

beego开发文档

皇虫



快速开始

重要概念

- **symbol** - 代码标识
 - 交易所代码
 - 交易标代码
- **mode** - 模式选择
 - 实时模式
 - 回测模式
- **context** - 上下文对象
 - **context.symbols** - 订阅代码集合
 - **context.now** - 当前时间
 - **context.data(symbol, frequency, count, fields)** - 数据滑窗
 - **context.account(account_id=None)** - 账户信息
 - **context.parameters** - 动态参数

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6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

交易标代码

交易标代码是指交易所给出的交易标的代码, 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

mode - 模式选择

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。

context - 上下文对象

context是策略运行上下文环境对象, 该对象将会在你的算法策略的任何方法之间做传递。

除了系统提供的, 用户也可以根据需求自己定义无限多种自己随后需要的属性

context.symbols - 订阅代码集合

返回用户订阅过的代码集合

context.now - 当前时间

实时模式返回当前本地时间, 回测模式返回当前回测时间

返回数据类型为datetime.datetime

context.data(symbol, frequency, count, fields) - 数据滑窗

订阅[bar](#)滑窗, frequency:频率, count:滑窗大小, fields : 所需bar的字段 , 如有多属性, 中间用 `,` 隔开

返回数据类型为dataframe

context.account(account_id=None) - 账户信息

账户信息 , 默认返回默认账户, 如多个账户需指定account_id

返回类型为[account](#) - 账户对象。

context.parameters - 动态参数

返回数据类型为字典, key为动态参数的key, 值为动态参数对象 , 参见[动态参数设置](#)

API介绍

枚举常量

- [OrderStatus](#) - 委托状态
- [OrderSide](#) - 委托方向
- [OrderType](#) - 委托类型
- [OrderDuration](#) - 委托时间属性
- [OrderQualifier](#) - 委托成交属性
- [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
- [CancelOrderRejectReason](#) - 取消订单拒绝原因
- [OrderStyle](#) - 订单类型
- [CashPositionChangeReason](#) - 仓位变更原因
- [SecType](#) - 标的类别

OrderStatus - 委托状态

1. <code>OrderStatus_Unknown</code>	<code>= 0</code>	
2. <code>OrderStatus_New</code>	<code>= 1</code>	# 已报
3. <code>OrderStatus_PartiallyFilled</code>	<code>= 2</code>	# 部成
4. <code>OrderStatus_Filled</code>	<code>= 3</code>	# 已成
5. <code>OrderStatus_DoneForDay</code>	<code>= 4</code>	#
6. <code>OrderStatus_Canceled</code>	<code>= 5</code>	# 已撤
7. <code>OrderStatus_PendingCancel</code>	<code>= 6</code>	# 待撤
8. <code>OrderStatus_Stopped</code>	<code>= 7</code>	#
9. <code>OrderStatus_Rejected</code>	<code>= 8</code>	# 已拒绝
10. <code>OrderStatus_Suspended</code>	<code>= 9</code>	# 挂起
11. <code>OrderStatus_PendingNew</code>	<code>= 10</code>	# 待报
12. <code>OrderStatus_Calculated</code>	<code>= 11</code>	#
13. <code>OrderStatus_Expired</code>	<code>= 12</code>	# 已过期
14. <code>OrderStatus_AcceptedForBidding</code>	<code>= 13</code>	#
15. <code>OrderStatus_PendingReplace</code>	<code>= 14</code>	#

OrderSide - 委托方向

1. OrderSide_Unknown = 0
2. OrderSide_Buy = 1 # 买入
3. OrderSide_Sell = 2 # 卖出

OrderType - 委托类型

1. OrderType_Unknown = 0
2. OrderType_Limit = 1 # 限价委托
3. OrderType_Market = 2 # 市价委托
4. OrderType_Stop = 3 # 止损止盈委托

OrderDuration - 委托时间属性

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

OrderQualifier - 委托成交属性

1. OrderQualifier_Unknown = 0
2. OrderQualifier_BOC = 1 # 对方最优价格(best of counterparty)


```
3. OrderQualifier_BOP      = 2          # 己方最优价格(best of
    party)
4. OrderQualifier_B5TC     = 3          # 最优五档剩余撤销(best 5
    then cancel)
5. OrderQualifier_B5TL     = 4          # 最优五档剩余转限价(best
    5 then limit)
```

