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

审阅

姓名	时间	职位

目录

1 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，请参阅系统架构了解整体设计模式、数据收集系统了解自动数据收集、后端 API 服务了解 REST 服务实现、前端应用程序了解用户界面以及数据库和存储了解数据持久性。

1.1. 目的

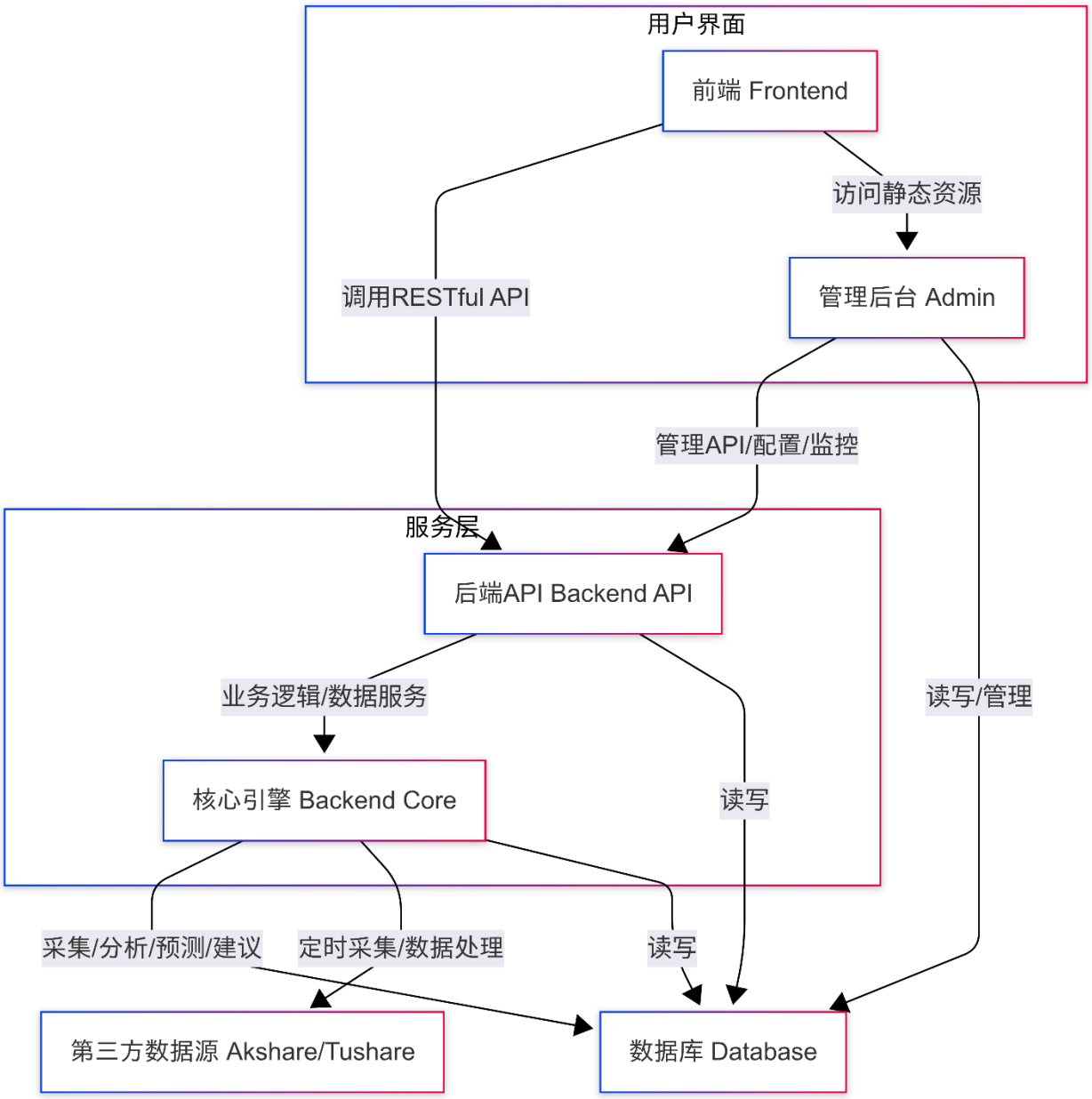
股票分析系统是针对中国股市数据分析的综合平台，提供：

- **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

1.2. 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。。

1.3. 系统组件图

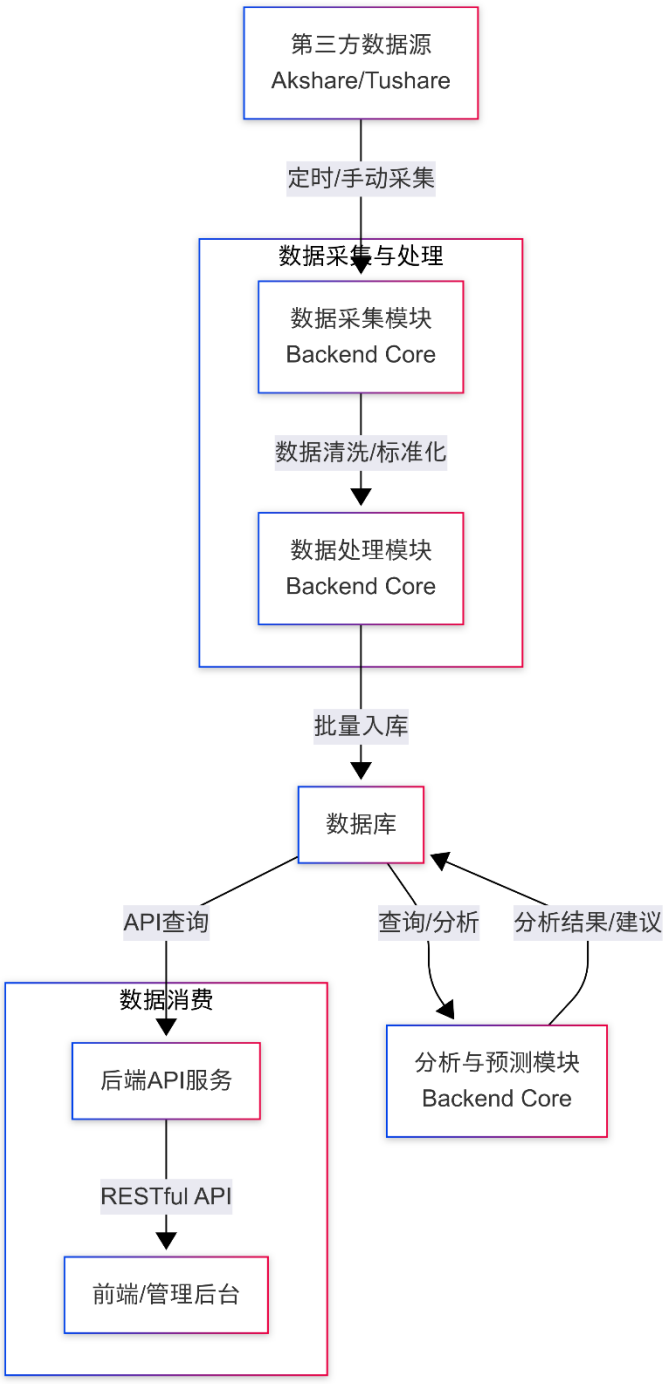


股票分析系统的主要组件及其关系如下：

- 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。

1.4. 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。

1.5. 关键组件

1.5.1. 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

1.5.2. 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

1.5.3. 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

1.6. 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…`

1.7. 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

1.8. 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

