法律声明

本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第五课

量化交易系统的工程实现

量化交易实战：

策略编写与系统搭建

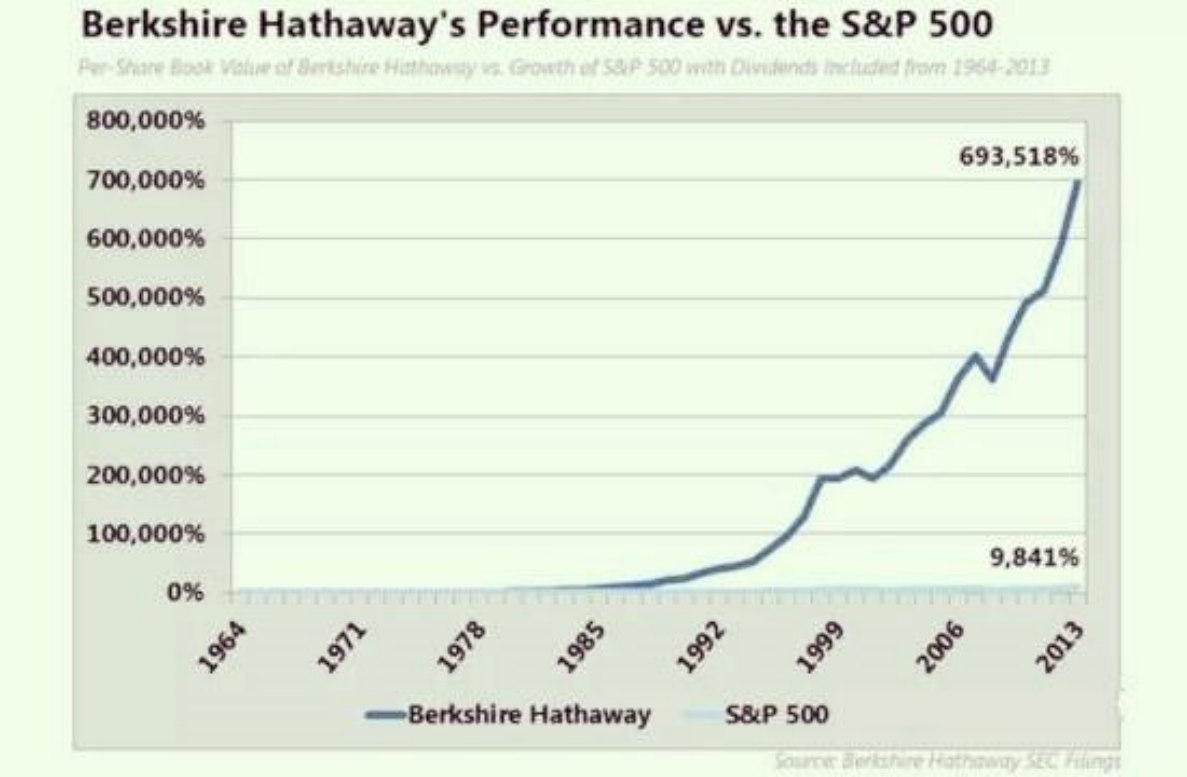
内容介绍

- 复利与不复利
- 复权的秘密和实践
- 行情数据和财报数据的抓取
- 实用量化交易系统的存储解决方案
- 实现一个低估值的股票池择时策略
- 实现回测框架
- 量化交易系统的优化

追求一时的刺激 还是最终的胜利

复利与不复利

巴菲特的复利故事



巴菲特收益率清单					
年份	收益率	年份	收益率	年份	收益率
1965 年	23.80%	1982 年	40.00%	1999 年	0.50%
1966 年	20.30%	1983 年	32.30%	2000 年	6.50%
1967 年	11.00%	1984 年	13.60%	2001 年	-6.20%
1968 年	19.00%	1985 年	48.20%	2002 年	10.00%
1969 年	16.20%	1986 年	26.10%	2003 年	21.00%
1970 年	12.00%	1987 年	19.50%	2004 年	10.50%
1971 年	16.40%	1988 年	20.10%	2005 年	6.40%
1972 年	21.70%	1989 年	44.40%	2006 年	18.40%
1973 年	4.70%	1990 年	7.40%	2007 年	11.00%
1974 年	5.50%	1991 年	39.60%	2008 年	-9.60%
1975 年	21.90%	1992 年	20.30%	2009 年	19.80%
1976 年	59.30%	1993 年	14.30%	2010 年	13.00%
1977 年	31.90%	1994 年	13.90%	2011 年	4.60%
1978 年	24.00%	1995 年	43.10%	2012 年	14.40%
1979 年	35.70%	1996 年	31.80%	2013 年	18.20%
1980 年	19.30%	1997 年	34.10%	2014 年	8.30%
1981 年	31.40%	1998 年	48.30%	平均收益	21.97%

人生就像滚雪球，重要的是发现很湿的雪和很长的坡。

— 沃伦·巴菲特（Warren E. Buffett）
伯克希尔－哈撒韦公司CEO

我们的合伙基金存在的根本原因，就是要以高于平均水平的收益率复利增长，而且长期资本损失的风险比主要投资公司更低。（巴菲特，1963）


平均收益水平
复利持续时间
控制损失风险

复利公式

$$F = P \times (1 + r)^n$$

$$r = \left({}^{F}/_{P}\right)^{\frac{1}{n}} - 1$$

其中：

- P：期初资产
- F：期末资产
- r：平均复合增长率，也被称为几何平均回报率（geometric average return），是指特定投资期间内的平均化复利率
- n：投资期数

单利（不复利）公式

$$F = P \times (1 + r \times n)$$

$$r = \frac{F/P - 1}{n}$$

其中：

- P：期初资产
- F：期末资产
- r：每期的平均收益率
- n：投资期数

年化收益率的计算


收益概述
策略收益
90.51%
策略年化收益
17.93%
基准收益
73.00%
Alpha
0.084
Beta
0.499
Sharpe
0.858
胜率
0.546
盈亏比
1.547
最大回撤
25.963%
其他指标
缩放：
1个月
1年
全部
策略收益
基准收益
超额收益
普通轴
对数轴
超额收益
时间：
2013-12-30
-
2017-12-29
120%
100%
80%
60%
40%
20%
0%
-20%
收益

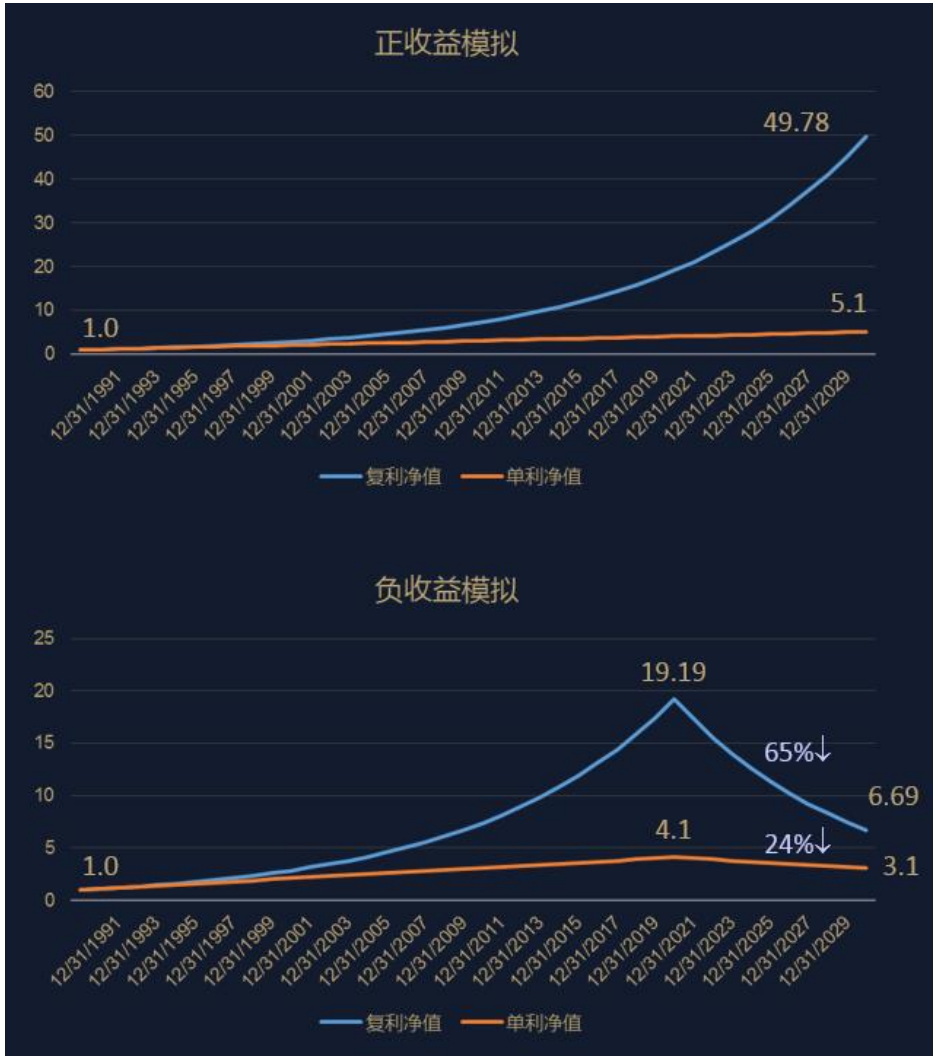
简单对比

复利

所有盈利都投入新一期的投资中：
“利滚利”

不复利

每期新投资都只用到原来的本金：
“落袋为安”



复利的要点

一个期望收益为正的交易系统
严控每期投资收益的回撤，尤其要避免“连续的”亏损
持之以恒

禁忌：大起大落、重仓加杠杆

If anything can go wrong, it will.
--- Murphy's Law

资金管理 vs. 凯利公式

$$f^* = \frac{bp - q}{b} = p - q\frac{1}{b}$$

其中：

- f* = 现有资金中用于下次投注或投资的比例
- b = 赔率（即：盈亏比 = 期望盈利 / 可能亏损）
- p = 成功概率（胜率）
- q = 失败概率（也就是1-p）

凯利公式：在已知胜率和盈亏比的情况下，每次（下注或投资）需要投入现有资金的多少（最优）比例，才能使得长期收益最大化。

思考题

- 尝试从凯利公式的定义出发，解释以下哪个做法，体现了长期盈利的更大的可能性：

  A. 盈利时加仓，亏损时减仓

  B. 盈利时减仓，亏损时加仓

“复利效应”小结

- 正“期望收益”系统
- 严控每期风险和回撤
- 连续亏损时要减仓，而不是相反
- 长期稳定，持之以恒

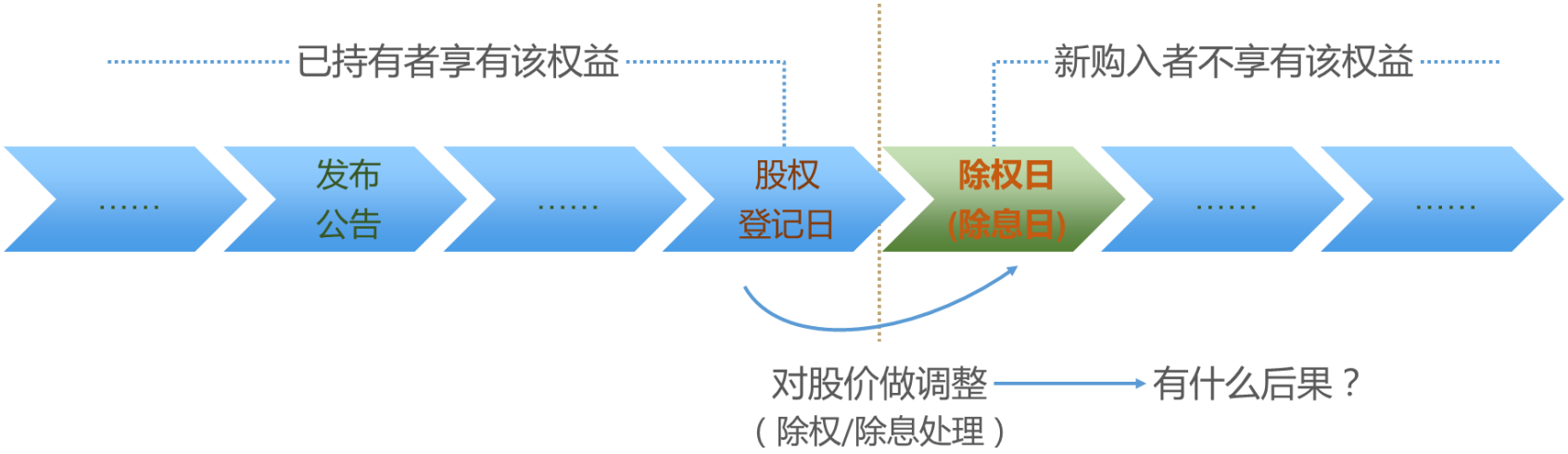
股票量化策略编写中最常见问题的规避

复权的秘密和实践

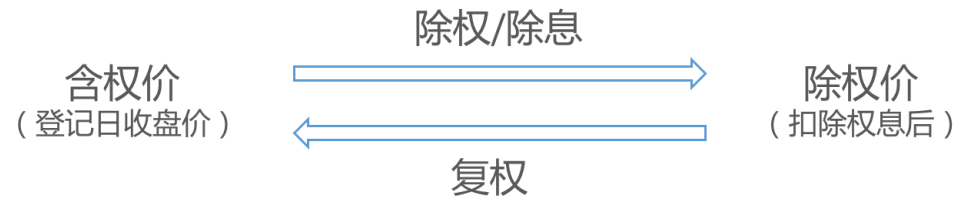
复习一下

- 因为以下事件，每股对应的权益发生了变化
  - 分红
    - 派息
    - 送股
    - 转股（转增）
  - 融资
    - 配股
- 为公平起见，对股价进行修正（除权/除息）
- 为便于观察股价的正常波动，将扣除的权息再回补过去（复权）

