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

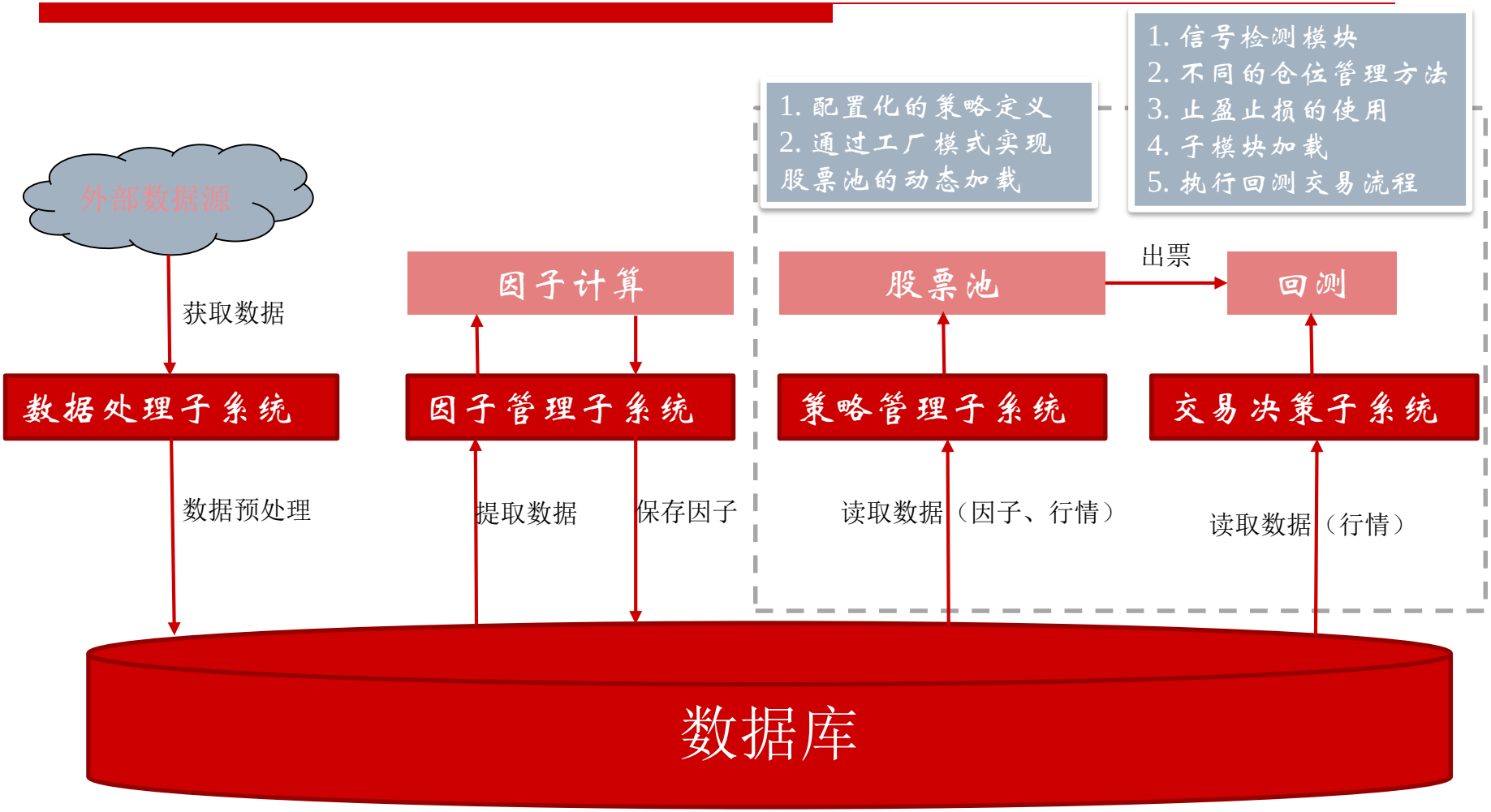
第六课

交易决策子系统的实现：信号计算、仓位管理、风险管理

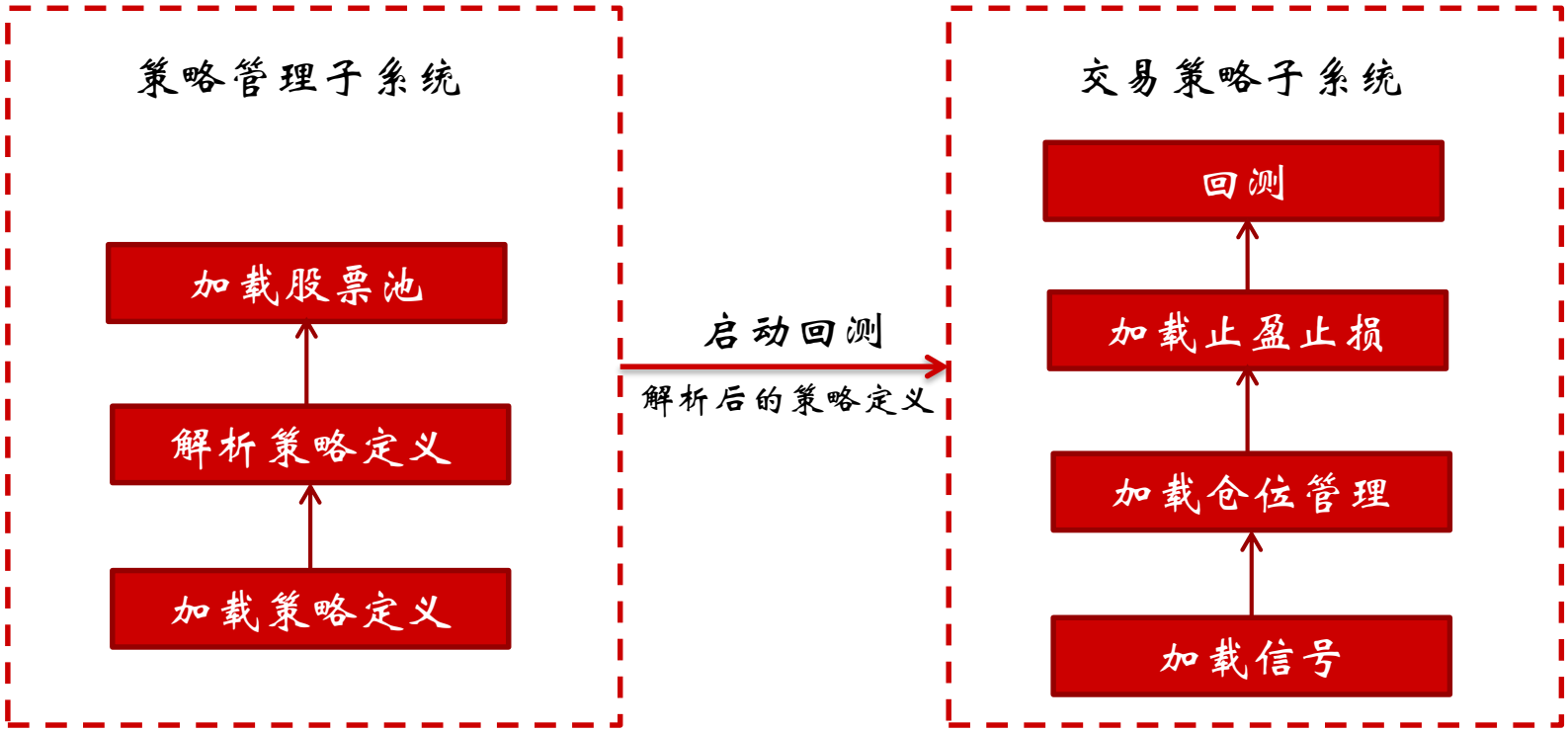
系统化构建量化交易体系：

模块2：**搭建自己的股票回测及交易平台**

模块化交易系统架构图


1. 信号检测模块
2. 不同的仓位管理方法
3. 止盈止损的使用
4. 子模块加载
5. 执行回测交易流程
1. 配置化的策略定义
2. 通过工厂模式实现股票池的动态加载
外部数据源
获取数据
因子计算
股票池
出票
回测
数据处理子系统
因子管理子系统
策略管理子系统
交易决策子系统
数据预处理
提取数据
保存因子
读取数据（因子、行情）
读取数据（行情）
数据库

工作流程


策略管理子系统
加载股票池
解析策略定义
加载策略定义
启动回测
解析后的策略定义
交易策略子系统
回测
加载止盈止损
加载仓位管理
加载信号

内容介绍

- 可配置的策略管理模块的实现
- 支持可配置策略的回测框架
- 信号检测模块的实现
- 仓位管理功能的实现
- 增加止损：触线止损和跟踪止损
- 增加止盈：触线止盈和回撤止盈

可配置的策略管理模块的实现

策略研发的实际场景

- 策略参数的频繁调整
  - 股票池
  - 头寸分配
  - 回测周期
  - 止盈止损
  - 加仓/减仓

策略的可定制参数

- 股票池
- 择时信号
  - 买入信号
  - 卖出信号
- 回测周期
  - 开始日期
  - 结束日期
- 仓位分配
- 止盈
- 止损

```
# 策略的定义文件，等号前面是属性，等号名称后面是属性值
# 策略名称
name=低PE策略
# 股票池名称
stock_pool=low_pe_stock_pool
# 股票的调整周期
interval=7
# 开始日期
begin_date=2015-01-01
# 结束日期
end_date=2015-12-31
# 总资金
capital=10000000
# 仓位分配
position=equal
# 单只股票资金分配
single_position=200000
# 资金上限
position_up_limit=200000
# 卖出信号
sell_signal=daily_k_close_down_break_ma10
# 买入信号
buy_signal=daily_k_close_up_break_ma10
```

策略的配置

策略配置文件的解析

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

from trading.backtest import Backtest
from strategy.strategy_option import StrategyOption
import sys, traceback,os


class Strategy:
    def __init__(self, name):
        # 策略的属性定义
        properties = dict()

        strategy_file = os.path.join(sys.path[0], 'strategies', name)

        if os.path.exists(strategy_file) is False:
            print("策略名文件名称有误：%s，请确认后重新输入。" % name, flush=True)
            return

        with open(strategy_file) as contents:
            for line in contents:
                if line.startswith("#") is False:
                    if line.index('=') > 0:
                        line = line.replace('\n', '')
                        configs = line.split('=')
                        properties[configs[0]] = configs[1]

        self.strategy_option = StrategyOption(properties)

    def backtest(self):
        backtest = Backtest(self.strategy_option)
        backtest.start()
```

策略回测入口