2 系统架构

2.1. 目的和范围

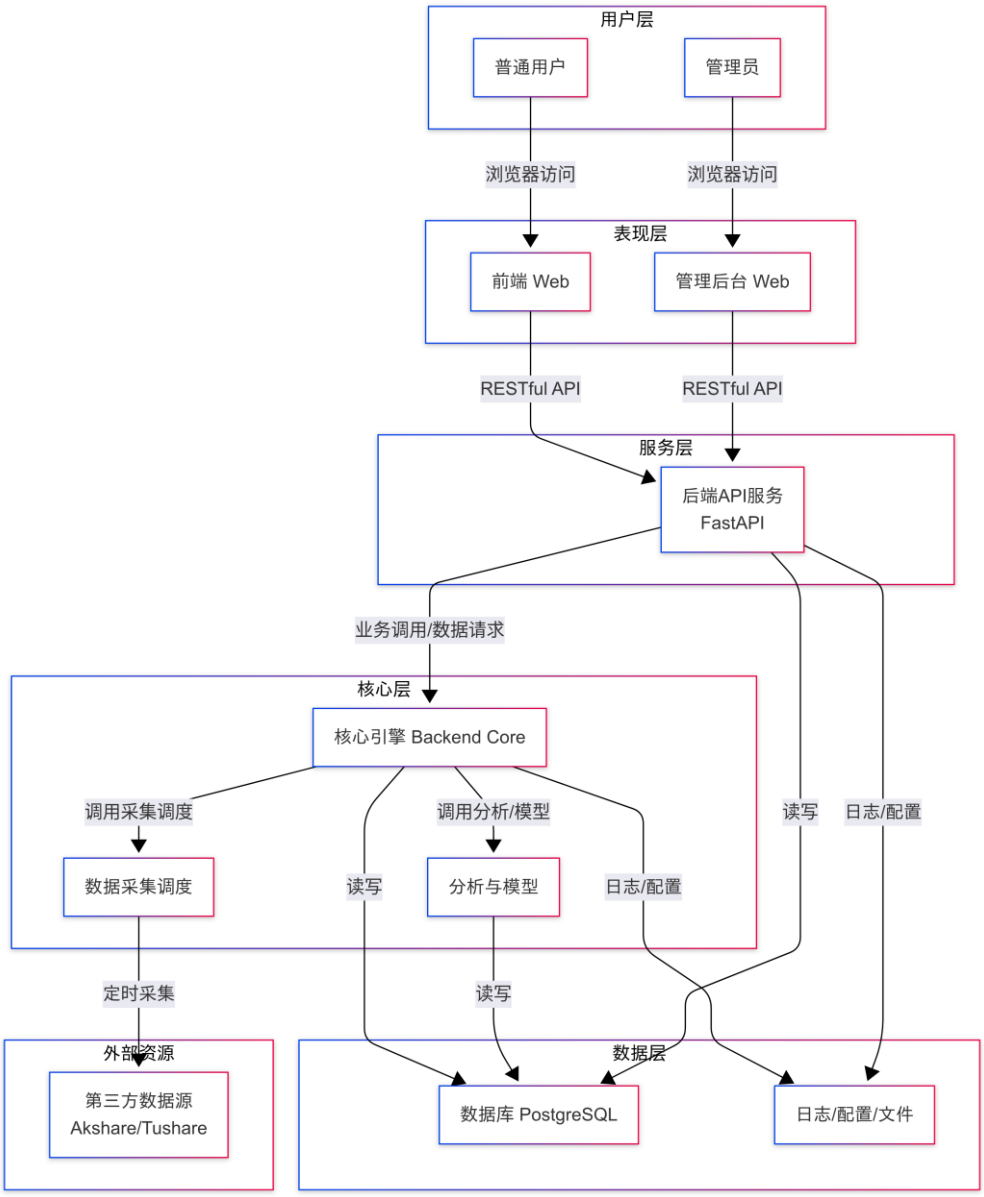
本章节全面概述股票分析系统的架构，涵盖整体系统设计、数据流模式以及整个平台的组件关

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**、**后端 API 服务**、**前端应用程序**和**数据库与存储**。

2.2. 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。

2. 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。

3. 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。

4. 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。

5. 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。

6. 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。

3 数据采集系统

3.1. 采集调度

3.1.1. 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**、**Tushare 数据收集器**和 **Watchlist 数据收集器**等。

3.1.2. 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)

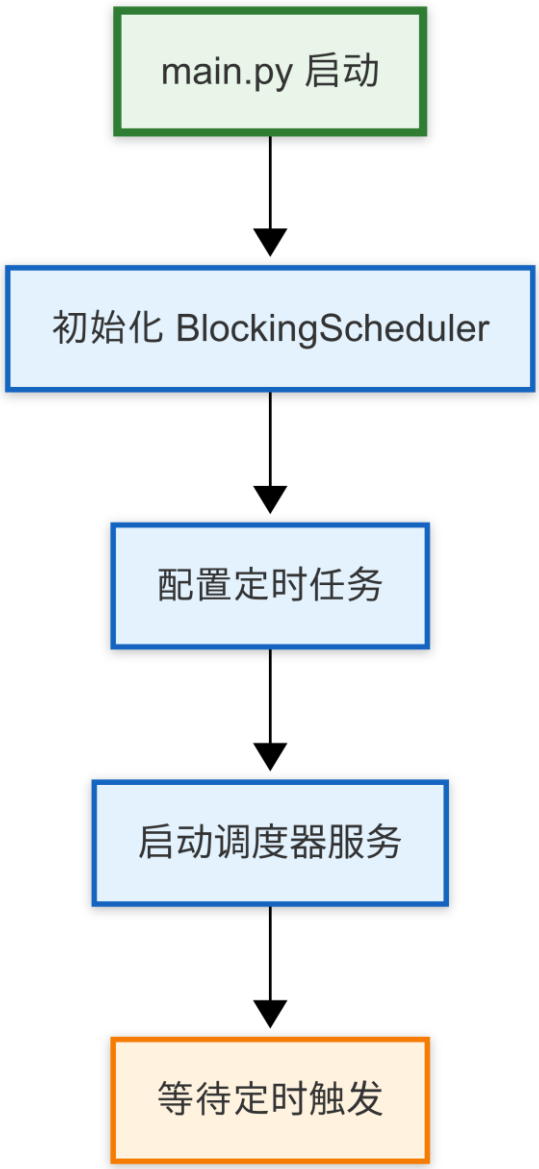
- Tushare 采集器：历史数据、实时数据
- `ak_collector`: AKShare 实时行情采集器

- 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

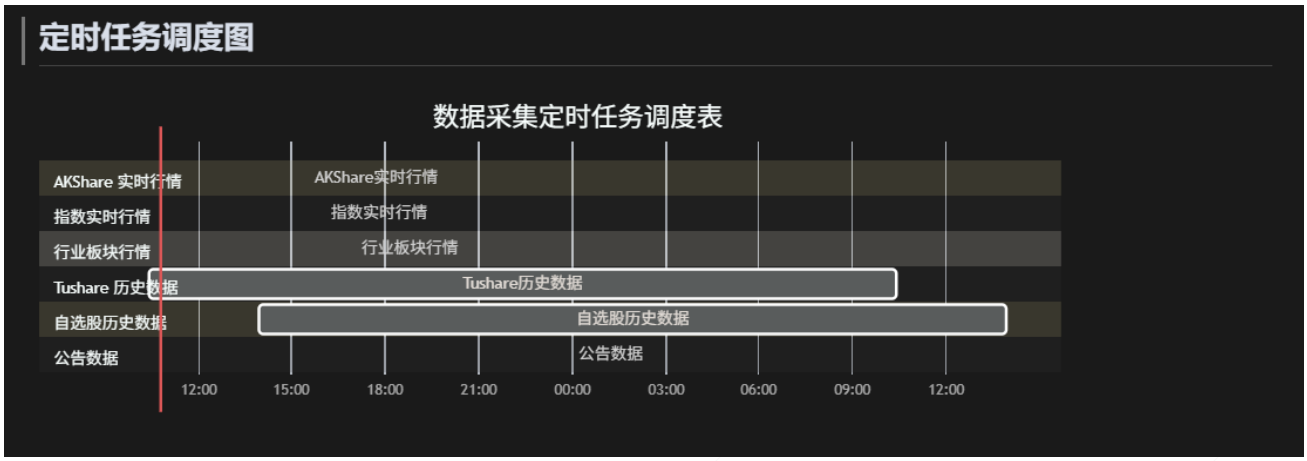
✧ 配置层(Configuration Layer)

