OpenAPI 介绍

概述

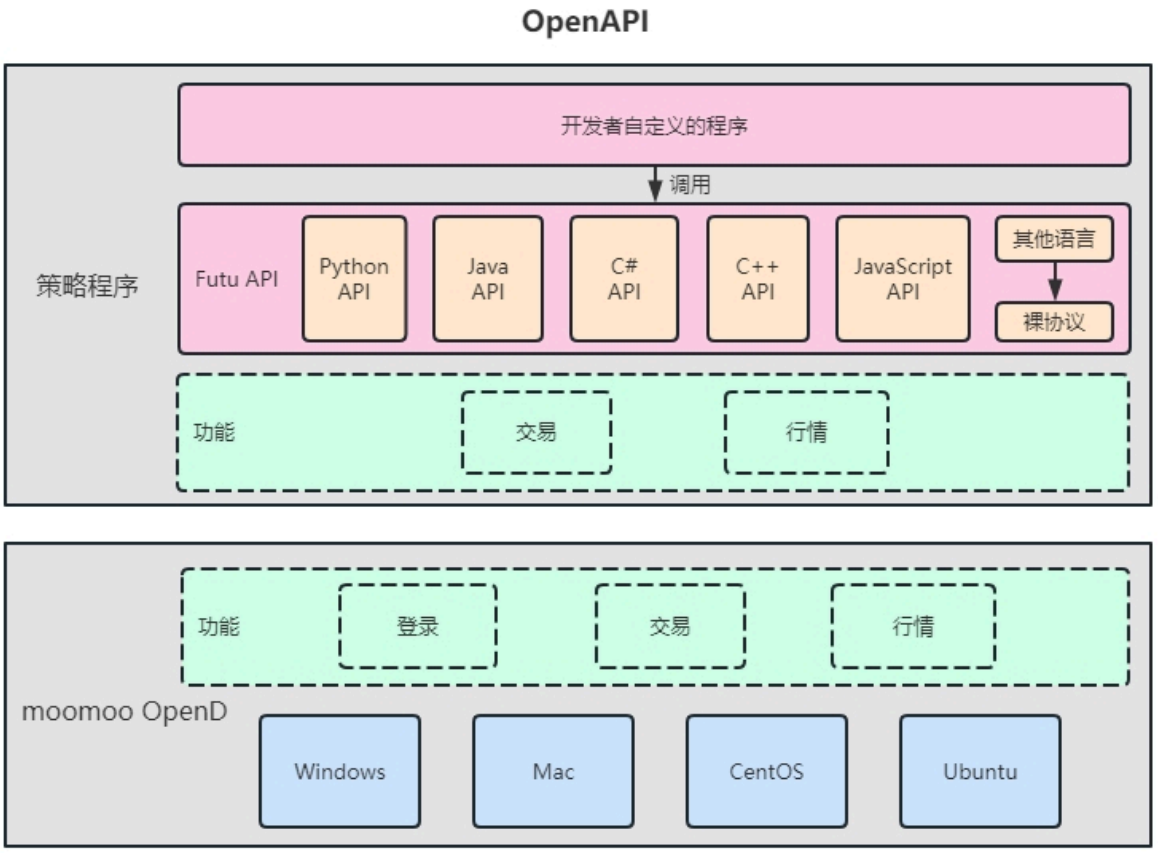
OpenAPI 量化接口，为您的程序化交易，提供丰富的行情和交易接口，满足每一位开发者的量化投资需求，助力您的宽客梦想。

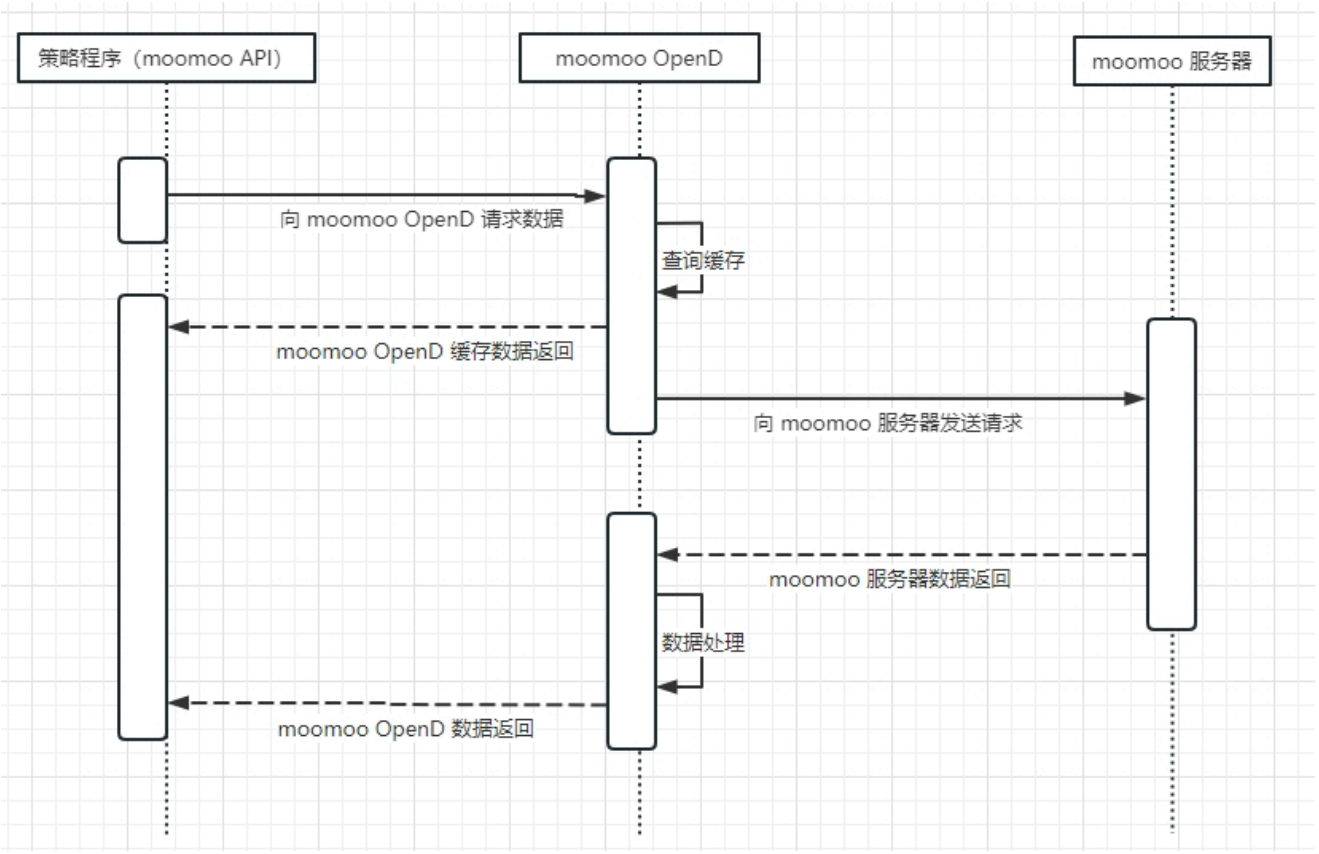
moomoo 用户可以 点击这里 了解更多。

OpenAPI 由 OpenD 和 moomoo API 组成：

- OpenD 是 moomoo API 的网关程序，运行于您的本地电脑或云端服务器，负责中转协议请求到富途后台，并将处理后的数据返回。
- moomoo API 是富途为主流的编程语言（Python、Java、C#、C++、JavaScript）封装的 API SDK，以方便您调用，降低策略开发难度。如果您希望使用的语言没有在上述之列，您仍可自行对接裸协议，完成策略开发。

下面的框架图和时序图，帮助您更好地了解 OpenAPI。





初次接触 OpenAPI，您需要进行如下两步操作：

第一步，在本地或云端安装并启动一个网关程序 OpenD。

OpenD 以自定义 TCP 协议的方式对外暴露接口，负责中转协议请求到富途服务器，并将处理后的数据返回，该协议接口与编程语言无关。

第二步，下载 moomoo API，完成 环境搭建，以便快速调用。

为方便您的使用，富途对主流的编程语言，封装了相应的 API SDK（以下简称 moomoo API）。

账号

OpenAPI 涉及 2 类账号，分别是 **平台账号** 和 **综合账户**。

平台账号

平台账号是您在 moomoo 的用户 ID（moomoo 号），此账号体系适用于moomoo APP、OpenAPI。 您可以使用平台账号（moomoo 号）和登录密码，登录 OpenD 并获取行情。

综合账户

综合账户支持以多种货币在同一个账户内交易不同市场品类（港股、美股、A股通、基金）。您可以通过一个账户进行全市场交易，不需要再管理多个账户。
综合账户包括综合账户 - 证券，综合账户 - 期货等业务账户：

- 综合账户 - 证券，用于交易全市场的股票、ETFs、期权等证券类产品。
- 综合账户 - 期货，用于交易全市场的期货产品，目前支持香港市场期货、美国市场 CME Group 期货、新加坡市场期货、日本市场期货。

功能

OpenAPI 的功能主要有两部分：行情和交易。

行情功能

行情数据品类

支持香港、美国、A 股市场的行情数据，涉及的品类包括股票、指数、期权、期货等，具体支持的品种见下表。
获取行情数据需要相关权限，如需了解行情权限的获取方式以及限制规则，请 点击这里。

市场	品种	moomoo 用户
香港市场	股票、ETFs、窝轮、牛熊、界内证	✓
	期权	✓
	期货	✓
	指数	✓
	板块	✓
美国市场	股票、ETFs	✓
	OTC 股票	X
	期权	✓
	期货	✓
	指数	X
	板块	✓
A 股市场	股票、ETFs	✓
	指数	✓
	板块	✓
新加坡市场	股票、ETFs、窝轮、REITs、DLCs	X
	期货	X
日本市场	股票、ETFs、REITs	X
	期货	X
澳大利亚市场	股票、ETFs	X
环球市场	外汇	X

行情数据获取方式

- 订阅并接收实时报价、实时 K 线、实时逐笔、实时摆盘等数据推送
- 拉取最新市场快照，历史 K 线等

交易功能

交易能力

支持香港、美国、A 股、新加坡、日本 5 个市场的交易能力，涉及的品类包括股票、期权、期货等，具体见下表：

市场	品种	模拟交易	真实交易						
			FUTU HK	Moomoo US	Moomoo SG	Moomoo AU	Moomoo MY	Moomoo CA	Moomoo JP
香港市场	股票、ETFs、窝轮、牛熊、界内证	✓	✓	✓	✓	✓	✓	X	X
	期权	✓	✓	X	X	X	X	X	X
	期货	✓	✓	X	X	X	X	X	X
美国市场	股票、ETFs	✓	✓	✓	✓	✓	✓	✓	✓
	期权	✓	✓	✓	✓	✓	✓	✓	✓
	期货	✓	✓	X	✓	X	✓	X	X
A 股市场	A 股通股票	✓	✓	✓	✓	X	✓	X	X
	非 A 股通股票	✓	X	X	X	X	X	X	X
新加坡市场	股票、ETFs、窝轮、REITs、DLCs	X	X	X	X	X	X	X	X
	期货	✓	✓	X	✓	X	X	X	X
日本市场	股票、ETFs、REITs	X	X	X	X	X	X	X	X
	期货	✓	✓	X	X	X	X	X	X

澳大利亚市场	股票、ETFs	X	X	X	X	X	X	X	X
加拿大市场	股票	X	X	X	X	X	X	X	X

交易方式

真实交易和模拟交易使用同一套交易接口。

特点

1. 全平台多语言：

- OpenD 支持 Windows、MacOS、CentOS、Ubuntu
- moomoo API 支持 Python、Java、C#、C++、JavaScript 等主流语言

2. 稳定极速免费：

- 稳定的技术架构，直连交易所一触即达
- 下单最快只需 0.0014 s
- 通过 OpenAPI 交易无附加收费

3. 丰富的投资品类：

- 支持美国、香港等多个市场的实时行情、实盘交易及模拟交易

4. 专业的机构服务：

- 定制化的行情交易解决方案

权限和限制

登录限制

开户限制

首先，您需要先在moomoo APP上，完成交易业务账户的开通，才能成功登录 OpenAPI。

合规确认

首次登录成功后，您需要完成问卷评估与协议确认，才能继续使用 OpenAPI。moomoo 用户请点击这里。

行情数据

行情数据的限制主要体现在以下几方面：

- 行情权限 —— 获取相关行情数据的权限
- 接口限频 —— 调用行情接口的频率限制
- 订阅额度 —— 同时订阅的实时行情的数量
- 历史 K 线额度 —— 每 30 天最多可拉取多少个标的的历史 K 线

行情权限

通过 OpenAPI 获取行情数据，需要相应的行情权限，OpenAPI 的行情权限跟 APP 的行情权限不完全一样，不同的权限等级对应不同的时延、摆盘档数以及接口使用权限。

部分品种行情，需要购买行情卡后方可获取，具体获取方式见下表。

市场	标的类别	获取方式

香港市场	证券类产品（含股票、ETFs、窝轮、牛熊、界内证）	* 中国内地IP客户：免费获取 LV2 行情。暂不支持获取 SF 权限。 * 港澳台及海外IP客户：免费获取 LV1 行情。如需获得 LV2 权限，请购买 港股 LV2 高级行情 。暂不支持获取 SF 权限。
	指数	
	板块	
	期权	* 中国内地IP客户：推广期免费获取 LV2 行情。 * 港澳台及海外IP客户：免费获取 LV1 行情，如需获得 LV2 权限，请购买 港股 LV2 + 期权期货 LV2 行情 。
	期货	
美国市场	证券类产品（含纽交所、美交所、纳斯达克上市的股票、ETFs）	* 与客户端行情权限不共用，如需获得 LV1 权限（基本报价，含夜盘），请购买 Nasdaq Basic 。 * 与客户端行情权限不共用，如需获得 LV2 权限（基本报价+深度摆盘，含夜盘深度摆盘），请购买 Nasdaq Basic+TotalView 。
	板块	
	OTC 股票	暂不支持获取
	期权（含普通股票期权、指数期权）	* 达到门槛 的客户：免费获得 LV1 权限。 * 未达到门槛 的客户：请购买 OPRA 期权 LV1 实时行情 获得 LV1 权限。
	期货	* 已开通期货账户 的客户： 如需获取 CME Group 行情 ，请购买 CME Group 期货 LV2 如需获取 CME 行情，请购买 CME 期货 LV2 如需获取 CBOT 行情，请购买 CBOT 期货 LV2 如需获取 NYMEX 行情，请购买 NYMEX 期货 LV2 如需获取 COMEX 行情，请购买 COMEX 期货 LV2 * 未开通期货账户的客户：不支持获取
	指数	暂不支持获取
A 股市场	证券类产品（含股票、ETFs）	* 中国内地 IP 个人客户：免费获取 LV1 行情。 * 港澳台及海外IP客户/机构客户：暂不支持。
	指数	
	板块	

提示

> 上述表格，中国内地IP客户和港澳台及海外IP客户，以 OpenD 登录的 IP 地址作为区分依据。

接口限频

为保护服务器，防止恶意攻击，所有需要向 moomoo 服务器发送请求的接口，都会有频率限制。
每个接口的限频规则会有不同，具体请参见每个接口页面下面的 接口限制 。

举例：
快照 接口的限频规则是：每 30 秒内最多请求 60 次快照。您可以每隔 0.5 秒请求一次匀速请求，也可以快速请求 60 次后，休息 30 秒，再请求下一轮。如果超出限频规则，接口会返回错误。

订阅额度 & 历史 K 线额度

订阅额度和历史 K 线额度限制如下：

用户类型	订阅额度	历史 K 线额度
开户用户	100	100
总资产达 1 万 HKD	300	300
以下三条满足任意一条即可： 1. 总资产达 50 万 HKD； 2. 月交易笔数 > 200； 3. 月交易额 > 200 万 HKD	1000	1000
以下三条满足任意一条即可： 1. 总资产达 500 万 HKD； 2. 月交易笔数 > 2000； 3. 月交易额 > 2000 万 HKD	2000	2000

1、总资产

总资产，是指您在 moomoo 证券的所有资产，包括：港、美、A 股证券账户，期货账户，基金资产以及债券资产，按照即时汇率换算成以港元为单位。

2、月交易笔数

月交易笔数，会综合您在 moomoo 证券的综合账户，在当前自然月与上一自然月的交易情况，

取您上个自然月的成交笔数与当前自然月的成交笔数的较大值进行计算，即：
max (上个自然月的成交笔数，当前自然月的成交笔数)。

3、月交易额

月交易额，会综合您在 moomoo 证券的综合账户，在当前自然月与上一自然月的交易情况，取您上个自然月的成交总金额与当前自然月的成交总金额的较大值进行计算，即：
max（上个自然月的成交总金额，当前自然月的成交总金额）
按照即期汇率换算成以港币为单位。其中，期货交易额的计算，需要乘以相应的调整系数（默认取 0.1），期货交易额计算公式如下：
期货交易额=∑（单笔成交数 * 成交价 * 合约乘数 * 汇率 * 调整系数）

4、订阅额度

订阅额度，适用于 订阅 接口。每只股票订阅一个类型即占用 1 个订阅额度，取消订阅会释放已占用的额度。 举例：
假设您的订阅额度是 100。 当您同时订阅了 HK.00700 的实时摆盘、US.AAPL 的实时逐笔、SH.600519 的实时报价时，此时订阅额度会占用 3 个，剩余的订阅额度为 97。 这时，如果您取消了 HK.00700 的实时摆盘订阅，您的订阅额度占用将变成 2 个，剩余订阅额度会变成 98。

5、历史 K 线额度

历史 K 线额度，适用于 获取历史 K 线 接口。最近 30 天内，每请求 1 只股票的历史 K 线，将会占用 1 个历史 K 线额度。最近 30 天内重复请求同一只股票的历史 K 线，不会重复累计。 同时，订阅同一股票的不同周期的K线只占用1个额度，不会重复累计。 举例：
假设您的历史 K 线额度是 100，今天是 2020 年 7 月 5 日。 您在 2020 年 6 月 5 日~2020 年 7 月 5 日之间，共计请求了 60 只股票的历史 K 线，则剩余的历史 K 线额度为 40。

> **提示**
>
> - 订阅额度和历史 K 线额度为系统自动分配，不需要手动申请。
> - 新入金的账户，额度等级会在 2 小时内自动生效。
> - 在途资产 不会用于额度计算。

交易功能

- 进行指定市场的交易时，需要先确认是否已开通该市场的交易业务账户。
  举例：您只能在美股交易业务账户下进行美股交易，无法在港股交易业务账户下进行美股交易。

费用

行情

中国内地 IP 个人客户，免费获取港股市场 LV2 行情及 A 股市场 LV1 行情。
部分品种行情，需要购买行情卡后方可获取。您可以在 行情权限 一节，进入具体的行情卡购买页面查看价格。

交易

通过 OpenAPI 进行交易，无附加收费，交易费用与通过 APP 交易的费用一致。具体收费方案如下表：

所属券商	收费方案
富途证券(香港)	收费方案
moomoo证券(美国)	收费方案
moomoo证券(新加坡)	收费方案
moomoo证券(澳大利亚)	收费方案
moomoo证券(马来西亚)	收费方案
moomoo证券(加拿大)	收费方案
moomoo证券(日本)	收费方案

可视化 OpenD

OpenD 提供可视化和命令行两种运行方式，这里介绍操作比较简单的可视化 OpenD。

如果想要了解命令行的方式请参考 命令行 OpenD 。

可视化 OpenD

第一步 下载

- 可视化 OpenD 支持 Windows、MacOS、CentOS、Ubuntu 四种系统。
- 您可以通过 moomoo 官网 下载。

第二步 安装运行

- 解压文件，找到对应的安装文件可一键安装运行。
- Windows 系统默认安装在 `%appdata%` 目录下。

第三步 配置

- 可视化 OpenD 启动配置在图形界面的右侧，如下图所示：

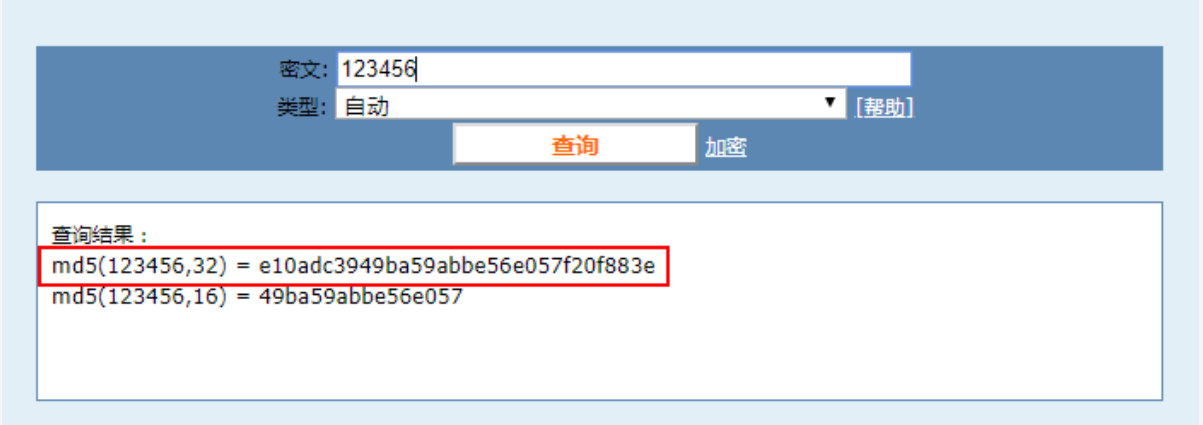
配置项列表:

配置项	说明
监听地址	API 协议监听地址
监听端口	API 协议监听端口
日志级别	OpenD 日志级别
语言	中英语言
期货交易 API 时区	期货交易 API 时区
API 推送频率	API 订阅数据推送频率控制
Telnet 地址	远程操作命令监听地址
Telnet 端口	远程操作命令监听端口
加密私钥路径	API 协议 RSA 加密私钥（PKCS#1）文件绝对路径
WebSocket 监听地址	WebSocket 服务监听地址

配置项	说明
WebSocket 端口	WebSocket 服务监听端口
WebSocket 证书	WebSocket 证书文件路径
WebSocket 私钥	WebSocket 证书私钥文件路径
WebSocket 鉴权密钥	密钥密文（32 位 MD5 加密 16 进制）

提示

- 可视化 OpenD，是通过启动命令行 OpenD 来提供服务，且通过 WebSocket 与命令行 OpenD 交互，所以必定启动 WebSocket 功能。
- 为保证您的证券业务账户安全，如果监听地址不是本地，您必须配置私钥才能使用交易接口。行情接口不受此限制。
- 当 WebSocket 监听地址不是本地，需配置 SSL 才可以启动，且证书私钥生成不可设置密码。
- 密文是明文经过 32 位 MD5 加密后用 16 进制表示的数据，搜索在线 MD5 加密（注意，通过第三方网站计算可能有记录撞库的风险）或下载 MD5 计算工具可计算得到。32 位 MD5 密文如下图红框区域（e10adc3949ba59abbe56e057f20f883e）：



- OpenD 默认读取同目录下的 OpenD.xml。在 MacOS 上，由于系统保护机制，OpenD.app 在运行时会被分配一个随机路径，导致无法找到原本的路径。此时有以下方法：
  - 执行 tar 包下的 fixrun.sh
  - 用命令行参数 `-cfg_file` 指定配置文件路径，见下面说明

> - 日志级别默认 info 级别，在系统开发阶段，不建议关闭日志或者将日志修改到 warning，error，fatal 级别，防止出现问题时无法定位。

第四步 登录

- 输入账号密码，点击登录。
  首次登录，您需要先完成问卷评估与协议确认，完成后重新登录即可。
  登录成功后，您可以看到自己的账号信息和 行情权限。

编程环境搭建

> **注意**
>
> 不同的编程语言，编程环境搭建的方法有所不同。

Python 环境

环境要求

- 操作系统要求：
  - Windows 7/10 的 32 或 64 位操作系统
  - Mac 10.11 及以上的 64 位操作系统
  - CentOS 7 及以上的 64 位操作系统
  - Ubuntu 16.04 以上的 64 位操作系统
- Python 版本要求：
  - Python 3.6 及以上

环境搭建

1. 安装 Python

为避免因环境问题导致的运行失败，我们推荐 Python 3.8 版本。

下载地址：Python 下载

▶ 提示

当安装成功后，执行如下命令来查看是否安装成功:
`python -V`（Windows） 或 `python3 -V`（Linux 和 Mac）

2. 安装 PyCharm（可选）

我们推荐您使用 PyCharm 作为 Python IDE（集成开发环境）。

3. 安装 TA-Lib（可选）

TA-Lib 用中文可以称作技术分析库，是一种广泛用在程序化交易中，进行金融市场数据的技术分析的函数库。它提供了多种技术分析的函数，方便我们量化投资中编程工作。

安装方法：在 cmd 中直接使用 pip 安装

```
$ pip install TA-Lib
```

> **提示**
>
> - 安装 TA-Lib 非必须，可先跳过该步骤

简易程序运行

Python 示例

第一步：下载安装登录 OpenD

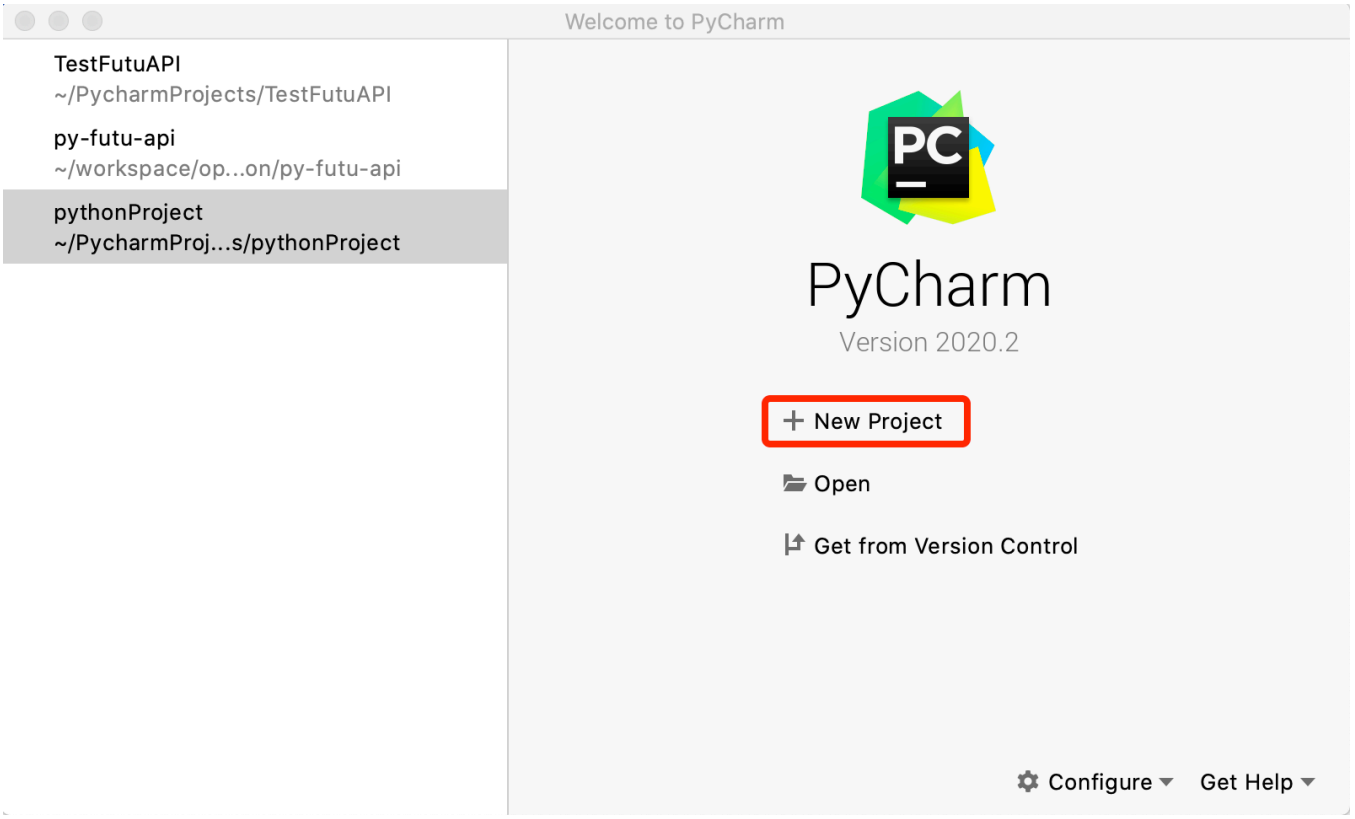
请参考 这里，完成 OpenD 的下载、安装和登录。

第二步：下载 Python API

- 方式一：在 cmd 中直接使用 pip 安装。
  - 初次安装：Windows 系统 `$ pip install moomoo-api`，Linux/Mac系统 `$ pip3 install moomoo-api`。
  - 二次升级：Windows 系统 `$ pip install moomoo-api --upgrade`，Linux/Mac系统 `$ pip3 install moomoo-api --upgrade`。
- 方式二：通过 moomoo 官网 下载最新版本的 Python API。

第三步：创建新项目

打开 PyCharm，在 Welcome to PyCharm 窗口中，点击 New Project。如果你已经创建了一个项目，可以选择打开该项目。



第四步：创建新文件

在该项目下，创建新 Python 文件，并把下面的示例代码拷贝到文件里。
示例代码功能包括查看行情快照、模拟交易下单。

```
from moomoo import *

quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)  # 创建行情对象
print(quote_ctx.get_market_snapshot('HK.00700'))  # 获取港股 HK.00700 的快照数据
quote_ctx.close() # 关闭对象，防止连接条数用尽


trd_ctx = OpenSecTradeContext(host='127.0.0.1', port=11111)  # 创建交易对象
print(trd_ctx.place_order(price=500.0, qty=100, code="HK.00700", trd_side=TrdSide

trd_ctx.close()  # 关闭对象，防止连接条数用尽
```

第五步：运行文件

右键点击运行，可以看到运行成功的返回信息如下：

```
2020-11-05 17:09:29,705 [open_context_base.py] _socket_reconnect_and_wait_ready:2
2020-11-05 17:09:29,705 [open_context_base.py] on_connected:344: Connected : conn
2020-11-05 17:09:29,706 [open_context_base.py] _handle_init_connect:445: InitConn
(0,        code          update_time  last_price  open_price  high_price  ...  af
0  HK.00700  2020-11-05 16:08:06       625.0       610.0       625.0  ...

[1 rows x 132 columns])
2020-11-05 17:09:29,739 [open_context_base.py] _socket_reconnect_and_wait_ready:2
2020-11-05 17:09:29,739 [network_manager.py] work:366: Close: conn_id=1
2020-11-05 17:09:29,739 [open_context_base.py] on_connected:344: Connected : conn
2020-11-05 17:09:29,740 [open_context_base.py] _handle_init_connect:445: InitConn
(0,        code stock_name trd_side order_type order_status  ... dealt_avg_price
0  HK.00700       腾讯控股      BUY     NORMAL   SUBMITTING  ...             0.0

[1 rows x 16 columns])
2020-11-05 17:09:32,843 [network_manager.py] work:366: Close: conn_id=2
(0,        code stock_name trd_side      order_type order_status  ... dealt_avg_p
0  HK.00700       腾讯控股      BUY  ABSOLUTE_LIMIT    SUBMITTED  ...

[1 rows x 16 columns])
```

交易策略搭建示例

> **提示**
>
> - 以下交易策略不构成投资建议，仅供学习参考。

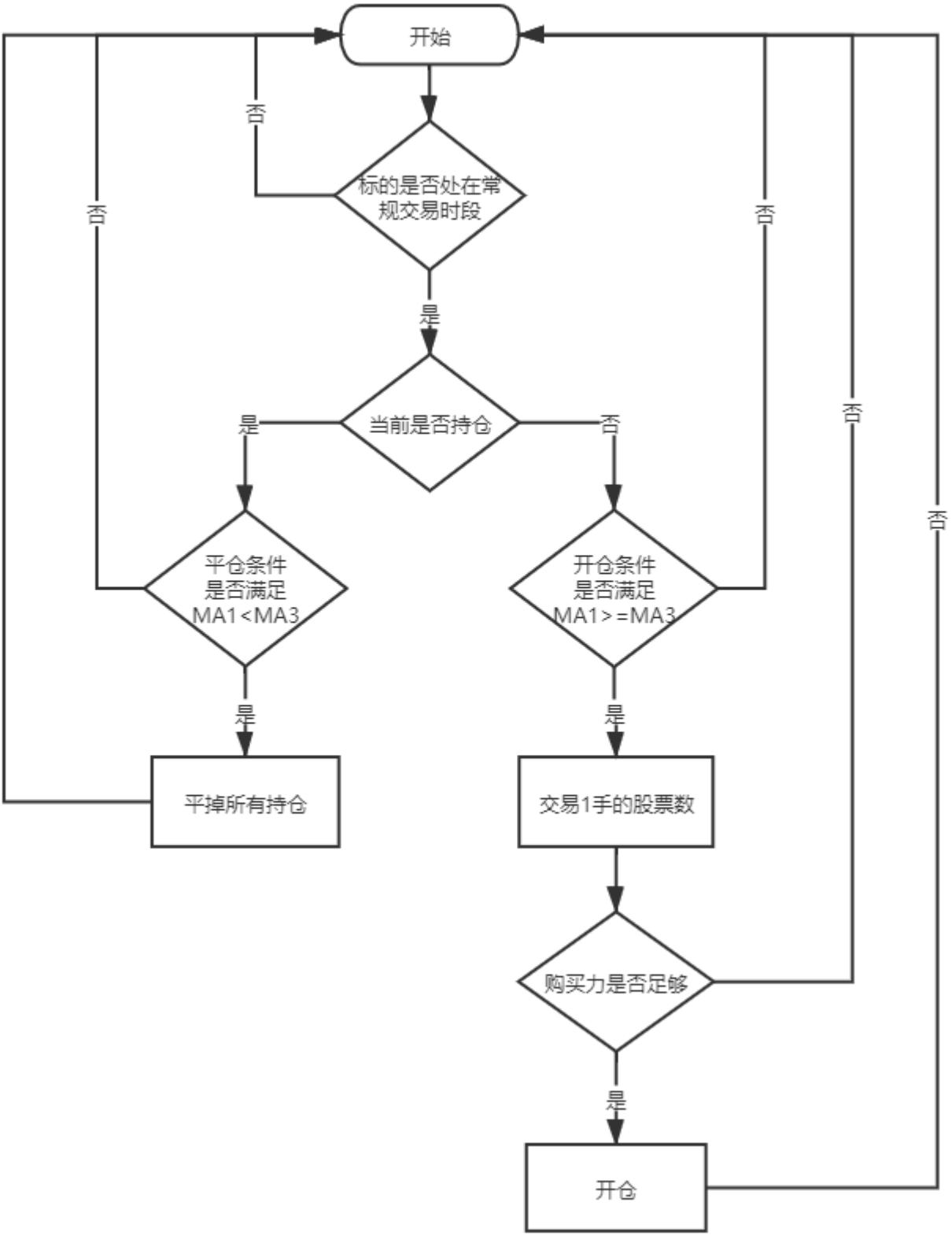
策略概述

构建一个双均线策略：

运用某一标的1分 K 线，计算出两条不同周期的移动平均线 MA1 和 MA3，跟踪 MA1 和 MA3 的相对大小，由此判断买卖时机。

当 MA1 >= MA3 时，判断该标的为强势状态，市场属于多头市场，采取开仓的操作；
当 MA1 < MA3 时，判断该标的为弱势状态，市场属于空头市场，采取平仓的操作。

流程图



代码示例

- Example

```
from moomoo import *

```

```python
############################## 全局变量设置 ##############################
MOOMOOOPEND_ADDRESS = '127.0.0.1'  # OpenD 监听地址
MOOMOOOPEND_PORT = 11111  # OpenD 监听端口

TRADING_ENVIRONMENT = TrdEnv.SIMULATE  # 交易环境：真实 / 模拟
TRADING_MARKET = TrdMarket.HK  # 交易市场权限，用于筛选对应交易市场权限的账户
TRADING_PWD = '123456'  # 交易密码，用于解锁交易
TRADING_PERIOD = KLType.K_1M  # 信号 K 线周期
TRADING_SECURITY = 'HK.00700'  # 交易标的
FAST_MOVING_AVERAGE = 1  # 均线快线的周期
SLOW_MOVING_AVERAGE = 3  # 均线慢线的周期

quote_context = OpenQuoteContext(host=MOOMOOOPEND_ADDRESS, port=MOOMOOOPEND_PORT)
trade_context = OpenSecTradeContext(filter_trdmarket=TRADING_MARKET, host=MOOMOOO


# 解锁交易
def unlock_trade():
    if TRADING_ENVIRONMENT == TrdEnv.REAL:
        ret, data = trade_context.unlock_trade(TRADING_PWD)
        if ret != RET_OK:
            print('解锁交易失败：', data)
            return False
        print('解锁交易成功！')
    return True


# 获取市场状态
def is_normal_trading_time(code):
    ret, data = quote_context.get_market_state([code])
    if ret != RET_OK:
        print('获取市场状态失败：', data)
        return False
    market_state = data['market_state'][0]
    '''
    MarketState.MORNING             港、A 股早盘
    MarketState.AFTERNOON           港、A 股下午盘，美股全天
    MarketState.FUTURE_DAY_OPEN     港、新、日期货日市开盘
    MarketState.FUTURE_OPEN         美期货开盘
    MarketState.FUTURE_BREAK_OVER   美期货休息后开盘
    MarketState.NIGHT_OPEN          港、新、日期货夜市开盘
    '''
    if market_state == MarketState.MORNING or \
                       market_state == MarketState.AFTERNOON or \
                       market_state == MarketState.FUTURE_DAY_OPEN  or \
```

```
                    market_state == MarketState.FUTURE_OPEN  or \
                    market_state == MarketState.FUTURE_BREAK_OVER  or \
                    market_state == MarketState.NIGHT_OPEN:
            return True
        print('现在不是持续交易时段。')
        return False


    # 获取持仓数量
    def get_holding_position(code):
        holding_position = 0
        ret, data = trade_context.position_list_query(code=code, trd_env=TRADING_ENV
        if ret != RET_OK:
            print('获取持仓数据失败：', data)
            return None
        else:
            for qty in data['qty'].values.tolist():
                holding_position += qty
            print('【持仓状态】 {} 的持仓数量为：{}'.format(TRADING_SECURITY, holding_
        return holding_position


    # 拉取 K 线，计算均线，判断多空
    def calculate_bull_bear(code, fast_param, slow_param):
        if fast_param <= 0 or slow_param <= 0:
            return 0
        if fast_param > slow_param:
            return calculate_bull_bear(code, slow_param, fast_param)
        ret, data = quote_context.get_cur_kline(code=code, num=slow_param + 1, ktype=
        if ret != RET_OK:
            print('获取K线失败：', data)
            return 0
        candlestick_list = data['close'].values.tolist()[::-1]
        fast_value = None
        slow_value = None
        if len(candlestick_list) > fast_param:
            fast_value = sum(candlestick_list[1: fast_param + 1]) / fast_param
        if len(candlestick_list) > slow_param:
            slow_value = sum(candlestick_list[1: slow_param + 1]) / slow_param
        if fast_value is None or slow_value is None:
            return 0
        return 1 if fast_value >= slow_value else -1


    # 获取一档摆盘的 ask1 和 bid1
```

```python
def get_ask_and_bid(code):
    ret, data = quote_context.get_order_book(code, num=1)
    if ret != RET_OK:
        print('获取摆盘数据失败：', data)
        return None, None
    return data['Ask'][0][0], data['Bid'][0][0]


# 开仓函数
def open_position(code):
    # 获取摆盘数据
    ask, bid = get_ask_and_bid(code)

    # 计算下单量
    open_quantity = calculate_quantity()

    # 判断购买力是否足够
    if is_valid_quantity(TRADING_SECURITY, open_quantity, ask):
        # 下单
        ret, data = trade_context.place_order(price=ask, qty=open_quantity, code=
                                              order_type=OrderType.NORMAL, trd_en
                                              remark='moving_average_strategy')
        if ret != RET_OK:
            print('开仓失败：', data)
    else:
        print('下单数量超出最大可买数量。')


# 平仓函数
def close_position(code, quantity):
    # 获取摆盘数据
    ask, bid = get_ask_and_bid(code)

    # 检查平仓数量
    if quantity == 0:
        print('无效的下单数量。')
        return False

    # 平仓
    ret, data = trade_context.place_order(price=bid, qty=quantity, code=code, trd
                    order_type=OrderType.NORMAL, trd_env=TRADING_ENVIRONMENT, rema
    if ret != RET_OK:
        print('平仓失败：', data)
        return False
    return True
```

```python

# 计算下单数量
def calculate_quantity():
    price_quantity = 0
    # 使用最小交易量
    ret, data = quote_context.get_market_snapshot([TRADING_SECURITY])
    if ret != RET_OK:
        print('获取快照失败：', data)
        return price_quantity
    price_quantity = data['lot_size'][0]
    return price_quantity


# 判断购买力是否足够
def is_valid_quantity(code, quantity, price):
    ret, data = trade_context.acctradinginfo_query(order_type=OrderType.NORMAL, c
                                                   trd_env=TRADING_ENVIRONMENT)
    if ret != RET_OK:
        print('获取最大可买可卖失败：', data)
        return False
    max_can_buy = data['max_cash_buy'][0]
    max_can_sell = data['max_sell_short'][0]
    if quantity > 0:
        return quantity < max_can_buy
    elif quantity < 0:
        return abs(quantity) < max_can_sell
    else:
        return False


# 展示订单回调
def show_order_status(data):
    order_status = data['order_status'][0]
    order_info = dict()
    order_info['代码'] = data['code'][0]
    order_info['价格'] = data['price'][0]
    order_info['方向'] = data['trd_side'][0]
    order_info['数量'] = data['qty'][0]
    print('【订单状态】', order_status, order_info)


############################ 填充以下函数来完成您的策略 ############################
# 策略启动时运行一次，用于初始化策略
def on_init():
```

```
    # 解锁交易（如果是模拟交易则不需要解锁）
    if not unlock_trade():
        return False
    print('************  策略开始运行 ***********')
    return True


# 每个 tick 运行一次，可将策略的主要逻辑写在此处
def on_tick():
    pass


# 每次产生一根新的 K 线运行一次，可将策略的主要逻辑写在此处
def on_bar_open():
    # 打印分隔线
    print('************************************')

    # 只在常规交易时段交易
    if not is_normal_trading_time(TRADING_SECURITY):
        return

    # 获取 K 线，计算均线，判断多空
    bull_or_bear = calculate_bull_bear(TRADING_SECURITY, FAST_MOVING_AVERAGE, SL

    # 获取持仓数量
    holding_position = get_holding_position(TRADING_SECURITY)

    # 下单判断
    if holding_position == 0:
        if bull_or_bear == 1:
            print('【操作信号】 做多信号，建立多单。')
            open_position(TRADING_SECURITY)
        else:
            print('【操作信号】 做空信号，不开空单。')
    elif holding_position > 0:
        if bull_or_bear == -1:
            print('【操作信号】 做空信号，平掉持仓。')
            close_position(TRADING_SECURITY, holding_position)
        else:
            print('【操作信号】 做多信号，无需加仓。')


# 委托成交有变化时运行一次
def on_fill(data):
    pass
```

```python

# 订单状态有变化时运行一次
def on_order_status(data):
    if data['code'][0] == TRADING_SECURITY:
        show_order_status(data)


################################## 框架实现部分，可忽略不看 ##########################
class OnTickClass(TickerHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        on_tick()


class OnBarClass(CurKlineHandlerBase):
    last_time = None
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(OnBarClass, self).on_recv_rsp(rsp_pb)
        if ret_code == RET_OK:
            cur_time = data['time_key'][0]
            if cur_time != self.last_time and data['k_type'][0] == TRADING_PERIOD
                if self.last_time is not None:
                    on_bar_open()
                self.last_time = cur_time


class OnOrderClass(TradeOrderHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret, data = super(OnOrderClass, self).on_recv_rsp(rsp_pb)
        if ret == RET_OK:
            on_order_status( data)


class OnFillClass(TradeDealHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret, data = super(OnFillClass, self).on_recv_rsp(rsp_pb)
        if ret == RET_OK:
            on_fill(data)


# 主函数
if __name__ == '__main__':
    # 初始化策略
    if not on_init():
        print('策略初始化失败，脚本退出！')
```

```
        quote_context.close()
        trade_context.close()
    else:
        # 设置回调
        quote_context.set_handler(OnTickClass())
        quote_context.set_handler(OnBarClass())
        trade_context.set_handler(OnOrderClass())
        trade_context.set_handler(OnFillClass())

        # 订阅标的合约的 逐笔，K 线和摆盘，以便获取数据
        quote_context.subscribe(code_list=[TRADING_SECURITY], subtype_list=[SubTy

```

- **Output**

```
**********  策略开始运行 **********
************************************
【持仓状态】 HK.00700 的持仓数量为：0
【操作信号】 做多信号，建立多单。
【订单状态】 SUBMITTING {'代码': 'HK.00700', '价格': 597.5, '方向': 'BUY', '数量':
【订单状态】 SUBMITTED {'代码': 'HK.00700', '价格': 597.5, '方向': 'BUY', '数量':
【订单状态】 FILLED_ALL {'代码': 'HK.00700', '价格': 597.5, '方向': 'BUY', '数量':
************************************
【持仓状态】 HK.00700 的持仓数量为：100.0
【操作信号】 做空信号，平掉持仓。
【订单状态】 SUBMITTING {'代码': 'HK.00700', '价格': 596.5, '方向': 'SELL', '数量'
【订单状态】 SUBMITTED {'代码': 'HK.00700', '价格': 596.5, '方向': 'SELL', '数量':
【订单状态】 FILLED_ALL {'代码': 'HK.00700', '价格': 596.5, '方向': 'SELL', '数量'
```

概述

- OpenD 是 moomoo API 的网关程序，运行于您的本地电脑或云端服务器，负责中转协议请求到富途服务器，并将处理后的数据返回。是运行 moomoo API 程序必要的前提。
- OpenD 支持 Windows、MacOS、CentOS、Ubuntu 四个平台。
- OpenD 集成了登录功能。运行时，需要使用 **平台账号**（moomoo 号）、**邮箱**、**手机号** 和 **登录密码** 进行登录。
- OpenD 登录成功后，会启动 Socket 服务以供 moomoo API 连接和通信。

运行方式

OpenD 目前提供两种安装运行方式，您可选择任一方式：

- 可视化 OpenD：提供界面化应用程序，操作便捷，尤其适合入门用户，安装和运行请参考 可视化 OpenD。
- 命令行 OpenD：提供命令行执行程序，需自行进行配置，适合对命令行熟悉或长时间在服务器上挂机的用户，安装和运行请参考 命令行 OpenD。

