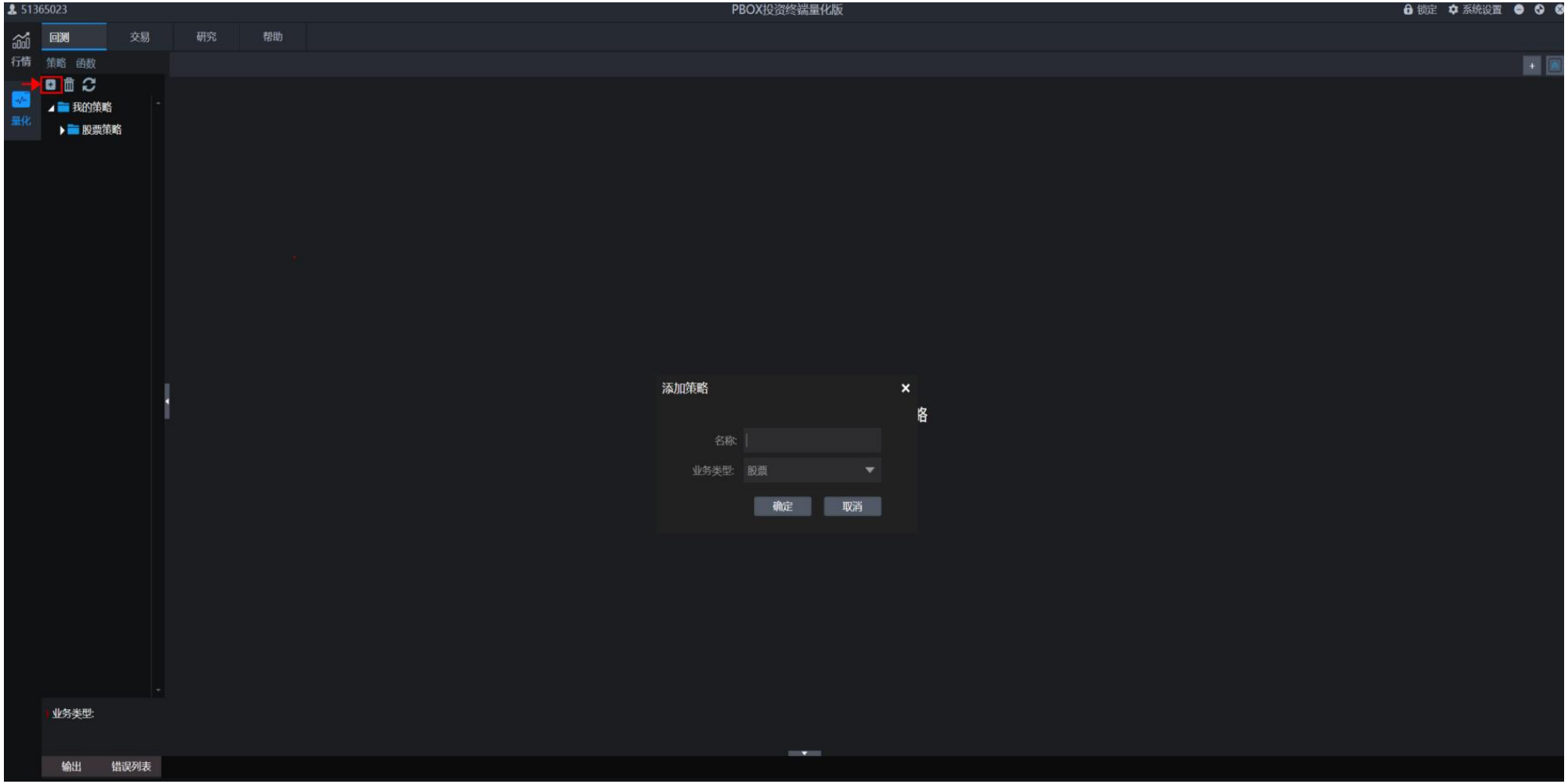
API 文档

使用说明

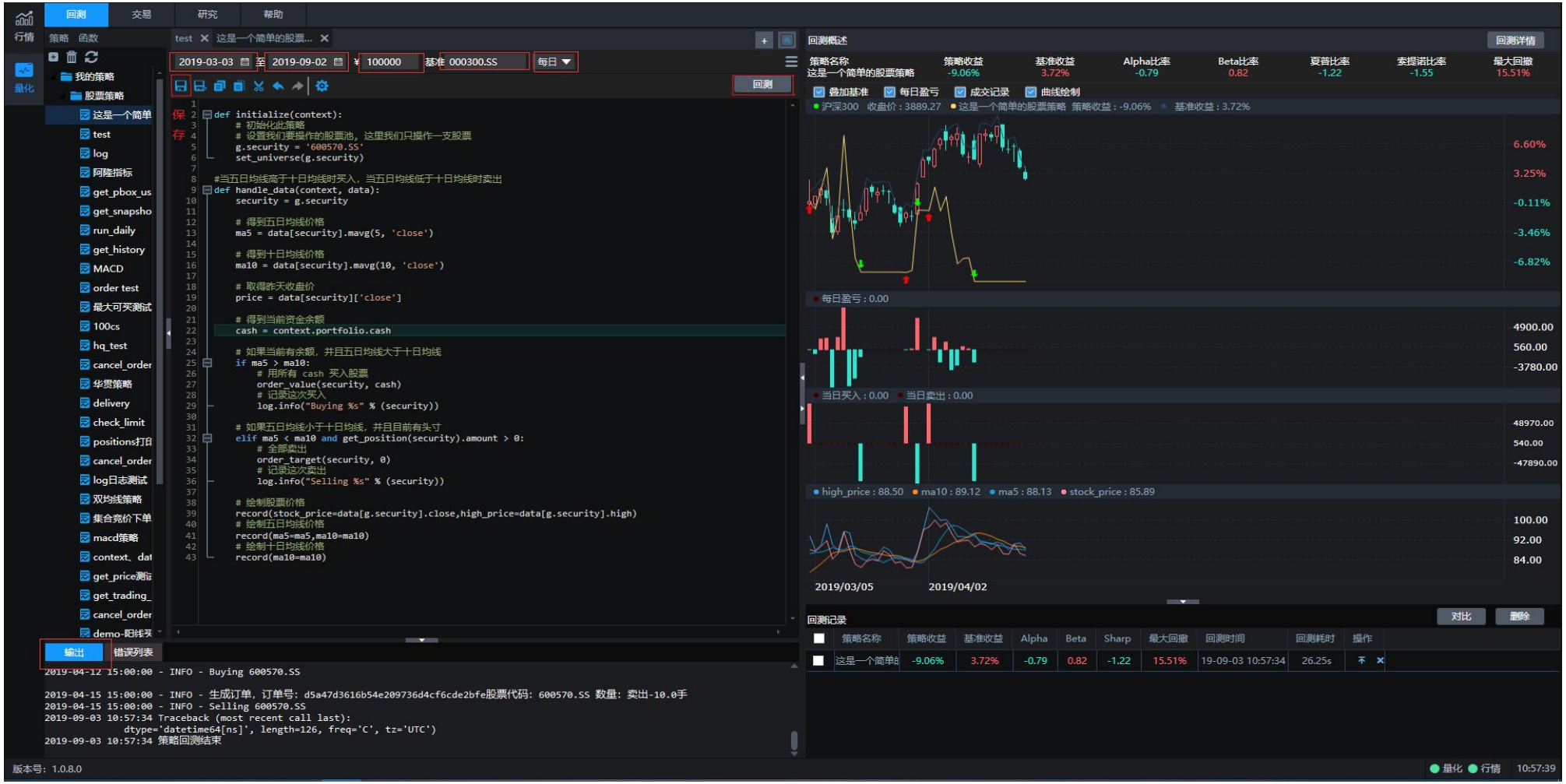
新建策略

开始回测和交易前需要先新建策略，点击下图中左上角标识进行策略添加。可以选择不同的业务类型（比如股票），然后给策略设定一个名称，添加成功后可以在默认策略模板基础上进行策略编写。


51365023
PBOX投资终端量化版
锁定
系统设置
回测
交易
研究
帮助
行情
策略
函数
量化
我的策略
股票策略
添加策略
名称:
业务类型:
股票
确定
取消
业务类型:
输出
错误列表

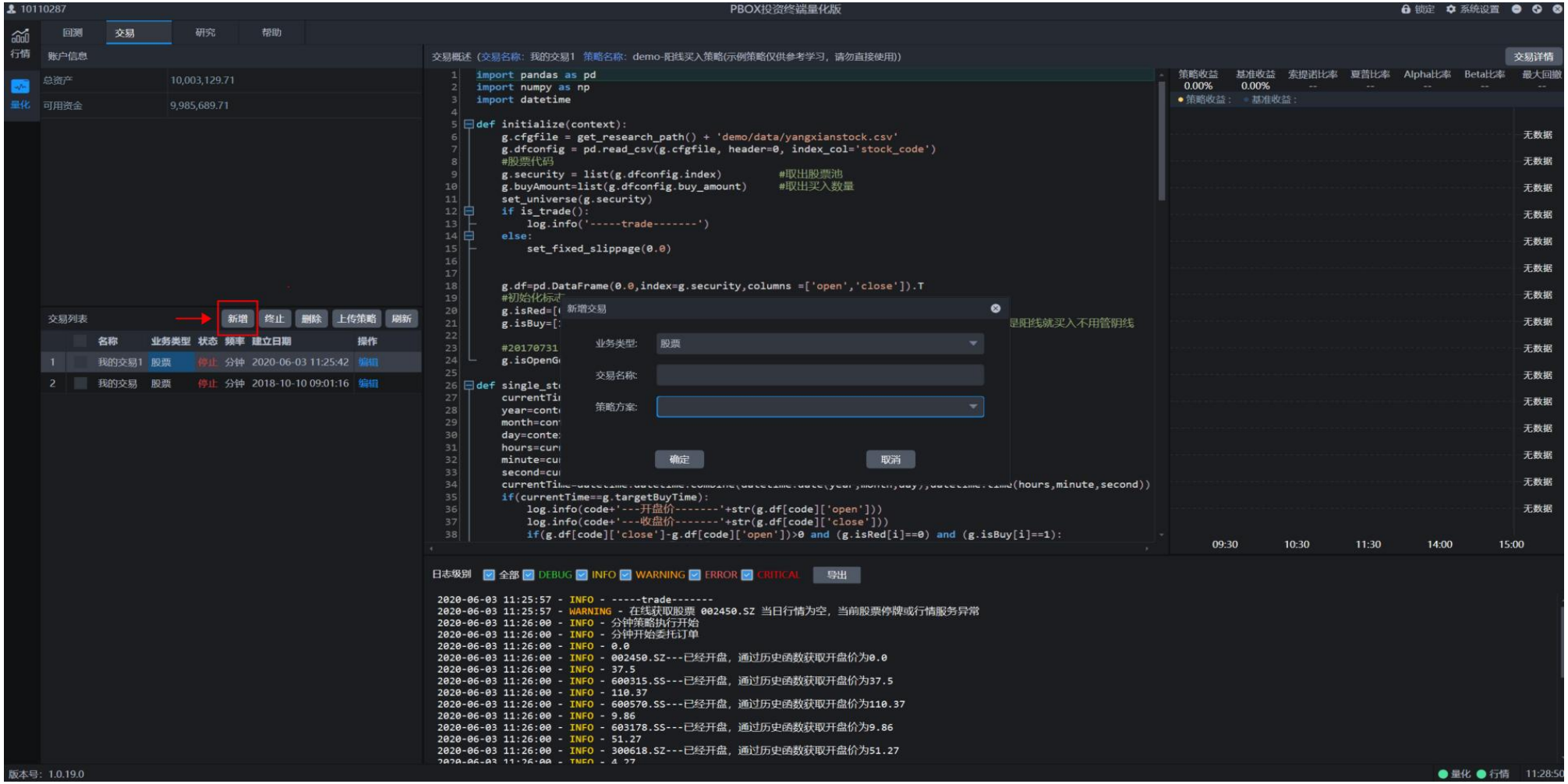
新建回测

策略添加完成后就可以开始进行回测操作了。回测之前需要对开始时间、结束时间、回测资金、回测基准、回测频率几个要素进行设定，设定完毕后点击保存。然后再点击回测按键，系统就会开始运行回测，回测的评价指标、收益曲线、日志都会在界面中展现。

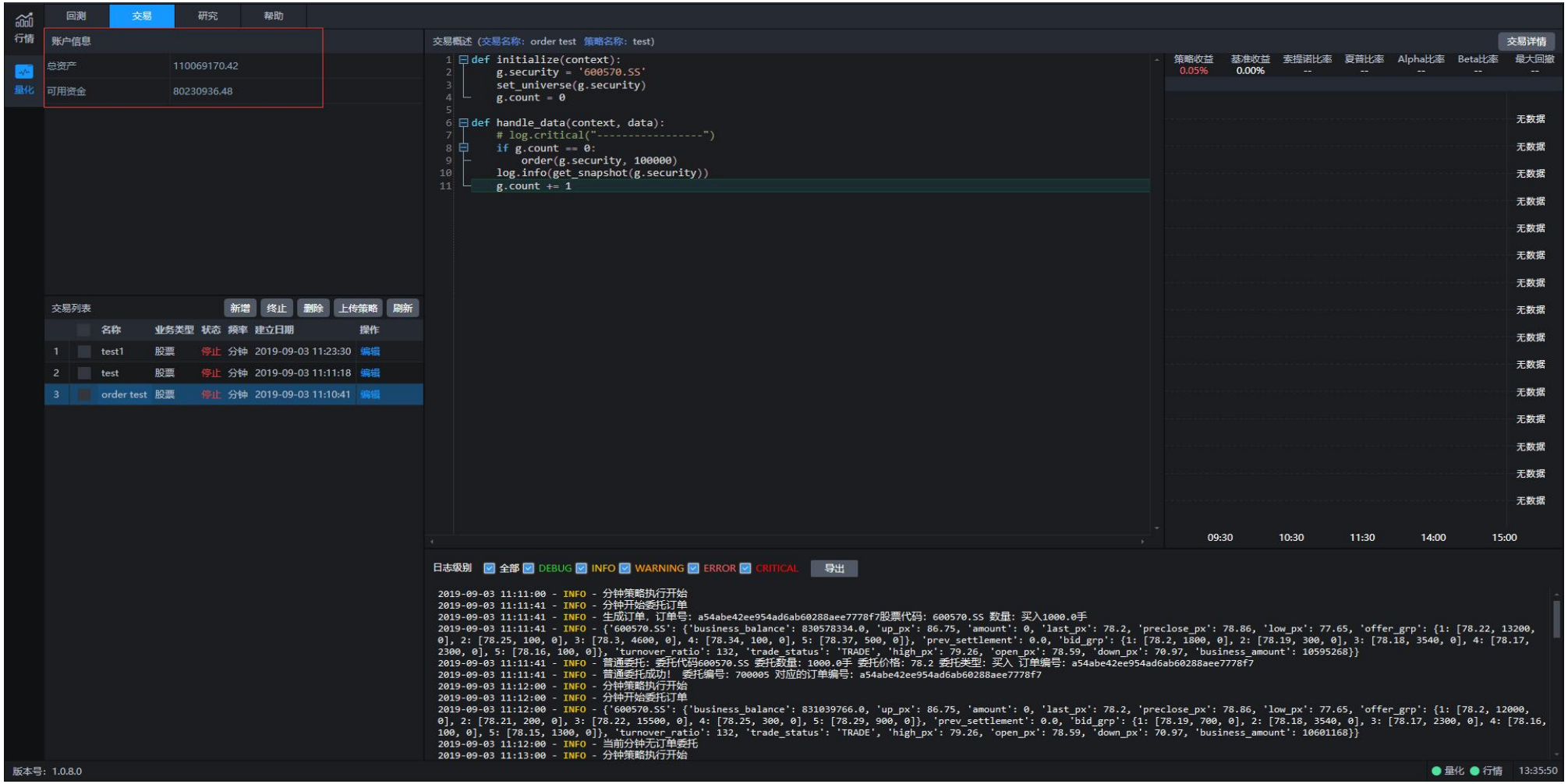


新建交易

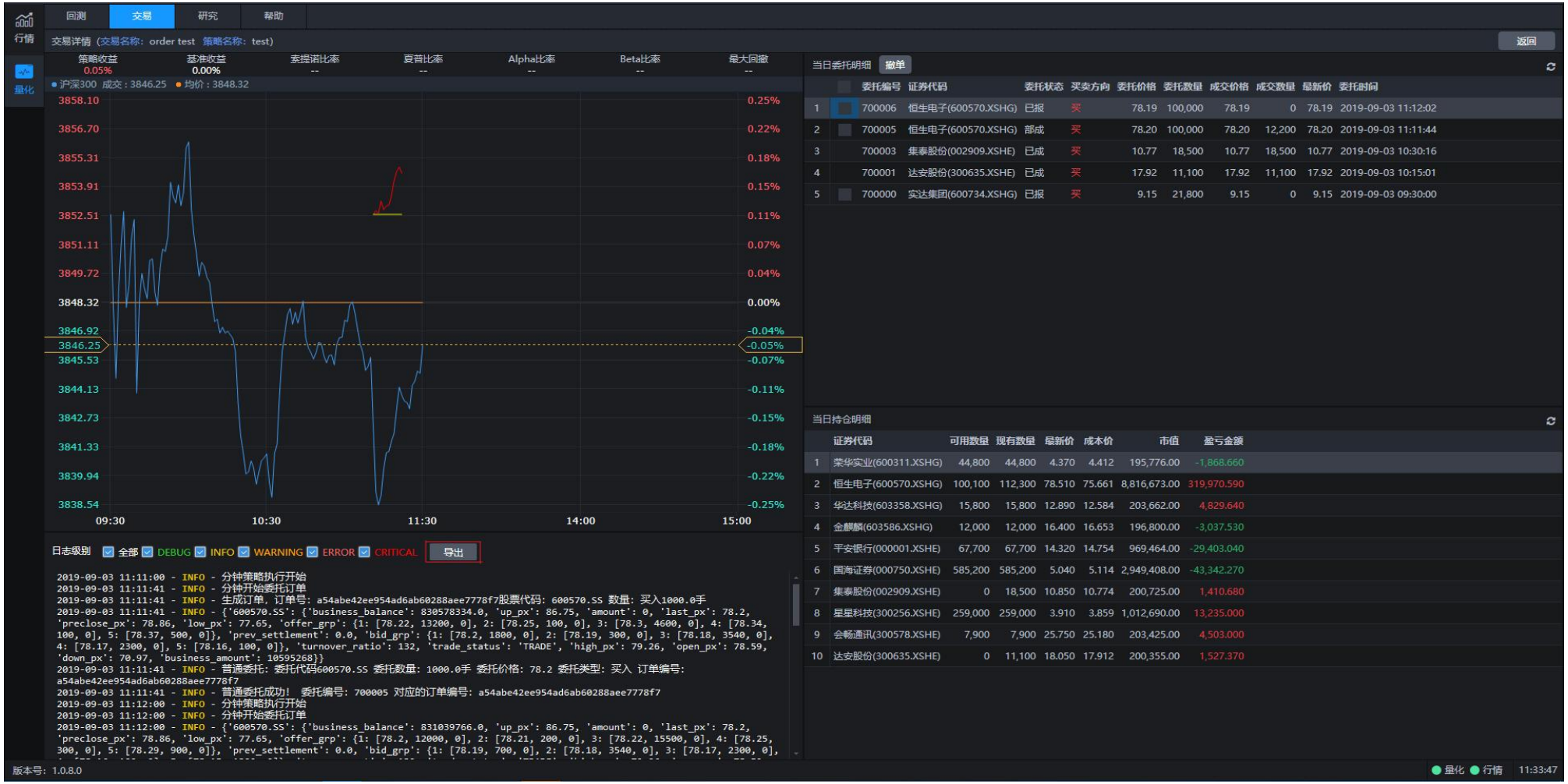
交易界面点击新增按键进行新增交易操作，策略方案中的对象为所有策略列表中的策略，给本次交易设定名称并点击确定后系统就开始运行交易了。



交易开始运行后，可以实时看到总资产和可用资金情况，同时可以在交易列表查询交易状态。



交易开始运行后，可以点击交易详情，查看策略评价指标、交易明细、持仓明细、交易日志。



策略运行周期

回测支持日线级别、分钟级别运行，详见 handle_data 方法。

交易支持日线级别、分钟级别、tick 级别运行，日线级别和分钟级别详见 handle_data 方法，tick 级别运行详见 run_interval 和 tick_data 方法。

频率：日线级别

当选择日线频率时，回测和交易都是每天运行一次，运行时间为每天盘后。

频率：分钟级别

当选择分钟频率时，回测和交易都是每分钟运行一次，运行时间为每根分钟 K 线结束。

频率：tick 级别

当选择 tick 频率时，交易最小频率可以达到 3 秒运行一次。

策略运行时间

盘前运行:

9:30 分钟之前为盘前运行时间，交易环境支持运行在 run_daily 中指定交易时间(如 time='09:15')运行的函数；回测环境和交易环境支持运行 before_trading_start 函数

盘中运行:

9:31(回测)/9:30(交易)~15:00 分钟为盘中运行时间，分钟级别回测环境和交易环境支持运行在 run_daily 中指定交易时间(如 time='14:30')运行的函数；回测环境和交易环境支持运行 handle_data 函数；交易环境支持运行 run_interval 函数

盘后运行:

15:30 分钟为盘后运行时间，回测环境和交易环境支持运行 after_trading_end 函数(该函数为定时运行)；15:00 之后交易环境支持运行在 run_daily 中指定交易时间(如 time='15:10')运行的函数，

交易策略委托下单时间

使用 order 系列接口进行股票委托下单:

1、从早盘开始交易时间（默认为 9:15，服务端可配置）到股票市场收盘（15:00），使用 order 系列接口进行股票委托下单直接报单到柜台。

2、早盘开始交易时间（默认为 9:15，服务端可配置）之前，使用 order 系列接口进行股票委托下单，将在股票市场开盘（9:30）统一委托报单到柜台。

3、股票市场收盘（15:00）之后，使用 order 系列接口进行股票委托下单，将在第二天股票市场开盘（9:30）统一委托报单到柜台。

4、在 15:00-AFTER_TRADING_END(默认 15:30)之间，使用 run_daily 进行委托下单直接报单到柜台。

回测支持业务类型

目前所支持的业务类型:

1、普通股票买卖(单位：股)。

2、可转债买卖(单位：张，T+0)。

3、融资融券担保品买卖(单位：股)。

4、期货投机类型交易(单位：手，T+0)。

5、LOF 基金买卖(单位：股)。

6、ETF 基金买卖(单位：股)。

交易支持业务类型

目前所支持的业务类型:

1、普通股票买卖(单位：股)。

2、可转债买卖(具体单位请咨询券商，T+0)。

3、融资融券交易(单位：股)。

4、ETF 申赎、套利(单位：份)。

5、国债逆回购(单位：份)。

6、期货投机类型交易(单位：手，T+0)。

7、LOF 基金买卖(单位：股)。

8、ETF 基金买卖(单位：股)。

开始写策略

简单但是完整的策略

先来看一个简单但是完整的策略:

```
def initialize(context):set_universe('600570.SS')def handle_data(context, data):pass
```

一个完整策略只需要两步:

1. set_universe: 设置我们要操作的股票池，上面的例子中，只操作一支股票:'600570.SS'，恒生电子。所有的操作只能对股票池的标的进行。
2. 实现一个函数:handle_data。

这是一个完整的策略，但是我们没有任何交易，下面我们来添加一些交易

添加一些交易

```
def initialize(context):g.security = '600570.SS'# 是否创建订单标识 g.flag = Falseset_universe(g.security)def handle_data(context, data):if n
ot g.flag:order(g.security, 1000)g.flag = True
```

这个策略里，当我们没有创建订单时就买入 1000 股'600570.SS'，具体的下单 API 请看 order 函数。这里我们有了交易，但是只是无意义的交易，没有依据当前的数据做出合理的分析。

实用的策略

下面我们来看一个真正实用的策略

在这个策略里，我们会根据历史价格做出判断:

- 如果上一时间点价格高出五天平均价 1%，则全仓买入
- 如果上一时间点价格低于五天平均价，则空仓卖出

```
def initialize(context):
    g.security = '600570.SS'
    set_universe(g.security)

def handle_data(context, data):
    security = g.security
    sid = g.security

    # 取得过去五天的历史价格
    df = get_history(5, '1d', 'close', security, fq=None, include=False)

    # 取得过去五天的平均价格
    average_price = round(df['close'][-5:].mean(), 3)

    # 取得上一时间点价格
    current_price = data[sid]['close']

    # 取得当前的现金
    cash = context.portfolio.cash

    # 如果上一时间点价格高出五天平均价 1%, 则全仓买入
    if current_price > 1.01*average_price:
        # 用所有 cash 买入股票
        order_value(g.security, cash)
```

```
        log.info('buy %s' % g.security)
    # 如果上一时间点价格低于五天平均价, 则空仓卖出
    elif current_price < average_price and get_position(security).amount > 0:
        # 卖出所有股票,使这只股票的最终持有量为 0
        order_target(g.security, 0)
        log.info('sell %s' % g.security)
```

模拟盘和实盘注意事项

关于持久化

为什么要做持久化处理

服务器异常、策略优化等诸多场景，都会使得正在进行的模拟盘和实盘策略存在中断后再重启的需求，但是一旦交易中止后，策略中存储在内存中的全局变量就清空了，因此通过持久化处理为量化交易保驾护航必不可少。

量化框架持久化处理

使用 pickle 模块保存股票池、账户信息、订单信息、全局变量 g 定义的变量等内容。

注意事项：

1. 框架会在 before_trading_start（隔日开始）、handle_data、after_trading_end 事件后触发持久化信息更新及保存操作；

2. 券商升级/环境重启后恢复交易时，框架会先执行策略 initialize 函数再执行持久化信息恢复操作。如果持久化信息保存有策略定义的全局对象 g 中的变量，将会以持久化信息中的变量覆盖掉 initialize 函数中初始化的该变量。
3. 全局变量 g 中不能被序列化的变量将不会被保存。您可在 initialize 中初始化该变量时名字以'__'开头；
4. 涉及到 IO(打开的文件，实例化的类对象等)的对象是不能被序列化的；
5. 全局变量 g 中以'__'开头的变量为私有变量，持久化时将不会被保存；

示例

```
class Test(object):count = 5def print_info(self):self.count += 1log.info("a" * self.count)def initialize(context):g.security = "600570.SS"
set_universe(g.security)# 初始化无法被序列化类对象，并赋值为私有变量，落地持久化信息时跳过保存该变量 g.__test_class = Test()def handle_data(conte
xt, data):# 调用私有变量中定义的方法 g.__test_class.print_info()
```

