极智量化帮助手册

目录

1.极智量化产品介绍

1.1 极智量化是什么

极智量化终端是基于 C 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。

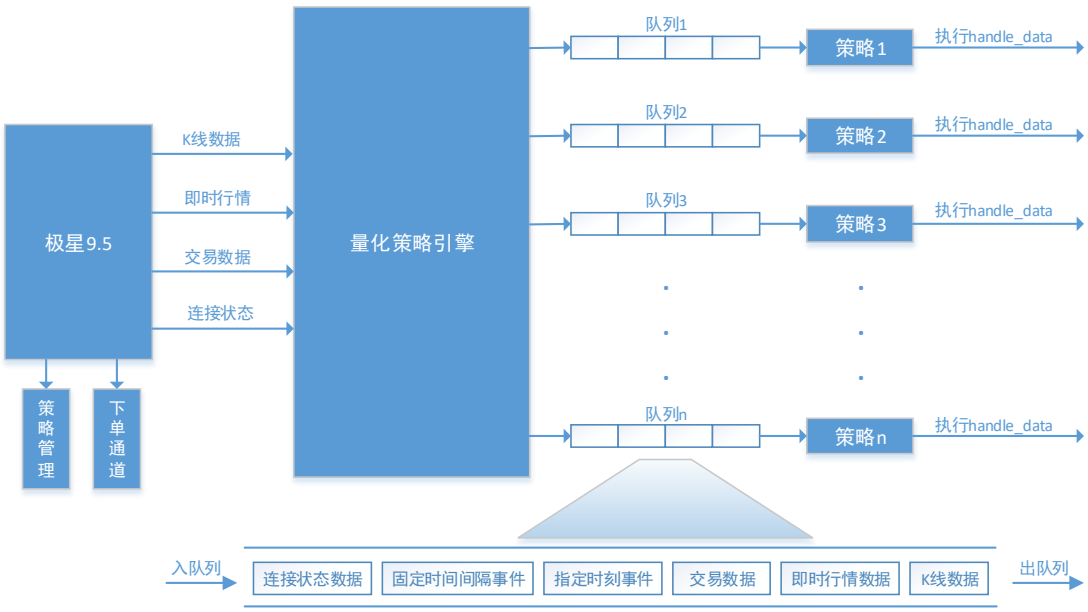
1.2 产品特点

- 策略引擎由 C 语言开发，保证策略高效运行；
- 用户策略接口为 python 接口，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，深度行情；
- 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更新，基准合约 K 线触发；
- 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 CTP、恒生、金仕达、启明星、北斗星等交易后台

1.3 编程风格

极智量化的用户策略支持的开发语言是 python。每个策略是一个单独的进程，每个策略资源独立，用户可以在策略中构建各种复杂结构的策略程序，同时保持高性能及高可读性。在运行多策略时可以充分利用多核、多 CPU 的计算能力，提高策略的运行速度。策略之间可以单独操作且不会对其他策略造成影响，可以随时对策略进行启动/停止/删除/修改操作。

1.4 系统架构



极智量化系统主要有两部分组成：极星 9. 5 和策略引擎。

极星 9.5：作为数据通道、下单通道和策略管理、策略指标展示界面，对策略的启动、停止、删除以及运行结果进行管理，并将行情数据、交易数据和服务器状态传递给策略引擎，以及作为量化端的下单通道。

量化策略引擎：负责接收极星 9.5 的数据并向各个策略分发。策略引擎通过队列和策略进行数据交互。策略引擎将每个策略所需数据分发到每个队列，每个策略从对应的队列中取出待触发事件运行策略。队列中的触发事件根据进入队列的顺序，以先入先出的原则对策略进行触发。关于触发事件的说明可参考触发方式。

2.安装

极智量化终端目前支持在 win7、win8、win8.1、win10 和 windows server 系列等操作系统上运行，在 win10 操作系统上运行是最稳定的。我们提供了一个在线安装包供用户下载和安装。该安装包会自动完成极智量化产品的安装并配置产品运行所需要的外部 python 环境，运行极智量化对系统的要求如下：

- Windows7 64 位及以上版本
- 屏幕分辨率最低支持 1024*768
- 内存 4G 及以上

安装步骤如下：

- 下载极智量化安装包，下载完成后双击 exe 程序进行安装；
- 安装过程持续 2 分钟左右，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，即可打开运行。后续如有新的更新可用，可在重启客户端后自动更新。

安装失败原因及解决办法：

1. 360 等杀毒软件的拦截会导致安装失败。需关闭杀毒软件后，再重新安装；
2. 网络原因。由于所处的网络 80 端口被禁用等原因导致无法访问网络，会出现安装失败的情况。用户需要检查自己的网络是否存在问题。

3.快速入门

本部分内容指导用户如何快速入门。首先对量化交易的基本概念进行了一个简单的介绍；然后对本产品的策略框架进行介绍，使客户对策略的结构有一个简单的介绍；并对界面的操作进行了展示；之后介绍了产品的 API 的使用方法以及 python 第三方库的安装使用，最后展示了如何编写一个简单的策略。

3.1 量化交易介绍

本部分对量化交易的概念进行介绍，使初学者对量化交易有一个初步的认识，并引导投资者改变传统的交易方式，争取从量化交易中获得超额的交易回报。

3.1.1 什么是量化交易

量化交易在智库百科上是这样定义的：指借助现代统计学和数学的方法，利用计算机技术来进行交易的证券投资方式。用户可以根据历史数据的规律，并结合自身的交易经验，总结出一个自己的策略。基于历史数据和实时数据，用户可以利用模型回测和模拟交易对策略进行不断的测试、验证，并根据回测报告对策略加以修正，以期获取可以持续的、稳定的且高于平均收益的超额回报。

量化交易起源于上世纪七十年代的股票市场，之后迅速发展和普及，尤其是在期货交易市场，程序化逐渐成为主流。有数据显示，国外成熟市场期货程序化交易已占据总交易量的 70%-80%，而国内则刚刚起步。手工交易中交易者的情绪波动等弊端越来越成为盈利的障碍，而程序化交易天然而成的精准性、100%执行率则为它的盈利带来了优势。

3.1.2 量化交易优势

1. 规避人的主观的情绪波动

量化交易有着严格的纪律性，这样做可以克服人性的弱点，如贪婪、恐惧、侥幸心理，也可以克服认知偏差。

2. 快速高效

量化交易投资是由计算机自动产生交易策略的一种投资方法，通过建立数学

模型来实现交易理念，并通过对模型不断优化、不断改进实现对市场交易机会的准确跟踪，提升交易胜率和盈利率。

3. 客观理性

量化交易投资具有完整的评价体系。模型建立后，通过对历史数据进行回测检验，确定模型在各个行情阶段均能有效运行，实现盈利；同时将模型加载至其他交易品种进行测试，确保模型具有较强的通用性。利用数学与统计学方法给与策略量化运行的结果一个客观理性的评价标准。

4. 妥善运用套利的思想

量化交易通过全面、系统性的扫描捕捉错误定价、错误估值带来的机会，从而发现估值洼地，并通过买入低估资产、卖出高估资产而获利。

5. 靠概率取胜

这表现为两个方面，一是定量投资不断的从历史中挖掘有望在未来重复的历史规律并且加以利用；二是在期货投资的实际过程中，运用概率分析，提高买卖成功的概率和仓位控制。

3.1.3 量化交易需要什么

1. 各种数据。要有能方便使用的各种投资相关的数据。这要考虑到各种数据的收集、存储、清洗、更新，以及数据取用时的边界、速度、稳定。

2. 一套量化交易的系统。要有能编写策略、执行策略、评测策略的系统。这要考虑到系统对各种策略编写的支持、系统进行回测与模拟的高仿真、系统执行策略的速度、系统评测策略的科学可靠性等。

3. 一定的编程基础。要有将自己的想法转化为用计算机编程语言表达出来的能力。

用户不需要为此担心，极智量化产品为期货投资者提供了全套的量化交易的工具和服务。用户需要做的只是需要学习如何使用该产品，以及学习如何将自己的交易思想编写成能够在我们产品上运行的策略即可。至于编程方便的能力，当然不是要求客户拥有像专业程序员那样的编程能力，用户只需掌握最基本的编程语法，即可实现大部分的交易策略。

3.2 量化策略框架

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。

3.2.1 策略编写的基本框架

在量化编辑器界面新建一个策略，我们会自动为用户创建一个空的量化策略：

```
import talib
```

```python
# 策略参数字典
g_params['p1'] = 20    # 参数示例


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


# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    pass


# 历史回测阶段结束时执行该函数一次
def hisover_callback(context):
    pass


# 策略退出前执行该函数一次
def exit_callback(context):
    pass
```

可以看到，该策略创建了四个函数体为空的函数，这四个函数作为约定函数，构成了策略的基本框架。约定函数作为策略的入口函数，用户必须实现对应的约定函数才可以正确的运行策略。

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。[界面配置信息可在这里修改，变量的初始化建议在这里完成]

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。[策略逻辑实现部分]

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]

需要说明的是除了 handle_data 函数外，其他三个约定函数在策略执行阶段最多执行一次。handle_data 作为策略逻辑实现的主要部分，K 线数据、即时行情

数据、交易数据、指定时刻事件、固定时间间隔事件、连接状态时间这些事件都会触发策略调用 handle_data 函数，每次触发 handle_data 执行的事件的数据可以通过 context 对象获取。触发方式的说明见 3.5.1 触发方式。

3.2.2 什么是回测

基于历史行情数据，用量化策略进行模拟交易，从而得到的收益以及净值变化情况。算法交易和其他投资门类差别很大，相比其他投资方式，在提供了足够丰富的数据后，算法交易可以根据历史数据，对未来收益有一个更好的估计。通过历史数据去估计未来收益，这样的过程称之为回测。

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虚拟撮合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。

3.2.3 如何解决程序错误

这部分内容在 7.6 节常见异常及处理方式部分进行了详细的总结，用户使用过程中遇到程序报错时，可参考这部分内容处理程序中遇到的错误。

3.3 量化部分操作说明

量化策略作为极智量化的重要功能已经整合进 9.5 客户端，现启动入口和行情分析、本地套利、期权策略等功能平级放置在极智量化的右上部分。更加方便快捷的使用量化交易，如下图示：

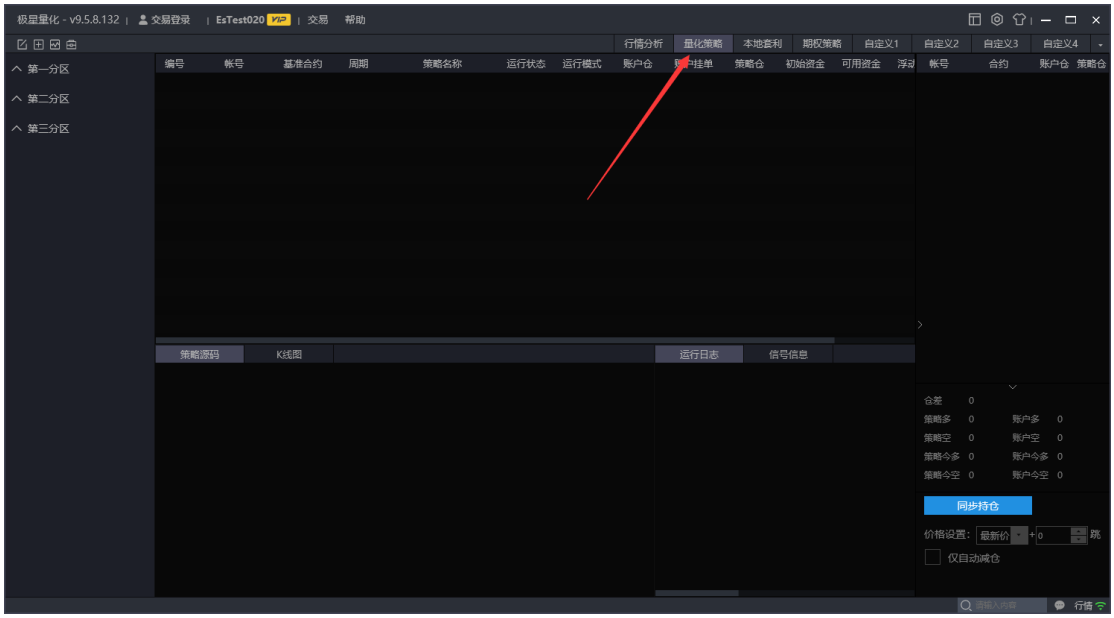
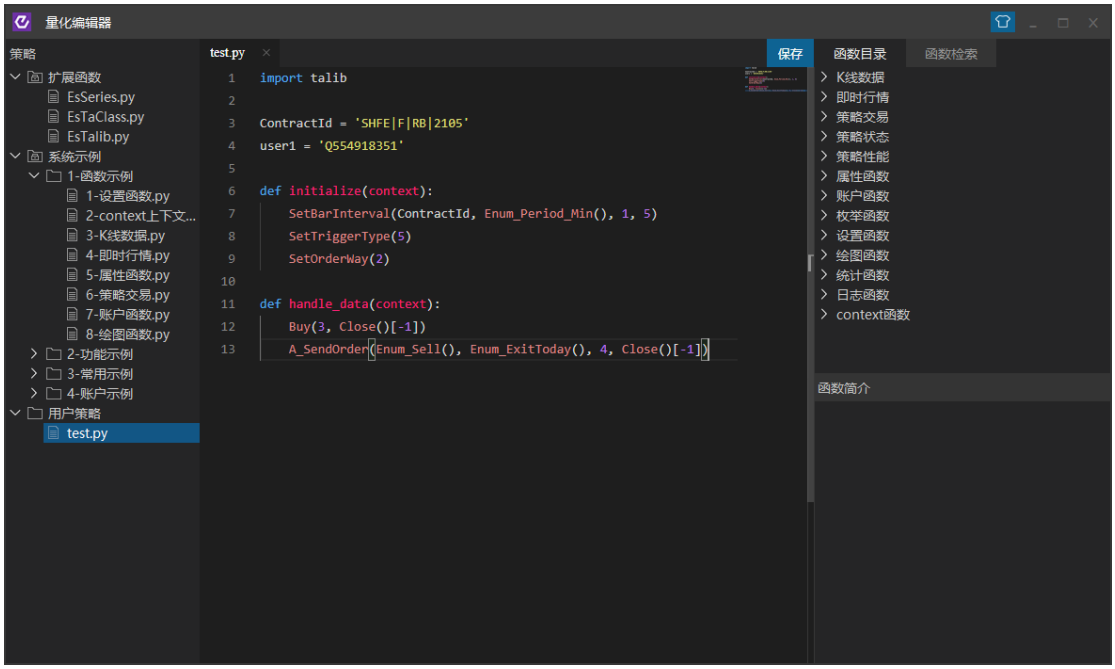


图 3.3.1

1. 编辑策略

图 3.3.2

在点击该图标后会启动策略编辑器，该编辑器作为策略编辑保存、函数说明查看使用，不再作为策略运行的入口。在界面左侧用户可以看到扩展策略、系统示例、用户策略三个分类，在系统示例中我们重新整理、分类、添加了一些示例策略，用户可以翻阅或进行回测，但是扩展函数、系统示例中的文件不可编辑修改，需要在用户策略中创建用户自己的策略。



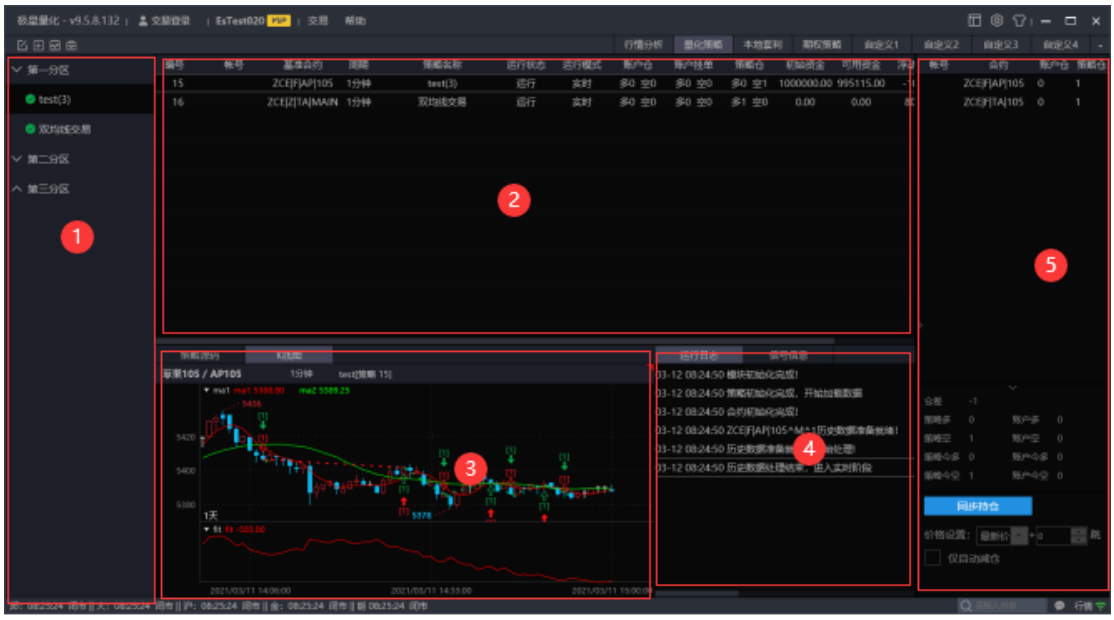
2. 添加策略

启动后在模型下拉框中选择需要加载的策略，如果策略中没有设置合约、K线类型、周期等参数需要在合约设置选项中添加新合约，并且可以添加多个合约。如果策略运行与账户有关，还可以在关联账户下拉框中选中您需要的的账户。

另外如果您需要更细致的更改有关参数，可以点击更多设置。在这里你可以对基本设置、发单设置、资金设置、参数设置按照您的需要进行更改。

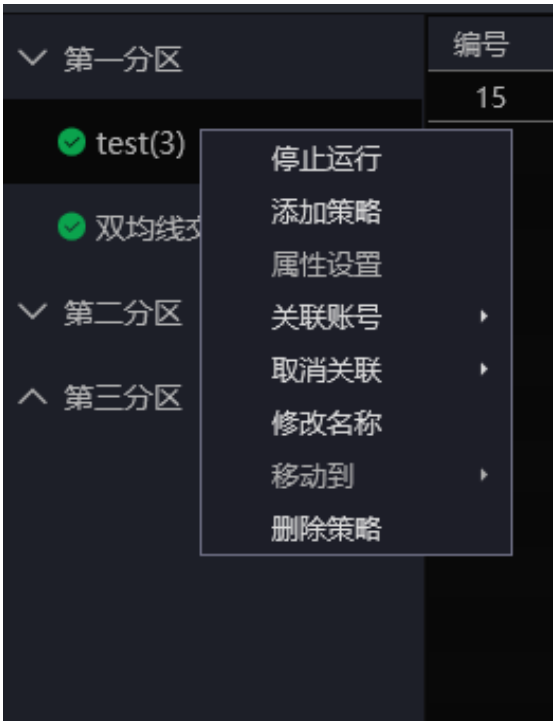
3. 策略运行数据

完成上述操作之后，策略即进入运行阶段，策略基本信息、策略源码、K 线图、日志等均显示在界面中。下面分区域进行说明：



区域 1：该区域按分区显示已经添加的策略，用户可以按个人喜好将策略区分开，或添加新的分区。选中某个策略，右键可以对策略进行激活或停止、属性设置、关联账号、删除策略等，另外可以双击某个策略进行策略的激活、停止。

选中不同的策略时，区域 2 的选中状态以及区域 3、4 的数据内容会自动切换。



区域 2：该区域只显示正在运行的策略在策略运行阶段，右键某一条策略可以查看回测报告（资金曲线、分析报告、阶段总结、交易详情），参数优化最后会做详细说明；

区域 3：该区域包括了策略源码的展示，以及策略运行的 K 线图（信号、绘图等）；

区域 4：包括了策略运行时产生的运行日志、信号信息。我们将旧版本的错误日志统一合并到运行日志中进行输出；

区域 5：该区域主要功能时为持仓同步，点击界面右上角图标可将该区域展示或折叠。

4. Python 库安装

因为我们内置的 python 依赖库有限，所以当用户需要安装更多依赖库的时候，我们也提供了一个更方便快捷的入口。如上左图所示，点击即进入右图所示依赖库安装界面。在该界面中用户可输入库名称选择安装，之后便可正常使用了。

5. 参数优化

我们发现在一段时间内表现很好的模型，过了一段时间就好像失效了一样，这种情况可能是由于模型参数不再适应当前市场行情引起的，我们需要统计历史数据寻找新的最优参数。大量的指标计算、参数筛选工作单凭人工计算几乎是不可能的，利用“参数优化”功能，可在指定的范围内让计算机筛选出最适合当前行情的参数。

下面以双均线模型作为参数优化的模板，说明具体使用方法。

首先我们设置两个参数 p1 和 p2，作为参数优化的对象。一个代表快线周期、另一个代表慢线周期。用作参数优化的值必须按照格式进行填写。

```
import talib

g_params['p1'] = 5
g_params['p2'] = 20
code1 = 'ZCE|F|AP|105'


def initialize(context):
    SetBarInterval(code1, 'M', 1, 200)
    SetTriggerType(5)
    SetOrderWay(2)
```

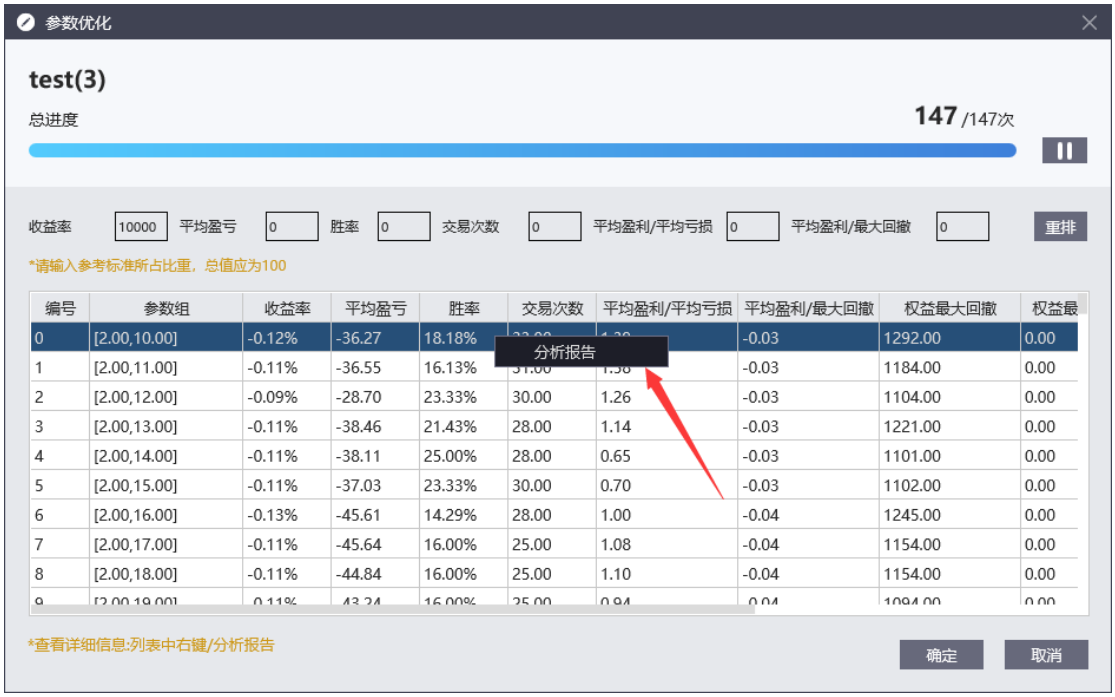
运行策略之后，右键策略，选择“参数优化”：

进入到参数优化界面，在该界面中可以设置参数的最小值、最大值和步长，我们默认步长为 1，用户可以按照自己的预期设置步长大小。参数会按照从最小值每次增加步长值一直到最大值结束。

