

目 录

matlab策略SDK概述

概述

matlab策略 SDK的安装

matlab策略运行

注意事项

典型策略场景

定时事件驱动

订阅数据驱动

成交回报事件驱动

指定账户交易

获取资金持仓信息交易策略

设定策略运行参数

仅提取数据

典型应用场景

策略组成文件

策略编写m文件-main.m

策略运行文件—run.m

策略结构要素

全局变量

初始化事件

设置滑窗标的序列

定时运行函数和定时事件

订阅数据滑窗和数据事件

交易事件驱动

其他事件驱动

存储自定义全局变量

策略停止函数—stop_strategy

获取数据

bar 数据结构说明

tick 数据结构说明

财务数据及其他业务数据结构

subscribe—订阅方式获取决策数据

current - 查询当前行情快照

history - 查询固定时间历史行情

history_n - 查询固定长度历史行情

get_fundamentals - 查询固定时间基本面数据

get_fundamentals_n - 查询固定长度基本面数据
get_instruments - 查询最新交易标的最新基本信息
get_history_instruments - 查询交易标的历史基本信息
get_instrumentinfos - 查询交易标的基本信息
get_constituents - 查询指数成份股信息
get_industry - 查询行业板块成分股
get_trading_dates - 查询交易日历表
get_previous_trading_date - 查询上一个交易日
get_next_trading_date - 查询下一个交易日
get_dividend - 查询分红送配信息
get_continuous_contracts - 获取主力连续合约

策略交易

订单数据结构说明
回报数据结构说明
持仓数据结构说明
资金数据结构说明
order_volume - 按指定量委托
order_value - 按指定价值委托
order_percent - 按总资产指定比例委托
order_target_volume - 调仓到目标持仓量
order_target_value - 调仓到目标持仓额
order_target_percent - 调仓到目标持仓比例（总资产的比例）
order_batch - 批量委托
order_cancel - 撤销指定委托
order_cancel_all - 撤销所有委托
order_close_all - 平当前所有可平持仓
get_unfinished_orders - 查询日内全部未结委托
get_orders - 查询日内全部委托
get_execution_reports - 查询日内全部执行回报
get_position - 持仓查询
get_cash - 资金查询

枚举常量

OrderStatus - 委托状态
OrderSide - 委托方向
OrderType - 委托类型
OrderDuration - 委托时间属性
OrderQualifier - 委托成交属性
ExecType - 执行回报类型

PositionEffect - 开平仓类型

PositionSide - 持仓方向

OrderRejectReason - 订单拒绝原因

CancelOrderRejectReason - 取消订单拒绝原因

OrderStyle - 订单类型

CashPositionChangeReason - 仓位变更原因

SecType - 标的类别

AccountStatus - 交易账户状态

错误码

matlab策略SDK概述

- [概述](#)
- [matlab策略 SDK的安装](#)
- [matlab策略运行](#)
- [注意事项](#)

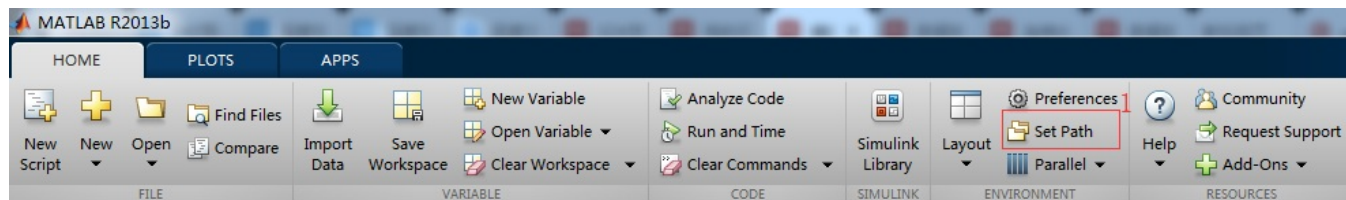
概述

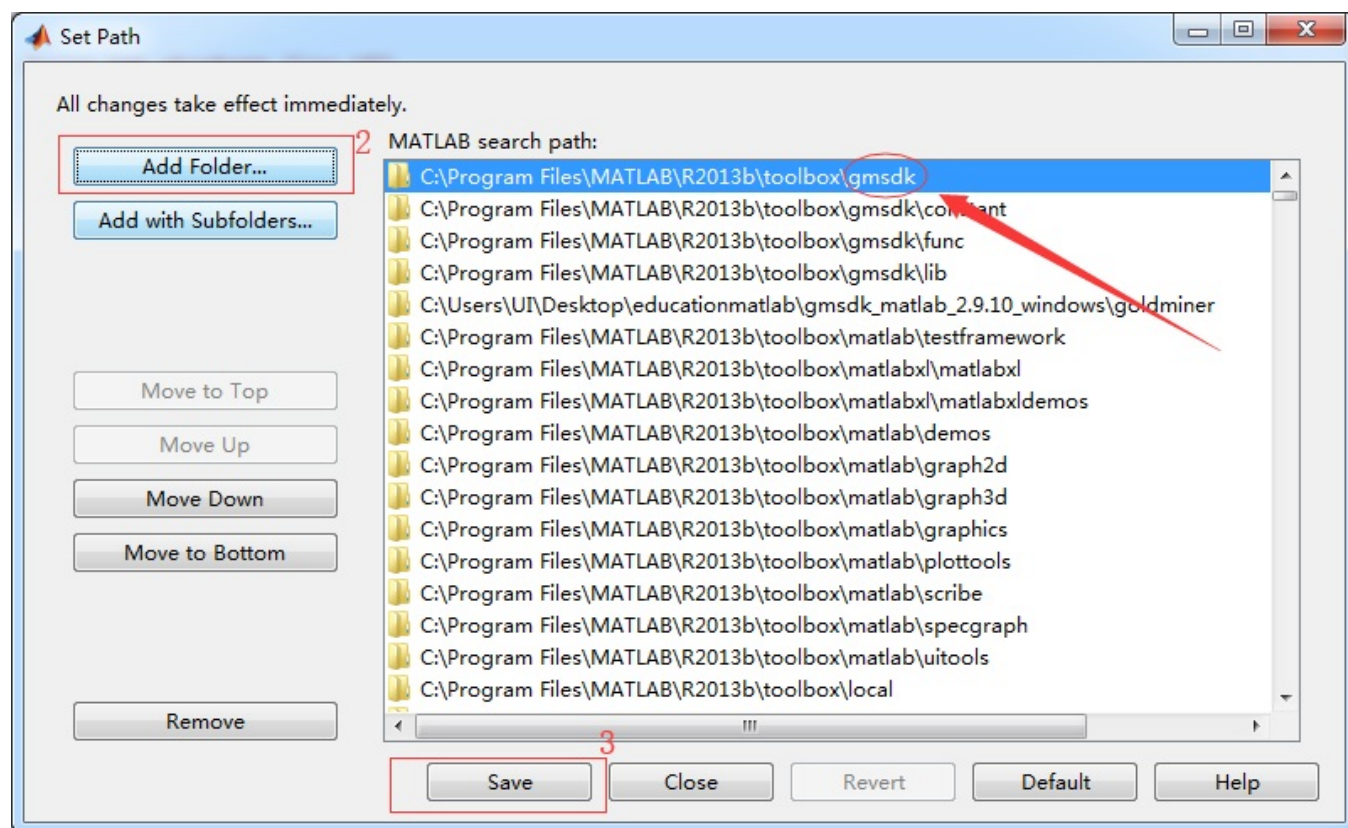
作为掘金3量化接口的一员，matlab语言SDK包含以下特点和功能：

- 矩阵化的数据格式，按时序、标的组成行情矩阵
- 采用单一函数面向过程的策略结构
- 支持行情滑窗和行情数据驱动
- 支持回测、实盘无缝切换
- 支持基本面和业务数据查询
- 支持查询持仓、资金信息
- 统一订单ID，支持下单、撤单和查询
- 支持订单查询和订单回报事件推送

matlab策略 SDK的安装

- matlab策略SDK作为一个matlab工具包，加载到matlab函数环境中即可，具体操作为在matlab主界面使用Set Path, Add Folder添加matlabSDK的工具包路径（SDK在终端内获取）





matlab策略运行

- matlab策略SDK内部已经设置好了与掘金服务器、终端的链接，配置正确token和策略ID后，可以直接运行策略（需要终端处于登录状态），策略运行的回测、仿真交易和实盘信息在终端查看

注意事项

- matlab策略回测时，出现无法暂停的情况，需要重启matlab，断点调试时，可调用stop_stragety函数停止策略

典型策略场景

- 定时事件驱动
- 订阅数据驱动
- 成交回报事件驱动
- 指定账户交易
- 获取资金持仓信息交易策略
- 设定策略运行参数
- 仅提取数据

定时事件驱动

- 初始化策略时设定策略的执行时间
- 到达设定时间时，自动执行策略
- 可以指定多个定时任务，通过任务名进行区分

```

1. function [Context] = main(Context,Event)
2. % 初始化
3. if Event.Init.flag==1
4.     schedule('algo01', '1d', '09:30:00');
5.     schedule('algo02', '1d', '14:55:00');
6.     return
7. end
8. %% 定时任务处理逻辑
9. if Event.algo01.flag==1
10.    disp (Context.now);
11. end
12. if Event.algo02.flag==1
13.    disp (Context.now);
14. end
15. end

```

订阅数据驱动

```

1. function [Context] = main(Context,Event)
2. % 初始化
3. if Event.Init.flag==1
4.     set_symbols({'SZSE.000001', 'SHSE.600000'});
5.     subscribe({}, '900s', 60);
6.     return
7. end
8. if Event.Bar.frequency_900s.flag==1
9.     disp(Context.data.frequency_900s.eob); %打印滑窗数据
10.    disp(Context.data.frequency_900s.symbols);
11.    disp(Context.data.frequency_900s.close);
12. end