运行时操作

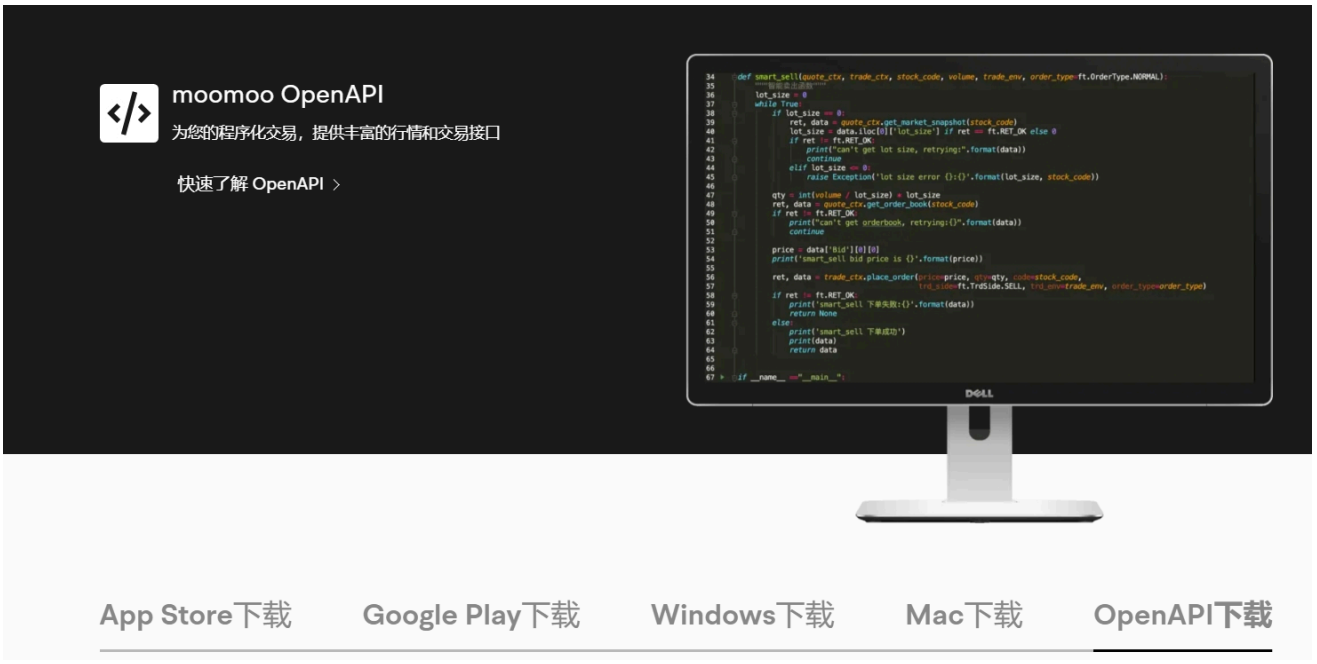
OpenD 在运行过程中，可以查看用户额度、行情权限、链接状态、延迟统计，以及操作关闭 API 连接、重登录、退出登录等运维操作。
具体方法可以查看下表：

方式	可视化 OpenD	命令行 OpenD
直接方式	界面查看或操作	命令行发送 运维命令
间接方式	通过 Telnet 发送 运维命令	通过 Telnet 发送 运维命令

命令行 OpenD

第一步 下载

- 命令行 OpenD 支持 Windows、MacOS、CentOS、Ubuntu 四种系统。
- 您可以通过 moomoo 官网 下载。



第二步 解压

- 解压上一步下载的文件，在文件夹中找到 OpenD 配置文件 OpenD.xml 和程序打包数据文件 Appdata.dat。
  - OpenD.xml 用于配置 OpenD 程序启动参数，若不存在则程序无法正常启动。
  - Appdata.dat 是程序需要用到的一些数据量较大的信息，打包数据减少启动下载该数据的耗时，若不存在则程序无法正常启动。
- 命令行 OpenD 支持用户自定义文件路径，详见 命令行启动参数。

第三步 参数配置

- 打开并编辑配置文件 OpenD.xml，如下图所示。普通使用仅需修改账号和登录密码，其他高阶选项可以根据下表的提示进行修改。

```xml
<moomoo_opend>
    <!-- 基础参数 -->
    <!-- Basic parameters -->
        <!-- 协议监听地址,不填默认127.0.0.1 -->
        <!-- Listening address. 127.0.0.1 by default -->
        <ip>127.0.0.1</ip>
        <!-- API接口协议监听端口 -->
        <!-- API interface protocol listening port -->
        <api_port>11111</api_port>
        <!-- 登录帐号 -->
        <!-- Login account -->
        <login_account>100000</login_account>
        <!-- 登录密码32位MD5加密16进制 -->
        <!-- Login password, 32-bit MD5 encrypted hexadecimal -->
        <!-- <login_pwd_md5>6e55f158a827b1a1c4321a245aaaad88</login_pwd_md5> -->
        <!-- 登录密码明文，密码密文存在情况下只使用密文 -->
        <!-- Plain text of login password. When cypher text exists, the cypher text will be used. -->
        <login_pwd>123456</login_pwd>
        <!-- FutuOpenD语言，en: 英文，chs: 简体中文 -->
        <!-- FutuOpenD language. en: English, chs: Simplified Chinese -->
        <lang>chs</lang>
    <!-- 进阶参数 -->
    <!-- Advanced parameters -->
        <!-- FutuOpenD日志等级，no，debug，info，warning，error，fatal -->
        <!-- FutuOpenD log level: no, debug, info, warning, error, fatal -->
        <log_level>info</log_level>
        <!-- API推送协议格式，0：pb，1：json -->
        <!-- API push protocol format. 0: pb, 1: json -->
        <push_proto_type>0</push_proto_type>
        <!-- API订阅数据推送频率控制，单位毫秒，目前不包括K线和分时，不设置则不限制频率-->
        <!-- Data Push Frequency, in milliseconds. Candlesticks and timeframes are not included. If not se
        <!-- <qot_push_frequency>1000</qot_push_frequency> -->
        <!-- Telnet监听地址,不填默认127.0.0.1 -->
        <!-- Telnet listening address. 127.0.0.1 by default -->
        <!-- <telnet_ip>127.0.0.1</telnet_ip> -->
        <!-- Telnet监听端口 -->
        <!-- Telnet listening port -->
        <!-- <telnet_port>22222</telnet_port> -->
        <!-- API协议加密私钥文件路径,不设置则不加密 -->
        <!-- File path for private key for API protocol enctyption. If not set, it will not be encrypted.
        <!-- <rsa_private_key>D:\rsa</rsa_private_key> -->
        <!-- 是否接收到价提醒推送，0：不接收，1：接收 -->
```

配置项列表：

配置项	说明
ip	监听地址
api_port	API 协议接收端口
login_account	登录帐号
login_pwd	登录密码明文
login_pwd_md5	登录密码密文（32 位 MD5 加密 16 进制）
lang	中英语言
log_level	OpenD 日志级别
push_proto_type	推送协议类型
qot_push_frequency	API 订阅数据推送频率控制

配置项	说明
telnet_ip	远程操作命令监听地址
telnet_port	远程操作命令监听端口
rsa_private_key	API 协议 RSA 加密私钥（PKCS#1）文件绝对路径
price_reminder_push	是否接收到价提醒推送
auto_hold_quote_right	被踢后是否自动抢权限
future_trade_api_time_zone	期货交易 API 时区
websocket_ip	WebSocket 服务监听地址
websocket_port	WebSocket 服务监听端口
websocket_key_md5	密钥密文（32 位 MD5 加密 16 进制）
websocket_private_key	WebSocket 证书私钥文件路径
websocket_cert	WebSocket 证书文件路径
pdt_protection	是否开启 防止被标记为日内交易者 的功能
dtcall_confirmation	是否开启 日内交易保证金追缴预警 的功能

提示

- 为保证您的证券业务账户安全，如果监听地址不是本地，您必须配置私钥才能使用交易接口。行情接口不受此限制。
- 当 WebSocket 监听地址不是本地，需配置 SSL 才可以启动，且证书私钥生成不可设置密码。
- 密文是明文经过 32 位 MD5 加密后用 16 进制表示的数据，搜索在线 MD5 加密（注意，通过第三方网站计算可能有记录撞库的风险）或下载 MD5 计算工具可计算得到。32 位 MD5 密文如下图红框区域（e10adc3949ba59abbe56e057f20f883e）：

- OpenD 默认读取同目录下的 OpenD.xml。在 MacOS 上，由于系统保护机制，OpenD.app 在运行时会被分配一个随机路径，导致无法找到原本的路径。此时有以下方法：
  - 执行 tar 包下的 fixrun.sh
  - 用命令行参数 `-cfg_file` 指定配置文件路径，见下面说明
- 日志级别默认 info 级别，在系统开发阶段，不建议关闭日志或者将日志修改到 warning，error，fatal 级别，防止出现问题时无法定位。

第四步 命令行启动

- 在命令行中切到前面解压文件夹 OpenD 文件所在的目录，使用如下命令启动，即可以 OpenD.xml 配置文件中的参数启动。
  - Windows：`OpenD`
  - Linux：`./OpenD`
  - MacOS：`./OpenD.app/Contents/MacOS/OpenD`

▶ 命令行启动参数

运维命令

通过命令行或者 Telnet 发送命令可以对 OpenD 做运维操作。

命令格式：`cmd -param_key1=param_value1 -param_key2=param_value2`

以 `help -cmd=exit` 为例，介绍Telnet的用法：

1. 在OpenD启动参数中，配置好 Telnet 地址和 Telnet 端口。

FutuOpenD.xml

```
<futu_opend>
    <!-- 基础参数 -->
    <!-- Basic parameters -->
        <!-- 协议监听地址,不填默认127.0.0.1 -->
        <!-- Listening address. 127.0.0.1 if not specified -->  // Listening address. 127.0.0.1 by default
        <ip>127.0.0.1</ip>
        <!-- API接口协议监听端口 -->
        <!-- API interface protocol listening port -->
        <api_port>11111</api_port>
        <!-- 登录帐号 -->
        <login_account>100000</login_account>
        <!-- 登录密码32位MD5加密16进制 -->
        <!-- <login_pwd_md5>6e55f158a827b1a1c4321a245aaaad88</login_pwd_md5> -->
        <!-- 登录密码明文，密码密文存在情况下只使用密文 -->
        <login_pwd>123456</login_pwd>
        <!-- FutuOpenD语言，en：英文，chs：简体中文 -->
        <lang>chs</lang>
    <!-- 进阶参数 -->
    <!-- Advanced parameters -->
        <!-- FutuOpenD日志等级，no,debug,info,warning,error,fatal -->
        <log_level>info</log_level>
        <!-- API推送协议格式，0:pb，1:json -->
        <!-- API push protocol format, 0:pb, 1:json -->
        <push_proto_type>0</push_proto_type>
        <!-- API订阅数据推送频率控制，单位毫秒，目前不包括K线和分时，不设置则不限制频率-->
        <!-- API subscription data push frequency control, in milliseconds, currently does not include K-line and time-sharing, if not
        <!-- <qot_push_frequency>1000</qot_push_frequency> -->
        <!-- Telnet监听地址,不填默认127.0.0.1 -->
        <!-- Telnet listening address, default 127.0.0.1 if not filled in -->  // Telnet listening address, 127.0.0.1 by default
        <telnet_ip>127.0.0.1</telnet_ip>
        <!-- Telnet监听端口 -->
        <!-- Telnet listening port -->
        <telnet_port>22222</telnet_port>
        <!-- API协议加密私钥文件路径,不设置则不加密 -->
        <!-- API protocol encrypted private key file path, if not set, it will not be encrypted -->  // File path for private key for
        <!-- <rsa_private_key>D:\rsa</rsa_private_key> -->
        <!-- 是否接收到价提醒推送，0：不接收，1：接收 -->
        <!-- Whether to receive the price reminder push, 0: not receive, 1: receive -->
```

2. 启动 OpenD（会同时启动 Telnet）。
3. 通过 Telnet，向 OpenD 发送 `help -cmd=exit` 命令。

```py
from telnetlib import Telnet
with Telnet('127.0.0.1', 22222) as tn:  # Telnet 地址为：127.0.0.1，Telnet 端口为
    tn.write(b'help -cmd=exit\r\n')
    reply = b''
    while True:
        msg = tn.read_until(b'\r\n', timeout=0.5)
        reply += msg
        if msg == b'':
            break
    print(reply.decode('gb2312'))
```

命令帮助

`help -cmd=exit`

查看指定命令详细信息，不指定参数则输出命令列表

- 参数:
  - cmd: 命令

退出程序

`exit`

退出 OpenD 程序

请求手机验证码

`req_phone_verify_code`

请求手机验证码，当启用设备锁并初次在该设备登录，要求做安全验证。

- 频率限制:
  - 每60秒内最多请求1次

输入手机验证码

`input_phone_verify_code -code=123456`

输入手机验证码，并继续登录流程。

- 参数:
  - code: 手机验证码
- 频率限制:
  - 每60秒内最多请求10次

请求图形验证码

`req_pic_verify_code`

请求图形验证码，当多次输入错登录密码时，需要输入图形验证码。

- 频率限制:

- 每60秒内最多请求10次

输入图形验证码

`input_pic_verify_code -code=1234`

输入图形验证码，并继续登录流程。

- 参数:
  - code: 图形验证码
- 频率限制:
  - 每60秒内最多请求10次

重登录

`relogin -login_pwd=123456`

当登录密码修改或中途打开设备锁等情况，要求用户重新登录时，可以使用该命令。只能重登当前帐号，不支持切换帐号。 密码参数主要用于登录密码修改的情况，不指定密码则使用启动时登录密码。

- 参数:
  - login_pwd: 登录密码明文
  - login_pwd_md5: 登录密码密文（32 位 MD5 加密 16 进制）
- 频率限制:
  - 每小时最多请求10次

检测与连接点之间的时延

`ping`

检测与连接点之前的时延

- 频率限制:

- 每60秒内最多请求10次

展示延迟统计报告

```
show_delay_report -detail_report_path=D:/detail.txt -push_count_type=sr2cs
```

展示延迟统计报告，包括推送延迟，请求延迟以及下单延迟。每日北京时间 6:00 清理数据。

- 参数:
  - detail_report_path: 文件输出路径（MAC 系统仅支持绝对路径，不支持相对路径），可选参数，若不指定则输出到控制台
  - Paramters: push_count_type: 推送延迟的类型(sr2ss，ss2cr，cr2cs，ss2cs，sr2cs)，默认sr2cs。
    - sr 指服务器接收时间(目前只有港股支持该时间)
    - ss 指服务器发出时间
    - cr 指 OpenD 接收时间
    - cs 指 OpenD 发出时间

关闭 API 连接

```
close_api_conn -conn_id=123456
```

关闭某条 API 连接，若不指定则关闭所有有

- 参数:
  - conn_id: API 连接 ID

展示订阅状态

```
show_sub_info -conn_id=123456 -sub_info_path=D:/detail.txt
```

展示某条连接的订阅状态，若不指定则展示所有有

- 参数:
  - conn_id: API 连接 ID

- sub_info_path: 文件输出路径（MAC 系统仅支持绝对路径，不支持相对路径），可选参数，若不指定则输出到控制台

请求最高行情权限

`request_highest_quote_right`

当高级行情权限被其他设备（如：桌面端/手机端）占用时，可使用该命令重新请求最高行情权限（届时，其他处于登录状态的设备将无法使用高级行情）。

- 频率限制:
  - 每60秒内最多请求10次

升级

`update`

运行该命令，可以一键更新 OpenD

行情接口总览

模块		接口名	功能简介
实时行情	订阅	subscribe	订阅实时数据，指定股票代码和订阅的数据类型即可
		unsubscribe	取消订阅
		unsubscribe_all	取消所有订阅
		query_subscription	查询订阅信息
	推送回调	StockQuoteHandlerBase	报价推送
		OrderBookHandlerBase	摆盘推送
		CurKlineHandlerBase	K 线推送
		TickerHandlerBase	逐笔推送
		RTDataHandlerBase	分时推送
		BrokerHandlerBase	经纪队列推送
	拉取	get_market_snapshot	获取市场快照
		get_stock_quote	获取订阅股票报价的实时数据，有订阅要求限制
		get_order_book	获取实时摆盘数据
		get_cur_kline	实时获取指定股票最近 num 个 K 线数据
		get_rt_data	获取指定股票的分时数据
		get_rt_ticker	获取指定股票的实时逐笔。取最近 num 个逐笔
		get_broker_queue	获取股票的经纪队列
基本数据		get_market_state	获取股票对应市场的市场状态

	get_capital_flow	获取个股资金流向
	get_capital_distribution	获取个股资金分布
	get_owner_plate	获取单支或多支股票的所属板块信息列表
	request_history_kline	获取 K 线，不需要事先下载 K 线数据
	get_rehab	获取给定股票的复权因子
相关衍生品	get_option_expiration_date	通过标的股票，查询期权链的所有到期日
	get_option_chain	通过标的股查询期权
	get_warrant	拉取窝轮和相关衍生品数据接口
	get_referencestock_list	获取证券的关联数据
	get_future_info	获取期货合约资料
全市场筛选	get_stock_filter	获取条件选股
	get_plate_stock	获取特定板块下的股票列表
	get_plate_list	获取板块集合下的子板块列表
	get_stock_basicinfo	获取指定市场中特定类型或特定股票的基本信息
	get_ipo_list	获取指定市场的 ipo 列表
	get_global_state	获取全局市场状态
	request_trading_days	获取交易日历
个性化	get_history_kl_quota	获取已使用过的额度，即当前周期内已经下载过多少只股票
	set_price_reminder	设置到价提醒
	get_price_reminder	获取对某只股票(某个市场)设置的到价提醒列表
	get_user_security_group	获取自选股分组列表

	get_user_security	获取指定分组的自选股列表
	modify_user_security	修改指定分组的自选股列表
	PriceReminderHandlerBase	到价提醒推送

行情对象

创建连接

`OpenQuoteContext(host='127.0.0.1', port=11111, is_encrypt=None)`

- **介绍**

  创建并初始化行情连接

- **参数**

参数	类型	说明
host	str	OpenD 监听的 IP 地址
port	int	OpenD 监听的端口
is_encrypt	bool	是否启用加密 

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111, is_encrypt=False)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

关闭连接

`close()`

- **介绍**

  关闭行情接口类对象。默认情况下，moomoo API 内部创建的线程会阻止进程退出，只有当所有 Context 都 close 后，进程才能正常退出。但通过 set_all_thread_daemon 可以设置所有内部线程为 daemon 线程，这时即使没有调用 Context 的 close，进程也可以正常退出。

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

启动

`start()`

- **介绍**

  启动异步接收推送数据

停止

`stop()`

- **介绍**

  停止异步接收推送数据

订阅反订阅

订阅

`subscribe(code_list, subtype_list, is_first_push=True, subscribe_push=True, is_detailed_orderbook=False, extended_time=False, session=Session.NONE)`

- **介绍**

  订阅注册需要的实时信息，指定股票和订阅的数据类型即可。

- **参数**

参数	类型	说明
code_list	list	需要订阅的股票代码列表
subtype_list	list	需要订阅的数据类型列表
is_first_push	bool	订阅成功之后是否立即推送一次缓存数据
subscribe_push	bool	订阅后是否推送
is_detailed_orderbook	bool	是否订阅详细的摆盘订单明细
extended_time	bool	是否允许美股盘前盘后数据
session	Session	美股订阅时段

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
err_message	NoneType	当 ret == RET_OK 时，返回 None

	str	当 ret != RET_OK 时，返回错误描述

- Example

```
import time
from moomoo import *
class OrderBookTest(OrderBookHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(OrderBookTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("OrderBookTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("OrderBookTest ", data) # OrderBookTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = OrderBookTest()
quote_ctx.set_handler(handler)  # 设置实时摆盘回调
quote_ctx.subscribe(['US.AAPL'], [SubType.ORDER_BOOK])  # 订阅买卖摆盘类型，OpenD
time.sleep(15)  #  设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
OrderBookTest  {'code': 'US.AAPL', 'name': '苹果', 'svr_recv_time_bid': '2025-04-
```

取消订阅

`unsubscribe(code_list, subtype_list, unsubscribe_all=False)`

- **介绍**

  取消订阅

- **参数**

参数	类型	说明
code_list	list	取消订阅的股票代码列表

参数	类型	说明
subtype_list	list	需要订阅的数据类型列表
unsubscribe_all	bool	取消所有订阅

- Return

参数	类型	说明
ret	RET_CODE	接口调用结果
err_message	NoneType	当 ret == RET_OK, 返回 None
	str	当 ret != RET_OK, 返回错误描述

- Example

```
from moomoo import *
import time
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

print('current subscription status :', quote_ctx.query_subscription())  # 查询初始
ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.QUOTE, SubType.T
# 先订阅了AAPL全时段 QUOTE 和 TICKER 两个类型。订阅成功后 OpenD 将持续收到服务器的推
if ret_sub == RET_OK:   # 订阅成功
    print('subscribe successfully! current subscription status :', quote_ctx.quer
    time.sleep(60)  # 订阅之后至少1分钟才能取消订阅
    ret_unsub, err_message_unsub = quote_ctx.unsubscribe(['US.AAPL'], [SubType.QU
    if ret_unsub == RET_OK:
        print('unsubscribe successfully! current subscription status:', quote_ctx
    else:
        print('unsubscription failed! ', err_message_unsub)
else:
    print('subscription failed', err_message)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
current subscription status : (0, {'total_used': 0, 'remain': 1000, 'own_used': 0
subscribe successfully! current subscription status : (0, {'total_used': 2, 'rema
unsubscribe successfully! current subscription status: (0, {'total_used': 1, 'rem
```

取消所有订阅

`unsubscribe_all()`

- **介绍**

取消所有订阅

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
err_message	NoneType	当 ret == RET_OK, 返回 None
	str	当 ret != RET_OK, 返回错误描述

- Example

```python
from moomoo import *
import time
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

print('current subscription status :', quote_ctx.query_subscription())  # 查询初始
ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.QUOTE, SubType.T
# 先订阅了AAPL全时段 QUOTE 和 TICKER 两个类型。订阅成功后 OpenD 将持续收到服务器的推
if ret_sub == RET_OK:  # 订阅成功
    print('subscribe successfully! current subscription status :', quote_ctx.quer
    time.sleep(60)  # 订阅之后至少1分钟才能取消订阅
    ret_unsub, err_message_unsub = quote_ctx.unsubscribe_all()  # 取消所有订阅
    if ret_unsub == RET_OK:
        print('unsubscribe all successfully! current subscription status:', quote
    else:
        print('Failed to cancel all subscriptions! ', err_message_unsub)
else:
```

```
        print('subscription failed', err_message)
    quote_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
current subscription status : (0, {'total_used': 0, 'remain': 1000, 'own_used': 0
subscribe successfully! current subscription status : (0, {'total_used': 2, 'rema
unsubscribe all successfully! current subscription status: (0, {'total_used': 0,
```

接口限制

- 支持多种实时数据类型的订阅，参见 SubType ，每支股票订阅一个类型占用一个额度。
- 订阅额度规则请参见 订阅额度 & 历史 K 线额度。
- 至少订阅一分钟才可以反订阅。
- 由于港股 SF 行情摆盘数据量较大，为保证 SF 行情的速度和 OpenD 的处理性能，目前 SF 权限用户仅限同时订阅 50 只证券类产品（含 hkex 的正股、窝轮、牛熊）的摆盘、经纪队列，剩余订阅额度仍可用于订阅其他类型，如：逐笔，买卖经纪等。
- 港股期权期货在 LV1 权限下，不支持订阅逐笔类型。

获取订阅状态

`query_subscription(is_all_conn=True)`

- **介绍**

  获取订阅信息

- **参数**

参数	类型	说明
is_all_conn	bool	是否返回所有连接的订阅状态

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	dict	当 ret == RET_OK，返回订阅信息数据
	str	当 ret != RET_OK，返回错误描述

  - 订阅信息数据字典格式如下：

```
{
    'total_used': 4,     # 所有连接已使用的订阅额度
    'own_used': 0,       # 当前连接已使用的订阅额度
    'remain': 496,       #  剩余的订阅额度
    'sub_list':          #  每种订阅类型对应的股票列表
    {
        '订阅的类型': 该订阅类型下所有已订阅股票列表,
        …
    }
}
```

- Example

```python
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

quote_ctx.subscribe(['HK.00700'], [SubType.QUOTE])
ret, data = quote_ctx.query_subscription()
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
{'total_used': 1, 'remain': 999, 'own_used': 1, 'sub_list': {'QUOTE': ['HK.00700
```

实时报价回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时报价回调，异步处理已订阅股票的实时报价推送。
  在收到实时报价数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateBasicQot_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回报价数据
	str	当 ret != RET_OK，返回错误描述

  - 报价数据格式如下：

字段	类型	说明
code	str	股票代码
data_date	str	日期
data_time	str	当前价更新时间
last_price	float	最新价格
open_price	float	今日开盘价

字段	类型	说明
high_price	float	最高价格
low_price	float	最低价格
prev_close_price	float	昨收盘价格
volume	int	成交数量
turnover	float	成交金额
turnover_rate	float	换手率
amplitude	int	振幅
suspension	bool	是否停牌
listing_date	str	上市日期
price_spread	float	当前向上的价差
dark_status	DarkStatus	暗盘交易状态
sec_status	SecurityStatus	股票状态
strike_price	float	行权价
contract_size	float	每份合约数
open_interest	int	未平仓合约数
implied_volatility	float	隐含波动率
premium	float	溢价
delta	float	希腊值 Delta
gamma	float	希腊值 Gamma
vega	float	希腊值 Vega

字段	类型	说明
theta	float	希腊值 Theta
rho	float	希腊值 Rho
index_option_type	IndexOptionType	指数期权类型
net_open_interest	int	净未平仓合约数
expiry_date_distance	int	距离到期日天数
contract_nominal_value	float	合约名义金额
owner_lot_multiplier	float	相等正股手数
option_area_type	OptionAreaType	期权类型（按行权时间）
contract_multiplier	float	合约乘数
pre_price	float	盘前价格
pre_high_price	float	盘前最高价
pre_low_price	float	盘前最低价
pre_volume	int	盘前成交量
pre_turnover	float	盘前成交额
pre_change_val	float	盘前涨跌额
pre_change_rate	float	盘前涨跌幅
pre_amplitude	float	盘前振幅
after_price	float	盘后价格
after_high_price	float	盘后最高价
after_low_price	float	盘后最低价

字段	类型	说明
after_volume	int	盘后成交量
after_turnover	float	盘后成交额
after_change_val	float	盘后涨跌额
after_change_rate	float	盘后涨跌幅
after_amplitude	float	盘后振幅
overnight_price	float	夜盘价格
overnight_high_price	float	夜盘最高价
overnight_low_price	float	夜盘最低价
overnight_volume	int	夜盘成交量
overnight_turnover	float	夜盘成交额
overnight_change_val	float	夜盘涨跌额
overnight_change_rate	float	夜盘涨跌幅
overnight_amplitude	float	夜盘振幅
last_settle_price	float	昨结
position	float	持仓量
position_change	float	日增仓

- **Example**

```
import time
from moomoo import *

class StockQuoteTest(StockQuoteHandlerBase):
    def on_recv_rsp(self, rsp_pb):
```

```
        ret_code, data = super(StockQuoteTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("StockQuoteTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("StockQuoteTest ", data) # StockQuoteTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = StockQuoteTest()
quote_ctx.set_handler(handler)  # 设置实时报价回调
ret, data = quote_ctx.subscribe(['US.AAPL'], [SubType.QUOTE])  # 订阅实时报价类型
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
time.sleep(15)  #  设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
StockQuoteTest        code name data_date data_time  last_price  open_price  high
0  US.AAPL    苹果                            0.0         0.0         0.0
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时报价 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

实时摆盘回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时摆盘回调，异步处理已订阅股票的实时摆盘推送。 在收到实时摆盘数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateOrderBook_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	dict	当 ret == RET_OK，返回摆盘数据
	str	当 ret != RET_OK，返回错误描述

  - 摆盘数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
svr_recv_time_bid	str	moomoo 服务器从交易所收到买盘数据的时间
svr_recv_time_ask	str	moomoo 服务器从交易所收到卖盘数据的时间

字段	类型	说明
Bid	list	每个元祖包含如下信息：委托价格，委托数量，委托订单数，委托订单明细
Ask	list	每个元祖包含如下信息：委托价格，委托数量，委托订单数，委托订单明细

其中，Bid 和 Ask 字段的结构如下：

```
'Bid': [ (bid_price1, bid_volume1, order_num, {'orderid1': order_volume1, 'orderid
'Ask': [ (ask_price1, ask_volume1, order_num, {'orderid1': order_volume1, 'orderid
```

- Example

```python
import time
from moomoo import *
class OrderBookTest(OrderBookHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(OrderBookTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("OrderBookTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("OrderBookTest ", data) # OrderBookTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = OrderBookTest()
quote_ctx.set_handler(handler)  # 设置实时摆盘回调
ret, data = quote_ctx.subscribe(['US.AAPL'], [SubType.ORDER_BOOK])  # 订阅买卖摆盘
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
time.sleep(15)  #  设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
1    OrderBookTest  {'code': 'US.AAPL', 'name': '苹果', 'svr_recv_time_bid': '', 'svr_
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时摆盘 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- 美股市场实时摆盘回调，会持续推送当前交易时段的实时摆盘，无需设置时段。

实时 K 线回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时 K 线回调，异步处理已订阅股票的实时 K 线推送。

  在收到实时 K 线数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateKL_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回 K 线数据数据
	str	当 ret != RET_OK，返回错误描述

  - K 线数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
time_key	str	时间
open	float	开盘价

字段	类型	说明
close	float	收盘价
high	float	最高价
low	float	最低价
volume	int	成交量
turnover	float	成交额
pe_ratio	float	市盈率
turnover_rate	float	换手率
last_close	float	昨收价
k_type	KLType	K 线类型

- Example

```
import time
from moomoo import *
class CurKlineTest(CurKlineHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(CurKlineTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("CurKlineTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("CurKlineTest ", data) # CurKlineTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = CurKlineTest()
quote_ctx.set_handler(handler)  # 设置实时K线回调
ret, data = quote_ctx.subscribe(['US.AAPL'], [SubType.K_1M], session=Session.ALL)
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
```

```
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
CurKlineTest        code name             time_key    open    close    high    low
0  US.AAPL   苹果  2025-04-07 05:15:00  180.39  180.26  180.46  180.2    1322  23
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时K 线 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- **期权**，仅提供日K, 1分K，5分K，15分K，60分K。

实时分时回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时分时回调，异步处理已订阅股票的实时分时推送。
  在收到实时分时数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateRT_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回分时数据
	str	当 ret != RET_OK，返回错误描述

  - 分时数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
time	str	时间
is_blank	bool	数据状态
opened_mins	int	零点到当前多少分钟

字段	类型	说明
cur_price	float	当前价格
last_close	float	昨天收盘的价格
avg_price	float	平均价格 
volume	float	成交量
turnover	float	成交金额

- Example

```
import time
from moomoo import *

class RTDataTest(RTDataHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(RTDataTest, self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("RTDataTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("RTDataTest ", data) # RTDataTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = RTDataTest()
quote_ctx.set_handler(handler)  # 设置实时分时推送回调
ret, data = quote_ctx.subscribe(['US.AAPL'], [SubType.RT_DATA], session=Session.A
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
RTDataTest        code name                time  is_blank  opened_mins  cur_pric
0  US.AAPL   苹果  2025-04-07 05:24:00     False          324     179.53      188
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时分时 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

实时逐笔回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时逐笔回调，异步处理已订阅股票的实时逐笔推送。
  在收到实时逐笔数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateTicker_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回逐笔数据
	str	当 ret != RET_OK，返回错误描述

  - 逐笔数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
sequence	int	逐笔序号
time	str	成交时间
price	float	成交价格

字段	类型	说明
volume	int	成交数量
turnover	float	成交金额
ticker_direction	TickerDirect	逐笔方向
type	TickerType	逐笔类型
push_data_type	PushDataType	数据来源

- Example

```
import time
from moomoo import *

class TickerTest(TickerHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, data = super(TickerTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("TickerTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("TickerTest ", data) # TickerTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = TickerTest()
quote_ctx.set_handler(handler)  # 设置实时逐笔推送回调
ret, data = quote_ctx.subscribe(['US.AAPL'], [SubType.TICKER], session=Session.AL
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
TickerTest        code name                    time   price  volume  turnover ti
0  US.AAPL   苹果  2025-04-07 05:25:44.116  179.81       9   1618.29          NEU
```

```
3
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时逐笔 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- 行情连接断开重连后，OpenD 拉取断开期间，距离当前最近的（最多 50 根）逐笔数据并推送，可通过逐笔推送类型字段区分

实时经纪队列回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  实时经纪队列回调，异步处理已订阅股票的实时经纪队列推送。
  在收到实时经纪队列数据推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdateBroker_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	tuple	当 ret == RET_OK，返回经纪队列数据
	str	当 ret != RET_OK，返回错误描述

  - 经纪队列元组内容如下：

字段	类型	说明
stock_code	str	股票
bid_frame_table	pd.DataFrame	买盘数据
ask_frame_table	pd.DataFrame	卖盘数据

    - bid_frame_table 格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
bid_broker_id	int	经纪买盘 ID
bid_broker_name	str	经纪买盘名称
bid_broker_pos	int	经纪档位
order_id	int	交易所订单 ID
order_volume	int	单笔委托数量

- ask_frame_table 格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
ask_broker_id	int	经纪卖盘 ID
ask_broker_name	str	经纪卖盘名称
ask_broker_pos	int	经纪档位
order_id	int	交易所订单 ID
order_volume	int	单笔委托数量

- **Example**

```
import time
from moomoo import *

class BrokerTest(BrokerHandlerBase):
    def on_recv_rsp(self, rsp_pb):
```

```
        ret_code, err_or_stock_code, data = super(BrokerTest, self).on_recv_rsp(r
        if ret_code != RET_OK:
            print("BrokerTest: error, msg: {}".format(err_or_stock_code))
            return RET_ERROR, data
        print("BrokerTest: stock: {} data: {} ".format(err_or_stock_code, data))
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = BrokerTest()
quote_ctx.set_handler(handler)  # 设置实时经纪推送回调
ret, data = quote_ctx.subscribe(['HK.00700'], [SubType.BROKER]) # 订阅经纪类型，O
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
BrokerTest: stock: HK.00700 data: [        code   name  bid_broker_id bid_broker_r
0   HK.00700  腾讯控股          5338          J.P.摩根               1      N/A
..       ...   ...            ...             ...             ...      ...
36  HK.00700  腾讯控股          8305  富途证券国际(香港)有限公司               4

[37 rows x 7 columns],         code   name  ask_broker_id ask_broker_name  ask_bro
0   HK.00700  腾讯控股          1179  华泰金融控股(香港)有限公司               1
..       ...   ...            ...             ...             ...      ...
39  HK.00700  腾讯控股          6996       中国投资信息有限公司               1

[40 rows x 7 columns]]
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取实时经纪队列 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- 港股 LV1 权限下，不支持获取经纪队列数据

获取快照

`get_market_snapshot(code_list)`

- **介绍**

  获取快照数据

- **参数**

参数	类型	说明
code_list	list	股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回股票快照数据
	str	当 ret != RET_OK，返回错误描述

  - 股票快照数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
update_time	str	当前价更新时间
last_price	float	最新价格
open_price	float	今日开盘价

字段	类型	说明
high_price	float	最高价格
low_price	float	最低价格
prev_close_price	float	昨收盘价格
volume	int	成交数量
turnover	float	成交金额
turnover_rate	float	换手率
suspension	bool	是否停牌
listing_date	str	上市日期
equity_valid	bool	是否正股
issued_shares	int	总股本
total_market_val	float	总市值
net_asset	int	资产净值
net_profit	int	净利润
earning_per_share	float	每股盈利
outstanding_shares	int	流通股本
net_asset_per_share	float	每股净资产
circular_market_val	float	流通市值
ey_ratio	float	收益率
pe_ratio	float	市盈率
pb_ratio	float	市净率

字段	类型	说明
pe_ttm_ratio	float	市盈率 TTM
dividend_ttm	float	股息 TTM，派息
dividend_ratio_ttm	float	股息率 TTM
dividend_lfy	float	股息 LFY，上一年度派息
dividend_lfy_ratio	float	股息率 LFY
stock_owner	str	窝轮所属正股的代码或期权的标的股代码
wrt_valid	bool	是否是窝轮
wrt_conversion_ratio	float	换股比率
wrt_type	WrtType	窝轮类型
wrt_strike_price	float	行使价格
wrt_maturity_date	str	格式化窝轮到期时间
wrt_end_trade	str	格式化窝轮最后交易时间
wrt_leverage	float	杠杆比率
wrt_ipop	float	价内/价外
wrt_break_even_point	float	打和点
wrt_conversion_price	float	换股价
wrt_price_recovery_ratio	float	正股距收回价
wrt_score	float	窝轮综合评分
wrt_code	str	窝轮对应的正股（此字段已废除，修改为

字段	类型	说明
		stock_owner)
wrt_recovery_price	float	窝轮收回价
wrt_street_vol	float	窝轮街货量
wrt_issue_vol	float	窝轮发行量
wrt_street_ratio	float	窝轮街货占比
wrt_delta	float	窝轮对冲值
wrt_implied_volatility	float	窝轮引伸波幅
wrt_premium	float	窝轮溢价
wrt_upper_strike_price	float	上限价
wrt_lower_strike_price	float	下限价
wrt_inline_price_status	PriceType	界内界外
wrt_issuer_code	str	发行人代码
option_valid	bool	是否是期权
option_type	OptionType	期权类型
strike_time	str	期权行权日
option_strike_price	float	行权价
option_contract_size	float	每份合约数
option_open_interest	int	总未平仓合约数
option_implied_volatility	float	隐含波动率
option_premium	float	溢价

字段	类型	说明
option_delta	float	希腊值 Delta
option_gamma	float	希腊值 Gamma
option_vega	float	希腊值 Vega
option_theta	float	希腊值 Theta
option_rho	float	希腊值 Rho
index_option_type	IndexOptionType	指数期权类型
option_net_open_interest	int	净未平仓合约数
option_expiry_date_distance	int	距离到期日天数
option_contract_nominal_value	float	合约名义金额
option_owner_lot_multiplier	float	相等正股手数
option_area_type	OptionAreaType	期权类型（按行权时间）
option_contract_multiplier	float	合约乘数
plate_valid	bool	是否为板块类型
plate_raise_count	int	板块类型上涨支数
plate_fall_count	int	板块类型下跌支数
plate_equal_count	int	板块类型平盘支数
index_valid	bool	是否有指数类型
index_raise_count	int	指数类型上涨支数
index_fall_count	int	指数类型下跌支数
index_equal_count	int	指数类型平盘支数

字段	类型	说明
lot_size	int	每手股数，股票期权表示每份合约的股数，期货表示合约乘数
price_spread	float	当前向上的摆盘价差
ask_price	float	卖价
bid_price	float	买价
ask_vol	float	卖量
bid_vol	float	买量
enable_margin	bool	是否可融资（已废弃）
mortgage_ratio	float	股票抵押率（已废弃）
long_margin_initial_ratio	float	融资初始保证金率（已废弃）
enable_short_sell	bool	是否可卖空（已废弃）
short_sell_rate	float	卖空参考利率（已废弃）
short_available_volume	int	剩余可卖空数量（已废弃）
short_margin_initial_ratio	float	卖空（融券）初始保证金率（已废弃）
sec_status	SecurityStatus	股票状态
amplitude	float	振幅
avg_price	float	平均价

字段	类型	说明
bid_ask_ratio	float	委比
volume_ratio	float	量比
highest52weeks_price	float	52 周最高价
lowest52weeks_price	float	52 周最低价
highest_history_price	float	历史最高价
lowest_history_price	float	历史最低价
pre_price	float	盘前价格
pre_high_price	float	盘前最高价
pre_low_price	float	盘前最低价
pre_volume	int	盘前成交量
pre_turnover	float	盘前成交额
pre_change_val	float	盘前涨跌额
pre_change_rate	float	盘前涨跌幅
pre_amplitude	float	盘前振幅
after_price	float	盘后价格
after_high_price	float	盘后最高价
after_low_price	float	盘后最低价
after_volume	int	盘后成交量
after_turnover	float	盘后成交额
after_change_val	float	盘后涨跌额

字段	类型	说明
after_change_rate	float	盘后涨跌幅
after_amplitude	float	盘后振幅
overnight_price	float	夜盘价格
overnight_high_price	float	夜盘最高价
overnight_low_price	float	夜盘最低价
overnight_volume	int	夜盘成交量
overnight_turnover	float	夜盘成交额
overnight_change_val	float	夜盘涨跌额
overnight_change_rate	float	夜盘涨跌幅
overnight_amplitude	float	夜盘振幅
future_valid	bool	是否期货
future_last_settle_price	float	昨结
future_position	float	持仓量
future_position_change	float	日增仓
future_main_contract	bool	是否主连合约
future_last_trade_time	str	最后交易时间
trust_valid	bool	是否基金
trust_dividend_yield	float	股息率
trust_aum	float	资产规模
trust_outstanding_units	int	总发行量

字段	类型	说明
trust_netAssetValue	float	单位净值
trust_premium	float	溢价
trust_assetClass	AssetClass	资产类别

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_market_snapshot(['HK.00700', 'US.AAPL'])
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
code   name              update_time  last_price  open_price  high_price  low_pric
0  HK.00700  腾讯控股       2025-04-07 16:09:07      435.40      441.80      462.4
1   US.AAPL    苹果  2025-04-07 05:30:43.301      188.38      193.89      199.88

   wrt_issue_vol  wrt_street_ratio  wrt_delta  wrt_implied_volatility  wrt_premiu
0            NaN               NaN        NaN                     NaN          Na
1            NaN               NaN        NaN                     NaN          Na

   trust_outstanding_units  trust_netAssetValue  trust_premium trust_assetClass p
0                      NaN                  NaN            NaN              N/A
1                      NaN                  NaN            NaN              N/A
HK.00700
['HK.00700', 'US.AAPL']
```

接口限制

- 每 30 秒内最多请求 60 次快照。
- 每次请求，接口参数 **股票代码列表** 支持传入的标的数量上限是 400 个。

获取实时报价

`get_stock_quote(code_list)`

- **介绍**

  获取已订阅股票的实时报价，必须要先订阅。

- **参数**

参数	类型	说明
code_list	list	股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回报价数据
	str	当 ret != RET_OK，返回错误描述

  - 报价数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
data_date	str	日期
data_time	str	当前价更新时间
last_price	float	最新价格

字段	类型	说明
open_price	float	今日开盘价
high_price	float	最高价格
low_price	float	最低价格
prev_close_price	float	昨收盘价格
volume	int	成交数量
turnover	float	成交金额
turnover_rate	float	换手率
amplitude	int	振幅
suspension	bool	是否停牌
listing_date	str	上市日期
price_spread	float	当前向上的价差
dark_status	DarkStatus	暗盘交易状态
sec_status	SecurityStatus	股票状态
strike_price	float	行权价
contract_size	float	每份合约数
open_interest	int	未平仓合约数
implied_volatility	float	隐含波动率
premium	float	溢价
delta	float	希腊值 Delta
gamma	float	希腊值 Gamma

字段	类型	说明
vega	float	希腊值 Vega
theta	float	希腊值 Theta
rho	float	希腊值 Rho
index_option_type	IndexOptionType	指数期权类型
net_open_interest	int	净未平仓合约数
expiry_date_distance	int	距离到期日天数
contract_nominal_value	float	合约名义金额
owner_lot_multiplier	float	相等正股手数
option_area_type	OptionAreaType	期权类型（按行权时间）
contract_multiplier	float	合约乘数
pre_price	float	盘前价格
pre_high_price	float	盘前最高价
pre_low_price	float	盘前最低价
pre_volume	int	盘前成交量
pre_turnover	float	盘前成交额
pre_change_val	float	盘前涨跌额
pre_change_rate	float	盘前涨跌幅
pre_amplitude	float	盘前振幅
after_price	float	盘后价格
after_high_price	float	盘后最高价

字段	类型	说明
after_low_price	float	盘后最低价
after_volume	int	盘后成交量
after_turnover	float	盘后成交额
after_change_val	float	盘后涨跌额
after_change_rate	float	盘后涨跌幅
after_amplitude	float	盘后振幅
overnight_price	float	夜盘价格
overnight_high_price	float	夜盘最高价
overnight_low_price	float	夜盘最低价
overnight_volume	int	夜盘成交量
overnight_turnover	float	夜盘成交额
overnight_change_val	float	夜盘涨跌额
overnight_change_rate	float	夜盘涨跌幅
overnight_amplitude	float	夜盘振幅
last_settle_price	float	昨结
position	float	持仓量
position_change	float	日增仓

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

```

```python
ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.QUOTE], subscrib
# 先订阅 K 线类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推送给
if ret_sub == RET_OK:  # 订阅成功
    ret, data = quote_ctx.get_stock_quote(['US.AAPL'])  # 获取订阅股票报价的实时数
    if ret == RET_OK:
        print(data)
        print(data['code'][0])   # 取第一条的股票代码
        print(data['code'].values.tolist())   # 转为 list
    else:
        print('error:', data)
else:
    print('subscription failed', err_message)
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
code name   data_date     data_time  last_price  open_price  high_price  low_pric
0  US.AAPL   苹果  2025-04-07  05:37:21.794      188.38      193.89      199.88
US.AAPL
['US.AAPL']
```

提示

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时报价回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

获取实时摆盘

`get_order_book(code, num=10)`

- **介绍**

  获取已订阅股票的实时摆盘，必须要先订阅。

- **参数**

参数	类型	说明
code	str	股票代码
name	str	股票名称
num	int	请求摆盘档数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	dict	当 ret == RET_OK，返回摆盘数据
	str	当 ret != RET_OK，返回错误描述

  - 摆盘数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
svr_recv_time_bid	str	富途服务器从交易所收到买盘数据的时间

字段	类型	说明
svr_recv_time_ask	str	富途服务器从交易所收到卖盘数据的时间
Bid	list	每个元祖包含如下信息：委托价格，委托数量，委托订单数，委托订单明细
Ask	list	每个元祖包含如下信息：委托价格，委托数量，委托订单数，委托订单明细

其中，Bid 和 Ask 字段的结构如下：

```
'Bid': [ (bid_price1, bid_volume1, order_num, {'orderid1': order_volume1, 'orderi
'Ask': [ (ask_price1, ask_volume1, order_num, {'orderid1': order_volume1, 'orderi
```

- Example

```python
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret_sub = quote_ctx.subscribe(['US.AAPL'], [SubType.ORDER_BOOK], subscribe_push=
# 先订阅买卖摆盘类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推
if ret_sub == RET_OK:  # 订阅成功
    ret, data = quote_ctx.get_order_book('US.AAPL', num=3)  # 获取一次 3 档实时摆
    if ret == RET_OK:
        print(data)
    else:
        print('error:', data)
else:
    print('subscription failed')
quote_ctx.close()  # 关闭当条连接，OpenD 会在 1 分钟后自动取消相应股票相应类型的订阅
```

- Output

```
{'code': 'US.AAPL', 'name': '苹果', 'svr_recv_time_bid': '2025-04-07 05:39:20.352
```

接口限制

- moomoo 服务器从交易所收到数据的时间字段，仅支持A股正股、港股正股、ETFs、窝轮、牛熊，且仅开盘时间才有此数据。
- moomoo 服务器从交易所收到数据的时间字段，部分情况下接收时间可能为零，例如：服务器重启或第一次推送的缓存数据。

提示

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时摆盘回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- 美股市场，会返回当前交易时段的实时摆盘数据，无需设置时段。

获取实时 K 线

`get_cur_kline(code, num, ktype=KLType.K_DAY, autype=AuType.QFQ)`

- **介绍**

  获取已订阅股票的实时 K 线数据，必须要先订阅。

- **参数**

参数	类型	说明
code	str	股票代码
name	str	股票名称
num	int	K 线数据个数
ktype	KLType	K 线类型
autype	AuType	复权类型

