中国外汇交易中心增值服务项目

量化-API用户手册

文档目录

第一章 策略模板

1.1 模板代码

```
from ComstarApi import *

def init(context):
    """
    初始化方法，仅在策略启动时执行一次，在方法内部，可以进行如下操作：
    1. 通过add_parameter方法添加参数
    2. 通过add_factor方法添加因子
    3. 通过context.xxx的方式添加全局变量
    """
    pass
def on_tick(context, tick):
    """
    逐笔成交及报价行情事件处理方法，仅在有逐笔成交或报价行情时被触发
    """
    pass
def on_bar(context, bar):
    """
    K-Bar事件处理方法，仅在有K-Bar行情时被触发
    """
    pass
@order_deal
def on_order(context, order):
    """
    订单事件处理方法，仅在新建订单及旧订单的订单状态发生变化时被触发
    成交会导致订单状态发生变化，因此可归类于旧订单的订单状态发生变化的情况
    """
    pass
def on_new_order(context, order):
    """ 新订单处理方式 """
    pass
def on_all_traded(context, order):
    """ 全部成交处理方式 """
    pass
def on_part_traded(context, order):
    """ 部分成交处理方式 """
    pass
def on_canceled(context, order):
    """ 撤单成功处理方式 """
    pass
def on_failed(context, order):
    """ 下单失败处理方式 """
    pass
```

第二章 API详情说明

2.1 事件触发API

- **用途：**
  行情、订单变化会触发调用的客户代码区。
- **API总览：**

序号	API名称	说明
1	init	策略信息初始化
2	on_order	订单状态信息推送
3	on_tick	逐笔行情推送
4	on_bar	K线行情推送
5	on_timer	定时器触发推送

2.1.1 init: 策略信息初始化

- **格式：**

```
def init(
        context,  #上下文对象
        )
```

- **参数说明：**

参数名	类型	说明
context	上下文对象	包含上下文信息，在各个事件响应函数中会作为参数传入，属性可通过点标记（'.'）定义以及获取。

- **功能：**
  init，策略初始化代码有关，与行情频率无关，只运行 1 次，用户策略可以在其中定义全局变量、创建因子、设置策略自定义参数、设置定时器等。
- **注意：**
  context 的用户自定义属性不可与系统使用字段重名，系统使用变量名详见 context 上下文变量。
- **调用频率：**
  策略启动时运行 1 次。用户不可调用。

- **返回值：**
  无。
- **示例：**

```
def init(context):
      context.aaa = 1
```

初始化时定义全局属性 aaa，并赋值为 1，定义后可全局使用该属性。

2.1.2 on_order: 订单状态信息推送

- **格式：**

```
def on_order(
            context,              #上下文对象
            order: OrderData      #订单对象
            )
```

- **参数说明：**

参数名	类型	说明
context	上下文对象	包含上下文信息，在各个事件响应函数中会作为参数传入，属性可通过点标记（'.'）定义以及获取。
order	Order	Order 类型的对象。

- **功能：**
  订单发生变化后，系统调用本函数，on_order 中的用户逻辑代码会被执行 1 次。任何原因导致的订单状态变动都会触发 on_order 调用，用户可以根据当前 order.order_status 识别触发原因：例如：NEW 表示订单已被中心确认新增、PARTIAL 表示有部分成交、FINISHED 表示有成交且订单已完成。CANCELED 说明因撤销订单导致 on_order 调用等。order_status 枚举值请见 OrderStatus 枚举类。
- **注意：**
  在 on_order 中，order.order_status 是最新状态，表示 on_order 本次触发的原因，此时在 on_order 中看到的损益、头寸等信息已是最新。
- **调用频率：**
  order 对象发生变化时（成交或订单状态转变），系统触发 on_order 调用。用户不可调用。
- **返回值：**
  无。
- **示例：**
  无。

2.1.3 on_tick: 逐笔行情推送

- **格式:**

```
def on_tick(
            context,          #上下文对象
            tick: TickData    #TickData对象
            )
```

- **参数说明:**

参数名	类型	说明
context	上下文对象	包含上下文信息，在各个事件响应函数中会作为参数传入，属性可通过点标记（'.'）定义以及获取。
tick	TickData	成交或盘口行情数据。

- **功能:**
  当有新的成交或盘口数据时都会触发 on_tick，用户策略如果是 tick 或 quote 策略，可以把代码写在该函数中。tick.category 字段可以识别该 tick 是逐笔成交 'deal'，还是盘口 'quote'。
- **注意:**
  无。
- **调用频率:**
  每当有 tick 或 quote 行情时系统触发 on_tick 调用。用户不可调用。
- **返回值:**
  无。
- **示例:**
  无。

2.1.4 on_bar: K线行情推送

- **格式:**

```
def on_bar(
           context,         #上下文对象
           bar: BarData     #BarData对象
           )
```

- **参数说明:**

参数名	类型	说明
context	上下文对象	包含上下文信息，在各个事件响应函数中会作为参数传入，属性可通过点标记（'. '）定义以及获取。
bar	BarData	bar 模式：成交行情的 Bar 数据 示意样例{('210210. CFXO', '1') : 1 分钟 BarData, ('210210. CFXO', '5') : 5 分钟 BarData}

- **功能：**
  当有新的 bar 数据时都会触发 on_bar，用户策略随 bar 行情而变的策略逻辑代码可以写在该函数中。
  bar.category 字段目前只有成交 'deal'。
- **Bar 模式：**

  on_bar 触发模式：不同 bar 分别到达，各自触发 on_bar。
- **注意：**
  无。
- **调用频率：**
  每当有 bar 行情时系统触发 on_bar 调用。用户不可调用。
- **返回值：**
  无。
- **示例：**

  bar 模式下，用户在 on_bar 中使用时，bar 是 BarData 类型，可直接用bar. close 来获取该 bar 的收盘价。

2.1.5 on_timer: 定时器触发推送

- **格式：**

```
def on_timer(
            context,           #上下文对象
            timer: TimerData   #时间类型对象
            )
```

- **参数说明：**

参数名	类型	说明
context	上下文对象	包含上下文信息，在各个事件响应函数中会作为参数传入，属性可通过点标记（'. '）定义以及获取。
timer	TimerData	有成员属性 trigger_time: str，触发时间样例：'2022-01-01 00: 00: 00' comments: str，时间规则备注 month: int，月 weekday: int，星期几 day: int，日期 hour: int，小时 minute: int，分钟 second: int，秒

- **解释：**
  定时时间一到，系统会调用本函数，on_timer 中的用户逻辑代码会被执行 1 次。使用该函数的前提是至少在 init 中配置了 1 条定时的时间规则（add_timer 函数），每个时间规则中的候选时间都可以触发 on_timer。用户可以将希望在固定时间进行的业务逻辑写到 on_timer 中，以实现定时触发用户逻辑。
- **调用频率：**
  定时被系统调用。用户不可调用。

2.2 其他函数

- **API总览：**

序号	名称	说明
1	context	上下文变量（全局变量）
2		行情设置
3	add_parameter	自定义参数
4	add_factor	自定义因子
5	add_timer	添加定时器触发规则

2.2.1 context: 上下文变量（全局变量）

- **解释：**

  可用于自定义跨越周期的全局变量，在各个事件响应函数中会作为参数传入，属性可通过点标记 (' . ') 定义以及获取。

- **自定义 context 属性示例：**

  在 init 中自定义全局变量 context. china = 666，则可在 on_tick、on_bar、on_order 等有参数 context 的函数中使用 context. china，其值为 666。

2.2.2 行情设置

- **说明：**

  在配置文件中配置订阅行情，全域变量设置为cs1、cs2、...、cs3，并得到对象 cs1、cs2、... 、cs8 等与行情有关的对象，可在策略中使用时，可以通过 context. cs1（cs2、cs3、... 、cs8 等以此类推）来引用有关字段。