ExecType - 执行回报类型

```
1. ExecType_Unknown = 0
2. ExecType_New = 1          # 已报
3. ExecType_DoneForDay = 4    #
4. ExecType_Canceled = 5     # 已撤销
5. ExecType_PendingCancel = 6 # 待撤销
6. ExecType_Stopped = 7      #
7. ExecType_Rejected = 8     # 已拒绝
8. ExecType_Suspended = 9    # 挂起
9. ExecType_PendingNew = 10   # 待报
10. ExecType_Calculated = 11  #
11. ExecType_Expired = 12    # 过期
12. ExecType_Restated = 13   #
13. ExecType_PendingReplace = 14 #
14. ExecType_Trade = 15      # 成交
15. ExecType_TradeCorrect = 16 #
16. ExecType_TradeCancel = 17 #
17. ExecType_OrderStatus = 18 # 委托状态
18. ExecType_CancelRejected = 19 # 撤单被拒绝
```

PositionEffect - 开平仓类型

```
1. PositionEffect_Unknown = 0
2. PositionEffect_Open = 1    # 开仓
3. PositionEffect_Close = 2   # 平仓，具体语义取决于
    对应的交易所
```

- | | |
|---|-------|
| 4. <code>PositionEffect_CloseToday = 3</code> | # 平今仓 |
| 5. <code>PositionEffect_CloseYesterday = 4</code> | # 平昨仓 |

PositionSide - 持仓方向

- | | |
|--|-------|
| 1. <code>PositionSide_Unknown = 0</code> | |
| 2. <code>PositionSide_Long = 1</code> | # 多方向 |
| 3. <code>PositionSide_Short = 2</code> | # 空方向 |

OrderRejectReason - 订单拒绝原因

- | | |
|---|-------------|
| 1. <code>OrderRejectReason_Unknown = 0</code> | # 未知原因 |
| 2. <code>OrderRejectReason_RiskRuleCheckFailed = 1</code> | # 不符合风控规则 |
| 3. <code>OrderRejectReason_NoEnoughCash = 2</code> | # 资金不足 |
| 4. <code>OrderRejectReason_NoEnoughPosition = 3</code> | # 仓位不足 |
| 5. <code>OrderRejectReason_IllegalAccountId = 4</code> | # 非法账户ID |
| 6. <code>OrderRejectReason_IllegalStrategyId = 5</code> | # 非法策略ID |
| 7. <code>OrderRejectReason_IllegalSymbol = 6</code> | # 非法交易标的 |
| 8. <code>OrderRejectReason_IllegalVolume = 7</code> | # 非法委托量 |
| 9. <code>OrderRejectReason_IllegalPrice = 8</code> | # 非法委托价 |
| 10. <code>OrderRejectReason_AccountDisabled = 10</code> | # 交易账号被禁止交易 |
| 11. <code>OrderRejectReason_AccountDisconnected = 11</code> | # 交易账号未连接 |
| 12. <code>OrderRejectReason_AccountLoggedout = 12</code> | # 交易账号 |

- | | | |
|--|--|-----------|
| 号未登录 | | |
| 13. OrderRejectReason_NotInTradingSession = 13 | | # 非交易时段 |
| 14. OrderRejectReason_OrderTypeNotSupported = 14 | | # 委托类型不支持 |
| 15. OrderRejectReason_Throttle = 15 | | # 流控限制 |

CancelOrderRejectReason - 取消订单拒绝原因

- | | | |
|---|--|---------|
| 1. CancelOrderRejectReason_OrderFinalized = 101 | | # 委托已完成 |
| 2. CancelOrderRejectReason_UnknownOrder = 102 | | # 未知委托 |
| 3. CancelOrderRejectReason_BrokerOption = 103 | | # 柜台设置 |
| 4. CancelOrderRejectReason_AlreadyInPendingCancel = 104 | | # 委托撤销中 |

OrderStyle - 订单类型

- | | | |
|---------------------------------|--|-------------|
| 1. OrderStyle_Unknown = 0 | | |
| 2. OrderStyle_Volume = 1 | | # 按指定量委托 |
| 3. OrderStyle_Value = 2 | | # 按指定价值委托 |
| 4. OrderStyle_Percent = 3 | | # 按指定比例委托 |
| 5. OrderStyle_TargetVolume = 4 | | # 调仓到目标持仓量 |
| 6. OrderStyle_TargetValue = 5 | | # 调仓到目标持仓额 |
| 7. OrderStyle_TargetPercent = 6 | | # 调仓到目标持仓比例 |

