PythonGo 策略开发文档

目 录

1. 函数说明

1.1 回调函数说明

函数名	函数说明	触发条件	推送数据
onInit	策略初始化回调	用户创建策略实例	无
onStart	策略启动回调	点击面板“运行”按钮	无
onStop	停止策略回调	点击面板“暂停”按钮	无
onTick	行情切片回调	收到推送的行情切片数据	VtTickData
onBar	1 分钟 K 线回调	收到推送的 K 线数据	VtBarData
onXminBar	X 分钟 K 线回调	BarManager 自行推送	VtBarData
onOrder	委托回报回调	收到推送的委托状态变化信息	VtOrderData
onTrade	成交回报回调	收到推送的成交状态信息	VtTradeData
onErr	错误回调	收到错误推送	记录错误信息

1.2 主要函数说明

函数名	函数功能	传入参数	返回数据
buy	期货买入开仓；股票买入	price, volume, symbol, exchange, stop, investor	vtOrderID
short	期货卖出开仓	price, volume, symbol, exchange, stop, investor	vtOrderID
cover	期货买入平仓	price, volume, symbol, exchange, stop, investor	vtOrderID
sell	期货卖出平仓；股票卖出	price, volume, symbol, exchange, stop, investor	vtOrderID
buy_fak	FAK 指令的 buy 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
short_fak	FAK 指令的 short 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
cover_fak	FAK 指令的 cover 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
sell_fak	FAK 指令的 sell 函数	price, volume, symbol, exchange, stop, investor	vtOrderID

buy_fok	FOK 指令的 buy 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
short_fok	FOK 指令的 short 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
cover_fok	FOK 指令的 cover 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
sell_fok	FOK 指令的 sell 函数	price, volume, symbol, exchange, stop, investor	vtOrderID
cover_t	强制平今的 cover 函数，不对仓位做检查	price, volume, symbol, exchange, stop, investor	vtOrderID
sell_t	强制平今的 sell 函数，不对仓位做检查	price, volume, symbol, exchange, stop, investor	vtOrderID
cover_y	强制平昨的 cover 函数，不对仓位做检查	price, volume, symbol, exchange, stop, investor	vtOrderID
sell_y	强制平昨的 sell 函数，不对仓位做检查	price, volume, symbol, exchange, stop, investor	vtOrderID
cancelOrder	撤单	vtOrderID	无
output	日志输出函数	无	日志信息
writeCtaLog	记录日志信息	待记录信息	无
getGui	创建界面	无	无
cloaseGui	关闭界面	无	无
putEvent	策略状态推送函数	无	无
setParam	刷新参数函数	参数的字典	无
getInvestorAccount(暂不支持)	获取账户资金信息	investorID	VtAccountData
getInvestorPosition	获取账户持仓信息	investorID	VtPositionData
loadDay	载入历史日线数据，支持近 2 年的数据	years, symbol, exchange, func	无
loadBar	载入历史分钟线数据，支持近 3 天的数据	days, symbol, exchange, func	无

