目录

极智量化介绍

极智量化是什么

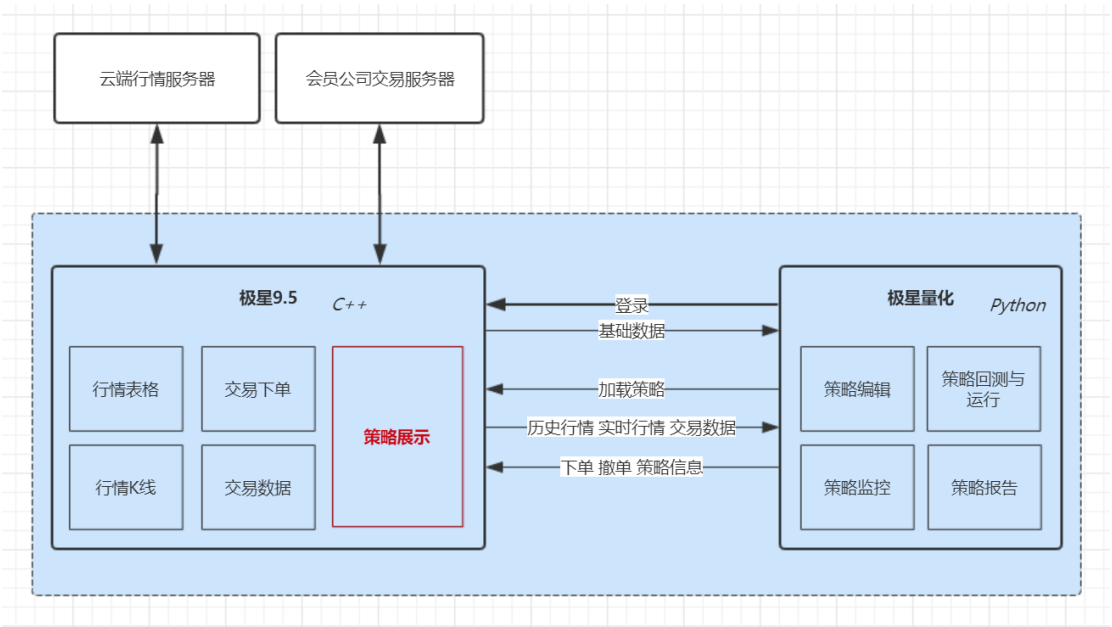
极智量化终端是基于 python 语言开发并在 windows 平台下运行的终端程序，具备自由开发、海量数据、安全保密等特性。本产品为投资者提供了包括历史数据-实时数据-开发调试-策略回测-模拟交易-实盘交易-运行监控-收益统计-风险管理的全套解决方案。

产品特点

- 产品由 python 开发，代码开源，支持自由开发、自由扩展；
- 用户策略开发由 python 实现，易上手，减少用户在编程语言学习上的精力投入；
- 提供海量数据，数据覆盖日线、分钟线、秒线、tick 类型，品种支持期货、期权、现货、跨期套利、品种套利、外汇、证券。
- 支持获取历史行情，即时行情；
- 每个运行的策略都是一个独立的进程，单策略运行异常不影响其他策略；
- 同一策略支持多周期多合约数据同时触发；
- 兼容 9.2 程序化、TB 的函数使用方法；
- 策略服务本地化， 研究成果保密；
- 可以进行策略编写、调试与回测，生成投资分析报告；
- 支持内外盘期货期权交易，支持多交易后台

系统架构

极智量化系统架构图如下：



极星 9.5： 由 C++实现，作为下单通道、数据通道和策略指标展示界面，连接行情云及会

员交易服务器
极智量化：由 python 实现，提供策略编写和回测、运行监控以及回测报告展示。所有策略逻辑都在极星量化端实现

Licence

十分钟教程

本部分将指导用户如何开始快速开始使用极智量化，这里列出了从安装到编写策略的方法。

1. 安装

极智量化终端目前支持在 win7、win8、win8.1、win10 等操作系统上运行，需预先安装 python3.7 环境及一些 python 的第三方安装包。为了减少用户在使用极智量化前在配置运行环境上花费大量的精力，我们提供了一个打包好的安装包供用户下载和安装。该安装包会自动完成安装极智量化产品和配置极智量化运行所需要的外部 python 环境，运行极智量化对系统的要求如下：

- Windows7 及以上版本
- 屏幕分辨率最低支持 1024*768
- 内存 2G 及以上

安装步骤如下：

- 下载极星量化安装包，下载完成后双击 exe 程序进行安装；
- 安装过程持续 5-10 分钟，请勿关闭安装窗口，如遇到杀毒软件，请选择“允许所有操作”；
- 安装完成后可，后续版本更新可通过运行安装路径下的 equant/update.bat 手动更新运行

2. 订阅并使用 K 线数据

要使用某合约的 K 线数据，首先需要订阅该合约的数据，通过调用 SetBarInterval()函数可以很方便的订阅指定合约的 K 线数据，这里以苹果主连合约为例，订阅苹果主连合约 1 分钟历史数据 500 根：

```
SetBarInterval("ZCE|Z|AP|MAIN", "M", 1, 500)
```

订阅合约 K 线数据之后，就可以在策略中通过调用 Open()、Close()、High()、Low()获取合约的高开低收历史数据或通过 HisData()获取各种历史数据数组。

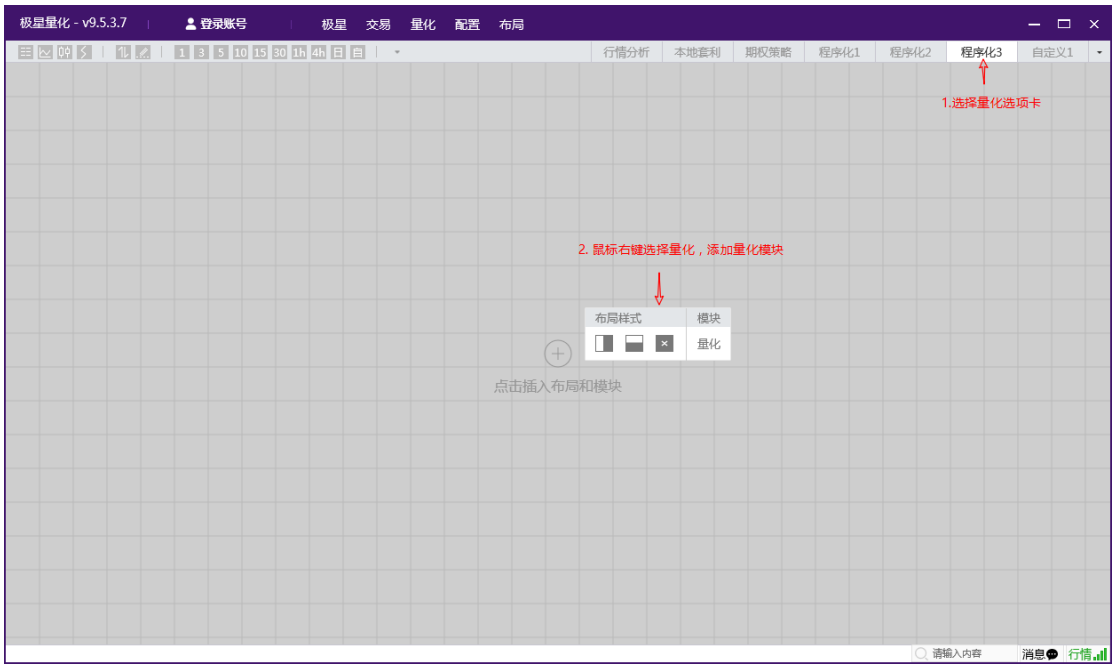
```
# 输出苹果主连合约 1 分钟 K 线的最新收盘价
LogInfo("最后一根 K 线的收盘价：  ", Close("ZCE|Z|AP|MAIN", "M", 1)[-1])
```

也可以多次调用 SetBarInterval()函数订阅不同合约、不同周期的数据。极智量化将把用户第一次通过 SetBarInterval()函数订阅的合约作为基准合约。基准合约的概念请参考常见问题中关于基准合约的解释。

到目前为止，你已经订阅了 K 线数据并能通过 API 接口查询所订阅的 K 线数据。下面介绍如何显示订阅的 K 线数据。

3. K 线数据展示

若想要你订阅的 K 线数据在图形界面上显示，需要在极星量化客户端中量化选项卡上插入量化模块:



当运行策略后，在该选项卡上会展示订阅的 K 线信息。

在该界面上鼠标右键选择“选择合约”会列出现在运行的策略 id，通过选择对应的策略 id 可以选择展示的策略 K 线，如下图所示:

该页面还提供插入窗口以支持多图标展示功能，通过在该界面上插入窗口以实现同时展示多个策略运行的 K 线和指标。如下图所示展示了四个策略的运行 K 线和指标等信息：

4. 触发类型

极智量化提供了五种触发方式供用户选择：

- K 线触发： 由 K 线数据触发策略，运行策略的 handle_data()函数。
- 即时行情触发：订阅的合约的即时行情更新时会触发策略，运行策略的 handle_data()函数
- 交易数据触发：订单的状态发生改变时会触发策略，运行策略的 handle_data()函数。这是一种常见的基于事件的触发机制，用户选择了该触发方式后，每当用户的订单状态发生改变，就会执行策略的 handle_data()函数
- 每隔固定时间触发：每隔固定的时间间隔会触发策略，运行策略的 handle_data()函数

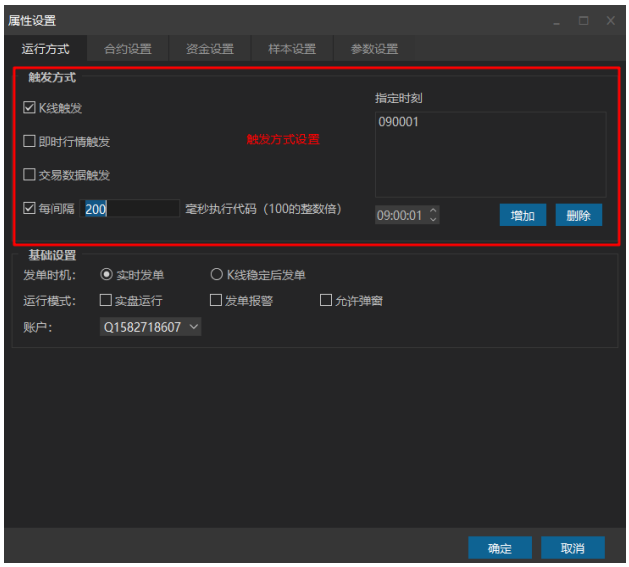
- 指定时刻触发：在用户指定的时刻会触发策略，运行策略的 handle_data()函数

每一个用户关心的数据变化时，都会根据用户设置的触发类型触发策略，调用策略的 handle_data 函数。以上触发类型可以任意组合，可以订阅一种触发类型，也可以同时订阅多种触发类型：

SetTriggerType(1)

SetTriggerType(2)

以上调用了两次 SetTriggerType()函数，设置即时行情和交易数据触发两种触发方式触发策略。也可以通过在运行设置界面上设置触发方式：



注意：实盘运行时且触发方式为 K 线触发时，若发单方式选择“K 线稳定后发单”，则策略会根据用户订阅的 K 线频率等到 K 线稳定后才会被触发，如订阅的 1 分钟苹果主连数据，则策略会在 1 分钟数据完成时才会被触发；若发单方式选择“实时发单”，则策略会把每个 Tick 当做 K 线被触发。

5. 策略触发的上下文环境

策略触发的上下文环境包含策略被触发时的触发方式的信息，策略的上下文环境包含在策略的入口函数（入口函数在编写自己的策略处会介绍）的 context 参数中，包含：

- context.strategyStatus(): 策略状态（回测阶段、实时阶段）
- context.triggerType(): 触发类型（即时行情、交易数据、定时、指定时刻、K 线触发）
- context.contractNo(): 触发合约
- context.kLineType(): 触发的 K 线类型(分笔、秒线、分钟、日线)
- context.kLineSlice(): 触发的 K 线周期
- context.tradeDate(): 触发的交易日
- context.dateTimeStamp(): 触发的时间戳
- context.triggerData(): 触发类型对应的数据（K 线 或 即时行情 或 定单）

这些函数可以更好的帮助用户掌握策略的运行信息。context 函数的具体用法可以参考示例策略的"策略的上下文环境"策略。其中 context.triggerData()函数的返回值会因不同的触发方式返回结果也不同，其中不同触发方式返回的字典结构的键值含义可以在 API 中的 context 函数部分的 triggerData 函数说明部分查看。

6. 策略交易

有两组函数可以用于发送委托单。第一组包含：

- Buy：产生一个多头建仓操作
- BuyToCover：产生一个空头平仓操作
- Sell：产生一个多头平仓操作
- SellShort： 产生一个空头建仓操作

第二组包含：

- A_SendOrder：针对指定的账户、商品发送委托单
- A_ModifyOrder: 发送改单指令
- A_DeleteOrder：针对指定的账户、商品发送撤单指令
- DeleteAllOrders：批量撤单

两组交易函数都既可以用在历史回测阶段也可以用在实盘交易阶段。第一组组下单函数的可选参数较少，表明用户可控制的下单参数较少。第二组交易函数可选参数更多，表明第二组函数可以为用户提供更智能、更精细化的下单控制。

要进行实盘交易，需要调用 SetActual()函数，设置策略可在实盘阶段运行。

```
Buy(1, Close()[-1], contractNo="ZCE|Z|AP|MAIN")
```

该下单函数会以当前 Bar 的收盘价买入一手苹果主连合约。在历史回测阶段，该下单函数会向历史回测引擎发送一笔委托单并立即成交，在实盘交易阶段，该函数会向交易后台发送一笔委托单。

```
A_SendOrder(Enum_Buy(), Enum_Entry(), 1, Close()[-1], contractNo="ZCE|Z|AP|MAIN",
userNo="Test")
```

该下单函数会针对指定的账户 Test，以苹果主连合约的最新收盘价买入一手该合约，发送成功该函数会返回 0 和订单编号，发送失败会返回失败信息。要针对 Test 账户下单，要确保在客户端上登录 Test 账户。

要进行撤单操作，使用 A_DeleteOrder()函数：

```
A_DeleteOrder(id)  # 撤订单号为 id 的订单
```

注意：在使用交易函数对某合约下单前，必须要用 SetBarInterval()函数订阅该合约的数据。

7. 用户参数

极智量化提供了定义用户参数的方法供用户使用，用户参数可以帮助用户在完成策略逻辑的开发后，不需要修改程序，只需要通过在界面上修改某些关键参数的值即可完成用新修改的参数重新运行策略，避免用户频繁修改策略代码。

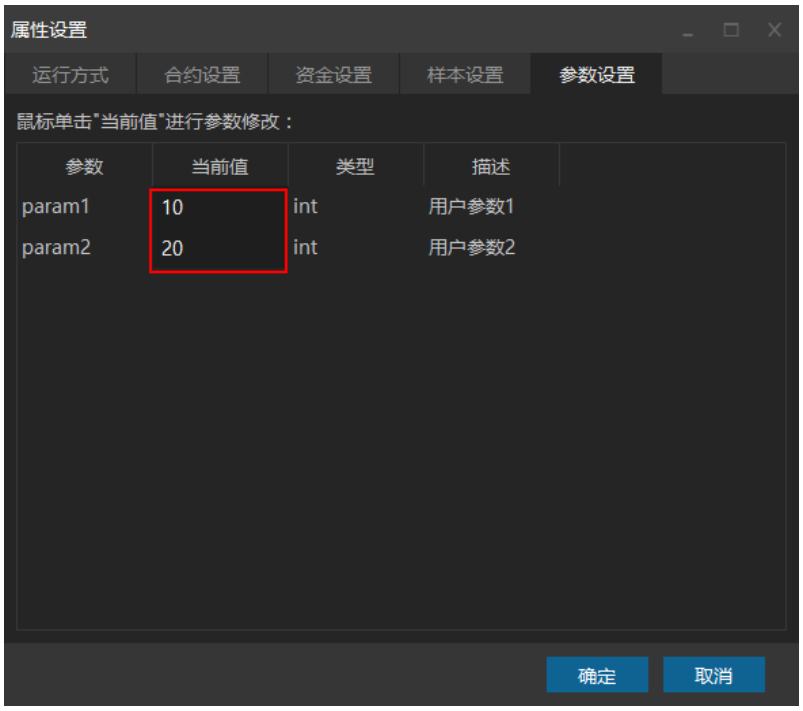
用户参数的定义格式如下：

```
g_params['param1'] = 1
g_params['param2'] = 2
```

这里定义了两个用户参数 param1 和 param2，定义用户参数的形式必须严格按照上述格式定义，否则会解析用户参数失败。解析的用户参数会在量化终端的运行设置界面中显示。

程序会自动解析用户参数的参数名、当前值、参数类型及对该参数的描述。要对用户参数进行修改，只需要鼠标单击当前值显示框，输入用户要更改的值即可，策略将以用户修改后的参数运行策略。在策略运行列表中鼠标右键选择属性设置:

弹出的窗口中会显示用户修改的该策略的最新参数信息。例如点击量化界面上的"运行"按钮，在弹出的窗口中的参数设置选项卡上修改 g_param1 的值为 10，g_param2 的值为 20，然后点击运行后，如果策略没有语法错误，会在策略运行界面显示新加载的策略运行信息，在策略信息上右键选择属性设置，弹出的窗口中会显示用户设置的最新的用户参数:



注意：通过鼠标右键选择"属性设置"弹出窗口，在该窗口上修改用户参数，会以新的用户参数重新运行策略，在该窗口中修改其他的设置均不会生效。

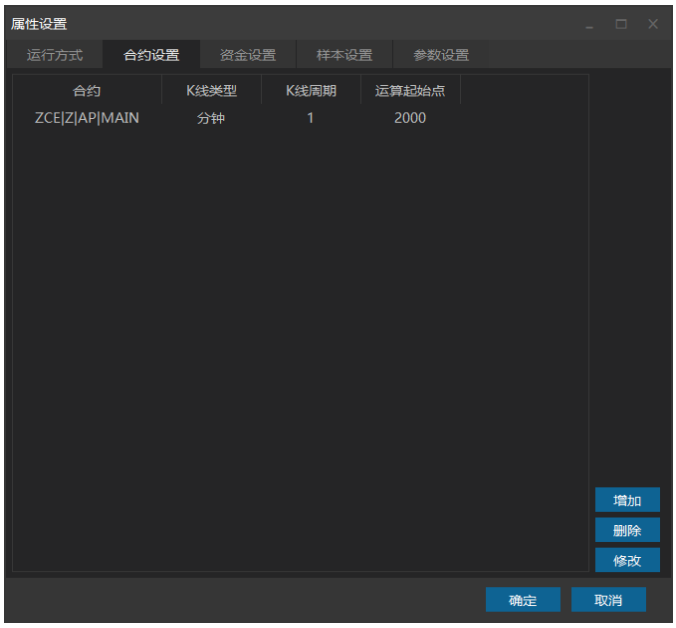
8. 持仓同步

在实盘交易时，可以进行持仓同步操作。在极智量化主界面的组合监控选项卡上会显示持仓信息：

持仓信息会列出策略仓和账户仓的持仓信息，点击“持仓一键同步”按钮可以对出现仓差的账户进行持仓同步操作。该功能首先将处于排队中的订单进行撤单，然后根据仓差情况发送等量的订单，发送订单的价格根据“同步设置”中的选项计算得出。

9. 策略运行

在极智量化终端策略列表中新建策略后，用户可以根据自己的交易逻辑编写自己的策略，编写完成后，点击"运行"按钮弹出属性设置窗口，用户可以在该窗口中设置触发方式、发单设置、账户设置、合约设置、资金设置、用户参数修改等策略运行的基础设置，设置完成后点击"确定"按钮即可运行用户策略：



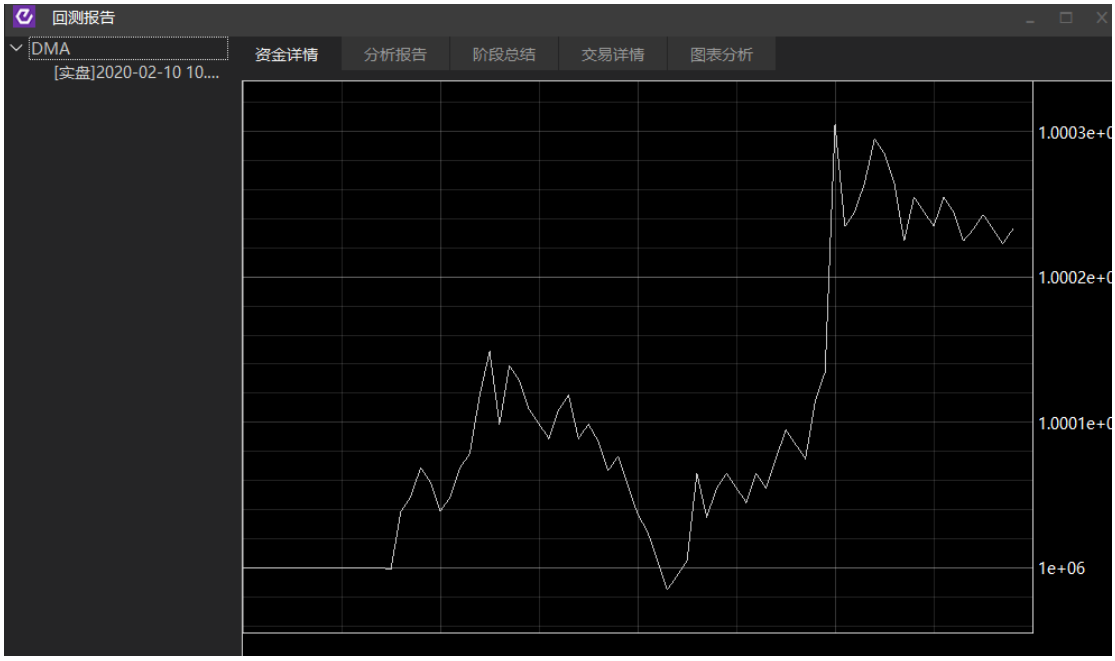
加载成功后的策略会在策略运行中显示运行信息。策略有语法错误导致运行失败的情况下，会在错误信息处显示错误详情，用户可根据错误信息修改策略后重新运行策略。

注意：通过属性设置界面和在策略中通过代码设置策略运行的属性，效果是一样的。如果既通过图形用户界面又通过代码对策略运行的同一属性设置，同一个不冲突的设置程序会合并两处设置结果，冲突的设置以策略代码中设置运行属性为准。例如：用户在界面上选择"触发方式"为"K 线触发"，同时设置"发单时机"为"实时发单"。在策略中通过 SetTriggerType(1)设置"触发方式"为"即时行情触发"，通过 SetOrderWay(2)设置"发单方式"为"K 线稳定后发单"，策略将以策略代码中设置的"K 线稳定后发单"的发单方式发单，触发方式将取界面设置和代码设置的触发方式的并集，即"即时行情触发"和"K 线触发"两种触发方式触发策略。

策略在历史阶段运行时，每笔历史 K 线数据都会触发策略执行 handle_data()函数的代码。历史数据运行完后，如果用户在策略中实现了 hisover_callback()函数，策略会在历史阶段结束后调用 hisover_callback()函数，之后策略会进入实盘阶段运行。实盘阶段，设置的各种触发类型的数据变化都会触发 handle_data()函数的执行。用户停止策略时，如果在策略中实现了 exit_callback()函数，策略停止时会执行 exit_callback()函数。

10. 测试报告

策略加载完成后，会在量化主窗口的策略运行处显示策略运行的概要信息。要查看策略运行的详细信息，可在策略运行信息上右键选择"投资报告"查看策略运行的详细信息：



回测报告列出了"资金详情"、"分析报告"、"阶段总结"、"交易详情"、"图表分析"等信息，用户可以根据回测报告分析策略的收益信息，并调整策略以期获得最大的投资收益。

11. 编写自己的策略

极智量化抽离了策略框架的所有技术细节，用户只需要调用提供的 API 编写自己的策略，以便将主要的精力放在策略开发和测试上，而不必关注过多的技术细节。

极智量化的 API 主要分为以下几类：约定函数、K 线数据、即时行情、策略交易、属性函数、策略状态、策略性能、账户函数、枚举函数、设置函数、绘图函数、统计函数、日志函数。其中约定函数作为 API 的入口函数，用户必须实现对应的约定函数才可以正确的运行策略。

约定函数包括：

initialize(context)：初始化方法，会在策略启动时运行一次。该函数中可以进行合约数据的订阅以及对策略运行的条件进行设置等。[必须实现]

handle_data(context)：该函数在策略收到每一笔关心的数据时都会被调用，策略的主要业务在该函数中实现。[必须实现]

hisover_callback(context)：该函数在策略运行的历史阶段结束时被调用。用户可以在该函数中可以对历史阶段的仓位进行平仓等操作。[选择实现]

exit_callback(context)：该函数在策略退出前被调用，用户可以在该函数中实现一些数据保存，仓位处理等操作。[选择实现]

极智量化策略的基本结构包括：模块导入、策略参数、全局变量、约定函数，下面实现一个简单的策略。

```
# 导入功能模块
```

```
import talib
import numpy as np

# 策略的任何初始化逻辑
def initialize(context):
    # 订阅郑商所苹果主连合约 1 分钟历史数据 500 根 SetBarInterval("ZCE|Z|AP|MAIN",
    "M", 1, 500)

# 初始化函数中订阅的合约的数据更新将会触发此段逻辑
def handle_data(context):
    LogInfo("每次策略被触发时调用")

# 策略历史阶段运行完成时将会执行此逻辑
def hisover_callback(context):
    LogInfo("策略历史阶段运行结束")

# 策略停止时将会运行此逻辑
def exit_callback(context):
    LogInfo("策略即将退出")
```

以上就实现了一个简单地策略，该策略导入了 numpy、talib 两个常用的 python 第三方库：

```
import talib
import numpy as np
```

订阅了苹果主连合约的历史数据：

```
SetBarInterval("ZCE|Z|AP|MAIN", "M", 1, 500)
```

并输出了一些日志信息：

```
LogInfo("每次策略被触发时调用")
```

该策略中可以不包含 hisover_callback()和 exit_callback()两个函数，这里只是为了演示上述两个函数在何时被策略调用。注意：若要策略能够被触发，必须在策略中调用 SetBarInterval()函数或在属性设置界面上设置合约。

接下来，我们需要获取数据，根据数据来确定我们的仓位逻辑，因此会使用到数据查询的 API 接口。这里列举部分数据查询接口：

- BarCount： 获取指定合约的 Bar 总数
- Open：指定合约指定周期的开盘价
- Close：指定合约指定周期的开盘价
- High：指定合约指定周期的最高价
- Low：指定合约指定周期的最低价
- Vol：成交量
- OpenInt：持仓量
- BuyPosition：获得当前持仓的买入方向的持仓量
- SellPosition：获得当前持仓的卖出方向的持仓量
- MarketPosition：获取当前的持仓状态

金融数据的技术分析指标的计算方法可以引用 python 第三方库 talib 所提供的方法。

获取到历史数据可以进行一些常用指标的计算，以 talib 库的 MA 指标为例：

```
# 计算 5 分钟苹果主连合约的收盘价移动平均值
ma1 = talib.MA(Close("ZCE|Z|AP|MAIN", "M", 1), 5)
# 计算 20 分钟苹果主连合约的收盘价移动平均值
ma2 = talib.MA(Close("ZCE|Z|AP|MAIN", "M", 1), 20)
```
上述 ma1 指标和 ma2 指标的计算用到了合约的收盘价数据 Close()，MA 指标方法是调用 python 第三方库 talib 的方法。修改上述策略如下：
```
import talib
import time

p1 = 5
p2 = 20
qty = 1

def initialize(context):
    SetBarInterval("ZCE|Z|AP|MAIN", "M", 1, 500)

def handle_data(context):
    # 判断合约的收盘价数据长度是否满足计算要求
    if len(Close()) < p2:
        return

    ma1 = talib.MA(Close(), p1)
    ma2 = talib.MA(Close(), p2)
    if ma1[-1] > ma2[-1] and MarketPosition() <= 0:
        LogInfo("金叉进行买入建仓操作")
    elif ma1[-1] < ma2[-1] and MarketPosition() >= 0:
        LogInfo("死叉进入卖出开仓操作")

def hisover_callback(context):
    LogInfo("策略历史阶段运行结束")

# 策略停止时将会运行此逻辑
def exit_callback(context):
    LogInfo("策略即将退出")
```
这里增加定义了三个全局变量 p1，p2，qty，并增加了交易逻辑：
```
if ma1[-1] > ma2[-1] and MarketPosition() <= 0:
   LogInfo("金叉进行买入建仓操作")
elif ma1[-1] < ma2[-1] and MarketPosition() >= 0:
    LogInfo("死叉进入卖出开仓操作")
```
接下来只需要调用交易接口就可以进行交易了。增加交易接口的策略修改如下：
```
import talib
import time

p1 = 5
```

```
p2 = 20
qty = 1

def initialize(context):
    SetBarInterval("ZCE|Z|AP|MAIN", "M", 1, 500)

def handle_data(context):
    # 判断合约的收盘价数据长度是否满足计算要求
    if len(Close()) < p2:
        return

    ma1 = talib.MA(Close(), p1)
    ma2 = talib.MA(Close(), p2)
    if ma1[-1] > ma2[-1] and MarketPosition() <= 0:
        Buy(qty, Close()[-1])
    elif ma1[-1] < ma2[-1] and MarketPosition() >= 0:
        SellShort(qty, Close()[-1])

def hisover_callback(context):
    LogInfo("策略历史阶段运行结束")

# 策略停止时将会运行此逻辑
def exit_callback(context):
    LogInfo("策略即将退出")
```

这样就完成了一个完整的双均线策略的编写。

API 文档

API 函数说明

约定函数

initialize
【必须实现】
【说明】
初始化方法，在策略运行时只会在启动时触发一次。用户使用该方法对策略运行的初始化信息进行设置。
【参数】
context-策略上下文
【示例】

```
def initialize(context):
    # 订阅 1 分钟 PTA 主连数据 500 根
    SetBarInterval("ZCE|Z|TA|MAIN", "M", 1, 500)
```

handle_data
【必须实现】
【说明】
该函数在策略收到每一笔关心的数据时都会被调用，策略的主要业务就在这里实现，包括交易数据的产生、订单的创建。
【参数】
context-策略上下文
【示例】

```
def handle_data(context):
    # 以 price 的价格买入 1 手 PTA 主连合约
    Buy(1, price, "ZCE|Z|TA|MAIN")
```

hisover_callback
【选择实现】
【说明】
历史回测阶段结束执行的回调函数。用户可在该函数中实现在历史回测阶段结束时需要进行的操作。
【参数】
context-策略上下文
【示例】

```
def hisover_callback(context):
    LogInfo("This function will be called before history phase ends")
```

exit_callback

【选择实现】
【说明】
该回调函数在策略退出前被调用。用户可在该函数中实现在策略停止前需要进行的操作
【参数】
context-策略上下文
【示例】
```
def exit_callback(context):
    LogInfo("This function will be called before strategy quits")
```

交易函数

Buy
【说明】
产生一个多头建仓操作，该下单函数可用于历史阶段和实盘阶段。
【语法】
void Buy(int orderQty=0, float orderPrice=0, string contractNo=None, bool needCover = True, string userNo='')
【参数】
orderQty (int)　买入数量，默认为 0
orderPrice (float)　买入价格，默认为 0
contractNo(str)　合约代码，不指定该参数默认使用基准合约
needCover(bool)　是否先平对手仓，默认值为 True
userNo(str)　用户编号，为字符串，不指定默认使用界面选定用户编号
【返回值】返回值
【备注】
产生一个多头建仓操作，无返回值。当参数 needCover 为 True 时，如果存在对手仓，会先平掉所有对手仓，然后进行开仓操作，当 needCover 为 False 时，只进行开仓操作。
该函数仅用于多头建仓，其处理规则如下：
如果当前持仓状态为持平，该函数按照参数进行多头建仓。
如果当前持仓状态为空仓，当参数 needCover 为 True 时，该函数先平掉所有空仓，同时按照参数进行多头建仓，两个动作同时发出。当 needCover 为 False 时，该函数仅进行多头建仓操作。
如果当前持仓状态为多仓，该函数将继续建仓，但具体是否能够成功建仓要取决于系统中关于连续建仓的设置，以及资金，最大持仓量等限制。
当委托价格超出 k 线的有效范围，在历史数据上，将会取最接近的有效价格发单；在实盘中，将会按照实际委托价格发单。
例如：当前 k 线有效价格为 50-100，用 buy(1, 10)发单，委托价将以 50 发单。
【示例】
```
# 平掉所有空仓，并用 10.2 的价格买入 50 张合约
Buy(50, 10.2)
# 平掉所有空仓，并用当前 Bar 的收盘价买入 10 张合约，马上发送委托
Buy(10, Close()[-1])
# 平掉所有空仓，并用现价买入 5 张合约, 马上发送委托
```

Buy(5, 0)
# 平掉所有空仓，并用现价买入交易设置中设置的默认下单手数，马上发送委托
Buy(0, 0)
# 平掉所有空仓，并用当前 Bar 收盘价买入 10 张合约，马上发送委托
Buy(10, Close()[-1])
# 用当前 Bar 收盘价买入 10 张合约，马上发送委托
Buy(10, Close()[-1], needCover=False)

BuyToCover
【说明】
产生一个空头平仓操作，该下单函数可用于历史阶段和实盘阶段。
【语法】
void BuyToCover(int orderQty =0, float orderPrice=0, string contractNo=None, string userNo='', char coverFlag = 'A')
【参数】
orderQty (int) 买入数量， 默认为 0
orderPrice(float) 买入价格，默认为 0
contractNo 合约代码，默认为基准合约
userNo 用户编号，为字符串，默认使用界面选定用户编号
coverFlag 平今平昨标志（此参数仅对 SHFE 和 INE 有效），默认设置为'A'自适应（先平昨再平今），若平昨，则需设置为'C'，若平今，则需设置为'T'
【返回值】
无返回值
【备注】
产生一个空头平仓操作，无返回值。
该函数仅用于空头平仓，其处理规则如下：
如果当前持仓状态为持平，该函数不执行任何操作。
如果当前持仓状态为多仓，该函数不执行任何操作。
如果当前持仓状态为空仓，如果此时 orderQty 使用默认值，该函数将平掉所有空仓，达到持平的状态，否则只平掉参数 orderQty 的空仓。
当委托价格超出 k 线的有效范围，在历史数据上，将会取最接近的有效价格发单；在实盘中，将会按照实际委托价格发单。
例如：当前 k 线有效价格为 50-100，用 BuyToCover(1,10)发单，委托价将以 50 发单。
【示例】
# 用 10.2 的价格空头买入 50 张合约
BuyToCover(50, 10.2)
# 用当前 Bar 收盘价空头买入 10 张合约，马上发送委托
BuyToCover(10, Close()[-1])
# 用现价空头买入 5 张合约，马上发送委托
BuyToCover(5, 0)
# 用现价按交易设置中设置的下单量，马上发送委托
BuyToCover(0, 0)

Sell

【说明】
产生一个多头平仓操作，该下单函数可用于历史阶段和实盘阶段。
【语法】
void Sell(int orderQty =0, float orderPrice=0, string contractNo=None, string userNo='', char coverFlag = 'A')
【参数】
orderQty 买入数量，为整型值，默认为 0
orderPrice 买入价格，为浮点数，默认为 0
contract 合约代码，为字符串，默认使用基准合约
userNo 用户编号，为字符串，默认使用界面选定用户编号
coverFlag 平今平昨标志（此参数仅对 SHFE 和 INE 有效）
【返回值】
无返回值
【备注】
产生一个多头平仓操作，无返回值。
该函数仅用于多头平仓，其处理规则如下：
如果当前持仓状态为持平，该函数不执行任何操作。
如果当前持仓状态为空仓，该函数不执行任何操作。
如果当前持仓状态为多仓，如果此时 orderQty 使用默认值，该函数将平掉所有多仓，达到持平的状态，否则只平掉参数 orderQty 的多仓。
当委托价格超出 k 线的有效范围，在历史数据上，将会取最接近的有效价格发单；在实盘中，将会按照实际委托价格发单。
例如：当前 k 线有效价格为 50-100，用 sell(1,10)发单，委托价将以 50 发单。
【示例】
```
# 用 10.2 的价格卖出 50 张合约
Sell(50, 10.2)
# 用当前 Bar 收盘价卖出 10 张合约，马上发送委托
Sell(10, Close()[-1])
# 用现价卖出 5 张合约，马上发送委托
Sell(5, 0)
# 用现价按交易设置中的设置,马上发送委托
Sell(0, 0)
```