“除权日”的概念



$$除权参考价 = \frac{除权前一日收盘价 + 配股价 * 配股率 - 每股派息}{1 + 配股比率 + 送股比率}$$



前复权


正邦科技 (日线,前复权)
深股通
设置均线
[可拖动]
日 期 2016 09/08
开 盘 8.15
最 高 8.39
最 低 8.10
收 盘 8.16
涨 跌 ▼0.11
涨跌幅 -1.33%
成交量 40.6万
金 额 3.36亿
振 幅 3.51%
换手率 3.48%
9.76
8.27
8.16
前复权（从后往前修正）
9.50
9.00
8.50
8.16
8.00

后复权


正邦科技 (日线,后复权)
深股通
设置均线
[可拖动]
日 期 2016 09/08
开 盘 119.75
最 高 123.20
最 低 119.03
收 盘 119.89
涨 跌 ▼1.59
涨跌幅 -1.31%
成交量 40.6万
金 额 3.36亿
振 幅 3.43%
换手率 3.48%
142.88
121.48
119.89
104.05
后复权（从前往后修正）
140.00
135.00
130.00
125.00
119.89
120.00
115.00
110.00
105.00

复权因子

$$单次除权因子_t = \frac{pre\_close_t}{close_{t-1}}$$ (在除权日 $\neq 1$，其它交易日 $\equiv 1$)

$$复权因子_k = \prod_{i=1}^{k} 单次除权因子_i$$ ($k = 1, 2, \ldots$，表示上市以来第k个交易日)

$$前复权价_t = \frac{price_t \times adjfactor_t}{adjfactor_{last}}$$

$$后复权价_t = \frac{price_t \times adjfactor_t}{1.0}$$

($adjfactor$：复权因子，即调整倍数)

行情数据举例

正邦科技（002157）：

code	time	open	close	low	high	pre_close	adjfactor
002157	2016-09-02	27.18	25.75	25.65	27.42	27.37...	4.975982
002157	2016-09-05	25.77	25.56	25.2	25.87	25.75	4.975982
002157	2016-09-06	25.59	25.96	25.41	25.98	25.56	4.975982
002157	2016-09-07	25.88	24.96	24.94	25.88	25.95...	4.975982
002157	2016-09-08	8.2	8.21	8.15	8.44	8.32	14.927946
002157	2016-09-09	8.06	7.98	7.96	8.22	8.21	14.927946
002157	2016-09-12	7.79	7.59	7.52	7.8	7.98	14.927946
002157	2016-09-13	7.63	7.67	7.54	7.69	7.59	14.927946

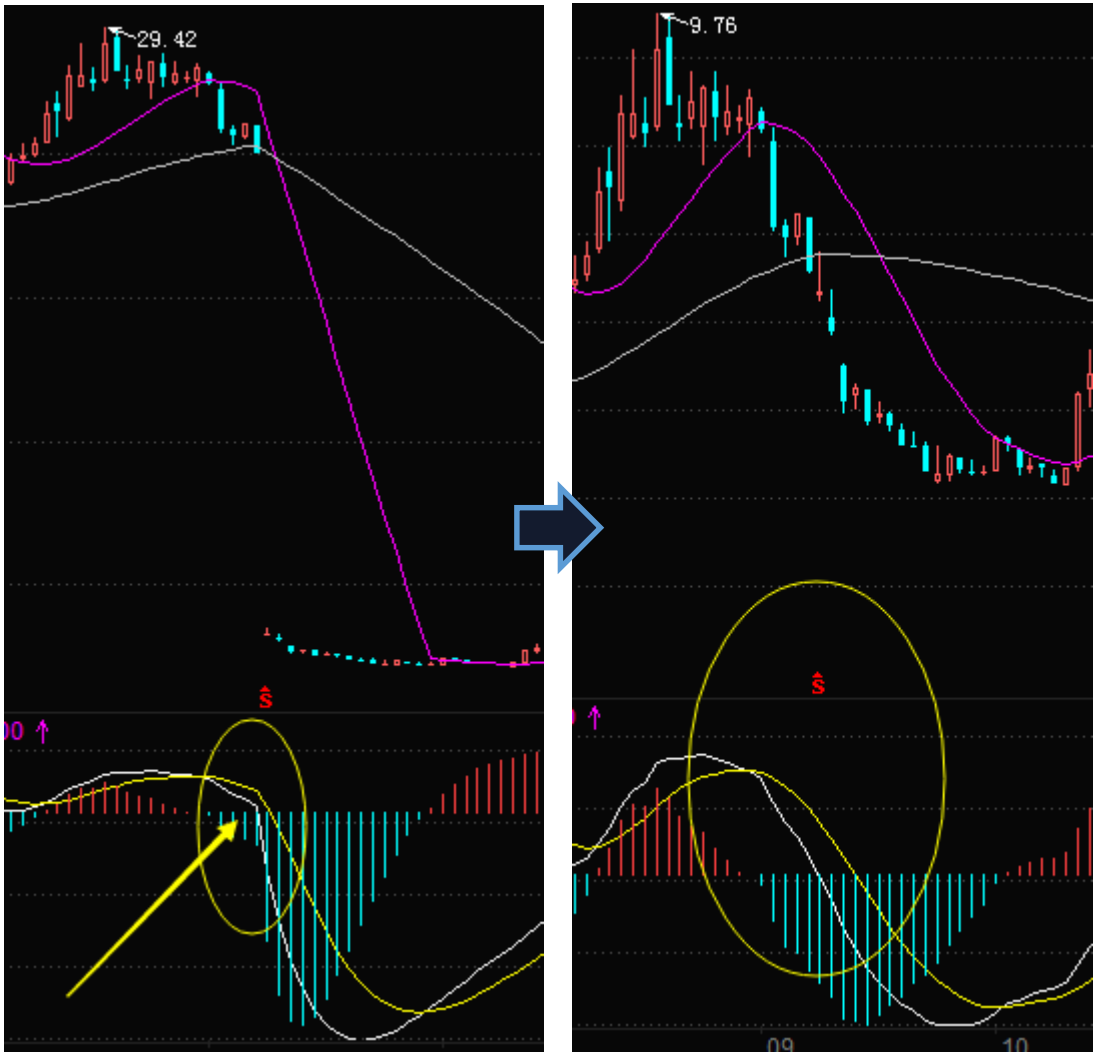
复权计算代码（例）

```
import pandas as pd
import pymongo

# read daily bars from MongoDB
cursor = daily_tbl.find({"code": “002157”, "trade_status": "交易",
                         "time": {"$gte": “2016-01-01”, "$lte": “2018-05-25”}},
                        {"_id": 0, “date": 1, "close": 1, "adjfactor": 1})
cursor.sort(“date", pymongo.ASCENDING)
df = pd.DataFrame.from_records(cursor)

lf = df.iloc[-1]["adjfactor"]
df[“hfq”] = df[“close”] * df[“adjfactor”]      # 后复权
df[“qfq”] = df[“close”] * df[“adjfactor”] / lf  # 前复权

df.index = df.pop(“date")
df.pop("adjfactor")
```