设置参数关系：就像示例中快线 p1 和慢线 p2，应该保持 p1<p2，这样不仅可以剔除掉没有意义的参数组，还可以减少优化的次数，加快速度。

用户可以选择启用参数关系，当然也可以不启用，将所有参数组全部优化一次。

启用线程数我们会按照您的计算机核数选择适合的线程数，建议使用默认值：



点击确定之后，等待参数优化结束，在列表中会将所有参数组的收益率、胜率等以列表的形式展示出来，您可以选择您关心的某组参数右键查看更详细的分析报告。

3.4 函数、API、库

3.4.1API 使用方法

API 是我们提供的供用户写策略调用的接口，是我们组织好的相关联的代码段，用户可以通过函数获得以括号内参数为条件的返回值。以 arr = Close('ZCE|Z|SR|MAIN', 'M', 1) 为例，arr 为白糖主连的一分钟收盘价的集合。

函数关键的两个部分：参数、返回值。

参数：我们提供的所有函数的参数填写方式主要是必选参数和默认参数两种。同样以上面 Close 函数为例，'ZCE|Z|SR|MAIN'、 'M'和 1 均是默认参数，所以 arr = Close('ZCE|Z|SR|MAIN', 'M', 1)同样可以写作 arr = Close()，即用户可以不填，所有函数参数均取默认值（基准合约的相关信息）。相对应，必选参数则是需要用户使用时必须填写的，否则函数无法使用。具体区分参数默认参数和必备参数可以翻阅函数目录查看 API 文档，在函数详情界面的参数一栏会做说明（见如何查看 API 目录）。

返回值：函数返回值即函数调用后我们得到的具体结果。返回值的类型和 Python 的变量类型一致。注意我们的函数多有使用列表作为返回值，同样以上面 Close 为例，它的返回值 arr 就是一个列表，它是很多根 K 线收盘价的集合，arr[-1]表示当前 Bar 收盘价。所以在使用时注意返回值的具体内容，否则可能导致策略将无法运行。

3.4.2 如何看 API 文档

我们为用户提供了以下几个大类的 API 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。

在量化编辑器的右侧我们可以看到函数目录，我们可以按照目录进行翻阅、学习（见下图 3.4.1）。同样我们支持对函数的检索功能，我们可以根据自己需要的功能，输入函数名或者中文描述进行查找（见下图 3.4.2）。另外，在查看某个策略的时候，如果想了解策略中函数的具体功能时，可以将鼠标停留在函数名上 2 秒，会有函数说明以弹窗的形式作展示（见下图 3.4.3）。

图 3.4.1

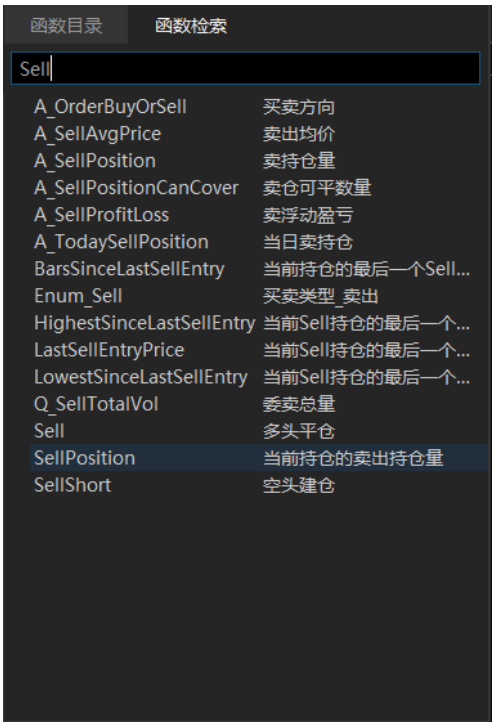


图 3.4.2

图 3.4.3

当在函数目录中选中某个函数时，在量化编辑器的右下窗口会展示函数的详细说明，包括：语法、参数、备注、示例等。用户在使用某个函数时，建议先研究一下函数说明，对函数的参数类型、值的范围、返回值的类型、注意事项等相关信息有一定的了解。另外我们也提供了简单的用法示例，可以指导用户如何使用该函数。（见图 3.4.4）

图 3.4.4

3.4.3 自定义函数用法

用户可以在策略中定义自己的函数以便在策略中调用，以下是简单的规则：

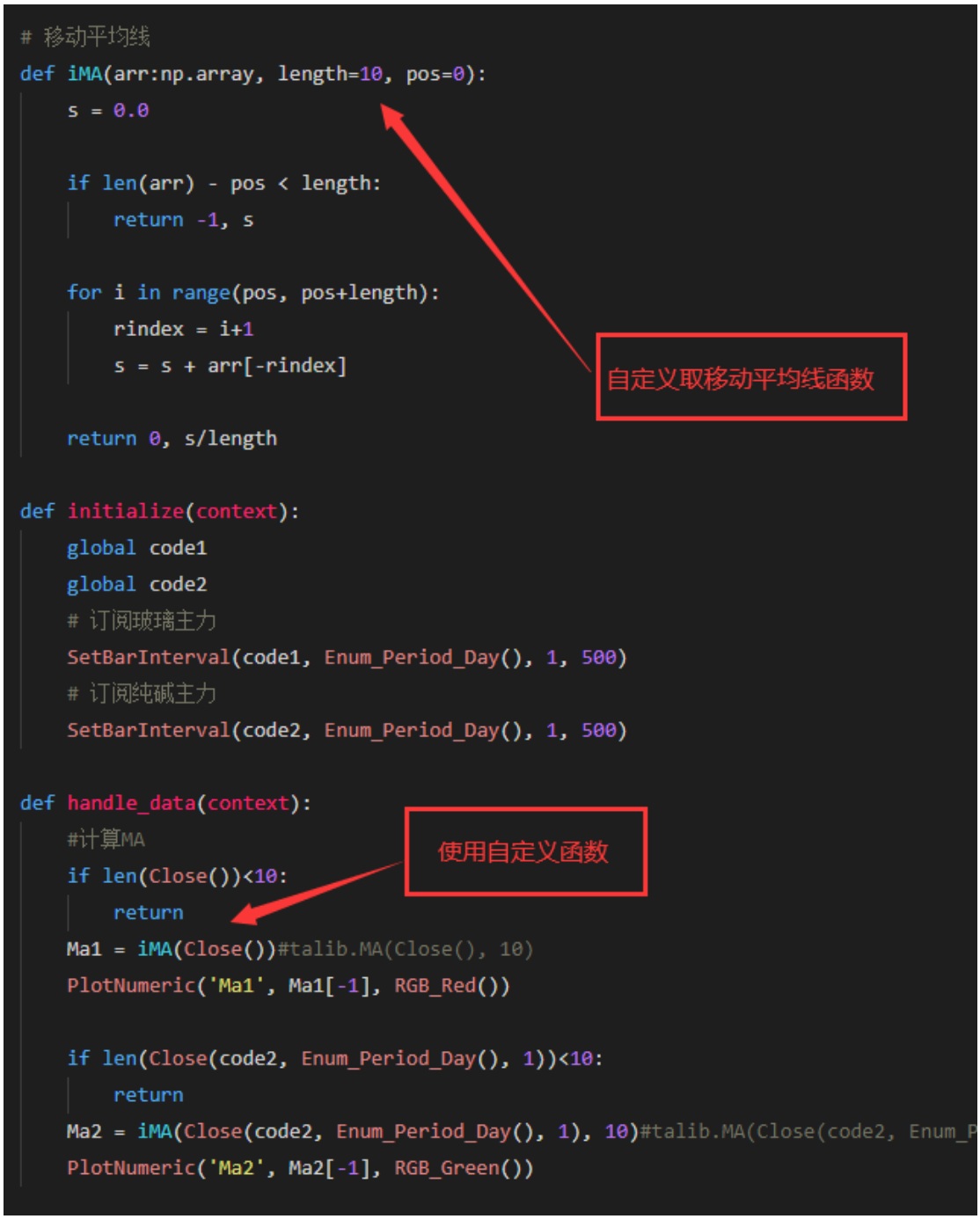
1. 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号()。
2. 任何传入参数必须放在圆括号中间。圆括号之间可以用于定义参数。
3. 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
4. 函数内容以冒号起始，并且缩进。
5. return［表达式]结束函数，选择性地返回一个值给调用方。不带 return 语句的函数相当于返回 None。

自定义函数语法如下：

```
def functionname(parameters):
    """函数_文档字符串"""
    function_suite
    return expression
```

其中 function_suite 表示函数体，是函数的具体实现；parameters 表示函数参数，可以为 0 个或多个；expression 表示函数的返回值。

具体的定义示例我们可以参照量化编辑器的策略中的系统示例策略---功能示例---多合约示例中函数定义方法进行学习和使用（见图 3.4.5）。




```
# 移动平均线
def iMA(arr:np.array, length=10, pos=0):
    s = 0.0

    if len(arr) - pos < length:
        return -1, s

    for i in range(pos, pos+length):
        rindex = i+1
        s = s + arr[-rindex]

    return 0, s/length

def initialize(context):
    global code1
    global code2
    # 订阅玻璃主力
    SetBarInterval(code1, Enum_Period_Day(), 1, 500)
    # 订阅纯碱主力
    SetBarInterval(code2, Enum_Period_Day(), 1, 500)

def handle_data(context):
    #计算MA
    if len(Close())<10:
        return
    Ma1 = iMA(Close())#talib.MA(Close(), 10)
    PlotNumeric('Ma1', Ma1[-1], RGB_Red())

    if len(Close(code2, Enum_Period_Day(), 1))<10:
        return
    Ma2 = iMA(Close(code2, Enum_Period_Day(), 1), 10)#talib.MA(Close(code2, Enum_P
    PlotNumeric('Ma2', Ma2[-1], RGB_Green())
```

图 3.4.5

3.4.4 常用的极智量化 API 介绍

1. K 线数据

历史 K 线信息主要包括以下信息：K 线的高开低收价格、成交量、持仓量、时间信息、K 线索引等信息。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，此外 HisData()可以等效获取指定类型的历史数据，HisBarsInfo()函数可以获取历史 K 线的详细信息。

Close 使用方法：

Close 作为其中使用较多的函数，经常作为下单指令的参数进行使用，Close()返回一个一定长度的数组。该数组按倒序的方式存储最新的收盘价，Close()[-1]代表最新的一根 K 线数据。

```
# 执行下单操作
if MarketPosition() <= 0 and ma1[-1] > ma2[-1]:
    Buy(1, Close()[-1])
if MarketPosition() >= 0 and ma1[-1] < ma2[-1]:
    SellShort(1, Close()[-1])
```

上图中 Buy，代表着以最近一根的收盘价买入一手。

在 K 线数据这组的函数 Open、Close、High、Low 使用方式完全一致，并且需要注意：

a. contractNo 参数不填写时，函数返回基准合约、基准周期对应的开盘价序列，其他参数填写或不填写都不生效

b. 该函数能取到的最大数据长度为用 SetBarInterval 函数订阅合约时的 barDataLen 参数的设置值，或是通过界面设置添加合约时在"引用根数"处设置的值

HisData 使用方法：

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

注意：

a. 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

b. 返回 numpy 数组，获取订阅的 maxLength 个指定的种类的历史数据

2. 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。基本的即时行情的数据都可以在该函数分类中提供的函数进行获取，全部函数可以翻看函数目录中的函数说明。调用该组函数时需要明确：历史阶段或没有即时行情时该函数返回默认值 0。

使用时我们可以将返回值直接输出，或者作为参数调用其他函数完成功能

LogInfo(“Q_Close: ”, Q_Close())

retCode, retMsg = A_SendOrder(Enum_Buy(), Enum_Exit(), 1, Q_AskPrice())

3. 策略交易

该函数分组列出了两组发单函数的中的一组，即 Buy，BuyToCover、Sell、SellShort。该组函数基本用于历史阶段进行简单发送订单，函数参数较少，用户使用起来比较简单。在实盘建议使用 A_SendOrder 函数，可以更详细的按照用户需求发送不同订单。

该函数以如下策略进行展示：

```
import talib

def initialize(context):
    SetTriggerType(5)
    SetOrderWay(2)

def handle_data(context):
    # 回测阶段 K 线触发
    if context.triggerType() == 'H':
        if CurrentBar() == 1:
            # 以最新收盘价，买开一手
            # 合约不设置时，以基准合约买入
            # needCover 默认为 True 此时如果当前有空仓，先平后开；False 时，不平仓，直接开仓
            Buy(1, Close()[-1])
        if CurrentBar() == 2:
            # 与 Buy 函数对应，以最新收盘价，卖开一手
            # 当前默认平仓为 True， 会将第一根 K 线买开全部平完之后，开 1 手空仓
            # 可参照输出的持仓信息对比用法
            SellShort(1, Close()[-1])
        if CurrentBar() == 3:
            # 以最新收盘价，平一手卖开仓
            # 合约不设置时，以基准合约为准
            BuyToCover(1, Close()[-1])
        if CurrentBar() == 4:
            # 与 BuyToCover 对应，该函数产生一个多头平仓操作
            # 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。
            # 由于前三根操作，当前买卖持仓均为 0，此时 Sell 函数不执行任何操作
```

```
        Sell(1, Close()[-1])
    LogInfo("++++++++++++++++++++++++++++++++")
    LogInfo("CurrentBar: ", CurrentBar())
    LogInfo("BuyPosition: ", BuyPosition())
    LogInfo("SellPosition: ", SellPosition())
    LogInfo("================================")
    #预期结果    1 1 0
    #            2 0 1
    #            3 0 0
    #            4 0 0
```

StartTrade、StopTrade 用法：

下图中示例了通过连接状态改变时进行暂停和恢复实盘交易的代码逻辑。交易服务器断开后暂停实盘交易，交易服务器连接时恢复实盘交易。

```
# 连状态触发
if context.triggerType() == 'N':
    if context.triggerData()['ServerType'] == 'T':# 交易
        if context.triggerData()['EventType'] == 1:# 交易连接
            # 盘中重新启动实盘交易
            StartTrade()
            LogInfo('StartTrade')
        else: # 交易断开
            # 盘中暂时停止实盘交易
            StopTrade()
            LogInfo('StopTrade')
        LogInfo('IsTradeAllowed', IsTradeAllowed())
```

4. 策略状态

当我们跑了一段时间的策略以后，需要获得该策略的状态的时候可以通过该组函数获得，其中包括：第一个建仓位置、建仓时间、策略 ID，持仓状态等。该组函数仅仅展示策略的具体信息。

该组函数中更多的使用到的时第一次建仓、平仓、最后一次建仓这些时机下的 Bar 计数、时间、价格等。账户信息不填时，统计策略中指定合约和投保标志的持仓的对应结果。只有在策略有持仓的状况下，该函数才有意义，否则返回-1。另外，在开仓 Bar 上为 0。

BarsSinceEntry 使用方法：

获取策略中的基准合约的投机仓的第一个建仓位置到当前位置的 Bar 计数

count = BarsSinceEntry()

获取策略中的用户名为"user"的 SR105 的投机仓第一个建仓位置到当前位置的 Bar 计数

count = BarsSinceEntry("ZCE|F|SR|905, "user")

MarketPosition 使用方法：

if(MarketPosition("ZCE|F|SR|905")==1)判断合约 ZCE|F|SR|905 当前是否持多仓

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

账户信息不填时，统计策略中指定合约和投保标志的持仓的对应结果。

返回值定义如下：

-1 当前位置持空仓数量大于持多仓数量

0 当前位置为持平

1 当前位置持多仓数量大于持空仓数量

该函数统计的持仓状态为虚拟回测引擎中的持仓对应的状态，与实盘账户中的持仓信息并不一致。

BarsLast 使用方法：

该函数参数要求比较严格，应该是一个 numpy 类型的 bool 数组，比如 Close() > Open()的结果就是一个 numpy 数组。将该值作为参数传入 BarLast 函数中即可。

BarsLast(Close() > Open());

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。

5. 策略性能

相对应策略状态，如果我们需要分析策略的优劣性能的时候就可以通过策略性能获取。该组函数提供了浮动盈亏、开仓次数、累计总利润、盈利成功率等函数。通过该函数返回值可以大致分析对应策略的性能。

该组函数比较简单，参数也比较少，可以在程序中直接调用或者输出函数返回值，例如：

```
LogInfo("CurrentEquity: ", CurrentEquity())
LogInfo("FloatProfit: ", FloatProfit())
LogInfo("GrossLoss: ", GrossLoss())
LogInfo("GrossProfit: ", GrossProfit())
LogInfo("Margin: ", Margin())
LogInfo("NetProfit: ", NetProfit())
LogInfo("NumAllTimes: ", NumAllTimes())
LogInfo("NumWinTimes: ", NumWinTimes())
```

6. 属性函数

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。

status = CommodityStatus('ZCE|F|SR')

CommodityStatus 函数优先返回品种状态信息，若没有品种的状态，则返回该品种所属的交易所的状态，python 版本没有品种状态时返回空字符串；

可能的返回值如下：

'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

TradeSvrState: 获取用户通过界面设置的关联账号或通过 SetUserNo 设置的第一个账户的连接状态，返回值 1 表示连接，2 表示断开

if TradeSvrState("12345678")==1:

ReloadStrategy()

当账号 12345678 连接正常时，重启策略

7. 账户函数

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

A_TotalPosition：

返回指定帐户下当前商品的总持仓，指定交易账号未登录或商品不存在时返回值为 0。

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。

适用于实时行情交易，不推荐在历史回测阶段使用。

A_OrderStatus：

返回指定帐户下当前商品的某个委托单的状态，返回值包括：

'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

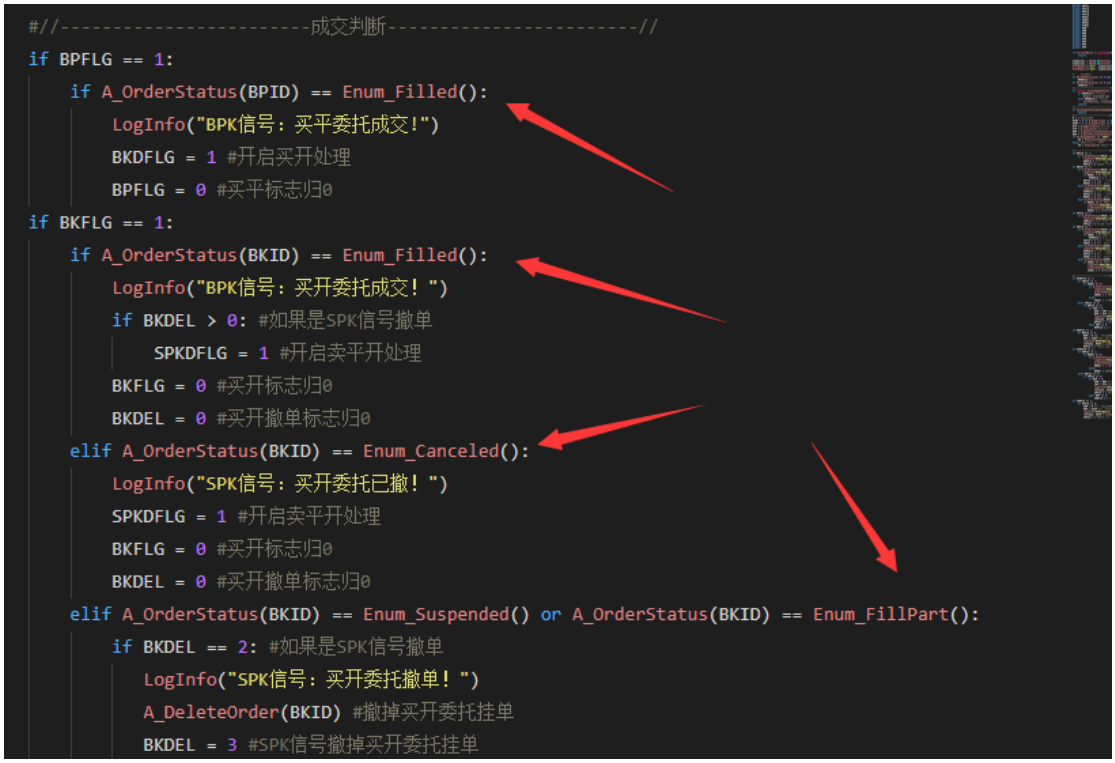
该函数返回值可以与委托状态枚举函数 Enum_Sended()、Enum_Accept()等函数进行比较，根据类型不同分别处理。

指定定单号不存在时返回值为空字符串“”。

指定定单号对应的合约在策略中没有订阅的话，返回值为“”。

适用于实时行情交易，不推荐在历史回测阶段使用。

用法示例可参考系统示例--账户示例中的策略代码：



```
#//----------------------成交判断----------------------//
if BPFLG == 1:
    if A_OrderStatus(BPID) == Enum_Filled():
        LogInfo("BPK信号：买平委托成交!")
        BKDFLG = 1 #开启买开处理
        BPFLG = 0 #买平标志归0
if BKFLG == 1:
    if A_OrderStatus(BKID) == Enum_Filled():
        LogInfo("BPK信号：买开委托成交！")
        if BKDEL > 0: #如果是SPK信号撤单
            SPKDFLG = 1 #开启卖平开处理
        BKFLG = 0 #买开标志归0
        BKDEL = 0 #买开撤单标志归0
    elif A_OrderStatus(BKID) == Enum_Canceled():
        LogInfo("SPK信号：买开委托已撤！")
        SPKDFLG = 1 #开启卖平开处理
        BKFLG = 0 #买开标志归0
        BKDEL = 0 #买开撤单标志归0
    elif A_OrderStatus(BKID) == Enum_Suspended() or A_OrderStatus(BKID) == Enum_FillPart():
        if BKDEL == 2: #如果是SPK信号撤单
            LogInfo("SPK信号：买开委托撤单！")
            A_DeleteOrder(BKID) #撤掉买开委托挂单
            BKDEL = 3 #SPK信号撤掉买开委托挂单
```

A_SendOrder：

针对指定的帐户、商品发送委托单。

部分参数类型说明：

orderType 定单类型，字符类型，默认值为'2'，可使用如Enum_Order_Market()、Enum_Order_Limit()定单类型枚举函数获取相应的类型，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

validType 定单有效类型，字符类型，默认值为'0'，可使用如 Enum_GFD() 定单有效类型枚举函数获取相应的类型，可选值为：
'4';//即时全部
'3';//即时部分
'0';//当日有效
'1';//长期有效
'2';//限期有效

hedge 投保标记，字符类型，默认值为’T’，可使用如 Enum_Speculate()、Enum_Hedge()定单投保标记枚举函数获取相应的类型，可选值为：
'T';//投机
'B';//套保
'S';//套利
'M';//做市

triggerType 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 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 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 触发价格，默认价格为 0。

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""。

返回结果形式未：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。

适用于实时行情交易，不推荐在历史回测阶段使用。

示例代码：

retCode, retMsg = A_SendOrder(Enum_Buy(), Enum_Exit(), 1, Q_AskPrice())

当 retCode 为 0 时表明发送定单信息成功，retMsg 为返回的下单编号 localOrderId。

8. 枚举函数

枚举类型，主要作为其他函数的参数值进行使用。或用来判断一个函数的返回值是否符合特定类型。

在使用 A_SendOrder 发送委托单时使用较多：

A_SendOrder(Enum_Buy(), Enum_Entry(), 1, Q_AskPrice(), ContractId, UserId, Enum_Order_Limit(), Enum_FOK(), Enum_Speculate())