SellShort
【说明】
产生一个空头建仓操作，该下单函数可用于历史阶段和实盘阶段。
【语法】
void SellShort(int orderQty =0, float orderPrice=0, string contractNo=None, bool needCover = True, string userNo='')
【参数】
orderQty (int)　买入数量，默认为 0
orderPrice (float)　买入价格，默认为 0
contractNo(str)　合约代码，不指定该参数默认使用基准合约
needCover(bool)　是否先平对手仓，默认值为 True

userNo(str)　用户编号，为字符串，不指定默认使用界面选定用户编号
【返回值】
无返回值
【备注】
产生一个空头建仓操作，无返回值。当参数 needCover 为 True 时，如果存在对手仓，会先平掉所有对手仓，然后进行开仓操作，当 needCover 为 False 时，只进行开仓操作。
该函数仅用于空头建仓，其处理规则如下：
如果当前持仓状态为持平，该函数按照参数进行空头建仓。
如果当前持仓状态为多仓，当参数 needCover 为 True 时，该函数先平掉所有空仓，同时按照参数进行空头建仓，两个动作同时发出。当 needCover 为 False 时，该函数仅进行空头建仓操作。
如果当前持仓状态为空仓，该函数将继续建仓，但具体是否能够成功建仓要取决于系统中关于连续建仓的设置，以及资金，最大持仓量等限制。
当委托价格超出 k 线的有效范围，在历史数据上，将会取最接近的有效价格发单；在实盘中，将会按照实际委托价格发单。
例如：当前 k 线有效价格为 50-100，用 SellShort(1,10)发单，委托价将以 50 发单。
【示例】
```
# 平掉所有多头仓位，并用 10.2 的价格空头卖出开仓 50 张合约
SellShort(50, 10.2)
# 平掉所有多头仓位，并用当前 Bar 收盘价空头卖出开仓 10 张合约，马上发送委托
SellShort (10, Close()[-1])
# 平掉所有多头仓位，并用现价空头卖出开仓 5 张合约，马上发送委托
SellShort (5, 0)
# 表示平掉所有多头仓位，并用现价按交易设置中设置的默认下单手数,马上发送委托
SellShort (0, 0)
#平掉所有多头仓位，并用当前 Bar 收盘价空头卖出 10 张合约，马上发送委托
SellShort (10, Close()[-1])
# 用当前 Bar 收盘价空头卖出开仓 10 张合约，马上发送委托
SellShort(10, Close()[-1], needCover=False)
```

StartTrade
【说明】
恢复实盘交易
【语法】
void StartTrade()
【参数】
无
【返回值】
无
【备注】
在策略运行时，使用 StopTrade 函数暂停策略向实盘发单后，可以通过 StartTrade 函数恢复策略向实盘发单的功能。
【示例】
无

StopTrade
【说明】
暂停实盘交易
【语法】
void StopTrade()
【参数】
无
【返回值】
无
【备注】
在策略运行时，使用 StopTrade 可以暂时停止策略向实盘发单。
【示例】
无

IsTradeAllowed
【说明】
是否允许实盘交易
【语法】
bool IsTradeAllowed()
【参数】
无
【备注】
获取策是否可以向实盘发单的布尔值，策略实盘运行时并且允许向实盘发单（未设置 StopTrade）时返回 True，否则返回 False
【示例】
无

K 线数据

BarCount
【说明】
指定合约 Bar 的总数，包括订阅的 K 线总数
【语法】
int BarCount(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(str) 合约编号，默认为基准合约
kLineType(char) k 线类型，可选值请参阅周期类型的枚举值
kLineVale(int) k 线周期
【返回值】
整形
【备注】
返回值为订阅的 K 线总数与新行情形成的指定周期（kLineVale）的 K 线数之和

Open
【说明】
指定合约指定周期的开盘价
【语法】
array Open(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(str) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【返回值】
返回截止当前 Bar 的所有开盘价
【备注】
简写 O, Tick 时为当前最新价，返回值 numpy 数组，包含截止当前 Bar 的所有开盘价
Open()[-1] 表示当前 Bar 开盘价，Open()[-2]表示上一个 Bar 开盘价，以此类推
【示例】
# 获取基准合约的所有开盘价数组
Open()
#获取白糖主连合约 1 分钟 K 线的所有开盘价数组
Open('ZCE|Z|SR|MAIN', 'M', 1)

Close
【说明】
指定合约指定周期的收盘价
【语法】
array Close(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(str) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【返回值】
返回截止当前 Bar 的所有收盘价
【备注】
简写 C, Tick 时为当前最新价，返回值 numpy 数组，包含截止当前 Bar 的所有开盘价
Close()[-1] 表示当前 Bar 收盘价，Close()[-2]表示上一个 Bar 收盘价，以此类推
【示例】
# 获取基准合约的所有收盘价数组
Close()
#获取白糖主连合约 1 分钟 K 线的所有收盘价数组
Close ('ZCE|Z|SR|MAIN', 'M', 1)

High
【说明】
指定合约指定周期的最高价

【语法】
array High (string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(str) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【返回值】
返回截止当前 Bar 的所有最高价
【备注】
简写 H， Tick 时为当前最新价，返回值为 numpy 数组，包含截止当前 Bar 的所有最高价
High('ZCE|Z|SR|MAIN', 'M', 1)[-1] 表示当前 Bar 最高价，High('ZCE|Z|SR|MAIN', 'M', 1)[-2]表示上一个 Bar 最高价，以此类推
【示例】
无

Low
【说明】
指定合约指定周期的最低价
【语法】
array Low (string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(str) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【返回值】
返回截止当前 Bar 的所有最低价
【备注】
简写 L， Tick 时为当前最新价，返回值 numpy 数组，包含截止当前 Bar 的所有最低价
Low()[-1] 表示当前 Bar 最低价，Low()[-2]表示上一个 Bar 最低价，以此类推
【示例】
无

OpenD
【说明】
指定合约指定周期 N 天前的开盘价
【语法】
float OpenD(int daysAgo=0, string contractNo='')
【参数】
daysAgo(int) 第几天前，默认为 0，即当天
contractNo(string) 合约编号，默认为基准合约
【返回值】
返回指定合约指定周期 N 天前的开盘价
【备注】
使用该函数前请确保在策略的 initial 方法中使用 SetBarInterval(contractNo, 'D', 1)方法订阅

contractNo 合约的日线信息；若 daysAgo 超过了订阅合约 contractNo 日线数据的样本数量，则返回为-1。
【示例】
# 获取白糖主连合约 3 天前的开盘价
OpenD(3，'ZCE|Z|SR|MAIN')

CloseD
【说明】
指定合约指定周期 N 天前的收盘价
【语法】
float CloseD(int daysAgo=0, string contractNo='')
【参数】
daysAgo(int) 第几天前，默认为 0，即当天
contractNo(string) 合约编号，默认为基准合约
【返回值】
返回指定合约指定周期 N 天前的收盘价
【备注】
使用该函数前请确保在策略的 initial 方法中使用 SetBarInterval(contractNo, 'D', 1)方法订阅 contractNo 合约的日线信息；若 daysAgo 超过了订阅合约 contractNo 日线数据的样本数量，则返回为-1。
【示例】
# 获取白糖主连合约 3 天前的收盘价
CloseD(3，'ZCE|Z|SR|MAIN')

HighD
【说明】
指定合约指定周期 N 天前的最高价
【语法】
float HighD(int daysAgo=0, string contractNo='')
【参数】
daysAgo(int) 第几天前，默认为 0，即当天
contractNo(string) 合约编号，默认为基准合约
【返回值】
返回指定合约指定周期 N 天前的最高价
【备注】
使用该函数前请确保在策略的 initial 方法中使用 SetBarInterval(contractNo, 'D', 1)方法订阅 contractNo 合约的日线信息；若 daysAgo 超过了订阅合约 contractNo 日线数据的样本数量，则返回为-1。
【示例】
# 获取白糖主连合约 3 天前的最高价
HighD(3，'ZCE|Z|SR|MAIN')

LowD
【说明】

指定合约指定周期 N 天前的最低价
【语法】
float LowD(int daysAgo=0, string contractNo='')
【参数】
daysAgo(int) 第几天前，默认为 0，即当天
contractNo(string) 合约编号，默认为基准合约
【返回值】
返回指定合约指定周期 N 天前的最低价
【备注】
使用该函数前请确保在策略的 initial 方法中使用 SetBarInterval(contractNo, 'D', 1)方法订阅 contractNo 合约的日线信息；若 daysAgo 超过了订阅合约 contractNo 日线数据的样本数量，则返回为-1。
【示例】
# 获取白糖主连合约 3 天前的最低价
LowD(3，'ZCE|Z|SR|MAIN')

Vol
【说明】
指定合约指定周期的成交量
【语法】
array Vol(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【返回值】
返回指定合约指定周期的成交量数组
【备注】
简写 V，返回 numpy 数组，包括截止当前 Bar 的所有成交量数据
Vol()[-1] 表示当前 Bar 成交量，Vol()[-2]表示上一个 Bar 成交量，以此类推
【示例】
无

OpenInt
【说明】
指定合约指定周期的持仓量
【语法】
numpy.array OpenInt(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
返回 numpy 数组，包括截止当前 Bar 的所有持仓量

OpenInt()[-1]表示当前 Bar 持仓量，OpenInt()[-2]表示上一个 Bar 持仓量，以此类推
【示例】
无

BarStatus
【说明】
指定合约当前 Bar 的状态值
【语法】
int BarStatus(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
返回值整型, 0 表示第一个 Bar，1 表示中间普通 Bar，2 表示最后一个 Bar
【示例】
无

CurrentBar
【说明】
指定合约当前 Bar 的索引值
【语法】
int CurrentBar(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
第一个 Bar 返回值为 0，其他 Bar 递增
当无数据时，不存在当前 Bar，返回-1
【示例】
无

Date
【说明】
当前 Bar 的日期
【语法】
int Date(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
简写 D，返回格式为 YYYYMMDD 的整数

【示例】
当前 Bar 对应的日期为 2019-03-25，则 Date 返回值为 20190325

Time
【说明】
当前 Bar 的时间
【语法】
float Time(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
简写 T，返回格式为 0.HHMMSS 的浮点数
【示例】
当前 Bar 对应的时间为 11:34:21，Time 返回值为 0.113421
当前 Bar 对应的时间为 09:34:00，Time 返回值为 0.0934
当前 Bar 对应的时间为 11:34:00，Time 返回值为 0.1134

TradeDate
【说明】
指定合约当前 Bar 的交易日
【语法】
int TradeDate(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
返回格式为 YYYYMMDD 的整数
【示例】
当前 Bar 对应的交易日为 2019-03-25，则 TradeDate 返回值为 20190325

HistoryDataExist
【说明】
指定合约的历史数据是否有效
【语法】
bool HistoryDataExist(string contractNo='', char kLineType='', int kLineValue=0)
【参数】
contractNo(string) 合约编号，默认基准合约
kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
【备注】
返回 Bool 值，有效返回 True，否则返回 False

【示例】
无

HisData
【说明】
获取各种历史数据数组
【语法】
numpy.array HisData(enum dataType, enum kLineType='', int kLineValue=0, string contractNo='', int maxLength=100)
【参数】
dataType(char) 指定历史数据的种类，可选的枚举函数和相应含义为：
Enum_Data_Close() : 收盘价
Enum_Data_Open() : 开盘价
Enum_Data_High() : 最高价
Enum_Data_Low(): 最低价
Enum_Data_Median(): 中间价
Enum_Data_Typical(): 标准价
Enum_Data_Weighted(): 加权收盘价
Enum_Data_Vol(): 成交量
Enum_Data_Opi(): 持仓量
Enum_Data_Time(): K 线时间
kLineType(char) 指定周期类型，可选的枚举函数和相应含义为：
Enum_Period_Tick(): 周期类型_分笔
Enum_Period_Min(): 周期类型_分钟
Enum_Period_Day(): 周期类型_日线
kLineValue(int) 周期数， 如：5 分钟线，周期数就是 5；50 秒线，周期数为 50
contractNo(string) 合约编号，为空时取当前合约
maxLength(int) 定返回历史数据数组的最大长度，默认值为 100
【备注】
1. 获取前要使用 SetBarInterval 订阅指定合约，指定周期，指定数量的历史数据，否则 HisData 取不到数据
2. 返回 numpy 数组，获取订阅的 maxLength 个指定的种类的历史数据
【示例】
# 获取白糖主连合约包含当前 Bar 在内的之前 1000 个 5 分钟线的收盘价
closeList = HisData(Enum_Data_Close(), Enum_Period_Min(), 5, "ZCE|Z|SR|MAIN", 1000)
closeList[-1] 表示当前 Bar 的收盘价，closeList[-2]表示上一个 Bar 的收盘价

HisBarsInfo
【说明】
获取最多 maxLength 根指定类型的历史 K 线详细数据
【语法】
list HisBarsInfo(string contractNo='', char kLineType='', int kLineValue=0, int maxLength=None)
【参数】
contractNo(string) 合约编号，为空时取当前合约

kLineType(char) K 线类型，可选值请参阅周期类型枚举函数
kLineValue(int) K 线周期
maxLength(int) 返回历史数据数组的最大长度，默认值为所有 K 线数据
若 contractNo, kLineType, kLineValue 同时不填，则取用于展示的合约及相应的 K 线类型、K 线周期
【备注】
返回列表，包括截止当前 Bar 的最多 maxLength 个 K 线的历史数据
列表中以字典的形式保存每个 K 线的数据，字典中每个键值的含义如下:
ContractNo 合约编号，如'NYMEX|Z|CL|MAIN'
DateTimeStamp 更新时间，如 20190521130800000
KLineIndex K 线索引，如 1
KLineQty K 线成交量，如 18
TotalQty 总成交量，如 41401
KLineSlice K 线周期， 如 1
KLineType K 线类型，如'M'
OpeningPrice 开盘价， 如 63.5
LastPrice 收盘价，如 63.49
SettlePrice 结算价，如 63.21
HighPrice 最高价，如 63.5
LowPrice 最低价， 如 63.49
PositionQty 总持仓，如 460816
TradeDate 交易日期，如 20190521
【示例】
```
# 获取合约 ZCE|Z|SR|MAIN 包含当前 Bar 在内的之前 1000 个历史 5 分钟 K 线 Bar 的详细信息
barList = HisBarsInfo("ZCE|Z|SR|MAIN", Enum_Period_Min(), 5, 1000)
# 当前 Bar 的详细信息
barInfo = barList[-1]
```

即时行情

Q_UpdateTime
【说明】
获取指定合约即时行情最近的更新时间
【语法】
string Q_UpdateTime(string contractNo='')
【参数】
contractNo(string) 合约编号, 默认当前合约
【备注】
返回格式为"YYYYMMDDHHMMSSmmm"的字符串，若指定合约即时行情最近的更新时间为 2019-05-21 10:07:46.000，则该函数放回为 20190521100746000
【示例】
无

Q_AskPrice
【说明】
合约最优卖价
【语法】
float Q_AskPrice(string contractNo='', int level=1)
【参数】
contractNo(string) 合约编号，默认当前合约
level(int) 档位数，默认 1 档
【备注】
返回浮点数, 可获取指定合约指定深度的最优卖价
【示例】
无

Q_AskVol
【说明】
合约最优卖量
【语法】
float Q_AskVol(string contractNo='', int level=1)
【参数】
contractNo(string) 合约编号，默认当前合约
level(int) 档位数，默认 1 档
【备注】
返回浮点数，可获取指定合约指定深度的最优卖量
【示例】
无

Q_AvgPrice
【说明】
当前合约的实时均价
【语法】
float Q_AvgPrice(string contractNo='')
【参数】
contractNo 合约编号，默认当前合约
【备注】
返回浮点数，返回实时均价即结算价
【示例】
无

Q_BidPrice
【说明】
合约最优买价
【语法】
float Q_BidPrice(string contractNo='', int level=1)

【参数】
contractNo(string) 合约编号，默认当前合约
level(int) 档位数，默认 1 档
【备注】
返回浮点数，可获取指定合约指定深度的最优买价
【示例】
无

Q_BidVol
【说明】
合约最优买量
【语法】
float Q_AskVol(string contractNo='', int level=1)
【参数】
contractNo(string) 合约编号，默认当前合约
level(int) 档位数，默认 1 档
【备注】
返回浮点数，可获取指定合约指定深度的最优买量
【示例】
无

Q_Close
【说明】
当日收盘价，未收盘则取昨收盘
【语法】
float Q_Close(string contractNo='')
【参数】
contractNo(string) 合约编号，默认当前合约
【备注】
返回浮点数
【示例】
无

Q_High
【说明】
当日最高价
【语法】
float Q_High(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_HisHigh
【说明】
历史最高价
【语法】
float Q_HisHigh(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数

【示例】
无

Q_HisLow
【说明】
历史最低价
【语法】
float Q_HisLow(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_Last
【说明】
最新价
【语法】
float Q_Last(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_LastDate
【说明】
最新成交日期
【语法】
int Q_LastDate(string contractNo='')
【参数】

contractNo(string) 合约编号
【备注】
返回当前公式应用商品的最新成交日期，格式为 YYYYMMDD 整数表示的日期
【示例】
无

Q_LastTime
【说明】
最新成交时间
【语法】
float Q_LastTime(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回当前公式应用商品的最新成交时间，以格式为 0.HHMMSSmmm 浮点数表示的时间
【示例】
无

Q_Low
【说明】
当日最低价
【语法】
float Q_Low(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_LowLimit
【说明】
当日跌停板价
【语法】
float Q_LowLimit(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_Open
【说明】

当日开盘价
【语法】
float Q_Open(string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_OpenInt
【说明】
持仓量
【语法】
float Q_ OpenInt (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数， 单位为手
【示例】
无

Q_PreOpenInt
【说明】
昨持仓量
【语法】
float Q_ PreOpenInt (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数， 单位为手
【示例】
无

Q_PreSettlePrice
【说明】
昨结算
【语法】
float Q_PreSettlePrice (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】

无

Q_PriceChg
【说明】
当日涨跌
【语法】
float Q_PriceChg (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_PriceChgRadio
【说明】
当日涨跌幅
【语法】
float Q_PriceChgRadio (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_TotalVol
【说明】
当日成交量
【语法】
float Q_ TotalVol (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_TurnOver
【说明】
当日成交额
【语法】
float Q_ TurnOver (string contractNo='')
【参数】

contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_UpperLimit
【说明】
当日涨停板价
【语法】
float Q_ UpperLimit (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_TheoryPrice
【说明】
当日期权理论价
【语法】
float Q_ TheoryPrice (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数
【示例】
无

Q_Sigma
【说明】
当日期权波动率
【语法】
float Q_ Sigma (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】
无

Q_Delta
【说明】

当日期权 Delta
【语法】
float Q_Delta (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】
无

Q_Gamma
【说明】
当日期权 Gamma
【语法】
float Q_Gamma (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】
无

Q_Vega
【说明】
当日期权 Vega
【语法】
float Q_Vega (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】
无

Q_Theta
【说明】
当日期权 Theta
【语法】
float Q_ Theta (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】

无

Q_Rho
【说明】
当日期权 Rho
【语法】
float Q_ Rho (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回浮点数，不存在时返回 None
【示例】
无

QuoteDataExist
【说明】
行情数据是否有效
【语法】
float QuoteDataExist (string contractNo='')
【参数】
contractNo(string) 合约编号
【备注】
返回 Bool 值，数据有效返回 True，否则 False
【示例】
无

CalcTradeDate
【说明】
计算指定合约，指定时间戳所属的交易日
【语法】
int CalcTradeDate(string contractNo='', int dateTimeStamp=0)
【参数】
contractNo(string) 合约编号，默认基准合约
dateTimeStamp(int) 时间戳，默认 0
【备注】
正常情况，返回指定合约指定时间戳所属的交易日
当返回值为-1 时，表示合约参数有误
当返回值为-2 时，表示时间戳参数有误，比如传入非交易时段时间戳
【示例】
CalcTradeDate(dateTimeStamp=20190830110000000)
CalcTradeDate('ZCE|Z|SR|MAIN', 20190830110000000)

属性函数

BarInterval
【说明】
返回界面合约图表 K 线周期数值
【语法】
int BarInterval()
【参数】
无
【备注】
返回界面图表 K 线周期数值，通常和 BarType 一起使用进行数据周期的判别
【示例】
当前数据周期为 1 日线，BarInterval()返回值为 1
当前数据周期为 22 日线，BarInterval()返回值为 22
当前数据周期为 60 分钟线，BarInterval()返回值为 60
当前数据周期为 1TICK 线，BarInterval()等于 1；br 大于当前数据周期为 5000 量线，BarInterval 等于 5000

BarType
【说明】
返回界面合约 K 线图表周期类型字符
【语法】
char BarType()
【参数】
无
【备注】
返回值为字符，通常和 BarInterval 一起使用进行数据周期的判别
返回值如下定义：
'T' 分笔
'S' 秒线
'M' 分钟
'D' 日线

BidAskSize
【说明】
行情深度
【语法】
int BidAskSize(string contractNo='')
【参数】
contractNo(string): 合约编号，为空时，取基准合约
【备注】
返回整型
【示例】

郑商所白糖的行情深度为 5，因此其 BidAskSize 等于 5。

ContractUnit
【说明】
每张合约包含的基本单位数量, 即每手乘数
【语法】
int ContractUnit(string contractNo='')
【参数】
contractNo(string): 合约编号，为空时，取基准合约
【备注】
返回整型，1 张合约包含多少标的物。
【示例】
无

ExchangeName
【说明】
合约对应交易所名称代码
【语法】
string ExchangeName(string contractNo='')
【参数】
contractNo(string): 合约编号，为空时，取基准合约
【备注】
返回字符串
【示例】
# 郑商所下各合约的交易所名称为："ZCE"
ExchangeName("ZCE|Z|TA|MAIN")

ExchangeTime
【说明】
交易所时间
【语法】
string ExchangeTime(string contractNo)
【参数】
exchangeNo(string): 交易所编号，例如"ZCE","DCE","SHFE","CFFEX","INE"
【备注】
返回字符串 "2019-07-05 22:11:00"
当交易所编号为无效编号时，返回空字符串
该函数返回的时间是系统时间
【示例】
ExchangeTime('ZCE')

ExchangeStatus
【说明】
交易所状态

【语法】
string ExchangeStatus(string exchangeNo)
【参数】
exchangeNo(string): 交易所编号，例如"ZCE","DCE","SHFE","CFFEX","INE"
【备注】
返回字符
'N' 未知状态
'I' 正初始化
'R' 准备就绪
'0' 交易日切换
'1' 竞价申报
'2' 竞价撮合
'3' 连续交易
'4' 交易暂停
'5' 交易闭市
'6' 竞价暂停
'7' 报盘未连
'8' 交易未连
'9' 闭市处理
【示例】
ExchangeStatus('ZCE')

CommodityStatus
【说明】
品种或合约交易状态
【语法】
string CommodityStatus(string commodityNo|string contractNo)
【参数】
commodityNo(string): 品种编号，例如"ZCE|F|SR", "DCE|F|I"
或者
contractNo(string): 合约编号，例如"ZCE|F|SR|001", "DCE|F|I|2001"
【备注】
返回字符
'N' 未知状态
'I' 正初始化
'R' 准备就绪
'0' 交易日切换
'1' 竞价申报
'2' 竞价撮合
'3' 连续交易
'4' 交易暂停
'5' 交易闭市
'6' 竞价暂停
'7' 报盘未连

'8'　交易未连
'9'　闭市处理
【示例】
CommodityStatus('ZCE|F|SR')

GetSessionCount
【说明】
获取交易时间段的个数
【语法】
int GetSessionCount(string contractNo='')
【参数】
contractNo(string): 合约编号，为空时，取基准合约
【备注】
返回整型
【示例】
无

GetSessionStartTime
【说明】
获取合约指定交易时间段的开始时间。
【语法】
float GetSessionStartTime(string contractNo='', int index=0)
【参数】
contractNo(string) 合约编号，为空时，取基准合约
index(int) 交易时间段的索引值，从 0 开始
【备注】
返回指定合约的交易时间段开始时间，格式为 0.HHMMSS 的浮点数。
【示例】
无

GetSessionEndTime
【说明】
获取合约指定交易时间段的结束时间。
【语法】
float GetSessionEndTime(string contractNo='', int index=0)
【参数】
contractNo(string) 合约编号，为空时，取基准合约。
index(int) 交易时间段的索引值，从 0 开始。
【备注】
返回指定合约的交易时间段结束时间，格式为 0.HHMMSS 的浮点数。
【示例】
contractNo = "ZCE|Z|SR|MAIN"
sessionCount = GetSessionCount(contractNo)
for i in range(0, sessionCount-1):

sessionEndTime = GetSessionEndTime(contractNo, i)

由于合约 ZCE|F|TA|MAIN 的第三段交易结束时间为 11:30:00，
所以 GetSessionEndTime("ZCE|F|TA|MAIN", 2)的返回值为 0.113

TradeSessionBeginTime
【说明】
获取指定合约指定交易日的指定交易时间段的开始时间戳。
【语法】
int TradeSessionBeginTime(string contractNo='', int tradeDate=0, int index=0)
【参数】
contractNo(string) 合约编号，为空时，取基准合约
tradeDate(int) 指定的交易日，默认 0
index(int) 交易时间段的索引值，从 0 开始，默认取第一个交易时段。
【备注】
返回时间戳类型，如 20190904213000000
【示例】
无

TradeSessionEndTime
【说明】
获取指定合约指定交易日的指定交易时间段的结束时间戳。
【语法】
int TradeSessionEndTime(string contractNo='', int tradeDate=0, int index=-1)
【参数】
contractNo(string) 合约编号，为空时，取基准合约
tradeDate(int) 指定的交易日，默认 0
index(int) 交易时间段的索引值，从 0 开始，默认取最后一个交易时段
【备注】
返回时间戳类型，如 20190904213000000
【示例】
无

GetNextTimeInfo
【说明】
获取指定合约指定时间点的下一个时间点及交易状态。
【语法】
dict GetNextTimeInfo(string contractNo, float timeStamp)
【参数】
contractNo(string) 合约编号
timeStr(float) 指定的时间点，格式为 0.HHMMSS
【备注】
返回时间字典，结构如下：
{

'Time' : 0.21,
'TradeState' : 3
}
其中 Time 对应的值表示指定时间 timeStamp 的下一个时间点，返回指定合约的交易时间段开始时间，格式为 0.HHMMSS 的浮点数。
TradeState 表示对应时间点的交易状态，数据类型为字符，可能出现的值及相应的状态含义如下：
1：集合竞价
2：集合竞价撮合
3：连续交易
4：暂停
5：闭市
6：闭市处理时间
0：交易日切换时间
N：未知状态
I：正初始化
R：准备就绪
异常情况返回为空字典：{}
【示例】

```
# 获取 22:00:00 后下一个时间点的时间和交易状态
GetNextTimeInfo('SHFE|Z|CU|MAIN', 0.22)
# 获取当前时间下一个时间点的时间和交易状态
import time # 需要在策略头部添加 time 库
curTime = time.strftime('0.%H%M%S', time.localtime(time.time()))
timeInfoDict = GetNextTimeInfo('SHFE|Z|CU|MAIN', curTime)
```

CurrentDate
【说明】
公式处于历史阶段时，返回历史 K 线当时的日期。处于实时阶段时，返回客户端所在操作系统的日期
【语法】

```
int CurrentDate()
```

【参数】
无
【备注】
格式为 YYMMDD 的整数。
【示例】
如果当前日期为 2019-7-13，CurrentDate()返回值为 20190713

CurrentTime
【说明】
公式处于历史阶段时，返回历史 K 线当时的时间。处于实时阶段时，返回客户端所在操作系统的时间
【语法】

float CurrentTime()
【参数】
无
【备注】
格式为 0.HHMMSS 的浮点数。
【示例】
如果当前时间为 11:34:21，CurrentTim()返回值为 0.113421。

TimeDiff
【说明】
返回两个时间之间的间隔秒数，忽略日期差异
【语法】
int TimeDiff(self, float datetime1, float datetime2)
【参数】
datetime1(float) 输入较早时间
datetime2(float) 输入较晚时间
【备注】
该函数只计算两个时间之间的差值，不考虑两个参数的日期
【示例】
# 返回两时间相差 10 秒
TimeDiff(20190404.104110, 20110404.104120)
# 返回两时间相差 2 分钟，即 120 秒
TimeDiff(20190404.1041, 20110404.1043)

IsInSession
【说明】
操作系统的当前时间是否为指定合约的交易时间。
【语法】
bool IsInSession(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基础合约
【备注】
获取操作系统的当前时间是否为指定合约的交易时间。
【示例】
如果当前时间为 11:34:21，IsInSession("ZCE|Z|TA|MAIN")返回值为 False。

MarginRatio
【说明】
获取合约默认保证金比率
【语法】
float MarginRatio(string contractNo='')
【参数】
contractNo 合约编号，为空时，取基准合约
【备注】

返回浮点数
【示例】
无

MaxBarsBack
【说明】
最大回溯 Bar 数
【语法】
float MaxBarsBack()
【参数】
无
【备注】
返回浮点数
【示例】
无

MaxSingleTradeSize
【说明】
单笔交易限量
【语法】
int MaxSingleTradeSize()
【参数】
无
【备注】
返回整型，单笔交易限量，对于不能交易的商品，返回-1，对于无限量的商品，返回 0
【示例】
无

PriceTick
【说明】
合约最小变动价
【语法】
int PriceTick(string contractNo='')
【参数】
contractNo(string) 合约编号，为空时，取基准合约
【备注】
无
【示例】
沪铝的最小变动价为 5，因此其 PriceTick 等于 5

OptionType
【说明】
返回期权的类型，是看涨还是看跌期权
【语法】

int OptionType(string contractNo='')
【参数】
contractNo(string) 合约编号，为空时，取基准合约
【备注】
返回整型，0 为看涨，1 为看跌，-1 为异常。
【示例】
无

PriceScale
【说明】
合约价格精度
【语法】
float PriceScale(string contractNo='')
【参数】
contractNo(string) 合约编号，为空时，取基准合约
【备注】
返回浮点数
【示例】
上期沪金的报价精确到小数点 2 位，则 PriceScale 为 1/100，PriceScale 的返回值为 0.01

Symbol
【说明】
获取展示合约，即基准合约的编号
【语法】
string Symbol()
【参数】
无
【备注】
期货、现货、指数：
<EXG>|<TYPE>|<ROOT>|<YEAR><MONTH>[DAY]
期权 ：
<EXG>|<TYPE>|<ROOT>|<YEAR><MONTH>[DAY]<CP><STRIKE>
跨期套利:
<EXG>|<TYPE>|<ROOT>|<YEAR><MONTH>[DAY]|<YEAR><MONTH>[DAY]
跨品种套利 ：
<EXG>|<TYPE>|<ROOT&ROOT>|<YEAR><MONTH>[DAY]
极星跨期套利 ：
<EXG>|s|<ROOT>|<YEAR><MONTH>[DAY]|<YEAR><MONTH>[DAY]
极星跨品种套利 ：
<EXG>|m|<ROOT-ROOT>|<YEAR><MONTH>|<YEAR><MONTH>
极星现货期货套利:
<EXG>|p|<ROOT-ROOT>||<YEAR><MONTH>
【示例】
"ZCE|F|SR|001", "ZCE|O|SR|001C5000"

SymbolName
【说明】
获取合约名称
【语法】
string SymbolName(string contractNo='')
【参数】
contractNo(string) 合约编号，为空时，取基准合约
【备注】
返回字符串
【示例】
"ZCE|F|SR|001"的合约名称为"白糖 001"

SymbolType
【说明】
获取合约所属的品种编号
【语法】
string SymbolType(string contractNo='')
【参数】
contractNo(string) 合约编号，为空时，取基准合约
【备注】
返回字符串
【示例】
"ZCE|F|SR|001"的品种编号为"ZCE|F|SR"

GetTrendContract
【说明】
获取商品主连/近月对应的合约
【语法】
string GetTrendContract(string contractNo="")
【参数】
contractNo(string) 取商品主连/近月编号，为空时，取基准合约
【备注】
返回字符串，若 contractNo 为具体的合约，则返回 contractNo
【示例】
GetTrendContract('DCE|Z|I|MAIN') 的返回为"DCE|F|I|1909"
GetTrendContract('DCE|Z|I|NEARBY') 的返回为"DCE|F|I|1907"

策略状态

AvgEntryPrice
【说明】
获得当前持仓指定合约的平均建仓价格。

【语法】
float AvgEntryPrice(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
无
【示例】
无

BarsSinceEntry
【说明】
获得当前持仓中指定合约的第一个建仓位置到当前位置的 Bar 计数。
【语法】
int BarsSinceEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓指定合约的第一个建仓位置到当前位置的 Bar 计数，返回值为整型。
只有当 MarketPosition != 0 时，即有持仓的状况下，该函数才有意义，否则返回-1。
注意：在开仓 Bar 上为 0。
【示例】
无

BarsSinceExit
【说明】
获得当前持仓中指定合约的最近平仓位置到当前位置的 Bar 计数。
【语法】
int BarsSinceExit(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓指定合约的最近平仓位置到当前位置的 Bar 计数，返回值为整型。
若从未平过仓，则返回-1。
注意：在平仓 Bar 上为 0。
【示例】
无

BarsSinceLastEntry
【说明】
获得当前持仓的最后一个建仓位置到当前位置的 Bar 计数。
【语法】
int BarsSinceLastEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约

【备注】
获得当前持仓指定合约的最后一个建仓位置到当前位置的 Bar 计数，返回值为整型。
若当前策略持仓为 0，则返回-1。
注意：在建仓 Bar 上为 0。
【示例】
无

BarsSinceLastBuyEntry
【说明】
获得当前持仓的最后一个 Buy 建仓位置到当前位置的 Bar 计数。
【语法】
int BarsSinceLastBuyEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓指定合约的最后一个 Buy 建仓位置到当前位置的 Bar 计数，返回值为整型。
若当前策略持仓为 0，则返回-1。
注意：在建仓 Bar 上为 0。
【示例】
无

BarsSinceLastSellEntry
【说明】
获得当前持仓的最后一个 Sell 建仓位置到当前位置的 Bar 计数。
【语法】
int BarsSinceLastSellEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓指定合约的最后一个 Sell 建仓位置到当前位置的 Bar 计数，返回值为整型。
若当前策略持仓为 0，则返回-1。
注意：在建仓 Bar 上为 0。
【示例】
无

BarsSinceToday
【说明】
获得当天的第一根 Bar 到当前的 Bar 个数。
【语法】
int BarsSinceToday(string contractNo='', char kLineType='', int kLineValue='')
【参数】
contractNo(string) 合约编号，默认为基准合约
kLineType(char) K 线类型
kLineValue(int) K 线周期

【备注】
无
【示例】
无

ContractProfit
【说明】
获得当前持仓的每手浮动盈亏。
【语法】
float ContractProfit(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓位置的每手浮动盈亏，返回值为浮点数。
【示例】
无

CurrentContracts
【说明】
获得策略当前的持仓合约数(净持仓)。
【语法】
int CurrentContracts(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得策略当前的持仓合约数，返回值为整数。
该函数返回策略当前的净持仓数量，多仓为正值，空仓为负值，持平返回 0。
【示例】
无

BuyPosition
【说明】
获得当前持仓的买入方向的持仓量。
【语法】
int BuyPosition(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得策略当前持仓的买入方向的持仓量，返回值为整数
【示例】
无

SellPosition
【说明】

获得当前持仓的卖出方向的持仓量。
【语法】
int SellPosition(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得策略当前持仓的卖出持仓量，返回值为整数。
【示例】
无

EntryDate
【说明】
获得当前持仓的第一个建仓位置的日期。
【语法】
int EntryDate(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
若策略当前持仓为 0，则返回无效日期：19700101，否则返回 YYYYMMDD 格式的日期。
【示例】
无

EntryPrice
【说明】
获得当前持仓的第一次建仓的委托价格。
【语法】
float EntryPrice(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的第一个建仓价格，返回值为浮点数。
若策略当前持仓为 0，则返回 0。
【示例】
无

EntryTime
【说明】
获得当前持仓的第一个建仓位置的时间。
【语法】
float EntryTime(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的第一个建仓时间，返回值为 0.HHMMSSmmm 格式的时间。

若策略当前持仓为 0，则返回 0。
【示例】
无

ExitDate
【说明】
获得最近平仓位置 Bar 日期。
【语法】
int ExitDate(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的最近平仓时间，返回值为 YYYYMMDD 格式的日期。
若从未平过仓，则返回无效日期：19700101。
【示例】
无

ExitPrice
【说明】
获得合约最近一次平仓的委托价格。
【语法】
float ExitPrice(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得最近平仓位置的平仓价格，返回值为浮点数。
若合约从未被平仓,则返回 0，否则返回合约最近一次平仓时的委托价格。
【示例】
无

ExitTime
【说明】
获得最近平仓位置 Bar 时间。
【语法】
float ExitTime(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得最近平仓位置 Bar 时间，返回值为 0.HHMMSSmmm 格式的时间。
若合约从未平过仓，则返回 0。
【示例】
无

LastEntryDate

【说明】
获得当前持仓的最后一个建仓位置的日期。
【语法】
int LastEntryDate(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的最后一个建仓位置的日期，返回值为 YYYYMMDD 格式的日期。
若策略当前持仓为 0，则返回无效日期:19700101。
【示例】
无

LastEntryPrice
【说明】
获得当前持仓的最后一次建仓的委托价格。
【语法】
float LastEntryPrice(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的最后一个建仓价格，返回值为浮点数。
若策略当前持仓为 0，则返回 0。
【示例】
无

LastBuyEntryPrice
【说明】
获得当前 Buy 持仓的最后一次建仓的委托价格。
【语法】
float LastBuyEntryPrice(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Buy 持仓的最后一个建仓价格，返回值为浮点数。
若策略当前 Buy 持仓为 0，则返回 0。
【示例】
无

LastSellEntryPrice
【说明】
获得当前 Sell 持仓的最后一次建仓的委托价格。
【语法】
float LastSellEntryPrice(string contractNo='')
【参数】

contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Sell 持仓的最后一个建仓价格，返回值为浮点数。
若策略当前 Sell 持仓为 0，则返回 0。
【示例】
无

HighestSinceLastBuyEntry
【说明】
获得当前 Buy 持仓的最后一次建仓以来的最高价。
【语法】
float HighestSinceLastBuyEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Buy 持仓的最后一个建仓以来的最高价，返回值为浮点数。
若策略当前 Buy 持仓为 0，则返回 0。
【示例】
无

