# 个股分时图功能实现

@stock\_manage.py 增加一个api， 通过akshare调用stock\_intraday\_em接口， 返回股票代码对应的当日分时数据。 前端@stock.js去除分时数据模拟生成相关逻辑，改为调用后台api获取分时数据并显示。

## 1. 测试用例设计（pytest+httpx，伪代码）

**目标API**：GET /api/stock/minute\_data\_by\_code?code=xxxxxx

**预期返回**：

* success: True
* data: 包含时间、价格、成交量等分时数据列表

**测试点：**

* 正常传入有效股票代码，返回分时数据，字段齐全，数据量大于0。
* 传入无效或不存在的股票代码，返回success: False，有错误信息。
* 未传code参数，返回400。

## 2. 后端API实现思路

* 路由：@router.get("/minute\_data\_by\_code")
* 参数：code（必填）
* 调用：ak.stock\_intraday\_em(symbol=code, adjust='qfq')
* 返回：时间、最新价、成交量、均价等（字段按akshare返回为主，必要时做字段映射/精简）

## 3. 前端stock.js调整思路

* 移除generateMinuteData等分时模拟数据生成逻辑。
* 新增loadMinuteData()方法，调用新API，获取数据后渲染分时图。
* switchChartType('minute')时，自动调用loadMinuteData()并刷新分时图。

已完成分时数据API的pytest+httpx测试用例编写，覆盖以下场景：

* 正常返回分时数据：传入有效股票代码，返回数据为list且字段齐全。
* 股票代码不存在：传入无效代码，返回404和错误信息。
* 缺少参数：未传code参数，返回400和错误信息。

后端 /api/stock/minute\_data\_by\_code 分时数据API 已实现，主要特性如下：

* **参数**：code（必填，股票代码）
* **功能**：调用 akshare 的 stock\_intraday\_em 获取当日分时数据
* **返回**：每条数据包含 time（时间）、price（最新价）、volume（成交量）、avg\_price（均价）、amount（成交额）、change\_percent（涨跌幅）、change（涨跌额）
* **异常处理**：无参数返回400，未找到返回404，其他异常返回500

前端分时图表改造方案如下：

### 3.1 变更点梳理

* **移除**：generateMinuteData 及相关模拟数据生成逻辑。
* **新增**：loadMinuteData 方法，通过API获取分时数据并渲染分时图。
* **调整**：
* initMinuteChart 初始化时不再设置模拟数据，或设置空数据。
* loadChartData 在 currentChartType === 'minute' 时，调用 loadMinuteData，并用API数据刷新分时图。
* 切换“分时”按钮时自动拉取并渲染分时数据。

### 3.2 主要实现思路

* loadMinuteData 方法：

1. 请求 /api/stock/minute\_data\_by\_code?code=xxxxxx
2. 拿到数据后，分别提取 time 作为 x 轴，price 作为 y 轴，刷新 ECharts 分时图。

### 3.3. 代码修改要点

* 删除 generateMinuteData 方法。
* initMinuteChart 只初始化空数据。
* loadMinuteData 负责请求API并渲染分时图。
* loadChartData 分时分支调用 loadMinuteData。

## 4 后端增加是否交易日判断

如果是交易日正常逻辑不变，如果是非交易日，通过akshare访问 【盘前数据】， 获取最近一个交易日的股票分钟数据。

#### 4.1 测试用例设计（伪代码）

* **交易日**：传入有效股票代码，返回当天分时数据，字段齐全。
* **非交易日**：传入有效股票代码，返回最近一个交易日的分钟数据，字段齐全，且数据日期为最近的交易日。
* **无效代码/缺参**：同原有测试。

#### 4.2 实现思路

* 判断当天是否为A股交易日（可用 ak.tool\_trade\_date\_hist\_sina() 或 ak.tool\_trade\_date\_hist\_sina()['trade\_date'].tolist() 判断）。
* 若为交易日，调用 ak.stock\_intraday\_em(symbol=code, adjust="qfq")（原逻辑）。
* 若为**非交易日**，调用 ak.stock\_zh\_a\_hist\_min\_em(symbol=code, period='1m', adjust='qfq')，取最近一个交易日的数据（可按日期分组取最大日期）。
* 返回数据结构与交易日一致。

