**Tick级精细委托管理**

原文作者：上弦之月

随着量化交易在国内金融市场越来越普及，CTA策略之间的竞争也变得越发激烈，除了体现在策略的核心交易信号方面外，也同样体现在策略的实盘委托执行中。大部分vn.py官方提供的CTA策略样例中，在K线回调函数on\_bar内采用的都是两步操作：

1.调用cancel\_all函数，全撤之前上一根K线已经挂出的委托。

2.检查核心交易信号，发出新一轮的委托挂单。

这种简单粗暴的写法，更多是出于简化策略执行中的状态机控制，帮助vn.py初学者的降低学习难度，但由于较多的重复操作，在实盘中的运行效果未必能达到最佳。

好在策略模板CtaTemplate中委托函数（buy/sell/short/cover）可以直接返回委托号信息，以及on\_order/on\_trade回调函数会推送委托和成交状态变化，结合少量的底层代码改造，我们就可以实现更加精细的Tick级别委托挂撤单管理，让策略的核心交易信号和委托执行算法更加有机地结合起来。

一、扩展OrderData对象

找到vn.py源代码所在的路径，使用VN Studio的情况下，应该位于C:\vnstudio\Lib\site-packages\vnpy，进入到目录vnpy\trader下找到object.py 文件

首先需要对OrderData类进行扩展，主要表现在：

1.类的初始化除了time属性外，还增加了date和cancel\_time属性。

2.在\_\_post\_init\_\_函数，增加了未成交量untraded和委托的具体日期时间datetime。

其中应该注意的是，由于每个行情API接口推送的时间格式不太相同，所以基于：

1.date格式是"%Y-%m-%d"还是"%Y%m%d";

2.time有无微秒级别数据;

要用到4种不同的处理方法得到datetime。

@dataclass

class OrderData(BaseData):

""" Order data contains information for tracking lastest status of a specific order."""  
 symbol: str exchange: Exchange

orderid: str  
 type: OrderType = OrderType.LIMIT

direction: Direction = ""

offset: Offset = Offset.NONE

price: float = 0

volume: float = 0

traded: float = 0

status: Status = Status.SUBMITTING  
 date: str = ""

time: str = ""

cancel\_time: str = ""

def \_\_post\_init\_\_(self):

""""""

self.vt\_symbol = f"{self.symbol}.{self.exchange.value}"

self.vt\_orderid = f"{self.gateway\_name}.{self.orderid}"  
 self.untraded = self.volume - self.traded  
 # With millisecond

if self.date and "." in self.time:

if "-"in self.date:

self.datetime=

datetime.strptime(" ".join([self.date, self.time]), "%Y-%m-%d %H:%M:%S.%f") else:

self.datetime=datetime.strptime("".join([self.date,self.time]),"%Y%m%d %H:%M:%S.%f")

# Without millisecond

elif self.date:

if "-" in self.date: self.datetime = datetime.strptime(" ".join([self.date, self.time]), "%Y-%m-%d %H:%M:%S") else: self.datetime = datetime.strptime(" ".join([self.date, self.time]), "%Y%m%d %H:%M:%S")  
  
 def is\_active(self): """ Check if the order is active. """ if self.status in ACTIVE\_STATUSES: return True else: return False  
 def create\_cancel\_request(self): """ Create cancel request object from order. """ req = CancelRequest( orderid=self.orderid, symbol=self.symbol, exchange=self.exchange ) return req

扩展持仓明细查询

修改PositionHolding对象

进入目录vnpy\trader打开converter.py文件，这一次我们对PositionHolding类进行修改，主要改动如下：

\_\_init\_\_类的初始化，新加入long\_pnl，long\_price，short\_pnl，short\_price这4个属性。即增加了持仓盈亏以及开仓均价；

