# 导入函数库

from jqdata import \*

import pandas as pd

import datetime

#from datetime import datetime

import time

import math

import datetime

import copy

import pandas as pd

pd.set\_option('display.max\_rows',500)

pd.set\_option('display.max\_columns',500)

pd.set\_option('display.width',1000)

# 初始化函数，设定基准等等

def initialize(context):

# 设定沪深300作为基准

set\_benchmark('000300.XSHG')

# 开启动态复权模式(真实价格)

set\_option('use\_real\_price', True)

# 输出内容到日志 log.info()

log.info('初始函数开始运行且全局只运行一次')

# 过滤掉order系列API产生的比error级别低的log

log.set\_level('order', 'error')

### 股票相关设定 ###

# 股票类每笔交易时的手续费是：买入时佣金万分之三，卖出时佣金万分之三加千分之一印花税, 每笔交易佣金最低扣5块钱

set\_order\_cost(OrderCost(close\_tax=0.001, open\_commission=0.0001, close\_commission=0.0003, min\_commission=5), type='stock')

#设定全局变量

g.year\_num = 3

g.revenue\_thre = 19

g.profit\_thre = 10

g.buylist = None

g.buylist\_this\_month = None

g.buylist\_ss = None

g.buylist\_sum = []

g.max\_total\_value = 0

#设定年化预期收益率，计算每月预期收益率

#当实际上每月收益率高于设定收益率x倍时，认为市场过热，减少股票池总市值至预期值

#当实际上每月收益率低于设定收益率1/x倍时，认为市场过冷，增加股票池总市值至预期值

g.last\_month\_total\_value = context.portfolio.total\_value

g.set\_return\_rate = 18

g.set\_return\_rate\_month = math.pow((100+g.set\_return\_rate)/100, 1/12 )-1

#寻找股票

# 第一步：找股票

# 第二部：鉴别股票

run\_monthly(find\_stocks, monthday=1, time='10:10')

#择时卖出

run\_monthly(select\_right\_time\_sell,monthday=3,time='10:10')

#择时买入

#1、一个月内最低点上涨10%买入

run\_monthly(select\_right\_time\_buy,monthday=6,time='10:10')

#市场过冷过热矫正

run\_monthly(hot\_cold\_correct\_frame,monthday=9,time='10:10')

#进行资产平衡

run\_monthly(assets\_balance\_frame,monthday=12,time='10:10')

#run\_daily(assets\_balance\_frame,time='every\_bar')

# 测试区域

#select\_stocks\_list = select\_stocks(context)

# print(select\_stocks\_list['code'])

# print(select\_stocks\_list['name'])

# print(len(select\_stocks\_list))

#run\_daily(period,time='every\_bar')

#run\_monthly(find\_stocks, monthday=1, time='10:30')

#stock\_code = '000603.XSHG'

#check\_stock(context, stock\_code)

#profit\_margin = get\_enterprise\_gross\_profit\_margin\_single\_enter(context, stock\_code)

#print(profit\_margin)

#find\_stocks(context)

#select\_right\_time\_buy(context)

#print(g.buylist)

# run\_monthly(func, monthday, time='9:30', reference\_security, force=False)

#获取年份列表

def get\_recent\_year\_list(context, num):

currentYear = context.current\_dt.year

year\_list = list(range(currentYear-1, currentYear-num-1, -1))

return year\_list

#获取当前时间的股票列表

def get\_stocks\_list(context):

date = context.current\_dt.strftime("%Y-%m-%d")

stocks\_list = get\_all\_securities(['stock'], date=date)

stocks\_list = stocks\_list[~stocks\_list.display\_name.str.contains('\\*|ST|退', regex=True)]

stock\_list = list(stocks\_list.index)

return list(stocks\_list.index)

def get\_stocks\_name(context):

date =context.current\_dt.strftime("%Y-%m-%d")

stocks\_list = get\_all\_securities(['stock'], date=date)

stock\_name = stocks\_list[['display\_name']]

stock\_name =stock\_name.reset\_index(drop = False)

stock\_name = stock\_name.rename(columns = {"index": "code"})

return stock\_name

#print(stocks\_list.columns)

