专业·创新·增值

金程教育
GOLDEN FUTURE

第一讲：优矿平台介绍

纪慧诚
金程教育资深培训讲师
CFA FRM RFP

CONTENTS

PROFESSIONAL · LEADING · VALUE-CREATING

专业来自101%的投入!

优矿平台简介

- 数据提供和量化策略研究及测试的平台
- 网址：https://uqer.io/home/
- 优势：

  依靠通联，提供大量因子数据

  活跃的社区

- 支持写法：Quartz3, Quartz2
- Python语言： python2.7

CONTENTS

PROFESSIONAL · LEADING · VALUE-CREATING

专业来自101%的投入!

➢ 策略回测编写框架

```
# 第一步：初始化参数
start = '2016-01-01'                            # 回测起始时间
end = '2017-01-01'                              # 回测结束时间
universe = DynamicUniverse('HS300')             # 证券池，支持股票和基金、期货
benchmark = 'HS300'                             # 策略参考基准
freq = 'd'                                      # 'd'表示使用日频率回测，'m'表示使用分钟频率回测
refresh_rate = 1                                # 执行handle_data的时间间隔


# 第二步：交易账户配置
accounts = {
    'self_account': AccountConfig(account_type='security', capital_base=10000000)
}


# 第三步：编写策略逻辑
def initialize(context):                        # 初始化参数，全局变量的设置，因子signal注册等
    pass

def handle_data(context):                       # 核心策略逻辑
    account = context.get_account('self_account')
```

➢ 第一步 初始化参数

- start与end（回测时间）
  - ✓ Quartz会自动截取start之后第一个交易日和end之前最后一个交易日进行回测

```
start = '2016-01-01'                                    # 回测起始时间
end = '2017-01-01'                                      # 回测结束时间
```

- benchmark（基准）

```
benchmark = 'HS300'                                     # 策略参考基准
```

- universe（证券池）
  - ✓ 策略回测的证券池，即下单与历史数据获取都只能限于universe中的证券。

```
universe = ['600000.XSHG']                              # 静态资产池，指定股票作为资产池
universe = set_universe('HS300', date='20170801')       # 指定证券池列表的日期获得静态资金池
universe = DynamicUniverse('HS300')                     # 动态资产池，部分板块支持，每天动态调整
```

➢ 第一步 初始化参数

- freq与refresh_rate
  - ✓ 共同决定回测频率和调仓频率
  - ✓ freq决定回测的最小时间单位，refresh_rate决定回测间隔时间

```
# 每五个交易日运行一遍策略算法，即handle_data函数
freq = 'd'
refresh_rate = 5

# 每周第一个交易日运行一遍策略算法
freq = 'd'
refresh_rate = Weekly(1)                    # Weekly(-1)，Monthly(1)

# 每天每分钟都运行策略
freq = 'm'
refresh_rate = (1, 1)                       # 第一个1指每天运行，第二个1指每分钟运行
```

- max_history_window
  - ✓ 获取数据时，默认支持回溯100天的数据。当回溯时间超过这个值时，使用max_history_window额外指定。
  - ✓ max_history_window=150

➢ 第二步 交易账户配置

- 使用字典完成账户创建，赋值给accounts
- key为自定义账户名，value为使用AccountConfig类生成的账户实例

```
# 定义账户参数
capital_base = 1000000                                          # 初始资金1000000
commission = Commission(0.0005, 0.0010, unit='perValue')        # 手续费设置，buycost=0.0005
                                                                # sellcost=0.0010，unit=perValue
                                                                # 意味以股价百分比收手续费，买进手续费为
                                                                # 万五，卖出手续费为千一
slippage = Slippage(0.002, unit='perValue')                     # 滑点标准设置
                                                                # 示例意味滑点为千二
position_base = {'000001.XSHE':100, '600000.XSHE':200}          # 初始持仓，一般较少使用
cost_base = {'000001.XSHE':10.00, '600000.XSHE':10.50}          # 初始持仓成本价，一般较少使用


# 进行账户配置
accounts = {'self_account' : AccountConfig(account_type='security', capital_base=capital_base,
commission=commission, slippage=slippage, position_base=position_base, cost_base=cost_base)
}
##################################################################################################
# 账户对象获取
def handle_data(context):
    account = context.get_account('self_account')
```

