2017 citi 四川大學 SICHUAN UNIVERSITY EFG 创业基金会

CITI FIANCIAL INNOVATION APPLICATION COMPETITION

“花旗杯”金融创新应用大赛

‘SmartTrade’ Automatic Trade System(ATS)

Technical Document

School：Shanghai University of Finance and Economics

Team：SUFE Renaissance Technologies

Director：Fei Xie Phd

上海财经大学 SHANGHAI UNIVERSITY OF FINANCE AND ECONOMICS 1917–2017

Menu

Require Document Menu

I.A brief introduction of documents

This document describes the background of the project, the needs of the industry, the entire system function, structure, and detailed description of each module content, function and logic.

II. A brief introduction of product

1) Project objectives

The goal of this project is to implement a China can be applied to the A stock market, the market for the past analysis model of linear programming and the multi factor time series analysis based on the design calculation of the ideal portfolio strategy, quantitative trading system and automatically by the system operation and information transmission in warehouse.

2) Background analysis

Since December 19, 1990, the Shanghai stock exchange to start business Chinese A stock market has gone through 27 years, but compared to European market for a hundred years of history, A stock market is still a young market, has started late, rapid development, continue to strengthen financial innovation characteristics, and in the background of rapid development of computer artificial intelligence under the quantitative trading system will also have the development prospects of the use of various strategies All flowers bloom together., system architecture will all run in such a mature growing competition in the market.

Foreign quantitative funds started earlier, according to Wilshire Associates statistics, the amount of investment funds in the U.S. stock market, a total of 1062, a total of 674 billion U.S. dollars of assets under management. As of May 26th this year, quantitative funds are the operation of the entire A stock market. A total of 163 teams, with the market value of A stock market and the total China for NYSE and Nasdaq market value and the 1/3 point of view, the healthy development of the stock market A can accommodate more number of quantitative funds in the next few years.

3) Description of problems and opportunities

The A stock market is still not a fully mature market, regulation is not perfect, at the same time, a larger proportion of individual investors, these individual investors have speculation, follow the trend of strong psychological, and the risk often take chances so slow. These also provide opportunities for quantifying the development of trading systems.

At the same time, most of the models used in the quantitative trading system in the market are mainly empirical algorithms, heuristic algorithms, multi factor models that do not involve time series, etc.. Each of these models has its own problems, for example, it may fail in a certain situation, or it needs to be artificially maintained and adjusted. Therefore, we need to expand the types of models, try to use the previously used model for calculation and operation, and analyze their advantages and disadvantages, and apply them properly. Quantitative trading system should also have some user-friendly monitoring and maintenance functions, such as test interface, market interface, position interface, etc. The running state of these interfaces can inform the user of the current system is convenient for users to carry out comparison, evaluation, selection strategy and the optimal parameters, and startup / shutdown system in timely case, or make adjustments on the system. In contrast, the system should also be adjusted for users, leaving interfaces.

4) Business requirements

In view of these points, we can sum up the main tasks that the project should be able to accomplish, that is, the business requirements of the system are as follows:

1. Can read and display the stock market information.
2. We can implement two models and the strategy based on them respectively, and calculate the ideal stock portfolio through the setting of strategy and parameters. The strategy should be able to perform better over the same period.
3. Allows users to simulate tests for selected policies and parameters, and display test results.
4. You can use the selected strategy and parameters for automated transactions.
5. Allow users to monitor and manage system operating status.

5) Project evaluation

The system is superior to human traders:

This quantitative trading system reacts faster than the human trader, can complete the transaction in a very short period of time, and can also monitor the price changes of a large number of traded products in the market at the same time. This advantage enables the system to complete the change of warehouse in a relatively short time, greatly reducing the risk of failure during the change of warehouse due to overtime and misoperation.

At the same time, the quantitative trading system can overcome the human impact of human traders, will not panic when the market is abnormal operation, and will not be too confident and make high-risk speculative trading. The quantitative trading system changes positions according to the strategy obtained by the algorithm analysis, and the transaction process is completely controllable, thus avoiding the uncertainty of manual trading.

This system is superior to other quantitative trading systems:

This project compared with the quantitative ordinary trading system, its innovation lies in the application of calculation using few trading in program of linear programming model and the multi factor time series analysis of the two models are the best combination. The essential purpose of the two models is to analyze and compare the historical data of a period of time, and find out an optimal portfolio that can perform well in the future.

The trading system in this project design than other program of automated trading systems more because of the back test for the warehouse strategy originates from the model of market performance, the system can according to the past market trend constantly optimizing strategy to flexibly adapt to the market, to avoid the user is forced to manually modify the strategy, save manpower cost. On the other hand, because of the algorithm mechanism, the future strategy directly from the previous market analysis, which can be adjusted slightly after being applied to different markets, such as futures, foreign exchange, overseas markets, compared with other quantitative trading system has more flexibility and universality, but also more suitable for

The interface designed by this system is simple and straightforward, and there is no superfluous and useless information. The trading system integrates some of the basic functions of most trading software on the market, such as market display, display of positions and so on, and has demonstrated the performance of the market and the operating status of the system. Moreover, in the test interface of strategy, the operation result of the strategy is clearly reflected by the performance of comparison object, which is convenient for users to observe and judge.

The experience of algorithm and heuristic algorithm using quantitative trading system is different from the existing market, the A stock market optimal combined to invest by the linear programming model of the case is scanty, and the application of linear programming model in the quantitative trading system is very rare. Because of this application has never been involved, the design of this project has unique features, as well as the infinite space and potential for future expansion. It has obtained achievements, the overall index of the linear programming model to calculate the strategy can clear the bull market trend with obvious advantages outperform the corresponding (Shanghai 300), from the recent A stock supervision direction, A shares are slowly to the more stable, more Manniu market transformation potential I can see clearly, in this context, the use of the model of the project will be able to get in the future more broad application.

III . User requirements description

We can deepen the business requirements mentioned earlier into detailed user requirements, namely, what specific functions the user needs to meet the needs of the user in the process. User requirements can be specifically summarized as follows:

Read and display stock market information:

1. information access needs: the system should first introduce and exchange interface, for access to the stock market, the transaction information, to ensure the system data source flow, the system can read the stock price history and calculation, also allows the system to automatically link to the exchange transaction transmission order.

2. information display requirements: the user can view the current position, strategy through the interface module of the system of the historical trend, the current market value of account information, keep the program run full control, transparent, user comparison.

Implementing models and calculating combinations:

3. operation strategy is calculated: the system using linear programming model, the multi factor time series analysis and other planning model, through the test analysis on the investment strategy of practical design, can calculate the ideal in different time span of the stock portfolio.

Linear programming model: Based on this strategy can track the underlying index, through the analysis of past market test, calculate the relative enhancement in the premise to control the tracking error under the combination of stock index, the pursuit of win and obtain excess returns.

Analysis of multi factor time series analysis: market price series and portfolio, through empirical mode decomposition obtained intrinsic mode equation, namely the decomposition of dynamic information affects the stock volatility, followed by iterative solution of the optimal solution for a relatively ideal portfolio. The results calculated by the model are preserved in the form of CSV, which include the number of shares in the portfolio, the respective stock codes and the market value of the portfolio. The above two models are the basis of system testing and transaction, and also ensure that the system can perform better. Analog test tuning parameters:

4. The adjustment and selection strategy: the user can choose to use that model specifically for the calculation of the strategy , than by setting the parameters of the model to make artificial adjustments to the strategy . After the setting is complete, the user can save and run the strategy as the goal, also can test first, and adjust the model parameters according to the test result constantly, and pursue the optimization of the strategy.

5. Testing: for the selected strategies and parameters, the system can test the market in the past as the subject, and the results of the test will be compared with the target used at that stage. Users can compare the trend and result of market performance and strategy performance in this period, and adjust the optimization strategy repeatedly.

Automated trading:

6. automatic change warehouse process: users can also choose to choose the strategy and parameters for trading. The system calculates the ideal combination

every day and calculates the difference between the ideal combination and the current portfolio (if the vacant position is not empty). It is divided into buying, adjusting, combining and selling positions. After triggering the change of stock, the system can sell and buy, then complete each stock transfer process, and automatically complete the transaction.

7., automatic trigger location: set the threshold adjustment mechanism, you can in the ideal combination is relatively cheap, trigger the transfer module, to lower the price to buy portfolio, save the cost for users to transfer positions. 8. special cases of adjustment: when the target is unable to tune warehouse stock transactions, the flexibility to use ETF, instead of cash, shares and other methods instead of the combination, guarantee the accuracy of the stock portfolio. 9. order process and parameter adjustment for the dramatic changes in the market price of the suspension, price limit and other special circumstances, transfer positions logic within the system will take alternative, Retry, upshift retry, under the limit and other methods to deal with. The user can also adjust the process details of the call / unwind process by setting up the interface and freely setting the parameters of each order (such as the number of attempts, the benchmark price, etc.). It can ensure the smooth operation of the buying and unwinding process while the stock market is changing rapidly. Inspection and management:

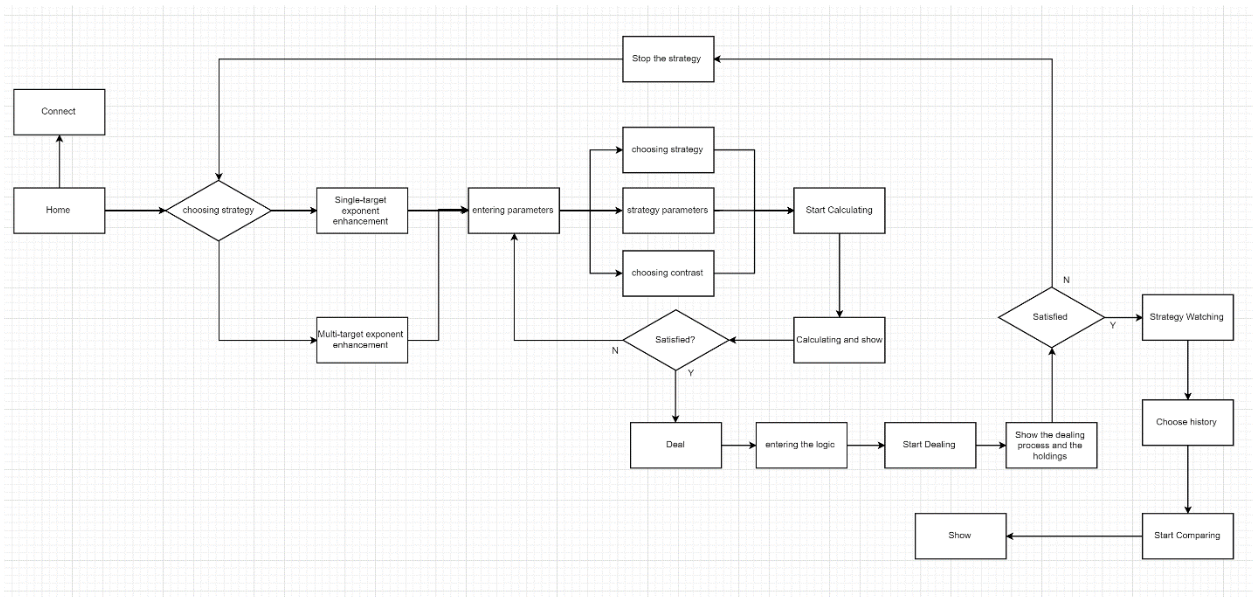
10. Analysis of historical operation: the system records the strategy of historical records, and the strategy of monitoring interface display, the user can view the control strategy and the trend index in a period of time, can also see the current

position, market value, and the selected index price, convenient for the user to monitor the operation at any time, and according to their own needs of opening or open.

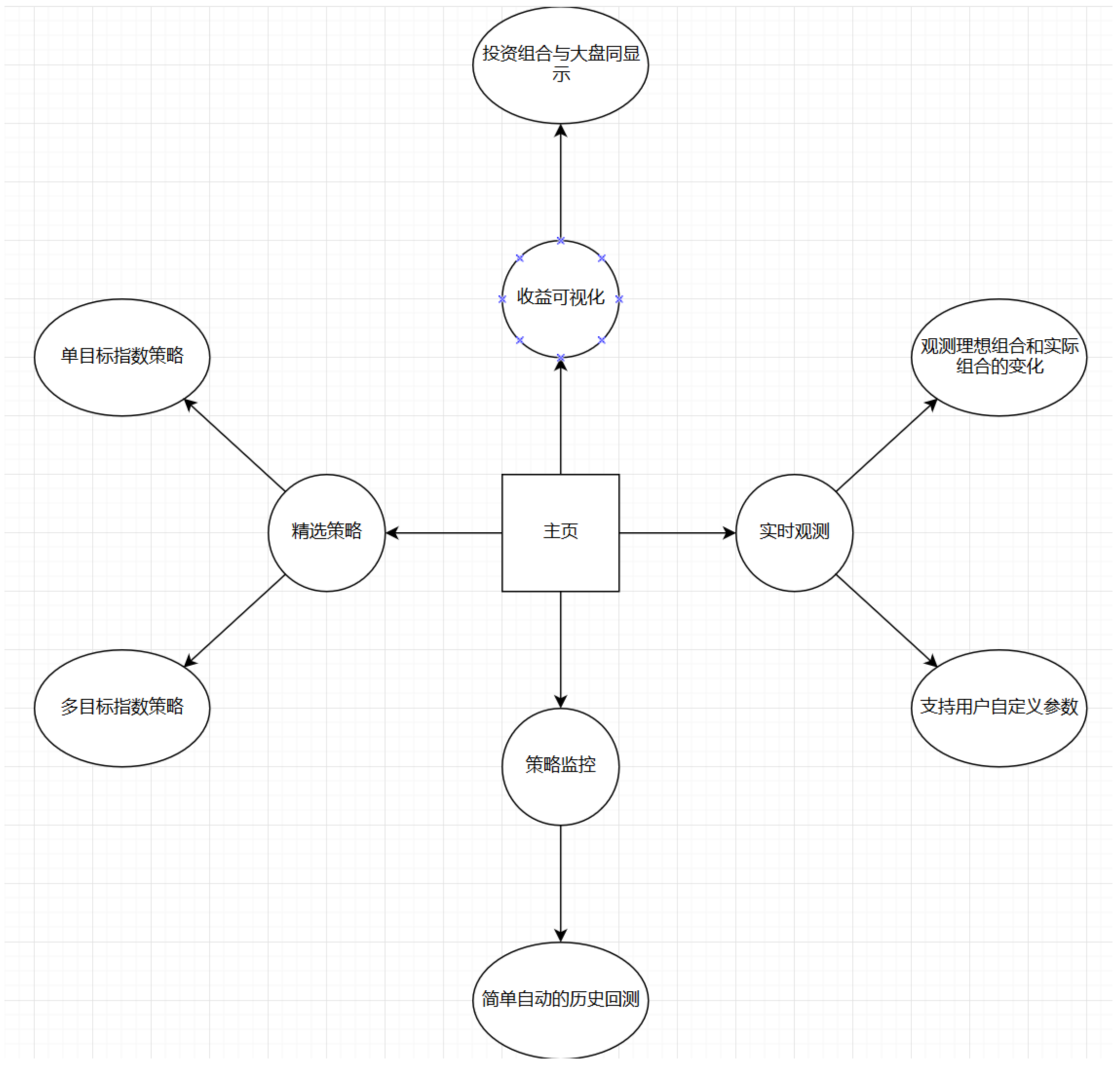
11., manual unwinding settings: the system also sets a manual unwinding button, when users predict the current market will be risk, or not satisfied with the operation of the strategy, you can carry out a key position. The system will sell the current stock portfolio in accordance with the established trading logic, and pause the strategy, so as to control the risk.

IV. Product description

1) Overall process



2) Function diagram



3) Functional structure diagram

module	sub module	Main function
The main interface module	Connecting	connected to establish the information connection with the exchange, so as to facilitate the acquisition of market and transaction information
Strategy	Sub-strategy-	Choose which model to use for Strategy

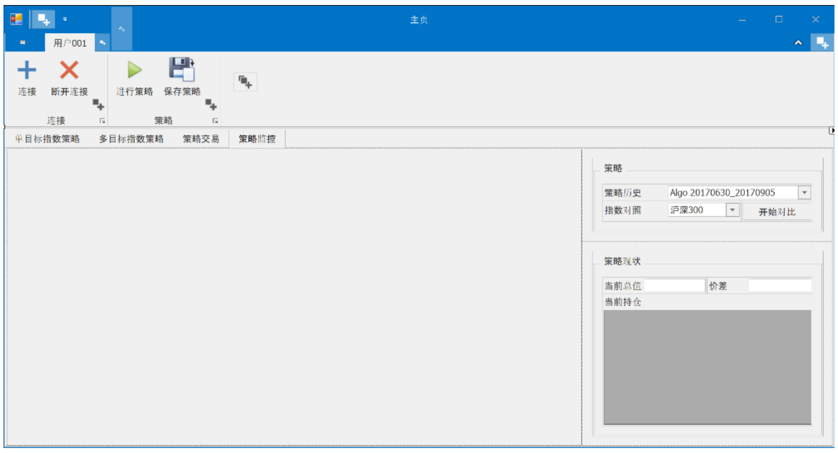
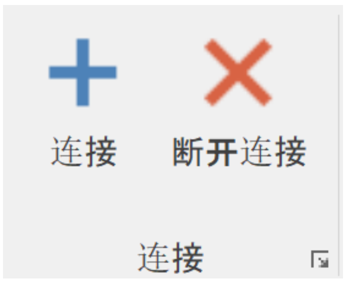
	selecting	calculations
	Single objective index	The calculations are performed using linear programming models
	multiple objective index	Strategy calculations are performed using multifactor time series analysis
	Parameter, contrast setting	Sets the parameters of the model with the index of the contrast
	Calculation and display strategy	Calculate the strategy and show the result of calculation
Transaction module	Edit strategy transaction logic	Modify various parameters in the trading strategy and adjust the transaction process
	Display transaction process	Show buying and selling operations and results during trading
	Inquiry positions	Displays current portfolio positions
	Pause strategy	Unwinding and stopping strategic operations
Monitor module	Inquiry positions	Displays current portfolio positions
	Query market value	Displays the current market value of the transaction account
	History-Analysis	The response strategy is expressed in the

		form of historical income line graphs

4) Product characteristics

1. main interface module

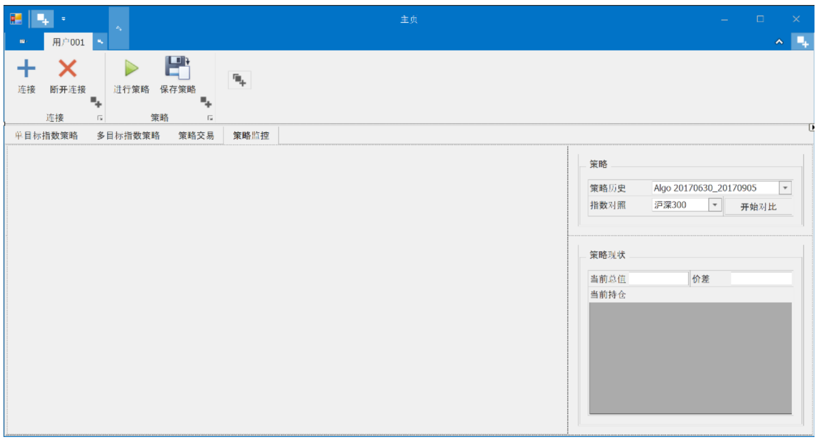
1.1 Connecting

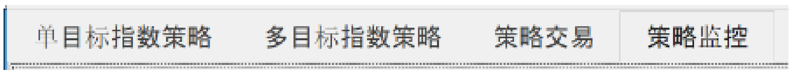
User scenarios	When the user starts the system
Function description	Establish the information connection with the stock exchange, so as to obtain the information of quotation and transaction
Input / pre conditions	The system starts and connects to the network
Requirement description	 主页 用户001 连接 断开连接 进行策略 保存策略 策略 单目标指数策略 多目标指数策略 策略交易 策略监控 策略 开始对比 策略现状 价差 当前持仓 策略历史 Algo 20170630_20170905 指数对照 沪深300 当前总值  (1) the user can click the link option information interface enable the system with the exchange system began to obtain market

	data and transaction data in the background, provide the data source for test strategy, operation strategy and view positions. (2) users can click to disconnect
Output / post condition	Click on the connection button, which opens the information interface between the system and the exchange, and begins to gain market data and transaction data. Click on disconnect to disconnect the information interface between the system and the exchange

2. Strategy　module

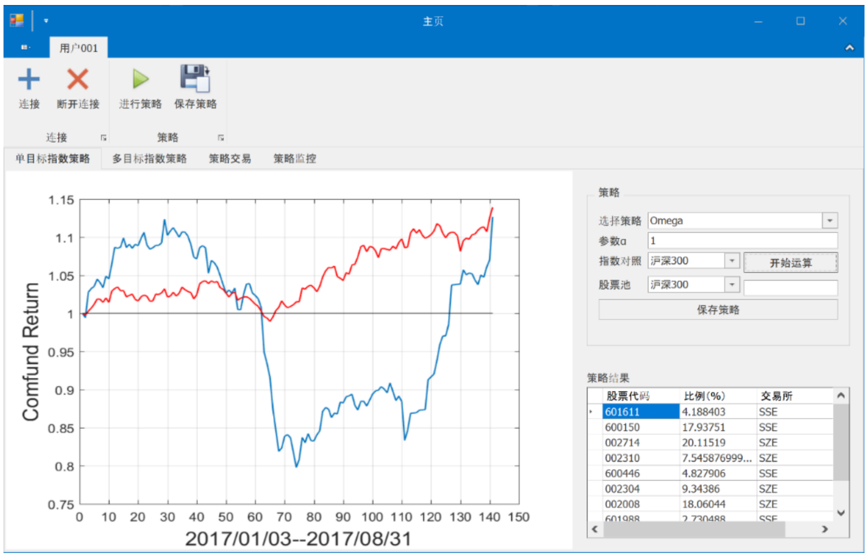
2.1 Sub-module-selecting

User scenarios	User selecting function
Function description	Choose which model to use for strategy　calculations
Input / pre conditions	The system has been established with the exchange
Requirement description	 主页 用户001 连接 断开连接 进行策略 保存策略 连接 策略 单目标指数策略 多目标指数策略 策略交易 策略监控 策略 开始对比 策略现状 当前持仓 策略历史 Algo 20170630_20170905 指数对照 沪深300 当前总值 价差

	 From the main interface as the main interface framework and each page provides the main interface container provides the basic functions of the main page; take the tab switch layout take two array of left right chart text, with PC user interface visual habits. The user can open the call interface of the strategy by specifically selecting which strategy 's tab sub page.
Output / post condition	Select the Single objective index strategy , are then entered using the linear programming model interface; Select the mutiple objective index strategy and enter the interface using multifactor time series analysis. Select strategy trading, then enter strategy, transaction interface, strategy monitoring. Select strategy monitoring, access strategy , historical performance, review interface.

