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

第四课

从0实现一个模块化交易系统

系统化构建量化交易体系：

模块2：**搭建自己的股票回测及交易平台**

内容介绍

- 自建交易系统的必要性
- 回测框架的Lite版本实现
- 模块化交易系统框架
- 开发环境准备
- 模块化系统框架实现
- 利用ECharts实现交互式净值曲线

为什么要重复发明轮子？

自建交易系统的必要性

第三方平台的局限性

- 黑箱和隐患：
  - 数据中断/缺项、未来函数等
  - Bug的不可预知性
- 可扩展性：
  - 不能直接引用外部算法结果
  - 回测速度慢
  - 权限受限
- 灵活性：
  - 代码复用程度低
  - 个性化数据的重复处理

从0到1

回测框架的Lite版本实现

完成的工作

- 实现一个示例策略
- 完成示例策略的回测

策略描述

- 股票池
  - 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
- 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

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

获取财报数据

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

import tushare as ts
from pandas import DataFrame
import matplotlib.pyplot as plt

def get_code_reports():
    """
    获取回测周期内的年度财报数据，组合成一个dict数据结构，key是股票代码，
       value是一个按照报告发布日期排序的列表，列表内的元素也是一个dict
    {'eps': 每股收益, 'announced_date': 公告日期}
    """

  # 这个tuple包含了三个元素，前两个分别是用来获取年度财报时的参数年份和季度，
    # 后面是一个发布年度财报的年份，因为一般情况下发布财报都是在第二年的4月底之前，
    # 所以这个年份比财报的年份晚一年
    report_date_tuples = [(2013, 4, 2014), (2014, 4, 2015),
         (2015, 4, 2016), (2016, 4, 2017)]

  # 要返回的数据结构
    code_report_dict = dict()

  # 循环获取所有指定报告期的数据
    for report_date_tuple in report_date_tuples:
     # 从Tushare获取年报数据
        df_reports = ts.get_report_data(report_date_tuple[0],
report_date_tuple[1])
```

获取财报数据

```
# 只需要股票代码、每股收益和公告日期三个字段
    codes = df_reports['code']
epses = df_reports['eps']
announced_dates = df_reports['report_date']

# 这个是报告发布时的年度
    announced_year = str(report_date_tuple[2])

# 拿到已经缓存的股票代码集合
    codes_of_cached_reports = set(code_report_dict.keys())

# 循环获取所有数据
    for report_index in df_reports.index:
   code = codes[report_index]
   eps = epses[report_index]
   announced_date = announced_dates[report_index]

   print('%s %5.2f %d %s' %
              (code, eps, report_date_tuple[0], announced_date), flush=True)

   # 如果eps是非数字，或者发布日期的月份超过了4月，就不作处理，因为股票在上市前
        # 也会发布财报，那么这个财报的发布日期可能不是定期报告所规定的时间范围，
        # 那么对这种上市之前的数据暂时不予处理
        if str(eps) != 'nan' and int(announced_date[0:2]) <= 4:
     # 组合成完整的公告年月日
           announced_date = announced_year + '-' + announced_date
     print('%s %5.2f %s' % (code, eps, announced_date), flush=True)
```

```
    # 如果当前股票不在需要返回的数据结构中，则添加到其中
            if code not in codes_of_cached_reports:
        code_report_dict[code] = []
        codes_of_cached_reports.add(code)

    # 将eps和公告日期添加到列表中
            code_report_dict[code].append(
                    {'eps': eps, 'announced_date': announced_date})

# 返回获取的数据
return code_report_dict
```

获取财报数据

```
def get_option_codes(code_report_dict, rebalance_date):
    """
    找到某个调整日符合股票池条件的股票列表
      :param code_report_dict: 股票对应的财报列表
      :param rebalance_date: 在平衡日期
      :return: 股票代码列表
      """

    # 如果股票和每股收益的dict是空的，则重新获取
       report_codes = list(code_report_dict.keys())
    if len(report_codes) == 0:
        get_code_report()
        report_codes = list(code_report_dict.keys())

    # 找到当期符合条件的EPS，股票代码和EPS
    code_eps_dict = dict()
    for code in report_codes:
        # 因为财报是按照公告日期从早到晚排列的，所以顺序查找
              reports = code_report_dict[code]
        # 用来保存最后一个公告日期小于等于当前日期的财报
             last_report = None
        for report in reports:
            announced_date = report['announced_date']
            # 如果公告日期大于当前调整日，则结束循环
                      if announced_date > rebalance_date:
                break
```

找到某个调整日符合股票池条件的备选股

找到某个调整日符合股票池条件的备选股

```
            last_report = report

        # 如果找到了正确时间范围的年报，并且eps大于0，才保留
               if last_report is not None and last_report['eps'] > 0 :
            print('%s, %s, %s, %5.2f' %
             (code, rebalance_date, last_report['announced_date'],
last_report['eps']), flush=True)

            code_eps_dict[code] = last_report['eps']


    # 只在符合EPS>0的范围，计算PE，并筛选股票
       validated_codes = list(code_eps_dict)

    df_pe = DataFrame(columns=['close', 'eps'])
    for code in validated_codes:
        # 用不复权的价格
              daily_k = ts.get_k_data(code,
                    autype=None, start=rebalance_date, end=rebalance_date)
        # 如果当前是停牌，就获取不到股价信息，那么不参与排名
              if daily_k.size > 0:
            close = daily_k.loc[daily_k.index[0]]['close']
            df_pe.loc[code] = {'eps': code_eps_dict[code], 'close': close}
            print('%s %6.2f' % (code, close), flush=True)
        else:
            print('%s 停牌' % (code), flush=True)
```

找到某个调整日符合股票池条件的备选股

```
# 计算PE
df_pe['pe'] = df_pe['close'] / df_pe['eps']
# 从小到大排序
   df_pe.sort_values('pe', ascending=True, inplace=True)
# 只保留小于30的数据
   df_pe = df_pe[df_pe['pe'] < 30]

# 返回排名靠前的100只股票代码
return list(df_pe.index)[0:100]
```

股票池

```
def stock_pool(begin_date, end_date):
    """
    实现股票池选股逻辑，找到指定日期范围的候选股票
        条件：0 < PE < 30，按从小到大排序，剔除停牌后，取前100个；再平衡周期：7个交易日
        :param begin_date: 开始日期
        :param end_date: 结束日期
        :return: tuple，再平衡的日期列表，以及一个dict(key: 再平衡日, value: 当期的股票列表)
    """

    # 获取财务数据
        code_report_dict = get_code_reports()

    # 股票池的再平衡周期
        rebalance_interval = 7

    # 因为上证指数没有停牌不会缺数，所以用它作为交易日历，
        szzz_hq_df = ts.get_k_data('000001', index=True, start=begin_date, end=end_date)
    all_dates = list(szzz_hq_df['date'])

    # 调整日和其对应的股票
        rebalance_date_codes_dict = dict()
    rebalance_dates =[]
```

股票池

```
  # 保存上一期的股票池
    last_phase_codes = []
  # 所有的交易日数
    dates_count = len(all_dates)
  # 用再平衡周期作为步长循环
    for index in range(0, dates_count, rebalance_interval):
      # 当前的调整日
          rebalance_date = all_dates[index]

      # 获取本期符合条件的备选股票
          this_phase_option_codes = get_option_codes(code_report_dict,
rebalance_date)

      # 本期入选的股票代码列表
          this_phase_codes = []

      # 找到在上一期的股票池，但是当前停牌的股票，保留在当期股票池中
          if len(last_phase_codes) > 0:
         for code in last_phase_codes:
             daily_k = ts.get_k_data(code, autype=None,
                                     start=rebalance_date, end=rebalance_date)
             if daily_k.size == 0:
                 this_phase_codes.append(code)

      print('上期停牌的股票：', flush=True)
      print(this_phase_codes, flush=True)
```

股票池

```
    # 剩余的位置用当前备选股票的
        option_size = len(this_phase_option_codes)
    if option_size > (100 - len(this_phase_codes)):
        this_phase_codes += this_phase_option_codes[0:100-len(this_phase_codes)]
    else:
        this_phase_codes += this_phase_option_codes

    # 当期股票池作为下次循环的上期股票池
        last_phase_codes = this_phase_codes

    # 保存到返回结果中
        rebalance_date_codes_dict[rebalance_date] = this_phase_codes
      rebalance_dates.append(rebalance_date)

    print('当前最终的备选票：%s' % rebalance_date, flush=True)
    print(this_phase_codes, flush=True)

return (rebalance_dates, rebalance_date_codes_dict)
```

股票池收益统计

```
def statistic_stock_pool_profit():
    """
    统计股票池的收益
       """
    # 设定评测周期
       # rebalance_dates, codes_dict = stock_pool('2008-01-01', '2018-06-30')
    rebalance_dates, codes_dict = stock_pool('2015-01-01', '2015-12-31')

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

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

    # 获取沪深300在统计周期内的第一天的值
        hs300_k = ts.get_k_data('000300', index=True,
                start=rebalance_dates[0], end=rebalance_dates[0])
    hs300_begin_value = hs300_k.loc[hs300_k.index[0]]['close']

