国信iQuant策略交易系统 Python API 使用手册

文档状态	[ ] 初稿 [ ] 评审通过 [ ] 修改 [√] 发布 [ ] 作废
文档标识	
当前版本	V1.0
作 者	模型部
部门 / 厂商	国信证券
完成日期	2019-08-07

目录

概述

感谢您使用国信 iQuant 极速策略交易系统平台，以下内容主要介绍国信 iQuant 系统平台的 Python API 的使用方法。内容较多，可使用 Ctrl + F 进行搜索。

国信 iQuant 极速策略交易系统平台与其他普通股票软件最大的不同点就是其提供模型编辑、模型交易和风控功能。通过模型编辑，用户可以快速的将自己的转化为计算机代码，形成自己的交易策略，让计算机帮助用户实现策略的回测检验并实现无人值守自动化交易。

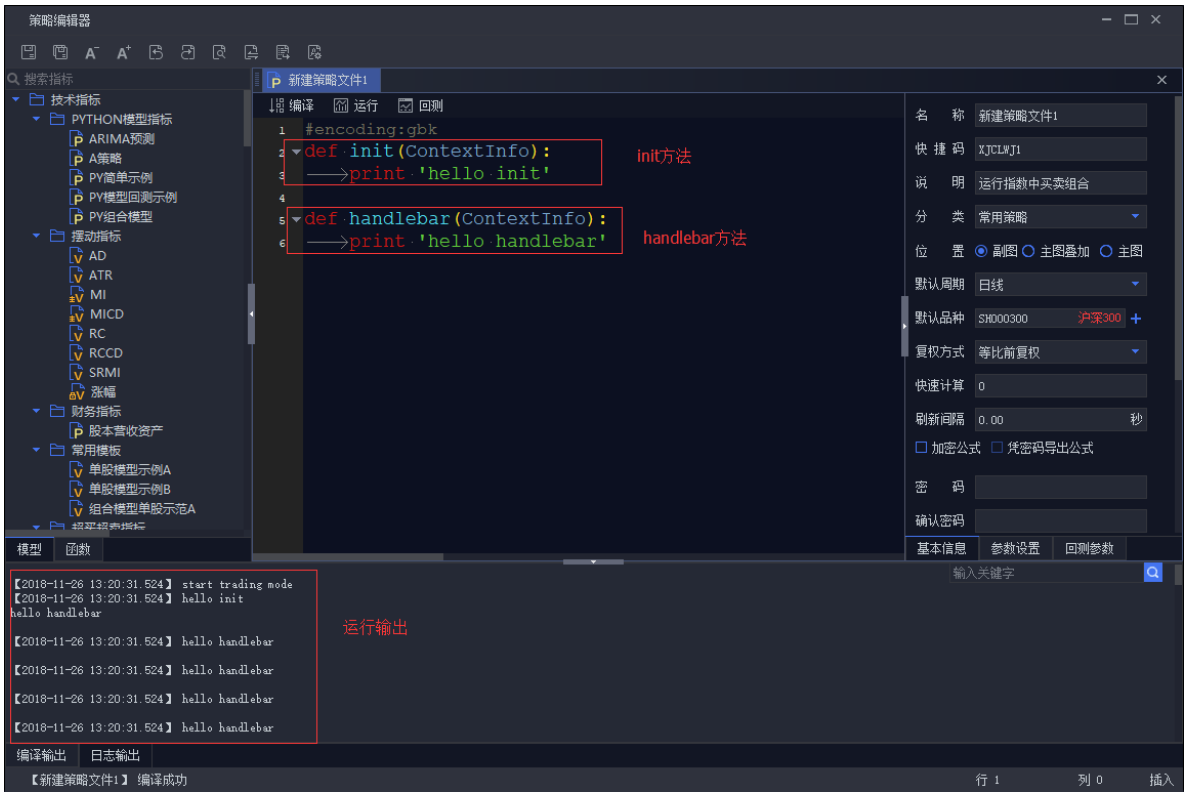
在程序化的支持上，国信平台目前支持的编程语言：Python 。在后续的规划中，国信 iQuant 系统将全面支持市面上主流的用于量化领域的编程语言。

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

1. 创建策略

1.1. 策略示例

1.1.1. 一个简单的 Python 策略



简单的Python策略

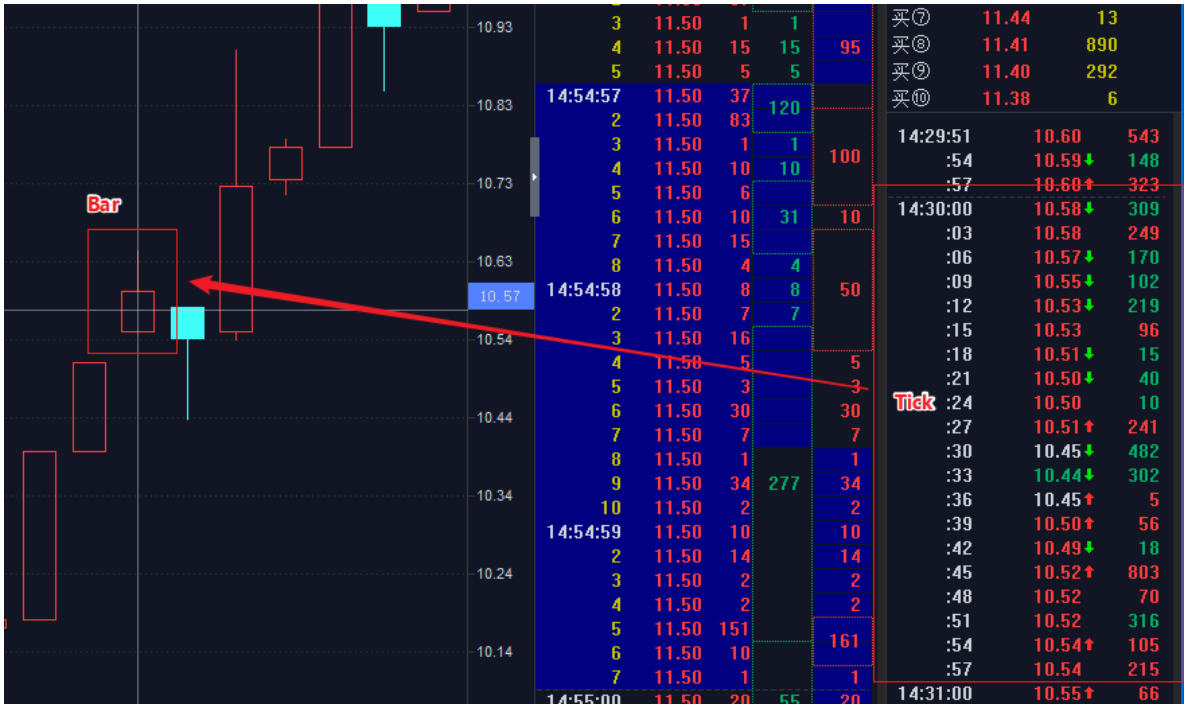
上图展示了如何在国信的平台上用 Python 写一个输出 hello world 的模型。图中的模型实现了一个 Python 模型**必须实现**的两个接口：**init** 与 **handlebar** 。输出窗口展示了这个模型的运行输出。系统首先调用 init 方法输出了hello init，随后在每根 K 线上调用一次 handlebar，输出一次 hello handlebar。

1.2. 运行机制

1.2.1. 重要概念

（1）Bar的概念

我们把单根 K 线称之为 Bar，每根 Bar 由 tick（分笔）组成。



分钟Bar 示例

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

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

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

（2）Init

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

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

Init 初始化

（3）Handlebar

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

（4）ContextInfo

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

1.2.2. Python 策略运行机制

用户在界面上提交请求后，在最终看到结果前会经历两步：

（1）创建模型，初始化环境，发送数据请求；

（2）数据到达后，调用 Python 的 init 函数进行 Python 初始化，然后运行 handlebar 方法；

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

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

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

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

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

获取当前 Bar 位置

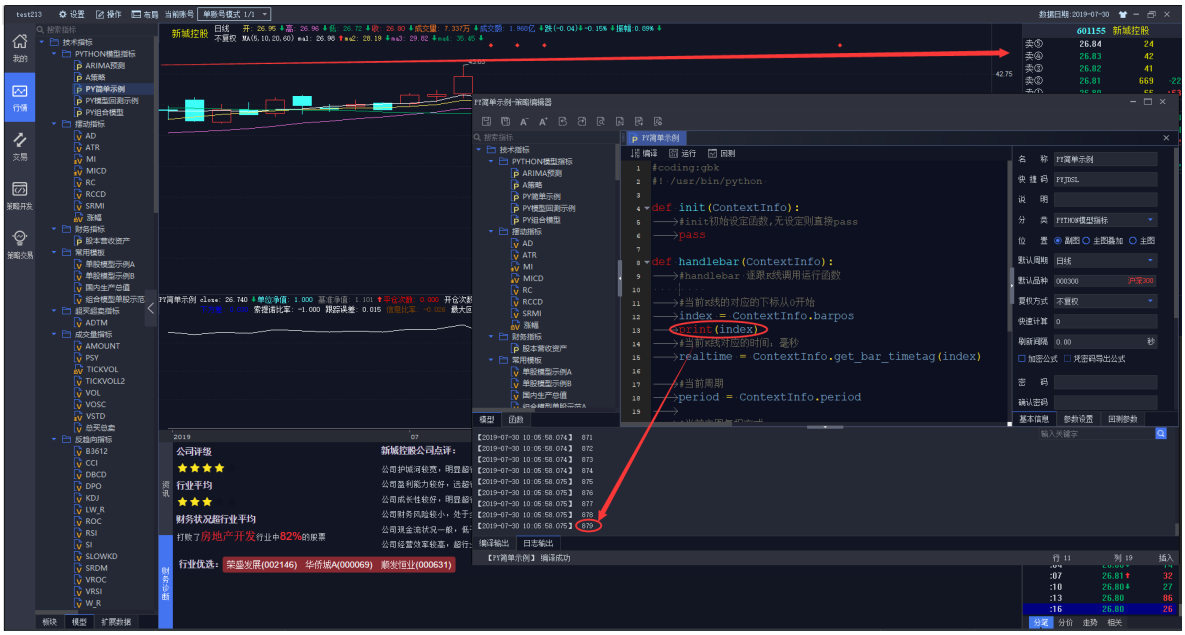
*注：

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

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

1.2.3. Python 交易函数运行机制

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



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

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

每个 tick 数据来时，最后一根 K 线会随着变动

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

一个模型判定：当收盘价 > 开盘价时产生模型信号触发下单

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

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

（1）设置 passorder 函数中的 quickTrade 参数为 1 ，立即下单；

（2）使用 do_order(ContextInfo) 函数。

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

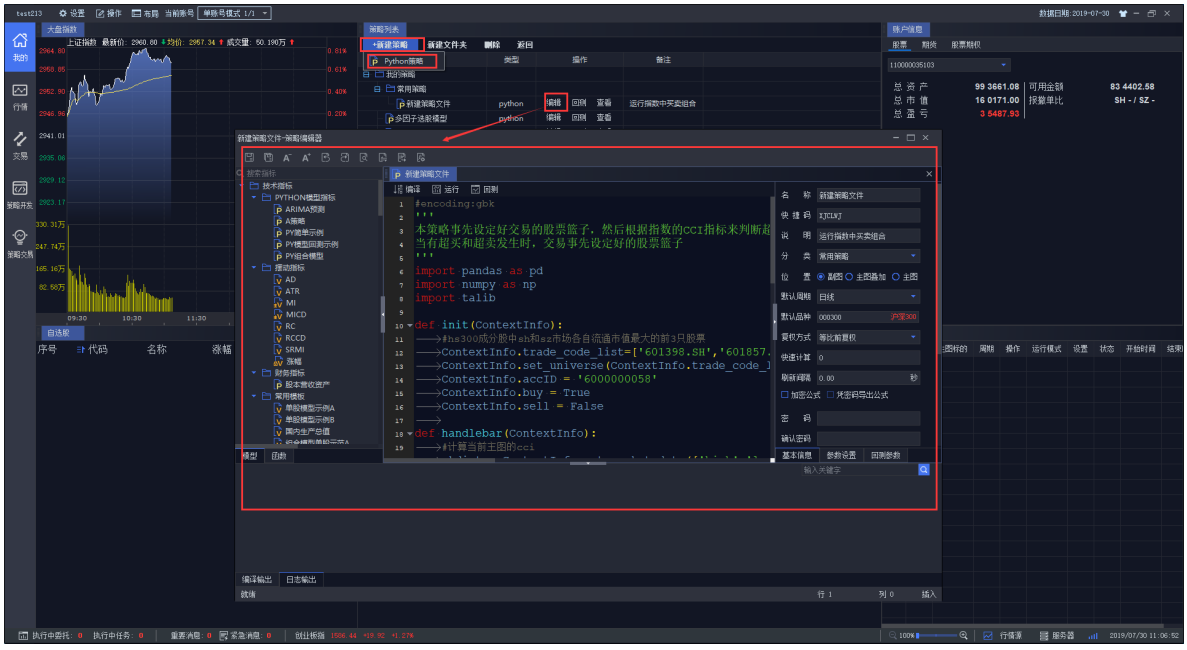
do_order(ContextInfo) 函数被调用后会把上一根 K 线生成的模型信号立刻发出，且只发一次，解决交易函数必须是在下一根 K 线的第一个 tick 数据时发信号的问题。

2. 创建一个 Python 策略

2.1. 新建一个 Python 策略

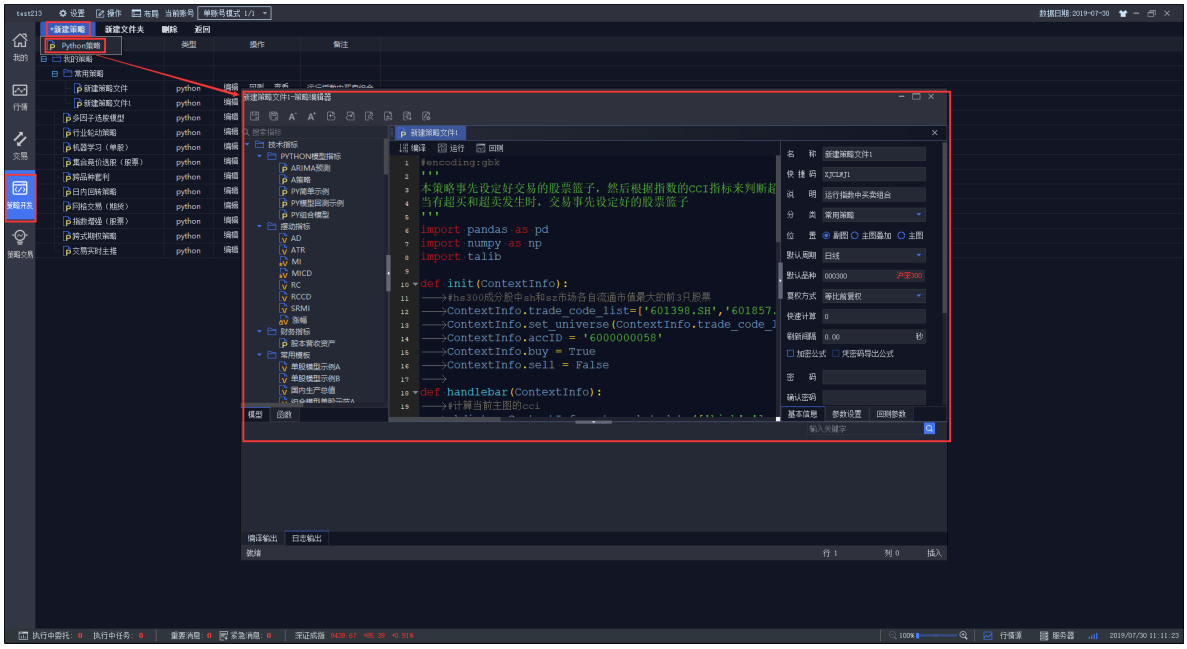
模型创建方法有三种：

方法一，使用系统预置的各种示例模型，点击后方"编辑'按钮，并在弹出的【策略编辑器】中以此示例模型代码为基础进行编写。或者点击新建模型，选择 Python 模型，在弹出的【策略编辑器】中从头到尾编写一个用户自己的量化模型。



我的主页-编辑模型

方法二，在【策略开发】界面，使用系统预置的各种示例模型，点击后方"编辑'按钮，并在弹出的【策略编辑器】中以此示例模型代码为基础进行编写。或者点击新建模型，选择 Python 模型，在弹出的【策略编辑器】中从头到尾编写一个用户自己的量化模型。