- DATA_COLLECTORS 统一配置管理

3.1.3. 系统启动流程

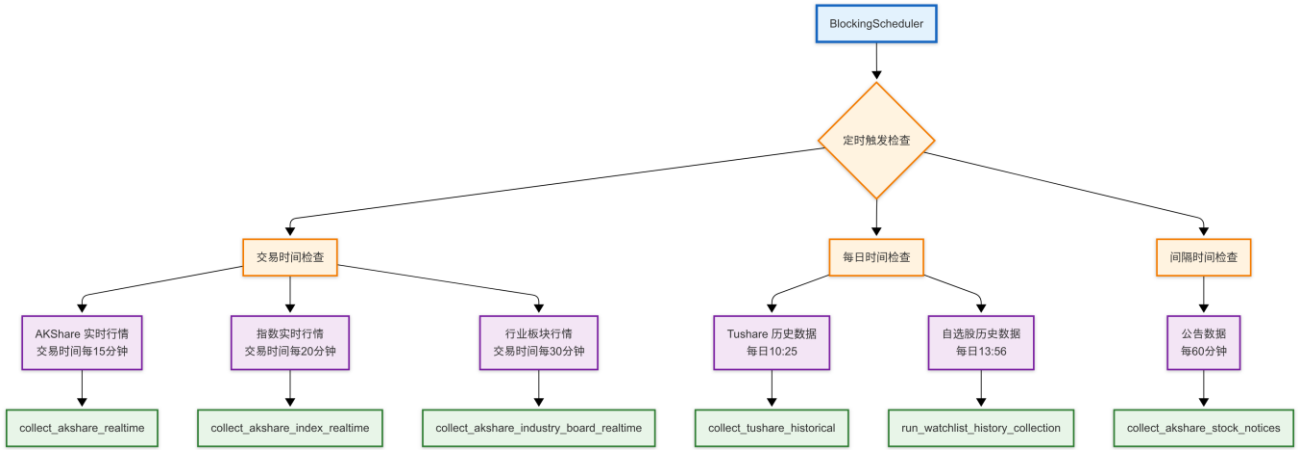


3.1.4. 采集任务计划

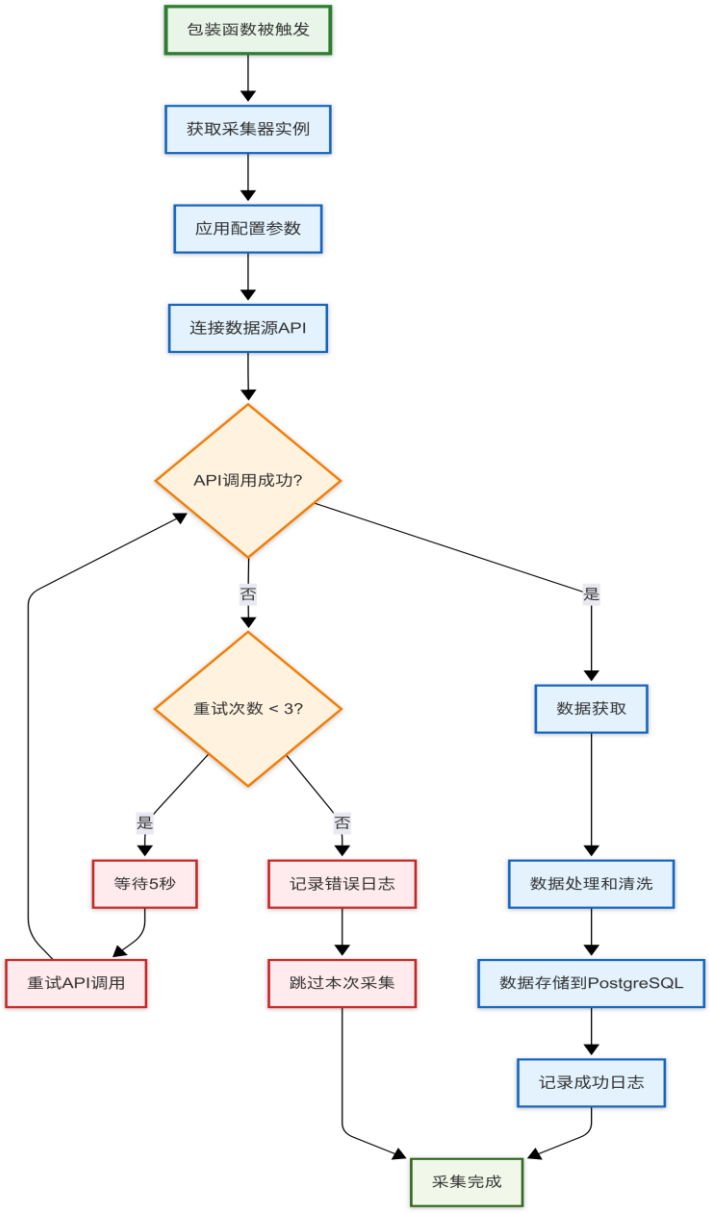


任务名称	调度策略	包装函数	采集器实例
AKShare 实时行情	交易时间每 15 分钟	`collect_akshare_realtime()`	`ak_ collector`
Tushare 历史数据	每日 10:25	`collect_tushare_historical()`	`tushare_hist_collector`
指数实时行情	交易时间每 20 分钟	`collect_akshare_index_realtime()`	`index_collector`
行业板块行情	交易时间每 30 分钟	`collect_akshare_industry_board_realtime()`	`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



3.1.5. 数据采集执行流程

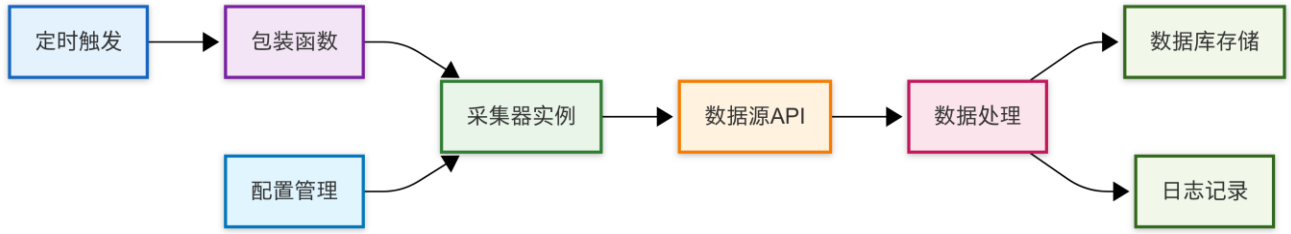


错误处理和重试机制

✧ **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

3.1.6. 数据流图



3.2. Akshare 数据采集器

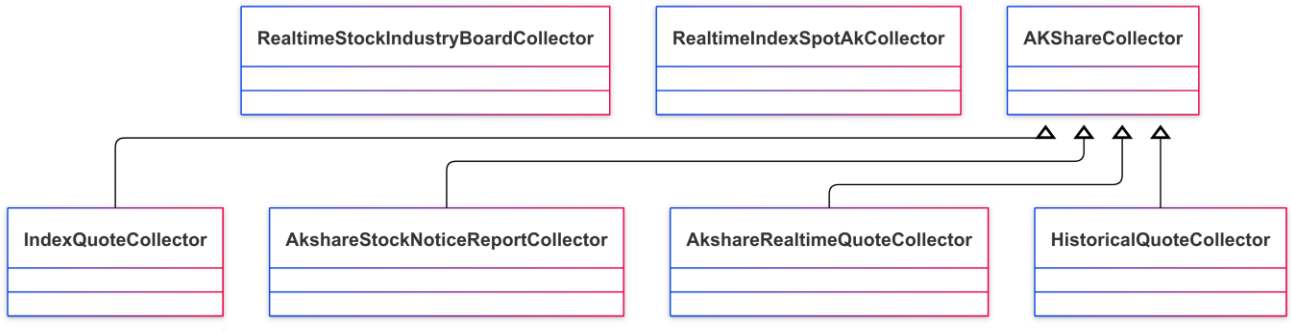
本章节介绍 Akshare 数据收集子系统，该系统负责通过 Akshare API 自动收集中国股市数据。该系统收集实时行情、行业板块表现、股票公告、指数数据和历史交易数据，并将其存储在 PostgreSQL 数据库表中。

