MyQuant新建Python策略

1.空策略

不使用模板。

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *

# 策略中必须有init方法
def init(context):
    pass

if __name__ == '__main__':
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')
```

2.定时任务(典型场景)

典型如选股交易策略，比如，策略每日收盘前10分钟执行：选股->决策逻辑->交易->退出，可能无需订阅实时数据。

```
# coding=utf-8
from __future__ import print_function, absolute_import

from gm.api import *


def init(context):
    # 每天14:50 定时执行algo任务
    schedule(schedule_func=algo, date_rule='daily', time_rule='14:50:00')


def algo(context):
    # 购买200股浦发银行股票
    order_volume(symbol='SHSE.600000', volume=200, side=1,
                 order_type=2, position_effect=1, price=0)


# 查看最终的回测结果
def on_backtest_finished(context, indicator):
    print(indicator)


if __name__ == '__main__':
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')

```

3.数据事件驱动(典型场景)

策略订阅的每个代码的每一个bar，都会触发策略逻辑。

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *


def init(context):
    # 订阅浦发银行，bar频率为一天
    subscribe(symbols='SHSE.600000', frequency='1d')


def on_bar(context, bars):
    # 打印当前获取的bar信息
    print(bars)

```

```
15
16 if __name__ == '__main__':
17     run(strategy_id='strategy_id',
18         filename='main.py',
19         mode=MODE_BACKTEST,
20         token='token_id',
21         backtest_start_time='2016-06-17 13:00:00',
22         backtest_end_time='2017-08-21 15:00:00')
23
```

4.时间序列数据事件驱动(典型场景)

策略订阅代码时指定数据窗口大小与周期，平台创建数据滑动窗口，加载初始数据，并在新的bar到来时自动刷新数据。bar事件触发时，策略可以取到订阅代码的准备好的时间序列数据。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      # 指定数据窗口大小为50
8      subscribe(symbols='SHSE.600000', frequency='1d', count=50)
9
10
11 def on_bar(context, bars):
12     # 打印频率为一天的浦发银行的50条最新bar的收盘价和bar开始时间
13     print(context.data(symbol='SHSE.600000', frequency='1d', count=50,
14                        fields='close,bob'))
15
16
17 if __name__ == '__main__':
18     run(strategy_id='strategy_id',
19         filename='main.py',
20         mode=MODE_BACKTEST,
21         token='token_id',
22         backtest_start_time='2016-06-17 13:00:00',
23         backtest_end_time='2017-08-21 15:00:00')
24
```

5.多个代码数据事件驱动(典型场景)

策略订阅多个代码，并且要求同一频度的数据到齐后，再触发事件。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      # 同时订阅浦发银行和平安银行,数据全部到齐再触发事件
```

```
    subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d', count=5,
              wait_group=True)


def on_bar(context, bars):
    for bar in bars:
        print(bar['symbol'], bar['eob'])


if __name__ == '__main__':
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')

```

6.默认账户交易(典型场景)

默认账户进行交易，下单时不指定account。

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *


def init(context):
    subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d')


def on_bar(context, bars):
    for bar in bars:
        # 不指定account 使用默认账户下单
        order_volume(symbol=bar['symbol'], volume=200, side=1,
                     order_type=2, position_effect=1, price=0)


# 查看最终的回测结果
def on_backtest_finished(context, indicator):
    print(indicator)


if __name__ == '__main__':
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')

```

7.显式指定交易账户(典型场景)

下单时指定交易账户，account等于账户id或者账户标题。

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *


def init(context):
    subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d')


def on_bar(context, bars):
    for bar in bars:
        # account等于账户id 或者账户标题 指定交易账户
        order_volume(symbol=bar['symbol'], volume=200, price=0, side=1,
                     order_type=2, position_effect=1, account='xxxxx')


# 查看最终的回测结果
def on_backtest_finished(context, indicator):
    print(indicator)


if __name__ == '__main__':
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2016-06-17 13:00:00',
        backtest_end_time='2017-08-21 15:00:00')

```

8.模式选择(典型场景)

策略支持两种运行模式，实时模式和回测模式，用户需要在运行策略时选择模式，执行run函数时mode=1表示回测模式，mode=0表示实时模式。

```
# coding=utf-8
from __future__ import print_function, absolute_import
from gm.api import *


def init(context):
    # 订阅浦发银行的tick
    subscribe(symbols='SHSE.600000', frequency='tick')


def on_tick(context, tick):
    # 打印当前获取的tick信息
    print(tick)
```

```
14
15
16 if __name__ == '__main__':
17     # mode=MODE_LIVE 实时模式
18     # mode=MODE_BACKTEST  回测模式，指定回测开始时间backtest_start_time和结束时间
   backtest_end_time
19
20     run(strategy_id='strategy_id',
21         filename='main.py',
22         mode=MODE_LIVE,
23         token='token_id',
24         backtest_start_time='2017-08-21 9:00:00',
25         backtest_end_time='2017-08-21 15:00:00')
26
```

9.数据研究(典型场景)

无需实时数据驱动策略，无需交易下单，只需提取数据的场景。

```
1 # coding=utf-8
2 from __future__ import print_function, absolute_import
3 from gm.api import *
4
5 # 设置token
6 set_token('xxxx')
7 # 查询历史行情
8 data = history(symbol='SHSE.600000', frequency='1d', start_time='2015-01-01', end_time='2015-
  12-31', fields='open,high,low,close')
9 print(data)
```

10.alpha对冲(股票+期货)

利用股指期货进行对冲的股票策略。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  from gm.api import *
4  '''
5  本策略每隔1个月定时触发计算SHSE.000300成份股的过去一天EV/EBITDA值并选取30只EV/EBITDA值最小且大于零
   的股票
6  对不在股票池的股票平仓并等权配置股票池的标的
7  并用相应的CFFEX.IF对应的真实合约等额对冲
8  回测数据为:SHSE.000300和他们的成份股和CFFEX.IF对应的真实合约
9  回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
10 '''
11
12
13 def init(context):
14     # 每月第一个交易日09:40:00的定时执行algo任务
15     schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
16
```

```python
    # 设置开仓在股票和期货的资金百分比(期货在后面自动进行杠杆相关的调整)
    context.percentage_stock = 0.4
    context.percentage_futures = 0.4


def algo(context):
    # 获取当前时刻
    now = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=now)
    # 获取沪深300成份股
    stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
                                        end_date=last_day)[0]['constituents'].keys()
    # 获取上一个工作日的CFFEX.IF对应的合约
    index_futures = get_continuous_contracts(csymbol='CFFEX.IF', start_date=last_day,
end_date=last_day)[-1]['symbol']
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=stock300, start_date=now,
end_date=now)
    not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
item['is_suspended']]
    # 获取成份股EV/EBITDA大于0并为最小的30个
    fin = get_fundamentals(table='trading_derivative_indicator',
symbols=not_suspended_symbols,
                           start_date=now, end_date=now, fields='EVEBITDA',
                           filter='EVEBITDA>0', order_by='EVEBITDA', limit=30, df=True)
    fin.index = fin.symbol
    # 获取当前仓位
    positions = context.account().positions()
    # 平不在标的池或不为当前股指期货主力合约对应真实合约的标的
    for position in positions:
        symbol = position['symbol']
        sec_type = get_instrumentinfos(symbols=symbol)[0]['sec_type']
        # 若类型为期货且不在标的池则平仓
        if sec_type == SEC_TYPE_FUTURE and symbol != index_futures:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print('市价单平不在标的池的', symbol)
        elif symbol not in fin.index:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

    # 获取股票的权重
    percent = context.percentage_stock / len(fin.index)
    # 买在标的池中的股票
    for symbol in fin.index:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调多仓到仓位', percent)