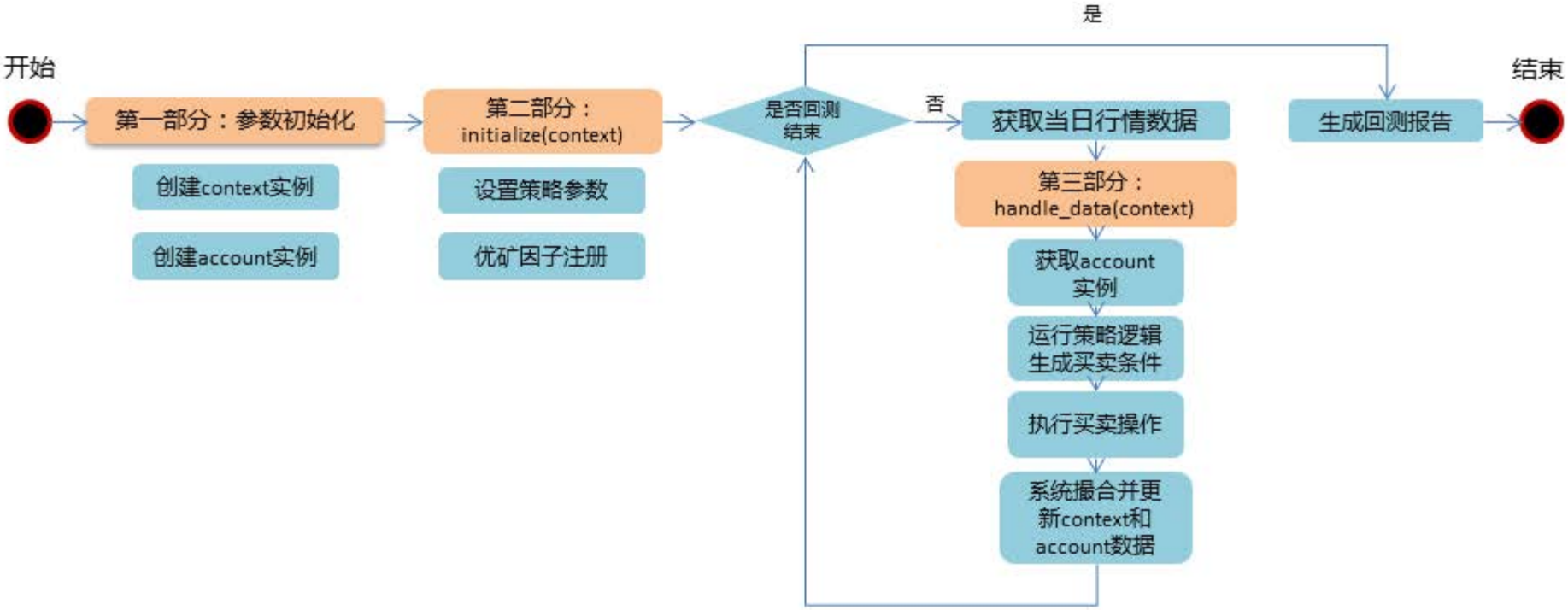
➢ 第三步 编写策略逻辑

- def initialize进行策略初始化
- def handle_data定义具体策略

```
# 策略初始化
def initialize(context):                    # 用于配置策略运行环境context实例的属性或自定义各种变量
    ...                                     # 在整个策略周期只执行一次
                                            # 可以通过给context添加新属性，自定义各种变量，如设置策略
                                              参数

# 编写策略算法
def handle_data(context):                   # 具体策略逻辑编写在此处
                                            # 会根据初始化参数中freq和refresh_rate确定运行频率并运行
    ...                                     # 可通过context获取运行时的行情数据、K线图、因子数据等数
                                              据，并根据分析的结果，通过交易账户对象进行订单委托。
```

➢ 策略日频率回测运行框架



➢ handle_data执行时机和系统订单撮合机制

- **handle_data 执行时机（避免未来数据错误）**
  - ✓ **日线频率回测**

    每天执行一次handle_data，执行时间为交易时间开始前，此时可获得当天盘前信息，以及截止到前一天的行情、因子等数据，不会获得当天的盘中行情；
  - ✓ **分钟频率回测**

    首先在开盘前执行一次handle_data，然后在盘中的每一分钟结束后执行一次handle_data（不包含收盘时刻）
- **订单撮合机制**

  handle_data中生成订单委托，在handle_data执行结束后，统一进行撮合
  - ✓ **日线频率回测**

    遵循**“先卖后买，开盘价撮合”**原则，即先处理卖出订单，后处理买入订单，卖出订单产生的现金，会参与买入订单的交易。由于下单是在当天开盘前，订单撮合会与当天日K线的开盘价进行比较，如果满足条件，就会撮合成交，如果不满足条件，则等待下一次撮合尝试。
  - ✓ **分钟频率回测**

