箭鱼客户端Python策略接口文档

策略逻辑简介

Python接口连接箭鱼客户端后，可以通过调用API函数进行发单撤单，并以回调函数的方式处理行情和委托回报。目前本接口不支持订阅和查询，客户端会将自己订阅的产品行情数据推送到Python接口。接口和客户端的连接建立后会先进行消息同步，客户端会先将股票产品信息、账户资金、股票持仓、当日的全部委托和成交数据推送到Python接口，并发送同步完成消息，之后客户端会将OES交易通道数据转发到Python接口。Python接口的行情数据结构和MDS相同，交易回报数据和OES相同，用户可查看MDS和OES的相关文档。

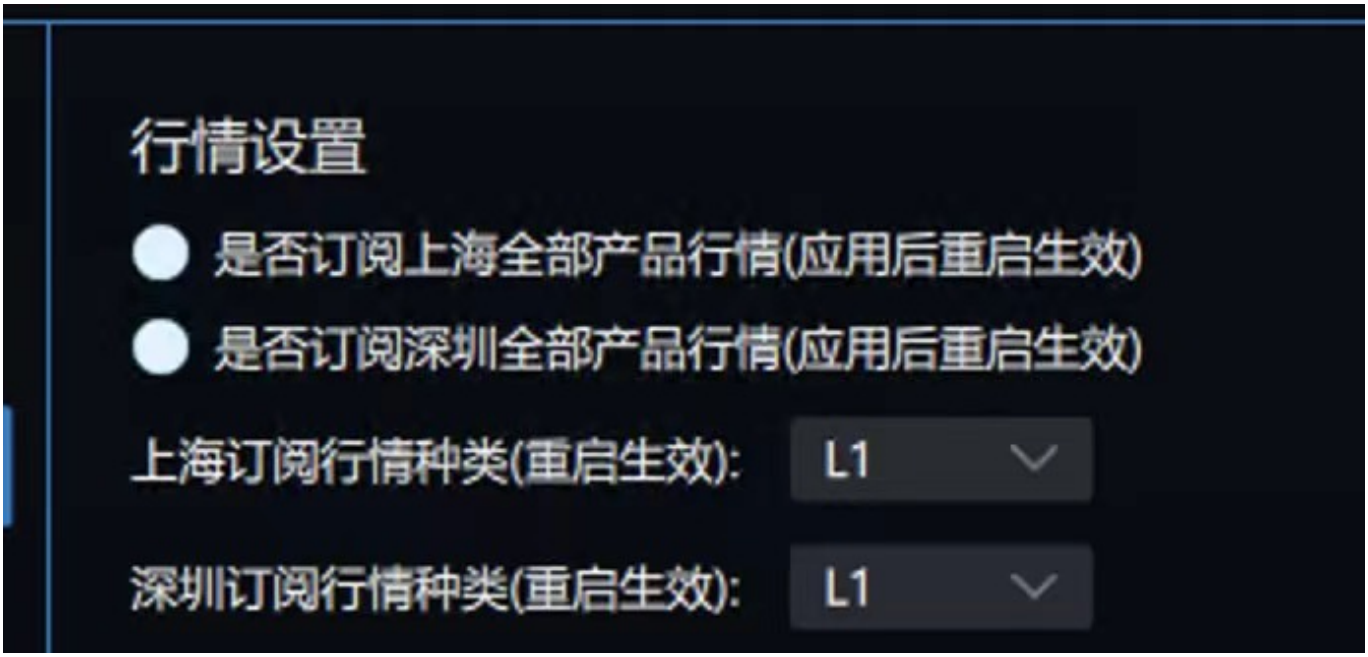
策略运行

在客户端导入策略时需要选择一个可执行文件作为入口，例如cmd脚本。示例中的strategy_exe.cmd中需要把Python解释器路径和Python脚本路径按实际情况修改：

```
C:\Python37\python.exe .\strategys\strategy_sample.py %1 %2 %3 %4 %5 %6 %7 %0
```

