系统设计

目录

1 文档介绍

1.1 文档目的

通过该文档来记录开发过程。该项目可能会持续相当长的一段时间，为避免每过一段时间就需要将整个项目梳理一遍而浪费大量时间，使用该文档来记录每一个阶段的开发内容，并拟定好下一步计划的内容。在此之上，记录自己的思路，逐渐改善内容。

1.2 项目背景

为了能够平稳走在自己理想的道路上，提高投资水平是必不可少的能力，并且由于自身有一定的计算机基础，所以利用计算机技术来为自己的投资提供更好的支持，在提高效率的同时提高自身专业水平，对工作也具有很大的好处。

1.3 需求概述

通过网站的形式提供方便的股票筛选、浏览等功能，并且逐渐开发量化策略来进行选股、选基。目标是能够实现较稳定并且年化收益率在 10% 以上

1.4 读者对象

个人

1.5 参考文档

暂无

2 系统结构环境

2.1 系统结构

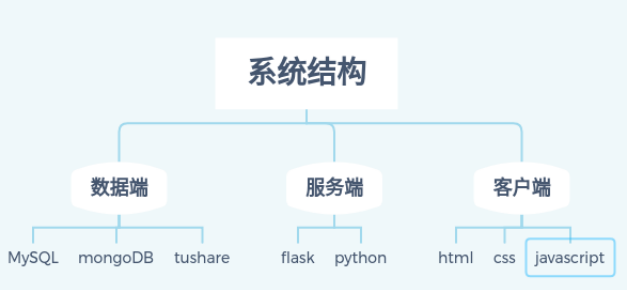

图 1: 系统结构

图中描述了该系统的主要结构，主要分为 3 层：数据端、服务端以及客户端。其中数据端负责存储数据，用到的数据库有 MySQL、mongoDB 以及金融数据接口 tushare；服务端用于响应客户端的各种请求，根据不同的请求从数据库中查询数据并进行处理，最后返回到客户端；客户端则负责实现用户界面、交互功能并可以将从服务端获得的数据进行适当方式的展示。

2.2 系统环境

2.2.1 数据端

MySQL	5.7.33
mongoDB	4.4.1
tushare	1.2.48

2.2.2 服务端

flask	1.0.2
python	3.6

3 功能设计

3.1 回测功能

回测功能作为量化选股、投资的基础部分，是长期开发的一个非常重要的部分，在此之上，为了提高自己的专业水平，决定自主开发一个加测功能，并在之后的过程中不断改进、完善。

该平台回测功能的开发参考了聚宽平台回测部分的一些类定义作为该平台的起步，之后的内容则根据个人的理解及思考。

目前实现的回测功能是简化的版本，没有考虑排队逐手交易、交易手续费等细节内容，下单即交易完成。主要的文件结构及其主要内容如下。

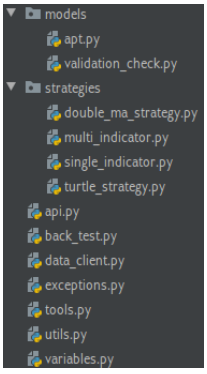

图 2: 文件结构

由图中可以看出，主要分为 3 个部分，分别为模型部分、策略部分以及基础的回测实现部分。接下来将会对这些文件分别对这些文件进行说明。

3.1.1 utils.py

该文件主要用于定义回测过程中需要用到的类，将交易过程分为不同的部分，使程序更加便于阅读和更改。

1) TradeLog: 该类用于记录交易过程信息，包括买入、卖出等内容，并可以导出为文件。交易记录有两种形式，一种为自然语言方式，即每笔交易都会组装成一句话；另一种为存储为 dataframe 实例。

   该类的成员方法有 add_log()、save_log()，分别用于向成员变量中添加记录以及生成文件。

图 3: TradeLog 类

2) GlobalVariable：该类用于存储回测过程中的全局变量，比如股池、持仓数、调仓周期等

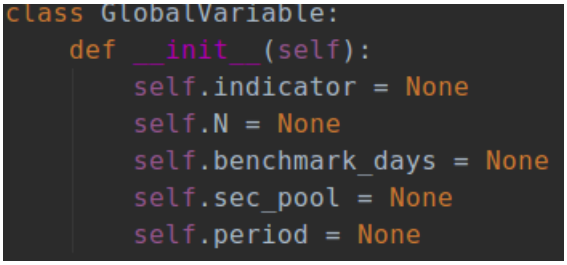

图 4: GlobalVariable 类

3) Position：持仓类，该类的每一个实例代表一支股票的持仓，包括股票代码、持仓数量、可用数量、持仓成本、持仓市值等变量。该类提供 trade()、daily_update() 以及 clear() 成员函数。

其中 trade() 方法用于实现单支股票的交易操作，包括买入、卖出操作并实现交易后相应变量值的修改，比如持仓成本、持仓市值等；daily_upate() 方法则用于每天收盘后持仓状态的变更，主要更新持仓可用数量，并以收盘价来更新持仓市值、浮动收益等；clear() 方法用于实现清空单支股票的持仓操作。

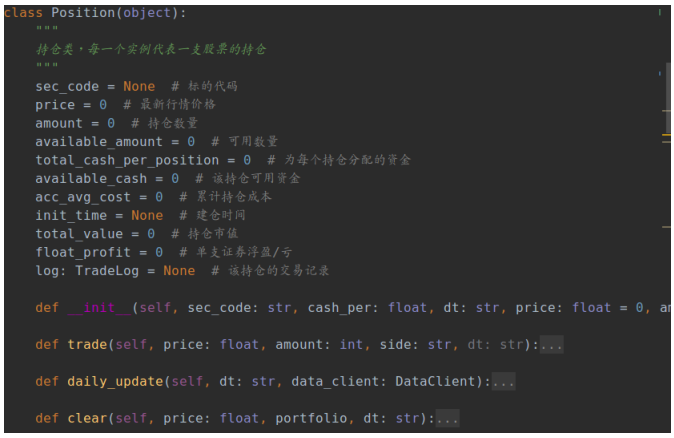

图 5: Position 类

4) Portfolio：资金账户类，该类用于模拟开户后的资金账户，包含的成员变量主要有累计出入金、可用资金、可取资金、持仓总市值、总资产等内容，该类提供的方法有 transfer()、trade()、daily_update()。

其中 transfer() 主要实现的是资金转入转出操作，并且更新相关的变量值；trade() 方法与持仓类的方法不同，该方法只的操作只针对资金变量，比如可用资金的更新；daily_update() 方法同样是针对资金变量的更新，其中主要是将可用资金转换为可取资金。

```
class Portfolio(object):
    """
    资金账户
    """
    inout_cash = 0  # 累计出入金，转入转出账户的累计资金
    available_cash = 0  # 可用资金，可以用来购买证券的资金
    transferable_cash = 0  # 可取资金，可以提现的资金
    # locked_cash = None  # 挂单锁住的资金
    sec_value = 0  # 持仓市值
    # total_cash = 0  # 总资金
    total_asset = 0  # 总资产
    profit = 0  # 收益
    log: TradeLog = None  # 交易记录
    log_path = None  # 日志路径

    def __init__(self, first_in: float, **kwargs):...

    def transfer(self, amount: float, side: str):...

    def trade(self, sec_code: str, price: float, amount: int, side: str, dt: str)

    def daily_update(self, positions: List[Position]):...

    def __del__(self):...
```

