

MyQuant新建Python策略

MyQuant新建Python策略

- 1.空策略
- 2.定时任务(典型场景)
- 3.数据事件驱动(典型场景)
- 4.时间序列数据事件驱动(典型场景)
- 5.多个代码数据事件驱动(典型场景)
- 6.默认账户交易(典型场景)
- 7.显式指定交易账户(典型场景)
- 8.模式选择(典型场景)
- 9.数据研究(典型场景)
- 10.alpha对冲(股票+期货)
- 11.集合竞价选股(股票)
- 12.多因子选股(股票)
- 13.网格交易(期货)
- 14.指数增强(股票)
- 15.跨品种套利(期货)
- 16.跨期套利(期货)
- 17.日内回转交易(股票)
- 18.做市商策略(期货)
- 19.海龟交易法(期货)
- 20.行业轮动(股票)
- 21.机器学习(股票)
- 22.参数优化(股票+期货)

1.空策略

不使用模板。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5  # 策略中必须有init方法
6  def init(context):
7      pass
8
9  if __name__ == '__main__':
10     run(strategy_id='strategy_id',
11         filename='main.py',
12         mode=MODE_BACKTEST,
13         token='token_id',
14         backtest_start_time='2016-06-17 13:00:00',
15         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。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3
4  from gm.api import *
5
6
7  def init(context):
8      # 每天14:50 定时执行algo任务
9      schedule(schedule_func=algo, date_rule='daily', time_rule='14:50:00')
10
11
12  def algo(context):
13      # 购买200股浦发银行股票
14      order_volume(symbol='SHSE.600000', volume=200, side=1,
15                  order_type=2, position_effect=1, price=0)
16
17
18  # 查看最终的回测结果
19  def on_backtest_finished(context, indicator):
20      print(indicator)
21
22
23  if __name__ == '__main__':
24      run(strategy_id='strategy_id',
25          filename='main.py',
26          mode=MODE_BACKTEST,
27          token='token_id',
28          backtest_start_time='2016-06-17 13:00:00',
29          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。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      # 订阅浦发银行，bar频率为一天
8      subscribe(symbols='SHSE.600000', frequency='1d')
9
10
11  def on_bar(context, bars):
12      # 打印当前获取的bar信息
13      print(bars)
14
```

```

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

```

```

8     subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d', count=5,
9               wait_group=True)
10
11
12 def on_bar(context, bars):
13     for bar in bars:
14         print(bar['symbol'], bar['eob'])
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

```

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。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d')
8
9
10 def on_bar(context, bars):
11     for bar in bars:
12         # 不指定account 使用默认账户下单
13         order_volume(symbol=bar['symbol'], volume=200, side=1,
14                     order_type=2, position_effect=1, price=0)
15
16
17 # 查看最终的回测结果
18 def on_backtest_finished(context, indicator):
19     print(indicator)
20
21
22 if __name__ == '__main__':
23     run(strategy_id='strategy_id',
24         filename='main.py',
25         mode=MODE_BACKTEST,
26         token='token_id',
27         backtest_start_time='2016-06-17 13:00:00',
28         backtest_end_time='2017-08-21 15:00:00')
29

```

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。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      subscribe(symbols='SHSE.600000,SZSE.000001', frequency='1d')
8
9
10 def on_bar(context, bars):
11     for bar in bars:
12         # account等于账户id 或者账户标题 指定交易账户
13         order_volume(symbol=bar['symbol'], volume=200, price=0, side=1,
14                       order_type=2, position_effect=1, account='xxxxx')
15
16
17 # 查看最终的回测结果
18 def on_backtest_finished(context, indicator):
19     print(indicator)
20
21
22 if __name__ == '__main__':
23     run(strategy_id='strategy_id',
24         filename='main.py',
25         mode=MODE_BACKTEST,
26         token='token_id',
27         backtest_start_time='2016-06-17 13:00:00',
28         backtest_end_time='2017-08-21 15:00:00')
29
```

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。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import
3  from gm.api import *
4
5
6  def init(context):
7      # 订阅浦发银行的tick
8      subscribe(symbols='SHSE.600000', frequency='tick')
9
10
11 def on_tick(context, tick):
12     # 打印当前获取的tick信息
13     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 '''
6 本策略每隔1个月定时触发计算SHSE.000300成份股在过去一天EV/EBITDA值并选取30只EV/EBITDA值最小且大于零
    的股票
7 对不在股票池的股票平仓并等权配置股票池的标的
8 并用相应的CFFEX.IF对应的真实合约等额对冲
9 回测数据为:SHSE.000300和他们的成份股和CFFEX.IF对应的真实合约
10 回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
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

```

```

17 # 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)
18 context.percentage_stock = 0.4
19 context.percentage_futures = 0.4
20
21
22 def algo(context):
23     # 获取当前时刻
24     now = context.now
25     # 获取上一个交易日
26     last_day = get_previous_trading_date(exchange='SHSE', date=now)
27     # 获取沪深300成份股
28     stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
29                                         end_date=last_day)[0]['constituents'].keys()
30     # 获取上一个工作日的CFFEX.IF对应的合约
31     index_futures = get_continuous_contracts(csymbol='CFFEX.IF', start_date=last_day,
32 end_date=last_day)[-1]['symbol']
33     # 获取当天有交易的股票
34     not_suspended_info = get_history_instruments(symbols=stock300, start_date=now,
35 end_date=now)
36     not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
37 item['is_suspended']]
38     # 获取成份股EV/EBITDA大于0并为最小的30个
39     fin = get_fundamentals(table='trading_derivative_indicator',
40 symbols=not_suspended_symbols,
41 start_date=now, end_date=now, fields='EVEBITDA',
42 filter='EVEBITDA>0', order_by='EVEBITDA', limit=30, df=True)
43     fin.index = fin.symbol
44     # 获取当前仓位
45     positions = context.account().positions()
46     # 平不在标的池或不作为当前股指期货主力合约对应真实合约的标的
47     for position in positions:
48         symbol = position['symbol']
49         sec_type = get_instrumentinfos(symbols=symbol)[0]['sec_type']
50         # 若类型为期货且不在标的池则平仓
51         if sec_type == SEC_TYPE_FUTURE and symbol != index_futures:
52             order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
53 position_side=PositionSide_Short)
54             print('市价单平不在标的池的', symbol)
55         elif symbol not in fin.index:
56             order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
57 position_side=PositionSide_Long)
58             print('市价单平不在标的池的', symbol)
59
60     # 获取股票的权重
61     percent = context.percentage_stock / len(fin.index)
62     # 买在标的池中的股票
63     for symbol in fin.index:
64         order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
65 position_side=PositionSide_Long)
66         print(symbol, '以市价单调多仓到仓位', percent)
67
68     # 获取股指期货的保证金比率

```

```

65     ratio = get_history_instruments(symbols=index_futures, start_date=last_day,
end_date=last_day)[0]['margin_ratio']
66     # 更新股指期货的权重
67     percent = context.percentage_futures * ratio
68     # 买入股指期货对冲
69     order_target_percent(symbol=index_futures, percent=percent, order_type=OrderType_Market,
70                           position_side=PositionSide_Short)
71     print(index_futures, '以市价单调空仓到仓位', percent)
72
73
74 if __name__ == '__main__':
75     '''
76     strategy_id策略ID,由系统生成
77     filename文件名,请与本文件名保持一致
78     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
79     token绑定计算机的ID,可在系统设置-密钥管理中生成
80     backtest_start_time回测开始时间
81     backtest_end_time回测结束时间
82     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
83     backtest_initial_cash回测初始资金
84     backtest_commission_ratio回测佣金比例
85     backtest_slippage_ratio回测滑点比例
86     '''
87     run(strategy_id='strategy_id',
88         filename='main.py',
89         mode=MODE_BACKTEST,
90         token='token_id',
91         backtest_start_time='2017-07-01 08:00:00',
92         backtest_end_time='2017-10-01 16:00:00',
93         backtest_adjust=ADJUST_PREV,
94         backtest_initial_cash=10000000,
95         backtest_commission_ratio=0.0001,
96         backtest_slippage_ratio=0.0001)
97

```

11.集合竞价选股(股票)

基于收盘价与前收盘价的选股策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  from gm.api import *
4
5  '''
6  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
7  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
8  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.
9  回测数据为:SHSE.000300在2015-01-15的成份股
10  回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
11  '''
12
13

```



