掘金量化帮助中心

快速开始

下载SDK

掘金量化平台提供策略开发服务包(SDK)用于策略开发者实现自己的策略。SDK下载地址请点击这里。Python SDK支持Windows + Python2.7/3.6 + 32位/64位、Linux Python2/3 x64共六种版本，下载时找到对应本地安装的Python版本的SDK包。需要注意的是，此处所指的32位/64位不是指系统的版本，而是本地Python的版本。

安装配置

- 在Windows系统，需要解压下载的SDK包，双击安装程序进行安装。在安装过程中，安装程序会自行寻找Python安装路径，若提示找不到Python，需要手动添加注册信息或重新安装Python。若Python3.6不是安装在个人文件夹下，需要以管理员权限运行SDK安装程序。
- 在Linux系统，解压ZIP文件，运行`pip install xxxx.whl`命令进行安装；python3使用`pip3 install xxxxx.whl`。Wheel文件名称以发布为准。

我的第一个策略

方式一，直接使用例子运行

解压SDK包，使用examples中的例子项目。

方式二，创建自己的策略

1. 参考这里注册账号，登录终端。
2. 参考这里创建策略基本架构。编程语言选择Python，订阅上交所浦发银行(证券代码：600000)的Tick行情。
3. 编写自己的策略逻辑。

以下是一个完整的策略代码示例额，策略逻辑：每收到一笔Tick行情，以最新价买入100股。

```
from gmsdk.api import StrategyBase
```

```
class MyStrategy(StrategyBase):
    def __init__(self, *args, **kwargs):
        super(MyStrategy, self).__init__(*args, **kwargs)

    def on_tick(self, tick):
        self.open_long(tick.exchange, tick.sec_id, tick.last_price, 100)
        print("OpenLong: exchange %s, sec_id %s, price %s" %
                (tick.exchange, tick.sec_id, tick.last_price))

if __name__ == '__main__':
    ret = MyStrategy(
        username='username',
        password='password',
        strategy_id='strategy_2',
        subscribe_symbols='SHSE.600000.tick',
        mode=2
    ).run()
    print(('exit code: ', ret))
```

4.编译策略并运行。策略运行起来后，控制台打印策略的每一笔下单记录，在掘金终端的模拟交易中可查看该策略的运行详情。

Python API范例

策略接口范例

策略构建

如何构建策略

构建策略的三种方式：

1.掘金终端构建自己的策略，然后在构建的策略类中重写基类方法

2.修改SDK包中的例子，构建自己的策略类

3.参考API定义文档，自定义策略类

策略运行

策略类应继承自StrategyBase基类，并以方法重写的方式满足策略开发需求。然后调用子类实例的`run()`方法，运行策略。

```
class Strategy(StrategyBase):
    ...

strategy.run()
```

如何初始化策略

策略在运行前，必须先初始策略对象。需要指定挖金子账户、密码、策略ID、订阅代码和运行模式。

示例：策略初始化并订阅上交所浦发银行的实时Tick数据和1min的Bar数据

方式一：以参数方式初始化策略

```
ret = Strategy(
        username='demo@myquant.cn',
        password='123456',
        strategy_id='strategy_2',
        subscribe_symbols='SHSE.600000.bar.60',
        mode=2
    )
```

策略支持四种运行模式及对应参数值：

- 1.不接收行情流：1
- 2.接收实时行情：2
- 3.模拟行情模式：3
- 4.回测模式：4

方式二：以配置文件方式初始化策略

策略初始化配置可以保存在*.ini为文件中，然后以文件路径为参数初始化策略。

```
ret = Strategy(config_file='strategy.ini')
```

strategy.ini文件配置示例：

```
[strategy]

;掘金用户名

username=-

;掘金密码

password=-

;策略ID

strategy_id=3c7e70be-7e02-11e5-b293-5ec5d48ea63a

;订阅证券代码或合约代码列表

subscribe_symbols=SHSE.600000.tick,SHSE.600000.bar.60

;行情模式，2-实时行情模式，3-模拟行情模式，4-回放行情模式

mode=2
```

```
;交易服务地址，使用掘金终端交易时，地址为localhost:8001，如果此项配置为空，则订单发往掘金云服务器

td_addr=localhost:8001
```

如何运行策略

完善策略逻辑后，先初始化策略，然后运行策略。策略运行后，策略开始接收数据并执行策略逻辑

```
strategy.run()
```

如何停止策略

1.直接关闭程序来停止策略 2.调用api方式停止策略

```
strategy.stop()
```

策略运行模式

策略支持三种运行模式：

1.实时行情模式：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。适用的场景是策略仿真交易和实盘交易阶段。

2.模拟行情模式：模拟行情是近期的历史行情，行情服务器将7*24小时不间断循环推送，推送频率近似实时行情。适用的场景是策略开发阶段，主要就是能方便随时随地都能有数据，能开发/调试策略，不受交易时段的限制。

3.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。

如何回测策略

策略使用回测模式初始化，并配置回测相关参数，即进入回测。

示例：回测上交所浦发银行的daily数据, 回测时间为 `2015-04-15 09:00:00` 到 `2015-05-15 15:00:00`, 策略初始资金为1,000,000, 委托成交为全部成交, 手续费率为零, 滑点比率为零, 数据使用前复权.

方式一：在策略初始化后调用backtest_config()方法

```
strategy = Strategy(
        username='demo@myquant.cn',
        password='123456',
        strategy_id='strategy_2',
        subscribe_symbols='SHSE.600000.bar.daily',
        mode=4,
        td_addr='localhost:8001')
strategy.backtest_config(
```

```
        start_time='2015-04-15 9:00:00',
        end_time='2015-05-15 15:00:00',
        initial_cash=1000000,
        transaction_ratio=1,
        commission_ratio=0,
        slippage_ratio=0,
        price_type=1,
        bench_symbol='SHSE.000300')
```

方式二：在配置文件中指定参数

```
strategy = Strategy(config_file='strategy.ini')
```

在strategy.ini配置文件中添加backtest节点

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;策略回测参数配置

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

[backtest]
;历史数据回放开始时间
start_time=2015-04-15 09:00:00

;历史数据回放结束时间
end_time=2015-05-15 15:00:00

;策略初始资金
initial_cash=1000000

;委托量成交比率，默认=1（每个委托100%成交）
transaction_ratio=1

;手续费率，默认=0（不计算手续费）
commission_ratio=0

;滑点比率，默认=0（无滑点）
slippage_ratio=0

;行情复权模式,0=不复权,1=前复权
price_type=1

;基准
bench_symbol=SHSE.000300
```

如何在非交易时间段调试策略

首先需关联模拟交易通道。

示例：订阅上交所浦发银行Tick行情和1min的Bar的模拟行情

方式一，通过参数的方式设置：

```
ret = strategy("your user name",
                "your password",
                "strategy id",
                "SHSE.600000.tick,SHSE.600000.bar.60",
                3,
                "localhost:8001")
```

方式二，通过配置方式初始化策略，策略的配置文件subscribe_symbols和mode节点设置如下：

```
;订阅证券代码或合约代码列表
subscribe_symbols=SHSE.600000.tick,SHSE.600000.bar.60

;行情模式，2-实时行情模式，3-模拟行情模式，4-回放行情模式
mode=3
```

如何设置策略仿真交易

首先需关联仿真交易通道。

示例：订阅上交所浦发银行Tick行情和1min的Bar的实时行情

方式一，通过参数的方式设置：

```
ret = strategy("your user name",
                "your password",
                "strategy id",
                "SHSE.600000.tick,SHSE.600000.bar.60",
                2,
                "localhost:8001")
```

方式二，通过配置方式初始化策略，策略的配置文件strategy节点设置如下：

```
;订阅证券代码或合约代码列表
subscribe_symbols=SHSE.600000.tick,SHSE.600000.bar.60

;行情模式，2-实时行情模式，3-模拟行情模式，4-回放行情模式
mode=2
```

如何设置策略实盘交易

1.关联实盘交易通道。

2.参考如何设置策略仿真交易，接收实时行情即可开始实盘交易。

策略事件

策略提供哪些事件

策略的基类提供策略3类事件：

1.登录事件：策略初始化时触发

2.行情数据事件：接收实时行情数据时触发，主要有Tick行情事件和Bar行情事件

3.交易相关事件：交易时触发，主要有下单、撤单、订单回报事件

用户重写自己关注事件的回调方法完善策略逻辑。

策略登录

如何处理策略登录事件

策略在初始化时将触发登录事件，可以在自己的策略类中重写策略基类的on_login方法，以便策略初始化时进行自定义操作

```
class Strategy(StrategyBase):
    def on_login(self):
        ...
```

数据接口范例

行情类型

掘金SDK提供回测行情、模拟行情、实时行情及历史行情数据

1.回测行情、模拟行情和实时行情在订阅了symbol后，在策略的on_tick方法和on_bar方法中接收行情数据

2.各频率的历史行情数据提供对应的API提取。

数据定义

数据类型分为tick数据和bar数据

1.Tick行情是指按交易所实际发送的行情数据

2.Bar数据是指各种频率的行情数据，可订阅 1分、15分、60分的实时Bar数据。

1. 日频数据提供DailyBar数据类型，日频仅在回测和历史数据提取时可用，不能实时订阅。

行情订阅方法

如何初始化行情接口

方式一，策略对象也提供行情接口，请参考如何初始化策略

方式二，获取行情对象实例，然后调用行情接口获取数据，该方式下仅获取数据

```
from gmsdk import md
ret = md.init("your user name", "your password")
```

如何订阅指定证券代码的tick行情

订阅上交所浦发银行和深交所平安银行的Tick行情数据，订阅的证券代码格式定义详细介绍请点击这里。

方式一，通过策略类对象订阅实时行情

1.策略初始化时订阅

```
ret = Strategy("your user name",
                "your password",
                "strategy id",
                "SHSE.600000.tick,SZSE.000001.tick",
                2,
                "localhost:8001")
```

2.策略程序运行中订阅

```
ret = strategy.subscribe("SHSE.600000.tick,SHSE.600000.bar.60")
```

方式二，通过行情类对象订阅实时行情，该方式仅提供行情数据

1.行情类对象初始化时订阅