9. 设置函数

该组函数主题提供用户在代码中设置交易账号、初始资金、手续费、交易方向、滑点损耗等等信息。与这些函数相对应的，界面设置中同样可以进行设置，更加方便使用。如果界面和代码同时设置了，将以代码为准或者合并两个值，同时设置。

设置函数基本在 initialize 中进行书写：

```
def initialize(context):
    # 建议用户在界面中设置，保存为默认值
    # 注意：设置函数需要在初始化函数中进行调用（除止损止盈外）
    # 未设置的参数，将以界面设置为准；当界面和代码中均有设置的时候，代码设置的参数会覆盖界面设置
    # （除 SetTriggerType， 该函数合并两种设置的参数）

    # 订阅历史 K 线数据（自动订阅即时行情）
    # 版本新增 maxLen，代表高开低收函数返回值可取到的最大数量（可提升策略性能）
    # 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, 500, 200)

    # 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
    #SetTriggerType(6)                              # 连接状态触发
```

```
    # 设置发单方式 仅对当 SetTriggerType 为 5（K 线触发）时才有意义
    SetOrderWay(1)                                  # 实时发单
    #SetOrderWay(2)                                 # K 线稳定后发单
```

```
    #SetActual()                                    # 设置实盘运行
    #SetUserNo("Test1")                             # 设置默认交易账号(已登录)，策略只接收界面和函数指定交易账号的交易数据

    #SetInitCapital(200*10000)
```

```
    #SetMargin(0, 0.08)                          # 第一个参数 0：按比例收
取保证金， 1：按定额收取保证金
    #SetTradeFee('O', 2, 5)
    #SetTradeDirection(0)                        # 双向交易
    #SetMinTradeQuantity(1)
    #SetHedge('T')                               # 默认使用投机类型
    #SetSlippage(1)                              # 设置滑点
```

10. 绘图函数

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

PlotNumeric：

```
    # 使用 talib 计算均价
    ma1 = talib.MA(Close(), p1)
    ma2 = talib.MA(Close(), p2)
    # 绘制红色指标图形 主图
    PlotNumeric("ma1", ma1[-1], RGB_Red())
    # 绘制红色指标图形 主图 不使用独立坐标
    PlotNumeric("ma2", ma2[-1], RGB_Green(), True, False)
    # 绘制红色指标图形 主图 不使用独立坐标 回溯偏移 2
    PlotNumeric("ma3", ma2[-1], RGB_Blue(), True, False, 2)
    # 绘制红色指标图形 副图 使用独立坐标
    PlotNumeric("fit", ma2[-1], RGB_Red(), False, True)
```

全部的绘图函数示例，见系统示例——函数示例——8-绘图函数

11. 统计函数

该组函数类似于 talib，提供给用户做数据统计或数据分析，现主有函数有：计算加权移动平均线、计算抛物线转向、求 N 周期前数据的值、求最高最低、获取最近 N 周期条件满足的计数、求是否上穿下穿、求波峰点波谷点。

Highest:

HH1 = Highest (Close(), 12); 计算 12 周期以来的收盘价的最高值；

HH2 = Highest (HisData(Enum_Data_Typical()), 10); 计算 10 周期以来高低收价格的平均值的最高值。

CrossOver:

CrossOver(Close[1], 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）。

12. 日志函数

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

日志函数中使用较多的是 LogInfo

LogInfo:

在运行日志窗口中打印用户指定的普通信息。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。

```
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))
```

Alert:

Alert("Hello"); 弹出提示

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']) + ''
```

13. context 对象

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

context.triggerType()示例：

首先需要在 initialize 设置不同的触发方式

```
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
```

然后在 handle_data 中就可以根据不同的触发方式返回值做不同的操作，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；'N' 连接状态触发；'Z' 市场状态触发

例如：

```
    # 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())
```

triggerData 获取当前触发类型对应的数据：

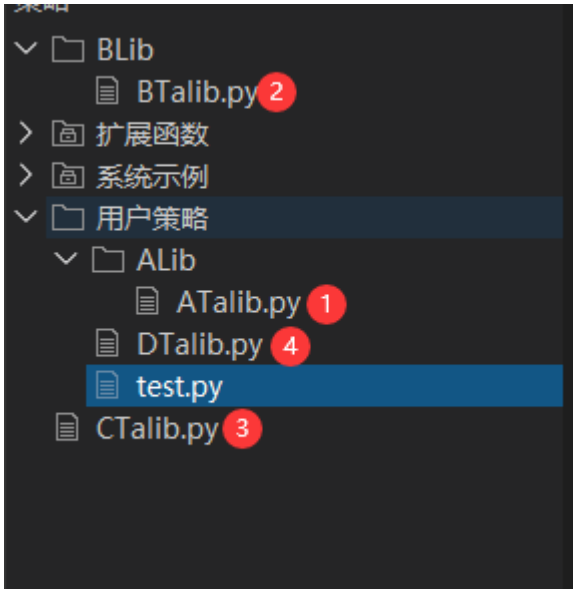
返回值类型为字典，具体字典关键可参照函数说明进行查看，想要使用没有个字段时的使用方法如下：

```
    # 定单状态变化    已发送--已受理--已排队--完全成交
    #                已发送--已受理--已排队--部分成交--（撤单）--已撤余单
    #                已发送--已受理--已排队--（撤单）--已撤单
    #                已发送--已受理--指令失败
    #                中间状态 已受理 已排队 完全成交 等可能会收到多次，请根据实际需要过滤重复
    LogInfo("报单状态: ", context.triggerData()['OrderState'])
```

3.4.5 导入自己实现的库

如果您已经自己实现了一个库文件，当需要在策略中使用的时候，我们建议您将个人库放置在量化策略存放的目录下面：

（...\Quant000150v9.5\Quant\Strategy)。最好的情况是在量化策略编辑器的用户策略下新建文件夹单独放置，便于区分且好管理。当然也可以放置在另外您觉得需要的位置。



例如，如果想在 test 策略中导入个人库，那么相对于 test，库文件的放置位置主要情形有以下四种：

1. 同级目录下（④）
2. 相同文件夹下的下级目录（①）
3. 上层目录中模块（③）
4. 其他目录(平级)下的模块（②）

其他情况不再做说明，用户如有需要，可自行探索。

针对上述四种的情况，不同的情况有多种不同的导入方法。为方便用户理解和使用，这里仅介绍一种可以统一解决四种情况的方法---按照相对路径导入。

当前的量化工作路径是：

C:\Users\XXXXX\AppData\Roaming\Quant000150v9.5，所以按照相对路径导入库时候，需要在相对路径后补全我们个人库所在的位置。导入示例代码如下：

```
import talib
```

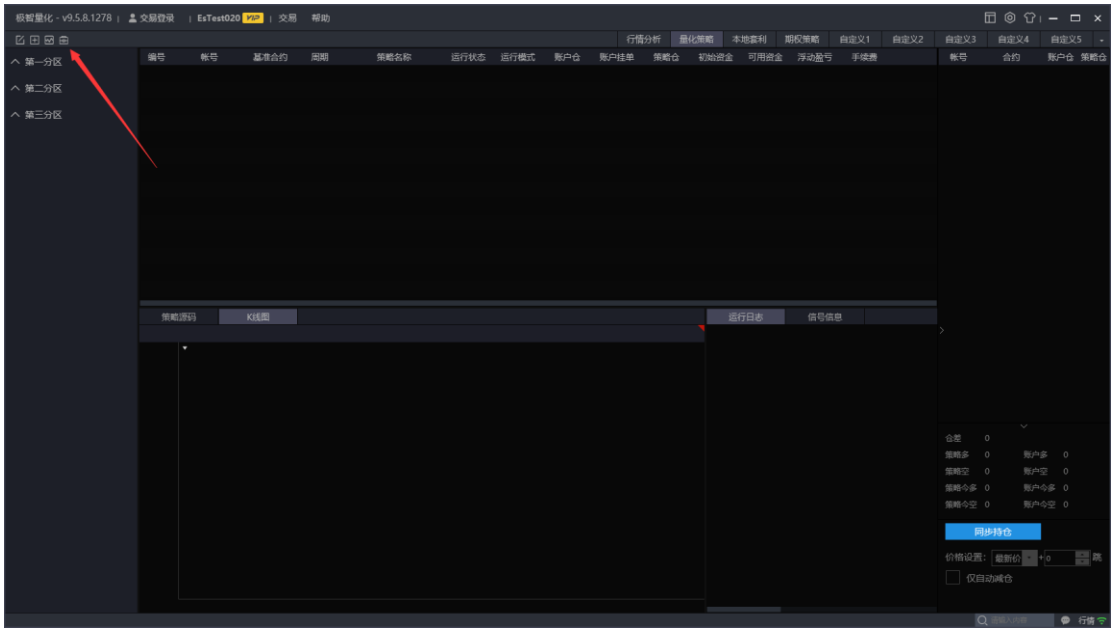
```
from Quant.Strategy.BLib import BTalib
from Quant.Strategy import CTalib
from Quant.Strategy.用户策略.ALib import ATalib
from Quant.Strategy.用户策略 import Dtalib

def initialize(context):
    SetBarInterval('ZCE|Z|SR|MAIN', 'M', 1, 200)
    SetTriggerType(5)
    SetOrderWay(2)

def handle_data(context):
    LogInfo("ATalib: ", ATalib.U_Lowest(Close()))
    LogInfo("BTalib: ", BTalib.U_Lowest(Close()))
    LogInfo("CTalib: ", CTalib.U_Lowest(Close()))
    LogInfo("DTalib: ", DTalib.U_Lowest(Close()))
```

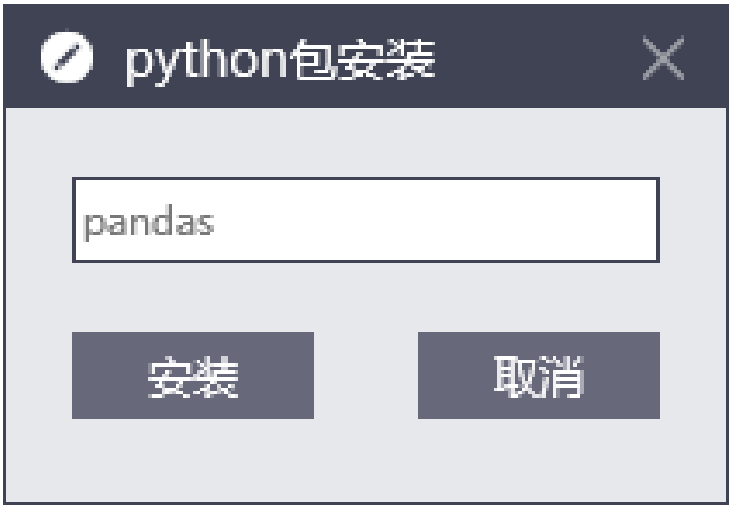
由于 ATalib 和 test.py 在同级，DTalib 位于 test.py 的下一层。这两种情况也可以简化为：

```
from ALib import ATalib
import DTalib
```

3.4.6 安装 python 第三方库

如果我们预装的第三方库不能满足您的使用需求时，我们在极智量化---量化策略中提供了更为方便的界面操作安装功能。如下图所示：





如果我们想要安装 pandas 等第三方库时，只需要在输入框中输入库名称，便可自动安装。如果想要指定安装的第三方库的版本号，也可以直接在 python 包安装的输入框中指定，和命令行安装方式相同。

3.5 编写第一个策略

上一节介绍了如何使用极智量化的 API 以及如何使用自定义库和安装 python 的第三方库，这部分开始介绍如何开发一个简单的策略。

3.5.1 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“添加策略”按钮，弹出策略运行设置对话框。该对话框上可以选择策略添加分区、合约设置以及需要关联的账号，更多设置中包含了基本设置，发单设置，资金设置和参数设置。

1）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删除，将添加的第一个合约作为基准合约。

2）基本设置

基本设置用来设置策略的触发方式：

触发方式说明可在 3.5.2 触发方式一节查看。

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。

默认数量：

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

保证金率：每手交易的保证金

开仓收费：进行开仓操作时手续费收费方式：固定值、比例，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，手续费为定单价格*每手乘数*比率(%)

平仓收费：进行平仓操作时手续费收费方式：固定值、比例，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，手续费为定单价格*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）发单设置

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。具体用户参数说明参见 4.3 参数优化。

3.5.2 触发方式

触发方式是极智量化产品运行策略必须设置的选项，否则策略将不会运行。极智量化提供了六种触发方式供用户选择：

- K 线触发：由基准合约的 K 线数据变化触发策略，运行策略的 handle_data()函数，其他合约+周期的历史数据会更新到本地，可以访问但是不触发。
- 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
- 连接状态触发：行情服务器的连接状态以及交易账号的连接状态的改变会触发策略，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。也可以通过在添加策略时在更多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。

3.5.3 获取数据

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 K 线数据函数如 Open()、Close()、High()、Low()等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。如：

```
# 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。但是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**，非基准合约并不会触发 handle_data 函数，但同样可以用上述方法获取非基准合约的 K 线数据信息。

3.5.4 获取 context 数据

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、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 线数据或即时行情数据或定单数据等)

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量化编辑器的“系统示例--1-函数示例--2-context 上下文.py”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量化编辑器的函数目录中的 context 函数部分的 triggerData 函数说明部分查看。

3.5.5 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：撤销当前策略订阅合约的所有排队单

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，该函数会向历史回测引擎发送一笔委托单的同时向交易后台发送一笔委托单。

```
    A_SendOrder(Enum_Buy(), Enum_Entry(), 1, Close()[-
2], 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。

3.5.6 编写自己的策略

对极智量化的 API 有了一定的了解后，用户可以着手编写一些简单的策略。前面已经对策略框架进行了介绍，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

# 策略参数字典
g_params['p1'] = 20    # 参数示例

# 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
```

并设置了一个用户参数 p1：

```
g_params['p1'] = 20     # 参数示例
```

g_params 是量化策略引擎为用户定义的变量，供用户参数优化使用。本策略中并未使用。

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：

```
# 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")
```

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：

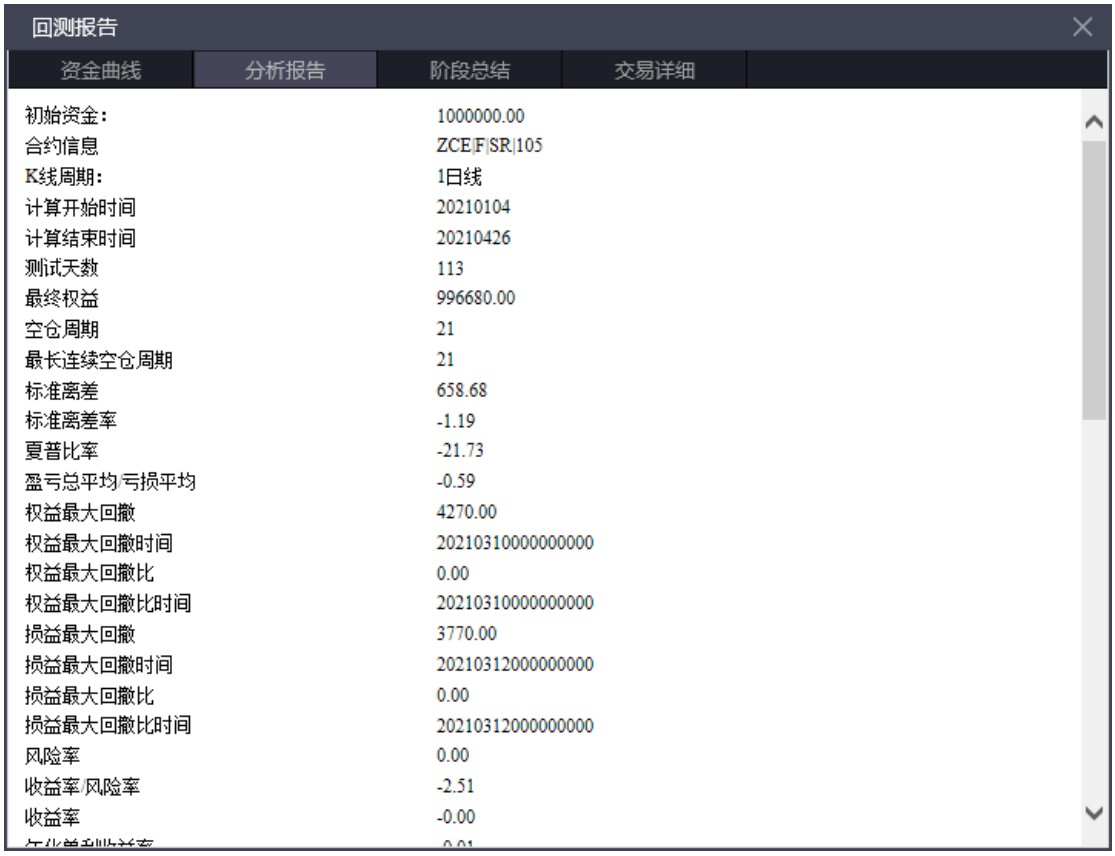
```
import talib
import time
```

```
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。

3.5.7 回测报告说明

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，可选中某个运行中的策略，右键选择"回测报告"查看策略运行的详细信息：



回测报告列出了"资金曲线"、"分析报告"、"阶段总结"、"交易详细"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。

分析报告部分指标计算方式：

测试天数	从测试数据开始到结束的天数
空仓周期数	空仓的周期数
最长连续空仓周期数	最长连续空仓的周期数
标准离差	盈亏的标准离差= √(SUM((每次盈亏-平均盈亏)^2,N)/ N)
标准离差率	盈亏的标准离差率=标准离差/平均盈亏

夏普比率	夏普比率=（平均年收益率-无风险利率）/收益率的标准离差率 计算公式：夏普比率=[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 1.5%） 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	平均盈亏和平均亏损的比值 =（(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	权益最大回撤出现的时间
权益最大回撤比	权益最大回撤和权益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	权益最大回撤比出现的时间
损益最大回撤	从测试开始到结束，动态损益计算出来的波段从高点到低点回撤的最大 值（损益最大回撤是以持仓等于 0 时的资金为标准计算的）
损益最大回撤时间	损益最大回撤出现的时间
损益最大回撤比	损益最大回撤和损益最大回撤时的最大权益的比值的最大值
损益最大回撤比时 间	损益最大回撤比出现的时间
风险率	本金最大回调/本金=（资金分配量 － 期间最小权益）/资金分配量

收益率/风险率	年化单利收益率/风险率
年化单利收益率	单利收益率/（测试天数/365）
月化单利收益率	单利收益率/（测试天数/30）
年化复利收益率	(期末权益/期初权益)^（365/测试天数）-1
月化复利收益率	(期末权益/期初权益)^（30/测试天数）-1
胜率	非亏损次数占总交易次数的百分比
平均盈利/平均亏损	平均盈利 = 总盈利/盈利交易次数 平均亏损 = 总亏损/亏损交易次数
平均盈利率/平均亏损率	平均盈利率 =SUM（单次盈利率) / 计算单次盈利率次数 平均亏损率 =SUM（单次亏损率）/ 计算单次亏损率次数 单次盈利率 = 上一次持仓为 0 到今次持仓为 0 期间的盈利占期初权益的 百分比 （盈利/期初权益） 单次亏损率 = 上一次持仓为 0 到今次持仓为 0 期间的亏损占期初权益的百分比 （亏损/期初权益）
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
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	平均每笔交易的盈亏=总利润率/总交易次数

平均盈利	平均每笔盈利交易的盈利 = 总盈利/总盈利次数（计算手续费）
平均亏损	平均每笔亏损交易的亏损 = 总亏损/总亏损次数（计算手续费）

4.常用功能介绍

4.1 多合约

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：

```
import talib as ta
import numpy as np

code1 = 'ZCE|Z|FG|MAIN'
code2 = 'ZCE|Z|SA|MAIN'

def initialize(context):
    global code1
    global code2
    # 订阅玻璃主力
    SetBarInterval(code1, Enum_Period_Day(), 1, 500)
    # 订阅纯碱主力
    SetBarInterval(code2, Enum_Period_Day(), 1, 500)
```

在策略中调用 SetBarInterval 函数订阅不同的合约，即可在策略中获取不同合约的数据。

极智量化可以使用 SetBarInterval 或 SubQuote 函数订阅合约，由于这两个函数都只支持订阅最多十个合约，因此极智量化最多支持订阅 20 个合约的数据。

4.2 多周期

同多合约订阅方式相同，用户只需要在策略中调用 SetBarInterval 函数订阅相同合约不同周期的数据，即可在策略中获取同一合约不同周期的数据。

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

```
ContractId = 'SHFE|Z|CU|MAIN'
def initialize(context):
    # 订阅 15 分钟
    SetBarInterval(ContractId, Enum_Period_Min(), 15, 100)
    # 订阅日线
    SetBarInterval(ContractId, Enum_Period_Day(), 1, 100)
```

极智量化目前支持订阅 Tick、秒、分钟、日这四个类型的 K 线数据。K 线周期可以任意指定。需要说明的是，订阅 Tick 和秒线数据时所指定的 K 线类型都为 Enum_Period_Tick()，或者用'T'表示。

订阅沪铜 2 秒线数据时 SetBarInterval 函数使用方法为：

```
    SetBarInterval('SHFE|Z|CU|MAIN', Enum_Period_Tick(), 2, 100)
```

订阅沪铜 Tick 数据时 SetBarInterval 函数使用方法为：

```
    SetBarInterval('SHFE|Z|CU|MAIN', Enum_Period_Tick(), 0, 100)
```

注意：订阅 Tick 数据时，周期值设为 0。

4.3 参数优化

用户在进行策略编写时，如何选择一组合适的参数，使策略能够获取长期、稳定的收益，对编写策略而言十分重要。如双均线策略的短周期和长周期的值如何选择才能获取尽可能多的交易回报呢？我们为用户提供了参数优化功能，该功能根据用户对参数设置的起始值、结束值和步长，运算所有不同的参数组合，以便将最佳的参数组合找出来。用户可以查看参数优化的操作方法了解如何进行参数优化。

我们将待优化的参数组进行循环遍历回测，记录每次回测结果和参数，然后就可以根据某种规则将回测结果排序，找到最佳的参数。当然，建议不要用“最佳”的参数，因为可能会出现“过拟合”问题。

量化策略引擎为用户定义了 g_params 变量供用户进行参数优化使用。g_params 是 python 的字典类型，要进行参数优化，用户首先需在策略中设置 g_params 的键值对：

```
g_params['fast'] = 12 # 快周期
g_params['slow'] = 26 # 慢周期
g_params['back'] = 9  # dea 周期
```

这里定义了三个用户参数，快周期 fast=12、慢周期 slow=26 和 dea 周期 back=9，设置完用户参数之后，在策略中计算 MACD 指标，填入这三个用户参数：

```
import talib
```

```
g_params['fast'] = 12 # 快周期
g_params['slow'] = 26 # 慢周期
g_params['back'] = 9  # dea 周期

qty = 1   # 下单量
macd_dx = 0.01 #macd 阀值

def initialize(context):
    # 设置 K 线稳定后发单
    SetOrderWay(2)
    # 设置基准合约，会覆盖界面设置的合约，建议通过界面设置(屏蔽
SetBarInterval 后则界面添加合约生效)
    SetBarInterval('ZCE|Z|SR|MAIN', 'M', 1, 200, g_params['slow']
 + g_params['back'] - 1)

def handle_data(context):
    # 等待数据就绪，否则计算结果为异常值
    if CurrentBar() < g_params['slow'] + g_params['back'] - 1:
        return

    # 计算 MACD
    diff, dea, macd = talib.MACD(Close(), g_params['fast'], g_par
ams['slow'], g_params['back'])

    # 突破下单
    if MarketPosition() <= 0 and macd[-1] > macd_dx:
        Buy(qty, Close()[-1])
    elif MarketPosition() >= 0 and macd[-1] < -macd_dx:
        SellShort(qty, Close()[-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)
```

运行策略，进行参数优化，并设置期望目标，根据每次回测结果，找出最优的参数组合。

g_params 使用时注意事项说明：

1. **提取时机 添加策略时**

一般情况下我们会已经写好了包含有 g_params 参数的策略：