2. 数据结构详情

2.1 行情切片数据

行情切片数据，来源为交易所推送的行情切片

```
class VtTickData(VtBaseData):
    """
    Tick 行情数据类,来源为交易所推送的行情切片
    """
    def __init__(self):
        """Constructor"""
        super(VtTickData, self).__init__()

        # 代码相关
        self.symbol = EMPTY_STRING          # 合约代码
        self.exchange = EMPTY_STRING        # 交易所代码
        self.vtSymbol = EMPTY_STRING        # 合约代码.交易所代码

        # 成交数据
        self.lastPrice = EMPTY_FLOAT        # 最新成交价
        self.lastVolume = EMPTY_INT         # 最新成交量
        self.volume = EMPTY_INT             # 今天总成交量
        self.openInterest = EMPTY_INT       # 持仓量
        self.time = EMPTY_STRING            # 时间  11:20:56.5
        self.date = EMPTY_STRING            # 日期  20151009
        self.datetime = None                # 合约时间

        # 常规行情
        self.openPrice = EMPTY_FLOAT        # 今日开盘价
        self.highPrice = EMPTY_FLOAT        # 今日最高价
        self.lowPrice = EMPTY_FLOAT         # 今日最低价
        self.preClosePrice = EMPTY_FLOAT

        self.upperLimit = EMPTY_FLOAT       # 涨停价
```

```
        self.lowerLimit = EMPTY_FLOAT     # 跌停价

        self.turnover = EMPTY_FLOAT       # 成交额

        # 五档行情
        self.bidPrice1 = EMPTY_FLOAT
        self.bidPrice2 = EMPTY_FLOAT
        self.bidPrice3 = EMPTY_FLOAT
        self.bidPrice4 = EMPTY_FLOAT
        self.bidPrice5 = EMPTY_FLOAT

        self.askPrice1 = EMPTY_FLOAT
        self.askPrice2 = EMPTY_FLOAT
        self.askPrice3 = EMPTY_FLOAT
        self.askPrice4 = EMPTY_FLOAT
        self.askPrice5 = EMPTY_FLOAT

        self.bidVolume1 = EMPTY_INT
        self.bidVolume2 = EMPTY_INT
        self.bidVolume3 = EMPTY_INT
        self.bidVolume4 = EMPTY_INT
        self.bidVolume5 = EMPTY_INT

        self.askVolume1 = EMPTY_INT
        self.askVolume2 = EMPTY_INT
        self.askVolume3 = EMPTY_INT
        self.askVolume4 = EMPTY_INT
        self.askVolume5 = EMPTY_INT
```

2.2 行情 K 线数据

行情 K 线数据，由交易所推送的行情切片数据合成

```
class VtBarData(VtBaseData):
    """
    K 线数据
```

```
    """
    def __init__(self):
        """Constructor"""
        super(VtBarData, self).__init__()

        self.vtSymbol = EMPTY_STRING      # vt 系统代码
        self.symbol = EMPTY_STRING        # 代码
        self.exchange = EMPTY_STRING      # 交易所

        self.open = EMPTY_FLOAT           # OHLC
        self.high = EMPTY_FLOAT
        self.low = EMPTY_FLOAT
        self.close = EMPTY_FLOAT

        self.date = EMPTY_STRING          # bar 开始的时间，日期
        self.time = EMPTY_STRING          # 时间
        self.datetime = None              # python 的 datetime 时间对象

        self.volume = EMPTY_INT           # 成交量
        self.openInterest = EMPTY_INT     # 持仓量
```

2.3 委托回报数据

委托回报数据，来源为交易所推送的委托回报

```
class VtOrderData(VtBaseData):
    """
    订单数据类,来源为交易所推送的委托回报
    """
    def __init__(self):
        """Constructor"""
        super(VtOrderData, self).__init__()

        # 代码编号相关
        self.symbol = EMPTY_STRING        # 合约代码
        self.exchange = EMPTY_STRING      # 交易所代码
```

```
        self.vtSymbol = EMPTY_STRING      # 交易所代码

        self.orderID = EMPTY_STRING       # 订单编号
        self.vtOrderID = EMPTY_STRING     # 订单编号

        # 报单相关
        self.direction = EMPTY_UNICODE    # 报单方向
        self.offset = EMPTY_UNICODE       # 报单开平仓
        self.price = EMPTY_FLOAT          # 报单价格
        self.priceType = EMPTY_UNICODE    # 报单价格
        self.totalVolume = EMPTY_INT      # 报单总数量
        self.tradedVolume = EMPTY_INT     # 报单成交数量
        self.status = EMPTY_UNICODE       # 报单状态

        self.orderTime = EMPTY_STRING     # 发单时间
        self.cancelTime = EMPTY_STRING    # 撤单时间

        # CTP/LTS 相关
        self.frontID = EMPTY_INT          # 前置机编号
        self.sessionID = EMPTY_INT        # 连接编号
```

