AlgoTrading 文档

发布 *0.1.4*

东兴投资

2015 年 12 月 23 日

目录

CHAPTER 1

介绍

1.1 组件

1. *DataAPI*

   提供部门内部数据的访问能力，如果开发基于部门数据库 (datacenter) 的策略需要该模块。

2. *Finance-Python*

   主要提供与金融数据相关的计算功能。

3. *AlgoTrading*

   基于事件循环的回测引擎。

4. *VisualPortfoilo*

   策略回测结果的可视化展现。可以单独使用。

以上所有的项目都可以在 svn 中找到，并且直接通过项目根目录的下的 `setup.py` 文件安装:

```
python setup.py install
```

1.2 流程

下面的图展示了，各个模块在运行时的互相作用关系:

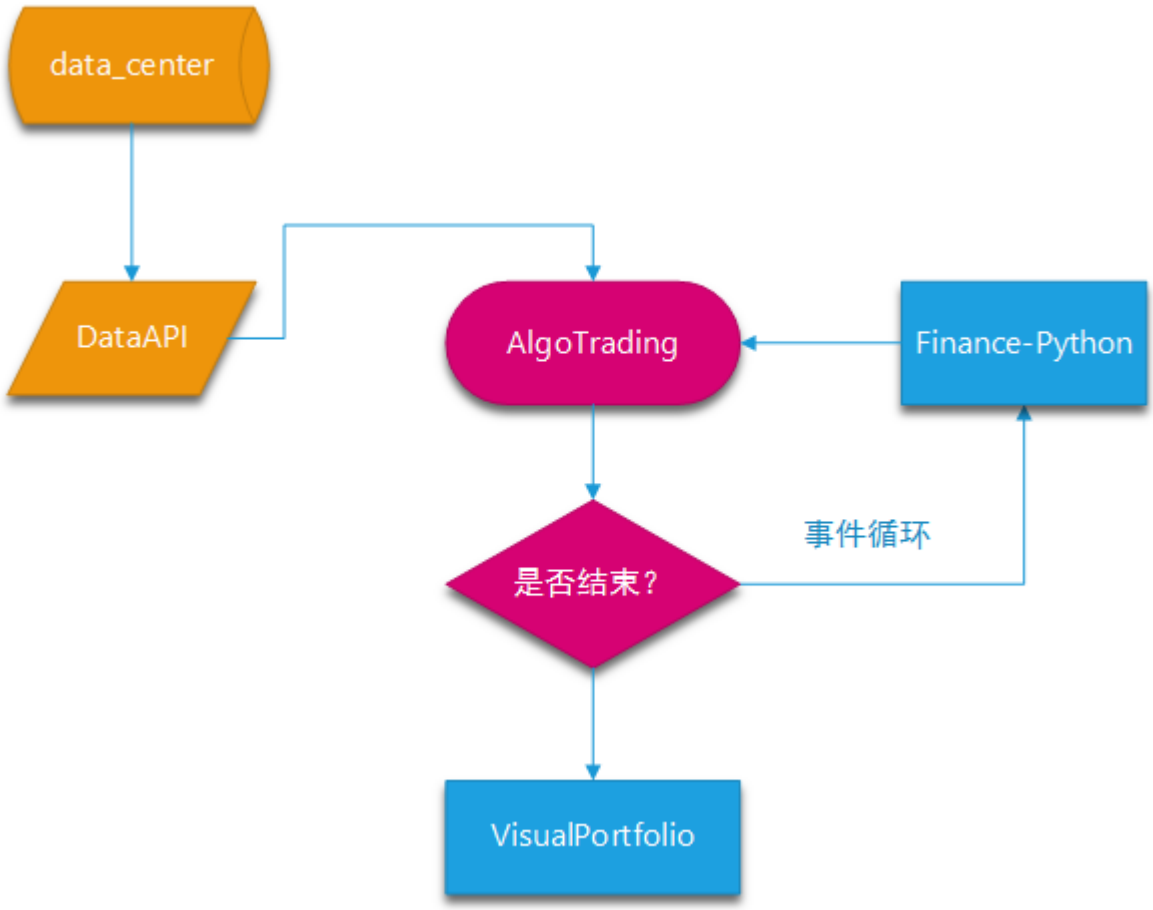

图 1.1: 策略运行回测流程图

1.3 如何开始一个策略

1.3.1 定义策略

用户自定义的策略继承自基类 `Strategy`, 其中用户需要自定义的包括两部分:

- `__init__`

  初始化函数，在策略启动的时候运行，主要用于定义比如:

  1. 全局变量
  2. 指标（由 Finance-Python 模块提供）

- `handle_data`

  行情数据处理函数，每根 bar 推送至回测引擎时候出发。这里是用户交易逻辑的主要定义点。

```
class UserStrategy(strategy):


        def __init__(self):
                ...


        def handle_data(self):
                ...
```

1.3.2 运行策略

当策略就绪之后，直接使用 strategyRunner 进行回测：

```
strategyRunner(strategy=UserStrategy, ...)
```

在 strategyRunner 中需要补充下面几个必填参数：

1. symbolList

   用户关注的行情数据代码，是一个字符串类型的数组（现阶段可以包括，股票、期货以及指数）

2. startDate

   回测周期开始时间，是 python 的 datetiem 类型对象。

3. endDate

   回测周期结束时间，是 python 的 datetiem 类型对象。

4. dataSource

   数据源，默认值为：DataSource.DXDataCenter，意味着使用部门的 datacenter 数据库。

一个典型的 strategyRunner 调用如下形式：

```
strategyRunner(strategy=UserStrategy, \
                symbolList=['600000.xshg', '000300.zicn', 'if1512'], \
                startDate=dt.datetime(2012, 1, 1), \
                endDate=dt.datetiem(2015, 11, 19))
```

CHAPTER 2

Hello World!

这里我们会给出一个最简单的“策略”，最主要的是帮助用户了解在 AlgoTrading 下编写策略的流程。

1. 导入模块的功能

```
In [1]: from AlgoTrading.api import *
```

2. 自定义策略

```
In [2]: class UserStrategy(Strategy):
   ...:     def __init__(self):
   ...:         pass
   ...:     def handle_data(self):
   ...:         self.order('600000.xshg', 1, 100)
   ...:
```

3. 运行策略

```
In [3]: %%time
   ...: import datetime as dt
   ...: res = strategyRunner(userStrategy=UserStrategy, \
   ...:                      symbolList=['600000.xshg'], \
   ...:                      startDate=dt.datetime(2012, 1, 1), \
   ...:                      endDate=dt.datetime(2015, 11, 30), \
   ...:                      benchmark='000300.zicn')
   ...:
Wall time: 4.16 s
```

你可以看到如下所示的回测结果输出：

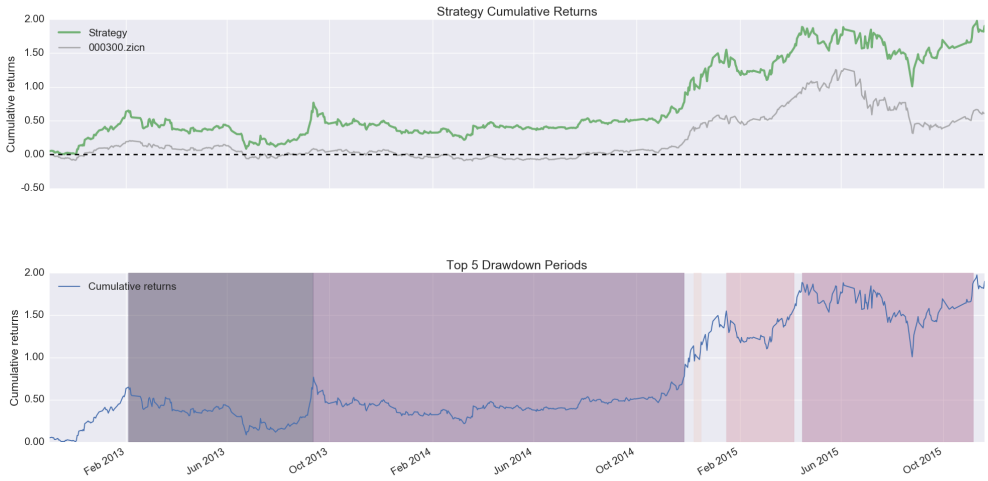

图 2.1: 策略运行回测结果

CHAPTER 3

策略

3.1 AlgoTrading 中的策略

策略是 AlgoTrading 框架下的核心。作为事件循环的主要中转点，策略负责将 MarketEvent 转化为 OrderEvent 并将其推送至队列中供 Execution 使用：