2.2Single objective index

User scenarios	You need to call the model for calculations
Function description	The calculations are performed using linear programming models
Input / pre	Establish a connection and the user has chosen the strategy to

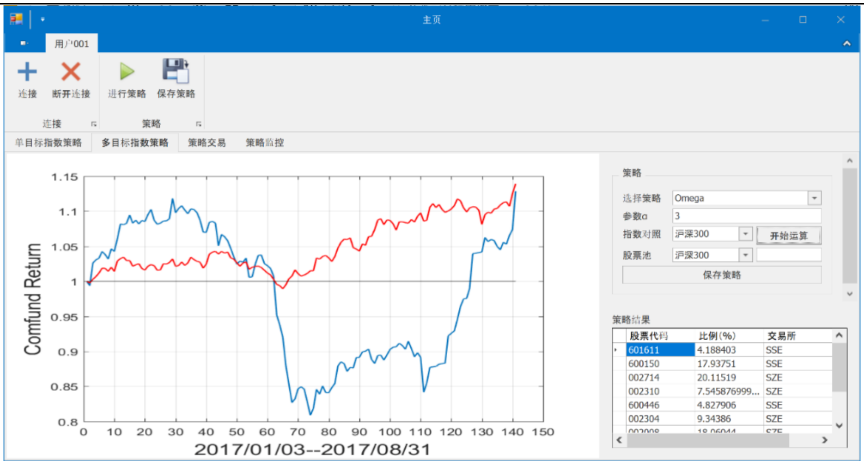
conditions	use
Requirement description	 The user can choose strategy strategy in the upper right corner, and then modify the strategy parameter value and control parameters, the choice of stock pool is calculated, then software will get strategy according to the results of call parameters Matlab and Gurobi in the last minute and the proportion of positions will be displayed in the lower right corner, the market trend and control index the trend shown on the left side of the chart.
Output / post condition	

选择策略 Omega
参数α 1
指数对照 沪深300
股票池 沪深300

股票代码	比例(%)	交易所
601611	4.188403	SSE
600150	17.93751	SSE
002714	20.11519	SZE
002310	7.545876999...	SZE
600446	4.827906	SSE
002304	9.34386	SZE
002008	18.06044	SZE
601988	2.730488	SSE

2.3mutiple objective index

User scenarios	You need to call the model for calculations
Function	Strategy calculations are performed using multifactor time series

description	analysis
Input / pre conditions	Establish a connection and the user has chosen the strategy to use
Requirement description	 Multi target enhancement and single page target enhanced index page is similar to that of the first strategy area in the upper right corner of the selection strategy, and then modify the strategy parameter value and control parameters, the choice of stock pool and calculation software, then according to the parameters of MATLAB and gurobi in the call to the strategy, and the proportion of positions will last time display in the lower right corner, the market trend and control index trend display on the left side of the chart.
Output / post condition	

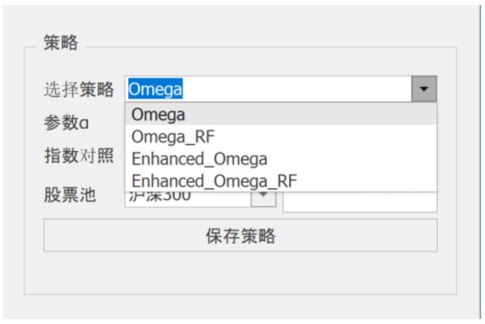
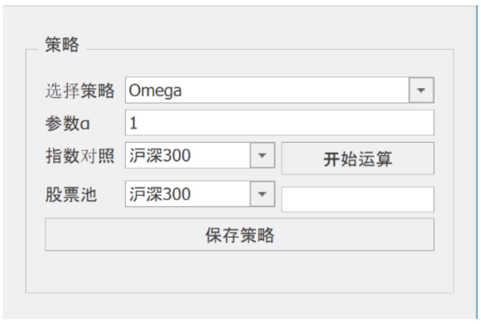
策略

选择策略: Omega
参数α: 3
指数对照: 沪深300
股票池: 沪深300

策略结果

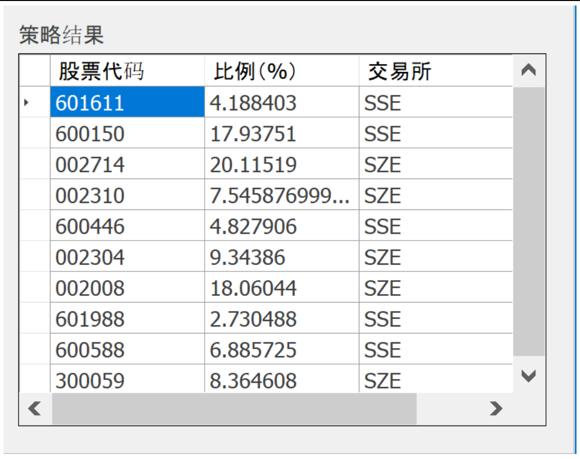
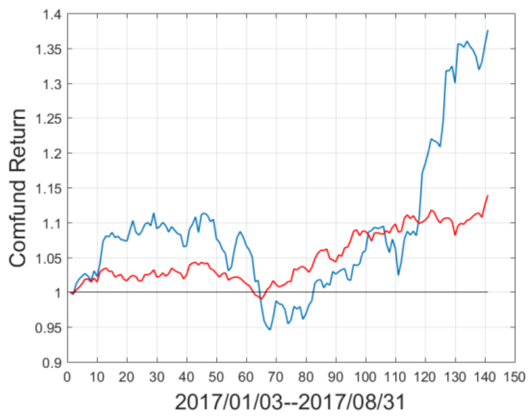
股票代码	比例(%)	交易所
601611	4.188403	SSE
600150	17.93751	SSE
002714	20.11519	SZE
002310	7.545876999...	SZE
600446	4.827906	SSE
002304	9.34386	SZE
002008	18.06044	SZE

2.4 Parameters and contrast settings

User scenarios	You need to call the model for calculations
Function description	Sets the parameters of the model with the index of the contrast
Input / pre conditions	Establish a connection and the user has chosen the strategy to use
Requirement description	 策略 选择策略 Omega ▾ Omega Omega_RF Enhanced_Omega Enhanced_Omega_RF 参数α 指数对照 股票池 沪深300 保存策略 Users can choose the model type in the select strategy drop-down box, including the linear programming model, Omega and Enhanced_Omega, and the mixed integer programming model Omega_RF and Enhanced_Omega_RF.  策略 选择策略 Omega 参数α 1 指数对照 沪深300 开始运算 股票池 沪深300 保存策略 The user can change the target of the strategy by filling in the value of the parameter alpha, which means that the strategy yield is higher than the target value.
Output / post condition	Save strategy -- save the selected strategy with a combination of parameters for next use.

	Start the operation -- calculate the strategy result

2.5Calculation and display strategy

User scenarios	You need to call the model for calculations
Function description	Calculate the strategy and show the result of calculation
Input / pre conditions	The user selects the strategy and the corresponding parameters
Requirement description	  In the set which model to use and the parameter value, can choose to begin to calculate test, test results in the calculation after the show in the strategy interface, including the proportion

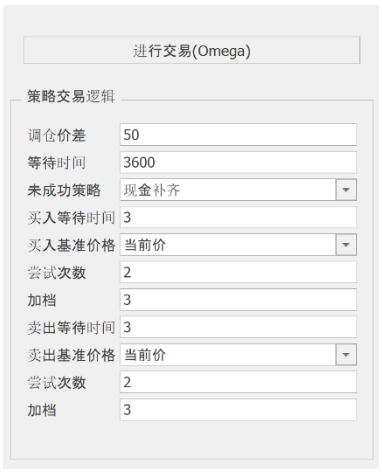
策略结果

股票代码	比例(%)	交易所
601611	4.188403	SSE
600150	17.93751	SSE
002714	20.11519	SZE
002310	7.545876999...	SZE
600446	4.827906	SSE
002304	9.34386	SZE
002008	18.06044	SZE
601988	2.730488	SSE
600588	6.885725	SSE
300059	8.364608	SZE

	of positions and final moments (above) and the trend of market trend and control index (below). In the figure below, the Y axis represents the composite yield of the index or strategy, and the X axis represents the time series, which is a trading day.
Output / post condition	

3.Transaction module

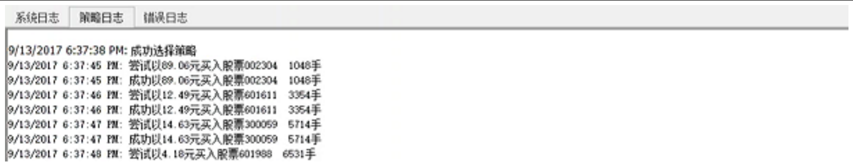
3.1Edit strategy transaction logic

User scenarios	User enabled strategy for transaction
Function description	Modify various parameters in the trading strategy and adjust the transaction process
Input / pre conditions	Set the target strategy and parameters
Requirement description	 进行交易(Omega) 策略交易逻辑 调仓价差 50 等待时间 3600 未成功策略 现金补齐 买入等待时间 3 买入基准价格 当前价 尝试次数 2 加档 3 卖出等待时间 3 卖出基准价格 当前价 尝试次数 2 加档 3 The strategy transaction interface is divided into two parts: the right side is the strategy transaction logic area, and the user can

	change the trading strategy here. Trading strategies include first suspension of judgment, if the stock is not transferred to the suspension successful strategy alternatives, including cash, stock replacement (Replacement Replacement Replacement shares as the Shanghai and Shenzhen 300 with the highest correlation coefficient of the stock before the three stocks) and ETF to replace, otherwise to the benchmark price of Gaga file for transfer positions. If in a waiting time after the transaction is not to retry the benchmark price of Gaga stalls for the warehouse, and in a sustained period of time is still not successful transfer positions, the system will be directly to limit price transactions. The whole process is aimed at lower prices to complete the Jiancang, while responding to the dynamic changes in the market. Set the price difference: set when the target portfolio is cheaper than the current combination, start clocking. Waiting time: set when the target portfolio meets the requirements of the current portfolio spreads, how long will it take to maintain the price difference and then adjust the position. Unsuccessful strategies: alternative methods include cash filling, alternative stock substitution, and ETF substitution Buy (sell) wait time: wait until each round is retried before

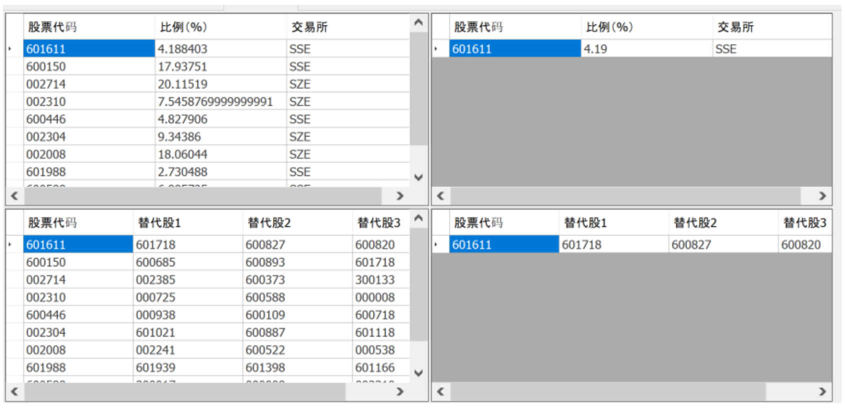
	leaving the sale. Buy (sell) base price: the price for each filling (selling order) is composed of the base value and the additional file. The reference value for the current stock price optional five (buy five to sell five shares and the current price). Buy (sell) add: fill in the additional price, the unit is divided. The number of attempts: set in the number of attempts is still not completed the system to limit the transaction directly fill in the buy / sell order.
Output / post condition	Start trading——Do transactions automatically start to tune positions according to strategy settings and display transaction returns in the strategy log.

3.2Display transaction process

User scenarios	The user wants to see the details of the strategy for the transaction
Function description	Transaction returns in Display transaction process
Input / pre conditions	Set the transaction logic and run
Requirement description	 9/13/2017 6:37:38 PM: 成功选择策略 9/13/2017 6:37:45 PM: 尝试以89.06元买入股票002304 1048手 9/13/2017 6:37:45 PM: 成功以89.06元买入股票002304 1048手 9/13/2017 6:37:46 PM: 尝试以12.49元买入股票601611 3354手 9/13/2017 6:37:46 PM: 成功以12.49元买入股票601611 3354手 9/13/2017 6:37:47 PM: 尝试以14.63元买入股票300059 5714手 9/13/2017 6:37:47 PM: 成功以14.63元买入股票300059 5714手 9/13/2017 6:37:48 PM: 尝试以4.18元买入股票601988 6531手

	The strategy log below the strategy interface shows the date and time of the system transaction, the price of the change stock, the stock code and the direction and number of the change warehouse, and shows the system running status.
Output / post condition	

3.3Inquiry positions

User scenarios	The user runs the strategy and wants to see how it works during the transaction
Function description	Displays current portfolio positions
Input / pre conditions	The strategy parameters are set up and run successfully for transactions
Requirement description	 The transaction interface chart is divided into four blocks, the upper left corner is decided by the strategy target positions, namely the ideal portfolio, including the stock code, display

股票代码	比例(%)	交易所
601611	4.188403	SSE
600150	17.93751	SSE
002714	20.11519	SZE
002310	7.5458769999999991	SZE
600446	4.827906	SSE
002304	9.34386	SZE
002008	18.06044	SZE
601988	2.730488	SSE

股票代码	比例(%)	交易所
601611	4.19	SSE

股票代码	替代股1	替代股2	替代股3
601611	601718	600827	600820
600150	600685	600893	601718
002714	002385	600373	300133
002310	000725	600588	000008
600446	000938	600109	600718
002304	601021	600887	601118
002008	002241	600522	000538
601988	601939	601398	601166

股票代码	替代股1	替代股2	替代股3
601611	601718	600827	600820

	market value of the proportion of the investment portfolio and exchange, on the upper right corner has been buying positions and occupied the market value of the proportion of three. A replacement shares the lower left-hand corner of each of the underlying stocks, substitute shares lower right corner for holding stock.
Output / post condition	

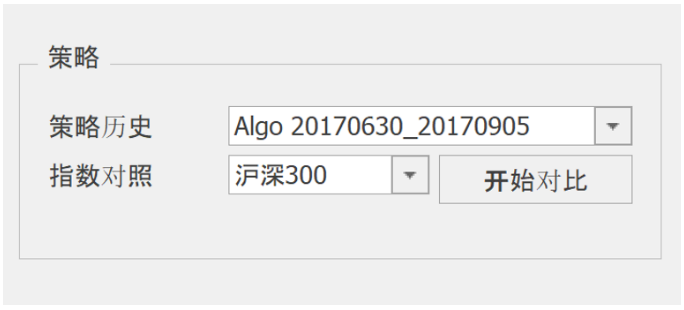
3.4Pause strategy

User scenarios	The user predicts the market risk or is not satisfied with the strategy
Function description	Unwinding and stopping strategic operations
Input / pre conditions	The strategy is running
Requirement description	Click the "start closing transaction interface button will hold all the stock to sell at the current price strategy, then stop selling strategy, process strategy is now below the exhibition log, show closing date and time, the selling price, stock code, number of hands.
Output / post	Unwind—Sold all stocks and stop the strategy.

condition	

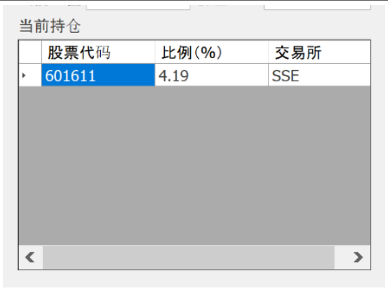
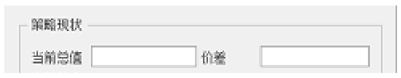
4．Monitor module

4.1 Historical contrast

User scenarios	Users have been running policies for a period of time to see strategy history
Function description	Displays the current market value of the transaction account
Input / pre conditions	The strategy runs effectively for a period of time and records the results of the operation
Requirement description	 策略 策略历史 Algo 20170630_20170905 指数对照 沪深300 开始对比 The user can choose to use that history to review the interface, or to set the index subject for reference.
Output / post condition	Start contrast - Show positions, current market capitalization, and historical benefit analysis

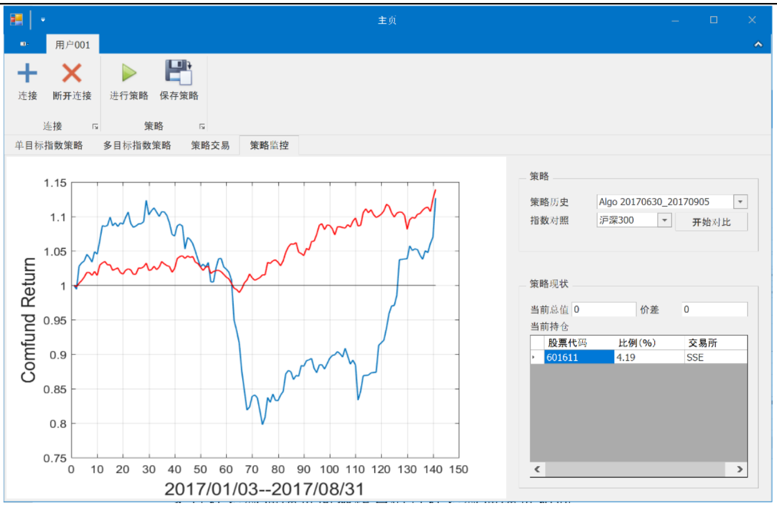
4.2Inquiry positions

User scenarios	View the current market value of the account

Function description	Shows the current market value and position of the trading account
Input / pre conditions	The strategy runs effectively for a period of time and records the results of the operation
Requirement description	 当前持仓 股票代码 | 比例(%) | 交易所 601611 | 4.19 | SSE  策略现状 当前总值 价差 This interface shows the current account of the user account, the stock code, the market capitalization ratio, and the stock exchange. It also shows the total market value of the current account, which directly reflects the operation of the system.
Output / post condition	

4.3 Historical yield analysis

User scenarios	View historical trends
Function description	The response strategy is expressed in the form of historical income line graphs
Input / pre conditions	The strategy runs effectively for a period of time and records the results of the operation

Requirement description	 主页 用户001 连接 断开连接 进行策略 保存策略 连接 策略 单目标指数策略 多目标指数策略 策略交易 策略监控 Comfund Return 2017/01/03--2017/08/31 开始对比 策略 策略历史 Algo 20170630_20170905 指数对照 沪深300 策略现状 当前总值 0 价差 0 当前持仓 股票代码 | 比例(%) | 交易所 601611 | 4.19 | SSE Selection strategy of the past history, and compared to an index, click start after the results comparison "appeared in the" strategies "and the left column of the figure, the blue line shows the composite yield trend, the red line represents as the control index of the trend of the two corresponding response strategies in the implementation stage in the past the relative performance of. The difference on the left shows the difference between the current strategy and the composite benefit, compared with the index chosen, which allows the user to monitor the effect of the strategy at any time, opening or unwinding at any time.
Output / post condition	

User Handbook

To run the product, first and foremost the user needs to click the “Connect” button on the top left corner of the product after opening it, in order to connect the market data server for more operation. This product supports two strategies – single target index strategy and multi-objective index strategy, the user can choose tab pages in the home page to change it. After the strategy is selected, the user needs to select the specific parameters of the strategy and the sub-strategies in the upper right corner of the tab, and then click the “Start” button to get the final result of the strategy.

