数据库接口使用文档

221275010 屈航

本数据库所提供的核心函数大致分为以下五类：**数据库交互接口函数、数据库和表建立文件(sql文件)、外部数据获取函数、外部数据存储函数、临时数据存储文件(json文件)**。对于前后端直接调用的函数是**数据库交互接口函数**，而其他的函数主要是数据库实现的细节函数，一般情况下前后端不进行调用。

1 数据库交互接口函数

数据库交互接口函数都位于`xxx_base.py`文件当中，具体位于以下四个文件当中：users_base.py、stock_base.py、market_base.py、history_base.py。四个函数中提供有各自作用的函数，分别是与用户信息配置和查找相关、与用户股票交易等操作行为相关、获取市场大盘指数实时数据相关、获取股票和大盘历史数据相关。

1.1 users_base.py文件

该文件当中提供的交互接口的函数与用户信息配置和查找相关，具体如下：

1. **user_register()函数**

```
#* DONE index=1 增：用户注册，增添用户信息在用户基础信息表中(user_trade.users)

def user_register(name:str = '李华', password:str = '111111', question:str = 'are you ok?',
answer:str = 'ok', say:int = 0) -> tuple[bool, str]:
    '增添用户信息在用户基础信息表中，name 应为唯一的标识，录入成功返回True，若不允许将该组信息录入，返回值将是False'

// 补充：
// 函数形参含义：'name'是用户名字，'password'是用户密码，'question'是用户密保问题，'answer'是用户密码答案，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收该函数的运行的成功与否的'Bool值'，第二个变量接收函数运行的报错信息或Successfully的字符串变量。
```

2. **user_search()函数**

```
#* DONE index = 2 查：用户正常登录，要根据用户名进行数据查询，查询用户的用户名是否存在，如果存在返回包括密码在内的数据(user_trade.users)

def user_search(name:str = '李华', password:str = '111111', say:int = 0) -> tuple[dict,
str]:
    "根据用户名进行数据查询，查询用户的用户名是否存在，如果存在返回包括密码在内的数据在tuple[dict, str]里，若不存在则返回tuple[none, str]，dict = {'id', 'name', 'password', 'question', 'answer', 'money'}"

// 补充：
// 函数形参含义：'name'是用户名字，'password'是用户密码，'question'是用户密保问题，'answer'是用户密码答案，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收对用户信息查询的'字典dict'，第二个变量接收函数运行的报错信息或Successfully的字符串变量。
// 字典字段含义：'id'--编号，'name'--用户名，'password'--用户密码，'question'--密保问题，'answer'--密保答案，'money'--账户资产。
```

3. **user_modify()函数**

```
#* DONE index = 3 改：用户登录之后，可以进行密码修改、密保问题和答案修改(user_trade.users)

def user_modify(name:str = '李华', old_password:str = '111111' ,new_password:str = '222222',
question:str = 'are you ok?', answer:str = 'ok', say:int = 0) -> tuple[bool, str]:
    '用户登录之后，可以进行密码修改、密保问题和答案修改，修改成功会返回True，修改不成功则返回
False'

// 补充：
// 函数形参含义：'name'是用户名字，'password'是用户密码，'question'是用户密保问题，'answer'是用
户密码答案，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收该函数的运行的成功与否的'Bool值'，第二个变
量接收函数运行的报错信息或Successfully的字符串变量。
```

4. user_delete()函数

```
#* DONE index = 4 删：用户登录之后，可以选择注销用户，根据 key = name删除(user_trade.users以及
其他关联表)

def user_delete(name:str = '李华', password:str = '111111', say:int = 0) -> tuple[bool,
str]:
    '用户登录之后，可以选择注销用户，根据 key = name删除(user_trade.users以及其他关联表)，删除
成功则返回True，失败则返回Fasle'

// 补充：
// 函数形参含义：'name'是用户名字，'password'是用户密码，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收该函数的运行的成功与否的'Bool值'，第二个变
量接收函数运行的报错信息或Successfully的字符串变量。
```

