

QMT_Python_API_Doc

文档版本: 3.3.6 Python 版本: 3.6.8 更新日期: 2021.12.20

更新说明

序号	更新时间	新增项
32	2021.12.20	修正示例中，“CSRC采矿业”为“CSRC1采矿业”
31	2021.10.27	删除函数ContextInfo.get_ext_all_data
30	2021.09.28	增加函数说明ContextInfo.get_option_list, ContextInfo.get_option_iv, ContextInfo.bsm_price, ContextInfo.bsm_iv, get_hkt_exchange_rate, get_enable_short_contract, get_unclosed_compacts, get_closed_compacts, get_option_subject_position, get_comb_option, make_option_combination, release_option_combination, credit_account_callback, query_credit_opvolume, credit_opvolume_callback, get_raw_financial_data, get_his_contract_list, get_option_udl_data, get_north_finance_change, get_hkt_statistics, get_hkt_details, get_market_time
29	2021.08.19	新增get_market_data_ex
28	2021.05.12	新增快速上手说明
27	2021.03.18	修改smart_algo_passorder, passorder(py)
26	2021.03.18	增加st_status, ContextInfo.get_his_st_data, ContextInfo.get_his_index_data, down_history_data, ContextInfo.subscribe_quote, ContextInfo.unsubscribe_quote, ContextInfo.get_all_subscription,
25	2021.03.18	增加algo_passorder(py), get_basket, set_basket, 修改get_market_data中period和dividend_type默认值为follow
24	2020.06.18	增加pause_task和resume_task
23	2020.06.18	增加enum_EOperationType下单操作类型/主要交易类型
22	2020.06.18	查询智能算法任务相关：smart_algo_passorder增加参数；order（委托对象）中增加m_nTaskId（任务号）；新增CTaskDetail任务对象
21	2020.05.21	ContextInfo.get_contract_expire_date获取期货合约到期日
20	2020.05.21	ContextInfo.get_instrumentdetail获取合约详细信息
19	2020.05.21	stop停止处理函数
18	2020.05.21	get_etf_iopv获取ETF的基金份额参考净值
17	2020.05.21	get_etf_info获取ETF申赎清单数据
16	2020.05.21	ContextInfo.get_turnover_rate获取换手率数据
15	2020.05.21	ContextInfo.get_local_data从本地获取行情数据
14	2020.05.21	ContextInfo.load_stk_vol_list获取一篮子证券编码及数量数据
13	2020.05.21	ContextInfo.load_stk_list获取一篮子证券编码数据

序号	更新时间	新增项
12	2020.05.21	ContextInfo.get_ext_all_data引用扩展数据在指定时间区域内的所有值和排名
11	2020.05.21	call_formula调用VBA组合模型
10	2020.05.20	passorder函数补充
9	2020.05.20	get_ipo_data获取当日新股新债信息
8	2020.05.20	get_new_purchase_limit获取账户新股申购额度
7	2020.05.20	stoploss_marketprice以市价单方式止损/止盈
6	2020.05.20	stoploss_limitprice以限价单方式止损/止盈
5	2020.03.03	run_time(funcName,period,startTime,market)
4	2020.03.03	ContextInfo.get_financial_data增加公告时间
3	2020.03.03	新增ContextInfo.get_option_detail_data获取指定期权品种的详细信息
2	2020.03.03	get_trade_detail_data中增加获取任务功能
1	2020.01.16	cancel_task(taskId,accountId,accountType,ContextInfo)

快速上手

重要概念

python下单流程简单描述：python产生交易信号->客户端->柜台->柜台(交易反馈)->客户端->python主推函数(order_callback等)

有效信号和无效信号

有效信号是指在最新bar对应的handlebar里调用交易函数产生的信号，此时python是否立刻产生委托由下文快速交易规则判断

无效下单是指在非最新bar对应的handlebar里调用交易函数产生的信号，此时下单委托会被python忽略。个别情况例外，详见下文快速交易规则

快速交易和非快速交易

快速交易是指当根bar内产生交易信号后，python立即把委托发送至客户端，此时客户端根据模拟信号规则判断是否将该信号发送至柜台

非快速交易是指当根bar内产生交易信号后，在下一个bar的第一个分笔到来时再把委托发送至客户端

支持快速交易的函数

1、cancel, passorder、algo_passorder或smart_algo_passorder当quickTrade参数设置为1 或2时

2、make_option_combination 和release_option_combination(期权组合持仓的构建和解除)

另外，make_option_combination 和release_option_combination 和passorder、algo_passorder或smart_algo_passorder当quickTrade参数设置为2时 python会立刻产生交易信号，即在任何位置产生的信号都是有效信号

模拟信号和实盘信号

模拟信号是指在模型交易里以模拟形式运行策略后，产生的委托为模拟信号，此时委托不会产生任务，也不会发送至柜台，即python产生的委托会在客户端节点被截断。python发出的所有交易信号（下单、撤单等）由此规则约束

模拟信号的例子



实盘信号是指在模型交易里以实盘形式运行策略后，产生的委托为实盘信号，此时会发送至柜台，也会产生任务，即生成了一条真实委托

实盘信号的例子



概述

感谢您使用迅投 QMT 极速策略交易系统，以下简称 QMT 系统，本文档主要介绍 QMT 系统 Python API 的使用方法。内容较多，可使用 Ctrl + F 进行关键字搜索。

QMT 系统与其他普通股票软件最大的不同点就是其提供历史数据下载、模型编辑、模型回测、模型交易、算法交易及交易风控等完整量化交易功能。通过 QMT 系统，用户可以快速的将自己的转化为计算机代码，形成自己的交易策略，让计算机帮助用户实现策略的回测检验并实现无人值守自动化交易。

在量化策略编写语言的支持上，QMT 系统目前支持两种：VBA 和 Python。在后续的规划中，QMT 系统将全面支持市面上主流的量化编程语言。

随着量化在国内的深入发展以及大量专业编程人员进入量化这个领域，市场上的量化投资者对 Python 的需求越来越大，QMT 系统的 Python API 就是为了满足这部分投资者而量身打造。QMT 系统的 Python API 既可以高效使用 QMT 系统底层的数据接口及交易接口，也可以方便地引入 Python 支持的第三方库，极大地便利了量化投资者的模型策略需求。

1. 创建策略

1.1. 策略示例

1.1.1. 一个简单的 Python 策略

The screenshot shows the QMT Strategy Editor interface. On the left is a tree view of available models and indicators. The main area contains a code editor with the following Python code:

```
#encoding:gbk
def init(ContextInfo):
    print 'hello init'

def handlebar(ContextInfo):
    print 'hello handlebar'
```

The code editor highlights the `init` and `handlebar` methods. To the right of the code editor is a configuration panel with settings like name, shortcut, and period. Below the code editor is a "运行输出" (Run Output) window showing the console logs:

```
【2018-11-26 13:20:31.524】 start trading mode
【2018-11-26 13:20:31.524】 hello init
hello handlebar
【2018-11-26 13:20:31.524】 hello handlebar
【2018-11-26 13:20:31.524】 hello handlebar
【2018-11-26 13:20:31.524】 hello handlebar
```

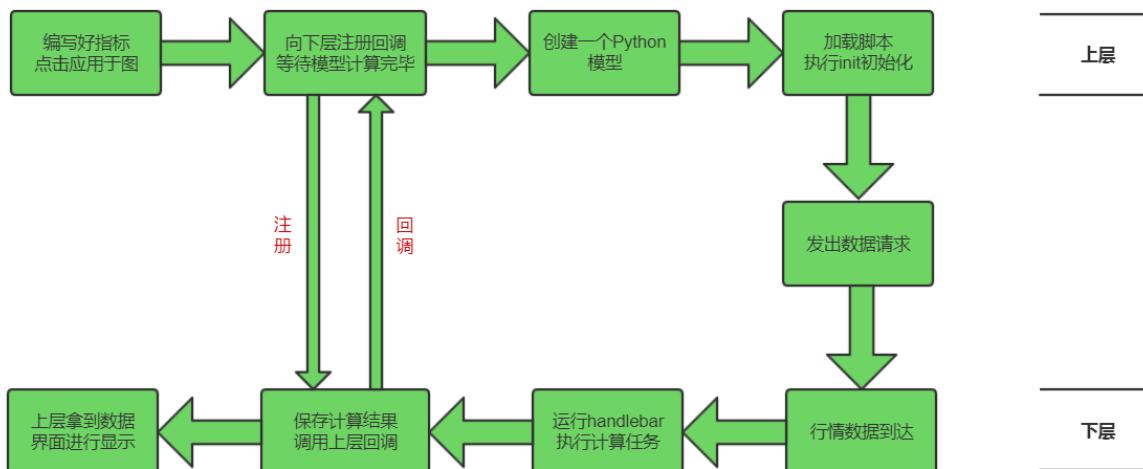
At the bottom, there are tabs for "编译输出" (Compile Output), "日志输出" (Log Output), and a message indicating the compilation was successful.

简单的Python策略

上图展示了如何在 QMT 系统上用 Python 写一个输出 hello world 的模型。图中的模型实现了一个 Python 模型 **必须实现** 的两个接口：** `init()`** 与 `handlebar()`。输出窗口展示了这个模型的运行输出。系统首先调用 `init()` 方法输出了 `hello init`，随后在每根 K 线上调用一次 `handlebar()`，输出一次 `hello handlebar`。

1.2. 运行机制

1.2.1. Python 运行机制



Python 模型运行流程图

上图为 QMT 系统 Python 模型运行流程图。当用户在 Python 平台编写好自己的策略后，可以点击【运行】按钮来运行脚本。界面层在向下层请求数据的过程中，会首先创建一个 Python 模型，这个模型关联当前界面运行的产品。模型创建完毕后，加载用户编写的 Python 脚本，并调用 init() 初始化函数。模型运行所需的数据也在此时向下层发出请求。下层在接收到请求后，会向服务器请求行情数据，并组织好数据格式。Python 模型根据返回的数据，运行 handlebar()，用户可以用 paint() 方法将希望显示的计算结果输出到界面上，界面会将输出结果展示出来。

1.2.2. 重要概念

(1) Bar的概念

我们把单根 K 线称之为 Bar，每根 Bar 由 tick（分笔）组成。



分钟 Bar 示例

QMT 系统的模型是根据行情驱动，逐 K 线运行，每根 K 线调用一次 Python 模型中的 handlebar(ContextInfo) 函数。

根据选择的运行周期不同，handlebar(ContextInfo) 函数的运行次数也不同。如选择在日线上运行策略，则 handlebar(ContextInfo) 函数每天被调用一次（盘中虽会每个 tick 调用一次，但只有最后一个 tick 才会判定交易函数是否被调用）。

(2) Init

init是一个Python模型的初始化方法。在模型加载的时候，系统会调用init方法，做一些必要的初始化，比如初始化股票池、初始化资金账号、初始化全局变量等。如果用户的模型无需做初始化，可以在方法体中写pass，但方法的定义必须存在，否则模型的运行会报错。

```

def init(ContextInfo):
    # 设定股票池方法一，取板块的成分股作为股票池，例如取“上证50”成分股,
    ContextInfo.trade_code_list = ContextInfo.get_stock_list_in_sector("上证50")
    # 设定股票池方法二，可以自定义将股票代码写到列表里
    ContextInfo.trade_code_list = ['600000.SH', '300462.SZ', '000697.SZ', '300008.SZ']
    ContextInfo.set_universe(ContextInfo.trade_code_list) #初始化设定股票池
    ContextInfo.accID = '6000000009' # 初始化设定资金账号
    ContextInfo.order = 10 # 初始化设置一些全局变量，例如设定买入的手数
    
```

Init 初始化

(3) Handlebar

handlebar 是整个 Python 模型中的核心执行函数。当模型从数据层获取到运行所需要的数据之后，会对数据集上的每一根 bar，调用一次 handlebar 函数，处理当前这根 bar 上的数据。也就是说，用户模型的核心逻辑都是写在该函数中的，如获取数据，设置下单条件等。在 handlebar 中处理完数据后，用户可以通过 paint 方法将需要绘图输出的结果返回给界面。界面会将输出结果如实的展示出来。

(4) ContextInfo

ContextInfo 是整个 Python 框架中的一个核心对象。它包含了各种与 Python 底层框架交互的 API 方法，也是一个全局的上下文环境，可以在 init 以及 handlebar 这两个函数中自由地传递用户创建的各种自定义数据。

1.2.3. Python 策略运行机制

用户在界面上提交请求后，在最终看到结果前会经历两步：

- (1) 创建模型，初始化环境，发送数据请求；
- (2) 数据到达后，调用 Python 的 init 函数进行 Python 初始化，然后运行 handlebar 方法；

ContextInfo 是 Python 模型中的全局对象，其中封装了 benchmark，universe 等变量，也封装了 get_history_data 等重要函数，是 init 与 handlebar 之间，以及各个 handlebar 之间进行信息传递的重要载体。用户也可以在其中封装自己想要定义的全局变量或函数，但注意不要与原有的重名。

init 和 handlebar 是 Python 模型中最重要的方法，也是唯二由 C++ 直接调用的方法，所有的执行代码都尽量写在这两个方法中或由其中的函数调用。

init 是 Python 的初始化方法，负责初始化模型运行所需的初始变量，如对于基准 benchmark 和股票池 universe 的初始化。

handlebar 是 Python 的核心执行函数。在 K 线图上运行时会根据主图的时间轴，每个时间点会进入相应的 handlebar 方法，可在 handlebar 中使用 ContextInfo.barpos 来获取当前的 bar 索引位置。

```
def handlebar(ContextInfo):  
    if ContextInfo.is_last_bar():#判断是否为最后一根K线，跳过历史K线，用最新的K线运行策略  
        d = ContextInfo.barpos#返回当前运行到的K线索引号  
        t = ContextInfo.get_bar_timetag(d) #获取当前K线对应的时间的时间戳  
        date = timetag_to_datetime(t, "%Y%m%d") #将时间戳转化成日期的格式  
        print d,t,date#可以通过python的print方法，将所需查看的内容打印到日志输出
```

获取当前 Bar 位置

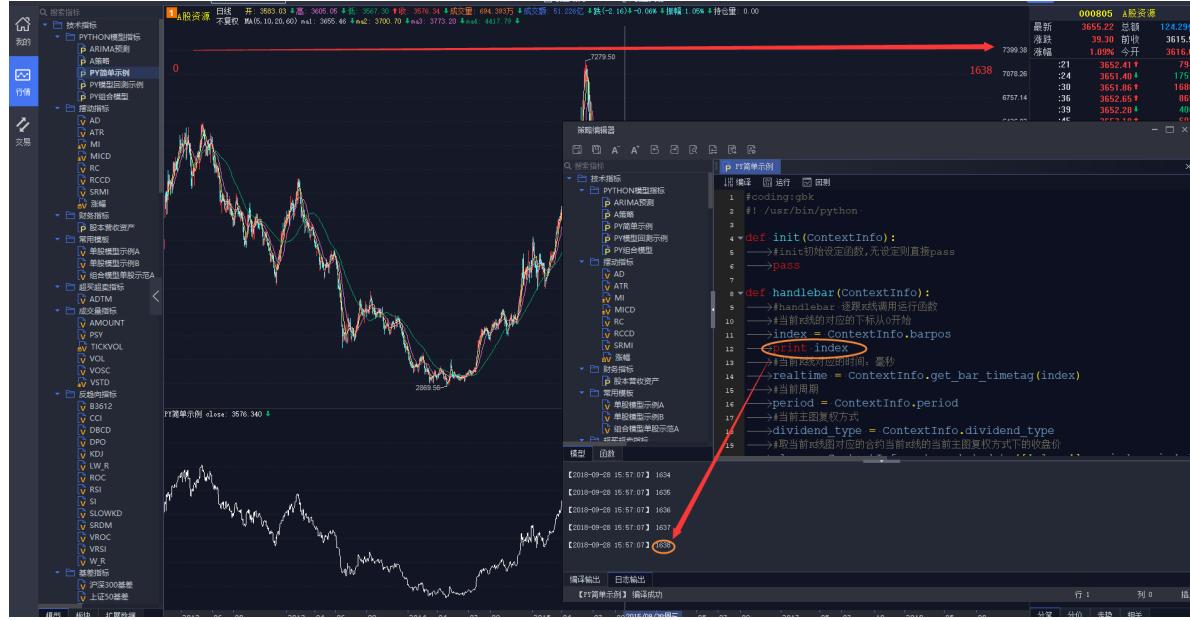
*注：

索引：索引的作用相当于图书的目录，可以根据目录中的页码（索引号）快速找到所需的内容。

时间戳：时间戳是指格林威治时间 1970 年 01 月 01 日 00 时 00 分 00 秒（北京时间 1970 年 01 月 01 日 08 时 00 分 00 秒）起至现在的总秒数。通俗的讲，时间戳是一份能够表示一份数据在一个特定时间点已经存在的完整的可验证的数据。

1.2.4. Python 交易函数运行机制

QMT 系统模型是根据行情驱动，逐 K 线运行的。即点击运行模型时，模型是从第 0 根 K 线开始运行到最后一根 K 线（如想加快模型运行速度，可以策略编辑器 - 基本信息中设置快速计算，限制计算范围，只计算最新的指定数量的 K 线范围），每根 K 线调用一次 Python 模型中的 handlebar(ContextInfo) 函数。



逐 K 线运行：从第 0 根 K 线一直调用 handlebar(ContextInfo) 运行到最后一根

在盘中，最后一根 K 线每变动一次，handlebar(ContextInfo) 函数被执行一次。这根 K 线的最后一个 tick 判定模型信号是否成立，如果模型信号成立，交易函数被调用，则在下一根 K 线的第一个 tick 发出下单信号，生成下单任务。



每个 tick 数据来时，最后一根 K 线会随着变动

```

26 → if close > open:
27 → →→→→ passorder(23, 1101, '8008882888', '510050.SH', 5, 0, 100, ContextInfo)

```

一个模型判定：当收盘价 > 开盘价时产生模型信号触发下单

如上图，当盘中运行到最后一根 K 线的时候，每个 tick 数据来时都会判定一下这个条件是否成立，当不是这根 K 线的最后一个 tick，之前的所有的 tick 成立的产生的信号就是虚的信号。只有当这个 K 线确定时，产生的信号才是有效信号，才会触发下单。

模型运行在日 K 线周期及日 K 线以上周期时，因为是在下一根 K 线的第一个 tick（最新行情 tick）发出下单信号，而下一个 K 线就是第二日了。所以模型运行当日无法下单，除非：

- (1) 设置 passorder 函数中的 quickTrade 参数为 1，立即下单；
- (2) 使用 do_order(ContextInfo) 函数。

quickTrade 参数设置为 1 可实现最后一根 K 线没有走完生成的模型信号也发出下单信号。

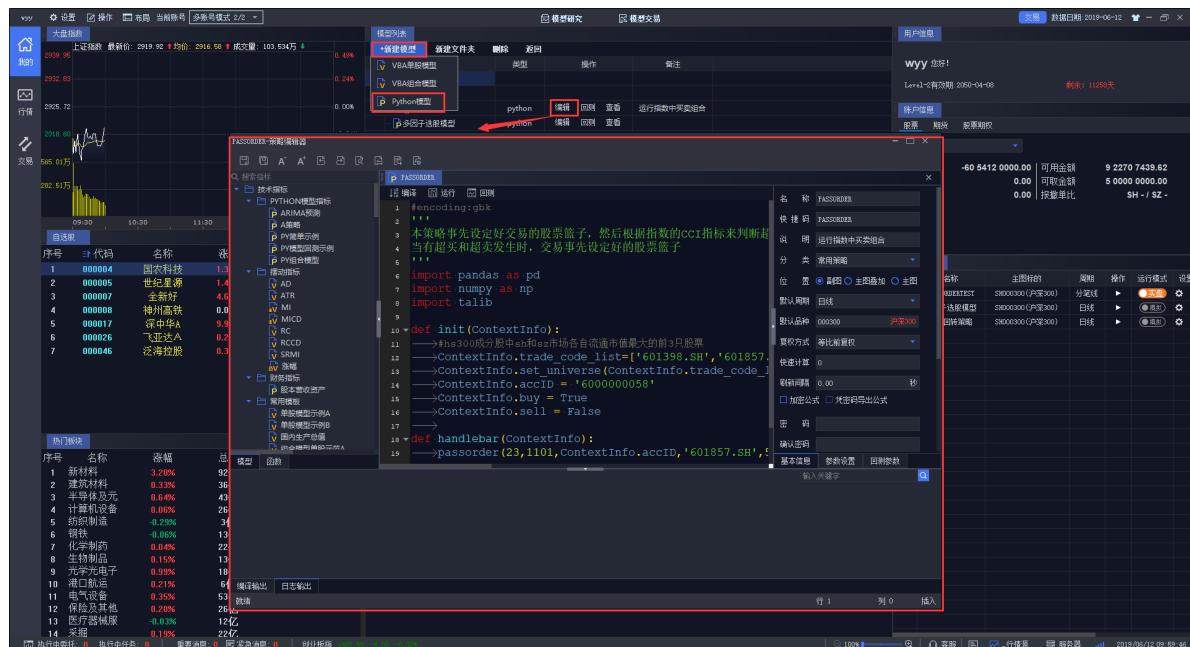
do_order(ContextInfo) 函数被调用后会把上一根 K 线生成的模型信号立刻发出，且只发一次，解决交易函数必须是在下一根 K 线的第一个 tick 数据时发信号的问题。

2. 创建一个 Python 策略

2.1. 新建一个 Python 策略

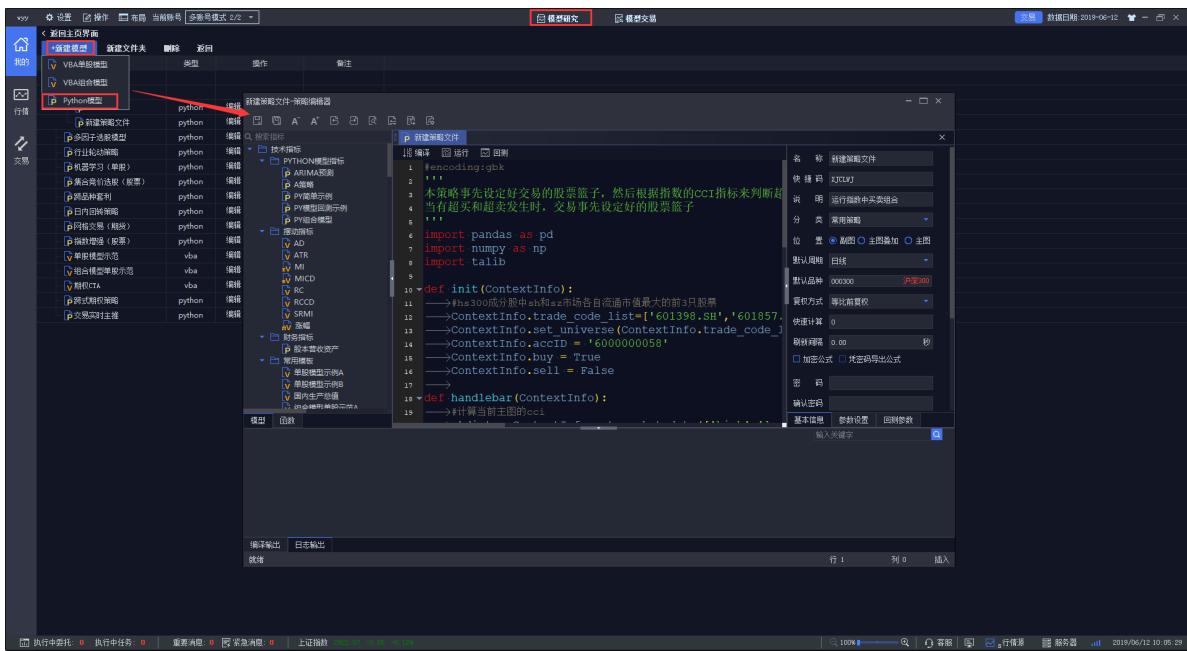
模型创建方法有三种：

方法一，在【模型研究】界面，使用系统预置的各种示例模型，点击后方“编辑”按钮，并在弹出的【策略编辑器】中以此示例模型代码为基础进行编写。



我的主页-编辑模型

方法二，在【模型研究】界面，点击新建模型，选择 Python 模型，在弹出的【策略编辑器】中从头到尾编写一个用户自己的量化模型。



模型研究 新建模型

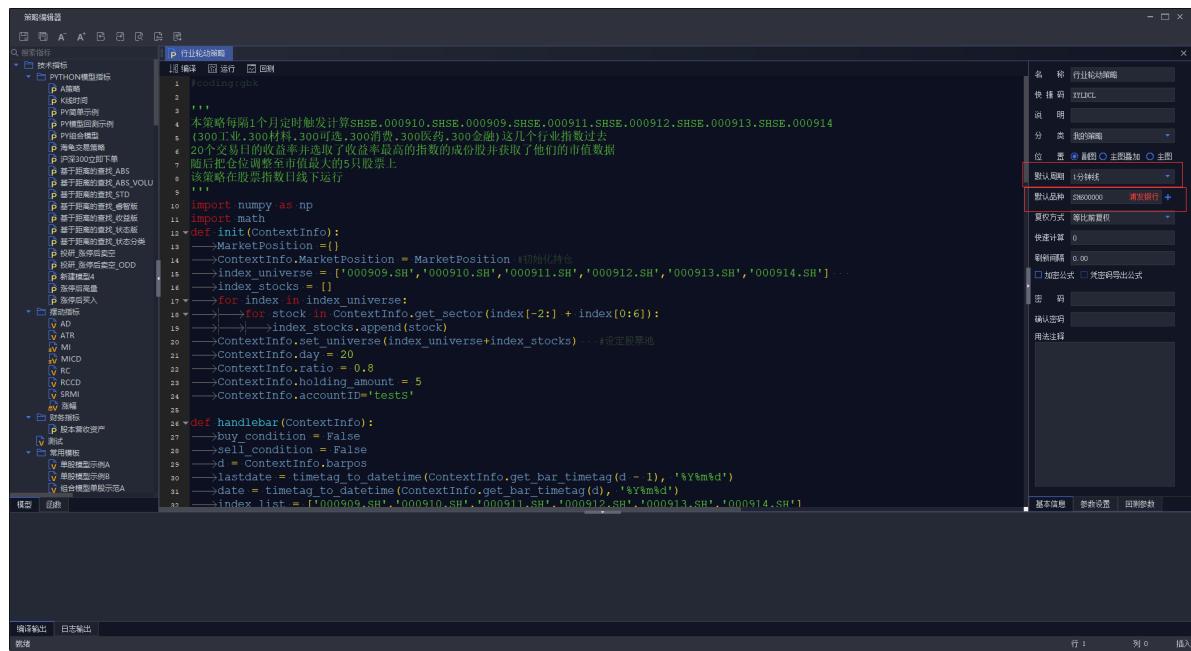
方法三，在模型管理面板右键，选择新建模型，并选择 Python 模型。



模型管理 新建模型

2.2. 策略编写

【策略编辑器】是迅投专门为模型开发者设计的，集成了模型列表、函数列表、函数帮助、模型基本信息、参数设置、回测参数等多个部分，拥有代码高亮、自动补全等便捷功能于一体的便捷的模型编辑、开发环境。



模型编辑页面 右侧可选择策略默认的周期、品种

编写 Python 策略需在开始时定义编码格式，如 gbk。

之后可选择导入第三方库，所选第三方库要在券商管理端白名单内才可运行。

Init 方法和 handlebar 方法的定义是必须的。Init 方法会在策略运行开始时调用一次，用以初始化所需对象（包裹在 ContextInfo 对象中传递），设定股票池等。

```

1 #coding:gbk → 定义编码格式
2
3 """本策略以0.8为初始权重跟踪指数标的沪深300中权重大于0.35%的成份股。
4 个股所占的百分比比(0.8*成份股权重)*100%.然后根据个股是否:
5 1.连续上涨5天 2.连续下跌5天
6 来判定个股是否为强势股/弱势股，并对其进行权重由0.8调至1.0或0.6"""
7 #在指数（例如HS300）日线下运行
8 import numpy as np → 导入第三方库
9
10 def __init__(ContextInfo):
11     #设置股票池
12     stock300 = ContextInfo.get_stock_list_in_sector('沪深300')
13     ContextInfo.stock300_weight = {}
14     stock300_symbol = []
15     stock300_weightlist = []
16     ContextInfo.index_code = ContextInfo.stockcode+"."+ContextInfo.market
17     for key in stock300:
18         #保留权重大于0.35%的成份股
19         if (ContextInfo.get_weight_in_index(ContextInfo.index_code, key) / 100) > 0.0035:
20             stock300_symbol.append(key)
21             ContextInfo.stock300_weight[key] = ContextInfo.get_weight_in_index(ContextInfo.index_code, key) / 100
22             stock300_weightlist.append(ContextInfo.get_weight_in_index(ContextInfo.index_code, key) / 100)
23             print("选择的成分股权重总和为:", np.sum(stock300_weightlist))
24             ContextInfo.set_universe(stock300_symbol)
25             #print ContextInfo.stock300_weight
26             #资产配置的初始权重,配比为0.6-0.8-1.0
27             ContextInfo.ratio = 0.8
28             #账号
29             ContextInfo.accountid = "test$"
30
31 def handlebar(ContextInfo):
32     buy_sum = 0
33     sell_sum = 0
34     index = ContextInfo.barpos
35     realtimetag = ContextInfo.get_bar_timetag(index)
36     print timetag_to_datetime(realtimetag, '%Y-%m-%d %H:%M:%S')
37     dict_close=ContextInfo.get_history_data(7,'1d','close',3)
38     #持仓市值
39     holdvalue = 0
40     #持仓
41     holdings=get_holdings(ContextInfo.accountid,"STOCK")

```

Handlebar 方法会在历史 K 线上逐 K 线调用，系统会保存函数所做更改。

在盘中交易时间，handlebar 函数会随行情推送（tick 数据）被调用，当一个 tick 数据为所在 K 线最后一个 tick 时，此 tick 调用的 handlebar 所做的更改会被系统保存，如有交易指令，会在下一根K线的第一个 tick 到来时发送；其他 tick 可以打印运行结果，但 handlebar 所做更改不会被保存，也不会发送交易信号。

编写创建完模型后，对应模型的基本信息和回测参数进行设置。

基本信息包括：

名称： 填写模型名称

快捷码： 默认根据模型名称自动生成拼音首字母拼写，如需自定义可以手动进行更改，用于键盘精灵快速引用模型

说明： 简单的说明模型功能

分类： 保存当前模型到某个分类下面

位置： 模型回测或运行时的位置，有副图、主图叠加、主图三种显示位置

默认周期： 点击模型回测或运行时的默认主图周期，可手动切换

默认品种： 点击模型回测或运行时的默认主图品种，可手动切换

复权方式： 提供不复权、前复权、后复权、等比前复权、等比后复权 5 种复权方式

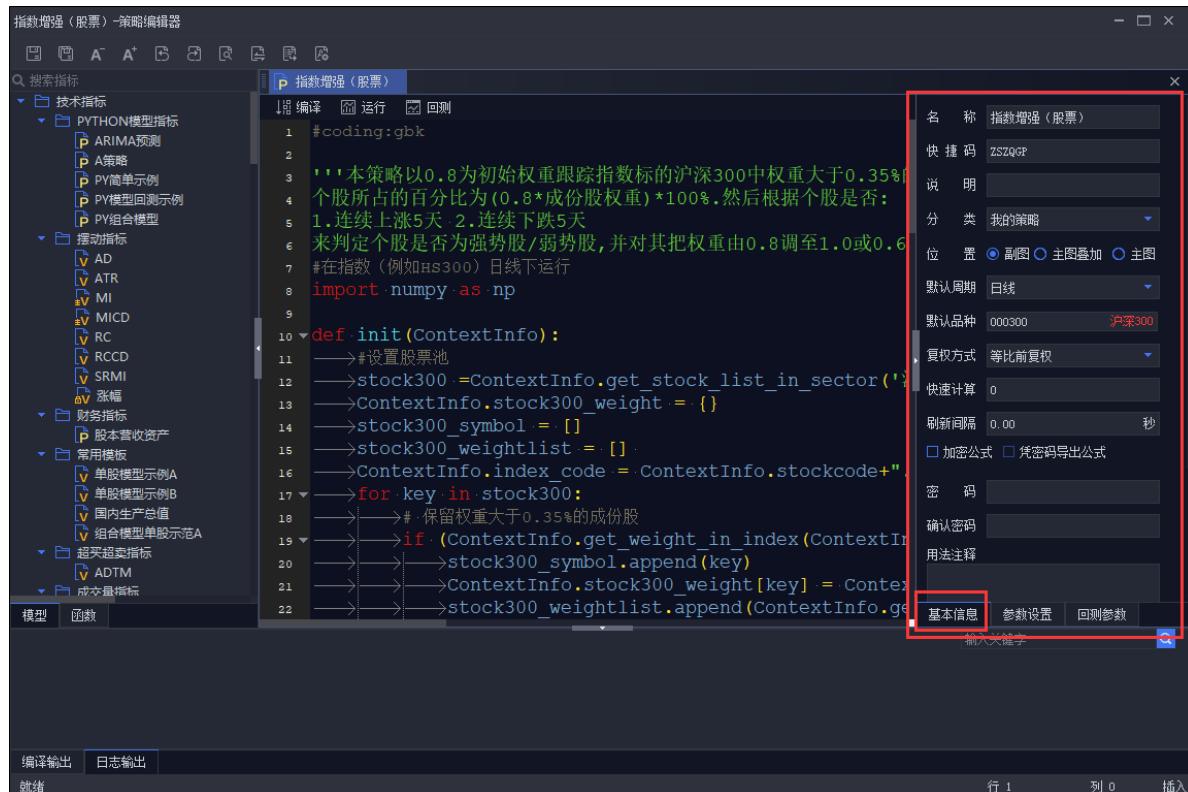
快速计算： 限制计算范围，默认为 0 时模型运行会从模型设置的默认品种（主图）的第一根 K 线开始计算，设置为 n 则从当前 K 线再往前 n 个 K 线开始计算

刷新间隔： 用来设置策略运行的时间间隔。设置了刷新间隔，即每隔一段时间策略按照当前行情运行一次

加密公式： 加密后的公式只有输入密码才可以查看源代码

凭密码导出公式： 此项只有在开启“加密公式”后才能生效，生效后只能使用密码导出到本地

用法注释： 简短的说明模型使用的一些注意项，可不填



策略编辑器 基本信息

回测模式指策略以历史行情为依据进行运算，投资者可观察该策略在历史行情所获得的年化收益率、夏普比率、最大回撤、信息比率等指标表现。

回测参数包括：

开始时间、结束时间： 设置模型回测时间区间

基准: 设置模型收益的参考基准

初始资金: 设置模型回测的初始资金

保证金比例: 设置期货的保证金比例

滑点: 设置回测撮合时的滑点，模拟真实交易的冲击成本

手续费类型: 支持按成交额比例或者固定值计算手续费

买入印花税: 设置买入印花税比例

卖出印花税: 设置卖出印花税比例

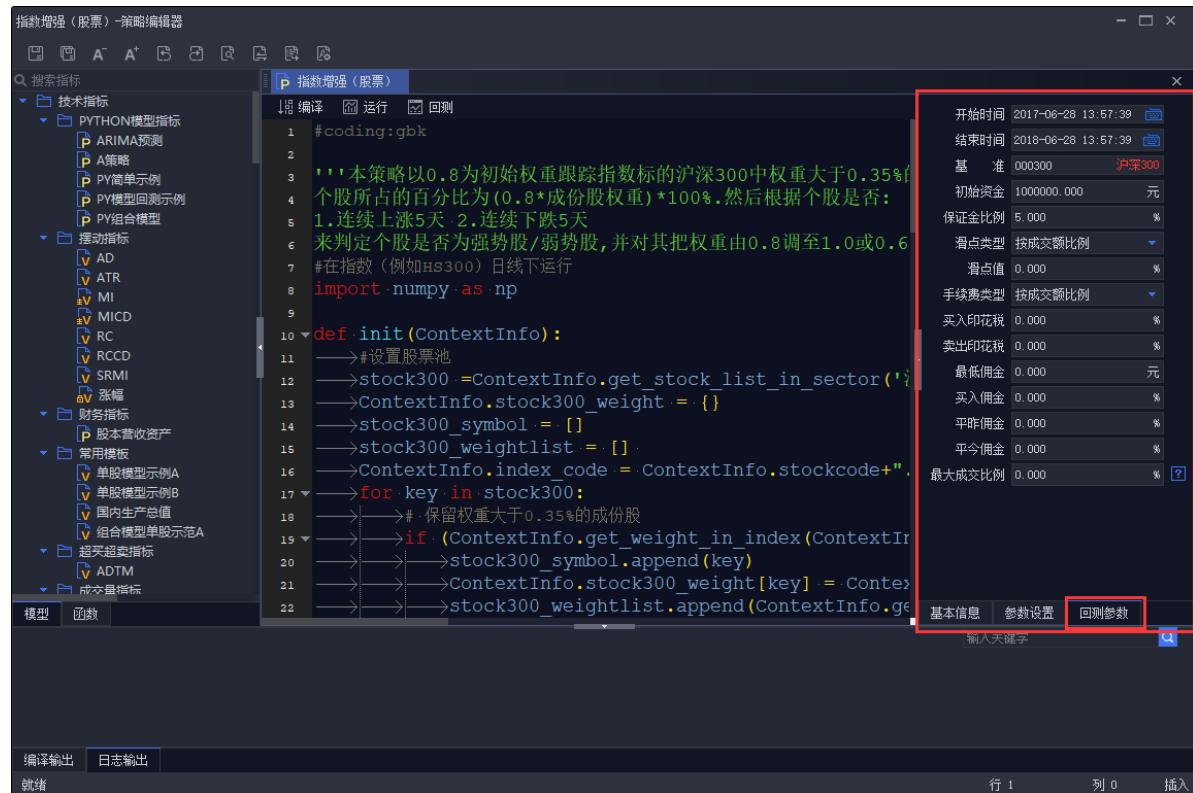
最低佣金: 设置单笔交易的最低佣金数额

买入佣金: 设置买入标的时的佣金比例

平昨佣金: 设置股票、期货平昨佣金比例

平今佣金: 设置期货平今佣金比例

最大成交比例: 控制回测中最大成交量不超过同期成交量*最大成交比例。可以点击此参数旁边的'?'按钮了解详情



策略编辑器 回测参数

使用者也可在参数设置中设置好参数值，参数名为变量名，模型中可以调用。最新值为变量默认值，运行/回测模式使用。

其中最小 / 最大 / 步长项，为遍历参数。初始项可不填。最小 / 最大都是包含在遍历区间内的，如图所示三个变量，测评时会遍历 $(20-3+1) \times (160-140) / 10 + 1 \times (-150+250) / 10 + 1$ 种组合。

新建策略文件-策略编辑器

搜索指标

技术指标

- PYTHON模型指标
 - ARIMA预测
 - A策略
 - PY简单示例
 - PY模型回测示例
 - PY组合模型
- 摆动指标
 - AD
 - ATR
 - MI
 - MICD
 - RC
 - RCCD
 - SRMI
 - 涨幅
- 财务指标
 - 股本营收资产
- 常用模板
 - 单股模型示例A
 - 单股模型示例B
 - 国内生产总值
 - 综合模型单股示例A

模型 函数

```
[2019-06-14 10:54:58.168] start back test mode
[2019-06-14 10:54:59.136] start back test mode
[2019-06-14 10:55:00.152] start back test mode
[2019-06-14 10:55:01.138] start back test mode
[2019-06-14 10:55:02.139] start back test mode
[2019-06-14 10:55:03.138] start back test mode
[2019-06-14 10:55:04.138] start back test mode
[2019-06-14 10:55:06.137] start back test mode
[2019-06-14 10:55:07.168] start back test mode
[2019-06-14 10:55:08.138] start back test mode
[2019-06-14 10:55:09.164] start back test mode
```

编译输出 日志输出

新建策略文件保存成功

行 15 列 18 插入

策略编辑器 参数设置

点击公式测评，可选择回测模式支持的指标，如单位净值，最大回撤等，作为评价标准。

新建策略文件-策略编辑器

搜索指标

技术指标

- PYTHON模型指标
 - ARIMA预测
 - A策略
 - PY简单示例
 - PY模型回测示例
 - PY组合模型
- 摆动指标
 - AD
 - ATR
 - MI
 - MICD
 - RC
 - RCCD
 - SRMI
 - 涨幅
- 财务指标
 - 股本营收资产
- 常用模板
 - 单股模型示例A
 - 单股模型示例B
 - 国内生产总值
 - 综合模型单股示例A

模型 函数

```
[2019-06-14 11:04:42.138] start back test mode
[2019-06-14 11:04:43.140] start back test mode
[2019-06-14 11:04:45.139] start back test mode
[2019-06-14 11:04:46.140] start back test mode
[2019-06-14 11:04:47.135] start back test mode
[2019-06-14 11:04:49.157] start back test mode
[2019-06-14 11:04:50.139] start back test mode
[2019-06-14 11:04:51.136] start back test mode
[2019-06-14 11:04:52.158] start back test mode
[2019-06-14 11:04:54.136] start back test mode
[2019-06-14 11:04:55.136] start back test mode
```

编译输出 日志输出

新建策略文件保存成功

行 15 列 18 插入

策略编辑器 公式评测

点击优化，评测结果弹窗显示不同参数变量组合下的回测结果，根据结果选择最优参数组合。可点击所需指标进行排序。需要注意的是，在测评之前，需要针对所选品种和周期补充数据。

测评结果

计算参数	CCI	can_buy	can_sell	单位净值	基准净值	开仓次数	平仓次数	胜率	基准年化收益	年化收益	贝塔系数	阿尔法系数	波动率	夏普比率	下方差	索提诺比率	跟踪误差	信息比率	最大回撤
578 (20,150,-200)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
579 (20,150,-190)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
580 (20,150,-180)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
581 (20,150,-170)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
582 (20,150,-160)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
583 (20,150,-150)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
584 (20,160,-250)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
585 (20,160,-240)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
586 (20,160,-230)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
587 (20,160,-220)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
588 (20,160,-210)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
589 (20,160,-200)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
590 (20,160,-190)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
591 (20,160,-180)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
592 (20,160,-170)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
593 (20,160,-160)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
594 (20,160,-150)	110.024	0	1	1	1.10297	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0

100%

关闭

2.3. 补充数据

在创建用户的模型之前，用户应使用客户端提供的“数据管理”功能，选择并补充模型所需的相应市场、品种以及对应周期的历史数据。

操作-数据管理

2.4. 策略运行

策略编写完毕后，点击编译，可保存策略。编译按钮在 Python 策略中只起保存功能，不会检查语法与引用的正误。之后点击运行可以看到策略运行效果（如有错误，会在日志输出的位置报错）。

The screenshot shows the 'New Strategy Editor' window. On the left is a tree view of strategy categories: Technical Indicators, PYTHON Model Indicators, Moving Averages, Financial Indicators, and Common Templates. The main area displays Python code for a strategy named '新建策略文件'. The code initializes ContextInfo with specific stocks and sets buy/sell flags. It also defines a handleBar function to calculate the CCI indicator. The right side contains a settings panel with fields for Name, Shortcut, Description, Category, Location, Default Period, Default品种, Authorization Method, Refresh Interval, Encryption, and Password.