```
if __name__ == '__main__':
    if len(sys.argv) == 3:
        if sys.argv[1] == '--name':
            strategy = Strategy(sys.argv[2])
            strategy.backtest()
    else:
        print("启动回测的方式：python strategy_module.py --name strategy_name")
        print("例如：python strategy_module.py --name low_pe_strategy")
```

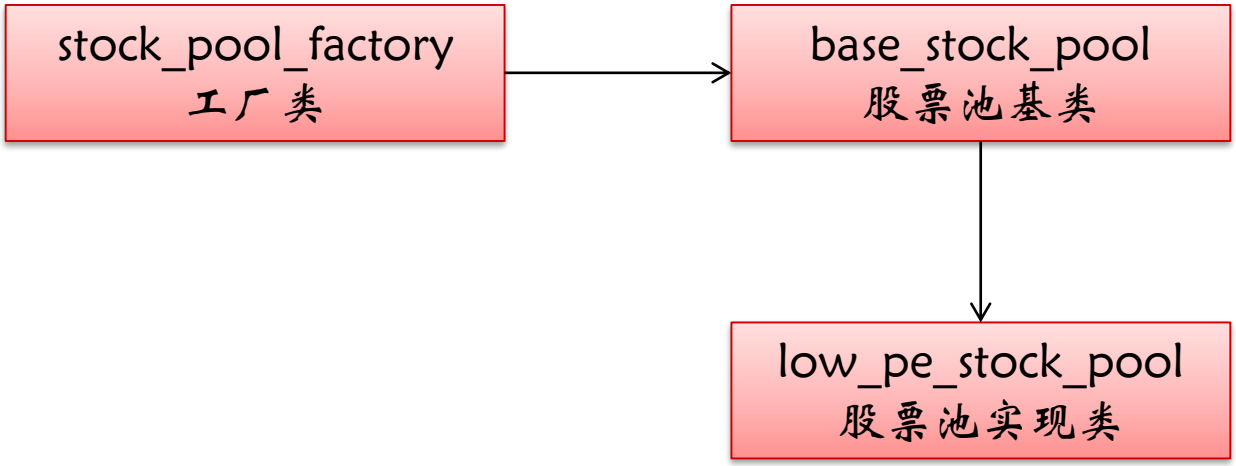
策略参数文件

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

from strategy.stock_pool.stock_pool_factory import StockPoolFactory
from datetime import datetime
from trading.signal.signal_factory import SignalFactory

“””
策略参数的定义
"""

class StrategyOption:
    def __init__(self, properties):
        self.properties = properties

    def capital(self):
        """
        获取策略的总资金配置
        :return: 如果没有设置，默认为1千万
        """
        if 'capital' in self.properties:
            return int(self.properties['capital'])
        else:
            return 1E7

    def stock_pool(self):
        if 'stock_pool' in self.properties:
            interval = int(self.properties['interval'])
            return StockPoolFactory.get_stock_pool(
                self.properties['stock_pool'],
                self.begin_date(),
                self.end_date(),
                interval)
        return None
```

```
def begin_date(self):
    """
    获取策略回测的开始日期
    :return: 默认为2015-01-01
    """
    if 'begin_date' in self.properties:
        return self.properties['begin_date']
    else:
        return '2015-01-01'

def end_date(self):
    """
    获取策略回测的结束日期
    :return: 默认为当前日期
    """
    if 'end_date' in self.properties:
        return self.properties['end_date']
    else:
        return datetime.now().strftime('%Y-%m-%d')

def single_position(self):
    return int(self.properties['single_position'])

def sell_signal(self, account):
    if 'sell_signal' in self.properties:
        return SignalFactory.get_signal(self.properties['sell_signal'], account)

def buy_signal(self, account):
    if 'buy_signal' in self.properties:
        return SignalFactory.get_signal(self.properties['buy_signal'], account)
```

策略参数文件

股票池的动态加载


stock_pool_factory
工厂类
base_stock_pool
股票池基类
low_pe_stock_pool
股票池实现类

strategy/stock_pool/stock_pool_factory.py

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

from strategy.stock_pool.low_pe_stock_pool import LowPeStockPool


class StockPoolFactory:
    @staticmethod
    def get_stock_pool(name, begin_date, end_date, interval):
        if name == 'low_pe_stock_pool':
            return LowPeStockPool(begin_date, end_date, interval)
```

股票池工厂类

strategy/strategy_option.py

```
 def stock_pool(self):
        if 'stock_pool' in self.properties:
            interval = int(self.properties['interval'])
            return StockPoolFactory.get_stock_pool(
                self.properties['stock_pool'],
                self.begin_date(),
                self.end_date(),
                interval)
        return None
```

股票池初始化

支持可配置策略的回测框架

实现功能

- 根据策略定义，加载交易决策子模块，完成回测交易流程
  - 信号
  - 仓位管理
  - 止盈止损
- 增加账户功能
  - 维护持仓股
  - 处理除权除息
  - 统计总市值

```
class Backtest:
    def __init__(self, strategy_option, begin_date=None, end_date=None):
        self.strategy_option = strategy_option
        if begin_date is None:
            self.begin_date = self.strategy_option.begin_date()
        else:
            self.begin_date = begin_date

        if end_date is None:
            self.end_date = self.strategy_option.end_date()
        else:
            self.end_date = end_date

        self.dm = DataModule()
        self.code_daily_cache = dict()

    def start(self):
        """
        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
        """
        # 初始总资金
        initial_capital = self.strategy_option.capital()
        # 初始现金
        cash = initial_capital
        # 单只股票仓位上限
        single_position = self.strategy_option.single_position()

        # 从获取策略配置中获取股票池
        stock_pool = self.strategy_option.stock_pool()

        # 保存持仓股的日期
        account = Account()

        # 获取卖出信号
        sell_signal = self.strategy_option.sell_signal(account)

        # 获取买入信号
        buy_signal = self.strategy_option.buy_signal(account)
```

回测的初始化

账户信息－初始化、持仓股管理

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

from data.data_module import DataModule
import traceback

"""
账户类，主要负责持仓股
"""


class Account:
    def __init__(self):
        self.holding = dict()
        self.holding_codes = set()
        self.dm = DataModule()

    def buy_in(self, code, volume, cost):
        self.holding[code] = {
            'volume': volume,
            'cost': cost,
            'last_value': cost}
        self.holding_codes.add(code)

    def sell_out(self, code):
        del self.holding[code]
        self.holding_codes.remove(code)

    def get_holding(self, code):
        """
        通过股票代码获取该股票的持仓情况
        :param code: 股票代码
        :return: 持仓对象，如果没有该股票的持仓，则返回None
        """
        if code in self.holding_codes:
            return self.holding[code]
        else:
            return None
```

账户信息－根据复权因子变化调整持仓量

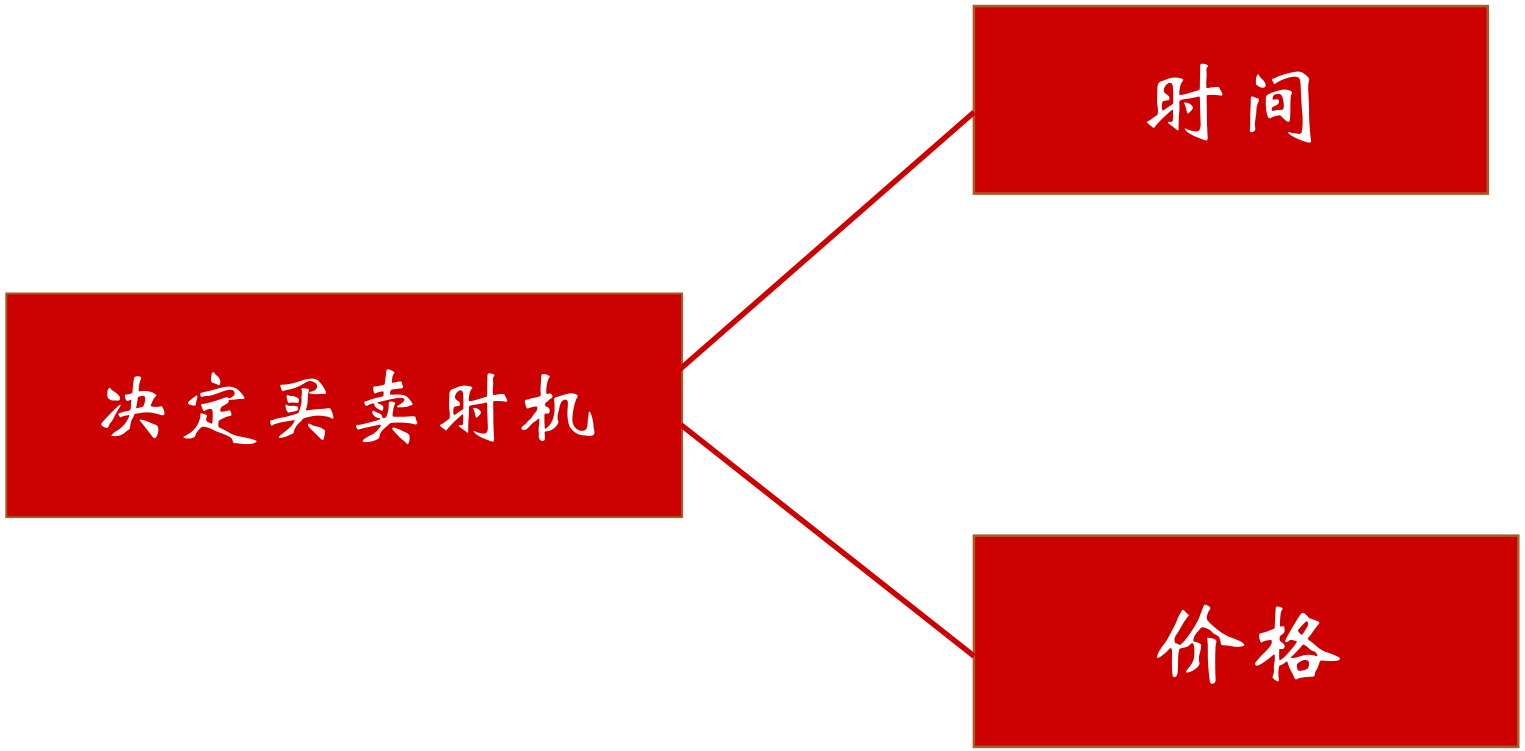
```
def adjust_holding_volume_at_open(self, last_date=None, current_date=None):
    """
    开盘时，处理持仓股的复权
    :param last_date: 上一个交易日
    :param current_date: 当前交易日
    """

    if last_date is not None and len(self.holding_codes) > 0:
        for code in self.holding_codes:
            try:
                dailies = self.dm.get_k_data(code=code, begin_date=last_date, end_date=current_date)
                if dailies.index.size == 2:
                    dailies.set_index(['date'], 1, inplace=True)
                    current_au_factor = dailies.loc[current_date]['au_factor']
                    before_volume = self.holding[code]['volume']
                    last_au_factor = dailies.loc[last_date]['au_factor']

                    after_volume = int(before_volume * (current_au_factor / last_au_factor))
                    self.holding[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, current_date), flush=True)
                traceback.print_exc()
```

账户信息－计算总市值

```
def get_total_value(self, date):
    """
    计算当期那持仓股在某一天的总市值，并更新持仓股的上一个市值
    :param date: 日期
    """
    total_value = 0
    dailies = self.dm.get_stocks_one_day_k_data(list(self.holding_codes), date=date)
    if dailies.index.size > 0:
        dailies.set_index(['code'], 1, inplace=True)
        for code in self.holding_codes:
            try:
                holding_stock = self.holding[code]
                value = dailies.loc[code]['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
                last_value = holding_stock['last_value']
                one_day_profit = (value - last_value) * 100 / last_value
                # 暂存当日市值
                self.holding[code]['last_value'] = value

                print('持仓: %s, %10.2f, %10.2f, %4.2f, %4.2f' %
                      (code, value, last_value, profit, one_day_profit))
            except:
                print('计算收益时发生错误: %s, %s' % (code, date), flush=True)

    return total_value
```