```

成交回报事件驱动

- 委托执行后，当委托状态发生变化，如，部分成交或者委托被拒绝时，会触发委托事件

```

1. function [Context] = main(Context,Event)
2. %% 编写策略
3. % 初始化操作
4. if Event.Init.flag==1%第一次启动仅执行初始化部分
5.     % 设置股票池和订阅的数据，如果订阅数据不指定，默认订阅股票池的所有数据
6.     set_symbols({'SZSE.000001','SZSE.000009'});
7.     subscribe({}, '900s',10);
8.     %设置启用的事件，启用委托状态变更事件
9.     set_event({'OrderStatus'})
10.    return
11. end
12. % 定时任务处理逻辑
13. if Event.Bar.frequency_900s.flag==1
14. [order]=order_volume('SZSE.000001', 100, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Market,PositionEffect.PositionEffect_Open,0); %标的，委托量，买入，市
    价，开仓，价格
15. end
16. % 委托状态变化事件
17. if Event.OrderStatus.flag ==1
18.     disp(Event.OrderStatus.data)
19. end
20. end

```

指定账户交易

- 在多账户场景下交易，用指定账户的ID来进行不同账户的下单
- 如果策略是单账户则不需要指定账户ID

```

1. function [Context] = main(Context,Event)
2. % 初始化
3. if Event.Init.flag==1
4.     schedule('algo01', '1d', '09:30:00');
5.     return
6. end
7. %% 定时任务处理逻辑
8. if Event.algo01.flag==1
9.     %标的，资金，买入，市价，开仓，价格，仅在实盘可以指定账户
10.    [order]=order_volume({'SHSE.600000'}, 100, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Market,PositionEffect.PositionEffect_Open,0, 'account1');
11. end
12. end
13. end

```

获取资金持仓信息交易策略

- 获取资金和持仓信息
- 在定时任务中使用资金和持仓信息决策，并进行交易

```

1. function [Context] = main(Context,Event)
2. %% 编写策略
3. % 开盘如果没有持仓，则买入0.8倍可用资金的平安银行股票
4. % 收盘检查可用持仓，如果有可用仓，则全部卖出
5. % 初始化操作
6. if Event.Init.flag==1
7.     schedule('algo01', '1d', '09:30:00');
8.     schedule('algo02', '1d', '14:55:00');
9.     Context.userdata.symbol = 'SZSE.000001'; %userdata, 设定
        用户自定义的全局变量
10.     return
11. end
12. %% 定时任务处理逻辑
13. symbol = Context.userdata.symbol;
14. [position] = get_position(); %position字
        段和info说明一一对应
15. ind = strcmp(position(:,3),symbol);
16. if Event.algo01.flag==1
17.     % 查询资金，买入当前资金的0.1的平安银行股票
18.     [cash]= get_cash(); %cash字段和
        info说明一一对应
19.     if sum(ind)==0 %没有仓位则开
        仓
20.         [order]=order_value(symbol, 0.1*cash{2,9}, 1, 2,1,0); %标的，资金，
        买入，市价，开仓，价格
21.     end
22. end
23. if Event.algo02.flag==1
24.     % 查询持仓，持有平安银行，且为亏损的则清仓
25.     if sum(ind)>0&&position{ind,15}==0
26.         [order]=order_volume(symbol, position{ind,14}, 2, 2,2,0); %标的，数量，
        卖出，市价，平仓，价格
27.     end
28. end
29. end

```

设定策略运行参数

- 策略运行必要参数：登录身份信息（token），策略身份信息（strategy_id）为必填信息，可由终端自动生成
- 回测参数用于控制回测方式运行的起止时间、复权方式、资金、交易信息、缓存等，有默认值
- 服务器地址默认本机地址，需要分机器部署时才需要指定填写

```

1. %% 设置策略运行参数
2. strategy_set.strategy_id = '';
3. strategy_set.token = '';
4. strategy_set.mode = 'MODE_BACKTEST'; %
        MODE_LIVE(实时)=1, MODE_BACKTEST(回测) =2
5. strategy_set.backtest_start_time = '2018-08-01 10:40:00';
6. strategy_set.backtest_end_time = '2018-08-10 10:50:00'; % 默认最近一
        个月
7. strategy_set.backtest_initial_cash = 1000000; % 默认资金一

```



```

    百万
8. % strategy_set.backtest_transaction_ratio = 0; % 默认成交比
    例1
9. % strategy_set.backtest_commission_ratio = 0.001; % 默认手续费
    千—1
10. % strategy_set.backtest_slippage_ratio = 0; % 默认滑点比
    率
11. % strategy_set.backtest_adjust = ADJUST_NONE; % 默认复权方
    式不复权, ADJUST_NONE(不复权)=0, ADJUST_PREV(前复权)=1, ADJUST_POST(后复权)=2
12. % strategy_set.backtest_check_cache = 1; % 默认使用缓
    存, 1 - 使用, 0 - 不使用
13. % strategy_set.serv_addr = ''; % 服务器地址
    可不填
14. %% 运行策略
15. run_strategy(strategy_set)
16. global context

```

仅提取数据

- 设置token信息，用于登录身份认证
- 使用提数接口提取数据

```

1. set_token('xxxxx')
2. % 获取财务数据（表名，标的名，开始时间，结束时间，表字段）
3. get_fundamentals('trading_derivative_indicator',{ 'SHSE.600000','SZSE.000001'}, '2018-
    04-01', '2018-08-01', { 'TCLOSE', 'NEGOTIABLEMV', 'TOTMKTCAP', 'TURNRATE' })

```

典型应用场景

使用财务接口获取指定月份的数据

参考处理代码：

```

1. function data01 = date_fl (data,month)
2.     for i = 1:length(data.symbols)
3.         date = cellstr(datestr(data.EndDate{i}, 'mm'));
4.         ind = strcmp(date,month);
5.         data01.symbols = data.symbols;
6.         diff_f = setdiff(fieldnames(data), 'symbols');
7.         for j = 1:length(diff_f)
8.             jj = diff_f{j};
9.             eval(['data01.',jj,'{i}=data.',jj,'{i}(ind);'])
10.        end
11.    end
12. end

```

完整示例：

```

1. [data] = get_fundamentals('cashflow_statement',{ 'SHSE.600000','SZSE.000001'}, '2016-01-01', '2018-08-01', {'ASSEIMPA','CASHFINALBALA','FINRELACASH'});
2. data01 = date_fl(data, '09')

```

输出结果

```

1. data01 =
2.
3.     包含以下字段的 struct:
4.
5.         symbols: {'SHSE.600000'  'SZSE.000001'}
6.         ASSEIMPA: {[0 0]  [0 0]}
7.         CASHFINALBALA: {[0 0]  [0 0]}
8.         EndDate: {[736603 736968]  [736603 736968]}
9.         FINRELACASH: {[0 0]  [0 0]}
10.        PubDate: {[736632 736996]  [736624 736989]}
11.
12. datestr([736603 736968],31)
13.
14. ans =
15.
16. 2016-09-30 00:00:00
17. 2017-09-30 00:00:00

```

策略组成文件

- 策略编写m文件-main.m
- 策略运行文件-run.m

策略编写m文件-main.m

- main函数为用户编写策略的主函数，策略逻辑需要按照事类型件分块编写，在main中可以获取数据和进行交易
- 当有效事件发生时，main行数将会被自动调用，通过if判断事件类型标识，执行指定类型事件的处理逻辑
- 首次调用为init初始化事件，作用是设定策略运行方式，包括订阅的数据范围，需要的监听的事件类型

```
1. [Context] = main(Context,Event)
2. if Event.Init.flag==1
3.     subscribe()
4. end
5.
6. end
```

参数	类型	说明
Event	struct	存放事件内容，一个事件分为flag标志位和事件数据data，flag为1时表示该事件发生
Context	struct	全局变量,存放策略级的系统数据，也可以存放用户自定义的数据

示例

```
1. function [Context] = main(Context,Event)
2. %% 均线突破策略
3. % 设定交易股票池，分别针对个股做以下决策
4. % 60分钟均线突破15分钟均线作为指标
5. % 如果向上突破，且无持仓，则按一定资金比例买入
6. % 如果向下突破，且有持仓，则卖出所有持仓
7. if Event.Init.flag==1 % 初始化操作
8.     insurance = get_industry('J68');
9.     set_symbols(insurance); %提取保险行业成分股，设定股票池
10.     subscribe(insurance, '60s',60);
11.     subscribe(insurance, '900s',60);
12.     Context.userdata.len_60s = 30; %userdata, 设定用户自定义的全局变量
13.     Context.userdata.len_15m = 20;
14.     return
15. end
16. %% 定时任务处理逻辑
17. len_1d=Context.userdata.len_60s;
18. len_15m=Context.userdata.len_15m;
19. cp_60s = Context.data.frequency_60s.close(:,end-len_1d:end);
20. cp_15m = Context.data.frequency_900s.close(:,end-len_15m:end);
21. if Event.Bar.frequency_900s.flag==1
22.     % 计算突破因子
```

```
23.     fact = mean(cp_60s,2)-mean(cp_15m,2);
24.     % 判断持仓，如果突破因子为正且持仓为空，则下单买入，否则清仓
25.     [cash]= get_cash();                                     %cash字段和
info说明一一对应
26.     [position] = get_position();
27.     for i =1:length(Context.symbols)
28.         ind = strcmp(position(:,3),Context.symbols{i});
29.         ind = find(ind==1);
30.         if fact(i)>0
31.             if sum(ind)==0                %没有仓位则开仓
32.                 [order]=order_value(Context.symbols{i}, 0.15*cash{2,4},
OrderSide.OrderSide_Buy,
OrderType.OrderType_Market,PositionEffect.PositionEffect_Open,0);           %标
的，资金，买入，市价，开仓，价格，市场场景下价格字段不生效
33.             end
34.         else
35.             if sum(ind)~=0                %没有仓位则开
仓
36.                 [order]=order_volume(Context.symbols{i}, position{ind,14},
OrderSide.OrderSide_Sell,
OrderType.OrderType_Market,PositionEffect.PositionEffect_Close,0);           %标的，
资金，买入，市价，开仓，价格
37.             end
38.         end
39.     end
40. end
```