策略中持久化处理方法

使用 pickle 模块保存 g 对象(全局变量)。

示例

```
import pickle
from collections import defaultdict
NOTEBOOK_PATH = '/home/fly/notebook/''''
持仓 N 日后卖出，仓龄变量每日 pickle 进行保存，重启策略后可以保证逻辑连贯
'''def initialize(context):#尝试启动 pickle 文件 try:with open(NOTEBOOK_PATH+'hold_days.pkl','rb') as f:g.hold_days = pickle.load(f)#定义空的
全局字典变量 except:g.hold_days = defaultdict(list)g.security = '600570.SS'set_universe(g.security)# 仓龄增加一天 def before_trading_start(co
```

```
ntext, data):if g.hold_days:g.hold_days[g.security] += 1# 每天将存储仓龄的字典对象进行 pickle 保存 def handle_data(context, data):if g.securit
y not in list(context.portfolio.positions.keys()) and g.security not in g.hold_days:order(g.security, 100)g.hold_days[g.security] = 1if g.
hold_days:if g.hold_days[g.security] > 5:order(g.security, -100)del g.hold_days[g.security]with open(NOTEBOOK_PATH+'hold_days.pkl','wb') a
s f:pickle.dump(g.hold_days,f,-1)
```

策略引擎简介

业务流程框架

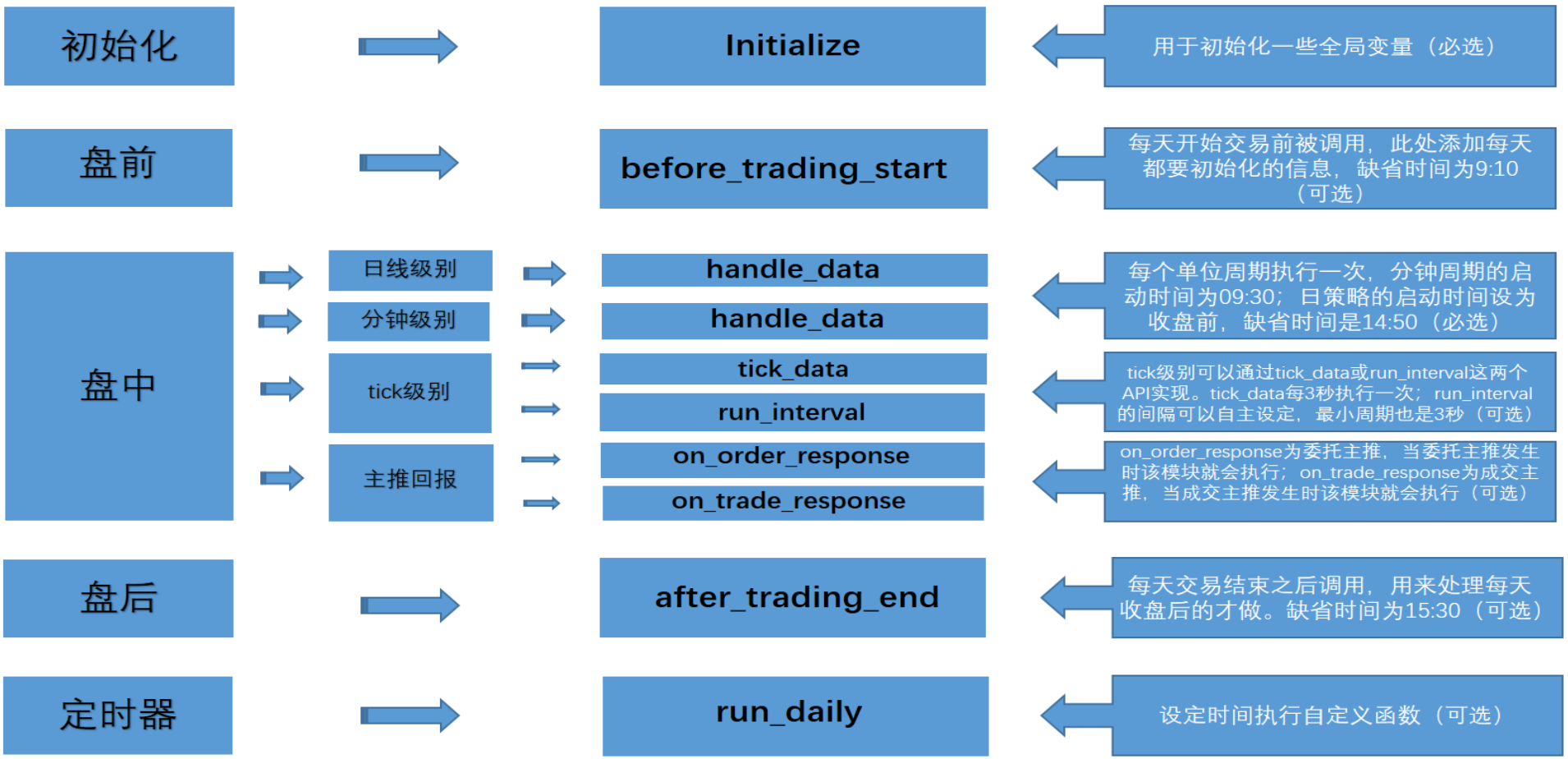
ptrade 量化引擎以事件触发为基础，通过初始化事件（initialize）、盘前事件（before_trading_start）、盘中事件（handle_data）、盘后事件（after_trading_end）来完成每个交易日的策略任务。

initialize 和 handle_data 是一个允许运行策略的最基础结构，也就是必选项，before_trading_start 和 after_trading_end 是可以按需运行的。

handle_data 仅满足日线和分钟级别的盘中处理，tick 级别的盘中处理则需要通过 tick_data 或者 run_interval 来实现。

ptrade 还支持委托主推事件（on_order_respense）、交易主推事件（on_trade_response），可以通过委托和成交的信息来处理策略逻辑，是 tick 级的一个补充。

除了以上的一些事件以外，ptrade 也支持通过定时任务来运行策略逻辑，可以通过 run_daily 接口实现。



initialize（必选）

```
initialize(context)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数用于初始化一些全局变量，是策略运行的唯二必须定义函数之一。

注意事项：

该函数只会在回测和交易启动的时候运行一次

可调用接口

set_universe(回测/交易)	set_benchmark(回测/交易)	set_commission(回测)
set_fixed_slippage(回测)	set_slippage(回测)	set_volume_ratio(回测)
set_limit_mode(回测)	set_yesterday_position(回测)	run_daily(回测/交易)
run_interval(交易)	get_trading_day(研究/回测/交易)	get_all_trades_days(研究/回测/交易)
get_trade_days(交易)	convert_position_from_csv(回测)	get_user_name(回测/交易)
is_trade(回测/交易)	get_research_path(回测/交易)	permission_test(交易)
set_future_commission(回测(期货))	set_margin_rate(回测(期货))	get_margin_rate(回测(期货))

create_dir(回测/交易)

参数

context: Context 对象，存放有当前的账户及持仓信息；

返回

None

示例

```
def initialize(context):#g 为全局对象 g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order('600570.SS',100)
```

before_trading_start（可选）

```
before_trading_start(context, data)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数在每天开始交易前被调用一次，用于添加每天都要初始化的信息，如无盘前初始化需求，该函数可以在策略中不做定义。

注意事项：

1. 在回测中，该函数在每个回测交易日 8:30 分执行。
2. 在交易中，该函数在开启交易时立即执行，从隔日开始每天 9:10 分(默认)执行。
3. 当在 9:10 前开启交易时，受行情未更新原因在该函数内调用实时行情接口会导致数据有误。可通过在该函数内 sleep 至 9:10 分或调用实时行情接口改为 run_daily 执行等方式进行避免。

可调用接口

set_universe(回测/交易)	get_Ashares(研究/回测/交易)	set_yesterday_position(回测)
get_stock_info(研究/回测/交易)	get_index_stocks(研究/回测/交易)	get_fundamentals(研究/回测/交易)
get_trading_day(回测/交易)	get_all_trades_days(研究/回测/交易)	get_trade_days(研究/回测/交易)
get_history(回测/交易)	get_price(研究/回测/交易)	get_individual_entrust(交易)
get_individual_transcation(交易)	convert_position_from_csv(回测)	get_stock_name(研究/回测/交易)
get_stock_status(研究/回测/交易)	get_stock_exrights(研究/回测/交易)	get_stock_blocks(研究/回测/交易)
get_etf_list(交易)	get_industry_stocks(研究/回测/交易)	get_user_name(回测/交易)

get_cb_list(交易)	get_deliver(交易)	get_fundjour(交易)
get_research_path(回测/交易)	get_market_list(研究/回测/交易)	get_market_detail(研究/回测/交易)
permission_test(交易)	get_trade_name(交易)	set_future_commission(回测(期货))
set_margin_rate(回测(期货))	get_margin_rate(回测(期货))	get_instruments(回测/交易(期货))
get_MACD(回测/交易)	get_KDJ(回测/交易)	get_RSI(回测/交易)
get_CCI(回测/交易)	create_dir(回测/交易)	

参数

context: Context 对象，存放有当前的账户及持仓信息；

data：保留字段暂无数据；

返回

None

示例

```
def initialize(context):
```

```
    #g 为全局变量
    g.security = '600570.SS'
    set_universe(g.security)def before_trading_start(context, data):
    log.info(g.security)def handle_data(context, data):
    order('600570.SS',100)
```

handle_data（必选）

```
handle_data(context, data)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数在交易时间内按指定的周期频率运行，是用于处理策略交易的主要模块，根据策略保存时的周期参数分为每分钟运行和每天运行，是策略运行的唯二必须定义函数之一。

注意事项：

1. 该函数每个单位周期执行一次
2. 如果是日线级别策略，每天执行一次。股票回测场景下，在 15:00 执行；股票交易场景下，执行时间为券商实际配置时间。
3. 如果是分钟级别策略，每分钟执行一次，股票回测场景下，执行时间为 9:31 -- 15:00，股票交易场景下，执行时间为 9:30 -- 14:59。

4. 回测与交易中，handle_data 函数不会在非交易日触发（如回测或交易起始日期为 2015 年 12 月 21 日，则策略在 2016 年 1 月 1 日-3 日时，handle_data 不会运行，4 日继续运行）。

可调用接口

get_trading_day(回测/交易)	get_all_trades_days(研究/回测/交易)	get_trade_days(研究/回测/交易)
get_history(回测/交易)	get_price(研究/回测/交易)	get_individual_entrust(交易)
get_individual_transcation(交易)	get_gear_price(交易)	get_stock_name(研究/回测/交易)
get_stock_status(研究/回测/交易)	get_stock_exrights(研究/回测/交易)	get_stock_blocks(研究/回测/交易)
get_index_stocks(研究/回测/交易)	get_industry_stocks(研究/回测/交易)	get_fundamentals(研究/回测/交易)
get_Ashares(研究/回测/交易)	get_snapshot(交易)	convert_position_from_csv(回测)
order(回测/交易)	order_target(回测/交易)	order_value(回测/交易)
order_target_value(回测/交易)	order_market(交易)	ipo_stocks_order(交易)
after_trading_order(交易)	after_trading_cancel_order(交易)	etf_basket_order(交易)
etf_purchase_redemption(交易)	cancel_order(回测/交易)	get_stock_info(研究/回测/交易)
get_order(回测/交易)	get_orders(回测/交易)	get_open_orders(回测/交易)

get_trades(回测/交易)	get_position(回测/交易)	get_positions(回测/交易)
get_etf_info(交易)	get_etf_stock_info(交易)	get_etf_stock_list(交易)
get_etf_list(交易)	get_all_orders(交易)	cancel_order_ex(交易)
debt_to_stock_order(交易)	get_user_name(回测/交易)	get_research_path(回测/交易)
get_marginsec_stocks(交易)	get_margincash_stocks(交易)	debt_to_stock_order(交易)
get_margin_contractreal(交易)	get_margin_contract(交易)	marginsec_direct_refund(交易)
get_margin_entrans_amount(交易)	get_margin_contract(交易)	margincash_direct_refund(交易)
marginsec_open(交易)	marginsec_close(交易)	margincash_open(交易)
margincash_close(交易)	margin_trade(交易)	get_marginsec_close_amount(交易)
get_marginsec_open_amount(交易)	get_margincash_close_amount(交易)	get_margincash_open_amount(交易)
get_cb_list(交易)	get_tick_direction(交易)	get_sort_msg(交易)
get_trade_name(交易)	get_margin_rate(回测(期货))	get_instruments(回测/交易(期货)
buy_open(回测/交易(期货)	sell_close(回测/交易(期货)	sell_close(回测/交易(期货)
buy_close(回测/交易(期货)	get_MACD(回测/交易)	get_KDJ(回测/交易)
get_RSI(回测/交易)	get_CCI(回测/交易)	create_dir(回测/交易)

参数

context: Context 对象，存放有当前的账户及持仓信息；

data：一个字典(dict)，key 是标的代码，value 是当时的 SecurityUnitData 对象，存放当前周期（日线策略，则是当天；分钟策略，则是这一分钟）的数据；

注意：为了加速，data 中的数据只包含股票池中所订阅标的的信息，可使用 data[security]的方式来获取当前周期对应的标的信息；

返回

None

示例

```
def initialize(context):#g 为全局变量 g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order('600570.SS',100)
```

after_trading_end（可选）

```
after_trading_end(context, data)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数会在每天交易结束之后调用，用于处理每天收盘后的操作，如无盘后处理需求，该函数可以在策略中不做定义。

注意事项：

1. 该函数只会执行一次
2. 该函数执行时间为由券商配置决定，一般为 15:30。

可调用接口

get_trades_file(回测)	get_stock_info(研究/回测/交易)	get_open_orders(回测/交易)
get_all_trades_days(研究/回测/交易)	get_trade_days(研究/回测/交易)	get_history(回测/交易)
get_price(研究/回测/交易)	get_individual_entrust(交易)	get_individual_transcation(交易)
get_Ashares(研究/回测/交易)	get_stock_name(研究/回测/交易)	get_stock_status(研究/回测/交易)
get_stock_exrights(研究/回测/交易)	get_stock_blocks(研究/回测/交易)	get_index_stocks(研究/回测/交易)
get_industry_stocks(研究/回测/交易)	get_fundamentals(研究/回测/交易)	get_user_name(回测/交易)

get_cb_list(交易)	get_deliver(交易)	get_fundjour(交易)
get_research_path(回测/交易)	get_trade_name(交易)	get_market_list(研究/回测/交易)
get_market_detail(研究/回测/交易)	permission_test(交易)	get_tick_direction(交易)
get_sort_msg(交易)	get_margin_rate(回测(期货))	get_margin_rate(回测(期货))
get_instruments(回测/交易(期货))	get_MACD(回测/交易)	get_KDJ(回测/交易)
get_RSI(回测/交易)	get_CCI(回测/交易)	create_dir(回测/交易)

参数

context: Context 对象，存放有当前的账户及持仓信息；

data：保留字段暂无数据；

返回

None

示例

```
def initialize(context):#g 为全局变量 g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order('600570.SS',100)d
ef after_trading_end(context, data):log.info(g.security)
```

tick_data（可选）

```
tick_data(context, data)
```

使用场景

该函数仅交易模块可用

接口说明

该函数可以用于处理 tick 级别策略的交易逻辑，每隔 3 秒执行一次，如无 tick 处理需求，该函数可以在策略中不做定义。

注意事项：

1. 该函数执行时间为 9:30 -- 14:59。
2. 该函数中只能使用 order_tick 进行对应的下单操作。
3. 该函数中的 data 和 handle_data 函数中的 data 是不一样的，请勿混肴。
4. 参数 data 中包含的逐笔委托，逐笔成交数据需开通 level2 行情才有数据推送，否则对应数据返回 None。

可调用接口

get_trading_day(研究/回测/交易)	get_all_trades_days(研究/回测/交易)	get_trade_days(研究/回测/交易)
get_cb_list(交易)	get_history(回测/交易)	get_price(研究/回测/交易)
get_individual_entrust(交易)	get_individual_transcation(交易)	get_tick_direction(交易)
get_sort_msg(交易)	get_etf_info(交易)	get_etf_stock_info(交易)
get_gear_price(交易)	get_snapshot(交易)	get_stock_name(研究/回测/交易)
get_stock_info(研究/回测/交易)	get_stock_status(研究/回测/交易)	get_stock_exrights(研究/回测/交易)
get_stock_blocks(研究/回测/交易)	get_index_stocks(研究/回测/交易)	get_etf_stock_list(交易)
get_industry_stocks(研究/回测/交易)	get_fundamentals(研究/回测/交易)	get_Ashares(研究/回测/交易)
get_etf_list(交易)	get_user_name(回测/交易)	get_research_path(研究/回测/交易)
get_trade_name(交易)	set_universe(回测/交易)	order(回测/交易)
order_target(回测/交易)	order_value(回测/交易)	order_target_value(回测/交易)
order_market(交易)	ipo_stocks_order(交易)	after_trading_order(交易)
after_trading_cancel_order(交易)	etf_basket_order(交易)	etf_purchase_redemption(交易)
get_positions(回测/交易)	order_tick(交易)	cancel_order(回测/交易)
cancel_order_ex(交易)	debt_to_stock_order(交易)	get_open_orders(回测/交易)

get_order(回测/交易)	get_orders(回测/交易)	get_all_orders(交易)
get_trades(回测/交易)	get_position(回测/交易)	margin_trade(交易)
margincash_open(交易)	margincash_close(交易)	margincash_direct_refund(交易)
marginsec_open(交易)	marginsec_close(交易)	marginsec_direct_refund(交易)
get_margincash_stocks(交易)	get_marginsec_stocks(交易)	get_margin_contract(交易)
get_margin_contractreal(交易)	get_margin_assert(交易)	get_assure_security_list(交易)
get_margincash_open_amount(交易)	get_margincash_close_amount(交易)	get_marginsec_open_amount(交易)
get_marginsec_close_amount(交易)	get_margin_entrans_amount(交易)	buy_open(回测/交易(期货))
sell_close(回测/交易(期货))	sell_open(回测/交易(期货))	buy_close(回测/交易(期货))
get_instruments(回测/交易(期货))	log(回测/交易)	is_trade(回测/交易)
check_limit(交易)	send_email(交易)	send_qywx(交易)
get_MACD(回测/交易)	get_KDJ(回测/交易)	get_RSI(回测/交易)
get_CCI(回测/交易)	create_dir(回测/交易)	

参数

context: Context 对象，存放有当前的账户及持仓信息；

data: 一个字典（dict），key 为对应的标的代码（如：'600570.SS'），value 为一个字典（dict），包含 order（逐笔委托）、tick（当前 tick 数据）、transcation（逐笔成交）三项

结构如下：

```
{'股票代码':
{
'order(最近一条逐笔委托)':DataFrame/None,
'tick(当前 tick 数据)':DataFrame,
'transcation(最近一条逐笔成交)':DataFrame/None,
}
}
```

每项具体介绍：

```
order - 逐笔委托对应 DataFrame 包含字段：
business_time：时间戳毫秒级
hq_px：价格
business_amount：委托量
order_no：委托编号
business_direction：委托方向
0 - 卖；
1 - 买；
2 - 借入；
3 - 出借；
trans_kind：委托类别
```

1-- 市价委托；
2 -- 限价委托；
3 -- 本方最优；
tick - tick 数据对应 DataFrame 包含字段：
amount：持仓量
avg_px：均价
bid_grp：买档位，dict 类型，内容如：{1:[42.71,200,0],2:[42.74,200,0],3:[42.75,700,...，以档位为 Key，以 list 为 Value，每个 Value 包含：委托价格、委托数量和委托笔数；
business_amount：成交数量；
business_amount_in：内盘成交量；
business_amount_out：外盘成交量
business_balance：成交金额；
business_count：成交笔数；
circulation_amount：流通股本；
close_px：今日收盘
current_amount：最近成交量(现手)；
down_px：跌停价格；
end_trade_date：最后交易日
entrust_diff：委差；
entrust_rate：委比；
high_px：最高价；
hsTimeStamp：时间戳，格式为 YYYYMMDDHHMISS，如 20170711141612，表示 2017 年 7 月 11 日 14 时 16 分 12 秒的 tick 数据信息；
issue_date：上市日期
last_px：最新成交价；
low_px：最低价；
offer_grp：卖档位，dict 类型，内容如：{1:[42.71,200,0],2:[42.74,200,0],3:[42.75,700,...，以档位为 Key，以 list 为 Value，每个 Value 包含：委托价格、委托数量和委托笔数；
open_px：今开盘价；
pb_rate：市净率；
pe_rate：动态市盈率；
preclose_px：昨收价；
prev_settlement：昨结算

start_trade_date：首个交易日
tick_size：最小报价单位
trade_mins：交易时间，距离开盘已过交易时间，如 100 则表示每日 240 分钟交易时间中的第 100 分钟；
trade_status：交易状态；
START -- 市场启动(初始化之后，集合竞价前)
PRETR -- 盘前
OCALL -- 开始集合竞价
TRADE -- 交易(连续撮合)
HALT -- 暂停交易
SUSP -- 停盘
BREAK -- 休市
POSTR -- 盘后
ENDTR -- 交易结束
STOPT -- 长期停盘，停盘 n 天，n>=1
DELISTED -- 退市
POSMT -- 盘后交易
PCALL -- 盘后集合竞价
INIT -- 盘后固定价格启动前
ENDPT -- 盘后固定价格闭市阶段
POSSP -- 盘后固定价格停牌
turnover_ratio：换手率
up_px：涨停价格；
vol_ratio：量比；
wavg_px：加权平均价；
transcation - 逐笔成交对应 DataFrame 包含字段：
business_time：时间戳毫秒级；
hq_px：价格；
business_amount：成交量；
trade_index：成交编号；
business_direction：成交方向；
0 – 卖；
1 – 买；

```
buy_no: 叫买方编号;
sell_no: 叫卖方编号;
trans_flag: 成交标记;
0 - 普通成交;
1 - 撤单成交;
trans_identify_am: 盘后逐笔成交序号标识;
0 - 盘中;
1 - 盘后;
channel_num: 成交通道信息;
```

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def tick_data(context,data):# 获取买一价 security = g.security
current_price = eval(data[security]['tick']['bid_grp'][0])[1][0]log.info(current_price)# 获取买二价# current_price = eval(data[security]['t
ick']['bid_grp'][0])[2][0]# 获取买三量# current_amount = eval(data[security]['tick']['bid_grp'][0])[3][1]# 获取 tick 最高价# current_high_pri
ce = data[security]['tick']['high_px'][0]# 最近一笔逐笔成交的成交量# transcation = data[security]["transcation"]# business_amount = list(tra
nscation["business_amount"])# if len(business_amount) > 0:#     log.info("最近一笔逐笔成交的成交量：%s" % business_amount[0])# 最近一笔逐笔委
托的委托类别# order = data[security]["order"]# trans_kind = list(order["trans_kind"])# if len(trans_kind) > 0:#     log.info("最近一笔逐笔委
托的委托类别：%s" % trans_kind[0])if current_price > 38.19:# 按买一档价格下单 order_tick(security, 100, 1)def handle_data(context, data):pass
```

on_order_response - 委托主推(可选)

```
on_order_response(context, order_list)
```

使用场景

该函数仅在交易模块可用

接口说明

该函数会在委托主推回调时响应，比引擎、get_order()和 get_orders()函数更新 Order 状态的速度更快，适合对速度要求比较高的策略。

注意事项：

1. 目前可接收股票、可转债、ETF、LOF、期货代码的主推数据。
2. 当接到策略外交易产生的主推时(需券商配置默认不推送)，由于没有对应的 Order 对象，主推信息中 order_id 字段赋值为""。
3. 当在主推里调用委托接口时，需要进行判断处理避免无限迭代循环问题。

可调用委托接口

order(回测/交易)	order_target(回测/交易)	order_value(回测/交易)
order_target_value(回测/交易)	order_market(交易)	ipo_stocks_order(交易)
after_trading_order(交易)	after_trading_cancel_order(交易)	etf_basket_order(交易)
etf_purchase_redemption(交易)	cancel_order(回测/交易)	margin_trade(交易)

margincash_open(交易)	margincash_close(交易)	margincash_direct_refund(交易)
margincash_open(交易)	marginsec_close(交易)	marginsec_direct_refund(交易)
get_user_name(回测/交易)	get_cb_list(交易)	get_instruments(回测/交易(期货))

参数

context: Context 对象，存放有当前的账户及持仓信息；

order_list：一个列表，当前委托单发生变化时，发生变化的委托单列表。委托单以字典形式展现，内容包括：'entrust_no'(委托编号), 'error_info'(错误信息), 'order_time'(委托时间), 'stock_code'(股票代码), 'amount'(委托数量), 'price'(委托价格), 'business_amount'(成交数量), 'status'(委托状态), 'order_id'(委托订单号), 'entrust_type'(委托类别), 'entrust_prop'(委托属性), 'order_id'(Order 订单编号)；

字段备注:

- status -- 委托状态，详见 Order 对象；
- entrust_type -- 委托类别(str)
    - 0 -- 委托
    - 2 -- 撤单

- 4 -- 确认
- 6 -- 信用融资
- 7 -- 信用融券
- 9 -- 信用交易

- entrust_prop -- 委托属性(str)
  - 0 -- 买卖
  - 1 -- 配股
  - 3 -- 申购
  - 7 -- 转股
  - 9 -- 股息
  - N -- ETF 申赎
  - Q -- 对手方最优价格
  - R -- 最优五档即时成交剩余转限价
  - S -- 本方最优价格
  - T -- 即时成交剩余撤销

- U -- 最优五档即时成交剩余撤销
- b -- 定价委托
- c -- 确认委托
- d -- 限价委托

返回

None

接收到的主推格式如下:

```
本交易产生的主推: [{'business_amount': 0.0, 'order_id': 'e71d1684c8a74b4ca00b3326c9eb8614', 'order_time': '2022-05-10 15:52:10.780', 'entrust_prop': '0', 'status': '2', 'price': 36.95, 'entrust_no': 700006, 'error_info': '', 'amount': 200, 'stock_code': '600570.SS', 'entrust_type': '0'}]非本交易产生的主推: [{'business_amount': 0.0, 'order_id': '', 'order_time': '2022-05-10 15:54:30.204', 'entrust_prop': '0', 'status': '2', 'price': 36.95, 'entrust_no': 700008, 'error_info': '', 'amount': 200, 'stock_code': '600570.SS', 'entrust_type': '0'}]
```

示例

```
def initialize(context):g.security = ['600570.SS','002416.SZ']set_universe(g.security)g.flag = 0def on_order_response(context, order_list):log.info(order_list)if(g.flag==0):order('600570.SS', 100)g.flag = 1else:log.info("end")def handle_data(context, data):order('600570.SS', 100)
```

on_trade_response - 交易主推(可选)