信号检测模块的实现

信号的作用



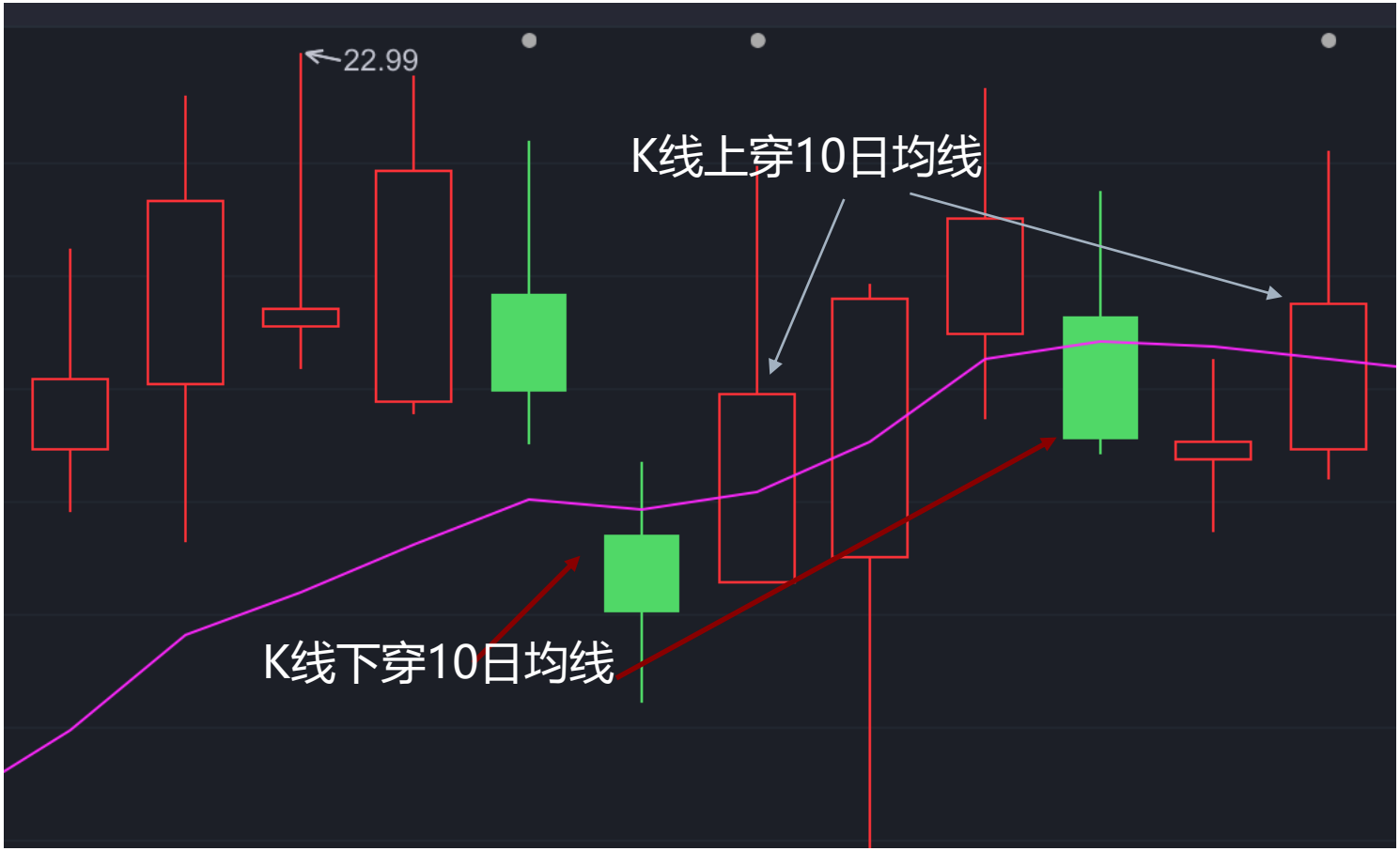
常见的买入信号

- 金叉：MACD金叉、KD金叉
- 上穿均线：K线上穿5日均线、10日均线
- 单针探底：当日K线呈现单针探底形状
- 定时信号：开盘后5分钟、收盘前5分钟
- 资金：主力资金加速流入

常见的卖出信号

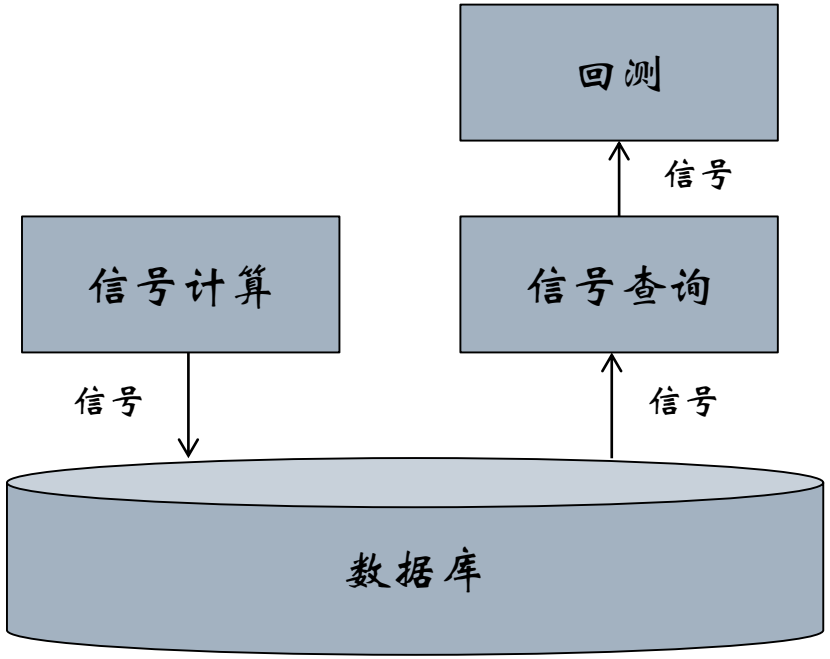
- 死叉：MACD死叉、KD死叉
- 下穿均线：K线下穿5日均线、10日均线
- 上涨SOT：当日K线呈现长上影线形状
- 定时信号：开盘后5分钟、收盘前5分钟
- 资金：主力资金加速流出

K线上穿/下穿10日均线



功能实现

- 信号计算
  - 实现信号算法
  - 检测历史信号
  - 保存到数据库
- 信号使用
  - 提供查询接口



信号计算的基类

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

from util.database import DB_CONN
from abc import abstractmethod

"""
信号计算的基类
"""


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

    @abstractmethod
    def compute(self, begin_date, end_date):
        """
        计算指定周期内的信号，并存储到数据库中
        :param begin_date: 开始日期
        :param end_date: 结束日期
        """
        pass
```

当日K线穿过10日均线的信号计算

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

from .base_signal_computer import BaseSignalComputer
from util.stock_util import get_all_codes
from data.data_module import DataModule
from pymongo import UpdateOne
import traceback

"""
当日K线上穿和下穿10日均线的判断
"""


class DailyKBreakMA10SignalComputer(BaseSignalComputer):
    def __init__(self):
        BaseSignalComputer.__init__(self, 'daily_k_break_ma10')

    def compute(self, begin_date=None, end_date=None):
        """
        计算指定日期内的信号
        :param begin_date: 开始日期
        :param end_date: 结束日期
        """
        codes = get_all_codes()

        dm = DataModule()
```

```
for code in codes:
    try:
        df_dailies = dm.get_k_data(code, autype='hfq', begin_date=begin_date, end_date=end_date)

        if df_dailies.index.size == 0:
            continue

        # 计算MA10
        df_dailies['ma'] = df_dailies['close'].rolling(10).mean()
        # 计算当日收盘和MA10的差值
        df_dailies['delta'] = df_dailies['close'] - df_dailies['ma']

        # 删除不再使用的ma和close列
        df_dailies.drop(['ma', 'close'], 1, inplace=True)

        # 判断突破类型
        index_size = df_dailies.index.size
        breaks = [0]
        for index in range(1, index_size):
            # 如果当前日期为停牌状态，则后面连续11日不参与计算
            if df_dailies.loc[df_dailies.index[index]]['is_trading'] is False:
                count = 10
                while count > 0:
                    index += 1
                    count -= 1
                    breaks.append(0)

                index += 1

            last = df_dailies.loc[df_dailies.index[index - 1]]['delta']
            current = df_dailies.loc[df_dailies.index[index]]['delta']

            # 向上突破设为1，向下突破设为-1，不是突破设为0
            break_direction = 1 if last <= 0 < current else -1 if last >= 0 > current else 0
            breaks.append(break_direction)

        # 设置突破信号
        df_dailies['break'] = breaks
```

当日K线穿过10日均线的信号计算

当日K线穿过10日均线的信号计算

```
    # 将日期作为索引
    df_dailies.set_index(['date'], 1, inplace=True)
    # 删除不再使用的trade_status和delta数据列
    df_dailies.drop(['is_trading', 'delta'], 1, inplace=True)
    # 只保留突破的日期
    df_dailies = df_dailies[df_dailies['break'] != 0]

    # 将信号保存到数据库
    update_requests = []
    for index in df_dailies.index:
        doc = {
            'code': code,
            'date': index,
            # 方向，向上突破 up，向下突破 down
            'direction': 'up' if df_dailies.loc[index]['break'] == 1 else 'down'
        }
        update_requests.append(
            UpdateOne(doc, {'$set': doc}, upsert=True))

    if len(update_requests) > 0:
        update_result = self.collection.bulk_write(update_requests, ordered=False)
        print('%s, upserted: %4d, modified: %4d' %
              (code, update_result.upserted_count, update_result.modified_count),
              flush=True)
except:
  traceback.print_exc()
```

信号的基类

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

from abc import abstractmethod

"""
信号的基类
"""

class BaseSignal:
    def __init__(self, account):
        self.account = account

    @abstractmethod
    def is_match(self, code, date):
        """
        验证股票在某个时刻是否符合某个信号
        :param code: 股票代码
        :param date: 日期
        :return:
        """
        pass
```

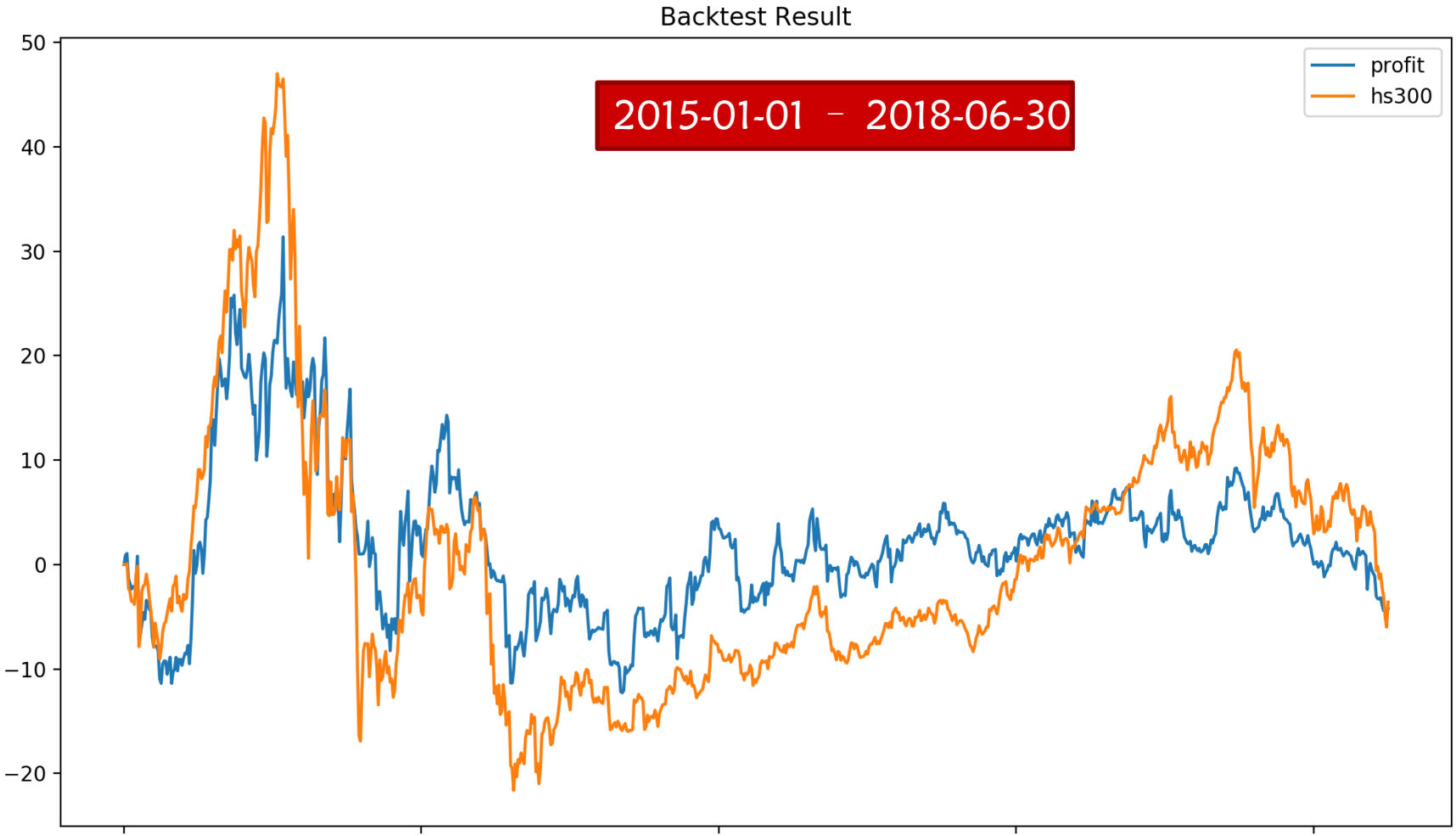
当日K线下穿10日均线

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

from data.data_module import DataModule
from .base_signal import BaseSignal
from util.database import DB_CONN

"""
当日K线下穿10日均线
"""


class DailyKCloseDownBreakMa10(BaseSignal):
    def __init__(self, account):
        BaseSignal.__init__(self, account)
        self.collection = DB_CONN['daily_k_break_ma10']

    def is_match(self, code, date):
        """
        股票是否在某日符合当日K线下穿10日均线的信号
        :param code: 股票代码
        :param date: 日期
        :return: True/False, True - 符合 False - 不符合
        """
        count = self.collection.count({'code': code, 'date': date, 'direction': 'down'})

        return count == 1
```