#### 4.3 代码实现要点

* 增加交易日判断逻辑。
* 非交易日时，处理分钟数据的日期筛选，返回最近一个交易日的所有分钟数据。

**实现方案如下：**

1. 判断今天是否为A股交易日（ak.tool\_trade\_date\_hist\_sina）。
2. 若是交易日，调用 ak.stock\_intraday\_em（原逻辑）。
3. 若非交易日，调用 ak.stock\_zh\_a\_hist\_min\_em，取最近一个交易日的分钟数据（按日期分组取最大日期）。
4. 日志打印输入参数、分支、返回条数、前3条数据。

# 个股K线图-日线图功能实现

## 后端

@stock\_manage.py 增加一个api， 通过akshare调用stock\_zh\_a\_hist接口。输入： 股票代码、周期（按天）、开始日期（当前日期-YYMMDD）、结束日期（当前日期往前5年-YYMMDD）、复权类型（前复权-qfq）；输出：日期、股票代码、开盘、收盘、最高、最低、成交量（手）、成交额（元）、振幅（%）、涨跌幅（%）、涨跌额（元）、换手率（%），后台打印输入输出日志。

## 前端

[@stock.js去除K线图](mailto:前端@stock.js去除K线图)中日线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示。

## 实现方案

### 3.1 后端API设计与测试用例（pytest+httpx，伪代码）

**API设计**

* 路由：/api/stock/kline\_hist
* 方法：GET
* 入参：code（股票代码）、period（如'daily'）、start\_date、end\_date、adjust（如'qfq'）
* 出参：列表，字段包括：日期、股票代码、开盘、收盘、最高、最低、成交量（手）、成交额（元）、振幅（%）、涨跌幅（%）、涨跌额（元）、换手率（%）

**测试点**

1. **正常查询**：传入有效股票代码、日线、起止日期、前复权，返回数据字段齐全，数据量大于0。
2. **无效代码**：返回404和错误信息。
3. **缺少参数**：返回400。
4. **字段格式**：所有数值字段小数点后两位，成交量为整数。

### 3.2 前端stock.js调整思路

* 移除K线图日线模拟数据生成逻辑。
* 新增loadKlineData()方法，调用新API，获取数据后渲染K线图。
* 切换到“K线-日线”标签时，自动调用API并刷新图表。

K线历史API的pytest+httpx测试用例已补充，覆盖：

* **正常返回**：字段齐全，数据量大于0。
* **股票代码不存在**：返回404和错误信息。
* **缺少参数**：返回400和错误信息。

后端 /api/stock/kline\_hist 已实现，特性如下：

* 支持参数：股票代码、周期（如daily）、开始/结束日期、复权类型
* 返回字段：日期、股票代码、开盘、收盘、最高、最低、成交量（手）、成交额（元）、振幅（%）、涨跌幅（%）、涨跌额（元）、换手率（%）
* 所有数值字段小数点后两位，成交量为整数
* 打印输入输出日志，便于调试

# 个股K线图-周线图功能实现

## 前端

[@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 周线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示。

## 后台

利旧 @stock\_manage.py /api/stock/kline\_hist， 无需修改。

前端已完成如下改造，K线图“周线”功能正式对接后端API：

* **注释/移除所有周线模拟数据生成逻辑**，彻底不再本地生成周线假数据。
* **loadKlineData方法**：根据当前周期（this.currentPeriod），自动切换API参数为period=weekly，获取真实周线数据并渲染。
* **切换周期按钮**：点击“周线”时，自动刷新K线图，显示后端返回的真实周线K线。

# 个股K线图-月线图功能实现

## 前端

[@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 月线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示。

## 后台

后台利旧 @stock\_manage.py /api/stock/kline\_hist， 无需修改。

前端已完成月线功能对接，具体如下：

* **彻底移除月线模拟数据生成逻辑**，所有月线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '1M'时，自动将API参数设为period=monthly，获取真实月线数据并渲染。
* **切换周期按钮**：点击“月线”时，自动刷新K线图，显示后端返回的真实月线K线。
* 你只需刷新页面，点击“月线”标签，即可看到真实历史月线K线图。