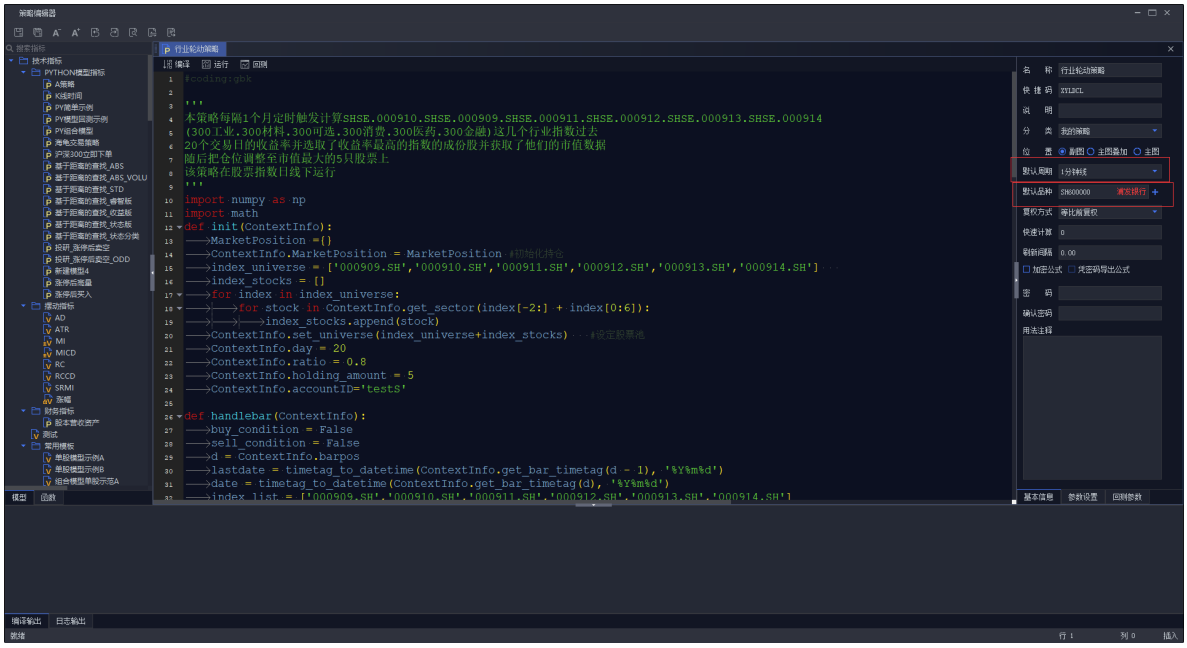
策略开发-新建模型

方法三，在模型管理面板右键，选择新建模型，并选择 Python 模型。

模型-新建模型

2.2. 策略编写

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

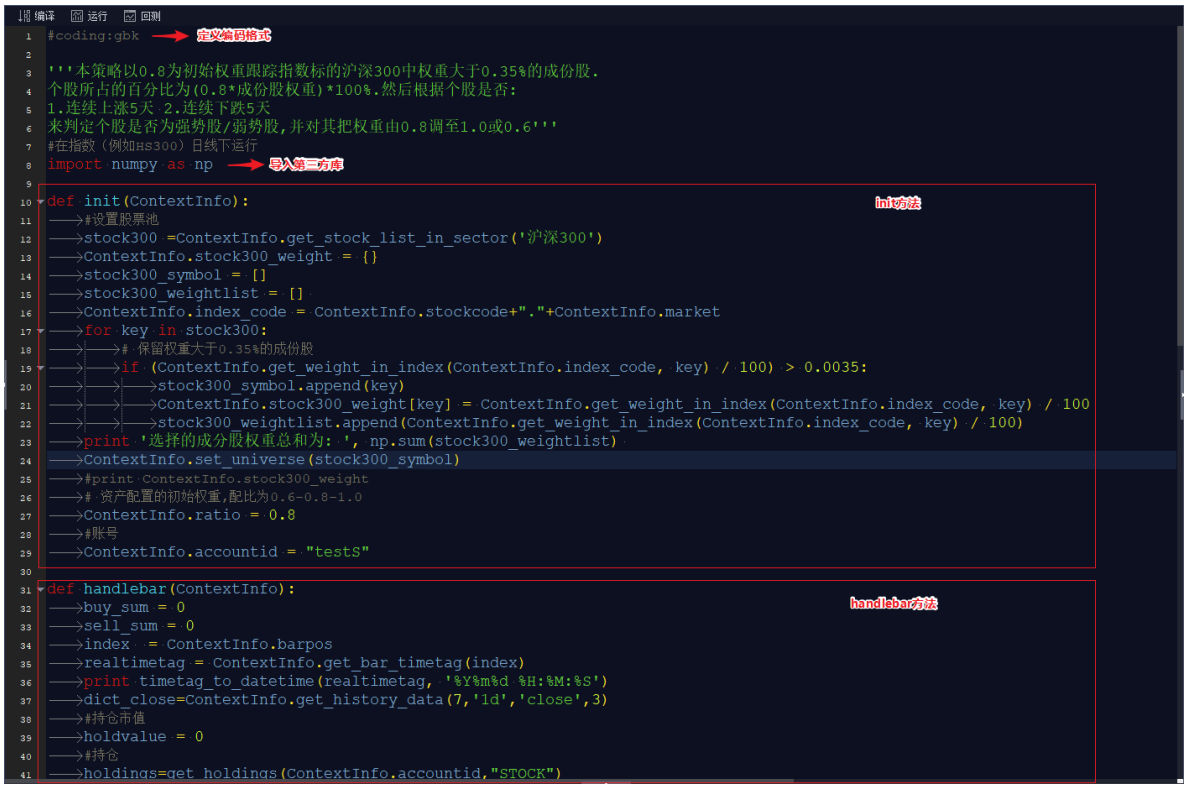


模型编辑页面 右侧可选择策略默认的周期、品种

编写 Python 策略需在开始时定义编码格式，如 gbk。

之后可选择导入第三方库，所选第三方库要在客户端白名单内才可运行。

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



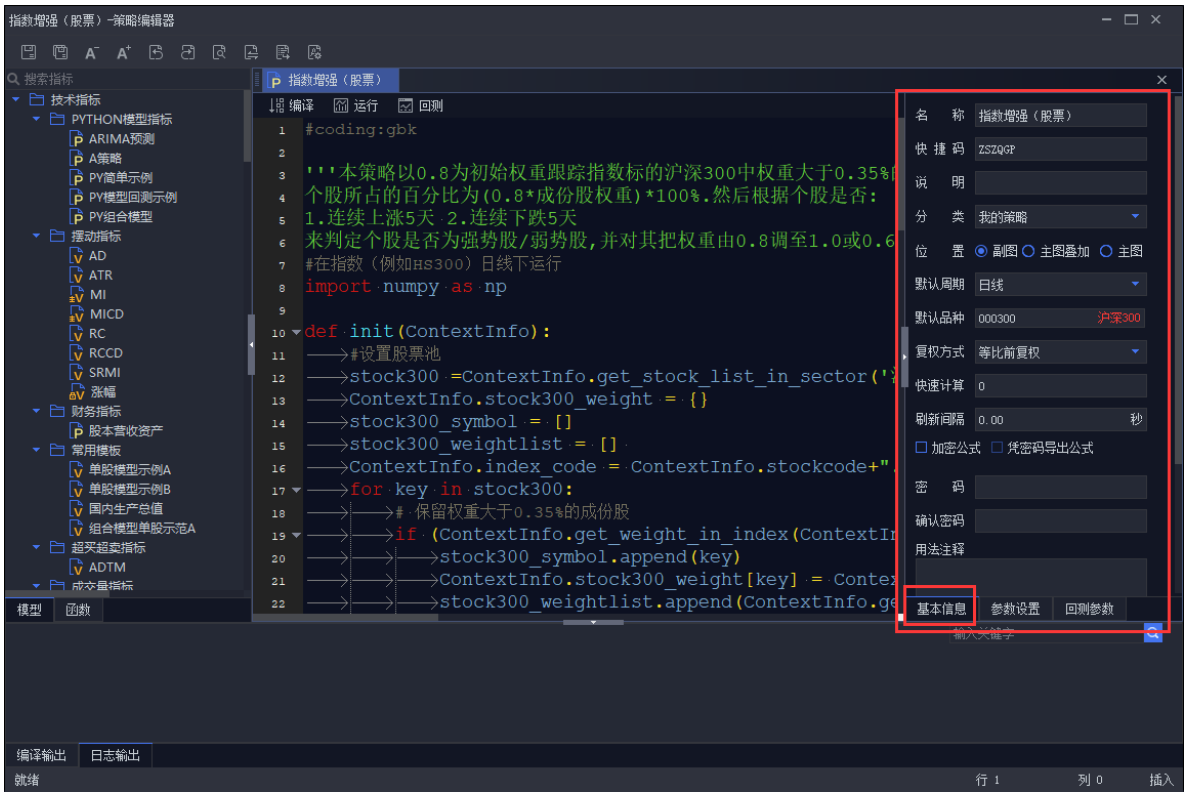
Handlebar 方法会在历史 K 线上逐 K 线调用，系统会保存函数所做更改。

在盘中交易时间，handlebar 函数会随行情推送（分笔数据）调用，当一个分笔数据为所在 K 线最后一个分笔时，此分笔调用的 handlebar 所做的更改会被系统保存，如有交易指令，会在下一个分笔时发送；其他分笔可以打印运行结果，但 handlebar 所做更改不会被保存，也不会发送交易信号。

编写创建完模型后，对应模型的基本信息和回测参数进行设置。

基本信息包括：

> **名称**：填写模型名称
>
> **快捷码**：默认根据模型名称自动生成拼音首字母拼写，如需自定义可以手动进行更改，用于键盘精灵快速引用模型
>
> **说明**：简单的说明模型功能
>
> **分类**：保存当前模型到某个分类下面
>
> **位置**：模型回测或运行时的位置，有副图、主图叠加、主图三种显示位置
>
> **默认周期**：点击模型回测或运行时的默认主图周期，可手动切换
>
> **默认品种**：点击模型回测或运行时的默认主图品种，可手动切换
>
> **复权方式**：提供不复权、前复权、后复权、等比前复权、等比后复权 5 种复权方式
>
> **快速计算**：限制计算范围，默认为 0 时模型运行会从模型设置的默认品种（主图）的第一根 K 线开始计算，设置为 n 则从当前 K 线再往前 n 个 K 线开始计算
>
> **刷新间隔**：用来设置策略运行的时间间隔。设置了刷新间隔，即每隔一段时间策略按照当前行情运行一次
>
> **加密公式**：加密后的公式只有输入密码才可以查看源代码
>
> **凭密码导出公式**：此项只有在开启 “加密公式” 后才能生效，生效后只能使用密码导出到本地
>
> **用法注释**：简短的说明模型使用的一些注意项，可不填

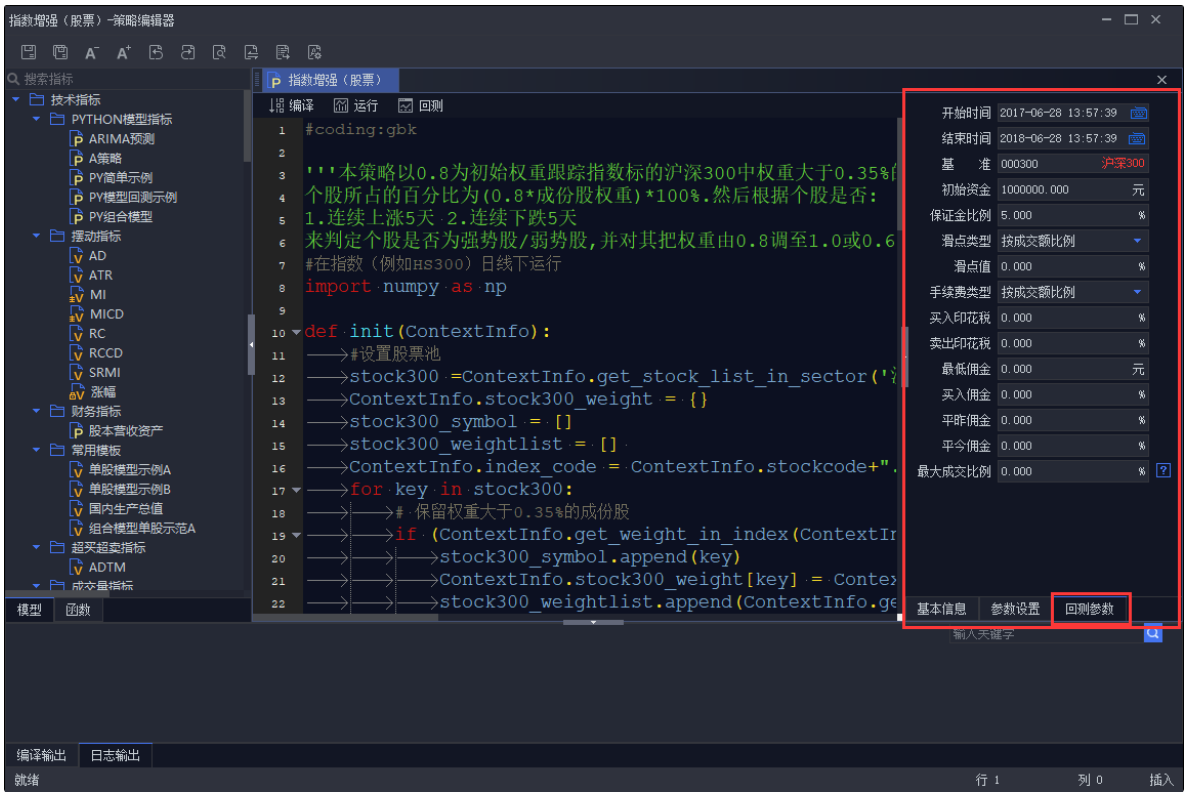


策略编辑器-基本信息

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

回测参数包括：

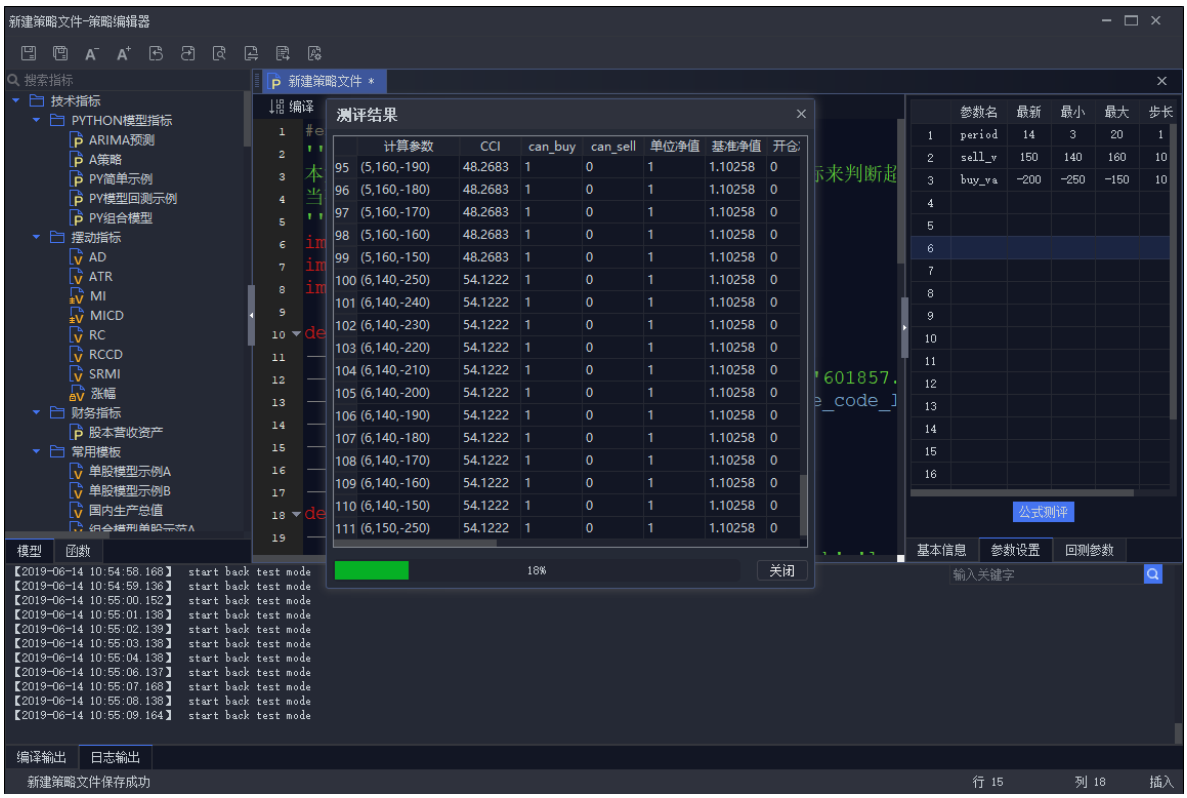
> **开始时间、结束时间**：设置模型回测时间区间
>
> **基准**：设置模型收益的参考基准
>
> **初始资金**：设置模型回测的初始资金
>
> **保证金比例**：设置期货的保证金比例
>
> **滑点**：设置回测撮合时的滑点，模拟真实交易的冲击成本
>
> **手续费类型**：支持按成交额比例或者固定值计算手续费
>
> **买入印花税**：设置买入印花税比例
>
> **卖出印花税**：设置卖出印花税比例
>
> **最低佣金**：设置单笔交易的最低佣金数额
>
> **买入佣金**：设置买入标的时的佣金比例
>
> **平昨佣金**：设置股票、期货平昨佣金比例
>
> **平今佣金**：设置期货平金佣金比例
>
> **最大成交比例**：控制回测中最大成交量不超过同期成交量*最大成交比例。可以点击此参数旁边的'？'按钮了解详情



策略编辑器-回测参数

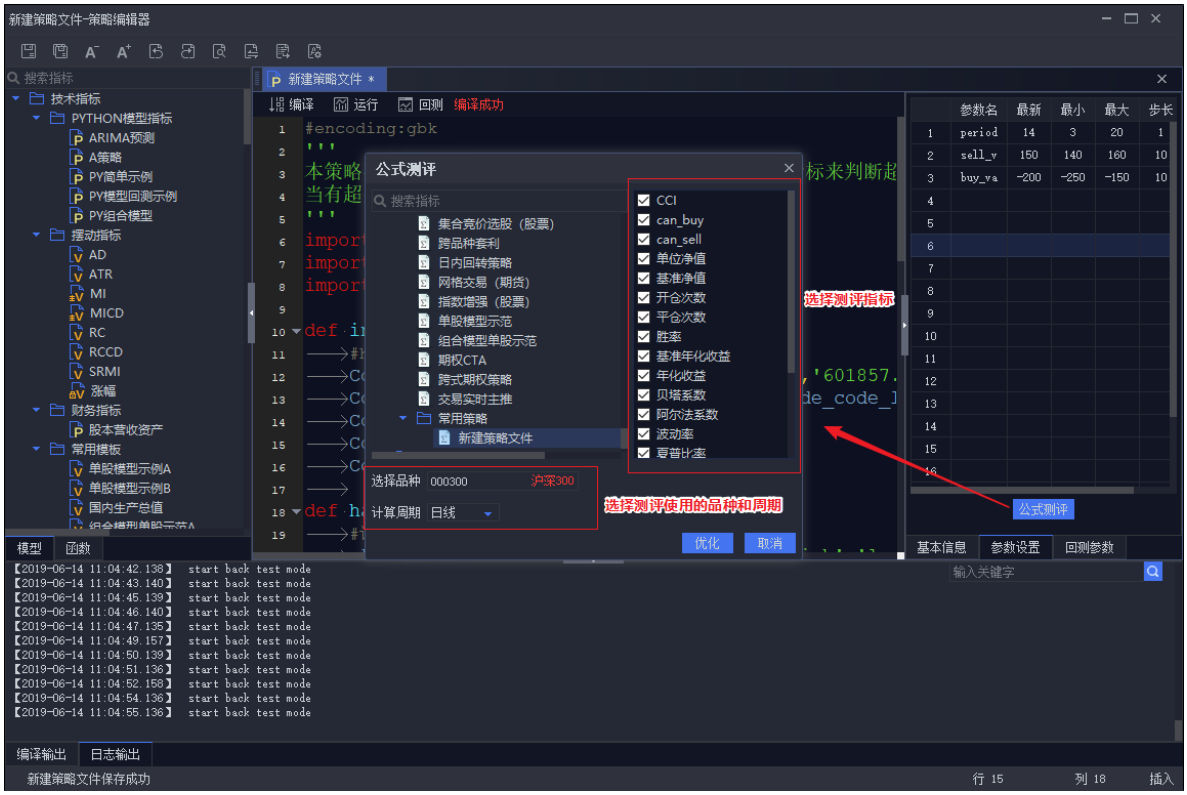
使用者也可在参数设置中设置好参数值，参数名为变量名，模型中可以调用。最新值为变量默认值，运行/回测模式使用。

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



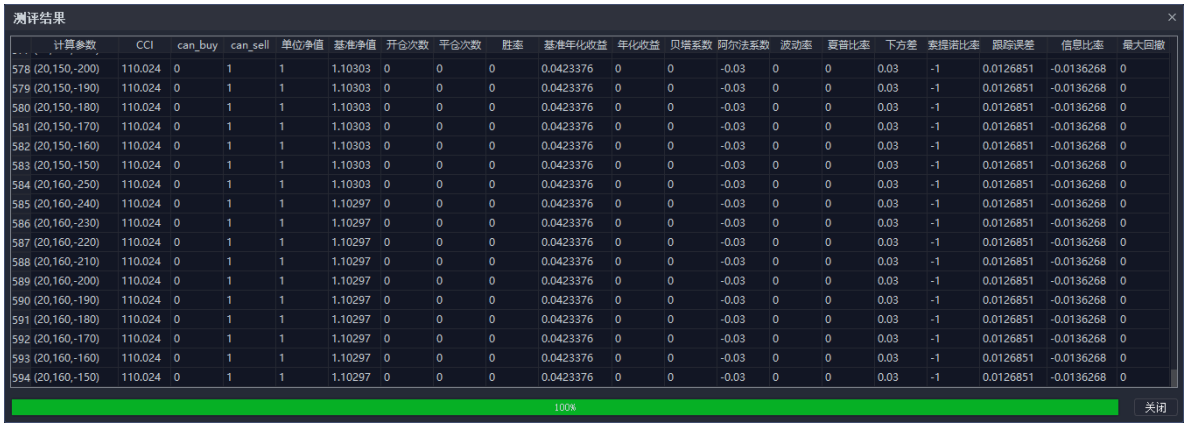
策略编辑器-参数设置

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



策略编辑器-公式评测

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

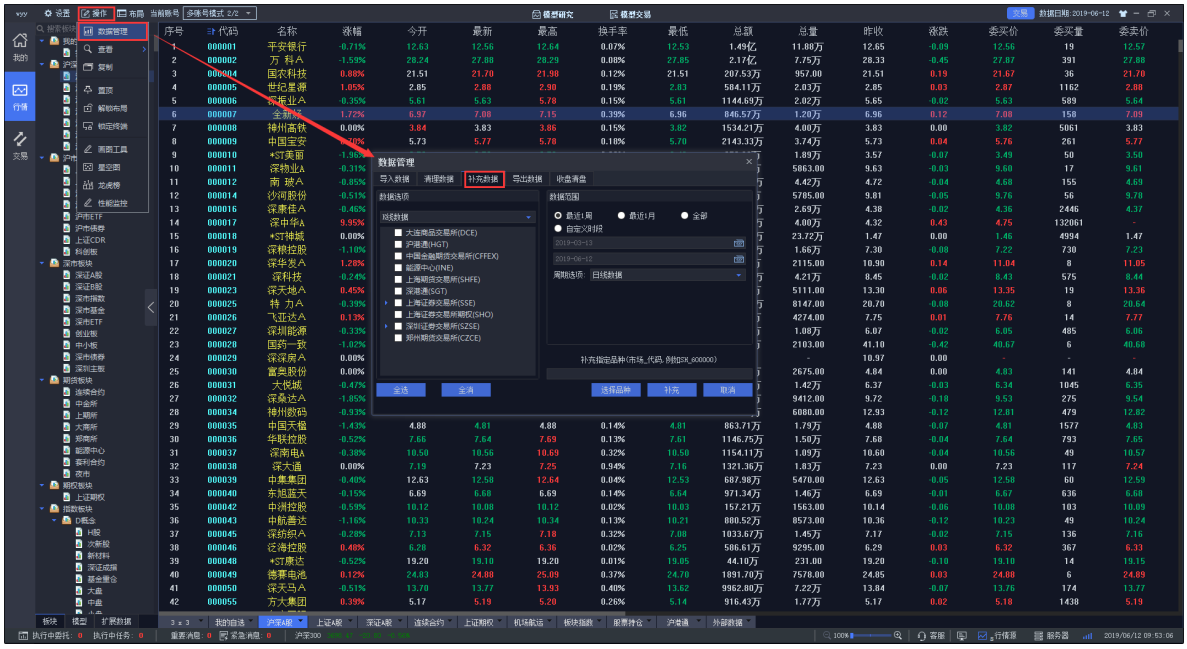


	计算参数	CCI	can_buy	can_sell	单位净值	基准净值	开仓次数	平仓次数	胜率	基准年化收益	年化收益	贝塔系数	阿尔法系数	波动率	夏普比率	下方差	索提诺比率	跟踪误差	信息比率	最大回撤
578	(20,150,-200)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268	0
579	(20,150,-190)	110.024	0	1	1	1.10303	0	0	0	0.0423376	0	0	-0.03	0	0	0.03	-1	0.0126851	-0.0136268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