  以 context. cs1 为例：

  1）context. cs1. cic_code 用于识别行情的代码。str 类型，如 200215. CFXO。

  2）context. cs1. interval 用于识别行情的频率。str 类型，如 0,1,5,10。

  3）context. cs1. platform 用于填 send_order 时的 platform_code 参数。str 类型。

  4）context. cs1. match 指示行情是否用于撮合，bool 类型。match 是否为 True 表示是否用于撮合，同一个券种只有 1 个订阅的行情的 match 可以为 True。

  5）context. cs1. security_id 券号，str 类型，如 200215，同 symbol。

  6）context. cs1. category，行情类型，包括盘口 'quote'，成交 'deal'。

  用户在发单等场景中要填写 cic_code、platform 等都可以用如上方式获得。

- **注意：**

同一种资产（相同券号）只能有 1 个行情的 match 可以为 True，该资产其余行情的 match 为 False。

- **参数说明：**

参数名	类型	说明
cic_code	str	cic 码
interval	str	字符类型的 1，表示 1 分钟 Bar 字符类型的 5，表示 5 分钟 Bar 字符类型的 10，表示 10 分钟 Bar 字符类型的 30，表示 30 分钟 Bar 字符类型的 60，表示 60 分钟 Bar 字符类型的 1D，表示 1 日 Bar 字符类型的 0，表示 tick、quote 的频率
platform	str	发单平台码
match	bool	是否用于撮合，True 或 False
security_id	str	券号，如 200215
category	str	行情类型，盘口：'报价行情'，成交：'逐笔成交'，bar：'K-bar'

2.2.3 add_parameter: 自定义参数

- **格式：**

```
add_parameter(kind, name, value, combo_values)
```

- **参数说明：**

参数名	类型	说明
kind	type	参数的类型，可填：float、int、str。
name	str	变量名称，用户自定义，不可重复。
value	str	变量的注释说明，不可重复。
combo_values	list	默认值为空列表 [ ]，则支持在界面手动填值，如果设置内容类似 [10, 15, 20]，则支持在界面下拉列表选择值。list 中元素值的类型要与 parameter_type 设置的类型一致。

- **功能：**

  定义代码中开放可在界面配置的参数名称和类型，用全局变量接收其返回值。当编译完成时，会在界面上出现相应的自定义策略参数，待用户填充后运行时，填充的值将在策略中被该函数的变量获得。通过 combo_values 参数可以预设参数的候选值供用户在界面上选择。

- **调用频率：**

只可在 init 中调用。不可在其他地方使用。

- **返回值：**

  在 init 中定义全局变量来接收该函数的返回值，这些变量在 init 中不能进行运算操作，而是用于在其他地方引用自定义策略参数。

- **备注：**

  配置多个参数不能重名。

  combo_values 可以使参数具有多个固定的预设值，并在界面上以下拉列表形式呈现。

  如果 combo_values 填 [ ]，则仍然在界面上以输入框呈现，等待用户手动填写。

  combo_values 的 list 中元素值的类型要与 parameter_type 设置的类型一致。

- **示例：**

  - 方式 1 无候选值：

    步骤 1：init 中进行策略参数定义：

    (注：name 取值必须和等号左边 context 的属性名称相同，注意 context 中属性命名唯一性。)

    ```
    context.volume = add_parameter(str, 'volume', '每手交易量')
    ```

    步骤 2：代码编译后界面上看到这个自定义的策略参数，用户为其填值 10000000。

    步骤 3：on_bar（或 on_tick、或 on_order）中引用该参数：

    全局变量 context. volume 的值即为 10000000。

  - 方式 2 有候选值：

    步骤 1：init 中进行策略参数定义：

    ```
    context.volume = add_parameter(str, 'volume', '每手交易量',
                            combo_values=[10000000,20000000,30000000])
    ```

    步骤 2：代码编译后界面上看到这个自定义的策略参数，用户为从下拉列表中选 20000000。

    步骤 3：on_bar（或 on_tick、或 on_order）中引用该参数：

    全局变量 context. volume 的值即为 20000000。

2.2.4 add_factor: 自定义因子

- **格式：**

  ```
  add_factor(kind, name, value)
  ```

- **参数说明：**

参数名	类型	说明
kind	type	参数的类型，可填：float、int、str。
name	str	变量的名称，必须和等号左边 context 的属性名称相同，用户自定义，不可重复。
value	str	变量在界面上展示的名称，用户自定义，不可重复。

- **功能：**

  定义代码中开放可在界面配置的因子名称和类型。

- **调用频率：**

  该函数只能在 init 中调用。不可在其他地方使用。

- **示例：**

  步骤1：init 中进行因子定义：

```
context. volume = add_factor (int, 'volume', '累加报价量')
```

  步骤2：on_bar（或 on_tick、或 on_order）中进行因子赋值：context. volume = 1.23，则该 bar（tick、order 变化）时刻该因子被赋值为 1.23。

  步骤3：当回测运行完后，因子 volume 将存下期间变化的全部值。

- **备注：**

  1）name 取值必须和等号左边 context 的属性名称相同，注意 context 中属性命名唯一性。

  2）因子每次赋值的标准用法为使用 float 或 int 类型单个值。目前也支持使用列表直接赋值，会自动取列表最后 1 个值用于因子赋值，如果列表为空则无操作。

  3）如果本次因子用 nan 或 None 赋值，则相当于没有新因子，会仍然保持前值。

  4）变量初始是None，需要用户自己去初始化

2.2.5 add_timer: 添加定时触发的时间规则

- **格式：**

```
def add_timer(
                month,      #月
                weekday,    #星期几
                day,        #日
                hour,       #小时
                minute,     #分钟
                second,     #秒
                time_str,   #时间
                comments    #备注
               )
```

- **解释：**

- add_timer 函数，用于配置 on_timer 的触发时间规则。

  可指定触发的月、星期几、日、时、分或具体时间。月、星期几、日、时、分，秒可用 int 填单值，也可以用 list 类型填多值。

- time_str 参数：优先级最高，直接指定时间字符串，默认为 None 表示不设置。如果 time_str 不为 None 则屏蔽所有其他时间条件参数。时间格式为 "2022-01-01 00:00:00.000"

- months、weekday、day、hour、minute、second 参数：当 time_str 为 None 时这 6 个参数才生效。

  weekday = 6，表示触发时间在星期六。weekday = [6, 7] 表示触发时间在周六和周日。

  weekday = None，表示不限制星期几，即任何一天都触发，相当于

  weekday = [1, 2, 3, 4, 5, 6, 7]。

  month的概念类似于weekday。

  day参数：如果为 None，表示不限制天，即每天。若指定天则表示设定月内对应的某天。

  hour 参数：如果为 None，表示不限制小时，即每个小时。若指定小时则表示设定相应的小时。
  minute 参数：为 None 或 0 都是整点触发，只有显性指定分钟才会设定分钟。

  second参数：如果为 None，表示不限制秒，即秒。若指定秒则表示设定相应的秒

- comments 参数：用户为这条时间规则设置的备注，这样辅助用户在 on_timer 中区分触发的原因。

- **调用频率：**
  该函数只能在 init 中调用。不可在其他地方使用。

- **举例：**

  1.

  ```
  add_timer(time_str='2022-01-01 10:00:00', comments='活跃时段发单逻辑 1')
  ```

  指定了 time_str 具体时间，所以只会在 2022-01-01 10:00:00 触发一次 on_timer。

  2.

  ```
  add_timer(weekday=1, hour=9, minute=30, comments='每周一开盘触发逻辑 1')
  ```

  设定每周一 9 点 30 分触发一次 on_timer。

  3.

  ```
  add_timer(weekday=[1,3,5], hour=[9,10,11,12,13,14,15,16])
  ```

  设定每周一、三、五，且 9 点到 16 点，整点触发

  4.

  ```
  add_timer(minute=[0,5,10,15,20,25,30,35,40,45,50,55], second=[10])
  ```

