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

第二课

上手搭建最简单的量化交易系统

量化交易实战：

策略编写与系统搭建

内容介绍

- 量化交易系统的处理流程
- 开发环境准备
- 获取行情和财报数据：API和爬虫
- 数据定时增量入库
- 实现一个低PE的股票池
- 开发一组交易信号
- 实现策略回测

目标

实现从TuShare和东财抓取行情和财务数据，研发低PE股票池策略并进行回测的完整流程

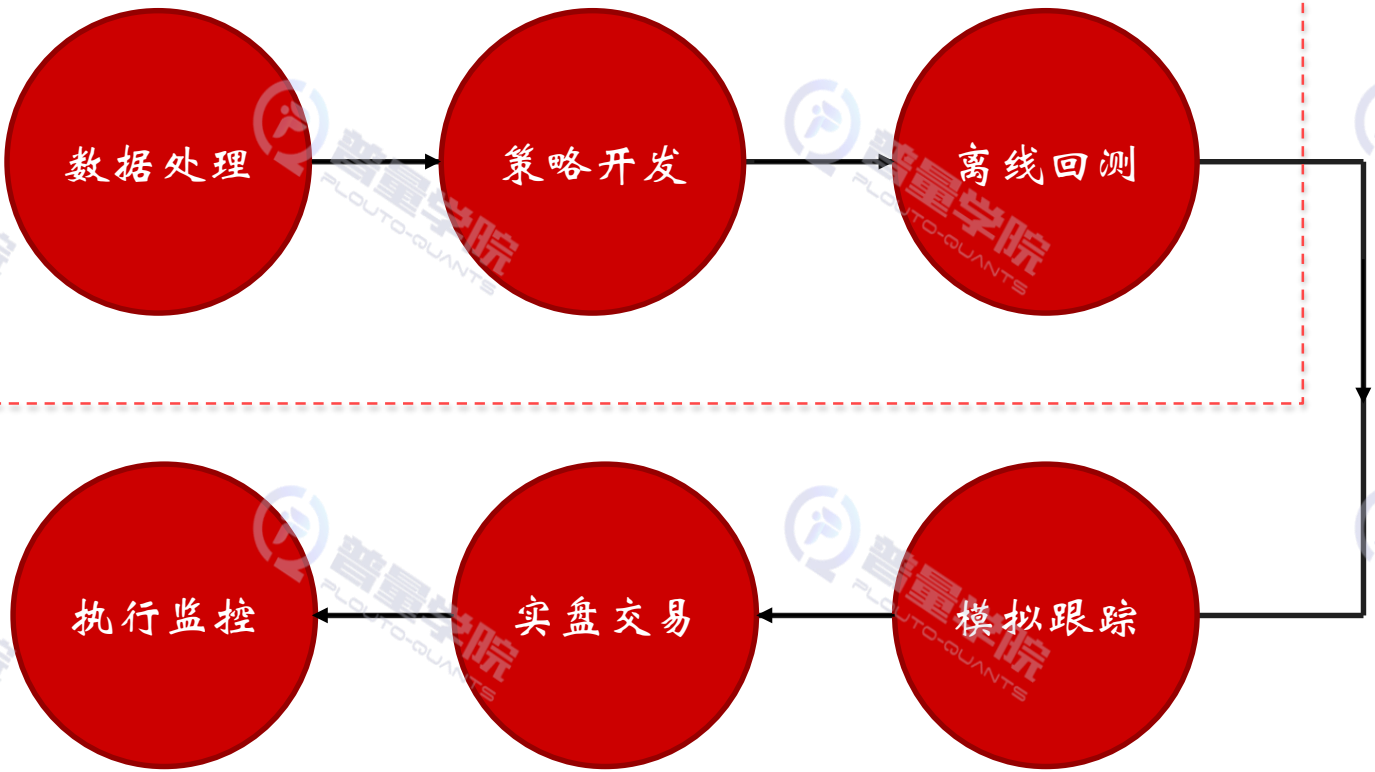
量化交易系统的工作流程

完整流程

- 从TuShare 获取行情数据
- 从东财网页获取财报数据
- 处理数据存在的问题

- 计算PE
- 实现股票池选股逻辑
- 实现股票池的收益统计

- 持仓股的除权除息处理
- 卖出和买入交易实现
- 收盘后交易信号判断
- 收益统计



低估值的股票池择时策略

- 股票池
  - 0 < PE < 30
  - PE从小到大排序，剔除停牌，取前100只
  - 调整周期：7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

开发环境准备

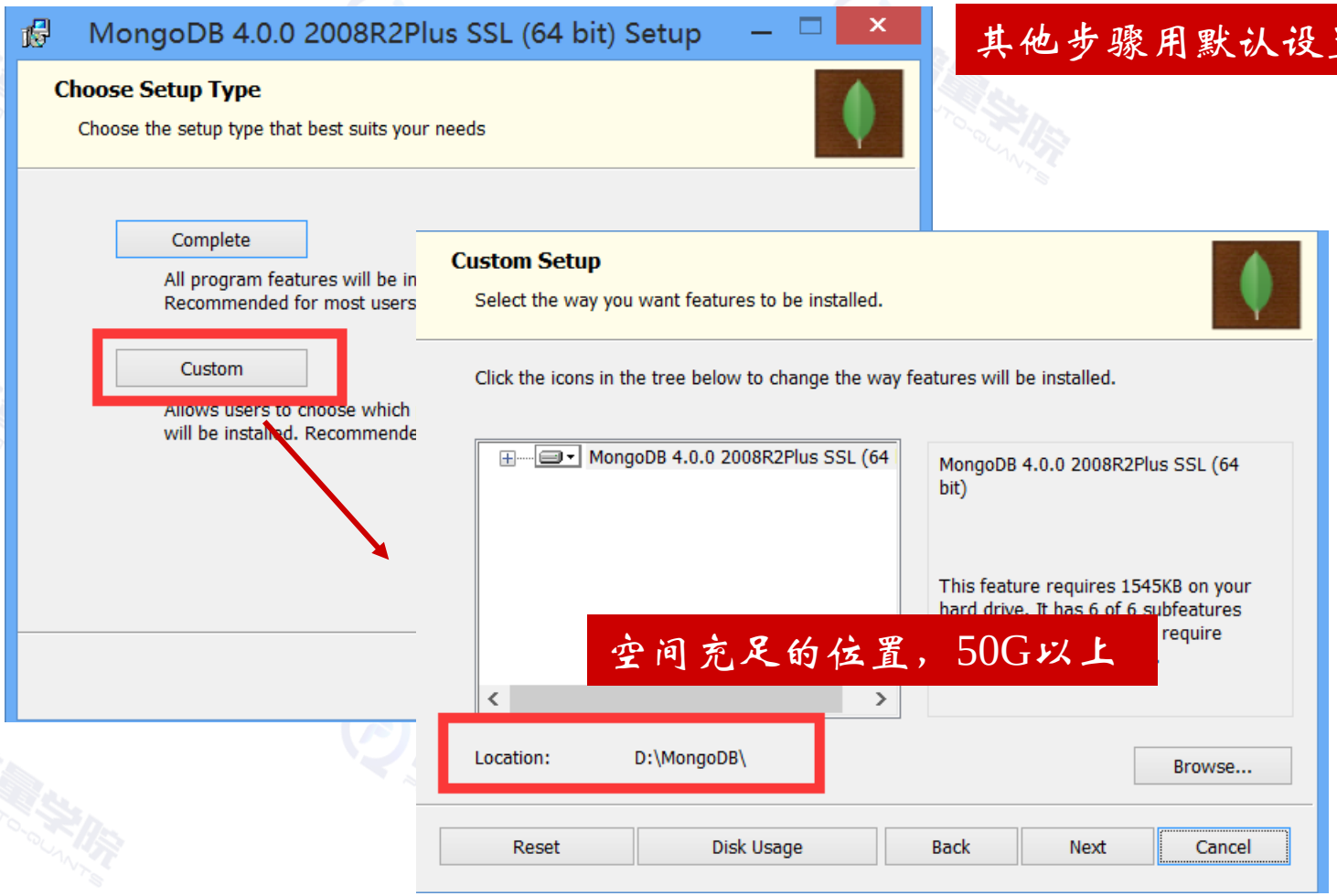
需要安装的软件

- MongoDB
- Python3.7.0
- PyCharm CE

MongoDB

- 版本：MongoDB Community Edition
- 安装
  - 文档：https://docs.mongodb.com/manual/installation/
  - 根据操作系统选择安装指导文档
- 下载地址（Windows）
  - https://www.mongodb.com/download-center?_ga=2.179942756.1915543413.1530790397-2013640144.1530790397#community
- 注意：
  - Windows应该安装为系统服务

MongoDB安装

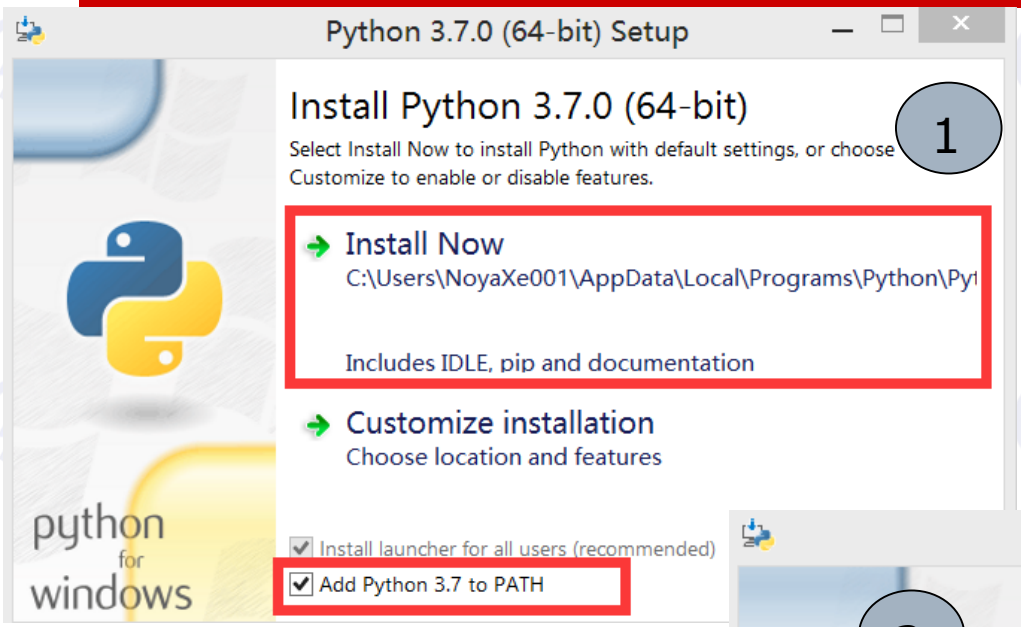


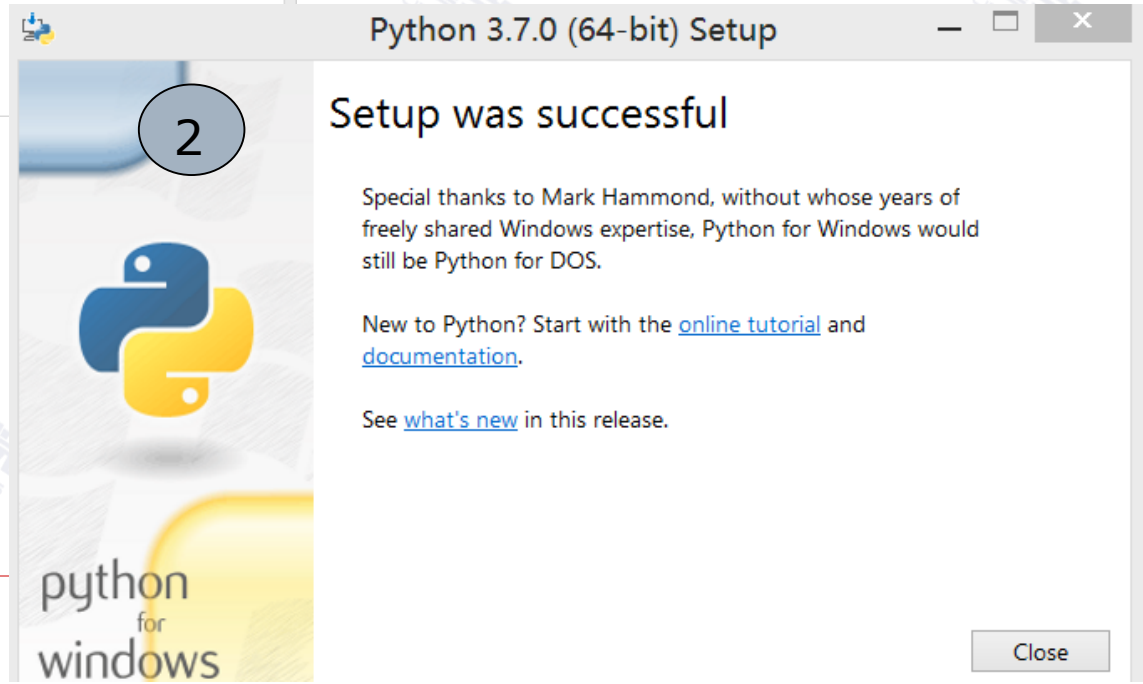
其他步骤用默认设置即可

Python

- 版本：3.7
- 下载中心地址
  - https://www.python.org/downloads/release/python-370/
  - 选择适合自己操作系统的版本
- 安装pymongo
  - 在线文档：http://api.mongodb.com/python/current/
  - pip install pymonogo

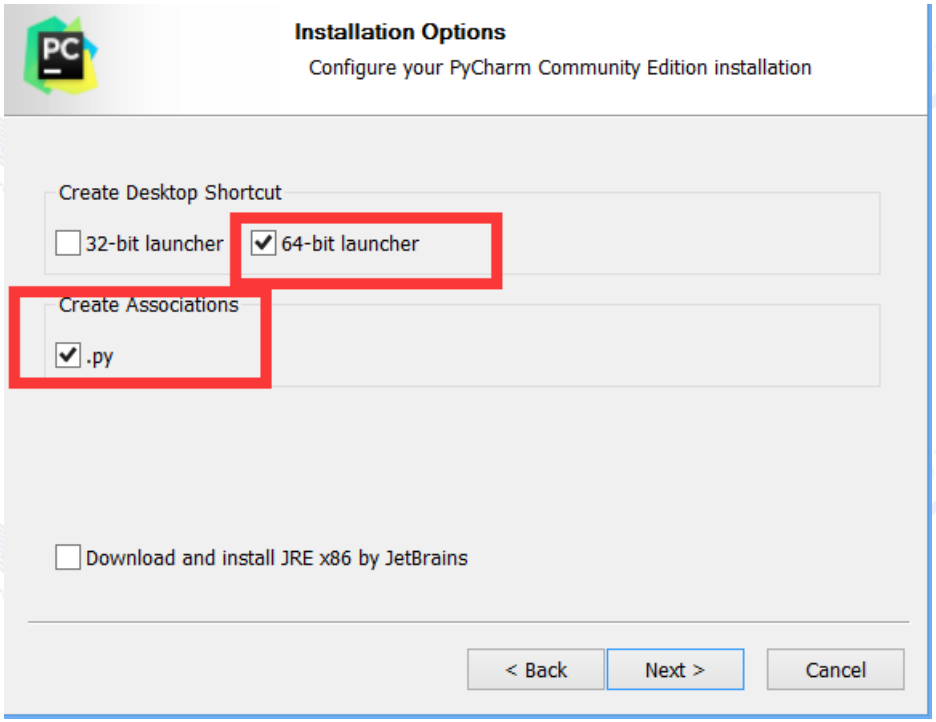
Python


Python 3.7.0 (64-bit) Setup
Install Python 3.7.0 (64-bit)
Select Install Now to install Python with default settings, or choose
Customize to enable or disable features.
1
Install Now
C:\Users\NoyaXe001\AppData\Local\Programs\Python\Pyt
Includes IDLE, pip and documentation
Customize installation
Choose location and features
python
for
windows
Install launcher for all users (recommended)
Add Python 3.7 to PATH