    遵循“先下单先处理，开盘价撮合”原则，即先下单的订单会先进行撮合尝试。订单撮合会与下一分钟K线的开盘价进行比较，如果满足条件，就会撮合成交，如果不满足条件，则等待下一次撮合尝试。

- 系统订单撮合机制
  - 具体成交条件
    - ✓ 市价单

      会以下一根K线的开盘价（openPrice）撮合成交。成交量不超过下一根K线的总成交量，如超过，剩余订单申报量等待下一次撮合成交。
    - ✓ 限价单

      会以下一根K线的开盘价（openPrice）判断是否成交，如果买单申报价小于openPrice 或卖单申报价大于 openPrice，则不成交，等待下一次撮合成交。成交量不超过下一根K线的总成交量，如超过，剩余订单申报量等待下一次撮合成交。
  - 当日收盘后，所有未成交订单都将被系统自动撤单

CONTENTS

PROFESSIONAL · LEADING · VALUE-CREATING

专业来自101%的投入!

Context（策略运行环境）

- Context 表示策略运行环境，包含运行时间、行情数据等内容，还可以用于存储策略中生成的临时数据的存储；
- 策略框架会在启动时创建 Context 的实例，并以参数形式传递给 initialize(context) 和 handle_data(context)，用于策略调度；
- 在回测时，context包含运行时间、回测参数、回测运行时数据等。模拟交易时，包含运行时间、模拟交易参数、实时运行数据等；

Context对象的属性和方法 CFA FRM AQF 视频课程，添加微信：cfafrm9998

```
# 属性
context.now                                  # 策略运行时即时时刻
context.current_date                         # 当前回测日期
context.current_minute                       # 当前运行时的分钟值
context.previous_date                        # 回测日期的前一交易日
# 方法
context.current_price()                      # 获得股票当前参考价格,开盘前运行，获得的是昨
                                               天的收盘价；盘中运行，获得的是最后一次成交价
context.get_account()                        # 获得名字为account_name的交易账户实例
context.get_universe()                       # 获得符合筛选条件的当天上市状态的资产池。
context.history()                            # 获取时间序列数据
```

context.method

- context.get_universe(asset_type, exclude_halt=False)
  - ✓ 获取符合筛选条件的当天上市状态的资产池

```
def handle_data(context):
    universe = context.get_universe()          # 在handle_data中编写获取符合条件资产池程序
                                               # 输出为['000001.XSHE', '600000.XSHG']形式
```

  - ✓ 参数

```
asset_type                                     # 获得指定类型的资产类型的universe
    # 参数可选范围
    # 'stock':股票列表                          'exchange_fund':场内基金列表
    # 'index':指数成分股列表                    'otc_fund':场外基金列表
    # 'futures':期货合约列表                    'base_futures':普通期货合约列表
    # 'continuous_futures':连续期货合约列表
exclude_halt=False                             # 去除资产池中的停盘股票。仅适用于股票
```

context.method

- context.history(symbol, attribute, time_range, freq='1d', style='sat', rtype='frame')
  - ✓ 获取K线图、因子等时间序列数据

```
def handle_data(context):
    data = context.history(['600000.XSHG'], 'closePrice', 10, freq='1d', rtype='frame',  style='sat')
# 在handle_data中编写获取数据程序
```

  - ✓ 参数 CFA FRM AQF 视频课程，添加微信：cfafrm9998

```
symbol                                  # 需要获取的证券列表，支持单个证券或证券列表
attribute                               # 需要获取的属性，支持单个值或属性列表
    # 'openPrice':前复权开盘价          'highPrice':前复权最高价
    # 'lowPrice':前复权最低价           'close_price':前复权收盘价
    # 'preClosePrice':前复权前收盘价    'turnoverVol':前复权成交量
    # 'turnoverValue':前复权成交额      'PE', 'PB'等优矿因子库因子
time_range                              # 所需回溯的历史K线图条数，和 freq 属性相对应
freq                                    # K线图周期，支持'1d', '1m', '5m', '15m', '30m', '60m' 等周期

style                                   # 数据返回的类型，可以选择'ast', 'sat'或者'tas'三
                                        # 种，其中'a'表示'attribute'、's'表示symbol、't'表
                                        # 示时间，三种选择分别对应着这三个维度呈现的顺序
rtype                                   # 返回值的数据类型。可以选择'frame', 'array' 两种
```

context.method

- context.history(symbol, attribute, time_range, freq='1d', style='sat', rtype='frame')
  - ✓ 结合style和rtype 的值，得到不同的返回数据结构