LowestSinceLastBuyEntry
【说明】
获得当前 Buy 持仓的最后一次建仓以来的最低价。
【语法】
float LowestSinceLastBuyEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Buy 持仓的最后一个建仓以来的最低价，返回值为浮点数。
若策略当前 Buy 持仓为 0，则返回 0。
【示例】
无

HighestSinceLastSellEntry
【说明】
获得当前 Sell 持仓的最后一次建仓以来的最高价。
【语法】
float HighestSinceLastSellEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Sell 持仓的最后一个建仓以来的最高价，返回值为浮点数。
若策略当前 Sell 持仓为 0，则返回 0。
【示例】

无

LowestSinceLastSellEntry
【说明】
获得当前 Sell 持仓的最后一次建仓以来的最低价。
【语法】
float LowestSinceLastSellEntry(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前 Sell 持仓的最后一个建仓以来的最低价，返回值为浮点数。
若策略当前 Sell 持仓为 0，则返回 0。
【示例】
无

LastEntryTime
【说明】
获得当前持仓的最后一个建仓位置的时间。
【语法】
float LastEntryTime(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓的最后一个建仓位置的时间，返回值为 0.HHMMSSmmm 格式的时间。
若策略当前持仓为 0，则返回 0。
【示例】
无

MarketPosition
【说明】
获得当前持仓状态 。
【语法】
int MarketPosition(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
获得当前持仓状态，返回值为整型。
返回值定义如下：
-1 当前位置为持空仓
0 当前位置为持平
1 当前位置为持多仓
【示例】
```
# 判断合约 ZCE|F|SR|905 当前是否持多仓
if(MarketPosition("ZCE|F|SR|905")==1): pass
```

# 判断合约 ZCE|F|SR|905 当前是否有持仓，无论持空仓或多仓
if(MarketPosition("ZCE|F|SR|905")!=0): pass

PositionProfit
【说明】
获得当前持仓的浮动盈亏 。
【语法】
float PositionProfit(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
若策略当前持仓为 0，则返回 0
【示例】
无

BarsLast
【说明】
返回最后一次满足条件时距离当前的 bar 数
【语法】
int BarsLast(bool condition)
【参数】
Condition(bool) 传入的条件表达式
【备注】
返回最后一次满足条件时距离当前的 bar 数。
【示例】
# 从当前 Bar 开始，最近出现 Close>Open 的 Bar 到当前 Bar 的偏移值。如果为 0，即当前 Bar 为最近的满足条件的 Bar。
BarsLast(Close() > Open())

StrategyId
【说明】
获取当前策略 Id
【语法】
int StrategyId()
【参数】
无
【备注】
无
【示例】
无

策略性能

Available
【说明】
返回策略当前可用虚拟资金。
【语法】
float Available()
【参数】
无
【备注】
无
【示例】
无

CurrentEquity
【说明】
返回策略的当前账户权益。
【语法】
float CurrentEquity()
【参数】
无
【备注】
无
【示例】
无

FloatProfit
【说明】
返回指定合约的浮动盈亏。
【语法】
float FloatProfit(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
无
【示例】
无

GrossLoss
【说明】
返回累计总亏损。
【语法】
float GrossLoss()

【参数】
无
【备注】
无
【示例】
无

GrossProfit
【说明】
返回累计总利润。
【语法】
float GrossProfit()
【参数】
无
【备注】
无
【示例】
无

Margin
【说明】
返回指定合约的持仓保证金。
【语法】
float Margin(string contractNo='')
【参数】
contractNo(string) 合约编号，默认为基准合约
【备注】
无
【示例】
无

NetProfit
【说明】
返回该账户下的平仓盈亏。
【语法】
float NetProfit()
【参数】
无
【备注】
无
【示例】
无

NumAllTimes

【说明】
返回该账户的开仓次数。
【语法】
int NumAllTimes()
【参数】
无
【备注】
无
【示例】
无

NumWinTimes
【说明】
返回该账户的盈利次数。
【语法】
int NumWinTimes()
【参数】
无
【备注】
无
【示例】
无

NumLoseTimes
【说明】
返回该账户的亏损次数。
【语法】
int NumLoseTimes()
【参数】
无
【备注】
无
【示例】
无

NumEventTimes
【说明】
返回该账户的保本次数。
【语法】
int NumEventTimes()
【参数】
无
【备注】
无

【示例】
无

PercentProfit
【说明】
返回该账户的盈利成功率。
【语法】
float PercentProfit()
【参数】
无
【备注】
无
【示例】
无

TradeCost
【说明】
返回该账户交易产生的手续费。
【语法】
float TradeCost()
【参数】
无
【备注】
无
【示例】
无

TotalTrades
【说明】
返回该账户的交易总开仓手数。
【语法】
int TotalTrades()
【参数】
无
【备注】
无
【示例】
无

账户函数

A_AccountID
【说明】

返回当前公式应用的交易帐户 ID。
【语法】
string A_AccountID()
【参数】
无
【备注】
返回当前公式应用的交易帐户 ID，返回值为字符串，无效时返回空串。
注：不能用于历史测试，仅适用于实时行情交易。
【示例】
无

A_AllAccountID
【说明】
返回所有已登录交易帐户 ID。
【语法】
list A_AllAccountID()
【参数】
无
【备注】
没有账号登录时，返回空列表
注：不能用于历史测试，仅适用于实时行情交易。
【示例】
无

A_GetAllPositionSymbol
【说明】
获得指定账户所有持仓合约。
【语法】
list A_GetAllPositionSymbol(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
该参数返回类型为字符串列表，列表内容为账户所有持仓合约列表。
【示例】
无

A_Cost
【说明】
返回指定交易帐户的手续费。
【语法】
string A_Cost(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】

返回指定交易帐户的手续费，返回值为浮点数。
【示例】
无

A_Assets
【说明】
返回指定交易帐户的动态权益。
【语法】
float A_Assets(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的动态权益，返回值为浮点数。
【示例】
无

A_Available
【说明】
返回指定交易帐户的可用资金。
【语法】
float A_Available(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的可用资金，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_Margin
【说明】
返回指定交易帐户的持仓保证金。
【语法】
float A_Margin(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的持仓保证金，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_ProfitLoss
【说明】

返回指定交易帐户的浮动盈亏。
【语法】
float A_ProfitLoss(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的浮动盈亏，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_CoverProfit
【说明】
返回当前账户的平仓盈亏。
【语法】
float A_CoverProfit(string userNo='')
【参数】
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的平仓盈亏，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_TotalFreeze
【说明】
返回指定交易帐户的冻结资金。
【语法】
float A_TotalFreeze(string userNo='')
【参数】
userNo 指定的交易账户，默认当前账户
【备注】
返回指定交易帐户的冻结资金，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_BuyAvgPrice
【说明】
返回指定帐户下当前商品的买入持仓均价。
【语法】
float A_BuyAvgPrice(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号

userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的买入持仓均价，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_BuyPosition
【说明】
返回指定帐户下当前商品的买入持仓。

【语法】
float A_BuyPosition(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的买入持仓，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
当前持多仓 2 手，A_BuyPosition 返回 2。

A_BuyPositionCanCover
【说明】
返回指定帐户下买仓可平数量。
【语法】
int A_BuyPositionCanCover(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
可平仓数量=持仓数量-已排队的挂单数量
【示例】
无

A_BuyProfitLoss
【说明】
返回指定帐户下当前商品的买入持仓盈亏。
【语法】
float A_BuyProfitLoss(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】

返回指定帐户下当前商品的买入持仓盈亏，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_SellAvgPrice
【说明】
返回指定帐户下当前商品的卖出持仓均价。
【语法】
float A_SellAvgPrice(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的卖出持仓均价，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_SellPosition
【说明】
返回指定帐户下当前商品的卖出持仓。
【语法】
float A_SellPosition(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的卖出持仓，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
当前持空仓 3 手，A_SellPosition 返回 3。

A_SellPositionCanCover
【说明】
返回指定帐户下卖仓可平数量。

【语法】
int A_SellPositionCanCover(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
可平仓数量=持仓数量-已排队的挂单数量

【示例】
无

A_SellProfitLoss
【说明】
返回指定帐户下当前商品的卖出持仓盈亏。
【语法】
float A_SellProfitLoss(string contractNo='', string userNo='')
【参数】
contractNo(string)指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的卖出持仓盈亏，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_TotalAvgPrice
【说明】
返回指定帐户下当前商品的持仓均价。

【语法】
float A_TotalAvgPrice(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的持仓均价，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_TotalPosition
【说明】
返回指定帐户下当前商品的总持仓。
【语法】
int A_TotalPosition(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的总持仓，返回值为浮点数。
该持仓为所有持仓的合计值，正数表示多仓，负数表示空仓，零为无持仓。
注：不能使用于历史测试，仅适用于实时行情交易。

【示例】
无

A_TotalProfitLoss
【说明】
返回指定帐户下当前商品的总持仓盈亏。
【语法】
float A_TotalProfitLoss(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的总持仓盈亏，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_TodayBuyPosition
【说明】
返回指定帐户下当前商品的当日买入持仓。
【语法】
float A_TodayBuyPosition(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的当日买入持仓，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_TodaySellPosition
【说明】
返回指定帐户下当前商品的当日卖出持仓。
【语法】
float A_TodaySellPosition(string contractNo='', string userNo='')
【参数】
contractNo(string) 指定商品的合约编号，为空时采用基准合约编号
userNo(string) 指定的交易账户，默认当前账户
【备注】
返回指定帐户下当前商品的当日卖出持仓，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderBuyOrSell

【说明】

返回指定帐户下当前商品的某个委托单的买卖类型。

【语法】

char A_OrderBuyOrSell(int|string localOrderId='')

【参数】

localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号

【备注】

返回指定帐户下当前商品的某个委托单的买卖类型，返回值为：

"B"：买入

"S"：卖出

"A"：双边

该函数返回值可以与 Enum_Buy、Enum_Sell 等买卖状态枚举值进行比较，根据类型不同分别处理。

注：不能使用于历史测试，仅适用于实时行情交易。

【示例】

```
nBorS = A_OrderBuyOrSell('1-1')
if nBorS == Enum_Buy():
    pass
```

A_OrderEntryOrExit

【说明】

返回指定帐户下当前商品的某个委托单的开平仓状态。

【语法】

char A_OrderEntryOrExit(int|string localOrderId='')

【参数】

localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号

【备注】

返回指定帐户下当前商品的某个委托单的开平仓状态，返回值：

"N"：无

"O"：开仓

"C"：平仓

"T"：平今

"1"：开平，应价时有效，本地套利也可以

"2"：平开，应价时有效，本地套利也可以

该函数返回值可以与 Enum_Entry、Enum_Exit 等开平仓状态枚举值进行比较，根据类型不同分别处理。

注：不能使用于历史测试，仅适用于实时行情交易。

【示例】

```
orderFlag = A_OrderEntryOrExit('1-1')
if orderFlag == Enum_Exit(): pass
```

A_OrderFilledLot

【说明】
返回指定帐户下当前商品的某个委托单的成交数量。
【语法】
float A_OrderFilledLot(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的成交数量，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderFilledPrice
【说明】
返回指定帐户下当前商品的某个委托单的成交价格。
【语法】
float A_OrderFilledPrice(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的成交价格，返回值为浮点数。
该成交价格可能为多个成交价格的平均值。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderLot
【说明】
返回指定帐户下当前商品的某个委托单的委托数量。
【语法】
float A_OrderLot(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的委托数量，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderPrice
【说明】
返回指定帐户下当前商品的某个委托单的委托价格。
【语法】
float A_OrderPrice(int|string localOrderId='')

【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的委托价格，返回值为浮点数。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderStatus
【说明】
返回指定帐户下当前商品的某个委托单的状态。
【语法】
char A_OrderStatus(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的状态，返回值：
"N"：无
"0"：已发送
"1"：已受理
"2"：待触发
"3"：已生效
"4"：已排队
"5"：部分成交
"6"：完全成交
"7"：待撤
"8"：待改
"9"：已撤单
"A"：已撤余单
"B"：指令失败
"C"：待审核
"D"：已挂起
"E"：已申请
"F"：无效单
"G"：部分触发
"H"：完全触发
"I"：余单失败
该函数返回值可以与委托状态枚举函数 Enum_Sended、Enum_Accept 等函数进行比较，根据类型不同分别处理。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_OrderIsClose

【说明】
判断某个委托单是否完结。
【语法】
bool A_OrderIsClose(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
当委托单是完结状态，返回 True，否则返回 False。
【示例】
无

A_OrderTime
【说明】
返回指定公式应用的帐户下当前商品的某个委托单的委托时间。
【语法】
struct_time A_OrderTime(int|string localOrderId='')
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回指定帐户下当前商品的某个委托单的委托时间，返回格式为 YYYYMMDD.hhmmss 的数值。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_FirstOrderNo
【说明】
返回指定账户第一个订单号。
【语法】
int A_FirstOrderNo(string contractNo1='', string contractNo2='', string userNo='')
【参数】
contractNo1(string) 合约代码，默认为遍历所有合约
contractNo2(string) 合约代码，默认为遍历所有合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有任何订单，否则，返回第一个订单的索引值，
该函数经常和 A_NextOrderNo 函数合用，用于遍历所有的订单。
【示例】
无

A_NextOrderNo
【说明】
返回指定账户下一个订单号。
【语法】

int A_NextOrderNo(int localOrderId=0, string contractNo1='', string contractNo2='', string userNo='')
【参数】
localOrderId(int) 定单号，默认为 0
contractNo1(string) 合约代码，默认为遍历所有合约
contractNo2(string) 合约代码，默认为遍历所有合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有任何订单，否则，返回处在 OrderNo 后面的订单索引值，
该函数常和 A_FirstOrderNo 联合使用。
【示例】
无

A_LastOrderNo
【说明】
返回指定账户最近发送的订单号。
【语法】
int A_LastOrderNo(string contractNo1='', string contractNo2='', string userNo='')
【参数】
contractNo1(string) 合约代码，默认为遍历所有合约
contractNo2 合约代码，默认为遍历所有合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有任何订单，否则，返回最后一个订单的索引值。
【示例】
无

A_FirstQueueOrderNo
【说明】
返回指定账户第一个排队(可撤)订单号。
【语法】
int A_FirstQueueOrderNo(string contractNo1='', string contractNo2='', string userNo='')
【参数】
contractNo1(string) 合约代码，默认为遍历所有合约
contractNo2(string) 合约代码，默认为遍历所有合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有任何可撤排队订单，否则，返回第一个订单的索引值。
该函数经常和 A_NextQueueOrderNo 函数合用，用于遍历排队中的订单。
【示例】
无

A_NextQueueOrderNo
【说明】

返回指定账户下一个排队(可撤)订单号。
【语法】
int A_NextQueueOrderNo(int localOrderId=0, string contractNo1='', string contractNo2='', string userNo='')
【参数】
localOrderId(int) 定单号，默认为 0
contractNo1(string) 合约代码，默认为遍历所有合约
contractNo2(string) 合约代码，默认为遍历所有合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有任何排队订单，否则，返回处在 OrderNo 后面的订单索引值，该函数常和 A_FirstQueueOrderNo 联合使用。
【示例】
无

A_AllQueueOrderNo
【说明】
返回指定账户所有排队(可撤)订单号。
【语法】
list A_AllQueueOrderNo(string contractNo='', string userNo='')
【参数】
contractNo(string) 合约代码，默认为遍历所有合约，指定后只遍历指定合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为空列表，表示没有任何排队订单，否则，返回包含处于排队中的委托定单号的列表。
【示例】
无

A_LatestFilledTime
【说明】
返回指定账户最新一笔完全成交委托单对应的时间。
【语法】
float A_LatestFilledTime(string contractNo='', string userNo='')
【参数】
contractNo(string) 合约代码，默认为遍历所有合约，指定后只遍历指定合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
若返回值为-1，表示没有对应的完全成交的委托，否则，返回最新一笔完全成交委托单对应的时间，返回格式为 YYYYMMDD.hhmmss 的数值。
【示例】
无

A_AllOrderNo

【说明】
返回包含指定合约指定账户所有订单号的列表。
【语法】
list A_AllOrderNo(string contractNo='', string userNo='')
【参数】
contractNo(string) 合约代码，默认为遍历所有合约，指定后只遍历指定合约
userNo(string) 指定的交易账户，默认当前账户
【备注】
无
【示例】
无

A_OrderContractNo
【说明】
返回订单的合约号。
【语法】
string A_OrderContractNo(int|string localOrderId=0)
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
返回结果如："ZCE|F|TA|305"等，
如果 localOrderId 没有对应的委托单，则返回结果为字符串。
【示例】
无

A_SendOrder
【说明】
针对指定的帐户、商品发送委托单。

【语法】
Int, string A_SendOrder(char orderDirct, char entryOrExit, int orderQty, float orderPrice, string contractNo='', string userNo='', char orderType='2', char validType='0', char hedge='T', char triggerType='N', char triggerMode='N', char triggerCondition='N', float triggerPrice=0)
【参数】
orderDirct(char) 发送委托单的买卖类型，取值为 Enum_Buy 或 Enum_Sell 之一
entryOrExit(char) 发送委托单的开平仓类型，取值为 Enum_Entry,Enum_Exit,Enum_ExitToday 之一
orderQty(int) 委托单的交易数量
orderPrice(float) 委托单的交易价格
contractNo(string) 商品合约编号，默认值为基准合约
userNo(string) 指定的账户名称，默认为界面选定的账户名称
orderType(char) 订单类型，字符类型，默认值为'2'，可选值为：
'1'：市价单
'2'：限价单

'3'：市价止损
'4'：限价止损
'5'：行权
'6'：弃权
'7'：询价
'8'：应价
'9'：冰山单
'A'：影子单
'B'：互换
'C'：套利申请
'D'：套保申请
'F'：行权前期权自对冲申请
'G'：履约期货自对冲申请
'H'：做市商留仓
可使用如 Enum_Order_Market、Enum_Order_Limit 等订单类型枚举函数获取相应的类型，
validType(char) 订单有效类型，字符类型，默认值为'0'， 可选值为：
'0'：当日有效
'1'：长期有效
'2'：限期有效
'3'：即时部分
'4'：即时全部
可使用如 Enum_GFD()、Enum_GTC()等订单有效类型枚举函数获取相应的类型，
Hedge(char) 投保标记，字符类型，默认值为'T'，可选值为：
'T'：投机
'B'：套保
'S'：套利
'M'：做市
可使用如 Enum_Speculate、Enum_Hedge 等订单投保标记枚举函数获取相应的类型，
triggerType(char) 触发委托类型，默认值为'N'，可用的值为：
'N'：普通单
'P'：预备单(埋单)
'A'：自动单
'C'：条件单
triggerMode(char) 触发模式，默认值为'N'，可用的值为：
'N'：普通单
'L'：最新价
'B'：买价
'A'：卖价
triggerCondition(char) 触发条件，默认值为'N'，可用的值为：
'N'：无
'g'：大于
'G'：大于等于
'l'：小于
'L'：小于等于

triggerPrice(float) 触发价格，默认价格为 0。
【备注】
针对当前公式指定的帐户、商品发送委托单，发送成功返回如"1-2"的下单编号，发送失败返回空字符串""。
返回结果形式为：retCode, retMsg，retCode 的数据类型为可以为负的整数, retMsg 的数据类型为字符串。
其中发送成功时 retCode 为 0，retMsg 为返回的下单编号 localOrderId，其组成规则为：策略 id-该策略中发送委托单的次数，所以下单编号"1-2"表示在策略 id 为 1 的策略中的第 2 次发送委托单返回的下单编号。
当发送失败时 retCode 为负数，retMsg 为返回的发送失败的原因，retCode 可能返回的值及含义如下：
-1：未选择实盘运行，请在设置界面勾选"实盘运行"，或者在策略代码中调用 SetActual()方法选择实盘运行；
-2：策略当前状态不是实盘运行状态，请勿在历史回测阶段调用该函数；
-3：未指定下单账户信息；
-4：输入的账户没有在极星客户端登录；
-5：请调用 StartTrade 方法开启实盘下单功能。
该函数直接发单，不经过任何确认，并会在每次公式计算时发送，一般需要配合着仓位头寸进行条件处理，在不清楚运行机制的情况下，请慎用。
注：该函数历史测试阶段下单调用的是 Buy、Sell 函数下单。
【示例】
```
# 当 retCode 为 0 时表明发送订单信息成功，retMsg 为返回的下单编号 localOrderId。
retCode, retMsg = A_SendOrder(Enum_Buy(), Enum_Exit(), 1, Q_AskPrice())
```

A_ModifyOrder
【说明】
发送改单指令。
【语法】
bool A_ModifyOrder(string localOrderId, char orderDirct, char entryOrExit, int orderQty, float orderPrice, string contractNo='', string userNo='', char orderType='2', char validType='0', char hedge='T', char triggerType='N', char triggerMode='N', char triggerCondition='N', float triggerPrice=0)
【参数】
localOrderId(int|string) 本地定单号，或者使用 A_SendOrder 返回的下单编号
orderDirct(char) 发送委托单的买卖类型，取值为 Enum_Buy 或 Enum_Sell 之一
entryOrExit(char) 发送委托单的开平仓类型，取值为 Enum_Entry,Enum_Exit,Enum_ExitToday 之一
orderQty(int) 委托单的交易数量
orderPrice(float) 委托单的交易价格
contractNo(string) 商品合约编号，默认值为基准合约
userNo(string) 指定的账户名称，默认为界面选定的账户名称
orderType(char) 订单类型，字符类型，默认值为'2'，可选值为：
'1'：市价单
'2'：限价单

'3'：市价止损
'4'：限价止损
'5'：行权
'6'：弃权
'7'：询价
'8'：应价
'9'：冰山单
'A'：影子单
'B'：互换
'C'：套利申请
'D'：套保申请
'F'：行权前期权自对冲申请
'G'：履约期货自对冲申请
'H'：做市商留仓
可使用如 Enum_Order_Market、Enum_Order_Limit 等订单类型枚举函数获取相应的类型，
validType(char) 订单有效类型，字符类型，默认值为'0'， 可选值为：
'0'：当日有效
'1'：长期有效
'2'：限期有效
'3'：即时部分
'4'：即时全部
可使用如 Enum_GFD、Enum_GTC 等订单有效类型枚举函数获取相应的类型，
hedge(char) 投保标记，字符类型，默认值为'T'，可选值为：
'T'：投机
'B'：套保
'S'：套利
'M'：做市
可使用如 Enum_Speculate、Enum_Hedge 等订单投保标记枚举函数获取相应的类型，
triggerType(char) 触发委托类型，默认值为'N'，可用的值为：
'N'：普通单
'P'：预备单(埋单)
'A'：自动单
'C'：条件单
triggerMode(char) 触发模式，默认值为'N'，可用的值为：
'N'：普通单
'L'：最新价
'B'：买价
'A'：卖价
triggerCondition(char) 触发条件，默认值为'N'，可用的值为：
'N'：无
'g'：大于
'G'：大于等于
'l'：小于
'L'：小于等于

triggerPrice(float) 触发价格，默认价格为 0。
【备注】
针对指定帐户、订单发送改单指令，发送成功返回 True，发送失败返回 False。
该函数直接发单，不经过任何确认，并会在每次公式计算时发送，一般需要配合着仓位头寸进行条件处理，在不清楚运行机制的情况下，请慎用。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_DeleteOrder
【说明】
针对指定帐户、商品发送撤单指令。
【语法】
bool A_DeleteOrder(int|string localOrderId)
【参数】
localOrderId(int|string) 定单号，或者使用 A_SendOrder 返回的下单编号
【备注】
针对指定帐户、商品发送撤单指令，发送成功返回 True，发送失败返回 False。
该函数直接发单，不经过任何确认，并会在每次公式计算时发送，一般需要配合着仓位头寸进行条件处理，在不清楚运行机制的情况下，请慎用。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
无

A_GetOrderNo
【说明】
获取下单编号对应的定单号和委托号。
【语法】
string, string A_GetOrderNo(string localOrderId)
【参数】
localOrderId(int|string) 使用 A_SendOrder 返回的下单编号
【备注】
针对当前策略使用 A_SendOrder 返回的下单编号，可以使用 A_GetOrderNo 获取下单编号对应的定单号和委托号。
由于使用 A_SendOrder 返回的下单编号 localOrderId 与策略相关，所以在策略重启后 localOrderId 会发生变化。
由于委托单对应的定单号与客户端有关，所以在客户端重启后，委托单对应的定单号可能会发生变化。
由于委托号是服务器生成的，所以在使用 A_SendOrder 得到下单编号后，如果服务器还没有返回相应的委托单信息，可能获取不到相应的定单号和委托号。
当 localOrderId 对应的定单号和委托号还没有从服务器返回，则对应的值为空字符串。
注：不能使用于历史测试，仅适用于实时行情交易。
【示例】
retCode, retMsg = A_SendOrder(.....)

```
time.sleep(5)
if retCode == 0:
    sessionId, orderNo = A_GetOrderNo(retMsg)
```

DeleteAllOrders

【说明】

批量撤单函数。

【语法】

bool DeleteAllOrders(string contractNo='', string userNo='')

【参数】

contractNo(string) 合约代码，默认为所有合约，指定后只撤指定合约

userNo(string) 指定的交易账户，默认当前账户

【备注】

本函数将检查指定账户下所有处于排队状态的订单，并依次发送撤单指令

【示例】

无

枚举函数

Enum_Buy

【说明】

返回买卖状态的买入枚举值

【语法】

char Enum_Buy()

【参数】

无

【备注】

返回字符

【示例】

无

Enum_Sell

【说明】

返回买卖状态的卖出枚举值

【语法】

char Enum_Sell()

【参数】

无

【备注】

返回字符

【示例】

无

Enum_Entry
【说明】
返回开平状态的开仓枚举值
【语法】
char Enum_Entry()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Exit
【说明】
返回开平状态的平仓枚举值
【语法】
char Enum_Exit()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_ExitToday
【说明】
返回开平状态的平今枚举值

【语法】
char Enum_ExitToday()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_EntryExitIgnore
【说明】
返回开平状态不区分开平的枚举值
【语法】
char Enum_EntryExitIgnore()
【参数】
无

【备注】
返回字符
【示例】
无

Enum_Sended
【说明】
返回委托状态为已发送的枚举值
【语法】
char Enum_Sended()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Accept
【说明】
返回委托状态为已受理的枚举值
【语法】
char Enum_Accept()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Triggering
【说明】
返回委托状态为待触发的枚举值
【语法】
char Enum_Triggering()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Active
【说明】
返回委托状态为已生效的枚举值

【语法】
char Enum_Active()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Queued
【说明】
返回委托状态为已排队的枚举值
【语法】
char Enum_Queued()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_FillPart
【说明】
返回委托状态为部分成交的枚举值
【语法】
char Enum_FillPart()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Filled
【说明】
返回委托状态为全部成交的枚举值
【语法】
char Enum_Filled()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Canceling
【说明】
返回委托状态为待撤的枚举值
【语法】
char Enum_Canceling()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Modifying
【说明】
返回委托状态为待改的枚举值
【语法】
char Enum_Modifying()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Canceled
【说明】
返回委托状态为已撤单的枚举值
【语法】
char Enum_Canceled()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_PartCanceled
【说明】
返回委托状态为已撤余单的枚举值
【语法】
char Enum_PartCanceled()
【参数】
无

【备注】
返回字符
【示例】
无

Enum_Fail
【说明】
返回委托状态为指令失败的枚举值
【语法】
char Enum_Fail()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Suspended
【说明】
返回委托状态为已挂起的枚举值
【语法】
char Enum_Suspended()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Apply
【说明】
返回委托状态为已申请的枚举值
【语法】
char Enum_Apply()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Period_Tick
【说明】
返回周期类型成交明细的枚举值

【语法】
char Enum_Period_Tick()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Period_Min
【说明】
返回周期类型分钟线的枚举值
【语法】
char Enum_Period_Min()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Period_Day
【说明】
返回周期类型日线的枚举值
【语法】
char Enum_Period_Day()
【参数】
无
【备注】
返回字符
【示例】
无

RGB_Red
【说明】
返回颜色类型红色的枚举值
【语法】
int RGB_Red()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Green
【说明】
返回颜色类型绿色的枚举值
【语法】
int RGB_Green()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Blue
【说明】
返回颜色类型蓝色的枚举值
【语法】
int RGB_Blue()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Yellow
【说明】
返回颜色类型黄色的枚举值
【语法】
int RGB_Yellow()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Purple
【说明】
返回颜色类型紫色的枚举值
【语法】
int RGB_Purple()
【参数】
无

【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Gray
【说明】
返回颜色类型灰色的枚举值
【语法】
int RGB_Gray()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

RGB_Brown
【说明】
返回颜色类型褐色的枚举值
【语法】
int RGB_Brown()
【参数】
无
【备注】
返回 16 进制颜色代码
【示例】
无

Enum_Order_Market
【说明】
返回订单类型市价单的枚举值
【语法】
char Enum_Order_Market()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Limit
【说明】
返回订单类型限价单的枚举值

【语法】
char Enum_Order_Limit()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_MarketStop
【说明】
返回订单类型市价止损单的枚举值
【语法】
char Enum_Order_MarketStop()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_LimitStop
【说明】
返回订单类型限价止损单的枚举值
【语法】
char Enum_Order_LimitStop()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Execute
【说明】
返回订单类型行权单的枚举值
【语法】
char Enum_Order_Execute()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Abandon
【说明】
返回订单类型弃权单的枚举值
【语法】
char Enum_Order_Abandon()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Enquiry
【说明】
返回订单类型询价单的枚举值
【语法】
char Enum_Order_Enquiry()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Offer
【说明】
返回订单类型应价单的枚举值
【语法】
char Enum_Order_Offer()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Iceberg
【说明】
返回订单类型冰山单的枚举值
【语法】
char Enum_Order_Iceberg()
【参数】
无

【备注】
返回字符
【示例】
无

Enum_Order_Ghost
【说明】
返回订单类型影子单的枚举值
【语法】
char Enum_Order_Ghost()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_Swap
【说明】
返回订单类型互换单的枚举值
【语法】
char Enum_Order_Swap()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_SpreadApply
【说明】
返回订单类型套利申请的枚举值
【语法】
char Enum_Order_SpreadApply()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_HedgApply
【说明】
返回订单类型套保申请的枚举值

【语法】
char Enum_Order_HedgApply()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_OptionAutoClose
【说明】
返回订单类型行权前期权自对冲申请的枚举值
【语法】
char Enum_Order_OptionAutoClose()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_FutureAutoClose
【说明】
返回订单类型履约期货自对冲申请的枚举值
【语法】
char Enum_Order_FutureAutoClose()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Order_MarketOptionKeep
【说明】
返回订单类型做市商留仓的枚举值
【语法】
char Enum_Order_MarketOptionKeep()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_GFD
【说明】
返回订单有效类型当日有效的枚举值
【语法】
char Enum_GFD()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_GTC
【说明】
返回订单有效类型当日有效的枚举值
【语法】
char Enum_GTC()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_GTD
【说明】
返回订单有效类型限期有效的枚举值
【语法】
char Enum_GTD()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_IOC
【说明】
返回订单有效类型即时部分有效的枚举值
【语法】
char Enum_IOC()
【参数】
无

【备注】
返回字符
【示例】
无

Enum_FOK
【说明】
返回订单有效类型即时全部有效的枚举值
【语法】
char Enum_FOK()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Speculate
【说明】
返回订单投保标记投机的枚举值
【语法】
char Enum_Speculate()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Hedge
【说明】
返回订单投保标记套保的枚举值
【语法】
char Enum_Hedge()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Spread
【说明】
返回订单投保标记套利的枚举值

【语法】
char Enum_Spread()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Market
【说明】
返回订单投保标记做市的枚举值
【语法】
char Enum_Market()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Close
【说明】
返回收盘价的枚举值
【语法】
char Enum_Data_Close()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Open
【说明】
返回开盘价的枚举值
【语法】
char Enum_Data_Open()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_High
【说明】
返回最高价的枚举值
【语法】
char Enum_Data_High()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Low
【说明】
返回最低价的枚举值
【语法】
char Enum_Data_Low()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Median
【说明】
返回中间价的枚举值，中间价=（最高价+最低价）/ 2
【语法】
char Enum_Data_Median()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Typical
【说明】
返回标准价的枚举值，标准价=（最高价+最低价+收盘价）/ 3
【语法】
char Enum_Data_Typical()
【参数】
无

【备注】
返回字符
【示例】
无

Enum_Data_Weighted
【说明】
返回加权收盘价的枚举值，加权收盘价=（最高价+最低价+开盘价+收盘价）/ 4
【语法】
char Enum_Data_Weighted()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Vol
【说明】
返回成交量的枚举值
【语法】
char Enum_Data_Vol()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Opi
【说明】
返回持仓量的枚举值
【语法】
char Enum_Data_Opi()
【参数】
无
【备注】
返回字符
【示例】
无

Enum_Data_Time
【说明】
返回 K 线时间的枚举值

【语法】
char Enum_Data_Time()
【参数】
无
【备注】
返回字符
【示例】
无

设置函数

SetUserNo
【说明】
设置实盘交易账户
【语法】
int SetUserNo(string userNo)
【参数】
userNo(string) 实盘交易账户，不能为空字符串
【备注】
返回整型, 0 成功，-1 失败
若需要添加多个不同的交易账号，则可多次调用该账户
【示例】
SetUserNo('ET001')

SetBarInterval
【说明】
设置指定合约的 K 线类型和 K 线周期，以及策略历史回测的起始点信息
【语法】
int SetBarInterval(string contractNo, char barType, int barInterval, int|string|char sampleConfig=2000)
【参数】
contractNo(string) 合约编号
barType(char) K 线类型 T 分笔，M 分钟，D 日线
barInterval(int) K 线周期
sampleConfig(int|string|char) 策略历史回测的起始点信息，可选的值为：
字符 A：使用所有 K 线
字符 N：不执行历史 K 线
整数 ：历史回测使用的 K 线根数
字符串 ：用于历史回测样本的起始日期，格式为 YYYYMMDD，精确到日，例如 2019-04-30 的日期格式为'20190430'
默认为使用 2000 根 K 线进行回测
【备注】
返回整型, 0 成功，-1 失败

通过该方法系统会订阅指定合约的 K 线数据，
对于相同的合约，如果使用该函数设置不同的 K 线类型(barType)和周期(barInterval)，则系统会同时订阅指定的 K 线类型和周期的行情数据
如果使用该方法订阅了多个合约，则第一条合约为基准合约
如果在策略中使用 SetBarInterval 方法订阅了合约，则在设置界面选中的基准合约便不再订阅
若要订阅秒线数据，可以设置 K 线类型为'T'，K 线类型为 n(n>0)，则表示订阅 n 秒的 K 线。
【示例】
# 订阅合约 ZCE|Z|SR|MAIN 的 3 分钟 K 线数据，并使用所有 K 线样本进行历史回测
SetBarInterval('ZCE|Z|SR|MAIN', 'M', 3, 'A')
# 订阅合约 ZCE|Z|SR|MAIN 的 3 分钟 K 线数据，并不使用 K 线样本进行历史回测
SetBarInterval('ZCE|Z|SR|MAIN', 'M', 3, 'N')
# 订阅合约 ZCE|Z|SR|MAIN 的 3 分钟 K 线数据，并使用 2000 根 K 线样本进行历史回测
SetBarInterval('ZCE|Z|SR|MAIN', 'M', 3, 2000)
# 订阅合约 ZCE|Z|SR|MAIN 的 3 分钟 K 线数据，由于 sampleConfig 的默认值为 2000，所以使用 2000 根 K 线样本进行历史回测
SetBarInterval('ZCE|Z|SR|MAIN', 'M', 3)
# 订阅合约 ZCE|Z|SR|MAIN 的 3 分钟 K 线数据，并使用 2019-04-30 起的 K 线进行历史回测
SetBarInterval('ZCE|Z|SR|MAIN', 'M', 3, '20190430')
# 订阅合约 ZCE|Z|SR|MAIN 的 1 秒钟 K 线数据，并使用 2000 根 K 线样本进行历史回测
SetBarInterval(' ZCE|Z|SR|MAIN', 'T', 1, 2000)

SetInitCapital
【说明】
设置初始资金，不设置默认 100 万
【语法】
int SetInitCapital(float capital=10000000)
【参数】
capital(float) 初始资金，默认为 10000000
【备注】
返回整型，0 成功，-1 失败
【示例】
SetInitCapital(200*10000), 设置初始资金为 200 万

SetMargin
【说明】
设置保证金参数，不设置或设置失败取界面设置的保证金比例

【语法】
int SetMargin(float type, float value=0, string contractNo='')

【参数】
Type(float) 0：按比例收取保证金， 1：按定额收取保证金
Value(float) 按比例收取保证金时的比例， 或者按定额收取保证金时的额度

contractNo(string) 合约编号，默认为基础合约
【备注】
返回整型，0 成功，-1 失败
【示例】
# 设置基础合约的保证金按比例收取 8%
SetMargin(0, 0.08)
# 设置合约 ZCE| Z |SR| MAIN 的保证金按额度收取 80000
SetMargin(1, 80000, 'ZCE|Z|SR|MAIN')

SetTradeFee
【说明】
设置手续费收取方式
【语法】
int SetTradeFee(string type, int feeType, float feeValue, string contractNo='')
【参数】
Type(string) 手续费类型，'A'-全部，'O'-开仓，'C'-平仓，'T'-平今
feeType(int) 手续费收取方式，1-按比例收取，2-按定额收取
feeValue(float) 按比例收取手续费时，feeValue 为收取比例；按定额收取手续费时，feeValue 为收取额度
contractNo(string) 合约编号，默认为基础合约
【备注】
返回整型，0 成功，-1 失败
【示例】
# 设置基础合约的开仓手续费为 5 元/手
SetTradeFee('O', 2, 5)
# 设置基础合约的开仓手续费为每笔 2%
SetTradeFee('O', 1, 0.02)
# 设置合约 ZCE|Z|SR|MAIN 的平今手续费为 5 元/手
SetTradeFee('T', 2, 5, "ZCE|Z|SR|MAIN")