整体数据采集编排和调度请参见**采集调度**。 Tushare API 的历史数据采集请参见 **Tushare 数据采集器**。

3.2.1. 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。

3.2.1.1. 类层次结构



说明

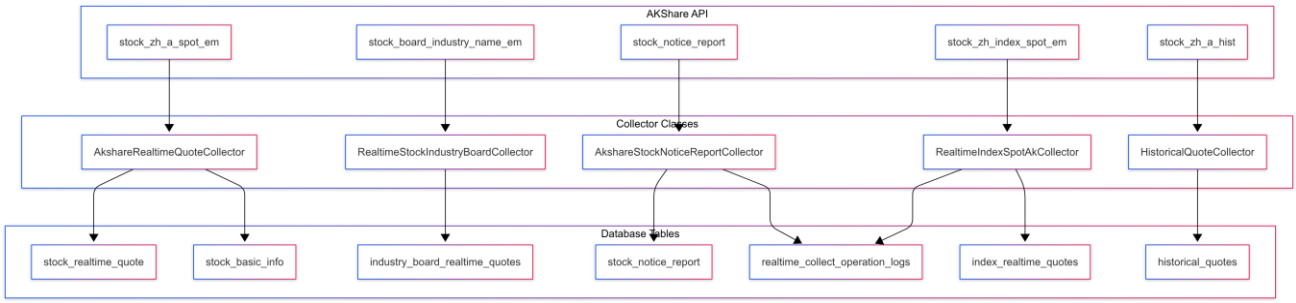
- 核心基类：AKShareCollector，其子类负责不同类型的数据采集。
- 部分采集器（如行业板块、指数实时采集器）未继承基类，但实现方式类似，属于同一采集体系。

- 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
- backend_core/data_collectors/akshare/realtime.py
- backend_core/data_collectors/akshare/historical.py
- backend_core/data_collectors/akshare/realtime_stock_notice_report_ak.py
- backend_core/data_collectors/akshare/realtime_stock_industry_board_ak.py
- 后端核心/数据收集器/akshare/realtime_index_spot_ak.py

3.2.1.2. 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。

3.2.2. 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

3.2.3. 实时数据采集器

3.2.3.1. 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。

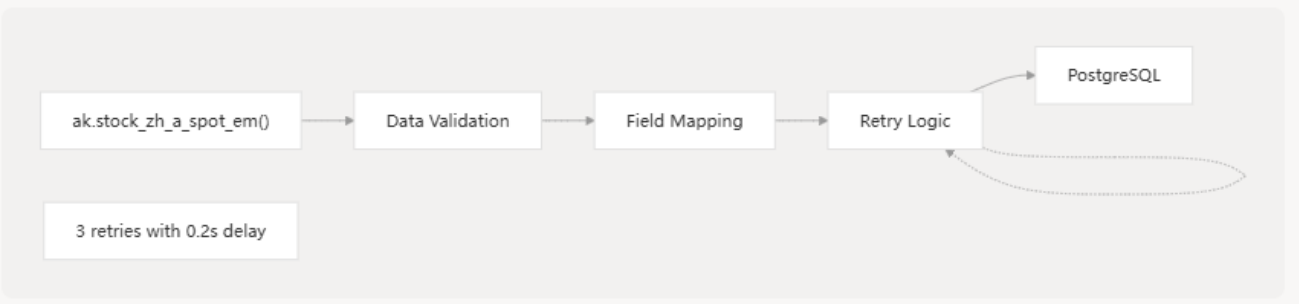
数据库模式：

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
- …

3.2.3.2. 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

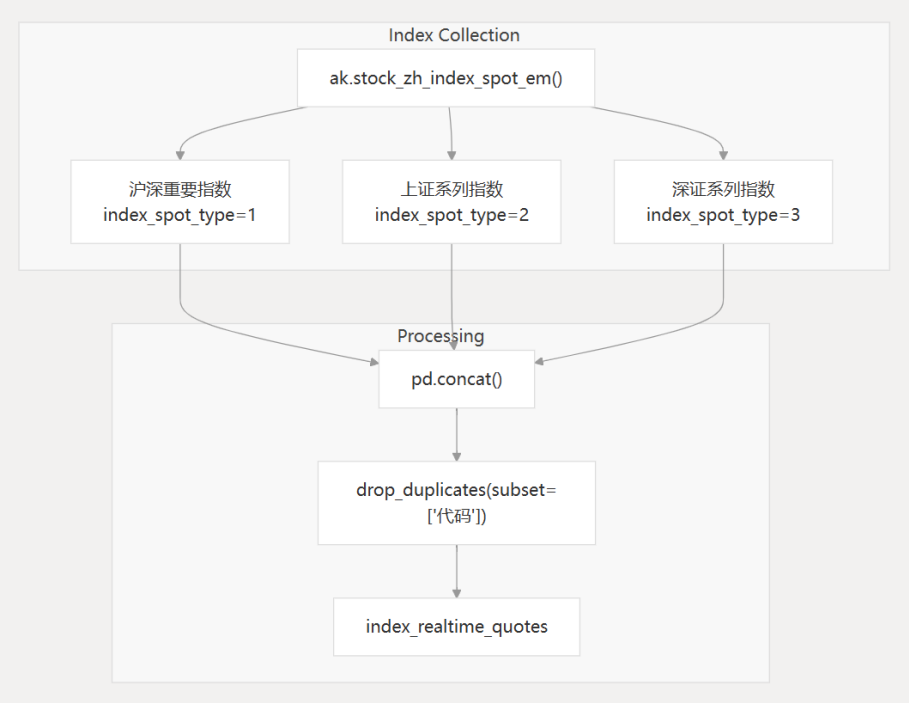
采集器系统字段映射说明：

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

3.2.3.3. 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：



3.2.3.4. 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：

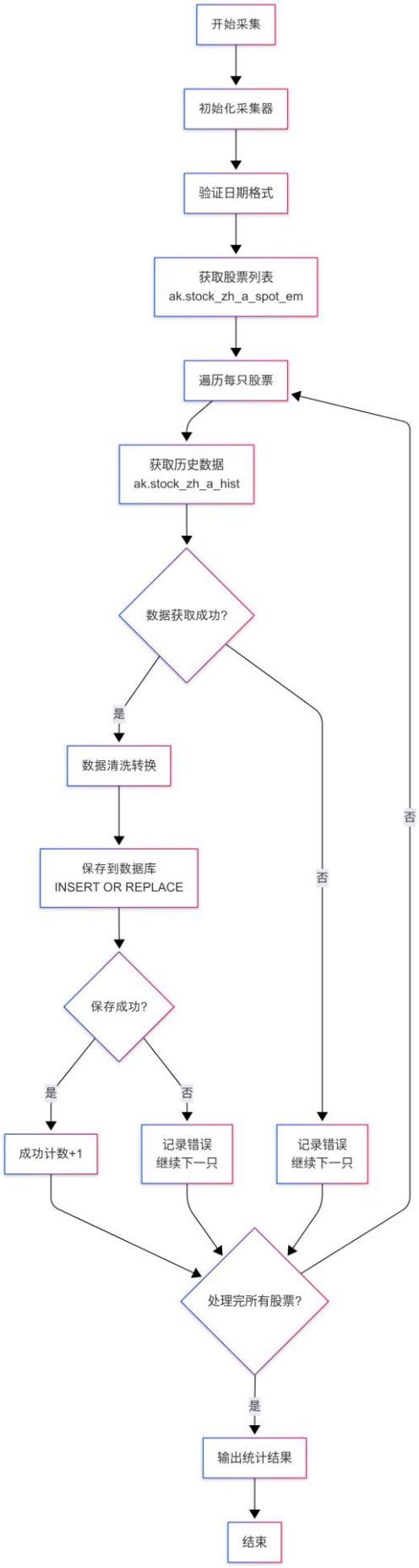
```
def collect_multiple_types(self, date_str: Optional[str] = None) -> bool:
    notice_types = ["全部", "重大事项", "财务报告", "融资公告",
                    "风险提示", "资产重组", "信息变更", "持股变动"]

    for notice_type in notice_types:
```

```
        self.collect_stock_notices(symbol=notice_type, date_str=date_str)
        time.sleep(2)  # Rate limiting
```