```
g_params['p1']=5
g_params['p2']=20

def initialize(context):
    SetBarInterval('ZCE|Z|SRIMAIN', 'M', 1, 200)
    SetTriggerType(5)
    SetOrderWay(2)

def handle_data(context):
    if len(Close()) < g_params['p2']:
        return

    #使用 talib 计算均价
    ma1=talib.MA(Close(), g_params['p1'])
    ma2=talib.MA(Close(), g_params['p2'])
```

然后添加策略运行，可以看到在参数优化处提取参数是完全有效的。如下所示：

但是当我们添加完策略后，如果我们想要再在策略中添加一个g_params['p3']=50，那么此时直接重新运行该策略时在参数优化界面是拿不到p3的值的，也就无法对包含有p3的情况进行优化。这种情况就需要将原来添加的策略删掉，重新添加该策略运行。

出现这种情况的原因，主要是我们提取参数的时机造成。我们只在添加策略时对所有参数进行提取，重新运行时不再进行参数提取操作。

2. 生效时机 函数内部使用 initialize 调用时生效

```
g_params['p1']=5
g_params['p2']=20
c = g_params['p2']
```

如上图所示，当我们定义了p1和p2之后，将某个值赋给变量c时。虽然在使用中用到c不会报错，但是当我们进行参数优化时，p1和p2的值会按照每一组新值进行重新赋值，但是变量c会一直保持第一次赋值的20不变。这样就对我们参数优化的正确性产生了影响。产生这种情况是和我们参数组赋值的机制有关，参数优化的每一组新的值都只在我们策略的约定函数中生效，在函数外部使用的g_params不会受参数新值的影响，所以进行参数优化时c的值20不再随参数p2的变化而变化。想要变量c的值随参数的优化值生效可以按如下方式进行操作：

```
g_params['p1']=5
g_params['p2']=20
c = 1

def initialize(context):
    global c
    SetBarInterval('ZCE|Z|SRIMAIN', 'M', 1, 200)
```

```
    SetTriggerType(5)
    SetOrderWay(2)
    c = g_params['p2']
```

先在函数外部定义全局变量 c，然后在 initialize 函数中将参数的值赋给 c。这样每次参数优化的值也同样会对变量 c 产生影响。

4.4 止损止盈

极智量化的止损止盈支持 3 种方式，固定点止损、固定点止盈和浮动止损。通过 API 函数设置止损止盈方式，策略会在止损止盈条件满足时自动进行操作。

1） 固定点止损

设置固定止损方式如下：

```
    SetStopPoint(10)
```

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，当前价格下跌不小于 5*10=50 时，即达到 14950，则进行止损平仓。用户还可以设置平仓的下单价格类型和超价点数。函数的具体用法可查看函数说明。

2） 固定点止盈

设置固定止盈的方式如下：

```
    SetWinPoint(10)
```

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，当前价格上涨不小于 5*10=50 时，即达到 15050，则进行止盈平仓。用户还可以设置平仓的下单价格类型和超价点数。函数的具体用法可查看函数说明。

3） 浮动止损

设置浮动止损的方式如下：

```
    SetFloatStopPoint(20, 10)
```

其中 20 表示启动点数，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；10 表示止损点数；如郑棉合约，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。用户还可以设置平仓的下单价格类型和超价点数。函数的具体用法可查看函数说明。

这是一种动态止损方法，止损价位会随着盈利的增加而变化，这种方法可以最大程度实现“让盈利奔跑”。如下图所示，这是以做多开仓为例，开仓后的行情最高价每上涨一个价位，止损平仓价就跟着上涨一个价位，当价格从最高

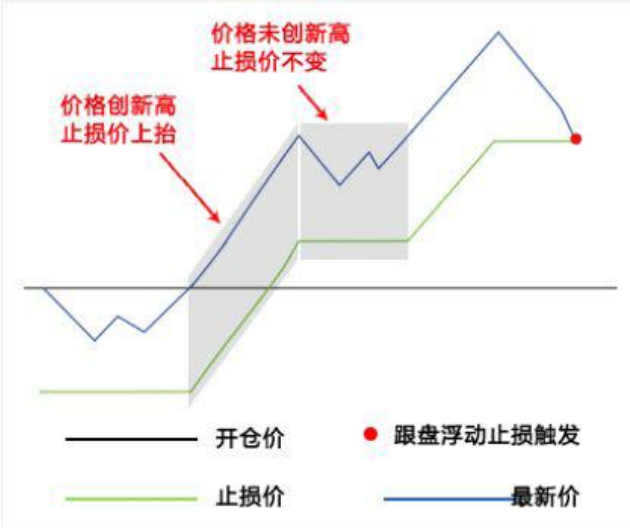
价回撤到设置的止损价差时，触发止损。下图为做多跟踪止损示意图，做空则相反。

最高价：此最高价是开仓后的最高价。

“跟盘浮动，价位回撤”计算公式：

多头止损价位 = 启用止损后的最高价 - 回撤价位

空头止损价位 = 启用止损后的最低价 + 回撤价位



注意：以上所有的止损止盈的点数都需乘以对应品种的最小变动价位

4.5 套利

极智量化不仅支持交易所标准合约的套利，也支持自定义的跨期、跨品种套利，用户可以根据需要自由的编写套利策略。

若要进行交易所标准套利，因为无法获取交易所标准套利合约的行情，需要先订阅对应的极星 SPD 套利合约的行情，根据极星 SPD 套利合约的行情，对交易所标准套利合约进行下单，示例策略如：

```
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|107'
code2 = 'ZCE|F|TA|109'
code = 'ZCE|S|TA|107|109'
```

```
spd = 'SPD|s|TA|107|109'

##########################################################################
################
# 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(), F
alse)
```

若要进行自定义的套利，用户只需要在策略中订阅每一腿的合约，然后根据自己的套利构思实现策略即可。以 TA107、TA109 合约套利为例，实现简单的布林带套利策略：

```
import talib
import numpy as np

code1="ZCE|F|TA|107"
code2="ZCE|F|TA|109"
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(), F
alse, True)
```

4.6 指标计算

极智量化的技术分析指标的计算方法是使用的 python 第三方库 talib 直接封装好的接口。talib 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。talib 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）

以 MA 指标的计算方法为例，首先需要在策略的开始部分导入 talib 库，才可以在策略中使用 talib 中的 MA 方法计算指标：

```
import talib
```

调用 K 线函数 Close()获取合约的收盘价数据列表，作为 MA 指标的数据源。计算方法如下：

```
    # 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)
```

这样就得到了 5 分钟和 20 分钟的苹果主连合约的收盘价移动平均指标数据。

同样可以计算收盘价的 MACD 指标：

```
    # 计算 MACD
    diff, dea, macd = talib.MACD(Close("ZCE|Z|AP|MAIN", "M", 1),
12, 26, 9)
```

其中 12 位快周期，26 位慢周期，9 是信号周期。

4.7 绘制指标

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输出值的显示颜色，绘制的图形被绘制到主图还是附图上，指标是否使用独立坐标绘制。这里以 PlotNumeric 函数为例，演示如何绘制指标线：

```
    ma5 = talib.MA(Close("ZCE|Z|SR|MAIN", "M", 1), 5)
    PlotNumeric("ma1", ma5[-1], 0xFF0000)
```

绘制的指标如下图所示，指标名称和指标值显示在 K 线图的左上角，指标线绘制在主图上，为图中的红色折线。

4.8 多档行情

通过极智量化提供的即时行情获取接口，用户可以很容易的获取到多档行情。我们为用户提供了 4 个函数用于获取合约的深度行情。内合约最多可以获取 5 档的深度行情，外盘合约可以最多获取 10 档的深度行情。

1）Q_BidPrice：获取合约最新买价