  设定每隔 5 分中在10秒时触发。

- **参数说明：**

参数名	类型	说明
month	Union[List,int]	月，可填None或list或int，默认None
weekday	Union[List,int]	星期几，可填None或list或int，默认None
day	Union[List,int]	日，可填None或list或int，默认None
hour	Union[List,int]	小时，可填None或list或int，默认None
minute	Union[List,int]	分钟，可填None或list或int，默认0
second	Union[List,int]	秒，可填None或list或int，默认0
time_str	str	直接指定str类型时间，默认None
comments	str	备注，可推入on_timer中的timer里，默认空字符串

2.3 行情访问API

- **用途：**
  使用户可以从不同维度获取行情
- **API总览：**

序号	API名称	说明
1	get_deal_data( )	获取历史逐笔成交行情
2	get_bar_data( )	获取历史成交 Bar 行情
3	get_quote_data( )	获取历史盘口行情
4	sd( )	获取时序数据
5	ss( )	获取截面数据
6	edb( )	获取键值查询数据

2.3.1 get_deal_data: 获取历史逐笔成交行情

- **格式：**

```
def get_deal_data(cic_code:str,          #cic码
                  count:int,             #get最近n个deal（包含当前最新）
                  fields:List[str],      #指定返回的域
                 )
```

- **参数说明：**

参数名	类型	说明
cic_code	str	cic 码唯一识别行情
count	int	向前查 count 个 deal（含当前）
fields	List[str]	待查询的行情域 参数填写方式：查询可填 ['volume', 'price']

- **fields的可选域**：

字段	含义
time	时间
volume	量
price	价格
yield	收益率

- **功能：**
  成交行情按数量查询函数，直接调用可以获取向前取 count 个（包括当前）行情。其中行情的具体字段由 fields 参数指定。
- **调用频率：**
  可在 init 之外的函数中调用。
- **注意：**
  1）要获得逐笔成交必须在行情设置中订阅逐笔行情。
  2）get_deal_n使用方式同get_deal_data，仅函数名不同（get_deal_n将在下个大版本移除）
- **返回值：**
  最大返回 500 条。返回字典，fields 为字典的键，数据为字典的值，值的类型为 list 。如果行情的某个域缺失数据，用 None 填充对齐，各个域查询的结果长度一致。
- **示例：**
  在行情设置中设置了 cs1 之后

```
his_data = get_deal_n(cic_code=context.cs1.cic_code,
                      count=10, #取最近10个成交行情（含当前行情）
                      fields=['volume','price'])
```

his_data 返回的值可能为

```
{
'volume':
[10000000,10000000,20000000,10000000,20000000,10000000,10000000,20000000,10000000,20000000],
'price':[100.1,100.2,100.3,100.2,100.3,100.1,100.2,100.3,100.2,100.3]
}
```

其中 his_data['volume'] 即为从当前往前数 10 个 deal 的 volume 列表。

2.3.2 get_bar_data: 获取历史成交 Bar 行情

- **格式：**

```
def get_bar_data(cic_code: str,          #cic码
                 interval: str,          #bar频率
                 count: int,             #get最近n个bar
                 fields: List[str],      #指定返回数据域
             )
```

- **参数说明：**

参数名	类型	说明
cic_code	str	cic 码唯一识别行情，详情见行情设置
interval	Interval 枚举类	Interval.ONE_MINUTE：1 分钟 Bar， 也可以直接用 '1' Interval.FIVE_MINUTE：5 分钟 Bar，也可以直接用 '5' Interval.TEN_MINUTE：10 分钟 Bar ，也可以直接用'10' Interval.HALF：30 分钟 Bar，也可以直接用 '30' Interval.HOUR：1 小时 Bar，也可以直接用 '60' Interval.DAILY：日 Bar，也可以直接用 '1D'
count	int	向前查 count 个 bar（含当前）
fields	List[str]	待查询的行情域 参数填写方式：单一域查询可填 'high' 或 ['high']，多个域查询可填 ['high', 'open']

- **fields的可选域**：

字段	含义
high	最高价
open	开盘价
low	最低价
close	收盘价
high_yield	最高收益率
open_yield	开盘收益率
low_yield	最低收益率
close_yield	收盘收益率
volume	量
time	时间

- **功能：**

  bar 行情查询。向前取 count 个（包括当前）行情。行情的具体字段由 fields 参数指定。

- **调用频率：**

  可在 init 之外的函数中调用。

- **注意：**

  1）要获得 n 分钟 bar 必须在行情设置中订阅 n 分钟 bar 行情。

  2）get_bar_n使用方式同get_bar_data，仅函数名不同（get_bar_n将在下个大版本移除）

- **返回值：**

  最大返回 500 条。返回字典，fields 为字典的键，数据为字典的值，值的类型为 list。如果行情的某个域缺失数据，用 None 填充对齐，各个域查询的结果长度一致。

- **示例：**

```
his_data = get_bar_data(cic_code=context.cs1.cic_code,
                        interval=Interval.FIVE_MINUTE,
                        count=10,                        #最近10个含当前
                        fields=['volume', 'open'])
```

  his_data 返回的值可能为

```
{
'volume':
[10000000,10000000,20000000,10000000,20000000,10000000,10000000,20000000,10000000,2
0000000] ,
'open':[100.1,100.2,100.3,100.2,100.3,100.1,100.2,100.3,100.2,100.3]
}
```

  his_data['open'] 即为最近 10 个（含当前）5 分钟 bar 的 open 价。

2.3.3 get_quote_data: 获取历史盘口行情

- **格式：**

```
def get_quote_data(cic_code:str,          #cic码
                   count:int,             #取几个盘口
                   fields:List[str],      #指定返回数据域
              )
```

- **参数说明：**

参数名	类型	说明
cic_code	str	cic 码唯一识别行情
count	int	向前查 count 个 quote
fields	List[str]	待查询的行情域 参数填写方式： 按域查询可填 ['offer_time', 'offer_price']

- **fields的可选域**：

字段	含义
bid_price	盘口买价，二维列表，每行一个盘口
bid_volume	盘口买量，二维列表，每行一个盘口
bid_time	盘口买时间 , 1 维列表
bid_yield	盘口买收益率 , 二维列表，每行一个盘口
offer_price	盘口卖价 , 二维列表，每行一个盘口
offer_volume	盘口卖量 , 二维列表，每行一个盘口
offer_time	盘口卖时间 , 1 维列表
offer_yield	盘口卖收益率 , 二维列表，每行一个盘口

- **功能：**

  盘口行情查询（包括当前）。取 n 个盘口行情，行情的具体字段由 cic_code 的 quote 行情的fields 字段指定。盘口一共 10 档，二维列表（n 行 10 列），每行为 1 个盘口含 10个元素表示 10个档位，行数为查到的盘口数量。

- **调用频率：**

  可在 init 之外的函数中调用。

- **注意：**

  1）要获得盘口数据必须在行情设置中订阅盘口行情。

  2）get_quote_n使用方式同get_quote_data，仅函数名不同（get_quote_n将在下个大版本移除）

- **返回值：**

  最大返回 500 条。返回字典，fields 为字典的键，数据为字典的值，值的类型为 list。如果某个盘口行情 bid_price 只有 2 档，则其余 8 档会用 None 填充，如果某个盘口的 bid_price 都没有则该盘口为 10个 None，其余字段同理。

- **示例：**

```
his_data = get_quote_n(cic_code=context.cs1.cic_code,
                       count=2,  #最近2个盘口（包含当前最新）
                       fields=['offer_volume', 'bid_price'])
```

his_data 返回的值可能为

```
{
'offer_volume':
[[10000000,20000000,10000000,10000000,10000000,20000000,10000000,10000000,10000000,
20000000],

[10000000,20000000,10000000,10000000,10000000,20000000,10000000,10000000,10000000,2
0000000]],
'bid_price':
[[100.12,100.12,100.08,100.06,100.05,100.04,100.03,100.02,100.01,100.03],
             [100.4,100.3,100.1,100.06,100.03,99.98,100.5,100.4,100.3,100.1,100.2]]
}
```

his_data['bid_volume'] 即为含当前 quote 共 2 个的 bid_ volume 的列表。

2.3.4 sd: 获取时序数据

- **格式：**

```
def sd(cic_code:List[str],          #cic码
       category:str,                #行情类型
       field: str,                  #查询域
       start_time: str = None,      #行情开始时间
       end_time: str = None,        #行情结束时间
       **kwargs                     #可选参数
             ) -> pandas.DataFrame
```