2.3. 补充数据

在创建用户的模型之前，用户应使用客户端提供的"数据管理'功能，选择

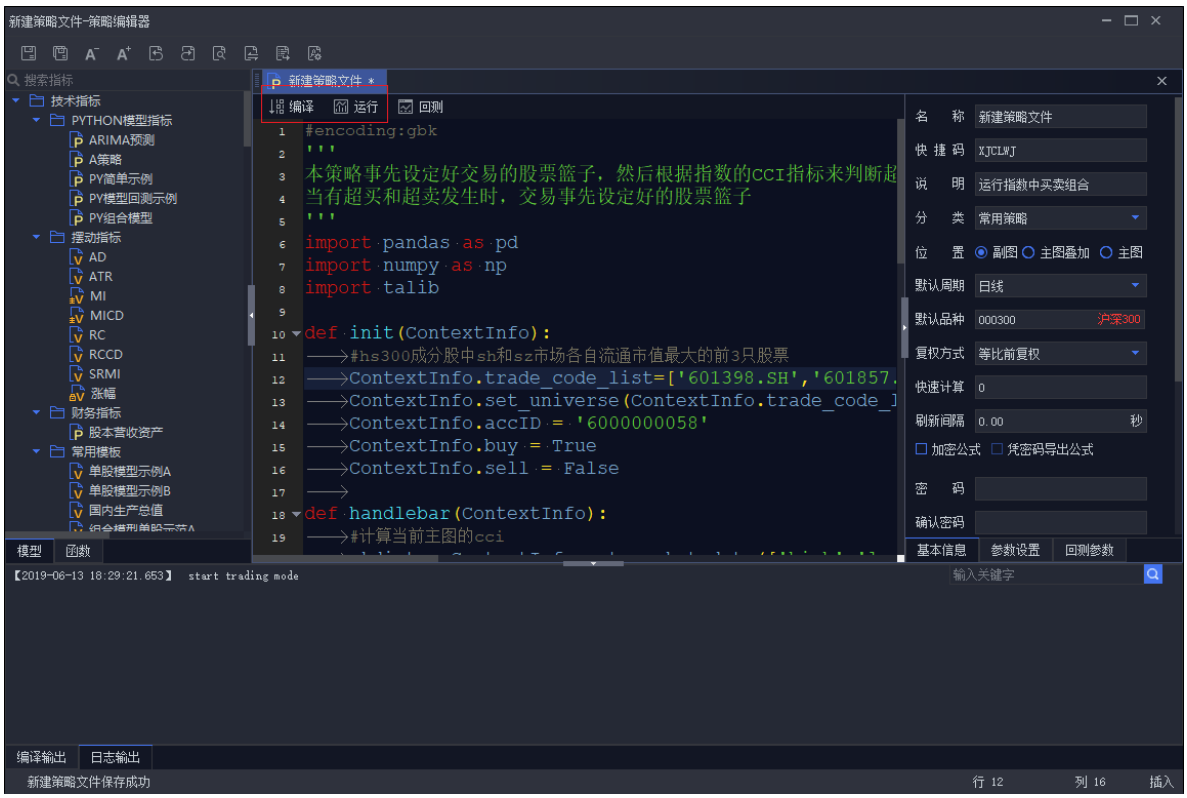
并补充模型所需的相应市场、品种以及对应周期的历史数据。



操作-数据管理

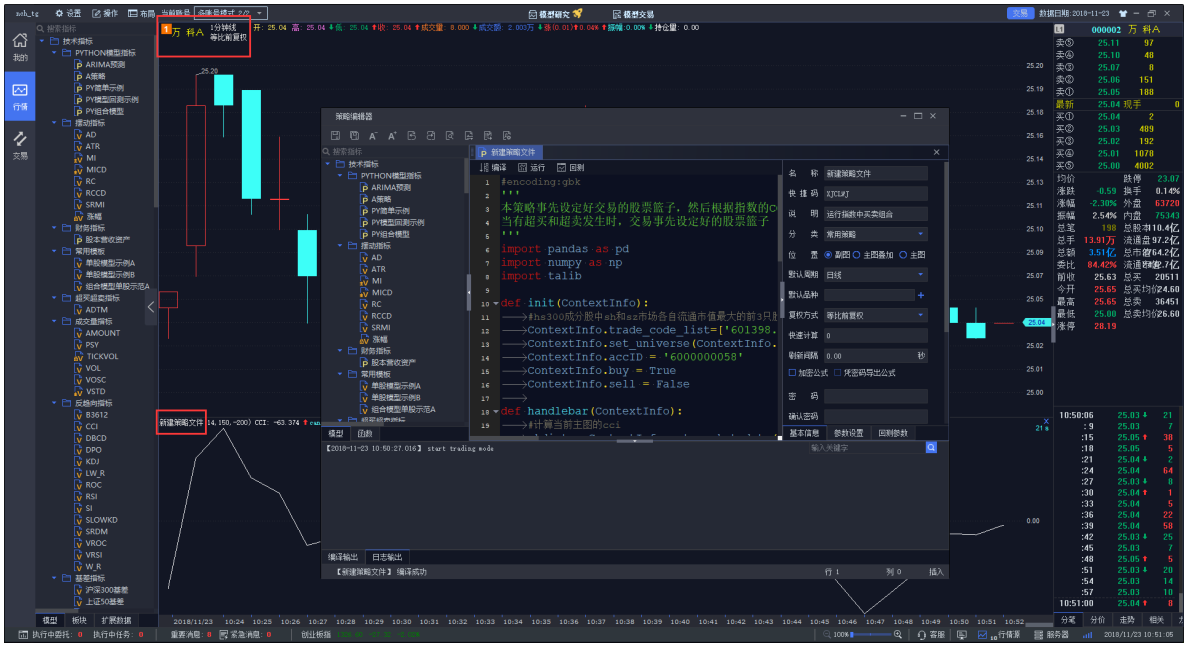
2.4. 策略运行

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



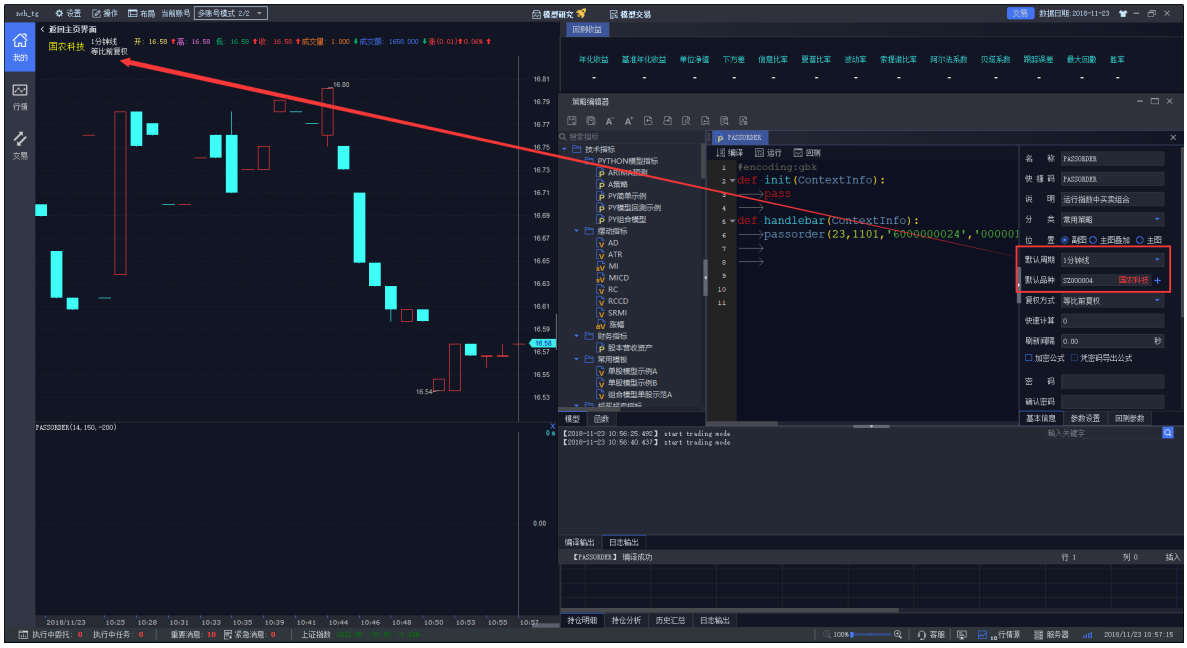
策略编辑器-运行

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



策略运行状态之一

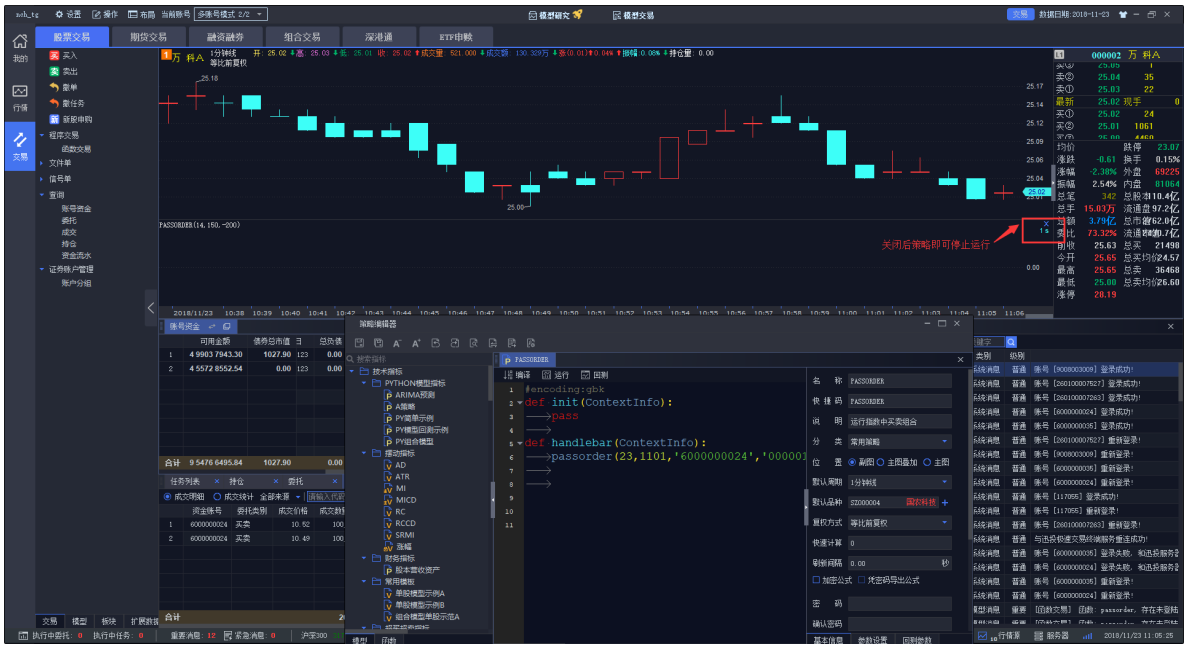
如系统当前界面处于“我的”界面或“策略开发”和“模型交易”等非行情界面，点击运行时，会基于策略编辑器-基本信息中所设置的默认周期和默认品种运行。



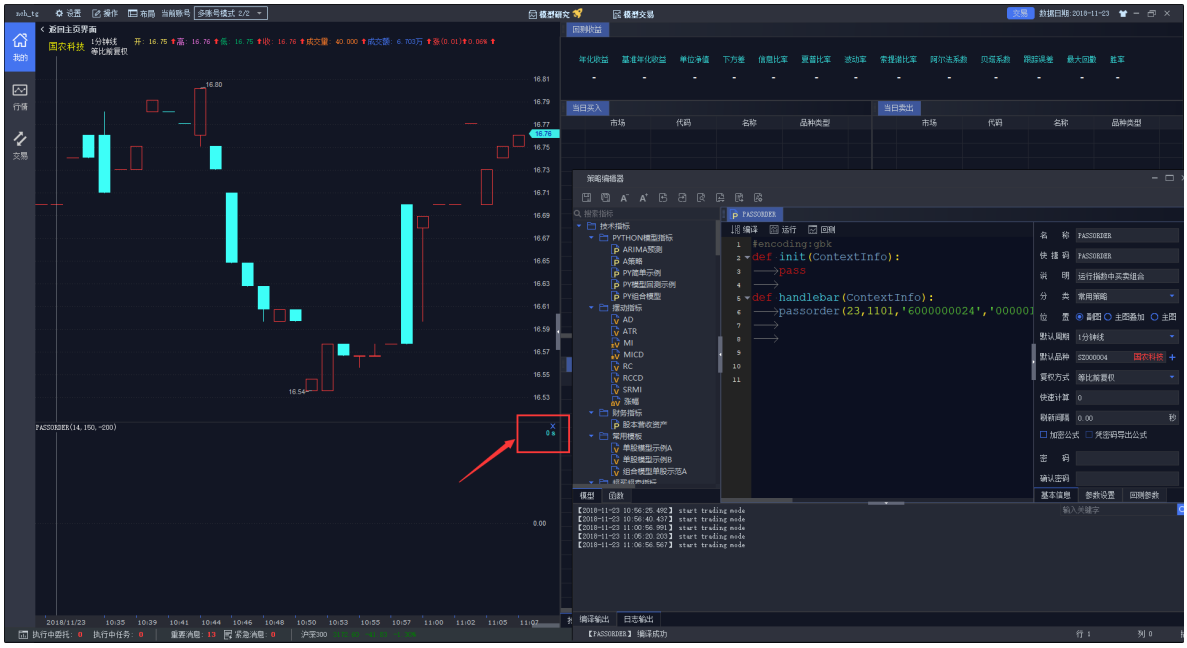
策略运行状态之二

当选择的运行位置为副图时， 

如想关闭策略，将主图下方策略运行的附图关闭即可。



关闭运行中的策略之一



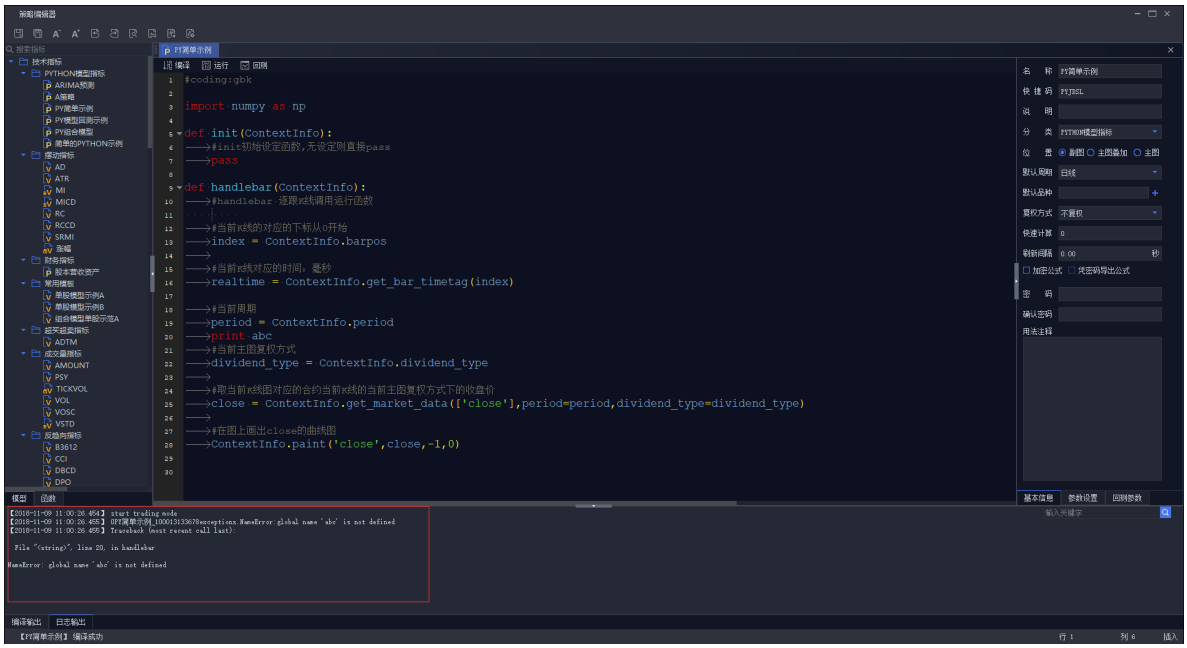
关闭运行中的策略之二

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

当选择的运行位置为主图时，键盘精灵输入KLINE即可结束模型运行。

2.5. 策略调试

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



策略调试-输出日志

2.6. 策略回测

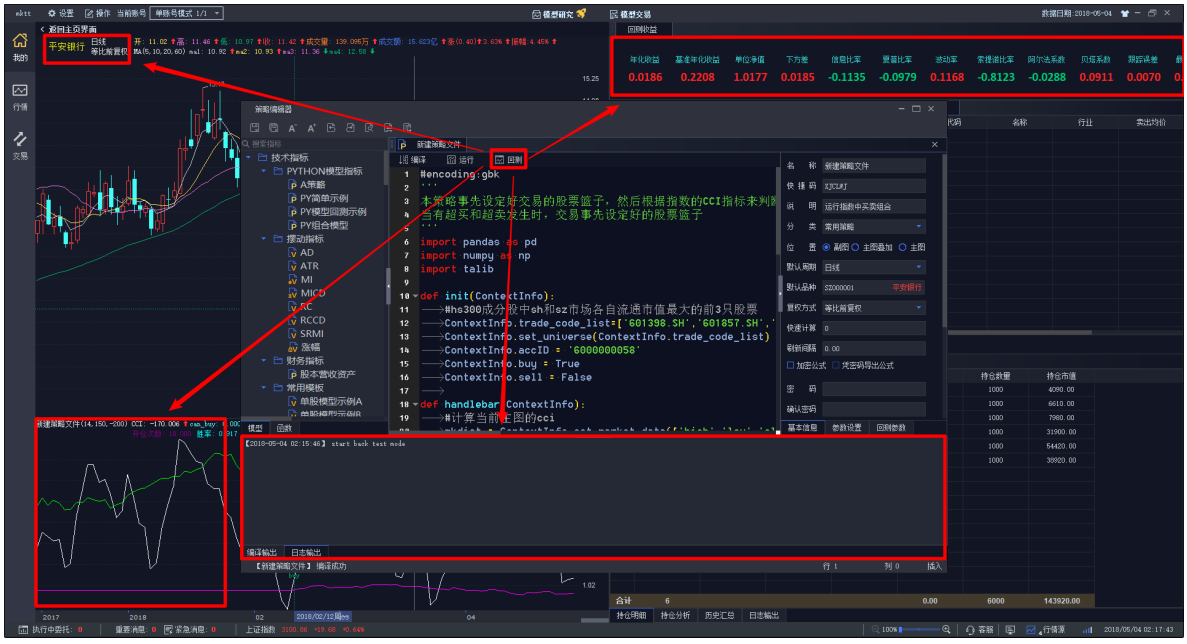
对某一策略编译成功后，点击回测，可以通过日志输出查看模型基于历史行情数据回测情况和表现。

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

用户在回测前，需根据此策略回测运行的主图、周期和时间，在【数据管理】中对行情数据进行下载补充。如回测时间设置为 20180930 至 20190530 ，运行主图为 SZ.000001 平安银行日线，补充数据时可进行如下设置。

操作-数据管理

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



策略编辑器-模型回测

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

以上列表均可鼠标右键复制和导出数据。

回测结果分析-回测结果随十字光标移动动态展示

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

回测结果买入

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

回测结果分析-附图指标输出

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

> **持仓分析**：可查看光标所在当天持仓的行业分布，展示在相关行业的市值情况、盈利情况、权重以及股票数量情况，鼠标右键可以复制和导出数据；可切换对比基准，和模型持仓进行对比。
>
> **历史板块汇总**：可查看模型自回测日期以来到光标所在日期该模型交易标的的汇总信息，包括累计盈亏、累计交易量、累计交易额、持仓天数等；点击选择板块，可以自行选择其常用板块进行板块的各项数据累计汇总；汇总数据均可以进行排序，鼠标右键可以复制和导出数据。
>
> **操作明细**：可查看模型回测的历史每一笔交易的明细。
>
> **日志输出**：可用于调试输出模型回测和运行情况。