SetActual
【说明】
设置策略在实盘上运行
【语法】
int SetActual()
【参数】
无
【备注】
返回整型，0 成功，-1 失败
【示例】
无

SetOrderWay
【说明】

设置发单方式
【语法】
int SetOrderWay(int type)
【参数】
type(int) 在实盘上的发单方式，1 表示实时发单，2 表示 K 线完成后发单
【备注】
返回整型，0 成功，-1 失败
【示例】
# 在实盘上使用实时数据运行策略，实时发单
SetOrderWay(1)
# 在实盘上使用实时数据运行策略，在 K 线稳定后发单
SetOrderWay(2)

SetTradeDirection
【说明】
设置交易方向
【语法】
int SetTradeDirection(int tradeDirection)
【参数】
tradeDirection(int) 设置交易方向
0：双向交易
1：仅多头
2：仅空头
【备注】
返回整型，0 成功，-1 失败
【示例】
SetTradeDirection(0) # 双向交易

SetMinTradeQuantity
【说明】
设置最小下单量，单位为手，默认值为 1 手。
【语法】
int SetMinTradeQuantity(int tradeQty=1)
【参数】
tradeQty(int) 最小下单量，默认为 1，不超过 1000
【备注】
返回整型，0 成功，-1 失败
【示例】
无

SetHedge
【说明】
设置投保标志
【语法】

int SetHedge(char hedge)
【参数】
Hedge(char) 投保标志
'T ': 投机
'B ': 套保
'S' : 套利
'M' : 做市
【备注】
返回整型，0 成功，-1 失败
【示例】
SetHedge('T') # 设置基础合约的投保标志为投机

SetSlippage
【说明】
设置滑点损耗
【语法】
int SetSlippage(float slippage)
【参数】
slippage(float) 滑点损耗
【备注】
返回整型，0 成功，-1 失败
【示例】
无

SetTriggerType
【说明】
设置触发方式
【语法】
int SetTriggerType(int type, int|list value=None)
【参数】
type(int) 触发方式，可使用的值为：
1 : 即时行情触发
2 : 交易数据触发
3 : 每隔固定时间触发
4 : 指定时刻触发
5 : K 线触发
value(int) 当触发方式是为每隔固定时间触发(type=3)时，value 为触发间隔，单位为毫秒，必须为 100 的整数倍
当触发方式为指定时刻触发(type=4)时，value 为触发时刻列表，时间的格式为'20190511121314'
当 type 为其他值时，该值无效，可以不填
【备注】
返回整型，0 成功，-1 失败
【示例】

SetTriggerType(3, 1000) # 每隔 1000 毫秒触发一次
SetTriggerType(4, ['084000', '084030', '084100']) # 指定时刻触发

SetWinPoint
【说明】
设置止盈点
【语法】
void SetWinPoint(int winPoint, int nPriceType = 0, int nAddTick = 0, string contractNo = "")
【参数】
winPoint(int) 赢利点数值，若当前价格相对于最近一次开仓价格的盈利点数达到或超过该值，就进行止盈
nPriceType(int) 平仓下单价格类型 0:最新价 1：对盘价 2：挂单价 3：市价 4：停板价，默认值为 0；
nAddTick(int) 超价点数 仅当 nPrice 为 0，1，2 时有效，默认为 0；
contractNo(string) 合约代码，默认为基准合约。
【备注】
止损止盈只对用 Buy、Sell 函数下单的方式有效，A_SendOrder 函数因为在历史阶段下单调用的是 Buy、Sell 函数，因此止损止盈对 A_SendOrder 函数在历史阶段下单也生效
【示例】
# 当价格相对于最近一次开仓价格超过 10 个点，进行止盈平仓。如郑棉合约多头：开仓价格为 15000，当前价格大于或等于 5*10=50 时，即达到 15050，则进行平仓。
SetWinPoint(10)

SetStopPoint
【说明】
设置止损点
【语法】
void SetStopPoint (int stopPoint, int nPriceType = 0, int nAddTick = 0, string contractNo = "")
【参数】
stopPoint(int) 止损点数，若当前价格相对于最近一次开仓价格亏损点数达到或跌破该值，就进行止损
nPriceType(int) 平仓下单价格类型， 0：最新价 1：对盘价 2：挂单价 3：市价 4：停板价，默认值为 0
nAddTick(int) 超价点数， 仅当 nPrice 为 0，1，2 时有效，默认为 0
contractNo(string) 合约代码，默认为基准合约
【备注】
止损止盈只对用 Buy、Sell 函数下单的方式有效，A_SendOrder 函数因为在历史阶段下单调用的是 Buy、Sell 函数，因此止损止盈对 A_SendOrder 函数在历史阶段下单也生效
【示例】
# 当价格跌破 10 个点，进行止损平仓。 如：如郑棉合约多头：开仓价格为 15000，当前价格小于或等于 5*10=50 时，即达到 14950，则进行平仓。
SetStopPoint(10)

SetFloatStopPoint

【说明】
设置浮动止损点
【语法】
int SetFloatStopPoint(int startPoint, int stopPoint, int nPriceType = 0, int nAddTick = 0, string contractNo = "")
【参数】
startPoint(int) 启动点数，当前价格相对于最后一次开仓价格盈利点数超过该值后启动浮动止损监控
stopPoint(int) 止损点数，若当前价格相对于最近一次开仓价格亏损点数达到或跌破该值，就进行止损
nPriceType(int) 平仓下单价格类型， 0：最新价 1：对盘价 2：挂单价 3：市价 4：停板价，默认为 0
nAddTick(int) 超价点数， 仅当 nPrice 为 0，1，2 时有效，默认为 0
contractNo(string) 合约代码，默认为基准合约
【备注】
止损止盈只对用 Buy、Sell 函数下单的方式有效，A_SendOrder 函数因为在历史阶段下单调用的是 Buy、Sell 函数，因此止损止盈对 A_SendOrder 函数在历史阶段下单也生效
【示例】
SetFloatStopPoint(20,10)
举例：郑棉合约，多头方向。开仓价格为 15000，当前价格突破 15100 后开启浮动止损，若此，止损点会随着价格上升而不断上升。假如价格上涨到 15300，则此时的止损价格为(15300-50),即 15250，若价格从 15300 回落到 15250，则进行自动平仓。

SetStopWinKtBlack
【说明】
设置不触发止损止盈和浮动止损的 K 线类型
【语法】
int SetStopWinKtBlack(int op, char kt)
【参数】
Op(int) 操作类型必须为 0: 取消设置，1: 增加设置，中的一个
kt(char) K 线类型必须为 'D'，'M'，'T'中的一个
【备注】
返回整型，0 成功，-1 失败
【示例】
返回整型，0 成功，-1 失败
止损止盈只对用 Buy、Sell 函数下单的方式有效，A_SendOrder 函数因为在历史阶段下单调用的是 Buy、Sell 函数，因此止损止盈对 A_SendOrder 函数在历史阶段下单也生效

SubQuote
【说明】
订阅指定合约的即时行情
【语法】
bool SubQuote(string contractNo1, string contractNo2, string contractNo3, ...)
【参数】

contractNo(string) 合约编号，为空不做任何操作
【备注】
该方法可用在策略的 initialize(context)方法中订阅指定合约的即时行情，也可在handle_data(context)方法中动态的订阅指定合约的即时行情。
【示例】
```
# 订阅合约 TA909 的即时行情
SubQuote("ZCE|F|TA|909")
# 订阅合约 TA909 和 TA910 的即时行情
SubQuote("ZCE|F|TA|909", "ZCE|F|TA|910")
#订阅 TA 品种下所有合约的即时行情
SubQuote("ZCE|F|TA")
```

UnsubQuote
【说明】
退订指定合约的即时行情。
【语法】
bool UnsubQuote(string contractNo1, string contractNo2, string contractNo3, ...)
【参数】
contractNo(string) 合约编号
【备注】
该方法可用在策略的 initialize(context)方法中退订指定合约的即时行情，也可在handle_data(context)方法中动态的退订指定合约的即时行情。
【示例】
```
# 退订合约'ZCE|F|SR|909'的即时行情
UnsubQuote('ZCE|F|SR|909')
# 退订合约'ZCE|F|SR|909'和'ZCE|F|SR|910'的即时行情
UnsubQuote('ZCE|F|SR|909', 'ZCE|F|SR|910')
# 退订合约商品'ZCE|F|SR'对应的所有合约的即时行情
UnsubQuote('ZCE|F|SR')
```

绘图函数

PlotNumeric
【说明】
在当前 Bar 输出一个数值
【语法】
void PlotNumeric(string name, float value, int color= 0xdd0000
, bool main=True, char axis=False, int barsback=0)
【参数】
name(string) 输出值的名称，不区分大小写
value(float) 输出的数值
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图

axis(bool) 指标是否使用独立坐标，True-独立坐标，False-非独立坐标，默认非独立坐标
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
在当前 Bar 输出一个数值，输出的值用于在上层调用模块显示。
【示例】
输出 MA1 的值
例 1：PlotNumeric ("MA1",Ma1Value)

PlotIcon
【说明】
在当前 Bar 输出一个图标
【语法】
void PlotIcon(float value,int icon, bool main=True, int barsback=0)
【参数】
Value(float) 输出的值
icon(int) 图标类型，0-默认图标，1-笑脸，2-哭脸，3-上箭头，4-下箭头，5-上箭头 2， 6-下箭头 2，7-喇叭，8-加锁，9-解锁，10-货币+，11-货币-，12-加号，13-减号，14-叹号，15-叉号
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
在当前 Bar 输出一个图标，输出的图标用于在上层调用模块显示
【示例】
# 输出一个叹号
PlotIcon(10,14)

PlotDot
【说明】
在当前 Bar 输出一个点
【语法】
void PlotDot(string name, float value, int icon=0, int color=0xdd0000, bool main=True, int barsback=0)
【参数】
value(float) 输出的值
icon(int) 图标类型 0-14，共 15 种样式，包括箭头，圆点，三角等，默认值为 0
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
在当前 Bar 输出一个点，输出的值用于在上层调用模块显示。
【示例】
PlotDot(name="Dot", value=Close()[-1], main=True)

PlotBar

【说明】
绘制一根 Bar
【语法】
void PlotBar(string name, float vol1, float vol2, int color=0xdd0000, bool main=True, bool filled=True, int barsback=0)
【参数】
name bar 名称
vol1(float) 柱子起始点
vol2(float) 柱子结束点
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
filled(bool) 是否填充，默认填充
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
在当前 Bar 输出一个 Bar，输出的值用于在上层调用模块显示。
【示例】
PlotBar("BarExample1", Vol()[-1], 0, RGB_Red())

PlotText
【说明】
在当前 Bar 输出字符串
【语法】
void PlotText(float value, string text, int color=0xdd0000, bool main=True, int barsback=0)
【参数】
value(float) 输出的价格
text(string) 输出的字符串，最多支持 19 个英文字符
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(True) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(False) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar。
【备注】
在当前 Bar 输出字符串，输出的值用于在上层调用模块显示
【示例】
PlotText(100, "ORDER")

PlotVertLine
【说明】
在当前 Bar 输出一个竖线
【语法】
void PlotVertLine(color=0xdd0000, bool main=True, bool axis=False, int barsback=0)
【参数】
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
axis(bool) 指标是否使用独立坐标，True-独立坐标，False-非独立坐标，默认非独立坐标
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar

【备注】
在当前 Bar 输出一条竖线，输出的值用于在上层调用模块显示。
【示例】
PlotVertLine(main=True, axis = True)

PlotPartLine
【说明】
绘制斜线段
【语法】
void PlotPartLine(string name, int index1, float price1, int count, float price2, int color=0xdd0000, bool main=True, bool axis=False, int width=1)
【参数】
name(string) 名称
index1(int) 起始 bar 索引
price1(float) 起始价格
count(int) 从起始 bar 回溯到结束 bar 的根数
price2(float) 结束价格
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
axis(bool) 指标是否使用独立坐标，True-独立坐标，False-非独立坐标，默认非独立坐标
width(int) 线段宽度，默认 1
【备注】
在指定区间输出一个斜线，输出的值用于在上层调用模块显示。
【示例】
idx1 = CurrentBar()
p1 = Close()[-1]
if idx1 >= 100:
count = 1
p2 = Close()[-2]
PlotPartLine("PartLine", idx1, p1, count, p2, RGB_Red(), True, True, 1)

PlotStickLine
【说明】
绘制竖线段
【语法】
void PlotStickLine(string name, float price1, float price2, int color=0xdd0000, bool main=True, bool axis=False, int barsback=0)
【参数】
name(string) 名称
price1(float) 起始价格
price2(flaot) 结束价格
color(int) 输出值的显示颜色，默认表示使用属性设置框中的颜色
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
axis(bool) 指标是否使用独立坐标，True-独立坐标，False-非独立坐标，默认非独立坐标

barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
在当前 Bar 输出一个竖线段，输出的值用于在上层调用模块显示。
【示例】
PlotStickLine("StickLine", Close()[-1], Open()[-1], RGB_Blue(), True, True, 0)

UnPlotText
【说明】
在当前 Bar 取消输出的字符串
【语法】
void UnPlotText(bool main=True, int barsback=0)
【参数】
Main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
Barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
无
【示例】
UnPlotText()

UnPlotIcon
【说明】
在当前 Bar 取消输出的 Icon
【语法】
void UnPlotIcon(bool main=True, int barsback=0)
【参数】
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
【示例】
UnPlotIcon()

UnPlotDot
【说明】
在当前 Bar 取消输出的 Dot
【语法】
void UnPlotDot(bool main=True, int barsback=0)
【参数】
name 名称
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
【示例】
UnPlotDot()

UnPlotBar
【说明】
在当前 Bar 取消输出的 Bar
【语法】
void UnPlotBar(string name, bool main=True, int barsback=0)
【参数】
name(string) 名称
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
无
【示例】
UnPlotBar("Bar")

UnPlotNumeric
【说明】
在当前 Bar 取消输出的 Numeric
【语法】
void UnPlotNumeric(string name, bool main=True, int barsback=0)
【参数】
name(string) 名称
main 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar。
【备注】
无
【示例】
UnPlotNumeric("numeric")

UnPlotVertLine
【说明】
在当前 Bar 取消输出的竖线
【语法】
void UnPlotVertLine(bool main=True, int barsback=0)
【参数】
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
无
【示例】
UnPlotVertLine()

UnPlotPartLine
【说明】
在当前 Bar 取消输出的斜线段

【语法】
void UnPlotPartLine(string name, int index1, int count, bool main=True)
【参数】
name(string) 名称
index1(int) 起始 bar 索引
count(int) 从起始 bar 回溯到结束 bar 的根数
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
【备注】
无
【示例】
UnPlotPartLine("PartLine", idx1, count, True)

UnPlotStickLine
【说明】
在当前 Bar 取消输出的竖线段
【语法】
void UnPlotStickLine(string name, bool main, int barsback=0)
【参数】
name(string) 名称
main(bool) 指标是否加载到主图，True-主图，False-幅图，默认主图
barsback(int) 从当前 Bar 向前回溯的 Bar 数，默认值为当前 Bar
【备注】
无
【示例】
UnPlotStickLine("StickLine")

统计函数

SMA
【说明】
获取加权移动平均值
【语法】
SMA(self, numpy.array price, int period, int weight)
【参数】
price(numpy.array) 序列值，numpy 数组
period(int) 周期
weight(int) 权重
【备注】
返回值为浮点型 numpy.array；
如果计算成功，此时返回值是计算出的 sma 值序列；
如果计算失败，此时返回值 numpy.array 为空
【示例】
SMA(Close(), 12, 2)

ParabolicSAR
【说明】
计算抛物线转向
【语法】
ParabolicSAR(self, numpy.array high, numpy.array low, float afstep, float aflimit)
【参数】
high(numpy.array) 最高价序列值，numpy 数组
low(numpy.array) 最低价序列值，numpy 数组
afstep(float) 加速因子
aflimit(float) 加速因子的限量
【备注】
返回值为四个值，均为数值型 numpy.array
第一个值序列为 oParClose,当前 bar 的停损值；
第二个值序列为 oParOpen, 下一 Bar 的停损值；
第三个值序列为 oPosition，输出建议的持仓状态，1 - 买仓，-1 - 卖仓；
第四个值序列为 oTransition, 输出当前 Bar 的状态是否发生反转，1 或 -1 为反转，0 为保持不变。
当输入 high,low 的 numpy 数组为空时，计算失败，返回的四个值均为空的 numpy.array
【示例】
ParabolicSAR(High(), Low(), 0.02, 0.2)

REF
【说明】
求 N 周期前数据的值
【语法】
float REF(float price, int length)
【参数】
price(float) 价格
length(int) 需要计算的周期数
【备注】
length 不能小于 0
【示例】
# 获得上一周期的收盘价，等价于 Close[-2]
REF(Close(), 1)
# 返回 10 周期前的高低收价格的平均值
REF((Close() + High() + Low())/ 3, 10)

Highest
【说明】
求最高
【语法】
numpy.array Highest(list|numpy.array price, int length)
【参数】

price(list|numpy.array) 用于求最高值的值，必须是数值型列表
length(int) 需要计算的周期数，为整型
【备注】
该函数计算指定周期内的数值型序列值的最高值，返回值为浮点数数字列表;
当 price 的类型不是 list 或者 price 的长度为 0 时，则返回为空的 numpy.array()
【示例】
计算 12 周期以来的收盘价的最高值
Highest (Close(), 12);
计算 10 周期以来高低收价格的平均值的最高值。
Highest (HisData(Enum_Data_Typical()), 10)

Lowest
【说明】
求最低
【语法】
numpy.array Lowest(list|numpy.array price, int length)
【参数】
price(list|numpy.array) 用于求最低值的值，必须是数值型列表
length(int) 需要计算的周期数，为整型
【备注】
该函数计算指定周期内的数值型序列值的最低值，返回值为浮点数数字列表;
当 price 的类型不是 list 或者 price 的长度为 0 时，则返回为空的 numpy.array()
【示例】
计算 12 周期以来的收盘价的最低值
Lowest (Close(), 12)
计算 10 周期以来高低收价格的平均值的最低值
Lowest (HisData(Enum_Data_Typical()), 10)

CountIf
【说明】
获取最近 N 周期条件满足的计数
【语法】
int CountIf(bool condition, int period)
【参数】
Condition(bool) 传入的条件表达式
Period(int) 计算条件的周期数
【备注】
获取最近 N 周期条件满足的计数
【示例】
# 最近 10 周期出现 Close>Open 的周期总数
CountIf(Close() > Open(), 10)

CrossOver
【说明】

求是否上穿
【语法】
bool CrossOver(np.array Price1, np.array Price2)
【参数】
Price1(np.array) 求相关系统的数据源 1，必须是 np 数组
Price2(np.array) 求相关系统的数据源 2，必须是 np 数组
【备注】
该函数返回 Price1 数值型序列值是否上穿 Price2 数值型序列值，返回值为布尔型。
【示例】
#判断上一个 Bar 的收盘价 Close 是否上穿 AvgPrice
CrossOver(Close(), AvgPrice);
注意：在使用判断穿越的函数时，要尽量避免使用例如 close 等不确定的元素，否则会导致信号消失，
一般情况下，Close 可以改用 High 和 Low 分别判断向上突破（函数 CrossOver）和向下突破（函数 CrossUnder）。

CrossUnder
【说明】
求是否下破
【语法】
Bool CrossUnder(np.array Price1, np.array Price2)
【参数】
Price1(np.array) 求相关系统的数据源 1，必须是 np 数组
Price2(np.array) 求相关系统的数据源 2，必须是 np 数组
【备注】
该函数返回 Price1 数值型序列值是否上穿 Price2 数值型序列值，返回值为布尔型。
【示例】
CrossUnder(Close(), AvgPrice); 判断上一个 Bar 的收盘价 Close 是否上穿 AvgPrice
注意：在使用判断穿越的函数时，要尽量避免使用例如 close 等不确定的元素，否则会导致信号消失，
一般情况下，Close 可以改用 High 和 Low 分别判断向上突破（函数 CrossOver）和向下突破（函数 CrossUnder）。

SwingHigh
【说明】
求波峰点
【语法】
float SwingHigh(np.array Price, int Length, int Instance, int Strength)
【参数】
Price(np.array) 用于求波峰点的值，必须是 np 数组或者序列变量
Length(int) 是需要计算的周期数，为整型
Instance(int) 设置返回哪一个波峰点，1 - 最近的波峰点，2 - 倒数第二个，以此类推
Strength(int) 设置转折点两边的需要的周期数，必须小于 Length
【备注】

该函数计算指定周期内的数值型序列值的波峰点，返回值为浮点数;
当序列值的 CurrentBar 小于 Length 时，该函数返回-1.0
【示例】
# 计算 Close 在最近 10 个周期的波峰点的值，最高点两侧每侧至少需要 2 个 Bar
SwingHigh(Close(), 10, 1, 2)

SwingLow
【说明】
求波谷点
【语法】
float SwingLow(np.array Price, int Length, int Instance, int Strength)
【参数】
Price(np.array) 用于求波峰点的值，必须是 np 数组或者序列变量
Length(int) 是需要计算的周期数，为整型
Instance(int) 设置返回哪一个波峰点，1 - 最近的波谷点，2 - 倒数第二个，以此类推
Strength(int) 设置转折点两边的需要的周期数，必须小于 Length
【备注】
该函数计算指定周期内的数值型序列值的波谷点，返回值为浮点数;
当序列值的 CurrentBar 小于 Length 时，该函数返回-1.0
【示例】
# 计算 Close 在最近 10 个周期的波谷点的值，最低点两侧需要至少 2 个 Bar
SwingLow(Close, 10, 1, 2)

日志函数

LogDebug
【说明】
在运行日志窗口中打印用户指定的调试信息。
【语法】
LogDebug(args)
【参数】
args 用户需要打印的内容，如需要在运行日志窗口中输出多个内容，内容之间用英文逗号分隔
【备注】
无
【示例】
accountId = A_AccountID()
LogDebug("当前使用的用户账户 ID 为 : ", accountId)
available = A_Available()
LogDebug("当前使用的用户账户 ID 为 : %s，可用资金为 : %10.2f" % (accountId, available))

LogInfo
【说明】

在运行日志窗口中打印用户指定的普通信息。
【语法】
LogInfo(args)
【参数】
args 用户需要打印的内容，如需要在运行日志窗口中输出多个内容，内容之间用英文逗号分隔
【备注】
无
【示例】
```
accountId = A_AccountID()
LogInfo("当前使用的用户账户 ID 为 : ", accountId)
available = A_Available()
LogInfo("当前使用的用户账户 ID 为 : %s， 可用资金为 : %10.2f" % (accountId, available))
```

LogWarn
【说明】
在运行日志窗口中打印用户指定的警告信息。
【语法】
LogWarn(args)
【参数】
args 用户需要打印的内容，如需要在运行日志窗口中输出多个内容，内容之间用英文逗号分隔
【备注】
无
【示例】
```
accountId = A_AccountID()
LogWarn("当前使用的用户账户 ID 为 : ", accountId)
available = A_Available()
LogWarn("当前使用的用户账户 ID 为 : %s， 可用资金为 : %10.2f" % (accountId, available))
```

LogError
【说明】
在运行日志窗口中打印用户指定的错误信息。
【语法】
LogError(args)
【参数】
args 用户需要打印的内容，如需要在运行日志窗口中输出多个内容，内容之间用英文逗号分隔
【备注】
无
【示例】
```
accountId = A_AccountID()
LogError("当前使用的用户账户 ID 为 : ", accountId)
available = A_Available()
```

LogError("当前使用的用户账户 ID 为 : %s，可用资金为 : %10.2f" % (accountId, available))

Alert
【说明】
弹出警告提醒
【语法】
void Alert(string Info, bool bKeep=True, string level='Signal')
【参数】
Info(string) 提醒的内容
bBeep(bool) 是否播放警告声音，默认为 True
level(string) 声音类型，包括'Signal'、'Info'、'Warn'、'Error'
【备注】
多行提示信息需要自行换行，例如：

```
AlertStr = '合约: ' + contNo + '
' '方向: ' + self._bsMap[direct] + self._ocMap[offset] + '
' + '数量: ' + str(share) + '
' + '价格: ' + str(price) + '
' + '时间: ' + str(curBar['DateTimeStamp']) + '
'
```

【示例】

```
Alert("Hello")     # 弹出提示
```

context 函数

strategyStatus
【说明】
获取当前策略状态
【语法】
context.strategyStatus()
【参数】
无
【备注】
返回字符，'H' 表示回测阶段；'C' 表示实时数据阶段
【示例】
无

triggerType
【说明】
获取当前触发类型
【语法】
context.triggerType()
【参数】
无

【备注】
返回字符, 'T' 定时触发; 'C' 周期性触发; 'K' 实时阶段 K 线触发; 'H' 回测阶段 K 线触发; 'S' 即时行情触发; 'O' 委托状态变化触发
【示例】
无

contractNo
【说明】
获取当前触发合约
【语法】
context.contractNo()
【参数】
无
【备注】
返回字符串，例如: 'SHFE|F|CU|1907'
【示例】
无

kLineType
【说明】
获取当前触发的 K 线类型
【语法】
context.kLineType()
【参数】
无
【备注】
返回字符, 'T' 分笔; 'M' 分钟; 'D' 日线
【示例】
无

kLineSlice
【说明】
获取当前触发的 K 线周期
【语法】
context.kLineSlice()
【参数】
无
【备注】
返回整型，例如 1
【示例】
无

tradeDate
【说明】

获取当前触发的交易日
【语法】
context.tradeDate()
【参数】
无
【备注】
返回字符串，YYYYMMDD 格式，'20190524'
【示例】
无

dateTimeStamp
【说明】
获取当前触发的时间戳
【语法】
context.dateTimeStamp()
【参数】
无
【备注】
返回字符串， YYYYMMDD 格式，'20190524'
【示例】
无

triggerData
【说明】
获取当前触发类型对应的数据

【语法】
context.triggerData()
【参数】
无
【备注】
K 线触发返回的是 K 线数据，返回字典类型的各键值含义如下：

'ContractNo':	合约编号
'DateTimeStamp':	时间戳
'HighPrice':	最高价
'KLineIndex':	K 线索引
'KLineQty':	K 线成交量
'KLineSlice':	K 线周期
'KLineType':	K 线类型
'LastPrice':	最新价(收盘价)
'LowPrice':	最低价
'OpeningPrice':	开盘价
'PositionQty':	持仓量
'Priority':	历史数据排序使用，无特别含义

'SettlePrice': 结算价
'TotalQty': 总成交量
'TradeDate': 交易日
'DateTimeStampForSort':
交易触发返回的是交易数据，返回字典类型的各键值含义如下：
UserNo : 用户名
Sign ： 标记
Cont : 行情合约
OrderType : 定单类型
ValidType : 有效类型
ValidTime : 有效日期时间(GTD 情况下使用)
Direct : 买卖方向
Offset : 开仓平仓 或 应价买入开平
Hedge : 投机保值
OrderPrice : 委托价格 或 期权应价买入价格
TriggerPrice : 触发价格
TriggerMode : 触发模式
TriggerCondition : 触发条件
OrderQty : 委托数量 或 期权应价数量
StrategyType : 策略类型
Remark :下单备注字段，只有下单时生效。
AddOneIsValid : T+1 时段有效(仅港交所)
OrderState : 委托状态
OrderId : 定单号
OrderNo : 委托号
MatchPrice : 成交价
MatchQty : 成交量
ErrorCode : 最新信息码
ErrorText : 最新错误信息
InsertTime : 下单时间
UpdateTime : 更新时间
StrategyId : 策略 id
StrategyOrderId : 订单的本地号
即时行情触发返回的是即时行情数据，返回字典类型的各键值含义如下：
0： 昨收盘价
1： 做结算价
2： 昨持仓量
3： 开盘价
4： 最新价
5： 最高价
6： 最低价
9： 涨停价
10： 跌停价
14： 收盘价

15： 结算价
16： 最新成交量
17： 最优买价
18： 最优买量
19： 最优卖价
20： 最优卖量
39： 买价 1
40： 买价 2
43： 卖价 1
44： 卖价 2
47： 最新价 1
48： 最新价 2
125： 预留关键字
【示例】
无

标准库说明

标准库是由官方维护会随着语言的演变而演变。标准库提供了设计范围非常广泛的组件，该库包含了许多功能丰富的以 C 编写的内置模块和以 python 编写的模块，因此该库提供了日常编程中许多问题的标准解决方案。
python 标准库的介绍参见标准库介绍。

第三方库说明

极智量化预装了以下用户常用的 python 第三方库，包括：Ta-Lib，pandas，NumPy。

NumPy

NumPy 是使用 Python 进行科学计算的基础软件包。它提供了多位数组对象，各种派生对象，以及用于数组快速操作的各种 API，包括数学，逻辑、形状操作，排序、选择、输入输出、离散傅里叶变化、基本线性代数，基本统计运算和随机模拟等等。这里不再赘述，感兴趣的用户可以参考下方 NumPy 介绍。

- NumPy 中文文档。
- NumPy 英文文档。

Pandas

Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。Pandas 纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具。pandas 提供了大量能使我们快速便捷地处理数据的函数和方法。Pandas 是使 Python 成为强大而高效的数据分析环境的重要因素之一。用户可参考下方 Pandas 文档进一步了解 Pandas 库的详细介绍和使用方法。

Pandas 中文文档。
Pandas 英文文档。

Ta-Lib

Ta-Lib 是一个 python 的金融指数处理库，它广泛应用于交易软件对金融市场数据进行技术分析。Ta-Lib 中包含了 150 多个指标，包括 ADX，MACD，PRS，随机指标，布林带，成交量指标，波动率指标等，以及一些常用的数学运算，数学变换函数。用户可参考下方链接进一步了解 Ta-Lib 库。

Ta-Lib 英文文档
Ta-Lib 函数简介（见群文件）

常见问题

这里总结了使用极智量化过程中的常用问题，这些问题将更好的帮助用户使用极智量化产品。

1. 关于极智量化

1）极智量化支持哪些品种的交易
极智量化目前支持期货、期权、现货、跨期套利、品种套利、外汇、证券等的交易。

2）极智量化目前支持连接哪些柜台
目前内盘支持 CTP 和启明星两种柜台，外盘支持北斗星。

3）极智量化支持内外盘套利么?
支持

4）极智量化支持 A 股实盘交易么
极智量化目前不支持 A 股实盘交易。

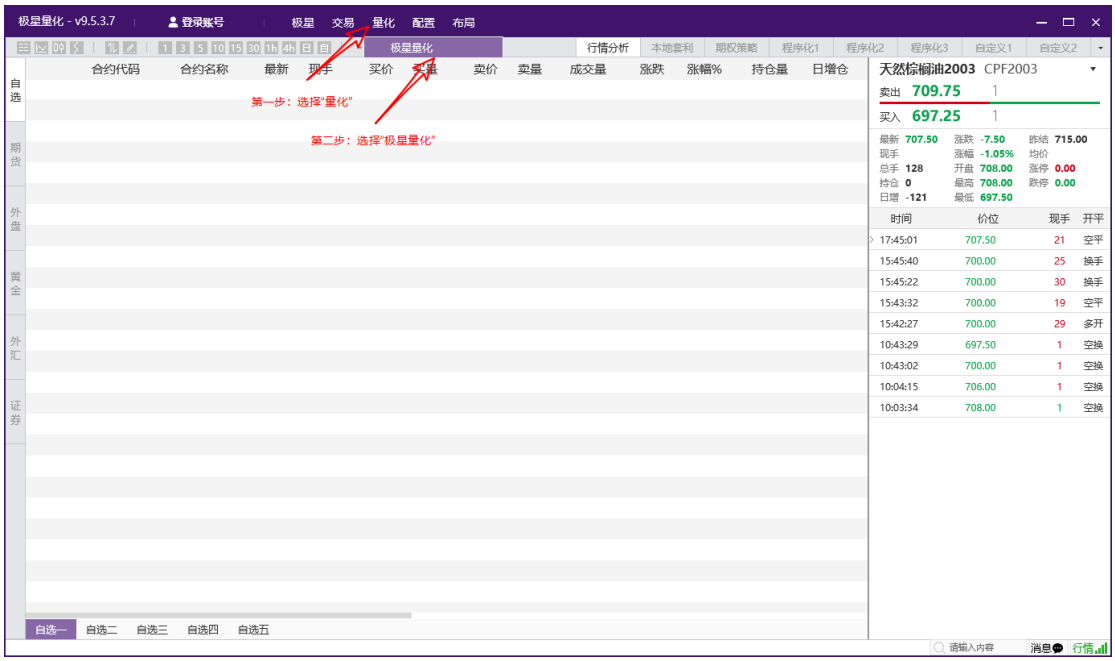
5）极星量化会收费么
极智量化是开源的项目，在符合 GPL2.0 协议的前提下，可以免费使用，自由开发，是完全免费的产品，用户可以放心使用。

6）使用过程中遇到问题怎么办?
我们鼓励您查看帮助文档查找问题的解决方法。如果仍然不能解决问题，请到 QQ 群: 472789093 详细描述您遇到的问题，我们的开发人员会在第一时间解决您遇到的问题。

7）常见 bug 或者警告的解决方法
Pass

2. 极智量化使用

1）如何打开量化窗口
如下图所示:



Step1：点击界面上的"量化"按钮；

Step2：选择"极星量化"

注意：第一次运行量化窗口时，打开界面过程可能会有些慢，请耐心等待 10 秒钟左右。

2）如何打开策略调试模式

极智量化终端目前已提供 LogInfo()、LogWarn()等函数，可打印日志到终端并写入日志文件。具体 LogInfo 函数的用法，请参阅 API 函数说明。如果客户想要使用 python 的 print 函数，则可通过修改极星 9.5 客户端目录中 equant.vbs 脚本文件的文件名来打开调试模式。

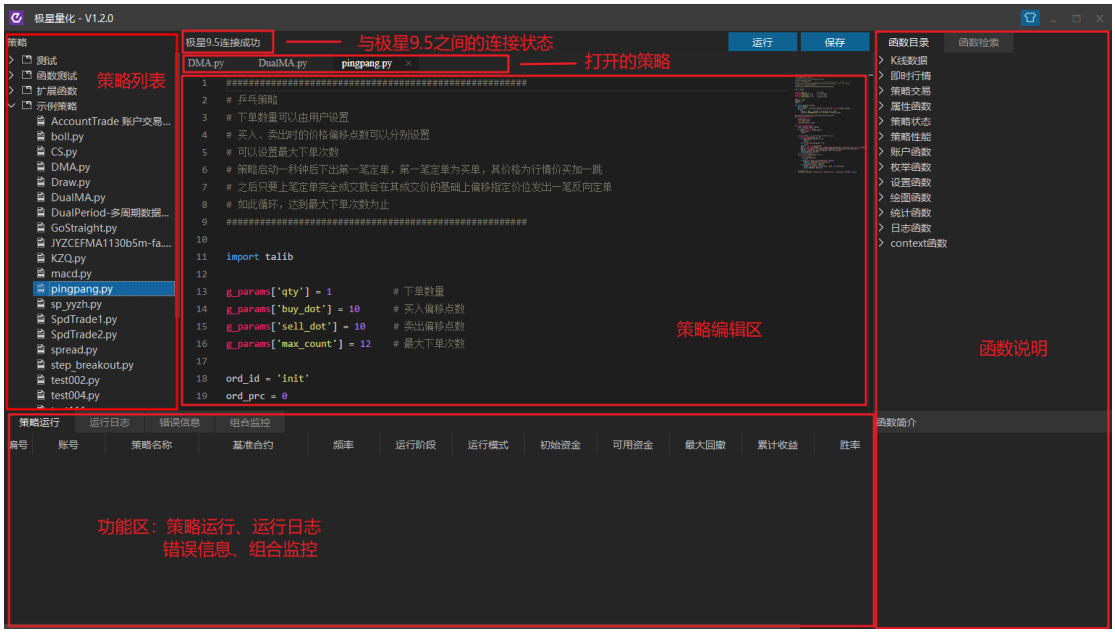
首先用户找到 equant.vbs 文件的文件位置：在桌面上找到极星量化图标，然后右键选择"打开文件所在的位置"，修改文件名，之后重新启动极星量化，打开量化窗口时会弹出命令行窗口，调试信息会显示在该命令行窗口中。

3）如何关闭策略调试模式

方法是上面方法的相反过程，用户将修改过的 equant.vbs 文件名重新修改为 equant.vbs，然后重启极星量化客户端，打开量化窗口，调试模式的命令行窗口将不显示。