5. user_search_by_name()函数

```
#* DONE index = 5 查：  用户要找回密码，此时只根据用户的name则返回所有的用户基本数据

def user_search_by_name(name:str = '李华', say:int = 0) -> tuple[dict, str]:
    "用户要找回密码，根据用户名进行数据查询，查询用户的用户名是否存在，如果存在返回包括密码在内的
数据在tuple[dict, str]里，若不存在则返回tuple[none, str], dict = {'id', 'name', 'password',
'question', 'answer', 'money'}"

// 补充：
// 函数形参含义：'name'是用户名字，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收对用户信息查询的'字典dict'，第二个变量接收
函数运行的报错信息或Successfully的字符串变量。
// 字典字段含义：'id'--编号，'name'--用户名，'password'--用户密码，'question'--密保问题，
'answer'--密保答案，'money'--账户资产。
```

6. user_money_modify()函数

```
#* DONE index = 6: 改 实现用户的充值 账户的钱数 += money

def user_money_modify(name:str = '李华', password:str = '111111', money:float = 0,
decrease:int = 0,say:int = 0) -> tuple[bool, str]:
    '用户登录之后，可以实现用户的充值，账户的钱数 += money，修改成功会返回True，修改不成功则返回
False'

// 补充：
// 函数参数含义：'name'是用户名字，'password'是用户密码，'money'是调整的资产数额，'decrease'是
资产减少状态位，默认为0，即资产增加，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收该函数的运行的成功与否的'Bool值'，第二个变
量接收函数运行的报错信息或Successfully的字符串变量。
```

1.2 stock_base.py文件

该文件当中提供的交互接口的函数与股票交易等操作行为相关，具体如下：

1. **股票实时数据存储字段含义表**：

```
CREATE TABLE current_stock( -- 当前的股票的数据，显示当前每一只股票的价格
    id INT PRIMARY KEY auto_increment,
    stock_code VARCHAR(500), -- 股票的编号
    stock_name VARCHAR(500), -- 股票的名称
    exchange VARCHAR(500), -- 交易所的名称 上证，深证
    price FLOAT, -- 最新价格（元）p
    up_down FLOAT, -- 涨跌幅（%）
    price_range FLOAT, -- 涨跌额（元）
    trade_num FLOAT, -- 成交量（手）
    trade_money FLOAT, -- 成交额（元）
    amplitude FLOAT, -- 振幅（%）
    high_price FLOAT, -- 最高价（元）
    low_price FLOAT, -- 最低价（元）
    today_open FLOAT, -- 今开 （元）
    yester_price FLOAT, -- 昨日收盘价（元）
    quantity_ratio FLOAT, -- 量比（%）
    turnover_ratio FLOAT, -- 换手（%）
    pe_ratio FLOAT, -- 市盈率(%) 总市值除以预估全年净利润
    pb_ratio FLOAT, --  市净率
    total_value FLOAT, -- 总市值（元）
    circle_value FLOAT, -- 流通市值（元）
    increase_ratio FLOAT, -- 涨速（%）
    five_up_down FLOAT, -- 5min 涨跌
    zdf60 FLOAT, -- 60日涨跌幅（%）
    zdfnc FLOAT, -- 年初至今涨跌幅（%）
    UNIQUE(stock_code),
    UNIQUE(stock_name)
);
```

2. **stock_search_by_code()函数**

```
#* DONE index = 1 根据stock_code进行数据查询股票实时数据
def stock_search_by_code(stock_code:str = 'sh600089', say:int = 0) -> tuple[dict, str]:
    "根据stock_code进行数据查询，stock_code = sh600089 或者600089均可，查询该股票实时数据，如果存在返回该股票的数据tuple[dict, str]里，若不存在则返回tuple[none, str]，dict = {'id', 'stock_code', 'stock_name', 'exchange', 'price', 'up_down', 'price_range', 'trade_num', 'trade_money', 'amplitude', 'high_price', 'low_price', 'today_open', 'yester_price', 'quantity_ratio', 'turnover_ratio', 'pe_ratio', 'pb_ratio', 'total_value', 'circle_value', 'increase_ratio', 'five_up_down', 'zdf60', 'dfnc'}"

// 补充：
// 函数形参含义：'stock_code'股票代码，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收股票数据的'字典dict'，第二个变量接收函数运行的报错信息或Successfully的字符串变量。
```