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回 K 线数据数据
	str	当 ret != RET_OK，返回错误描述

  - K 线数据格式如下：

字段	类型	说明
code	str	股票代码

字段	类型	说明
name	str	股票名称
time_key	str	时间
open	float	开盘价
close	float	收盘价
high	float	最高价
low	float	最低价
volume	int	成交量
turnover	float	成交额
pe_ratio	float	市盈率
turnover_rate	float	换手率
last_close	float	昨收价

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.K_DAY], subscri
# 先订阅 K 线类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推送
if ret_sub == RET_OK:  # 订阅成功
    ret, data = quote_ctx.get_cur_kline('US.AAPL', 2, KLType.K_DAY, AuType.QFQ)
    if ret == RET_OK:
        print(data)
        print(data['turnover_rate'][0])   # 取第一条的换手率
        print(data['turnover_rate'].values.tolist())   # 转为 list
    else:
        print('error:', data)
else:
```

```
    print('subscription failed', err_message)
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
code name             time_key    open   close    high     low      volume      tu
0  US.AAPL   苹果  2025-04-03 00:00:00  205.54  203.19  207.49  201.25  103419006
1  US.AAPL   苹果  2025-04-04 00:00:00  193.89  188.38  199.88  187.34  125910913
0.00689
[0.00689, 0.00838]
```

接口限制

- 此接口为获取实时 K 线接口，最多能获取最近的 1000 根。如需获取历史 K 线，请参考 获取历史 K 线
- 市盈率和换手率字段，只有日 K 及以上周期的正股才有数据
- **期权**，仅提供日K, 1分K，5分K，15分K，60分K。

提示

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时 K 线回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

获取实时分时

`get_rt_data(code)`

- **介绍**

  获取已订阅股票的实时分时数据，必须要先订阅。

- **参数**

参数	类型	说明
code	str	股票

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回分时数据
	str	当 ret != RET_OK，返回错误描述

  - 分时数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
time	str	时间
is_blank	bool	数据状态
opened_mins	int	零点到当前多少分钟

字段	类型	说明
cur_price	float	当前价格
last_close	float	昨天收盘的价格
avg_price	float	平均价格 
volume	float	成交量
turnover	float	成交金额

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.RT_DATA], subscr
# 先订阅分时数据类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推
if ret_sub == RET_OK:   # 订阅成功
    ret, data = quote_ctx.get_rt_data('US.AAPL')   # 获取一次分时数据
    if ret == RET_OK:
        print(data)
    else:
        print('error:', data)
else:
    print('subscription failed', err_message)
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
code  name                time  is_blank  opened_mins  cur_price  last_close   a
0    US.AAPL   苹果  2025-04-06 20:01:00     False         1201     183.00      1
..       ...    ...                  ...       ...          ...        ...
586  US.AAPL   苹果  2025-04-07 05:47:00     False          347     181.26      1

[587 rows x 10 columns]
```

> **提示**

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时分时回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

获取实时逐笔

`get_rt_ticker(code, num=500)`

- **介绍**

  获取已订阅股票的实时逐笔数据，必须要先订阅。

- **参数**

参数	类型	说明
code	str	股票代码
num	int	最近逐笔个数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回逐笔数据
	str	当 ret != RET_OK，返回错误描述

  - 逐笔数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
sequence	int	逐笔序号
time	str	成交时间

字段	类型	说明
price	float	成交价格
volume	int	成交数量
turnover	float	成交金额
ticker_direction	TickerDirect	逐笔方向
type	TickerType	逐笔类型

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret_sub, err_message = quote_ctx.subscribe(['US.AAPL'], [SubType.TICKER], subscri
# 先订阅逐笔类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推送给
if ret_sub == RET_OK:  # 订阅成功
    ret, data = quote_ctx.get_rt_ticker('US.AAPL', 2)  # 获取美股AAPL最近2个逐笔
    if ret == RET_OK:
        print(data)
        print(data['turnover'][0])   # 取第一条的成交金额
        print(data['turnover'].values.tolist())   # 转为 list
    else:
        print('error:', data)
else:
    print('subscription failed', err_message)
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
code name                    time    price  volume  turnover ticker_direction
0  US.AAPL   苹果  2025-04-07 05:50:23.745  181.70       2    363.40          NEU
1  US.AAPL   苹果  2025-04-07 05:50:24.170  181.73       1    181.73          NEU
363.4
[363.4, 181.73]
```

接口限制

- 最多能获取最近 1000 个逐笔数据，更多历史逐笔数据暂未提供
- 港股期权期货在 LV1 权限下，不支持获取逐笔

提示

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时逐笔回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

获取实时经纪队列

`get_broker_queue(code)`

- **介绍**

  获取已订阅股票的实时经纪队列数据，必须要先订阅。

- **参数**

参数	类型	说明
code	str	股票代码

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
bid_frame_table	pd.DataFrame	当 ret == RET_OK，bid_frame_table 返回买盘经纪队列数据
	str	当 ret != RET_OK，bid_frame_table 返回错误描述
ask_frame_table	pd.DataFrame	当 ret == RET_OK，ask_frame_table 返回卖盘经纪队列数据
	str	当 ret != RET_OK，ask_frame_table 返回错误描述

  - 买盘经纪队列格式如下：

字段	类型	说明
code	str	股票代码

字段	类型	说明
name	str	股票名称
bid_broker_id	int	经纪买盘 ID
bid_broker_name	str	经纪买盘名称
bid_broker_pos	int	经纪档位
order_id	int	交易所订单 ID
order_volume	int	单笔委托数量

- 卖盘经纪队列格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
ask_broker_id	int	经纪卖盘 ID
ask_broker_name	str	经纪卖盘名称
ask_broker_pos	int	经纪档位
order_id	int	交易所订单 ID
order_volume	int	单笔委托数量

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret_sub, err_message = quote_ctx.subscribe(['HK.00700'], [SubType.BROKER], subscr
# 先订阅经纪队列类型。订阅成功后 OpenD 将持续收到服务器的推送，False 代表暂时不需要推
if ret_sub == RET_OK:   # 订阅成功
    ret, bid_frame_table, ask_frame_table = quote_ctx.get_broker_queue('HK.00700
    if ret == RET_OK:
```

```
        print(bid_frame_table)
    else:
        print('error:', bid_frame_table)
else:
    print(err_message)
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
        code  name  bid_broker_id bid_broker_name  bid_broker_pos order_id order_
0   HK.00700  腾讯控股           5338          J.P.摩根               1      N/A
..       ...   ...             ...             ...             ...      ...
36  HK.00700  腾讯控股           8305  富途证券国际(香港)有限公司               4

[37 rows x 7 columns]
```

提示

- 此接口提供了一次性获取实时数据的功能，如需持续获取推送数据，请参考 实时经纪队列回调 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?
- 港股 LV1 权限下，不支持获取经纪队列数据

获取标的市场状态

`get_market_state(code_list)`

- **介绍**

  获取指定标的的市场状态

- **参数**

参数	类型	说明
code_list	list	需要查询市场状态的股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回市场状态数据
	str	当 ret != RET_OK，返回错误描述

  - 市场状态数据

字段	类型	说明
code	str	股票代码
stock_name	str	股票名称
market_state	MarketState	市场状态

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
```

```python

ret, data = quote_ctx.get_market_state(['SZ.000001', 'HK.00700'])
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code          stock_name   market_state
0  SZ.000001     平安银行      AFTERNOON
1  HK.00700      腾讯控股      AFTERNOON
```

接口限制

- 每 30 秒内最多请求 10 次获取标的市场状态接口。
- 每次请求的股票代码个数上限为 400 个。

获取资金流向

`get_capital_flow(stock_code, period_type = PeriodType.INTRADAY, start=None, end=None)`

- **介绍**

  获取个股资金流向

- **参数**

参数	类型	说明
stock_code	str	股票代码
period_type	PeriodType	周期类型
start	str	开始时间
end	str	结束时间

  - start 和 end 的组合如下

start 类型	end 类型	说明
str	str	start 和 end 分别为指定的日期
None	str	start 为 end 往前 365 天
str	None	end 为 start 往后 365 天
None	None	end 为 当前日期，start 往前 365 天

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果

data	pd.DataFrame	当 ret == RET_OK，返回资金流向数据
	str	当 ret != RET_OK，返回错误描述

- 资金流向数据格式如下：

字段	类型	说明
in_flow	float	整体净流入
main_in_flow	float	主力大单净流入
super_in_flow	float	特大单净流入
big_in_flow	float	大单净流入
mid_in_flow	float	中单净流入
sml_in_flow	float	小单净流入
capital_flow_item_time	str	开始时间
last_valid_time	str	数据最后有效时间

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_capital_flow("HK.00700", period_type = PeriodType.INTRA
if ret == RET_OK:
    print(data)
    print(data['in_flow'][0])    # 取第一条的净流入的资金额度
    print(data['in_flow'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- **Output**

```
    last_valid_time       in_flow  ...  main_in_flow  capital_flow_item_time
0               N/A -1.857915e+08  ... -1.066828e+08     2021-06-08 00:00:00
..              ...           ...  ...           ...                     ...
245             N/A  2.179240e+09  ...  2.143345e+09     2022-06-08 00:00:00

[246 rows x 8 columns]
-185791500.0
[-185791500.0, -18315000.0, -672100100.0, -714394350.0, -698391950.0, -818886750
..                   ...           ...                   ...
2031460.0, 638067040.0, 622466600.0, -351788160.0, -328529240.0, 715415020.0, 767
```

接口限制

- 每 30 秒内最多请求 30 次获取资金流向接口。
- 仅支持正股、窝轮和基金。
- 历史周期（日、月、年）仅提供最近 1 年数据；实时周期仅提供最新一天的数据。
- 返回数据只包括盘中数据，不包含盘前盘后数据。

获取资金分布

`get_capital_distribution(stock_code)`

- **介绍**

  获取资金分布

- **参数**

参数	类型	说明
stock_code	str	股票代码

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回股票资金分布数据
	str	当 ret != RET_OK，返回错误描述

  - 资金分布数据格式如下：

字段	类型	说明
capital_in_super	float	流入资金额度，特大单
capital_in_big	float	流入资金额度，大单
capital_in_mid	float	流入资金额度，中单
capital_in_small	float	流入资金额度，小单
capital_out_super	float	流出资金额度，特大单

字段	类型	说明
capital_out_big	float	流出资金额度，大单
capital_out_mid	float	流出资金额度，中单
capital_out_small	float	流出资金额度，小单
update_time	str	更新时间字符串

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_capital_distribution("HK.00700")
if ret == RET_OK:
    print(data)
    print(data['capital_in_big'][0])    # 取第一条的流入资金额度，大单
    print(data['capital_in_big'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
   capital_in_super  capital_in_big  ...  capital_out_small          update_time
0      2.261085e+09    2.141964e+09  ...       2.887413e+09  2022-06-08 15:59:59

[1 rows x 9 columns]
2141963720.0
[2141963720.0]
```

接口限制

- 每 30 秒内最多请求 30 次获取资金分布接口。
- 仅支持正股、窝轮和基金。
- 更多资金分布介绍，请参考 这里。

- 返回数据只包括盘中数据，不包含盘前盘后数据。

获取股票所属板块

`get_owner_plate(code_list)`

- **介绍**

  获取单支或多支股票的所属板块信息列表

- **参数**

参数	类型	说明
code_list	list	股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回所属板块数据
	str	当 ret != RET_OK，返回错误描述

  - 所属板块数据格式如下：

字段	类型	说明
code	str	证券代码
name	str	股票名称
plate_code	str	板块代码
plate_name	str	板块名字
plate_type	Plate	板块类型

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

code_list = ['HK.00001']
ret, data = quote_ctx.get_owner_plate(code_list)
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['plate_code'].values.tolist())   # 板块代码转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
        code name          plate_code plate_name plate_type
0   HK.00001   长和  HK.HSI Constituent      恒指成份股      OTHER
..       ...  ...                  ...         ...        ...
8   HK.00001   长和           HK.BK1983     香港股票ADR      OTHER

[9 rows x 5 columns]
HK.00001
['HK.HSI Constituent', 'HK.GangGuTong', 'HK.BK1000', 'HK.BK1061', 'HK.BK1107', 'H
```

接口限制

- 每 30 秒内最多请求 10 次获取股票所属板块接口
- 每次请求的股票列表中，股票个数上限为 200 个
- 仅支持正股和指数

获取历史 K 线

`request_history_kline(code, start=None, end=None, ktype=KLType.K_DAY, autype=AuType.QFQ, fields=[KL_FIELD.ALL], max_count=1000, page_req_key=None, extended_time=False)`

- **介绍**

  获取历史 K 线

- **参数**

参数	类型	说明
code	str	股票代码
start	str	开始时间
end	str	结束时间
ktype	KLType	K 线类型
autype	AuType	复权类型
fields	KLFields	需返回的字段列表
max_count	int	本次请求最大返回的 K 线根数
page_req_key	bytes	分页请求
extended_time	bool	是否允许美股盘前盘后数据

  - start 和 end 的组合如下

Start 类型	End 类型	说明
str	str	start 和 end 分别为指定的日期

Start 类型	End 类型	说明
None	str	start 为 end 往前 365 天
str	None	end 为 start 往后 365 天
None	None	end 为当前日期，start 往前 365 天

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回历史 K 线数据
	str	当 ret != RET_OK，返回错误描述
page_req_key	bytes	下一页请求的 key

  - 历史 K 线数据格式如下:

字段	类型	说明
code	str	股票代码
name	str	股票名称
time_key	str	K 线时间
open	float	开盘价
close	float	收盘价
high	float	最高价
low	float	最低价
pe_ratio	float	市盈率

字段	类型	说明
turnover_rate	float	换手率
volume	int	成交量
turnover	float	成交额
change_rate	float	涨跌幅
last_close	float	昨收价

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret, data, page_req_key = quote_ctx.request_history_kline('US.AAPL', start='2019
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['close'].values.tolist())   # 第一页收盘价转为 list
else:
    print('error:', data)
while page_req_key != None:  # 请求后面的所有结果
    print('**********************************')
    ret, data, page_req_key = quote_ctx.request_history_kline('US.AAPL', start='
    if ret == RET_OK:
        print(data)
    else:
        print('error:', data)
print('All pages are finished!')
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
code  name            time_key       open      close       high        low  pe_
0  US.AAPL   苹果  2019-09-11 00:00:00  52.631194  53.963447  53.992409  52.54913
..       ...   ...                  ...        ...        ...        ...
4  US.AAPL   苹果  2019-09-17 00:00:00  53.087346  53.265945  53.294907  52.88461

[5 rows x 13 columns]
```

```
US.AAPL
[53.9634465, 53.84156475, 52.7953125, 53.072865, 53.265945]
************************************
       code  name             time_key       open      close       high        lo
0  US.AAPL   苹果  2019-09-18 00:00:00  53.352831  53.76554  53.784847  52.961844
All pages are finished!
```

接口限制

- 分 K 提供最近 8 年数据，日 K 提供最近 20 年的数据，日 K 以上不限制。
- 我们会根据您账户的资产和交易的情况，下发历史 K 线额度。因此，30 天内您只能获取有限只股票的历史 K 线数据。具体规则参见 订阅额度 & 历史 K 线额度。您当日消耗的历史 K 线额度，会在 30 天后自动释放。
- 每 30 秒内最多请求 60 次历史 K 线接口。注意：如果您是分页获取数据，此限频规则仅适用于每只股票的首页，后续页请求不受限频规则的限制。
- **换手率**，仅提供日 K 及以上级别。
- **期权**，仅提供日K, 1分K，5分K，15分K，60分K。
- 美股 **盘前、盘后、夜盘 K 线**，仅支持 60 分钟及以下级别。由于美股盘前盘后和夜盘时段为非常规交易时段，此时段的 K 线数据可能不足 2 年。
- 美股的 **成交额**，仅提供 2015-10-12 之后的数据。

获取复权因子

`get_rehab(code)`

- **介绍**

  获取股票的复权因子

- **参数**

参数	类型	说明
code	str	股票代码

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回复权数据
	str	当 ret != RET_OK，返回错误描述

  - 复权数据格式如下：

字段	类型	说明
ex_div_date	str	除权除息日
split_base	float	拆股分子
split_ert	float	拆股分母
join_base	float	合股分子
join_ert	float	合股分母

字段	类型	说明
split_ratio	float	拆合股比例
per_cash_div	float	每股派现
bonus_base	float	送股分子
bonus_ert	float	送股分母
per_share_div_ratio	float	送股比例
transfer_base	float	转增股分子
transfer_ert	float	转增股分母
per_share_trans_ratio	float	转增股比例
allot_base	float	配股分子
allot_ert	float	配股分母
allotment_ratio	float	配股比例
allotment_price	float	配股价
add_base	float	增发股分子
add_ert	float	增发股分母
stk_spo_ratio	float	增发比例
stk_spo_price	float	增发价格
spin_off_base	float	分立分子
spin_off_ert	float	分立分母
spin_off_ratio	float	分立比例
forward_adj_factorA	float	前复权因子 A

字段	类型	说明
forward_adj_factorB	float	前复权因子 B
backward_adj_factorA	float	后复权因子 A
backward_adj_factorB	float	后复权因子 B

前复权价格 = 不复权价格 × 前复权因子 A + 前复权因子 B
后复权价格 = 不复权价格 × 后复权因子 A + 后复权因子 B

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_rehab("HK.00700")
if ret == RET_OK:
    print(data)
    print(data['ex_div_date'][0])    # 取第一条的除权除息日
    print(data['ex_div_date'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    ex_div_date  split_ratio  per_cash_div  per_share_div_ratio  per_share_trans_
0   2005-04-19          NaN          0.07                  NaN
..         ...          ...           ...                  ...
15  2019-05-17          NaN          1.00                  NaN

[16 rows x 16 columns]
2005-04-19
['2005-04-19', '2006-05-15', '2007-05-09', '2008-05-06', '2009-05-06', '2010-05-
```

接口限制

- 每 30 秒内最多请求 60 次获取复权因子接口。

获取期权链到期日

`get_option_expiration_date(code, index_option_type=IndexOptionType.NORMAL)`

- **介绍**

  通过标的股票，查询期权链的所有到期日。如需获取完整期权链，请配合 获取期权链 接口使用。

- **参数**

参数	类型	说明
code	str	标的股票代码
index_option_type	IndexOptionType	指数期权类型

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回期权链到期日相关数据
	str	当 ret != RET_OK，返回错误描述

  - 期权链到期日数据格式如下：

字段	类型	说明
strike_time	str	期权链行权日
option_expiry_date_distance	int	距离到期日天数
expiration_cycle	ExpirationCycle	交割周期

- **Example**

```python
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret, data = quote_ctx.get_option_expiration_date(code='HK.00700')
if ret == RET_OK:
    print(data)
    print(data['strike_time'].values.tolist())  # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
  strike_time  option_expiry_date_distance expiration_cycle
0  2021-04-29                            4              N/A
1  2021-05-28                           33              N/A
2  2021-06-29                           65              N/A
3  2021-07-29                           95              N/A
4  2021-09-29                          157              N/A
5  2021-12-30                          249              N/A
6  2022-03-30                          339              N/A
['2021-04-29', '2021-05-28', '2021-06-29', '2021-07-29', '2021-09-29', '2021-12-
```

> **接口限制**
>
> - 每 30 秒内最多请求 60 次获取期权链到期日接口

获取期权链

```
get_option_chain(code, index_option_type=IndexOptionType.NORMAL, start=None, end=None, option_type=OptionType.ALL, option_cond_type=OptionCondType.ALL, data_filter=None)
```

- **介绍**

  通过标的股票查询期权链。此接口仅返回期权链的静态信息，如需获取报价或摆盘等动态信息，请用此接口返回的股票代码，自行 订阅 所需要的类型。

- **参数**

参数	类型	说明
code	str	标的股票代码
index_option_type	IndexOptionType	指数期权类型
start	str	开始日期，该日期指到期日
end	str	结束日期（包括这一天），该日期指到期日
option_type	OptionType	期权看涨看跌类型
option_cond_type	OptionCondType	期权价内外类型
data_filter	OptionDataFilter	数据筛选条件

  - start 和 end 的组合如下：

Start 类型	End 类型	说明
str	str	start 和 end 分别为指定的日期
None	str	start 为 end 往前 30 天

Start 类型	End 类型	说明
str	None	end 为 start 往后30天
None	None	start 为当前日期，end 往后 30 天

- OptionDataFilter 字段如下

字段	类型	说明
implied_volatility_min	float	隐含波动率过滤起点
implied_volatility_max	float	隐含波动率过滤终点
delta_min	float	希腊值 Delta 过滤起点
delta_max	float	希腊值 Delta 过滤终点
gamma_min	float	希腊值 Gamma 过滤起点
gamma_max	float	希腊值 Gamma 过滤终点
vega_min	float	希腊值 Vega 过滤起点
vega_max	float	希腊值 Vega 过滤终点
theta_min	float	希腊值 Theta 过滤起点
theta_max	float	希腊值 Theta 过滤终点
rho_min	float	希腊值 Rho 过滤起点
rho_max	float	希腊值 Rho 过滤终点
net_open_interest_min	float	净未平仓合约数过滤起点
net_open_interest_max	float	净未平仓合约数过滤终点
open_interest_min	float	未平仓合约数过滤起点
open_interest_max	float	未平仓合约数过滤终点

字段	类型	说明
vol_min	float	成交量过滤起点
vol_max	float	成交量过滤终点

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回期权链数据
	str	当 ret != RET_OK，返回错误描述

  - 期权链数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	名字
lot_size	int	每手股数，期权表示每份合约股数
stock_type	SecurityType	股票类型
option_type	OptionType	期权类型
stock_owner	str	标的股
strike_time	str	行权日
strike_price	float	行权价
suspension	bool	是否停牌
stock_id	int	股票 ID

字段	类型	说明
index_option_type	IndexOptionType	指数期权类型
expiration_cycle	ExpirationCycle	交割周期
option_standard_type	OptionStandardType	期权标准类型
option_settlement_mode	OptionSettlementMode	期权结算方式

- Example

```
from moomoo import *
import time
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret1, data1 = quote_ctx.get_option_expiration_date(code='HK.00700')

filter1 = OptionDataFilter()
filter1.delta_min = 0
filter1.delta_max = 0.1

if ret1 == RET_OK:
    expiration_date_list = data1['strike_time'].values.tolist()
    for date in expiration_date_list:
        ret2, data2 = quote_ctx.get_option_chain(code='HK.00700', start=date, end
        if ret2 == RET_OK:
            print(data2)
            print(data2['code'][0])  # 取第一条的股票代码
            print(data2['code'].values.tolist())  # 转为 list
        else:
            print('error:', data2)
        time.sleep(3)
else:
    print('error:', data1)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
                    code                 name  lot_size stock_type option_type s
0     HK.TCH210429C350000   腾讯 210429 350.00 购       100       DRVT        CAL
1     HK.TCH210429P350000   腾讯 210429 350.00 沽       100       DRVT         PU
```

```
2     HK.TCH210429C360000   腾讯 210429 360.00 购       100       DRVT        CAL
3     HK.TCH210429P360000   腾讯 210429 360.00 沽       100       DRVT         PU
4     HK.TCH210429C370000   腾讯 210429 370.00 购       100       DRVT        CAL
5     HK.TCH210429P370000   腾讯 210429 370.00 沽       100       DRVT         PU
HK.TCH210429C350000
['HK.TCH210429C350000', 'HK.TCH210429P350000', 'HK.TCH210429C360000', 'HK.TCH2104
...
                    code                 name  lot_size stock_type option_type stoc
0   HK.TCH220330C490000  腾讯 220330 490.00 购       100       DRVT        CALL
1   HK.TCH220330P490000  腾讯 220330 490.00 沽       100       DRVT         PUT
2   HK.TCH220330C500000  腾讯 220330 500.00 购       100       DRVT        CALL
3   HK.TCH220330P500000  腾讯 220330 500.00 沽       100       DRVT         PUT
4   HK.TCH220330C510000  腾讯 220330 510.00 购       100       DRVT        CALL
5   HK.TCH220330P510000  腾讯 220330 510.00 沽       100       DRVT         PUT
HK.TCH220330C490000
['HK.TCH220330C490000', 'HK.TCH220330P490000', 'HK.TCH220330C500000', 'HK.TCH2203
```

接口限制

- 每 30 秒内最多请求 10 次获取期权链接口
- 传入的时间跨度上限为 30 天

提示

- 此接口不支持查询已过期的期权链，**结束日期** 参数请输入今天或未来的日期
- Open interest (OI) 数据每日更新，更新时点取决于具体交易所。美股期权在盘前时段更新，港股期权在盘后更新。

筛选窝轮

`get_warrant(stock_owner='', req=None)`

- **介绍**

  筛选窝轮（仅用于筛选香港市场的窝轮、牛熊证、界内证）

- **参数**

参数	类型	说明
stock_owner	str	所属正股的股票代码
req	WarrantRequest	筛选参数组合

  - WarrantRequest 类型字段说明如下：

字段	类型	说明
begin	int	数据起始点
num	int	请求数据个数
sort_field	SortField	根据哪个字段排序
ascend	bool	排序方向
type_list	list	窝轮类型过滤列表
issuer_list	list	发行人过滤列表
maturity_time_min	str	到期日过滤范围的开始时间
maturity_time_max	str	到期日过滤范围的结束时间
ipo_period	IpoPeriod	上市时段

字段	类型	说明
price_type	PriceType	价内/价外
status	WarrantStatus	窝轮状态
cur_price_min	float	最新价的过滤下限
cur_price_max	float	最新价的过滤上限
strike_price_min	float	行使价的过滤下限
strike_price_max	float	行使价的过滤上限
street_min	float	街货占比的过滤下限
street_max	float	街货占比的过滤上限
conversion_min	float	换股比率的过滤下限
conversion_max	float	换股比率的过滤上限
vol_min	int	成交量的过滤下限
vol_max	int	成交量的过滤上限
premium_min	float	溢价的过滤下限
premium_max	float	溢价的过滤上限
leverage_ratio_min	float	杠杆比率的过滤下限
leverage_ratio_max	float	杠杆比率的过滤上限
delta_min	float	对冲值的过滤下限
delta_max	float	对冲值的过滤上限
implied_min	float	引伸波幅的过滤下限
implied_max	float	引伸波幅的过滤上限

字段	类型	说明
recovery_price_min	float	收回价的过滤下限
recovery_price_max	float	收回价的过滤上限
price_recovery_ratio_min	float	正股距收回价的过滤下限
price_recovery_ratio_max	float	正股距收回价的过滤上限

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	tuple	当 ret == RET_OK，返回窝轮数据
	str	当 ret != RET_OK，返回错误描述

  - 窝轮数据组成如下：

字段	类型	说明
warrant_data_list	pd.DataFrame	筛选后的窝轮数据
last_page	bool	是否是最后一页
all_count	int	筛选结果中的窝轮总数量

    - warrant_data_list 返回的 pd dataframe 数据格式：

字段	类型	说明
stock	str	窝轮代码
stock_owner	str	所属正股
type	WrtType	窝轮类型

字段	类型	说明
issuer	Issuer	发行人
maturity_time	str	到期日
list_time	str	上市时间
last_trade_time	str	最后交易日
recovery_price	float	收回价
conversion_ratio	float	换股比率
lot_size	int	每手数量
strike_price	float	行使价
last_close_price	float	昨收价
name	str	名称
cur_price	float	当前价
price_change_val	float	涨跌额
status	WarrantStatus	窝轮状态
bid_price	float	买入价
ask_price	float	卖出价
bid_vol	int	买量
ask_vol	int	卖量
volume	int	成交量
turnover	float	成交额
score	float	综合评分

字段	类型	说明
premium	float	溢价
break_even_point	float	打和点
leverage	float	杠杆比率
ipop	float	价内/价外
price_recovery_ratio	float	正股距收回价
conversion_price	float	换股价
street_rate	float	街货占比
street_vol	int	街货量
amplitude	float	振幅
issue_size	int	发行量
high_price	float	最高价
low_price	float	最低价
implied_volatility	float	引伸波幅
delta	float	对冲值
effective_leverage	float	有效杠杆
upper_strike_price	float	上限价
lower_strike_price	float	下限价
inline_price_status	PriceType	界内界外

- **Example**

```python
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

req = WarrantRequest()
req.sort_field = SortField.TURNOVER
req.type_list = WrtType.CALL
req.cur_price_min = 0.1
req.cur_price_max = 0.2
ret, ls = quote_ctx.get_warrant("HK.00700", req)
if ret == RET_OK:  # 先判断接口返回是否正常，再取数据
    warrant_data_list, last_page, all_count = ls
    print(len(warrant_data_list), all_count, warrant_data_list)
    print(warrant_data_list['stock'][0])    # 取第一条的窝轮代码
    print(warrant_data_list['stock'].values.tolist())   # 转为 list
else:
    print('error: ', ls)

req = WarrantRequest()
req.sort_field = SortField.TURNOVER
req.issuer_list = ['UB','CS','BI']
ret, ls = quote_ctx.get_warrant(Market.HK, req)
if ret == RET_OK:
    warrant_data_list, last_page, all_count = ls
    print(len(warrant_data_list), all_count, warrant_data_list)
else:
    print('error: ', ls)

quote_ctx.close()  # 所有接口结尾加上这条 close，防止连接条数用尽
```

- Output

```
2 2
    stock        name stock_owner  type issuer maturity_time   list_time last_tra
0   HK.20306  腾讯麦银零乙购A.C    HK.00700  CALL     MB    2020-12-01  2019-06-2
1   HK.16545  腾讯法兴一二购B.C    HK.00700  CALL     SG    2021-02-26  2020-07-1
HK.20306
['HK.20306', 'HK.16545']

200 358
    stock        name stock_owner    type issuer maturity_time   list_time last_
0    HK.19839  平安瑞银零乙购A.C    HK.02318    CALL     UB    2020-12-31  2017-1
1    HK.20084  平安中银零乙购A.C    HK.02318    CALL     BI    2020-12-31  2017-1
```

```
......
198  HK.56886  恒指瑞银三一牛F.C   HK.800000    BULL     UB    2023-01-30  2020-0
199  HK.56895  小米瑞银零乙牛D.C    HK.01810     BULL     UB    2020-12-30  2020-0
```

接口限制

- 港股 BMP 权限不支持调用此接口
- 每 30 秒内最多请求 60 次筛选窝轮接口
- 每次请求的数据个数上限为 200 个

获取窝轮和期货列表

`get_referencestock_list(code, reference_type)`

- **介绍**

  获取证券的关联数据，如：获取正股相关窝轮、获取期货相关合约

- **参数**

参数	类型	说明
code	str	证券代码
reference_type	SecurityReferenceType	要获得的相关数据

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回证券的关联数据
	str	当 ret != RET_OK，返回错误描述

  - 证券的关联数据格式如下：

字段	类型	说明
code	str	证券代码
lot_size	int	每手股数，期货表示合约乘数
stock_type	SecurityType	证券类型
stock_name	str	证券名字

字段	类型	说明
list_time	str	上市时间
wrt_valid	bool	是否是窝轮
wrt_type	WrtType	窝轮类型
wrt_code	str	所属正股
future_valid	bool	是否是期货
future_main_contract	bool	是否主连合约
future_last_trade_time	str	最后交易时间

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

# 获取正股相关的窝轮
ret, data = quote_ctx.get_referencestock_list('HK.00700', SecurityReferenceType.W
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
print('**************************************')
# 港期相关合约
ret, data = quote_ctx.get_referencestock_list('HK.A50main', SecurityReferenceType
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
        code  lot_size stock_type stock_name   list_time  wrt_valid wrt_type  wr
0     HK.24719      1000    WARRANT    腾讯东亚九四沽A  2018-07-20       True
..         ...       ...        ...                ...       ...        ...
1617  HK.63402     10000    WARRANT    腾讯高盛一八牛Y  2020-11-26       True

[1618 rows x 11 columns]
HK.24719
['HK.24719', 'HK.27886', 'HK.28621', 'HK.14339', 'HK.27952', 'HK.18693', 'HK.203
...        ...       ...        ...        ...         ...        ...      ...
'HK.63402']
****************************************
        code  lot_size stock_type         stock_name list_time  wrt_valid  wrt_ty
0  HK.A50main      5000     FUTURE      安硕富时 A50 ETF主连(2012)
..         ...       ...        ...                ...       ...        ...
5  HK.A502106      5000     FUTURE      安硕富时 A50 ETF2106               False

[6 rows x 11 columns]
HK.A50main
['HK.A50main', 'HK.A502011', 'HK.A502012', 'HK.A502101', 'HK.A502103', 'HK.A5021
```

接口限制

- 每 30 秒内最多请求 10 次获取证券关联数据接口
- 当获取正股相关窝轮时，不受上述限频限制

获取期货合约资料

`get_future_info(code_list)`

- **介绍**

  获取期货合约资料

- **参数**

参数	类型	说明
code_list	list	股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回期货合约资料数据
	str	当 ret != RET_OK，返回错误描述

  - 期货合约资料数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称
owner	str	标的
exchange	str	交易所
type	str	合约类型

字段	类型	说明
size	float	合约规模
size_unit	str	合约规模单位
price_currency	str	报价货币
price_unit	str	报价单位
min_change	float	最小变动
min_change_unit	str	最小变动的单位
trade_time	str	交易时间
time_zone	str	时区
last_trade_time	str	最后交易时间
exchange_format_url	str	交易所规格链接 url
origin_code	str	实际合约代码

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_future_info(["HK.MPImain", "HK.HAImain"])
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code      name        owner exchange  type     size size_unit price_currency p
0  HK.MPImain   内房期货主连  恒生中国内地地产指数       港交所  股指期货     50.0
1  HK.HAImain   海通证券期货主连     HK.06837      港交所  股票期货  10000.0
HK.MPImain
['HK.MPImain', 'HK.HAImain']
```

接口限制

- 每 30 秒内最多请求 30 次获取期货合约资料接口
- 每次请求的代码列表中，期货个数上限为 200 个

条件选股

`get_stock_filter(market, filter_list, plate_code=None, begin=0, num=200)`

- **介绍**

  条件选股

- **参数**

参数	类型	说明
market	Market	市场标识
filter_list	list	筛选条件的列表
plate_code	str	板块代码
begin	int	数据起始点
num	int	请求数据个数

  - SimpleFilter 对象相关参数如下：

字段	类型	说明
stock_field	StockField	简单属性
filter_min	float	区间下限
filter_max	float	区间上限
is_no_filter	bool	该字段是否不需要筛选
sort	SortDir	排序方向

  - AccumulateFilter 对象相关参数如下：

字段	类型	说明
stock_field	StockField	累积属性
filter_min	float	区间下限
filter_max	float	区间上限
is_no_filter	bool	该字段是否不需要筛选
sort	SortDir	排序方向
days	int	所筛选的数据的累计天数

- FinancialFilter 对象相关参数如下：

字段	类型	说明
stock_field	StockField	财务属性
filter_min	float	区间下限
filter_max	float	区间上限
is_no_filter	bool	该字段是否不需要筛选
sort	SortDir	排序方向
quarter	FinancialQuarter	财报累积时间

- CustomIndicatorFilter 对象相关参数如下：

字段	类型	说明
stock_field1	StockField	自定义技术指标属性
stock_field1_para	list	自定义技术指标属性参数
relative_position	RelativePosition	相对位置

字段	类型	说明
stock_field2	StockField	自定义技术指标属性
stock_field2_para	list	自定义技术指标属性参数
value	float	自定义数值
ktype	KLType	K线类型 KLType
consecutive_period	int	筛选连续周期（consecutive_period）都符合条件的数据
is_no_filter	bool	该字段是否不需要筛选

- PatternFilter 对象相关参数如下：

字段	类型	说明
stock_field	StockField	形态技术指标属性
ktype	KLType	K线类型 KLType （仅支持K_60M，K_DAY，K_WEEK，K_MON 四种时间周期）
consecutive_period	int	筛选连续周期（consecutive_period）都符合条件的数据
is_no_filter	bool	该字段是否不需要筛选

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	tuple	当 ret == RET_OK，返回选股数据
	str	当 ret != RET_OK，返回错误描述

- 选股数据元组组成如下：

字段	类型	说明
last_page	bool	是否是最后一页
all_count	int	列表总数量
stock_list	list	选股数据

- FilterStockData 类型的字段格式：

字段	类型	说明
stock_code	str	股票代码
stock_name	str	股票名字
cur_price	float	最新价
cur_price_to_highest_52weeks_ratio	float	(现价 - 52周最高)/52周最高
cur_price_to_lowest_52weeks_ratio	float	(现价 - 52周最低)/52周最低
high_price_to_highest_52weeks_ratio	float	(今日最高 - 52周最高)/52周最高
low_price_to_lowest_52weeks_ratio	float	(今日最低 - 52周最低)/52周最低
volume_ratio	float	量比
bid_ask_ratio	float	委比
lot_price	float	每手价格
market_val	float	市值
pe_annual	float	市盈率
pe_ttm	float	市盈率 TTM

字段	类型	说明
pb_rate	float	市净率
change_rate_5min	float	五分钟价格涨跌幅
change_rate_begin_year	float	年初至今价格涨跌幅
ps_ttm	float	市销率 TTM
pcf_ttm	float	市现率 TTM
total_share	float	总股数
float_share	float	流通股数
float_market_val	float	流通市值
change_rate	float	涨跌幅
amplitude	float	振幅
volume	float	日均成交量
turnover	float	日均成交额
turnover_rate	float	换手率
net_profit	float	净利润
net_profix_growth	float	净利润增长率
sum_of_business	float	营业收入
sum_of_business_growth	float	营业同比增长率
net_profit_rate	float	净利率
gross_profit_rate	float	毛利率

字段	类型	说明
debt_asset_rate	float	资产负债率
return_on_equity_rate	float	净资产收益率
roic	float	投入资本回报率
roa_ttm	float	资产回报率 TTM
ebit_ttm	float	息税前利润 TTM
ebitda	float	税息折旧及摊销前利润
operating_margin_ttm	float	营业利润率 TTM
ebit_margin	float	EBIT 利润率
ebitda_margin	float	EBITDA 利润率
financial_cost_rate	float	财务成本率
operating_profit_ttm	float	营业利润 TTM
shareholder_net_profit_ttm	float	归属于母公司的净利润
net_profit_cash_cover_ttm	float	盈利中的现金收入比例
current_ratio	float	流动比率
quick_ratio	float	速动比率
current_asset_ratio	float	流动资产率
current_debt_ratio	float	流动负债率
equity_multiplier	float	权益乘数
property_ratio	float	产权比率

字段	类型	说明
cash_and_cash_equivalents	float	现金和现金等价
total_asset_turnover	float	总资产周转率
fixed_asset_turnover	float	固定资产周转率
inventory_turnover	float	存货周转率
operating_cash_flow_ttm	float	经营活动现金流 TTM
accounts_receivable	float	应收账款净额
ebit_growth_rate	float	EBIT 同比增长率
operating_profit_growth_rate	float	营业利润同比增长率
total_assets_growth_rate	float	总资产同比增长率
profit_to_shareholders_growth_rate	float	归母净利润同比增长率
profit_before_tax_growth_rate	float	总利润同比增长率
eps_growth_rate	float	EPS 同比增长率
roe_growth_rate	float	ROE 同比增长率
roic_growth_rate	float	ROIC 同比增长率
nocf_growth_rate	float	经营现金流同比增长率
nocf_per_share_growth_rate	float	每股经营现金流同比增长率
operating_revenue_cash_cover	float	经营现金收入比
operating_profit_to_total_profit	float	营业利润占比
basic_eps	float	基本每股收益

字段	类型	说明
diluted_eps	float	稀释每股收益
nocf_per_share	float	每股经营现金净流量
price	float	最新价格
ma	float	简单均线
ma5	float	5日简单均线
ma10	float	10日简单均线
ma20	float	20日简单均线
ma30	float	30日简单均线
ma60	float	60日简单均线
ma120	float	120日简单均线
ma250	float	250日简单均线
rsi	float	RSI的值
ema	float	指数移动均线
ema5	float	5日指数移动均线
ema10	float	10日指数移动均线
ema20	float	20日指数移动均线
ema30	float	30日指数移动均线
ema60	float	60日指数移动均线
ema120	float	120日指数移动均线

字段	类型	说明
ema250	float	250日指数移动均线
kdj_k	float	KDJ 指标的 K 值
kdj_d	float	KDJ 指标的 D 值
kdj_j	float	KDJ 指标的 J 值
macd_diff	float	MACD 指标的 DIFF 值
macd_dea	float	MACD 指标的 DEA 值
macd	float	MACD 指标的 MACD 值
boll_upper	float	BOLL 指标的 UPPER 值
boll_middler	float	BOLL 指标的 MIDDLER 值
boll_lower	float	BOLL 指标的 LOWER 值

- **Example**

```
from moomoo import *
import time

quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
simple_filter = SimpleFilter()
simple_filter.filter_min = 2
simple_filter.filter_max = 1000
simple_filter.stock_field = StockField.CUR_PRICE
simple_filter.is_no_filter = False
# simple_filter.sort = SortDir.ASCEND

financial_filter = FinancialFilter()
financial_filter.filter_min = 0.5
financial_filter.filter_max = 50
financial_filter.stock_field = StockField.CURRENT_RATIO
financial_filter.is_no_filter = False
financial_filter.sort = SortDir.ASCEND
```

```
financial_filter.quarter = FinancialQuarter.ANNUAL

custom_filter = CustomIndicatorFilter()
custom_filter.ktype = KLType.K_DAY
custom_filter.stock_field1 = StockField.KDJ_K
custom_filter.stock_field1_para = [10,4,4]
custom_filter.stock_field2 = StockField.KDJ_K
custom_filter.stock_field2_para = [9,3,3]
custom_filter.relative_position = RelativePosition.MORE
custom_filter.is_no_filter = False

nBegin = 0
last_page = False
ret_list = list()
while not last_page:
    nBegin += len(ret_list)
    ret, ls = quote_ctx.get_stock_filter(market=Market.HK, filter_list=[simple_f
    if ret == RET_OK:
        last_page, all_count, ret_list = ls
        print('all count = ', all_count)
        for item in ret_list:
            print(item.stock_code)  # 取股票代码
            print(item.stock_name)  # 取股票名称
            print(item[simple_filter])   # 取 simple_filter 对应的变量值
            print(item[financial_filter])   # 取 financial_filter 对应的变量值
            print(item[custom_filter])  # 获取 custom_filter 的数值
    else:
        print('error: ', ls)
    time.sleep(3)  # 加入时间间隔，避免触发限频

quote_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
39 39 [ stock_code:HK.08103  stock_name:HMVOD视频  cur_price:2.69  current_ratio(
HK.08103
HMVOD视频
2.69
2.69
4.413
...
HK.00306
冠忠巴士集团
```

```
2.29
2.29
49.769
```

提示

- 利用获取子板块列表函数 获取子板块代码，条件选股支持的板块分别为
  1. 港股的行业板块和概念板块。
  2. 美股的行业板块
  3. 沪深的行业板块，概念板块和地域板块
- 支持的板块指数代码

代码	说明
HK.Motherboard	港股主板
HK.GEM	港股创业板
HK.BK1911	H 股主板
HK.BK1912	H 股创业板
US.NYSE	纽交所
US.AMEX	美交所
US.NASDAQ	纳斯达克
SH.3000000	上海主板
SZ.3000001	深证主板
SZ.3000004	深证创业板

接口限制

- 每 30 秒内最多请求 10 次条件选股接口
- 每页返回的筛选结果最多 200 个
- 建议筛选条件不超过 250 个，否则可能会出现“业务处理超时没返回”

- 累积属性的同一筛选条件数量上限 10 个
- 如果使用“最新价”等动态数据作为排序字段，在多页获取的间隙，数据的排序有可能发生变化
- 非同类指标不支持比较，仅限于同类指标之间建立比较关系，跨不同类型的指标比较会报错。例如：MA5 和 MA10 可以建立关系。MA5和EMA10不能建立关系。
- 自定义指标属性的同一类筛选条件超出数量上限10个
- 简单属性，财务属性，形态属性不支持对同一字段重复指定筛选条件
- 条件选股暂不支持美股盘前盘后、夜盘，筛选结果均按照盘中数据返回

获取板块内股票列表

`get_plate_stock(plate_code, sort_field=SortField.CODE, ascend=True)`

- **介绍**

  获取指定板块内的股票列表，获取股指的成分股

- **参数**

参数	类型	说明
plate_code	str	板块代码
sort_field	SortField	排序字段
ascend	bool	排序方向

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回板块股票数据
	str	当 ret != RET_OK，返回错误描述

  - 板块股票数据

字段	类型	说明
code	str	股票代码
lot_size	int	每手股数，期货表示合约乘数
stock_name	str	股票名称

字段	类型	说明
stock_type	SecurityType	股票类型
list_time	str	上市时间
stock_id	int	股票 ID
main_contract	bool	是否主连合约
last_trade_time	str	最后交易时间

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_plate_stock('HK.BK1001')
if ret == RET_OK:
    print(data)
    print(data['stock_name'][0])    # 取第一条的股票名称
    print(data['stock_name'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code  lot_size stock_name  stock_owner  stock_child_type stock_type   list_ti
0   HK.00462      4000       天然乳品          NaN               NaN      STOCK
..       ...       ...        ...          ...               ...        ...
9   HK.06186      1000       中国飞鹤          NaN               NaN      STOCK

[10 rows x 10 columns]
天然乳品
['天然乳品', '现代牧业', '雅士利国际', '原生态牧业', '中国圣牧', '中地乳业', '庄园牧
```

接口限制

- 每 30 秒内最多请求 10 次获取板块内股票列表接口

▶ 常用的板块、指数代码

获取板块列表

`get_plate_list(market, plate_class)`

- **介绍**

  获取板块列表

- **参数**

参数	类型	说明
market	Market	市场标识
plate_class	Plate	板块分类

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回板块列表数据
	str	当 ret != RET_OK，返回错误描述

  - 板块列表数据格式如下：

字段	类型	说明
code	str	板块代码
plate_name	str	板块名字
plate_id	str	板块 ID

- **Example**

```python
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_plate_list(Market.HK, Plate.CONCEPT)
if ret == RET_OK:
    print(data)
    print(data['plate_name'][0])    # 取第一条的板块名称
    print(data['plate_name'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code plate_name plate_id
0   HK.BK1000      做空集合股    BK1000
..        ...         ...       ...
77  HK.BK1999       殡葬概念    BK1999

[78 rows x 3 columns]
做空集合股
['做空集合股', '阿里概念股', '雄安概念股', '苹果概念', '一带一路', '5G概念', '夜店股
```

> **接口限制**
>
> - 每 30 秒内最多请求 10 次获取板块列表接口

获取静态数据

`get_stock_basicinfo(market, stock_type=SecurityType.STOCK, code_list=None)`

- **介绍**

  获取静态数据

- **参数**

参数	类型	说明
market	Market	市场类型
stock_type	SecurityType	股票类型，但不支持传入 SecurityType.DRVT
code_list	list	股票列表

  注：当 market 和 code_list 同时存在时，会忽略 market，仅对 code_list 进行查询。

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回股票静态数据
	str	当 ret != RET_OK，返回错误描述

  - 股票静态数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	股票名称

字段	类型	说明
lot_size	int	每手股数，期权表示每份合约股数，期货表示合约乘数
stock_type	SecurityType	股票类型
stock_child_type	WrtType	窝轮子类型
stock_owner	str	窝轮所属正股的代码，或期权标的股的代码
option_type	OptionType	期权类型
strike_time	str	期权行权日
strike_price	float	期权行权价
suspension	bool	期权是否停牌
listing_date	str	上市时间
stock_id	int	股票 ID
delisting	bool	是否退市
index_option_type	str	指数期权类型
main_contract	bool	是否主连合约
last_trade_time	str	最后交易时间
exchange_type	ExchType	所属交易所