```
on_trade_response (context, trade_list)
```

使用场景

该函数仅在交易模块可用

接口说明

该函数会在成交主推回调时响应，比引擎和 get_trades()函数更新 Order 状态的速度更快，适合对速度要求比较高的策略。

注意事项：

1. 目前可接收股票、可转债、ETF、LOF、期货代码的主推数据。
2. 当接到策略外交易产生的主推时(需券商配置默认不推送)，由于没有对应的 Order 对象，主推信息中 order_id 字段赋值为""。
3. 当在主推里调用委托接口时，需要进行判断处理避免无限迭代循环问题。

可调用委托接口

order(回测/交易)	order_target(回测/交易)	order_value(回测/交易)
order_target_value(回测/交易)	order_market(交易)	ipo_stocks_order(交易)
after_trading_order(交易)	after_trading_cancel_order(交易)	etf_basket_order(交易)

etf_purchase_redemption(交易)	cancel_order(回测/交易)	margin_trade(交易)
margincash_open(交易)	margincash_close(交易)	margincash_direct_refund(交易)
margincash_open(交易)	marginsec_close(交易)	marginsec_direct_refund(交易)
get_user_name(回测/交易)	get_cb_list(交易)	get_instruments(回测/交易(期货))

参数

context: Context 对象，存放有当前的账户及持仓信息；

trade_list：一个列表，当前成交单发生变化时，发生变化的成交单列表。成交单以字典形式展现，内容包括：'entrust_no'(委托编号), 'business_time'(成交时间), 'stock_code'(股票代码), 'entrust_bs'(成交方向), 'business_amount'(成交数量), 'business_price'(成交价格), 'business_balance'(成交额), 'business_id'(成交编号), 'status'(委托状态), 'order_id'(Order 订单编号)；

字段备注:

- entrust_bs -- 成交方向(str)，1-买，2-卖；
- status -- 委托状态，详见 Order 对象；

返回

None

接收到的主推格式如下:

```
本交易产生的主推：[{'status': '8', 'business_id': '76', 'business_amount': 200, 'order_id': 'e71d1684c8a74b4ca00b3326c9eb8614', 'entrust_no': 700006, 'business_balance': 7390.000000000001, 'business_price': 36.95, 'stock_code': '600570.SS', 'entrust_bs': '1', 'business_time': '2022-05-10 15:51:47'}]非本交易产生的主推：[{'status': '8', 'business_id': 'b155235000000003', 'business_amount': 200, 'order_id': '', 'entrust_no': 700007, 'business_balance': 3000.0, 'business_price': 15.0, 'stock_code': '000001.SZ', 'entrust_bs': '1', 'business_time': '2022-05-10 15:52:35'}]
```

示例

```
def initialize(context):g.security = ['600570.SS','002416.SZ']set_universe(g.security)g.flag = 0def on_trade_response(context, trade_list):log.info(trade_list)if(g.flag==0):order('600570.SS', 100)g.flag = 1else:log.info("end")def handle_data(context, data):order('600570.SS', 100)
```

策略 API 介绍

设置函数

set_universe-设置股票池

```
set_universe(security_list)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数用于设置或者更新此策略要操作的股票池。

注意事项：

股票策略中，该函数只用于设定 get_history 函数的默认 security_list 入参，除此之外并无其他用处，因此为非必须设定的函数。

参数

security_list: 股票列表，支持单支或者多支股票(list[str]/str)

返回

None

示例

```
def initialize(context):g.security = ['600570.SS','600571.SS']# 将 g.security 中的股票设置为股票池 set_universe(g.security)def handle_data(co
```

```
ntext, data):# 获取初始化设定的股票池行情数据 his = get_history(5, '1d', 'close', security_list=None)
```

set_benchmark-设置基准

```
set_benchmark(sids)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数用于设置策略的比较基准，前端展现的策略评价指标都基于此处设置的基准标的。

注意事项：

此函数只能在 initialize 使用。

参数

security：股票/指数/ETF 代码(str)

默认设置

如果不做基准设置，默认选定沪深 300 指数(000300.SS)的每日价格作为判断策略好坏和一系列风险值计算的基准。如果要指定其他股票/指数/ETF 的价格作为基准，就需要使用 set_benchmark。

返回

None

示例

```
def initialize(context):g.security = '000001.SZ'set_universe(g.security)#将上证 50（000016.SS）设置为参考基准 set_benchmark('000016.SS')def handle_data(context, data):order('000001.SZ',100)
```

set_commission-设置佣金费率

```
set_commission(commission_ratio=0.0003, min_commission=5.0, type="STOCK")
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置佣金费率。

注意事项：

关于回测手续费计算：手续费=佣金费+经手费

佣金费=佣金费率*交易总金额(若佣金费计算后小于设置的最低佣金，则佣金费取最小佣金)

经手费=经手费率(万分之 0.487)*交易总金额

参数

commission_ratio：佣金费率，默认股票每笔交易的佣金费率是万分之三，ETF 基金、LOF 基金每笔交易的佣金费率是万分之八。(float)

min_commission：最低交易佣金，默认每笔交易最低扣 5 元佣金。(float)

type：交易类型，不传参默认为 STOCK(目前只支持 STOCK, ETF, LOF)。(string)

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)#将佣金费率设置为万分之三，将最低手续费设置为 3 元 set_commission(commi
```

```
ssion_ratio =0.0003, min_commission=3.0)def handle_data(context, data):pass
```

set_fixed_slippage-设置固定滑点

```
set_fixed_slippage(fixedslippage=0.0)
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置固定滑点，滑点在真实交易场景是不可避免的，因此回测中设置合理的滑点有利于让回测逼近真实场景。

注意事项：

无

参数

fixedslippage：固定滑点，委托价格与最后的成交价格的价差设置，这个价差是一个固定的值(比如 0.02 元，撮合成交时委托价格加减 0.01 元)。最终的成交价格=委托价格±fixedslippage(float)/2，默认固定滑点为 0.0。

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)# 将滑点设置为固定的 0.2 元，即原本买入交易的成交价为 10 元，则设置之后成交价将变成 10.1 元 set_fixed_slippage(fixedslippage=0.2)def handle_data(context, data):pass
```

set_slippage-设置滑点

```
set_slippage(slippage=0.1)
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置滑点比例，滑点在真实交易场景是不可避免的，因此回测中设置合理的滑点有利于让回测逼近真实场景。

注意事项：

无

参数

slippage：滑点比例，委托价格与最后的成交价格的价差设置，这个价差是当时价格的一个百分比(比如 0.2%，撮合成交时委托价格加减当时价格的 0.1%)。最终成交价格=委托价格±委托价格*slippage(float)/2，默认滑点比例为 0.1。

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)#将滑点影响比例设置为 0.2set_slippage(slippage = 0.2)def handle_data(context, data):pass
```

set_volume_ratio-设置成交比例

```
set_volume_ratio(volume_ratio=0.25)
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置回测中单笔委托的成交比例，使得盘口流动性方面的设置尽量逼近真实交易场景。

注意事项：

假如委托下单数量大于成交比例计算后的数量，系统会按成交比例计算后的数量撮合，差额部分委托数量不会继续挂单。

参数

volume_ratio：设置成交比例，默认 0.25，即指本周期最大成交数量为本周期市场可成交总量的四分之一(float)

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)#将最大成交数量设置为本周期可成交总量的二分之一 set_volume_ratio(volume_ratio = 0.5)def handle_data(context, data):pass
```

set_limit_mode-设置成交数量限制模式

```
set_limit_mode(limit_mode='LIMIT')
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置回测的成交数量限制模式。对于月度调仓等低频策略，对流动性冲击不是很敏感，不做成交量限制可以让回测更加便捷。

注意事项：

不做限制之后实际撮合成交量是可以大于该时间段的实际成交总量。

参数

limit_mode：设置成交数量限制模式，即指回测撮合交易时对成交数量是否做限制进行控制(str)

默认为限制，入参'LIMIT'，不做限制则入参'UNLIMITED'

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)#回测中不限制成交数量 set_limit_mode('UNLIMITED')def handle_data(con
```

```
text, data):pass
```

set_yesterday_position– 设置底仓

```
set_yesterday_position(poslist)
```

使用场景

该函数仅在回测模块可用

接口说明

该函数用于设置回测的初始底仓。

注意事项：

该函数会使策略初始化运行就创建出持仓对象，里面包含了设置的持仓信息。

参数

poslist：list 类型数据，该 list 中是字典类型的元素，参数不能为空(list[dict[str:str],...])；

数据格式及参数字段如下：

```
[{
'sid':标的代码,
'amount':持仓数量,
'enable_amount':可用数量,
'cost_basis':每股的持仓成本价格,
}]
```

参数也可通过 csv 文件的形式传入，参考接口 convert_position_from_csv

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)# 设置底仓 pos={}pos['sid'] = "600570.SS"pos['amount'] = "1000"pos['enable_amount'] = "600"pos['cost_basis'] = "55"set_yesterday_position([pos])def handle_data(context, data):#卖出 100 股 order(g.security,-100)
```

定时周期性函数

run_daily-按日周期处理

```
run_daily(context, func, time='9:31')
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该函数用于以日为单位周期性运行指定函数，可对运行触发时间进行指定。

注意事项：

1、该函数只能在初始化阶段 initialize 函数中调用。

2、该函数可以多次设定，以实现多个定时任务。

3、股票策略回测中，当回测周期为分钟时，time 的取值指定在 09:31~11:30 与 13:00~15:00 之间，当回测周期为日时，无论设定值是多少都只会在 15:00 执行；交易中不受此时间限制。

参数

context: Context 对象，存放有当前的账户及持仓信息(Context)；

func：自定义函数名称，此函数必须以 context 作为参数(Callable[[Context], None])；

time：指定周期运行具体触发运行时间点，默认为 9:31 分(str)，交易场景可设置范围：00:00~23:59。

返回

None

示例

```
# 定义一个财务数据获取函数，每天执行一次 def initialize(context):run_daily(context, get_finance)g.security = '600570.SS'set_universe(g.securi
ty)def get_finance(context):re = get_fundamentals(g.security,'balance_statement','total_assets')log.info(re)def handle_data(context, dat
a):pass
```

run_interval - 按设定周期处理

```
run_interval(context, func, seconds=10)
```

使用场景

该函数仅在交易模块可用

接口说明

该函数用于以设定时间间隔（单位为秒）周期性运行指定函数，可对运行触发时间间隔进行指定。

注意事项：

1、该函数只能在初始化阶段 initialize 函数中调用。

2、该函数可以多次设定，会以多个线程并行运行，但要小心不同线程之间的逻辑关联处理

3、seconds 设置最小时间间隔为 3 秒，小于 3 秒默认设定为 3 秒。

参数

context: Context 对象，存放有当前的账户及持仓信息(Context)；

func：自定义函数名称，此函数必须以 context 作为参数(Callable[[Context], None])；

seconds：设定时间间隔（单位为秒），取值为正整数(int)。

返回

None

示例

```
# 定义一个周期处理函数，每 10 秒执行一次 def initialize(context):run_interval(context, interval_handle, seconds = 10)g.security = '600570.SS's
```

```
et_universe(g.security)def interval_handle(context):snapshot = get_snapshot(g.security)log.info(snapshot)def handle_data(context, data):pa
ss
```

获取信息函数

获取基础信息

get_trading_day - 获取交易日期

```
get_trading_day(day)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该函数用于获取当前时间数天前或数天后的交易日期。

注意事项：

1、默认情况下，回测中当前时间为策略中调用该接口的回测日日期(context.blotter.current_dt)。

2、默认情况下，研究中当前时间为调用当天日期。

3、默认情况下，交易中当前时间为调用当天日期。

参数

day：表示天数，正的为数天后，负的为数天前，day 取 0 表示获取当前交易日，如果当前日期为非交易日则返回下一交易日的日期。day 默认取值为 0，不建议获取交易所还未公布的交易日期(int)；

返回

date：datetime.date 日期对象

示例

```
def initialize(context):g.security = ['600670.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):# 获取后一天的交易日期 previous_trading_date = get_trading_day(1)log.info(previous_trading_date)# 获取前一天的交易日期 next_trading_date = get_trading_day(-1)log.info(next_trading_date)
```

get_all_trades_days - 获取全部交易日期

```
get_all_trades_days(date=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该函数用于获取某个日期之前的所有交易日日期。

注意事项：

1、默认情况下，回测中 date 为策略中调用该接口的回测日日期(context.blotter.current_dt)。

2、默认情况下，研究中 date 为调用当天日期。

3、默认情况下，交易中 date 为调用当天日期。

参数

date：如'2016-02-13'或'20160213'

返回

一个包含所有交易日的 numpy.ndarray

示例

```
def initialize(context):# 获取当前回测日期之前的所有交易日 all_trades_days = get_all_trades_days()log.info(all_trades_days)all_trades_days_da
te = get_all_trades_days('20150312')log.info(all_trades_days_date)g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handl
e_data(context, data):pass
```

get_trade_days - 获取指定范围交易日期

```
get_trade_days(start_date=None, end_date=None, count=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该函数用于获取指定范围交易日期。

注意事项:

1、默认情况下，回测中 end_date 为策略中调用该接口的回测日日期(context.blotter.current_dt)。

2、默认情况下，研究中 end_date 为调用当天日期。

3、默认情况下，交易中 end_date 为调用当天日期。

参数

start_date：开始日期，与 count 二选一，不可同时使用。如'2016-02-13'或'20160213',开始日期最早不超过 1990 年(str)；

end_date：结束日期，如'2016-02-13'或'20160213'。如果输入的结束日期大于今年则至多返回截止到今年的数据(str)；

count：数量，与 start_date 二选一，不可同时使用，必须大于 0。表示获取 end_date 往前的 count 个交易日，包含 end_date 当天。count 建议不大于 3000，即返回数据的开始日期不早于 1990 年(int)；

返回

一个包含指定范围交易日的 numpy.ndarray

示例

```
def initialize(context):# 获取指定范围内交易日 trade_days = get_trade_days('2016-01-01', '2016-02-01')log.info(trade_days)g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):# 获取回测日期往前 10 天的所有交易日，包含历史回测日期 trading_days = get_trade_days(count=10)log.info(trading_days)
```

获取市场信息

get_market_list-获取市场列表

```
get_market_list()
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该函数用于返回当前市场列表目录。

注意事项：

回测和交易中仅限 before_trading_start 和 after_trading_end 中使用

参数

无

返回

返回 pandas.DataFrame 对象，返回字段包括:

finance_mic – 市场编码(str:str)

finance_name – 市场名称(str:str)

示例

```
get_market_list()
```

如返回：

	finance_mic	finance_name
0	A	美国证券交易所
1	CBJC	北京邮票
2	CBOT	芝加哥商品期货
3	CCFX	中国金融期货交易所
4	CCGG	中国国际文交所
5	CCJC	卡巴拉长江交易所
...	...	...
66	XFUND	基金
67	XHKG-SS	沪港通
68	XHKG-SZ	深港通

	finance_mic	finance_name
69	XSGE	上海期货交易所
70	XSHO	上海个股期权
71	XZCE	郑州商品交易所
72	YCME	渝川玉石

get_market_detail-获取市场详细信息

```
get_market_detail(finance_mic)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该函数用于返回市场编码对应的详细信息。

注意事项：

回测和交易中仅限 before_trading_start 和 after_trading_end 中使用

参数

finance_mic: 市场代码，相关市场编码参考 get_market_list 返回信息(str)。

返回

返回市场详细信息，类型为 pandas.DataFrame 对象，返回字段包括：

产品代码: prod_code(str:str)

产品名称: prod_name(str:str)

类型代码: hq_type_code(str:str)

时间规则: trade_time_rule(str:numpy.int64)

返回如下:

```
      hq_type_code prod_code prod_name  trade_time_rule
0              MRI    000001      上证指数               01              MRI    000002      A股指数               02              MRI    000003
    B股指数               03              MRI    000004      工业指数               04              MRI    000005      商业指数               05
            MRI    000006      地产指数               06              MRI    000007      公用指数               07              MRI    000008
  综合指数               0
```

示例

```
# 获取上海证券交易所相关信息 'XSHG'/'SS'get_market_detail('XSHG')
```

get_cb_list-获取可转债市场代码表

```
get_cb_list()
```

使用场景

该函数仅在交易模块可用

接口说明

返回当前可转债市场的所有代码列表(包含停牌代码)。

注意事项：

为减小对行情服务压力，该函数在交易模块中同一分钟内多次调用返回当前分钟首次查询的缓存数据。

参数

无

返回

返回当前可转债市场的所有代码列表(包含停牌代码)(list)。失败则返回空列表[]。

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)run_daily(context, get_trade_cb_list, "9:25")def before_trading_st
art(context, data):# 每日清空，避免取到昨日市场代码表 g.trade_cb_list = []def handle_data(context, data):pass# 获取当天可交易的可转债代码列表 def
 get_trade_cb_list(context):cb_list = get_cb_list()cb_snapshot = get_snapshot(cb_list)# 代码有行情快照并且交易状态不在暂停交易、停盘、长期停
盘、退市状态的判定为可交易代码 g.trade_cb_list = [cb_code for cb_code in cb_list ifcb_snapshot.get(cb_code, {}).get("trade_status") not in[No
ne, "HALT", "SUSP", "STOPT", "DELISTED"]]log.info("当天可交易的可转债代码列表为：%s" % g.trade_cb_list)
```

获取行情信息

get_history - 获取历史行情

```
get_history(count, frequency='1d', field='close', security_list=None, fq=None, include=False, fill='nan')
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取最近 N 条历史行情 K 线数据。支持多股票、多行情字段获取。

注意事项：

该接口只能获取 2005 年后的数据。

针对停牌场景，我们没有跳过停牌的日期，无论对单只股票还是多只股票进行调用，时间轴均为二级市场交易日日历，停牌时使用停牌前的数据填充，成交量为 0，日 K 线可使用成交量为 0 的逻辑进行停牌日过滤。

参数

count： K 线数量，大于 0，返回指定数量的 K 线行情；必填参数；入参类型：int；

frequency：K 线周期，现有支持 1 分钟线(1m)、5 分钟线(5m)、15 分钟线(15m)、30 分钟线(30m)、60 分钟线(60m)、120 分钟线(120m)、日线(1d)、周线(1w/weekly)、月线(mo/monthly)、季度线(1q/quarter)和年线(1y/yearly)频率的数据；选填参数，默认为'1d'；入参类型：str；

field：指明数据结果集中所支持输出的行情字段；选填参数，默认为['open','high','low','close','volume','money','price']；入参类型：list[str,str]或 str；输出字段包括：

- open -- 开盘价，字段返回类型：numpy.float64；
- high -- 最高价，字段返回类型：numpy.float64；

- low--最低价，字段返回类型：numpy.float64；
- close-- 收盘价，字段返回类型：numpy.float64；
- volume-- 交易量，字段返回类型：numpy.float64；
- money-- 交易金额，字段返回类型：numpy.float64；
- price-- 最新价，字段返回类型：numpy.float64；
- preclose-- 昨收盘价，字段返回类型：numpy.float64(仅日线返回)；
- high_limit-- 涨停价，字段返回类型：numpy.float64(仅日线返回)；
- low_limit-- 跌停价，字段返回类型：numpy.float64(仅日线返回)；
- unlimited-- 判断查询日是否是无涨跌停限制(1:该日无涨跌停限制;0:该日不是无涨跌停限制)，字段返回类型：numpy.float64(仅日线返回)；

security_list：要获取数据的股票列表；选填参数，None 表示在上下文中的 universe 中选中的所有股票；入参类型：list[str,str]或 str；

fq：数据复权选项，支持包括，pre-前复权，post-后复权，dypre-动态前复权，None-不复权；选填参数，默认为 None；入参类型：str；

include：是否包含当前周期，True–包含，False-不包含；选填参数，默认为 False；入参类型：bool；

fill：行情获取不到某一时刻的分钟数据时，是否用上一分钟的数据进行填充该时刻数据，'pre'–用上一分钟数据填充，'nan'–NaN 进行填充(仅交易有效)；选填参数，默认为'nan'；入参类型：str；

返回

第一种返回数据：

当获取单支股票（单只股票必须为字符串类型 security_list='600570.SS'，不能用 security_list=['600570.SS']）的时候，无论行情字段 field 入参单个或多个，返回的都是 pandas.DataFrame 对象，行索引是 datetime.datetime 对象，列索引是行情字段,为 str 类型。比如：

如果当前时间是 2017-04-18，get_history(5, '1d', 'open', '600570.SS', fq=None, include=False)将返回：

	open
2017-04-11	40.30
2017-04-12	40.08
2017-04-13	40.03
2017-04-14	40.04
2017-04-17	39.90

第二种返回数据：

当获取多支股票（多只股票必须为 list 类型，特殊情况：当 list 只有一个股票时仍然当做多股票处理，比如 security_list=['600570.SS']）的时候，如果行情字段 field 入参为单个，返回的是 pandas.DataFrame 对象，行索引是 datetime.datetime 对象，列索引是股票代码的编号,为 str 类型。比如：

如果当前时间是 2017-04-18，get_history(5,'1d','open',['600570.SS','600571.SS'],fq=None,include=False)将返回：

	600570.SS	600571.SS
2017-04-11	40.30	17.81
2017-04-12	40.08	17.56
2017-04-13	40.03	17.42
2017-04-14	40.04	17.40
2017-04-17	39.90	17.49

第三种返回数据：

当获取多支股票（多只股票必须为 list 类型，特殊情况：当 list 只有一个股票时仍然当做多股票处理，比如 security_list=['600570.SS']）的时候，如果行情字段 field 入参为多个，则返回 pandas.Panel 对象，items 索引是行情字段（如'open'、'close'等），里面是很多 pandas.DataFrame 对象，每个 pandas.DataFrame 的行索引是 datetime.datetime 对象， 列索引是股票代码,为 str 类型，比如:

如果当前时间是 2015-01-07，get_history(2,frequency='1d',field=['open','close'],security_list=['600570.SS','600571.SS'],fq=None,include=False)['open']将返回:

	600570.SS	600571.SS
2015-01-05	54.77	26.93
2015-01-06	51.00	25.83

假如要对 panel 索引中的对象进行转换，比如将 items 索引由行情字段转换成股票代码，可以通过 panel_info = panel_info.swapaxes("minor_axis", "items")的方法转换。

比如:

panel_info = get_history(2, frequency='1d', field=['open','close'], security_list=['600570.SS', '600571.SS'], fq=None, include=False)

按默认索引：df = panel_info['open']

对默认索引做转换：panel_info = panel_info.swapaxes("minor_axis", "items")

转换之后的索引：df = panel_info['600570.SS']

关于 numpy 和 pandas，请看下面的第三方库介绍。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def before_trading_start(context, data):# 获取农业版块过去 10 天的每日收盘价 industry_info = get_history(10, frequency="1d", field="close", security_list="A01000.XBHS")log.info(industry_info)def handle_data(context, data):# 股票池中全部股票过去 5 天的每日收盘价 his = get_history(5, '1d', 'close', security_list=g.security)log.info('股票池中全部股票过去 5 天的每日收盘价')log.info(his)# 获取 600570(恒生电子)过去 5 天的每天收盘价,# 一个 pd.Series 对象, index 是 datatimelog.info('获取
```

```
600570(恒生电子)过去 5 天的每天收盘价')log.info(his['600570.SS'])# 获取 600570(恒生电子)昨天(数组最后一项)的收盘价 log.info('获取 600570(恒生电子)昨天的收盘价')log.info(his['600570.SS'][-1])# 获取股票池中全部股票昨天的收盘价 log.info('获取股票池中全部股票昨天的收盘价')log.info(his.iloc[-1])# 获取 600570(恒生电子)昨天(数组最后一项)的收盘价 log.info('获取 600570(恒生电子)昨天的收盘价')log.info(his.iloc[-1]['600570.SS'])# 取得每一列的平均值 log.info('取得每一列的平均值')log.info(his.mean())# 获取股票池中全部股票的过去 10 分钟的成交量 his1 = get_history(10, '1m', 'volume')log.info('获取股票池中全部股票的过去 10 分钟的成交量')log.info(his1)# 获取恒生电子的过去 5 天的每天的收盘价 his2 = get_history(5, '1d', 'close', security_list='600570.SS')log.info('获取恒生电子的过去 5 天的每天的收盘价')log.info(his2)# 获取恒生电子的过去 5 天的每天的后复权收盘价 his3 = get_history(5, '1d', 'close', security_list='600570.SS', fq='post')log.info('获取恒生电子的过去 5 天的每天的后复权收盘价')log.info(his3)# 获取恒生电子的过去 5 周的每周的收盘价 his4 = get_history(5, '1w', 'close', security_list='600570.SS')log.info('获取恒生电子的过去 5 天的每天的收盘价')log.info(his4)# 获取多只股票的开盘价和收盘价数据 panel_info = get_history(2, frequency='1d', field=['open','close'], security_list=g.security)open_df = panel_info['open']log.info('获所有股票的取开盘价数据')log.info(open_df)df = open_df['600570.SS']log.info('仅获取恒生电子的开盘价数据')log.info(df)# panel 索引中的对象进行转换 panel_info2 = panel_info.swapaxes("minor_axis", "items")df = panel_info2['600570.SS']log.info('仅获取恒生电子的开盘价和收盘价数据')log.info(df)open_df = df['open']log.info('获取恒生电子的开盘价数据')log.info(open_df)
```

get_price - 获取历史数据

```
get_price(security, start_date=None, end_date=None, frequency='1d', fields=None, fq=None, count=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取指定日期的前 N 条的历史行情 K 线数据或者指定时间段内的历史行情 K 线数据。支持多股票、多行情字段获取。

注意事项:

1、start_date 与 count 必须且只能选择输入一个，不能同时输入或者同时都不输入。

2、针对停牌场景，我们没有跳过停牌的日期，无论对单只股票还是多只股票进行调用，时间轴均为二级市场交易日日历，停牌时使用停牌前的数据填充，成交量为 0，日 K 线可使用成交量为 0 的逻辑进行停牌日过滤。

3、数据返回内容不包括当天数据。

4、count 只针对'daily', 'weekly', 'monthly', 'quarter', 'yearly', '1d', '1m', '5m', '15m', '30m', '60m', '120m', '1w', 'mo', '1q', '1y'频率有效，并且输入日期的类型需与频率对应。

5、'weekly', '1w', 'monthly', 'mo', 'quarter', '1q', 'yearly', '1y'频率不支持 start_date 和 end_date 组合的入参，只支持 end_date 和 count 组合的入参形式。