回测结果分析-绩效分析、当日买入、当日卖出、当日持仓


持仓分析
行业分类 申万一级行业
分类名 权重(%) 股票数量 盈利 市值
1 SW家用电器成份 37.81 1 -1220.00 54420.00
2 SW房地产成份 22.17 1 -4570.00 31900.00
3 SW电子成份 27.04 1 380.00 38920.00
4 SW采掘成份 5.54 1 -280.00 7980.00
5 SW银行成份 7.43 2 -910.00 10700.00
合计 100.00% 6 -6600.00 143920.00
权重分布
家用电器 37.81%
电子 27.04%
采掘 5.54%
银行 7.43%
房地产 22.17%
基准 沪深300
行业分类 申万一级行业
行业分析
基金
基准
40.00%
30.00%
20.00%
10.00%
0%
37.81%
4.95%
27.04%
5.27%
22.17%
5.33%
7.43%
17.00%
5.54%
1.72%
家用电器
电子
房地产
银行
采掘
持仓明细
持仓分析
历史汇总
日志输出

回测结果分析-持仓分析

历史品种汇总

高级查询

	市场	代码	名称	品种类型	行业	多空	累计盈亏	累计交易量	累计
1	SH	601288	农业银行	股票	银行	-	563.85	3000	
2	SH	601398	工商银行	股票	银行	-	1101.41	3000	
3	SH	601857	中国石油	股票	采掘	-	50.23	3000	
4	SZ	000002	万 科A	股票	房地产	-	2031.56	3000	
5	SZ	000333	美的集团	股票	家用电器	-	4973.37	3000	
6	SZ	002415	海康威视	股票	电子	-	8953.53	3000	
合计		6					17673.96	18000	

历史板块汇总

选择板块 清空 高级查询

	代码	板块名称	盈利	交易费用	市值	效益权重	持仓天数	买卖累计	盈
1	SW采掘成份	SW采掘成份	50.23	0.00	23730.15	0.00%	56	3000	
2	SW化工成份	SW化工成份	0.00	0.00	0.00	0.00%	0	0	
3	SW钢铁成份	SW钢铁成份	0.00	0.00	0.00	0.00%	0	0	
4	SW有色金属成份	SW有色金属成份	0.00	0.00	0.00	0.00%	0	0	
5	SW建筑材料成份	SW建筑材料成份	0.00	0.00	0.00	0.00%	0	0	
6	SW建筑装饰成份	SW建筑装饰成份	0.00	0.00	0.00	0.00%	0	0	
7	SW电气设备成份	SW电气设备成份	0.00	0.00	0.00	0.00%	0	0	
8	SW机械设备成份	SW机械设备成份	0.00	0.00	0.00	0.00%	0	0	
9	SW国防军工成份	SW国防军工成份	0.00	0.00	0.00	0.00%	0	0	
10	SW汽车成份	SW汽车成份	0.00	0.00	0.00	0.00%	0	0	
11	SW家用电器成份	SW家用电器成份	4973.37	0.00	127926.63	0.00%	56	3000	
12	SW纺织服装成份	SW纺织服装成份	0.00	0.00	0.00	0.00%	0	0	
13	SW轻工制造成份	SW轻工制造成份	0.00	0.00	0.00	0.00%	0	0	
14	SW商业贸易成份	SW商业贸易成份	0.00	0.00	0.00	0.00%	0	0	
15	SW农林牧渔成份	SW农林牧渔成份	0.00	0.00	0.00	0.00%	0	0	
16	SW食品饮料成份	SW食品饮料成份	0.00	0.00	0.00	0.00%	0	0	
合计	28		17673.96		342288.97				

持仓明细 持仓分析 历史汇总 日志输出

回测结果分析-历史品种汇总、历史板块汇总

回测结果分析-操作明细、日志输出

2.7. 回测、运行两种模式的区别

回测模式在【策略开发】下进行，运行模式在【模型交易】下进行。

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

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

3. Python API 手册

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

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

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

（4）采用的是所见即所得的策略运行机制，完美展示策略运行结果。

3.1. 对第三方库的支持

国信 Python API 是基于 **Python 3.X** 规范的标准量化投资策略应用程序接口。我司主要通过以下两种方式对外提供：

3.1.1. 系统自带的 Python 环境

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

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

3.1.2. 用户自行安装 Python 环境

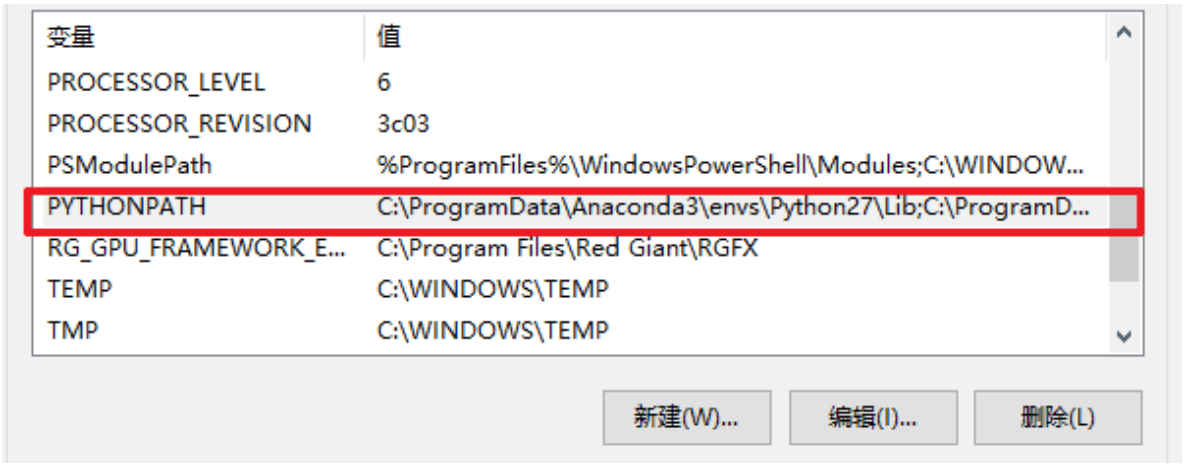
对于有经验的Python开发者来说，国信提供了自行引入第三方库的方式。为了引入额外的第三方库，用户需要做如下一些操作：

（1）删除国信iQuant系统自带的Python库；

（2）安装对应的 Python 3.6 版本；

（3）通过pip或者其它方式安装所需的第三方库。

用户使用自己安装的 Python 解释器时，需在系统变量中新增一个名为 PYTHONPATH 的环境变量，这个环境变量中至少要加两个路径，一个是 Lib，一个是 DLLs。



经过以上几个步骤后，用户就可以在国信的 Python 编辑器中使用已经安装好的 Python 第三方库。

3.2. 使用说明

3.2.1. 概述

3.2.1.1. 一般规定

在编写一个策略时，首先需要在代码的最前一行写上：

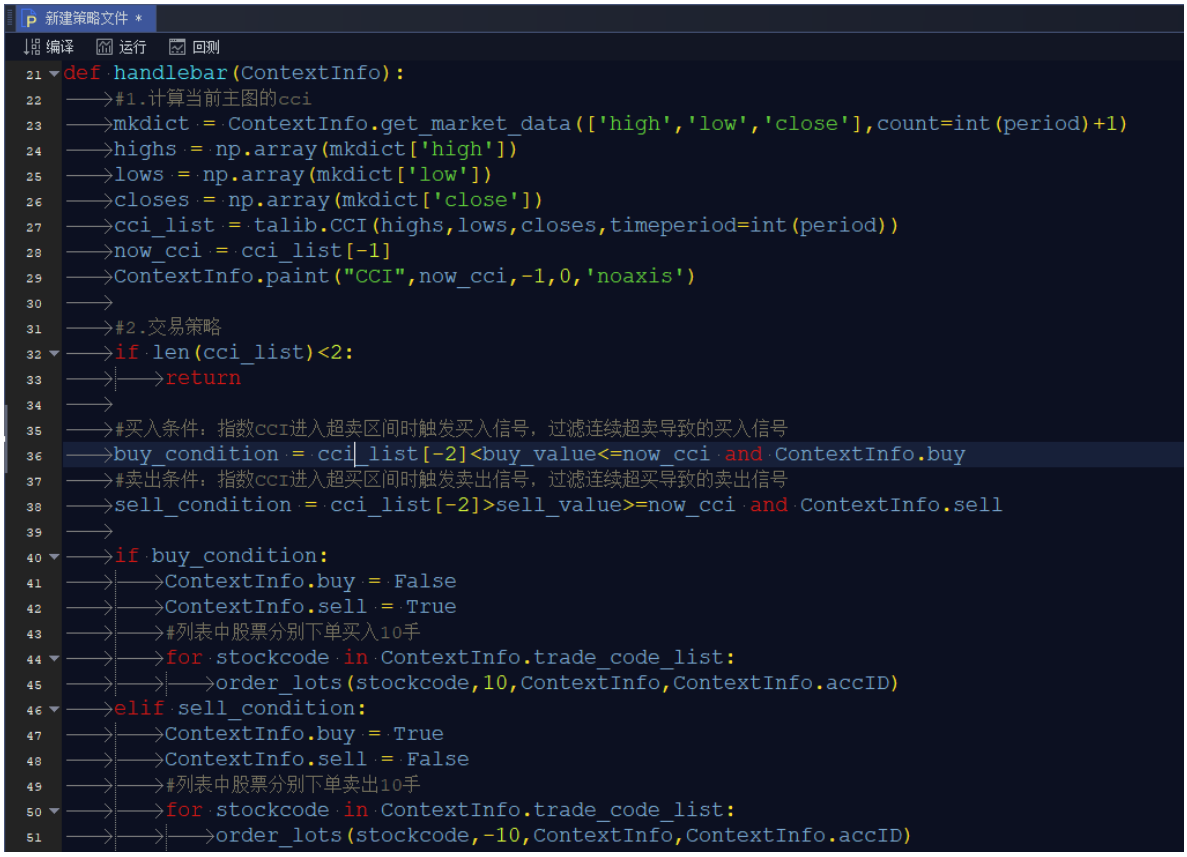
#coding:gbk 统一脚本的编码格式是GBK;

3.2.1.2. 重要方法

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

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

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



Python策略中必须有包含如下两个基本方法，否则策略将运行不起来。

（1）初始化函数 init()

用法：init(ContextInfo)

释义：初始化函数，只在整个策略开始时调用运行一次

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

返回：无

示例：

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

（2）行情事件函数 handlebar()

用法：handlebar(ContextInfo)

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

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

返回：无

示例：

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

3.2.2. ContextInfo对象

ContextInfo 是策略运行环境对象，是 init() 和 handlebar() 这两个基本方法必传参数，里面包括了终端自带的属性和方法，还可以添加自定义属性。

*注：除特殊标明外，以下函数均支持回测和实盘/模拟运行模式。

（1）设定股票池 ContextInfo.set_universe()

用法：ContextInfo.set_universe(stocklist)

释义：设定股票池

参数：list

返回：无

示例：

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

（2）设定交易账号 ContextInfo.set_account()

用法：ContextInfo.set_account(account)

释义：设定交易账号

参数：string

返回：无

示例：

```
def init(ContextInfo):
    account = '6000000223'
    ContextInfo.set_account(account)
```

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

用法：ContextInfo.start / ContextInfo.end

释义：设定回测起止时间，读写

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

参数：无

返回：无

示例：

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

（4）设定回测初始资金 ContextInfo.capital

用法：ContextInfo.capital

释义：设定回测初始资金，读写，默认为 1000000

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

参数：无

返回：number

示例：

```
def init(ContextInfo):
    ContextInfo.capital = 10000000

def handlebar(ContextInfo):
    print(ContextInfo.capital)
```

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

用法：ContextInfo.set_slippage(slippageType, slippage)

释义：设定策略回测滑点，默认值 0.00

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

参数：

- slippageType：滑点类型，可选值：

  > 0：tick跳数设置滑点
  >
  > 1：按照固定值（价格）设置滑点
  >
  > 2：价格比例设置滑点

- slippage：滑点值

返回：无

示例：

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

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

用法：ContextInfo.set_commission(commissionType, commissionList)

释义：设定策略回测各种手续费率，默认类型值 0 按比例，默认值 0.000

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

参数：

- commissionType：number，可选值：

  > 0：按比例
  >
  > 1：按每手（股）

- commissionList：list，包含六个值，commissionList = [open_tax, close_tax, open_commission, close_commission, close_tdaycommission, min_commission]

  > open_tax：买入印花税
  >
  > close_tax：卖出印花税
  >
  > open_commission：开仓手续费;
  >
  > close_commission：平仓（平昨）手续费
  >
  > close_tdaycommission：平今手续费
  >
  > min_commission：最少手续费

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

返回：无

示例1：

```
def init(ContextInfo):
    # 万三
    commission = 0.0003

    # 设定开仓手续费和平仓手续费以及平今手续费均为 0.0003，其余为 0
    ContextInfo.set_commission(commission)
```

示例2：

```
def init(ContextInfo):
    commissionList = [0,0.0001, 0.0003, 0.0003, 0, 5]

    # 设定买入印花税为 0，卖出印花税为 0.0001，开仓手续费和平仓（平昨）手续费均为万三，平今手续费为 0，最小手续费为 5
    ContextInfo.set_commission(0, commissionList)
```

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

用法：ContextInfo.get_universe()

释义：获取股票池中的股票

参数：无

返回：list

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_universe())
```

（8）获取当前周期 ContextInfo.period

用法：ContextInfo.period

释义：获取当前周期，即基本信息中设置的默认周期，只读

参数：无

返回：string，返回值含义：

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

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.period)
```

（9）获取当前运行到 K 线索引号 ContextInfo.barpos

用法：ContextInfo.barpos

释义：获取当前运行到 K 线索引号，只读，索引号从0开始

参数：无

返回：number

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.barpos)
```

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

用法：ContextInfo.time_tick_size

释义：获取当前图 K 线数目，只读

参数：无

返回：number

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.time_tick_size)
```

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

用法：ContextInfo.is_last_bar()

释义：判定是否为最后一根 K 线

参数：无

返回：bool，返回值含义：

> True：是
>
> False：否

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.is_last_bar())
```

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

用法：ContextInfo.is_new_bar()

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

参数：无

返回：bool，返回值含义：

> True：是
>
> False：否

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.is_new_bar())
```

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

用法：ContextInfo.is_suspended_stock(stockcode.market)

释义：判定股票是否停牌

参数：股票市场和代码

返回：bool，返回值含义：

> True：停牌
>
> False：未停牌

示例：

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

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

用法：is_sector_stock(sectorname, market, stockcode)

释义：判定给定股票代码是否在指定的板块中

参数：

- sectorname：string，板块名
- market：string，市场
- stockcode：string，股票代码

返回：number，返回值含义：

> 1：True
>
> 0：False

示例：

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

（15）判定给定股票是否属于某个类别 is_typed_stock()

用法：is_typed_stock(stocktypenum, market, stockcode)

释义：判定给定股票是否属于某个类别

参数：

- stocktypenum：number，类别标号，如股票类别标号为100003，详细见5.2. 附录3 is_typed_stock 函数证券分类表，或者见 xtstocktype.lua 文件

- market：string，市场
- stockcode：string，股票代码

返回：number，返回值含义：

> 1：True
>
> 0：False

示例：

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

（16）判定给定股票代码是否在指定的行业分类中 get_industry_name_of_stock()

用法：get_industry_name_of_stock(industryType, stockcode)

释义：判定给定股票代码是否在指定的行业分类中

参数：

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

返回：string：对应行业名，找不到则返回空 string

示例：

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

（17）获取当前图代码 ContextInfo.stockcode

用法：ContextInfo.stockcode

释义：获取当前图代码，只读

参数：无

返回：string：对应主图代码

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.stockcode)
```

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

用法：ContextInfo.dividend_type

释义：获取当前主图复权处理方式

参数：无

返回：string，返回值含义：

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

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.dividend_type)
```

（19）获取当前主图市场 ContextInfo.market

用法：ContextInfo.market

释义：获取当前主图市场，只读

参数：无

返回：string：对应主图市场

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.market)
```

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

用法：ContextInfo.get_stock_name('stockcode')

释义：根据代码获取名称

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

返回：string（GBK编码）

示例：

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

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

用法：ContextInfo.do_back_test

释义：设定是否开启回测模式，只读，默认值为 False

参数：无

返回：bool

（22）获取回测基准 ContextInfo.benchmark

用法：ContextInfo.benchmark

释义：获取回测基准

*注：此函数只支持回测模式。

参数：无

返回：string

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.benchmark)
```

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

用法：ContextInfo.data_info_level

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

*注：此函数只支持回测模式。

参数：无

返回：无

示例：

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

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

用法：get_result_records (recordtype, index, ContextInfo)

释义：获取某个记录类型对应的某个时刻的记录情况。

*注：模型回测时有效，获取的为回测面板中的记录结果

参数：

- Recordtype：string，面板类型，可选值：

  > 'buys'：买入持仓
  >
  > 'sells'：卖出持仓
  >
  > 'holdings'：当前持仓

> 'historysums'：历史汇总
>
> 'dealdetails'：交易细明

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

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

> market：string，市场代码
>
> stockcode：string，合约代码
>
> open_close：number，期货开平：1 开 0 平；股票： 1 买 0 卖
>
> direction：number，方向：1 多 -1 空；回购：1 逆回购 -1 正回购
>
> trade_price：number，成交价格
>
> current_price：number，最新价，持仓中用这个价
>
> profit：number，持仓盈亏
>
> position：number，仓位数量
>
> current_weight：number，仓位权重[暂无有效数据]
>
> benefit_weight：number，盈利占比权重，记录类型是 'historysums' 时有效
>
> holding_periods：number，累计持仓天数，记录类型是 'historysums' 时有效
>
> buy_sell_times：number，交易次数，记录类型是 'historysums' 时有效
>
> commission：number，手续费
>
> trade_balance：number，成交额或市值
>
> operate_type：number，操作类型，记录类型是 'dealdetails' 时有效
>
> trade_date：number，交易日期，毫秒标记，记录类型是 'dealdetails' 时有效

示例：

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

3.2.3. 获取数据

*注：除特殊标明外，以下函数均支持回测和实盘/模拟运行模式。

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

用法：ContextInfo.get_last_volume(stockcode)

释义：获取最新流通股本

参数：string：必须是 'stock.market' 形式

返回：number

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_last_volume('000002.SZ'))
```

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

用法：ContextInfo.get_bar_timetag(index)

释义：获取当前 K 线对应时间的时间戳

参数：number：K 线索引号

返回：number

示例：

```
def handlebar(ContextInfo):
    index = ContextInfo.barpos
    print(ContextInfo.get_bar_timetag(index))
```

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

用法：ContextInfo.get_tick_timetag()

释义：获取当前主图品种最新分笔对应的时间的时间戳

参数：无

返回：number

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_tick_timetag())
```

（4）获取指数成份股 ContextInfo.get_sector()

用法：ContextInfo.get_sector(sector, realtime)

释义：获取板块成份股，只支持取指数成份股

参数：

- string：必须是 'stock.market' 形式，如 '000300.SH' ，可取如沪深300（000300.SH）、中证500（000905.SH）、上证50（000016.SH）等指数的历史成份股
- realtime：毫秒级时间戳

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

示例：

```
def handlebar(ContextInfo):
    index = ContextInfo.barpos
    realtime = ContextInfo.get_bar_timetag(index)
    print(ContextInfo.get_sector('000300.SH', realtime))
```

（5）获取行业成份股 ContextInfo.get_industry()

用法：ContextInfo.get_industry(industry)

释义：获取行业成份股，证监会行业，行业列表详见附录2

参数：string：如 'CSRC采矿业'

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

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_industry('CSRC采矿业'))
```

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

用法：ContextInfo.get_stock_list_in_sector(sectorname, realtime)

释义：获取板块成份股，支持客户端左侧板块列表中任意的板块，包括自定义板块

参数：

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

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

示例：

```
def handlebar(ContextInfo):
    index = ContextInfo.barpos
    realtime = ContextInfo.get_bar_timetag(index)
    print(ContextInfo.get_stock_list_in_sector('沪深300', realtime))
```

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

用法：ContextInfo.get_weight_in_index(indexcode, stockcode)

释义：获取某只股票在某指数中的绝对权重

参数：

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

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

示例：

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

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

用法：ContextInfo.get_contract_multiplier(contractcode)

释义：获取合约乘数

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

返回：number

示例：

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

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

用法：ContextInfo.get_risk_free_rate(index)

释义：获取无风险利率，用十年期国债收益率CGB10Y作无风险利率

参数：number：当前图 K 线索引号

返回：number

示例：

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

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

用法：ContextInfo.get_date_location(strdate)

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

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

返回：number

示例：

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

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

用法：ContextInfo.get_slippage()

释义：获取策略设定的滑点

*注：此函数只支持回测模式。

参数：无

返回：dict，key 包括：

> slippage_type
>
> slippage

示例：