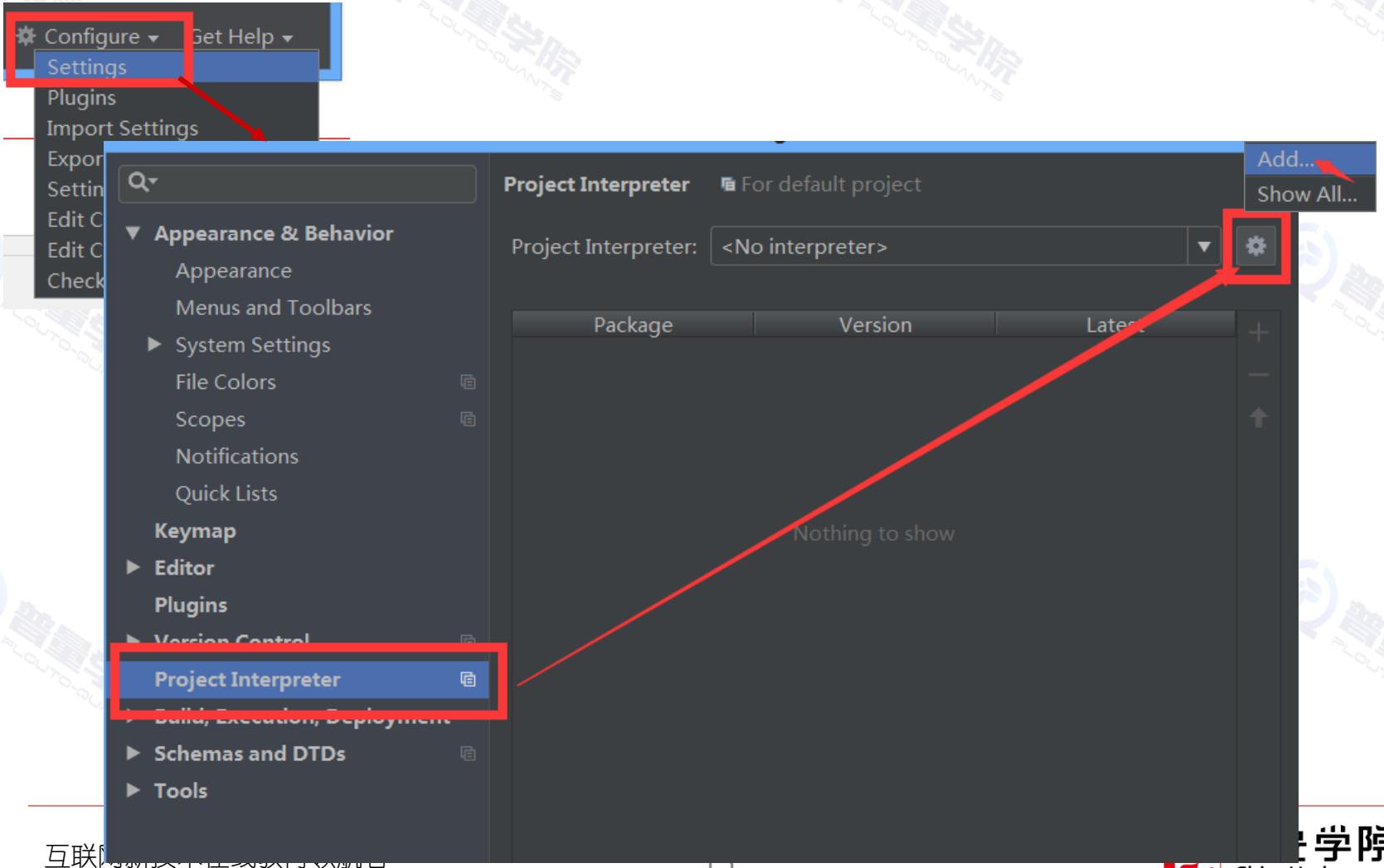

Python 3.7.0 (64-bit) Setup
2
Setup was successful
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
New to Python? Start with the online tutorial and documentation.
See what's new in this release.
python
for
windows
Close

PyCharm

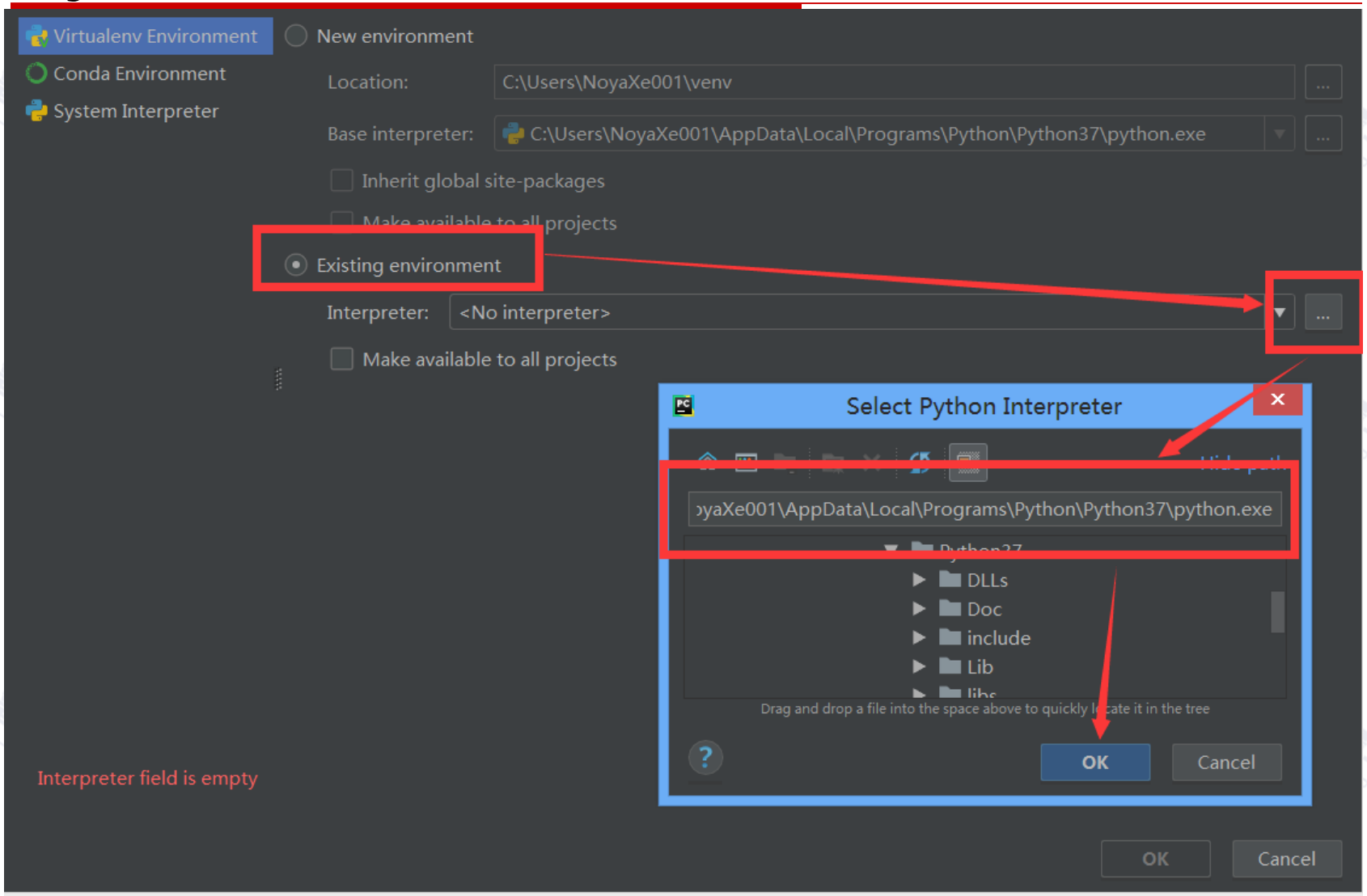
- 版本：Community
- 下载地址
  - http://www.jetbrains.com/pycharm/download/#section=windows



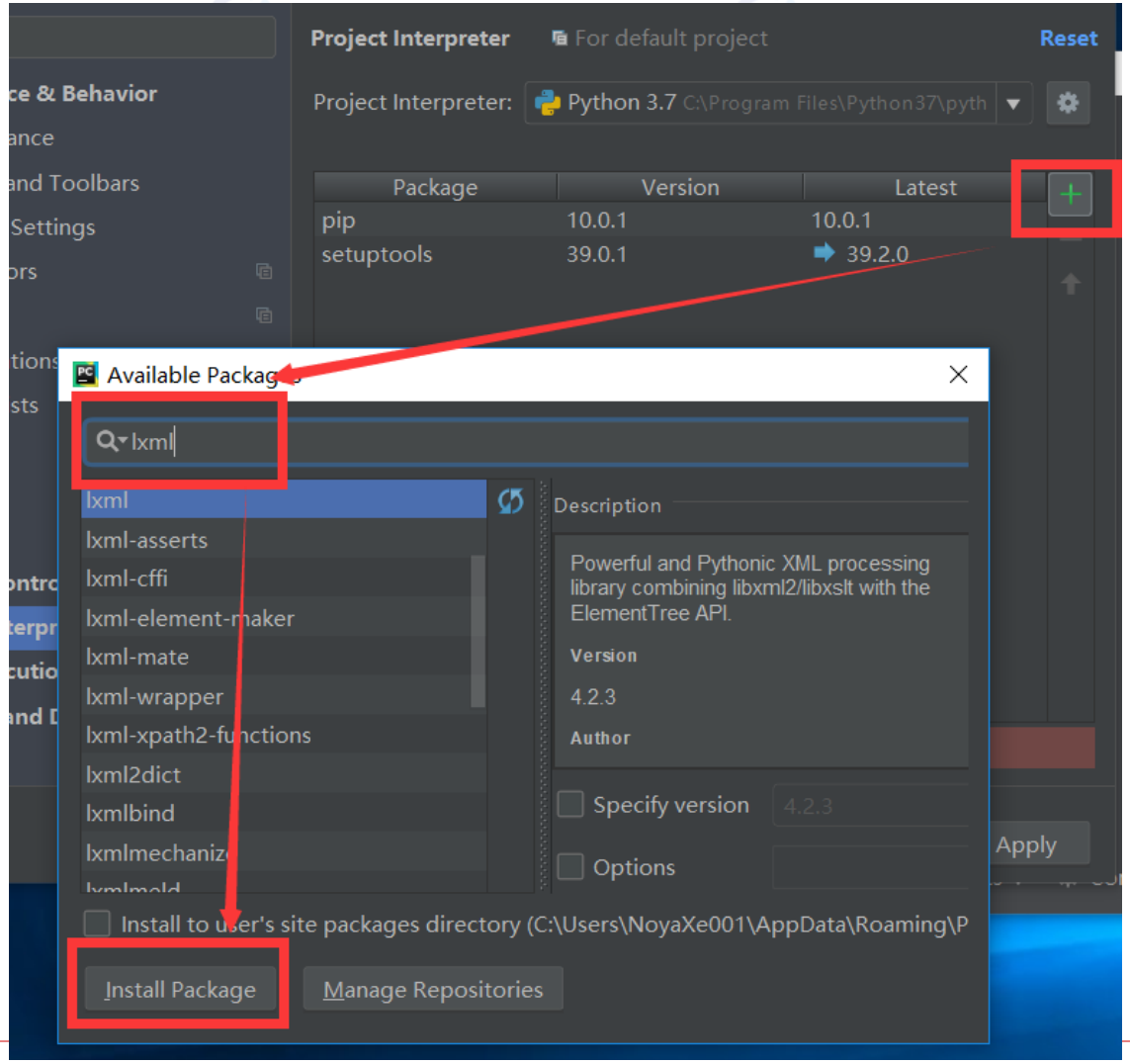
PyCharm设置


Configure
Get Help
Settings
Plugins
Import Settings
Project Interpreter
For default project
Add...
Show All...
Project Interpreter:
<No interpreter>
Package
Version
Nothing to show
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Keymap
Editor
Plugins
Project Interpreter
Schemas and DTDs
Tools

PyCharm设置


Virtualenv Environment
Conda Environment
System Interpreter
New environment
Location:
C:\Users\NoyaXe001\venv
Base interpreter:
C:\Users\NoyaXe001\AppData\Local\Programs\Python\Python37\python.exe
Inherit global site-packages
Existing environment
Interpreter:
<No interpreter>
Make available to all projects
Interpreter field is empty
OK
Cancel
Select Python Interpreter
oyaXe001\AppData\Local\Programs\Python\Python37\python.exe
DLLs
Doc
include
Lib
Drag and drop a file into the space above to quickly locate it in the tree
OK
Cancel

PyCharm设置 – 安装Python包



数据—量化之血

获取行情和财报数据

股票分时图


贵州茅台 所属行业: 酿酒行业
沪股通 融资融券
762.46
760.79
759.11
757.44
755.76
754.09
752.41
750.74
749.07
747.39
745.72
744.04
742.37
740.69
1.56%
1.34%
1.12%
0.89%
0.67%
0.45%
0.22%
0.00%
0.22%
0.45%
0.67%
0.89%
1.12%
1.34%
1758
1172
586
MACD (12,26,9) DIF:0.548↑ DEA:0.430↑ MACD:0.237↓
改参数 加指标 换指标
3.09
1.02
-1.05
-3.12
-2.13
11/19
9:30
10:30
11:30
13:13
14:00
15:00

K线图（蜡烛图）


沪股通
融资融券
设置均线
799.06
469.98
750.00
700.00
650.00
600.00
550.00
500.00
VOL(5,10) VOLUME: 50925.000 MAVOL1: 63702.600 MAVOL2: 57966.600
改参数
加指标
换指标
12.80
6.33
0.00
资金趋势 资金流（万）: -6478.501 ma5: -25786.404 ma10: -12764.132 ma20: -5938.698
4.87
-1319.30
-4.98
10
11
12
2018