6、返回的周线数据是由日线数据进行合成。

7、该接口只能获取 2005 年后的数据。

参数

security：一支股票代码或者一个股票代码的 list(list[str]/str)

start_date：开始时间，默认为空。传入格式仅支持：YYYYmmdd、YYYY-mm-dd、YYYY-mm-dd HH:MM、YYYYmmddHHMM，如'20150601'、'2015-06-01'、'2015-06-01 10:00'、'201506011000'(str)；

end_date：结束时间，默认为空，传入格式仅支持：YYYYmmdd、YYYY-mm-dd、YYYY-mm-dd HH:MM、YYYYmmddHHMM，如'20150601'、'2015-06-01'、'2015-06-01 14:00'、'201506011400'(str)；

frequency： 单位时间长度，现有支持 1 分钟线(1m)、5 分钟线(5m)、15 分钟线(15m)、30 分钟线(30m)、60 分钟线(60m)、120 分钟线(120m)、日线(1d)、周线(1w/weekly)、月线(mo/monthly)、季度线(1q/quarter)和年线(1y/yearly)频率数据(str)；

fields：指明数据结果集中所支持输出字段(list[str]/str)，输出字段包括 ：

- open -- 开盘价(str:numpy.float64)；
- high -- 最高价(str:numpy.float64)；
- low --最低价(str:numpy.float64)；
- close -- 收盘价(str:numpy.float64)；
- volume -- 交易量(str:numpy.float64)；
- money -- 交易金额(str:numpy.float64)；
- price -- 最新价(str:numpy.float64)；

- preclose -- 昨收盘价(str:numpy.float64)(仅日线返回)；
- high_limit -- 涨停价(str:numpy.float64)(仅日线返回)；
- low_limit -- 跌停价(str:numpy.float64)(仅日线返回)；
- unlimited -- 判断查询日是否无涨跌停限制(1：该日无涨跌停限制；0：该日有涨跌停限制)(str:numpy.float64)(仅日线返回)；

fq：数据复权选项，支持包括，pre-前复权，post-后复权，None-不复权(str)；

count：大于 0，不能与 start_date 同时输入，获取 end_date 前 count 根的数据，不支持除天（'daily'/'1d'）、分钟（'1m'）、5 分钟线('5m')、15 分钟线('15m')、30 分钟线('30m')、60 分钟线('60m')、120 分钟线('120m')、周（'weekly'/'1w'）、（'monthly'/'mo'）、（'quarter'/'1q'）和（'yearly'/'1y'）以外的其它频率(int)；

返回

get_price 对于多股票和多字段不同场景下获取返回数据的规则与 get_history 一致，如下：

第一种返回数据：

当获取单支股票（单只股票必须为字符串类型 security='600570.SS'，不能用 security=['600570.SS']）和单个或多个字段的时候，返回的是 pandas.DataFrame 对象，行索引是 datetime.datetime 对象，列索引是行情字段，为 str 类型。

例如，输入为 get_price(security='600570.SS',start_date='20170201',end_date='20170213',frequency='1d')时，将返回：

```
openclose high low volume       price   money     preclose    high_limit    low_limit    unlimited
2017-02-0344.4743.9044.5043.584418325.043.90193895820.044.26   48.6939.83    02017-02-0643.9144.1044.3043.664428487.044.10194979290.043.90
   48.2939.51      02017-02-0744.0543.5244.0743.345649251.043.52246776480.044.10   48.5139.69      02017-02-0843.5944.5944.7843.531257023
3.044.59557883600.043.52   47.8739.17     02017-02-0944.7444.7445.2844.399240223.044.74413875390.044.59   49.0540.13      02017-02-1044.8
044.6244.9844.418097465.044.62361757300.044.74   49.2140.27      02017-02-1344.3244.8945.9844.0214931596.044.89672360490.044.62   49.0840.
16      0
```

第二种返回数据：

当获取多支股票（多只股票必须为 list 类型，特殊情况：当 list 只有一个股票时仍然当做多股票处理，比如 security=['600570.SS']）和单个字段的时候，返回的是 pandas.DataFrame 对象，行索引是 datetime.datetime 对象，列索引是股票代码的编号，为 str 类型。

例如，输入为 get_price(['600570.SS'], start_date='20170201', end_date='20170213', frequency='1d', fields='open')时，将返回：

```
600570.SS2017-02-03     44.472017-02-06     43.912017-02-07     44.052017-02-08     43.592017-02-09     44.742017-02-10     44.802017-0
2-13     44.32
```

第三种返回数据：

如果是获取多支股票（多只股票必须为 list 类型，特殊情况：当 list 只有一个股票时仍然当做多股票处理，比如 security=['600570.SS']）和多个字段，则返回 pandas.Panel 对象，items 索引是行情字段，为 str 类型（如'open'、'close'等），里面是很多 pandas.DataFrame 对象，每个 pandas.DataFrame 的行索引是 datetime.datetime 对象， 列索引是股票代码，为 str 类型。

例如，输入为 get_price(['600570.SS','600571.SS'], start_date='20170201', end_date='20170213', frequency='1d', fields=['open','close'])['open']时，将返回：

```
600570.SS   600571.SS2017-02-03    44.47        19.362017-02-06    43.91        19.002017-02-07    44.05        19.272017-02-08    43.59
  19.102017-02-09    44.74        19.472017-02-10    44.80        19.572017-02-13    44.32        19.22
```

假如要对 panel 索引中的对象进行转换，比如将 items 索引由行情字段转换成股票代码，可以通过 panel_info = panel_info.swapaxes("minor_axis", "items")的方法转换。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获得 600570.SS(恒生电子)的 2015 年 01 月的天数据，只获取 open 字段 price_open = get_price('600570.SS', start_date='20150101', end_date='20150131', frequency='1d')['open']log.info(price_open)# 获取指定结束日期前 count 天到结束日期的所有开盘数据# price_open = get_price('600570.SS', end_date='20150131', frequency='daily', count=10)['open']# log.info(price_open)# 获取股票指定结束时间前 count 分钟到指定结束时间的所有数据# stock_info = get_price('600570.SS', end_date='2015-01-31 10:00', frequency='1m', count=10)# log.info(stock_info)# 获取指定结束日期前 count 周到结束日期所在周的所有开盘数据# week_open = get_price('600570.SS', end_date='20150131', frequency='1w', count=10)['open']# log.info(week_open)# 获取多只股票# 获取沪深 300 的 2015 年 1 月的天数据，返回一个[pandas.Panel]security_list = get_index_stocks('000300.XBHS', '20150101')price = get_price(security_list, start_date='20150101', end_date='20150131')log.info(price)# 获取某股票开盘价，行索引是[datetime.datetime]对象，列索引是行情字段 price_open = price['open'][security_list[0]]log.info(price_open)# 获取农业版块指定结束日期前 count 天到结束日期的数据 industry_info = get_price("A01000.XBHS", end_date="20210315", frequency="daily", count=10)log.info(industry_info)
```

get_individual_entrust– 获取逐笔委托行情

```
get_individual_entrust(stocks=None, data_count=50, start_pos=0, search_direction=1)
```

使用场景

该函数在交易模块可用

接口说明

该接口用于获取当日逐笔委托行情数据。

注意事项：

1、沪深市场都有逐笔委托数据；

2、逐笔委托，逐笔成交数据需开通 level2 行情才有数据推送，否则无数据返回；

参数

stocks: 默认为当前股票池中代码列表(list[str])；

data_count: 数据条数，默认为 50，最大为 200(int)；

start_pos: 起始位置，默认为 0(int)；

search_direction: 搜索方向（1 向前，2 向后），默认为 1(int)；

返回

Pandas.panel 对象

Items axis: 股票代码列表(str);

Major_axis axis: 数据索引为自然数列(DataFrame);

Minor_axis axis: 包含以下信息:

- business_time: 时间戳毫秒级(str:numpy.int64);
- hq_px: 价格(str:numpy.int64);
- business_amount: 委托数量(str:numpy.int64);
- order_no: 委托编号(str:numpy.int64);
- business_direction: 委托方向，0:卖，1:买，2:借入，3:出借(str:numpy.int64);
- trans_kind: 委托类别，深圳市场(1:市价委托，2:限价委托，3:本方最优)，上海市场(4:增加订单 5:删除订单)(str:numpy.int64);

示例

```
def initialize(context):g.security = '000001.SZ'set_universe(g.security)def handle_data(context, data):#获取当前股票池逐笔委托数据 entrust =
get_individual_entrust()log.info(entrust)#获取指定股票列表逐笔委托数据 entrust = get_individual_entrust(['000002.SZ','000032.SZ'])log.info(en
```

```
trust)#获取委托量 business_amount = entrust['000002.SZ']['business_amount']log.info('逐笔数据的委托量为：%s' % business_amount)
```

get_individual_transcation– 获取逐笔成交行情

```
get_individual_transcation(stocks=None, data_count=50, start_pos=0, search_direction=1)
```

使用场景

该函数在交易模块可用

接口说明

该接口用于获取当日逐笔成交行情数据。

注意事项：

1、沪深市场都有逐笔成交数据；

2、逐笔委托，逐笔成交数据需开通 level2 行情才有数据推送，否则无数据返回；

参数

stocks: 默认为当前股票池中代码列表(list[str])；

data_count: 数据条数，默认为 50，最大为 200(int)；

start_pos: 起始位置，默认为 0(int)；

search_direction: 搜索方向（1 向前，2 向后），默认为 1(int)；

返回

Pandas.panel 对象

Items axis: 股票代码列表(str)；

Major_axis axis: 数据索引为自然数列(DataFrame)；

Minor_axis axis: 包含以下信息：

- business_time: 时间戳毫秒级(str:numpy.int64)；
- hq_px: 价格(str:numpy.float64)；
- business_amount: 成交数量(str:numpy.int64)；
- trade_index: 成交编号(str:numpy.int64)；
- business_direction: 成交方向，0:卖，1:买，2:借入，3:出借(str:numpy.int64)；

- buy_no: 叫买方编号(str:numpy.int64)；
- sell_no: 叫卖方编号(str:numpy.int64)；
- trans_flag: 成交标记（0：普通成交，1：撤单成交）(str:numpy.int64)；
- trans_identify_am: 盘后逐笔成交序号标识(0 表示盘中，1 表示盘后)(str:numpy.int64)；
- channel_num: 成交通道信息(str:numpy.int64)；

示例

```
def initialize(context):g.security = '000001.SZ'set_universe(g.security)def handle_data(context, data):#获取当前股票池逐笔成交数据 transcatio
n = get_individual_transcation()log.info(transcation)#获取指定股票列表逐笔成交数据 transcation = get_individual_transcation(['000002.SZ','000
032.SZ'])log.info(transcation)#获取成交量 business_amount = transcation['000002.SZ']['business_amount']log.info('逐笔数据的成交量为：%s' % bu
siness_amount)
```

get_tick_direction– 获取分时成交行情

```
get_tick_direction(symbols=None, query_date=0, start_pos=0, search_direction=1, data_count=50)
```

使用场景

该函数在交易模块可用

接口说明

该接口用于获取当日分时成交行情数据。

注意事项：

1、沪深市场都有分时成交数据；

2、分时成交数据需开通 level2 行情才有数据推送，否则无数据返回；

参数

symbols: 默认为当前股票池中代码列表(list[str])；

query_date: 查询日期，默认为 0，返回当日日期数据(目前行情只支持查询当日的数据，格式为 YYYYMMDD)(int)；

start_pos: 起始位置，默认为 0(int)；

search_direction: 搜索方向（1 向前，2 向后），默认为 1(int)；

data_count: 数据条数，默认为 50，最大为 200(int)；

返回

返回一个 OrderedDict 对象，包含每只代码的分时成交行情数据。(OrderedDict([(),()...]))

返回结果字段介绍：

- time_stamp: 时间戳毫秒级(str:numpy.int64)；
- hq_px: 价格(str:numpy.float64)；
- hq_px64: 价格(str:numpy.int64)(行情暂不支持，返回均为 0)；
- business_amount: 成交数量(str:numpy.int64)；
- business_balance: 成交金额(str:numpy.int64)；
- business_count: 成交笔数(str:numpy.int64)；
- business_direction: 成交方向（0：卖，1：买，2：平盘)(str:numpy.int64)；
- amount: 持仓量(str:numpy.int64)(行情暂不支持，返回均为 0)；
- start_index: 分笔关联的逐笔开始序号(str:numpy.int64)(行情暂不支持，返回均为 0)；
- end_index: 分笔关联的逐笔结束序号(str:numpy.int64)(行情暂不支持，返回均为 0)；

示例

```
def initialize(context):g.security = '000001.SZ'set_universe(g.security)def handle_data(context, data):#获取 000001.SZ 的分时成交数据 directi
on_data = get_tick_direction(g.security)log.info(direction_data)#获取指定股票列表分时成交数据 direction_data = get_tick_direction(['000002.SZ
','000032.SZ'])log.info(direction_data)#获取成交量 business_amount = direction_data['000002.SZ']['business_amount']log.info('分时成交的成交
量为：%s' % business_amount)
```

get_sort_msg – 获取板块、行业的涨幅排名

```
get_sort_msg(sort_type_grp=None, sort_field_name=None, sort_type=1, data_count=100)
```

使用场景

该函数在交易模块可用

接口说明

该接口用于获取板块、行业的涨幅排名。

参数

sort_type_grp: 板块或行业的代码(list[str]/str)；(暂时只支持 XBHS.DY 地域、XBHS.GN 概念、XBHS.ZJHHY 证监会行业、XBHS.ZS 指数、XBHS.HY 行业等)

sort_field_name: 需要排序的字段(str)；该字段支持输入的参数如下：

- preclose_px: 昨日收盘价；
- open_px: 今日开盘价；
- last_px: 最新价；

- high_px: 最高价；
- low_px: 最低价；
- wavg_px: 加权平均价；
- business_amount: 总成交量；
- business_balance: 总成交额；
- px_change: 涨跌额；
- amplitude: 振幅；
- px_change_rate: 涨跌幅；
- circulation_amount: 流通股本；
- total_shares: 总股本；
- market_value: 市值；
- circulation_value: 流通市值；
- vol_ratio: 量比；
- rise_count: 上涨家数；
- fall_count: 下跌家数；

sort_type: 排序方式，默认降序(0:升序，1:降序)(int)；

data_count: 数据条数，默认为 100，最大为 10000(int)；

返回

正常返回一个 List 列表，里面包含板块、行业代码的涨幅排名信息(list[dict{str:str,...},...])，

返回每个代码的信息包含以下字段内容：

- prod_code: 行业代码(str:str)；
- prod_name: 行业名称(str:str)；
- hq_type_code: 行业板块代码(str:str)；
- time_stamp: 时间戳毫秒级(str:int)；
- trade_mins: 交易分钟数(str:int)；
- trade_status: 交易状态(str:str)；
- preclose_px: 昨日收盘价(str:float)；
- open_px: 今日开盘价(str:float)；
- last_px: 最新价(str:float)；

- high_px: 最高价(str:float)；
- low_px: 最低价(str:float)；
- wavg_px: 加权平均价(str:float)；
- business_amount: 总成交量(str:int)；
- business_balance: 总成交额(str:int)；
- px_change: 涨跌额(str:float)；
- amplitude: 振幅(str:int)；
- px_change_rate: 涨跌幅(str:float)；
- circulation_amount: 流通股本(str:int)；
- total_shares: 总股本(str:int)；
- market_value: 市值(str:int)；
- circulation_value: 流通市值(str:int)；
- vol_ratio: 量比(str:float)；
- shares_per_hand: 每手股数(str:int)；
- rise_count: 上涨家数(str:int)；
- fall_count: 下跌家数(str:int)；

- member_count: 成员个数(str:int)；
- rise_first_grp: 领涨股票(其包含以下五个字段)(str:list[dict{str:int,str:str,str:str,str:float,str:float},...])；
    - prod_code: 股票代码(str:str)；
    - prod_name: 证券名称(str:str)；
    - hq_type_code: 类型代码(str:str)；
    - last_px: 最新价(str:float)；
    - px_change_rate: 涨跌幅(str:float)；
- fall_first_grp: 领跌股票(其包含以下五个字段)(str:list[dict{str:int,str:str,str:str,str:float,str:float},...])；
    - prod_code: 股票代码(str:str)；
    - prod_name: 证券名称(str:str)；
    - hq_type_code: 类型代码(str:str)；
    - last_px: 最新价(str:float)；
    - px_change_rate: 涨跌幅(str:float)；

示例

```
def initialize(context):g.security = '000001.SZ'set_universe(g.security)def handle_data(context, data):#获取 XBHS.DY 板块的涨幅排名信息 sort_data = get_sort_msg(sort_type_grp='XBHS.DY', sort_field_name='preclose_px', sort_type=1, data_count=100)log.info(sort_data)#获取 sort_data 排序第一条代码的数据 sort_data_first = sort_data[0]log.info(sort_data_first)
```

get_etf_info - 获取 ETF 信息

```
get_etf_info(etf_code)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于获取单支或者多支 ETF 的信息。

注意事项：

无

参数

etf_code：单支 ETF 代码或者一个 ETF 代码的 list，必传参数(list[str]/str)

返回

正常返回一个 dict 类型字段，包含每只 ETF 信息，key 为 ETF 代码，values 为包含 etf 信息的 dict。异常返回空 dict，如{}(dict[str:dict[...]])

返回结果字段介绍：

- etf_redemption_code -- 申赎代码(str:str)；
- publish -- 是否需要发布 IOPV(str:int)；
- report_unit -- 最小申购、赎回单位(str:int)；
- cash_balance -- 现金差额(str:float)；
- max_cash_ratio -- 现金替代比例上限(str:float)；
- pre_cash_componet -- T-1 日申购基准单位现金余额(str:float)；
- nav_percu -- T-1 日申购基准单位净值(str:float)；
- nav_pre -- T-1 日基金单位净值(str:float)；
- allot_max -- 申购上限(str:float)；
- redeem_max -- 赎回上限(str:float)；

字段备注:

- publish -- 是否需要发布 IOPV，1 是需要发布，0 是不需要发布；

返回如下:

```
{'510020.SS': {'nav_percu': 206601.39, 'redeem_max': 0.0, 'nav_pre': 0.207, 'report_unit': 1000000, 'max_cash_ratio': 0.4,'cash_balance':
 -813.75, 'etf_redemption_code': '510021', 'pre_cash_componet': 598.39, 'allot_max': 0.0, 'publish': 1}}
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#ETF 信息 etf_info = get_etf_info('5
10020.SS')log.info(etf_info)etfs_info = get_etf_info(['510020.SS','510050.SS'])log.info(etfs_info)
```

get_etf_stock_info - 获取 ETF 成分券信息

```
get_etf_stock_info(etf_code,security)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于获取 ETF 成分券信息。

注意事项:

无

参数

etf_code:单支 ETF 代码，必传参数(str)

security：单只股票代码或者一个由多只股票代码组成的列表，必传参数(list[str]/str)

返回

正常返回一个 dict 类型字段，包含每只 etf 代码中成分股的信息。异常返回空 dict，如{}(dict[str:dict[...]])

返回结果字段介绍：

- code_num -- 成分券数量(str:float)；
- cash_replace_flag -- 现金替代标志(str:str)；
    - '0' -- 禁止替代；
    - '1' -- 允许替代；
    - '2' -- 必须替代；
    - '3' -- 非沪市退补现金替代；

- '4' -- 非沪市必须现金替代;
- '5' -- 非沪深退补现金替代;
- '6' -- 非沪深必须现金替代;

- replace_ratio -- 保证金率（溢价比率），允许现金替代标的此字段有效(str:float);
- replace_balance -- 替代金额,必须现金替代标的此字段有效(str:float);
- is_open -- 停牌标志，0-停牌，1-非停牌(str:int);

返回如下:

```
{'600000.SS': {'cash_replace_flag': '1', 'replace_ratio': 0.1, 'is_open': 1, 'code_num': 4700.0, 'replace_balance': 0.0}}
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#ETF 成分券信息 stock_info = get_etf_stock_info('510050.SS','600000.SS')log.info(stock_info)stocks_info = get_etf_stock_info('510050.SS',['600000.SS','600036.SS'])log.info(stocks_info)
```

get_gear_price– 获取指定代码的档位行情价格

```
get_gear_price(sids)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于获取指定代码的档位行情价格。

注意事项：

获取实时行情快照失败时返回档位内容为空 dict({"bid_grp":{}, "offer_grp":{}})

若无 L2 行情时，委托笔数字段返回 0。

参数

sids：股票代码(list[str]/str)；

返回

包含以下信息(dict[str:dict[int:list[float,int,int],...],...])：

- bid_grp:委买档位(str:dict[int:list[float,int,int],...])；
- offer_grp:委卖档位(str:dict[int:list[float,int,int],...])；

```
单只代码返回：{'bid_grp': {1: [价格, 委托量,委托笔数], 2: [价格, 委托量,委托笔数], 3: [价格, 委托量,委托笔数], 4: [价格, 委托量,委托笔数], 5: [价格, 委托量,委托笔数]},'offer_grp': {1: [价格, 委托量,委托笔数], 2: [价格, 委托量,委托笔数], 3: [价格, 委托量,委托笔数], 4: [价格, 委托量,委托笔数], 5: [价格, 委托量,委托笔数]}}多只代码返回：{代码：{'bid_grp': {1: [价格, 委托量,委托笔数], 2: [价格, 委托量,委托笔数], 3: [价格, 委托量,委托笔数], 4: [价格, 委托量,委托笔数], 5: [价格, 委托量,委托笔数]},'offer_grp': {1: [价格, 委托量,委托笔数], 2: [价格, 委托量,委托笔数], 3: [价格, 委托量,委托笔数], 4: [价格, 委托量,委托笔数], 5: [价格, 委托量,委托笔数]}}}
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#获取 600570.SS 当前档位行情 gear_price = get_gear_price('600570.SS')log.info(gear_price)#获取 600571.SS 当前档位行情 gear_price = get_gear_price('600571.SS')log.info(gear_price)
```

get_snapshot - 取行情快照

```
get_snapshot(security)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于获取实时行情快照。

注意事项：

无

参数

security： 单只股票代码或者多只股票代码组成的列表，必填字段(list[str]/str)；

返回

正常返回一个 dict 类型数据，包含每只股票代码的行情快照信息。异常返回空 dict，如{}(dict[str:dict[...]])

快照包含以下信息：

- prod_code:证券代码(str:dict)；
- bid_grp:委买档位(第一档包含委托队列（仅 L2 支持）)(str:dict[int:list[float,int,int,{int:int,...}],int:list[float,int,int]...])；
- offer_grp:委卖档位(第一档包含委托队列（仅 L2 支持）)(str:dict[int:list[float,int,int,{int:int,...}],int:list[float,int,int]...])；
- open_px:今开盘价(str:float)；
- high_px:最高价(str:float)；
- low_px:最低价(str:float)；
- last_px:最新成交价(str:float)；
- preclose_px:昨收价(str:float)；
- business_balance:总成交额(str:float)；

- business_amount:总成交量(str:int)；
- amount:持仓量(str:int)；
- prev_settlement:昨结算(str:float)；
- turnover_ratio:换手率(str:int)；
- trade_status:交易状态(str:str)；
- up_px:涨停价格(str:float)；
- down_px:跌停价格(str:float)；
- entrust_rate:委比(str:float)；
- vol_ratio:量比(str:float)；
- entrust_diff:委差(str:float)；
- pe_rate:动态市盈率(str:float)；
- pb_rate:市净率(str:float)；
- circulation_amount:流通股本(str:int)；
- wavg_px:加权平均价(str:float)；
- px_change_rate:涨跌幅(str:float)；
- issue_date:上市日期(str:int)；

- hsTimeStamp:时间戳(str:float)；
- total_bidqty:委买量(str:int)；
- total_offerqty:委卖量(str:int)；
- total_bid_turnover:委买金额(str:int)；
- total_offer_turnover:委卖金额(str:int)

字段备注:

- bid_grp -- 委买档位，{'bid_grp':{1:[价格，委托量,委托笔数,委托对列{}],2:[价格，委托量,委托笔数],3:[价格，委托量,委托笔数],4:[价格,委托量,委托笔数],5:[价格，委托量,委托笔数]}}；
- offer_grp -- 委卖档位，{'offer_grp':{1:[价格，委托量,委托笔数,委托对列{}],2:[价格，委托量,委托笔数],3:[价格，委托量,委托笔数],4:[价格，委托量,委托笔数],5:[价格，委托量,委托笔数]}}；
- total_bid_turnover/total_offer_turnover,委买金额/委卖金额主推数据(tick 数据中)不支持(值为 0)，仅在线请求中支持；
- trade_status -- 交易状态；
    - START -- 市场启动(初始化之后，集合竞价前)
    - PRETR -- 盘前
    - OCALL -- 开始集合竞价

- TRADE -- 交易(连续撮合)
- HALT -- 暂停交易
- SUSP -- 停盘
- BREAK -- 休市
- POSTR -- 盘后
- ENDTR -- 交易结束
- STOPT -- 长期停盘，停盘 n 天，n>=1
- DELISTED -- 退市
- POSMT -- 盘后交易
- PCALL -- 盘后集合竞价
- INIT -- 盘后固定价格启动前
- ENDPT -- 盘后固定价格闭市阶段
- POSSP -- 盘后固定价格停牌

返回如下:


```
{'600570.SS': {'amount': 0, 'vol_ratio': 0.69, 'open_px': 41.2, 'up_px': 45.94,'turnover_ratio': 0.0057, 'wavg_px': 41.8, 'entrust_diff':
 -117.2, 'low_px': 41.01, 'circulation_amount': 1461560480,'business_balance': 351217087.0, 'px_change_rate': 0.31, 'preclose_px': 41.76,
```

'prev_settlement': 0.0, 'business_amount': 8402391,'pb_rate': 10.76, 'last_px': 41.89, 'offer_grp': {1: [41.89, 4400, 0, {}], 2: [41.9, 13020, 0], 3: [41.91, 2300, 0], 4: [41.92, 600, 0],5: [41.93, 400, 0]}, 'hsTimeStamp': 20220617132109340, 'total_bidqty': 9000, 'trade_status': 'TRADE', 'down_px': 37.58,'bid_grp': {1: [41.88, 1200, 0, {}], 2: [41.87, 2300, 0], 3: [41.85, 200, 0], 4: [41.84, 500, 0], 5: [41.83, 4800, 0]}, 'high_px': 42.29,'issue_date': 0, 'pe_rate': 4294596.65, 'entrust_rate': -0.3943, 'total_bid_turnover': 0, 'total_offer_turnover': 0, 'total_offerqty': 20720}}

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 行情快照 snapshot = get_snapshot(g.security)log.info(snapshot)
```

获取股票信息

get_stock_name - 获取股票名称

```
get_stock_name(stocks)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口可获取股票、可转债、ETF 等名称。

注意事项：

无

参数

stocks：股票代码(list[str]/str)；

返回

股票名称字典，dict 类型，key 为股票代码，value 为股票名称，当没有查询到相关数据或者输入有误时 value 为 None(dict[str:str])；

```
{'600570.SS': '恒生电子'}
```

示例

```
def initialize(context):g.security = ['600570.SS', '600571.SS']set_universe(g.security)def handle_data(context, data):#获取 600570.SS 股票名称 stock_name = get_stock_name(g.security[0])log.info(stock_name)#获取股票池所有的股票名称 stock_names = get_stock_name(g.security)log.info(stock_names)
```

get_stock_info - 获取股票基础信息

```
get_stock_info(stocks, field=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口可获取股票、可转债、ETF 等基础信息。

注意事项：

field 不做入参时默认只返回 stock_name 字段

参数

stocks：股票代码(list[str]/str)；

field：指明数据结果集中所支持输出字段(list[str]/str)，输出字段包括 ：

- stock_name -- 股票代码对应公司名(str:str)；
- listed_date -- 股票上市日期(str:str)；
- de_listed_date -- 股票退市日期，若未退市，返回 2900-01-01(str:str)；

返回

嵌套 dict 类型，包含内容为 field 中指定内容，若 field=None，返回股票基础信息仅包含对应公司名(dict[str:dict[str:str,...],...])

```
{'600570.SS': {'stock_name': '恒生电子', 'listed_date': '2003-12-16', 'de_listed_date': '2900-01-01'}}
```

示例

```
def initialize(context):g.security = ['600570.SS', '600571.SS']set_universe(g.security)def handle_data(context, data):#获取单支股票的基础信息 stock_info = get_stock_info(g.security[0])log.info(stock_info)#获取多支股票的基础信息 stock_infos = get_stock_info(g.security, ['stock_name','listed_date','de_listed_date'])log.info(stock_infos)
```

get_stock_status– 获取股票状态信息

```
get_stock_status(stocks, query_type='ST', query_date=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取指定日期股票的 ST、停牌、退市属性。

注意事项：

无

参数

stocks: 例如 ['000001.SZ','000003.SZ']。该字段必须输入，否则返回 None(list[str]/str)；

query_type: 支持以下三种类型属性的查询，默认为'ST'(str)；

具体支持输入的字段包括 ：

- 'ST' – 查询是否属于 ST 股票
- 'HALT' – 查询是否停牌
- 'DELISTING' – 查询是否退市

query_date: 格式为 YYYYmmdd，默认为 None,表示当前日期（回测为回测当前周期，研究与交易则取系统当前时间）(str)；

返回

返回 dict 类型，每支股票对应的值为 True 或 False，当没有查询到相关数据或者输入有误时返回 None(dict[str:bool,...])；

```
{'600570': None}
```

示例

```
def initialize(context):g.security = ['600397.SS', '600701.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):stocks_
```

```
list = g.security
filter_stocks = []# 判断股票是否为 ST、停牌或者退市的股票 st_status = get_stock_status(stocks_list, 'ST')# 将不是 ST 的股票筛选出来 for i in stock
s_list:if st_status[i] is not True:filter_stocks.append(i)# 获取股票停牌信息# halt_status = get_stock_status(stocks_list, 'HALT')# 获取指定
日期的对应属性# halt_status = get_stock_status(stocks_list, 'HALT', '20180312')# 获取股票退市信息# delist_status = get_stock_status(stocks_li
st, 'DELISTING')log.info('筛选不是 ST 的股票列表: %s' % filter_stocks)
```

get_stock_exrights - 获取股票除权除息信息

```
get_stock_exrights(stock_code, date=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取股票除权除息信息。

注意事项:

无

参数

stock_code;str 类型, 股票代码(str);

date: 查询该日期的除权除息信息，默认获取该股票历史上所有除权除息信息，e.g.