    # 通过净值计算累计收益
       net_value = 1
    for _index in range(1, len(rebalance_dates) - 1):
        last_rebalance_date = rebalance_dates[_index - 1]
        current_rebalance_date = rebalance_dates[_index]
        # 获取上一期的股票池
                codes = codes_dict[last_rebalance_date]
```

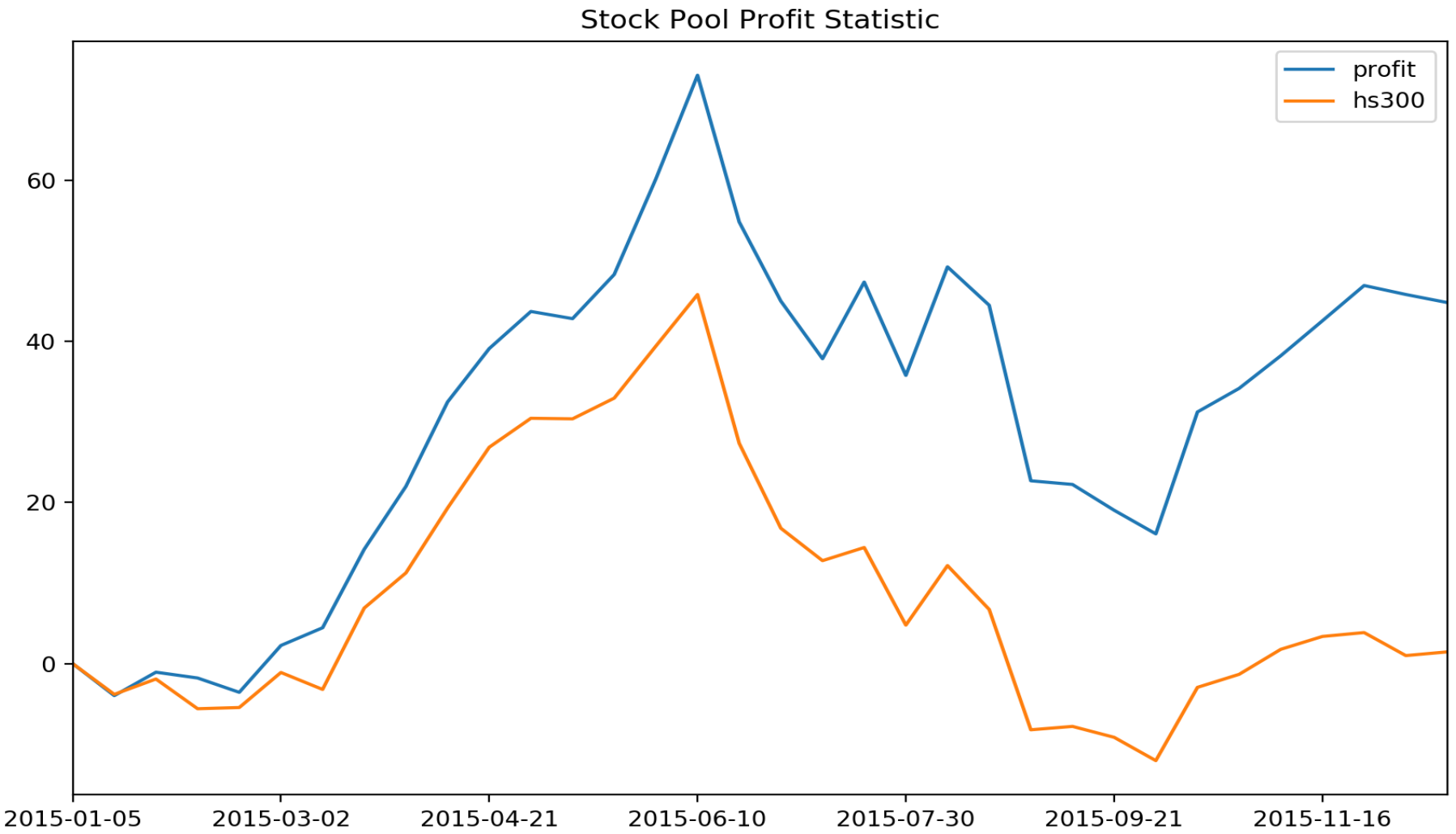
股票池收益统计

```
# 统计当期的收益
    profit_sum = 0
   prifit_code_count = 0 # 参与统计收益的股票个数
   for code in codes:
  daily_ks = ts.get_k_data(code, autype='hfq',
                start=last_rebalance_date, end=current_rebalance_date)

 index_size = daily_ks.index.size
 # 如果没有数据，则跳过，长期停牌
       if index_size == 0:
     continue

   # 买入价
   in_price = daily_ks.loc[daily_ks.index[0]]['close']
   # 卖出价
   out_price = daily_ks.loc[daily_ks.index[index_size - 1]]['close']
       profit_sum += (out_price - in_price)/in_price
    prifit_code_count += 1

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

股票池收益统计

```
        hs300_k_current = ts.get_k_data('000300', index=True,
                        start=current_rebalance_date, end=current_rebalance_date)
        hs300_close = hs300_k_current.loc[hs300_k_current.index[0]]['close']

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

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

股票池收益统计


Stock Pool Profit Statistic
profit
hs300
60
40
20
0
2015-01-05
2015-03-02
2015-04-21
2015-06-10
2015-07-30
2015-09-21
2015-11-16

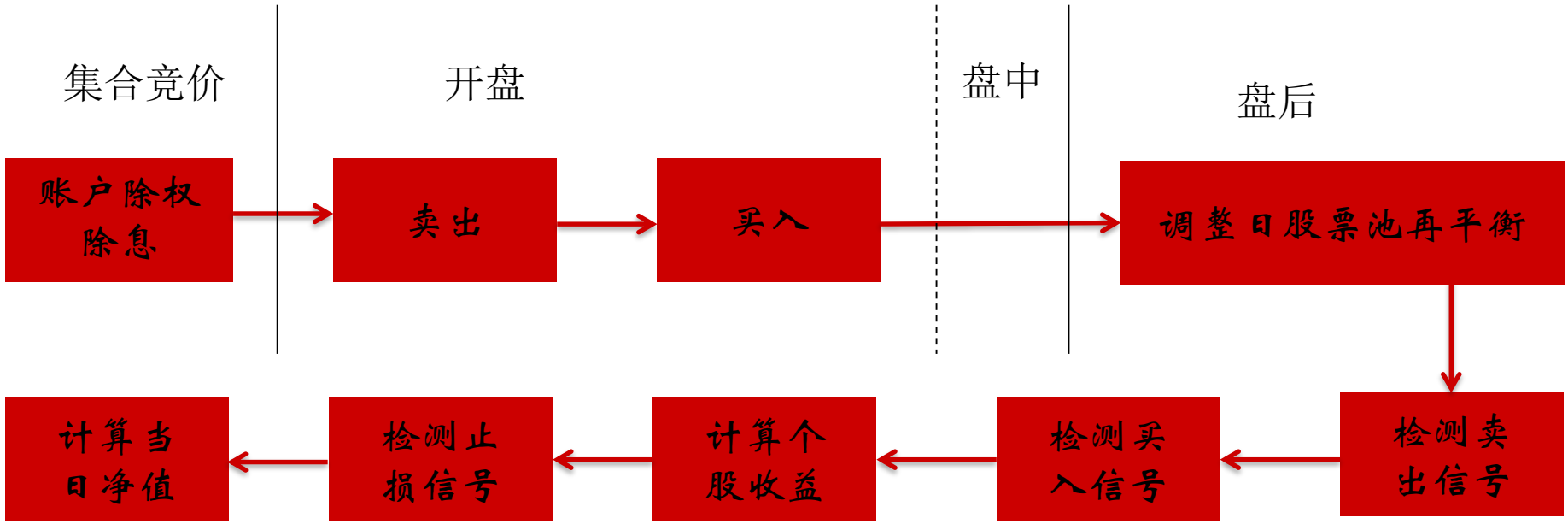
当日K线上穿10日均线

```
def is_k_up_break_ma10(code, begin_date, end_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 begin_date: 开始日期
    :param end_date: 结束日期
    :return: True/False
    """

    # 如果没有指定开始日期和结束日期，则直接返回False
    if begin_date is None or end_date is None:
        return False

    # 如果股票当日停牌或者是下跌，则返回False
    daily_ks = ts.get_k_data(code, autype='hfq', start=begin_date, end=end_date)

    # 需要判断两日的K线和10日均线的相对位置，所以如果K线数不满足11个，
    # 也就是无法计算两个MA10，则直接返回False
    index_size = daily_ks.index.size
    if index_size < 11:
        return False

    daily_ks['ma'] = daily_ks['close'].rolling(10).mean()  # 计算MA10
    daily_ks['delta'] = daily_ks['close'] - daily_ks['ma']  # 比较收盘价和MA10的关系

    return daily_ks.loc[daily_ks.index[9]]['delta'] <= 0 <
daily_ks.loc[daily_ks.index[10]]['delta']
```

```
def is_k_down_break_ma10(code, begin_date, end_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 begin_date: 开始日期
    :param end_date: 结束日期
    :return: True/False
    """

    # 如果没有指定开始日期和结束日期，则直接返回False
    if begin_date is None or end_date is None:
        return False

    daily_ks = ts.get_k_data(code, autype='hfq', start=begin_date, end=end_date)

    # 需要判断两日的K线和10日均线的相对位置，所以如果K线数不满足11个，
    # 也就是无法计算两个MA10，则直接返回False
    if daily_ks.index.size < 11:
        return False

    daily_ks['ma'] = daily_ks['close'].rolling(10).mean()    # 计算MA10
    daily_ks['delta'] = daily_ks['close'] - daily_ks['ma']   # 计算收盘价和MA10的差

    return daily_ks.loc[daily_ks.index[9]]['delta'] >= 0 >
daily_ks.loc[daily_ks.index[10]]['delta']
```

当日K线下穿10日均线

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

几个需要注意的问题

- 除权除息
- 停牌
- 涨跌停
  - 涨停几乎无法买入
  - 跌停几乎无法卖出
- 交易费和冲击成本
- 买入量：整手数
- 卖出在前，买入在后

补充其他条件

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

缓存回测周期的日线数据

```
# 获取股票池数据
rebalance_dates, date_codes_dict = stock_pool(begin_date, end_date)

# 获取回测周期内股票池内所有股票的收盘价和前收价
all_option_code_set = set()
for rebalance_date in rebalance_dates:
    for code in date_codes_dict[rebalance_date]:
        all_option_code_set.add(code)