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
ret, data = quote_ctx.get_stock_basicinfo(Market.HK, SecurityType.STOCK)
if ret == RET_OK:
    print(data)
else:
```

```
    print('error:', data)
print('****************************************')
ret, data = quote_ctx.get_stock_basicinfo(Market.HK, SecurityType.STOCK, ['HK.069
if ret == RET_OK:
    print(data)
    print(data['name'][0])  # 取第一条的股票名称
    print(data['name'].values.tolist())  # 转为 list
else:
    print('error:', data)
quote_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
         code              name  lot_size stock_type stock_child_type stock_owner c
0      HK.00001              长和       500      STOCK              N/A
...         ...               ...       ...        ...               ...         .
2592   HK.09979      绿城管理控股      1000      STOCK              N/A

[2593 rows x 16 columns]
****************************************
         code            name  lot_size stock_type stock_child_type stock_owner op
0  HK.06998      嘉和生物-B       500      STOCK              N/A
1  HK.00700      腾讯控股         100      STOCK              N/A
嘉和生物-B
['嘉和生物-B', '腾讯控股']
```

提示

- 当传入程序无法识别的股票时（包括很久之前退市的股票和不存在的股票），此接口仍然返回股票信息，用"是否退市"字段来标识该股票不存在。统一处理为：代码正常显示，股票名显示为"未知股票"，其他字段均为默认值（整型默认是0，字符串默认是空字符串）。
- 此接口与其他的行情接口不同，其他接口遇到程序无法识别的股票时，会拒绝请求并返回错误描述"未知股票"。

获取 IPO 信息

`get_ipo_list(market)`

- **介绍**

  获取指定市场的 IPO 信息

- **参数**

参数	类型	说明
market	Market	市场标识

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回 IPO 数据
	str	当 ret != RET_OK，返回错误描述

  - IPO 数据

字段	类型	说明
code	str	股票代码
name	str	股票名称
list_time	str	上市日期，美股是预计上市日期
list_timestamp	float	上市日期时间戳，美股是预计上市日期时间戳
apply_code	str	申购代码（A 股适用）

字段	类型	说明
issue_size	int	发行总数（A 股适用）；发行量（美股适用）
online_issue_size	int	网上发行量（A 股适用）
apply_upper_limit	int	申购上限（A 股适用）
apply_limit_market_value	int	顶格申购需配市值（A 股适用）
is_estimate_ipo_price	bool	是否预估发行价（A 股适用）
ipo_price	float	发行价（A 股适用）
industry_pe_rate	float	行业市盈率（A 股适用）
is_estimate_winning_ratio	bool	是否预估中签率（A 股适用）
winning_ratio	float	中签率（A 股适用）
issue_pe_rate	float	发行市盈率（A 股适用）
apply_time	str	申购日期字符串（A 股适用）
apply_timestamp	float	申购日期时间戳（A 股适用）
winning_time	str	公布中签日期字符串（A 股适用）
winning_timestamp	float	公布中签日期时间戳（A 股适用）
is_has_won	bool	是否已经公布中签号（A 股适用）
winning_num_data	str	中签号（A 股适用）
ipo_price_min	float	最低发售价（港股适用）；最低发行价（美股适用）
ipo_price_max	float	最高发售价（港股适用）；最高发行价（美股适用）
list_price	float	上市价（港股适用）

字段	类型	说明
lot_size	int	每手股数
entrance_price	float	入场费（港股适用）
is_subscribe_status	bool	是否为认购状态
apply_end_time	str	截止认购日期字符串（港股适用）
apply_end_timestamp	float	截止认购日期时间戳

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_ipo_list(Market.HK)
if ret == RET_OK:
    print(data)
    print(data['code'][0])    # 取第一条的股票代码
    print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code      name   list_time  list_timestamp apply_code issue_size online_issue
0  HK.06666  恒大物业  2020-12-02    1.606838e+09        N/A        N/A
1  HK.02110  裕勤控股  2020-12-07    1.607270e+09        N/A        N/A
HK.06666
['HK.06666', 'HK.02110']
```

接口限制

- 每 30 秒内最多请求 10 次获取 IPO 信息接口

获取全局市场状态

`get_global_state()`

- **介绍**

  获取全局状态

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	dict	当 ret == RET_OK 时，返回全局状态
	str	当 ret != RET_OK，返回错误描述

  - 全局状态字典格式如下：

字段	类型	说明
market_sz	MarketState	深圳市场状态
market_sh	MarketState	上海市场状态
market_hk	MarketState	香港市场状态
market_hkfuture	MarketState	香港期货市场状态
market_usfuture	MarketState	美国期货市场状态
market_us	MarketState	美国市场状态
market_sgfuture	MarketState	新加坡期货市场状态
market_jpfuture	MarketState	日本期货市场状态

字段	类型	说明
server_ver	str	OpenD 版本号
trd_logined	bool	True：已登录交易服务器，False：未登录交易服务器
qot_logined	bool	True：已登录行情服务器，False：未登录行情服务器
timestamp	str	当前格林威治时间戳
local_timestamp	float	OpenD 运行机器的当前时间戳
program_status_type	ProgramStatusType	当前状态
program_status_desc	str	额外描述

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
print(quote_ctx.get_global_state())
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- **Output**

```
(0, {'market_sz': 'MORNING', 'market_us': 'AFTER_HOURS_END', 'market_sh': 'MORNI
```

获取交易日历

```
request_trading_days(market=None, start=None, end=None, code=None)
```

- **介绍**

  请求指定市场 / 指定标的的交易日历。
  注意：该交易日是通过自然日剔除周末和节假日得到，未剔除临时休市数据。

- **参数**

参数	类型	说明
market	TradeDateMarket	市场类型
start	str	起始日期
end	str	结束日期
code	str	股票代码

  注：当 market 和 code 同时存在时，会忽略 market，仅对 code 进行查询。

  - start 和 end 的组合如下

Start 类型	End 类型	说明
str	str	start 和 end 分别为指定的日期
None	str	start 为 end 往前 365 天
str	None	end 为 start 往后 365 天
None	None	start 为往前 365 天，end 当前日期

- **返回**

参数	类型	说明

ret	RET_CODE	接口调用结果
data	list	当 ret == RET_OK 时，返回交易日数据。list 中元素类型为 dict
	str	当 ret != RET_OK 时，返回错误描述

- 交易日数据格式如下：

字段	类型	说明
time	str	时间
trade_date_type	TradeDateType	交易日类型

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.request_trading_days(market=TradeDateMarket.HK, start='2020
if ret == RET_OK:
    print('HK market calendar:', data)
else:
    print('error:', data)
print('****************************************')
ret, data = quote_ctx.request_trading_days(start='2020-04-01', end='2020-04-10',
if ret == RET_OK:
    print('HK.00700 calendar:', data)
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- **Output**

```
HK market calendar: [{'time': '2020-04-01', 'trade_date_type': 'WHOLE'}, {'time'
****************************************
HK.00700 calendar: [{'time': '2020-04-01', 'trade_date_type': 'WHOLE'}, {'time':
```

接口限制

- 每 30 秒内最多请求 30 次获取交易日接口。
- 历史交易日历提供过去 10 年的数据，未来交易日历提供到今年 12 月 31 日。

获取历史 K 线额度使用明细

`get_history_kl_quota(get_detail=False)`

- **介绍**

  获取历史 K 线额度使用明细

- **参数**

参数	类型	说明
get_detail	bool	是否返回拉取历史 K 线的详细纪录

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	tuple	当 ret == RET_OK，返回历史 K 线额度数据
	str	当 ret != RET_OK，返回错误描述

  - 历史 K 线额度数据格式如下：

字段	类型	说明
used_quota	int	已用额度
remain_quota	int	剩余额度
detail_list	list	拉取历史 K 线的详细纪录，含股票代码和拉取时间

    - detail_list 数据列格式如下

字段	类型	说明
code	str	股票代码
name	str	股票名称
request_time	str	最后一次拉取的时间字符串

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_history_kl_quota(get_detail=True)  # 设置 true 代表需要
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
(2, 98, {'code': 'HK.00123', 'name': '越秀地产', 'request_time': '2023-06-20 19:5
```

接口限制

- 我们会根据您账户的资产和交易的情况，下发历史 K 线额度。因此，30 天内您只能获取有限只股票的历史 K 线数据。具体规则参见 订阅额度 & 历史 K 线额度。
- 您当日消耗的历史 K 线额度，会在 30 天后自动释放。

设置到价提醒

`set_price_reminder(code, op, key=None, reminder_type=None, reminder_freq=None, value=None, note=None)`

- **介绍**

  新增、删除、修改、启用、禁用指定股票的到价提醒

- **参数**

参数	类型	说明
code	str	股票代码
op	SetPriceReminderOp	操作类型
key	int	标识，新增和删除全部的情况不需要填
reminder_type	PriceReminderType	到价提醒的类型，删除、启用、禁用的情况下会忽略该入参
reminder_freq	PriceReminderFreq	到价提醒的频率，删除、启用、禁用的情况下会忽略该入参
value	float	提醒值，删除、启用、禁用的情况下会忽略该入参
note	str	用户设置的备注，仅支持 20 个以内的中文字符，删除、启用、禁用的情况下会忽略该入参
reminder_session_list	list	美股到价提醒的时段列表，删除、启用、禁用的情况下会忽略该入参 

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
key	int	当 ret == RET_OK 时，返回操作的到价提醒 key
	str	当 ret != RET_OK，返回错误描述

- Example

```
from moomoo import *
import time
class PriceReminderTest(PriceReminderHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, content = super(PriceReminderTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("PriceReminderTest: error, msg: %s" % content)
            return RET_ERROR, content
        print("PriceReminderTest ", content) # PriceReminderTest 自己的处理逻辑
        return RET_OK, content
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = PriceReminderTest()
quote_ctx.set_handler(handler)
ret, data = quote_ctx.get_market_snapshot(['US.AAPL'])
if ret == RET_OK:
    bid_price = data['bid_price'][0]  # 获取实时买一价
    ask_price = data['ask_price'][0]  # 获取实时卖一价
    # 设置当AAPL全时段卖一价低于（ask_price-1）时提醒
    ret_ask, ask_data = quote_ctx.set_price_reminder(code='US.AAPL', op=SetPriceR
    if ret_ask == RET_OK:
        print('卖一价低于（ask_price-1）时提醒设置成功：', ask_data)
    else:
        print('error:', ask_data)
    # 设置当AAPL全时段买一价高于（bid_price+1）时提醒
    ret_bid, bid_data = quote_ctx.set_price_reminder(code='US.AAPL', op=SetPriceR
    if ret_bid == RET_OK:
        print('买一价高于（bid_price+1）时提醒设置成功：', bid_data)
    else:
        print('error:', bid_data)
time.sleep(15)
quote_ctx.close()
```

- Output

```
卖一价低于（ask_price-1）时提醒设置成功： 1744022257023211123
买一价高于（bid_price+1）时提醒设置成功： 1744022257052794489
```

提示

- API 中成交量设置统一以股为单位。但是 moomoo 客户端中，A 股是以手为单位展示
- 到价提醒类型，存在最小精度，如下：

  TURNOVER_UP：成交额最小精度为 10 元（人民币元，港元，美元）。传入的数值会自动向下取整到最小精度的整数倍。如果设置【00700成交额102元提醒】，设置后会得到【00700成交额100元提醒】；如果设置【00700 成交额 8 元提醒】，设置后会得到【00700 成交额 0 元提醒】。

  VOLUME_UP：A 股成交量最小精度为 1000 股，其他市场股票成交量最小精度为 10 股。传入的数值会自动向下取整到最小精度的整数倍。

  BID_VOL_UP、ASK_VOL_UP：A 股的买一卖一量最小精度为 100 股。传入的数值会自动向下取整到最小精度的整数倍。

  其余到价提醒类型精度支持到小数点后 3 位

接口限制

- 每 30 秒内最多请求 60 次设置到价提醒接口
- 每只股票每种类型可设置的提醒上限是 10 个

获取到价提醒列表

`get_price_reminder(code=None, market=None)`

- **介绍**

  获取对指定股票 / 指定市场设置的到价提醒列表

- **参数**

参数	类型	说明
code	str	股票代码
market	Market	市场类型

  注：code 和 market 都存在的情况下，code 优先。

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回到价提醒数据
	str	当 ret != RET_OK，返回错误描述

  - 到价提醒数据格式如下：

字段	类型	说明
code	str	股票代码
key	int	标识，用于修改到价提醒
reminder_type	PriceReminderType	到价提醒的类型

字段	类型	说明
reminder_freq	PriceReminderFreq	到价提醒的频率
value	float	提醒值
enable	bool	是否启用
note	str	备注
reminder_session_list	list	美股到价提醒时段列表

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_price_reminder(code='US.AAPL')
if ret == RET_OK:
    print(data)
    print(data['key'].values.tolist())   # 转为 list
else:
    print('error:', data)
print('**************************************')
ret, data = quote_ctx.get_price_reminder(code=None, market=Market.US)
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果到价提醒列表不为空
        print(data['code'][0])    # 取第一条的股票代码
        print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
code name                 key   reminder_type reminder_freq   value  enable note
0  US.AAPL   苹果  1744021708234288125    BID_PRICE_UP        ALWAYS  184.37    T
1  US.AAPL   苹果  1744022257052794489    BID_PRICE_UP        ALWAYS  185.50    T
2  US.AAPL   苹果  1744021708211891867  ASK_PRICE_DOWN        ALWAYS  182.54    T
3  US.AAPL   苹果  1744022257023211123  ASK_PRICE_DOWN        ALWAYS  183.70    T
```

```
[1744021708234288125, 1744022257052794489, 1744021708211891867, 1744022257023211
****************************************
      code name                 key    reminder_type reminder_freq   value  enabl
0  US.AAPL   苹果  1744021708234288125     BID_PRICE_UP        ALWAYS  184.37     T
1  US.AAPL   苹果  1744022257052794489     BID_PRICE_UP        ALWAYS  185.50     T
2  US.AAPL   苹果  1744021708211891867   ASK_PRICE_DOWN        ALWAYS  182.54     T
3  US.AAPL   苹果  1744022257023211123   ASK_PRICE_DOWN        ALWAYS  183.70     T
4  US.NVDA  英伟达  1739697581665326308       PRICE_DOWN         ALWAYS  102.00
US.AAPL
['US.AAPL', 'US.AAPL', 'US.AAPL', 'US.AAPL', 'US.NVDA']
```

接口限制

- 每 30 秒内最多请求 10 次获取到价提醒列表接口

获取自选股列表

`get_user_security(group_name)`

- **介绍**

  获取指定分组的自选股列表

- **参数**

参数	类型	说明
group_name	str	需要查询的自选股分组名称

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回自选股数据
	str	当 ret != RET_OK，返回错误描述

  - 自选股数据格式如下：

字段	类型	说明
code	str	股票代码
name	str	名字
lot_size	int	每手股数，期权表示每份合约股数，期货表示合约乘数
stock_type	SecurityType	股票类型
stock_child_type	WrtType	窝轮子类型

字段	类型	说明
stock_owner	str	窝轮所属正股的代码，或期权标的股的代码
option_type	OptionType	期权类型
strike_time	str	期权行权日
strike_price	float	期权行权价
suspension	bool	期权是否停牌
listing_date	str	上市时间
stock_id	int	股票 ID
delisting	bool	是否退市
main_contract	bool	是否主连合约
last_trade_time	str	最后交易时间

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_user_security("A")
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果自选股列表不为空
        print(data['code'][0])    # 取第一条的股票代码
        print(data['code'].values.tolist())   # 转为 list
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
    code    name  lot_size stock_type stock_child_type stock_owner option_type st
0  HK.HSImain  恒指期货主连        50     FUTURE              N/A
1  HK.00700    腾讯控股       100      STOCK             N/A
HK.HSImain
['HK.HSImain', 'HK.00700']
```

提示

系统分组的中英文对应名称如下

中文	英文
全部	All
沪深	CN
港股	HK
美股	US
期权	Options
港股期权	HK options
美股期权	US options
特别关注	Starred
期货	Futures

接口限制

- 每 30 秒内最多请求 10 次获取自选股列表接口
- 不支持持仓（Positions），基金宝（Mutual Fund），外汇（Forex）分组的查询

获取自选股分组

`get_user_security_group(group_type = UserSecurityGroupType.ALL)`

- **介绍**

  获取自选股分组列表

- **参数**

参数	类型	说明
group_type	UserSecurityGroupType	分组类型

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK，返回自选股分组数据
	str	当 ret != RET_OK，返回错误描述

  - 自选股分组数据格式如下：

字段	类型	说明
group_name	str	分组名
group_type	UserSecurityGroupType	分组类型

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.get_user_security_group(group_type = UserSecurityGroupType
```

```python
if ret == RET_OK:
    print(data)
else:
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
        group_name group_type
0           期权     SYSTEM
..          ...        ...
12           C      CUSTOM

[13 rows x 2 columns]
```

> **接口限制**
>
> - 每 30 秒内最多请求 10 次获取自选股分组接口

修改自选股列表

`modify_user_security(group_name, op, code_list)`

- **介绍**

  修改指定分组的自选股列表（系统分组不支持修改）

- **参数**

参数	类型	说明
group_name	str	需要修改的自选股分组名称
op	ModifyUserSecurityOp	操作类型
code_list	list	股票列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
msg	str	当 ret == RET_OK，返回“success”
		当 ret != RET_OK，msg 返回错误描述

- **Example**

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)

ret, data = quote_ctx.modify_user_security("A", ModifyUserSecurityOp.ADD, ['HK.00
if ret == RET_OK:
    print(data) # 返回 success
else:
```

```
    print('error:', data)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
success
```

接口限制

- 每 30 秒内最多请求 10 次修改自选股列表接口
- 仅支持修改自定义分组，不支持修改系统分组
- “全部”自选股列表的数量存在上限：无交易客户 500 个，有交易客户 2000 个（向其他分组增加自选股时，“全部”列表中也会同步增加）
- 如果有同名的分组，会操作排序在第一个的分组

到价提醒回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  到价提醒通知回调，异步处理已设置到价提醒的通知推送。
  在收到实时到价提醒通知推送后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Qot_UpdatePriceReminder_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	dict	当 ret == RET_OK，返回到价提醒
	str	当 ret != RET_OK，返回错误描述

  - 到价提醒

字段	类型	说明
code	str	股票代码
name	str	股票名称
price	float	当前价格
change_rate	str	当前涨跌幅

字段	类型	说明
market_status	PriceReminderMarketStatus	触发的时间段
content	str	到价提醒文字内容
note	str	备注 
key	int	到价提醒标识
reminder_type	PriceReminderType	到价提醒的类型
set_value	float	用户设置的提醒值
cur_value	float	提醒触发时的值

- Example

```
import time
from moomoo import *

class PriceReminderTest(PriceReminderHandlerBase):
    def on_recv_rsp(self, rsp_pb):
        ret_code, content = super(PriceReminderTest,self).on_recv_rsp(rsp_pb)
        if ret_code != RET_OK:
            print("PriceReminderTest: error, msg: %s" % content)
            return RET_ERROR, content
        print("PriceReminderTest ", content) # PriceReminderTest 自己的处理逻辑
        return RET_OK, content
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = PriceReminderTest()
quote_ctx.set_handler(handler)  # 设置到价提醒通知回调
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()   # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

- Output

```
PriceReminderTest  {'code': 'US.AAPL', 'name': '苹果', 'price': 185.750, 'change_
```

提示

- 此接口提供了持续获取推送数据的功能，如需一次性获取实时数据，请参考 获取到价提醒 接口
- 获取实时数据 和 实时数据回调 的差别，请参考 如何通过订阅接口获取实时行情?

行情定义

累积过滤属性

> StockField

- `NONE`

  未知

- `CHANGE_RATE`

  涨跌幅 

- `AMPLITUDE`

  振幅 

- `VOLUME`

  日均成交量 

- `TURNOVER`

  日均成交额 

- `TURNOVER_RATE`

  换手率 

资产类别

> AssetClass

- `UNKNOW`

  未知

- `STOCK`

  股票
- `BOND`

  债券
- `COMMODITY`

  商品
- `CURRENCY_MARKET`

  货币市场
- `FUTURE`

  期货
- `SWAP`

  掉期（互换）

公司行动

暗盘状态

> DarkStatus

- `NONE`

  无暗盘交易
- `TRADING`

  暗盘交易中
- `END`

  暗盘交易结束

财务过滤属性

> StockField

- `NONE`

  未知

- `NET_PROFIT`

  净利润 

- `NET_PROFIX_GROWTH`

  净利润增长率 

- `SUM_OF_BUSINESS`

  营业收入 

- `SUM_OF_BUSINESS_GROWTH`

  营收同比增长率 

- `NET_PROFIT_RATE`

  净利率 

- `GROSS_PROFIT_RATE`

  毛利率 

- `DEBT_ASSET_RATE`

  资产负债率 

- `RETURN_ON_EQUITY_RATE`

  净资产收益率 

- `ROIC`

投入资本回报率

- `ROA_TTM`

  资产回报率 TTM

- `EBIT_TTM`

  息税前利润 TTM

- `EBITDA`

  税息折旧及摊销前利润

- `OPERATING_MARGIN_TTM`

  营业利润率 TTM

- `EBIT_MARGIN`

  EBIT 利润率

- `EBITDA_MARGIN`

  EBITDA 利润率

- `FINANCIAL_COST_RATE`

  财务成本率

- `OPERATING_PROFIT_TTM`

  营业利润 TTM

- `SHAREHOLDER_NET_PROFIT_TTM`

  归属于母公司的净利润

- `NET_PROFIT_CASH_COVER_TTM`

  盈利中的现金收入比例

- `CURRENT_RATIO`

  流动比率

- `QUICK_RATIO`

  速动比率 
- `CURRENT_ASSET_RATIO`

  流动资产率 
- `CURRENT_DEBT_RATIO`

  流动负债率 
- `EQUITY_MULTIPLIER`

  权益乘数 
- `PROPERTY_RATIO`

  产权比率 
- `CASH_AND_CASH_EQUIVALENTS`

  现金和现金等价物 
- `TOTAL_ASSET_TURNOVER`

  总资产周转率 
- `FIXED_ASSET_TURNOVER`

  固定资产周转率 
- `INVENTORY_TURNOVER`

  存货周转率 
- `OPERATING_CASH_FLOW_TTM`

  经营活动现金流 TTM 
- `ACCOUNTS_RECEIVABLE`

  应收账款净额 
- `EBIT_GROWTH_RATE`

EBIT 同比增长率 

- `OPERATING_PROFIT_GROWTH_RATE`

  营业利润同比增长率 

- `TOTAL_ASSETS_GROWTH_RATE`

  总资产同比增长率 

- `PROFIT_TO_SHAREHOLDERS_GROWTH_RATE`

  归母净利润同比增长率 

- `PROFIT_BEFORE_TAX_GROWTH_RATE`

  总利润同比增长率 

- `EPS_GROWTH_RATE`

  EPS 同比增长率 

- `ROE_GROWTH_RATE`

  ROE 同比增长率 

- `ROIC_GROWTH_RATE`

  ROIC 同比增长率 

- `NOCF_GROWTH_RATE`

  经营现金流同比增长率 

- `NOCF_PER_SHARE_GROWTH_RATE`

  每股经营现金流同比增长率 

- `OPERATING_REVENUE_CASH_COVER`

  经营现金收入比 

- `OPERATING_PROFIT_TO_TOTAL_PROFIT`

  营业利润占比 

- `BASIC_EPS`

  基本每股收益 
- `DILUTED_EPS`

  稀释每股收益 
- `NOCF_PER_SHARE`

  每股经营现金净流量 

财务过滤属性周期

> FinancialQuarter

- `NONE`

  未知
- `ANNUAL`

  年报
- `FIRST_QUARTER`

  一季报
- `INTERIM`

  中报
- `THIRD_QUARTER`

  三季报
- `MOST_RECENT_QUARTER`

  最近季报

自定义技术指标属性

> StockField

- `NONE`

  未知

- `PRICE`

  最新价格

- `MA`

  简单均线

- `MA5`

  5日简单均线（不建议使用）

- `MA10`

  10日简单均线（不建议使用）

- `MA20`

  20日简单均线（不建议使用）

- `MA30`

  30日简单均线（不建议使用）

- `MA60`

  60日简单均线（不建议使用）

- `MA120`

  120日简单均线（不建议使用）

- `MA250`

  250日简单均线（不建议使用）

- `RSI`

  RSI 

- `EMA`

  指数移动均线

- `EMA5`

  5日指数移动均线（不建议使用）

- `EMA10`

  10日指数移动均线（不建议使用）

- `EMA20`

  20日指数移动均线（不建议使用）

- `EMA30`

  30日指数移动均线（不建议使用）

- `EMA60`

  60日指数移动均线（不建议使用）

- `EMA120`

  120日指数移动均线（不建议使用）

- `EMA250`

  250日指数移动均线（不建议使用）

- `KDJ_K`

  KDJ 指标的 K 值 

- `KDJ_D`

  KDJ 指标的 D 值 

- `KDJ_J`

  KDJ 指标的 J 值 

- `MACD_DIFF`

MACD 指标的 DIFF 值

- `MACD_DEA`

  MACD 指标的 DEA 值

- `MACD`

  MACD

- `BOLL_UPPER`

  BOLL 指标的 UPPER 值

- `BOLL_MIDDLER`

  BOLL 指标的 MIDDLER 值

- `BOLL_LOWER`

  BOLL 指标的 LOWER 值

- `VALUE`

  自定义数值（stock_field1 不支持此字段）

相对位置

> RelativePosition

- `NONE`

  未知

- `MORE`

  大于，stock_field1 位于stock_field2 的上方

- `LESS`

  小于，stock_field1 位于stock_field2 的下方

- `CROSS_UP`

升穿，stock_field1 从下往上穿stock_field2

- `CROSS_DOWN`

  跌穿，stock_field1 从上往下穿stock_field2

形态技术指标属性

> PatternField

- `NONE`

  未知

- `MA_ALIGNMENT_LONG`

  MA多头排列（连续两天MA5>MA10>MA20>MA30>MA60，且当日收盘价大于前一天收盘价）

- `MA_ALIGNMENT_SHORT`

  MA空头排列（连续两天MA5<MA10<MA20<MA30<MA60，且当日收盘价小于前一天收盘价）

- `EMA_ALIGNMENT_LONG`

  EMA多头排列（连续两天EMA5>EMA10>EMA20>EMA30>EMA60，且当日收盘价大于前一天收盘价）

- `EMA_ALIGNMENT_SHORT`

  EMA空头排列（连续两天EMA5<EMA10<EMA20<EMA30<EMA60，且当日收盘价小于前一天收盘价）

- `RSI_GOLD_CROSS_LOW`

  RSI低位金叉（50以下，短线RSI上穿长线RSI（前一日短线RSI小于长线RSI，当日短线RSI大于长线RSI））

- `RSI_DEATH_CROSS_HIGH`

  RSI高位死叉（50以上，短线RSI下穿长线RSI（前一日短线RSI大于长线RSI，当日短线RSI小于长线RSI））

- `RSI_TOP_DIVERGENCE`

  RSI顶背离（相邻的两个K线波峰，后面的波峰对应的CLOSE>前面的波峰对应的CLOSE，后面波峰的RSI12值<前面波峰的RSI12值）

- `RSI_BOTTOM_DIVERGENCE`

  RSI底背离（相邻的两个K线波谷，后面的波谷对应的CLOSE<前面的波谷对应的CLOSE，后面波谷的RSI12值>前面波谷的RSI12值）

- `KDJ_GOLD_CROSS_LOW`

  KDJ低位金叉（D值小于或等于30，且前一日K值小于D值，当日K值大于D值）

- `KDJ_DEATH_CROSS_HIGH`

  KDJ高位死叉（D值大于或等于70，且前一日K值大于D值，当日K值小于D值）

- `KDJ_TOP_DIVERGENCE`

  KDJ顶背离（相邻的两个K线波峰，后面的波峰对应的CLOSE>前面的波峰对应的CLOSE，后面波峰的J值<前面波峰的J值）

- `KDJ_BOTTOM_DIVERGENCE`

  KDJ底背离（相邻的两个K线波谷，后面的波谷对应的CLOSE<前面的波谷对应的CLOSE，后面波谷的J值>前面波谷的J值）

- `MACD_GOLD_CROSS_LOW`

  MACD低位金叉（DIFF上穿DEA（前一日DIFF小于DEA，当日DIFF大于DEA））

- `MACD_DEATH_CROSS_HIGH`

  MACD高位死叉（DIFF下穿DEA（前一日DIFF大于DEA，当日DIFF小于DEA））

- `MACD_TOP_DIVERGENCE`

  MACD顶背离（相邻的两个K线波峰，后面的波峰对应的CLOSE>前面的波峰对应的CLOSE，后面波峰的macd值<前面波峰的macd值）

- `MACD_BOTTOM_DIVERGENCE`

MACD底背离（相邻的两个K线波谷，后面的波谷对应的CLOSE<前面的波谷对应的CLOSE，后面波谷的macd值>前面波谷的macd值）

- `BOLL_BREAK_UPPER`

  BOLL突破上轨（前一日股价低于上轨值，当日股价大于上轨值）

- `BOLL_BREAK_LOWER`

  BOLL突破下轨（前一日股价高于下轨值，当日股价小于下轨值）

- `BOLL_CROSS_MIDDLE_UP`

  BOLL向上破中轨（前一日股价低于中轨值，当日股价大于中轨值）

- `BOLL_CROSS_MIDDLE_DOWN`

  BOLL向下破中轨（前一日股价大于中轨值，当日股价小于中轨值）

自选股分组类型

> UserSecurityGroupType

- `NONE`

  未知

- `CUSTOM`

  自定义分组

- `SYSTEM`

  系统分组

- `ALL`

  全部分组

指数期权类别

> IndexOptionType

- `NONE`

  未知

- `NORMAL`

  普通的指数期权

- `SMALL`

  小型指数期权

上市时段

> IpoPeriod

- `NONE`

  未知

- `TODAY`

  今日上市

- `TOMORROW`

  明日上市

- `NEXTWEEK`

  未来一周上市

- `LASTWEEK`

  过去一周上市

- `LASTMONTH`

  过去一月上市

窝轮发行商

> Issuer

- `UNKNOW`

  未知
- `SG`

  法兴
- `BP`

  法巴
- `CS`

  瑞信
- `CT`

  花旗
- `EA`

  东亚
- `GS`

  高盛
- `HS`

  汇丰
- `JP`

  摩通
- `MB`

  麦银
- `SC`

  渣打

- `UB`

  瑞银
- `BI`

  中银
- `DB`

  德银
- `DC`

  大和
- `ML`

  美林
- `NM`

  野村
- `RB`

  荷合
- `RS`

  苏皇
- `BC`

  巴克莱
- `HT`

  海通
- `VT`

  瑞通
- `KC`

比联

- `MS`

  摩利

- `GJ`

  国君

- `XZ`

  星展

- `HU`

  华泰

- `KS`

  韩投

- `CI`

  信证

K 线字段

> KL_FIELD

- `ALL`

  所有

- `DATE_TIME`

  时间

- `HIGH`

  最高价

- `OPEN`

开盘价

- `LOW`

  最低价

- `CLOSE`

  收盘价

- `LAST_CLOSE`

  昨收价

- `TRADE_VOL`

  成交量

- `TRADE_VAL`

  成交额

- `TURNOVER_RATE`

  换手率

- `PE_RATIO`

  市盈率

- `CHANGE_RATE`

  涨跌幅

K 线类型

> KLType

- `NONE`

  未知

- `K_1M`

1分 K

- `K_DAY`

  日 K

- `K_WEEK`

  周 K 

- `K_MON`

  月 K 

- `K_YEAR`

  年 K 

- `K_5M`

  5分 K

- `K_15M`

  15分 K

- `K_30M`

  30分 K 

- `K_60M`

  60分 K

- `K_3M`

  3分 K

- `K_QUARTER`

  季 K 

周期类型

> PeriodType

- `INTRADAY`

  实时

- `DAY`

  日

- `WEEK`

  周

- `MONTH`

  月

到价提醒市场状态

> PriceReminderMarketStatus

- `NONE`

  未知

- `OPEN`

  盘中

- `US_PRE`

  美股盘前

- `US_AFTER`

  美股盘后

- `US_OVERNIGHT`

  美股夜盘

自选股操作

> ModifyUserSecurityOp

- `NONE`

  未知

- `ADD`

  新增

- `DEL`

  删除自选

- `MOVE_OUT`

  移出分组

期权类型（按行权时间）

> OptionAreaType

- `NONE`

  未知

- `AMERICAN`

  美式

- `EUROPEAN`

  欧式

- `BERMUDA`

  百慕大

期权价内/外

> OptionCondType

- `ALL`

  所有

- `WITHIN`

  价内

- `OUTSIDE`

  价外

期权类型（按方向）

> OptionType

- `ALL`

  所有

- `CALL`

  看涨期权

- `PUT`

  看跌期权

板块集合类型

> Plate

- `ALL`

所有板块

- `INDUSTRY`

  行业板块

- `REGION`

  地域板块 

- `CONCEPT`

  概念板块

- `OTHER`

  其他板块 

到价提醒频率

> PriceReminderFreq

- `NONE`

  未知

- `ALWAYS`

  持续提醒

- `ONCE_A_DAY`

  每日一次

- `ONCE`

  仅提醒一次

到价提醒类型

> PriceReminderType

- `NONE`

  未知

- `PRICE_UP`

  价格涨到

- `PRICE_DOWN`

  价格跌到

- `CHANGE_RATE_UP`

  日涨幅超 

- `CHANGE_RATE_DOWN`

  日跌幅超 

- `FIVE_MIN_CHANGE_RATE_UP`

  5 分钟涨幅超 

- `FIVE_MIN_CHANGE_RATE_DOWN`

  5 分钟跌幅超 

- `VOLUME_UP`

  成交量超过

- `TURNOVER_UP`

  成交额超过

- `TURNOVER_RATE_UP`

  换手率超过 

- `BID_PRICE_UP`

  买一价高于

- `ASK_PRICE_DOWN`

卖一价低于

- `BID_VOL_UP`

  买一量高于

- `ASK_VOL_UP`

  卖一量高于

- `THREE_MIN_CHANGE_RATE_UP`

  3 分钟涨幅超 

- `THREE_MIN_CHANGE_RATE_DOWN`

  3 分钟跌幅超 

窝轮价内/外

> PriceType

- `UNKNOW`

  未知

- `OUTSIDE`

  价外，界内证表示界外

- `WITH_IN`

  价内，界内证表示界内

逐笔推送类型

> PushDataType

- `UNKNOW`

  未知

- `REALTIME`

  实时推送的数据

- `BYDISCONN`

  与富途服务器连接断开期间，拉取补充的数据

- `CACHE`

  非实时非连接断开补充数据

行情市场

> Market

- `NONE`

  未知市场

- `HK`

  香港市场

- `US`

  美国市场

- `SH`

  沪股市场

- `SZ`

  深股市场

- `SG`

  新加坡市场

- `JP`

  日本市场

- `AU`

  澳大利亚市场
- `CA`

  加拿大市场
- `MY`

  马来西亚市场
- `FX`

  外汇市场

市场状态

> MarketState

各市场状态的对应时段：点击这里了解更多

- `NONE`

  无交易
- `AUCTION`

  盘前竞价
- `WAITING_OPEN`

  等待开盘
- `MORNING`

  早盘
- `REST`

  午间休市
- `AFTERNOON`

午盘 / 美股持续交易时段

- `CLOSED`

  收盘

- `PRE_MARKET_BEGIN`

  美股盘前交易时段

- `PRE_MARKET_END`

  美股盘前交易结束

- `AFTER_HOURS_BEGIN`

  美股盘后交易时段

- `AFTER_HOURS_END`

  美股盘后结束

- `OVERNIGHT`

  美股夜盘交易时段

- `NIGHT_OPEN`

  夜市交易时段

- `NIGHT_END`

  夜市收盘

- `NIGHT`

  美指期权夜市交易时段

- `TRADE_AT_LAST`

  美指期权盘尾交易时段

- `FUTURE_DAY_OPEN`

  日市交易时段

- `FUTURE_DAY_BREAK`

  日市休市
- `FUTURE_DAY_CLOSE`

  日市收盘
- `FUTURE_DAY_WAIT_OPEN`

  期货待开盘
- `HK_CAS`

  港股盘后竞价
- `FUTURE_NIGHT_WAIT`

  夜市等待开盘（已废弃）
- `FUTURE_AFTERNOON`

  期货下午开盘（已废弃）
- `FUTURE_SWITCH_DATE`

  美期待开盘
- `FUTURE_OPEN`

  美期交易时段
- `FUTURE_BREAK`

  美期中盘休息
- `FUTURE_BREAK_OVER`

  美期休息后交易时段
- `FUTURE_CLOSE`

  美期收盘
- `STIB_AFTER_HOURS_WAIT`

科创板的盘后撮合时段（已废弃）

- `STIB_AFTER_HOURS_BEGIN`

  科创板的盘后交易开始（已废弃）

- `STIB_AFTER_HOURS_END`

  科创板的盘后交易结束（已废弃）

美股时段

> Session

- `NONE`

  未知

- `RTH`

  美股盘中时段

- `ETH`

  美股盘中+盘前盘后

- `OVERNIGHT`

  美股夜盘时段 (仅用于交易接口)

- `ALL`

  美股全时段 (用于行情&交易接口)

行情权限

> QotRight

- `UNKNOW`

  未知

- `BMP`

  BMP（此权限不支持订阅）

- `LEVEL1`

  Level1

- `LEVEL2`

  Level2

- `SF`

  港股 SF 高级全盘行情

- `NO`

  无权限

关联数据类型

> SecurityReferenceType

- `UNKNOW`

  未知

- `WARRANT`

  正股相关的窝轮

- `FUTURE`

  期货主连的相关合约

K 线复权类型

> AuType

- `NONE`

不复权

- `QFQ`

  前复权

- `HFQ`

  后复权

股票状态

> SecurityStatus

- `NONE`

  未知

- `NORMAL`

  正常状态

- `LISTING`

  待上市

- `PURCHASING`

  申购中

- `SUBSCRIBING`

  认购中

- `BEFORE_DRAK_TRADE_OPENING`

  暗盘开盘前

- `DRAK_TRADING`

  暗盘交易中

- `DRAK_TRADE_END`

暗盘已收盘

- `TO_BE_OPEN`

  待开盘

- `SUSPENDED`

  停牌

- `CALLED`

  已收回

- `EXPIRED_LAST_TRADING_DATE`

  已过最后交易日

- `EXPIRED`

  已过期

- `DELISTED`

  已退市

- `CHANGE_TO_TEMPORARY_CODE`

  公司行动中，交易关闭，转至临时代码交易

- `TEMPORARY_CODE_TRADE_END`

  临时买卖结束，交易关闭

- `CHANGED_PLATE_TRADE_END`

  已转板，旧代码交易关闭

- `CHANGED_CODE_TRADE_END`

  已换代码，旧代码交易关闭

- `RECOVERABLE_CIRCUIT_BREAKER`

  可恢复性熔断

- `UN_RECOVERABLE_CIRCUIT_BREAKER`

  不可恢复性熔断
- `AFTER_COMBINATION`

  盘后撮合
- `AFTER_TRANSATION`

  盘后交易

股票类型

> SecurityType

- `NONE`

  未知
- `BOND`

  债券
- `BWRT`

  一揽子权证
- `STOCK`

  正股
- `ETF`

  信托,基金
- `WARRANT`

  窝轮
- `IDX`

  指数

- `PLATE`

  板块
- `DRVT`

  期权
- `PLATESET`

  板块集
- `FUTURE`

  期货

设置到价提醒操作类型

> SetPriceReminderOp

- `NONE`

  未知
- `ADD`

  新增
- `DEL`

  删除
- `ENABLE`

  启用
- `DISABLE`

  禁用
- `MODIFY`

  修改

- `DEL_ALL`

  删除全部（删除指定股票下的所有到价提醒）

排序方向

> SortDir

- `NONE`

  不排序

- `ASCEND`

  升序

- `DESCEND`

  降序

排序字段

> SortField

- `NONE`

  未知

- `CODE`

  代码

- `CUR_PRICE`

  最新价

- `PRICE_CHANGE_VAL`

  涨跌额

- `CHANGE_RATE`

涨跌幅 %

- `STATUS`

  状态

- `BID_PRICE`

  买入价

- `ASK_PRICE`

  卖出价

- `BID_VOL`

  买量

- `ASK_VOL`

  卖量

- `VOLUME`

  成交量

- `TURNOVER`

  成交额

- `AMPLITUDE`

  振幅 %

- `SCORE`

  综合评分

- `PREMIUM`

  溢价 %

- `EFFECTIVE_LEVERAGE`

  有效杠杆

- `DELTA`

  对冲值 
- `IMPLIED_VOLATILITY`

  引伸波幅 
- `TYPE`

  类型
- `STRIKE_PRICE`

  行权价
- `BREAK_EVEN_POINT`

  打和点
- `MATURITY_TIME`

  到期日
- `LIST_TIME`

  上市日期
- `LAST_TRADE_TIME`

  最后交易日
- `LEVERAGE`

  杠杆比率
- `IN_OUT_MONEY`

  价内/价外 %
- `RECOVERY_PRICE`

  收回价 
- `CHANGE_PRICE`

换股价

- `CHANGE`

  换股比率

- `STREET_RATE`

  街货比 %

- `STREET_VOL`

  街货量

- `WARRANT_NAME`

  窝轮名称

- `ISSUER`

  发行人

- `LOT_SIZE`

  每手

- `ISSUE_SIZE`

  发行量

- `UPPER_STRIKE_PRICE`

  上限价 

- `LOWER_STRIKE_PRICE`

  下限价 

- `INLINE_PRICE_STATUS`

  界内界外 

- `PRE_CUR_PRICE`

  盘前最新价

- `AFTER_CUR_PRICE`

  盘后最新价
- `PRE_PRICE_CHANGE_VAL`

  盘前涨跌额
- `AFTER_PRICE_CHANGE_VAL`

  盘后涨跌额
- `PRE_CHANGE_RATE`

  盘前涨跌幅 %
- `AFTER_CHANGE_RATE`

  盘后涨跌幅 %
- `PRE_AMPLITUDE`

  盘前振幅 %
- `AFTER_AMPLITUDE`

  盘后振幅 %
- `PRE_TURNOVER`

  盘前成交额
- `AFTER_TURNOVER`

  盘后成交额
- `LAST_SETTLE_PRICE`

  昨结
- `POSITION`

  持仓量
- `POSITION_CHANGE`

日增仓

简单过滤属性

> StockField

- `NONE`

  未知

- `STOCK_CODE`

  股票代码，不能填区间上下限值。

- `STOCK_NAME`

  股票名称，不能填区间上下限值。

- `CUR_PRICE`

  最新价 

- `CUR_PRICE_TO_HIGHEST52_WEEKS_RATIO`

  (CP - WH52) / WH52
  CP：现价
  WH52：52 周最高
  对应 PC 端“离 52 周高点百分比” 

- `CUR_PRICE_TO_LOWEST52_WEEKS_RATIO`

  (CP - WL52) / WL52
  CP：现价
  WL52：52 周最低
  对应 PC 端“离 52 周低点百分比” 

- `HIGH_PRICE_TO_HIGHEST52_WEEKS_RATIO`

  (TH - WH52) / WH52
  TH：今日最高
  WH52：52 周最高
  

- `LOW_PRICE_TO_LOWEST52_WEEKS_RATIO`

  (TL - WL52) / WL52
  TL：今日最低
  WL52：52 周最低
  

- `VOLUME_RATIO`

  量比

- `BID_ASK_RATIO`

  委比

- `LOT_PRICE`

  每手价格

- `MARKET_VAL`

  市值

- `PE_ANNUAL`

  市盈率(静态)

- `PE_TTM`

  市盈率 TTM

- `PB_RATE`

  市净率

- `CHANGE_RATE_5MIN`

  五分钟价格涨跌幅

- `CHANGE_RATE_BEGIN_YEAR`

  年初至今价格涨跌幅

- `PS_TTM`

市销率 TTM

- `PCF_TTM`

  市现率 TTM

- `TOTAL_SHARE`

  总股数

- `FLOAT_SHARE`

  流通股数

- `FLOAT_MARKET_VAL`

  流通市值

订阅类型

> SubType

- `NONE`

  未知

- `QUOTE`

  基础报价

- `ORDER_BOOK`

  摆盘

- `TICKER`

  逐笔

- `RT_DATA`

  分时

- `K_DAY`

日 K

- `K_5M`

  5 分 K

- `K_15M`

  15 分 K

- `K_30M`

  30 分 K

- `K_60M`

  60 分 K

- `K_1M`

  1 分 K

- `K_WEEK`

  周 K

- `K_MON`

  月 K

- `BROKER`

  经纪队列

- `K_QURATER`

  季 K

- `K_YEAR`

  年 K

- `K_3M`

  3 分 K

逐笔成交方向

> TickerDirect

- `NONE`

  未知

- `BUY`

  外盘 

- `SELL`

  内盘 

- `NEUTRAL`

  中性盘 

逐笔成交类型

> TickerType

- `UNKNOWN`

  未知

- `AUTO_MATCH`

  自动对盘

- `LATE`

  开市前成交盘

- `NON_AUTO_MATCH`

  非自动对盘

- `INTER_AUTO_MATCH`

  同一证券商自动对盘
- `INTER_NON_AUTO_MATCH`

  同一证券商非自动对盘
- `ODD_LOT`

  碎股交易
- `AUCTION`

  竞价交易
- `BULK`

  批量交易
- `CRASH`

  现金交易
- `CROSS_MARKET`

  跨市场交易
- `BULK_SOLD`

  批量卖出
- `FREE_ON_BOARD`

  离价交易
- `RULE127_OR155`

  第 127 条交易（纽交所规则）或第 155 条交易
- `DELAY`

  延迟交易
- `MARKET_CENTER_CLOSE_PRICE`

中央收市价

- `NEXT_DAY`

  隔日交易

- `MARKET_CENTER_OPENING`

  中央开盘价交易

- `PRIOR_REFERENCE_PRICE`

  前参考价

- `MARKET_CENTER_OPEN_PRICE`

  中央开盘价

- `SELLER`

  卖方

- `T`

  T 类交易（盘前和盘后交易）

- `EXTENDED_TRADING_HOURS`

  延长交易时段

- `CONTINGENT`

  合单交易

- `AVERAGE_PRICE`

  平均价成交

- `OTC_SOLD`

  场外售出

- `ODD_LOT_CROSS_MARKET`

  碎股跨市场交易

- `DERIVATIVELY_PRICED`

  衍生工具定价

- `REOPENINGP_RICED`

  再开盘定价

- `CLOSING_PRICED`

  收盘定价

- `COMPREHENSIVE_DELAY_PRICE`

  综合延迟价格

- `OVERSEAS`

  交易的一方不是香港交易所的成员，属于场外交易

交易日查询市场

> TradeDateMarket

- `NONE`

  未知

- `HK`

  香港市场 

- `US`

  美国市场 

- `CN`

  A 股市场

- `NT`

  深（沪）股通

- `ST`

  港股通（深、沪）

- `JP_FUTURE`

  日本期货

- `SG_FUTURE`

  新加坡期货

交易日类型

> TradeDateType

- `WHOLE`

  全天交易

- `MORNING`

  上午交易，下午休市

- `AFTERNOON`

  下午交易，上午休市

窝轮状态

> WarrantStatus

- `NONE`

  未知

- `NORMAL`

  正常状态

- `SUSPEND`

停牌

- `STOP_TRADE`

  终止交易

- `PENDING_LISTING`

  等待上市

窝轮类型

> WrtType

- `NONE`

  未知

- `CALL`

  认购窝轮

- `PUT`

  认沽窝轮

- `BULL`

  牛证

- `BEAR`

  熊证

- `INLINE`

  界内证

所属交易所

> ExchType

- `NONE`

  未知
- `HK_MAINBOARD`

  港交所·主板
- `HK_GEMBOARD`

  港交所·创业板
- `HK_HKEX`

  港交所
- `US_NYSE`

  纽交所
- `US_NASDAQ`

  纳斯达克
- `US_PINK`

  OTC市场
- `US_AMEX`

  美交所
- `US_OPTION`

  美国
- `US_NYMEX`

  NYMEX
- `US_COMEX`

  COMEX
- `US_CBOT`

CBOT

- `US_CME`

  CME

- `US_CBOE`

  CBOE

- `CN_SH`

  上交所

- `CN_SZ`

  深交所

- `CN_STIB`

  科创板

- `SG_SGX`

  新交所

- `JP_OSE`

  大阪交易所

证券标识

Security

```
message Security
{
    required int32 market = 1; //QotMarket，行情市场
    required string code = 2; //代码
}
```