    # 获取股指期货的保证金比率
```

```
    ratio = get_history_instruments(symbols=index_futures, start_date=last_day,
end_date=last_day)[0]['margin_ratio']
    # 更新股指期货的权重
    percent = context.percentage_futures * ratio
    # 买入股指期货对冲
    order_target_percent(symbol=index_futures, percent=percent, order_type=OrderType_Market,
                         position_side=PositionSide_Short)
    print(index_futures, '以市价单调空仓到仓位', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

11.集合竞价选股(股票)

基于收盘价与前收盘价的选股策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from gm.api import *

'''
本策略通过获取SHSE.000300沪深300的成份股数据并统计其30天内
开盘价大于前收盘价的天数,并在该天数大于阈值10的时候加入股票池
随后对不在股票池的股票平仓并等权配置股票池的标的,每次交易间隔1个月.
回测数据为:SHSE.000300在2015-01-15的成份股
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


```

```python
def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    # context.count_bench累计天数阈值
    context.count_bench = 10
    # 用于对比的天数
    context.count = 30
    # 最大交易资金比例
    context.ratio = 0.8


def algo(context):
    # 获取当前时间
    now = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=now)
    # 获取沪深300成份股
    context.stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
                                                end_date=last_day)[0]['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=context.stock300, start_date=now,
end_date=now)
    not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
item['is_suspended']]

    trade_symbols = []
    if not not_suspended_symbols:
        print('没有当日交易的待选股票')
        return

    for stock in not_suspended_symbols:
        recent_data = history_n(symbol=stock, frequency='1d', count=context.count,
fields='pre_close,open',
                                fill_missing='Last', adjust=ADJUST_PREV, end_time=now,
df=True)
        diff = recent_data['open'] - recent_data['pre_close']
        # 获取累计天数超过阈值的标的池.并剔除当天没有交易的股票
        if len(diff[diff > 0]) >= context.count_bench:
            trade_symbols.append(stock)

    print('本次股票池有股票数目: ', len(trade_symbols))
    # 计算权重
    percent = 1.0 / len(trade_symbols) * context.ratio
    # 获取当前所有仓位
    positions = context.account().positions()
    # 如标的池有仓位,平不在标的池的仓位
    for position in positions:
        symbol = position['symbol']
        if symbol not in trade_symbols:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

```

```
    # 对标的池进行操作
    for symbol in trade_symbols:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调整至权重', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

12.多因子选股(股票)

基于Fama三因子构成的多因子策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
from gm.api import *
from pandas import DataFrame

'''
本策略每隔1个月定时触发,根据Fama-French三因子模型对每只股票进行回归，得到其alpha值。
假设Fama-French三因子模型可以完全解释市场，则alpha为负表明市场低估该股，因此应该买入。
策略思路：
计算市场收益率、个股的账面市值比和市值,并对后两个进行了分类,
根据分类得到的组合分别计算其市值加权收益率、SMB和HML.
对各个股票进行回归(假设无风险收益率等于0)得到alpha值.
选取alpha值小于0并为最小的10只股票进入标的池
平掉不在标的池的股票并等权买入在标的池的股票
回测数据:SHSE.000300的成份股
```

```
回测时间:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    # 数据滑窗
    context.date = 20
    # 设置开仓的最大资金量
    context.ratio = 0.8
    # 账面市值比的大/中/小分类
    context.BM_BIG = 3.0
    context.BM_MID = 2.0
    context.BM_SMA = 1.0
    # 市值大/小分类
    context.MV_BIG = 2.0
    context.MV_SMA = 1.0


# 计算市值加权的收益率,MV为市值的分类,BM为账目市值比的分类
def market_value_weighted(stocks, MV, BM):
    select = stocks[(stocks.NEGOTIABLEMV == MV) & (stocks.BM == BM)]
    market_value = select['mv'].values
    mv_total = np.sum(market_value)
    mv_weighted = [mv / mv_total for mv in market_value]
    stock_return = select['return'].values
    # 返回市值加权的收益率的和
    return_total = []
    for i in range(len(mv_weighted)):
        return_total.append(mv_weighted[i] * stock_return[i])
    return_total = np.sum(return_total)
    return return_total


def algo(context):
    # 获取上一个交易日的日期
    last_day = get_previous_trading_date(exchange='SHSE', date=context.now)
    # 获取沪深300成份股
    context.stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
                                                end_date=last_day)[0]
['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended = get_history_instruments(symbols=context.stock300, start_date=last_day,
end_date=last_day)
    not_suspended = [item['symbol'] for item in not_suspended if not item['is_suspended']]
    fin = get_fundamentals(table='trading_derivative_indicator', symbols=not_suspended,
start_date=last_day, end_date=last_day,
                           fields='PB,NEGOTIABLEMV', df=True)

    # 计算账面市值比,为P/B的倒数
    fin['PB'] = (fin['PB'] ** -1)
    # 计算市值的50%的分位点,用于后面的分类
```

```
    size_gate = fin['NEGOTIABLEMV'].quantile(0.50)
    # 计算账面市值比的30%和70%分位点,用于后面的分类
    bm_gate = [fin['PB'].quantile(0.30), fin['PB'].quantile(0.70)]
    fin.index = fin.symbol
    x_return = []
    # 对未停牌的股票进行处理
    for symbol in not_suspended:
        # 计算收益率
        close = history_n(symbol=symbol, frequency='1d', count=context.date + 1,
end_time=last_day, fields='close',
                          skip_suspended=True, fill_missing='Last', adjust=ADJUST_PREV,
df=True)['close'].values
        stock_return = close[-1] / close[0] - 1
        pb = fin['PB'][symbol]
        market_value = fin['NEGOTIABLEMV'][symbol]
        # 获取[股票代码. 股票收益率, 账面市值比的分类, 市值的分类, 流通市值]
        if pb < bm_gate[0]:
            if market_value < size_gate:
                label = [symbol, stock_return, context.BM_SMA, context.MV_SMA,
market_value]
            else:
                label = [symbol, stock_return, context.BM_SMA, context.MV_BIG,
market_value]
        elif pb < bm_gate[1]:
            if market_value < size_gate:
                label = [symbol, stock_return, context.BM_MID, context.MV_SMA,
market_value]
            else:
                label = [symbol, stock_return, context.BM_MID, context.MV_BIG,
market_value]
        elif market_value < size_gate:
            label = [symbol, stock_return, context.BM_BIG, context.MV_SMA, market_value]
        else:
            label = [symbol, stock_return, context.BM_BIG, context.MV_BIG, market_value]
        if len(x_return) == 0:
            x_return = label
        else:
            x_return = np.vstack([x_return, label])

    stocks = DataFrame(data=x_return, columns=['symbol', 'return', 'BM', 'NEGOTIABLEMV',
'mv'])
    stocks.index = stocks.symbol
    columns = ['return', 'BM', 'NEGOTIABLEMV', 'mv']
    for column in columns:
        stocks[column] = stocks[column].astype(np.float64)
    # 计算SMB.HML和市场收益率
    # 获取小市值组合的市值加权组合收益率
    smb_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
             market_value_weighted(stocks, context.MV_SMA, context.BM_MID) +
             market_value_weighted(stocks, context.MV_SMA, context.BM_BIG)) / 3

    # 获取大市值组合的市值加权组合收益率
    smb_b = (market_value_weighted(stocks, context.MV_BIG, context.BM_SMA) +
```

```
            market_value_weighted(stocks, context.MV_BIG, context.BM_MID) +
            market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 3

    smb = smb_s - smb_b
    # 获取大账面市值比组合的市值加权组合收益率
    hml_b = (market_value_weighted(stocks, context.MV_SMA, context.BM_BIG) +
             market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 2
    # 获取小账面市值比组合的市值加权组合收益率
    hml_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
             market_value_weighted(stocks, context.MV_BIG, context.BM_SMA)) / 2

    hml = hml_b - hml_s
    close = history_n(symbol='SHSE.000300', frequency='1d', count=context.date + 1,
                      end_time=last_day, fields='close', skip_suspended=True,
                      fill_missing='Last', adjust=ADJUST_PREV, df=True)['close'].values
    market_return = close[-1] / close[0] - 1
    coff_pool = []
    # 对每只股票进行回归获取其alpha值
    for stock in stocks.index:
        x_value = np.array([[market_return], [smb], [hml], [1.0]])
        y_value = np.array([stocks['return'][stock]])
        # OLS估计系数
        coff = np.linalg.lstsq(x_value.T, y_value)[0][3]
        coff_pool.append(coff)