# 个股K线图-小时线图功能实现

## 后台

@stock\_manage.py 增加一个api， 通过akshare调用stock\_zh\_a\_hist\_min\_em接口。输入： 股票代码、周期（1、5、15、30、60，单位：分钟）、开始日期（当前日期：YYYY-MM-DD HH:MM:SS）、结束日期（当前日期往前2年：YYYY-MM-DD HH:MM:SS）、复权类型（前复权-qfq）；输出：时间、股票代码、开盘、收盘、最高、最低、成交量（手）、成交额（元）、振幅（%）、涨跌幅（%）、涨跌额（元）、换手率（%），后台打印输入输出日志。

## 前端

[前端@stock.js 注释去除K线图](mailto:前端@stock.js去除K线图)小时线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示。

后端已完成小时线（及任意分钟线）K线API：

* 新增 /api/stock/kline\_min\_hist，支持参数：股票代码、周期（1/5/15/30/60分钟）、开始/结束时间（精确到秒）、复权类型。
* 自动格式化时间参数，返回字段与日K一致，所有数值字段小数点后两位，成交量为整数。
* 日志详细，便于排查。

前端已完成小时线（1小时K线）功能对接，具体如下：

* **彻底移除小时线模拟数据生成逻辑**，所有小时线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '1h'时，自动调用/api/stock/kline\_min\_hist，获取2年区间的1小时K线数据并渲染。
* **切换周期按钮**：点击“1小时”标签时，自动刷新K线图，显示后端返回的真实1小时K线。

# 个股K线图-30分钟线图功能实现

## 前端

[前端@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 30分钟线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示，除前台传递周期参数为30分外，其他参数值相同。

## 后台

后台利旧调用 @stock\_manage.py， /api/stock/kline\_min\_hist， 除前台传递周期参数为30分外，其他无需修改。

前端已完成30分钟K线图功能对接，具体如下：

* **彻底移除30分钟线模拟数据生成逻辑**，所有30分钟线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '30m'时，自动调用/api/stock/kline\_min\_hist，period参数为30，获取2年区间的30分钟K线数据并渲染。
* **切换周期按钮**：点击“30分钟”标签时，自动刷新K线图，显示后端返回的真实30分钟K线。

# 个股K线图-15分钟线图功能实现

## 前端

[前端@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 15分钟线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示，除前台传递周期参数为15分外，其他参数值相同。

## 后台

后台利旧调用 @stock\_manage.py， /api/stock/kline\_min\_hist， 除前台传递周期参数为15分外，其他无需修改。

前端已完成15分钟K线图功能对接，具体如下：

* **彻底移除15分钟线模拟数据生成逻辑**，所有15分钟线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '15m'时，自动调用/api/stock/kline\_min\_hist，period参数为15，获取2年区间的15分钟K线数据并渲染。
* **切换周期按钮**：点击“15分钟”标签时，自动刷新K线图，显示后端返回的真实15分钟K线。

# 个股K线图-5分钟线图功能实现

## 前端

[前端@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 5分钟线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示，除前台传递周期参数为5分外，其他参数值相同。

## 后台

后台利旧调用 @stock\_manage.py， /api/stock/kline\_min\_hist， 除前台传递周期参数为5分外，其他无需修改。

前端已完成5分钟K线图功能对接，具体如下：

* **彻底移除5分钟线模拟数据生成逻辑**，所有5分钟线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '5m'时，自动调用/api/stock/kline\_min\_hist，period参数为5，获取2年区间的5分钟K线数据并渲染。
* **切换周期按钮**：点击“5分钟”标签时，自动刷新K线图，显示后端返回的真实5分钟K线。

# 个股K线图-1分钟线图功能实现

## 前端

[前端@stock.js 注释掉K线图](mailto:前端@stock.js去除K线图) 1分钟线标签页数据模拟生成相关逻辑，改为调用后台api获取历史数据并显示，除前台传递周期参数为1分外，其他参数值相同。