K 线数据

KLine

```
message KLine
{
    required string time = 1; //时间戳字符串（格式：yyyy-MM-dd HH:mm:ss）
    required bool isBlank = 2; //是否是空内容的点,若为 true 则只有时间信息
    optional double highPrice = 3; //最高价
    optional double openPrice = 4; //开盘价
    optional double lowPrice = 5; //最低价
    optional double closePrice = 6; //收盘价
    optional double lastClosePrice = 7; //昨收价
    optional int64 volume = 8; //成交量
    optional double turnover = 9; //成交额
    optional double turnoverRate = 10; //换手率（该字段为百分比字段，展示为小数表示
    optional double pe = 11; //市盈率
    optional double changeRate = 12; //涨跌幅（该字段为百分比字段，默认不展示 %，如
    optional double timestamp = 13; //时间戳
}
```

基础报价的期权特有字段

OptionBasicQotExData

```
message OptionBasicQotExData
{
    required double strikePrice = 1; //行权价
    required int32 contractSize = 2; //每份合约数(整型数据)
    optional double contractSizeFloat = 17; //每份合约数（浮点型数据）
    required int32 openInterest = 3; //未平仓合约数
    required double impliedVolatility = 4; //隐含波动率（该字段为百分比字段，默认不
    required double premium = 5; //溢价（该字段为百分比字段，默认不展示 %，如 20 实
    required double delta = 6; //希腊值 Delta
    required double gamma = 7; //希腊值 Gamma
    required double vega = 8; //希腊值 Vega
    required double theta = 9; //希腊值 Theta
    required double rho = 10; //希腊值 Rho
    optional int32 netOpenInterest = 11; //净未平仓合约数，仅港股期权适用
    optional int32 expiryDateDistance = 12; //距离到期日天数，负数表示已过期
    optional double contractNominalValue = 13; //合约名义金额，仅港股期权适用
    optional double ownerLotMultiplier = 14; //相等正股手数，指数期权无该字段，仅港
```

```
    optional int32 optionAreaType = 15; //OptionAreaType，期权类型（按行权时间）
    optional double contractMultiplier = 16; //合约乘数
    optional int32 indexOptionType = 18; //IndexOptionType，指数期权类型
}
```

基础报价的期货特有字段

FutureBasicQotExData

```
message FutureBasicQotExData
{
    required double lastSettlePrice = 1; //昨结
    required int32 position = 2; //持仓量
    required int32 positionChange = 3; //日增仓
    optional int32 expiryDateDistance = 4; //距离到期日天数
}
```

基础报价

BasicQot

```
message BasicQot
{
    required Security security = 1; //股票
    optional string name = 24; // 股票名称
    required bool isSuspended = 2; //是否停牌
    required string listTime = 3; //上市日期字符串（此字段停止维护，不建议使用，格式
    required double priceSpread = 4; //价差
    required string updateTime = 5; //最新价的更新时间字符串（格式：yyyy-MM-dd HH:
    required double highPrice = 6; //最高价
    required double openPrice = 7; //开盘价
    required double lowPrice = 8; //最低价
    required double curPrice = 9; //最新价
    required double lastClosePrice = 10; //昨收价
    required int64 volume = 11; //成交量
    required double turnover = 12; //成交额
    required double turnoverRate = 13; //换手率（该字段为百分比字段，默认不展示 %，
    required double amplitude = 14; //振幅（该字段为百分比字段，默认不展示 %，如 20
```

```
    optional int32 darkStatus = 15; //DarkStatus, 暗盘交易状态
    optional OptionBasicQotExData optionExData = 16; //期权特有字段
    optional double listTimestamp = 17; //上市日期时间戳（此字段停止维护，不建议使用
    optional double updateTimestamp = 18; //最新价的更新时间戳，对其他字段不适用
    optional PreAfterMarketData preMarket = 19; //盘前数据
    optional PreAfterMarketData afterMarket = 20; //盘后数据
    optional int32 secStatus = 21; //SecurityStatus, 股票状态
    optional FutureBasicQotExData futureExData = 22; //期货特有字段
}
```

盘前盘后数据

PreAfterMarketData

```
//美股支持盘前盘后数据
//科创板仅支持盘后数据：成交量，成交额
message PreAfterMarketData
{
    optional double price = 1;  // 盘前或盘后## 价格
    optional double highPrice = 2;  // 盘前或盘后## 最高价
    optional double lowPrice = 3;  // 盘前或盘后## 最低价
    optional int64 volume = 4;  // 盘前或盘后## 成交量
    optional double turnover = 5;  // 盘前或盘后## 成交额
    optional double changeVal = 6;  // 盘前或盘后## 涨跌额
    optional double changeRate = 7;  // 盘前或盘后## 涨跌幅（该字段为百分比字段，默
    optional double amplitude = 8;  // 盘前或盘后## 振幅（该字段为百分比字段，默认不
}
```

分时数据

TimeShare

```
message TimeShare
{
    required string time = 1; //时间字符串（格式：yyyy-MM-dd HH:mm:ss）
    required int32 minute = 2; //距离0点过了多少分钟
    required bool isBlank = 3; //是否是空内容的点,若为 true 则只有时间信息
    optional double price = 4; //当前价
```

```
    optional double lastClosePrice = 5; //昨收价
    optional double avgPrice = 6; //均价
    optional int64 volume = 7; //成交量
    optional double turnover = 8; //成交额
    optional double timestamp = 9; //时间戳
}
```

证券基本静态信息

SecurityStaticBasic

```

message SecurityStaticBasic
{
    required Qot_Common.Security security = 1; //股票
    required int64 id = 2; //股票 ID
    required int32 lotSize = 3; //每手数量,期权类型表示一份合约的股数
    required int32 secType = 4; //Qot_Common.SecurityType,股票类型
    required string name = 5; //股票名字
    required string listTime = 6; //上市时间字符串（此字段停止维护，不建议使用，格式
    optional bool delisting = 7; //是否退市
    optional double listTimestamp = 8; //上市时间戳（此字段停止维护，不建议使用）
    optional int32 exchType = 9; //Qot_Common.ExchType,所属交易所
}
```

窝轮额外静态信息

WarrantStaticExData

```
message WarrantStaticExData
{
    required int32 type = 1; //Qot_Common.WarrantType,窝轮类型
    required Qot_Common.Security owner = 2; //所属正股
}
```

期权额外静态信息

OptionStaticExData

```
message OptionStaticExData
{
    required int32 type = 1; //Qot_Common.OptionType,期权
    required Qot_Common.Security owner = 2; //标的股
    required string strikeTime = 3; //行权日（格式：yyyy-MM-dd）
    required double strikePrice = 4; //行权价
    required bool suspend = 5; //是否停牌
    required string market = 6; //发行市场名字
    optional double strikeTimestamp = 7; //行权日时间戳
    optional int32 indexOptionType = 8; //Qot_Common.IndexOptionType，指数期权的类
     optional int32 expirationCycle = 9; // ExpirationCycle，交割周期
    optional int32 optionStandardType = 10; // OptionStandardType，标准期权
    optional int32 optionSettlementMode = 11; // OptionSettlementMode，结算方式
}
```

期货额外静态信息

FutureStaticExData

```
message FutureStaticExData
{
    required string lastTradeTime = 1; //最后交易日，只有非主连期货合约才有该字段
    optional double lastTradeTimestamp = 2; //最后交易日时间戳，只有非主连期货合约才
    required bool isMainContract = 3; //是否主连合约
}
```

证券静态信息

SecurityStaticInfo

```
message SecurityStaticInfo
{
    required SecurityStaticBasic basic = 1; //证券基本静态信息
    optional WarrantStaticExData warrantExData = 2; //窝轮额外静态信息
    optional OptionStaticExData optionExData = 3; //期权额外静态信息
    optional FutureStaticExData futureExData = 4; //期货额外静态信息
}
```

买卖经纪

Broker

```
message Broker
{
    required int64 id = 1; //经纪 ID
    required string name = 2; //经纪名称
    required int32 pos = 3; //经纪档位

    //以下为港股 SF 行情特有字段
    optional int64 orderID = 4; //交易所订单 ID，与交易接口返回的订单 ID 并不一样
    optional int64 volume = 5; //订单股数
}
```

逐笔成交

Ticker

```
message Ticker
{
    required string time = 1; //时间字符串（格式：yyyy-MM-dd HH:mm:ss）
    required int64 sequence = 2; // 唯一标识
    required int32 dir = 3; //TickerDirection，买卖方向
    required double price = 4; //价格
    required int64 volume = 5; //成交量
    required double turnover = 6; //成交额
    optional double recvTime = 7; //收到推送数据的本地时间戳，用于定位延迟
```

```
    optional int32 type = 8; //TickerType, 逐笔类型
    optional int32 typeSign = 9; //逐笔类型符号
    optional int32 pushDataType = 10; //用于区分推送情况，仅推送时有该字段
    optional double timestamp = 11; //时间戳
}
```

买卖档明细

OrderBookDetail

```
message OrderBookDetail
{
    required int64 orderID = 1; //交易所订单 ID，与交易接口返回的订单 ID 并不一样
    required int64 volume = 2; //订单股数
}
```

买卖档

OrderBook

```
message OrderBook
{
    required double price = 1; //委托价格
    required int64 volume = 2; //委托数量
    required int32 orederCount = 3; //委托订单个数
    repeated OrderBookDetail detailList = 4; //订单信息，港股 SF，美股深度摆盘特有
}
```

持股变动

ShareHoldingChange

```
message ShareHoldingChange
{
```

```
    required string holderName = 1; //持有者名称（机构名称 或 基金名称 或 高管姓名）
    required double holdingQty = 2; //当前持股数量
    required double holdingRatio = 3; //当前持股百分比（该字段为百分比字段，默认不展
    required double changeQty = 4; //较上一次变动数量
    required double changeRatio = 5; //较上一次变动百分比（该字段为百分比字段，默认
    required string time = 6; //发布时间（格式：yyyy-MM-dd HH:mm:ss）
    optional double timestamp = 7; //时间戳
}
```

单个订阅类型信息

SubInfo

```
message SubInfo
{
    required int32 subType = 1;  //Qot_Common.SubType,订阅类型
    repeated Qot_Common.Security securityList = 2;  //订阅该类型行情的证券
}
```

单条连接订阅信息

ConnSubInfo

```
message ConnSubInfo
{
    repeated SubInfo subInfoList = 1; //该连接订阅信息
    required int32 usedQuota = 2; //该连接已经使用的订阅额度
    required bool isOwnConnData = 3; //用于区分是否是自己连接的数据
}
```

板块信息

PlateInfo

```
message PlateInfo
{
    required Qot_Common.Security plate = 1; //板块
    required string name = 2; //板块名字
    optional int32 plateType = 3; //PlateSetType 板块类型，仅3207（获取股票所属板块
}
```

复权信息

Rehab

```
message Rehab
{
    required string time = 1; //时间字符串（格式：yyyy-MM-dd）
    required int64 companyActFlag = 2; //公司行动(CompanyAct)组合标志位,指定某些字
    required double fwdFactorA = 3; //前复权因子 A
    required double fwdFactorB = 4; //前复权因子 B
    required double bwdFactorA = 5; //后复权因子 A
    required double bwdFactorB = 6; //后复权因子 B
    optional int32 splitBase = 7; //拆股(例如，1拆5，Base 为1，Ert 为5)
    optional int32 splitErt = 8;
    optional int32 joinBase = 9; //合股(例如，50合1，Base 为50，Ert 为1)
    optional int32 joinErt = 10;
    optional int32 bonusBase = 11; //送股(例如，10送3，Base 为10,Ert 为3)
    optional int32 bonusErt = 12;
    optional int32 transferBase = 13; //转赠股(例如，10转3，Base 为10,Ert 为3)
    optional int32 transferErt = 14;
    optional int32 allotBase = 15; //配股(例如，10送2，配股价为6.3元，Base 为10，E
    optional int32 allotErt = 16;
    optional double allotPrice = 17;
    optional int32 addBase = 18; //增发股(例如，10送2，增发股价为6.3元，Base 为10,
    optional int32 addErt = 19;
    optional double addPrice = 20;
    optional double dividend = 21; //现金分红(例如，每10股派现0.5元,则该字段值为0.0
    optional double spDividend = 22; //特别股息(例如，每10股派特别股息0.5元,则该字段
    optional double timestamp = 23; //时间戳
}
```

> - 公司行动组合标志位参见 CompanyAct

交割周期

> ExpirationCycle

- `NONE`

  未知

- `WEEK`

  周期权

- `MONTH`

  月期权

- `END_OF_MONTH`

  月末期权

- `QUARTERLY`

  季期权

- `WEEKMON`

  周期权-周一

- `WEEKTUE`

  周期权-周二

- `WEEKWED`

  周期权-周三

- `WEEKTHU`

  周期权-周四

- `WEEKFRI`

  周期权-周五

期权标准类型

> OptionStandardType

- `NONE`

  未知

- `STANDARD`

  标准期权

- `NON_STANDARD`

  非标准期权

期权结算方式

> OptionSettlementMode

- `NONE`

  未知

- `AM`

  亚式期权

- `PM`

  路径依赖型

股票持有者（已废弃）

> StockHolder

- `NONE`

  未知
- `INSTITUTE`

  机构
- `FUND`

  基金
- `EXECUTIVE`

  高管

交易接口总览

模块	接口名	功能简介
账户	Get Account List	获取交易业务账户列表
	Unlock Trading	解锁交易
资产持仓	Get Account Financial Information	获取账户资金数据
	Get Maximum Tradable Quantity	查询账户最大可买卖数量
	Get Positions List	获取持仓列表
	Get Margin Trading Data	获取融资融券数据
	Get Cash Flow Summary	查询账户现金流水
订单	Place Order	下单
	Modify or Cancel Order	改单撤单
	Get Order list	查询未完成订单
	Get Order Fees	查询订单费用
	Get Historical Order List	查询历史订单
	Order Callback	订单回调
	Trade Data Callback	订阅交易推送
成交	Get Today's Executed Trades	查询当日成交
	Get Historical Executed Trades	查询历史成交
	Trade Execution Callback	成交回调

交易对象

创建连接

`OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, is_encrypt=None, security_firm=SecurityFirm.FUTUSECURITIES)`

`OpenFutureTradeContext(host='127.0.0.1', port=11111, is_encrypt=None, security_firm=SecurityFirm.FUTUSECURITIES)`

- **介绍**

  根据交易品类，选择账户，并创建对应的交易对象。

实例	账户
OpenSecTradeContext	证券账户
OpenFutureTradeContext	期货账户

- **参数**

参数	类型	说明
filter_trdmarket	TrdMarket	筛选对应交易市场权限的账户
host	str	OpenD 监听的 IP 地址
port	int	OpenD 监听的 IP 端口
is_encrypt	bool	是否启用加密
security_firm	SecurityFirm	所属券商

- **Example**

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
trd_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

关闭连接

`close()`

- **介绍**

  关闭交易对象。默认情况下，moomoo API 内部创建的线程会阻止进程退出，只有当所有 Context 都 close 后，进程才能正常退出。但通过 set_all_thread_daemon 可以设置所有内部线程为 daemon 线程，这时即使没有调用 Context 的 close，进程也可以正常退出。

- **Example**

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
trd_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽
```

获取交易业务账户列表

`get_acc_list()`

- **介绍**

  获取交易业务账户列表。
  要调用其他交易接口前，请先获取此列表，确认要操作的交易业务账户无误。

- **参数**

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回交易业务账户列表
	str	当 ret != RET_OK 时，返回错误描述

  - 交易业务账户列表格式如下：

字段	类型	说明
acc_id	int	交易业务账户
trd_env	TrdEnv	交易环境
acc_type	TrdAccType	账户类型
uni_card_num	str	综合账户卡号，同移动端内的展示
card_num	str	业务账户卡号
security_firm	SecurityFirm	所属券商
sim_acc_type	SimAccType	模拟账户类型

字段	类型	说明
trdmarket_auth	list	交易市场权限
acc_status	TrdAccStatus	账户状态
acc_role	TrdAccRole	账户结构
jp_acc_type	list	日本账户类型

- **说明**

  当开通了港/美股期权模拟交易后，此接口在获取港/美交易账号列表时，会返回2个模拟交易账号。其中第1个为原先的账号，第2个是期权模拟交易账号。 目前OpenAPI拉取的美股模拟交易账号和移动端不是同一个账户，点击这里了解更多。

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.get_acc_list()
if ret == RET_OK:
    print(data)
    print(data['acc_id'][0])  # 取第一个账号
    print(data['acc_id'].values.tolist())  # 转为 list
else:
    print('get_acc_list error: ', data)
trd_ctx.close()
```

- Output

```
              acc_id   trd_env acc_type       uni_card_num           card_num
0  281756420273981734      REAL   MARGIN  10018561211263256   1001100530724347
1             3450310  SIMULATE     CASH                N/A                N/A
2             3548732  SIMULATE   MARGIN                N/A                N/A
281756420273981734
[281756420273981734, 3450310, 3548732]
```

解锁交易

`unlock_trade(password=None, password_md5=None, is_unlock=True)`

- **介绍**

  解锁或锁定交易

- **参数**

参数	类型	说明
password	str	交易密码
password_md5	str	交易密码的 32 位 MD5 加密（全小写）
is_unlock	bool	解锁或锁定

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
msg	NoneType	当 ret == RET_OK 时，返回 None
	str	当 ret != RET_OK 时，返回错误描述

- **Example**

```
from moomoo import *
pwd_unlock = '123456'
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.unlock_trade(pwd_unlock)
if ret == RET_OK:
    print('unlock success!')
else:
```

```
8       print('unlock_trade failed: ', data)
9   trd_ctx.close()
```

- Output

```
1   unlock success!
```

提示

- 真实账户调用 下单 或 改单撤单 接口，需要先解锁交易；模拟账户无需解锁。
- 解锁或锁定交易，是针对 OpenD 的操作，只要有一个连接解锁，其他连接都可以调用交易接口。
- 强烈建议，通过外网连接 OpenD 进行实盘交易的客户，使用加密通道，参见 启用协议加密。
- OpenAPI 不支持富途令牌，如果开通了富途令牌，则会解锁失败，需要关闭令牌功能后再使用 OpenAPI 解锁。

接口限制

- 单用户ID 每 30 秒内最多请求 10 次解锁交易接口

查询账户资金

```
accinfo_query(trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, refresh_cache=False, currency=Currency.HKD, asset_category=AssetCategory.NONE)
```

- **介绍**

  查询交易业务账户的资产净值、证券市值、现金、购买力等资金数据。

- **参数**

参数	类型	说明
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
refresh_cache	bool	是否刷新缓存
currency	Currency	资金的展示货币
asset_category	AssetCategory	资产类别

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回资金数据
	str	当 ret != RET_OK 时，返回错误描述

  - 资金数据格式如下：

字段	类型	说明
power	float	最大购买力
max_power_short	float	卖空购买力
net_cash_power	float	现金购买力
total_assets	float	总资产净值
securities_assets	float	证券资产净值
fund_assets	float	基金资产净值
bond_assets	float	债券资产净值
cash	float	现金
market_val	float	证券市值
long_mv	float	多头市值
short_mv	float	空头市值
pending_asset	float	在途资产
interest_charged_amount	float	计息金额
frozen_cash	float	冻结资金
avl_withdrawal_cash	float	现金可提
max_withdrawal	float	最大可提
currency	Currency	计价货币
available_funds	float	可用资金
unrealized_pl	float	未实现盈亏
realized_pl	float	已实现盈亏

字段	类型	说明
risk_level	CltRiskLevel	风控状态
risk_status	CltRiskStatus	风险状态
initial_margin	float	初始保证金
margin_call_margin	float	Margin Call 保证金
maintenance_margin	float	维持保证金
hk_cash	float	港元现金
hk_avl_withdrawal_cash	float	港元可提
hkd_net_cash_power	float	港元现金购买力
hkd_assets	float	港股资产净值
us_cash	float	美元现金
us_avl_withdrawal_cash	float	美元可提
usd_net_cash_power	float	美元现金购买力
usd_assets	float	美股资产净值
cn_cash	float	人民币现金
cn_avl_withdrawal_cash	float	人民币可提
cnh_net_cash_power	float	人民币现金购买力
cnh_assets	float	A股资产净值
jp_cash	float	日元现金
jp_avl_withdrawal_cash	float	日元可提
jpy_net_cash_power	float	日元现金购买力

字段	类型	说明
jpy_assets	float	日股资产净值
sg_cash	float	新元现金
sg_avl_withdrawal_cash	float	新元可提
sgd_net_cash_power	float	新元现金购买力
sgd_assets	float	新股资产净值
au_cash	float	澳元现金
au_avl_withdrawal_cash	float	澳元可提
aud_net_cash_power	float	澳元现金购买力
aud_assets	float	澳股资产净值
ca_cash	float	加元现金
ca_avl_withdrawal_cash	float	加元可提
cad_net_cash_power	float	加元现金购买力
cad_assets	float	加元资产净值
my_cash	float	令吉现金
my_avl_withdrawal_cash	float	令吉可提
myr_net_cash_power	float	令吉现金购买力
myr_assets	float	令吉资产净值
is_pdt	bool	是否为 PDT 账户
pdt_seq	string	剩余日内交易次数
beginning_dtbp	float	初始日内交易购买力

字段	类型	说明
remaining_dtbp	float	剩余日内交易购买力
dt_call_amount	float	日内交易待缴金额
dt_status	DtStatus	日内交易限制情况

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.accinfo_query()
if ret == RET_OK:
    print(data)
    print(data['power'][0])  # 取第一行的购买力
    print(data['power'].values.tolist())  # 转为 list
else:
    print('accinfo_query error: ', data)
trd_ctx.close()  # 关闭当条连接
```

- Output

```
power  max_power_short  net_cash_power  total_assets  securities_assets  fund_ass
0  465453.903307    465453.903307            0.0   289932.0404        197028.220
465453.903307
[465453.903307]
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询账户资金接口
- 调用此接口，只有在刷新缓存时，才受到限频限制

查询最大可买可卖

```
acctradinginfo_query(order_type, code, price, order_id=None, adjust_limit=0, trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, session=Session.NONE, jp_acc_type=SubAccType.JP_GENERAL, position_id=NONE)
```

- **介绍**

  查询指定交易业务账户下的最大可买卖数量，亦可查询指定交易业务账户下指定订单的最大可改成的数量。

  现金账户请求期权不适用。

- **参数**

参数	类型	说明
order_type	OrderType	订单类型
code	str	证券代码
price	float	报价
order_id	str	订单号
adjust_limit	float	价格微调幅度
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
session	Session	美股交易时段
jp_acc_type	SubAccType	日本账户类型
position_id	int	持仓ID

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回账号列表
	str	当 ret != RET_OK 时，返回错误描述

  - 账号列表格式如下：

字段	类型	说明
max_cash_buy	float	现金可买
max_cash_and_margin_buy	float	最大可买
max_position_sell	float	持仓可卖
max_sell_short	float	可卖空
max_buy_back	float	平仓需买入
long_required_im	float	买 1 张合约所带来的初始保证金变动。
short_required_im	float	卖 1 张合约所带来的初始保证金变动。
session	Session	交易订单时段（仅用于美股）

- **Example**

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', p
ret, data = trd_ctx.acctradinginfo_query(order_type=OrderType.NORMAL, code='US.A
if ret == RET_OK:
    print(data)
    print(data['max_cash_and_margin_buy'][0])  # 最大融资可买数量
else:
```

```
    print('acctradinginfo_query error: ', data)
trd_ctx.close()  # 关闭当条连接
```

- Output

```
    max_cash_buy  max_cash_and_margin_buy  max_position_sell  max_sell_short  max
0           0.0                  1500.0                0.0             0.0
1500.0
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询最大可买可卖接口

提示

- 现金业务账户不支持交易衍生品，因此不支持通过现金业务账户查询期权的最大可买可卖。
- 期货的最大可买，需自行计算，公式：floor(最大购买力/买 1 张合约所带来的初始保证金变动)。其中，最大购买力来自查询账户资金，买 1 张合约所带来的初始保证金变动来自本接口。

查询持仓

```
position_list_query(code='', position_market=TrdMarket.NONE, pl_ratio_min=None, pl_ratio_max=None, trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, refresh_cache=False, asset_category=AssetCategory.NONE)
```

- **介绍**

  查询交易业务账户的持仓列表

- **参数**

参数	类型	说明
code	str	代码过滤
position_market	TrdMarket	持仓所属市场过滤
pl_ratio_min	float	当前盈亏比例下限过滤，仅返回高于此比例的持仓
pl_ratio_max	float	当前盈亏比例上限过滤，低于此比例的会返回
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
refresh_cache	bool	是否刷新缓存
asset_category	AssetCategory	资产类别

- **返回**

参数	类型	说明

ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回持仓列表
	str	当 ret != RET_OK 时，返回错误描述

- 持仓列表

字段	类型	说明
position_side	PositionSide	持仓方向
code	str	股票代码
stock_name	str	股票名称
position_market	TrdMarket	持仓所属市场
qty	float	持有数量
can_sell_qty	float	可用数量
currency	Currency	交易货币
nominal_price	float	市价
cost_price	float	摊薄成本价（证券账户），平均开仓价（期货账户）
cost_price_valid	bool	成本价是否有效
average_cost	float	平均成本价
diluted_cost	float	摊薄成本价
market_val	float	市值
pl_ratio	float	盈亏比例（摊薄成本价模式）
pl_ratio_valid	bool	盈亏比例是否有效

字段	类型	说明
pl_ratio_avg_cost	float	盈亏比例（平均成本价模式）
pl_val	float	盈亏金额
pl_val_valid	bool	盈亏金额是否有效
today_pl_val	float	今日盈亏金额
today_trd_val	float	今日交易金额
today_buy_qty	float	今日买入总量
today_buy_val	float	今日买入总额
today_sell_qty	float	今日卖出总量
today_sell_val	float	今日卖出总额
unrealized_pl	float	未实现盈亏
realized_pl	float	已实现盈亏
position_id	int	持仓ID

- **Example**

```
from futu import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.position_list_query()
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果持仓列表不为空
        print(data['stock_name'][0])  # 获取持仓第一个股票名称
        print(data['stock_name'].values.tolist())  # 转为 list
else:
    print('position_list_query error: ', data)
trd_ctx.close()  # 关闭当条连接
```

- **Output**

```
        code stock_name position_market    qty  can_sell_qty  cost_price  cost_pri
0  US.AAPL       苹果                HK  400.0         400.0      53.975
苹果
['苹果']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询持仓接口
- 调用此接口，只有在刷新缓存时，才受到限频限制

获取融资融券数据

`get_margin_ratio(code_list)`

- **介绍**

  查询股票的融资融券数据。

- **参数**

参数	类型	说明
code_list	list	股票代码列表

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回融资融券数据
	str	当 ret != RET_OK 时，返回错误描述

  - 融资融券数据格式如下：

字段	类型	说明
code	str	股票代码
is_long_permit	bool	是否允许融资
is_short_permit	bool	是否允许融券
short_pool_remain	float	卖空池剩余
short_fee_rate	float	融券参考利率

字段	类型	说明
alert_long_ratio	float	融资预警比率
alert_short_ratio	float	融券预警比率
im_long_ratio	float	融资初始保证金率
im_short_ratio	float	融券初始保证金率
mcm_long_ratio	float	融资 margin call 保证金率
mcm_short_ratio	float	融券 margin call 保证金率
mm_long_ratio	float	融资维持保证金率
mm_short_ratio	float	融券维持保证金率

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.get_margin_ratio(code_list=['US.AAPL','US.FUTU'])
if ret == RET_OK:
    print(data)
    print(data['is_long_permit'][0])  # 取第一条的是否允许融资
    print(data['im_short_ratio'].values.tolist())  # 转为 list
else:
    print('error:', data)
trd_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽
```

- Output

```
       code  is_long_permit  is_short_permit  short_pool_remain  short_fee_rate
0  US.AAPL            True             True          1826900.0            0.89
1  US.FUTU            True             True          1150600.0            0.95
True
[60.0, 50.0]
```

接口限制

- 单用户ID 每 30 秒内最多请求 10 次获取融资融券数据接口。
- 每次请求，接口参数股票代码列表，支持传入的标的数量上限是 100 个。
- 支持美国、香港、A股市场的股票和ETF。

查询账户现金流水

`get_acc_cash_flow(clearing_date='', trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, cashflow_direction=CashFlowDirection.NONE)`

- **介绍**

  查询交易业务账户在指定日期的现金流水数据。数据覆盖出入金、调拨、货币兑换、买卖金融资产、融资融券利息等所有导致现金变动的事项。

- **参数**

参数	类型	说明
clearing_date	str	清算日期
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
cashflow_direction	CashFlowDirection	筛选现金流方向

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回交易业务账户现金流水列表格式
	str	当 ret != RET_OK 时，返回错误描述

  - 交易业务账户现金流水列表格式如下：

字段	类型	说明
cashflow_id	int	现金流唯一标识
clearing_date	str	清算日期
settlement_date	str	交收日期
currency	Currency	币种
cashflow_type	str	现金流类型
cashflow_direction	CashFlowDirection	现金流方向
cashflow_amount	float	金额（正数表示流入，负数表示流出）
cashflow_remark	str	备注

- Example

```
from futu import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.get_acc_cash_flow(clearing_date='2025-02-18', trd_env=TrdEnv
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果现金流水列表不为空
        print(data['cashflow_type'][0])  # 获取第一条流水的现金流类型
        print(data['cashflow_amount'].values.tolist())  # 转为 list
else:
    print('get_acc_cash_flow error: ', data)
trd_ctx.close()

```

- Output

```
   cashflow_id     clearing_date     settlement_date     currency     cashflow_ty
0  16308           2025-02-27        2025-02-28          HKD            其他
1  16357           2025-02-27        2025-03-03          HKD            其他
2  16360           2025-02-27        2025-02-27          USD            基金赎回
```

```
3  16384          2025-02-27       2025-02-27          HKD              基金赎回
其他
[0.00, -104000.00, 23000.00, 104108.96]
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 20 次现金流水接口。
- 现金流水，按照时间的“顺序”进行排列。
- 模拟交易和 moomoo US 账户暂不支持查询现金流水。

下单

```
place_order(price, qty, code, trd_side, order_type=OrderType.NORMAL, adjust_limit=0, trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, remark=None, time_in_force=TimeInForce.DAY, fill_outside_rth=False, aux_price=None, trail_type=None, trail_value=None, trail_spread=None, session=Session.NONE, jp_acc_type=SubAccType.JP_GENERAL, position_id=NONE)
```

- **介绍**

  下单

  > **提示**
  >
  > Python API 是同步的，但网络收发是异步的。当 place_order 对应的应答数据包与 响应成交推送回调 或 响应订单推送回调 间隔很短时，就可能出现 place_order 的数据包先返回，但回调函数先被调用的情况。例如：可能先调用了 响应订单推送回调，然后 place_order 这个接口才返回。

- **参数**

参数	类型	说明
price	float	订单价格
qty	float	订单数量
code	str	标的代码
trd_side	TrdSide	交易方向
order_type	OrderType	订单类型
adjust_limit	float	价格微调幅度
trd_env	TrdEnv	交易环境

参数	类型	说明
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
remark	str	备注
time_in_force	TimeInForce	有效期限
fill_outside_rth	bool	是否允许盘前盘后
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比
trail_spread	float	指定价差
session	Session	美股交易时段
jp_acc_type	SubAccType	日本账户类型
position_id	int	持仓ID

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回订单列表
	str	当 ret != RET_OK 时，返回错误描述

  - 订单列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
order_type	OrderType	订单类型
order_status	OrderStatus	订单状态
order_id	str	订单号
code	str	股票代码
stock_name	str	股票名称
qty	float	订单数量
price	float	订单价格
create_time	str	创建时间
updated_time	str	最后更新时间
dealt_qty	float	成交数量
dealt_avg_price	float	成交均价
last_err_msg	str	最后的错误描述
remark	str	下单时备注的标识
time_in_force	TimeInForce	有效期限
fill_outside_rth	bool	是否允许盘前盘后（用于港股盘前竞价与美股盘前盘后）
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比

字段	类型	说明
trail_spread	float	指定价差
session	Session	交易订单时段（仅用于美股）

- Example

```
from moomoo import *
pwd_unlock = '123456'
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.unlock_trade(pwd_unlock)  # 若使用真实账户下单，需先对账户进行解
if ret == RET_OK:
    ret, data = trd_ctx.place_order(price=510.0, qty=100, code="US.AAPL", trd_sid
    if ret == RET_OK:
        print(data)
        print(data['order_id'][0])  # 获取下单的订单号
        print(data['order_id'].values.tolist())  # 转为 list
    else:
        print('place_order error: ', data)
else:
    print('unlock_trade failed: ', data)
trd_ctx.close()
```

- Output

```

       code stock_name trd_side order_type order_status          order_id    qty
0  US.AAPL         苹果        BUY     NORMAL   SUBMITTING  381960065487095000  100
381960065487095000
['381960065487095000']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 15 次下单接口，且连续两次请求的间隔不可小于 0.02 秒。
- 真实账户调用下单接口前，需要先进行 解锁；模拟账户无需解锁。

提示

- 各订单类型对应的必传参数：点击这里 了解更多
- 各券商针对不同交易品种，对单笔订单股数有所限制，超出限制会导致下单失败：点击这里 了解更多
- 对于 **可做空标的**，暂不支持锁仓功能，故无法同时持有相同产品的多头头寸和空头头寸。
- 如果希望对 **可做空标的** 进行 **平仓** 操作，需要自行判断持仓头寸的方向，然后提交一笔反向的相同数量的订单完成平仓操作。
- 如果希望对 **可做空标的** 进行 **反手** 操作，需要两步：1. 先判断持仓头寸的方向，并提交一笔反向的相同数量的订单完成平仓操作；2. 提交一笔反向的订单，完成反向订单的提交。
  举例：A 当前持有 1 手 HK.HSI2012 期货合约的多单，如果希望反手，必须先 卖出 1 手 HK.HSI2012 完成平仓，再卖出 1 手 HK.HSI2012 完成空单的建立。
- 美股全时段交易，仅支持限价单，订单期限可以选择当日有效或撤单前有效。选择全时段，交易者可以一次挂单参与多个时段（夜盘、盘前、盘中、盘后时段）的交易，全时段交易时间是星期日到星期四 20:00 - 次日20:00（美东时间）
- 美股模拟交易不支持盘前盘后与夜盘。

改单撤单

```
modify_order(modify_order_op, order_id, qty, price, adjust_limit=0, trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, aux_price=None, trail_type=None, trail_value=None, trail_spread=None)
```

- **介绍**

  修改订单的价格和数量、撤单、操作订单的失效和生效、删除订单等。
  如果是 A 股通市场，将不支持改单。可撤单。删除订单是 OpenD 本地操作。

- **参数**

参数	类型	说明
modify_order_op	ModifyOrderOp	改单操作类型
order_id	str	订单号
qty	float	订单改单后的数量
price	float	订单改单后的价格
adjust_limit	float	价格微调幅度
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比
trail_spread	float	指定价差

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回改单信息
	str	当 ret != RET_OK 时，返回错误描述

  - 改单信息格式如下：

字段	类型	说明
trd_env	TrdEnv	交易环境
order_id	str	订单号

- **Example**

```
from moomoo import *
pwd_unlock = '123456'
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', p
ret, data = trd_ctx.unlock_trade(pwd_unlock)  # 若使用真实账户改单/撤单，需先对账户
if ret == RET_OK:
    order_id = "8851102695472794941"
    ret, data = trd_ctx.modify_order(ModifyOrderOp.CANCEL, order_id, 0, 0)
    if ret == RET_OK:
        print(data)
        print(data['order_id'][0])  # 获取改单的订单号
        print(data['order_id'].values.tolist())  # 转为 list
    else:
        print('modify_order error: ', data)
else:
    print('unlock_trade failed: ', data)
trd_ctx.close()
```

- **Output**

```
    trd_env             order_id
0    REAL      8851102695472794941
8851102695472794941
['8851102695472794941']
```

`cancel_all_order(trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, trdmarket=TrdMarket.NONE)`

- **介绍**

  撤消全部订单。模拟交易以及 A 股通账户暂不支持全部撤单。

- **参数**

参数	类型	说明
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
trdmarket	TrdMarket	指定交易市场

- **返回**

参数	类型	说明
ret	str	接口调用结果。ret == RET_OK 代表接口调用正常，ret != RET_OK 代表接口调用失败
data	str	当 ret == RET_OK，返回"success"
		当 ret != RET_OK，返回错误描述

  - 全部撤单信息格式如下：

字段	类型	说明
trd_env	TrdEnv	交易环境
order_id	str	订单号

- Example

```
from moomoo import *
pwd_unlock = '123456'
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', p
ret, data = trd_ctx.unlock_trade(pwd_unlock)  # 若使用真实账户改单/撤单，需先对账户
if ret == RET_OK:
    ret, data = trd_ctx.cancel_all_order()
    if ret == RET_OK:
        print(data)
    else:
        print('cancel_all_order error: ', data)
else:
    print('unlock_trade failed: ', data)
trd_ctx.close()
```

- Output

```
success
```

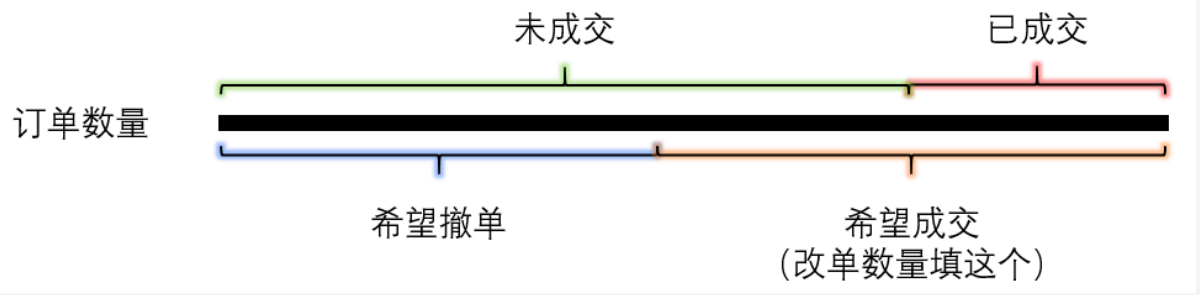
接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 20 次改单撤单接口，且连续两次请求的间隔不可小于 0.04 秒。
- 真实账户调用改单撤单接口前，需要先进行 解锁；模拟账户无需解锁。

提示

- 若执行 **修改订单** 操作，各类订单类型对应的必传参数，可 点击这里 了解更多。
- 如果希望执行 **改单操作** 去 **修改订单数量**，此接口入参的订单数量 **qty**，应该等于期望成交的总数量。
  举例： 一笔订单数量是 N 股，已部分成交 n 股。对于暂未成交的 (N-n) 股，如果您

希望撤掉其中的 x 股，**modify_order_op** 应选择 NORMAL，**qty** 应传 (N-x)。



- 如果希望执行 **撤单操作**，此接口入参的 **modify_order_op** 应该选择 CANCEL。
  举例： 一笔订单数量是 N 股，已部分成交 n 股。如果希望将未成交的 (N-n) 股全部撤掉，modify_order_op 应选择 CANCEL，此时 qty 和 price 的入参会被忽略。

查询未完成订单

`order_list_query(order_id="", order_market=TrdMarket.NONE, status_filter_list=[], code='', start='', end='', trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, refresh_cache=False)`

- **介绍**

  查询指定交易业务账户的未完成订单列表

- **参数**

参数	类型	说明
order_id	str	订单号过滤
order_market	TrdMarket	订单标的所属市场过滤
status_filter_list	list	订单状态过滤
code	str	代码过滤
start	str	开始时间
end	str	结束时间
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
refresh_cache	bool	是否刷新缓存

- **返回**

参数	类型	说明

ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回订单列表
	str	当 ret != RET_OK 时，返回错误描述

- 订单列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
order_type	OrderType	订单类型
order_status	OrderStatus	订单状态
order_id	str	订单号
code	str	股票代码
stock_name	str	股票名称
order_market	TrdMarket	订单标的所属市场
qty	float	订单数量
price	float	订单价格
currency	Currency	交易货币
create_time	str	创建时间
updated_time	str	最后更新时间
dealt_qty	float	成交数量
dealt_avg_price	float	成交均价
last_err_msg	str	最后的错误描述
remark	str	下单时备注的标识

字段	类型	说明
time_in_force	TimeInForce	有效期限
fill_outside_rth	bool	是否允许盘前盘后（用于港股盘前竞价与美股盘前盘后）
session	Session	交易订单时段（仅用于美股）
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比
trail_spread	float	指定价差
jp_acc_type	SubAccType	日本账户类型

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
ret, data = trd_ctx.order_list_query()
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果订单列表不为空
        print(data['order_id'][0])  # 获取未完成订单的第一个订单号
        print(data['order_id'].values.tolist())  # 转为 list
else:
    print('order_list_query error: ', data)
trd_ctx.close()
```

- Output

```
        code stock_name   order_amrket      trd_side           order_type   order
0   US.AAPL         US          BUY           NORMAL  CANCELLED_ALL  664446861527
6644468615272262086
['6644468615272262086']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询未完成订单接口
- 调用此接口，只有在刷新缓存时，才受到限频限制

提示

- 未完成订单，按照时间的“顺序”进行排列，即：先提交的订单在前，后提交的订单在后

查询历史订单

`history_order_list_query(status_filter_list=[], code='', order_market=TrdMarket.NONE, start='', end='', trd_env=TrdEnv.REAL, acc_id=0, acc_index=0)`

- **介绍**

  查询指定交易业务账户的历史订单列表

- **参数**

参数	类型	说明
status_filter_list	list	订单状态过滤
code	str	代码过滤
order_market	TrdMarket	订单标的所属市场过滤
start	str	开始时间
end	str	结束时间
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号

  - start 和 end 的组合如下

Start 类型	End 类型	说明
str	str	start 和 end 分别为指定的日期
None	str	start 为 end 往前 90 天

Start 类型	End 类型	说明
str	None	end 为 start 往后 90 天
None	None	start 为往前 90 天，end 当前日期

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回订单列表
	str	当 ret != RET_OK 时，返回错误描述

  - 订单列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
order_type	OrderType	订单类型
order_status	OrderStatus	订单状态
order_id	str	订单号
code	str	股票代码
stock_name	str	股票名称
order_market	TrdMarket	订单标的所属市场
qty	float	订单数量
price	float	订单价格
currency	Currency	交易货币

字段	类型	说明
create_time	str	创建时间
updated_time	str	最后更新时间
dealt_qty	float	成交数量
dealt_avg_price	float	成交均价
last_err_msg	str	最后的错误描述
remark	str	下单时备注的标识
time_in_force	TimeInForce	有效期限
fill_outside_rth	bool	是否允许盘前盘后（用于港股盘前竞价与美股盘前盘后）
session	Session	交易订单时段（仅用于美股）
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比
trail_spread	float	指定价差
jp_acc_type	SubAccType	日本账户类型

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', pc
ret, data = trd_ctx.history_order_list_query()
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果订单列表不为空
        print(data['order_id'][0])  # 获取持仓第一个订单号
        print(data['order_id'].values.tolist())  # 转为 list
```

```
else:
    print('history_order_list_query error: ', data)
trd_ctx.close()
```

- Output

```
        code stock_name order_market   trd_side           order_type   order_stat
0   HK.00700        HK           BUY           NORMAL  CANCELLED_ALL  664446861527
6644468615272262086
['6644468615272262086']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询历史订单接口

提示

- 历史订单，按照时间的“倒序”进行排列，即：后提交的订单在前，先提交的订单在后

响应订单推送回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  响应订单推送，异步处理 OpenD 推送过来的订单状态信息。
  在收到 OpenD 推送过来的订单状态信息后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。

- **参数**

参数	类型	说明
rsp_pb	Trd_UpdateOrder_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回订单列表
	str	当 ret != RET_OK 时，返回错误描述

  - 订单列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
order_type	OrderType	订单类型
order_status	OrderStatus	订单状态
order_id	str	订单号

字段	类型	说明
code	str	股票代码
stock_name	str	股票名称
qty	float	订单数量
price	float	订单价格
currency	Currency	交易货币
create_time	str	创建时间
updated_time	str	最后更新时间
dealt_qty	float	成交数量
dealt_avg_price	float	成交均价
last_err_msg	str	最后的错误描述
remark	str	下单时备注的标识
time_in_force	TimeInForce	有效期限
fill_outside_rth	bool	是否允许盘前盘后（仅用于美股）
session	Session	交易订单时段（仅用于美股）
aux_price	float	触发价格
trail_type	TrailType	跟踪类型
trail_value	float	跟踪金额/百分比
trail_spread	float	指定价差

- **Example**

```python
from moomoo import *
from time import sleep
class TradeOrderTest(TradeOrderHandlerBase):
    """ order update push"""
    def on_recv_rsp(self, rsp_pb):
        ret, content = super(TradeOrderTest, self).on_recv_rsp(rsp_pb)
        if ret == RET_OK:
            print("* TradeOrderTest content={}\n".format(content))
        return ret, content

trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
trd_ctx.set_handler(TradeOrderTest())
print(trd_ctx.place_order(price=518.0, qty=100, code="US.AAPL", trd_side=TrdSide

sleep(15)
trd_ctx.close()
```

- Output

```
* TradeOrderTest content=  trd_env      code stock_name  dealt_avg_price  dealt_
0    REAL  US.AAPL       苹果                0.0        0.0  100.0  7262526370867
```

查询订单费用

`order_fee_query(order_id_list=[], acc_id=0, acc_index=0, trd_env=TrdEnv.REAL)`

- **介绍**

  查询指定订单的收费明细（最低版本要求：8.2.4218）

- **参数**

参数	类型	说明
order_id_list	list	订单号列表
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回订单费用列表
	str	当 ret != RET_OK 时，返回错误描述