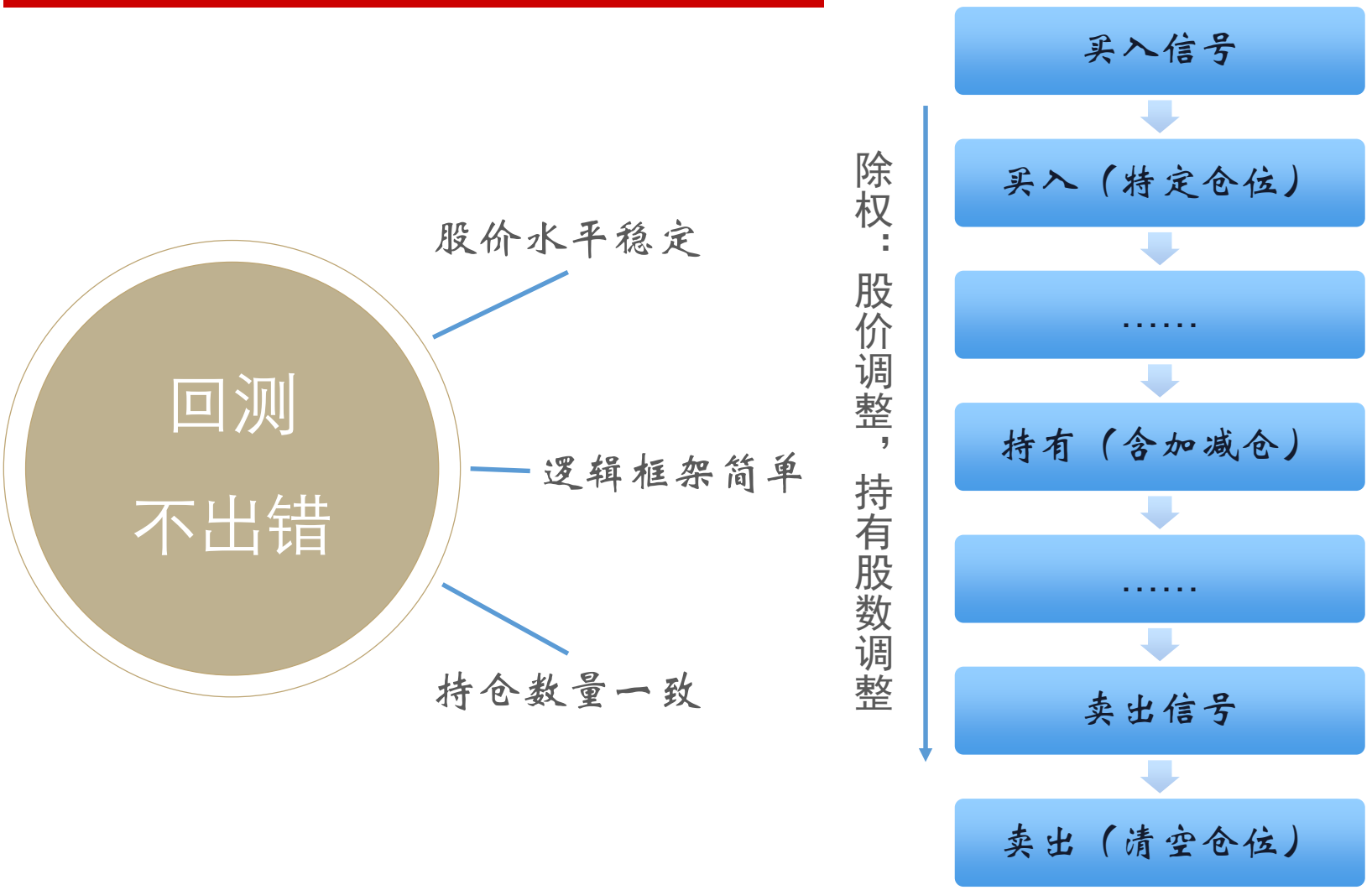
复权在量化中的作用（1）

指标
不失真

波动区间连续
无异常跳变点

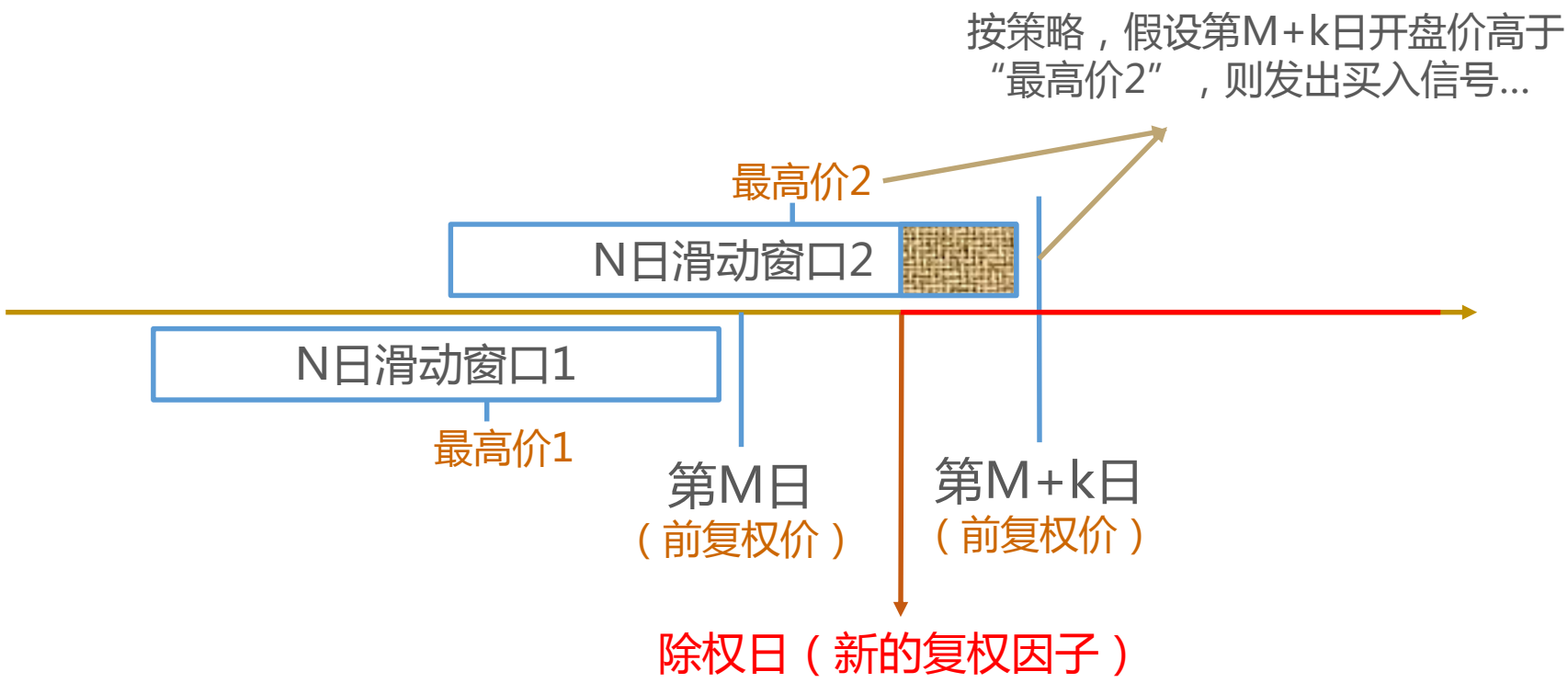


复权在量化中的作用（2）


回测
不出错
股价水平稳定
逻辑框架简单
持仓数量一致
除权：股价调整，持有股数调整
买入信号
买入（特定仓位）
……
持有（含加减仓）
……
卖出信号
卖出（清空仓位）

复权“陷阱”举例

- 应用场景（一个假想的突破型量化策略）：
  - 从前向后分段地读取行情数据，每次W天，并计算其中每天的前复权价
  - 同时，计算出每个交易日的前N日滑动窗口内（已复权）的最高价（N>>W）
  - 如果某个交易日当天开盘价高于前N日最高价，则发出买入信号



但不幸的是，这个信号可能是错误的（误报或漏报）！
因为“第M+k日开盘价”和“最高价2”那一天用的基准复权因子不是同一个，
做突破与否的比较时，计算并使用的复权价与真实的复权价之间有偏差…

解决方案

1. 读取回测区间的行情数据，并一次性地对所有交易日做前复权处理
2. 定点复权：逐日地对当天价格和前N日价格（共N+1天）重新做前复权处理
3. 采用后复权方案

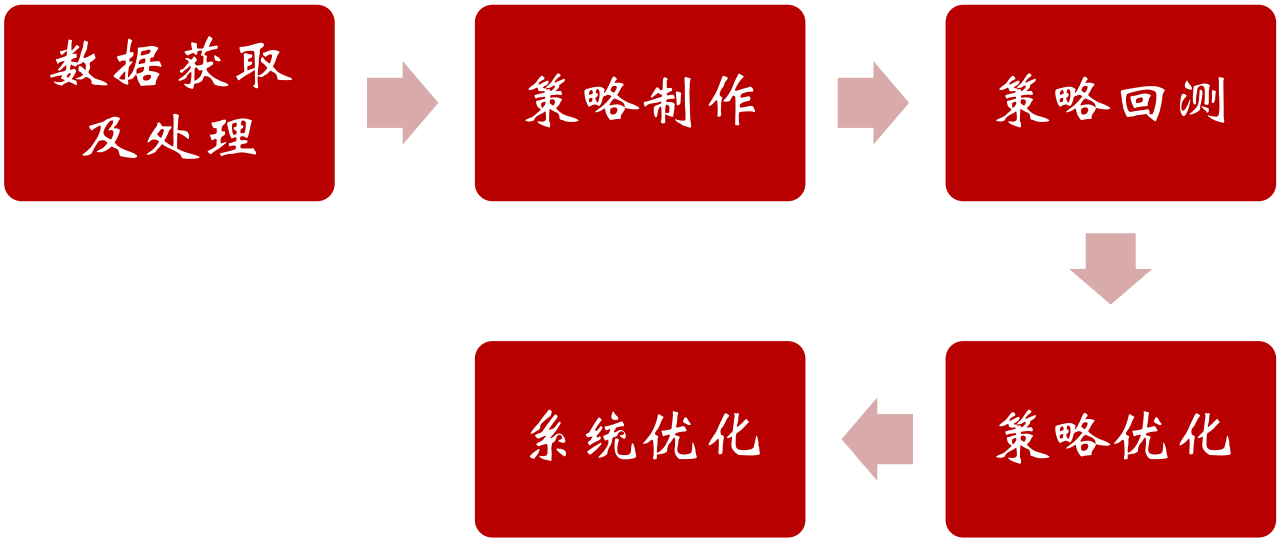
【思考题】

分析以上三种解决方案的利弊

不积跬步无以至千里

量化策略开发的过程

完整流程



低估值的股票池择时策略

- 股票池
  - 0 < PE_TTM < 30
  - 最多100只
  - 调整周期：20个交易日
- 择时
  - 买入：当日K线上穿10日均线
  - 卖出：
    - 当日K线下穿10日均线
    - 被调出股票池

数据—量化之血

行情和财报数据的抓取

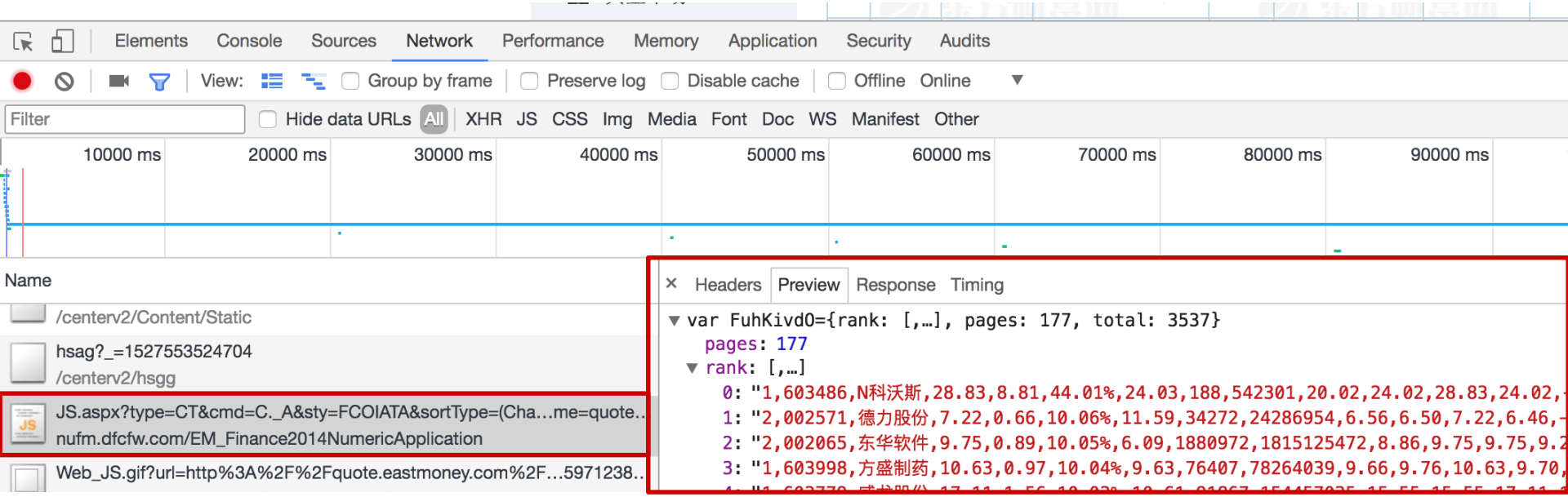
任务

- 抓取股票列表
- 抓取股票日K线
- 抓取股票的财报数据

股票列表页面分析

序号	代码	名称	相关链接	最新价	涨跌幅	涨跌额	成交量(手)	成交额	振幅	最高	最低	今开	昨收	五分钟涨跌	量比	换手率	市盈率	加自
1	603486	N科沃斯	股吧 资金流 数据	28.83	44.01%	8.81	188.00	54.23万	24.03%	28.83	24.02	24.02	20.02	0.00%	-	0.05%	125.33	
2	002571	德力股份	股吧 资金流 数据	7.22	10.06%	0.66	3.43万	2429万	11.59%	7.22	6.46	6.50	6.56	0.00%	2.03	1.35%	-	
3	002065	东华软件	股吧 资金流 数据	9.75	10.05%	0.89	188.1万	18.15亿	6.09%	9.75	9.21	9.75	8.86	0.00%	3.92	6.74%	56.89	
4	603998	方盛制药	股吧 资金流 数据	10.63	10.04%	0.97	7.64万	7826万	9.63%	10.63	9.70	9.76	9.66	0.00%	2.27	1.80%	49.01	
5	603779	威龙股份	股吧 资金流 数据	17.11	10.03%	1.56	9.19万	1.54亿	10.61%	17.11	15.46	15.55	15.55	0.00%	7.21	9.92%	52.35	
6	300209	天泽信息	股吧 资金流 数据	19.20	10.03%	1.75	3.62万	6876万	8.02%	19.20	17.80	17.83	17.45	0.00%	1.24	1.44%	194.13	
7	603099	长白山	股吧 资金流 数据	12.96	10.02%	1.18	9.70万	1.24亿	7.56%	12.96	12.07	12.07	11.78	0.00%	5.04	3.64%	-	
8	300682	朗新科技	股吧 资金流 数据	24.26	10.02%	2.21	13.92万	3.26亿	12.52%	24.26	21.50	21.98	22.05	0.00%	2.63	30.93%	-	
9	002461	珠江啤酒	股吧 资金流 数据	7.03	10.02%	0.64	45.77万	3.08亿	11.42%	7.03	6.30	6.41	6.39	0.00%	2.54	3.36%	217.84	

股票列表页面分析


Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
View:
Group by frame
Preserve log
Disable cache
Offline
Online
Filter
Hide data URLs
All
XHR
JS
CSS
Img
Media
Font
Doc
WS
Manifest
Other
10000 ms
20000 ms
30000 ms
40000 ms
50000 ms
60000 ms
70000 ms
80000 ms
90000 ms
Name
/centerv2/Content/Static
hsag?_=1527553524704
/centerv2/hsgg
JS.aspx?type=CT&cmd=C._A&sty=FCOIATA&sortType=(Cha...me=quote..
nufm.dfcfw.com/EM_Finance2014NumericApplication
Web_JS.gif?url=http%3A%2F%2Fquote.eastmoney.com%2F...5971238..
Headers
Preview
Response
Timing
var FuhKivd0={rank: [,…], pages: 177, total: 3537}
pages: 177
rank: [,…]
0: "1,603486,N科沃斯,28.83,8.81,44.01%,24.03,188,542301,20.02,24.02,28.83,24.02,
1: "2,002571,德力股份,7.22,0.66,10.06%,11.59,34272,24286954,6.56,6.50,7.22,6.46,
2: "2,002065,东华软件,9.75,0.89,10.05%,6.09,1880972,1815125472,8.86,9.75,9.75,9.2
3: "1,603998,方盛制药,10.63,0.97,10.04%,9.63,76407,78264039,9.66,9.76,10.63,9.70,

股票列表地址解析

http://nufm.dfcfw.com/EM_Finance2014NumericApplication/JS.aspx?

type=CT&cmd=C._A&sty=FCOIATA&

sortType=(ChangePercent)&sortRule=-1&

page=1&pageSize=20&

分页参数

js=var%20FuhKivdO={rank:[(x)],pages:(pc),total:(tot)}&

返回格式

token=7bc05d0d4c3c22ef9fca8c2a912d779c&jsName=quote_123

&_g=0.6286069159115 89&_=1527553524705

股票列表抓取地址构造

http://nufm.dfcfw.com/EM_Finance2014NumericApplication/JS.aspx?