    # 获取alpha最小并且小于0的10只的股票进行操作(若少于10只则全部买入)
    stocks['alpha'] = coff_pool
    stocks = stocks[stocks.alpha < 0].sort_values(by='alpha').head(10)

    symbols_pool = stocks.index.tolist()
    positions = context.account().positions()

    # 平不在标的池的股票
    for position in positions:
        symbol = position['symbol']
        if symbol not in symbols_pool:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)

    # 获取股票的权重
    percent = context.ratio / len(symbols_pool)
    # 买在标的池中的股票
    for symbol in symbols_pool:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调多仓到仓位', percent)


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
```

```
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

13.网格交易(期货)

基于网格交易方法的交易策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
import pandas as pd
from gm.api import *

'''
本策略首先计算了SHFE.rb1801过去300个1min收盘价的均值和标准差
并用均值加减2和3个标准差得到网格的区间分界线,分别配以0.3和0.5的仓位权重
然后根据价格所在的区间来配置仓位:
(n+k1*std,n+k2*std],(n+k2*std,n+k3*std],(n+k3*std,n+k4*std],(n+k4*std,n+k5*std],(n+k5*std,n+k6*std]
(n为收盘价的均值,std为收盘价的标准差,k1-k6分别为[-40, -3, -2, 2, 3, 40],其中-40和40为上下界,无实际意义)
[-0.5, -0.3, 0.0, 0.3, 0.5](资金比例,此处负号表示开空仓)
回测数据为:SHFE.rb1801的1min数据
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    context.symbol = 'SHFE.rb1801'
    # 订阅SHFE.rb1801, bar频率为1min
    subscribe(symbols=context.symbol, frequency='60s')
    # 获取过去300个价格数据
    timeseries = history_n(symbol=context.symbol, frequency='60s', count=300, fields='close', fill_missing='Last',
```

```
                               end_time='2017-07-01 08:00:00', df=True)['close'].values
    # 获取网格区间分界线
    context.band = np.mean(timeseries) + np.array([-40, -3, -2, 2, 3, 40]) *
np.std(timeseries)
    # 设置网格的仓位
    context.weight = [0.5, 0.3, 0.0, 0.3, 0.5]


def on_bar(context, bars):
    bar = bars[0]
    # 根据价格落在(-40,-3],(-3,-2],(-2,2],(2,3],(3,40]的区间范围来获取最新收盘价所在的价格区间
    grid = pd.cut([bar.close], context.band, labels=[0, 1, 2, 3, 4])[0]
    # 获取多仓仓位
    position_long = context.account().position(symbol=context.symbol,
side=PositionSide_Long)
    # 获取空仓仓位
    position_short = context.account().position(symbol=context.symbol,
side=PositionSide_Short)
    # 若无仓位且价格突破则按照设置好的区间开仓
    if not position_long and not position_short and grid != 2:
        # 大于3为在中间网格的上方,做多
        if grid >= 3:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单开多仓到仓位', context.weight[grid])
        if grid <= 1:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单开空仓到仓位', context.weight[grid])
    # 持有多仓的处理
    elif position_long:
        if grid >= 3:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单调多仓到仓位', context.weight[grid])
        # 等于2为在中间网格,平仓
        elif grid == 2:
            order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单全平多仓')
        # 小于1为在中间网格的下方,做空
        elif grid <= 1:
            order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单全平多仓')
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
```

```
            print(context.symbol, '以市价单开空仓到仓位', context.weight[grid])
    # 持有空仓的处理
    elif position_short:
        # 小于1为在中间网格的下方,做空
        if grid <= 1:
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单调空仓到仓位', context.weight[grid])
        # 等于2为在中间网格,平仓
        elif grid == 2:
            order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单全平空仓')
        # 大于3为在中间网格的上方,做多
        elif grid >= 3:
            order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
                                 position_side=PositionSide_Short)
            print(context.symbol, '以市价单全平空仓')
            order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单开多仓到仓位', context.weight[grid])


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

14.指数增强(股票)

追踪指数的基础上，以增加超额收益，跑赢对标指数为目的的策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import numpy as np
from gm.api import *
from pandas import DataFrame

'''
本策略以0.8为初始权重跟踪指数标的沪深300中权重大于0.35%的成份股.
个股所占的百分比为(0.8*成份股权重/所选股票占沪深300的总权重)*100%.然后根据个股是否
连续上涨5天;连续下跌5天
来判定个股是否为强势股/弱势股,并对其把权重由0.8调至1.0或0.6
回测数据为:SHSE.000300中权重大于0.35%的成份股
回测时间为:2017-07-01 08:50:00到2017-10-01 17:00:00
'''


def init(context):
    # 资产配置的初始权重,配比为0.6-0.8-1.0
    context.ratio = 0.8
    # 获取沪深300当时的成份股和相关数据
    stock300 = get_history_constituents(index='SHSE.000300', start_date='2017-06-30',
end_date='2017-06-30')[0][
        'constituents']
    stock300_symbol = []
    stock300_weight = []

    for key in stock300:
        # 保留权重大于0.35%的成份股
        if (stock300[key] / 100) > 0.0035:
            stock300_symbol.append(key)
            stock300_weight.append(stock300[key] / 100)

    context.stock300 = DataFrame([stock300_weight], columns=stock300_symbol, index=
['weight']).T
    context.sum_weight = np.sum(stock300_weight)
    print('选择的成分股权重总和为: ', context.sum_weight * 100, '%')
    subscribe(symbols=stock300_symbol, frequency='1d', count=5, wait_group=True)


def on_bar(context, bars):
    # 若没有仓位则按照初始权重开仓
    for bar in bars:
        symbol = bar['symbol']
        position = context.account().position(symbol=symbol, side=PositionSide_Long)
        if not position:
            buy_percent = context.stock300['weight'][symbol] / context.sum_weight *
context.ratio
            order_target_percent(symbol=symbol, percent=buy_percent,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
```

```
            print(symbol, '以市价单开多仓至仓位:', buy_percent * 100, '%')
        else:
            # 获取过去5天的价格数据,若连续上涨则为强势股,权重+0.2;若连续下跌则为弱势股,权重-0.2
            recent_data = context.data(symbol=symbol, frequency='1d', count=5, fields='close')['close'].tolist()
            if all(np.diff(recent_data) > 0):
                buy_percent = context.stock300['weight'][symbol] / context.sum_weight * (context.ratio + 0.2)
                order_target_percent(symbol=symbol, percent=buy_percent, order_type=OrderType_Market,
                                     position_side=PositionSide_Long)
                print('强势股', symbol, '以市价单调多仓至仓位:', buy_percent * 100, '%')
            elif all(np.diff(recent_data) < 0):
                buy_percent = context.stock300['weight'][symbol] / context.sum_weight * (context.ratio - 0.2)
                order_target_percent(symbol=symbol, percent=buy_percent, order_type=OrderType_Market,
                                     position_side=PositionSide_Long)
                print('弱势股', symbol, '以市价单调多仓至仓位:', buy_percent * 100, '%')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:50:00',
        backtest_end_time='2017-10-01 17:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

15.跨品种套利(期货)