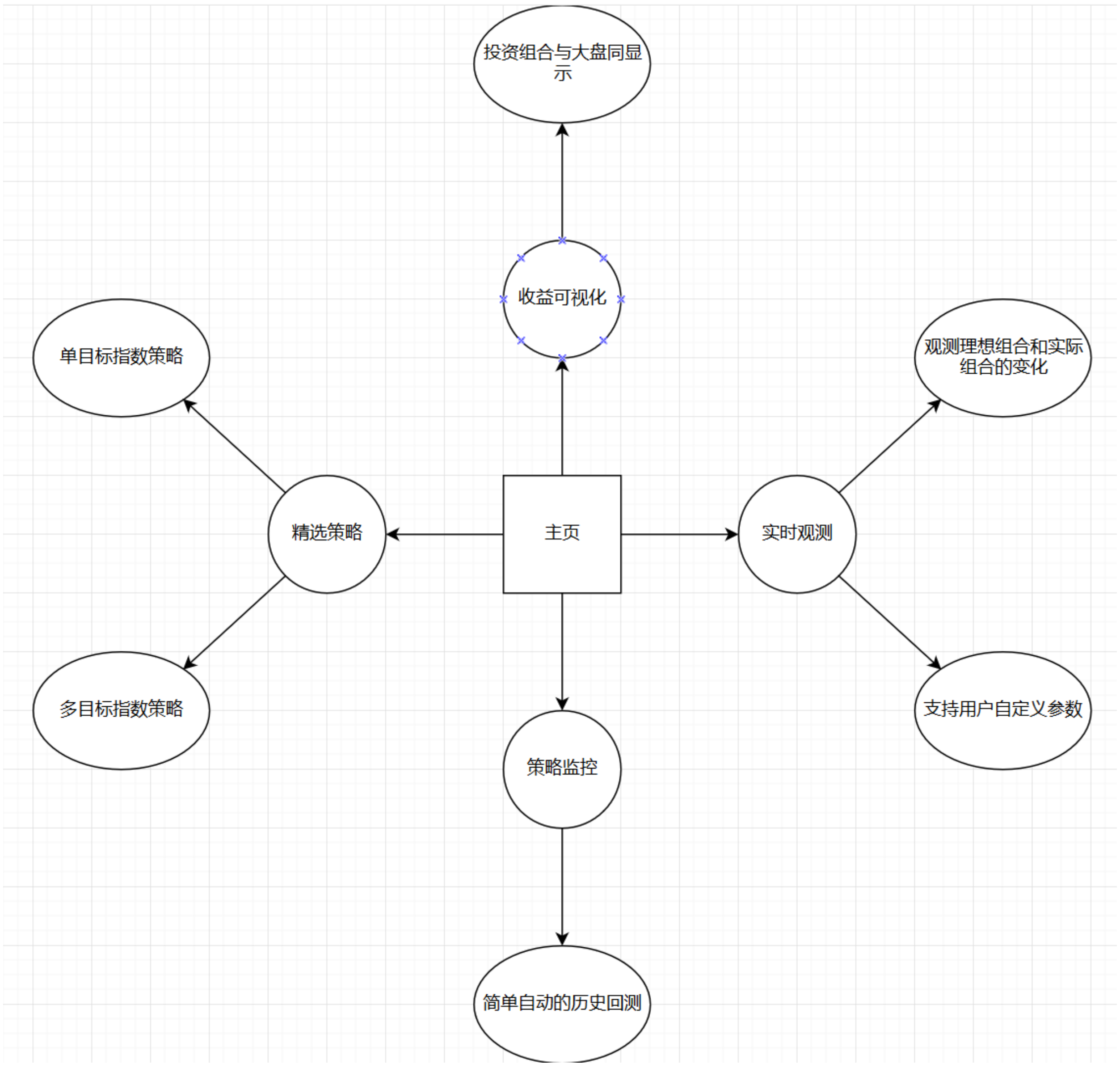
And then user can operate on artificial stock market. Click to enter the third tab, the user can firstly fill the various labels with the desired strategy trading logic, and then click the “Start trade” button to start trading. Users can see how their portfolio is generated from the main tables, and ultimately in accordance with a certain logic to buy their desired stocks.

In addition, the product can also store historical information to continue monitoring. In the strategy monitoring tab page, users can see through the line chart to get their former strategy’ s pros and cons or gain position situation and present value for the table.

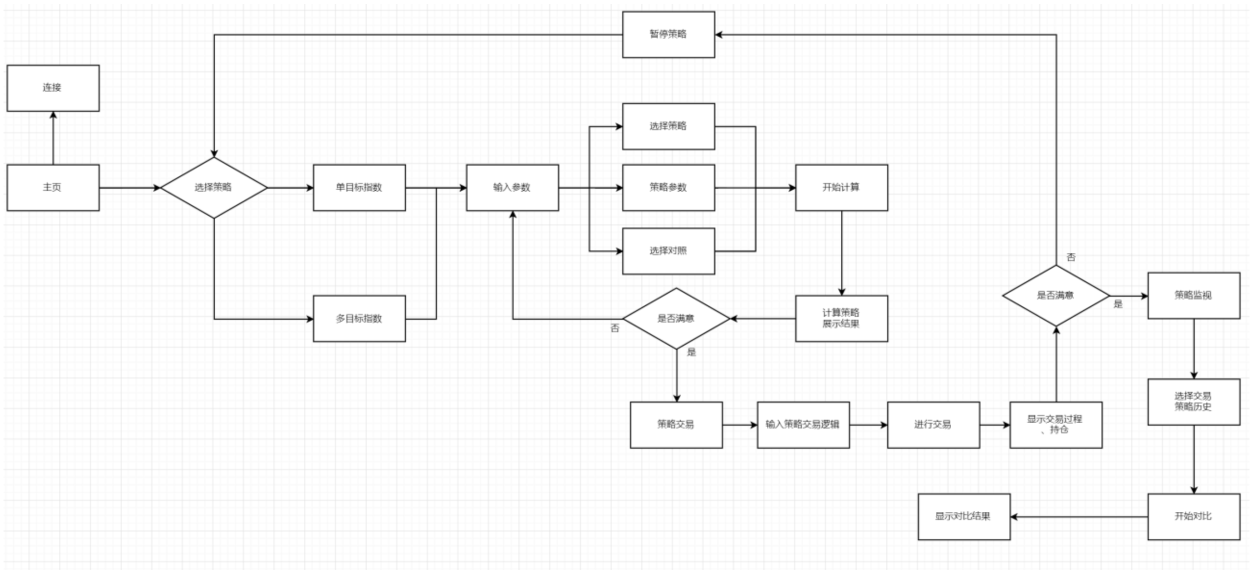
System Implement Catalogues

1. Functions:

Functions	Sub functions
Select strategy	Select sub strategy
	Select benchmark index
	Line chart of historical revenues
Trade	Edit trade logic
	Query position
Strategy watching	Query position
	Query present value
	Line chart of historical revenues


投资组合与大盘同显示
收益可视化
单目标指数策略
观测理想组合和实际组合的变化
精选策略
主页
实时观测
多目标指数策略
支持用户自定义参数
策略监控
简单自动的历史回测

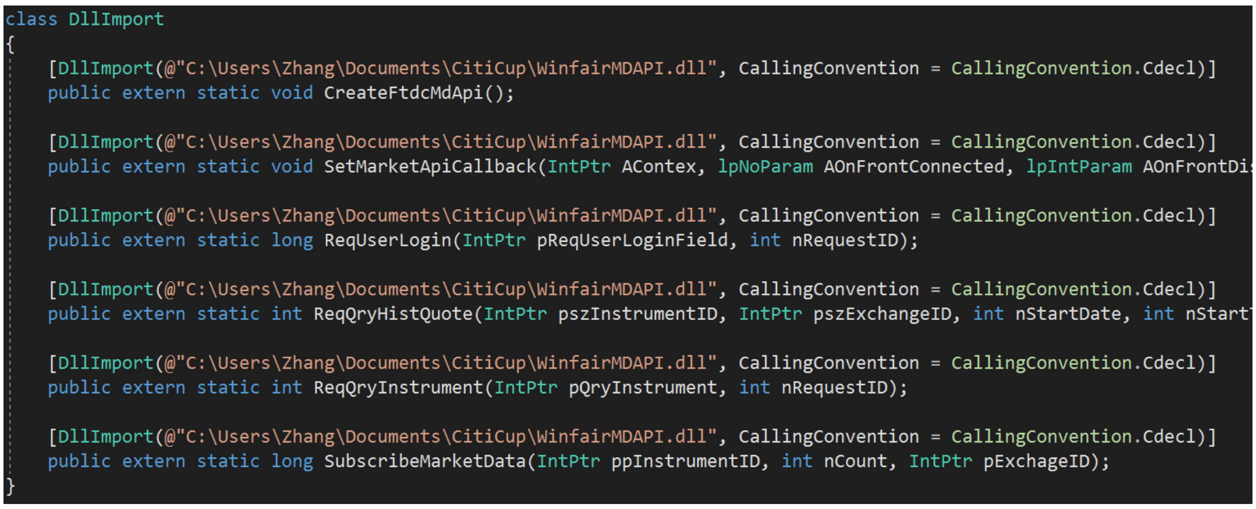
Functions



User Flow Diagram

2. Important Codes:

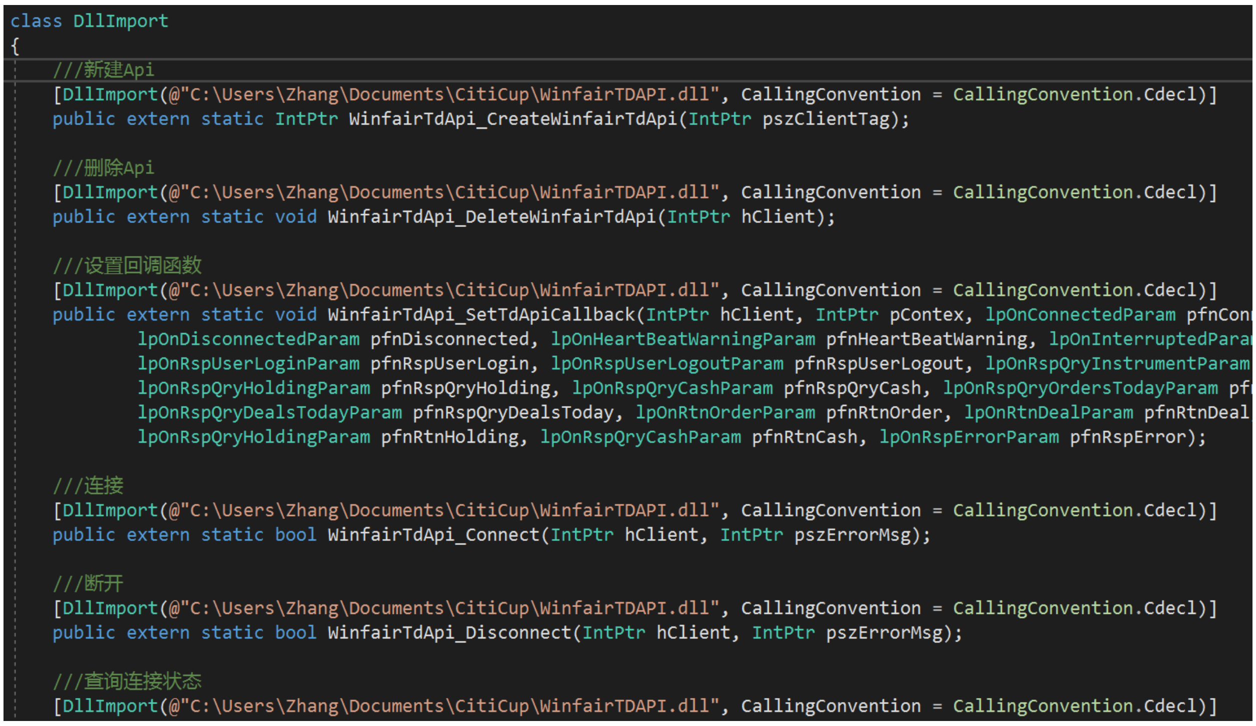
- Import WinfairMDAPI



```
class DllImport
{
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static void CreateFtdcMdApi();

    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static void SetMarketApiCallback(IntPtr AContex, lpNoParam AOnFrontConnected, lpIntParam AOnFrontDi

    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static long ReqUserLogin(IntPtr pReqUserLoginField, int nRequestID);

    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static int ReqQryHistQuote(IntPtr pszInstrumentID, IntPtr pszExchangeID, int nStartDate, int nStart

    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static int ReqQryInstrument(IntPtr pQryInstrument, int nRequestID);

    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairMDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static long SubscribeMarketData(IntPtr ppInstrumentID, int nCount, IntPtr pExchageID);
}
```

Use DllImport importing WinfairMDAPI to get market data.

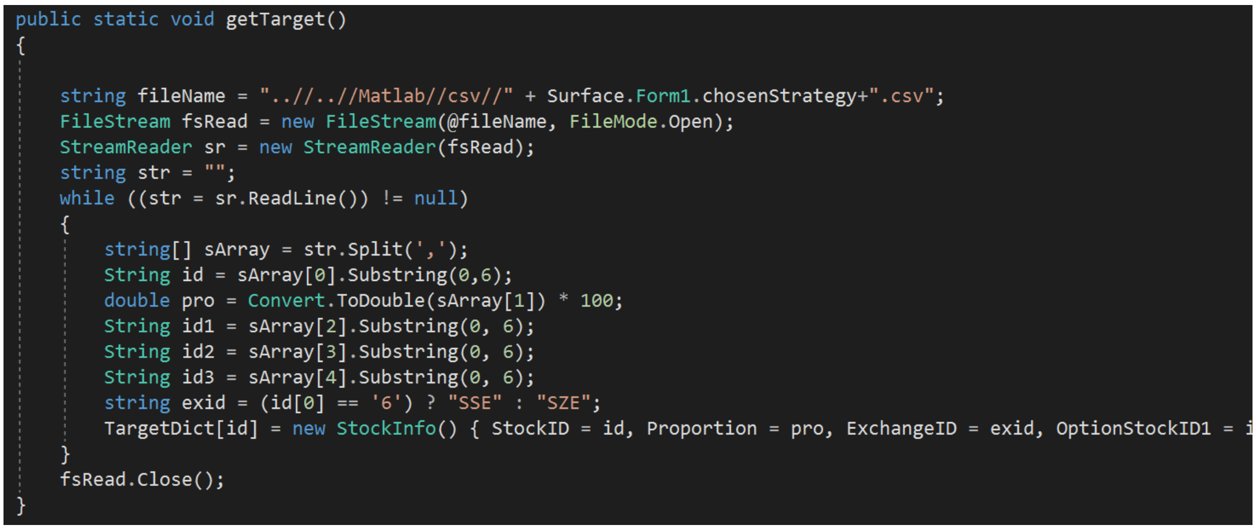
- Import WinfairTDAPI



```
class DllImport
{
    ///新建Api
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static IntPtr WinfairTdApi_CreateWinfairTdApi(IntPtr pszClientTag);

    ///删除Api
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static void WinfairTdApi_DeleteWinfairTdApi(IntPtr hClient);

    ///设置回调函数
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static void WinfairTdApi_SetTdApiCallback(IntPtr hClient, IntPtr pContex, lpOnConnectedParam pfnCon
            lpOnDisconnectedParam pfnDisconnected, lpOnHeartBeatWarningParam pfnHeartBeatWarning, lpOnInterruptedPara
            lpOnRspUserLoginParam pfnRspUserLogin, lpOnRspUserLogoutParam pfnRspUserLogout, lpOnRspQryInstrumentParam
            lpOnRspQryHoldingParam pfnRspQryHolding, lpOnRspQryCashParam pfnRspQryCash, lpOnRspQryOrdersTodayParam pf
            lpOnRspQryDealsTodayParam pfnRspQryDealsToday, lpOnRtnOrderParam pfnRtnOrder, lpOnRtnDealParam pfnRtnDeal
            lpOnRspQryHoldingParam pfnRtnHolding, lpOnRspQryCashParam pfnRtnCash, lpOnRspErrorParam pfnRspError);

    ///连接
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static bool WinfairTdApi_Connect(IntPtr hClient, IntPtr pszErrorMsg);

    ///断开
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
    public extern static bool WinfairTdApi_Disconnect(IntPtr hClient, IntPtr pszErrorMsg);

    ///查询连接状态
    [DllImport(@"C:\Users\Zhang\Documents\CitiCup\WinfairTDAPI.dll", CallingConvention = CallingConvention.Cdecl)]
```

Use DllImport importing WinfairTDAPI to get trade data.

