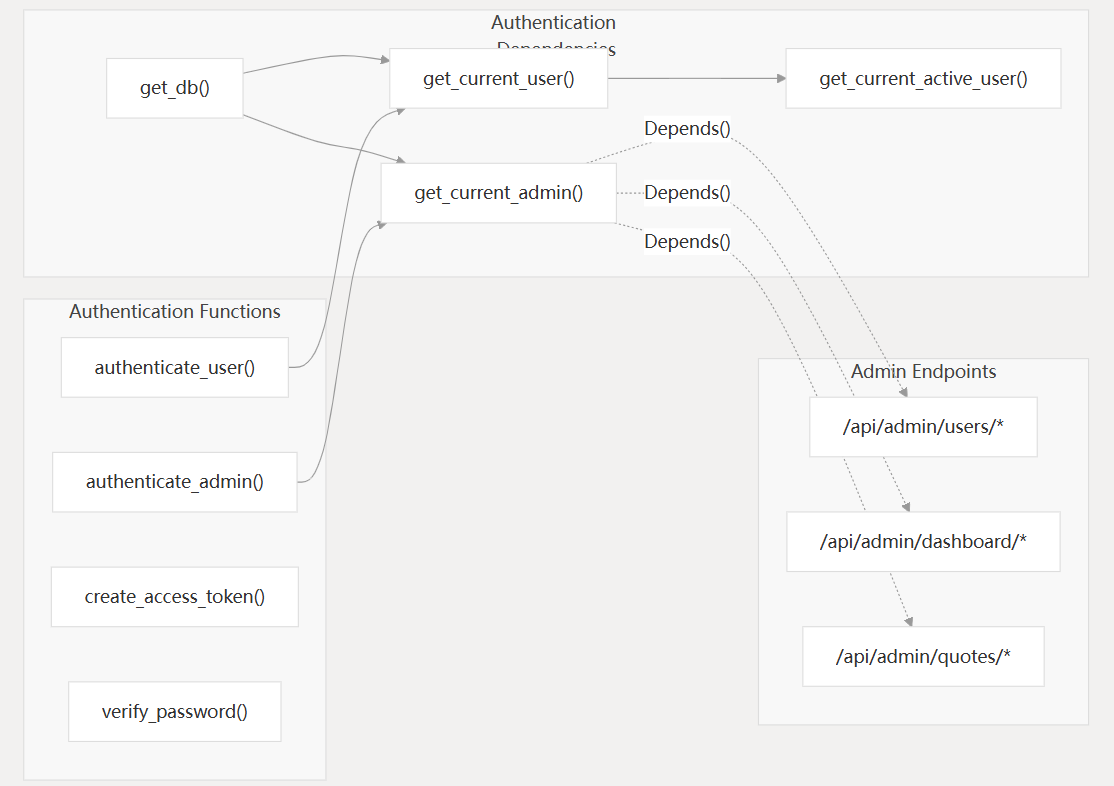
系统支持按股票代码、股票名称、日期范围（今日、周、月、自定义）和分页进行筛选。

资料来源：[backend\_api/admin/quotes.py48-144](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/quotes.py#L48-L144)

### 身份验证依赖项和安全性

#### 依赖注入模式

系统使用 FastAPI 的依赖注入来跨管理端点强制执行身份验证：



资料来源：[backend\_api/admin/\_\_init\_\_.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/__init__.py#L89-L108) [backend\_api/admin/users.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/users.py) [backend\_api/admin/dashboard.py](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/admin/dashboard.py)

#### 安全配置

系统实现了多项安全措施：



### 前后端集成

#### API通信模式

前端应用程序使用一致的模式进行经过身份验证的 API 调用：

1. 令牌存储：JWT 令牌存储在localStorage
2. 自动标题：authFetch()自动添加Authorization标题
3. 错误处理：身份验证失败时自动重定向到登录
4. 令牌刷新：状态检查和验证

#### 配置管理

。。。

# 前端应用程序

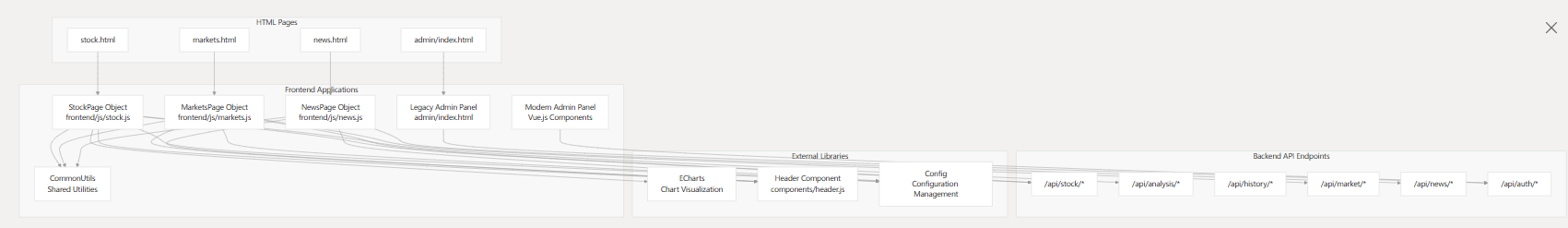
本文档介绍为股票分析系统提供用户界面的前端应用程序。前端由多个使用 HTML、CSS 和 JavaScript 构建的 Web 应用程序组成，这些应用程序与后端 API 交互，以提供实时股票分析、市场数据可视化、新闻聚合和管理功能。

前端应用程序被组织成不同的界面，以适应不同的用户工作流程：股票分析 **(stock.html)**、市场概览 (**markets.html**)、新闻和信息展示、历史数据管理以及管理面板。有关服务于这些前端的后端 API 的信息，请参阅[**后端 API 服务**](https://deepwiki.com/wangj9601/stock_quote_analayze/4-backend-api-services)。