期货的跨品种套利策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
```

```python
from gm.api import *
import numpy as np

'''
本策略首先滚动计算过去30个1min收盘价的均值,然后用均值加减2个标准差得到布林线.
若无仓位,在最新价差上穿上轨时做空价差;下穿下轨时做多价差
若有仓位则在最新价差回归至上下轨水平内时平仓
回测数据为:SHFE.rb1801和SHFE.hc1801的1min数据
回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 进行套利的品种
    context.goods = ['SHFE.rb1801', 'SHFE.hc1801']
    # 订阅行情
    subscribe(symbols=context.goods, frequency='60s', count=31, wait_group=True)


def on_bar(context, bars):
    # 获取两个品种的时间序列
    data_rb = context.data(symbol=context.goods[0], frequency='60s', count=31,
fields='close')
    close_rb = data_rb.values
    data_hc = context.data(symbol=context.goods[1], frequency='60s', count=31,
fields='close')
    close_hc = data_hc.values
    # 计算价差
    spread = close_rb[:-1] - close_hc[:-1]
    # 计算布林带的上下轨
    up = np.mean(spread) + 2 * np.std(spread)
    down = np.mean(spread) - 2 * np.std(spread)
    # 计算最新价差
    spread_now = close_rb[-1] - close_hc[-1]
    # 无交易时若价差上(下)穿布林带上(下)轨则做空(多)价差
    position_rb_long = context.account().position(symbol=context.goods[0],
side=PositionSide_Long)
    position_rb_short = context.account().position(symbol=context.goods[0],
side=PositionSide_Short)
    if not position_rb_long and not position_rb_short:
        if spread_now > up:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓一手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓一手')
        if spread_now < down:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
```

```
            print(context.goods[0], '以市价单开多仓一手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓一手')
    # 价差回归时平仓
    elif position_rb_short:
        if spread_now <= up:
            order_close_all()
            print('价格回归,平所有仓位')
            # 跌破下轨反向开仓
        if spread_now < down:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[0], '以市价单开多仓一手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓一手')
    elif position_rb_long:
        if spread_now >= down:
            order_close_all()
            print('价格回归,平所有仓位')
            # 涨破上轨反向开仓
        if spread_now > up:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓一手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓一手')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
```

```
        backtest_start_time='2017-09-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=500000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

16.跨期套利(期货)

期货的跨期套利策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import sys
import numpy as np
from gm.api import *
try:
    import statsmodels.tsa.stattools as ts
except:
    print('请安装statsmodels库')
    sys.exit(-1)

'''
本策略根据EG两步法(1.序列同阶单整2.OLS残差平稳)判断序列具有协整关系后(若无协整关系则全平仓位不进行操作)
通过计算两个价格序列回归残差的均值和标准差并用均值加减0.9倍标准差得到上下轨
在价差突破上轨的时候做空价差;在价差突破下轨的时候做多价差
若有仓位,在残差回归至上下轨内的时候平仓
回测数据为:SHFE.rb1801和SHFE.rb1805的1min数据
回测时间为:2017-09-25 08:00:00到2017-10-01 15:00:00
'''


# 协整检验的函数
def cointegration_test(series01, series02):
    urt_rb1801 = ts.adfuller(np.array(series01), 1)[1]
    urt_rb1805 = ts.adfuller(np.array(series02), 1)[1]
    # 同时平稳或不平稳则差分再次检验
    if (urt_rb1801 > 0.1 and urt_rb1805 > 0.1) or (urt_rb1801 < 0.1 and urt_rb1805 < 0.1):
        urt_diff_rb1801 = ts.adfuller(np.diff(np.array(series01)), 1)[1]
        urt_diff_rb1805 = ts.adfuller(np.diff(np.array(series02)), 1)[1]
        # 同时差分平稳进行OLS回归的残差平稳检验
        if urt_diff_rb1801 < 0.1 and urt_diff_rb1805 < 0.1:
            matrix = np.vstack([series02, np.ones(len(series02))]).T
            beta, c = np.linalg.lstsq(matrix, series01)[0]
            resid = series01 - beta * series02 - c
            if ts.adfuller(np.array(resid), 1)[1] > 0.1:
                result = 0.0
            else:
                result = 1.0
            return beta, c, resid, result
```

```
        else:
            result = 0.0
            return 0.0, 0.0, 0.0, result

    else:
        result = 0.0
        return 0.0, 0.0, 0.0, result


def init(context):
    context.goods = ['SHFE.rb1801', 'SHFE.rb1805']
    # 订阅品种
    subscribe(symbols=context.goods, frequency='60s', count=801, wait_group=True)


def on_bar(context, bars):
    # 获取过去800个60s的收盘价数据
    close_01 = context.data(symbol=context.goods[0], frequency='60s', count=801,
fields='close')['close'].values
    close_02 = context.data(symbol=context.goods[1], frequency='60s', count=801,
fields='close')['close'].values
    # 展示两个价格序列的协整检验的结果
    beta, c, resid, result = cointegration_test(close_01, close_02)
    # 如果返回协整检验不通过的结果则全平仓位等待
    if not result:
        print('协整检验不通过,全平所有仓位')
        order_close_all()
        return

    # 计算残差的标准差上下轨
    mean = np.mean(resid)
    up = mean + 1.5 * np.std(resid)
    down = mean - 1.5 * np.std(resid)
    # 计算新残差
    resid_new = close_01[-1] - beta * close_02[-1] - c
    # 获取rb1801的多空仓位
    position_01_long = context.account().position(symbol=context.goods[0],
side=PositionSide_Long)
    position_01_short = context.account().position(symbol=context.goods[0],
side=PositionSide_Short)
    if not position_01_long and not position_01_short:
        # 上穿上轨时做空新残差
        if resid_new > up:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0] + '以市价单开空仓1手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1] + '以市价单开多仓1手')
        # 下穿下轨时做多新残差
```

```
        if resid_new < down:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[0], '以市价单开多仓1手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓1手')
    # 新残差回归时平仓
    elif position_01_short:
        if resid_new <= up:
            order_close_all()
            print('价格回归,平掉所有仓位')
        # 突破下轨反向开仓
        if resid_new < down:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[0], '以市价单开多仓1手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[1], '以市价单开空仓1手')
    elif position_01_long:
        if resid_new >= down:
            order_close_all()
            print('价格回归,平所有仓位')
        # 突破上轨反向开仓
        if resid_new > up:
            order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Short)
            print(context.goods[0], '以市价单开空仓1手')
            order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
                                position_side=PositionSide_Long)
            print(context.goods[1], '以市价单开多仓1手')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
```

```
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-25 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=500000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

17.日内回转交易(股票)

基于股票日内偏离度回归的日内回转策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import sys
try:
    import talib
except:
    print('请安装TA-Lib库')
    sys.exit(-1)

from gm.api import *

'''
本策略首先买入SHSE.600000股票10000股
随后根据60s的数据计算MACD(12,26,9),
在MACD>0的时候买入100股;在MACD<0的时候卖出100股
但每日操作的股票数不超过原有仓位,并于收盘前把仓位调整至开盘前的仓位
回测数据为:SHSE.600000的60s数据
回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 设置标的股票
    context.symbol = 'SHSE.600000'
    # 用于判定第一个仓位是否成功开仓
    context.first = 0
    # 订阅浦发银行, bar频率为1min
    subscribe(symbols=context.symbol, frequency='60s', count=35)
    # 日内回转每次交易100股
    context.trade_n = 100
    # 获取昨今天的时间
    context.day = [0, 0]
    # 用于判断是否触发了回转逻辑的计时
    context.ending = 0


```

```
def on_bar(context, bars):
    bar = bars[0]
    if context.first == 0:
        # 最开始配置仓位
        # 需要保持的总仓位
        context.total = 10000
        # 购买10000股浦发银行股票
        order_volume(symbol=context.symbol, volume=context.total, side=PositionSide_Long,
                     order_type=OrderType_Market, position_effect=PositionEffect_Open)
        print(context.symbol, '以市价单开多仓10000股')
        context.first = 1.
        day = bar.bob.strftime('%Y-%m-%d')
        context.day[-1] = day[-2:]
        # 每天的仓位操作
        context.turnaround = [0, 0]
        return