3. **stock_search_by_name()函数**

```
#* DONE index = 2 根据stock_name进行数据查询股票实时数据
def stock_search_by_name(stock_name:str = '贵州茅台', say:int = 0) -> tuple[dict, str]:
    "根据stock_name进行数据查询，查询该股票数据，如果存在返回该股票的实时数据tuple[dict, str]
里，若不存在则返回tuple[none, str],dict = {'id', 'stock_code', 'stock_name', 'exchange',
'price', 'up_down', 'price_range', 'trade_num', 'trade_money', 'amplitude', 'high_price',
'low_price', 'today_open', 'yester_price', 'quantity_ratio', 'turnover_ratio', 'pe_ratio',
'pb_ratio', 'total_value', 'circle_value', 'increase_ratio', 'five_up_down', 'zdf60',
'dfnc'}"

// 补充：
// 函数形参含义：'stock_name'股票名称，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收股票数据的'字典dict'，第二个变量接收函数运
行的报错信息或Successfully的字符串变量。
```

4. stock_search_all()函数

```
#* DONE index = 3 获取数据库中全部的stock数据，以列表形式返回大约5000只股票数据
def stock_search_all(say:int = 0) -> list:
    '获取数据库中全部的stock的实时数据，以列表形式返回大约5000只股票数据'

// 补充：
// 函数形参含义：'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收所有股票数据的'list'，第二个变量接收函数运
行的报错信息或Successfully的字符串变量。
```

5. user_stock_fetch()函数

```
#* DONE index = 4 查看用户的持仓信息
def user_stock_fetch(name:str = '李华',say:int = 0) -> tuple[list[dict], str]:
    "查看用户的持仓信息,dict = {'id', 'name', 'stock_code', 'stock_name', 'num'}"

// 补充：
// 函数形参含义：'name'用户名字，'say'为函数调试信息打印状态位。
// 函数返回值含义：一般使用两个变量接收，第一个变量接收某一用户的所有的持仓股票列表'list',元素是
每一个股票的数据'字典dict'，第二个变量接收函数运行的报错信息或Successfully的字符串变量。
```

6. user_stock_modify()函数

```
#* DONE index = 5 修改用户的持仓信息
def user_stock_modify(name:str = '李华', stock_code:str = 'sh600089', stock_name:str = '特变
电工', num:int = 0, decrease:int = 0,say:int = 0) -> tuple[bool, str]:
    '修改用户的持仓信息，只需要用户名，股票代码，股票名称，交易数量，decrease默认为0，若置位为1则
是实现该股票的持有减少'

//补充：
//函数形参含义：'num'是所持股数，'decrease'是持仓记录删除状态位，默认为0，即持仓记录插入。
```

7. user_stock_clear()函数

```
#* DONE index = 6 DONE 清除持仓记录中num = 0的股票持有记录
def user_stock_clear() -> None:
'清除持仓记录中num = 0的股票持有记录'
```

8. user_choose_modify()函数

```
#* DONE index = 7  修改用户的自选个股信息
def user_choose_modify(name:str = '李华', stock_code:str = 'sh600089', stock_name:str = '特变电工', decrease:bool = 0,say:int = 0) -> tuple[bool, str]:
    '修改用户的自选个股信息，只需要用户名，股票代码，股票名称decrease默认为0即收藏股票，若置位为1则是实现该股票收藏的删除'

//补充:
//函数形参含义：'decrease'是自选个股记录删除状态位，默认为0，即自选个股记录插入。
```