## 概述

### 前端架构概述

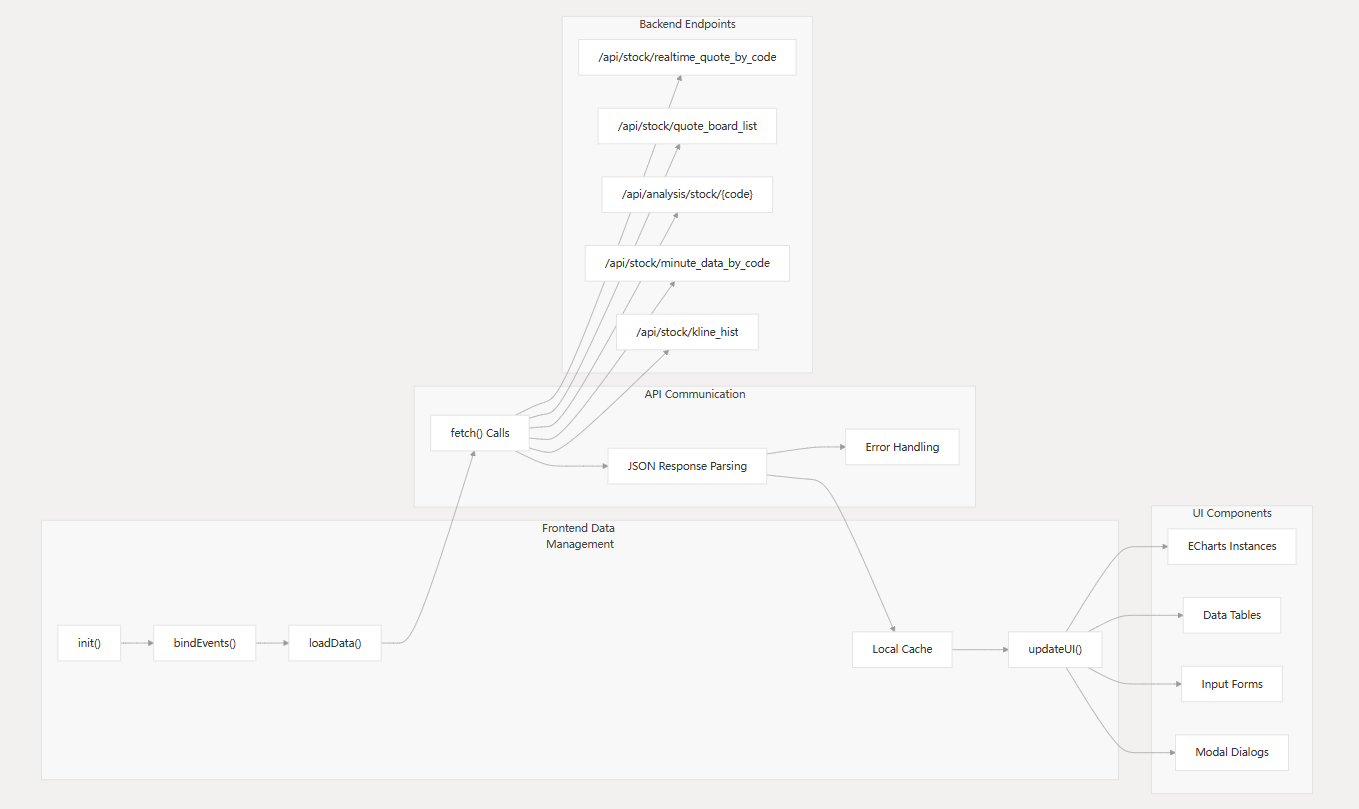
前端应用程序遵循模块化 JavaScript 架构，其中每个主要接口都实现为具有初始化、事件绑定和数据管理方法的单例对象。



资料来源：[前端/js/stock.js34-68](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L34-L68) [前端/js/markets.js24-46](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L24-L46) [前端/js/news.js3-14](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/news.js#L3-L14) [前端/markets.html12-13](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/markets.html#L12-L13)

### 数据流和API集成

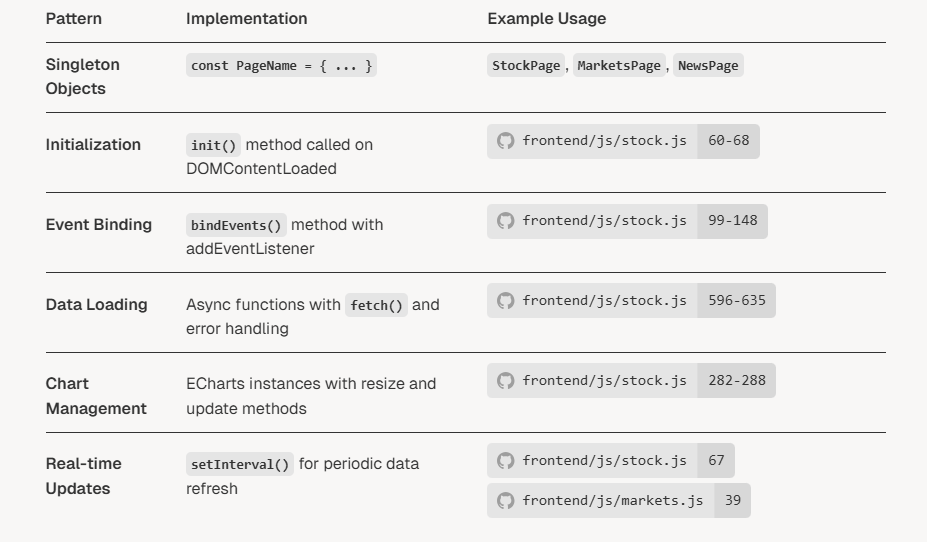
前端应用程序通过 RESTful API 与后端通信，每个主要接口管理自己的数据加载和状态管理。



资料来源：[前端/js/stock.js596-635](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L596-L635) [前端/js/markets.js214-243](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L214-L243) [backend\_api/stock/stock\_manage.py269-316](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L269-L316)

### 常见前端模式

所有前端应用程序都共享初始化、数据加载和 UI 管理的通用模式：



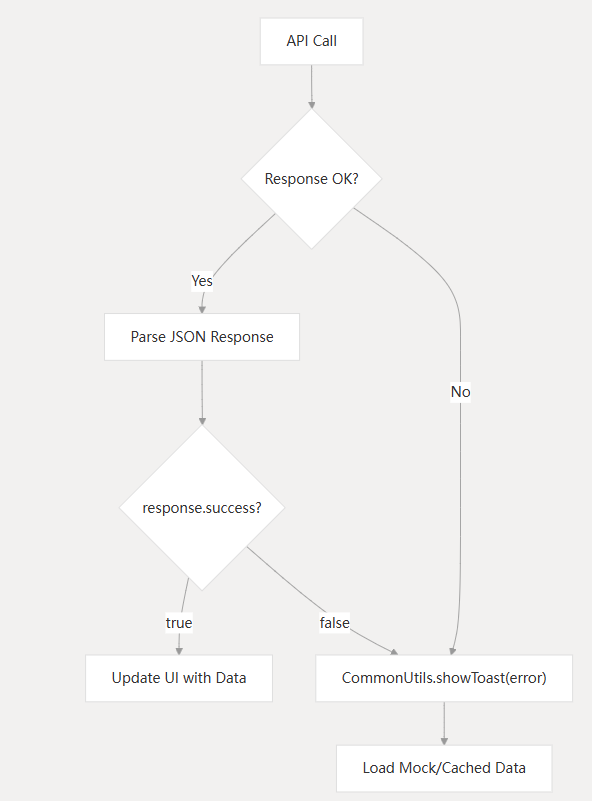
Config前端应用程序通过对象和常量使用一致的 API 基本 URL 配置API\_BASE\_URL：



资料来源：[前端/js/stock.js57](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L57-L57) [前端/js/markets.js](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L32-L32)  [前端/js/news.js7](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/news.js#L7-L7)

### 错误处理和用户反馈

前端应用程序通过方法实现一致的错误处理和用户反馈机制CommonUtils.showToast()，并在 API 不可用时进行优雅降级。



资料来源：[前端/js/stock.js627-634](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L627-L634) [前端/js/markets.js236-242](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L236-L242) [前端/js/stock.js785-794](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L785-L794)

## 股票详情界面

股票详情界面是主要的前端应用，用于展示股票的综合信息、实时报价、交互式图表、技术分析以及个股的财务数据。本文档涵盖了主页面**stock.html**及其相关的 JavaScript 控制器，该控制器为用户提供详细的股票分析功能，包括价格图表、技术指标、财务指标、新闻聚合和交易建议。

有关市场整体数据和排名的信息，请参阅[**市场与新闻界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.2-markets-and-news-interface)。有关历史数据管理和导出功能，请参阅[**历史数据界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.3-historical-data-interface)。