2.4 成交回报数据

成交回报数据，来源为交易所推送的成交回报

```
class VtTradeData(VtBaseData):
    """
    成交数据类,来源为交易所推送的成交回报
    """
    def __init__(self):
        """Constructor"""
        super(VtTradeData, self).__init__()

        # 代码编号相关
        self.symbol = EMPTY_STRING        # 合约代码
        self.exchange = EMPTY_STRING      # 交易所代码
```

```
        self.vtSymbol = EMPTY_STRING      # 合约代码.交易所代码

        self.tradeID = EMPTY_STRING       # 成交编号
        self.vtTradeID = EMPTY_STRING     # 成交编号

        self.orderID = EMPTY_STRING       # 订单编号
        self.vtOrderID = EMPTY_STRING     # 订单编号

        # 成交相关
        self.direction = EMPTY_UNICODE    # 成交方向
        self.offset = EMPTY_UNICODE       # 成交开平仓
        self.price = EMPTY_FLOAT          # 成交价格
        self.volume = EMPTY_INT           # 成交数量
        self.tradeTime = EMPTY_STRING     # 成交时间

        self.commission = EMPTY_FLOAT     # 手续费
```

2.5 用户持仓数据

用户持仓数据，用来记录账户的持仓情况

```
class VtPositionData(VtBaseData):
    """持仓数据类"""
    def __init__(self):
        """Constructor"""
        super(VtPositionData, self).__init__()

        # 代码编号相关
        self.symbol = EMPTY_STRING              # 合约代码
        self.exchange = EMPTY_STRING            # 交易所代码
        self.vtSymbol = EMPTY_STRING            # 合约在 vt 系统中的唯一代码，合约代码.
交易所代码

        # 持仓相关
        self.direction = EMPTY_STRING           # 持仓方向
        self.position = EMPTY_INT               # 持仓量
```

```
        self.frozen = EMPTY_INT                 # 冻结数量
        self.price = EMPTY_FLOAT                # 持仓均价
        self.vtPositionName = EMPTY_STRING  # 持仓在 vt 系统中的唯一代码，通常是 vtSymbol.方向
        self.investorID = EMPTY_STRING          # 投资者
        self.investor = EMPTY_STRING            # 投资者
        self.positionProfit = EMPTY_FLOAT       # 平仓盈亏

        # 20151020 添加
        self.ydPosition = EMPTY_INT             # 昨持仓
```

2.6 资金账户数据

资金账户数据，用来记录账户的资金情况

```
class VtAccountData(VtBaseData):
    """账户数据类"""
    def __init__(self):
        """Constructor"""
        super(VtAccountData, self).__init__()

        # 账号代码相关
        self.datetime = None
        self.accountID = EMPTY_STRING           # 账户代码
        self.vtAccountID = EMPTY_STRING     # 账户在 vt 中的唯一代码，通常是 Gateway 名.账户代码

        # 数值相关
        self.preBalance = EMPTY_FLOAT           # 昨日账户结算净值
        self.balance = EMPTY_FLOAT              # 账户净值
        self.available = EMPTY_FLOAT            # 可用资金
        self.commission = EMPTY_FLOAT           # 今日手续费
        self.margin = EMPTY_FLOAT               # 保证金占用
        self.closeProfit = EMPTY_FLOAT          # 平仓盈亏
        self.positionProfit = EMPTY_FLOAT       # 持仓盈亏
```

2.7 合约数据

合约数据，用来记录合约的状态信息

```
class VtContractData(VtBaseData):
    """合约详细信息类"""
    def __init__(self):
        """Constructor"""
        super(VtBaseData, self).__init__()

        self.symbol = EMPTY_STRING    # 代码
        self.exchange = EMPTY_STRING    # 交易所代码
        self.vtSymbol = EMPTY_STRING    # 合约代码.交易所代码
        self.name = EMPTY_UNICODE    # 合约中文名

        self.productClass = EMPTY_UNICODE    # 合约类型
        self.size = EMPTY_INT    # 合约大小
        self.priceTick = EMPTY_FLOAT    # 合约最小价格 TICK

        # 期权相关
        self.strikePrice = EMPTY_FLOAT    # 期权行权价
        self.underlyingSymbol = EMPTY_STRING    # 标的物合约代码
        self.optionType = EMPTY_UNICODE    # 期权类型
```