#获取最近一年的毛利率

def get\_enterprise\_gross\_profit\_margin(stock\_list, statDateYear, gross\_profit\_margin\_thre=20):

df = get\_fundamentals(query(

indicator.code, indicator.gross\_profit\_margin

).filter(

indicator.gross\_profit\_margin > gross\_profit\_margin\_thre,

indicator.code.in\_(stock\_list)

).order\_by(

# 按市值降序排列

indicator.gross\_profit\_margin.desc()

).limit(

# 最多返回100个

4000

),

statDate= statDateYear)

return df

#获取某只股票最近一个季度的毛利率

def get\_enterprise\_gross\_profit\_margin\_single\_enter(context, stock\_code):

current\_day = context.current\_dt.strftime("%Y-%m-%d")

#current\_day = "2008-08-04"

#print(current\_day)

#print(stock\_code)

df = get\_history\_fundamentals(stock\_code, fields=[indicator.gross\_profit\_margin],

watch\_date=current\_day, count=1, interval='1q', stat\_by\_year=False)

#print(df)

if len(df)>0:

gross\_profit\_margin\_single\_enter = list(df['gross\_profit\_margin'])[0]

#print(stock\_code+"在"+current\_day+"的毛利:"+str(gross\_profit\_margin\_single\_enter))

else:

gross\_profit\_margin\_single\_enter = 40

#print("未找到"+stock\_code+"在"+current\_day+"的毛利,设为默认值40")

return gross\_profit\_margin\_single\_enter

def check\_stocks\_gross\_profit\_margin(context, stock\_code,gross\_profit\_margin=10):

gross\_profit\_margin\_single\_enter = get\_enterprise\_gross\_profit\_margin\_single\_enter(context, stock\_code)

if gross\_profit\_margin\_single\_enter > gross\_profit\_margin:

return True

else:

return False

#获取股票的列表的营收增长率

def get\_enterprise\_revenue(stock\_list, statDateYear, revenue\_thre, profit\_thre=-0.1):

df = get\_fundamentals(query(

indicator.code, indicator.inc\_total\_revenue\_year\_on\_year

).filter(

indicator.inc\_total\_revenue\_year\_on\_year > revenue\_thre,

indicator.inc\_net\_profit\_year\_on\_year > profit\_thre,

indicator.code.in\_(stock\_list)

).order\_by(

# 按市值降序排列

indicator.inc\_total\_revenue\_year\_on\_year.desc()

).limit(

# 最多返回100个

4000

),

statDate= statDateYear)

return df

#获取最近几年营收增长大于阈值的企业

def get\_enterprice\_revenue\_recent\_year(context, year\_num, revenue\_thre, profit\_thre):

year\_list = get\_recent\_year\_list(context, year\_num)

#输入研究的当前节点

stock\_list = get\_stocks\_list(context)

df\_sum = None

for year in year\_list:

df\_current\_year = get\_enterprise\_revenue(stock\_list, year, revenue\_thre, profit\_thre)

#print(year, "年数据量大小:", len(df\_current\_year) )

new\_revenue\_name = str(year) + "\_inc\_total\_revenue\_year\_on\_year"

df\_current\_year = df\_current\_year.rename(columns = {"inc\_total\_revenue\_year\_on\_year": new\_revenue\_name})

if df\_sum is None:

df\_sum = df\_current\_year

else:

df\_sum = pd.merge(df\_sum, df\_current\_year)

#print("合并数据量大小：",len(df\_sum) )

#获取最近一年的毛利率

gross\_profit\_margin = get\_enterprise\_gross\_profit\_margin(stock\_list, year\_list[0], gross\_profit\_margin\_thre=20)

df\_sum = pd.merge(df\_sum, gross\_profit\_margin)

#print("毛利率筛选之后数据量大小：",len(df\_sum) )

#df\_sum.to\_csv('fundamentaldfsum.csv',encoding = 'utf\_8\_sig')

#增加公司名称

stocks\_name = get\_stocks\_name(context)

#print(stocks\_name)

df\_sum = pd.merge(df\_sum, stocks\_name)

stock\_name\_tmp = df\_sum.display\_name

df\_sum = df\_sum.drop('display\_name',axis=1)

df\_sum.insert(1,'name',stock\_name\_tmp)

return df\_sum

#选择股票

def select\_stocks(context):

df = get\_enterprice\_revenue\_recent\_year(context, g.year\_num, g.revenue\_thre, g.profit\_thre)

return df

#检验多股票

def check\_stocks(context, buylist):