4）量化主界面介绍

量化主程序如图：



策略列表：策略列表中展示了一些示例策略，用户可以在该区域进行策略的新建、删除、导入、重命名、定位等操作；

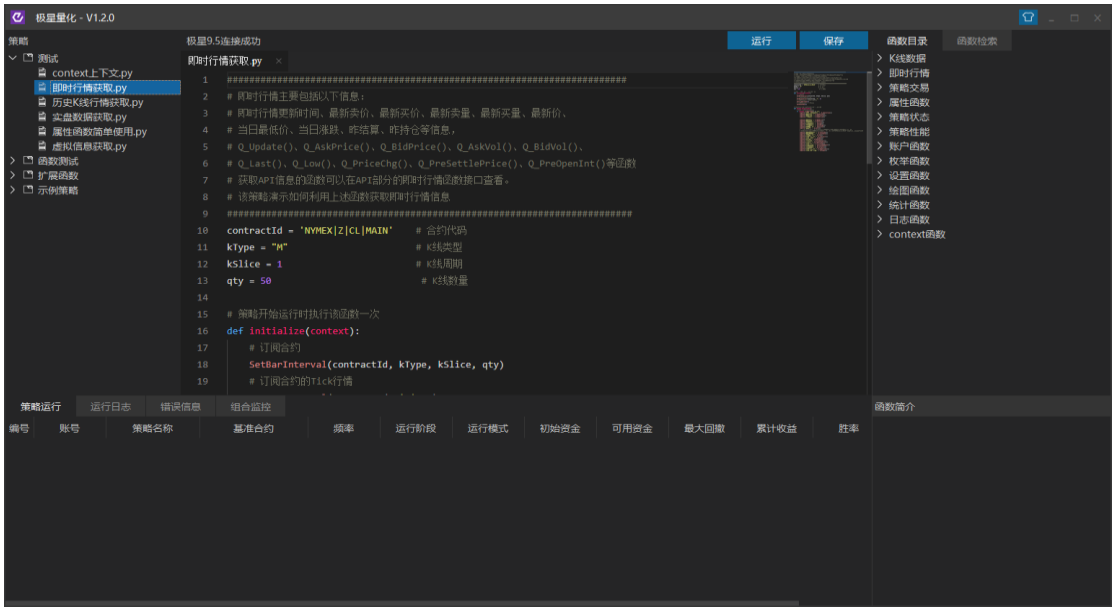
策略编辑区：用于展示打开的策略并提供策略的编辑功能，用户可在此区域编辑策略；

功能区：功能区提供了策略运行信息展示、运行日志（包括：用户日志、系统日志、下单信号）、错误信息、组合监控四部分。策略运行展示区可以对策略进行启动、停止、删除控制，并可以查看测试报告、图标展示和属性设置操作；系统日志部分用于显示系统日志、下单信号以及策略运行过程中用户输出的用户日志；错误信息处显示策略运行的错误信息；组合监控区提供持仓同步功能；

函数说明区：用于展示极智量化提供的 API 函数列表和每个函数的使用说明。

5）如何查看示例策略

在打开的量化窗口主界面的策略列表中选择一个策略，双击即可打开策略：

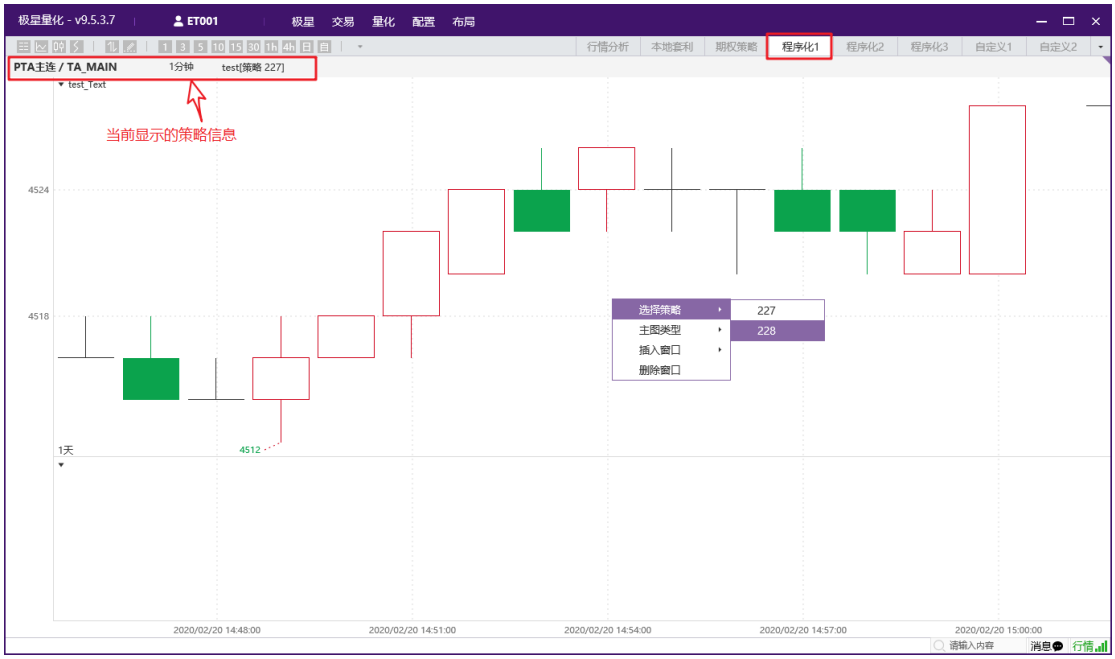


6）如何切换图表展示

有两种方式可以切换极星量化上程序化的图表展示：

方法 1：策略监控列表中，在策略上右键选择“图表展示”，则在极星 9.5 客户端上会展示所选的策略的运行图表

方法 2：在极星 9.5 程序化页面，鼠标右键选择“选择策略”，然后选择相应的策略 id 号即可切换图表展示，如下图所示：



7）如何修改极智量化界面的背景色？

在极智量化的标题栏，鼠标单击换肤按钮即可实现极智量化主界面的背景色修改。目前极智量化提供了两种背景色：纯黑色背景和纯白色背景。

8）如何使用极智量化主界面函数搜索功能？

主界面的函数检索功能可以很方便帮助用户查找所需的函数。

可以在搜索框中输入函数名或者函数描述的关键字，下方将列出所有满足条件的搜索结果。如下图在搜索框中输入“开盘”，这里将列出函数描述中所有包含“开盘”关键字的函数，单击搜索结果即可查看函数说明。

9）如何将指标在策略图表的主图或副图上展示上展示
极智量化提供了丰富的绘图函数供用户调用，具体函数可在 api 中的绘图函数部分查看，其中每个绘图函数都有一个 main 参数，可以设置 main=True 表示在主图上加载指标，main=False 表示在副图上加载指标，具体使用方法见示例策略—基本使用中的“在主图上画指标线”和“在副图上画指标线”

10）极智量化图表展示支持多个副图展示么？
目前极智量化只支持显示一个副图。

11）极智量化支持多账户交易么？
极智量化支持多账户交易。多账户交易需要用户在极星 9.5 登录交易账号，并在策略中利用 SetUserNo()函数设置需要进行交易的账户。

12）极智量化支持期权自动化交易么？
极智量化支持自动化交易。对于用户来讲，需要确定交易账户是否拥有期权合约的交易权限。需要说明的是：目前用户申请的内盘模拟账户默认没有期权交易的权限，外盘模拟账户有期权交易的权限。若需要内盘模拟账户拥有期权交易权限，需要联系管理员开通。

13）历史数据支持本地下载么？
目前历史数据不支持本地下载。如果用户想获得本地数据的话，可以参考示例策略—基本使用中“读写 Excel 文件”，将获取到的历史数据保存到本地。

3. 策略运行

1）关于界面设置和代码设置生效问题
极智量化提供了两种设置策略运行初始条件的方式，一种是通过运行设置界面设置，另一种是通过策略的 initialize()函数设置策略的运行条件。
这里就涉及两种设置方法的优先性问题。两种设置方法都将生效，但如果两种设置方法存在冲突的情况，策略将以 initialize()函数中设置的参数为准运行策略。如界面上在合约设置

出选择了合约，但是在策略 initialize()函数中通过 SetBarInterval 函数也设置了合约，则策略将以 initialize()函数中设置的合约运行策略。
注意：设置触发方式时，如果界面和代码中均设置了触发方式，如果设置的触发方式不冲突的情况下，策略将取两种设置的触发方式的并集运行策略。

2）什么是基准合约?
如果同时订阅了多个合约的数据，极智量化将自动将第一个合约作为基准合约，基准合约是作为图表展示的合约，同时许多 API 函数中参数为合约编号的函数的合约编号的缺省值也是基准合约。

3）关于发单机制中 K 线稳定后发单和实盘发单的含义?
用户可以通过界面设置发单时机，也可以通过 SetOrderWay()在策略 initialize()函数中设置发单方式。发单时机的设置是针对策略在实盘阶段运行的，历史回测阶段的发单方式只能是 K 线稳定后发单。
K 线稳定后发单：K 线稳定后发单与用户订阅的合约 K 线类型、K 线周期有关。当用户订阅的是五分钟的 K 线，K 线稳定后发单是当 5 分钟 K 线完成后策略才会被触发，此时若策略满足发单条件，则发送委托，不满足则不发送委托。
实时发单：当策略被触发且满足发单条件时立即发送委托。注意：此时若用户订阅的触发方式中有 K 线触发时，实时阶段 TICK 也被作为 K 线处理，因此策略每个 TICK 都会被触发。

4）如何判断策略为何种方式触发?
策略触发方式的判断可参考示例策略—基本使用—6.触发方式判断

5）策略文件运行后，监控列表中停止该策略，然后对策略进行改动，在监控列表中启动策略说明
策略 A 运行后在监控列表中会展示策略运行信息，此时在监控列表中右键选择“停止”暂停策略运行，然后对策略文件进行修改并保存。再在策略监控列表中对暂停的 A 策略右键选择“启动”，则此时将以修改后的策略运行，不过对策略的 initialize 函数进行的修改不生效。

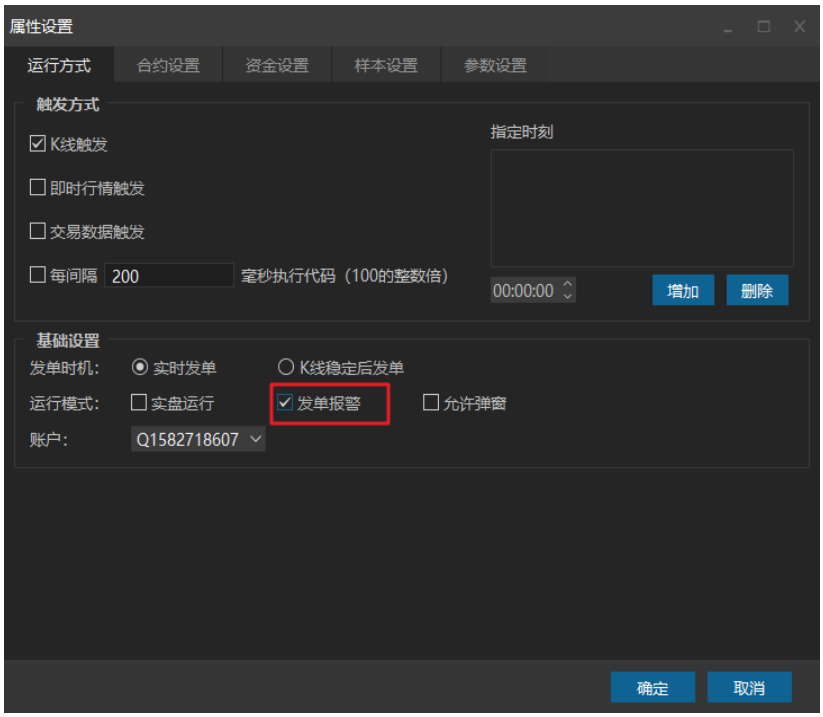
6）策略每次被触发运行的时间和触发间隔时间的说明
策略每次被触发运行 handle_data()需要一定的时间 t1，该时间为策略运行一次需要的时间，策略两次触发的时间间隔记为 t2，若 t1<t2，表示策略运行的时间小于策略触发的时间，此时策略运行的时间和策略触发之间不存在冲突。若 t1>t2，则表示策略运行的时间大于策略触发的间隔时间，此时每次策略被触发都会造成后续触发的延时。例如：策略为即时行情触发，新行情每隔 500 毫秒触发一次策略，若 handle_data 运行的时间是 100 毫秒，第一个行情在 0 时刻触发策略，handle_data 运行时间是 100ms，handle_data 运行完继续等待触发，在第 500ms 时一个新行情到来，重新触发策略，策略运行时间 100ms，运行完继续等待触发。
若策略运行时间为 1 秒的话，第一个行情在 0 时刻触发策略，此时 handle_data()运行，运行时间为 1 秒，这样策略在第 1 秒时才能运行结束，而第二个触发条件在第 500 毫秒已经到来，由于策略第一个触发策略还未运行完，因此到来的触发条件只能等待策略运行完才

能触发策略继续进行，这样就会对策略下一次的触发造成延时，随着策略的运行，后序触发造成的延时会不断累加。

7）运行策略如何切换合约?
用户在界面上设置合约运行的策略，每次切换合约时需要再界面上重新设置合约然后运行;
用户在策略代码的 initialize()函数中设置合约运行的策略，每次需要修改代码中的合约代码然后运行

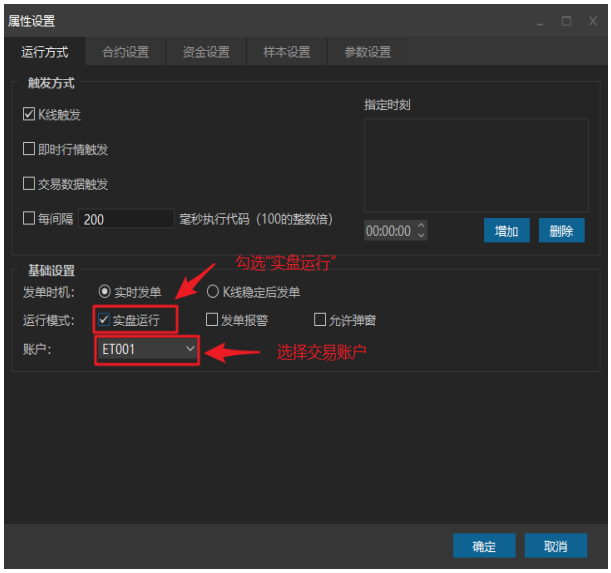
8）策略运行过程中，账户掉线，策略会下单失败么？重新连接后呢?
策略在实盘运行时，账户掉线的话会导致下单失败。
若用户重新登录上账户，则策略会继续下单。

9）如何加入声音预警
极智量化提供了发单预警功能，若用户需要下单提醒的话，可以在界面上勾选“发单预警”功能:



若用户需要自定义加入预警功能，也可以安装第三方声音模块的库并在策略中使用。

10）策略如何实盘运行
要让策略在实盘阶段运行并交易，需要设置策略实盘运行,可以通过运行设置界面设置，也可以在策略中调用 SetActual(),除了设置实盘运行外，还需要在极星 9.5 客户端登录对应的内盘或外盘交易账户，并在运行设置界面选择想要进行交易的账户，或者通过 SetUserNo()在策略中设置要进行交易的账户，内盘合约要登录并设置对应的内盘交易账户，外盘合约要登录对应的外盘交易账户只有满足这两个条件策略才能进行实盘交易。
界面上的设置方法如下:



策略中的设置方法见示例策略—基本使用中的实盘运行策略。

12）极智量化如何下套利合约单

极智量化支持下套利单，具体方法见示例策略策略交易中的 12.套利合约交易演示 1 和 13.套利合约交易演示 2.

13）同时订阅一个合约的大小周期问题?

同时订阅一个合约的大小周期存在问题。

14）对郑商所、大商所的合约下单时指定订单平今平昨标志为平今会生效么?

设置订单的平今平昨标志位为平今时当且仅当下单合约为上期所的合约时该参数才会生效，对于非上期所的合约设置平今平昨标志为平今时，极智量化自动处理为平仓。

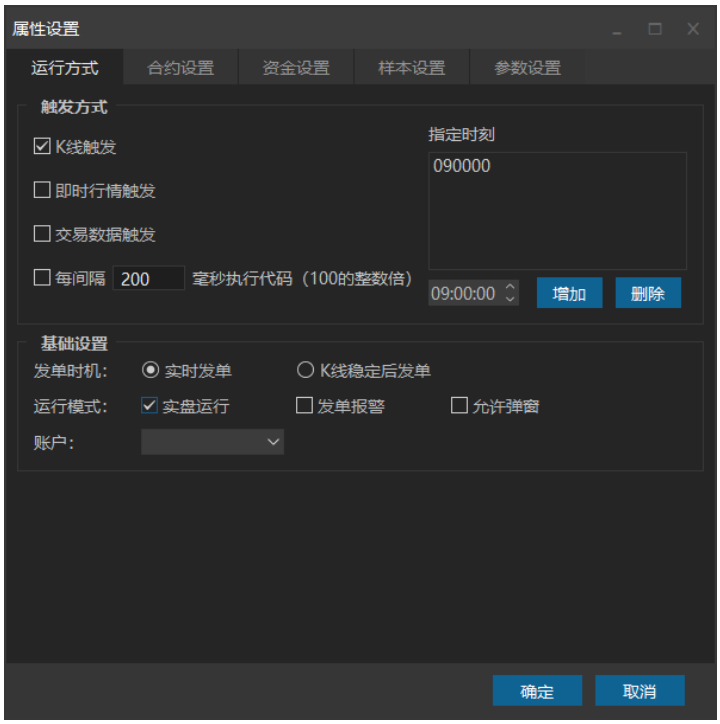
4. 运行设置说明:

极智量化提供两种设置策略运行的方式，通过界面的属性设置对策略运行进行设置或在策略的初始化函数中设置，这里介绍通过界面设置策略运行属性。

点击极智量化主界面上的"运行"按钮，弹出策略运行设置对话框。该对话框上的选项卡列出了运行方式、合约设置、资金设置、样本设置、参数设置。

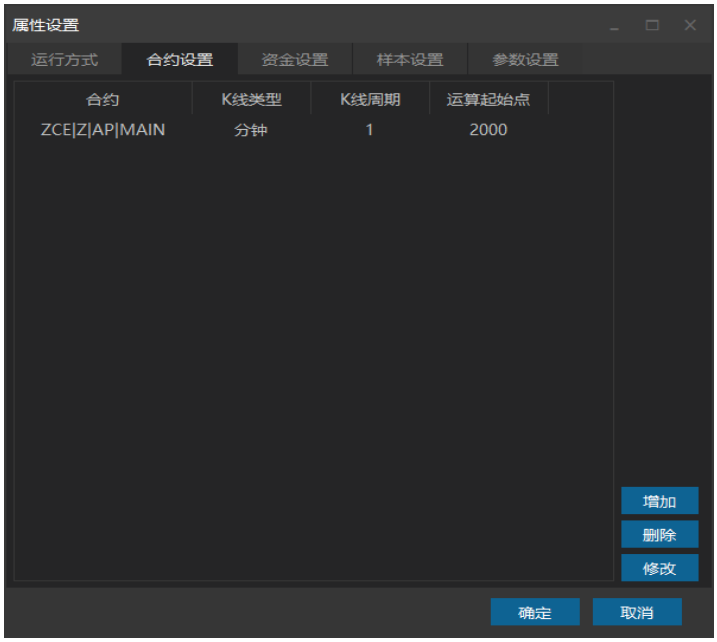
1）运行方式

运行方式可以设置触发方式、发单设置、运行模式、交易账户



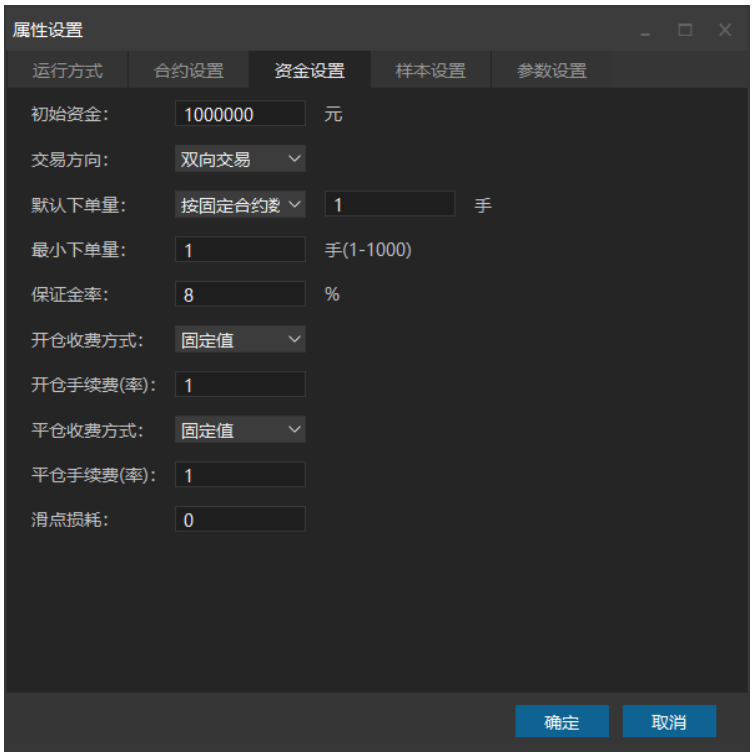
触发方式说明可在十分钟入门教程中的触发方式查看。

2）合约设置



合约设置中可以设置需要进行回测的合约，可以添加多个合约，也可以对添加的合约进行修改和删除，将添加的第一个合约作为基准合约。

3）资金设置



资金设置可以设置以下信息:

初始资金: 虚拟交易的初始资金。在"初始资金"输入框中输入资金。设置完成后，交易指令在进行测试时，用该资金进行盈利计算。初始资金不能小于 1000 元。

交易方向: 设置当前策略的交易方向: 双向交易、仅多头、仅空头。

默认下单量:

按固定合约数: 指定默认的下单手数

按资金比例: 使用全部资金的百分比来下单，下单量等于总资金乘以百分比除以订单价格

按固定资金: 使用指定资金数来下单，下单量等于指定资金除以订单价格

最小下单量: 每手交易的最小下单手数

保证金率: 每手交易的保证金

开仓收费方式: 进行开仓操作时手续费收费方式: 固定值、比例

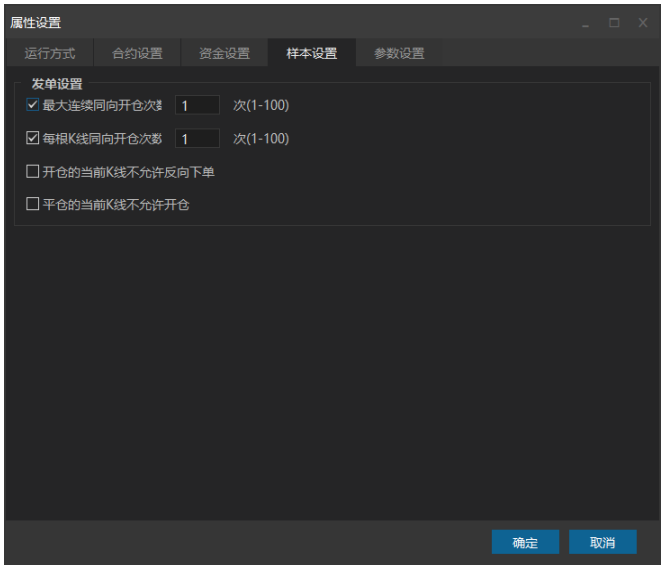
开仓手续费: 开仓手续费收取方式为固定值时，每手交易按设置的手续费固定值收取，收取方式为比例时，手续费为订单价格*每手乘数*比率(%)

平仓收费方式: 进行平仓操作时手续费收费方式: 固定值、比例

平仓手续费: 平仓手续费收取方式为固定值时，每手交易按设置的手续费固定值收取，收取方式为比例时，手续费为订单价格*每手乘数*比率(%)。

滑点损耗: 下单的点位和最后成交的点位的差值。

4) 样本设置



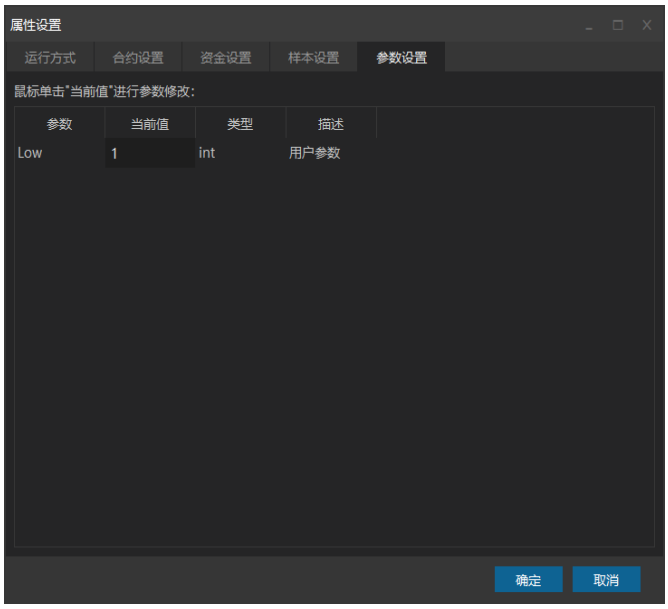
最大连续同向开仓次数： 不选中则表示执行程序化策略时连续多根 K 线达到开仓条件时不做限制，始终执行策略；选中则表示执行程序化策略时连续多跟 K 线达到开仓条件时会根据用户设置的次数执行策略，若超出用户设置的次数则跳过，次数设置输入范围在 1-100 之间"

每根 K 线同向开仓次数：不选中则表示执行程序化策略时同一根 K 线达到多次开仓条件时不做限制，始终执行策略；选中则表示执行程序化策略同一根 K 线达到多次开仓条件时会根据用户设置的次数执行策略，若超出用户设置的次数则跳出，次数设置输入范围在 1-100 之间

开仓的当前 K 线不允许反向下单：不选中则表示执行程序化策略时在同一根 K 线上开仓之后再触发平仓条件时不做限制，执行策略；选中则表示执行程序化策略时在同一根 K 线上开仓之后再发出平仓条件时不做处理

平仓的当前 K 线不允许开仓：不选中则表示执行程序化策略时在同一根 K 线上平仓之后再触发开仓条件时不做限制，执行策略；选中则表示执行程序化策略时在同一根 K 线上平仓之后再触发开仓条件时不做处理

5）参数设置



当策略中包含用户参数时，该界面会显示用户参数，可以对设置的用户参数值进行修改。具体用户参数说明参见十分钟入门教程中用户参数的说明。

5. 回测

1）回测的作用?
策略有效性体现了用户的交易思想，通过历史数据的回测，可以检测策略的有效性

2）极智量化最多提供多少历史数据用于历史回测?
极智量化由于后台限制的原因，目前提供用于历史回测的最大数据量是 8 万根。

3）如何设置可以使策略只进行历史回测，不运行实盘阶段的数据?
极智量化默认是运行完用户订阅的历史数据后自动运行实时阶段的数据的。若指向策略进行历史回测，可在策略的 handle_data()函数的起始位置加入以下代码:

```
if not context.triggerType() == "H":   # 判断触发方式是否为历史数据触发
    return
```

4）回测可以用 A_SendOrder()函数下单么?
回测时可以用 A_SendOrder()函数下单。
回测阶段用 ASendOrder()函数下单程序将自动转化为利用 Buy()、Sell()函数下单

5）回测阶段的开平仓机制是如何撮合成交的
回测阶段的撮合基准采用的是虚拟后台的撮合机制，此时若有委托单，且该订单满足资金、仓位的条件，则虚拟后台会立即撮合成交。

6）回测支持 TICK 数据回测么?
极智量化支持 TICK 级数据回测。TICK 回测时订阅的 K 线类型为"T", K 线周期为 0，如:

```
SetBarInterval("ZCE|Z|TA|MAIN", "T", 0, 8000)
```

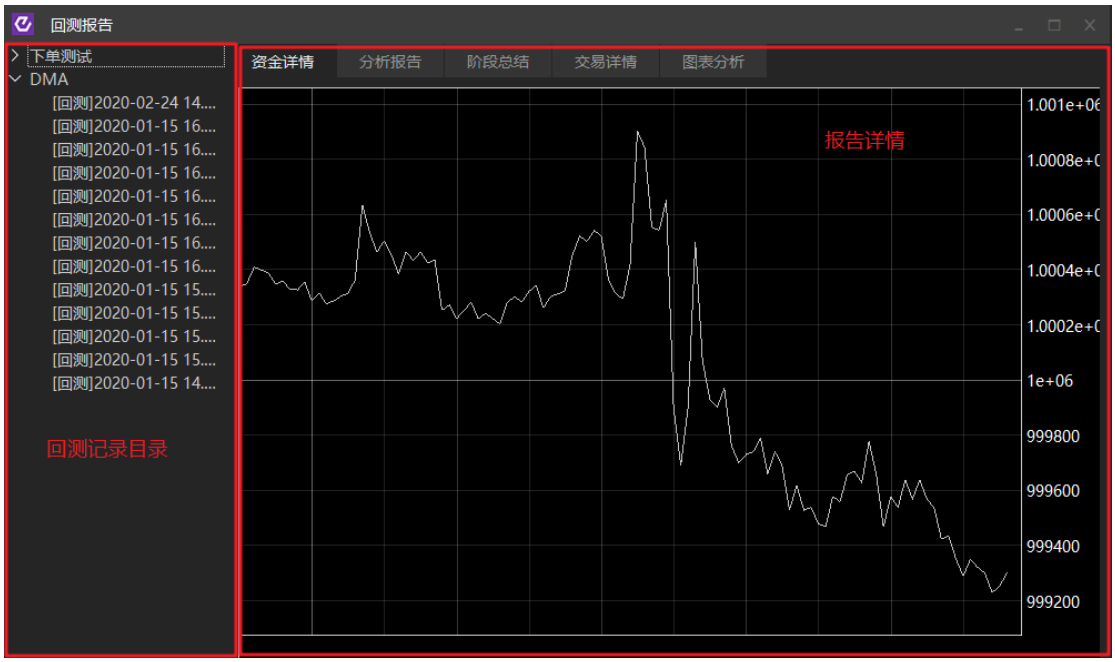
7）极智量化回测如何订阅秒线数据?
订阅秒线数据时，只需选择 K 线类型为"T"，K 线周期为不为 0 的值即可:

```
SetBarInterval("ZCE|Z|TA|MAIN", "T", 1，1000)    # 订阅合约 1 秒线
```

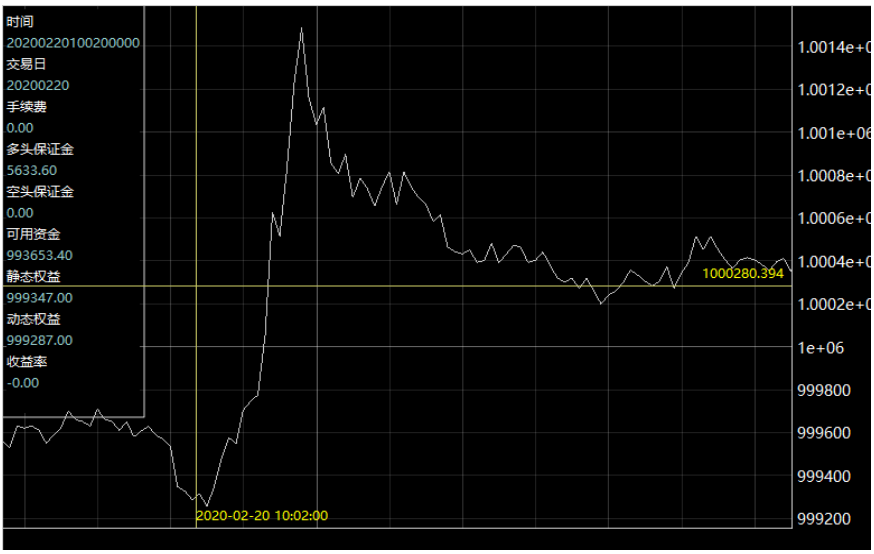
8）历史回测和实盘运行有何区别?
历史回测可以更好的检验策略的有效性。历史回测时所用到的数据为历史阶段的数据，此时策略的触发方式只能是 K 线触发，发单方式只能为 K 线稳定后发单，因为历史阶段 K 线已经完成且稳定。
实盘阶段运行策略可以更好的检验策略在实盘阶段的表现。和历史阶段不同，实盘阶段数据存在更多的不确定性。
以下列出了历史回测和实盘运行的不同之处:
手续费不同：模拟交易可以自己设置手续费，实盘下单时手续费是由交易所和期货公司设置的;
滑点影响：历史回测时滑点可以自行设置来模拟实盘阶段的滑点，实盘阶段成交结果以交易所成交结果为准，系统只是同步交易所成交结果。

下单数量的限制：历史阶段对下单数量没有限制，实盘阶段的下单数量要小于单笔最大数量的限制才能委托成功；
撮合机制不同：历史阶段的订单撮合机制和实盘阶段的撮合机制不同，历史阶段的订单撮合机制为回测后台撮合的，此时订单只要满足开平仓的仓位以及可用资金允许下单，订单就可以成交。实盘阶段的成交是由交易所进行撮合的。
资金、持仓信息不同：历史回测阶段运行时的初始资金是由用户设置的，历史阶段对应的账户是由回测引擎模拟的虚拟账户，用户持仓的初始值为 0，实时阶段的资金和账户持仓对应的是真实交易账户的资金和持仓信息。

9）回测报告说明
回测报告界面如图所示：



回测报告包含回测报告目录和报告详情两部分，通过单击左侧目录可以切换回测报告。
报告详情包括资金详情，分析报告，阶段总结，交易详情，图标分析五部分，下面对着五部分进行介绍：
资金详情：资金详情记录了虚拟初始资金的每一笔资金变化，鼠标点击资金详情曲线会有资金的详细信息，点击键盘上的上下键可以放大和缩小曲线，鼠标左右拖动可以拖动资金曲线：



分析报告：分析报告对交易策略的报表进行了详细分析

这里列出了分析报告的每一项的含义

资金	初始投入总资金
合约信息	交易合约信息
计算开始时间	回测开始时间
计算结束时间	回测结束时间

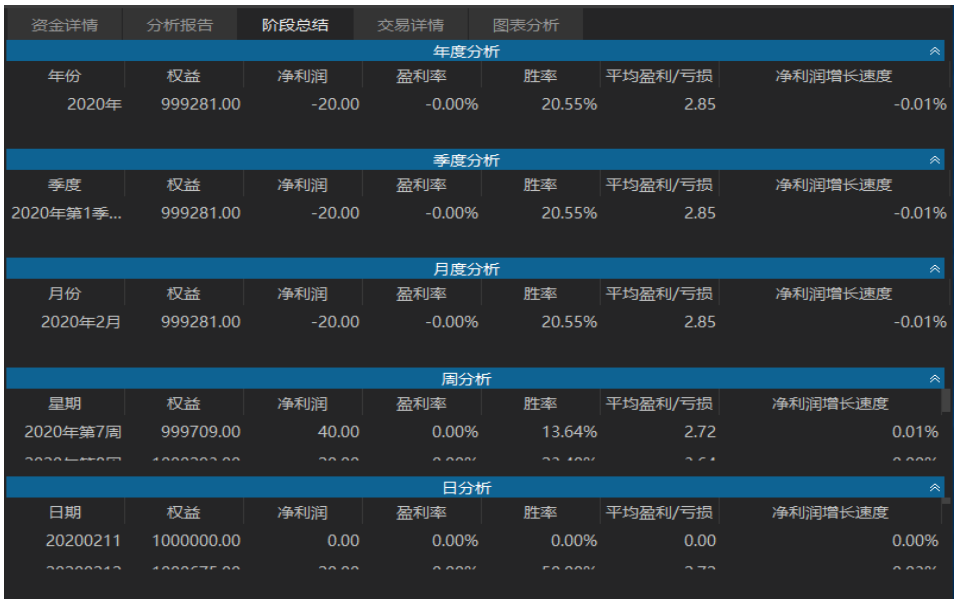
测试天数	测试天数
最终权益	包括当前的可用资金和浮动盈亏
空仓周期数	没有持仓的周期数
最长连续空仓周期数	连续无持仓的 K 线根数
标准离差	$\left(\sqrt{\left(\sum_1^N \left(\text{每次盈亏} - \text{平均盈亏}\right) \wedge 2\right)/N}\right.$，注意每次亏损代入的为负值)
标准离差率	标准离差/平均盈亏
夏普比率	夏普比率计算公式： = [E(Rp) - Rf]/σp； E(Rp)：平均年收益率 = 年化单利收益率 Rf：无风险利率（大约是 3%） σp：收益率的标准差率（年化标准差率） = 标准差率/sqrt (测试天数/365) 标准差率 = 标准离差/初始资金
***盈亏总平均/亏损平均**	（（总盈利-总亏损）/交易次数） / （总亏损/亏损交易次数）
权益最大回撤	从测试开始到结束，动态权益计算出来的波段从高点到低点回撤的最大值

权益最大回撤时间	最大回撤出现的时间
权益最大回撤比	最大回撤和最大回撤时的最大权益的比值的最大值
权益最大回撤比时间	最大回撤比出现的时间
风险率	本金最大回调/本金（本金最大回调=计算比初始资金小的权益当中，回调最大的值，即最小权益与初始权益的差值）
收益率/风险率	收益率/风险率（收益率指的是年化单利收益率，本金：初始资金。年化单利收益率=盈利/初始资金/(交易天数/365)，其中，交易天数指的是回测报告中的测试天数）
盈利率	盈利占初始资金百分比（盈利为平仓盈亏的和）
年化单利收益率	单利收益率/（交易天数/365）（单利收益率=盈利率=盈利/初始资金）
月化单利收益率	单利收益率/（交易天数/30）（单利收益率=盈利率=盈利/初始资金）
年化复利收益率	期末权益/起初权益^（365/交易天数）-1
月化复利收益率	期末权益/起初权益^（30/交易天数）-1
胜率	非亏损交易周次数/总交易次数（非亏损交易次数/总交易次数）

***平均盈利/平均亏损**	平均亏损 = 总亏损/亏损交易次数；平均盈利 = 总盈利/盈利交易次数
净利润	净利润 = 总盈利 – 总亏损
总盈利	盈利的总和
总亏损	亏损的总和
其中持仓浮盈	其中持仓的浮动盈亏，最后交易状态如果是有持仓，按照收盘价计算盈亏。（算法怎么实现呢）
***交易次数**	发生交易的次数
***盈利比率**	盈利次数/总交易次数
***盈利次数**	盈利的交易次数
***亏损次数**	亏损的交易次数
***持平次数**	持平的交易次数
平均盈亏(利润)	总利润率/总交易次数
平均盈利	平均盈利 = 总盈利/总盈利次数（计算手续费）
平均亏损	平均亏损 = 总亏损/总亏损次数（计算手续费）
盈利持续最大天数	盈利持续的最大天数
盈利持续最大天数出现时间	盈利持续的最大天数出现时间