3. 常量定义

3.1 默认空值

```
EMPTY_STRING = ''
EMPTY_UNICODE = u''
EMPTY_INT = 0
EMPTY_FLOAT = 0.0
```

3.2 委托方向常量

```
DIRECTION_NONE = u'无方向'
DIRECTION_LONG = u'多'
DIRECTION_SHORT = u'空'
DIRECTION_UNKNOWN = u'未知'
DIRECTION_NET = u'净'
DIRECTION_SELL = u'卖出'          # IB 接口
```

3.3 开平常量

```
OFFSET_NONE = u'无开平'
OFFSET_OPEN = u'开仓'
OFFSET_CLOSE = u'平仓'
OFFSET_CLOSETODAY = u'平今'
OFFSET_CLOSEYESTERDAY = u'平昨'
OFFSET_UNKNOWN = u'未知'
```

3.4 委托状态常量

```
STATUS_NOTTRADED = u'未成交'
STATUS_PARTTRADED = u'部分成交'
STATUS_ALLTRADED = u'全部成交'
STATUS_PARTTRADED_PARTCANCELLED = u'部成部撤'
STATUS_CANCELLED = u'已撤销'
STATUS_REJECTED = u'拒单'
```

STATUS_UNKNOWN = u'未知'

3.5 合约类型常量

PRODUCT_EQUITY = u'股票'
PRODUCT_FUTURES = u'期货'
PRODUCT_OPTION = u'期权'
PRODUCT_INDEX = u'指数'
PRODUCT_COMBINATION = u'组合'
PRODUCT_FOREX = u'外汇'
PRODUCT_UNKNOWN = u'未知'
PRODUCT_SPOT = u'现货'
PRODUCT_DEFER = u'延期'
PRODUCT_NONE = ''

3.6 价格类型常量

PRICETYPE_LIMITPRICE = u'限价'
PRICETYPE_MARKETPRICE = u'市价'
PRICETYPE_FAK = u'FAK'
PRICETYPE_FOK = u'FOK'

3.7 期权类型

OPTION_CALL = u'看涨期权'
OPTION_PUT = u'看跌期权'

3.8 交易所类型

EXCHANGE_SSE = 'SSE' # 上交所
EXCHANGE_SZSE = 'SZSE' # 深交所
EXCHANGE_CFFEX = 'CFFEX' # 中金所
EXCHANGE_SHFE = 'SHFE' # 上期所
EXCHANGE_CZCE = 'CZCE' # 郑商所
EXCHANGE_DCE = 'DCE' # 大商所
EXCHANGE_SGE = 'SGE' # 上金所

EXCHANGE_INE = 'INE' # 国际能源交易中心
EXCHANGE_UNKNOWN = 'UNKNOWN' # 未知交易所
EXCHANGE_NONE = '' # 空交易所
EXCHANGE_HKEX = 'HKEX' # 港交所
EXCHANGE_HKFE = 'HKFE' # 香港期货交易所

EXCHANGE_LME = 'LME'
EXCHANGE_GTJA = 'GTJA'
EXCHANGE_BXG = 'BXG'
EXCHANGE_SGX = 'SGX'

EXCHANGE_BMD = 'BMD'
EXCHANGE_CBOT = 'CBOT'
EXCHANGE_NYBOT = 'NYBOT'
EXCHANGE_TOCOM = 'TOCOM'
EXCHANGE_KRX = 'KRX'
EXCHANGE_LIFFE = 'LIFFE'
EXCHANGE_COMEX = 'COMEX'