图 6: Portfolio 类

5) Metrics：指标类，该类主要用于保存回测的各种指标，比如收益、收益率、年化收益率、基准收益率、策略的 alpha、beta 值、最大回撤、夏普比率等。该类提供的方法主要为记录指标的辅助作用以及计算需要处理的指标值，有 set_basic_profit_rate()、add_profit()、cal_sharpe()、cal_max_drawdown()、period_profit_rate() 方法。

```
class Metrics(object):
    """
    需要计算的各个指标
    """
    float_profit = []  # 浮盈/浮亏
    float_profit_rate = []  # 浮盈/亏率
    trade_date = []
    basic_index = None  # 目前还未用到
    principal = None  # 本金
    anni_profit_rate = 0  # 年化收益率
    basic_profit_rate = []  # 基准收益率
    alpha = None  # Alpha值
    beta = None  # Beta值
    max_drawdown_rate = 0  # 最大回撤率
    sharpe_ratio = 0  # 夏普比率 无风险利率选一年期定期存款利率1.5%

    def __init__(self, principal, basic_index):...

    def set_basic_profit_rate(self, index_code: str, start_dt: str, end_dt: str, fre

    def add_profit(self, profit: float, dt: str):...

    def cal_sharpe(self):...

    def cal_max_drawdown(self):...

    def period_profit_rate(self):...
```

图 7: Metrics 类

6) AccountInfo：账户类，该类用于模拟开户后获得的账户，主要包括 3 个大部分，资金账户、持仓以及指标，其中持仓为一个数组，用于放置多支股票的持仓，分别为相应类的实例，除此之外，还有一些辅助性质变量，比如当前日期、上一交易日、运行的参数以及用于获取数据的接口。该类提供一些设置成员变量值的方法以及一些综合性的方法，主要有 get_sec_amount() 来获取指定证券的持仓数量，has_position() 来判断是否有指定证券的持仓并返回相应的持仓实例，daily_update() 方法分别调用持仓和资金账户的相应方法来更新整个账户，clear_position() 方法通过调用各个持仓的相应方法来清空所有的持仓。

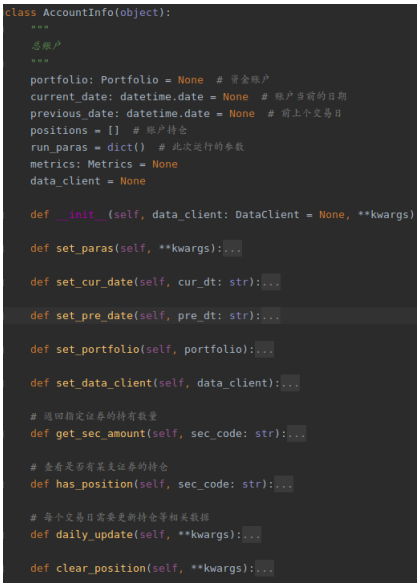

图 8: AccountInfo 类

7) Strategy：策略基类，该类作为一个基类，实现了基本的回测功能，所有具体的策略类都会继承自该类也因此能够进行回测。该基类的成员变量包括回测的对象即一个账户类 AccountInfo 的实例、回测的全局变量类 GlobalVariable 的实例、数据接口类的实例等。该类的成员方法主要是回测功能的各个组件，参照聚宽平台的回测功能，该类列出了一些回测中将来可能会调用的方法，但暂未实现。

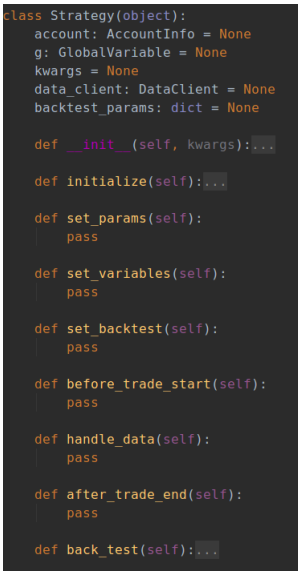

图 9: Strategy 类

该基类的 back_test() 方法实现了一个简单的回测功能，该方法中，首先需要确定回测的开始和结束时间，之后需要确定开始日期前的最后一个交易日，之后便开始进行循环。

每次循环中，首先需要确定当天是否为交易日，之后会调用开盘前的处理成员方法，开盘后，如果当天是调仓日则调用处理数据的方法完成交易，如果不是，则直接进入收盘阶段，收盘阶段则

负责完成账户的每日更新并且调用收盘后的处理方法，将调仓计数器加 1 后则表示完成一天的工作。之后，需要更新日期，包括当前日期以及上一个交易日日期。

8) BasicInfo：基础信息类，该类主要保存证券的基础信息，包括股票列表、指数列表、基金列表以及交易日历等，这些数据都是在运行期间可能会多次调用的信息，通过将这些信息保存下来供多次使用从而避免多次从数据库中查询相应的数据。

该类当前实现了词语翻译的方法，主要作用是将一些中文术语转换为相应的系统通用的标识符，以便于数据查询等。

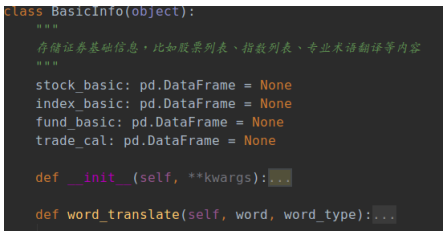

图 10: BasicInfo 类

3.1.2 data_client.py

该文件用于定义整个系统的数据接口，用于从数据库中获取数据，并在此基础上，将数据加工成需要的内容及格式。目前用到的数据库有 MySQL 以及 mongoD，其中 MySQL 主要用于存储证券基本信息比如股票列表、指数列表等、指数 k 线、个股 k 线、资金流数据等，mongoDB 主要用于储存股票的财务指标比如利润表、资产负债表数据等，除了本地数据外，该系统还用到了外部的数据接口，主要为 tushare 金融数据，通过该接口可以获得各种金融数据，比如 k 线、财务数据、基金数据等，本地的大部分数据也是取自于 tushare，为了便于查询以及减少 tushare 宽带资源的使用，将需要的数据存入本地数据库。

该文件中定义了 DataClient 类，该类将目前用到的数据库接口集中到同一个类中，以便提高该类的适用性。其中连接 MySQL 使用的是 MySQLdb 第三方库，通过建立连接并获取游标 cursor 便可以执行 sql 语句来进行相应的操作；mongoDB 的操作则使用 pymongo 第三方库，利用该库中的 MongoClient() 创建客户端后便可以查询数据；tushare 数据则是使用官方提供的接口来获取数据。

该类提供成员方法按功能大概分为 4 个部分：提供基础信息的成员方法、k 线相关成员方法、财务数据相关成员方法以及用于条件判断的相关成员方法。

1) 基础信息部分。该部分主要用于提供证券相关的基本信息，该部分当前包括的成员方法有 stock_basic()、index_basic()、index_weight()、get_pre_trade_dt()、get_back_trade_dt()、init_start_trade_date()、get_sec_pool()。

stock_basic() 成员方法参考 tushare 同名接口来从本地数据库中查询股票列表，该方法提供一些参数来筛选相应条件的股票；

index_basic() 成员方法参考 tushare 同名接口来从本地数据库中查询指数列表，该方法提供一些参数来筛选相应条件的指数；

index_weight() 成员方法用来查询相应指数的成分股。该方法通过调用 tushare 的同名方法来获取成分股，其中 tushare 的功能是获取一个时间段内的指数成分股，而如果获得一个时间点的成分股，就需要在原接口的基础上选出离目的时间点最近一次成分股。

