

# 目 录

## 快速开始

### 指引

- 快速新建策略
- 编译策略
- 获取SDK
- 建立我们第一个策略
- 策略应该是这样的
- 继承策略基类
- 重改关注事件
- 在OnInit里订阅行情，初始化
- 在main里实例化一个派生类对象
- 开始运行
- 订阅行情策略示例

### 典型场景

- 空策略
- 定时任务
- 数据事件驱动
- 默认交易账号
- 显示指定交易账号
- 模式选择
- 数据研究

## 重要概念

- symbol - 代码标识

- 策略运行模式

## 策略基类

### 基类原型

- 策略类简介
- 策略类定义

### 基本成员函数

- Strategy - 构造函数
- Run - 运行策略
- Stop - 停止策略
- SetToken - 设置用户token
- SetMode - 设置运行模式
- SetStrategyId - 设置策略ID
- GetAccountStatus - 获取策略所有账户状态

GetAccountStatus - 获取指定账户状态

Schedule - 预设定时任务

Now - 获取当前时间

SetBacktestConfig - 设置回测参数

行情成员函数

- subscribe - 订阅行情
- unsubscribe - 退订行情

普通交易成员函数

- GetAccounts - 查询交易账号
- PlaceOrder - 按指定量委托
- OrderVolume - 按指定量委托
- OrderValue - 按指定价值委托
- OrderPercent - 按总资产指定比例委托
- OrderTargetVolume - 调仓到目标持仓量
- OrderTargetValue - 调仓到目标持仓额
- OrderTargetPercent - 调仓到目标持仓比例（总资产的比例）
- OrderCloseAll - 平当前所有可平持仓
- OrderCancel - 委托撤单
- OrderCancelAll - 撤销所有委托
- GetOrders - 查询所有委托
- GetUnfinishedOrders - 查询未结委托
- GetExecutionReports - 查询成交
- GetCash - 查询资金
- GetPosition - 查询持仓

两融业务成员函数

- CreditBuyingOnMargin - 融资买入
- CreditShortSelling - 融券卖出
- CreditRepayShareByBuyingShare - 买券还券
- CreditRepayShareByBuyingShare - 卖券还款
- CreditBuyingOnCollateral - 担保品买入
- CreditSellingOnCollateral - 担保品卖出
- CreditRepayShareDirectly - 直接还券
- CreditRepayCashDirectly - 直接还款
- CreditCollateralIn - 担保品转入
- CreditCollateralOut - 担保品转出
- CreditGetCollateralInstruments - 查询担保证券
- CreditGetBorrowableInstruments - 查询融券标的证券
- CreditGetBorrowableInstrumentsPositions - 查询融券账户头寸

CreditGetContracts - 查询融资融券合约

CreditGetCash - 查询融资融券资金

#### 算法交易成员函数

OrderAlgo - 委托算法单

AlgoOrderCancel - 撤单算法委托

AlgoOrderPause - 暂停/恢复算法单

GetAlgoOrders - 查询算法委托

GetChildOrders - 查询算法子委托

#### 新股业务成员函数

IPOBuy - 新股新债申购

IPOGetQuota - 查询客户新股新债申购额度

IPOGetInstruments - 查询当日新股新债清单

IPOGetMatchNumber - 配号查询

IPOGetLotInfo - 中签查询

#### 基金业务成员函数

FundEtfBuy - ETF申购

FundEtfRedemption - ETF赎回

FundSubScribing - 基金认购

FundBuy - 基金申购

FundRedemption - 基金赎回

#### 债券业务成员函数

BondReverseRepurchaseAgreement - 国债逆回购

BondConvertibleCall - 可转债转股

BondConvertiblePut - 可转债回售

BondConvertiblePutCancel - 可转债回售撤销

#### 动态参数成员函数

AddParameters - 添加参数

DelParameters - 删除参数

SetParameters - 设置参数

GetParameters - 获取参数

SetSymbols - 设置标的

GetSymbols - 获取标的

#### 事件成员函数

OnInit - 初始化完成

OnTick - 收到Tick行情

OnBar - 收到bar行情

OnOrderStatus - 委托变化

OnExecutionReport - 执行回报

OnParameter - 参数变化  
OnSchedule - 定时任务触发  
OnBacktestFinished - 回测完成后收到绩效报告  
OnAccountStatus - 实盘账号状态变化  
OnError - 错误产生  
OnStop - 收到策略停止信号  
OnMarketDataConnected - 数据服务已经连接上  
OnTradeDataConnected - 交易已经连接上  
OnMarketDataDisconnected - 数据连接断开了  
OnTradeDataDisconnected - 交易连接断开了

## 数据查询函数

### 数据查询函数

GMApi 静态方法  
SetToken - 设置token  
SetAddr - 设置终端服务地址  
Current - 查询当前行情快照  
HistoryTicks - 查询历史Tick行情  
HistoryBars - 查询历史Bar行情  
HistoryTicksN - 查询最新n条Tick行情  
HistoryBarsN - 查询最新n条Bar行情  
GetFundamentals - 查询基本面数据  
GetFundamentalsN - 查询基本面数据最新n条  
GetInstruments - 查询最新交易标的信息  
GetHistoryInstruments - 查询交易标的历史数据  
GetInstrumentinfos - 查询交易标的基本信息  
GetConstituents - 查询指数成份股  
GetIndustry - 查询行业股票列表  
GetTradingDates - 查询交易日历  
GetPreviousTradingDate - 返回指定日期的上一个交易日  
GetNextTradingDate - 返回指定日期的下一个交易日  
GetDividend - 查询分红送配  
GetContinuousContracts - 获取连续合约

## 结果集合类

### GMDDataList

GMDDataList 类定义

使用举例

### GMDData

GMDData 类定义

## 使用举例

### 数据结构

#### 数据类

Tick - Tick结构

Bar - Bar结构

#### 交易类

Account - 账户结构

AccountStatus - 账户状态结构

PlaceOrders - 下单参数结构

Order - 委托结构

ExecRpt - 回报结构

Cash - 资金结构

Position - 持仓结构

Indicator - 绩效指标结构

Parameter - 动态参数结构

### 枚举常量

OrderStatus - 委托状态

OrderSide - 委托方向

SecType - 标的类别

OrderType - 委托类型

ExecType - 执行回报类型

PositionEffect - 开平仓类型

PositionSide - 持仓方向

OrderRejectReason - 订单拒绝原因

CashPositionChangeReason - 仓位变更原因

OrderDuration - 委托时间属性

OrderQualifier - 委托成交属性

AccountState - 交易账户状态

StrategyMode - 策略模式

Adjust - 复权方式

### 错误码

# 指引

- [快速新建策略](#)
- [编译策略](#)
- [获取SDK](#)
- [建立我们第一个策略](#)
- [策略应该是这样的](#)
- [继承策略基类](#)
- [重改关注事件](#)
- [在OnInit里订阅行情，初始化](#)
- [在main里实例化一个派生类对象](#)
- [开始运行](#)
- [订阅行情策略示例](#)
  - [源文件](#)

## 快速新建策略

- 下载掘金3终端
- 打开终端后，登陆掘金账号点击研究策略，新建策略或者点击右上角新建策略



- 新建一个典型默认账户交易策略  
新建C#的默认账户交易策略



## 编译策略

- 打开新建策略文件目录  
策略文件目录内容可以拷贝到本地其他盘符也可以进行编译生成



名称	修改日期	类型	大小
bin	2018/8/8 14:35	文件夹	
Properties	2018/8/8 14:35	文件夹	
App.config	2018/8/8 14:35	XML Configurati...	1 KB
DefaultAccountTransaction.csproj	2018/8/8 14:35	Visual C# 项目文件	4 KB
DefaultAccountTransaction.sln	2018/8/8 14:35	Visual Studio 解	2 KB
Program.cs	2018/8/8 14:35	Visual C# Sourc...	3 KB

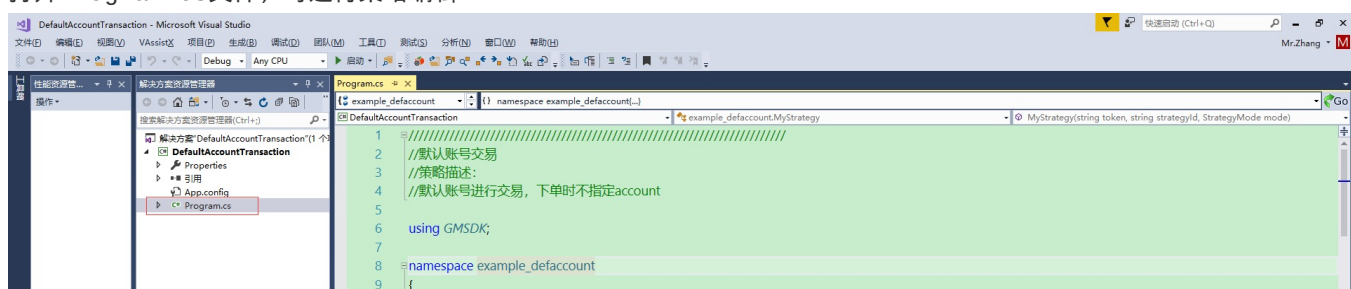
- 打开工程文件 sln 文件

需要用 visual studio 打开工程文件（注意：visual studio 2013及以下版本需安装.net framework 4.5.2）

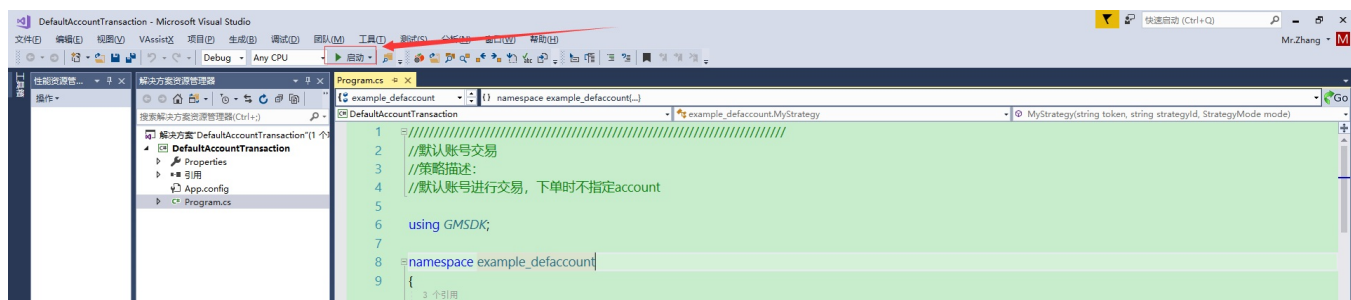
名称	修改日期	类型	大小
bin	2018/8/8 14:59	文件夹	
Properties	2018/8/8 14:35	文件夹	
App.config	2018/8/8 14:35	XML Configurati...	1 KB
DefaultAccountTransaction.csproj	2018/8/8 14:35	Visual C# 项目文件	4 KB
DefaultAccountTransaction.sln	2018/8/8 14:35	Visual Studio 解	2 KB
Program.cs	2018/8/8 14:59	Visual C# Sourc...	3 KB

- 编写策略

打开Program.cs文件，可进行策略编辑



## 编译并运行策略



- 查看运行结果

掘金客户端中关闭新建策略窗口并打开回测结果列表



## 查看回测结果

回测时间	开始日期	截止日期	累计收益率	最大回撤	夏普比率	回测进度	耗时	备注	操作
2018-08-08 14:59:07	2016-07-11	2017-07-11	-9.80%	18.66%	-0.82	100%	1,804秒		删除

## 回测相关数据指标





## 获取SDK

- 下载sdk: [点击下载](#)
  - 解压后得到:
    - example: 示例代码
    - gmsdk : C# SDK

```

1.  |—example
2.  |   ...
3.  |—gmsdk
4.  |
5.  |   |—x86
6.  |       gmsdk.dll
7.  |       gmsdk-net.dll
8.  |       protobuf-net.dll
9.  |   |—x64
10. |       ...

```

或

- NuGet方式安装SDK
  - 方式一: visual studio
    - a. 项目->管理NuGet程序包->浏览
    - b. 搜索 gmsdk-net,  选择gmsdk-net-x86,  选择gmsdk-net-x64
    - c. 选择最新版, 安装
  - 方式二: 程序包管理器控制台

32位

```
1. PM> Install-Package gmsdk-net-x86
```

64位

```
1. PM> Install-Package gmsdk-net-x64
```

注意

根据策略选择32位或64位版本 切勿混装

## 建立我们第一个策略

- 打开Visual Studio新建空白工程并新建源码文件
- 工程中引用 `gmsdk-net.dll`
- 引入命名空间：GMSDK

```
1. using GMSDK;
```

- 将 `gmsdk.dll`, `protobuf-net.dll`放到策略执行文件所在目录

## 策略应该是这样的

- 继承策略基类
- 重改关注事件
- 在OnInit里订阅行情，初始化
- 在main方法中实例化一个派生类对象
- 设置token, 策略id, 和mode
- 开始运行

## 继承策略基类

```
1.  
2. public class MyStrategy: Strategy  
3. {  
4.     public MyStrategy(string token, string strategyId, StrategyMode mode) : base(token,  
5.         strategyId, mode) {}  
6. }
```

## 重改关注事件

```
1. public class MyStrategy: Strategy  
2. {  
3.     public MyStrategy(string token, string strategyId, StrategyMode mode) : base(token,  
4.         strategyId, mode) {}  
5.     //重写OnInit事件, 进行策略开发  
6.     public override void OnInit()  
7.     {  
8.         Console.WriteLine("OnInit");  
9.         return;  
10. }
```

```
10.     }  
11. }
```

## 在OnInit里订阅行情，初始化

```
1. class MyStrategy :public Strategy  
2. {  
3.     public MyStrategy(string token, string strategyId, StrategyMode mode) : base(token,  
        strategyId, mode) {}  
4.  
5.     //重写OnInit事件，进行策略开发  
6.     public override void OnInit()  
7.     {  
8.         Console.WriteLine("OnInit");  
9.         Subscribe("SHSE.600000", "tick");  
10.        return;  
11.    }  
12. }
```

## 在main里实例化一个派生类对象

1. 获取token：打开客户端->点击右上角用户头像 -> 系统设置 -> 复制token
2. 获取策略id：打开客户端->策略研究->右上角新建策略->新建C#策略->复制策略ID
3. 策略模式：
  - o MODE\_LIVE
  - o MODE\_BACKTEST

```
1. MyStrategy s("27cbdfd8cd9b86dea554a5612baa4a8eee51af79", "536f1097-8b27-11e8-b6af-  
    94c69161828a", StrategyMode.MODE_LIVE);
```

## 开始运行

```
1. s.Run();
```

## 订阅行情策略示例

### 源文件

```
1. using GMSDK;  
2.  
3. namespace example  
4. {  
5.     public class MyStrategy : Strategy  
6.     {  
7.         public MyStrategy(string token, string strategyId, StrategyMode mode) :  
            base(token, strategyId, mode) { }  
8.     }
```

```

9.         //重写OnInit事件，进行策略开发
10.        public override void OnInit()
11.        {
12.            System.Console.WriteLine("OnInit");
13.            //订阅行情数据
14.            Subscribe("SHSE.600000", "tick");
15.            return;
16.        }
17.
18.        public override void OnTick(Tick tick)
19.        {
20.            System.Console.WriteLine("{0,-50}{1}", "代码", tick.symbol);
21.            System.Console.WriteLine("{0,-50}{1}", "时间", tick.createdAt);
22.            System.Console.WriteLine("{0,-50}{1}", "最新价", tick.price);
23.            System.Console.WriteLine("{0,-50}{1}", "开盘价", tick.open);
24.            System.Console.WriteLine("{0,-50}{1}", "最高价", tick.high);
25.            System.Console.WriteLine("{0,-50}{1}", "最低价", tick.low);
26.            System.Console.WriteLine("{0,-50}{1}", "成交总量", tick.cumVolume);
27.            System.Console.WriteLine("{0,-50}{1}", "成交总额/最新成交额, 累计值",
tick.cumAmount);
28.            System.Console.WriteLine("{0,-50}{1}", "合约持仓量(期), 累计值",
tick.cumPosition);
29.            System.Console.WriteLine("{0,-50}{1}", "瞬时成交额", tick.lastAmount);
30.            System.Console.WriteLine("{0,-50}{1}", "瞬时成交量", tick.lastVolume);
31.            System.Console.WriteLine("{0,-50}{1}", "(保留)交易类型, 对应多开, 多平等类型",
tick.tradeType);
32.            System.Console.WriteLine("{0,-50}{1}", "一档委买价",
tick.quotes[0].bidPrice);
33.            System.Console.WriteLine("{0,-50}{1}", "一档委买量",
tick.quotes[0].bidVolume);
34.            System.Console.WriteLine("{0,-50}{1}", "一档委卖价",
tick.quotes[0].askPrice);
35.            System.Console.WriteLine("{0,-50}{1}", "一档委卖量",
tick.quotes[0].askVolume);
36.        }
37.    }
38.    class Program
39.    {
40.        static void Main(string[] args)
41.        {
42.            MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
"536f1097-8b27-11e8-b6af-94c69161828a", StrategyMode.MODE_BACKTEST);
43.            s.SetBacktestConfig("2017-07-25 8:20:00", "2018-07-25 17:30:00");
44.            s.Run();
45.            System.Console.WriteLine("回测完成!");
46.            System.Console.Read();
47.        }
48.    }
49. }

```

## 典型场景

- [空策略](#)
- [定时任务](#)
- [数据事件驱动](#)
- [默认交易账号](#)
- [显示指定交易账号](#)
- [模式选择](#)
- [数据研究](#)