策略运行文件—run.m

run函数用于运行策略，通过参数控制策略的运行模式、回测参数、身份信息等

参数说明：

参数名	是否必填	描述
strategy_set.strategy_id	必填	策略ID用于终端识别策略身份
strategy_set.mode	非必填	策略运行的时间模式，MODE_LIVE(实时), MODE_BACKTEST(回测)
strategy_set.token	必填	账户身份信息
strategy_set.backtest_start_time	非必填	回测模式开始时间
strategy_set.backtest_end_time	非必填	回测模式结束时间
strategy_set.backtest_initial_cash	非必填	回测模式的初始资金

strategy_set.backtest_transaction_ratio	非必填	回测模式的成交比例，默认为1
strategy_set.backtest_commission_ratio	非必填	回测模式的手续费，默认为0.001
strategy_set.backtest_slippage_ratio	非必填	回测模式的滑点比率，默认为0
strategy_set.backtest_adjust	非必填	回测数据的复权方式，ADJUST_NONE(不复权)=0，ADJUST_PREV(前复权)=1，ADJUST_POST(后复权)=2
strategy_set.backtest_check_cache	非必填	回测模式使用缓存数据 默认使用缓存，1 - 使用， 0 - 不使用
strategy_set.serv_addr	非必填	默认为本地终端服务地址，可不填

函数原型：

```

1. %% 设置策略运行参数
2. strategy_set.strategy_id = 'xxxxxxx';
3. strategy_set.token = 'xxxxxx';
4. strategy_set.mode = 'MODE_BACKTEST'; %
   MODE_LIVE(实时), MODE_BACKTEST(回测)
5. strategy_set.backtest_start_time = '2018-08-01 10:40:00';
6. strategy_set.backtest_end_time = '2018-08-10 10:50:00'; % 默认最近一
   个月
7. strategy_set.backtest_initial_cash = 1000000; % 默认资金一
   百万
8. % strategy_set.backtest_transaction_ratio = 0; % 默认成交比
   例1
9. % strategy_set.backtest_commission_ratio = 0.001; % 默认手续费
   千一1
10. % strategy_set.backtest_slippage_ratio = 0; % 默认滑点比
   率
11. % strategy_set.backtest_adjust = ADJUST_NONE; % 默认复权方
   式不复权, ADJUST_NONE(不复权)=0, ADJUST_PREV(前复权)=1, ADJUST_POST(后复权)=2
12. % strategy_set.backtest_check_cache = 1; % 默认使用缓
   存, 1 - 使用, 0 - 不使用
13. % strategy_set.serv_addr = ''; % 服务器地址
   可不填
14. %% 运行策略
15. run_strategy(strategy_set)
16. global context % 显示全局变
   量内的信息

```

说明： run_strategy用于启动运行策略，设置好策略运行的模式，并将策略初始化函数中订阅的数据和定时的任务设定到策略驱动中

策略结构要素

- 全局变量
- 初始化事件
- 设置滑窗标的序列
 - 设置驱动事件
- 定时运行函数和定时事件
- 订阅数据滑窗和数据事件
- 交易事件驱动
- 其他事件驱动
- 存储自定义全局变量
- 策略停止函数–stop_strategy

main函数内主要由事件驱动、全局变量、数据接口和交易接口组成的策略编基本接口，用户根据这些基本接口来组织策略的结构

全局变量

全局变量存放了系统变量，包括订阅数据滑窗、策略运行时间、标的、事件等也用于存放用户自定义的全局变量
函数原型：

```
1. % 全局变量直接赋值即可
2. symbols = Context.symbols;
3. nowtime = Context.now;
4. data_15close = Context.data.frequency_900s.close;
```

说明：

变量名	类型	描述
symbols	cell	滑窗数据的标的列表，通过set_symbols进行设定，必须在订阅数据前设定，否则可能导致数据滑窗数据缺失
now	cell	字符格式的当前时间点
data	struct	订阅数据滑窗，按照频率、字段、数据矩阵结构存储，如Context.data.frequency_900s.close，行索引为symbols，列索引eob
event	struct	存放系统使用事件的标识和参数
userdata	struct	用户保存的全局变量，可在后续周期中调用

初始化事件

初始化事件是策略逻辑启动前，初始化策略运行条件
策略的运行方式采用事件触发的方式，当检测到事件发生时则启动对应的事件逻辑，初始化一般用于设定策略需要的驱动方式，如定时驱动，订阅滑窗数据驱动，成交回报驱动等

原型

```
1. if Event.Init.flag==1
2.     % 编写初始化内容
```

```
3. end
```

参数说明

变量名	类型	描述
Event.Init.flag	mat	初始化事件的标识，当初始化调用main时，Event.Init.flag==1

注意

Init事件是启动策略必须执行事件，一般执行一次，在其条件内进行数据订阅、定时任务的启动

设置滑窗标的序列

```
1. set_symbols(symbols);
```

说明：symbols集合用于生成统一的滑窗数据，保证滑窗数据矩阵行索引和symbols对齐，便于数据的矩阵运算

设置驱动事件

原型

```
1. set_event( event_name ,bool);
```

说明：具体事件名称查看后续事件说明内容

定时运行函数和定时事件

定时功能的使用由两部分组成

- 定时任务设置函数
- 定时任务接收函数

```
1. % 定时任务设置函数
2. schedule(schedule_func, date_rule, time_rule);
3. % 定时任务接收函数
4. if Event.schedule_func.flag==1
5. % 定时任务处理逻辑
6. end
```

参数说明

变量名	类型	描述
schedule_func	char	策略定时执行的事件名称
date_rule	char	定时日期，格式为'n'+ 'd/w/m'，目前仅支持数字1，字母分别代表日，周，月
time_rule	char	定时事件，格式为'hh:mm:ss'

示例

```

1. % 每个交易日定时启动两个不同的定时任务，在任务中查看任务启动的时间点
2. if Event.Init.flag==1
3.     schedule('algo01', '1d', '09:30:00');
4.     schedule('algo02', '1d', '14:55:00');
5.     return
6. end
7. %% 定时任务处理逻辑
8. if Event.algo01.flag==1
9.     disp (Context.now);
10. end
11. if Event.algo02.flag==1
12.     disp (Context.now);
13. end

```

注意

在使用定时函数schedule定时成功后，会在Event事件集中自动生成定时函数中的同名标识；
定时任务可以启动多个

订阅数据滑窗和数据事件

数据事件驱动的使用由两部分组成

- 订阅数据函数
- 数据事件接收函数

订阅行情数据

函数原型：

```

1. % 设置数据滑窗的标的列表
2. set_symbols(symbols)
3. % 订阅函数
4. subscribe(symbols, frequency, count, is_event)
5. % 接收函数
6. if Event.Bar.flag==1%
7.
8. end

```

参数说明

变量名	类型	描述
symbols	cell	策略定时执行的事件名称
frequency	mat	订阅数据的频率，支持支持 'tick', '1d', 'Ns'
count	mat	订阅的滑窗数据的长度
is_event	mat	是否作为事件方式进行驱动，1表示数据到达时驱动，0表示不驱动（暂只针对tick数据订阅生效）

示例

```

1. function [Context] = main(Context, Event)
2. % 初始化

```



```
3. if Event.Init.flag==1
4.     set_symbols({'SZSE.000001', 'SHSE.600000'});
5.     subscribe({}, '900s', 60, 1, {});
6.     return
7. end
8. if Event.Bar.frequency_900s.flag==1
9.     disp(Context.data.frequency_900s.eob); %打印滑窗数据
10.    disp(Context.data.frequency_900s.symbols);
11.    disp(Context.data.frequency_900s.close);
12. end
```

说明

滑窗标的列表设置的作用是规定滑窗数据的以相同的标的行进行数据对齐
订阅函数subscribe中，symbols不填默认使用滑窗标的列表的全部标的进行订阅
行情字段参数暂时不生效
数据滑窗时间判断表示为Event.Bar.flag, 支持独立的频率时间驱动，使用Event.Bar.frequency_900s.flag

交易事件驱动

交易事件的使用由三部分组成

- 使用set_event启动交易事件类型
- 接收交易事件驱动
- 获取交易事件信息

交易事件类型

交易事件	说明
OrderStatus	委托状态更新事件，响应委托状态更新事情，下单后及委托状态更新时被触发。
ExecutionReport	委托执行回报事件，响应委托被执行事件，委托成交后被触发。
AccountStatus	交易账户状态更新事件

函数原型

```
1. %设置启用的事件，启用委托状态变更事件
2.     set_event({'OrderStatus'})
3. % 委托状态变化事件
4. if Event.OrderStatus.flag ==1
5.     disp(Event.OrderStatus.data); %打印事件内容
6. end
```