    # 更新最新的日期
    day = bar.bob.strftime('%Y-%m-%d %H:%M:%S')
    context.day[0] = bar.bob.day
    # 若为新的一天,获取可用于回转的昨仓
    if context.day[0] != context.day[-1]:
        context.ending = 0
        context.turnaround = [0, 0]
    if context.ending == 1:
        return

    # 若有可用的昨仓则操作
    if context.total >= 0:
        # 获取时间序列数据
        symbol = bar['symbol']
        recent_data = context.data(symbol=symbol, frequency='60s', count=35,
fields='close')
        # 计算MACD线
        macd = talib.MACD(recent_data['close'].values)[0][-1]
        # 根据MACD>0则开仓,小于0则平仓
        if macd > 0:
            # 多空单向操作都不能超过昨仓位,否则最后无法调回原仓位
            if context.turnaround[0] + context.trade_n < context.total:
                # 计算累计仓位
                context.turnaround[0] += context.trade_n
                order_volume(symbol=context.symbol, volume=context.trade_n,
side=PositionSide_Long,
                             order_type=OrderType_Market,
position_effect=PositionEffect_Open)
                print(symbol, '市价单开多仓', context.trade_n, '股')
        elif macd < 0:
            if context.turnaround[1] + context.trade_n < context.total:
                context.turnaround[1] += context.trade_n
                order_volume(symbol=context.symbol, volume=context.trade_n,
side=PositionSide_Short,
                             order_type=OrderType_Market,
position_effect=PositionEffect_Close)
```

```
                print(symbol, '市价单平多仓', context.trade_n, '股')
        # 临近收盘时若仓位数不等于昨仓则回转所有仓位
        if day[11:16] == '14:55' or day[11:16] == '14:57':
            position = context.account().position(symbol=context.symbol,
side=PositionSide_Long)
            if position['volume'] != context.total:
                order_target_volume(symbol=context.symbol, volume=context.total,
order_type=OrderType_Market,
                                    position_side=PositionSide_Long)
                print('市价单回转仓位操作...')
                context.ending = 1
        # 更新过去的日期数据
        context.day[-1] = context.day[0]


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=2000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

18.做市商策略(期货)

基于Tick价差的交易策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from gm.api import *

'''
本策略通过不断对CZCE.CF801进行
买一价现价单开多仓和卖一价平多仓;
卖一价现价单开空仓和买一价平空仓来做市
```

```
并以此赚取差价
回测数据为:CZCE.CF801的tick数据
回测时间为:2017-09-29 11:25:00到2017-09-29 11:30:00
需要特别注意的是:本平台对于回测对限价单固定完全成交,本例子 仅供参考.
敬请通过适当调整回测参数
1.backtest_commission_ratio回测佣金比例
2.backtest_slippage_ratio回测滑点比例
3.backtest_transaction_ratio回测成交比例
以及优化策略逻辑来达到更贴近实际的回测效果
'''


def init(context):
    # 订阅CZCE.CF801的tick数据
    context.symbol = 'CZCE.CF801'
    subscribe(symbols=context.symbol, frequency='tick')


def on_tick(context, tick):
    quotes = tick['quotes'][0]
    # 获取持有的多仓
    positio_long = context.account().position(symbol=context.symbol, side=PositionSide_Long)
    # 获取持有的空仓
    position_short = context.account().position(symbol=context.symbol,
side=PositionSide_Short)
    print(quotes['bid_p'])
    print(quotes['ask_p'])
    # 没有仓位则双向开限价单
    # 若有仓位则限价单平仓
    if not positio_long:
        # 获取买一价
        price = quotes['bid_p']
        print('买一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=1, price=price,
order_type=OrderType_Limit,
                            position_side=PositionSide_Long)
        print('CZCE.CF801开限价单多仓1手')
    else:
        # 获取卖一价
        price = quotes['ask_p']
        print('卖一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=0, price=price,
order_type=OrderType_Limit,
                            position_side=PositionSide_Long)
        print('CZCE.CF801平限价单多仓1手')
    if not position_short:
        # 获取卖一价
        price = quotes['ask_p']
        print('卖一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=1, price=price,
order_type=OrderType_Limit,
                            position_side=PositionSide_Short)
        print('CZCE.CF801卖一价开限价单空仓')
```

```
    else:
        # 获取买一价
        price = quotes['bid_p']
        print('买一价为: ', price)
        order_target_volume(symbol=context.symbol, volume=0, price=price,
order_type=OrderType_Limit,
                            position_side=PositionSide_Short)
        print('CZCE.CF801买一价平限价单空仓')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    backtest_transaction_ratio回测成交比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-29 11:25:00',
        backtest_end_time='2017-09-29 11:30:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=500000,
        backtest_commission_ratio=0.00006,
        backtest_slippage_ratio=0.0001,
        backtest_transaction_ratio=0.5)

```

19.海龟交易法(期货)

基于海龟交易法则的交易策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals

import sys

import numpy as np
import pandas as pd

try:
    import talib
except:
```

```
    print('请安装TA-Lib库')
    sys.exit(-1)
from gm.api import *

'''
本策略通过计算CZCE.FG801和SHFE.rb1801的ATR.唐奇安通道和MA线,
当价格上穿唐奇安通道且短MA在长MA上方时开多仓;当价格下穿唐奇安通道且短MA在长MA下方时开空仓(8手)
若有多仓则在价格跌破唐奇安平仓通道下轨的时候全平仓位,否则根据跌破
持仓均价 - x(x=0.5,1,1.5,2)倍ATR把仓位平至6/4/2/0手
若有空仓则在价格涨破唐奇安平仓通道上轨的时候全平仓位,否则根据涨破
持仓均价 + x(x=0.5,1,1.5,2)倍ATR把仓位平至6/4/2/0手
回测数据为:CZCE.FG801和SHFE.rb1801的1min数据
回测时间为:2017-09-15 09:15:00到2017-10-01 15:00:00
'''


def init(context):
    # context.parameter分别为唐奇安开仓通道.唐奇安平仓通道.短ma.长ma.ATR的参数
    context.parameter = [55, 20, 10, 60, 20]
    context.tar = context.parameter[4]
    # context.goods交易的品种
    context.goods = ['CZCE.FG801', 'SHFE.rb1801']
    # 订阅context.goods里面的品种, bar频率为1min
    subscribe(symbols=context.goods, frequency='60s', count=101)
    # 止损的比例区间


def on_bar(context, bars):
    bar = bars[0]
    symbol = bar['symbol']
    recent_data = context.data(symbol=symbol, frequency='60s', count=101,
fields='close,high,low')
    close = recent_data['close'].values[-1]
    # 计算ATR
    atr = talib.ATR(recent_data['high'].values, recent_data['low'].values,
recent_data['close'].values,
                    timeperiod=context.tar)[-1]
    # 计算唐奇安开仓和平仓通道
    context.don_open = context.parameter[0] + 1
    upper_band = talib.MAX(recent_data['close'].values[:-1], timeperiod=context.don_open)
[-1]
    context.don_close = context.parameter[1] + 1
    lower_band = talib.MIN(recent_data['close'].values[:-1], timeperiod=context.don_close)
[-1]
    # 若没有仓位则开仓
    position_long = context.account().position(symbol=symbol, side=PositionSide_Long)

    position_short = context.account().position(symbol=symbol, side=PositionSide_Short)
    if not position_long and not position_short:
        # 计算长短ma线.DIF
        ma_short = talib.MA(recent_data['close'].values, timeperiod=(context.parameter[2] +
1))[-1]
```

```
        ma_long = talib.MA(recent_data['close'].values, timeperiod=(context.parameter[3] + 1))[-1]
        dif = ma_short - ma_long
        # 获取当前价格
        # 上穿唐奇安通道且短ma在长ma上方则开多仓
        if close > upper_band and (dif > 0):
            order_target_volume(symbol=symbol, volume=8, position_side=PositionSide_Long, order_type=OrderType_Market)
            print(symbol, '市价单开多仓8手')
        # 下穿唐奇安通道且短ma在长ma下方则开空仓
        if close < lower_band and (dif < 0):
            order_target_volume(symbol=symbol, volume=8, position_side=PositionSide_Short, order_type=OrderType_Market)
            print(symbol, '市价单开空仓8手')
    elif position_long:
        # 价格跌破唐奇安平仓通道全平仓位止损
        if close < lower_band:
            order_close_all()
            print(symbol, '市价单全平仓位')
        else:
            # 获取持仓均价
            vwap = position_long['vwap']
            # 获取持仓的资金
            band = vwap - np.array([200, 2, 1.5, 1, 0.5, -100]) * atr
            # 计算最新应持仓位
            grid_volume = int(pd.cut([close], band, labels=[0, 1, 2, 3, 4])[0]) * 2
            order_target_volume(symbol=symbol, volume=grid_volume, position_side=PositionSide_Long,
                                order_type=OrderType_Market)
            print(symbol, '市价单平多仓到', grid_volume, '手')
    elif position_short:
        # 价格涨破唐奇安平仓通道或价格涨破持仓均价加两倍ATR平空仓
        if close > upper_band:
            order_close_all()
            print(symbol, '市价单全平仓位')
        else:
            # 获取持仓均价
            vwap = position_short['vwap']
            # 获取平仓的区间
            band = vwap + np.array([-100, 0.5, 1, 1.5, 2, 200]) * atr
            # 计算最新应持仓位
            grid_volume = int(pd.cut([close], band, labels=[0, 1, 2, 3, 4])[0]) * 2
            order_target_volume(symbol=symbol, volume=grid_volume, position_side=PositionSide_Short,
                                order_type=OrderType_Market)
            print(symbol, '市价单平空仓到', grid_volume, '手')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
```

```
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-09-15 09:15:00',
        backtest_end_time='2017-10-01 15:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

20.行业轮动(股票)

基于沪深300的行业指数的行业轮动策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
import numpy as np
from gm.api import *

'''
本策略每隔1个月定时触发计算
SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.000914
(300工业.300材料.300可选.300消费.300医药.300金融)这几个行业指数过去
20个交易日的收益率,随后选取了收益率最高的指数的成份股中流通市值最大的5只股票
对不在股票池的股票平仓并等权配置股票池的标的
回测数据为:SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.000914和他们的成份股
回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
'''


def init(context):
    # 每月第一个交易日的09:40 定时执行algo任务
    schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
    # 用于筛选的行业指数
    context.index = ['SHSE.000910', 'SHSE.000909', 'SHSE.000911', 'SHSE.000912',
'SHSE.000913', 'SHSE.000914']
    # 用于统计数据的天数
    context.count = 20
    # 最大下单资金比例
    context.ratio = 0.8