update\_position函数新增缓存持仓盈亏和开仓均价；

update\_order函数首先修改了活动委托的定义，把【委托提交中】剔除，即在update\_order函数只处理未成交或者部分成交状态的委托。

class PositionHolding: """"""  
 def \_\_init\_\_(self, contract: ContractData):        """""" self.vt\_symbol = contract.vt\_symbol self.exchange = contract.exchange  
        self.active\_orders = {}         self.long\_pos = 0 self.long\_pnl = 0 self.long\_price = 0 self.long\_yd = 0 self.long\_td = 0  
 self.short\_pos = 0 self.short\_pnl = 0 self.short\_price = 0 self.short\_yd = 0 self.short\_td = 0  
 self.long\_pos\_frozen = 0 self.long\_yd\_frozen = 0 self.long\_td\_frozen = 0  
 self.short\_pos\_frozen = 0 self.short\_yd\_frozen = 0 self.short\_td\_frozen = 0  
 def update\_position(self, position: PositionData): """""" if position.direction == Direction.LONG: self.long\_pos = position.volume self.long\_pnl = position.pnl self.long\_price = position.price self.long\_yd = position.yd\_volume self.long\_td = self.long\_pos - self.long\_yd self.long\_pos\_frozen = position.frozen else: self.short\_pos = position.volume self.short\_pnl = position.pnl self.short\_price = position.price self.short\_yd = position.yd\_volume self.short\_td = self.short\_pos - self.short\_yd self.short\_pos\_frozen = position.frozen  
 def update\_order(self, order: OrderData): """""" #active\_orders只记录未成交和部分成交委托单 if order.status in [Status.NOTTRADED, Status.PARTTRADED]: self.active\_orders[order.vt\_orderid] = order else: if order.vt\_orderid in self.active\_orders: self.active\_orders.pop(order.vt\_orderid)  
 self.calculate\_frozen() ......

新增get\_position\_detail函数

进入目录vnpy\app\cta\_strategy打开engine.py文件，这一次我们要新增一个函数get\_position\_detail。该函数功能就是获取我们修改后的PositionHolding对象，从而知道更加详细的持仓信息，如开仓均价，持仓盈亏等：

from collections import defaultdict,OrderedDict......  
 def get\_position\_detail(self, vt\_symbol): """ 查询long\_pos,short\_pos(持仓)，long\_pnl,short\_pnl(盈亏),active\_order(未成交字典) 收到PositionHolding类数据        """ try: return self.offset\_converter.get\_position\_holding(vt\_symbol) except: self.write\_log(f"当前获取持仓信息为：{self.offset\_converter.get\_position\_holding(vt\_symbol)},等待获取持仓信息") position\_detail = OrderedDict() position\_detail.active\_orders = {} position\_detail.long\_pos = 0 position\_detail.long\_pnl = 0 position\_detail.long\_yd = 0 position\_detail.long\_td = 0 position\_detail.long\_pos\_frozen = 0 position\_detail.long\_price = 0 position\_detail.short\_pos = 0 position\_detail.short\_pnl = 0 position\_detail.short\_yd = 0 position\_detail.short\_td = 0 position\_detail.short\_price = 0 position\_detail.short\_pos\_frozen = 0 return position\_detail

然后，为了让交易策略能够直接从引擎调用get\_position\_detail函数，对CTA策略模板也得增加一个调用函数。在同一目录找到template.py文件，开打后在CtaTemplate类中加入以下代码即可：

def get\_position\_detail(self, vt\_symbol: str):        """""" return self.cta\_engine.get\_position\_detail(vt\_symbol)

修改CTA策略代码

底层的功能都添加完毕了，那么现在轮到对具体交易策略逻辑进行改动，从而实现基于实时tick行情的追单和撤单。

添加策略运行时变量

包括委托状态的触发控制器、具体委托量以及拆单间隔：

def \_\_init\_\_(self, cta\_engine, strategy\_name, vt\_symbol, setting): """""" super().\_\_init\_\_(cta\_engine, strategy\_name, vt\_symbol, setting)                #状态控制初始化 self.chase\_long\_trigger = False self.chase\_sell\_trigger = False self.chase\_short\_trigger = False self.chase\_cover\_trigger = False self.long\_trade\_volume = 0 self.short\_trade\_volume = 0 self.sell\_trade\_volume = 0 self.cover\_trade\_volume = 0         self.chase\_interval = 10    #拆单间隔:秒......