# 缓存股票的日线数据
code_daily_dict = dict()
for code in all_option_code_set:
    dailies_df = ts.get_k_data(code, autype=None, start=begin_date, end=end_date)
    dailies_hfq_df = ts.get_k_data(code, autype='hfq', start=begin_date,
end=end_date)
    # 计算复权因子
       dailies_df['au_factor'] = dailies_hfq_df['close'] / dailies_df['close']
    dailies_df.set_index(['date'], inplace=True)

    code_daily_dict[code] = dailies_df
```

处理除权除息

```
# 当期持仓股票列表
before_sell_holding_codes = list(holding_code_dict.keys())

# 处理持仓股的除权除息
if last_date is not None and len(before_sell_holding_codes) > 0:
    for code in before_sell_holding_codes:
        try:
            dailies = code_daily_dict[code]
            # 上一个交易日的复权因子
            current_au_factor = dailies.loc[_date]['au_factor']
            before_volume = holding_code_dict[code]['volume']
            #
            last_au_factor = dailies.loc[last_date]['au_factor']

            after_volume = int(before_volume * (current_au_factor /
last_au_factor))
            holding_code_dict[code]['volume'] = after_volume
            print('持仓量调整：%s, %6d, %10.6f, %6d, %10.6f' %
                  (code, before_volume, last_au_factor, after_volume,
current_au_factor), flush=True)
        except:
            print('持仓量调整时，发生错误：%s, %s' % (code, _date), flush=True)
            traceback.print_exc()
```

卖出待卖股票

```
# 卖出
if len(to_be_sold_codes) > 0:
    for code in to_be_sold_codes:
        try:
            if code in before_sell_holding_codes:
                holding_stock = holding_code_dict[code]
                holding_volume = holding_stock['volume']
                sell_price = code_daily_dict[code].loc[_date]['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)
        except:
            print('卖出时，发生异常：%s, %s' % (code, _date), flush=True)
            traceback.print_exc()

print('卖出后，现金: %10.2f' % cash)
```

```
# 买入
if len(to_be_bought_codes) > 0:
    for code in to_be_bought_codes:
        try:
            if cash > single_position:
                buy_price = code_daily_dict[code].loc[_date]['open']
                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), flush=True)
        except:
            print('买入时，发生错误：%s, %s' % (code, _date), flush=True)
            traceback.print_exc()

print('买入后，现金: %10.2f' % cash)
```

买入待买股票

股票池再平衡

```
# 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 rebalance_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)
```

卖出和买入信号检测

```
# 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, begin_date=signal_begin_date,
end_date=_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, begin_date=signal_begin_date,
end_date=_date):
            to_be_bought_codes.add(_code)
```

统计收益

```
# 计算总资产
total_value = 0
for code in holding_codes:
    try:
        holding_stock = holding_code_dict[code]
        value = code_daily_dict[code].loc[_date][‘close’] * holding_stock[‘volume’]
        total_value += value

        # 计算总收益
        profit = (value - holding_stock[‘cost’]) * 100 / holding_stock[‘cost’]
        # 计算单日收益
        one_day_profit = (value - holding_stock[‘last_value’]) * 100 /
holding_stock[‘last_value’]
        # 暂存当日市值
        holding_stock[‘last_value’] = value

        print(‘持仓: %s, %10.2f, %4.2f, %4.2f’ % (code, value, profit, one_day_profit))

        # 保存每一日股票的持仓数
        code_date_volume_dict[code + '_' + _date] = holding_stock['volume']
    except:
        print('计算收益时发生错误: %s, %s' % (code, _date), flush=True)
        traceback.print_exc()
```

统计收益

```
total_capital = total_value + cash

hs300_k_current = dm.get_k_data('000300', index=True, start=_date, end=_date)
hs300_current_value = hs300_k_current.loc[hs300_k_current.index[0]]['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

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

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

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

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

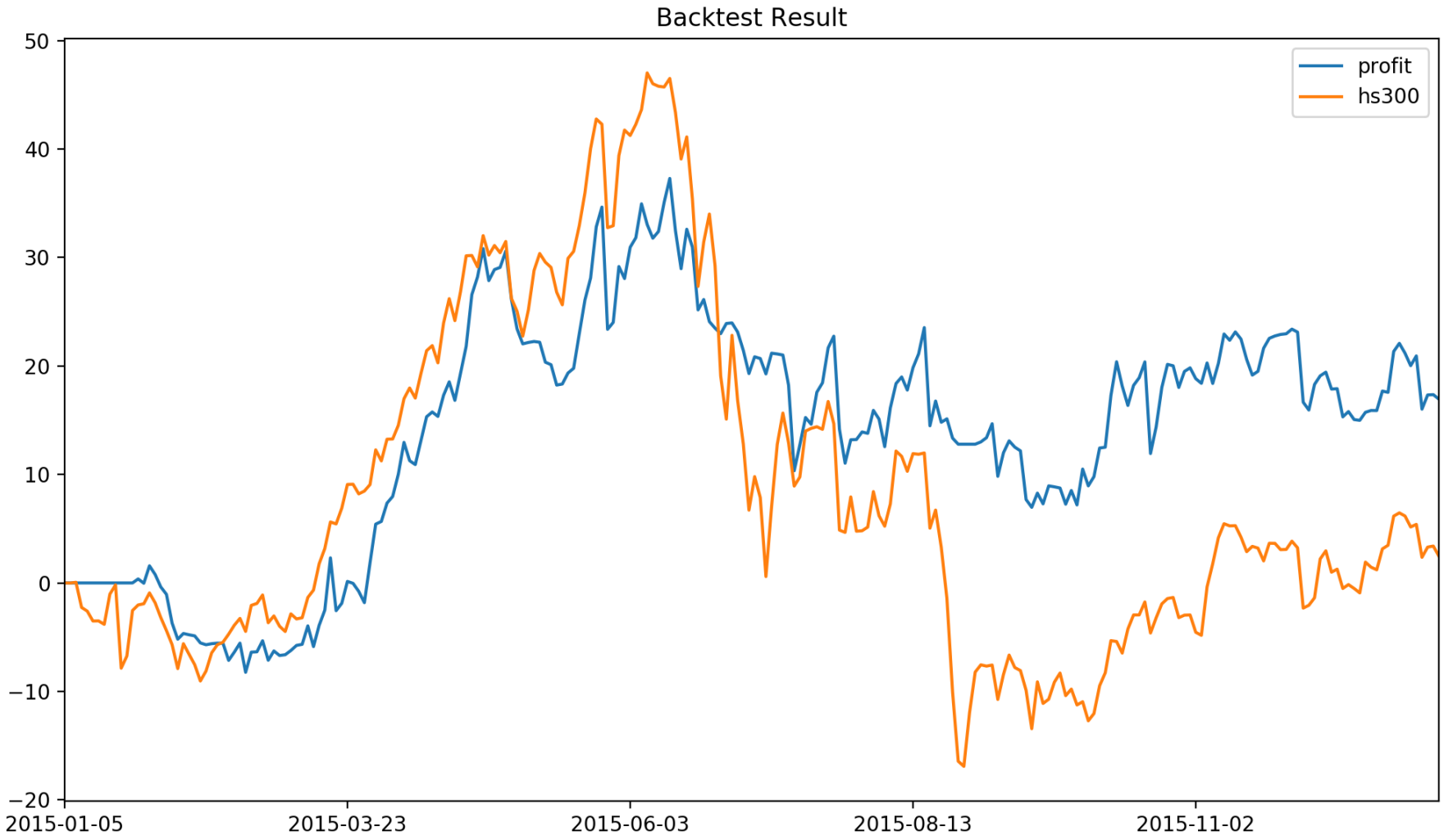
R_f - 无风险收益

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
```

```
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


Backtest Result
profit
hs300
50
40
30
20
10
0
−10
−20
2015-01-05
2015-03-23
2015-06-03
2015-08-13
2015-11-02