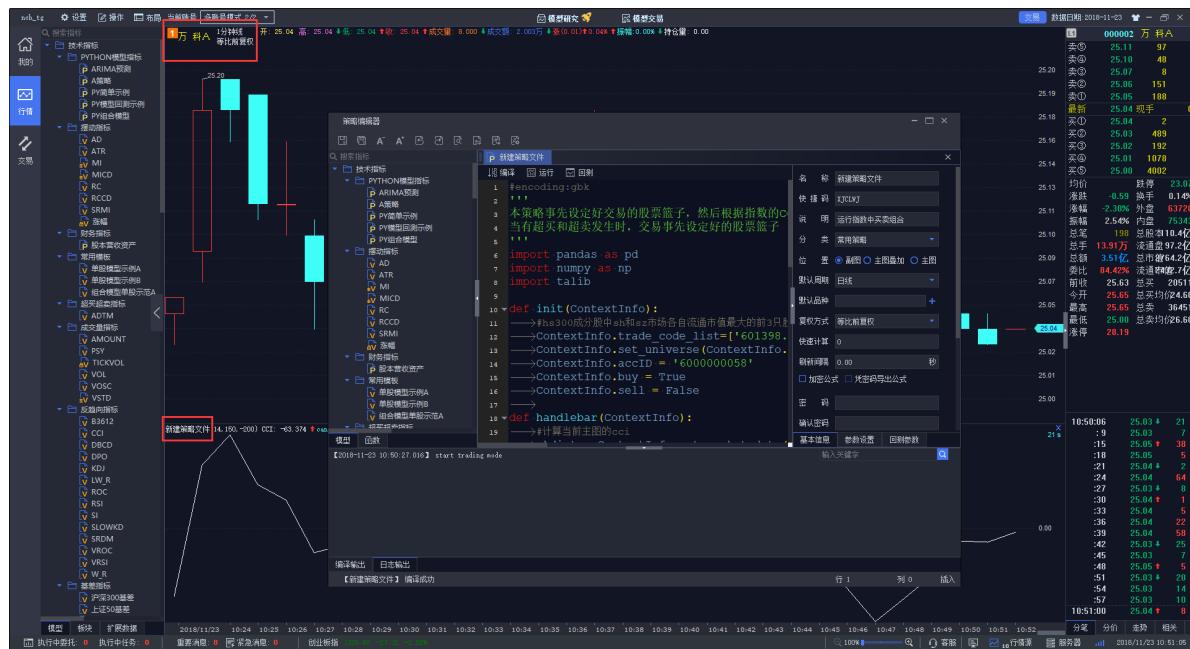
```

#encoding:gbk
...
本策略事先设定好交易的股票篮子，然后根据指数的cci指标来判断超当有超买和超卖发生时，交易事先设定好的股票篮子
...
import pandas as pd
import numpy as np
import talib
...
def init(ContextInfo):
    ...
    ContextInfo.trade_code_list=['601398.SH','601857.SH']
    ContextInfo.set_universe(ContextInfo.trade_code_list)
    ContextInfo.accID = '6000000058'
    ContextInfo.buy = True
    ContextInfo.sell = False
...
def handleBar(ContextInfo):
    ...

```

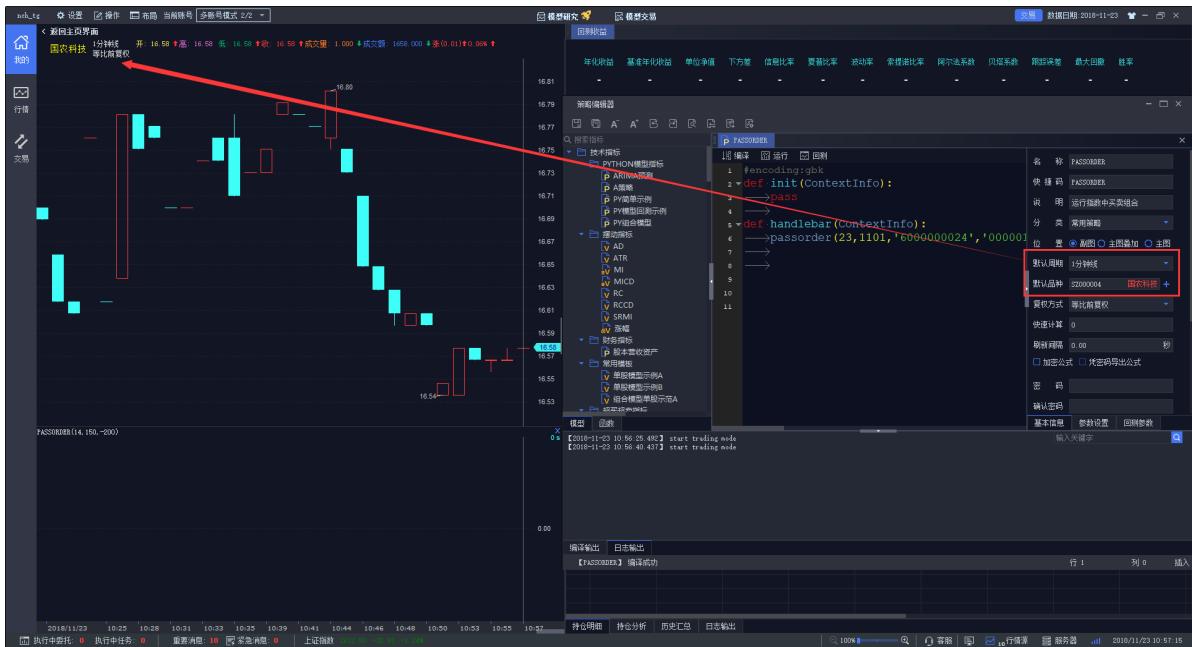
策略编辑器-运行

如当前系统所处界面为“行情”界面或“交易”界面，点击运行之前，需在行情中手动设置好K线品种和周期，点击运行后，策略即可在当前主图下运行，如下图所示。



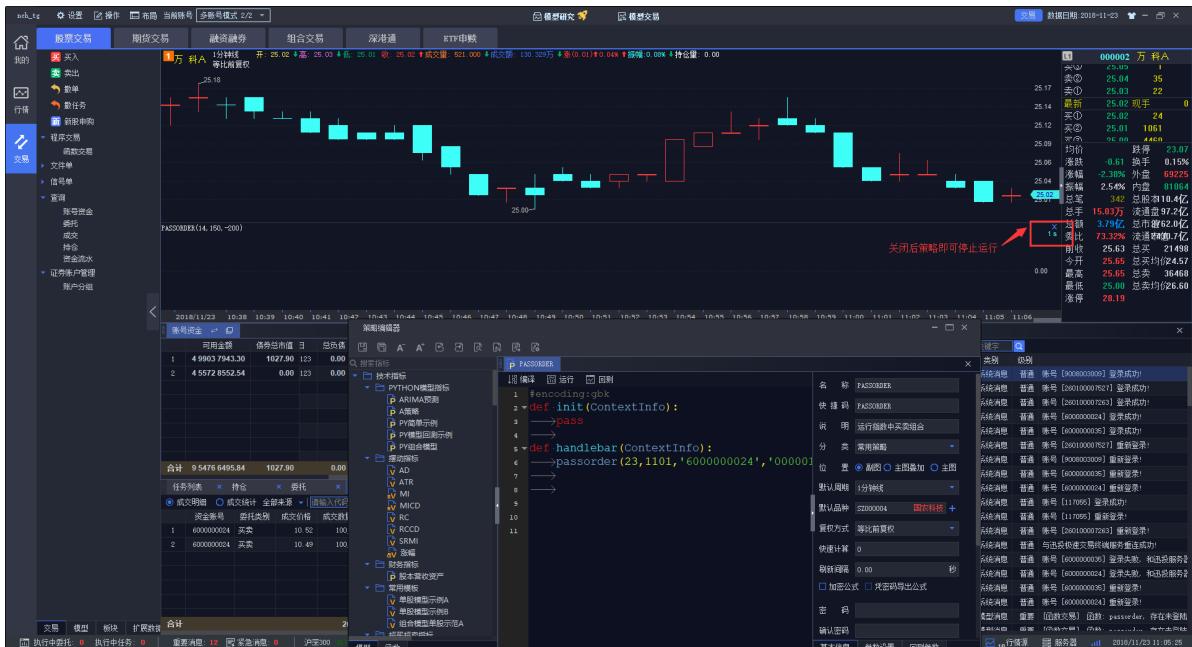
策略运行状态之一

如系统当前界面处于“我的主页”界面或“模型研究”和“模型交易”等非行情界面，点击运行时，会基于策略编辑器 - 基本信息中所设置的默认周期和默认品种运行。

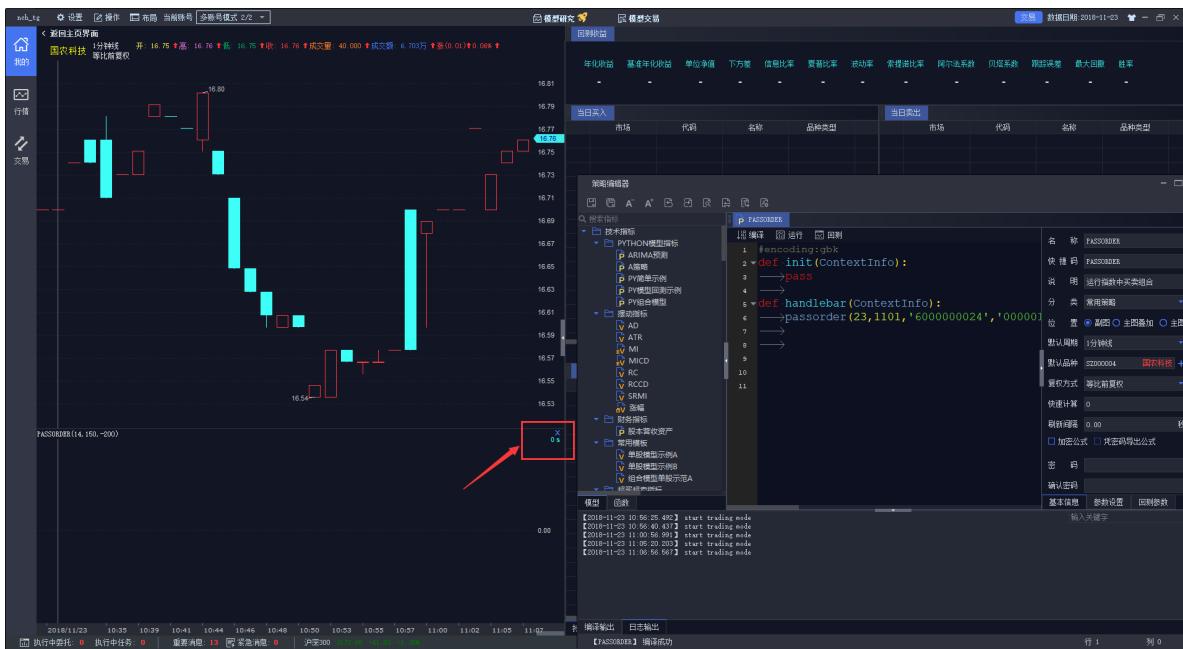


策略运行状态之二

当选择的运行位置为副图时，**位 置** **副图** **主图叠加** **主图** 如想关闭策略，将主图下方策略运行的附图关闭即可。



关闭运行中的策略之一



关闭运行中的策略之二

当选择的运行位置为主图叠加时，如想关闭策略，在主图上右键单击取消叠加指标即可。

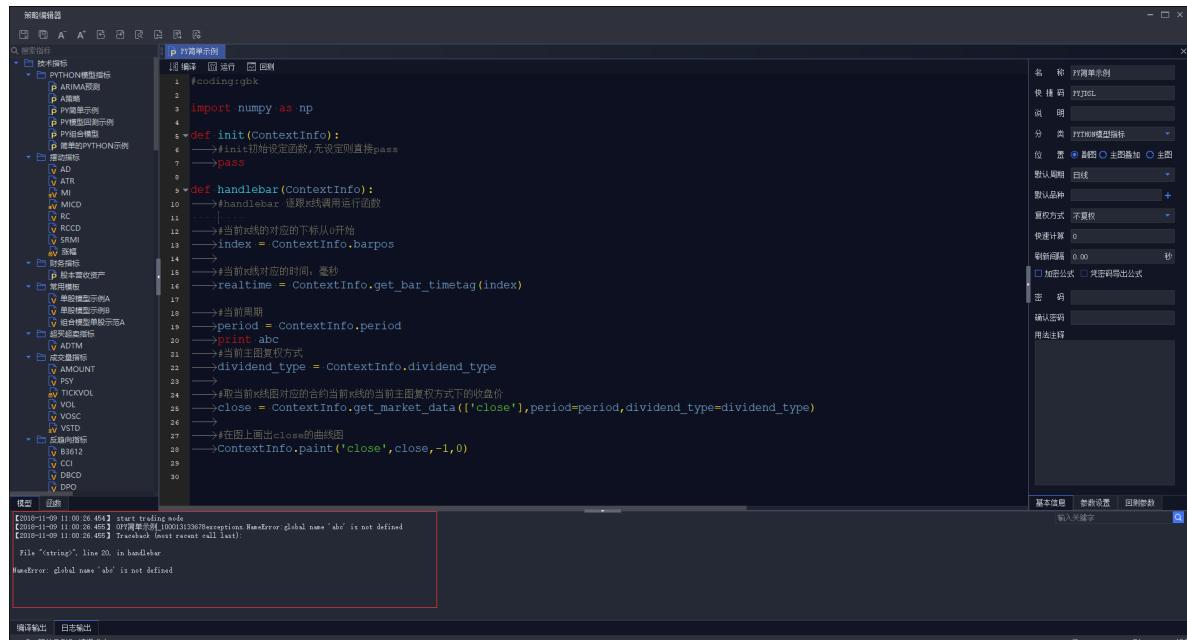
当选择的运行位置为主图时，键盘精灵输入KLINE即可结束模型运行。

关闭运行中的策略之三



2.5. 策略调试

如果策略运行不成功，需要进行策略调试这一步。当运行出错时，报错信息会显示在日志输出面板，以供修改调试之用。



策略调试-输出日志

2.6. 策略回测

对某一策略编译成功后，点击回测，可以通过日志输出查看模型基于历史行情数据回测情况和表现。

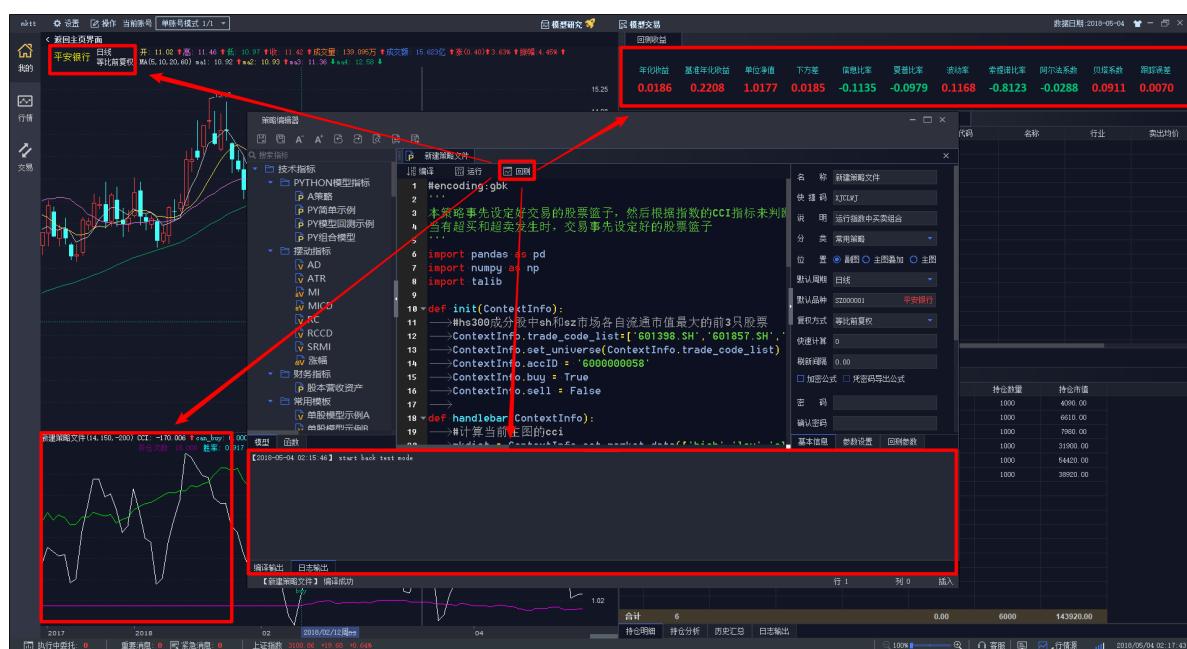
在回测之前，需要设置好策略回测运行的主图品种和周期，以及相关的回测参数。回测主图和周期可以在策略编辑器-基本信息中进行设置，回测开始和结束时间、基准、费率等可以在策略编辑器 - 回测参数中设置。

用户在回测前，需根据此策略回测运行的主图、周期和时间，在【数据管理】中对行情数据进行下载补充。如回测时间为 20180930 至 20190531，运行主图为 SZ.000001 平安银行日线，补充数据时可进行如下设置。



操作-数据管理

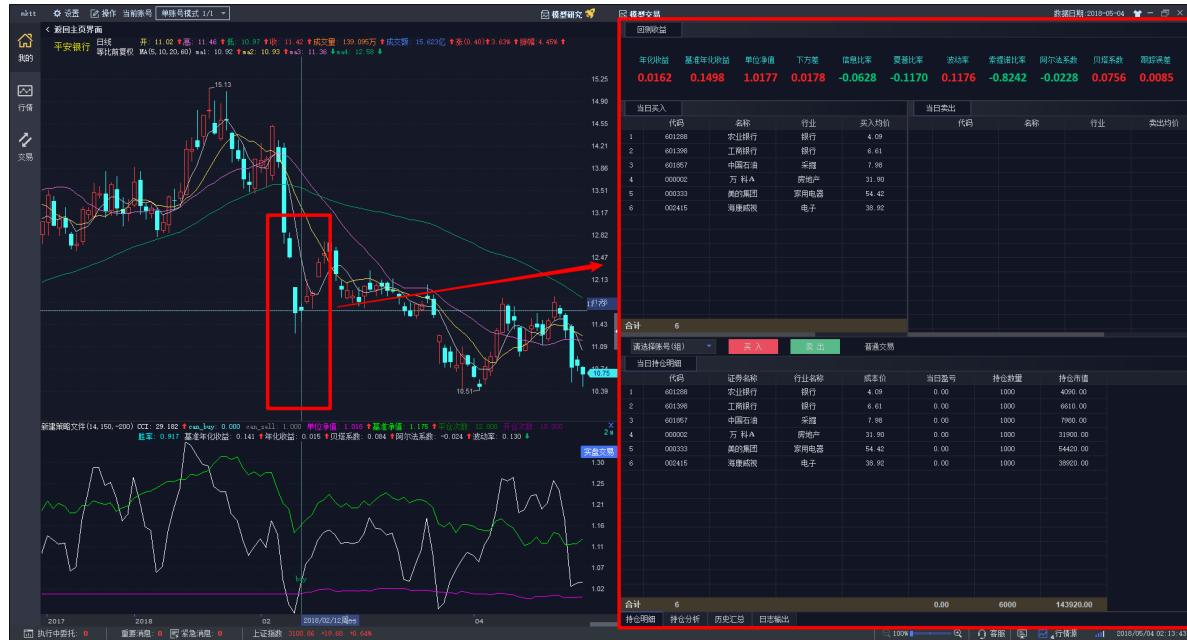
如果回测正常的话，主界面会跳转到模型设置的默认标的和默认周期界面，并输出模型绩效分析结果。



策略编辑器 模型回测

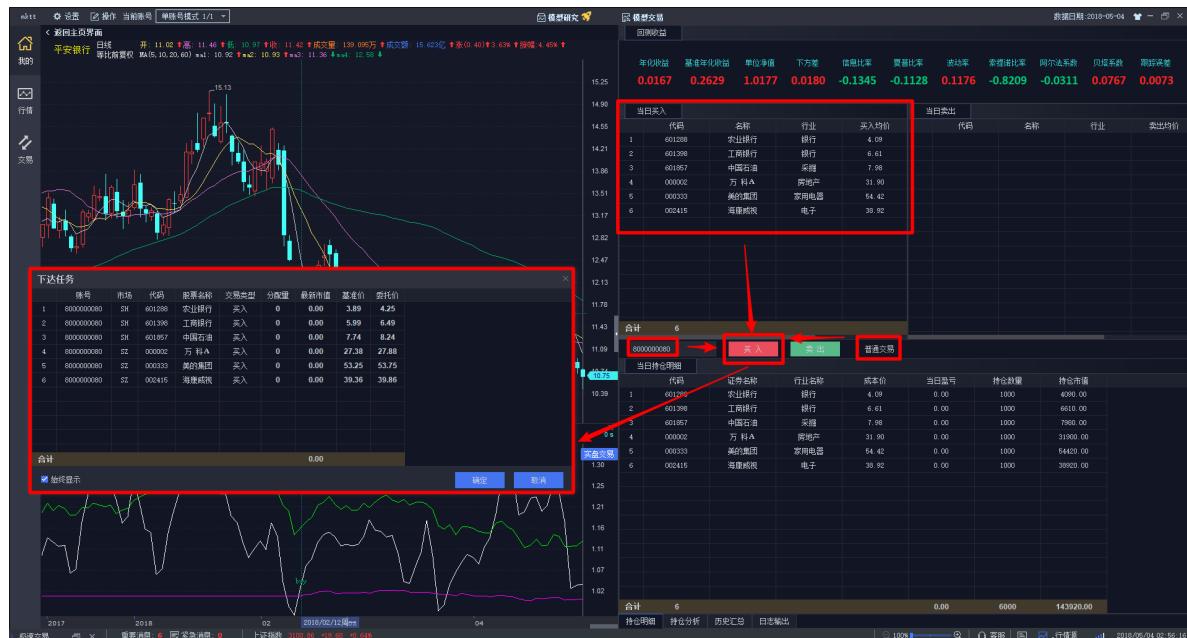
此时，可最小化或者关闭【策略编辑器】，并对回测结果进行分析，随着光标在 K 线主图上的移动，右边回测结果展示窗口会动态显示截止光标所在当日的绩效分析结果（包括年化收益，基准年化收益，单位净值，下方差，信息比率，夏普比率，波动率，索提诺比率，阿尔法系数，贝塔系数，跟踪误差，最大回撤，胜率等）、当日买入、当日卖出、持仓列表。

以上列表均可鼠标右键复制和导出数据。



回测结果分析-回测结果随十字光标移动动态展示

如需根据模型生成的买入、卖出列表进行手工交易，则可直接点击“买入”或“卖出”按钮，系统会弹出下单界面由用户进行确认后进行普通交易下单或算法交易下单。交易方式可点击卖出按钮右侧的普通交易进行切换设置。



回测结果买入

副图回测指标：提供图形化的展示，除去绩效分析的相关指标外，用户可以通过编辑模型代码自定义输出一些特色指标，鼠标右键可以选择复制模型运行结果（每一天的数据）。



回测结果分析-附图指标输出

另外，回测结果还提供了持仓分析、历史板块汇总、操作明细、日志输出等信息，方便用户进行深入分析。

持仓分析：可查看光标所在当天持仓的行业分布，展示在相关行业的市值情况、盈利情况、权重以及股票数量情况，鼠标右键可以复制和导出数据；可切换对比基准，和模型持仓进行对比。

历史板块汇总：可查看模型自回测日期以来到光标所在日期该模型交易标的的汇总信息，包括累计盈亏、累计交易量、累计交易额、持仓天数等；点击选择板块，可以自行选择其常用板块进行板块的各项数据累计汇总；汇总数据均可以进行排序，鼠标右键可以复制和导出数据。

操作明细：可查看模型回测的历史每一笔交易的明细。

日志输出：可用于调试输出模型回测和运行情况。

回测收益											
年化收益	基准年化收益	单位净值	下方差	信息比率	夏普比率	波动率	索提诺比率	阿尔法系数	贝塔系数	跟踪误差	最
0.0186	0.2208	1.0177	0.0185	-0.1135	-0.0979	0.1168	-0.8123	-0.0288	0.0911	0.0070	0.
当日买入						当日卖出					
代码	名称	行业	买入均价	代码	名称	行业	卖出均价				
1 601288	农业银行	银行	4.09								
2 601398	工商银行	银行	6.61								
3 601857	中国石油	采掘	7.98								
4 000002	万科A	房地产	31.90								
5 000333	美的集团	家用电器	54.42								
6 002415	海康威视	电子	38.92								
合计 6											
8000000080			买入	卖出	普通交易						
当日持仓明细											
代码	证券名称	行业名称	成本价	当日盈亏	持仓数量	持仓市值					
1 601288	农业银行	银行	4.09	0.00	1000	4090.00					
2 601398	工商银行	银行	6.61	0.00	1000	6610.00					
3 601857	中国石油	采掘	7.98	0.00	1000	7980.00					
4 000002	万科A	房地产	31.90	0.00	1000	31900.00					
5 000333	美的集团	家用电器	54.42	0.00	1000	54420.00					
6 002415	海康威视	电子	38.92	0.00	1000	38920.00					
合计 6 0.00 6000 143920.00											
持仓明细 持仓分析 历史汇总 日志输出											

回测结果分析-绩效分析、当日买入、当日卖出、当日持仓



回测结果分析-持仓分析

历史品种汇总								
高级查询								
	市场	代码	名称	品种类型	行业	多空	累计盈亏	累计交易量
1	SH	601288	农业银行	股票	银行	-	563.85	3000
2	SH	601398	工商银行	股票	银行	-	1101.41	3000
3	SH	601857	中国石油	股票	采掘	-	50.23	3000
4	SZ	000002	万科A	股票	房地产	-	2031.56	3000
5	SZ	000333	美的集团	股票	家用电器	-	4973.37	3000
6	SZ	002415	海康威视	股票	电子	-	8953.53	3000
合计		6					17673.96	18000
历史板块汇总								
选择板块								
高级查询								
	代码	板块名称	盈利	交易费用	市值	效益权重	持仓天数	买卖累计
1	SW采掘成份	SW采掘成份	50.23	0.00	23730.15	0.00%	56	3000
2	SW化工成份	SW化工成份	0.00	0.00	0.00	0.00%	0	0
3	SW钢铁成份	SW钢铁成份	0.00	0.00	0.00	0.00%	0	0
4	SW有色金属成份	SW有色金属成份	0.00	0.00	0.00	0.00%	0	0
5	SW建筑材料成份	SW建筑材料成份	0.00	0.00	0.00	0.00%	0	0
6	SW建筑装饰成份	SW建筑装饰成份	0.00	0.00	0.00	0.00%	0	0
7	SW电气设备成份	SW电气设备成份	0.00	0.00	0.00	0.00%	0	0
8	SW机械设备成份	SW机械设备成份	0.00	0.00	0.00	0.00%	0	0
9	SW国防军工成份	SW国防军工成份	0.00	0.00	0.00	0.00%	0	0
10	SW汽车成份	SW汽车成份	0.00	0.00	0.00	0.00%	0	0
11	SW家用电器成份	SW家用电器成份	4973.37	0.00	127926.63	0.00%	56	3000
12	SW纺织服装成份	SW纺织服装成份	0.00	0.00	0.00	0.00%	0	0
13	SW轻工制造成份	SW轻工制造成份	0.00	0.00	0.00	0.00%	0	0
14	SW商业贸易成份	SW商业贸易成份	0.00	0.00	0.00	0.00%	0	0
15	SW农林牧渔业成份	SW农林牧渔业成份	0.00	0.00	0.00	0.00%	0	0
16	SW食品饮料成份	SW食品饮料成份	0.00	0.00	0.00	0.00%	0	0
合计		28	17673.96		342288.97			
持仓明细								
持仓分析								
历史汇总								
日志输出								

回测结果分析-历史品种汇总、历史板块汇总

操作明细							
	代码	名称	行业	操作时间	操作价格	数量	市值
1	601288	农业银行	银行	2017-03-22 00:0	3.09	1000	3086.15
2	601398	工商银行	银行	2017-03-22 00:0	4.46	1000	4455.07
3	601857	中国石油	采掘	2017-03-22 00:0	7.87	1000	7869.45
4	000002	万科A	房地产	2017-03-22 00:0	20.38	1000	20376.73
5	000333	美的集团	家用电器	2017-03-22 00:0	33.39	1000	33386.63
6	002415	海康威视	电子	2017-03-22 00:0	20.72	1000	20716.47
7	601288	农业银行	银行	2017-06-01 00:0	3.41	1000	3410.00
8	601398	工商银行	银行	2017-06-01 00:0	5.08	1000	5076.48
9	601857	中国石油	采掘	2017-06-01 00:0	7.74	1000	7741.25
10	000002	万科A	房地产	2017-06-01 00:0	20.40	1000	20396.08
11	000333	美的集团	家用电器	2017-06-01 00:0	36.50	1000	36500.00
12	002415	海康威视	电子	2017-06-01 00:0	27.95	1000	27950.00
13	601288	农业银行	银行	2017-08-14 00:0	3.61	1000	3610.00
14	601398	工商银行	银行	2017-08-14 00:0	5.49	1000	5490.00
15	601857	中国石油	采掘	2017-08-14 00:0	7.88	1000	7880.70
16	000002	万科A	房地产	2017-08-14 00:0	21.23	1000	21227.78
17	000333	美的集团	家用电器	2017-08-14 00:0	40.12	1000	40120.00
18	002415	海康威视	电子	2017-08-14 00:0	30.15	1000	30150.00
19	601288	农业银行	银行	2017-08-30 00:0	3.85	1000	3850.00
20	601398	工商银行	银行	2017-08-30 00:0	5.97	1000	5970.00
21	601857	中国石油	采掘	2017-08-30 00:0	8.06	1000	8059.13
22	000002	万科A	房地产	2017-08-30 00:0	23.24	1000	23240.00
23	000333	美的集团	家用电器	2017-08-30 00:0	41.98	1000	41980.00
24	002415	海康威视	电子	2017-08-30 00:0	31.87	1000	31870.00
25	601288	农业银行	银行	2018-02-12 00:0	4.09	1000	4090.00
合计		30			30000	558331.91	

持仓明细	持仓分析	历史汇总	日志输出
------	------	------	-------------

回测结果分析-操作明细、日志输出

2.7. 回测、运行两种模式的区别

在模型编辑器中，有“回测”和“运行”两个按钮，分别代表两种模式，它们之间的区别如下：

(1) 回测模式指策略以历史行情为依据，以回测参数中的开始时间、结束时间为回测时间区间进行运算，投资者可观察该策略在历史行情所获得的年化收益率、夏普比率、最大回撤、信息比率等指标表现。

(2) 运行模式指策略根据实时行情信号进行运算，以主图行情开始时间到当前时间为运行区间，进行策略的模拟运行，但不进行真实的委托。

注：如果需要向模拟/实盘柜台发送真实的委托，请将策略加入到“模型交易”中。

3. Python API 手册

(1) 迅投 Python API 是绑定在我们的主流系统中的，底层是 C++，保证了迅投 Python API 对行情数据，财务数据等的高访问速度。

(2) 支持跨资产类别的量化交易策略的回测、模拟交易功能。依照我们提供的的策略模板，可以编程出各种独特的投资策略，进行单一市场或跨市场交易。

(3) 可以便捷的对投资策略进行日频率、分钟频率的历史回测，了解策略在市场中的历史表现，比较不同策略的优劣；也可以对策略进行模拟交易，了解策略在市场中的真实表现，更加准确的对策略进行评估。

(4) 采用的是所见即所得的策略运行机制，完美展示策略运行结果。

3.1. 对第三方库的支持

QMT Python API 根据客户端版本不同分别提供基于 **Python 2.X** 和 **Python 3.X** 规范的标准量化投资策略应用程序接口，本文档示例代码基于 **Python 3.X** 规范。我司主要通过以下两种方式对外提供：

3.1.1. 系统自带的 Python 环境

QMT 系统的安装包默认自带 Python 运行环境。用户安装完迅投客户端后，默认可以直接使用 Python。在这个打包的 Python 环境中，迅投除了提供标准的 Python api 带的库外，还集成了如下一些第三方库：

名称	说明
NumPy	NumPy (Numeric Python) 提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。专为进行严格的数字处理而产生。
Pandas	Python Data Analysis Library 或 Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。Pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。
Patsy	一个线性模型分析和构建工具库。
SciPy	SciPy 函数库在 NumPy 库的基础上增加了众多的数学、科学以及工程计算中常用的库函数。例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等等。
Statsmodels	Python 的统计建模和计量经济学工具包，包括一些描述统计、统计模型估计和推断。
TA_Lib	称作技术分析库，是一种广泛用在程序化交易中进行金融市场数据的技术分析的函数库。它提供了多种技术分析的函数，可以大大方便我们量化投资中编程工作，内容包括：多种指标，如 ADX, MACD, RSI, 布林轨道等；K 线形态识别，如黄昏之星，锤形线等等。

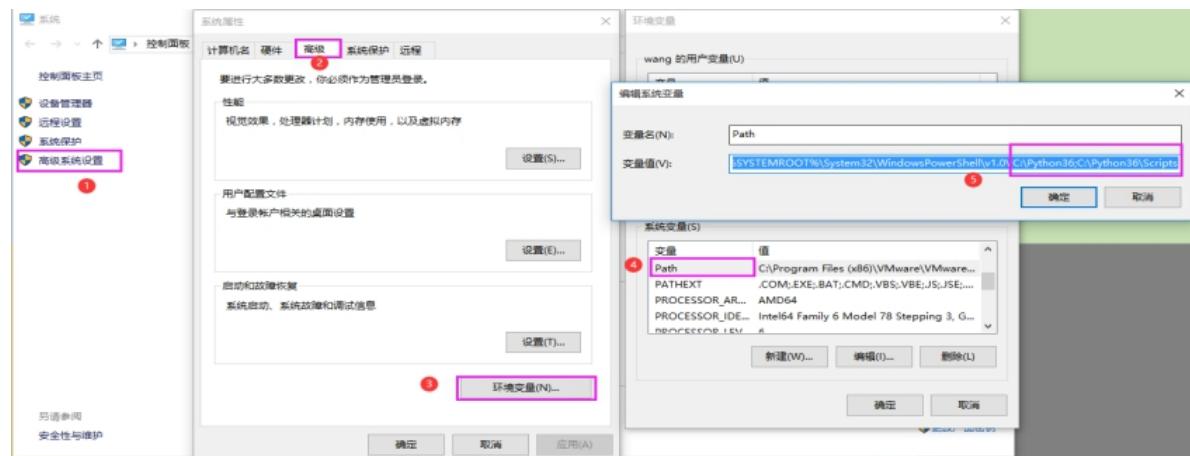
3.1.2. 第三方库导入指引

除迅投提供的标准 Python api 和集成的部分第三方库，用户也可自己在 Python 官网下载其他所需第三方库，使用方式如下：

(1) 本地安装 Python 环境，下载 python3.6，Python 官网：<https://www.python.org/downloads/release/python-360/>