type=CT&cmd=C._A&sty=FCOIATA&

sortType=(ChangePercent)&sortRule=-1&

page=1&pageSize=500&

增加每次返回的数目，减少请求次数

js={“stocks”:[(x)],”pages”:(pc)}&

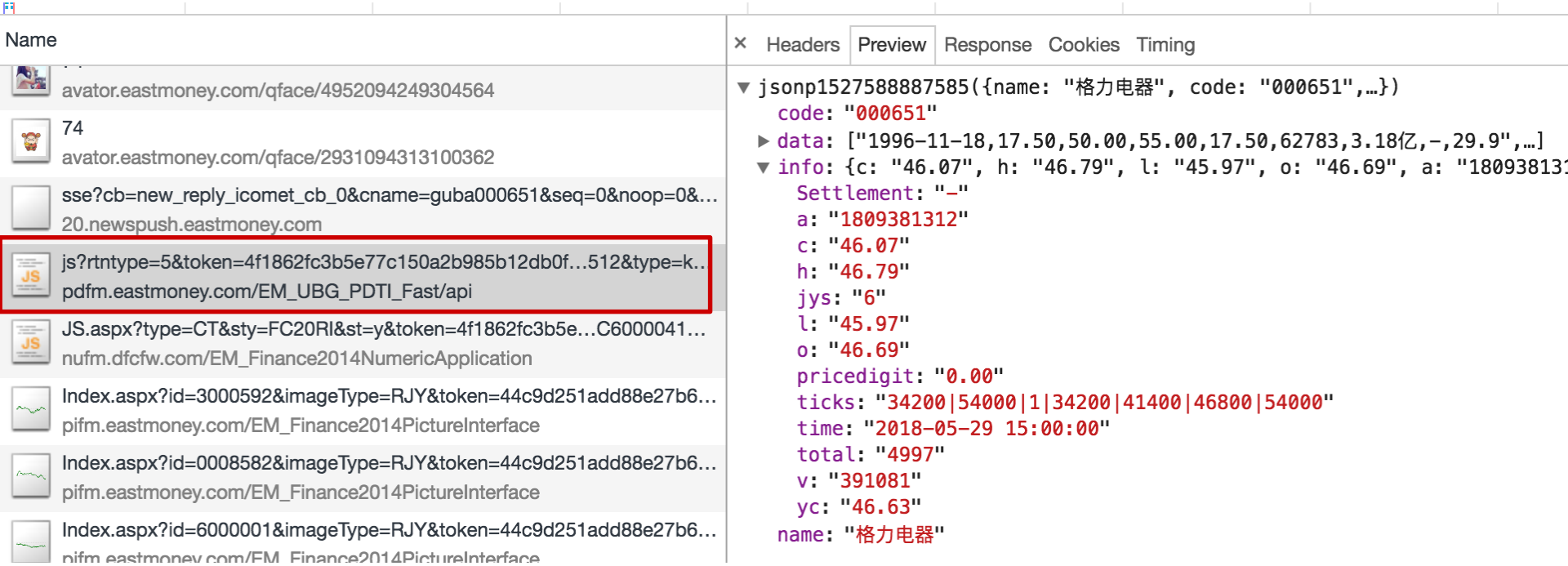
去掉前缀，只保留JSON格式部分

token=7bc05d0d4c3c22ef9fca8c2a912d779c&jsName=quote_123&_g=0.6286069159115 89&_=1527553524705

股票日K页面分析

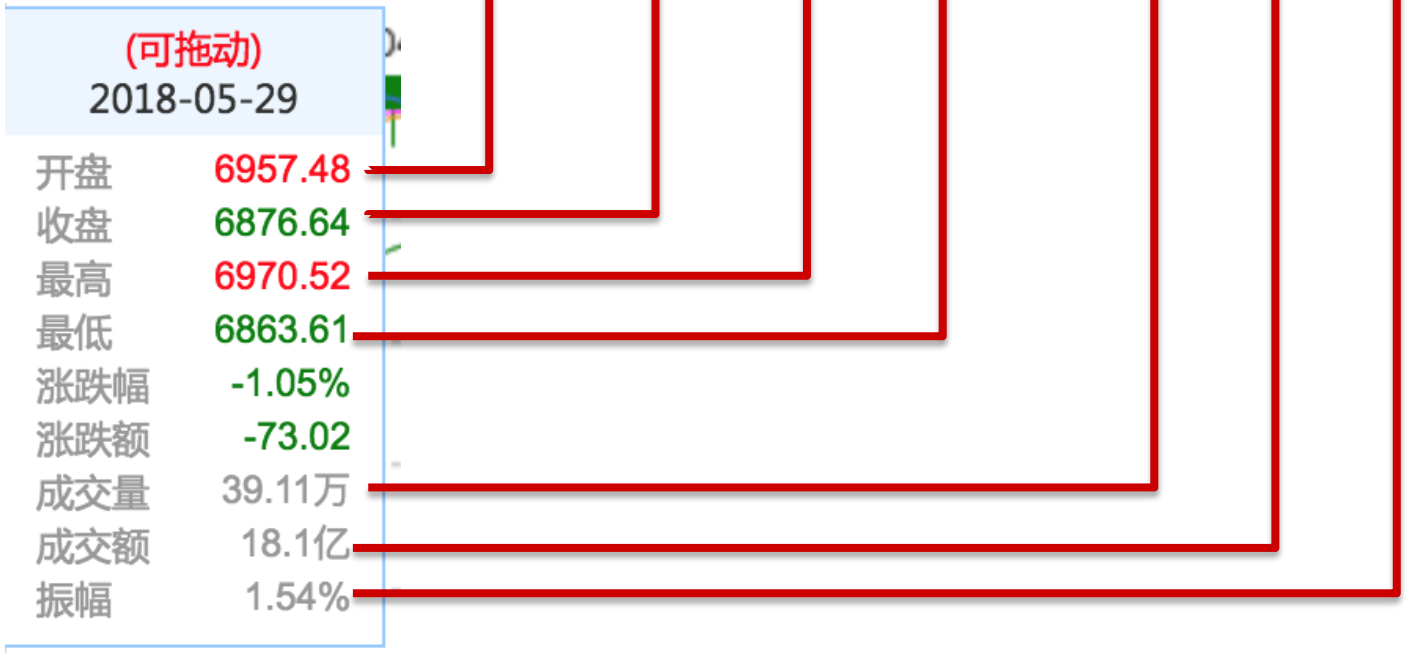

000651 格力电器
融
深股通
+自选
46.07
-0.56
-1.20%
2018-05-29 15:10:03
行情数据
最新 46.07
均价 46.27
成交量 39.11万手
今开 46.69
昨收 46.63
成交额 18.09亿
最高 46.79
最低 45.97
盘前
分时
日K
周K
月K
5分钟
15分钟
30分钟
60分钟
拉长K线
缩短K线
后复权
格力电器[000651]
MA5: 6874.56
MA10: 6983.69
MA20: 7006.64
MA60: 7195.18
7004.10
6617.17
7800.00
7600.00
7400.00
7200.00
7000.00
6800.00
6600.00
2018-04
2018-05
东方财富网
eastmoney.com
主图指标
均线
EXPMA
SAR
BOLL
BBI

股票日K页面分析


Name
avator.eastmoney.com/qface/4952094249304564
74
avator.eastmoney.com/qface/2931094313100362
sse?cb=new_reply_icomet_cb_0&cname=guba000651&seq=0&noop=0&...
20.newspush.eastmoney.com
js?rtntype=5&token=4f1862fc3b5e77c150a2b985b12db0f...512&type=k..
pdfm.eastmoney.com/EM_UBG_PDTI_Fast/api
JS.aspx?type=CT&sty=FC20RI&st=y&token=4f1862fc3b5e...C6000041...
nufm.dfcfw.com/EM_Finance2014NumericApplication
Index.aspx?id=3000592&imageType=RJY&token=44c9d251add88e27b6...
pifm.eastmoney.com/EM_Finance2014PictureInterface
Index.aspx?id=0008582&imageType=RJY&token=44c9d251add88e27b6...
pifm.eastmoney.com/EM_Finance2014PictureInterface
Index.aspx?id=6000001&imageType=RJY&token=44c9d251add88e27b6...
Headers
Preview
Response
Cookies
Timing
jsonp1527588887585({name: "格力电器", code: "000651",…})
code: "000651"
data: ["1996-11-18,17.50,50.00,55.00,17.50,62783,3.18亿,-,29.9",…]
info: {c: "46.07", h: "46.79", l: "45.97", o: "46.69", a: "18093813
Settlement: "-"
a: "1809381312"
c: "46.07"
h: "46.79"
jys: "6"
l: "45.97"
o: "46.69"
pricedigit: "0.00"
ticks: "34200|54000|1|34200|41400|46800|54000"
time: "2018-05-29 15:00:00"
total: "4997"
v: "391081"
yc: "46.63"
name: "格力电器"

股票日K结果分析

```
    "2018-05-23,6961.40,6896.20,6961.40,6880.56,399706,18.5亿,1.16%,0.67",
    "2018-05-24,6896.20,6816.67,6919.67,6807.54,451000,20.7亿,1.63%,0.76",
    "2018-05-25,6816.67,6833.62,6879.25,6802.32,336852,15.4亿,1.13%,0.56",
    "2018-05-28,6842.74,6949.66,6974.43,6811.45,501538,23.3亿,2.39%,0.84",
    "2018-05-29,6957.48,6876.64,6970.52,6863.61,391081,18.1亿,1.54%,0.66"
]
```



股票日K抓取地址分析

http://pdfm.eastmoney.com/EM_UBG_PDTI_Fast/api/js?rtntype=

5

&token=4f1862fc3b5e77c150a2b985b12db0fd&

id=0006512

&type=k&

authorityType=ba

&cb=jsonp1527588887585

股票日K抓取地址构造

http://pdfm.eastmoney.com/EM_UBG_PDTI_Fast/api/js?

rtntype=1　返回类型设为1，减少干扰

&token=4f1862fc3b5e77c150a2b985b12db0fd&

id=6000001　股票代码，后缀 1- 沪市，2 – 深市

&type=k

&authorityType=　复权类型，为空表示不复权, fa - 前复权，ba - 后复权

&cb=

```python
import urllib3
import json


def crawl_stock_list():
    """
    抓取股票代码列表
    """
    url = """http://nufm.dfcfw.com/EM_Finance2014NumericApplication/JS.aspx?
    type=CT&cmd=C._A&sty=FCOIATA&sortType=(ChangePercent)&sortRule=-1&
    page={0}&pageSize=500&js={"stocks":[(x)],"pages":(pc)}
    &token=7bc05d0d4c3c22ef9fca8c2a912d779c&jsName=quote_123&_g=0.628606915911589&_=1527553524705#"""

    stock_codes = []

    # 分页抓取，直到获取所有的股票代码
    page = 1
    while 1:
        # 抓取当前页数据
        response = crawl(page, url)
        # 转化为JSON
        data = json.loads(response, 'UTF-8')

        # 解析获取所有股票代码
        stocks = data['stocks']
        for stock in stocks:
            fields = stock.split(',')
            print('code: %s, name: %s' % (fields[1], fields[2]))
            stock_codes.append(fields[1])

        # 判断是否抓取到最后一页，如果已经到了最后一页，则结束股票列表的抓取
        pages = data['pages']
        print('Crawled%d/%d' % (page, pages))
        if page == pages:
            break

        page += 1

    return stocks
```

抓取股票列表

```python
def crawl_k_data():
    # 先获取所有的股票列表
    codes = crawl_stock_list()

    # 创建连接池
    conn_pool = urllib3.PoolManager()

    url = """http://pdfm.eastmoney.com/EM_UBG_PDTI_Fast/api/js?
    rtntype=1&token=4f1862fc3b5e77c150a2b985b12db0fd&id={0}&type=k&authorityType=&cb="""

    # 抓取所有股票
    for code in codes:
        response = None
        # 不同的股票设置不同的后缀
        if code[0:2] == '60':
            response = conn_pool.request('GET', url.replace('{0}', code + '1'))
        elif code[0:2] in ['30', '00']:
            response = conn_pool.request('GET', url.replace('{0}', code + '2'))

        # 如果获取到Response，则跳过股票
        if response is None:
            continue
```

抓取日K行情

后复权

```
# 解析抓取结果
hq_str = response.data.decode('UTF-8')[1:-1]

hq_lines = hq_str.split("\n")

# 获取每一个字段
for line in hq_lines:
    if len(line) > 0:
        # 逗号分隔
        fields = line.split(',')
        # 日期
        _date = fields[0]
        # 开盘价
        _open = float(fields[1])
        # 收盘价
        _close = float(fields[2])
        # 最高价
        _high = float(fields[3])
        # 最低价
        _low = float(fields[4])
        # 成交量，转化为股数
        _volume = int(fields[5]) * 100
        # 成交额，转化为元
        amount = float(fields[6][0:-1])
        amount_unit = fields[6][-1]
        if amount_unit == '亿':
            amount *= amount * 1e8
        elif amount_unit == '万':
            amount *= amount * 1e4

        print('date: %s, open: %5.2f, close: %5.2f, high: %5.2f,  low: %5.2f, amount: %10d, volume: %8d' %
         (_date, _open, _close, _high, _low, amount, _volume))
```