量化基本功

- 从数据源获取数据
- 发现并处理数据中存在的问题
- 通过基础数据计算出策略所需数据
- 了解数据不同数据源的差异

小结

不同的交易软件处理后的分钟数据

不完全一致？

明确一个行情数据接收和处理的概念

为今后搭建量化系统和进行策略开发打下基础

任务

- 从Tushare获取历史行情
  - 后复权（daily_hfq）
  - 不复权（daily）
- 从东方财富抓取财务报表数据

code：股票代码
date：日期
index：是否为指数
open：开盘价
close：收盘价
high：最高价
low：最低价
volume：成交量

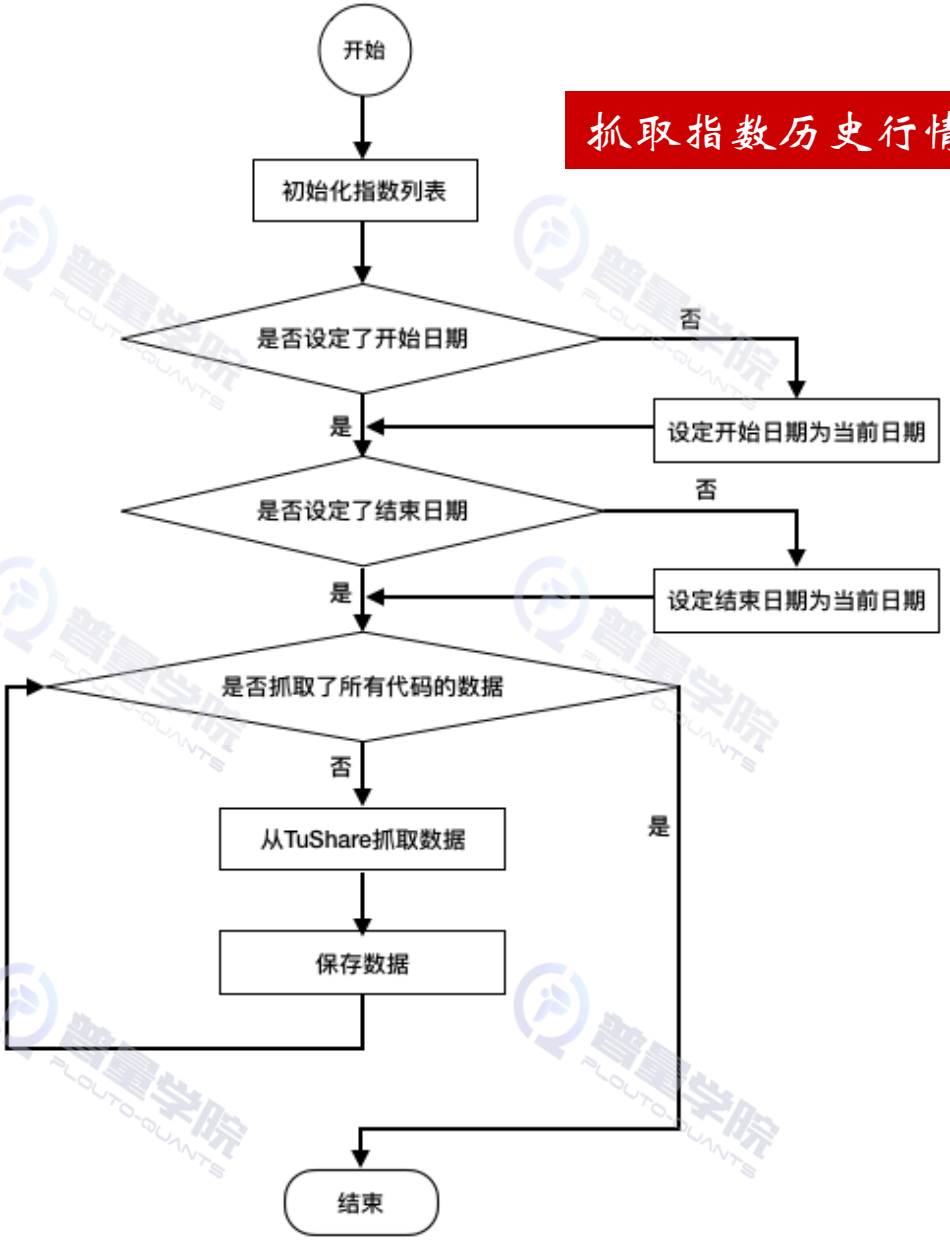
历史行情抓取－初始化

```
from pymongo import UpdateOne
from database import DB_CONN
import tushare as ts
from datetime import datetime

"""
从tushare获取日K数据，保存到本地的MongoDB数据库中
"""


class DailyCrawler:
    def __init__(self):
        """
        初始化
        """

        # 创建daily数据集
        self.daily = DB_CONN['daily']
        # 创建daily_hfq数据集
        self.daily_hfq = DB_CONN['daily_hfq']
```

历史行情抓取－执行入口

```
# 抓取程序的入口函数
if __name__ == '__main__':
    dc = DailyCrawler()
    # 抓取指定日期范围的指数日行情
    # 这两个参数可以根据需求改变，时间范围越长，抓取时花费的时间就会越长
    dc.crawl_index('2015-01-01', '2015-12-31')
    # 抓取指定日期范围的股票日行情
    # 这两个参数可以根据需求改变，时间范围越长，抓取时花费的时间就会越长
    dc.crawl('2015-01-01', '2015-12-31')
```

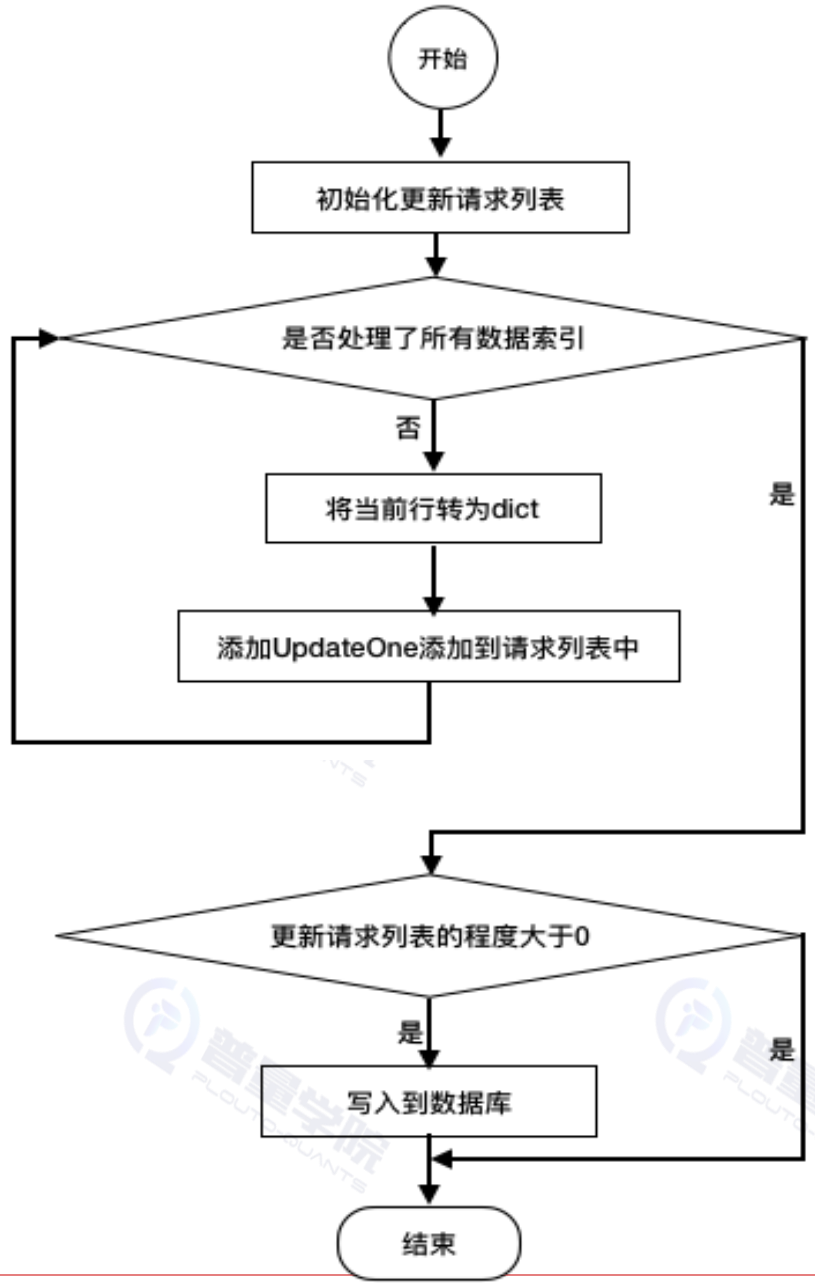
抓取指数历史行情 – 流程图



抓取指数历史行情 – 代码

```
def crawl_index(self, begin_date=None, end_date=None):
    """
    抓取指数的日K数据。
    指数行情的主要作用：
    1. 用来生成交易日历
    2. 回测时做为收益的对比基准

    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 指定抓取的指数列表，可以增加和改变列表里的值
    index_codes = ['000001', '000300', '399001', '399005', '399006']

    # 当前日期
    now = datetime.now().strftime('%Y-%m-%d')
    # 如果没有指定开始，则默认为当前日期
    if begin_date is None:
        begin_date = now

    # 如果没有指定结束日，则默认为当前日期
    if end_date is None:
        end_date = now

    # 按照指数的代码循环，抓取所有指数信息
    for code in index_codes:
        # 抓取一个指数的在时间区间的数据
        df_daily = ts.get_k_data(code, index=True, start=begin_date, end=end_date)
        # 保存数据
        self.save_data(code, df_daily, self.daily, {'index': True})
```

保存数据—流程图

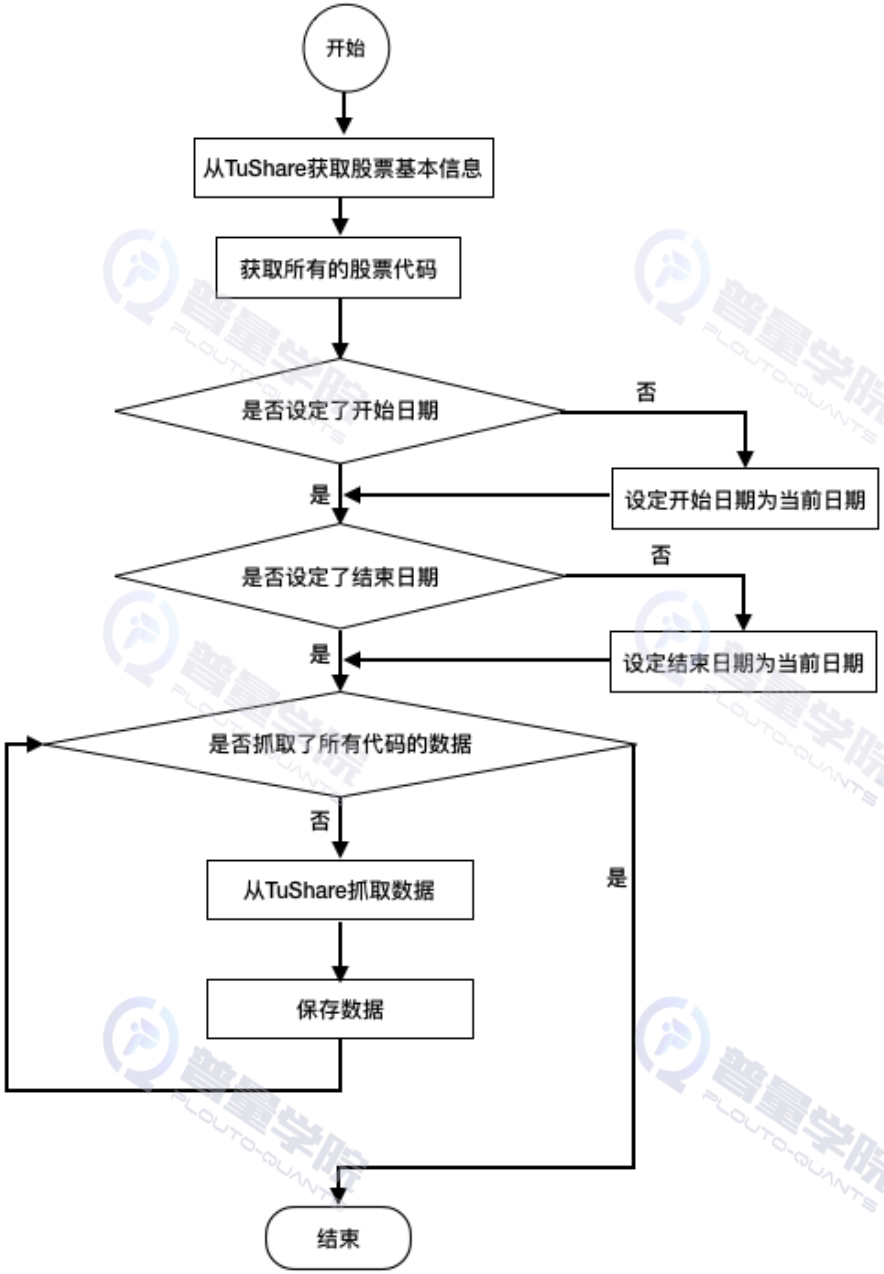

开始
初始化更新请求列表
是否处理了所有数据索引
否
将当前行转为dict
添加UpdateOne添加到请求列表中
是
更新请求列表的程度大于0
是
写入到数据库
是
结束

保存数据—代码

```
def save_data(self, code, df_daily, collection, extra_fields=None):
    """
    将从网上抓取的数据保存到本地MongoDB中

    :param code: 股票代码
    :param df_daily: 包含日线数据的DataFrame
    :param collection: 要保存的数据集
    :param extra_fields: 除了K线数据中保存的字段，需要额外保存的字段
    """

    # 数据更新的请求列表
    update_requests = []

    # 将DataFrame中的行情数据，生成更新数据的请求
    for df_index in df_daily.index:
        # 将DataFrame中的一行数据转dict
        doc = dict(df_daily.loc[df_index])
        # 设置股票代码
        doc['code'] = code

        # 如果指定了其他字段，则更新dict
        if extra_fields is not None:
            doc.update(extra_fields)
        )
```

保存数据—代码

```
    # 生成一条数据库的更新请求
    # 注意：
    # 需要在code、date、index三个字段上增加索引，否则随着数据量的增加，
    # 写入速度会变慢，创建索引的命令式：
    # db.daily.createIndex({'code':1,'date':1,'index':1})
    update_requests.append(
        UpdateOne(
            {'code': doc['code'], 'date': doc['date'], 'index': doc['index']},
            {'$set': doc},
            upsert=True)

# 如果写入的请求列表不为空，则保存都数据库中
if len(update_requests) > 0:
    # 批量写入到数据库中，批量写入可以降低网络IO，提高速度
    update_result = collection.bulk_write(update_requests, ordered=False)
    print('保存日线数据，代码： %s，插入：%4d条，更新：%4d条' %
          (code, update_result.upserted_count, update_result.modified_count),
          flush=True)
```


开始
从TuShare获取股票基本信息
获取所有的股票代码
是否设定了开始日期
否
设定开始日期为当前日期
是
是否设定了结束日期
否
设定结束日期为当前日期
是
是否抓取了所有代码的数据
否
从TuShare抓取数据
保存数据
是
结束

抓取所有股票历史行情 – 代码

```
def crawl(self, begin_date=None, end_date=None):
    """
    抓取股票的日K数据，主要包含了不复权和后复权两种

    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 通过tushare的基本信息API，获取所有股票的基本信息
    stock_df = ts.get_stock_basics()
    # 将基本信息的索引列表转化为股票代码列表
    codes = list(stock_df.index)

    # 当前日期
    now = datetime.now().strftime('%Y-%m-%d')

    # 如果没有指定开始日期，则默认为当前日期
    if begin_date is None:
        begin_date = now

    # 如果没有指定结束日期，则默认为当前日期
    if end_date is None:
        end_date = now

    for code in codes:
        # 抓取不复权的价格
        df_daily = ts.get_k_data(code, autype=None, start=begin_date, end=end_date)
        self.save_data(code, df_daily, self.daily, {'index': False})

        # 抓取后复权的价格
        df_daily_hfq = ts.get_k_data(code, autype='hfq', start=begin_date, end=end_date)
        self.save_data(code, df_daily_hfq, self.daily_hfq, {'index': False})
```