## 空策略

```

1. ///////////////////////////////////////////////////////////////////
2. //空策略
3. using GMSDK;
4.
5. namespace example
6. {
7.     public class MyStrategy : Strategy
8.     {
9.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
base(token, strategyId, mode) { }
10.
11.         //重写OnInit事件, 进行策略开发
12.         public override void OnInit()
13.         {
14.             System.Console.WriteLine("OnInit");
15.             //订阅行情数据
16.             Subscribe("SHSE.600000", "tick");
17.             return;
18.         }
19.
20.         public override void OnTick(Tick tick)
21.         {
22.             System.Console.WriteLine("{0,-50}{1}", "代码", tick.symbol);
23.             System.Console.WriteLine("{0,-50}{1}", "时间", tick.createdAt);
24.             System.Console.WriteLine("{0,-50}{1}", "最新价", tick.price);
25.             System.Console.WriteLine("{0,-50}{1}", "开盘价", tick.open);
26.             System.Console.WriteLine("{0,-50}{1}", "最高价", tick.high);
27.             System.Console.WriteLine("{0,-50}{1}", "最低价", tick.low);
28.             System.Console.WriteLine("{0,-50}{1}", "成交总量", tick.cumVolume);
29.             System.Console.WriteLine("{0,-50}{1}", "成交总额/最新成交额, 累计值",
tick.cumAmount);
30.             System.Console.WriteLine("{0,-50}{1}", "合约持仓量(期), 累计值",
tick.cumPosition);
31.             System.Console.WriteLine("{0,-50}{1}", "瞬时成交额", tick.lastAmount);
32.             System.Console.WriteLine("{0,-50}{1}", "瞬时成交量", tick.lastVolume);
33.             System.Console.WriteLine("{0,-50}{1}", "(保留)交易类型, 对应多开, 多平等类型",
tick.tradeType);
34.             System.Console.WriteLine("{0,-50}{1}", "一档委买价",
tick.quotes[0].bidPrice);

```

```

35.         System.Console.WriteLine("{0,-50}{1}", "一档委买量",
tick.quotes[0].bidVolume);
36.         System.Console.WriteLine("{0,-50}{1}", "一档委卖价",
tick.quotes[0].askPrice);
37.         System.Console.WriteLine("{0,-50}{1}", "一档委卖量",
tick.quotes[0].askVolume);
38.     }
39. }
40. class Program
41. {
42.     static void Main(string[] args)
43.     {
44.         MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
"536f1097-8b27-11e8-b6af-94c69161828a", StrategyMode.MODE_BACKTEST);
45.         s.SetBacktestConfig("2017-07-25 8:20:00", "2018-07-25 17:30:00");
46.         s.Run();
47.         System.Console.WriteLine("回测完成!");
48.         System.Console.Read();
49.     }
50. }
51. }

```

## 定时任务

```

1.  //////////////////////////////////////
2. //定时任务
3. //策略描述：
4.     典型如选股交易。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。
5.
6. using GMSDK;
7.
8. namespace example_schedule
9. {
10.     public class MyStrategy : Strategy
11.     {
12.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
base(token, strategyId, mode) { }
13.
14.         //重写OnInit事件，进行策略开发
15.         public override void OnInit()
16.         {
17.             System.Console.WriteLine("OnInit");
18.
19.             //设置定时任务
20.             Schedule("1d", "13:24:00");
21.             return;
22.         }
23.
24.         //定时任务触发事件
25.         public override void OnSchedule(string data_rule, string timeRule)
26.         {
27.             //购买200股浦发银行股票

```

```

28.         GMData<Order> o = OrderValue("SHSE.600000", 200, OrderSide.OrderSide_Buy,
OrderType.OrderType_Market, PositionEffect.PositionEffect_Open, 0);
29.         if (o.status == 0)    //该判断仅表示函数调用无异常
30.         {
31.             //
32.         }
33.     }
34.
35.     //回测完成后收到绩效报告
36.     public override void OnBacktestFinished(Indicator indicator)
37.     {
38.         System.Console.WriteLine("OnBacktestFinished:");
39.         System.Console.WriteLine("{0,-50}{1}", "账号ID:", indicator.accountId);
40.         System.Console.WriteLine("{0,-50}{1}", "累计收益率:", indicator.pnlRatio);
41.         System.Console.WriteLine("{0,-50}{1}", "年化收益率:",
indicator.pnlRatioAnnual);
42.         System.Console.WriteLine("{0,-50}{1}", "夏普比率:", indicator.sharpRatio);
43.         System.Console.WriteLine("{0,-50}{1}", "最大回撤:", indicator.maxDrawdown);
44.         System.Console.WriteLine("{0,-50}{1}", "风险比率:", indicator.riskRatio);
45.         System.Console.WriteLine("{0,-50}{1}", "开仓次数:", indicator.openCount);
46.         System.Console.WriteLine("{0,-50}{1}", "平仓次数:", indicator.closeCount);
47.         System.Console.WriteLine("{0,-50}{1}", "盈利次数:", indicator.winCount);
48.         System.Console.WriteLine("{0,-50}{1}", "亏损次数:", indicator.loseCount);
49.         System.Console.WriteLine("{0,-50}{1}", "胜率:", indicator.winRatio);
50.         System.Console.WriteLine("{0,-50}{1}", "指标创建时间:",
indicator.createdAt);
51.         System.Console.WriteLine("{0,-50}{1}", "指标变更时间:",
indicator.updatedAt);
52.     }
53. }
54. class Program
55. {
56.     static void Main(string[] args)
57.     {
58.         MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
"536f1097 - 8b27 - 11e8 - b6af - 94c69161828a", StrategyMode.MODE_BACKTEST);
59.         s.SetBacktestConfig("2016-07-11 17:20:00", "2017-07-11 17:30:00");
60.         s.Run();
61.         System.Console.WriteLine("回测完成!");
62.         System.Console.Read();
63.     }
64. }
65. }

```

## 数据事件驱动

```

1.  //////////////////////////////////////
2.  //数据事件驱动
3.  //策略描述：
4.  //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
5.
6.  using GMSDK;

```

```

7.
8. namespace example_dataevent
9. {
10.     public class MyStrategy : Strategy
11.     {
12.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
13.         base(token, strategyId, mode) { }
14.
15.         //重写OnInit事件, 进行策略开发
16.         public override void OnInit()
17.         {
18.             System.Console.WriteLine("OnInit");
19.
20.             //订阅浦发银行, bar频率为一天
21.             Subscribe("SHSE.600000", "1d");
22.             return;
23.         }
24.
25.         //重写OnBar事件
26.         public override void OnBar(Bar bar)
27.         {
28.             System.Console.WriteLine("OnBar :");
29.             System.Console.WriteLine("{0, -50}{1}", "代码:", bar.symbol);
30.             System.Console.WriteLine("{0, -50}{1}", "bar的开始时间:", bar.bob);
31.             System.Console.WriteLine("{0, -50}{1}", "bar的结束时间:", bar.eob);
32.             System.Console.WriteLine("{0, -50}{1}", "开盘价:", bar.open);
33.             System.Console.WriteLine("{0, -50}{1}", "收盘价:", bar.close);
34.             System.Console.WriteLine("{0, -50}{1}", "最高价:", bar.high);
35.             System.Console.WriteLine("{0, -50}{1}", "最低价:", bar.low);
36.             System.Console.WriteLine("{0, -50}{1}", "成交量:", bar.volume);
37.             System.Console.WriteLine("{0, -50}{1}", "成交金额:", bar.amount);
38.             System.Console.WriteLine("{0, -50}{1}", "前收盘价:", bar.preClose);
39.             System.Console.WriteLine("{0, -50}{1}", "持仓量:", bar.position);
40.             System.Console.WriteLine("{0, -50}{1}", "bar频度:", bar.frequency);
41.         }
42.     }
43.
44.     class Program
45.     {
46.         static void Main(string[] args)
47.         {
48.             MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
49.             "536f1097 - 8b27 - 11e8 - b6af - 94c69161828a", StrategyMode.MODE_BACKTEST);
50.             s.SetBacktestConfig("2016-07-11 17:20:00", "2017-07-11 17:30:00");
51.             s.Run();
52.             System.Console.WriteLine("回测完成!");
53.             System.Console.Read();
54.         }
55.     }
56. }

```

## 默认交易账号



```

1. ///////////////////////////////////////////////////////////////////
2. //默认账号交易
3. //策略描述：
4. //默认账号进行交易，下单时不指定account
5.
6. using GMSDK;
7.
8. namespace example_defaccount
9. {
10.     public class MyStrategy : Strategy
11.     {
12.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
13.         base(token, strategyId, mode) { }
14.
15.         //重写OnInit事件，进行策略开发
16.         public override void OnInit()
17.         {
18.             System.Console.WriteLine("OnInit");
19.             Subscribe("SHSE.600000,SZSE.000001", "1d");
20.
21.             return;
22.         }
23.
24.         //重写OnBar事件
25.         public override void OnBar(Bar bar)
26.         {
27.             //不指定account 使用默认账户下单
28.             OrderVolume(bar.symbol, 200, OrderSide.OrderSide_Buy,
29.             OrderType.OrderType_Market, PositionEffect.PositionEffect_Open, 0);
30.         }
31.
32.         //回测完成后收到绩效报告
33.         public override void OnBacktestFinished(Indicator indicator)
34.         {
35.             System.Console.WriteLine("OnBacktestFinished:");
36.             System.Console.WriteLine("{0,-50}{1}", "账号ID:", indicator.accountId);
37.             System.Console.WriteLine("{0,-50}{1}", "累计收益率:", indicator.pnlRatio);
38.             System.Console.WriteLine("{0,-50}{1}", "年化收益率:",
39.             indicator.pnlRatioAnnual);
40.             System.Console.WriteLine("{0,-50}{1}", "夏普比率:", indicator.sharpRatio);
41.             System.Console.WriteLine("{0,-50}{1}", "最大回撤:", indicator.maxDrawdown);
42.             System.Console.WriteLine("{0,-50}{1}", "风险比率:", indicator.riskRatio);
43.             System.Console.WriteLine("{0,-50}{1}", "开仓次数:", indicator.openCount);
44.             System.Console.WriteLine("{0,-50}{1}", "平仓次数:", indicator.closeCount);
45.             System.Console.WriteLine("{0,-50}{1}", "盈利次数:", indicator.winCount);
46.             System.Console.WriteLine("{0,-50}{1}", "亏损次数:", indicator.loseCount);
47.             System.Console.WriteLine("{0,-50}{1}", "胜率:", indicator.winRatio);
48.             System.Console.WriteLine("{0,-50}{1}", "指标创建时间:",
49.             indicator.createdAt);
50.             System.Console.WriteLine("{0,-50}{1}", "指标变更时间:",
51.             indicator.updatedAt);
52.         }
53.     }
54. }
55. class Program

```

```

50.     {
51.         static void Main(string[] args)
52.         {
53.             MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
54. "536f1097 - 8b27 - 11e8 - b6af - 94c69161828a", StrategyMode.MODE_BACKTEST);
55.             s.SetBacktestConfig("2016-07-11 17:20:00", "2017-07-11 17:30:00");
56.             s.Run();
57.             System.Console.WriteLine("回测完成!");
58.             System.Console.Read();
59.         }
60.     }

```

## 显示指定交易账号

```

1.  //////////////////////////////////////
2.  //显示指定交易账号
3.  //策略描述：
4.  //下单时指定交易账号，account参数传账号id或者账号标题
5.
6.
7.  using GMSDK;
8.
9.  namespace example_defaccount
10. {
11.     public class MyStrategy : Strategy
12.     {
13.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
14. base(token, strategyId, mode) { }
15.
16.         //重写OnInit事件，进行策略开发
17.         public override void OnInit()
18.         {
19.             System.Console.WriteLine("OnInit");
20.             Subscribe("SHSE.600000,SZSE.000001", "1d");
21.
22.             return;
23.         }
24.
25.         //重写OnBar事件
26.         public override void OnBar(Bar bar)
27.         {
28.             //不指定account 使用默认账户下单
29.             OrderVolume(bar.symbol, 200, OrderSide.OrderSide_Buy,
30. OrderType.OrderType_Market, PositionEffect.PositionEffect_Open, 0, ba8785aa-8641-11e8-
31. 98cb-7085c223669d);
32.         }
33.
34.         //回测完成后收到绩效报告
35.         public override void OnBacktestFinished(Indicator indicator)
36.         {
37.             System.Console.WriteLine("OnIndicator:");
38.             System.Console.WriteLine("{0,-50}{1}", "账号ID:", indicator.accountId);

```

```

36.         System.Console.WriteLine("{0,-50}{1}", "累计收益率:", indicator.pnlRatio);
37.         System.Console.WriteLine("{0,-50}{1}", "年化收益率:",
indicator.pnlRatioAnnual);
38.         System.Console.WriteLine("{0,-50}{1}", "夏普比率:", indicator.sharpRatio);
39.         System.Console.WriteLine("{0,-50}{1}", "最大回撤:", indicator.maxDrawdown);
40.         System.Console.WriteLine("{0,-50}{1}", "风险比率:", indicator.riskRatio);
41.         System.Console.WriteLine("{0,-50}{1}", "开仓次数:", indicator.openCount);
42.         System.Console.WriteLine("{0,-50}{1}", "平仓次数:", indicator.closeCount);
43.         System.Console.WriteLine("{0,-50}{1}", "盈利次数:", indicator.winCount);
44.         System.Console.WriteLine("{0,-50}{1}", "亏损次数:", indicator.loseCount);
45.         System.Console.WriteLine("{0,-50}{1}", "胜率:", indicator.winRatio);
46.         System.Console.WriteLine("{0,-50}{1}", "指标创建时间:",
indicator.createdAt);
47.         System.Console.WriteLine("{0,-50}{1}", "指标变更时间:",
indicator.updatedAt);
48.     }
49. }
50. class Program
51. {
52.     static void Main(string[] args)
53.     {
54.         MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
"536f1097 - 8b27 - 11e8 - b6af - 94c69161828a", StrategyMode.MODE_BACKTEST);
55.         s.SetBacktestConfig("2016-07-11 17:20:00", "2017-07-11 17:30:00");
56.         s.Run();
57.         System.Console.WriteLine("回测完成!");
58.         System.Console.Read();
59.     }
60. }
61. }

```

## 模式选择

```

1.  //////////////////////////////////////
2.  //模式选择
3.  //策略描述:
4.  //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, 实例化策略参数
mode=MODE_BACKTEST 表示回测模式, mode=MODE_LIVE表示实时模式
5.
6.  using GMSDK;
7.
8.  namespace example
9.  {
10.     public class MyStrategy : Strategy
11.     {
12.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
base(token, strategyId, mode) { }
13.
14.         //重写OnInit事件, 进行策略开发
15.         public override void OnInit()
16.         {
17.             System.Console.WriteLine("OnInit");

```

```

18.         //订阅行情数据
19.         Subscribe("SHSE.600000", "tick");
20.         return;
21.     }
22.
23.     public override void OnTick(Tick tick)
24.     {
25.         System.Console.WriteLine("{0,-50}{1}", "代码", tick.symbol);
26.         System.Console.WriteLine("{0,-50}{1}", "时间", tick.createdAt);
27.         System.Console.WriteLine("{0,-50}{1}", "最新价", tick.price);
28.         System.Console.WriteLine("{0,-50}{1}", "开盘价", tick.open);
29.         System.Console.WriteLine("{0,-50}{1}", "最高价", tick.high);
30.         System.Console.WriteLine("{0,-50}{1}", "最低价", tick.low);
31.         System.Console.WriteLine("{0,-50}{1}", "成交总量", tick.cumVolume);
32.         System.Console.WriteLine("{0,-50}{1}", "成交总额/最新成交额, 累计值",
tick.cumAmount);
33.         System.Console.WriteLine("{0,-50}{1}", "合约持仓量(期), 累计值",
tick.cumPosition);
34.         System.Console.WriteLine("{0,-50}{1}", "瞬时成交额", tick.lastAmount);
35.         System.Console.WriteLine("{0,-50}{1}", "瞬时成交量", tick.lastVolume);
36.         System.Console.WriteLine("{0,-50}{1}", "(保留)交易类型, 对应多开, 多平等类型",
tick.tradeType);
37.         System.Console.WriteLine("{0,-50}{1}", "一档委买价",
tick.quotes[0].bidPrice);
38.         System.Console.WriteLine("{0,-50}{1}", "一档委买量",
tick.quotes[0].bidVolume);
39.         System.Console.WriteLine("{0,-50}{1}", "一档委卖价",
tick.quotes[0].askPrice);
40.         System.Console.WriteLine("{0,-50}{1}", "一档委卖量",
tick.quotes[0].askVolume);
41.     }
42. }
43. class Program
44. {
45.     static void Main(string[] args)
46.     {
47.         MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",
"536f1097-8b27-11e8-b6af-94c69161828a", StrategyMode.MODE_LIVE);
48.         s.Run();
49.         System.Console.Read();
50.     }
51. }
52. }

```

## 数据研究

```

1. //////////////////////////////////////
2. //数据研究
3. //策略描述：
4. //无需实时数据驱动策略，无需交易下单，只是取数据的场景
5.
6. using GMSDK;