```

14 def init(context):
15     # 每月第一个交易日的09:40 定时执行algo任务
16     schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
17     # context.count_bench累计天数阈值
18     context.count_bench = 10
19     # 用于对比的天数
20     context.count = 30
21     # 最大交易资金比例
22     context.ratio = 0.8
23
24
25 def algo(context):
26     # 获取当前时间
27     now = context.now
28     # 获取上一个交易日
29     last_day = get_previous_trading_date(exchange='SHSE', date=now)
30     # 获取沪深300成份股
31     context.stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
32                                               end_date=last_day)[0]['constituents'].keys()
33     # 获取当天有交易的股票
34     not_suspended_info = get_history_instruments(symbols=context.stock300, start_date=now,
35 end_date=now)
36     not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
37 item['is_suspended']]
38
39     trade_symbols = []
40     if not not_suspended_symbols:
41         print('没有当日交易的待选股票')
42         return
43
44     for stock in not_suspended_symbols:
45         recent_data = history_n(symbol=stock, frequency='1d', count=context.count,
46 fields='pre_close,open',
47                                     fill_missing='Last', adjust=ADJUST_PREV, end_time=now,
48 df=True)
49         diff = recent_data['open'] - recent_data['pre_close']
50         # 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
51         if len(diff[diff > 0]) >= context.count_bench:
52             trade_symbols.append(stock)
53
54     print('本次股票池有股票数目: ', len(trade_symbols))
55     # 计算权重
56     percent = 1.0 / len(trade_symbols) * context.ratio
57     # 获取当前所有仓位
58     positions = context.account().positions()
59     # 如标的池有仓位,平不在标的池的仓位
60     for position in positions:
61         symbol = position['symbol']
62         if symbol not in trade_symbols:
63             order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
64 position_side=PositionSide_Long)
65             print('市价单平不在标的池的', symbol)

```

```

63     # 对标的池进行操作
64     for symbol in trade_symbols:
65         order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
66                             position_side=PositionSide_Long)
67         print(symbol, '以市价单调整至权重', percent)
68
69
70 if __name__ == '__main__':
71     '''
72     strategy_id策略ID,由系统生成
73     filename文件名,请与本文件名保持一致
74     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
75     token绑定计算机的ID,可在系统设置-密钥管理中生成
76     backtest_start_time回测开始时间
77     backtest_end_time回测结束时间
78     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
79     backtest_initial_cash回测初始资金
80     backtest_commission_ratio回测佣金比例
81     backtest_slippage_ratio回测滑点比例
82     '''
83     run(strategy_id='strategy_id',
84         filename='main.py',
85         mode=MODE_BACKTEST,
86         token='token_id',
87         backtest_start_time='2017-07-01 08:00:00',
88         backtest_end_time='2017-10-01 16:00:00',
89         backtest_adjust=ADJUST_PREV,
90         backtest_initial_cash=10000000,
91         backtest_commission_ratio=0.0001,
92         backtest_slippage_ratio=0.0001)
93

```

12.多因子选股(股票)

基于Fama三因子构成的多因子策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  import numpy as np
4  from gm.api import *
5  from pandas import DataFrame
6
7  '''
8  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。
9  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。
10  策略思路:
11  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,
12  根据分类得到的组合分别计算其市值加权收益率、SMB和HML。
13  对各个股票进行回归(假设无风险收益率等于0)得到alpha值。
14  选取alpha值小于0并为最小的10只股票进入标的池
15  平掉不在标的池的股票并等权买入在标的池的股票
16  回测数据:SHSE.000300的成份股

```

```

17 回测时间:2017-07-01 08:00:00到2017-10-01 16:00:00
18 '''
19
20
21 def init(context):
22     # 每月第一个交易日的09:40 定时执行algo任务
23     schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
24     # 数据滑窗
25     context.date = 20
26     # 设置开仓的最大资金量
27     context.ratio = 0.8
28     # 账面市值比的大/中/小分类
29     context.BM_BIG = 3.0
30     context.BM_MID = 2.0
31     context.BM_SMA = 1.0
32     # 市值大/小分类
33     context.MV_BIG = 2.0
34     context.MV_SMA = 1.0
35
36
37 # 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
38 def market_value_weighted(stocks, MV, BM):
39     select = stocks[(stocks.NEGOTIABLEMV == MV) & (stocks.BM == BM)]
40     market_value = select['mv'].values
41     mv_total = np.sum(market_value)
42     mv_weighted = [mv / mv_total for mv in market_value]
43     stock_return = select['return'].values
44     # 返回市值加权的收益率的和
45     return_total = []
46     for i in range(len(mv_weighted)):
47         return_total.append(mv_weighted[i] * stock_return[i])
48     return_total = np.sum(return_total)
49     return return_total
50
51
52 def algo(context):
53     # 获取上一个交易日的日期
54     last_day = get_previous_trading_date(exchange='SHSE', date=context.now)
55     # 获取沪深300成份股
56     context.stock300 = get_history_constituents(index='SHSE.000300', start_date=last_day,
57                                                 end_date=last_day)[0]
58     ['constituents'].keys()
59     # 获取当天有交易的股票
60     not_suspended = get_history_instruments(symbols=context.stock300, start_date=last_day,
61                                             end_date=last_day)
62     not_suspended = [item['symbol'] for item in not_suspended if not item['is_suspended']]
63     fin = get_fundamentals(table='trading_derivative_indicator', symbols=not_suspended,
64                           start_date=last_day, end_date=last_day,
65                           fields='PB,NEGOTIABLEMV', df=True)
66
67     # 计算账面市值比,为P/B的倒数
68     fin['PB'] = (fin['PB'] ** -1)
69     # 计算市值的50%的分位点,用于后面的分类

```

```

67     size_gate = fin['NEGOTIABLEMV'].quantile(0.50)
68     # 计算账面市值比的30%和70%分位点,用于后面的分类
69     bm_gate = [fin['PB'].quantile(0.30), fin['PB'].quantile(0.70)]
70     fin.index = fin.symbol
71     x_return = []
72     # 对未停牌的股票进行处理
73     for symbol in not_suspended:
74         # 计算收益率
75         close = history_n(symbol=symbol, frequency='1d', count=context.date + 1,
end_time=last_day, fields='close',
76                             skip_suspended=True, fill_missing='Last', adjust=ADJUST_PREV,
df=True)['close'].values
77         stock_return = close[-1] / close[0] - 1
78         pb = fin['PB'][symbol]
79         market_value = fin['NEGOTIABLEMV'][symbol]
80         # 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]
81         if pb < bm_gate[0]:
82             if market_value < size_gate:
83                 label = [symbol, stock_return, context.BM_SMA, context.MV_SMA,
market_value]
84             else:
85                 label = [symbol, stock_return, context.BM_SMA, context.MV_BIG,
market_value]
86             elif pb < bm_gate[1]:
87                 if market_value < size_gate:
88                     label = [symbol, stock_return, context.BM_MID, context.MV_SMA,
market_value]
89                 else:
90                     label = [symbol, stock_return, context.BM_MID, context.MV_BIG,
market_value]
91             elif market_value < size_gate:
92                 label = [symbol, stock_return, context.BM_BIG, context.MV_SMA, market_value]
93             else:
94                 label = [symbol, stock_return, context.BM_BIG, context.MV_BIG, market_value]
95             if len(x_return) == 0:
96                 x_return = label
97             else:
98                 x_return = np.vstack([x_return, label])
99
100     stocks = DataFrame(data=x_return, columns=['symbol', 'return', 'BM', 'NEGOTIABLEMV',
'mv'])
101     stocks.index = stocks.symbol
102     columns = ['return', 'BM', 'NEGOTIABLEMV', 'mv']
103     for column in columns:
104         stocks[column] = stocks[column].astype(np.float64)
105     # 计算SMB.HML和市场收益率
106     # 获取小市值组合的市值加权组合收益率
107     smb_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
108             market_value_weighted(stocks, context.MV_SMA, context.BM_MID) +
109             market_value_weighted(stocks, context.MV_SMA, context.BM_BIG)) / 3
110
111     # 获取大市值组合的市值加权组合收益率
112     smb_b = (market_value_weighted(stocks, context.MV_BIG, context.BM_SMA) +