3.2.4. 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：


开始采集
初始化采集器
验证日期格式
获取股票列表
ak.stock_zh_a_spot_em
遍历每只股票
获取历史数据
ak.stock_zh_a_hist
数据获取成功?
是
否
数据清洗转换
保存到数据库
INSERT OR REPLACE
保存成功?
是
否
成功计数+1
记录错误
继续下一只
记录错误
继续下一只
处理完所有股票?
否
是
输出统计结果
结束

3.2.5. 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

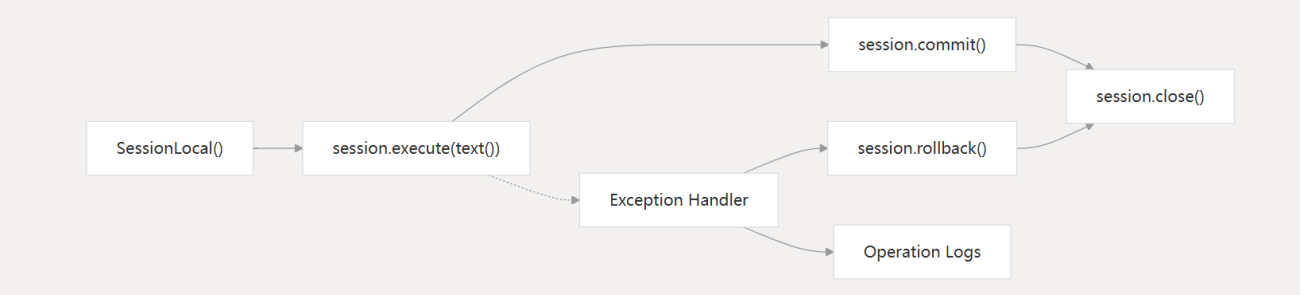
信号处理允许在长时间运行的操作期间正常关闭：

```
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
```

3.2.6. 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。

3.3. Tushare 数据采集器

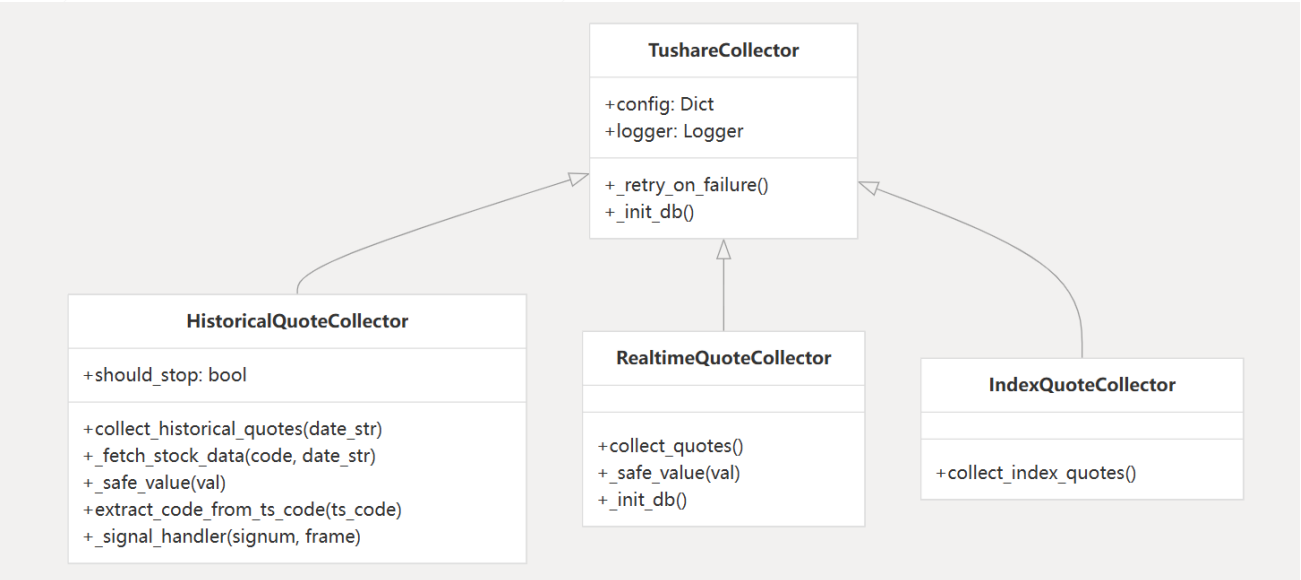
Tushare 数据采集子系统通过 Tushare API 提供专门的金融市场数据采集功能。该模块处理历史股票报价、实时市场数据和指数信息采集。

对于基于 Akshare 的数据收集，请参阅 **Akshare 数据采集器**。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**。