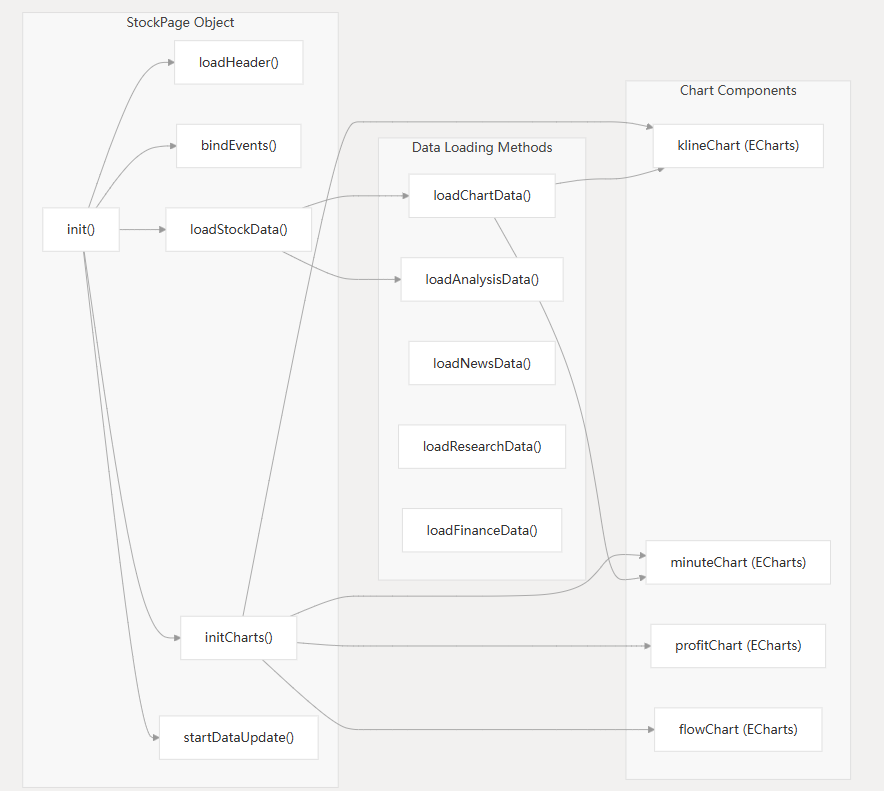
### 系统架构概述

股票详情界面遵循客户端 MVC 模式，以**StockPage**对象作为主控制器。该界面通过 REST API 调用集成多个数据源，并通过带有交互式图表的选项卡式面板呈现信息。

资料来源：[前端/js/stock.js34-68](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L34-L68) [backend\_api/stock/stock\_manage.py1-45](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L1-L45)

### 核心组件和数据流

该界面通过几个相互连接的组件进行操作，这些组件管理数据加载、用户交互和视觉呈现。

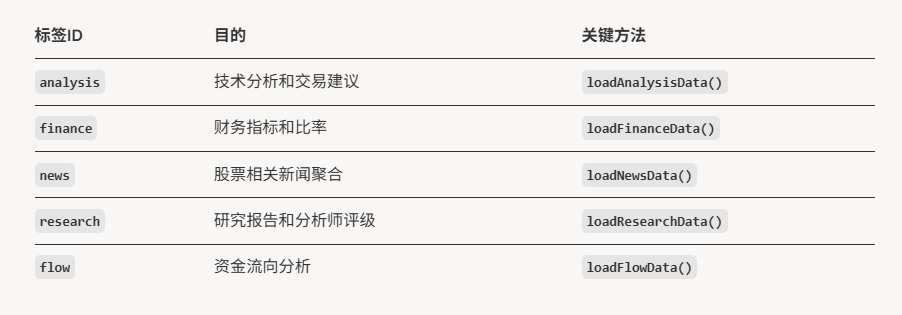


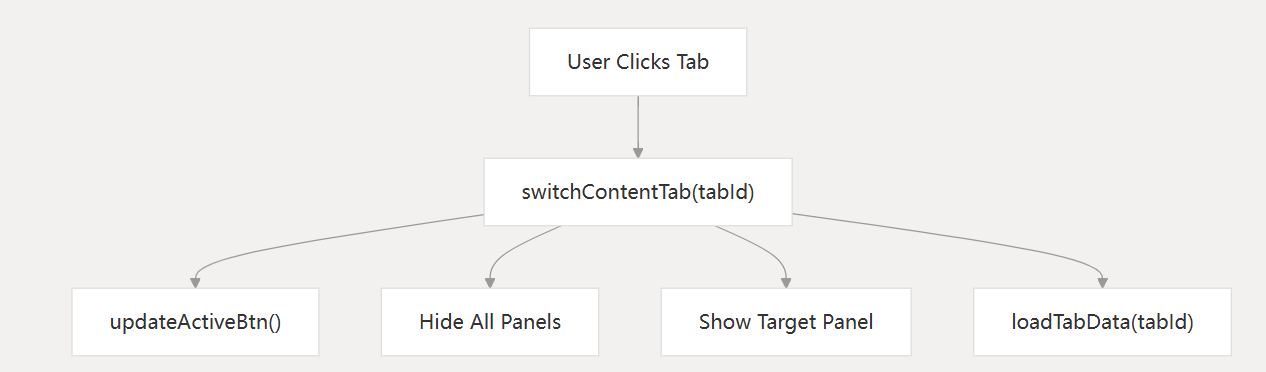
### 用户界面元素和交互

该界面提供了多个交互元素，这些元素以选项卡式面板的形式组织，并带有图表控件和实时数据更新。

#### 标签管理系统

该界面采用选项卡式布局，包含五个主要部分：





#### 图表类型选择和周期控制

用户可以通过交互式控件在不同的图表类型和时间段之间切换。

### API集成和数据管理

该接口集成了多个后端 API，用于获取实时和历史数据。主要使用的 API 端点如下：

**实时股票数据API**

资料来源：[前端/js/stock.js596-634](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L596-L634) [backend\_api/stock/stock\_manage.py269-316](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L269-L316) [backend\_api/stock/stock\_manage.py319-396](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L319-L396)

**智能分析集成**

该界面包括智能分析功能，可提供交易建议、技术指标和价格预测：

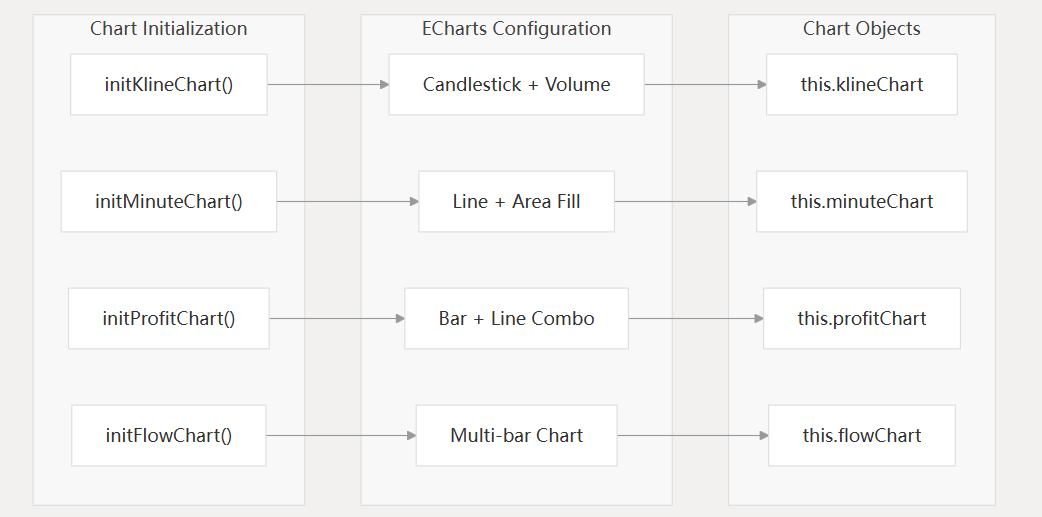
资料来源：[前端/js/stock.js742-795](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L742-L795) [前端/js/stock.js1030-1081](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L1030-L1081)

