项目名称

UML model report

**Table of Content**

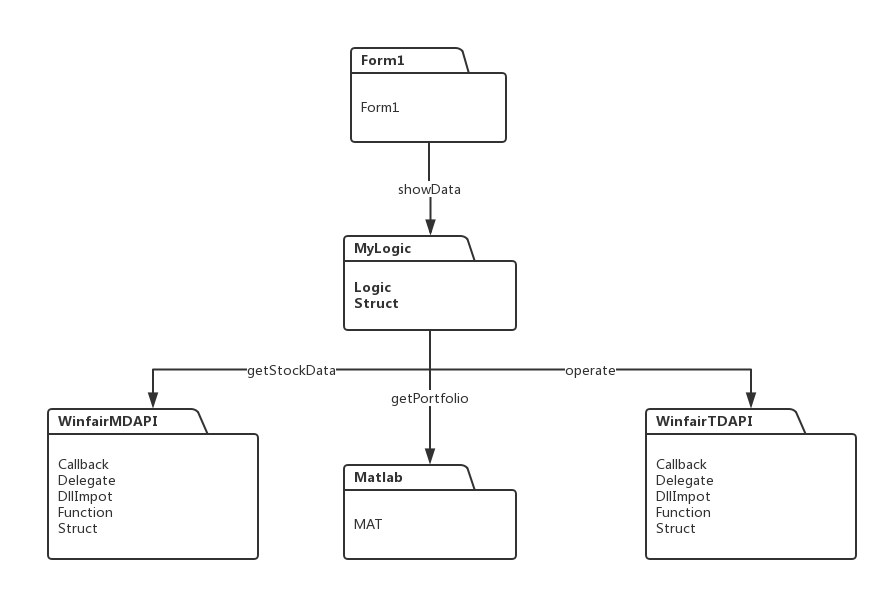
[1. The overall structure of the system 2](#_Toc471501064)

[2. Class diagram degisn of every function model 3](#_Toc471501068)

[3. System class diagram design 10](#_Toc471501068)

[4. Interface description 11](#_Toc471501068)

**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.

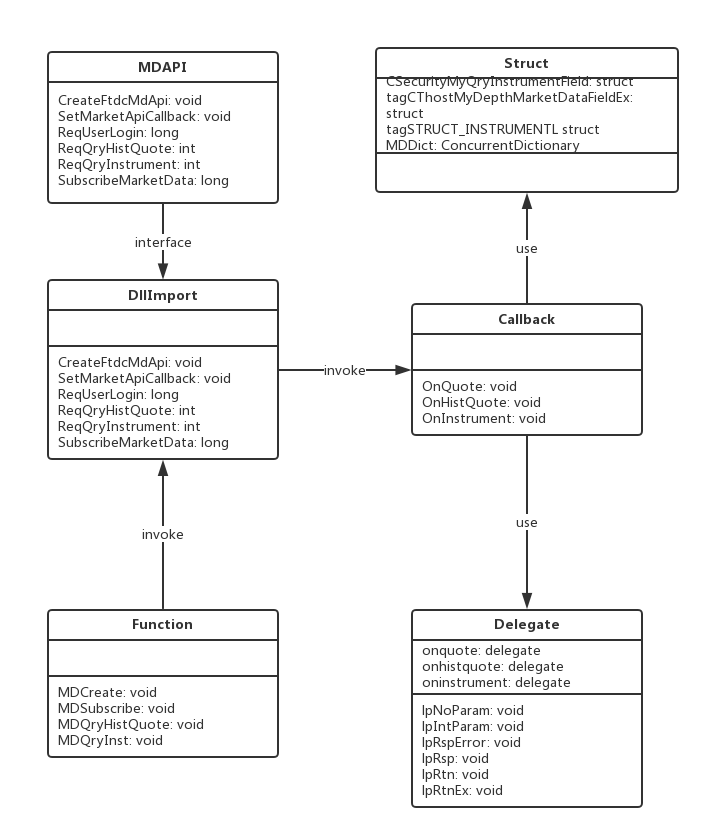
④ MyLogic: trading logic part is responsible for judging when to buy and sell, control the threshold of the sale. Get the portfolio from the Matlab section and perform specific transaction operations through the WinfairTDAPI section. At the same time, the part also needs to be provided by WinfairMDAPI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