亏损持续最大天数	亏损持续的最大天数
亏损持续最大天数出现时间	亏损持续最大天数出现的时间
盈利环比增加持续最大天数	盈利环比增加持续的最大天数
盈利环比增加持续最大天数出现时间	盈利环比增加持续的最大天数出现时间
亏损环比增加持续最大天数	亏损环比增加持续的最大天数
亏损环比增加持续最大天数出现时间	亏损环比增加持续的最大天数出现时间
手续费	交易产生的总手续费
成交额	成交价*（开仓或者平仓手数）*交易单位
滑点损耗	

阶段总结：阶段总结包含年度分析、季度分析、月度分析、周分析、日分析。


资金详情 | 分析报告 | 阶段总结 | 交易详情 | 图表分析

年度分析

年份	权益	净利润	盈利率	胜率	平均盈利/亏损	净利润增长速度
2020年	999281.00	-20.00	-0.00%	20.55%	2.85	-0.01%

季度分析

季度	权益	净利润	盈利率	胜率	平均盈利/亏损	净利润增长速度
2020年第1季...	999281.00	-20.00	-0.00%	20.55%	2.85	-0.01%

月度分析

月份	权益	净利润	盈利率	胜率	平均盈利/亏损	净利润增长速度
2020年2月	999281.00	-20.00	-0.00%	20.55%	2.85	-0.01%

周分析

星期	权益	净利润	盈利率	胜率	平均盈利/亏损	净利润增长速度
2020年第7周	999709.00	40.00	0.00%	13.64%	2.72	0.01%

日分析

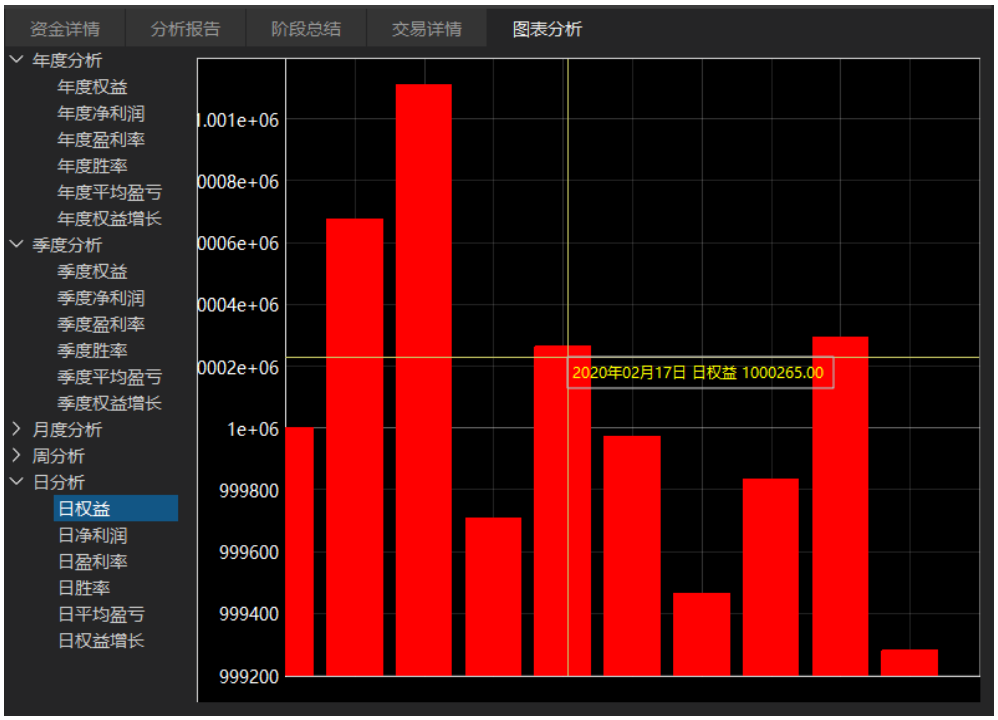
日期	权益	净利润	盈利率	胜率	平均盈利/亏损	净利润增长速度
20200211	1000000.00	0.00	0.00%	0.00%	0.00	0.00%

交易详情：记录了每次下单的详细信息


资金详情 | 分析报告 | 阶段总结 | 交易详情 | 图表分析

时间	合约	交易类型	下单类型	成交数量	成交价	成交额	委托数
20200212 ...	ZCE|F|AP|005	买开	限价单	1	7016.00	70160.0	
20200212 ...	ZCE|F|AP|005	卖平	限价单	1	7024.00	70240.0	
20200212 ...	ZCE|F|AP|005	卖开	限价单	1	7024.00	70240.0	
20200212 ...	ZCE|F|AP|005	买平	限价单	1	7018.00	70180.0	
20200212 ...	ZCE|F|AP|005	买开	限价单	1	7018.00	70180.0	
20200212 ...	ZCE|F|AP|005	卖平	限价单	1	7010.00	70100.0	
20200212 ...	ZCE|F|AP|005	卖开	限价单	1	7010.00	70100.0	
20200212 ...	ZCE|F|AP|005	买平	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	买开	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	卖平	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	卖开	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	买平	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	买开	限价单	1	7011.00	70110.0	
20200212 ...	ZCE|F|AP|005	卖平	限价单	1	7006.00	70060.0	
20200212 ...	ZCE|F|AP|005	卖开	限价单	1	7006.00	70060.0	
20200212 ...	ZCE|F|AP|005	买平	限价单	1	7007.00	70070.0	
20200212	ZCE|F|AP|005	买开	限价单	1	7007.00	70070.0	

图表展示：图表分析是对阶段总结的信息的展示



6. 其他常见问题

1）python 中索引与切片的用法
在使用 API 函数过程中经常遇到 Close()[-1]、Open()[-1]等利用索引取值的操作，具体-1 代表什么含义，用户可以参考这里了解 python 中索引与切片的用法。

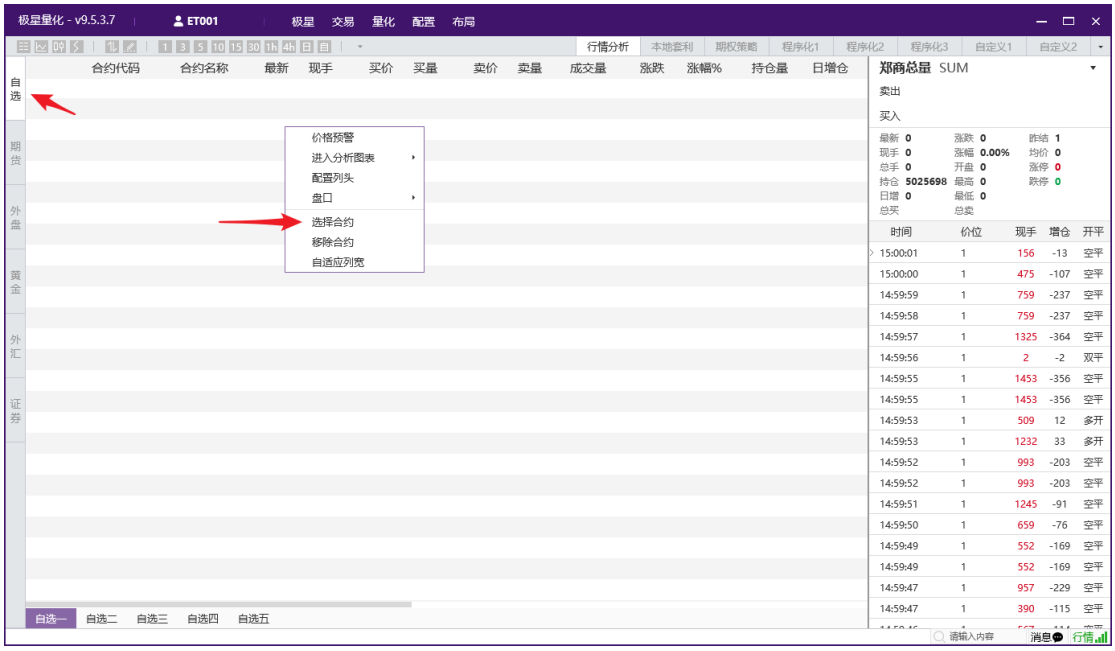
2）策略监控列表中的策略 id 如何归零？
在安装文件所在文件夹下，找到./equant/src/config/StrategyContext.json 文件，用记事本打开该文件，在文件的末尾有一个“MaxStrategyId”字段，将后面的数字改为 0 即可。

3）合约代码的组成及其含义
合约代码一般有四个部分组成，分别为：交易所、品种类型、品种、合约，每部分之间由竖线分割，如下是郑商所 PTA005 合约的合约代码：
ZCE|F|TA|005
其中：ZCE 表示郑商所，还可以为 DCE、SHFE、CFFEX、NYMEX、COMEX 等
F 代表期货，其他的品种类型包括：
Y：现货
O：期权
S：跨品种套利
Z：指数
M：跨品种套利
T：股票
X：外汇
I：外汇
C：外汇

TA：品种代码，表示 PTA
合约：具体月份，005 表示 5 月份

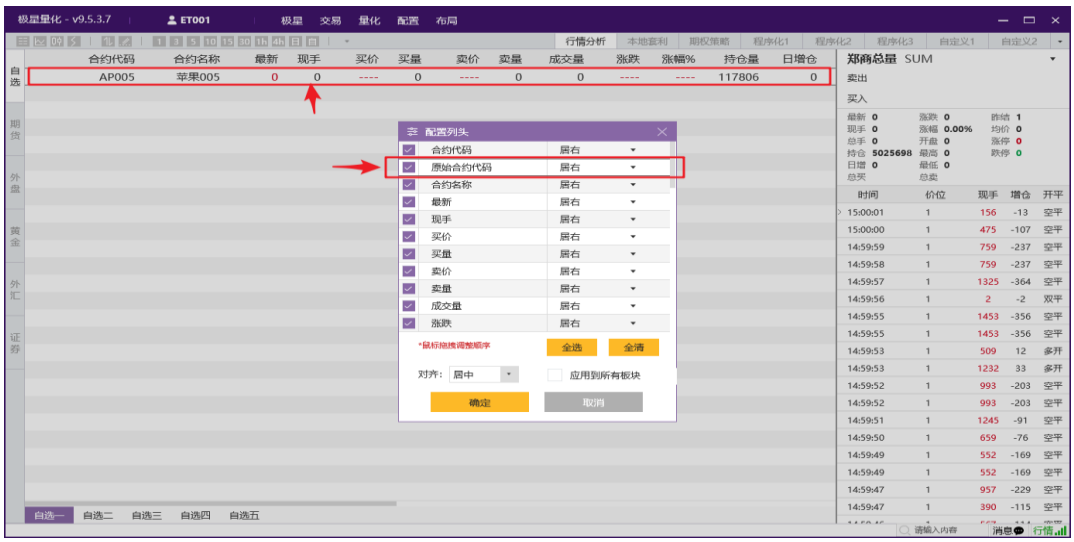
4）开发策略所用的语言是什么？
目前极智量化开发策略支持的编程语言为 python。

5）如何查看某个合约的代码？
在极星 9.5 客户端上的自选模块，右键选择“选择合约”，如下图所示：



在弹出的“选择合约”窗口上选择想要查看的合约，然后点击“确定”按钮：

在添加的合约上右键选择配置列头，在弹出的窗口上勾选上“原始合约代码”，点击确定：

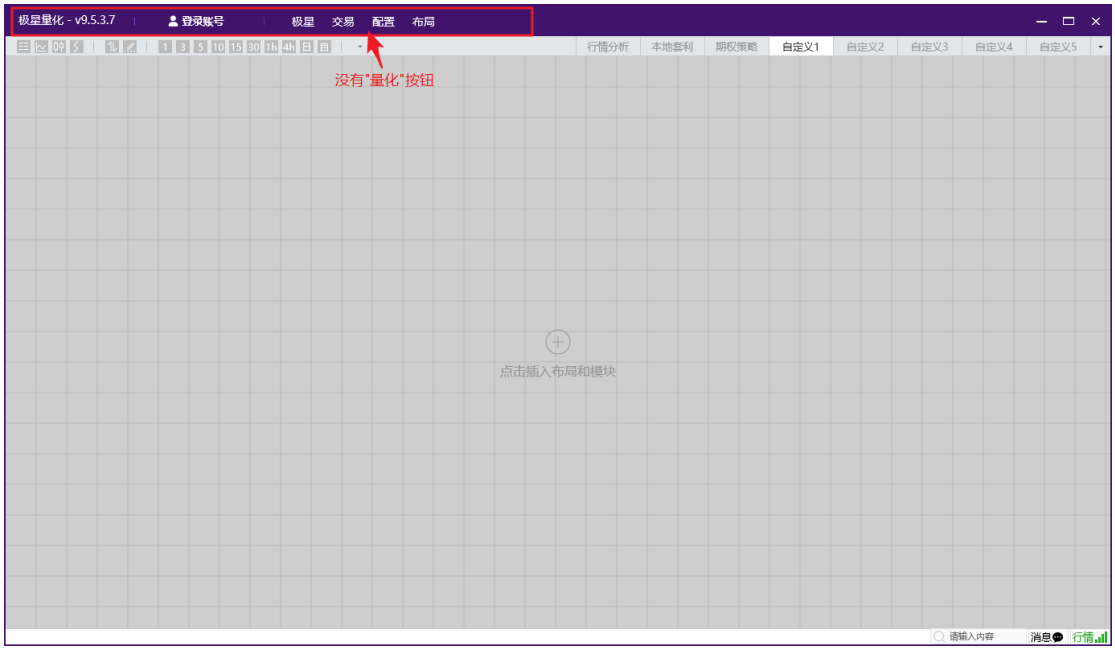


即可查看合约代码：

注意：上海黄金交易所和外汇品种，由于其原始合约代码中只到品种，需要再原始合约后面加上“|”，如上海黄金交易所的黄金 100g 的原始合约代码为：”SGE|P|AU100G|”；另外合约代码中不能包含加号，因此上海黄金交易所的白银(T+D)的正确合约代码为：”SGE|Y|AG(TD)|”。

6）启动极星 9.5 后为什么没有量化按钮？

如图，启动极星 9.5 之后”量化”按钮不显示：



这种情况出现的原因有两种可能：

- 极星 9.5 关闭后极智量化主程序未关闭

- 极智量化不正常退出引起的 python 进程未全部关闭

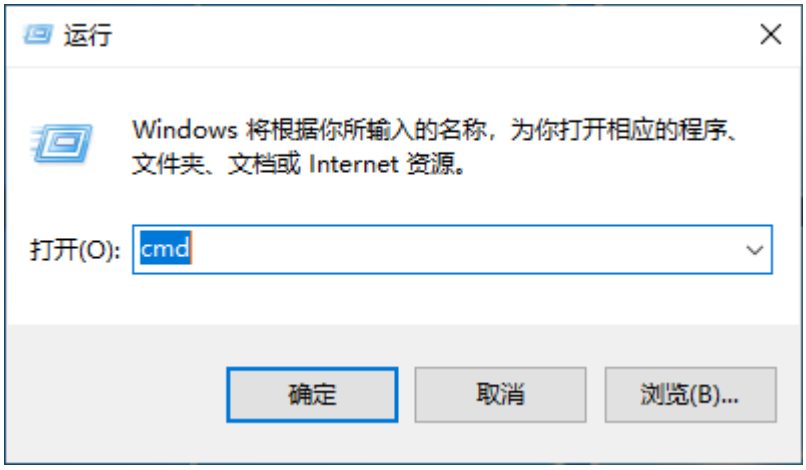
对于第一种情况，解决方法是关闭极智量化和极星 9.5 客户端，然后重新启动极星 9.5 即可；
对于第二种情况，解决方法是打开任务管理器，关闭进程管理器中未关闭的 python.exe 进程，然后宠幸启动极星 9.5 即可。

7）极智量化的日志文件存放在什么位置？
极智量化的日志文件包括系统日志、用户日志、下单信号日志三部分，日志文件存放在安装文件夹下的 ./equant/src/log 文件夹下，其中 equant.log 为系统日志文件，trade.dat 为交易日志文件，user.log 为用户日志文件，日志文件加下还有一个 his 文件夹存放的是转存的历史日志文件。

8）策略文件所在的位置
极智量化策略文件存放的位置在安装文件夹下的./equant/src/strategy 文件夹下。

9）回测报告文件所在的位置
极智量化策略回测报告文件存放的位置在安装文件夹下的./equant/src/reportdata 下。

10）如何自己增加 python 的第三方库？
极智量化支持丰富的 python 第三方库，若我们提供的 python 第三方库无法满足您的需求，您也可以通过 python 指令自行安装。
Step1：找到极智量化安装位置下的 Miniconda 文件夹并进入，安装位置为：
C:\Users****\AppData\Roaming\equant_pkg\Miniconda
其中****为当前的用户名。
Step2：组合键 win + R 打开运行窗口，输入 cmd 打开命令行窗口



Step3：在命令行窗口输入以下代码：
cd C:\Users****\AppData\Roaming\equant_pkg\Miniconda
Step4：输入以下命令即可安装 python 第三方库
.\python -m pip install xxxx
如遇下载过程很慢的情况，可以修改安装镜像源的位置，命令如下：
.\python -m pip install -i https://pypi.tuna.tsinghua.edu.cn/simple xxxx
其中 xxxx 为要安装的第三方库的名称。

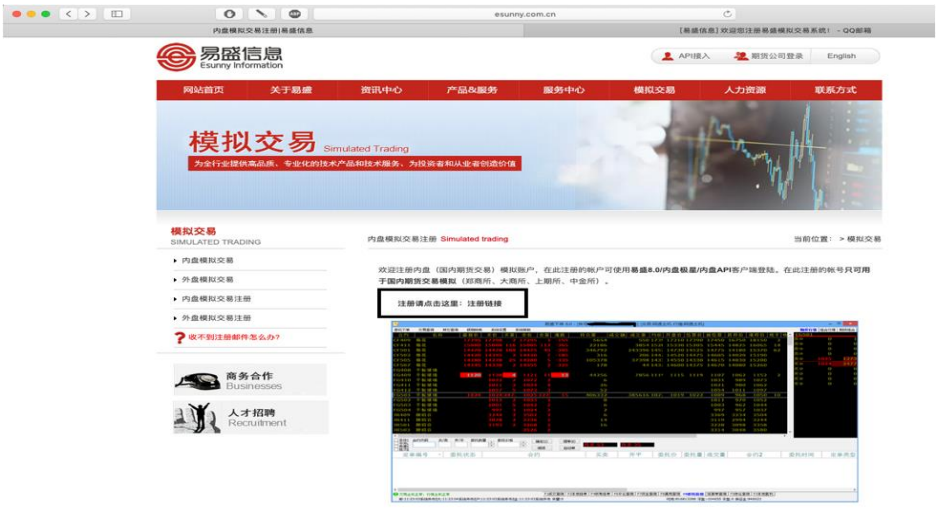
11）如何注册模拟交易账号?

模拟交易可以更好的帮助用户观察策略在实盘交易时的运行情况。模拟交易的前提是注册模拟账号，注册模拟账户的流程如下:

Step1：模拟账号注册请点击这里

Step2：在打开的页面上选择内盘模拟交易注册或外盘模拟交易注册，如下图所示:

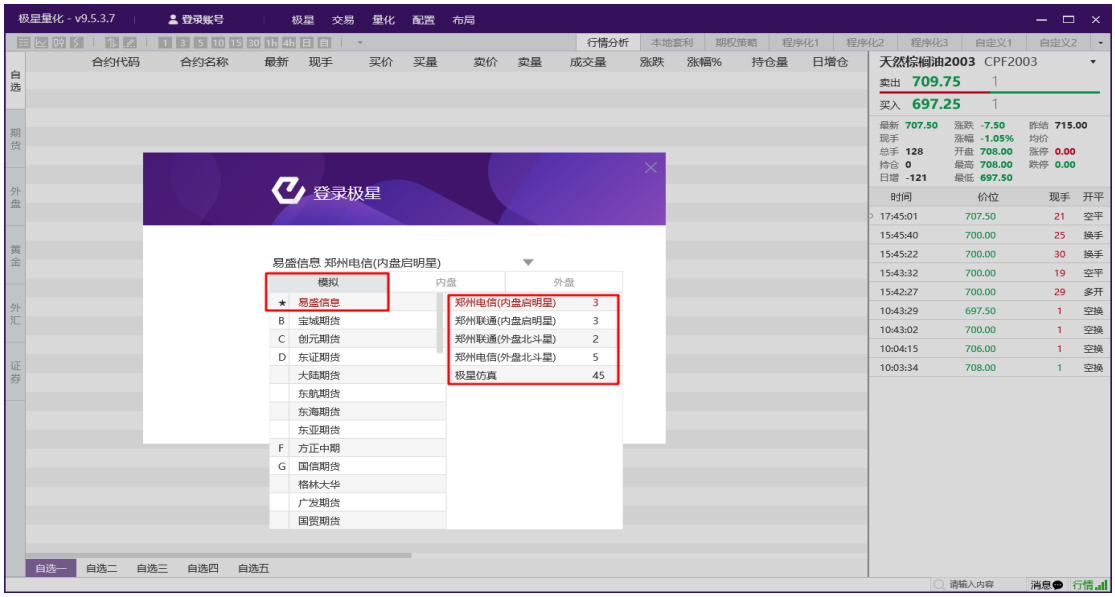
Step3：在界面上点击注册链接:



Step4：填入资料，等待账号和密码发到您的 qq 邮箱。

Step5：收到邮件，点击右键中的链接进行激活。

注意：一天后才可以登录极星 9.5，如果注册的是内盘账号，登录时请选择内盘启明星，如果注册的是外盘账号，请选择外盘北斗星：



示例策略

基本使用

1. 利用日志函数输出用户日志

```
##############################################################
#日志函数包括 LogDebug、LogInfo、LogWarn、LogError 这四个函数
#用户可以利用这些日志函数在极智量化主界面上的运行日志---用户
#日志处显示想要输出的信息
#日志函数有助于用户更好的了解和分析策略的运行情况
#该策略演示如何使用日志函数
##############################################################
contractId = "NYMEX|Z|CL|MAIN"       # 将合约代码定义为变量，修改合约编号时只需修改该变量值

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式为 K 线触发
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 利用日志函数输出用户日志
    LogInfo("handle_data 被调用")
```

2. 触发方式判断

```
##############################################################
#极智量化支持 1：即时行情触发 2：交易数据触发 3：每隔固定时间触发
#4：指定时刻触发 5：K 线触发，五种触发方式触发策略
#策略会根据用户设置的触发方式调用策略的 handle_data()函数
#该策略订阅了以上五种触发方式，展示不同的触发方式触发策略
##############################################################
import talib

contractId = "NYMEX|Z|CL|MAIN"

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
```

```
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式
    SetTriggerType(1)    # 即时行情触发
    SetTriggerType(2)    # 交易数据触发
    SetTriggerType(3, 1000)    # 每隔固定时间触发：时间间隔为 1000 毫秒
    SetTriggerType(4, ['223000', '223030'])    # 指定两个时刻触发
    SetTriggerType(5)    # K 线触发

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 判断策略是由哪种触发方式触发
    if context.triggerType() == "T":   # 定时触发
        LogInfo("定时触发")
    elif context.triggerType() == "C":   # 周期触发
        LogInfo("周期触发")
    elif context.triggerType() == "H":   # 回测阶段 K 线触发
        LogInfo("回测阶段 K 线触发")
    elif context.triggerType() == "S":   # 即时行情触发
        LogInfo("即时行情触发")
    elif context.triggerType() == "O":   # 委托状态变化触发
        LogInfo("委托状态变化触发")
```

3. 在主图上画指标线

```
#########################################################
#极智量化提供了多种绘图函数用于绘制指标、符号、线、柱子
#文字等，用户可以调用这些绘图函数绘制图形，
#可以通过这些绘图函数的参数决定绘制的图形被绘制到主图还
#是附图上，该策略演示了利用绘图函数在主图上绘制指标线
#########################################################
import talib

contractId = "NYMEX|Z|CL|MAIN"      # 将合约代码定义为变量，修改合约编号时只需修改该变量值
period = 5          # 指标计算周期

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式为 K 线触发
    SetTriggerType(5)



# 策略触发事件每次触发时都会执行该函数
```

```
def handle_data(context):
    # 先判断订阅合约的收盘价长度是否小于 period，小于的话则不满足计算长度，跳出这次触发
    if len(Close()) < period:
        return
    # 计算收盘价的五周期的移动平均值
    ma5 = talib.MA(Close(), period)
    # 在主图上画 ma5 指标线，指标名称为 ma5
    PlotNumeric("ma5", ma5[-1], main=True)
```

4. 在附图上画指标线

```
############################################################
#极智量化提供了多种绘图函数用于绘制指标、符号、线、柱子
#文字等，用户可以调用这些绘图函数绘制图形，
#可以通过这些绘图函数的参数决定绘制的图形被绘制到主图还
#是附图上，该策略演示了利用绘图函数在附图上绘制指标线
############################################################
import talib

contractId = "NYMEX|Z|CL|MAIN"      # 将合约代码定义为变量，修改合约编号时只需修改该变量值
period = 5          # 指标计算周期

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式为 K 线触发
    SetTriggerType(5)


# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 先判断订阅合约的收盘价长度是否小于 period，小于的话则不满足计算长度，跳出这次触发
    if len(Close()) < period:
        return
    # 计算收盘价的五周期的移动平均值
    ma5 = talib.MA(Close(), period)
    # 在主图上画 ma5 指标线，指标名称为 ma5
    PlotNumeric("ma5", ma5[-1], main=False)
```

5. 策略的上下文环境

```
#############################################################
#极智量化为用户提供了四个入口函数用于编写策略：
#initialize()、handle_data()、hisover_callback()、exit_callback()，
#每个入口函数都包含一个 context 参数用于带入策略的上下文信息，
#上下文信息包括：strategyStatus()当前策略状态、triggerType()当前
#触发类型、contractNo()当前触发合约、kLineType 当前触发的 K 线
#类型、kLineSlice()当前 K 线触发的 K 线周期、tradeDate()当前触发
#的交易日、dateTimeStamp()当前触发的时间戳、triggerData()当前
#触发类型对应的数据
#该策略展示如何使用这些方法查看策略当前运行的上下文信息
#############################################################
contractId = "NYMEX|Z|CL|MAIN"       # 将合约代码定义为变量，修改合约编号时只需修改
该变量值

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式
    SetTriggerType(1)    # 即时行情触发
    SetTriggerType(2)    # 交易数据触发
    SetTriggerType(4, ['090500'])  # 定时触发：9:05 触发策略
    SetTriggerType(5)    # K 线触发

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 判断当前的触发方式
    if context.triggerType() == "S":       # 即时行情触发
        LogInfo("触发方式为即时行情触发")
        # 打印 K 线的当前状态：'H' 表示回测阶段; 'C' 表示实时数据阶段
        # 'O': 委托状态变化触发； 'T': 定时触发
        LogInfo("策略当前状态: ", context.strategyStatus())
        # 打印当前策略的触发合约
        LogInfo("策略当前触发合约: ", context.contractNo())
        # 打印当前触发的 K 线类型，'T'：分笔， 'M'：分钟，'D'：日线
        LogInfo("策略当前触发的 K 线类型: ", context.kLineType())
        # 打印当前触发的 K 线周期
        LogInfo("策略当前触发的 K 线周期: ", context.kLineSlice())
        # 打印当前触发的交易日
        LogInfo("策略当前触发的交易日: ", context.tradeDate())
        # 打印当前触发的时间戳
        LogInfo("策略当前触发的时间戳: ", context.dateTimeStamp())
```

```
        # 打印当前触发类型对应的数据详情
        LogInfo("策略当前触发类型对应的数据: ", context.triggerData())

    elif context.triggerType() == "H":  # 历史阶段 K 线触发
        LogInfo("触发方式为历史阶段 K 线触发")
        # 打印 K 线的当前状态：'H' 表示回测阶段; 'C' 表示实时数据阶段
        # 'O': 委托状态变化触发； 'T': 定时触发
        LogInfo("策略当前状态: ", context.strategyStatus())
        # 打印当前策略的触发合约
        LogInfo("策略当前触发合约: ", context.contractNo())
        # 打印当前触发的 K 线类型，'T'：分笔， 'M'：分钟，'D'：日线
        LogInfo("策略当前触发的 K 线类型: ", context.kLineType())
        # 打印当前触发的 K 线周期
        LogInfo("策略当前触发的 K 线周期: ", context.kLineSlice())
        # 打印当前触发的交易日
        LogInfo("策略当前触发的交易日: ", context.tradeDate())
        # 打印当前触发的时间戳
        LogInfo("策略当前触发的时间戳: ", context.dateTimeStamp())
        # 打印当前触发类型对应的数据详情
        LogInfo("策略当前触发类型对应的数据: ", context.triggerData())

    elif context.triggerType() == "T":  # 定时触发
        LogInfo("触发方式为定时触发")
        # 打印 K 线的当前状态：'H' 表示回测阶段; 'C' 表示实时数据阶段
        # 'O': 委托状态变化触发； 'T': 定时触发
        LogInfo("策略当前状态: ", context.strategyStatus())
        # 打印当前策略的触发合约
        LogInfo("策略当前触发合约: ", context.contractNo())
        # 打印当前触发的 K 线类型，'T'：分笔， 'M'：分钟，'D'：日线
        LogInfo("策略当前触发的 K 线类型: ", context.kLineType())
        # 打印当前触发的 K 线周期
        LogInfo("策略当前触发的 K 线周期: ", context.kLineSlice())
        # 打印当前触发的交易日
        LogInfo("策略当前触发的交易日: ", context.tradeDate())
        # 打印当前触发的时间戳
        LogInfo("策略当前触发的时间戳: ", context.dateTimeStamp())
        # 打印当前触发类型对应的数据详情
        LogInfo("策略当前触发类型对应的数据: ", context.triggerData())

    elif context.triggerType() == "T":  # 委托状态变化触发
        LogInfo("触发方式为委托状态变化触发")
        # 打印 K 线的当前状态：'H' 表示回测阶段; 'C' 表示实时数据阶段
        # 'O': 委托状态变化触发； 'T': 定时触发
        LogInfo("策略当前状态: ", context.strategyStatus())
```

```
    # 打印当前策略的触发合约
    LogInfo("策略当前触发合约: ", context.contractNo())
    # 打印当前触发的 K 线类型，'T'：分笔， 'M'：分钟，'D'：日线
    LogInfo("策略当前触发的 K 线类型: ", context.kLineType())
    # 打印当前触发的 K 线周期
    LogInfo("策略当前触发的 K 线周期: ", context.kLineSlice())
    # 打印当前触发的交易日
    LogInfo("策略当前触发的交易日: ", context.tradeDate())
    # 打印当前触发的时间戳
    LogInfo("策略当前触发的时间戳: ", context.dateTimeStamp())
    # 打印当前触发类型对应的数据详情
    LogInfo("策略当前触发类型对应的数据: ", context.triggerData())
```

6. 实盘运行

```
############################################################
#极智量化客户端支持策略实盘运行
#要让策略在实盘阶段运行并交易，需要设置策略实盘运行,可以通过运
#行设置界面设置，也可以在策略中调用 SetActual(),除了设置实盘运
#行外，还需要在极星 9.5 客户端登录对应的内盘或外盘交易账户，
#并在运行设置界面选择想要进行交易的账户，或者通过 SetUserNo()
#在策略中设置要进行交易的账户，内盘合约要登录并设置对应的内盘交
#易账户，外盘合约要登录对应的外盘交易账户只有满足这两个条件策略
#才能进行实盘交易，该策略只是演示如何设置可以进行实盘交易
############################################################
import talib

contractId = "NYMEX|Z|CL|MAIN"

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据，不执行历史 K 线
    SetBarInterval(contractId, "M", 1, 'N')
    # 设置触发方式
    SetTriggerType(5)     # K 线触发
    SetActual()             # 设置实盘运行
    SetUserNo("Test")     # 设置实盘交易账户， 用户需要将"Test"修改为在极星 9.5 登录的
用户名

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    if context.strategyStatus() == "C":
        LogInfo("实盘运行")
```

7. 下单

```
##########################################################################
#极智量化有两种类型的下单函数可供用户进行下单操作
#Buy()、BuyToCover()、Sell()、SellShort()函数既可用在历史阶段
#进行策略回测，也可以用在实盘阶段，
#A_SendOrder()函数同样既可以在历史阶段又可以在实盘阶段使用，
#A 函数在实盘阶段使用时提供更多的参数供用户选择，提供更精细的下单控制
#这里演示如何应用两种类型下单函数进行下单操作，实盘阶段下单要设置
#实盘运行，设置方法见示例 5，该策略只是演示实盘交易，请不要将
#该策略在真实实盘交易时运行，否则会造成在实盘时连续开仓。
#以下是策略进行下单操作的例子，演示了如何利用 Buy 函数和
#A_SendOrder 函数下单
##########################################################################
import talib

contractId = "NYMEX|Z|CL|MAIN"

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅 contractId 的一分钟数据 500 根
    SetBarInterval(contractId, "M", 1, 500)
    # 设置触发方式
    SetTriggerType(5)    # K 线触发
    SetActual()          # 设置实盘运行
    SetUserNo("Test")    # 设置实盘交易账户，用户需要将"Test"修改为在极星 9.5 登录的
用户名

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 历史阶段
    if context.strategyStatus() == 'H':
        Buy(1, Close()[-1])    # Buy 函数以收盘价买入一手基准合约
        LogInfo("历史阶段买入一手合约")
    # 实盘阶段
    elif context.strategyStatus() == 'C':
        # 实盘阶段以最新价买入一手合约，retCode 为 0 代表订单发送成功
        retCode, retMsg = A_SendOrder(Enum_Buy(), Enum_Entry(), 1, Q_Last(),
contractNo=contractId)
        if retCode == 0:
            LogInfo("实盘阶段发送订单成功")
        else:
            LogInfo("实盘阶段发送订单失败")
```

8. 多合约触发

```
################################################
#极智量化策略支持多周期多合约数据同时触发，用户
#可以在策略中订阅不同合约不同周期的数据，
#该策略演示如何订阅多合约数据用于触发策略
################################################
import talib

contractId1 = "ZCE|Z|AP|MAIN"        # 将合约代码定义为变量，修改合约编号时只需修改该变量值
contractId2 = "ZCE|Z|SR|MAIN"        # 将合约代码定义为变量，修改合约编号时只需修改该变量值

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅合约数据，调用两次 SetBarInterval 函数，代表订阅了两个合约的 K 线数据，也可以订阅任意多个
    # 先订阅的合约会被作为基准合约用于展示 K 线和下单信号
    SetBarInterval(contractId1, "M", 1, 5)      # 订阅 contractId1 合约历史数据
    SetBarInterval(contractId2, "M", 1, 5)      # 订阅 contractId2 合约历史数据
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 通过输出信息观察策略当前被哪个合约的数据触发
    LogInfo("策略当前被{%s}合约触发" % context.contractNo())
```

9. 读写 excel 文件

```
#########################################################
# 该策略展示了如何读写 excel 文件
# 读写 excel 文件需要安装 python 第三库: xlrd，xlwt，xlutils
# 安装第三方库的方法可以参考帮助文档中的方法
# 策略中读取的 excel 所在位置为"E:\main_cont.xlsx"，用户可以修
# 改该文件路径，设置成自己本地文件的路径
# main_cont.xlsx 文件中的文件在 sheet1 表中，文件格式为：
# 第一行为表头：交易所、类别、品种号、年月份
# 第二行为表头对应的内容：如 ZCE、F、AP、005
# 该策略订阅 main_cont.xlsx 中包含的合约的即时行情，并将即时行情
# 对应的合约号、最新价、成交量、买卖价、买卖量到 excel 表格中保存
# 到本地文件"E:\cont_sanp.xls"中
#########################################################
import talib
```

```
import xlrd
import xlwt
from xlutils.copy import copy

rd_ex_file = r"E:\main_cont.xlsx"          # 带读取的文件所在位置，用户可修改该路径
wt_ex_file = r"E:\cont_sanp.xls"           # 写入文件的文件位置，用户预先不需要创建

cont_sheet = 'Sheet1'
snap_sheet = "Sheet1"

#合约列表
g_contList = []

def initialize(context):
    global g_contList
    #设置基准合约
    SetBarInterval("NYMEX|F|CL|MAIN", 'M', 1, 1)
    #合约表格
    rd_work    = xlrd.open_workbook(rd_ex_file)
    rd_table = rd_work.sheet_by_name(cont_sheet)

    #从合约表格中读取合约进行配置
    for i in range(rd_table.nrows):
        #跳过表头
        if i == 0:continue
        rows = rd_table.row_values(i)
        #月份如果为浮点数，转换为字符
        if isinstance(rows[3], float):
            rows[3] = str(int(rows[3]))
        elif isinstance(rows[3], str):
            rows[3] = str(rows[3])
        #月份保持最少 3 位，郑商所品种存在 001，excel 表中是 1
        rows[3] = rows[3].zfill(3)
        #拼接为合约格式'ZCE|F|SR|909'
        cont = '|'.join(rows)
        g_contList.append(cont)
        #订阅即时行情
        SubQuote(cont)

    #行情表格
    wt_work    = xlwt.Workbook()
    wt_table = wt_work.add_sheet(snap_sheet)
    table_head = ['合约', '时间戳', '最新价', '买价', '卖价', '成交量', '买量', '卖量']
    for i in range(len(table_head)):
```

```
            wt_table.write(0, i, table_head[i])

        wt_work.save(wt_ex_file)

def handle_data(context):
        global g_contList

        #跳过回测阶段
        if context.triggerType() == 'H':
            return
        #追加写入 excel 表
        r_xls = xlrd.open_workbook(wt_ex_file) #  读取 excel 文件
        row = r_xls.sheets()[0].nrows #  获取已有的行数
        excel = copy(r_xls) #  将 xlrd 的对象转化为 xlwt 的对象
        wt_table = excel.get_sheet(0) #  获取要操作的 sheet

        #保存合约号、最新价、成交量、买卖价、买卖量到 excel 表格中
        for cont in g_contList:
            wt_table.write(row, 0, cont)
            wt_table.write(row, 1, Q_UpdateTime(cont))
            wt_table.write(row, 2, Q_Last(cont))
            wt_table.write(row, 3, Q_BidPrice(cont))
            wt_table.write(row, 4, Q_AskPrice(cont))
            wt_table.write(row, 5, Q_TotalVol(cont))
            wt_table.write(row, 6, Q_BidVol(cont))
            wt_table.write(row, 7, Q_AskVol(cont))
            row += 1
        excel.save(wt_ex_file)
```