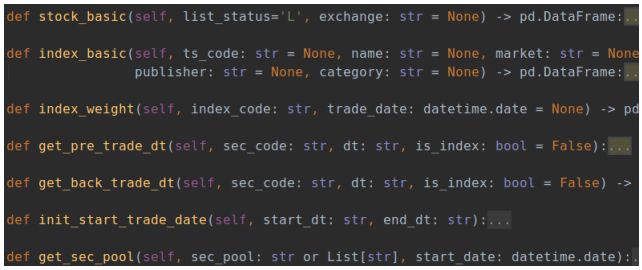

图 11: 基础信息部分

get_pre_trade_dt() 成员方法用于获取指定个股目标日期之前的最近的一个交易日日期，其中个股代码和目标日期由参数给出；

get_back_trade_dt() 成员方法与前一个方法类似，用于获取指定个股目标日期后的第一个交易日日期；

init_start_trade_date() 成员方法用于获得回测开始时的第一个交易日以及该交易日的前一交易日日期。具体方法是通过在交易日历中以传入的开始日期为分界线获取相应的日期；

get_sec_pool() 成员方法用于将传入的代码参数转化为数组形式的股池，其中传入的参数可能为字符串或字符串数组，其中如果是数组，则直接返回，如果是字符串，则判断是指数代码还是股票代码，如果为指数代码，则通过调用相应成员方法返回成分股，如果是股票代码，则返回包含该代码的数组。

2) k 线数据部分。该部分主要用于获取 k 线的数据以及在此基础之上加工生成的数据，比如价格、均线、交易量、收益率等内容。该部分当前包含的成员方法有：get_k_data()、get_ma()、get_price()、get_volume()、get_profit_rate()、get_profit_rates()、get_index_data()、get_price_list()。

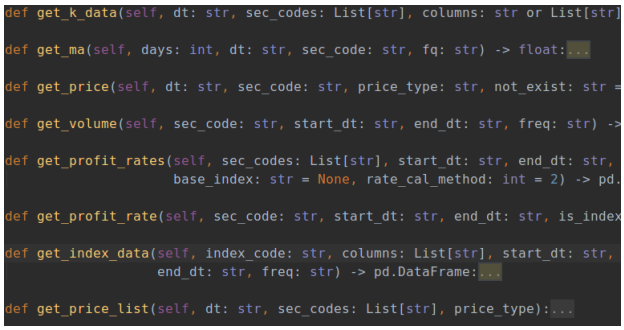

图 12: k 线数据部分

get_k_data()，作为一个核心方法，该成员方法负责从 MySQL 数据库中查询 k 线数据，目前数据库中只包含了日 k 数据，属性包含每日价格、成交量、每日指标、一些常用的均线数据以及一些辅助功能的数据。该方法在查询时要指定日期、股票列表、查询的具体属性，另外，作为可选项，可以指定查询的条件，比如收盘价高于某一个数值，前提是数据表中应包含该属性；

get_ma()，该成员方法主要用于从数据库中查询指定天数的均线，目前该方法只能查询数据库中存在的均线，还无法查询任意天数的均线；

get_price()，该成员方法主要用于查询指定目标的在指定日期的某一种价格比如收盘价，该方法不常用并且之后可能会删除；

get_volume()，该成员方法用于查询指定证券在一段时间内的成交量，将来应该会删去；

get_profit_rate()，该成员方法用于计算指定证券在指定时间范围内的收益率，目前提供 2 种计算方法，分别为通过价格计算以及通过涨跌幅计算。通过价格计算的方法为取得最后一天以及第一天的价格进行计算，但是，考虑到分红对价格的影响，该方法还需要改进；通过涨跌幅计算的方法是从第一天开始依次计算收益率，该方法可以不用考虑分红对价格的影响，但是需要考虑分红使持仓市值变化的情况；

get_profit_rates()，该成员方法用于计算一系证券的收益率，通过依次调用前一个成员方法并最后返回 DataFrame 实例，该方法还提供返回基准收益率的功能，基准收益率即指定指数在同一时间段内的收益率；

get_index_data()，该成员方法用于返回指定指数在指定时间范围内的相应行情数据，获取的数据项由参数给定；

get_price_list()，该成员方法用于返回一系列证券的价格；

3) 财务数据部分。该部分主要用于获取财务数据以及在此基础之上加工生成的数据，比如净资产收益率、资产负债率等财务指标。该部分当前包含的成员方法有 get_fina_report()、get_fina_data()、get_fina_list()、get_fina_date_range()、get_pe()、get_ps()、get_pe_list()、get_ps_list()。

```
def get_fina_report(self, sec_code: str, year: int, quarter: int, **kwargs
def get_fina_data(self, sec_code: str, columns: List[str], **kwargs) -> pd.
def get_fina_list(self, sec_codes: List[str], columns: List[str], **kwargs
def get_fina_date_range(self, sec_codes: str or List[str], year: int, quar
def get_pe(self, dt: str, sec_code: str, pe_type: str) -> float:...
def get_ps(self, dt: str, sec_code: str) -> float:...
def get_pe_list(self, dt: str, sec_codes: List[str], pe_type: List[str]) ->
def get_ps_list(self, dt: str, sec_codes: List[str]) -> pd.DataFrame:...
```

图 13: 财务数据部分

get_fina_report()，该成员方法用于获取指定证券在某期的财务指标数据，具体时间由参数的 year 和 quarter 给定，在此基础上，通过可选参数 filters，可以通过具体的条件进行筛选，比如获取净资产收益率 roe 大于 10 的财报，如果不存在该期财报或者财报不符合筛选的条件，则会返回一个只有一个证券代码的 DataFrame 实例；

get_fina_data()，该成员方法与通过具体财报期来获取数据的方法不同，该方法是通过指定时间范围来获取该范围内的财务指标数据，除此之外，在从数据库中查询数据的依据也不同。一个财报数据记录存在两个日期，一个日期表示该财报的期数，通常以一个季度的最后一天来表示该季度的财报；另一个日期是该财报发布的日期，只有到了这个日期，才能读取该相应季度的财报数据。该方法提供一个可选参数 limit，用于指定获取财报的个数，当时间范围内财报期数过多时，通过该参数来限制返回的数量；

get_fina_list()，该成员函数用于获取一系列股票的财报数据，该方法提供的可选参数有 year、quarter、截止日期 end_dt 以及筛选条件 filters，该方法通过判断传入可选参数的值来选择使用相应的读取财报成员方法，并且可以通过 filters 来筛选符合条件的数据。最后返回相应的 DataFrame 实例；

get_fina_date_range()，该成员方法用于获取一系列股票的财报发布日期间隔，即从某期财报的发布日到下一期财报的发布日；

get_pe()，该成员方法利用财报数据计算指定证券在指定日期的市盈率，方法是用股价除以最近一期财报的每股利润，由于目前 k 线数据提供了包括市盈率，市净率等指标在内的每日指标，而且财报的查询问题比较复杂以及发布的每股收益不是一整年的，所以该方法当前没有使用；