当日K线上穿10日均线

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

from data.data_module import DataModule
from .base_signal import BaseSignal
from util.database import DB_CONN

"""
当日K线上传10日均线
"""


class DailyKCloseUpBreakMa10(BaseSignal):
    def __init__(self, account):
        BaseSignal.__init__(self, account)
        self.collection = DB_CONN['daily_k_break_ma10']

    def is_match(self, code, date):
        """
        股票是否在某日符合当日K线上传10日均线的信号
        :param code: 股票代码
        :param date: 日期
        :return: True/False, True - 符合 False - 不符合
        """
        count = self.collection.count({'code': code, 'date': date, 'direction': 'up'})

        return count == 1
```

回测结果


Backtest Result
2015-01-01 - 2018-06-30
profit
hs300
50
40
30
20
10
0
−10
−20

休息一下

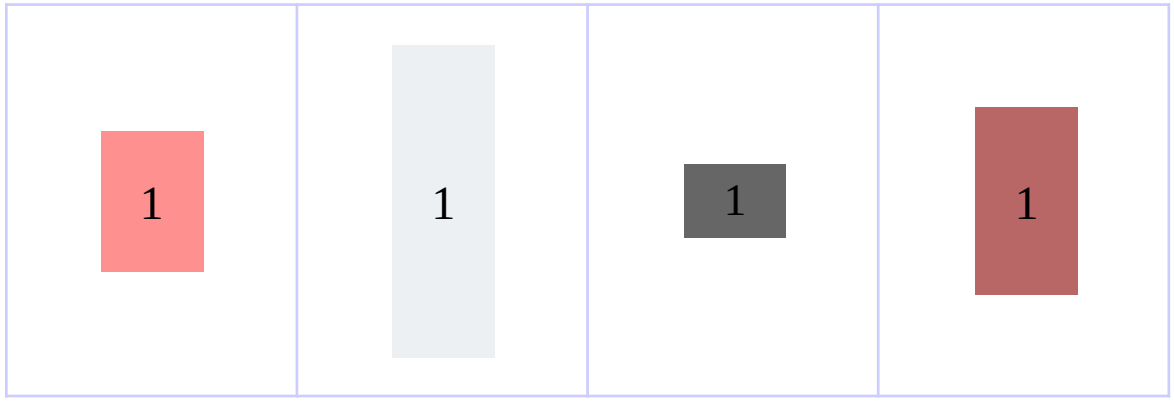
5分钟后回来

仓位管理的功能实现

仓位管理

每个交易单位采用固定金额

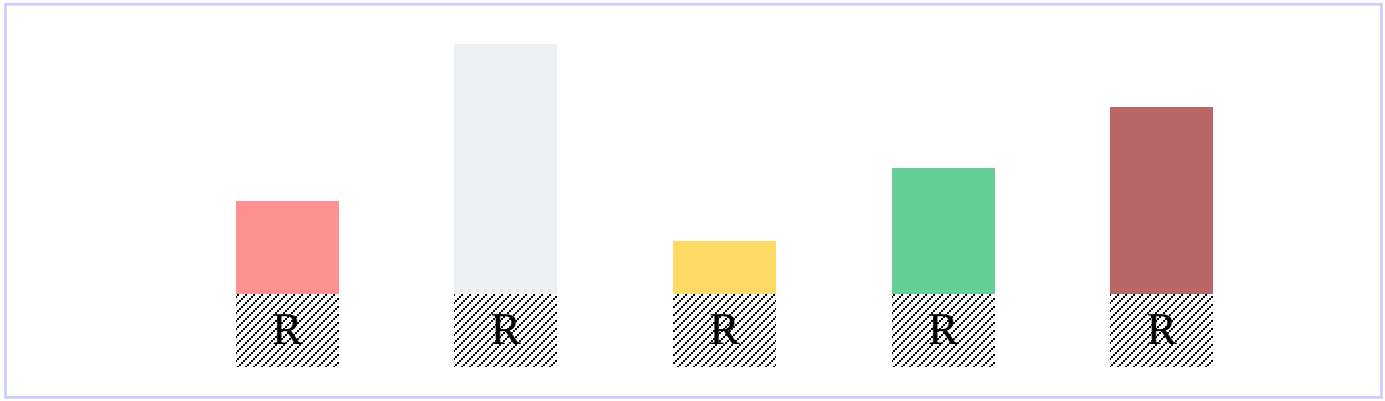
资金池:



把资金等分为相同金额的若干份，在出现买入信号的情况下，每份只允许交易一个单位的投资标的（比如1手股票，或是1份期货合约）

百分比风险

资金池:

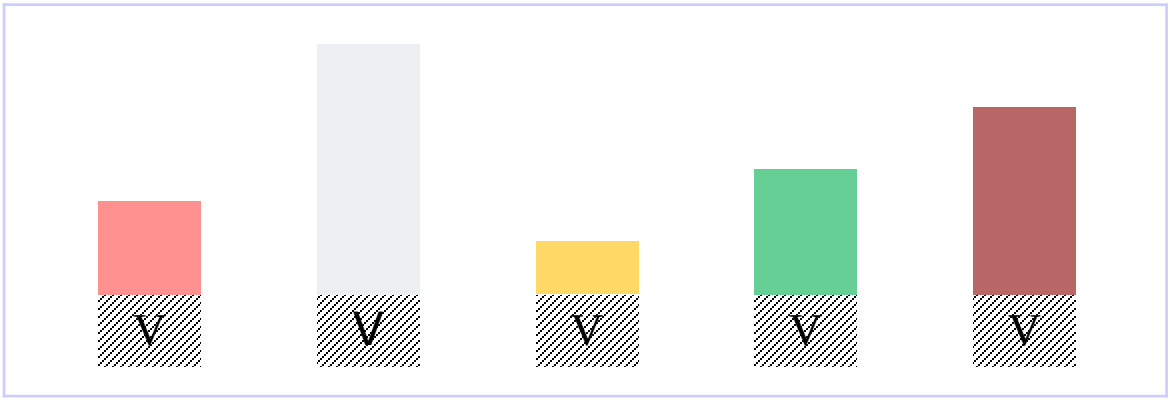


根据每次交易允许承担的最大风险占总资金的比例，以及每个投资标的可接受的最大损失（即初始止损额度R），折算出可建立头寸的单位个数

$$P(\text{头寸规模}) = C(\text{总风险})/R(\text{每股风险})$$

百分比波动幅度模型

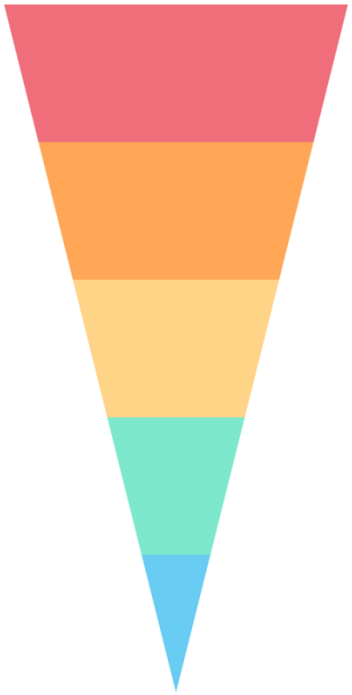
资金池:



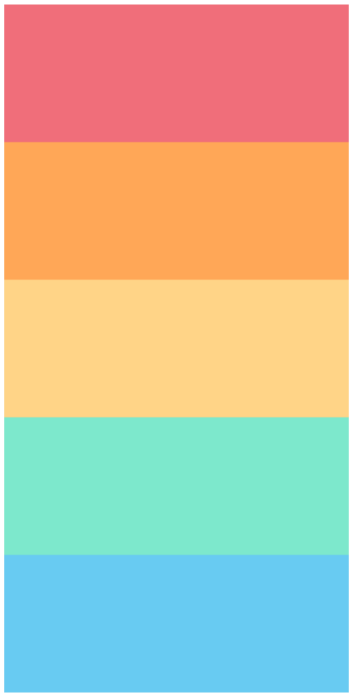
根据每次交易允许承担的最大风险占总资金的比例，以及每个投资标的在一段时间内的价格波动幅度（即可能有利或不利的价格变动范围V），折算出可建立头寸的单位个数

经典的加减仓方法

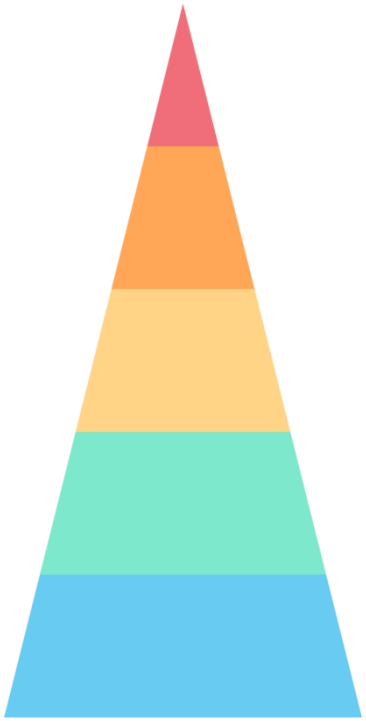
漏斗



矩形



金字塔



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

from .atr_position import ATRPosition


class PositionPolicyFactory:
    @staticmethod
    def get_add_position_policy(account, strategy_option):
        name = strategy_option.properties['add_position']
        if 'atr' == name:
            return ATRPosition(account, strategy_option)
```

ATR加仓 – 工厂类

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

  from abc import abstractmethod


  class BasePosition:
      def __init__(self, account, strategy_option):
          self.account = account
          self.strategy_option = strategy_option

      @abstractmethod
      def update_holding(self, code, date, update_holding):
          pass