- **参数说明：**

参数名	类型	说明
cic_code	List[str]	cic 码唯一识别行情,支持多资产查询
category	str	行情类型，category为“1”或者“3”，“1”为quote行情、“3”为deal行情
field	str	行情查询域，只支持单域查询 可通过dir(comstarlib)查看所有可查询域
start_time	str	查询行情开始时间
end_time	str	查询行情结束时间
**kwargs		可选参数，若聚合生成K线数据，必须传递interval字段，参考示例

- **fields的可选域**：

  1） Tick行情可选域

参数名	含义
transact_time	时间
volume	量
price	价格
tick_yield	收益率

  2）Quote行情可选域

参数名	含义
transact_time	时间
bid_price	盘口买价，为list类型，默认所有档位
bid_volume	盘口买量，为list类型，默认所有档位
bid_yield	盘口买收益率，为list类型，默认所有档位
offer_price	盘口卖价，为list类型，默认所有档位
offer_volume	盘口卖量，为list类型，默认所有档位
offer_yield	盘口卖收益率，为list类型，默认所有档位
bid_price_x	盘口买价，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，bid_price_1；n可通过查询bid_price获取具体为几档，对应取值
bid_volume_x	盘口买量，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，bid_volume_1
bid_yield_x	盘口买收益率，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，bid_yield_1
offer_price_x	盘口卖价，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，offer_price_1
offer_volume_x	盘口卖量，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，offer_volume_1
offer_yield_x	盘口卖收益率，x为具体取哪一档位的盘口数据，可选值1-n，例：取第一档，offer_yield_1

3）Bar行情可选域

参数名	含义
transact_time	时间
open	开盘价
high	最高价
close	收盘价
low	最低价
open_yield	开盘收益率
high_yield	最高收益率
close_yield	收盘收益率
low_yield	最低收益率
volume	量

- **关键字参数kwargs可选域**：

参数名	含义
interval	K线生成频率，Interval对象，详见参见 Interval 类
count	查询行情数量，默认为5000

- **功能：**

  查询行情的详细信息，不设置筛选条件，默认返回5000条。

- **调用频率：**

  可在 init 之外的函数中调用。

- **注意：**

  1）查询必须指定cic_code和category（用于区分行情类型）。

  2）若根据Tick行情聚合K线，必须指定interval的值，interval的类型为Interval，详见参见 Interval 类

- **返回值：**

  返回类型为pandas.DataFrame

- **示例：**

  1）查询报价行情，category="1"

  - 单cic查询，查询域为bid_price具体档位，指定查询返回数据为5条

```
result = comstarlib.sd(cic_code=["USDCNY-CFXO"], category="1", field="bid_price_1",
count=5)
```

result 返回的值为pandas.DataFrame，返回结构如下图所示，index为transact_time。

```
                      USDCNY-CFXO
transact_time
20230607-17:12:30.556        7.12
20230607-17:12:33.057        7.12
20230608-18:22:16.769      7.1222
20230609-14:02:28.690      7.1182
20230609-14:33:31.199      7.1182
```

- 单cic查询，查询域为bid_price所有档位

```
result = comstarlib.sd(cic_code=["USDCNY-CFXO"], category="1", field="bid_price",
count=5)
```

result返回值为list，如下：

```
[                     USDCNY-CFXO
transact_time
20230607-17:12:30.556        7.12
20230607-17:12:33.057        7.12
20230608-18:22:16.769      7.1222
20230609-14:02:28.690      7.1182
20230609-14:33:31.199      7.1182,                      USDCNY-CFXO
transact_time
20230607-17:12:30.556         NaN
20230607-17:12:33.057         NaN
20230608-18:22:16.769         NaN
20230609-14:02:28.690       7.117
20230609-14:33:31.199       7.117]
```

返回数据list中DataFrame的个数由行情实际档位数决定，可通过len(result)查看当前行情档位数，通过list取值方法取具体档位，或者使用for循环逐个取出。具体参考代码如下：

```
result = [pandas.DataFrame,pandas.DataFrame,pandas.DataFrame,pandas.DataFrame...]
length = len(result)
bid_price_1 = result[0]  #取bid_price第一档
for i in range(0,length):
  print(result[i])       #遍历取出所有档
```

- 多cic查询，查询域为bid_price具体档位，指定查询返回数据为5条

```
result = comstarlib.sd(cic_code=["USDCNY-CFXO", "200210-CFXO"], category="1",
field="bid_price_1",count=5)
```

返回值为pandas.DataFrame类型，获取USDCNY-CFXO、200210-CFXO 两个券bid_price_1同一时间段对比数据，以transact_time为index，时间不一致默认补NaN，如下所示：

```
                     USDCNY-CFXO 200210-CFXO
transact_time
20230607-17:12:30.556        7.12         NaN
20230607-17:12:33.057        7.12         NaN
20230608-18:22:16.769      7.1222         NaN
20230609-14:02:28.690      7.1182         NaN
20230609-14:33:31.199      7.1182         NaN
20230630-12:00:55.221         NaN     99.9479
20230630-12:00:56.205         NaN      99.926
20230630-12:00:57.678         NaN     99.8946
20230630-12:00:58.135         NaN     99.8568
20230630-12:00:59.159         NaN     99.8096
```

2）查询成交行情，category="3"，限制行情时间范围

```
result = comstarlib.sd(cic_code=["200215.CFXO"], category="3", field="tick_yield",
                                    start_time="2023-06-21 00:12:30.556",
                                    end_time="2023-06-21 14:31:12.360", count=5)
```

返回值为pandas.DataFrame类型如下图所示：

```
                     200215.CFXO
transact_time
20230621-14:30:22.111      2.0088
20230621-14:30:23.111      2.0088
20230621-14:30:24.111      2.0088
20230621-14:30:25.111      2.0088
20230621-14:30:26.111      2.0088
20230621-14:30:44.111      2.0088
20230621-14:31:02.360       2.098
20230621-14:31:12.360       2.098
```

若查询限制查询时间以及查询数量，则只有查询时间生效。

3）查询K线行情，category="3"

聚合200210.CFXO、990002.CFXO的deal行情，inertval是Interval.HOUR即3600s，查询域为volume

```
result = comstarlib.sd(cic_code=["200210.CFXO", "990002.CFXO"], category="3",
field="volume", count=5,
                                   interval=Interval.HOUR)
```

result返回值类型为pandas.DataFrame，获取200210.CFXO、990002.CFXO 两个券volume同一时间段对比数据，以transact_time为index，时间不一致默认补NaN，如下所示：

```
                     200210.CFXO 990002.CFXO
transact_time
20230630-10:00:00.000   900000000           0
20230630-11:00:00.000   460000000           0
20230630-12:00:00.000  1320000000           0
20230630-14:00:00.000           0           0
20230630-15:00:00.000   290000000           0
```

注意：若聚合K线，category为"3"且必须指定interval，为Interval对象

2.3.5 ss: 获取截面数据

- **格式：**

```
def ss(cic_code:List[str],      #cic码
       category:str,            #行情类型
       fields: str = "count",   #查询域
       **kwargs                 #可选参数
             ) -> pandas.DataFrame
```

- **参数说明：**

参数名	类型	说明
cic_code	List[str]	cic 码唯一识别行情,支持多资产查询,例如["210215-CFXO",200210-CFXO"]
category	str	行情类型，category为“1”或者“3”，“1”为quote行情、“3”为deal行情
fields	list	指定查询内容，暂只支持为count，查询某个行情的数量
**kwargs		可选参数，若查询固定时间段内需配置start_time和end_time，默认所有

- **关键字参数kwargs可选域**：

参数名	含义
start_time	行情开始时间，时间格式为"'%Y-%m-%d %H:%M:%S.%f'"
end_time	行情结束时间，时间格式为"'%Y-%m-%d %H:%M:%S.%f'"