该方法可以计算 3 种市盈率：静态市盈率、动态市盈率、滚动市盈率（TTM）。其中静态市盈率是指当前股价除以去年年报的每股收益；动态市盈率是指当前的股价除以本年的预估市盈率；滚动市盈率是根据最近四期财报的收益来计算市盈率；

get_pb()，该成员方法用于计算指定证券在指定日期的市净率，即通过指定日期的股价除以每股净资产 bps。由于当前 k 线数据中已经包含了每日的市净率，该方法没有使用；

get_pe_list()，该成员方法用于获取一系列证券的市盈率，即为每支证券通过调用相应成员方法来获取对应市盈率；

get_pb_list()，该成员方法用于调用相应成员方法获取一系列证券的市净率。

4) 条件判断部分。该部分主要用于判断各种可能出现的条件，比如是否为交易日等。该部分当前包含的成员方法有 is_trade()、is_marketday()。

```
def is_trade(self, sec_code: str, dt: str) -> bool:...

def is_marketday(self, date: datetime.date) -> int:...
```

图 14: 条件判断部分

is_trade()，该成员方法用于判断指定证券在指定日期是否交易，返回 bool 类型值；

is_marketday()，该成员方法用于判断 a 股大指定日期是否交易，返回 int 类型值，用 0 或 1 来表示当天不开盘或开盘。

3.1.3 api.py

该文件用于编写回测过程中的操作函数，将证券交易过程中人的行为以及相关的处理操作编写成函数，使得回测过程更加清晰。当前该文件中包含了 order() 函数。

1) 下单函数 order() 模拟交易过程中的下单操作，即交易者选定证券、数量以及价格并提交后的操作，该函数根据这些参数，调整账户之中相关的值，比如资金、持仓、市值等属性。由于目前该系统的回测还比较简单，所以当前的下单操作是即下单即完成成交，并且不考虑手续费等细节内容。

传入该函数的参数中有账户 AccountInfo 实例和回测全局变量 GlobalVariable 实例，根据下单的类型分为“买入”和“卖出”两种操作，“买入”操作会先判断账户中是否包含标的持仓，根据结果执行不同的操作，如果无标的持仓，则会新建持仓 Position 实例并添加到账户持仓数组中并调整资金账户的属性值；“卖出”操作则会从账户持仓中寻找该标的的持仓，查找到之后分别对持仓实例以及资金账户实例进行操作。

3.1.4 tools.py

该文件用于编写整个系统中可能多次用到的、能够代表一定意义的工具函数，用于实现特定的功能，能够使程序更简洁、条理更清晰。

1) MySqlServer 类，该类用于提供一个本地 MySql 的连接并实现简单的执行查询语句并返回目标数据的 DataFrame 实例。通过该类，可以在编写程序过程中需要访问数据库时能够方便的创建连接，省去再次设置数据库参数的过程；

2) is_str()，该函数用于判断目标参数是否是字符串，该函数用到 six 库，之后可能会删除；

3) to_date_str()，该函数用于将传入的 datetime 实例转化为字符串。由于 datetime 实例可能会包含时间部分，而有的地方只需要日期部分，所以设置一个参数来表示是否是只保留日期部分，该函数还提供一个参数来设置日期字符串的分隔符；

4) to_date()，该函数用于将传入的日期字符串转化为 datetime.date 实例，如果传入的是 datetime.datetime 实例，则会只保留日期部分，该函数还需要提供日期字符串中的分隔符参数，便于将字符串中的内容解析出来；

5) float_precision()，该函数用于将传入的浮点数精确到小数点后指定位数；

6) fq_trans()，数据库中 k 线分为日 k 线、周 k 线、月 k 线等其他种类，为了将不同种类进行区分，为数据库表设置了不同的后缀，该函数用于将传入的表示频率的单个字母转化为数据库表的后缀；

7) attr_explain()，该函数用于将系统中使用的英文字符串转化为其中文含义；

8) get_quarter()，该函数根据传入的日期字符串或 datetime.date 类型实例，返回该日期所在的季度；

9) cal_stock_amount()，由于 a 股交易是以手（100 股）为单位，所以根据传入的资金参数，需要经过一系列计算才可以得出最大的购买量。该函数用于根据传入的资金参数返回最大购买量；

10) corr_check()，该函数用于根据传入的相关矩阵，筛选出相关系数大于指定值的属性，并以数组的形式返回；

11) chg_dt_format()，该函数用于改变日期字符串的分隔符，将原来的分隔符修改为新的分隔符；

12) chg_dt()，该函数用于根据传入的日期来计算该日期向前或向后移动指定天数的日期，并且可以通过设置参数值来选择移动的自然日还是交易日。如果是选择的是交易日，则通过交易日历来获取目标结果，如果不是交易日，则直接通过 datetime.date 来操作；

13) call_back()，回调函数，在多进程运行时，为了能够显示错误信息，需要回调函数来输出错误信息；

14) chg_quarter()，简单的将传入的季度向后推一季度；

15) cal_ma()，用于计算均线值，计算方法为（前一天均值 × 均值天数 + 当天的值 - 求前一天均值时的第一个值）/ 均值天数；

16) get_fina_end_date()，该函数根据传入的年 year、季度 quarter 值来获取财报的 end_date 值，即所传入季度的最后一天；

17) get_mongo_symbol()，在根据指标数值筛选财报时，mongoDB 的判断符号与普通的 “><=” 不同，需要将这些符号转化为 mongoDB 支持的符号；

18) get_option_query()，该函数用于将字典格式的筛选条件根据目标数据库来转化为其支持的条件语句，其中 mysql 数据库返回的是字符串，mongoDB 返回的是字典。

3.1.5 variables.py

该文件主要用于编写整个系统的全局变量，当项目越来越大时，修改一个特定值将会越来越麻烦，所以通过设定全局变量来简化该过程。

目前该文件定义的全局变量有返回信息显示的级别以及错误信息显示的级别，其中显示返回信息共分别 4 个级别，错误级别显示共分别 3 个等级，不同等级会显示不同内容的信息，如下图所示。

```
# 等级1，显示回测开始与结束
ECHO_INFO_BE = 1
# 等级2，显示买入卖出操作
ECHO_INFO_TRADE = 2
# 等级3，显示每日账户更新
ECHO_INFO_ACCOUNT = 3
# 等级4，将信息输出到文件中
ECHO_INFO_SAVE = 4

# 报错等级1，不显示错误
ERROR_L1 = 1
# 报错等级2，将错误输入到终端
ERROR_L2 = 2
# 报错等级3，将错误作为异常
ERROR_L3 = 3
```

图 15: 当前项目的全局变量

3.1.6 exceptions.py

该文件主要用于编写自定义异常，在项目不断扩大的过程中，可能涉及的异常也会越来越多，为了使错误更加清晰以及便于维护，因此会自定义异常，目前定义的异常如下图。

```
class TransferError(Exception):
    """
    转账异常，主要有转出金额超出可取金额；调用转账函数传入错误参数
    """
    pass


class MoneyError(Exception):
    """
    资金异常
    """
    pass


class ParamError(Exception):
    pass


class SQLError(Exception):
    pass
```