### 图表实现和可视化

该界面使用 ECharts 库实现跨多种图表类型的交互式数据可视化。

#### 图表初始化和配置

每种图表类型都有特定的初始化参数和数据格式：



#### 数据加载和图表更新

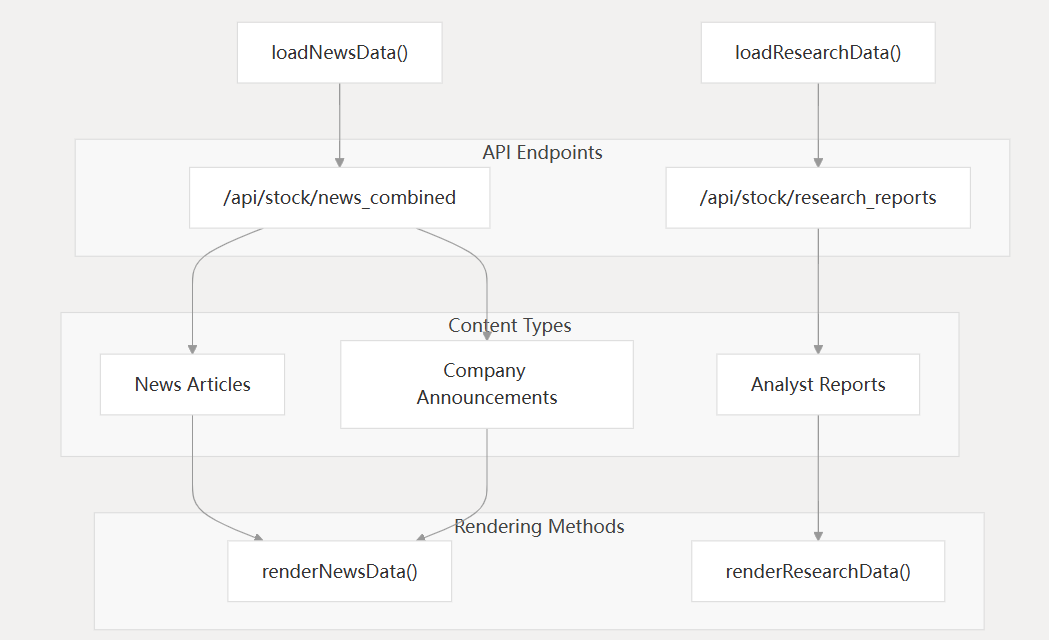
图表数据根据用户选择动态加载并自动刷新：



### 新闻与研报整合

该界面汇总了多种类型的金融内容，包括新闻文章、公司公告和研究报告。

**内容聚合**



### 错误处理和回退机制

该界面包括全面的错误处理、后备数据生成和用户反馈系统：

**分析数据回退**

当智能分析API发生故障时，系统会生成mock数据来维持接口功能。

**PDF下载处理**

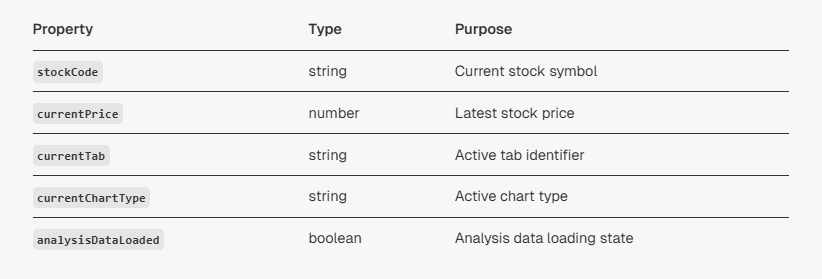
研究报告包括 PDF 下载功能以及针对跨域限制的多种后备策略。

### 技术实现细节

前端应用程序通过方法实现一致的错误处理和用户反馈机制CommonUtils.showToast()，并在 API 不可用时进行优雅降级。

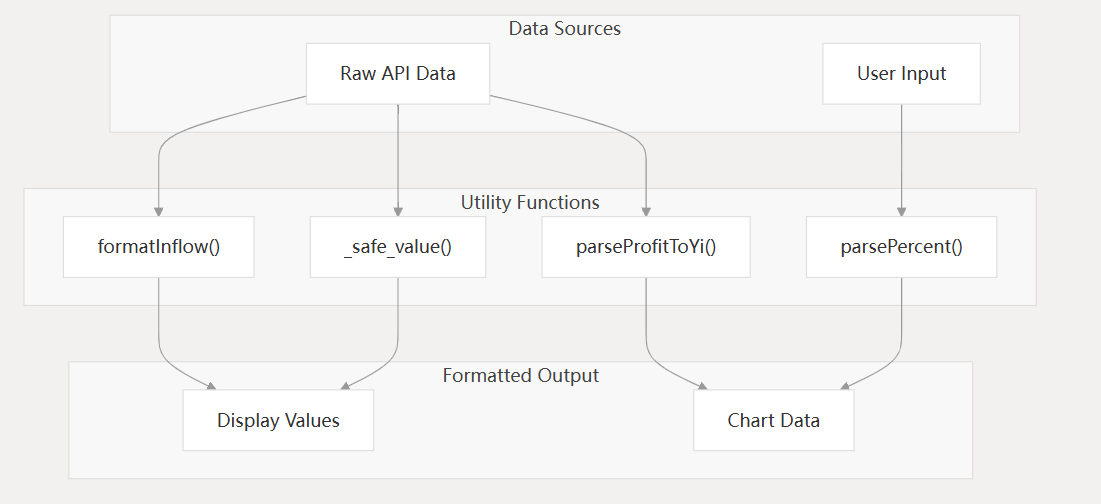
**状态管理**

该**StockPage**对象通过实例属性维护应用程序状态：



**数据格式和实用程序**

该接口包括用于数据格式化和验证的实用函数：



资料来源：[前端/js/stock.js1-26](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock.js#L1-L26) [backend\_api/stock/stock\_manage.py38-44](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/stock_manage.py#L38-L44)

股票详情界面提供了一个全面的单一股票分析环境，该环境通过由 API 端点支持的结构良好的 JavaScript 架构集成了实时数据、历史分析、新闻聚合和交互式可视化。

## 市场行情与新闻界面

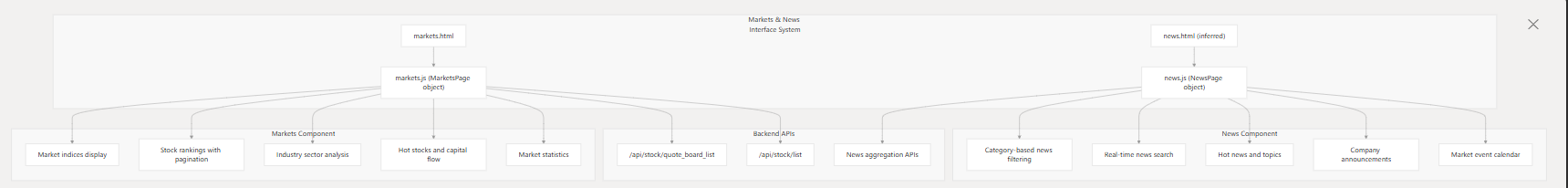
本文档介绍市场与新闻界面，该界面提供全面的仪表板，用于查看股票市场数据、财经新闻和市场分析。该界面由两个主要部分组成：用于实时股票数据和排名的市场概览，以及用于提供财经信息和市场洞察的新闻聚合系统。