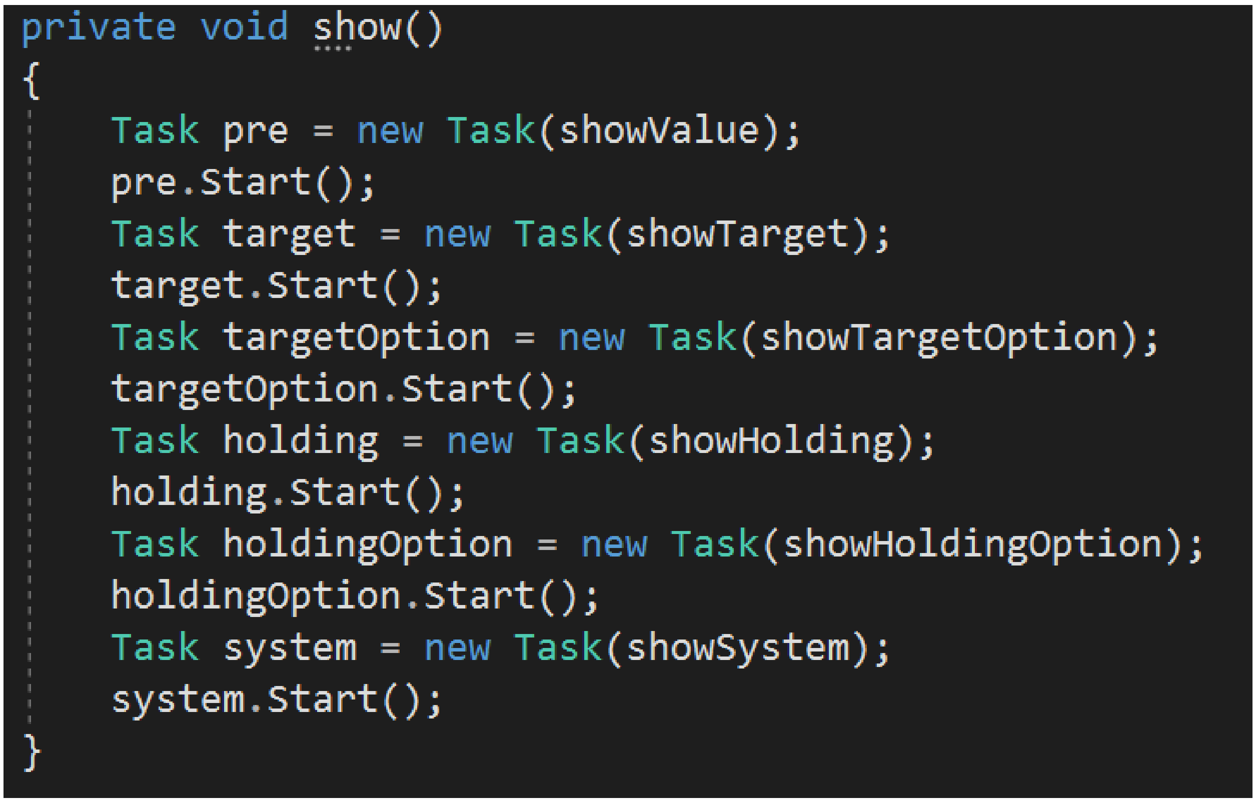
- Import strategy



```
public static void getTarget()
{

    string fileName = "..//..//Matlab//csv//" + Surface.Form1.chosenStrategy+".csv";
    FileStream fsRead = new FileStream(@fileName, FileMode.Open);
    StreamReader sr = new StreamReader(fsRead);
    string str = "";
    while ((str = sr.ReadLine()) != null)
    {
        string[] sArray = str.Split(',');
        String id = sArray[0].Substring(0,6);
        double pro = Convert.ToDouble(sArray[1]) * 100;
        String id1 = sArray[2].Substring(0, 6);
        String id2 = sArray[3].Substring(0, 6);
        String id3 = sArray[4].Substring(0, 6);
        string exid = (id[0] == '6') ? "SSE" : "SZE";
        TargetDict[id] = new StockInfo() { StockID = id, Proportion = pro, ExchangeID = exid, OptionStockID1 = i
    }
    fsRead.Close();
}
```

Use csv to get target portfolio.

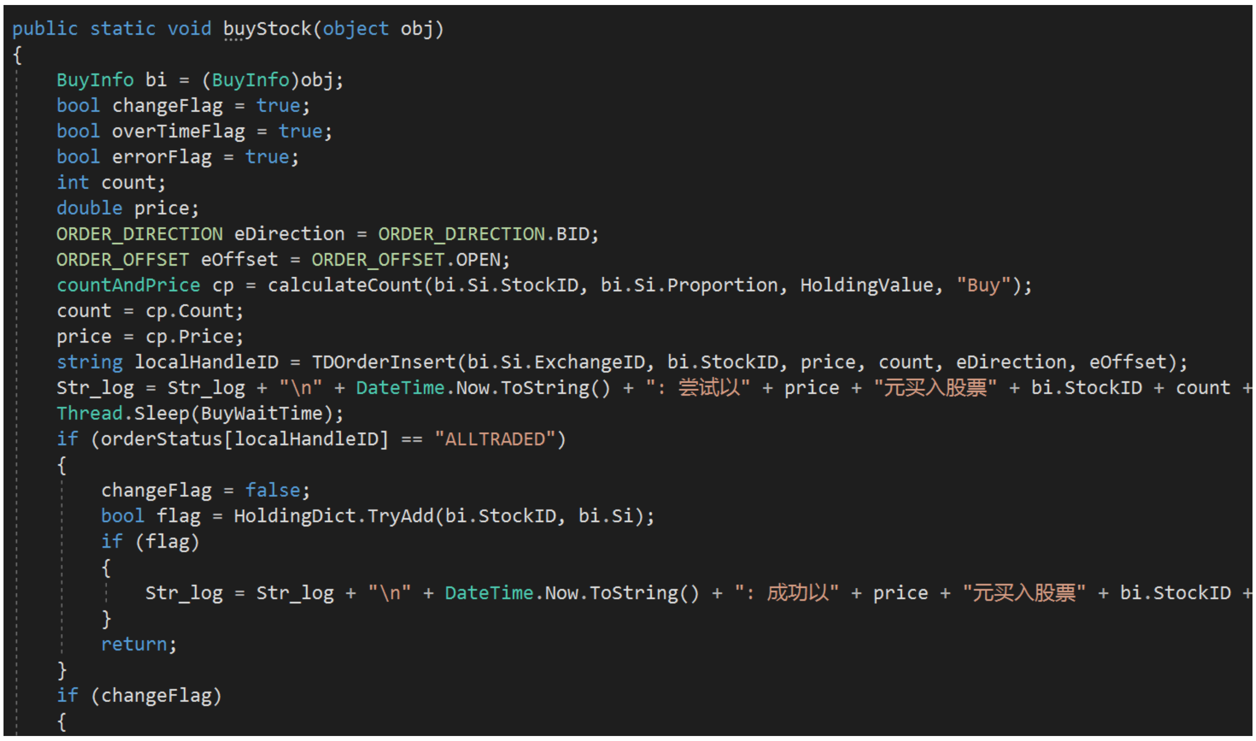
- Refresh log periodically



```
private void show()
{
    Task pre = new Task(showValue);
    pre.Start();
    Task target = new Task(showTarget);
    target.Start();
    Task targetOption = new Task(showTargetOption);
    targetOption.Start();
    Task holding = new Task(showHolding);
    holding.Start();
    Task holdingOption = new Task(showHoldingOption);
    holdingOption.Start();
    Task system = new Task(showSystem);
    system.Start();
}
```

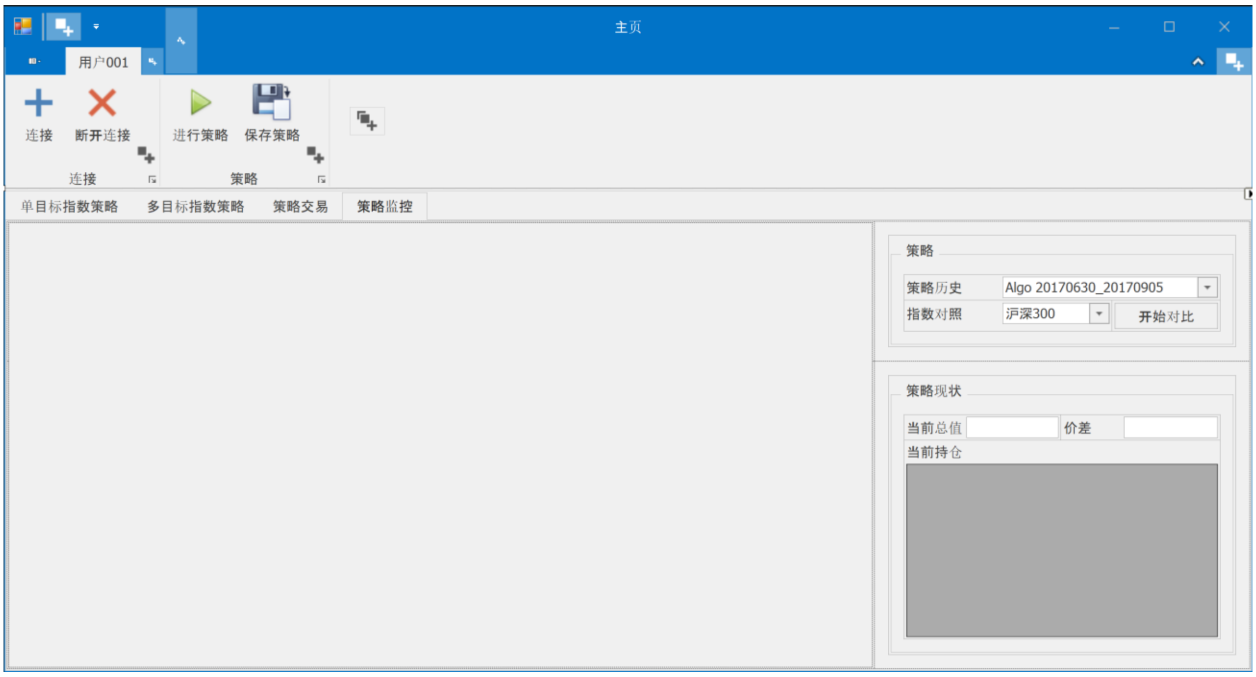
Use multi-task to refresh periodically.

- Change holding



```
public static void buyStock(object obj)
{
    BuyInfo bi = (BuyInfo)obj;
    bool changeFlag = true;
    bool overTimeFlag = true;
    bool errorFlag = true;
    int count;
    double price;
    ORDER_DIRECTION eDirection = ORDER_DIRECTION.BID;
    ORDER_OFFSET eOffset = ORDER_OFFSET.OPEN;
    countAndPrice cp = calculateCount(bi.Si.StockID, bi.Si.Proportion, HoldingValue, "Buy");
    count = cp.Count;
    price = cp.Price;
    string localHandleID = TDOrderInsert(bi.Si.ExchangeID, bi.StockID, price, count, eDirection, eOffset);
    Str_log = Str_log + "\n" + DateTime.Now.ToString() + ": 尝试以" + price + "元买入股票" + bi.StockID + count +
    Thread.Sleep(BuyWaitTime);
    if (orderStatus[localHandleID] == "ALLTRADED")
    {
        changeFlag = false;
        bool flag = HoldingDict.TryAdd(bi.StockID, bi.Si);
        if (flag)
        {
            Str_log = Str_log + "\n" + DateTime.Now.ToString() + ": 成功以" + price + "元买入股票" + bi.StockID +
        }
        return;
    }
    if (changeFlag)
    {
```

3. Main Frame:

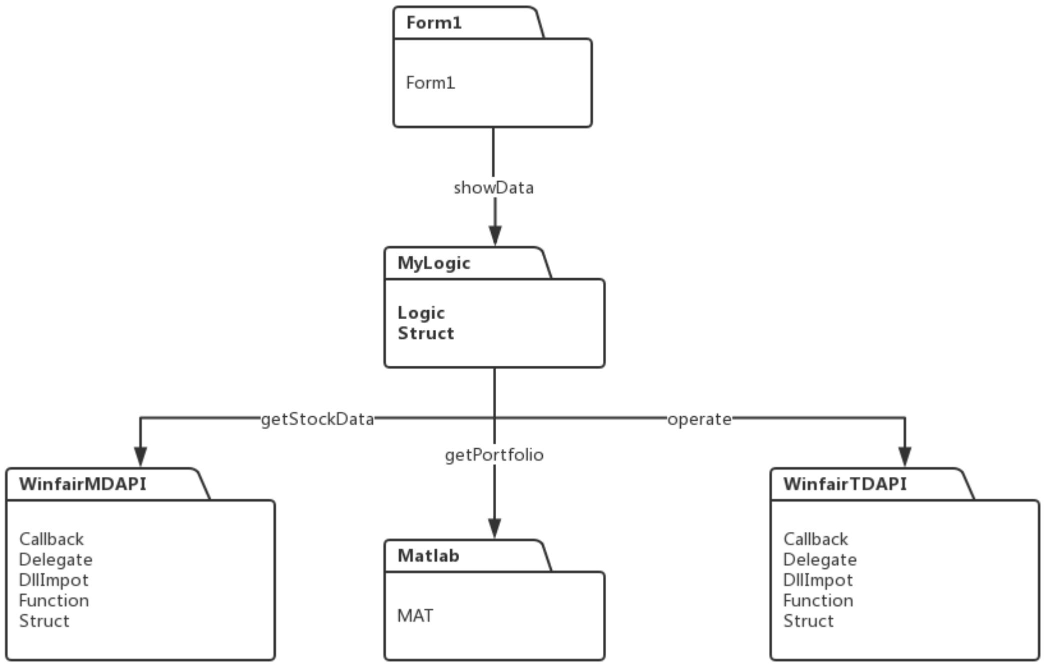

主页
用户001
连接
断开连接
进行策略
保存策略
连接
策略
单目标指数策略
多目标标指数策略
策略交易
策略监控
策略
策略历史
Algo 20170630_20170905
指数对照
沪深300
开始对比
策略现状
当前总值
价差
当前持仓

项目名称

UML model report

Table of Content

1. The overall structure of the system



The structure of the software is divided into five parts, respectively, for the market-related folder WinfairMDAPI, interactive folder with Matlab, order operation folder WinfairTDAPI, transaction logic folder MyLogic and interface file Form1. This section will be covered in detail below.

① WinfairMDAPI: This section is mainly responsible for obtaining and processing stock data through the market value interface. The processed data is transferred to the Matlab section and the transaction logic section.

② Matlab: This part is mainly responsible for the acquisition of stock data to Matlab code algorithm part. It is also responsible for transferring the algorithm's portfolio to the trading logic.

③ WinfairTDAPI: This part is mainly responsible for the implementation of the transaction through the order interface, check the order situation. And the trading logic part has a strong interaction.

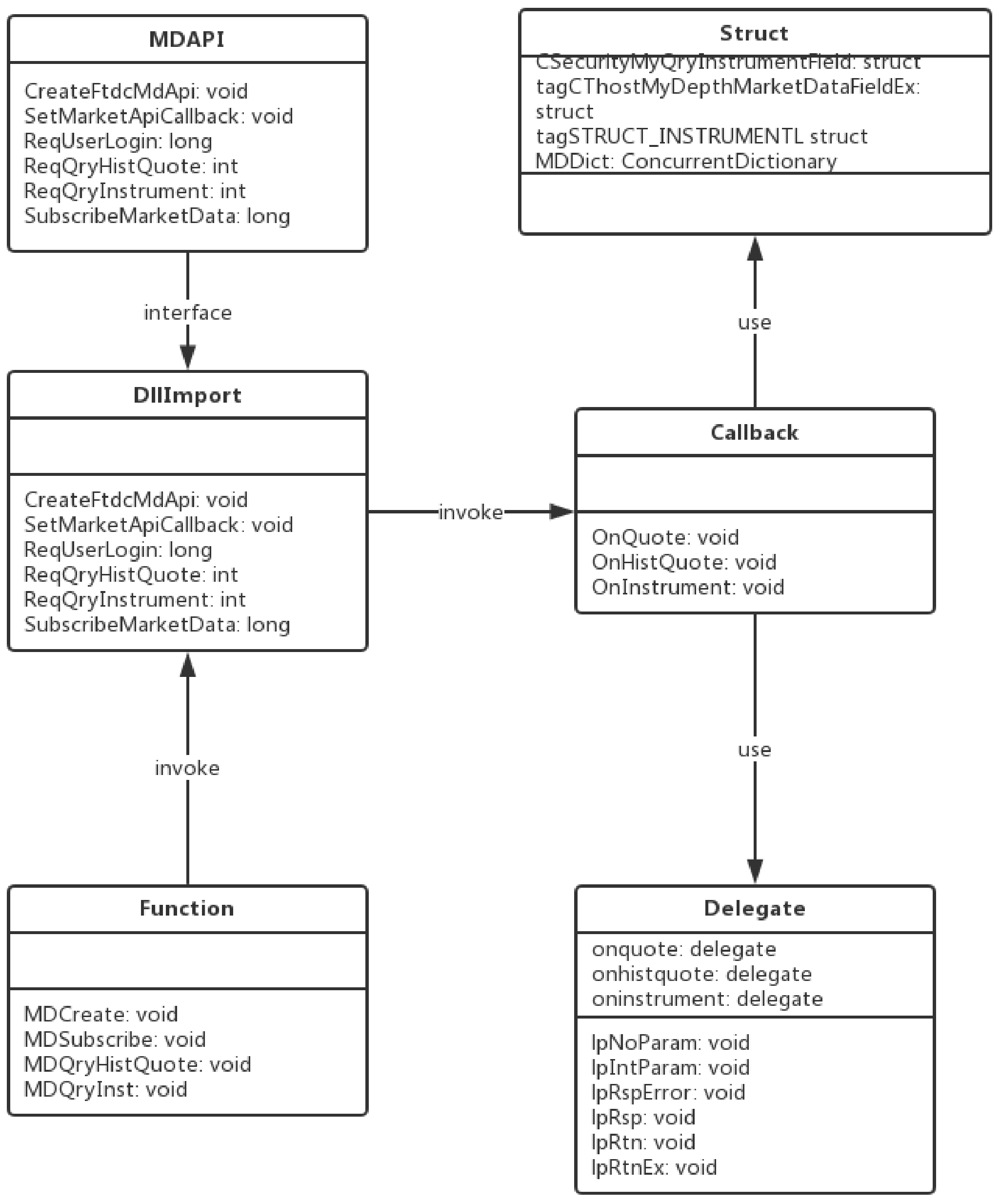
④ MyLogic: trading logic part is responsible for judging when to buy and sell, control the threshold of the sale. Get the portfolio from the Matlab section and perform specific transaction operations through the WinfairTDAPI section. At the same time, the part also needs to be provided by WinfairMDAPI stock data, combined with the portfolio to calculate the required market value.

⑤ Form1: interface display part of the main part of the transaction is responsible for displaying the logical part of the market value of the data and the various parts of the tips, such as the completion of the connection. As a display module, the click event of each button and the trigger event of the display control are included.

2. Class diagram design of every function model

The following function modules as a unit, the specific description of the software class diagram design.

2.1 Function one (obtain data about stock market)



The Get Stock Data module is implemented primarily through the DllImport class, the Function class, the Callback class, the Delegate class, the Struct class, and the MD interface under the WinfairMDAPI folder. Each of the following categories were described in detail.

① MDAPI: MD interface written by C / C + +, the software through the Dll and delegate call the asynchronous interface. The specific functions of the interface will be described later.