图 16: 自定义异常

3.1.7 back_test.py

back_test.py

3.2 策略列表

策略列表, 这块先不写，等能整合到平台再写

3.2.1 海龟策略 turtle_strategy.py

海龟策略

3.2.2 双均线 double_ma_strategy.py

双均线

3.2.3 单因子 single_indicator.py

单因子

3.2.4 多因子 multi_indicator.py

多因子

3.3 金融模型

该部分主要用于用代码来实现用到的或常见的金融数学模型。

3.3.1 APT 模型 apt.py

套利定价理论 APT（Arbitrage Pricing Theory）认为，套利行为是现代有效率市场（即市场均衡价格形成的一个决定因素。如果市场未达到均衡状态的话，市场上就会存在无风险套利机会，并且用多个因素来解释风险资产收益，并根据无套利原则，得到风险资产均衡收益与多个因素之间存在（近似的）线性关系，而 CAMP 模型预测所有证券的收益率都与唯一的公共因子（市场证券组合）的收益率存在着线性关系。(http://www.jinrongbaike.com/doc-view-7599.htm)

APT 模型本质上是多元线性回归，根据多个自变量共同来预测因变量，该模型中，自变量就是各个因子，而因变量就是风险资产收益。

$$r = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n + \varepsilon$$

为了实现该模型，首先需要选定因子，因子的选择有不同的方式，我们可以将财务指标作为因子，也可以使用自己挖掘的因子或者其他方面的因子。目前系统处于刚起步阶段，所以现在使用财务指标来作为模型的因子。在此基础上，由于财务指标的种类很多，仅仅在 tushare 中就可以取到超过 100 种的指标，将所有的这些指标都作为模型的因子显然是不现实的，不仅需要计算相当长的时间，模型的效果也很难令人满意。所以我们就需要使用特定的方法来选择效果更好的指标，关于选择的方法以及用到的模型将会在下一部分进行说明，此处我们使用有效性检验的方法来选取因子，通过筛选出与收益率相关性较高的财务指标来作为该模型的因子。

在选出一定数量的因子后，还需要检验各因子之间的相关性，如果相关性高的因子同时存在，不仅会造成不必要的运算量，而且对模型的准确度也会产生反而的影响。所以我们需要将相关性高的因子保留其中之一并剔除其余因子。

选择好因子之后，就需要获取因子的数值（也称为因子暴露）以及对应因子下的收益率（也称为因子收益），并将其处理成如下向量形式。其中 X 表示的是各因子即所选财务指标的值，Y 表示与该财务指标对应的一段时间内的资产收益率。为了获得足够量的数据，我们需要从时间和空间上进行扩展。

$$\begin{bmatrix} X & Y \end{bmatrix} = \begin{bmatrix} x_1, & x_2, & \ldots, & x_n, & y \end{bmatrix}$$

从时间上，选取不同时间的财务数据以及在相应时间段内的资产收益率。财务数据是上市公司每个季度发布的财报中的数据，所以每个季度都可以获得一组上市公司的数据，每年 4 组数据，获取多年的数据就可以有效扩大数据量；从空间上，我国有很多上市公司，仅 A 股就有 4000 多家公司，每家公司每年都会发布 4 期财报，获取每家公司多年的财报，能够得到比较充足的数据。与每期财报的财务数据指标相对应的相应个股在财报发布后一段时间内的收益率，该时间区间需要再进一步选择。

数据准备完毕后，就需要进行多元回归，该过程分为训练以及测试两部分，将总数据集分为训练集以及测试集，根据测试的结果与实际结果的差异来评判模型效果的好坏。

该文件实现的类 APT 实现了包括根据指标获取指定日期范围、指定股池中个股的财务数据、资产收益率，并对指标进行相关性检验。之后将数据集划分为训练集和测试集，调用 sklearn 第三方库的线性回归库进行训练并测试。该类的结构如下，实现的成员方法有 correlation_check()、data_handle()、regression()、predict()。

1. correlation_check()，相关性检验，该成员方法用于计算各个财务指标之间的相关性，并用热力图展示出来，半成品，先简单略过；
2. data_handle()，该成员方法用于获取财务指标数据以及相应的一段时间内的资产收益率。该方法传入的参数有指定起始财报的年 year、季度 quarter，表示需要获取财报的财报期数 repo_num，表示计算资产收益率的时间范围的 test_days 以及其他辅助性的参数；
3. regression()，该成员方法用于训练以及测试多元回归模型，并输出测试的结果与实际结果之间的离散方差。
4. predict()，半成品，先略过。

```
class APT(object):
    indicators = []

    def __init__(self, **kwargs):...

    def correlation_check(self, **kwargs):...

    def data_handle(self, **kwargs):...

    def regression(self, **kwargs):...

    def predict(self, indicators: pd.DataFrame):...
```

图 17: APT 类

该文件在定义 APT 类以外还编写了一个函数 cal_profit_rate()，该函数用于在多进程计算收益率过程中子进程调用。APT 类的成员方法 data_handle() 成员方法实现了多进程计算收益率的功能来减少程序执行时间，该方法使用的是 multiprocessing 第三方库，通过定义进程池 Pool、利用队列 Manager().Queue() 来传递数据，每个子进程执行该计算收益率函数并将结果存入队列，在计算结束后从队列中将结果取出并存入 DataFrame 实例。

该文件中的成员方法与函数在计算收益率时，提供了不同的方法，一种是利用单因子回测的方式来模仿真正的交易来获取收益率，另一种是根据 k 线中的价格或涨跌幅来计算，不同方法具有不同的特点，之后也会继续完善收益率计算的过程，比如分红的影响。

3.3.2 有效性检验 validation_check.py

多因子模型是量化投资中应用非常广泛的一种模型，基本原理是在一系列有效因子的基础上，通过综合考察不同因子的重要程度而建立的选股标准，其假设股票收益率能被一组共同因子和个股特异因素所解释（http://caifuhao.eastmoney.com/news/20190912151117410111340?spm=001.cfh），所以因子的选择就成为了多因子模型非常重要的一环。选择有效的因子不仅能够让模型减少不必要的计算时间，而且能够提高模型的准确性，具有很重要的意义。

有效性检验可以有多种方法，比如因子 IC 值检验、回归法检验、分层法检验、因子的逻辑性及普适性检验等（参考网页同上）。本系统目前采用的是因子 IC 值的检验，并在此基础上实现 IR 值的计算，至于其他的方法不作介绍，如果有时间的话，会实现其他方法当作练习。

IC 即信息系数（Information Coefficient），表示所选股票的因子值与股票下期收益率的截面相关系数，通过 IC 值可以判断因子值对下期收益率的预测能力。信息系数的绝对值越大，该因子越有效（https://blog.csdn.net/m0_37876745/article/details/89326308）。

通常 IC 大于 3% 或者小于-3%，则认为因子比较有效。常见的 IC 有两种，一是 Normal IC（类比皮尔森相关系数概念），另一个是 Rank IC（类比斯皮尔曼相关系数）。