```
def init(ContextInfo):
    ContextInfo.set_slippage(0.003)

def handlebar(ContextInfo):
    print(ContextInfo.get_slippage())
```

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

用法：ContextInfo.get_commission()

释义：获取策略设定的各种手续费率

*注：此函数只支持回测模式。

参数：无

返回：dict，key 包括

> commission_type
>
> open_tax
>
> close_tax
>
> open_commission
>
> close_commission
>
> close_tdaycommission
>
> min_commission

示例：

```
def init(ContextInfo):
    ContextInfo.set_commission(0.0003)

def handlebar(ContextInfo):
    print(ContextInfo.get_commission())
```

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

用法：ContextInfo.get_net_value(index)

释义：获取策略回测的净值

*注：此函数只支持回测模式。

参数：index = ContextInfo.barpos

返回：number

示例：

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

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

用法：ContextInfo.get_financial_data(fieldList, stockList, startDate, endDate)

释义：获取财务数据

参数：

- fieldList：字段列表，如 ['CAPITALSTRUCTURE.total_capital', 'ASHAREINCOME.net_profit_incl_min_int_inc'] （例子中取了股本结构中的总股本，与利润表中的净利润），更多支持字段参见4. 财务数据接口使用方法
- stockList：股票列表，如 ['600000.SH', '000001.SZ']
- startDate：开始时间，如 '20171209'
- endDate：结束时间，如 '20171212'

返回：

> 代码1-时间1：pandas.Series index = 字段
>
> 代码1-时间n：pandas.DataFrame index = 时间, columns = 字段
>
> 代码n-时间1：pandas.DataFrame index = 代码, columns = 字段
>
> 代码n-时间n：pandas.Panel items = 代码, major_axis = 时间, minor_axis = 字段

*注：必须安装pandas

示例：

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

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

用法：ContextInfo.get_financial_data(tabname, colname, market, code, barpos) (与上一个 ContextInfo.get_financial_data 可同时使用)

释义：获取财务数据

参数：

- tabname：表格名称
- colname：字段名称
- market：市场
- code：股票代码
- barpos：当前 bar 的索引

返回：number

示例：

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

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

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

释义：获取历史行情数据

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

参数：

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

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

- field：string，需获取的历史数据的类型，可选值：

> 'open'
>
> 'high'
>
> 'low'
>
> 'close'
>
> 'quoter'（结构见get_market_data）

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

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

- skip_paused：默认参数，bool，是否停牌填充，默认填充

*注：可缺省参数：dividend_type，skip_paused

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

示例：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000300.SH', '000004.SZ'])

def handlebar(ContextInfo):
    # 获取股票池中所有股票的最近两日的收盘价
    hisdict = ContextInfo.get_history_data(2, '1d', 'close')
    for k, v in hisdict.items():
        if len(v) > 1:
            # 今日涨幅
            print(k, ':', v[1] - v[0])
```

（17）获取历史行情数据 ContextInfo.get_market_data()

用法：ContextInfo.get_market_data(fields, stock_code = [], start_time = '', end_time = '', skip_paused = True, period = '1d', dividend_type = 'none', count = -1)

释义：获取历史行情数据

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

参数：

- fields：字段列表：

> 'open'：开
>
> 'high'：高
>
> 'low'：低
>
> 'close'：收

> 'volume'：成交量
>
> 'amount'：成交额
>
> 'settle'：结算价
>
> 'quoter'：分笔数据（包括历史）

- 'quoter'分笔数据结构：dict

  {

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

  }

- stock_code ：默认参数，合约代码列表，合约格式 code.market, 如 '600000.SH'，不指定时为当前图合约
- start_time：默认参数，开始时间，格式 '20171209' 或 '20171209010101'
- end_time：默认参数，结束时间，格式 '20171209' 或 '20171209010101'
- skip_paused：默认参数，可选值：

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

- period：默认参数，周期类型：

  > 'tick'：分笔线
  >
  > '1d'：日线
  >
  > '1m'：1分钟线
  >
  > '3m'：3分钟线
  >
  > '5m'：5分钟线
  >
  > '15m'：15分钟线
  >
  > '30m'：30分钟线
  >
  > '1h'：小时线

> '1w'：周线
> '1mon'：月线
> '1q'：季线
> '1hy'：半年线
> '1y'：年线

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

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

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

  count 和开始时间、结束时间同时设置的效果如下表：

count 取值	**时间设置是否生效**	**开始时间和结束时间设置效果**
count >= 0	**无效**	从当前时间往回取多少个周期的值，取值数量完全取决于 count
count = -1	**生效**	同时设置开始时间和结束时间，在所设置的时间段内取值
count = -1	**生效**	开始时间结束时间都不设置，取当前最新bar的值
count = -1	**生效**	只设置开始时间，取所设开始时间到当前时间的值
count = -1	**生效**	只设置结束时间，取股票上市第一根 bar 到所设结束时间的值

*注：

1. 特别的，默认参数需带上参数名方可生效，调用方式与 Python 别无二致
2. 策略点击运行，以策略代码中所设置的开始时间和结束时间为准，策略点击回测，以 '回测参数' 里所设置的开始时间和结束时间为准。
3. 可缺省参数：start_time，end_time，skip_paused，dividend_type，count

返回：

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

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

代码：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000831.SZ'])

def handlebar(ContextInfo):
    df = ContextInfo.get_market_data(['close'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type =
'front')
    print(df)
```

输出：

```
17.40
```

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

代码：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000831.SZ'])

def handlebar(ContextInfo):
    df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type =
'front')
    print(df)
```

输出：

```
close 17.40
```

```
open 17.11
```

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

代码：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000831.SZ'])

def handlebar(ContextInfo):
    df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type =
'front', count = 2)
    print(df)
```

输出：

```
          close   open
```

```
20190618  17.59  17.60
```

```
20190619  17.40  17.11
```

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

代码：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000831.SZ', '600000.SH'])

def handlebar(ContextInfo):
    df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type =
'front', count = -1)
    print(df)
```

输出：

```
          close   open

000831.SZ  17.40  17.11

600000.SH  11.88  12.04
```

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

代码：

```
def init(ContextInfo):
    ContextInfo.set_universe(['000831.SZ', '600000.SH'])

def handlebar(ContextInfo):
    df = ContextInfo.get_market_data(['close', 'open'], stock_code =
ContextInfo.get_universe(), skip_paused = True, period = '1d', dividend_type =
'front', count = 2)
    print(df)
```

输出：

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

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

Items axis: 000831.SZ to 600000.SH

Major_axis axis: 20190618 to 20190619

Minor_axis axis: close to open
```

*注：

1. 需要下载历史数据；
2. pandas相关介绍详见pandas官方文档；
3. Python需要安装pandas；
4. 时间区间：两边闭区间。

5. get_history_data() 和 get_market_data() 两者区别：get_history_data() 是获取设定的股票池中股票某段行情，主要用来构建简单的指标；get_market_data() 可获取任意股票行情，返回行情数据可以是 dataframe ，可用于复杂的分析。

示例1：

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

示例2：

```
def init(ContextInfo):
    # 先设置股票池，此时股票池只有一个股票
    ContextInfo.set_universe(['000001.SZ'])

def handlebar(ContextInfo):
    # 获取平安银行当前 bar 的收盘价
    close = ContextInfo.get_market_data(['close'], stock_code =
ContextInfo.get_universe(), period = '1d', dividend_type = 'front', count = 1)

    # 打印平安银行当前bar的收盘价
    print(close)  # close 是一个数值

    # 获取平安银行最新的开高低收四个价格
    df1 = ContextInfo.get_market_data(['open', 'high', 'low', 'close'],
stock_code = ContextInfo.get_universe(), period = '1d', dividend_type = 'front',
count = 1)

    # 打印平安银行当前bar的最高价
    print(df1['high'])  # df1 是一个 pandas.series

    # 获取平安银行最近两天的高开低收
    df2 = ContextInfo.get_market_data(['open', 'high', 'low', 'close'],
stock_code = ContextInfo.get_universe(), period = '1d', dividend_type = 'front',
count = 2)

    # 打印平安银行昨天的收盘价
    print(df2['close'][-2])  # df2 是一个 pandas.dataframe
```

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

用法：ContextInfo.get_divid_factors(stock.market)

释义：获取除权除息日和复权因子

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

返回：dict，除权除息日与复权因子的映射字典

示例：

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

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

用法：ContextInfo.get_main_contract(codemarket)

释义：获取当前期货主力合约

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

返回：str，合约代码

示例：

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

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

用法：timetag_to_datetime(timetag, format)

释义：将毫秒时间转换成日期时间

参数：

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

返回：str，合约代码

示例：

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

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

用法：ContextInfo.get_total_share(stockcode)

释义：获取总股数

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

返回：number

示例：

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

（22）获取指定个股 / 合约 / 指数的 K 线（交易日）列表 ContextInfo.get_trading_dates()

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

释义：获取指定个股 / 合约 / 指数的 K 线（交易日）列表

参数：

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

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

*注：可缺省参数：start_date，end_date

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

示例：

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

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

用法：ContextInfo.get_svol(stockcode)

释义：根据代码获取对应股票的内盘成交量

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

返回：string

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_svol('000001.SZ'))
```

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

用法：ContextInfo.get_bvol(stockcode)

释义：根据代码获取对应股票的外盘成交量

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

返回：string

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_bvol('000001.SZ'))
```

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

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

释义：获取龙虎榜数据

参数：

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

startTime：起始时间，如 '20170101'

endTime：结束时间，如 '20180101'

返回：Dataframe，字段：

> reason：上榜原因
>
> close：收盘价
>
> spreadRate：涨跌幅
>
> TurnoverVolune：成交量
>
> Turnover_Amount：成交金额
>
> buyTraderBooth：买方席位,datframe
>
> sellTraderBooth：卖方席位,datframe

- buyTraderBooth 或 sellTraderBooth 包含字段：

  > traderName：交易营业部名称
  >
  > buyAmount：买入金额
  >
  > buyPercent：买入金额占总成交占比

> sellAmount：卖出金额
>
> sellPercent：卖出金额占总成交占比
>
> totalAmount：该席位总成交金额
>
> rank：席位排行
>
> direction：买卖方向

示例：

```
def handlebar(ContextInfo):
    print(ContextInfo.get_longhubang(['000002.SZ'],'20100101','20180101'))
```

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

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

释义：获取十大股东数据

参数：

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

返回：

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

- 字段

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

示例：

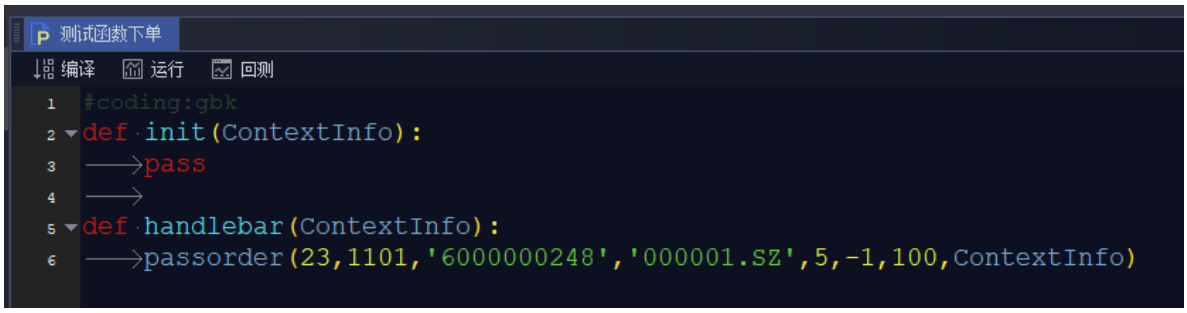
```
def handlebar(ContextInfo):
    print(ContextInfo.get_top10_share_holder(['000002.SZ'], 'flow_holder',
'20100101', '20180101'))
```

3.2.4. 交易函数

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

运作流程：

（1）编写测试策略



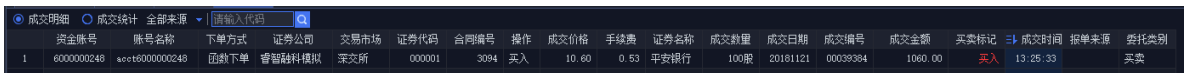
（2）运行策略

（3）触发下单，解析参数

（4）生成任务

（5）形成委托，过风控

（6）成交



3.2.4.1. 交易函数基本信息

国信 iQuant 极速策略交易系统提供了一系列的 Python 下单函数接口，如下所示。

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

（1）交易函数

函数名	释义
passorder	综合交易下单
get_trade_detail_data	取交易细明数据函数
get_value_by_order_id	根据委托Id取委托或成交信息
get_last_order_id	获取最新的委托或成交的委托Id
can_cancel_order	查询委托是否可撤销
cancel	取消委托
do_order	实时触发前一根bar信号函数

（2）股票下单函数

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

（3）期货下单函数

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

3.2.4.2. 交易函数介绍

（1）综合交易下单 passorder()

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

释义：综合交易下单

参数：

- opType，操作类型，可选值：

  期货六键：

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

  期货四键：

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

  期货两键：

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

  股票买卖：

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

  融资融券：

  > 27：融资买入
  > 28：融券卖出
  > 29：买券还券
  > 30：直接还券
  > 31：卖券还款
  > 32：直接还款

> 33：信用账号股票买入
>
> 33：信用账号股票卖出

组合交易：

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

期权交易：

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

- orderType，下单方式

  *注：

  > 一、期货不支持 1102 和 1202
  >
  > 二、对所有账号组的操作相当于对账号组里的每个账号做一样的操作，如 passorder(23, 1202, 'testS', '000001.SZ', 5, -1, 50000, ContextInfo)，意思就是对账号组 testS 里的所有账号都以最新价开仓买入 50000 元市值的 000001.SZ 平安银行

  可选值：

  > 1101：单股、单账号、普通、股/手方式下单
  >
  > 1102：单股、单账号、普通、金额（元）方式下单（只支持股票）
  >
  > 1113：单股、单账号、总资产、比例 [0 ~ 1] 方式下单
  >
  > 1123：单股、单账号、可用、比例[0 ~ 1]方式下单

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

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

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

  > accountID = 'stockAccountID, futureAccountID'
  >
  > orderCode = 'basketName, futureName'
  >
  > hedgeRatio：套利比例（0 ~ 2 之间值，相当于 %0 至 200% 套利）
  >
  > volume：份数 \ 资金 \ 比例
  >
  > orderType（特殊设置）

  - orderType 可选值：

    > 2331：组合、套利、合约价值自动套利、按组合股票数量方式下单
    >
    > 2332：组合、套利、按合约价值自动套利、按组合股票权重方式下单
    >
    > 2333：组合、套利、按合约价值自动套利、按账号可用方式下单

- accountID，资金账号

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

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

- orderCode，下单代码

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

  *注：

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

- prType，下单选价类型

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

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

> -1：无效（实际下单时,需要用交易面板交易函数那设定的选价类型）
> 0：卖5价
> 1：卖4价
> 2：卖3价
> 3：卖2价
> 4：卖1价
> 5：最新价
> 6：买1价
> 7：买2价（组合不支持）
> 8：买3价（组合不支持）
> 9：买4价（组合不支持）
> 10：买5价（组合不支持）
> 11：（指定价）模型价（只对单股情况支持,对组合交易不支持）
> 12：市价
> 13：挂单价
> 14：对手价

- price，下单价格

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

  *注：

  > 一、单股下单时，prType 是模型价时 price 有效；其它情况无效；即单股时， prType 参数为 11 时被使用。 prType 参数不为 11 时也需填写，填写的内容可为 -1，0，2，100 等任意数字；
  >
  > 二、组合下单时，是组合套利时，price 作套利比例有效，其它情况无效。

- volume，下单数量（股 / 手 / 元 / % ）

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

  根据 orderType 值最后一位确定 volume 的单位：

  单股下单时：

  > 1：股 / 手
  > 2：金额（元）
  > 3：比例（% ）

  组合下单时：

  > 1：按组合股票数量（份）
  > 2：按组合股票权重（元）
  > 3：按账号可用（% ）

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

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

- quickTrade，int，设定是否立即触发下单，可选值：

  > 0：否
  >
  > 1：是

  *注：passorder是对最后一根 K 线完全走完后生成的模型信号在下一根 K 线的第一个tick数据来时触发下单交易；采用quickTrade参数设置为1时，只要策略模型中调用到passorder交易函数就触发下单交易。

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 单股单账号期货最新价买入 10 手
    passorder(0, 1101, 'test', target, 5, -1, 10, ContextInfo)

    # 单股单账号期货指定价买入 10 手
    passorder(0, 1101, 'test', target, 11, 3000, ContextInfo)

    # 单股单账号股票最新价买入 100 股（1 手）
    passorder(23, 1101, 'test', target, 5, 0, 100, ContextInfo)

    # 单股单账号股票指定价买入 100 股（1 手）
    passorder(23, 1101, 'test', target, 11, 7, 100, ContextInfo)
```

（2）获取交易细明数据 get_trade_detail_data()

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

释义：获取交易细明数据函数

参数：

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

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

- strDatatype：string，数据类型：

> 'POSITION'：持仓
>
> 'ORDER'：委托
>
> 'DEAL' ：成交
>
> 'ACCOUNT'：账号

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

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

*注：有四种交易相关信息，包括：

> POSITION：持仓
>
> ORDER：委托
>
> DEAL：成交
>
> ACCOUNT：账号

示例：

```
def handlebar(ContextInfo):
    obj_list = get_trade_detail_data('6000000248','stock','position')
    for obj in obj_list:
        print(obj.m_strInstrumentID)
        # 查看有哪些属性字段
        print(dir(obj))

    # 可用资金的查询
    acct_info = get_trade_detail_data('6000000248', 'stock', 'account')
    for i in acct_info:
        print(i.m_dAvailable)

    # 持仓查询
    position_info = get_trade_detail_data('6000000248', 'stock', 'position')
    for i in position_info:
        print(i.m_strInstrumentID, i.m_nVolume)
```

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

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

释义：根据委托号获取委托或成交信息。

参数：

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

> 'FUTURE'：期货
>
> 'STOCK'：股票
>
> 'CREDIT'：信用
>
> 'HUGANGTONG'：沪港通

> 'SHENGANGTONG'：深港通
>
> 'STOCK_OPTION'：期权

- strDatatype：string，数据类型：

> 'ORDER'：委托
>
> 'DEAL' ：成交

返回：pythonObj

示例：

```
def init(ContextInfo):
    ContextInfo.accid = '6000000248'

def handlebar(ContextInfo):
    orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')
    print(orderid)
    obj = get_value_by_order_id(orderid,ContextInfo.accid, 'stock', 'order')
    print(obj.m_strInstrumentID)
```

（4）获取最新的委托或成交的委托号 get_last_order_id()

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

释义：获取最新的委托或成交的委托号。

参数：

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

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

- strDatatype：string，数据类型：

> 'ORDER'：委托
>
> 'DEAL' ：成交

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

返回：String，委托号，如果没找到返回 '-1'。

示例：

```
def init(ContextInfo):
    ContextInfo.accid = '6000000248'

def handlebar(ContextInfo):
    orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')
    print(orderid)
    obj = get_value_by_order_id(orderid,ContextInfo.accid, 'stock', 'order')
    print(obj.m_strInstrumentID)
```

（5）查询委托是否可撤销 can_cancel_order()

用法：can_cancel_order(orderId, accountID, strAccountType)

释义：查询委托是否可撤销。

参数：

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

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

返回：bool，是否可撤，返回值含义：

> True：可撤销
>
> False：不可撤销

示例：

```
def init(ContextInfo):
    ContextInfo.accid = '6000000248'

def handlebar(ContextInfo):
    orderid = get_last_order_id(ContextInfo.accid,'stock','order')
    print(orderid)
    can_cancel = can_cancel_order(orderid,ContextInfo.accid,'stock')
    print('是否可撤:', can_cancel)
```

（6）取消委托 cancel()

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

释义：取消委托。

参数：

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

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

- ContextInfo：pythonobj

返回：bool，是否发出了取消委托信号，返回值含义：

> True：是
>
> False：否

示例：

```
'''
（1）下单前,根据 get_trade_detail_data 函数返回账号的信息，判定资金是否充足，账号是否在登录状态，统计持仓情况等等。
（2）满足一定的模型条件，用 passorder 下单。
（3）下单后，时刻根据 get_last_order_id 函数获取委托和成交的最新id，注意如果委托生成了，就有了委托号（这个id需要自己保存做一个全局控制）。
（4）用该委托号根据 get_value_by_order_id 函数查看委托的状态，各种情况等。
当一个委托的状态变成“已成'后，那么对应的成交 deal 信息就有一条成交数据；用该委托号可查看成交情况。
*注：委托列表和成交列表中的委托号是一样的,都是这个 m_strOrderSysID 属性值。
可用 get_last_order_id 获取最新的 order 的委托号,然后根据这个委托号获取 deal 的信息，当获取成功后，也说明这笔交易是成了，可再根据 position 持仓信息再进一步验证。
（5）根据委托号获取委托信息，根据委托状态，或模型设定，用 cancel 取消委托。
'''

def init(ContextInfo):
    ContextInfo.accid = '6000000248'

def handlebar(ContextInfo):
    orderid = get_last_order_id(ContextInfo.accid, 'stock', 'order')
    print(cancel(orderid, ContextInfo.accid, 'stock', ContextInfo))
```