  - 订单列表格式如下：

字段	类型	说明
order_id	str	订单号
fee_amount	float	总费用

字段	类型	说明
fee_details	list	收费明细

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', pc
ret1, data1 = trd_ctx.history_order_list_query(status_filter_list=[OrderStatus.FI
if ret1 == RET_OK:
    if data1.shape[0] > 0:  # 如果订单列表不为空
        ret2, data2 = trd_ctx.order_fee_query(data1['order_id'].values.tolist())
        if ret2 == RET_OK:
            print(data2)
            print(data2['fee_details'][0])  # 打印第一笔订单的收费明细
        else:
            print('order_fee_query error: ', data2)
else:
    print('order_list_query error: ', data1)
trd_ctx.close()
```

- Output

```
                                          order_id  fee_amount
0  v3_20240314_12345678_MTc4NzA5NzY5OTA3ODAzMzMwN       10.46  [(佣金, 5.85), (平
1  v3_20240318_12345678_MTM5Nzc5MDYxNDY1NDM1MDI1M        2.25  [(佣金, 0.99), (平
[('佣金', 5.85), ('平台使用费', 2.7), ('期权监管费', 0.11), ('期权清算费', 0.18), (
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询订单费用接口。
- 仅支持查询 2018-01-01 之后的订单。
- 模拟账户不支持查询订单费用。
- 加拿大券商账户不支持查询订单费用。

订阅交易推送

Python 不需要订阅交易推送

查询当日成交

`deal_list_query(code="", deal_market=TrdMarket.NONE, trd_env=TrdEnv.REAL, acc_id=0, acc_index=0, refresh_cache=False)`

- **介绍**

  查询指定交易业务账户的当日成交列表。
  该接口只支持实盘交易，不支持模拟交易。

- **参数**

参数	类型	说明
code	str	代码过滤
deal_market	TrdMarket	成交标的所属市场过滤
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号
refresh_cache	bool	是否刷新缓存

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回交易成交列表
	str	当 ret != RET_OK 时，返回错误描述

  - 交易成交列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
deal_id	str	成交号
order_id	str	订单号
code	str	股票代码
stock_name	str	股票名称
deal_market	TrdMarket	成交标的所属市场
qty	float	成交数量
price	float	成交价格
create_time	str	创建时间
counter_broker_id	int	对手经纪号
counter_broker_name	str	对手经纪名称
status	DealStatus	成交状态
jp_acc_type	SubAccType	日本账户类型

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', pc
ret, data = trd_ctx.deal_list_query()
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果成交列表不为空
        print(data['order_id'][0])  # 获取当日成交的第一个订单号
        print(data['order_id'].values.tolist())  # 转为 list
else:
    print('deal_list_query error: ', data)
trd_ctx.close()
```

- Output

```
    code stock_name        deal_market        deal_id             order_id    qty
0  US.AAPL      苹果         US    5056208452274069375  4665291631090960915  100.0
4665291631090960915
['4665291631090960915']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询当日成交接口
- 调用此接口，只有在刷新缓存时，才受到限频限制

提示

- 当日成交，按照时间的“顺序”进行排列，即：先成交的记录在前，后成交的记录在后

查询历史成交

`history_deal_list_query(code='', deal_market=TrdMarket.NONE, start='', end='', trd_env=TrdEnv.REAL, acc_id=0, acc_index=0)`

- **介绍**

  查询指定交易业务账户的历史成交列表。
  该接口只支持实盘交易，不支持模拟交易。

- **参数**

参数	类型	说明
code	str	代码过滤
deal_market	TrdMarket	成交标的所属市场过滤
start	str	开始时间
end	str	结束时间
trd_env	TrdEnv	交易环境
acc_id	int	交易业务账户 ID
acc_index	int	交易业务账户列表中的账户序号

  - start 和 end 的组合如下

Start 类型	End 类型	说明
str	str	start 和 end 分别为指定的日期
None	str	start 为 end 往前 90 天
str	None	end 为 start 往后 90 天

Start 类型	End 类型	说明
None	None	start 为往前 90 天，end 当前日期

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回交易成交列表
	str	当 ret != RET_OK 时，返回错误描述

  - 交易成交列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
deal_id	str	成交号
order_id	str	订单号
code	str	股票代码
stock_name	str	股票名称
deal_market	TrdMarket	成交标的所属市场
qty	float	成交数量
price	float	成交价格
create_time	str	创建时间
counter_broker_id	int	对手经纪号
counter_broker_name	str	对手经纪名称

字段	类型	说明
status	DealStatus	成交状态
jp_acc_type	SubAccType	日本账户类型

- Example

```
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', p
ret, data = trd_ctx.history_deal_list_query()
if ret == RET_OK:
    print(data)
    if data.shape[0] > 0:  # 如果成交列表不为空
        print(data['deal_id'][0])  # 获取历史成交的第一个成交号
        print(data['deal_id'].values.tolist())  # 转为 list
else:
    print('history_deal_list_query error: ', data)
trd_ctx.close()
```

- Output

```
    code stock_name         deal_market        deal_id             order_id    q
0  US.AAPL       苹果      US   5056208452274069375  4665291631090960915  100.0
5056208452274069375
['5056208452274069375']
```

接口限制

- 同一账户ID(acc_id) 每 30 秒内最多请求 10 次查询历史成交接口

提示

- 历史成交，按照时间的“倒序”进行排列，即：后成交的记录在前，先成交的记录在后

响应成交推送回调

`on_recv_rsp(self, rsp_pb)`

- **介绍**

  响应成交推送，异步处理 OpenD 推送过来的成交状态信息。
  在收到 OpenD 推送过来的成交状态信息后会回调到该函数，您需要在派生类中覆盖 on_recv_rsp。
  该接口只支持实盘交易，不支持模拟交易。

- **参数**

参数	类型	说明
rsp_pb	Trd_UpdateOrderFill_pb2.Response	派生类中不需要直接处理该参数

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	pd.DataFrame	当 ret == RET_OK 时，返回交易成交列表
	str	当 ret != RET_OK 时，返回错误描述

  - 交易成交列表格式如下：

字段	类型	说明
trd_side	TrdSide	交易方向
deal_id	str	成交号
order_id	str	订单号

字段	类型	说明
code	str	股票代码
stock_name	str	股票名称
qty	float	成交数量
price	float	成交价格
create_time	str	创建时间
counter_broker_id	int	对手经纪号
counter_broker_name	str	对手经纪名称
status	DealStatus	成交状态

- Example

```
from moomoo import *
from time import sleep
class TradeDealTest(TradeDealHandlerBase):
    """ order update push"""
    def on_recv_rsp(self, rsp_pb):
        ret, content = super(TradeDealTest, self).on_recv_rsp(rsp_pb)
        if ret == RET_OK:
            print("TradeDealTest content={}".format(content))
        return ret, content

trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US, host='127.0.0.1', po
trd_ctx.set_handler(TradeDealTest())
print(trd_ctx.place_order(price=595.0, qty=100, code="US.AAPL", trd_side=TrdSide
sleep(15)
trd_ctx.close()
```

- Output

```
TradeDealTest content=  trd_env      code stock_name             deal_id
```

```
2    0    REAL  US.AAPL      苹果  2511067564122483295  8561504228375901919  100.0
```

交易定义

账户风控状态

> CltRiskLevel

- `NONE`

  未知

- `SAFE`

  安全

- `WARNING`

  预警

- `DANGER`

  危险

- `ABSOLUTE_SAFE`

  绝对安全

- `OPT_DANGER`

  危险 

> **提示**
>
> - 查询期货账户的风险状态，建议使用 risk_status 字段， 返回结果详见 CltRiskStatus

货币类型

> Currency

- `NONE`

  未知货币
- `HKD`

  港元
- `USD`

  美元
- `CNH`

  离岸人民币
- `JPY`

  日元
- `SGD`

  新元
- `AUD`

  澳元
- `CAD`

  加拿大元
- `MYR`

  马来西亚林吉特

跟踪类型

TrailType

- `NONE`

  未知
- `RATIO`

比例

- `AMOUNT`

  金额

修改订单操作

> ModifyOrderOp

- `NONE`

  未知操作

- `NORMAL`

  修改订单

- `CANCEL`

  撤单 

- `DISABLE`

  使失效 

- `ENABLE`

  使生效 

- `DELETE`

  删除 

成交状态

> DealStatus

- `OK`

  正常

- `CANCELLED`

  成交被取消

- `CHANGED`

  成交被更改

订单状态

> OrderStatus

- `NONE`

  未知状态

- `WAITING_SUBMIT`

  待提交 

- `SUBMITTING`

  提交中 

- `SUBMITTED`

  已提交，等待成交 

- `FILLED_PART`

  部分成交 

- `FILLED_ALL`

  全部已成交

- `CANCELLED_PART`

  部分成交，剩余部分已撤单

- `CANCELLED_ALL`

  全部已撤单，无成交

- `FAILED`

  下单失败，服务拒绝

- `DISABLED`

  已失效

- `DELETED`

  已删除，无成交的订单才能删除

订单类型

> **提示**
>
> - 实盘交易中，各个品类支持的订单类型
> - 模拟交易中，仅支持限价单(NORMAL)和市价单(MARKET)。

> OrderType

- `NONE`

  未知类型

- `NORMAL`

  限价单

- `MARKET`

  市价单

- `ABSOLUTE_LIMIT`

  绝对限价订单

- `AUCTION`

  竞价市价单

- `AUCTION_LIMIT`

竞价限价单

- `SPECIAL_LIMIT`

  特别限价单

- `SPECIAL_LIMIT_ALL`

  特别限价且要求全部成交订单

- `STOP`

  止损市价单

- `STOP_LIMIT`

  止损限价单

- `MARKET_IF_TOUCHED`

  触及市价单（止盈）

- `LIMIT_IF_TOUCHED`

  触及限价单（止盈）

- `TRAILING_STOP`

  跟踪止损市价单

- `TRAILING_STOP_LIMIT`

  跟踪止损限价单

- `TWAP_LIMIT`

  时间加权限价算法单（港股和美股）

- `TWAP`

  时间加权市价算法单（仅美股）

- `VWAP_LIMIT`

  成交量加权限价算法单（港股和美股）

- `VWAP`

  成交量加权市价算法单（仅美股）

持仓方向

> PositionSide

- `NONE`

  未知方向

- `LONG`

  多仓

- `SHORT`

  空仓

账户类型

> TrdAccType

- `NONE`

  未知类型

- `CASH`

  现金账户

- `MARGIN`

  保证金账户

- `TFSA`

  加拿大免税账户

- `RRSP`

加拿大注册退休账户

- `SRRSP`

  加拿大配偶退休账户

- `DERIVATIVE`

  日本衍生品账户

交易环境

> TrdEnv

- `SIMULATE`

  模拟环境

- `REAL`

  真实环境

交易市场

> TrdMarket

- `NONE`

  未知市场

- `HK`

  香港市场

- `US`

  美国市场

- `CN`

  A 股市场

- `HKCC`

  香港 A 股通市场

- `FUTURES`

  期货市场

- `FUTURES_SIMULATE_US`

  美国期货模拟市场

- `FUTURES_SIMULATE_HK`

  香港期货模拟市场

- `FUTURES_SIMULATE_SG`

  新加坡期货模拟市场

- `FUTURES_SIMULATE_JP`

  日本期货模拟市场

- `HKFUND`

  香港基金市场

- `USFUND`

  美国基金市场

- `SG`

  新加坡市场

- `JP`

  日本市场

- `AU`

  澳大利亚市场

- `MY`

马来西亚市场 

- `CA`

  加拿大市场 

账户状态

> TrdAccStatus

- `ACTIVE`

  生效账户

- `DISABLED`

  失效账户

账户结构

> TrdAccRole

- `NONE`

  未知

- `MASTER`

  主账户

- `NORMAL`

  普通账户

- `IPO`

  马来西亚IPO账户

交易证券市场

交易方向

> TrdSide

- `NONE`

  未知方向

- `BUY`

  买入

- `SELL`

  卖出

- `SELL_SHORT`

  卖空 

- `BUY_BACK`

  买回 

> **提示**
>
> **下单** 接口的交易方向，建议仅使用 `买入` 和 `卖出` 两个方向作为入参。`卖空` 和 `买回` 仅适用于日本券商，其他券商仅用于 **查询今日订单**，**查询历史订单**，**响应订单推送回调**，**查询当日成交**，**查询历史成交**，**响应成交推送回调** 接口的返回字段展示。

订单有效期

> TimeInForce

- `DAY`

当日有效

- `GTC`

  撤单前有效

账户所属券商

> SecurityFirm

- `NONE`

  未知

- `FUTUSECURITIES`

  富途证券（香港）

- `FUTUINC`

  moomoo证券(美国)

- `FUTUSG`
  moomoo证券(新加坡)

- `FUTUAU`
  moomoo证券(澳大利亚)

- `FUTUCA`
  moomoo证券(加拿大)

- `FUTUMY`
  moomoo证券(马来西亚)

- `FUTUJP`
  moomoo证券(日本)

模拟交易账户类型

SimAccType

- `NONE`

  未知
- `STOCK`

  股票模拟账户
- `OPTION`

  期权模拟账户
- `FUTURES`

  期货模拟账户

风险状态

> CltRiskStatus

- `NONE`

  未知
- `LEVEL1`

  非常安全
- `LEVEL2`

  安全
- `LEVEL3`

  较安全
- `LEVEL4`

  较低风险
- `LEVEL5`

  中等风险

- `LEVEL6`

  偏高风险
- `LEVEL7`

  预警
- `LEVEL8`

  危险
- `LEVEL9`

  危险

日内交易限制情况

> DtStatus

- `NONE`

  未知
- `Unlimited`

  无限次
- `EM_Call`

  EM-Call
- `DT_Call`

  DT-Call

现金流方向

> CashFlowDirection

- `NONE`

未知

- `IN`

  现金流入

- `OUT`

  现金流出

日本子账户类型

> SubAccType

- `NONE`

  未知

- `JP_GENERAL`

  一般-Long

- `JP_TOKUTEI`

  特定-Long

- `JP_NISA_GENERAL`

  一般NISA

- `JP_NISA_TSUMITATE`

  累计NISA

- `JP_GENERAL_SHORT`

  一般-short

- `JP_TOKUTEI_SHORT`

  特定-short

- `JP_HONPO_GENERAL`

本国信用交易抵押品-一般

- `JP_GAIKOKU_GENERAL`

  外国信用交易抵押品-一般

- `JP_HONPO_TOKUTEI`

  本国信用交易抵押品-特定

- `JP_GAIKOKU_TOKUTEI`

  外国信用交易抵押品-特定

- `JP_DERIVATIVE_LONG`

  衍生品子账户-Long

- `JP_DERIVATIVE_SHORT`

  衍生品子账户-Short

- `JP_HONPO_DERIVATIVE_GENERAL`

  本国衍生品证据金子账户-一般

- `JP_GAIKOKU_DERIVATIVE_GENERAL`

  外国衍生品证据金子账户-一般

- `JP_HONPO_DERIVATIVE_TOKUTEI`

  本国衍生品证据金子账户-特定

- `JP_GAIKOKU_DERIVATIVE_TOKUTEI`

  外国衍生品证据金子账户-特定

资产类别

> AssetCategory

- `NONE`

未知

- `JP`

  本国

- `US`

  外国

交易品类

TrdCategory

```
enum TrdCategory
{
    TrdCategory_Unknown = 0; //未知品类
    TrdCategory_Security = 1; //证券
    TrdCategory_Future = 2; //期货
}
```

账户现金信息

AccCashInfo

```
message AccCashInfo
{
    optional int32 currency = 1;         // 货币类型，取值参考 Currency
    optional double cash = 2;            // 现金结余
    optional double availableBalance = 3;    // 现金可提金额
    optional double netCashPower = 4;      // 现金购买力
}
```

分市场资产信息

AccMarketInfo

```
message AccCashInfo
{
    optional int32 trdMarket = 1;         // 交易市场，参见TrdMarket的枚举定义
    optional double assets = 2;           // 分市场资产信息
}
```

交易协议公共参数头

TrdHeader

```
message TrdHeader
{
  required int32 trdEnv = 1; //交易环境，参见 TrdEnv 的枚举定义
  required uint64 accID = 2; //业务账号，业务账号与交易环境、市场权限需要匹配，否则
  required int32 trdMarket = 3; //交易市场，参见 TrdMarket 的枚举定义
  optional int32 jpAccType = 4; //JP子账户类型，取值见 TrdSubAccType
}
```

交易业务账户

TrdAcc

```
message TrdAcc
{
  required int32 trdEnv = 1; //交易环境，参见 TrdEnv 的枚举定义
  required uint64 accID = 2; //业务账号
  repeated int32 trdMarketAuthList = 3; //业务账户支持的交易市场权限，即此账户能交易
  optional int32 accType = 4;   //账户类型，取值见 TrdAccType
  optional string cardNum = 5;  //卡号
  optional int32 securityFirm = 6; //所属券商，取值见SecurityFirm
  optional int32 simAccType = 7; //模拟交易账号类型，取值见SimAccType
  optional string uniCardNum = 8;  //所属综合账户卡号
  optional int32 accStatus = 9; //账号状态，取值见TrdAccStatus
  optional int32 accRole = 10; //账号分类，是不是主账号，取值见TrdAccRole
  repeated int32 jpAccType = 11; //JP子账户类型，取值见 TrdSubAccType
}
```

```
13
14
```

账户资金

Funds

```
message Funds
{
  required double power = 1; //最大购买力（此字段是按照 50% 的融资初始保证金率计算得
  required double totalAssets = 2; //资产净值
  required double cash = 3; //现金（仅单币种账户使用此字段，综合账户请使用 cashInfo
  required double marketVal = 4; //证券市值，仅证券账户适用
  required double frozenCash = 5; //冻结资金
  required double debtCash = 6; //计息金额
  required double avlWithdrawalCash = 7; //现金可提（仅单币种账户使用此字段，综合账

  optional int32 currency = 8;            //币种，本结构体资金相关的货币类型，取值参
  optional double availableFunds = 9;     //可用资金，期货适用
  optional double unrealizedPL = 10;      //未实现盈亏，期货适用
  optional double realizedPL = 11;        //已实现盈亏，期货适用
  optional int32 riskLevel = 12;           //风控状态，参见 CltRiskLevel，期货适用
  optional double initialMargin = 13;      //初始保证金
  optional double maintenanceMargin = 14;  //维持保证金
  repeated AccCashInfo cashInfoList = 15;  //分币种的现金、现金可提和现金购买力（仅
  optional double maxPowerShort = 16; //卖空购买力（此字段是按照 60% 的融券保证金率
  optional double netCashPower = 17;  //现金购买力（仅单币种账户使用此字段，综合账户
  optional double longMv = 18;        //多头市值
  optional double shortMv = 19;       //空头市值
  optional double pendingAsset = 20;  //在途资产
  optional double maxWithdrawal = 21;          //融资可提，仅证券账户适用
  optional int32 riskStatus = 22;              //风险状态，参见 CltRiskStatus，共
  optional double marginCallMargin = 23;       //   Margin Call 保证金

  optional bool isPdt = 24;               //是否PDT账户，仅moomoo证券(美国)账户适用
  optional string pdtSeq = 25;            //剩余日内交易次数，仅被标记为 PDT 的moo
  optional double beginningDTBP = 26;     //初始日内交易购买力，仅被标记为 PDT 的m
  optional double remainingDTBP = 27;     //剩余日内交易购买力，仅被标记为 PDT 的m
  optional double dtCallAmount = 28;      //日内交易待缴金额，仅被标记为 PDT 的moo
  optional int32 dtStatus = 29;                //日内交易限制情况，取值见 DTStatus

  optional double securitiesAssets = 30; // 证券资产净值
  optional double fundAssets = 31; // 基金资产净值
```

```
  optional double bondAssets = 32; // 债券资产净值

  repeated AccMarketInfo marketInfoList = 33; //分市场资产信息
}
```

账户持仓

Position

```
message Position
{
    required uint64 positionID = 1;      //持仓 ID，一条持仓的唯一标识
    required int32 positionSide = 2;     //持仓方向，参见 PositionSide 的枚举定义
    required string code = 3;            //代码
    required string name = 4;            //名称
    required double qty = 5;             //持有数量，2位精度，期权单位是"张"，下同
    required double canSellQty = 6;      //可用数量，是指持有的可平仓的数量。可用数量
    required double price = 7;           //市价，3位精度，期货为2位精度
    optional double costPrice = 8;       //摊薄成本价（证券账户），平均开仓价（期货账
    required double val = 9;             //市值，3位精度，期货此字段值为0
    required double plVal = 10;          //盈亏金额，3位精度，期货为2位精度
    optional double plRatio = 11;        //盈亏百分比(平均成本价模式)，无精度限制，如
    optional int32 secMarket = 12;       //证券所属市场，参见 TrdSecMarket 的枚举定义

     //以下是此持仓今日统计
    optional double td_plVal = 21;       //今日盈亏金额，3位精度，下同，期货为2位精度
    optional double td_trdVal = 22;      //今日交易额，期货不适用
    optional double td_buyVal = 23;      //今日买入总额，期货不适用
    optional double td_buyQty = 24;      //今日买入总量，期货不适用
    optional double td_sellVal = 25;     //今日卖出总额，期货不适用
    optional double td_sellQty = 26;     //今日卖出总量，期货不适用

    optional double unrealizedPL = 28;       //未实现盈亏（仅期货账户适用）
    optional double realizedPL = 29;         //已实现盈亏（仅期货账户适用）
    optional int32 currency = 30;         // 货币类型，取值参考 Currency
    optional int32 trdMarket = 31;  //交易市场，参见 TrdMarket 的枚举定义

    optional double dilutedCostPrice = 32;      //摊薄成本价，仅支持证券账户使用
    optional double averageCostPrice = 33;      //平均成本价，模拟交易证券账户不适用
    optional double averagePlRatio = 34;        //盈亏百分比(平均成本价模式)，无精度
}
```

订单

Order

```
message Order
{
    required int32 trdSide = 1; //交易方向，参见 TrdSide 的枚举定义
    required int32 orderType = 2; //订单类型，参见 OrderType 的枚举定义
    required int32 orderStatus = 3; //订单状态，参见 OrderStatus 的枚举定义
    required uint64 orderID = 4; //订单号
    required string orderIDEx = 5; //扩展订单号(仅查问题时备用)
    required string code = 6; //代码
    required string name = 7; //名称
    required double qty = 8; //订单数量，2位精度，期权单位是"张"
    optional double price = 9; //订单价格，3位精度
    required string createTime = 10; //创建时间，严格按 YYYY-MM-DD HH:MM:SS 或 YYY
    required string updateTime = 11; //最后更新时间，严格按 YYYY-MM-DD HH:MM:SS 或
    optional double fillQty = 12; //成交数量，2位精度，期权单位是"张"
    optional double fillAvgPrice = 13; //成交均价，无精度限制
    optional string lastErrMsg = 14; //最后的错误描述，如果有错误，会有此描述最后一
    optional int32 secMarket = 15; //证券所属市场，参见 TrdSecMarket 的枚举定义
    optional double createTimestamp = 16; //创建时间戳
    optional double updateTimestamp = 17; //最后更新时间戳
    optional string remark = 18; //用户备注字符串，最大长度64字节
    optional double auxPrice = 21; //触发价格
    optional int32 trailType = 22; //跟踪类型，参见Trd_Common.TrailType的枚举定义
    optional double trailValue = 23; //跟踪金额/百分比
    optional double trailSpread = 24; //指定价差
    optional int32 currency = 25;         // 货币类型，取值参考 Currency
    optional int32 trdMarket = 26;  //交易市场，参见TrdMarket的枚举定义
    optional int32 session = 27; //美股订单时段，参见Common.Session的枚举定义
    optional int32 jpAccType = 28; //JP子账户类型，取值见 TrdSubAccType
}
```

订单费用条目

OrderFeeItem

```
message OrderFeeItem
{
    optional string title = 1; //费用名字
    optional double value = 2; //费用金额
}
```

订单费用

OrderFee

```
message OrderFee
{
    required string orderIDEx = 1; //扩展订单号
    optional double feeAmount = 2; //费用总额
    repeated OrderFeeItem feeList = 3; //费用明细
}
```

成交

OrderFill

```
message OrderFill
{
     required int32 trdSide = 1; //交易方向，参见 TrdSide 的枚举定义
    required uint64 fillID = 2; //成交号
    required string fillIDEx = 3; //扩展成交号(仅查问题时备用)
    optional uint64 orderID = 4; //订单号
    optional string orderIDEx = 5; //扩展订单号(仅查问题时备用)
    required string code = 6; //代码
    required string name = 7; //名称
    required double qty = 8; //成交数量，2位精度，期权单位是"张"
    required double price = 9; //成交价格，3位精度
    required string createTime = 10; //创建时间（成交时间），严格按 YYYY-MM-DD HH:
    optional int32 counterBrokerID = 11; //对手经纪号，港股有效
    optional string counterBrokerName = 12; //对手经纪名称，港股有效
    optional int32 secMarket = 13; //证券所属市场，参见 TrdSecMarket 的枚举定义
```

```
    optional double createTimestamp = 14; //创建时间戳
    optional double updateTimestamp = 15; //最后更新时间戳
    optional int32 status = 16; //成交状态，参见 OrderFillStatus 的枚举定义
    optional int32 trdMarket = 17;  //交易市场，参见TrdMarket的枚举定义
    optional int32 jpAccType = 18; //JP子账户类型，取值见 TrdSubAccType
}
```

最大可交易数量

MaxTrdQtys

```
message MaxTrdQtys
{
    //因目前服务器实现的问题，卖空需要先卖掉多头持仓才能再卖空，是分开两步卖的，买回
    required double maxCashBuy = 1;              //现金可买（期权的单位是“张”，期货
    optional double maxCashAndMarginBuy = 2;     //最大可买（期权的单位是“张”，期货
    required double maxPositionSell = 3;         //持仓可卖（期权的单位是“张”）
    optional double maxSellShort = 4;            //可卖空（期权的单位是“张”，期货账
    optional double maxBuyBack = 5;              //平仓需买入（当持有净空仓时，必须
    optional double longRequiredIM = 6;          //买 1 张合约所带来的初始保证金变动
    optional double shortRequiredIM = 7;         //卖 1 张合约所带来的初始保证金变动
}
```

现金流水数据

FlowSummaryInfo

```
message FlowSummaryInfo
{
    optional string clearingDate = 1; //清算日期
    optional string settlementDate = 2; //结算日期
    optional int32 currency = 3; //币种
    optional string cashFlowType = 4; //现金流类型
    optional int32 cashFlowDirection = 5; //现金流方向 TrdCashFlowDirection
    optional double cashFlowAmount = 6; //金额
    optional string cashFlowRemark = 7; //备注
    optional uint64 cashFlowID = 8; //现金流 ID
}
```

```
10
11
```

过滤条件

TrdFilterConditions

```
message TrdFilterConditions
{
  repeated string codeList = 1; //代码过滤，只返回包含这些代码的数据，没传不过滤
  repeated uint64 idList = 2; //ID 主键过滤，只返回包含这些 ID 的数据，没传不过滤，
  optional string beginTime = 3; //开始时间，严格按 YYYY-MM-DD HH:MM:SS 或 YYYY-MM
  optional string endTime = 4; //结束时间，严格按 YYYY-MM-DD HH:MM:SS 或 YYYY-MM-D
  repeated string orderIDExList = 5; // 服务器订单ID列表，可以用来替代orderID列表，
  optional int32 filterMarket = 6; //指定交易市场，参见TrdMarket的枚举定义
}
```

基础功能

设置接口信息

`set_client_info(client_id, client_ver)`

- **介绍**

  设置调用接口信息, 非必调接口

- **参数**

  - client_id: client 的标识
  - client_ver: client 的版本号

- **Example**

```
from moomoo import *
SysConfig.set_client_info("MymoomooAPI", 0)
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.close()
```

设置协议格式

`set_proto_fmt(proto_fmt)`

- **介绍**

  设置通讯协议 body 格式, 目前支持 Protobuf|Json 两种格式，默认 ProtoBuf, 非必调接口

- **参数**

  - proto_fmt: 协议格式，参见ProtoFMT

```
from moomoo import *
SysConfig.set_proto_fmt(ProtoFMT.Protobuf)
```

```
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.close()
```

- Example

对所有连接设置协议加密

`enable_proto_encrypt(is_encrypt)`

- **介绍**

  对所有连接的请求和返回内容加密。如需了解协议加密流程，详见 这里。

- **参数**

参数	类型	说明
is_encrypt	bool	是否启用加密

- Example

```
from moomoo import *
SysConfig.enable_proto_encrypt(is_encrypt = True)
SysConfig.set_init_rsa_file("conn_key.txt")   # rsa 私钥文件路径
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.close()
```

设置私钥路径

`set_init_rsa_file(file)`

- **介绍**

  设置 RSA 私钥文件路径。如需了解协议加密流程，详见 这里。

- **参数**

参数	类型	说明
file	str	私钥文件路径

- Example

```
from moomoo import *
SysConfig.enable_proto_encrypt(is_encrypt = True)
SysConfig.set_init_rsa_file("conn_key.txt")   # rsa 私钥文件路径
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.close()
```

设置线程模式

`set_all_thread_daemon(all_daemon)`

- **介绍**

  是否设置所有内部创建的线程为 daemon 线程。

  - 若设置为 daemon 线程：主线程退出后，则进程也退出。
    例如：使用实时回调接口时，需要自己保证主线程存活，否则主线程退出后，进程也退出，您将不会再接收到推送数据。
  - 若设置为非 daemon 线程：主线程退出后，进程不会退出。
    例如：在创建行情或交易对象后，若不调用 close() 关闭连接，即使主线程退出，进程不会退出。

- **参数**

参数	类型	说明
all_daemon	bool	是否设置为 daemon 线程

- Example

```
from moomoo import *
SysConfig.set_all_thread_daemon(True)
```

```
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
# 不调用 quote_ctx.close()，进程也会退出
```

设置回调

`set_handler(handler)`

- **介绍**

  设置异步回调处理对象

- **参数**

  - handler: 回调处理对象

类	说明
SysNotifyHandlerBase	OpenD 通知处理基类
StockQuoteHandlerBase	报价处理基类
OrderBookHandlerBase	摆盘处理基类
CurKlineHandlerBase	实时 K 线处理基类
TickerHandlerBase	逐笔处理基类
RTDataHandlerBase	分时数据处理基类
BrokerHandlerBase	经济队列处理基类
PriceReminderHandlerBase	到价提醒处理基类
TradeOrderHandlerBase	订单处理基类
TradeDealHandlerBase	成交处理基类

```
import time
from moomoo import *
class OrderBookTest(OrderBookHandlerBase):
    def on_recv_rsp(self, rsp_str):
```

```
        ret_code, data = super(OrderBookTest,self).on_recv_rsp(rsp_str)
        if ret_code != RET_OK:
            print("OrderBookTest: error, msg: %s" % data)
            return RET_ERROR, data
        print("OrderBookTest ", data) # OrderBookTest 自己的处理逻辑
        return RET_OK, data
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = OrderBookTest()
quote_ctx.set_handler(handler)  # 设置实时摆盘回调
quote_ctx.subscribe(['HK.00700'], [SubType.ORDER_BOOK])  # 订阅买卖摆盘类型，OpenD
time.sleep(15)  #  设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()  # 关闭当条连接，OpenD 会在1分钟后自动取消相应股票相应类型的订阅
```

获取连接 ID

`get_sync_conn_id()`

- **介绍**

  获取连接 ID，连接初始化成功后才会有值

- **返回**

  - conn_id: 连接 ID

- Example

```
from moomoo import *
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
quote_ctx.get_sync_conn_id()
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

事件通知回调

`SysNotifyHandlerBase`

- **介绍**

  通知 OpenD 一些重要消息，类似连接断开等

- **协议 ID**

1003

- **返回**

参数	类型	说明
ret	RET_CODE	接口调用结果
data	tuple	当 ret == RET_OK 时，返回 **事件通知数据**
	str	当 ret != RET_OK，返回错误描述

  - **事件通知数据** 的格式如下：

参数	类型	说明
notify_type	SysNotifyType	通知类型
sub_type	ProgramStatusType	子类型。当 notify_type == SysNotifyType.PROGRAM_STATUS 时，sub_type 返回程序状态类型
	GtwEventType	子类型。当 notify_type == SysNotifyType.GTW_EVENT 时，sub_type 返回 OpenD 事件通知类型
	0	当 notify_type != SysNotifyType.PROGRAM_STATUS 且 notify_type != SysNotifyType.GTW_EVENT 时，sub_type 返回 0
msg	dict	事件信息。当 notify_type == SysNotifyType.CONN_STATUS 时，msg 返回 **连接状态事件信息** 字典
		事件信息。当 notify_type == SysNotifyType.QOT_RIGHT 时，msg 返回 **行情权限事件信息** 字典

- **连接状态事件信息** 字典结构如下（连接状态类型为 bool，True 表示连接正常，False 表示连接断开）：

```
{
    'qot_logined': bool1,
    'trd_logined': bool2,
}
```

- **行情权限事件信息** 字典结构如下（点击了解 行情权限）：

```
{
    'hk_qot_right': value1,
    'hk_option_qot_right': value2,
    'hk_future_qot_right': value3,
    'us_qot_right': value4,
    'us_option_qot_right': value5,
    'us_future_qot_right': value6,  // 已废弃
    'cn_qot_right': value7,
     'us_index_qot_right': value8,
     'us_otc_qot_right': value9,
     'sg_future_qot_right': value10,
     'jp_future_qot_right': value11,
     'us_future_qot_right_cme': value12,
     'us_future_qot_right_cbot': value13,
     'us_future_qot_right_nymex': value14,
     'us_future_qot_right_comex': value15,
     'us_future_qot_right_cboe': value16,
}
```

- Example

```
import time
from moomoo import *


class SysNotifyTest(SysNotifyHandlerBase):
    def on_recv_rsp(self, rsp_str):
        ret_code, data = super(SysNotifyTest, self).on_recv_rsp(rsp_str)
        notify_type, sub_type, msg = data
```

```
        if ret_code != RET_OK:
            logger.debug("SysNotifyTest: error, msg: {}".format(msg))
            return RET_ERROR, data
        if notify_type == SysNotifyType.GTW_EVENT:  # OpenD 事件通知
            print("GTW_EVENT, type: {} msg: {}".format(sub_type, msg))
        elif notify_type == SysNotifyType.PROGRAM_STATUS:  # 程序状态变化通知
            print("PROGRAM_STATUS, type: {} msg: {}".format(sub_type, msg))
        elif notify_type == SysNotifyType.CONN_STATUS:  ## 连接状态变化通知
            print("CONN_STATUS, qot: {}".format(msg['qot_logined']))
            print("CONN_STATUS, trd: {}".format(msg['trd_logined']))
        elif notify_type == SysNotifyType.QOT_RIGHT:  # 行情权限变化通知
            print("QOT_RIGHT, hk: {}".format(msg['hk_qot_right']))
            print("QOT_RIGHT, hk_option: {}".format(msg['hk_option_qot_right']))
            print("QOT_RIGHT, hk_future: {}".format(msg['hk_future_qot_right']))
            print("QOT_RIGHT, us: {}".format(msg['us_qot_right']))
            print("QOT_RIGHT, us_option: {}".format(msg['us_option_qot_right']))
            print("QOT_RIGHT, cn: {}".format(msg['cn_qot_right']))
              print("QOT_RIGHT, us_index: {}".format(msg['us_index_qot_right']))
              print("QOT_RIGHT, us_otc: {}".format(msg['us_otc_qot_right']))
              print("QOT_RIGHT, sg_future: {}".format(msg['sg_future_qot_right']
              print("QOT_RIGHT, jp_future: {}".format(msg['jp_future_qot_right']
            print("QOT_RIGHT, us_future_cme: {}".format(msg['us_future_qot_right_
            print("QOT_RIGHT, us_future_cbot: {}".format(msg['us_future_qot_right
            print("QOT_RIGHT, us_future_nymex: {}".format(msg['us_future_qot_righ
            print("QOT_RIGHT, us_future_comex: {}".format(msg['us_future_qot_righ
            print("QOT_RIGHT, us_future_cboe: {}".format(msg['us_future_qot_right
        return RET_OK, data


quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111)
handler = SysNotifyTest()
quote_ctx.set_handler(handler)  # 设置回调
time.sleep(15)  # 设置脚本接收 OpenD 的推送持续时间为15秒
quote_ctx.close()  # 结束后记得关闭当条连接，防止连接条数用尽`
```

通用定义

接口调用结果

> RET_CODE

- `RET_OK`

  成功

- `RET_ERROR`

  失败

协议格式

> ProtoFMT

- `Protobuf`

  Google Protobuf 格式

- `Json`

  Json 格式

包加密算法

程序状态类型

> ProgramStatusType

- `NONE`

  未知

- `LOADED`

  已完成必要模块加载

- `LOGING`

  登录中

- `NEED_PIC_VERIFY_CODE`

  需要图形验证码

- `NEED_PHONE_VERIFY_CODE`

  需要手机验证码

- `LOGIN_FAILED`

  登录失败

- `FORCE_UPDATE`

  客户端版本过低

- `NESSARY_DATA_PREPARING`

  正在拉取必要信息

- `NESSARY_DATA_MISSING`

  缺少必要信息

- `UN_AGREE_DISCLAIMER`

  未同意免责声明

- `READY`

  正常可用状态

- `FORCE_LOGOUT`

  OpenD 登录后被强制退出登录

网关事件通知类型

> GtwEventType

- `LocalCfgLoadFailed`

  本地配置文件加载失败

- `APISvrRunFailed`

  网关监听服务运行失败

- `ForceUpdate`

  强制升级网关

- `LoginFailed`

  登录富途服务器失败

- `UnAgreeDisclaimer`

  未同意免责声明，无法运行

- `LOGIN_FAILED`

  登录失败

- `NetCfgMissing`

  缺少网络连接配置

- `KickedOut`

  登录被踢下线

- `LoginPwdChanged`

  登录密码变更

- `BanLogin`

  牛牛后台不允许该账号登录

- `NeedPicVerifyCode`

  登录需要输入图形验证码

- `NeedPhoneVerifyCode`

  登录需要输入手机验证码
- `AppDataNotExist`

  程序打包数据丢失
- `NessaryDataMissing`

  必要的数据没同步成功
- `TradePwdChanged`

  交易密码变更通知
- `EnableDeviceLock`

  需启用设备锁

系统通知类型

> SysNotifyType

- `GTW_EVENT`

  网关事件
- `PROGRAM_STATUS`

  程序状态变化
- `CONN_STATUS`

  与后台服务的连接状态变化
- `QOT_RIGHT`

  行情权限变化

包唯一标识

PacketID

```
message PacketID
{
	required uint64 connID = 1; //当前 TCP 连接的连接 ID，一条连接的唯一标识，In
	required uint32 serialNo = 2; //自增序列号
}
```

程序状态

ProgramStatus

```
message ProgramStatus
{
	required ProgramStatusType type = 1; //当前状态
	optional string strExtDesc = 2; // 额外描述
}
```

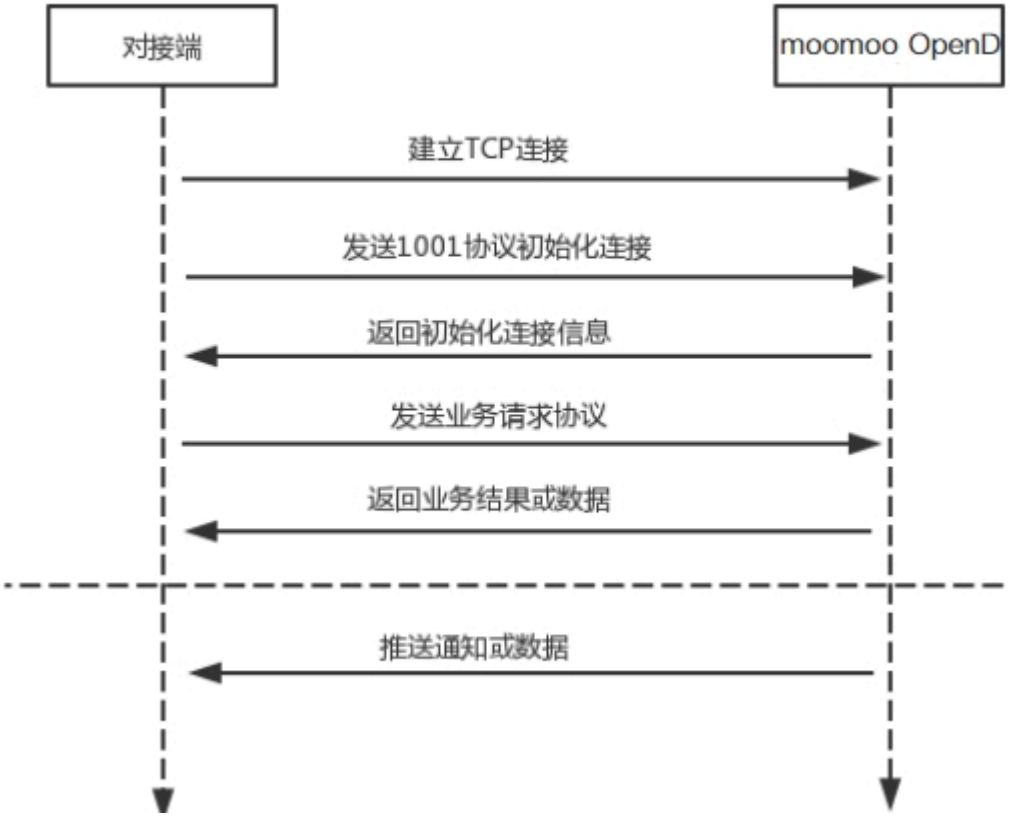
底层协议介绍

moomoo API 是 moomoo 为主流的编程语言（Python、Java、C#、C++、JavaScript）封装的 API SDK，以方便您调用，降低策略开发难度。
这部分主要介绍策略脚本与 OpenD 服务之间通信的底层协议，适用于非上述 5 种编程语言用户，自行对接实现底层裸协议。

> **提示**
>
> - 如果您使用的编程语言在上述的 5 种主流编程语言之内，可以直接跳过这部分内容。

协议请求流程

- 建立连接
- 初始化连接
- 请求数据或接收推送数据
- 定时发送 KeepAlive 保持连接



协议设计

协议数据包括协议头以及协议体，协议头固定字段，协议体根据具体协议决定。

协议头

```
struct APIProtoHeader
{
    u8_t szHeaderFlag[2];
    u32_t nProtoID;
    u8_t nProtoFmtType;
    u8_t nProtoVer;
    u32_t nSerialNo;
    u32_t nBodyLen;
    u8_t arrBodySHA1[20];
    u8_t arrReserved[8];
};
```

字段	说明
szHeaderFlag	包头起始标志，固定为“FT”
nProtoID	协议 ID
nProtoFmtType	协议格式类型，0 为 Protobuf 格式，1 为 Json 格式
nProtoVer	协议版本，用于迭代兼容，目前填 0
nSerialNo	包序列号，用于对应请求包和回包，要求递增
nBodyLen	包体长度
arrBodySHA1	包体原始数据(解密后)的 SHA1 哈希值
arrReserved	保留 8 字节扩展

> **提示**

- u8_t 表示 8 位无符号整数，u32_t 表示 32 位无符号整数
- OpenD 内部处理使用 Protobuf，因此协议格式建议使用 Protobuf，减少 Json 转换开销
- nProtoFmtType 字段指定了包体的数据类型，回包会回对应类型的数据；推送协议数据类型由 OpenD 配置文件指定
- **arrBodySHA1 用于校验请求数据在网络传输前后的一致性，必须正确填入**
- **协议头的二进制流使用的是小端字节序，即一般不需要使用 ntohl 等相关函数转换数据**

协议体

Protobuf 协议请求包体结构

```
message C2S
{
    required int64 req = 1;
}

message Request
{
    required C2S c2s = 1;
}
```

Protobuf 协议回应包体结构

```
message S2C
{
    required int64 data = 1;
}

message Response
{
    required int32 retType = 1 [default = -400]; //RetType，返回结果
    optional string retMsg = 2;
    optional int32 errCode = 3;
    optional S2C s2c = 4;
}
```

字段	说明
c2s	请求参数结构
req	请求参数，实际根据协议定义
retType	请求结果
retMsg	若请求失败，说明失败原因
errCode	若请求失败对应错误码
s2c	回应数据结构，部分协议不返回数据则无该字段
data	回应数据，实际根据协议定义

提示

- 包体格式类型请求包由协议头 nProtoFmtType 指定，OpenD 主动推送格式在 InitConnect 设置。
- 原始协议文件格式是以 Protobuf 格式定义，若需要 json 格式传输，建议使用 protobuf3 的接口直接转换成 json。
- 枚举值字段定义使用有符号整形，注释指明对应枚举，枚举一般定义于 Common.proto，Qot_Common.proto，Trd_Common.proto 文件中。
- 协议中价格、百分比等数据用浮点类型来传输，直接使用会有精度问题，需要根据精度（如协议中未指明，默认小数点后三位）做四舍五入之后再使用。

心跳保活