Normal IC，即某时点某因子在全部股票的暴露值与其下期回报的截面相关系数，f_{t-1} 为 t-1 期股票的因子值，r_t 为 t 期的股票收益率。

Rank IC，即某时点某因子在全部股票暴露值排名与其下期回报排名的截面相关系数，$order_{t-1}^{f}$ 为 t-1 期各股票的因子值排名，$order_t^r$ 为 t 期各股票收益率排名。

$$NormalIC = corr(f_{t-1}, r_t)$$
$$RankIC = corr(order_{t-1}^{f}, order_t^{f})$$

IR 即信息比率（Information Ratio），是超额收益的均值与标准差之比，可以根据 IC 近似计算，公式如下。IR=IC 的多周期均值/IC 的标准方差，代表因子获取稳定 Alpha 的能力，整个加测时段由多个调仓周期组成，每一个周期都会计算出一个不同的 IC 值，IR 等于多个调仓周期的 IC 均值除以这些 IC 的标准方差，所以 IR 兼顾了因子的选股能力（由 IC 代表）和因子选股能力的稳定性（由 IC 的标准方差的倒数代表）（https://blog.csdn.net/m0_37876745/article/details/89326308）。

$$IR = \frac{\overline{IC}}{std(IC)}$$

该文件中定义的类 ValidationCheck 的作用就是实现 IC 值以及 IR 值的计算，其中 IC 值又分为 Normal IC 和 Rank IC。该类实现的成员方法包括 ic_value_check()、ir_value_check()、cal_profit_rates() 以及 sort_indicator()，如下图。

```
class ValidationCheck(object):
    sec_pool = None
    indicator = None
    start_date: datetime.date = None
    end_date: datetime.date = None
    data_client = None

    def __init__(self, indicator: str, sec_pool: str or Lis

    def ic_value_check(self, value_type: str = 'rank', para

    def ir_value_check(self, value_type: str = 'rank', para

    def cal_profit_rates(self, fina_data, dt_range, rate_ca

    def sort_indicator(self, fina_data, partition=False):..
```

图 18: ValidationCheck 类

1. ic_value_check()，该成员方法用来计算 ic 值，包括 Normal IC 和 Rank IC 值，可以通过参数选择，该方法还支持多进程运算，可以通过参数选择。作为可选参数，test_days 表示计算收益率时的日期范围，year、quarter 用于指定财报，rate_cal_method 用于指定计算收益率的方法，共有 3 种方法可以选择，save_res 参数可能用来将得到的财务指标数据和收益率存入文件，echo_info 则用来运行过程中输出信息。

   计算 Rank IC 和 Normal IC 的区别就是在获取财务数据以及收益率后，Rank IC 需要将财务数据或收益率排序并添加新的表示排名的一列，最后求相关系数时用这表示排名的两列来计算，Normal IC 则直接用财务数据以及收益率来计算相关系数。

2. ir_value_check()，该成员方法用于计算 IR 值，即通过多次调用 ic_value_check() 来得到一组 ic 值，并求该组 ic 值的均值以及标准差来求得 IR 值。

3. cal_profit_rates()，该成员方法用于计算个股在一段时间内的收益率。计算过程同样可以分为同步以及异步，其中异步利用利用 multiprocessing 第三方库来创建进程池和队列，各子进程计算出结果后将结果导入队列中，父进程将这些结果存入 DataFrame 实例中。

   收益率的计算方法分为 3 种：通过回测获得、通过 k 线价格数据以及 k 线的涨跌幅来计算，不同方法具有不同的特点。

4. sort_indicator()，该成员方法用于将数据排序并返回排序后的结果。在不同种类的指标中，不同指标可能会有不同的比较方法，比如市盈率 pe，该值比较时需要分成正和负两个部分，正数部分数值越小，排名越前，而负数部分数值越小，排名越后，再比如净资产收益率 roe，该指标比较时不需要区分正数和负数，数值越大，排名则越靠前。

该文件中在定义的类之外同样定义了一个计算收益率的函数，该函数用于异步计算时在子进程中调用。由于多进程第三方库的原因，子进程中执行的函数必须在 ___main___ 部分能够直接调用。

3.4 数据端

数据端主要用来负责将数据存入本地数据库，其中，数据的来源主要有 tushare、swindex 等第三方库、网页数据等。当前的工作主要在于从第三方库读取并存储数据，还示过多涉及网页数据爬

取等其他方向。在此之外，数据端部分还负责对数据库的操作包括对库、数据表等内容的创建和操作等内容。

其中通过第三方库来获取并存储数据包括数据初始化存储以及之后在此之上的更新问题。该两个问题分别用两个文件 collect_data.py 和 daily_update.py 来处理，其中，collect_data.py 文件用于编写最初阶段从第三方库中读取并存储数据，daily_update.py 则负责在当前已有数据的基础上进行每日更新操作，即将每天新增的数据存入到已有的数据库中。

3.4.1 collect_data.py

该文件编写的函数用于从第三方库 tushare、tushare 读取需要的数据并存入相应的数据库中，其中 tushare 是一个包含着各种各样金融数据的接口，包括股票基础信息、行情数据、财务指标、基金数据、汇率数据等内容，是该系统最重要的一个数据源，但是由于个人原因以及该接口中不同数据需要相应数量的积分要求来读取，所以在该数据源的基础上，寻找其他可靠的数据源用来补充相应部分的数据。swindex 第三方库就是用来读取申万行业行情数据，由于该库是基于申万网站的数据，所以当前只能获取一级行业的行情数据。

对于不同类型的数据，根据其特点用不同类型的数据库来存储能够达到更好的效果。该系统用到的数据库有 MySQL 以及 mongoDB，其中 MySQL 是很常用的关系型数据库，而 mongoDB 数据库是本人在该系统中首次使用，所以会从基本内容开始应用。

根据目前的进度,该文件中定义的函数有 get_sw_index()、get_fina_data()、get_income_data()、get_balance_data()、get_cashflow_data()、get_stock_k()、get_index_k()、stock_dk()、index_dk()、get_backup_dk()。

```
def get_sw_index():...

def get_fina_data(attrs: List[str] = None):...

def get_income_data():...

def get_balance_data():...

def get_cashflow_data():...

def get_stock_k(code: str, start_date=None, freq='D'):...

def get_index_k(code: str, start_date=None, freq='D'):...

def stock_dk():...

def index_dk():...

def get_backup_dk():...
```

图 19: collect_data.py 文件内容

1. get_sw_index()，该函数用于从 swindex 第三方库中获取申万一级指数行情数据并存入到本地数据库中。对于每个一级行业代码，先获取其日线的价格、涨跌幅以及交易量等基础数据，再

获取换手率、市值等这些每日指标数据，将两者合并后存入数据库；

2. get_fina_data()，该函数用于从 tushare 获取财务指标数据，并将其存入 mongoDB 中；
3. get_income_data()，该函数用于从 tushare 获取利润数据并存入 mongoDB 中；
4. get_balance_data()，该函数用于从 tushare 获取资产负债表数据并存入 mongoDB 中；
5. get_cashflow_data()，该函数用于从 tushare 获取现金流量表数据并存入 mongoDB 中；
6. get_stock_k()，该函数主要用于从 tushare 中获取个股的 k 线数据并存入本地数据库，根据当前的进度，该函数只获取日线级别的数据，其他级别的数据将会随着项目的进展而完善；
7. get_index_k()，该函数用于从 tushare 获取指数的 k 线数据并存入本地数据库，当函数的情况同个股 k 线函数；
8. stock_dk()，该函数用于指定获取 k 线数据的个股范围，对该范围内的个股，分别调用相应的函数；
9. index_dk()，该函数用于指定获取 k 线数据的指数范围；
10. get_backup_dk()，该函数从 tushare 获取备用行情数据，作用目前还未知。