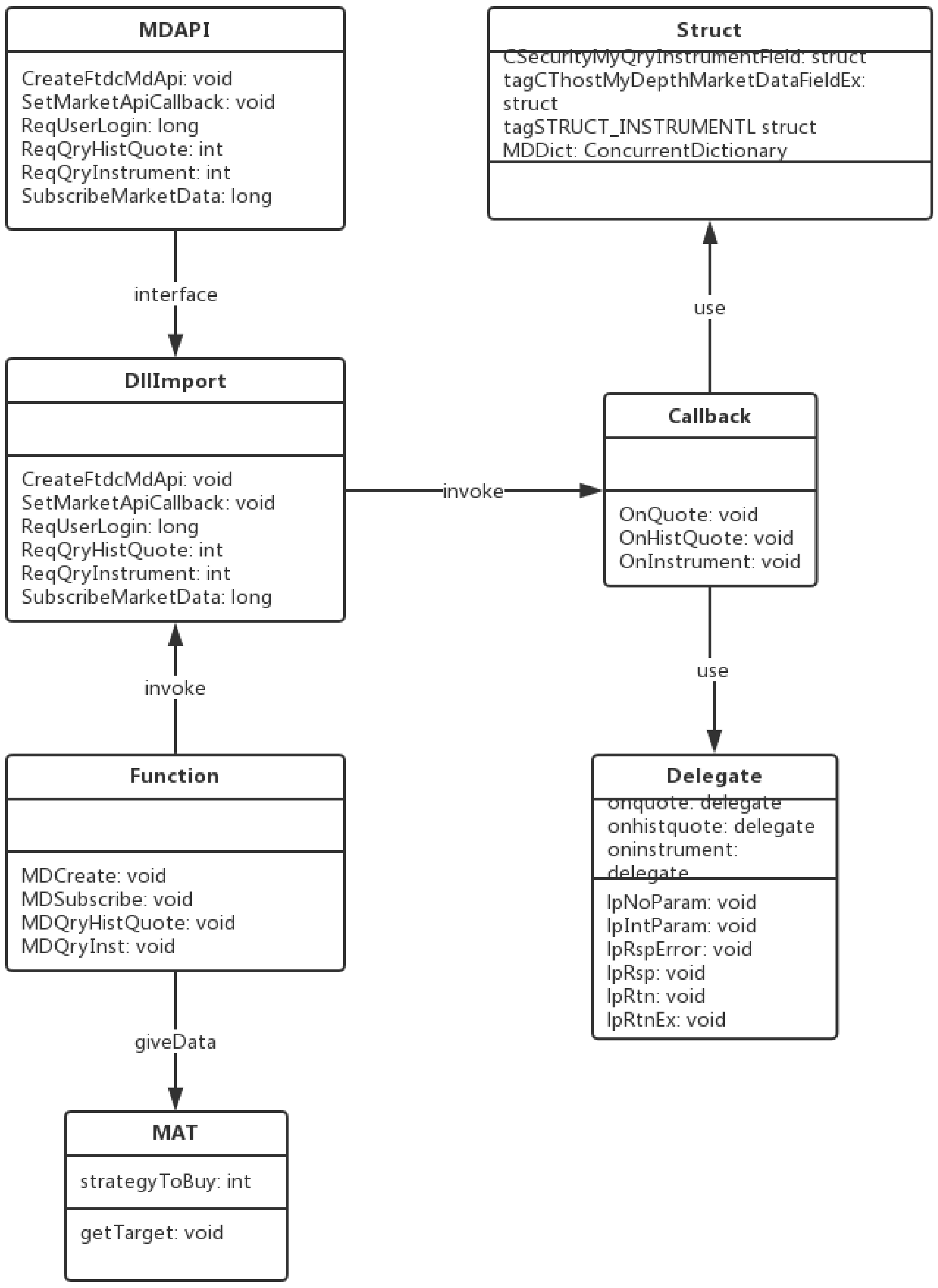
② DllImport: The main role of the class is to call the interface required Dll into the namespace and the required method to declare.

③ Delegate: This class defines and instantiates the delegates that C # calls C / C ++ to use.

④ Callback: This class defines the various callback functions when the interface is invoked asynchronously.

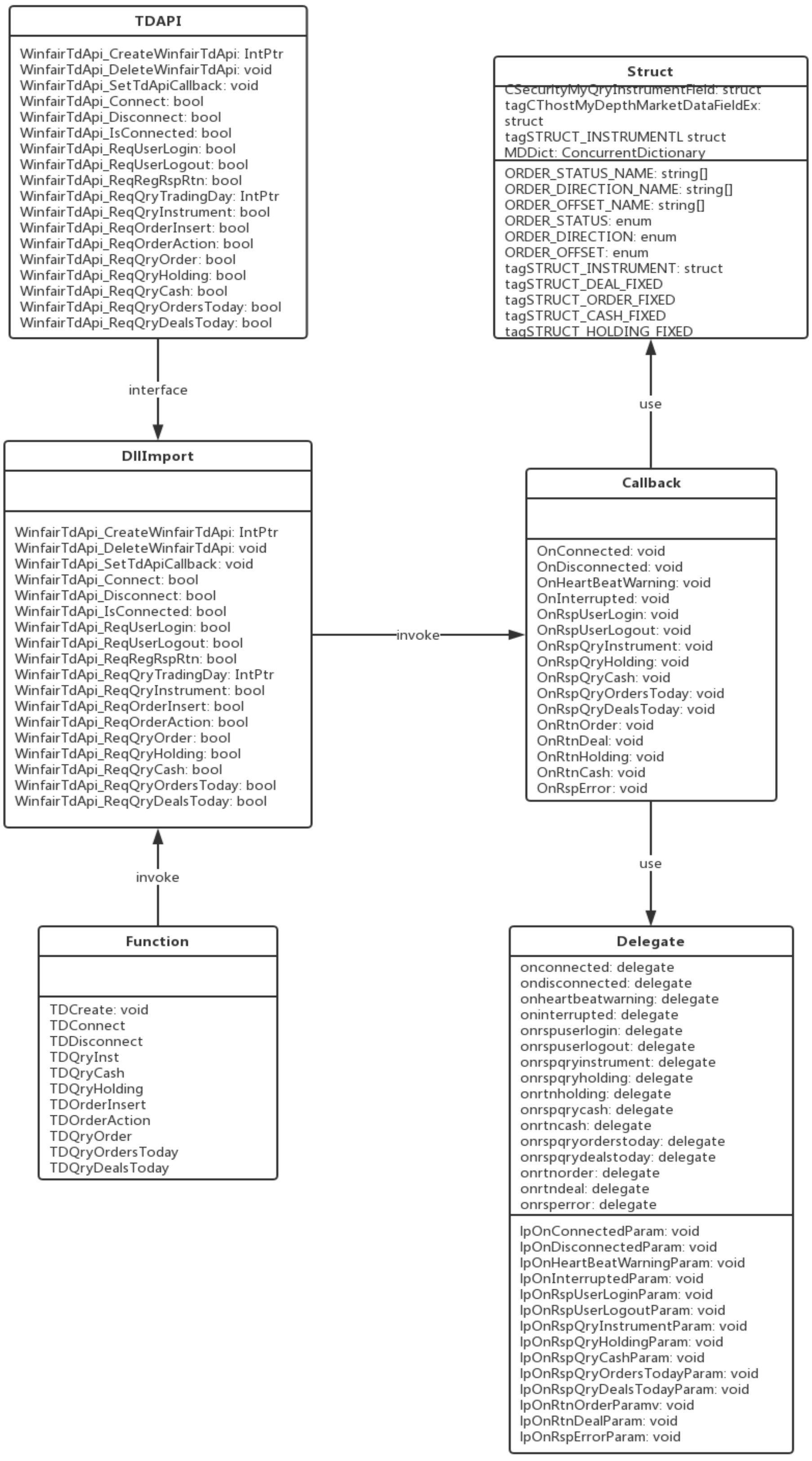
⑤ Struct: The main function of this class is the storage interface provides a variety of structural data.

2.2 Function two (obtain investing portfolio)



The function module is partially completed by the Matlab part and the acquisition of the stock data. The latter provides the stock data needed for the execution of the algorithm. And then Mat class calls Matlab code into Dll, in order to achieve the acquisition of the portfolio.

2.3 Function three (take order operations)


TDAPI
WinfairTdApi_CreateWinfairTdApi: IntPtr
WinfairTdApi_DeleteWinfairTdApi: void
WinfairTdApi_SetTdApiCallback: void
WinfairTdApi_Connect: bool
WinfairTdApi_Disconnect: bool
WinfairTdApi_IsConnected: bool
WinfairTdApi_ReqUserLogin: bool
WinfairTdApi_ReqUserLogout: bool
WinfairTdApi_ReqRegRspRtn: bool
WinfairTdApi_ReqQryTradingDay: IntPtr
WinfairTdApi_ReqQryInstrument: bool
WinfairTdApi_ReqOrderInsert: bool
WinfairTdApi_ReqOrderAction: bool
WinfairTdApi_ReqQryOrder: bool
WinfairTdApi_ReqQryHolding: bool
WinfairTdApi_ReqQryCash: bool
WinfairTdApi_ReqQryOrdersToday: bool
WinfairTdApi_ReqQryDealsToday: bool
interface
Struct
CSecurityMyQryInstrumentField: struct
tagCThostMyDepthMarketDataFieldEx: struct
tagSTRUCT_INSTRUMENTL struct
MDDict: ConcurrentDictionary
ORDER_STATUS_NAME: string[]
ORDER_DIRECTION_NAME: string[]
ORDER_OFFSET_NAME: string[]
ORDER_STATUS: enum
ORDER_DIRECTION: enum
ORDER_OFFSET: enum
tagSTRUCT_INSTRUMENT: struct
tagSTRUCT_DEAL_FIXED
tagSTRUCT_ORDER_FIXED
tagSTRUCT_CASH_FIXED
tagSTRUCT_HOLDING_FIXED
use
DllImport
WinfairTdApi_CreateWinfairTdApi: IntPtr
WinfairTdApi_DeleteWinfairTdApi: void
WinfairTdApi_SetTdApiCallback: void
WinfairTdApi_Connect: bool
WinfairTdApi_Disconnect: bool
WinfairTdApi_IsConnected: bool
WinfairTdApi_ReqUserLogin: bool
WinfairTdApi_ReqUserLogout: bool
WinfairTdApi_ReqRegRspRtn: bool
WinfairTdApi_ReqQryTradingDay: IntPtr
WinfairTdApi_ReqQryInstrument: bool
WinfairTdApi_ReqOrderInsert: bool
WinfairTdApi_ReqOrderAction: bool
WinfairTdApi_ReqQryOrder: bool
WinfairTdApi_ReqQryHolding: bool
WinfairTdApi_ReqQryCash: bool
WinfairTdApi_ReqQryOrdersToday: bool
WinfairTdApi_ReqQryDealsToday: bool
invoke
Callback
OnConnected: void
OnDisconnected: void
OnHeartBeatWarning: void
OnInterrupted: void
OnRspUserLogin: void
OnRspUserLogout: void
OnRspQryInstrument: void
OnRspQryHolding: void
OnRspQryCash: void
OnRspQryOrdersToday: void
OnRspQryDealsToday: void
OnRtnOrder: void
OnRtnDeal: void
OnRtnHolding: void
OnRtnCash: void
OnRspError: void
invoke
use
Function
TDCreate: void
TDConnect
TDDisconnect
TDQryInst
TDQryCash
TDQryHolding
TDOrderInsert
TDOrderAction
TDQryOrder
TDQryOrdersToday
TDQryDealsToday
Delegate
onconnected: delegate
ondisconnected: delegate
onheartbeatwarning: delegate
oninterrupted: delegate
onrspuserlogin: delegate
onrspuserlogout: delegate
onrspqryinstrument: delegate
onrspqryholding: delegate
onrtnholding: delegate
onrspqrycash: delegate
onrtncash: delegate
onrspqryorderstoday: delegate
onrspqrydealstoday: delegate
onrtnorder: delegate
onrtndeal: delegate
onrsperror: delegate
lpOnConnectedParam: void
lpOnDisconnectedParam: void
lpOnHeartBeatWarningParam: void
lpOnInterruptedParam: void
lpOnRspUserLoginParam: void
lpOnRspUserLogoutParam: void
lpOnRspQryInstrumentParam: void
lpOnRspQryHoldingParam: void
lpOnRspQryCashParam: void
lpOnRspQryOrdersTodayParam: void
lpOnRspQryDealsTodayParam: void
lpOnRtnOrderParamv: void
lpOnRtnDealParam: void
lpOnRspErrorParam: void

The execution order operation module is implemented mainly through the DllImport class, the Function class, the Callback class, the Delegate class, the Struct class, and the TD interface under the WinfairTDAPI folder. Each of the following categories were described in detail.

① TDAPI: MD interface written by C / C + +, the software through the Dll and delegate call the asynchronous interface. The specific functions of the interface will be described later.

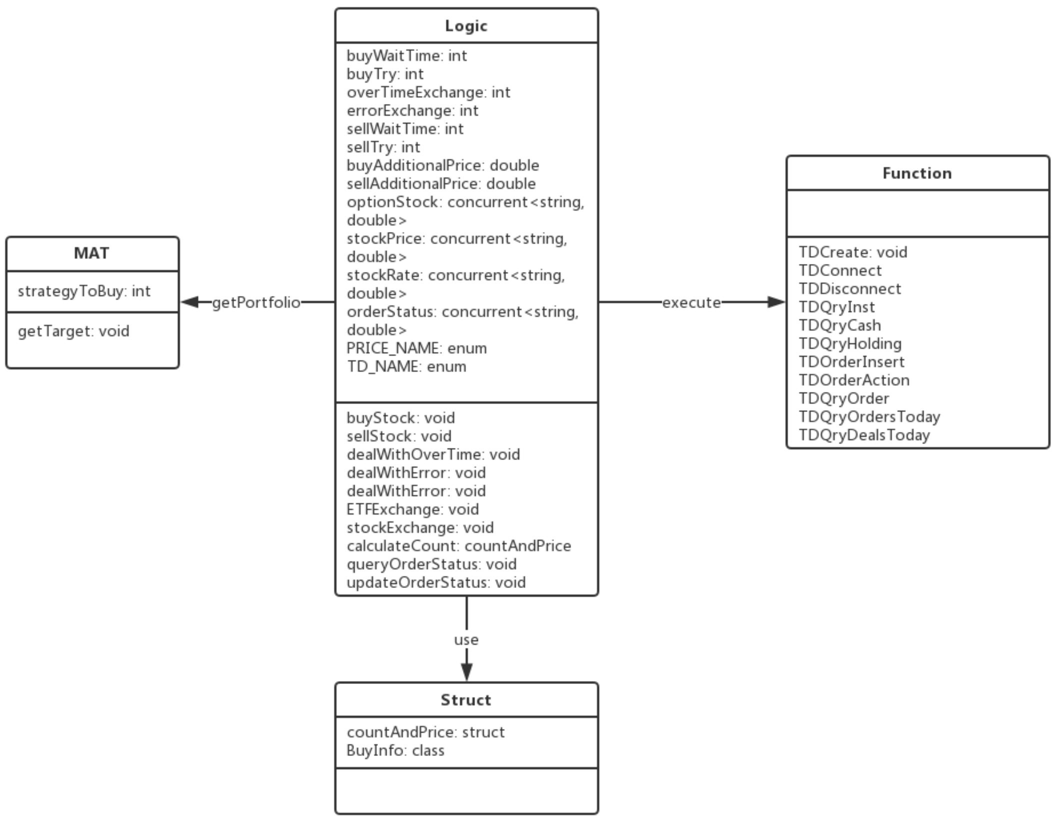
② DllImport: The main role of the class is to call the interface required Dll into the namespace and the required method to declare.

③ Delegate: This class defines and instantiates the delegates that C # calls C / C ++ to use.

④ Callback: This class defines the various callback functions when the interface is invoked asynchronously.

⑤ Struct: The main function of this class is the storage interface provides a variety of structural data.

2.4 Function four (perform trading logic)



The function module involves the transaction logic part, the Matlab part and the implementation of the transaction operation part, the three parts of the interaction. Specific steps are as follows:

① The Logic class gets the portfolio-related data through the MAT class.

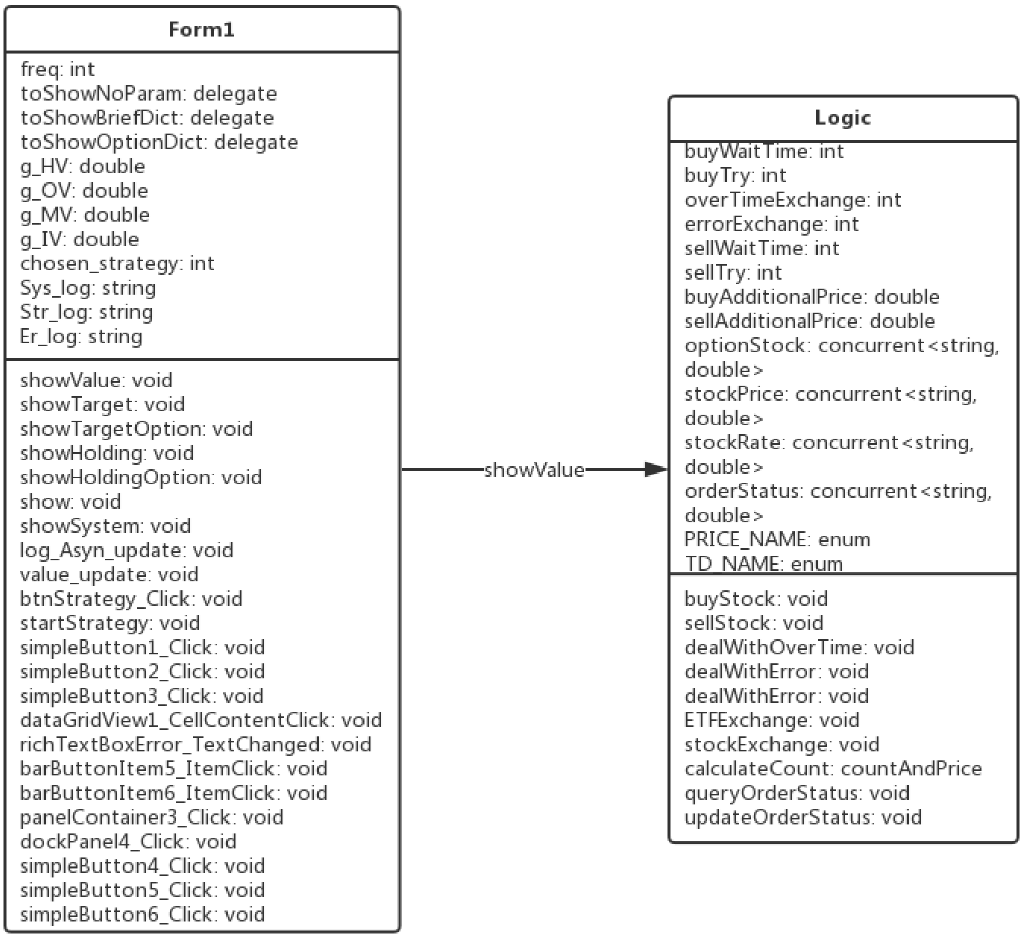
② through the relevant formula to calculate the combined market value and buy the market value after the sale to determine whether it exceeds the threshold of buying or selling, and thus decide whether to buy.

③ If you need to buy or sell orders and other operations, Logic class need to call the implementation of the transaction part of the operation method, the order to buy or sell.

④ After buying or selling, the Logic class still needs to track the order status in real time through the method in the Function to determine whether the order is completed or not to respond differently.

⑤ If the order transaction times out or unsuccessful, the Logic class will make the appropriate price adjustment, or after a certain period of time to continue the transaction operation.

2.5 Function five (display)



The functional module is responsible for all the visualization of the entire software. It includes visualization of various market values, visualization of portfolios, visualization of user customization options, prompting information, and visualization of error messages. The following will be described in this part of the visualization were described.