#检验历年的财报， 营收

current\_day = context.current\_dt.strftime("%Y-%m-%d")

i = 0

buylist\_check = []

for stock\_code in buylist:

i = i + 1

#print('进度:'+str(i)+'/'+str(len(buylist)))

check\_flag = check\_stock(context, stock\_code)

check\_margin = check\_stocks\_gross\_profit\_margin(context, stock\_code,gross\_profit\_margin=15)

if check\_flag > 0:

#print('未增长季度个数为：'+str(check\_flag)+'，移除：'+stock\_code)

pass

elif check\_margin == False:

#print('最近一季财报毛利率低于阈值：'+str(check\_flag)+'，移除：'+stock\_code)

pass

else:

buylist\_check.append(stock\_code)

return buylist\_check

def check\_stock(context, stock\_code):

current\_day = context.current\_dt.strftime("%Y-%m-%d")

stock\_info = get\_security\_info(stock\_code)

stock\_start\_year = stock\_info.start\_date.year

current\_year = datetime.datetime.strptime(current\_day, "%Y-%m-%d").year

df = get\_fundamentals(query(indicator).filter(indicator.code == stock\_code),date=current\_day)

last\_mom = list(df['statDate'])[0]

current\_loc = datetime.datetime.strptime(last\_mom, "%Y-%m-%d").month/3

current\_loc = int(current\_loc)

df\_fundamentals = None

for year in range(current\_year, stock\_start\_year-1,-1):

for i in range(current\_loc, 0,-1):

fundamental\_date = str(year)+'q'+str(i)

#print(fundamental\_date)

df = get\_fundamentals(query(indicator).filter(indicator.code == stock\_code),

statDate= fundamental\_date)

#df

if df\_fundamentals is None:

df\_fundamentals = df

else:

df\_fundamentals = df\_fundamentals.append(df)

current\_loc = 4

#df\_fundamentals

df\_fundamentals['revenue\_year\_flag'] = df\_fundamentals['inc\_total\_revenue\_year\_on\_year'].apply(lambda x: -1 if x <0 else 0)

df\_fun\_reve\_list = list(df\_fundamentals['revenue\_year\_flag'])

#print(df\_fundamentals[['statDate.1','inc\_total\_revenue\_year\_on\_year', 'revenue\_year\_flag']])

df\_fun\_reve\_list = list(df\_fundamentals['revenue\_year\_flag'])

stock\_reve\_count = df\_fun\_reve\_list.count(-1)

#print(stock\_code+'营收未增长季度计数：', stock\_reve\_count)

return stock\_reve\_count

#择时买入

#1、一个月内最低点上涨10%买入

#择时卖出

#1、持有收益率小于-10%卖出

#2、 比最近的一段时间的最高点下降-10卖出

#

def find\_stocks(context):

# 代码：找出市值排名最小的前stocksnum只股票作为要买入的股票

# 获取上证指数和深证综指的成分股代码并连接，即为全A股市场所有股票的股票代码

# 用加号可以连接两个list

stock\_start\_month =context.current\_dt.month

print("月份：",stock\_start\_month)

if stock\_start\_month != 8:

print("跳过")

return

else:

print("是八月")

df = select\_stocks(context)

# 选取股票代码并转为list

g.buylist\_ss = list(df['code'])

#计算应该买的股票总和，历史的股票+加上当期的寻找到的

#将不合格的筛选出去

g.buylist\_sum = g.buylist\_sum + g.buylist\_ss

print("筛选前股票数量："+str(len(g.buylist\_sum)))

print(g.buylist\_sum)

g.buylist\_sum = check\_stocks(context, g.buylist\_sum)

print("筛选后股票数量："+str(len(g.buylist\_sum)))

print(g.buylist\_sum)

return

#择时买入，对于每只股票在一个月内的最低点上涨10买入

#3.0版本 直接买 不等待

def select\_right\_time\_buy(context):

print("开始买入")

g.buylist\_this\_month = [i for i in g.buylist\_sum if i not in context.portfolio.positions.keys()]

g.buylist\_this\_month = check\_stocks(context, g.buylist\_this\_month)

print("本月要买"+str(len(g.buylist\_this\_month)))

print(g.buylist\_this\_month )

stock\_num = len(g.buylist\_this\_month)

if stock\_num < 1:

g.buylist\_this\_month = copy.deepcopy(g.buylist\_sum)

stock\_num = len(g.buylist\_this\_month)

if stock\_num <1:

return

position\_per\_stk = context.portfolio.available\_cash / stock\_num

position\_per\_stk\_the = context.portfolio.total\_value / 10

position\_per\_stk = position\_per\_stk if position\_per\_stk < position\_per\_stk\_the else position\_per\_stk\_the

for stock\_code in g.buylist\_this\_month:

close\_price\_30 = attribute\_history(stock\_code, 30, '1d', ['close'])