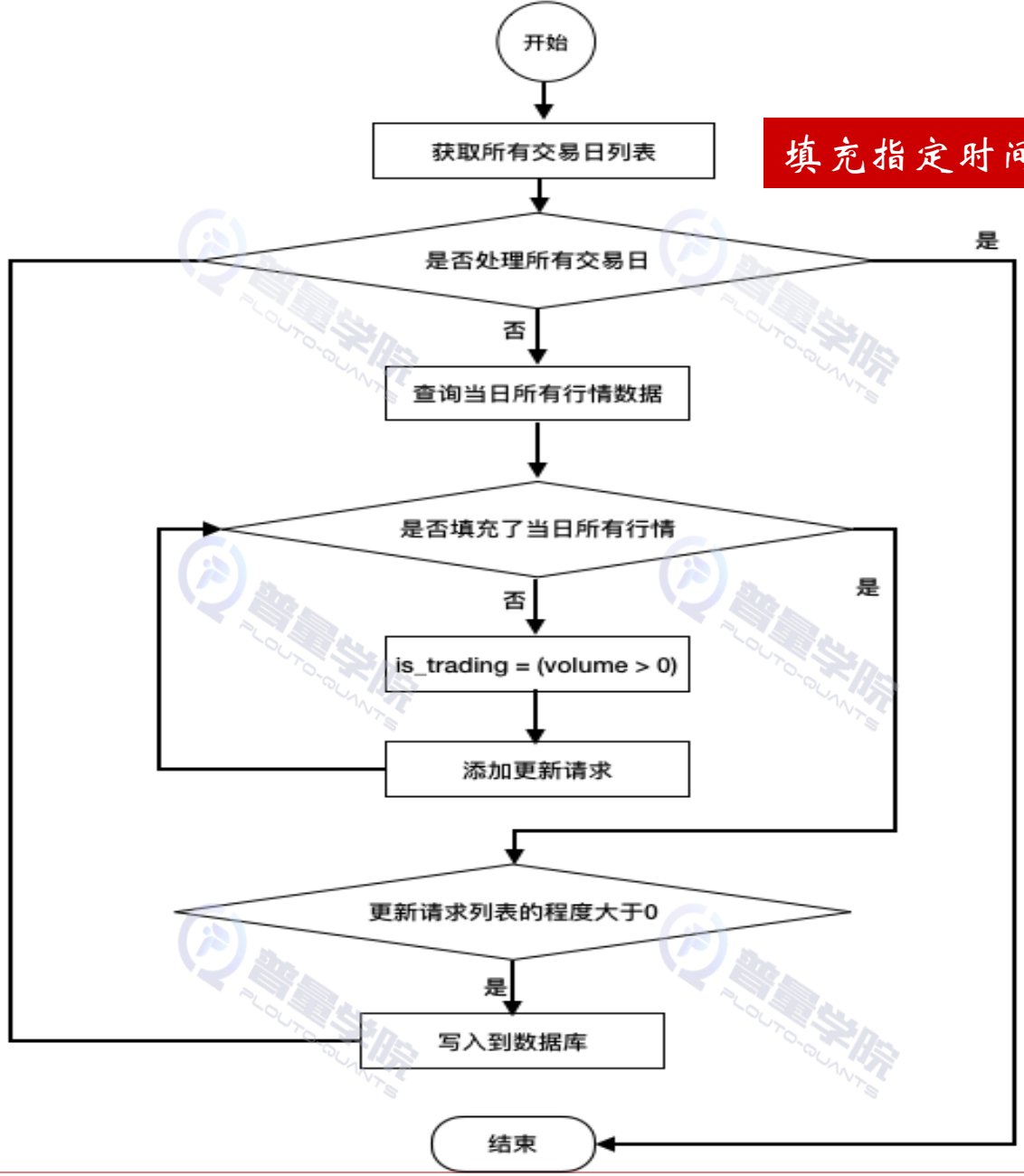
补充停牌的日K数据

- 问题：
  - 从TuShare获取的数据，停牌日没有数据
- 影响
  - 回测时，不能直接参与账户的净值计算，导致账户的净值以及收益计算不准确
- 解决方法
  - 增加is_trading字段，区分停牌日和交易日
  - 补充停牌日的日K数据，根据当前数据现状，填充的数据为：open、close、high、low为停牌前最后一个交易日的close，volume为0，is_trading为false。

讲解的知识点（内部用，不对外）

- 为什么需要补充停牌日的K线数据，对比补充前和补充后的系统代码复杂度
- 为什么要增加一个is_trading
- 停牌日K线数据的字段
- 为什么要计算复权因子
- 计算复权因子的公式是什么
- 强调要对所使用的数据有比较全面的认识和深刻的理解
- 如何运行代码
- 可以修改的参数有哪些，
- 修改后会有什么作用
- 流程图和关键代码

预计花费时间10分钟左右



填充指定时间段内的is_trading

```
from datetime import datetime, timedelta

from pymongo import UpdateOne, ASCENDING

from database import DB_CONN
from stock_util import get_trading_dates, get_all_codes

"""
对日行情数据做进一步的处理：
1. 填充is_trading字段，is_trading用来区分某只股票在某个交易日是否为停牌
2. 填充停牌日的行情数据
3. 填充复权因子和前收
"""


def fill_is_trading_between(begin_date=None, end_date=None):
    """
    填充指定时间段内的is_trading字段
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 获取指定日期范围的所有交易日列表，按日期正序排列
    all_dates = get_trading_dates(begin_date, end_date)

    # 循环填充所有交易日的is_trading字段
    for date in all_dates:
        # 填充daily数据集
        fill_single_date_is_trading(date, 'daily')
        # 填充daily_hfq数据集
        fill_single_date_is_trading(date, 'daily_hfq')
```

填充指定时间段内的is_trading

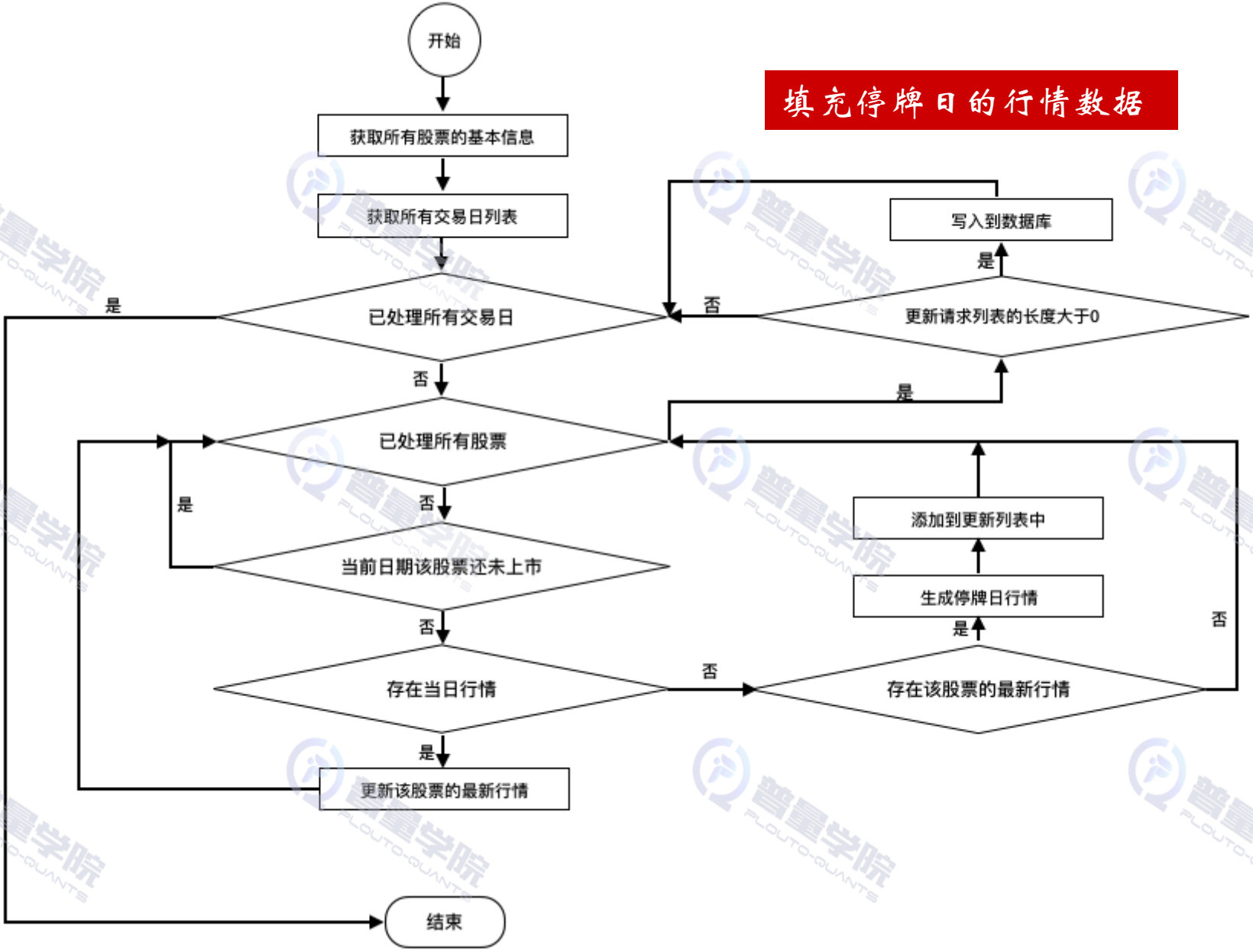
```
def fill_is_trading(date=None):
    """
    为日线数据增加is_trading字段，表示是否交易的状态，True - 交易  False - 停牌
    从Tushare来的数据不包含交易状态，也不包含停牌的日K数据，为了系统中使用的方便，我们需要填充停牌是的K数据。
    一旦填充了停牌的数据，那么数据库中就同时包含了停牌和交易的数据，为了区分这两种数据，就需要增加这个字段。

    在填充该字段时，要考虑到是否最坏的情况，也就是数据库中可能已经包含了停牌和交易的数据，但是却没有is_trading
    字段。这个方法通过交易量是否为0，来判断是否停牌
    """

    if date is None:
        all_dates = get_trading_dates()
    else:
        all_dates = [date]

    for date in all_dates:
        fill_single_date_is_trading(date, 'daily')
        fill_single_date_is_trading(date, 'daily_hfq')
```

填充指定时间段内的is_trading

```
def fill_single_date_is_trading(date, collection_name):
    """
    填充某一个日行情的数据集的is_trading
    :param date: 日期
    :param collection_name: 集合名称
    """
    print('填充字段， 字段名: is_trading，日期: %s，数据集: %s' %
          (date, collection_name), flush=True)
    daily_cursor = DB_CONN[collection_name].find(
        {'date': date},
        projection={'code': True, 'volume': True, '_id': False},
        batch_size=1000)

    update_requests = []
    for daily in daily_cursor:
        # 当日成交量大于0，则为交易状态
        is_trading = daily['volume'] > 0

        update_requests.append(
            UpdateOne(
                {'code': daily['code'], 'date': date},
                {'$set': {'is_trading': is_trading}}))

    if len(update_requests) > 0:
        update_result = DB_CONN[collection_name].bulk_write(update_requests, ordered=False)
        print('填充字段， 字段名: is_trading，日期: %s，数据集: %s，更新: %4d条' %
              (date, collection_name, update_result.modified_count), flush=True)
```

填充指定时间段内的is_trading

填充停牌日的行情数据


开始
获取所有股票的基本信息
获取所有交易日列表
已处理所有交易日
是
否
已处理所有股票
是
否
当前日期该股票还未上市
是
否
存在当日行情
是
否
更新该股票的最新行情
存在该股票的最新行情
是
否
生成停牌日行情
添加到更新列表中
是
否
更新请求列表的长度大于0
是
否
写入到数据库
结束

```
def fill_daily_k_at_suspension_days(begin_date=None, end_date=None):
    """
    填充指定日期范围内，股票停牌日的行情数据。
    填充时，停牌的开盘价、最高价、最低价和收盘价都为最近一个交易日的收盘价，成交量为0，
    is_trading是False

    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    # 当前日期的前一天
    before = datetime.now() - timedelta(days=1)
    # 找到据当前最近一个交易日的所有股票的基本信息
    basics = []
    while 1:
        # 转化为str
        last_trading_date = before.strftime('%Y-%m-%d')
        # 因为TuShare的基本信息最早知道2016-08-09，所以如果日期早于2016-08-09
        # 则结束查找
        if last_trading_date < '2016-08-09':
            break

        # 找到当日的基本信息
        basic_cursor = DB_CONN['basic'].find(
            {'date': last_trading_date},
            # 填充时需要用到两个字段股票代码code和上市日期timeToMarket，
            # 上市日期用来判断
            projection={'code': True, 'timeToMarket': True, '_id': False},
            # 一次返回5000条，可以降低网络IO开销，提高速度
            batch_size=5000)
```

填充停牌日的行情数据

```
    # 将数据放到basics列表中
    basics = [basic for basic in basic_cursor]

    # 如果查询到了数据，在跳出循环
    if len(basics) > 0:
        break

    # 如果没有找到数据，则继续向前一天
    before -= timedelta(days=1)

# 获取指定日期范围内所有交易日列表
all_dates = get_trading_dates(begin_date, end_date)

# 填充daily数据集中的停牌日数据
fill_daily_k_at_suspension_days_at_date_one_collection(
    basics, all_dates, 'daily')
# 填充daily_hfq数据中的停牌日数据
fill_daily_k_at_suspension_days_at_date_one_collection(
    basics, all_dates, 'daily_hfq')
```

填充停牌日的行情数据

填充停牌日的行情数据

```
def fill_daily_k_at_suspension_days_at_date_one_collection(
        basics, all_dates, collection):
    """
    更新单个数据集的单个日期的数据
    :param basics:
    :param all_dates:
    :param collection:
    :return:
    """
    code_last_trading_daily_dict = dict()
    for date in all_dates:
        update_requests = []
        last_daily_code_set = set(code_last_trading_daily_dict.keys())
        for basic in basics:
            code = basic['code']
            # 如果循环日期小于
            if date < basic['timeToMarket']:
                print('日期: %s, %s 还没上市，上市日期: %s' % (date, code, basic['timeToMarket']),
flush=True)
            else:
                # 找到当日数据
                daily = DB_CONN[collection].find_one({'code': code, 'date': date})
                if daily is not None:
                    code_last_trading_daily_dict[code] = daily
                    last_daily_code_set.add(code)
                else:
```

填充停牌日的行情数据

```
if code in last_daily_code_set:
                    last_trading_daily = code_last_trading_daily_dict[code]
                    suspension_daily_doc = {
                        'code': code,
                        'date': date,
                        'close': last_trading_daily['close'],
                        'open': last_trading_daily['close'],
                        'high': last_trading_daily['close'],
                        'low': last_trading_daily['close'],
                        'volume': 0,
                        'is_trading': False
                    }
                    update_requests.append(
                        UpdateOne(
                            {'code': code, 'date': date},
                            {'$set': suspension_daily_doc},
                            upsert=True))
    if len(update_requests) > 0:
        update_result = DB_CONN[collection].bulk_write(update_requests, ordered=False)
        print('填充停牌数据，日期：%s，数据集：%s，插入：%4d条，更新：%4d条' %
              (date, collection, update_result.upserted_count, update_result.modified_count),
flush=True)
```

```
def fill_au_factor_pre_close(begin_date, end_date):
    """
    为daily数据集填充：
    1. 复权因子au_factor，复权的因子计算方式：au_factor = hfq_close/close
    2. pre_close = close(-1) * au_factor(-1)/au_factor
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """
    all_codes = get_all_codes()

    for code in all_codes:
        hfq_daily_cursor = DB_CONN['daily_hfq'].find(
            {'code': code, 'date': {'$lte': end_date, '$gte': begin_date}, 'index': False},
            sort=[('date', ASCENDING)],
            projection={'date': True, 'close': True})

        date_hfq_close_dict = dict([(x['date'], x['close']) for x in hfq_daily_cursor])

        daily_cursor = DB_CONN['daily'].find(
            {'code': code, 'date': {'$lte': end_date, '$gte': begin_date}, 'index': False},
            sort=[('date', ASCENDING)],
            projection={'date': True, 'close': True}
        )

        last_close = -1
        last_au_factor = -1

        update_requests = []
```

填充复权因子字段和前收字段

填充复权因子字段和前收字段

```
for daily in daily_cursor:
            date = daily['date']
            try:
                close = daily['close']

                doc = dict()

                au_factor = round(date_hfq_close_dict[date] / close, 2)
                doc['au_factor'] = au_factor
                if last_close != -1 and last_au_factor != -1:
                    pre_close = last_close * last_au_factor / au_factor
                    doc['pre_close'] = round(pre_close, 2)

                last_au_factor = au_factor
                last_close = close

                update_requests.append(
                    UpdateOne(
                        {'code': code, 'date': date, 'index': False},
                        {'$set': doc}))
            except:
                print('计算复权因子时发生错误，股票代码：%s，日期：%s' % (code, date), flush=True)
                # 恢复成初始值，防止用错
                last_close = -1
                last_au_factor = -1

        if len(update_requests) > 0:
            update_result = DB_CONN['daily'].bulk_write(update_requests, ordered=False)
            print('填充复权因子和前收，股票：%s，更新：%4d条' %
                  (code, update_result.modified_count), flush=True)
```

获取股票基本信息

- 用途
  - 获取每日的股票列表
- 主要字段
  - 股票代码
  - 股票名称
  - 股本（总股本、流通股本）
  - 上市日期
  - 日期
- 来源
  - TuShare -- get_stock_basics