```

```

113         market_value_weighted(stocks, context.MV_BIG, context.BM_MID) +
114         market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 3
115
116     smb = smb_s - smb_b
117     # 获取大账面市值比组合的市值加权组合收益率
118     hml_b = (market_value_weighted(stocks, context.MV_SMA, context.BM_BIG) +
119             market_value_weighted(stocks, context.MV_BIG, context.BM_BIG)) / 2
120     # 获取小账面市值比组合的市值加权组合收益率
121     hml_s = (market_value_weighted(stocks, context.MV_SMA, context.BM_SMA) +
122             market_value_weighted(stocks, context.MV_BIG, context.BM_SMA)) / 2
123
124     hml = hml_b - hml_s
125     close = history_n(symbol='SHSE.000300', frequency='1d', count=context.date + 1,
126                      end_time=last_day, fields='close', skip_suspended=True,
127                      fill_missing='Last', adjust=ADJUST_PREV, df=True)['close'].values
128     market_return = close[-1] / close[0] - 1
129     coff_pool = []
130     # 对每只股票进行回归获取其alpha值
131     for stock in stocks.index:
132         x_value = np.array([[market_return], [smb], [hml], [1.0]])
133         y_value = np.array([stocks['return'][stock]])
134         # OLS估计系数
135         coff = np.linalg.lstsq(x_value.T, y_value)[0][3]
136         coff_pool.append(coff)
137
138     # 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)
139     stocks['alpha'] = coff_pool
140     stocks = stocks[stocks.alpha < 0].sort_values(by='alpha').head(10)
141
142     symbols_pool = stocks.index.tolist()
143     positions = context.account().positions()
144
145     # 平不在标的池的股票
146     for position in positions:
147         symbol = position['symbol']
148         if symbol not in symbols_pool:
149             order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
150                                 position_side=PositionSide_Long)
151             print('市价单平不在标的池的', symbol)
152
153     # 获取股票的权重
154     percent = context.ratio / len(symbols_pool)
155     # 买在标的池中的股票
156     for symbol in symbols_pool:
157         order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
158                             position_side=PositionSide_Long)
159         print(symbol, '以市价单调多仓到仓位', percent)
160
161
162 if __name__ == '__main__':
163     '''
164     strategy_id策略ID,由系统生成
165     filename文件名,请与本文件名保持一致

```

```

166 mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
167 token绑定计算机的ID,可在系统设置-密钥管理中生成
168 backtest_start_time回测开始时间
169 backtest_end_time回测结束时间
170 backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
171 backtest_initial_cash回测初始资金
172 backtest_commission_ratio回测佣金比例
173 backtest_slippage_ratio回测滑点比例
174 '''
175 run(strategy_id='strategy_id',
176       filename='main.py',
177       mode=MODE_BACKTEST,
178       token='token_id',
179       backtest_start_time='2017-07-01 08:00:00',
180       backtest_end_time='2017-10-01 16:00:00',
181       backtest_adjust=ADJUST_PREV,
182       backtest_initial_cash=10000000,
183       backtest_commission_ratio=0.0001,
184       backtest_slippage_ratio=0.0001)
185

```

13. 网格交易(期货)

基于网格交易方法的交易策略。

```

1 # coding=utf-8
2 from __future__ import print_function, absolute_import, unicode_literals
3 import numpy as np
4 import pandas as pd
5 from gm.api import *
6
7 ...
8 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
9 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
10 然后根据价格所在的区间来配置仓位：
11 (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]
12 (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)
13 [-0.5, -0.3, 0.0, 0.3, 0.5](资金比例,此处负号表示开空仓)
14 回测数据为:SHFE.rb1801的1min数据
15 回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
16 '''
17
18
19 def init(context):
20     context.symbol = 'SHFE.rb1801'
21     # 订阅SHFE.rb1801, bar频率为1min
22     subscribe(symbols=context.symbol, frequency='60s')
23     # 获取过去300个价格数据
24     timeseries = history_n(symbol=context.symbol, frequency='60s', count=300,
        fields='close', fill_missing='Last',

```

[illegible]

```

69         print(context.symbol, '以市价单开空仓到仓位', context.weight[grid])
70     # 持有空仓的处理
71     elif position_short:
72         # 小于1为在中间网格的下方,做空
73         if grid <= 1:
74             order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
75                                     position_side=PositionSide_Short)
76             print(context.symbol, '以市价单调空仓到仓位', context.weight[grid])
77         # 等于2为在中间网格,平仓
78         elif grid == 2:
79             order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
80                                     position_side=PositionSide_Short)
81             print(context.symbol, '以市价单全平空仓')
82         # 大于3为在中间网格的上方,做多
83         elif grid >= 3:
84             order_target_percent(symbol=context.symbol, percent=0,
order_type=OrderType_Market,
85                                     position_side=PositionSide_Short)
86             print(context.symbol, '以市价单全平空仓')
87             order_target_percent(symbol=context.symbol, percent=context.weight[grid],
order_type=OrderType_Market,
88                                     position_side=PositionSide_Long)
89             print(context.symbol, '以市价单开多仓到仓位', context.weight[grid])
90
91
92 if __name__ == '__main__':
93     '''
94     strategy_id策略ID,由系统生成
95     filename文件名,请与本文件名保持一致
96     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
97     token绑定计算机的ID,可在系统设置-密钥管理中生成
98     backtest_start_time回测开始时间
99     backtest_end_time回测结束时间
100    backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
101    backtest_initial_cash回测初始资金
102    backtest_commission_ratio回测佣金比例
103    backtest_slippage_ratio回测滑点比例
104    '''
105    run(strategy_id='strategy_id',
106        filename='main.py',
107        mode=MODE_BACKTEST,
108        token='token_id',
109        backtest_start_time='2017-07-01 08:00:00',
110        backtest_end_time='2017-10-01 16:00:00',
111        backtest_adjust=ADJUST_PREV,
112        backtest_initial_cash=10000000,
113        backtest_commission_ratio=0.0001,
114        backtest_slippage_ratio=0.0001)
115

```

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。

```
1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3
4  import numpy as np
5  from gm.api import *
6  from pandas import DataFrame
7
8  '''
9  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。
10  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
11  连续上涨5天;连续下跌5天
12  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
13  回测数据为:SHSE.000300中权重大于0.35%的成份股
14  回测时间为:2017-07-01 08:50:00到2017-10-01 17:00:00
15  '''
16
17
18  def init(context):
19      # 资产配置的初始权重,配比为0.6-0.8-1.0
20      context.ratio = 0.8
21      # 获取沪深300当时的成份股和相关数据
22      stock300 = get_history_constituents(index='SHSE.000300', start_date='2017-06-30',
23      end_date='2017-06-30')[0][
24          'constituents']
25      stock300_symbol = []
26      stock300_weight = []
27
28      for key in stock300:
29          # 保留权重大于0.35%的成份股
30          if (stock300[key] / 100) > 0.0035:
31              stock300_symbol.append(key)
32              stock300_weight.append(stock300[key] / 100)
33
34      context.stock300 = DataFrame([stock300_weight], columns=stock300_symbol, index=
35      ['weight']).T
36      context.sum_weight = np.sum(stock300_weight)
37      print('选择的成分股权重总和为: ', context.sum_weight * 100, '%')
38      subscribe(symbols=stock300_symbol, frequency='1d', count=5, wait_group=True)
39
40  def on_bar(context, bars):
41      # 若没有仓位则按照初始权重开仓
42      for bar in bars:
43          symbol = bar['symbol']
44          position = context.account().position(symbol=symbol, side=PositionSide_Long)
45          if not position:
46              buy_percent = context.stock300['weight'][symbol] / context.sum_weight *
47              context.ratio
48              order_target_percent(symbol=symbol, percent=buy_percent,
49              order_type=OrderType_Market,
50              position_side=PositionSide_Long)
```

```

48         print(symbol, '以市价单开多仓至仓位:', buy_percent * 100, '%')
49     else:
50         # 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
51         recent_data = context.data(symbol=symbol, frequency='1d', count=5,
fields='close')['close'].tolist()
52         if all(np.diff(recent_data) > 0):
53             buy_percent = context.stock300['weight'][symbol] / context.sum_weight *
(context.ratio + 0.2)
54             order_target_percent(symbol=symbol, percent=buy_percent,
order_type=OrderType_Market,
55                                     position_side=PositionSide_Long)
56             print('强势股', symbol, '以市价单调多仓至仓位:', buy_percent * 100, '%')
57         elif all(np.diff(recent_data) < 0):
58             buy_percent = context.stock300['weight'][symbol] / context.sum_weight *
(context.ratio - 0.2)
59             order_target_percent(symbol=symbol, percent=buy_percent,
order_type=OrderType_Market,
60                                     position_side=PositionSide_Long)
61             print('弱势股', symbol, '以市价单调多仓至仓位:', buy_percent * 100, '%')
62
63
64 if __name__ == '__main__':
65     '''
66     strategy_id策略ID,由系统生成
67     filename文件名,请与本文件名保持一致
68     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
69     token绑定计算机的ID,可在系统设置-密钥管理中生成
70     backtest_start_time回测开始时间
71     backtest_end_time回测结束时间
72     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
73     backtest_initial_cash回测初始资金
74     backtest_commission_ratio回测佣金比例
75     backtest_slippage_ratio回测滑点比例
76     '''
77     run(strategy_id='strategy_id',
78         filename='main.py',
79         mode=MODE_BACKTEST,
80         token='token_id',
81         backtest_start_time='2017-07-01 08:50:00',
82         backtest_end_time='2017-10-01 17:00:00',
83         backtest_adjust=ADJUST_PREV,
84         backtest_initial_cash=10000000,
85         backtest_commission_ratio=0.0001,
86         backtest_slippage_ratio=0.0001)
87