3.4.2 daily_update.py

在初始存储数据的基础上，该文件用于负责数据的更新。目前该文件实现的是日 k 数据的更新，由于 tushare 中 k 线数据中价格的前复权以及均线计算的特点，日 k 的更新包括 3 个过程：区间 k 线数据的获取并存入原表中、修改原数据的复权价格以及均线数据的计算。

其中区间 k 线数据的获取及获取，该过程与初次获取数据时调用的函数为同一个函数，除了时间参数不同外，其他都是相同的。如果没有指定更新的开始日期和结束日期，将会使用原数据库表中最后一个日期作为开始日期，以当前日期作为结束日期；

tushare 价格数据的前复权方式是通过复权因子计算得来，如果原表中的价格为前复权价格，那么更新时需要以最新的复权因子为基准，计算原表数据的前复权价格，其中 $adjfactor_i$ 为所更新价格当天的复权因子，$adjfactor_n$ 表示最新的复权因子，p_i 为更新前的前复权价格。

$$p = \frac{adjfactor_i}{adjfactor_n} * p_i$$

最后需要更新均值数据，从 tushare 读取数据时可以添加指定天数的均值数据，但是每此读取的数据中，由于数据量不足，部分均值数据会为空。因此更新数据时，需要手动计算新加的均值数据，而且也需要将原先的均值数据进行复权处理，复权的方式同上。

需要注意的是，由于不同财经平台的复权算法不同，因此也会有不同的复权价格，这一点可能会影响系统运行的可靠性。之后可靠会根据效果将价格都转换为复权之前的数据。

该文件中定义了一个类 UpdateDataClient，该类可以用于向数据库中更新数据，同时也实现了向不同数据库查询数据并返回的成员方法。

在定义类之外，实现了两个函数用于将数据库中的数据进行复权，即 daily_update() 和 update_data() 函数。

由于该文件中的功能还未正式投入使用，所以就简单说明一下，之后在投入使用之前还会进行改善。

图 20: daily_update.py 文件

3.4.3 spider.py

该文件目前使用的是以前写的爬虫文件，而且还未完全搬移过来，先跳过。

3.5 服务端

服务端用于响应客户端的请求、从数据端获取数据并处理成适当的形式传送到客户端。

本系统中 web 服务器使用的是 Flask 框架。Flask 是一个微型的 Python 开发的 Web 框架，而且 python 在数据处理上有着很大的优势，服务器可以很好的和其他功能模块组合。

3.5.1 app.py

该文件的用于设置路由，对于来自客户端的不同请求，可以分别为不同路由编写不同的函数，特别是在传输数据时。Flask 中用 route() 装饰器将 URL 绑定到函数，在地址栏中输入相应的 URL，便可以将相应函数的输出显示在浏览器中。

该文件中的当前定义的路由主要包括两部分，第一部分是用于响应页面请求，第二部分则是用于响应客户端的数据请求。

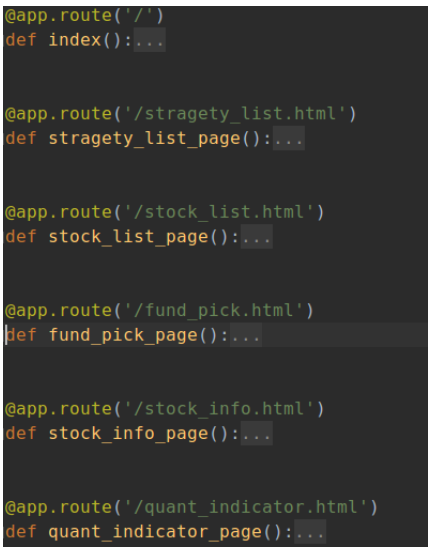

图 21: 页面请求路由

```python
@app.route('/quant/init', methods=['GET', 'POST'])
def quant_init():...


@app.route('/stock_pick/init', methods=['GET', 'POST'])
def stock_pick_init():...


@app.route('/stock_pick/search', methods=['GET', 'POST'])
def stock_pick_search():...


@app.route('/data/kline', methods=['GET', 'POST'])
def get_k_data():...


@app.route('/data/profit_line', methods=['GET', 'POST'])
def get_profit_line():...
```

图 22: 数据请求路由

第一部分中,当前定义的响应页面请求路由有'/'、'/stragety_list.html'、'/stock_list.html'、'/fund_pick.html'、'/stock_info.html'、'quant_indicator.html'，这些路由分别绑定了其对应的函数。

1. /，该 url 对应的是网站的首页；
2. /stragety_list.html，该 url 对应是是选股功能中的策略列表页面，通过选择不同的策略可以选出不同的股票；
3. /stock_list.html，该路径对应的是选择相应策略后用于列出股票以及实现其他功能的页面；
4. /fund_pick.html，该 url 用于列出基金，目前尚未开发；
5. /stock_info.html，该 url 对应的是具体个股的信息页面，目前尚未开发，可以之后会应用同花顺下问财的个股页面；
6. /quant_indicator.html，该 url 用于二次利用多空指标，目前计划是仿照专栏中的各个可视化图，之后可以会添加新内容，目前尚未开发。

第二部分中,当前定义的响应数据请求路由有'/quant/init'、'/stock_pick/init'、'/stock_pick/search'、'/data/kline'、'/data/profit_line'，这些路由分别绑定了对应的函数。

1. /quant/init，该 url 用于在进入多空指标页面后的初始化，目前实现的是为客户端返回多空指标的数据；
2. /stock_pick/init，该 url 用于进入股票选择页面后，返回页面初始显示的个股列表及其信息；
3. /stock_pick/search，该 url 用于在设定一定条件后，将符合筛选条件的股票及其信息返回到客户端；
4. /data/kline，该 url 用于向客户端返回指定个股的 k 线数据用于绘制 k 线图；
5. /data/profit_line，该 url 用于向客户端返回指定时期范围内个股的收益率以及其他指标数据。

3.5.2 server.py

服务端的功能除了设定路由来响应客户端请求之外，还要实现根据请求获取并处理数据的功能，该文件主要用于实现这一项功能。

该文件以编写一个 Server 类的形式来实现各种当前用到的服务，该类的成员变量即是该系统中定义的数据库接口以及一些可能会重复用到的基本信息，这也是通过以类的形式来实现服务的原因之一，这样不同的服务方法中不需要各自与数据库连接。

目前该类中实现的成员方法即服务主要有 k_data()、get_profit_line()、get_fina_data()、get_quant_data()、sec_filter()、add_option()。

```
class Server(object):
    mysql = tool.MySqlServer()
    data_client = DataClient()
    basic_info = BasicInfo()

    def k_data(self, sec_code: str, **kwargs):...

    @staticmethod
    def get_profit_line():...

    def get_fina_data(self, sec_codes, **kwargs):...

    def get_quant_data(self):...

    def sec_filter(self, options: List[str]):...

    @staticmethod
    def add_option(key: str, words: List[str], filters: di
```