财务数据抓取

数据中心

数据全景图 >>

首页

资金流向

特色数据

新股数据

研究报告

年报季报

最新业绩报表

分红送配

2018中报预告

2018一季报

2018一季报快报

2018一季报预告

2018一季报预披露

亿嘉和(603666)业绩报表明细

个股业绩报表：输代码、名称或拼音 查询 查看最新

代码	名称	最新价	涨跌额	涨跌幅(%)	振幅(%)	成交量(手)	成交额(万)	昨收	今开	最高	最低	换手率(%)	量比	市盈率	加自选
603666	亿嘉和	-	-	-	-	-	-	34.46	-	-	-	-	-	13.28	

截止日期	每股收益(元)	每股收益(扣除)(元)	营业收入			净利润			每股净资产(元)	净资产收益率(%)	每股经营现金流量(元)	销售毛利率(%)	利润分配	股息率(%)	首次公告日期	最新公告日期
			营业收入(元)	同比增长(%)	季度环比增长(%)	净利润(元)	同比增长(%)	季度环比增长(%)								
2018 03-31	0.6487	-	7918.87万	2709.30	-	3414.25万	436.02	-	6.5363	-	0.2662	65.83	-	-	2018 05-29	2018 05-29
2017 12-31	2.6409	2.3563	3.74亿	42.70	-	1.39亿	153.37	-	5.8876	51.41	0.4498	67.22	-	-	2018 01-26	2018 05-29
2017 03-31	-0.1931	-	281.88万	-	-	-1016.09万	-	-	-	-	-1.3434	59.25	-	-	2018 05-29	2018 05-29
2016 12-31	1.069	1.3759	2.62亿	37.78	-	5485.72万	305.92	-	3.8167	38.23	1.6693	57.04	-	-	2017 03-31	2018 05-29
2015 12-31	0.3861	1.1464	1.90亿	73.26	-	1351.43万	39.68	-	1.8472	23.14	1.1706	51.99	-	-	2017 03-31	2018 05-29
2014 12-31	0.4837	0.4297	1.10亿	-	-	967.48万	-	-	0.3721	65.62	0.1401	28.12	-	-	2017 03-31	2017 03-31

财报抓取地址构造

http://dcfm.eastmoney.com//em_mutisvcexpandinterface/api/js/get

?

type=YJBB20_YJBB&

token=70f12f2f4f091e459a279469fe49eca5&

st=reportdate&sr=-1&

filter=(scode=600000) 股票代码

&p=1&ps=50& 分页参数

js={“pages”:(tp),”data”:(x)}

```
def crawl_finance_report():
    # 先获取所有的股票列表
    codes = crawl_stock_list()

    # 创建连接池
    conn_pool = urllib3.PoolManager()

    url = """http://dcfm.eastmoney.com//em_mutisvcexpandinterface/api/js/get?
    type=YJBB20_YJBB&token=70f12f2f4f091e459a279469fe49eca5&st=reportdate&
    sr=-1&filter=(scode={0})&p=1&ps=100&js={%22pages%22:(tp),%22data%22:%20(x)}"""

    cookie = """emstat_bc_emcount=21446959091031597218; pgv_pvi=8471522926; st_pvi=95785429701209; _ga=GA1.2.700565749.1
    user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.335

    for code in codes:
        response = conn_pool.request('GET', url.replace('{0}', code),
            headers={
            'Cookie': cookie,
            'User-Agent': user_agent})

        # 解析抓取结果
        result = json.loads(response.data.decode('UTF-8'), 'UTF-8')

        reports = result['data']

        for report in reports:
            # 报告期
            report_date = report['reportdate'][0:10]
            # 公告日期
            announced_date = report['latestnoticedate'][0:10]
            # 每股收益
            eps = report['basiceps']
```

休息一下

5分钟后回来

存的下，找得到

实用量化交易系统的存储解决方案

数据特点

```
"code" : "600021",
"time" : "2018-02-05",
"name" : "上海电力",
"open" : 8,
"close" : 8.13,
"high" : 8.18,
"low" : 7.93,
"pre_close" : 8.08,
"amount" : 25018841,
"volume" : 3098554,
"pct_chg" : 0.62,
"chg" : 0.05,
"trade_status" : "交易"
```

字段不固定
随时增加

```
db = MongoClient('mongodb://127.0.0.1:27017')['quant']
```

```
        # 更新的requests
        update_requests = []
        # 获取每一个字段
        for line in hq_lines:
            if len(line) > 0:
                # 逗号分隔
                fields = line.split(',')

                # 成交额，转化为元
                amount = float(fields[6][0:-1])
                amount_unit = fields[6][-1]
                if amount_unit == '亿':
                    amount *= amount * 1e8
                elif amount_unit == '万':
                    amount *= amount * 1e4
```

保存行情数据到MongoDB

```
# 构造保存用的document
doc = {
    # 股票代码
    'code': code,
    # 日期
    'time': fields[0],
    # 开盘价
    'open': float(fields[1]),
    # 收盘价
    'close': float(fields[2]),
    # 最高价
    'high': float(fields[3]),
    # 最低价
    'low': float(fields[4]),
    # 成交量，转化为股数
    'volume': int(fields[5]) * 100,
    # 成交额
    'amount': amount
}

# 创建更新的请求
update_requests.append(
    UpdateOne(
        {'code': code, 'time': fields[0]},
        {'$set': doc},
        upsert=True))

if len(update_requests) > 0:
    update_result = db.daily_k.bulk_write(update_requests, ordered=False)
    print('code: %s, modified: %4d, upserted: %4d' %
     (code, update_result.modified_count, update_result.upserted_count))
```

其他数据集（collection）


```
"code" : "603666",
"report_date" : "2018-03-31",
"announced_date" : "2018-05-29",
"eps" : 0.6487
```

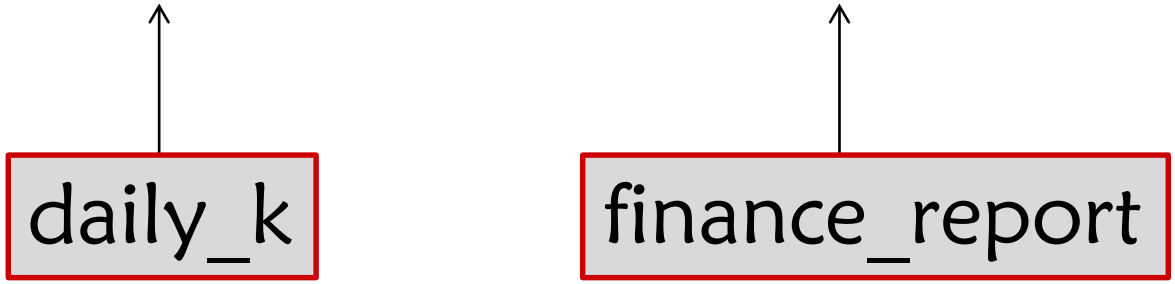

finance_report

后复权的行情，数据结构和daily_k相同，只是数值不同

daily_k_ba

计算滚动市盈率

PE_TTM = 股价/ 连续四个季度的每股收益之和



```python
def compute_pe_ttm(code, _date):
    """
    计算股票在某只的滚动市盈率
    """
    finance_report_cursor = db.finance_report.find(
        {'code': code, 'announced_date' : {'$lte': _date}},
        sort=[('announced_date', DESCENDING)],
        limit=4
        )

    finance_reports = [x for x in finance_report_cursor]

    # 如果数据不满四个季度，则不参与计算
    if len(finance_reports) < 4:
        return

    # 取得当日的股价
    daily = db.daily_k.find_one({'code': code, 'time': _date})
    if daily is None:
        return

    # 计算滚动市盈率并保存到daily_k中
    eps = 0
    for finance_report in finance_reports:
        eps += finance_report['eps']

    if eps != 0:
        pe_ttm = daily['close'] / eps

        db.daily.update_one(
            {'code': code, 'time': _date},
            {'$set': {'pe_ttm':  pe_ttm}})
```

未来函数

量化策略

实现一个低估值的股票池择时策略

策略回顾

- 股票池
  - 0 < PE_TTM < 30
  - 最多100只
  - 调整周期：20个交易日
- 择时
  - 买入：当日K线上传10日均线
  - 卖出：
    - 当日K线下穿10日均线
    - 被调出股票池

股票池

```
def stock_pool(begin_date, end_date):
    """
    股票池
    :param begin_date: 开始日期
    :param end_date: 结束日期
    :return: tuple, 所有调整日, 以及调整日和代码列表对应的dict
    """

    adjust_date_codes_dict = dict()

    # 获取指定时间区间的所有交易日
    time_cursor = db.daily_k.find(
        {'code': '000001.SH', 'time':{'$gte': begin_date, '$lte': end_date}},
        sort=[('time', ASCENDING)],
        projection={'time': True},
        batch_size=1000
        )
    all_dates = [x['time'] for x in time_cursor]

    # 上一期的所有股票代码
    last_phase_codes = []
    # 调整周期是20个交易日
    adjust_interval = 20
    # 所有的调整日
    all_adjust_dates = []
```

股票池

```
# 在调整日调整股票池
for _index in range(0, len(all_dates), adjust_interval) :
    # 保存调整日
    adjust_date = all_dates[_index]
    all_adjust_dates.append(adjust_date)

    # 查询出调整当日，0 < pe_ttm < 30，且非停牌的股票
    # 最重要的一点是，按照pe_ttm正序排列，只取前100只
    daily_cursor = db.daily_k.find(
        {'time': adjust_date, 'code': {'$regex': '[6|0|3]0\d{4}$'}, 'pe_ttm': {'$lt': 30, '$gt': 0}, 'trade_status': '交易'},
        sort=[('pe_ttm', ASCENDING)],
        projection={'code': True},
        limit=100
        )

    codes = [x['code'] for x in daily_cursor]

    # 本期股票列表
    this_phase_codes = []

    # 查询出上次股票池中正在停牌的股票
    if len(last_phase_codes) > 0:
        suspension_cursor = db.daily_k.find(
            {'code': {'$in': last_phase_codes}, 'time':adjust_date, 'trade_status': '停牌'},
            projection={'code': True}
            )
        suspension_codes = [x['code'] for x in suspension_cursor]

        # 保留股票池中正在停牌的股票
        this_phase_codes = suspension_codes
```

股票池

```
    # 用新的股票将剩余位置补齐
    this_phase_codes += codes[0 : 100 - len(this_phase_codes)]
    # 将本次股票设为下次运行的时的上次股票池
    last_phase_codes = this_phase_codes

    # 建立该调整日和股票列表的对应关系
    adjust_date_codes_dict[adjust_date] = this_phase_codes

# 返回结果
return all_adjust_dates, adjust_date_codes_dict
```

股票池检验

```
def find_out_stocks(last_phase_codes, this_phase_codes):
    """
    找到上期入选本期被调出的股票，这些股票将必须卖出
    :param last_phase_codes: 上期的股票列表
    :param this_phase_codes: 本期的股票列表
    :return: 被调出的股票列表
    """
    out_stocks = []

    for code in last_phase_codes:
        if code not in this_phase_codes:
            out_stocks.append(code)

    return out_stocks
```

股票池检验

```
def evaluate_stock_pool():
    """
    对股票池做一个简单的评价
    """
    # 设定评测周期
    adjust_dates, codes_dict = stock_pool('2015-01-01', '2018-05-25')

    # 用DataFrame保存收益
    df_profit = pd.DataFrame(columns=['profit', 'hs300'])

    df_profit.loc[adjust_dates[0]] = {'profit': 0, 'hs300': 0}

    hs300_begin_value = db.daily.find_one({'code': '000300.SH', 'time': adjust_dates[0]})['close']

    # 通过净值计算累计收益
    net_value = 1
```