## 后台

后台利旧调用 @stock\_manage.py， /api/stock/kline\_min\_hist， 除前台传递周期参数为1分外，其他无需修改。

前端已完成1分钟K线图功能对接，具体如下：

* **彻底移除1分钟线模拟数据生成逻辑**，所有1分钟线数据均通过API获取。
* **loadKlineData方法**：支持this.currentPeriod === '1m'时，自动调用/api/stock/kline\_min\_hist，period参数为1，获取2年区间的1分钟K线数据并渲染。
* **切换周期按钮**：点击“1分钟”标签时，自动刷新K线图，显示后端返回的真实1分钟K线。

# 个股资金流向功能实现

## 前端

前端@stock.js 资金流向标签页，注释模拟数据，调用 /api/stock/fund\_flow, 显示真实资金流向数据。

## 后台

后台@backend\_api, stock目录下，新建个股资金流向python文件，新增根据股票代码查询个股资金流向api，api前缀为/api/stock，调用akshare框架.

前台已完成以下修改：

* 注释了资金流向的模拟数据生成函数（generateFlowData、generateFlowDates）。
* 重写了loadFlowData 方法，改为调用后端 /api/stock/fund\_flow/{code} 接口，获取真实资金流向数据并渲染到flowChart。
* 你只需保证后端API返回的数据字段与前端解析字段一致（如主力净流入-净额、散户净流入-净额、日期等）。

# 个股财务数据-关键指标展示功能实现

## 后端

@stock\_manage.py 增加根据股票代码查询个股最新报告期财务报表api，通过akshare接口获取股票对应的最新报告期的财务指标，包括市盈率、市净率、净资产收益率(ROE)、资产收益率(ROA)、营业收入、净利润、每股收益（EPS）、每股净资产（BPS）等参数。

## 前端

前端stock.js调用api，获取市盈率、市净率、净资产收益率(ROE)、资产收益率(ROA)、营业收入、净利润、每股收益（EPS）、每股净资产（BPS）等参数，给stock.html页面 财务数据 标签页区域对应的股票财务指标赋值。

# 个股财务数据-盈利能力展现实现

## 后端

@stock\_manage.py 增加根据股票代码symbol、指标报告类型indicator查询个股最新报告期财务报表api，通过akshare 【stock\_financial\_abstract\_ths】接口获取股票对应的最新报告期的主要财务指标列表。

输入：symbol，indicator

输出：报告期、净资产收益率(ROE)、资产收益率(ROA)、营业总收入、净利润、基本每股收益（EPS）、每股净资产（BPS）等财务指标数据列表。

## 前端

前端stock.js调用api，获取报告期、净资产收益率(ROE)、资产收益率(ROA)、营业收入、净利润、每股收益（EPS）、每股净资产（BPS）等财务指标数据列表，给stock.html页面 财务数据标签页 盈利能力区域的柱图赋值。

# akshare指数实时行情采集-东财

## 后端

1. @backend\_core\data\_collectors\akshare目录下，
   1. 参考realtime.py新建realtime\_index\_spot\_ak.py，定义指数实时行情采集类 ，api输入输出参考 https://akshare.akfamily.xyz/data/index/index.html中stock\_zh\_index\_spot\_em函数说明。 需要新增指数实时行情表，字段需要涵盖stock\_zh\_index\_spot\_em返回的参数列表，以及更新日期等字段；
   2. 采集数据写入指数实时行情表， 成功或失败写操作日志表realtime\_collect\_operation\_logs。数据库读取配置等信息参考realtime.py, 数据库路径请从 @/backend\_core config.py取值。
   3. 参考maket\_sourtes.py,调用三次, 指数类型index\_spot\_type 1:沪深重要指数; 2:上证系列指数; 3:深证系列指数;
2. @backend\_core\data\_collectors\目录下，main.py新增一个定时任务，引入上面的指数实时行情采集类。每5分钟采集一次。
3. [修改@backend\_api下market\_routes.py](mailto:修改@backend_api下market_routes.py), 修改api /indices，修改为 上证指数、深圳成指、创业板指、沪深300的指数实时行情， 改成从数据库表index\_realtime\_quotes中读取。