'20180228'/20180228/datetime.date(2018,2,28)(str/int/datetime.date)

返回

输入日期若没有除权除息信息则返回 None,有相关数据则返回 pandas.DataFrame 类型数据

例如输入 get_stock_exrights('600570.SS')，返回

```
allotted_ps   rationed_ps   rationed_px   bonus_ps   exer_forward_a   exer_forward_b   exer_backward_a   exer_backward_b
date
20040604  0.0         0.0          0.0          0.43      0.046077        -1.433           1.000000        0.43020050601  0.5         0.0
          0.0          0.20      0.046077        -1.413           1.500000        0.63020050809  0.4         0.0          0.0          0.
00      0.069115        -1.404           2.100000        0.63020060601  0.4         0.0          0.0          0.11      0.096762
 -1.404           2.940000        0.86120070423  0.3         0.0          0.0          0.10      0.135466        -1.394           3.82200
0        1.15520080528  0.6         0.0          0.0          0.07      0.176106        -1.380           6.115200        1.42220090423
0.5         0.0          0.0          0.10      0.281770        -1.368           9.172799        2.03420100510  0.4         0.0
0.0          0.05      0.422654        -1.340           12.841919       2.49220110517  0.0         0.0          0.0          0.05
0.591716        -1.318           12.841919       3.13420120618  0.0         0.0          0.0          0.08      0.591716        -1.289
        12.841919       4.16220130514  0.0         0.0          0.0          0.10      0.591716        -1.242           12.841919
5.44620140523  0.0         0.0          0.0          0.16      0.591716        -1.182           12.841919       7.50120150529  0.0
  0.0          0.0          0.18      0.591716        -1.088           12.841919       9.81220160530  0.0         0.0          0.0
   0.26      0.591716        -0.981           12.841919       13.15120170510  0.0         0.0          0.0          0.10      0.591716
     -0.827           12.841919       14.43520180524  0.0         0.0          0.0          0.29      0.591716        -0.768           1
2.841919       18.15920190515  0.3         0.0          0.0          0.32      0.591716        -0.597           16.694494       22.26920
200605  0.3         0.0          0.0          0.53      0.769231        -0.407           21.702843       31.117
```

返回结果字段介绍：

- date -- 日期(索引列，类型为 int64)；
- allotted_ps -- 每股送股(str:numpy.float64)；
- rationed_ps -- 每股配股(str:numpy.float64)；
- rationed_px -- 配股价(str:numpy.float64)；
- bonus_ps -- 每股分红(str:numpy.float64)；
- exer_forward_a -- 前复权除权因子 A；用于计算前复权价格(前复权价格=A*价格+B)(str:numpy.float64)
- exer_forward_b -- 前复权除权因子 B；用于计算前复权价格(前复权价格=A*价格+B)(str:numpy.float64)
- exer_backward_a -- 后复权除权因子 A；用于计算后复权价格(后复权价格=A*价格+B)(str:numpy.float64)
- exer_backward_b -- 后复权除权因子 B；用于计算后复权价格(后复权价格=A*价格+B)(str:numpy.float64)

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):stock_exrights = get_stock_exrights
(g.security)log.info('the stock exrights info of security %s:\n%s' % (g.security, stock_exrights))
```

get_stock_blocks - 获取股票所属板块信息

```
get_stock_blocks(stock_code)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取股票所属板块。

注意事项：

该函数获取的是当下的数据，因此回测不能取到真正匹配回测日期的数据，注意未来函数

参数

stock_code: 股票代码(str)；

返回

dict 类型，包含所属行业、板块等详细信息(dict[str:list[list[str,str],...],...])，如：

```
{'HGT': [['HGTHGT.XBHK', '沪股通']],'HY': [['710200.XBHS', '计算机应用']],'DY': [['DY1172.XBHS', '浙江板块']],
'ZJHHY': [['I65000.XBHS', '软件和信息技术服务业']],'GN': [['003596.XBHS', '融资融券'], ['003631.XBHS', '转融券标的'], ['003637.XBHS', '互联网金融'], ['003665.XBHS', '电商概念'], ['003707.XBHS', '沪股通'], ['003718.XBHS', '证金持股'], ['003800.XBHS', '人工智能'], ['003830.XBHS', '区块链'], ['031027.XBHS', 'MSCI 概念'], ['B10003.XBHS', '蚂蚁金服概念']]}
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):blocks = get_stock_blocks(g.securit
y)log.info('security %s in these blocks:\n%s' % (g.security, blocks))
```

get_index_stocks- 获取指数成分股

```
get_index_stocks(index_code,date)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取一个指数在平台可交易的成分股列表，指数列表

注意事项：

1、在回测中，date 不入参默认取当前回测周期所属历史日期

2、在研究中，date 不入参默认取的是当前日期

3、在交易中，date 不入参默认取的是当前日期

参数

index_code：指数代码，尾缀必须是.SS 如沪深 300：000300.SS(str)

date：日期，输入形式必须为'YYYYMMDD'，如'20170620'，不输入默认为当前日期(str)；

返回

返回股票代码的 list(list[str,...])。

```
['000001.SZ', '000002.SZ', '000063.SZ', '000069.SZ', '000100.SZ', '000157.SZ', '000425.SZ', '000538.SZ', '000568.SZ', '000625.SZ', '00065
1.SZ', '000725.SZ', '000728.SZ', '000768.SZ', '000776.SZ','000783.SZ', '000786.SZ', ..., '603338.SS', '603939.SS', '603233.SS', '600426.SS
', '688126.SS', '600079.SS', '600521.SS', '600143.SS', '000800.SZ']
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def before_trading_start(context, data):# 获取当前所有沪深 300 的股票 g.stocks = get_index_stocks('000300.XBHS')log.info(g.stocks)# 获取 2016 年 6 月 20 日所有沪深 300 的股票，设为股票池 g.stocks = get_index_stocks('000300.XBHS','20160620')set_universe(g.stocks)log.info(g.stocks)def handle_data(context, data):pass
```

get_etf_stock_list - 获取 ETF 成分券列表

```
get_etf_stock_list(etf_code)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于获取目标 ETF 的成分券列表

注意事项：

无

参数

etf_code：单支 ETF 代码，必传参数(str)

返回

正常返回一个 list 类型字段，包含每只 etf 代码所对应的成分股。异常返回空 list，如[](list[str,...])

```
['600000.SS', '600010.SS', '600016.SS']
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def before_trading_start(context, data):#ETF 成分券列表 stock_list =
 get_etf_stock_list('510020.SS')log.info(stock_list)def handle_data(context, data):pass
```

get_industry_stocks- 获取行业成份股

```
get_industry_stocks(industry_code)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取一个行业的所有股票，行业列表

注意事项：

该函数获取的是当下的数据，因此回测不能取到真正匹配回测日期的数据，注意未来函数

参数

industry_code: 行业编码，尾缀必须是.XBHS 如农业股：A01000.XBHS(str)

返回

返回股票代码的 list(list[str,...])

```
['300970.SZ', '300087.SZ', '300972.SZ', '002772.SZ', '000998.SZ', '002041.SZ', '600598.SS', '600371.SS', '600506.SS', '300511.SZ', '60035
9.SS', '600354.SS', '601118.SS', '600540.SS', '300189.SZ','600313.SS', '600108.SS']
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def before_trading_start(context, data):# 获取农业的股票，设为股票池
stocks = get_industry_stocks('A01000.XBHS')set_universe(stocks)log.info(stocks)def handle_data(context, data):pass
```

get_fundamentals-获取财务数据

```
get_fundamentals(security, table, fields = None, date = None, start_year = None, end_year = None, report_types = None, date_type = None, me
rge_type = None)
```

使用场景

该函数可在研究、回测、交易模块使用

接口说明

该接口用于获取财务三大报表数据、日频估值数据、各项财务能力指标数据。

注意事项：

1、该接口为 http 在线获取，会存在因网络拥堵等原因导致应答失败的情况，如果返回数据结果为空请多次尝试，策略中请增加保护机制。

2、该接口有流量限制，每秒不得调用超过 100 次，单次最大调用量是 500 条数据，每一条数据的定义为：一个股票对应一个表的一个字段，相当于最大不超过 5 万条。因此如果涉及多股多字段的查询要考虑限流情况，依据实际调用场景加入 sleep 做时间间隔，方法可参考示例。

参数

为保持各表接口统一，输入字段略有不同，具体可参见 财务数据的 API 接口说明

security：一支股票代码或者多只股票代码组成的 list(list[str])

table：财务数据表名，输入具体表名可查询对应表中信息(str)

表名	包含内容
valuation	估值数据
balance_statement	资产负债表
income_statement	利润表
cashflow_statement	现金流量表
growth_ability	成长能力指标
profit_ability	盈利能力指标

表名	包含内容
eps	每股指标
operating_ability	营运能力指标
debt_paying_ability	偿债能力指标

fields：指明数据结果集中所需输出业务字段，支持多个业务字段输出（list 类型），如 fields=['settlement_provi', 'client_provi'](list[str])；

输出具体字段请参考 财务数据的 API 接口说明

date：查询日期，按日期查询模式，返回查询日期之前对应的财务数据，输入形式如'20170620'，回测中支持 datetime.date 时间格式输入，不能与 start_year 与 end_year 同时作用。回测中，支持按日期查询模式，不传入 date 默认取回测时的日期(str)；

start_year：查询开始年份，按年份查询模式，返回输入年份范围内对应的财务数据，如'2015'，start_year 与 end_year 必须同时输入，且不能与 date 同时作用(str)

end_year：查询截止年份，按年份查询模式，返回输入年份范围内对应的财务数据，如'2015'，start_year 与 end_year 必须同时输入，且不能与 date 同时作用(str)

report_types：财报类型；如果为年份查询模式（start_year/end_year），不输入 report_types 返回当年可查询到的全部类型财报；如果为日期查询模式（date），不输入 report_types 返回距离指定日期最近一份财报(str)。

- report_types='1':表示获取一季度财报
- report_types='2':表示获取半年报
- report_types='3':表示获取截止到三季度财报
- report_types='4':表示获取年度财报

date_type：数据参考时间设置，该参数只适用于按日期查询模式(date 参数模式)(int)：

- date_type 不传或传入 date_type = None，返回发布日期（publ_date）在查询日期（date）之前指定财报类型数据（report_types），若未指定财报类型（report_types）则默认为离查询日期（date）最近季度的数据，数据未公布用 NAN 填充
- date_type 传入 1，返回会计周期（end_date）在查询日期（date）之前指定财报类型数据（report_types），若未指定财报类型（report_types）则默认为查询日期（date）最近季度会计周期的数据，数据未公布用 NAN 填充

merge_type：数据更新设置；相关财务数据信息会不断进行修正更新，为了避免未来数据影响，可以通过参数获取原始发布或最新发布数据信息；只有部分表包含此字段(int)：

- merge_type 不传或传入 merge_type = None，获取首次发布的数据，即使实际数据发生变化，也只返回原样数据信息；回测场景为避免未来数据建议使用此模式
- merge_type=传入 1，获取最新发布的数据，更新数据范围包括但不限于相关日期数据，研究场景或交易场景建议使用此模式

注意：

- date 字段与 start_year/end_year 不能同时输入，否则按日期查询模式（date 参数模式）。
- 当 date 和 start_year/end_year 相关数据都不传入时，默认为按日期查询模式（date 参数模式），研究和回测中 date 取值有所不同：在研究中，date 取的是当前日期/

返回

返回值形式根据输入参数类型不同而有所区分：

1.按日期查询模式（date 参数模式）返回数据类型为 pandas.DataFrame 类型，索引为股票代码，如 get_fundamentals('600000.SS','balance_statement',date='20161201')将返回：

	secu_abbr	end_date	publ_date	total_assets	.....	total_liability
600000.SS	浦发银行	2016-09-30	2016-10-29	5.56e+12	.....	5.20e+12

2.按年份查询模式（start_year/end_year 参数模式）返回数据类型为 pandas.Panel 类型，索引为股票代码，其中包含的 DataFrame 索引为返回股票对应会计日期（end_date），如 get_fundamentals(['600000.SS', '600570.SS', '000002.SZ'], 'balance_statement', start_year='2016', end_year='2016')将返回：

000002.SZ

	secu_code	secu_abbr	publ_date	total_assets	.....	total_liability
2016/12/31	000002.SZ	万科A	2017/4/1	5.86E+12	.....	5.48E+12

600570.SS

	secu_code	secu_abbr	publ_date	total_assets	.....	total_liability
2016/12/31	600570.SS	恒生电子	2017/4/1	5.86E+12	.....	5.48E+12

600000.SS

	secu_code	secu_abbr	publ_date	total_assets	.....	total_liability
2016/12/31	600000.SS	浦发银行	2017/4/1	5.86E+12	.....	5.48E+12
2016/9/30	600000.SS	浦发银行	2016/10/29	5.56E+12	.....	5.20E+12
2016/6/30	600000.SS	浦发银行	2016/8/11	5.37E+12	.....	5.02E+12
2016/3/31	600000.SS	浦发银行	2016/4/30	5.23E+12	.....	4.88E+12

示例

```
import time
def initialize(context):g.security = '600570.SS'set_universe(g.security)def before_trading_start(context, data):# 假设取 4000 股*10 年一季报数据为 4 万条，之后再取中报又是 4 万条，因为规则要求每秒不得调用超过 100 次（单次最大调用量是 500 条数据），调用过程就需要 sleep1 秒，防止流控触发。funda_data = get_fundamentals(g.security, 'balance_statement', fields = 'total_assets', start_year='2011', end_year='2020', report_types = '1')time.sleep(1)funda_data = get_fundamentals(g.security, 'balance_statement', fields = 'total_assets', start_year='2010', end_year='2020', report_types = '2')def handle_data(context, data):# 获取股票池 stocks = get_index_stocks('000906.XBHS')# 指定股票池 stocks = ['600000.SS','600570.SS']# 获取数据的两种模式# 1. 按日期查询模式（默认以发布日期为参考时间）：返回输入日期之前对应的财务数据# 在回测中获取单一股票中对应回测日期资产负债表中资产总计（total_assets）数据#（回测中 date 默认获取回测日期，无需传入 date，除非在回测中获取指定某个日期的数据，日期格式如”20160628”）get_fundamentals('600000.SS', 'balance_statement', 'total_assets')# 获取股票池中对应上市公司在 2016 年 6 月 28 日之前发布的最近季度（即 2016 年一季度）# 的资产负债表中资产总计（total_assets）数据，如果到查询日期为止一季度数据还,未发布则所有数据用 Nan 填充 get_fundamentals(stocks, 'balance_statement', 'total_assets','20160628')# 获取股票池中对应上市公司在 2016 年 6 月 28 日最近会计周期（即 20160331）的资产负# 债表中资产总计（total_assets）数据，如果未查到相关数据则用 Nan 填充 get_fundamentals(stocks, 'balance_statement', 'total_assets','20160628', date_type=1)# 获取股票池中对应上市公司发布日期在 2016 年 6 月 28 日之前，年度（即 2015 年年报）# 资产负债表中资产总计（total_assets）数据，如果到查询日期为止还未发布则所有数据用 Nan 填充 get_fundamentals(stocks, 'balance_statement', 'total_assets', '20160628', report_types='4')# 获取股票池中对应上市公司 2016 年 6 月 28 日最近季度资产负债表中对应 fields 字段数据 fields =['sold_buyback_secu_proceeds','specific_account_payable']get_fundamentals(stocks, 'balance_statement', fields,'20160628',)# 获取股票池中对应上市公司 2016 年 6 月 28 日最近季度资产负债表中对应 fields 字段最新数据，# 如果最近更新日期（发布日期）在 2016 年 6 月 28 日之后则无法获取对应数据 fields =['sold_buyback_secu_proceeds','specific_account_payable']get_fundamentals(stocks, 'balance_statement', fields,'20160628',merge_type=1)# 2. 按年份查询模式：返回输入年份范围内对应季度的财务数据# 获取公司浦发银行(600000.SS)从 2013 年至 2015 年第一季度资产负债表中资
```

```
产总计（total_assets）数据 get_fundamentals('600000.SS','balance_statement','total_assets',start_year='2013',end_year='2015', report_types='1')# 获取股票池中对应上市公司从 2013 年至 2015 年年度资产负债表中对应 fields 字段数据 fields =['sold_buyback_secu_proceeds','specific_account_payable']get_fundamentals(stocks,'balance_statement',fields,start_year='2013',end_year='2015', report_types='4')
```

get_Ashares– 获取指定日期 A 股代码列表

```
get_Ashares(date=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

该接口用于获取指定日期沪深市场的所有 A 股代码列表

注意事项：

1、在回测中，date 不入参默认取回测日期，默认值会随着回测日期变化而变化，等于 context.current_dt

2、在研究中，date 不入参默认取当天日期

3、在交易中，date 不入参默认取当天日期

参数

date：格式为 YYYYmmdd

返回

股票代码列表，list 类型(list[str,...])

```
['000001.SZ', '000002.SZ', '000004.SZ', '000005.SZ', '000006.SZ', '000007.SZ', '000008.SZ', '000009.SZ', '000010.SZ', '000011.SZ', '00001
2.SZ', '000014.SZ', '000016.SZ', '000017.SZ', '000018.SZ', '000019.SZ','000020.SZ', '000021.SZ', '000023.SZ', '000024.SZ', '000025.SZ', '0
00026.SZ', '000027.SZ',..., '603128.SS', '603167.SS', '603333.SS', '603366.SS', '603399.SS', '603766.SS', '603993.SS']
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#沪深 A 股代码 ashares = get_Ashares
()log.info('%s A 股数量为%s' % (context.blotter.current_dt,len(ashares)))ashares = get_Ashares('20130512')log.info('20130512 A 股数量为%s'%l
en(ashares))
```

get_etf_list - 获取 ETF 代码

```
get_etf_list()
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于获取柜台返回的 ETF 代码列表

注意事项：

无

返回

正常返回一个 list 类型对象，包含所有 ETF 代码。异常返回空 list，如[](list[str,...])。

```
['510010.SS', '510020.SS', '510030.SS', '510050.SS', '510060.SS', '510180.SS', '510300.SS', '510310.SS', '510330.SS', '511800.SS', '51181
0.SS', '511820.SS', '511830.SS', '511880.SS', '511990.SS', '512010.SS','512510.SS', '159001.SZ', '159003.SZ', '159005.SZ', '159901.SZ', '1
59903.SZ', '159905.SZ', '159906.SZ', '159909.SZ', '159910.SZ', '159919.SZ', '159923.SZ', '159923.SZ', '159924.SZ', '159925.SZ', '159927.SZ
','159928.SZ', '159929.SZ']
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#ETF 代码列表 etf_code_list = get_etf
_list()log.info('ETF 列表为%s' % etf_code_list)
```

获取其他信息

get_trades_file– 获取对账数据文件

```
get_trades_file(save_path='')
```

使用场景

该函数仅在回测模块可用

接口说明

该接口用于获取对账数据文件

注意事项：

文件目录的命名需要遵守如下规则：

1、长度不能超过 256 个字符；

2、名称中不能出下如下字符：:?,@#$&();\"\'<>`~!%^*;

参数

save_path：导出对账数据存储的路径， 默认在 notebook 的根目录下(str)；

返回

成功返回导出文件的路径,失败返回 None(str);

```
导出数据格式的说明:交易数据文件的组织格式为 csv 文件，表头信息为：订单编号，成交编号，委托编号，标的代码，交易类型，成交数量，成交价，成交金额，交易费用，交易时间，对应的表头字段为：[order_id,trading_id,entrust_id,security_code,order_type,volume,price,total_money,trading_fee, trade_time]
```

注意:

order_id 列中可能出现如下几种取值:

1、M000000，通过外部系统委托的成交数据;

2、类似 a6fbc145958843cc86639b23fbcfdc4c 的字符串，通过平台委托的成交数据;

3、H000000，引入对账数据接口前的版本产生的交易数据;

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 委托 order_obj = order(g.security, 100)log.info('订单编号为：%s'% order_obj)def after_trading_end(context, data):# 获取对账数据，存放到默认目录 data_path = get_trades_file()log.info(data_path)# 获取对账数据，存放到 notebook 下的指定目录 user_data_path = get_trades_file('user_data/data')log.info(user_data_path)
```

convert_position_from_csv– 获取设置底仓的参数列表(股票)

```
convert_position_from_csv(path)
```

使用场景

该函数仅在回测模块可用

接口说明

该接口用于从 csv 文件中获取设置底仓的参数列表

注意事项：

文件目录的命名需要遵守如下规则：

1、长度不能超过 256 个字符；

2、名称中不能出下如下字符：:?,@#$&();\"\'<>`~!%^*;

参数

path: csv 文件对应路径及文件名(需要在研究中上传该文件)(str)；

csv 文件内容格式要求如下:

```
sid,enable_amount,amount,cost_basis
600570.SS,10000,10000,45
```

- sid: 标的代码(str)；
- amount: 持仓数量(str)；
- enable_amount: 可用数量(str)；
- cost_basis: 每股的持仓成本价格(str)：

返回

用于设置底仓的参数列表，该 list 中是字典类型的元素；

返回一个 list，该 list 中是一个字典类型的元素(list[dict[str:str],...])，如：

```
[{
'sid':标的代码,
'amount':持仓数量,
'enable_amount':可用数量,
'cost_basis':每股的持仓成本价格,
}]
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)# 设置底仓 poslist= convert_position_from_csv("Poslist.csv")set_yes
terday_position(poslist)def handle_data(context, data):# 卖出 100 股 order(g.security, -100)
```

get_user_name– 获取登录终端的资金账号

```
get_user_name()
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取登录终端的账号

注意事项：

无

返回

返回登录终端的资金账号(str)或者 None。如果查询成功登录终端的资金账号(str)，失败则返回 None。

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)g.user_name = get_user_name()def before_trading_start(context, dat
a):g.flag = Falsedef handle_data(context, data):# 账号为 123456789 且当日未委托过，买入 100 股 if g.user_name == "123456789" and not g.flag:# 买
```

```
入 100 股 order(g.security, 100)g.flag = True
```

get_deliver– 获取历史交割单信息

```
get_deliver(start_date, end_date)
```

使用场景

该函数仅在交易模块使用；仅支持 before_trading_start 和 after_trading_end 阶段调用

接口说明

该接口用来获取账户历史交割单信息。

注意事项：

1、开始日期 start_date 和结束日期 end_date 为必传字段

2、仅支持查询上一个交易日（包含）之前的交割单信息

3、因不同柜台返回的字段存在差异，因此接口返回的为柜台原数据，使用时请根据实际柜台信息做字段解析

4、该接口仅支持查询普通股票账户（非两融）

参数

start_date: 开始日期，输入形式仅支持"YYYYmmdd"，如'20170620';

end_date: 结束日期，输入形式仅支持"YYYYmmdd"，如'20170620';