（7）实时触发前一根 bar 信号函数 do_order()

用法：do_order(ContextInfo)

释义：实时触发前一根bar信号函数。

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

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

参数：无

返回：无

示例：

```
# 实现跑日 K 线及以上周期下,在固定时间点把前一周期的交易信号发送出去
def init(ContextInfo):
    pass

def handlebar(ContextInfo):
    order_lots('000002.SZ', 1, ContextInfo, '600000248')
    ticktimetag = ContextInfo.get_tick_timetag()
    int_time = int(timetag_to_datetime(ticktimetag, '%H%M%S'))
    if 100500 <= int_time < 100505:
        do_order(ContextInfo)
```

（8）指定手数交易 order_lots()

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

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

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下 1 手买入
    order_lots('000002.SZ', 1, ContextInfo, '600000248')

    # 用对手价下 1 手卖出
    order_lots('000002.SZ', -1, 'COMPETE', ContextInfo, '600000248')

    # 用指定价 37.5 下 2 手卖出
    order_lots('000002.SZ', -2, 'fix', 37.5, ContextInfo, '600000248')
```

（9）指定价值交易 order_value()

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

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

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下 10000 元买入
    order_value('000002.SZ', 10000, ContextInfo, '600000248')

    # 用对手价下 10000 元卖出
    order_value('000002.SZ', -10000, 'COMPETE', ContextInfo, '600000248')

    # 用指定价 37.5 下 20000 元卖出
    order_value('000002.SZ', -20000, 'fix', 37.5, ContextInfo, '600000248')
```

（10）指定比例交易 order_percent()

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

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

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下 5.1% 价值买入
    order_percent('000002.SZ', 0.051, ContextInfo, '600000248')

    # 用对手价下 5.1% 价值卖出
    order_percent('000002.SZ', -0.051, 'COMPETE', ContextInfo, '600000248')

    # 用指定价 37.5 下 10.2% 价值卖出
    order_percent('000002.SZ', -0.102, 'fix', 37.5, ContextInfo, '600000248')
```

（11）指定目标价值交易 order_target_value()

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

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

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下调仓到 10000 元持仓
    order_target_value('000002.SZ', 10000, ContextInfo, '600000248')

    # 用对手价调仓到 10000 元持仓
    order_target_value('000002.SZ', 10000, 'COMPETE', ContextInfo, '600000248')

    # 用指定价 37.5 下调仓到 20000 元持仓
    order_target_value('000002.SZ', 20000, 'fix', 37.5, ContextInfo,
'600000248')
```

（12）指定目标比例交易 order_target_percent()

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

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

参数：

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

> 'LATEST'：最新
>
> 'FIX'：指定
>
> 'HANG'：挂单
>
> 'COMPETE'：对手

> 'MARKET'：市价
>
> 'SALE5', 'SALE4', 'SALE3', 'SALE2', 'SALE1'：卖5-1价
>
> 'BUY1', 'BUY2', 'BUY3', 'BUY4', 'BUY5'：买1-5价

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下买入调仓到 5.1% 持仓
    order_target_percent('000002.SZ', 0.051, ContextInfo, '600000248')

    # 用对手价调仓到 5.1% 持仓
    order_target_percent('000002.SZ', 0.051, 'COMPETE', ContextInfo,
'600000248')

    # 用指定价 37.5 调仓到 10.2% 持仓
    order_target_percent('000002.SZ', 0.102, 'fix', 37.5, ContextInfo,
'600000248')
```

（13）指定股数交易 order_shares()

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

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

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下 100 股买入
    order_shares('000002.SZ', 100, ContextInfo, '600000248')

    # 用对手价下 100 股卖出
    order_shares('000002.SZ', -100, 'COMPETE', ContextInfo, '600000248')

    # 用指定价 37.5 下 200 股卖出
    order_shares('000002.SZ', -200, 'fix', 37.5, ContextInfo, '600000248')
```

（14）期货买入开仓 buy_open()

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

释义：期货买入开仓

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价 1 手买入开仓
    buy_open('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手买入开仓
    buy_open('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手买入开仓
    buy_open('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

（15）期货买入开仓（平今优先） buy_close_tdayfirst()

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

释义：期货买入开仓，平今优先

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价 1 手买入平仓，平今优先
    buy_close_tdayfirst('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手买入平仓，平今优先
    buy_close_tdayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手买入平仓，平今优先
    buy_close_tdayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

（16）期货买入开仓（平昨优先） buy_close_ydayfirst()

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

释义：期货买入开仓，平昨优先

参数：

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

> 'LATEST'：最新
>
> 'FIX'：指定

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价 1 手买入平仓，平昨优先
    buy_close_ydayfirst('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手买入平仓，平昨优先
    buy_close_ydayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手买入平仓，平昨优先
    buy_close_ydayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

（17）期货卖出开仓 sell_open()

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

释义：期货卖出开仓

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价 1 手卖出开仓
    sell_open('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手卖出开仓
    sell_open('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手卖出开仓
    sell_open('IF1805.IF', 2, 'fix',3750, ContextInfo, '110476')
```

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

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

释义：期货卖出平仓，平今优先

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价下 1 手卖出平仓，平今优先
    sell_close_tdayfirst('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手卖出平仓，平今优先
    sell_close_tdayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手卖出平仓，平今优先
    sell_close_tdayfirst('IF1805.IF', 1, 'fix', 3750, ContextInfo, '110476')
```

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

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

释义：期货卖出平仓，平今优先

参数：

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

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

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

返回：无

示例：

```
def handlebar(ContextInfo):
    # 按最新价 1 手卖出平仓，平昨优先
    sell_close_ydayfirst('IF1805.IF', 1, ContextInfo, '110476')

    # 用对手价 1 手卖出平仓，平昨优先
    sell_close_ydayfirst('IF1805.IF', 1, 'COMPETE', ContextInfo, '110476')

    # 用指定价 3750 元 2 手卖出平仓，平昨优先
    sell_close_ydayfirst('IF1805.IF', 2, 'fix', 3750, ContextInfo, '110476')
```

（20）获取两融负债合约明细 get_debt_contract()

用法：get_debt_contract(accId)

释义：获取信用账户负债合约明细

参数：

- accId：信用账户

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

示例：

```
def handlebar(ContextInfo):
    obj_list = get_debt_contract('6000000248')
    for obj in obj_list:
        # 输出负债合约名
        print(obj.m_strInstrumentName)
```

（21）获取两融担保标的明细 get_assure_contract()

用法：get_debt_contract(accId)

释义：获取信用账户担保合约明细

参数：

- accId：信用账户

示例：

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

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

用法：get_enable_short_contract(accId)

释义：获取信用账户当前可融券的明细

参数：

- accId：信用账户

示例：

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

3.2.5. 成交回报实时主推函数

*注：成交回报实时主推函数仅在实盘运行模式下生效。

（1）资金账号状态变化主推 account_callback()

用法：account_callback(ContextInfo, accountInfo)

释义：当资金账号状态有变化时，运行这个函数

参数：

- ContextInfo：特定对象
- accountInfo：资金账号对象，可对应查看5.4. 附录4 交易函数内含属性说明

返回：无

示例：

```
def init(ContextInfo):
    # 设置对应的资金账号
    ContextInfo.set_account('6000000058')

def handlebar(ContextInfo):
    pass

def account_callback(ContextInfo, accountInfo):
    print('accountInfo')
    print(accountInfo.m_strStatus) # m_strStatus 为资金账号的属性之一，表示资金账号的状态
```

（2）账号委托状态变化主推 order_callback()

用法：order_callback(ContextInfo, accountInfo)

释义：当账号委托状态有变化时，运行这个函数

参数：

- ContextInfo：特定对象
- orderInfo：委托账号对象，可对应查看5.4. 附录4 交易函数内含属性说明

返回：无

示例：

```
def init(ContextInfo):
    # 设置对应的资金账号
    ContextInfo.set_account('6000000058')

def handlebar(ContextInfo):
    pass

def order_callback(ContextInfo, orderInfo):
    print('orderInfo')
```

（3）账号成交状态变化主推 deal_callback()

用法：deal_callback(ContextInfo, dealInfo)

释义：当账号成交状态有变化时，运行这个函数

参数：

- ContextInfo：特定对象
- dealInfo：资金账号对象，可对应查看5.4. 附录4 交易函数内含属性说明

返回：无

示例：

```
def init(ContextInfo):
    # 设置对应的资金账号
    ContextInfo.set_account('6000000058')

def handlebar(ContextInfo):
    pass

def deal_callback(ContextInfo, dealInfo):
    print('dealInfo')
```

（4）账号持仓状态变化主推 position_callback()

用法：position_callback(ContextInfo, positonInfo)

释义：当账号持仓状态有变化时，运行这个函数

参数：

- ContextInfo：特定对象
- positonInfo：资金账号对象，可对应查看5.4. 附录4 交易函数内含属性说明

返回：无

示例：

```
def init(ContextInfo):
    # 设置对应的资金账号
    ContextInfo.set_account('6000000058')

def handlebar(ContextInfo):
    pass

def position_callback(ContextInfo, positonInfo):
    print('positonInfo')
```

3.2.6. 引用函数

*注：以下函数均支持回测和实盘/模拟运行模式。

（1）获取扩展数据 ext_data()

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

释义：获取扩展数据

参数：

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

  > 0：不偏移

> N：向右偏移N
>
> -N：向左偏移N

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

返回：number

示例：

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

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

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

释义：获取引用的扩展数据的数值在所有品种中的排名

参数：

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

> 0：不偏移
>
> N：向右偏移N
>
> -N：向左偏移N

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

返回：number

示例：

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

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

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

释义：获取引用的扩展数据的数值在指定时间区间内所有品种中的排名

参数：

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

返回：pythonDict

示例：

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

（4）获取扩展数据在指定时间区间内的值 ext_data_range()

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

释义：获取扩展数据在指定时间区间内的值

参数：

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

返回：pythonDict

示例：

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

（5）获取因子数据 get_factor_value()

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

释义：获取因子数据

参数：

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

返回：number

示例：

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

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

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

释义：获取引用的因子数据的数值在所有品种中排名

参数：

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

返回：number

示例：

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

3.2.7. 绘图函数

*注：以下函数均支持回测和实盘/模拟运行模式。

（1）在界面上画图 ContextInfo.paint()

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

释义：在界面上画图

参数：

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

  > 0：曲线
  > 42：柱状线

- color：string，颜色（不填默认为白色）目前支持以下几种颜色：

  > blue：蓝
  > brown：棕
  > cyan：蓝绿
  > green：绿
  > magenta：品红
  > red：红
  > white：白
  > yellow：黄

- limit：string，画线控制，可取值：

  > 'noaxis'： 不影响坐标画线
  > 'nodraw' ：不画线

返回：无

示例：

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

（2）在图形上显示文字 ContextInfo.draw_text()

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

释义：在图形上显示数字

参数：

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

返回：无

示例：

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

（3）在图形上显示数字 ContextInfo.draw_number()

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

释义：在图形上显示数字

参数：

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

返回：无

示例：

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

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

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

释义：在数字1和数字2之间绘垂直线

参数：

- cond：bool，条件

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

  > blue：蓝
  >
  > brown：棕
  >
  > cyan：蓝绿
  >
  > green：绿
  >
  > magenta：品红
  >
  > red：红
  >
  > white：白
  >
  > yellow：黄

- limit：string，画线控制，可取值：

  > 'noaxis'： 不影响坐标画线
  >
  > 'nodraw' ：不画线

返回：无

示例：

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

（5）在图形上绘制小图标 ContextInfo.draw_icon()

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

释义：在图形上绘制小图标

参数：

- cond：bool，条件
- height：number，图标的位置
- text：number，图标的类型，可取值：

  > 1：椭圆
  >
  > 0：矩形

返回：无

示例：

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

4. 财务数据接口使用方法

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

4.1. Python接口

用法1

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

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

返回：

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

（1）代码列表 1 时间范围为 1，返回： pandas.Series index = 字段

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

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

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

示例：

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

*注：

选择按照公告期取数和按照报告期取数的区别：

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

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

用法2

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

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

返回：

number

示例：

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

4.2. 财务数据字段对照表

4.2.1. 资产负债表 (ASHAREBALANCESHEET)

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

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

4.2.2. 利润表 (ASHAREINCOME)

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

4.2.3. 现金流量表 (ASHARECASHFLOW)

中文字段	迅投字段
收到其他与经营活动有关的现金	other_cash_recp_ral_oper_act
经营活动现金流入小计	stot_cash_inflows_oper_act
支付给职工以及为职工支付的现金	cash_pay_beh_empl
支付的各项税费	pay_all_typ_tax
支付其他与经营活动有关的现金	other_cash_pay_ral_oper_act
经营活动现金流出小计	stot_cash_outflows_oper_act
经营活动产生的现金流量净额	net_cash_flows_oper_act
取得投资收益所收到的现金	cash_recp_return_invest
处置固定资产、无形资产和其他长期投资收到的现金	net_cash_recp_disp_fiolta
投资活动现金流入小计	stot_cash_inflows_inv_act
投资支付的现金	cash_paid_invest
购建固定资产、无形资产和其他长期投资支付的现金	cash_pay_acq_const_fiolta
支付其他与投资的现金	other_cash_pay_ral_inv_act
投资活动产生的现金流出小计	stot_cash_outflows_inv_act
投资活动产生的现金流量净额	net_cash_flows_inv_act
吸收投资收到的现金	cash_recp_cap_contrib
取得借款收到的现金	cash_recp_borrow
收到其他与筹资活动有关的现金	other_cash_recp_ral_fnc_act
筹资活动现金流入小计	stot_cash_inflows_fnc_act
偿还债务支付现金	cash_prepay_amt_borr
分配股利、利润或偿付利息支付的现金	cash_pay_dist_dpcp_int_exp
支付其他与筹资的现金	other_cash_pay_ral_fnc_act
筹资活动现金流出小计	stot_cash_outflows_fnc_act
筹资活动产生的现金流量净额	net_cash_flows_fnc_act

中文字段	迅投字段
汇率变动对现金的影响	eff_fx_flu_cash
现金及现金等价物净增加额	net_incr_cash_cash_equ
销售商品、提供劳务收到的现金	goods_sale_and_service_render_cash
收到的税费与返还	tax_levy_refund
购买商品、接受劳务支付的现金	goods_and_services_cash_paid
处置子公司及其他收到的现金	net_cash_deal_subcompany
其中子公司吸收现金	cash_from_mino_s_invest_sub
处置固定资产、无形资产和其他长期资产支付的现金净额	fix_intan_other_asset_dispo_cash_payment
报告截止日	m_timetag
公告日	m_anntime

4.2.4. 股本表 (CAPITALSTRUCTURE)

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

4.2.5. 主要指标 (PERSHAREINDEX)

中文字段	迅投字段
每股经营活动现金流量	s_fa_ocfps
每股净资产	s_fa_bps
基本每股收益	s_fa_eps_basic
稀释每股收益	s_fa_eps_diluted
每股未分配利润	s_fa_undistributedps
每股资本公积金	s_fa_surpluscapitalps
扣非每股收益	adjusted_earnings_per_share
主营收入	inc_revenue
毛利润	inc_gross_profit
利润总额	inc_profit_before_tax
净利润	du_profit
归属于母公司所有者的净利润	inc_net_profit
扣非净利润	adjusted_net_profit
净资产收益率	du_return_on_equity
销售毛利率	sales_gross_profit
主营收入同比增长	inc_revenue_rate
净利润同比增长	du_profit_rate
归属于母公司所有者的净利润同比增长	inc_net_profit_rate
扣非净利润同比增长	adjusted_net_profit_rate
营业总收入滚动环比增长	inc_total_revenue_annual
归属净利润滚动环比增长	inc_net_profit_to_shareholders_annual
扣非净利润滚动环比增长	adjusted_profit_to_profit_annual
加权净资产收益率	equity_roe
摊薄净资产收益率	net_roe
摊薄总资产收益率	total_roe
毛利率	gross_profit
净利率	net_profit
实际税率	actual_tax_rate
预收款营业收入	pre_pay_operate_income
销售现金流营业收入	sales_cash_flow

中文字段	迅投字段
资产负债比率	gear_ratio
存货周转率	inventory_turnover

5. 附录

5.1. 附录1 市场简称代码

> 上证所　SH
>
> 深交所　SZ
>
> 大商所　DF
>
> 郑商所　ZF
>
> 上期所　SF
>
> 中金所　IF
>
> 外部自定义市场 ED

5.2. 附录2 CSRC行业列表

证监会行业

> CSRC保险业
>
> CSRC采矿业
>
> CSRC餐饮业
>
> CSRC仓储业
>
> CSRC电力、热力、燃气及水生产和供应业
>
> CSRC电力、热力生产和供应业
>
> CSRC电信、广播电视和卫星传输服务
>
> CSRC房地产业
>
> CSRC房屋建筑业
>
> CSRC纺织服装、服饰业
>
> CSRC纺织业
>
> CSRC非金属矿采选业
>
> CSRC非金属矿物制品业
>
> CSRC废弃资源综合利用业
>
> CSRC公共设施管理业
>
> CSRC广播、电视、电影和影视录音制作业

CSRC航空运输业

CSRC黑色金属矿采选业

CSRC互联网和相关服务

CSRC化学纤维制造业

CSRC货币金融服务

CSRC计算机、通信和其他电子设备制造业

CSRC家具制造业

CSRC建筑安装业

CSRC建筑业

CSRC建筑装饰和其他建筑业

CSRC交通运输、仓储和邮政业

CSRC教育

CSRC金融业

CSRC金属制品业

CSRC酒、饮料和精制茶制造业

CSRC开采辅助活动

CSRC科学研究和技术服务业

CSRC林业

CSRC零售业

CSRC煤炭开采和洗选业

CSRC农、林、牧、渔服务业

CSRC农、林、牧、渔业

CSRC农副食品加工业

CSRC农业

CSRC批发和零售业

CSRC批发业

CSRC皮革、毛皮、羽毛及其制品和制鞋业

CSRC其他金融业

CSRC其他制造业

CSRC汽车制造业

CSRC燃气生产和供应业

CSRC软件和信息技术服务业

CSRC商务服务业

CSRC生态保护和环境治理业

CSRC石油和天然气开采业

CSRC食品制造业

CSRC水的生产和供应业

CSRC水利、环境和公共设施管理业

CSRC水上运输业

CSRC体育

CSRC铁路、船舶、航空航天和其他运输设备制造业

CSRC铁路运输业

CSRC通用设备制造业

CSRC土木工程建筑业

CSRC卫生

CSRC卫生和社会工作

CSRC文化、体育和娱乐业

CSRC文化艺术业

CSRC文教、工美、体育和娱乐用品制造业

CSRC橡胶和塑料制品业

CSRC新闻和出版业

CSRC信息传输、软件和信息技术服务业

CSRC畜牧业

CSRC研究和试验发展

CSRC医药制造业

CSRC仪器仪表制造业

CSRC印刷和记录媒介复制业

CSRC邮政业

CSRC有色金属矿采选业

CSRC渔业

CSRC制造业

CSRC住宿和餐饮业

CSRC住宿业

CSRC专业技术服务业

CSRC专用设备制造业

CSRC资本市场服务

CSRC综合

CSRC租赁和商务服务业

5.2. 附录3 is_typed_stock 函数证券分类表

注：代表任意阿拉伯数字

（1）基础类型

类型描述	市场代码	类别标号
沪市	******,SH	1
深市	******,SZ	2
中金	******|*****|****|***,IF	3
上期	******|*****|****|***,SF	4
大商	******|*****|****|***,DF	5
郑商	******|*****|****|***,ZF	6
开放基金	******,OF	7
股票期权	******,SHO	8
新三板	******,NEEQ	9
沪市A股	60****,SH	101
沪市B股	90****,SH	102
沪市封基	50****,SH	103
沪市指数	000***,SH	104
沪市ETF	510***|511***|512***|513***|518***,SH	105
沪市权证	58****,SH	106
沪市申购	73****|78****,SH	107
沪市可交换公司债券	132***,SH	108
沪市可交换公司债券质押券出入库	133***,SH	109
沪市可交换公司债券换股	192***,SH	110
沪市并购重组私募债券挂牌转让	1355**|1356**|1357**|1358**|1359**,SH	111
沪市证券公司短期债券挂牌转让	1350**|1351**|1352**|1353**|1354**,SH	112
沪市信贷资产支持证券交易 asset-backed securities	128***,SH	113
沪市公司债券质押券入库	102***|134***,SH	114
沪市公司债	122***|123***|124***|127***|136***,SH	115
沪市公开发行优先股交易	330***,SH	116
沪市非公开发行优先股转让	360***,SH	117
沪市公开发行优先股申购	770***,SH	118
沪市公开发行优先股配股/配售	771***,SH	119
沪市公开发行优先股申购款分配	772***,SH	120
沪市公开发行优先股申购配号	773***,SH	121
沪市国债回购（席位托管方式）	201***,SH	122
沪市企业债回购	202***,SH	123
沪市国债买断式回购	203***,SH	124
沪市新质押式国债回购	204***,SH	125
沪市附息国债	010***|019***,SH	127
沪市金融债	018***,SH	128
沪市贴现国债	020***,SH	129
沪市中央政府债（国债）	010***|019***|020***,SH	130
沪市分离债	126***,SH	131
沪市资产证券化	121***,SH	132
沪市信贷资产支持	128***,SH	133
沪市企业债（席位托管方式）	120***|129***,SH	134
沪市可转债	100***|110***|112***|113***|128***,SH	135
沪市地方债	130***|140***,SH	136
沪市政府债（国债+地方债）	010***|019***|020***|130***,SH	137
上海可交换私募债	1380**,SH	138
沪市标准券	888880|SHRQ88,SH	139
沪市封闭式基金	500***|5058**,SH	140
沪市政策性金融债	018***|028***|038***,SH	141
沪市上海质押代码	09****|102***|103***|104***|105***|106***|107***|108***|133***,SH	142
2000年前发行国债	009***,SH	143
记账式贴现国债质押式回购标准券入库	107***,SH	144
公司债质押式回购标准券入库	1040**|1041**|1042**|1043**|1044**,SH	145