3.3.1. 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。

3.3.1.1. 类层次结构



3.3.1.2. 数据库集成

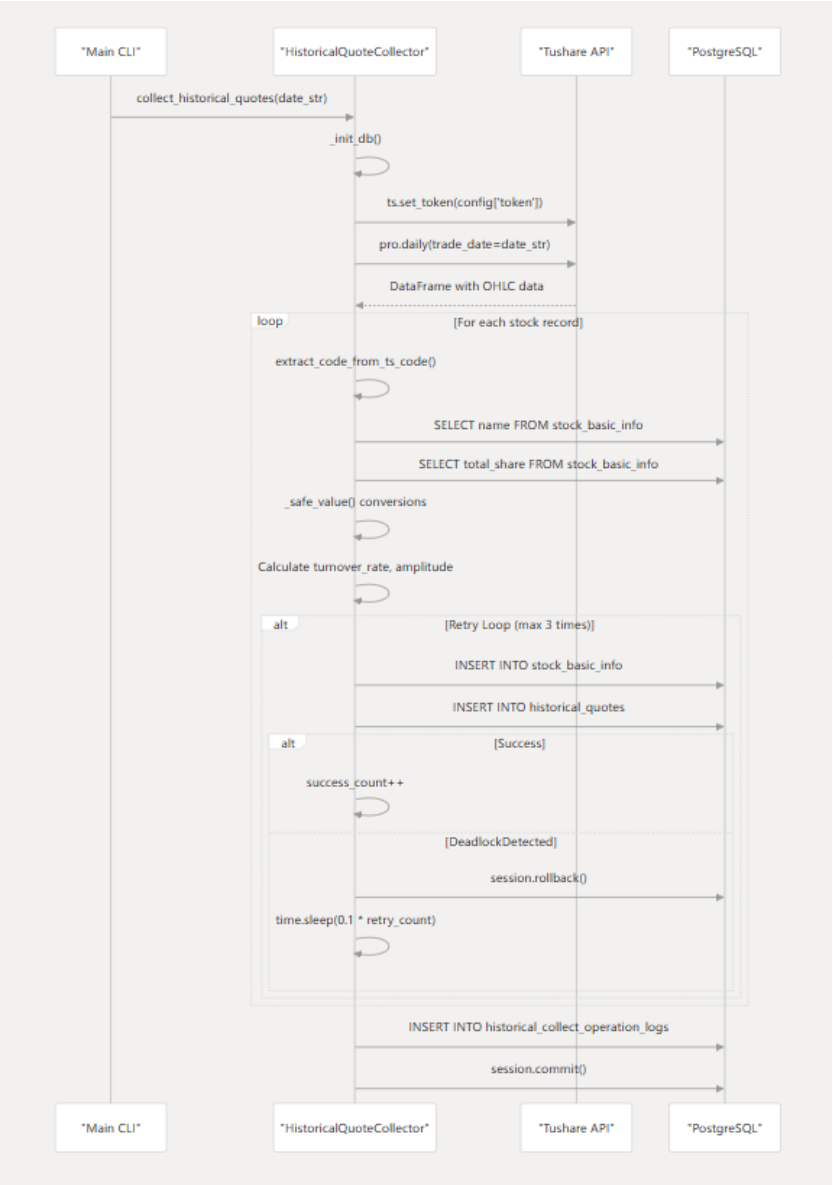
3.3.2. 历史行情数据采集器

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

3.3.2.1. 主要特性

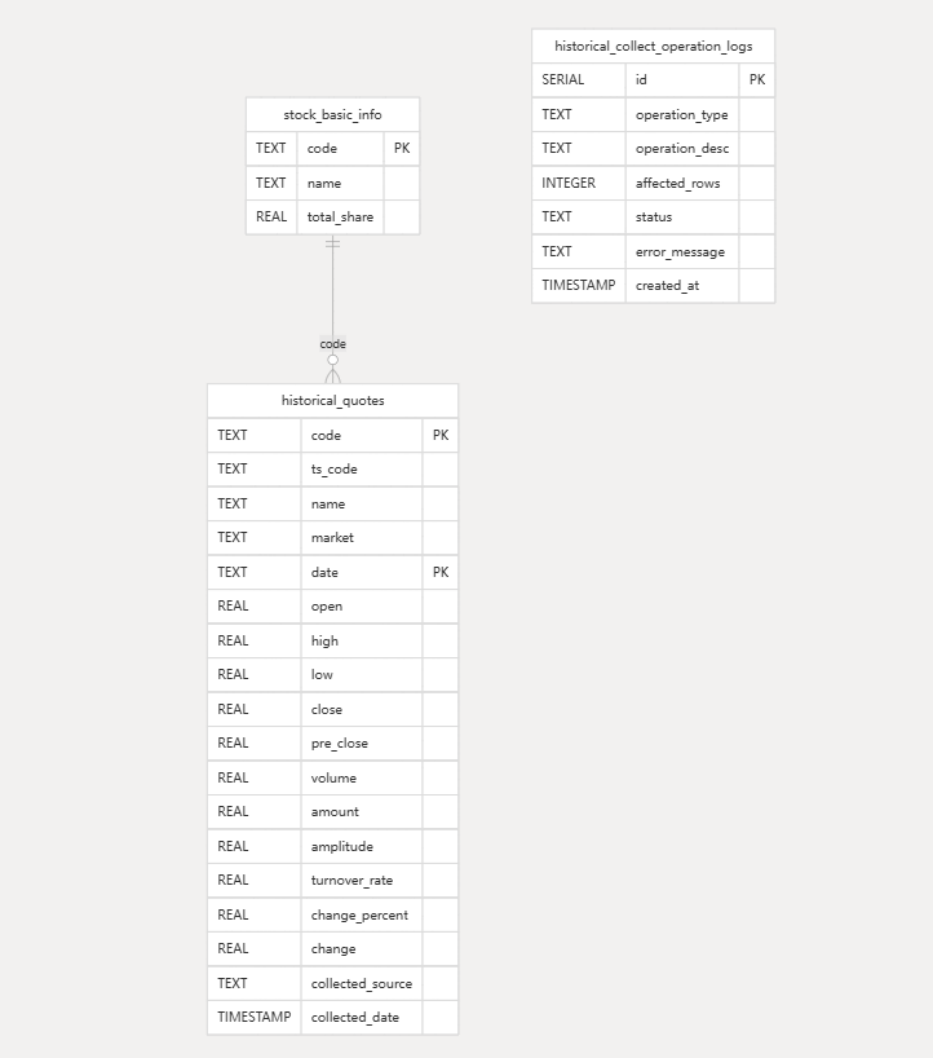
特性	实现	目的
信号处理	`_setup_signal_handlers()`	SIGINT/SIGTERM 时正常关闭
死锁恢复	采用指数退避算法的重试机制	数据库事务弹性
数据验证	`_safe_value()`用于数字转换	干净数据处理
股票代码解析	`extract_code_from_ts_code()`	Tushare 格式规范化
批处理	每 100 条记录提交一次	性能优化

3.3.2.2. 数据采集流程



3.3.2.3. 数据库模式操作

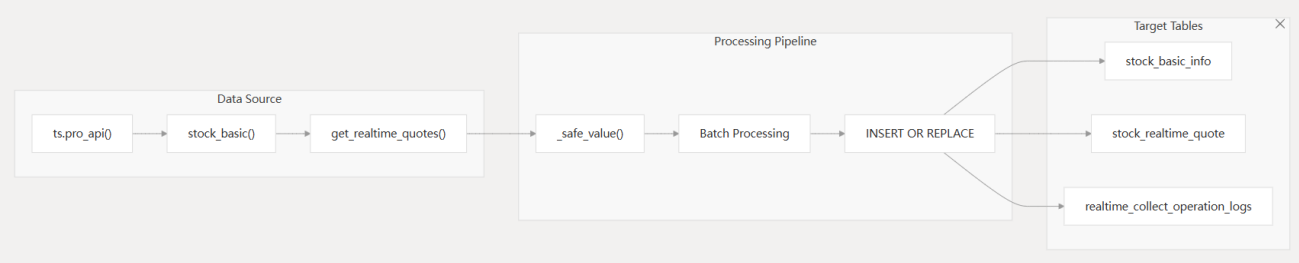
历史采集器管理三个主要相关表:



3.3.3. 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。

3.3.3.1. 采集流程



3.3.4. 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。

3.3.4.1. 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")
```

3.3.5. 配置和操作

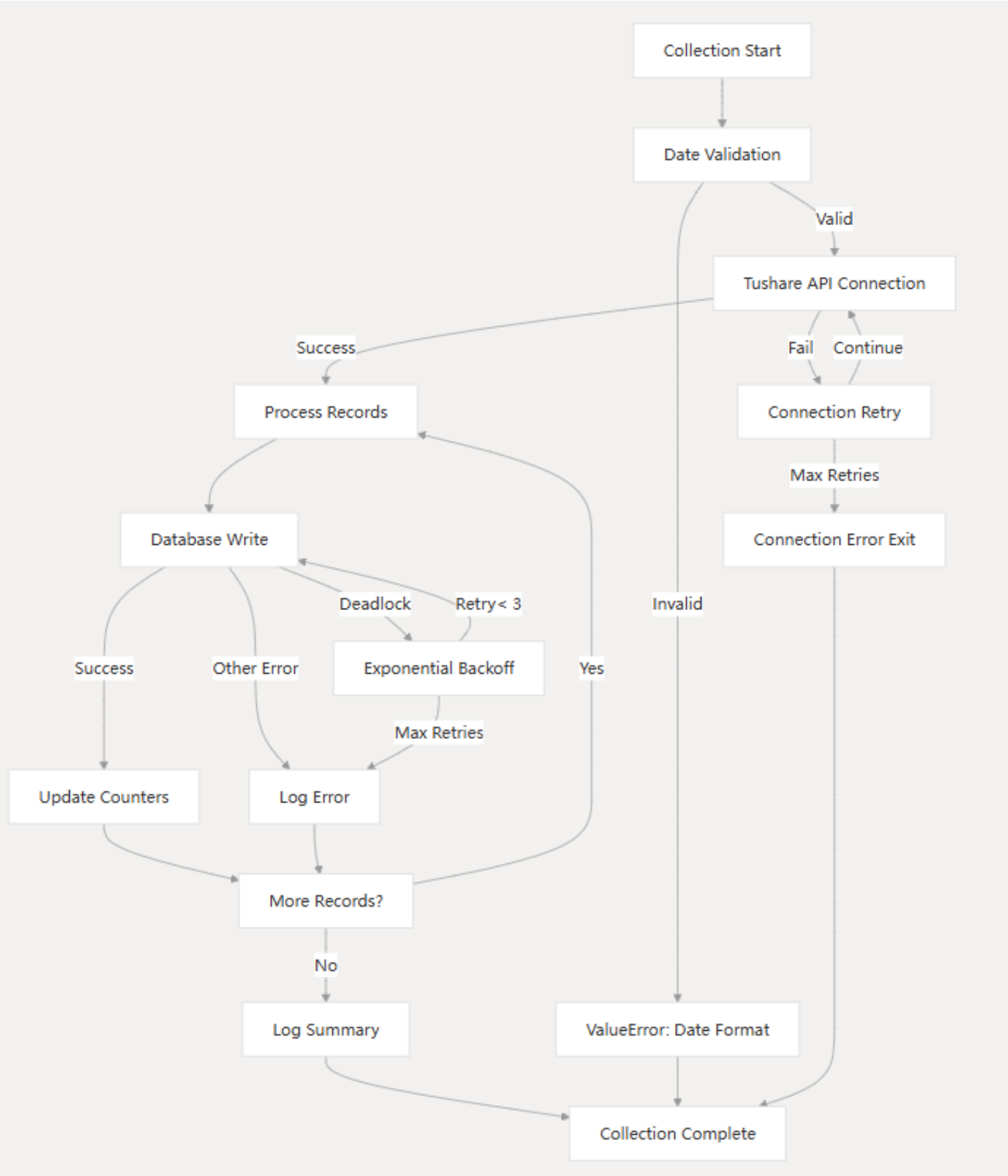
通过 `IndexQuoteCollector` 使用 Tushare 专门的 `pro_bar` 接口采集市场指数数据。

3.3.5.1. 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）

3.3.5.2. 错误处理策略


Collection Start
Date Validation
Valid
Tushare API Connection
Success
Fail
Continue
Process Records
Connection Retry
Max Retries
Database Write
Connection Error Exit
Deadlock
Retry < 3
Invalid
Success
Other Error
Exponential Backoff
Yes
Max Retries
Update Counters
Log Error
More Records?
No
Log Summary
ValueError: Date Format
Collection Complete

3.3.6. 数据质量和验证

3.3.6.1. 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。

3.4. 关注股票列表数据采集

3.4.1. 目的和范围

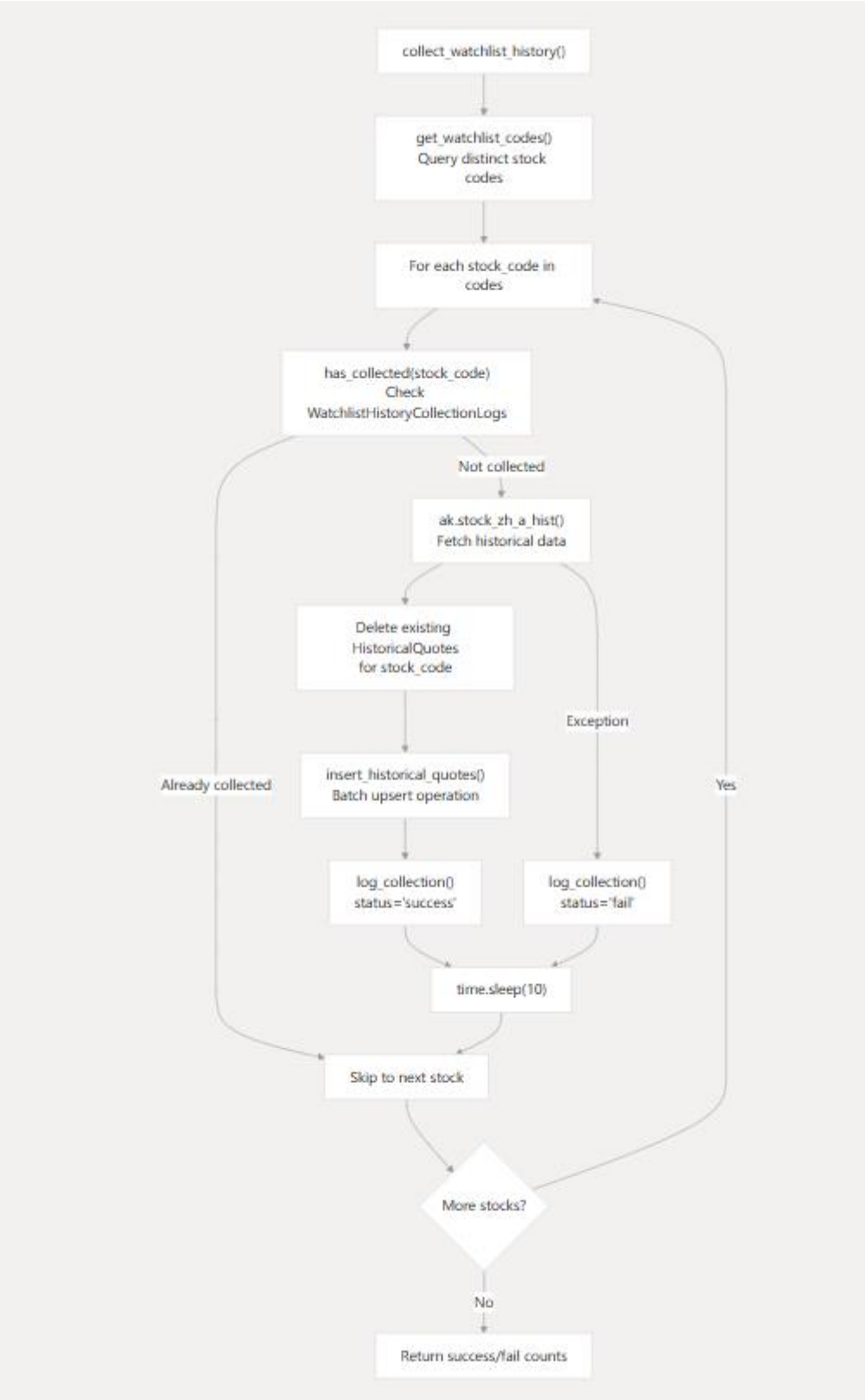
关注列表数据采集系统是数据采集层的一个专用组件，用于检索用户已添加到个人关注列表的股票的历史数据。与 **Tushare 数据采集器**中涵盖的全面历史数据采集器不同，该系统针对特定股票子集按需运行，以优化数据存储和采集效率。

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** 和**历史数据接口**中描述的历史数据分析功能。

3.4.2. 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。

3. 4. 3. 采集工作流程


collect_watchlist_history()
get_watchlist_codes()
Query distinct stock codes
For each stock_code in codes
has_collected(stock_code)
Check WatchlistHistoryCollectionLogs
Not collected
ak.stock_zh_a_hist()
Fetch historical data
Delete existing HistoricalQuotes for stock_code
Exception
Already collected
insert_historical_quotes()
Batch upsert operation
Yes
log_collection()
status='success'
log_collection()
status='fail'
time.sleep(10)
Skip to next stock
More stocks?
No
Return success/fail counts

3.4.4. 主要功能

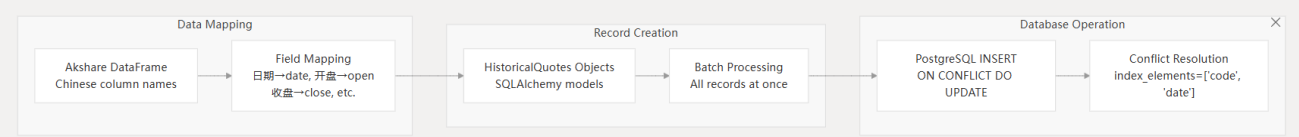
功能	目的	返回类型
`get_watchlist_codes()`	从用户关注列表中检索唯一股票代码	`List[str]`
`has_collected()`	检查股票的历史数据采集是否完成	`Bool`
`insert_historical_quotes()`	执行历史价格数据的批量更新	`Int`