休息一下

5分钟后回来

化繁为简，分而治之

模块化交易系统框架

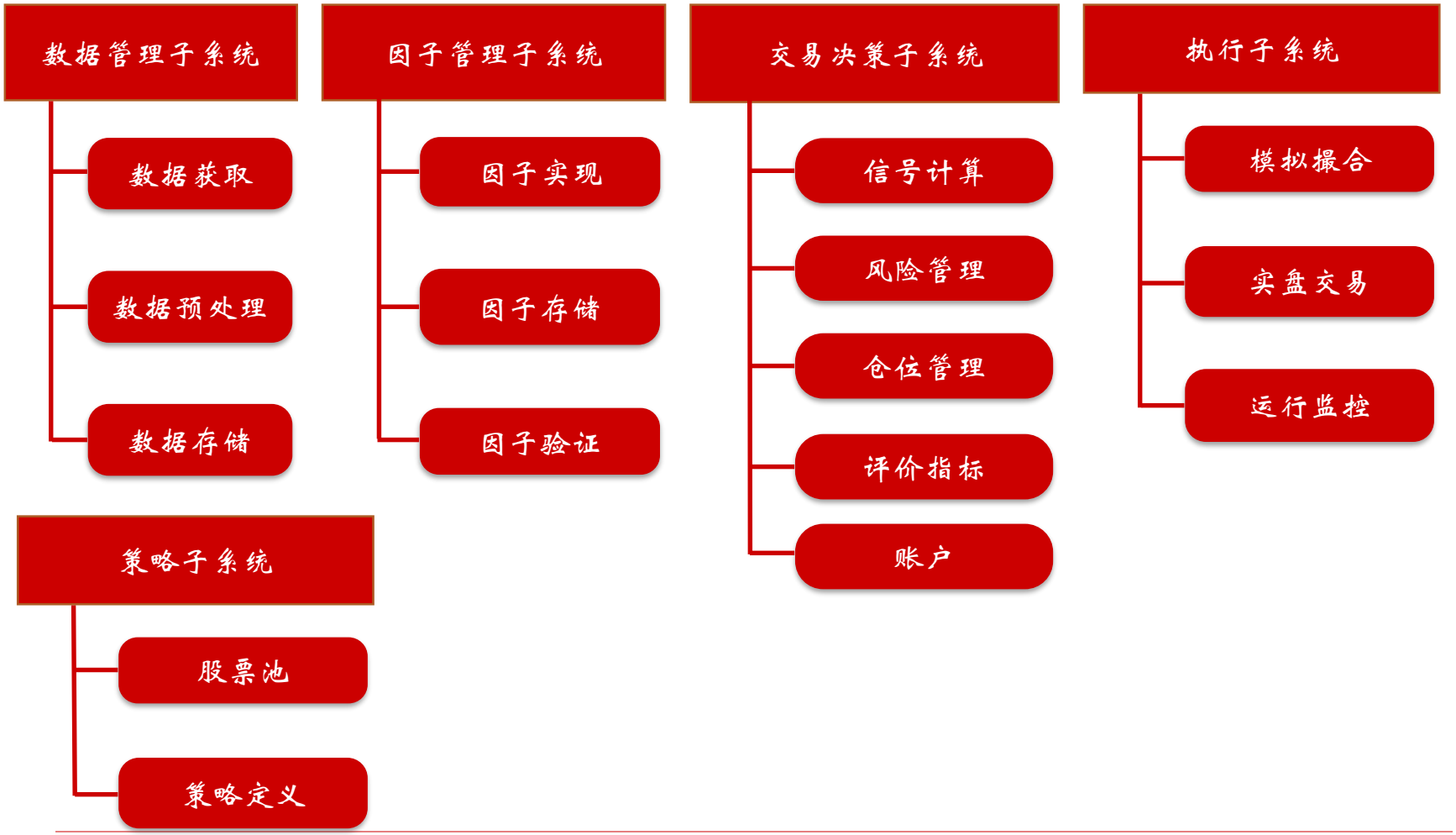
策略研发的实际工作情况

- 策略参数的频繁调整
  - 头寸分配
  - 回测周期
  - 止盈止损
  - 加仓/减仓
- 不同策略的深入分析
- 多策略的组合回测

模块化设计的优点

- 框架的逻辑结果更清晰
- 团队协作效率提高
  - 策略研发
  - 工程实现
- 复用度提高
  - 代码复用
  - 数据复用
- 可维护性提高

模块化


数据管理子系统
数据获取
数据预处理
数据存储
因子管理子系统
因子实现
因子存储
因子验证
交易决策子系统
信号计算
风险管理
仓位管理
评价指标
账户
执行子系统
模拟撮合
实盘交易
运行监控
策略子系统
股票池
策略定义