图 23: Server 类

1. k_data()，该成员方法用于返回指定个股的 k 线数据，包括基础的价格、均值价格以及交易量等数据。客户端利用该数据绘制 k 线图；

2. get_profit_line()，该方法用于返回回测之后的收益率、夏普率等指标，用于衡量策略的收益能力；

3. get_fina_data()，该方法用于返回指定个数序列的财务指标数据以及一部分行情的每日指标数据。该方法还提供按条件筛选的功能，当前具体条件是该方法可以返回的数据属性，将符合筛选条件的数据返回。

4. get_quant_data()，该方法用于返回多空因子的量化指标数据，目前只返回多空因子值以及指数的每日 k 线，还需要进一步开发；

5. sec_filter()，该方法和下一个方法 add_option() 共同完成了将客户端传入的筛选条件处理成合适的格式。其中该函数用于将传入的筛选的条件进行分类，不同类型的筛选条件需要进行不同的处理，之后调用 add_option() 成员方法将已经分类好的条件进一步处理，形成特定构造结构的变量供最终处理。最后根据处理后的条件查询相应的财务指标数据。

6. add_option()，该方法用于将分类好的筛选条件处理成具有层次的字典变量，第一级是不同指标的类型，比如财务指标、行情指标等，第二级是具体的指标名称，该名称使用的是系统通用的英文字符串，比如 “roe”，第三级为指标的范围操作符，比如 “>” 等，第四级为具体的数值。

3.6 客户端

客户端是用 html、css、javascript 等来实现的网页客户端，网页可以很方便通过浏览器来访问，而且也对现有的网页技术比较熟悉，因此决定使用网页的形式来实现客户端。

目前已经设定的网页主要有 index.html、stock_list.html、stock_info.html、fund_pick.html、stragety_list.html 以及 quant_indicator.html，并且为了实现目标功能，还编写了相应的 css、javascript 文件.

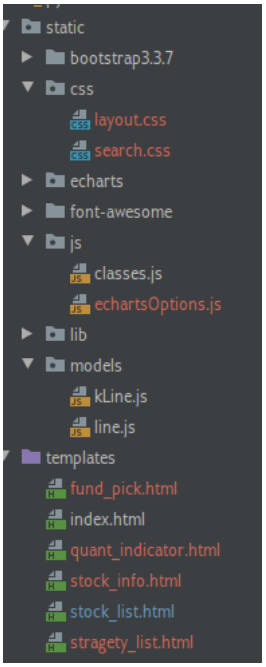

图 24: 客户端文件目录

3.6.1 主页 index.html

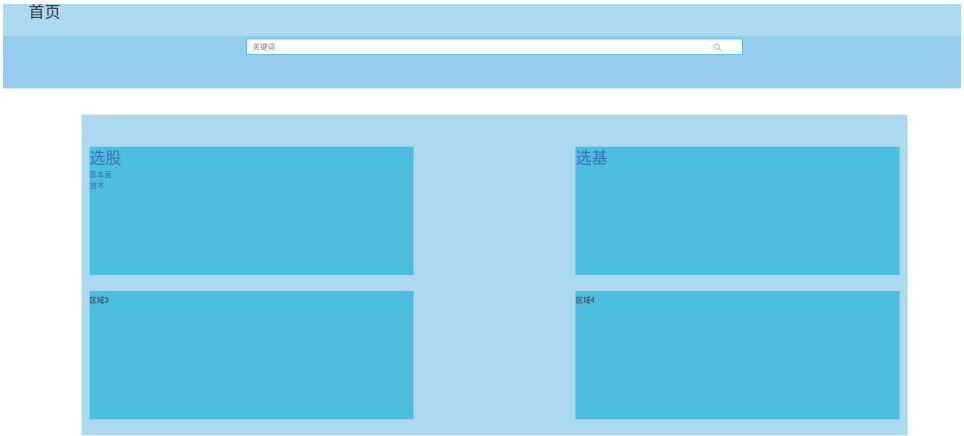

图 25: 主页页面

主页分为不同的部分，不同部分用于导向不同的页面，还待进一步开发。

3.6.2 策略列表页面 stragety_list.html

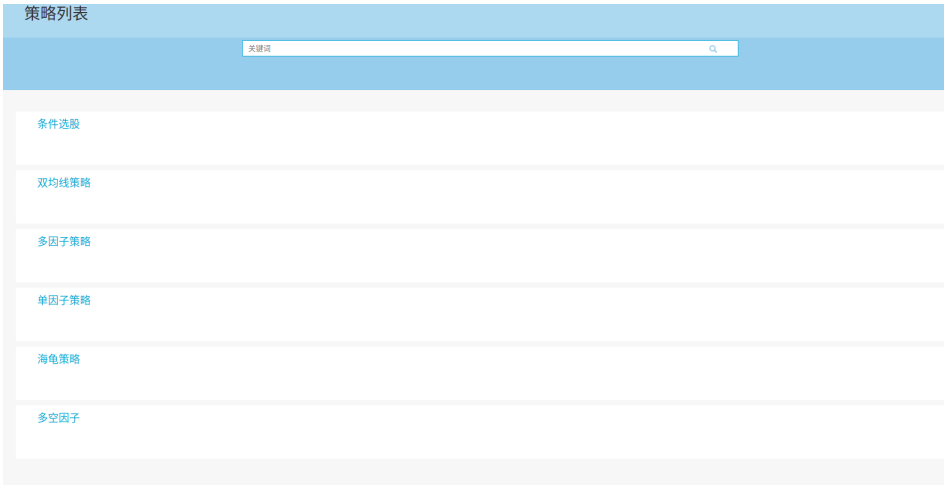

图 26: 策略列表页面

该页面用于列出已经实现或计划的策略，通过点击相应策略，可以进入相应的页面。目前的工作主要集中在条件选股中，多空因子只做了可视化图，其他页面还未开始。

3.6.3 条件筛选页面 stock_list.html

图 27: 条件筛选初始页面

该页面的中间部分为条件设定区域，该区域必选项为股池以及日期，股池用于确定股票的范围，日期则是用于确定目标位置。当前股池提供的选项有沪深 300、中证 500 等常用指数，通过点击 “+” 号来添加条件。

图 28: 添加条件框

目前作为测试提供的条件有净资产收益率 roe，市净率 pb，市盈率 pe。

图 29: 设置筛选条件

当设定目标条件并确定范围后，点击条件之后的“放大镜”图标来搜索，在这些添加的条件中，可以点击其后的“x”号来删除该条件。点击搜索后，就可以在页面中列出符合条件的股票，当前只在页面中列出了前 20 支。

图 30: 搜索之后的股票列表

对于每一支股票，可以设置其展示的信息，通过点击每支股票右下方的按钮可以显示出设置框。

图 31: 信息设置框

3.6.4 多空因子页面 quant_indicator.html

该页面利用多空因子专栏的数据，将其指标进行二次开发，目前的计划是先将专栏中的可视化图形实现。


多空因子指标
关键词
多空因子
多空因子
沪深300
2020-12-01
2020-12-11
2020-12-23
2021-01-05
2021-01-15
2021-01-27

图 32: 多空因子页面