```
syntax = "proto2";
package KeepAlive;
option java_package = "com.moomoo.openapi.pb";
option go_package = "github.com/moomooopen/mmapi4go/pb/keepalive";

import "Common.proto";

message C2S
{
```

```
10      required int64 time = 1; //客户端发包时的格林威治时间戳，单位秒
11  }
12
13  message S2C
14  {
15      required int64 time = 1; //服务器回包时的格林威治时间戳，单位秒
16  }
17
18  message Request
19  {
20      required C2S c2s = 1;
21  }
22
23  message Response
24  {
25      required int32 retType = 1 [default = -400]; //RetType,返回结果
26      optional string retMsg = 2;
27      optional int32 errCode = 3;
28
29      optional S2C s2c = 4;
30  }
```

- **介绍**

  心跳保活

- **协议 ID**

  1004

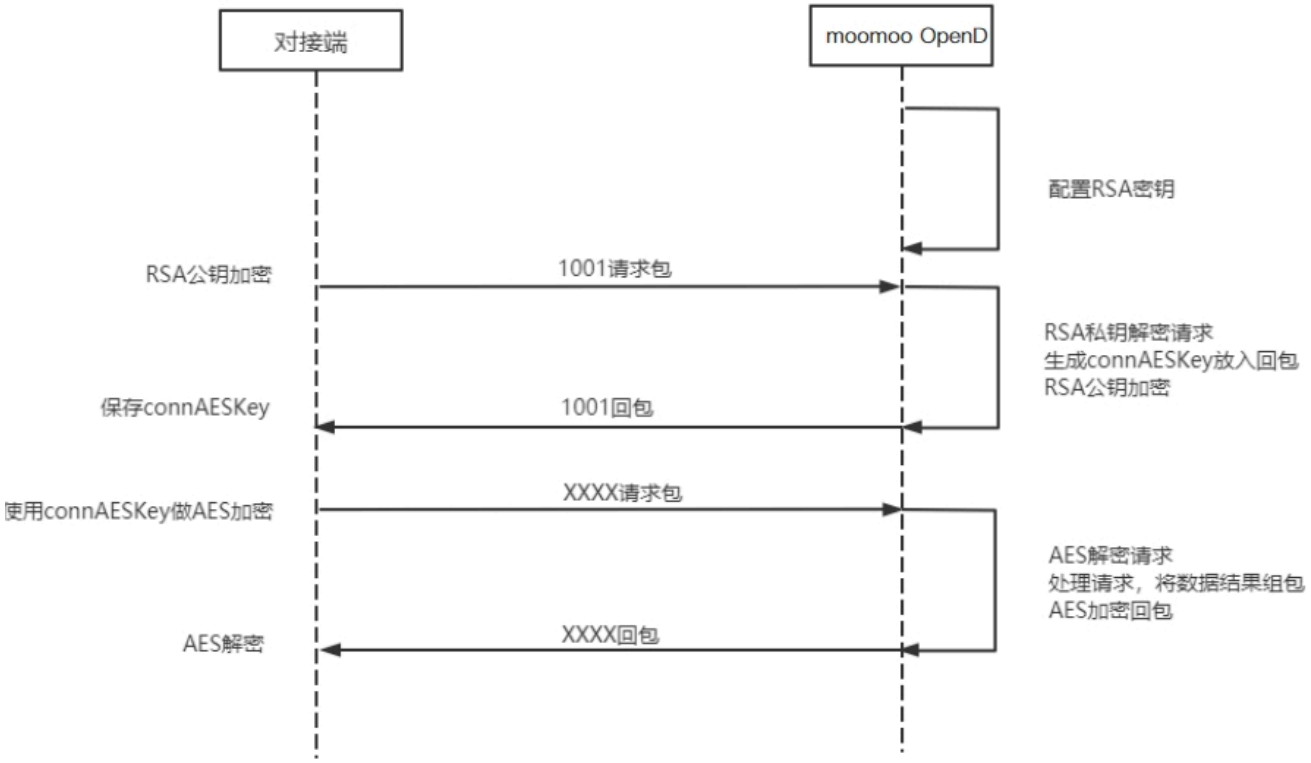
- **使用**

  根据初始化链接返回的心跳保活间隔时间，向 OpenD 发送保活协议

加密通信流程

- 若 OpenD 配置了加密，InitConnect 初始化连接协议必须使用 RSA 公钥加密，后续其他协议使用 InitConnect 返回的随机密钥进行 AES 加密通信。
- OpenD 的加密流程借鉴了 SSL 协议，但考虑到一般是本地部署服务和应用，简化了相关流程，OpenD 与接入 Client 共用了同一个 RSA 私钥文件，请妥善保存和分发私钥文件。

- 可到这个 网址 在线生成随机 RSA 密钥对，密钥格式必须为 PCKS#1，密钥长度 512，1024 都可以，不要设置密码，将生成的私钥复制保存到文件中，然后将私钥文件路径配置到 OpenD 配置 约定的 rsa_private_key 配置项中。
- **建议有实盘交易的用户配置加密，避免账户和交易信息泄露。**



RSA 加解密

- OpenD 配置 约定 rsa_private_key 为私钥文件路径
- OpenD 与接入客户端共用相同的私钥文件
- RSA 加解密仅用于 InitConnect 请求，用于安全获取其它请求协议的对称加密 Key
- OpenD 的 RSA 密钥为 1024 位，填充方式 PKCS1，公钥加密，私钥解密，公钥可通过私钥生成
- Python API 参考实现：RsaCrypt 类的 encrypt / decrypt 接口

发送数据加密

- RSA 加密规则:若密钥位数是 key_size，单次加密串的最大长度为 (key_size)/8 - 11，目前位数 1024，一次加密长度可定为 100。
- 将明文数据分成一个或数个最长 100 字节的小段进行加密，拼接分段加密数据即为最终的 Body 加密数据。

接收数据解密

- RSA 解密同样遵循分段规则，对于 1024 位密钥，每小段待解密数据长度为 128 字节。
- 将密文数据分成一个或数个 128 字节长的小段进行解密，拼接分段解密数据即为最终的 Body 解密数据。

AES 加解密

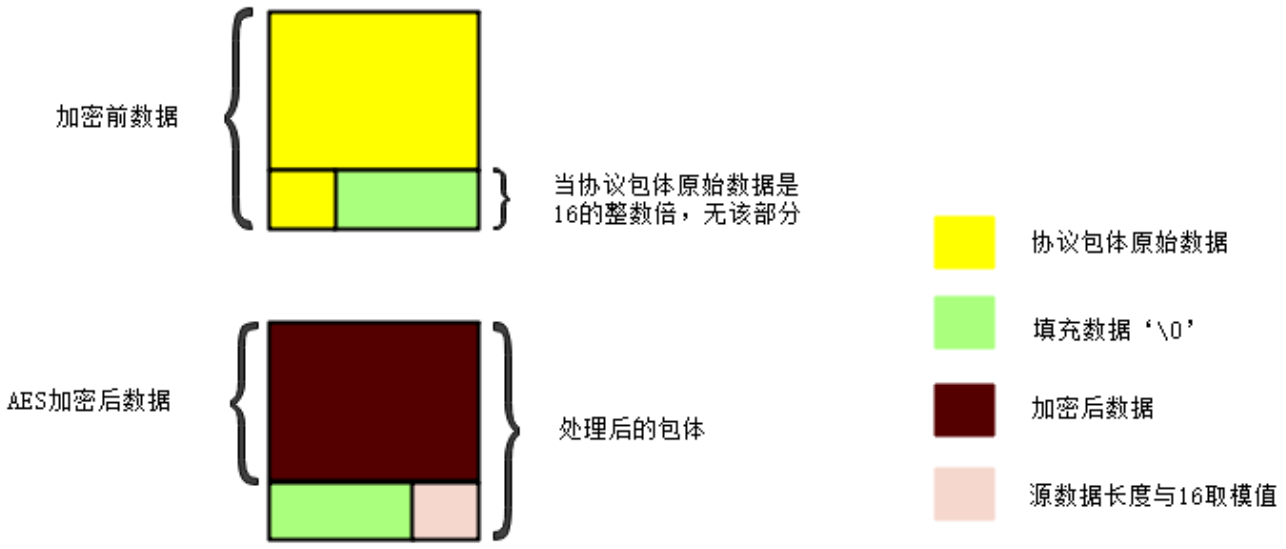
- 加密 key 由 InitConnect 协议返回
- 默认使用的是 AES 的 ecb 加密模式。
- Python API 参考实现: ConnMng 类的 encrypt_conn_data / decrypt_conn_data 接口

发送数据加密

- AES 加密要求源数据长度必须是 16 的整数倍，故需补'0'对齐后再加密，记录 mod_len 为源数据长度与 16 取模值。
- 因加密前有可能对源数据作修改，故需在加密后的数据尾再增加一个 16 字节的填充数据块，其最后一个字节赋值 mod_len，其余字节赋值'0'，将加密数据和额外的填充数据块拼接作为最终要发送协议的 body 数据。

接收数据解密

- 协议 body 数据，先将最后一个字节取出，记为 mod_len，然后将 body 截掉尾部 16 字节填充数据块后再解密（与加密填充额外数据块逻辑对应）。
- mod_len 为 0 时，上述解密后的数据即为协议返回的 body 数据，否则需截掉尾部(16 - mod_len)长度的用于填充对齐的数据。


加密前数据
当协议包体原始数据是
16的整数倍，无该部分
AES加密后数据
处理后的包体
协议包体原始数据
填充数据‘\0’
加密后数据
源数据长度与16取模值

OpenD 相关

Q1：OpenD 因未完成“问卷评估及协议确认”自动退出

A: 您需要进行相关问卷评估及协议确认，才可以使用 OpenD，请先 前往完成。

Q2：OpenD 因“程序自带数据不存在“退出

A: 一般因权限问题导致自带数据拷贝失败，可以尝试将程序目录下 *Appdata.dat* 解压后的文件拷贝到程序数据目录下。

- windows 程序数据目录: `%appdata%/com.moomoo.OpenD/F3CNN`
- 非 windows 程序数据目录: `~/.com.moomoo.OpenD/F3CNN`

Q3：OpenD 服务启动失败

A: 请检查：

1. 是否有其他程序占用所配置的端口；
2. 是否已经有配置了相同端口的 OpenD 在运行。

Q4：如何验证手机验证码？

A: 在 OpenD 界面上或远程到 Telnet 端口，输入命令 `input_phone_verify_code -code=123456` 。

> **提示**
>
> - 123456 是收到的手机验证码
> - -code=123456 前有空格

Q5：是否支持其他编程语言？

A: OpenD 有对外提供基于 socket 的协议，目前我们提供并维护 Python，C++，Java，C# 和 JavaScript 接口，下载入口。

如果上述语言仍不能满足您的需求，您可以自行对接 Protobuf 协议。

Q6：在同一设备多次验证设备锁

A: 设备标识随机生成并存放于

windows: %appdata%/com.moomoo.OpenD/F3CNN/Device.dat 文件中。 非windows: ~/.com.moomoo.OpenD/F3CNN/Device.dat

> **提示**
>
> 1. 如果文件被删除或损坏，OpenD 会重新生成新设备标识，然后验证设备锁。
> 2. 另外镜像拷贝部署的用户需要注意，如果多台机器的 Device.dat 内容相同，也会导致这些机器多次验证设备锁。删除 Device.dat 文件即可解决。

Q7：OpenD 是否有提供 Docker 镜像?

A: 目前没有提供。

Q8：一个账号可以登录多个 OpenD 吗?

A: 一个账号可以在多台机器上登录 OpenD 或者其他客户终端，最多允许 10 个 OpenD 终端同时登录。同时有"行情互踢"的限制，只能有一个 OpenD 获得最高权限行情。例如：两个终端登录同一个账号，只能有一个港股 LV2 行情，另一个是港股 BMP 行情。

Q9：如何控制 OpenD 和其他客户端（桌面端和移动端）的行情权限?

A: 应交易所的规定，多个终端同时在线会有"行情互踢"的限制，只能有一个终端获得最高权限行情。OpenD 命令行版本的启动参数中，内置了 auto_hold_quote_right 参数，用于灵活配置行情权限。当该参数选项开启时，OpenD 在行情权限被抢后，会自动抢回。如果 10 秒内再次被抢，则其他终端获得最高行情权限（OpenD 不会再抢）。

Q10：如何优先保证 OpenD 行情权限?

A:

1. 将 OpenD 启动参数 auto_hold_quote_right 配置为 1；
2. 保证不要在移动端或桌面端富途牛牛上在 10 秒内连续两次抢最高权限（登录算一次，点击“重启行情”算第二次）。

Q11：如何优先保证移动端（或桌面端）的行情权限?

A: OpenD 启动参数 auto_hold_quote_right 设置为 0，移动端或桌面端富途牛牛在 OpenD 之后登录即可。

Q12：使用可视化 OpenD 记住密码登录，长时间挂机后提示连接断开，需要重新登录?

A: 使用可视化 OpenD，如果选择记住密码登录，用的是记录在本地的令牌。由于令牌有时间限制，当令牌过期后，如果出现网络波动或富途后台发布，就可能导致与后台断开连接后无法自动连接上的情况。因此，可视化 OpenD 如果希望长时间挂机，建议手动输入密码登录，由 OpenD 自动处理该情况。

Q13：遇到产品缺陷，如何请富途的研发工程师排查日志?

A:

1. 与客服沟通问题表现，详述：发生错误的时间、OpenD 版本号、 API 版本号、脚本语言名、接口名或协议号、含详细入参和返回的短代码或截图。
2. 客服确认是产品缺陷后，如需进一步日志排查，研发工程师会主动联系。
3. 部分问题须提供 OpenD 日志，方便定位确认问题。交易类问题需要 info 日志级别，行情类问题需要 debug 日志级别。日志级别 log_level 可以在 *OpenD.xml* 中 配置 ，配置后需要重启 OpenD 方能生效，待问题复现后，将该段日志打包发给富途研发工程师。

> **提示**
>
> 日志路径如下：
> windows： `%appdata%/com.moomoo.OpenD/Log`
>
> 非 windows： `~/.com.moomoo.OpenD/Log`

Q14：脚本连接不上 OpenD

A: 请先尝试检查：

1. 脚本连接的端口与 OpenD 配置的端口是否一致。
2. 由于 OpenD 连接上限为 128，是否有无用连接未关闭。
3. 检查监听地址是否正确，如果脚本和 OpenD 不在同一机器，OpenD 监听地址需要设置成 0.0.0.0 。

Q15：连接上一段时间后断开

A: 如果是自己对接协议，检查下是否有定时发送心跳维持连接。

Q16：Linux 下通过 multiprocessing 模块以多进程方式运行 Python 脚本，连不上 OpenD?

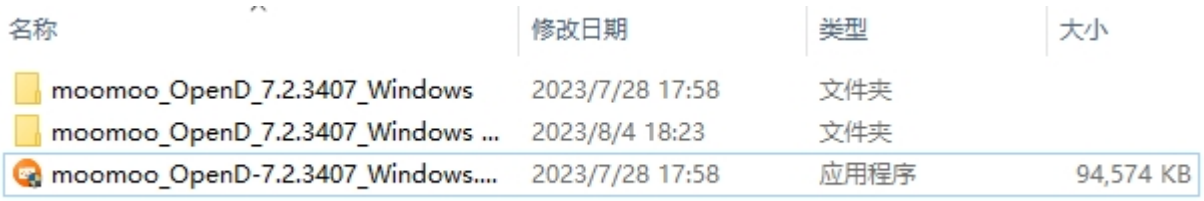
A: Linux/Mac 环境下以默认方式创建进程后，父进程中 py-moomoo-api 内部创建的线程将会在子进程中消失，导致程序内部状态错误。
可以用 spawn 方式来启动进程：

```py
import multiprocessing as mp
mp.set_start_method('spawn')
p = mp.Process(target=func)
```

Q17：如何在一台电脑同时登录两个 OpenD?

A: 可视化 OpenD 不支持，命令行 OpenD 支持。

1. 解压从官网下载的文件，复制整个命令行 OpenD 文件夹（如 OpenD_5.2.1408_Windows）得到副本（此处以 Windows 为例，其他系统可采取相同操作）。

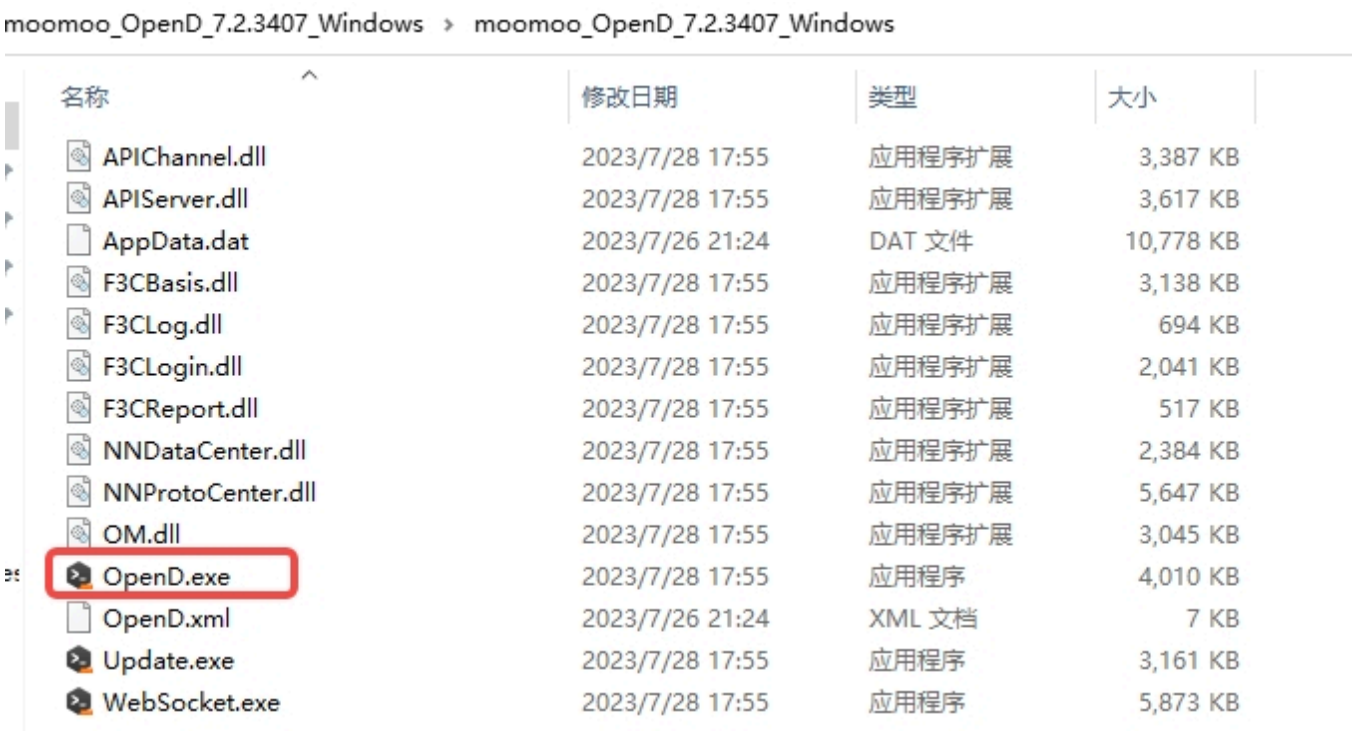


2. 分别打开两个命令行 OpenD 文件夹配置好两份 OpenD.xml 文件。

第一份配置文件参数：api_port = 11111，login_account = 登录账号1，login_pwd = 登录密码1

第二份配置文件参数：api_port = 11112，login_account = 登录账号2，login_pwd = 登录密码2

3. 配置完成后，分别打开两个 OpenD 程序运行。



4. 调用接口时，注意接口的参数 **`port`** （OpenD 监听端口）与 OpenD.xml 文件中的参数 **`api_port`** 为对应关系

   例如：

```py
from moomoo import *

# 向账号1登录的 OpenD 进行请求
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11111, is_encrypt=False)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽

# 向账号2登录的 OpenD 进行请求
quote_ctx = OpenQuoteContext(host='127.0.0.1', port=11112, is_encrypt=False)
quote_ctx.close() # 结束后记得关闭当条连接，防止连接条数用尽
```

Q18：行情权限被其他客户端踢掉，如何通过脚本执行抢权限的运维命令？

A：

1. 在OpenD启动参数中，配置好 Telnet 地址和 Telnet 端口。

FutuOpenD.xml

```
<futu_opend>
    <!-- 基础参数 -->
    <!-- Basic parameters -->
        <!-- 协议监听地址,不填默认127.0.0.1 -->
        <!-- Listening address. 127.0.0.1 if not specified -->  // Listening address. 127.0.0.1 by default
        <ip>127.0.0.1</ip>
        <!-- API接口协议监听端口 -->
        <!-- API interface protocol listening port -->
        <api_port>11111</api_port>
        <!-- 登录帐号 -->
        <login_account>100000</login_account>
        <!-- 登录密码32位MD5加密16进制 -->
        <!-- <login_pwd_md5>6e55f158a827b1a1c4321a245aaaad88</login_pwd_md5> -->
        <!-- 登录密码明文，密码密文存在情况下只使用密文 -->
        <login_pwd>123456</login_pwd>
        <!-- FutuOpenD语言，en：英文，chs：简体中文 -->
        <lang>chs</lang>
    <!-- 进阶参数 -->
    <!-- Advanced parameters -->
        <!-- FutuOpenD日志等级，no,debug,info,warning,error,fatal -->
        <log_level>info</log_level>
        <!-- API推送协议格式，0:pb, 1:json -->
        <!-- API push protocol format, 0:pb, 1:json -->
        <push_proto_type>0</push_proto_type>
        <!-- API订阅数据推送频率控制，单位毫秒，目前不包括K线和分时，不设置则不限制频率-->
        <!-- API subscription data push frequency control, in milliseconds, currently does not include K-line and time-sharing, if not
        <!-- <qot_push_frequency>1000</qot_push_frequency> -->
        <!-- Telnet监听地址,不填默认127.0.0.1 -->
        <!-- Telnet listening address, default 127.0.0.1 if not filled in -->  // Telnet listening address, 127.0.0.1 by default
        <telnet_ip>127.0.0.1</telnet_ip>
        <!-- Telnet监听端口 -->
        <!-- Telnet listening port -->
        <telnet_port>22222</telnet_port>
        <!-- API协议加密私钥文件路径,不设置则不加密 -->
        <!-- API protocol encrypted private key file path, if not set, it will not be encrypted -->  // File path for private key for
        <!-- <rsa_private_key>D:\rsa</rsa_private_key> -->
        <!-- 是否接收到价提醒推送，0：不接收，1：接收 -->
        <!-- Whether to receive the price reminder push, 0: not receive, 1: receive -->
```

2. 启动 OpenD（会同时启动 Telnet）。
3. 当发现行情权限被抢之后，您可以参考如下代码示例，通过 Telnet，向 OpenD 发送 `request_highest_quote_right` 命令。

```py
from telnetlib import Telnet
with Telnet('127.0.0.1', 22222) as tn:  # Telnet 地址为：127.0.0.1，Telnet 端口为:
    tn.write(b'request_highest_quote_right\r\n')
    reply = b''
    while True:
        msg = tn.read_until(b'\r\n', timeout=0.5)
        reply += msg
        if msg == b'':
            break
    print(reply.decode('gb2312'))
```

Q19：OpenD 自动升级失败

A： 通过 `update` 命令执行 OpenD 自动更新失败，可能的原因：

- 文件被其他进程占用：可以尝试关闭其他 OpenD 进程，或者重启系统后，再次执行 `update` 如果以上仍无法解决，可以通过官网自行下载更新。

Q20：ubuntu22无法启动可视化 OpenD?

A： 在有些Linux发行版（例如Ubuntu 22.04）运行可视化OpenD时，可能会提示： `dlopen(): error loading libfuse.so.2` 。 这是因为这些系统没有默认安装libfuse。通常可以手动安装来解决，例如对于Ubuntu22.04，可以在命令行运行：

```
sudo apt update
sudo apt install -y libfuse2
```

安装成功后就可以正常运行可视化OpenD了。详细信息请参考：https://docs.appimage.org/user-guide/troubleshooting/fuse.html。

Q21：Linux上如何在后台运行命令行OpenD?

A：先切到 OpenD 所在目录，配置好 OpenD.xml 之后，执行如下命令

```
nohup ./moomoo OpenD &
```

行情相关

Q1：订阅失败

A: 订阅接口返回错误，有以下两类常见情况：

- 订阅额度不足：

  订阅额度规则参见 订阅额度 & 历史 K 线额度

- 订阅权限不足：

  支持订阅的行情权限见下表

市场	品种	支持订阅的行情权限
香港市场	股票	LV1, LV2, SF
	期权	LV1, LV2
	期货	LV1, LV2
美国市场	股票	LV1, LV2
	期权	LV1
	期货	LV1, LV2
A 股市场	股票	LV1

  获取行情权限的方式参见 行情权限

  注意：若账号拥有上述权限，但仍订阅失败，可能存在被其他终端 踢掉行情权限 的情况。

Q2：反订阅失败

A: 订阅至少一分钟后才能反订阅。

Q3：反订阅成功但没释放额度

A: 所有连接都对该行情反订阅，才会释放额度。

举例：A 连接和 B 连接都在订阅 HK.00700 的摆盘，当 A 连接反订阅后，由于 B 连接仍在调用腾讯的摆盘数据，因此 OpenD 的额度不会释放，直至所有连接都反订阅 HK.00700 的摆盘。

Q4：订阅不足一分钟关闭脚本连接，会释放额度吗?

A: 不会。连接关闭后，订阅时长不足一分钟的标的类型，会在达到一分钟后才自动反订阅，并释放相应的订阅额度。

Q5：请求限频的具体限制逻辑是怎样?

A: 30 秒内最多 n 次，是指第 1 次和第 n+1 次请求间隔需要大于 30 秒。

Q6：自选股添加不上是什么原因?

A: 请先检查是否有超出上限，或者删除一部分自选。

Q7：为什么 OpenAPI 端的美股报价和牛牛显示端的全美综合报价有不同?

A: 由于美股交易分散在很多家交易所，富途有提供两种美股基本报价行情，一种是 Nasdaq Basic（Nasdaq 交易所的报价），另一种是全美综合报价（全美13家交易所的报价）。而 OpenAPI 的美股正股行情目前仅支持通过行情卡购买的方式获取 Nasdaq Basic，不支持全美综合报价。因此，如果您同时购买了显示端的全美综合报价行情卡，和仅用于 OpenAPI 的 Nasdaq Basic 行情卡，确实有可能出现牛牛显示端和 OpenAPI 端的报价差异。
因此，如果您发现美股当天开盘价与客户端显示不一致，这是因为OpenAPI实时上游行情仅会获取 Nasdaq Basic 数据。

Q8：OpenAPI 行情卡在哪里购买?

A:

- 港股市场
  - 港股 LV2 高级行情（仅港澳台及海外 IP）
  - 港股 LV2 + 期权期货 LV2 行情（仅港澳台及海外 IP）
- 美股市场
  - Nasdaq Basic
  - Nasdaq Basic+TotalView (Non-Pro)
  - Nasdaq Basic+TotalView (Pro)
  - 期权 OPRA 实时行情

Q9：为什么有时候，获取实时数据的 get 接口响应比较慢?

A: 因为获取实时数据的 get 接口需要先订阅，并依赖后台给 OpenD 的推送。如果用户刚订阅就立刻用 get 接口请求，OpenD 有可能尚未收到后台推送。为了防止这种情况的发生，get 接口内置了等待逻辑，3 秒内收到推送会立刻返回给脚本，超过 3 秒仍未收到后台推送，才会给脚本返回空数据。
涉及的 get 接口包括：get_rt_ticker、get_rt_data、get_cur_kline、get_order_book、get_broker_queue、get_stock_quote。因此，当发现获取实时数据的 get 接口响应比较慢时，可以先检查一下是否是无成交数据的原因。

Q10：购买 OpenAPI 美股 Nasdaq Basic 行情卡后，可以获取哪些数据?

A: Nasdaq Basic 行情卡购买激活后，可以获取的品类涵盖 Nasdaq、NYSE、NYSE MKT 交易所上市证券（包括美股正股和 ETF，不包括美股期货和美股期权）。
支持的数据接口包括：快照，历史 K 线，实时逐笔订阅，实时一档摆盘订阅，实时 K 线订阅，实时报价订阅，实时分时订阅，到价提醒。

Q11：各个行情品类的摆盘支持多少档?

A:

行情品类	LV1	LV2	SF
港股（含正股、窝轮、牛熊、界内证）	/	10	全盘+千笔明细
港股期权期货	1	10	/
美股（含 ETF）	1	60档	/
美股期权	1	/	/
美股期货	/	40档	/
A 股	5	/	/

Q12：为什么我购买激活了行情卡之后，OpenD 仍然没有行情权限?

A:

1. 由于 OpenAPI 的行情权限跟 APP 的行情权限不完全一样，部分行情卡仅适用于 APP 端（例如：OpenAPI美股行情卡需单独购买）。请先确认您所购买的行情卡是否是 OpenD 适用的。我们已将 OpenAPI 适用的 **所有** 行情卡列在《权限与限制》一节，请点击 这里 查看。
2. 行情卡购买激活成功后，是立即生效的。请 **重新启动 OpenD** 后，再次查看权限状态。

Q13：如何通过订阅接口获取实时行情?

第一步：订阅

将标的的代码和数据类型传入 订阅接口，完成订阅。

订阅接口支持了实时报价、实时摆盘、实时逐笔、实时分时、实时 K 线、实时经纪队列数据的获取。订阅成功后，OpenD 会持续收到富途服务器的实时数据推送。

注意：订阅额度会根据您的总资产、交易笔数和交易量，来进行分配，具体规则参见 订阅额度 & 历史 K 线额度。所以，如果您的订阅额度不足，可以先检查一下是否有无用的订阅在占用额度，及时 反订阅 即可释放已占用的订阅额度。

第二步：取数据

如何将订阅推送的数据从 OpenD 取回脚本呢？我们提供了如下两种方式：

方式 1：实时数据回调
设置相应的回调函数，来异步处理 OpenD 收到的数据推送。

设置好回调函数后，OpenD 会将收到的实时数据，立即推给脚本的回调函数进行处理。

如果所订阅的标的比较活跃，此时的推送数据可能数据量较大且频率较高。如果您希望适当降低 OpenD 给脚本的推送频率，建议在 OpenD 启动参数 中配置 API 推送频率（`qot_push_frequency`）。

方式 1 涉及的接口包括：实时报价回调、实时摆盘回调、实时 K 线回调、实时分时回调、实时逐笔回调、实时经纪队列回调。

方式 2：获取实时数据
通过获取实时数据接口，可以将 OpenD 收到的最新的数据，取回脚本。这种方式更加灵活，脚本不需要处理海量的推送。只要 OpenD 在持续接收富途服务器的推送，脚本可以随用随取，不用不取。

由于是从 OpenD 接收的推送数据中取，所以这类接口没有频率限制。

方式 2 涉及的接口包括：获取实时报价、获取实时摆盘、获取实时 K 线、获取实时分时、获取实时逐笔、获取实时经纪队列。

Q14：各个市场状态对应什么时间段?

A:

市场	品类	市场状态	时间段（当地时间）
香港市场	证券类产品（含股票、ETFs、窝轮、牛熊、界内证）	* NONE：无交易	CST 08:55 - 09:00
		* AUCTION：盘前竞价	CST 09:00 - 09:20
		* WAITING_OPEN：等待开盘	CST 09:20 - 09:30

		* MORNING：早盘	CST 09:30 - 12:00
		* REST: 午间休市	CST 12:00 - 13:00
		* AFTERNOON：午盘	CST 13:00 - 16:00
		* HK_CAS：港股盘后竞价（港股市场增加 CAS 机制对应的市场状态）	CST 16:00 - 16:08
		* CLOSED：收盘	CST 16:08 - 08:55（T+1）
	期权、期货（仅日市）	* NONE：期权待开盘	CST 08:55 - 09:30
		* MORNING：早盘	CST 09:30 - 12:00
		* REST: 午间休市	CST 12:00 - 13:00
		* AFTERNOON：午盘	CST 13:00 - 16:00
		* CLOSED：收盘	CST 16:00 - 08:55（T+1）
	期货（日夜市）	* FUTURE_DAY_WAIT_FOR_OPEN：期货待开盘	不同品种交易时间不同
		* NIGHT_OPEN: 夜市交易时段	
		* NIGHT_END：夜市收盘	
		* FUTURE_DAY_WAIT_FOR_OPEN：期货待开盘	

		* FUTURE_DAY_OPEN：日市交易时段	
		* FUTURE_DAY_CLOSE：日市收盘	
美国市场	证券类产品（含股票、ETFs）	* PRE_MARKET_BEGIN：美股盘前交易时段	EST 04:00 - 09:30
		* AFTERNOON：美股持续交易时段	EST 09:30 - 16:00
		* AFTER_HOURS_BEGIN：美股盘后交易时段	EST 16:00 - 20:00
		* AFTER_HOURS_END：美股盘后收盘	EST 20:00 - 04:00（T+1）
		* OVERNIGHT：美股夜盘交易时段	EST 20:00 - 04:00（T+1）
	期权	* NONE：期权待开盘	不同品种交易时间不同
		* REST：美指期权午间休市	
		* AFTERNOON：美股持续交易时段	
		* TRADE_AT_LAST：美指期权盘尾交易时段	
		* NIGHT：美指期权夜市交易时段	
		* CLOSED：收盘	
	期货	* FUTURE_SWITCH_DATE：美期待开盘	不同品种交易时间不同
		* FUTURE_OPEN：美期交易时段	
		* FUTURE_BREAK：美期中盘休息	
		* FUTRUE_BREAK_OVER：美期休息后交易时段	

		* FUTURE_CLOSE：美期收盘	
A股市场	证券类产品（含股票、ETFs）	* NONE：无交易	CST 08:55 - 09:15
		* Auction：盘前竞价	CST 09:15 - 09:25
		* WAITING_OPEN：等待开盘	CST 09:25 - 09:30
		* MORNING：早盘	CST 09:30 - 11:30
		* REST：午间休市	CST 11:30 - 13:00
		* AFTERNOON：午盘	CST 13:00 - 15:00
		* CLOSED：收盘	CST 15:00 - 08:55（T+1）
新加坡市场	期货	* FUTURE_DAY_WAIT_FOR_OPEN：期货待开盘	不同品种交易时间不同
		* NIGHT_OPEN：夜市交易时段	
		* NIGHT_END：夜市收盘	
		* FUTURE_DAY_OPEN：日市交易时段	
		* FUTURE_DAY_CLOSE：日市收盘	
日本市场	期货	* FUTURE_DAY_WAIT_FOR_OPEN：期货待开盘	JST 16:25（T-1）- 16:30（T-1）
		* NIGHT_OPEN：夜市交易时段	JST 16:30（T-1） - 05:30

		* NIGHT_END：夜市收盘	JST 05:30 - 08:45
		* FUTURE_DAY_OPEN：日市交易时段	JST 08:45 - 15:15
		* FUTURE_DAY_CLOSE：日市收盘	JST 15:15 - 16:25

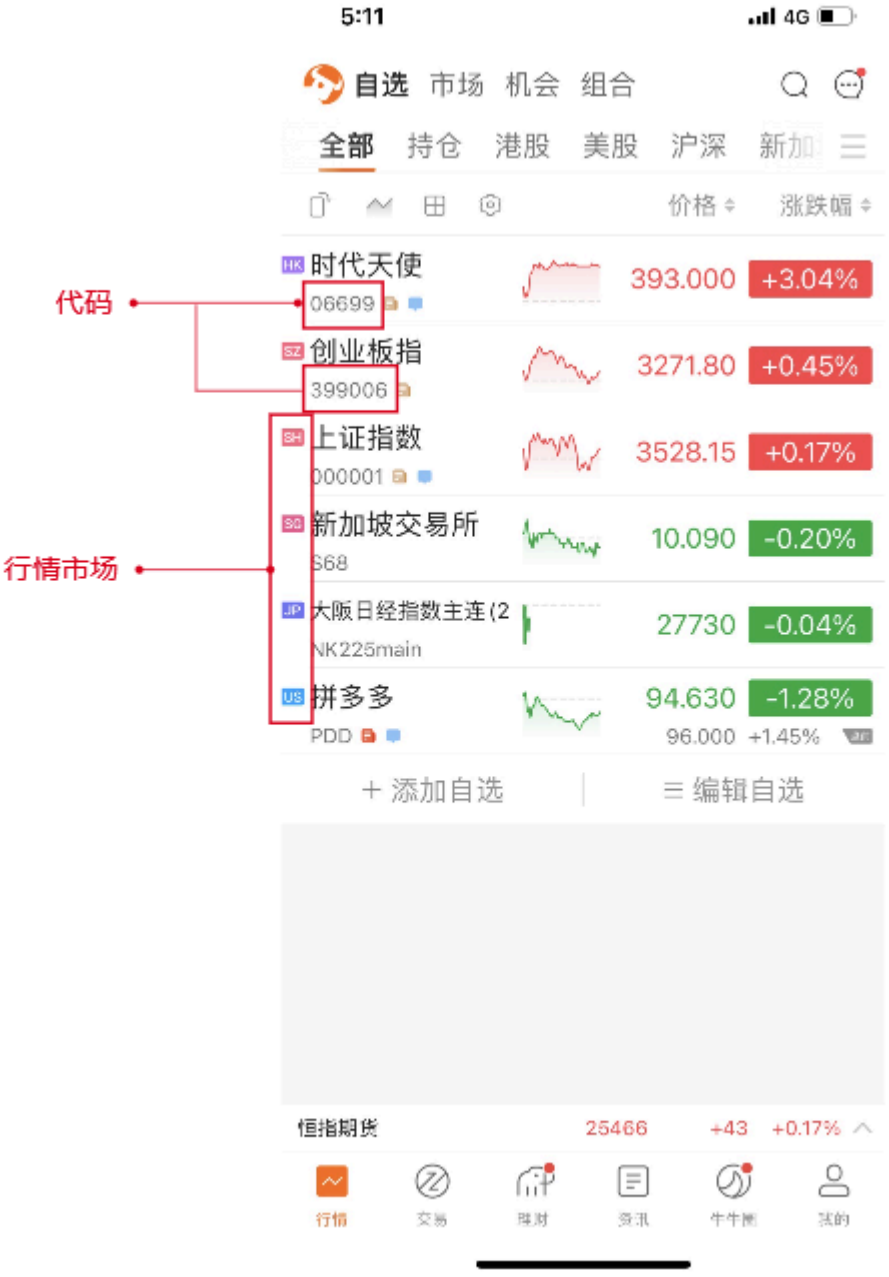
* CST, EST, JST 分别表示中国时间，美东时间，日本时间

Q15：接口参数股票代码的格式

A：

- 使用不同编程语言的用户，需要的股票代码的格式不同：
  - **Python 用户**
    股票代码 code 格式：`行情市场.代码`。
    例如：腾讯控股，参数 code 传入'HK.00700'。
  - **非 Python 用户**
    股票结构参见 Security。
    例如：腾讯控股，参数 market 传入 QotMarket_HK_Security，参数 code 传入'00700'。
- 查询方式：
  通过 APP 查看代码和行情市场：行情 > 自选 > 全部。

行情市场定义，请参考 这里。



Q16：复权因子相关

A：

概述

所谓 复权 就是对股价和成交量进行权息修复，按照股票的实际涨跌绘制股价走势图，并把成交量调整为相同的股本口径。

公司行动（如：拆股、合股、送股、转增股、配股、增发股、分红）均可能对股价产生影响，而复权计算可对量价进行调整，剔除公司行动的影响，保持股价走势的连续性。

名词解释

- 公司行动：上市公司进行一些股权、股票等影响公司股价和股东持仓变化的行为。
- 前复权：保持现有的股价不变，以当前的股价为基准，对以前的股价进行复权计算。
- 后复权：保持先前的股价不变，以过去的股价为基准，对以后的股价进行复权计算。
- 复权因子：即权息修复比例，用于计算复权后的价格及持仓数量。
- 除权除息日：即股权登记日下一个交易日。在股票的除权除息日，证券交易所都要计算出股票的除权除息价，以作为股民在除权除息日开盘的参考。其意义是股票股利分配给股东的日期。

复权方法

主流的复权计算方法分为两种：事件法和连乘法；而 OpenAPI 针对不同市场使用不同的计算方法。

- 事件复权法：通过还原除权除息的各类事件进行复权；存在两个复权因子（复权因子 A 和 复权因子 B），复权因子 B 主要调整现金分红对股价的影响，而复权因子 A 调整其他公司行动对股价的影响。
- 连乘复权法：通过复权因子连乘的方式进行复权，只保留 复权因子 A（或将 复权因子 B 置为 0），复权因子 A 为 除权除息日前收盘价/该日经权息调整后的前收盘价。

> **提示**
>
> - OpenAPI 对美股前复权使用连乘法，即将 复权因子 B 置为0。
> - OpenAPI 对除美股以外的标的（A股、港股、新加坡股票等）及美股后复权使用事件法。

计算公式

单次复权

- 前复权：
  前复权价格 = 不复权价格 × 前复权因子 A + 前复权因子 B

- 后复权：
  后复权价格 = 不复权价格 × 后复权因子 A + 后复权因子 B

多次复权

- 前复权：按照时间顺序，筛选出大于计算日期的复权因子，优先使用时间较早的复权因子进行复权计算。以两次复权为例：

$$Price_n^{adjusted} = (Price_n * FactorA_{n+1}^{adjusted} + FactorB_{n+1}^{adjusted}) * FactorA_{n+2}^{adjusted} + FactorB_{n+2}^{adjusted}$$

$Price_n^{adjusted}$：被计算当天的前复权价格

$Price_n$：当天的不复权价格

$FactorA_{n+1}^{adjusted}$：后一天的前复权因子 A

$FactorB_{n+1}^{adjusted}$：后一天的前复权因子 B

$FactorA_{n+2}^{adjusted}$：后两天的前复权因子 A

$FactorB_{n+2}^{adjusted}$：后两天的前复权因子 B

- 后复权：按照时间倒序，筛选出小于等于计算日期的复权因子，优先使用时间较晚的复权因子进行复权计算。以两次复权为例：

$$Price_n^{cumulative} = (Price_n * FactorA_n^{cumulative} + FactorB_n^{backward}) * FactorA_{n-1}^{cumulative} + FactorB_{n-1}^{cumulative}$$

$Price_n^{cumulative}$：被计算当天的后复权价格

$Price_n$：当天的不复权价格

$FactorA_n^{cumulative}$：当天的后复权因子 A

$FactorB_n^{cumulative}$：当天的后复权因子 B

$FactorA_{n-1}^{cumulative}$：前一天的后复权因子 A

$FactorB_{n-1}^{cumulative}$：前一天的后复权因子 B

示例

单次前复权示例

以牧原股份为例：

- 筛选复权因子如下：

除权除息日	股票代码	方案说明	前复权因子 A	前复权因子 B
2021/06/03	SZ.002714	10转4.0股派14.61元（含税）	0.71429	-1.04357

- 不复权数据如下：

日期	股票代码	不复权收盘价
2021/06/02	SZ.002714	93.11
2021/06/03	SZ.002714	66.25

- 前复权数据如下：

日期	股票代码	前复权收盘价
2021/06/02	SZ.002714	65.4639719
2021/06/03	SZ.002714	66.25

- 前复权数据计算方法：
  牧原股份在 2021/06/03 进行拆股及现金分红行动（10转4.0股派14.61元），根据前复权计算公式对 2021/06/02 的收盘价进行调整计算，则：前复权价格（65.4639719） = 不复权价格（93.11） × 前复权因子 A（0.71429） + 前复权因子 B（-1.04357）

多次后复权示例

接上一个例子，计算牧原股份在 2021/06/02 的后复权价格：

- 筛选复权因子如下：

除权除息日	股票代码	方案说明	后复权因子A	后复权因子B
2014/07/04	SZ.002714	10派2.34元（含税）	1	0.234
2015-06-10	SZ.002714	10转10.0股派0.61元（含税）	2	0.061
2016-07-08	SZ.002714	10转10.0股派3.53元（含税）	2	0.353
2017-07-11	SZ.002714	10转8.0股派6.9元（含税）	1.8	0.69
2018-07-03	SZ.002714	10派6.91元（含税）	1	0.691
2019-07-04	SZ.002714	10派0.5元（含税）	1	0.05
2020-06-04	SZ.002714	10转7.0股派5.5元（含税）	1.7	0.55

- 不复权数据如下：

日期	股票代码	不复权收盘价
2021/06/02	SZ.002714	93.11

- 后复权数据如下：

日期	股票代码	后复权收盘价
2021/06/02	SZ.002714	1152.7226

- 后复权数据计算方法：
  为了计算牧原股份在 2021/06/02 的后复权价格，需要将早于 2021/06/02 的复权事件进行一一复权，得到最后的后复权价格，具体计算如下：

不复权价格

日期	股票代码	不复权收盘价
2021/06/02	SZ.002714	93.11

后复权价格

日期	股票代码	不复权收盘价
2021/06/02	SZ.002714	1152.7226

((((((93.11 x 1.7 + 0.55) x 1 + 0.05) x 1 + 0.691) x 1.8 + 0.69) x 2 + 0.353) x 2 + 0.061) x 1 + 0.234 = 1152.7226

复权因子

除权除息日	股票代码	方案说明	后复权因子 A	后复权因子 B
2014/07/04	SZ.002714	10派2.34元（含税）	1	0.234
2015-06-10	SZ.002714	10转10.0股派0.61元（含税）	2	0.061
2016-07-08	SZ.002714	10转10.0股派3.53元（含税）	2	0.353
2017-07-11	SZ.002714	10转8.0股派6.9元（含税）	1.8	0.69
2018-07-03	SZ.002714	10派6.91元（含税）	1	0.691
2019-07-04	SZ.002714	10派0.5元（含税）	1	0.05
2020-06-04	SZ.002714	10转7.0股派5.5元（含税）	1.7	0.55

交易相关

Q1：模拟交易相关

A:

概述

模拟交易是在真实的市场环境中，用虚拟资金做交易，不会对您的真实账户的资产造成影响。

交易时间

模拟交易仅支持在常规交易时段交易，不支持在非交易时段、美股盘前盘后时段、A股港股盘前盘后竞价时段交易。详情可点击 模拟交易规则。

支持品类

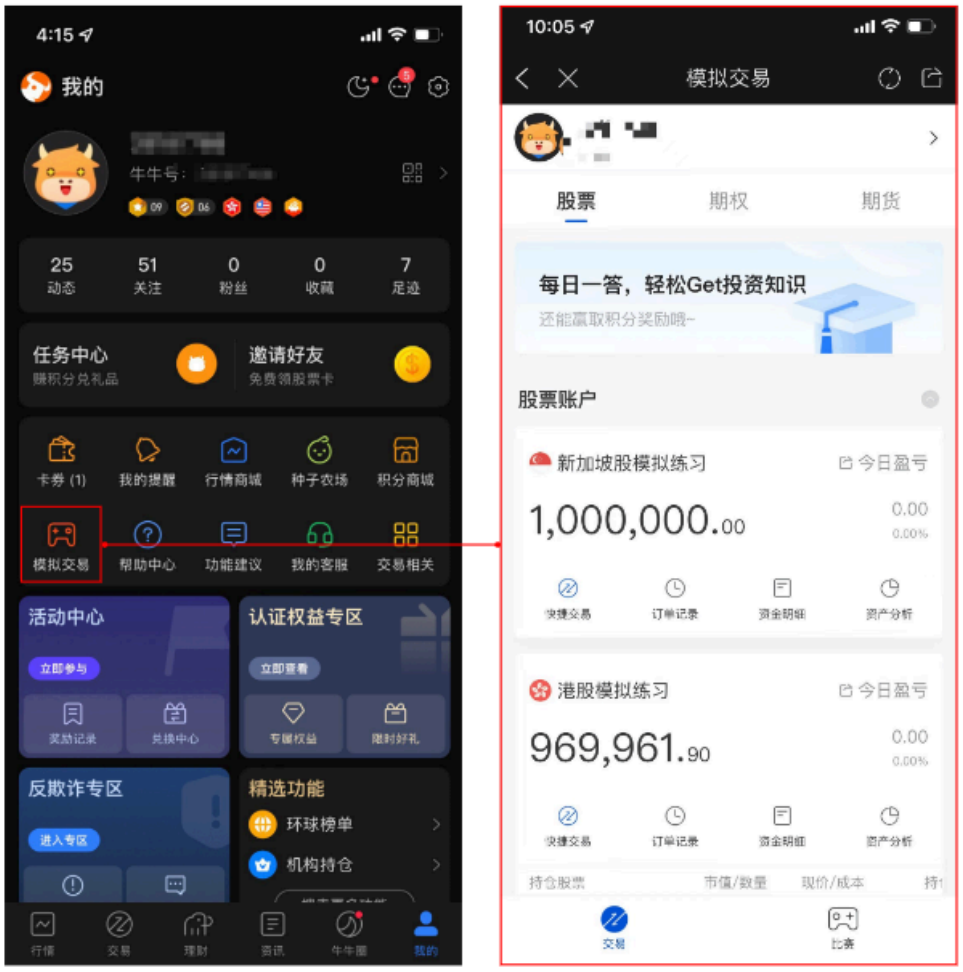
OpenAPI 支持模拟交易的品类请参考 这里。

订单

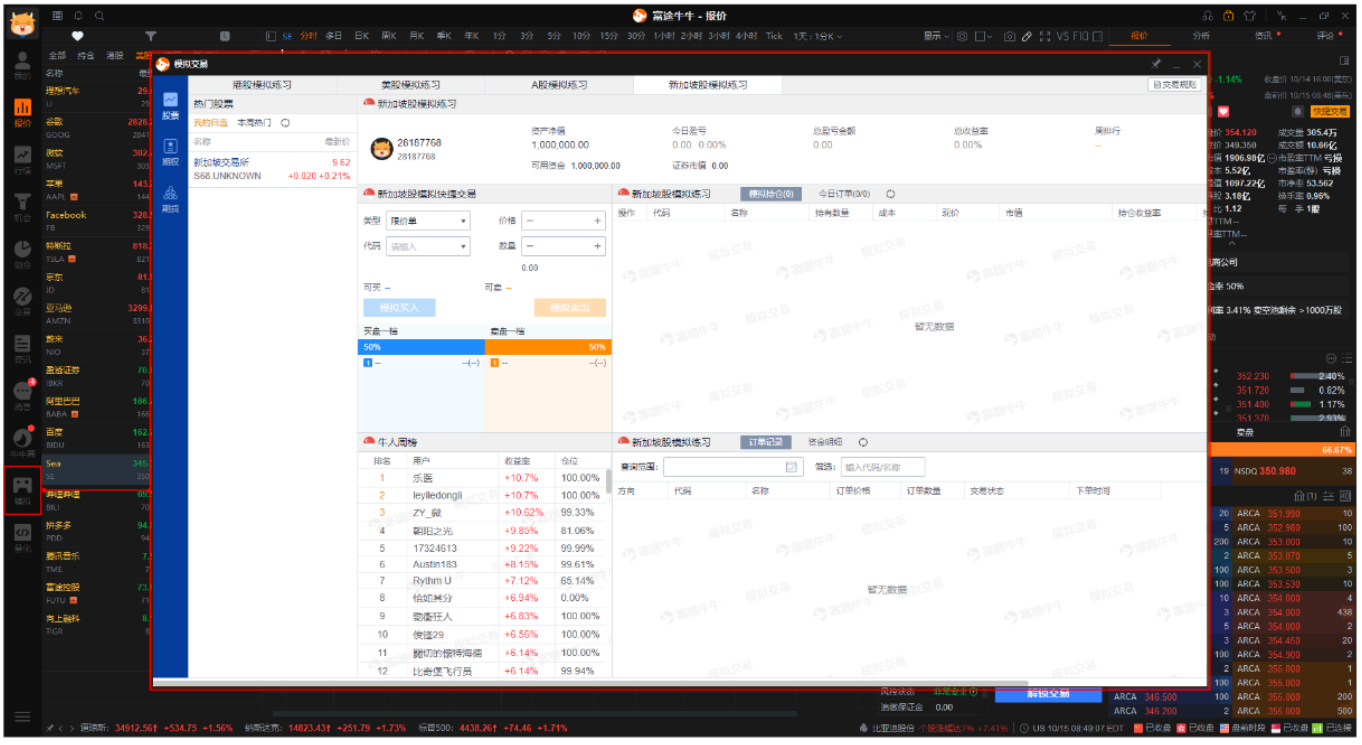
1. 订单类型：限价单和市价单。
2. 改单操作类型：模拟交易不支持使生效、使失效、删除，仅支持支持修改订单、 撤单。
3. 成交：模拟交易不支持成交相关操作，包括 查询今日成交、查询历史成交、响应成交推送回调。
4. 有效期限：模拟交易有效期限仅支持当日有效。
5. 卖空：期权和期货支持卖空。股票仅美股支持卖空。

操作平台

1. 移动端：我的 — 模拟交易



2. 桌面端：左侧模拟 tab



3. 网页端：模拟交易界面

4. OpenAPI：在调用接口时，设置参数交易环境为模拟环境即可。详见 如何使用 OpenAPI 进行模拟交易。

> **提示**
>
> - 以上四种方式只是操作平台不同，四种方式操作的模拟账户是共通的。

如何使用 OpenAPI 进行模拟交易?

创建连接

先根据交易品种 创建相应的连接 。当交易品种是股票或期权时，请使用 `OpenSecTradeContext` 。当交易品种是期货时，请使用 `OpenFutureTradeContext` 。

获取交易业务账户列表

使用 获取交易业务账户列表 查看交易账户（包括模拟账户、真实账户）。以 Python 为例：返回字段交易环境 `trd_env` 为 `SIMULATE` ，表示模拟账户。

- Example: Stocks and Options