返回

返回一个 list 类型对象(list[dict,...])，包含一个或 N 个 dict，每个 dict 为一条交割单信息，其中包含柜台返回的字段信息，失败则返回[]。


```
[{'entrust_way': '7', 'exchange_fare': 0.04, 'post_balance': 3539128.83, 'stock_account': '0010110920', 'exchange_farex': 0.0, 'fare0': 0.
5, 'report_milltime': 110400187, 'business_balance': 2987.0, 'exchange_fare5': 0.0, 'fare_remark': '内部:.5( | ,费用类别:9999)', 'client_id
': '10110920', 'uncome_flag': '0', 'exchange_fare0': 0.03, 'exchange_fare2': 0.0, 'fare1': 0.0, 'init_date': 20210811, 'stock_code': '1626
05', 'occur_amount': 1000.0, 'report_time': 110400, 'entrust_bs': '1', 'seat_no': '123456', 'business_id': '0110351000000242', 'business_a
mount': 1000.0, 'business_time': 110351, 'fund_account': '10110920', 'begin_issueno': ' ', 'post_amount': 1000.0, 'correct_amount': 0.0, '
money_type': '0', 'client_name': '客户10110920', 'business_type': '0', 'business_flag': 4002, 'clear_balance': -2987.5, 'exchange_fare1':
0.0, 'date_back': 20210811, 'branch_no': 1011, 'serial_no': 153, 'occur_balance': -2987.5, 'stock_name': '景顺鼎益', 'curr_time': 173028, '
exchange_fare4': 0.0, 'brokerage': 0.0, 'business_name': '证券买入', 'order_id': 'F04Z', 'business_times': 1, 'entrust_date': 20210811, 're
mark': '证券买入;uft节点:31;', 'exchange_fare6': 0.0, 'standard_fare0': 0.5, 'exchange_fare3': 0.01, 'farex': 0.0, 'clear_fare0': 0.46, 'en
trust_no': 38, 'profit': 0.0, 'exchange_type': '2', 'fare2': 0.0, 'business_no': 181, 'stock_type': 'L', 'fare3': 0.0, 'business_status':
'0', 'business_price': 2.987, 'position_str': '020210811010110000000153', 'stock_name_long': '景顺鼎益LOF', 'report_no': 38, 'correct_bala
nce': 0.0, 'exchange_rate': 0.0}]
```


示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)def before_trading_start(context, data):h = get_deliver('20210101
', '20211117')log.info(h)def handle_data(context, data):pass
```

get_fundjour– 获取历史资金流水信息

```
get_fundjour(start_date, end_date)
```

使用场景

该函数仅在交易模块使用；仅支持 before_trading_start 和 after_trading_end 阶段调用

接口说明

该接口用来获取账户历史资金流水信息。

注意事项：

1、开始日期 start_date 和结束日期 end_date 为必传字段

2、仅支持查询上一个交易日（包含）之前的资金流水信息

3、因不同柜台返回的字段存在差异，因此接口返回的为柜台原数据，使用时请根据实际柜台信息做字段解析

4、该接口仅支持查询普通股票账户（非两融）

参数

start_date: 开始日期，输入形式仅支持"YYYYmmdd"，如'20170620'；

end_date: 结束日期，输入形式仅支持"YYYYmmdd"，如'20170620'；

返回

返回一个 list 类型对象(list[dict,...])，包含一个或 N 个 dict，每个 dict 为一条资金流水，其中包含柜台返回的字段信息，失败则返回[]。


```
[{'post_balance': 3260341.36, 'init_date': 20210104, 'asset_prop': '0', 'serial_no': 1, 'business_flag': 4002, 'occur_balance': -10598.21,
 'exchange_type': '0', 'stock_name': ' ', 'business_date': 20210104, 'business_price': 0.0, 'bank_no': '0', 'occur_amount': 0.0, 'remark':
 '证券买入,恒生电子,100 股,价格 105.93', 'stock_account': ' ', 'money_type': '0', 'fund_account': '10110920', 'position_str': '20210104010110
000000001', 'bank_name': '内部银行', 'business_name': '证券买入', 'stock_code': ' ', 'curr_date': 20210104, 'entrust_bs': ' ', 'business_tim
e': 171730}]
```


示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)def before_trading_start(context, data):h = get_fundjour('20210101
', '20211117')log.info(h)def handle_data(context, data):pass
```

get_research_path– 获取研究路径

```
get_research_path()
```

使用场景

该函数可在回测、交易模块使用

接口说明

该接口用于获取研究根目录路径，该路径为'/home/fly/notebook/'。

注意事项：

无

参数

无

返回

返回一个字符串类型对象(str)

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)path = get_research_path()def handle_data(context, data):pass
```

get_trade_name– 获取交易名称

```
get_trade_name()
```

使用场景

该函数仅在交易模块使用

接口说明

该接口用于获取当前交易的名称。

注意事项:

无

参数

无

返回

返回一个字符串类型对象(str)

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)def handle_data(context, data):name = get_trade_name()
```

交易相关函数

注意：代码精度位为 3 位小数的类型(后台已保护为 3 位)，如 ETF、国债；代码精度为 2 位小数类型，需要在传参时限制价格参数的精度，如股票。

股票交易函数

order-按数量买卖

```
order(security, amount, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于买卖指定数量为 amount 的股票，同时支持国债逆回购

注意事项：

1、支持交易场景的逆回购交易。委托方向为卖出(amount 必须为负数)，逆回购最小申购金额为 1000 元(10 张)，因此本接口 amount 入参应大于等于 10(10 张)，否则会导致委托失败。

2、回测场景，amount 有最小下单数量校验，股票、ETF、LOF：100 股，可转债：10 张；交易场景接口不做 amount 校验，直接报柜台。

3、交易场景如果 limit_price 字段不入参，系统会默认用行情快照数据最新价报单，假如行情快照获取失败会导致委托失败，系统会在日志中增加提醒。

4、由于下述原因，回测中实际买入或者卖出的股票数量有时候可能与委托设置的不一样，针对上述内容调整，系统会在日志中增加警告信息：

1. 根据委托买入数量与价格经计算后的资金数量，大于当前可用资金；
2. 委托卖出数量大于当前可用持仓数量；
3. 每次交易股票时取整 100 股，交易可转债时取整 10 张，但是卖出所有股票时不受此限制；
4. 股票停牌、股票未上市或者退市、股票不存在；
5. 回测中每天结束时会取消所有未完成交易；

参数

security：股票代码(str)；

amount: 交易数量，正数表示买入，负数表示卖出(int)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):#以系统最新价委托 order('600570.SS', 100)# 逆回购 1000 元 order('131810.SZ', -10)#以 39 块价格下一个限价单 order('600570.SS', 100, limit_price=39)
```

order_target - 指定目标数量买卖

```
order_target(security, amount, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于买卖股票，直到股票最终数量达到指定的 amount

注意事项：

1、该函数不支持逆回购交易。

2、该函数在委托股票时取整 100 股，委托可转债时取整 10 张。

3、交易场景如果 limit_price 字段不入参，系统会默认用行情快照数据最新价报单，假如行情快照获取失败会导致委托失败，系统会在日志中增加提醒。

4、因可能造成重复下单，因此建议在交易中谨慎使用该接口。具体原因如下：

- 柜台返回持仓数据体现当日变化(由柜台配置决定)：交易场景中持仓信息同步有时滞，一般在 6 秒左右，假如在这 6 秒之内连续下单两笔或更多 order_target 委托，由于持仓数量不会瞬时更新，会造成重复下单。
- 柜台返回持仓数据体现当日变化(由柜台配置决定)：第一笔委托未完全成交，如果不对第一笔做撤单再次 order_target 相同的委托目标数量，引擎不会计算包括在途的总委托数量，也会造成重复下单。
- 柜台返回持仓数据不体现当日变化(由柜台配置决定)：这种情况下持仓数量只会一天同步一次，必然会造成重复下单。

针对以上几种情况，假如要在交易场景使用该接口，首先要确定券商柜台的配置，是否实时更新持仓情况，其次需要增加订单和持仓同步的管理，来配合 order_target 使用。

参数

security：股票代码(str)；

amount：期望的最终数量(int)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):#买卖恒生电子股票数量到 100 股 order_target('600570.SS', 100)#卖出恒生电子所有股票 if data['600570.SS']['close'] > 39:order_target('600570.SS', 0)
```

order_value - 指定目标价值买卖

```
order_value(security, value, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于买卖指定价值为 value 的股票

注意事项：

1、该函数不支持逆回购交易。

2、该函数在委托股票时取整 100 股，委托可转债时取整 10 张。

3、交易场景如果 limit_price 字段不入参，系统会默认用行情快照数据最新价报单，假如行情快照获取失败会导致委托失败，系统会在日志中增加提醒。

参数

security：股票代码(str)；

value：股票价值(float)

limit_price：买卖限价(float)

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):#买入价值为 10000 元的恒生电子股票 order_value('600570.SS', 10000)if data['600570.SS']['close'] > 39:#卖出价值为 10000 元的恒生电子股票 order_value('600570.SS', -10000)
```

order_target_value - 指定持仓市值买卖

```
order_target_value(security, value, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于调整股票持仓市值到 value 价值

注意事项：

1、该函数不支持逆回购交易。

2、该函数在委托股票时取整 100 股，委托可转债时取整 10 张。

3、交易场景如果 limit_price 字段不入参，系统会默认用行情快照数据最新价报单，假如行情快照获取失败会导致委托失败，系统会在日志中增加提醒。

4、因可能造成重复下单，因此建议在交易中谨慎使用该接口。具体原因如下：

- 柜台返回持仓数据体现当日变化(由柜台配置决定)：交易场景中持仓信息同步有时滞，一般在 6 秒左右，假如在这 6 秒之内连续下单两笔或更多 order_target_value 委托，由于持仓市值不会瞬时更新，会造成重复下单。
- 柜台返回持仓数据体现当日变化(由柜台配置决定)：第一笔委托未完全成交，如果不对第一笔做撤单再次 order_target_value 相同的委托目标金额，引擎不会计算包括在途的总委托数量，也会造成重复下单。
- 柜台返回持仓数据不体现当日变化(由柜台配置决定)：这种情况下持仓金额只会一天同步一次，必然会造成重复下单。

针对以上几种情况，假如要在交易场景使用该接口，首先要确定券商柜台的配置，是否实时更新持仓情况，其次需要增加订单和持仓同步的管理，来配合 order_target_value 使用。

参数

security：股票代码(str)；

value: 期望的股票最终价值(float)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):#买卖股票到指定价值 order_target_value('600570.SS', 10000)#卖出当前所有恒生电子的股票 if data['600570.SS']['close'] > 39:order_target_value('600570.SS', 0)
```

order_market - 按市价进行委托

```
order_market(security, amount, market_type, limit_price=None)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于使用多种市价类型进行委托

注意事项：

1、支持逆回购交易。委托方向为卖出(amount 必须为负数)，逆回购最小申购金额为 1000 元(10 张)，因此本接口 amount 入参应大于等于 10(10 张)，否则会导致委托失败。

2、不支持可转债交易。

3、该函数中 market_type 是必传字段，如不传入参数会出现报错。

参数

security：股票代码(str)；

amount：交易数量，正数表示买入，负数表示卖出(int)；

market_type：市价委托类型，上证非科创板股票支持参数 1、4，上证科创板股票支持参数 0、1、2、4，深证股票支持参数 0、2、3、4、5，必传参数(int)

limit_price：保护限价(仅限科创板代码)(float)；

- 0：对手方最优价格；

- 1：最优五档即时成交剩余转限价；
- 2：本方最优价格；
- 3：即时成交剩余撤销；
- 4：最优五档即时成交剩余撤销；
- 5：全额成交或撤单；

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):security = g.security
order_market(security, 100, 1)
```

ipo_stocks_order - 新股一键申购

```
ipo_stocks_order(market_type=None, black_stocks=None)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于一键申购当日全部新股

注意事项：

申购黑名单的股票代码必须为申购代码，代码可以是 6 位数（不带尾缀），也可以带尾缀入参,比如：black_stocks='787001'或 black_stocks='787001.SS'。

参数

market_type：申购代码所属市场，不传时默认申购全部新股(int)；

black_stocks：黑名单股票，可以是单个股票或者股票列表，传入的黑名单股票将不做申购，不传时默认申购全部新股(str/list)；

- 0：上证普通代码；
- 1：上证科创板代码；
- 2：深证普通代码；
- 3：深证创业板代码；
- 4：可转债代码；

返回

返回 dict 类型，包含委托代码、委托编号、委托状态(委托失败为 0，委托成功为 1)、委托数量等信息(dict[str:dict[str:str,str:int,str:float],...])

示例

```
import time
def initialize(context):g.security = "600570.SS"set_universe(g.security)g.flag = Falsedef before_trading_start(context, data):g.flag = Fal
sedef handle_data(context, data):if not g.flag:# 上证普通代码 log.info("申购上证普通代码：")ipo_stocks_order(market_type=0)time.sleep(5)# 上
证科创板代码 log.info("申购上证科创板代码：")ipo_stocks_order(market_type=1)time.sleep(5)# 深证普通代码 log.info("申购深证普通代码：")ipo_stocks
_order(market_type=2)time.sleep(5)# 深证创业板代码 log.info("申购深证创业板代码：")ipo_stocks_order(market_type=3)time.sleep(5)# 可转债代码 lo
g.info("申购可转债代码：")ipo_stocks_order(market_type=4)time.sleep(5)g.flag = True
```

after_trading_order - 盘后固定价委托(股票)

```
after_trading_order(security, amount, entrust_price)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于盘后固定价委托申报

注意事项：

1、entrust_price 为必传字段

2、盘后固定价委托时间为 9:30~11:30,13:00~15:30

参数

security：股票代码(str)；

amount：交易数量，正数表示买入，负数表示卖出(int)；

entrust_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = "300001.SZ"set_universe(g.security)# 15:00-15:30 期间使用 run_daily 进行盘后固定价委托 run_daily(context,
 order_test, time="15:15")g.flag = Falsedef order_test(context):snapshot = get_snapshot(g.security)if snapshot is not None:last_px = snaps
hot[g.security].get("last_px", 0)if last_px > 0:after_trading_order(g.security, 200, float(last_px))def handle_data(context, data):if not
g.flag:snapshot = get_snapshot(g.security)if snapshot is not None:last_px = snapshot[g.security].get("last_px", 0)if last_px > 0:after_tra
```

```
ding_order(g.security, 200, float(last_px))g.flag = True
```

after_trading_cancel_order - 盘后固定价委托撤单(股票)

```
after_trading_cancel_order(order_param)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于盘后固定价委托取消订单，根据 Order 对象或 order_id 取消订单。

注意事项：

无

参数

order_param: Order 对象或者 order_id(Order/str)

返回

None

示例

```
import time
def initialize(context):g.security = "300001.SZ"set_universe(g.security)# 15:00-15:30 期间使用 run_daily 进行盘后固定价委托、盘后固定价委托撤单
run_daily(context, order_test, time="15:15")g.flag = Falsedef order_test(context):snapshot = get_snapshot(g.security)if snapshot is not No
ne:last_px = snapshot[g.security].get("last_px", 0)if last_px > 0:order_id = after_trading_order(g.security, 200, float(last_px))time.slee
p(5)after_trading_cancel_order(order_id)def handle_data(context, data):if not g.flag:snapshot = get_snapshot(g.security)if snapshot is not
 None:last_px = snapshot[g.security].get("last_px", 0)if last_px > 0:order_id = after_trading_order(g.security, 200, float(last_px))time.s
leep(5)after_trading_cancel_order(order_id)g.flag = True
```

etf_basket_order - ETF 成分券篮子下单

```
etf_basket_order(etf_code ,amount, price_style=None, position=True, info=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于 ETF 成分券篮子下单。

注意事项：

无

参数

etf_code：单支 ETF 代码，必传参数(str)

amount：下单篮子份数，正数表示买入，负数表示卖出，必传参数(int)

price_style：设定委托价位，可传入’ B1’ 、’ B2’ 、’ B3’ 、’ B4’ 、’ B5’ 、’ S1’ 、’ S2’ 、’ S3’ 、’ S4’ 、’ S5’ 、’ new’ ，分别为买一~买五、卖一~卖五、最新价，默认为最新价(str)

position：取值 True 和 False，仅在篮子买入时使用。申购是否使用持仓替代，True 为使用，该情况下篮子股票买入时使用已有的持仓部分；False 为不使用。默认使用持仓替代(bool)

info：dict 类型，成份股信息。key 为成分股代码，values 为 dict 类型，包含的成分股信息字段作为 key(Mapping[str, Mapping[str, Union[int, float]]]):

- cash_replace_flag -- 设定现金替代标志，1 为替代，0 为不替代，仅允许替代状态的标的传入有效，否则无效，如不传入 info 或不传入该字段信息系统默认为成分股不做现金替代
- position_replace_flag -- 设定持仓替代标志，1 为替代，0 为不替代，如不传入 info 或不传入该字段信息按 position 参数的设定进行计

算

- limit_price -- 设定委托价格，如不传入 info 或不传入该字段信息按 price_style 参数的设定进行计算

返回

创建订单成功，正常返回一个 dict 类型字段， key 为股票代码，values 为 Order 对象的 id，失败则返回空 dict，如{}(dict[str:str])

示例

```
def initialize(context):g.security = get_Ashares()set_universe(g.security)def handle_data(context, data):#ETF 成分券篮子下单 etf_basket_orde
r('510050.SS' ,1, price_style='S3',position=True)stock_info = {'600000.SS':{'cash_replace_flag':1,'position_replace_flag':1,'limit_price':
12}}etf_basket_order('510050.SS' ,1, price_style='S2',position=False, info=stock_info)
```

etf_purchase_redemption - ETF 基金申赎接口

```
etf_purchase_redemption(etf_code,amount,limit_price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用、仅在股票交易模块可用

接口说明

该接口用于单只 ETF 基金申赎。

注意事项：

无

参数

etf_code：单支 ETF 代码，必传参数(str)

amount：基金申赎数量，正数表示申购，负数表示赎回(int)

返回

创建订单成功，返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '510050.SS'set_universe(g.security)def handle_data(context, data):#ETF 申购 etf_purchase_redemption('510050.SS',900000)#ETF 赎回 etf_purchase_redemption('510050.SS',-900000,limit_price = 2.9995)
```

get_positions - 获取多支股票持仓信息

```
get_positions(security)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取多支股票的持仓信息详情。

注意事项：

无

参数

security：股票代码，可以为一个列表，不传时默认为获取所有持仓(list[str]/str)；

返回

返回一个数据字典，键为股票代码，值为 Position 对象(dict[str:Position])，如下：

注意：四位尾缀或者两位尾缀代码皆可作为键取到返回的数据字典值，如'600570.XSHG'或者'600570.SS'。

```
{'600570.XSHG': <Position {'business_type': 'stock', 'short_amount': 0, 'contract_multiplier': 1, 'short_pnl': 0, 'delivery_date': None, '
today_short_amount': 0, 'last_sale_price': 118.7, 'sid': '600570.SS','enable_amount': 100, 'margin_rate': 1, 'amount': 200, 'long_amount':
 0, 'short_cost_basis': 0, 'today_long_amount': 0, 'cost_basis': 117.9, 'long_pnl': 0, 'long_cost_basis': 0}>}
```

示例

```
def initialize(context):g.security = ['600570.SS','600000.SS']set_universe(g.security)def handle_data(context, data):log.info(get_position
s('600570.SS'))log.info(get_positions(g.security))log.info(get_positions())
```

公共交易函数

order_tick - tick 行情触发买卖

```
order_tick(sid, amount, priceGear='1', limit_price=None)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于在 tick_data 模块中进行买卖股票下单，可设定价格档位进行委托。

注意事项：

该函数只能在 tick_data 模块中使用

参数

sid：股票代码(str)；

amount：交易数量，正数表示买入，负数表示卖出(int)

priceGear：盘口档位，level1:1~5 买档/-1~-5 卖档，level2:1~10 买档/-1~-10 卖档(str)

limit_price：买卖限价，当输入参数中也包含 priceGear 时，下单价格以 limit_price 为主(float)；

返回

返回一个委托流水编号(str)

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)def tick_data(context,data):security = g.security
current_price = eval(data[security]['tick']['bid_grp'][0])[1][0]if current_price > 56 and current_price < 57:# 以买一档下单 order_tick(g.sec
urity, -100, "1")# 以卖二档下单 order_tick(g.security, 100, "-2")# 以指定价格下单 order_tick(g.security, 100, limit_price=56.5)def handle_dat
a(context, data):pass
```

cancel_order - 撤单

```
cancel_order(order_param)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于取消订单，根据 Order 对象或 order_id 取消订单。

注意事项：

无

参数

order_param: Order 对象或者 order_id(Order/str)

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):_id = order(g.security, 100)cancel_
```

```
order(_id)log.info(get_order(_id))
```

cancel_order_ex - 撤单

```
cancel_order_ex(order_param)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于取消订单，根据 get_all_orders 返回列表中的单个字典取消订单。

注意事项：

该函数仅可撤 get_all_orders 函数返回的可撤状态订单。

账户多个交易运行时调用该函数会撤销其他交易产生的订单，可能对其他正在运行的交易策略产生影响。

参数

order_param: get_all_orders 函数返回列表的单个字典(dict)

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)g.count = 0def handle_data(context, data):if g.count == 0:log.info
("当日全部订单为: %s" % get_all_orders())# 遍历账户当日全部订单，对已报、部成状态订单进行撤单操作 for _order in get_all_orders():if _order['statu
s'] in ['2', '7']:cancel_order_ex(_order)if g.count == 1:# 查看撤单是否成功 log.info("当日全部订单为: %s" % get_all_orders())g.count += 1
```

debt_to_stock_order - 债转股委托

```
debt_to_stock_order(security, amount)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于可转债转股操作。

注意事项:

无

参数

security: 可转债代码(str)

amount: 委托数量(int)

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)。

示例

```
def initialize(context):g.security = "600570.SS"set_universe(g.security)def before_trading_start(context, data):g.count = 0def handle_data(context, data):if g.count == 0:# 对持仓内的国贸转债进行转股操作 debt_to_stock_order("110033.SS", -1000)g.count += 1# 查看委托状态 log.info(get_orders())g.count += 1
```

get_open_orders - 获取未完成订单

```
get_open_orders(security=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取当天所有未完成的订单，或按条件获取指定未完成的订单。

注意事项：

无

参数

security：标的代码，如'600570.SS'，不传时默认为获取所有未成交订单(str)；

返回

返回一个 list，该 list 中包含多个 Order 对象(list[Order,...])。

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):for _sec in g.securi
ty:_id = order(_sec, 100, limit_price = 30)# 当运行周期为分钟则可获取本周期及之前所有未完成的订单 dict_list = get_open_orders()log.info(dict_li
st)# 当运行周期为天，可在 after_trading_end 中调用此函数获取当天未完成的订单 def after_trading_end(context, data):dict_list = get_open_orders()l
og.info(dict_list)
```

get_order - 获取指定订单

```
get_order(order_id)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取指定编号订单。

注意事项：

无

获取指定编号订单。

参数

order_id：订单编号(str)

返回

返回一个 list，该 list 中只包含一个 Order 对象(list[Order])。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order_id = order(g.security, 100)cu
rrent_order = get_order(order_id)log.info(current_order)
```

get_orders - 获取全部订单

```
get_orders(security=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取策略内所有订单，或按条件获取指定订单。

注意事项：

无

参数

security：标的代码，如'600570.SS'，不传时默认为获取所有订单(str)；

返回

返回一个 list，该 list 中包含多个 Order 对象(list[Order,...])。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):_id = order(g.security, 100)order_o
bj = get_orders()log.info(order_obj)
```

get_all_orders - 获取账户当日全部订单

```
get_all_orders(security=None)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于获取账户当日所有订单(包含非本交易的订单记录)，或按条件获取指定代码的订单。

注意事项：

1、该函数返回账户当日在柜台的全部委托记录，不能查询策略中待报、未报状态的委托。

2、该函数返回的可撤委托仅可通过 cancel_order_ex 函数进行撤单，且非本交易的委托进行撤单仅可通过本函数查询委托状态更新。

参数

security：标的代码，如'600570.SS'，不传时默认为获取所有订单(str)；

返回

返回一个 list，该 list 中包含多条订单记录(list[dict, ...])：

[{'symbol': 股票代码(str), 'entrust_no': 委托编号(str), 'amount': 委托数量(int), 'entrust_bs': 委托方向(int), 'price': 委托价格(float), 'status': 委托状态(str), 'filled_amount': 成交数量(int)}, ...]