# 首页自选股列表显示改造

## 后端

1. [修改@backend\_api下watchlist\_manage.py](mailto:修改@backend_api下watchlist_manage.py), 修改get\_watchlist函数，目前的逻辑为从ak.stock\_bid\_ask\_em(symbol=code)取实时行情报价，修改为从 stock\_realtime\_quote表中读取股票代码对应的实时行情数据，数据显示格式化处理，打印关键输入输出日志。

# 行业板块实时行情数据采集-东财-akshare

## 设计：

**后端**

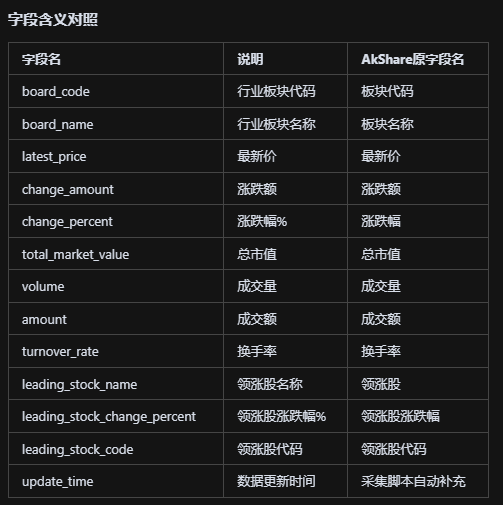
1. @backend\_core\data\_collectors\akshare目录下，
   1. 参考realtime\_index\_spot\_ak.py新建realtime\_stock\_industry\_board\_ak.py，定义行业板块实时行情数据采集类 ，api输入输出参考 @web https://akshare.akfamily.xyz/data/stock/stock.html#id355中stock\_board\_industry\_name\_em函数说明。 需要新增行业板块实时行情表industry\_board\_realtime\_quotes，字段需要涵盖stock\_board\_industry\_name\_em返回的参数列表，以及更新日期等字段；
   2. 采集数据写入industry\_board\_realtime\_quots表， 成功或失败写操作日志表realtime\_collect\_operation\_logs。数据库读取配置路径请从 @/backend\_core config.py取值。
2. @backend\_core\data\_collectors\目录下，main.py新增一个定时任务，引入上面的行业板块实时行情采集类。每30分钟采集一次。
3. [修改@backend\_api下market\_routes.py](mailto:修改@backend_api下market_routes.py), 请通读get\_industry\_board函数现有代码处理业务逻辑，然后修改代码，输入输出参数不变，改成从industry\_board\_realtime\_quotes数据库表中读取，数据库读取方式请参考get\_market\_indices()函数。

## 实现：

a) 新建 realtime\_stock\_industry\_board\_ak.py，采集行业板块实时行情

* 参考 realtime\_index\_spot\_ak.py 的结构，定义采集类。
* 采集数据源为 AkShare 的 stock\_board\_industry\_name\_em。
* 采集结果写入新表 industry\_board\_realtime\_quotes，字段涵盖 AkShare 返回的所有参数及更新时间。
* 采集成功/失败写入 realtime\_collect\_operation\_logs 日志表。
* 数据库路径、表名等配置从 config.py 读取。

b) 新建表结构说明

* industry\_board\_realtime\_quotes 字段（参考 AkShare 文档和实际返回）：
* 
* CREATE TABLE "industry\_board\_realtime\_quotes" (
* "board\_code" TEXT,
* "board\_name" TEXT,
* "latest\_price" REAL,
* "change\_amount" REAL,
* "change\_percent" REAL,
* "total\_market\_value" REAL,
* "volume" REAL,
* "amount" REAL,
* "turnover\_rate" REAL,
* "leading\_stock\_name" TEXT,
* "leading\_stock\_change\_percent" REAL,
* "leading\_stock\_code" TEXT,
* "update\_time" TIMESTAMP,
* PRIMARY KEY("board\_code", "update\_time")
* );

c) 采集流程

* 读取数据库配置（路径等）从 config.py。
* 调用 AkShare stock\_board\_industry\_name\_em 获取数据。
* 批量写入/更新 industry\_board\_realtime\_quotes。
* 写入操作日志表 realtime\_collect\_operation\_logs，记录采集时间、状态、异常等。