10. 用户参数

```
############################################################
#  极智量化提供了定义用户参数的方法供用户使用，用户参数的定义
#需要满足特定的格式才能被程序正确解析, 用户参数的用法可以参考
#入门教程中对用户参数的介绍，  这里只是演示 g_params 的定义方法
#以及在策略中如何使用
############################################################
import talib

g_params['Lots'] = 1 #aaa
g_params['AAAA'] = 2
g_params['BNBB'] = 'CCC'    # BNBB 参数
g_params['dddddd'] = 8.2    # dddddd 参数
g_params['Lots'] = 1
```

```
g_params['Zhisun'] = 5
g_params['Zhiying'] = 200
g_params['软件开仓开关'] = True
g_params['移动止盈'] = True
g_params['启动点数'] = 50
g_params['最小回调点数'] = 10
g_params['回调比例'] = 5
g_params['最小获利点数'] = 3
g_params['保本开关'] = True
g_params['保本启动点数'] = 10
g_params['保本点数'] = 2
g_params['放量倍数'] = 3
g_params['反手次数限制'] = 3
g_params['见顶根数'] = 9
g_params['日内平仓'] = False

def initialize(context):
    SetBarInterval("DCE|Z|I|MAIN", 'M', 1, 1)

def handle_data(context):
    LogInfo("用户参数为：  ",  g_params)
```

11. 盘中启动停止实盘交易

```
##########################################
# 盘中启动停止实盘交易策略
# 用到的函数接口是 StartTrade 和 StopTrade
# StopTrade 可以暂时停止策略实盘运行
# startTrade 可以重启暂停的实盘交易
# 本策略演示这两个函数的用法
##########################################
import talib

userNo = 'ET001'                  # 账户名
code = 'NYMEX|Z|CL|MAIN'      # 合约代码
handle_times = 1                  # 计次数
start_flag = False               # 启动停止交易标志位

def initialize(context):
    SetBarInterval(code, 'M', 1, 'N')   # 订阅合约，不执行历史 K 线
    SetUserNo(userNo)                        # 设置交易账户
    SetTriggerType(1)                       # 即时行情触发
    SetActual()                               # 实盘运行
```

```
def handle_data(context):
    global handle_times, start_flag
    LogInfo(handle_times)

    if context.triggerType() == 'H':
        return

    if handle_times % 100 == 0:
        if start_flag:
            # 盘中重新启动实盘交易
            StartTrade()
            start_flag = False
            LogInfo('StartTrade')
        else:
            # 盘中暂时停止实盘交易
            StopTrade()
            start_flag = True
            LogInfo('StopTrade')

    handle_times += 1            # 计次数累加
```

12. 属性函数简单使用

```
###############################################################################
# 该策略演示属性函数的简单使用
# 属性函数主要用于获取订阅的合约信息、交易所信息
# 通过属性函数可以获取到合约的合约编号、合约名称最小变动价位、每手乘数、单笔交易限量、
# 合约的朱联合约编号、交易时段个数、交易时段的起始、结束时间、交易所状态、品种交易状态、
# 当前时间是否处于交易时段等信息
# 属性函数包含的函数可以查看 API 函数中的属性函数部分
###############################################################################
contractId = "ZCE|Z|TA|MAIN"
kType = "M"                                  # K 线类型
kSlice = 1                                   # K 线周期
qty = 50                                     # K 线数量

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅合约
    SetBarInterval(contractId, kType, kSlice, qty)
    # 设置触发方式为 K 线触发
    SetTriggerType(5)
```

```
    # 设置发单方式为 K 线稳定后发单
    SetOrderWay(2)


# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    #以下函数参数不填的情况均默认使用订阅的基准合约信息：即第一次使用SetBarInterval()函数订阅的合约信息
    # 或者是在界面上选择的第一个合约的信息

    # 基准合约的 K 线周期：这里返回的是 kSlice 的值
    LogInfo("界面合约图表 K 线周期数值: ", BarInterval())
    # 基准合约 K 线周期类型："D"、"M"、"T"
    LogInfo("界面合约图表 K 线周期类型: ", BarType())
    LogInfo("买卖盘个数: ", BidAskSize())
    LogInfo("订阅合约的每手乘数: ", ContractUnit())
    LogInfo("合约对应交易所名称: ", ExchangeName(ExchangeName()))
    # 注意: ExchangeTime 返回的是本机的系统时间
    LogInfo("合约对应交易所时间: ", ExchangeTime(ExchangeName()))
    # ExchangeStatus 返回交易所的状态，返回值含义可查看该函数说明
    LogInfo("合约对应交易所状态: ", ExchangeStatus(ExchangeName()))
    # 返回品种交易状态，返回值含义可查看该函数说明
    LogInfo("品种或合约交易状态: ", CommodityStatus(contractId))

    LogInfo("交易时间段的个数: ", GetSessionCount())
    LogInfo("交易时间段的开始时间: ", GetSessionStartTime())
    LogInfo("交易时间段的结束时间: ", GetSessionEndTime())

    # 获取当前触发的时间戳，返回值为字符串
    dTimeStamp = context.dateTimeStamp()
    # 计算当前触发时间戳对应的交易日
    tradedate = CalcTradeDate(contractId, int(dTimeStamp))
    # calcTradeDate 函数当合约参数有误或者时间戳参数有误时返回值为-1 或者-2，详见函数说明
    if not (tradedate == -1 or tradedate == -2):
        LogInfo("指定合约指定交易日的指定交易时间段的开始时间戳: ", TradeSessionBeginTime(contractId, tradedate))
        LogInfo("指定合约指定交易日的指定交易时间段的结束时间戳: ", TradeSessionEndTime(contractId, tradedate, -1))
    LogInfo("指定合约指定时间点的下一个时间点及交易状态: ", GetNextTimeInfo(contractId, timeStr=Time()))

    LogInfo("K 线当前日期: ", CurrentDate())
    LogInfo("K 线当前时间: ", CurrentTime())
```

```
    # IsInSession 用于判断操作系统的当前时间是否为指定合约的交易时间
    LogInfo("当前时间是否为交易时间: ", IsInSession())

    # 获取合约信息
    LogInfo("合约默认保证金比率: ", MarginRatio())
    LogInfo("最大回溯 Bar 数: ", MaxBarsBack())
    LogInfo("单笔交易限量: ", MaxSingleTradeSize())
    LogInfo("合约最小变动价: ", PriceTick())
    LogInfo("合约价格精度: ", PriceScale())
    LogInfo("合约编号: ", Symbol())
    LogInfo("合约名称: ", SymbolName())
    LogInfo("合约所属的品种编号: ", SymbolType())
    LogInfo("商品主连/近月对应的合约: ", GetTrendContract())
```

13. 获取历史 K 线行情

```
#####################################################################
# 历史 K 线信息主要包括以下信息：
# 合约编号、时间戳、交易日、最高价、最低价、开盘价、收盘价、K 线索引、
# K 线周期、K 线类型、K 线成交量、K 线持仓量等信息
# 可以用 HisBarsInfo()函数获取历史 K 线的详细信息，HisData()、Open()、
# Close()、High()、Low()、Vol()、Date()、TradeDate()、
# CurrentBar()、Time()等函数提供了具体的获取某种历史数据信息的方法，
# 该策略演示如何利用上述函数获取 K 线信息
#####################################################################
import talib

contractId = 'NYMEX|Z|CL|MAIN'      # 合约代码
kType = "M"                          # K 线类型
kSlice = 1                           # K 线周期
qty = 50                             # K 线数量

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅合约
    SetBarInterval(contractId, kType, kSlice, qty)
    # 设置触发方式为 K 线触发
    SetTriggerType(5)
    # 设置发单方式为实时发单
    SetOrderWay(1)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 输出当前的 K 线索引，索引值从 1 开始
```

```
    LogInfo("当前 K 线索引: ", CurrentBar())
    # 输出当前 Bar 的状态值, 0 表示第一个 Bar,1 表示中间普通 Bar,2 表示最后一个 Bar
    LogInfo("当前 Bar 的状态值：", BarStatus())
    LogInfo("当前 Bar 的时间：  ", Time())
    LogInfo("当前 Bar 的日期：", Date())
    LogInfo("当前 Bar 的交易日: ", TradeDate())

    # 输出截止当前的 K 线的历史数据详情
    # HisBarsInfo 可以获取当前可以获取到的全部 K 线的历史信息数组
    # 索引-1 表示取最新的 K 线信息
    LogInfo("截止当前 K 线详细数据：", HisBarsInfo()[-1])
    # 输出当前可以获取到的收盘价数组
    LogInfo("截止当前 K 线收盘价数据: ", Close())
    LogInfo("截止当前 K 线开盘价数据: ", Open())
    LogInfo("截止当前 K 线最高价数据: ", High())
    LogInfo("截止当前 K 线最低价数据: ", Low())
    LogInfo("截止当前 K 线成交量数据: ", Vol())
    LogInfo("截止当前 K 线持仓量数据: ", OpenInt())
    # HisData 可以获取指定历史数据类型的数据数组，如最高价、最低价、开盘价、收盘价等
    # 这里使用 HisData 获取收盘价数组数据，获取长度最大为 100 个的标准价数组，
    # 数组长度不足 100 个时，取当前可以取到的某长度的数组，
    # Enum_data_Typical()使用说明可以参考枚举函数的用法
    LogInfo("截止当前 K 线标准价信息：  ", HisData(Enum_Data_Typical()))
```

14. 获取即时行情

```
#############################################################################
# 即时行情主要包括以下信息：
# 即时行情更新时间、最新卖价、最新买价、最新卖量、最新买量、最新价、
# 当日最低价、当日涨跌、昨结算、昨持仓等信息，
# Q_Update()、Q_AskPrice()、Q_BidPrice()、Q_AskVol()、Q_BidVol()、
# Q_Last()、Q_Low()、Q_PriceChg()、Q_PreSettlePrice()、Q_PreOpenInt()等函数
# 获取 API 信息的函数可以在 API 部分的即时行情函数接口查看。
# 该策略演示如何利用上述函数获取即时行情信息
#############################################################################
contractId = 'NYMEX|Z|CL|MAIN'      # 合约代码
kType = "M"                           # K 线类型
kSlice = 1                            # K 线周期
qty = 50                              # K 线数量

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅合约
```

```
    SetBarInterval(contractId, kType, kSlice, qty)
    # 订阅合约的 Tick 行情
    SetBarInterval(contractId, 'T', 0)
    # 设置触发方式为即时行情触发
    SetTriggerType(1)
    # 设置发单方式为实时发单
    SetOrderWay(1)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 即时行情触发时打印信息
    if context.triggerType() == "S":
        LogInfo("即时行情更新时间: ", Q_UpdateTime())
        LogInfo("最新卖价: ", Q_AskPrice())
        LogInfo("最新卖量: ", Q_UpdateTime())
        LogInfo("实时均价: ", Q_AvgPrice())

        LogInfo("最新买价: ", Q_BidPrice())
        LogInfo("最新买量: ", Q_BidVol())
        LogInfo("当日收盘价: ", Q_Close())
        LogInfo("当日最高价: ", Q_High())
        LogInfo("历史最高价: ", Q_HisHigh())
        LogInfo("历史最低价: ", Q_HisLow())
        LogInfo("最新价: ", Q_Last())
        # 使用 Q_LastTime()和 Q_LastDate()函数需要在设置界面或者在策略代码中添加了
        # SetBarInterval('NYMEX|Z|CL|MAIN', 'T', 0)为合约 NYMEX|Z|CL|MAIN 订阅了 Tick
行情！否则报错
        LogInfo("最新成交时间: ", Q_LastTime())
        LogInfo("最新成交日期: ", Q_LastDate())
        LogInfo("当日最低价: ", Q_Low())
        LogInfo("当日跌停板价: ", Q_LowLimit())
        LogInfo("当日开盘价: ", Q_Open())
        LogInfo("持仓量: ", Q_OpenInt())
        LogInfo("昨持仓量: ", Q_PreOpenInt())
        LogInfo("昨结算: ", Q_PreSettlePrice())
        LogInfo("当日涨跌: ", Q_PriceChg())
        LogInfo("当日涨跌幅: ", Q_PriceChgRadio())
        LogInfo("当日成交量: ", Q_TotalVol())
        LogInfo("当日成交额: ", Q_TurnOver())
        LogInfo("当日涨停板价: ", Q_UpperLimit())
        LogInfo("当日期权波动率: ", Q_Sigma())
```

15. 获取交易账户信息

```
##########################################################################
```

```
# 该策略演示账户函数即 A_开头的函数的简单使用
# 账户函数是用来获取交易账户的资金信息、持仓信息、交易信息的，
# 资金信息包括动态权益、可用资金、平仓盈亏、浮动盈亏、交易手续费、持仓保证金等，
# 持仓信息包括买入均价、买持仓量、卖出均价、卖持仓量、总持仓量、总持仓均价、
# 买仓可平数量、卖仓可平数量等
# 交易信息包括账户的订单列表、订单状态、订单委托数量、订单委托价格、订单成交价格、
# 订单成交数量、订单委托时间、订单对应的合约号等
# 因此要使用账户函数获取到指定账户的信息必须要保证该账户已经在极星 9.5 上登录
# 账户函数包含的函数可以查看 API 账户函数列表
##########################################################################
contractId = 'ZCE|Z|TA|MAIN'          # 合约代码
kType = "M"                             # K 线类型
kSlice = 1                              # K 线周期
qty = 50                                # K 线数量
userNo = "Test"                  # 修改账户为用户自己客户端登录的账户

# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅合约
    SetBarInterval(contractId, kType, kSlice, qty)
    # 设置触发方式为即时行情触发
    SetTriggerType(1)
    # 设置发单方式为实时发单
    SetOrderWay(1)
    SetUserNo(userNo)   # 设置账户名称，用户根据自己登陆的账户修改为自己的账户名
    SetActual()             # 设置实盘运行

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 以下函数中的参数均使用的是默认参数，表示取基准合约和最后调用 SetUserNo 设置的账户的信息

    # A_AccountID 返回当前公式应用的交易账户 ID
    # 当通过 SetUserNo 设置多个交易账户时，该函数返回最后一次调用 SetUserNo 设置的账户名称
    LogInfo("交易账户 ID: ", A_AccountID())
    # A_AllAccountID()返回用户在极星 9.5 上登录的所有交易账户的 ID 号
    LogInfo("所有交易账户 ID: ", A_AllAccountID())
    # A_GetAllPositionSymbol()函数返回指定的交易账户的所有持仓合约
    LogInfo("账户所有持仓合约: ", A_GetAllPositionSymbol())

    # 输出指定交易账户的手续费、动态权益、可用资金、持仓保证金、浮动盈亏、平仓盈亏、冻结保证金
    LogInfo("交易帐户的手续费: ", A_Cost())
```

```
    LogInfo("帐户的动态权益: ", A_Assets())
    LogInfo("帐户的可用资金: ", A_Available())
    LogInfo("帐户的持仓保证金: ", A_Margin())
    LogInfo("帐户的浮动盈亏: ", A_ProfitLoss())
    LogInfo("账户的平仓盈亏: ", A_CoverProfit())
    LogInfo("帐户的冻结资金: ", A_TotalFreeze())

    # 输出指定交易账户的当前商品的买入持仓量、买入持仓均价
    LogInfo("帐户下当前商品的买入持仓均价: ", A_BuyAvgPrice())
    LogInfo("帐户下当前商品的买入持仓量: ", A_BuyPosition())
    # 输出指定交易账户的当前商品的买仓可平数量、买入持仓盈亏
    LogInfo("帐户下买仓可平数量: ", A_BuyPositionCanCover())
    LogInfo("帐户下当前商品的买入持仓盈亏: ", A_BuyProfitLoss())
    # 输出指定交易账户的当前商品的卖出持仓量、卖出持仓均价
    LogInfo("帐户下当前商品的卖出持仓均价: ", A_SellAvgPrice())
    LogInfo("帐户下当前商品的卖出持仓: ", A_SellPosition())
    # 输出指定交易账户的当前商品的卖仓可平数量、卖出持仓盈亏
    LogInfo("帐户下卖仓可平数量: ", A_SellPositionCanCover())
    LogInfo("帐户下当前商品的卖出持仓盈亏: ", A_SellProfitLoss())

    # 输出当前账户指定商品的持仓均价、总持仓量、总持仓盈亏、当日买入持仓量、当日卖出持仓量
    LogInfo("帐户下当前商品的持仓均价: ", A_TotalAvgPrice())
    LogInfo("帐户下当前商品的总持仓量: ", A_TotalPosition())
    LogInfo("帐户下当前商品的总持仓盈亏: ", A_TotalProfitLoss())
    LogInfo("帐户下当前商品的当日买入持仓量: ", A_TodayBuyPosition())
    LogInfo("帐户下当前商品的当日卖出持仓量: ", A_TodaySellPosition())


    # 获取当前账户下所有订单号的列表
    orderList = A_AllOrderNo()
    # 遍历当前交易账户的所有订单号
    for orderNo in orderList:
        # 输出某一订单号的信息
        LogInfo("------------------------------------------------------------")
        LogInfo("该订单号为：  ", orderNo)
        # A_OrderBuyOrSell()函数返回订单买卖类型，返回值含义如下：
        # 'B'：买入，'S'：卖出，'A'：双边
        LogInfo("帐户下当前商品的某个委托单的买卖类型: ", A_OrderBuyOrSell(orderNo))
        # A_OrderEntryOrExit 函数返回订单开平仓状态，返回值含义如下
        # 'N'：无, 'O': 开仓, 'C'：平仓, 'T'：平今
        # '1'：开平，应价时有效，本地套利也可以
        # '2'：平开，应价时有效，本地套利也可以
        LogInfo("帐户下当前商品的某个委托单的开平仓状态：",
```

```
A_OrderEntryOrExit(orderNo))

        LogInfo("帐户下当前商品的某个委托单的成交数量: ", A_OrderFilledLot(orderNo))
        LogInfo("帐户下当前商品的某个委托单的成交价格: ", A_OrderFilledPrice(orderNo))
        LogInfo("帐户下当前商品的某个委托单的委托数量: ", A_OrderLot(orderNo))
        LogInfo("帐户下当前商品的某个委托单的委托价格: ", A_OrderPrice(orderNo))
        # A_OrderStatus()函数返回委托单的状态，返回值含义可查看函数说明
        LogInfo("帐户下当前商品的某个委托单的状态: ", A_OrderStatus(orderNo))
        LogInfo("委托单是否完结: ", A_OrderIsClose(orderNo))
        # 输出订单的委托时间
        LogInfo("帐户下当前商品的某个委托单的委托时间: ", A_OrderTime(orderNo))
        # 订单所对应的下单合约
        LogInfo("订单的合约号： ", A_OrderContractNo())
        # 输出 orderNo 对应的订单的下一个订单号
        LogInfo("当前账户下一个订单号： ", A_NextOrderNo(orderNo))
        # 输出 orderNo 对应的订单的下一个订单号
        LogInfo("当前账户的下一个排队(可撤)订单号： ", A_NextQueueOrderNo(orderNo))

    LogInfo("当前账户的第一个订单： ", A_FirstOrderNo())
    LogInfo("当前账户的最后（最新）一个订单号： ", A_LastOrderNo())
    LogInfo("当前账户的第一个排队(可撤)订单号： ", A_FirstQueueOrderNo())
    LogInfo("当前账户所有排队(可撤)订单号： ", A_AllQueueOrderNo())
    LogInfo("当前账户最新一笔完全成交委托单对应的时间： ", A_LatestFilledTime())
    LogInfo("当前账户所有订单号的列表： ", A_AllOrderNo())
```

16.策略回测统计信息

```
######################################################################
# 该策略演示策略回测的交易统计信息
# 包括持仓的统计信息和虚拟账户的回测统计信息
# 持仓信息主要包括建仓位置、建仓价格、建仓时间和仓位信息等，对应的 API
# 函数可以在“策略状态”部分查看，
# 虚拟账户的回测统计信息包括资金信息和交易信息两部分：
# 动态权益、可用资金、浮动盈亏、总盈利、总亏损、盈利次数、亏损次数、
# 开仓次数等简要的信息，其中资金信息的初始资金是由用户自己设置，
# 对应的 API 函数可以在“策略性能”部分查看
######################################################################
import talib

p1 = 5                  # 短周期
p2 = 20                 # 长周期
qty = 1                 # 下单手数
contractId = "ZCE|Z|TA|MAIN"     # 合约代码
kType = "M"         # K 线类型
```

```
kSlice = 1          # K 线周期

def initialize(context):
    SetBarInterval(contractId, kType, kSlice, 2000)   # 订阅合约
    SetTriggerType(5)           # 设置 K 线触发
    SetOrderWay(1)                # 设置发单方式为：实时发单

# 历史回测执行逻辑
def his_trigger(ma1, ma2):
    if ma1[-1] > ma2[-1] and MarketPosition() <= 0:
        Buy(qty, Close()[-1])
    elif ma1[-1] < ma2[-1] and MarketPosition() >= 0:
        SellShort(qty, Close()[-1])

def handle_data(context):
    if len(Close()) < p2:
        return

    ma1 = talib.MA(Close(), p1)     # 短周期指标计算
    ma2 = talib.MA(Close(), p2)     # 长周期指标计算

    # 执行历史阶段下单逻辑
    his_trigger(ma1, ma2)

    # 输出建仓信息
    LogInfo("当前持仓指定合约的平均建仓价格: ", AvgEntryPrice())
    LogInfo("当前持仓指定合约的第一个建仓位置到当前位置的 Bar 计数: ", BarsSinceEntry())
    LogInfo("当前持仓指定合约的最近平仓位置到当前位置的 Bar 计数: ", BarsSinceExit())
    LogInfo("当前持仓指定合约的最后一个建仓位置到当前位置的 Bar 计数: ",
BarsSinceLastEntry())
    LogInfo("当前持仓指定合约的最后一个 Buy 建仓位置到当前位置的 Bar 计数: ",
BarsSinceLastBuyEntry())
    LogInfo("当前持仓指定合约的最后一个 Sell 建仓位置到当前位置的 Bar 计数: ",
BarsSinceLastSellEntry())
    LogInfo("当前持仓指定合约的当天的第一根 Bar 到当前的 Bar 个数: ", BarsSinceToday())
    LogInfo("当前持仓指定合约的每手浮动盈亏: ", ContractProfit())

    LogInfo("策略当前的持仓合约数: ", CurrentContracts())
    LogInfo("当前持仓指定合约的买入方向的持仓量: ", BuyPosition())
    LogInfo("当前持仓指定合约的卖出方向的持仓量: ", SellPosition())
    LogInfo("当前持仓指定合约的第一个建仓位置的日期: ", EntryDate())
    LogInfo("当前持仓指定合约的第一次建仓的委托价格: ", EntryPrice())

    LogInfo("当前持仓指定合约的第一个建仓位置的时间: ", EntryTime())
```

```
    LogInfo("当前持仓指定合约的最近平仓位置 Bar 日期: ", ExitDate())
    LogInfo("当前持仓指定合约的最近一次平仓的委托价格: ", ExitPrice())
    LogInfo("当前持仓指定合约的最近平仓位置 Bar 时间: ", ExitTime())
    LogInfo("当前持仓指定合约的最后一个建仓位置的日期: ", LastEntryDate())

    LogInfo("当前持仓指定合约的最后一次建仓的委托价格: ", LastEntryPrice())

    LogInfo("当前持仓指定合约的 Buy 持仓的最后一次建仓的委托价格: ", LastBuyEntryPrice())
    LogInfo("当前持仓指定合约的 Sell 持仓的最后一次建仓的委托价格: ", LastSellEntryPrice())
    LogInfo("当前持仓指定合约的 Buy 持仓的最后一次建仓以来的最高价: ", HighestSinceLastBuyEntry())
    LogInfo("当前持仓指定合约的 Buy 持仓的最后一次建仓以来的最低价: ", LowestSinceLastBuyEntry())
    LogInfo("当前持仓指定合约的 Sell 持仓的最后一次建仓以来的最高价: ", HighestSinceLastSellEntry())
    LogInfo("当前持仓指定合约的 Sell 持仓的最后一次建仓以来的最低价: ", LowestSinceLastSellEntry())
    LogInfo("当前持仓指定合约的持仓的最后一个建仓位置的时间: ", LastEntryTime())
    LogInfo("当前持仓指定合约的持仓状态 : ", MarketPosition())
    LogInfo("当前持仓指定合约的浮动盈亏: ", PositionProfit())

    LogInfo("当前策略 Id: ", StrategyId())


    # 输出策略运行的统计信息
    LogInfo("策略当前可用虚拟资金: ", Available())
    LogInfo("策略当前的账户权益: ", CurrentEquity())
    LogInfo("策略指定合约的浮动盈亏: ", FloatProfit())
    LogInfo("策略当前累计总亏损: ", GrossLoss())
    LogInfo("策略当前累计总利润: ", GrossProfit())
    LogInfo("策略当前指定合约的持仓保证金: ", Margin())
    LogInfo("策略当前的平仓盈亏: ", NetProfit())
    LogInfo("策略当前的开仓次数: ", NumAllTimes())
    LogInfo("策略当前的盈利次数: ", NumWinTimes())

    LogInfo("策略当前的亏损次数: ", NumLoseTimes())
    LogInfo("策略当前的保本次数: ", NumEventTimes())
    LogInfo("策略当前的盈利成功率: ", PercentProfit())
    LogInfo("策略当前交易产生的手续费: ", TradeCost())
    LogInfo("策略当前交易总开仓手数: ", TotalTrades())
```

策略交易

1.双均线策略

```
######################################################################
# 双均线策略
# 历史阶段用了 buy、sell 进行下单测试
# 实盘阶段用了 A 函数进行更精细的下单控制
# 策略中定义了两个用户函数 his_trigger()和 tim_trigger()分别用于历史
# 和实盘阶段调用，策略根据短周期和长周期金叉死叉情况和持仓情况决定下单
# 逻辑
######################################################################
import talib
import time

p1 = 5                # 短周期
p2 = 20               # 长周期
qty = 1               # 下单手数
contractId = "ZCE|Z|TA|MAIN"     # 合约代码
kType = "M"         # K 线类型
kSlice = 1          # K 线周期

def initialize(context):
    SetBarInterval(contractId, kType, kSlice, 2000)   # 订阅合约
    SetTriggerType(1)           # 设置即时行情触发
    SetTriggerType(5)           # 设置 K 线触发
    SetOrderWay(1)                # 设置发单方式为：实时发单
    SetActual()                   # 设置实盘运行
    SetUserNo("Test")           # 账户名修改为用户 9.5 登录的账号，内盘交易设置内盘账
号，外盘交易设置外盘账号

# 历史回测执行逻辑
def his_trigger(ma1, ma2):
    if ma1[-1] > ma2[-1] and MarketPosition() <= 0:
        Buy(qty, Close()[-1])
    elif ma1[-1] < ma2[-1] and MarketPosition() >= 0:
        SellShort(qty, Close()[-1])

# 实盘阶段执行逻辑
def tim_trigger(ma1, ma2):
    if ma1[-1] > ma2[-1] and A_TotalPosition() <= 0:
        prc = min(Q_BidPrice() + PriceTick(), Q_UpperLimit())
        if A_TotalPosition() < 0:
            A_SendOrder(Enum_Buy(), Enum_ExitToday(), A_SellPosition(), prc)
```

```
            A_SendOrder(Enum_Buy(), Enum_Entry(), qty, prc)
        elif ma1[-1] < ma2[-1] and A_TotalPosition() >= 0:
            prc = max(Q_AskPrice() - PriceTick(), Q_LowLimit())
            if A_TotalPosition() > 0:
                A_SendOrder(Enum_Sell(), Enum_ExitToday(), A_BuyPosition(), prc)
            A_SendOrder(Enum_Sell(), Enum_Entry(), qty, prc)

def handle_data(context):
    if len(Close()) < p2:
        return

    # 历史、实盘阶段判断
    his = True if context.strategyStatus() == "H" else False

    ma1 = talib.MA(Close(), p1)     # 短周期指标计算
    ma2 = talib.MA(Close(), p2)     # 长周期指标计算

    if his:
        # 执行历史阶段下单逻辑
        his_trigger(ma1, ma2)
    else:
        # 执行实盘阶段下单逻辑
        tim_trigger(ma1, ma2)

    PlotNumeric("ma1", ma1[-1], 0xFF0000)        # 在主图绘制 ma1 指标
    PlotNumeric("ma2", ma2[-1], 0x00aa00)        # 在主图绘制 ma2 指标
    # 计算收益
    fit = NetProfit() + FloatProfit() - TradeCost() if his else A_CoverProfit() + A_ProfitLoss() -
A_Cost()
    PlotNumeric("fit", fit, 0x0000FF, False)     # 在附图绘制策略收益
```

2.乒乓策略

```
#########################################################
# 乒乓策略
# 下单数量可以由用户设置
# 买入、卖出时的价格偏移点数可以分别设置
# 可以设置最大下单次数
# 策略启动一秒钟后下出第一笔定单，第一笔定单为买单，其价格为行情价买加一跳
# 之后只要上笔定单完全成交就会在其成交价的基础上偏移指定价位发出一笔反向定单
# 如此循环，达到最大下单次数为止
#########################################################

import talib
```

```
g_params['qty'] = 1                  # 下单数量
g_params['buy_dot'] = 10             # 买入偏移点数
g_params['sell_dot'] = 10            # 卖出偏移点数
g_params['max_count'] = 12           # 最大下单次数

ord_id = 'init'
ord_prc = 0
count = 0

contractId = "ZCE|Z|TA|MAIN"     # 合约代码
kType = "M"          # K 线类型
kSlice = 1           # K 线周期

# 实盘下单
def send_ord(direct, price):
    global ord_id
    err, ord_id =    A_SendOrder(direct, Enum_Entry(), g_params['qty'], price)
    if err:
        ord_id = 'buy' if direct == Enum_Buy() else 'sell'
        LogError('下单失败：ord: %s, err:%-4d' % (ord_id, err))

########################################################
def initialize(context):
    SetBarInterval(contractId, kType, kSlice, 'N')
    # 交易触发
    SetTriggerType(2)
    # 固定 1 秒触发 1 次
    SetTriggerType(3, 1000)
    SetActual()

def handle_data(context):
    global ord_id, count, ord_prc
    # 达到下单次数，停止交易
    if count >= g_params['max_count']:
        StopTrade()
        return

    # 收到已排队时增加下单计数，收到完全成交时反向下单
    if context.triggerType() == 'O':
        if A_OrderStatus(ord_id) == '4':
            count += 1
            return
        elif A_OrderStatus(ord_id) != '6':
```

```
            return
        order = context.triggerData()
        dot = g_params['sell_dot'] if A_OrderBuyOrSell(ord_id) == Enum_Buy() else -
g_params['buy_dot']
        direct = Enum_Buy() if A_OrderBuyOrSell(ord_id) == Enum_Sell() else Enum_Sell()
        ord_prc = A_OrderFilledPrice(ord_id) + dot * PriceTick()
        send_ord(direct, ord_prc)
    # 初始状态或下单失败时
    elif context.triggerType() == 'C':
        # 不在交易时段则退出
        if not IsInSession():
            return
        if ord_id == 'init' and Q_BidPrice() > 0.0001:
            ord_prc = Q_BidPrice() + PriceTick()
            LogInfo("2: ", Enum_Buy(), ord_prc)
        elif ord_id in ['buy', 'sell']:
            direct = Enum_Sell() if ord_id == 'sell' else Enum_Buy()
            send_ord(direct, ord_prc)
```

3.海龟交易法则

```
##############################################################################
# 海龟交易法则
##############################################################################
import talib as ta
import numpy as np
#import math

RiskRatio = 1      # % Risk Per N ( 0 - 100)
ATRLength = 20     # 平均波动周期 ATR Length
boLength = 20      # 短周期 BreakOut Length
fsLength = 55      # 长周期 FailSafe Length
teLength = 10      # 离市周期 Trailing Exit Length
LPTF = True        # 使用入市过滤条件
PreEP = 0          # 前一次开仓的价格
AvgTR = 1          # 真实价格波动范围指数平均

SendOrderThisBar = False       # 当前 Bar 有过交易
PreBreakoutFailure = False     # 前一次突破是否失败

myEntryPrice = 0
myExitPrice = 0
DonchianHi = 0
DonchianLo = 0
```

```
fsDonchianHi = 0
fsDonchianLo = 0
ExitHighestPrice = 0
ExitLowestPrice = 0

ContractId = 'SHFE|Z|CU|MAIN'

# 求真实最高价，真实最低价，真实范围序列值
def TrueRange(barsinfo:list):
    if len(barsinfo) <= 0:
        return None, None, None

    ths = []
    tls = []
    trs = []
    hs = []
    ls = []

    for i, p in enumerate(barsinfo):
        high = barsinfo[i]["HighPrice"]
        low = barsinfo[i]["LowPrice"]
        hs.append(high)
        ls.append(low)

        if i == 0:
            th = high
            tl = low
        else:
            pclose = barsinfo[i-1]["LastPrice"]
            th = high if high >= pclose else pclose
            tl = low    if low    <= pclose else pclose

        tr = th - tl
        ths.append(th)
        tls.append(tl)
        trs.append(tr)

    return np.array(hs), np.array(ls), np.array(trs)

# 求指数平均
def XAverage(prices:np.array, length):
    fac = 2.0/(length+1)
    xa = 1
```

```
    if len(prices) < length:
        return xa

    for i, p in enumerate(prices):
        if i == 0:
            xa = p
        else:
            xa = xa + fac * (p - xa)

    return xa

def initialize(context):
    global ContractId
    SetBarInterval(ContractId, 'M', 1, 2000)
    SetTriggerType(5)
    SetOrderWay(1)
    SetActual()

def handle_data(context):
    global RiskRatio, ATRLength, boLength
    global teLength, fsLength, LPTF
    global PreEP, AvgTR, SendOrderThisBar
    global PreBreakoutFailure, ContractId
    global myEntryPrice, myExitPrice
    global ExitHighestPrice, ExitLowestPrice
    global DonchianHi, DonchianLo
    global fsDonchianHi, fsDonchianLo

    if BarStatus() == 0:
        PreEP = np.nan
        PreBreakoutFailure = False

    MinPoint = PriceTick()

    barsinfo = HisBarsInfo()

    bslen = len(barsinfo)

    if bslen < fsLength:
        return

    op = barsinfo[-1]["OpeningPrice"]

    highs, lows, trs = TrueRange(barsinfo)
```

```
    high    = highs[-1]
    low     = lows[-1]
    phigh = high if len(highs) <= 1 else highs[-2]
    plow    = low    if len(lows)    <= 1 else lows[-2]

    N = AvgTR
    AvgTR = XAverage(trs, ATRLength)

    TotalEquity = Available() + Margin()
    TurtleUnits = (TotalEquity*RiskRatio/100)/(N*ContractUnit(ContractId))
    TurtleUnits = int(TurtleUnits)
    maxUnites = int(Available()/(high*ContractUnit(ContractId)))
    TurtleUnits = min(maxUnites , TurtleUnits)
    LogInfo("avl:%f, mar:%f, conu:%f, tu:%f, N:%f, mp:%f, cidx:%d, bslen:%d\n" \
        %(Available(), Margin(), ContractUnit(ContractId), TurtleUnits, N, MinPoint,
CurrentBar(), bslen))

    DonchianHi     = ta.MAX(highs[:-1], timeperiod=boLength)[-1]
    DonchianLo     = ta.MIN(lows[:-1],    timeperiod=boLength)[-1]
    fsDonchianHi = ta.MAX(highs[:-1], timeperiod=fsLength)[-1]
    fsDonchianLo = ta.MIN(lows[:-1],    timeperiod=fsLength)[-1]
    ExitHighestPrice = ta.MAX(highs[:-1], timeperiod=teLength)[-1]
    ExitLowestPrice = ta.MIN(lows[:-1],    timeperiod=teLength)[-1]

    LogInfo("mpos:%d, bpos:%d, spos:%d\n" %(MarketPosition(), BuyPosition(), SellPosition()))
    LogInfo("low:%f,high:%f,dh:%f,dl:%f,fdh:%f,fdl:%f,ehp:%f,elp:%f,pep:%f,n:%f\n" %(low, high,
DonchianHi, \
        DonchianLo, fsDonchianHi, fsDonchianLo, ExitHighestPrice, ExitLowestPrice,PreEP,N))

    # 当不使用过滤条件，或者使用过滤条件并且条件为 PreBreakoutFailure 为 True 进行后
续操作
    if MarketPosition() == 0 and ((not LPTF) or PreBreakoutFailure):
        # 突破开仓
        if high > DonchianHi and TurtleUnits >= 1:
            # 开仓价格取突破上轨+一个价位和最高价之间的较小值，这样能更接近真实
情况，并能尽量保证成交
            myEntryPrice = min(high, DonchianHi + MinPoint)
            # 大跳空的时候用开盘价代替
            myEntryPrice = max(op, myEntryPrice)
            PreEP = myEntryPrice
            Buy(TurtleUnits, myEntryPrice)
            SendOrderThisBar = True
            PreBreakoutFailure = False
```

```
        if low < DonchianLo and TurtleUnits >= 1:
            # 开仓价格取突破下轨-一个价位和最低价之间的较大值，这样能更接近真实情况，并能尽量保证成交
            myEntryPrice = max(low, DonchianLo - MinPoint)
            # 大跳空的时候用开盘价代替
            myEntryPrice = min(op, myEntryPrice)
            PreEP = myEntryPrice
            SellShort(TurtleUnits, myEntryPrice)
            SendOrderThisBar = True
            PreBreakoutFailure = False

    # 长周期突破开仓  Failsafe Breakout point
    if MarketPosition() == 0:
        if high > fsDonchianHi and TurtleUnits >= 1:
            # 开仓价格取突破上轨+一个价位和最高价之间的较小值，这样能更接近真实情况，并能尽量保证成交
            myEntryPrice = min(high, fsDonchianHi + MinPoint)
            # 大跳空的时候用开盘价代替
            myEntryPrice = max(op, myEntryPrice)
            PreEP = myEntryPrice
            Buy(TurtleUnits, myEntryPrice)
            SendOrderThisBar = True
            PreBreakoutFailure = False

        if low < fsDonchianLo and TurtleUnits >= 1:
            # 开仓价格取突破下轨-一个价位和最低价之间的较大值，这样能更接近真实情况，并能尽量保证成交
            myEntryPrice = max(low, fsDonchianLo - MinPoint)
            # 大跳空的时候用开盘价代替
            myEntryPrice = min(op, myEntryPrice)
            PreEP = myEntryPrice
            SellShort(TurtleUnits, myEntryPrice)
            SendOrderThisBar = True
            PreBreakoutFailure = False

    # 有多仓的情况
    if MarketPosition() == 1:
        if low < ExitLowestPrice:
            myExitPrice = max(low, ExitLowestPrice - MinPoint)
            myExitPrice = min(op, myExitPrice)
            # 全平多仓
            bpos = BuyPosition()
            if bpos > 0:
```

```
                Sell(bpos, myExitPrice)
        else:
            if PreEP != np.nan and TurtleUnits >= 1:
                if high >= PreEP + 0.5*N:
                    myEntryPrice = high
                    PreEP = myEntryPrice
                    Buy(TurtleUnits, myEntryPrice)
                    SendOrderThisBar = True

            # 止损指令，加仓 Bar 不止损
            if low <= (PreEP - 2*N) and SendOrderThisBar == False:
                myExitPrice = PreEP - 2*N
                # 全平多仓
                bpos = BuyPosition()
                if bpos > 0:
                    Sell(bpos, myExitPrice)
                PreBreakoutFailure = True
    # 有空仓的情况
    elif MarketPosition() == -1:
        # 求出持空仓时离市的条件比较值
        if high > ExitHighestPrice:
            myExitPrice = min(high, ExitHighestPrice + MinPoint)
            # 大跳空的时候用开盘价代替
            myExitPrice = max(op, myExitPrice)
            # 全平空仓
            spos = SellPosition()
            if spos > 0:
                BuyToCover(spos, myExitPrice)
        else:
            if PreEP != np.nan and TurtleUnits >= 1:
                if low <= PreEP - 0.5*N:
                    myEntryPrice = low
                    PreEP = myEntryPrice
                    SellShort(TurtleUnits, myEntryPrice)
                    SendOrderThisBar = True
            # 止损指令，加仓 Bar 不止损
            if high >= (PreEP + 2*N) and SendOrderThisBar == False:
                myExitPrice = PreEP + 2*N
                # 全平空仓
                spos = SellPosition()
                if spos > 0:
                    BuyToCover(spos, myExitPrice)
                PreBreakoutFailure = True
```