字段备注:

- entrust_bs -- 成交方向，1-买，2-卖；
- status -- 委托状态，详见 Order 对象中 status 字段；

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取账户当日委托 600570 代码的全部订单 log.info('当日委托 600570 代码的全部订单：%s' % get_all_orders(g.security))# 获取账户当日全部订单 log.info('当日全部订单：%s' % get_all_orders())
```

get_trades - 获取当日成交订单

```
get_trades()
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取策略内当日已成交订单详情。

注意事项：

1、为减小对柜台压力，该函数在股票交易模块中同一分钟内多次调用返回当前分钟首次查询的缓存数据；

2、该接口会返回当日截止到当前时间段内的成交数据；

参数

无

返回

返回数据：

一笔订单对应一笔成交:{'订单编号':[[成交编号,委托编号，标的代码，买卖类型，成交数量，成交价格，成交金额，成交时间]]}，如下:

注意：标的代码尾缀为四位，上证为 XSHG，深圳为 XSHE，如需对应到代码请做代码尾缀兼容

```
{'ba6a80d9746347a99c050b29069807c7': [[5001, 700001, '600570.XSHG', '买', 100000, 86.60, 8660000.0, datetime.datetime(2021, 7, 13, 15, 0)]]}
```

一笔订单对应多笔成交:{'订单编号':[[成交编号 1,委托编号，标的代码，买卖类型,成交数量,成交价格,成交金额,成交时间],[成交编号 2,委托编号，

标的代码，买卖类型,成交数量,成交价格,成交金额,成交时

间]]}(dict[str:list[list[int,int,str,str,int,int,numpy.float64,numpy.float64,datetime.datetime]]])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#获取当日成交数据 trades_info = get_trades()log.info(trades_info)
```

get_position - 获取持仓信息

```
get_position(security)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

该接口用于获取某个标的持仓信息详情。

注意事项：

无

参数

security：标的代码，如'600570.SS'，不传时默认为获取所有持仓(str)；

返回

返回一个 Position 对象(Position)。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):position = get_position(g.security)
log.info(position)
```

融资融券专用函数

融资融券交易类函数

margin_trade - 担保品买卖

```
margin_trade(security, amount, limit_price=None, market_type=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融回测、两融交易模块可用。

接口说明

该接口用于担保品买卖。

注意事项：

限价和委托类型都不传时默认取当前最新价进行委托；限价和委托类型都传入时以 market_type 为准

参数

security：股票代码(str)；

amount：交易数量，正数表示买入，负数表示卖出(int)；

limit_price：买卖限价(float)；

market_type：市价委托类型，上证非科创板股票支持参数 1、4，上证科创板股票支持参数 0、1、2、4，深证股票支持参数 0、2、3、4、5(int)；

- 0：对手方最优价格；
- 1：最优五档即时成交剩余转限价；
- 2：本方最优价格；
- 3：即时成交剩余撤销；
- 4：最优五档即时成交剩余撤销；
- 5：全额成交或撤单；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 以系统最新价委托 margin_trade(g.security, 100)# 以 72 块价格下一个限价单 margin_trade(g.security, 100, limit_price=72)# 以最优五档即时成交剩余撤销委托 margin_trade(g.security, 200, market_type=4)
```

margincash_open - 融资买入

```
margincash_open(security, amount, limit_price=None, market_type=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于融资买入。

注意事项：

限价和委托类型都不传时默认取当前最新价进行委托；限价和委托类型都传入时以 market_type 为准

参数

security：股票代码(str)；

amount：交易数量，输入正数(int)；

limit_price：买卖限价(float)；

market_type：市价委托类型，上证非科创板股票支持参数 1、4，上证科创板股票支持参数 0、1、2、4，深证股票支持参数 0、2、3、4、5(int)；

- 0：对手方最优价格；
- 1：最优五档即时成交剩余转限价；
- 2：本方最优价格；
- 3：即时成交剩余撤销；
- 4：最优五档即时成交剩余撤销；
- 5：全额成交或撤单；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 以 72 块价格下一个限价单 margincash_open(g.security, 100, limit_price=72)# 以对手方最优价格委托 margincash_open(g.security, 200, market_type=1)
```

margincash_close - 卖券还款

```
margincash_close(security, amount, limit_price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于卖券还款。

注意事项：

限价和委托类型都不传时默认取当前最新价进行委托；限价和委托类型都传入时以 market_type 为准

参数

security：股票代码(str)；

amount：交易数量，输入正数(int)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 卖 100 股还款 margincash_close(security, 100)
```

margincash_direct_refund - 直接还款

```
margincash_direct_refund(value)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于直接还款。

注意事项：

无

参数

value：还款金额(float)；

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取负债总额 fin_compact_balance = get_margin_assert().get('fin_compact_balance')# 还款 margincash_direct_refund(fin_compact_balance)
```

marginsec_open - 融券卖出

```
marginsec_open(security, amount, limit_price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于融券卖出。

注意事项：

限价和委托类型都不传时默认取当前最新价进行委托；限价和委托类型都传入时以 market_type 为准

参数

security：股票代码(str)；

amount：交易数量，输入正数(int)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 融券卖出 100 股 marginsec_open(security, 100)
```

marginsec_close - 买券还券

```
marginsec_close(security, amount, limit_price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于买券还券。

注意事项：

限价和委托类型都不传时默认取当前最新价进行委托；限价和委托类型都传入时以 market_type 为准

参数

security：股票代码(str)；

amount：交易数量，输入正数(int)；

limit_price：买卖限价(float)；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None(str)

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
#买 100 股还券 marginsec_close(security, 100)
```

marginsec_direct_refund - 直接还券

```
marginsec_direct_refund(security, amount)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于直接还券。

注意事项：

无

参数

security：股票代码(str)；

amount：交易数量，输入正数(int)；

返回

None

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
#买 100 股 marginsec_direct_refund(security, 100)
```

融资融券查询类函数

get_margincash_stocks - 获取融资标的列表

```
get_margincash_stocks()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于获取融资标的。

注意事项：

无

参数

无

返回

返回上交所、深交所最近一次披露的的可融资标的列表的 list(list[str,...])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取最新的融资标的列表 margincash_s
```

```
tocks = get_margincash_stocks()log.info(margincash_stocks)
```

get_marginsec_stocks - 获取融券标的列表

```
get_marginsec_stocks()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于获取融券标的。

注意事项：

无

参数

无

返回

返回上交所、深交所最近一次披露的的可融券标的列表的 list(list[str,...])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取最新的融券标的列表 marginsec_st
ocks = get_marginsec_stocks()log.info(marginsec_stocks)
```

get_margin_contract - 合约查询

```
get_margin_contract()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于合约查询。

注意事项：

无

参数

无

返回

正常返回一个 DataFrame 类型字段，columns 为每个合约所包含的信息，异常返回 None

合约包含以下信息：

- open_date:开户日期(str:int)；
- compact_id:合约编号(str:str)；
- stock_code:证券代码(str:str)；
- entrust_no:委托编号(str:int)；
- entrust_price:委托价格(str:float)；
- entrust_amount:委托数量(str:float)；
- business_amount:成交数量(str:float)；
- business_balance:成交金额(str:float)；
- compact_type:合约类别(str:str)；
- compact_status:合约状态(str:str)；

- repaid_interest:已还利息(str:float)；
- repaid_amount:已还数量(str:float)；
- repaid_balance:已还金额(str:float)；
- used_bail_balance:已用保证金(str:float)；
- ret_end_date:归还截止日(str:int)；
- date_clear:清算日期(str:int)；
- fin_income:融资合约盈亏(str:float)；
- slo_income:融券合约盈亏(str:float)；
- total_debit:负债总额(str:float)；
- compact_interest:合约利息金额(str:float)；
- real_compact_interest:日间实时利息金额(str:float)；
- real_compact_balance:日间实时合约金额(str:float)；
- real_compact_amount:日间实时合约数量(str:float)；

字段备注:

- compact_type -- 合约类别，0-融资，1-融券，2-其他负债；；

- compact_status -- 合约状态；
    - 0 -- 开仓未归还；
    - 1 -- 部分归还；
    - 2 -- 合约已过期；
    - 3 -- 客户自行归还；
    - 4 -- 手工了结；
    - 5 -- 未形成负债；

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取最新合约 df = get_margin_contr
act()log.info(df)
```

get_margin_contractreal - 实时合约流水查询

```
get_margin_contractreal()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于实时合约流水查询。

注意事项：

无

参数

无

返回

正常返回一个 DataFrame 类型字段，columns 为每个合约所包含的信息，异常返回 None

实时合约流水包含以下信息：

- init_date:交易日期(str:int)；
- compact_id:合约编号(str:str)；
- client_id:客户编号(str:str)；

- money_type:币种类别(str:str)；
- market_type:证券市场(str:str)；
- entrust_no:委托编号(str:int)；
- compact_type:合约类别(str:str)；
- stock_code:证券代码(str:str)；
- business_flag:业务标志(str:int)；
- occur_balance:发生金额(str:float)；
- post_balance:后资金额(str:float)；
- occur_amount:发生数量(str:float)；
- post_amount:后证券额(str:float)；
- occur_fare:发生费用(str:float)；
- post_fare:后余费用(str:float)；
- occur_interest:发生利息(str:float)；
- post_interest:后余利息(str:float)；
- remark:备注(str:str)；

字段备注:

- compact_type -- 合约类别，0-融资，1-融券，2-其他负债;;

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取实时流水 df = get_margin_contr
actreal()log.info(df)
```

get_margin_assert - 信用资产查询

```
get_margin_assert()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于信用资产查询。

注意事项:

无

参数

无

返回

正常返回一个 dict 类型字段，包含所有信用资产信息。异常返回空 dict，如{}(dict[str:float,...])

实时合约流水包含以下信息：

- assure_asset:担保资产(str:float)；
- total_debit:负债总额(str:float)；
- enable_bail_balance:可用保证金(str:float)；
- assure_enbuy_balance:买担保品可用资金(str:float)；
- fin_enrepaid_balance:现金还款可用资金(str:float)；
- fin_max_quota:融资额度上限(str:float)；
- fin_enable_quota:融资可用额度(str:float)；
- fin_used_quota:融资已用额度(str:float)；
- fin_compact_balance:融资合约金额(str:float)；

- fin_compact_fare:融资合约费用(str:float)；
- fin_compact_interest:融资合约利息(str:float)；
- slo_enable_quota:融券可用额度(str:float)；
- slo_compact_fare:融券合约费用(str:float)；
- slo_compact_interest:融券合约利息(str:float)；

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取信用账户资产信息 margin_assert
= get_margin_assert()log.info(margin_assert)
```

get_assure_security_list - 担保券查询

```
get_assure_security_list()
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于担保券查询。

注意事项：

无

参数

无

返回

返回上交所、深交所最近一次披露的担保券列表的 list(list[str,...])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 获取最新的担保券列表 assure_securit
y = get_assure_security_list()log.info(assure_security)
```

get_margincash_open_amount - 融资标的最大可买数量查询

```
get_margincash_open_amount(security, price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于融资标的最大可买数量查询。

注意事项：

无

参数

security：股票代码(str)；

price：限定价格(float)；

返回

正常返回一个 dict 类型对象，key 为股票代码，values 为最大数量。异常返回空 dict，如{}(dict[str:int])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 查询恒生电子最大可融资买入数量 margincash_open_dict = get_margincash_open_amount(security)if margincash_open_dict is not None:log.info(marg
incash_open_dict.get(security))
```

get_margincash_close_amount - 卖券还款标的最大可卖数量查询

```
get_margincash_close_amount(security, price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于卖券还款标的最大可卖数量查询。

注意事项：

无

参数

security：股票代码(str)；

price：限定价格(float)；

返回

正常返回一个 dict 类型对象，key 为股票代码，values 为最大数量。异常返回空 dict，如{}(dict[str:int])

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 查询恒生电子最大可卖券还款数量 margincash_close_dict = get_margincash_close_amount(security)if margincash_close_dict is not None:log.info(m
argincash_close_dict.get(security))
```

get_marginsec_open_amount - 融券标的最大可卖数量查询

```
get_marginsec_open_amount(security, price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于融券标的最大可卖数量查询。

注意事项：

无

参数

security：股票代码(str)；

price：限定价格(float)；

返回

正常返回一个 dict 类型对象，key 为股票代码，values 为最大数量。异常返回空 dict，如{}(dict[str:int])

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 查询中信证券最大可融券卖出数量 marginsec_open_dict = get_marginsec_open_amount(security)if marginsec_open_dict is not None:log.info(margins
ec_open_dict.get(security))
```

get_marginsec_close_amount - 买券还券标的最大可买数量查询

```
get_marginsec_close_amount(security, price=None)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于买券还券标的最大可买数量查询。

注意事项：

无

参数

security：股票代码(str)；

price：限定价格(float)；

返回

正常返回一个 dict 类型对象，key 为股票代码，values 为最大数量。异常返回空 dict，如{}(dict[str:int])

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 查询中信证券最大可买券还券数量 marginsec_close_dict = get_marginsec_close_amount(security)if marginsec_close_dict is not None:log.info(marg
insec_close_dict.get(security))
```

get_margin_entrans_amount - 现券还券数量查询

```
get_margin_entrans_amount(security)
```

使用场景

该函数仅支持 Ptrade 客户端可用，仅在两融交易模块可用。

接口说明

该接口用于现券还券数量查询。

注意事项：

无

参数

security：股票代码(str)；

返回

正常返回一个 dict 类型对象，key 为股票代码，values 为最大数量。异常返回空 dict，如{}(dict[str:int])

示例

```
def initialize(context):g.security = '600030.SS'set_universe(g.security)def handle_data(context, data):security = g.security
# 查询中信证券最大可现券还券数量 margin_entrans_dict = get_margin_entrans_amount(security)if margin_entrans_dict is not None:log.info(margin_
entrans_dict.get(security))
```

期货专用函数

期货交易类函数

buy_open - 多开

```
buy_open(contract, amount, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

买入开仓

注意:

不同期货品种每一跳的价格变动都不一样，limit_price 入参的时候要参考对应品种的价格变动规则，如 limit_price 不做入参则会以交易的行情快照最新价或者回测的分钟最新价进行报单；

根据交易所规则，每天结束时会取消所有未完成交易；

参数

contract：期货合约代码；

amount：交易数量，正数；

limit_price：买卖限价；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None。

示例

```
def initialize(context):g.security = ['IF1712.CCFX', 'CU1806.XSGE']set_universe(g.security)def handle_data(context, data):#买入开仓 buy_open('IF1712.CCFX', 1)#买入开仓(限定点数为 52220)buy_open('CU1806.XSGE', 1, limit_price=52220)
```

sell_close - 多平

```
sell_close(contract, amount, limit_price=None, close_today=False)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

卖出平仓

注意：

不同期货品种每一跳的价格变动都不一样，limit_price 入参的时候要参考对应品种的价格变动规则，如 limit_price 不做入参则会以交易的行情快照最新价或者回测的分钟最新价进行报单；

根据交易所规则，每天结束时会取消所有未完成交易；

参数

contract：期货合约代码；

amount：交易数量，正数；

limit_price：买卖限价；

close_today：平仓方式。close_today=False 为优先平昨仓，不足部分再平今仓；close_today=True 为仅平今仓，委托数量若大于今仓系统会调整为今仓数量。close_today=True 仅对上海期货交易所生效，其他交易所无需入参 close_today 字段，若设置为 True 系统会警告，并强行转换为 close_today=False。

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None。

示例

```
def initialize(context):g.security = ['IF1712.CCFX', 'CU1806.XSGE']set_universe(g.security)def handle_data(context, data):#卖出平仓 sell_close('IF1712.CCFX', 1)#卖出平今仓(限定点数为 52220)sell_close ('CU1806.XSGE', 1, limit_price=52220, close_today=True)#卖出平仓(限定点数为 52220)sell_close ('CU1806.XSGE', 1, limit_price=52220)
```

sell_open - 空开

```
sell_open(contract, amount, limit_price=None)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

卖出开仓

注意：

不同期货品种每一跳的价格变动都不一样，limit_price 入参的时候要参考对应品种的价格变动规则，如 limit_price 不做入参则会以交易的行情快照最新价或者回测的分钟最新价进行报单；

根据交易所规则，每天结束时会取消所有未完成交易；

参数

contract：期货合约代码；

amount：交易数量，正数；

limit_price：买卖限价；

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None。

示例

```
def initialize(context):g.security = ['IF1712.CCFX', 'CU1806.XSGE']set_universe(g.security)def handle_data(context, data):#卖出开仓 sell_open('IF1712.CCFX', 1)#卖出开仓(限定点数为 52220)sell_open ('CU1806.XSGE', 1, limit_price=52220)
```

buy_close - 空平

```
buy_close(contract, amount, limit_price=None, close_today=False)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

买入平仓

注意:

不同期货品种每一跳的价格变动都不一样，limit_price 入参的时候要参考对应品种的价格变动规则，如 limit_price 不做入参则会以交易的行情快照最新价或者回测的分钟最新价进行报单;

根据交易所规则，每天结束时会取消所有未完成交易;

参数

contract：期货合约代码；

amount：交易数量，正数；

limit_price：买卖限价；

close_today：平仓方式。close_today=False 为优先平昨仓，不足部分再平今仓；close_today=True 为仅平今仓，委托数量若大于今仓系统会调整为今仓数量。close_today=True 仅对上海期货交易所生效，其他交易所无需入参 close_today 字段，若设置为 True 系统会警告，并强行转换为 close_today=False。

返回

Order 对象中的 id 或者 None。如果创建订单成功，则返回 Order 对象的 id，失败则返回 None。

示例

```
def initialize(context):g.security = ['IF1712.CCFX', 'CU1806.XSGE']set_universe(g.security)def handle_data(context, data):#买入平仓 buy_close('IF1712.CCFX', 1)#买入平今仓(限定点数为 52220)buy_close ('CU1806.XSGE', 1, limit_price=52220, close_today=False)#买入平仓(限定点数为 52220)buy_close ('CU1806.XSGE', 1, limit_price=52220)
```

期货查询类函数

get_margin_rate- 获取用户设置的保证金比例

```
get_margin_rate(transaction_code)
```

使用场景

该函数仅在回测模块可用

接口说明

获取用户设置的保证金比例

参数

transaction_code：期货合约的交易代码，str 类型，如沪铜 2112（"CU2112"）的交易代码为"CU"；

返回

用户设置的保证金比例，float 浮点型数据，默认返回交易所设定的保证金比例；

示例

```
def initialize(context):g.security = "CU2112.XSGE"set_universe(g.security)# 设置沪铜品种的保证金比例为 8%set_margin_rate("CU", 0.08)def before_trading_start(context, data):# 获取沪铜品种的保证金比例 margin_rate = get_margin_rate("CU")log.info(margin_rate)# 获取苹果品种的保证金比例 margin_rate = get_margin_rate("AP")log.info(margin_rate)def handle_data(context, data):pass
```

get_instruments- 获取合约信息

```
get_instruments(contract)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

获取合约的上市的具体信息

参数

contract：字符串，期货的合约代码，str 类型；

返回

FutureParams 对象，主要返回的字段为:

- contract_code -- 合约代码，str 类型；
- contract_name -- 合约名称，str 类型；
- exchange -- 交易所：大商所、郑商所、上期所、中金所，str 类型；
- trade_unit -- 交易单位，int 类型；
- contract_multiplier -- 合约乘数，float 类型；
- delivery_date -- 交割日期，str 类型；
- listing_date -- 上市日期，str 类型；
- trade_code -- 交易代码，str 类型；
- margin_rate -- 保证金比例，float 类型；

注意:

期货实盘模块中，由于行情源的限制，涨跌幅目前暂无法提供

示例

```
def initialize(context):g.security = ["CU2112.XSGE", "IF2112.CCFX"]set_universe(g.security)def before_trading_start(context, data):# 获取股票池代码合约信息 for security in g.security:info = get_instruments(security)log.info(info)def handle_data(context, data):pass
```

期货设置类函数

set_future_commission - 设置期货手续费

```
set_future_commission(transaction_code, commission)
```

使用场景

该函数仅在回测模块可用

接口说明

设置期货手续费，手续费是按照交易代码进行设置的

参数

transaction_code：期货合约的交易代码，str 类型，如沪铜 2112（"CU2112"）的交易代码为"CU"；

commission：手续费，浮点型，设置说明：

- 当交易时的手续费是按手数收取时，则这里应当设置为每手收取的金额，例如：将期货的手续费设置为 2 元/手，此处应填写 2；
- 当交易时的手续费是按总成交额收取时，则这里应当设置为总成交额的比例，例如：将期货的手续费费率设置为 0.4/万，此处应填写

0.00004;

返回

None

示例

```
def initialize(context):g.security = "CU2112.XSGE"set_universe(g.security)# 设置沪铜的手续费，0.4/万 set_future_commission("CU", 0.00004)# 设置沪金的手续费，2 元/手 set_future_commission("AU", 2)def handle_data(context, data):# 买入沪铜 2112buy_open(g.security, 2)# 买入沪金 2112buy_open("AU2112.XSGE", 20)
```

set_margin_rate - 设置期货保证金比例

```
set_margin_rate(transaction_code, margin_rate)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

设置期货收取的保证金比例，保证金比例是按照交易代码进行设置的

参数

transaction_code：期货合约的交易代码，str 类型，如沪铜 2112（"CU2112"）的交易代码为"CU"；

margin_rate：保证金比例，浮点型，将对应期货的保证金比例设置为 5%则输入 0.05；

返回

None

示例

```
def initialize(context):g.security = "CU2112.XSGE"set_universe(g.security)# 设置沪铜品种收取的保证金比例设置为 5%set_margin_rate("CU", 0.05)d
ef handle_data(context, data):# 买入沪铜 2112buy_open(g.security, 10)
```

计算函数

技术指标计算函数

get_MACD - 异同移动平均线

```
get_MACD(close, short=12, long=26, m=9)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

获取异同移动平均线 MACD 指标的计算结果

参数

close：价格的时间序列数据, numpy.ndarray 类型；

short：短周期, int 类型；

long：长周期, int 类型；

m：移动平均线的周期, int 类型；

返回

MACD 指标 dif 值的时间序列, numpy.ndarray 类型

MACD 指标 dea 值的时间序列, numpy.ndarray 类型

MACD 指标 macd 值的时间序列, numpy.ndarray 类型

示例

```
def initialize(context):g.security = "600570.XSHG"set_universe(g.security)def handle_data(context, data):h = get_history(100, '1d', ['clos
e','high','low'], security_list=g.security)close_data = h['close'].values
macdDIF_data, macdDEA_data, macd_data = get_MACD(close_data, 12, 26, 9)dif = macdDIF_data[-1]dea = macdDEA_data[-1]macd = macd_data[-1]
```

get_KDJ - 随机指标

```
get_KDJ(high, low, close, n=9, m1=3, m2=3)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

获取随机指标 KDJ 指标的计算结果

参数

high：最高价的时间序列数据, numpy.ndarray 类型；

low：最低价的时间序列数据, numpy.ndarray 类型；

close：收盘价的时间序列数据, numpy.ndarray 类型；

n: 周期, int 类型；

m1: 参数 m1, int 类型；

m2: 参数 m2, int 类型；

返回

KDJ 指标 k 值的时间序列, numpy.ndarray 类型

KDJ 指标 d 值的时间序列, numpy.ndarray 类型

KDJ 指标 j 值的时间序列, numpy.ndarray 类型

示例

```
def initialize(context):g.security = "600570.XSHG"set_universe(g.security)def handle_data(context, data):h = get_history(100, '1d', ['clos
e','high','low'], security_list=g.security)high_data = h['high'].values
low_data = h['low'].values
close_data = h['close'].values
```

```
k_data, d_data, j_data = get_KDJ(high_data, low_data, close_data, 9, 3, 3)k = k_data[-1]d = d_data[-1]j = j_data[-1]
```

get_RSI - 相对强弱指标

```
get_RSI(close, n=6)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

获取相对强弱指标 RSI 指标的计算结果

参数

close：价格的时间序列数据, numpy.ndarray 类型；

n: 周期, int 类型；

返回

RSI 指标 rsi 值的时间序列, numpy.ndarray 类型

示例

```
def initialize(context):g.security = "600570.XSHG"set_universe(g.security)def handle_data(context, data):h = get_history(100, '1d', ['clos
e','high','low'], security_list=g.security)close_data = h['close'].values
rsi_data = get_RSI(close_data, 6)rsi = rsi_data[-1]
```

get_CCI - 顺势指标

```
get_CCI(close, n=14)
```

使用场景

该函数仅在回测、交易模块可用

接口说明

获取顺势指标 CCI 指标的计算结果

参数

high：最高价的时间序列数据, numpy.ndarray 类型；

low：最低价的时间序列数据, numpy.ndarray 类型；

close：收盘价的时间序列数据, numpy.ndarray 类型；

n: 周期, int 类型；

返回

CCI 指标 cci 值的时间序列, numpy.ndarray 类型

示例

```
def initialize(context):g.security = "600570.XSHG"set_universe(g.security)def handle_data(context, data):h = get_history(100, '1d', ['clos
e','high','low'], security_list=g.security)high_data = h['high'].values
low_data = h['low'].values
close_data = h['close'].values
cci_data = get_CCI(high_data, low_data, close_data, 14)cci = cci_data[-1]
```

其他函数

log-日志记录

```
log(content)
```

使用场景

该函数仅在回测、交易模块可用。

接口说明

该接口用于打印日志。

支持如下场景的日志记录：

```
log.debug("debug")log.info("info")log.warning("warning")log.error("error")log.critical("critical")
```

与 python 的 logging 模块用法一致

注意事项：

无

参数

参数可以是字符串、对象等。

返回

None

示例

```
# 打印出一个格式化后的字符串 g.security='600570.SS'log.info("Selling %s, amount=%s" % (g.security, 10000))
```

is_trade-业务代码场景判断

```
is_trade()
```

使用场景

该函数仅在回测、交易模块可用。

接口说明

该接口用于提供业务代码执行场景判断依据，明确标识当前业务代码运行场景为回测还是交易。因部分函数仅限回测或交易场景使用，该函数可以协助区分对应场景，以便限制函数可以在一套策略代码同时兼容回测与交易场景。

注意事项：

无

参数

无

返回

布尔类型，当前代码在交易中运行返回 True，当前代码在回测中运行返回 False(bool)。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):_id = order(g.security, 100)if is_trade():log.info("当前运行场景：交易")else:log.info("当前运行场景：回测")
```

check_limit - 代码涨跌停状态判断

```
check_limit(security)
```

使用场景

该函数仅在交易模块可用。

接口说明

该接口用于标识当日股票的涨跌停情况。

注意事项：

无

参数

security：单只股票代码或者多只股票代码组成的列表，必填字段(list[str]/str)；

返回

正常返回一个 dict 类型数据，包含每只股票代码的涨停状态。多只股票代码查询时其中部分股票代码查询异常则该代码返回既不涨停也不跌停状态 0。(dict[str:int])

涨跌停状态说明：

- 2：触板涨停（已经是涨停价格，但还有卖盘）；
- 1：涨停；
- 0：既不涨停也不跌停；
- -1：跌停；
- -2：触板跌停（已经是跌停价格，但还有买盘）；

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):# 代码涨跌停状态 stock_flag = check_l
imit(g.security)log.info(stock_flag)
```

send_email - 发送邮箱信息

```
send_email(send_email_info, get_email_info, smtp_code, info='', path='', subject='')
```

使用场景

该函数仅在交易模块可用。

接口说明

该接口用于通过 QQ 邮箱发送邮件内容。

注意事项：

1、该接口需要服务端连通外网，是否开通由所在券商决定

2、是否允许发送附件（即 path 参数），由所在券商的配置管理决定

3、邮件中接受到的附件为文件名而非附件路径

参数

send_email_info：发送方的邮箱地址，必填字段，如:50xxx00@qq.com(str)；

get_email_info：接收方的邮箱地址，必填字段，如:[50xxx00@qq.com, 1xxx10@126.com](list[str]/str)；

smtp_code：邮箱的 smtp 授权码，注意，不是邮箱密码，必填字段(str)；

info：发送内容，选填字段，默认空字符串(str)；

path：附件路径，选填字段，如:'/home/fly/notebook/stock.csv'，默认空字符串(str)；

subject：邮件主题，默认空字符串(str)；

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#发送文字信息 send_email('53xxxxxx7@qq.com', ['53xxxxx7@qq.com', 'Kxxxxn@126.com'], 'phfxxxxxxxxxxcd', info='今天的股票池信息')
```

send_qywx - 发送企业微信信息