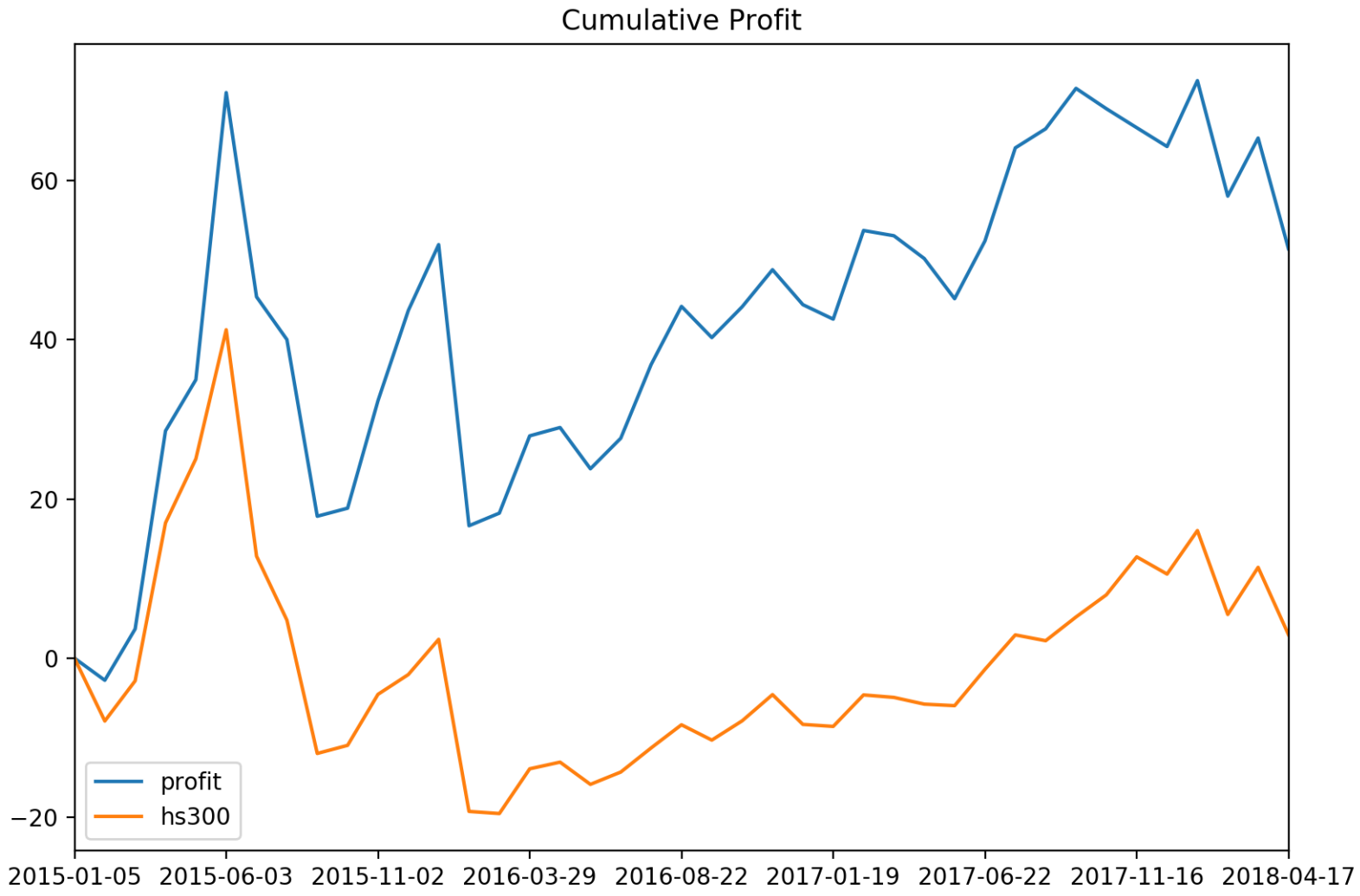
股票池检验

```
for _index in range(1, len(adjust_dates) - 1):
    last_adjust_date = adjust_dates[_index - 1]
    current_adjust_date = adjust_dates[_index]
    # 获取上一期的股票池
    codes = codes_dict[last_adjust_date]

    # 构建股票代码和后复权买入价格的股票
    code_buy_close_dict = dict()
    buy_daily_cursor = db.daily_k_ba.find(
        {'code': {'$in': codes}, 'time': last_adjust_date},
        projection={'close':True, 'code': True}
        )

    for buy_daily in buy_daily_cursor:
        code = buy_daily['code']
        code_buy_close_dict[code] = buy_daily['close']

    # 获取到期的股价
    sell_daily_cursor = db.daily_k_ba.find(
        {'code': {'$in': codes}, 'time': current_adjust_date},
        projection={'close':True, 'code': True}
        )
```

股票池检验

```
    # 计算单期收益
    profit_sum = 0
    for sell_daily in sell_daily_cursor:
        code = sell_daily['code']

        buy_close = code_buy_close_dict[code]
        sell_close = sell_daily['close'] * sell_daily['adjfactor']

        profit_sum += (sell_close - buy_close)/buy_close

    profit = round(profit_sum/len(codes), 4)

    hs300_close = db.daily.find_one({'code': '000300.SH', 'time': current_adjust_date})['close']

    # 计算净值和累积收益
    net_value = net_value * (1 + profit)
    df_profit.loc[current_adjust_date] = {
    'profit': round((net_value - 1) * 100, 4),
    'hs300': round((hs300_close - hs300_begin_value) * 100/hs300_begin_value, 4)}

# 绘制曲线
df_profit.plot(title='Stock Pool', kind='line')
# 显示图像
plt.show()
```

股票池检验


Cumulative Profit
60
40
20
0
−20
profit
hs300
2015-01-05
2015-06-03
2015-11-02
2016-03-29
2016-08-22
2017-01-19
2017-06-22
2017-11-16
2018-04-17

```python
def compare_close_2_ma_10(dailies):
    """
    比较当前的收盘价和MA10的关系
    :param dailies: 日线列表，10个元素，最后一个是当前交易日
    :return: 0 相等，1 大于， -1 小于，None 结果未知
    """
    current_daily = dailies[9]
    close_sum = 0
    count = 0
    for daily in dailies:
        # 10天当中，只要有一天停牌则返回False
        if daily['trade_status'] == '停牌':
            return None

        # 用后复权累计
        close_sum += daily['close']
        count += 1

    if count < 10:
        return None

    # 计算MA10
    ma_10 = close_sum/10

    # 判断收盘价和MA10的大小
    post_adjuested_close = current_daily['close']
    differ = post_adjuested_close - ma_10
    if differ > 0:
        return 1
    elif differ < 0 :
        return -1
    else:
        return 0
```

复权！！！

```python
def is_k_up_break_ma10(code, _date):
    """
    判断某只股票在某日是否满足K线上穿10日均线

    :param code: 股票代码
    :param _date: 日期
    :return: True/False
    """

    # 如果股票当日停牌或者是下跌，则返回False
    current_daily = db.daily.find_one(
        {'code': code, 'time': _date, 'trade_status': '交易'})
    if current_daily is None or current_daily['pct_chg'] < 0:
        return False

    # 计算MA10
    daily_cursor = db.daily.find(
        {'code': code, 'time': {'$lte': _date}},
        sort=[('time', DESCENDING)],
        limit = 11,
        projection={'close': True, 'adjfactor': True, 'trade_status': True}
        )

    dailies = [x for x in daily_cursor]

    if len(dailies) < 11:
        return False

    dailies.reverse()

    last_close_2_last_ma10 = compare_close_2_ma_10(dailies[0:10])
    current_close_2_current_ma10 = compare_close_2_ma_10(dailies[1:])

    if last_close_2_last_ma10 is None or current_close_2_current_ma10 is None:
        return False

    # 判断收盘价和MA10的大小
    return last_close_2_last_ma10 <= 0 and current_close_2_current_ma10 == 1
```

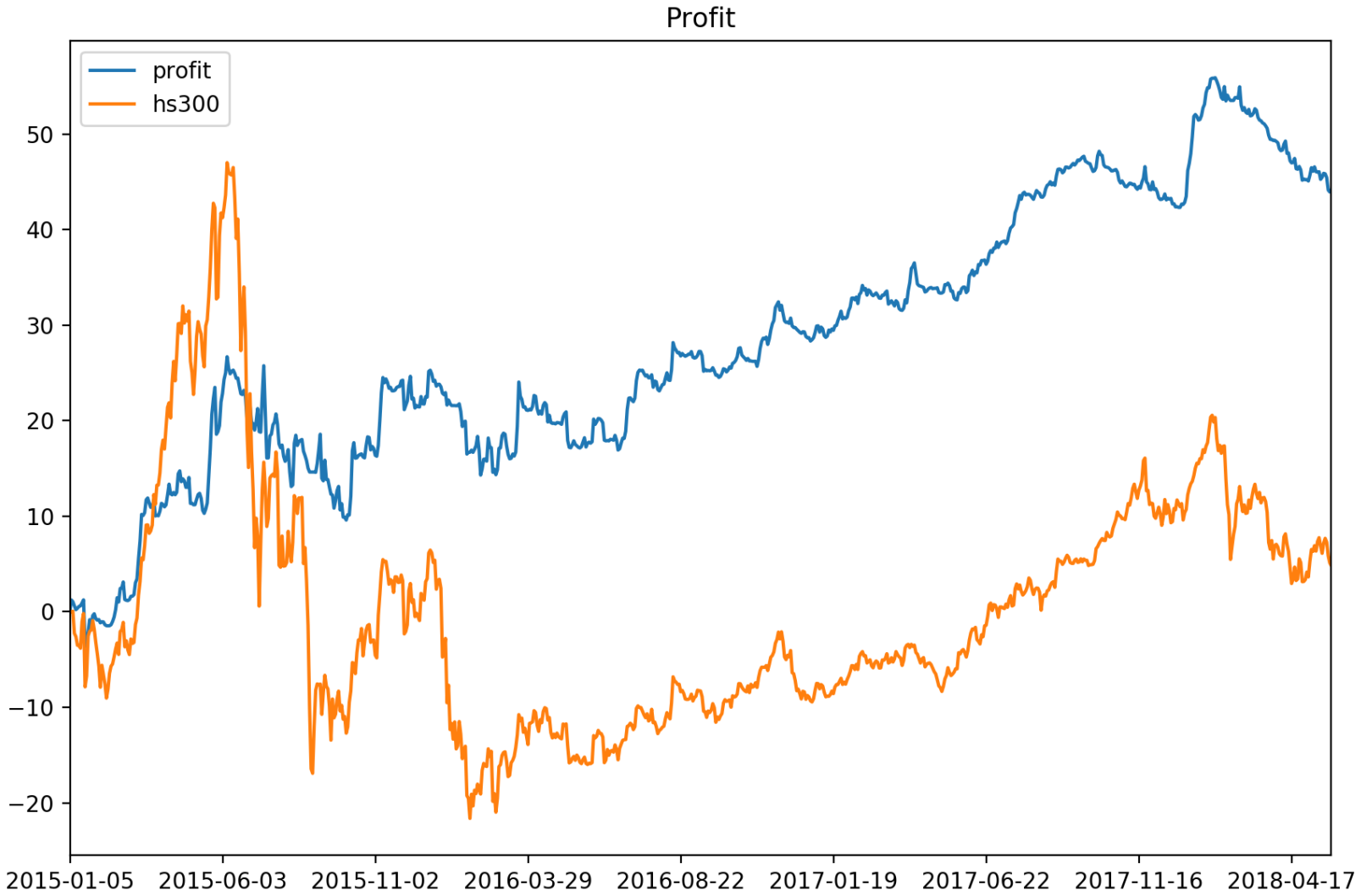
买入信号

```python
def is_k_down_break_ma10(code, _date):
    """
    判断某只股票在某日是否满足K线下穿10日均线

    :param code: 股票代码
    :param _date: 日期
    :return: True/False
    """

    # 如果股票当日停牌或者是下跌，则返回False
    current_daily = db.daily.find_one(
        {'code': code, 'time': _date, 'trade_status': '交易'})
    if current_daily is None or current_daily['pct_chg'] > 0:
        return False

    # 计算MA10
    daily_cursor = db.daily.find(
        {'code': code, 'time': {'$lte': _date}},
        sort=[('time', DESCENDING)],
        limit = 11,
        projection={'close': True, 'adjfactor': True, 'trade_status': True}
        )

    dailies = [x for x in daily_cursor]

    if len(dailies) < 11:
        return False

    dailies.reverse()

    last_close_2_last_ma10 = compare_close_2_ma_10(dailies[0:10])
    current_close_2_current_ma10 = compare_close_2_ma_10(dailies[1:])

    if last_close_2_last_ma10 is None or current_close_2_current_ma10 is None:
        return False

    # 判断收盘价和MA10的大小
    return last_close_2_last_ma10 >= 0 and current_close_2_current_ma10 == -1
```

卖出信号

卖出信号

回测结果


Profit
profit
hs300
50
40
30
20
10
0
−10
−20
2015-01-05
2015-06-03
2015-11-02
2016-03-29
2016-08-22
2017-01-19
2017-06-22
2017-11-16
2018-04-17

