3.2 双均线交易

策略的思路就是：

短周期均线上穿长周期均线，平掉空头仓位，开出多头仓位；

短周期均线下破长周期均线，平掉多头仓位，开出空头仓位。

通过这个策略我们可以了解下如何在 python 里面结合 TBQuant 的运行机制写策略。主要关注点：

1）策略的类如何定义和启动

策略的类的定义需要定义类的初始化和类的各种事件；

策略的类的启动则直接用 tbpy.exe()指令启动。

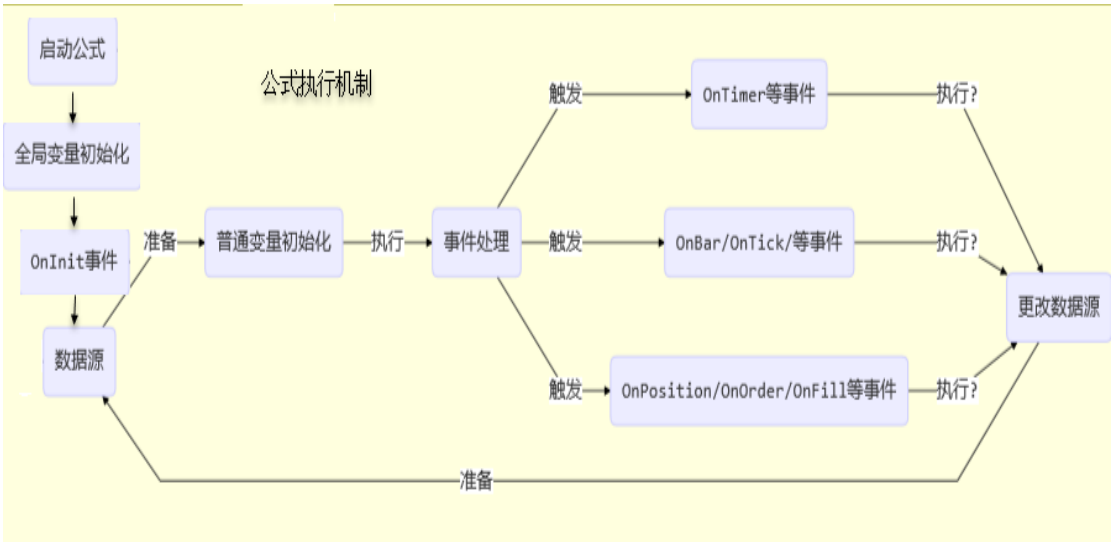
2）各种事件的主要完成什么功能

事件驱动的事件功能简介可以参照学习资料《事件驱动的案例讲解》。我们这里只对本策略用到的事件做出介绍。

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：	计算均线指标、根据规则发送委托单
on_tick 事件：	接收实时 tick,用于交易委托的委托价确定
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

3）事件驱动的运行机制是怎么样？

这里只给出事件驱动的示意图，关于事件驱动的细节可以查看学习资料《事件驱动的案例讲解》。



4）self 和 context 的使用

这两个参数在每个事件驱动中都是默认要用的。所以如果对类不熟悉的化，需要简单了解下 self 和 context 的意义和用法。
self 是在类的方法中的，在调用此方法时，不用给 self 赋值，Python 会自动给他赋值，而且这个值就是类的实例--对象本身。
Context 是上下文，可以理解为成一个环境参数的结合,在当前环境下你能拿到的参数都可以从 context 出发去拿环境。
比如在代码中，在初始化时定义了 context 的 K 线，TICK，账户信息，定时器等。在其它事件驱动中都可以读取到这些信息。

5）关于交易账户的处理

交易账户是在 TBQuant 里面登录完成，在 PYTHON 的策略参数当中制定账户 ID，就可以使用策略读取账户相关信息并发送委托指令，驱动 TBQuant 与交易所进行通讯。