- **功能：**

  查询行情的数量。

- **调用频率：**

  可在 init 之外的函数中调用。

- **注意：**

  1）指定cic_code查询必须指定category（用于区分行情类型）。

- **返回值：**

  返回类型为panda.DataFrame,columns=[cic_code,count]

- **示例：**

  查询USDCNY-CFXO、210215-CFXO 在固定时间范围内的行情数量，若行情不存在则为None，如下所示

```
result = comstarlib.ss(["USDCNY-CFXO", "210215-CFXO"], category="1",
start_time="2023-06-07 17:12:30.556",
                               end_time="2023-06-20 17:12:30.556")
```

返回类型为pd.DataFrame，columns=[cic_code,count]，返回结构如下所示：

```
      cic_code  count
0  USDCNY-CFXO      5
1  210215-CFXO   1733
```

2.3.6 edb: 获取键值查询数据

- **格式：**

```
def edb(cic_code:str,                   #cic码
        category:str,                   #行情类型
        fields: List[str] = None,       #查询域
        **kwargs                        #可选参数
             ) -> pandas.DataFrame
```

- **参数说明：**

参数名	类型	说明
cic_code	str	cic 码唯一识别行情,单资产查询
category	str	行情类型，category为“1”或者“3”，“1”为quote行情、“3”为deal行情
fields	list	行情查询域名，可选域参考sd接口说明，支持多域查询，例如["price","volume"] 也可通过dir(comstarlib)查看所有可查询域
**kwargs		可选参数

- **关键字参数kwargs可选域：**

参数名	含义
start_time	行情开始时间，时间格式为"'%Y-%m-%d %H:%M:%S.%f'"
end_time	行情结束时间，时间格式为"'%Y-%m-%d %H:%M:%S.%f'"
interval	K线生成频率，Interval对象，详见参见 Interval 类
count	行情查询数量，默认为5000

- **功能：**

  获取键值查询数据。

- **调用频率：**

  可在 init 之外的函数中调用。

- **注意：**

  1）指定cic_code查询必须指定category（用于区分行情类型）。

- **返回值：**

  返回类型为panda.DataFrame

- **示例：**

  1）查询报价行情，category="1"

  - 查询域为bid_price具体档位以及bid_price所有档位，指定查询返回数据为5条

```
result = comstarlib.edb("USDCNY-CFXO", category="1", fields=["bid_price",
"bid_price_1"], count=5)
```

result 返回的值为pandas.DataFrame，返回结构如下图所示，index为transact_time。

```
                          cic_code        bid_price bid_price_1
transact_time
20230607-17:12:30.556  USDCNY-CFXO           [7.12]        7.12
20230607-17:12:33.057  USDCNY-CFXO           [7.12]        7.12
20230608-18:22:16.769  USDCNY-CFXO         [7.1222]      7.1222
20230609-14:02:28.690  USDCNY-CFXO  [7.1182, 7.117]      7.1182
20230609-14:33:31.199  USDCNY-CFXO  [7.1182, 7.117]      7.1182
```

2）查询成交行情，category="3"，限制行情时间范围

```
result = comstarlib.edb("200210.CFXO", category="3", fields=["price", "volume"],
                                    start_time="2023-06-30 12:00:55.221",
                                    end_time="2023-06-30 14:31:12.360")
```

返回值为pandas.DataFrame类型如下图所示：

```
                          cic_code  price    volume
transact_time
20230630-12:00:59.110  200210.CFXO  100.7  20000000
20230630-12:01:59.360  200210.CFXO  103.0  30000000
20230630-12:02:59.190  200210.CFXO  100.0  20000000
20230630-12:03:59.010  200210.CFXO   99.0  30000000
20230630-12:04:59.110  200210.CFXO  100.1  20000000
20230630-12:05:59.110  200210.CFXO  100.2  20000000
20230630-12:06:59.110  200210.CFXO  100.3  20000000
20230630-12:07:59.110  200210.CFXO  100.4  20000000
20230630-12:08:59.110  200210.CFXO  100.5  20000000
20230630-12:09:59.999  200210.CFXO  100.6  20000000
```

若查询限制查询时间以及查询数量，则只有查询时间生效。

3）查询K线行情，category="3"

聚合200210.CFXO的deal行情，inertval是Interval.HOUR即3600s，查询域为["high", "low"]

```
result = comstarlib.edb(cic_code="200210.CFXO", category="3", fields=["high",
"low"], count=5,
                                    interval=Interval.HOUR)
```

```
                          cic_code   high    low
transact_time
20230630-11:00:00.000  200210.CFXO  103.0   99.0
20230630-12:00:00.000  200210.CFXO  103.0   99.0
20230630-14:00:00.000  200210.CFXO  100.6  100.6
20230630-15:00:00.000  200210.CFXO  103.0   99.0
20230630-16:00:00.000  200210.CFXO  103.0   99.0
```

注意：若聚合K线，category为"3"且必须指定interval，为Interval对象

2.4 交易API

- **用途：**
  发送订单、撤销已挂未成交订单。
- **API总览：**

序号	API名称	说明
1	send_order( )	发单函数
2	cancel_orders( )	批量撤销订单
3	cancel_all_orders( )	撤销全部订单

2.4.1 send_order: 发单函数

- **格式：**

```
def send_order(
        platform_code: str,             #发单平台码
        security_id: str,               #券号
        volume: int,                    #委托量
        price: float,                   #委托价格
        order_type: OrderType,          #订单类型
        side: OrderSide,                #委托方向
        strategy_group='default',       #策略分组
        comments=''                     #备注
              )
```

- **参数说明：**

参数名	类型	说明
platform_code	str	发单平台码
security_id	str	券号
volume	int	发单量：债券面额 100 元的百分之一为 1 个单位。 例如：volume 为 1 千万时，表示 10 万份面额 100 的债券。
price	float	用户填入的发单价格
order_type	OrderType 枚举类	指明订单类型： 1. 目前仅支持 LIMIT 表示限价单
side	OrderSide 枚举类	指明发买单还是卖单 1. OrderSide.BUY 发买单，OrderSide.BUY.value 值为1 2. OrderSide.SELL发卖单，OrderSide.SELL.value 值为-1
strategy_group	str	策略分组：可将同一个策略分成若干组，每组可分别统计头寸、损益 策略分组名 strategy_group 的默认值为 'default'
comments	str	备注，方便核对订单，可在模拟、实盘日志中记录

- **功能：**

  填入交易平台、券号、量、价、限价单、方向、策略分组标记、备注，进行发单。返回 str 类型的 order_id 。

  platform_code：发单平台码，可通过行情设置提供的对象取到，如订阅行情 1，context.cs1.platform。

  order_type：目前仅支持 OrderType.LIMIT 类型。

  side：委托方向，枚举类，买为 OrderSide.BUY，OrderSide.BUY.value 为 1。卖为 OrderSide.SELL，OrderSide.SELL.value 为 -1。

  strategy_group：策略分组名的默认值为'default'，可用于分组统计仓位、损益等。

  comments：自定义备注，可以在发单时自由填写，会在模拟、实盘的日志看到。

- **注意：**

  发单后如果有成交或状态变化，成交与状态信息不会在本 tick 或 bar 中获知，会于所订阅的任意下一行情推送过来的时候获知（回测如此，模拟和实盘则是发生变动后即推，不等行情）。当收到成交或状态变化信息时，on_order 中的代码会被触发。

- **调用频率：**

  可在 init、on_order 之外的函数中调用。因为发单、撤单本身都会触发 on_order 调用，因此不可在 on_order 中发单、撤单。

- **返回值：**

  订单发送成功返回 str 类型的 order_id。订单未能送出返回 0。

- **示例：**

```
order_id = send_order(
               platform_code=context.cs1.platform,
               security_id=context.cs1.symbol,
               volume=10000000,
               price=100.0,
               order_type=OrderType.LIMIT,
               side=OrderSide.SELL,
               strategy_group='default'
                      )
```

