Contents

**No table of contents entries found.**

# Overview

## 数据

根据策略需要准备数据，数据分基本面数据（季度）、行情数据（high, low, close等）、日度事件。

基本面数据

获取方式：数据库函数

参数：变量名（‘ACCORECE‘），变量类型（Float），表名（’TQ\_FIN\_PROBALSHEETNEW‘）

自定义日度事件数据

两种方式获取：

1. 数据库函数 参数：变量名（‘TOTMKTCAP‘），变量类型（Float），表名（‘TQ\_QT\_SKDAILYPRICE‘），开始日期（'20140101'），结束日期（'20140331‘）

2. 人工生成 要求：类型为pd.DataFrame，每一行按时间从远及近顺序排列，必须有两列名为’sid’（股票内码）和’dt’（事件发生时刻，类型为UTC datetime）。注意该数据框必须sort by dt

行情数据

不需要用户准备，引擎自动生成

举例：

*为了调用数据库函数，先要导入数据库包*

from zipline.dailydata import DBProxy

*定义日度数据流（事件）开始、结束时间*

period\_start = '20140101'

period\_end = '20140331'

*通过数据库**函数* ***\_get\_fundamentals****调**取应收账款（ACCORECE）数据，类型为浮**点型，该变量位于表****TQ\_FIN\_PROBALSHEETNEW*** *注意只有同时含有****PUBLISHDATE, ENDDATE, SECODE****域的表内的变量可以被该函数调用*

accorece = dbProxy.\_get\_fundamentals({'ACCORECE': float}, 'TQ\_FIN\_PROBALSHEETNEW')

*通过数据库函数* ***\_get\_sn\_ts\_optional****调取总市值（TOTMKTCAP）变量，类型为浮点型，该变量位于表****TQ\_QT\_SKDAILYP******RICE*** *注意只**有含有****TRADEDATE, SECODE****域的表内的变量可以被该函数调用*

cap = dbProxy.\_get\_sn\_ts\_optional({'TOTMKTCAP': float}, 'TQ\_QT\_SKDAILYPRICE', period\_start, period\_end)

*人工生成高管增持减持数据（事件）流：*

import pandas as pd

import os

path = r“C:\myfolder” *给出文件路径*

data = pd.read\_excel(path+"\\"+f, skiprows=0, header=1) *读取文件*

data = data[[u'代码',u'公告日期',u'占流通股比(%)']]

data.columns = ['sid','dt','shares\_chg\_ratio'] *给数据框每一列取名字*

data = data.iloc[0:-2,:] *平移一下*

sharechange = data.sort(columns='dt') *按事件发生先后顺序排列*

sharechange.to\_pickle(path+'\\shareholding change.pkl') *保存处理后数据到pickle文件以便下次直接读取*

## 编写策略

第一步 导入可能会用到的api函数

import zipline as zp

from zipline.api import (

add\_history,

history,

get\_fundamentals,

order,

order\_target\_percent,

order\_target\_value,

cancel\_order,

record,

symbol,

set\_slippage,

set\_commission

)

第二步 初始化

def initialize(context):

*# 注册两个时间序列，供之后策略逻辑使用（例如需要做回归）*

add\_history(20, '1d', 'price')

add\_history(10, ‘1d’, ‘price’)

*# 初始化context中的一些自定义变量*

context.old\_status = None

context.touchlimit = 0

第三步 写策略

def handle\_data(context, data):

*# 策略逻辑部分，可以通过调取api函数和对象进行构建，不会有返回值，一般目标是在一定条件下call 各类 order函数（参见api函数部分）来传达下单命令。当然下单命令不一定要在这里进行，更高级的下单方式今后再做介绍。目前来说，把下单放在这一部分应该可以满足所有回测逻辑的需求。*