做好准备

开发环境准备

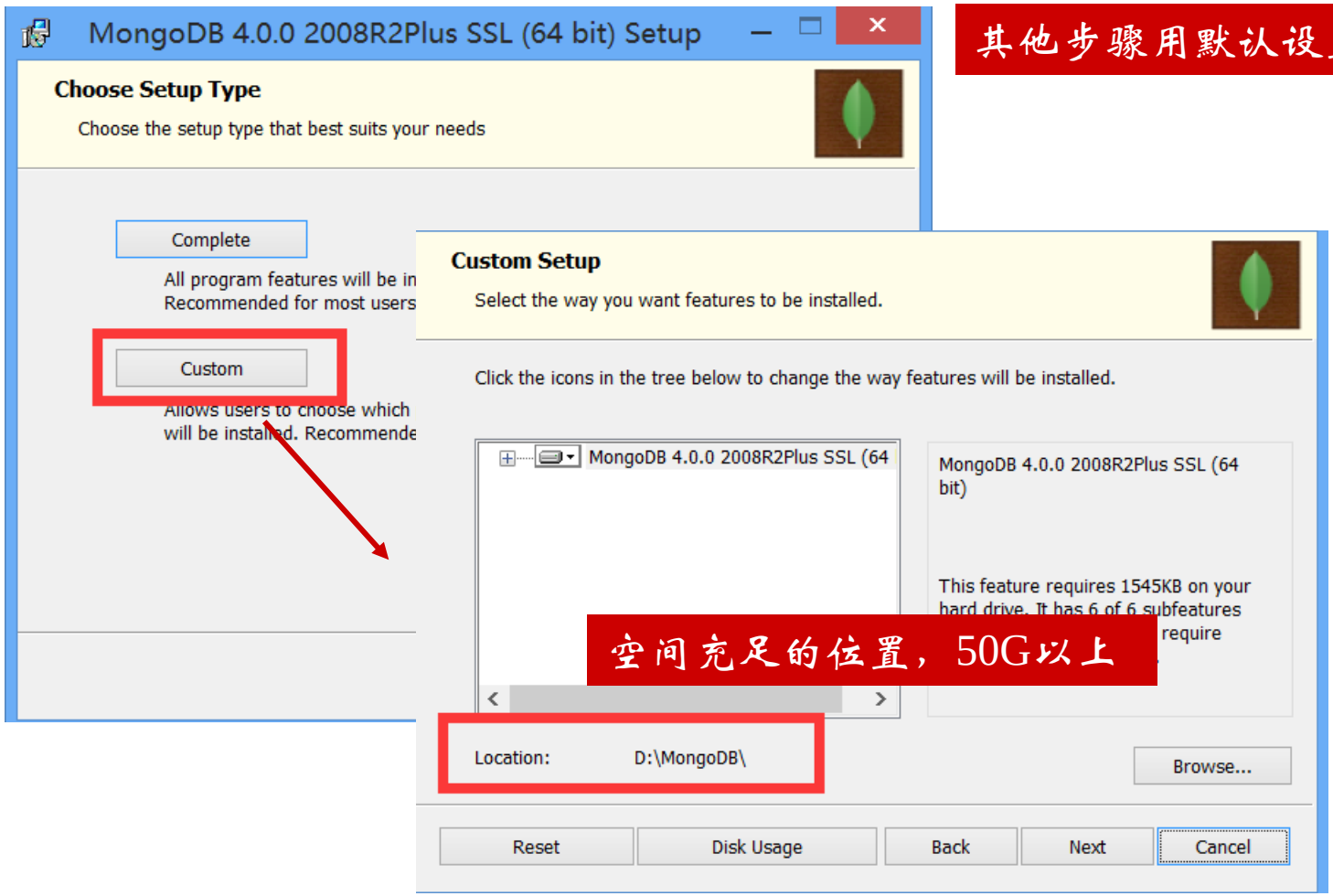
需要安装的软件

- 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

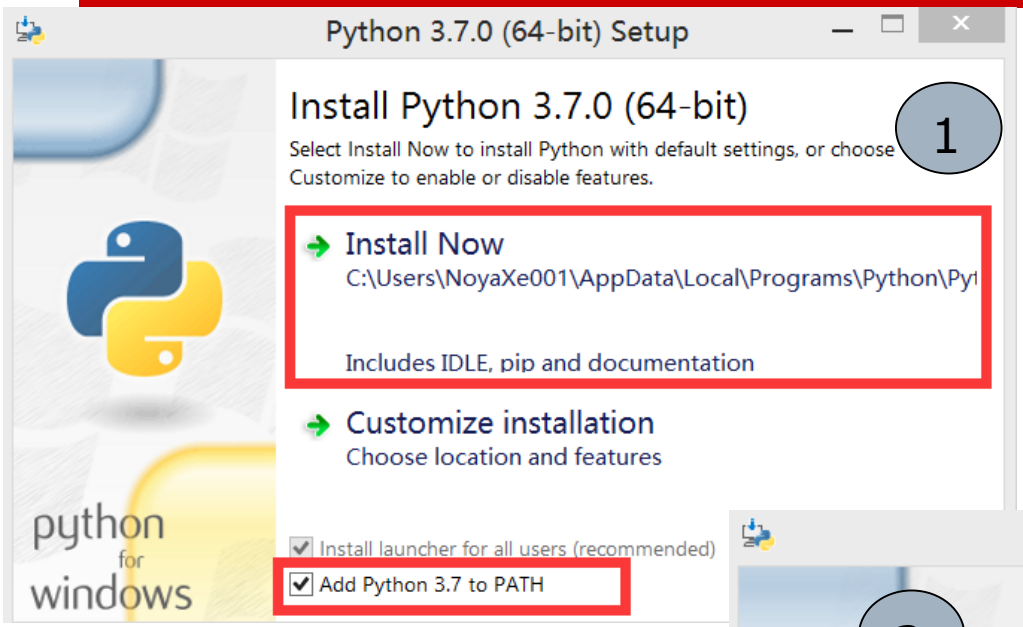


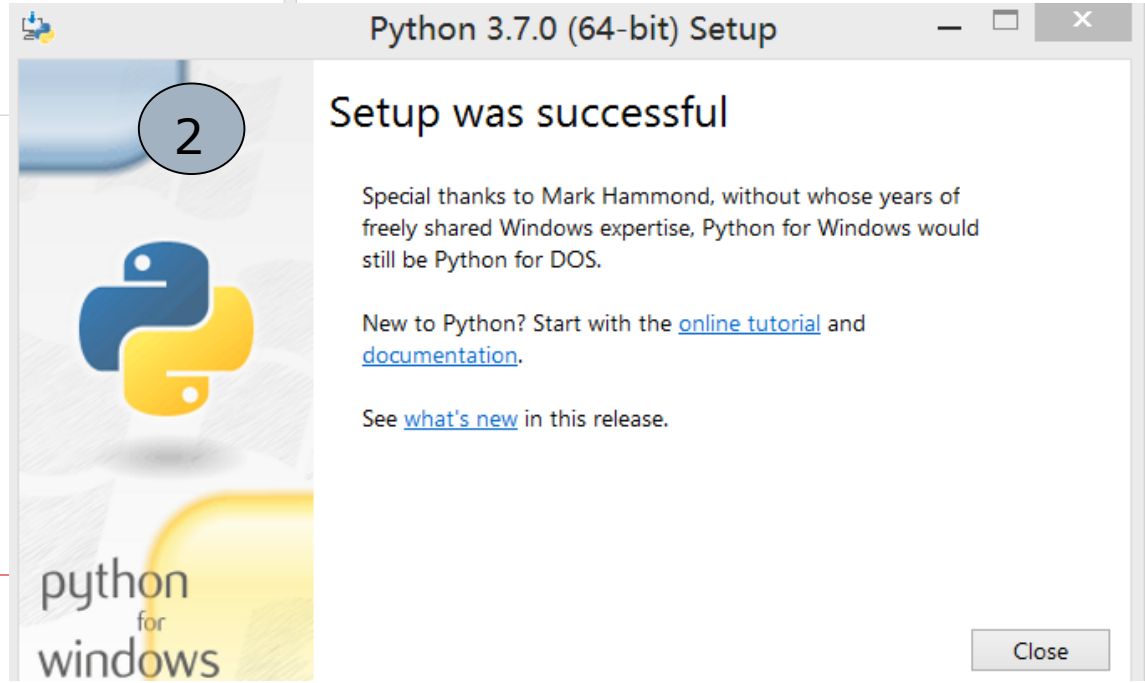
其他步骤用默认设置即可

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

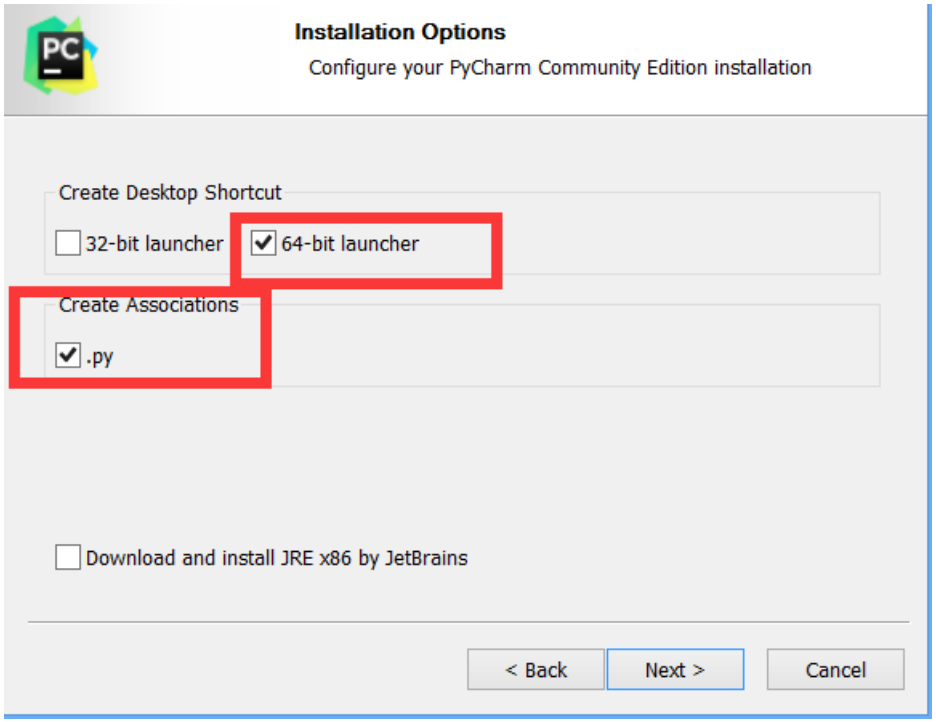
Python


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

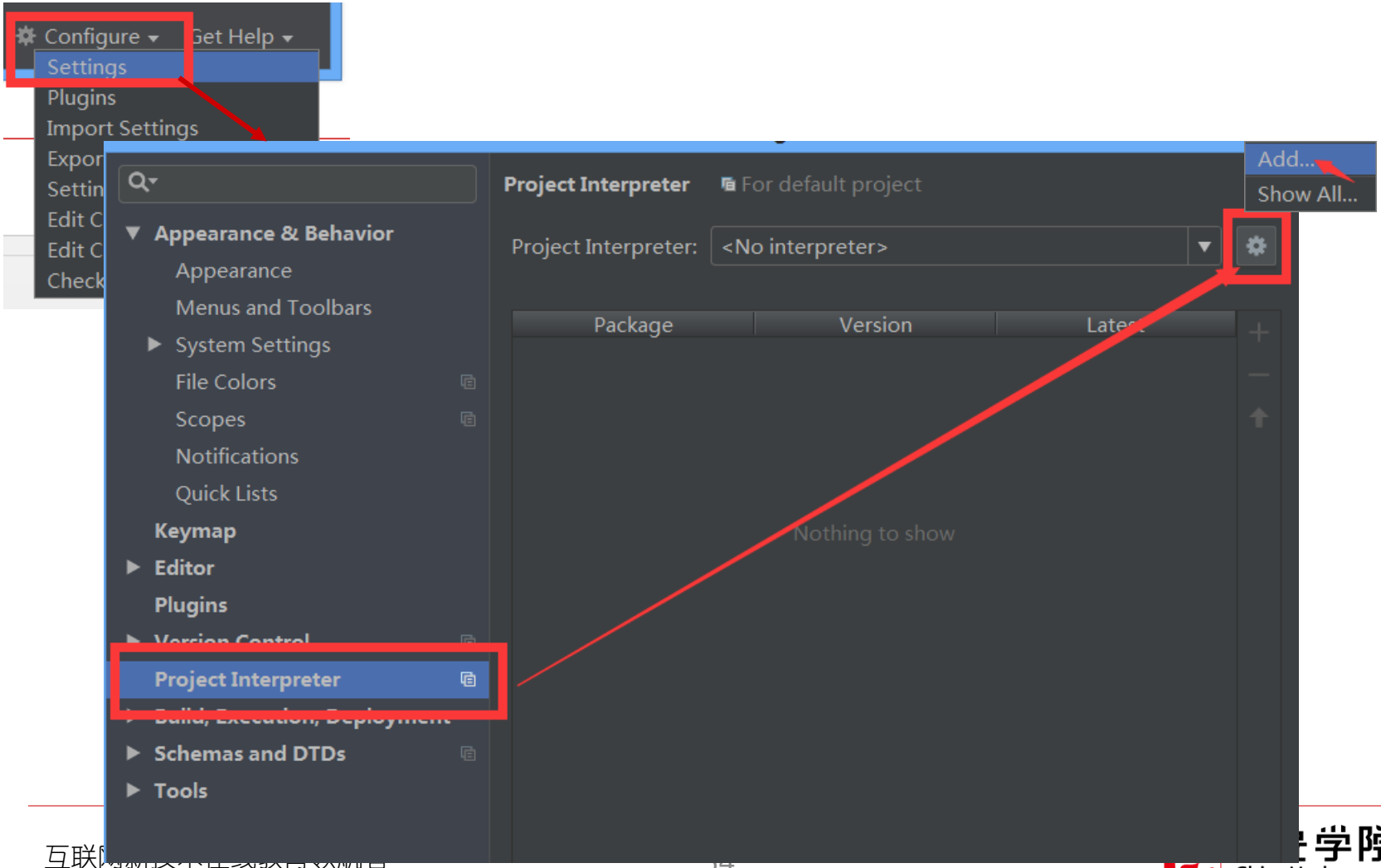