```

15.跨品种套利(期货)

期货的跨品种套利策略。

```

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

```

```

3  from gm.api import *
4  import numpy as np
5
6  '''
7  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.
8  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
9  若有仓位则在最新价差回归至上下轨水平内时平仓
10  回测数据为:SHFE.rb1801和SHFE.hc1801的1min数据
11  回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
12  '''
13
14
15  def init(context):
16      # 进行套利的品种
17      context.goods = ['SHFE.rb1801', 'SHFE.hc1801']
18      # 订阅行情
19      subscribe(symbols=context.goods, frequency='60s', count=31, wait_group=True)
20
21
22  def on_bar(context, bars):
23      # 获取两个品种的时间序列
24      data_rb = context.data(symbol=context.goods[0], frequency='60s', count=31,
25      fields='close')
26      close_rb = data_rb.values
27      data_hc = context.data(symbol=context.goods[1], frequency='60s', count=31,
28      fields='close')
29      close_hc = data_hc.values
30      # 计算价差
31      spread = close_rb[:-1] - close_hc[:-1]
32      # 计算布林带的上下轨
33      up = np.mean(spread) + 2 * np.std(spread)
34      down = np.mean(spread) - 2 * np.std(spread)
35      # 计算最新价差
36      spread_now = close_rb[-1] - close_hc[-1]
37      # 无交易时若价差上(下)穿布林带上(下)轨则做空(多)价差
38      position_rb_long = context.account().position(symbol=context.goods[0],
39      side=PositionSide_Long)
40      position_rb_short = context.account().position(symbol=context.goods[0],
41      side=PositionSide_Short)
42      if not position_rb_long and not position_rb_short:
43          if spread_now > up:
44              order_target_volume(symbol=context.goods[0], volume=1,
45              order_type=OrderType_Market,
46              position_side=PositionSide_Short)
47              print(context.goods[0], '以市价单开空仓一手')
48              order_target_volume(symbol=context.goods[1], volume=1,
49              order_type=OrderType_Market,
50              position_side=PositionSide_Long)
51              print(context.goods[1], '以市价单开多仓一手')
52          if spread_now < down:
53              order_target_volume(symbol=context.goods[0], volume=1,
54              order_type=OrderType_Market,
55              position_side=PositionSide_Long)

```

```

49         print(context.goods[0], '以市价单开多仓一手')
50         order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
51                             position_side=PositionSide_Short)
52         print(context.goods[1], '以市价单开空仓一手')
53         # 价差回归时平仓
54         elif position_rb_short:
55             if spread_now <= up:
56                 order_close_all()
57                 print('价格回归,平所有仓位')
58                 # 跌破下轨反向开仓
59             if spread_now < down:
60                 order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
61                                     position_side=PositionSide_Long)
62                 print(context.goods[0], '以市价单开多仓一手')
63                 order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
64                                     position_side=PositionSide_Short)
65                 print(context.goods[1], '以市价单开空仓一手')
66         elif position_rb_long:
67             if spread_now >= down:
68                 order_close_all()
69                 print('价格回归,平所有仓位')
70                 # 涨破上轨反向开仓
71             if spread_now > up:
72                 order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
73                                     position_side=PositionSide_Short)
74                 print(context.goods[0], '以市价单开空仓一手')
75                 order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
76                                     position_side=PositionSide_Long)
77                 print(context.goods[1], '以市价单开多仓一手')
78
79
80 if __name__ == '__main__':
81     '''
82     strategy_id策略ID,由系统生成
83     filename文件名,请与本文件名保持一致
84     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
85     token绑定计算机的ID,可在系统设置-密钥管理中生成
86     backtest_start_time回测开始时间
87     backtest_end_time回测结束时间
88     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
89     backtest_initial_cash回测初始资金
90     backtest_commission_ratio回测佣金比例
91     backtest_slippage_ratio回测滑点比例
92     '''
93     run(strategy_id='strategy_id',
94         filename='main.py',
95         mode=MODE_BACKTEST,
96         token='token_id',

```

```

97     backtest_start_time='2017-09-01 08:00:00',
98     backtest_end_time='2017-10-01 16:00:00',
99     backtest_adjust=ADJUST_PREV,
100     backtest_initial_cash=500000,
101     backtest_commission_ratio=0.0001,
102     backtest_slippage_ratio=0.0001)
103

```

16. 跨期套利(期货)

期货的跨期套利策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  import sys
4  import numpy as np
5  from gm.api import *
6  try:
7      import statsmodels.tsa.stattools as ts
8  except:
9      print('请安装statsmodels库')
10     sys.exit(-1)
11
12     '''
13     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
14     操作)
15     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
16     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
17     若有仓位,在残差回归至上下轨内的时候平仓
18     回测数据为:SHFE.rb1801和SHFE.rb1805的1min数据
19     回测时间为:2017-09-25 08:00:00到2017-10-01 15:00:00
20     '''
21
22     # 协整检验的函数
23     def cointegration_test(series01, series02):
24         urt_rb1801 = ts.adfuller(np.array(series01), 1)[1]
25         urt_rb1805 = ts.adfuller(np.array(series02), 1)[1]
26         # 同时平稳或不平稳则差分再次检验
27         if (urt_rb1801 > 0.1 and urt_rb1805 > 0.1) or (urt_rb1801 < 0.1 and urt_rb1805 < 0.1):
28             urt_diff_rb1801 = ts.adfuller(np.diff(np.array(series01)), 1)[1]
29             urt_diff_rb1805 = ts.adfuller(np.diff(np.array(series02)), 1)[1]
30             # 同时差分平稳进行OLS回归的残差平稳检验
31             if urt_diff_rb1801 < 0.1 and urt_diff_rb1805 < 0.1:
32                 matrix = np.vstack([series02, np.ones(len(series02))]).T
33                 beta, c = np.linalg.lstsq(matrix, series01)[0]
34                 resid = series01 - beta * series02 - c
35                 if ts.adfuller(np.array(resid), 1)[1] > 0.1:
36                     result = 0.0
37                 else:
38                     result = 1.0
39             return beta, c, resid, result

```