EXCHANGE_SMART = 'SMART' # IB 智能路由（股票、期权）
EXCHANGE_NYMEX = 'NYMEX' # IB 期货
EXCHANGE_GLOBEX = 'GLOBEX' # CME 电子交易平台
EXCHANGE_IDEALPRO = 'IDEALPRO' # IB 外汇 ECN

EXCHANGE_CME = 'CME' # CME 交易所
EXCHANGE_ICE = 'ICE' # ICE 交易所

EXCHANGE_OANDA = 'OANDA' # OANDA 外汇做市商
EXCHANGE_OKCOIN = 'OKCOIN' # OKCOIN 比特币交易所
EXCHANGE_HUOBI = 'HUOBI' # 火币比特币交易所
EXCHANGE_LHANG = 'LHANG' # 链行比特币交易所

3.9 货币类型

CURRENCY_USD = 'USD' # 美元

```
CURRENCY_CNY = 'CNY'    # 人民币
CURRENCY_HKD = 'HKD'    # 港币
CURRENCY_UNKNOWN = 'UNKNOWN'    # 未知货币
CURRENCY_NONE = ''    # 空货币
```

4. 策略开发教程

4.1 简单策略入门

首先，让我们一步一步来开发一个简单的策略--超过价格触发跌停价 FAK 买入。通过这个简单策略的讲解，希望能让您快速熟悉策略模板，加入策略开发的大家庭。

4.1.1 导入模块

导入策略所需的 Python 模块

```
# encoding: UTF-8


from __future__ import division
from ctaBase import *
from ctaTemplate import *
```

4.1.2 定义策略类

类名必须与文件名相同，继承策略模板 CtaTemplate

```
class DEMOStrategy(CtaTemplate):
    """超过价格发单"""
    vtSymbol = 'rb1801'
    exchange = 'SHFE'
    className = 'DEMOStrategy'
    author = u'rock'
    name = EMPTY_UNICODE    # 策略实例名称
```

4.1.3 定义策略参数和状态变量

- 参数：策略可调整的变量，如买入触发价(P)和下手单数(V)
- 状态变量：用于实盘展示的变量，如交易中(trading)和仓位(pos)

```
    # 策略参数
    P = 20    # 买入触发价
    V = 1    # 下单手数
```

```
# 参数列表，保存了参数的名称
paramList = ['P',
             'V']

# 变量列表，保存了变量的名称
varList = ['trading',
           'pos']
```

4.1.4 定义参数和变量的映射表

```
# 参数映射表
paramMap = {'P': u'买触发价',
            'V': u'下单手数',
            'exchange': u'交易所',
            'vtSymbol': u'合约'}

# 变量映射表
varMap = {'trading': u'交易中',
          'pos': u'仓位'}
```

4.1.5 初始化参数

在策略类的初始化函数中，初始化策略参数

```
def __init__(self, ctaEngine=None, setting={}):
    """Constructor"""
    super(DEMOStrategy, self).__init__(ctaEngine, setting)

    self.P = 1   # 下单手数
    self.V = 1   # 下单手数
```

4.1.6 编写行情处理函数

在这个例子中我们过滤了无效行情，在价格满足条件后发单

```
def onTick(self, tick):
    """收到行情 TICK 推送（必须由用户继承实现）"""
    super(DEMOStrategy, self).onTick(tick)
    # 过滤涨跌停和集合竞价
    if tick.lastPrice == 0 or tick.askPrice1 == 0 or tick.bidPrice1 == 0:
        return
    if tick.lastPrice > self.P:
        self.orderID = self.buy_fak(tick.lastPrice, self.V)
```

4.1.7. 完善策略结构

继承实现 onTrade 和 onStart 函数

```
def onTrade(self, trade, log=False):
    super(DEMOStrategy, self).onTrade(trade, log=True)


def onStart(self):
    super(DEMOStrategy, self).onStart()
```