结果：订阅的行情 1 的券发单卖出 1000 万。

2.4.2 cancel_orders: 批量撤销订单

- **格式：**

```
def cancel_orders(order_ids: List [str],  #订单号
                  )
```

- **参数说明：**

参数名	类型	说明
order_ids	List[str]	订单号列表

- **功能：**
  根据指定的订单号列表撤销订单。忽略其中的非活跃订单。
- **调用频率：**
  可在 init、on_order 之外的函数中调用。因为发单、撤单本身都会触发 on_order 调用，因此不可在 on_order 中发单、撤单。
- **返回值：**
  无返回值。
- **示例：**

```
cancel_orders(order_ids)
```

撤销 context.cs1.symbol 该券 A 组的全部待成交订单。

2.4.3 cancel_all_orders: 撤销全部订单

- **格式：**

```
def cancel_all_orders()
```

- **功能：**
  撤销全部待成交订单。
- **调用频率：**
  可在 init、on_order 之外的函数中调用。因为发单、撤单本身都会触发 on_order 调用，因此不可在 on_order 中发单、撤单。
- **返回值：**
  无返回值。
- **示例：**

```
cancel_all_orders()
```

  撤销状态为 NEW、PARTIAL的全部待成交订单。
- **参数说明：**
  无。

2.5 交易信息查询API

- **用途：**
  动态获得交易中的有关信息，如仓位、损益、订单等，以方便用户做策略条件判断。
- **API总览：**

序号	API名称	说明
1	get_order_ids( )	订单条件筛选
2	get_order_objs( )	获得order对象列表
3	get_position( )	获得持仓信息（单一分组）
4	get_portfolio( )	获得账户组合信息
5	get_position_pnl( )	单一券分组损益
6	get_portfolio_pnl( )	账户组合损益

2.5.1 get_order_ids: 订单条件筛选

- **格式：**

```
def get_order_ids(
        security_id:str,                    #券号，必填
        strategy_group:str,                 #指定分组，必填
        side:List[OrderSide],               #指定方向
        order_status:List[OrderStatus],     #order状态
        volume_more_equal:float,            #待成交量大于等于，
        volume_less_equal:float,            #待成交量小于等于，
        price_more_equal:float,             #指定委托价大于等于，
        price_less_equal:float,             #指定委托价小于等于，
                )
```

- **参数说明：**

参数名	类型	说明
security_id	str	券号，必填
strategy_group	str	策略分组，必填
order_status	List[OrderStatus]	订单状态列表，默认参数：[OrderStatus.INITIAL, OrderStatus.NEW, OrderStatus.PARTIAL]
side	List[OrderSide]	买卖方向列表， 默认 [OrderSide.BUY, OrderSide.SELL]
volume_more_equal	float	待成交量大于等于， 默认 None，不检查条件
volume_less_equal	float	待成交量小于等于， 默认 None，不检查条件
price_more_equal	float	委托价格大于等于， 默认 None，不检查条件
price_less_equal	float	委托价格小于等于， 默认 None，不检查条件

- **功能：**

  根据指定 order 筛选条件返回 order_id 列表。没有满足条件的返回空列表。如果只填前 2 个参数，则默认把未完成订单都查出，不考虑条件。

- **注意：**

  量使用剩余量 leaves 做筛选，价使用委托价 submit_price 做筛选。

  _more_equal 后缀理解为 >= 号，_less_equal 后缀理解为 <= 号。
  1. volume_more_equal 填 1 千万时，会筛选量 >= 1 千万的订单，volume_less_equal 填 2 千万时，会筛选量 <=2 千万的订单，同时填以上两个条件会选出量 =[1 千万， 2 千万] 区间的订单。
  2. 相反，同时填 volume_less_equal 为 1 千万，volume_more_equal 为 2 千万时，会选出量 (0, 1 千万] 和 [2 千万，+无穷) 的订单。
  3. price 筛选规则与 volume 筛选规则一样。
  4. OrderStatus 支持订单状态筛选。
  5. side 支持买卖方筛选。

- **调用频率：**

  可在 init 之外的函数中调用。

- **返回值：**
  订单 id(str 类型) 的列表

- **示例1：**

```
order_ids = get_order_ids(security_id='200215',
                          strategy_group='A',
                          side=[OrderSide.BUY])
```

  返回 200215 在 A 分组下的全部待成交买单 id。

- **示例2：**

```
order_ids = get_order_ids(security_id='200215',
                          strategy_group='A',
                          side=[OrderSide.BUY],
                          order_status=[OrderStatus.NEW],
                          volume_more_equal=20000000,
                          price_less_equal=99.96)
```

返回 200215 券，下在 A 分组中的，状态为 New，方向为 BUY，待成交量 >=2 千万，委托价 <=99.96 的所有订单 id 的列表。

2.5.2 get_order_objs: 获得 order 对象列表

- **格式：**

```
def get_order_objs(
                   order_ids: List[str]     #order的id
                   )
```

- **参数说明：**

参数名	类型	说明
order_ids	List[str]	订单号列表

- **功能：**
  根据指定 order_ids 列表返回多个 order 对象列表。
- **注意：**
  无。
- **调用频率：**
  可在 init 之外的函数中调用。
- **应用场景举例：**
  用户获取多个 order 对象。
- **返回值：**
  Order 对象列表。用户自行可以取出 list 中的 order 对象，并通过 order.side、order.price 等方式查看该 order 的详情，order 对象的属性见 Order 订单类。
- **示例：**

```
objs = get_order_objs(order_ids)。
```

objs 是 order 对象的 list，每个 order 对象内可见 order 的委托价、剩余量等有关的详细信息。

2.5.3 get_position: 获得持仓信息

- **格式:**

```
def get_position(
                security_id: str,      #券号
                strategy_group: str    #策略分组
                )
```

- **参数说明:**

参数名	类型	说明
security_id	str	券号
strategy_group	str	策略分组

- **功能:**

  根据指定 security_id、strategy_group 返回 position 对象，用于查询某券某分组的持仓信息、持仓损益等信息。

- **调用频率:**
  可在 init 之外的函数中调用。

- **应用场景举例:**
  用户获取 position 对象（以券+分组区分）。

- **返回值:**
  position 对象。无论是否查询到，都会返回 position 的对象，没查到的 position 为该目标券组的初始化状态对象。

- **示例:**
  假设 200215 的 A 组已有持仓空头 1 千万，未实现损益为 1234，已实现损益为 5678。

```
pos = get_position(security_id='200215',
                   strategy_group='A')
```

  pos.side 为 OrderSide.SELL, pos.side.value 为 -1,
  pos.hold_volume 为 10000000,

2.5.4 get_portfolio: 获得账户组合信息

- **格式:**

```
def get_portfolio()
```

- **功能:**
  返回 portfolio 对象，portfolio 记录了该策略交易过的券和所属分组有哪些。

- **注意：**
  无。
- **调用频率：**
  可在 init 之外的函数中调用。
- **应用场景举例：**
  希望依次查询不同券+分组的头寸之前，调用该函数可得到所有券+分组信息。
- **返回值：**
  portfolio 对象。
- **示例：**

```
portfolio = get_portfolio()
```

  返回值为

```
{'200215':['A','B'],'200210':['A']}
```

  即可知道当前策略对 200215 券在 A、B 分组下过单可能有头寸，200210 在 A 分组可能有头寸，可进一步通过 get_position 函数分别获取各个持仓。
- **参数说明：**
  无。

2.5.5 get_position_pnl: 单一券分组损益

- **格式：**

```
def get_position_pnl(
                    security_id:str,    #券号
                    strategy_group:str  #策略分组
                    )
```

- **参数说明：**

参数名	类型	说明
security_id	str	券号
strategy_group	str	策略分组

- **功能：**
  根据指定 security_id、strategy_group 确定一个 position 对象记录某券某分组的总损益、已实现损益、未实现损益。
- **注意：**
  无。