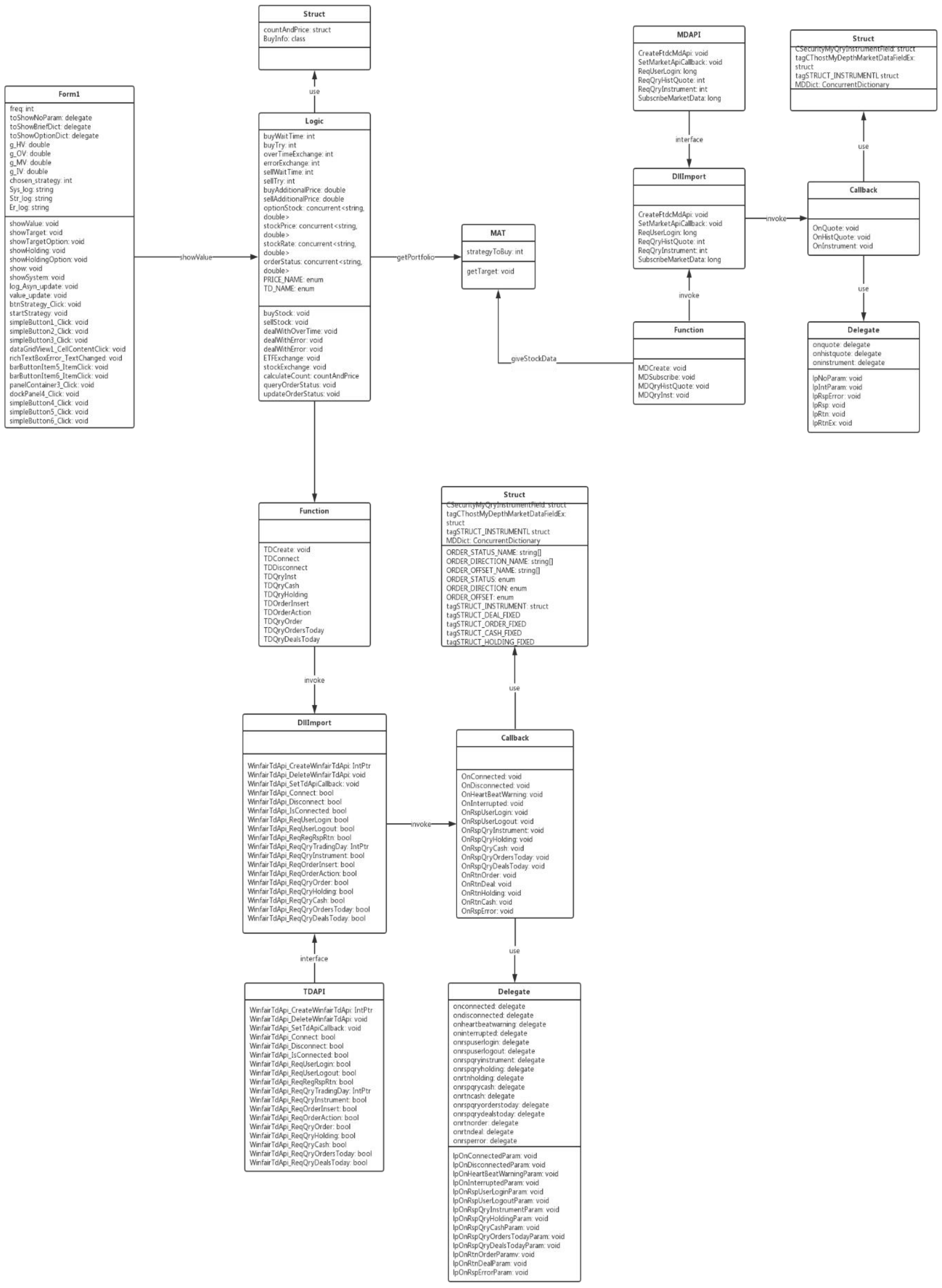
① visualization of various market values: by calling the Logic class in the market value of data, combined with a variety of display methods, the user needs a variety of market value data display,

② Portfolio visualization: the principle and the same, but also through the transaction logic part of the interaction, access to portfolio data and display.

③ user-defined options visualization: the user can customize the transaction time through the interface, trading strategies.

④ prompt information and error information visualization: in all parts of the software, will produce some tips and error messages. For example: interface connection is successful, order insertion is successful, transaction timeout. The Form1 class collects this information and displays it through the corresponding display method.

3. System class diagram design


Struct
countAndPrice: struct
BuyInfo: class
use
Form1
freq: int
toShowNoParam: delegate
toShowBriefDict: delegate
toShowOptionDict: delegate
g_HV: double
g_OV: double
g_MV: double
g_IV: double
chosen_strategy: int
Sys_log: string
Str_log: string
Er_log: string
showValue: void
showTarget: void
showTargetOption: void
showHolding: void
showHoldingOption: void
show: void
showSystem: void
log_Asyn_update: void
value_update: void
btnStrategy_Click: void
startStrategy: void
simpleButton1_Click: void
simpleButton2_Click: void
simpleButton3_Click: void
dataGridView1_CellContentClick: void
richTextBoxError_TextChanged: void
barButtonItem5_ItemClick: void
barButtonItem6_ItemClick: void
panelContainer3_Click: void
dockPanel4_Click: void
simpleButton4_Click: void
simpleButton5_Click: void
simpleButton6_Click: void
showValue
Logic
buyWaitTime: int
buyTry: int
overTimeExchange: int
errorExchange: int
sellWaitTime: int
sellTry: int
buyAdditionalPrice: double
sellAdditionalPrice: double
optionStock: concurrent<string, double>
stockPrice: concurrent<string, double>
stockRate: concurrent<string, double>
orderStatus: concurrent<string, double>
PRICE_NAME: enum
TD_NAME: enum
buyStock: void
sellStock: void
dealWithOverTime: void
dealWithError: void
dealWithError: void
ETFExchange: void
stockExchange: void
calculateCount: countAndPrice
queryOrderStatus: void
updateOrderStatus: void
getPortfolio
MAT
strategyToBuy: int
getTarget: void
giveStockData
MDAPI
CreateFtdcMdApi: void
SetMarketApiCallback: void
ReqUserLogin: long
ReqQryHistQuote: int
ReqQryInstrument: int
SubscribeMarketData: long
interface
DllImport
CreateFtdcMdApi: void
SetMarketApiCallback: void
ReqUserLogin: long
ReqQryHistQuote: int
ReqQryInstrument: int
SubscribeMarketData: long
invoke
Function
MDCreate: void
MDSubscribe: void
MDQryHistQuote: void
MDQryInst: void
Struct
CSecurityMyQryInstrumentField: struct
tagCThostMyDepthMarketDataFieldEx: struct
tagSTRUCT_INSTRUMENTL struct
MDDict: ConcurrentDictionary
use
invoke
Callback
OnQuote: void
OnHistQuote: void
OnInstrument: void
use
Delegate
onquote: delegate
onhistquote: delegate
oninstrument: delegate
lpNoParam: void
lpIntParam: void
lpRspError: void
lpRsp: void
lpRtn: void
lpRtnEx: void
Function
TDCreate: void
TDConnect
TDDisconnect
TDQryInst
TDQryCash
TDQryHolding
TDOrderInsert
TDOrderAction
TDQryOrder
TDQryOrdersToday
TDQryDealsToday
Struct
CSecurityMyQryInstrumentfield: struct
tagCThostMyDepthMarketDataFieldEx: struct
tagSTRUCT_INSTRUMENTL struct
MDDict: ConcurrentDictionary
ORDER_STATUS_NAME: string[]
ORDER_DIRECTION_NAME: string[]
ORDER_OFFSET_NAME: string[]
ORDER_STATUS: enum
ORDER_DIRECTION: enum
ORDER_OFFSET: enum
tagSTRUCT_INSTRUMENT: struct
tagSTRUCT_DEAL_FIXED
tagSTRUCT_ORDER_FIXED
tagSTRUCT_CASH_FIXED
tagSTRUCT_HOLDING_FIXED
invoke
use
DllImport
WinfairTdApi_CreateWinfairTdApi: IntPtr
WinfairTdApi_DeleteWinfairTdApi: void
WinfairTdApi_SetTdApiCallback: void
WinfairTdApi_Connect: bool
WinfairTdApi_Disconnect: bool
WinfairTdApi_IsConnected: bool
WinfairTdApi_ReqUserLogin: bool
WinfairTdApi_ReqUserLogout: bool
WinfairTdApi_ReqRegRspRtn: bool
WinfairTdApi_ReqQryTradingDay: IntPtr
WinfairTdApi_ReqQryInstrument: bool
WinfairTdApi_ReqOrderInsert: bool
WinfairTdApi_ReqOrderAction: bool
WinfairTdApi_ReqQryOrder: bool
WinfairTdApi_ReqQryHolding: bool
WinfairTdApi_ReqQryCash: bool
WinfairTdApi_ReqQryOrdersToday: bool
WinfairTdApi_ReqQryDealsToday: bool
invoke
Callback
OnConnected: void
OnDisconnected: void
OnHeartBeatWarning: void
OnInterrupted: void
OnRspUserLogin: void
OnRspUserLogout: void
OnRspQryInstrument: void
OnRspQryHolding: void
OnRspQryCash: void
OnRspQryOrdersToday: void
OnRspQryDealsToday: void
OnRtnOrder: void
OnRtnDeal: void
OnRtnHolding: void
OnRtnCash: void
OnRspError: void
interface
use
TDAPI
WinfairTdApi_CreateWinfairTdApi: IntPtr
WinfairTdApi_DeleteWinfairTdApi: void
WinfairTdApi_SetTdApiCallback: void
WinfairTdApi_Connect: bool
WinfairTdApi_Disconnect: bool
WinfairTdApi_IsConnected: bool
WinfairTdApi_ReqUserLogin: bool
WinfairTdApi_ReqUserLogout: bool
WinfairTdApi_ReqRegRspRtn: bool
WinfairTdApi_ReqQryTradingDay: IntPtr
WinfairTdApi_ReqQryInstrument: bool
WinfairTdApi_ReqOrderInsert: bool
WinfairTdApi_ReqOrderAction: bool
WinfairTdApi_ReqQryOrder: bool
WinfairTdApi_ReqQryHolding: bool
WinfairTdApi_ReqQryCash: bool
WinfairTdApi_ReqQryOrdersToday: bool
WinfairTdApi_ReqQryDealsToday: bool
Delegate
onconnected: delegate
ondisconnected: delegate
onheartbeatwarning: delegate
oninterrupted: delegate
onrspuserlogin: delegate
onrspuserlogout: delegate
onrspqryinstrument: delegate
onrspqryholding: delegate
onrtnholding: delegate
onrspqrycash: delegate
onrtncash: delegate
onrspqryorderstoday: delegate
onrspqrydealstoday: delegate
onrtnorder: delegate
onrtndeal: delegate
onrsperror: delegate
lpOnConnectedParam: void
lpOnDisconnectedParam: void
lpOnHeartBeatWarningParam: void
lpOnInterruptedParam: void
lpOnRspUserLoginParam: void
lpOnRspUserLogoutParam: void
lpOnRspQryInstrumentParam: void
lpOnRspQryHoldingParam: void
lpOnRspQryCashParam: void
lpOnRspQryOrdersTodayParam: void
lpOnRspQryDealsTodayParam: void
lpOnRtnOrderParamv: void
lpOnRtnDealParam: void
lpOnRspErrorParam: void

The figure above shows the class diagram design for the entire system. The detailed description has been completed in the sub-function module introduction.

4. Interface description

WinfairTDAPI is an asynchronous transaction interface library implemented using the C export function. Through the interface can be related to the transaction function, including the declaration and withdrawal orders, order status inquiries, fund inquiries, positions inquiries, commission and the day of the transaction, etc .; private flow of the main push information include: commission status return, turnover returns, Supported in virtual account mode) and position status returns (supported only in virtual traders account mode).

WinfairTDAPI's trader account can bind physical accounts. In the binding mode, the query funds and positions are the physical funds and positions of the physical account, the funds and positions are private mode, even if the same physical account binding, each transaction is independent of each other, there will not be entrusted and transactions Mixed question.

4.1 Interface development

Funtion CreateWinfairTdApi

Create a transaction interface handle, after the creation is complete, the client enters the function call phase.

Prototype:

HANDLE CreateWinfairTdApi (const char * pszClientTag);

parameter:

pszClientTag: API instance ID (different instances are different)

return value:

HANDLE: instance handle

Note:

A dll supports multiple client clients, using different handles to distinguish, but different instances must be different.

Function SetTdApiCallback

Set all kinds of callback function, to achieve the main push private flow.

Prototype：

void SetTdApiCallback(HANDLEhClient, void * pContex, lpOnConnectedParam pfnConnected, lpOnDisconnectedParam pfnDisconnected, lpOnHeartBeatWarningParam pfnHeartBeatWarning, lpOnInterruptedParam pfnInterrupted, lpOnRspUserLoginParam pfnRspUserLogin, lpOnRspUserLogoutParam pfnRspUserLogout, lpOnRspQryInstrumentParam pfnRspQryInstrument, lpOnRspQryHoldingParam pfnRspQryHolding, lpOnRspQryCashParam pfnRspQryCash, lpOnRspQryOrdersTodayParam pfnRspQryOrdersToday, lpOnRspQryDealsTodayParam pfnRspQryDealsToday, lpOnRtnOrderParam pfnRtnOrder, lpOnRtnDealParam pfnRtnDeal, lpOnRspQryHoldingParam pfnRtnHolding, lpOnRspQryCashParam pfnRtnCash, lpOnRspErrorParam pfnRspError);

Parameter：

hClient: handle

pContex: context

pfnConnected: OnConnect function pointer

pfnDisconnected: OnDisconnected function pointer

pfnHeartBeatWarning: OnHeartBeatWarning function pointer

pfnInterrupted: OnInterrupted function pointer

pfnRspUserLogin: OnRspUserLogin function pointer

pfnRspUserLogout: OnRspUserLogout function pointer

pfnRspQryInstrument: OnRspQryInstrument function pointer

pfnRspQryHolding: OnRspQryHolding function pointer

pfnRspQryCash: OnRspQryCash function pointer

pfnRspQryOrdersToday: OnRspQryOrdersToday function pointer

pfnRspQryDealsToday: OnRspQryDealsToday function pointer

pfnRtnOrder: OnRtnOrder function pointer

pfnRtnDeal: OnRtnDeal function pointer

pfnRtnHolding: OnRspQryHolding function pointer

pfnRtnCash: OnRspQryCash function pointer

pfnRspError: OnRspError function pointer

Function Connect

This method is invoked when the client establishes a communication connection with the trading system.

Prototype:

bool Connect (HANDLEhClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

connection succeeded

Function Disconnect

This method is invoked when the client disconnects from the trading system.

Prototype:

bool Disconnect (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

Disconnected successfully

Function IsConnected

Determine whether the client and the trading system are successfully connected.

Prototype:

bool IsConnected (HANDLE hClient);

parameter:

hClient: handle

return value:

Whether to connect

Function ReqUserLogin

The user issues a login request.

Prototype:

bool ReqUserLogin (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

User login is successful

Note:

In the connect will be automatically called, without manual call. After the connection is successful, call the ReqRegRspRtn method to subscribe to the private stream.

Function ReqUserLogout

The user issues a logout request.

Prototype:

bool ReqUserLogout (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

User logged out successful

Note:

In the disconnect will automatically call, without manual call.

Function ReqRegRspRtn

User subscribes to private streams. Private flows are information related to the transaction, including OnRtnOrder, OnRtnDeal, OnRtnCach, OnRtnHolding.

Prototype:

bool ReqRegRspRtn (HANDLE hClient, int nMode, char * pszErrorMsg);

parameter:

hClient: handle

nMode: 0 - from scratch, 1 - subscribed from last breakout

pszErrorMsg (return value): Error message that requires the caller to open up at least

256 bytes of memory space and pass in the pointer.

return value:

User subscription private flow is successful

Function ReqQryTradingDay

The user inquires about the trading day

Prototype:

char * ReqQryTradingDay (HANDLE hClient);

parameter:

hClient: handle

Return Type:

YYYYMMDD

Function ReqQryInstrumen

Asynchronous queryable tradable varieties

Prototype:

bool ReqQryInstrument (HANDLE hClient, char * pszErrorMsg, int nQueryType = 0,

char * pszExchangeInstrument = NULL);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least

256 bytes of memory space and pass in the pointer.

nQueryType: query type. 0 - the whole market, 1 by the exchange query, 2 by variety query (such as "SSE.600000, SZE.000001", separated by commas)

pszExchangeInstrument: the exchange code. "SSE" for the Shanghai Stock Exchange, "SZE" for the Shenzhen Stock Exchange, "CFFEX" in the gold, "SHFE" on the period, "DCE" big business, "CZCE" Zheng Shang.

return value:

Query the success of the transaction varieties.

Function ReqOrderInser

The client issues a declaration entry request.

Prototype:

charder, const char * pszFundid, const char * pszCombiID, char char *, ps, p, xp pszIdentity, char * pszLocalHandleID, char * pszErrorMsg);

parameter:

hClient: handle

pszExchangeID: exchange code

pszInstrumentID: transaction variety code

dOrderPrice: quote price

nOrderVolume: number of orders

eDirection: the direction of the report.

eOffset: bulletin open (in bound mode, the stock can be ignored)

pszAdapterTag: empty. Transaction interface label, physical account binding.

pszStockPoolTag: empty. Securities pool label.

pszFundID: empty. Fund code.

pszCombiID: empty. Combination code.

pszIdentity (return value): Trader code plus random code generated every time landing

pszLocalHandleID (return value): the current number of copies of the current record, each will be increased after the order

pszErrorMsg (return value): the error message, the caller needs to open up at least 2560 bytes of memory space (some physical account error message is longer, requires a larger space), and pass the pointer.

return value:

The report entry request was successful

Note:

The enumeration of the direction of the transaction:

{BID, ASK, MARGINBUY, SHORTSELL, COLLBUY, COLLSELL, UNKNOWNDIRECTION}

Delivery direction Chinese meaning:

{"Buy", "sell", "buy money", "sell sell", "guarantee buy", "sell", "unknown"

Announcement:

{OPEN, CLOSE, CLOSETODAY, CLOSEBYFORCE, UNKNOWNOFFSET}

Newspaper open the Chinese meaning:

{"Open", "open", "flat", "strong", "unknown"}

Function ReqOrderAction

The client issues a withdrawal request

Prototype:

bool ReqOrderAction (HANDLE hClient, DWORD dwUniqueID, const char * pszIdentity, const char * pszLocalHandleID, const char * pszExchangeID, const char * pszOrderSysID, char * pszErrorMsg);

parameter:

hClient: handle

dwUniqueID: order number unique code

pszIdentity: Connect random code, each generated random code

pszLocalHandleID: local order number, each time after the order will increase

pszExchangeID: exchange code

pszOrderSysID: exchange order number

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

Cancel the request successfully

Remarks (order positioning method):

There are three types of order positioning methods, one is dwUniqueID (unique number), the second is pszIdentity (connection random code) + pszLocalHandleID (local order number), the third is pszExchangeID (exchange code) + pszOrderSysID (exchange orders number). It is recommended to use the second method.

Function ReqQryOrder

Inquire query request, synchronization initiative to query a single order.

Prototype:

bool ReqQryOrder (HANDLE hClient, STRUCT_ORDER_FIXED * pstOrder, char * pszErrorMsg);

parameter:

hClient: handle

pstOrder (return value): report the address of the query structure

STRUCT_ORDER_FIXED structure:

TypedefstructtagSTRUCT_ORDER_FIXED

{

ORDER_STATUS ordStatus; // order status

char szTradingAdapterTag [NAME_LENGTH]; // transaction interface label

char szStockPoolTag [NAME_LENGTH]; // Securities pool label

char szFundID [NAME_LENGTH]; // fund code

char szCombiID [NAME_LENGTH]; // combination code

DWORD dwUniqueID; // unique code

char szSessionID [NAME_LENGTH]; // session ID

```
char szFrontID [NAME_LENGTH]; // front machine ID

char szTradingDay [NAME_LENGTH]; // trading day

char szOrderRef [NAME_LENGTH]; // quote reference

char szExchangeID [NAME_LENGTH]; // exchange code

char szOrderSysID [NAME_LENGTH]; // exchange

char szOrderLocalID [NAME_LENGTH]; // pre-machine local code

char szIdentity [NAME_LENGTH]; // connect random code

char szLocalHandleID [NAME_LENGTH]; // local order number

char szInstrumentID [NAME_LENGTH]; // stock code

ORDER_OFFSET ordOffset; // open a flat

ORDER_DIRECTION ordDirection; // trading direction

char cPriceType; / / report type: 0 - limit orders

UINT nOrderVolume; // report volume

double dOrderPrice; // report price

UINT nDealVolume; // volume

double dTotalAmount; // turnover

double dAveragePrice; // average transaction price

UINT nCanceledVolume; // withdrawal quantity

UINT nOrderDate; // order date

UINT nOrderTime; // order time

UINT nUpdateDate; // Order Status Update Date

UINT nUpdateTime; // Order Status Update Time
```

STRUCT_INSTRUMENT stInstrument; // Securities information

char szErrorMsg [NAME_LENGTH * 2]; // delegate error message

In the case of

}

}

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

The request for the inquiry request was successful

Note:

Order status enumeration:

{UNSENT, SENDED, QUEUEING, PARTTRADED, CANCELING, PARTTRADED_CANCELING, CANCELED, PARTTRADED_CANCELED, ALLTRADED, FAILED, UNKNOWNSTATUS}

Order status of the Chinese meaning:

"" Has not reported "," reported "," queue "," into the "," withdrawal of a single "," into a single withdrawal "," has been withdrawn "," ministry of withdrawal "," has become "," Failed "," unknown "}

Function ReqQryHolding

Asynchronous request query positions. In the binding mode will call the physical account query function, will be affected by the physical account refresh restrictions, it should try to reduce such active inquiries, should be based on commission and transaction returns based on local computing positions.