      @abstractmethod
      def compute_position(self, code, date):
          pass

      @abstractmethod
      def get_add_signal(self, code, date):
          pass
```

ATR加仓 – 基类

ATR加仓 – 更新持仓股信息

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

from .base_position import BasePosition
from util.database import DB_CONN
from pymongo import DESCENDING
from pandas import DataFrame
from data.data_module import DataModule
from datetime import datetime, timedelta
import traceback


class ATRPosition(BasePosition):
    def __init__(self, account, strategy_option):
        BasePosition.__init__(self, account, strategy_option)
        self.dm = DataModule()

    def update_holding(self, code, date, updated_holding):
        """
        更新持仓股，为新的持仓股增加ATR、加仓次数
        :param code: 股票代码
        :param date: 日期
        :param updated_holding: 更新后的持仓股
        """
        try:
            existing_holding = self.account.get_holding(code)
            if existing_holding is not None and 'atr' not in existing_holding:
                atr = self.compute_atr(code, date)
                if atr is not None:
                    updated_holding['atr'] = atr
                    updated_holding['add_times'] = 0
                    self.account.update_holding(code, updated_holding)
        except:
            print('加仓策略，更新持仓股信息时，发生错误，股票代码：%s，日期：%s，' % (code, date), flush=True)
            traceback.print_exc()
```

ATR加仓 – 计算加仓信号

```
def get_add_signal(self, code, date):
    """
    如果符合加仓条件，则计算加仓信号
    :param code: 股票代码
    :param date: 日期
    :return: 加仓信号，如果没有则返回None，正常信号包括：code - 股票代码，position - 仓位
    """
    add_signal = None

    try:
        df_daily = self.dm.get_k_data(code, autype=None, begin_date=date, end_date=date)
        if df_daily.index.size > 0:
            holding_stock = self.account.get_holding(code)

            df_daily.set_index(['date'], 1, inplace=True)
            daily = df_daily.loc[date]
            if daily ['is_trading']:
                au_factor = daily ['au_factor']
                # ATR
                atr = holding_stock['atr'] / au_factor
                # 已加仓次数
                add_times = holding_stock['add_times']
                # 末次加仓价格，比较时用的实际价格
                last_buy_price = holding_stock['last_buy_hfq_price']
                hfq_close = daily['close'] * au_factor
                # 最多加仓4次 价格超过上一个加仓点+atr
                if add_times < 4 and hfq_close - last_buy_price > atr:
                    position = self.compute_position(code, date)
                    add_signal = {'code': code, 'position': position}
    except:
        print('计算加仓信号时，发生错误，股票代码：%s, 日期：%s' % (code, date), flush=True)

    return add_signal
```

```
def compute_atr(self, code, date):
    """
    计算ATR
    :param code: 股票代码
    :param date: 日期
    :return: ATR的值
    """
    minute_cursor = DB_CONN['minute'].find(
        {'code': code, 'time': {'$lte': date + ' 15:00'}},
        sort=[('time', DESCENDING)],
        projection={'date': True, 'time': True, 'close': True, 'high': True, 'low':
True, '_id': False},
        limit=60)
    minutes = [minute for minute in minute_cursor]

    if len(minutes) >= 60:

        before_date = (datetime.strptime(date, '%Y-%m-%d') -
timedelta(days=20)).strftime('%Y-%m-%d')
        df_daily = self.dm.get_k_data(code, autype=None, begin_date=before_date,
end_date=date)
        if df_daily.index.size == 0:
            return None

        df_daily.set_index(['date'], 1, inplace=True)
```

ATR加仓 – 计算ATR

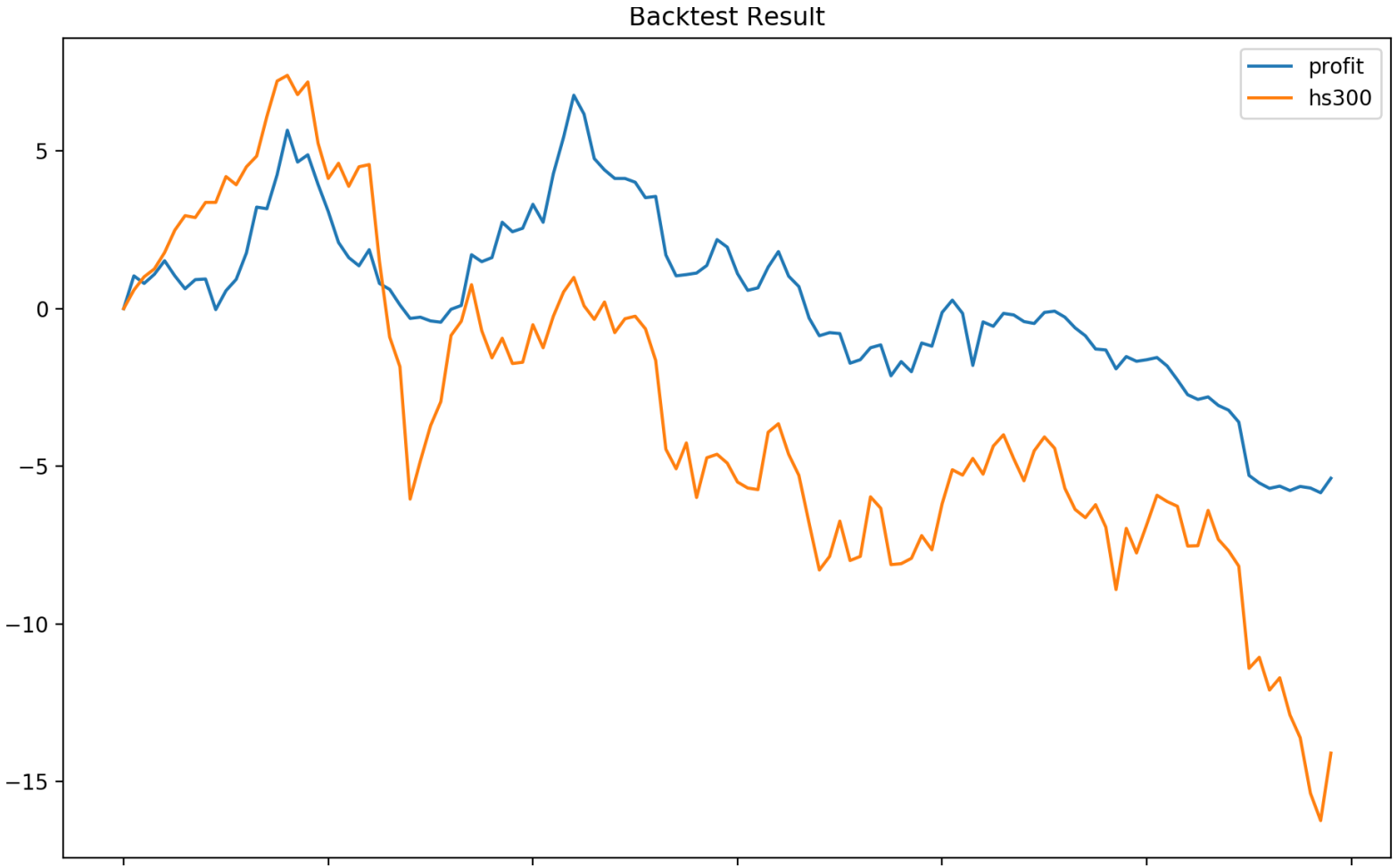
ATR加仓 – 计算ATR

```
 atr_df = DataFrame(columns=['tr'])
    for index in range(1, len(minutes)):
        minute = minutes[index]
        minute_date = minute['date']
        try:
            au_factor = df_daily.loc[minute_date]['au_factor']
            high = minute['high'] * au_factor
            low = minute['low'] * au_factor
            pre_close = minutes[index - 1]['close'] * au_factor
            atr_df.loc[index] = {
                'tr': max([high - low, abs(high - pre_close), abs(pre_close - low)])
            }
        except:
            print('计算ATR发生异常，股票代码：%s，  分钟线日期：%s' % (code, minute_date),
flush=True)
            traceback.print_exc()

    atr = atr_df.rolling(window=60).mean()['tr'][59]
    return atr
else:
    print('加仓策略（ATR），分钟线数据不足，股票代码：%s，日期：%s，分钟线数：%2d'
          % (code, date, len(minutes)), flush=True)
    return None
```

ATR加仓 – 计算加仓仓位

```
def compute_position(self, code, date):
    """
    计算应该分配的仓位， 这里采用固定仓位，总资金的0.015
    :param code: 股票代码
    :param date: 日期
    :return:
    """
    amount = self.strategy_option.capital() * 0.015
    return amount
```

ATR加仓的回测结果



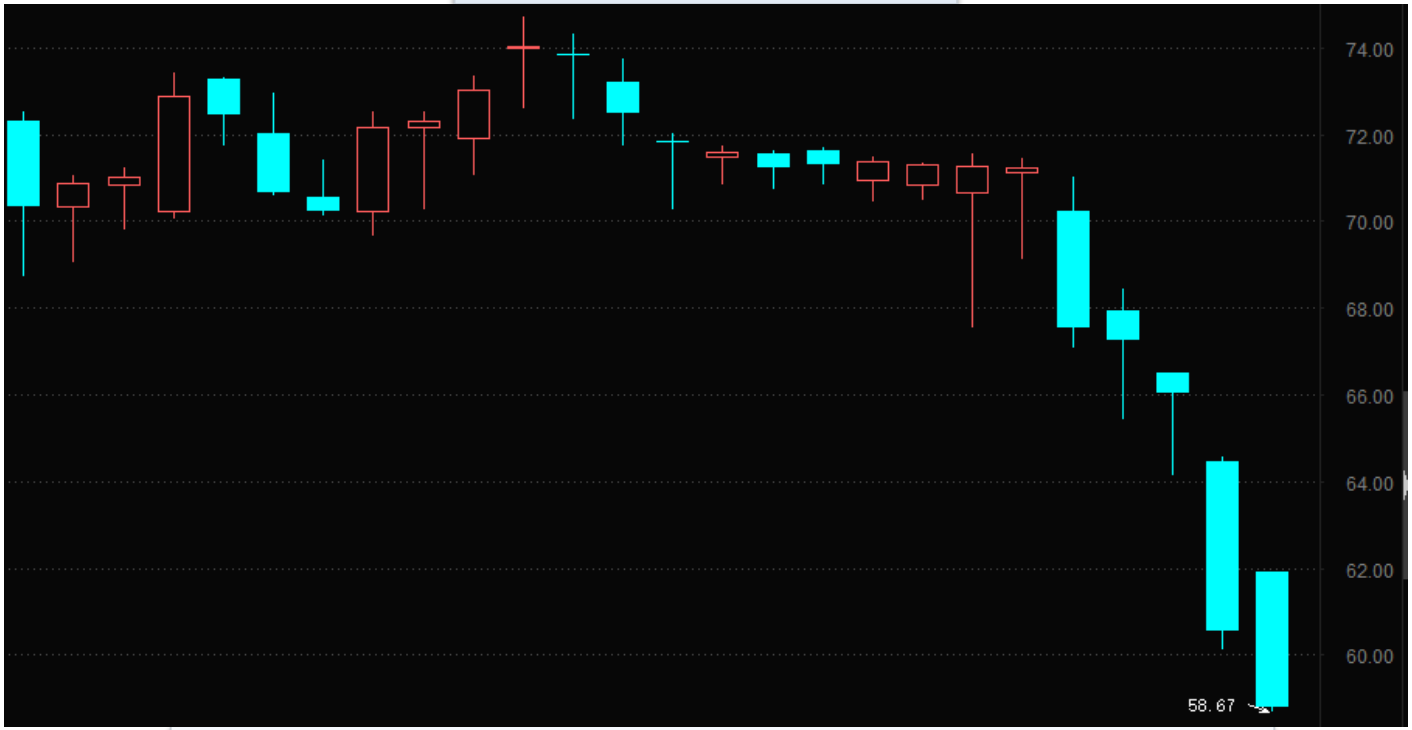
止损：触线止损和跟踪止损

常用的止损策略

- 触线止损
  - 一笔交易愿意承担的最大损失
  - 取值如果比较主观，可能错过反弹价
- 跟踪止损
  - 设定初始止损线
  - 随着价格上涨，止损线随之上浮
- 波动率标准差止损
  - 止损价设定为波动率的标准差
  - 例如，波动率的标准差为D，取止损价为D
  - 市场是非标准正态分布，需要10%的修正