```
from gmsdk import md
ret = md.init("your user name",
              "your password",
              2,
              "SHSE.600000.tick,SZSE.000001.tick")
```

2.程序运行过程中订阅

```
ret = md.subscribe("SHSE.600000.tick,SZSE.000001.tick")
```

如何订阅指定证券代码的Bar行情

示例：订阅上交所浦发银行的1min Bar行情和深交所平安银行的30s Bar行情

方式一，通过策略类对象订阅实时行情

1.策略初始化时订阅

```
ret = strategy("your user name",
               "your password",
               "strategy id",
               "SHSE.600000.bar.60,SZSE.000001.bar.30",
               3,
               "localhost:8001")
```

2.策略运行过程中订阅

```
ret = strategy.subscribe("SHSE.600000.bar.60,SZSE.000001.bar.30")
```

方式二，通过行情类对象订阅实时行情，该方式仅提供行情数据

1.行情类对象初始化时订阅

```
ret = md.init("your user name",
              "your password",
              2,
              "SHSE.600000.bar.60,SZSE.000001.bar.30")
```

2.程序运行过程中订阅

```
ret = md.subscribe("SHSE.600000.bar.60,SZSE.000001.bar.30")
```

如何订阅指定证券代码的日频行情

日频行情(DailyBar)仅在策略回测时订阅使用，其他策略运行模式下订阅将接收不到日频行情

示例：订阅上交所浦发银行的日频行情和深交所平安银行的日频行情

通过策略类对象订阅实时行情

1.策略初始化时订阅

```
ret = strategy("your user name",
               "your password",
               "strategy id",
               "SHSE.600000.bar.daily,SZSE.000001.bar.daily",
               3,
               "localhost:8001")
```

2.策略运行中订阅

```
ret = strategy.subscribe("SHSE.600000.daily,SZSE.000001.daily")
```

如何退订指定证券代码的行情

示例：退订上交所浦发银行的tick行情和深交所平安银行的1min Bar行情

方式一，通过策略类对象退订：

```
ret = strategy.unsubscribe("SHSE.600000.tick,SZSE.000001.bar.60")
```

方式二，通过行情类对象退订：

```
ret = md.unsubscribe("SHSE.600000.tick,SZSE.000001.bar.60")
```

历史数据提取

如何提取指定时间的Tick数据

示例：提取上交所浦发银行和深交所平安银行 `2015-10-29 9:30:00` 到 `2015-10-29 15:00:00` 的Tick数据

方式一，通过策略类对象接口提取

```
ticks = strategy.get_ticks("SHSE.600000,SZSE.000001",
                           "2015-10-29 9:30:00",
                           "2015-10-29 15:00:00")
```

方式二，通过行情类对象接口提取

```
ticks = md.get_ticks("SHSE.600000,SZSE.000001",
                     "2015-10-29 9:30:00",
                     "2015-10-29 15:00:00")
```

如何提取指定时间的Bar数据

示例：提取上交所浦发银行和深交所平安银行2015-10-29 10:00:00到2015-10-29 15:00:00的1min Bar数据,提取30s、5min频率的bar数据修改参数bar_type为30、300即可。

方式一，通过策略类对象接口提取

```
bars = strategy.get_bars("SHSE.600000,SZSE.000001",
                         60,
                         "2015-10-29 10:00:00",
                         "2015-10-29 15:00:00")
```

方式二，通过行情类对象接口提取

```
bars = md.get_bars("SHSE.600000,SZSE.000001",
                   60,
                  "2015-10-29 10:00:00",
                  "2015-10-29 15:00:00")
```

如何提取指定日期的日频数据

示例：提取上交所浦发银行和深交所平安银行2014-10-29到2015-10-29 的日频数据(DailyBar)。

方式一，通过策略类对象接口提取

```
daily_bars = strategy.get_dailybars("SHSE.600000,SZSE.000001", "2014-10-29", "2015-10-29")
```

方式二，通过行情类对象接口提取

```
daily_bars = md.get_dailybars("SHSE.600000,SZSE.000001","2014-10-29", "2015-10-29")
```

如何提取最近N笔tick数据

示例：提取上交所浦发银行和深交所平安银行最近100笔Tick数据

方式一，通过策略类对象接口提取

```
ticks = strategy.get_last_n_ticks("SHSE.600000,SZSE.000001", 100)
```

方式二，通过行情类对象接口提取

```
ticks = md.get_last_n_ticks("SHSE.600000,SZSE.000001", 100)
```

如何提取最近N个Bar数据

示例：提取上交所浦发银行和深交所平安银行最近20笔1min频率的Bar数据，提取30s、5min频率的bar数据修改参数bar_type为30、300即可。

方式一，通过策略类对象接口提取

```
bars = strategy.get_last_n_bars("SHSE.600000,SZSE.000001", 60, 20)
```

方式二，通过行情类对象接口提取

```
bars = md.get_last_n_bars("SHSE.600000,SZSE.000001", 60, 20)
```

如何提取最近N个交易日的日频数据

示例：提取上交所浦发银行和深交所平安银行最近20笔日频数据(DailyBar)

方式一，通过策略类对象接口提取

```
daily_bars = strategy.get_last_n_dailybars("SHSE.600000,SZSE.000001", 20)
```

方式二，通过行情类对象接口提取

```
daily_bars = md.get_last_n_dailybars("SHSE.600000,SZSE.000001", 20)
```

如何提取最近1笔tick数据

示例：提取上交所浦发银行和深交所平安银行最近1笔Tick数据

方式一，通过策略类对象接口提取

```
ticks = strategy.get_last_ticks("SHSE.600000,SZSE.000001")
```

方式二，通过行情类对象接口提取

```
ticks = md.get_last_ticks("SHSE.600000,SZSE.000001")
```

如何提取最近1个Bar数据

示例：提取上交所浦发银行和深交所平安银行最近1个1min频率的Bar数据，提取30s、5min频率的bar数据修改参数bar_type为30、300即可。

方式一，通过策略类对象接口提取

```
bars = strategy.get_last_bars("SHSE.600000,SZSE.000001", 60)
```

方式二，通过行情类对象接口提取

```
bars = md.get_last_bars("SHSE.600000,SZSE.000001", 60)
```

如何提取最近1笔日频数据

示例：提取上交所浦发银行和深交所平安银行最近1笔日频数据(DailyBar)

方式一，通过策略类对象接口提取

```
daily_bars = strategy.get_last_dailybars("SHSE.600000,SZSE.000001")
```

方式二，通过行情类对象接口提取

```
daily_bars = md.get_last_dailybars("SHSE.600000,SZSE.000001")
```

行情事件方法

如何处理tick事件和bar事件

tick、bar事件在接收实时行情时触发,在tick和bar事件的回调方法on_tick、on_bar中可接收订阅的Tick行情和Bar行情

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_tick、on_bar方法。

```
class Strategy(StrategyBase):
    def on_tick(self, tick):
        ...

    def on_bar(self, bar):
        ...
```

方式二，在纯行情模式中处理

1.编写tick事件和bar事件的回调方法

```
def on_tick(tick):
    ...

def on_bar(bar):
    ...
```

2.订阅tick事件和bar事件

```
md.ev_tick += on_tick
md.ev_bar += on_bar
```

如何处理行情状态事件

在自己的策略类中重写策略基类的on_md_event方法，行情状态事件在开盘、收盘、回放行情结束时触发

```
class Strategy(StrategyBase):
    def on_md_event(self, evt):
        ...
```

交易接口范例

交易相关数据类型

交易涉及到持仓Position、委托Order、成交回报ExecRpt、资金Cash、绩效指标Indicator等数据类型。

委托

如何开多仓

示例：市价买入1000股上交所浦发银行股票，市价开1手IF1512的多单

注意：股票、基金等现货品种只有买入和卖出，对应下单类型是开多仓open_long和平多仓close_long；

方式一，调用开多仓的异步接口open_long

1.在策略类的方法中调用，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.open_long("SHSE", "600000", 0, 1000)
        self.open_long("CFFEX", "IF1512", 0,  1)

Strategy(...).run()
```

2.通过交易服务类对象调用,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event():
    td.open_long("SHSE", "600000", 0, 1000)
    td.open_long("CFFEX", "IF1512", 0,  1)

td.ev_some_event += on_some_event
td.init(...)
td.run()
```

方式二，调用开多仓的同步接口open_long_sync，无需策略执行Run方法

1.通过策略类对象调用

```
strategy = Strategy(...)
strategy.open_long_sync("SHSE", "600000", 0, 1000)
strategy.open_long_sync("CFFEX", "IF1512", 0,  1)
```

2.通过交易服务类对象调用

```
from gmsdk import td

td.init(...)
td.open_long_sync("SHSE", "600000", 0, 1000)
td.open_long_sync("CFFEX", "IF1512", 0,  1)
```

如何平多仓

示例：市价卖出1000股上交所浦发银行股票，市价平1手IF1512的多单

注意：股票、基金等现货品种只有买入和卖出，对应下单类型是开多仓open_long和平多仓close_long；

方式一，调用平多仓的异步接口close_long

1.在策略类的方法中调用接口，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.close_long("SHSE", "600000", 0, 1000)
        self.close_long("CFFEX", "IF1512", 0,  1)

Strategy(...).run()
```

2.通过交易服务类对象调用接口,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    td.close_long("SHSE", "600000", 0, 1000)
    td.close_long("CFFEX", "IF1512", 0,  1)

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

方式二，调用平多仓的同步接口close_long_sync，无需策略执行Run方法

1.通过策略类对象调用接口

```
strategy = Strategy(...)
strategy.close_long_sync("SHSE", "600000", 0, 1000)
strategy.close_long_sync("CFFEX", "IF1512", 0,  1)
```

2.通过交易服务类对象调用接口

```
from gmsdk import td

td.init(...)
td.close_long_sync("SHSE", "600000", 0, 1000)
td.close_long_sync("CFFEX", "IF1512", 0,  1)
```