```

```

7.
8. namespace example_datares
9. {
10.     public class MyStrategy : Strategy
11.     {
12.         public MyStrategy(string token, string strategyId, StrategyMode mode) :
13.         base(token, strategyId, mode) { }
14.
15.         //重写OnInit事件, 进行策略开发
16.         public override void OnInit()
17.         {
18.             System.Console.WriteLine("OnInit");
19.
20.             GMDDataList<Tick> ht = GMApi.HistoryTicks("SZSE.000002", "2017-07-11
21.             10:20:00", "2017-07-11 10:30:00");
22.             if (ht.status == 0)
23.             {
24.                 foreach (var tick in ht.data)
25.                 {
26.                     System.Console.WriteLine("{0,-50}{1}", "代码", tick.symbol);
27.                     System.Console.WriteLine("{0,-50}{1}", "时间", tick.createdAt);
28.                     System.Console.WriteLine("{0,-50}{1}", "最新价", tick.price);
29.                     System.Console.WriteLine("{0,-50}{1}", "开盘价", tick.open);
30.                     System.Console.WriteLine("{0,-50}{1}", "最高价", tick.high);
31.                     System.Console.WriteLine("{0,-50}{1}", "最低价", tick.low);
32.                     System.Console.WriteLine("{0,-50}{1}", "成交总量", tick.cumVolume);
33.                     System.Console.WriteLine("{0,-50}{1}", "成交总额/最新成交额, 累计值",
34.                     tick.cumAmount);
35.                     System.Console.WriteLine("{0,-50}{1}", "合约持仓量(期), 累计值",
36.                     tick.cumPosition);
37.                     System.Console.WriteLine("{0,-50}{1}", "瞬时成交额",
38.                     tick.lastAmount);
39.                     System.Console.WriteLine("{0,-50}{1}", "瞬时成交量",
40.                     tick.lastVolume);
41.                     System.Console.WriteLine("{0,-50}{1}", "(保留)交易类型, 对应多开, 多平
42.                     等类型", tick.tradeType);
43.                     System.Console.WriteLine("{0,-50}{1}", "一档委买价",
44.                     tick.quotes[0].bidPrice);
45.                     System.Console.WriteLine("{0,-50}{1}", "一档委买量",
46.                     tick.quotes[0].bidVolume);
47.                     System.Console.WriteLine("{0,-50}{1}", "一档委卖价",
48.                     tick.quotes[0].askPrice);
49.                     System.Console.WriteLine("{0,-50}{1}", "一档委卖量",
50.                     tick.quotes[0].askVolume);
51.                 }
52.             }
53.             return;
54.         }
55.     }
56.     class Program
57.     {
58.         static void Main(string[] args)
59.         {
60.             MyStrategy s = new MyStrategy("27cbdfd8cd9b86dea554a5612baa4a8eee51af79",

```

```
    "536f1097-8b27-11e8-b6af-94c69161828a", StrategyMode.MODE_BACKTEST);  
50.         s.SetBacktestConfig("2017-07-25 8:20:00", "2018-07-25 17:30:00");  
51.         s.Run();  
52.         System.Console.Read();  
53.     }  
54. }  
55. }
```

# 重要概念

- [symbol](#) - 代码标识
  - [交易所代码](#)
  - [交易标的代码](#)
- [策略运行模式](#)
  - [实时模式](#)
  - [回测模式](#)

## symbol - 代码标识

掘金代码 (**symbol**) 是掘金平台用于唯一标识交易标的的代码，

格式为：交易所代码.交易标代码， 比如深圳平安的**symbol** 示例：`SZSE.000001`

### 交易所代码

目前掘金支持国内的7个交易所， 各交易所的代码缩写如下：

市场中文名	市场代码
上交所	SHSE
深交所	SZSE
中金所	CFFEX
上期所	SHFE
大商所	DCE
郑商所	CZCE
上海国际能源交易中心	INE

### 交易标的代码

交易表代码是指交易所给出的交易标的的代码， 包括股票， 期货， 期权， 指数， 基金等代码。

具体的代码请参考交易所的给出的证券代码定义

## 策略运行模式

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。

### 实时模式

订阅行情服务器推送的实时行情，也就是交易所的实时行情，只在交易时段提供。

### 回测模式

订阅指定时段、指定交易代码、指定数据类型的行情，行情服务器将按指定条件全速回放对应的行情数据。适用的场景是策略回测阶段，快速验证策略的绩效是否符合预期。

# 基类原型

- [策略类简介](#)
- [策略类定义](#)

## 策略类简介

策略类集成了行情、交易和事件的接口，用户的策略都从此类继承实现自己的业务逻辑。每个进程只能实例化一个策略类对象。

## 策略类定义

```

1. public class Strategy
2. {
3.     //策略基类构造函数
4.     //token :
5.     //strategy_id : 策略ID
6.     //mode : 运行模式
7.     public Strategy(string token, string strategyId, int mode);
8.
9.
10. //=====策略参数类函数
=====
11.     //添加参数
12.     public int AddParameters(Parameter param);
13.
14.     //删除参数
15.     public int DelParameters(string keys);
16.
17.     //获取标的
18.     public GMDaDataList<string> GetSymbols();
19.
20.     //设置标的
21.     public int SetSymbols(string symbols);
22.
23.     //设置回测参数
24.     public int SetBacktestConfig(string startTime, string endTime, double
initialCash = 10000000, double transactionRatio = 1, double commissionRatio = 0, double
slippageRatio = 0, int adjust = 0, int checkCache = 1);
25.
26.     //设置参数
27.     public int SetParameters(List<Parameter> parameters);
28.
29.
30. //=====交易函数
=====
31.     //查询交易账号
32.     public GMDaDataList<Account> GetAccounts();
33.
34.     //查询资金
35.     public GMDaDataList<Cash> GetCash(string account = null);
36.

```



```

37.      //查询成交
38.      public GMDaDataList<ExecRpt> GetExecutionReports(string account = null);
39.
40.      //查询委托
41.      public GMDaDataList<Order> GetOrders(string account = null);
42.
43.      //查询持仓
44.      public GMDaDataList<Position> GetPosition(string account = null);
45.
46.      //查询未结委托
47.      public GMDaDataList<Order> GetUnfinishedOrders(string account = null);
48.
49.      //委托撤单,
50.      public int OrderCancel(string clOrdIds);
51.
52.      //撤销所有委托
53.      public void OrderCancelAll();
54.
55.      //平当前所有可平持仓
56.      public GMDaDataList<Order> OrderCloseAll();
57.
58.      //按总资产指定比例委托
59.      public GMDaData<Order> OrderPercent(string symbol, double percent, int side, int
orderType, int positionEffect, double price = 0, string account = null);
60.
61.      //调仓到目标持仓比例（总资产的比例）
62.      public GMDaData<Order> OrderTargetPercent(string symbol, double percent, int
positionSide, int orderType, double price = 0, string account = null);
63.
64.      //调仓到目标持仓额
65.      public GMDaData<Order> OrderTargetValue(string symbol, double value, int
positionSide, int orderType, double price = 0, string account = null);
66.
67.      //调仓到目标持仓量
68.      public GMDaData<Order> OrderTargetVolume(string symbol, int volume, int positionSide,
int orderType, double price = 0, string account = null);
69.
70.      //按指定价值委托
71.      public GMDaData<Order> OrderValue(string symbol, double value, int side, int
orderType, int positionEffect, double price = 0, string account = null);
72.
73.      //按指定量委托
74.      public GMDaData<Order> OrderVolume(string symbol, int volume, OrderSide side,
OrderType orderType, PositionEffect positionEffect, double price = 0, string account =
null);
75.
76.
77.      //=====基础函数=====
78.      //运行策略
79.      public int Run();
80.
81.      //定时任务
82.      public int Schedule(string dataRule, string timeRule);
83.
84.      //停止策略
85.      public int Stop();

```

```

86.
87.     //当前事件
88.     public long Now();
89.
90.     //设置token
91.     public int SetToken(string token);
92.
93.     //设置运行模式
94.     public int SetMode(StrategyMode mode);
95.
96.     //设置策略ID
97.     public int SetStrategyId(string strategyId);
98.
99.     //查询指定账户状态
100.    public AccountStatus GetAccountStatus(string accountId);
101.
102.    //查询所有账户状态
103.    public List<AccountStatus> GetAccountStatus()
104.
105.    //=====数据函数
=====
106.    //订阅行情
107.    public int Subscribe(string symbols, string frequency, bool unsubscribePrevious =
false);
108.
109.    //退订行情
110.    public int Unsubscribe(string symbols, string frequency);
111.
112.
113.    //=====事件函数
=====
114.    //初始化完成
115.    public virtual void OnInit();
116.
117.    //实盘账号状态变化
118.    public virtual void OnAccountStatus(AccountStatus accountStatus);
119.
120.    //收到bar行情
121.    public virtual void OnBar(Bar bar);
122.
123.    //cash发生变化
124.    public virtual void OnCashStatus(Cash order);
125.
126.    //错误事件
127.    public virtual void OnError(int errorCode, string errorMsg);
128.
129.    //执行回报
130.    public virtual void OnExecutionReport(ExecRpt rpt);
131.
132.    //回测结束
133.    public virtual void OnBacktestFinished(Indicator indicator);
134.
135.    //数据库已连接
136.    public virtual void OnMarketDataConnected();
137.
138.    //数据库断开
139.    public virtual void OnMarketDataDisconnected();

```

```
140.  
141.    //委托发生变化  
142.    public virtual void OnOrderStatus(Order order);  
143.  
144.    //运行时参数发生变化  
145.    public virtual void OnParameter(List<Parameter> param);  
146.  
147.    //position发生变化  
148.    public virtual void OnPosition(Position position);  
149.  
150.    //定时任务触发  
151.    public virtual void OnSchedule(string dataRule, string timeRule);  
152.  
153.    //策略结束  
154.    public virtual void OnStop();  
155.  
156.    //收到tick行情  
157.    public virtual void OnTick(Tick tick);  
158.  
159.    //交易已连接  
160.    public virtual void OnTradeDataConnected();  
161.  
162.    //交易断开  
163.    public virtual void OnTradeDataDisconnected();  
164. }
```

# 基本成员函数

- [Strategy](#) - 构造函数
- [Run](#) - 运行策略
- [Stop](#) - 停止策略
- [SetToken](#) - 设置用户token
- [SetMode](#) - 设置运行模式
- [SetStrategyId](#) - 设置策略ID
- [GetAccountStatus](#) - 获取策略所有账户状态
- [GetAccountStatus](#) - 获取指定账户状态
- [Schedule](#) - 预设定时任务
- [Now](#) - 获取当前时间
- [SetBacktestConfig](#) - 设置回测参数

## Strategy - 构造函数

构造策略对象。

函数原型：

```
1. Strategy();
2. Strategy(string token, string strategyId, StrategyMode mode);
```

参数：

参数名	类型	说明
token	string	系统权限密钥, 可在终端系统设置-密钥管理中生成
strategyId	string	策略ID, 在终端中获取
mode	StrategyMode	策略模式, 参见 <code>enum StrategyMode</code>

注意事项：

- 一个进程只能构造一个策略对象。

## Run - 运行策略

运行策略。只有调用Run后，才会驱动所有的事件，如行情接入与交易事件。

函数原型：

```
1. int Run();
```

参数：

参数名	类型	说明
返回值	int	如果策略正常退出返回0， 非正常退出返回错误码

注意事项：

调用Run会阻塞线程，策略进入事件驱动状态，所以所有初始操作（如读配置文件，分配缓冲区等）都应该在Run之前完成，如果run退出，意味着策略运行结束，整个进程应该就此退出。

## Stop - 停止策略

用于停止策略，也就是如果调用Run()之后，在某个事件响应中调用Stop，这是Run就是退出，并返回0。

函数原型：

```
1. void Stop();
```

## SetToken - 设置用户token

函数原型：

```
1. int SetToken(string token)
```

参数：

参数名	类型	说明
token	string	系统权限密钥, 可在终端系统设置-密钥管理中生成

注意事项：

不管是从构造函数传入还成员函数传入，`token`，`strategyId`，`mode` 都是必须要设置的参数。

## SetMode - 设置运行模式

函数原型

```
1. int SetMode(StrategyMode mode)
```

参数：

参数名	类型	说明
mode	StrategyMode	策略运行模式，参见 <code>StrategyMode</code>

注意事项：

不管是从构造函数传入还成员函数传入，`token`，`strategyId`，`mode` 都是必须要设置的参数

## SetStrategyId - 设置策略ID

函数原型：

```
1. int SetStrategyId(string strategyId)
```

注意事项：

不管是从构造函数传入还成员函数传入，`token`，`strategyId`，`mode` 都是必须要设置的参数

参数:

参数名	类型	说明
strategyId	string	策略ID, 在终端中获取

## GetAccountStatus - 获取策略所有账户状态

函数原型:

```
1. List<AccountStatus> GetAccountStatus()
```

参数:

参数名	类型	说明
返回值	List	AccountStatus 列表

示例:

```
1. //获取当策略所有账户状态
2. var status_l = GetAccountStatus();
3. //遍历账户
4. foreach (var status in status_l)
5. {
6.     //打印 AccountStatus 字段
7.     System.Console.WriteLine("accountId: {0}, accountName: {1}, state: {2}, errorCode: {3}, errorMsg: {4}", status.accountId, status.accountName, status.state, status.errorCode, status.errorMsg);
8. }
```

## GetAccountStatus - 获取指定账户状态

函数原型:

```
1. AccountStatus GetAccountStatus(string accountId)
```

参数:

参数名	类型	说明
返回值	AccountStatus	账户状态结构

## Schedule - 预设定时任务

在指定时间自动执行策略算法，通常用于选股类型策略。Schedule一般在OnInit中调用。如果Schedule预设成功，那么达成预设时间条件时，OnSchedule会被调用，并在OnSchedule的参数中返回设置的 `dataRule` 和 `timeRule` 。Schedule可以调用多次，设置多个不同定时任务。

函数原型:

```
1. int Schedule(string dataRule, string timeRule);
```

参数：

参数名	类型	说明
dataRule	string	n + 时间单位， 可选'd/w/m' 表示n天/n周/n月
timeRule	string	执行算法的具体时间（%H:%M:%S 格式）
返回值	int	预设成功返回0， 预设失败返回错误码

示例：

```
1.
2. #每天的19:06:20执行
3. Schedule(dateRule="1d", timeRule="19:06:20")
4.
5. #每月的第一个交易日的09:40:00执行
6. Schedule(dateRule="1m", time_rule="9:40:00")
```

注意事项：

- 现在 dataRule 暂只支持 1d, 1w, 1m， 任意n后续会支持。
- 1w, 1m 只在回测中支持，实盘模式中不支持。

## Now - 获取当前时间

函数原型：

```
1. DateTime Now();
```

参数：

参数名	类型	说明
返回值	DateTime	当前时间

注意事项：

- 实时模式下，返回当前的系统时间。回测模式下，返回当前的回测时间点。格式是DateTime。

## SetBacktestConfig - 设置回测参数

如果mode设置为回测模式，则在调用Run之前，需要先设置本函数设置回测参数。在实时模式下，该调用被忽略。

函数原型：

```
1. int SetBacktestConfig(
2.     string startTime,
3.     string endTime,
4.     double initialCash = 1000000,
5.     double transactionRatio = 1,
```

```
6.     double commissionRatio = 0,
7.     double slippageRatio = 0,
8.     Adjust    adjust = 0,
9.     int      checkCache = 1
10. );
```

参数：

参数名	类型	说明
startTime	string	回测开始时间 (%Y-%m-%d %H:%M:%S格式)
endTime	string	回测结束时间 (%Y-%m-%d %H:%M:%S格式)
initialCash	double	回测初始资金，默认1000000
transactionRatio	double	回测成交比例，默认1.0，即下单100%成交
commissionRatio	double	回测佣金比例，默认0
slippageRatio	double	回测滑点比例，默认0
adjust	Adjust	复权方式，参见 <code>enum Adjust</code>
checkCache	int	回测是否使用缓存：1 - 使用， 0 - 不使用；默认使用

注意：

startTime和endTime中月,日,时,分,秒均可以只输入个位数，例： "2016-6-7 9:55:0" 或 "2017-8-1 14:6:0" ，但若对应位置为零，则0不可被省略,比如不能输入 "2017-8-1 14:6: "



# 行情成员函数

- [subscribe](#) - 订阅行情
- [unsubscribe](#) - 退订行情

## subscribe - 订阅行情

订阅行情推送，实时模式下订阅实时行情推送，回测模式下订阅历史行情推送。订阅tick会触发OnTick回调，订阅bar则触发OnBar回调。

函数原型：

```
1. int Subscribe(string symbols, string frequency, bool unsubscribePrevious = false);
```

参数：

参数名	类型	说明
symbols	string	订阅标的代码列表，字符串格式，如有多个代码，中间用 , （英文逗号）隔开
frequency	string	频率，支持 “tick”，“1d”，“15s”，“30s” 等
unsubscribePrevious	bool	是否取消过去订阅的symbols，默认false不取消，输入true则取消所有原来的订阅。
返回值	int	订阅成功返回0， 订阅失败返回错误码

示例：

```
1. //订阅 SHSE.600000和 SZSE.000001 两个标的的tick行情
2. Subscribe(symbols="SHSE.600000,SHSE.600004", frequency="tick");
3.
4. //订阅 SHSE.600000和 SZSE.000001 两个标的的1分钟bar
5. Subscribe(symbols="SHSE.600000,SHSE.600004", frequency="60s");
```

## unsubscribe - 退订行情

退订已经订阅行情推送， 与Subscribe作用相返。

函数原型：

```
1. int unsubscribe(string symbols, string frequency);
```

参数：

参数名	类型	说明
symbols	string	退订标的代码列表，字符串格式，如有多个代码，中间用 , （英文逗号）隔开
frequency	string	频率，支持 “tick”，“1d”，“15s”，“30s” 等
返回值	int	退订成功返回0， 退订失败返回错误码

示例：

1. //退订 SHSE.600000和 SZSE.000001 两个标的的tick行情
2. `Unsubscribe(symbols="SHSE.600000,SHSE.600004", frequency="tick");`