4.唐氏通道交易系统

```
########################################################
# 唐氏通道交易策略
########################################################
import talib
import numpy as np

# 全局变量定义
M = 5
N = 5
MaxPositionNum = 3     # 最大开仓数
StopPoint = 30             # 止损点
WinPoint = 100             # 止赢点
FloatStopStart = 50    # 浮动止损开始点
FloatStopPoint = 20    # 浮动止损点

HHV = np.array([])
LLV = np.array([])

ContractId = 'SHFE|Z|CU|MAIN'

def initialize(context):
    global ContractId
    SetBarInterval(ContractId, 'M', 1, 500)
    SetTriggerType(5)

def handle_data(context):
    global ContractId

    # 最少获取 10 根数据
    bars = HisBarsInfo()
    barLen = len(bars)
    if barLen < 10:
        return

    close    = bars[-1]["LastPrice"]
    pclose = bars[-2]["LastPrice"]
    high     = bars[-1]["HighPrice"]
    phigh    = bars[-2]["HighPrice"]
    low      = bars[-1]["LowPrice"]
    plow     = bars[-2]["LowPrice"]

    # 求 M 周期最高
```

```
    HHV = Highest(High().tolist(), M)
    # 求 N 周期最低
    LLV = Lowest(Low().tolist(), N)

    # 绘制指标线
    PlotNumeric("LAST_HHV", HHV[-2], RGB_Red())
    PlotNumeric("LAST_LLV", LLV[-2], RGB_Green())

    # 根据持仓情况下单
    if high > HHV[-2]:
        if (CurrentContracts() < MaxPositionNum) or (MarketPosition() < 0):
            Buy(1, high)
    elif low < LLV[-2]:
        if (abs(CurrentContracts()) < MaxPositionNum) or (MarketPosition() > 0):
            SellShort(1, low)

    # 止损
    SetStopPoint(StopPoint)
    # 止赢
    SetWinPoint(WinPoint)
    # 浮动止损
    SetFloatStopPoint(FloatStopStart, FloatStopPoint)
```

5.账户交易示例

```
################################
#账户交易示例
################################
import talib

# 定义全局变量
ShortPeriod = 5
LongPeriod = 20
buyPos = 0
sellPos = 0

OrderNum = 1
TakeProfitSet = 45
StopLossSet = 20

MyEnterPrice = 0
EnterOrderId = -1
ExitOrderId = -1
retEnter = -1
```

```
retExit = -1
ContractId = "SHFE|Z|CU|MAIN"
UserId = "ET001"


# 初始化函数
def initialize(context):
    # 使用全局变量
    global ContractId, UserId

    SetBarInterval(ContractId, 'M', 1, 100)
    SetTriggerType(5)
    SetActual()

# 策略触发执行函数
def handle_data(context):
    # 使用全局变量
    global EnterOrderId, ExitOrderId
    global ShortPeriod, LongPeriod
    global buyPos, sellPos
    global OrderNum, TakeProfitSet, StopLossSet
    global MyEnterPrice, ContractId, UserId
    global retEnter, retExit

    # 定义局部变量
    MovePoint = 10

    MinPoint = PriceTick()
    MyEnterPrice = A_TotalAvgPrice()

    ma1 = talib.MA(Close(), timeperiod=ShortPeriod)
    ma2 = talib.MA(Close(), timeperiod=LongPeriod)

    #记录指标
    PlotNumeric('MA1', ma1[-1], color=RGB_Red())
    PlotNumeric('MA2', ma2[-1], color=RGB_Green())

    #撤消未成交开仓单
    if retEnter == 0 and A_OrderStatus(EnterOrderId) != 6:
        #A_DeleteOrder(EnterOrderId)
        LogInfo("撤销未成交的开仓单：%s, 订单状态值：%d, 可用资
金：%f\n" %(EnterOrderId, A_DeleteOrder(EnterOrderId), A_FreeMargin()))
        retEnter = -1
```

```
    #撤消未成交平仓单
    if retExit == 0 and A_OrderStatus(ExitOrderId) != 6:
        #A_DeleteOrder(ExitOrderId)
        LogInfo("撤销未成交的平仓单：%s, 订单状态值：%d, 可用资
金：%f\n" %(ExitOrderId, A_DeleteOrder(ExitOrderId), A_FreeMargin()))
        retExit = -1


    #打印账户当前合约持仓信息
    LogInfo("Buy:%d, Sell:%d, Total:%d\n" %(A_BuyPosition(), A_SellPosition(),
A_TotalPosition()))


    #策略执行区
    #有多仓的情况
    if A_BuyPosition() > 0:
        StopProfit = MyEnterPrice + TakeProfitSet * MinPoint
        StopLoss = MyEnterPrice - StopLossSet * MinPoint
        #止盈
        if Q_Close() >= StopProfit:
            retExit, ExitOrderId = A_SendOrder(Enum_Sell(), Enum_Exit(), OrderNum,
Q_BidPrice() - MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
            if retExit == 0:
                LogInfo("定单号：%s, 合约：%s, 卖出平仓数量：%d, 价
格: %f\n" %(ExitOrderId, ContractId, OrderNum, Q_BidPrice() - MovePoint * MinPoint))
            else:
                LogInfo("卖出平仓  error: %s\n" % ExitOrderId)
        #止损
        elif Q_Close() <= StopLoss:
            retExit, ExitOrderId = A_SendOrder(Enum_Sell(), Enum_Exit(), OrderNum,
Q_BidPrice() - MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
            if retExit == 0:
                LogInfo("定单号：%s, 合约：%s, 卖出平仓数量：%d, 价
格: %f\n" %(ExitOrderId, ContractId, OrderNum, Q_BidPrice() - MovePoint * MinPoint))
            else:
                LogInfo("卖出平仓  error: %s\n" % ExitOrderId)
        else:
            LogInfo("卖出平仓不操作，总仓位:%d, 多头仓位:%d, 空头仓位:%d\n" %
(A_TotalPosition(), A_BuyPosition(), A_SellPosition()))
    #有空仓的情况
    elif A_SellPosition() > 0:
        StopProfit = MyEnterPrice - TakeProfitSet * MinPoint
```

```
            StopLoss = MyEnterPrice + StopLossSet * MinPoint
            #止盈
            if Q_Close() <= StopProfit:
                retExit, ExitOrderId = A_SendOrder(Enum_Buy(), Enum_Exit(), OrderNum,
Q_AskPrice() + MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
                if retExit == 0:
                    LogInfo("定单号：%s，合约：%s，买入平仓数量：%d，价
格: %f\n" %(ExitOrderId, ContractId, OrderNum, Q_AskPrice() + MovePoint * MinPoint))
                else:
                    LogInfo("买入平仓  error: %s\n" % ExitOrderId)
            #止损
            elif Q_Close() >= StopLoss:
                retExit, ExitOrderId = A_SendOrder(Enum_Buy(), Enum_Exit(), OrderNum,
Q_AskPrice() + MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
                if retExit == 0:
                    LogInfo("定单号：%s，合约：%s，买入平仓数量：%d，价
格: %f\n" %(ExitOrderId, ContractId, OrderNum, Q_AskPrice() - MovePoint * MinPoint))
                else:
                    LogInfo("买入平仓  error: %s\n" % ExitOrderId)
            else:
                LogInfo("买入平仓不操作，总仓位:%d，多头仓位:%d，空头仓位:%d\n" %
(A_TotalPosition(), A_BuyPosition(), A_SellPosition()))
        #没有持仓开仓
        elif A_TotalPosition() == 0:
            if ma1[-1] > ma2[-1]:
                retEnter, EnterOrderId = A_SendOrder(Enum_Buy(), Enum_Entry(), OrderNum,
Q_AskPrice() + MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
                if retEnter == 0:
                    LogInfo("定单号：%s，合约：%s，买入开仓数量：%d，价
格: %f\n" %(EnterOrderId, ContractId, OrderNum, Q_AskPrice() + MovePoint * MinPoint))
                else:
                    LogInfo("买入开仓  error: %s\n" % EnterOrderId)
            elif ma1[-1] < ma2[-1]:
                retEnter, EnterOrderId = A_SendOrder(Enum_Sell(), Enum_Entry(), OrderNum,
Q_BidPrice()  - MovePoint * MinPoint, ContractId, UserId, Enum_Order_Limit(), Enum_FOK(),
Enum_Speculate())
                if retEnter == 0:
                    LogInfo("定单号：%s，合约：%s，卖出开仓数量：%d，价
格: %f\n" %(EnterOrderId, ContractId, OrderNum, Q_BidPrice() - MovePoint * MinPoint))
                else:
                    LogInfo("卖出开仓  error: %s\n" % EnterOrderId)
```

```
    else:
        LogInfo("不操作，总仓位:%d，多头仓位:%d，空头仓位:%d\n" % (A_TotalPosition(),
A_BuyPosition(), A_SellPosition()))
```

6.止损止盈交易

```
##########################################################
#止损止盈交易
#与止损止盈有关的函数如下：
#SetStopPoint()、SetFloatStopPoint()、SetWinPoint()
#SetStopWinKtBlack，具体用法参见函数说明
#该策略演示如何使用止损止盈函数
##########################################################
import talib

g_params['lose'] = 3                #止损点
g_params['win'] = 10                #止赢点
g_params['float_start'] = 0        #浮动启动点
g_params['float_dot'] = 5          #浮动止损点
g_params['stop_switch'] = 3        #止损开关，1 止损,2 止盈,3 止损+止盈,4 浮动,5 止损+浮
动,6 止盈+浮动,7 全部
g_params['single'] = True          #边单持仓

userNo = 'ET001'
contNo = 'NYMEX|Z|CL|MAIN'
# contNo = 'SHFE|F|CU|1910'
cFlag = 'A'

#################################################################
buy = True
def single_test():
    global buy
    if MarketPosition() == 0:
        if buy:
            Buy(1, Close()[-1])
        else:
            SellShort(1, Close()[-1])
        buy = not buy

def double_test():
    if BuyPosition() == 0:
        Buy(1, Low()[-1], needCover = False)
    if SellPosition() == 0:
        SellShort(1, High()[-1], needCover = False)
```

```
def set_stop():
    if g_params['stop_switch'] & 1:
        SetStopPoint(g_params['lose'])
    if g_params['stop_switch'] & 2:
        SetWinPoint(g_params['win'])
    if g_params['stop_switch'] & 4:
        SetFloatStopPoint(g_params['float_start'], g_params['float_dot'])


######################################################################
def initialize(context):
    SetUserNo('ET001')
    SetBarInterval(contNo , 'M', 1, 1000)
    SetActual()
    LogInfo('********', g_params)
    set_stop()

def handle_data(context):
    LogInfo('========', g_params)
    set_stop()

    if g_params['single']:
        single_test()
    else:
        double_test()
```

7.MACD 策略

```
########################################
# MACD 策略
########################################
import talib

fast = 12 #  快周期
slow = 26 #  慢周期
back = 9    # dea 周期
qty = 1     #  下单量
macd_dx = 0.01 #macd 阀值

contractId = "ZCE|Z|TA|MAIN"     #  合约代码
kType = "M"         # K 线类型
kSlice = 1          # K 线周期
```

```
def initialize(context):
    SetBarInterval(contractId, KTpe(), kSlice, 2000)
    SetTriggerType(5)
    SetOrderWay(2)

def handle_data(context):
    # 等待数据就绪，否则计算果结为异常值
    if len(Close()) < slow + back - 1:
        return

    # 计算 MACD
    diff, dea, macd = talib.MACD(Close(), fast, slow, back)

    # 突破下单
    if MarketPosition() != 1 and macd[-1] > macd_dx:
        Buy(qty, Open()[-1])
    elif MarketPosition() != -1 and macd[-1] < -macd_dx:
        SellShort(qty, Open()[-1])

    # 绘制 MACD 曲线
    PlotStickLine('macd', 0, macd[-1], RGB_Red() if macd[-1] > 0 else RGB_Blue(), False, False)
    PlotNumeric('diff', diff[-1], RGB_Red(), False, False)
    PlotNumeric('dea', dea[-1], RGB_Blue(), False, False)
    # 绘制盈亏曲线
    PlotNumeric("profit", NetProfit() + FloatProfit() - TradeCost(), 0xcccccc, False, True)
```

8.套利的布林带策略

```
########################
# 套利的布林带策略
########################
import talib
import numpy as np

code1="ZCE|F|TA|005"
code2="ZCE|F|TA|009"
p1=20
dot=2
qty=1

bt = 'M'       #barType
bi = 1          #barInterval

def initialize(context):
```

```
    SetBarInterval(code1, bt, bi, 2000)
    SetBarInterval(code2, bt, bi, 2000)
    SetOrderWay(2)

spds = []
def handle_data(context):
    prc_lst1 = Close(code1, bt, bi)
    prc_lst2 = Close(code2, bt, bi)
    if len(prc_lst1) == 0 or len(prc_lst2) == 0:
        return

    # 生成价差序列
    global spds
    spd_c = prc_lst1[-1] - prc_lst2[-1]
    if len(prc_lst1) > len(spds):
        spds.append(spd_c)
    else:
        spds[-1] = spd_c

    if len(spds) < p1:
        return

    # 计算价差布林通道
    upp, mid, low = talib.BBANDS(np.array(spds), p1, 2, 2)

    # 突破追单
    if spd_c < upp[-1] and MarketPosition(code1) <= 0:
        Buy(qty, prc_lst1[-1], code1)
        SellShort(qty, prc_lst2[-1], code2)
    elif spd_c > low[-1] and MarketPosition(code1) >= 0:
        SellShort(qty, prc_lst1[-1], code1)
        Buy(qty, prc_lst2[-1], code2)

    # 绘制指标线
    PlotNumeric("prc", spd_c, 0x000000, False)
    PlotNumeric('upp', upp[-1], RGB_Red(), False)
    PlotNumeric('mid', mid[-1], RGB_Blue(), False)
    PlotNumeric('low', low[-1], RGB_Green(), False)
    PlotNumeric("fit", NetProfit() - TradeCost(), RGB_Purple(), False, True)
```

9.基于布林带策略的内外盘交易

```
##########################################################
# 基于布林带策略的内外盘套利
```

```
# 内外盘价差计算时需注意内外盘交易时段及不同币种间的汇率
# 下单时需要注意内外盘合约的交易单位及每手乘数
############################################################

import talib
import numpy as np

code1    = 'DCE|Z|A|MAIN'     # 内盘合约
code2    = 'CBOT|Z|S|MAIN'    # 外盘合约
old_ind = -1                  # K 线索引
spds = []                     # 价差队列

#参数定义
g_params['user1']    = 'Test'              # 内盘账户，用户需要将用户名修改为自己登录的用户名
g_params['user2']    = 'ET001'             # 外盘账户，用户需要将用户名修改为自己登录的用户名
g_params['rate']     = 7.08                # 美元汇率
g_params['qty']       = 1                  # 下单数量
g_params['period'] = 10                    # 计算周期

# 浮点数比较大小函数
def float_cmp(val1, val2):
    val = val1 - val2
    if val > 0.000001:
        return 1
    elif val < -0.000001:
        return -1
    else:
        return 0

# 判断合约 code 在时间点 tim 时是否处于开盘阶段
def is_trading(code, tim):
    count = GetSessionCount(code)
    for i in range(0, count):
        b = GetSessionStartTime(code, i)
        e = GetSessionEndTime(code, i)
        if e < b:
            if b <= tim <= 0.24 or 0 <= tim < e:
                return True
        elif b <= tim < e:
            return True
    return False
```

```
def initialize(context):
    # 订阅行情
    SetBarInterval(code1, 'M', 1, 3000)
    SetBarInterval(code2, 'M', 1, 3000)
    SetTriggerType(1)   # 即时行情触发
    SetTriggerType(5)   # K 线触发
    SetOrderWay(2)     # K 线稳定后发单
    SetActual()         # 实盘运行

def handle_data(context):
    # 不在交易时段则不运行
    if not is_trading(code1, Time(code1, 'M', 1)) or not is_trading(code2, Time(code2, 'M', 1)):
        return

    # 获得参数
    user1 = g_params['user1']
    user2 = g_params['user2']
    rate = g_params['rate']
    qty = g_params['qty']
    p = g_params['period']

    # 数据不足时结果为异常值, 这里先对获取到的
    count1 = len(Close(code1, 'M', 1))
    count2 = len(Close(code2, 'M', 1))
    if count1 == 0 or count2 == 0:
        return
    if not (count1 >= p or count2 >= p):
        return

    # 生成价差队列
    spd_c = Close(code1, 'M', 1)[-1] - Close(code2, 'M', 1)[-1] * rate

    global spds, old_ind
    if CurrentBar(code1, 'M', 1) > old_ind:
        spds.append(spd_c)
        old_ind = CurrentBar(code1, 'M', 1)
    else:
        spds[-1] = spd_c

    # 用布林函数计算布林带上中下轨
    upp, mid, low = talib.BBANDS(np.array(spds), p, 2, 2)

    # 布林带震荡区间下单
    # 取下单价，历史阶段取收盘价，实盘截断取行情挂单价
```

```
    is_his = BarStatus(code1, 'M', 1) != 2 or BarStatus(code2, 'M', 1) != 2
    bid1 = Close(code1, 'M', 1)[-1] if is_his else Q_BidPrice(code1)
    ask1 = Close(code1, 'M', 1)[-1] if is_his else Q_AskPrice(code1)
    bid2 = Close(code2, 'M', 1)[-1] if is_his else Q_BidPrice(code2)
    ask2 = Close(code2, 'M', 1)[-1] if is_his else Q_AskPrice(code2)
    # 高抛低收
    if MarketPosition(code1) != -1 and float_cmp(spds[-1], upp[-1]) > 0:
        SellShort(qty, bid1, code1, userNo=user1)
        Buy(qty, ask2, code2, userNo=user2)
        LogInfo('sell spd', bid1, ask2)
    elif MarketPosition(code1) != 1 and float_cmp(spds[-1], low[-1]) < 0:
        Buy(qty, ask1, code1, userNo=user1)
        SellShort(qty, bid2, code2, userNo=user2)
        LogInfo('buy  spd', ask1, bid2)

    # 绘制指标线
    PlotNumeric('upp', upp[-1], RGB_Red(), False)
    PlotNumeric('mid', mid[-1], RGB_Blue(), False)
    PlotNumeric('low', low[-1], RGB_Green(), False)
    PlotNumeric('spd', spds[-1], RGB_Purple(), False)
    PlotNumeric("fit", NetProfit() + TradeCost(), RGB_Brown(), False, True)
```

10.阶段突破交易

```
########################################################################
# 阶段突破策略:
# 记录早上 split_tim 时间点前的高低点
# split_tim 时间点之后突破就开仓，止损 stop_dot 点
# 当日市闭前 cover_min 分钟开始清仓，实盘清仓会自动追价，每秒一次，先撤后平
########################################################################
import math

split_tim = 0.10   # 分割点时间，统计分割点之前的高低点，分割点之后突破就开仓
stop_dot = 20      # 开仓后的止损点数
cover_min = 10     # 当日闭市前 10 分钟开始仓清
qty = 1            # 单笔开仓量

hi = 0             # 最高
lo = 100000000     # 最低
curr_date = -1     # 前当交易日

contractId = "ZCE|Z|AP|MAIN"
kType = "M"
kSlice = 1
```

```
def initialize(context):
    SetBarInterval(contractId, kType, kSlice, 1000)
    # K 线稳定后发单
    SetOrderWay(2)
    # 一秒钟执行一次的定时器
    SetTriggerType(3, 1000)
    # 设置止损
    SetStopPoint(stop_dot, 2, 1)

# 分钟数转小数时间
def min2float(val):
    return (math.ceil(val / 60) * 40 + val) * 0.0001

def handle_data(context):
    global hi, lo, curr_date
    #换日时初始化最高最低数值
    if curr_date != TradeDate():
        hi = 0
        lo = 100000000
        curr_date = TradeDate()

    # 获得当前时间及合约的开闭市时间
    mid_tim = split_tim
    tim = Time()
    count = GetSessionCount()
    if count <= 0:
        LogError('时间节点获取错误', context.contractNo())
        return
    op_tim = GetSessionStartTime(index = 0)
    cl_tim = GetSessionEndTime(index = count - 1)
    if op_tim > cl_tim:
        if tim <= cl_tim:
            tim += 0.24
        if mid_tim <= cl_tim:
            mid_tim += 0.24
        cl_tim += 0.24

    # 尾盘清仓
    if cl_tim - min2float(cover_min) <= tim <= cl_tim:
        b_qty = BuyPosition()
        s_qty = SellPosition()
        if b_qty > 0:
            Sell(b_qty, Close()[-1])
```

```
        if s_qty > 0:
            BuyToCover(s_qty, Close()[-1])

        # 实盘阶段，清仓追价，每 1 秒判断一次，先撤再平
        if context.triggerType() == 'C':
            DeleteAllOrders()
            b_qty = A_BuyPosition()
            s_qty = A_SellPosition()
            if b_qty > 0:
                A_SendOrder(Enum_Sell(), Enum_ExitToday(), b_qty, Q_BidPrice())
            if s_qty > 0:
                A_SendOrder(Enum_Buy(), Enum_ExitToday(), s_qty, Q_AskPrice())

    # 忽略其他时段的定时器消息
    elif context.triggerType() == 'C':
        return

    # 记录时间分割点之前的高低值，支持夜盘阶段
    elif op_tim <= tim < mid_tim:
        hi = max(hi, High()[-1])
        lo = min(lo, Low()[-1])

    # 时间分割点之后，突破下单，并设置止损
    elif hi != 0:
        if Close()[-1] > hi:
            Buy(qty, Open()[-1], context.contractNo(), False)
        elif Close()[-1] < lo:
            SellShort(qty, Open()[-1], context.contractNo(), False)

    if hi == 0:
        return
    # 绘制最高最低曲线
    PlotNumeric('hi', hi, RGB_Red())
    PlotNumeric('lo', lo, RGB_Blue())
    # 绘制盈亏曲线
    PlotNumeric("profit", NetProfit() + FloatProfit() - TradeCost(), 0xFF00FF, False)
```

11. 麦语言转换示例

```
################################################################################
# 文华麦语言转 EQuant 策略示例：
# 麦语言与 EQuant 的函数对照见 QQ 群 472789093 里的群文件：极星量化与麦语言对照.html
################################################################################
```

```
################################################################################
# 麦语言代码：
################################################################################
# //该策略为趋势跟踪交易策略，适用较大周期，如日线。
# //该模型仅用作模型开发案例，依此入市，风险自负。
# //////////////////////////////////////////////////////
# MOVAVGVAL:MA((HIGH+LOW+CLOSE)/3,AVGLENGTH);//三价均线
# TRUEHIGH1:=IF(HIGH>REF(C,1),HIGH,REF(C,1));
# TRUELOW1:=IF(LOW<=REF(C,1),LOW,REF(C,1));
# TRUERANGE1:=IF(ISLASTBAR,H-L,TRUEHIGH1-TRUELOW1);
# UPBAND:MOVAVGVAL+MA(TRUERANGE1,ATRLENGTH);
# DNBAND:MOVAVGVAL-MA(TRUERANGE1,ATRLENGTH);//通道上下轨
# LIQUIDPOINT:=MOVAVGVAL;//出场条件
# MOVAVGVAL>REF(MOVAVGVAL,1)&&C>UPBAND,BK;//三价均线向上，并且价格上破通道上轨，开多单
# C<LIQUIDPOINT,SP;//持有多单时，价格下破三价均线，平多单
# MOVAVGVAL<REF(MOVAVGVAL,1)&&C<DNBAND,SK;//三价均线向下，并且价格下破通道下轨，开空单
# C>LIQUIDPOINT,BP;//持有空单时，价格上破三价均线，平空单
# AUTOFILTER;


################################################################################
# EQuant 代码：
################################################################################
import talib
import numpy as np

# 策略参数
g_params['QTY'] = 1              #下单量
g_params['AVGLENGTH'] = 20
g_params['ATRLENGTH'] = 3

contractId = "ZCE|Z|AP|MAIN"
kType = "M"
kSlice = 1

AVGS = []
TRUERANGE1 = []

class lst():
    def update_lst(self, lst, val):
        if len(Close()) > len(lst):
```

```
            lst.append(val)
        else:
            lst[-1] = val

def initialize(context):
    SetBarInterval(contractId, kType, kSlice, 1000)
    # 触发方式设置为 K 线触发
    SetTriggerType(5)
    SetOrderWay(2)

def handle_data(context):
    if len(Close()) < 2:
        return
    QTY = g_params['QTY']
    AVGLENGTH = g_params['AVGLENGTH']
    ATRLENGTH = g_params['ATRLENGTH']

    ls = lst()
    global AVGS, TRUERANGE1
    ls.update_lst(AVGS, (High()[-1] + Low()[-1] + Close()[-1]) / 3)
    TRUEHIGH1       = High()[-1] if High()[-1] > Close()[-2] else Close()[-2]
    TRUELOW1        = Low()[-1] if Low()[-1] <= Close()[-2] else Close()[-2]
    ls.update_lst(TRUERANGE1, High()[-1] - Low()[-1] if BarStatus() == 2 else TRUEHIGH1 -
TRUELOW1)
    if len(AVGS) < AVGLENGTH + 1:
        return

    # 三价均线
    MOVAVGVAL       = talib.MA(np.array(AVGS), AVGLENGTH)
    # 通道上下轨
    ATR             = talib.MA(np.array(TRUERANGE1), ATRLENGTH)
    UPBAND          = MOVAVGVAL[-1] + ATR[-1]
    DNBAND          = MOVAVGVAL[-1] - ATR[-1]
    # 出场条件
    LIQUIDPOINT     = MOVAVGVAL[-1]

    # 三价均线向上，并且价格上破通道上轨，开多单
    if MOVAVGVAL[-1] > MOVAVGVAL[-2] and Close()[-1] > UPBAND:
        Buy(QTY, Close()[-1], needCover = False)
    # 持有多单时，价格下破三价均线，平多单
    if Close()[-1] < LIQUIDPOINT:
        Sell(QTY, Close()[-1])
    # 三价均线向下，并且价格下破通道下轨，开空单
    if MOVAVGVAL[-1] < MOVAVGVAL[-2] and Close()[-1] < DNBAND:
```

```
        SellShort(QTY, Close()[-1], needCover = False)
    # 持有空单时，价格上破三价均线，平空单
    if Close()[-1] > LIQUIDPOINT:
        BuyToCover(QTY, Close()[-1])

    # 绘制指标线
    PlotNumeric('MOVAVGVAL', MOVAVGVAL[-1], 0x0000ff)
    PlotNumeric('UPBAND', UPBAND, 0xff0000)
    PlotNumeric('DNBAND', DNBAND, 0x00ff00)
    # 盈亏曲线
    PlotNumeric('PROFIT', NetProfit() - TradeCost(), RGB_Purple(), False)
```

12. 套利合约交易演示 1

```
##############################################################################
##
# 此策略仅用于套利功能演示
# 套利合约的双 MA 策略 交易用 Buy、Sell 函数
# 取套利行情使用极星 SPD 套利合约，下单和交易数据用交易所套利合约
# 使用 Buy，SellShort 函数下的单
##############################################################################
##

import talib


# 策略参数字典
g_params['p1'] = 5              # 短周期
g_params['p2'] = 15             # 长周期
g_params['dot'] = 1             # 突破点位

code1 = 'ZCE|F|TA|005'
code2 = 'ZCE|F|TA|009'
code = 'ZCE|S|TA|005|009'
spd = 'SPD|s|TA|005|009'

##############################################################################
##
# 策略开始运行时执行该函数一次
def initialize(context):
    # 订阅策略逻辑触发行情
    SetBarInterval(spd, 'M', 1, 2000)
    # 必须订阅下单合约对应的行情，否则会下单失败
    SetBarInterval(code, 'M', 1, 2000)
```

```
# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 不是触发合约的行情直接退出
    if context.contractNo() != spd:
        return
    if CurrentBar() < g_params['p2']:
        return

    _Close = Close(spd, 'M', 1)
    ma1 = talib.MA(_Close, g_params['p1'])[-1]
    ma2 = talib.MA(_Close, g_params['p2'])[-1]
    dot = g_params['dot'] * PriceTick(code1)

    if ma1 - ma2 > dot and BuyPosition(code) == 0:
        Buy(1, _Close[-1], code)
    elif ma2 - ma1 > dot and SellPosition(code) == 0:
        SellShort(1, _Close[-1], code)


    PlotNumeric('ma1', ma1, RGB_Red())
    PlotNumeric('ma2', ma2, RGB_Blue())
    PlotNumeric("fit", NetProfit() - TradeCost(), RGB_Purple(), False)
```

13. 套利合约的交易演示 2

```
#########################################################################
###############
# 此策略仅用于套利功能演示
# 套利合约的双 MA 策略 交易用账户函数
# 取套利行情使用极星 SPD 套利合约，  下单使用交易所套利合约
# 使用 A_SendOrder 函数下的单
# 历史阶段套利持仓取交易所套利合约的虚拟持仓
# 实时阶段的套利持仓需要根据单腿持仓计算获得
#########################################################################
###############

import talib


# 策略参数字典
g_params['p1'] = 5              # 短周期
g_params['p2'] = 15             # 长周期
g_params['dot'] = 1             # 突破点位
```

```
code1 = 'ZCE|F|TA|005'
code2 = 'ZCE|F|TA|009'
code = 'ZCE|S|TA|005|009'
spd = 'SPD|s|TA|005|009'

############################################################################
###############
# 套利持仓获取
# 用 Buy、Sell 函数下单时，virtual 始终取 True
# 用 A 函数下套利单时， 历史阶段 virtual 取 True，实时阶段 virtual 取 False
############################################################################
###############
def spd_buy_postion(virtual):
    if virtual:
        return BuyPosition(code)
    else:
        b = A_BuyPosition(code1)
        s = A_SellPosition(code2)
        return min(b, s)

def spd_sell_postion(virtual):
    if virtual:
        return SellPosition(code)
    else:
        s = A_SellPosition(code1)
        b = A_BuyPosition(code2)
        return min(s, b)

############################################################################
###############
# 策略开始运行时执行该函数一次
def initialize(context):
    SetBarInterval(spd, 'M', 1, 2000)
    SetBarInterval(code, 'M', 1, 2000)
    SetOrderWay(2)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    # 不是触发合约的行情直接退出
    if context.contractNo() != spd:
        return
    if CurrentBar() < g_params['p2']:
        return
```

```
_Close = Close(spd, 'M', 1)
ma1 = talib.MA(_Close, g_params['p1'])[-1]
ma2 = talib.MA(_Close, g_params['p2'])[-1]
dot = g_params['dot'] * PriceTick(code1)

his = context.strategyStatus() == 'H'
if ma1 - ma2 > dot and spd_buy_postion(his) == 0:
    A_SendOrder(Enum_Buy(), Enum_Entry(), 1, _Close[-1], code)
    spos = spd_sell_postion(his)
    if spos > 0:
        A_SendOrder(Enum_Buy(), Enum_Exit(), spos, _Close[-1], code)
elif ma2 - ma1 > dot and spd_sell_postion(his) == 0:
    A_SendOrder(Enum_Sell(), Enum_Entry(), 1, _Close[-1], code)
    bpos = spd_buy_postion(his)
    if bpos > 0:
        A_SendOrder(Enum_Sell(), Enum_Exit(), bpos, _Close[-1], code)


PlotNumeric('ma1', ma1, RGB_Red())
PlotNumeric('ma2', ma2, RGB_Blue())
PlotNumeric("fit", NetProfit() - TradeCost(), RGB_Purple(), False)
```

参与开源项目

如何贡献代码

机构或个人可申请加入极智量化开源项目组，向极智量化项目推送优化、改进方案或代码。
如果您愿意加入我们的项目，请阅读以下文档，希望以下内容可以给您解答您的疑惑，给您带来帮助。
极智量化的开发工作在 equant 上进行，无论是团队成员还是个人贡献代码都要以同样的方式进行代码提交。

分支管理

master 分支为最新稳定版本，只有团队成员在发布新版本时才会将充分测试的 develop 分支合并到 master 分支上。
develop 分支为最新开发版本，提交代码需要保证所提交的代码通过测试。
如果是修复 bug, 需要额外创建 bug/xxx 分支来进行代码提交, 测试 通过后提交 pull request 并等待项目管理员的 merge check。
如果是增加 feature，需要额外创建 feature/xxx 分支来进行代码提交，并完善文档和测试脚本，测试通过后提交 pull request 并等待项目管理员的 merge check。

Bugs

如果您在使用的过程中发现了 Bug，请通过 https://github.com/fanliangde/equant/issues 来提交并描述相关的问题，您也可以在这里查看其它的 issue，通过解决这些 issue 来贡献代码。

Pull Request
如果您是第一次通过 Pull Request 来提交代码，您可以参考 How to Contribute to an Open Source Project on GitHub 来了解 Contribute Workflow。
我们会认真审核您的 Pull Request，并给出如下三种回应:

- Merge Pull Request：合并您的代码进入 develop 分支
- Pending：如果发现有一些地方还需要完善，我们会给出具体的完善建议，并等待您的进一步提交。
- Won't Merge：如果发现您的 Pull Request 不适合合并，我们会给出具体的解释，并关闭相应的 issue。

流程

在提交 Pull Request 前，请确保您是按照如下流程进行代码的开发和测试的:

1. Fork epolestar equant
2. 基于 develop 分支 创建新的分支，分支命名需要遵循 分支管理 中的命名规则。
3. 如果您修改了代码，请保证通过测试。
4. 如果您修改了 API，请保证文档也同时更新。
5. 如果您增加了新的功能，请保证增加测试代码。