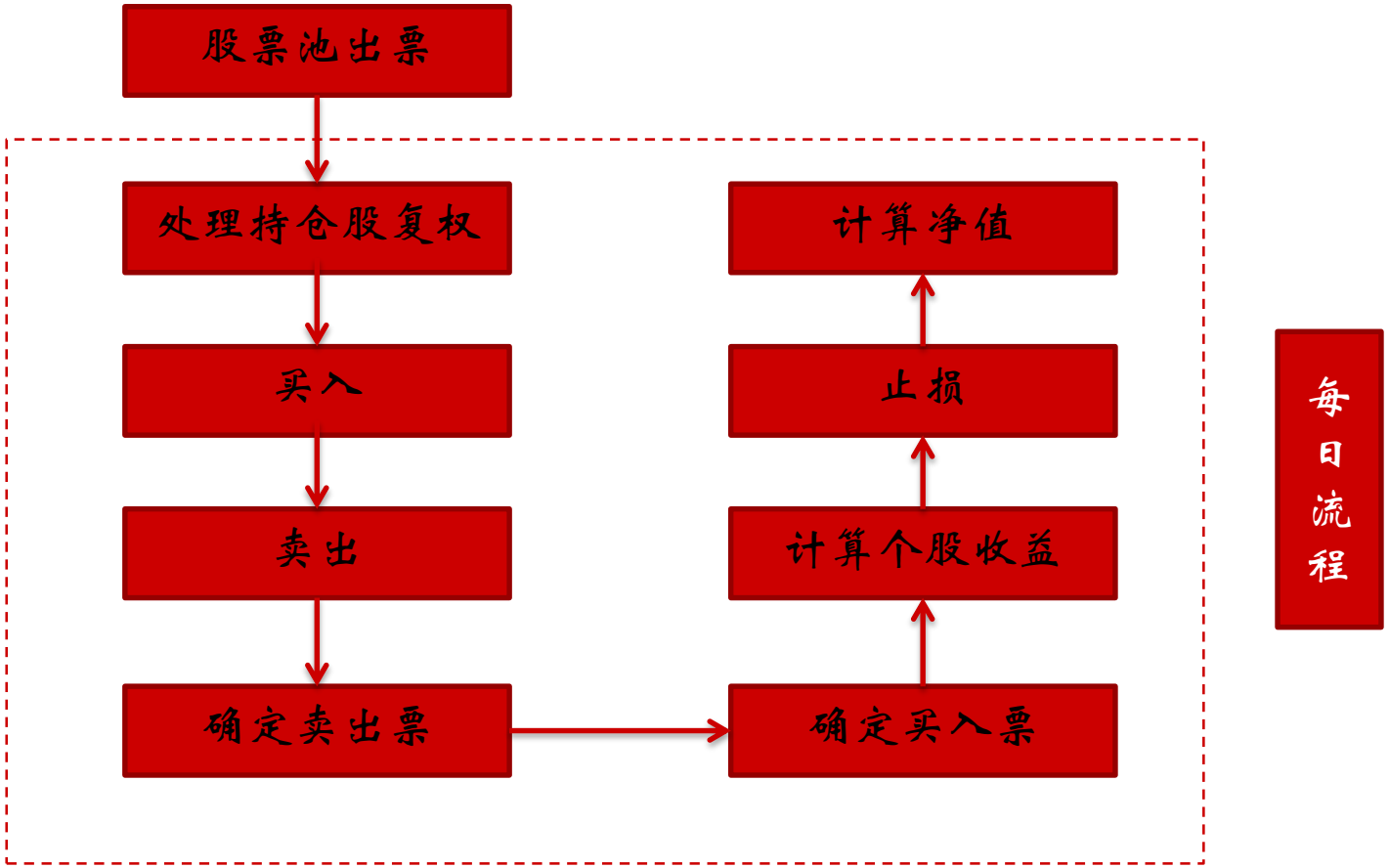
看效果

实现回测

刚才策略遗漏的东西

- 总资金
  - 1000万元
- 头寸分配
  - 均分
  - 每支最多20万
- 止损
  - 总亏损超过2%
  - 单日亏损超过1%

股票池策略的回测基本流程



初始化

```
def backtest(begin_date, end_date):
    """
    回测逻辑
    """

    # 初始资金
    total_capital = 1E7
    # 初始现金
    cash = 1E7
    # 总市值
    total_value = 0
    # 单只股票的可分配头寸
    single_position = 2E5

    df_profit = pd.DataFrame(columns=['profit', 'hs300'])

    # 获取回测内的所有交易日
    all_dates_cursor = db.daily_k.find(
        {'code':'000001.SH', 'time': {'$lte': end_date, '$gte': begin_date}},
        sort=[('time', ASCENDING)],
        projection={'time':True},
        batch_size=1000)

    all_dates = [x['time'] for x in all_dates_cursor]
```

处理持仓股的复权问题

```
# 处理复权
if last_date is not None and len(before_sell_holding_codes) > 0:
    last_daily_cursor = db.daily.find(
        {'code' :{'$in': before_sell_holding_codes}, 'time': last_date},
        projection={'code':True, 'adjfactor': True})

    code_last_adjactor_dict= dict()
    for last_daily in last_daily_cursor:
        code_last_adjactor_dict[last_daily['code']] = last_daily['adjfactor']

    current_daily_cursor = db.daily.find(
        {'code' :{'$in': before_sell_holding_codes}, 'time': _date},
        projection={'code':True, 'adjfactor': True})

    for current_daily in current_daily_cursor:
        current_adjfactor = current_daily['adjfactor']
        code = current_daily['code']
        before_volume = holding_code_dict[code]['volume']
        last_adjfactor = code_last_adjactor_dict[code]
        after_volume = int(before_volume * ( current_adjfactor / last_adjfactor))
        holding_code_dict[code]['volume'] = after_volume
        print('持仓量调整: %s, %6d, %10.6f, %6d, %10.6f' %
            (code, before_volume, last_adjfactor, after_volume, current_adjfactor))

# 买入
```

daily是添加了复权因子的daily_k

卖出

```
# 卖出
if len(to_be_sold_codes) > 0:
    sell_daily_cursor = db.daily.find(
        {'code': {'$in': list(to_be_sold_codes)}, 'time': _date, 'trade_status': '交易'},
        projection={'open': True, 'code': True}
        )


    for sell_daily in sell_daily_cursor:
        code = sell_daily['code']
        if code in before_sell_holding_codes:
            holding_stock = holding_code_dict[code]
            holding_volume = holding_stock['volume']
            sell_price = sell_daily['open']
            sell_amount = holding_volume * sell_price
            cash += sell_amount

            cost = holding_stock['cost']
            single_profit = (sell_amount - cost) * 100/ cost
            print('卖出 %s, %6d, %6.2f, %8.2f, %4.2f' %
                (code, holding_volume, sell_price, sell_amount, single_profit))

            del holding_code_dict[code]
            to_be_sold_codes.remove(code)

print('卖出后，现金: %10.2f' % cash)
```

买入

```
# 买入
if len(to_be_bought_codes) > 0:
    buy_daily_cursor = db.daily.find(
        {'code': {'$in': list(to_be_bought_codes)}, 'time': _date, 'trade_status': '交易'},
        projection={'code': True, 'open': True}
        )

    for buy_daily in buy_daily_cursor:
        if cash > single_position:
            buy_price = buy_daily['open']
            code = buy_daily['code']
            volume = int(int(single_position/buy_price)/100) * 100
            buy_amount = buy_price * volume
            cash -= buy_amount
            holding_code_dict[code] = {
            'volume': volume,
            'cost': buy_amount,
            'last_value': buy_amount}

            print('买入 %s, %6d, %6.2f, %8.2f' % (code, volume, buy_price, buy_amount))

print('买入后，现金: %10.2f' % cash)
```

找到第二天要交易的股票

```
# 持仓股代码列表
holding_codes = list(holding_code_dict.keys())
# 如果调整日，则获取新一期的股票列表
if _date in adjust_dates:
    # 暂存为上期的日期
    if this_phase_codes is not None:
        last_phase_codes = this_phase_codes
    this_phase_codes = date_codes_dict[_date]

    # 找到所有调出股票代码，在第二日开盘时卖出
    if last_phase_codes is not None:
        out_codes = find_out_stocks(last_phase_codes, this_phase_codes)
        for out_code in out_codes:
            if out_code in holding_code_dict:
                to_be_sold_codes.add(out_code)

# 检查是否有需要第二天卖出的股票
for holding_code in holding_codes:
    if is_k_down_break_ma10(holding_code, _date):
        to_be_sold_codes.add(holding_code)

# 检查是否有需要第二天买入的股票
to_be_bought_codes.clear()
if this_phase_codes is not None:
    for _code in this_phase_codes:
        if _code not in holding_codes and is_k_up_break_ma10(_code, _date):
            to_be_bought_codes.add(_code)
```

计算净值

```
# 计算总资产
total_value = 0
holding_daily_cursor = db.daily.find(
    {'code': {'$in': holding_codes}, 'time':_date},
    projection={'close': True, 'code': True}
    )
for holding_daily in holding_daily_cursor:
    code = holding_daily['code']
    holding_stock = holding_code_dict[code]
    value = holding_daily['close'] * holding_stock['volume']
    total_value += value

    profit = (value - holding_stock['cost']) * 100/holding_stock['cost']
    one_day_profit = (value - holding_stock['last_value']) * 100/holding_stock['last_value']

    holding_stock['last_value'] = value
    # 如果亏损大于2%，则止损
    if profit <= -2 or one_day_profit <= -1:
        to_be_sold_codes.add(code)
    print('持仓: %s, %10.2f, %4.2f, %4.2f' %
     (code, value, profit, one_day_profit))

total_capital = total_value + cash

hs300_current_value = db.daily.find_one(
{'code': '000300.SH', 'time':_date},
projection={'close': True})['close']

print('收盘后，现金: %10.2f, 总资产: %10.2f' % (cash, total_capital))
last_date = _date
df_profit.loc[_date] = {
'profit': round(100 * (total_capital - 1e7)/1e7, 2),
'hs300': round(100 * (hs300_current_value - hs300_begin_value)/hs300_begin_value, 2)
}
```

最大回撤

```
"""
计算最大回撤
"""
def compute_drawdown(profits):
    # 最大回撤初始值设为0
    max_drawdown = 0
    size = len(profits)
    index = 0
    # 双层循环找出最大回撤
    for profit in profits:
        for sub_profit in profits[index:]:
            drawdown = 1 - sub_profit/profit
            if drawdown > max_drawdown:
                max_drawdown = drawdown

        index += 1

    print(max_drawdown)
```

$$\text{Sharpe Ratio} = \frac{E(R_P) - R_f}{\sigma_P}$$

```
"""
计算夏普比率
"""
def compute_sharpe_ratio(net_values):
    # 总交易日数
    trading_days = len(net_values)
    # 所有收益的DataFrame
    profit_df = pd.DataFrame(columns={'profit'})
    # 收益之后，初始化为第一天的收益
    profit_df.loc[0] = {'profit': round((net_values[0]-1) * 100, 2)}
    # 计算每天的收益
    for index in range(1, trading_days):
        # 计算每日的收益变化
        profit = (net_values[index] - net_values[index - 1])/net_values[index -1]
        profit = round(profit * 100, 2)
        profit_df.loc[index] = {'profit': profit}

    # 计算标准差
    profit_std = pow(profit_df.var()['profit']/245, 1/2)

    #年化收益
    annual_profit = compute_annual_profit(trading_days, net_values[-1])

    # 夏普比率
    sharpe_ratio = (annual_profit - 4.75)/profit_std
```