```

```python

def algo(context):
    # 获取当天的日期
    today = context.now
    # 获取上一个交易日
    last_day = get_previous_trading_date(exchange='SHSE', date=today)
    return_index = []
    # 获取并计算行业指数收益率

    for i in context.index:
        return_index_his = history_n(symbol=i, frequency='1d', count=context.count,
fields='close,bob',
                                     fill_missing='Last', adjust=ADJUST_PREV,
end_time=last_day, df=True)
        return_index_his = return_index_his['close'].values
        return_index.append(return_index_his[-1] / return_index_his[0] - 1)
    # 获取指定数内收益率表现最好的行业
    sector = context.index[np.argmax(return_index)]
    print('最佳行业指数是: ', sector)
    # 获取最佳行业指数成份股
    symbols = get_history_constituents(index=sector, start_date=last_day, end_date=last_day)
[0]['constituents'].keys()
    # 获取当天有交易的股票
    not_suspended_info = get_history_instruments(symbols=symbols, start_date=today,
end_date=today)
    not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
item['is_suspended']]

    # 获取最佳行业指数成份股的市值，从大到小排序并选取市值最大的5只股票
    fin = get_fundamentals(table='trading_derivative_indicator',
symbols=not_suspended_symbols, start_date=last_day,
                           end_date=last_day, limit=5, fields='NEGOTIABLEMV', order_by='-
NEGOTIABLEMV', df=True)
    fin.index = fin['symbol']
    # 计算权重
    percent = 1.0 / len(fin.index) * context.ratio
    # 获取当前所有仓位
    positions = context.account().positions()
    # 如标的池有仓位,平不在标的池的仓位
    for position in positions:
        symbol = position['symbol']
        if symbol not in fin.index:
            order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print('市价单平不在标的池的', symbol)
    # 对标的池进行操作
    for symbol in fin.index:
        order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
                             position_side=PositionSide_Long)
        print(symbol, '以市价单调整至仓位', percent)


if __name__ == '__main__':
```

```
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-07-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

21.机器学习(股票)

基于机器学习算法支持向量机SVM的交易策略。

```
# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
from datetime import datetime
import numpy as np
from gm.api import *
import sys
try:
    from sklearn import svm
except:
    print('请安装scikit-learn库和带mkl的numpy')
    sys.exit(-1)

'''
本策略选取了七个特征变量组成了滑动窗口长度为15天的训练集,随后训练了一个二分类(上涨/下跌)的支持向量机模型.
若没有仓位则在每个星期一的时候输入标的股票近15个交易日的特征变量进行预测,并在预测结果为上涨的时候购买标的.
若已经持有仓位则在盈利大于10%的时候止盈,在星期五损失大于2%的时候止损.
特征变量为:1.收盘价/均值2.现量/均量3.最高价/均价4.最低价/均价5.现量6.区间收益率7.区间标准差
训练数据为:SHSE.600009上海机场,时间从2016-04-01到2017-07-30
回测时间为:2017-08-01 09:00:00到2017-09-05 09:00:00
'''


```

```python
def init(context):
    # 订阅上海机场的分钟bar行情
    context.symbol = 'SHSE.600009'
    subscribe(symbols=context.symbol, frequency='60s')
    start_date = '2016-03-01'  # SVM训练起始时间
    end_date = '2017-06-30'  # SVM训练终止时间
    # 用于记录工作日
    # 获取目标股票的daily历史行情
    recent_data = history(symbol=context.symbol, frequency='1d', start_time=start_date, end_time=end_date, fill_missing='Last',
                          df=True)
    days_value = recent_data['bob'].values
    days_close = recent_data['close'].values
    days = []
    # 获取行情日期列表
    print('准备数据训练SVM')
    for i in range(len(days_value)):
        days.append(str(days_value[i])[0:10])

    x_all = []
    y_all = []
    for index in range(15, (len(days) - 5)):
        # 计算三星期共15个交易日相关数据
        start_day = days[index - 15]
        end_day = days[index]
        data = history(symbol=context.symbol, frequency='1d', start_time=start_day, end_time=end_day, fill_missing='Last',
                       df=True)
        close = data['close'].values
        max_x = data['high'].values
        min_n = data['low'].values
        amount = data['amount'].values
        volume = []
        for i in range(len(close)):
            volume_temp = amount[i] / close[i]
            volume.append(volume_temp)

        close_mean = close[-1] / np.mean(close)  # 收盘价/均值
        volume_mean = volume[-1] / np.mean(volume)  # 现量/均量
        max_mean = max_x[-1] / np.mean(max_x)  # 最高价/均价
        min_mean = min_n[-1] / np.mean(min_n)  # 最低价/均价
        vol = volume[-1]  # 现量
        return_now = close[-1] / close[0]  # 区间收益率
        std = np.std(np.array(close), axis=0)  # 区间标准差

        # 将计算出的指标添加到训练集X
        # features用于存放因子
        features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now, std]
        x_all.append(features)