# 获取沪深京A股最新行情，返回涨幅最多的前limit个股票 –改造

**后端**

1. [修改@backend\_api\stock\stock\_manage.py](mailto:修改@backend_api\stock\stock_manage.py), 请通读get\_quote\_
2. board函数现有代码处理业务逻辑，然后修改代码，输入输出参数不变，改成不管是否工作日，都从stock\_realtime\_quote数据库表中读取。

# 获取个股新闻信息数据

1. 后端新增 [@backend\_api\stock\， 新增个股新闻公告研报.py](mailto:修改@backend_api\stock\stock_manage.py)，py文件以英文命名。新增调用 AkShare框架的 stock\_news\_em接口的api ，获取东方财富指定个股的新闻资讯数据。

* 输入：symbol
* 输出：指定 symbol 当日最近 100 条新闻资讯数据，包括关键词、新闻标题、新闻内容、发布时间、文章来源、新闻链接等。

1. 数据库新增个股新闻公告表， 保存股票当日最近100条新闻资讯数据。
2. 前端 @frontend\stock.html, stock.js, 调用此后端api，显示个股对应新闻真实数据。

# 沪深京A股公告数据采集

1. @backend\_core\data\_collectors\akshare目录下，
   1. 参考realtime.py新建 realtime\_ stock\_notice\_report\_ak.py，定义A股公告历史数据采集类，api输入输出依据 @web https://akshare.akfamily.xyz/data/stock/stock.html#id179中stock\_notice\_report函数说明。 需要新增A股公告数据表，字段需要涵盖stock\_notice\_report返回的参数列表，以及更新日期等字段，

函数输入：“全部”，“指定日期”；

函数输出：代码、名称、公告标题、公告类型、公告日期、网址。

* 1. 采集数据写入A股公告数据表， 成功或失败写操作日志表realtime\_collect\_operation\_logs。数据库读取配置等信息参考realtime.py, 数据库路径请从 @/backend\_core\config\config.py取值。

1. @backend\_core\data\_collectors\目录下，main.py新增一个定时任务，引入上面的指数实时行情采集类。每日60分钟采集一次。

# 个股研报查询api实现

@stock\_news.py 新增api research\_reports，然后把async def get\_stock\_news\_combined函数中读取研报的代码逻辑，迁移到新的 research\_reports函数中，输入输出参考research\_reports。

@stock\_news.py 中research\_reports函数，读取个股研报信息后，写入个股研报信息表，数据库字段包括序号、股票代码、股票简称、报告名称、东财评级、机构、近一个月研报数、2024-盈利预测-收益、2024-盈利预测-市盈率、2025-盈利预测-收益、2025-盈利预测-市盈率、2026-盈利预测-收益、2026-盈利预测-市盈率、行业、日期、报告PDF链接。个股研报信息表需要新增。

# backend\_api数据库访问代码改造

参考backend\_core 目录，backend\_api 目录相关程序代码数据库访问切换到postgreSQL， 数据库访问架构都改为 sqlalchemy。

## 一、现状总结

1. **已有 SQLAlchemy 架构**

* backend\_api/database.py 已经实现了 SQLAlchemy 的 engine、SessionLocal、Base、get\_db 等标准结构，且数据库连接参数通过 config.py 配置，支持通过环境变量切换数据库类型。
* 绝大多数业务代码（如 user\_manage.py、watchlist\_manage.py、admin/quotes.py 等）已经采用 Session = Depends(get\_db) 方式，直接用 SQLAlchemy ORM 查询和操作。

1. **仍有直接使用 sqlite3 的代码**

主要集中在如下文件：

* backend\_api/market\_routes.py：直接用 sqlite3 查询行情数据。
* backend\_api/start.py：初始化数据库时直接用 sqlite3 创建文件。
* backend\_api/check\_db.py：直接用 sqlite3 检查表结构和数据。
* backend\_api/migrate\_db.py、backend\_api/create\_stock\_news\_table.py（如有）：直接用 sqlite3 操作表结构。