4.2 复杂策略进阶

现在我们来学习一个相对复杂的策略——双均线策略。基本的策略开发流程在 4.1 章节中已经做了介绍，所以本节我们着重了解一些不一样内容。

4.2.1 策略交易逻辑

- 当短期平均线向上穿越长期平均线时，意味着买入信号
- 当短期平均线向下穿越长期平均线时，意味着卖出信号

4.2.2 参数与变量

- 策略参数：
  - N——快均线周期
  - P——慢均线周期
  - A——止损比例
  - V——下单手数
- 策略状态变量：

- ma0——慢均线数值
- ma1——快均线数值
- pos——策略仓位

4.2.3 主体编写逻辑

“双均线策略”通过 K 线的数据计算出技术指标；通过技术指标产生交易信号；通过执行信号来完成发单撤单操作。策略的 onBar 函数如下所示，我们将按照 onBar 函数中的主体逻辑来完成策略的整体编写。

```
def onBar(self, bar):
    """收到 Bar 推送（必须由用户继承实现）"""
    self.bar = bar
    if self.tradeDate != bar.date:
        self.tradeDate = bar.date

    # 记录数据
    if not self.am.updateBar(bar):
        return

    # 计算指标
    self.getCtaIndictor(bar)

    # 计算信号
    self.getCtaSignal(bar)

    # 简易信号执行
    self.execSignal(self.V)

    # 发出状态更新事件
    if self.widget is not None and self.bar is not None:
        data = {'bar': self.bar,
                'sig': self.signal,
                'ma0': self.ma0,
                'ma1': self.ma1,
                'cost': self.cost}
```

```
            self.widget.addBar(data)
        if self.trading:
            self.putEvent()
```

4.2.4 缓存数据

有两个辅助类，分别为 BarManager 和 ArrayManager。BarManger 可以基于 Tick 合成 1 分钟 K 线，也可以基于 1 分钟 K 线合成 X 分钟的 K 线；ArrayManager 负责 K 线时间序列的维护，以及常用技术指标的计算。本策略中的数据缓存我们通过 ArraryManager 来实现。

```
        # 记录数据
        if not self.am.updateBar(bar):
            return
```

4.2.5 计算指标数据

ArrayManager 中已经封装好了一些技术指标计算的功能函数，直接调用即可。

```
    def getCtaIndictor(self, bar):
        """计算指标数据"""
        ma = self.am.sma(self.P, True)
        ma1 = self.am.sma(self.N, True)
        self.ma0, self.ma00 = ma[-1], ma[-2]
        self.ma1, self.ma10 = ma1[-1], ma1[-2]
```

4.2.6 计算交易信号

根据上一步中计算得出的快均线和慢均线的数值，我们可以计算出开仓的买入信号和卖出信号；并且可以利用反向信号和止损设置，产生平仓信号。

```
    def getCtaSignal(self, bar):
        """计算交易信号"""
        close = bar.close
        hour = bar.datetime.hour
        minute = bar.datetime.minute
        # 定义尾盘，尾盘不交易并且空仓
```

```
        self.endOfDay = hour == 14 and minute >= 40
        # 判断是否要进行交易
        self.buySig = self.ma1 > self.ma0 and self.ma10 < self.ma00
        self.shortSig = self.ma1 < self.ma0 and self.ma10 > self.ma00
        self.coverSig = self.buySig or close >= self.cost + self.A * close
        self.sellSig = self.shortSig or close <= self.cost - self.A * close
        # 交易价格
        self.longPrice = bar.close
        self.shortPrice = bar.close
```