如何开空仓

示例：市价开1手IF1512的空单

方式一，调用开空仓的异步接口open_short

1.在策略类的方法中调用接口，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.open_short("CFFEX", "IF1512", 0,  1)

Strategy(...).run()
```

2.通过交易服务类对象调用接口,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    td.open_short("CFFEX", "IF1512", 0,  1)

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

方式二，调用开空仓的同步接口open_short_sync，无需策略执行Run方法

1.通过策略类对象调用接口

```
strategy = Strategy(...)
order_ret = strategy.open_short_sync("CFFEX", "IF1512", 0,  1)
```

2.通过交易服务类对象调用接口

```
from gmsdk import td
td.init(...)
td.open_short_sync("CFFEX", "IF1512", 0,  1)
```

如何平空仓

示例：市价平1手IF1512的空单

方式一，调用平空仓的异步接口close_short

1.在策略类的方法中调用接口，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.close_short("CFFEX", "IF1512", 0,  1)

Strategy(...).run()
```

2.通过交易服务类对象调用接口,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    td.close_short("CFFEX", "IF1512", 0,  1)

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

方式二，调用平空仓的同步接口close_short_sync，无需策略执行Run方法

1.通过策略类对象调用接口

```
strategy = Strategy(...)
order_ret = strategy.close_short_sync("CFFEX", "IF1512", 0,  1)
```

2.通过交易服务类对象调用接口

```
from gmsdk import td
td.init(...)
td.close_short_sync("CFFEX", "IF1512", 0,  1)
```

如何自定义下单

示例：市价开1手IF1512的多单

定义委托单对象

```
order = Order()
order.exchange = "CFFEX"
order.sec_id = "IF1512"
order.side = 1
order.position_effect = 1
order.price = 0
order.volume = 1
```

说明：

order.side: 设置买卖方向OrderSide

order.position_effect：设置开平类型PositionEffect,上期所可设置平今平昨

order.price：价格为 0 则表示市价单，否则为限价单

方式一，调用原生下单的异步接口place_order

1.在策略类的方法中调用接口，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.place_order(order)

Strategy(...).run()
```

2.通过交易服务类对象调用接口,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    td.place_order(order)
td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

方式二，调用原生下单的同步接口place_order_sync，无需策略执行Run方法

1.通过策略类对象调用接口

```
strategy = Strategy(...)
strategy.place_order_sync(order)
```

2.通过交易服务类对象调用接口

```
from gmsdk import td
td.init(...)
td.place_order_sync(order);
```

如何撤回未成交订单

方式一，调用异步撤单接口cancel_order

1.在策略类的方法中调用接口，必须在策略执行了Run方法后调用有效

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        self.cancel_order(order_id)

Strategy(...).run()
```

2.通过交易服务类对象调用接口,必须执行对象的Run方法后调用有效

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    td.cancel_order(order_id)

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

方式二，调用同步撤单接口cancel_order_sync，无需策略执行Run方法

1.通过策略类对象调用接口

```
strategy = Strategy(...)
strategy.cancel_order(order_id)
```

2.通过交易服务类对象调用接口

```
from gmsdk import td
td.init(...)
td.cancel_order(order_id)
```

如何查询指定委托当前详情

1.在策略类的方法中调用接口

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        order = self.get_order(order_id)

Strategy(...).run()
```

2.通过交易服务类对象调用接口

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    order = td.get_order(order_id)

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

回报

订单的回报以交易事件的方式给出，当前提供的交易事件有：

1. 委托执行回报事件
2. 订单拒绝事件
3. 订单被柜台接受事件
4. 订单状态变更事件
5. 订单全部成交事件
6. 订单部分成交事件
7. 订单停止成交事件
8. 订单撤单成功事件
9. 订单撤单拒绝事件

具体事件处理请参考交易事件处理章节

资金

如何查询资金

调用get_cash方法查询当前策略的资金信息。

1. 在策略类的方法中直接调用接口

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        cash = self.get_cash()

Strategy(...).run()
```

1. 通过交易服务类对象调用接口

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    cash = td.get_cash()

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

持仓

如何获取指定证券代码的持仓

调用get_position方法查询单一交易品种的持仓情况。

示例：获取上交所浦发银行持仓情况

1. 在策略类的方法中直接调用持仓接口

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        postion = self.get_position("SHSE", "600000", 1)

Strategy(...).run()
```

1. 通过交易服务类对象调用持仓接口

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
```

```
    postion = td.get_position("SHSE", "600000", 1)
td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

如何获取当前策略的全部持仓信息

调用get_positions方法获取当前策略的全部持仓信息。

1. 在策略类的方法中直接调用接口

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        pos_list = self.get_positions()

Strategy(...).run()
```

1. 通过交易服务类对象调用接口

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    pos_list = td.get_positions()

td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

绩效

如何获取当前策略的绩效指标

调用get_indicator方法获取当前策略的绩效指标。

1. 在策略类的方法中直接调用接口

```
class Strategy(StrategyBase):
    def on_some_event(self): ## eg. def on_tick(self, tick):
        indicator = self.get_indicator()

Strategy(...).run()
```

1. 通过交易服务类对象调用接口

```
from gmsdk import td

def on_some_event(): ##eg. def on_tick(tick):
    indicator = td.get_indicator()
td.ev_some_event += on_some_event ##eg. td.ev_tick += on_tick
td.init(...)
td.run()
```

交易事件

当我们在程序下一笔委托单到交易服务时，将产生一系列交易事件

如何处理订单执行回报事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_execrpt方法，on_execrpt方法在委托执行时触发，订单的任何执行回报都会触发本事件

```
class Strategy(StrategyBase):
    def on_execrpt(self, rpt):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写委托执行的回调方法

```
def on_execrpt(rpt):
    ...
```

1. 订阅委托执行事件

```
from gmsdk import td
td.ev_execrpt += on_execrpt
td.init(...).run()
```

如何处理订单拒绝事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_rejected方法，on_order_rejected方法在订单拒绝时触发

```
class Strategy(StrategyBase):
    def on_order_rejected(self, order):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写委订单拒绝的回调方法

```
def on_order_rejected(order):
    ...
```

1. 订阅订单拒绝事件

```
from gmsdk import td
td.ev_order_rejected += on_order_rejected
td.init(...)
td.run()
```

如何处理订单被交易所接收事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_new方法，当订单已被交易所接受时触发

```
class Strategy(StrategyBase):
    def on_order_new(self, order):
        ...

ret = Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单接收的回调方法

```
def on_order_new(order):
   ...
```

1. 订阅订单接收事件

```
from gmsdk import td
td.ev_order_new += on_order_new
td.init(...)
td.run()
```

如何处理订单状态更新事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_status方法，当订单状态更新时触发

```
class Strategy(StrategyBase):
    def on_order_status(self, order):
        ...

ret = Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单接收的回调方法

```
def on_order_status(order):
    ...
```

1. 订阅订单状态更新事件

```
from gmsdk import td
td.ev_order_status += on_order_status
td.init(...)
td.run()
```

如何处理订单全部成交事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_filled方法,当一笔订单完全成交时触发

```
class Strategy(StrategyBase):
    def on_order_filled(self, order):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单全成的回调方法

```
def on_order_filled(order):
    ...
```

1. 订阅订单全成事件

```
from gmsdk import td
td.ev_order_filled += on_order_filled
td.init(...)
td.run()
```

如何处理委托单部分成交的事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_partially_filled方法,当一笔订单有部分成交时触发

```
class Strategy(StrategyBase):
    def on_order_partially_filled(self, order):
        ...
```

```
Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单部分成交的回调方法

```
def on_order_partially_filled(order):
    ...
```

1. 订阅订单部分成交事件

```
from gmsdk import td
td.ev_order_partially_filled += on_order_partially_filled
td.init(...)
td.run()
```

如何处理停止订单执行事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_stop_executed方法,当订单停止执行时触发，例如限价单收市时还未成交

```
class Strategy(StrategyBase):
    def on_order_stop_executed(self, order):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单停止执行的回调方法

```
def on_order_stop_executed(order):
    ...
```

1. 订阅订单停止执行事件

```
from gmsdk import td
td.ev_order_stop_executed += on_order_stop_executed
td.init(...)
td.run()
```

如何处理撤单拒绝事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_cancel_rejected方法,当撤单拒绝时触发

```
class Strategy(StrategyBase):
    def on_order_cancel_rejected(self, order):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写订单停止执行的回调方法

```
def on_order_cancel_rejected(rpt):
    ...
```

1. 订阅订单撤单拒绝事件

```
from gmsdk import td
td.ev_order_cancel_rejected += on_order_cancel_rejected
td.init(...)
td.run()
```

如何处理委托撤单成功事件

方式一，在策略类中处理

在自己的策略类中重写策略基类的on_order_cancelled方法,当委托单的状态发生变化时触发

```
class Strategy(StrategyBase):
    def on_order_cancelled(self, order):
        ...

Strategy(...).run()
```

方式二，通过交易服务类对象处理

1. 编写委托单状态变化的回调方法

```
def on_order_cancelled(order):
    ...
```

1. 订阅撤单成功事件

```
from gmsdk import td
td.ev_order_cancelled += on_order_cancelled
td.init(...)
td.run()
```

错误与版本等接口范例

如何获取API版本信息

调用get_version方法。

```
import gmsdk
gmsdk.get_version()
```

如何获取API返回值及错误事件参数值的文本信息

API返回值及错误事件的参数值都有统一的定义，调用get_strerror方法获取返回值对应的文本信息。

如何获取策略运行中的错误信息

在自己的策略类中重写策略基类的on_error方法,当策略执行过程有任何错误都会回调该方法，通过参数获取错误信息

```
class Strategy(StrategyBase):
    def on_error(self, error_code, error_msg):
        ...

Strategy(...).run()
```