获取基本信息

```
#  -*- coding: utf-8 -*-

from datetime import datetime, timedelta

import tushare as ts
from pymongo import UpdateOne

from database import DB_CONN
from stock_util import get_trading_dates

"""
从tushare获取股票基础数据，保存到本地的MongoDB数据库中
"""


def crawl_basic(begin_date=None, end_date=None):
    """
    抓取指定时间范围内的股票基础信息
    :param begin_date: 开始日期
    :param end_date: 结束日期
    """

    if begin_date is None:
        begin_date = (datetime.now() - timedelta(days=1)).strftime('%Y-%m-%d')

    if end_date is None:
        end_date = (datetime.now() - timedelta(days=1)).strftime('%Y-%m-%d')

    all_dates = get_trading_dates(begin_date, end_date)

    for date in all_dates:
        try:
            crawl_basic_at_date(date)
        except:
            print('抓取股票基本信息时出错，日期：%s' % date, flush=True)
```

```
def crawl_basic_at_date(date):
    df_basics = ts.get_stock_basics(date)

    # 如果当日没有基础信息，在不做操作
    if df_basics is None:
        return

    update_requests = []
    codes = set(df_basics.index)
    for code in codes:
        doc = dict(df_basics.loc[code])
        try:
            # 将20180101转换为2018-01-01的形式
            time_to_market = datetime \
                .strptime(str(doc['timeToMarket']), '%Y%m%d').strftime('%Y-%m-%d')

            totals = float(doc['totals'])
            outstanding = float(doc['outstanding'])
            doc.update({
                'code': code, 'date': date,'timeToMarket': time_to_market, 'outstanding': outstanding, 'totals': totals
            })

            update_requests.append(
                UpdateOne({'code': code, 'date': date}, {'$set': doc}, upsert=True))
        except:
            print('发生异常，股票代码：%s，日期：%s' % (code, date), flush=True)
            print(doc, flush=True)

    if len(update_requests) > 0:
        update_result = DB_CONN['basic'].bulk_write(update_requests, ordered=False)

        print('抓取股票基本信息，日期：%s，插入：%4d条，更新：%4d条' %
              (date, update_result.upserted_count, update_result.modified_count), flush=True)


if __name__ == '__main__':
    crawl_basic('2015-01-01', '2018-06-30')
```

获取基本信息

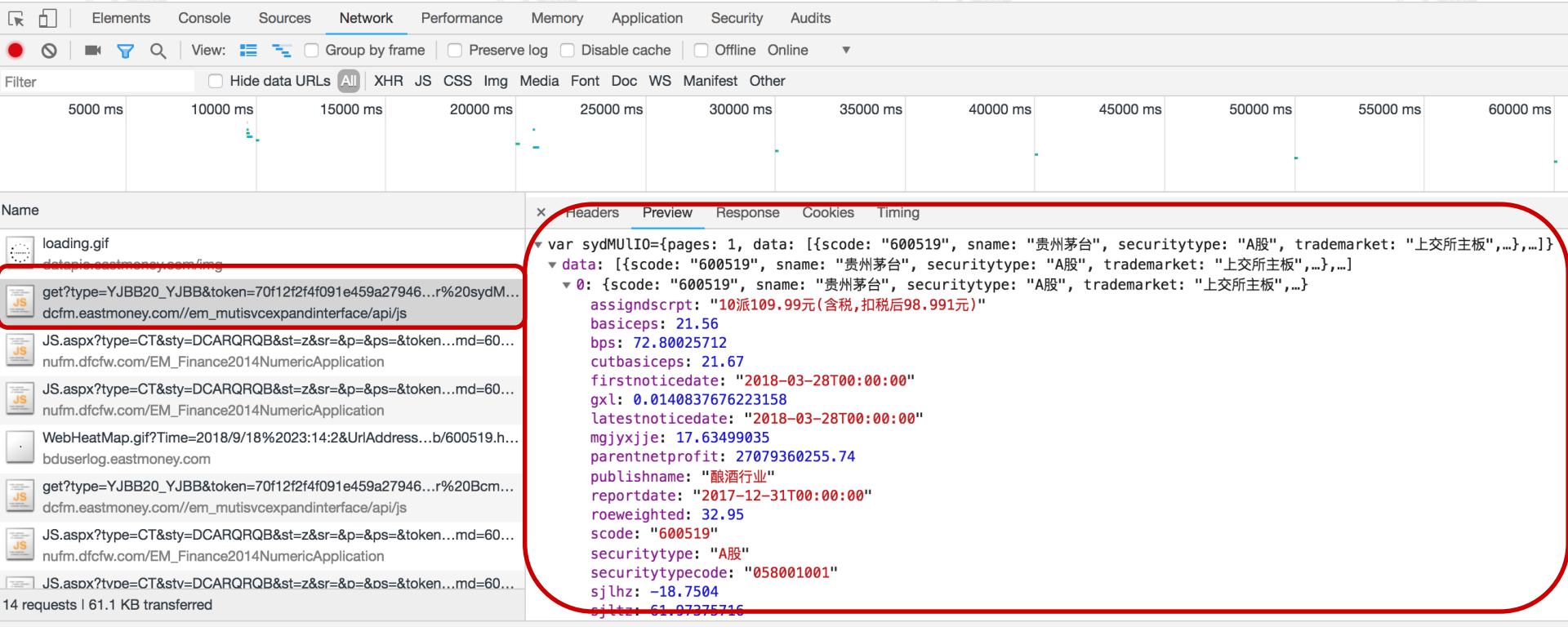
从网页抓取财报数据

财务数据抓取

业绩报表 | 业绩快报 | 业绩预告 | 预约披露时间 | 资产负债表 | 利润表 | 现金流量表

报告期 全部	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 06-30	12.55	-	353亿	38.27	-8.37	158亿	40.12	-14.69	74.3495	15.87	14.118	90.94	不分配不转增	-	2018 08-02	2018 08-02
2018 03-31	6.77	-	184亿	32.21	10.98	85.07亿	38.93	19.89	79.5718	8.89	3.9289	91.31	-	-	2018 04-28	2018 04-28
2017 12-31	21.56	21.67	611亿	52.07	-12.73	271亿	61.97	-18.75	72.8003	32.95	17.635	89.80	10派109.99元(含税,扣税后98.991元)	1.41	2018 03-28	2018 03-28
2017 09-30	15.91	-	445亿	61.58	64.01	200亿	60.31	70.31	67.152	24.97	18.1395	89.93	-	-	2017 10-26	2017 10-26
2017 06-30	8.96	8.98	255亿	36.06	-16.77	113亿	27.81	-16.26	60.1996	14.33	5.5209	89.62	不分配不转增	-	2017 07-28	2018 08-02
2017 03-31	4.87	-	139亿	35.73	10.23	61.23亿	25.24	43.98	62.9023	8.06	4.8631	91.16	-	-	2017 04-25	2018 04-28
2016 12-31	13.31	13.5	402亿	20.06	43.52	167亿	7.84	16.10	58.0276	24.44	29.8132	91.23	10派67.87元(含税,扣税后61.083元)	1.49	2017 04-15	2018 03-28
2016 09-30	9.92	-	275亿	16.00	3.63	125亿	9.11	-6.40	54.6433	18.45	25.8998	91.64	-	-	2016 10-29	2017 10-26
2016 06-30	7.01	7	187亿	15.77	-17.21	88.03亿	11.59	-19.96	51.7267	12.88	10.8295	91.88	不分配不转增	-	2017 07-28	2017 07-28

获取数据抓取地址


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
5000 ms
10000 ms
15000 ms
20000 ms
25000 ms
30000 ms
35000 ms
40000 ms
45000 ms
50000 ms
55000 ms
60000 ms
Name
loading.gif
get?type=YJBB20_YJBB&token=70f12f2f4f091e459a27946...r%20sydM...
dcfm.eastmoney.com//em_mutisvcexpandinterface/api/js
JS.aspx?type=CT&sty=DCARQRQB&st=z&sr=&p=&ps=&token...md=60...
nufm.dfcfw.com/EM_Finance2014NumericApplication
JS.aspx?type=CT&sty=DCARQRQB&st=z&sr=&p=&ps=&token...md=60...
nufm.dfcfw.com/EM_Finance2014NumericApplication
WebHeatMap.gif?Time=2018/9/18%2023:14:2&UrlAddress...b/600519.h...
bduserlog.eastmoney.com
get?type=YJBB20_YJBB&token=70f12f2f4f091e459a27946...r%20Bcm...
dcfm.eastmoney.com//em_mutisvcexpandinterface/api/js
JS.aspx?type=CT&sty=DCARQRQB&st=z&sr=&p=&ps=&token...md=60...
nufm.dfcfw.com/EM_Finance2014NumericApplication
14 requests | 61.1 KB transferred
Headers
Preview
Response
Cookies
Timing
var sydMUlIO={pages: 1, data: [{scode: "600519", sname: "贵州茅台", securitytype: "A股", trademarket: "上交所主板",…},…]}
data: [{scode: "600519", sname: "贵州茅台", securitytype: "A股", trademarket: "上交所主板",…},…]
0: {scode: "600519", sname: "贵州茅台", securitytype: "A股", trademarket: "上交所主板",…}
assigndscrpt: "10派109.99元(含税,扣税后98.991元)"
basiceps: 21.56
bps: 72.80025712
cutbasiceps: 21.67
firstnoticedate: "2018-03-28T00:00:00"
gxl: 0.0140837676223158
latestnoticedate: "2018-03-28T00:00:00"
mgjyxjje: 17.63499035
parentnetprofit: 27079360255.74
publishname: "酿酒行业"
reportdate: "2017-12-31T00:00:00"
roeweighted: 32.95
scode: "600519"
securitytype: "A股"
securitytypecode: "058001001"
sjlhz: -18.7504

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)}

抓取财报数据

```
import json, traceback, urllib3
from pymongo import UpdateOne
from database import DB_CONN
from stock_util import get_all_codes

user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/66.0.3359.181 Safari/537.36'

def crawl_single_page(page):
    """
    抓取单页数据
    """
    url = 'http://nufm.dfcfw.com/EM_Finance2014NumericApplication/JS.aspx?cb=&type=CT&' \
          'token=4f1862fc3b5e77c150a2b985b12db0fd&js=%7B%22data%22%3A%5B(x)%5D%2C%22records' \
          'Total%22%3A(tot)%2C%22recordsFiltered%22%3A(tot)%7D&cmd=C._A&sty=FCOIATC&st=(ChangePercent)&sr=-
1&p={0}&ps=200'

    # url = 'http://www.baidu.com'

    try:
        # 创建连接池
        conn_pool = urllib3.PoolManager()
        response = conn_pool.request('GET', url.replace('{0}', str(page)), headers={'User-Agent':
user_agent})

        return response.data.decode('UTF-8')
    except:
        traceback.print_exc()
        return None
```

抓取财报数据

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

    # 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={page}&ps={pageSize}&js={"pages":(tp),"data":%20(x)}'

    cookie = ..
    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'))

        reports = result['data']

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

```
            update_requests.append(
                UpdateOne(
                    {'code': code, 'report_date': doc['report_date']},
                    {'$set': doc}, upsert=True))

        if len(update_requests) > 0:
            update_result = DB_CONN['finance_report'].bulk_write(update_requests, ordered=False)
            print('股票 %s, 财报，更新 %d, 插入 %d' %
                  (code, update_result.modified_count, update_result.upserted_count))


if __name__ == "__main__":
    crawl_finance_report()
```

抓取财报数据

数据定时增量入库

每天定时抓取

```
#  -*- coding: utf-8 -*-

import schedule
from daily_crawler import DailyCrawler
import time
from datetime import datetime

"""
每天下午15:30执行抓取，只有周一到周五才真正执行抓取任务
"""

def crawl_daily():
    dc = DailyCrawler()
    now_date = datetime.now()
    weekday = now_date.strftime('%w')
    if 0 < weekday < 6:
        now = now_date.strftime('%Y-%m-%d')
        dc.crawl_index(begin_date=now, end_date=now)
        dc.crawl(begin_date=now, end_date=now)


if __name__ == '__main__':
    schedule.every().day.at("15:30").do(crawl_daily)
    while True:
        schedule.run_pending()
        time.sleep(10)
```

实现一个低PE的股票池

条件

- 0 < PE < 30
- PE从小到大排序，剔除停牌，取前100只
- 再平衡周期：7个交易日

需要处理的数据

- PE – 市盈率

  市盈率 = 股价/每股收益（EPS)

  市盈率 = 市值/净利润

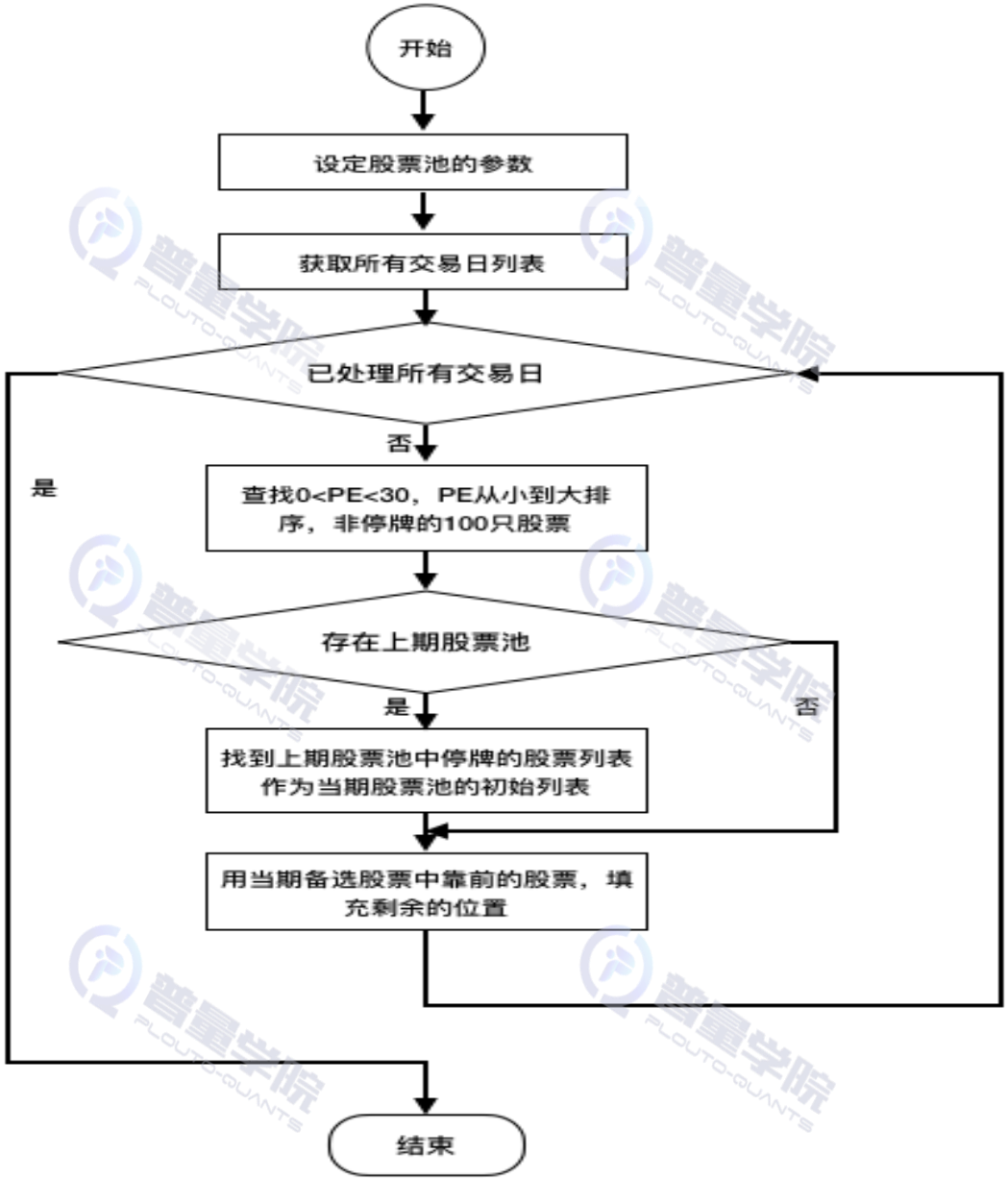
- 未来函数
  - 每股收益以公告日期为准

计算市盈率
文件：pe_computing.py

```
#  -*- coding: utf-8 -*-

"""
compute_pe：计算市盈率
"""

from pymongo import DESCENDING, UpdateOne

from database import DB_CONN
from stock_util import get_all_codes

finance_report_collection = DB_CONN['finance_report']
daily_collection = DB_CONN['daily']


def compute_pe():
    """
    计算股票在某只的市盈率
    """

    # 获取所有股票
    codes = get_all_codes()

    for code in codes:
        print('计算市盈率, %s' % code)
        daily_cursor = daily_collection.find(
            {'code': code},
            projection={'close': True, 'date': True})

        update_requests = []
        for daily in daily_cursor:
            _date = daily['date']
```

```
 update_requests = []
        for daily in daily_cursor:
            _date = daily['date']

            finance_report = finance_report_collection.find_one(
                {'code': code, 'report_date': {'$regex': '\d{4}-12-31'}, 'announced_date':
{'$lte': _date}},
                sort=[('announced_date', DESCENDING)]
            )

            if finance_report is None:
                continue

            # 计算滚动市盈率并保存到daily_k中
                      eps = 0
            if finance_report['eps'] != '-':
                eps = finance_report['eps']

            # 计算PE
            if eps != 0:
                update_requests.append(UpdateOne(
                    {'code': code, 'date': _date},
                    {'$set': {'pe': round(daily['close'] / eps, 4)}}))

        if len(update_requests) > 0:
            update_result = daily_collection.bulk_write(update_requests, ordered=False)
            print('更新PE, %s, 更新: %d' % (code, update_result.modified_count))


if __name__ == "__main__":
    compute_pe()
```