9. **user_stock_record()函数**

```
#* DONE index = 8 实现用户的交易记录的存储
def user_stock_record(name:str = '李华', stock_code:str = 'sh600089', stock_name:str = '特变电工', num:int = 0, price:float = 0.0, time:str = '' ,decrease:bool = 0, say:int = 0) ->tuple[bool, str]:
    '实现用户的交易记录的存储，注意num必须要非负数，decrease默认是0，表示买入，decrease如果是1则是卖出'

// 补充:
// 函数形参含义：'num'是交易股数且'num'大于等于0，'price'是当时的交易价格，'time'是交易时间，'decrease'是股票买入卖出状态位，默认为0，即股票的买入。
```

10. **user_choose_fetch()函数**

```
#* DONE index = 9 查看用户的自选个股信息
def user_choose_fetch(name:str = '李华',say:int = 0) -> tuple[list[dict], str]:
    "查看用户的自选个股信息,dict = {'id', 'name', 'stock_code', 'stock_name'}"

//补充:
//函数返回值含义：返回一个自选个股的列表'list'，列表的元素则是每一个自选个股的信息字典'dict'。
```

11. **user_record_fetch()函数**

```
#* DONE index = 10 查看用户的交易记录信息
def user_record_fetch(name:str = '李华',say:int = 0) -> tuple[list[dict], str]:
    "查看用户的自选个股信息,dict = {'id', 'name', 'stock_code', 'stock_name', 'num', 'price', 'time'}, num 是正数代表是买入，负数代表卖出"

//补充:
//返回值含义：返回值是一个交易记录列表'list'，列表元素是每一次交易的具体信息字典'dict'。
```

1.3 market_base.py文件

该文件当中提供的接口函数与获取市场大盘指数实时数据相关

。

1. **大盘指数实时数据存储字段含义表**

```
CREATE TABLE market(
    id INT PRIMARY KEY auto_increment,
    market_code VARCHAR(500), -- 上证指数的代码
    market_name VARCHAR(500), -- 上证指数的名称
    price FLOAT, -- 现价
    high_price FLOAT, -- 最高价（元）
    low_price FLOAT, -- 最低价（元）
    price_fluctuation FLOAT, -- 涨跌幅（%）
    amplitude FLOAT, -- 振幅（%）
```

```
    yesterday_end FLOAT, -- 昨收
    today_open FLOAT, -- 今开
    trade_money FLOAT, -- 成交额（元）
    high52 FLOAT, -- 52周最高
    low52 FLOAT, -- 52周最低
    UNIQUE(market_code)
);
```

2. **market_search_by_code()函数**

```
#* DONE index = 1 根据market_code进行数据查询股票实时数据
def market_search_by_code(market_code:str = 'sh000001', say:int = 0) -> tuple[dict, str]:
    "根据market_code（必须有市场标识）进行大盘指数数据查询，market_code = sh000001，查询该大盘指数实时数据，如果存在返回该指数的数据tuple[dict，str]里，若不存在则返回tuple[none，str],dict = {'id', 'market_code', 'market_name', 'price', 'high_price', 'low_price', 'price_fluctuation', 'amplitude', 'yesterday_end', 'today_open', 'trade_money', 'high52', 'low52'}"

// 补充：
// 形参含义补充：'market_code'必须要有市场标识。
```

3. **market_search_by_name()函数**

```
#* DONE index = 2 根据market_name进行数据查询股票实时数据
def market_search_by_name(market_name:str = '上证指数', say:int = 0) -> tuple[dict, str]:
    "根据market_name（上证指数、深证成指、北证50、沪深300）进行数据查询，查询该大盘指数数据，如果存在返回该指数的实时数据tuple[dict，str]里，若不存在则返回tuple[none，str]，dict = {'id', 'market_code', 'market_name', 'price', 'high_price', 'low_price', 'price_fluctuation', 'amplitude', 'yesterday_end', 'today_open', 'trade_money', 'high52', 'low52'}"

// 补充：
// 形参含义补充：'market_name'的选项为上证指数、深证成指、北证50、沪深300。
```