最大亏损

买入：第一根K线收



问题：最大亏损设定为多少合适？

止损线的确定

- 方法一
  - 通过预期的收益风险比来确定
  - 假如收益风险比为4，股票预期收益为20%，则止损线为5%。
- 方法二
  - 考察股票特征，比如近期的波动情况

止损策略的工厂类

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

from .fix_stop_loss import FixStopLoss


class StopLossFactory:
    @staticmethod
    def get_stop_loss_policy(account, strategy_option):
        name = strategy_option.properties['stop_loss']
        if 'fixed' == name:
            max_loss = float(strategy_option.properties['max_loss'])
            return FixStopLoss(account, max_loss)
```

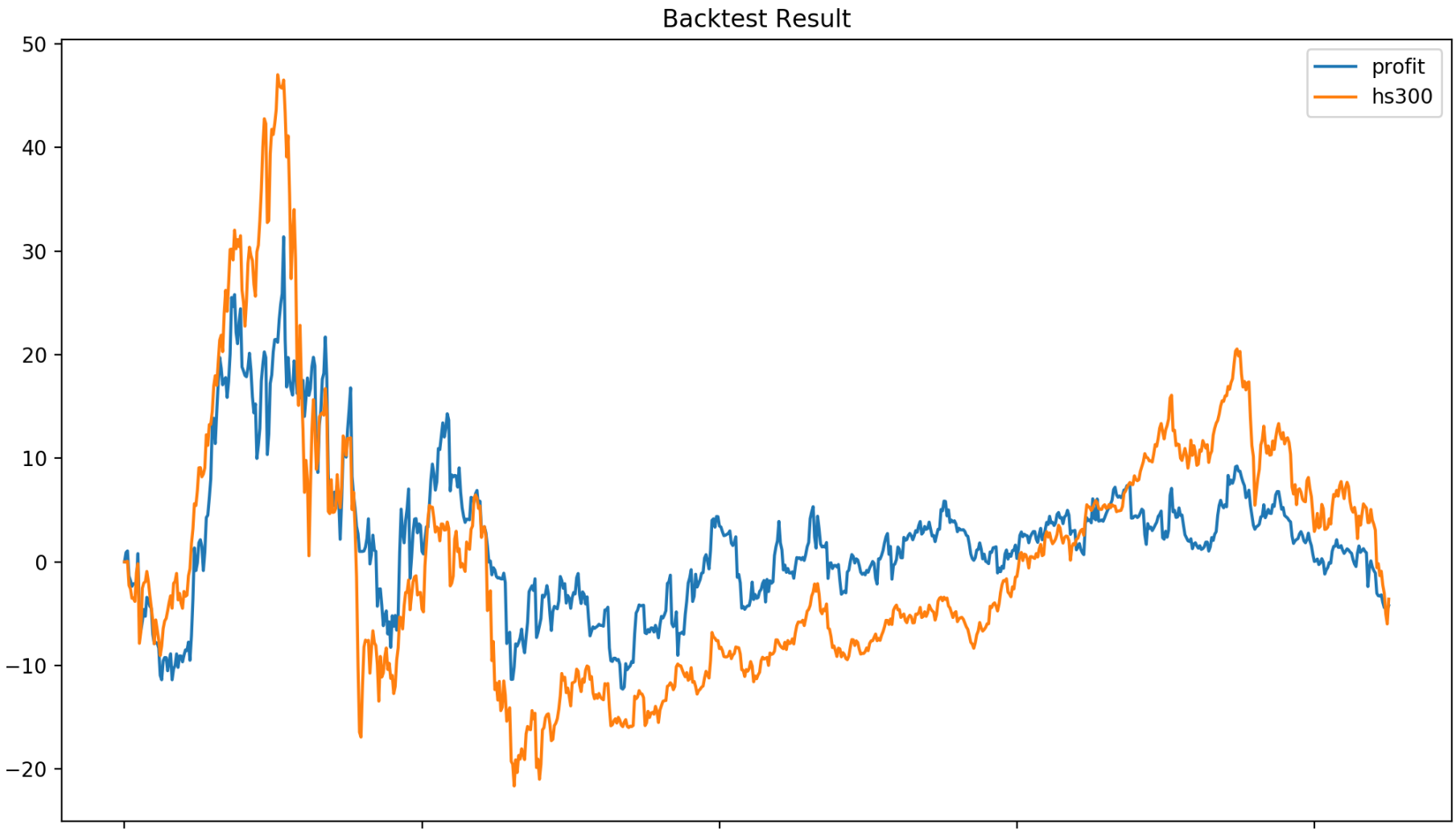
```
#  -*- coding: utf-8 -*-

from abc import abstractmethod


class BaseStopLoss:
    def __init__(self, account):
        self.account = account

    @abstractmethod
    def update_holding(self, code, date):
        """
        更新持仓股的一些信息
        :param code: 股票代码
        :param date: 日期
        """
        pass

    @abstractmethod
    def is_stop(self, code, date):
        """
        是否需要止损
        :param code: 股票代码
        :param date: 日期
        :return: True - 止损 False - 不止损
        """
        pass
```

止损的基类

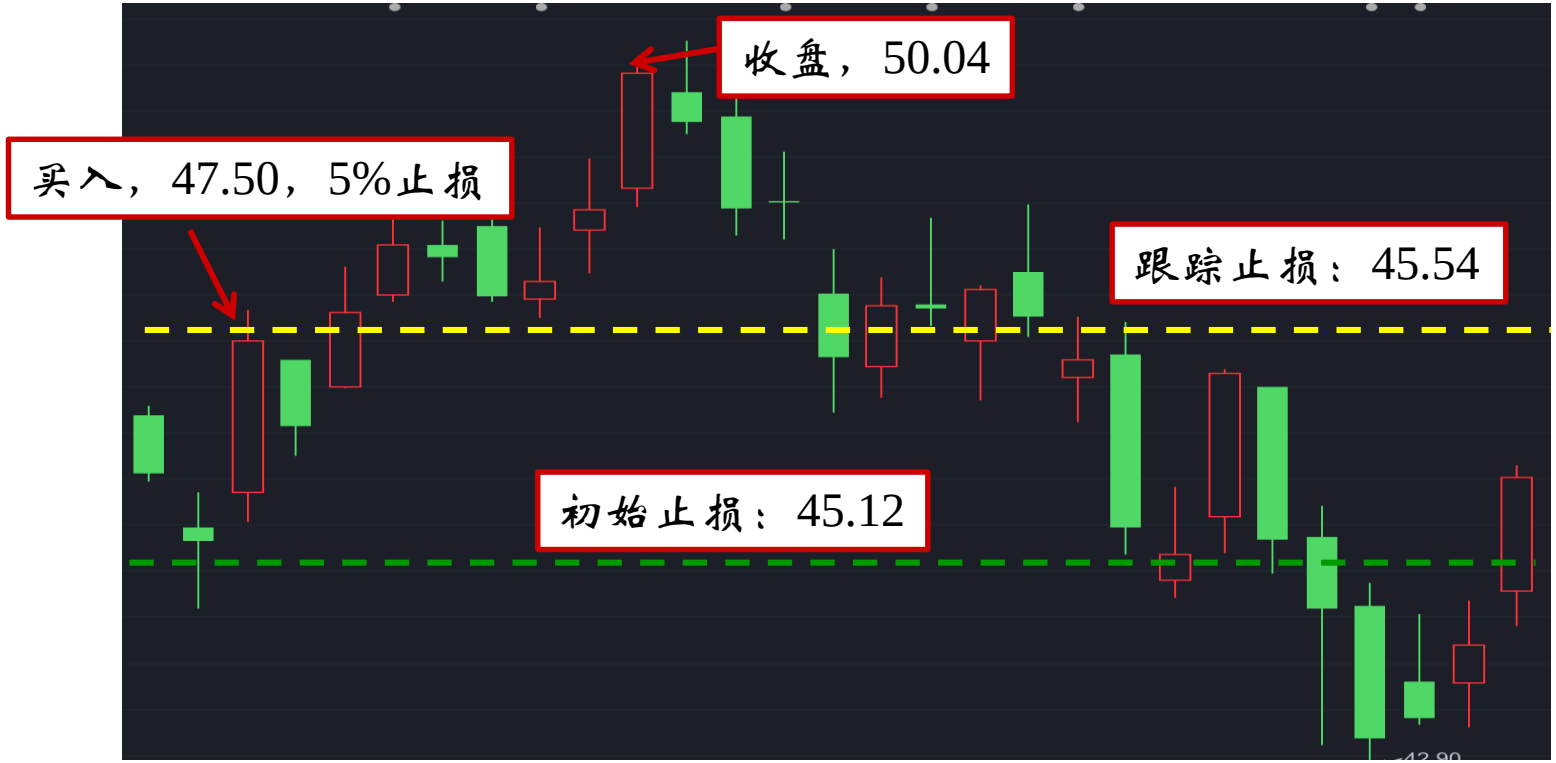
```
#  -*- coding: utf-8 -*-

from .base_stop_loss import BaseStopLoss
from data.data_module import DataModule


class FixStopLoss(BaseStopLoss):
    def __init__(self, account, max_loss):
        BaseStopLoss.__init__(self, account)
        self.max_loss = max_loss

    def update_holding(self, code, date):
        pass

    def is_stop(self, code, date):
        """
        判断股票在当前日期是否需要止损
        :param code: 股票代码
        :param date: 日期
        :return: True - 止损， False - 不止损
        """
        holding_stock = self.account.get_holding(code)
        dm = DataModule()
        if holding_stock is not None:
            df_daily = dm.get_k_data(code, autype=None, begin_date=date, end_date=date)
            if df_daily.index.size > 0:
                df_daily.set_index(['date'], 1, inplace=True)

                profit = (holding_stock['volume'] * df_daily.loc[date]['close'] - holding_stock['cost']) \
                        * 100 / holding_stock['cost']

                return (profit < 0) & (abs(profit) >= abs(self.max_loss))

        return False
```

触线止损

触线止损的回测结果



跟踪止损



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

from .base_stop_loss import BaseStopLoss
from data.data_module import DataModule


class TrackingStopLoss(BaseStopLoss):
    def __init__(self, account, max_loss):
        BaseStopLoss.__init__(self, account)
        self.max_loss = max_loss
        self.dm = DataModule()

    def update_holding(self, code, date):
        """
        更新持仓股的最高价
        :param code:
        :param date:
        :return:
        """
        df_daily = self.dm.get_k_data(code, autype='hfq', begin_date=date,
end_date=date)
        if df_daily.index.size > 0:
            df_daily.set_index(['date'], 1, inplace=True)
            close = df_daily.loc[date]['close']
            holding = self.account.get_holding(code)
            if 'highest' not in holding or holding['highest'] < close:
                holding['highest'] = close
                self.account.update_holding(code, holding)
```