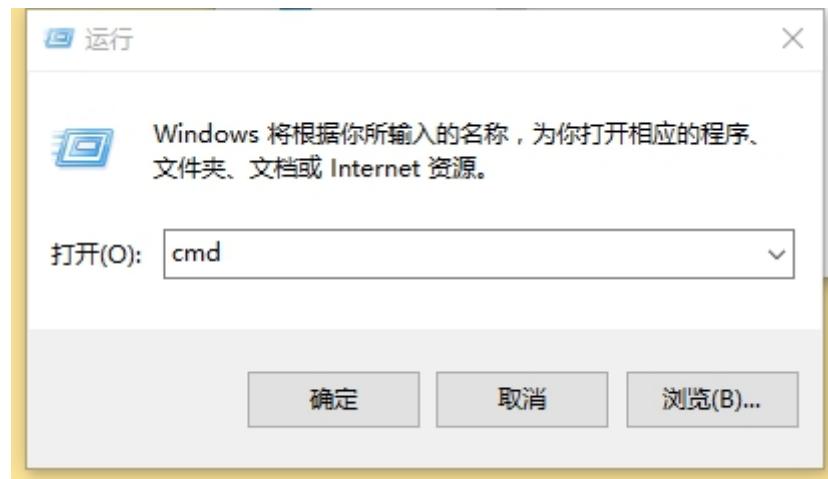
(2) 安装位置：C:\Python36

新增环境变量：我的电脑--属性--高级系统设置--高级--环境变量--path:
C:\Python36;C:\Python36\Scripts



(3) Python环境检查

Win+R 打开运行,输入cmd



检查Python变量

```
C:\Users\wang>python
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

(4) 安装第三方库

安装前先确认客户端安装目录，根据个人电脑进行调整。

安装时若遇到下面错误提示，请执行pip更新命令 python -m pip install --upgrade pip

```
C:\Users\wang>pip install -t E:\[REDACTED] 正券QMT交易端20962\bin.x64\Lib\site-packages
You must give at least one requirement to install (see "pip help install")
You are using pip version 9.0.1, however version 21.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\wang>python -m pip install --upgrade pip
```

安装三方库命令 pip install openpyxl -t E:\QMT交易端20962\bin.x64\Lib\site-packages

```
C:\Users\wang>pip install openpyxl -t E:\[REDACTED]证券QMT交易端20962\bin.x64\Lib\site-packages
Collecting openpyxl
  Downloading openpyxl-3.0.6-py2.py3-none-any.whl (242 kB)
|████████████████████████████████████████████████████████████████████████| 242 kB 595 kB/s
Collecting jdcal
  Downloading jdcal-1.4.1-py2.py3-none-any.whl (9.5 kB)
Collecting et_xmlfile
  Downloading et_xmlfile-1.0.1.tar.gz (8.4 kB)
Using legacy 'setup.py install' for et_xmlfile, since package 'wheel' is not installed.
Installing collected packages: jdcal, et_xmlfile, openpyxl
  Running setup.py install for et_xmlfile ... done
Successfully installed et_xmlfile-1.0.1 jdcal-1.4.1 openpyxl-3.0.6
WARNING: Target directory E:\中信建投证券QMT交易端20962\bin.x64\Lib\site-packages\__pycache__ already exists. Specify --upgrade to force replacement.
```

(5) 检查安装结果

安装位置\bin.x64\Lib\site-packages检查安装库

名称	修改日期	类型	大小
ecos.py	2019/11/25 11:18	Python File	3 KB
pvectorc.cp36-win_amd64.pyd	2019/11/25 11:18	Python Extension	33 KB
jupyter.py	2019/11/25 11:18	Python File	1 KB
kiwisolver.cp36-win_amd64.pyd	2019/11/25 11:18	Python Extension	140 KB
entrypoints.py	2019/11/25 11:18	Python File	9 KB
pandocfilters.py	2019/11/25 11:18	Python File	9 KB
_mssql.cp36-win_amd64.pyd	2019/11/25 11:17	Python Extension	2,178 KB
six.py	2019/11/25 11:17	Python File	32 KB
pylab.py	2019/11/25 11:16	Python File	1 KB
README.txt	2019/11/25 11:16	文本文档	1 KB
pickleshare.py	2019/11/25 11:16	Python File	10 KB
setuptools.py	2019/11/25 11:16	Python File	1 KB
termcolor.py	2019/11/25 11:16	Python File	5 KB
openpyxl-3.0.6.dist-info	2021/3/4 14:24	文件夹	
et_xmlfile	2021/3/4 14:24	文件夹	
et_xmlfile-1.0.1-py3.6.egg-info	2021/3/4 14:24	文件夹	

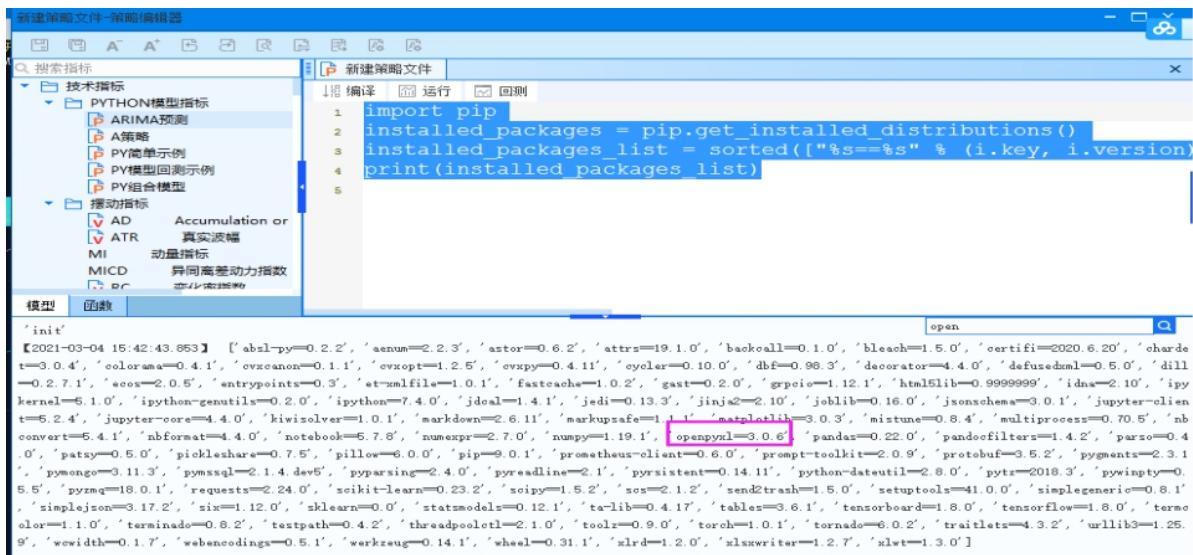
新建策略输出确认

```
import pip

installed_packages = pip.get_installed_distributions()

installed_packages_list = sorted(["%s==%s" % (i.key, i.version) for i in installed_packages])

print(installed_packages_list)
```



```
import pip

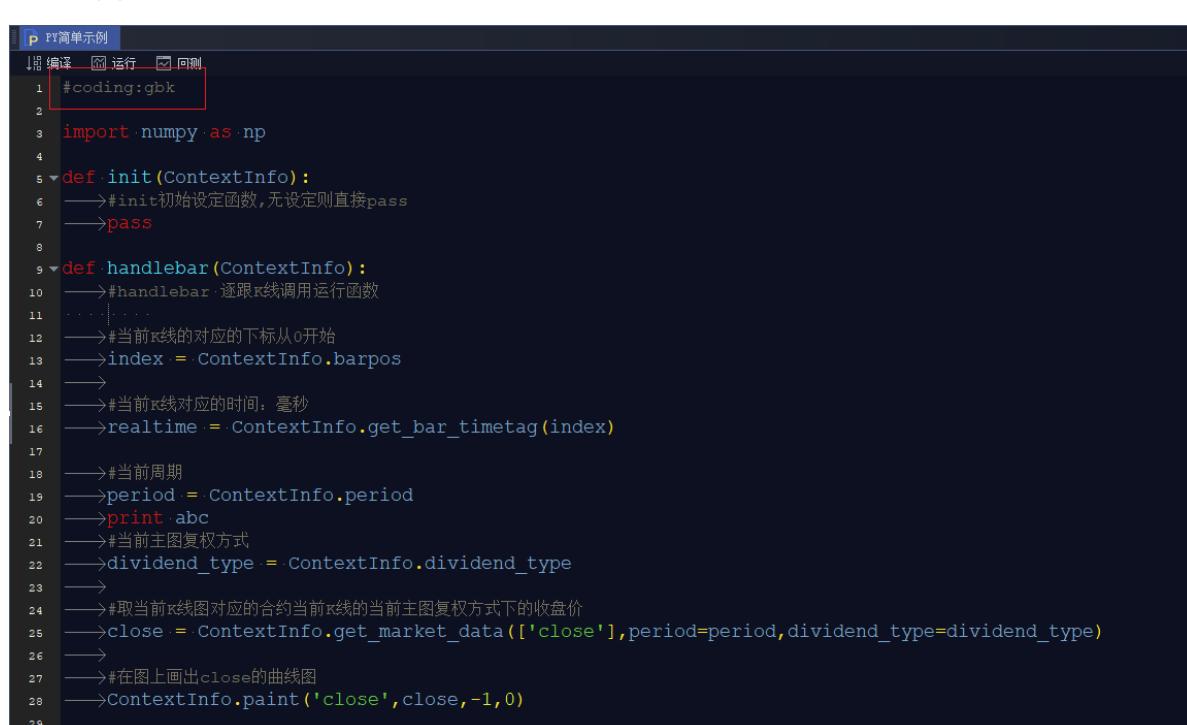
installed_packages = pip.get_installed_distributions()
installed_packages_list = sorted(["%s==%s" % (i.key, i.version) for i in installed_packages])
print(installed_packages_list)
```

3.2. 使用说明

3.2.1. 概述

在编写一个策略时，首先需要在代码的最前一行写上：

```
#coding:gbk 统一脚本的编码格式是GBK
```



```
PT简单示例
1 #coding:gbk
2
3 import numpy as np
4
5 def init(ContextInfo):
6     #init初始设定函数,无设定则直接pass
7     pass
8
9 def handlebar(ContextInfo):
10    #handlebar 逐根K线调用运行函数
11    ...
12    #当前K线的对应的下标从0开始
13    index = ContextInfo.barpos
14    ...
15    #当前K线对应的时间:毫秒
16    realtime = ContextInfo.get_bar_timetag(index)
17
18    #当前周期
19    period = ContextInfo.period
20    print abc
21    #当前主图复权方式
22    dividend_type = ContextInfo.dividend_type
23    ...
24    #取当前K线图对应的合约当前K线的当前主图复权方式下的收盘价
25    close = ContextInfo.get_market_data(['close'], period=period, dividend_type=dividend_type)
26    ...
27    #在图上画出close的曲线图
28    ContextInfo.paint('close', close, -1, 0)
29
```

3.2.1.2. 重要方法

QMT 系统 Python API 策略代码结构分两个部分，初始化函数 `init()` 和行情事件函数 `handlebar()`。

初始化函数 `init()` 在整个策略中只执行一次，一般在此函数中设置交易佣金、滑点、基准等一些常用参数。



```
新建策略文件 *
1 #encoding:gbk
2 """
3 本策略事先设定好交易的股票篮子，然后根据指数的CCI指标来判断超买和超卖
4 当有超买和超卖发生时，交易事先设定好的股票篮子
5 """
6 import pandas as pd
7 import numpy as np
8 import talib
9
10 def init(ContextInfo):
11     #hs300成分股中sh和sz市场各自流通市值最大的前3只股票
12     ContextInfo.trade_code_list=['601398.SH','601857.SH','601288.SH','000333.SZ','002415.SZ','000002.SZ']
13     #设定股票池
14     ContextInfo.set_universe(ContextInfo.trade_code_list)
15     #设定交易账号
16     ContextInfo.accID = '6000000058'
17     #设定买卖状态
18     ContextInfo.buy = True
19     ContextInfo.sell = False
20     """


```

行情事件函数 `handlebar()` 为行情数据的处理函数，当前图中每根 K 线为一个行情数据，`handlebar()` 在每根 K 线上分别依次执行一次。特别的，对于实时行情，最后一根 K 线还没确定时，每个 tick 执行一次 `handlebar()`。`handlebar()` 里面一般写整个策略的执行逻辑。

```
21 def handlebar(ContextInfo):
22     #1.计算当前主图的cci
23     mdict = ContextInfo.get_market_data(['high','low','close'],count=int(period)+1)
24     highs = np.array(mdict['high'])
25     lows = np.array(mdict['low'])
26     closes = np.array(mdict['close'])
27     cci_list = talib.CCI(highs,lows,closes,timeperiod=int(period))
28     now_cci = cci_list[-1]
29     ContextInfo.paint("CCI",now_cci,-1,0,'noaxis')
30
31     #2.交易策略
32     if len(cci_list)<2:
33         return
34
35     #买入条件：指数CCI进入超卖区间时触发买入信号，过滤连续超卖导致的买入信号
36     buy_condition = cci_list[-2]<buy_value<=now_cci and ContextInfo.buy
37     #卖出条件：指数CCI进入超买区间时触发卖出信号，过滤连续超买导致的卖出信号
38     sell_condition = cci_list[-2]>sell_value>=now_cci and ContextInfo.sell
39
40     if buy_condition:
41         ContextInfo.buy = False
42         ContextInfo.sell = True
43         #列表中股票分别下单买入10手
44         for stockcode in ContextInfo.trade_code_list:
45             order_lots(stockcode,10,ContextInfo,ContextInfo.accID)
46     elif sell_condition:
47         ContextInfo.buy = True
48         ContextInfo.sell = False
49         #列表中股票分别下单卖出10手
50         for stockcode in ContextInfo.trade_code_list:
51             order_lots(stockcode,-10,ContextInfo,ContextInfo.accID)
```

Python策略中必须有包含这两个基本方法，否则策略将运行不起来。

(1) 初始化函数 init()

用法: init(ContextInfo)

释义: 初始化函数，只在整个策略开始时调用运行一次

参数: ContextInfo: 策略运行环境对象，可以用于存储自定义的全局变量

返回: 无

示例:

```
1 def init(ContextInfo):
2     ContextInfo.initProfit = 0
```

(2) 行情事件函数 handlebar()

用法: handlebar(ContextInfo)

释义: 行情事件函数，每根 K 线运行一次；实时行情获取状态下，先每根历史 K 线运行一次，再在每个 tick 数据来后驱动运行一次

参数: ContextInfo: 策略运行环境对象，可以用于存储自定义的全局变量

返回: 无

示例:

```
1 | def handlebar(ContextInfo):  
2 |     # 输出当前运行到的 K 线的位置  
3 |     print(ContextInfo.barpos)
```

3.2.2. ContextInfo 对象

ContextInfo 是策略运行环境对象，是 init() 和 handlebar() 这两个基本方法必传参数，里面包括了终端自带的属性和方法。一般情况下不建议对ContextInfo添加自定义属性，ContextInfo会随着bar的切换而重置到上一根bar的结束状态，建议用自建的全局变量来存储。

*注：除特殊标明外，以下函数均支持回测和实盘/模拟运行模式。

(1) 设定股票池 ContextInfo.set_universe()

用法： ContextInfo.set_universe(stocklist)

释义： 设定股票池

参数： list

返回： 无

示例：

```
1 | def init(ContextInfo):  
2 |     stocklist = ['000300.SH', '000004.SZ']  
3 |     ContextInfo.set_universe(stocklist)
```

(2) 设定交易账号 ContextInfo.set_account()

用法： ContextInfo.set_account(account)

释义： 设定交易账号，并将该账号用于之后的交易主推订阅。

***注：**

一、可多次调用以设置多个账号，应在init中进行设置完毕，init执行后再设置将不再订阅交易主推；

二、调用passorder传入账号为空时会使用最后一次设置的账号作为下单账号。

参数： string

返回： 无

示例：

```
1 | def init(ContextInfo):  
2 |     account = '6000000223'  
3 |     ContextInfo.set_account(account)
```

(3) 设定回测起止时间 ContextInfo.start / ContextInfo.end

用法: ContextInfo.start / ContextInfo.end

释义: 设定回测起止时间, 标准格式如"2009-07-14 01:13:30", 读写

*注:

- 一、此函数只支持回测模式;
- 二、仅在init中设置生效, 应在init中设置完毕;
- 三、缺省值为策略编辑界面设定的回测时间范围;
- 四、回测起止时间也可在策略编辑器的回测参数面板中设置, 若两处同时设置, 则以代码中设置的值为准;
- 五、结束时间小于等于开始时间则计算范围为空。

参数: 无

返回: 无

示例:

```
1 | def init(ContextInfo):  
2 |     ContextInfo.start = '2012-06-06 10:00:00'  
3 |     ContextInfo.end = '2013-08-08 14:30:00'
```

(4) 设定回测初始资金 ContextInfo.capital

用法: ContextInfo.capital

释义: 设定回测初始资金, 读写, 默认为 1000000

*注: 此函数只支持回测模式。回测初始资金也可在策略编辑器的回测参数面板中设置, 若两处同时设置, 则以代码中设置的值为准。

参数: 无

返回: number

示例:

```
1 | def init(ContextInfo):  
2 |     ContextInfo.capital = 10000000  
3 |  
4 | def handlebar(ContextInfo):  
5 |     print(ContextInfo.capital)
```

(5) 设定策略回测滑点 ContextInfo.set_slippage()

用法: ContextInfo.set_slippage(slippageType, slippage)

释义: 设定策略回测滑点, 默认值 0.00

*注: 此函数只支持回测模式。回测滑点也可在策略编辑器的回测参数面板中设置, 若两处同时设置, 则以代码中设置的值为准。

参数:

- slippageType: 滑点类型, 可选值:
 - 0: tick跳数设置滑点
 - 1: 按照固定值 (价格) 设置滑点
 - 2: 价格比例设置滑点
- slippage: 滑点值

返回: 无

示例:

```
1 | def init(ContextInfo):  
2 |     # 按照固定值 (价格) 设置滑点值为 0.01  
3 |     ContextInfo.set_slippage(1, 0.01)
```

(6) 设定策略回测各种手续费率 ContextInfo.set_commission()

用法: ContextInfo.set_commission(commissionType, commissionList)

释义: 设定策略回测各种手续费率, 默认类型值 0 按比例, 默认值 0.000

*注: 此函数只支持回测模式。回测各种手续费率也可在策略编辑器的回测参数面板中设置, 若两处同时设置, 则以代码中设置的值为准。

参数:

- commissionType: number, 可选值:
 - 0: 按比例
 - 1: 按每手 (股)
- commissionList: list, 包含六个值, commissionList = [open_tax, close_tax, open_commission, close_commission, close_tdaycommission, min_commission]
 - open_tax: 买入印花税
 - close_tax: 卖出印花税
 - open_commission: 开仓手续费;
 - close_commission: 平仓 (平昨) 手续费
 - close_tdaycommission: 平今手续费
 - min_commission: 最少手续费

*注: 如果只填写一个参数则代表输入的参数值赋值给 open_commission = close_commission = close_today_commission, 其他的值均为 0, 这时 commissionType 为 0

返回: 无

示例1:

```
1 | def init(ContextInfo):  
2 |     # 万三  
3 |     commission = 0.0003  
4 |  
5 |     # 设定开仓手续费和平仓手续费以及平今手续费均为 0.0003, 其余为 0  
6 |     ContextInfo.set_commission(commission)
```

示例2:

```
1 | def init(ContextInfo):  
2 |     commissionList = [0,0.0001, 0.0003, 0.0003, 0, 5]  
3 |  
4 |     # 设定买入印花税为 0, 卖出印花税为 0.0001, 开仓手续费和平仓（平昨）手续费均为万三, 平  
5 |     ContextInfo.set_commission(0, commissionList)
```

(7) 获取股票池中的股票 ContextInfo.get_universe()

用法: ContextInfo.get_universe()

释义: 获取股票池中的股票

参数: 无

返回: list

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_universe())
```

(8) 获取当前周期 ContextInfo.period

用法: ContextInfo.period

释义: 获取当前周期, 即基本信息中设置的默认周期, 只读

参数: 无

返回: string, 返回值含义:

'1d': 日线

'1m': 1分钟线

'3m': 3分钟线

'5m': 5分钟线

'15m': 15分钟线

'30m': 30分钟线

'1h': 小时线

'1w': 周线

'1mon': 月线

'1q': 季线

'1hy': 半年线

'1y': 年线

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.period)
```

(9) 获取当前运行到 K 线索引号 ContextInfo.barpos

用法： ContextInfo.barpos

释义： 获取当前运行到 K 线索引号，只读，索引号从0开始

参数： 无

返回： number

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.barpos)
```

(10) 获取当前图 K 线数目 ContextInfo.time_tick_size

用法： ContextInfo.time_tick_size

释义： 获取当前图 K 线数目，只读

参数： 无

返回： number

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.time_tick_size)
```

(11) 判定是否为最后一根 K 线 ContextInfo.is_last_bar()

用法： ContextInfo.is_last_bar()

释义： 判定是否为最后一根 K 线

参数： 无

返回： bool，返回值含义：

True: 是

False: 否

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.is_last_bar())
```

(12) 判定是否为新的 K 线 ContextInfo.is_new_bar()

用法: ContextInfo.is_new_bar()

释义: 某根 K 线的第一个 tick 数据到来时, 判定该 K 线为新的 K 线

参数: 无

返回: bool, 返回值含义:

True: 是

False: 否

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.is_new_bar())
```

(13) 判定股票是否停牌 ContextInfo.is_suspended_stock()

用法: ContextInfo.is_suspended_stock(stockcode.market)

释义: 判定股票是否停牌

参数: 股票市场和代码

返回: bool, 返回值含义:

True: 停牌

False: 未停牌

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.is_suspended_stock('600004.SH'))
```

(14) 判定给定股票代码是否在指定的板块中 is_sector_stock()

用法: is_sector_stock(sectorname, market, stockcode)

释义: 判定给定股票代码是否在指定的板块中

参数:

- sectorname: string, 板块名
- market: string, 市场

- stockcode: string, 股票代码

返回: bool, 返回值含义:

True: 在板块中

False: 不在板块中

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(is_sector_stock('沪深300', 'SH', '600000'))
```

(15) 判定给定股票是否属于某个类别 is_typed_stock()

用法: is_typed_stock(stocktypenum, market, stockcode)

释义: 判定给定股票是否属于某个类别

参数:

- stocktypenum: number, 类别标号, 如股票类别标号为100003, 详细见[5.2. 附录2 is_typed_stock 函数证券分类表](#), 或者见 xtstocktype.lua 文件
- market: string, 市场
- stockcode: string, 股票代码

返回: number, 返回值含义:

1: True

0: False

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(is_typed_stock(100003, 'SH', '600000'))
```

(16) 判定给定股票代码是否在指定的行业分类中

get_industry_name_of_stock()

用法: get_industry_name_of_stock(industryType, stockcode)

释义: 判定给定股票代码是否在指定的行业分类中

参数:

- industryType: string, 行业类别, 有 'CSRC' 和 'SW' 两种
- stockcode: string, 形式如 'stockcode.market', 如 '600000.SH'

返回: string: 对应行业名, 找不到则返回空 string

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(get_industry_name_of_stock('SW', '600000.SH'))
```

(17) 获取当前图代码 ContextInfo.stockcode

用法: ContextInfo.stockcode

释义: 获取当前图代码, 只读

参数: 无

返回: string: 对应主图代码

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.stockcode)
```

(18) 获取当前主图复权处理方式 ContextInfo.dividend_type

用法: ContextInfo.dividend_type

释义: 获取当前主图复权处理方式

参数: 无

返回: string, 返回值含义:

'none': 不复权
'front': 向前复权
'back': 向后复权
'front_ratio': 等比向前复权
'back_ratio': 等比向后复权

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.dividend_type)
```

(19) 获取当前主图市场 ContextInfo.market

用法: ContextInfo.market

释义: 获取当前主图市场, 只读

参数: 无

返回: string: 对应主图市场

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.market)
```

(20) 根据代码获取名称 ContextInfo.get_stock_name()

用法: ContextInfo.get_stock_name('stockcode')

释义: 根据代码获取名称

参数: stockcode: 股票代码, 如'000001.SZ', 缺省值 '' 默认为当前图代码

返回: string (GBK编码)

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_stock_name('000001.SZ'))
```

(21) 根据代码返回对应股票的上市时间 get_open_date()

用法: get_open_date('stockcode')

释义: 根据代码返回对应股票的上市时间

参数: stockcode: 股票代码, 如'000001.SZ', 缺省值 '' 默认为当前图代码

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_open_date('000001.SZ'))
```

(22) 表示当前是否开启回测模式 ContextInfo.do_back_test

用法: ContextInfo.do_back_test

释义: 设定是否开启回测模式, 只读, 默认值为 False

参数: 无

返回: bool

(23) 获取回测基准 ContextInfo.benchmark

用法: ContextInfo.benchmark

释义: 获取回测基准, 只读

*注: 此函数只支持回测模式。

参数: 无

返回: string

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.benchmark)
```

(24) 设定回测系统输出日志显示级别 ContextInfo.data_info_level

用法: ContextInfo.data_info_level

释义: 设定回测系统输出日志显示级别, 默认是0, 可设置的值有: 0 信息, 1 警告, 2 错误, 3 致命, 根据设定显示大于等于该级别的日志

*注: 此函数只支持回测模式。

参数: 无

返回: 无

示例:

```
1 | def init(ContextInfo):  
2 |     # 显示'错误'级别以上的信息  
3 |     ContextInfo.data_info_level = 2
```

(25) 获取某个记录类型对应的某个时刻的记录情况 get_result_records()

用法: get_result_records (recordtype, index, ContextInfo)

释义: 获取某个记录类型对应的某个时刻的记录情况。

*注: 模型回测时有效, 获取的为回测面板中的记录结果

参数:

- Recordtype: string, 面板类型, 可选值:

```
'buys': 买入持仓  
'sells': 卖出持仓  
'holdings': 当前持仓  
'historysums': 历史汇总  
'dealdetails': 交易明细
```

- index: number, 当前主图对应 K 线索引
- ContextInfo: PythonObj, Python对象, 这里必须是ContextInfo

返回: list, 返回的 list 结构中包含 0 个或多个 Python 对象, 该 Python 对象内含如下属性值:

```
market: string, 市场代码  
stockcode: string, 合约代码  
open_close: number, 期货开平: 1 开 0 平; 股票: 1 买 0 卖  
direction: number, 方向: 1 多 -1 空; 回购: 1 逆回购 -1 正回购  
trade_price: number, 持仓成本
```

current_price: number, 最高价, 持仓中用这个价
profit: number, 持仓盈亏
position: number, 仓位数量
current_weight: number, 仓位权重[暂无有效数据]
benefit_weight: number, 盈利占比权重, 记录类型是 'historysums' 时有效
holding_periods: number, 累计持仓天数, 记录类型是 'historysums' 时有效
buy_sell_times: number, 交易次数, 记录类型是 'historysums' 时有效
commission: number, 手续费
trade_balance: number, 成交额或市值
operate_type: number, 操作类型, 记录类型是 'dealdetails' 时有效
trade_date: number, 交易日期, 毫秒标记, 记录类型是 'dealdetails' 时有效

示例:

```
1 def handlebar(ContextInfo):
2     index = ContextInfo.barpos
3     print(get_result_records('buys', index, ContextInfo))
```

(26) 设置定时器 run_time(funcName,period,startTime,market)

用法: run_time(funcName,period,startTime,market)

释义: 设置定时器

*注: 模型回测时无效, 定时器没有结束方法, 会随着策略的结束而结束。另外定时器函数在一次运行前会先等待一个period

参数:

- funcName: 回调函数名
- period: 重复调用的时间间隔,'5nSecond'表示每5秒运行1次回调函数,'5nDay'表示每5天运行一次回调函数,'500nMillisecond'表示每500毫秒运行1次回调函数
- startTime: 表示定时器第一次启动的时间,如果要定时器立刻启动,可以设置历史的时间

回调函数参数: ContextInfo: 策略模型全局对象

示例:

```
1 import time
2 def init(ContextInfo):
3     ContextInfo.run_time("myHandlebar", "5nSecond", "2019-10-14 13:20:00", "SH")
4 def myHandlebar(ContextInfo):
5     print('hello world')
6 def handlebar(ContextInfo):
7     pass
8 #此例为自2019-10-14 13:20:00后每5s运行一次myHandlebar
```

(27) 停止处理函数 stop()

用法: stop(ContextInfo)

释义: PY策略模型关闭停止前运行到的函数，复杂策略模型，如中间有起线程可通过在该函数内实现停止线程操作。

*注：PY策略可不用该函数

参数: ContextInfo: 策略模型全局对象

示例:

```
1 | def stop(ContextInfo):
2 |     print('strategy is stop !')
```

3.2.3. 获得数据

*注：除特殊标明外，以下函数均支持回测和实盘/模拟运行模式。

(1) 获取最新流通股本 ContextInfo.get_last_volume()

用法: ContextInfo.get_last_volume(stockcode)

释义: 获取最新流通股本

参数: string: 必须是 'stock.market' 形式

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_last_volume('000002.sz'))
```

(2) 获取当前 K 线对应时间的时间戳 ContextInfo.get_bar_timetag()

用法: ContextInfo.get_bar_timetag(index)

释义: 获取当前 K 线对应时间的时间戳

参数: number: K 线索引号

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     index = ContextInfo.barpos
3 |     print(ContextInfo.get_bar_timetag(index))
```

(3) 获取当前主图品种最新分笔对应的时间的时间戳 ContextInfo.get_tick_timetag()

用法: ContextInfo.get_tick_timetag()

释义: 获取当前主图品种最新分笔对应的时间的时间戳

参数: 无

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_tick_timetag())
```

(4) 获取指数成份股 ContextInfo.get_sector()

用法: ContextInfo.get_sector(sector, realtime)

释义: 获取板块成份股，只支持取指数成份股

参数:

- string: 必须是 'stock.market' 形式, 如 '000300.SH' , 可取如沪深300 (000300.SH) 、中证500 (000905.SH) 、上证50 (000016.SH) 等指数的历史成份股
- realtime: 毫秒级时间戳, 如不填则默认取目前最新成份股

返回: list: 内含成份股代码, 里面股票代码为 '000002.SZ' 形式

示例:

```
1 | def handlebar(ContextInfo):
2 |     index = ContextInfo.barpos
3 |     realtime = ContextInfo.get_bar_timetag(index)
4 |     print(ContextInfo.get_sector('000300.SH', realtime))
```

(5) 获取行业成份股 ContextInfo.get_industry()

用法: ContextInfo.get_industry(industry)

释义: 获取行业成份股

参数: string: 如 'CSRC1矿业'

返回: list: 内含成份股代码, 里面股票代码为 '000002.SZ' 形式

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_industry('CSRC1采矿业'))
```

(6) 获取板块成份股 ContextInfo.get_stock_list_in_sector()

用法: ContextInfo.get_stock_list_in_sector(sectorname, realtime)

释义: 获取板块成份股，支持客户端左侧板块列表中任意的板块，包括自定义板块

参数:

- string: 板块名, 如'沪深300', '中证500'、'上证50'、'我的自选'等
- realtime: 毫秒级时间戳

返回: list: 内含成份股代码, 代码形式为 'stockcode.market', 如 '000002.SZ'

示例:

```
1 | def handlebar(ContextInfo):  
2 |     index = ContextInfo.barpos  
3 |     realtime = ContextInfo.get_bar_timetag(index)  
4 |     print(ContextInfo.get_stock_list_in_sector('沪深300', realtime))
```

(7) 获取某只股票在某指数中的绝对权重 ContextInfo.get_weight_in_index()

用法: ContextInfo.get_weight_in_index(indexcode, stockcode)

释义: 获取某只股票在某指数中的绝对权重

参数:

- indexcode: string, 指数代码, 形式如 'stockcode.market', 如 '000300.SH'
- stockcode: string, 股票代码, 形式如 'stockcode.market', 如 '600004.SH'

返回: number: 返回的数值单位是 %, 如 1.6134 表示权重是 1.6134%

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_weight_in_index('000300.SH', '000002.SZ'))
```

(8) 获取合约乘数 ContextInfo.get_contract_multiplier()

用法: ContextInfo.get_contract_multiplier(contractcode)

释义: 获取合约乘数

参数: string: 合约代码, 形式如 'code.market', 如 'IF1707.IF'

返回: number

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_contract_multiplier('IF1707.IF'))
```

(9) 获取无风险利率 ContextInfo.get_risk_free_rate()

用法: ContextInfo.get_risk_free_rate(index)

释义: 获取无风险利率, 用十年期国债收益率CGB10Y作无风险利率

参数: number: 当前图 K 线索引号

返回: number

示例:

```
1 def handlebar(ContextInfo):
2     index = ContextInfo.barpos
3     print(ContextInfo.get_risk_free_rate(index))
```

(10) 获取给定日期对应的 K 线索引号 ContextInfo.get_date_location()

用法: ContextInfo.get_date_location(strdate)

释义: 获取给定日期对应的 K 线索引号, 如给定日期小于当前图 K 线对应的最早的日期, 结果返回 0; 如给定日期大于当前图 K 线对应的最新日期, 结果返回最新 K 线的索引号

参数: string: 形式如 'yyyymmdd' , 如 '20170711'

返回: number

示例:

```
1 def handlebar(ContextInfo):
2     print(ContextInfo.get_date_location('20170711'))
```

(11) 获取策略设定的滑点 ContextInfo.get_slippage()

用法: ContextInfo.get_slippage()

释义: 获取策略设定的滑点

*注: 此函数只支持回测模式。

参数: 无

返回: dict, key 包括:

```
slippage_type
slippage
```

示例:

```
1 def init(ContextInfo):
2     ContextInfo.set_slippage(0.003)
3
4 def handlebar(ContextInfo):
5     print(ContextInfo.get_slippage())
```

(12) 获取策略设定的各种手续费率 ContextInfo.get_commission()

用法: ContextInfo.get_commission()

释义: 获取策略设定的各种手续费率

*注: 此函数只支持回测模式。

参数: 无

返回: dict, key 包括

```
commission_type  
open_tax  
close_tax  
open_commission  
close_commission  
close_tdaycommission  
min_commission
```

示例:

```
1 def init(ContextInfo):  
2     ContextInfo.set_commission(0.0003)  
3  
4 def handlebar(ContextInfo):  
5     print(ContextInfo.get_commission())
```

(13) 获取策略回测的净值 ContextInfo.get_net_value()

用法: ContextInfo.get_net_value(index)

释义: 获取策略回测的净值

*注: 此函数只支持回测模式。

参数: index = ContextInfo.barpos

返回: number

示例:

```
1 def handlebar(ContextInfo):  
2     index = ContextInfo.barpos  
3     print(ContextInfo.get_net_value(index))
```

(14) 获取财务数据 ContextInfo.get_financial_data()

用法:

```
ContextInfo.get_financial_data(fieldList,stockList,startDate,endDate,report_type='announce_time')
```

释义: 获取财务数据

参数:

- fieldList: 字段列表, 如 ['CAPITALSTRUCTURE.total_capital', 'ASHAREINCOME.net_profit_incl_min_int_inc'] (例子中取了股本结构中的总股本, 与利润表中的净利润), 更多支持字段参见[4. 财务数据接口使用方法](#)
- stockList: 股票列表, 如 ['600000.SH', '000001.SZ']
- startDate: 开始时间, 如 '20171209'
- endDate: 结束时间, 如 '20171212'
- report_type: 时间类型, 可缺省, 默认是按照数据的公告期为区分取数据, 设置为'report_time'为按照报告期取数据, 可选值:'announce_time','report_time'

返回:

函数根据stockList代码列表,startDate,endDate时间范围的大小范围不同的数据类型

- (1)代码列表1-时间范围为1返回: pandas.Series index=字段
 - (2)代码列表1-时间范围为n返回: pandas.DataFrame index=时间,columns=字段
 - (3)代码列表n-时间范围为1返回: pandas.DataFrame index=代码,columns=字段
 - (4)代码列表n-时间范围为n返回: pandas.Panel items=代码,major_axis=时间,minor_axis=字段
- 选择按照公告期取数和按照报告期取数的区别:若某公司当年4月26日发布上年度年报,如果选择按照公告期取数,则当年4月26日之后至下个财报发布日期之间的
数据都是上年度年报的财务数据,如果选择按照报告期取数,则上年度第4季度(上年度10月1日-12月31日)的数据就是上年度报告期的数据.代码1-时间1: pandas.Series index = 字段

*注: 必须安装pandas

示例:

```
1 def handlebar(ContextInfo):  
2     # 取股本结构中的总股本, 与利润表中的净利润  
3     fieldList = ['CAPITALSTRUCTURE.total_capital',  
4                  'ASHAREINCOME.net_profit_incl_min_int_inc']  
5     stockList = ['600000.SH', '000001.SZ']  
6     startDate = '20171209'  
7     endDate = '20171212'  
8     # 获取20171209-20171212时间段浦发银行和平安银行的总股本及利润表的净利润  
9     ContextInfo.get_financial_data(fieldList, stockList, startDate, endDate)
```

(15) 获取财务数据 ContextInfo.get_financial_data()

用法:

```
ContextInfo.get_financial_data(tabname,colname,market,code,report_type='report_time',barpos)  
(与上一个 ContextInfo.get_financial_data 可同时使用)
```

释义: 获取财务数据

参数:

- tablename: 表格名称
- colname: 字段名称
- market: 市场
- code: 股票代码
- barpos: 当前 bar 的索引
- report_type: 时间类型, 可缺省, 默认是按照数据的公告期为区分取数据, 设置为'report_time'为按照报告期取数据, 可选值:'announce_time', 'report_time'

返回: number

示例:

```
1 | def handlebar(ContextInfo):  
2 |     index = ContextInfo.barpos  
3 |     # 获取当前时间点浦发银行利润表的净利润, 更多支持字段参见[财务数据接口使用方法]  
4 |     ContextInfo.get_financial_data('ASHAREINCOME',  
5 |         'net_profit_incl_min_int_inc', 'SH', '600000', index)
```

(16) 获取历史行情数据 ContextInfo.get_history_data()

用法: ContextInfo.get_history_data(len, period, field, dividend_type = 0, skip_paused = True)

释义: 获取历史行情数据

*注: 必须先通过 ContextInfo.set_universe() 设定基础股票池, 获取历史行情数据是获取的是股票池中的历史行情数据。

参数:

- len: number, 需获取的历史数据长度
- period: string, 需获取的历史数据的周期, 可选值:

```
'tick': 分笔线  
'1d': 日线  
'1m': 1分钟线  
'3m': 3分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'30m': 30分钟线  
'1h': 小时线  
'1w': 周线  
'1mon': 月线  
'1q': 季线  
'1hy': 半年线  
'1y': 年线
```

- field: string, 需获取的历史数据的类型, 可选值:

```
'open'
```

```
'high'  
'low'  
'close'  
'quoter' (结构见get_market_data)
```

- dividend_type: 默认参数, number, 除复权, 默认不复权, 可选值:

```
0: 不复权  
1: 向前复权  
2: 向后复权  
3: 等比向前复权  
4: 等比向后复权
```

- skip_paused: 默认参数, bool, 是否停牌填充, 默认填充

*注: 可缺省参数: dividend_type, skip_paused

返回: 一个字典dict结构, key 为 stockcode.market, value 为行情数据 list, list 中第 0 位为最早的价格, 第 1 位为次早价格, 依次下去。

示例:

```
1 def init(ContextInfo):  
2     ContextInfo.set_universe(['000300.SH', '000004.SZ'])  
3  
4 def handlebar(ContextInfo):  
5     # 获取股票池中所有股票的最近两日的收盘价  
6     hisdict = ContextInfo.get_history_data(2, '1d', 'close')  
7     for k, v in hisdict.items():  
8         if len(v) > 1:  
9             # 今日涨幅  
10            print(k, ':', v[1] - v[0])
```

(17) 获取行情数据 ContextInfo.get_market_data()

用法: ContextInfo.get_market_data(fields, stock_code = [], start_time = "", end_time = "", skip_paused = True, period = 'follow', dividend_type = 'follow', count = -1)

释义: 获取行情数据

参数:

- fields: 字段列表:

```
'open': 开  
'high': 高  
'low': 低  
'close': 收  
'volume': 成交量  
'amount': 成交额  
'settle': 结算价
```

```
'quoter': 分笔数据 (包括历史)
    ○ 'quoter'分笔数据结构: dict
    {
        lastPrice: 最高价
        amount: 成交额
        volume: 成交量
        pvolumn: 前成交量
        openInt: 持仓量
        stockStatus: 股票状态
        lastSettlementPrice: 最新结算价
        open: 开盘价
        high: 最高价
        low: 最低价
        settlementPrice: 结算价
        lastClose: 收盘价
        askPrice: 列表, 卖价五档
        bidPrice: 列表, 买价五档
        askVol: 列表, 卖量五档
        bidVol: 列表, 买量五档
    }
```

- stock_code : 默认参数, 合约代码列表, 合约格式 code.market, 如 '600000.SH', 不指定时为当前图合约

- start_time: 默认参数, 开始时间, 格式 '20171209' 或 '20171209010101'
- end_time: 默认参数, 结束时间, 格式 '20171209' 或 '20171209010101'
- skip_paused: 默认参数, 可选值:

```
true: 如果是停牌股, 会自动填充未停牌前的价格作为停牌日的价格  
False: 停牌数据为nan
```

- period: 默认参数, 周期类型:

```
'tick': 分笔线  
'1d': 日线  
'1m': 1分钟线  
'3m': 3分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'30m': 30分钟线  
'1h': 小时线  
'1w': 周线  
'1mon': 月线
```

'1q': 季线

'1hy': 半年线

'1y': 年线

- dividend_type: 默认参数, 缺省值为 'none', 除复权, 可选值:

'none': 不复权

'front': 向前复权

'back': 向后复权

'front_ratio': 等比向前复权

'back_ratio': 等比向后复权

- count: 默认参数, 缺省值为 -1, 大于等于 0 时, 效果与 get_history_data 保持一致。

count 和开始时间、结束时间同时设置的效果如下表:

count 取值	时间设置是否生效	开始时间和结束时间设置效果
count >= 0	生效	返回数量取决于开始时间与结束时间和count与结束时间的交集
count = -1	生效	同时设置开始时间和结束时间, 在所设置的时间段内取值
count = -1	生效	开始时间结束时间都不设置, 取当前最新bar的值
count = -1	生效	只设置开始时间, 取所设开始时间到当前时间的值
count = -1	生效	只设置结束时间, 取股票上市第一根 bar 到所设结束时间的值

*注:

1. 特别的, 默认参数需带上参数名方可生效, 调用方式与 Python 别无二致
2. 策略点击运行, 以策略代码中所设置的开始时间和结束时间为准, 策略点击回测, 以 '回测参数' 里所设置的开始时间和结束时间为准。
3. 可缺省参数: start_time, end_time, skip_paused, dividend_type, count
4. 传入count和end_time的组合时, 返回的行情数量为以end_time为结束时间且数量为count根

返回:

count	字段数量	股票数量	时间点	返回类型
=-1	=1	=1	=1	float
=-1	>1	=1	默认值	pandas.Series
>=-1	>=1	=1	>=1	pandas.DataFrame(字段数量和时间点不同时为1)
=-1	>=1	>1	默认值	pandas.DataFrame
>1	=1	=1	=1	pandas.DataFrame
>=-1	>=1	>1	>=1	pandas.Panel

*注 时间点是指start_time和end_time组成的时间区间， 默认值是指start_time和end_time均传入默认值

count ≥ 0 ，不同的参数，返回的数据结构不同，以下举例说明了不同的数据结构的类型。（代码指股票代码；时间即为函数中所设置的时间，主要由 count、start_time、end_time 决定；字段即 fields 里的字段）

(1) 1支股票代码，在1个时间点（count 缺省默认为 -1，即为当前时间点的 bar），取1个字段的值，则返回相应字段的值。

代码：

```

1 def init(ContextInfo):
2     ContextInfo.set_universe(['000831.SZ'])
3
4 def handlebar(ContextInfo):
5     df = ContextInfo.get_market_data(['close'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type
= 'front')
6     print(df)

```

输出：

17.40

(2) 1支股票代码，在一个时间点（count、start_time、end_time 都缺省，即为当前时间点的 bar），取多个字段的值，返回 pandas.Series (pandas 一维数组) 类型的值。

代码：

```
1 def init(ContextInfo):
2     ContextInfo.set_universe(['000831.SZ'])
3
4 def handlebar(ContextInfo):
5     df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type
= 'front')
6     print(df)
```

输出：

```
close 17.40
```

```
open 17.11
```

(3) 1支股票代码，在一个时间段取值，返回 pandas.DataFrame (pandas 二维表格型数据结构) 类型的值。

代码：

```
1 def init(ContextInfo):
2     ContextInfo.set_universe(['000831.SZ'])
3
4 def handlebar(ContextInfo):
5     df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type
= 'front', count = 2)
6     print(df)
```

输出：

```
close open
```

```
20190618 17.59 17.60
```

```
20190619 17.40 17.11
```

(4) 多个股票代码，在一个时间点取值，返回 pandas.DataFrame (pandas 二维表格型数据结构) 类型的值。

代码：

```
1 def init(ContextInfo):
2     ContextInfo.set_universe(['000831.SZ', '600000.SH'])
3
4 def handlebar(ContextInfo):
5     df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type
= 'front', count = -1)
6     print(df)
```

输出：

```
close open
```

```
000831.SZ 17.40 17.11
```

```
600000.SH 11.88 12.04
```

(5) 多支股票代码，在一个时间段取值，返回 pandas.Panel (pandas 三维数组结构) 类型的值。

代码：

```
1 def init(ContextInfo):
2     ContextInfo.set_universe(['000831.SZ', '600000.SH'])
3
4 def handlebar(ContextInfo):
5     df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type
= 'front', count = 2)
6     print(df)
```

输出：

```
<class 'pandas.core.panel.Panel'>

Dimensions: 2 (items) x 2 (major_axis) x 2 (minor_axis)

Items axis: 000831.SZ to 600000.SH

Major_axis axis: 20190618 to 20190619

Minor_axis axis: close to open
```

*注：

1. 需要下载历史数据；
2. pandas相关介绍详见pandas官方文档；
3. Python需要安装pandas；
4. 时间区间：两边闭区间。
5. get_history_data() 和 get_market_data() 两者区别：get_history_data() 是获取设定的股票池中股票某段行情，主要用来构建简单的指标；get_market_data() 可获取任意股票行情，返回行情数据可以是 dataframe，可用于复杂的分析。

示例1：

```
1 def handlebar(ContextInfo):
2     # 以 period 参数指定需要的周期
3     Result = ContextInfo.get_market_data(['open', 'high', 'low', 'close'],
['600000.SH'], '20170101', '20170102', True, '1d', 'none')
4     d_m = ContextInfo.get_market_data(['close'], ['600000.SH'], start_time =
'20181230', end_time = '20190107', period = '1d')
5     print(d_m)
6     m_m = ContextInfo.get_market_data(['close'], ['600000.SH'], start_time =
'20181230', end_time = '20190107', period = '1m')
7     print(m_m)
```

示例2：

```
1 def init(ContextInfo):
```

```

2     # 先设置股票池, 此时股票池只有一个股票
3     ContextInfo.set_universe(['000001.SZ'])
4
5     def handlebar(ContextInfo):
6         # 获取平安银行当前 bar 的收盘价
7         close = ContextInfo.get_market_data(['close'], stock_code =
ContextInfo.get_universe(), period = '1d', dividend_type = 'front', count =
1)
8
9         # 打印平安银行当前bar的收盘价
10        print(close) # close 是一个数值
11
12        # 获取平安银行最新的开高低收四个价格
13        df1 = ContextInfo.get_market_data(['open', 'high', 'low', 'close'],
stock_code = ContextInfo.get_universe(), period = '1d', dividend_type =
'front', count = 1)
14
15        # 打印平安银行当前bar的最高价
16        print(df1['high']) # df1 是一个 pandas.series
17
18        # 获取平安银行最近两天的高开低收
19        df2 = ContextInfo.get_market_data(['open', 'high', 'low', 'close'],
stock_code = ContextInfo.get_universe(), period = '1d', dividend_type =
'front', count = 2)
20
21        # 打印平安银行昨天的收盘价
22        print(df2['close'][-2]) # df2 是一个 pandas.dataframe

```

(18) 获取分笔数据 ContextInfo.get_full_tick()

用法: ContextInfo.get_full_tick(stock_code=[])

释义: 获取最新分笔数据, 该函数只能取最新的分笔, 不能取历史分笔

参数:

- stock_code: number, 默认参数, 合约代码列表, 如['600000.SH','600036.SH'], 不指定时为当
前主图合约。

返回: 根据stock_code返回一个dict, 该字典的key值是股票代码, 其值仍然是一个dict, 在该dict中存
放股票代码对应的数据。该字典数据key值有:

```

lastPrice:最新价
amount:成交额
volume:成交量
pvolume:前成交量
openInt:持仓量
stockStatus:股票状态
lastSettlementPrice:最新结算价
open:开盘价
high:最高价

```

low:最低价
settlementPrice:结算价
lastClose:收盘价
askPrice:列表,卖价五档
bidPrice:列表,买价五档
askVol:列表,卖量五档
bidVol:列表,买量五档

示例:

```
1 def handlebar(ContextInfo):
2     if not ContextInfo.is_last_bar():
3         return
4     Result=ContextInfo.get_full_tick(['600000.SH', '000001.SZ'])
```

(19) 获取除权除息日和复权因子 ContextInfo.get_divid_factors()

用法: ContextInfo.get_divid_factors(stock.market)

释义: 获取除权除息日和复权因子

参数: stock.market: 股票代码.市场代码, 如 '600000.SH'

返回: dict, 除权除息日与复权因子的映射字典

示例:

```
1 def handlebar(ContextInfo):
2     Result = ContextInfo.get_divid_factors('600000.SH')
```

(20) 获取当前期货主力合约 ContextInfo.get_main_contract()

用法: ContextInfo.get_main_contract(codemarket)

释义: 获取当前期货主力合约

参数: codemarket: 合约和市场, 合约格式为品种名加00, 如IF00.IF, zn00.SF

返回: str, 合约代码

示例:

```
1 def handlebar(ContextInfo):
2     print(ContextInfo.get_main_contract('IF00.IF'))
```

(21) 将毫秒时间转换成日期时间 timetag_to_datetime()

用法: timetag_to_datetime(timetag, format)

释义: 将毫秒时间转换成日期时间

参数:

- timetag: 毫秒时间, 1512748800000
- Format: 格式字符串, '%Y-%m-%d %H:%M:%S' 等任意格式

返回: str, 合约代码

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(timetag_to_datetime(1512748860000, '%Y%m%d %H:%M:%S'))
```

(22) 获取总股数 ContextInfo.get_total_share()

用法: ContextInfo.get_total_share(stockcode)

释义: 获取总股数

参数: stockcode: string, 股票代码, 缺省值 "", 默认为当前图代码, 如: '600000.SH'

返回: number

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_total_share('600000.SH'))
```

(23) 获取指定个股 / 合约 / 指数的 K 线 (交易日) 列表

ContextInfo.get_trading_dates()

用法: ContextInfo.get_trading_dates(stockcode, start_date, end_date, count, period)

释义: 获取指定个股 / 合约 / 指数的 K 线 (交易日) 列表

*注: 该接口在init函数里不可用

参数:

- stockcode: string, 股票代码, 缺省值 "", 默认为当前图代码, 如 '600000.SH'
- start_date: string, 开始时间, 缺省值 "", 为空时不使用, 如 '20170101', '20170101000000'
- end_date: string, 结束时间, 缺省值 "", 默认为当前bar的时间, 如 '20170102', '20170102000000'
- count: int, K 线个数, 必须大于 0, 取包括 end_date 往前的 count 个 K 线, start_date 不为空时不使用
- period: string, K 线类型如下:

'1d': 日线

'1m': 1分钟线

```
'3m': 3分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'30m': 30分钟线  
'1h': 小时线  
'1w': 周线  
'1mon': 月线  
'1q': 季线  
'1hy': 半年线  
'1y': 年线
```

*注：可缺省参数：start_date, end_date

返回：List，K线周期（交易日）列表，period为日线时返回['20170101', '20170102', ...]，其它返回['20170101010000', '20170102020000', ...]

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_trading_dates('600000.SH', '20170101', '20170401',  
1, '1d'))
```

(24) 根据代码获取对应股票的内盘成交量 ContextInfo.get_svol()

用法：ContextInfo.get_svol(stockcode)

释义：根据代码获取对应股票的内盘成交量

参数：stockcode：股票代码，如'000001.SZ'，缺省值"，默认为当前图代码

返回：string

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_svol('000001.sz'))
```

(25) 根据代码获取对应股票的外盘成交量 ContextInfo.get_bvol()

用法：ContextInfo.get_bvol(stockcode)

释义：根据代码获取对应股票的外盘成交量

参数：stockcode：股票代码，如'000001.SZ'，缺省值"，默认为当前图代码

返回：string

示例：

```
1 | def handlebar(ContextInfo):  
2 |     print(ContextInfo.get_bvol('000001.sz'))
```

(26) 获取龙虎榜数据 ContextInfo.get_longhubang()

用法: ContextInfo.get_longhubang(stock_list, startTime, endTime)

释义: 获取龙虎榜数据

参数:

stock_list: 股票列表, list, 如 ['600000.SH', '600036.SH']

startTime: 起始时间, 如 '20170101'

endTime: 结束时间, 如 '20180101'

返回: Dataframe, 字段:

reason: 上榜原因

close: 收盘价

spreadRate: 涨跌幅

TurnoverVolune: 成交量

Turnover_Amount: 成交金额

buyTraderBooth: 买方席位,datframe

sellTraderBooth: 卖方席位,datframe

- buyTraderBooth 或 sellTraderBooth 包含字段:

traderName: 交易营业部名称

buyAmount: 买入金额

buyPercent: 买入金额占总成交占比

sellAmount: 卖出金额

sellPercent: 卖出金额占总成交占比

totalAmount: 该席位总成交金额

rank: 席位排行

direction: 买卖方向

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ContextInfo.get_longhubang(['000002.sz'], '20100101', '20180101'))
```

(27) 获取十大股东数据 ContextInfo.get_top10_share_holder()

用法: get_top10_share_holder(self, stock_list, data_name, start_time, end_time)

释义: 获取十大股东数据

参数:

- data_name: flow_holder 或 holder

- stock_list: 股票列表, list, 如['600000.SH', '600036.SH']
- startTime: 起始时间, 如 '20170101'
- endTime: 结束时间, 如 '20180101'

返回:

series (一只股票一个季度)
dataframe (多只股票一个季度数据或者一只股票多个季度数据)
panel (多只股票多个季度)

- 字段

holdName: 股东名称
holderType: 持股类型
holdNum: 持股数量,
changReason: 变动原因
holdRatio: 持股比例
stockType: 股份性质
rank: 持股排名
status: 持股状态
changNum: 增减数量
changeRatio: 增减比例

示例:

```
1 def handlebar(ContextInfo):  
2     print(ContextInfo.get_top10_share_holder(['000002.SZ'], 'flow_holder',  
3 '20100101', '20180101'))
```

(28) 获取指定期权品种的详细信息

ContextInfo.get_option_detail_data(optioncode)

用法: ContextInfo.get_option_detail_data(optioncode)

释义: 获取指定期权品种的详细信息

参数:

- optioncode: 期权代码,如'10001506.SHO',当填写空字符串时候默认为当前主图的期权品种

返回:

dict

- 字段

ExchangeID:期权市场代码
InstrumentID:期权代码
ProductID:期权标的的产品ID
OpenDate:发行日期

ExpireDate:到期日
PreClose:前收价格
SettlementPrice:前结算价格
UpStopPrice:当日涨停价
DownStopPrice:当日跌停价
LongMarginRatio:多头保证金率
ShortMarginRatio:空头保证金率
PriceTick:最小变价单位
VolumeMultiple:合约乘数
MaxMarketOrderVolume:涨跌停价最大下单量
MinMarketOrderVolume:涨跌停价最小下单量
MaxLimitOrderVolume:限价单最大下单量
MinLimitOrderVolume:限价单最小下单量
OptUnit:期权合约单位
MarginUnit:期权单位保证金
OptUndlCode:期权标的证券代码
OptUndlMarket:期权标的证券市场
OptExercisePrice:期权行权价
NeeqExeType:全国股转转让类型
OptUndlRiskFreeRate:期权标的无风险利率
OptUndlHistoryRate:期权标的波动率
EndDelivDate:期权行权终止日

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print ContextInfo.get_option_detail_data('10001506.SH')
```

(29) 获取一篮子证券编码数据 ContextInfo.load_stk_list()

用法: ContextInfo.load_stk_list(filePath, fileName)

释义: 获取一篮子证券编码数据

txt文件信息格式: 600000.SH,600004.SH,600006.SH,或600000.SH600004.SH600006.SH(最后一个字段600006.SH需要增加分隔符)

csv文件信息格式: 600000.SH,600004.SH,600006.SH,(最后一个600006.SH需要增加分隔符)

参数:

- filePath: 文件路径
- fileName: 文件名(比如: a.txt,b.csv)

返回: 返回文档内容

示例:

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.load_stk_list('D:/data/','list.txt')
```

(30) 获取一篮子证券编码及数量数据ContextInfo.load_stk_vol_list()

用法: ContextInfo.load_stk_vol_list(filePath, fileName)

释义: 获取一篮子证券编码及数量数据

txt文件信息格式: 600000.SH,100,600004.SH,200,或600000.SH100600004.SH200(篮子数据成对存在(前面是合约代码, 后面是数量)且数量字段200后需要增加分隔符)

csv文件信息格式: 600000.SH,100,600004.SH,200,(篮子数据成对存在(前面是合约代码, 后面是数量)且数量字段200后需要增加分隔符)

参数:

- filePath: 文件路径
- fileName: 文件名(比如: a.txt,b.csv)

返回: 返回文档内容

示例:

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.load_stk_vol_list('D:/data/','list.txt')
```

(31) 获取本地行情数据 ContextInfo.get_local_data()

用法:

ContextInfo.get_local_data(stock_code,start_time='',end_time='',period='1d',divid_type='none',count=-1)

释义: 获取本地行情数据

参数:

- stock_code: 默认参数, 合约代码code.market不指定时为当前图合约
- start_time: 默认参数, 开始时间, 格式'20171209'或'20171209010101'
- end_time: 默认参数, 结束时间, 格式同start_time
- period: string, 默认参数, , K线类型, 可选值:

'tick': 分笔线 (只用于获取'quoter'字段数据)

'realtime': 实时线

'1d': 日线

'md': 多日线

'1m': 1分钟线

'3m': 3分钟线

'5m': 5分钟线

'15m': 15分钟线

'30m': 30分钟线
'mm': 多分钟线
'1h': 小时线
'mh': 多小时线
'1w': 周线
'1mon': 月线
'1q': 季线
'1hy': 半年线
'1y': 年线

- dividend_type: 默认参数, number, 除复权, 可选值:

'none': 不复权
'front': 向前复权
'back': 向后复权
'front_ratio': 等比向前复权
'back_ratio': 等比向后复权

- count: 默认参数, 大于等于0时, 若指定了start_time, end_time, 此时以end_time为基准向前取count条; 若start_time, end_time缺省, 默认取本地数据最新的count条数据; 若start_time, end_time, count都缺省时, 默认取本地全部数据。

特别的, 默认参数需带上参数名方可生效, 调用方式与python别无二致。

返回: 返回一个dict, 键值为timetag, value为另一个dict(valuedict)

- period='tick'时函数获取分笔数据, valuedict字典数据key值有:

lastPrice: 最新价
amount: 成交额
volume: 成交量
pvolume: 前成交量
openInt: 持仓量
stockStatus: 股票状态
lastSettlementPrice: 最新结算价
open: 开盘价
high: 最高价
low: 最低价
settlementPrice: 结算价
lastClose: 收盘价
askPrice: 列表,卖价五档
bidPrice: 列表,买价五档
askVol: 列表,卖量五档
bidVol: 列表,买量五档

- period为其他值时, valuedict字典数据key值有:

amount: 成交额
volume: 成交量
open: 开盘价
high: 最高价
low: 最低价
close: 收盘价

*注：时间区间两边闭区间

示例：

```
1 def handlebar(ContextInfo):  
2     Result=ContextInfo.get_local_data(stock_code='600000.SH',start_time='20170101',end_time='20170102',period='1d',divid_type='none')
```

(32) 获取换手率 ContextInfo.get_turnover_rate()

用法： ContextInfo.get_turnover_rate(stock_list,startTime,endTime)

释义： 获取换手率

参数：

- stock_list: 股票列表, list,如['600000.SH','000001.SZ']
- startTime: 起始时间, 如'20170101'
- endTime: 结束时间, 如'20180101'

返回： dataframe

示例：

```
1 def handlebar(ContextInfo):  
2     print(  
3         ContextInfo.get_turnover_rate(['000002.SZ'],'20170101','20180101'))
```

(33) 根据ETF基金代码获取ETF申赎清单及对应成分股数据 get_etf_info()

用法： get_etf_info(stockcode)

释义： 根据ETF基金代码获取ETF申赎清单及对应成分股数据

参数：

- stockcode: ETF基金代码 (如"510050.SH")

返回： 返回一个dict, 键值为timetag, value为另一个dict(valuedict)

- etfCode: ETF代码
- etfExchID: ETF市场
- prCode: 基金申赎代码
- cashBalance: 现金差额 (单位: 元)
- maxCashRatio: 现金替代比例上限
- reportUnit: 最小申购、赎回单位 (单位: 份)

- name: 基金名称
- navPerCU: 最小申购、赎回单位净值 (单位: 元)
- nav: 基金份额净值 (单位: 元)
- ecc: 预估现金差额 (单位: 元)
- needPublish: 是否需要公布IOPV (1: 是, 0: 否)
- enableCreation: 是否允许申购 (1: 是, 0: 否)
- enableRedemption: 是否允许赎回 (1: 是, 0: 否)
- creationLimit: 申购上限 (单位:份, 0: 不限制)
- redemptionLimit: 赎回上限 (单位:份, 0: 不限制)
- tradingDay: 交易日期 (格式YYYYMMDD)
- preTradingDay: 前交易日期 (格式YYYYMMDD)
- stocks: 成分股列表
- exchangeID: ETF基金市场代码
- etfCode: ETF基金代码
- etfName: ETF基金名称
- componentExchID: 成份股市场代码
- componentCode: 成份股代码
- componentName: 成份股名称
- componentVolume: 成份股数量
- ReplaceFlag: 替代标记(48: 禁止替代, 49: 允许替代, 50: 必须替代, 51: 替补替代)
- ReplaceRatio: 溢价比率
- ReplaceBalance: 替代金额

示例:

```
1 | get_etf_info("510050.SH")#获取ETF基金代码为511050的全部ETF申赎清单数据
```

(34) 根据ETF基金代码获取ETF的基金份额参考净值 get_etf_iopv()

用法: get_etf_iopv(stockcode)

释义: 根据ETF基金代码获取ETF的基金份额参考净值

参数:

- stockcode: ETF基金代码 (如"510050.SH")

返回: IOPV, 基金份额参考净值

示例:

```
1 | get_etf_iopv("510050.SH")
```

(35) 根据代码获取合约详细信息 ContextInfo.get_instrumentdetail()

用法: ContextInfo.get_instrumentdetail(stockcode)

释义: 根据代码获取合约详细信息

参数:

- stockcode: string, 股票代码,如'600000.SH'

返回: 根据stockcode返回一个dict。该字典数据key值有：

ExchangeID: 合约市场代码
InstrumentID: 合约代码
InstrumentName: 合约名称
ProductID: 合约的品种ID(期货)
ProductName: 合约的品种名称(期货)
CreateDate: 上市日期(期货)
OpenDate: IPO日期(股票)
ExpireDate: 退市日或者到期日
PreClose: 前收盘价格
SettlementPrice: 前结算价格
UpStopPrice: 当日涨停价
DownStopPrice: 当日跌停价
FloatVolumn: 流通股本
TotalVolumn: 总股本
LongMarginRatio: 多头保证金率
ShortMarginRatio: 空头保证金率
PriceTick: 最小变价单位
VolumeMultiple: 合约乘数(对期货以外的品种, 默认是1)
MainContract: 主力合约标记
LastVolume: 昨日持仓量
InstrumentStatus: 合约停牌状态
IsTrading: 合约是否可交易
IsRecent: 是否是近月合约

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print ContextInfo.get_instrumentdetail('600000.SH')
```

(36) 获取期货合约到期日 ContextInfo.get_contract_expire_date()

用法: ContextInfo.get_contract_expire_date(codemarket)

释义: 获取期货合约到期日

参数:

- Codemarket: 合约和市场,如IF00.IF,zn00.SF

返回: int, 合约到期日

示例:

```
1 | def handlebar(ContextInfo):
2 |     print( ContextInfo.get_contract_expire_date('IF00.IF'))
```

(37) 获取历史st状态 st_status()

用法: st_status(stockcode)

释义: 获取历史st状态, 本函数需要下载历史ST数据

参数:

- stockcode: 股票代码, 如000004.SZ (可为空, 为空时取主图代码)

返回:

0: 正常
1: ST
2: *ST
3: PT

示例:

```
1 | def handlebar(ContextInfo):
2 |     a:st_status('000004.SZ');
3 |     b:st_status('');
```

(38) 获取某只股票ST的历史ContextInfo.get_his_st_data()

用法: ContextInfo.get_his_st_data(stockcode)

释义: 获取某只股票ST的历史

参数:

- stockcode: string, 股票代码, 'stkcode.market', 如'000004.SZ'

返回: dict,st历史, key为ST,*ST,PT,历史未ST会返回{}

示例:

```
1 | def handlebar(ContextInfo):
2 |     print( ContextInfo.get_his_st_data('000004.SZ'))
```

(39) 获取某只指数历史调整日的历史的成分股列表

ContextInfo.get_his_index_data()

用法: ContextInfo.get_his_index_data(stockcode)

释义: 获取某只指数历史调整日的历史的成分股列表，需要下载历史数据

参数:

- stockcode: string, 股票代码, 'stkcode.market', 如'000300.SH'

返回: dict,{date:[stockList]};

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print( ContextInfo.get_his_index_data('000300.SH'))#获取沪深300函数历次调整时  
      的指数成分列表,计算对应权重可以进一步取历史股本进行计算
```

(40) 下载指定合约代码指定周期对应时间范围的行情数据down_history_data()

用法: down_history_data(stockcode,period,startTime,endTime)

释义: 下载指定合约代码指定周期对应时间范围的行情数据

参数:

- stockcode: string, 股票代码, 形式如'stkcode.market', 如'600000.SH'
- period: string, K线周期类型, 'tick': 分笔线, '1d': 日线, '1m': 分钟线, '5m': 5分钟线
- startTime, endTime: string, 起止时间, "20200101"或"20200101093000", 可以为空

示例:

```
1 | down_history_data("600000.SH", "1d", "", "")
```

(41) 订阅行情数据ContextInfo.subscribe_quote()

用法: ContextInfo.subscribe_quote(stockcode,period,dividend_type,callback)

释义: 订阅行情数据

参数:

- stockcode: string, 股票代码, 'stkcode.market', 如'600000.SH'
- period: string, K线周期类型, 'follow': 当前主图周期, 'tick': 分笔线, '1d': 日线, '1m': 分钟线, '5m': 5分钟线, (暂只支持分笔线周期)
- dividend_type: string, 除复权, 'follow': 当前图复权方式, 'none': 不复权, 'front': 向前复权, 'back': 向后复权, 'front_ratio': 等比向前复权, 'back_ratio': 等比向后复权, (分笔周期返回数据均为不复权)
- callback: 推送行情的回调

返回: 订阅号, 用于反订阅

示例:

```
1 | def quote_callback(s):
2 |     def callback(data):
3 |         print(s,data)
4 |         return
5 |     return callback
6 | def init(ContextInfo):
7 |
ContextInfo.subscribe_quote("600000.SH","tick","none",quote_callback('600000.SH'))
```

(42) 反订阅行情数据ContextInfo.unsubscribe_quote()

用法: ContextInfo.unsubscribe_quote(subId)

释义: 反订阅行情数据

参数:

- subId: 行情订阅返回的订阅号

示例:

```
1 | ContextInfo.unsubscribe_quote(subId)
```

(43) 获取当前所有的行情订阅信息ContextInfo.get_all_subscription()

用法: ContextInfo.get_all_subscription()

释义: 反订阅行情数据

返回: dict: "stockCode": 合约代码, "period": 周期, "dividendType": 除权方式

示例:

```
1 | data=ContextInfo.get_all_subscription()
```

(44) 获取行情数据 第二版 ContextInfo.get_market_data_ex()

用法: ContextInfo.get_market_data_ex(fields=[], stock_code=[], period='follow', start_time='', end_time='', count=-1, dividend_type='follow', fill_data=True, subscribe=True)

释义: 获取行情数据 第二版

参数:

- fields: list, 字段列表, 对不同周期的数据取值范围不同。默认为空list, 代表所有字段。
- stock_code: list, 合约代码列表
- period: string, 周期, 默认为'follow', 为当前主图周期, 可选值:

注: 行情数据字段列表见附录5.5

'tick': 分笔线

```
'1d': 日线  
'1m': 1分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'l2quote': Level2行情快照  
'l2quoteaux': Level2行情快照补充  
'l2order': Level2逐笔委托  
'l2transaction': Level2逐笔成交  
'l2transactioncount': Level2大单统计  
'l2orderqueue': Level2委买委卖队列
```

- start_time: 开始时间, 为空视为最早, 时间格式为'20201231'或'20201231093000'
- end_time: 结束时间, 为空视为最新, 时间格式为'20201231'或'20201231093000'
- count: 数据最大个数, -1视为不做个数限制
- dividend_type: 复权方式, 默认为'follow', 为当前主图复权方式, 可选值:

```
'none': 不复权  
'front': 前复权  
'back': 后复权  
'front_ratio': 等比前复权  
'back_ratio': 等比后复权
```

- fill_data: 停牌填充方式, 默认为True (暂不可用)
- subscribe: 订阅数据开关, 默认为True, 设置为False时不做数据订阅, 只读取本地已有数据。
 - 订阅模式下, 除分笔以外的周期只会自动补充当天数据, 分笔周期不自动补充数据。

返回: {code1: data1, code2: data2, ...}

示例:

```
1 | data=ContextInfo.get_market_data_ex(['open', 'high', 'low', 'close'],  
  ['000300.SH'], period='1d', start_time='', end_time='', count=-1, dividend_type='f  
ollow', fill_data=True, subscribe=True)
```

(45) 获取指定期权列表 ContextInfo.get_option_list()

用法: ContextInfo.get_option_list(undl_code, dedate, opttype, isavailable)

释义: 获取指定期权列表。如获取历史期权, 需先下载过期合约列表

参数:

- undl_code: 期权标的代码, 如'510300.SH'
- dedate: 期权到期月或当前交易日期, "YYYYMM"格式为期权到期月, "YYYYMMDD"格式为获取当前日期交易的期权
- opttype: 期权类型, 默认值为空, "CALL", "PUT", 为空时认购沽都取
- isavailable: 是否可交易, 当dedate的格式为"YYYYMMDD"格式为获取当前日期交易的期权时, isavailable为True时返回当前可用, 为False时返回当前和历史可用

返回: 期权合约列表list

示例:

```
1 data=ContextInfo.get_option_list('510300.SH','202101',"CALL")#获取到期月份为  
2 202101的上交所510300ETF认购合约  
3 data=ContextInfo.get_option_list('510300.SH','20210104',"CALL",True);获取  
20210104当天上交所510300ETF可交易的认购合约  
3 data=ContextInfo.get_option_list('510300.SH','20210104',"CALL",False);获取  
20210104当天上交所510300ETF已经上市的认购合约(包括退市)
```

(46) 获取股票期权的实时隐含波动率 ContextInfo.get_option_iv()

用法: ContextInfo.get_option_iv(optioncode)

释义: 获取股票期权的实时隐含波动率

参数:

- optioncode: 期权代码, 如'10003280.SHO'

返回: double

示例:

```
1 def handlebar(ContextInfo):  
2     print( ContextInfo.get_option_iv('10003280.SHO'))
```

(47) 基于BS模型计算欧式期权理论价格 ContextInfo.bsm_price()

用法: ContextInfo.bsm_price(optionType,objectPrices,strikePrice,riskFree,sigma,days,dividend)

释义: 基于Black-Scholes-Merton模型, 输入期权标的价格、期权行权价、无风险利率、期权标的年化波动率、剩余天数、标的分红率、计算期权的理论价格

参数:

- optionType: 期权类型, 认购: 'C', 认沽: 'P'
- objectPrices: 期权标的价格, 可以是价格列表或者单个价格
- strikePrice: 期权行权价
- riskFree: 无风险收益率
- sigma: 标的波动率
- days: 剩余天数
- dividend: 分红率

返回:

objectPrices为float时, 返回float

objectPrices为list时, 返回list

计算结果最小值0.0001, 结果保留4位小数, 输入非法参数返回nan

示例:

```

1 import numpy as np
2 object_prices=list(np.arange(3,4,0.01));
3 #计算剩余15天的行权价3.5的认购期权,在无风险利率3%,分红率为0,标的年化波动率为23%时标的价格
从3元到4元变动过程中期权理论价格序列
4 prices=ContextInfo.bsm_price('C',object_prices,3.5,0.03,0.23,15,0)
5 print(prices);
6 #计算剩余15天的行权价3.5的认购期权,在无风险利率3%,分红率为0,标的年化波动率为23%时标的价格
为3.51元的平值期权的理论价格
7 price=ContextInfo.bsm_price('C',3.51,3.5,0.03,0.23,15,0)
8 print(price);

```

(48) 基于BS模型计算欧式期权隐含波动率 ContextInfo.bsm_iv()

用法: ContextInfo.bsm_iv(optionType,objectPrices,strikePrice,optionPrice,riskFree,days,dividend)

释义: 基于Black-Scholes-Merton模型,输入期权标的价格、期权行权价、期权现价、无风险利率、剩余天数、标的分红率,计算期权的隐含波动率

参数:

- optionType: 期权类型, 认购: 'C', 认沽: 'P'
- objectPrice: 期权标的价格
- strikePrice: 期权行权价
- optionPrice: 期权现价
- riskFree: 无风险收益率
- days: 剩余天数
- dividend: 分红率

返回: double

示例:

```

1 iv=ContextInfo.bsm_iv('C',3.51,3.5,0.0725,0.03,15);
2 #计算剩余15天的行权价3.5的认购期权,在无风险利率3%,分红率为0时,标的现价3.51元,期权价格
0.0725元时的隐含波动率

```

(49) 取原始财务数据 ContextInfo.get_raw_financial_data()

用法:

ContextInfo.get_raw_financial_data(fieldList,stockList,startDate,endDate,report_type='announce_time')

释义: 取原始财务数据, 与get_financial_data相比不填充每个交易日的数据

参数:

- fieldList 字段列表, list, 如: ['资产负债表.固定资产','利润表.净利润']
- stockList: 股票列表, list, 如: ['600000.SH','000001.SZ']
- startDate: 开始时间, string, 如: '20171209'
- endDate: 结束时间, string, 如: '20171212'
- report_type: 时间类型, 可缺省, 默认是按照数据的公告期为区分取数据, 设置为'report_time'为按照报告期取数据, 可选值:'announce_time','report_time'

返回:

函数根据stockList代码列表,startDate,endDate时间范围的大小范围不同的数据类型

- (1)代码列表1-时间范围为1返回: pandas.Series index=字段
- (2)代码列表1-时间范围为n返回: pandas.DataFrame index=时间,columns=字段
- (3)代码列表n-时间范围为1返回: pandas.DataFrame index=代码,columns=字段
- (4)代码列表n-时间范围为n返回: pandas.Panel items=代码,major_axis=时间,minor_axis=字段

选择按照公告期取数和按照报告期取数的区别:若某公司当年4月26日发布上年度年报,如果选择按照公告期取数,则当年4月26日之后至下个财报发布日期之间的

数据都是上年度年报的财务数据,如果选择按照报告期取数,则上年度第4季度(上年度10月1日-12月31日)的数据就是上年度报告期的数据.

示例:

```
1 fieldList =
2     ['ASHAREBALANCESHEET.fix_assets', 'ASHAREINCOME.net_profit_incl_min_int_inc']
3 stockList = ['600000.SH', '000001.SZ']
4 startDate = '20200101'
5 endDate = '20210101'
6 ContextInfo.get_raw_financial_data(fieldList, stockList, startDate, endDate)
```

(50) 获取市场已退市合约 ContextInfo.get_his_contract_list()

用法: ContextInfo.get_his_contract_list(market)

释义: 获取市场已退市合约, 需要手动补充过期合约列表

参数:

- market:市场,SH,SZ,SHO,SZO,IF等

返回: list,合约代码列表

示例:

```
1 def handlebar(ContextInfo):
2     print( ContextInfo.get_his_contract_list('IF'))
3
4 def handlebar(ContextInfo):
5     print( ContextInfo.get_his_contract_list('SHO'))
```

(51) 获取指定期权标的对应的期权品种列表

ContextInfo.get_option_undl_data()

用法: ContextInfo.get_option_undl_data(undl_code_ref)

释义: 获取指定期权标的对应的期权品种列表

参数:

- undl_code_ref:期权标的代码,如'510300.SH', 传空字符串时获取全部标的的数据

返回:

指定期权标的代码时返回对应该标的的期权合约列表list

期权标的代码为空字符串时返回全部标的对应的品种列表的字典dict

示例：