日线情况	DataFrame	array
ast	{'closePrice': 000001.XSHE 600000.XSHG 2017-08-01 10.10 10.00 2017-08-02 10.50 10.30 }	'closePrice':{ '000001.XSHE':array([10.10, 10.50]), '600000.XSHG':array([10.00, 10.30]) 'time':array(['2017-08-01', '2017-08-02'], type='S8')
sat	{'000001.XSHE': closePrice openPrice 2017-08-01 10.10 10.20 2017-08-02 10.50 10.20 }	'000001.XSHE':{ 'closePrice':array([10.10, 10.50]), 'openPrice':array([10.20, 10.20]) 'symbol':array(['000001.XSHE', '600000.XSHG'], dtype='S8')
tas	{'2017-08-02': closePrice openPrice 000001.XSHE 10.50 10.20 600000.XSHG 10.30 10.10 }	'2018-08-02':{ 'closePrice':array([10.50, 10.30]), 'openPrice':array([10.20, 10.10]), 'symbol':array(['000001.XSHE', '600000.XSHG'], dtype='S11')

StockAccount（账户）

- StockAcount 表示股票交易账户，在参数初始化时创建
  - ✓ 交易账户实例获取

```
def handle_data(context):                                    # 在handle_data中编写获取程序
    account = context.get_account('self_account')            # 获得交易账户实例后赋值给account
```

- ✓ account属性


```
# 属性
account.cash                                                 # 当前账户可用资金
account.portforlio_value                                     # 当前账户总权益

# 方法
account.get_position()                                       # 获得指定资产的position实例
account.get_positions()                                      # 获得所有账户持仓的字典，key为证券代码，
                                                               value为position实例
account.order()                                              # 进行订单委托
account.order_to()                                           # 通过下单，将某个股票的持仓调整到持有多少手
account.order_pct()                                          # 根据当前的账户总价值，按一定比例下单
account.order_pct_to()                                       # 通过下单，将某个股票的持仓调整到持有总价值
                                                               的一定比例
account.close_all_positions()                                # 卖掉所有当前持仓
```

account.method

- account.get_position(symbol)
  - ✓ 获取指定资产的持仓信息,如果没有持仓返回None（如果直接调用account.attribute可能会报错）

```
def handle_data(context):
    account = context.get_account('self_account')   # 在handle_data中编写获取account实例
    position = account.get_position('600001.XSHG')  # 获取600001当前持仓情况
```

- account.get_positions(exclude_halt=False)
  - ✓ 获取所有持仓，返回字典，key为证券代码，value为position实例

```
def handle_data(context):
    account = context.get_account('self_account')   # 在handle_data中编写获取account实例
    positions = account.get_positions()             # 获取当前所有持仓

    # 参数
    # exclude_halt=False                            # 移除持仓中，当天无法交易的资产
```

account.method

- account.order(symbol, amount, price=0, otype='market')
  - ✓ 根据指定参数，进行订单委托

```
def handle_data(context):
    account = context.get_account('my_account')      # 在handle_data中编写获取account实例
    universe = context.get_universe()                # 获取股票池中股票列表

    for stock in universe:
        account.order(stock, 100)          # 对在股票池中每一个股票以市价买入一手
# 参数
# symbol:需要交易的证券代码，必须包含后缀,应为字符串
# amount:需要交易的证券代码为symbol的证券数量，为正则为买入，为负则为卖出
# price:下限价单时的下单价格（仅日内策略可用）
# otype:交易指令类型，可选 'market' （市价单）和 'limit' （限价单）（为limit时仅日内策略可用）
```

- account.order_to(symbol, amount, price=0, otype='market')
  - ✓ 通过下单，将某个股票的持仓调整到持有多少手
  - ✓ 策略框架会自动计算当前持仓和目标持仓的差额，并进行下单
  - ✓ 参数参照account.order()

account.method

- account.order_pct(symbol, pct)
  - ✓ 根据账户当前总价值，按一定比例下单

```
def handle_data(context):
    account = context.get_account('self_account')    # 在handle_data中编写获取account实例
    universe = context.get_universe()                # 获取股票池中股票列表

    for stock in universe:
        account.order_pct(stock, 0.1)                # 对在股票池中每一个股票以市价买入当前
                                                     # 总价值的10%
# 参数
# symbol:需要交易的证券代码，必须包含后缀,应为字符串
# pct:需要交易的证券代码为symbol的证券占虚拟账户当前总价值的百分比，范围为 0 ~ 1，为正则为买入，
#     为负则为卖出
```