计算市盈率
文件：pe_computing.py


开始
设定股票池的参数
获取所有交易日列表
已处理所有交易日
否
是
查找0<PE<30，PE从小到大排序，非停牌的100只股票
存在上期股票池
是
否
找到上期股票池中停牌的股票列表作为当期股票池的初始列表
用当期备选股票中靠前的股票，填充剩余的位置
结束

```
from pymongo import ASCENDING, DESCENDING
import pandas as pd
import matplotlib.pyplot as plt
from database import DB_CONN
from stock_util import get_trading_dates


daily = DB_CONN['daily']
daily_hfq = DB_CONN['daily_hfq']


def stock_pool(begin_date, end_date):
    """
    股票池的选股逻辑

    :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
    """

    """
    下面的几个参数可以自己修改
    """
    # 调整周期是7个交易日，可以改变的参数
    adjust_interval = 7
    # PE的范围
    pe_range = (0, 30)
    # PE的排序方式， ASCENDING - 从小到大，DESCENDING - 从大到小
    sort = ASCENDING
    # 股票池内的股票数量
    pool_size = 100
```

股票池
文件：stock_pool_strategy.py

```
# 返回值：调整日和当期股票代码列表
adjust_date_codes_dict = dict()
# 返回值：所有的调整日列表
all_adjust_dates = []

# 获取指定时间范围内的所有交易日列表，按照日期正序排列
all_dates = get_trading_dates(begin_date=begin_date, end_date=end_date)

# 上一期的所有股票代码
last_phase_codes = []
# 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)

    print('调整日期:  %s' % adjust_date, flush=True)

    # 查询出调整当日，0 < pe < 30，且非停牌的股票
    # 最重要的一点是，按照pe正序排列，只取前100只
    daily_cursor = daily.find(
        {'date': adjust_date, 'pe': {'$lt': pe_range[1], '$gt': pe_range[0]},
         'is_trading': True},
        sort=[('pe', sort)],
        projection={'code': True},
        limit=pool_size
    )

    # 拿到所有的股票代码
    codes = [x['code'] for x in daily_cursor]

    # 本期股票列表
    this_phase_codes = []
```

股票池

文件：stock_pool_strategy.py

```
# 如果上期股票代码列表不为空，则查询出上次股票池中正在停牌的股票
    if len(last_phase_codes) > 0:
        suspension_cursor = daily.find(
            # 查询是股票代码、日期和是否为交易，这里is_trading=False
            {'code': {'$in': last_phase_codes}, 'date': adjust_da
            # 只需要使用股票代码
            projection={'code': True}
        )
        # 拿到股票代码
        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

    # 打印出所有停牌的股票代码
    print('上期停牌', flush=True)
    print(this_phase_codes, flush=True)

    # 用新的股票将剩余位置补齐
    this_phase_codes += codes[0: pool_size - 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

    print('最终出票', flush=True)
    print(this_phase_codes, flush=True)

# 返回结果
return all_adjust_dates, adjust_date_codes_dict
```

股票池

文件：stock_pool_strategy.py

```
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


def statistic_stock_pool():
    """
    统计股票池的收益
    """

    # 找到指定时间范围的股票池数据，这里的时间范围可以改变
    adjust_dates, codes_dict = stock_pool('2015-01-01', '2015-12-31')

    # 用DataFrame保存收益，profit是股票池的收益，hs300是用来对比的沪深300的涨跌幅
    df_profit = pd.DataFrame(columns=['profit', 'hs300'])

    # 统计开始的第一天，股票池的收益和沪深300的涨跌幅都是0
    df_profit.loc[adjust_dates[0]] = {'profit': 0, 'hs300': 0}

    # 找到沪深300第一天的值，后面的累计涨跌幅都是和它比较
    hs300_begin_value = daily.find_one({'code': '000300', 'index': True, 'date': adjust_dates[0]})['close']
```

股票池
文件：stock_pool_strategy.py

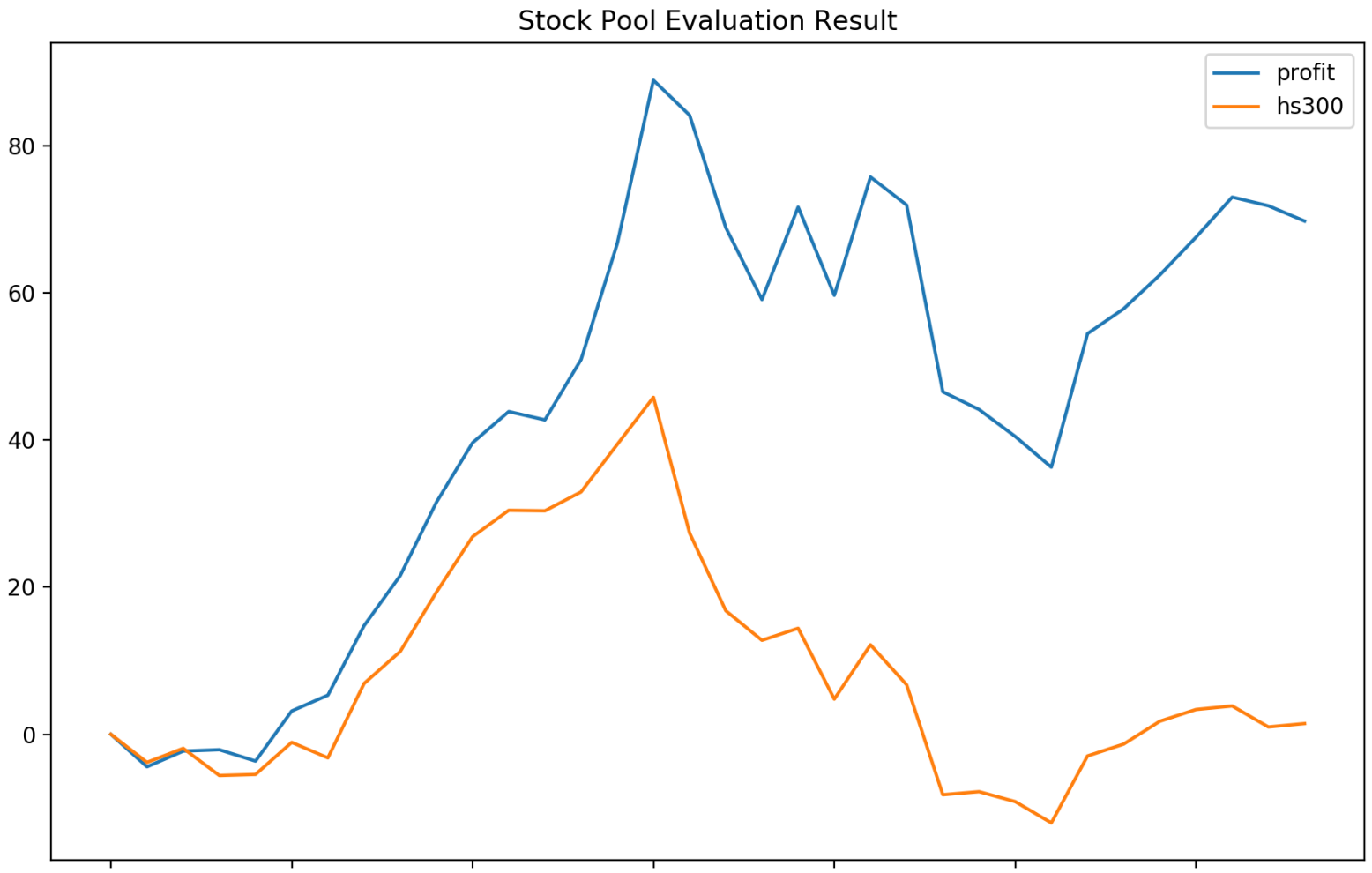
```
"""
通过净值的方式计算累计收益：
累计收益 = 期末净值 - 1
第N期净值的计算方法：
net_value(n) = net_value(n-1) + net_value(n-1) * profit(n)
             = net_value(n-1) * (1 + profit(n))
"""
# 设定初始净值为1
net_value = 1
# 在所有调整日上统计收益，循环时从1开始，因为每次计算要用到当期和上期
for _index in range(1, len(adjust_dates) - 1):
    # 上一期的调整日
    last_adjust_date = adjust_dates[_index - 1]
    # 当期的调整日
    current_adjust_date = adjust_dates[_index]
    # 上一期的股票代码列表
    codes = codes_dict[last_adjust_date]

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

    # 找到上期股票的在当前调整日时的收盘价
    # 1. 这里用的是后复权的价格，保持价格的连续性
    # 2. 当前的调整日，也就是上期的结束日
    sell_daily_cursor = daily_hfq.find(
        {'code': {'$in': codes}, 'date': current_adjust_date},
        # 只需要用到收盘价来计算收益
        projection={'close': True, 'code': True}
    )

    # 初始化所有股票的收益之和
    profit_sum = 0
    # 参与收益统计的股票数量
    count = 0
```

股票池
文件：stock_pool_strategy.py

```
# 循环累加所有股票的收益
        for sell_daily in sell_daily_cursor:
            # 股票代码
            code = sell_daily['code']

            # 如果该股票存在股票池开始时的收盘价，则参与收益统计
            if code in code_buy_close_dict:
                # 选入股票池时的价格
                buy_close = code_buy_close_dict[code]
                # 当前的价格
                sell_close = sell_daily['close']
                # 累加所有股票的收益
                profit_sum += (sell_close - buy_close) / buy_close

                # 参与收益计算的股票数加1
                count += 1

        # 如果股票数量大于0，才统计当期收益
        if count > 0:
            # 计算平均收益
            profit = round(profit_sum / count, 4)

            # 当前沪深300的值
            hs300_close = daily.find_one({'code': '000300', 'index': True, 'date': current_adjust_date})['close']

            # 计算净值和累积收益，放到DataFrame中
            net_value = net_value * (1 + profit)
            df_profit.loc[current_adjust_date] = {
                # 乘以100，改为百分比
                'profit': round((net_value - 1) * 100, 4),
                # 乘以100，改为百分比
                'hs300': round((hs300_close - hs300_begin_value) * 100 / hs300_begin_value, 4)}

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

股票池
文件：stock_pool_strategy.py

股票池收益统计

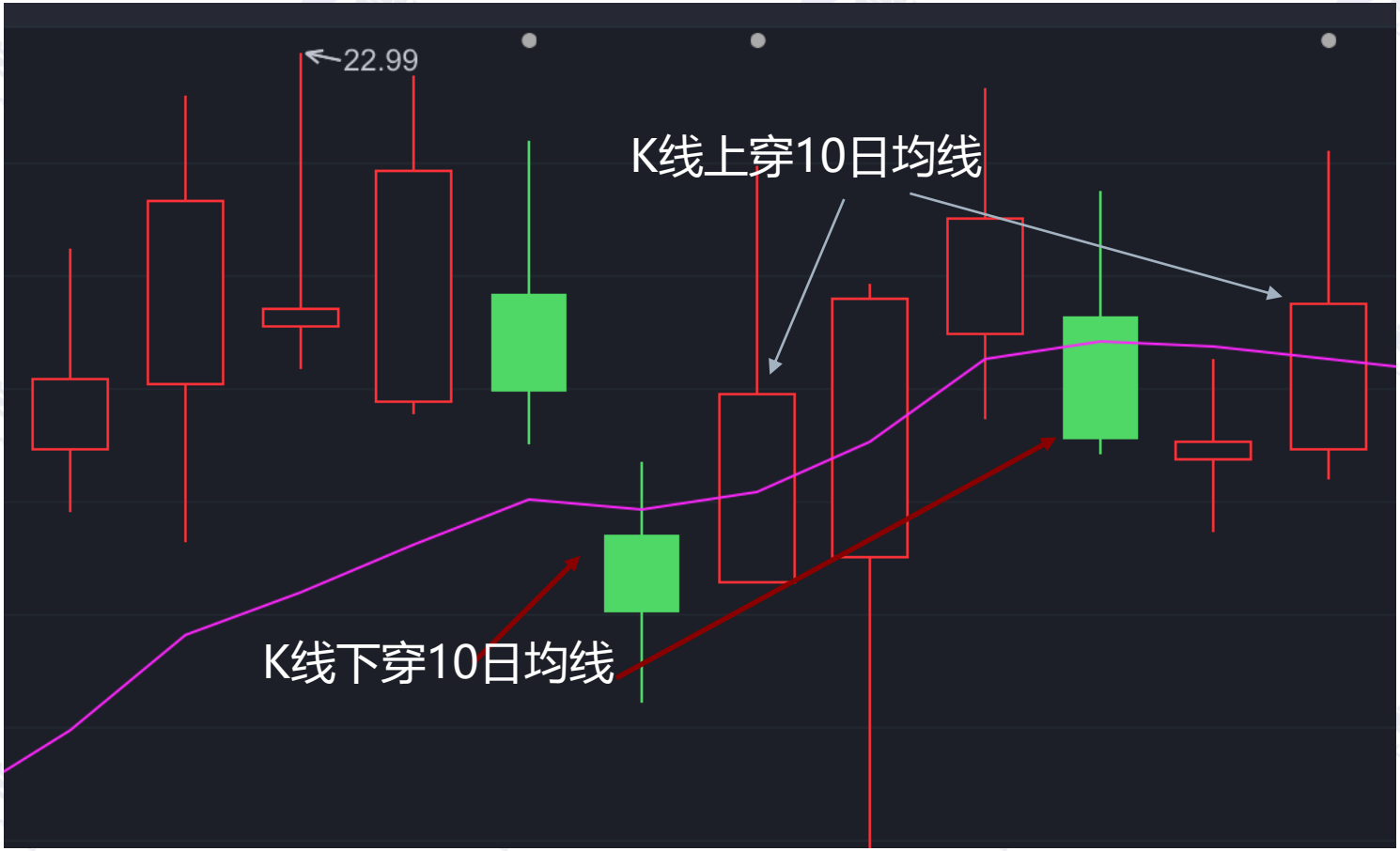


休息一下

5分钟后回来

开发一组交易信号

K线上穿/下穿10日均线



```
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
    for daily in dailies:
        # 10天当中，只要有一天停牌则返回False
        if 'is_trading' not in daily or daily['is_trading'] is False:
            return None

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

    # 计算MA10
    ma_10 = close_sum / 10

    # 判断收盘价和MA10的大小
    post_adjusted_close = current_daily['close']
    differ = post_adjusted_close - ma_10

    # print('计算信号，股票： %s, 收盘价：%7.2f, MA10: %7.2f, 差值：%7.2f' %
    #       (code, post_adjusted_close, ma_10, differ), flush=True)
    if differ > 0:
        return 1
    elif differ < 0:
        return -1
    else:
        return 0
```

比较当日K线和10日均线的关系

文件：backtest.py

```
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
    """