```
1 | data=ContextInfo.get_option_undl_data('510300.SH')
```

(52) 获取对应周期的北向数据 ContextInfo.get_north_finance_change()

用法： ContextInfo.get_north_finance_change(period)

释义： 获取对应周期的北向数据

参数：

- period:1d日线数据1m一分钟数据

返回：

根据period返回一个dict，该字典的key值是北向数据的时间戳，其值仍然是一个dict，其值的key值是北向数据的字段类型，其值是对应字段的值。该字典数据key值有：

```
hgtNorthBuyMoney:HGT北向买入资金  
hgtNorthSellMoney:HGT北向卖出资金  
hgtSouthBuyMoney:HGT南向买入资金  
hgtSouthSellMoney:HGT南向卖出资金  
sgtNorthBuyMoney:SGT北向买入资金  
sgtNorthSellMoney:SGT北向卖出资金  
sgtSouthBuyMoney:SGT南向买入资金  
sgtSouthSellMoney:SGT南向卖出资金  
hgtNorthNetInFlow:HGT北向资金净流入  
hgtNorthBalanceByDay:HGT北向当日资金余额  
hgtSouthNetInFlow:HGT南向资金净流入  
hgtSouthBalanceByDay:HGT南向当日资金余额  
sgtNorthNetInFlow:SGT北向资金净流入  
sgtNorthBalanceByDay:SGT北向当日资金余额  
sgtSouthNetInFlow:SGT南向资金净流入  
sgtSouthBalanceByDay:SGT南向当日资金余额
```

示例：

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.get_north_finance_change('1d')
```

(53) 获取指定品种的持股统计 ContextInfo.get_hkt_statistics()

用法: ContextInfo.get_hkt_statistics(stockcode)

释义: 获取指定品种的持股统计

参数:

- stockcode: string, 必须是'stock.market'形式

返回:

根据stockcode返回一个dict, 该字典的key值是北向持股统计数据的时间戳, 其值仍然是一个dict, 其值的key值是北向持股统计数据的字段类型, 其值是对应字段的值, 该字典数据key值有:

stockCode:股票代码

ownSharesAmount:持股数量, 单位: 股

ownSharesMarketValue:持股市值, 单位: 元

ownSharesRatio:持股数量占比, 单位: %

ownSharesNetBuy:净买入, 单位: 元, 浮点数 (当日持股-前一日持股)

示例:

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.get_hkt_statistics('600000.SH')
```

(54) 获取指定品种的持股明细 ContextInfo.get_hkt_details()

用法: ContextInfo.get_hkt_details(stockcode)

释义: 获取指定品种的持股明细

参数:

- stockcode: string, 必须是'stock.market'形式

返回:

根据stockcode返回一个dict, 该字典的key值是北向持股明细数据的时间戳, 其值仍然是一个dict, 其值的key值是北向持股明细数据的字段类型, 其值是对应字段的值, 该字典数据key值有:

stockCode:股票代码

ownSharesCompany: 机构名称

ownSharesAmount:持股数量, 单位: 股

ownSharesMarketValue:持股市值, 单位: 元

ownSharesRatio:持股数量占比, 单位: %

ownSharesNetBuy:净买入, 单位: 元, 浮点数 (当日持股-前一日持股)

示例:

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.get_hkt_details('600000.SH')
```

(55) 获取指定市场的最新时间 get_market_time()

用法: get_market_time(market)

释义: 根据指定的市场, 获取对应的市场的最新的时间

参数:

- market: 市场代码 (如"SH")

返回: 市场最新时间

示例:

```
1 | get_market_time("SH")
```

3.2.4. 交易函数

交易函数下单跟界面人工点击下单走的下单、风控流程是一样的, 唯一的区别是: 交易函数下单通过解析下单函数接口传过来的参数, 形成相应的下单任务; 而界面人工点击下单通过解析界面的输入、选择, 形成相应的下单任务。

运作流程:

(1) 编写测试策略



```
#coding:gbk
def __init__(ContextInfo):
    pass
def handlebar(ContextInfo):
    passorder(23, 1101, '6000000248', '000001.SZ', 5, -1, 100, ContextInfo)
```

(2) 运行策略



(3) 触发下单, 解析参数



(4) 生成任务



任务	账号	代码	名称	交易参数	交易类型	金额	进度	委托详情	开始时间	结束时间	操作	单次数	状态
任务2893	6000000248	000001	平安银行	普通	买入	1060.00	100 / 100	已报! 收到!	13:25:33	13:25:35		0	已完成

(5) 形成委托，过风控

全部状态	全部来源	金选	反选	选中数据	全部撤单	最新	撤卖	委托	请输入代码	清输入代码	清
1	acct6000000248	6000000248	深交所	000001	20181121	0	平安银行	10.60	13:25:42	0股	已报

(6) 成交

成交明细	成交统计	全部来源	请输入代码	清
1	6000000248	acct6000000248	函数下单	香智金融模拟

3.2.4.1. 交易函数基本信息

QMT 系统提供了一系列的 Python 下单函数接口，如下所示。

*注：在回测模式中，交易函数调用虚拟账号进行交易，在历史 K 线上记录买卖点，用以计算策略净值/回测指标；实盘运行调用策略中设置的资金账号进行交易，产生实际委托；模拟运行模式下交易函数无效。其中，`can_cancel_order`, `cancel_task`, `cancel`和`do_order`交易函数在回测模式中无实际意义，不建议使用。

(1) 交易函数

函数名	释义
passorder	综合交易下单
algo_passorder	算法交易下单
smart_algo_passorder	智能算法交易
get_trade_detail_data	取交易细明数据函数
get_value_by_order_id	根据委托ID取委托或成交信息
get_last_order_id	获取最新的委托或成交的委托ID
can_cancel_order	查询委托是否可撤销
cancel	取消委托
cancel_task	撤销任务
pause_task	暂停任务
resume_task	继续任务
do_order	实时触发前一根bar信号函数
stoploss_limitprice	以限价单方式止损/止盈
stoploss_marketprice	以市价单方式止损/止盈
get_new_purchase_limit	获取账号新股申购额度

(2) 股票下单函数

函数名	释义
order_lots	指定手数交易
order_value	指定价值交易
order_percent	一定比例下单
order_target_value	目标价值下单
order_target_percent	目标比例下单
order_shares	指定股数交易

(3) 期货下单函数

函数名	释义
buy_open	买入开仓
sell_close_tdayfirst	卖出平仓, 平今优先
sell_close_ydayfirst	卖出平仓, 平昨优先
sell_open	卖出开仓
buy_close_tdayfirst	买入平仓, 平今优先
buy_close_ydayfirst	买入平仓, 平昨优先

3.2.4.2. 交易函数介绍

(1) 综合交易下单 passorder()

用法: passorder(opType, orderType, accountid, orderCode, prType, modelprice, volume[, strategyName, quickTrade, userOrderId], ContextInfo)

释义: 综合交易下单

参数:

- opType, 操作类型, 可选值:

期货六键:

- 0: 开多
- 1: 平昨多
- 2: 平今多
- 3: 开空
- 4: 平昨空
- 5: 平今空

期货四键:

- 6: 平多,优先平今
- 7: 平多,优先平昨
- 8: 平空,优先平今
- 9: 平空,优先平昨

期货两键:

- 10: 卖出,如有多仓,优先平仓,优先平今,如有余量,再开空
- 11: 卖出,如有多仓,优先平仓,优先平昨,如有余量,再开空
- 12: 买入,如有空仓,优先平仓,优先平今,如有余量,再开多
- 13: 买入,如有空仓,优先平仓,优先平昨,如有余量,再开多
- 14: 买入,不优先平仓
- 15: 卖出,不优先平仓

股票买卖：

- 23：股票买入，或沪港通、深港通股票买入
- 24：股票卖出，或沪港通、深港通股票卖出

融资融券：

- 27：融资买入
- 28：融券卖出
- 29：买券还券
- 30：直接还券
- 31：卖券还款
- 32：直接还款
- 33：信用账号股票买入
- 34：信用账号股票卖出

组合交易：

- 25：组合买入，或沪港通、深港通的组合买入
- 26：组合卖出，或沪港通、深港通的组合卖出
- 27：融资买入
- 28：融券卖出
- 29：买券还券
- 31：卖券还款
- 33：信用账号股票买入
- 34：信用账号股票卖出
- 35：普通账号一键买卖
- 36：信用账号一键买卖
- 40：期货组合开多
- 43：期货组合开空
- 46：期货组合平多,优先平今
- 47：期货组合平多,优先平昨
- 48：期货组合平空,优先平今
- 49：期货组合平空,优先平昨

期权交易：

- 50：买入开仓
- 51：卖出平仓
- 52：卖出开仓
- 53：买入平仓
- 54：备兑开仓
- 55：备兑平仓
- 56：认购行权
- 57：认沽行权

58: 证券锁定

59: 证券解锁

ETF交易:

60: 申购

61: 赎回

专项两融:

70: 专项融资买入

71: 专项融券卖出

72: 专项买券还券

73: 专项直接还券

74: 专项卖券还款

75: 专项直接还款

可转债:

80: 普通账户转股

81: 普通账户回售

82: 信用账户转股

83: 信用账户回售

- orderType, 下单方式

*注:

一、期货不支持 1102 和 1202

二、对所有账号组的操作相当于对账号组里的每个账号做一样的操作，如 passorder(23, 1202, 'testS', '000001.SZ', 5, -1, 50000, ContextInfo)，意思就是对账号组 testS 里的所有账号都以最新价开仓买入 50000 元市值的 000001.SZ 平安银行；
passorder(60,1101,"test",'510050.SH',5,-1,1,ContextInfo)意思就是账号test申购1个单位(900000股)的华夏上证50ETF(只申购不买入成分股)。

可选值:

1101: 单股、单账号、普通、股/手方式下单

1102: 单股、单账号、普通、金额(元)方式下单(只支持股票)

1113: 单股、单账号、总资产、比例[0~1]方式下单

1123: 单股、单账号、可用、比例[0~1]方式下单

1201: 单股、账号组(无权重)、普通、股/手方式下单

1202: 单股、账号组(无权重)、普通、金额(元)方式下单(只支持股票)

1213: 单股、账号组(无权重)、总资产、比例[0~1]方式下单

1223: 单股、账号组(无权重)、可用、比例[0~1]方式下单

2101: 组合、单账号、普通、按组合股票数量(篮子中股票设定的数量)方式下单 > 对应 volume 的单位为篮子的份

2102: 组合、单账号、普通、按组合股票权重（篮子中股票设定的权重）方式下单 > 对应 volume 的单位为元

2103: 组合、单账号、普通、按账号可用方式下单 > （底层篮子股票怎么分配？答：按可用资金比例后按篮子中股票权重分配，如用户没填权重则按相等权重分配）只对股票篮子支持

2201: 组合、账号组（无权重）、普通、按组合股票数量方式下单

2202: 组合、账号组（无权重）、普通、按组合股票权重方式下单

2203: 组合、账号组（无权重）、普通、按账号可用方式下单只对股票篮子支持

- 组合套利交易接口特殊设置 (accountID、orderType 特殊设置)

passorder(opType, orderType, accountID, orderCode, prType, hedgeRatio, volume, ContextInfo)

accountID = 'stockAccountID, futureAccountID'

orderCode = 'basketName, futureName'

hedgeRatio: 套利比例 (0 ~ 2 之间值, 相当于 %0 至 200% 套利)

volume: 份数 \ 资金 \ 比例

orderType (特殊设置)

- orderType 可选值:

2331: 组合、套利、合约价值自动套利、按组合股票数量方式下单

2332: 组合、套利、按合约价值自动套利、按组合股票权重方式下单

2333: 组合、套利、按合约价值自动套利、按账号可用方式下单

- accountID, 资金账号

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注: 下单的账号ID (可多个) 或账号组名或套利组名 (一个篮子一个套利账号, 如 accountID = '股票账户名, 期货账号')

- orderCode, 下单代码

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注:

一、如果是单股或单期货、港股，则该参数填合约代码；

二、如果是组合交易,则该参数填篮子名称；

三、如果是组合套利, 则填一个篮子名和一个期货合约名 (如orderCode = '篮子名, 期货合约名')

- prType, 下单选价类型

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

可选值 (特别的对于套利, 这个 prType 只对篮子起作用, 期货的采用默认的方式) :

-1: 无效 (实际下单时, 需要用交易面板交易函数那设定的选价类型)

0: 卖5价

1: 卖4价

2: 卖3价

3: 卖2价
4: 卖1价
5: 最新价
6: 买1价
7: 买2价 (组合不支持)
8: 买3价 (组合不支持)
9: 买4价 (组合不支持)
10: 买5价 (组合不支持)
11: (指定价) 模型价 (只对单股情况支持,对组合交易不支持)
12: 涨跌停价
13: 挂单价
14: 对手价
27: 市价即成剩撤(仅对股票期权申报有效)
28: 市价即全成否则撤(仅对股票期权申报有效)
29: 市价剩转限价(仅对股票期权申报有效)
42: 最优五档即时成交剩余撤销申报(仅对上交所申报有效)
43: 最优五档即时成交剩转限价申报(仅对上交所申报有效)
44: 对手方最优价格委托(仅对深交所申报有效)
45: 本方最优价格委托(仅对深交所申报有效)
46: 即时成交剩余撤销委托(仅对深交所申报有效)
47: 最优五档即时成交剩余撤销委托(仅对深交所申报有效)
48: 全额成交或撤销委托(仅对深交所申报有效)
49: 科创板盘后定价

- price, 下单价格

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注:

一、单股下单时, prType 是模型价/科创板盘后定价时 price 有效; 其它情况无效; 即单股时, prType 参数为 11, 49 时被使用。prType 参数不为 11, 49 时也需填写, 填写的内容可为 -1, 0, 2, 100 等任意数字;

二、组合下单时, 是组合套利时, price 作套利比例有效, 其它情况无效。

- volume, 下单数量 (股 / 手 / 元 / %)

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

根据 orderType 值最后一位确定 volume 的单位:

单股下单时:

- 1: 股 / 手
- 2: 金额 (元)
- 3: 比例 (%)

组合下单时:

1: 按组合股票数量 (份)

2: 按组合股票权重 (元)

3: 按账号可用 (%)

- strategyName, string, 自定义策略名, 可缺省不写, 用来区分 order 委托和 deal 成交来自不同的策略。根据该策略名, get_trade_detail_data, get_last_order_id 函数可以获取相应策略名对应的委托或持仓结果。

*注: strategyName 只对同账号本地客户端有效, 即 strategyName 只对当前客户端下的单进行策略区分, 且该策略区分只能当前客户端使用。

- quickTrade, int, 设定是否立即触发下单, 可选值:

0: 否

1: 是

*注: passorder是对最后一根K线完全走完后生成的模型信号在下一根K线的第一个tick数据来时触发下单交易; 采用quickTrade参数设置为1时, 非历史bar上执行时 (ContextInfo.is_last_bar() 为True), 只要策略模型中调用到就触发下单交易。quickTrade参数设置为2时, 不判断bar状态, 只要策略模型中调用到就触发下单交易, 历史bar上也能触发下单, 请谨慎使用。

- userOrderId, string, 用户自设委托 ID, 可缺省不写, 写的时候必须把起前面的 strategyName 和 quickTrade 参数也填写。对应 order 委托对象和 deal 成交对象中的 m_strRemark 属性, 通过 get_trade_detail_data 函数或委托主推函数 order_callback 和成交主推函数 deal_callback 可拿到这两个对象信息。
- userOrderParam, string, 用户自定义交易参数模板名称, 可缺省不写, 写的时候必须把起前面的 strategyName 和 quickTrade 参数也填写。

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     # 单股单账号期货最新价买入 10 手  
3     passorder(0, 1101, 'test', target, 5, -1, 10, ContextInfo)  
4  
5     # 单股单账号期货指定价买入 10 手  
6     passorder(0, 1101, 'test', target, 11, 3000, ContextInfo)  
7  
8     # 单股单账号股票最新价买入 100 股 (1 手)  
9     passorder(23, 1101, 'test', target, 5, 0, 100, ContextInfo)  
10  
11    # 单股单账号股票指定价买入 100 股 (1 手)  
12    passorder(23, 1101, 'test', target, 11, 7, 100, ContextInfo)
```

(2) 智能算法交易 smart_algo_passorder()

用法:

smart_algo_passorder(opType,orderType,accountid,orderCode,prType,modelprice,volume,strageName,quickTrade,userid,smartAlgoType,limitOverRate,minAmountPerOrder,[targetPriceLevel,startTime,endTime,limitControl],ContextInfo)

释义: 智能算法交易

参数:

- opType, 操作类型, 可选值:

股票买卖:

23: 股票买入, 或沪港通、深港通股票买入

24: 股票卖出, 或沪港通、深港通股票卖出

融资融券:

27: 融资买入

28: 融券卖出

29: 买券还券

30: 直接还券

31: 卖券还款

32: 直接还款

33: 信用账号股票买入

34: 信用账号股票卖出

35: 普通账号一键买卖

36: 信用账号一键买卖

- orderType, 下单方式

可选值:

1101: 单股、单账号、普通、股/手方式下单

1102: 单股、单账号、普通、金额(元)方式下单(只支持股票)

1113: 单股、单账号、总资产、比例[0~1]方式下单

1123: 单股、单账号、可用、比例[0~1]方式下单

1201: 单股、账号组(无权重)、普通、股/手方式下单

1202: 单股、账号组(无权重)、普通、金额(元)方式下单(只支持股票)

1213: 单股、账号组(无权重)、总资产、比例[0~1]方式下单

1223: 单股、账号组(无权重)、可用、比例[0~1]方式下单

- accountID, 资金账号

*注: 下单的账号ID(可多个)或账号组名或套利组名(一个篮子一个套利账号,如accountID='股票账户名,期货账号')

- orderCode, 下单代码

*注: 有两种如果是单股或单期货、港股,则该参数填合约代码;如果是组合交易,则该参数填篮子名称.如果是组合套利,则填一个篮子名和一个期货合约名(如orderCode='篮子名,期货合约名')

- prType, 下单选价类型

*注: 特别的对于套利:这个prType只对篮子起作用, 期货的采用默认的方式

可选值:

0: 卖5价

1: 卖4价

2: 卖3价

3: 卖2价

- 4: 卖1价
- 5: 最高价
- 6: 买1价
- 7: 买2价(组合不支持)
- 8: 买3价(组合不支持)
- 9: 买4价(组合不支持)
- 10: 买5价(组合不支持)
- 11: (指定价) 模型价 (只对单股情况支持,对组合交易不支持)
- 12: 涨跌停价
- 13: 挂单价
- 14: 对手价

如果prType = 12, 则智能算法下涨跌停价, 其他情况下限价

- price, 下单价格

*注: prType是模型价时price有效; 其它情况无效
- volume, 下单数量 (股 / 元 / %)

根据 orderType 值最后一位确定 volume 的单位:

单股下单时:

- 1: 股
- 2: 金额 (元)
- 3: 比例 (%)

- strageName, 策略名

不同于passorder, 这里不可缺省
- quickTrade, 参数内容说明

quickTrade: int, 是否立马触发下单, 0 否, 1 是

*注:

passorder是对最后一根K线完全走完后生成的模型信号在下一根K线的第一个tick数据来时触发下单交易;

采用quickTrade参数设置为1时, 非历史bar上执行时 (ContextInfo.is_last_bar()为True), 只要策略模型中调用到就触发下单交易。

quickTrade参数设置为2时, 不判断bar状态, 只要策略模型中调用到就触发下单交易, 历史bar上也能触发下单, 请谨慎使用。

- userid, 投资备注

不同于passorder, 这里不可缺省
- smartAlgoType, string, 智能算法类型

可选值:

- VWAP: VWAP
- TWAP: TWAP
- VP: 跟量
- PINLINE: 跟价
- DMA: 快捷
- FLOAT: 盘口

SWITCH: 换仓

ICEBERG: 冰山

MOC: 尾盘

- limitOverRate, int, 量比, 数据范围0-100, 如果输入其他无效值, 则limitOverRate为0。
*注: 网格算法无此项。
- minAmountPerOrder, int, 智能算法最小委托金额, 数据范围0-100000, 默认为0。
- targetPriceLevel, 智能算法目标价格

可选值:

1: 己方盘口1

2: 己方盘口2

3: 己方盘口3

4: 己方盘口4

5: 己方盘口5

6: 最新价

7: 对方盘口

*注:

一、输入无效值则targetPriceLevel为1;

二、本项只针对冰山算法,其他算法可缺省。

- startTime, 智能算法开始时间

格式"HH:MM:SS", 如"10:30:00"。如果缺省值, 则默认为"09:30:00"

- endTime, 智能算法截止时间

格式"HH:MM:SS", 如"14:30:00"。如果缺省值, 则默认为"15:30:00"

- limitControl, 涨跌停控制

1: 涨停不卖跌停不卖

0: 无

默认值为1

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     smart_algo_passorder(23,1101,'600000105','000001.SZ',5,-1,50000,"strageName  
",0,"remark","TWAP",20,0,ContextInfo);#账户600000105最新价开仓买入50000股的  
000001.SZ平安银行,使用TWAP智能算法,量比20%,最小买卖金额0  
3     smart_algo_passorder(23,1101,'600000105','000001.SZ',5,-1,50000,"strageName  
",1,"remark","TWAP",20,0,0,'09:30:00','14:00:00',ContextInfo);#账户600000105最  
新价快速交易开仓买入50000股的000001.SZ平安银行,使用TWAP智能算法,量比20%,最小买卖金额0 且  
有效时长为09:30-14:00  
4     smart_algo_passorder(23,1101,'600000105','000001.SZ',5,-1,50000,"strageName  
",1,"remark","TWAP",20,0,0,'09:30:00','14:00:00',0,ContextInfo);#账户600000105最  
新价快速交易开仓买入50000股的000001.SZ平安银行,使用TWAP智能算法,量比20%,最小买卖金额0  
且有效时长为09:30-14:00,不对智能算法涨停做限制
```

(3) 获取交易明细数据 get_trade_detail_data()

用法: get_trade_detail_data(accountID, strAccountType, strDatatype, strategyName) 或不区分策略
get_trade_detail_data(accountID, strAccountType, strDatatype)

释义: 获取交易明细数据函数

参数:

- accountID: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票
'CREDIT': 信用
'HUGANGTONG': 沪港通
'SHENGANGTONG': 深港通
'STOCK_OPTION': 期权

- strdatatype: string, 数据类型:

'POSITION': 持仓
'ORDER': 委托
'DEAL': 成交
'ACCOUNT': 账号
'TASK': 任务

- strategyName: string, 策略名, 对应 passorder 下单函数中的参数 strategyName 的值, 只对委托 'ORDER'、成交 'DEAL' 起作用。

返回: list, list 中放的是 PythonObj, 通过 dir(pythonobj) 可返回某个对象的属性列表。

*注: 有五种交易相关信息, 包括:

POSITION: 持仓
ORDER: 委托
DEAL: 成交
ACCOUNT: 账号
TASK: 任务

示例:

```
1 def handlebar(ContextInfo):
2     obj_list = get_trade_detail_data('6000000248', 'stock', 'position')
3     for obj in obj_list:
4         print(obj.m_strInstrumentID)
5         # 查看有哪些属性字段
6         print(dir(obj))
7
8     # 可用资金查询
9     acct_info = get_trade_detail_data('6000000248', 'stock', 'account')
```

```
10     for i in acct_info:
11         print(i.m_dAvailable)
12
13     # 当前持仓查询
14     position_info = get_trade_detail_data('6000000248', 'stock', 'position')
15     for i in position_info:
16         print(i.m_strInstrumentID, i.m_nVolume)
```

(4) 根据委托号获取委托或成交信息 get_value_by_order_id()

用法: get_value_by_order_id(orderId, accountID, strAccountType, strDatatype)

释义: 根据委托号获取委托或成交信息。

参数:

- orderId: string, 委托号。
- accountID: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票
'CREDIT': 信用
'HUGANGTONG': 沪港通
'SHENGANGTONG': 深港通
'STOCK_OPTION': 期权

- strDatatype: string, 数据类型:

'ORDER': 委托
'DEAL': 成交

返回: pythonObj

示例:

```
1 def init(ContextInfo):
2     ContextInfo.accid = '6000000248'
3
4 def handlebar(ContextInfo):
5     orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')
6     print(orderid)
7     obj = get_value_by_order_id(orderid, ContextInfo.accid, 'stock', 'order')
8     print(obj.m_strInstrumentID)
```

(5) 获取最新的委托或成交的委托号 get_last_order_id()

用法: get_last_order_id(accountID, strAccountType, strDatatype, strategyName) 或不区分策略
get_last_order_id(accountID, strAccountType, strDatatype)

释义: 获取最新的委托或成交的委托号。

参数:

- accountID: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票
'CREDIT': 信用
'HUGANGTONG': 沪港通
'SHENGANGTONG': 深港通
'STOCK_OPTION': 期权

- strdatatype: string, 数据类型:

'ORDER': 委托
'DEAL' : 成交

- strategyName, string, 策略名, 对应 passorder 下单函数中的参数 strategyName 的值。

返回: String, 委托号, 如果没找到返回 '-1'。

示例:

```
1 def init(ContextInfo):  
2     ContextInfo.accid = '6000000248'  
3  
4 def handlebar(ContextInfo):  
5     orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')  
6     print(orderid)  
7     obj = get_value_by_order_id(orderid, ContextInfo.accid, 'stock', 'order')  
8     print(obj.m_strInstrumentID)
```

(6) 查询委托是否可撤销 can_cancel_order()

用法: can_cancel_order(orderId, accountID, strAccountType)

释义: 查询委托是否可撤销。

参数:

- orderId, string, 委托号。
- accountID: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票

```
'CREDIT': 信用  
'HUGANGTONG': 沪港通  
'SHENGANGTONG': 深港通  
'STOCK_OPTION': 期权
```

返回: bool, 是否可撤, 返回值含义:

True: 可撤销

False: 不可撤销

示例:

```
1 def init(ContextInfo):  
2     ContextInfo.accid = '6000000248'  
3  
4 def handlebar(ContextInfo):  
5     orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')  
6     print(orderid)  
7     can_cancel = can_cancel_order(orderid, ContextInfo.accid, 'stock')  
8     print('是否可撤:', can_cancel)
```

(7) 取消委托 cancel()

用法: cancel(orderId, accountId, accountType, ContextInfo)

释义: 取消委托。

参数:

- orderId: string, 委托号。
- accountId: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

```
'FUTURE': 期货  
'STOCK': 股票  
'CREDIT': 信用  
'HUGANGTONG': 沪港通  
'SHENGANGTONG': 深港通  
'STOCK_OPTION': 期权
```

- ContextInfo: pythonobj

返回: bool, 是否发出了取消委托信号, 返回值含义:

True: 是

False: 否

示例:

```
1 ...  
2 (1) 下单前, 根据 get_trade_detail_data 函数返回账号的信息, 判定资金是否充足, 账号是否在  
3 登录状态, 统计持仓情况等等。
```

```

3 (2) 满足一定的模型条件，用 passorder 下单。
4 (3) 下单后，时刻根据 get_last_order_id 函数获取委托和成交的最新id，注意如果委托生成
5 了，就有了委托号（这个id需要自己保存做一个全局控制）。
6 (4) 用该委托号根据 get_value_by_order_id 函数查看委托的状态，各种情况等。
7 当一个委托的状态变成“已成”后，那么对应的成交 deal 信息就有一条成交数据；用该委托号可查看成
8 交情况。
9 *注：委托列表和成交列表中的委托号是一样的，都是这个 m_strorderSysID 属性值。
10 可用 get_last_order_id 获取最新的 order 的委托号，然后根据这个委托号获取 deal 的信息，当
11 获取成功后，也说明这笔交易是成了，可再根据 position 持仓信息再进一步验证。
12 (5) 根据委托号获取委托信息，根据委托状态，或模型设定，用 cancel 取消委托。
13 ...
14
15 def init(ContextInfo):
16     ContextInfo.accid = '6000000248'
17
18 def handlebar(ContextInfo):
19     orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')
20     print(cancel(orderid, ContextInfo.accid, 'stock', ContextInfo))

```

(8) 撤销任务 `cancel_task()`

用法： `cancel_task(taskId, accountId, accountType, ContextInfo)`

释义： 撤销任务。

参数：

- `taskId`, string, 任务编号, 为空的时候表示撤销所有该资金账号可撤的任务。
- `accountId`: string, 资金账号。
- `strAccountType`: string, 账号类型, 可选值:

'FUTURE': 期货
 'STOCK': 股票
 'CREDIT': 信用
 'HUGANGTONG': 沪港通
 'SHENGANGTONG': 深港通
 'STOCK_OPTION': 期权

- `ContextInfo`: pythonobj

返回： bool, 是否发出了取消任务信号，返回值含义：

True: 是

False: 否

示例：

```
1  ...
2  (1) 根据get_trade_detail_data函数返回任务的信息，获取任务编号（m_nTaskId），任务状态
3  等等；
4  (2) 根据任务编号，用cancel_task取消委托。
5  ...
6  def init(ContextInfo):
7      ContextInfo.accid = '6000000248'
8
9  def handlebar(ContextInfo):
10     objlist = get_trade_detail_data(ContextInfo.accid,'stock','task')
11     for obj in obj_list:
12         cancel_task(obj.m_nTaskId,ContextInfo.accid,'stock',ContextInfo)
```

(9) 暂停任务 pause_task()

用法: pause_task(taskId,accountId,accountType,ContextInfo)

释义: 暂停智能算法任务。

参数:

- taskId: string, 任务编号, 为空的时候表示暂停所有该资金账号可暂停的任务。
- accountId: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票
'CREDIT': 信用
'HUGANGTONG': 沪港通
'SHENGANGTONG': 深港通
'STOCK_OPTION': 期权

- ContextInfo: pythonobj

返回: bool, 是否发出了暂停任务信号, 返回值含义:

True: 是

False: 否

示例:

```
1   ...
2   (1) 根据get_trade_detail_data函数返回任务的信息, 获取任务编号(m_nTaskID), 任务状态
3   等等;
4   (2) 根据任务编号, 用pause_task暂停智能算法任务。
5   ...
6 def init(ContextInfo):
7     ContextInfo.accid = '6000000248'
8 def handlebar(ContextInfo):
9     objlist = get_trade_detail_data(ContextInfo.accid, 'stock', 'task')
10    for obj in obj_list:
11        pause_task(obj.m_nTaskID, ContextInfo.accid, 'stock', ContextInfo)
```

(10) 继续任务 cancel_task()

用法: cancel_task(taskId, accountId, accountType, ContextInfo)

释义: 继续智能算法任务。

参数:

- taskId: string, 任务编号, ,为空的时候表示启动所有该资金账号已暂停的任务。
- accountId: string, 资金账号。
- strAccountType: string, 账号类型, 可选值:

'FUTURE': 期货
'STOCK': 股票
'CREDIT': 信用
'HUGANGTONG': 沪港通
'SHENGANGTONG': 深港通
'STOCK_OPTION': 期权

- ContextInfo: pythonobj

返回: bool, 是否发出了重启信号, 返回值含义:

True: 是

False: 否

示例:

```

1   ...
2   (1) 根据get_trade_detail_data函数返回任务的信息, 获取任务编号(m_nTaskID), 任务状态
3   等等;
4   (2) 根据任务编号, 用resume_task启动已暂停智能算法任务。
5   ...
6 def init(ContextInfo):
7     ContextInfo.accid = '6000000248'
8 def handlebar(ContextInfo):
9     objlist = get_trade_detail_data(ContextInfo.accid, 'stock', 'task')
10    for obj in obj_list:
11        resume_task(obj.m_nTaskID, ContextInfo.accid, 'stock', ContextInfo)

```

(11) 实时触发前一根 bar 信号函数 do_order()

用法: do_order(ContextInfo)

释义: 实时触发前一根bar信号函数。

系统实盘中交易下单函数一般是把上一个周期产生的信号在最新的周期的第一个 tick 下单出去, 而日 K 线第一个 tick 是在 9:25 分集合竞价结束时产生, 如果策略模型在 9:25 分之后跑又想把前一天的下单信号发出去, 就可用 do_order 函数配合实现。

特别的, 需要注意, 有调用 do_order 函数的策略模型跑在 9:25 分之前或别的日内周期下时, 原有下单函数和 do_order 函数都有下单信号, 有可能导致重复下单。.

参数: 无

返回: 无

示例:

```

1 # 实现跑日 K 线及以上周期下, 在固定时间点把前一周期的交易信号发送出去
2 def init(ContextInfo):
3     pass
4
5 def handlebar(ContextInfo):
6     order_lots('000002.SZ', 1, ContextInfo, '600000248')
7     ticktimetag = ContextInfo.get_tick_timetag()
8     int_time = int(timetag_to_datetime(ticktimetag, '%H%M%S'))
9     if 100500 <= int_time < 100505:
10         do_order(ContextInfo)

```

(12) 指定手数交易 order_lots()

用法: order_lots(stockcode, lots[, style, price], ContextInfo[, accId])

释义: 指定手数交易, 指定手数发送买/卖单。如有需要落单类型当做一个参数传入, 如果忽略掉落单类型, 那么默认以最新价下单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- lots: 手数, int

- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

```
'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价
```

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- acclid: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价下 1 手买入
3     order_lots('000002.SZ', 1, ContextInfo, '600000248')
4
5     # 用对手价下 1 手卖出
6     order_lots('000002.SZ', -1, 'COMPETE', ContextInfo, '600000248')
7
8     # 用指定价 37.5 下 2 手卖出
9     order_lots('000002.SZ', -2, 'fix', 37.5, ContextInfo, '600000248')
```

(13) 指定价值交易 order_value()

用法: order_value(stockcode, value[, style, price], ContextInfo[, acclid])

释义: 指定价值交易, 使用想要花费的金钱买入 / 卖出股票, 而不是买入 / 卖出想要的股数, 正数代表买入, 负数代表卖出。股票的股数总是会被调整成对应的 100 的倍数 (在中国 A 股市场 1 手是 100 股)。当您提交一个卖单时, 该方法代表的意义是您希望通过卖出该股票套现的金额, 如果金额超出了您所持有股票的价值, 那么您将卖出所有股票。需要注意, 如果资金不足, 该 API 将不会创建发送订单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- value: 金额 (元), double
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

```
'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
```

'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- acld: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     # 按最新价下 10000 元买入  
3     order_value('000002.SZ', 10000, ContextInfo, '600000248')  
4  
5     # 用对手价下 10000 元卖出  
6     order_value('000002.SZ', -10000, 'COMPETE', ContextInfo, '600000248')  
7  
8     # 用指定价 37.5 下 20000 元卖出  
9     order_value('000002.SZ', -20000, 'fix', 37.5, ContextInfo, '600000248')
```

(14) 指定比例交易 order_percent()

用法: order_percent(stockcode, percent[, style, price], ContextInfo[, acld])

释义: 指定比例交易, 发送一个等于目前投资组合价值 (市场价值和目前现金的总和) 一定百分比的买 / 卖单, 正数代表买, 负数代表卖。股票的股数总是会被调整成对应的一手的股票数的倍数 (1 手是 100 股)。百分比是一个小数, 并且小于或等于1 (小于等于100%) , 0.5 表示的是 50%。需要注意, 如果资金不足, 该 API 将不会创建发送订单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- percent: 比例, double
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- acld: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价下 5.1% 价值买入
3     order_percent('000002.sz', 0.051, ContextInfo, '600000248')
4
5     # 用对手价下 5.1% 价值卖出
6     order_percent('000002.sz', -0.051, 'COMPETE', ContextInfo, '600000248')
7
8     # 用指定价 37.5 下 10.2% 价值卖出
9     order_percent('000002.sz', -0.102, 'fix', 37.5, ContextInfo, '600000248')
```

(15) 指定目标价值交易 order_target_value()

用法: order_target_value(stockcode, tar_value[, style, price], ContextInfo[, accId])

释义: 指定目标价值交易，买入 / 卖出并且自动调整该证券的仓位到一个目标价值。如果还没有任何该证券的仓位，那么会买入全部目标价值的证券；如果已经有了该证券的仓位，则会买入 / 卖出调整该证券的现在仓位和目标仓位的价值差值的数目的证券。需要注意，如果资金不足，该API将不会创建发送订单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- tar_value: 目标金额 (元), double, 非负数
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价下调仓到 10000 元持仓
3     order_target_value('000002.sz', 10000, ContextInfo, '600000248')
4
5     # 用对手价调仓到 10000 元持仓
6     order_target_value('000002.sz', 10000, 'COMPETE', ContextInfo,
7                         '600000248')
8
9     # 用指定价 37.5 下调仓到 20000 元持仓
10    order_target_value('000002.sz', 20000, 'fix', 37.5, ContextInfo,
11                         '600000248')
```

(16) 指定目标比例交易 order_target_percent()

用法: order_target_percent(stockcode, tar_percent[, style, price], ContextInfo[, acclId])

释义: 指定目标比例交易，买入 / 卖出证券以自动调整该证券的仓位到占有一个指定的投资组合的目标百分比。投资组合价值等于所有已有仓位的价值和剩余现金的总和。买 / 卖单会被下舍入一手股数（A 股是 100 的倍数）的倍数。目标百分比应该是一个小数，并且最大值应该小于等于1，比如 0.5 表示 50%，需要注意，如果资金不足，该API将不会创建发送订单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- tar_percent: 目标百分比 [0 ~ 1], double
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- acclId: 账号, string

返回: 无

示例:

```

1 def handlebar(ContextInfo):
2     # 按最新价下买入调仓到 5.1% 持仓
3     order_target_percent('000002.sz', 0.051, ContextInfo, '600000248')
4
5     # 用对手价调仓到 5.1% 持仓
6     order_target_percent('000002.sz', 0.051, 'COMPETE', ContextInfo,
7     '600000248')
8
9     # 用指定价 37.5 调仓到 10.2% 持仓
10    order_target_percent('000002.sz', 0.102, 'fix', 37.5, ContextInfo,
11    '600000248')

```

(17) 指定股数交易 order_shares()

用法: order_shares(stockcode, shares[, style, price], ContextInfo[, accId])

释义: 指定股数交易，指定股数的买 / 卖单,最常见的落单方式之一。如有需要落单类型当做一个参数传入，如果忽略掉落单类型，那么默认以最新价下单。

参数:

- stockcode: 代码, string, 如 '000002.SZ'
- shares: 股数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
 'FIX': 指定
 'HANG': 挂单
 'COMPETE': 对手
 'MARKET': 市价
 'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1': 卖5-1价
 'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5': 买1-5价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```