CashPositionChangeReason - 仓位变更原因

1. `CashPositionChangeReason_Unknown = 0`
2. `CashPositionChangeReason_Trade = 1` # 交易
3. `CashPositionChangeReason_Inout = 2` # 出入金 / 出入持仓

SecType - 标的类别

1. `SEC_TYPE_STOCK = 1` # 股票
2. `SEC_TYPE_FUND = 2` # 基金
3. `SEC_TYPE_INDEX = 3` # 指数
4. `SEC_TYPE_FUTURE = 4` # 期货
5. `SEC_TYPE_OPTION = 5` # 期权
6. `SEC_TYPE_CONFUTURE = 10` # 虚拟合约

错误码

错误码	描述
0	成功
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

1.2.3 错误码

数据结构

快速开始 - 如何使用本文档

啊啊啊这是重新做了编辑

感谢您选择掘金量化!

如果您之前有量化的经验, 那么您可以直接阅览快速创建我的策略来熟悉平台的使用.

如果您之前没有经验, 也可以先阅览快速创建我的策略来了解掘金支持的策略模式.

紧接着, 我们介绍了掘金量化中重要概念和数据结构, 您可以熟悉下这些内容, 方便策略的编写与研究.

在后面我们详细介绍了掘金量化的python sdk, 介绍了平台上可供使用的、丰富多样的API.

我们把枚举常量和错误码放在在最后, 您在用到的时候可以随时查阅.

祝您使用愉快!

快速开始 - 快速创建我的策略

- [定时任务](#)
- [数据事件驱动](#)
- [时间序列数据事件驱动](#)
- [多个代码数据事件驱动](#)
- [默认账户交易](#)
- [指定账户交易](#)
- [回测模式与实时模式](#)
 - [回测模式](#)
 - [实时模式](#)
- [提取数据研究](#)

定时任务

以下的范例代码片段是一个非常简单的例子，在每个交易日的14:50:00 市价购买200股浦发银行股票：

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3.
4. from gm.api import *
5.
6. def init(context):
7.     schedule(schedule_func=algo, date_rule='1d',
8.               time_rule='14:50:00')
9.
10. def algo(context):
11.     # 购买200股浦发银行股票
12.     order_volume(symbol='SHSE.600000', volume=200,
13.                  side=OrderSide_Buy, order_type=OrderType_Market,
14.                  position_effect=PositionEffect_Open, price=0)
15. if __name__ == '__main__':
```

3. 快速开始 - 快速创建我的策略

```
16.     run(strategy_id='strategy_1', filename='main.py',
           mode=MODE_BACKTEST, token='token_id',
17.         backtest_start_time='2016-06-17 13:00:00',
           backtest_end_time='2017-08-21 15:00:00')
```

整个策略需要三步:

1. 设置初始化函数: `init`, 使用 `schedule` 函数进行定时任务配置
2. 配置任务, 到点会执行该任务
3. 执行策略

数据事件驱动

策略订阅的每个代码的每一个bar, 都会触发策略逻辑

以下的范例代码片段是一个非常简单的例子, 订阅浦发银行的日线bar, bar数据的更新会自动触发on_bar的调用:

```
1.  # coding=utf-8
2.  from __future__ import print_function, absolute_import,
   unicode_literals
3.  from gm.api import *
4.
5.
6.  def init(context):
7.      subscribe(symbols='SHSE.600000', frequency='1d')
8.
9.
10. def on_bar(context, bars):
11.     # 打印当前获取的bar信息
12.     bar = bars[0]
13.     # 执行策略逻辑操作
14.     print(bar)
15.
16. if __name__ == '__main__':
17.     run(strategy_id='strategy_1', filename='main.py',
           mode=MODE_BACKTEST, token='token_id',
18.         backtest_start_time='2016-06-17 13:00:00',
```

3. 快速开始 - 快速创建我的策略