## 普通交易成员函数

- [GetAccounts](#) - 查询交易账号
- [PlaceOrder](#) - 按指定量委托
- [OrderVolume](#) - 按指定量委托
- [OrderValue](#) - 按指定价值委托
- [OrderPercent](#) - 按总资产指定比例委托
- [OrderTargetVolume](#) - 调仓到目标持仓量
- [OrderTargetValue](#) - 调仓到目标持仓额
- [OrderTargetPercent](#) - 调仓到目标持仓比例（总资产的比例）
- [OrderCloseAll](#) - 平当前所有可平持仓
- [OrderCancel](#) - 委托撤单
- [OrderCancelAll](#) - 撤销所有委托
- [GetOrders](#) - 查询所有委托
- [GetUnfinishedOrders](#) - 查询未结委托
- [GetExecutionReports](#) - 查询成交
- [GetCash](#) - 查询资金
- [GetPosition](#) - 查询持仓

### GetAccounts - 查询交易账号

用于查询交易账号配置信息。多用于实盘时，策略同时关联多个交易账号的时候，获取所有交易账号的信息，所返回的账号id( `accountId` )用于后续各个交易api的入参，即指定操作某个交易账户。  
如果关联的交易账号只有一个，一般用不到此函数。

函数原型：

```
1. GMDDataList<Account> GetAccounts();
```

参数：

参数名	类型	说明
返回值	GMDDataList<Account>	一个GMDDataList结构

### PlaceOrder - 按指定量委托

按指定量委托，如果调用成功，后续委托单状态变化将会触发on\_order\_status回调。

函数原型：

```
1. GMDData<Order> PlaceOrder(PlaceOrderReq oreq)
```

参数：

参数名	类型	说明
oreq	PlaceOrderReq	下单参数结构

注意：

- 1. 仅支持一个标的代码，若交易代码输入有误，终端会拒绝此单，并显示 委托代码不正确 。
- 2. 若下单数量输入有误，终端会拒绝此单，并显示 委托量不正确 。股票买入最小单位为 100 ，卖出最小单位为 1 ，如存在不足100股的持仓一次性卖出；期货买卖最小单位为 1 ， 向下取整 。
- 3. 若仓位不足，终端会拒绝此单，显示 仓位不足 。平仓时股票默认 平昨仓 ，期货默认 平今仓 。应研究需要， 股票也支持卖空操作 。
- 4. OrderType优先级高于price, 若指定OrderType\_Market下市价单，使用价格为最新一个tick中的最新价，price参数失效。则price参数失效。若OrderType\_Limit限价单，仿真模式价格错误，终端拒绝此单，显示委托价格错误， 回测模式下对价格无限制 。
- 5. 函数调用成功并不意味着委托已经成功，只是意味委托单已经成功发出去， 委托是否成功根据onOrderStatus，或GetOrder来判断。

## OrderVolume - 按指定量委托

按指定量委托， 如果调用成功，后续委托单状态变化将会触发on\_order\_status回调。

函数原型：

```
1. GMDData<Order> OrderVolume(string symbol, int volume, OrderSide side, OrderType orderType, PositionEffect positionEffect, double price = 0, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
side	OrderSide	委托方向 参见 enum OrderSide
orderType	OrderType	委托类型 参见 enum OrderType
positionEffect	PositionEffect	开平类型 参见 enum PositionSide
price	double	委托价格
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 null
返回值	Order	一个Order结构， 如果函数调用失败， Order.status 值为 OrderStatus_Rejected , Order.ordRejReasonDetail 为错误原因描述， 其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识， 可用于追溯订单状态或撤单

示例：

```
1. //以11块的价格限价买入10000股浦发银行
2. GMDData<Order> o = orderVolume("SHSE.600000", 10000, OrderSide.OrderSide_Buy, OrderType.OrderType_Limit, PositionEffect.PositionEffect_Open, 11);
```

注意：

- 1. 仅支持一个标的代码，若交易代码输入有误，终端会拒绝此单，并显示 委托代码不正确 。

- 2.若下单数量输入有误，终端会拒绝此单，并显示 委托量不正确。股票买入最小单位为 100，卖出最小单位为 1，如存在不足100股的持仓一次性卖出;期货买卖最小单位为 1，向下取整。
- 3.若仓位不足，终端会拒绝此单，显示 仓位不足。平仓时股票默认 平昨仓，期货默认 平今仓。应研究需要，股票也支持卖空操作。
- 4.OrderType优先级高于price,若指定OrderType\_Market下市价单，使用价格为最新一个tick中的最新价，price参数失效。则price参数失效。若OrderType\_Limit限价单，仿真模式价格错误，终端拒绝此单，显示委托价格错误，回测模式下对价格无限制。
- 5.函数调用成功并不意味着委托已经成功，只是意味委托单已经成功发出去，委托是否成功根据OnOrderStatus，或GetOrder来判断。

## OrderValue - 按指定价值委托

按指定价值委托，如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调。

函数原型：

```
1. GMDData<Order> OrderValue(string symbol, double value, OrderSide side, OrderType
    orderType, PositionEffect positionEffect, double price = 0, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
value	int	股票价值
side	OrderSide	委托方向 参见 enum OrderSide
orderType	OrderType	委托类型 参见 enum OrderType
positionEffect	PositionEffect	开平类型 参见 enum PositionSide
price	double	委托价格
account	string	实盘账号id,关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 null
返回值	Order	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected， Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

示例：

```
1. //下限价单，以11元每股的价格买入价值为100000元的SHSE.600000，根据volume = value / price,计算并
    取整得到volume = 9000
2. GMDData<Order> o = order_value("SHSE.600000", 100000, OrderSide.OrderSide_Buy,
    OrderType.OrderType_Limit, positionEffect.PositionEffect_Open, 11);
```

注意：

- 1.仅支持一个标的代码，若交易代码输入有误，终端会拒绝此单，并显示 委托代码不正确。

2. 根据指定价值计算购买标的数量, 即 `value/price`。股票买卖最小单位为 `100`, 不足100部分 `向下取整`, 如存在不足100的持仓一次性卖出; 期货买卖最小单位为 `1`, `向下取整`。

3. 若仓位不足, 终端会拒绝此单, 显示 `仓位不足`。平仓时股票默认 `平昨仓`, 期货默认 `平今仓`。应研究需要, `股票`也支持卖空操作。

4. OrderType优先级高于price, 若指定OrderType\_Market下市价单, 使用价格为最新一个tick中的最新价, price参数失效。则price参数失效。若OrderType\_Limit限价单, 仿真模式价格错误, 终端拒绝此单, 显示委托价格错误, `回测模式`下对价格无限制。

5. 函数调用成功并不意味着委托已经成功, 只是意味委托单已经成功发出去, 委托是否成功根据OnOrderStatus, 或GetOrder来判断。

## OrderPercent - 按总资产指定比例委托

按总资产指定比例委托, 如果调用成功, 后续委托单状态变化将会触发OnOrderStatus回调。

函数原型:

```
1. GMDData<Order> OrderPercent(string symbol, double percent, OrderSide side, OrderType orderType, PositionEffect positionEffect, double price = 0, string account = null)
```

参数:

参数名	类型	说明
symbol	string	标的代码, 只能单个标的
percent	double	委托占总资产比例
side	OrderSide	委托方向 参见 <code>enum OrderSide</code>
orderType	OrderType	委托类型 参见 <code>enum OrderType</code>
positionEffect	PositionEffect	开平类型 参见 <code>enum PositionSide</code>
price	double	委托价格
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为 null
返回值	Order	一个Order结构, 如果函数调用失败, Order.status 值为 <code>OrderStatus_Rejected</code> , Order.ordRejReasonDetail 为错误原因描述, 其它情况表示函数调用成功, Order.clOrdId 为本次委托的标识, 可用于追溯订单状态或撤单

示例:

```
1. //当前总资产为1000000。下限价单, 以11元每股的价格买入SHSE.600000, 期望买入比例占总资产的10%, 根据
   volume = nav * precent / price 计算取整得出 volume = 9000
2.
3. GMDData<Order> o = OrderPercent("SHSE.600000", 0.1, OrderSide.OrderSide_Buy,
   OrderType.OrderType_Limit, PositionEffect.PositionEffect_Open, 11);
```

注意:

1. 仅支持一个标的代码, 若交易代码输入有误, 终端会拒绝此单, 并显示 `委托代码不正确`。

2. 根据指定比例计算购买标的数量, 即 `(nav*precent)/price` , 股票买卖最小单位为 `100` , 不足100部分 `向下取整` , 如存在不足100的持仓一次性卖出; 期货买卖最小单位为 `1` , `向下取整` 。
3. 若仓位不足, 终端会拒绝此单, 显示 `仓位不足` 。平仓时股票默认 `平昨仓` , 期货默认 `平今仓` 。应研究需要, `股票` 也支持卖空操作 。
4. OrderType优先级高于price, 若指定OrderType\_Market下市价单, 使用价格为最新一个tick中的最新价, price参数失效。则price参数失效。若OrderType\_Limit限价单, 仿真模式价格错误, 终端拒绝此单, 显示委托价格错误, `回测模式` 下对价格无限制 。
5. 函数调用成功并不意味着委托已经成功, 只是意味委托单已经成功发出去, 委托是否成功根据OnOrderStatus, 或GetOrder来判断。

## OrderTargetVolume - 调仓到目标持仓量

调仓到目标持仓量, 如果调用成功, 后续委托单状态变化将会触发OnOrderStatus回调。

函数原型:

```
1. GMDData<Order> OrderTargetVolume(string symbol, int volume, PositionSide positionSide,
    OrderType orderType, double price = 0, string account = null)
```

参数:

参数名	类型	说明
symbol	string	标的代码, 只能单个标的
volume	int	期望的最终数量
positionSide	PositionSide	持仓方向 参见 <code>enum PositionSide</code> )
orderType	OrderType	委托类型 参见 <code>enum OrderType</code>
price	double	委托价格
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	Order	一个Order结构, 如果函数调用失败, Order.status 值为 <code>OrderStatus_Rejected</code> , Order.ordRejReasonDetail 为错误原因描述, 其它情况表示函数调用成功, Order.clOrdId 为本次委托的标识, 可用于追溯订单状态或撤单

示例:

```
1. //当前SHSE.600000多方向持仓量为0, 期望持仓量为10000, 下单量为期望持仓量 - 当前持仓量 = 10000
2.
3. GMDData<Order> o = OrderTargetVolume("SHSE.600000", 10000,
    PositionSide.PositionSide_Long, OrderType.OrderType_Limit, 11);
```

注意:

1. 仅支持一个标的代码, 若交易代码输入有误, 终端会拒绝此单, 并显示 `委托代码不正确` 。
2. 根据目标数量计算下单数量, 系统判断开平仓类型。若下单数量有误, 终端拒绝此单, 并显示 `委托量不正确` 。若实际需要买入数量为0, 则订单会被拒绝, `终端无显示, 无回报` 。股票买卖最小单位为 `100` , 不足100部分 `向下取整` , 如

存在不足100的持仓一次性卖出;期货买卖最小单位为 1 ， 向下取整 。

3.若仓位不足，终端会拒绝此单，显示 仓位不足 。平仓时股票默认 平昨仓 ，期货默认 平今仓 。应研究需要， 股票也支持卖空操作 。

4.OrderType优先级高于price,若指定OrderType\_Market下市价单，使用价格为最新一个tick中的最新价，price参数失效。则price参数失效。若OrderType\_Limit限价单，仿真模式价格错误，终端拒绝此单，显示委托价格错误， 回测模式下对价格无限制 。

5.函数调用成功并不意味着委托已经成功，只是意味委托单已经成功发出去， 委托是否成功根据OnOrderStatus，或GetOrder来判断。

## OrderTargetValue - 调仓到目标持仓额

调仓到目标持仓额，如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调。

函数原型：

```
1. GMDData<Order> OrderTargetValue(string symbol, double value, PositionSide positionSide,
    OrderType orderType, double price = 0, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
value	int	期望的股票最终价值
positionSide	PositionSide	持仓方向 参见 enum PositionSide )
orderType	OrderType	委托类型 参见 enum OrderType
price	double	委托价格
account	string	实盘账号id,关联多实盘账号时填写，可以从 get_accounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	Order	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

示例：

```
1. //当前SHSE.600000多方向当前持仓量为0，目标持有价值为100000的该股票，根据value / price 计算取整得出目标持仓量volume为9000，目标持仓量 - 当前持仓量 = 下单量为9000
2.
3. GMDData<Order> o = OrderTargetValue("SHSE.600000", 100000,
    PositionSide.PositionSide_Long, OrderType.OrderType_Limit, 11);
```

注意：

- 1.仅支持一个标的代码，若交易代码输入有误，终端会拒绝此单，并显示 委托代码不正确 。
- 2.根据目标数量计算下单数量，系统判断开平仓类型。若下单数量有误，终端拒绝此单，并显示 委托量不正确 。若实际需要买入数量为0，则订单会被拒绝， 终端无显示，无回报 。股票买卖最小单位为 100 ，不足100部分 向下取整 ，如



存在不足100的持仓一次性卖出；期货买卖最小单位为 `1`，`向下取整`。

3.若仓位不足，终端会拒绝此单，显示 `仓位不足`。平仓时股票默认 `平昨仓`，期货默认 `平今仓`。应研究需要，`股票`也支持卖空操作。

4.OrderType优先级高于price,若指定OrderType\_Market下市价单，使用价格为最新一个tick中的最新价，price参数失效。则price参数失效。若OrderType\_Limit限价单，仿真模式价格错误，终端拒绝此单，显示委托价格错误，`回测模式`下对价格无限制。

5.函数调用成功并不意味着委托已经成功，只是意味委托单已经成功发出去，委托是否成功根据OnOrderStatus，或GetOrder来判断。

## OrderTargetPercent - 调仓到目标持仓比例（总资产的比例）

调仓到目标持仓比例（总资产的比例），如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调。

函数原型：

```
1. GMDData<Order> OrderTargetPercent(string symbol, double percent, PositionSide positionSide, OrderType orderType, double price = 0, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
percent	double	期望的最终总资产比例
positionSide	PositionSide	持仓方向 参见 <code>enum PositionSide</code> )
orderType	OrderType	委托类型 参见 <code>enum OrderType</code>
price	double	委托价格
account	string	实盘账号id,关联多实盘账号时填写，可以从 get_accounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	Order	一个Order结构，如果函数调用失败，Order.status 值为 <code>OrderStatus_Rejected</code> ，Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功，Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

示例：

```
1. //当前总资产价值为1000000，目标为以11元每股的价格买入SHSE.6000000的价值占总资产的10%，根据volume = nav * percent / price 计算取整得出应持有9000股。当前该股持仓量为零，因此买入量为9000
2.
3. GMDData<Order> o = OrderTargetPercent("SHSE.6000000", 0.1, PositionSide.PositionSide_Long, OrderType.OrderType_Limit, 11);
```

注意：

1.仅支持一个标的代码，若交易代码输入有误，终端会拒绝此单，并显示 `委托代码不正确`。

2.根据目标比例计算下单数量，为占 `总资产(nav)` 比例，系统判断开平仓类型。若下单数量有误，终端拒绝此单，并显示 `委托量不正确`。若实际需要买入数量为0，则本地拒绝此单，`终端无显示，无回报`。股票买卖最小单位为 `100`，

不足100部分 `向下取整`，如存在不足100的持仓一次性卖出；期货买卖最小单位为 `1`，`向下取整`。

3.若仓位不足，终端会拒绝此单，显示 `仓位不足`。平仓时股票默认 `平昨仓`，期货默认 `平今仓`。应研究需要，`股票`也支持卖空操作。

4.OrderType优先级高于price,若指定OrderType\_Market下市价单，使用价格为最新一个tick中的最新价，price参数失效。则price参数失效。若OrderType\_Limit限价单，仿真模式价格错误，终端拒绝此单，显示委托价格错误，`回测模式`下对价格无限制。

5.函数调用成功并不意味着委托已经成功，只是意味委托单已经成功发出去，委托是否成功根据OnOrderStatus，或GetOrder来判断。

## OrderCloseAll - 平当前所有可平持仓

平当前所有可平持仓，如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调

函数原型：

```
1. GMDDataList<Order> OrderCloseAll()
```

参数：

参数名	类型	说明
返回值	GMDDataList<order>	一个 GMDDataList<order> 对象

## OrderCancel - 委托撤单

撤销单个委托单，如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调

函数原型：

```
1. int OrderCancel(string clOrdIds, string account = null)
```

参数：

参数名	类型	说明
clOrdIds	string	委托单的客户id，可以在下单或查单时获得
account	string	实盘账号id，关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	int	成功返回0，失败返回错误码

## OrderCancelAll - 撤销所有委托

撤销所有委托，如果调用成功，后续委托单状态变化将会触发OnOrderStatus回调

函数原型：

```
1. int OrderCancelAll();
```

参数：

参数名	类型	说明
返回值	int	成功返回0， 失败返回错误码

## GetOrders - 查询所有委托

查询所有委托单

函数原型：

```
1. GMDataList<Order> GetOrders(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	GMDataList<Order>	一个 GMDataList<order> 对象

## GetUnfinishedOrders - 查询未结委托

查询所有未结委托

函数原型：

```
1. GMDataList<Order> GetUnfinishedOrders(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	GMDataList<Order>	一个 GMDataList<order> 对象

## GetExecutionReports - 查询成交

查询所有成交

函数原型：

```
1. GMDataList<ExecRpt> GetExecutionReports(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的成交
返回值	GMDataList<ExecRpt>	一个 GMDataList<ExecRpt> 对象

## GetCash - 查询资金

查询资金

函数原型：

```
1. GMDataList<Cash> GetCash(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为NULL，则返回所有账号的资金
返回值	GMDataList<Cash>	一个 GMDataList<Cash> 对象