1 def handlebar(ContextInfo):
2     # 按最新价下 100 股买入
3     order_shares('000002.sz', 100, ContextInfo, '600000248')
4
5     # 用对手价下 100 股卖出
6     order_shares('000002.sz', -100, 'COMPETE', ContextInfo, '600000248')
7
8     # 用指定价 37.5 下 200 股卖出
9     order_shares('000002.sz', -200, 'fix', 37.5, ContextInfo, '600000248')

```

(18) 期货买入开仓 buy_open()

用法: buy_open(stockcode, amount[, style, price], ContextInfo[, accId])

释义: 期货买入开仓

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE1': 卖一价
'BUY1': 买一价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     # 按最新价 1 手买入开仓  
3     buy_open('IF1805.IF', 1, ContextInfo, '110476')  
4  
5     # 用对手价 1 手买入开仓  
6     buy_open('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')  
7  
8     # 用指定价 3750 元 2 手买入开仓  
9     buy_open('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

(19) 期货买入平仓 (平今优先) buy_close_tdayfirst()

用法: buy_close_tdayfirst(stockcode, amount[, style, price], ContextInfo[, accId])

释义: 期货买入平仓, 平今优先

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新

'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE1': 卖一价
'BUY1': 买一价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     # 按最新价 1 手买入平仓, 平今优先  
3     buy_close_tdayfirst('IF1805.IF', 1, ContextInfo, '110476')  
4  
5     # 用对手价 1 手买入平仓, 平今优先  
6     buy_close_tdayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')  
7  
8     # 用指定价 3750 元 2 手买入平仓, 平今优先  
9     buy_close_tdayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

(20) 期货买入平仓 (平昨优先) `buy_close_ydayfirst()`

用法: `buy_close_ydayfirst(stockcode, amount[, style, price], ContextInfo[, accId])`

释义: 期货买入开仓, 平昨优先

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE1': 卖一价
'BUY1': 买一价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价 1 手买入平仓, 平昨优先
3     buy_close_ydayfirst('IF1805.IF', 1, ContextInfo, '110476')
4
5     # 用对手价 1 手买入平仓, 平昨优先
6     buy_close_ydayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')
7
8     # 用指定价 3750 元 2 手买入平仓, 平昨优先
9     buy_close_ydayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

(21) 期货卖出开仓 sell_open()

用法: sell_open(stockcode, amount[, style, price], ContextInfo[, accId])

释义: 期货卖出开仓

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:

'LATEST': 最新
'FIX': 指定
'HANG': 挂单
'COMPETE': 对手
'MARKET': 市价
'SALE1': 卖一价
'BUY1': 买一价

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价 1 手卖出开仓
3     sell_open('IF1805.IF', 1, ContextInfo, '110476')
4
5     # 用对手价 1 手卖出开仓
6     sell_open('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')
7
8     # 用指定价 3750 元 2 手卖出开仓
9     sell_open('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

(22) 期货卖出平仓 (平今优先) sell_close_tdayfirst()

用法: sell_close_tdayfirst(stockcode, amount[, style, price], ContextInfo[, accId])

释义: 期货卖出平仓，平今优先

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:
 - 'LATEST': 最新
 - 'FIX': 指定
 - 'HANG': 挂单
 - 'COMPETE': 对手
 - 'MARKET': 市价
 - 'SALE1': 卖一价
 - 'BUY1': 买一价
- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- accId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):
2     # 按最新价下 1 手卖出平仓, 平今优先
3     sell_close_tdayfirst('IF1805.IF', 1, ContextInfo, '110476')
4
5     # 用对手价 1 手卖出平仓, 平今优先
6     sell_close_tdayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')
7
8     # 用指定价 3750 元 2 手卖出平仓, 平今优先
9     sell_close_tdayfirst('IF1805.IF', 1, 'fix', 3750, ContextInfo, '110476')
```

(23) 期货卖出平仓 (平昨优先) sell_close_ydayfirst()

用法: sell_close_ydayfirst(stockcode, amount[, style, price], ContextInfo[, accId])

释义: 期货卖出平仓，平今优先

参数:

- stockcode: 代码, string, 如 'IF1805.IF'
- amount: 手数, int
- style: 下单选价类型, string, 默认为最新价 'LATEST', 可选值:
 - 'LATEST': 最新

```
'FIX': 指定  
'HANG': 挂单  
'COMPETE': 对手  
'MARKET': 市价  
'SALE1': 卖一价  
'BUY1': 买一价
```

- price: 价格, double
- ContextInfo: PythonObj, Python 对象, 这里必须是 ContextInfo
- acclId: 账号, string

返回: 无

示例:

```
1 def handlebar(ContextInfo):  
2     # 按最新价 1 手卖出平仓, 平昨优先  
3     sell_close_ydayfirst('IF1805.IF', 1, ContextInfo, '110476')  
4  
5     # 用对手价 1 手卖出平仓, 平昨优先  
6     sell_close_ydayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')  
7  
8     # 用指定价 3750 元 2 手卖出平仓, 平昨优先  
9     sell_close_ydayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

(24) 获取两融负债合约明细 get_debt_contract()

用法: get_debt_contract(acclId)

释义: 获取信用账户负债合约明细

参数:

- acclId: 信用账户

返回: list, list 中放的是 PythonObj, 通过 dir(pythonobj) 可返回某个对象的属性列表。

示例:

```
1 def handlebar(ContextInfo):  
2     obj_list = get_debt_contract('6000000248')  
3     for obj in obj_list:  
4         # 输出负债合约定名  
5         print(obj.m_strInstrumentName)
```

(25) 获取两融担保标的明细 get_assure_contract()

用法: get_assure_contract(acclId)

释义: 获取信用账户担保合约明细

参数:

- accId: 信用账户

示例:

```
1 def handlebar(ContextInfo):
2     obj_list = get_assure_contract('6000000248')
3     for obj in obj_list:
4         # 输出担保合约名
5         print(obj.m_strInstrumentName)
```

(26) 获取可融券明细 get_enable_short_contract()

用法: get_enable_short_contract(accId)

释义: 获取信用账户当前可融券的明细

参数:

- accId: 信用账户

示例:

```
1 def handlebar(ContextInfo):
2     obj_list = get_enable_short_contract('6000000248')
3     for obj in obj_list:
4         # 输出可融券合约名
5         print(obj.m_strInstrumentName)
```

(27) 以限价单方式止损/止盈 stoploss_limitprice()

用法: stoploss_limitprice(mode,accountid,stockcode,stopprice,limitprice,volume,orderderction)

释义: 以限价单方式止损/止盈

参数:

- mode: int, 止损: 0, 止盈: 1
- accountid: string, 账号
- stockcode: string, 需要止损的品种, 代码.市场格式(600000.SH,IF2003.IF)
- stopprice: double, 止损点
- limitprice: double, 限价单的指定价
- volume: int, 止损单的下单量, 不填写时候默认平该品种全仓
- orderderction: string, 买卖方向, 期货止损时候必须填写"short"对应平空, "long"对应平多, 品种为股票时填写""

示例:

```
1 def handlebar(ContextInfo):
2     stoploss_limitprice(0,'10455','IF2003.IF',3000.12,3000.05,1,"short")
```

(28) 以市价单方式止损/止盈 stoploss_marketprice()

用法: stoploss_marketprice(mode,accountid,stockcode,stopprice,volume,orderderction)

释义: 以市价单方式止损/止盈

参数:

- mode: int, 止损: 0, 止盈: 1
- accountid: string, 账号
- stockcode: string, 需要止损的品种, 代码.市场格式(600000.SH,IF2003.IF)
- stopprice: double, 止损点
- volume: int, 止损单的下单量, 不填写时候默认平该品种全仓
- orderderction: string, 买卖方向, 期货止损时候必须填写"short"对应平空, "long"对应平多, 品种为股票时填写""

示例:

```
1 | def handlebar(ContextInfo):  
2 |     stoploss_marketprice(0, '10455', 'IF2003.IF', 3000.12, 1, "long")
```

(29) 获取当日新股新债信息 get_ipo_data()

用法: get_ipo_data([.type])

释义: 获取当日新股新债信息, 返回结果为一个字典,包括新股申购代码,申购名称,最大申购数量,最小申购数量等数据

参数:

- type: 为空时返回新股新债信息, type="STOCK"时只返回新股申购信息, type="BOND"时只返回新债申购信息

示例:

```
1 | def init(ContextInfo):  
2 |     ipoData=get_ipo_data()#返回新股新债信息  
3 |     ipoStock=get_ipo_data("STOCK")#返回新股信息  
4 |     ipoCB=get_ipo_data("BOND")#返回新债申购信息
```

(30) 获取账户新股申购额度get_new_purchase_limit()

用法: get_new_purchase_limit(accid)

释义: 获取账户新股申购额度, 返回结果为一个字典,包括上海主板,深圳市场,上海科创板的申购额度

参数:

- accid: 资金账号, 必须时股票账号或者信用账号

示例:

```
1 | def init(ContextInfo):  
2 |     ContextInfo.accid="10000001"#返回新股新债信息  
3 |     purchase_limit=get_new_purchase_limit(ContextInfo.accid)
```

(31) 算法交易下单algo_passorder()

用法: algo_passorder(opType, orderType, accountid, orderCode, prType, modelprice, volume[, strategyName, quickTrade, userOrderId, userOrderParam], ContextInfo)

释义: 算法交易下单，此时使用交易面板-程序交易-函数交易-函数交易参数中设置的下单类型（普通交易,算法交易,随机量交易）。

如果函数交易参数使用未修改的默认值，此函数和passorder函数一致。

设置了函数交易参数后，将会使用函数交易参数的超价等拆单参数,如果入参的prType = -1，同时将会使用函数交易参数的报价方式。

参数:

- opType, 操作类型, 可选值:

期货六键:

0: 开多
1: 平昨多
2: 平今多
3: 开空
4: 平昨空
5: 平今空

期货四键:

6: 平多,优先平今
7: 平多,优先平昨
8: 平空,优先平今
9: 平空,优先平昨

期货两键:

10: 卖出,如有多仓,优先平仓,优先平今,如有余量,再开空
11: 卖出,如有多仓,优先平仓,优先平昨,如有余量,再开空
12: 买入,如有空仓,优先平仓,优先平今,如有余量,再开多
13: 买入,如有空仓,优先平仓,优先平昨,如有余量,再开多
14: 买入,不优先平仓
15: 卖出,不优先平仓

股票买卖:

23: 股票买入, 或沪港通、深港通股票买入
24: 股票卖出, 或沪港通、深港通股票卖出

融资融券:

27: 融资买入
28: 融券卖出
29: 买券还券

- 30: 直接还券
- 31: 卖券还款
- 32: 直接还款
- 33: 信用账号股票买入
- 34: 信用账号股票卖出

组合交易：

- 25: 组合买入, 或沪港通、深港通的组合买入
- 26: 组合卖出, 或沪港通、深港通的组合卖出
- 27: 融资买入
- 28: 融券卖出
- 29: 买券还券
- 31: 卖券还款
- 33: 信用账号股票买入
- 33: 信用账号股票卖出
- 40: 期货组合开多
- 43: 期货组合开空
- 46: 期货组合平多,优先平今
- 47: 期货组合平多,优先平昨
- 48: 期货组合平空,优先平今
- 49: 期货组合平空,优先平昨

期权交易：

- 50: 买入开仓
- 51: 卖出平仓
- 52: 卖出开仓
- 53: 买入平仓
- 54: 备兑开仓
- 55: 备兑平仓
- 56: 认购行权
- 57: 认沽行权
- 58: 证券锁定
- 59: 证券解锁

- orderType, 下单方式

*注：

- 一、期货不支持 1102 和 1202
- 二、对所有账号组的操作相当于对账号组里的每个账号做一样的操作, 如 passorder(23, 1202, 'testS', '000001.SZ', 5, -1, 50000, ContextInfo), 意思就是对账号组 testS 里的所有账号都以最新价开仓买入 50000 元市值的 000001.SZ 平安银行; passorder(60,1101,"test",'510050.SH',5,-1,1,ContextInfo)意思就是账号test申购1个单位(900000股)的华夏上证50ETF(只申购不买入成分股)。

可选值：

- 1101：单股、单账号、普通、股/手方式下单
 - 1102：单股、单账号、普通、金额（元）方式下单（只支持股票）
 - 1113：单股、单账号、总资产、比例 [0 ~ 1] 方式下单
 - 1123：单股、单账号、可用、比例[0 ~ 1]方式下单
-
- 1201：单股、账号组（无权重）、普通、股/手方式下单
 - 1202：单股、账号组（无权重）、普通、金额（元）方式下单（只支持股票）
 - 1213：单股、账号组（无权重）、总资产、比例 [0 ~ 1] 方式下单
 - 1223：单股、账号组（无权重）、可用、比例 [0 ~ 1] 方式下单
-
- 2101：组合、单账号、普通、按组合股票数量（篮子中股票设定的数量）方式下单 > 对应 volume 的单位为篮子的份
 - 2102：组合、单账号、普通、按组合股票权重（篮子中股票设定的权重）方式下单 > 对应 volume 的单位为元
 - 2103：组合、单账号、普通、按账号可用方式下单 > （底层篮子股票怎么分配？答：按可用资金比例后按篮子中股票权重分配，如用户没填权重则按相等权重分配）只对股票篮子支持
-
- 2201：组合、账号组（无权重）、普通、按组合股票数量方式下单
 - 2202：组合、账号组（无权重）、普通、按组合股票权重方式下单
 - 2203：组合、账号组（无权重）、普通、按账号可用方式下单只对股票篮子支持

- accountID, 资金账号

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注：下单的账号ID（可多个）或账号组名或套利组名（一个篮子一个套利账号，如 accountID = '股票账户名, 期货账号'）

- orderCode, 下单代码

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注：

- 一、如果是单股或单期货、港股，则该参数填合约代码；
- 二、如果是组合交易，则该参数填篮子名称；
- 三、如果是组合套利，则填一个篮子名和一个期货合约名（如orderCode = '篮子名, 期货合约名'）

- prType, 下单选价类型

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

可选值（特别的对于套利，这个 prType 只对篮子起作用，期货的采用默认的方式）：

- 1：无效（实际下单时，需要用交易面板交易函数那设定的选价类型）
- 0：卖5价
- 1：卖4价
- 2：卖3价

3: 卖2价
4: 卖1价
5: 最新价
6: 买1价
7: 买2价 (组合不支持)
8: 买3价 (组合不支持)
9: 买4价 (组合不支持)
10: 买5价 (组合不支持)
11: (指定价) 模型价 (只对单股情况支持,对组合交易不支持)
12: 涨跌停价
13: 挂单价
14: 对手价
27: 市价即成剩撤(仅对股票期权申报有效)
28: 市价即全成否则撤(仅对股票期权申报有效)
29: 市价剩转限价(仅对股票期权申报有效)
42: 最优五档即时成交剩余撤销申报(仅对上交所申报有效)
43: 最优五档即时成交剩转限价申报(仅对上交所申报有效)
44: 对手方最优价格委托(仅对深交所申报有效)
45: 本方最优价格委托(仅对深交所申报有效)
46: 即时成交剩余撤销委托(仅对深交所申报有效)
47: 最优五档即时成交剩余撤销委托(仅对深交所申报有效)
48: 全额成交或撤销委托(仅对深交所申报有效)
49: 科创板盘后定价

- price, 下单价格

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

*注:

一、单股下单时, prType 是模型价/科创板盘后定价时 price 有效; 其它情况无效; 即单股时, prType 参数为 11, 49 时被使用。prType 参数不为 11, 49 时也需填写, 填写的内容可为 -1, 0, 2, 100 等任意数字;

二、组合下单时, 是组合套利时, price 作套利比例有效, 其它情况无效。

- volume, 下单数量 (股 / 手 / 元 / %)

passorder(opType, orderType, accountID, orderCode, prType, price, volume, ContextInfo)

根据 orderType 值最后一位确定 volume 的单位:

单股下单时:

- 1: 股 / 手
- 2: 金额 (元)
- 3: 比例 (%)

组合下单时:

1: 按组合股票数量 (份)

2: 按组合股票权重 (元)

3: 按账号可用 (%)

- strategyName, string, 自定义策略名, 可缺省不写, 用来区分 order 委托和 deal 成交来自不同的策略。根据该策略名, get_trade_detail_data, get_last_order_id 函数可以获取相应策略名对应的委托或持仓结果。

*注: strategyName 只对同账号本地客户端有效, 即 strategyName 只对当前客户端下的单进行策略区分, 且该策略区分只能当前客户端使用。

- quickTrade, int, 设定是否立即触发下单, 可选值:

0: 否

1: 是

*注: passorder是对最后一根K线完全走完后生成的模型信号在下一根K线的第一个tick数据来时触发下单交易; 采用quickTrade参数设置为1时, 非历史bar上执行时 (ContextInfo.is_last_bar() 为True), 只要策略模型中调用到就触发下单交易。quickTrade参数设置为2时, 不判断bar状态, 只要策略模型中调用到就触发下单交易, 历史bar上也能触发下单, 请谨慎使用。

- userOrderId, string, 用户自设委托 ID, 可缺省不写, 写的时候必须把起前面的 strategyName 和 quickTrade 参数也填写。对应 order 委托对象和 deal 成交对象中的 m_strRemark 属性, 通过 get_trade_detail_data 函数或委托主推函数 order_callback 和成交主推函数 deal_callback 可拿到这两个对象信息。
- userOrderParam, string, 用户自定义交易参数, 主要用于修改算法交易的参数。

m_eOrderType普通交易:0算法交易:1、随机量交易:2

m_ePriceType报价方式:数值同pyType

m_nMaxOrderCount最大下单次数

m_bSinglePriceRange波动区间是否单向

m_ePriceRangeType波动区间类型按比例:0,按数值1

m_dPriceRangeValue波动区间(按数值)[0-1]

m_dPriceRangeRate波动区间(按比例)

m_eSuperPriceType单笔超价类型按比例:0,按数值1

m_dSuperPriceRate单笔超价(按比例)[0-1]

m_dSuperPriceValue单笔超价(按数值)

m_eVolumeType单笔基准量类型(与界面列表的序号一致)卖1+2+3+4+5量:0,卖1+2+3+4量:1,...目标剩余量:11,

m_dVolumeRate单比下单比率

m_nSingleNumMin单比下单量最小值

m_nSingleNumMax单比下单量最大值

m_nValidTimeElapse有效持续时间

m_eUndealtEntrustRule未成委托处理数值同pyType

m_bUseTrigger是否触价

m_eTriggerType触价类型:最新价大于:0,最新价小于:2

m_dTriggerPrice触价价格

示例：

```
1 | userparam="m_nMaxOrderCount:20,m_eSuperPriceType:1,m_dSuperPriceValue:1.12"
2 | algo_passorder(23,1101,ContextInfo.accid,'000001.sz',11,15,1000,'',1,'strReMa
  rk',userparam,ContextInfo);
3 | #表示修改算法交易的最大委托次数为20,单笔下单基准类型为按价格类型超价,单笔超价1.12元,其他参
  数同函数交易参数中设置
```

(32) 获取股票篮子 get_basket()

用法： get_basket(basketName)

释义： 获取股票篮子

参数：

- basketName：股票篮子名称

示例：

```
1 | print( get_basket('basket1') )
```

(33) 设置股票篮子 set_basket()

用法： set_basket(basketDict)

释义： 设置passorder的股票篮子,仅用于passorder进行篮子交易,设置成功后,用get_basket可以取出后即可进行passorder组合交易下单

参数：

- basketDict：股票篮子 {'name':股票篮子名称,'stocks':[{'stock':股票名称,'weight':权重,'quantity':数量,'optType':交易类型}]}。

示例：

```
1 | print( get_basket('basket1') )
```

(34) 获取未了结负债合约明细 get_unclosed_compacts()

用法： get_unclosed_compacts(accountID,accountType)

释义： 获取未了结负债合约明细

参数：

- accountID：str，资金账号
- accountType：str，账号类型

返回：

list([CStkUnclosedCompacts, ...]) 负债列表，CStkUnclosedCompacts属性如下：

```
m_strAccountID - str 账号ID
```

m_nBrokerType - int 账号类型(1-期货账号,2-股票账号,3-信用账号,5-期货期权账号,6-股票期权账号,7-沪港通账号,11-深港通账号)

m_strExchangeID - str 市场

m_strInstrumentID - str 证券代码

m_eCompactType - int 合约类型(32-不限制,48-融资,49-融券)

m_eCashgroupProp - int 头寸来源(32-不限制,48-普通头寸,49-专项头寸)

m_nOpenDate - int 开仓日期(如'20201231')

m_nBusinessVol - int 合约证券数量

m_nRealCompactVol - int 未还合约数量

m_nRetEndDate - int 到期日(如'20201231')

m_dBusinessBalance - float 合约金额

m_dBusinessFare - float 合约息费

m_dRealCompactBalance - float 未还合约金额

m_dRealCompactFare - float 未还合约息费

m_dRepaidFare - float 已还息费

m_dRepaidBalance - float 已还金额

m_strCompactId - str 合约编号

m_strEntrustNo - str 委托编号

m_nRepayPriority - int 偿还优先级

m_strPositionStr - str 定位串

m_eCompactRenewalStatus - int 合约展期状态(48-可申请,49-已申请,50-审批通过,51-审批不通过,52-不可申请,53-已执行,54-已取消)

m_nDeferTimes - int 展期次数

示例：

```
1 | get_unclosed_compacts('6000000248', 'CREDIT')
```

(35) 获取已了结负债合约明细 `get_closed_compacts()`

用法： `get_closed_compacts(accountID,accountType)`

释义： 获取已了结负债合约明细

参数：

- accountID: str, 资金账号
- accountType: str, 账号类型

返回：

`list([CStkUnclosedCompacts, ...])` 负债列表, `CStkUnclosedCompacts`属性如下:

m_strAccountID - str 账号ID
m_nBrokerType - int 账号类型(1-期货账号,2-股票账号,3-信用账号,5-期货期权账号,6-股票期权账号,7-沪港通账号,11-深港通账号)
m_strExchangeID - str 市场
m_strInstrumentID - str 证券代码
m_eCompactType - int 合约类型(32-不限制,48-融资,49-融券)
m_eCashgroupProp - int 头寸来源(32-不限制,48-普通头寸,49-专项头寸)
m_nOpenDate - int 开仓日期(如'20201231')
m_nBusinessVol - int 合约证券数量
m_nRetEndDate - int 到期日(如'20201231')
m_nDateClear - int 了结日期(如'20201231')
m_nEntrustVol - int 委托数量
m_dEntrustBalance - float 委托金额
m_dBusinessBalance - float 合约金额
m_dBusinessFare - float 合约息费
m_dRepaidFare - float 已还息费
m_dRepaidBalance - float 已还金额
m_strCompactId - str 合约编号
m_strEntrustNo - str 委托编号
m_strPositionStr - str 定位串

示例：

```
1 | get_closed_compacts('6000000248', 'CREDIT')
```

(36) 获取沪深港通汇率数据 get_hkt_exchange_rate()

用法： get_hkt_exchange_rate(accountID,accountType)

释义： 获取沪深港通汇率数据

参数：

- accountID: string,账号;
- accountType:string,账号类型,必须填HUGANGTONG或者SHENGANGTONG

返回：

dict,字段释义：

bidReferenceRate:买入参考汇率

askReferenceRate:卖出参考汇率

dayBuyRiseRate:日间买入参考汇率浮动比例

daySaleRiseRate:日间卖出参考汇率浮动比例

示例:

```
1 def init(ContextInfo):
2     data=get_hkt_exchange_rate('6000000248','HUGANGTONG')
3     print(data)
```

(37) 取可融券明细 get_enable_short_contract()

用法: get_enable_short_contract(accountID)

释义: 取可融券明细

参数:

- accountID: string,账号

返回: list,list中放的是PythonObj,通过dir(pythonobj)可返回某个对象的属性列表

示例:

```
1 def handlebar(ContextInfo):
2     obj_list = get_enable_short_contract('6000000248')
3     for obj in obj_list:
4         print( obj.m_strInstrumentName)
```

(38) 取期权标的持仓 get_option_subject_position()

用法: get_option_subject_position(accountID)

释义: 取期权标的持仓

参数:

- accountID: string,账号

返回: list,list中放的是PythonObj,通过dir(pythonobj)可返回某个对象的属性列表

示例:

```
1 data=get_option_subject_position('880399990383')
2 print(len(data));
3 for obj in data:
4     print(obj.m_strInstrumentName,obj.m_lockVol,obj.m_coveredVol);
```

(39) 取期权组合持仓 get_comb_option()

用法: get_comb_option(accountID)

释义: 取期权组合持仓

参数:

- accountID: string,账号

返回: list,list中放的是PythonObj,通过dir(pythonobj)可返回某个对象的属性列表

示例:

```
1 obj_list=get_comb_option('880399990383')
2 print(len(obj_list));
3 for obj in obj_list:
4
    print(obj.m_strCombCodeName,obj.m_strCombID,obj.m_nVolume,obj.m_nFrozenVolume)
```

(40) 构建期权组合持仓 make_option_combination()

用法:

make_option_combination(combType,orderCodeDict,modelVolume,accountID,strategyName,userOrderID,ContextInfo)

释义: 构建期权组合持仓

参数:

- combType: 组合策略类型

- 50:认购牛市价差策略
- 51:认沽熊市价差策略
- 52:认沽牛市价差策略
- 53:认购熊市价差策略
- 54:跨式空头
- 55:宽跨式空头
- 56:保证金开仓转备兑开仓
- 57:备兑开仓转保证金开仓

- orderCodeDict: 期权组合, {option:holdType}。其中, option:期权代码,格式如10000001.SHO, holdType:

- 48:权利
- 49:义务
- 50:备兑

- modelVolume: 下单量
- strategyName: 策略名
- userOrderId: 投资备注

示例:

```
1 ContextInfo.accid='880399990383'
2 make_option_combination(50,
3 {'10003006.SH0':48,'10003259.SH0':49},1,ContextInfo.accid,'stragegyName','str
4 Remark',ContextInfo);
5 #第一个参数50为认购牛市价差策略
#第二个参数10003006.SH0(50ETF购6月3400),48(权利仓)10003259.SH0(50ETF购6月
4400)49(义务仓)
#50ETF购6月3400/50ETF购6月4400g构建认购牛市价差策略
```

(41) 解除期权组合持仓 release_option_combination()

用法: release_option_combination(combID,accountID,strategyName,userOrderId,contextInfo)

释义: 解除期权组合持仓

参数:

- combID:持仓中期权组合编码
- accountID: string,账号
- strategyName:string,策略名
- userOrderId:string,投资备注

示例:

```
1 ContextInfo.accid='880399990383'
2 release_option_combination('v950034404',ContextInfo.accid,'stragegyName','use
rOrderId',ContextInfo)
3 #解除组合号码为v950034404的组合期权(同组合策略持仓中的组合号码)
```

3.2.5. 成交回报实时主推函数

*注: 成交回报实时主推函数仅在实盘运行模式下生效。

(1) 资金账号状态变化主推 account_callback()

用法: account_callback(ContextInfo, accountInfo)

释义: 当资金账号状态有变化时, 运行这个函数

参数:

- ContextInfo: 特定对象
- accountInfo: 资金账号对象, 可对应查看[5.3. 附录3 交易函数内含属性说明](#)

返回: 无

示例:

```
1 def init(ContextInfo):
2     # 设置对应的资金账号
3     ContextInfo.set_account('6000000058')
4
5 def handlebar(ContextInfo):
6     pass
7
8 def account_callback(ContextInfo, accountInfo):
9     print('accountInfo')
10    print(accountInfo.m_strStatus) # m_strStatus 为资金账号的属性之一，表示资金账号的状态
```

(2) 账号委托状态变化主推 order_callback()

用法: order_callback(ContextInfo, orderInfo)

释义: 当账号委托状态有变化时，运行这个函数

参数:

- ContextInfo: 特定对象
- orderInfo: 委托账号对象，可对应查看[5.3. 附录3 交易函数内含属性说明](#)

返回: 无

示例:

```
1 def init(ContextInfo):
2     # 设置对应的资金账号
3     ContextInfo.set_account('6000000058')
4
5 def handlebar(ContextInfo):
6     pass
7
8 def order_callback(ContextInfo, orderInfo):
9     print('orderInfo')
```

(3) 账号成交状态变化主推 deal_callback()

用法: deal_callback(ContextInfo, dealInfo)

释义: 当账号成交状态有变化时，运行这个函数

参数:

- ContextInfo: 特定对象
- dealInfo: 资金账号对象，可对应查看[5.3. 附录3 交易函数内含属性说明](#)

返回: 无

示例:

```
1 def init(ContextInfo):
2     # 设置对应的资金账号
3     ContextInfo.set_account('6000000058')
4
5 def handlebar(ContextInfo):
6     pass
7
8 def deal_callback(ContextInfo, dealInfo):
9     print('dealInfo')
```

(4) 账号持仓状态变化主推 position_callback()

用法: position_callback(ContextInfo, positonInfo)

释义: 当账号持仓状态有变化时，运行这个函数

参数:

- ContextInfo: 特定对象
- positonInfo: 资金账号对象，可对应查看[5.3. 附录3 交易函数内含属性说明](#)

返回: 无

示例:

```
1 def init(ContextInfo):
2     # 设置对应的资金账号
3     ContextInfo.set_account('6000000058')
4
5 def handlebar(ContextInfo):
6     pass
7
8 def position_callback(ContextInfo, positonInfo):
9     print('positonInfo')
```

(5) 账号异常下单主推 orderError_callback()

用法: orderError_callback(ContextInfo,orderArgs,errMsg)

释义: 当账号下单异常时，运行这个函数

参数:

- ContextInfo: 特定对象
- orderArgs: 下单参数，可对应查看[\[5.3. 附录3 交易函数内含属性说明\]](#)
- errMsg: 错误信息

返回: 无

示例:

```
1 def init(ContextInfo):
2     # 设置对应的资金账号
3     ContextInfo.set_account('6000000058')
4
5 def handlebar(ContextInfo):
6     pass
7
8 def position_callback(ContextInfo,orderArgs,errMsg):
9     print('orderArgs')
10    print(errMsg)
```

(6) 查询信用账户明细 query_credit_account()

用法: query_credit_account(accountId,seq,ContextInfo)

释义: 查询信用账户明细。本函数只能有一个查询，如果前面的查询正在进行中，后面的查询将会提前返回。本函数从服务器查询数据，建议平均查询时间间隔30s一次，不可频繁调用。

参数:

- accountId: string, 查询的两融账号
- seq: int, 查询序列号，建议输入唯一值以便对应结果回调

示例:

```
1 query_credit_account(ContextInfo.accid,int(time.time()),ContextInfo);
2      # 查询accid账号某时刻资金账号详细
```

(7) 查询信用账户明细回调 credit_account_callback()

用法: credit_account_callback(ContextInfo,seq,result)

释义: 查询信用账户明细回调

参数:

- ContextInfo: 策略模型全局对象
- seq:query_credit_account时输入查询seq
- result:查询到的结果

(8) 查询两融最大可下单量 query_credit_opvolume()

用法: query_credit_opvolume(accountId,stockCode,opType,prType,price,seq,ContextInfo)

释义: 查询两融最大可下单量。本函数一次最多查询200只股票的两融最大下单量，且同时只能有一个查询,如果前面的查询正在进行中,后面的查询将会提前返回。本函数从服务器查询数据,建议平均查询时间间隔30s一次,不可频繁调用。

参数:

- accountId:查询的两融账号
- stockCode:需要查询的股票代码,stockCode为List的类型,可以查询多只股票
- opType:两融下单类型,同passorder的下单类型

- prType:报单价格类型,同passorder的报价类型
- seq:查询序列号,int型,建议输入唯一值以便对应结果回调
- price:报价(非限价单可以填任意值),如果stockCode为List类型,报价也需要为长度相同的List

示例:

```

1 | query_credit_opvolume(ContextInfo.accid, '600000.SH', 33, 11, 10, int(time.time()))
   ,ContextInfo);
2 | #查询accid账号担保品买入600000.SH限价10元的最大可下单量
3 | query_credit_opvolume(ContextInfo.accid, ['600000.SH', '000001.SZ'], 33, 11,
   [10,20], int(time.time()), ContextInfo);
4 | #查询accid账号担保品买入600000.SH限价10元,000001.SZ担保品买入限价20元的最大可下单量

```

(9) 查询两融最大可下单量的回调 credit_opvolume_callback()

用法: credit_opvolume_callback(ContextInfo,accid,seq,ret,result)

释义: 查询两融最大可下单量的回调。

参数:

- ContextInfo: 策略模型全局对象
- accid:查询的账号
- seq:query_credit_opvolume时输入查询seq
- ret:查询结果状态。正常返回:1,正在查询中:-1,输入账号非法:-2,输入查询参数非法:-3,超时等服务器回报错:-4
- result:查询到的结果

示例:

```

1 | query_credit_account(ContextInfo.accid, int(time.time()), ContextInfo);
2 | # 查询accid账号某时刻资金账号详细

```

3.2.6. 引用函数

*注: 以下函数均支持回测和实盘/模拟运行模式。

(1) 获取扩展数据 ext_data()

用法: ext_data(extdataname, stockcode, deviation, ContextInfo)

释义: 获取扩展数据

参数:

- extdataname: string, 扩展数据名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- deviation: number, K线偏移, 可取值:

0: 不偏移

N: 向右偏移N

-N: 向左偏移N

- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ext_data('mycci', '600000.SH', 0, ContextInfo))
```

(2) 获取引用的扩展数据的数值在所有品种中的排名 ext_data_rank()

用法: ext_data_rank(extdataname, stockcode, deviation, ContextInfo)

释义: 获取引用的扩展数据的数值在所有品种中的排名

参数:

- extdataname: string, 扩展数据名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- deviation: number, K 线偏移, 可取值:

 0: 不偏移

 N: 向右偏移N

 -N: 向左偏移N

- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(ext_data_rank('mycci', '600000.SH', 0, ContextInfo))
```

(3) 获取引用的扩展数据的数值在指定时间区间内所有品种中的排名 ext_data_rank_range()

用法: ext_data_rank_range(extdataname, stockcode, begintime, endtime, ContextInfo)

释义: 获取引用的扩展数据的数值在指定时间区间内所有品种中的排名

参数:

- extdataname: string, 扩展数据名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- begintime: string, 区间的起始时间 (包括该时间点在内) 格式为 '2016-08-02 12:12:30'
- endtime: string, 区间的结束时间 (包括该时间点在内) 格式为 '2017-08-02 12:12:30'
- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: pythonDict

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ext_data_rank_range('mycci', '600000.SH', '2016-08-02 12:12:30',  
|       '2017-08-02 12:12:30', ContextInfo))
```

(4) 获取扩展数据在指定时间区间内的值 ext_data_range()

用法: ext_data_range(extdataname, stockcode, begintime, endtime, ContextInfo)

释义: 获取扩展数据在指定时间区间内的值

参数:

- extdataname: string, 扩展数据名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- begintime: string, 区间的起始时间 (包括该时间点在内) 格式为 '2016-08-02 12:12:30'
- endtime: string, 区间的结束时间 (包括该时间点在内) 格式为 '2017-08-02 12:12:30'
- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: pythonDict

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(ext_data_range('mycci', '600000.SH', '2016-08-02 12:12:30', '2017-  
|       '08-02 12:12:30', ContextInfo))
```

(5) 获取因子数据 get_factor_value()

用法: get_factor_value(factorname, stockcode, deviation, ContextInfo)

释义: 获取因子数据

参数:

- factorname: string, 因子名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- deviation: number, K 线偏移, 0 不偏移, N 向右偏移 N, -N 向左偏移 N
- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: number

示例:

```
1 | def handlebar(ContextInfo):  
2 |     print(get_factor_value('zzz', '600000.SH', 0, ContextInfo))
```

(6) 获取引用的因子数据的数值在所有品种中排名 get_factor_rank()

用法: get_factor_rank(factorname, stockcode, deviation, ContextInfo)

释义: 获取引用的因子数据的数值在所有品种中排名

参数:

- factorname: string, 因子名
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- deviation: number, K 线偏移, 0不偏移, N向右偏移N, -N向左偏移N
- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     print(get_factor_rank('zzz', '600000.SH', 0, ContextInfo))
```

(7) 获取引用的 VBA 模型运行的结果 call_vba()

用法: call_vba(factorname, stockcode, [period, dividend_type, barpos,]ContextInfo)

释义: 获取引用的 VBA 模型运行的结果

*注: 使用该函数时需补充好本地 K 线或分笔数据

参数:

- factorname: string, 模型名及引用变量, 如 'MA.ma1'
- stockcode: string, 形式如 'stkcodemarket', 如 '600000.SH'
- period: string, K 线周期类型, 可缺省, 默认为当前主图周期线型, 可选值:

```
'tick': 分笔线  
'1d': 日线  
'1m': 1分钟线  
'3m': 3分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'30m': 30分钟线  
'1h': 小时线  
'1w': 周线  
'1mon': 月线  
'1q': 季线  
'1hy': 半年线  
'1y': 年线
```

- dividend_type: string, 除复权, 可缺省, 默认当前图复权方式, 可选值:

```
'none': 不复权  
'front': 向前复权  
'back': 向后复权  
'front_ratio': 等比向前复权  
'back_ratio': 等比向后复权
```

- barpos: number, 对应 bar 下标, 可缺省, 默认当前主图调用到的 bar 的对应下标

- ContextInfo: pythonObj, Python 对象, 这里必须是 ContextInfo

返回: number

示例:

```
1 | def handlebar(ContextInfo):
2 |     call_vba('MA.ma1', '600036.SH', ContextInfo)
```

(8) 调用VBA组合模型call_formula()

用法:

```
call_formula(formulaName,code,period,divideType,basket,argsDict,startime="",endTime="",count=-1)
```

释义: python调用组合模型, 返回GruopModelInfo结构体

*注: 使用该函数时需补充好本地 K 线或分笔数据

参数:

- formulaName: string, 组合模型名
- stockcode: string, 组合模型主图代码形式如'stkcode.market',如'000300.SH'
- period: string, K 线周期类型, 可缺省, 默认为当前主图周期线型, 可选值:

```
'tick': 分笔线  
'1d': 日线  
'1m': 1分钟线  
'3m': 3分钟线  
'5m': 5分钟线  
'15m': 15分钟线  
'30m': 30分钟线  
'1h': 小时线  
'1w': 周线  
'1mon': 月线  
'1q': 季线  
'1hy': 半年线  
'1y': 年线
```

- dividend_type: string, 除复权, 可缺省, 默认当前图复权方式, 可选值:

```
'none': 不复权  
'front': 向前复权  
'back': 向后复权  
'front_ratio': 等比向前复权  
'back_ratio': 等比向后复权
```

- basket: dict, 组合模型的股票池权重, 形如{'SH600000':0.06,'SZ000001':0.01}

- argsDict: dict, 组合模型的入参, {参数名:参数值}, 形如{'a':1}
- startTime: string, 缺失参数, 组合模型运行起始时间, 形如:'20200101'
- endTime: string, 缺失参数, 组合模型运截止时间, 形如:'20200101'
- count: int, 缺省参数, 组合模型运行范围为向前count根bar

返回: GruopModellInfo结构体。GruopModellInfo.result为当前模型的运行结果, 运行GruopModellInfo.unsubscribe()会取消组合模型的运行, GruopModellInfo随着python的析构也会调用取消组合模型运行。

示例:

```

1 def handlebar(ContextInfo):
2     basket={'SH600000':0.06,'SZ000001':0.01}
3     argsDict={'a':100}
4
5     groupModelInfo=call_formula('testGroupModel','000300.SH',basket,argsDict);
6     print(groupModelInfo.result);
7     groupModelInfo.unsubscribe();#取消组合模型运行

```

3.2.7. 绘图函数

*注: 以下函数均支持回测和实盘/模拟运行模式。

(1) 在界面上画图 ContextInfo.paint()