#print(close\_price\_30)

min\_price\_30 = close\_price\_30['close'].min()

current\_price = close\_price\_30['close'][-1]

if current\_price/ min\_price\_30 - 1 >0.1 :

position\_per\_stk = position\_per\_stk if position\_per\_stk/current\_price > 100 else current\_price\*200

order\_bak = order\_value(stock\_code, position\_per\_stk)

if order\_bak is not None:

g.buylist\_this\_month.remove(stock\_code)

#print('买入'+str(position\_per\_stk)+'的'+stock\_code)

else:

#print('买入'+str(position\_per\_stk)+'的'+stock\_code+'失败')

pass

#择时卖出

#1、3.0版本 卖一家公司的理由就是公司的基本面发生了变化

def select\_right\_time\_sell(context):

g.buylist\_sum = check\_stocks(context, g.buylist\_sum)

for stock\_code in context.portfolio.positions:

if stock\_code not in g.buylist\_sum:

order\_bak = order\_target\_value(stock\_code, 0)

if order\_bak is not None:

#print('基本面发生变化全部卖出：'+stock\_code)

pass

return

#资产再平衡

def assets\_balance(context):

stock\_num = len(context.portfolio.positions)

position\_per\_stk\_target = context.portfolio.positions\_value /(stock\_num+1) # 1代表现金份额

print("进行资产平衡")

for stock\_code in context.portfolio.positions:

order\_bak = order\_target\_value(stock\_code, position\_per\_stk\_target)

if order\_bak is not None:

print(stock\_code + "调整至"+ str(position\_per\_stk\_target))

return

#资产再平衡框架

def assets\_balance\_frame(context):

value\_list = [0]

for stock\_code in context.portfolio.positions:

value\_list.append( context.portfolio.positions[stock\_code].value)

value\_max = max(value\_list)

#如果某类资产超过总资产的20%，则进行资产再平衡

if value\_max > context.portfolio.total\_value\*0.2:

assets\_balance(context)

# g.max\_total\_value = context.portfolio.total\_value if context.portfolio.total\_value > g.max\_total\_value else g.max\_total\_value

# #如果产生了一次10%的回撤，进行一次资产再平衡

# if context.portfolio.total\_value / g.max\_total\_value < 0.9:

# assets\_balance(context)

# g.max\_total\_value = context.portfolio.total\_value

return

def hot\_cold\_correct(context, coefficient):

for stock\_code in context.portfolio.positions:

stk\_target = context.portfolio.positions[stock\_code].value \* coefficient

#stk\_target = round(stk\_target/100)\*100

order\_bak = order\_target\_value(stock\_code, stk\_target)

if order\_bak is not None:

print(stock\_code + "调整至"+ str(stk\_target))

return

#市场过冷过热调节

def hot\_cold\_correct\_frame(context):

print("设定月收益率"+str(g.set\_return\_rate\_month))

print("资产增值情况"+str(context.portfolio.total\_value / g.last\_month\_total\_value))

if context.portfolio.total\_value / g.last\_month\_total\_value -1 > g.set\_return\_rate\_month \* 4:

print("市场过热")

coe = g.last\_month\_total\_value / context.portfolio.total\_value \* (1+g.set\_return\_rate\_month)

hot\_cold\_correct(context, coe)

elif context.portfolio.total\_value / g.last\_month\_total\_value -1 < g.set\_return\_rate\_month \* 1/ 4:

print("市场过冷")

coe = g.last\_month\_total\_value / context.portfolio.total\_value \* (1+g.set\_return\_rate\_month)

hot\_cold\_correct(context, coe)

else:

print("市场冷热正常不进行调节")

pass

g.last\_month\_total\_value = context.portfolio.total\_value