    # 从后复权的日行情数据集中根据股票代码、日期和是否为正常交易的条件查询一条数据，
    # 如果能找到数据，则认为当日股票是正常交易状态，否则为停牌
    current_daily = DB_CONN['daily_hfq'].find_one(
        {'code': code, 'date': _date, 'is_trading': True})

    # 没有找到股票的日行情数据，则认为不符合日线收盘价上穿10日均线的条件
    if current_daily is None:
        print('计算信号，K线上穿MA10，当日没有K线，股票 %s，日期：%s' % (code, _date), flush=True)
        return False

    # 从后复权的日行情数据集中查询是11条数据，因为要计算连着两个交易日的10日均线价格，所以是需要11条数据。
    # 才能保证取到的是邻近的10个交易日的数据。
    daily_cursor = DB_CONN['daily_hfq'].find(
        {'code': code, 'date': {'$lte': _date}},
        limit=11,
        # 10日均线计算的时候是包含当日在内的向前的连续10个交易日的收盘价的平均值，所以要按照时间倒序排列，
        sort=[('date', DESCENDING)],
        # 计算价格均线时，只需要用到价格，并且如果连续10个交易日内有停牌情况，则不进行计算，
        # 所以projection里只需要close和is_trading。
        projection={'code': True, 'close': True, 'is_trading': True}
    )
```

当日K线上穿10日均线
文件：backtest.py

```
    # 从游标取出日行情数据放进列表中
    dailies = [x for x in daily_cursor]

    # 如果数据不满足11个，也就说，无法计算两个交易日的MA10，则认为不符合上穿的条件
    if len(dailies) < 11:
        print('计算信号，K线上穿MA10，前期K线不足，股票 %s，日期：%s' % (code, _date), flush=True)
        return False

    # 查询时是倒序排列的，而计算MA10时是向前10根，所以要将顺序反转
    dailies.reverse()

    # 计算前一个交易日收盘价和MA10的关系
    last_close_2_last_ma10 = compare_close_2_ma_10(dailies[0:10])
    # 计算当前交易日收盘价和MA10的关系
    current_close_2_current_ma10 = compare_close_2_ma_10(dailies[1:])

    # 将关键数据打印出来，以便于比对
    print('计算信号，K线上穿MA10，股票：%s，日期：%s， 前一日 %s，当日：%s' %
          (code, _date, str(last_close_2_last_ma10), str(current_close_2_current_ma10)), flush=True)

    # 前一日或者当日任意一天的收盘价和MA10的关系不存在，则都认为不符合上穿的条件
    if last_close_2_last_ma10 is None or current_close_2_current_ma10 is None:
        return False

    # 只有前一日收盘价小于等于10，且当前交易日的收盘价大于MA10，则认为当日收盘价上穿MA10
    is_break = (last_close_2_last_ma10 <= 0) & (current_close_2_current_ma10 == 1)

    print('计算信号，K线上穿MA10，股票：%s，日期：%s， 前一日 %s，当日：%s，突破：%s' %
          (code, _date, str(last_close_2_last_ma10), str(current_close_2_current_ma10), str(is_break)),
flush=True)

    # 返回判断结果
    return is_break
```

当日K线上穿10日均线
文件：backtest.py

```
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
    """
    # 从后复权的日行情数据集中根据股票代码、日期和是否为正常交易的条件查询一条数据，
    # 如果能找到数据，则认为当日股票是正常交易状态，否则为停牌
    current_daily = DB_CONN['daily'].find_one(
        {'code': code, 'date': _date, 'is_trading': True})

    # 没有找到股票的日行情数据，则认为不符合日线收盘价下穿10日均线的条件
    if current_daily is None:
        print('计算信号，K线下穿MA10，当日没有K线，股票 %s，日期: %s' % (code, _date), flush=True)
        return False

    # 从后复权的日行情数据集中查询是11条数据，因为要计算连着两个交易日的10日均线价格，所以是需要11条数据。
    # 才能保证取到的是邻近的10个交易日的数据。
    daily_cursor = DB_CONN['daily_hfq'].find(
        {'code': code, 'date': {'$lte': _date}},
        limit=11,
        # 10日均线计算的时候是包含当日在内的向前的连续10个交易日的收盘价的平均值，所以要按照时间倒序排列，
        sort=[('date', DESCENDING)],
        # 计算价格均线时，只需要用到价格，并且如果连续10个交易日内有停牌情况，则不进行计算，
        # 所以projection里只需要close和is_trading。
        projection={'code': True, 'close': True, 'is_trading': True}
    )

    # 从游标取出日行情数据放进列表中
    dailies = [x for x in daily_cursor]
```

当日K线下穿10日均线
文件：backtest.py

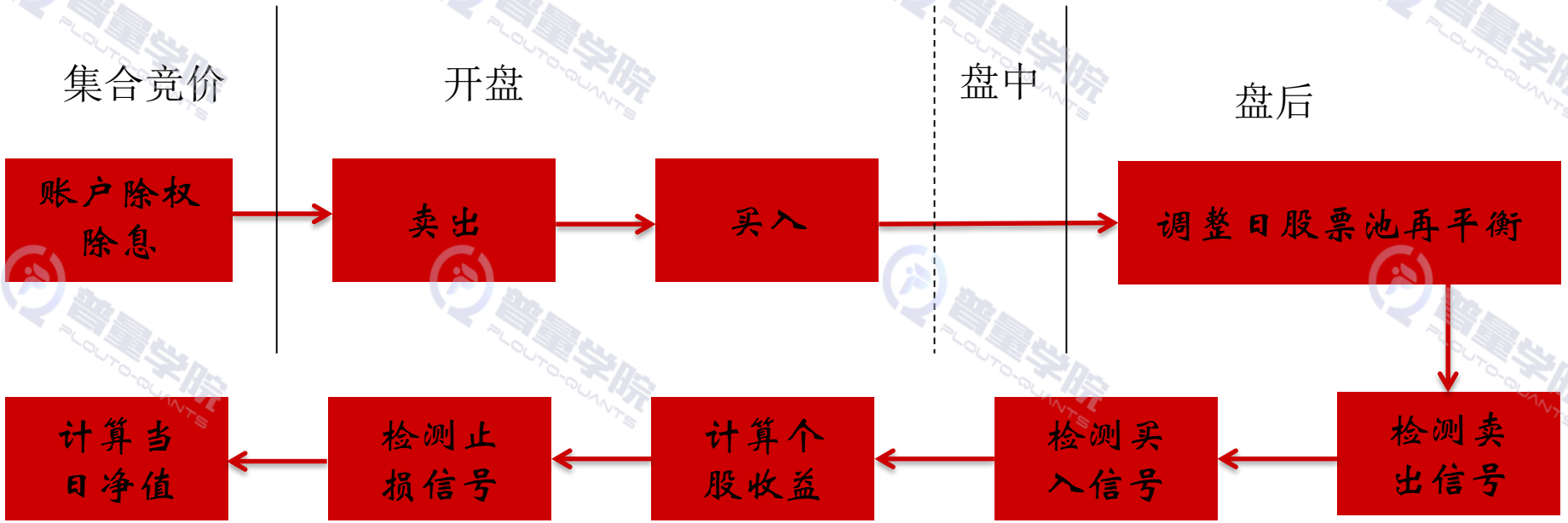
```
    # 如果数据不满足11个，也就说，无法计算两个交易日的MA10，则认为不符合下穿的条件
    if len(dailies) < 11:
        print('计算信号，K线下穿MA10，前期K线不足，股票 %s，日期：%s' % (code, _date), flush=True)
        return False

    # 查询时是倒序排列的，而计算MA10时是向前10根，所以要将顺序反转
    dailies.reverse()

    # 计算前一个交易日收盘价和MA10的关系
    last_close_2_last_ma10 = compare_close_2_ma_10(dailies[0:10])
    # 计算当前交易日收盘价和MA10的关系
    current_close_2_current_ma10 = compare_close_2_ma_10(dailies[1:])

    # 前一日或者当日任意一天的收盘价和MA10的关系不存在，则都认为不符合下穿的条件
    if last_close_2_last_ma10 is None or current_close_2_current_ma10 is None:
        return False

    # 只有前一日收盘价大于等于10，且当前交易日的收盘价小于MA10，则认为当日收盘价下穿MA10
    is_break = (last_close_2_last_ma10 >= 0) & (current_close_2_current_ma10 == -1)

    print('计算信号，K线下穿MA10，股票：%s，日期：%s， 前一日 %s，当日：%s，突破：%s' %
          (code, _date, str(last_close_2_last_ma10), str(current_close_2_current_ma10),
str(is_break)), flush=True)

    return is_break
```

当日K线下穿10日均线

文件：backtest.py

实现低PE策略的回测

回测基本流程


集合竞价
开盘
盘中
盘后
账户除权除息
卖出
买入
调整日股票池再平衡
检测卖出信号
检测买入信号
计算个股收益
检测止损信号
计算当日净值

补充其他条件

- 总资金
  - 1000万元
- 头寸分配
  - 均分
  - 每只20万

```
def backtest(begin_date, end_date):
    """
    策略回测。结束后打印出收益曲线(沪深300基准)、年化收益、最大回撤、

    :param begin_date: 回测开始日期
    :param end_date: 回测结束日期
    """

    # 初始现金1000万
    cash = 1E7
    # 单只股票的仓位是20万
    single_position = 2E5

    # 时间为key的净值、收益和同期沪深基准
    df_profit = pd.DataFrame(columns=['net_value', 'profit', 'hs300'])

    # 获取回测开始日期和结束之间的所有交易日，并且是按照正序排列
    all_dates = get_trading_dates(begin_date, end_date)

    # 获取沪深300的在回测开始的第一个交易日的值
    hs300_begin_value = DB_CONN['daily'].find_one(
        {'code': '000300', 'index': True, 'date': all_dates[0]},
        projection={'close': True})['close']

    # 获取回测周期内的股票池数据，
    # adjust_dates：正序排列的调整日列表；
    # date_codes_dict：  调整日和当期的股票列表组成的dict，key是调整日，value是股票代码列表
    adjust_dates, date_codes_dict = stock_pool(begin_date, end_date)
```

回测流程-初始化

文件：backtest.py

```
# 股票池上期股票代码列表
last_phase_codes = None
# 股票池当期股票代码列表
this_phase_codes = None
# 待卖的股票代码集合
to_be_sold_codes = set()
# 待买的股票代码集合
to_be_bought_codes = set()
# 持仓股票dict，key是股票代码，value是一个dict，
# 三个字段分别为：cost - 持仓成本，volume - 持仓数量，last_value：前一天的市值
holding_code_dict = dict()
# 前一个交易日
last_date = None
# 在交易日的顺序，一天天完成信号检测
for _date in all_dates:
    print('Backtest at %s.' % _date)

    # 当期持仓股票的代码列表
    before_sell_holding_codes = list(holding_code_dict.keys())
```

回测流程-初始化

文件：backtest.py

```
if last_date is not None and len(before_sell_holding_codes) > 0:
    # 从daily数据集中查询出所有持仓股的前一个交易日的复权因子
    last_daily_cursor = DB_CONN['daily'].find(
        {'code': {'$in': before_sell_holding_codes}, 'date': last_date, 'index': False},
        projection={'code': True, 'au_factor': True})

    # 构造一个dict，key是股票代码，value是上一个交易日的复权因子
    code_last_aufactor_dict = dict([(daily['code'], daily['au_factor']) for daily in last_daily_cursor])

    # 从daily数据集中查询出所有持仓股的当前交易日的复权因子
    current_daily_cursor = DB_CONN['daily'].find(
        {'code': {'$in': before_sell_holding_codes}, 'date': _date, 'index': False},
        projection={'code': True, 'au_factor': True})

    # 一只股票一只股票进行处理
    for current_daily in current_daily_cursor:
        # 当前交易日的复权因子
        current_aufactor = current_daily['au_factor']
        # 股票代码
        code = current_daily['code']
        # 从持仓股中找到该股票的持仓数量
        last_volume = holding_code_dict[code]['volume']
        # 如果该股票存在前一个交易日的复权因子，则对持仓股数量进行处理
        if code in code_last_aufactor_dict:
            # 上一个交易日的复权因子
            last_aufactor = code_last_aufactor_dict[code]
            # 计算复权因子变化后的持仓股票数量，如果复权因子不发生变化，那么持仓数量是不发生变化的
            # 相关公式是：
            # 市值不变：last_close * last_volume = pre_close * current_volume
            # 价格的关系：last_close * last_aufactor = pre_close * current_aufactor
            # 转换之后得到下面的公式：
            current_volume = int(last_volume * (current_aufactor / last_aufactor))
            # 改变持仓数量
            holding_code_dict[code]['volume'] = current_volume
            print('持仓量调整：%s, %6d, %10.6f, %6d, %10.6f' %
                  (code, last_volume, last_aufactor, current_volume, current_aufactor))
```

回测流程-持仓股除权除息

文件：backtest.py

```
# 如果有待卖股票，则继续处理
if len(to_be_sold_codes) > 0:
    # 从daily数据集中查询所有待卖股票的开盘价，这里用的不复权的价格，以模拟出真实的交易情况
    sell_daily_cursor = DB_CONN['daily'].find(
        {'code': {'$in': list(to_be_sold_codes)}, 'date': _date, 'index': False, 'is_trading': True},
        projection={'open': True, 'code': True}
    )

    # 一只股票一只股票处理
    for sell_daily in sell_daily_cursor:
        # 待卖股票的代码
        code = sell_daily['code']
        # 如果股票在持仓股里
        if code in before_sell_holding_codes:
            # 获取持仓股
            holding_stock = holding_code_dict[code]
            # 获取持仓数量
            holding_volume = holding_stock['volume']
            # 卖出价格为当日开盘价
            sell_price = sell_daily['open']
            # 卖出获得金额为持仓量乘以卖出价格
            sell_amount = holding_volume * sell_price
            # 卖出得到的资金加到账户的可用现金上
            cash += sell_amount

            # 获取该只股票的持仓成本
            cost = holding_stock['cost']
            # 计算持仓的收益
            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))

            # 删除该股票的持仓信息
            del holding_code_dict[code]
            to_be_sold_codes.remove(code)
```

回测流程－卖出

文件：backtest.py

```
if len(to_be_bought_codes) > 0:
    # 获取所有待买入股票的开盘价
    buy_daily_cursor = DB_CONN['daily'].find(
        {'code': {'$in': list(to_be_bought_codes)}, 'date': _date, 'is_trading': True, 'index':
False},
        projection={'code': True, 'open': True}
    )

    # 处理所有待买入股票
    for buy_daily in buy_daily_cursor:
        # 判断可用资金是否够用
        if cash > single_position:
            # 获取买入价格
            buy_price = buy_daily['open']
            # 获取股票代码
            code = buy_daily['code']
            # 获取可买的数量，数量必须为正手数
            volume = int(int(single_position / buy_price) / 100) * 100
            # 买入花费的成本为买入价格乘以实际的可买入数量
            buy_amount = buy_price * volume
            # 从现金中减去本次花费的成本
            cash -= buy_amount
            # 增加持仓股中
            holding_code_dict[code] = {
                'volume': volume,          # 持仓量
                'cost': buy_amount,        # 持仓成本
                'last_value': buy_amount   # 初始前一日的市值为持仓成本
            }

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

回测流程 – 买入

文件：backtest.py

回测流程 – 第二日待交易
文件：backtest.py

```
if _date in adjust_dates:
    print('股票池调整日：%s，备选股票列表：' % _date, flush=True)

    # 如果上期股票列表存在，也就是当前不是第一期股票，则将
    # 当前股票列表设为上期股票列表
    if this_phase_codes is not None:
        last_phase_codes = this_phase_codes

    # 获取当期的股票列表
    this_phase_codes = date_codes_dict[_date]
    print(this_phase_codes, flush=True)

    # 如果存在上期的股票列表，则需要找出被调出的股票列表
    if last_phase_codes is not None:
        # 找到被调出股票池的股票列表
        out_codes = find_out_stocks(last_phase_codes, this_phase_codes)
        # 将所有被调出的且是在持仓中的股票添加到待卖股票集合中
        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)
```