用法: ContextInfo.paint(name, value, index, line_style, color = 'white', limit = "")

释义: 在界面上画图

参数:

- name: string, 需显示的指标名
- value: number, 需显示的数值
- index: number, 显示索引位置, 填 -1 表示按主图索引显示
- line_style: number, 线型, 可取值:

0: 曲线

42: 柱状线

- color: string, 颜色 (不填默认为白色) 目前支持以下几种颜色:

blue: 蓝

brown: 棕

cyan: 蓝绿

green: 绿

magenta: 品红

red: 红

white: 白

yellow: 黄

- limit: string, 画线控制, 可取值:

'noaxis': 不影响坐标画线

'nodraw' : 不画线

返回: 无

示例:

```
1 | def handlebar(ContextInfo):  
2 |     realtimetag = ContextInfo.get_bar_timetag(ContextInfo.barpos)  
3 |     value = ContextInfo.get_close_price('', '', realtimetag)  
4 |     ContextInfo.paint('close', value, -1, 0, 'white','noaxis')
```

(2) 在图形上显示文字 ContextInfo.draw_text()

用法: ContextInfo.draw_text(condition, position, text)

释义: 在图形上显示数字

参数:

- condition: 条件
- Position: 位置
- text: 文字

返回: 无

示例:

```
1 | def handlebar(ContextInfo):  
2 |     ContextInfo.draw_text(1, 10, '文字')
```

(3) 在图形上显示数字 ContextInfo.draw_number()

用法: ContextInfo.draw_number(cond, height, number, precision)

释义: 在图形上显示数字

参数:

- cond: bool, 条件
- height: number, 显示文字的高度位置
- text: string, 显示的数字
- precision: 为小数显示位数 (取值范围 0 - 7)

返回: 无

示例:

```
1 | def handlebar(ContextInfo):  
2 |     close = ContextInfo.get_market_data(['close'])  
3 |     ContextInfo.draw_number(1 > 0, close, 66, 1)
```

(4) 在数字 1 和数字 2 之间绘垂直线 ContextInfo.draw_vertline()

用法: ContextInfo.draw_vertline(cond, number1, number2, color = "", limit = "")

释义: 在数字1和数字2之间绘垂直线

参数:

- cond: bool, 条件
- number1: number, 数字1
- number2: number, 数字2
- color: string, 颜色 (不填默认为白色) 目前支持以下几种颜色:

blue: 蓝

brown: 棕

cyan: 蓝绿

green: 绿

magenta: 品红

red: 红

white: 白

yellow: 黄

- limit: string, 画线控制, 可取值:

'noaxis': 不影响坐标画线

'nodraw' : 不画线

返回: 无

示例:

```
1 | def handlebar(ContextInfo):  
2 |     close = ContextInfo.get_market_data(['close'])  
3 |     open = ContextInfo.get_market_data(['open'])  
4 |     ContextInfo.draw_vertline(1 > 0, close, open, 'cyan')
```

(5) 在图形上绘制小图标 ContextInfo.draw_icon()

用法: ContextInfo.draw_icon(cond, height, type)

释义: 在图形上绘制小图标

参数:

- cond: bool, 条件
- height: number, 图标的位罝
- type: number, 图标的类型, 可取值:

1: 椭圆

0: 矩形

返回: 无

示例：

```
1 | def handlebar(ContextInfo):
2 |     close = ContextInfo.get_market_data(['close'])
3 |     ContextInfo.draw_icon(1 > 0, close, 0)
```

4. 财务数据接口使用方法

财务数据接口通过读取下载本地的数据取数，使用前需要补充本地数据。除公告日期和报表截止日期为时间戳毫秒格式其他单位为元或%，数据主要包括资产负债表(ASHAREBALANCESHEET)、利润表(ASHAREINCOME)、现金流量表(ASHARECASHFLOW)、股本表(CAPITALSTRUCTURE)的主要字段数据以及经过计算的主要财务指标数据(PERSHAREINDEX)。建议使用本文档对照表中的英文表名和迅投英文字段。

4.1. Python接口

4.1.1. 用法1

```
ContextInfo.get_financial_data(fieldList, stockList, startDate, endDate, report_type =  
'announce_time')
```

字段名	类型	释义与用例
fieldList	List (必须)	财报字段列表: ['ASHAREBALANCESHEET.fix_assets', '利润表.净利润']
stockList	List (必须)	股票列表: ['600000.SH', '000001.SZ']
startDate	Str (必须)	开始时间: '20171209'
endDate	Str (必须)	结束时间: '20171212'
report_type	Str (可选)	报表时间类型, 可缺省, 默认是按照数据的公告期为区分取数据, 设置为'report_time'为按照报告期取数据, 'announce_time'为按照公告日期取数据

返回：

函数根据stockList代码列表,startDate,endDate时间范围, 返回不同的的数据类型。如下：

- (1) 代码列表 1 时间范围为 1, 返回: pandas.Series index = 字段
- (2) 代码列表 1 时间范围为 n, 返回: pandas.DataFrame index = 时间, columns = 字段
- (3) 代码列表 n 时间范围为 1, 返回: pandas.DataFrame index = 代码, columns = 字段

(4) 代码列表 n 时间范围为 n, 返回: pandas.Panel items = 代码, major_axis = 时间, minor_axis = 字段

示例:

```
1 def handlebar(ContextInfo):
2     #取总股本和净利润
3     fieldList = ['CAPITALSTRUCTURE.total_capital', '利润表.净利润']
4     stockList = ['600000.SH', '000001.SZ']
5     startDate = '20171209'
6     endDate = '20171212'
7     ContextInfo.get_financial_data(fieldList, stockList, startDate, endDate,
report_type = 'report_time')
```

*注:

选择按照公告期取数和按照报告期取数的区别:

若某公司当年 4 月 26 日发布上年度年报, 如果选择按照公告期取数, 则当年 4 月 26 日之后至下个财报发布日期之间的数据都是上年度年报的财务数据。

若选择按照报告期取数, 则上年度第 4 季度 (上年度 10 月 1 日 - 12 月 31 日) 的数据就是上年度报告期的数据。

4.1.2. 用法2

ContextInfo.get_financial_data(tabname, colname, market, code, report_type = 'report_time', barpos) (与用法 1 可同时使用)

字段名	类型	释义与用例
tabname	Str (必须)	表名: 'ASHAREBALANCESHEET'
colname	Str (必须)	字段名: 'fix_assets'
market	Str (必须)	市场: 'SH'
code	Str (必须)	代码: '600000'
report_type	Str (可选)	报表时间类型, 可缺省, 默认是按照数据的公告期为区分取数据, 设置为 'report_time' 为按照报告期取数据, 'announce_time' 为按照公告日期取数据
barpos	number	当前 bar 的索引

返回:

number

示例:

```
1 | def handlebar(ContextInfo):
2 |     index = ContextInfo.barpos
3 |     ContextInfo.get_financial_data('ASHAREBALANCESHEET', 'fix_assets', 'SH',
|       '600000', index);
```

4.2. vba接口

4.2.1. 用法1

根据公告期获取报表原始数据:

```
getfindata('表格名称', '字段名称')
```

示例:

固定资产:

```
1 | getfindata('ASHAREBALANCESHEET', 'fix_assets')
```

4.2.2. 用法2

获取每年固定某个季度的数据，该年所有数据都是该季度的数据:

```
getfindata('表格名称', '字段名称', session)
```

示例:

固定资产:

```
1 | //获取每年三季报
2 | getfindata('ASHAREBALANCESHEET', 'fix_assets', 3)
```

4.3. 财务数据字段对照表

4.3.1. 资产负债表 (ASHAREBALANCESHEET)

中文字段	迅投字段
应收利息	int_rcv
可供出售金融资产	fin_assets_avail_for_sale
持有至到期投资	held_to_mty_invest
长期股权投资	long_term_eqy_invest
固定资产	fix_assets
无形资产	intang_assets
递延所得税资产	deferred_tax_assets
资产总计	tot_assets
交易性金融负债	tradable_fin_liab
应付职工薪酬	empl_ben_payable
应交税费	taxes_surcharges_payable
应付利息	int_payable
应付债券	bonds_payable
递延所得税负债	deferred_tax_liab
负债合计	tot_liab
实收资本(或股本)	cap_stk
资本公积金	cap_rsrv
盈余公积金	surplus_rsrv
未分配利润	undistributed_profit
归属于母公司股东权益合计	tot_shrhldr_eqy_excl_min_int
少数股东权益	minority_int
负债和股东权益总计	tot_liab_shrhldr_eqy
所有者权益合计	total_equity
货币资金	cash_equivalents
应收票据	bill_receivable
应收账款	account_receivable
预付账款	advance_payment
其他应收款	other_receivable
其他流动资产	other_current_assets

中文字段	迅投字段
流动资产合计	total_current_assets
存货	inventories
在建工程	constru_in_process
工程物资	construction_materials
长期待摊费用	long_deferred_expense
非流动资产合计	total_non_current_assets
短期借款	shortterm_loan
应付股利	dividend_payable
其他应付款	other_payable
一年内到期的非流动负债	non_current_liability_in_one_year
其他流动负债	other_current_liability
长期应付款	longterm_account_payable
应付账款	accounts_payable
预收账款	advance_peceipts
流动负债合计	total_current_liability
应付票据	notes_payable
长期借款	long_term_loans
专项应付款	grants_received
其他非流动负债	other_non_current_liabilities
非流动负债合计	non_current_liabilities
专项储备	specific_reserves
商誉	goodwill
报告截止日	m_timetag
公告日	m_anntime

4.3.2. 利润表 (ASHAREINCOME)

中文字段	迅投字段
投资收益	plus_net_invest_inc
联营企业和合营企业的投资收益	incl_inc_invest_assoc_jv_entp
营业税金及附加	less_taxes_surcharges_ops
营业总收入	revenue
营业总成本	total_operating_cost
营业收入	revenue_inc
营业成本	total_expense
资产减值损失	less_impair_loss_assets
营业利润	oper_profit
营业外收入	plus_non_oper_rev
营业外支出	less_non_oper_exp
利润总额	tot_profit
所得税	inc_tax
净利润	net_profit_incl_min_int_inc
归属净利润	net_profit_excl_min_int_inc
管理费用	less_genl_admin_exp
销售费用	sale_expense
财务费用	financial_expense
综合收益总额	total_income
归属于少数股东的综合收益总额	total_income_minority
公允价值变动收益	change_income_fair_value
已赚保费	earned_premium
报告截止日	m_timetag
公告日	m_anntime

4.3.3. 现金流量表 (ASHARECASHFLOW)

中文字段	迅投字段
收到其他与经营活动有关的现金	other_cash_recp_ral_oper_act
经营活动现金流入小计	stot_cash_inflows_oper_act
支付给职工以及为职工支付的现金	cash_pay_beh_empl
支付的各项税费	pay_all_typ_tax
支付其他与经营活动有关的现金	other_cash_pay_ral_oper_act
经营活动现金流出小计	stot_cash_outflows_oper_act
经营活动产生的现金流量净额	net_cash_flows_oper_act
取得投资收益所收到的现金	cash_recp_return_invest
处置固定资产、无形资产和其他长期投资收到的现金	net_cash_recp_disp_fiolta
投资活动现金流入小计	stot_cash_inflows_inv_act
投资支付的现金	cash_paid_invest
购建固定资产、无形资产和其他长期投资支付的现金	cash_pay_acq_const_fiolta
支付其他与投资的现金	other_cash_pay_ral_inv_act
投资活动产生的现金流出小计	stot_cash_outflows_inv_act
投资活动产生的现金流量净额	net_cash_flows_inv_act
吸收投资收到的现金	cash_recp_cap_contrib
取得借款收到的现金	cash_recp_borrow
收到其他与筹资活动有关的现金	other_cash_recp_ral_fnc_act
筹资活动现金流入小计	stot_cash_inflows_fnc_act
偿还债务支付现金	cash_prepay_amt_borr
分配股利、利润或偿付利息支付的现金	cash_pay_dist_dpcp_int_exp
支付其他与筹资的现金	other_cash_pay_ral_fnc_act
筹资活动现金流出小计	stot_cash_outflows_fnc_act
筹资活动产生的现金流量净额	net_cash_flows_fnc_act
汇率变动对现金的影响	eff_fx_flu_cash
现金及现金等价物净增加额	net_incr_cash_cash_equ
销售商品、提供劳务收到的现金	goods_sale_and_service_render_cash
收到的税费与返还	tax_levy_refund

中文字段	迅投字段
购买商品、接受劳务支付的现金	goods_and_services_cash_paid
处置子公司及其他收到的现金	net_cash_deal_subcompany
其中子公司吸收现金	cash_from_mino_s_invest_sub
处置固定资产、无形资产和其他长期资产支付的现金净额	fix_intan_other_asset_dispo_cash_payment
报告截止日	m_timetag
公告日	m_anntime

4.3.4. 股本表 (CAPITALSTRUCTURE)

中文字段	迅投字段
总股本	total_capital
已上市流通A股	circulating_capital
限售流通股份	restrict_circulating_capital
报告截止日	m_timetag
公告日	m_anntime

4.3.5. 主要指标 (PERSHAREINDEX)

中文字段	迅投字段
每股经营活动现金流量	s_fa_ocfps
每股净资产	s_fa_bps
基本每股收益	s_fa_eps_basic
稀释每股收益	s_fa_eps_diluted
每股未分配利润	s_fa_undistributedeps
每股资本公积金	s_fa_surpluscapitalps
扣非每股收益	adjusted_earnings_per_share
主营收入	inc_revenue
毛利润	inc_gross_profit
利润总额	inc_profit_before_tax
净利润	du_profit
归属于母公司所有者的净利润	inc_net_profit
扣非净利润	adjusted_net_profit
净资产收益率	du_return_on_equity
销售毛利率	sales_gross_profit
主营收入同比增长	inc_revenue_rate
净利润同比增长	du_profit_rate
归属于母公司所有者的净利润同比增长	inc_net_profit_rate
扣非净利润同比增长	adjusted_net_profit_rate
营业总收入滚动环比增长	inc_total_revenue_annual
归属净利润滚动环比增长	inc_net_profit_to_shareholders_annual
扣非净利润滚动环比增长	adjusted_profit_to_profit_annual
加权净资产收益率	equity_roe
摊薄净资产收益率	net_roe
摊薄总资产收益率	total_roe
毛利率	gross_profit
净利率	net_profit
实际税率	actual_tax_rate
预收款营业收入	pre_pay_operate_income

中文字段	迅投字段
销售现金流营业收入	<code>sales_cash_flow</code>
资产负债比率	<code>gear_ratio</code>
存货周转率	<code>inventory_turnover</code>

5. 附录

5.1. 附录1 市场简称代码

市场	代码
上证所	SH
深交所	SZ
大商所	DF
郑商所	ZF
上期所	SF
中金所	IF
沪港通	HGT
深港通	SGT
外部自定义市场	ED

5.2. 附录2 `is_typed_stock` 函数证券分类表

注：*代表任意阿拉伯数字

5.2.1. 基础类型

类型描述	市场代码	类别号
沪市	*****, SH	1
深市	*****, SZ	2
中金	*****\ *****\ ****\ ***, IF	3
上期	*****\ *****\ ****\ ***, SF	4
大商	*****\ *****\ ****\ ***, DF	5
郑商	*****\ *****\ ****\ ***, ZF	6
开放基金	***** , OF	7
股票期权	***** , SHO	8
新三板	***** , NEEQ	9
沪市A股	60****, SH	101
沪市B股	90****, SH	102
沪市封基	50****, SH	103
沪市指数	000***, SH	104
沪市ETF	510***\ 511***\ 512***\ 513***\ 518***, SH	105
沪市权证	58****, SH	106
沪市申购	73****\ 78****, SH	107
沪市可交換公司債券	132***, SH	108
沪市可交換公司債券質押券出入庫	133***, SH	109
沪市可交換公司債券換股	192***, SH	110
沪市并购重组私募債券挂牌转让	1355**\ 1356**\ 1357**\ 1358**\ 1359**, SH	111
沪市证券公司短期债券挂牌转让	1350**\ 1351**\ 1352**\ 1353**\ 1354**, SH	112
沪市信貸资产支持证券交易 asset-backed securities	128***, SH	113
沪市公司債券質押券入库	102***\ 134***, SH	114
沪市公司債	122***\ 123***\ 124***\ 127***\ 136***, SH	115
沪市公开发行优先股交易	330***, SH	116
沪市非公开发行优先股转让	360***, SH	117
沪市公开发行优先股申购	770***, SH	118
沪市公开发行优先股配股/配售	771***, SH	119
沪市公开发行优先股申购分配	772***, SH	120
沪市公开发行优先股申购配号	773***, SH	121
沪市国债回购(席位托管方式)	201***, SH	122
沪市企业债回购	202***, SH	123
沪市国债买断式回购	203***, SH	124
沪市新质押式国债回购	204***, SH	125
沪市附息国债	010***\ 019***, SH	127
沪市金融债	018***, SH	128
沪市贴现国债	020***, SH	129
沪市中央政府债(国债)	010***\ 019***\ 020***, SH	130
沪市分离债	126***, SH	131
沪市资产证券化	121***, SH	132
沪市信貸资产支持	128***, SH	133
沪市企业债(席位托管方式)	120***\ 129***, SH	134
沪市可转债	100***\ 110***\ 112***\ 113***\ 128***, SH	135
沪市地方债	130***\ 140***, SH	136
沪市政府债(国债+地方债)	010***\ 019***\ 020***\ 130***, SH	137
上海可交换私募债	1380**, SH	138
沪市标准券	888880\ SHRQ88, SH	139
沪市封闭式基金	500***\ 5058**, SH	140
沪市政策性金融债	018***\ 028***\ 038***, SH	141
沪市上海质押代码	09****\ 102***\ 103***\ 104***\ 105***\ 106***\ 107***\ 108***\ 133***, SH	142
2000年前发行国债	009***, SH	143

类型描述	市场代码	类别号
记账式贴现国债质押式回购标准券入库	107***,SH	144
公司债质押式回购标准券入库	1040**\ 1041**\ 1042**\ 1043**\ 1044**,SH	145
国债分销	7510**\ 7511**,SH	146
地方政府债质押式回购标准券入库	106***\ 141***,SH	147
地方政府债分销	75190*\ 75191*\ 75192*\ 75193*\ 75194*\ 75195*\ 75196*,SH	148
分离债质押式回购标准券入库	1050**\ 1051**\ 1052**\ 1053**\ 1054**\ 1055**\ 1056**\ 1057*\ 1058**,SH	149
债券质押式报价回购	205***,SH	150
中小企业私募债券在固定收益平台转让	125***,SH	151
跨境ETF	513030\ 513500\ 513100\ 510900\ 513600\ 513660,SH	152
跨境LOF	,SH	153
上海创新型封闭式基金	5058**,SH	154
上海的固定收益类	511***,SH	155
上海黄金	518**0,SH	156
上海实时申赎货币基金	5198**\ 5195**\ 5199**,SH	157
上海交易型货币基金	5116**\ 5117**\ 5118**\ 5119**,SH	158
上海股票申购代码	730***\ 732***\ 780***,SH	159
上海债券申购代码	733***\ 783***,SH	160
上海基金申购代码	735***,SH	161
上海新股配号	741***\ 791***\ 736***,SH	162
上海配售首发配号	747***\ 797***,SH	163
上海可转债资金申购配号	744***\ 794***\ 756***,SH	164
上海申购款	740***\ 790***\ 734***,SH	165
上海发债款	743***\ 793***\ 755***,SH	166
上海配股代码	700***\ 760***\ 742***,SH	167
上海配转债代码	704***\ 764***\ 753***,SH	168
上海LOF	501***,SH	169
上海分级基金	502***,SH	170
沪新股额	SHXGED,SH	171
沪增发股	730***\ 731***\ 780***\ 781***,SH	172
上海跨境ETF申赎代码	513031\ 513501\ 513101\ 510901\ 513601\ 513661,SH	173
上海债券ETF	511010\ 511210\ 511220,SH	174
上海企业债券挂牌转让	139***,SH	175
上海可交换私募债	1380**,SH	176
沪市1天回购	204001,SH	177
沪市2天回购	204002,SH	178
沪市3天回购	204003,SH	179
沪市4天回购	204004,SH	180
沪市7天回购	204007,SH	181
沪市14天回购	204014,SH	182
沪市28天回购	204028,SH	183
沪市28天以上回购	204091\ 204182,SH	184
上海分级基金子基金	502**1\ 502**2\ 502**4\ 502**5\ 502**7\ 502008\ 502018\ 502028\ 502038\ 502058\ 502049\ 502050,SH	185
深市A股	00****\ 30****,SZ	10001
深市B股	20****,SZ	10002
深市封基	15****\ 16****\ 18****,SZ	10003
深市主板	000***\ 001***,SZ	10004
深市小板	002***\ 003***\ 004***,SZ	10005
深市创业板	30****,SZ	10006
深市指数	39****,SZ	10007
深市ETF	159***,SZ	10008
深市权证	03****,SZ	10009
深市国债回购(131990不是的,需要业务支持)	131990,SH	10010
深市附息国债	100***\ 101***\ 102***\ 103***\ 104***\ 105***\ 106***\ 107***,SZ	10011

类型描述	市场代码	类别号
深市贴现国债	108***,SZ	10012
深市公司债	112***,SZ	10013
深市企业债	111***,SZ	10014
深市分离债	115***,SZ	10015
深市私募债	118***\ 114***,SZ	10016
深市专项资金管理规划	119***,SZ	10017
深市地方政府债	109***,SZ	10018
深市可转债	12****,SZ	10019
深市标准券	131990\ SZRQ88,SZ	10020
深市封闭式基金	184***,SZ	10021
深市LOF	16****,SZ	10022
深市分级基金	150***\ 151***,SZ	10023
深市 中小企业可交换私募债	117***,SZ	10024
深市证券公司次级债	1189**,SZ	10025
深市其他中小企业私募债	1180**\ 1181**\ 1182**\ 1183**\ 1184**\ 1185**\ 1186**\ 1187**\ 1188**,SZ	10026
深市资产支持证券	1190**\ 1191**\ 1192**\ 1193**\ 1194**,SZ	10027
深市分级基金子基金	150***\ 151***,SZ	10028
深市跨境ETF	159920\ 159941,SZ	10029
深市跨境LOF	160125\ 160416\ 160717\ 160719\ 161116\ 161210\ 161714\ 161815\ 162411\ 164701\ 164815\ 165510\ 165513,SZ	10030
深市创新型封闭式基金	150***,SZ	10031
深市主板可转换公司债券	127***,SZ	10032
深市创业板可转换公司债券	123***,SZ	10033
深市中小板可转换公司债券	128***,SZ	10034
深市国债回购(131900不是的,需要业务支持)	131***,SZ	10035
深市黄金	159934,SZ	10036
深市实时申赎货币基金	1590**,SZ	10037
深新股额	SZXGED,SZ	10038
深增发股	07****\ 37***,SZ	10039
深圳配股	08****,SZ	10040
深圳债券ETF 仅此一支	159926,SZ	10041
深市1天回购	131810,SZ	10042
深市2天回购	131811,SZ	10043
深市3天回购	131800,SZ	10044
深市4天回购	131809,SZ	10045
深市7天回购	131801,SZ	10046
深市14天回购	131802,SZ	10047
深市28天回购	131803,SZ	10048
深市28天以上回购	131805\ 131806,SZ	10049
中金所指数期货	IF****\ IH****\ IC***\ IF**\ IH**\ IC**,IF	30001
中金所国债期货	TF****\ T****\ HTFF****\ TF**\ T**\ HTFF**,TF	30002

5.2.2. 扩展类型

类型描述	类型集合	类别标号
沪深A股	沪市A股\ 深市A股	100001
沪深B股	沪市B股\ 深市B股	100002
沪深狭义股票	沪深A股\ 沪深B股	100003
沪深封基	沪市封基\ 深市封基	100004
指数	沪市指数\ 深市指数	100005
商品期货	上期\ 大商\ 郑商	100006
期货市场	中金\ 商品期货	100007
股票	沪市\ 深市	100008
深市中央政府债（国债）	深市附息国债\ 深市贴现国债	100009
深市政府债	深市中央政府债（国债）\ 深市地方政府债	100010
深市所有债券	深市附息国债\ 深市贴现国债\ 深市地方政府债\ 深市可转债\ 深市企业债\ 深市分离债\ 深市私募债\ 深市专项资金管理规划	100011
广义的股票	!指数	100012
ETF	沪市ETF\ 深市ETF	100013
封闭式基金	沪市封闭式基金\ 深市封闭式基金	100014
权证	沪市权证\ 深市权证	100015
债券	上海债券\ 深市所有债券	100016
深市国债回购	深市国债回购(131900不是的，需要业务支持)&(!深市国债回购(131990不是的，需要业务支持))	100017
标准券	沪市标准券\ 深市标准券	100018
债券回购	沪市国债回购（席位托管方式）\ 深市国债回购	100020
质押式回购	沪市新质押式国债回购\ 深市国债回购	100021
黄金	上海黄金\ 深市黄金	100022
实时申赎货币基金	上海实时申赎货币基金\ 深市实时申赎货币基金	100023
货币基金	实时申赎货币基金\ 上海交易型货币基金	100024
上海申购代码	上海股票申购代码\ 上海债券申购代码\ 上海基金申购代码	100025

类型描述	类型集合	类别标号
跨境ETF	深市跨境ETF\ 跨境ETF	100026
跨境LOF	跨境LOF\ 深市跨境LOF	100027
可交易的	沪深A股\ 沪深封基\ ETF\ 权证\ 沪市申购\ 深市创业板\ 债券回购\ 债券ETF\ 上海的固定收益类\ 黄金\ 货币基金\ 深市中央政府债（国债）\ 沪市中央政府债（国债）\ 沪市地方债\ 深市地方政府债\ 上海申购代码\ 沪市上海质押代码\ 跨境ETF\ 跨境LOF\ 上海配股代码\ 上海配转债代码\ 深市可转债\ 沪市可转债\ 沪增发股\ 深增发股	100028
主板	沪市A股\ 深市主板	100029
回转交易	债券\ 黄金\ 上海的固定收益类\ 权证\ 跨境ETF\ 跨境LOF\ 上海交易型货币基金	100030
上海配号	上海新股配号\ 上海配售首发配号\ 上海可转债资金申购配号	100031
沪深新股申购额度	沪新股额\ 深新股额	100032
沪深配股代码	上海配股代码\ 深圳配股	100033
固定收益： 跟踪债券指数的交易型 开放式指数 基金、交易 型货币市场 基金	上海的固定收益类	100034
固定收益类	沪市附息国债\ 深市附息国债\ 沪市贴现国债\ 深市贴现国债\ 沪市政府债（国债+地方债）\ 深市政府债\ 深市企业债\ 沪市企业债（席位托管方式）\ 深市私募债\ 沪市可转债\ 深市可转债\ 沪市分离债\ 深市分离债\ 债券回购\ 标准券\ 货币基金\ 上海的固定收益类	100035
分级基金	上海分级基金\ 深市分级基金	100036
LOF	上海LOF\ 深市LOF	100037
债券ETF	上海债券ETF\ 深圳债券ETF	100038
上海债券	沪市政府债（国债+地方债）\ 沪市可转债\ 沪市公司债\ 沪市企业债（席位托管方式）\ 沪市资产证券化\ 沪市分离债\ 沪市金融债\ 沪市信贷资产支持\ 沪市可交换公司债券\ 沪市并购重组私募债券挂牌转让\ 沪市证券公司短期债券挂牌转让\ 上海可交换私募债\ 上海企业债券挂牌转让\ 上海可交换私募债	100039
1天逆回购	沪市1天回购\ 深市1天回购	100040
2天逆回购	沪市2天回购\ 深市2天回购	100041

类型描述	类型集合	类别标号
3天逆回购	沪市3天回购\ 深市3天回购	100042
4天逆回购	沪市4天回购\ 深市4天回购	100043
7天逆回购	沪市7天回购\ 深市7天回购	100044
14天逆回购	沪市14天回购\ 深市14天回购	100045
28天逆回购	沪市28天回购\ 深市28天回购	100046
28天以上逆回购	沪市28天以上回购\ 深市28天以上回购	100047

5.3. 附录3 交易函数内含属性说明

5.3.1. account 资金账号对象

m_dMaxMarginRate: 保证金比率，股票的保证金率等于 1，股票不需要

m_dFrozenMargin: 冻结保证金，外源性，股票的保证金就是冻结资金，股票不需要

m_dFrozenCash: 冻结金额，内外源冻结保证金和手续费四个的和

m_dFrozenCommission: 冻结手续费，外源性冻结资金源

m_dRisk: 风险度，风险度，冻结资金 / 可用资金，股票不需要

m_dNav: 单位净值

m_dPreBalance: 期初权益，股票不需要，也叫静态权益

m_dBalance: 总资产，动态权益，即市值

m_dAvailable: 可用金额

m_dCommission: 手续费，已经用掉的手续费

m_dPositionProfit: 持仓盈亏

m_dCloseProfit: 平仓盈亏，股票不需要

m_dCashIn: 出入金净值

m_dCurrMargin: 当前使用的保证金，股票不需要

m_dInitBalance: 初始权益

m_strStatus: 状态

m_dInitCloseMoney: 期初平仓盈亏，初始平仓盈亏

m_dInstrumentValue: 总市值，合约价值，合约价值

m_dDeposit: 入金

m_dWithdraw: 出金

m_dPreCredit: 上次信用额度，股票不需要
m_dPreMortgage: 上次质押，股票不需要
m_dMortgage: 质押，股票不需要
m_dCredit: 信用额度，股票不需要
m_dAssetBalance: 证券初始资金，股票不需要
m_strOpenDate: 起始日期，股票不需要
m_dFetchBalance: 可取金额
m_strTradingDate: 交易日
m_dStockValue: 股票总市值，期货没有
m_dLoanValue: 债券总市值，期货没有
m_dFundValue: 基金总市值，包括ETF和封闭式基金，期货没有
m_dRepurchaseValue: 回购总市值，所有回购，期货没有
m_dLongValue: 多单总市值，现货没有
m_dShortValue: 单总市值，现货没有
m_dNetValue: 净持仓总市值，净持仓市值 = 多 - 空
m_dAssureAsset: 净资产
m_dTotalDebit: 总负债
m_dEntrustAsset: 可信资产，用于校对
m_dInstrumentValueRMB: 总市值（人民币），沪港通
m_dSubscribeFee: 申购费，申购费
m_dGoldValue: 库存市值，黄金现货库存市值
m_dGoldFrozen: 现货冻结，黄金现货冻结
m_dMargin: 占用保证金，维持保证金
m_strMoneyType: 币种
m_dPurchasingPower: 购买力，盈透购买力
m_dRawMargin : 原始保证金
m_dBuyWaitMoney: 买入待交收金额（元），买入待交收
m_dSellWaitMoney: 卖出待交收金额（元），卖出待交收
m_dReceiveInterestTotal: 本期间应计利息
m_dRoyalty: 权利金收支，期货期权用
m_dFrozenRoyalty: 冻结权利金，期货期权用
m_dRealUsedMargin: 实时占用保证金，用于股票期权
m_dRealRiskDegree: 实时风险度

5.3.2. order 委托对象

m_strExchangeID: 证券市场
m_strExchangeName: 交易市场
m_strProductID: 品种代码
m_strProductName: 品种名称
m_strInstrumentID: 证券代码
m_strInstrumentName: 证券名称, 合约名称
m_nTaskId: 任务号
m_strOrderRef: 内部委托号, 下单引用等于股票的内部委托号
m_nOrderPriceType: EBrokerPriceType 类型, 例如市价单、限价单
m_nDirection: EEntrustBS 类型, 操作, 多空, 期货多空, 股票买卖永远是 48, 其他的 dir 同理
m_nOffsetFlag: EOffset_Flag_Type 类型, 操作, 期货开平, 股票买卖其实就是开平
m_nHedgeFlag: EHedge_Flag_Type 类型, 投保
m_dLimitPrice: 委托价格, 限价单的限价, 就是报价
m_nVolumeTotalOriginal: 委托量, 最初委托量
m_nOrderSubmitStatus: EEntrustSubmitStatus 类型, 报单状态, 提交状态, 股票中不需要报单状态
m_strOrderSysID: 合同编号, 委托号
m_nOrderStatus: EEntrustStatus, 委托状态
m_nVolumeTraded: 成交数量, 已成交量
m_nVolumeTotal: 委托剩余量, 当前总委托量, 股票的含义是总委托量减去成交量
m_nErrorID: 状态信息
m_dFrozenMargin: 冻结金额, 冻结保证金
m_dFrozenCommission: 冻结手续费
m_strInsertDate: 委托日期, 报单日期
m_strInsertTime: 委托时间
m_dTradedPrice: 成交均价 (股票)
m_dCancelAmount: 已撤数量
m_strOptName: 买卖标记, 展示委托属性的中文
m_dTradeAmount: 成交金额, 成交额, 期货 = 均价 * 量 * 合约乘数
m_eEntrustType: EEntrustTypes, 委托类别
m_strCancelInfo: 废单原因
m_strUnderCode: 标的证券代码
m_eCoveredFlag: ECoveredFlag, 备兑标记 '0' - 非备兑, '1' - 备兑

m_strCompactNo: 合约编号

m_strRemark: 投资备注

5.3.3. deal 成交对象

m_strExchangeID: 证券市场, 交易所代码

m_strExchangeName: 交易市场, 交易所名称

m_strProductID: 品种代码

m_strProductName: 品种名称

m_strInstrumentID: 证券代码

m_strInstrumentName: 证券名称

m_strTradeID: 成交编号

m_nTaskId: 任务号

m_strOrderRef: 下单引用, 等于股票的内部委托号

m_strOrderSysID: 合同编号, 报单编号, 委托号

m_nDirection: EEntrustBS, 买卖方向

m_nOffsetFlag: EOffset_Flag_Type, 开平, 股票的买卖

m_nHedgeFlag: EHedge_Flag_Type, 投保, 股票不需要

m_dPrice: 成交均价

m_nVolume: 成交量, 期货单位手, 股票做到股

m_strTradeDate: 成交日期

m_strTradeTime: 成交时间

m_dComssion: 手续费

m_dTradeAmount: 成交额, 期货 = 均价 * 量 * 合约乘数

m_nOrderPriceType: EBrokerPriceType 类型, 例如市价单、限价单

m_strOptName: 买卖标记, 展示委托属性的中文

m_eEntrustType: EEntrustTypes 类型, 委托类别

m_eFutureTradeType: EFutureTradeType 类型, 成交类型

m_nRealOffsetFlag: EOffset_Flag_Type 类型, 实际开平, 主要是区分平今和平昨

m_eCoveredFlag: ECoveredFlag类型, 备兑标记 '0' - 非备兑, '1' - 备兑

m_nCloseTodayVolume: 平今量, 不显示

m_dOrderPriceRMB: 委托价格 (人民币), 目前用于港股通

m_dPriceRMB: 成交价格 (人民币), 目前用于港股通

m_dTradeAmountRMB: 成交金额 (人民币), 目前用于港股通

m_dReferenceRate: 汇率, 目前用于港股通

m_strXTTTrade: 是否是迅投交易
m_strCompactNo: 合约编号
m_dCloseProfit: 平仓盈亏，目前用于外盘
m_strRemark: 投资备注

5.3.4. position 持仓对象

m_strExchangeID: 证券市场
m_strExchangeName: 市场名称
m_strProductID: 品种代码
m_strProductName: 品种名称
m_strInstrumentID: 证券代码
m_strInstrumentName: 证券名称
m_nHedgeFlag: EHedge_Flag_Type 类型，投保，股票不需要
m_nDirection: EEntrustBS 类型，买卖；股票不需要
m_strOpenDate: 成交日期
m_strTradeID: 成交号，最初开仓位的成交
m_nVolume: 当前拥股，持仓量
m_dOpenPrice: 持仓成本
m_strTradingDay: 在实盘运行中是当前交易日，在回测中是股票最后交易过的日期
m_dMargin: 使用的保证金，历史的直接用 ctp 的，新的自己用成本价 * 存量 * 系数算，股票不需要
m_dOpenCost: 开仓成本，等于股票的成本价 * 第一次建仓的量，后续减持不影响，不算手续费，股票不需要
m_dSettlementPrice: 最新价，结算价，对于股票的当前价
m_nCloseVolume: 平仓量，等于股票已经卖掉的 股票不需要
m_dCloseAmount: 平仓额，等于股票每次卖出的量 * 卖出价 * 合约乘数（股票为 1）的累加 股票不需要
m_dFloatProfit: 浮动盈亏，当前量 * (当前价 - 开仓价) * 合约乘数（股票为 1）
m_dCloseProfit: 平仓盈亏，平仓额 - 开仓价 * 平仓量 * 合约乘数（股票为 1） 股票不需要
m_dMarketValue: 市值，合约价值
m_dPositionCost: 持仓成本，股票不需要
m_dPositionProfit: 持仓盈亏，股票不需要
m_dLastSettlementPrice: 最新结算价，股票不需要
m_dInstrumentValue: 合约价值，股票不需要
m_bIsToday: 是否今仓

m_strStockHolder: 股东账号
m_nFrozenVolume: 冻结数量, 冻结持仓, 期货不用这个字段, 冻结数量
m_nCanUseVolume: 可用余额, 可用持仓, 期货不用这个字段, 股票的可用数量
m_nOnRoadVolume: 在途股份, 在途持仓, 期货不用这个字段, 股票的在途数量
m_nYesterdayVolume: 昨夜拥股, 期货不用这个字段, 股票的股份余额
m_dLastPrice: 最高价, 结算价, 对于股票的当前价
m_dProfitRate: 盈亏比例, 持仓盈亏比例
m_eFutureTradeType: EFutureTradeType, 成交类型
m_strExpireDate: 到期日, 逆回购用
m_strComTradeID: 组合成交号, 套利成交 ID
m_nLegId: 组合序号, 组合 ID
m_dTotalCost: 累计成本, 自定义累计成本, 股票信用用到
m_dSingleCost: 单股成本, 自定义单股成本, 股票信用用到
m_nCoveredVolume: 备兑数量, 用于个股期权
m_eSideFlag: ESideFlag 持仓类型, 用于个股期权, 标记 '0' - 权利, '1' - 义务, '2' - '备兑'
m_dReferenceRate: 汇率, 目前用于港股通
m_dStructFundVol: 分级基金可用 (可分拆或可合并)
m_dRedemptionVolume: 分级基金可赎回量
m_nPREnableVolume: 申赎可用量 (记录当日申购赎回的股票或基金数量)
m_dRealUsedMargin: 实时占用保证金, 用于期权
m_dRoyalty: 权利金