    # 准备算法需要用到的数据
    for i in range(len(days_close) - 20):
        if days_close[i + 20] > days_close[i + 15]:
```

```
            label = 1
        else:
            label = 0
        y_all.append(label)

    x_train = x_all[: -1]
    y_train = y_all[: -1]
    # 训练SVM
    context.clf = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False,
                          tol=0.001, cache_size=400, verbose=False, max_iter=-1,
                          decision_function_shape='ovr', random_state=None)
    context.clf.fit(x_train, y_train)
    print('训练完成!')


def on_bar(context, bars):
    bar = bars[0]
    # 获取当前年月日
    today = bar.bob.strftime('%Y-%m-%d')
    last_day = get_previous_trading_date(exchange='SHSE', date=today)
    # 获取数据并计算相应的因子
    # 于星期一的09:31:00进行操作
    # 当前bar的工作日
    weekday = datetime.strptime(today, '%Y-%m-%d').isoweekday()
    # 获取模型相关的数据
    # 获取持仓
    position = context.account().position(symbol=context.symbol, side=PositionSide_Long)
    # 如果bar是新的星期一且没有仓位则开始预测
    if not position and weekday == 1:
        # 获取预测用的历史数据
        data = history_n(symbol=context.symbol, frequency='1d', end_time=last_day, count=15,
                         fill_missing='Last', adjust=ADJUST_PREV, df=True)
        close = data['close'].values
        train_max_x = data['high'].values
        train_min_n = data['low'].values
        train_amount = data['amount'].values
        volume = []
        for i in range(len(close)):
            volume_temp = train_amount[i] / close[i]
            volume.append(volume_temp)

        close_mean = close[-1] / np.mean(close)
        volume_mean = volume[-1] / np.mean(volume)
        max_mean = train_max_x[-1] / np.mean(train_max_x)
        min_mean = train_min_n[-1] / np.mean(train_min_n)
        vol = volume[-1]
        return_now = close[-1] / close[0]
        std = np.std(np.array(close), axis=0)

        # 得到本次输入模型的因子
        features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now, std]
```

```
        features = np.array(features).reshape(1, -1)
        prediction = context.clf.predict(features)[0]
        # 若预测值为上涨则开仓
        if prediction == 1:
            # 获取昨收盘价
            context.price = close[-1]
            # 把浦发银行的仓位调至95%
            order_target_percent(symbol=context.symbol, percent=0.95,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
            print(context.symbol, '以市价单开多仓到仓位0.95')
    # 当涨幅大于10%,平掉所有仓位止盈
    elif position and bar.close / context.price >= 1.10:
        order_close_all()
        print(context.symbol, '以市价单全平多仓止盈')
    # 当时间为周五并且跌幅大于2%时,平掉所有仓位止损
    elif position and bar.close / context.price < 1.02 and weekday == 5:
        order_close_all()
        print(context.symbol, '以市价单全平多仓止损')


if __name__ == '__main__':
    '''
    strategy_id策略ID,由系统生成
    filename文件名,请与本文件名保持一致
    mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
    token绑定计算机的ID,可在系统设置-密钥管理中生成
    backtest_start_time回测开始时间
    backtest_end_time回测结束时间
    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
    backtest_initial_cash回测初始资金
    backtest_commission_ratio回测佣金比例
    backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-08-01 09:00:00',
        backtest_end_time='2017-09-05 09:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=10000000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)

```

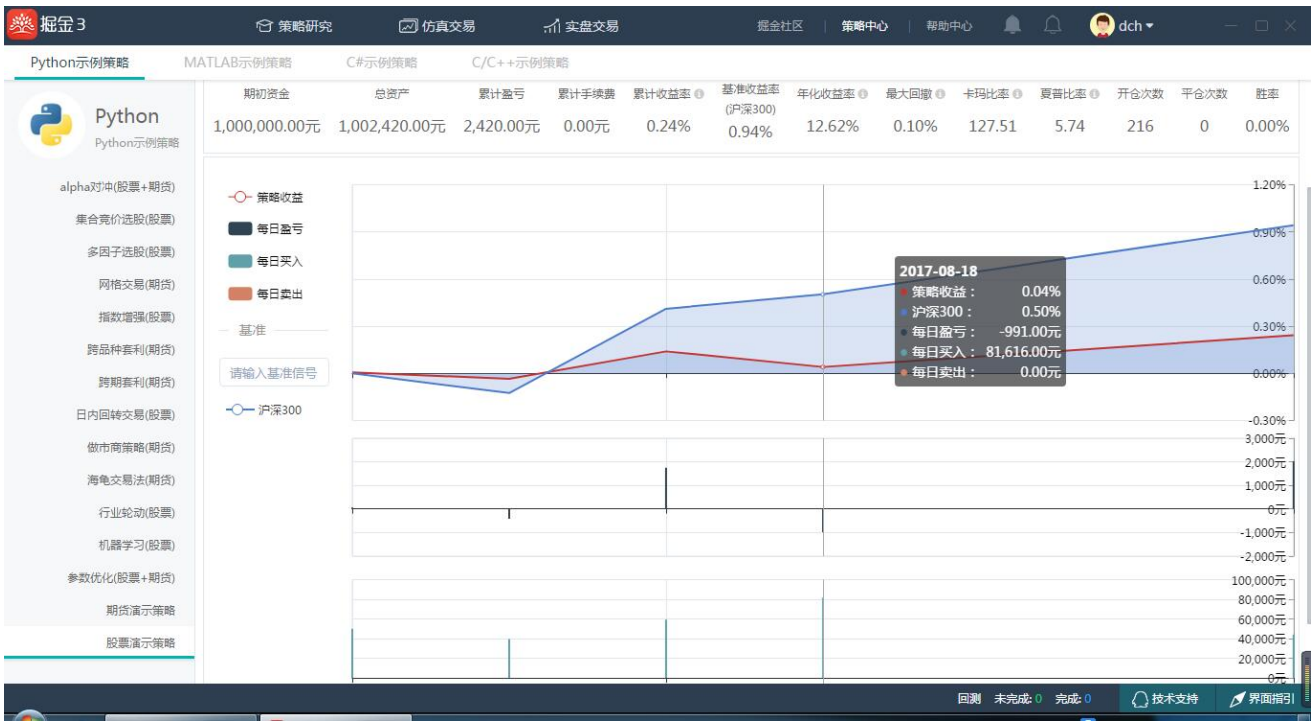
22.参数优化(股票+期货)

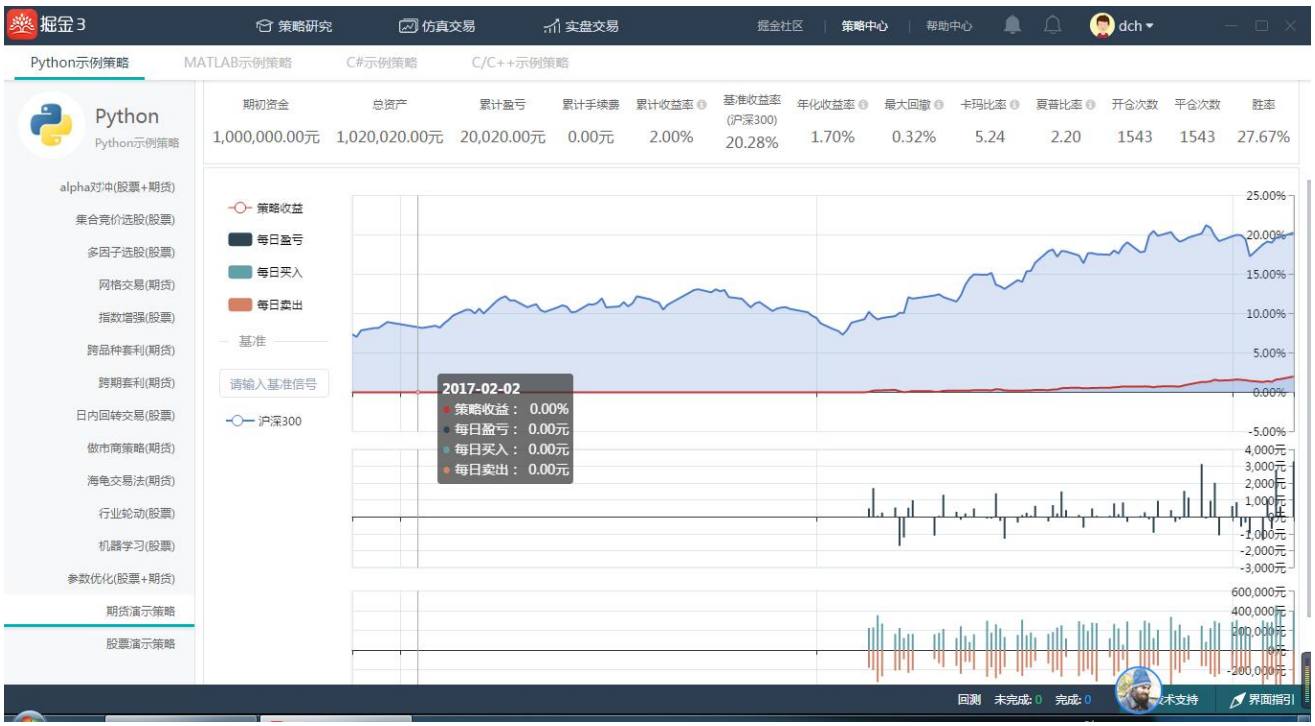
基于循环遍历回测的参数优化方法。