- **调用频率：**
  可在 init 之外的函数中调用。
- **返回值：**
  列表：[total_pnl, realized_pnl, unrealized_pnl],
  total_pnl 为该 position 的总损益、realized_pnl 为该 position 的已实现损益、unrealized_pnl 为该 position 的未实现损益。
- **应用场景举例：**
  用户想知道 200215，分组 A 的持仓损益。
- **示例：**

```
pos_pnl = get_position_pnl(security_id='200215',
                           strategy_group='A')
```

  其中 pos_pnl[0] 为总损益，pos_pnl[1] 为已实现损益，pos_pnl[2] 为未实现损益。

2.5.6 get_portfolio_pnl: 账户组合损益

- **格式：**

```
def get_portfolio_pnl()
```

- **功能：**
  返回 portfolio 中所有 position 的当前持仓总损益、已实现、未实现损益。
- **注意：**
  无。
- **调用频率：**
  可在 init 之外的函数中调用。
- **应用场景举例：**
  用户希望知道当前账户的持仓总损益。
- **返回值：**
  列表：[total_pnl, realized_pnl, unrealized_pnl],
  total_pnl 为账户总损益、realized_pnl 为账户已实现损益、unrealized_pnl 为账户未实现损益。
- **示例：**

```
portfolio_pnl = get_portfolio_pnl()
```

  其中 portfolio_pnl[0] 为 total_pnl, portfolio_pnl[1] 为 realized_pnl, portfolio_pnl[2] 为 unrealized_pnl。
- **参数说明：**
  无。

2.6 工具 API

- **用途：**
  方便用户处理数据的工具函数。
- **总览：**

序号	名称	说明
1	cal_deltatime	时间计算
2	order_deal	订单事件分发装饰器
3	format_data	格式化数据

2.6.1 cal_deltatime: 时间计算

- **格式：**

```
def cal_deltatime(
                  time_str: str,       #时间格式'2021-01-01 09:20:30.000'
                  delta_hour: int,     #加减小时数
                  delta_minute: int,   #加减分钟数
                  delta_second: float, #加减秒数
                 )
```

- **参数说明：**

参数名	类型	说明
time_str	str	str 时间，‘2021-01-01 09: 20: 30.000'
delta_hour: int	int	加减小时数，默认值：0
delta_minute: int	int	加减分钟数，默认值：0
delta_second: float	float	加减秒数，默认值：0

- **功能：**

  方便用户直接将系统中涉及到的时间字符串做时间加、减处理，得到新的时间。返回字典类型的时间。delta_hour 表示 +- 的小时数，delta_minute、delta_second 同理。
- **返回值：**

  返回值字典的 key 为 'time_int'，'hour'，'minute'，'second'，'time_str'。
- **注意：**

  delta_hour、delta_minute、delta_second 可正负可零。返回字典中 time_int 为整型毫秒数。
- **调用频率：**

任意地方可调用。

- **应用场景举例：**

  假设当前某个时间字符串 time1 = '2021-01-01 09: 03: 30.000'

  用户想要 time1 时间之前 1 小时 5 分钟的时间是多少。

```
cur = cal_deltatime(transact_time, hour=-1, minute=-5, second=0)
```

  返回值为

```
{'time_int': 1609464030000, 'hour': 7, 'minute': 58, 'second': 30.0, 'time_str':
'2021-01-01 08:20:30.000'}
```

2.6.2 order_deal: 订单事件分发装饰器

- **格式：**

```
@order_deal
def on_order(context,order):
    pass
```

- **功能：**

  实现订单的事件分发。若使用装饰器，则被装饰函数不执行即使用装饰器之后on_order内部的代码逻辑不执行。根据订单状态分别执行下列的函数，下方列出订单状态与函数之间的对应关系：

订单状态	函数
OrderStatus.NEW	on_new_order()
OrderStatus.FINISHED	on_all_traded()
OrderStatus.PARTIAL	on_part_traded()
OrderStatus.CANCELED	on_canceled()
OrderStatus.FAILED	on_failed()

- **分发函数：**

```
def on_new_order(context, order):
    """ 新订单处理方式 """
    pass

def on_all_traded(context, order):
    """ 全部成交处理方式 """
    pass
```

```
def on_part_traded(context, order):
    """ 部分成交处理方式 """
    pass

def on_canceled(context, order):
    """ 撤单成功处理方式 """
    pass

def on_failed(context, order):
    """ 下单失败处理方式 """
    pass
```

- **应用场景：**

  在on_order上添加装饰器。

2.6.3 format_data: 格式化数据

- **格式：**

```
def format_data(
        data_1dlist: 一维列表,  #可能含 None 的数据列表
        method: 填充方式        #'ffill'、'bfill'、'fill'、'zero'、'nan'
                )
```

- **解释：**

  用于方便用户使用统一的逻辑填充可能存在的 None。

- **函数逻辑：**

  填充方式：

  'ffill' 为前值填充、'bfill' 为后值填充、'fill' 为全填充（先前值填充再后值填充）、'zero' 为 0 填充、'nan' 用 numpy. nan 填充。

- **注意：**

  无。

- **调用频率：**

  可在 init 之外的函数中调用，一般跟在获取行情 API 后面使用。

- **返回值：**

  numpy 的数组类型。

- **示例：**

```
his_data = get_deal_n(cic_code='190215.CFXO',
                      count=5,        #最近 5 个 deal 行情（含当期）
                      fields=['time', 'volume', 'price'])
data1 = his_data['price']
data2 = format_data(data_1dlist=data1, method='ffill')
```

假设 data1 为 [99, None, 99.1, None, 98]，则 data2 为 [99, 99, 99.1, 99.1, 98]

- **参数说明：**

参数名	类型	说明
data_1dlist	一维列表	任意数值型的一维列表都可以用这个做填充。
method	str	填充方式： 'ffill'：前值填充、 'bfill'：后值填充、 'fill'：全填充（先前值填充再后值填充）、 'zero'：0 填充、 'nan'：numpy. nan 填充。

2.7 通用数据类

- **用途：**
  系统中使用的向客户开放的类。
- **总览：**

序号	名称	说明
1	TickData	逐笔成交类
2	QuoteData	盘口行情类
3	BarData	Bar 行情信息类
4	OrderData	订单类
5	OrderStatus	订单状态 枚举类
6	OrderType	订单类型 枚举类
7	OrderSide	交易方向 枚举类
8	Position	仓位头寸类
9	Portfolio	账户组合类
10	Interval	行情频率 枚举类

2.7.1 TickData: 逐笔成交类

- **功能：**
  逐笔成交行情属性。
  逐笔成交行情：date_type='Tick' 时表示行情为逐笔成交行情 。
- **更新频率：**
  无。
- **应有场景举例：**
  无。
- **参数说明：**

属性名	类型	说明
cic_code	str	行情 cic 码，如 '200210. CFXO'。
category	str	category == 'quote'，表示盘口行情。
interval	Interval	详见 Interval 类。
transact_time	str	含义：行情时间 '2021-01-01 09: 20: 30.000'。
price	float	逐笔成交价格。
volume	float	逐笔成交量。
tick_yield	float	逐笔成交收益率。

2.7.2 QuoteData: 盘口行情类

- **功能：**
  盘口行情属性。
  盘口行情：date_type='QUOTE' 时表示行情为盘口行情，每个盘口行情默认为 10 档，第 0 档行情为公有行情、其余 9 档私有行情，档不足 10 档的会用 None 填充。比如 [99, 98, 97, None, None, None, None, None, None,None]。

  盘口行情继承了TickData的属性
- **更新频率：**
  无。
- **应有场景举例：**
  无。
- **参数说明：**

属性名	类型	说明
cic_code	str	行情 cic 码，如 '200210. CFXO'。
category	str	category == 'quote'，表示盘口行情。
interval	Interval	详见 Interval 类。
transact_time	str	含义：行情时间 '2021-01-01 09: 20: 30.000'。
price	float	逐笔成交价格。
volume	float	逐笔成交量。
tick_yield	float	逐笔成交收益率。
bid_price	list	盘口 Bid 档位价格。
offer_price	list	盘口 Offer 档位价格。
bid_volume	list	盘口 Bid 档位行情。
offer_volume	list	盘口 Offer 档位行情。
bid_yield	list	盘口 Bid 收益率。
offer_yield	list	盘口 Offer 收益率。