类型描述	市场代码	类别标号
国债分销	7510**|7511**,SH	146
地方政府债质押式回购标准券入库	106***|141***,SH	147
地方政府债分销	75190*|75191*|75192*|75193*|75194*|75195*|75196*,SH	148
分离债质押式回购标准券入库	1050**|1051**|1052**|1053**|1054**|1055**|1056**|1057*|1058**,SH	149
债券质押式报价回购	205***,SH	150
中小企业私募债券在固定收益平台转让	125***,SH	151
跨境ETF	513030|513500|513100|510900|513600|513660,SH	152
跨境LOF	,SH	153
上海创新型封闭式基金	5058**,SH	154
上海的固定收益类	511***,SH	155
上海黄金	518**0,SH	156
上海实时申赎货币基金	5198**|5195**|5199**,SH	157
上海交易型货币基金	5116**|5117**|5118**|5119**,SH	158
上海股票申购代码	730***|732***|780***,SH	159
上海债券申购代码	733***|783***,SH	160
上海基金申购代码	735***,SH	161
上海新股配号	741***|791***|736***,SH	162
上海配售首发配号	747***|797***,SH	163
上海可转债资金申购配号	744***|794***|756***,SH	164
上海申购款	740***|790***|734***,SH	165
上海发债款	743***|793***|755***,SH	166
上海配股代码	700***|760***|742***,SH	167
上海配转债代码	704***|764***|753***,SH	168
上海LOF	501***,SH	169
上海分级基金	502***,SH	170
沪新股额	SHXGED,SH	171
沪增发股	730***|731***|780***|781***,SH	172
上海跨境ETF申赎代码	513031|513501|513101|510901|513601|513661,SH	173
上海债券ETF	511010|511210|511220,SH	174
上海企业债券挂牌转让	139***,SH	175
上海可交换私募债	1380**,SH	176
沪市1天回购	204001,SH	177
沪市2天回购	204002,SH	178
沪市3天回购	204003,SH	179
沪市4天回购	204004,SH	180
沪市7天回购	204007,SH	181
沪市14天回购	204014,SH	182
沪市28天回购	204028,SH	183
沪市28天以上回购	204091|204182,SH	184
上海分级基金子基金	502**1|502**2|502**4|502**5|502**7|502008|502018|502028|502038|502058|502049|502050,SH	185
深市A股	00****|30****,SZ	10001
深市B股	20****,SZ	10002
深市封基	15****|16****|18****,SZ	10003
深市主板	000***|001***,SZ	10004
深市小板	002***|003***|004***,SZ	10005
深市创业板	30****,SZ	10006
深市指数	39****,SZ	10007
深市ETF	159***,SZ	10008
深市权证	03****,SZ	10009
深市国债回购(131990不是的,需要业务支持)	131990,SZ	10010
深市附息国债	100***|101***|102***|103***|104***|105***|106***|107***,SZ	10011
深市贴现国债	108***,SZ	10012
深市公司债	112***,SZ	10013

类型描述	市场代码	类别标号
深市企业债	111***,SZ	10014
深市分离债	115***,SZ	10015
深市私募债	118***|114***,SZ	10016
深市专项资金管理规划	119***,SZ	10017
深市地方政府债	109***,SZ	10018
深市可转债	12****,SZ	10019
深市标准券	131990|SZRQ88,SZ	10020
深市封闭式基金	184***,SZ	10021
深市LOF	16****,SZ	10022
深市分级基金	150***|151***,SZ	10023
深市 中小企业可交换私募债	117***,SZ	10024
深市证券公司次级债	1189**,SZ	10025
深市其他中小企业私募债	1180**|1181**|1182**|1183**|1184**|1185**|1186**|1187**|1188**,SZ	10026
深市资产支持证券	1190**|1191**|1192**|1193**|1194**,SZ	10027
深市分级基金子基金	150***|151***,SZ	10028
深市跨境ETF	159920|159941,SZ	10029
深市跨境LOF	160125|160416|160717|160719|161116|161210|161714|161815|162411|164701|164815|165510|165513,SZ	10030
深市创新型封闭式基金	150***,SZ	10031
深市主板可转换公司债券	127***,SZ	10032
深市创业板可转换公司债券	123***,SZ	10033
深市中小板可转换公司债券	128***,SZ	10034
深市国债回购(131900不是的，需要业务支持)	131***,SZ	10035
深市黄金	159934,SZ	10036
深市实时申赎货币基金	1590**,SZ	10037
深新股额	SZXGED,SZ	10038
深增发股	07****|37***,SZ	10039
深圳配股	08****,SZ	10040
深圳债券ETF 仅此一支	159926,SZ	10041
深市1天回购	131810,SZ	10042
深市2天回购	131811,SZ	10043
深市3天回购	131800,SZ	10044
深市4天回购	131809,SZ	10045
深市7天回购	131801,SZ	10046
深市14天回购	131802,SZ	10047
深市28天回购	131803,SZ	10048
深市28天以上回购	131805|131806,SZ	10049
中金所指数期货	IF****|IH****|IC****|IF**|IH**|IC**,IF	30001
中金所国债期货	TF****|T****|HTFF****|TF**|T**|HTFF**,IF	30002

（2）扩展类型

类型描述	类型集合	类别标号
沪深A股	沪市A股|深市A股	100001
沪深B股	沪市B股|深市B股	100002
沪深狭义股票	沪深A股|沪深B股	100003
沪深封基	沪市封基|深市封基	100004
指数	沪市指数|深市指数	100005
商品期货	上期|大商|郑商	100006
期货市场	中金|商品期货	100007
股票	沪市|深市	100008
深市中央政府债（国债）	深市附息国债|深市贴现国债	100009
深市政府债	深市中央政府债（国债）|深市地方政府债	100010
深市所有债券	深市附息国债|深市贴现国债|深市地方政府债|深市可转债|深市企业债|深市分离债|深市私募债|深市专项资金管理规划	100011
广义的股票	!指数	100012
ETF	沪市ETF|深市ETF	100013
封闭式基金	沪市封闭式基金|深市封闭式基金	100014
权证	沪市权证|深市权证	100015
债券	上海债券|深市所有债券	100016
深市国债回购	深市国债回购(131900不是的，需要业务支持)&(!深市国债回购(131990不是的，需要业务支持))	100017
标准券	沪市标准券|深市标准券	100018
债券回购	沪市国债回购（席位托管方式）|深市国债回购	100020
质押式回购	沪市新质押式国债回购|深市国债回购	100021
黄金	上海黄金|深市黄金	100022
实时申赎货币基金	上海实时申赎货币基金|深市实时申赎货币基金	100023
货币基金	实时申赎货币基金|上海交易型货币基金	100024
上海申购代码	上海股票申购代码|上海债券申购代码|上海基金申购代码	100025

类型描述	类型集合	类别标号
跨境ETF	深市跨境ETF|跨境ETF	100026
跨境LOF	跨境LOF|深市跨境LOF	100027
可交易的	沪深A股|沪深封基|ETF|权证|沪市申购|深市创业板|债券回购|债券ETF|上海的固定收益类|黄金|货币基金|深市中央政府债（国债）|沪市中央政府债（国债）|沪市地方债|深市地方政府债|上海申购代码|沪市上海质押代码|跨境ETF|跨境LOF|上海配股代码|上海配转债代码|深市可转债|沪市可转债|沪增发股|深增发股	100028
主板	沪市A股|深市主板	100029
回转交易	债券|黄金|上海的固定收益类|权证|跨境ETF|跨境LOF|上海交易型货币基金	100030
上海配号	上海新股配号|上海配售首发配号|上海可转债资金申购配号	100031
沪深新股申购额度	沪新股额|深新股额	100032
沪深配股代码	上海配股代码|深圳配股	100033
固定收益：跟踪债券指数的交易型开放式指数基金、交易型货币市场基金	上海的固定收益类	100034
固定收益类	沪市附息国债|深市附息国债|沪市贴现国债|深市贴现国债|沪市政府债（国债+地方债）|深市政府债|深市企业债|沪市企业债（席位托管方式）|深市私募债|沪市可转债|深市可转债|沪市分离债|深市分离债|债券回购|标准券|货币基金|上海的固定收益类	100035
分级基金	上海分级基金|深市分级基金	100036
LOF	上海LOF|深市LOF	100037
债券ETF	上海债券ETF|深圳债券ETF	100038
上海债券	沪市政府债（国债+地方债）|沪市可转债|沪市公司债|沪市企业债（席位托管方式）|沪市资产证券化|沪市分离债|沪市金融债|沪市信贷资产支持|沪市可交换公司债券|沪市并购重组私募债券挂牌转让|沪市证券公司短期债券挂牌转让|上海可交换私募债|上海企业债券挂牌转让|上海可交换私募债	100039
1天逆回购	沪市1天回购|深市1天回购	100040
2天逆回购	沪市2天回购|深市2天回购	100041

类型描述	类型集合	类别标号
3天逆回购	沪市3天回购|深市3天回购	100042
4天逆回购	沪市4天回购|深市4天回购	100043
7天逆回购	沪市7天回购|深市7天回购	100044
14天逆回购	沪市14天回购|深市14天回购	100045
28天逆回购	沪市28天回购|深市28天回购	100046
28天以上逆回购	沪市28天以上回购|深市28天以上回购	100047

5.4. 附录4 交易函数内含属性说明

5.4.1. account 资金账号对象

> m_dMaxMarginRate： 保证金比率 股票的保证金率等于1 股票不需要
>
> m_dFrozenMargin：冻结保证金,外源性 股票的保证金就是冻结资金 股票不需要
>
> m_dFrozenCash：冻结金额,内外源冻结保证金和手续费四个的和
>
> m_dFrozenCommission：冻结手续费 ,外源性冻结资金源
>
> m_dRisk：风险度,风险度 冻结资金/可用资金 股票不需要
>
> m_dNav：单位净值
>
> m_dPreBalance：期初权益 股票不需要 也叫静态权益
>
> m_dBalance：总资产,动态权益 即市值
>
> m_dAvailable：可用金额
>
> m_dCommission：手续费,已经用掉的手续费
>
> m_dPositionProfit：持仓盈亏,
>
> m_dCloseProfit：平仓盈亏 股票不需要
>
> m_dCashIn：出入金净值
>
> m_dCurrMargin：当前使用的保证金 股票不需要
>
> m_dInitBalance： 初始权益
>
> m_strStatus：状态
>
> m_dInitCloseMoney：期初平仓盈亏,初始平仓盈亏
>
> m_dInstrumentValue：总市值,合约价值,合约价值
>
> m_dDeposit：入金
>
> m_dWithdraw：出金

m_dPreCredit：上次信用额度 股票不需要

m_dPreMortgage：上次质押 股票不需要

m_dMortgage：质押 股票不需要

m_dCredit：信用额度 股票不需要

m_dAssetBalance： 证券初始资金,股票不需要

m_strOpenDate：起始日期股票不需要

m_dFetchBalance：可取金额

m_strTradingDate：交易日

m_dStockValue：股票总市值，期货没有

m_dLoanValue：债券总市值，期货没有

m_dFundValue：基金总市值，包括ETF和封闭式基金，期货没有

m_dRepurchaseValue：回购总市值，所有回购，期货没有

m_dLongValue：多单总市值，现货没有

m_dShortValue：单总市值，现货没有

m_dNetValue：净持仓总市值，净持仓市值=多-空

m_dAssureAsset：净资产

m_dTotalDebit：总负债

m_dEntrustAsset：可信资产,用于校对

m_dInstrumentValueRMB：总市值（人民币）,沪港通

m_dSubscribeFee：申购费,申购费

m_dGoldValue：库存市值,黄金现货库存市值

m_dGoldFrozen：现货冻结,黄金现货冻结

m_dMargin：占用保证金, 维持保证金

m_strMoneyType：币种

m_dPurchasingPower：购买力, 盈透购买力

m_dRawMargin ：原始保证金

m_dBuyWaitMoney：买入待交收金额（元）,买入待交收

m_dSellWaitMoney：卖出待交收金额（元）,卖出待交收

m_dReceiveInterestTotal：本期间应计利息

m_dRoyalty：权利金收支,期货期权用

m_dFrozenRoyalty：冻结权利金 期货期权用

m_dRealUsedMargin：实时占用保证金 用于股票期权

m_dRealRiskDegree：实时风险度

5.4.2. order 委托对象

> m_strExchangeID：证券市场
>
> m_strExchangeName：交易市场
>
> m_strProductID：品种代码
>
> m_strProductName：品种名称
>
> m_strInstrumentID：证券代码
>
> m_strInstrumentName：证券名称,合约名称
>
> m_strOrderRef：内部委托号,下单引用等于股票的内部委托号
>
> m_nOrderPriceType：EBrokerPriceType 类型，例如市价单 限价单
>
> m_nDirection：EEntrustBS 类型,操作,多空,期货多空 股票买卖永远是48，其他的dir同理
>
> m_nOffsetFlag：EOffset_Flag_Type类型,操作,期货开平，股票买卖其实就是开平
>
> m_nHedgeFlag：EHedge_Flag_Type类型, 投保
>
> m_dLimitPrice：委托价格,限价单的限价，就是报价
>
> m_nVolumeTotalOriginal：委托量,最初委托量
>
> m_nOrderSubmitStatus：EEntrustSubmitStatus类型，报单状态 ,提交状态，股票中不需要报单状态
>
> m_strOrderSysID：合同编号,委托号
>
> m_nOrderStatus：EEntrustStatus 委托状态
>
> m_nVolumeTraded：成交数量 ,已成交量
>
> m_nVolumeTotal：委托剩余量 ,当前总委托量 股票的含义是总委托量减去成交量
>
> m_nErrorID：状态信息
>
> m_dFrozenMargin：冻结金额,冻结保证金
>
> m_dFrozenCommission：冻结手续费
>
> m_strInsertDate：委托日期,报单日期
>
> m_strInsertTime：委托时间
>
> m_dTradedPrice：成交均价(股票)
>
> m_dCancelAmount：已撤数量
>
> m_strOptName：买卖标记,展示委托属性的中文
>
> m_dTradeAmount：成交金额, 成交额 期货=均价*量*合约乘数
>
> m_eEntrustType：EEntrustTypes, 委托类别
>
> m_strCancelInfo：废单原因
>
> m_strUnderCode：标的证券代码
>
> m_eCoveredFlag：ECoveredFlag 备兑标记 '0' - 非备兑，'1' - 备兑
>
> m_strCompactNo：合约编号

5.4.3. deal 成交对象

> m_strExchangeID：证券市场,交易所代码
>
> m_strExchangeName：交易市场,交易所名称
>
> m_strProductID：品种代码
>
> m_strProductName：品种名称
>
> m_strInstrumentID：证券代码
>
> m_strInstrumentName：证券名称
>
> m_strTradeID：成交编号
>
> m_strOrderRef：下单引用 等于股票的内部委托号
>
> m_strOrderSysID：合同编号,报单编号,委托号
>
> m_nDirection：EEntrustBS 买卖方向
>
> m_nOffsetFlag：EOffset_Flag_Type 开平 股票的买卖
>
> m_nHedgeFlag：EHedge_Flag_Type //投保 股票不需要
>
> m_dPrice： 成交均价
>
> m_nVolume：成交量 期货单位手 股票做到股
>
> m_strTradeDate：成交日期
>
> m_strTradeTime：成交时间
>
> m_dComssion：手续费
>
> m_dTradeAmount： 成交额 期货=均价*量*合约乘数
>
> m_nOrderPriceType：EBrokerPriceType 类型，例如市价单 限价单
>
> m_strOptName：买卖标记 //展示委托属性的中文
>
> m_eEntrustType：EEntrustTypes类型, 委托类别
>
> m_eFutureTradeType：EFutureTradeType类型,成交类型
>
> m_nRealOffsetFlag：EOffset_Flag_Type类型,实际开平,主要是区分平今和平昨
>
> m_eCoveredFlag：ECoveredFlag类型, 备兑标记 '0' - 非备兑，'1' - 备兑
>
> m_nCloseTodayVolume：平今量, 不显示
>
> m_dOrderPriceRMB：委托价格（人民币）//目前用于港股通
>
> m_dPriceRMB：成交价格（人民币）//目前用于港股通
>
> m_dTradeAmountRMB：成交金额（人民币） //目前用于港股通
>
> m_dReferenceRate： //汇率,目前用于港股通
>
> m_strXTTrade： 是否是国信交易
>
> m_strCompactNo： 合约编号
>
> m_dCloseProfit：平仓盈亏 ,目前用于外盘