跟踪止损－更新最高价

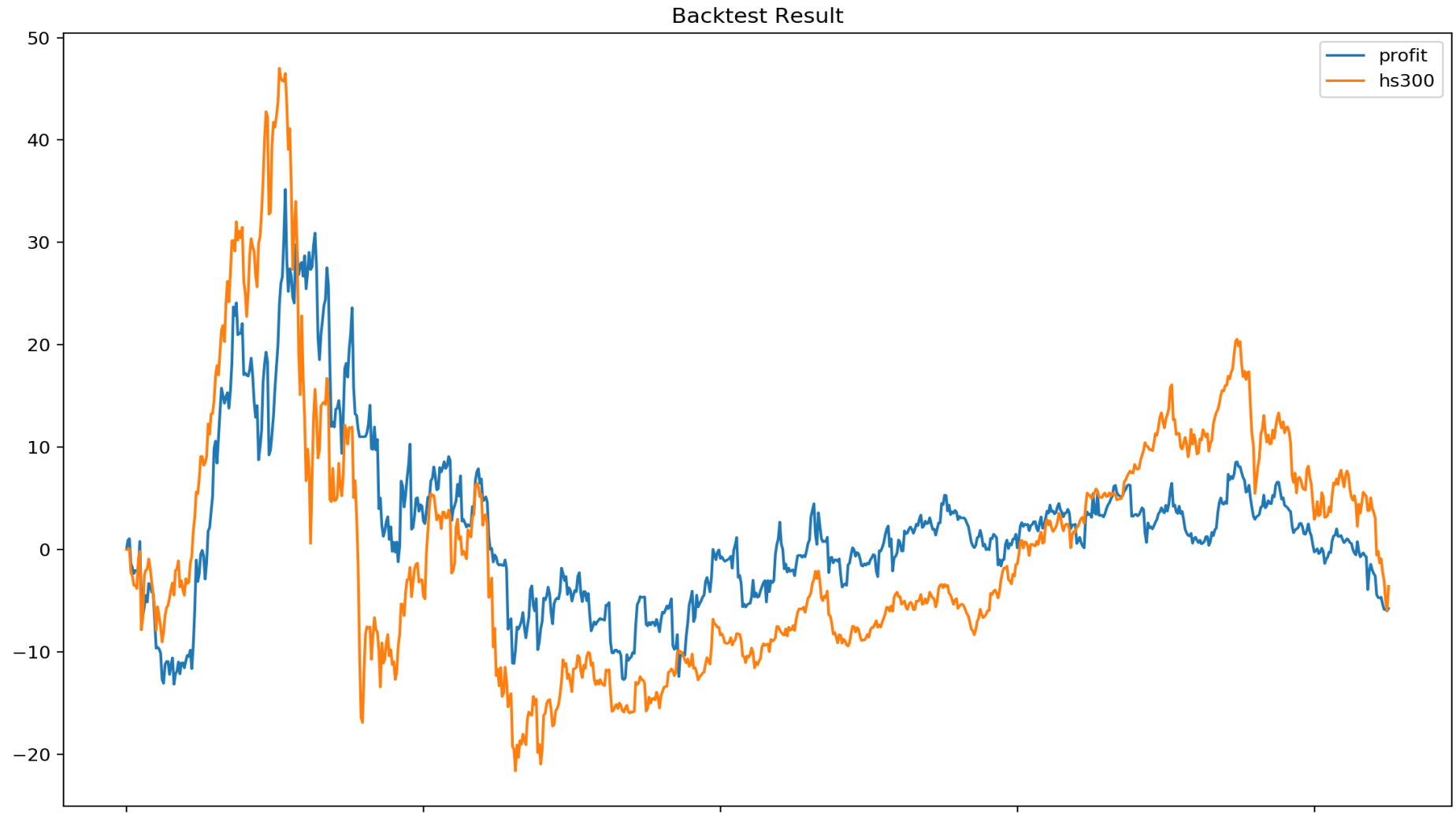
跟踪止损 – 判断是否需要止损

```
def is_stop(self, code, date):
        """
        判断股票在当前日期是否需要止损
        :param code: 股票代码
        :param date: 日期
        :return: True - 止损， False - 不止损
        """
        holding_stock = self.account.get_holding(code)
        if holding_stock is not None:
            df_daily = self.dm.get_k_data(code, autype='hfq', begin_date=date,
end_date=date)
            if df_daily.index.size > 0:
                df_daily.set_index(['date'], 1, inplace=True)
                close = df_daily.loc[date]['close']

                if 'highest' in holding_stock and close < holding_stock['highest']:
                    profit = (close - holding_stock['highest']) \
                             * 100 / holding_stock['highest']

                    return (profit < 0) & (abs(profit) >= abs(self.max_loss))

        return False
```

止损策略的工厂类

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

from .fix_stop_loss import FixStopLoss
from .tracking_stop_loss import TrackingStopLoss


class StopLossFactory:
    @staticmethod
    def get_stop_loss_policy(account, strategy_option):
        name = strategy_option.properties['stop_loss']
        if 'fixed' == name:
            max_loss = float(strategy_option.properties['max_loss'])
            return FixStopLoss(account, max_loss)
        elif 'tracking' == name:
            max_loss = float(strategy_option.properties['max_loss'])
            return TrackingStopLoss(account, max_loss)
```

跟踪止损的回测结果



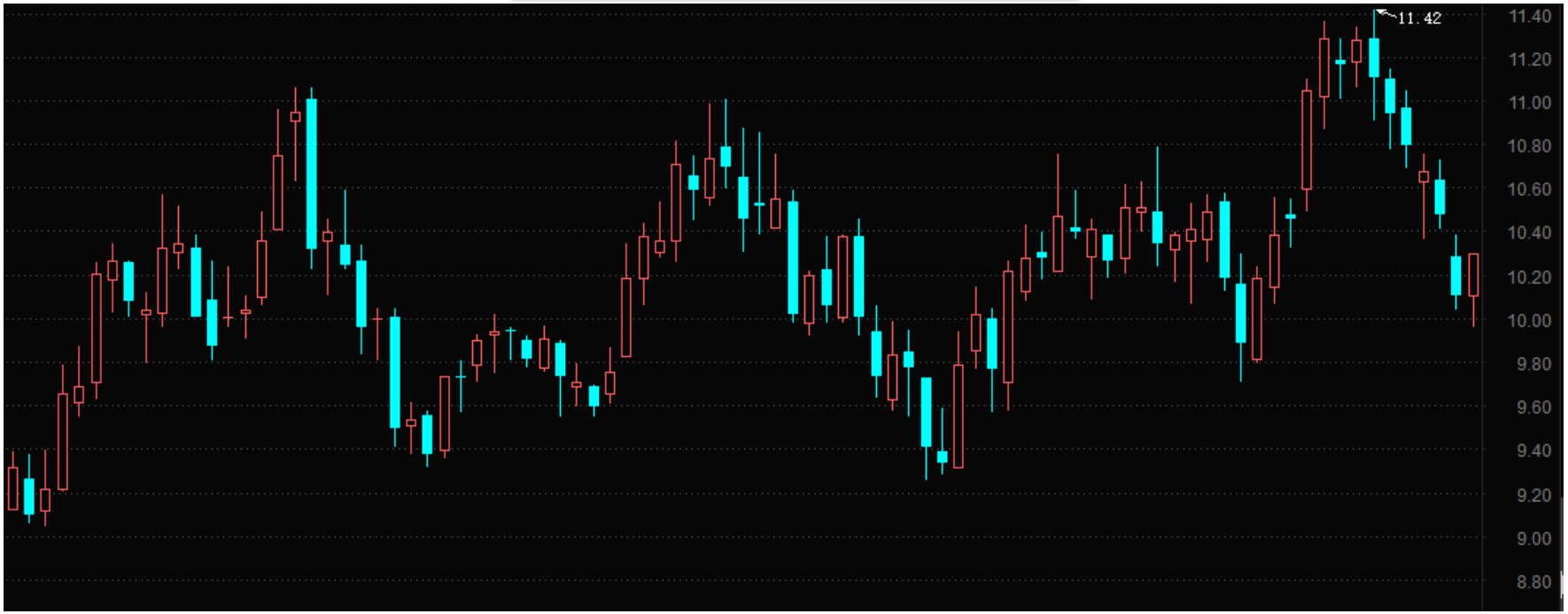
止盈：触线止盈和回撤止盈

常用的止盈策略

- 触线止盈
  - 一笔交易期望获得的最大利润
  - 设置为平均波动率的倍数
- 回撤止盈
  - 最新高点的固定折回百分比
  - 例如，获利10%以上回撤2.5%止盈
  - 适合程序化执行
- 大幅波动
  - 当市场出现不利于你的大幅波动时止盈
  - 例如，平均波动率为R，当出现2R时止盈
  - 不能单独使用，需要保护性止盈

最大利润

买入：第一根K线收盘



问题：按照最大盈利止盈方法，如何设定止盈线？

止盈线的设置

- 方法一
  - 股价上涨10%止盈
- 方法二
  - 考察股票特征，比如近期的波动情况
  - 该股票在上下20%区间内反复震荡
  - 止盈线：股价上涨20%止盈

止赢策略的工厂类

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


from .fix_stop_profit import FixStopProfit


class StopProfitFactory:
    @staticmethod
    def get_stop_profit_policy(account, strategy_option):
        name = strategy_option.properties['stop_profit']
        if 'fixed' == name:
            max_profit = float(strategy_option.properties['max_profit'])
            return FixStopProfit(account, max_profit)
```

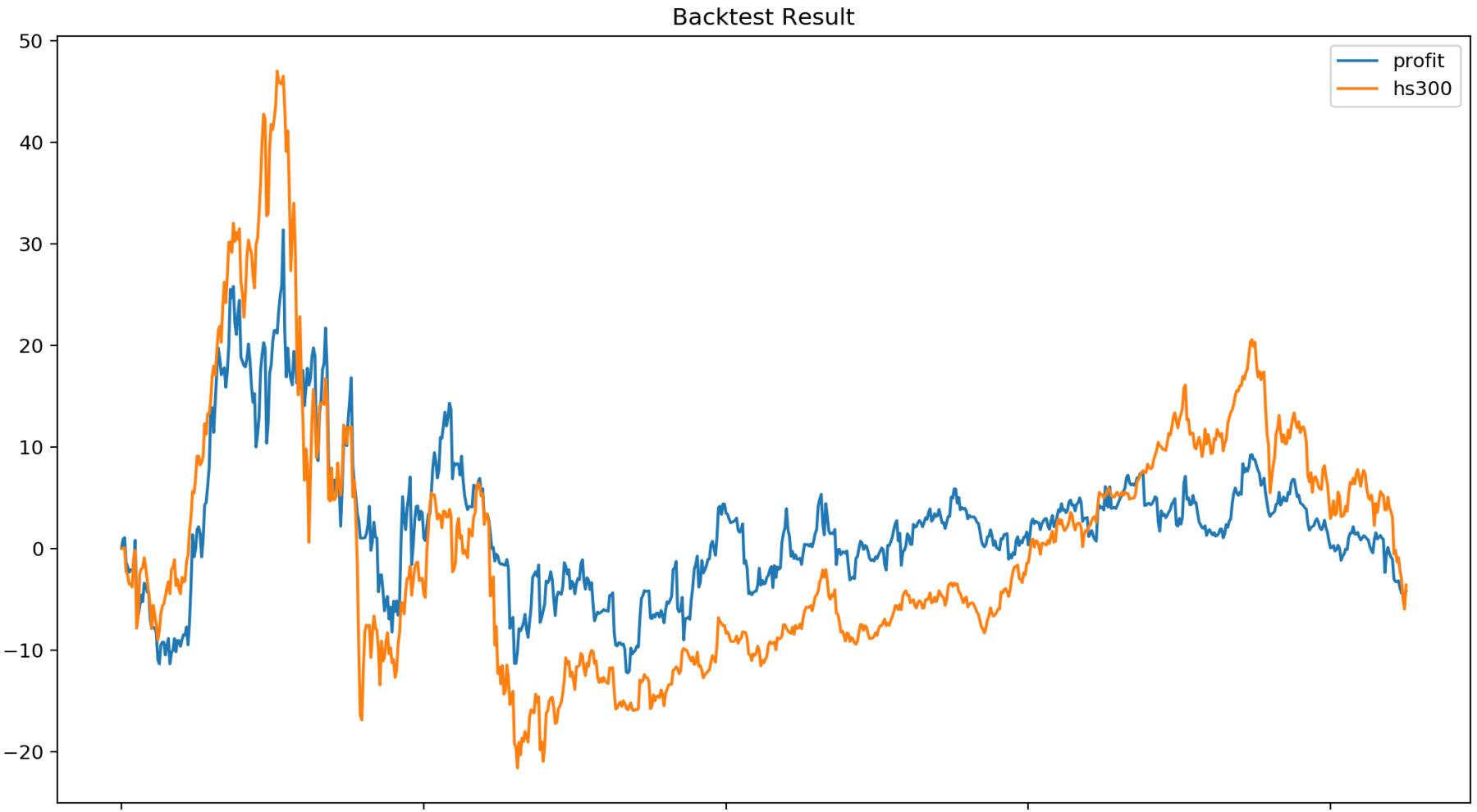
止赢的基类

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

from abc import abstractmethod


class BaseStopProfit:
    def __init__(self, account):
        self.account = account