第四步 设定回测参数并生成回测器

TradingDictionary = {'initialize' : initialize,

handle\_data' : handle\_data,

'period\_start': period\_start,

'period\_end': period\_end,

'benchmark': 2070000061, *# 基准股指或股票，用于内部模块计算超额收益等*

'warming\_period':10,

}

algo = zp.TradingAlgorithm(\*\*TradingDictionary) *# 生成回测器*

第五步 运行，并获取结果

results = algo.run(False, cap, incl\_index, fundamental\_data = [accorece])

注意第一个参数是bool型变量，True代表用输入的数据覆盖掉之前生成回测器时设定的开始日和结束日，新的开始日结束日会根据输入的数据而定。

从第二个参数直到fundamental\_data关键字，都是日度事件数据，每一个都是根据dt排序的dataframe，传入fundamental\_data必须使用fundamental\_data = list形式传入一个list of fundamentals，内容是之前准备的各种基本面数据表，每个域对应一张表。若没有基本面数据，则可以缺省fundamental\_data关键字。

平台提供非常多的api函数，给用户编写策略提供了诸多方便。下表为常用api

|  |  |  |
| --- | --- | --- |
| API函数 | 功能 | 用法 |
| get\_datetime | 获取目前（回测中的时刻）的时刻（日） | get\_datetime()  不用传任何参数 |
| set\_slippage | 设置市场冲击模型（有默认） | set\_slippage(MySlippage(0.1, 0.0)) |
| set\_commission | 设置佣金模型（有默认） | set\_commission(PerDollar\_A()) |
| record | 记录历史数据，供回测结束后核对 | record(keyed argument list)  *if sid in data:*  *record(close=data[sid].price,*  *short\_mavg=short\_mavg[sid],*  *long\_mavg=long\_mavg[sid])* |
| order | 下单命令，单位是股 | order(sid, amount,  limit\_price=None,  stop\_price=None)  若缺省limit\_price, stop\_price，则默认 market order, 同时存在则为stop-limit order |
| order\_value | 下单，买进（或卖出）一定价值的股票，单位是元 | order\_value(sid, value,  limit\_price=None, stop\_price=None,  ) |
| order\_target | 下单，买进（或卖出）以达到预定的股票仓位，单位是股。买进卖出的量等于目标量减去目前仓位再减去挂单的量（之前挂的单） | order\_target(sid, target,  limit\_price=None, stop\_price=None,  ) |
| order\_target\_value | 下单，买进（或卖出）以达到预定的股票仓位，单位是元。买进卖出的量等于目标量减去目前仓位再减去挂单的量（之前挂的单） | order\_target\_value(sid, target,  limit\_price=None, stop\_price=None,  ) |
| order\_target\_percent | 下单，买进（或卖出）以达到预定的股票仓位，单位是净值百分比。买进卖出的量等于目标量减去目前仓位再减去挂单的量（之前挂的单）。净值包含现金。 | order\_target\_percent(sid, target,  limit\_price=None, stop\_price=None,  ) |
| get\_open\_orders | 获取目前挂出的还未执行的订单。 | get\_open\_orders(sid=None)  默认给出所有订单，给出sid时仅仅返回那一只股票的订单，返回值order对象组成的list，order对象将在后面详细讨论 |
| get\_order | 获取指定order\_id的order对象 | get\_order(order\_id) |
| cancel\_order | 取消订单(order) | cancel\_order(order)，  传入一个order对象，往往需要先调用get\_open\_orders或get\_order获得order对象 |
| add\_history | 注册一个历史序列（为之后获取每日历史序列） | add\_history( lookback\_days, frequency, field, ffill = True) 如果回测时需要过去n个交易日的序列，则lookback\_days设为n， frequency目前只支持’1d’，表示日度，field是历史序列的域，一般为’price’, ‘volume’等（理论上所有浮点型日度数据域都可以支持），ffill 表示是否向前填充值（当某日值为空时），默认为是。该函数通常位于initialize下 |
| history | 获取当日行情序列 | history(N, frequency, field, ffill=True)  返回一个数据框包含当日所有A股往前N个历史交易日（包括当日）的历史行情，由前往后排列，最后一行是当日的观测值。N不能超过add\_history中设置的lookback，field可以是’price’, ‘volume’等，但必须在initialize函数中通过add\_history注册过 |
| get\_fundamentals | 获取基本面数据框 | get\_fundamentals(field, nlookback = 1)  所有A股过去nlookback个季度的基本面数据，N行nlookback列。该函数会自动判断最近的一个季度（只要有一只股票的某季度的值是available，则该季度被认为是available，最近的一个季度是所有available季度中最靠近回测点的。因此在最新季报刚出的那段时间，取出的最左边一列（最近的季度）会有很多NaN（空值） |
| set\_max\_position\_size | 为可持有的头寸做一个限制。  如果系统认为该订单被执行后会让我们的某只股票头寸超过限制，则该单不能通过 | set\_max\_position\_size(  sid=None,  max\_shares=None,  max\_notional=None)  对sid（股票内码），做金额或股数限制 |
| set\_max\_order\_size | 为可下单的规模做一个限制 | set\_max\_order\_size(  sid=None,  max\_shares=None,  max\_notional=None) |
| set\_max\_order\_count | 为每天的下单数做限制，超出部分不能成功 | set\_max\_order\_count(max\_count) |
| set\_long\_only | 不允许持有负的头寸（限制卖空） | set\_long\_only() |
|  |  |  |