```
Q_BidPrice("ZCE|Z|SR|MAIN", 3)  #获取白糖三挡行情的买价
```

2）Q_BidVol：获取合约最新买量

```
Q_BidVol("ZCE|Z|SR|MAIN", 3)  #获取白糖 3 档行情的买量
```

3）Q_AskPrice：获取合约的最新卖价

```
 Q_AskPrice("ZCE|Z|SR|MAIN", 2)  #获取白糖 2 档行情的卖价
```

4）Q_BidVol：获取合约最新卖量

```
Q_BidVol("ZCE|Z|SR|MAIN", 2)  #获取白糖 2 档行情的卖量
```

4.9 持仓同步

采用 Buy、Sell 函数下单的策略，经常会出现策略持仓和账户实际持仓不一致的情况。我们称为存在“仓差”，为了解决这个问题，系统提供了持仓一键同步功能供用户选择使用。当然，用户也可以采用 A_SendOrder 函数自行编写策略来实现持仓矫正功能。

如下图所示，点击“同步持仓”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定单进行撤单，然后根据仓差情况发送等量的定单。发送定单的价格是根据“价格设置”中的选项计算得出。

注意：使用持仓同步功能时，用户首先要登录交易账号并设置策略为实盘运行。持仓同步会对列表中的所有合约进行持仓同步。

5.极智量化与相关产品接口名称对比

5.1 极智量化与 TB 函数对比

这里列出了极智量化与 ETL 和 TB 的 API 接口函数名的对照表：

数据函数						
编号	equant(19)	ETL（14）	TB（12）	说明		
1	BarCount			指定合约 Bar 的总数		
2	BarStatus	BarStatus	BarStatus	当前 Bar 的状态		
3	Close	Close/C	Close/C	收盘价	CLOSE C	
4	CloseD			指定合约 N 天前的收盘价		
5	CurrentBar	CurrentBar	CurrentBar	当前 Bar 的索引值		
6	Date	Date/D	Date/D	当前 Bar 的日期		
7	High	High/H	High/H	最高价	HIGH H	
8	HighD			指定合约 N 天前的最高价		
9	HisBarsInfo			获取最多 maxLength 根指定类型的历史 K 线详细数		
10	HisData	HisData		获得各种历史数据		
11	HistoryDataExist	HistoryDataExist	HistoryDataExist	历史数据是否存在		
12	Low	Low/L	Low/L	最低价	LOW L	

13	LowD			指定合约 N 天前的最低价		
14	Open	Open/O	Open/O	开盘价	OPEN O	
15	OpenD			指定合约 N 天前的开盘价		
16	OpenInt	OpenInt	OpenInt	持仓量		
17	Time	Time /T	Time /T	当前 Bar 的时间		
18	TradeDate	TradeDate		当前 Bar 的交易日		
19	Vol	Vol/V	Vol/V	成交量		
20		TodayAvgPrice		获取当日均价		
21			Truedate	求指定 bar 的真正交易日期		
行情函数						
编号	equant(33)	ETL（20）	TB（35）	说明		
1	CalcTradeDate			计算指定合约指定时间戳所属的交易日		
2	Q_AskPrice	Q_AskPrice	Q_AskPrice	最新卖价		
3	Q_AskVol	Q_AskVol	Q_AskVol	最新申卖量		
4	Q_AvgPrice	Q_AvgPrice	Q_AvgPrice	实时均价	AVPRICE SETTLE	
5	Q_BidPrice	Q_BidPrice	Q_BidPrice	最新买盘价格		
6	Q_BidVol	Q_BidVol	Q_BidVol	最新申买量		
7	Q_Close	Q_Close	Q_Close	最新价或收盘价		
8	Q_Delta			当日期权 Delta		
9	Q_Gamma			当日期权 Gamma		
10	Q_High	Q_High	Q_High	当日最高价		
11	Q_HisHigh	Q_HisHigh	Q_HisHigh	历史最高价		
12	Q_HisLow	Q_HisLow	Q_HisLow	历史最低价		
13	Q_Last		Q_Last	最新价		
14	Q_LastDate		Q_LastDate	最新成交日期		
15	Q_LastTime		Q_LastTime	最新成交时间		
16	Q_Low	Q_Low	Q_Low	当日最低价		
17	Q_LowerLimit	Q_LowerLimit	Q_LowerLimit	当日跌停板价		
18	Q_Open	Q_Open	Q_Open	当日开盘价		
19	Q_OpenInt	Q_OpenInt	Q_OpenInt	持仓量		
20	Q_PreOpenInt	Q_PreOpenInt	Q_PreOpenInt	昨日持仓量		
21	Q_PreSettlePrice	Q_PreSettlePrice	Q_PreSettlePrice	昨日结算价		
22	Q_PriceChg	Q_PriceChg	Q_PriceChg	当日涨跌		
23	Q_PriceChgRatio	Q_PriceChgRatio	Q_PriceChgRatio	当日涨跌幅		
24	Q_Rho			当日期权 Rho		
25	Q_Sigma			当日期权波动率		
26	Q_TheoryPrice			当日期权理论价		
27	Q_Theta			当日期权 Theta		
28	Q_TotalVol	Q_TotalVol	Q_TotalVol	当日成交量		
29	Q_TurnOver	Q_TurnOver	Q_TurnOver	成交金额		

30	Q_UpdateTime			行情更新时间		
31	Q_UpperLimit	Q_UpperLimit	Q_UpperLimit	当日涨停板价		
32	Q_Vega			当日期权 Vega		
33	QuoteDataExist		QuoteDataExist	行情数据是否有效		
34			Q_AskPriceFlag	卖盘价格变化标志		
35			Q_BidPriceFlag	买盘价格变化标志		
36			Q_InsideVol	商品的内盘		
37			Q_LastFlag	最新价变化标志		
38			Q_LastVol	商品的现手		
39			Q_OpenIntFlag	持仓量变化标志		
40			Q_Oscillation	商品的振幅		
41			Q_OutsideVol	商品的外盘		
42			Q_TickChg	最新笔升跌		
43			Q_TodayEntryVol	当日开仓量		
44			Q_TodayExitVol	当日平仓量		
属性函数						
编号	equant(28)	ETL （18）	TB（26）	说明		
1	BarInterval	BarInterval	BarInterval	当前图表的周期值		
2	BarType	BarType	BarType	当前图表的周期类型		
3	BidAskSize		BidAskSize	当前公式应用商品数据的买卖盘个数		
4	CommodityStatu	SymbolStatus		交易状态		
5	ContractUnit	ContractUnit	ContractUnit	每张合约的单位数量		
6	CurrentDate			获取当前日期		
7	CurrentTime			获取当前时间		
8	ExchangeName	ExchangeName	ExchangeName	商品的交易所名称		
9	ExchangeStatus	ExchangeStatus		交易所的状态信息		
10	ExchangeTime	ExchangeTime		交易所的时间信息		
11	GetNextTimeInfo			获取下一个时间点信息		
12	GetSessionCount			交易时段个数		
13	GetSessionEndTime			交易时段结束时间		
14	GetSessionStartTime			交易时段起始时间		
15	GetTrendContract			获取主连/近月对应的合约		
16	IsInSession			当前时间是否为交易时间		
17	MarginRatio	SeatMargin		合约的保证金率		
18	MaxBarsBack		MaxBarsBack	获得公式应用所需的最大回溯 Bar 数		
19	MaxSingleTradeSize		MaxSingleTradeSi	当前公式应用商品的单笔交易限量		
20	OptionType	OptionsType		买权卖权标志		
21	PriceScale	PriceScale	PriceScale	计数单位报价精确度		
22	PriceTick			最小变动价		
23	Symbol	Symbol	Symbol	当前图表商品的代码		

24	SymbolName	SymbolName	SymbolName	当前图表商品的名称		
25	SymbolType		SymbolType	当前公式应用商品的类型		
26	TimeDiff			返回两个时间之间的间隔秒数		
27	TradeSessionBeginTime			交易时段开始时间戳		
28	TradeSessionEndTime			交易时段结束时间戳		
29		BigPointValue	BigPointValue	一个点的价值		
30		ExpiredDate	ExpiredDate	最后交易日		
31		MinMove	MinMove	商品价格最小变动量		
32		ExecPrice		执行价格		
33		SymbolKind		合约类型(1 期货 2 期权)		
34		DesCommodity		期权标的合约编码		
35			CanMarketOrder	当前公式应用商品是否支持市价委托		
36			CanStopOrder	当前公式应用商品是否支持 STOP 委托		
37			CanTrade	当前公式应用商品是否支持交易		
38			Category	当前公式应用商品的大类信息		
39			ContractSize	当前商品的合约大小		
40			CurrencyName	当前公式应用商品交易的货币名称		
41			CurrencySymbol	当前公式应用商品交易的货币符号		
42			DataCount	当前公式应用图表数据源的总数		
43			GetUserID	当前登录的用户 I		
44			InitialMargin	当前公式应用商品的初始保证金		
45			MaintenanceMarg	当前公式应用商品的维持保证金		
46			MarginRatio	当前公式应用商品的默认保证金比率		
47	QuoteSvrState			获取行情服务器连接状态		
48	TradeSvrState			获取交易服务器连接状态		
交易函数						
	equant(10)	ETL （7）	TB（4）	说明		
1	Buy	Buy	Buy	产生一个多头建仓操作		
2	BuyToCover	BuyToCover	BuyToCover	产生一个空头平仓操作		
3	IsTradeAllowed			是否允许实盘交易		
4	Sell	Sell	Sell	产生一个多头平仓操作		
5	SellShort	SellShort	SellShort	产生一个空头建仓操作		
6	SetFloatStopPoin	SetFloatStopPoint		设置浮动止损		
7	SetStopPoint	SetStopPoint		设置止损		
8	SetWinPoint	SetWinPoint		设置止盈		
9	StartTrade			开启实盘交易		
10	StopTrade			暂停实盘交易		
11	UnloadStrategy			停止策略		
12	ReloadStrategy			暂停实盘交易		

策略状态						
编号	equant(30)	ETL （20）	TB（22）	说明		
1	AvgEntryPrice	AvgEntryPrice	AvgEntryPrice	获得当前持仓的平均建仓价格		
2	BarsLast			获取最后一次满足条件时距离当前的 bar 数		
3	BarsSinceEntry	BarsSinceEntry	BarsSinceEntry	获得当前持仓的第一个建仓位置到当前位置的 Bar		
4	BarsSinceExit	BarsSinceExit	BarsSinceExit	获得最近平仓位置到当前位置的 Bar 计数		
5	BarsSinceLastBuyEntry			当前持仓的最后一个 Buy 建仓位置到当前位置的 Bar		
6	BarsSinceLastEntr	BarsSinceLastEn	BarsSinceLastEntry	获得当前持仓的最后一个建仓位置到当前位置的 Ba		
7	BarsSinceLastSellEntry			当前持仓的最后一个 Sell 建仓位置到当前位置的 Bar		
8	BarsSinceToday			当天的第一根 Bar 到当前的 Bar 个数		
9	BuyPosition			当前持仓的买入持仓量		
10	ContractProfit	ContractProfit	ContractProfit	获得当前持仓的每手浮动盈亏		
11	CurrentContracts	CurrentContract	CurrentContracts	获得当前持仓的持仓合约数		
12	EntryDate	EntryDate	EntryDate	获得当前持仓的第一个建仓位置的日期		
13	EntryPrice	EntryPrice	EntryPrice	获得当前持仓的第一个建仓价格		
14	EntryTime	EntryTime	EntryTime	获得当前持仓的第一个建仓位置的时间		
15	ExitDate	ExitDate	ExitDate	获得最近平仓位置 Bar 日期		
16	ExitPrice	ExitPrice	ExitPrice	获得最近平仓位置的平仓价格		
17	ExitTime	ExitTime	ExitTime	获得最近平仓位置 Bar 时间		
18	HighestSinceLastBuyEntry			当前 Buy 持仓的最后一个建仓以来的最高价		
19	HighestSinceLastSellEntry			当前 Sell 持仓的最后一个建仓以来的最高价		
20	LastBuyEntryPrice			当前 Buy 持仓的最后一个建仓价格		
21	LastEntryDate	LastEntryDate	LastEntryDate	获得当前持仓的最后一个建仓位置的日期		
22	LastEntryPrice	LastEntryPrice	LastEntryPrice	获得当前持仓的最后一个建仓价格		
23	LastEntryTime	LastEntryTime	LastEntryTime	获得当前持仓的最后一个建仓位置的时间		
24	LastSellEntryPrice			当前 Sell 持仓的最后一个建仓价格		
25	LowestSinceLastBuyEntry			当前 Buy 持仓的最后一个建仓以来的最低价		
26	LowestSinceLastSellEntry			当前 Sell 持仓的最后一个建仓以来的最低价		
27	MarketPosition	MarketPosition	MarketPosition	获得当前持仓状态		
28	PositionProfit	PositionProfit	PositionProfit	获得当前持仓的浮动盈亏		
29	SellPosition			当前持仓的卖出持仓量		
30	StrategyId			获取当前策略 Id		
31		CurrentEntries	CurrentEntries	获得当前持仓的建仓次数		
32		MaxPositionLoss	MaxPositionLoss	获得当前持仓的最大浮动亏损数		
33		MaxPositionProf	MaxPositionProfit	获得当前持仓的最大浮动盈利数		
34			MaxContracts	获得当前持仓的最大持仓合约数		
35			MaxPositionProfit	获得当前持仓的最大持仓合约数		
策略性能						
编号	equant(14)	ETL（16）	TB（19）			
1	Available	Available		获得策略当前的可用虚拟资金		

2	CurrentEquity			账户权益		
3	FloatProfit	FloatProfit		获得策略当前浮动盈亏		
4	GrossLoss	GrossLoss	GrossLoss	获得累计的总亏损		
5	GrossProfit	GrossProfit	GrossProfit	获得累计的总盈利		
6	Margin			持仓保证金		
9	NetProfit	NetProfit	NetProfit	获得策略当前平仓盈亏		
10	NumAllTimes	NumAllTimes		获得开仓次数		
11	NumEventTimes	NumEventTimes		获得保本次数		
12	NumLoseTimes	NumLoseTimes		获得亏损次数		
13	NumWinTimes	NumWinTimes		获得盈利次数		
14	PercentProfit	PercentProfit	PercentProfit	获得盈利的成功率		
15	TotalTrades	TotalTrades		获得交易的总开仓手数		
16	TradeCost	TradeCost		获得交易产生的手续费		
17		NumEventTrade	NumEventTrades	获得保本交易的总手数		
18		NumLoseTrades	NumLoseTrades	获得亏损交易的总手数		
19		NumWinTrades	NumWinTrades	获得盈利交易的总手数		
20			AvgBarsEvenTrade	获得保本交易的平均持仓 Bar 数		
21			AvgBarsLosTrade	获得亏损交易的平均持仓 Bar 数		
22			AvgBarsWinTrade	获得盈利交易的平均持仓 Bar 数		
23			LargestLosTrade	获得最大单次交易亏损数		
24			LargestWinTrade	获得最大单次交易盈利数		
25			MaxConsecLosers	获得最大连续亏损交易手数		
26			MaxConsecWinner	获得最大连续盈利交易手数		
27			MaxContractsHeld	获得最大的持仓合约数		
28			TotalBarsEvenTrad	获得保本交易的总持仓 Bar 数		
29			TotalBarsLosTrade	获得亏损交易的总持仓 Bar 数		
30			TotalBarsWinTrad	获得盈利交易的总持仓 Bar 数		

账户函数

编号	equant(48)	ETL（49）	TB（45）	说明		
1	A_AccountID	A_AccountID	A_AccountID	当前交易账户 ID		
2	A_AllAccountID	A_GetAccountNames		策略启动发单时勾选的账户数组		
3	A_AllOrderNo			所有订单号		
4	A_AllQueueOrderNo			当前账户所有排队(可撤)订单		
5	A_Assets	A_Assets	A_CurrentEquity	当前账户的权益		
6	A_Available	A_Available	A_FreeMargin	当前账户的可用资金		
7	A_BuyAvgPrice	A_BuyAvgPrice	A_BuyAvgPrice	当前账户买仓均价		
8	A_BuyPosition	A_BuyPosition	A_BuyPosition	当前账户买仓总量		
9	A_BuyPositionCa	A_BuyPositionCanCover		当前账户买仓可平数量		
10	A_BuyProfitLoss		A_BuyProfitLoss	买浮动盈亏		
11	A_Cost	A_Cost		当前账户的手续费		

12	A_CoverProfit	A_CoverProfit		当前账户的平仓盈亏		
13	A_DeleteOrder	A_DeleteOrder	A_DeleteOrder	发送撤单指令		
14	A_FirstOrderNo	A_FirstOrderNo		当前账户第一个单子		
15	A_FirstQueueOrd	A_FirstQueueOrderNo		当前账户第一个排队单子		
16	A_GetAllPositionSymbol			所有持仓合约		
17	A_GetOrderNo			获取定单号和委托号		
18	A_LastOrderNo	A_LastOrderNo	A_GetLastOrderIn	当前账户最近一个单子		
19	A_LatestFilledTime			当前账户最新一笔成交委托对应的时间		
20	A_Margin	A_Margin		当前账户的持仓保证金		
21	A_ModifyOrder			改单		
22	A_NextOrderNo	A_NextOrderNo		当前账户下一个单子		
23	A_NextQueueOr	A_NextQueueOrderNo		当前账户下一个排队单子		
24	A_OrderBuyOrSell	A_OrderBuyOrS	A_OrderBuyOrSell	订单买卖类型		
25	A_OrderContract	A_OrderContrac	A_OrderContractN	订单合约号		
26	A_OrderEntryOrE	A_OrderEntryOr	A_OrderEntryOrExi	订单开平类型		
27	A_OrderFilledLot	A_OrderFilledLot	A_OrderFilledLot	订单成交量		
28	A_OrderFilledPric	A_OrderFilledPri	A_OrderFilledPrice	订单成交价格		
29	A_OrderIsClose			定单是否完结		
30	A_OrderLot	A_OrderLot	A_OrderLot	订单委托数量		
31	A_OrderPrice	A_OrderPrice	A_OrderPrice	订单委托价格		
32	A_OrderStatus	A_OrderStatus	A_OrderStatus	查询订单状态		
33	A_OrderTime	A_OrderTime	A_OrderTime	订单下单时间		
34	A_ProfitLoss		A_ProfitLoss	交易帐户的浮动盈亏		
35	A_SellAvgPrice	A_SellAvgPrice	A_SellAvgPrice	当前账户卖仓均价		
36	A_SellPosition	A_SellPosition	A_SellPosition	当前账户卖仓总量		
37	A_SellPositionCan	A_SellPositionCanCover		当前账户空仓可平数量		
38	A_SellProfitLoss		A_SellProfitLoss	卖浮动盈亏		
39	A_SendOrder	A_SendOrder	A_SendOrder	针对当前账户发送委托单		
40	A_TodayBuyPositi	A_TodayBuyPosi	A_TodayBuyPositi	当前账户当日开买仓量		
41	A_TodaySellPositi	A_TodaySellPosi	A_TodaySellPositi	当前账户当日开卖仓量		
42	A_TotalAvgPrice	A_TotalAvgPrice	A_TotalAvgPrice	当前账户总持仓均价		
43	A_TotalFreeze			冻结资金		
44	A_TotalPosition	A_TotalPosition	A_TotalPosition	当前账户总持仓量		
45	A_TotalProfitLoss		A_PositionProfitLo	总浮动盈亏		
46	DeleteAllOrders	DeleteAllOrders		批量撤单函数		
47	StarTrade	StarTrade		开启交易		
48	StopTrade	StopTrade		暂停交易		
49		A_HoldProfit		当前账户的持仓盈亏		
50		A_IsConnected		判断是否连接交易服务器并登录成功		
51		A_LastQueueOr	A_GetLastOpenOr	当前账户最近一个排队单子		
52		A_OrderContractNo2		第二腿订单合约号		

53		A_OrderWay		订单的来源		
54		A_StrategyPosition		获取当前账户策略仓的数量		
55		A_OrderAccount		订单的用户账号		
56		A_SetCurrentAccount		改变当前策略的主账户		
57		A_Execute		发送期权行权指令		
58		A_Abandon		发送期权弃权指令		
59		A_Enquiry		发送期权询价指令		
60		A_Offer		发送期权应价指令		
61		SetOrderFlag		设置订单标识		
62			A_BrokerID	交易帐户对应的交易商 ID		
63			A_GetOpenOrder	未成交委托单数量		
64			A_GetOrderCount	当日委托单数量		
65			A_OpenOrderBuy	未成交委托单的买卖类型		
66			A_OpenOrderCont	委托单的合同号		
67			A_OpenOrderEntr	未成交委托单的开平仓状态		
68			A_OpenOrderFille	未成交委托单的成交数量		
69			A_OpenOrderFille	未成交委托单的成交价格		
70			A_OpenOrderLot	未成交委托单的委托数量		
71			A_OpenOrderPrice	未成交委托单的委托价格		
72			A_OpenOrderStat	未成交委托单的状态		
73			A_OpenOrderTime	未成交委托单的委托时间		
74			A_OrderCanceledL	委托单的撤单数量		
75			A_PreviousEquity	交易帐户的昨日结存		
76			A_TodayDeposit	交易帐户的当日入金		
77			A_TodayDrawing	交易帐户的当日出金		
78			AccountDataExist	帐户数据是否有效		
金融及统计函数						
编号	equant(10)	ETL（58）	TB（72）	说明		
1		AdaptiveMovAv	AdaptiveMovAvg	求卡夫曼自适应移动平均		
2		Average /MA	Average /MA	求平均		
3		AverageFC	AverageFC	快速计算平均值		
4		AvgPrice	AvgPrice	求平均价格		
5		AverageD	AverageD	求 N 天以来的价格平均值		
6		AvgDeviation	AvgDeviation	求平均背离		
7		AvgTrueRange	AvgTrueRange	求平均真实范围		
8		BarsSinceToday	BarsSinceToday	当天的第一个数据到当前的 Bar 数		
9		CoefficientR	CoefficientR	求皮尔森相关系数		
10	CountIf	CountIf	CountIf	获取最近 N 周期条件满足的计数		
11		Correlation	Correlation	求相关系数		
12		Covar	Covar	求协方差		

13	CrossOver	CrossOver	CrossOver	求是否上穿		
14	CrossUnder	CrossUnder	CrossUnder	求是否下破		
15		Cum	Cum	求累计值		
16		CloseD	CloseD	求 N 天前的收盘价		
17		DEMA	DEMA	求双指数移动平均		
18		Detrend	Detrend	求趋势平滑		
19		Extremes	Extremes	求极值		
20	Highest	Highest	Highest	求最高		
21		HighestFC	HighestFC	快速计算最高		
22		HighestBar	HighestBar	求最高值出现的 Bar		
23		HighD	HighD	求 N 天前的最高价		
24		LinearReg	LinearReg	求线性回归		
25	Lowest	Lowest	Lowest	求最低		
26		LowestFC	LowestFC	快速计算最低		
27		LowestBar	LowestBar	求最低只出现的 Bar		
28		LowD	LowD	求 N 天前的最低价		
29		NthCon	NthCon	第 N 个满足条件的 Bar 距当前的 Bar 数目		
30		NthExtremes	NthExtremes	求 N 极值		
31		NthHigher	NthHigher	求第 N 高		
32		NthHigherBar	NthHigherBar	求第 N 高出现的 Bar		
33		NthLower	NthLower	求第 N 低		
34		NthLowerBar	NthLowerBar	求第 N 低出现的 Bar		
35		OpenD	OpenD	求 N 天前的开盘价		
36		OpenIntD	OpenIntD	求 N 天前的持仓量		
37	ParabolicSAR	ParabolicSAR	ParabolicSAR	求抛物线转向		
38		PercentChange	PercentChange	求涨跌幅		
39		PercentR	PercentR	求威廉指标		
40		Pivot	Pivot	求转折		
41		RateOfChange	RateOfChange	求变动率		
42	SMA	SMA	SMA	求移动平均		
43		StandardDev	StandardDev	求标准差		
44		Summation	Summation	求和		
45		SummationFC	SummationFC	快速求和		
46	SwingHigh	SwingHigh	SwingHigh	求波峰点		
47	SwingLow	SwingLow	SwingLow	求波谷点		
48		TrueHigh	TrueHigh	求真实高点		
49		TrueLow	TrueLow	求真实低点		
50		TrueRange	TrueRange	求真实范围		
51		VariancePS	VariancePS	求估计方差		
52		VolD	VolD	求 N 天前的成交量		
53		WAverage	WAverage	求权重平均		

54		XAverage	XAverage	求指数平均		
55	REF	REF		数据回溯		
56		MACD		求指数平滑异同平均		
57		EMA		求指数平均		
58		RSI		相对强弱指标值		
59			DataConvert	跨周期数据转换函数		
60			DevSqrd	求偏差均方和		
61			Fisher	求 Fisher 变换		
62			FisherInv	求反 Fisher 变换		
63			HarmonicMean	求调和平均数		
64			HighestBarFC	求最高值出现的 Bar(快速计算版本)		
65			LowestBarFC	求最低值出现的 Bar(快速计算版本)		
66			Kurtosis	求峰度系数		
67			LinearRegAngle	求线性回归角度		
68			LinearRegSlope	求线性回归斜率		
69			LinearRegValue	求线性回归值		
70			Permutation	求排列		
71			PriceOscillator	求振荡		
72			SAverage	求平滑平均		
73			Skewness	求偏度系数		
74			SwingHighBar	求波峰点出现的 Bar		
75			SwingLowBar	求波谷点出现的 Bar		
76			TrueDate	求指定 bar 的真正交易日期		
字符串函数						
编号	eauant(0)	ETL（11）	TB（10）	说明		
1		Exact	Exact	判断两字符串是否完全相等区分大小写		
2		Left	Left	前 ICount 长度的字符串		
3		Len	Len	字符串的长度		
4		Lower	Lower	转化为小写		
5		Mid	Mid	从 IFirst 开始共 ICount 个字符组成的字符串		
6		Right	Right	字符串右端指定长度的字符串		
7		Text	Text	将数字转化为字符串		
8		Trim	Trim	去除字符串两端的空格		
9		Upper	Upper	转为大写		
10		Value	Value	将字符串转化为数字		
11		FindSubString		查找子字符串		
时间函数						
	equant(0)	ETL（38）	TB（37）	说明		

1		CurrentTime	CurrentTime	当前时间		
2		CurrentDate	CurrentDate	当前日期		
3		DateAdd	DateAdd	增加指定天数后的日期		
4		DateDiff	DateDiff	两个日期之间的日期间隔		
5		DateTimeToStri	DateTimeToString	将日期时间值转化为字符串类型		
6		DateToString	DateToString	将日期值转化为字符串类型		
7		Day	Day	获得当前 Bar 的日信息		
8		Friday	Friday	获得星期五的值		
9		Hour	Hour	获得当前 Bar 的小时信息		
10		HourFromDateT	HourFromDateTim	获取输入日期时间的小时信息		
11		MakeDate	MakeDate	将参数生成日期值		
12		MakeDateTime	MakeDateTime	将参数生成日期时间值		
13		MakeTime	MakeTime	将参数生成时间值		
14		Minute	Minute	获得当前 Bar 的分钟信息		
15		MinuteFromDat	MinuteFromDateTi	获取输入日期时间的分钟信息		
16		Monday	Monday	获得星期一的值		
17		Month	Month	获得当前 Bar 的月信息		
18		MonthFromDat	MonthFromDateTi	获取输入日期时间的月信息		
19		Saturday	Saturday	获得星期六的值		
20		Second	Second	获得当前 Bar 的秒信息		
21		SecondFromDat	SecondFromDate	获取输入日期时间的秒信息		
22		StringToDate	StringToDate	将字符串 YYYY-MM-DD 转化为日期		
23		StringToDateTi	StringToDateTime	将字符串 YYYY-MM-DDHH:MM:SS 转化为日期时间		
24		StringToTime	StringToTime	将字符串 HH:MM:SS 转化为时间		
25		Sunday	Sunday	获得星期日的值		
26		SystemDateTim	SystemDateTime	获取交易平台的当前日期时间		
27		Thursday	Thursday	获取星期四的值		
28		TimeDiff	TimeDiff	两个日期之间的间隔秒数		
29		TimeToString	TimeToString	将时间值转化为字符串类型		
30		Tuesday	Tuesday	获取星期二的值		
31		Wednesday	Wednesday	获取星期三的值		
32		Weekday	Weekday	获得当前 Bar 的周信息		
33		WeekdayFromD	WeekdayFromDat	获取输入日期时间的周信息		
34		Year	Year	获得当前 Bar 的年信息		
35		YearFromDateTi	YearFromDateTim	获取输入日期时间的年信息		
36		GetTickCount		获取客户端自开机以来经历的毫秒数		
37		DayFromDateTime		获取输入日期时间的日信息		
38		TimeAdd		增加指定秒数后的日期		
39			MilliSecond	获得当前 Bar 的毫秒信息		
40			MilliSecondFromD	获取输入日期时间的毫秒信息		

数学函数						
编号	equant(0)	ETL（32）	TB（43）	说明		
1		Abs	Abs	参数的绝对值		
2		Acos	Acos	数字的反余弦值		
3		Asin	Asin	参数的反正弦值		
4		Atan	Atan	参数的反正切值		
5		Atan2	Atan2	给定的 X 及 Y 坐标值的反正切值		
6		Combin	Combin	计算从给定数目的对象集合中提取若干对象的组合		
7		Cos	Cos	给定角度的余弦值		
8		Cosh	Cosh	参数的双曲余弦值		
9		Ctan	Ctan	给定角度的余切值		
10		Exp	Exp	e 的 n 次幂		
11		Fact	Fact	数的阶乘		
12		Floor	Floor	不大于 num 的最大整数		
13		FracPart	FracPart	一个数的小数部分		
14		IntPart	IntPart	一个数的整数部分		
15		Ln	Ln	一个数的自然对数		
16		Log	Log	按所指定的底数一个数的对数		
17		Max	Max	两数中较大的一个		
18		Min	Min	两数中较小的一个		
19		Mod	Mod	两数相除的余数		
20		Neg	Neg	参数的负绝对值		
21		Pi	Pi	数字 3.1415926535898		
22		Power	Power	给定数字的乘幂		
23		Rand	Rand	位于两个指定数之间的一个随机整数		
24		Round	Round	某个数字按指定位数舍入后的数字		
25		Sign	Sign	数字的符号 1:正数-1:负数 0:零		
26		Sin	Sin	给定角度的正切值		
27		Sinh	Sinh	某一数字的双曲正弦值		
28		Sqr	Sqr	参数的平方		
29		Sqrt	Sqrt	正平方根		
30		Tan	Tan	给定角度的正切值		
31		Tanh	Tanh	某一数字的双曲正切值		
32		Ceil		不小于 num 的最小整数		
33			Ceiling	将参数 Number 沿绝对值增大的方向舍入为最接近		
34			Acosh	参数的反双曲余弦值		
35			Asinh	参数的反双曲正弦值		
36			Atanh	参数的反双曲正切值		
37			Even	沿绝对值增大方向取整后最接近的偶数		
38			Odd	对指定数值进行舍入后的奇数		
39			RoundDown	靠近零值向下（绝对值减小的方向）舍入数字		

40			RoundUp	远离零值向上（绝对值增大的方向）舍入数字		
41			Median	求中位数		
42			MidPoint	求中点		
43			Mode	求众数		
44			Momentum	求动量		
颜色函数						
编号	equant(7)	ETL（19）	TB（17）	说明		
1	RGB_Blue	Blue	Blue	蓝色		
2	RGB_Brown	DarkBrown	DarkBrown	茶色		
3	RGB_Gray	DarkGray	DarkGray	深灰色		
4	RGB_Green	Green	Green	绿色		
5	RGB_Purple			紫色		
6	RGB_Red	Red	Red	红色		
7	RGB_Yellow	Yellow	Yellow	黄色		
8		DefaultColor	DefaultColor	默认颜色(红色)		
9		Rgb	Rgb	自定义颜色		
10		Black	Black	黑色		
11		Cyan	Cyan	青色		
12		DarkCyan	DarkCyan	深青色		
13		DarkGreen	DarkGreen	深绿色		
14		DarkMagenta	DarkMagenta	深褐色		
15		DarkRed	DarkRed	深红色		
16		LightGray	LightGray	浅灰色		
17		Magenta	Magenta	褐色		
18		White	White	白色		
19		ColorUp		系统上涨色		
20		ColorDown		系统下跌色		
枚举函数						
编号	equant(58)	ETL（58）	TB（12）	说明		
1	Enum_Buy	Enum_Buy	Enum_Buy	买卖类型_买入		
2	Enum_Sell	Enum_Sell	Enum_Sell	买卖类型_卖出		
3	Enum_Entry	Enum_Entry	Enum_Entry	开平类型_开仓		
4	Enum_Exit	Enum_Exit	Enum_Exit	开平类型_平仓		
5	Enum_ExitToday	Enum_ExitToday	Enum_ExitToday	开平类型_平今		
6	Enum_EntryExitIgnore			开平类型_不区分开平		
7	Enum_Data_Close	Enum_Data_Close		收盘价枚举值		
8	Enum_Data_Ope	Enum_Data_Open		开盘价枚举值		
9	Enum_Data_High	Enum_Data_High		最高价枚举值		
10	Enum_Data_Low	Enum_Data_Low		最低价枚举值		

11	Enum_Data_Medi	Enum_Data_Median		中间价枚举值(高+低)/2		
12	Enum_Data_Typic	Enum_Data_Typical		标准价枚举值(高+低+收)/3		
13	Enum_Data_Weig	Enum_Data_Weighted		加权收盘价枚举值(高+低+开+收)/4		
14	Enum_Data_Vol	Enum_Data_Vol		成交量枚举值		
15	Enum_Data_Opi	Enum_Data_Opi		持仓量枚举值		
16	Enum_Data_Time	Enum_Data_Time		K 线时间枚举值		
17		Enum_Period_Default		周期类型_当前图表周期		
18	Enum_Period_Tic	Enum_Period_Tick		周期类型_分笔		
19		Enum_Period_Second		周期类型_秒线		
20		Enum_Period_SecondX		周期类型_多秒		
21	Enum_Period_Min			周期类型_分钟		
22		Enum_Period_Min1		周期类型_1 分钟		
23		Enum_Period_Min3		周期类型_3 分钟		
24		Enum_Period_Min5		周期类型_5 分钟		
25		Enum_Period_Min15		周期类型_15 分钟		
26		Enum_Period_Min30		周期类型_30 分钟		
27		Enum_Period_Min60		周期类型_60 分钟		
28		Enum_Period_Min120		周期类型_102 分钟		
29		Enum_Period_Min240		周期类型_240 分钟		
30		Enum_Period_MinX		周期类型_多分钟		
31	Enum_Period_Da	Enum_Period_Day		周期类型_日线		
32		Enum_Period_Week		周期类型_周线		
33		Enum_Period_Month		周期类型_月线		
34		Enum_Period_Year		周期类型_年线		
35		Enum_Period_DayX		周期类型_多日线		
36		Enum_OW_All		所有		
37		Enum_OW_ETrade		E-Trade		
38		Enum_OW_ProxyETrade		代理单		
39		Enum_OW_JTrade		J-Trade		
40		Enum_OW_Manual		人工单		
41		Enum_OW_Carry		Carry 单		
42		Enum_OW_Delivery		交割行权		
43		Enum_OW_Program		程式单		
44		Enum_OW_Execute		行权		
45		Enum_OW_Abandon		弃权		
46		Enum_OW_Channel		通道费		
47	Enum_Order_Market			订单类型_市价单		
48	Enum_Order_Limit			订单类型_限价单		
49	Enum_Order_MarketStop			订单类型_市价止损单		
50	Enum_Order_LimitStop			订单类型_限价止损单		
51	Enum_Order_Execute			订单类型_行权单		

52	Enum_Order_Abandon			订单类型_弃权单		
53	Enum_Order_Enquiry			订单类型_询价单		
54	Enum_Order_Offer			订单类型_应价单		
55	Enum_Order_Iceberg			订单类型_冰山单		
56	Enum_Order_Ghost			订单类型_影子单		
57	Enum_Order_Swap			订单类型_互换		
58	Enum_Order_SpreadApply			订单类型_套利申请		
59	Enum_Order_HedgApply			订单类型_套保申请		
60	Enum_Order_OptionAutoClose			订单类型_行权前期权自对冲申请		
61	Enum_Order_FutureAutoClose			订单类型_履约期货自对冲申请		
62	Enum_Order_MarketOptionKeep			订单类型_做市商留仓		
63	Enum_GFD			订单有效类型_当日有效		
64	Enum_GTC			订单有效类型_当日有效		
65	Enum_GTD			订单有效类型_限期有效		
66	Enum_IOC			订单有效类型_即时部分		
67	Enum_FOK			订单有效类型_即时全部		
68	Enum_Speculate			订单投保标记_投机		
69	Enum_Hedge			订单投保标记_套保		
70	Enum_Spread			订单投保标记_套利		
71	Enum_Market			订单投保标记_做市		
72	Enum_Sended	Enum_Sended		已发送		
73	Enum_Accept	Enum_Accept		已受理		
74	Enum_Triggering	Enum_Triggering		待触发		
75	Enum_Active	Enum_Active		已生效		
76	Enum_Queued	Enum_Queued		已排队		
77	Enum_Modifying	Enum_Modifying		待改		
78	Enum_PartCancel	Enum_PartCanceled		已撤余单		
79	Enum_Fail	Enum_Fail		指令失败		
80	Enum_Suspende	Enum_Suspended		已挂起		
81	Enum_Apply	Enum_Apply		已申请		
82	Enum_Canceling	Enum_Canceling	Enum_Canceling	待撤		
83	Enum_Canceled	Enum_Canceled	Enum_Canceled	已撤单		
84	Enum_Filled	Enum_Filled	Enum_Filled	完全成交		
85	Enum_FillPart	Enum_PartFilled	Enum_FillPart	部分成交		
86			Enum_Declare	委托状态的正在申报枚举值		
87			Enum_Declared	委托状态的已申报枚举值		
88			Enum_Deleted	委托状态的已废除枚举值		
绘图函数						
编号	equant(16)	ETL（16）	TB（4）	说明		
1	PlotBar	PlotBar		绘制柱子		

2	PlotDot	PlotDot		绘制一个点		
3	PlotIcon	PlotIcon		绘制一个系统图标		
4	PlotNumeric	PlotNumeric	PlotNumeric	绘制指标线		
5	PlotPartLine	PartLine		绘制斜线段		
6	PlotStickLine	PlotStickLine		绘制一条竖线段		
7	PlotText	PlotText	PlotString	绘制一个字符串		
8	PlotVertLine	PlotVertLine		绘制一条竖线		
9	UnPlotBar			取消绘制的柱子		
10	UnPlotDot	UnPlotDot		取消绘制的点		
11	UnPlotIcon	UnPlotIcon		取消绘制的图标		
12	UnPlotNumeric			取消绘制的指标线		
13	UnPlotPartLine	UnPlotPartLine		取消绘制的斜线段		
14	UnPlotStickLine	UnPlot		取消 PlotNumeirc 所绘制的线		
15	UnPlotText	UnPlotText		取消 PlotText 绘制的字符串		
16	UnPlotVertLine	UnPlotVertLine		取消绘制的竖线		
17		ColorBar		设置 K 线颜色变色 K 线		
18		SetOwnAxis		设置一个指标线为自由坐标		
19			UnPlot	删除曾经输出的值		
20			PlotBool	在当前 Bar 输出一个布尔值		
设置函数						
编号	equant(17)					
1	SetUserNo			设置交易账号		
2	SetBarInterval			设置 K 线类型		
3	SetInitCapital			设置初始资金		
4	SetMargin			设置保证金		
5	SetTradeFee			设置手续费		
6	SetActual			设置实盘运行		
7	SetOrderWay			设置发单方式		
8	SetTradeDirection			设置交易方向		
9	SetMinTradeQuantity			设置最小下单量		
10	SetHedge			设置投保标志		
11	SetSlippage			设置滑点损耗		
12	SetTriggerType			设置触发方式		
13	SetWinPoint			设置策略的止盈点		
14	SetStopPoint			设置策略的止损点		
15	SetFloatStopPoint			设置策略的浮动止损点		
16	SubQuote			订阅指定合约即时行情		
17	UnsubQuote			退订指定合约即时行情		
日志函数						

编号	equant(5)					
1	LogDebug			打印调试信息		
2	LogInfo			打印普通信息		
3	LogWarn			打印警告信息		
4	LogError			打印错误信息		
5	Alert			弹出警告提醒		
context 函数						
编号	equant(8)					
1	strategyStatus			获取当前策略状态		
2	triggerType			获取当前触发类型		
3	contractNo			获取当前触发合约		
4	kLineType			获取当前触发的 K 线类型		
5	kLineSlice			获取当前触发的 K 线周期		
6	tradeDate			获取当前触发的交易日		
7	dateTimeStamp			获取当前触发的时间戳		
8	triggerData			获取当前触发类型对应的数据		
其他函数						
编号	equant(0)	ETL（21）	TB（18）	编号		
1		Alert	Alert	弹出警告窗口或声音		
2			AlertEnabled	当前公式应用的报警设置		
3		Commentary		向调试窗口输出一行字符串		
4			Commentary	超级图表当前 Bar 添加一行注释信息		
5		InvalidString	InvalidString	无效字符串的值""		
6		InvalidNumeric	InvalidNumeric	无效数字的值		
7		FileAppend	FileAppend	向文件追加一行文字		
8		FileDelete	FileDelete	删除文件		
9		IIF	IIF	根据逻辑真假值不同的数值		
10		IIFString	IIFString	根据逻辑真假值不同的字符串		
11		SetGlobalVar	SetGlobalVar	设置一个全局变量		
12		GetGlobalVar	GetGlobalVar	读取一个全局变量的值		
13		Print		向调试窗口输出一个字符串或数字		
14		ValueWhen		最后一次满足条件的值		
15		Filter		条件过滤		
16		IsTradeAllowed		是否允许自动交易		
17		GetCausation		获得本次策略的触发原因		
18		GetCausationOrder		触发本次运行的订单号		
19		Sound		播放一个声音		
20			PlayWavSound	播放指定路径的 Wav 声音文件		
21		SetELProfileString		把给定的键名及其值写入到公式信息文件的相应块		

22		GetELProfileString		读取公式信息文件指定块中的键名对应的字符串		
23		SetELProfileString2File		把给定的键名及其值写入到公式信息文件的相应块		
24		GetELProfileString2File		读取公式信息文件指定块中的键名对应的字符串		
25			SetTBProfileString	把给定的键名及其值写入到公式信息文件的相应块		
26			GetTBProfileString	读取公式信息文件指定块中的键名对应的字符串		
27			SetTBProfileString	把给定的键名及其值写入到公式信息文件的相应块		
28			GetTBProfileString	读取公式信息文件指定块中的键名对应的字符串		
29			FormulaName	获得当前执行的公式名称		
30			InvalidInteger	整型的无效值		
数组函数						
编号		ETL（19）	TB（0）	说明		
1		ArrAdd		添加数据到数组末端		
2		ArrInsert		在数组指定位置插入数据		
3		ArrRemove		删除数组中某元素		
4		ArrSub		数组的子集		
5		ArrLength		获得数组的长度		
6		ArrSort		对数组进行排序		
7		ArrPrint		输出数组到控制台		
8		ArrClear		清空数组		
9		ToArray		将序列变量复制到数组		
10		ArrRevers		将数组反转		
11		ArrSetSize		设置数组的大小以 value 填充		
12		ArrFind		查找数组满足条件元素的索引值		
13		iMA		求平均		
14		iHHV		求最高		
15		iLLV		求最低		
16		iEMA		求指数平均		
17		iMACD		求指数平滑异同平均		
18		iSMA		求数组的权重平均值		
19		iRSI		求数组的相对强弱指数		
组合性能						
编号		ETL（0）	TB（9）	说明		
1			Portfolio_GrossLos	获得投资组合的毛损		
2			Portfolio_GrossPro	获得投资组合的毛利		
3			Portfolio_MaxDra	获得投资组合的最大资产回撤		
4			Portfolio_MaxDra	获得投资组合的最大资产回撤比率		
5			Portfolio_NetProfit	获得投资组合的净利润		
6			Portfolio_PercentP	获得投资组合的交易成功率		
7			Portfolio_NumWin	获得投资组合的交易盈利手数		

8			Portfolio_NumLoss	获得投资组合的交易亏损手数		
9			Portfolio_TotalTra	获得投资组合的总交易手数		
组合状态						
编号		ETL（0）	TB（7）	说明		
1			Portfolio_CurrentC	获得按当前 Bar 开盘价计算的可用资金		
2			Portfolio_CurrentE	获得投资组合的当前建仓次数		
3			Portfolio_CurrentE	获得投资组合的动态权益		
4			Portfolio_InitCapit	获得投资组合的初始资金		
5			Portfolio_PositionP	获得投资组合的浮动盈亏		
6			Portfolio_TotalProf	获得投资组合的累计交易盈亏		
7			Portfolio_UsedMar	获得当前的持仓保证金		

5.2 极智量化与麦语言函数对比

这里列出了极智量化与麦语言的 API 接口函数名的对照表：

引用数据函数	
麦语言	equant
AVPRICE	Q_AvgPrice
SETTLE	Q_AvgPrice
CLOSE	Close
C	C
HIGH	High
H	H
LOW	Low
L	L
OPEN	Open
O	O
OPI	OpenInt
REF	REF
VOL	Vol
V	V
GETPRICE(, 'OPEN')	Q_Open
GETPRICE(, 'HIGH')	Q_High
GETPRICE(, 'LOW')	Q_Low
GETPRICE(, 'CLOSE')	Q_Close
GETPRICE(, 'NEW')	Q_Last
GETPRICE(, 'AVPRICE')	Q_AvgPrice
GETPRICE(, 'SETTLE')	Q_AvgPrice
GETPRICE(, 'YCLOSE')	
GETPRICE(, 'YSETTLE')	Q_PreSettlePrice

GETPRICE(, 'BID1')	Q_BidPrice
GETPRICE(, 'BIDVOL1')	Q_BidVol
GETPRICE(, 'ASK1')	Q_AskPrice
GETPRICE(, 'ASKVOL1')	Q_AskVol
GETPRICE(, 'VOLUME')	Q_TotalVol
GETPRICE(, 'OPI')	Q_OpenInt
GETPRICE(, 'DELTAVOL')	
GETPRICE(, 'DELTAOPI')	Q_OpenInt - Q_PreOpenInt
金融统计函数	
麦语言	equant
BARSLAST	
COUNT	CountIf
DMA	
EMA	talib.EMA
EMA2	talib.WMA
HHV	talib.MAX
HHVBARS	talib.MAXINDEX
LLV	talib.MIN
LLVBARS	talib.MININDEX
MA	talib.MA
SAR	talib.SAR
SMA	talib.SMA
SUM	talib.SUM
SUMBARS	
TRMA	talib.trima
TSMA	
数学统计函数	
麦语言	equant
AVEDEV	
DEVSQ	
FORCAST	talib.LINEARREG
VAR	var
VARP	
SLOPE	talib.LINEARREG_SLOPE
STD	talib.STDDEV
STDP	
逻辑判断函数	
麦语言	equant
BETWEEN	
CROSS	CrossOver

FILTER	
EXIST	CountIf() > 0
EVERY	CountIf(N)==N
LAST	
LONGCROSS	
IFESLE(X, A, B)	x if a else b
ISDOWN	C()[-1] < O()[-1]
ISEQUAL	C()[-1]==O()[-1]
ISUP	C()[-1] > O()[-1]
VALUEWHEN	
数学运算函数	
麦语言	equant
ABS(X)	math.fabs
ACOS	math.acos
ASIN	math.asin
ATAN	math.atan
COS	math.cos
EXP	math.exp
CUBE	math.pow(x, 3)
CEILING	math.ceil
FLOOR	math.floor
INTPART	math.trunc
LN	math.log
LOG	math.log10
MAX	max
MIN	min
MOD	math.modf
NOT	not
POW	math.pow
REVERSE(X)	~x
RANGE(A, B, C0)	b > a > c
SGN(X)	1 if x >= 0 else -1
SIN	math.sin
SQRT	math.sqrt
SQUARE	math.pow(x, 2)
TAN	math.tan
时间函数	
麦语言	equant
BARPOS	CurrentBar
DATE	Date
TIME	Time

CLOSEMINUTE	
DAY	
HOUR	
MINUTE	
MONTH	
WEEKDAY	
YEAR	
绘图函数	
麦语言	equant
BACKGROUNDSTYLE	
ICON	
WORD	PlotText
SOUND	
DRAWICON	PlotIcon
DRAWLINE	
DTAWTEXT	
DRAWSL	PlotPartLine
DRAWKLING	
DRAWNUMBER	PlotNumeric
FILLRGN	
PLAYSOUND	
KTEXT	PlotText
POLYLINE	
PARTLINE	
STICKLINE	
VERTLINE	PlotVertLine
画线函数	
麦语言	equant
ANGLELING	
GOLDENLINE	
HORIZONTALLINE	
TRENDLINES	
WAVERULER	
DRAWANGLELING	
DRAWGOLDENLINE	
DRAWHORIZONTALLINE	
DRAWTRENDLINE	
DRAWWAVERULER	
WAVEPEAK	
WAVEVALLEY	

颜色函数	
麦语言	equant
COLORRED	RGB_Red
COLORGREEN	RGB_Green
COLORBLUE	RGB_Blue
COLORMAGENTA	RGB_Purple
COLORYELLOW	RGB_Yellow
COLORLIGHTGREY	RGB_Gray
COLORLIGHTERD	0xff8080
COLORLIGHGREEN	0x80ff80
COLORLIGHBLUE	0x8080ff
COLORBLACK	0x000000
COLORWHITE	0xffffff
COLORCYAN	0x00ffff
未来函数	
麦语言	equant
BACKSET	
REFX	
PEAK	SwingHigh
PEAKBARS	
TROUGH	SwingLow
TROUGHBARS	
ZIGZAG	
ISLASTBAR	BarStatus()==2
#IMPORT [] AS	
头寸函数	
麦语言	equant
SETDEALPERCENT	
SETDEALVOL	
信号记录函数	
麦语言	equant
BARSBK	BarsSinceLastBuyEntry
BARSSK	BarsSinceLastSellEntry
BKPRICE	LastBuyEntryPrice
SKPRICE	LastSellEntryPrice
交易指令	
麦语言	equant
BK	Buy(needCover=False,)
BP	BuyToCover

SK	SellShort(needCover=False,)
SP	Sell
BPK	Buy
SPK	SellShort
BUY	Buy
SELL	Sell
BKSK	Buy(needCover=False,)
SKBK	SellShort(needCover=False,)
BPSP	BuyToCover
SPBP	Sell

6.C 版极智量化与 python 版本极智量化对比

C 版极智量化是由 C 语言实现的量化框架，在底层运行机制和 python 版本保持一致的同时，具有更加稳定、运行速度更快、内存占用更少的优点。C 版极智量化兼容 python 版本的策略，且 C 版极智量化策略开发仍然是用 python 语言作为策略开发语言，用户可以使用在 python 版本上开发的策略在 C 版上运行。C 版量化相对于 python 版本的量化也进行了相应的改动，用户在使用时需要注意。下面列举了 C 版和 python 版量化的具体不同之处。

- 新增函数接口：
  1） 策略交易函数分类中新增 UnloadStrategy 和 ReloadStrategy 函数，UnloadStrategy 用于停止策略，ReloadStrategy 用于重新启动策略；
  2） 属性函数分类中新增 TradeSvrState 和 QuoteSvrState 函数用于获取交易服务器连接状态和行情服务器连接状态
- 新增触发方式：

1. 新增连接状态'N'触发

连接状态触发指的是交易服务器和行情服务器的连接状态。当这两个服务器的连接状态发生改变时，如果订阅连接状态触发，策略会被触发，此时 context.triggerType() == 'N'（注意：只有通过 SetUserNo 设置的账号或在界面上通过关联账号设置的账号的连接状态改变才会触发策略，与策略无关的账号的状态改变不会触发策略）；

2. 新增市场状态'Z'触发

市场状态触发指的是各个交易所的状态发生改变时会触发策略。市场状态触发只有与策略相关的交易所或品种的状态改变时才会触发策略。由于市场状态属于交易数据，用户需要登录交易账户，并订阅交易数据触发，策略才会被触发，此时 context.triggerType()=='Z';

- 触发方式修改：

C 版本与 python 版本的多合约数据触发相比，C 版本除了基准合约数据触发策略外，其他合约的数据能正常获取，但是不再提供除基准合约外的其他合约的数据触发策略的支持。Python 版本的是所有合约的数据都会触发策略，这点用户需要注意！

- 接口修改：

【设置函数】

1. C 版的 SetBarInterval 函数新增 barDataLen 引用根数参数，详见策略编辑界面的函数说明；

C 版的 SetBarInterval 选择不执行历史 K 线，一根历史 K 线也不执行，原来 python 版本是至少执行一根 K 线；

C 版的 SetBarInterval 最多支持订阅 10 个合约；

2. C 版的 SetUserNo 函数可一次设置多个账号（上限 10 个），不可重复调用；

3. C 版的 SetTriggerType 函数新增订阅连接状态触发；

C 版的 SetTriggerType 函数当订阅交易数据触发时，只有与策略相关联的合约的交易数据的变化才会触发策略；python 版本的交易数据触发是任何合约的交易数据都会触发策略；

4. C 版的 SubQuote 函数只能在 initialize 函数中订阅合约，且最多支持订阅十个合约的即时行情；python 版本可以动态订阅合约；

5. C 版的 UnSubQuote 函数可忽略，策略运行中不需处理，策略停止时自动退订；

6. C 版的设置函数中除止损止盈相关函数外，其余函数都只能在 initialize 函数中调用，否则会报错；

【K 线函数】

1. C 版的 Open、Close、High、Low、Vol、OpenInt 等 K 线数据函数能取到的最大数据长度为用 SetBarInterval 函数订阅合约时的 barDataLen 参数的设置值，或是通过界面设置添加合约时在"引用根数"处设置的值，python 版本取到的是全量的 K 线数据。此处修改时为了提高访问数据的效率；
2. C 版的 HisBarsInfo 获取历史 K 线详情数据的 maxLength 默认值改为 100，python 版本的默认值为 None，获取所有 K 线数据；

【策略交易】

1. C 版的 Buy、Sell、BuyToCover、SellShort 这些策略交易函数新增投保关键字，默认为投机（无套保需求可忽略）；

【策略状态】

1. 策略状态函数新增 userNo、hedge 关键字（单个账号和无套保需求可忽略）

【属性函数】

1. C 版 BidAskSize 函数：ZCE、SHFE、INE 返回值为 5，DCE、CFFEX 返回 1， 其他返回 10；
2. C 版的 ExchangeTime 函数返回值为交易所交易服务器的时间，python 版本该函数返回值为系统时间；
3. CommodityStatus 函数优先返回品种状态信息，若没有品种的状态，则返回该品种所属的交易所的状态，python 版本没有品种状态时返回空字符串；

【账户函数】

1. C 版的 A_AccountID 返回设置（界面设置或代码设置）的第一个账号，A_AllAccountID 返回设置的所有账号，界面设置和代码设置冲突时以代码为准，Python 版本的 A_AllAccountId 函数返回客户端登录的所有账号；
2. C 版的 A_GetAllPositionSymbol 函数返回指定账户与策略相关的持仓合约。python 版本是返回账户中所有持仓合约的合约编号；
3. C 版的 A_FirstOrderNo、A_NextOrderNo、A_LastOrderNo 删除了 sCont2 参数；
4. C 版 A 函数中关于持仓信息的函数增加投保类型关键字（无套保需求可忽略）；
5. C 版的 A 函数中有定单号参数的函数，指定定单号对应的合约在策略中没有订阅的话，返回值为 0 或空字符串。Python 版本的该类函数只要定单号存在就可以取到该定单的相关信息；

【日志函数】

1. C 版的 Log 函数最多支持 1024 字节的日志长度，超出部分将不显示；
2. C 版的 Alert 函数在客户端右下角弹出消息窗口，且该窗口会自动关闭

【Context 函数】

1. C 版的 context.triggerType()函数的触发方式中新增了”N”和”Z”两种类型；
2. C 版的 context.triggerData 函数不同触发方式的触发数据略有调整

7.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。

7.1 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、恒生、金仕达和启明星柜台，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）极智量化支持在服务器上运行么？

极智量化支持在服务器上运行（测试下不太兼容的服务器版本）

6）极智量化会收费么

极智量化是完全免费的产品，用户可以放心使用。

7）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。

7.2 极智量化使用

1）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。

2）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 客户端上登录交易账号，并在策略中利用 SetUserNo()函数设置需要进行交易的账户。

3）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。

4）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。

7.3 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？

函数目录中 context 函数分类中的 triggerType 函数可以用于获取当前的触发方式，具体判断方法如下代码所示：