```

40
41         else:
42             result = 0.0
43             return 0.0, 0.0, 0.0, result
44
45     else:
46         result = 0.0
47         return 0.0, 0.0, 0.0, result
48
49
50 def init(context):
51     context.goods = ['SHFE.rb1801', 'SHFE.rb1805']
52     # 订阅品种
53     subscribe(symbols=context.goods, frequency='60s', count=801, wait_group=True)
54
55
56 def on_bar(context, bars):
57     # 获取过去800个60s的收盘价数据
58     close_01 = context.data(symbol=context.goods[0], frequency='60s', count=801,
59 fields='close')['close'].values
60     close_02 = context.data(symbol=context.goods[1], frequency='60s', count=801,
61 fields='close')['close'].values
62     # 展示两个价格序列的协整检验的结果
63     beta, c, resid, result = cointegration_test(close_01, close_02)
64     # 如果返回协整检验不通过的结果则全平仓位等待
65     if not result:
66         print('协整检验不通过,全平所有仓位')
67         order_close_all()
68         return
69
70     # 计算残差的标准差上下轨
71     mean = np.mean(resid)
72     up = mean + 1.5 * np.std(resid)
73     down = mean - 1.5 * np.std(resid)
74     # 计算新残差
75     resid_new = close_01[-1] - beta * close_02[-1] - c
76     # 获取rb1801的多空仓位
77     position_01_long = context.account().position(symbol=context.goods[0],
78 side=PositionSide_Long)
79     position_01_short = context.account().position(symbol=context.goods[0],
80 side=PositionSide_Short)
81     if not position_01_long and not position_01_short:
82         # 上穿上轨时做空新残差
83         if resid_new > up:
84             order_target_volume(symbol=context.goods[0], volume=1,
85 order_type=OrderType_Market,
86 position_side=PositionSide_Short)
87             print(context.goods[0] + '以市价单开空仓1手')
88             order_target_volume(symbol=context.goods[1], volume=1,
89 order_type=OrderType_Market,
90 position_side=PositionSide_Long)
91             print(context.goods[1] + '以市价单开多仓1手')
92         # 下穿下轨时做多新残差

```

```

87         if resid_new < down:
88             order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
89                                     position_side=PositionSide_Long)
90             print(context.goods[0], '以市价单开多仓1手')
91             order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
92                                     position_side=PositionSide_Short)
93             print(context.goods[1], '以市价单开空仓1手')
94         # 新残差回归时平仓
95         elif position_01_short:
96             if resid_new <= up:
97                 order_close_all()
98                 print('价格回归,平掉所有仓位')
99             # 突破下轨反向开仓
100             if resid_new < down:
101                 order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
102                                     position_side=PositionSide_Long)
103                 print(context.goods[0], '以市价单开多仓1手')
104                 order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
105                                     position_side=PositionSide_Short)
106                 print(context.goods[1], '以市价单开空仓1手')
107             elif position_01_long:
108                 if resid_new >= down:
109                     order_close_all()
110                     print('价格回归,平所有仓位')
111                 # 突破上轨反向开仓
112                 if resid_new > up:
113                     order_target_volume(symbol=context.goods[0], volume=1,
order_type=OrderType_Market,
114                                     position_side=PositionSide_Short)
115                     print(context.goods[0], '以市价单开空仓1手')
116                     order_target_volume(symbol=context.goods[1], volume=1,
order_type=OrderType_Market,
117                                     position_side=PositionSide_Long)
118                     print(context.goods[1], '以市价单开多仓1手')
119
120
121 if __name__ == '__main__':
122     '''
123     strategy_id策略ID,由系统生成
124     filename文件名,请与本文件名保持一致
125     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
126     token绑定计算机的ID,可在系统设置-密钥管理中生成
127     backtest_start_time回测开始时间
128     backtest_end_time回测结束时间
129     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
130     backtest_initial_cash回测初始资金
131     backtest_commission_ratio回测佣金比例
132     backtest_slippage_ratio回测滑点比例
133     '''

```

```

134     run(strategy_id='strategy_id',
135         filename='main.py',
136         mode=MODE_BACKTEST,
137         token='token_id',
138         backtest_start_time='2017-09-25 08:00:00',
139         backtest_end_time='2017-10-01 16:00:00',
140         backtest_adjust=ADJUST_PREV,
141         backtest_initial_cash=500000,
142         backtest_commission_ratio=0.0001,
143         backtest_slippage_ratio=0.0001)
144

```

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。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  import sys
4  try:
5      import talib
6  except:
7      print('请安装TA-Lib库')
8      sys.exit(-1)
9
10 from gm.api import *
11
12 '''
13 本策略首先买入SHSE.600000股票10000股
14 随后根据60s的数据计算MACD(12,26,9),
15 在MACD>0的时候买入100股;在MACD<0的时候卖出100股
16 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
17 回测数据为:SHSE.600000的60s数据
18 回测时间为:2017-09-01 08:00:00到2017-10-01 16:00:00
19 '''
20
21
22 def init(context):
23     # 设置标的股票
24     context.symbol = 'SHSE.600000'
25     # 用于判定第一个仓位是否成功开仓
26     context.first = 0
27     # 订阅浦发银行, bar频率为1min
28     subscribe(symbols=context.symbol, frequency='60s', count=35)
29     # 日内回转每次交易100股
30     context.trade_n = 100
31     # 获取昨今天的时间
32     context.day = [0, 0]
33     # 用于判断是否触发了回转逻辑的计时
34     context.ending = 0
35
36

```



```

37 def on_bar(context, bars):
38     bar = bars[0]
39     if context.first == 0:
40         # 最开始配置仓位
41         # 需要保持的总仓位
42         context.total = 10000
43         # 购买10000股浦发银行股票
44         order_volume(symbol=context.symbol, volume=context.total, side=PositionSide_Long,
45                       order_type=OrderType_Market, position_effect=PositionEffect_Open)
46         print(context.symbol, '以市价单开多仓10000股')
47         context.first = 1.
48         day = bar.bob.strftime('%Y-%m-%d')
49         context.day[-1] = day[-2:]
50         # 每天的仓位操作
51         context.turnaround = [0, 0]
52         return
53
54     # 更新最新的日期
55     day = bar.bob.strftime('%Y-%m-%d %H:%M:%S')
56     context.day[0] = bar.bob.day
57     # 若为新的一天,获取可用于回转的昨仓
58     if context.day[0] != context.day[-1]:
59         context.ending = 0
60         context.turnaround = [0, 0]
61     if context.ending == 1:
62         return
63
64     # 若有可用的昨仓则操作
65     if context.total >= 0:
66         # 获取时间序列数据
67         symbol = bar['symbol']
68         recent_data = context.data(symbol=symbol, frequency='60s', count=35,
69                                   fields='close')
69         # 计算MACD线
70         macd = talib.MACD(recent_data['close'].values)[0][-1]
71         # 根据MACD>0则开仓,小于0则平仓
72         if macd > 0:
73             # 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
74             if context.turnaround[0] + context.trade_n < context.total:
75                 # 计算累计仓位
76                 context.turnaround[0] += context.trade_n
77                 order_volume(symbol=context.symbol, volume=context.trade_n,
78                               side=PositionSide_Long,
79                               order_type=OrderType_Market,
80                               position_effect=PositionEffect_Open)
81                 print(symbol, '市价单开多仓', context.trade_n, '股')
82             elif macd < 0:
83                 if context.turnaround[1] + context.trade_n < context.total:
84                     context.turnaround[1] += context.trade_n
85                     order_volume(symbol=context.symbol, volume=context.trade_n,
86                                   side=PositionSide_Short,
87                                   order_type=OrderType_Market,
88                                   position_effect=PositionEffect_Close)

```

```

85         print(symbol, '市价单平多仓', context.trade_n, '股')
86         # 临近收盘时若仓位数不等于昨仓则回转所有仓位
87         if day[11:16] == '14:55' or day[11:16] == '14:57':
88             position = context.account().position(symbol=context.symbol,
side=PositionSide_Long)
89             if position['volume'] != context.total:
90                 order_target_volume(symbol=context.symbol, volume=context.total,
order_type=OrderType_Market,
91                                     position_side=PositionSide_Long)
92                 print('市价单回转仓位操作...')
93                 context.ending = 1
94         # 更新过去的日期数据
95         context.day[-1] = context.day[0]
96
97
98 if __name__ == '__main__':
99     '''
100     strategy_id策略ID,由系统生成
101     filename文件名,请与本文件名保持一致
102     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
103     token绑定计算机的ID,可在系统设置-密钥管理中生成
104     backtest_start_time回测开始时间
105     backtest_end_time回测结束时间
106     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
107     backtest_initial_cash回测初始资金
108     backtest_commission_ratio回测佣金比例
109     backtest_slippage_ratio回测滑点比例
110     '''
111     run(strategy_id='strategy_id',
112         filename='main.py',
113         mode=MODE_BACKTEST,
114         token='token_id',
115         backtest_start_time='2017-09-01 08:00:00',
116         backtest_end_time='2017-10-01 16:00:00',
117         backtest_adjust=ADJUST_PREV,
118         backtest_initial_cash=2000000,
119         backtest_commission_ratio=0.0001,
120         backtest_slippage_ratio=0.0001)
121