个股分析功能请参见[**股票详情界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.1-stock-details-interface)，历史数据管理请参见[**历史数据界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/5.3-historical-data-interface)。

### 目的和架构

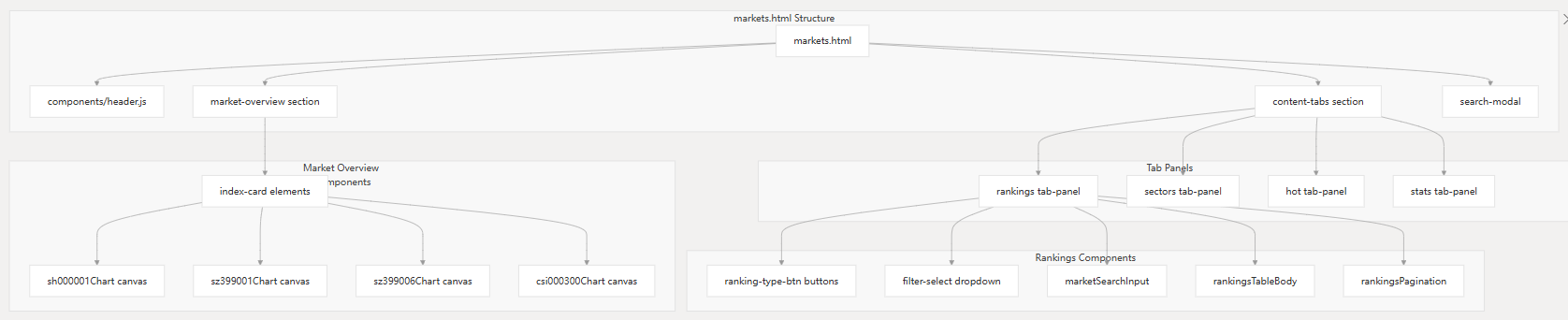
市场与新闻界面由两个互补的单页应用程序组成，提供全面的市场信息。市场组件将实时市场数据汇总到有序的标签页中，而新闻组件则通过分类的内容提要提供金融新闻、分析和市场洞察。

#### 总体接口架构



资料来源：[前端/markets.html1-454](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/markets.html#L1-L454) [前端/js/markets.js1-622](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L1-L622) [前端/js/news.js](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/news.js#L1-L446)

#### 组件结构



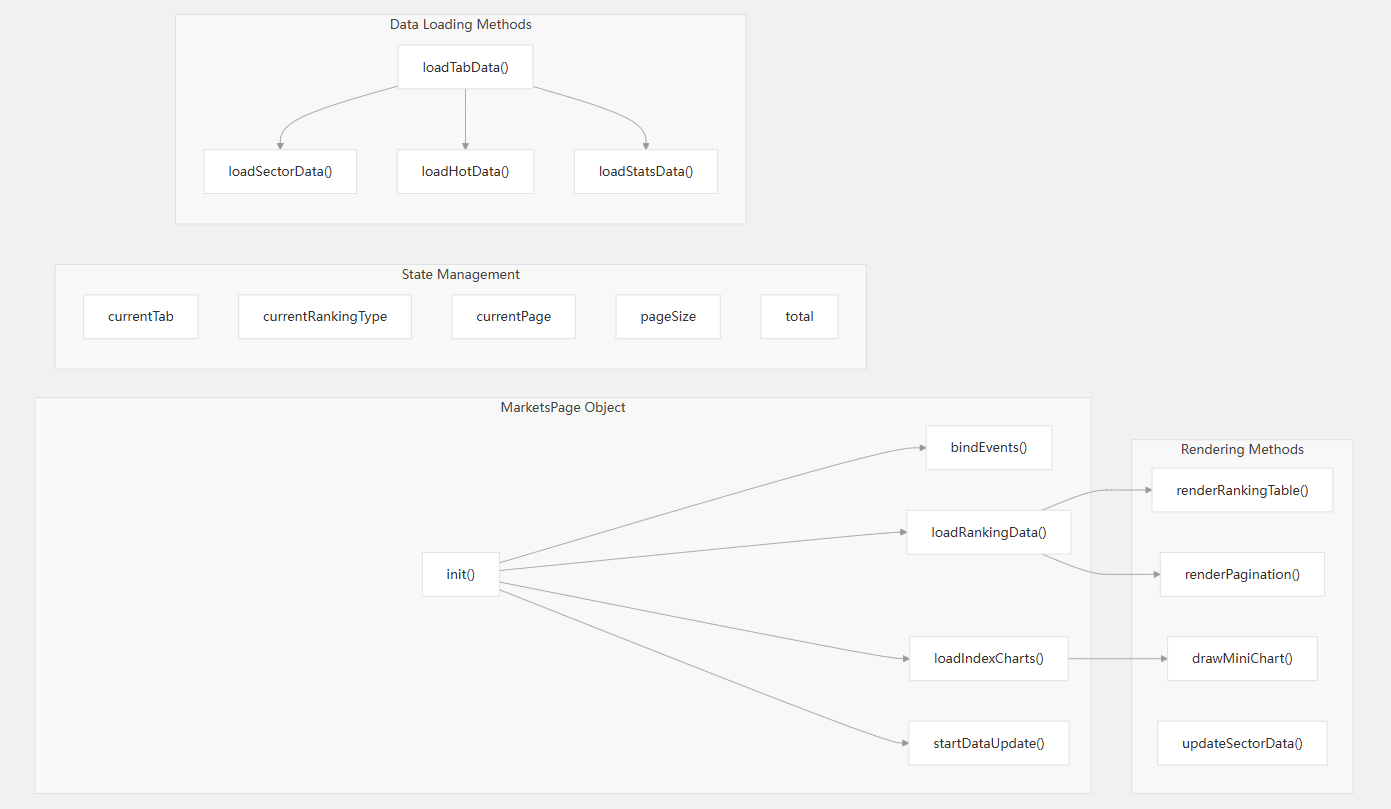
资料来源：[前端/markets.html1-454](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/markets.html#L1-L454) [前端/js/markets.js1-622](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L1-L622)