```
send_qywx(corp_id, secret, agent_id, info='', path='', toparty='', touser= '', totag= '')
```

使用场景

该函数仅在交易模块可用。

接口说明

该接口用于通过企业微信发送内容，使用方法请查看 企业微信功能使用手册。

注意事项：

1、该接口需要服务端连通外网，是否开通由所在券商决定

2、是否允许发送文件（即 path 参数），由所在券商的配置管理决定

3、企业微信不能同时发送文字和文件，当同时入参 info 和 path 的时候，默认发送文件

4、企业微信接受到的文件为文件名而非文件路径

参数

corp_id：企业 ID，必填字段(str)；

secret：企业微信应用的密码，必填字段(str)；

agent_id：企业微信应用的 ID，必填字段(str)；

info：发送内容，选填字段，默认空字符串(str)；

path：发送文件，选填字段，如:'/home/fly/notebook/stock.csv'，默认空字符串(str)；

toparty：发送对象为部门，选填字段，默认空字符串(str)，多个对象之间用 '|' 符号分割；

touser：发送内容为个人，选填字段，默认空字符串(str)，多个对象之间用 '|' 符号分割；

totag：发送内容为分组，选填字段，默认空字符串(str)，多个对象之间用 '|' 符号分割；

注意：toparty、touser、totag 如果都不传入，接口默认发送至应用中设定的第一个 toparty

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):#发送文字信息 send_qywx('wwxxxxxxxxx
xxxf9', 'hixxxxxxxxxxxxxxxxxxBX8', '10xxxx3', info='已触发委托买入', toparty='1|2')
```

permission_test-权限校验

```
permission_test(account=None, end_date=None)
```

使用场景

该函数仅在交易模块可用

接口说明

该接口用于账号和有效期的权限校验，用户可以在接口中入参指定账号和指定有效期截止日，策略运行时会校验运行策略的账户与指定账户是否相符，以及运行当日日期是否超过指定的有效期截止日，任一条件校验失败，接口都会返回 False，两者同时校验成功则返回 True。校验失败会在策略日志中提示原因。

注意事项：

如果需要使用授权模式下载功能，不要在接口中入参，策略编码时候直接调用 permission_test()，授权工具会把需要授权的账号和有效期信息放到策略文件中。

该函数仅在 initialize、before_trading_start、after_trading_end 模块中支持调用

参数

account：授权账号，选填字段，如果不填就代表不需要验证账号(str)；

end_date：授权有效期截止日，选题字段，如果不填就代表不需要验证有效期(str)，日期格式必须为'YYYYmmdd'的 8 位日期格式，如'20200101'；

返回

布尔类型，校验成功返回 True，校验失败返回 False(bool)。

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):passdef after_trading_end(context, data):# 需要用授权模式下载功能的情况下不用入参 flag = permission_test()if not flag:raise RuntimeError('授权不通过，终止程序，抛出异常')# 不需要用授权模式下载功能的情况下通过入参来进行授权校验 flag = permission_test(account='10110922',end_date='20220101')if not flag:raise RuntimeError('授权不通过，终止程序，抛出异常')
```

create_dir-创建文件目录路径

```
create_dir(user_path=None)
```

使用场景

该函数在研究、回测、交易模块可用

接口说明

由于 ptrade 引擎禁用了 os 模块，因此用户无法在策略中通过编写代码实现子目录创建。用户可以通过此接口来创建文件的子目录路径。

注意事项：

文件根目录路径为'/home/fly/notebook'。

参数

user_path：子目录路径，选填字段，(str)。

比如 user_path='download'，会在研究中生成/home/fly/notebook/download 的目录；

比如 user_path='download/2022'，会在研究中生成/home/fly/notebook/download/2022 的目录；

返回

None

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)create_dir(user_path=g.security)def handle_data(context, data):pas
```

```
s
```

对象

g-全局对象

使用场景

该对象仅支持回测、交易模块。

对象说明

全局对象 g，用于存储用户的各类可被不同函数（包括自定义函数）调用的全局数据,如：

```
g.security = None #股票池
```

注意事项：

无

示例

```
def initialize(context):g.security = "600570.SS"g.count = 1g.flag = 0set_universe(g.security)def handle_data(context, data):log.info(g.sec
urity)log.info(g.count)log.info(g.flag)
```

Context-上下文对象

使用场景

该对象仅支持回测、交易模块。

对象说明

类型为业务上下文对象

注意事项：

无

内容：

```
capital_base -- 起始资金 previous_date -- 前一个交易日 sim_params -- SimulationParameters 对象 capital_base -- 起始资金 data_frequency -- 数据频率 portfolio -- 账户信息，可参考 Portfolio 对象 initialized -- 是否执行初始化 slippage -- 滑点，VolumeShareSlippage 对象 volume_limit -- 成交限量 price_impact -- 价格影响力 commission -- 佣金费用，Commission 对象 tax—印花税费率 cost—佣金费率 min_trade_cost—最小佣金 blotter -- Blotter 对象（记录）current_dt -- 当前单位时间的开始时间，datetime.datetime 对象（北京时间）recorded_vars -- 收益曲线值
```

示例

```
def initialize(context):g.security = ['600570.SS', '000001.SZ']set_universe(g.security)def handle_data(context, data):#获得当前回测相关时间
pre_date = context.previous_date
```

```
log.info(pre_date)year = context.blotter.current_dt.year
log.info(year)month = context.blotter.current_dt.month
log.info(month)day = context.blotter.current_dt.day
log.info(day)hour = context.blotter.current_dt.hour
log.info(hour)minute = context.blotter.current_dt.minute
log.info(minute)second = context.blotter.current_dt.second
log.info(second)#得到"年-月-日"格式 date = context.blotter.current_dt.strftime("%Y-%m-%d")log.info(date)#得到周几 weekday = context.blotter.c
urrent_dt.isoweekday()log.info(weekday)
```

SecurityUnitData

使用场景

该对象仅支持回测、交易模块。

对象说明

一个单位时间内的股票的数据，是一个字典，根据 sid 获取 BarData 对象数据

注意事项:

无

基本属性

以下属性也能通过 get_history/get_price 获取到

```
dt 时间 open 时间段开始时价格 close 时间段结束时价格 price 结束时价格 low 最低价 high 最高价 volume 成交的股票数量 money 成交的金额
```

Portfolio

使用场景

该对象仅支持回测、交易模块。

对象说明

对象数据包含账户当前的资金，标的信息，即所有标的操作仓位的信息汇总

注意事项：

无

内容

```
cash 当前可用资金（不包含冻结资金）positions 当前持有的标的(包含不可卖出的标的)，dict 类型，key 是标的代码，value 是 Position 对象 portfolio_value 当前持有的标的和现金的总价值 positions_value 持仓价值 capital_used 已使用的现金 returns 当前的收益比例，相对于初始资金 pnl 浮动盈亏 start_date 开始时间
```

示例

```
def initialize(context):g.security = "600570.SS"set_universe([g.security])def handle_data(context, data):log.info(context.portfolio.portfolio_value)
```

Position

使用场景

该对象仅支持回测、交易模块。

对象说明

持有的某个标的的信息。

注意事项：

无

内容

```
sid 标的代码 enable_amount 可用数量 amount 总持仓数量 last_sale_price 最新价格 cost_basis 持仓成本价格(期货不支持)today_amount 今日开仓数量(期货不支持，且仅回测有效)期货专用字段：delivery_date 交割日，期货使用 today_short_amount 空头今仓数量 today_long_amount 多头今仓数量 long_cost_basis 多头持仓成本 short_cost_basis 空头持仓成本 margin_rate 保证金比例 contract_multiplier 合约乘数 long_amount 多头总持仓量 short_amount 空头总持仓量 long_pnl 多头浮动盈亏 short_pnl 空头浮动盈亏 long_enable_amount 多头可用数量 short_enable_amount 多空头可用数量 business_type 业务类型
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order(g.security,1000)position = ge
t_position(g.security)log.info(position)
```

Order

使用场景

该对象仅支持回测、交易模块。

对象说明

买卖订单信息

注意事项:

无

内容

```
id -- 订单号 dt -- 订单产生时间 limit -- 指定价格 symbol -- 标的代码(备注：标的代码尾缀为四位，上证为 XSHG，深圳为 XSHE，如需对应到代码请做代码尾缀兼
容)amount -- 下单数量，买入是正数，卖出是负数 created -- 订单生成时间，datetime.date 对象 filled -- 成交数量，买入时为正数，卖出时为负数 entrust_no
 -- 委托编号 priceGear -- 盘口档位 status -- 订单状态(str)，该字段取值范围：'0' -- "未报"'1' -- "待报"'2' -- "已报"'3' -- "已报待撤"'4' -- "部成待
撤"'5' -- "部撤"'6' -- "已撤"'7' -- "部成"'8' -- "已成"'9' -- "废单"'+' -- "已受理"'-' -- "已确认"'V' -- "已确认"
```

示例

```
def initialize(context):g.security = '600570.SS'set_universe(g.security)def handle_data(context, data):order(g.security, 100)order_obj = g
et_orders()log.info(order_obj)
```

策略示例

集合竞价追涨停策略

```
def initialize(context):# 初始化此策略# 设置我们要操作的股票池, 这里我们只操作一支股票 g.security = '600570.SS'set_universe(g.security)#每天 9:2
3 分运行集合竞价处理函数 run_daily(context, aggregate_auction_func, time='9:23')

def aggregate_auction_func(context):stock = g.security
#最新价 snapshot = get_snapshot(stock)price = snapshot[stock]['last_px']#涨停价 up_limit = snapshot[stock]['up_px']#如果最新价不小于涨停价, 买
入 if float(price) >= float(up_limit):order(g.security, 100, limit_price=up_limit)def handle_data(context, data):pass
```

tick 级别均线策略

```
def initialize(context):# 初始化此策略# 设置我们要操作的股票池, 这里我们只操作一支股票 g.security = '600570.SS'set_universe(g.security)#每 3 秒运
行一次主函数 run_interval(context, func, seconds=3)#盘前准备历史数据 def before_trading_start(context, data):history = get_history(10, '1d',
'close', g.security, fq='pre', include=False)g.close_array = history['close'].values

#当五日均线高于十日均线时买入, 当五日均线低于十日均线时卖出 def func(context):stock = g.security

#获取最新价 snapshot = get_snapshot(stock)price = snapshot[stock]['last_px']# 得到五日均线价格 days = 5ma5 = get_MA_day(stock, days, g.close_
array[-4:], price)
    # 得到十日均线价格 days = 10ma10 = get_MA_day(stock, days, g.close_array[-9:], price)# 得到当前资金余额 cash = context.portfolio.cash
```

```
# 如果当前有余额，并且五日均线大于十日均线 if ma5 > ma10:# 用所有 cash 买入股票 order_value(stock, cash)# 记录这次买入 log.info("Buying %s" % (stock))# 如果五日均线小于十日均线，并且目前有头寸 elif ma5 < ma10 and get_position(stock).amount > 0:# 全部卖出 order_target(stock, 0)# 记录这次卖出 log.info("Selling %s" % (stock))

#计算实时均线函数 def get_MA_day(stock,days,close_array,current_price):close_sum = close_array[-(days-1):].sum()MA = (current_price + close_sum)/days
return MA
def handle_data(context, data):pass
```

双均线策略

```
def initialize(context):# 初始化此策略# 设置我们要操作的股票池，这里我们只操作一支股票 g.security = '600570.SS'set_universe(g.security)#当五日均线高于十日均线时买入，当五日均线低于十日均线时卖出 def handle_data(context, data):security = g.security
#得到十日历史价格 df = get_history(10, '1d', 'close', security, fq=None, include=False)# 得到五日均线价格 ma5 = round(df['close'][-5:].mean(), 3)# 得到十日均线价格 ma10 = round(df['close'][-10:].mean(), 3)# 取得昨天收盘价 price = data[security]['close']# 得到当前资金余额 cash = context.portfolio.cash

# 如果当前有余额，并且五日均线大于十日均线 if ma5 > ma10:# 用所有 cash 买入股票 order_value(security, cash)# 记录这次买入 log.info("Buying %s" % (security))# 如果五日均线小于十日均线，并且目前有头寸 elif ma5 < ma10 and get_position(security).amount > 0:# 全部卖出 order_target(security, 0)# 记录这次卖出 log.info("Selling %s" % (security))
```

融资融券双均线策略

```
def initialize(context):# 初始化策略# 设置我们要操作的股票池，这里我们只操作一支股票 g.security = "600570.SS"set_universe(g.security)def before_trading_start(context, data):# 买入标识 g.order_buy_flag = False# 卖出标识 g.order_sell_flag = False#当五日均线高于十日均线时买入，当五日均线低于十日均线时卖出 def handle_data(context, data):# 得到十日历史价格 df = get_history(10, "1d", "close", g.security, fq=None, include=False)# 得到五日均线价格 ma5 = round(df["close"][-5:].mean(), 3)# 得到十日均线价格 ma10 = round(df["close"][-10:].mean(), 3)# 取得昨天收盘价 price = data[g.security]["close"]# 如果五日均线大于十日均线 if ma5 > ma10:if not g.order_buy_flag:# 获取最大可融资数量 amount = get_margincash_open_amount
```

```
(g.security).get(g.security)# 进行融资买入操作 margincash_open(g.security, amount)# 记录这次操作 log.info("Buying %s Amount %s" % (g.securit
y, amount))# 当日已融资买入 g.order_buy_flag = True# 如果五日均线小于十日均线，并且目前有头寸 elif ma5 < ma10 and get_position(g.security).amoun
t > 0:if not g.order_sell_flag:# 获取标的卖券还款最大可卖数量 amount = get_margincash_close_amount(g.security).get(g.security)# 进行卖券还款操
作 margincash_close(g.security, -amount)# 记录这次操作 log.info("Selling %s Amount %s" % (g.security, amount))# 当日已卖券还款 g.order_sell_fl
ag = True
```

macd 策略

指数平滑均线函数，以 price 计算，可以选择收盘、开盘价等价格，N 为时间周期，m 用于计算平滑系数 a=m/(N+1)，EXPMA1 为前一日值

```
def f_expma(N,m,EXPMA1,price):a = m/(N+1)EXPMA2 = a * price + (1 - a)*EXPMA1
return EXPMA2 #2 为后一天值#定义 macd 函数，输入平滑系数参数、前一日值，输出当日值 def macd(N1,N2,N3,m,EXPMA12_1,EXPMA26_1,DEA1,price):EXPMA12_2
= f_expma(N1,m,EXPMA12_1,price)EXPMA26_2 = f_expma(N2,m,EXPMA26_1,price)DIF2 = EXPMA12_2 - EXPMA26_2
a = m/(N3+1)DEA2 = a * DIF2 + (1 - a)*DEA1
BAR2=2*(DIF2-DEA2)return EXPMA12_2,EXPMA26_2,DIF2,DEA2,BAR2
def initialize(context):global init_price
init_price = None# 获取沪深 300 股票 g.security = get_index_stocks('000300.SS')#g.security = ['600570.SS']# 设置我们要操作的股票池，这里我们只操
作一支股票 set_universe(g.security)def handle_data(context, data):# 获取历史数据，这里只获取了 2 天的数据，如果希望最终 MACD 指标结果更准确最好是获
取# 从股票上市至今的所有历史数据，即增加获取的天数 close_price = get_history(2, '1d', field='close', security_list=g.security)#如果是停牌不进行
计算 for security in g.security:if data[security].is_open >0:global init_price,EXPMA12_1,EXPMA26_1,EXPMA12_2,EXPMA26_2,DIF1,DIF2,DEA1,DEA2

if init_price is None:init_price = close_price[security].mean()#nan 和 N-1 个数，mean 为 N-1 个数的均值 EXPMA12_1 = init_price
EXPMA26_1 = init_price
DIF1 = init_price
DEA1 = init_price
# m 用于计算平滑系数 a=m/(N+1)m = 2.0#设定指数平滑基期数 N1 = 12N2 = 26N3 = 9EXPMA12_2,EXPMA26_2,DIF2,DEA2,BAR2 = macd(N1,N2,N3,m,EXPMA12_1,EXP
MA26_1,DEA1,close_price[security][-1])# 取得当前价格 current_price = data[security].price
# 取得当前的现金 cash = context.portfolio.cash
# DIF、DEA 均为正，DIF 向上突破 DEA，买入信号参考 if DIF2 > 0 and DEA2 > 0 and DIF1 < DEA1 and DIF2 > DEA2:# 计算可以买多少只股票 number_of_shares
```

```
 = int(cash/current_price)# 购买量大于 0 时，下单 if number_of_shares > 0:# 以市单价买入股票，日回测时即是开盘价 order(security, +number_of_share
s)# 记录这次买入 log.info("Buying %s" % (security))# DIF、DEA 均为负，DIF 向下突破 DEA，卖出信号参考 elif DIF2 < 0 and DEA2 < 0 and DIF1 > DEA1 a
nd DIF2 < DEA2 and get_position(security).amount > 0:# 卖出所有股票,使这只股票的最终持有量为 0order_target(security, 0)# 记录这次卖出 log.info
("Selling %s" % (security))# 将今日的值赋给全局变量作为下一次前一日的值 DEA1 = DEA2
DIF1 = DIF2
EXPMA12_1 = EXPMA12_2
EXPMA26_1 = EXPMA26_2
```

常见问题

使用本平台受阻，可参考常见问题说明

支持的三方库

序号	三方库名称	版本号
1	APScheduler	3.3.1
2	arch	3.2
3	bcolz	1.2.1
4	beautifulsoup4	4.6.0
5	bleach	1.5.0

序号	三方库名称	版本号
6	boto	2.43.0
7	Bottleneck	1.0.0
8	bz2file	0.98
9	cachetools	3.1.0
10	click	4.0
11	contextlib2	0.4.0
12	crypto	1.4.1
13	cvxopt	1.1.8
14	cx-Oracle	8.0.1
15	cycler	0.10.0
16	cyordereddict	0.2.2
17	Cython	0.22.1
18	decorator	4.0.10
19	entrypoints	0.2.2

序号	三方库名称	版本号
20	fastcache	1.0.2
21	gensim	0.13.3
22	h5py	2.6.0
23	hmmlearn	0.2.0
24	html5lib	0.9999999
25	ipykernel	4.5.0
26	ipython	5.1.0
27	ipython-genutils	0.1.0
28	ipywidgets	5.2.2
29	jieba	0.38
30	Jinja2	2.8
31	jsonpickle	1.0
32	jsonschema	2.5.1
33	jupyter	1.0.0

序号	三方库名称	版本号
34	jupyter-client	4.4.0
35	jupyter-console	5.0.0
36	jupyter-core	4.2.0
37	jupyter-kernel-gateway	1.1.1
38	Keras	2.3.1
39	Keras-Applications	1.0.8
40	Keras-Preprocessing	1.1.0
41	line-profiler	2.1.2
42	Logbook	1.4.3
43	lxml	4.5.0
44	Markdown	2.2.0
45	MarkupSafe	0.23
46	matplotlib	1.5.3
47	mistune	0.7.3

序号	三方库名称	版本号
48	Naked	0.1.31
49	nbconvert	4.2.0
50	nbformat	4.1.0
51	networkx	1.9.1
52	nose	1.3.6
53	notebook	4.2.3
54	numexpr	2.6.1
55	numpy	1.11.2
56	pandas	0.23.4
57	patsy	0.4.0
58	pexpect	4.2.1
59	pickleshare	0.7.4
60	pip	9.0.1
61	pkgconfig	1.0.0

序号	三方库名称	版本号
62	prompt-toolkit	1.0.8
63	protobuf	3.3.0
64	ptvsd	2.2.0
65	ptyprocess	0.5.1
66	PyBrain	0.3
67	pycrypto	2.6.1
68	Pygments	2.1.3
69	PyMySQL	0.9.3
70	pyparsing	2.1.10
71	python-dateutil	2.7.5
72	pytz	2015.4
73	PyWavelets	0.4.0
74	PyYAML	3.13
75	pyzmq	16.1.0.dev0

序号	三方库名称	版本号
76	qtconsole	4.2.1
77	requests	2.7.0
78	retrying	1.3.3
79	scikit-learn	0.18
80	scipy	0.18.0
81	seaborn	0.7.1
82	setuptools	28.7.1
83	setuptools-scm	3.1.0
84	shellescape	3.4.1
85	simplegeneric	0.8.1
86	simplejson	3.17.0
87	six	1.10.0
88	sklearn	0.0
89	smart-open	1.3.5

序号	三方库名称	版本号
90	SQLAlchemy	1.0.8
91	statsmodels	0.10.2
92	TA-Lib	0.4.10
93	tables	3.3.0
94	tabulate	0.7.5
95	tensorflow	1.3.0rc1
96	tensorflow-tensorboard	0.1.2
97	terminado	0.6
98	Theano	0.8.2
99	toolz	0.7.4
100	tornado	4.4.2
101	traitlets	4.3.1
102	tushare	1.2.48
103	tzlocal	1.3

序号	三方库名称	版本号
104	wcwidth	0.1.7
105	Werkzeug	0.12.2
106	wheel	0.29.0
107	widgetsnbextension	1.2.6
108	xcsc-tushare	1.0.0
109	xgboost	0.6a2
110	xlrd	1.1.0
111	xlwt	1.3.0
112	zipline	0.8.3

接口版本变动

当前版本：PBOXQT1.0V202202.00.000

变动版本	变动内容

变动版本	变动内容
PBOXQT1.0V202101.03.000	1. on_order_response()主推信息中新增 entrust_type、entrust_prop 字段； 2. 修复信用交易接口兼容问题； 3. get_price()、get_history()支持周频(1w)行情获取； 4. 由于行情源不再更新维护，get_fundamentals()接口去除 share_change 表；
PBOXQT1.0V202101.04.000	1. 修复 get_all_orders()获取特定委托状态报错问题，status 字段返回数据类型从 int 改为 str； 2. on_order_response()、on_trade_response()支持获取非本策略交易的主推信息(需券商配置默认不推送)，且 on_order_response 推送非本策略交易的主推信息时不包含 order_id 字段； 3. 相关功能优化；
PBOXQT1.0V202101.05.000	1. 信用账户支持 ipo_stocks_order()接口调用； 2. 由于行情源返回信息不包含，get_fundamentals()获取 growth_ability、profit_ability、eps、operating_ability、debt_paying_ability 表不再返回 company_type 字段； 3. 由于上交所债券业务规则变更，调用 debt_to_stock_order()接口对上海市场可转债进行转股操作时需传入可转债代码，不再传入转股代码；
PBOXQT1.0V202101.06.000	1. 新增 get_user_name()API 接口，用于获取登录终端的资金账号；

变动版本	变动内容
	2. 研究中 get_price()新增支持获取周线数据； 3. get_snapshot()新增支持获取 XBHS 行业版块市场数据； 4. send_qywx()新增 toparty(发送对象为部门)、touser(发送内容为个人)、totag(发送内容为分组)入参；
PBOXQT1.0V202101.07.000	1. get_snapshot()新增 wavg_px(加权平均价)、px_exchange_rate(涨跌幅)出参； 2. 可转债回测业务新增支持 T+0； 3. 新增支持融资融券回测业务，融资融券专用函数中暂只支持 margin_trade()接口；
PBOXQT1.0V202101.08.000	1. initialize 对部分 API 接口调用进行限制，仅 initialize 可调用接口说明中的 API 可在 initialize 函数内使用； 2. before_trading_start 和 after_trading_end 对两融委托 API 接口调用进行限制； 3. 修复仅单笔成交订单时调用 get_trades()返回格式有误问题； 4. 修复交易场景中获取当日 K 线 14:58、14:59 分价格为 0 问题； 5. send_email()发送邮件信息新增 path(附件路径)、subject(邮件主题)入参； 6. 新增 get_cb_list()获取可转债列表； 7. 新增 get_deliver()获取历史交割单信息； 8. 新增 get_fundjour()获取历史资金流水信息；

变动版本	变动内容
	9. 新增 get_research_path()获取研究路径； 10. get_market_detail()新增支持在回测、交易场景中调用；
PBOXQT1.0V202101.09.000	1. get_market_detail()限制仅 before_trading_start 和 after_trading_end 中使用； 2. get_snapshot()新增返回 hsTimeStamp(快照时间戳)字段；对接 L2 行情买卖一档新增返回委托队列； 3. ipo_stocks_order()新增 black_stocks(新股/债黑名单)入参； 4. on_order_response()新增返回 error_info(错误信息)字段；
PBOXQT1.0V202201.00.000	1. get_individual_transcation()新增返回 buy_no(叫买方编号)、sell_no(叫卖方编号)、trans_flag(成交标记)、trans_identify_am(盘后逐笔成交序号标识)、channel_num(成交通道信息)字段； 2. get_margin_contract()新增返回 compact_interest(合约利息金额)、real_compact_interest(日间实时利息金额)、real_compact_balance(日间实时合约金额)、real_compact_amount(日间实时合约数量)字段； 3. get_price()、get_history()新增支持：1 月(mo)、1 季度(1q)、1 年(1y)频率行情获取； 4. set_commission()中 type 字段新增支持传入"LOF"类型； 5. get_individual_entrust()和 get_individual_transcation()返回内容中 hq_px 字段值缩小 1000 倍，返回为真实价格； 6. 新增支持期货日盘回测功能、期货日盘交易功能(对接 UFT 柜台)，期货 API 接口详见量化帮助文档期货专用函数模块；

变动版本	变动内容
	7. 新增 get_tick_direction()获取分时成交行情； 8. 新增 get_sort_msg()获取版块、行业的涨幅排名； 9. 新增 permission_test()权限校验；
PBOXQT1.0V202201.01.000	1. 修复委托状态类型不一致问题，get_orders()、get_all_orders()以及 Order 对象中的委托状态字段数据类型从 int 统一为 str； 2. 新增 get_trade_name()获取交易名称； 3. tick_data 中可调用接口完善； 4. 研究中 get_stock_name()、get_stock_info()新增支持获取可转债、ETF、LOF 品种； 5. get_history()新增 fill(填充类型)入参； 6. get_price()、get_history()新增支持：5 分钟(5m)、15 分钟(15m)、30 分钟(30m)、60 分钟(60m)、120 分钟(120m)频率行情获取；
PBOXQT1.0V202202.00.000	1. log 新增支持 DEBUG 级别日志记录； 2. get_price()、get_history()新增返回 preclose(昨收盘价)、high_limit(涨停价)、low_limit(跌停价)、unlimited(是否无涨跌停限制)字段； 3. get_snapshot()新增返回 total_bidqty(委买量)、total_offerqty(委卖量)、total_bid_turnover(委买金额)、total_offer_turnover(委卖金额)字段；

变动版本	变动内容
	4. on_trade_response()新增返回 order_id(Order 订单编号)字段； 5. 当接到策略外交易产生的主推时(需券商配置默认不推送)，由于没有对应的 Order 对象，on_order_response()、on_trade_response()中 order_id 字段赋值为""； 6. on_trade_response()新增接收撤单的成交主推，详见接口说明注意事项； 7. tick_data 中可调用接口完善； 8. 弃用 set_close_position_type()(设置期货平仓方式)、get_close_position_type()(获取期货平仓方式)API 接口； 9. 期货 Position 对象中删除 close_position_type(平仓方式)字段； 10. sell_close()、buy_close()新增 close_today(平仓方式)入参； 11. 新增 get_MACD()异同移动平均线； 12. 新增 get_KDJ()随机指标； 13. 新增 get_RSI()相对强弱指标； 14. 新增 get_CCI()顺势指标； 15. 新增 create_dir()创建文件目录路径；