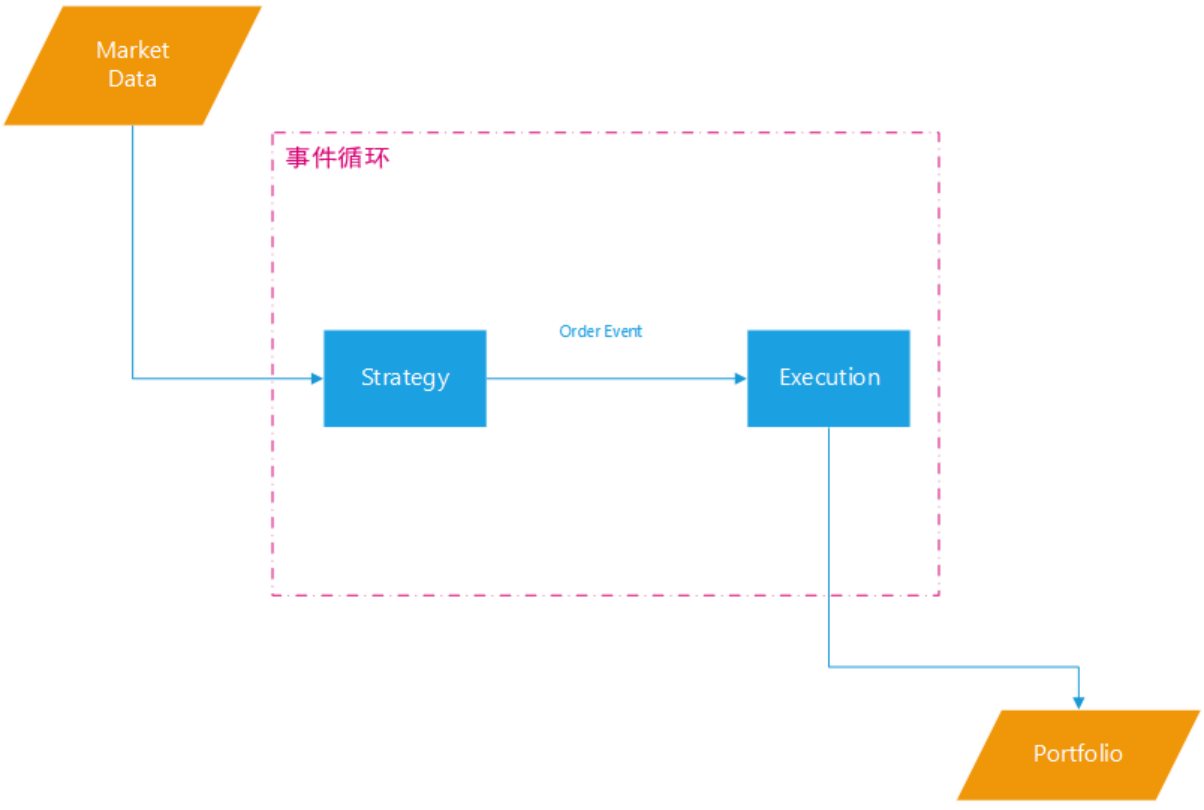

图 3.1: 策略与其他组件的交互

3.2 自定义策略

对于用户而言，策略的逻辑需要他来给出，而给出的方式就是定义如下的自定义策略类型：

```
class UserStrategy(Strategy):
    ...
```

在这个自定义的策略下，用户需要完成两个成员：

- `__init__`

  初始化函数，在策略启动的时候运行，主要用于定义比如：

  1. 全局变量
  2. 指标（由 Finance-Python 模块提供）

- `handle_data`

  行情数据处理函数，每根 bar 推送至回测引擎时候出发。这里是用户交易逻辑的主要定义点。

3.3 策略的成员

策略的基类 `Strategy` 中已经定义了很多成员作为交易的功能函数，例如：下单、获取证券列表等功能。特别需要关注的，例如：

- `order`

  `order` 函数按照指定方向交易指定证券，交易的数量由 `quantity` 参数确定，例如：

  表 3.1: `order` 前账户持仓

证券代码	数量
600000	200
000001	300

  在下达 `order('600000', direction=1, quantity=300)` 的指令后：

  表 3.2: `order` 后账户持仓

证券代码	数量
600000	500
000001	300

- order_to

  order_to 函数按照指定方向交易指定证券，交易的目标由 quantity 参数确定，例如：

  表 3.3: order_to 前

  账户持仓

证券代码	数量
600000	200
000001	300

  在下达 order_to('600000', direction=1, quantity=300) 的指令后：

  表 3.4: order_to 后

  账户持仓

证券代码	数量
600000	300
000001	300

- tradableAssets 属性

  该属性会返回当前行情 bar 下，可交易证券列表。例如用户订阅了包含 3 个代码的行情信息：60000、IF1512 以及 000300。那么在 tradableAssets 属性下会返回的只会包含 600000 以及 IF1512，而指数代码 000300 则不会在其中。