## GetPosition - 查询持仓

查询所有持仓

函数原型：

```
1. GMDataList<Position> GetPosition(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的持仓
返回值	GMDataList<Position>	一个 GMDataList<Position> 对象

# 两融业务成员函数

- `CreditBuyingOnMargin` - 融资买入
- `CreditShortSelling` - 融券卖出
- `CreditRepayShareByBuyingShare` - 买券还券
- `CreditRepayShareByBuyingShare` - 卖券还款
- `CreditBuyingOnCollateral` - 担保品买入
- `CreditSellingOnCollateral` - 担保品卖出
- `CreditRepayShareDirectly` - 直接还券
- `CreditRepayCashDirectly` - 直接还款
- `CreditCollateralIn` - 担保品转入
- `CreditCollateralOut` - 担保品转出
- `CreditGetCollateralInstruments` - 查询担保证券
- `CreditGetBorrowableInstruments` - 查询融券标的证券
- `CreditGetBorrowableInstrumentsPositions` - 查询融券账户头寸
- `CreditGetContracts` - 查询融资融券合约
- `CreditGetCash` - 查询融资融券资金

## CreditBuyingOnMargin - 融资买入

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditBuyingOnMargin(string symbol, int volume, double price,
    OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
    OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
    OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData\	

示例：

```
1. //以价格11.9融资买入10000股浦发银行
2. GMDData<Order> o = CreditBuyingOnMargin("SHSE.600000", 10000, 11.9);
```

## CreditShortSelling - 融券卖出

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditShortSelling(string symbol, int volume, double price,
    OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
    OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
    OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 <code>null</code>
返回值	GMDData<Order>	Order结构

示例：

```
1. //以价格11.9融券卖出10000股浦发银行
2. GMDData<Order> o = CreditShortSelling("SHSE.600000", 10000, 11.9);
```

注意：

融券卖出一般不支持市价单，以柜台为准

## CreditRepayShareByBuyingShare - 买券还券

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditRepayShareByBuyingShare(string symbol, int volume, double
    price, OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
    OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
    OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 <code>null</code>
返回值	GMDData<Order>	-

示例：

```

1. //以价格11.9买入10000股浦发银行还券
2. GMDData<Order> o = CreditRepayShareByBuyingShare("SHSE.600000", 10000, 11.9);

```

## CreditRepayShareByBuyingShare - 卖券还款

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```

1. public GMDData<Order> CreditRepayCashBySellingShare(string symbol, int volume, double price, OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration = OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier = OrderQualifier.OrderQualifier_Unknown, string account = null)

```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 <code>null</code>
返回值	GMDData<Order>	-

示例：

```
1. //以价格11.9卖出10000股浦发银行还款
2. GMDData<Order> o = CreditRepayCashBySellingShare("SHSE.600000", 10000, 11.9);
```

## CreditBuyingOnCollateral - 担保品买入

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditBuyingOnCollateral(string symbol, int volume, double price,
    OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
    OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
    OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData\	

示例：

```
1. //以价格11.9买入10000股浦发银行
2. GMDData<Order> o = CreditBuyingOnCollateral("SHSE.600000", 10000, 11.9);
```

## CreditSellingOnCollateral - 担保品卖出

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditSellingOnCollateral(string symbol, int volume, double price,
    OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
    OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
```



```
OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
orderType	int	委托类型 参见 <code>enum OrderType</code>
orderDuration	int	委托时间属性 参见 <code>enum OrderDuration</code>
orderQualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 <code>null</code>
返回值	GMDData<Order>	-

示例：

```
1. //以价格11.9卖出10000股浦发银行
2. GMDData<Order> o = CreditSellingOnCollateral("SHSE.600000", 10000, 11.9);
```

## CreditRepayShareDirectly - 直接还券

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditRepayShareDirectly(string symbol, int volume, double price,
OrderType orderType = OrderType.OrderType_Limit, OrderDuration orderDuration =
OrderDuration.OrderDuration_Unknown, OrderQualifier orderQualifier =
OrderQualifier.OrderQualifier_Unknown, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>get_accounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为 <code>null</code>
返回值	GMDData<Order>	-

示例：

```
1. //还券10000股浦发银行
```

```
2. GMDData<Order> o = CreditRepayShareDirectly("SHSE.600000", 10000);
```

## CreditRepayCashDirectly - 直接还款

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public double CreditRepayCashDirectly(double amount, string account = null)
```

参数：

参数名	类型	说明
amount	double	还款金额
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	double	成功还款的金额

示例：

```
1. //还款 100000块
2. double repayAmount = CreditRepayCashDirectly(100000);
```

## CreditCollateralIn - 担保品转入

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMDData<Order> CreditCollateralIn(string symbol, int volume, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码, 只能单个标的
volume	int	转入数量
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	GMDData<Order>	-

示例：

```

1. //担保品转入10000股浦发银行
2. GMDData<Order> o = CreditCollateralIn("SHSE.600000", 10000);

```

## CreditCollateralOut - 担保品转出

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```

1. public GMDData<Order> CreditCollateralOut(string symbol, int volume, string account = null)

```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData\	

示例：

```

1. //担保品转出10000股浦发银行
2. GMDData<Order> o = CreditCollateralOut("SHSE.600000", 10000);

```

## CreditGetCollateralInstruments - 查询担保证券

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```

1. GMDDataList<CollateralInstrument> CreditGetCollateralInstruments(string account = null)

```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	GMDDataList<CollateralInstrument>	一个CollateralInstrument结构列表

## CreditGetBorrowableInstruments - 查询融券标的证券

注：融资融券暂时仅支持实盘委托，不支持仿真交易

查询标的证券，可做融券标的股票列表

函数原型：

```
1. public GMDaDataList <BorrowableInstrument> CreditGetBorrowableInstruments(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	GMDaDataList<BorrowableInstrument>	一个BorrowableInstrument结构列表

标的证券，可做融券标的股票列表

## CreditGetBorrowableInstrumentsPositions - 查询融券账户头寸

注：融资融券暂时仅支持实盘委托，不支持仿真交易

查询券商融券账户头寸，可用融券的数量

函数原型：

```
1. public GMDaDataList <BorrowableInstrumentPosition> CreditGetBorrowableInstrumentsPositions(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	GMDaDataList<BorrowableInstrumentPosition>	一个BorrowableInstrumentPosition结构列表

## CreditGetContracts - 查询融资融券合约

注：融资融券暂时仅支持实盘委托，不支持仿真交易

查询融资融券合约，负债

函数原型：

```
1. public GMDataList <CreditContract> CreditGetContracts(string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	<code>GMDataList&lt;CreditContract&gt;</code>	一个CreditContract结构数组

## CreditGetCash - 查询融资融券资金

注：融资融券暂时仅支持实盘委托，不支持仿真交易

函数原型：

```
1. public GMData<CreditCash> CreditGetCash(string account = null)
```

参数：

参数名	类型	说明
cash	CreditCash	出参，返回资金信息
account	string	账号ID <code>accountId</code> ，如果输入为null，则返回所有账号的委托
返回值	<code>GMData&lt;CreditCash&gt;</code>	-

示例：

```
1. CreditCash cash;
2. GMData<CreditCash> ccash = CreditGetCash(cash);
```

# 算法交易成员函数

- [OrderAlgo](#) - 委托算法单
- [AlgoOrderCancel](#) - 撤单算法委托
- [AlgoOrderPause](#) - 暂停/恢复算法单
- [GetAlgoOrders](#) - 查询算法委托
- [GetChildOrders](#) - 查询算法子委托

## OrderAlgo - 委托算法单

注：仅支持实时模式，部分券商版本可用

下算法单

函数原型：

```
1. public GMDData<AlgoOrder> OrderAlgo(string symbol, int volume, PositionEffect positionEffect, OrderSide side, OrderType orderType, string algoName, string algoParam, double price = 0, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
positionEffect	int	开平类型 参见 <code>enum PositionSide</code>
side	int	委托方向 参见 <code>enum OrderSide</code>
orderType	int	委托类型 参见 <code>enum OrderType</code>
price	double	委托价格
algoParam	struct	算法参数 参见 <code>struct AlgoParam</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>GetAccounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	<code>GMDData&lt;AlgoOrder&gt;</code>	如果函数调用失败， <code>AlgoOrder.status</code> 值为 <code>OrderStatus_Rejected</code> ， <code>Order.ordRejReasonDetail</code> 为错误原因描述，其它情况表示函数调用成功， <code>AlgoOrder.clOrdId</code> 为本次委托的标识，可用于追溯订单状态或撤单

示例：

```
1. //用算法 `ATS-SMART` 委托买入10000股浦发银行
2.
3. OrderAlgo("SHSE.600000", 10000, PositionEffect.PositionEffect_Open,
  OrderSide.OrderSide_Buy, OrderType.OrderType_Market, "ATS-SMART",
  "start_time&&1617069219|end_time&&1617080019||stop_sell_when_dl&&1||cancel_when_pl&&0||n
```

## AlgoOrderCancel - 撤单算法委托

注：仅支持实时模式，部分券商版本可用

撤销算法单

函数原型：

```
1. public int AlgoOrderCancel(string clOrdId, string account = null)
```

参数：

参数名	类型	说明
clOrdId	string	委托单的客户id，可以在下单或查单时获得
account	string	实盘账号id，关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	int	成功返回0， 失败返回错误码

## AlgoOrderPause - 暂停/恢复算法单

注：仅支持实时模式，部分券商版本可用

暂停/恢复算法单

函数原型：

```
1. public int AlgoOrderPause(string clOrdId, AlgoOrderStatus status, string account = null)
```

参数：

参数名	类型	说明
clOrdId	string	委托单的客户id，可以在下单或查单时获得
status	int	参考 <code>AlgoOrderStatus</code>
account	string	实盘账号id，关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	int	成功返回0， 失败返回错误码

## GetAlgoOrders - 查询算法委托

注：仅支持实时模式，部分券商版本可用

查询所有算法委托单

函数原型：

```
1. public GMDataList<AlgoOrder> GetAlgoOrders(string clOrdId = null, string account = null)
```

参数：

参数名	类型	说明
account	string	账号ID， 如果输入为null， 则返回所有账号的委托
返回值	GMDataList<AlgoOrder>	一个AlgoOrder结构列表

## GetChildOrders - 查询算法子委托

注：仅支持实时模式，部分券商版本可用

查询子单

函数原型：

```
1. public GMDataList<Order> GetChildOrders(string clOrdId, string account = null)
```

参数：

参数名	类型	说明
clOrdId	string	母单ID
account	string	账号ID， 如果输入为null， 则返回所有账号的委托
返回值	GMDataList<Order>	AlgoOrder结构列表



# 新股业务成员函数

- IPOBuy - 新股新债申购
- IPOGetQuota - 查询客户新股新债申购额度
- IPOGetInstruments - 查询当日新股新债清单
- IPOGetMatchNumber - 配号查询
- IPOGetLotInfo - 中签查询

## IPOBuy - 新股新债申购

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> IPOBuy(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	申购价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述， 其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

示例：

```
1. //申购1000股的SHSE.688001
2. GMDData<Order> o = IPOBuy("SHSE.688001", 1000, 42.0);
```

## IPOGetQuota - 查询客户新股新债申购额度

注：仅在实盘中可以使用

函数原型：

```
1. public GMDDataList<IPOQI> IPOGetQuota(string account = null)
```

参数：

参数名	类型	说明
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null

返回值	GMDaDataList<IPOQI>	返回每个板块的申购额度
-----	---------------------	-------------

示例：

```
1. GMDaDataList<IPOQI> da = IPOGetQuota();
```

## IPOGetInstruments - 查询当日新股新债清单

注：仅在实盘中可以使用

函数原型：

```
1. public GMDaDataList<IPOInstruments> IPOGetInstruments(string account = null)
```

参数：

参数名	类型	说明
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	GMDaDataList<IPOInstruments>	一个IPOInstruments结构列表

## IPOGetMatchNumber - 配号查询

注：仅在实盘中可以使用

函数原型：

```
1. public GMDaDataList<IPOMatchNumber> IPOGetMatchNumber(string account = null)
```

参数：

参数名	类型	说明
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	GMDaDataList<IPOMatchNumber>	一个IPOMatchNumber结构列表

## IPOGetLotInfo - 中签查询

注：仅在实盘中可以使用

函数原型：

```
1. public GMDaDataList<IPOLotInfo> IPOGetLotInfo(string account = null)
```

参数：

参数名	类型	说明
account	string	实盘账号id, 关联多实盘账号时填写, 可以从 GetAccounts获取, 也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号, 可以设置为null
返回值	GMDataList<IPOLotInfo>	一个IPOLotInfo结构列表

# 基金业务成员函数

- [FundEtfBuy](#) - ETF申购
- [FundEtfRedemption](#) - ETF赎回
- [FundSubScribing](#) - 基金认购
- [FundBuy](#) - 基金申购
- [FundRedemption](#) - 基金赎回

## FundEtfBuy - ETF申购

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> FundEtfBuy(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	申购份额
price	double	申购价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为NULL
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected , Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

## FundEtfRedemption - ETF赎回

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> FundEtfRedemption(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	赎回份额
price	double	赎回价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为NULL

返回值	<div>GMDData&lt;Order&gt;</div>	一个Order结构，如果函数调用失败，Order.status 值为 OrderStatus_Rejected，Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功，Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单
-----	---------------------------------	---

## FundSubScribing - 基金认购

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> FundSubScribing(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	认购份额
price	double	认购价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为NULL
返回值	<div>GMDData&lt;Order&gt;</div>	一个Order结构，如果函数调用失败，Order.status 值为 OrderStatus_Rejected，Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功，Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

## FundBuy - 基金申购

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> FundBuy(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	申购份额
price	double	申购价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为NULL
返回值	<div>GMDData&lt;Order&gt;</div>	一个Order结构，如果函数调用失败，Order.status 值为 OrderStatus_Rejected，Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功，Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

# FundRedemption - 基金赎回

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> FundRedemption(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	赎回份额
price	double	赎回价
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为NULL
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述， 其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

# 债券业务成员函数

- [BondReverseRepurchaseAgreement](#) - 国债逆回购
- [BondConvertibleCall](#) - 可转债转股
- [BondConvertiblePut](#) - 可转债回售
- [BondConvertiblePutCancel](#) - 可转债回售撤销

## BondReverseRepurchaseAgreement - 国债逆回购

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> BondReverseRepurchaseAgreement(string symbol, int volume, double price = 0, OrderType orderType = OrderType.OrderType_Limit, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	委托价格
order_type	int	委托类型 参见 <code>enum OrderType</code>
order_duration	int	委托时间属性 参见 <code>enum OrderDuration</code>
order_qualifier	int	委托成交属性 参见 <code>enum OrderQualifier</code>
account	string	实盘账号id, 关联多实盘账号时填写，可以从 <code>GetAccounts</code> 获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	<code>GMDData&lt;Order&gt;</code>	一个Order结构，如果函数调用失败， <code>Order.status</code> 值为 <code>OrderStatus_Rejected</code> ， <code>Order.ordRejReasonDetail</code> 为 错误原因描述，其它情况表示函数调用成功， <code>Order.clOrdId</code> 为本次委托的标识，可用于追溯订单状态或撤单

## BondConvertibleCall - 可转债转股

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> BondConvertibleCall(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量

price	double	转股价（大部分柜台忽略，可填0）
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

## BondConvertiblePut - 可转债回售

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> BondConvertiblePut(string symbol, int volume, double price, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
price	double	回售价（大部分柜台忽略，可填0）
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于追溯订单状态或撤单

## BondConvertiblePutCancel - 可转债回售撤销

注：仅在实盘中可以使用

函数原型：

```
1. public GMDData<Order> BondConvertiblePutCancel(string symbol, int volume, string account = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，只能单个标的
volume	int	委托数量
account	string	实盘账号id, 关联多实盘账号时填写，可以从 GetAccounts获取，也可以从终端实盘账号配置里拷贝。如果策略只关联一个账号，可以设置为null
返回值	GMDData<Order>	一个Order结构，如果函数调用失败， Order.status 值为 OrderStatus_Rejected ， Order.ordRejReasonDetail 为错误原因描述，其它情况表示函数调用成功， Order.clOrdId 为本次委托的标识，可用于



		追溯订单状态或撤单
--	--	-----------

# 动态参数成员函数

- [AddParameters](#) - 添加参数
- [DelParameters](#) - 删除参数
- [SetParameters](#) - 设置参数
- [GetParameters](#) - 获取参数
- [SetSymbols](#) - 设置标的
- [GetSymbols](#) - 获取标的

## AddParameters - 添加参数

添加动态参数， 添加成功后， 参数将在终端上显示。

函数原型：

```
1. int AddParameters(List<Parameter> parameters)
2. int AddParameters(Parameter parameter)
```

参数：

参数名	类型	说明
parameters	List	<code>Parameter</code> 列表
parameter	Parameter	一个 <code>Parameter</code> 对象
返回值	int	成功返回0， 失败返回错误码

## DelParameters - 删除参数

删除动态参数

函数原型：

```
1. int DelParameters(string keys)
```

参数：

参数名	类型	说明
keys	string	对应参数的键值， 多个参数使用 <code>,</code> 间隔，如 <code>key1, key2, ...</code>
返回值	int	成功返回0， 失败返回错误码

## SetParameters - 设置参数

设置参数值

函数原型：

```
1. int SetParameters(List<Parameter> parameters)
```

```
2. int SetParameters(Parameter parameter)
```

参数：

参数名	类型	说明
parameters	List	Parameter 列表
parameter	Parameter	Parameter 对象
返回值	int	成功返回0， 失败返回错误码