5.3.5. CCreditAccountDetail信用账号对象(非查柜台)

m_dMaxMarginRate: 保证金比率, 股票的保证金率等于 1, 股票不需要
m_dFrozenMargin: 冻结保证金, 外源性, 股票的保证金就是冻结资金, 股票不需要
m_dFrozenCash: 冻结金额, 内外源冻结保证金和手续费四个的和
m_dFrozenCommission: 冻结手续费, 外源性冻结资金源
m_dRisk: 风险度, 冻结资金/可用资金, 股票不需要
m_dNav: 单位净值
m_dPreBalance: 期初权益, 股票不需要, 也叫静态权益
m_dBalance: 总资产, 动态权益, 即市值
m_dAvailable: 可用金额
m_dCommission: 手续费, 已经用掉的手续费
m_dPositionProfit: 持仓盈亏

m_dCloseProfit: 平仓盈亏, 股票不需要
m_dCashIn: 出入金净值
m_dCurrMargin: 当前使用的保证金, 股票不需要
m_dInitBalance: 初始权益
m_strStatus: 状态
m_dInitCloseMoney: 期初平仓盈亏, 初始平仓盈亏
m_dInstrumentValue: 总市值, 合约价值, 合约价值
m_dDeposit: 入金
m_dWithdraw: 出金
m_dPreCredit: 上次信用额度, 股票不需要
m_dPreMortgage: 上次质押, 股票不需要
m_dMortgage: 质押, 股票不需要
m_dCredit: 信用额度, 股票不需要
m_dAssetBalance: 证券初始资金, 股票不需要
m_strOpenDate: 起始日期股票不需要
m_dFetchBalance: 可取金额
m_strTradingDate: 交易日
m_dStockValue: 股票总市值, 期货没有
m_dLoanValue: 债券总市值, 期货没有
m_dFundValue: 基金总市值, 包括 ETF 和封闭式基金, 期货没有
m_dRepurchaseValue: 回购总市值, 所有回购, 期货没有
m_dLongValue: 多单总市值, 现货没有
m_dShortValue: 单总市值, 现货没有
m_dNetValue: 净持仓总市值, 净持仓市值 = 多 - 空
m_dAssureAsset: 净资产
m_dTotalDebt: 总负债
m_dEntrustAsset: 可信资产, 用于校对
m_dInstrumentValueRMB: 总市值 (人民币), 沪港通
m_dSubscribeFee: 申购费, 申购费
m_dGoldValue: 库存市值, 黄金现货库存市值
m_dGoldFrozen: 现货冻结, 黄金现货冻结
m_dMargin: 占用保证金, 维持保证金
m_strMoneyType: 币种
m_dPurchasingPower: 购买力, 盈透购买力

m_dRawMargin : 原始保证金
m_dBuyWaitMoney: 买入待交收金额 (元) , 买入待交收
m_dSellWaitMoney: 卖出待交收金额 (元) , 卖出待交收
m_dReceiveInterestTotal: 本期间应计利息
m_dRoyalty: 权利金收支, 期货期权用
m_dFrozenRoyalty: 冻结权利金, 期货期权用
m_dRealUsedMargin: 实时占用保证金, 用于股票期权
m_dRealRiskDegree: 实时风险度
m_dPerAssurescaleValue: 个人维持担保比例
m_dEnableBailBalance: 可用保证金
m_dUsedBailBalance: 已用保证金
m_dAssureEnbuyBalance: 可买担保品资金
m_dFinEnbuyBalance: 可买标的券资金
m_dSloEnrepaidBalance: 可还券资金
m_dFinEnrepaidBalance: 可还款资金
m_dFinMaxQuota: 融资授信额度
m_dFinEnableQuota: 融资可用额度
m_dFinUsedQuota: 融资已用额度
m_dFinUsedBail: 融资已用保证金
m_dFinCompactBalance: 融资合约金额
m_dFinCompactFare: 融资合约费用
m_dFinCompactInterest: 融资合约利息
m_dFinMarketValue: 融资市值
m_dFinIncome: 融资合约盈亏
m_dSloMaxQuota: 融券授信额度
m_dSloEnableQuota: 融券可用额度
m_dSloUsedQuota: 融券已用额度
m_dSloUsedBail: 融券已用保证金
m_dSloCompactBalance: 融券合约金额
m_dSloCompactFare: 融券合约费用
m_dSloCompactInterest: 融券合约利息
m_dSloMarketValue: 融券市值
m_dSloIncome: 融券合约盈亏
m_dOtherFare: 其它费用

m_dUnderlyMarketValue: 标的证券市值
m_dFinEnableBalance: 可融资金额
m_dDiffEnableBailBalance: 可用保证金调整值
m_dBuySecuRepayFrozenMargin: 买券还券冻结资金
m_dBuySecuRepayFrozenCommission: 买券还券冻结手续费
m_dSpecialEnableBalance: 专项可融金额
m_dEncumberedAssets: 担保资产
m_dSloSellBalance: 融券卖出资金
m_dDiffAssureEnbuyBalance: 可买担保品资金调整值
m_dDiffFinEnbuyBalance: 可买标的券资金调整值
m_dDiffFinEnrepaidBalance: 可还款资金调整值
m_dOtherRealCompactBalance: 其他负债合约金额
m_dOtherFinCompactInterest: 其他负债合约利息金额
m_dUsedSloSellBalance: 已用融券卖出资金
m_dFetchAssetBalance: 可提出资产总额
m_dTotalEnableQuota: 可用总信用额度
m_dTotalUsedQuota: 已用总信用额度;
m_dDebtProfit: 负债总浮盈
m_dDebtLoss: 负债总浮亏
m_nContractEndDate: 合同到期日期
m_dFinDebt: 融资负债
m_dFinProfitAmortized: 融资浮盈折算
m_dSloProfit: 融券浮盈
m_dSloProfitAmortized: 融券浮盈折算
m_dFinLoss: 融资浮亏
m_dSloLoss: 融券浮亏

5.3.6. CreditSloEnableAmount 可融券明细对象

m_nPlatformID: 平台号
m_strBrokerID: 经纪公司编号
m_strBrokerName: 证券公司, 期货公司, 经纪公司, 证券公司
m_strAccountID: 资金账号, 账号, 账号, 资金账号
m_strExchangeID: 交易所
m_strInstrumentID: 证券代码

m_strInstrumentName: 股票名称
m_dSloRatio: 融券保证金比例
m_eSloStatus: EXTSubjectsStatus, 融券状态
m_nEnableAmount: 融券可融数量
m_eQuerySloType: EXTSIoTypeQueryMode, 查询类型
m_strExpireDate: 到期日期

5.3.7. StkCompacts负债合约对象

m_strExchangeID: 交易所
m_strInstrumentID: 证券代码
m_strExchangeName: 交易所名称
m_strInstrumentName: 股票名称
m_nOpenDate: 合约开仓日期
m_strCompactId: 合约编号
m_dCrdtRatio: 融资融券保证金比例
m_strEntrustNo: 委托编号
m_dEntrustPrice: 委托价格
m_nEntrustVol: 委托数量
m_nBusinessVol: 合约开仓数量
m_dBusinessBalance: 合约开仓金额
m_dBusinessFare: 合约开仓费用
m_eCompactType: EXTCompactType, 合约类型
m_eCompactStatus: EXTCompactStatus, 合约状态
m_dRealCompactBalance: 未还合约金额
m_nRealCompactVol: 未还合约数量
m_dRealCompactFare: 未还合约费用
m_dRealCompactInterest: 未还合约利息
m_dRepaidInterest: 已还利息
m_nRepaidVol: 已还数量
m_dRepaidBalance: 已还金额
m_dCompactInterest: 合约总利息
m_dUsedBailBalance: 占用保证金
m_dYearRate: 合约年利率
m_nRetEndDate: 归还截止日

m_strDateClear: 了结日期
m_strPositionStr: 定位串
m_dPrice: 最新价
m_nOpenTime: 合约开仓时间
m_nCancelVol: 合约撤单数量
m_eCashgroupProp: EXTCompactBrushSource, 头寸来源
m_dUnRepayBalance: 负债金额
m_nRepayPriority: 偿还优先级
m_dRealDefaultInterest: 未还罚息
m_dOtherRealCompactBalance: 其他负债合约金额
m_dOtherRealCompactInterest: 其他负债合约利息金额

5.3.8. StkSubjects担保标的对象

m_nPlatformID: 平台号//目前主要用于区别不同的行情，根据此来选择对应行情
m_strBrokerID: 经纪公司编号
m_strBrokerName: 证券公司, 期货公司, 经纪公司, 证券公司
m_strExchangeID: 交易所
m_strInstrumentID: 证券代码
m_strInstrumentName: 股票名称
m_dSloRatio: 融券保证金比例
m_eSloStatus: EXTSubjectsStatus, 融券状态
m_nEnableAmount: 融券可融数量
m_dFinRatio: 融资保证金比例
m_eFinStatus: EXTSubjectsStatus, 融资状态
m_dAssureRatio: 担保品折算比例
m_eAssureStatus: EXTSubjectsStatus, 是否可做担保
m_dFinRate: 融资日利率
m_dSloRate: 融券日利率
m_dFinPenaltyRate: 融资日罚息利率
m_dSloPenaltyRate: 融券日罚息利率
m_strAccountID: 资金账号, 账号, 账号, 资金账号
m_eAssureUseSloCashStatus: EXTSpecialAssure, 是否可以用融券资金买入

5.3.9 PassorderArguments 下单函数参数对象

opType: passorder的opType参数
orderType: passorder的orderType参数
accountId:资金账号
orderCode:交易代码
prType: passorder的prType, 价格类型
modelPrice:下单价格
modelVolume: 下单量 (手数或股数)
strategyName:策略名 _ &&& _ 投资备注

5.3.10 CTaskDetail 任务对象

m_nTaskId: 任务号
m_eStatus: 任务状态 ETaskStatus类型,见ETaskStatus说明
m_strMsg: 任务状态消息
m_startTime: 任务开始时间, 时间戳类型
m_endTime: 任务结束时间, 时间戳类型
m_cancelTime: 任务取消时间
m_nBusinessNum: 已成交量
m_bServerRun: 是否服务器运行
m_nGroupId: 组合Id
m_stockCode: 下单代码(不针对组合下单)
m_strAccountID: 下单用户(单用户下单)
m_eOperationType: 下单操作: 开平、多空.....EOperationType类型, 见EOperationType说明
m_eOrderType: 算法交易、普通交易 EOrderType类型, 见EOrderType说明
m_ePriceType: 报价方式: 对手、最新..... EPriceType类型见EPriceType说明
m_dFixPrice: 委托价
m_nNum: 委托量

5.3.11 CCreditDetail 两融资金信息(查柜台)

m_dPerAssurescaleValue:维持担保比例
m_dBalance: 总资产
m_dTotalDebt: 总负债
m_dAssureAsset: 净资产
m_dMarketValue: 总市值
m_dEnableBailBalance:可用保证金

```
m_dAvailable: 可用资金  
m_dFinDebt: 融资负债  
m_dFinDealAvl: 融资本金  
m_dFinFee: 融资息费  
m_dSloDebt: 融券负债  
m_dSloMarketValue: 融券市值  
m_dSloFee: 融券息费  
m_dOtherFare: 其它费用  
m_dFinMaxQuota: 融资授信额度  
m_dFinEnableQuota: 融资可用额度  
m_dFinUsedQuota: 融资冻结额度  
m_dSloMaxQuota: 融券授信额度  
m_dSloEnableQuota: 融券可用额度  
m_dSloUsedQuota: 融券冻结额度  
m_dSloSellBalance: 融券卖出资金  
m_dUsedSloSellBalance: 已用融券卖出资金  
m_dSurplusSloSellBalance: 剩余融券卖出资金  
m_dStockValue: 股票市值  
m_dFundValue: 基金市值  
error: 错误信息
```

5.3.12 CLockPosition期权标的持仓

```
1 m_strAccountID 账号名  
2  
3 m_strExchangeID 交易所  
4  
5 m_strExchangeName 交易所名  
6  
7 m_strInstrumentID 标的代码  
8  
9 m_strInstrumentName 标的名称  
10  
11 m_totalvol 总持仓量  
12  
13 m_lockvol 可用锁定量  
14  
15 m_unlockvol 未锁定量  
16  
17 m_coveredvol 备兑量  
18  
19 m_nOnRoadcoveredvol 在途备兑量
```

5.3.13 CStkOptCombPositionDetail 期权组合持仓

```
1 m_strAccountID 账号名
2
3 m_strExchangeID 交易所
4
5 m_strExchangeName 交易所名
6
7 m_strContractAccount 合约账号
8
9 m_strCombID 组合编号
10
11 m_strCombCode 组合策略编码
12
13 m_strCombCodeName 组合策略名称
14
15 m_nVolume 持仓量
16
17 m_nFrozenVolume 冻结数量
18
19 m_nCanUseVolume 可用数量
20
21 m_strFirstCode 合约一
22
23 m_eFirstCodeType 合约一类型 认购:48,认沽:49
24
25 m_strFirstCodeName 合约一名称
26
27 m_eFirstCodePosType 合约一持仓类型 权利:48,义务:49,备兑:50
28
29 m_nFirstCodeAmt 合约一数量
30
31 m_strSecondCode 合约二
32
33 m_eSecondCodeType 合约二类型 认购:48,认沽:49
34
35 m_strSecondCodeName 合约二名称
36
37 m_eSecondCodePosType 合约二持仓类型 权利:48,义务:49,备兑:50
38
39 m_nSecondCodeAmt 合约二数量
40
41 m_dCombBailBalance 占用保证金
```

5.4. 附录4 对象中属性一些需要的状态字段释义

4-1.enum_EEntrustBS //买卖方向

```
ENTRUST_BUY: 48 //买入, 多
ENTRUST_SELL: 49 //卖出, 空
ENTRUST_PLEDGE_IN: 81 //质押入库
ENTRUST_PLEDGE_OUT: 66 //质押出库
```

4-2.EEntrustSubmitStatus//报单状态

```
48 //已经提交  
49 //撤单已经提交  
50 //修改已经提交  
51 //已经接受  
52 //报单已经被拒绝  
53 //撤单已经被拒绝  
54 //改单已经被拒绝
```

4-3.enum_EEntrustTypes //委托类型

```
ENTRUST_BUY_SELL: 48 //买卖  
ENTRUST_QUERY: 49 //查询  
ENTRUST_CANCEL: 50 //撤单  
ENTRUST_APPEND: 51 //补单  
ENTRUST_CONFIRM: 52 //确认  
ENTRUST_BIG: 53 //大宗  
ENTRUST_FIN: 54 //融资委托  
ENTRUST_SLO: 55 //融券委托  
ENTRUST_CLOSE: 56 //信用平仓  
ENTRUST_CREDIT_NORMAL: 57 //信用普通委托  
ENTRUST_CANCEL_OPEN: 58 //撤单补单  
ENTRUST_TYPE_OPTION_EXERCISE: 59 //行权  
ENTRUST_TYPE_OPTION_SECU_LOCK: 60 //锁定  
ENTRUST_TYPE_OPTION_SECU_UNLOCK: 61 //解锁  
ENTRUST_QUOTATION_REPURCHASE: 62; //报价回购  
ENTRUST_TYPE_OPTION_ABANDON : 63; //放弃行权  
ENTRUST_AGREEMENT_REPURCHASE: 64; //协议回购  
ENTRUST_TYPE_OPTION_COMB_EXERCISE: 65; //组合行权  
ENTRUST_TYPE_OPTION_BUILD_COMB_STRATEGY : 66; //构建组合策略持仓  
ENTRUST_TYPE_OPTION_RELEASE_COMB_STRATEGY: 67; //解除组合策略持仓  
ENTRUST_TYPE_LMT_LOAN: 68; //转融通出借  
ENTRUST_TYPE_LMT_LOAN_DEFER: 69; //转融通出借展期  
ENTRUST_TYPE_LMT_LOAN_FINISH_AHEAD: 70; //转融通出借提前了结  
ENTRUST_CROSS_MARKET_IN 71; //跨市场场内
```

```
ENTRUST_CROSS_MARKET_OUT 72; //跨市场场外
```

4-4.enum_EEntrustStatus //委托状态

```
ENTRUST_STATUS_WAIT_END: 0 //委托状态已经在 ENTRUST_STATUS_CANCELED 或以上,  
但是成交数额还不够, 等成交回报来
```

```
ENTRUST_STATUS_UNREPORTED: 48 //未报
```

```
ENTRUST_STATUS_WAIT_REPORTING: 49 //待报
```

```
ENTRUST_STATUS_REPORTED: 50 //已报
```

```
ENTRUST_STATUS_REPORTED_CANCEL: 51 //已报待撤
```

```
ENTRUST_STATUS_PARTSUCC_CANCEL: 52 //部成待撤
```

```
ENTRUST_STATUS_PART_CANCEL: 53 //部撤
```

```
ENTRUST_STATUS_CANCELED: 54 //已撤
```

```
ENTRUST_STATUS_PART_SUCC: 55 //部成
```

```
ENTRUST_STATUS_SUCCEEDED: 56 //已成
```

```
ENTRUST_STATUS_JUNK: 57 //废单
```

```
ENTRUST_STATUS_DETERMINED: 86 //已确认
```

```
ENTRUST_STATUS_UNKNOWN: 255 //未知
```

4-5.enum_EHedge_Flag_Type

```
HEDGE_FLAG_SPECULATION: 49 //投机
```

```
HEDGE_FLAG_ARBITRAGE: 50 //套利
```

```
HEDGE_FLAG_HEDGE: 51 //套保
```

4-6.enum_EFutureTradeType // 成交类型

```
FUTRUE_TRADE_TYPE_COMMON: 48 //普通成交
```

```
FUTURE_TRADE_TYPE_OPTIONSEXECUTION: 49 //期权成交, optionsExecution
```

```
FUTURE_TRADE_TYPE_OTC: 50 //OTC 成交
```

```
FUTURE_TRADE_TYPE_EFPDIRVED: 51 //期转现衍生成交
```

```
FUTURE_TRADE_TYPE_COMBINATION_DERIVED: 52 //组合衍生成交
```

4-7.enum_EBrokerPriceType // 价格类型

```
BROKER_PRICE_ANY: 49 //市价
```

```
BROKER_PRICE_LIMIT: 50 //限价
```

```
BROKER_PRICE_BEST: 51 //最优价
```

```
BROKER_PRICE_PROP_ALLOTMENT: 52 //配股
```

```
BROKER_PRICE_PROP_REFER: 53 //转托
```

BROKER_PRICE_PROP_SUBSCRIBE: 54 //申购
BROKER_PRICE_PROP_BUYBACK: 55 //回购
BROKER_PRICE_PROP_PLACING: 56 //配售
BROKER_PRICE_PROP_DECIDE: 57 //指定
BROKER_PRICE_PROP_EQUITY: 58 //转股
BROKER_PRICE_PROP_SELLBACK: 59 //回售
BROKER_PRICE_PROP_DIVIDEND: 60 //股息
BROKER_PRICE_PROP_SHENZHEN_PLACING: 68 //深圳配售确认
BROKER_PRICE_PROP_CANCEL_PLACING: 69 //配售放弃
BROKER_PRICE_PROP_WDZY: 70 //无冻质押
BROKER_PRICE_PROP_DJZY: 71 //冻结质押
BROKER_PRICE_PROP_WDJY: 72 //无冻解押
BROKER_PRICE_PROP_JDJY: 73 //解冻解押
BROKER_PRICE_PROP ETF: 81 //ETF申购
BROKER_PRICE_PROP_VOTE: 75 //投票
BROKER_PRICE_PROP_YYSGYS: 92 //要约收购预售
BROKER_PRICE_PROP_YSYYJC: 77 //预售要约解除
BROKER_PRICE_PROP_FUND_DEVIDEND //基金设红
BROKER_PRICE_PROP_FUND_ENTRUST: 79 //基金申赎
BROKER_PRICE_PROP_CROSS_MARKET: 80 //跨市转托
BROKER_PRICE_PROP_EXERCIS: 83 //权证行权
BROKER_PRICE_PROP_PEER_PRICE_FIRST: 84 //对手方最优价格
BROKER_PRICE_PROP_L5_FIRST_LIMITPX: 85 //最优五档即时成交剩余转限价
BROKER_PRICE_PROP_MIME_PRICE_FIRST: 86 //本方最优价格
BROKER_PRICE_PROP_INSTBUSI_RESTCANCEL: 87 //即时成交剩余撤销
BROKER_PRICE_PROP_L5_FIRST_CANCEL: 88 //最优五档即时成交剩余撤销
BROKER_PRICE_PROP_FULL_REAL_CANCEL: 89 //全额成交并撤单
BROKER_PRICE_PROP_DIRECT_SECU_REPAY: 101 //直接还券
BROKER_PRICE_PROP_FUND_CHAIHE: 90 //基金拆合
BROKER_PRICE_PROP_DEBT_CONVERSION: 91 //债转股
BROKER_PRICE_BID_LIMIT: 92 //港股通竞价限价
BROKER_PRICE_ENHANCED_LIMIT: 93 //港股通增强限价
BROKER_PRICE RETAIL_LIMIT: 94 //港股通零股限价
BROKER_PRICE_PROP_INCREASE_SHARE: 'j' //增发

```
BROKER_PRICE_PROP_COLLATERAL_TRANSFER: 107 //担保品划转  
BROKER_PRICE_PROP_NEEQ_PRICING: 'w' //定价 (全国股转 - 挂牌公司交易 - 协议转让)  
BROKER_PRICE_PROP_NEEQ_MATCH_CONFIRM: 'x' //成交确认 (全国股转 - 挂牌公司交易 - 协议转让)  
BROKER_PRICE_PROP_NEEQ_MUTUAL_MATCH_CONFIRM: 'y' //互报成交确认 (全国股转 - 挂牌公司交易 - 协议转让)  
BROKER_PRICE_PROP_NEEQ_LIMIT: 'z' //限价 (用于挂牌公司交易 - 做市转让 - 限价买卖和两网及退市交易-限价买卖)
```

4-8.enum_EOffset_Flag_Type //操作类型

```
EOFF_THOST_FTDC_OF_INVALID: -1  
EOFF_THOST_FTDC_OF_Open: 48 //买入, 开仓  
EOFF_THOST_FTDC_OF_Close: 49 //卖出, 平仓  
EOFF_THOST_FTDC_OF_ForceClose: 50 //强平  
EOFF_THOST_FTDC_OF_CloseToday: 51 //平今  
EOFF_THOST_FTDC_OF_CloseYesterday: 52 //平昨  
EOFF_THOST_FTDC_OF_ForceOff: 53 //强减  
EOFF_THOST_FTDC_OF_LocalForceClose: 54 //本地强平  
EOFF_THOST_FTDC_OF_PLEDGE_IN: 81 //质押入库  
EOFF_THOST_FTDC_OF_PLEDGE_OUT: 66 //质押出库  
EOFF_THOST_FTDC_OF_ALLOTMENT: 67 //股票配股
```

4-9.enum EXTSubjectsStatus //融资融券状态

```
SUBJECTS_STATUS_NORMAL: 48 //正常  
SUBJECTS_STATUS_PAUSE: 49 //暂停  
SUBJECTS_STATUS_NOT: 50 //作废
```

4-10.enum EXTSIoTypeQueryMode //查询类型

```
XT_SLOTYPE_QUERYMODE_NOMARL: 48 //普通  
XT_SLOTYPE_QUERYMODE_SPECIAL: 49 //专项
```

4-11.enum EXTCompactType //合约类型

```
COMPACT_TYPE_ALL: 32 //不限制  
COMPACT_TYPE_FIN: 48 //融资  
COMPACT_TYPE_SLO: 49 //融券
```

4-12 enum EXTCompactStatus //合约状态

```
COMPACT_STATUS_ALL: 32 //不限制  
COMPACT_STATUS_UNDONE: 48 //未归还  
COMPACT_STATUS_PART_DONE: 49 //部分归还  
COMPACT_STATUS_DONE: 50 //已归还  
COMPACT_STATUS_DONE_BY_SELF: 51 //自行了结  
COMPACT_STATUS_DONE_BY_HAND: 52 //手工了结  
COMPACT_STATUS_NOT_DEBT: 53 //未形成负债  
COMPACT_STATUS_EXPIRY: 54 //合约已过期
```

4-13.enum EXTCompactBrushSource //头寸来源

```
XT_COMPACT_BRUSH_SOURCE_ALL: 32 //不限制  
XT_COMPACT_BRUSH_SOURCE_NORMAL: 48 //普通头寸  
XT_COMPACT_BRUSH_SOURCE_SPECIAL: 49 //专项头寸
```

4-14.enum _EXTSpecialAssure //是否可以用融券资金买入

```
ASSURE_USE_SLO_CASH_DISABLE: 48 //担保品买入不允许使用融券资金  
ASSURE_USE_SLO_CASH_ENABLE: 49 //担保品买入允许使用融券资金
```

4-15 enum_EOperationType : // 下单操作类型 主要交易类型

```
OPT_OPEN_LONG|开多: 0; // 期货操作的第一个, 请保持此OPT在期货操作的第一个, 否则客户端判断操作是期货还是股票的代码会有问题  
OPT_CLOSE_LONG_HISTORY:1 //平昨多 黄金用平多表示  
OPT_CLOSE_LONG_TODAY: 2 //平今多  
OPT_OPEN_SHORT: 3 //开空  
OPT_CLOSE_SHORT_HISTORY: 4 //平昨空,平昨空,平昨空,平昨空,平昨空,平昨空,平昨空,平昨空,平空; // 黄金用平空表示  
OPT_CLOSE_SHORT_TODAY: 5 //平今空;  
OPT_CLOSE_LONG_TODAY_FIRST: 6 //平多优先平今;  
OPT_CLOSE_LONG_HISTORY_FIRST: 7 //平多优先平昨;  
OPT_CLOSE_SHORT_TODAY_FIRST: 8 //平空优先平今;  
OPT_CLOSE_SHORT_HISTORY_FIRST: 9 //平空优先平昨;  
OPT_CLOSE_LONG_TODAY_HISTORY_THEN_OPEN_SHORT: 10 //卖出优先平今;  
OPT_CLOSE_LONG_HISTORY_TODAY_THEN_OPEN_SHORT: 11 //卖出优先平昨;  
OPT_CLOSE_SHORT_TODAY_HISTORY_THEN_OPEN_LONG: 12 //买入优先平今;
```

```
OPT_CLOSE_SHORT_HISTORY_TODAY_THEN_OPEN_LONG: 13 //买入优先平昨;  
OPT_CLOSE_LONG: 14 //平多  
OPT_CLOSE_SHORT: 15 //平空  
OPT_OPEN: 16 //开仓  
OPT_CLOSE: 17 // 平仓  
OPT_BUY: 18 //买入  
OPT_SELL: 19 //卖出  
OPT_FIN_BUY: 20 //融资买入  
OPT_SLO_SELL: 21 //融券卖出  
OPT_BUY_SECU_REPAY: 22 //买券还券  
OPT_DIRECT_SECU_REPAY: 23 //直接还券  
OPT_SELL_CASH_REPAY: 24 //卖券还款  
OPT_DIRECT_CASH_REPAY: 25 //直接还款
```

4-16 enum_EOrderType : // 算法交易、普通交易类型

```
OTP_ORDINARY; 0 //常规  
OTP_ALGORITHM; 1 //算法交易  
OTP_RANDVOLUME; 2 // 随机量交易  
OTP_ALGORITHM3 3 // 算法交易3  
OTP_ZXJT; 4 // 中信建投算法  
OTP_ZSGS; 5 // 隔时交易  
OTP_ORDINARY_BASKET_TRIGGER_SINGLE_ORDER; 6 //普通交易的触价单笔委托方式  
OTP_ALGORITHM_BASKET_TRIGGER_SINGLE_ORDER; 7 //算法交易的触价单笔委托方式  
OTP_ZXZQ; 8 // 中信证券算法  
OTP_GENUS; 9 //金纳算法  
OTP_JAZZ; 10 //爵士算法  
OTP_VWAP; 11 //智能VWAP  
OTP_TWAP; 12 //智能TWAP  
OTP_XTALGO; 13 //智能算法  
OTP_HUACHUANG; 14 //华创算法  
OTP_HUARUN; 15 //华润算法  
OTP_CUSTOM; 16 //回转算法  
OPT_EXTERN; 17 //主动算法  
OTP_GUANGFA; 18 //广发算法
```

4-17 enum_EPriceType : // 价格类型

```
PRTP_SALE5; 0          // 卖5
PRTP_SALE4; 1          // 卖4
PRTP_SALE3; 2          // 卖3
PRTP_SALE2; 3          // 卖2
PRTP_SALE1; 4          // 卖1
PRTP_LATEST; 5         // 最高价
PRTP_BUY1; 6           // 买1
PRTP_BUY2; 7           // 买2
PRTP_BUY3; 8           // 买3
PRTP_BUY4; 9           // 买4
PRTP_BUY5; 10          // 买5
PRTP_FIX; 11           // 指定价
PRTP_MARKET; 12         // 市价_涨跌停价
PRTP_HANG; 13           // 挂单价
PRTP_COMPETE; 14        // 对手价
PRTP_AUTO; 15           // 自动盘口
PRTP_CLOSE; 16           // 昨收价
PRTP_AVERAGE; 17         // 大宗加权平均价
PRTP_MARKET_BEST; 18      // 市价_最优价
PRTP_MARKET_CANCEL; 19      // 市价_即成剩撤
PRTP_MARKET_CANCEL_ALL; 20     // 市价_全额成交或撤
PRTP_MARKET_CANCEL_1; 21      // 市价_最优1档即成剩撤
PRTP_MARKET_CANCEL_5; 22      // 市价_最优5档即成剩撤
PRTP_MARKET_CONVERT_1; 23     // 市价_最优1档即成剩转
PRTP_MARKET_CONVERT_5; 24     // 市价_最优5档即成剩转
PRTP_STK_OPTION_ASK; 25       // 询价
PRTP_STK_OPTION_FIX_CANCEL_ALL; 26    // 限价即时全部成交否则撤单
PRTP_STK_OPTION_MARKET_CACEL_LEFT; 27   // 市价即时成交剩余撤单
PRTP_STK_OPTION_MARKET_CANCEL_ALL; 28      // 市价即时全部成交否则撤单
PRTP_STK_OPTION_MARKET_CONVERT_FIX; 29     // 市价剩余转限价
PRTP_SALE6; 30           // 卖6
PRTP_SALE7; 31           // 卖7
PRTP_SALE8; 32           // 卖8
PRTP_SALE9; 33           // 卖9
```

```
PRTP_SALE10; 34      // 卖10
PRTP_BUY6; 35        // 买6
PRTP_BUY7; 36        // 买7
PRTP_BUY8; 37        // 买8
PRTP_BUY9; 38        // 买9
PRTP_BUY10; 39       // 买10
PRTP_UPPER_LIMIT_PRICE; 40      //涨停价
PRTP_LOWER_LIMIT_PRICE; 41      //跌停价
PRTP_MARKET_SH_CONVERT_5_CANCEL; 42      //最优五档即时成交剩余撤销
PRTP_MARKET_SH_CONVERT_5_LIMIT; 43      //最优五档即时成交剩转限价
PRTP_MARKET_PEER_PRICE_FIRST; 44      //对手方最优价格委托
PRTP_MARKET_MINE_PRICE_FIRST; 45      //本方最优价格委托
PRTP_MARKET_SZ_INSTBUSI_RESTCANCEL; 46      //即时成交剩余撤销委托
PRTP_MARKET_SZ_CONVERT_5_CANCEL; 47      //最优五档即时成交剩余撤销委托
PRTP_MARKET_SZ_FULL_REAL_CANCEL; 48      //全额成交或撤销委托
PRTP_AFTER_FIX_PRICE; 49      // 盘后定价申报
```

4-18 enum_ETaskStatus : // 任务状态

```
TASK_STATUS_UNKNOWN : 0;//未知
TASK_STATUS_WAITING;//等待
TASK_STATUS_COMMITTING;
TASK_STATUS_RUNNING;//执行中
TASK_STATUS_PAUSE;//暂停
TASK_STATUS_CANCELING_DEPRECATED;//撤销中
TASK_STATUS_EXCEPTION_CANCELING_DEPRECATED;//异常撤销中
TASK_STATUS_COMPLETED;//完成
TASK_STATUS_CANCELED;//已撤
TASK_STATUS_REJECTED;//打回
TASK_STATUS_EXCEPTION_CANCELED;//异常终止
TASK_STATUS_DROPPED;//放弃，目前用于组合交易中，放弃补单
TASK_STATUS_FORCE_CANCELED_DEPRECATED;//强制终止
```

5.5. 附录5 行情数据字段列表

tick - 分笔数据

- `time` - int 时间戳
- `stime` - str 时间戳
- `lastPrice` - float 最高价
- `open` - float 开盘价
- `high` - float 最高价
- `low` - float 最低价
- `lastClose` - float 前收盘价
- `amount` - float 成交总额
- `volume` - int 成交总量
- `pvolume` - int 原始成交总量(未经过股手转换的成交总量)
- `stockStatus` - int 证券状态
- `openInt` - int 持仓量
- `transactionNum` - float 成交笔数(期货没有, 单独计算)
- `lastSettlementPrice` - float 前结算(股票为0)
- `settlementPrice` - float 今结算(股票为0)
- `askPrice` - list[float] 多档委卖价
- `askvol` - list[int] 多档委卖量
- `bidPrice` - list[float] 多档委买价
- `bidvol` - list[int] 多档委买量

1m / 5m / 1d - K线数据

- `time` - int 时间戳
- `stime` - str 时间戳
- `open` - float 开盘价
- `high` - float 最高价
- `low` - float 最低价
- `close` - float 收盘价
- `volume` - int 成交量
- `amount` - float 成交额
- `settlementPrice` - float 今结算
- `openInterest` - int 持仓量

Level2 行情快照

- `time` - int 时间戳
- `stime` - str 时间戳
- `lastPrice` - float 最高价
- `open` - float 开盘价
- `high` - float 最高价
- `low` - float 最低价
- `amount` - float 成交额
- `volume` - int 成交总量
- `pvolume` - int 原始成交总量(未经过股手转换的成交总量)
- `stockStatus` - int 证券状态
- `openInt` - int 持仓量
- `transactionNum` - int 成交笔数(期货没有, 单独计算)
- `lastClose` - float 前收盘价
- `lastSettlementPrice` - float 前结算(股票为0)

- `settlementPrice` - float 今结算(股票为0)
- `askPrice` - list[float] 多档委卖价
- `askVol` - list[int] 多档委卖量
- `bidPrice` - list[float] 多档委买价
- `bidVol` - list[int] 多档委买量

Level2 行情快照补充

- `time` - int 时间戳
- `stime` - str 时间戳
- `avgBidPrice` - float 委买均价
- `totalBidQuantity` - int 委买总量
- `avgOfferPrice` - float 委卖均价
- `totalOfferQuantity` - int 委卖总量
- `withdrawBidQuantity` - int 买入撤单总量
- `withdrawBidAmount` - float 买入撤单总额
- `withdrawOfferQuantity` - int 卖出撤单总量
- `withdrawOfferAmount` - float 卖出撤单总额

Level2 逐笔委托

- `time` - int 时间戳
- `stime` - float 时间戳
- `price` - float 委托价
- `volume` - int 委托量
- `entrustNo` - int 委托号
- `entrustType` - int 委托类型
- `entrustDirection` - int 委托方向
 - 0 - 未知
 - 1 - 买入
 - 2 - 卖出
 - 4 - 撤单

Level2 逐笔成交

- `time` - int 时间戳
- `stime` - str 时间戳
- `price` - float 成交价
- `volume` - int 成交量
- `amount` - float 成交额
- `tradeIndex` - int 成交记录号
- `buyNo` - int 买方委托号
- `sellNo` - int 卖方委托号
- `tradeType` - int 成交类型
- `tradeFlag` - int 成交标志
 - 0 - 未知

- 1 - 外盘
- 2 - 内盘
- 3 - 撤单

Level2 逐笔成交统计

- `time` - int 时间戳
- `stime` - str 时间戳
- `bidNumber` - int 主买单总单数
- `bidMostVolmue` - int 主买特大单成交量
- `bidBigVolmue` - int 主买大单成交量
- `bidMediumVolmue` - int 主买中单成交量
- `bidSmallVolmue` - int 主买小单成交量
- `offNumber` - int 主卖单总单数
- `offMostVolmue` - int 主卖特大单成交量
- `offBigVolmue` - int 主卖大单成交量
- `offMediumVolmue` - int 主卖中单成交量
- `offSmallVolmue` - int 主卖小单成交量
- `bidMostAmount` - float 主买特大单成额
- `bidBigAmount` - float 主买大单成交额
- `bidMediumAmount` - float 主买中单成交额
- `bidSmallAmount` - float 主买小单成交额
- `offMostAmount` - float 主卖特大单成交额
- `offBigAmount` - float 主卖大单成交额
- `offMediumAmount` - float 主卖中单成交额
- `offSmallAmount` - float 主卖小单成交额
- `ddx` - float 大单动向
- `ddy` - float 涨跌动因
- `ddz` - float 大单差分
- `zjbyNetInflow` - int 资金博弈 净流入
- `zjbyMost` - int 资金博弈 超大单
- `zjbyBig` - int 资金博弈 大单
- `zjbyMedium` - int 资金博弈 中单
- `zjbySmall` - int 资金博弈 小单
- `netOrder` - int 净挂
- `netwithdraw` - int 净撤
- `withdrawBid` - int 总撤买
- `withdrawOff` - int 总撤卖
- `unactiveBidMostVolmue` - int 被动买特大单成交量
- `unactiveBidBigVolmue` - int 被动买大单成交量
- `unactiveBidMediumVolmue` - int 被动买中单成交量
- `unactiveBidSmallVolmue` - int 被动买小单成交量
- `unactiveOffMostVolmue` - int 被动卖特大单成交量
- `unactiveOffBigVolmue` - int 被动卖大单成交量
- `unactiveOffMediumVolmue` - int 被动卖中单成交量
- `unactiveOffSmallVolmue` - int 被动卖小单成交量
- `unactiveBidMostAmount` - float 被动买特大单成额
- `unactiveBidBigAmount` - float 被动买大单成交额
- `unactiveBidMediumAmount` - float 被动买中单成交额
- `unactiveBidSmallAmount` - float 被动买小单成交额

- `unactiveOffMostAmount` - float 被动卖特大单成交额
- `unactiveOffBigAmount` - float 被动卖大单成交额
- `unactiveOffMediumAmount` - float 被动卖中单成交额
- `unactiveOffSmallAmount` - float 被动卖小单成交额

Level2 委买委卖队列

- `time` - int 时间戳
- `stime` - str 时间戳
- `bidLevelPrice` - float 委买价
- `bidLevelVolume` - list[int] 委买量
- `offerLevelPrice` - float 委卖价
- `offerLevelVolume` - list[int] 委卖量

证券状态

- 0,10 - 默认为未知
- 11 - 开盘前S
- 12 - 集合竞价时段C
- 13 - 连续交易T
- 14 - 休市B
- 15 - 闭市E
- 16 - 波动性中断V
- 17 - 临时停牌P
- 18 - 收盘集合竞价U
- 19 - 盘中集合竞价M
- 20 - 暂停交易至闭市N
- 21 - 获取字段异常
- 22 - 盘后固定价格行情
- 23 - 盘后固定价格行情完毕

委托方向

- 0 - 未知
- 1 - 买入
- 2 - 卖出
- 4 - 撤单

成交标记

- 0 - 未知
- 1 - 外盘
- 2 - 内盘
- 3 - 撤单