### 市场行情接口组件

市场行情界面通过有组织的基于标签的布局提供实时股票市场数据，并具有全面的排名和分析功能。

**JavaScript架构**

该**MarketsPage**对象集中所有市场功能并通过其属性和方法维护应用程序状态。



### 市场行情指数展示

顶部区域显示四大主要市场指数的实时数据和迷你图表。每张指数卡均显示当前值、变化量、变化百分比以及趋势可视化。

**索引卡结构**



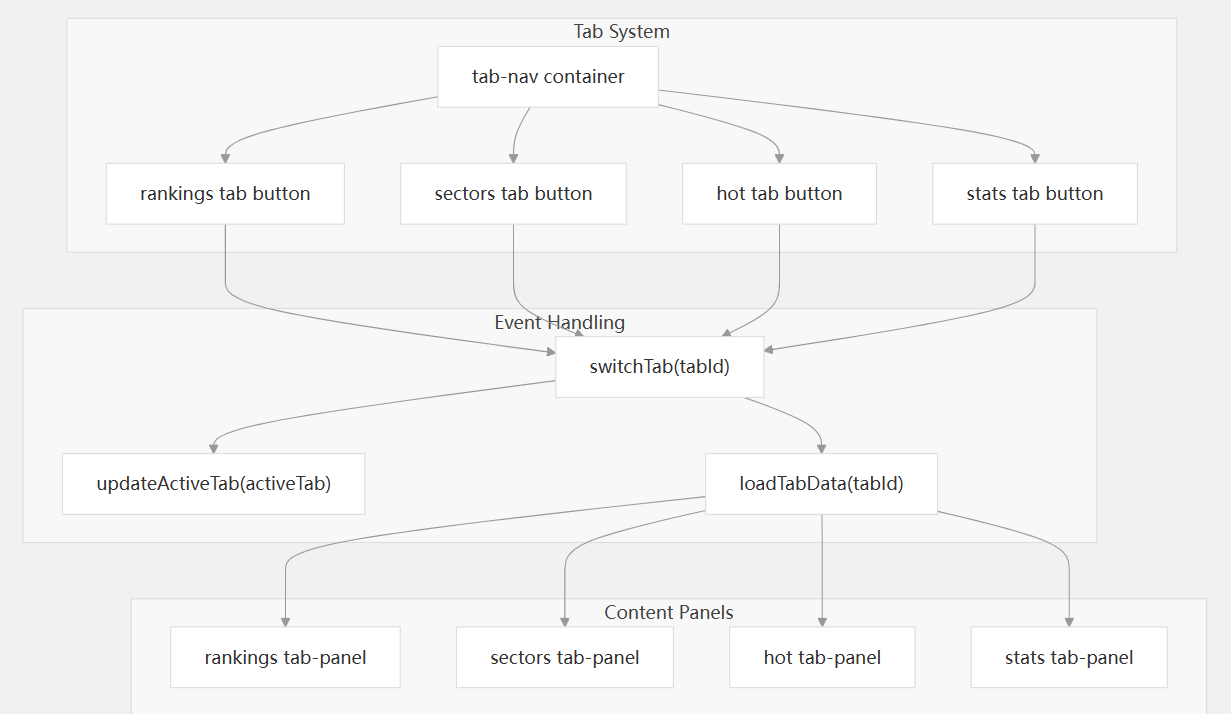
**图表渲染**

该drawMiniChart()方法使用 HTML5 Canvas 生成趋势可视化。

### 标签导航区域

该界面将内容组织成四个主要选项卡，每个选项卡显示市场数据的不同方面。

**选项卡配置**

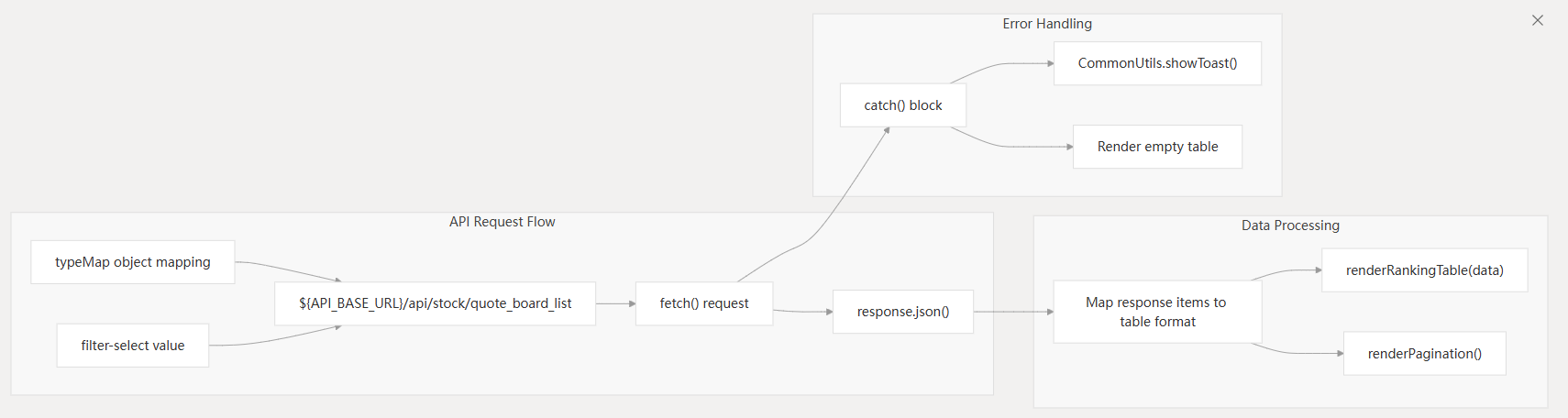


### 排名标签实现

排名选项卡提供具有不同排名标准和市场过滤选项的可排序股票列表。

**API集成**

该loadRankingData()方法从支持分页的后端 API 获取数据：



**排名类型**

该loadRankingData()方法从支持分页的后端 API 获取数据：

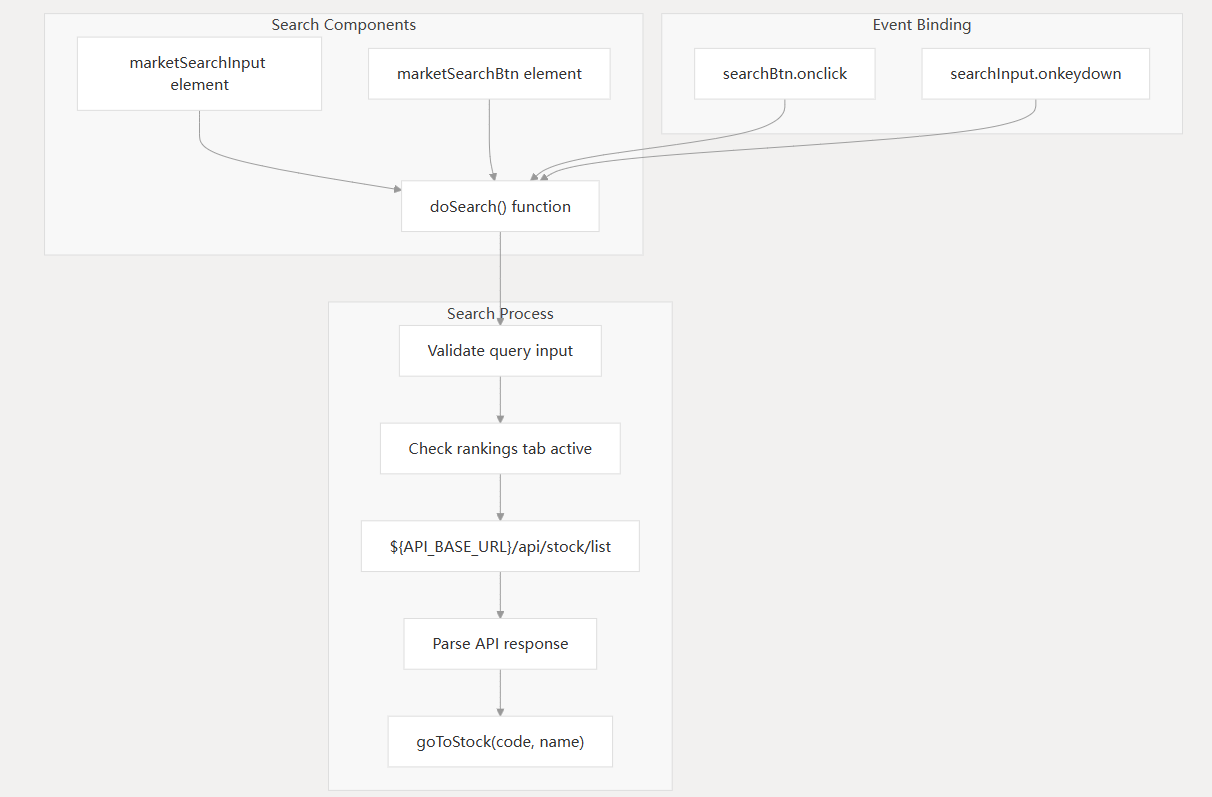


资料来源：[前端/js/markets.js201-243](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L201-L243) [前端/markets.html105-148](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/markets.html#L105-L148)