Python API接口

枚举常量定义

OrderStatus

订单状态。

```
OrderStatus_New = 1,                        #已报
OrderStatus_PartiallyFilled = 2,            #部成
OrderStatus_Filled = 3,                     #已成
OrderStatus_DoneForDay = 4,                 #
OrderStatus_Canceled = 5,                   #已撤
OrderStatus_PendingCancel = 6,              #待撤
OrderStatus_Stopped = 7,                    #停止
OrderStatus_Rejected = 8,                   #已拒绝
OrderStatus_Suspended = 9,                  #挂起
OrderStatus_PendingNew = 10,                #待报
OrderStatus_Calculated = 11,                #计算
OrderStatus_Expired = 12,                   #已过期
OrderStatus_AcceptedForBidding = 13,        #接受竞价
OrderStatus_PendingReplace = 14             #待修改
```

OrderRejectReason

订单拒绝原因。

```
OrderRejectReason_UnknownReason = 1,                #未知原因
OrderRejectReason_RiskRuleCheckFailed = 2,          #不符合风控规则
OrderRejectReason_NoEnoughCash = 3,                 #资金不足
OrderRejectReason_NoEnoughPosition = 4,             #仓位不足
OrderRejectReason_IllegalStrategyID = 5,            #非法策略ID
OrderRejectReason_IllegalSymbol = 6,                #非法交易标的
OrderRejectReason_IllegalVolume = 7,                #非法委托量
OrderRejectReason_IllegalPrice = 8,                 #非法委托价
OrderRejectReason_NoMatchedTradingChannel = 9,      #没有匹配的交易通道
OrderRejectReason_AccountForbidTrading = 10,        #交易账号被禁止交易
OrderRejectReason_TradingChannelNotConnected = 11,  #交易通道未连接
OrderRejectReason_StrategyForbidTrading = 12,       #策略不允许交易
OrderRejectReason_NotInTradingSession = 13          #非交易时段
CancelOrderRejectReason_OrderFinalized = 101        #订单已是最终状态
CancelOrderRejectReason_UnknownOrder = 102          #未知订单
CancelOrderRejectReason_BrokerOption = 103          #柜台拒绝
CancelOrderRejectReason_AlreadyInPendingCancel = 104        #重复撤单
```

OrderSide

订单方向。

```
OrderSide_Bid = 1  ## 多方向
OrderSide_Ask = 2  ## 空方向
```

OrderType

订单类型。

```
OrderType_LMT = 0,       ## 限价委托(limit)
OrderType_BOC = 1,       ## 对方最优价格(best of counterparty)
OrderType_BOP = 2,       ## 己方最优价格(best of party)
OrderType_B5TC = 3,      ## 最优五档剩余撤销(best 5 then cancel)
OrderType_B5TL = 4,      ## 最优五档剩余转限价(best 5 then limit)
OrderType_IOC = 5,       ## 即时成交剩余撤销(immediately or cancel)
OrderType_FOK = 6,       ## 即时全额成交或撤销(fill or kill)
OrderType_AON = 7,       ## 全额成交或不成交(all or none)
OrderType_MTL = 8,       ## 市价剩余转限价(market then limit)
OrderType_EXE = 9        ## 期权行权(option execute)
```

ExecType

订单执行回报类型。

```
ExecType_New = 1                ## 交易所已接受订单
ExecType_DoneForDay = 4
ExecType_Canceled = 5           ## 已撤
ExecType_PendingCancel = 6      ## 待撤
ExecType_Stopped = 7            ## 已停
ExecType_Rejected = 8           ## 已拒绝
ExecType_Suspended = 9          ## 暂停
ExecType_PendingNew = 10        ## 待接受
ExecType_Calculated = 11        ## 已折算
ExecType_Expired = 12           ## 过期
ExecType_Restated = 13          ## 重置
```

```
ExecType_PendingReplace = 14    ## 待修改
ExecType_Trade = 15             ## 交易
ExecType_TradeCorrect = 16      ## 交易更正
ExecType_TradeCancel = 17       ## 交易取消
ExecType_OrderStatus = 18       ## 更新订单状态
ExecType_CancelRejected = 19    ## 撤单被拒绝
```

PositionEffect

开平仓类型。

```
PositionEffect_Open = 1             ## 开仓
PositionEffect_Close = 2            ## 平仓
PositionEffect_CloseToday = 3       ## 平今仓
PositionEffect_CloseYesterday = 4   ## 平昨仓
```

交易数据类型

交易数据类型主要包括委托，执行回报，资金，持仓，绩效等数据类型。

Order

委托订单。

```
class Order(object):

    def __init__(self):
        self.strategy_id = ''                  ## 策略ID
        self.account_id = ''                   ## 交易账号

        self.cl_ord_id = ''                    ## 客户端订单ID
        self.order_id = ''                     ## 柜台订单ID
        self.ex_ord_id = ''                    ## 交易所订单ID

        self.exchange = ''                     ## 交易所代码
        self.sec_id = ''                       ## 证券ID

        self.position_effect = 0               ## 开平标志
        self.side = 0                          ## 买卖方向
        self.order_type = 0                    ## 订单类型
        self.order_src = 0                     ## 订单来源
        self.status = 0                        ## 订单状
        self.ord_rej_reason = 0                ## 订单拒绝原因
        self.ord_rej_reason_detail = ''        ## 订单拒绝原因描述

        self.price = 0.0                       ## 委托价
        self.stop_price = 0.0;                 ## 止损价
        self.volume = 0.0                      ## 委托量
        self.filled_volume = 0.0               ## 已成交量
        self.filled_vwap = 0.0                 ## 已成交均价
        self.filled_amount = 0.0               ## 已成交额

        self.sending_time = 0.0                ## 委托下单时间
        self.transact_time = 0.0               ## 最新一次成交时间
```

ExecRpt

委托执行回报。

```
class ExecRpt(object):

    def __init__(self):
        self.strategy_id = ''                   ## 策略ID

        self.cl_ord_id = ''                     ## 客户端订单ID
        self.order_id = ''                      ## 交易所订单ID
        self.exec_id = ''                       ## 订单执行回报ID

        self.exchange = ''                      ## 交易所代码
        self.sec_id = ''                        ## 证券ID

        self.position_effect = 0                ## 开平标志
        self.side = 0                           ## 买卖方向
        self.ord_rej_reason = 0                 ## 订单拒绝原因
        self.ord_rej_reason_detail = ''         ## 订单拒绝原因描述
        self.exec_type = 0                      ## 订单执行回报类型

        self.price = 0.0                        ## 成交价
        self.volume = 0.0                       ## 成交量
        self.amount = 0.0                       ## 成交额

        self.transact_time = 0.0                ## 交易时间
```

Cash

资金。

```
class Cash(object):

    def __init__(self):
        self.strategy_id = ''           ## 策略ID
        self.account_id = ''            ## 账户id
        self.currency = 0               ## 币种

        self.nav = 0.0                  ## 资金余额
        self.fpnl = 0.0                 ## 浮动收益
        self.pnl = 0.0                  ## 净收益
        self.profit_ratio = 0.0         ## 收益率
        self.frozen = 0.0               ## 持仓冻结金额
        self.order_frozen = 0.0         ## 挂单冻结金额
        self.available = 0.0            ## 可用资金

        self.cum_inout = 0.0            ## 累计出入金
        self.cum_trade = 0.0            ## 累计交易额
        self.cum_pnl = 0.0              ## 累计收益
        self.cum_commission = 0.0       ## 累计手续费

        self.last_trade = 0.0           ## 最新一笔交易额
        self.last_pnl = 0.0             ## 最新一笔交易收益
        self.last_commission = 0.0      ## 最新一笔交易手续费
        self.last_inout = 0.0           ## 最新一次出入金
        self.change_reason = 0          ## 变动原因
```

```
        self.transact_time = 0.0        ## 交易时间
```

Position

持仓。

```
class Position(object):
    def __init__(self):
        self.strategy_id = ''           ## 策略ID
        self.account_id = ''            ## 账户id
        self.exchange = ''              ## 交易所代码
        self.sec_id = ''                ## 证券ID
        self.side = 0                   ## 买卖方向
        self.volume = 0.0               ## 持仓量
        self.volume_today = 0.0         ## 今仓量
        self.amount = 0.0               ## 持仓额
        self.vwap = 0.0                 ## 持仓均价

        self.price = 0.0                ## 当前行情价格
        self.fpnl = 0.0                 ## 持仓浮动盈亏
        self.cost = 0.0                 ## 持仓成本
        self.order_frozen = 0.0         ## 挂单冻结仓位
        self.available = 0.0            ## 可平仓位
        self.available_today = 0.0      ## 可平今仓位(volume_today-order_frozen_today)
        self.avaliable_yesterday = 0.0  ## 可平昨仓位(available - available_today)
        self.order_frozen_today = 0.0   ## 挂单冻结今仓
        self.last_price = 0.0           ## 上一笔成交价
        self.last_volume = 0.0          ## 上一笔成交量
        self.init_time = 0.0            ## 初始建仓时间
        self.transact_time = 0.0        ## 上一仓位变更时间
```

Indicator

绩效。

```
class Indicator(object):

    def __init__(self):
        self.strategy_id = ''                       ## 策略ID
        self.account_id = ''                        ## 账号ID

        self.nav = 0.0                              ## 净值(cum_inout + cum_pnl + fpnl -
cum_commission)
        self.pnl = 0.0                              ## 净收益(nav-cum_inout)
        self.profit_ratio = 0.0                     ## 收益率(pnl/cum_inout)
        self.profit_ratio_bench = 0.0               ## 基准收益率
        self.sharp_ratio = 0.0                      ## 夏普比率
        self.risk_ratio = 0.0                       ## 风险比率
        self.trade_count = 0                        ## 交易次数
        self.win_count = 0                          ## 盈利次数
        self.lose_count = 0                         ## 亏损次数
        self.win_ratio = 0.0                        ## 胜率
        self.max_profit = 0.0                       ## 最大收益
        self.min_profit = 0.0                       ## 最小收益
        self.max_single_trade_profit = 0.0          ## 最大单次交易收益
        self.min_single_trade_profit = 0.0          ## 最小单次交易收益
```

```
        self.daily_max_single_trade_profit = 0.0    ## 今日最大单次交易收益
        self.daily_min_single_trade_profit = 0.0    ## 今日最小单次交易收益
        self.max_position_value = 0.0               ## 最大持仓市值或权益
        self.min_position_value = 0.0               ## 最小持仓市值或权益
        self.max_drawdown = 0.0                     ## 最大回撤
        self.daily_pnl = 0.0                        ## 今日收益
        self.daily_return = 0.0                     ## 今日收益率
        self.annual_return = 0.0                    ## 年化收益率

        self.cum_inout = 0.0                        ## 累计出入金
        self.cum_trade = 0.0                        ## 累计交易额
        self.cum_pnl = 0.0                          ## 累计平仓收益(没扣除手续费)
        self.cum_commission = 0.0                   ## 累计手续费

        self.transact_time = 0.0                    ## 指标计算时间
```