`log_collection()`	记录采集尝试的结果和错误	`None`
`collect_watchlist_history()`	采集流程的主要编排功能	`Dict[str, int]`

3.4.5. 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。

3.4.5.1. 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。



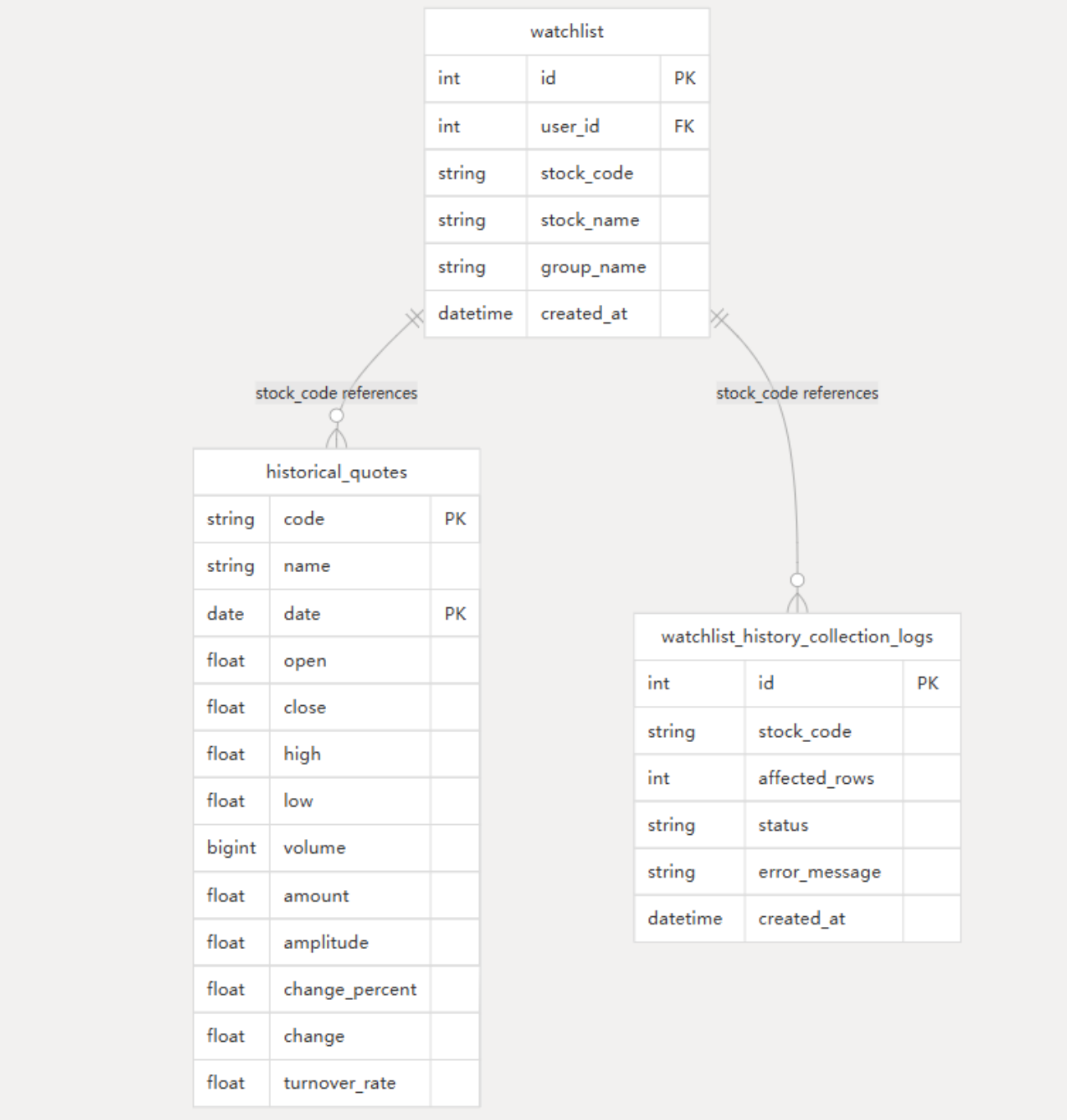
upsert 操作处理以下字段：

open,close,high,low,volume， amount,amplitude， change_percent,changeturnover_rate, code， name,date

3.4.6. 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

3.4.7. 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	

4 后端 API 服务

4.1. 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**。有关使用这些 API 的前端应用程序，请参阅**前端应用程序**。有关数据库模型和架构的详细信息，请参阅**数据库和存储**。

4.1.1. 应用程序架构

后端使用 FastAPI 框架构建，并组织成模块化路由器组件。主应用程序负责协调 CORS 中间件、请求日志记录、静态文件服务和路由器注册。

4.1.1.1. FastAPI 应用程序架构

4.1.1.2. 应用程序配置和启动

4.1.2. 路由器组织

后端 API 被组织成功能性的路由器模块，每个模块负责处理特定的领域。路由器通过调用注册到主 FastAPI 应用程序中 app.include_router()。

4.1.2.1. 路由器注册流程

4.1.2.2. API 端点组织

4.1.3. 请求处理流程

4.1.4. 数据库集成

4.1.5. 启动和发展

4.1.6. 静态文件服务？

5 前端应用程序

6 数据库和存储

行情数据延迟：实时行情数据在数据源更新后，前端展示延迟不超过 [5] 秒。
页面加载速度：各主要页面（自选、个股详情）加载时间不超过 [5] 秒。
图表响应：K 线图缩放、平移、切换周期、叠加指标等操作响应流畅，延迟不超过 [500] 毫秒。
API 响应时间：大多数 API 请求应在 [5] 秒内响应。
数据采集效率：能够在规定时间内完成日终、周终等数据采集任务。

7 配置和部署

- 用户认证与授权：确保用户账号安全，API 调用权限控制。
- 数据传输安全：前后端、后端各服务之间使用加密通信（HTTPS, TLS）。
- 敏感数据保护：用户密码、API 密钥等敏感信息加密存储，防止数据泄露。
- 防御常见攻击：防御 SQL 注入、XSS、CSRF、DDoS 等安全威胁。
- 预测模型安全：保护预测模型算法和数据不被非法获取。

7.1. 高可用性需求

系统可用性：系统核心功能（行情、个股详情）年可用率目标达到 [99% 或更高]。
数据一致性：确保存储和展示的数据准确和一致。
容错性：系统应能处理部分组件故障，避免整个系统崩溃。
数据备份与恢复：定期备份数据，具备快速恢复能力。

柱子	类型	约束	描述
id	Integer	主键，自增	唯一任务标识符
task_type	String(20)	非空	任务类型：“实时”或“历史”
status	String(20)	非空	状态：‘待处理’、‘运行’、‘完成’、‘失败’
progress	Float	默认值：0.0	完成百分比（0.0-1.0）
error_message	String(200)	可空	任务失败时的错误详细信息
started_at	DateTime	可空	任务开始时间戳
completed_at	DateTime	可空	任务完成时间戳

柱子	类型	约束	描述
created_at	DateTime	默认值：now()	任务创建时间戳
updated_at	DateTime	默认值：now()，自动更新	上次状态更新时间戳