### 搜索和导航

市场行情页面包括搜索功能，允许用户快速查找并导航到特定股票。

**搜索实现**



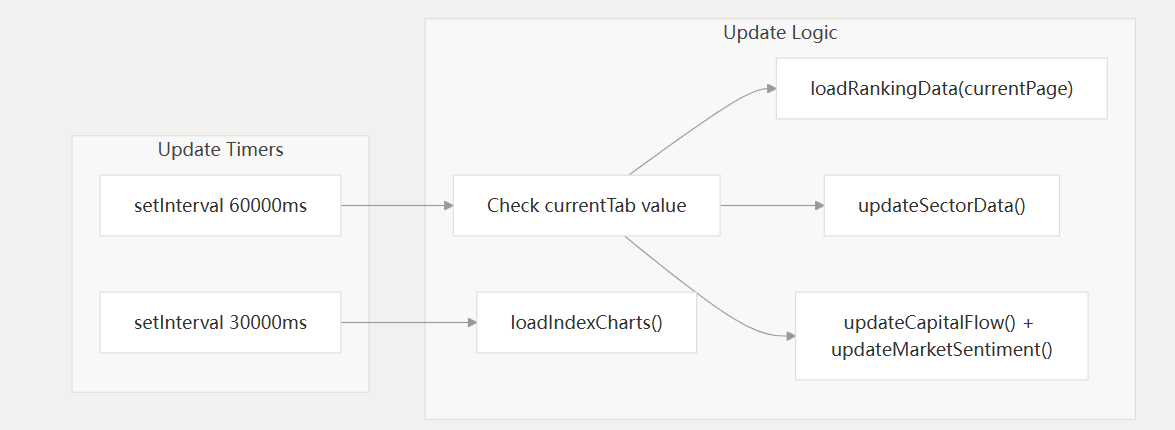
### 实时数据更新

系统实现数据自动刷新，保证市场信息保持最新。

**更新计划**

该**startDataUpdate()**方法建立了两个更新间隔：





### 数据格式化和显示

该系统包括用于财务数据呈现的综合格式化实用程序。

**格式化方法**



资料来源：[前端/js/markets.js436-461](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/markets.js#L436-L461)

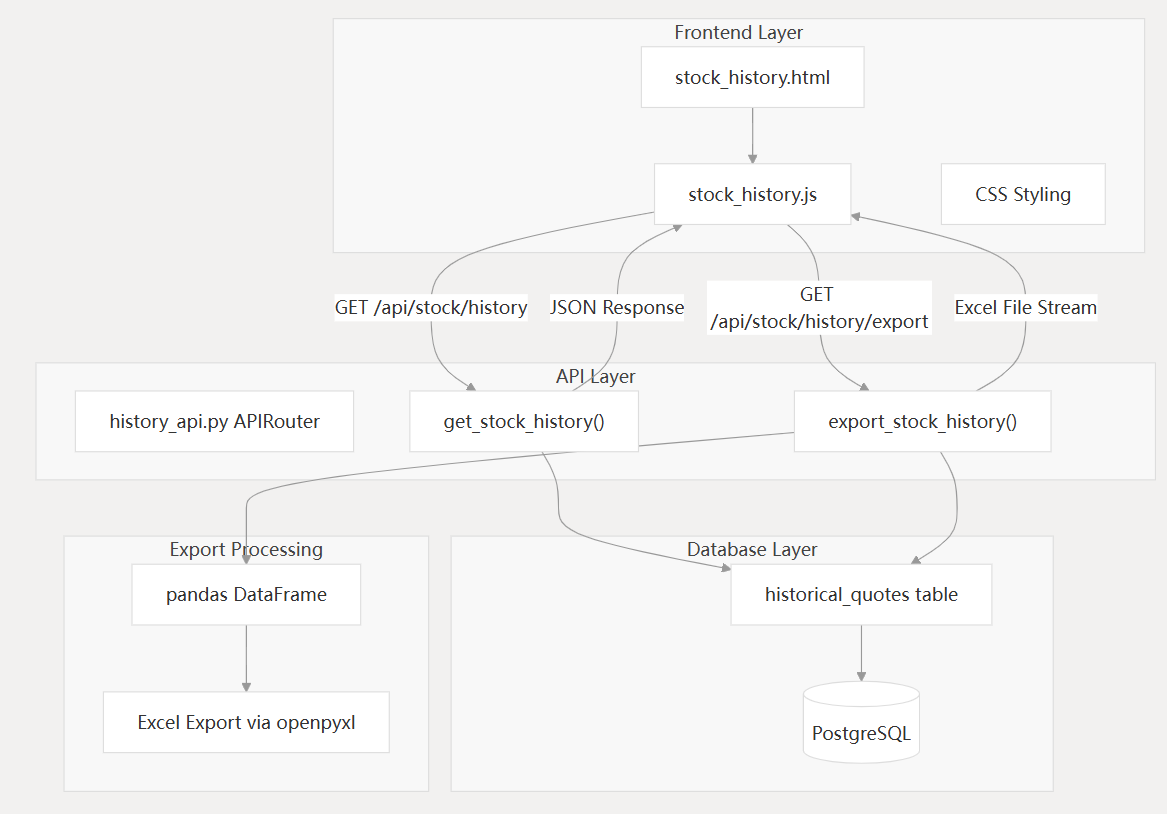
## 历史数据查询导出界面

历史数据界面提供了一个基于 Web 的全面系统，用于查看、筛选和导出历史股价数据。该系统由一个 FastAPI 后端（提供来自 PostgreSQL 数据库的分页历史数据）和一个响应式前端界面（以表格形式显示数据，并具有交互式筛选和导出功能）组成。

股票实时数据显示请参**股票详情界面**，通用行情数据界面请参见**市场**[**行情界面**](https://deepwiki.com/wangj9601/stock_quote_analayze/4.4-market-data-apis)。

### 系统架构

历史数据接口采用三层架构，包括 FastAPI 后端、PostgreSQL 数据库和通过 REST API 进行通信的 JavaScript 前端。



资料来源：[backend\_api/stock/history\_api.py1-196](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L1-L196) [前端/stock\_history.html1-169](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock_history.html#L1-L169) [前端/js/stock\_history.js1-164](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock_history.js#L1-L164)