- avaliableForTrade

  该方法返回指定证券在当前账户中的可交易数量（包括可买入和可卖出）。该方法会正确处理例如 T+1 交易方式惯例，例如：用户在 2015 年 12 月 21 日开盘时刻通过 order 指令下单购入某 A 股 300 股并且成功成交。那么在当天任意一根 bar 上调用 avaliableForTrade 函数获取的可交易数量都为 (0, 0)。在下一个交易日，可交易数量会被更新为 (300, 0)。

- keep

  该函数提供了常用的用户自定义日志功能，其中：

  label：值的名称，它将在最后生成的用户信息 DataFrame 中以列名的形式出现；value：值的信息，一般来说是一个数字或者字符串等；time：值的时间戳，会出现在 DataFrame 中的行坐标中。默认为 None，使用当前的 bar 时间戳。

  例如一个典型的对应：

```
        lable('signal', 2.0)
        label('index', 4.0)
```

最后生成的用户信息表可能如下：

表 3.5: 用户记录信息展示

timeStamp	signal	index
2015-12-21	2.0	4.0

3.4 策略模块

class AlgoTrading.api.**Strategy**

avaliableForBuyBack(*symbol*)

返回指定证券当前可买回数量

参数 **symbol** – 证券代码

返回 int

avaliableForSale(*symbol*)

返回指定证券当前可卖出数量

参数 **symbol** – 证券代码

返回 int

avaliableForTrade(*symbol*)

返回指定证券当前账户可交易数量，返回为一个 tuple 类型，分别为可卖出数量和可买回数量

参数 **symbol** – 证券代码

返回 tuple

cash

返回当前账户现金

返回 float

current_datetime

获取策略当前运行的 bar 的时间戳

返回 datetime.datetime

`infoView()`

返回当前所保留的全部用户信息

返回 pandas.DataFrame

`keep(label, value, time=None)`

将用户需要保留的信息保存到指定的时间戳下，供回测后查看

参数

- `label` – 指定信息的名称
- `value` – 指定信息的值
- `time` – 指定的时间戳，若为 None，则使用当前 bar 的时间戳

返回 None

`order(symbol, direction, quantity)`

交易指定量的指定证券

参数

- `symbol` – 证券代码
- `direction` – 方向，1 为买入，-1 为卖出

:param quantity: 交易量:return: None

`order_to(symbol, direction, quantity)`

交易指定证券至指定要求的仓位

参数

- `symbol` – 证券代码
- `direction` – 方向，1 为买入，-1 为卖出
- `quantity` – 指定要求的仓位

返回 None

`tradableAssets`

获取当前所有可交易证券代码列表

返回 list

`universe`

获取当前所有代码列表（包括指数等非交易型代码）

返回 list

CHAPTER 4

资产类型

4.1 资产模块

class AlgoTrading.api.XSHGStock

上海证券交易所股票

commission = **PerValue(buyCost=0.0, sellCost=0.001)**

exchange = **'XSHG'**

lag = **1**

margin = **0.0**

minimum = **100**

multiplier = **1.0**

settle = **1.0**

short = **False**

class AlgoTrading.api.XSHEStock

深圳证券交易所股票

commission = **PerValue(buyCost=0.0, sellCost=0.001)**

exchange = **'XSHE'**

lag = **1**

margin = **0.0**

minimum = **100**

multiplier = **1.0**

settle = 1.0

short = False

class AlgoTrading.api.IFFutures

commission = PerValue(buyCost=0.00015, sellCost=0.00015)

exchange = 'CFFEX'

lag = 0

margin = 0.0

minimum = 1

multiplier = 300

settle = 0.0

short = True

class AlgoTrading.api.ICFutures

commission = PerValue(buyCost=0.00015, sellCost=0.00015)

exchange = 'CFFEX'

lag = 0

margin = 0.0

minimum = 1

multiplier = 200

settle = 0.0

short = True

class AlgoTrading.api.IHFutures

commission = PerValue(buyCost=0.00015, sellCost=0.00015)

exchange = 'CFFEX'

lag = 0

margin = 0.0

minimum = 1

multiplier = 300

settle = 0.0

short = True