Python 3.7.0 (64-bit) Setup
2
Setup was successful
Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.
New to Python? Start with the online tutorial and documentation.
See what's new in this release.
python
for
windows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

PyCharm

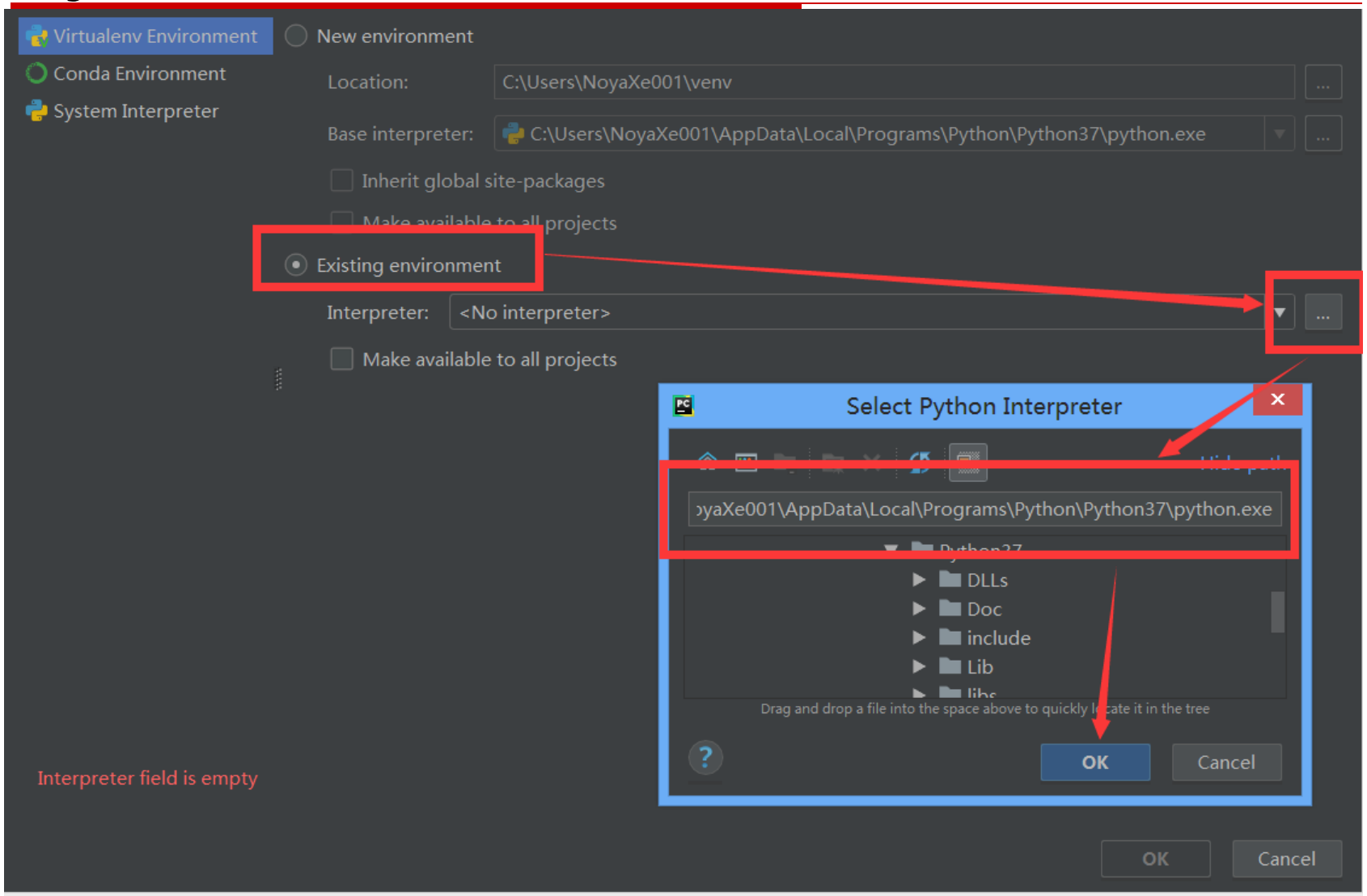
- 版本：Community
- 下载地址
  - http://www.jetbrains.com/pycharm/download/#section=windows