回测流程－计算净值
文件：backtest.py

```
# 计算总资产
total_value = 0
holding_daily_cursor = DB_CONN['daily'].find(
    {'code': {'$in': holding_codes}, 'date': _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
    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_CONN['daily'].find_one(
    {'code': '000300', 'index': True, 'date': _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] = {
    'net_value': round(total_capital / 1e7, 2),
    'profit': round(100 * (total_capital - 1e7) / 1e7, 2),
    'hs300': round(100 * (hs300_current_value - hs300_begin_value) / hs300_begin_value, 2)
}
```

策略的评价指标—年化收益

$$Years = \frac{TradingDays}{AnnualTradingDays} \quad (1)$$

$$NetValue = (1 + AnnualProfit)^{Years} \quad (2)$$

$$AnnualProfit = \sqrt[Years]{NetValue} - 1 \quad (3)$$

$$AnnualProfit = NetValue^{\frac{1}{Years}} - 1 \quad (4)$$

```
def compute_annual_profit(trading_days, net_value):
    """
    计算年化收益
    """

    annual_profit = 0
    if trading_days > 0:
        # 计算年数
        years = trading_days/245
        # 计算年化收益
        annual_profit = pow(net_value, 1/years) - 1

    annual_profit = round(annual_profit * 100, 2)

    return annual_profit
```

年化收益

策略的评价指标—夏普比率

$$ProfitMean = \frac{1}{N}\sum_{i=0}^{N} Profit_i$$

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

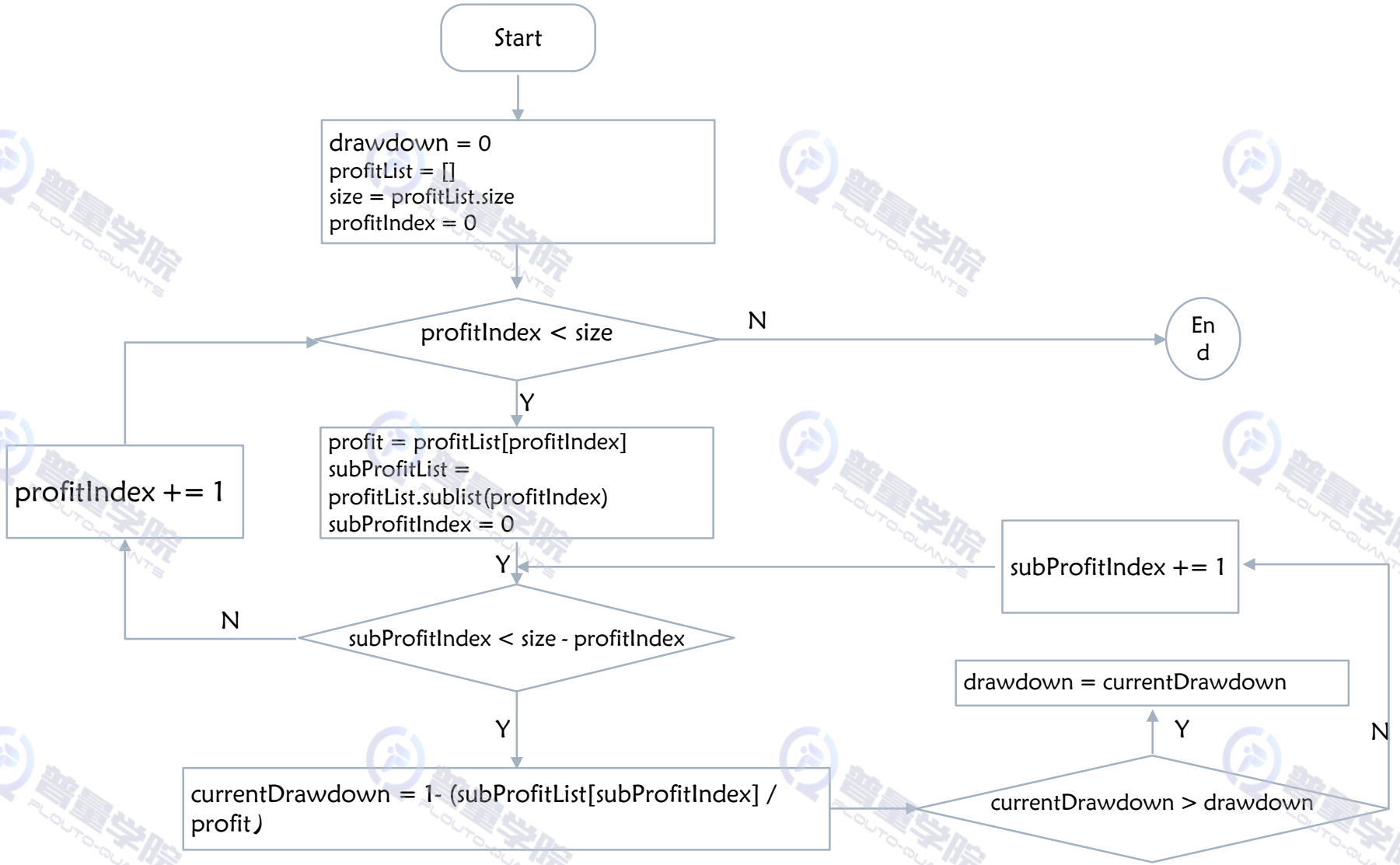
$$ProfitStd = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(Profit_i - ProfitMean)^2}$$

$$SharpeRatio = \frac{AnnalProfit - R_f}{ProfitStd}$$

R_f - 无风险收益

夏普比率

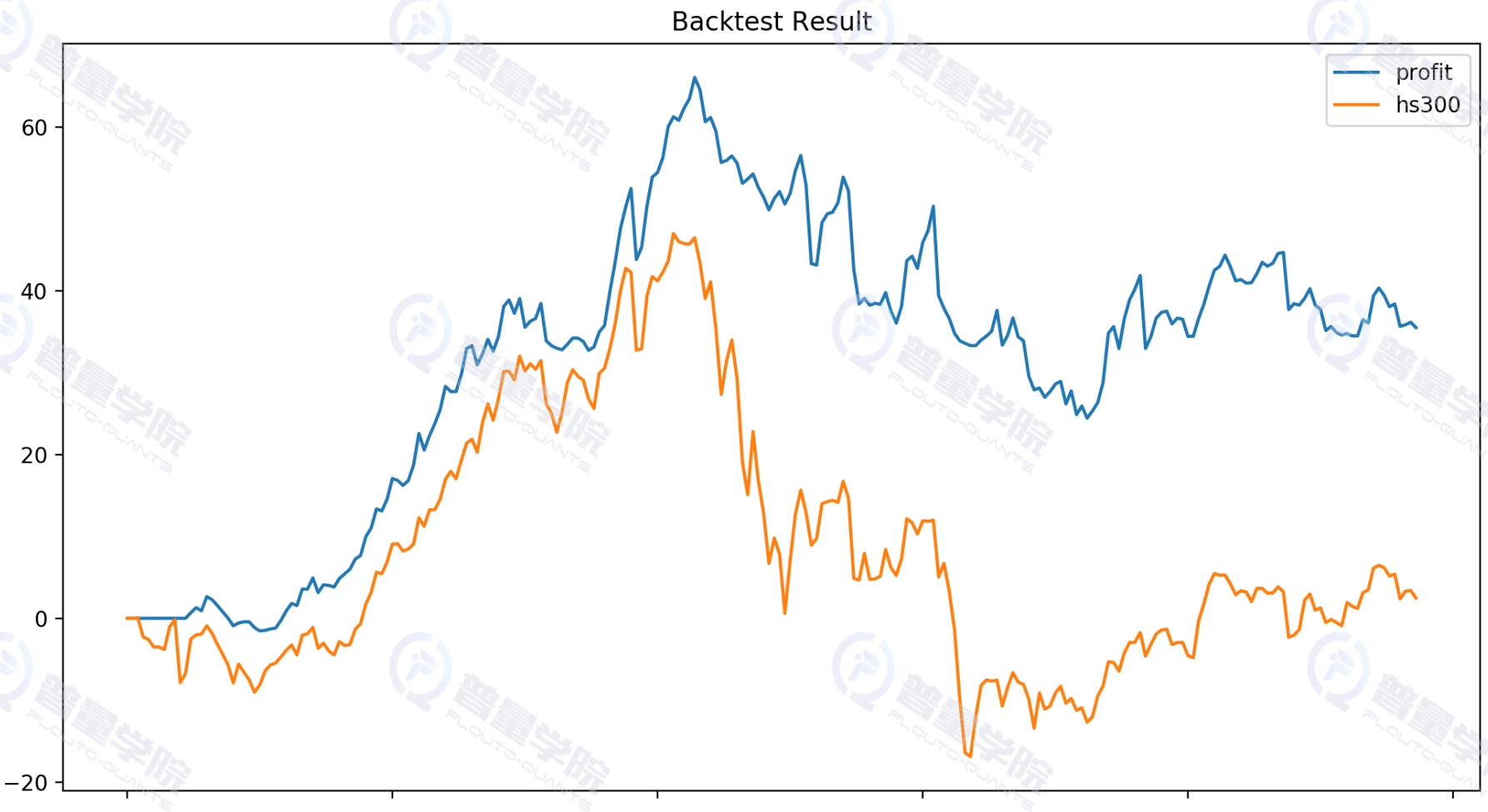
```
def compute_sharpe_ratio(net_values):
    """
    计算夏普比率
    :param net_values: 净值列表
    """

    # 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'], 1/2)

    #年化收益
    annual_profit = compute_annual_profit(trading_days, net_values[-1])
  sharpe_ratio = (annual_profit - 4.75)/profit_std # 夏普比率

  return annual_profit, sharpe_ratio
```


Start
drawdown = 0
profitList = []
size = profitList.size
profitIndex = 0
profitIndex < size
N
End
Y
profit = profitList[profitIndex]
subProfitList =
profitList.sublist(profitIndex)
subProfitIndex = 0
profitIndex += 1
Y
subProfitIndex += 1
N
subProfitIndex < size - profitIndex
drawdown = currentDrawdown
Y
Y
N
currentDrawdown = 1- (subProfitList[subProfitIndex] /
profit)
currentDrawdown > drawdown

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

        index += 1

    return max_drawdown
```

最大回撤

回测结果

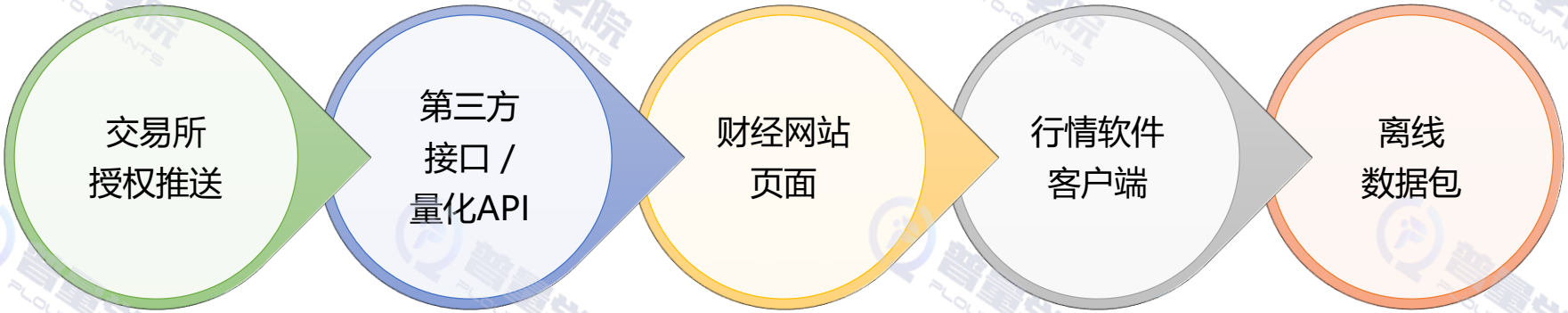


讲解的知识点（内部用，不对外）

- 各个数据源的介绍，按照逐字稿

预计花费时间3分钟左右

行情数据来源（例）



上海
证券交易所
深圳
证券交易所

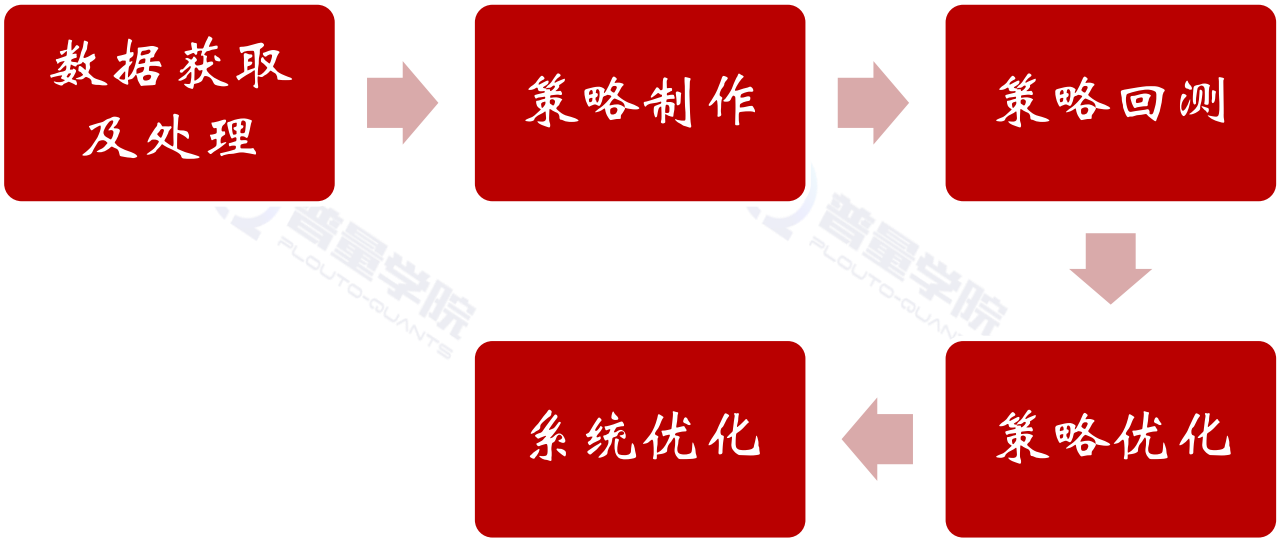
Wind
东财Choice
掘金量化
Tushare
通联数据
JQData

新浪财经
东方财富
巨潮资讯

通达信
东方财富
同花顺
飞狐交易师
图表交易软件
券商软件

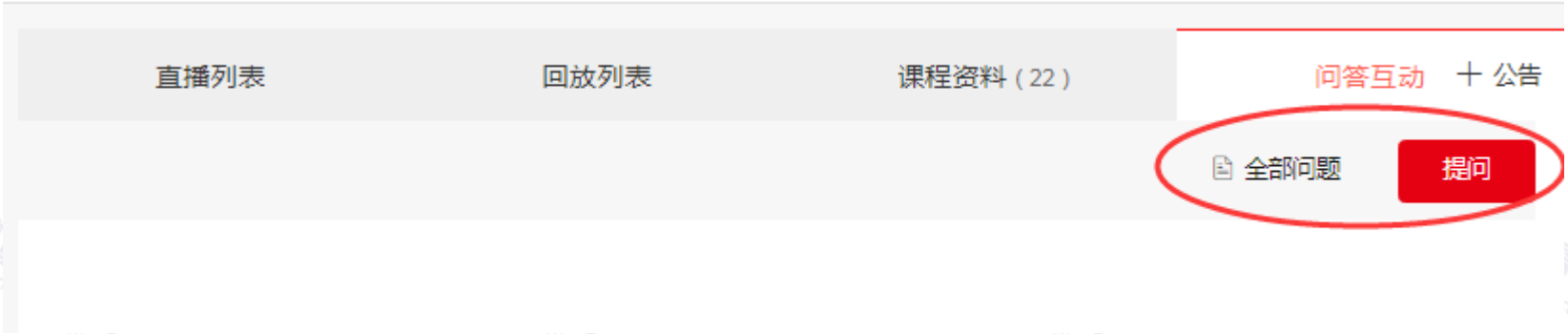
交易所
Wind
淘宝
预测者

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