## GetParameters - 获取参数

获取参数值

函数原型：

```
1. GMDDataList<Parameter> GetParameters()
```

参数：

参数名	类型	说明
返回值	GMDDataList	一个 GMDDataList 对象

## SetSymbols - 设置标的

设置交易标的， 设置成功后， 标的将在终端上显示。

函数原型：

```
1. int SetSymbols(string symbols);
```

参数：

参数名	类型	说明
symbols	string	symbol列表，逗号分隔
返回值	int	成功返回0， 失败返回错误码

## GetSymbols - 获取标的

获取交易标的

函数原型：

```
1. GMDDataList<string> GetSymbols()
```

参数：

--	--	--

参数名	类型	说明
返回值	GMDataList	一个 GMDataList 对象

# 事件成员函数

- [OnInit](#) - 初始化完成
- [OnTick](#) - 收到Tick行情
- [OnBar](#) - 收到bar行情
- [OnOrderStatus](#) - 委托变化
- [OnExecutionReport](#) - 执行回报
- [OnParameter](#) - 参数变化
- [OnSchedule](#) - 定时任务触发
- [OnBacktestFinished](#) - 回测完成后收到绩效报告
- [OnAccountStatus](#) - 实盘账号状态变化
- [OnError](#) - 错误产生
- [OnStop](#) - 收到策略停止信号
- [OnMarketDataConnected](#) - 数据服务已经连接上
- [OnTradeDataConnected](#) - 交易已经连接上
- [OnMarketDataDisconnected](#) - 数据连接断开了
- [OnTradeDataDisconnected](#) - 交易连接断开了

## OnInit - 初始化完成

sdk初始化完成时触发，用户可以改写此成员函数，在些订阅行情，提取历史数据等初始化操作。

函数原型：

```
1. virtual void OnInit();
```

## OnTick - 收到Tick行情

收到Tick行情时触发

函数原型：

```
1. virtual void OnTick(Tick tick);
```

参数：

参数名	类型	说明
tick	Tick	收到的Tick行情

## OnBar - 收到bar行情

收到bar行情时触发

函数原型：

```
1. virtual void OnBar(Bar bar);
```

参数：

参数名	类型	说明
bar	List	收到的Bar行情

## OnOrderStatus - 委托变化

响应委托状态更新事情，下单后及委托状态更新时被触发。

注意：交易账户重连后，会重新推送一遍交易账户登录成功后查询回来的所有委托

函数原型：

```
1. virtual void OnOrderStatus(Order order);
```

参数：

参数名	类型	说明
order	Order	发生变化的委托

注意：

1. 交易服务连接断开重连后，会自动重新推送一次所有委托(包含近期委托)。
2. 交易账号错误断开到“已登陆”状态后，会自动重新推送一次所有委托(包含近期委托)。
3. 交易服务连接断开重连事件通过OnTradeDataConnected()回调通知。
4. 交易账号错误断开到“已登陆”事件通过OnAccountStatus()回调通知。
5. 主动查询日内全部委托记录和未结委托的方式为GetOrders()和GetUnfinishedOrders()函数。

## OnExecutionReport - 执行回报

响应委托被执行事件，委托成交或者撤单拒绝后被触发。

注意：交易账户重连后，会重新推送一遍交易账户登录成功后查询回来的所有执行回报

函数原型：

```
1. virtual void OnExecutionReport(ExecRpt rpt);
```

参数：

参数名	类型	说明
rpt	ExecRpt	收到的回报

注意：

1. 交易服务连接断开重连后，会自动重新推送一次所有成交(包含近期成交)。
2. 交易账号错误断开到“已登陆”状态后，会自动重新推送一次所有成交(包含近期成交)。
3. 交易服务连接断开重连事件通过OnTradeDataConnected()回调通知。
4. 交易账号错误断开到“已登陆”事件通过OnAccountStatus()回调通知。
5. 主动查询日内全部执行回报的方式为GetExecutionReports()函数。

## OnParameter - 参数变化

参数变化时触发，一般是终端修了动态参数

函数原型：

```
1. virtual void OnParameter(List<Parameter> param);
```

参数：

参数名	类型	说明
param	List	变化的参数

## OnSchedule - 定时任务触发

预设任务时间条件符合时触发

函数原型：

```
1. virtual void OnSchedule(string dataRule, string timeRule);
```

参数：

参数名	类型	说明
dataRule	string	设置的 dataRule
timeRule	string	设置的 timeRule

## OnBacktestFinished - 回测完成后收到绩效报告

回测完成后收到绩效报告时触发

函数原型：

```
1. virtual void OnBacktestFinished(Indicator indicator);
```

参数：

参数名	类型	说明
dataRule	Indicator	设置的 dataRule

## OnAccountStatus - 实盘账号状态变化

实盘账号状态变化时触发，比如实盘账号登录，退出登录等

函数原型：

```
1. virtual void OnAccountStatus(AccountStatus accountStatus);
```

参数：

参数名	类型	说明
accountStatus	AccountStatus	对应变化的账号

## OnError - 错误产生

有错误产生时触发，比如网络断开。

函数原型：

```
1. virtual void OnError(int errorCode, string errorMsg);
```

参数：

参数名	类型	说明
errorCode	int	错误码
errorMsg	string	错误信息

## OnStop - 收到策略停止信号

终端点击停止策略时触发

函数原型：

```
1. virtual void OnStop();
```

## OnMarketDataConnected - 数据服务已经连接上

数据服务已经连接时触发

函数原型：

```
1. virtual void OnMarketDataConnected();
```

## OnTradeDataConnected - 交易已经连接上

交易已经连接时触发

函数原型：

```
1. virtual void OnTradeDataConnected();
```

## OnMarketDataDisconnected - 数据连接断开了

数据连接断开时触发

函数原型：



```
1. virtual void OnMarketDataDisconnected();
```

## OnTradeDataDisconnected - 交易连接断开了

---

交易连接断开时触发

函数原型:

```
1. virtual void OnTradeDataDisconnected();
```

# 数据查询函数

- [GMApi 静态方法](#)
- [SetToken](#) - 设置token
- [SetAddr](#) - 设置终端服务地址
- [Current](#) - 查询当前行情快照
- [HistoryTicks](#) - 查询历史Tick行情
- [HistoryBars](#) - 查询历史Bar行情
- [HistoryTicksN](#) - 查询最新n条Tick行情
- [HistoryBarsN](#) - 查询最新n条Bar行情
- [GetFundamentals](#) - 查询基本面数据
- [GetFundamentalsN](#) - 查询基本面数据最新n条
- [GetInstruments](#) - 查询最新交易标的信息
- [GetHistoryInstruments](#) - 查询交易标的历史数据
- [GetInstrumentinfos](#) - 查询交易标的基本信息
- [GetConstituents](#) - 查询指数成份股
- [GetIndustry](#) - 查询行业股票列表
- [GetTradingDates](#) - 查询交易日历
- [GetPreviousTradingDate](#) - 返回指定日期的上一个交易日
- [GetNextTradingDate](#) - 返回指定日期的下一个交易日
- [GetDividend](#) - 查询分红送配
- [GetContinuousContracts](#) - 获取连续合约

## GMApi 静态方法

### SetToken - 设置token

函数原型

```
1. static int SetToken(string token);
```

参数

参数名	类型	说明
token	string	改参数通过 客户端->系统设置->(秘钥管理)token 获得
返回值	int	0 成功, 其他表示错误码

示例

```
1. GMApi.SetToken("27cbdfd8cd9b86dea554a5612baa4a8eee");
```

### SetAddr - 设置终端服务地址

设置终端服务地址，默认本地地址，可不填

函数原型：

```
1. void SetAddr(string addr)
```

参数：

参数名	类型	说明
addr	string	ip+终端端口，如"127.0.0.1:7001"

注意：

- 1. 如运行策略与终端不在同一设备上，必须调用此方法，7001 为终端端口
- 2. 不关注策略运行结果，可不启动终端并设置终端服务地址为 discovery.myquant.cn:7061

## Current - 查询当前行情快照

查询当前行情快照，返回tick数据。回测时，返回回测时间点的tick数据

函数原型：

```
1. static GMDataList<Tick> Current(string symbols);
```

参数：

参数名	类型	说明
symbols	string	查询代码，如有多个代码，中间用 , （英文逗号）隔开
返回值	GMDataList	一个 GMDataList<Tick> 对象

示例：

```
1. static GMDataList<Tick> currentData = GMApi.Current("SZSE.000001,SZSE.000002");
```

注意：

startTime和endTime中月,日,时,分,秒均可以只输入个位数,例: "2010-7-8 9:40:0" 或 "2017-7-30 12:3:0" ,但若对应位置为零,则 0 不可被省略,如不可输入 "2017-7-30 12:3: "

## HistoryTicks - 查询历史Tick行情

函数原型：

```
1. static GMDataList<Tick> HistoryTicks(string symbols, string startTime, string endTime, Adjust adjust = Adjust.ADJUST_NONE, string adjustEndTime = null, bool skipSuspended = true, string fillMissing = null)
```

参数：

参数名	类型	说明
symbols	string	标的代码,若要获取多个标的的历史数据,可以采用中间用 , （英文逗号）隔开的方法
		开始时间 (%Y-%m-%d %H:%M:%S 格式),其中日线包含start_time数据, 非

startTime	string	日线不包含start_time数据
endTime	string	结束时间 (%Y-%m-%d %H:%M:%S 格式),
adjust	Adjust	复权方式, 参见 <code>enum Adjust</code>
adjustEndTime	string	复权基点时间, 默认当前时间
skipSuspended	bool	是否跳过停牌, 默认跳过
fillMissing	string	填充方式, <code>None</code> - 不填充, <code>'NaN'</code> - 用空值填充, <code>'Last'</code> - 用上一个值填充, 默认None
返回值	GMDaDataList	一个 <code>GMDaDataList&lt;Tick&gt;</code> 对象

示例:

```
1. GMDaDataList<Tick> historyTick = GMApi.HistoryTicks("SHSE.000300", "2010-07-28 08:00:00", "2017-07-30 16:00:00")
```

注意:

1. startTime和endTime中月,日,时,分,秒均可以只输入个位数,例: `"2010-7-8 9:40:0"` 或 `"2017-7-30 12:3:0"`,但若对应位置为零,则 `0` 不可被省略,如不可输入 `"2017-7-30 12:3: "`

2. skipSuspended 和 fillMissing 参数暂不支持

## HistoryBars - 查询历史Bar行情

函数原型:

```
1. static GMDaDataList<Bar> HistoryBars(string symbols, string frequency, string startTime, string endTime, Adjust adjust = Adjust.ADJUST_NONE, string adjustEndTime = null, bool skipSuspended = true, string fillMissing = null)
```

参数:

参数名	类型	说明
symbols	string	标的代码,若要获取多个标的的历史数据,可以采用中间用 <code>,</code> (英文逗号) 隔开的方法
frequency	string	频率, 支持 <code>'1d'</code> , <code>'15s'</code> , <code>'30s'</code> , <code>'60s'</code> , <code>'3600s'</code> 等
startTime	string	开始时间 (%Y-%m-%d %H:%M:%S 格式),其中日线包含start_time数据, 非日线不包含start_time数据
endTime	string	结束时间 (%Y-%m-%d %H:%M:%S 格式),
adjust	Adjust	复权方式, 参见 <code>enum Adjust</code>
adjustEndTime	string	复权基点时间, 默认当前时间
skipSuspended	bool	是否跳过停牌, 默认跳过
fillMissing	string	填充方式, <code>None</code> - 不填充, <code>'NaN'</code> - 用空值填充, <code>'Last'</code> - 用上一个值填充, 默认None
返回值	GMDaDataList	一个 <code>GMDaDataList&lt;Bar&gt;</code> 对象

示例:

```
1. //获取1分钟的bar
2. GMDataList<Bar> historyBar = GMApi.HistoryBars("SHSE.000300", "60s", "2010-07-28 08:00:00", "2017-07-30 16:00:00");
3.
4. //获取60分钟的bar
5. GMDataList<Bar> historyBar = GMApi.HistoryBars("SHSE.000300", "3600s", "2010-07-28 08:00:00", "2017-07-30 16:00:00");
6.
7. //获取日线
8. GMDataList<Bar> historyBar = GMApi.HistoryBars("SHSE.000300", "1d", "2010-07-28 08:00:00", "2018-07-30 16:00:00");
```

注意：

- 1.startTime和endTime中月,日,时,分,秒均可以只输入个位数,例: "2010-7-8 9:40:0" 或 "2017-7-30 12:3:0" ,但若对应位置为零,则 0 不可被省略,如不可输入 "2017-7-30 12:3: "
- 2. skipSuspended 和 fillMissing 参数暂不支持

## HistoryTicksN - 查询最新n条Tick行情

函数原型：

```
1. static GMDataList<Tick> HistoryTicksN(string symbols, int n, string endTime = null, Adjust adjust = Adjust.ADJUST_NONE, string adjustEndTime = null, bool skipSuspended = true, string fillMissing = null)
```

参数：

参数名	类型	说明
symbols	string	标的代码(只允许单个标的的代码字符串)
n	int	获取行情的数量
endTime	string	从指定时间开始, 往前 取行情, 如果为NULL, 那么为当前时间开始(回测模式下为当前回测时间点)
adjust	Adjust	复权方式, 参见 enum Adjust
adjustEndTime	string	复权基点时间, 默认当前时间
skipSuspended	bool	是否跳过停牌, 默认跳过
fillMissing	string	填充方式, None - 不填充, 'NaN' - 用空值填充, 'Last' - 用上一个值填充, 默认None
返回值	GMDataList	一个 GMDataList<Tick> 对象

示例：

```
1.
2. //获取沪深300最新10条tick
3. GMDataList<Tick> ticks = GMApi.HistoryTicksN("SHSE.000300", 10);
```

注意：

1. endTime中月,日,时,分,秒均可以只输入个位数,例: "2010-7-8 9:40:0" 或 "2017-7-30 12:3:0",但若对应位置为零,则 0 不可被省略,如不可输入 "2017-7-30 12:3: "

2. skipSuspended 和 fillMissing 参数暂不支持

## HistoryBarsN - 查询最新n条Bar行情

函数原型:

```
1. static GMDaDataList<Bar> HistoryBarsN(string symbols, string frequency, int n, string
    endTime = null, Adjust adjust = Adjust.ADJUST_NONE, string adjustEndTime = null, bool
    skipSuspended = true, string fillMissing = null)
```

参数:

参数名	类型	说明
symbols	string	标的代码(只允许单个标的的代码字符串)
frequency	string	频率, 支持 '1d', '15s', '30s', '60s', '3600s' 等
n	int	获取行情的数量
endTime	string	从指定时间开始, 往前 取行情, 如果为NULL, 那么为当前时间开始(回测模式下为当前回测时间点)
adjust	Adjust	复权方式, 参见 enum Adjust
adjustEndTime	string	复权基点时间, 默认当前时间
skipSuspended	bool	是否跳过停牌, 默认跳过
fillMissing	string	填充方式, None - 不填充, 'NaN' - 用空值填充, 'Last' - 用上一个值填充, 默认None
返回值	GMDaDataList	一个 GMDaDataList<Bar> 对象

示例:

```
1.
2. //获取沪深300最新10条1分钟bar
3. GMDaDataList<Bar> bars = GMApi.HistoryBarsN("SHSE.000300", "60s", 10);
```

注意:

1. endTime中月,日,时,分,秒均可以只输入个位数,例: "2010-7-8 9:40:0" 或 "2017-7-30 12:3:0",但若对应位置为零,则 0 不可被省略,如不可输入 "2017-7-30 12:3: "

2. skipSuspended 和 fillMissing 参数暂不支持

## GetFundamentals - 查询基本面数据

函数原型:

```
1. static GMDaData<DataTable> GetFundamentals(string table, string symbols, string
    startDate, string endDate, string fields = null, string filter = null, string orderBy =
    null, int limit = 100000)
```

参数:

参数名	类型	说明
table	string	表名, 只支持单表查询. 具体表名及fields字段名及filter可过滤的字段参考 <a href="#">财务数据文档</a>
symbols	string	标的代码, 多个代码可用 , (英文逗号)分割, 如 "symbol1,symbol2"
startDate	string	开始时间, (%Y-%m-%d 格式)
endDate	string	结束时间, (%Y-%m-%d 格式)
fields	string	查询字段 (必填)
filter	string	查询过滤,使用方法参考例3、例4
orderBy	string	排序方式, 默认 null . TCLOSE 表示按 TCLOSE 升序排序. -TCLOSE 表示按 TCLOSE 降序排序. TCLOSE, -NEGOTIABLEMV 表示按 TCLOSE 升序, NEGOTIABLEMV 降序综合排序
limit	int	数量. 为保护服务器, 一次查询最多返回 40000 条记录
返回值	GMDData	一个 GMDData<DataTable> 对象

示例:

例1: 取股票代码 SHSE.600000, SZSE.000001 , 离 2017-01-01 最近一个交易日的 股票交易财务衍生表 的 TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI 字段的值

```
1. GMDData<DataTable> ds = GMApi.GetFundamentals("trading_derivative_indicator",
"SHSE.600000,SZSE.000001", "2017-01-01", "2017-01-01",
2. "TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI")
```

例2: 取股票代码 SHSE.600000, SZSE.000001 , 指定时间段 2016-01-01 -- 2017-01-01 股票交易财务衍生表的 TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI 字段的值

```
1. GMDData<DataTable> ds = GMApi.GetFundamentals("trading_derivative_indicator",
"SHSE.600000,SZSE.000001", "2016-01-01", "2017-01-01",
2. "TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI")
```

例3: 取指定股票 SHSE.600000, SHSE.600001, SHSE.600002 离 2017-01-01 最近一个交易日, 且满足条件 PCTTM > 0 and PCTTM < 10 股票交易财务衍生表 的 TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI 字段的值

```
1. GMDData<DataTable> ds = GMApi.GetFundamentals("trading_derivative_indicator",
"SHSE.600000,SHSE.600001,SHSE.600002", "2017-01-01", "2017-01-01", "PCTTM > 0 and PCTTM
< 10",
2. "TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAAEI, PETTMNPAAEI")
```