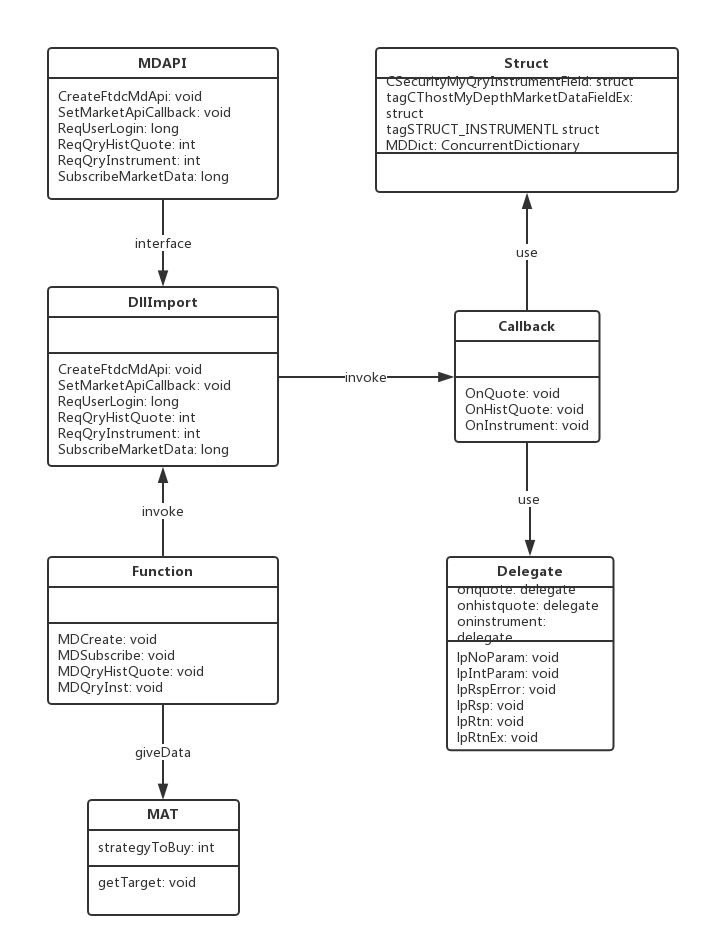
② DllImport: The main role of the class is to call the interface required Dll into the namespace and the required method to declare.

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.

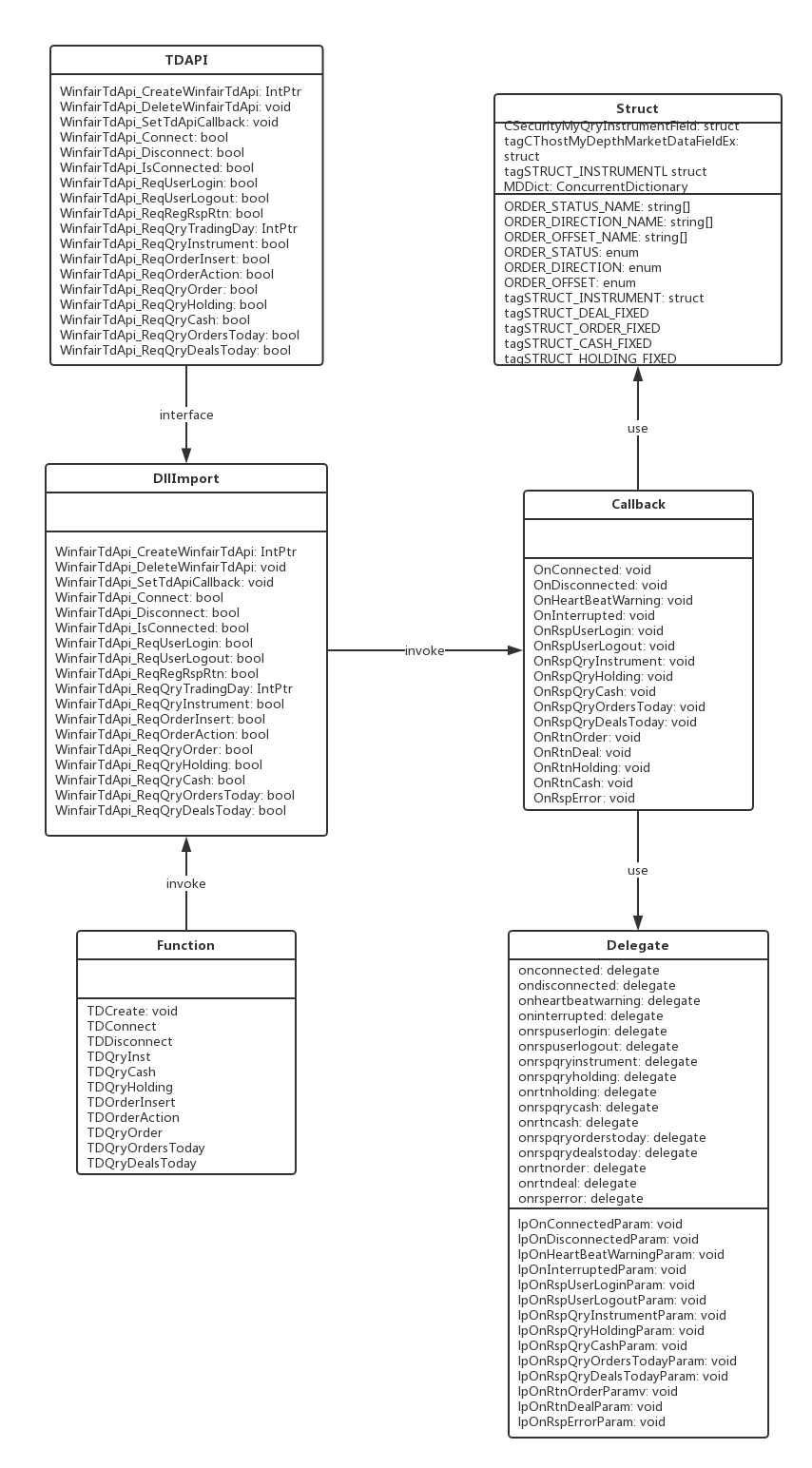
⑤ Struct: The main function of this class is the storage interface provides a variety of structural data.