修改on\_tick函数逻辑

需要在on\_tick函数增加的对委托挂撤单管理逻辑如下：

调用cta策略模板新增的get\_position\_detail函数，通过engine获取活动委托字典active\_orders。注意的是，该字典只缓存未成交或者部分成交的委托，其中key是字符串格式的vt\_orderid，value对应OrderData对象；

engine取到的活动委托为空，表示委托已完成，即order\_finished=True；否则，表示委托还未完成，即order\_finished=False，需要进行精细度管理；

精细管理的第一步是先处理最老的活动委托，先获取委托号vt\_orderid和OrderData对象，然后对OrderData对象的开平仓属性（即offst）判断是进行开仓追单还是平仓追单：

开仓追单情况下，先得到未成交量order.untraded，若当前委托超过10秒还未成交（chase\_interval = 10)，并且没有触发追单（chase\_long\_trigger = False），先把该委托撤销掉，然后把触发追单器启动；

平仓追单情况下，同样先得到未成交量，若委托超时还未成交并且平仓触发器没有启动，先撤单，然后启动平仓触发器；

当所有未成交委托处理完毕后，活动委托字典将清空，此时order\_finished状态从False变成True，用最新的买卖一档行情，该追单开仓的追单开仓，该追单平仓的赶紧平仓，每次操作后恢复委托触发器的初始状态：

def on\_tick(self, tick: TickData): """""" active\_orders = self.get\_position\_detail(tick.vt\_symbol).active\_orders  
 if active\_orders: #委托完成状态 order\_finished = False vt\_orderid = list(active\_orders.keys())[0] #委托单vt\_orderid order = list(active\_orders.values())[0] #委托单字典  
 #开仓追单，部分交易没有平仓指令(Offset.NONE) if order.offset in (Offset.NONE, Offset.OPEN): if order.direction == Direction.LONG: self.long\_trade\_volume = order.untraded if (tick.datetime - order.datetime).seconds > self.chase\_interval and self.long\_trade\_volume > 0 and (not self.chase\_long\_trigger) and vt\_orderid: #撤销之前发出的未成交订单 self.cancel\_order(vt\_orderid) self.chase\_long\_trigger = True elif order.direction == Direction.SHORT: self.short\_trade\_volume = order.untraded if (tick.datetime - order.datetime).seconds > self.chase\_interval and self.short\_trade\_volume > 0 and (not self.chase\_short\_trigger) and vt\_orderid: self.cancel\_order(vt\_orderid) self.chase\_short\_trigger = True #平仓追单 elif order.offset in (Offset.CLOSE, Offset.CLOSETODAY): if order.direction == Direction.SHORT: self.sell\_trade\_volume = order.untraded if (tick.datetime - order.datetime).seconds > self.chase\_interval and self.sell\_trade\_volume > 0 and (not self.chase\_sell\_trigger) and vt\_orderid: self.cancel\_order(vt\_orderid) self.chase\_sell\_trigger = True if order.direction == Direction.LONG: self.cover\_trade\_volume = order.untraded if (tick.datetime - order.datetime).seconds > self.chase\_interval and self.cover\_trade\_volume > 0 and (not self.chase\_cover\_trigger) and vt\_orderid: self.cancel\_order(vt\_orderid) self.chase\_cover\_trigger = True else: order\_finished = True if self.chase\_long\_trigger and order\_finished: self.buy(tick.ask\_price\_1, self.long\_trade\_volume) self.chase\_long\_trigger = False elif self.chase\_short\_trigger and order\_finished: self.short(tick.bid\_price\_1, self.short\_trade\_volume) self.chase\_short\_trigger = False elif self.chase\_sell\_trigger and order\_finished: self.sell(tick.bid\_price\_1, self.sell\_trade\_volume) self.chase\_sell\_trigger = False elif self.chase\_cover\_trigger and order\_finished: self.cover(tick.ask\_price\_1, self.cover\_trade\_volume) self.chase\_cover\_trigger = False

最后需要注意的是，Tick级精细挂撤单的管理逻辑，无法通过K线来进行回测检验，因此通过仿真交易（比如期货基于SimNow）进行充分的测试就是重中之重了。