BrokerAccount

柜台账户

```
class BrokerAccount(object):
    def __init__(self):
        self.account_id = ''                        # 柜台账号ID
        self.username = ''                          # 柜台账号
        self.permissible = 0                        # 允许交易
        self.status = 0                             # 账号当前状态
```

StrategyParameter

策略参数

```
class StrategyParameter(object):
    def __init__(self):
        self.name = ''                              # 参数名
        self.value = 0.0                            # 参数值
        self.min = 0.0                              # 可设置的最小值
        self.max = 0.0                              # 可设置的最大值
        self.readonly = false                       # 是否只读
        self.group = ''                             # 组名
        self.intro = ''                             # 参数说明
```

StrategySymbol

策略交易标的

```
class StrategySymbol(object):
    def __init__(self):
        self.symbol = ''                             # 交易代码
        self.exchange = ''                          # 交易所代码
        self.sec_id = ''                            # 证券ID
```

行情数据类型

行情数据类型有Tick，Bar，DailyBar。

Tick

逐笔行情数据。

```
class Tick(object):
    def __init__(self):
        self.exchange = ''          ## 交易所代码
        self.sec_id = ''            ## 证券ID
        self.utc_time = 0.0         ## 行情时间戳
        self.strtime = ''           ## 可视化时间
        self.last_price = 0.0       ## 最新价
        self.open = 0.0             ## 开盘价
        self.high = 0.0             ## 最高价
        self.low = 0.0              ## 最低价

        self.cum_volume = 0.0       ## 成交总量/最新成交量,累计值
        self.cum_amount = 0.0       ## 成交总金额/最新成交额,累计值
        self.cum_position = 0.0     ## 合约持仓量(期),累计值
        self.last_volume = 0        ## 瞬时成交量(中金所提供)
        self.last_amount = 0.0      ## 瞬时成交额

        self.upper_limit = 0.0      ## 涨停价
        self.lower_limit = 0.0      ## 跌停价
        self.settle_price = 0.0     ## 今日结算价
        self.trade_type = 0         ## (保留)交易类型,对应多开,多平等类型 0:'上一个tick没有成交量',
1:'双开', 2: '双平', 3: '多开', 4:'空开', 5: '空平', 6:'多平', 7:'多换', 8:'空换'
        self.pre_close = 0.0        ## 昨收价

        self.bids = []  ## [(price, volume), (price, volume), ...] ## 1-5档买价,量
        self.asks = []  ## [(price, volume), (price, volume), ...] ## 1-5档卖价,量
```

Bar

各种周期的Bar数据。

```
class Bar(object):

    def __init__(self):
        self.exchange = ''        ## 交易所代码
        self.sec_id = ''          ## 证券ID

        self.bar_type = 0         ## bar类型，以秒为单位，比如1分钟bar, bar_type=60
        self.strtime = ''         ## Bar开始时间
        self.utc_time = 0.0       ## Bar开始时间
        self.strendtime = ''      ## Bar结束时间
        self.utc_endtime = 0.0    ## Bar结束时间
        self.open = 0.0           ## 开盘价
        self.high = 0.0           ## 最高价
        self.low = 0.0            ## 最低价
        self.close = 0.0          ## 收盘价
        self.volume = 0.0         ## 成交量
        self.amount = 0.0         ## 成交额
        self.pre_close            ## 前收盘价
        self.position;            ## 持仓量
        self.adj_factor           ## 复权因子
        self.flag                 ## 除权出息标记
```

DailyBar

日频数据，在Bar数据的基础上，还包含结算价，涨跌停价等静态数据。

```
class Dailybar(object):

    def __init__(self):
        self.exchange = ''          ## 交易所代码
        self.sec_id = ''            ## 证券ID

        self.bar_type = 0           ## bar类型
        self.strtime = ''           ## 可视化时间
        self.utc_time = 0.0         ## 行情时间戳

        self.open = 0.0             ## 开盘价
        self.high = 0.0             ## 最高价
        self.low = 0.0              ## 最低价
        self.close = 0.0            ## 收盘价
        self.volume = 0.0           ## 成交量
        self.amount = 0.0           ## 成交额

        self.position = 0.0         ## 仓位
        self.settle_price = 0.0     ## 结算价
        self.upper_limit = 0.0      ## 涨停价
        self.lower_limit = 0.0      ## 跌停价
        self.pre_close              ## 前收盘价
        self.adj_factor             ## 复权因子
        self.flag                   ## 除权出息，停牌等标记
```

Instrument

交易代码数据类型

```
class Instrument(object):
    def __init__(self):
        self.symbol = ''                ## 交易代码
        self.sec_type = 0               ## 代码类型
        self.sec_name = ''              ## 代码名称
        self.multiplier = 0.0           ## 合约乘数
        self.margin_ratio = 0.0         ## 保证金比率
        self.price_tick = 0.0           ## 价格最小变动单位
        self.upper_limit = 0.0          ## 当天涨停板
        self.lower_limit = 0.0          ## 当天跌停板
        self.is_active = 0              ## 当天是否交易
        self.update_time = ''           ## 更新时间
```

Constituent

成份股数据类型

```
class Constituent(object):
    def __init__(self):
        self.symbol = ''                ## 交易代码
        self.weight = 0.0               ## 代码权重
```

FinancialIndex

财务指标

```
class FinancialIndex(object):
    def __init__(self):
        self.symbol = ''                        #股票代码
        self.pub_date = ''                      #公告日期
        self.eps = 0.0                          #每股收益
        self.bvps = 0.0                         #每股净资产
        self.cfps = 0.0                         #每股现金流
        self.afps = 0.0                         #每股公积金
        self.total_asset = 0.0                  #总资产
        self.current_asset = 0.0                #流动资产
        self.fixed_asset = 0.0                  #固定资产
        self.liability = 0.0                    #负债合计
        self.current_liability = 0.0            #流动负债
        self.longterm_liability = 0.0           #长期负债
        self.equity = 0.0                       #所有者权益
        self.income = 0.0                       #主营业务收入
        self.operating_profit = 0.0             #主营业务利润
        self.net_profit = 0.0                   #净利润
```

ShareIndex

股本指标

```
class ShareIndex(Object):
    def __init__(self):
        self.symbol = ''                        #股票代码
        self.pub_date = ''                      #公告日期
        self.total_share = 0.0                  #总股本
        self.flow_a_share = 0.0                 #流通A股
        self.nonflow_a_share = 0.0              #限售流通A股
```

MarketIndex

市场指标

```
class MarketIndex(Object):
    def __init__(self):
        self.symbol = ''                #股票代码
        self.pub_date = ''              #公告日期
        self.pe_ratio = 0.0             #市盈率
        self.pb_ratio = 0.0             #市净率
        self.ps_ratio = 0.0             #市销率
        self.market_value = 0.0         #市值
        self.market_value_flow = 0.0    #流通市值
```

TradeDate

交易日类型

```
class TradeDate(object):
    def __init__(self):
        self.utc_time = 0.0                 ## UTC时间戳[带毫秒]
        self.strtime = ''                   ## 交易日
```

StockAdjustFactor

复权因子

```
class StockAdjustFactor(object):
    def __init__(self):
        self.symbol         ##股票代码
        self.trade_date     ##交易日
        self.adj_factor     ##复权因子
```

StockDivident

分红送股事件明细

```
class StockDivident(object):
    def __init__(self):
        self.symbol                     ##股票代码
        self.div_date                   ##除权除息日
        self.cash_div                   ##每股派现
        self.share_div_ratio            ##每股送股比例
        self.share_trans_ratio          ##每股转增股比例
        self.allotment_ratio            ##每股配股比例
        self.allotment_price            ##配股价
```

VirtualContract

虚拟合约明细

```
class VirtualContract(object):
    def __init__(self):
        self.vsymbol        ##主力合约或连接合约代码
        self.symbol         ##真实symbol
        self.trade_date     ##交易日
```

策略初始化方法

init

初始化策略，设置服务地址，账号密码，策略ID，以及需要订阅的symbol列表。策略自动连接服务，并登陆，订阅对应的行情，准备好运行。

函数原型：

```
__init__(self,
        username,
        password,
        strategy_id,
        subscribe_symbols='',
        mode=2,
        td_addr='',
        config_file=None,
        config_file_encoding='utf-8')
```

参数：

参数名	类型	说明
username	string	掘金终端账号
password	string	掘金终端密码
strategy_id	string	策略ID
subscribe_symbols	string	行情订阅的代码列表
mode	int	枚举类型，行情模式
td_addr	string	交易服务器uri, 如设置为localhost:8001，则终端用户指向本地客户端，如果设置为空, 则使用掘金云端服务

返回值：

返回值编码

示例：

初始化时订阅上交所浦发银行的tick和1min bar实时行情

```
ret = Strategy("your user name",
               "your password",
               "strategy id",
               "SHSE.600000.tick,SHSE.600000.bar.60",
               2,
               "localhost:8001")
```

注意项：

symbol_list 订阅代码表,参数格式如下:

订阅串有三节或四节组成,用'.'分隔，格式：

对应交易所exchange.代码code.数据类型data_type.周期类型 `bar_type`

只有订阅bar数据时, 才用到第四节, 周期类型才有作用 交易所exchange统一四个字节:

CFFEX-中金所 SHFE-上期所 DCE-大商所 CZCE-郑商所 SHSE-上交所 SZSE-深交所

支持6种格式的订阅,使用如下:

- SHSE.* : 上交所,所有数据
- SHSE.600000.* : 上交所,600000,所有数据
- SHSE.600000.tick : 上交所,600000, tick数据
- SHSE.600000.bar.60: 上交所, 600000, 1分钟(60秒)Bar数据
- SHSE.600000.*,SHSE.600004.* : 上交所,600000和600004所有数据(订阅多个代码)

init_with_config

使用配置文件初始化策略，配置文件已设置服务地址，账号密码，策略ID，行情模式以及需要订阅的symbol列表

函数原型：

```
__init__(self,
         username,
         password,
         strategy_id,
         subscribe_symbols='',
         mode=2,
         td_addr='',
         config_file=None,
         config_file_encoding='utf-8')
```

参数：

参数名	类型	说明
config_file	string	策略配置文件路径
config_file_encoding	string	策略配置文件编码

返回值：

返回值编码

示例：

strategy.ini文件配置：

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;策略基本配置
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
[strategy]
;掘金用户名
username=-

;掘金密码
password=-

;策略ID
strategy_id=3c7e70be-7e02-11e5-b293-5ec5d48ea63a

;订阅证券代码或合约代码列表
subscribe_symbols=SHSE.600000.tick,SHSE.600000.bar.60

;行情模式，2-实时行情模式，3-模拟行情模式，4-回放行情模式
mode=2

;交易服务地址，使用掘金终端交易时，地址为localhost:8001，如果此项配置为空，则订单发往掘金云服务器
td_addr=localhost:8001

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;策略回测参数配置
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
[backtest]
;历史数据回放开始时间
start_time=2014-10-25 09:30:00

;历史数据回放结束时间
end_time=2015-10-29 15:15:00

;策略初始资金
initial_cash=1000000

;委托量成交比率，默认=1（每个委托100%成交）
transaction_ratio=1

;手续费率，默认=0（不计算手续费）
commission_ratio=0.0008

;滑点比率，默认=0（无滑点）
slippage_ratio=0.246

;行情复权模式,0=不复权,1=前复权
price_type=1

;基准
bench_symbol=SHSE.000300
```

```
ret = Strategy(config_file="strategy.ini")
```

backtest_config

回测参数设置, 仅在回测模式下有效

函数原型：

```
backtest_config(self,
```

```
                start_time,
                end_time,
                initial_cash=1000000,
                transaction_ratio=1,
                commission_ratio=0,
                slippage_ratio=0,
                price_type=1,
                bench_symbol='SHSE.000300',
                check_cache=1)
```

参数：

参数名	类型	说明
start_time	string	回放行情开始时间，格式：`yyyy-mm-dd HH:MM:SS`
end_time	string	回放行情结束时间，格式：`yyyy-mm-dd HH:MM:SS`
initial_cash	float	回测初始资金，默认1000000
transaction_ratio	float	委托量成交比率,默认1，按委托量全部成交
commission_ratio	float	手续费率，默认0，无手续费
slippage_ratio	float	滑点比率，默认0，无滑点
price_type	int	复权方式，0-不复权，1-前复权
bench_symbol	string	基准代码
check_cache	int	回测时是否使用本地缓存数据，0-不使用，1-使用, 默认为值为 1

返回值：

返回值编码

策略启停方法

run

运行策略

函数原型：

```
run()
```

参数：

无

返回值：

返回值编码

stop

停止策略

参数：

无

返回值：

无

行情订阅方法

subscribe

订阅行情,策略类和行情服务了都提供该接口。

函数原型:

```
subscribe(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	订阅代码列表

返回值:

返回值编码

示例：

通过策略对象订阅上交所浦发银行Tick和1分钟Bar实时数据

```
strategy.subscribe("SHSE.600000.tick,SHSE.600000.bar.60")
```

注意项：

symbol_list 订阅代码表的参数格式 当在回测模式下，需要显式关闭本地缓存api才能正常工作, 如何关闭本地缓存，请参见 backtest_config

unsubscribe

退订指定行情，策略类和行情服务了都提供该接口。

函数原型:

```
unsubscribe(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	订阅代码列表

返回值:

返回值编码

示例：

通过策略对象退订上交所浦发银行Tick和1分钟Bar实时数据

```
strategy.unsubscribe("SHSE.600000.tick,SHSE.600000.bar.60")
```

注意项：

symbol_list 订阅代码表的参数格式 当在回测模式下，需要显式关闭本地缓存api才能正常工作, 如何关闭本地缓存，请参见 backtest_config

resubscribe

重置订阅条件，相当于先退订原来所有行情，再重新订阅指定的行情

函数原型:

```
resubscribe(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	订阅代码列表

返回值:

返回值编码。

symbol_list 订阅代码表的参数格式 当在回测模式下，需要显式关闭本地缓存api才能正常工作, 如何关闭本地缓存，请参见 backtest_config

数据提取方法

get_ticks

提取指定时间段的历史Tick数据，支持单个代码提取或多个代码组合提取。策略类和行情服务类都提供该接口。

函数原型:

```
get_ticks(symbol_list, begin_time, end_time)
```

参数：

参数名	类型	说明
symbol_list	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码
begin_time	string	开始时间, 如2015-10-30 09:30:00
end_time	string	结束时间, 如2015-10-30 15:00:00

返回值：

Tick列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行2015-10-30 09:30:00到2015-10-30 15:00:00时间段的所有Tick数据

```
ticks = strategy.get_ticks("SHSE.600000,SZSE.000001",
            "2015-10-30 09:30:00",
            "2015-10-30 15:00:00")
```

get_bars

提取指定时间段的历史Bar数据，支持单个代码提取或多个代码组合提取。策略类和行情服务类都提供该接口。

函数原型:

```
get_bars(symbol_list, bar_type, begin_time, end_time)
```

参数：

参数名	类型	说明
symbol_list	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码
bar_type	int	bar周期，以秒为单位，比如60即1分钟bar
begin_time	string	开始时间, 如2015-10-30 09:30:00
end_time	string	结束时间, 如2015-10-30 15:00:00

返回值：

Bar列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行2015-10-30 09:30:00到2015-10-30 15:00:00时间段的1分钟Bar数据

```
bars = strategy.get_bars("SHSE.600000,SZSE.000001",
            60,
            "2015-10-30 09:30:00",
            "2015-10-30 15:00:00")
```

get_dailybars

提取指定时间段的历史日周期Bar数据，支持单个代码提取或多个代码组合提取。DailyBar比Bar多了部分静态数据，如结算价，涨跌停等。策略类和行情服务类都提供该接口。

函数原型:

```
get_dailybars(symbol_list, begin_time, end_time)
```

参数：

参数名	类型	说明
symbol_list	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码
begin_time	string	开始日期, 如2015-10-19
end_time	string	结束日期, 如2015-10-30

返回值：

DailyBar列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行2015-10-19到2015-10-30时间段的DailyBar数据

```
var dailybars = strategy.get_dailybars("SHSE.600000,SZSE.000001",
            "2015-10-19 00:00:00",
            "2015-10-30 00:00:00")
```

get_last_ticks

提取最新的1条Tick数据，支持单个代码提取或多个代码组合提取。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_ticks(symbol_list)
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码

返回值：

Tick列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行的最近一笔Tick数据

```
ticks = strategy.get_last_ticks("SHSE.600000,SZSE.000001")
```

get_last_bars

提取最新1条Bar数据，支持单个代码提取或多个代码组合提取。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_bars(symbol_list, bar_type)
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码
bar_type	int	bar周期，以秒为单位，比如60即1分钟bar

返回值：

Bar列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行的最新的1笔1min Bar数据

```
bars = strategy.get_last_bars("SHSE.600000,SZSE.000001", 60)
```

get_last_dailybars

提取最新1条DailyBar数据，支持单个代码提取或多个代码组合提取。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_dailybars(symbol_list)
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000，同时支持多只代码

返回值：

DailyBar列表

示例：

通过策略对象提取上交所浦发银行和深交所平安银行的最近一笔DailyBar数据

```
dailybars = strategy.get_last_dailybars("SHSE.600000,SZSE.000001")
```

get_last_n_ticks

提取单个代码最新n条Tick数据,策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_ticks(symbol, n, end_time='')
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000
n	int	提取的数据条数
end_time	string	指定截止时间, 如2015-10-30 15:00:00

返回值：

Tick列表

示例：

通过策略对象提取上交所浦发银行的最近10笔Tick数据

```
dailybars = strategy.get_last_n_ticks("SHSE.600000")
```

get_last_n_bars

提取单个代码的最新n条Bar数据,策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_bars(symbol, bar_type, n, end_time='')
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000
bar_type	int	bar周期，以秒为单位，比如60即1分钟bar
n	int	提取的数据条数
end_time	string	指定截止时间, 如2015-10-30 15:00:00

返回值：

Bar列表

示例：

通过策略对象提取上交所浦发银行的最近10笔1分钟Bar数据

```
dailybars = strategy.get_last_n_bars("SHSE.600000", 60,  10)
```

get_last_n_dailybars

提取单个代码的最新n条DailyBar数据, 策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_dailybars(symbol, n, end_time='')
```

参数：

参数名	类型	说明
symbol	string	证券代码, 带交易所代码以确保唯一，如SHSE.600000
n	int	提取的数据条数
end_time	string	指定截止时间, 如2015-10-30 15:00:00

返回值：

Bar列表

示例：

通过策略对象提取上交所浦发银行的最近10笔1min DailyBar数据

```
dailybars = strategy.get_last_n_dailybars("SHSE.600000", 10)
```

get_instruments

提取交易代码。策略类和行情服务类都提供该接口。

函数原型:

```
get_instruments(exchange, sec_type, is_active)
```

参数：

参数名	类型	说明
exchange	string	交易所代码
sec_type	int	代码类型:1 股票，2 基金，3 指数，4 期货，5 ETF
is_active	int	当天是否交易：1 是，0 否

返回值：

Instrument对象列表

get_instruments_by_name

根据期货品种提取交易代码。策略类和行情服务类都提供该接口。

函数原型:

```
get_instruments_by_name(name)
```

参数：

参数名	类型	说明
name	string	期货品种，如'ag', 'ic'

返回值：

Instrument对象列表

get_constituents

提取指数的成分股代码。策略类和行情服务类都提供该接口。

函数原型:

```
get_constituents(index_symbol)
```

参数：

参数名	类型	说明
index_symbol	string	指数代码

返回值：

Constituent对象列表

get_financial_index

按时间周期提取FinancialIndex,按时间升序排列。策略类和行情服务类都提供该接口。

函数原型:

```
get_financial_index(symbol, t_begin, t_end)
```

参数：

参数名	类型	说明
symbol	string	品种代码, 如SHSE.600000
t_begin	string	开始时间, 如2013-8-14 00:00:00
t_end	string	结束时间, 如2013-8-15 00:00:00

返回值：

FinancialIndex对象列表

get_last_financial_index

提取快照, 即最新的FinancialIndex，支持一次性提取多个代码的快照。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_financial_index(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	多个品种代码列表, 如SHSE.600000,SZSE.000001

返回值：

FinancialIndex对象列表

get_last_n_financial_index

提取最近n条FinancialIndex。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_financial_index(symbol, n)
```

参数：

参数名	类型	说明
symbol	string	代码, 如SHSE.600000
n	string	数据个数

返回值：

FinancialIndex对象列表

get_share_index

按时间周期提取ShareIndex,按时间升序排列。策略类和行情服务类都提供该接口。

函数原型:

```
get_share_index(symbol, t_begin, t_end)
```

参数：

参数名	类型	说明
symbol	string	品种代码, 如SHSE.600000
t_begin	string	开始时间, 如2013-8-14 00:00:00
t_end	string	结束时间, 如2013-8-15 00:00:00

返回值：

ShareIndex对象列表

get_last_share_index

提取快照, 即最新的ShareIndex，支持一次性提取多个代码的快照。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_share_index(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	多个品种代码列表, 如SHSE.600000,SZSE.000001

返回值：

ShareIndex对象列表

get_last_n_share_index

提取最近n条ShareIndex。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_share_index(symbol, n)
```

参数：

参数名	类型	说明
symbol	string	代码, 如SHSE.600000
n	string	数据个数

返回值：

ShareIndex对象列表

get_market_index

按时间周期提取MarketIndex,按时间升序排列。策略类和行情服务类都提供该接口。

函数原型:

```
get_market_index(symbol, t_begin, t_end)
```

参数：

参数名	类型	说明
symbol	string	品种代码, 如SHSE.600000

t_begin	string	开始时间, 如2013-8-14 00:00:00
t_end	string	结束时间, 如2013-8-15 00:00:00

返回值：

MarketIndex对象列表

get_last_market_index

按时间周期提取MarketIndex,按时间升序排列。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_market_index(symbol_list)
```

参数：

参数名	类型	说明
symbol_list	string	多个品种代码列表, 如SHSE.600000,SZSE.000001

返回值：

MarketIndex对象列表

get_last_n_market_index

按时间周期提取MarketIndex,按时间升序排列。策略类和行情服务类都提供该接口。

函数原型:

```
get_last_n_market_index(symbol, n)
```

参数：

参数名	类型	说明
symbol	string	代码, 如SHSE.600000
n	string	数据个数

返回值：

MarketIndex对象列表

get_calendar

获取交易所交易日历。策略类和行情服务类都提供该接口。

函数原型:

```
get_calendar(exchange, start_time, end_time)
```

参数：

参数名	类型	说明
exchange	string	交易所, 如SHSE
start_time	string	开始时间, 如2016-01-01
end_time	string	结束时间, 如2016-03-15

返回值：

TradeDate对象列表

get_stock_adj

查询复权因子

函数原型:

```
get_stock_adj(symbol, start_time, end_time)
```

参数：

参数名	类型	说明
symbol	string	如:SZSE.000001
start_time	string	查询开始时间
end_time	string	查询结束时间

返回值：

StockAdjustFactor 列表

get_divident

查询分红送股明细

函数原型:

```
get_divident(symbol, start_time, end_time)
```

参数：

参数名	类型	说明
symbol	string	如:SZSE.000001
start_time	string	查询开始时间
end_time	string	查询结束时间

返回值：

StockDivident 列表

get_virtual_contract

查询虚拟合约和真实合约对应关系

函数原型:

```
get_virtual_contract(vsymbol, start_time, end_time)
```

参数：

参数名	类型	说明
vsymbol	string	如:CFFEX.IF, CFFEX.IF00
start_time	string	查询开始时间
end_time	string	查询结束时间

返回值：

返回VirtualContract列表

交易接口方法

open_long

异步开多仓，以参数指定的symbol、价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口

函数原型:

```
open_long(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上交所SHSE
sec_id	string	证券代码,如浦发银行600000
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	委托量

返回值：

委托下单生成的Order对象

示例：

市价买入1000股上交所浦发银行

```
order = open_long("SHSE", "600000", 0, 1000)
```

open_short

异步开空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
open_short(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如金融期货交易所 CFFEX
sec_id	string	证券代码,如股指期货合约1511 IF1511
price	float	委托价，如果price=0,为市价单，否则为限价单

volume	float	委托量

返回值：

委托下单生成的Order对象

示例：

在策略类的方法中以市价开1手股指期货合约IF1511的空单

```
order = open_short("CFFEX", "IF1511", 0, 1)
```

注意事项：

1. 该接口为异步下单接口，需策略Run之后才能正常运行
2. 平仓接口只对期货有效，现货不存在空仓单

close_long

异步平多仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
close_long(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上交所 SHSE
sec_id	string	证券代码,如浦发银行 600000
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

委托下单生成的Order对象

示例：

在策略类的方法中以市价卖出1000股上交所浦发银行

```
order = close_long("SHSE", "600000", 0, 1000)
```

注意事项：

该接口为异步下单接口，需策略Run之后才能正常运行

close_long_yesterday

异步平昨多仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。 此api用于平上期所昨仓，策略类和交易服务类都提供该接口。

函数原型:

```
close_long_yesterday(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上期所SHFE
sec_id	string	证券代码
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

委托下单生成的Order对象

示例：

以市价平1手白银合约ag1512的多单,

```
order = close_long_yesterday("SHFE", "ag1512", 0, 1)
```

注意事项：

1. 该接口为异步下单接口，需策略Run之后才能正常运行
2. 平仓接口只对期货有效，现货不存在平仓

close_short

异步平空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
close_short(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如股指期货交易所CFFEX
sec_id	string	证券代码,如股指期货合约IF1511
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

在策略类的方法中以市价平1手股指期货合约IF1511的空单

```
order = close_short("CFFEX", "IF1511", 0, 1)
```

注意事项：

1. 该接口为异步下单接口，需策略Run之后才能正常运行
2. 平仓接口只对期货有效，现货不存在平仓

close_short_yesterday

异步平昨空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。 此api用于平上期所昨仓，策略类和交易服务类都提供该接口。

函数原型:

```
close_short_yesterday(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码，如上期所SHFE

sec_id	string	证券代码
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

以市价平1手白银合约ag1512的空单

```
order = close_short_yesterday("SHFE", "ag1512", 0, 1)
```

注意事项：

1. 该接口为异步下单接口，需策略Run之后才能正常运行
2. 平仓接口只对期货有效，现货不存在平仓

place_order

异步下单原生函数，需要创建Order对象，填充对应字段，一般异步下单接口建议使用open_long、open_short、close_long、close_short4个快捷下单接口。如果价格price字段为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
place_order(order)
```

参数：

参数名	类型	说明
order	Order	委托Order对象

返回值：

直接返回参数Order对象

示例：

在策略类的方法中以市价开1手IF1511的多单

```
order = Order()
order.exchange = "CFFEX"
order.sec_id = "IF1511"
order.side = 1
order.position_effect = 1
order.price = 0
order.volume = 1

order_ret = place_order(order)
```

注意事项：

该接口为异步下单接口，需策略Run之后才能正常运行

cancel_order

异步撤单接口，根据参数cl_ord_id指定的客户端订单ID，撤销之前的下单委托。撤单是否成功取决于订单当前的状态。

函数原型:

```
cancel_order(cl_ord_id)
```

参数：

参数名	类型	说明
cl_ord_id	string	委托订单的客户方id

返回值：

返回值编码

示例：

```
ret = cancel_order(order.cl_ord_id)
```

注意事项：

1. 该接口为异步下单接口，需策略Run之后才能正常运行
2. 执行的结果由on_execrpt, on_order_cancelled, on_order_cancel_rejected回调方法返回

open_long_sync

同步开多仓接口，以参数指定的symbol, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
open_long_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上交所SHSE
sec_id	string	证券代码,如浦发银行600000
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	委托量

返回值：

返回委托下单生成的Order对象

示例：

通过策略对象调用该接口，以市价买入1000股上交所浦发银行

```
order = strategy.open_long_sync("SHSE", "600000", 0, 1000)
```

open_short_sync

同步开空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
open_short_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如金融期货交易所 CFFEX
sec_id	string	证券代码,如股指期货1511合约 IF1511
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	委托量

返回值：

返回委托下单生成的Order对象

示例：

通过策略对象调用该接口，以市价开1手股指期货IF1511合约的空单

```
strategy.open_short_sync("CFFEX", "IF1511", 0, 1)
```

注意事项：

平仓接口只对期货有效，现货不存在空仓单

close_long_sync

同步平多仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
close_long_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上交所SHSE
sec_id	string	证券代码,如浦发银行 600000
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

通过策略对象调用该接口，以市价卖出1000股上交所浦发银行

```
strategy.close_long("SHSE", "600000", 0, 1000)
```

close_long_yesterday_sync

同步平昨多仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。 此api用于平上期所昨仓，策略类和交易服务类都提供该接口。

函数原型:

```
close_long_yesterday_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码，如上期所SHFE
sec_id	string	证券代码，如浦发银行 600000
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

以市价平1手白银合约ag1512的多单

```
order = close_long_yesterday_sync("SHFE", "ag1512", 0, 1)
```

注意事项：

1.平仓接口只对期货有效，现货不存在平仓

close_short_sync

同步平空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
close_short_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明

exchange	string	交易所代码, 如股指期货交易所CFFEX
sec_id	string	证券代码,如股指期货合约IF1511
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

通过策略对象调用该接口，在策略类的方法中以市价平1手股指期货合约IF1511的空单