2.7.3 BarData: Bar 行情信息类

- **功能：**
  Bar 行情数据类。
- **结构：**

属性名	类型	说明
cic_code	str	行情 cic 码，如'200210. CFXO'。
category	str	category == 'deal'。
transact_time	str	含义：行情时间 '2021-01-01 09: 20: 30.000'。
open	float	开盘净价。
high	float	最高净价。
low	float	最低净价。
close	float	收盘净价。
open_yield	float	开盘收益率。
high_yield	float	收益率的最大值。
low_yield	float	收益率的最小值。
close_yield	float	收盘收益率。
volume	float	成交量。
interval	Interval 枚举类	Bar 频率, 详见 Interval 类。
weighting_yield	float	加权收益率
weighting_price	float	加权价

2.7.4 OrderData: 订单类

- **功能:**

  订单数据类，包含订单有关内容。作为 on_order 函数的参数传入，在 on_order 中可以获知发生变化的 order 对象的最新值。

  订单状态变化，成交发生等都会修改 order 类的对象实例。

- **更新频率:**
  发单、撤单、成交、订单状态变化。
- **结构:**

属性名	类型	说明
security_id	str	行情代码，如 190215、200210 等。
platform_code	str	发单平台码
order_id	str	订单 id，下单成功后会返回该值。
price	str	价格
side	OrderSide 枚举类	订单方向：OrderSide. BUY，OrderSide. SELL OrderSide. BUY. value 为 1，表示买入， OrderSide. SELL. value 为 -1，表示卖出。
submit_price	float	成交价。
order_status	OrderStatus 枚举类	参见 OrderStatus 类。
order_type	OrderType 枚举类	订单类型：详见 OrderType 1：OrderType. LIMIT 限价订单。
submit_volume	float	订单委托量。
strategy_group	str	策略分组。
leaves	float	剩余交易量
comments	str	send_order 时的备注。
order_change_time	str	订单修改时间
order_first_time	str	订单时间
last	str	最新交易量
is_trade	str	是否交易，默认值为false

2.7.5 OrderStatus: 订单状态 枚举类

- **功能：**
  用于标识订单状态。可帮助识别 on_order 的触发原因。
- **更新频率：**
  每当订单状态发生变化时修改 order 对象中的 order_status。
- **应用场景举例：**
  无。
- **参数说明：**

属性名	枚举类属性的 value	说明
INITIAL	OrderStatus. INITIAL. value 等于 0	初始化
NEW	OrderStatus. NEW. value 等于 1	新订单
PARTIAL	OrderStatus. PARTIAL. value 等于 2	部分成交
CANCELED	OrderStatus. CANCELED. value 等于 11	已撤销
FINISHED	OrderStatus. FINISHED. value 等于 12	全部成交
FAILED	OrderStatus. FAILED. value 等于 13	发单失败

2.7.6 OrderType: 订单类型 枚举类

- **功能:**
  用于标识订单类型，目前仅支持限价单。发单函数需要填此字段。为枚举类。
- **更新频率:**
  用户发单时设置。
- **示例:**

```
order_id = send_order(..., order_type = OrderType.LIMIT,...)
```

- **参数说明:**

属性名	说明
OrderType. LIMIT	OrderType. LIMIT. value 等于“限价单”。

2.7.7 OrderSide: 交易方向 枚举类

- **功能:**
  枚举类，表示交易方向。
- **使用场景举例:**

```
target_side = OrderSide(int(numpy.sign(a-b)))
target_volume = target_side.value*10000000  #直接根据a-b的结果计算发单量
if target_side == OrderSide.SELL:
    send_order(..., volume=target_volume, side=target_side,...)
```

- **发单 side 参数填写示例:**
  发买单：send_order(..., side = OrderSide.BUY, ...)
- **结构:**

属性名	说明
BUY	OrderSide. BUY. value 等于 1，买
SELL	OrderSide. SELL. value 等于 -1，卖

2.7.8 Position: 仓位头寸类

- **功能：**

  Position 类主要用于记录特定券种特定分组的头寸、已实现损益、未实现损益有关信息。即该策略的每个券的每个分组建立一个 Position。

- **更新频率：**
  第一次对某券某分组发单时，创建一个 Position 类的对象，每当该券该分组有成交发生时，该 Position 类的对象被更新。在进入 on_order 中时已更新。

- **应用场景举例：**
  某用户对 200215 发过 2 次单，1 次 strategy_group 是 'A'，另 1 次是 'B'，则系统中维护 2 个 Position 类的对象。

- **结构：**

属性名	类型	说明
security_id	str	券号
strategy_group	str	策略分组
side	OrderSide	当前持仓方向
hold_volume	float	持仓量
open_close_flag	bool	是否有开平仓
accum_realized_pnl	float	累计已实现损益
unrealized_pnl	float	未实现损益
current_side	OrderSide	当日方向
current_hold_volume	flost	当日净敞口量
previous_side	OrderSide	非当日方向
previous_hold_volumn	flost	非当日净敞口量

2.7.9 Portfolio: 账户组合类

- **功能：**
  每个策略 1 个。目前只记录账户有哪些 Position。通过 positions 字段可以知道当前策略有哪些 Position 类的对象在维护。可方便通过 get_position 取到想查看的持仓对象。
- **结构：**

属性名	类型	说明
positions	dict	账户关联的 Position。 例如：{'200215': ['A', 'B'], '200210': ['A']}

2.7.10 Interval: 行情频率 枚举类

- **功能：**
  用于指定行情频率的枚举类，可通过点标记 ('. ') 获取相关属性。
- **示例：**
  在需要填频率的参数时，参考以下方式，表示频率为 1 分钟的 bar

```
get_bar_n(..., interval = Interval.ONE_MINUTE, ...)
```

- **参数说明：**

属性名	说明
ONE_MINUTE	Interval. ONE_MINUTE 作用：指定行情频率为 1 分钟 Bar Interval. ONE_MINUTE. value 等于 '1'
FIVE_MINUTE	Interval. FIVE_MINUTE 作用：指定行情频率为 5 分钟 Bar Interval. FIVE_MINUTE. value 等于 '5'
TEN_MINUTE	Interval. TEN_MINUTE 作用：指定行情频率为 10 分钟 Bar Interval. TEN_MINUTE. value 等于 '10'
HALF	Interval. HALF 作用：指定行情频率为 30 分钟 Bar Interval. HALF. value 等于 '30'
HOUR	Interval. HOUR 作用：指定行情频率为 1 小时 Bar Interval. HOUR. value 等于 '60'
DAILY	Interval. DAILY 作用：指定行情频率为 1 日 Bar Interval. DAILY. value 等于 '1D'
TICK	Interval. TICK 作用：指定行情频率为逐笔成交行情、盘口行情 Interval. TICK. value 等于 '0'
FIVE_SECOND	nterval. FIVE_SECOND 作用：指定行情频率为 5秒 Bar Interval. FIVE_SECOND.value 等于 '5s'
TEN_SECOND	Interval. TEN_SECOND 作用：指定行情频率为10秒 Bar Interval. TEN_SECOND.value 等于 '10s'
FIFTEEN_SECOND	Interval. FIFTEEN_SECOND 作用：指定行情频率为 15秒 Bar Interval. FIFTEEN_SECOND.value 等于 '15s'
THIRTY_SECOND	Interval. THIRTY_SECOND 作用：指定行情频率为 30秒 Bar Interval. THIRTY_SECOND.value 等于 '30s'

最新变动

日期	版本	描述
2023-04-23	V1.1.1	1. 新增format_data接口 2. 添加定时器函数add_timer 3. 策略模板修改，新增on_timer定时器触发推送。 4. 指定get_position返回结果类型为Position对象 5. 新增获取时序数据接口sd 6.新增获取截面数据接口ss
2023-06-30	V1.2.0	1.新增edb键值查询接口 2.时序查询接口sd入参类型调整，补充示例代码 3.Interval对象新增秒级频率