```
•# coding=utf-8
from __future__ import print_function, absolute_import, unicode_literals
```

```python

import multiprocessing

import numpy as np
import pandas as pd
import talib
from gm.api import *

'''
基本思想：设定所需优化的参数数值范围及步长，将参数数值循环输入进策略，进行遍历回测，
        记录每次回测结果和参数，根据某种规则将回测结果排序，找到最好的参数。
1、定义策略函数
2、多进程循环输入参数数值
3、获取回测报告，生成DataFrame格式
4、排序
本程序以双均线策略为例，优化两均线长短周期参数。
'''


# 原策略中的参数定义语句需要删除！
def init(context):
    context.sec_id = 'SHSE.600000'
    subscribe(symbols=context.sec_id, frequency='1d', count=31, wait_group=True)


def on_bar(context, bars):
    close = context.data(symbol=context.sec_id, frequency='1d', count=31, fields='close')
['close'].values
    MA_short = talib.MA(close, timeperiod=context.short)
    MA_long = talib.MA(close, timeperiod=context.long)
    position = context.account().position(symbol=context.sec_id, side=PositionSide_Long)
    if not position and not position:
        if MA_short[-1] > MA_long[-1] and MA_short[-2] < MA_long[-2]:
            order_target_percent(symbol=context.sec_id, percent=0.8,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)
    elif position:
        if MA_short[-1] < MA_long[-1] and MA_short[-2] > MA_long[-2]:
            order_target_percent(symbol=context.sec_id, percent=0,
order_type=OrderType_Market,
                                 position_side=PositionSide_Long)


# 获取每次回测的报告数据
def on_backtest_finished(context, indicator):
    data = [indicator['pnl_ratio'], indicator['pnl_ratio_annual'], indicator['sharp_ratio'],
indicator['max_drawdown'],
            context.short, context.long]
    # 将回测报告加入全局list，以便记录
    context.list.append(data)


def run_strategy(short, long, a_list):
```

```
    from gm.model.storage import context
    # 用context传入参数
    context.short = short
    context.long = long
    # a_list一定要传入
    context.list = a_list
    '''
        strategy_id策略ID,由系统生成
        filename文件名,请与本文件名保持一致
        mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
        token绑定计算机的ID,可在系统设置-密钥管理中生成
        backtest_start_time回测开始时间
        backtest_end_time回测结束时间
        backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
        backtest_initial_cash回测初始资金
        backtest_commission_ratio回测佣金比例
        backtest_slippage_ratio回测滑点比例
    '''
    run(strategy_id='strategy_id',
        filename='main.py',
        mode=MODE_BACKTEST,
        token='token_id',
        backtest_start_time='2017-05-01 08:00:00',
        backtest_end_time='2017-10-01 16:00:00',
        backtest_adjust=ADJUST_PREV,
        backtest_initial_cash=50000,
        backtest_commission_ratio=0.0001,
        backtest_slippage_ratio=0.0001)


if __name__ == '__main__':
    # 生成全局list
    manager = multiprocessing.Manager()
    a_list = manager.list()
    # 循环输入参数数值回测
    for short in range(5, 10, 2):
        for long in range(10, 21, 5):
            process = multiprocessing.Process(target=run_strategy, args=(short, long,
a_list))
            process.start()
            process.join()
    # 回测报告转化成DataFrame格式
    a_list = np.array(a_list)
    final = pd.DataFrame(a_list,
                         columns=['pnl_ratio', 'pnl_ratio_annual', 'sharp_ratio',
'max_drawdown', 'short', 'long'])
    # 回测报告排序
    final = final.sort_values(axis=0, ascending=False, by='pnl_ratio')
    print(final)

```


掘金3
策略研究
仿真交易
实盘交易
掘金社区
策略中心
帮助中心
dch
Python示例策略
MATLAB示例策略
C#示例策略
C/C++示例策略
Python
Python示例策略
alpha对冲(股票+期货)
集合竞价选股(股票)
多因子选股(股票)
网格交易(期货)
指数增强(股票)
跨品种套利(期货)
跨期套利(期货)
日内回转交易(股票)
做市商策略(期货)
海龟交易法(期货)
行业轮动(股票)
机器学习(股票)
参数优化(股票+期货)
期货演示策略
股票演示策略
期初资金
总资产
累计盈亏
累计手续费
累计收益率
基准收益率(沪深300)
年化收益率
最大回撤
卡玛比率
夏普比率
开仓次数
平仓次数
胜率
1,000,000.00元
1,002,420.00元
2,420.00元
0.00元
0.24%
0.94%
12.62%
0.10%
127.51
5.74
216
0
0.00%
策略收益
每日盈亏
每日买入
每日卖出
基准
请输入基准信号
沪深300
2017-08-18
策略收益： 0.04%
沪深300： 0.50%
每日盈亏： -991.00元
每日买入： 81,616.00元
每日卖出： 0.00元
回测 未完成: 0 完成: 0
技术支持
界面指引


掘金3
策略研究
仿真交易
实盘交易
掘金社区
策略中心
帮助中心
dch
Python示例策略
MATLAB示例策略
C#示例策略
C/C++示例策略
Python
Python示例策略
alpha对冲(股票+期货)
集合竞价选股(股票)
多因子选股(股票)
网格交易(期货)
指数增强(股票)
跨品种套利(期货)
跨期套利(期货)
日内回转交易(股票)
做市商策略(期货)
海龟交易法(期货)
行业轮动(股票)
机器学习(股票)
参数优化(股票+期货)
期货演示策略
股票演示策略
期初资金
总资产
累计盈亏
累计手续费
累计收益率
基准收益率(沪深300)
年化收益率
最大回撤
卡玛比率
夏普比率
开仓次数
平仓次数
胜率
1,000,000.00元
1,020,020.00元
20,020.00元
0.00元
2.00%
20.28%
1.70%
0.32%
5.24
2.20
1543
1543
27.67%
策略收益
每日盈亏
每日买入
每日卖出
基准
请输入基准信号
沪深300
2017-02-02
策略收益： 0.00%
每日盈亏： 0.00元
每日买入： 0.00元
每日卖出： 0.00元
回测 未完成: 0 完成: 0
界面指引