```
strategy.close_short_sync("CFFEX", "IF1511", 0, 1)
```

注意事项：

平仓接口只对期货有效，现货不存在平仓

close_short_yesterday_sync

同步平昨空仓接口，以参数指定的exchange, 证券代码sec_id, 价和量下单。如果价格为0，为市价单，否则为限价单。此api用于平上期所昨仓，策略类和交易服务类都提供该接口。

函数原型:

```
close_short_yesterday_sync(exchange, sec_id, price, volume)
```

参数：

参数名	类型	说明
exchange	string	交易所代码, 如上期所SHFE
sec_id	string	证券代码,如股指期货合约IF1511
price	float	委托价，如果price=0,为市价单，否则为限价单
volume	float	平仓量

返回值：

返回委托下单生成的Order对象

示例：

以市价平1手白银合约ag1512的空单

```
strategy.close_short_yesterday_sync("SHFE", "ag1512", 0, 1)
```

注意事项：

平仓接口只对期货有效，现货不存在平仓

place_order_sync

同步下单原生函数，需要创建Order对象，填充对应字段，一般同步下单接口建议使用open_long_sync、open_short_sync、close_long_sync、close_short_sync 4个快捷下单接口。如果价格price字段为0，为市价单，否则为限价单。策略类和交易服务类都提供该接口。

函数原型:

```
place_order_sync(order)
```

参数：

参数名	类型	说明
order	Order	委托Order对象

返回值：

直接返回参数Order对象

示例：

通过策略对象调用该接口，以市价开1手IF1511的多单

```
order = Order()
order.exchange = "CFFEX"
order.sec_id = "IF1511"
order.side = 1
order.position_effect = 1
order.price = 0
order.volume = 1

order_ret = place_order_sync(order)
```

cancel_order_sync

同步撤单接口，根据参数clordid指定的客户端订单ID，撤销之前的下单委托。撤单是否成功取决于订

单当前的状态。策略类和交易服务类都提供该接口。

函数原型:

```
cancel_order_sync(cl_ord_id)
```

参数：

参数名	类型	说明
cl_ord_id	string	委托订单的客户方id

返回值：

返回值编码

示例：

```
ret = cancel_order_sync(order.cl_ord_id)
```

注意事项：

执行的结果由on_execrpt, on_order_cancelled, on_order_cancel_rejected回调方法返回

get_order

查询单个委托信息，策略类和交易服务类都提供该接口。

函数原型:

```
get_order(cl_ord_id)
```

参数：

参数名	类型	说明
cl_ord_id	string	委托订单的客户方id

返回值：

order对象

get_orders

按时间段查询委托信息列表，策略类和交易服务类都提供该接口。

函数原型:

```
get_orders(start_time, end_time)
```

参数：

参数名	类型	说明
start_time	string	开始时间，如2016-01-01 00:00:00
end_time	string	开始时间，如2016-01-02 00:00:00

返回值：

order对象列表

get_orders_by_symbol

按时间段和代码查询委托信息列表，策略类和交易服务类都提供该接口。

函数原型:

```
get_orders_by_symbol(exchange, sec_id, start_time, end_time)
```

参数：

参数名	类型	说明
exchange	string	交易所名称， 如SHSE
sec_id	string	证券代码，如600000
start_time	string	开始时间，如2016-01-01 00:00:00
end_time	string	开始时间，如2016-01-02 00:00:00

返回值：

order对象列表

get_unfinished_orders

查询未完成委托信息，策略类和交易服务类都提供该接口。

函数原型:

```
get_unfinished_orders()
```

参数：

无

返回值：

order对象列表

get_cash

查询当前策略的资金信息，策略类和交易服务类都提供该接口。

函数原型:

```
get_cash()
```

参数：

无

返回值：

Cash对象，当前策略的资金信息

示例：

在策略类的方法中查询当前策略的资金信息

```
cash = get_cash()
```

get_position

查询当前策略指定symbol（由交易所代码和证券ID组成）和买卖方向的持仓信息。策略类和交易服务类都提供该接口。

函数原型:

```
get_position(exchange, sec_id, side);
```

参数：

参数名	类型	说明
exchange	string	交易所代码
sec_id	string	证券代码
side	int	买卖方向

返回值：

Position对象，持仓信息

示例：

在策略类的方法中查询买入浦发银行的持仓信息

```
postion = get_position("SHSE", "600000", 1)
```

get_positions

查询当前策略的全部持仓信息。策略类和交易服务类都提供该接口。

函数原型:

```
get_positions()
```

参数：

无

返回值：

当前策略全部持仓列表

示例：

在策略类的方法中查询当前策略全部持仓信息

```
postion = get_positions()
```

get_indicator

查询当前策略的绩效信息。策略类和交易服务类都提供该接口。

函数原型:

```
get_indicator()
```

参数：

无

返回值：

当前策略的绩效信息Indicator

示例：

在策略类的方法中查询当前策略的绩效信息

```
indicator = get_indicator()
```

get_broker_accounts

获取柜台交易账号列表。策略类和交易服务类都提供该接口。

函数原型:

```
get_broker_accounts()
```

参数：

无

返回值：

BrokerAccount对象列表

get_broker_cash

获取柜台交易账号资金。策略类和交易服务类都提供该接口。

函数原型:

```
get_broker_cash()
```

参数：

无

返回值：

Cash对象列表

get_broker_positions

获取柜台交易账号持仓。策略类和交易服务类都提供该接口。

函数原型:

```
get_broker_positions()
```

参数：

无

返回值：

Position对象列表

行情事件方法

on_tick

响应Tick事件，收到Tick数据后本函数被调用。

函数原型:

```
on_tick(tick)
```

参数：

参数名	类型	说明
tick	Tick	Tick数据

返回值：

无

on_bar

响应Bar事件，收到Bar数据后本函数被调用。

函数原型:

```
on_bar(bar)
```

参数：

参数名	类型	说明
bar	Bar	Bar数据

返回值：

无

交易事件方法

on_execrpt

响应委托执行回报事件，收到Execution数据后本函数被调用。

函数原型:

```
on_execrpt(rpt)
```

参数：

参数名	类型	说明
rpt	ExecRpt	执行回报数据

返回值：

无

on_order_rejected

响应订单被拒绝事件，收到Order变更数据后本函数被调用。

函数原型:

```
on_order_rejected(order)
```

参数：

参数名	类型	说明

order	Order	最新的订单数据

返回值：

无

on_order_new

响应订单被柜台接收事件，收到Order变更数据后本函数被调用。

函数原型:

```
on_order_new(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_status

响应订单状态更新事件，Order状态变更后本函数被调用。

函数原型:

```
on_order_status(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_filled

响应订单完全成交事件，收到Order变更数据后本函数被调用。

函数原型:

```
on_order_filled(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_partially_filled

响应订单部分成交事件，收到Order变更数据后本函数被调用。

函数原型:

```
on_order_partially_filled(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_stop_executed

响应订单停止执行事件，比如，限价单到收市仍然未能成交。收到Order变更数据后本函数被调用。

函数原型:

```
on_order_stop_executed(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_cancelled

响应订单撤单成功事件，收到Order变更数据后本函数被调用。

函数原型:

```
on_order_cancelled(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

on_order_cancel_rejected

响应订单撤单请求被拒绝事件，收到Execution数据后本函数被调用。ord_rej_reason说明为什么撤单失败。

函数原型:

```
on_order_cancel_rejected(order)
```

参数：

参数名	类型	说明
order	Order	最新的订单数据

返回值：

无

其他方法

on_login

策略登录事件，初始化策略时本函数被调用。

函数原型:

```
on_login()
```

参数：

无

返回值：

无

on_error

响应错误事件，策略内部出现错误时，比如行情或交易连接断开，数据错误，超时等，将触发本函数。

函数原型:

```
on_error(error_code, error_msg)
```

参数：

参数名	类型	说明
error_code	int	返回值编码
error_msg	string	错误信息

返回值：

无

on_backtest_finish

回测结束事件，在回测结束时触发。

函数原型:

```
on_backtest_finish(indicator)
```

参数：

参数名	类型	说明
indicator	object	回测的绩效

返回值：

无

set_timer

设置策略定时事件的时间间隔，单位为毫秒。

函数原型:

```
set_timer(interval)
```

参数：

参数名	类型	说明
interval	int	定时器时间间隔

返回值：

无

unset_timer

解除该时间间隔对应的定时器，单位为毫秒。

函数原型:

```
unset_timer(interval)
```

参数：

参数名	类型	说明
interval	int	定时器时间间隔

返回值：

无

on_timer

策略定时事件，按设定的时间间隔定时调用。

函数原型:

```
on_timer(interval)
```

参数：

参数名	类型	说明
interval	int	定时器时间间隔

返回值：

无

set_timeout_val

设置同步API的超时时间，系统默认为30秒

函数原型:

```
set_timeout_val(seconds)
```

参数：

参数名	类型	说明
seconds	int	超时时间，单位为秒

返回值：

无

get_timeout_val

获取同步API的超时时间

函数原型:

```
get_timeout_val()
```

参数：

无

返回值：

超时时间，单位为秒

get_strerror

根据错误码错误详细错误信息

函数原型:

```
get_strerror(err_code)
```

参数：

参数名	类型	说明
err_code	int	返回值编码

返回值：

错误信息

on_md_event

响应行情状态事件，收到MarketDataEvent数据后本函数被调用。

函数原型:

```
on_md_event(md_event)
```

参数：

参数名	类型	说明
md_event	MDEvent	开盘，收盘，回放行情结束等事件

返回值：

无

get_version

获取SDK版本号。

函数原型:

```
get_version()
```

参数：

无

返回值：

返回当前版本号信息

to_dict

类型转换函数，将GMSDK内置类型对象转换为dict对象; 使用前从gmsdk引入,即`from gmsdk import to_dict`。

函数原型:

```
to_dict(obj)
```

参数：

参数名	类型	说明
obj	object	GMSDK内置类型对象

返回值：

dict对象

□ 上一章

下一章 □