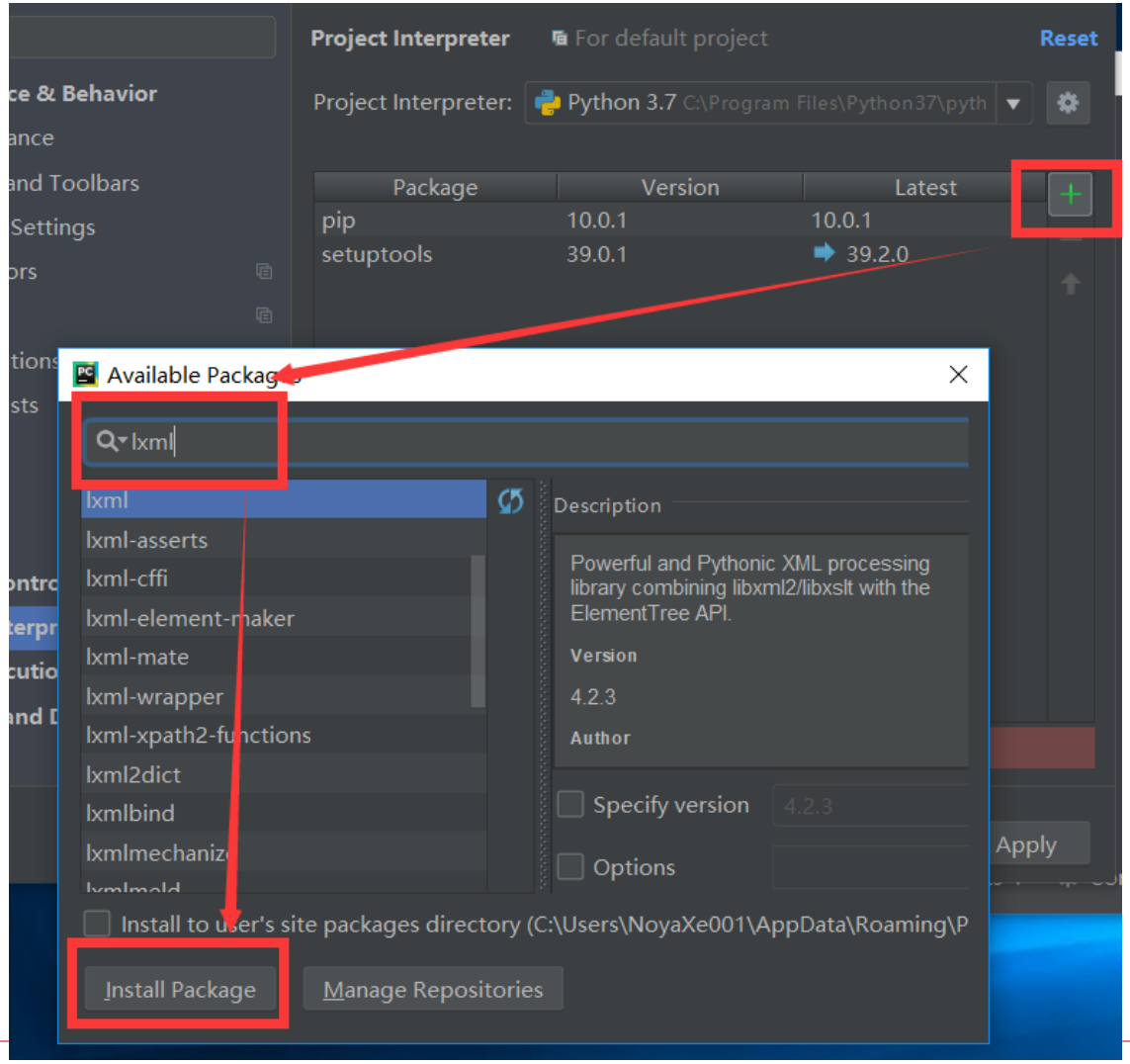
PyCharm设置


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

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



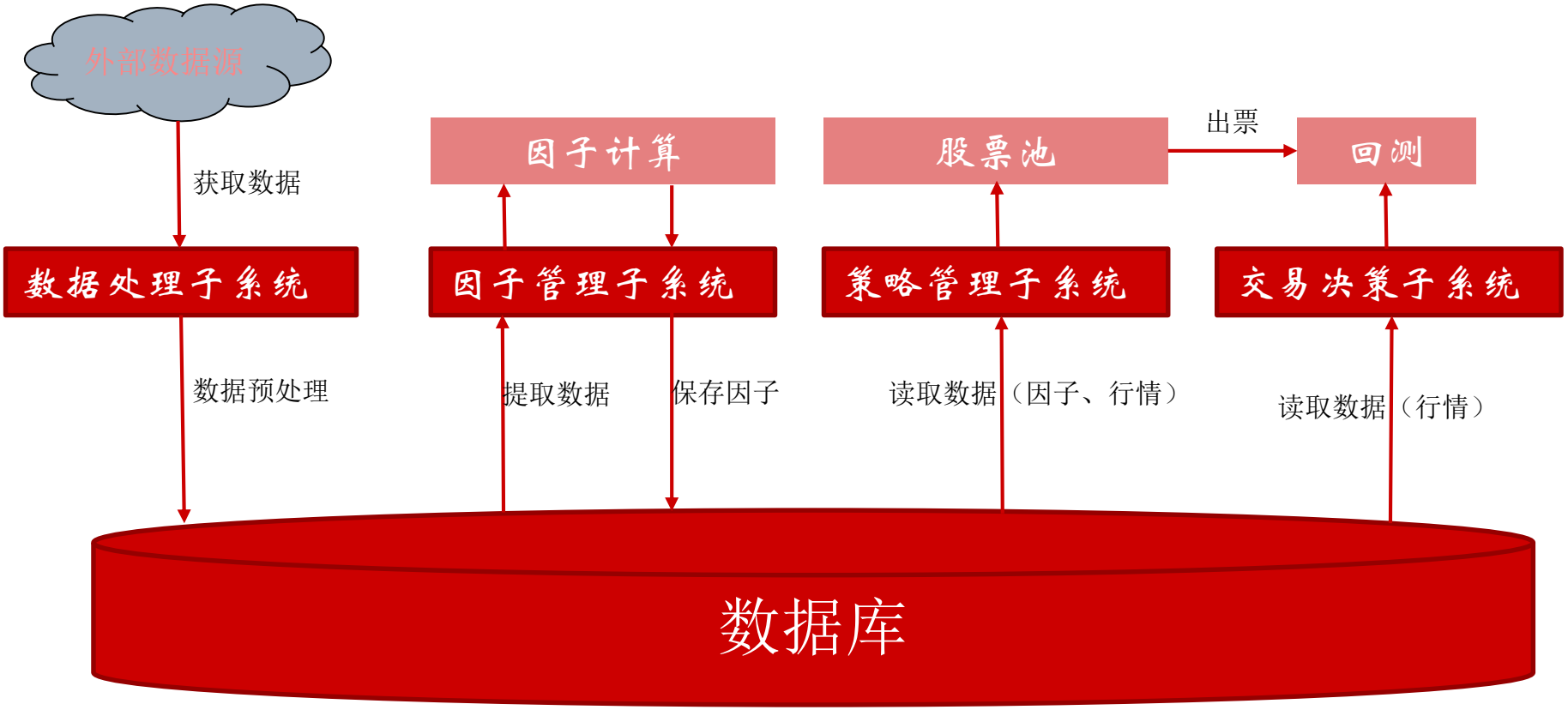
搭建框架

模块化交易系统框架接口实现

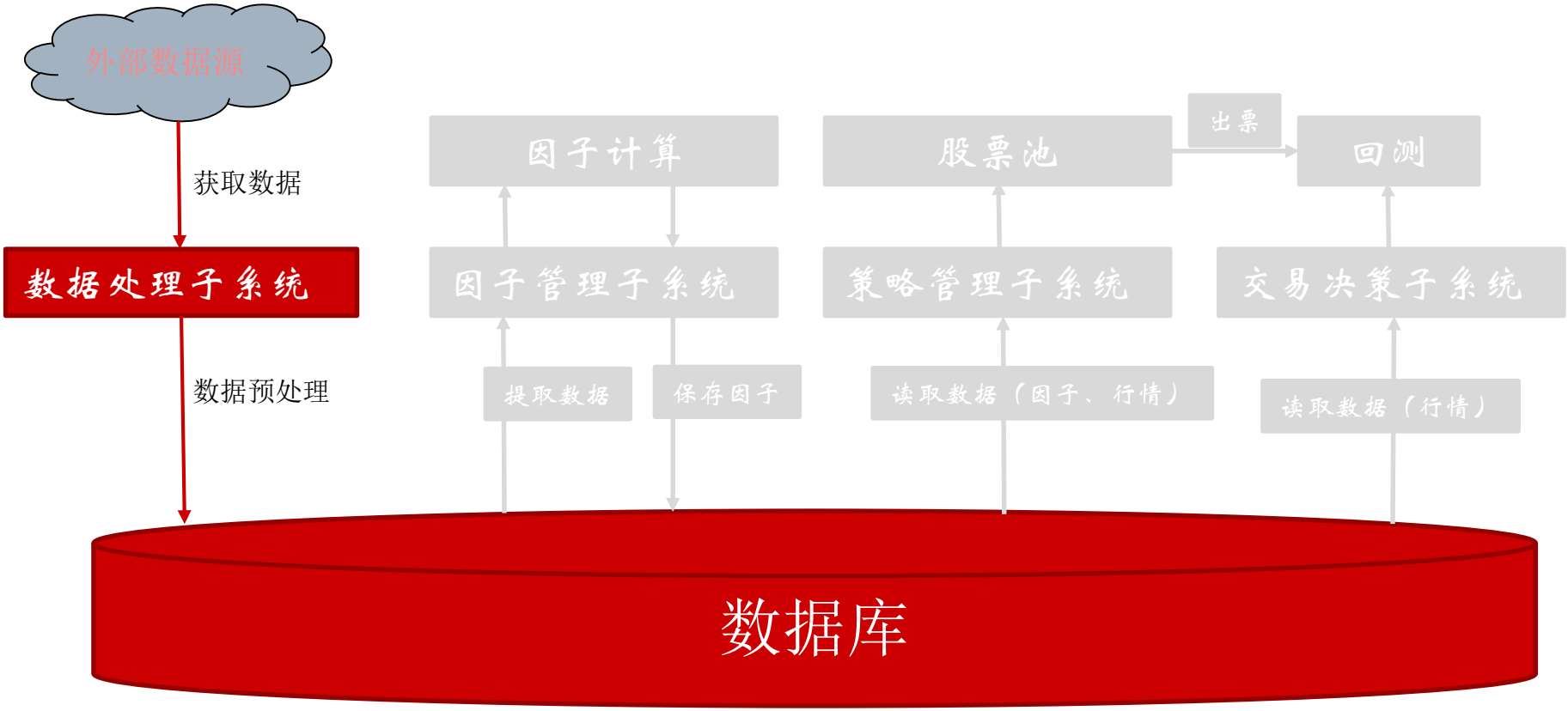
目录结构


回测入口文件
backtest.py
data
数据处理子系统
因子管理子系统
factor
strategy
策略管理子系统
交易决策子系统
trading
util
工具文件
执行子系统
对接第三方
模拟盘