- account.order_pct_to(symbol, pct)
  - ✓ 通过下单，将某个股票的持仓调整到持有总价值的一定比例
  - ✓ 策略框架会自动计算当前持仓和目标持仓的差额，并进行下单
  - ✓ 用法参照account.order()

Position（仓位）

- 表示某个资产的持仓情况
  - ✓ position实例获取

```
def handle_data(context):                                  # 在handle_data中编写获取程序
    account = context.get_account('my_account')            # 先获得account账户实例
    position = account.get_position(symbol)                # 获取position实例并赋值给position
```

  - ✓ position属性


```
# 属性
position.amount                                            # 持仓数量
position.available_amount                                  # 可卖出持仓数量
position.profit                                            # 持仓浮动盈亏（随市场价格实时变动）
position.cost                                              # 平均开仓成本
position.value                                             # 持仓市值（随市场价格实时变动）
```

CONTENTS

PROFESSIONAL · LEADING · VALUE-CREATING

专业来自101%的投入!

- 优矿因子库（https://uqer.io/help/appendixFactors）

因子类别	示例
成交量性因子	VOL5：5日平均换手率
股指与市值类因子	PE：市盈率
趋势类因子	MACD：平滑异同移动平均线
偿债能力资本结构因子	DebtsAssetRatio：债务总资产比
收益类因子	20日收益方差：Variance20
盈利能力收益质量	GrossIncomeRatio：销售毛利率
能量型因子	BullPower：多头力道
每股指标类因子	DividendPS：每股股利
超买超卖型因子	CCI5：5日顺势指标
运营能力因子	AccountsPayablesTRate：应付账款周转率
均线型因子	MA20：20日移动均线
成长能力类因子	NetAssetGrowRate：净资产增长率
基础科目与衍生类因子	EBIT：息税前利润
现金流指标	CTOP：现金流市值比
分析师预期类因子	DAREC：分析师推荐评级变化

Signal框架用途

- 优矿因子数据在策略运行环境中使用signal框架注册，表明需要相关数据后既可使用


```
# 因子注册
def initialize(context):                                    # 在initialize中编写信号注册程序
    pe = Signal('PE')
    context.signal_generator = SignalGenerator(pe)

# 因子使用
def handle_data(context):
    # 方式一：使用context.history()进行获取
    pe = context.history('600000.XSHG', time_range=1, attribute='PE')
    # 方式二：使用context.signal_result()进行以获取
    pe = context.signal_result('PE')
                          # context.signal_result()运行逻辑
                          # 取到相应信号的计算结果。其结构为字典，键为信号的name属性值，值为
                          # 以context.get_universe() 为索引的Series，表示相应信号的计算结果，横截面数据；
    # 方式三：使用优矿DataAPI，获得某一些股票具体某一天的因子数据；
    pe = DataAPI.MktStockFactorsOneDayGet (tradeDate=context.previous_date, secID=u"",
         ticker=u"000001,600030", field=u"ticker,PE",pandas="1")
```

Factor

- 针对某个因子，进行筛选的约束条件。可以应用在StockScreener和DynamicUniverse中
- 条件编写及筛选方法

```
# 条件编写
Factor.<factor_name>.<筛选条件>

# 筛选条件
value_range(lbound, ubound)          # 筛选因子值处于上下界之间的证券（包含两端）
pct_range(lbound, ubound)            # 筛选因子值处于百分比上下分位点之间的证券（包含两
                                       端），默认升序排列
num_range(lbound, ubound):           # 筛选因子值处于上下界序号之间的证券（包含两端），默
                                       认升序排列
nlarge(n)                            # 筛选因子值最大的n只证券
nsmall(n)                            # 筛选因子值最小的n只证券
```

Factor筛选程序编写方式

- 方式一：StockScreener(<筛选条件表达式>)
- 方式二：DynamicUniverse('A').apply_filter(<筛选条件表达式>)
- 方式三：创建空list对象，将符合条件的股票放进list;(策略常用)


```
# 第一步：初始化参数
...
# 方式一
universe = StockScreener(Factor.PE.nlarge(50)  # 选出PE值较大的50支股票
# 方式二
universe = DynamicUniverse('HS300').apply_filter(Factor.PE.nlarge(50))
                                                  # 从沪深300中选出PE值较大的50支股票
# 第二步：交易账户配置
accounts = {
    'self_account': AccountConfig(account_type='security', capital_base=10000000)
}
# 第三步：编写策略逻辑
def initialize(context):                    # 初始化策略运行环境
    pass
def handle_data(context):                   # 核心策略逻辑
    # 在此处编写方式三的运行逻辑
```

Thank you!