Prototype:

bool ReqQryHolding (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

The warehouse query request was successful.

Function ReqQryCash

Asynchronous request to inquire about funds. In the binding mode will call the physical account query function, will be affected by the physical account refresh restrictions, it should try to reduce such active inquiries, should be based on commission and transaction returns based on local computing funds.

Prototype:

bool ReqQryCash (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least

256 bytes of memory space and pass in the pointer.

return value:

Funding request is successful.

Function ReqQryOrdersToday

Asynchronous request query the day commission.

Prototype:

bool ReqQryOrdersToday (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least

256 bytes of memory space and pass in the pointer.

return value:

The date of the query request was successful.

Function ReqQryDealsToday

Asynchronous request query the day of the transaction.

Prototype:

bool ReqQryDealsToday (HANDLE hClient, char * pszErrorMsg);

parameter:

hClient: handle

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

return value:

The date of the transaction request was successful.

4.2 Callback Functions

OnConnected

This method is invoked when the client establishes a connection with the trading system.

Prototype:

void OnConnected (void * pContex, bool bSuccess, const char * pszUsername, const char * pszIdentity, const char * pszErrorMsg);

parameter:

pContex: User incoming context

bSuccess: Whether the connection was successful

pszUsername: user name

pszIdentity: connect random code

pszErrorMsg (return value): error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

OnDisconnected

This method is invoked when the client is disconnected from the trading system.

Prototype:

void OnDisconnected (void * pContex, const char * pszUsername, const char * pszIdentity, const char * pszErrorMsg);

parameter:

pContex: User incoming context

bSuccess: Whether the connection was successful

pszUsername: user name

pszIdentity: connect random code

pszErrorMsg (return value): Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

OnHeartBeatWarning

Heartbeat timeout warning. The method is called when a message is not received for a long time. The heartbeat timeout system is automatically connected without manual reconnection.

Prototype:

void OnHeartBeatWarning (void * pContex, const char * pszCmdAddress, const

char * pszBrdAddress, const char * pszIdentity);

parameter:

pContex: User incoming context

pszCmdAddress: server command port address

pszBrdAddress: Server broadcast port address

pszIdentity: connect random code

OnInterrupted

Connection interrupt warning. The connection interrupt system is automatically connected without manual reconnection.

Prototype:

void OnInterrupted (void * pContex, const char * pszUsername, const char * pszIdentity);

parameter:

pContex: User incoming context

pszUsername: user name

pszIdentity: connect random code

OnRspUserLogin

When the client sends a login request, the transaction system returns a response, the method will be called.

Prototype:

void OnRspUserLogin (void * pContex, boolbSuccess, const char * pszTraingDay, const char * pszUsername, const char * pszIdentity, const char * pszErrorMsg);

parameter:

pContex: User incoming context

bSuccess: whether successful

pszTraingDay: trading day

pszUsername: user name

pszIdentity: connect random code

pszErrorMsg: error message

OnRspUserLogout

When the client sends an exit request, the transaction system returns a response when the method is called.

Prototype:

void OnRspUserLogout (void * pContex, const char * pszUsername, const char * pszIdentity, const char * pszErrorMsg);

parameter:

pContex: User incoming context

pszUsername: user name

pszIdentity: connect random code

pszErrorMsg: error message

OnRspQryInstrument

Contract query response. When the client sends a contract query instruction, the transaction system returns a response, the method will be called.

Prototype:

void OnRspQryInstrument (void * pContex, bool bSuccess, STRUCT_INSTRUMENT * pInstruments, int nCount, const char * pszErrorMsg);

parameter:

pContex: User incoming context

bSuccess: whether successful

pInstruments: Trading Variant STRUCT_INSTRUMENT Structure Array Pointer, you can use pInstruments [n] to access the n + 1 record.

STRUCT_INSTRUMENT structure

{

/// contract code

charInstrumentID [31];

```
/// exchange code

charExchangeID [9];

/// contract name

charInstrumentName [21];

/// contract in the exchange code

charExchangeInstID [31];

///Product Code

charProductID [31];

///product type

charProductClass;

/// delivery year

int deliveredYear

/// delivery month

int DeliveryMonth;

/ / Market price of the largest single orders

int MaxMarketOrderVolume;

/ / Market price of the smallest single orders

int MinMarketOrderVolume;

/// limit the largest single orders

int MaxLimitOrderVolume;

/// limit order minimum order quantity

int MinLimitOrderVolume;
```

```
/// contract quantity multiplier

int VolumeMultiple;

/// minimum change price

double price

/// Create day

char CreateDate [9];

/// Listing date

char OpenDate [9];

///expiry date

char ExpireDate [9];

/// start the day of delivery

char StartDelivDate [9];

/// end the day of delivery

char EndDelivDate [9];

/// contract lifecycle status

char InstLifePhase;

/// whether it is currently trading

int IsTrading

/// Position type

char PositionType;

/ / Can the bill can be withdrawn

int OrderCanBeWithdraw;
```

```
/// Min buy a single unit

int MinBuyVolume;

/// sell the lowest single unit

int MinSellVolume;

/// stock privilege template code

char RightModelID [31];

/ / Position transaction type

char PosTradeType;

/// market code

char MarketID [31];

/// option execution price

double ExecPrice;

/// option one - hand margin

double unitMargin

/// contract type

char InstrumentType;

/// Position date type

char PositionDateType;

/ / Long margin rate

double LongMarginRatio;

/ / Short margin rate

double shortMarginRatio;
```

}

nCount: number of transactions trades

pszErrorMsg: error message

OnRspQryHolding

Investor Position Query Response. The method is invoked when the client sends an investor's position inquiry instruction and the trading system returns a response.

Prototype:

void OnRspQryHolding (void * pContex, bool bSuccess, STRUCT_HOLDING_FIXED * pHoldings, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pHoldings: Investor Position Structure STRUCT_HOLDING_FIXED Array pointer that can be used to access the n + 1 record at the same time using pHoldings [n].

STRUCT_HOLDING_FIXED structure:

{

char szExchangeID [NAME_LENGTH]; // exchange

char szCode [NAME_LENGTH]; // stock code

char szName [NAME_LENGTH]; // securities name

char szTradingAdapterTag [NAME_LENGTH];

char szStockPoolTag [NAME_LENGTH];

char szCombiID [NAME_LENGTH]; // combination ID (PB account support)

char szFundID [NAME_LENGTH]; // Fund Product ID (PB Account Support)

int nSecuType; // securities type

ORDER_DIRECTION ordDirection; // direction

int nTotalVolume; // total

int nAvailableVolume; // available

int nBuyFrozenVol; / / buy the amount of frozen (binding mode only part of the broker support)

int nSellFrozenVol; / / sell the frozen amount (binding mode only part of the broker support)

double dMarketValue; // market value

double dCurrentPrice; // current price

double dCostPrice; / / average cost price (binding mode only part of the broker support)

double dFloatingPL; / / floating profit and loss (binding mode only part of the broker support)

double dFloatingReturn; // floating rate of return (only part of the brokerage support mode)

double dReturn; / / yield (binding mode only part of the broker support)

double dClosePrice; // average open price (virtual mode only)

double dMargin; // occupancy margin (virtual mode only) double dClosedPL; / / open profit (binding mode only part of the broker support)

int nBuyVolToday; / / this purchase (binding mode only part of the broker support)

int nSellVolToday; // this amount of sales (binding mode only part of the broker support)

int nOpenedVolume; // total open amount (virtual mode only)

double dAverageOpenPrice; // average opening price (virtual mode support only)

int nClosedVolume; // total open positions (virtual mode only)

double dAverageClosePrice; // average rollover price (virtual mode support only)

}

nCount: total number of positions

nOperationSeq: operation serial number. Indicating the return of the new information on the table, the greater the number of operations, the new table.

pszErrorMsg: error message

OnRspQryCash

Request to query the account financial response. When the client sends a request to query the capital account instruction, the transaction system returns the response, the method will be called.

Prototype:

void OnRspQryCash (void * pContex, bool bSuccess, STRUCT_CASH_FIXED * pCashes, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pCashes: Account cash structure STRUCT_CASH_FIXED array index, use pCashes [n] to access the n + 1 record.

STRUCT_CASH_FIXED structure:

{

int nCurrencyType; // currency: 0 - RMB

double dTotalAmount; // total

double dAvaliableAmount; // available quota

double dFrozenAmount; / / freeze the amount of (binding mode only part of the broker support)

double dWithdrawableAmount; / / desirable amount (binding mode only part of the broker support)

double dMargin; // Margin (Batch mode only part of the broker support)

double dFuturesFloatingPL; // Futures Floating (only Virtual Mode Support)

double dCommission; // commission (virtual mode only)

double dTax; // stamp duty (virtual mode only)

double dFee; // exchange fee (virtual mode only)

char szTradingAdapterTag [NAME_LENGTH];

char szStockPoolTag [NAME_LENGTH];

char szFundID [NAME_LENGTH]; // Fund Product ID (PB Account Support)

char szCombiID [NAME_LENGTH]; // combination ID (PB account support)

}

nCount: Total number of records

nOperationSeq: operation serial number. Indicating the return of the new information on the table, the greater the number of operations, the new table.

pszErrorMsg: error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer

OnRspQryOrdersToday

Inquiry request for the current day. This method is invoked when the transaction system returns a response after the client issues a daily report request.

Prototype:

void OnRspQryOrdersToday (void * pContex, bool bSuccess, STRUCT_ORDER_FIXED * pOrders, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pOrders: Bulletin information structure array pointer, STRUCT_ORDER_FIXED structure See the description in the ReqQryOrder method. You can use pOrders [n] to access the n + 1 delegate record.

nCount: the number of day order records.

nOperationSeq: operation serial number. Indicating that the return of the form of information on the degree of old and new, the greater the operation sequence, the new table.

pszErrorMsg: Error message that requires the caller to open up at least 256 bytes of memory space and pass in the pointer.

OnRspQryDealsToday

The day of the transaction inquiry response. The method is invoked when the transaction system returns a response after the client issues the day's transaction inquiry instruction.

Prototype:

void OnRspQryDealsToday (void * pContex, bool bSuccess, STRUCT_DEAL_FIXED * pDeals, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pDeals: deal

pDeals structure:

{

char szTradeID [NAME_LENGTH]; // transaction number

char szTradingAdapterTag [NAME_LENGTH]; // transaction interface label

char szStockPoolTag [NAME_LENGTH]; // Securities pool label

char szFundID [NAME_LENGTH]; // fund code

char szCombiID [NAME_LENGTH]; // combination code

UINT nTradeDate; // Date of transaction

UINT nTradeTime; // transaction time

DWORD dwUniqueID; // unique code

char szTradingDay [NAME_LENGTH]; // trading day

char szExchangeID [NAME_LENGTH]; // exchange code

char szOrderSysID [NAME_LENGTH]; // The return code returned by the transaction

char szOrderLocalID [NAME_LENGTH]; // front machine under the order number

char szOrderRef [NAME_LENGTH]; // quote reference

char szIdentity [NAME_LENGTH]; // Dealer code plus random code generated for each login

char szLocalHandleID [NAME_LENGTH]; // local order number

char szInstrumentID [NAME_LENGTH]; // transaction variety code

ORDER_OFFSET ordOffset; / / report open open

ORDER_DIRECTION ordDirection; / / report the direction of the transaction

UINT nDealVolume; // volume

double dTotalAmount; // turnover

double dAveragePrice; // average transaction price

}

nCount: number of transactions

pDeals: Array pointer to the transaction information structure STRUCT_DEAL_FIXED. You can use pDeals [n] to access the n + 1 delegate record.

nOperationSeq: operation serial number. Indicating the return of the new information on the table, the greater the number of operations, the new table.

pszErrorMsg: error message

OnRtnOrder

Report return. When the client makes a report entry, report operation and other reasons (such as partial transactions) lead to changes in the status of the bill, the trading system will automatically notify the client, the method will be called.

Prototype:

void OnRtnOrder (void * pContex, bool bSuccess, STRUCT_ORDER_FIXED * pOrder, bool bCurrentConnection, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pOrder: bill of lading

STRUCT_ORDER_FIXED structure: See the instructions in ReqQryOrder

bCurrentConnection: whether the current connection (based on identityID)

pszErrorMsg: error message

OnRtnDeal

Turnover return. The transaction system will notify the client when the transaction occurs, and the method is called.

Prototype:

void OnRtnDeal (void * pContex, bool bSuccess, STRUCT_DEAL_FIXED * pDeal, bool bCurrentConnection, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pDeal: Description of the transaction record pointer, STRUCT_DEAL_FIXED See the description in the OnRspQryDealsToday method

bCurrentConnection: whether the current connection

pszErrorMsg: error message.

OnRtnHolding

Position returns. This method is invoked when the user's position changes (only supported in the virtual traders account mode).

Prototype:

void OnRtnHolding (void * pContex, boolbSuccess, STRUCT_HOLDING_FIXED * pHoldings, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pHoldings: Investor Position Structure STRUCT_HOLDING_FIXED Array pointer that can be used to access the n + 1 record at the same time using pHoldings [n].

STRUCT_HOLDING_FIXED structure See the description in the OnRspQryHoling method

nCount: number of positions recorded

nOperationSeq :: operation serial number. Indicating that the return of the form of information on the degree of old and new, the greater the operation sequence, the new table.

pszErrorMsg: error message

OnRtnCash

Cash return. This method is invoked when the user's cash changes (only supported in the virtual traders account mode).

Prototype:

void OnRtnCash (void * pContex, bool bSuccess, STRUCT_CASH_FIXED * pCashes, int nCount, int nOperationSeq, const char * pszErrorMsg);

parameter:

pContex: user-defined context

bSuccess: whether successful

pCashes: cash record pointer, STRUCT_CASH_FIXED structure See the description in the OnRspQryCash method

nCount: number of cash records

nOperationSeq :: operation serial number. Indicating that the return of the form of information on the degree of old and new, the greater the operation sequence, the new table.

pszErrorMsg: error message

OnRspError

Wrong return. When the system has an error, the method is called.

Prototype:

void OnRspError (void * pContex, DWORD dwUniqueID, UINT nErrorClass, UINT nErrorID, const char * pszErrorInfo, bool bCurrentConnection);

parameter:

pContex: user-defined context

dwUniqueID: unique code

nErrorClass: Error type

nErrorID: Error ID

pszErrorInfo (return value): error message

bCurrentConnection: whether the current connection

Strategy Document

Strategy 1

1. Strategy description

The strategy uses the algorithm of " Linear Programming models based on Omega Ratio for the Enhanced Index Tracking Problem" (G.Guastaroba, R.Mansini, W.Ogryczak, 2016), which is published in the European Journal of Operational Research.

Under the basic framework of the index enhancement strategy, it defines the ratio of the excess rate of return to the downside. Based on the historical data, the optimal combination is predicted by maximizing the Omega Ratio, and the optimal combination is considered in the next The period has a high rate of return at the same time only bear the limited risk.

The main contribution of this paper is to construct the two linear programming models (OR model and EOR model). Then, two mixed integer programming models (OR-RF model) are constructed according to the real factors in the stock market. And EOR-RF models), and linear programming problems and mixed integer programming problems exist for the stable use of commercial solvers (such as CPLEX, GUROBI, etc.).

The paper validates the validity of the four models, and uses the historical data of the FTSE 100 index, the Hong Kong Hang Seng Index and the German DAX Index. Since the basic idea of the paper is to forecast the future with historical data, the paper is based on the stock The data can be divided into different data sets, which verifies the investment rate of the model in different market environment and different stock movements. It can prove the generality of the four strategies.

On the basis of the realization of the original paper, the participating groups constructed three different types to derive a number of variants of the original strategy and verify their effectiveness. In addition, in order to test the effectiveness of the strategy in the mainland market, we selected the HS300 Index, the SZ100 Index and other historical data to do a back test, proved that the original strategy and derivative strategies have a good effect.

2. Model description

Omega Ratio：

$$\Omega_{\mu^{\alpha}}(R_x) = \frac{E\{(R_x-\mu^{\alpha})_+\}}{E\{(R_x-\mu^{\alpha})_-\}} = \frac{\sum_{t=1}^{T} max\{y_t-\mu^{\alpha},0\}p_t}{\sum_{t=1}^{T} max\{\mu^{\alpha}-y_t,0\}p_t}$$

Where $\mu^{\alpha} = \mu^{I} + \alpha$, $\mu^{I}$is the mean of the yield in the period of [0, T], the weight is $\frac{1}{T'}$; α is a freely adjustable parameter, Rate higher than the target value of the broader market. The $y_t = \sum_{j=1}^{n} r_{jt}x_j$represents the yield of the selected portfolio at time t, where r_{jt} represents the yield of the jth stock at time t; $x_j$represents the yield of the jth The stock should hold the number for the decision variable.

The numerator represents the mean of the excess return in the [0, T] period, and the denominator indicates the mean value of the downside risk in the [0, T] period. We believe that by maximizing the ratio of the two, the portfolio has a higher yield While at the same time suffers from limited risk. Based on this idea, the following planning model is obtained:

Min $\frac{z-\mu^{\alpha}}{z_1}$

s.t. $\sum_{j=1}^{n} x_j = 1$

$\sum_{j=1}^{n} \mu_j x_j = z$
$\sum_{j=1}^{n} r_{jt} x_j = y_t$ for $t = 1, \dots, T$
$\sum_{t=1}^{T} d_t p_t = z_1$

$d_t \geq \mu^{\alpha} - y_t$ for t=1,…,T

$d_t \geq 0$ for t=1,…,T

After the skills in operational research, it is transformed into the following linear programming model:

1) OR model

Min $v - \mu^{\alpha} v_0$

s.j. $\sum_{j=1}^{n} \tilde{x}_j = v_0$

$v_0 \leq M$
$\tilde{x}_j \geq 0$ for $j = 1, \dots, n$
$\sum_{j=1}^{n} \mu_j \tilde{x}_j = v$

$\sum_{j=1}^{n} r_{jt} \tilde{x}_j = \tilde{y}_t$ for t=1,…,T

$\sum_{t=1}^{T} \tilde{d}_t p_t = 1$
$\tilde{d}_t \geq \mu^{\alpha} v_0 - \tilde{y}_t$
$\tilde{d}_t \geq 0$ for $t = 1, \dots, T$

2). EOR model

After changing the parameter μ^{α} in the constraint to r_t^{α} .the effect of the model will be improved remarkably. The significance of this substitution is that the model takes into account the influencing factors of the stock trajectory, rather than simply considering the average of time. After the change model is as follows:

Min $v - \mu^{\alpha} v_0$

s.t. $\sum_{j=1}^{n} \tilde{x}_j = v_0, v_0 \leq M, \tilde{x}_j \geq 0$ for $j = 1, \dots, n$

$$\sum_{j=1}^{n} \mu_j \tilde{x}_j = v$$
$$\sum_{j=1}^{n} r_{jt} \tilde{x}_j = \tilde{y}_t \quad \text{for } t = 1, \dots, T$$
$$\sum_{t=1}^{T} p_t \tilde{d}_t = 1$$
$$\tilde{d}_t \geq r_t^{\alpha} v_0 - \tilde{y}_t, \tilde{d}_t \geq 0 \ \text{ for } t = 1, \dots, T$$

3). OR-RF model and EOR-RF model

This part of the OR model and EOR model in the stock market to add two real constraints, first, out of the control fee considerations, the selected portfolio of the number of constituent stocks can not be too much; secondly,

The share of the selected stock can not be too low or too high.