说明：

事件数据格式参照交易对象说明
在回测模式下，交易事件和下单执行函数是同步生效的，即下单后会立刻获取交易事件

其他事件驱动

其他需要策略处理的事件使用方式

- 使用set_event启动事件类型

存储自定义全局变量

- 接收事件驱动
- 获取事件内容

其他事件类型

事件名称	说明
BacktestFinished	回测结束事件
Error	错误事件
MarketDataConnected	实时行情网络连接成功事件
MarketDataDisconnected	实时行情网络连接断开事件
TradeDataConnected	交易通道网络连接成功事件
TradeDataDisconnected	交易通道网络连接断开事件

函数原型

```
1. % 开启回报事件
2. set_event({'BacktestFinished','Error','MarketDataConnected','TradeDataConnected',...
3. 'MarketDataDisconnected','TradeDataDisconnected'})
4. % 回测结束事件
5. if Event.BacktestFinished.flag ==1
6. % 回测结束事件
7. data = Event.BacktestFinished.data
8. end
9. % 错误事件
10. if Event.Error.flag ==1
11.
12. end
13. % 行情连接事件
14. if Event.MarketDataConnected.flag ==1
15.
16. end
```

存储自定义全局变量

策略中用于下次决策使用或多周期使用的数据可以保存在自定义全局变量中

函数原型

```
1. % 本次保存
2. Context.userdata = user_data;
3. % 下次使用
4. user_data = Context.userdata ;
```

说明

Context.userdata 可以用作结构体存储多个变量

策略停止函数—stop_strategy

用于停止策略，可用在仿真的断点调试，连续回测是无法停止策略，需要关闭matlab程序

函数原型

```
1. stop_strategy()
```

获取数据

- [bar 数据结构说明](#)
- [tick 数据结构说明](#)
- [财务数据及其他业务数据结构](#)
- [subscribe—订阅方式获取决策数据](#)
- [current - 查询当前行情快照](#)
- [history - 查询固定时间历史行情](#)
- [history_n - 查询固定长度历史行情](#)
- [get_fundamentals - 查询固定时间基本面数据](#)
- [get_fundamentals_n - 查询固定长度基本面数据](#)
- [get_instruments - 查询最新交易标的最新基本信息](#)
- [get_history_instruments - 查询交易标的历史基本信息](#)
- [get_instrumentinfos - 查询交易标的基本信息](#)
- [get_constituents - 查询指数成份股信息](#)
- [get_industry - 查询行业板块成分股](#)
- [get_trading_dates - 查询交易日历表](#)
- [get_previous_trading_date - 查询上一个交易日](#)
- [get_next_trading_date - 查询下一个交易日](#)
- [get_dividend - 查询分红送配信息](#)
- [get_continuous_contracts - 获取主力连续合约](#)

bar 数据结构说明

- bar数据采用结构体+二维矩阵形式存储，通过策略的全局变量Context.data获取或调用历史行情接口history获取
 - 数据矩阵的行索引为symbols字段
 - 列索引eob字段
- 函数原型

```
1. % 数据滑窗获取bar行情的频率为60s的close字段
2. Context.data.frequency_60s.close
```

字段描述

字段名	类型	描述
symbols	cell	标的代码
eob	cell	字符格式的bar结束时间点
eobnum	mat	数字格式的bar结束时间点
open	mat	开盘价
high	mat	最高价
low	mat	最低价
close	mat	收盘价
amount	mat	成交额
volume	mat	成交量
position	mat	持仓量（仅期货）

示例

```
1. Context.data.frequency_60s =
2.
3.     symbols: {'SZSE.000001' 'SHSE.600000'}
4.     eob: {1x5510 cell}
5.     eobnum: [1x5510 double]
6.     open: [2x5510 double]
7.     high: [2x5510 double]
8.     low: [2x5510 double]
9.     close: [2x5510 double]
10.    amount: [2x5510 double]
11.    volume: [2x5510 double]
12.    position: [2x5510 double]
```

tick 数据结构说明

- tick数据采用结构体+一维表结构格式存储，通过策略的全局变量Context.data.tick获取或调用历史行情接口history获取
 - 数据表的行索引为symbols字段
 - 列索引为二维的eob+symbols
- 函数原型

```
1. % 获取tick行情的close字段
2. Context.data.frequency_60s.close
```

字段描述

字段名	类型	描述
symbols	cell	标的代码
open	cell	开盘价
high	cell	最高价
low	cell	最低价
close	cell	收盘价
cumVolume	cell	成交总量/最新成交量, 累计值
cumAmount	cell	成交总金额/最新成交额, 累计值
tradeType	cell	交易类型 1: '双开', 2: '双平', 3: '多开', 4: '空开', 5: '空平', 6: '多平', 7: '多换', 8: '空换'
lastVolume	cell	瞬时成交量
cumPosition	cell	合约持仓量(期), 累计值 (股票此值为0)
lastAmount	cell	瞬时成交额
createdAt	cell	创建时间
quotes	struct	股票提供买卖5档数据, 包含买卖到五档行情

其中五档报价quote结构如下：

--	--	--

字段名	类型	描述
bidPrice	cell	委买价五档
bidVolume	cell	委买量五档
askPrice	cell	委卖价五档
askVolume	cell	委卖量五档

注意：

- tick数据各字段的为cell格式，获取数据时需要先用symbols索引找到对应行位置，用created_at找到数据的列位置
- 可能会有买档或卖档报价缺失，比如跌停时无买档报价（没有bid_p, bid_v），涨停时无卖档报价（没有ask_p, ask_v）
- 集合竞价时tick的买卖价均为0

财务数据及其他业务数据结构

财务数据采用和tick数据相同的结构来存储，每个财务指标独立，以symbols列进行索引

业务数据采用cell表结构存储，首行为字段名，具体格式参照接口示例

subscribe—订阅方式获取决策数据

订阅获取决策数据由三部分组成

- 发起订阅数据
- 事件数据获取
- 时序数据获取

订阅数据

参数说明

函数原型：

```
1. # 设置滑窗股票标的，订阅行情数据
2. set_symbols('SZSE.000001');
3. subscribe(symbols, frequency, count, wait_group, wait_group_timeout,
   unsubscribe_previous)
```

事件数据获取

行情事件分为on_tick和on_bar事件

```
1. if Event.Bar.flag == 1
2.     .....
3. end
```

```
1. if Event.Tick.flag == 1
2.     data_event = Event.on_tick.data
3.     .....
4. end
```

时序数据获取

```
1. if Event.on_bar.flag == 1
2.     # 数据存放在全局变量中
3.     data = Context.data.frequency_1d.close
4.     data_time = Context.data.frequency_1d.eob
5.     symbols = Context.data.frequency_1d.symbols
6.     .....
7. end
```

subscribe订阅函数说明

参数名	类型	说明
symbols	cell	证券代码，cell 格式，支持一个或多个，如{'SZSE.000001'}
frequency	char	实时订阅数据的频率，支持 'tick'，'1d'，'15s'，'30s'，回测历史数据支持任意频率
count	mat	数据滑窗的长度，正整数
wait_group	bool	是否到齐方式驱动，仅对bar数据驱动生效；1表示到齐后驱动，0表示单个bar到达时驱动
wait_group_timeout	int	在到齐方式驱动式，最大等待时间
unsubscribe_previous	bool	填入true表示取消前订阅，false表示不取消,默认为取消前一次订阅

set_symbols设定滑窗标的的池函数

在订阅数据前需要设置标准滑窗的标的列表，用于确定策略需要接收的订阅数据

参数名	类型	说明
symbols	cell	证券代码，cell 格式，支持一个或多个，如{'SZSE.000001'}，

说明

标的声明用于标准化滑窗数据的symbols索引，即订阅的数据滑窗按照该排列顺序进行填充

注意：

策略在当次运行中，在回测模式时是不可以多次订阅数据的，过程中需要的标的和数据都需要预先订阅好；在实时状态时，每次重新订阅都需要set_symbols设置当前需要订阅的标的，否则策略不会接收新的标的（直接取数接口无此类限制）
示例

```
1. function [Context] = main(Context,Event)
2.
3. % 初始化操作
4. if Event.Init.flag==1
5.     set_symbols('SZSE.000001');
6.     subscribe({}, '900s', 60, 1, 1, 10, true);
7.     return
8. end
9. if Event.frequency_1d.flag==1
10. % 获取15分钟收盘价时间序列数据
11.     cp_15m = Context.data.frequency_900s.close;
12.     time_15m = Context.data.frequency_900s.eobnum;
13. end
```

unsubscribe 退订函数说明

参数名	类型	说明
symbols	cell	证券代码，cell 格式，支持一个或多个，如{'SZSE.000001'}
frequency	char	订阅数据的频率，频率，支持 'tick'，'1d'，'15s'，'30s'

示例

```
1. unsubscribe( 'SHSE.600000', '60s')
```

current - 查询当前行情快照

查询当前行情快照，返回tick数据，回测时，返回回测时间点的tick数据

函数原型：

```
1. data = current(symbols,fields)
```

参数名	类型	说明
symbols	cell	查询代码，如{'symbol1'，'symbol2'}

示例

```
1. tick = current({'SZSE.000001'})
```

返回值：

```
1. tick =
2.
3.     symbols: {'SZSE.000001'}
4.     createdAt: {[1x1 cell]}
5.     price: {[10.2000]}
6.     open: {[10.0300]}
7.     high: {[10.2700]}
8.     low: {[10.0300]}
9.     cumVolume: {[64832749]}
10.    cumAmount: {[6.5857e+08]}
11.    cumPosition: {[0]}
12.    lastAmount: {[118325]}
13.    lastVolume: {[11600]}
14.    tradeType: {[8]}
15.    quotes: [1x1 struct]
```

