3.3 一个简单的高频交易策略

策略思路：

每个 TICK 都会检查持仓状态，间隔一定时间就可以发送委托。

当前时刻可能形成的最大多头持仓如果超出限制，则交易方向为空头；

当前时刻可能形成的最大空头持仓如果超出限制，则交易方向为多头；

关注点：

在双均线的案例当中我们熟悉了在 PYTHON 中如何利用 TBQuant 的事件驱动机制进行交易。在本案例和上一个案例的区别其实不是很大。主要体现在双均线交易是放在 onbar 事件中，而本策略是放在 ontick 事件中进行。

事件	事件的功能
on_init 事件：	订阅 TICK、订阅 BAR 数据、订阅账户、设置定时器（用于委托超时的撤单）
on_bar 事件：	
on_tick 事件：	接收实时 tick， 读取目前的持仓，根据规则发送委托
on_position 事件：	获取最新持仓，用于交易委托的类型判断
on_order 事件：	提出已经处于完成状态的委托单编号，只留下未完成的委托单
on_fill 事件：	打印成交
on_timer 事件：	定时检查委托超时进行撤单
push_order_id：	记录委托单的发送时间

完整代码如下：

```
# encoding utf-8
import tbpy
import sys
import datetime


# 自定义策略，须继承 tbpy.IStrategy
class MyStrategy(tbpy.IStrategy):
    #策略的初始化：传入最大持仓、撤单时间、账户 ID、合约代码
    def __init__(self, max_pos, interval_secs, account_id, symbol):
        super().__init__('MyStrategy')
        self._side = 0
        self._max_pos = max_pos
        self._interval_secs = interval_secs
        self._account_id = account_id
        self._symbol = symbol
        self._account = None
        self._pos = None
        self._order_dict = {}
        self._time= None
```

```
        pass

    def __del__(self):
        pass

    # 初始化事件函数：订阅 TICK、订阅账户、设置定时器、读取初始仓位
    def on_init(self, context):
        ret = context.subscribe_tick(symbol=self._symbol)
        if ret is not None:
            print(ret)
            tbpy.exit()
        self._account = context.subscribe_account(account_id=self._account_id)
        if self._account is None:
            print(tbpy.get_last_err())
            tbpy.exit()
        context.create_timer(interval_millsecs=2*100)
        self._pos = self._account.get_position(symbol=self._symbol)
        print('on_init success.')

    # BAR 事件函数
    def on_bar(self, context, bars, symbol, flag):
        pass

    # TICK 事件函数：读取目前的持仓，根据规则发送委托。
    def on_tick(self, context, tick):
        if self._time is not None and (datetime.datetime.now() -
self._time).seconds <= self._interval_secs:
            return
        l_up_pos = self._pos.l_current_volume + self._pos.l_active_volume -
self._pos.l_active_close_volume
        s_up_pos = self._pos.s_current_volume + self._pos.s_active_volume -
self._pos.s_active_close_volume
        if l_up_pos >self._max_pos:
            self._side = -1
        if s_up_pos >self._max_pos:
            self._side = 1

        if self._side > 0:
            if s_up_pos == 0:
                self.push_order_id(self._account.buy(symbol=tick.symbol,
volume=1, price=tick.last))
            elif self._pos.s_current_volume > 0:
                self.push_order_id(self._account.buy2cover(symbol=tick.symbol,
volume=1, price=tick.last))
```

```
        else:
            if l_up_pos == 0:
                self.push_order_id(self._account.sell2short(symbol=tick.symbol,
volume=1, price=tick.last))
            elif self._pos.l_current_volume > 0:
                self.push_order_id(self._account.sell(symbol=tick.symbol,
volume=1, price=tick.last))
        self._time = datetime.datetime.now()

    # 持仓事件函数；读取最新持仓
    def on_position(self, context, pos):
        print(pos)
        self._pos = pos

    # 委托事件函数：剔除已经处于完成状态的委托单编号，只留下未完成的委托单
    def on_order(self, context, order):
        print(order)
        if order.status == tbpy.OrderStatus.NewReject or order.status ==
tbpy.OrderStatus.AllFill or \
                        order.status == tbpy.OrderStatus.Canceled or
order.status == tbpy.OrderStatus.CanceledFill:
            self._order_dict.pop(order.order_id)

    # 成交事件函数：打印成交
    def on_fill(self, context, fill):
        print(fill)
        pass

    # 定时器事件函数：委托超时撤单
    def on_timer(self, context, id, millsecs):
        now_time = datetime.datetime.now()
        for key, value in self._order_dict.items():
            if (now_time - value).seconds >= self._interval_secs:
                self._account.cancel_order(order_id=key)
    #push_order_id：记录委托单的发送时间
    def push_order_id(self, order_id_list):
        send_time = datetime.datetime.now()
        for id in order_id_list:
            self._order_dict[id] = send_time

if __name__ == '__main__':
    # TBPY 模块初始化
    ret = tbpy.init()
    if ret is False:
```

```
        print('init fail.')
        sys.exit()

rb_main = tbpy.get_main_instrument(underlying_symbol='rb.SHFE')
if rb_main is None:
        sys.exit()
# 声明用户策略对象
strategy = MyStrategy(5, 5, 'tbyihao2', rb_main.symbol)
# 进入 tbpy 事件循环
tbpy.exe()
```

运行结果