例4: 取指定股票 SHSE.600000, SZSE.000001 离 2016-01-20 最近一个财报, 同时满足条件 CURFDS > 0 and TOTLIABSHAREQUI > 0 的 资产负债 的数据

```
1. GMDData<DataTable> ds = GMApi.GetFundamentals("balance_sheet", "2016-01-20", "2016-01-20",
2.     "CURFDS, SETTRESEDEPO, PLAC, TRADFINASSET, ',
3.     symbols='SHSE.600000, SZSE.000001'",
4.     "CURFDS > 0 and TOTLIABSHAREQUI > 0"
5. )
```

注意：

- 1.当 startDate == endDate 时，取所举每个股票离 endDate 最近业务日期（交易日期或报告日期）一条数据,当 startDate < endDate 时，取指定时间段的数据,当 startDate > endDate 时，返回 空list/空DataFrame
- 2.startDate和endDate中月,日均可以只输入个位数,例： '2010-7-8' 或 '2017-7-30'
- 3.在该函数中，table参数只支持输入一个表名，若表名输入错误或以 'table1,table2' 方式输入多个表名，函数返回空结果集
- 4.若表名输入正确，但查询字段中出现非指定字符串，则返回错误

## GetFundamentalsN - 查询基本面数据最新n条

取指定股票的最近 endDate 的 n 条记录

函数原型：

```
1. static GMDData<DataTable> GetFundamentalsN(string table, string symbols, string endDate,
    int n = 1, string fields = null, string filter = null, string orderBy = null)
```

参数：

参数名	类型	说明
table	string	表名。具体表名及fields字段名及filter可过滤的字段参考 <a href="#">财务数据文档</a>
symbols	string	标的代码，多个代码可用 , （英文逗号）分割
endDate	string	结束时间，(%Y-%m-%d 格式)
fields	string	查询字段（必填）
filter	string	查询过滤,,使用方法参考 GetFundamentals 的例3、例4
n	int	每个股票取最近的数量(正整数)
返回值	GMDData	一个 GMDData<DataTable> 对象

示例：

例：取股票代码 SHSE.600000, SZSE.000001 , 离 2017-01-01 最近3条(每个股票都有3条) 股票交易财务衍生表的 TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAEI, PETTMNPAEI 字段的值

```
1. GMDData<DataTable> ds = GMApi.GetFundamentalsN("trading_derivative_indicator",
    "SHSE.600000, SZSE.000001", "2017-01-01", 3,
2.     "TCLOSE, NEGOTIABLEMV, TOTMKTCAP, TURNRATE, PELFY, PETTM, PEMRQ, PELFYNPAEI, PETTMNPAEI"
```



```
"TCLOSE,NEGOTIABLEMV,TOTMKTCAP,TURNRATE,PELFY,PETTM,PEMRQ,PELFYNPAAEI,PETTMNPAAEI")
```

注意：

- 1.endDate中月,日均可以只输入个位数,例: '2010-7-8' 或 '2017-7-30'
- 2.在该函数中, table参数只支持输入一个表名, 若表名输入错误或以 'table1,table2' 方式输入多个表名, 函数返回空结果集
- 3.若表名输入正确, 但查询字段中出现非指定字符串, 则返回错误

## GetInstruments - 查询最新交易标的信息

查询最新交易标的信息, 有基本数据及最新日频数据

函数原型：

```
1. static GMDData<DataTable> GetInstruments(string symbols = null, string exchanges = null, string secTypes = null, string fields = null)
```

参数：

参数名	类型	说明
symbols	string	股票代码, 多个代码可用 , (英文逗号)分割, null表示所有
exchanges	string	交易所代码, 多个交易所代码可用 , (英文逗号)分割, null表示所有
secTypes	string	指定类别, 品种类型, stock: 股票, fund: 基金, index: 指数, future: 期货, option: 期权, confuture: 虚拟合约, 多个品种可用 , (英文逗号)分割, null表示所有品种
fields	string	查询字段 默认null 表示所有
返回值	GMDData	一个 GMDData<DataTable> 对象

fields 字段：

字段名	类型	说明
symbol	string	标的代码
sec_level	int	1-正常, 2-ST 股票, 3-*ST 股票, 4-股份转让, 5-处于退市整理期的证券, 6-上市开放基金LOF, 7-交易型开放式指数基金(ETF), 8-非交易型开放式基金(暂不交易, 仅揭示基金净值及开放申购赎回业务), 9-仅提供净值揭示服务的开放式基金;, 10-仅在协议交易平台挂牌交易的证券, 11-仅在固定收益平台挂牌交易的证券, 12-风险警示产品, 13-退市整理产品, 99-其它
is_suspended	int	是否停牌. 1: 是, 0: 否
multiplier	double	合约乘数
margin_ratio	double	保证金比率
settle_price	double	结算价
position	int	持仓量
pre_close	double	昨收价
pre_settle	double	昨结算价
upper_limit	double	涨停价

lower_limit	double	跌停价
adj_factor	double	复权因子. 基金跟股票才有
created_at	DateTime	交易日期

注意：

- 1. 停牌时且股票发生除权除息，涨停价和跌停价可能有误差
- 2. 预上市股票以1900-01-01为虚拟发布日期，未退市股票以2038-01-01为虚拟退市日期。

示例：

```
1. GMApi.GetInstruments("SZSE.000001");           //获取平安银行股票信息
2. GMApi.GetInstruments(null, "SZSE,SHSE", "stock"); //获取深交所、上交所所有股票信息
```

## GetHistoryInstruments - 查询交易标的历史数据

返回指定symbols的标的日频历史数据

函数原型：

```
1. static GMDData<DataTable> GetHistoryInstruments(string symbols, string startDate, string
    endDate, string fields = null)
```

参数：

参数名	类型	说明
symbols	string	标的代码，多个代码可用 , (英文逗号)分割, 也支持 ['symbol1', 'symbol2'] 这种列表格式，是必填参数
startDate	string	开始时间. (%Y-%m-%d 格式)
endDate	string	结束时间. (%Y-%m-%d 格式)
fields	string	查询字段. null表示所有字段
返回值	GMDData	一个 GMDData<DataTable> 对象

fields 字段：

字段名	类型	说明
symbol	string	标的代码
sec_level	int	1- 正常, 2-ST 股票, 3-*ST 股票, 4-股份转让, 5-处于退市整理期的证券, 6- 上市开放基金LOF, 7- 交易型开放式指数基金 (ETF), 8- 非交易型开放式基金 (暂不交易, 仅揭示基金净值及开放申购赎回业务), 9- 仅提供净值揭示服务的开放式基金; 10- 仅在协议交易平台挂牌交易的证券, 11- 仅在固定收益平台挂牌交易的证券, 12- 风险警示产品, 13- 退市整理产品, 99- 其它
is_suspended	int	是否停牌. 1: 是, 0: 否
multiplier	double	合约乘数
margin_ratio	double	保证金比率
settle_price	double	结算价

position	int	持仓量
pre_close	double	昨收价
pre_settle	double	昨结算价
upper_limit	double	涨停价
lower_limit	double	跌停价
adj_factor	double	复权因子.基金跟股票才有
created_at	DateTime	交易日期

示例:

```
1. GMApi.GetHistoryInstruments("SZSE.000001,SZSE.000002", "2017-09-19", "2017-09-19");
```

注意:

- 1.停牌时且股票发生除权除息，涨停价和跌停价可能有误差
- 2.为保护服务器，单次查询最多返回 3300 条记录
- 3.startDate和endDate中月,日均可以只输入个位数,例: '2010-7-8' 或 '2017-7-30'

## GetInstrumentinfos - 查询交易标的基本信息

获取到交易标的基本信息，与时间无关。

函数原型:

```
1. static GMDData<DataTable> GetInstrumentinfos(string symbols = null, string exchanges = null, string secTypes = null, string names = null, string fields = null)
```

参数:

参数名	类型	说明
symbols	string	标的代码，多个代码可用 , (英文逗号)分割，NULL 表示所有
exchanges	string	交易市场代码，多个交易所代码可用 , (英文逗号)分割，NULL 表示所有市场
secTypes	string	指定类别，品种类型， stock: 股票, fund: 基金, index: 指数, future: 期货, option: 期权, confuture: 虚拟合约，多个品种可用 , (英文逗号)分割，null表示所有品种
names	string	查询代码，NULL 表示所有
fields	string	查询字段 NULL 表示所有
返回值	GMDData	一个 GMDData<DataTable> 对象

字段:

字段名	类型	说明
symbol	string	标的代码
sec_type	int	1: 股票, 2: 基金, 3: 指数, 4: 期货, 5: 期权, 10: 虚拟合约
exchange	string	见 <a href="#">交易所代码</a>

sec_id	string	代码
sec_name	string	名称
price_tick	double	最小变动单位
listed_date	DateTime	上市日期
delisted_date	DateTime	退市日期

示例：

```
1. GMDData<DataTable> ds = GMApi.GetInstrumentinfos("SHSE.000001,SHSE.000002");
```

注意：

1.对于检索所需标的信息的4种手段 symbols,exchanges,secTypes,names ,若输入参数之间出现任何矛盾（换句话说，所有的参数限制出满足要求的交集为空），则返回空结果集  
例如 GetInstrumentinfos("SZSE","4") 返回的是空结果集

## GetConstituents - 查询指数成份股

函数原型：

```
1. static GMDData<DataTable> GetConstituents(string index, string tradeDate = null)
```

参数：

参数名	类型	说明
index	string	表示指数的symbol,比如上证指数SHSE.000001, 不支持多个 symbol
trade_date	string	交易日（%Y-%m-%d 格式） NULL 表示最新日期
返回值	GMDData	一个 GMDData<DataTable> 对象

字段：

字段名	类型	说明
symbol	string	标的代码
weight	double	权重

示例：

```
1. GMDData<DataTable> ds = GMApi.GetConstituents("SHSE.000001", "2017-07-10");
```

注意：

1.tradeDate 中月,日均可以只输入个位数,例： "2010-7-8" 或 "2017-7-30" ,但若对应位置为零,则 0 不可被省略

## GetIndustry - 查询行业股票列表

函数原型：

```
1. static GMDaDataList<string> GetIndustry(string code)
```

参数：

参数名	类型	说明
code	string	行业代码
返回值	GMDaDataList	一个 GMDaDataList<string> 对象

示例：

```
1. #返回所有以J6开头的行业代码对应的成分股 (包括: J66, J67, J68, J69的成分股)
2. GMDaDataList<string> da = GMApi.GetIndustry("j6");
```

注意：

1. 在该函数中，code参数只支持输入一个行业代码

## GetTradingDates - 查询交易日历

函数原型：

```
1. static GMDaDataList<DateTime> GetTradingDates(string exchange, string startDate, string endDate)
```

参数：

参数名	类型	说明
exchange	string	交易市场代码
startDate	string	开始时间 (%Y-%m-%d 格式)
endDate	string	结束时间 (%Y-%m-%d 格式)
返回值	GMDaDataList	一个 GMDaDataList<DateTime> 对象

示例：

```
1. GMDaDataList<DateTime> da = GMApi.GetTradingDates("SZSE", "2017-01-01", "2017-01-30");
```

注意：

1. exchange 参数仅支持输入单个交易所代码
2. startDate 和 endDate 中月, 日均可以只输入个位数,  
例: "2010-7-8" 或 "2017-7-30"

## GetPreviousTradingDate - 返回指定日期的上一个交易日

函数原型：

```
1. static GMDData<DateTime> GetPreviousTradingDate(string exchange, string date)
```

参数：

参数名	类型	说明
exchange	string	交易市场代码
date	string	时间（%Y-%m-%d 格式）
返回值	GMDData	一个 GMDData<DateTime> 对象

示例：

```
1. GMDData<DateTime> dt = GMApi.GetPreviousTradingDate("SZSE", "2017-05-01");
```

注意：

1. exchange 参数仅支持输入单个交易所代码
2. date 中月,日均可以只输入个位数,  
例: '2010-7-8' 或 '2017-7-30'

## GetNextTradingDate - 返回指定日期的下一个交易日

函数原型：

```
1. static GMDData<DateTime> GetNextTradingDate(string exchange, string date)
```

参数：

参数名	类型	说明
exchange	string	交易市场代码
date	string	时间（%Y-%m-%d 格式）
返回值	GMDData	一个 GMDData<DateTime> 对象

示例：

```
1. GMDData<DateTime> dt = GMApi.GetNextTradingDate("SZSE", "2017-05-01");
```

注意：

1. exchange 参数仅支持输入单个交易所代码
2. date 中月,日均可以只输入个位数,  
例: "2010-7-8" 或 "2017-7-30"

## GetDividend - 查询分红送配

函数原型：

```
1. static GMDData<DataTable> GetDividend(string symbol, string startDate, string endDate = null)
```

参数：

参数名	类型	说明
symbol	string	标的代码，（必填）
startDate	string	开始时间（%Y-%m-%d 格式）
endDate	string	结束时间（%Y-%m-%d 格式）
返回值	GMDData	一个 GMDData<DataTable> 对象

返回字段：

字段名	类型	说明
symbol	string	标的代码
cash_div	double	每股派现
allotment_ratio	double	每股配股比例
allotment_price	double	配股价
share_div_ratio	double	每股送股比例
share_trans_ratio	double	每股转增比例
created_at	DateTime	创建时间

示例：

```
1. GMDData<DataTable> ds = GMApi.GetDividend("SHSE.600000", "2000-01-01", "2017-12-31");
```

注意：

- 1. 在该函数中，symbol参数只支持输入一个标的代码
- 2. startDate 和 endDate 中月,日均可以只输入个位数，  
例： "2010-7-8" 或 "2017-7-30"

## GetContinuousContracts - 获取连续合约

函数原型：

```
1. static GMDData<DataTable> GetContinuousContracts(string csymbol, string startDate = null, string endDate = null)
```

参数：

参数名	类型	说明
csymbol	string	虚拟合约代码，比如获取主力合约，输入SHFE.AG;获取连续合约，输入SHFE.AG01

startDate	string	开始时间 (%Y-%m-%d 格式)
endDate	string	结束时间 (%Y-%m-%d 格式)
返回值	GMDData	一个 GMDData<DataTable> 对象

返回字段：

字段名	类型	说明
symbol	string	真实合约
created_at	DateTime	交易日

示例：

```
1. GMDData<DataTable> ds = GMApi.GetContinuousContracts("SHFE.AG", "2017-07-01", "2017-08-01");
```

注意：

- 1. 在该函数中，symbol参数只支持输入一个标的代码
- 2. startDate 和 endDate 中月,日均可以只输入个位数, 例： '2017-7-1' 或 '2017-8-1'



# GMDDataList

- [GMDDataList 类定义](#)
- [使用举例](#)

## GMDDataList 类定义

**GMDDataList** 模板类是所有返回列表型数据函数的标准返回， 表示一个列表数据存储。类声明如下：

```
1. public class GMDDataList<T>
2. {
3.     public List<T> data;           //数据
4.     public int status;            //状态码， 0表示成功， 其它表示错误码
5. }
```

## 使用举例

```
1. //查询一段tick行情
2. GMDDataList<Tick> dl = GMApi.HistoryTicks("SHSE.600000", "2018-07-16 09:30:00", "2018-
   07-16 10:30:00");
3.
4. if (dl.status == 0) //判断查询是否成功
5. {
6.     //遍历行情数组
7.     foreach(var v in dl.data)
8.     {
9.         Console.WriteLine("{0} {1}", v.symbol, v.price);
10.    }
11. }
```

# GMData

- [GMData 类定义](#)
- [使用举例](#)

## GMData 类定义

**GMData** 模板类是所有返回类数据函数的标准返回， 表示一个对象数据。类声明如下：

```
1. public class GMData<T>
2. {
3.     public T data;           //数据
4.     public int status;       //状态码， 0表示成功， 其它表示错误码
5. }
```

## 使用举例

```
1. //获取深交所最新的代码信息
2. GMData<DataTable> ds = GMApi.GetInstruments(null, "SZSE");
3. if(ds.status == 0)
4. {
5.     var dt = ds.data;
6.     for (int i = 0; i < dt.Rows.Count; i++)
7.     {
8.         Console.WriteLine("{0},{1},{2} ", dt.Rows[i]["symbol"], dt.Rows[i]
["sec_level"], dt.Rows[i]["pre_close"]);
9.     }
10. }
```

# 数据类

- Tick - Tick结构
  - 报价 [Quote](#)
- Bar - Bar结构

## Tick - Tick结构

逐笔行情数据

```

1. public class Tick
2. {
3.     public string symbol;           //symbol
4.
5.     public DateTime createdAt;      //时间
6.     public float price;             //最新价
7.     public float open;             //开盘价
8.     public float high;             //最高价
9.     public float low;              //最低价
10.    public double cumVolume;        //成交总量
11.    public double cumAmount;        //成交总金额/最新成交额, 累计值
12.    public System.Int64 cumPosition; //合约持仓量(期), 累计值
13.    public double lastAmount;       //瞬时成交额
14.    public int lastVolume;          //瞬时成交量
15.    public int tradeType;           //交易类型, 对应多开, 多平等类型
16.    public double iopv;             //基金份额参考净值
17.    public Quote[] quotes;          //报价, 下标从0开始, 0-表示第一档, 1-表示第二档, 依
    次类推
18. };

```

报价 [Quote](#)

```

1. public class Quote
2. {
3.     public float bidPrice;          //本档委买价
4.     public System.Int64 bidVolume;  //本档委买量
5.     public float askPrice;          //本档委卖价
6.     public System.Int64 askVolume;  //本档委卖量
7. };

```

## Bar - Bar结构

bar数据是指各种频率的行情数据

```

1. public class Bar
2. {
3.     public string symbol;
4.     public DateTime bob;           //bar的开始时间
5.     public DateTime eob;          //bar的结束时间