说明：

- 1. 若输入包含无效标的代码，无效代码对应的数据为空
- 2. 仅在策略实时运行模式下获取行情tick快照数据，其他情况下数据为0
- 3. fields用于选择需要的字段，取较少的字段可以大幅提高取数效率

history - 查询固定时间历史行情

按起始日期区间查询历史行情数据

函数原型：

```
1. [ data ] = history( symbols, frequency, start_time, end_time,adjust, adjust_end_time)
```

说明：

参数名	类型	说明
symbols	cell	查询代码，如{'symbol1', 'symbol2'}
frequency	char	频率，支持 'tick', '1d', '15s', '30s' 等，默认 '1d'，详情见股票行情数据和期货行情数据
start_time	char	开始时间 (YY-MM-DD hh-mm-ss格式),
end_time	char	结束时间 (YY-MM-DD hh-mm-ss格式)
adjust	char	ADJUST_NONE : 不复权, ADJUST_PREV : 前复权, ADJUST_POST : 后复权 默认不复权
adjust_end_time	char	复权基点时间，默认当前时间
skip_suspended	bool	是否跳过停牌，默认跳过 (true或者false，暂不支持)

示例

```
1. [data] = history({'SZSE.000001','SHSE.600000'},'60s','2018-08-01','2018-09-01','ADJUST_NONE')
```

返回值：

```
1. data =
2.
3.     symbols: {'SZSE.000001' 'SHSE.600000'}
4.     eob: {1x5510 cell}
5.     eobnum: [1x5510 double]
6.     open: [2x5510 double]
7.     high: [2x5510 double]
8.     low: [2x5510 double]
9.     close: [2x5510 double]
10.    amount: [2x5510 double]
11.    volume: [2x5510 double]
12.    position: [2x5510 double]
```

说明：

1. 若输入包含无效标的代码，无效代码对应的数据为空

history_n - 查询固定长度历史行情

函数原型：

```
1. [ data ] = history_n( symbols, frequency, count, end_time, adjust, adjust_end_time, skip_suspended, fields)
```

说明：

参数名	类型	说明
symbols	cell	查询代码，如{'symbol1', 'symbol2'}
frequency	char	频率，支持 'tick', '1d', '15s', '30s' 等，默认 '1d'，详情见股票行情数据和期货行情数据
count	double	获取数据的长度
end_time	char	结束时间（YY-MM-DD hh-mm-ss格式）
adjust	char	ADJUST_NONE ：不复权，ADJUST_PREV ：前复权，ADJUST_POST ：后复权 默认不复权
adjust_end_time	char	复权基点时间，默认当前时间
skip_suspended	bool	是否跳过停牌，默认跳过（true或者false）

示例

```
1. [data] = history_n({'SZSE.000001', 'SHSE.600000'}, '60s', 1000, '2018-09-01', 'ADJUST_NONE')
```

返回值：

```
1. data =
2.
3.     symbols: {'SZSE.000001' 'SHSE.600000'}
4.     eob: {1x1000 cell}
5.     eobnum: [1x1000 double]
6.     open: [2x1000 double]
7.     high: [2x1000 double]
8.     low: [2x1000 double]
9.     close: [2x1000 double]
10.    amount: [2x1000 double]
11.    volume: [2x1000 double]
12.    position: [2x1000 double]
```

get_fundamentals - 查询固定时间基本面数据

函数原型：

```
1. [ data ] = get_fundamentals( table, symbols, start_date, end_date, fields )
```

说明：

示例

```
1. [data] = get_fundamentals('trading_derivative_indicator',{
    'SHSE.600000', 'SZSE.000001'}, '2018-04-01', '2018-08-01',
    {'TCLOSE', 'NEGOTIABLEMV', 'TOTMKTCAP', 'TURNRATE'})
```

返回值：

```
1. data =
2.
3.     symbols: {2x1 cell}
4.     pub_adte: {2x1 cell}
5.     EndDate: {2x1 cell}
6.     TCLOSE: {2x1 cell}
7.     NEGOTIABLEMV: {2x1 cell}
8.     TOTMKTCAP: {2x1 cell}
9.     TURNRATE: {2x1 cell}
```

说明：
如果查询基本面数据接口的查询字段出错，则直接返回错误（1017错误）

get_fundamentals_n - 查询固定长度基本面数据

函数原型：

```
1. [data] = get_fundamentals_n( table, symbols, count, end_date, fields )
```

说明：

参数名	类型	说明
table	char	表名，只支持单表查询。具体表名及fields字段参考 财务数据文档
symbols	cell	标的代码，多个代码可用，(英文逗号)分割，也支持 ['symbol1', 'symbol2'] 这种列表格式
count	char	每个股票数据的数量
end_date	char	结束时间，(YY-MM-DD 格式)
fields	str	查询字段（必填）

示例

```
1. data = get_fundamentals_n('trading_derivative_indicator',{
    'SHSE.600000','SZSE.000001'}, '2017-04-01', 3,
    {'TCLOSE','NEGOTIABLEMV','TOTMKTCAP','TURNRATE'})
```

返回值：

```
1. data =
2.
3.     symbols: {2x1 cell}
4.     pub_adte: {2x1 cell}
5.     EndDate: {2x1 cell}
6.     TCLOSE: {2x1 cell}
7.     NEGOTIABLEMV: {2x1 cell}
8.     TOTMKTCAP: {2x1 cell}
9.     TURNRATE: {2x1 cell}
```

说明：

get_instruments - 查询最新交易标的最新基本信息

查询最新交易标的信息, 有基本数据及最新日频数据

函数原型:

```
1. [ data ] = get_instruments( symbols , exchanges , sec_types , fields )
```

说明:

参数名	类型	说明
symbols	cell	标的代码, 多个代码可用 , (英文逗号)分割, 也支持 ['symbol1', 'symbol2'] 这种列表格式
exchanges	char	交易所代码, 多个交易所代码可用 , (英文逗号)分割, NULL表示所有
secTypes	mat	标的类型, 1: 股票, 2: 基金, 3: 指数, 4: 期货, 5: 期权, 8: 可转债, 10: 虚拟合约, 只能输入单个品种
fields	int	查询字段, 默认表示所有

示例

```
1. data = get_instruments({'SZSE.000001'}, 'SZSE', 1)
```

返回值:

```
1. data =
2.
3. Columns 1 through 8
4.
5. 'symbol'      'sec_level'    'is_suspended'  'multiplier'    'margin_ratio'
6. 'settle_price' 'position'      'pre_close'
7. 'SZSE.000001' [      1]      [      0]      [      1]      [      1]
8. [      0]      [      0]      [ 9.9600]
9.
10. Columns 9 through 13
11.
12. 'pre_settle'   'upper_limit'   'lower_limit'   'adj_factor'    'created_at'
13. [      0]      [ 10.9600]      [ 8.9600]      [ 117.1530]     '2018-09-13'
```

get_history_instruments - 查询交易标的历史基本信息

返回指定symbols的标的日频历史数据

函数原型:

```
1. [ data ] = get_history_instruments( symbols , start_date , end_date , fields )
```

说明:

参数名	类型	说明
-----	----	----

symbols	string	标的代码，多个代码可用，(英文逗号)分割，也支持 ['symbol1', 'symbol2'] 这种列表格式，是必填参数
startDate	char	开始时间 (YY-MM-DD 格式)
endDate	char	结束时间 (YY-MM-DD 格式)
fields	cell	查询字段，默认表示所有

示例

```
1. [ data ] = get_history_instruments({'SZSE.000001', 'SZSE.000002'}, '2018-04-19', '2018-06-19')
```

返回值：

```
1.
2. data =
3.
4.     symbols: {2x1 cell}
5.     created_at: [1x41 double]
6.     sec_level: [2x41 double]
7.     is_suspended: [2x41 double]
8.     multiplier: [2x41 double]
9.     margin_ratio: [2x41 double]
10.    settle_price: [2x41 double]
11.    position: [2x41 double]
12.    pre_close: [2x41 double]
13.    pre_settle: [2x41 double]
14.    upper_limit: [2x41 double]
15.    lower_limit: [2x41 double]
16.    adj_factor: [2x41 double]
```

get_instrumentinfos - 查询交易标的基本信息

获取到交易标的基本信息

函数原型：

```
1. [ data ] = get_instrumentinfos ( symbols, exchanges, sec_types, names, fields )
```

说明：

参数名	类型	说明
symbols	cell	标的代码，多个代码可用，(英文逗号)分割，也支持 ['symbol1', 'symbol2'] 这种列表格式
exchanges	char	交易所代码，多个交易所代码可用，(英文逗号)分割，NULL表示所有
secTypes	mat	标的类型， 1：股票， 2：基金， 3：指数， 4：期货， 5：期权， 8：可转债， 10：虚拟合约，只能输入单个品种
names	char	查询名称，默认所有名称字符
fields	int	查询字段，默认表示所有

示例

```
1. [ data ] = get_instrumentinfos ({'SZSE.000001'}, 'SZSE')
```

返回值:

```
1. data =
2.
3.      'symbol'      'sec_type'      'exchange'      'sec_id'      'sec_name'      'price_tick'
   'listed_date'    'delisted_date'
4.      'SZSE.000001' [      1]      'SZSE'      '000001'      '平安银行'      [      0.0100]
   '1991-04-02'      '2037-12-31'
```

get_constituents - 查询指数成份股信息

函数原型:

```
1. instruments = get_constituents(index, trade_date)
```

参数:

参数名	类型	说明
index	char	指数代码
trade_date	char	结束时间 (YY-MM-DD 格式)