这些文件需要迁移为 SQLAlchemy 访问，并适配 PostgreSQL。

1. **数据库配置**

* backend\_api/config.py 通过 DATABASE\_URL 环境变量控制数据库类型，默认是 SQLite，需要改为 PostgreSQL 连接串（如：postgresql+psycopg2://user:password@host:port/dbname）。

## 二、改造分步方案

### 1. 数据库配置切换

* 修改 .env 或环境变量，设置 DATABASE\_URL 为 PostgreSQL 连接串。
* 确认 requirements.txt 已包含 psycopg2（PostgreSQL 驱动）。

### 2. 统一数据库访问为 SQLAlchemy

#### 需要重点重构的文件：

* backend\_api/market\_routes.py
* 将所有 sqlite3.connect、cursor、execute、fetch\* 相关代码，改为 SQLAlchemy 查询。
* 新建/补充对应的 ORM Model（如 IndexRealtimeQuotes、IndustryBoardRealtimeQuotes），并用 db.query(...).filter(...).order\_by(...).all() 方式获取数据。
* backend\_api/start.py
* 数据库初始化部分，直接调用 from backend\_api.database import init\_db; init\_db()，不再用 sqlite3 创建文件。
* backend\_api/check\_db.py
* 检查表结构和数据部分，改为 SQLAlchemy 查询元数据和数据。
* backend\_api/migrate\_db.py、backend\_api/create\_stock\_news\_table.py
* 所有表结构变更、建表、索引等操作，改为 SQLAlchemy 的 Base.metadata.create\_all() 或 Alembic 迁移。

#### 其余业务代码

* 只要是 Session = Depends(get\_db) 并用 db.query(...) 的，基本无需大改，只需确保 Model 兼容 PostgreSQL。

### 3. ORM Model 适配

* 检查 backend\_api/models.py，所有表的字段类型、主键自增、时间戳等，需兼容 PostgreSQL（如 autoincrement=True、server\_default 等）。
* 特别注意 SQLite 和 PostgreSQL 在自增主键、时间类型、唯一索引等方面的差异。

## 三、具体重构建议

### 1. 修改数据库连接为 PostgreSQL

.env 示例：

text

Apply to work\_log.md



DATABASE\_URL=postgresql+psycopg2://youruser:yourpassword@localhost:5432/yourdb

requirements.txt 增加：

text

Apply to work\_log.md



psycopg2-binary>=2.8

### 2. 以 market\_routes.py 为例的重构思路

* 新建 ORM Model（如未定义）：

python

Apply to work\_log.md



*# models.py*

  class IndexRealtimeQuotes(Base):

      \_\_tablename\_\_ = "index\_realtime\_quotes"

      id = Column(Integer, *primary\_key*=True, *autoincrement*=True)

      code = Column(String)

      name = Column(String)

      price = Column(Float)

      change = Column(Float)

      pct\_chg = Column(Float)

      volume = Column(Float)

      update\_time = Column(DateTime)

*# ... 其他字段*

* 查询替换：

python

Apply to work\_log.md



*from* backend\_api.models *import* IndexRealtimeQuotes

*from* sqlalchemy.orm *import* Session

  @router.get("/indices")

  def get\_market\_indices(*db*: Session = Depends(get\_db)):

      index\_codes = {...}

      indices\_data = []

*for* name, code *in* index\_codes.items():

          row = db.query(IndexRealtimeQuotes).filter(IndexRealtimeQuotes.code == code).order\_by(IndexRealtimeQuotes.update\_time.desc()).first()

*if* row:

*# 转换为 dict 并处理字段*

              indices\_data.append({...})

*return* JSONResponse({'success': True, 'data': indices\_data})

* 其他所有 sqlite3 相关代码，均用 SQLAlchemy ORM 查询/插入/更新/删除替换。

## 四、后续步骤

1. 先完成配置和 Model 适配，确保能连接 PostgreSQL 并自动建表。
2. 逐步替换所有 sqlite3 相关代码为 SQLAlchemy ORM。
3. 测试所有接口，确保数据读写正常。
4. 如有历史数据迁移需求，可写脚本将 SQLite 数据导入 PostgreSQL。