```

```
6.     public float open;           //开盘价
7.     public float close;          //收盘价
8.     public float high;           //最高价
9.     public float low;            //<最低价
10.    public double volume;         //成交量
11.    public double amount;         //成交金额
12.    public float preClose;        //昨收盘价，只有日频数据赋值
13.
14.    public System.Int64 position;  //持仓量
15.    public string frequency;       //bar频度
16. };
```

# 交易类

- [Account](#) - 账户结构
- [AccountStatus](#) - 账户状态结构
- [PlaceOrders](#) - 下单参数结构
- [Order](#) - 委托结构
- [ExecRpt](#) - 回报结构
- [Cash](#) - 资金结构
- [Position](#) - 持仓结构
- [Indicator](#) - 绩效指标结构
- [Parameter](#) - 动态参数结构

## Account - 账户结构

```

1. public class Account
2. {
3.     public string accountId;           //账号ID
4.     public string accountName;        //账户登录名
5.     public string intro;              //账号描述
6.     public string comment;            //账号备注
7. };

```

## AccountStatus - 账户状态结构

```

1. public class AccountStatus
2. {
3.     public string accountId;           //账号ID
4.     public string accountName;        //账户登录名
5.     public int state;                  //账户状态
6.     public int errorCode;              //错误码
7.     public string errorMsg;            //错误信息
8. };

```

## PlaceOrders - 下单参数结构

```

1. public struct PlaceOrderReq
2. {
3.     public string symbol;              //交易标的
4.     public int volume;                 //下单数量
5.     public OrderSide side;             //委托方向
6.     public OrderType type;             //委托类型
7.     public PositionEffect positionEffect; //开平方向
8.     public double price;               //委托价格
9.     public OrderDuration orderDuration; //委托时间属性
10.    public OrderQualifier orderQualifier; //委托成交属性
11.    public string account;              //账号
12. }

```

## Order - 委托结构

```

1. public class Order
2. {
3.     public string strategyId;           //策略ID
4.     public string accountId;           //账号ID
5.     public string accountName;         //账户登录名
6.     public string clOrdId;             //委托客户端ID
7.     public string orderId;             //委托柜台ID
8.     public string exOrdId;             //委托交易所ID
9.     public string symbol;              //symbol
10.    public int side;                    //买卖方向, 取值参考enum OrderSide
11.    public int positionEffect;          //开平标志, 取值参考enum PositionEffect
12.    public int positionSide;            //持仓方向, 取值参考enum PositionSide
13.
14.    public int orderType;                //委托类型, 取值参考enum OrderType
15.    public int orderDuration;            //委托时间属性, 取值参考enum
16.    OrderDuration
17.    public int orderQualifier;           //委托成交属性, 取值参考enum
18.    OrderQualifier
19.    public int orderSrc;                 //委托来源, 取值参考enum OrderSrc
20.
21.    public int status;                   //委托状态, 取值参考enum OrderStatus
22.    public int ordRejReason;             //委托拒绝原因, 取值参考enum
23.    OrderRejectReason
24.    public string ordRejReasonDetail;    //委托拒绝原因描述
25.
26.    public double price;                 //委托价格
27.    public double stopPrice;             //委托止损/止盈触发价格
28.
29.    public int orderStyle;               //委托风格, 取值参考 enum OrderStyle
30.    public Int64 volume;                 //委托量
31.    public double value;                 //委托额
32.    public double percent;               //委托百分比
33.    public Int64 targetVolume;           //委托目标量
34.    public double targetValue;           //委托目标额
35.    public double targetPercent;         //委托目标百分比
36.
37.    public Int64 filledVolume;           //已成量
38.    public double filledVwap;            //已成均价
39.    public double filledAmount;          //已成金额
40.    public double filledCommission;      //已成手续费
41.
42.    public DateTime createdAt;            //委托创建时间
43.    public DateTime updatedAt;           //委托更新时间
44. };

```

## ExecRpt - 回报结构

```

1. public class ExecRpt
2. {
3.     public string strategyId;           //策略ID

```

```

4.     public string accountId;                //账号ID
5.     public string accountName;             //账户登录名
6.     public string clOrdId;                 //委托客户端ID
7.     public string orderId;                 //委托柜台ID
8.     public string execId;                  //委托回报ID
9.     public string symbol;                  //symbol
10.
11.     public int positionEffect;              //开平标志, 取值参考enum PositionEffect
12.     public int side;                       //买卖方向, 取值参考enum OrderSide
13.     public int ordRejReason;                //委托拒绝原因, 取值参考enum
      OrderRejectReason
14.     public string ordRejReasonDetail;       //委托拒绝原因描述
15.     public int execType;                   //执行回报类型, 取值参考enum ExecType
16.
17.     public double price;                    //委托成交价格
18.     public Int64 volume;                    //委托成交量
19.     public double amount;                   //委托成交金额
20.     public double commission;              //委托成交手续费
21.     public double cost;                     //委托成交成本金额
22.     public DateTime createdAt;              //回报创建时间
23.
24. };

```

## Cash - 资金结构

```

1. public class Cash
2. {
3.     public string accountId;                //账号ID
4.     public string accountName;             //账户登录名
5.
6.     public int currency;                    //币种
7.
8.     public double nav;                      //净值(cum_inout + cum_pnl + fpnl -
      cum_commission)
9.     public double pnl;                      //净收益(nav-cum_inout)
10.    public double fpnl;                      //浮动盈亏(sum(each position fpnl))
11.    public double frozen;                    //持仓占用资金
12.    public double orderFrozen;               //挂单冻结资金
13.    public double available;                 //可用资金
14.
15.    public double balance;                   //资金余额
16.
17.    public double cumInout;                  //累计出入金
18.    public double cumTrade;                  //累计交易额
19.    public double cumPnl;                    //累计平仓收益(没扣除手续费)
20.    public double cumCommission;             //累计手续费
21.
22.    public double lastTrade;                 //上一次交易额
23.    public double lastPnl;                   //上一次收益
24.    public double lastCommission;            //上一次手续费
25.    public double lastInout;                 //上一次出入金
26.    public int changeReason;                  //资金变更原因, 取值参考enum
      CashPositionChangeReason

```

```

27.     public string changeEventId;                //触发资金变更事件的ID
28.
29.     public DateTime createdAt;                  //资金初始时间
30.     public DateTime updatedAt;                  //资金变更时间
31.
32. };

```

## Position - 持仓结构

```

1. public class Position
2. {
3.     public string accountId;                    //账号ID
4.     public string accountName;                 //账户登录名
5.
6.     public string symbol;                      //symbol
7.     public int side;                           //持仓方向，取值参考enum PositionSide
8.     public Int64 volume;                       //总持仓量；昨持仓量(volume-volume_today)
9.     public Int64 volumeToday;                  //今日持仓量
10.    public double vwap;                         //持仓均价
11.    public double amount;                       //持仓额(volume*vwap*multiplier)
12.
13.    public double price;                        //当前行情价格
14.    public double fpnl;                         //持仓浮动盈亏((price-
    vwap)*volume*multiplier)
15.    public double cost;                         //持仓成本
    (vwap*volume*multiplier*margin_ratio)
16.    public Int64 orderFrozen;                   //挂单冻结仓位
17.    public Int64 orderFrozenToday;              //挂单冻结今仓位
18.    public Int64 available;                     //非挂单冻结仓位(volume-order_frozen); 可平昨
    仓位(available-available_today)
19.    public Int64 availableToday;                //非挂单冻结今仓位(volume_today-
    order_frozen_today)
20.    public Int64 availableNow;                  //当前可平仓位
21.    public double marketValue;                 //持仓市值
22.
23.    public double lastPrice;                    //上一次成交价
24.    public Int64 lastVolume;                    //上一次成交量
25.    public Int64 lastInout;                     //上一次出入持仓量
26.    public int changeReason;                    //仓位变更原因，取值参考enum
    CashPositionChangeReason
27.    public string changeEventId;                //触发资金变更事件的ID
28.
29.    public int hasDividend;                     //持仓区间有分红配送
30.    public DateTime createdAt;                  //建仓时间（实盘不支持）
31.    public DateTime updatedAt;                  //仓位变更时间（实盘不支持）
32.
33. };

```

## Indicator - 绩效指标结构

```

1. public class Indicator

```



```

2. {
3.     public string accountId;           //账号ID
4.     public double pnlRatio;           //累计收益率(pnl/cum_inout)
5.     public double pnlRatioAnnual;     //年化收益率
6.     public double sharpRatio;        //夏普比率
7.     public double maxDrawdown;       //最大回撤
8.     public double riskRatio;         //风险比率
9.     public int openCount;            //开仓次数
10.    public int closeCount;           //平仓次数
11.    public int winCount;             //盈利次数
12.    public int loseCount;            //亏损次数
13.    public double winRatio;          //胜率
14.
15.    public DateTime createdAt;        //指标创建时间
16.    public DateTime updatedAt;       //指标变更时间
17. };

```

## Parameter - 动态参数结构

```

1.
2. public class Parameter
3. {
4.     public string key;               //参数键
5.     public double value;            //参数值
6.     public double min;              //可设置的最小值
7.     public double max;              //可设置的最大值
8.     public string name;             //参数名
9.     public string intro;            //参数说明
10.    public string group;             //组名
11.    public bool readonlyFlag;        //是否只读
12. };

```

## 枚举常量

- `OrderStatus` - 委托状态
- `OrderSide` - 委托方向
- `SecType` - 标的类别
- `OrderType` - 委托类型
- `ExecType` - 执行回报类型
- `PositionEffect` - 开平仓类型
- `PositionSide` - 持仓方向
- `OrderRejectReason` - 订单拒绝原因
- `CashPositionChangeReason` - 仓位变更原因
- `OrderDuration` - 委托时间属性
- `OrderQualifier` - 委托成交属性
- `AccountState` - 交易账户状态
- `StrategyMode` - 策略模式
- `Adjust` - 复权方式

### OrderStatus - 委托状态

```

1. enum OrderStatus
2. {
3.     OrderStatus_Unknown = 0,
4.     OrderStatus_New = 1,           //已报
5.     OrderStatus_PartiallyFilled = 2, //部成
6.     OrderStatus_Filled = 3,        //已成
7.     OrderStatus_Canceled = 5,      //已撤
8.     OrderStatus_PendingCancel = 6, //待撤
9.     OrderStatus_Rejected = 8,      //已拒绝
10.    OrderStatus_Suspended = 9,      //挂起
11.    OrderStatus_PendingNew = 10,     //待报
12.    OrderStatus_Expired = 12,       //已过期
13.
14. };

```

### OrderSide - 委托方向

```

1. enum OrderSide
2. {
3.     OrderSide_Unknown = 0,
4.     OrderSide_Buy = 1,           //买入
5.     OrderSide_Sell = 2,         //卖出
6. };

```

### SecType - 标的类别

```

1. enum SecType
2. {
3.     SEC_TYPE_STOCK = 1,         //股票

```

```

4. SEC_TYPE_FUND = 2,           //基金
5. SEC_TYPE_INDEX = 3,         //指数
6. SEC_TYPE_FUTURE = 4,        //期货
7. SEC_TYPE_OPTION = 5,        //期权
8. SEC_TYPE_CONFUTURE = 10     //虚拟合约
9. }

```

## OrderType - 委托类型

```

1. enum OrderType
2. {
3.     OrderType_Unknown = 0,
4.     OrderType_Limit = 1,      //限价委托
5.     OrderType_Market = 2,     //市价委托
6.     OrderType_Stop = 3,       //止损止盈委托
7. };

```

## ExecType - 执行回报类型

```

1. enum ExecType
2. {
3.     ExecType_Unknown = 0,
4.     ExecType_New = 1,         //已报
5.     ExecType_Canceled = 5,    //已撤销
6.     ExecType_PendingCancel = 6, //待撤销
7.     ExecType_Rejected = 8,    //已拒绝
8.     ExecType_Suspended = 9,   //挂起
9.     ExecType_PendingNew = 10,  //待报
10.    ExecType_Expired = 12,     //过期
11.    ExecType_Trade = 15,       //成交
12.    ExecType_OrderStatus = 18, //委托状态
13.    ExecType_CancelRejected = 19, //撤单被拒绝
14. };

```

## PositionEffect - 开平仓类型

```

1. enum PositionEffect
2. {
3.     PositionEffect_Unknown = 0,
4.     PositionEffect_Open = 1,   //开仓
5.     PositionEffect_Close = 2,  //平仓,具体语义取决于对应的交易所
6.     PositionEffect_CloseToday = 3, //平今仓
7.     PositionEffect_CloseYesterday = 4, //平昨仓
8. };

```

## PositionSide - 持仓方向

```

1. enum PositionSide
2. {
3.     PositionSide_Unknown = 0,
4.     PositionSide_Long = 1,      //多方向
5.     PositionSide_Short = 2,     //空方向
6. };

```

## OrderRejectReason - 订单拒绝原因

```

1. enum OrderRejectReason
2. {
3.     OrderRejectReason_Unknown = 0,                //未知原因
4.     OrderRejectReason_RiskRuleCheckFailed = 1,     //不符合风控规则
5.     OrderRejectReason_NoEnoughCash = 2,           //资金不足
6.     OrderRejectReason_NoEnoughPosition = 3,        //仓位不足
7.     OrderRejectReason_IllegalAccountId = 4,        //非法账户ID
8.     OrderRejectReason_IllegalStrategyId = 5,       //非法策略ID
9.     OrderRejectReason_IllegalSymbol = 6,          //非法交易代码
10.    OrderRejectReason_IllegalVolume = 7,           //非法委托量
11.    OrderRejectReason_IllegalPrice = 8,            //非法委托价
12.    OrderRejectReason_AccountDisabled = 10,        //交易账号被禁止交易
13.    OrderRejectReason_AccountDisconnected = 11,    //交易账号未连接
14.    OrderRejectReason_AccountLoggedout = 12,       //交易账号未登录
15.    OrderRejectReason_NotInTradingSession = 13,    //非交易时段
16.    OrderRejectReason_OrderTypeNotSupported = 14,  //委托类型不支持
17.    OrderRejectReason_Throttle = 15,              //流控限制
18.    OrderRejectReason_SymbolSuspended = 16,        //交易代码停牌
19.    OrderRejectReason_Internal = 999,              //内部错误
20.
21.    CancelOrderRejectReason_OrderFinalized = 101,   //委托已完成
22.    CancelOrderRejectReason_UnknownOrder = 102,    //未知委托
23.    CancelOrderRejectReason_BrokerOption = 103,    //柜台设置
24.    CancelOrderRejectReason_AlreadyInPendingCancel = 104, //委托撤销中
25. };

```

## CashPositionChangeReason - 仓位变更原因

```

1. enum CashPositionChangeReason
2. {
3.     CashPositionChangeReason_Unknown = 0,
4.     CashPositionChangeReason_Trade = 1,           //交易
5.     CashPositionChangeReason_Inout = 2,           //出入金/出入持仓
6.     CashPositionChangeReason_Dividend = 3,        //分红送股
7. };

```

## OrderDuration - 委托时间属性

```

1. enum OrderDuration

```

```

2. {
3.     OrderDuration_Unknown = 0,
4.     OrderDuration_FAK = 1, //即时成交剩余撤销(fill and kill)
5.     OrderDuration_FOK = 2, //即时全额成交或撤销(fill or kill)
6.     OrderDuration_GFD = 3, //当日有效(good for day)
7.     OrderDuration_GFS = 4, //本节有效(good for section)
8.     OrderDuration_GTD = 5, //指定日期前有效(good till date)
9.     OrderDuration_GTC = 6, //撤销前有效(good till cancel)
10.    OrderDuration_GFA = 7, //集合竞价前有效(good for auction)
11. };

```

## OrderQualifier - 委托成交属性

```

1. enum OrderQualifier
2. {
3.     OrderQualifier_Unknown = 0,
4.     OrderQualifier_BOC = 1, //对方最优价格(best of counterparty)
5.     OrderQualifier_BOP = 2, //己方最优价格(best of party)
6.     OrderQualifier_B5TC = 3, //最优五档剩余撤销(best 5 then cancel)
7.     OrderQualifier_B5TL = 4, //最优五档剩余转限价(best 5 then limit)
8. };

```

## AccountState - 交易账户状态

```

1.
2. enum AccountState
3. {
4.     State_UNKNOWN = 0, //未知
5.     State_CONNECTING = 1, //连接中
6.     State_CONNECTED = 2, //已连接
7.     State_LOGGEDIN = 3, //已登录
8.     State_DISCONNECTING = 4, //断开中
9.     State_DISCONNECTED = 5, //已断开
10.    State_ERROR = 6 //错误
11. };

```

## StrategyMode - 策略模式

```

1.
2. public enum StrategyMode
3. {
4.     MODE_UNDEF = 0, //未定义, 策略不会运行
5.     MODE_LIVE = 1, //实盘与仿真模式
6.     MODE_BACKTEST = 2 //回测模式
7. };

```

## Adjust - 复权方式

```
1. public enum Adjust
2. {
3.     ADJUST_NONE = 0,          //(不复权)
4.     ADJUST_PREV = 1,         //(前复权)
5.     ADJUST_POST = 2          //(后复权)
6. }
```

# 错误码

错误码	描述
0	成功
1000	错误或无效的token
1001	无法连接到终端服务
1010	无法获取掘金服务器地址列表
1011	消息包解析错误
1012	网络消息包解析错误
1013	交易服务调用错误
1014	历史行情服务调用错误
1015	策略服务调用错误
1016	动态参数调用错误
1017	基本面数据服务调用错误
1018	回测服务调用错误
1019	交易网关服务调用错误
1020	无效的ACCOUNT_ID
1021	非法日期格式
1100	交易消息服务连接失败
1101	交易消息服务断开
1200	实时行情服务连接失败
1201	实时行情服务连接断开
1300	初始化回测失败，可能是终端未启动或无法连接到终端
1301	回测时间区间错误
1302	回测读取缓存数据错误
1303	回测写入缓存数据错误