```
backtest_end_time='2017-08-21 15:00:00')
```

整个策略需要三步:

1. 设置初始化函数: init, 使用subscribe函数进行数据订阅
2. 实现一个函数: on_bar, 来根据数据推送进行逻辑处理
3. 执行策略

时间序列数据事件驱动

策略订阅代码时指定数据窗口大小与周期, 平台创建数据滑动窗口, 加载初始数据, 并在新的bar到来时自动刷新数据。

on_bar事件触发时, 策略可以取到订阅代码的准备好的时间序列数据。

以下的范例代码片段是一个非常简单的例子, 订阅浦发银行的日线bar, bar数据的更新会自动触发on_bar的调用, 每次调用 `context.data` 来获取最新的50条日线bar信息:

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
5.
6. def init(context):
7.     subscribe(symbols='SHSE.600000', frequency='1d', count=50)
8.
9.
10. def on_bar(context, bars):
11.     print(context.data(symbol='SHSE.600000', frequency='1d',
12. count=50, fields='close,bob'))
13.
14. if __name__ == '__main__':
15.     run(strategy_id='strategy_1', filename='main.py',
16. mode=MODE_BACKTEST, token='token_id',
17. backtest_start_time='2016-06-17 13:00:00',
18. backtest_end_time='2017-08-21 15:00:00')
```

3. 快速开始 - 快速创建我的策略

整个策略需要三步:

1. 设置初始化函数: `init`, 使用 `subscribe` 函数进行数据订阅
2. 实现一个函数: `on_bar`, 来根据数据推送进行逻辑处理, 通过 `context.data` 获取数据滑窗
3. 执行策略

多个代码数据事件驱动

策略订阅多个代码, 并且要求同一频度的数据到齐后, 再触发事件。

以下的范例代码片段是一个非常简单的例子, 订阅浦发银行和平安银行的日线bar, 在浦发银行bar和平安银行bar到齐后会自动触发`on_bar`的调用:

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
5.
6. def init(context):
7.     subscribe(symbols='SHSE.600000,SZSE.000001',
8.               frequency='1d', count=5, wait_group=True)
9.
10. def on_bar(context, bars):
11.     for bar in bars:
12.         print(bar['symbol'], bar['eob'])
13.
14.
15. if __name__ == '__main__':
16.     run(strategy_id='strategy_1', filename='main.py',
17.         mode=MODE_BACKTEST, token='token_id',
18.         backtest_start_time='2016-06-17 13:00:00',
19.         backtest_end_time='2017-08-21 15:00:00')
```

整个策略需要三步:

3. 快速开始 - 快速创建我的策略

1. 设置初始化函数: `init`, 使用 `subscribe` 函数进行数据订阅多个代码, 设置 `wait_group=True`
2. 实现一个函数: `on_bar`, 来根据数据推送(多个代码行情)进行逻辑处理
3. 执行策略

默认账户交易

如果策略只关联一个账户, 该账户是策略的默认账户。

进行交易时可以不指定交易账户, 以默认账户进行交易。

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
5.
6. def init(context):
7.     subscribe(symbols='SHSE.600000,SZSE.000001',
8.               frequency='1d')
9.
10. def on_bar(context, bars):
11.     for bar in bars:
12.         order_volume(symbol=bar['symbol'], volume=200,
13.                       side=OrderSide_Buy,
14.                       order_type=OrderType_Market,
15.                       position_effect=PositionEffect_Open,
16.                       price=0)
17.
18. if __name__ == '__main__':
19.     run(strategy_id='strategy_1', filename='main.py',
20.         mode=MODE_BACKTEST, token='token_id',
21.         backtest_start_time='2016-06-17 13:00:00',
22.         backtest_end_time='2017-08-21 15:00:00')
```

指定账户交易

3. 快速开始 - 快速创建我的策略

如果策略在多个账户交易，需要显式指定交易账户。

以下的范例代码片段是一个非常简单的例子，在下单时使用指定账户 `account='account_1'`（参数为账户名称或账户ID）来进行交易：

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
5.
6. def init(context):
7.     subscribe(symbols='SHSE.600000,SZSE.000001',
8. frequency='1m')
9.
10. def on_bar(context, bars):
11.     for bar in bars:
12.         order_volume(symbol=bar['symbol'], volume=200,
13. side=OrderSide_Buy,
14. order_type=OrderType_Market,
15. position_effect=PositionEffect_Open, price=0,
16. account='account_1')
17.
18. if __name__ == '__main__':
19.     run(strategy_id='strategy_1', filename='main.py',
20. mode=MODE_BACKTEST, token='token_id',
21. backtest_start_time='2016-06-17 13:00:00',
22. backtest_end_time='2017-08-21 15:00:00')
```

回测模式与实时模式

掘金3策略只有两种模式, 回测模式(backtest)与实时模式(live)。在加载策略时指定mode参数。

回测模式

3. 快速开始 - 快速创建我的策略

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3.
4. run(strategy_id='strategy_1', filename='main.py',
   mode=MODE_BACKTEST, token='token_id',
5.      backtest_start_time='2016-06-17 13:00:00',
   backtest_end_time='2017-08-21 15:00:00')
```