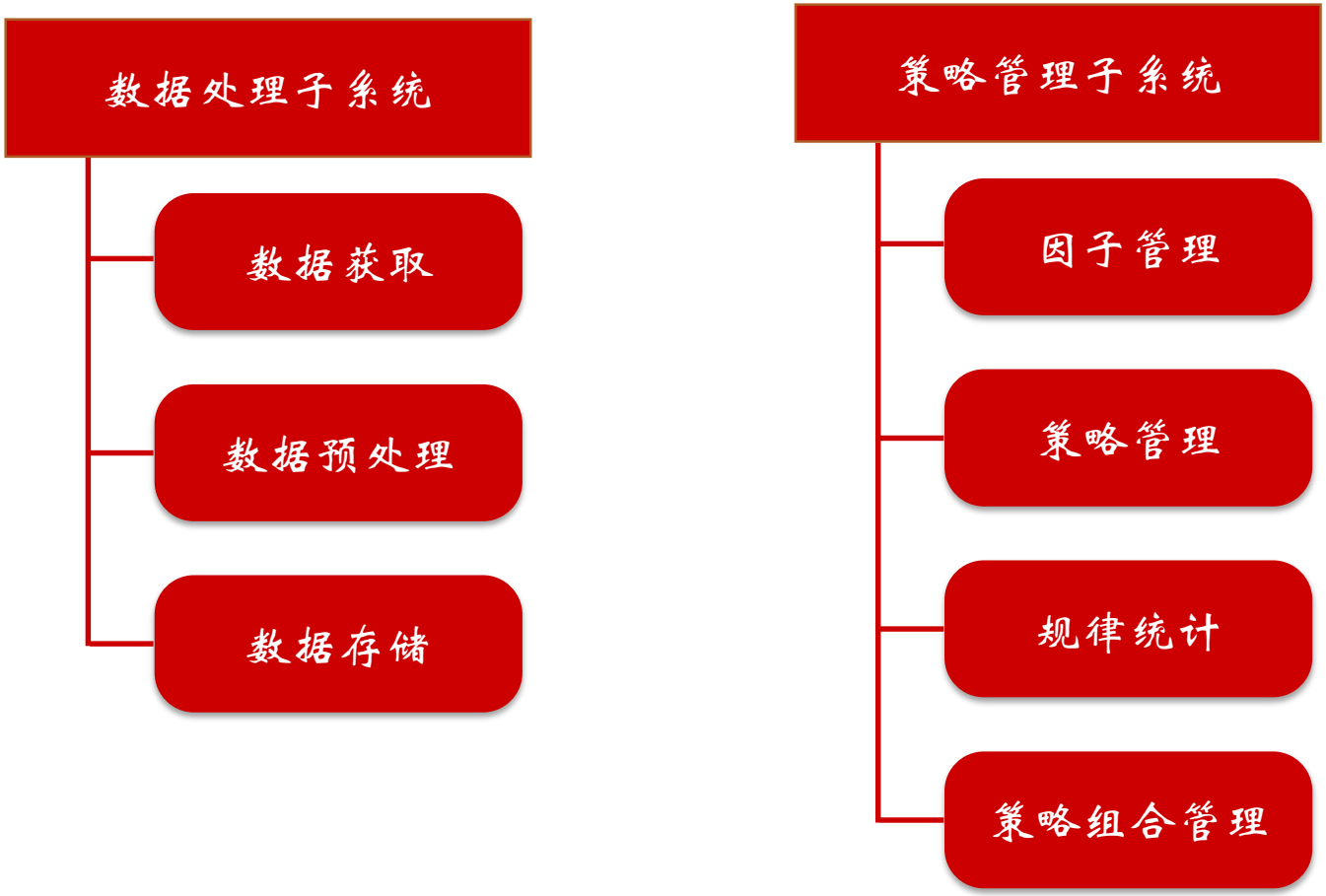
更多细节

- 费用
  - 手续费
  - 滑点
- 交易
  - 集合竞价
  - 涨停/跌停
- 更多统计
  - 资金利用率
  - 仓位占比
  - 盈亏比

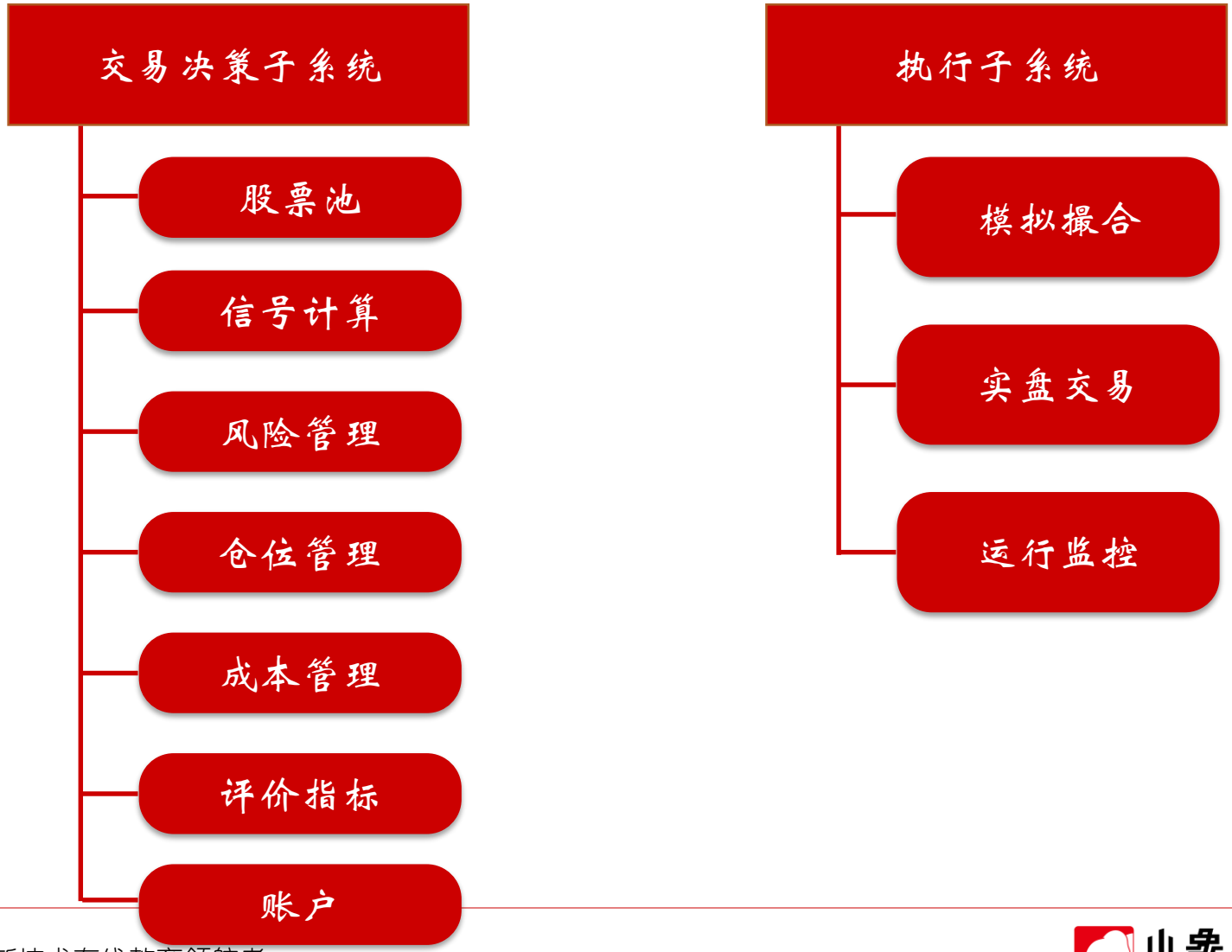
好用才是王道

量化交易系统的优化

模块化



模块化



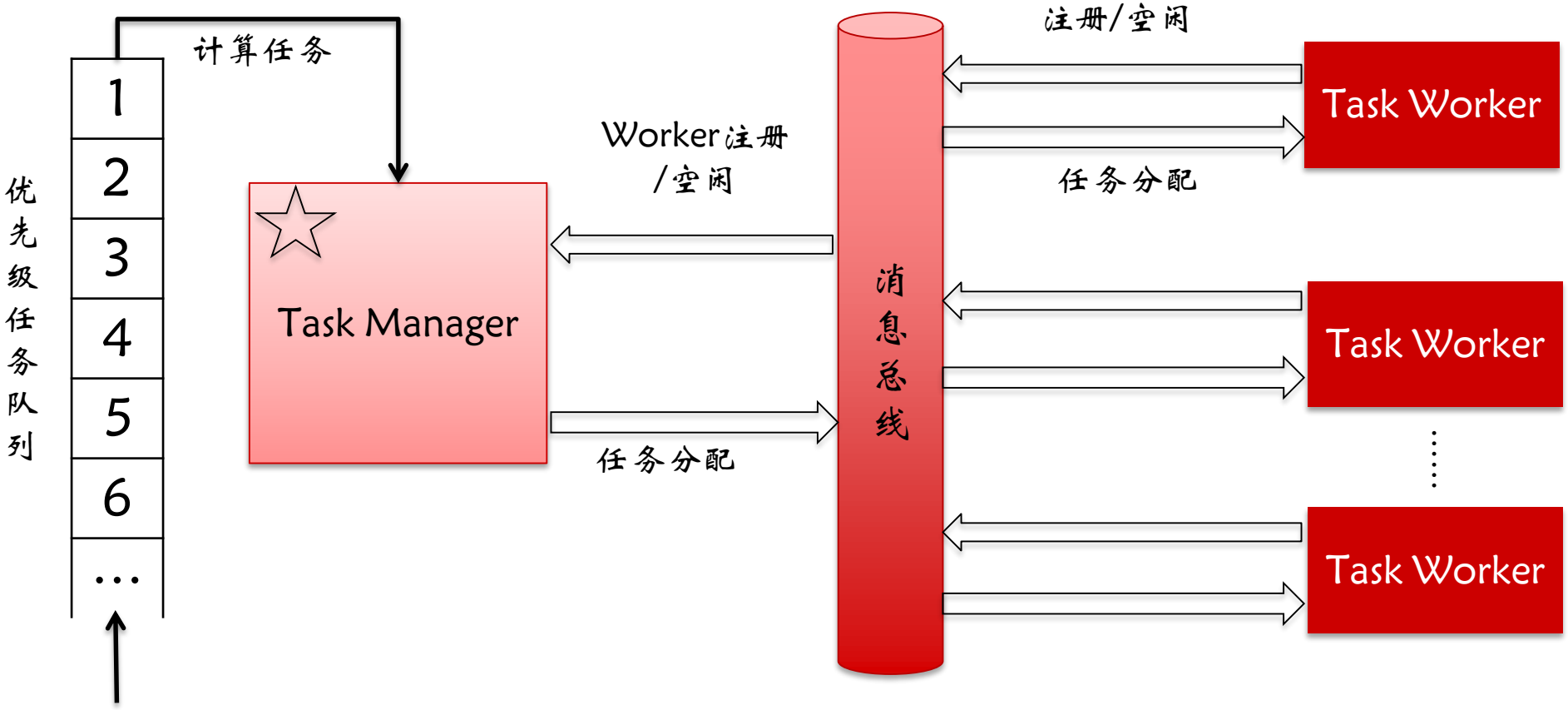
因子计算任务的优化

- 系统需求
  - 每日增量计算当日所有因子
  - 支持历史数据提取
  - 支持单个或者多个因子的历史数据提取
  - 历史数据的提取任务不能阻碍当日提取

因子计算任务的优化

- 关键技术
  - 优先级任务队列
  - 任务的Manager和Worker的角色管理
  - Worker的状态管理

因子计算任务的优化


计算任务
1
2
3
4
5
6
…
优先级任务队列
Task Manager
Worker注册/空闲
任务分配
消息总线
注册/空闲
任务分配
Task Worker
Task Worker
Task Worker

消息驱动的被动式系统性能优化

- 被动式系统特点
  - 模块依赖
    - 模块A的执行依赖模块B的执行状态或者输出
  - 模块众多，插拔式结构
    - 有些实验性质的模块，在正式并入系统之前也需要和线上的模块一样运行，通过验证，也可以随时上线和下线
  - 要求时延低
  - 从交易所接受的行情数据会被多个子系统使用，包括实盘模块

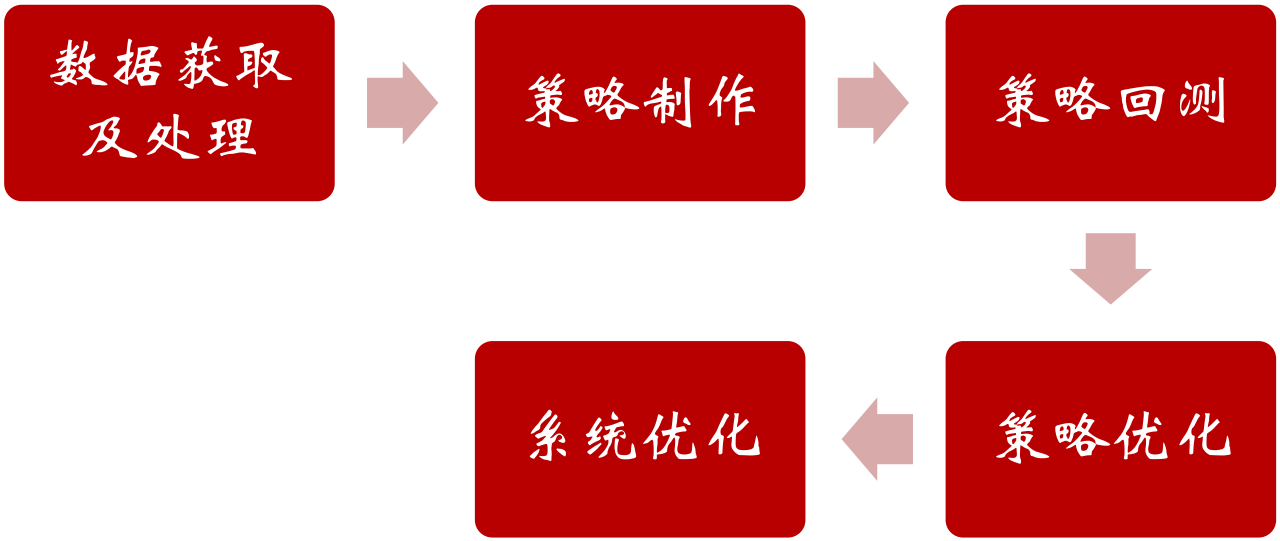
消息驱动的被动式系统性能优化

- 系统间通信机制
  - 主动调用机制
    - 模块A需要模块B帮助完成某项任务，就主动调用模块B
  - 发布订阅机制
    - 典型的生产者-消费者模式
    - 模块B完成了某件事，发布一个消息，所有订阅了这个消息的模块就可以基于这个消息执行自己的任务

消息驱动的被动式系统性能优化

- Kafka
  - 分布式、延迟低、持久化
- 在交易系统中使用场景
  - 行情数据分发
  - 解决不同任务模块之间的依赖
- 需要注意的问题
  - 硬盘空间的使用量
  - 客户端的offset回跳

总结



问答互动

在所报课的课程页面，
1、点击“全部问题”显示本课程所有学员提问的问题。
2、点击“提问”即可向该课程的老师和助教提问问题。

联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**

THANKS