客户端启动策略进程时，会把策略进程的工作目录设置为客户端安装目录。

没有订阅接口，客户端会将自己订阅的产品行情数据推送到Python接口。 在客户端中可以设置订阅全部行情：



如果没有进行这项设置，那客户端订阅的产品为持仓加在下单框输入过的产品。

回调函数定义

基类如下

```python
class StrategyBase(object):

    def on_sync_asset(self, data):
        pass
```

```
    def on_sync_hold(self, data):
        pass

    def on_sync_ord(self, data):
        pass

    def on_sync_trd(self, data):
        pass

    def on_sync_stock(self, data):
        pass

    def on_sync_finish(self):
        pass

    def on_l1_snapshot(self, head, body):
        pass

    def on_l2_snapshot(self, head, body):
        pass

    def on_l2_trade(self, data):
        pass

    def on_l2_order(self, data):
        pass

    def on_business_reject(self, head, body):
        pass

    def on_order_insert(self, head, body):
        pass

    def on_order_report(self, head, body):
        pass

    def on_trade_report(self, head, body):
        pass

    def on_cash_variation(self, head, body):
        pass

    def on_stock_holding_variation(self, head, body):
        pass

    def on_quit(self):
        pass
```

- `on_sync_asset`

> 同步账户资金数据

参数	Python类型	对应的OES结构体
data	ctypes的Structure	OesCashAssetItemT

- `on_sync_hold`

> 同步账户股票持仓数据

参数	Python类型	对应的OES结构体
data	ctypes的Structure	OesStkHoldingItemT

- `on_sync_ord`

> 同步当天的委托数据

参数	Python类型	对应的OES结构体
data	ctypes的Structure	OesOrdItemT

- `on_sync_trd`

> 同步当天的成交数据

参数	Python类型	对应的OES结构体
data	ctypes的Structure	OesTrdItemT

- `on_sync_stock`

> 同步股票产品信息数据

参数	Python类型	对应的OES结构体
data	ctypes的Structure	OesStockItemT

- `on_sync_finish`

> 同步完成消息会触发该方法的调用

- `on_l1_snapshot`

> Level1快照数据的推送会触发该方法的调用

参数	Python类型	对应的MDS结构体
head	ctypes的Structure	MdsMktDataSnapshotHeadT
body	ctypes的Structure	MdsStockSnapshotBodyT

- `on_l2_snapshot`

> Level2快照数据的推送会触发该方法的调用

参数	Python类型	对应的MDS结构体
head	ctypes的Structure	MdsMktDataSnapshotHeadT
body	ctypes的Structure	MdsL2StockSnapshotBodyT

- `on_l2_trade`

> Level2逐笔成交数据的推送会触发该方法的调用

参数	Python类型	对应的MDS结构体
data	ctypes的Structure	MdsL2TradeT

- `on_l2_order`

> Level2逐笔委托数据的推送会触发该方法的调用

参数	Python类型	对应的MDS结构体
data	ctypes的Structure	MdsL2OrderT

- `on_business_reject`

> OES业务拒绝数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesOrdRejectT

- `on_order_insert`

> OES委托已生成数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesOrdCnfmT

- `on_order_report`

> 交易所委托回报数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesOrdCnfmT

- `on_trade_report`

> 交易所成交回报数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesTrdCnfmT

- `on_cash_variation`

> 资金变动回报数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesCashAssetReportT

- `on_stock_holding_variation`

> 股票持仓变动回报数据的推送会触发该方法的调用

参数	Python类型	对应的OES结构体
head	ctypes的Structure	OesRptMsgHeadT
body	ctypes的Structure	OesStkHoldingReportT

- `on_quit`

> 策略退出消息会触发该方法的调用

主动函数定义

Python模块`swordfish_api.strategy_engine`:

- `engine_quit`

> 退出策略

- `send_order`

> 下单

参数	类型	说明
security_id	字符串	股票代码,例如`'600000'`
mkt_id	int	见OES枚举体eOesMarketIdT
bs_type	int	见OES枚举体eOesBuySellTypeT
ord_type	int	见OES枚举体eOesOrdTypeShT eOesOrdTypeSzT
ord_qty	int	下单量
ord_price	int	下单价,1元=10000

返回值	类型	说明
strategy_ord_id	int	策略委托编号

- `send_notify_msg`

> 发送客户端通知消息

参数	类型	说明
msg	字符串	通知消息
msg_level	int	通知消息等级,见CLIENT_API_NOTIFY_LEVEL的相关常量定义

- `is_owned_by_myself`

> 判断OES回报信息对应的委托是不是本策略发出的

参数	类型	说明
user_info	int	OES回报数据中的userInfo.i64

Python模块`swordfish_api.strategy_base`:

- `do`

> 启动策略

参数	类型	说明
strategy	Python对象	策略实例

- `get_strategy_order_id`

> 获取OES委托回报对应的策略委托编号

参数	类型	说明
userinfo	Python对象	OES回报数据中的userInfo

返回值	类型	说明
strategy_ord_id	int	策略委托编号

运行说明

- swordfish_api文件夹需要放到客户端目录下。
- strategy_exe.cmd中的Python解释器路径和入口脚本路径需要根据实际情况修改。

调试方式

可以通过添加命令行参数，使得策略可以在其它IDE软件中启动，从而可以使用其它IDE软件提供的调试功能进行调试。

在IDE中执行策略代码前先执行：

```
import sys
sys.argv.append("tcp://127.0.0.1:20003")  # 交易流地址
sys.argv.append("tcp://127.0.0.1:20001")  # 行情流地址
sys.argv.append("tcp://127.0.0.1:30000")  # 委托流地址
sys.argv.append("test")  # 策略名称
sys.argv.append("99")    # 环境号
sys.argv.append("93635570")  # 策略id(随便填写一个整数)
sys.argv.append("0")     # 当前最大委托序号
sys.argv.append(r"D:\client_ui_strategy\strategy_exe.cmd")  # 策略可执行文件
的绝对路径
```

其中交易流地址、行情流地址、委托流地址和环境号可以从客户端信息中获取。

当前最大委托序号从策略委托列表获取。

代码示例

```
import sys
if '.' not in sys.path:
    sys.path.append('.')
```

```
from swordfish_api import strategy_engine
from swordfish_api.strategy_base import StrategyBase, do
from swordfish_api.client_api import CLIENT_API_NOTIFY_LEVEL_GENERAL
from swordfish_api.mds_struct import eMdsExchangeIdT
from swordfish_api.oes_struct import eOesMarketIdT, eOesBuySellTypeT,
eOesOrdTypeT


class MyStrategy(StrategyBase):
    def __init__(self):
        self.msg_count = 0

    def on_snapshot(self, head, body):
        self.msg_count += 1
        if self.msg_count > 10000:
            rc = strategy_engine.quit()
            print(f"发送策略主动退出消息, 设置退出标志且返回-1 rc: {rc}")
            return -1
        if self.msg_count % 50 == 0:
            if head.exchId == eMdsExchangeIdT.MDS_EXCH_SSE.value:
                mkt_id = eOesMarketIdT.OES_MKT_SH_ASHARE.value
            else:
                mkt_id = eOesMarketIdT.OES_MKT_SZ_ASHARE.value
            rc = strategy_engine.send_order(body.SecurityID.decode(),
mkt_id,

eOesBuySellTypeT.OES_BS_TYPE_BUY.value,

eOesOrdTypeT.OES_ORD_TYPE_LMT.value, 100,
                                            body.TradePx)
            print(f"发送委托信息 rc: {rc}")

    def on_l1_snapshot(self, head, body):
        print("on_l1_snapshot")
        self.on_snapshot(head, body)

    def on_l2_snapshot(self, head, body):
        print("on_l2_snapshot")
        self.on_snapshot(head, body)

    def on_l2_trade(self, data):
        print("on_l2_trade")

    def on_l2_order(self, data):
        print("on_l2_order")

    def on_business_reject(self, head, body):
        print("on_business_reject")

    def on_order_insert(self, head, body):
        print("on_order_insert")
        strategy_engine.send_notify_msg(f"
{strategy_engine.Engine.strategy_name} 收到交易数据",
                                        CLIENT_API_NOTIFY_LEVEL_GENERAL)
```

```
    def on_order_report(self, head, body):
        print("on_order_report")

    def on_trade_report(self, head, body):
        print("on_trade_report")

    def on_cash_variation(self, head, body):
        print("on_cash_variation")

    def on_stock_holding_variation(self, head, body):
        print("on_stock_holding_variation")


if __name__ == '__main__':
    do(MyStrategy())
```