5.4.4. position 持仓对象

> m_strExchangeID：证券市场;
>
> m_strExchangeName：市场名称
>
> m_strProductID：品种代码
>
> m_strProductName：品种名称
>
> m_strInstrumentID：证券代码
>
> m_strInstrumentName：证券名称
>
> m_nHedgeFlag：EHedge_Flag_Type类型, 投保 ,股票不需要
>
> m_nDirection：EEntrustBS类型, 买卖 ; 股票不需要
>
> m_strOpenDate：成交日期
>
> m_strTradeID：成交号,最初开仓位的成交
>
> m_nVolume：当前拥股,持仓量,
>
> m_dOpenPrice：成本价,开仓价
>
> m_strTradingDay：当前交易日
>
> m_dMargin：使用的保证金 历史的直接用ctp的，新的自己用成本价*存量*系数算 股票不需要
>
> m_dOpenCost：开仓成本 等于股票的成本价*第一次建仓的量，后续减持不影响，不算手续费 股票不需要
>
> m_dSettlementPrice：最新价,结算价,对于股票的当前价
>
> m_nCloseVolume：平仓量 等于股票已经卖掉的 股票不需要
>
> m_dCloseAmount：平仓额 等于股票每次卖出的量*卖出价*合约乘数（股票为1）的累加 股票不需要
>
> m_dFloatProfit：浮动盈亏 当前量*（当前价-开仓价）*合约乘数（股票为1）
>
> m_dCloseProfit：平仓盈亏 平仓额 - 开仓价*平仓量*合约乘数（股票为1） 股票不需要
>
> m_dMarketValue：市值 合约价值
>
> m_dPositionCost：持仓成本 股票不需要
>
> m_dPositionProfit：持仓盈亏 股票不需要
>
> m_dLastSettlementPrice：最新结算价 股票不需要
>
> m_dInstrumentValue：合约价值 股票不需要
>
> m_bIsToday：是否今仓
>
> m_strStockHolder：股东账号
>
> m_nFrozenVolume：冻结数量,冻结持仓,期货不用这个字段，冻结数量
>
> m_nCanUseVolume：可用余额,可用持仓, 期货不用这个字段，股票的可用数量
>
> m_nOnRoadVolume：在途股份,在途持仓, 期货不用这个字段，股票的在途数量
>
> m_nYesterdayVolume：昨夜拥股, 期货不用这个字段，股票的股份余额

m_dLastPrice：最新价,结算价 对于股票的当前价

m_dProfitRate：盈亏比例,持仓盈亏比例

m_eFutureTradeType：EFutureTradeType 成交类型

m_strExpireDate：到期日，逆回购用

m_strComTradeID：组合成交号 , 套利成交Id

m_nLegId：组合序号 , 组合Id

m_dTotalCost：累计成本,自定义累计成本 股票信用用到

m_dSingleCost：,单股成本 ,自定义单股成本 股票信用用到

m_nCoveredVolume：备兑数量, 用于个股期权

m_eSideFlag：ESideFlag 持仓类型 , 用于个股期权，标记 '0' - 权利，'1' - 义务，'2' - '备兑'

m_dReferenceRate： 汇率,目前用于港股通

m_dStructFundVol： 分级基金可用（可分拆或可合并）

m_dRedemptionVolume： 分级基金可赎回量

m_nPREnableVolume： 申赎可用量（记录当日申购赎回的股票或基金数量）

m_dRealUsedMargin：实时占用保证金, 用于期权

m_dRoyalty： 权利金

5.4.5. CCredictDetail信用账号对象

m_dMaxMarginRate： 保证金比率 股票的保证金率等于1 股票不需要

m_dFrozenMargin：冻结保证金,外源性 股票的保证金就是冻结资金 股票不需要

m_dFrozenCash：冻结金额,内外源冻结保证金和手续费四个的和

m_dFrozenCommission：冻结手续费 ,外源性冻结资金源

m_dRisk：风险度,风险度 冻结资金/可用资金 股票不需要

m_dNav：单位净值

m_dPreBalance：期初权益 股票不需要 也叫静态权益

m_dBalance：总资产,动态权益 即市值

m_dAvailable：可用金额

m_dCommission：手续费,已经用掉的手续费

m_dPositionProfit：持仓盈亏,

m_dCloseProfit：平仓盈亏 股票不需要

m_dCashIn：出入金净值

m_dCurrMargin：当前使用的保证金 股票不需要

m_dInitBalance： 初始权益

m_strStatus：状态

m_dInitCloseMoney：期初平仓盈亏,初始平仓盈亏

m_dInstrumentValue：总市值,合约价值,合约价值

m_dDeposit：入金

m_dWithdraw：出金

m_dPreCredit：上次信用额度 股票不需要

m_dPreMortgage：上次质押 股票不需要

m_dMortgage：质押 股票不需要

m_dCredit：信用额度 股票不需要

m_dAssetBalance： 证券初始资金,股票不需要

m_strOpenDate：起始日期股票不需要

m_dFetchBalance：可取金额

m_strTradingDate：交易日

m_dStockValue：股票总市值，期货没有

m_dLoanValue：债券总市值，期货没有

m_dFundValue：基金总市值，包括ETF和封闭式基金，期货没有

m_dRepurchaseValue：回购总市值，所有回购，期货没有

m_dLongValue：多单总市值，现货没有

m_dShortValue：单总市值，现货没有

m_dNetValue：净持仓总市值，净持仓市值=多-空

m_dAssureAsset：净资产

m_dTotalDebit：总负债

m_dEntrustAsset：可信资产,用于校对

m_dInstrumentValueRMB：总市值（人民币）,沪港通

m_dSubscribeFee：申购费,申购费

m_dGoldValue：库存市值,黄金现货库存市值

m_dGoldFrozen：现货冻结,黄金现货冻结

m_dMargin：占用保证金, 维持保证金

m_strMoneyType：币种

m_dPurchasingPower：购买力, 盈透购买力

m_dRawMargin ：原始保证金

m_dBuyWaitMoney：买入待交收金额（元）,买入待交收

m_dSellWaitMoney：卖出待交收金额（元）,卖出待交收

m_dReceiveInterestTotal：本期间应计利息

m_dRoyalty：权利金收支,期货期权用

m_dFrozenRoyalty：冻结权利金 期货期权用

m_dRealUsedMargin：实时占用保证金 用于股票期权

m_dRealRiskDegree：实时风险度

m_dPerAssurescaleValue:个人维持担保比例

m_dEnableBailBalance:可用保证金

m_dUsedBailBalance:已用保证金

m_dAssureEnbuyBalance:可买担保品资金

m_dFinEnbuyBalance:可买标的券资金

m_dSloEnrepaidBalance:可还券资金

m_dFinEnrepaidBalance:可还款资金

m_dFinMaxQuota:融资授信额度

m_dFinEnableQuota:融资可用额度

m_dFinUsedQuota:融资已用额度

m_dFinUsedBail:融资已用保证金额

m_dFinCompactBalance:融资合约金额

m_dFinCompactFare:融资合约费用

m_dFinCompactInterest:融资合约利息

m_dFinMarketValue:融资市值

m_dFinIncome:融资合约盈亏

m_dSloMaxQuota:融券授信额度

m_dSloEnableQuota:融券可用额度

m_dSloUsedQuota:融券已用额度

m_dSloUsedBail:融券已用保证金额

m_dSloCompactBalance:融券合约金额

m_dSloCompactFare:融券合约费用

m_dSloCompactInterest:融券合约利息

m_dSloMarketValue:融券市值

m_dSloIncome:融券合约盈亏

m_dOtherFare:其它费用

m_dUnderlyMarketValue:标的证券市值

m_dFinEnableBalance :可融资金额

m_dDiffEnableBailBalance:可用保证金调整值

m_dBuySecuRepayFrozenMargin:买券还券冻结资金

m_dBuySecuRepayFrozenCommission:买券还券冻结手续费

> m_dSpecialEnableBalance:专项可融金额
>
> m_dEncumberedAssets:担保资产
>
> m_dSloSellBalance:融券卖出资金
>
> m_dDiffAssureEnbuyBalance:可买担保品资金调整值
>
> m_dDiffFinEnbuyBalance:可买标的券资金调整值
>
> m_dDiffFinEnrepaidBalance:可还款资金调整值
>
> m_dOtherRealCompactBalance ：其他负债合约金额
>
> m_dOtherFinCompactInterest：其他负债合约利息金额
>
> m_dUsedSloSellBalance:已用融券卖出资金
>
> m_dFetchAssetBalance:可提出资产总额
>
> m_dTotalEnableQuota:可用总信用额度
>
> m_dTotalUsedQuota:已用总信用额度;
>
> m_dDebtProfit:负债总浮盈
>
> m_dDebtLoss:负债总浮亏
>
> m_nContractEndDate:合同到期日期
>
> m_dFinDebt:融资负债
>
> m_dFinProfitAmortized:融资浮盈折算
>
> m_dSloProfit:融券浮盈
>
> m_dSloProfitAmortized:融券浮盈折算
>
> m_dFinLoss:融资浮亏
>
> m_dSloLoss:融券浮亏

5.4.6. CreditSloEnableAmount 可融券明细对象

> m_nPlatformID : 平台号
>
> m_strBrokerID : 经纪公司编号
>
> m_strBrokerName : 证券公司,期货公司,经纪公司,证券公司
>
> m_strAccountID:资金账号,账号,账号,资金账号
>
> m_strExchangeID:交易所
>
> m_strInstrumentID:证券代码
>
> m_strInstrumentName:股票名称
>
> m_dSloRatio:融券保证金比例
>
> m_eSloStatus:EXTSubjectsStatus 融券状态
>
> m_nEnableAmount:融券可融数量
>
> m_eQuerySloType:EXTSloTypeQueryMode 查询类型

> m_strExpireDate:到期日期

5.4.7. StkCompacts负债合约对象

> m_strExchangeID:交易所
>
> m_strInstrumentID:证券代码
>
> m_strExchangeName:交易所名称
>
> m_strInstrumentName:股票名称
>
> m_nOpenDate:合约开仓日期
>
> m_strCompactId:合约编号
>
> m_dCrdtRatio:融资融券保证金比例
>
> m_strEntrustNo:委托编号
>
> m_dEntrustPrice:委托价格
>
> m_nEntrustVol:委托数量
>
> m_nBusinessVol:合约开仓数量
>
> m_dBusinessBalance:合约开仓金额
>
> m_dBusinessFare:合约开仓费用
>
> m_eCompactType:EXTCompactType 合约类型
>
> m_eCompactStatus:EXTCompactStatus 合约状态
>
> m_dRealCompactBalance:未还合约金额
>
> m_nRealCompactVol:未还合约数量
>
> m_dRealCompactFare:未还合约费用
>
> m_dRealCompactInterest:未还合约利息
>
> m_dRepaidInterest:已还利息
>
> m_nRepaidVol:已还数量
>
> m_dRepaidBalance:已还金额
>
> m_dCompactInterest:合约总利息
>
> m_dUsedBailBalance:占用保证金
>
> m_dYearRate:合约年利率
>
> m_nRetEndDate:归还截止日
>
> m_strDateClear:了结日期
>
> m_strPositionStr:定位串
>
> m_dPrice:最新价
>
> m_nOpenTime:合约开仓时间
>
> m_nCancelVol:合约撤单数量
>
> m_eCashgroupProp:EXTCompactBrushSource 头寸来源

> m_dUnRepayBalance:负债金额
>
> m_nRepayPriority:偿还优先级
>
> m_dRealDefaultInterest:未还罚息
>
> m_dOtherRealCompactBalance :其他负债合约金额
>
> m_dOtherRealCompactInterest :其他负债合约利息金额

5.4.8. StkSubjects担保标的对象

> m_nPlatformID : 平台号// 目前主要用于区别不同的行情, 根据此来选择对应行情
>
> m_strBrokerID : 经纪公司编号
>
> m_strBrokerName :证券公司,期货公司,经纪公司,证券公司
>
> m_strExchangeID:交易所
>
> m_strInstrumentID:证券代码
>
> m_strInstrumentName:股票名称
>
> m_dSloRatio:融券保证金比例
>
> m_eSloStatus:EXTSubjectsStatus 融券状态
>
> m_nEnableAmount:融券可融数量
>
> m_dFinRatio:融资保证金比例
>
> m_eFinStatus:EXTSubjectsStatus 融资状态
>
> m_dAssureRatio担保品折算比例
>
> m_eAssureStatus:EXTSubjectsStatus 是否可做担保
>
> m_dFinRate: 融资日利率
>
> m_dSloRate: 融券日利率
>
> m_dFinPenaltyRate: 融资日罚息利率
>
> m_dSloPenaltyRate:融券日罚息利率
>
> m_strAccountID:资金账号,账号,账号,资金账号
>
> m_eAssureUseSloCashStatus:EXTSpecialAssure:是否可以用融券资金买入

5.4.9. 对象中属性一些需要的状态字段释义

5-1.enum_EEntrustBS //买卖方向

> ENTRUST_BUY|买入,多,:48; // 买入
>
> ENTRUST_SELL|卖出,空,:49; // 卖出
>
> ENTRUST_PLEDGE_IN|质押入库:81; // 质押入库
>
> ENTRUST_PLEDGE_OUT|质押出库:66; // 质押出库

5-2.EEntrustSubmitStatus//报单状态

> 48 已经提交
>
> 49 撤单已经提交
>
> 50 修改已经提交
>
> 51 已经接受
>
> 52 报单已经被拒绝
>
> 53 撤单已经被拒绝
>
> 54 改单已经被拒绝

5-3.enum_EEntrustTypes //委托类型

> ENTRUST_BUY_SELL :48; // 买卖
>
> ENTRUST_QUERY :49; // 查询
>
> ENTRUST_CANCEL :50; // 撤单
>
> ENTRUST_APPEND :51; // 补单
>
> ENTRUST_COMFIRM :52; // 确认
>
> ENTRUST_BIG :53; // 大宗
>
> ENTRUST_FIN :54; // 融资委托
>
> ENTRUST_SLO :55; // 融券委托
>
> ENTRUST_CLOSE :56; // 信用平仓
>
> ENTRUST_CREDIT_NORMAL :57; // 信用普通委托
>
> ENTRUST_CANCEL_OPEN :58; // 撤单补单
>
> ENTRUST_TYPE_OPTION_EXERCISE :59; // 行权
>
> ENTRUST_TYPE_OPTION_SECU_LOCK :60; // 锁定
>
> ENTRUST_TYPE_OPTION_SECU_UNLOCK :61; // 解锁

5-4.enum_EEntrustStatus //委托状态

> ENTRUST_STATUS_WAIT_END: 0; //委托状态已经在ENTRUST_STATUS_CANCELED或以上，但是成交数额还不够，等成交回报来
>
> ENTRUST_STATUS_UNREPORTED :48; // 未报
>
> ENTRUST_STATUS_WAIT_REPORTING :49; // 待报
>
> ENTRUST_STATUS_REPORTED :50; // 已报
>
> ENTRUST_STATUS_REPORTED_CANCEL :51; // 已报待撤
>
> ENTRUST_STATUS_PARTSUCC_CANCEL :52; // 部成待撤
>
> ENTRUST_STATUS_PART_CANCEL :53; // 部撤

> ENTRUST_STATUS_CANCELED :54; // 已撤
>
> ENTRUST_STATUS_PART_SUCC :55; // 部成
>
> ENTRUST_STATUS_SUCCEEDED :56; // 已成
>
> ENTRUST_STATUS_JUNK :57; // 废单
>
> ENTRUST_STATUS_DETERMINED :86; //已确认
>
> ENTRUST_STATUS_UNKNOWN :255; // 未知

5-5.enum_EHedge_Flag_Type

> HEDGE_FLAG_SPECULATION :49; // 投机
>
> HEDGE_FLAG_ARBITRAGE :50; // 套利
>
> HEDGE_FLAG_HEDGE :51; // 套保

5-6.enum_EFutureTradeType // 成交类型

> FUTRUE_TRADE_TYPE_COMMON :48; // 普通成交
>
> FUTURE_TRADE_TYPE_OPTIONSEXECUTION :49; // 期权成交ptionsExecution
>
> FUTURE_TRADE_TYPE_OTC :50; // OTC成交
>
> FUTURE_TRADE_TYPE_EFPDIRVED :51; // 期转现衍生成交
>
> FUTURE_TRADE_TYPE_COMBINATION_DERIVED :52; // 组合衍生成交

5-7.enum_EBrokerPriceType // 价格类型

> BROKER_PRICE_ANY :49; // 市价
>
> BROKER_PRICE_LIMIT :50; // 限价
>
> BROKER_PRICE_BEST :51; // 最优价
>
> BROKER_PRICE_PROP_ALLOTMENT :52; // 配股
>
> BROKER_PRICE_PROP_REFER :53; // 转托
>
> BROKER_PRICE_PROP_SUBSCRIBE :54; // 申购
>
> BROKER_PRICE_PROP_BUYBACK :55; // 回购
>
> BROKER_PRICE_PROP_PLACING :56; // 配售
>
> BROKER_PRICE_PROP_DECIDE :57; //指定
>
> BROKER_PRICE_PROP_EQUITY :58; // 转股
>
> BROKER_PRICE_PROP_SELLBACK :59; // 回售
>
> BROKER_PRICE_PROP_DIVIDEND :60; // 股息
>
> BROKER_PRICE_PROP_SHENZHEN_PLACING :68; // 深圳配售确认
>
> BROKER_PRICE_PROP_CANCEL_PLACING :69; // 配售放弃

BROKER_PRICE_PROP_WDZY :70; // 无冻质押

BROKER_PRICE_PROP_DJZY :71; // 冻结质押

BROKER_PRICE_PROP_WDJY :72; // 无冻解押

BROKER_PRICE_PROP_JDJY :73; // 解冻解押

BROKER_PRICE_PROP_ETF :81; // ETF申购

BROKER_PRICE_PROP_VOTE :75; // 投票

BROKER_PRICE_PROP_YYSGYS :92; // 要约收购预售

BROKER_PRICE_PROP_YSYYJC :77; // 预售要约解除

BROKER_PRICE_PROP_FUND_DEVIDEND; // 基金设红

BROKER_PRICE_PROP_FUND_ENTRUST :79; // 基金申赎

BROKER_PRICE_PROP_CROSS_MARKET :80; // 跨市转托

BROKER_PRICE_PROP_EXERCIS :83; // 权证行权

BROKER_PRICE_PROP_PEER_PRICE_FIRST :84; // 对手方最优价格

BROKER_PRICE_PROP_L5_FIRST_LIMITPX :85; // 最优五档即时成交剩余转限价

BROKER_PRICE_PROP_MIME_PRICE_FIRST :86; // 本方最优价格

BROKER_PRICE_PROP_INSTBUSI_RESTCANCEL :87; // 即时成交剩余撤销

BROKER_PRICE_PROP_L5_FIRST_CANCEL :88; // 最优五档即时成交剩余撤销

BROKER_PRICE_PROP_FULL_REAL_CANCEL :89; // 全额成交并撤单

BROKER_PRICE_PROP_DIRECT_SECU_REPAY :101; // 直接还券

BROKER_PRICE_PROP_FUND_CHAIHE :90; // 基金拆合

BROKER_PRICE_PROP_DEBT_CONVERSION :91; // 债转股

BROKER_PRICE_BID_LIMIT :92; // 港股通竞价限价

BROKER_PRICE_ENHANCED_LIMIT :93; // 港股通增强限价

BROKER_PRICE_RETAIL_LIMIT :94; // 港股通零股限价

BROKER_PRICE_PROP_INCREASE_SHARE :'j'; // 增发

BROKER_PRICE_PROP_COLLATERAL_TRANSFER :107; // 担保品划转

BROKER_PRICE_PROP_NEEQ_PRICING :'w'; // 定价（全国股转-挂牌公司交易-协议转让）

BROKER_PRICE_PROP_NEEQ_MATCH_CONFIRM :'x'; // 成交确认（全国股转-挂牌公司交易-协议转让）

BROKER_PRICE_PROP_NEEQ_MUTUAL_MATCH_CONFIRM :'y'; // 互报成交确认（全国股转-挂牌公司交易-协议转让）

BROKER_PRICE_PROP_NEEQ_LIMIT :'z'; // 限价（用于挂牌公司交易-做市转让-限价买卖和两网及退市交易-限价买卖）

5-8.enum_EOffset_Flag_Type //操作类型

> EOFF_THOST_FTDC_OF_INVALID: -1;
>
> EOFF_THOST_FTDC_OF_Open|买入,开仓 :48; // 开仓
>
> EOFF_THOST_FTDC_OF_Close|卖出,平仓 :49; // 平仓
>
> EOFF_THOST_FTDC_OF_ForceClose :50; // 强平
>
> EOFF_THOST_FTDC_OF_CloseToday :51; // 平今
>
> EOFF_THOST_FTDC_OF_CloseYesterday :52; // 平昨
>
> EOFF_THOST_FTDC_OF_ForceOff :53; // 强减
>
> EOFF_THOST_FTDC_OF_LocalForceClose :54; // 本地强平
>
> EOFF_THOST_FTDC_OF_PLEDGE_IN :81; // 质押入库
>
> EOFF_THOST_FTDC_OF_PLEDGE_OUT :66; // 质押出库
>
> EOFF_THOST_FTDC_OF_ALLOTMENT :67; // 股票配股

5-9.enum EXTSubjectsStatus //融资融券状态

> SUBJECTS_STATUS_NORMAL: 48;//正常
>
> SUBJECTS_STATUS_PAUSE: 49;//暂停
>
> SUBJECTS_STATUS_NOT : 50;//作废

5-10.enum EXTSloTypeQueryMode //查询类型

> XT_SLOTYPE_QUERYMODE_NOMARL: 48;//普通
>
> XT_SLOTYPE_QUERYMODE_SPECIAL: 49;//专项

5-11.enum EXTCompactType //合约类型

> COMPACT_TYPE_ALL:32//不限制
> COMPACT_TYPE_FIN:48//融资
> COMPACT_TYPE_SLO:49//融券

5-12 enum EXTCompactStatus //合约状态

> COMPACT_STATUS_ALL:32//不限制
>
> COMPACT_STATUS_UNDONE: 48//未归还
>
> COMPACT_STATUS_PART_DONE:49//部分归还
>
> COMPACT_STATUS_DONE:50//已归还
>
> COMPACT_STATUS_DONE_BY_SELF:51//自行了结
>
> COMPACT_STATUS_DONE_BY_HAND:52//手工了结
>
> COMPACT_STATUS_NOT_DEBT:53//未形成负债
>
> COMPACT_STATUS_EXPIRY: 54//合约已过期

5-13.enum EXTCompactBrushSource //头寸来源

> XT_COMPACT_BRUSH_SOURCE_ALL:32//不限制
> XT_COMPACT_BRUSH_SOURCE_NORMAL:48 //普通头寸
> XT_COMPACT_BRUSH_SOURCE_SPECIAL:49 //专项头寸

5-14.enum_EXTSpecialAssure //是否可以用融券资金买入

> ASSURE_USE_SLO_CASH_DISABLE: 48 //担保品买入不允许使用融券资金
>
> ASSURE_USE_SLO_CASH_ENABLE: 49 //担保品买入允许使用融券资金