示例

```
1. instruments1 = get_constituents('SHSE.000300', '2020-10-29');
```

返回值

```
1. instruments1 =
2.
3.      301×3 cell 数组
4.
5.      {'created_at'}      {'symbol'      }      {'weight'}
6.      {'2020-09-30'}      {'SHSE.600999'}      {[0.4900]}
7.      {'2020-09-30'}      {'SHSE.601238'}      {[0.0500]}
8.      {'2020-09-30'}      {'SZSE.002241'}      {[0.5600]}
9.      {'2020-09-30'}      {'SZSE.002032'}      {[0.0800]}
10.     {'2020-09-30'}      {'SHSE.600362'}      {[0.0900]}
11.     {'2020-09-30'}      {'SHSE.600703'}      {[0.3300]}
12.     {'2020-09-30'}      {'SHSE.600585'}      {[0.8100]}
13.     {'2020-09-30'}      {'SHSE.601138'}      {[0.1700]}
14.     {'2020-09-30'}      {'SHSE.600221'}      {[0.1200]}
15.     {'2020-09-30'}      {'SHSE.600061'}      {[0.1600]}
16.     {'2020-09-30'}      {'SZSE.000977'}      {[0.1900]}
17.     {'2020-09-30'}      {'SZSE.002673'}      {[0.1000]}
18.     {'2020-09-30'}      {'SZSE.000876'}      {[0.3600]}
```

```

19.      {'2020-09-30'}      {'SHSE.600115'}      {[0.1400]}
20.      {'2020-09-30'}      {'SHSE.600741'}      {[0.2400]}
21.      {'2020-09-30'}      {'SHSE.601398'}      {[1.0600]}
22.      {'2020-09-30'}      {'SHSE.601186'}      {[0.2300]}
23.      {'2020-09-30'}      {'SHSE.600928'}      {[0.0400]}
24.      {'2020-09-30'}      {'SHSE.601888'}      {[1.3300]}
25.      {'2020-09-30'}      {'SZSE.002410'}      {[0.4000]}
26.      .....

```

get_industry - 查询行业板块成分股

函数原型：

```
1. [ data ] = get_industry( code )
```

说明

参数名	类型	说明
code	char	行业代码 不区分大小写（仅支持输入单一代码）

示例

```
1. [ data ] = get_industry( 'j6' )
```

返回值

```

1. data =
2.
3.      'SHSE.600000'
4.      'SHSE.600016'
5.      'SHSE.600030'
6.      'SHSE.600036'
7.      'SHSE.600053'
8.      'SHSE.600061'
9.      'SHSE.600109'
10.     'SHSE.600155'
11.     'SHSE.600291'
12.     'SHSE.600318'
13.     .....

```

get_trading_dates - 查询交易日历表

函数原型：

```
1. [ data ] = get_trading_dates( exchange, start_date, end_date )
```

说明

--	--	--

参数名	类型	说明
exchange	char	见交易市场代码
start_date	char	开始时间 (YY-MM-DD 格式)
end_date	char	结束时间 (YY-MM-DD 格式)

示例

```
1. [ data ] = get_trading_dates('SZSE', '2018-01-01', '2018-08-30')
```

返回值

```
1. data =
2.
3.     datestr: {1x163 cell}
4.     datenum: [1x163 double]
```

说明:

返回值会返回不同格式的日期类型，字符格式和matlab数值格式

get_previous_trading_date - 查询上一个交易日

函数原型:

```
1. [ data ] = get_previous_trading_date( exchange, date )
```

说明

参数名	类型	说明
exchange	char	见交易市场代码
start_date	char	当前日期(YY-MM-DD 格式)

示例

```
1. [ data ] = get_previous_trading_date('SZSE', '2018-08-30')
```

返回值

```
1. data =
2.
3.     datestr: '2018-08-28'
4.     datenum: 7.3709e+05
```

get_next_trading_date - 查询下一个交易日

函数原型:


```
1. [ data ] = get_next_trading_date(exchange, date)
```

示例

```
1. [ data ] = get_next_trading_date('SZSE', '2018-08-31')
```

返回值

```
1. data =
2.
3.     datestr: '2018-09-02'
4.     datenum: 7.3706e+05
```

get_dividend - 查询分红送配信息

函数原型:

```
1. = get_dividend(symbol, start_date, end_date=None)
```

示例

```
1. [ data ] = get_dividend({'SHSE.600000'}, '2015-01-01', '2018-08-31')
```

返回值

```
1. data =
2.
3.     'SHSE.600000'    [0.7570]    [0]    [ 0]    [0]    [0]    '2015-06-22'
4.     'SHSE.600000'    [0.5150]    [0]    [0.1000]    [0]    [0]    '2016-06-22'
5.     'SHSE.600000'    [0.2000]    [0]    [0.3000]    [0]    [0]    '2017-05-24'
6.     'SHSE.600000'    [0.1000]    [0]    [ 0]    [0]    [0]    '2018-07-12'
```

get_continuous_contracts - 获取主力连续合约

函数原型:

```
1. [data] = get_continuous_contracts(csymbols, start_date=None, end_date=None)
```

示例

```
1. get_continuous_contracts({'SHFE.AG'}, '2018-07-01', '2018-08-01')
```

返回值

```
1. data =
2.
```

3.	'symbol'	'created_at'
4.	'SHFE.ag1812'	'2018-06-30'
5.	'SHFE.ag1812'	'2018-07-01'
6.	'SHFE.ag1812'	'2018-07-02'
7.	'SHFE.ag1812'	'2018-07-03'
8.	'SHFE.ag1812'	'2018-07-04'
9.	'SHFE.ag1812'	'2018-07-05'
10.	'SHFE.ag1812'	'2018-07-06'
11.	'SHFE.ag1812'	'2018-07-07'
12.	'SHFE.ag1812'	'2018-07-08'
13.	'SHFE.ag1812'	'2018-07-09'
14.	

策略交易

- [订单数据结构说明](#)
- [回报数据结构说明](#)
- [持仓数据结构说明](#)
- [资金数据结构说明](#)
- [order_volume](#) - 按指定量委托
- [order_value](#) - 按指定价值委托
- [order_percent](#) - 按总资产指定比例委托
- [order_target_volume](#) - 调仓到目标持仓量
- [order_target_value](#) - 调仓到目标持仓额
- [order_target_percent](#) - 调仓到目标持仓比例（总资产的比例）
- [order_batch](#) - 批量委托
- [order_cancel](#) - 撤销指定委托
- [order_cancel_all](#) - 撤销所有委托
- [order_close_all](#) - 平当前所有可平持仓
- [get_unfinished_orders](#) - 查询日内全部未结委托
- [get_orders](#) - 查询日内全部委托
- [get_execution_reports](#) - 查询日内全部执行回报
- [get_position](#)-持仓查询
- [get_cash](#)-资金查询

策略交易相关的功能包括：

- 交易接口、查询委托、查询交易回报
 - 查询资金信息、查询持仓信息
- 交易接口面对策略的函数形式保持不变，策略运行与回测、仿真、实盘时，接口底层自动适配不同交易通道，策略无需做额外处理

订单数据结构说明

订数据是由交易接口返回的数据集，在接口调用成功时即生成，识别主键为clOrdId，随之委托的成交进度，委托单的状态会不断更新，但主键保持不变

属性	类型	说明
strategyId	char	策略ID
accountId	char	账号ID
accountName	char	账户登录名
clOrdId	char	委托客户端ID，下单生成，固定不变，这个才是策略识别的主键
orderId	char	委托柜台ID（系统字段）
exOrdId	char	委托交易所ID（系统字段）
symbol	char	标的代码
side	double	买卖方向 取值参考 OrderSide
positionEffect	double	开平标志 取值参考 PositionEffect
positionSide	double	持仓方向 取值参考 PositionSide
orderType	double	委托类型 取值参考 OrderType
orderDuration	double	委托时间属性 取值参考 OrderDuration

orderQualifier	double	委托成交属性 取值参考 OrderQualifier
orderSrc	double	委托来源（系统字段）
status	double	委托状态 取值参考 OrderStatus
ordRejReason	double	委托拒绝原因 取值参考 OrderRejejectReason
ordRejReasonDetail	char	委托拒绝原因描述
price	double	委托价格
stopPrice	double	委托止损/止盈触发价格
orderStyle	double	委托风格 取值参考 OrderStyle
volume	double	委托量
value	double	委托额
percent	double	委托百分比
targetVolume	double	委托目标量
targetValue	double	委托目标额
targetPercent	double	委托目标百分比
filledVolume	double	已成量
filledVwap	double	已成均价
filledAmount	double	已成金额
filledCommission	double	已成手续费
createdAt	char	委托创建时间
updatedAt	char	委托更新时间

回报数据结构说明

ExecRpt - 回报返回值字段

属性	类型	说明
strategyId	char	策略ID
accountId	char	账号ID
accountName	char	账户登录名
clOrdId	char	委托客户端ID，下单生成，固定不变，这个才是策略识别的主键
orderId	char	委托柜台ID（系统字段）
ex_ord_id	char	委托交易所ID
positionEffect	double	开平标志 取值参考 PositionEffect
side	double	买卖方向 取值参考 OrderSide
ordRejReason	double	委托拒绝原因 取值参考 OrderRejectReason
ordRejReasonDetail	char	委托拒绝原因描述
execType	double	执行回报类型 取值参考 ExecType
price	double	委托成交价格
volume	double	委托成交量

amount	double	委托成交金额
createdAt	char	回报创建时间

持仓数据结构说明

Position - 持仓返回值字段

属性	类型	说明
accountId	char	账号ID
accountName	char	账户登录名
symbol	char	标的代码
side	double	持仓方向 取值参考 PositionSide
volume	double	总持仓量; 昨持仓量 (volume - volume_today)
volumeToday	double	今日持仓量
vwap	double	持仓均价 $new_vwap = ((position.vwap \ position.volume) + (trade.volumetrade.price)) / (position.volume + trade.volume)$
amount	double	持仓额 (volumevwapmultiplier)
price	double	当前行情价格 (回测时值为0)
fpn1	double	持仓浮动盈亏 ((price - vwap) volume multiplier) (回测模式fpn1只有仓位变化时或者一天更新一次, 仿真模式3s更新一次)
cost	double	持仓成本 (vwap volume multiplier * margin_ratio)
orderFrozen	double	挂单冻结仓位
orderFrozenToday	double	挂单冻结今仓仓位
available	double	非挂单冻结总仓位 (volume - order_frozen); 可平昨仓位 (available - available_today)
availableToday	double	非挂单冻结今仓位 (volume_today - order_frozen_today)(仅期货)
lastPrice	double	上一次成交价 (回测时值为0)
lastVolume	double	上一次成交量 (回测时值为0)
lastInout	double	上一次出入持仓量 (回测时值为0)
changeReason	double	仓位变更原因, 取值参考 CashPositionChangeReason
changeEventId	char	触发资金变更事件的ID
hasDividend	double	持仓区间有分红配送
createdAt	char	建仓时间
updatedAt	char	仓位变更时间