```
if context.triggerType() == "S":     # 即时行情触发
    LogInfo("触发方式为即时行情触发")
if context.triggerType() == "T":     # 定时触发
    LogInfo("触发方式为定时触发")
```

5）策略文件运行后，策略运行目录中停止该策略，然后对策略文件进行改动，在策略运行列表中重新运行该策略运行的策略是改动前还是改动后的策略？

在策略运行列表中重新运行时运行的策略是改动后的策略。

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，无法重新设置合约；

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，可以在添加策略时在设置界面上选择更多设置—发单设置界面上勾选“发单预警”功能：

若用户需要自定义加入预警功能，也可以安装第三方的声音模块库并在策略中使用。

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：

添加策略处选择要关联的账号，然后在更多设置-发单设置处选择实盘运行：

11）极智量化如何下套利合约单

极智量化支持下套利单，具体方法见量化编辑器的示例策略目录下的系统示例--常用示例—布林带套利.py 和布林带套利(内外盘).py

12）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。

13) 关联账号与取消关联

用户可以在加载策略时选择需要关联的账号，也可以在策略运行过程中选择关联账号和取消关联账号。

7.4 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 A_SendOrder()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|SR|MAIN", 'T', 1, 200)  # 订阅白糖主连合约 1 秒线
```

7.5 常见异常及处理方式

python 提供了许多标准异常事件，如 Syntax，InputError，OSError 等。当 python 无法正常处理你的程序时，python 解释器就会抛出异常，从而影响程序的正常执行。本部分结合用户编写策略过程中经常遇到的异常事件进行介绍，并给出修改建议。

提示：其他未涉及的内容用户可以自行百度错误信息，一般都能找到答案。

AssertionError

介绍

python 语言有一个内置的 assert 语句，该语句可以创建基于逻辑断言的简单调试消息输出。Assert 函数的语法为：

assert condition, error_message(optional)

其中 condition 是一个表达式，例如 a=1，当表达式结果为 false 时即 a 不等于 1 时，触发 AssertionError 异常并输出 error_message 信息，其中 error_message 是用户想要输出的异常信息，该参数是可选的。

示例

```
    x = 1
    y = 0
    assert y != 0, "操作非法，分母不能为 0"
    LogInfo(x / y)
```

运行上述代码，在客户端会提示如下错误信息：

该段代码定义了两个简单的变量，并用断言语句判断 y 是否等于 0，等于 0 时判处 AssertionError 异常并输出错误信息，不等于 0 时输出 x 和 y 相除的结果。

总结

用户策略中出现 AssertionError 异常时一般是由 assert 语句抛出的。用户在策略中使用 assert 语句可以在确保条件不满足时抛出异常，停止策略，并帮助用户准确的查看引起异常的原因。

AttributeError

介绍

AttributeError 是当非法的属性引用或属性分配失败时所引发的错误。当引用无效的属性时，通常会引起 AttributeError，即属性错误。当对一个整形变量 a 使用 a.append 方法时，很明显将引起 AttributeError，因为整型变量没有 append 属性。

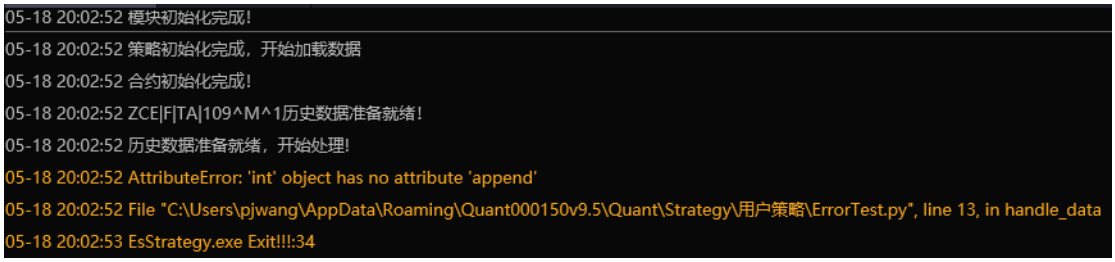
示例

例 1

正如在介绍部分说明的，对整型变量使用 append 方法将会引起错误：

```
a = 1
a.append(2)
```

执行上述代码，会输出如下信息：



输出的错误信息如：

AttributeError: "int" object has no attribute "append"

提示用户 int 对象没有 append 属性。

例 2

拼写错误也会导致 AttributeError，另外 python 是大小写敏感的：

```
ma = talib.Ma(Close(), 5)
```

```
    LogInfo(ma[-1])
```

在策略中执行上述代码，会输出如下信息：

```
05-18 20:07:28 模块初始化完成!
05-18 20:07:28 策略初始化完成，开始加载数据
05-18 20:07:28 合约初始化完成!
05-18 20:07:29 ZCE|F|TA|109^M^1历史数据准备就绪!
05-18 20:07:29 历史数据准备就绪，开始处理!
05-18 20:07:29 AttributeError: module 'talib' has no attribute 'Ma'
05-18 20:07:29 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 12, in handle_data
05-18 20:07:29 EsStrategy.exe Exit!!!:35
```

该段代码本意是要利用 talib 库的 MA 方法计算 5 周期收盘价的均值，但在写策略时确出现了拼写错误，MA 写为了 Ma，导致 python 解释器无法识别 Ma 属性导致错误，提示用户：

AttributeError: module 'talib' has no attribute 'Ma'

用户在编写策略时，遇到这种类型的错误时，首先需要检查是否存在拼写错误，如果确定没有拼写错误，再去检查是否存在该属性。也可以使用 try-except 语句主动检查在使用中是否出现了 AttributeError 错误，如：

```
    try:
        ma = talib.Ma(Close(), 5)
        LogInfo(ma[-1])
    except AttributeError as e:
        LogInfo(e)
        LogInfo("出现了属性错误，请检查")
```

再次提醒用户，捕获错误不是目的，而是要在出现错误时要做何种操作。

总结

AttributeError 也是用户在使用过程中经常用到的错误，出现这类错误的大多数原因是属性拼写错误，或者对象没有这个属性，又或者是 Python 的库中没有该属性。用户在编写策略时需要注意拼写错误问题，另外使用 try-except 语句也可以主动捕获 AttributeError 异常。同样，使用 pycharm 编辑器可以有助于发现这类错误。