```

18.做市商策略(期货)

基于Tick价差的交易策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  from gm.api import *
4
5  '''
6  本策略通过不断对CZCE.CF801进行
7  买一价现价单开多仓和卖一价平多仓;
8  卖一价现价单开空仓和买一价平空仓来做市

```

```

9  并以此赚取差价
10  回测数据为:CZCE.CF801的tick数据
11  回测时间为:2017-09-29 11:25:00到2017-09-29 11:30:00
12  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.
13  敬请通过适当调整回测参数
14  1.backtest_commission_ratio回测佣金比例
15  2.backtest_slippage_ratio回测滑点比例
16  3.backtest_transaction_ratio回测成交比例
17  以及优化策略逻辑来达到更贴近实际的回测效果
18  '''
19
20
21  def init(context):
22      # 订阅CZCE.CF801的tick数据
23      context.symbol = 'CZCE.CF801'
24      subscribe(symbols=context.symbol, frequency='tick')
25
26
27  def on_tick(context, tick):
28      quotes = tick['quotes'][0]
29      # 获取持有的多仓
30      positio_long = context.account().position(symbol=context.symbol, side=PositionSide_Long)
31      # 获取持有的空仓
32      position_short = context.account().position(symbol=context.symbol,
33      side=PositionSide_Short)
34      print(quotes['bid_p'])
35      print(quotes['ask_p'])
36      # 没有仓位则双向开限价单
37      # 若有仓位则限价单平仓
38      if not positio_long:
39          # 获取买一价
40          price = quotes['bid_p']
41          print('买一价为: ', price)
42          order_target_volume(symbol=context.symbol, volume=1, price=price,
43          order_type=OrderType_Limit,
44          position_side=PositionSide_Long)
45          print('CZCE.CF801开限价单多仓1手')
46      else:
47          # 获取卖一价
48          price = quotes['ask_p']
49          print('卖一价为: ', price)
50          order_target_volume(symbol=context.symbol, volume=0, price=price,
51          order_type=OrderType_Limit,
52          position_side=PositionSide_Long)
53          print('CZCE.CF801平限价单多仓1手')
54      if not position_short:
55          # 获取卖一价
56          price = quotes['ask_p']
57          print('卖一价为: ', price)
58          order_target_volume(symbol=context.symbol, volume=1, price=price,
59          order_type=OrderType_Limit,
60          position_side=PositionSide_Short)
61          print('CZCE.CF801卖一价开限价单空仓')

```

```

58     else:
59         # 获取买一价
60         price = quotes['bid_p']
61         print('买一价为: ', price)
62         order_target_volume(symbol=context.symbol, volume=0, price=price,
order_type=OrderType_Limit,
63                             position_side=PositionSide_Short)
64         print('CZCE.CF801买一价平限价单空仓')
65
66
67 if __name__ == '__main__':
68     '''
69     strategy_id策略ID,由系统生成
70     filename文件名,请与本文件名保持一致
71     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
72     token绑定计算机的ID,可在系统设置-密钥管理中生成
73     backtest_start_time回测开始时间
74     backtest_end_time回测结束时间
75     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
76     backtest_initial_cash回测初始资金
77     backtest_commission_ratio回测佣金比例
78     backtest_slippage_ratio回测滑点比例
79     backtest_transaction_ratio回测成交比例
80     '''
81     run(strategy_id='strategy_id',
82         filename='main.py',
83         mode=MODE_BACKTEST,
84         token='token_id',
85         backtest_start_time='2017-09-29 11:25:00',
86         backtest_end_time='2017-09-29 11:30:00',
87         backtest_adjust=ADJUST_PREV,
88         backtest_initial_cash=500000,
89         backtest_commission_ratio=0.00006,
90         backtest_slippage_ratio=0.0001,
91         backtest_transaction_ratio=0.5)
92

```

19.海龟交易法(期货)

基于海龟交易法则的交易策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3
4  import sys
5
6  import numpy as np
7  import pandas as pd
8
9  try:
10     import talib
11 except:

```

```

12     print('请安装TA-Lib库')
13     sys.exit(-1)
14 from gm.api import *
15
16 '''
17 本策略通过计算CZCE.FG801和SHFE.rb1801的ATR.唐奇安通道和MA线,
18 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)
19 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
20 持仓均价 - x(x=0.5,1,1.5,2)倍ATR把仓位平至6/4/2/0手
21 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
22 持仓均价 + x(x=0.5,1,1.5,2)倍ATR把仓位平至6/4/2/0手
23 回测数据为:CZCE.FG801和SHFE.rb1801的1min数据
24 回测时间为:2017-09-15 09:15:00到2017-10-01 15:00:00
25 '''
26
27
28 def init(context):
29     # 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
30     context.parameter = [55, 20, 10, 60, 20]
31     context.tar = context.parameter[4]
32     # context.goods交易的品种
33     context.goods = ['CZCE.FG801', 'SHFE.rb1801']
34     # 订阅context.goods里面的品种, bar频率为1min
35     subscribe(symbols=context.goods, frequency='60s', count=101)
36     # 止损的比例区间
37
38
39 def on_bar(context, bars):
40     bar = bars[0]
41     symbol = bar['symbol']
42     recent_data = context.data(symbol=symbol, frequency='60s', count=101,
43 fields='close,high,low')
44     close = recent_data['close'].values[-1]
45     # 计算ATR
46     atr = talib.ATR(recent_data['high'].values, recent_data['low'].values,
47 recent_data['close'].values,
48                     timeperiod=context.tar)[-1]
49     # 计算唐奇安开仓和平仓通道
50     context.don_open = context.parameter[0] + 1
51     upper_band = talib.MAX(recent_data['close'].values[:-1], timeperiod=context.don_open)
52 [-1]
53     context.don_close = context.parameter[1] + 1
54     lower_band = talib.MIN(recent_data['close'].values[:-1], timeperiod=context.don_close)
55 [-1]
56     # 若没有仓位则开仓
57     position_long = context.account().position(symbol=symbol, side=PositionSide_Long)
58
59     position_short = context.account().position(symbol=symbol, side=PositionSide_Short)
60     if not position_long and not position_short:
61         # 计算长短ma线.DIF
62         ma_short = talib.MA(recent_data['close'].values, timeperiod=(context.parameter[2] +
63 1))[-1]

```

```

59     ma_long = talib.MA(recent_data['close'].values, timeperiod=(context.parameter[3] +
60 1))[-1]
61     dif = ma_short - ma_long
62     # 获取当前价格
63     # 上穿唐奇安通道且短ma在长ma上方则开多仓
64     if close > upper_band and (dif > 0):
65         order_target_volume(symbol=symbol, volume=8, position_side=PositionSide_Long,
66 order_type=OrderType_Market)
67         print(symbol, '市价单开多仓8手')
68     # 下穿唐奇安通道且短ma在长ma下方则开空仓
69     if close < lower_band and (dif < 0):
70         order_target_volume(symbol=symbol, volume=8, position_side=PositionSide_Short,
71 order_type=OrderType_Market)
72         print(symbol, '市价单开空仓8手')
73     elif position_long:
74         # 价格跌破唐奇安平仓通道全平仓位止损
75         if close < lower_band:
76             order_close_all()
77             print(symbol, '市价单全平仓位')
78         else:
79             # 获取持仓均价
80             vwap = position_long['vwap']
81             # 获取持仓的资金
82             band = vwap - np.array([200, 2, 1.5, 1, 0.5, -100]) * atr
83             # 计算最新应持仓位
84             grid_volume = int(pd.cut([close], band, labels=[0, 1, 2, 3, 4])[0]) * 2
85             order_target_volume(symbol=symbol, volume=grid_volume,
86 position_side=PositionSide_Long,
87 order_type=OrderType_Market)
88             print(symbol, '市价单平多仓到', grid_volume, '手')
89     elif position_short:
90         # 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
91         if close > upper_band:
92             order_close_all()
93             print(symbol, '市价单全平仓位')
94         else:
95             # 获取持仓均价
96             vwap = position_short['vwap']
97             # 获取平仓的区间
98             band = vwap + np.array([-100, 0.5, 1, 1.5, 2, 200]) * atr
99             # 计算最新应持仓位
100             grid_volume = int(pd.cut([close], band, labels=[0, 1, 2, 3, 4])[0]) * 2
101             order_target_volume(symbol=symbol, volume=grid_volume,
102 position_side=PositionSide_Short,
103 order_type=OrderType_Market)
104             print(symbol, '市价单平空仓到', grid_volume, '手')
105
106 if __name__ == '__main__':
107     '''
108     strategy_id策略ID,由系统生成
109     filename文件名,请与本文件名保持一致
110     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST

```

```

107 token绑定计算机的ID,可在系统设置-密钥管理中生成
108 backtest_start_time回测开始时间
109 backtest_end_time回测结束时间
110 backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
111 backtest_initial_cash回测初始资金
112 backtest_commission_ratio回测佣金比例
113 backtest_slippage_ratio回测滑点比例
114 '''
115 run(strategy_id='strategy_id',
116     filename='main.py',
117     mode=MODE_BACKTEST,
118     token='token_id',
119     backtest_start_time='2017-09-15 09:15:00',
120     backtest_end_time='2017-10-01 15:00:00',
121     backtest_adjust=ADJUST_PREV,
122     backtest_initial_cash=10000000,
123     backtest_commission_ratio=0.0001,
124     backtest_slippage_ratio=0.0001)
125

```

20.行业轮动(股票)

基于沪深300的行业指数的行业轮动策略。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  import numpy as np
4  from gm.api import *
5
6  '''
7  本策略每隔1个月定时触发计算
8  SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.000914
9  (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
10  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
11  对不在股票池的股票平仓并等权配置股票池的标的
12  回测数据为:SHSE.000910.SHSE.000909.SHSE.000911.SHSE.000912.SHSE.000913.SHSE.000914和他们的成份
13  股
14  回测时间为:2017-07-01 08:00:00到2017-10-01 16:00:00
15  '''
16
17  def init(context):
18      # 每月第一个交易日的09:40 定时执行algo任务
19      schedule(schedule_func=algo, date_rule='1m', time_rule='09:40:00')
20      # 用于筛选的行业指数
21      context.index = ['SHSE.000910', 'SHSE.000909', 'SHSE.000911', 'SHSE.000912',
22                      'SHSE.000913', 'SHSE.000914']
23      # 用于统计数据的天数
24      context.count = 20
25      # 最大下单资金比例
26      context.ratio = 0.8

```

```

26
27 def algo(context):
28     # 获取当天的日期
29     today = context.now
30     # 获取上一个交易日
31     last_day = get_previous_trading_date(exchange='SHSE', date=today)
32     return_index = []
33     # 获取并计算行业指数收益率
34
35     for i in context.index:
36         return_index_his = history_n(symbol=i, frequency='1d', count=context.count,
37 fields='close,bob',
38                                     fill_missing='Last', adjust=ADJUST_PREV,
39 end_time=last_day, df=True)
40         return_index_his = return_index_his['close'].values
41         return_index.append(return_index_his[-1] / return_index_his[0] - 1)
42     # 获取指定数内收益率表现最好的行业
43     sector = context.index[np.argmax(return_index)]
44     print('最佳行业指数是: ', sector)
45     # 获取最佳行业指数成份股
46     symbols = get_history_constituents(index=sector, start_date=last_day, end_date=last_day)
47     [0]['constituents'].keys()
48     # 获取当天有交易的股票
49     not_suspended_info = get_history_instruments(symbols=symbols, start_date=today,
50 end_date=today)
51     not_suspended_symbols = [item['symbol'] for item in not_suspended_info if not
52 item['is_suspended']]
53
54     # 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
55     fin = get_fundamentals(table='trading_derivative_indicator',
56 symbols=not_suspended_symbols, start_date=last_day,
57 end_date=last_day, limit=5, fields='NEGOTIABLEMV', order_by='-
58 NEGOTIABLEMV', df=True)
59     fin.index = fin['symbol']
60     # 计算权重
61     percent = 1.0 / len(fin.index) * context.ratio
62     # 获取当前所有仓位
63     positions = context.account().positions()
64     # 如标的池有仓位,平不在标的池的仓位
65     for position in positions:
66         symbol = position['symbol']
67         if symbol not in fin.index:
68             order_target_percent(symbol=symbol, percent=0, order_type=OrderType_Market,
69 position_side=PositionSide_Long)
70             print('市价单平不在标的池的', symbol)
71     # 对标的池进行操作
72     for symbol in fin.index:
73         order_target_percent(symbol=symbol, percent=percent, order_type=OrderType_Market,
74 position_side=PositionSide_Long)
75         print(symbol, '以市价单调整至仓位', percent)
76
77 if __name__ == '__main__':

```



```

72     '''
73     strategy_id策略ID,由系统生成
74     filename文件名,请与本文件名保持一致
75     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
76     token绑定计算机的ID,可在系统设置-密钥管理中生成
77     backtest_start_time回测开始时间
78     backtest_end_time回测结束时间
79     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
80     backtest_initial_cash回测初始资金
81     backtest_commission_ratio回测佣金比例
82     backtest_slippage_ratio回测滑点比例
83     '''
84     run(strategy_id='strategy_id',
85         filename='main.py',
86         mode=MODE_BACKTEST,
87         token='token_id',
88         backtest_start_time='2017-07-01 08:00:00',
89         backtest_end_time='2017-10-01 16:00:00',
90         backtest_adjust=ADJUST_PREV,
91         backtest_initial_cash=10000000,
92         backtest_commission_ratio=0.0001,
93         backtest_slippage_ratio=0.0001)
94

```

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。

```

1  # coding=utf-8
2  from __future__ import print_function, absolute_import, unicode_literals
3  from datetime import datetime
4  import numpy as np
5  from gm.api import *
6  import sys
7  try:
8      from sklearn import svm
9  except:
10     print('请安装scikit-learn库和带mk1的numpy')
11     sys.exit(-1)
12
13     '''
14     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.
15     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.
16     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.
17     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
18     训练数据为:SHSE.600009上海机场,时间从2016-04-01到2017-07-30
19     回测时间为:2017-08-01 09:00:00到2017-09-05 09:00:00
20     '''
21
22

```

```

23 def init(context):
24     # 订阅上海机场的分钟bar行情
25     context.symbol = 'SHSE.600009'
26     subscribe(symbols=context.symbol, frequency='60s')
27     start_date = '2016-03-01' # SVM训练起始时间
28     end_date = '2017-06-30' # SVM训练终止时间
29     # 用于记录工作日
30     # 获取目标股票的daily历史行情
31     recent_data = history(symbol=context.symbol, frequency='1d', start_time=start_date,
end_time=end_date, fill_missing='Last',
32                             df=True)
33     days_value = recent_data['bob'].values
34     days_close = recent_data['close'].values
35     days = []
36     # 获取行情日期列表
37     print('准备数据训练SVM')
38     for i in range(len(days_value)):
39         days.append(str(days_value[i])[0:10])
40
41     x_all = []
42     y_all = []
43     for index in range(15, (len(days) - 5)):
44         # 计算三星期共15个交易日相关数据
45         start_day = days[index - 15]
46         end_day = days[index]
47         data = history(symbol=context.symbol, frequency='1d', start_time=start_day,
end_time=end_day, fill_missing='Last',
48                         df=True)
49         close = data['close'].values
50         max_x = data['high'].values
51         min_n = data['low'].values
52         amount = data['amount'].values
53         volume = []
54         for i in range(len(close)):
55             volume_temp = amount[i] / close[i]
56             volume.append(volume_temp)
57
58         close_mean = close[-1] / np.mean(close) # 收盘价/均值
59         volume_mean = volume[-1] / np.mean(volume) # 现量/均量
60         max_mean = max_x[-1] / np.mean(max_x) # 最高价/均价
61         min_mean = min_n[-1] / np.mean(min_n) # 最低价/均价
62         vol = volume[-1] # 现量
63         return_now = close[-1] / close[0] # 区间收益率
64         std = np.std(np.array(close), axis=0) # 区间标准差
65
66         # 将计算出的指标添加到训练集x
67         # features用于存放因子
68         features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now, std]
69         x_all.append(features)
70
71     # 准备算法需要用到数据
72     for i in range(len(days_close) - 20):
73         if days_close[i + 20] > days_close[i + 15]:

```