资金数据结构说明

Cash - 资金返回值字段

属性	类型	说明
accountId	char	账号ID
accountName	char	账户登录名

currency	double	币种
nav	double	净值, 总权益
pn1	double	净收益
fpn1	double	浮动盈亏
frozen	double	持仓占用资金
orderFrozen	double	挂单冻结资金
available	double	可用资金
cumInout	double	累计出入金
cumTrade	double	累计交易额
cumPn1	double	累计平仓收益(没扣除手续费)
cumCommission	double	累计手续费
lastTrade	double	上一次交易额
lastPn1	double	上一次收益
lastCommission	double	上一次手续费
lastInout	double	上一次出入金
changeReason	double	资金变更原因 取值参考 CashPositionChangeReason
changeEventId	char	触发资金变更事件的ID
createdAt	char	资金初始时间
updatedAt	char	资金变更时间

order_volume - 按指定量委托

函数原型：

```
1. [ order ] = order_volume( symbol, volume, side, order_type, position_effect, price, account )
```

参数说明

参数	类型	说明
symbols	cell	下单标的
volume	double	委托量，单位为：股票为股，期货为手
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID
返回值	cell	order格式的订单信息

示例

```
1. [order]=order_volume('SZSE.000001', 100, OrderSide.OrderSide_Buy,
```

```
OrderType.OrderType_Market, PositionEffect.PositionEffect_Open, 0)
```

返回值

```
1. order =
2.
3.     Columns 1 through 9
4.
5.     'strategyId'    'accountId'    'accountName'    'clOrdId'    'orderId'
6.     'exOrdId'    'symbol'    'side'    'positionEffect'
7.     [1x36 char]    [1x36 char]    ''    '000000001'    ''    ''
8.     'SZSE.000001'    [ 1]    [ 1]
9.
10.    Columns 10 through 17
11.
12.    'positionSide'    'orderType'    'orderDuration'    'orderQualifier'    'orderSrc'
13.    'status'    'ordRejReason'    'ordRejReasonDetail'
14.    [ 1]    [ 2]    [ 0]    [ 0]    [ 0]
15.    [ 3]    [ 0]    ''
16.
17.    Columns 18 through 26
18.
19.    'price'    'stopPrice'    'orderStyle'    'volume'    'value'    'percent'
20.    'targetVolume'    'targetValue'    'targetPercent'
21.    [8.9800]    [ 0]    [ 1]    [ 100]    [898.0000]    [8.9800e-04]
22.    [ 200]    [ 1.7960e+03]    [ 0.0018]
23.
24.    Columns 27 through 31
25.
26.    'filledVolume'    'filledVwap'    'filledAmount'    'filledCommission'
27.    'updatedAt'
28.    [ 100]    [ 8.9800]    [ 898.0000]    [ 0.8980]    '0001-01-
29.    01 00:00:00'
```

order_value - 按指定价值委托

函数原型：

```
1. [ order ] = order_value ( symbol, value, side, order_type, position_effect, price,
                             account )
```

参数说明

参数	类型	说明
symbols	cell	下单标的
value	double	委托资金量
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID

返回值	cell	order格式的订单信息
-----	------	--------------

示例

```
1. [order]=order_value('SZSE.000001', 10000, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Market,PositionEffect.PositionEffect_Open,0)
```

order_percent - 按总资产指定比例委托

函数原型：

```
1. [ order ] = order_percent( symbol, percent, side, order_type,position_effect,price,
    account)
```

参数说明

参数	类型	说明
symbols	cell	下单标的
percent	double	委托占总资金的百分比
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID
返回值	cell	order格式的订单信息

示例

```
1. [order]=order_percent('SZSE.000001', 0.1, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Market,PositionEffect.PositionEffect_Open,0);
```

order_target_volume - 调仓到目标持仓量

函数原型：

```
1. [ order ] = order_target_volume( symbol, volume, side, order_type,price, account)
```

参数说明

参数	类型	说明
symbols	cell	下单标的
volume	double	目标持仓量，单位为：股票为股，期货为手
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型

account	char	仿真和模拟可不填，实盘多账户填入账户的ID
返回值	cell	order格式的订单信息

示例

```
1. [order]=order_target_volume('SZSE.000001', 10000, OrderSide.OrderSide_Buy, OrderType.OrderType_Market,0)
```

order_target_value - 调仓到目标持仓额

函数原型：

```
1. [ order] =order_target_value( symbol, value, side, order_type,price, account)
```

参数说明

参数	类型	说明
symbols	cell	下单标的
value	double	目标持仓价值
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID
返回值	cell	order格式的订单信息

示例

```
1. [order]=order_target_value('SZSE.000001', 10000, OrderSide.OrderSide_Buy, OrderType.OrderType_Market,0)
```

order_target_percent - 调仓到目标持仓比例（总资产的比例）

函数原型：

```
1. [ order] = order_target_percent( symbol, percent, side, order_type, price, account)
```

参数说明

参数	类型	说明
symbols	cell	下单标的
percent	double	目标总资产管比例
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型

position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID

示例

```
1. [order]=order_target_percent('SZSE.000001', 0.5, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Market,0)
```

order_batch - 批量委托

相比循环下单，批量委托在下单流程上一次完成，延迟更低

函数原型：

```
1. [ orders ] = order_batch( order_cell)
```

参数说明

order_cell 是由以下参数组成的委托矩阵，

参数	类型	说明
symbols	cell	下单标的
percent	double	目标总资产管比例
side	double	OrderSide 交易方向枚举类型
order_type	double	OrderType 交易类型枚举类型
position_effect	double	PositionEffect 开平仓枚举类型
account	char	仿真和模拟可不填，实盘多账户填入账户的ID

示例

```
1. order_cell = {'SHSE.600000', 100, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Limit,PositionEffect.PositionEffect_Open, 11;
2.     'SHSE.600004', 100, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Limit,PositionEffect.PositionEffect_Open, 18};
3. [ orders ] = order_batch( order_cell);
```

order_cancel - 撤销指定委托

函数原型

```
1. [ reg ] = order_cancel( clOrdIds, account)
```

说明：

clOrdIds字段数据由订单数据中的clOrdId获取，返回结果为撤单成功或者失败

order_cancel_all - 撤销所有委托

函数原型:

```
1. order_cancel_all()
```

order_close_all - 平当前所有可平持仓

函数原型:

```
1. order_close_all()
```

get_unfinished_orders - 查询日内全部未结委托

函数原型:

```
1. get_unfinished_orders()
```

说明:

返回cell格式的订单数据表

get_orders - 查询日内全部委托

函数原型:

```
1. get_orders()
```

说明:

返回cell格式的订单数据列表

get_execution_reports - 查询日内全部执行回报

函数原型:

```
1. get_execution_reports()
```

示例

```
1. [ excerpt ] = get_execution_reports( )
```

返回值

```
1. excerpt =  
2.   
3. Columns 1 through 9  
4.   
5. 'strategyId' 'accountId' 'accountName' 'clOrdId' 'orderId'  
   'execId' 'symbol' 'positionEffect' 'side'
```

```

6.      [1x36 char]      ''      ''      '000000341'      ''      ''
'SZSE.000001'      [      1]      [  1]
7.      [1x36 char]      ''      ''      '000000340'      ''      ''
'SZSE.000001'      [      1]      [  1]
8.
9.      Columns 10 through 17
10.
11.      'ordRejReason'      'ordRejReasonDetail'      'execType'      'price'      'volume'
'amount'      'commission'      'cost'
12.      [      0]      ''      [      15]      [8.9800]      [ 1100]
[9.8780e+03]      [      0]      [  0]
13.      [      0]      ''      [      15]      [8.9800]      [ 1100]
[9.8780e+03]      [      0]      [  0]
14.
15.      Column 18
16.
17.      'createdAt'
18.      '0001-01-01 00:00:00'
19.      '0001-01-01 00:00:00'

```

get_position-持仓查询

```
1. [ position ] = get_position( account )
```

示例：

```
1. [ position ] = get_position( )
```

返回值

```

1. position =
2.
3.      Columns 1 through 9
4.
5.      'accountId'      'accountName'      'symbol'      'side'      'volume'
'volumeToday'      'vwap'      'amount'      'price'
6.      [1x36 char]      ''      'SZSE.000001'      [  1]      [ 2200]      [
2200]      [8.9800]      [1.9756e+04]      [  0]
7.
8.      Columns 10 through 17
9.
10.      'fpn1'      'cost'      'orderFrozen'      'orderFrozenToday'      'available'
'available_today'      'lastPrice'      'lastVolume'
11.      [242.0013]      [1.9756e+04]      [      0]      [      0]      [ 2200]
[      2200]      [      0]      [      0]
12.
13.      Columns 18 through 22
14.
15.      'lastInout'      'changeReason'      'hasDividend'      'createdAt'
'updatedAt'
16.      [      0]      [      0]      [      0]      '2018-08-06 09:45:00'      '2018-
08-06 09:45:00'