    @abstractmethod
    def update_holding(self, code, date):
        """
        更新持仓股的一些信息
        :param code: 股票代码
        :param date: 日期
        """
        pass

    @abstractmethod
    def is_stop(self, code, date):
        """
        是否需要止损
        :param code: 股票代码
        :param date: 日期
        :return: True - 止损 False - 不止损
        """
        pass
```

触线止盈

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

from .base_stop_profit import BaseStopProfit
from data.data_module import DataModule


class FixStopProfit(BaseStopProfit):
    def __init__(self, account, max_profit):
        BaseStopProfit.__init__(self, account)
        self.max_profit = max_profit

    def update_holding(self, code, date):
        pass

    def is_stop(self, code, date):
        """
        判断是否需要止盈，盈利超过指定值，则止盈
        :param code: 股票代码
        :param date: 日期
        :return: True - 止盈，False - 不止盈
        """
        holding_stock = self.account.get_holding(code)
        dm = DataModule()
        if holding_stock is not None:
            df_daily = dm.get_k_data(code, autype=None, begin_date=date, end_date=date)
            if df_daily.index.size > 0:
                df_daily.set_index(['date'], 1, inplace=True)

                profit = (holding_stock['volume'] * df_daily.loc[date]['close'] - holding_stock['cost']) \
                         * 100 / holding_stock['cost']

                return profit >= self.max_profit

        return False
```

触线止盈的回测结果



回撤止盈

- 第一个黄圈
  - 股价达到48，符合上涨20%以上
  - 回撤为6%左右

- 第二个黄圈
  - 股价达到56，符合上涨20%以上
  - 回撤为10%左右

```
def update_holding(self, code, date):
    “””
    更新持仓股超过指定收益后的最高市值
    :param code:
    :param date:
    :return:
    """
    df_daily = self.dm.get_k_data(code, autype=None, begin_date=date, end_date=date)
    if df_daily.index.size > 0:
        df_daily.set_index(['date'], 1, inplace=True)
        close = df_daily.loc[date]['close']
        holding = self.account.get_holding(code)

        current_value = holding['volume'] * close
        if 'highest_value' in holding:
            if current_value > holding['highest_value']:
                holding['highest_value'] = current_value
                self.account.update_holding(code, holding)
        else:
            # 判断是否已经达到了最高的收益线
            profit = (current_value - holding[’cost']) * 100/holding[’cost']
            if profit > self.max_profit:
                holding['highest_value'] = current_value
                self.account.update_holding(code, holding)
```

更新持仓股超过最大收益后的最大市值

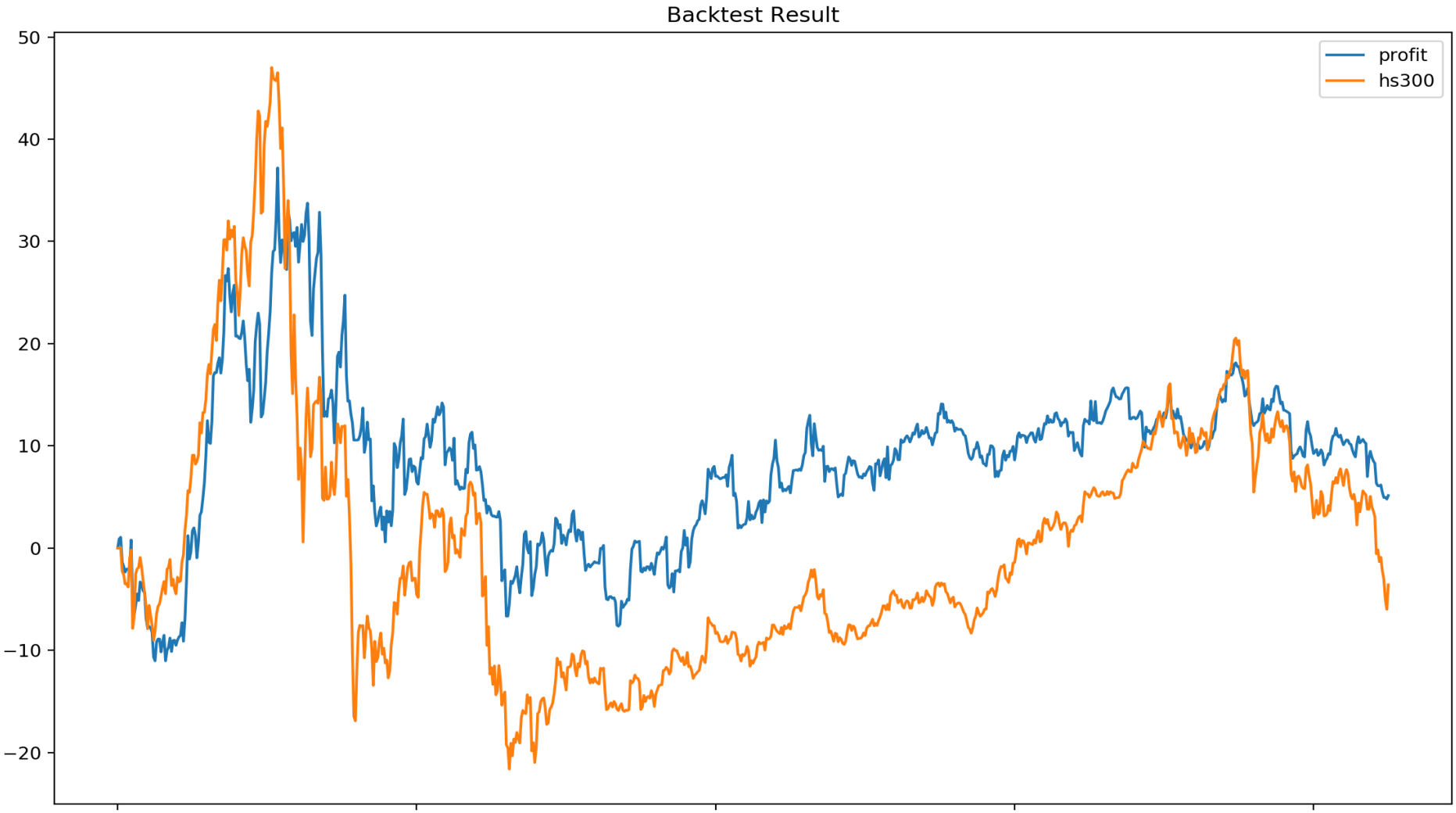
```
def is_stop(self, code, date):
    """
    判断股票在当前日期是否需要止盈，如果当前收益相对于最高收益的回撤已经达到了指定幅度，则止盈
    :param code: 股票代码
    :param date: 日期
    :return: True - 止盈， False - 不止盈
    """
    holding_stock = self.account.get_holding(code)
    if holding_stock is not None:
        df_daily = self.dm.get_k_data(code, autype=None, begin_date=date, end_date=date)
        if df_daily.index.size > 0:
            df_daily.set_index(['date'], 1, inplace=True)
            current_value = df_daily.loc[date]['close'] * holding_stock['volume']
            # 计算回落的百分比
            if 'highest_value' in holding_stock and current_value <
holding_stock['highest_value']:
                profit = (current_value - holding_stock['highest_value']) \
                        * 100 / holding_stock['highest_value']

                return profit < 0 and abs(profit) >= abs(self.profit_drawdown)

    return False
```

根据有过的最大市值和当前市值判断是否需要止盈

止赢策略的工厂类

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


from .fix_stop_profit import FixStopProfit
from .tracking_stop_profit import TrackingStopProfit


class StopProfitFactory:
    @staticmethod
    def get_stop_profit_policy(account, strategy_option):
        name = strategy_option.properties['stop_profit']
        if 'fixed' == name:
            max_profit = float(strategy_option.properties['max_profit'])
            return FixStopProfit(account, max_profit)
        if 'tracking' == name:
            max_profit = float(strategy_option.properties['max_profit'])
            profit_drawdown = float(strategy_option.properties['profit_drawdown'])
            return TrackingStopProfit(account, max_profit, profit_drawdown)
```

回撤止盈的回测结果



总结

- 策略管理子系统
  - 配置化的策略制作
  - 股票池的动态加载
- 交易决策子系统
  - 信号模块的实现
  - 仓位管理的实现
  - 止损的概念和实现
  - 止盈的概念和实现

课后练习

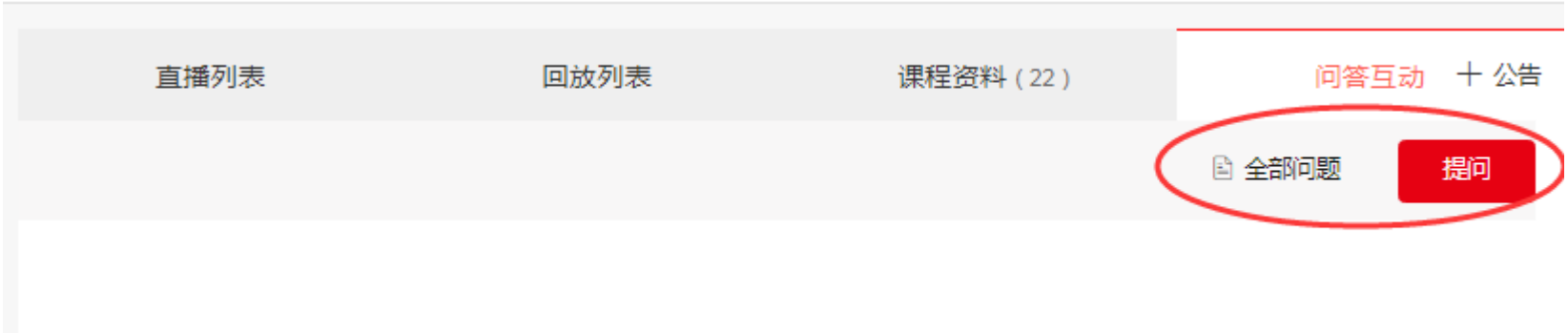
- 实现一个止损或者止盈的方法：
  - 编写源码
  - 完成回测，对结果进行分析（需附结果图），
  - 和没有使用该方法的回测结果比较，分析回测变好或者变差的原因

下节课预告

- 题目：执行子系统的实现——模拟撮合和实盘接口
  - 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