完整代码如下：

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


class My_DualMA(tbpy.IStrategy):
    #策略的初始化：传入均线参数、资金账户 ID、合约代码
    def __init__(self, fast_length, slow_length, account_id, symbol):
        super().__init__('My_DualMA')
        self._fast_length = fast_length
        self._slow_length = slow_length
        self._account_id = account_id
        self._symbol = symbol
        self._account = None
        self._tick = None
        self._pos = None
        self._order_dict = {}
        self._timer_id = 0
        pass

    def __del__(self):
        pass
    #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）
    def on_init(self, context):
        ret = context.subscribe_tick(symbol=self._symbol)
        if ret is not None:
            print(ret)
            tbpy.exit()
        ret = context.subscribe_bar(symbol=self._symbol, frequency='10s',
begin_time=datetime.datetime.now()-
```

```
datetime.timedelta(minutes=self._slow_length), sliding_window=self._slow_length)
        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()
        self._timer_id = context.create_timer(interval_millsecs=5000)
        self._tick = tbpy.get_current_tick(symbol=self._symbol)
        self._pos = self._account.get_position(symbol=self._symbol)
        print('on_init success.')
    #on_bar 事件：计算均线指标、根据规则发送委托单
    def on_bar(self, context, bars, symbol, flag):
        print(bars[len(bars)-1])
        # flag=0 历史数据; flag=1 实时数据
        if flag != 1:
            return
        if self._tick is None:
            return
        if self._account.get_status() != tbpy.AccountStatus.OnService:
            return
        fast_avg = sum(bar.close for bar in bars[(-self._fast_length - 1):-1])
/ self._fast_length
        slow_avg = sum(bar.close for bar in bars[(-self._slow_length - 1):-1])
/ self._slow_length
        if fast_avg > slow_avg:
            if self._pos is None:
                self.push_order_id(self._account.buy(symbol=symbol,   volume=1,
price=self._tick.last))
            else:
                if self._pos.s_can_cover_volume > 0:
                    self.push_order_id(self._account.buy2cover(symbol=symbol,
volume=self._pos.s_can_cover_volume, price=self._tick.last))
                if  self._pos.l_current_volume  +  self._pos.l_active_volume  -
self._pos.l_active_close_volume < 2:
                    self.push_order_id(self._account.buy(symbol=symbol,
volume=1, price=self._tick.last))
        elif fast_avg < slow_avg:
            if self._pos is None:
                self.push_order_id(self._account.sell2short(symbol=symbol,
volume=1, price=self._tick.last))
            else:
                if self._pos.l_can_sell_volume > 0:
```

```
                    self.push_order_id(self._account.sell(symbol=symbol,
volume=self._pos.l_can_sell_volume, price=self._tick.last))
                if  self._pos.s_current_volume  +  self._pos.s_active_volume  -
self._pos.s_active_close_volume < 2:
                    self.push_order_id(self._account.sell2short(symbol=symbol,
volume=1, price=self._tick.last))

    #on_tick 事件: 接收实时 tick,用于交易委托的委托价确定
    def on_tick(self, context, tick):
        # print(tick)
        self._tick = tick
    #on_position 时间: 获取最新持仓，用于交易委托的类型判断
    def on_position(self, context, pos):
        print(pos)
        self._pos = pos
    #on_order 事件: 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)
    #on_fill 事件: 打印成交
    def on_fill(self, context, fill):
        print(fill)
        pass
    #on_timer 事件: 定时检查委托超时进行撤单
    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 >= 10:
                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 初始化
    ret = tbpy.init()
    if ret is False:
        print('init fail.')
```

```
        sys.exit()
    #获取 rb 的主力合约
    rb_main = tbpy.get_main_instrument(underlying_symbol='rb.SHFE')
    if rb_main is None:
        sys.exit()
    #定义一个策略对象
    strategy = My_DualMA(5, 10, 'tbyihao1', rb_main.symbol)
    #执行策略
    tbpy.exe()
```

运行结果：