4. **market_search_all()函数**

```
#* DONE index = 3 获取数据库中全部的market数据，以列表形式返回4个大盘数据
def market_search_all(say:int = 0) -> list:
    '获取数据库中全部的market的实时数据，以列表形式返回4个大盘数据'
```

1.4 history_base.py文件

1. **股票和大盘指数历史数据的存储字段含义表**

股票历史数据结构

```
CREATE TABLE sh600519(
    id INT PRIMARY KEY auto_increment,
    time VARCHAR(100), -- 交易时间
    stock_code VARCHAR(500), -- 股票的代码
    stock_name VARCHAR(500), -- 股票的名称
    exchange VARCHAR(10), -- 交易所标识
    open_price FLOAT, -- 开盘价（元）
    close_price FLOAT, -- 收盘价（元）
    high_price FLOAT, -- 最高价（元）
    low_price FLOAT, -- 最低价（元）
    trade_num FLOAT, -- 成交量（手）
    trade_money FLOAT, -- 成交额（元）
    amplitude FLOAT, -- 振幅（%）
    up_down FLOAT, -- 涨跌幅（%）
```

```
    price_range FLOAT, -- 涨跌额（元）
    turnover_ratio FLOAT, -- 换手率
    UNIQUE(time)
);
```

大盘指数历史数据结构

```
CREATE TABLE sh000001(
    id INT PRIMARY KEY auto_increment,
    time VARCHAR(100), -- 交易时间
    market_code VARCHAR(500), -- 指数的代码
    market_name VARCHAR(500), -- 指数的名称
    exchange VARCHAR(10), -- 交易所标识
    open_price FLOAT, -- 开盘价（元）
    close_price FLOAT, -- 收盘价（元）
    high_price FLOAT, -- 最高价（元）
    low_price FLOAT, -- 最低价（元）
    trade_num FLOAT, -- 成交量（手）
    trade_money FLOAT, -- 成交额（元）
    amplitude FLOAT, -- 振幅（%）
    up_down FLOAT, -- 涨跌幅（%）
    price_range FLOAT, -- 涨跌额（元）
    turnover_ratio FLOAT, -- 换手率
    UNIQUE(time)
);
```

2. history_stock_search()函数

```
#* DONE index = 1 获取股票的历史数据
def history_stock_search(stock_code:str = 'sh600089', say:int = 0)->tuple[list[dict], str]:
    "获取股票的历史数据，传入参数股票代码stock_code，例如sh600089或者600089，返回历史数据,dict = {'id', 'time', 'market_code', 'market_name', 'exchange', 'open_price', 'close_price', 'high_price', 'low_price', 'trade_num', 'trade_money', 'amplitude', 'up_down', 'price_range', 'turnover_ratio'}"

// 补充：
// 函数形参含义：'market_code'的可以带市场标识也可以不带市场标识，带市场标识则函数运行的速度更快。
// 返回值含义：返回一个历史记录列表'list'，列表元素是字典'dict'，字典记录的是该次历史数据的具体记录。
```

3. history_market_search()函数

```
#* DONE index = 2 获取大盘指数的历史数据
def history_market_search(market_code:str = 'sh000001', say:int = 0)->tuple[list[dict], str]:
    "获取大盘指数的历史数据，传入大盘指数代码market_code，sh000001、sz399001、bj899050、sh000300，必须有市场标识如sh、sz、bj，返回历史数据,dict = {'id', 'time', 'market_code', 'market_name', 'exchange', 'open_price', 'close_price', 'high_price', 'low_price', 'trade_num', 'trade_money', 'amplitude', 'up_down', 'price_range', 'turnover_ratio'}"

// 补充：
// 函数形参含义：market_code的可选值为sh000001、sz399001、bj899050、sh000300
```

2 数据库和表建立文件(sql文件)