IndentationError

介绍

Python 是一种通过缩进组织代码的语言。要求严格的代码缩进是 python 语法的一大特色。就像 C 语言的花括号一样。Python 每行代码前的缩进都有语法和逻辑上的意义。相同缩进行的代码是处于同一范围内的。每行代码中开头的空格数用于计算该代码的缩进级别。代码如果缩进不正确，整个代码将不会执行，python 解释器将会提示 IndentationError。python 每四个空格作为一个缩进级别。

导致缩进错误的主要原因为：

- 代码前后缩进量不一致
- Tab 与空格键混用

示例

这里举例说明 IndentationError 如何产生以及如何处理这种错误

例 1 代码前后缩进量不一致

```
s = 'IndentationError'
if s == 'IndentationError':
LogInfo('This is IndentationError test!')
else:
  pass
```

运行上述代码，将会提示如下错误信息：

IndentationError: expected an indented block

该段代码的缩进问题有两个，第一个是 if 判断条件后面的 LogInfo 语句没有用四个空格进行缩进，第二个错误是 else 后面的 pass 应该是四个空格缩进，但是确用了两个空格。第二个错误不容易看出来，用户可以借助 pycharm 软件很容易的发现这个错误：

```
25        if s == 'IndentationError':
26        LogInfo('This is IndentationError test!')
27        else:
28          pass
```

如图，pycharm 编译器用红色曲线标示了错误的位置。

为了解决此错误，必须为 if 后面的 print 语句和 else 后面的 pass 语句添加适当的缩进。新代码如下：

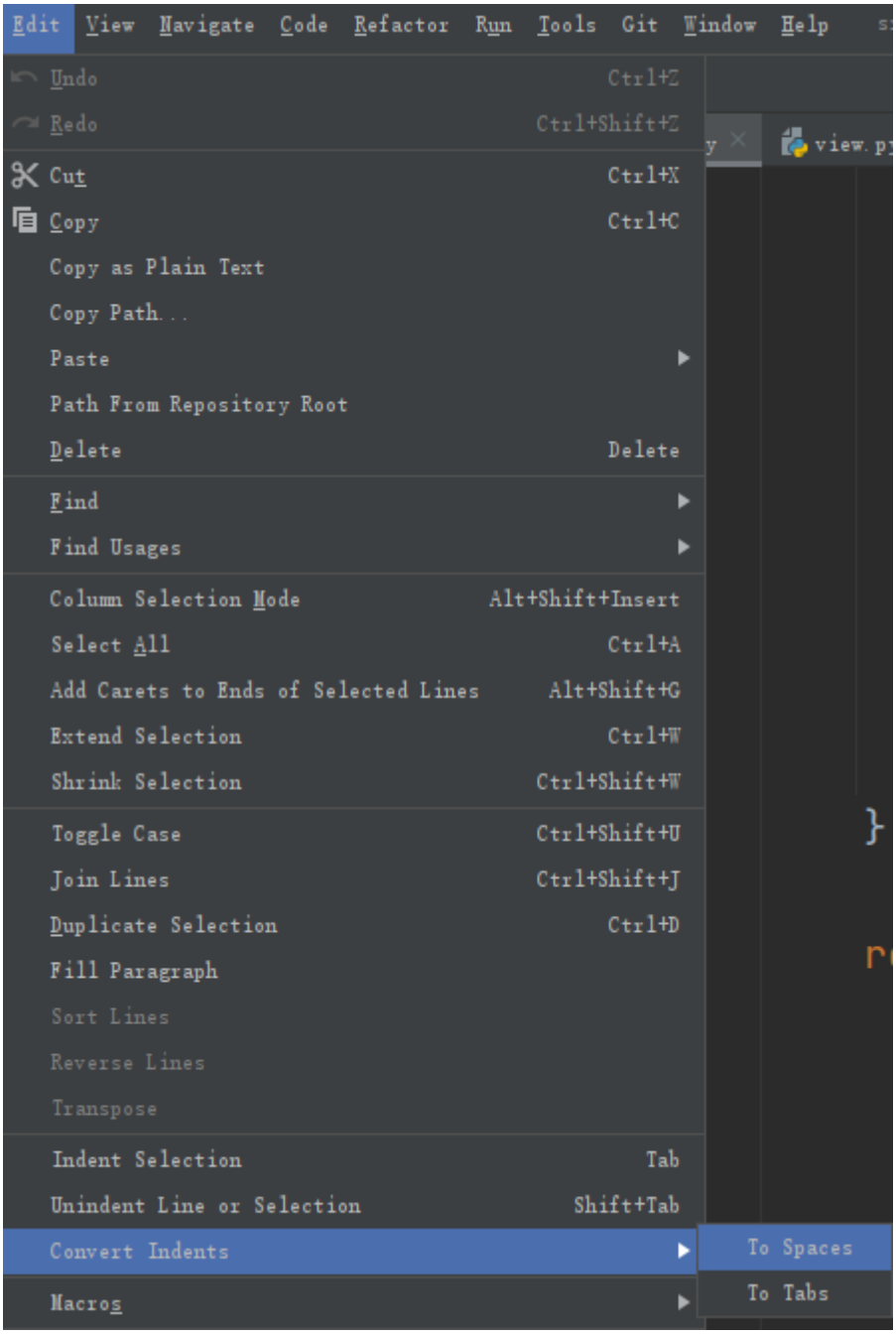
```
s = 'IndentationError'
if s == 'IndentationError':
    LogInfo('This is IndentationError test!')
else:
    pass
```

例 2 Tab 与空格键混用

当用户在代码中缩进采用 Tab 与空格混用，尤其是从网上复制代码时很容易出现缩进错误，报错信息如下：

IndentationError: unindent does not match any outer indentation level

用户在遇到这种报错信息时，同样可以利用 pycharm 编辑器解决这个问题，选中所有文本，然后在编辑器菜单栏，选择"Edit"-"Convert Indents"-"To Spaces"进行统一转换，如下图所示：



总结

python 利用空格和制表符来使代码保持逻辑关系。用户在编写策略代码时需要特别注意缩进问题，代码的缩进问题很可能会使策略想要表达的哪些代码属于哪个逻辑块产生很大的变化，用户需要特别注意。另外，借助 pycharm 工具可以辅助用户查找缩进错误。

IndexError

介绍

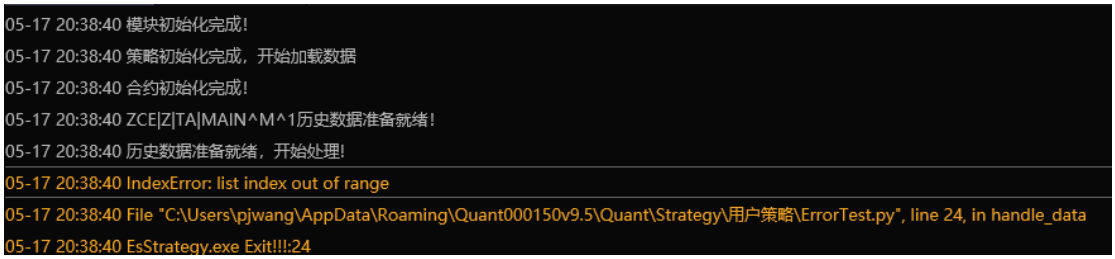
IndexError 是程序运行时出现的一种异常。从字面意思可以理解，IndexError 是索引错误，当指定为下标的索引不在列表的索引范围内，会引发程序的索引错误。

示例

例 1

```
names = ['a', 'b', 'c']
LogInfo(names[3])
```

执行上述代码，会输出如下信息：



python 中，list 是一个列表类型的数据结构，可以使用索引访问列表元素。列表中元素的索引范围为从 0 到 n-1，其中 n 为列表中的元素个数，对于以上代码中的 names 对象，有三个元素，因此 names 的索引范围为 0-2，用索引 3 访问列表元素，会提示：

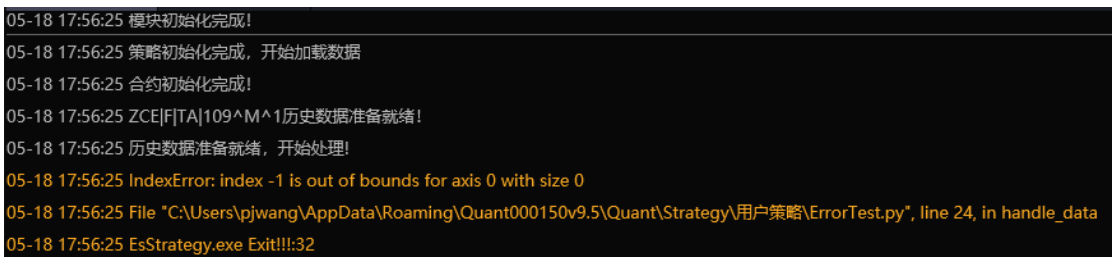
IndexError: list index out of range

这是 IndexError 最常出现的错误形式

例 2

```
cont = "ZCE|F|TA|109"

# 策略开始运行时执行该函数一次
def initialize(context):
    SetBarInterval(cont, "M", 1, 20)
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    LogInfo(Close(cont, "M", 2)[-1])
```

执行上述代码，会输出如下信息：



这里订阅了 PTA109 一分钟合约的数据，但是在 handle_data 函数中却用 Close 函数获取 PTA109 两分钟数据，并用索引-1 取 PTA109 合约两分钟数据的最新收盘价，因为并没有订阅 PTA109 两分钟的数据，无法取到数据，所以 Close(cont, "M", 2)返回值为空的数组，用索引取值就会提示如下错误：

IndexError: index -1 is out of bounds for axis 0 with size 0

用户在遇到这种报错信息的错误时，可以使用 len 函数检查下对象的长度，再去访问数据，如:

```
if len(Close(cont, "M", 2)) >= 1:
    LogInfo(Close(cont, "M", 2)[-1])
```

例 3

```
price = Close()[-1] + Open()[-1]
LogInfo(price[-1])
```

执行上述代码，会输出如下信息：

```
05-17 20:48:48 模块初始化完成!
05-17 20:48:48 策略初始化完成，开始加载数据
05-17 20:48:48 合约初始化完成!
05-17 20:48:48 ZCE|F|TA|109^M^1历史数据准备就绪!
05-17 20:48:48 ZCE|F|TA|110^M^1历史数据准备就绪!
05-17 20:48:48 历史数据准备就绪，开始处理!
05-17 20:48:48 IndexError: invalid index to scalar variable.
05-17 20:48:48 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 27, in handle_data
05-17 20:48:49 EsStrategy.exe Exit!!!:28
```

代码中取当前的收盘价和开盘价，将两者的和的值赋给 price，并输出 price[-1]的值，但由于 price 是标量，不可以使用索引取值，因此会提示：

IndexError: invalid index to scalar variable

表示对一个标量使用了非法的下标操作。用户遇到策略提示上述错误信息时，需要检查取索引的操作数是不是标量，是不是支持取索引操作，可以使用 type 方法查看对象的类型。如下代码所示：

```
price = Close()[-1] + Open()[-1]
if type(price) == list:
    LogInfo(price[-1])
```

总结

IndexError 是引用列表的索引且该索引不在列表索引范围内时所引发的错误，当对不支持索引操作的对象使用索引操作取值时也会发生这类错误，用户可以使用 try-except 语句块主动捕获这类异常。

IOError

介绍

在 Python 中使用输入和输出操作时，如果遇到与文件相关的错误，则代码将引发 IOError。当我们尝试打开一个不存在的文件时，将产生 IOError。

通常，当由于 IO 原因（例如“磁盘已满”或“找不到文件”）而导致输入输出操作（例如使用 open()打开文件，方法或简单的打印语句）失败时，会引发 IOError。

示例

```
import sys
file = open('myfile.txt', 'r')
lines = file.readline()
```

上述代码，试图以只读模式打开 myfile.txt 文件，并读取一整行，但是这段代码有一个问题，就是如果 myfile.txt 文件不存在的话，程序会报错，报错信息如下所示：

```
05-17 10:45:13 模块初始化完成!
05-17 10:45:13 策略初始化完成，开始加载数据
05-17 10:45:13 合约初始化完成!
05-17 10:45:13 ZCE|Z|TA|MAIN^M^1历史数据准备就绪!
05-17 10:45:13 历史数据准备就绪，开始处理!
05-17 10:45:13 FileNotFoundError: [Errno 2] No such file or directory: 'myfile.txt'
05-17 10:45:13 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 24, in handle_data
05-17 10:45:13 EsStrategy.exe Exit!!!:2
```

图中的报错信息 FileNotFoundError 属于 IOError 的一种。

如果程序中需要进行文件操作时，建议使用 try except 语句块进行处理，主动对可能出现的文件读写错误进行处理，如果有语句需要在出错后也要执行，可以使用 try except finally 语句块：

```
import sys
try:
    file = open ( "myfile.txt", 'r' )
except IOError as e:
    LogInfo(e)
finally:  # finally 的代码肯定执行，finally 语句时可选的
    LogInfo("这个语句不管是否出错一定执行！")
```

运行上述代码返回的信息如下，这段代码中增加了异常捕获处理：

```
05-17 14:38:35 模块初始化完成!
05-17 14:38:35 策略初始化完成，开始加载数据
05-17 14:38:35 合约初始化完成!
05-17 14:38:35 ZCE|Z|TA|MAIN^M^1历史数据准备就绪!
05-17 14:38:35 历史数据准备就绪，开始处理!
05-17 14:38:35 [   INFO][8]: [Errno 2] No such file or directory: 'myfile.txt'
05-17 14:38:35 [   INFO][8]: 这个语句不管是否出错一定执行!
```

总结

IOError 错误一般是文件名错误或者位置错误引起的。许多情况都会导致该错误的出现。推荐使用 try-except 语句块处理这种情况，这将为操作文件输入输出的用户节省大量的工作。捕获错误不是目的，最终的目的是要程序能够正确的执行。

KeyError

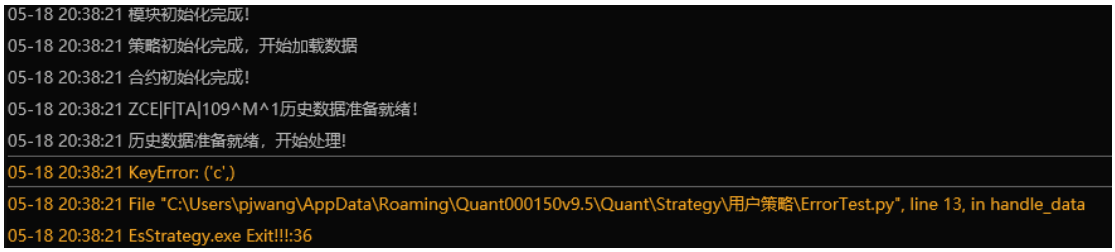
介绍

当用户尝试访问一个不在字典中的 key 时，会引发 KeyError 异常。python 字典类型的官方文档介绍，当访问映射的键并且该映射中没有找到该键时，会出现这种错误。映射是将一组值映射到另一组的数据结构。python 中最常见的映射结构是 dict，即字典。

示例

```
d = {'a': 1, 'b': 2}
d['c']  # 此处键'c'不存在，会报错
```

在策略中执行上述代码，会输出如下信息：



代码中定义了一个字典结构 d，并设置了两个键值 a 和 b，此时访问字典 d 的键 c，由于字典 d 中不存在键 c，索引会引发 KeyError 错误。可以使用字典类型提供的 get 方法，该方法在取不到某一键的值时会返回一个默认值，对上述代码进行修改，得到：

```
d = {'a': 1, 'b': 2}
LogInfo(d.get('c'))
```

用户可以自行百度查找了解更详细的用法。当然用户还可以使用 if 条件判断：

```
d = {'a': 1, 'b': 2}
if  'c' in d:
    LogInfo(d.get('c'))
else:
    LogInfo("字典 d 中没有键 c 对应的值")
```

总结

KeyError 大部分是由于错误的字典键查找导致的错误。另外需要注意，如果出现这类错误，可能需要在提示错误的行号附近查找错误原因。如果仍然不能找到错误位置，可以尝试使用 try-except 代码块逐行排查错误。

ModuleNotFoundError

模块是 python 不可或缺的部分。当使用模块时，经常会遇到 ModuleNotFoundError 错误。这是因为 python 无法成功导入模式所导致的。错误信息通常如：

ModuleNotFoundError: No module named '***'

当用户忘记安装需要用到的库时，会引发此错误；当用户自定义的模块中不允许导入相对文件时，也会引起该错误。

示例

例 1 未安装依赖项

```
import pandas as pd
```

运行此语句，若提示：

ModuleNotFoundError: No module named 'pandas'

这可能是因为用户未安装 pandas 库导致的这个错误。用户可以使用客户端界面上的 python 包安装功能安装 pandas 库。

例 2 用户定义的模块

python 有两种导入类型，绝对导入和相对导入。Python import 的搜索路径如下：

1. 在当前目录下搜索该模块
2. 在环境变量 PYTHONPATH 中指定的路径列表中依次搜索
3. 在 python 安装路径的 lib 库中搜索

我们创建一个目录结构如下的项目：

main.py

config.py

function

　　test.py

其中 main.py、config.py、function 在同一个文件目录下，function 是一个文件夹，test.py 文件在 function 文件夹下

在 config.py 文件中声明一个列表：

```
test_lsit = ['a', 'b', 'c', 'd']
```

接下来，编写 test.py 文件，实现一个打印 test_list 的函数：

```
def print_test(lst):
    for t in lst:
        print(t)
```

最后，在 main.py 中编写主程序，该程序首先导入 test 和 config 模块，然后执行 test 模块的 print_test 函数:

```
import test
import config

test.print_test(config.test_list)
```

运行上述代码，将会提示类似如下的错误：

ModuleNotFoundError: No module named 'test'

错误的原因是”test” 在 function 文件夹中，不在当前的工作路径中。要修改这个问题，需要做如下修改：

```
from function import test
import config

test.print_test(config.test_list)
```

注意：之所以能直接导入 config 是因为 config 文件与我们正在执行的程序位于同一个文件夹。关于如何导入自定义的库详细说明，用户可以查看该文档中的导入自己实现的库。

总结

当 Python 无法找到模块时，引发 ModuleNotFoundError。导致此错误的最常见原因是忘记安装模块或错误地导入模块。如果使用外部模块，需要检查是否已经安装此模块。如果要编写用户定义的模块，则必须仔细检查导入语句，确保它们导入的相对路径的正确性。

NameError

介绍

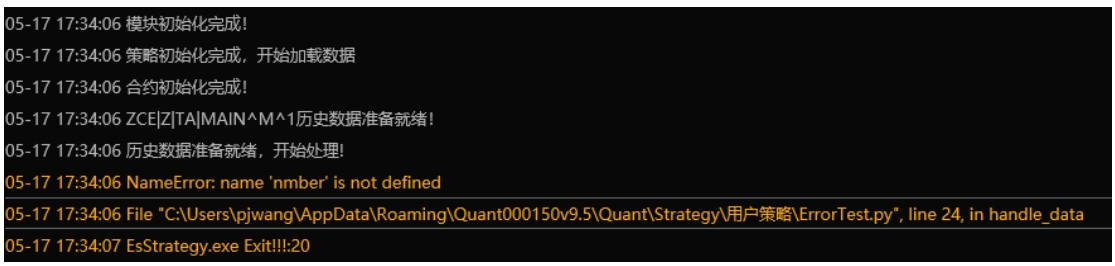
当执行一个事先没有声明的函数，或者使用未提前定义的变量、库、字符串时，python 解释器会生成 NameError 错误。解释器在执行时遇到无法识别的全局变量或局部变量时，也会生成 NameError。

示例

例 1

```
    number = 1
    LogInfo(nmber)
```

执行该段代码，会提示如下信息：



上述代码定义了一个变量 number，并用 LogInfo 输出 nmber 的值，但并没有对 nmber 进行定义，因此导致了 NameError 错误。Python 中对变量进行访问前，都需要先确定该变量是否已经定义。这段代码中的 NameError 主要是拼写错误造成的：对 number 变量访问时少了一个字母 u。

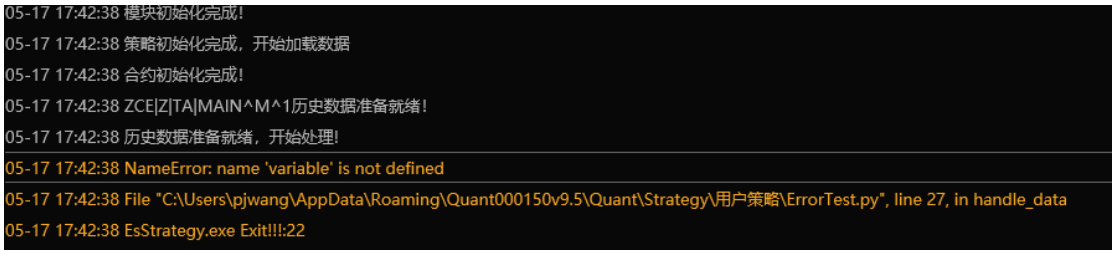
可以使用 try-except 语句处理 NameError 异常，如下所示：

```
try:
    number = 1
    LogInfo(number)
except NameError as e:
    LogInfo("执行代码出现 NameError 异常")
```

例 2

```
def func():
    variable = "GeeksforGeeks"
    LogInfo(variable)


func()
LogInfo(variable)
```

执行上述代码，会提示如下信息：



该段代码在 func 函数中定义了一个变量 variable，该变量为局部变量，作用域为 func 函数，在 func 函数中访问 variable 变量时不会报错，但是在 func 函数外部访问 variable 变量时会提示：

NameError:name ‘variable’ is not defined

函数外部方为 variable 变量时由于超出了该变量的作用域范围，因此 python 解释器会提示该变量未定义。

总结

当解释器无法识别用户在代码中使用的 name 时，解释器将会抛出 NameError。使用 Python try-except 语句可以避免 NameError，这可以向用户表明代码块中发生的 NameError 而不实际抛出错误。出现 NameError 错误时，用户可以检查错误信息中提示 NameError 的对象是否定义，如果已经定义，检查下是否在该对象的作用于范围内，还要注意拼写错误。另外，使用 Pycharm 代

码编辑器，也可以更好的发现 NameError 错误。Pycharm 编辑器会在未识别的标识符下方用红色的曲线标出。

SyntaxError

介绍

运行 python 代码时，python 解释器将首先对代码进行解析，以便将其转换为 python 字节码并执行。在代码解析阶段，解释器将查找可能存在的语法错误。如果你的代码无法被解析成功，则表明代码的某处使用了错误的语法，解释器将尝试向用户显示错误发生的位置。虽然 python 会帮助你定位语法错误出现的位置，但有时并不能完全准确的定位错误出现的具体位置，需要在错误位置的附近行查找问题。引起语法错误的常见原因如下：

- 拼写错误，python 关键字缺失或使用错误
- 缺少必需的空格
- 缺少引号
- 误用了块语句（if-else，循环）
- 缺少赋值运算符(=)
- 非法的变量声明
- 非法的函数调用或定义

示例

这里举例说明 Syntax 如何产生以及如何处理这种错误

例 1 拼写错误，python 关键字缺失或使用错误

Python 关键字是一组受保护的关键字，在 Python 中具有特殊含义。这些关键字不能在代码中用作标识符，变量或函数名称。它们是 python 语言的一部分，只能在 Python 允许的上下文中使用。

用户可能会遇到以下三种常见的关键字错误

1. 关键字拼写错误
2. 缺少关键字
3. 误用关键字

```
fro i in range(10):
    pass
```

运行上述代码会输出如下信息：

```
05-19 11:13:48 SyntaxError: invalid syntax (ErrorTest.py, line 13)
05-19 11:13:49 EsStrategy.exe Exit!!!:43
```