```

74         label = 1
75     else:
76         label = 0
77     y_all.append(label)
78
79     x_train = x_all[:-1]
80     y_train = y_all[:-1]
81     # 训练SVM
82     context.clf = svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
shrinking=True, probability=False,
83                             tol=0.001, cache_size=400, verbose=False, max_iter=-1,
84                             decision_function_shape='ovr', random_state=None)
85     context.clf.fit(x_train, y_train)
86     print('训练完成!')
87
88
89 def on_bar(context, bars):
90     bar = bars[0]
91     # 获取当前年月日
92     today = bar.bob.strftime('%Y-%m-%d')
93     last_day = get_previous_trading_date(exchange='SHSE', date=today)
94     # 获取数据并计算相应的因子
95     # 于星期一的09:31:00进行操作
96     # 当前bar的工作日
97     weekday = datetime.strptime(today, '%Y-%m-%d').isoweekday()
98     # 获取模型相关的数据
99     # 获取持仓
100    position = context.account().position(symbol=context.symbol, side=PositionSide_Long)
101    # 如果bar是新的星期且没有仓位则开始预测
102    if not position and weekday == 1:
103        # 获取预测用的历史数据
104        data = history_n(symbol=context.symbol, frequency='1d', end_time=last_day,
count=15,
105                        fill_missing='Last', adjust=ADJUST_PREV, df=True)
106        close = data['close'].values
107        train_max_x = data['high'].values
108        train_min_n = data['low'].values
109        train_amount = data['amount'].values
110        volume = []
111        for i in range(len(close)):
112            volume_temp = train_amount[i] / close[i]
113            volume.append(volume_temp)
114
115        close_mean = close[-1] / np.mean(close)
116        volume_mean = volume[-1] / np.mean(volume)
117        max_mean = train_max_x[-1] / np.mean(train_max_x)
118        min_mean = train_min_n[-1] / np.mean(train_min_n)
119        vol = volume[-1]
120        return_now = close[-1] / close[0]
121        std = np.std(np.array(close), axis=0)
122
123        # 得到本次输入模型的因子
124        features = [close_mean, volume_mean, max_mean, min_mean, vol, return_now, std]

```

```

125     features = np.array(features).reshape(1, -1)
126     prediction = context.clf.predict(features)[0]
127     # 若预测值为上涨则开仓
128     if prediction == 1:
129         # 获取昨收盘价
130         context.price = close[-1]
131         # 把浦发银行的仓位调至95%
132         order_target_percent(symbol=context.symbol, percent=0.95,
order_type=OrderType_Market,
133                             position_side=PositionSide_Long)
134         print(context.symbol, '以市价单开多仓到仓位0.95')
135     # 当涨幅大于10%,平掉所有仓位止盈
136     elif position and bar.close / context.price >= 1.10:
137         order_close_all()
138         print(context.symbol, '以市价单全平多仓止盈')
139     # 当时间为周五并且跌幅大于2%时,平掉所有仓位止损
140     elif position and bar.close / context.price < 1.02 and weekday == 5:
141         order_close_all()
142         print(context.symbol, '以市价单全平多仓止损')
143
144
145 if __name__ == '__main__':
146     '''
147     strategy_id策略ID,由系统生成
148     filename文件名,请与本文件名保持一致
149     mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
150     token绑定计算机的ID,可在系统设置-密钥管理中生成
151     backtest_start_time回测开始时间
152     backtest_end_time回测结束时间
153     backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
154     backtest_initial_cash回测初始资金
155     backtest_commission_ratio回测佣金比例
156     backtest_slippage_ratio回测滑点比例
157     '''
158     run(strategy_id='strategy_id',
159         filename='main.py',
160         mode=MODE_BACKTEST,
161         token='token_id',
162         backtest_start_time='2017-08-01 09:00:00',
163         backtest_end_time='2017-09-05 09:00:00',
164         backtest_adjust=ADJUST_PREV,
165         backtest_initial_cash=10000000,
166         backtest_commission_ratio=0.0001,
167         backtest_slippage_ratio=0.0001)
168

```

22.参数优化(股票+期货)

基于循环遍历回测的参数优化方法。

```

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

```

```

3
4 import multiprocessing
5
6 import numpy as np
7 import pandas as pd
8 import talib
9 from gm.api import *
10
11 '''
12 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，
13 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。
14 1、定义策略函数
15 2、多进程循环输入参数数值
16 3、获取回测报告，生成DataFrame格式
17 4、排序
18 本程序以双均线策略为例，优化两均线长短周期参数。
19 '''
20
21
22 # 原策略中的参数定义语句需要删除！
23 def init(context):
24     context.sec_id = 'SHSE.600000'
25     subscribe(symbols=context.sec_id, frequency='1d', count=31, wait_group=True)
26
27
28 def on_bar(context, bars):
29     close = context.data(symbol=context.sec_id, frequency='1d', count=31, fields='close')
30     ['close'].values
31     MA_short = talib.MA(close, timeperiod=context.short)
32     MA_long = talib.MA(close, timeperiod=context.long)
33     position = context.account().position(symbol=context.sec_id, side=PositionSide_Long)
34     if not position and not position:
35         if MA_short[-1] > MA_long[-1] and MA_short[-2] < MA_long[-2]:
36             order_target_percent(symbol=context.sec_id, percent=0.8,
37 order_type=OrderType_Market,
38 position_side=PositionSide_Long)
39
40         elif position:
41             if MA_short[-1] < MA_long[-1] and MA_short[-2] > MA_long[-2]:
42                 order_target_percent(symbol=context.sec_id, percent=0,
43 order_type=OrderType_Market,
44 position_side=PositionSide_Long)
45
46
47 # 获取每次回测的报告数据
48 def on_backtest_finished(context, indicator):
49     data = [indicator['pnl_ratio'], indicator['pnl_ratio_annual'], indicator['sharp_ratio'],
50 indicator['max_drawdown'],
51 context.short, context.long]
52     # 将回测报告加入全局list，以便记录
53     context.list.append(data)
54
55
56 def run_strategy(short, long, a_list):

```

```

52     from gm.model.storage import context
53     # 用context传入参数
54     context.short = short
55     context.long = long
56     # a_list一定要传入
57     context.list = a_list
58     '''
59         strategy_id策略ID,由系统生成
60         filename文件名,请与本文件名保持一致
61         mode实时模式:MODE_LIVE回测模式:MODE_BACKTEST
62         token绑定计算机的ID,可在系统设置-密钥管理中生成
63         backtest_start_time回测开始时间
64         backtest_end_time回测结束时间
65         backtest_adjust股票复权方式不复权:ADJUST_NONE前复权:ADJUST_PREV后复权:ADJUST_POST
66         backtest_initial_cash回测初始资金
67         backtest_commission_ratio回测佣金比例
68         backtest_slippage_ratio回测滑点比例
69     '''
70     run(strategy_id='strategy_id',
71         filename='main.py',
72         mode=MODE_BACKTEST,
73         token='token_id',
74         backtest_start_time='2017-05-01 08:00:00',
75         backtest_end_time='2017-10-01 16:00:00',
76         backtest_adjust=ADJUST_PREV,
77         backtest_initial_cash=50000,
78         backtest_commission_ratio=0.0001,
79         backtest_slippage_ratio=0.0001)
80
81
82 if __name__ == '__main__':
83     # 生成全局list
84     manager = multiprocessing.Manager()
85     a_list = manager.list()
86     # 循环输入参数数值回测
87     for short in range(5, 10, 2):
88         for long in range(10, 21, 5):
89             process = multiprocessing.Process(target=run_strategy, args=(short, long,
a_list))
90             process.start()
91             process.join()
92     # 回测报告转化成DataFrame格式
93     a_list = np.array(a_list)
94     final = pd.DataFrame(a_list,
95                          columns=['pnl_ratio', 'pnl_ratio_annual', 'sharp_ratio',
'max_drawdown', 'short', 'long'])
96     # 回测报告排序
97     final = final.sort_values(axis=0, ascending=False, by='pnl_ratio')
98     print(final)
99

```