$\tilde{x}_j \leq M z_j$ for j=1,…,n

$\sum_{j=1}^{n} z_j \leq K$

$z_j \in \{0,1\}$ for j=1,…,n

$\tilde{x}_j \geq \varepsilon_j v_{0j}$ for j=1,…,n

$0 \leq v_{0j} \leq M z_j$ for j=1,…,n

$v_{0j} \leq v_0$ for j=1,…,n

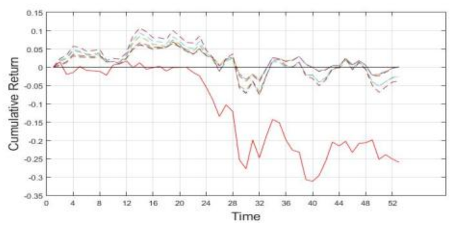
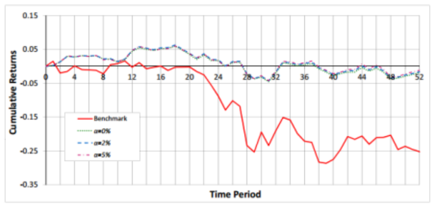
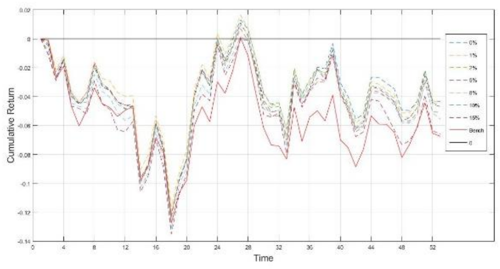
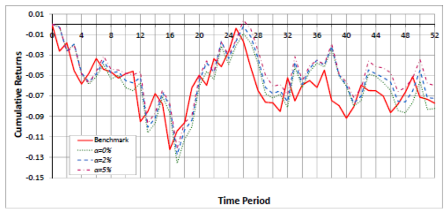
$v_0 - v_{0j} + M z_j \leq M$ for j=1,…,n

3. Algorithm description

Gurobi is a large-scale mathematical programming solver developed by the United States Gurobi company, especially in the mixed integer programming problem on the outstanding performance, fast and high accuracy. This group uses Matlab to call the Gurobi solver to solve the model.

4. Experimental data

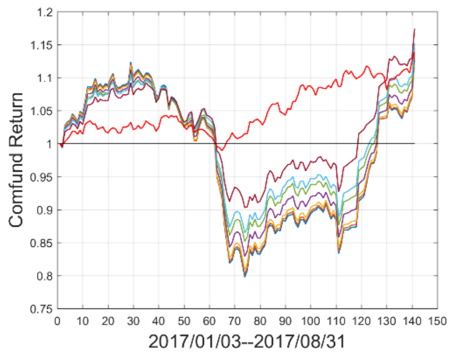
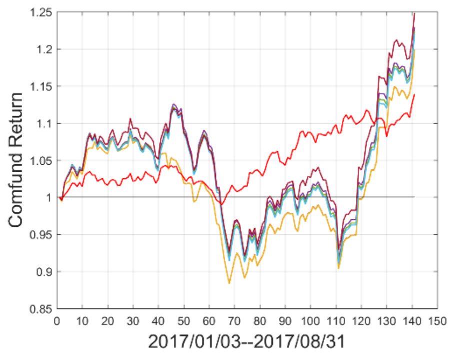
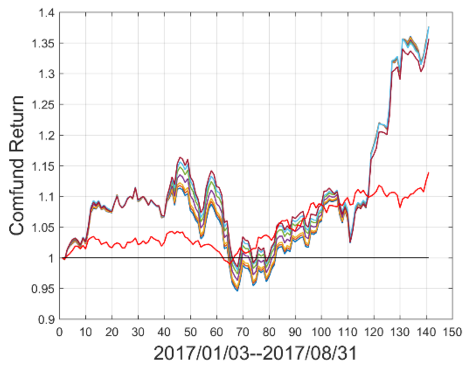
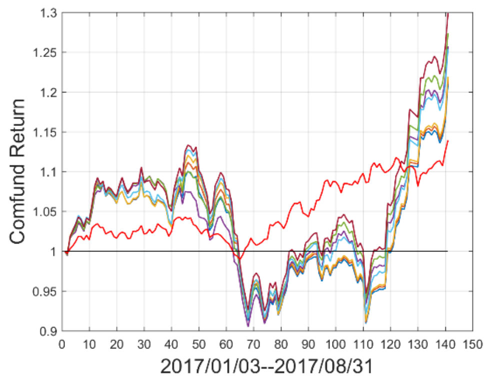
The results of the experiments with the same data set repeat the experimental results to prove the correctness of our algorithm:

Repeat result  The red curve is Benchmark; Others are experiment results.	Original Result  The red curve is Benchmark; Others are experiment results.
Repeat result  The red curve is Benchmark; Others are experiment results.	Original Result  Figure 35: Out-of-sample cumulative returns (instance ORL-IT3): A comparison among the optimal portfolio (EOR model) and the market index. The red curve is Benchmark; Others are experiment results.

The highly consistent results demonstrate the correctness of the algorithm when we reproduce. And these shows the good performance of these four models.

5. Chinese market HS 300 backtesting results

Based on the high degree of irrationality of the Chinese market, we selected a relatively short period as an experimental parameter for back experiment.

 The red curve is Benchmark; Others are experiment results.	 The red curve is Benchmark; Others are experiment results.
 The red curve is Benchmark; Others are experiment results.	 The red curve is Benchmark; Others are experiment results.

Different models have their own advantages and disadvantages, these four pictures shows recently the EOR model can get rid of the downside risk effectively. And as the benchmark increases, these four model performs quite well.

The Second Strategy

1. Strategy:

   The second strategy realize the method proposed in a paper, Enhanced Index Tracking with Multiple Time-scale Analysis, written by Qian Li and Liang Bao.

And the paper was published in the Economic Modelling.

The paper establishes a multi-objective programing model with 4 objectives. Based on the enhanced index method, the authors made their effort to capture the dynamic information in the fluctuation of stocks. The enhanced index method tries to track the index and beat the market at the same time. In order to capture the dynamic information, the authors calculate the difference between the two time-series of index prices and the prices of portfolio. In this way, we can get a negative return deviation time-series and a positive return deviation time-series. Then, the authors apply the technique in spectral analysis, EEMD (ensemble empirical mode decomposition), to the two time-series respectively. After that, we have a set of intrinsic mode functions (IMFs) representing the different frequencies. In the context of enhanced index method, we need to minimize the IMFs which are representing the low frequency obtained from negative return deviation time-series, meanwhile, maximize the IMFs which are representing the low frequency obtained from positive return deviation time-series.

In order to solve the multi-objective programing problem, the authors try to find the Pareto optimal solution set. Basically, a solution is Pareto optimal if we cannot enhance one of objective while maintaining the performance of other objectives. All the Pareto optimal solutions constitute the Pareto optimal set.

Pareto optimal solution means a trade-off between all the objectives. That is, we must sacrifice one or more objectives if we want to improve any of the objectives. After we find the Pareto optimal set, the authors select the solution with the largest value of excesses return over tracking error as the final portfolio.

The nonconcave and nonconvex programing problem cannot be solved by simplex method of interior point method. The authors decide on genetic algorithm and combine it with immune algorithm. The combined immune genetic algorithm can find the approximal Pareto optimal solutions in limited number of iterations.

Authors validated their model in the last part of the paper. Their data sets include Hang seng 33, Dax 200, FTSE 100, S&P 100 and Nikkei 225. And for each stock index and it component stocks, they collected the weekly data from March 1992 to September 1997. By doing this, the authors proved that after capturing the tracking dynamics, the performance of enhanced index method can be improved significantly. Our group applied this model to CSI 300 and proved that this model is suitable for mainland market.

2. Model

The four objectives are shown below:

(a) Maximize the excesses return;

(b) Minimize the tracking error between the portfolio and the index. We define the tracking error as negative return deviation since we prefer that the portfolio perform better than index;

(c) Minimize the low frequency factors from tracking error;

(d) Maximize the low frequency factors of excess return;

The constraints are:

(a) Short positions are forbidden;

(b) The proportion of each security in the portfolio must satisfy the proportion limits defined;

(c) No cash inflow and outflow in the investment horizon;

(d) The total transaction costs incurred cannot exceed a certain proportion of initial investment amount;

Model:

$$\min \ \frac{1}{T}\left[\sum_{t=1}^{T}\left(r_p^t - R^t\right)_-^2\right]^{1/2}$$

$$\max \frac{1}{T}\sum_{t=1}^{T}\left(r_p^t - R^t\right)$$

$$\min \sum_{t=1}^{T}\sum_{j=a+1}^{m+1}\left(sn_j^t\right)^2$$

$$\max \sum_{t=1}^{T}\sum_{j=a+1}^{m+1}\left(sp_j^t\right)^2$$

$$\text{s.j. } q_i \geq 0$$

$$\epsilon \leq \frac{p_i^T q_i}{\sum_{i=1}^{N} p_i^T q_i} \leq \delta$$

$$\sum_{i=1}^{N} p_i^T q_i = \sum_{i=1}^{N} p_i^T q_i^0$$

$$\eta \sum_{i=1}^{N} p_i^T \left|q_i - q_i^0\right| \leq \lambda C$$

where $r_p^t := ln\left[\frac{\sum_{i=1}^{N} p_i^t q_i}{\sum_{i=1}^{N} p_i^{t-1} q_i}\right]$, $R^t := ln\frac{I^t}{I^{t-1}}$.

The parameters are shown below:

r_p^t	The single period continuous time return of the tracking portfolio at time t.	R^t	The single period continuous time return of the market index at time t.
T	Decision time point. [0,T] is the in-sample time period to select new tracking portfolio.	I^t	Index value at time t (t=0,…,T).
m	The number of IMFs.	a	The number of low frequency IMFs.
sn_j^t	IMFs obtained from negative return deviation time-series.	sp_j^t	IMFs obtained from positive return deviation time-series.
q_i	The number of units of stock i that are selected	p_i^t	Price of stock i at time t.

	to hold in the tracking portfolio.		
ϵ	Minimum proportion of each security in the portfolio.	δ	Maximum proportion of each security in the portfolio.
λ	The maximum proportion of transaction costs incurred in total value.	C	The total value of the current tracking portfolio at time T.
η	The proportion rate of the transaction value consumed by transaction costs.		

3. Algorithm

Authors decide on the elite immune genetic algorithm, which is different from the general genetic algorithm in the definition of fitness. We have to rank the antibody vector twice in different ways in order to find out the Pareto optimal solutions. And the calculation of fitness is based on the result of two ranks.

(a) Dominance ranking:

The authors use dominance ranking method combined with crowding-distance measure to evaluate the performance of antibodies. The procedure of dominance ranking works as follows:

Step 1: Identify non-Pareto dominated antibodies from population (B) and separate them into two sub-set, non-dominated population (ND) and dominated population (D).

Step 2: Define the crowding distance as follows:

$$\lambda(b) := \sum_{i=1}^{k} \frac{\lambda_i(b)}{f_i^{max} - f_i^{min}}$$

where b represents the antibody, f_i^{max} and f_i^{min} represent the maximum and minimum value of the ith objective and

$$\lambda_i(b) := \begin{cases} \infty, if \ \ f_i(b) = min \ \ f_i(b') or f_i(b) = max \ \ f_i(b') \\ \min\{ \ f_i(b') - \ \ f_i(b'') | \ f_i(b'') < \ \ f_i(b) < \ \ f_i(b') \end{cases}$$

Calculate the crowding-distance values of all individuals and sort the non-dominated antibodies in descending order of crowding-distance. The highest ranking antibodies in ND get rank=1 and the second one get rank=2, and so forth. (Assume the last one equals to d)

Step 3: Identify the non-dominated antibodies from population D and set their dominance ranking= d+1. Exclude them from the population D and set d=d+1.

Step 4: Repeat Step3 until all the antibodies in population D are non-dominated.

(b) Feasibility ranking:

Define the constraint violation as follows:

$$\text{Constraint violation}(b_i) = \eta \sum_{i=1}^{N} p_i^T \left|q_i - q_i^0\right| - \lambda C$$

The first three constraints are guaranteed in the procedure of initialization. Any solution with the value of constraint violation less than or equal to zero will be feasible. Calculate all the feasibility ranking of population and sort them in ascending order and set their feasibility ranking=1,2,3,... successively. Count the number of feasible solution denoted by h_f.

After the calculation of dominance ranking and feasibility ranking, we define the fitness of antibodies as follows:

$$\text{Fitness}(b_i) = h - \left[\frac{h_f}{h} rank_1(b_i) + \frac{h - h_f}{h} rank_2(b_i)\right]$$

Where h represents the size of antibody population, $rank_1$ represents the dominance ranking and $rank_2$ represents the feasibility ranking. Note that the rankings of certain antibody are related to the population, so we have to recalculate the rankings if the population has some changes.

The procedure of the algorithm as follows:

1. $\mathrm{t}=0, \mathrm{E}(\mathrm{t})=\phi, \mathrm{A}(\mathrm{t})=\phi, \mathrm{C}(\mathrm{t})=\phi$. Generate the initial antibody population $\mathrm{B}(\mathrm{t})$ as follows, and denote the size of the population by h:

$$b_i=\left[\frac{\varepsilon \sum_{i=1}^{N} p_i^T q_i^0}{p_i^T}, \frac{\delta \sum_{i=1}^{N} p_i^T q_i^0}{p_i^T}\right] \quad \text{for } \mathrm{i}=1, \ldots, \mathrm{h}$$

2. Calculate the fitness of each antibody in $\mathrm{B}(\mathrm{t})$ and sort them in descending order. Select the first n_E to form the elite group $\mathrm{E}(\mathrm{t}+1)$.
3. If $\mathrm{t} \geq \mathrm{Gmax}$ (maximum number of generations) is satisfied, export $\mathrm{E}(\mathrm{t}+1)$ as the output of the algorithm, stop; Otherwise, t=t+1. Particularly, if the improvement of the 4 objectives is less than 10^{-6}, export $\mathrm{E}(\mathrm{t}+1)$ as the output of the algorithm, stop.
4. Compare the size of $\mathrm{E}(\mathrm{t})$ with n_A. Let $\mathrm{A}(\mathrm{t})=\mathrm{E}(\mathrm{t})$ if the size of $\mathrm{E}(\mathrm{t})$ is not greater than n_A. Otherwise, calculate the fitness value of all individuals in $\mathrm{E}(\mathrm{t})$, sort them in descending order of fitness value, and choose the first n_A individuals to form $\mathrm{A}(\mathrm{t})$.
5. Apply proportional cloning to $\mathrm{A}(\mathrm{t})$. That is, each antibody in $\mathrm{A}(\mathrm{t})$ is reproduced proportionally into a clone population with a certain size. The clone size depends on the value of fitness function of each antibody. The greater the fitness value of an individual, the more times the individual will be reproduced. Thus, the clone population $\mathrm{C}(\mathrm{t})$ is obtained.

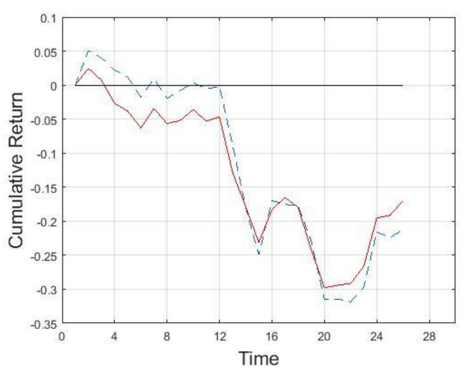
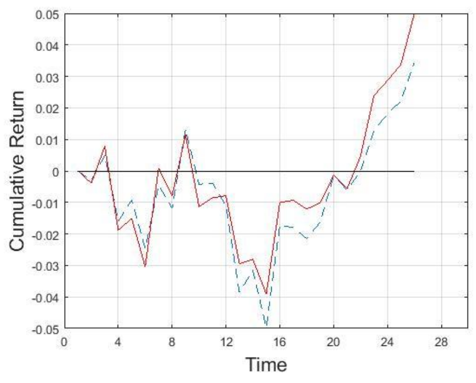
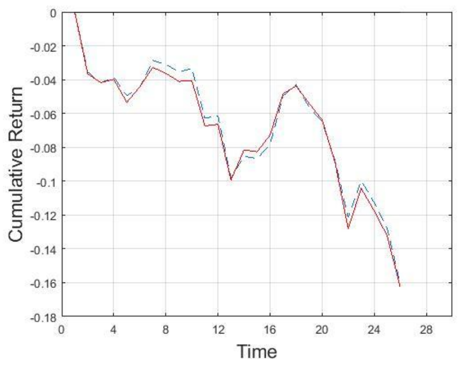
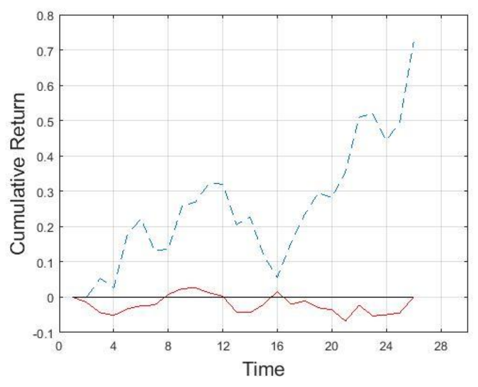
6. Perform recombination and hypermutation on $\mathrm{C(t)}$. The procedure of recombination selects one individual with equal probability from the two offspring s generated by a general cross-over operator on each cloned antibody with an antibody selected randomly from the original antibody population. After recombination, the clone population is recombined. Each element of the antibody vector in the clone population is changed by a general mutation operator with certain probability. Set the resulting population as $C'(\mathrm{t})$.
7. Get the antibody population $\mathrm{B(t)}$ by combining the $C'(\mathrm{t})$ with $\mathrm{E(t)}$; go to step 2.

4. Experiments

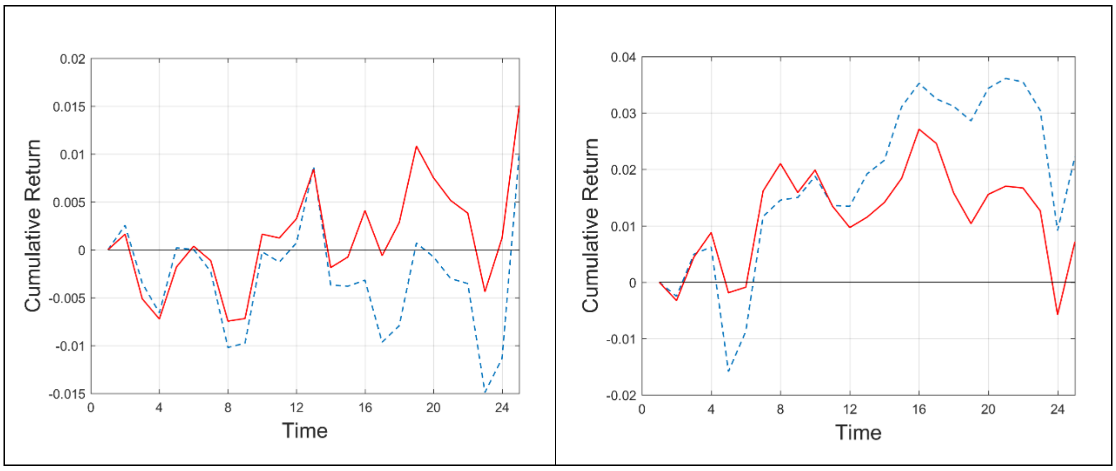
Number of ensemble members for EEMD	100	S.D. of white noise series	0.2
ε	0	δ	0.1
C	10^6	η	0.01
h	100	n_E	100
n_A	20	Crossover rate	1
Mutation probability	0.1	Gmax	500

n_C	100		

In different market trends, the performance of our portfolio is different.

The red curve represents the index while the other represent the portfolio we selected based on our model. When the market is fluctuating violently, our portfolio can significantly capture the trend, and in the continuous down to be able to accurately track the broader market. In the market stable fluctuations, our portfolio benefits of obvious advantages.



The Back testing results of HS300 stock market during recent period.

We use the data range from May to August in 2017.

The red curve is the performance of Benchmark, which is HS300 Index.

The blue curve is the performance of our portfolio.

Recently, the market trend is fluctuating, our portfolio can significantly follow and outperform the benchmark.