提示 SyntaxError: invalid syntax，这种错误需要用户在指定错误的位置仔细查找是否有关键字拼写错误。这里错误是因为误将 for 拼写为了 fro；

```
for i range(10):
    pass
```

运行上述代码会显示 SyntaxError: invalid syntax。这里错误是 for 语句中缺少 in 关键字；

```
names = ['a', 'b', 'c']
if 'a' in names:
    LogInfo('a found')
    break

if 'b' in names:
    LogInfo('b found')
    continue
```

运行上述的代码会显示

SyntaxError：'break' outside loop

或

SyntaxError: 'continue' not properly in loop

注意，只允许在特定情况下使用关键字。如果使用不正确，则 Python 代码中的语法将无效。正如这个例子中的错误：在循环外部使用循环 continue 或 break。break 和 continue 关键字必须在循环体中使用；

```
pass = True

def pass():
    LogInfo("This is wrong")
```

运行上述代码会显示 SyntaxError: invalid syntax，这种错误可能对于不知道 pass 是 python 关键字的用户来说并不容易发现。用户可能需要查找下对应 python 版本的关键字列表，熟悉下 python 预留的关键字。

可以通过：

```
import keyword
print(keyword.kwlist)
```

或

```
keyword.iskeyword('pass')
```

判断某一标识符是否是 python 预留的关键字；

例 2 缺少必需的空格

```
name = "code leaks"
if name == "code leaks":
```

```
print("hello")
else:
    print("who?")
```

运行上述代码，会提示：

```
05-19 11:29:03 IndentationError: expected an indented block (ErrorTest.py, line 15)
05-19 11:29:10 EsStrategy.exe Exit!!!:45
```

python 编程语言需要缩进块组织代码，if 条件为 true 后，由于 print 语句前缺少缩进的空格，导致程序错误。

例 3 缺少圆括号，方括号或引号

python 中圆括号、方括号、引号缺失或不匹配导致的语法错误在很长的嵌套括号行或多行块中很难发现。Python 的错误提示有助于用户查找这类错误：

```
message = 'don't'
```

运行该语句会提示：SyntaxError: invalid syntax

错误原因是在单引号结束后出现了一个 t’，可以做如下修改解决这类错误：

1. 用反斜杠(‘don\’t’)转义单引号
2. 字符串外层用双引号(“don’t”)

另一个常见的错误是未闭合字符串。

```
message = "This is an unclosed string
```

运行该语句会提示：

SyntaxError: EOL while scanning string literal，

表示关闭打开的字符串之前，python 解释器到达了行尾（EOL）。

对于缺少括号的情况和上述缺少引号的情况大致相同。如：

```
def foo():
    return [1, 2, 3


print(foo())
```

运行此代码，会提示 print(foo())处存在语法错误，python 解释器认为列表中含有三个元素: 1, 2, 3 print(foo())，Python 使用空格进行逻辑分组，因为没有逗号或括号将 3 与 print(foo())分开，Python 将它们合并为列表的第三个元素。

例 4 误用块语句（if-else 循环）

```
    name="code leaks"
    if name == "code leaks"
        LogInfo("Bingo!")
    else:
```

```
        LogInfo("Wrong!")
```

运行上述代码，会提示 Syntax，可以看到在 if 判断条件的后面缺少了一个冒号(:)。同样，while 语句块和 for 语句块在使用中同样要注意这个问题。

例 5 误用赋值运算符 (=)

```
len('hello') = 5
```

运行这段代码会提示：SyntaxError: can't assign to function call

```
'foo' = 1
```

运行这段代码会提示：SyntaxError: can't assign to literal

```
1 = 'foo'
```

运行这段代码会提示：SyntaxError: can't assign to literal

第一个例子尝试将 5 赋值给 len 函数调用的结果，第二个和第三个例子尝试给字符串和整数赋一个常量值。这些错误通常是由于用户尝试给一个常量赋值引起的。

用户的目的可能不是为常量或函数调用的结果分配值。如果用户在编码过程中不小心漏掉了一个等号(=)，就会引起这个错误。这会使判断是否相等操作变为赋值操作。

当 python 解释器指出无法对某一对象赋值时，首先需要检查是否在需要用比较运算符(==)的地方误用了赋值运算符(=)。该错误也可能因为用户试图给 python 关键字赋值导致。

例 6 无效的变量声明

python 的变量命名规则为：

- 变量名只能是字母、数字或下划线的任意组合；
- 变量名的第一个字符不能是数字
- 变量名不能是 python 关键字，如 and、for、class、in 等

违法以上命名规则的变量声明会引起 SyntaxError 错误。

```
    123a = "abc"
```

因为变量声明时违法了变量名的第一个字符不能是数字规则，执行该代码时会报错，用户在对变量命名时应该遵循正确的命名规则。

例 7 无效的函数调用或定义

python 的函数声明也需要符合要求的语法。首先需要正确的使用空格和冒号(:)，还需要传递正确的函数参数：

```
    def func():
        LogInfo("Hi")
    func(1)
```

因 func 函数不需传入参数，但在 func 函数调用时传入了一个参数，因此同样会报错。

例 8 字典语法错误

Python 字典键和值是用冒号分隔的，如果用等号分隔会引起语法错误。例如：

```
ages = {'Tom' = 24}
```

同样，在定义字典时缺少逗号同样会引起错误：

```
ages = {
  'pam': 24,
  'jim': 25
  'michael': 26
}
LogInfo(f"Michael is {ages['michael']} years odl.")
```

可以看到在代码的第三行字典的第二个条目'jim': 25 后面缺少逗号，运行该段代码将会输出：

SyntaxError: invalid syntax

注意，错误信息定位显示的错误在第 4 行而不是第 3 行。python 解释器只能定位最先发现问题的地方，当用户遇到 SyntaxError 的定位信息但指定的错误位置看起来没有问题时，可以在错误位置的上面几行代码查找下问题。

总结

正如上述总结的那样，许多情况都会导致 SyntaxError，并且 python 解释器给出的错误位置通常并不能准确的指出错误的发生位置，需要用户仔细的去找出错误所在，用户可以借助 pycharm 等第三方编译软件进行错误的查找。

TypeError

介绍

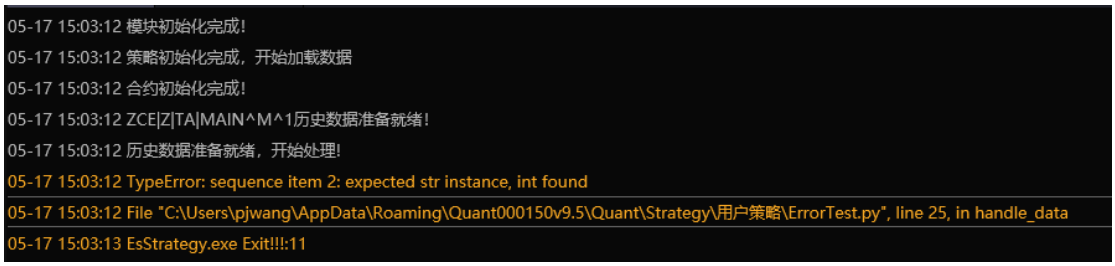
python 中对一个对象执行错误的类型操作会引起 TypeError 错误。例如，如果对列表对象而不是 int 对象做取平方根操作时，python 解释器将会产生该错误。当对一个对象执行不支持的操作时，也会生成该错误，例如对字符串和 list 对象执行相加操作也会引起 TypeError 错误。

示例

例 1

```
list1 = ['1', '2', 3, 4]
list2 = 's'
result = list2.join(list1)
LogInfo(result)
```

运行上述代码，返回的信息如下：



在上述代码中将 list1 和 list2 用 join 函数将两个对象连接在一起，但 list2 中的元素并不全为字符串对象，因此运行上述代码 python 解释器会提示：

TypeError:sequence item 2: expected str instance,int found

意思是序列索引为 2 的对象,即序列的第三个元素 3，是整型变量而不是字符串变量。

可以对 list1 中的元素执行 str 类型转换，将 list1 中的元素转换为字符串类型，防止出现 TypeError 错误：

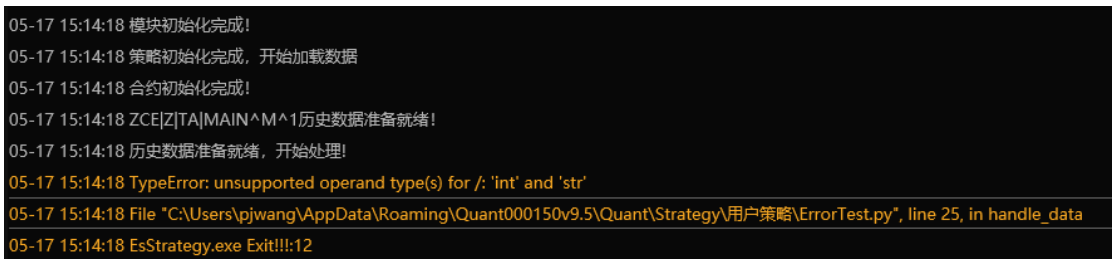
```
list1 = ['1', '2', 3, 4]
list2 = 's';
result = list2.join(str(itm) for itm in list1)
LogInfo(result)
```

运行上述代码，将输出：1's'2's'3's'4

例 2

```
a = 1
b = 's'
c = a / b
LogInfo(c)
```

运行上述代码，返回的信息如下：



上述代码中对字符串对象 b 执行除操作，因此 python 解释器会抛出 TypeError，并指出 int 和 str 对象之间不支持该操作。

可以对该段代码进行修改，增加对操作数的类型判断：

```
a = 1
    b = 's'
    if(type(b) != int or type(a) != int):
        # 这里可以执行用户需要的操作
```

```
        LogInfo("One of the number is not integer")
    else:
        c = a / b
    LogInfo(c)
```

上述代码增加了用 type 方法对 a 和 b 的类型进行判断，类型都为 int 就执行除操作。

总结

在使用规程中，尽量检查要传递给操作的对象的类型，以及特定的对象是否支持该操作。用户写策略过程中可以通过在此操作之前添加一个额外的类型检查步骤（例如例 2 中的 if 语句进行类型判断）或使用 try-catch 语句块来避免此错误。如果类型不符合要求，可以输出一个指示错误的信息，或停止策略执行。

UnboundLocalError

介绍

当一个局部变量未声明前就对该变量进行访问，会提示 UnboundLocalError，要理解局部变量的概念，这里需要简单的介绍一下 python 的命名空间及 LEGB 规则：

- L-Local 函数或者类的方法内部
- E-Enclosed 嵌套函数（一个函数包裹另一个函数）
- G-Global 模块中的全局变量
- B-Builtin 指 python 为自己保留的特殊名称

python 在查找“名称”时，是按照 LEGB 的顺序查找的，即：L->E->G->B，查找一个名为 x 的变量，python 首先在函数内部，局部（Local）范围来查找这个变量；如果没有找到，则到包含这个函数定义的外围去查找（称作 Enclosing），这个外围或许是另外一个函数（包括匿名函数）。如果还是没有，继续朝外查找，一直到模块级别，从这里定义了全局（Global）变量中寻找；如果仍然没有找到，则查找 Python 内置变量(Built-in)，看是否有相同名字的。

注：在上述查找过程中，一旦变量找到，就不再继续朝外围查找。也就是说 LEGB 同时也定义了从 L 到 B 得优先级。

用户可以点击链接了解 LEGB 的详细介绍：

python LEGB 规则

用户理解了这个规则，也就能理解为什么有时需要在策略的函数中将变量声明为 global 变量的原因了。

示例

```
import talib
```

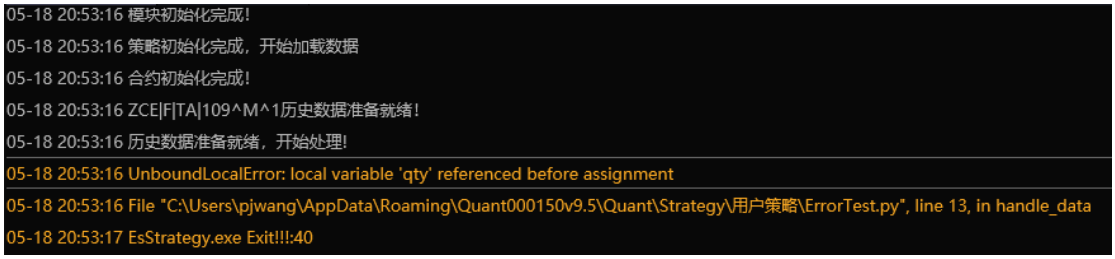
```
cont = "ZCE|F|TA|109"

qty = 1
# 策略开始运行时执行该函数一次
def initialize(context):
    SetBarInterval(cont, "M", 1, 20)
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    LogInfo(qty)
```

上述示例在 handle_data 函数外定义了一个变量 qty，该变量由于在函数外部定义，因此 qty 是全局变量（Global），此时在 handle_data 中访问 qty 变量，因为在局部（L）和嵌套（E）中均未找到，因此访问全局变量 qty，策略会输出 qty 的值 1。

下面稍微对策略中进行修改，在 handle_data 中先访问 qty，然后对 qty 进行重新赋值：

```
import talib

cont = "ZCE|F|TA|109"

qty = 1
# 策略开始运行时执行该函数一次
def initialize(context):
    SetBarInterval(cont, "M", 1, 20)
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    LogInfo(qty)
    qty = 2
    LogInfo(qty)
```

此时程序会输出如下信息：



我们仅仅是在 handle_data 函数中重新对 qty 赋值，结果程序输出了如下错误信息：

UnboundLocalError: local variable ‘qty’ referenced before assignment

提示局部变量 qty 在声明之前进行了引用。同样，根据 LEGB 规则，python 首先在函数内部（L）查找变量 qty，在函数内部找到了局部变量 qty，这样局部变量 qty 将屏蔽全局变量 qty，同时可以看到在 handle_data 函数内部声明 qty 之前，调用了 LogInfo(qty)，但此时 qty 还没有声明。因此提示 UnBoundLocalError：局部变量 qty 在声明之前进行了引用。可以在 handle_data 中将 qty 变量声明为全局变量，这样 handle_data 中访问的 qty 就变成了全局变量：

```
import talib

cont = "ZCE|F|TA|109"

qty = 1
# 策略开始运行时执行该函数一次
def initialize(context):
    SetBarInterval(cont, "M", 1, 20)
    SetTriggerType(5)

# 策略触发事件每次触发时都会执行该函数
def handle_data(context):
    global qty
    LogInfo(qty)
    qty = 2
    LogInfo(qty)
```

总结

理解了 LEGB 规则对于理解 UnboundLocalError 至关重要，不仅如此，用户对 python 中变量的访问顺序也会十分明确。

ValueError

介绍

ValueError 为 python 编程语言的内置异常类型。当操作或函数接收到类型正确但值不合适的参数时引发该错误，并且通常 ValueError 并不像其他异常那样对错误的信息给出了精确的描述。

示例

例 1

```
import math
math.sqrt(-10)
```

运行上述代码，会输出如下错误信息：

```
05-17 16:30:37 模块初始化完成!
05-17 16:30:37 策略初始化完成，开始加载数据
05-17 16:30:37 合约初始化完成!
05-17 16:30:37 ZCE|Z|TA|MAIN^M^1历史数据准备就绪!
05-17 16:30:37 历史数据准备就绪，开始处理!
05-17 16:30:37 ValueError: math domain error
05-17 16:30:37 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 24, in handle_data
05-17 16:30:38 EsStrategy.exe Exit!!!:16
```

上述代码中用 python 的 math 库中的求平方根方法 sqrt 对-10 求平方根，但求平方根的操作数必须是正整数，因此，python 解释器会抛出异常：

ValueError: math domain error，

提示该错误为数学领域的错误，但并没有指出-10 不能求平方根。

可以用 try-except 语句块处理这类异常：

```
import math
a = 4
try:
    LogInfo(math.sqrt(a))
except ValueError as e:
    LogInfo(f"sqrt 的操作数{a}不是正整数")
```

如果 a 不为正整数的话，该段代码会输出错误信息。

例 2

```
int('abc')
```

运行上述代码，会输出如下错误信息：

```
05-17 15:46:59 模块初始化完成!
05-17 15:46:59 策略初始化完成，开始加载数据
05-17 15:46:59 合约初始化完成!
05-17 15:46:59 ZCE|Z|TA|MAIN^M^1历史数据准备就绪!
05-17 15:46:59 历史数据准备就绪，开始处理!
05-17 15:46:59 ValueError: invalid literal for int() with base 10: 'abc'
05-17 15:46:59 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 23, in handle_data
05-17 15:47:00 EsStrategy.exe Exit!!!:14
```

在上述代码中将字符串’abc’转化为 int 类型， python 解释器会提示：

ValueError: invalid literal for int() with base 10: ‘abc’，

表示试图将一个与数据无关的类型转化为整型。

例 3

```
lst = [1,2,3,4,5]
a,b,c = lst
LogInfo(a)
LogInfo(b)
LogInfo(c)
```

运行上述代码，会输出如下错误信息：

```
05-17 15:53:34 模块初始化完成!
05-17 15:53:34 策略初始化完成，开始加载数据
05-17 15:53:34 合约初始化完成!
05-17 15:53:34 ZCE|Z|TA|MAIN^M^1历史数据准备就绪!
05-17 15:53:34 历史数据准备就绪，开始处理!
05-17 15:53:34 ValueError: too many values to unpack (expected 3)
05-17 15:53:34 File "C:\Users\pjwang\AppData\Roaming\Quant000150v9.5\Quant\Strategy\用户策略\ErrorTest.py", line 24, in handle_data
05-17 15:53:34 EsStrategy.exe Exit!!!:15
```

在上述代码中，将列表变量中的值分配给对象 a，b，c，但变量的数量必须与列表中元素的数量相等，否则 python 解释器会提示：

ValueError: too many values to unpack(expected 3)

表示需要分配的值的个数和给定的变量个数不匹配，这里 lst 中有五个元素，但是等号左侧只提供了三个变量 a，b，c。

总结

在 python 中，传递具有错误的值的参数时会引起 ValueError 错误。这类异常同样可以用 try-except 语句块捕获并进行适当的处理。

7.6 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）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

3）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。

4）极星 9.5 客户端如何配置背景色？

在极星 9.5 客户端标题栏可以设置背景：

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：

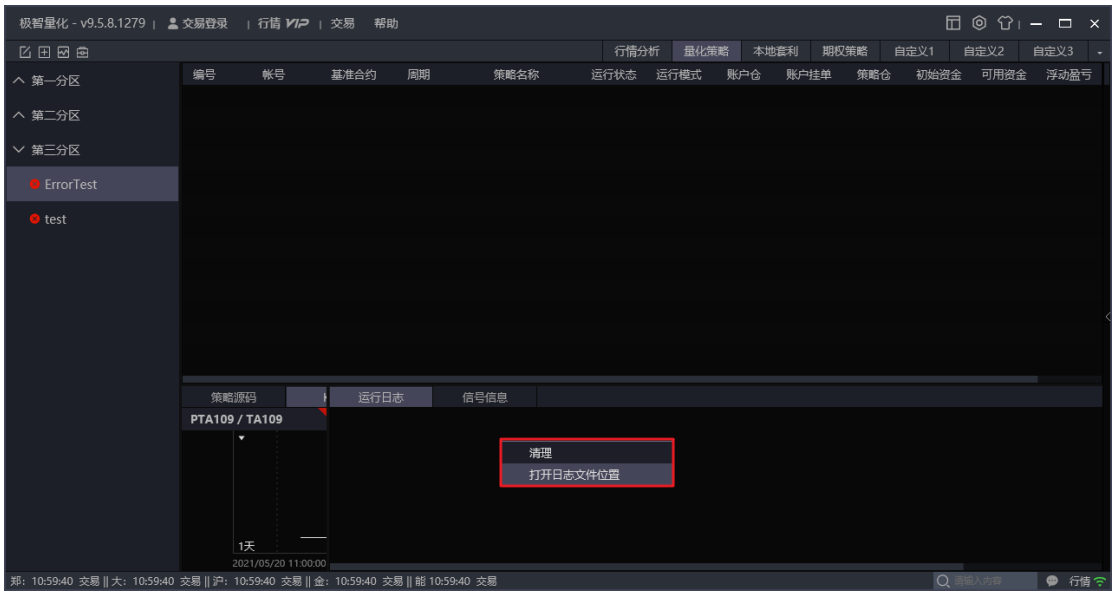
在添加的合约上右键选择配置列头，在弹出的窗口上勾选上“原始合约代码”，点击确定：

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）极智量化的日志文件存放在什么位置？

用户可以在主界面的量化策略页面下，在运行日志或信号信息处右键选择“打开日志文件位置”，找到日志文件



7）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./Quant/strategy 文件夹下。

8）如何注册模拟交易账号？

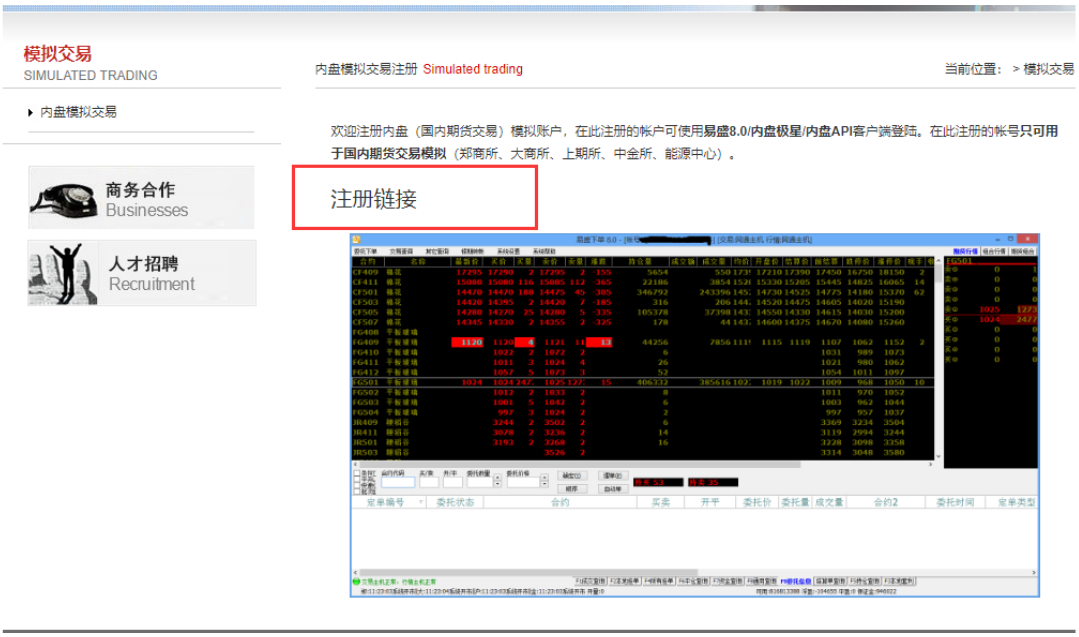
模拟交易可以更好的帮助用户观察策略在实盘交易时的运行情况。模拟交易的前提是注册模拟账号，注册模拟账户的流程如下：

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：在界面上点击注册模拟交易账号：

Step4：点击注册连接：



Step5：在新打开的界面中填入个人资料（示例界面为临时界面，实际使用中可能会有不同）。

Step5：注册信息提交之后，一天后就可以登录极星 9.5。模拟账号需要在您 QQ 号前加字母 Q，即 Q+QQ 号。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：

模拟		内盘		外盘	
★	易盛信息		郑州电信(内盘启明星)		2
B	宝城期货		郑州联通(内盘启明星)		2
C	创元期货		郑州联通(外盘北斗星)		3
D	东证期货		郑州电信(外盘北斗星)		2
	大陆期货		极星仿真		23
	东航期货				
	大地期货				
F	方正中期				
G	广发期货				
	国富期货				
	冠通期货				
	广州期货				
H	宏源期货				