## 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）



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.

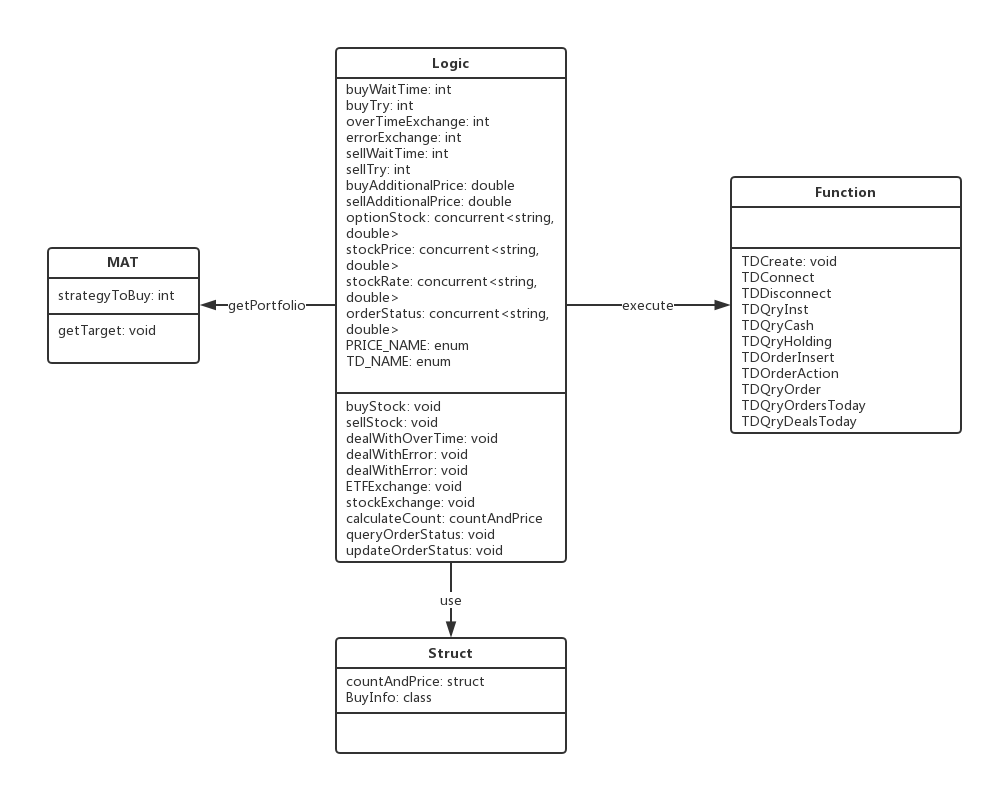
② DllImport: The main role of the class is to call the interface required Dll into the namespace and the required method to declare.

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.

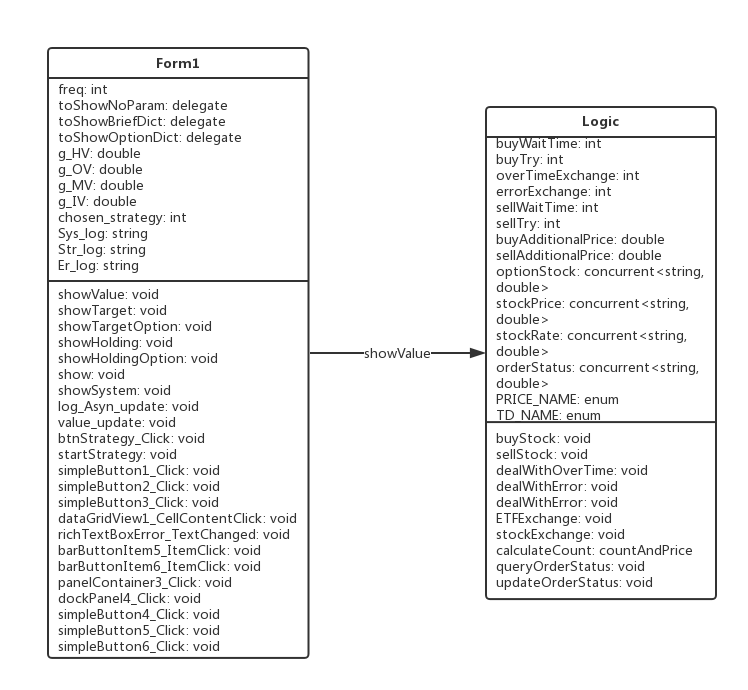
## ② through the relevant formula to calculate the combined market value and buy the market value after the sale to determine whether it exceeds the threshold of buying or selling, and thus decide whether to buy.