`mode=MODE_BACKTEST` 表示回测模式。

实时模式

掘金3策略只有两种模式, 回测模式(backtest)与实时模式(live)。在加载策略时指定mode参数。

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
5. run(strategy_id='strategy_id', filename='main.py',
   mode=MODE_LIVE, token='token_id')
```

`mode=MODE_LIVE` 表示实时模式。

提取数据研究

如果只想提取数据, 无需实时数据驱动策略, 无需交易下单可以直接通过数据查询函数来进行查询。

```
1. # coding=utf-8
2. from __future__ import print_function, absolute_import,
   unicode_literals
3. from gm.api import *
4.
```

3. 快速开始 - 快速创建我的策略

```
5. set_token('xxxx')
6. data = history(symbol='SHSE.600000', frequency='1d',
    start_time='2015-01-01', end_time='2015-12-31',
    fields='open,high,low,close')
```

整个过程只需要两步:

1. set_token 设置用户token , 如果token不正确, 函数调用会抛出异常
2. 调用数据查询函数 , 直接进行数据查询

数据结构 - 数据类

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

Tick - Tick对象

逐笔行情数据

参数名	类型	说明
symbol	str	标的代码
open	float	开盘价
high	float	最高价
low	float	最低价
price	float	最新价
cum_volume	float	成交总量/最新成交量,累计值
cum_amount	float	成交总金额/最新成交额,累计值
trade_type	int	交易类型 1: '双开' , 2: '双平' , 3: '多开' , 4: '空开' , 5: '空平' , 6: '多平' , 7: '多换' , 8: '空换'
last_volume	int	瞬时成交额
cum_position	int	合约持仓量(期),累计值
last_amount	float	瞬时成交额
created_at	datetime.datetime	创建时间
quotes	list[quote]	买卖1-5档 (如下)

Quote - 报价

参数名	类型	说明
bid_p	float	买一价

4. 数据结构 - 数据类

bid_v	int	买一量
ask_p	float	卖一价
ask_v	int	卖一量

Bar - Bar对象

bar数据是指各种频率的行情数据

参数名	类型	说明
symbol	str	标的代码
frequency	str	频率, 支持多种频率, 具体见 股票行情数据 和 期货行情数据
open	float	开盘价
close	float	收盘价
high	float	最高价
low	float	最低价
amount	float	成交额
volume	float	成交量
position	long	持仓量
pre_close	float	前收盘价
bob	datetime.datetime	bar开始时间
eob	datetime.datetime	bar结束时间

数据结构 - 交易类

- [Account](#) - 账户对象
- [Order](#) - 委托对象
- [ExecRpt](#) - 回报对象
- [Cash](#) - 资金对象
- [Position](#) - 持仓对象
- [Indicator](#) - 绩效指标对象

Account - 账户对象

属性	类型	说明
id	str	账户id
title	str	账户标题
cash	dict	资金字典
positions(symbol=' ', side=None)	list	持仓情况 列表, 默认全部持仓, 可根据 symbol, side参数缩小查询范围
position(symbol, side)	dict	持仓情况

Order - 委托对象

属性	类型	说明
strategy_id	str	策略ID
account_id	str	账号ID
account_name	str	账户登录名
cl_ord_id	str	委托客户端ID
order_id	str	委托柜台ID
ex_ord_id	str	委托交易所ID
symbol	str	标的代码

5. 数据结构 - 交易类

side	int	买卖方向 取值参考 OrderSide
position_effect	int	开平标志 取值参考 PositionEffect
position_side	int	持仓方向 取值参考 PositionSide
order_type	int	委托类型 取值参考 OrderType
order_duration	int	委托时间属性 取值参考 OrderDuration
order_qualifier	int	委托成交属性 取值参考 OrderQualifier
order_src	int	委托来源
status	int	委托状态 取值参考 OrderStatus
ord_rej_reason	int	委托拒绝原因 取值参考 OrderRejejectReason
ord_rej_reason_detail	str	委托拒绝原因描述
price	float	委托价格
stop_price	float	委托止损/止盈触发价格
order_style	int	委托风格 取值参考 OrderStyle
volume	int	委托量