```

get_cash—资金查询

```
1. [ cash ] = get_cash( account )
```

示例

```
1. [ cash ] = get_cash( )
```

返回值

```
1. position =
2.
3. Columns 1 through 8
4.
5. 'accountId' 'accountName' 'currency' 'nav' 'pnl' 'fpnl'
   'frozen' 'orderFrozen'
6. [1x36 char] '' [ 0] [9.9989e+05] [-107.7559]
   [-87.9999] [1.9756e+04] [ 0]
7.
8. Columns 9 through 16
9.
10. 'available' 'balance' 'cumInout' 'cumTrade' 'cumPnl'
    'cumCommission' 'lastTrade' 'lastPnl'
11. [9.8022e+05] [ 0] [ 1000000] [1.9756e+04] [ 0] [
    19.7560] [9.8780e+03] [ 0]
12.
13. Columns 17 through 22
14.
15. 'lastCommission' 'lastInout' 'changeReason' 'changeEventId' 'createdAt'
    'updatedAt'
16. [ 9.8780] [ 0] [ 0] '' '2018-08-05
    10:40:00' '2018-08-06 09:45:00'
```

枚举常量

- [OrderStatus](#) - 委托状态
- [OrderSide](#) - 委托方向
- [OrderType](#) - 委托类型
- [OrderDuration](#) - 委托时间属性
- [OrderQualifier](#) - 委托成交属性
- [ExecType](#) - 执行回报类型
- [PositionEffect](#) - 开平仓类型
- [PositionSide](#) - 持仓方向
- [OrderRejectReason](#) - 订单拒绝原因
- [CancelOrderRejectReason](#) - 取消订单拒绝原因
- [OrderStyle](#) - 订单类型
- [CashPositionChangeReason](#) - 仓位变更原因
- [SecType](#) - 标的类别
- [AccountStatus](#) - 交易账户状态

OrderStatus - 委托状态

```

1. OrderStatus_Unknown = 0
2. OrderStatus_New = 1           % 已报
3. OrderStatus_PartiallyFilled = 2       % 部成
4. OrderStatus_Filled = 3           % 已成
5. OrderStatus_Canceled = 5         % 已撤
6. OrderStatus_PendingCancel = 6       % 待撤
7. OrderStatus_Rejected = 8         % 已拒绝
8. OrderStatus_Suspended = 9        % 挂起
9. OrderStatus_PendingNew = 10       % 待报
10. OrderStatus_Expired = 12        % 已过期

```

OrderSide - 委托方向

```

1. OrderSide_Unknown = 0
2. OrderSide_Buy = 1           % 买入
3. OrderSide_Sell = 2         % 卖出

```

OrderType - 委托类型

```

1. OrderType_Unknown = 0
2. OrderType_Limit = 1         % 限价委托
3. OrderType_Market = 2       % 市价委托
4. OrderType_Stop = 3         % 止损止盈委托

```

OrderDuration - 委托时间属性

仅在实盘模式生效，具体执行模式请参考交易所给出的定义

1. OrderDuration_Unknown	= 0	
2. OrderDuration_FAK	= 1	% 即时成交剩余撤销(fill and kill)
3. OrderDuration_FOK	= 2	% 即时全额成交或撤销(fill or kill)
4. OrderDuration_GFD	= 3	% 当日有效(good for day)
5. OrderDuration_GFS	= 4	% 本节有效(good for section)
6. OrderDuration_GTD	= 5	% 指定日期前有效(goodtilldate)
7. OrderDuration_GTC	= 6	% 撤销前有效(goodtillcancel)
8. OrderDuration_GFA	= 7	% 集合竞价前有效(good for auction)

OrderQualifier - 委托成交属性

仅在实盘模式生效，具体执行模式请参考交易所给出的定义

1. OrderQualifier_Unknown	= 0	
2. OrderQualifier_BOC	= 1	% 对方最优价格(best of counterparty)
3. OrderQualifier_BOP	= 2	% 己方最优价格(best of party)
4. OrderQualifier_B5TC	= 3	% 最优五档剩余撤销(best 5 then cancel)
5. OrderQualifier_B5TL	= 4	% 最优五档剩余转限价(best 5 then limit)

ExecType - 执行回报类型

1. ExecType_Unknown	= 0	
2. ExecType_New	= 1	% 已报
3. ExecType_Canceled	= 5	% 已撤销
4. ExecType_PendingCancel	= 6	% 待撤销
5. ExecType_Rejected	= 8	% 已拒绝
6. ExecType_Suspended	= 9	% 挂起
7. ExecType_PendingNew	= 10	% 待报
8. ExecType_Expired	= 12	% 过期
9. ExecType_Trade	= 15	% 成交
10. ExecType_OrderStatus	= 18	% 委托状态
11. ExecType_CancelRejected	= 19	% 撤单被拒绝

PositionEffect - 开平仓类型

1. PositionEffect_Unknown	= 0	
2. PositionEffect_Open	= 1	% 开仓
3. PositionEffect_Close	= 2	% 平仓，具体语义取决于对应的交易所
4. PositionEffect_CloseToday	= 3	% 平今仓
5. PositionEffect_CloseYesterday	= 4	% 平昨仓

PositionSide - 持仓方向

1. PositionSide_Unknown	= 0	
2. PositionSide_Long	= 1	% 多方向
3. PositionSide_Short	= 2	% 空方向

OrderRejectReason - 订单拒绝原因

1. OrderRejectReason_Unknown = 0	% 未知原因
2. OrderRejectReason_RiskRuleCheckFailed = 1	% 不符合风控规则
3. OrderRejectReason_NoEnoughCash = 2	% 资金不足
4. OrderRejectReason_NoEnoughPosition = 3	% 仓位不足
5. OrderRejectReason_IllegalAccountId = 4	% 非法账户ID
6. OrderRejectReason_IllegalStrategyId = 5	% 非法策略ID
7. OrderRejectReason_IllegalSymbol = 6	% 非法交易标的
8. OrderRejectReason_IllegalVolume = 7	% 非法委托量
9. OrderRejectReason_IllegalPrice = 8	% 非法委托价
10. OrderRejectReason_AccountDisabled = 10	% 交易账号被禁止交易
11. OrderRejectReason_AccountDisconnected = 11	% 交易账号未连接
12. OrderRejectReason_AccountLoggedout = 12	% 交易账号未登录
13. OrderRejectReason_NotInTradingSession = 13	% 非交易时段
14. OrderRejectReason_OrderTypeNotSupported = 14	% 委托类型不支持
15. OrderRejectReason_Throttle = 15	% 流控限制

CancelOrderRejectReason - 取消订单拒绝原因

1. CancelOrderRejectReason_OrderFinalized = 101	% 委托已完成
2. CancelOrderRejectReason_UnknownOrder = 102	% 未知委托
3. CancelOrderRejectReason_BrokerOption = 103	% 柜台设置
4. CancelOrderRejectReason_AlreadyInPendingCancel = 104	% 委托撤销中

OrderStyle - 订单类型

1. OrderStyle_Unknown = 0	
2. OrderStyle_Volume = 1	% 按指定量委托
3. OrderStyle_Value = 2	% 按指定价值委托
4. OrderStyle_Percent = 3	% 按指定比例委托
5. OrderStyle_TargetVolume = 4	% 调仓到目标持仓量
6. OrderStyle_TargetValue = 5	% 调仓到目标持仓额
7. OrderStyle_TargetPercent = 6	% 调仓到目标持仓比例

CashPositionChangeReason - 仓位变更原因

1. CashPositionChangeReason_Unknown = 0	
2. CashPositionChangeReason_Trade = 1	% 交易
3. CashPositionChangeReason_Inout = 2	% 出入金 / 出入持仓

SecType - 标的类别

1. SEC_TYPE_STOCK = 1	% 股票
2. SEC_TYPE_FUND = 2	% 基金

3.	SEC_TYPE_INDEX = 3	% 指数
4.	SEC_TYPE_FUTURE = 4	% 期货
5.	SEC_TYPE_OPTION = 5	% 期权
6.	SEC_TYPE_CONFUTURE = 10	% 虚拟合约

AccountStatus - 交易账户状态

1.	State_UNKNOWN = 0	#未知
2.	State_CONNECTING = 1	#连接中
3.	State_CONNECTED = 2	#已连接
4.	State_LOGGEDIN = 3	#已登录
5.	State_DISCONNECTING = 4	#断开中
6.	State_DISCONNECTED = 5	#已断开
7.	State_ERROR = 6	#错误

错误码

错误码	描述
0	成功
1000	错误或无效的token
1001	无法连接到终端服务
1010	无法获取掘金服务器地址列表
1011	消息包解析错误
1012	网络消息包解析错误
1013	交易服务调用错误
1014	历史行情服务调用错误
1015	策略服务调用错误
1016	动态参数调用错误
1017	基本面数据服务调用错误
1018	回测服务调用错误
1019	交易网关服务调用错误
1020	无效的ACCOUNT_ID
1021	非法日期格式
1100	交易消息服务连接失败
1101	交易消息服务断开
1200	实时行情服务连接失败
1201	实时行情服务连接断开
1300	初始化回测失败，可能是终端未启动或无法连接到终端
1301	回测时间区间错误
1302	回测读取缓存数据错误
1303	回测写入缓存数据错误