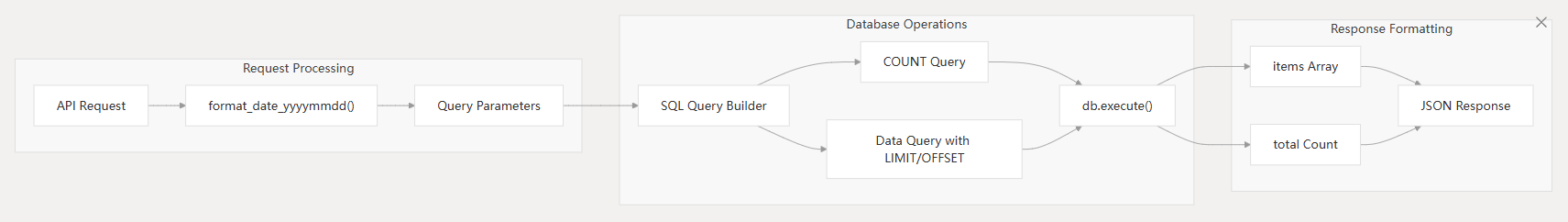
### 后端API实现

后端通过history\_api.py路由器提供两个主要端点用于历史数据访问。

**核心API端点**



**数据检索过程**



该函数使用针对表get\_stock\_history()的参数化 SQL 查询实现具有日期范围过滤的服务器端分页historical\_quotes。[backend\_api/stock/history\_api.py36-81](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L36-L81)

**Excel导出处理**

导出功能使用 **pandas** 和 **openpyxl** 生成具有以下数据转换的格式化 Excel 文件：

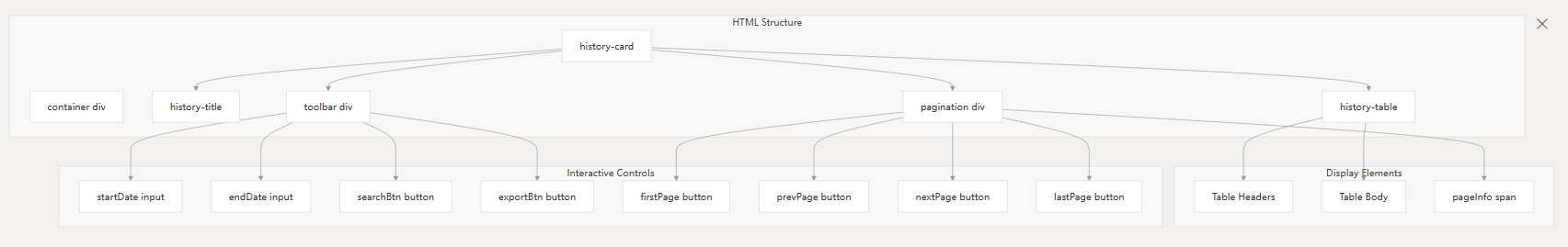
* 体积格式化：将原始体积转换为万（10,000）个单位
* 金额格式：将原始金额转换为亿（100M）单位
* 百分比格式：添加 % 符号，保留 2 位小数
* 颜色编码：红色字体表示正向变化，绿色字体表示负向变化

资料来源：[backend\_api/stock/history\_api.py 83-196](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L83-L196)

### 前端界面实现

前端由一个嵌入 CSS 的 HTML 页面和一个单独的 JavaScript 模块组成，用于处理所有用户交互和 API 通信。

**用户界面组件**



资料来源：[前端/stock\_history.html](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/stock_history.html#L136-L162)

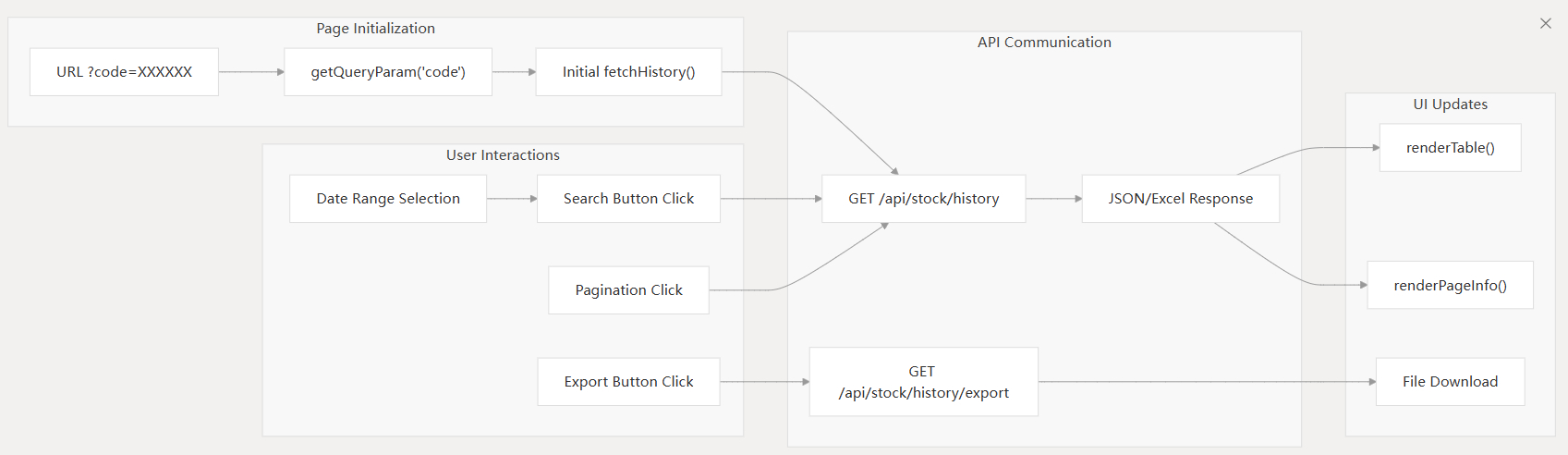
**JavaScript 功能**

该stock\_history.js模块通过几个关键功能管理状态和 API 交互：



系统使用 URL 参数检索股票代码，getQueryParam('code')并通过全局变量维护分页状态。

**数据流和状态管理**



### 数据格式化和显示

该界面在后端导出和前端显示中应用一致的财务数据格式：

**前端数据格式化**

* 体积显示：原始值通过以下方式转换为万个单位formatVolume()
* 金额显示：数值≥1亿显示为亿，≥1万显示为万，其他显示原始值formatAmount()
* 变化指标：正向变化使用cell-up类别（红色），负向变化使用cell-down类别（绿色）
* 百分比值：以 % 符号和 2 位小数显示

**后端导出格式**

Excel 导出通过 pandas 和 openpyxl 应用了额外的格式：

* 列标题采用中文，字体粗体
* 根据内容长度自动调整列宽
* 价格变动列的条件字体颜色
* 一致的数字格式和适当的舍入

资料来源：[前端/js/stock\_history.js26-44](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/frontend/js/stock_history.js#L26-L44) [backend\_api/stock/history\_api.py105-187](https://github.com/wangj9601/stock_quote_analayze/blob/681f78eb/backend_api/stock/history_api.py#L105-L187)

## 新版管理界面

### 目的和范围

### 架构概述

### 技术堆栈和配置

### 视图层架构

### 仪表板界面

### 用户管理界面

### 使用 Pinia 进行状态管理

### API服务层

### 日志界面

## 常见前端实用程序

# 数据库和存储

## 数据库架构

### 数据库技术栈

### Sqlalchemy模型架构

## 核心数据模型

## 表关系和约束

## 常见存储模式

# 配置和部署

## 系统配置管理

### 核心配置结构

### 数据库配置

### 身份验证配置

### 数据采集配置

## 开发环境设置

### 依赖管理和启动过程

### 开发服务器配置

### 日志配置

## 部署架构

### 生产环境结构

### 服务启动和流程管理

### 环境变量管理

### 安全配置