## 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.

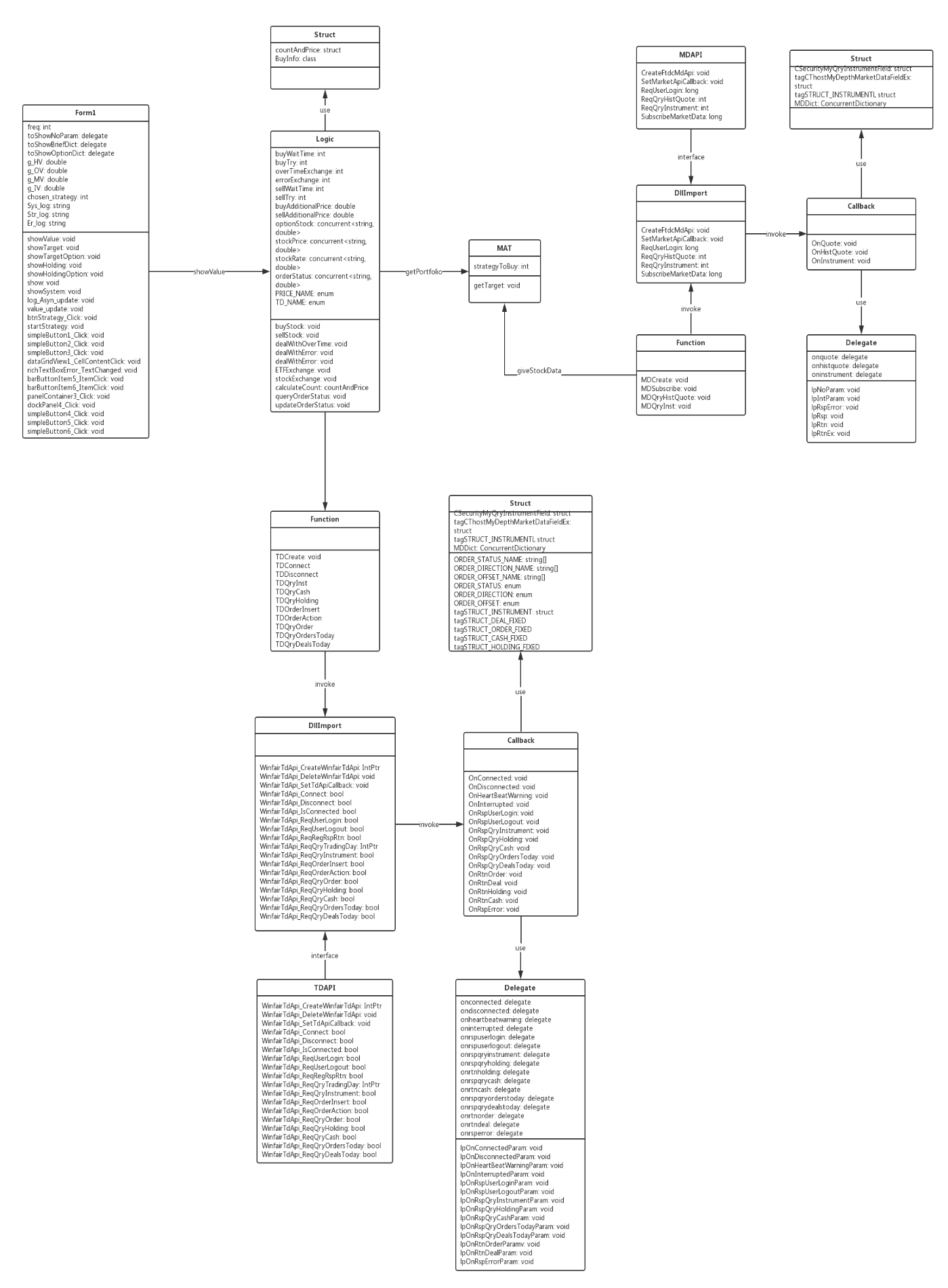
① visualization of various market values: by calling the Logic class in the market value of data, combined with a variety of display methods, the user needs a variety of market value data display,

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



# 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