建议在运行之前先讲这个sql里面的数据库和表都先建好，否则接口函数可能找不到数据库或者表。

`trade.sql`中构造了数据库`user_trade`，以及用户的配置表，股票实时数据表，用户持仓表，用户自选个股表，用户交易记录表等，各个表之间含有**外键级联**。

`stock_history.sql`中构造了数据库`stock_history`，主要构造了一个股票实时数据的表结构示意，而在该数据库中则存储有5000多个存储有股票的历史数据的表，也就是一共5000多个表。

`market.sql`中构造了数据库`market`，主要构造了类似上面股票实时数据表，实现了大盘数据表。

`market_history.sql`中构造了数据库`market_history`，这个数据库中为每一个大盘数据存储了一个表，这个sql文件里面展示有表的结构。

3 外部数据获取函数

这一部分函数主要是通过外部库去获取原始的股票数据，以及对股票数据进行进一步的处理和存储在json文件当中。这一部分函数配合外部数据存储函数实现数据库的股票数据存储更新。

外部数据获取函数分别位于文件：`stock_now_fetch.py`,`stock_history.py`,`market.py`,`market_history.py`。

主要有以下几个函数：

```
#* DONE index = 1沪交所的股票数据
def fetch_sh_stock_now(say:int = 0) ->tuple[bool, str]:
    '沪交所的股票数据并存入stock_sh_now.json'

#* DONE index = 2深交所的股票数据
def fetch_sz_stock_now(say:int = 0)->tuple[bool, str]:
    '深交所的股票数据并存入stock_sz_now.json'

#* DONE index = 3 获得大盘数据的实时数据
def market_now(say:int = 0) -> tuple[bool,str]:
    '获得大盘数据的实时数据'

#* DONE index = 4 获取symbol的历史数据并存入文件
def stock_history_fetch(symbol:str = "600519", period="daily" ,start_date:str = "20220101",
end_date:str = "20240531",say:int = 0) -> tuple[bool,str]:
    "获取symbol的历史数据并存入文件"

#* DONE index = 5 大盘指数的历史指数的获取
def market_history_fetch(symbol:str = "000300", period="daily" ,start_date:str = "20220101",
end_date:str = "20240531",say:int = 0) -> tuple[bool,str]:
    "获取大盘指数的历史数据并存入文件"
```

4 外部数据存储函数

外部数据存储函数主要负责将通过外部库抓取到的股票和大盘指数数据进行进一步的处理，并将数据存入数据库中，同时也完成更新。

外部数据存储函数分别位于文件：
`stock_zh_now_update.py`,`stock_history.py`,`market.py`,`market_history.py`。

主要有以下几个函数：

```python
#* DONE index = 1 更新数据库当中的上海、深圳证券交易所的股票实时数据
def update_stock_sh_now(say:int = 0) -> tuple[bool, str]:
    '用于更新上海证券交易所 和 深圳证券交易所 股票的实时数据'

#* DONE index = 2 将sh,sz的股票历史数据存入数据库
def stock_history_save(say:int = 0)->tuple[bool, str]:
    '将sh,sz的股票历史数据存入数据库'

#* DONE index = 3 将大盘数据的实时数据处理后存入数据库
def market_now_save() -> tuple[bool, str]:
    '将大盘数据的实时数据处理后存入数据库'

#* DONE index = 4 大盘指数的历史指数的处理和存入数据库
def market_history_save(say:int = 0)->tuple[bool, str]:
    '将sh,sz的股票历史数据存入数据库'
```

5 临时数据存储文件（json文件）

主要包括一些json文件，这些文件大多都是通过外部库爬取的股票和大盘指数的实时数据和历史数据，因为数据量庞大，暂存与json文件当中，以便于进行数据处理和表的生成。

主要有以下几个json文件：
`stock_sz_now.json`，`stock_sh_now.json`，`market_now.json`，`market_history.json`。

以上就是该数据库接口文档的全部内容，望对前后端的实现有所帮助。