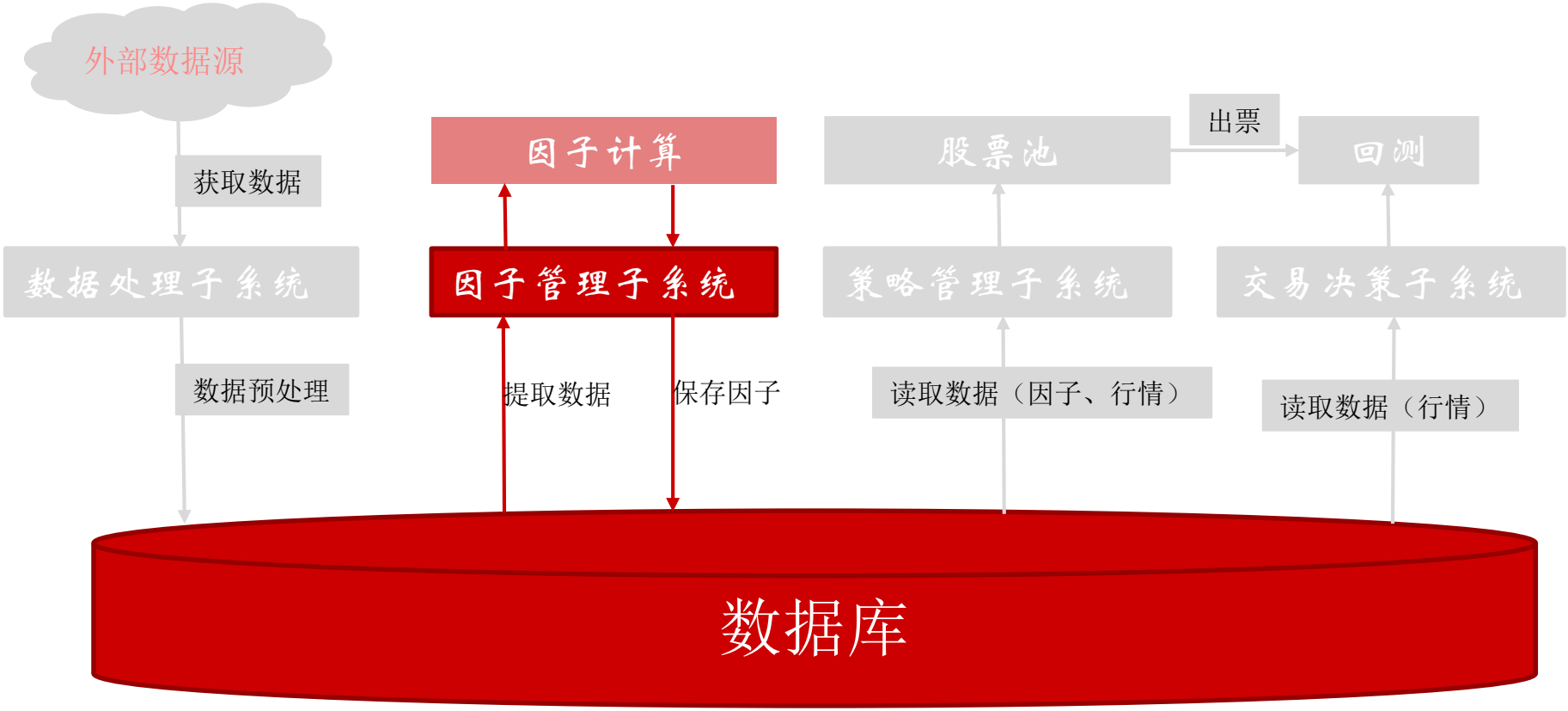
示例策略回测的数据处理流程



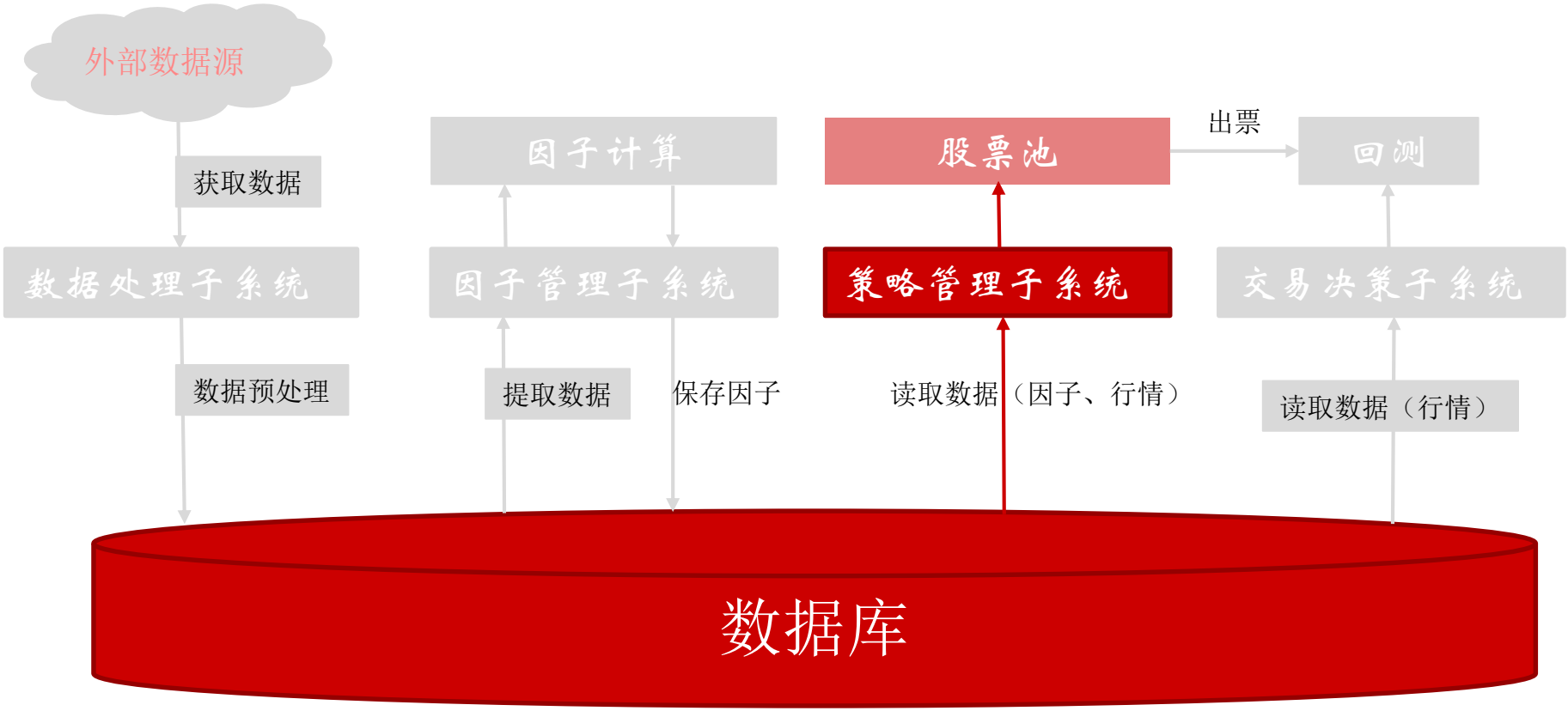
示例策略回测的数据处理流程



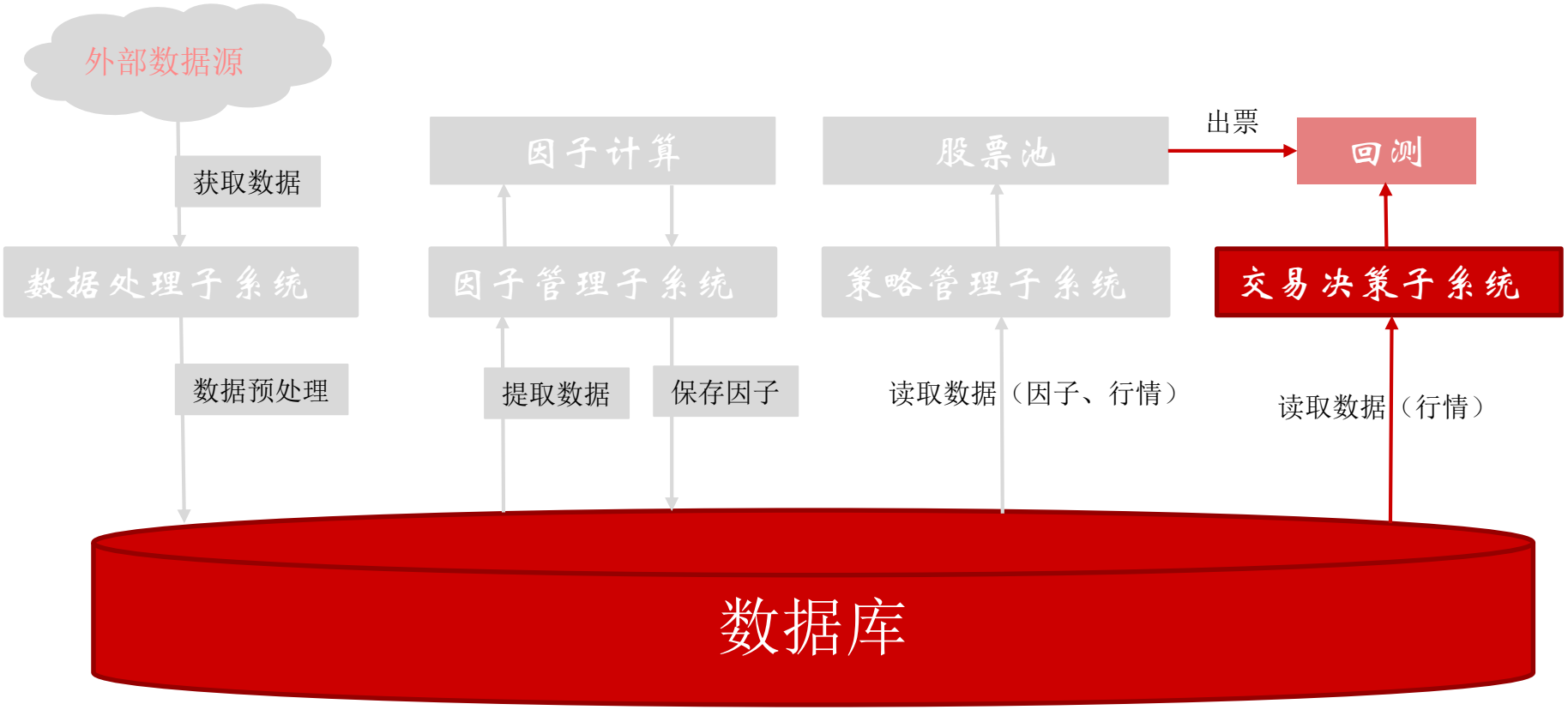
示例策略回测的数据处理流程



示例策略回测的数据处理流程



示例策略回测的数据处理流程



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

from pymongo import MongoClient
from abc import abstractmethod
from util.database import DB_CONN

"""
因子的基类。
"""


class BaseFactor:
    def __init__(self, name):
        self.name = name
        self.collection = DB_CONN[name]

    @abstractmethod
    def compute(self):
        pass
```

因子的基类

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

from pymongo import UpdateOne
from base_factor import BaseFactor
from data.finance_report_crawler import get_code_reports
import tushare as ts

class PEFactor(BaseFactor):
    def __init__(self):
        BaseFactor.__init__(self, name='pe')

    def compute(self):
        code_report_dict = self.get_code_reports()

        codes = set(code_report_dict.keys())
        for code in codes:
            dailies = ts.get_k_data(code, autype=None, start='2015-01-01', end='2015-01-
10')
            # 如果没有合适的数据
            if dailies.index.size == 0:
                continue

            # 业绩报告列表
            reports = code_report_dict[code]

            dailies.set_index(['date'], inplace=True)

            update_requests = []
```

PE因子

```
for current_date in dailies.index:
    # 用来保存最后一个公告日期小于等于当前日期的财报
    last_report = None
    for report in reports:
        announced_date = report['announced_date']
        # 如果公告日期大于当前调整日，则结束循环
        if announced_date > current_date:
            break

        last_report = report

    # 如果找到了正确时间范围的年报， 则计算PE
    if last_report is not None:
        pe = dailies.loc[current_date]['close']/last_report['eps']

        print('%s, %s, %s, eps: %5.2f, pe: %6.2f' %
              (code, current_date, last_report['announced_date'], last_report['eps'], pe),
              flush=True)

        update_requests.append(
            UpdateOne(
                {'code': code, 'date': current_date},
                {'$set': {'code': code, 'date': current_date, 'pe': pe}}, upsert=True))

if len(update_requests) > 0:
    save_result = self.collection.bulk_write(update_requests, ordered=False)
    print('股票代码: %s, 因子: %s, 插入: %4d, 更新: %4d' %
          (code, self.name, save_result.upserted_count, save_result.modified_count),
flush=True)
```

PE因子

颜值就是战斗力

利用ECharts实现交互式净值曲线

实现的功能

- 解析日志文件
- 生成净值曲线
- 鼠标在曲线上移动时，显示日期以及收益和基准收益
- 点击净值曲线上的点，显示当日的买入和卖出记录

ECharts下载

http://echarts.baidu.com/download.html

使用流程

1. 完成策略回测，回测过程中的日志重定向到文件中，例如backtest_result
2. 双击打开visual文件夹中的profit.html
3. 点击页面上的加载“选择日志”后面的选择文件
4. 加载第1步中生成的backtest_result，收益曲线就会被绘制出来
5. 点击曲线上的点，可以在图下方看到“买入记录”和“卖出记录”

总结

- Lite版回测框架的实现
- 模块化交易系统的结构
- 模块化实现了回测流程
- ECharts实现交互式净值曲线

课后练习（1）

- 以本课中的PE因子为例，练习用Python进行MongoDB的基本操作
  - 插入记录
  - 删除记录
  - 更新记录
  - 条件查询（包括排序）
  - 索引管理

课后练习（2）

- 根据课件中关于回测中应该注意的问题的提示，你认为还应该加入哪些细节，可以让回测更接近实际情况？
- 实现这些细节时，还需要补充哪些数据？

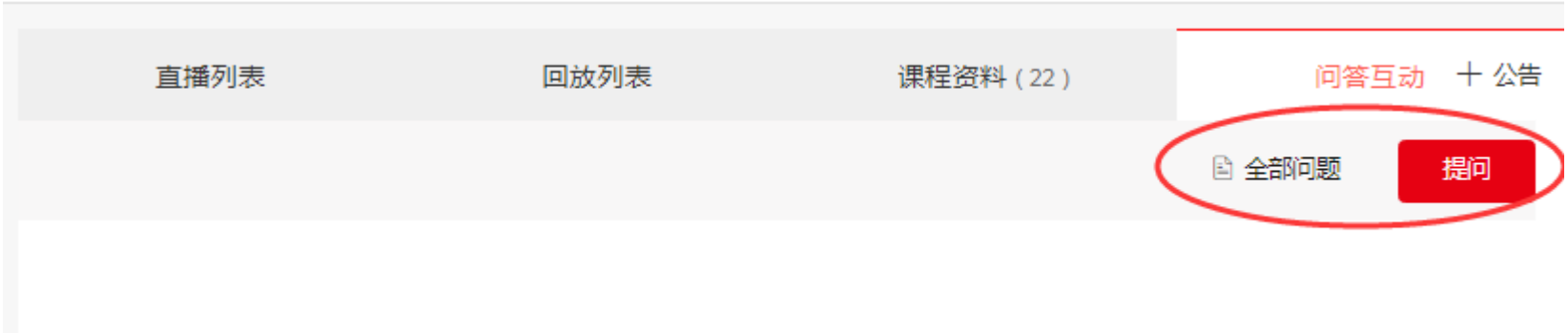
下节课预告

- 题目：数据管理和因子管理子系统的实现

  - 编程语言和运行平台：
    Python、MongoDB

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