value	int	委托额
percent	float	委托百分比
target_volume	int	委托目标量
target_value	int	委托目标额
target_percent	float	委托目标百分比
filled_volume	int	已成量
filled_vwap	float	已成均价
filled_amount	float	已成金额

created_at	datetime.datetime	委托创建时间
updated_at	datetime.datetime	委托更新时间

ExecRpt - 回报对象

属性	类型	说明
strategy_id	str	策略ID
account_id	str	账号ID
account_name	str	账户登录名
cl_ord_id	str	委托客户端ID
order_id	str	委托柜台ID
ex_ord_id	str	委托交易所ID
position_effect	int	开平标志 取值参考 PositionEffect
side	int	买卖方向 取值参考 OrderSide
ord_rej_reason	int	委托拒绝原因 取值参考 OrderRejectReason
ord_rej_reason_detail	str	委托拒绝原因描述
exec_type	int	执行回报类型 取值参考 ExecType
price	float	委托成交价格
volume	int	委托成交量
amount	float	委托成交金额
created_at	datetime.datetime	回报创建时间

Cash - 资金对象

属性	类型	说明
account_id	str	账号ID

account_name	str	账户登录名
currency	int	币种
nav	float	净值
pnl	float	净收益
fpnl	float	浮动盈亏
frozen	float	持仓占用资金
order_frozen	float	挂单冻结资金
available	float	可用资金
cum_inout	float	累计出入金
cum_trade	float	累计交易额
cum_pnl	float	累计平仓收益(没扣除手续费)
cum_commission	float	累计手续费
last_trade	float	上一次交易额
last_pnl	float	上一次收益
last_commission	float	上一次手续费
last_inout	float	上一次出入金
change_reason	int	资金变更原因 取值参考 CashPositionChangeReason
change_event_id	str	触发资金变更事件的ID
created_at	datetime.datetime	资金初始时间
updated_at	datetime.datetime	资金变更时间

Position - 持仓对象

属性	类型	说明
account_id	str	账号ID
account_name	str	账户登录名
symbol	str	标的代码

5. 数据结构 - 交易类

side	int	持仓方向 取值参考 PositionSide
volume	int	总持仓量; 昨持仓量 $(\text{volume} - \text{volume_today})$
volume_today	int	今日持仓量
vwap	float	持仓均价
amount	float	持仓额 $(\text{volume} * \text{vwap} * \text{multiplier})$
price	float	当前行情价格
fpnl	float	持仓浮动盈亏 $((\text{price} - \text{vwap}) * \text{volume} * \text{multiplier})$
cost	float	持仓成本 $(\text{vwap} * \text{volume} * \text{multiplier} * \text{margin_ratio})$
order_frozen	int	挂单冻结仓位
order_frozen_today	int	挂单冻结今仓仓位
available	int	可平总仓位 $(\text{volume} - \text{order_frozen})$; 可平昨仓位 $(\text{available} - \text{available_today})$
available_today	int	可平今仓位 $(\text{volume_today} - \text{order_frozen_today})$
last_price	float	上一次成交价
last_volume	int	上一次成交量
last_inout	int	上一次出入持仓量
change_reason	int	仓位变更原因, 取值参考 CashPositionChangeReason
change_event_id	str	触发资金变更事件的ID
created_at	datetime.datetime	建仓时间
updated_at	datetime.datetime	仓位变更时间

Indicator - 绩效指标对象

属性	类型	说明
account_id	str	账号ID
pnl_ratio	double	累计收益率(pnl/cum_inout)
pnl_ratio_annual	double	年化收益率
sharp_ratio	double	夏普比率
max_drawdown	double	最大回撤
risk_ratio	double	风险比率
open_count	int	开仓次数
close_count	int	平仓次数
win_count	int	盈利次数
lose_count	int	亏损次数
win_ratio	double	胜率
created_at	datetime.datetime	指标创建时间
updated_at	datetime.datetime	指标变更时间

< >