## 对象

***context 对象***

来源：初始化时产生，在handle\_data函数中可直接获取

域：默认域、用户自定义域（可以在initialize中初始化自定义域）

更新：一些域每日机器自动更新，用户定义的域只有用户主动改变才会更新

context中的默认域：

context.portfolio：portfolio对象，供用户查看当前组合情况

***data对象***

来源：自动产生，在handle\_data函数中可直接获取

一个dictionary，包含了所有当天的事件数据（今后可能变为分钟级），包括价格事件。

结构：data = {sid1: {‘price’:81, ’high’:83, ’low’:80, …, ’cap’:12100, ’incl\_index’:0},

sid2: {‘price’:47.5, ’high’:48, ’low’:44, …, ’cap’:288900, ’incl\_index’:1},

sid3: …}

具体每一个data[sid]中包含哪些域除了默认的（price, high, low, open, volume）之外取决于回测时用户向系统feed的自定义日度时间数据，准备日度事件数据的过程请参考开头数据部分，提供数据的过程参考本文档最后一部分：运行。

该对象数据也可以用来过滤股票，设定股票池。

***portfolio对象***

来源：自动产生，需要用context.portfolio获取

域：

portfolio.capital\_used = 目前为止使用的现金，正表示盈余，负表示用出，基准为初始现金

portfolio.starting\_cash = 回测开始时的现金总量

portfolio.portfolio\_value = 组合+现金净值

portfolio.pnl = 累积损益（元）

portfolio.returns = 累积收益率

portfolio.cash = 目前现金总额 （portfolio.starting\_cash + portfolio.capital\_used = portfolio.cash）

portfolio.start\_date = 回测开始日

portfolio.positions\_value = self.ending\_value 组合当前的市场价值

portfolio.positions = 头寸情况，返回一个dictionary { sid1: pos1, sid2:pos2 }，通过portfolio.positions[sid]获取单个position对象

***position对象***

来源：自动产生，需要通过pos = portfolio.positions[sid]访问

域：

pos.amount = 头寸数量（股）

pos.cost\_basis = 头寸平均交易成本（元）

pos.last\_sale\_price = 该股票上一个交易日的市场价

***Order对象***

域：

order.sid：股票内码

order.dt：最近一次fill时间

order.amount：下单量（单位股，正负代表买卖）

order.open\_amount：目前还未交易的量

order.filled：已经交易的量

order.commission：为此order交易至今积累的佣金

order.stop：stop price

order.limit：limit price

order.stop\_reached：是否触发stop

order.limit\_reached：是否触发limit