4.2.7 执行交易信号

如果之前有未成交的挂单，优先撤单；之后执行上一步中计算出的交易信号，完成发单。

```
    def execSignal(self, volume):
        """交易信号执行"""
        pos = self.pos[self.vtSymbol]
        endOfDay = self.endOfDay
        # 挂单未成交
        if self.orderID is not None:
            self.cancelOrder(self.orderID)

        self.signal = 0
        # 当前无仓位
        if pos == 0 and not self.endOfDay:
            # 买开，卖开
            if self.shortSig:
                self.signal = -self.shortPrice
                self.orderID = self.short(self.shortPrice, volume)
            elif self.buySig:
                self.signal = self.longPrice
                self.orderID = self.buy(self.longPrice, volume)

        # 持有多头仓位
        elif pos > 0 and (self.sellSig or self.endOfDay):
```

```
            self.signal = -self.shortPrice
            self.orderID = self.sell(self.shortPrice, pos)

        # 持有空头仓位
        elif pos < 0 and (self.coverSig or self.endOfDay):
            self.signal = self.longPrice
            self.orderID = self.cover(self.longPrice, -pos)
```

4.2.8 策略状态推送

最后需要发出状态更新事件，把策略的运行状态推送给 K 线界面和无限易的 C++策略引擎。

```
        # 发出状态更新事件
        if self.widget is not None and self.bar is not None:
            data = {'bar': self.bar,
                    'sig': self.signal,
                    'ma0': self.ma0,
                    'ma1': self.ma1,
                    'cost': self.cost}
            # 推送至 K 线界面
            self.widget.addBar(data)
        if self.trading:
            # 推送至无限易 C++策略引擎
            self.putEvent()
```

4.2.9 策略界面支持

如果策略需要 K 线显示的功能，我们需要在初始化函数中，定义买卖信号及在主图、附图中显示的技术指标；并需要调用 getGui 函数，完成界面的初始化。

```
def __init__(self, ctaEngine=None, setting={}):
    """Constructor"""
    super(DMAStrategy, self).__init__(ctaEngine, setting)
```

```
        # 启动界面
        self.signal = 0    # 买卖标志
        self.mainSigs = ['ma0', 'ma1', 'cost']    # 主图显示
        self.subSigs = []    # 副图显示
        self.getGui()
```

4.2.10 完善策略结构

策略启动时，需要载入历史数据，并创建界面；策略停止时，需要清理数据并关闭界面。

```
    def onStart(self):
        self.loadBar(3)
        super(DMAStrategy, self).onStart()
        self.getGui()

    def onStop(self):
        super(DMAStrategy, self).onStop()
        if self.widget is not None:
            self.widget.clear()
            self.closeGui()
```

4.2.11 完整策略代码

如果您需要查看“双均线策略”完整源代码，请点击以下链接。

DMAStrategy.py

4.3 K 线相关的类和函数

4.3.1 BarManager 类

BarManger 是一个 K 线合成器类，策略模板中一般实例化为 self.bm，包含如下两个方法。

- updateTick

接收 Tick 数据，合成 1 分钟的 K 线 Bar 数据，并推送策略模板中的 onBar

函数。

- updateBar

接收 1 分钟 K 线 Bar 数据，合成 X 分钟的 K 线 xminBar 数据，并推送策略模板中的 onXminBar 函数。

4.3.2 ArrayManager 类

ArrayManager 是一个 K 线序列管理工具类，策略模板中一般实例化为 self.am，包含如下两类方法。

- updateBar

接收 K 线数据（1 分钟的 Bar 和 X 分钟的 xminBar 均可），缓存在其实例 self.am 的数据序列中。

- 各种技术指标计算函数，类似 talib 的功能。

4.3.3 onTick、onBar 与 onXminBar

- onTick

收到 Tick 行情推送的回调函数。

通过 self.bm.updateTick(tick)，把 Tick 数据合成为 1 分钟的 K 线 Bar 数据。

- onBar

收到 Bar 推送的回调函数。

策略只需要处理 1 分钟 K 线时，通过 self.am.updateBar(bar)，记录数据；收到 Bar 推送时，计算交易指标，产生交易信号。

策略需要处理 X 分钟 K 线时，通过 self.bm.updateBar(bar)，把 1 分钟的 Bar 数据合成为 X 分钟的 xminBar 数据。

- onXminBar

收到 xminBar 推送的回调函数。

通过 self.am.updateBar(bar)，记录数据；收到 xminBar 推送时，计算交易指标，产生交易信号。