```py
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, security_firm
#trd_ctx = OpenFutureTradeContext(host='127.0.0.1', port=11111, is_encrypt=None, security_firm=SecurityF
ret, data = trd_ctx.get_acc_list()
if ret == RET_OK:
    print(data)
    print(data['acc_id'][0])  # get the first account id
    print(data['acc_id'].values.tolist())  # convert to list format
else:
    print('get_acc_list error: ', data)
trd_ctx.close()
```

- Output

```py
              acc_id   trd_env acc_type          card_num   security_firm  \
0  281756480572583411      REAL   MARGIN  1001318721909873  FUTUSECURITIES
1             9053218  SIMULATE     CASH               N/A             N/A
2             9048221  SIMULATE   MARGIN               N/A             N/A

  sim_acc_type  trdmarket_auth
0          N/A  [HK, US, HKCC]
1        STOCK            [HK]
2       OPTION            [HK]
```

> **提示**
>
> - 模拟交易中，区分股票账户和期权账户，股票账户只能交易股票，期权账户只能交易期权；以 Python 为例：返回字段中模拟账户类型 `sim_acc_type` 为 `STOCK` ，表示股票账户；为 `OPTION` ，表示期权账户。

- Example: Futures

```py
from moomoo import *
#trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, security_fir
trd_ctx = OpenFutureTradeContext(host='127.0.0.1', port=11111, is_encrypt=None, security_firm=SecurityFi
ret, data = trd_ctx.get_acc_list()
if ret == RET_OK:
    print(data)
    print(data['acc_id'][0])  # get the first account id
    print(data['acc_id'].values.tolist())  # convert to list format
else:
    print('get_acc_list error: ', data)
trd_ctx.close()
```

- Output

```py
    acc_id   trd_env acc_type card_num security_firm sim_acc_type  \
0  9497808  SIMULATE   MARGIN      N/A           N/A      FUTURES
1  9497809  SIMULATE   MARGIN      N/A           N/A      FUTURES
2  9497810  SIMULATE   MARGIN      N/A           N/A      FUTURES
3  9497811  SIMULATE   MARGIN      N/A           N/A      FUTURES

          trdmarket_auth
0  [FUTURES_SIMULATE_HK]
1  [FUTURES_SIMULATE_US]
2  [FUTURES_SIMULATE_SG]
3  [FUTURES_SIMULATE_JP]
```

下单

使用 下单接口 时，设置交易环境为模拟环境即可。以 Python 为例：`trd_env = TrdEnv.SIMULATE`。

- Example

```py
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, security_firm
ret, data = trd_ctx.place_order(price=510.0, qty=100, code="HK.00700", trd_side=TrdSide.BUY, trd_env=Trd
if ret == RET_OK:
    print(data)
else:
    print('place_order error: ', data)
trd_ctx.close()
```

- Output

```py
     code stock_name    trd_side order_type    order_status  order_id qty  price     create_time    updat
0    HK.00700 腾讯控股  BUY  NORMAL    SUBMITTING    4642000476506964749    100.0    510.0    2021-10-09
```

撤单改单

使用 撤单接口 时，设置交易环境为模拟环境即可。以 Python 为例： `trd_env = TrdEnv.SIMULATE` 。

- Example

```py
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, security_firm
order_id = "4642000476506964749"
ret, data = trd_ctx.modify_order(ModifyOrderOp.CANCEL, order_id, 0, 0, trd_env=TrdEnv.SIMULATE)
if ret == RET_OK:
    print(data)
else:
    print('modify_order error: ', data)
trd_ctx.close()
```

- Output

```py
    trd_env             order_id
0  SIMULATE  4642000476506964749
```

查询历史订单

使用 查询历史订单接口 时，设置交易环境为模拟环境即可。以 Python 为例： `trd_env = TrdEnv.SIMULATE` 。

- Example

```py
from moomoo import *
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK, host='127.0.0.1', port=11111, security_firm
ret, data = trd_ctx.history_order_list_query(trd_env=TrdEnv.SIMULATE)
if ret == RET_OK:
    print(data)
else:
    print('history_order_list_query error: ', data)
trd_ctx.close()
```

- Output

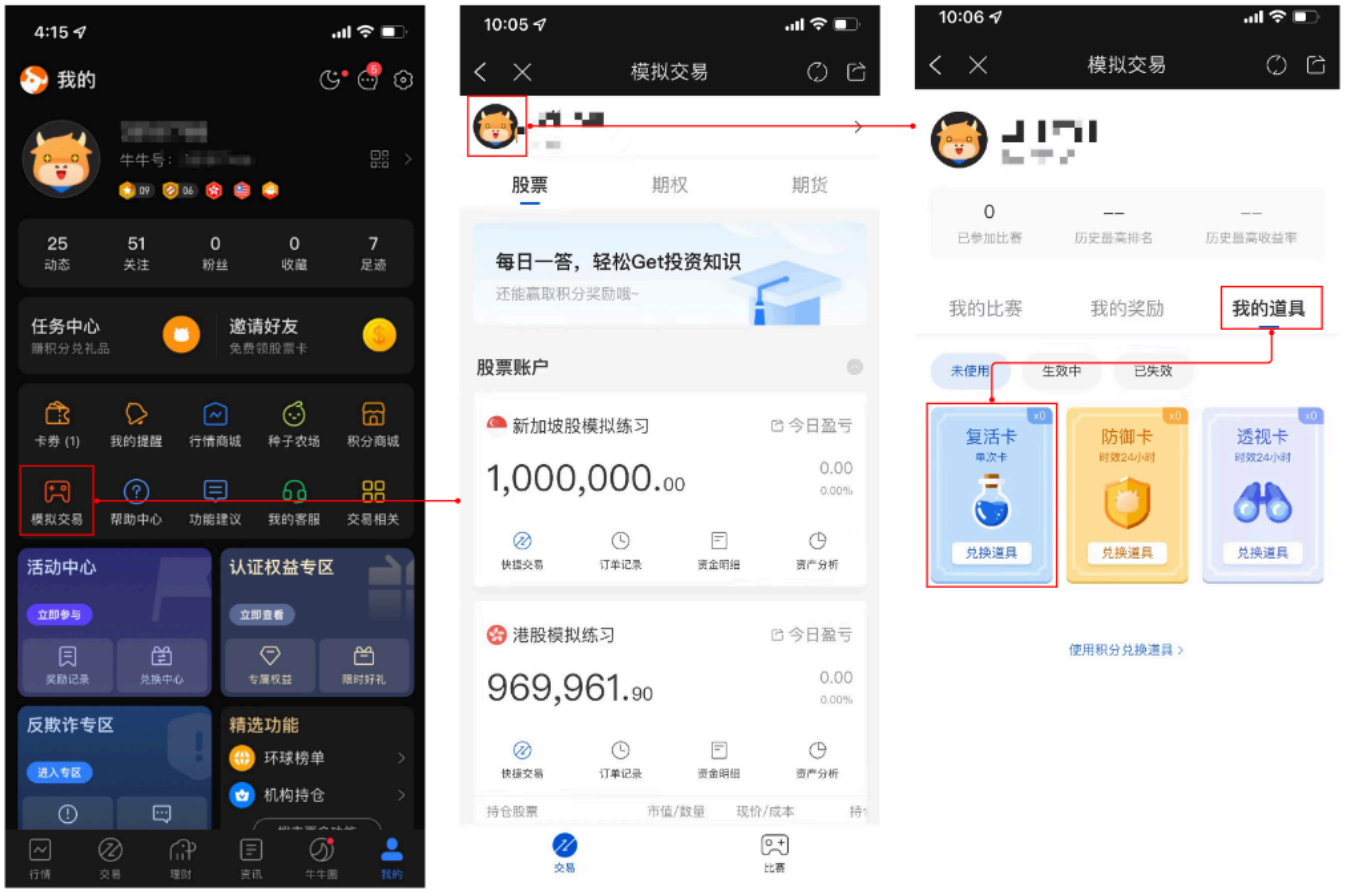
```py
     code stock_name    trd_side order_type    order_status  order_id qty  price    create_time   updat
0    HK.00700 腾讯控股  BUY  ABSOLUTE_LIMIT    CANCELLED_ALL 4642000476506964749    100.0    510.0
```

如何重置模拟账户?

目前 OpenAPI 不支持重置模拟账户，您可在移动端使用复活卡重置指定模拟账户，重置后账户资金将恢复至初始值，历史订单将会被清空。

具体操作

移动端：我的 — 模拟交易 — 我的头像 — 我的道具 — 复活卡。



Q2：是否支持 A 股交易?

A: 模拟交易支持 A 股交易。但真实交易仅可通过 A 股通交易部分 A 股，具体详见 A 股通名单。

Q3：各市场支持的交易方向

A: 除了期货，其他股票都只支持传入 BUY 和 SELL 两个交易方向。在空仓情况下传入 SELL，产生的订单交易方向是卖空。

Q4：真实交易中，各市场支持的订单类型

A:

市场	品种	限价单	市价单	竞价限价单	竞价市价单	绝对限价单	特别限价单	特别限价且要求全部	止损市价单	止损限价单	触及市价单（止盈）	触及限价单（止盈）	跟踪止损市价单	跟踪止损限价单

								成交订单					
香港市场	证券类产品（含股票、ETFs、窝轮、牛熊、界内证）	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	期权	✓	X	-	-	-	-	-	X	✓	X	✓	X	✓
	期货	✓	✓	-	✓	-	-	-	✓	✓	✓	✓	✓	✓
美国市场	证券类产品（含股票、ETFs）	✓	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓
	期权	✓	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓
	期货	✓	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓
A股通市场	证券类产品（含股票、ETFs）	✓	X	-	-	-	-	-	X	✓	X	✓	X	✓
新加坡市场	期货	✓	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓
日本市场	期货	✓	✓	-	-	-	-	-	✓	✓	✓	✓	✓	✓

Q5：各市场支持的订单操作

A:

- 港股支持改单、撤单、生效、失效、删除
- 美股仅支持改单和撤单
- A 股通仅支持撤单
- 期货支持改单、撤单、删除

Q6：OpenD 启动参数 future_trade_api_time_zone 如何使用?

A：由于期货账户支持交易的品种分布在全球多个交易所，交易所的所属时区各有不同，因此期货交易 API 的时间显示就成为了一个问题。
OpenD 启动参数中新增了 future_trade_api_time_zone 这一参数，供全球不同地区的期货交易者灵活指定时区。默认时区为 UTC+8，如果您更习惯美东时间，只需将此参数配置为 UTC-5 即可。

> **提示**
>
> - 此参数仅会对期货交易接口类对象生效。港股交易、美股交易、A 股通交易接口类对象的时区，仍然按照交易所所在的时区进行显示。
> - 此参数会影响的接口包括：响应订单推送回调，响应成交推送回调，查询今日订单，查询历史订单，查询当日成交，查询历史成交，下单。

Q7：通过 OpenAPI 下的订单，能在 APP 上面看到吗?

A：可以看到。
通过 OpenAPI 成功发出下单指令后，您可以在 APP 的 **交易** 页面，查看今日订单、订单状态、成交情况等等，也可以在 **消息—订单消息** 中收到成交提醒的通知。

Q8：哪些品类支持在非交易时段下单?

A：所有的订单，都需要在开盘期间才能够成交。
OpenAPI 仅对一部分品类，支持了 **非交易时段下单** 的功能（APP 上支持更多品类的非交易时段下单功能）。具体请参考下表：

市场	标的类型	模拟交易	真实交易						
			Futu HK	Moomoo US	Moomoo SG	Moomoo AU	Moomoo MY	Moomoo CA	Moomoo JP
香港市场	股票、ETFs、窝轮、牛熊、界内证	✓	✓	✓	✓	✓	✓	X	X
	期权	✓	✓	X	X	X	X	X	X
	期货	✓	✓	X	X	X	X	X	X

美国市场	股票、ETFs	✓	✓	✓	✓	✓	✓	✓	✓
	期权	✓	✓	✓	✓	✓	✓	✓	✓
	期货	✓	✓	X	✓	X	✓	X	X
A股市场	A 股通股票	✓	✓	✓	✓	X	X	X	X
	非 A 股通股票	✓	X	X	X	X	X	X	X
新加坡市场	股票、ETFs、窝轮、REITs、DLCs	X	X	X	X	X	X	X	X
	期货	✓	✓	X	✓	X	X	X	X
日本市场	股票、ETFs、REITs	X	X	X	X	X	X	X	X
	期货	✓	✓	X	X	X	X	X	X
澳大利亚市场	股票、ETFs	X	X	X	X	X	X	X	X
加拿大市场	股票	X	X	X	X	X	X	X	X

> **提示**
>
> - ✓：支持非交易时段下单
> - X：暂不支持非交易时段下单（或暂不支持交易）

Q9：对于下单接口，各订单类型对应的必传参数以及券商对单笔订单的

下单限制

A1: 各订单类型对应的必传参数

参数	限价单	市价单	竞价限价单	竞价市价单	绝对限价单	特别限价单	特别限价且要求全部成交订单	止损市价单	止损限价单	触及市价单（止盈）	触及限价单（止盈）	跟踪止损市价单	跟踪止损限价单
price	✓		✓		✓	✓	✓		✓		✓		
qty	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
code	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
trd_side	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
order_type	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
trd_env	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
aux_price								✓	✓	✓	✓		
trail_type												✓	✓
trail_value												✓	✓
trail_spread													✓

`Python 用户` 注意，place_order 并未对 price 设置默认值，对于上述五类订单类型，仍需对 price 传参，price 可以传入任意值。

A2：各券商对单笔订单的股数及金额限制

券商	品类	单笔订单股数上限	单笔订单金额上限
FUTU HK	A股通	1,000,000 股	￥5,000,000
	美股	500,000 股	$5,000,000
	香港股票期货/期权	3,000 手	无限制
moomoo US	美股	500,000 股	$10,000,000
moomoo SG	美股	500,000 股	$5,000,000
moomoo AU	美股	无限制	无限制

Q10：对于改单接口，修改订单时，各订单类型对应的必传参数

A:

参数	限价单	市价单	竞价限	竞价市	绝对限	特别限	特别限	止损市	止损限	触及市价单	触及限价单	跟踪止	跟踪止

			价单	价单	价单	价单	价且要求全部成交订单	价单	价单	(止盈)	(止盈)	损市价单	损限价单
modify_order_op	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
order_id	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
price	✓		✓		✓	✓	✓		✓		✓		
qty	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
trd_env	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
aux_price								✓	✓	✓	✓		
trail_type												✓	✓
trail_value												✓	✓
trail_spread													✓

`Python 用户` 注意，modify_order 并未对 price 设置默认值，对于上述五类订单类型，仍需对 price 传参，price 可以传入任意值。

Q11：交易接口返回“当前证券业务账户尚未同意免责协议”？

A：
点击下方链接完成协议确认，重启 OpenD 即可正常使用交易功能。

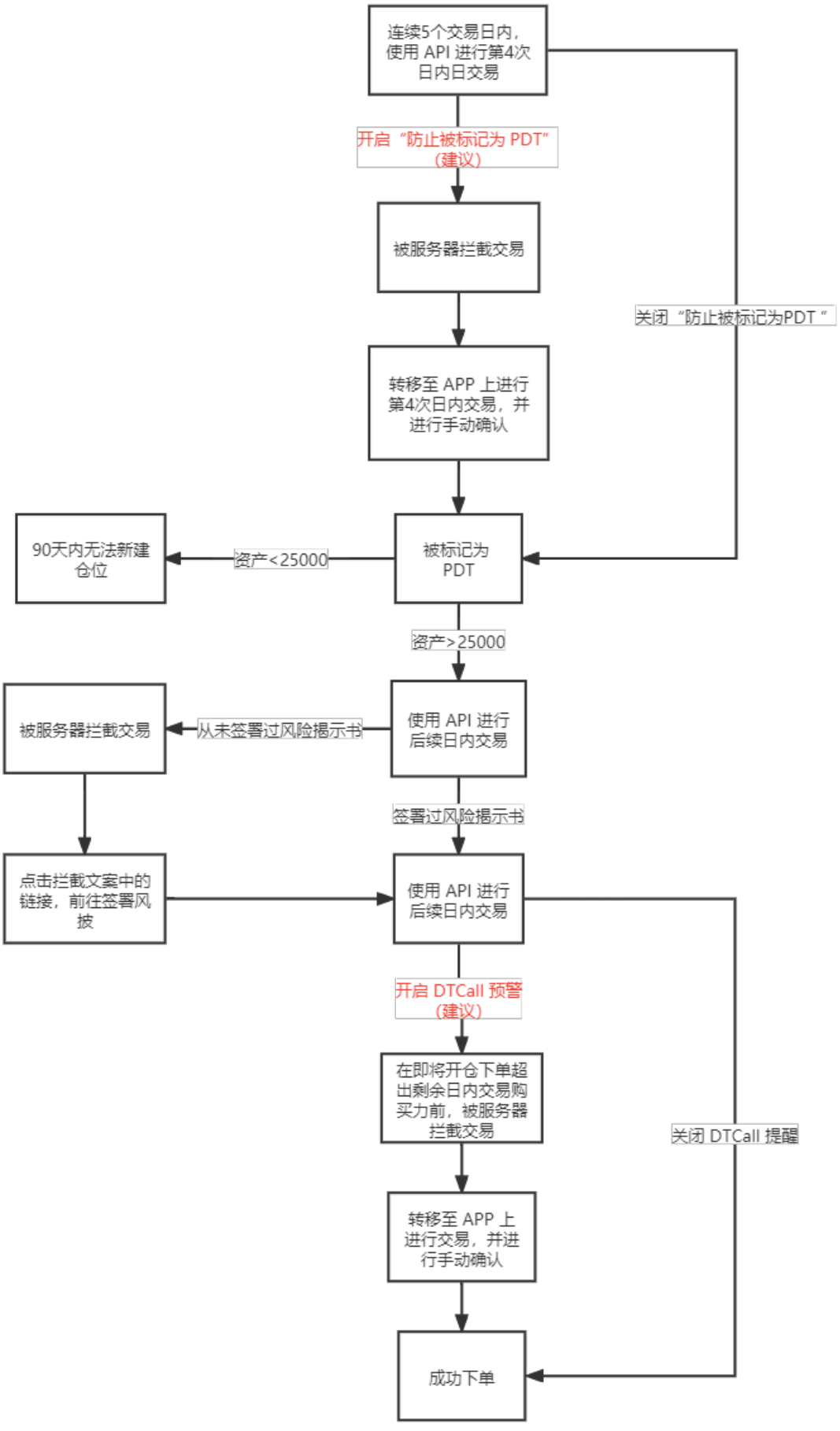
所属券商	协议确认
FUTU HK	点击这里
Moomoo US	点击这里
Moomoo SG	点击这里
Moomoo AU	点击这里
Moomoo CA	点击这里
Moomoo MY	点击这里
Moomoo JP	点击这里

Q12：典型日内交易者（PDT）相关

概述

客户使用moomoo证券(美国) 账户进行日内交易时，会受到美国 FINRA 的监管限制（此为美国券商受到的监管要求，与交易股票的所属市场无关。其他国家或地区的券商 的交易账户则不受此限制）。若用户在任意连续的5个交易日内，进行日内交易 3 次以上，则会被标记为典型日内交易者（PDT）。
更多详情，点击这里

进行日内交易的流程图



我愿意被标记为 PDT，且不希望程式交易被打断，如何关闭“防止被标记为 PDT”？

A:
当您在连续的 5 个交易日内，进行第 4 次日内交易时，为了防止您被无意识地标记为 PDT，服务器会对此交易进行拦截。若您主动想被标记为 PDT，并且不希望服务器拦截，可以采取以下措施：
在 命令行 OpenD 中配置参数，将启动参数 `pdt_protection` 的值修改为 0，以关闭"防止被标记为日内交易者"的功能。

```
<!-- FUTU US 专用参数 -->
<!-- Specific parameters for FUTU US -->
<!-- 是否开启 防止被标记为日内交易者 的功能，0：否，1：是-->
    <!-- 开启功能后，我们会在您将要被标记 PDT 时阻止您的下单，但不确保您一定不被标记。若您被标记 PDT，当您的账户权益小于$25000时，您将无法开仓。-->
    <!-- Whether to turn on the Pattern Day Trade Protection, 0: No, 1: Yes  -->
    <!-- When this parameter is set as 1, we will prevent you from placing orders which might mark you as a Pattern Day Trader(PDT). The Protection c
    <pdt_protection>1</pdt_protection>
```

注意：若您被标记 PDT，当您的账户权益小于$25000时，您将无法开仓。

如何关闭 DTCall 预警提醒?

A:
您被标记为 PDT 后，需要留意账户的日内交易购买力（DTBP），日内交易超出 DTBP 时将收到日内交易保证金追缴（DTCall）。服务器会在您即将开仓下单超出剩余日内交易购买力前，阻止您的下单。若您仍然希望进行下单，并且不希望服务器拦截，可以采取以下措施：
在 命令行 OpenD 中配置参数，将启动参数 `dtcall_confirmation` 的值修改为 0，以关闭"日内交易保证金追缴预警"的功能。

```
<!-- 是否开启 日内交易保证金追缴预警 的功能，0：否，1：是 -->
<!-- 开启功能后，我们会在您即将开仓下单超出剩余日内交易购买力前阻止您的下单。提醒您当前开仓订单的市值大于您的剩余日内交易购买力，若您在今日平仓当前标的，
<!-- Whether to turn on the Day-Trading Call Warning, 0: No, 1: Yes  -->
<!-- When this parameter is set as 1, we will prevent you from placing orders which might exceed your remaining day-trading buying power. We will alert y
<dtcall_confirmation>1</dtcall_confirmation>
```

注意：若您开仓订单的市值大于您的剩余日内交易购买力，并且在今日平仓当前标的，您将会收到日内交易保证金追缴通知（Day-Trading Call），只能通过存入资金才能解除。

如何查看 DTBP 的值?

A:
通过 查询账户资金 接口，可以获取日内交易相关的返回值，如：剩余日内交易次数、初始日内交易购买力、剩余日内交易购买力等。

Q13：如何跟踪订单成交状态

A: 下单后，可使用以下接口跟踪订单成交状态：

交易环境	接口
真实交易	响应订单推送回调，响应成交推送回调
模拟交易	响应订单推送回调

注意：对于非 python 语言用户，在使用上述两个接口之前，需要先进行 订阅交易推送

响应订单推送回调 的特点：

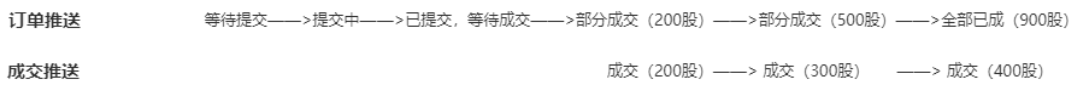
反馈 整个订单 的信息变动。当以下 8 个字段发生变化时，会触发订单推送：
订单状态 ， 订单价格 ， 订单数量 ， 成交数量 ， 触发价格 ， 跟踪类型 ， 跟踪金额/百分比 ， 指定价差

因此，当您进行下单、改单，撤单、使生效、使失效操作，或者订单在市场中发生了高级订单被触发、有成交变动的情况，都会触发订单推送。您只需要调用 响应成交推送回调，即可监听这些信息。

响应成交推送回调 的特点：

只反馈 单笔成交 的信息。当以下 1 个字段发生变化时，会触发订单推送：
成交状态

举例：假设一笔限价单订单 900 股，分成了 3 次才完全成交，每次成交分别是：200、300、400 股。



Q14：下单接口返回“此产品最小单位为 xxx，请调整至最小单位的整数倍后再提交”？

A:
对于不同市场的标的，交易所有着不同的最小变动单位要求。如果提交的订单价格不符合要求，订单将会被拒绝。各市场价位规则如下：

价位规则

香港市场

以港交所官方说明为准，点击 这里。

A 股市场

股票价位：0.01。

美国市场

股票价位：

合约价格	价位
$1 以下	$0.0001
$1 以上	$0.01

期权价位：

合约价格	价位
$0.10 - $3.00	$0.01 或者 $0.05

$3.00 以上	$0.05 或者 $0.10

期货价位：不同合约价位规则不同。可以通过 获取期货合约资料 接口的返回字段 `最小变动的单位` 查看。

怎么避免订单价格不在价位上?

- 方法一：通过 获取实时摆盘 接口，获取合法的交易价格。交易所摆盘上的价位一定是合法的价位。
- 方法二：通过 下单 接口的参数 `价格微调幅度` ，将传入价格自动调整到合法的交易价格上。

  例如：假设腾讯控股当前市价为 359.600，根据价位规则，对应的最小变动价位为 0.200。

  假设您的下单传入订单价格为 359.678，价格微调幅度为 0.0015，代表接受 OpenD 对传入价格自动向上调整到最近的合法价位，且不能超过 0.15%。此情景下，向上最近的合法价格为 359.800，价格实际需要调整的幅度为 0.034%，符合价格微调幅度的要求，因此最终提交的订单价格为 359.800。

  若价格微调幅度设置数值小于实际需要调整的幅度，OpenD 自动调整价位失败，订单仍会返回报错"订单价格不在价位上"。

Q15：我的购买力足够，为什么下市价单会返回"购买力不足"?

A:

为什么市价单会提示购买力不足

- 出于风控考量，系统给了市价单较高的购买力系数。在所有订单参数都相同的情况下，选择市价单会比限价单占用更多的购买力。
- 而且对于不同的品种，和不同的市场情况，风控系统会对市价单的购买力系数做动态调整。所以在下市价单时，若您通过最大购买力去计算最大可买数量，计算的结果很可能是不准确的。

如何计算正确的可买数量

不建议自己计算，您可以通过 查询最大可买可卖 接口获取正确的可买数量。

如何尽可能买更多

您可以用价格为对价的限价单，替代市价单进行交易。
其中，对价：买1价（下卖单时）或 卖1价（下买单时）

Q16：API模拟交易下单，为什么移动端看不到?

A:
移动端、桌面端、网页端，美股模拟交易账户，已经从【美股模拟账户】升级成为功能更丰富的【美股融资融券账户】。
OpenAPI 暂未升级（规划中），目前只能使用旧的【美股模拟账户】，且旧的【美股模拟账户】无法在其他客户端上展示，请谨慎使用。

Q17：交易接口参数使用说明

1. 什么是交易对象?

您的平台账号下一般会开设一个保证金综合账户，其中有多个交易子账户（正常有两个，一个综合证券账户，一个综合期货账户；根据需要还可能有综合外汇账户等其他子账户）。一些特殊用户或机构客户可能会在多个券商下开设多个综合账户。
创建交易对象，是初步筛选子账户的过程。

- 使用 OpenSecTradeContext 创建的交易对象，调用 get_acc_list 时只会返回**证券交易账户**
- 使用 OpenFutureTradeContext 创建的交易对象，调用 get_acc_list 时只会返回**期货交易账户**

参数 security_firm 用来筛选对应归属券商的账户，参数 filter_trdmarket 用来筛选对应交易市场权限的账户。

1.1 security_firm 券商参数

OpenAPI 目前支持的券商有 这些。
创建的交易对象，在调用 get_acc_list 时，会返回 security_firm 对应券商的真实账户和所有模拟交易账户（这是因为模拟交易没有券商的概念，所以无论 security_firm 传什么，都会返回所有的模拟账户）。
security_firm 的默认值是 FUTUSECURITIES，FUTU HK 券商账户可以不填此参数，但需要获取其他券商的账户时，需要修改券商参数。

- Example 1

```py
trd_ctx = OpenSecTradeContext(security_firm=SecurityFirm.FUTUSECURITIES)
ret, data = trd_ctx.get_acc_list()
print(data)
```

- Output

```py
              acc_id   trd_env acc_type      uni_card_num          card_num   security_firm sim_acc_typ
0  281756478396547854      REAL   MARGIN  1001200163530138  1001369091153722  FUTUSECURITIES          N/
1             3450309  SIMULATE     CASH               N/A               N/A             N/A        STOC
2             3548731  SIMULATE   MARGIN               N/A               N/A             N/A       OPTIC
3  281756455998014447      REAL   MARGIN               N/A  1001100320482767  FUTUSECURITIES          N/
```

- Example 2

```py
trd_ctx = OpenSecTradeContext(security_firm=SecurityFirm.FUTUSG)
ret, data = trd_ctx.get_acc_list()
print(data)
```

- Output

```
    acc_id   trd_env acc_type uni_card_num card_num security_firm sim_acc_type trdmarket_auth acc_status
0  3450309  SIMULATE     CASH          N/A      N/A           N/A        STOCK           [HK]     ACTIVE
1  3548731  SIMULATE   MARGIN          N/A      N/A           N/A       OPTION           [HK]     ACTIVE
```

1.2 filter_trdmarket 交易市场参数

OpenAPI 目前支持的交易市场有 这些。
创建的交易对象，在调用 get_acc_list 时，会返回所有拥有 filter_trdmarket 市场交易权限的账户；当 filter_trdmarket 入参传 NONE 时，不过滤市场，返回所有的账户。
filter_trdmarket 的默认参数是 HK，在综合账户体系下，这个参数用来筛选不同市场下的模拟交易账户。

- Example 1

```py
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US)
ret, data = trd_ctx.get_acc_list()
print(data)
```

- Output

```
                acc_id   trd_env acc_type      uni_card_num          card_num   security_firm sim_acc_typ
0  281756478396547854      REAL   MARGIN  1001200163530138  1001369091153722  FUTUSECURITIES          N/
1             3450310  SIMULATE   MARGIN              N/A               N/A             N/A        STOC
2             3548732  SIMULATE   MARGIN              N/A               N/A             N/A       OPTIC
3  281756460292981743      REAL   MARGIN              N/A  1001100520714263  FUTUSECURITIES          N/
```

- Example 2

```py
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.NONE)
ret, data = trd_ctx.get_acc_list()
print(data)
```

- Output

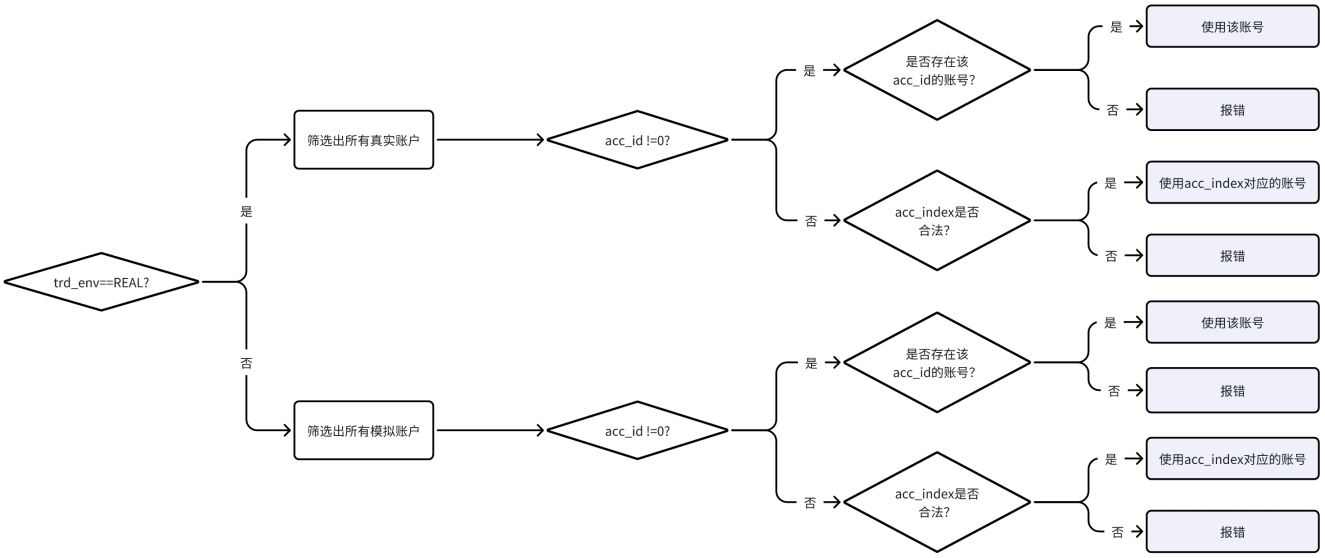
```
                acc_id   trd_env acc_type      uni_card_num          card_num   security_firm sim_acc_ty
0  281756478396547854      REAL   MARGIN  1001200163530138  1001369091153722  FUTUSECURITIES          N
1             3450309  SIMULATE     CASH              N/A               N/A             N/A        STO
2             3450310  SIMULATE   MARGIN              N/A               N/A             N/A        STO
3             3450311  SIMULATE     CASH              N/A               N/A             N/A        STO
4             3548732  SIMULATE   MARGIN              N/A               N/A             N/A       OPTI
5             3548731  SIMULATE   MARGIN              N/A               N/A             N/A       OPTI
6  281756455998014447      REAL   MARGIN              N/A  1001100320482767  FUTUSECURITIES          N
7  281756460292981743      REAL   MARGIN              N/A  1001100520714263  FUTUSECURITIES          N
```

```
8   281756468882916335    REAL   MARGIN          N/A  1001100610464507  FUTUSECURITIES
9   281756507537621999    REAL     CASH          N/A  1001100910390035  FUTUSECURITIES
10  281756550487294959    REAL     CASH          N/A  1001101010406844  FUTUSECURITIES
```

> **提示**
>
> 当 filter_trdmarket 入参NONE时，可以返回所有的交易账户。其中第0行是真实账户，1~5行均为模拟交易账户，6~10行是已失效的真实账户。这些失效账户都是单市场账户，现已被综合账户替代。但历史订单和历史成交还在这些已失效的账户中，可以通过这些账户来查询。
>
> OpenFutureTradeContext 对象中没有 filter_trdmarket 参数，只有 security_firm 参数，功能与 OpenSecTradeContext 一样。

2. 交易接口参数

在使用具体的交易接口（如下单、查询订单列表）时，接口中的 `trd_env`，`acc_index` 和 `acc_id` 参数，会先筛选确认一个唯一的账户，对此账户实施对应的接口行为。



> **总结**
>
> 1. 根据 trd_env 筛选出真实账户还是模拟账户
> 2. 在筛选结果中，优先选择 acc_id 指定的账户
> 3. 如果 acc_id 为0，则通过 acc_index选取对应账号
> 4. 报错场景：指定的 acc_id 不存在，或 acc_index 超出范围

3. 应用举例

3.1 综合证券账户实盘下单

```py
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.NONE, security_firm=SecurityFirm.FUTUSECURITIES
ret, data = trd_ctx.unlock_trade("123123")
if ret == RET_OK:
```

```
    print("解锁成功")
    ret, data = trd_ctx.place_order(45, 200, 'HK.00700', TrdSide.BUY,
                                    order_type=OrderType.NORMAL,
                                    trd_env=TrdEnv.REAL,  # 和默认参数一样，可以不填
                                    acc_id=0)  # 和默认参数一样，可以不填
    print(data)
```

3.2 综合期货账户查询实盘订单列表

```py
trd_ctx = OpenFutureTradeContext(security_firm=SecurityFirm.FUTUSECURITIES)

ret, data = trd_ctx.order_list_query(trd_env=TrdEnv.REAL,   # 和默认参数一样，可以不填
                                     acc_id=0)  # 和默认参数一样，可以不填
print(data)
```

3.3 港股模拟现金账户查询账户资金

```py
# filter_trdmarket 填 TrdMarket.HK
# trd_env 填 TrdEnv.SIMULATE
# acc_index 填 0
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.HK)
ret, data = trd_ctx.accinfo_query(trd_env=TrdEnv.SIMULATE, acc_index=0)
print(data)
```

3.4 美股模拟保证金账户下单期权

```py
# 通过 filter_trdmarket 和 trd_env 筛选完之后只剩两个账户
# 第0个是美股现金账户（交易股票）,第1个是美股保证金账户（交易期权）
# acc_index 填 1 指定美股保证金账户
trd_ctx = OpenSecTradeContext(filter_trdmarket=TrdMarket.US)
ret, data = trd_ctx.place_order(10, 1, code="US.AAPL250618P550000",trd_side=TrdSide.BUY,
                                trd_env=TrdEnv.SIMULATE,
                                acc_index=1)
print(data)
```

3.5 日本期货模拟账户查询最大可买卖

```py
# 将 get_acc_list 的结果打印出来，可以看到日本期货模拟账户的 acc_id 是 6271199
# 请求最大可买卖接口时传入这个 acc_id
trd_ctx = OpenFutureTradeContext()
ret, data = trd_ctx.acctradinginfo_query(order_type=OrderType.NORMAL,
                                         price=5000,
                                         trd_env=TrdEnv.SIMULATE,
                                         acc_id=6271199,
```

```
                                        code="JP.NK225main")
print(data)
```

4. OpenAPI 中的账户如何与 APP/桌面端对应

APP 上的账户仅显示卡号后 4 位数字，我们将 get_acc_list 的返回结果打印出来后，有 uni_card_num 列和 card_num 列，分别对应综合账户的卡号，和单币种账户（已废弃）的卡号。通过卡号后 4 位数就能把 API 中获取到的账号与 APP 上对应起来了。

其他

Q1：如何编译C++ API?

A: moomoo api c++ SDK支持Windows/MacOS/Linux，每个系统提供了以下编译环境生成的库文件：

操作系统	编译工具
Windows	Visual Studio 2013
Centos 7	g++ 4.8.5
Ubuntu 16.04	g++ 5.4.0
MacOS	XCode 11

如果编译器版本不同，或依赖的protobuf版本不同，则可能需要自己使用源码重新编译MMAPI和protobuf，源码位置见下图目录：

```
MMAPI目录结构：
+---Bin                                存放各个系统默认编译环境编译出的依赖库
+---Include                            存放公共头文件，以及proto协议生成的.h/.cc文件
+---Sample                             示例工程
\---Src
    +---MMAPI                          MMAPI源码
    +---protobuf-all-3.5.1.tar.gz      protobuf源码
```

编译步骤：

1. 重新编译protobuf：生成libprotobuf静态库
2. 从协议proto文件中生成C++文件
3. 重新编译MMAPI: 源码在Src/MMAPI，生成libMMAPI静态库

步骤1： 重新编译protobuf：

- Windows：

    - 安装CMake
    - 打开VS命令行工具，cd到protobuf/cmake目录
    - 执行：cmake -G "Visual Studio 12 2019" -DCMAKE_INSTALL_PREFIX=install -Dprotobuf_BUILD_TESTS=OFF 这样会生成Visual Studio 2019的项目文件，其它版本Visual Studio请修改-G参数
    - 打开生成的Visual Studio项目文件，平台工具集设置为v120_xp，编译即可
- Linux（参考protobuf/src/README）
    - 执行 ./autogen.sh
    - 执行 CXXFLAGS="-std=gnu++11" ./configure --disable-shared
    - 执行 make
    - 将生成的libprotobuf.a放入Bin/Linux目录
- MacOS（参考protobuf/src/README）
    - 使用brew安装这些依赖库：autoconf automake libtool
    - 执行./configure CC=clang CXX="clang++ -std=gnu++11 -stdlib=libc++" --disable-shared

步骤2: 重新生成proto代码

- 上面编译Protobuf后会同时生成可执行文件protoc。用protoc将Include/Proto下面的.proto文件生成对应的.h和.cc文件。例如命令以下命令会从Common.proto生成对应的Common.pb.h和Common.pb.cc
    - protoc -I="MMAPI路径/Include/Proto" --cpp_out="." MMAPI路径/Include/Proto/Common.proto
- 将生成的.h和.cc文件放到Include/Proto下面

步骤3: 重新编译MMAPI

- Windows：新建Visual Studio C++静态库工程，将Src/MMAPI和Include下的源码加入工程中，平台工具集设置为v120_xp，然后编译
- Mac：新建XCode C++静态库工程，将Src/MMAPI和Include下的源码加入工程中，然后编译
- Linux：使用CMake编译MMAPI静态库，在MMAPI路径/Src目录下执行：
    - cmake -DTARGET_OS=Linux

Q2：有没有更完整的策略样例可以参考?

A:

- Python 策略样例在 /moomoo/examples/ 文件夹下。您可以通过执行如下命令，找到 Python API 的安装路径：

```
import moomoo
print(moomoo.__file__)
```

- C# 策略样例在 /MMAPI4NET/Sample/ 文件夹下
- Java 策略样例在 /MMAPI4J/sample/ 文件夹下
- C++ 策略样例在 /MMAPI4CPP/Sample/ 文件夹下
- JavaScript 策略样例在 /MMAPI4JS/sample/ 文件夹下

Q3：使用 python API 导入异常

场景一：已经在 Python 环境中安装了 moomoo 模块，仍然提示 No module named 'moomoo'?
很可能是因为当前 IDE 所使用的 interpreter 并不是你装过 moomoo 模块的 interpreter。也就是说，您的电脑可能装了两个以上的 Python 环境。 您可以操作如下两步：

1. 在 Python 中运行如下代码，得到当前 interpreter 的路径：

```
import sys
print(sys.executable)
```

示例图：

```
In[3]: import sys
  ...: print(sys.executable)
D:\software\anaconda3\python.exe
```

2. 在命令行中，执行 `$ D:\software\anaconda3\python.exe -m pip install moomoo-api` （其中前半部分的文件路径来自第 1 步打印的路径）。 这样就可以在当前的 interpreter 中也安装一份 moomoo 模块。

Q4： import 成功了，仍然调用不了相关接口?

A：通常遇到这种情况，需要确认一下：成功导入的 moomoo，是不是真正的 moomoo API 模块。以下几种场景也可能 import 成功。

场景一：存在与“moomoo”重名的文件

1. 当前文件名是 moomoo.py
2. 当前文件所在目录下存在另一个名为 moomoo.py 的文件
3. 当前文件所在目录下存在名为 `/moomoo` 的文件夹

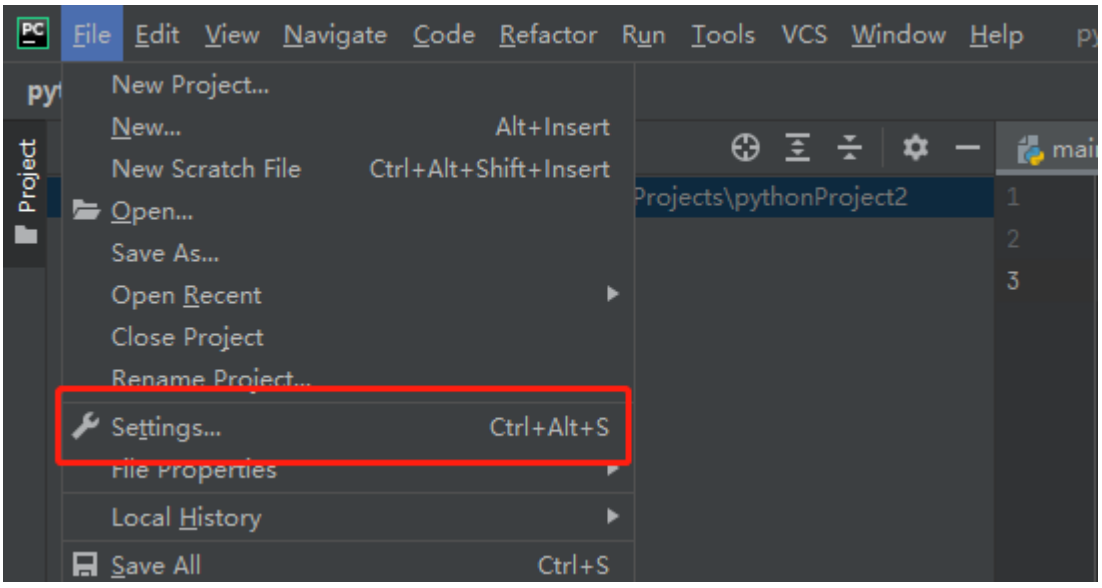
因此，我们强烈建议您，在给文件 / 文件夹 / 工程起名的时候，不要起名叫“moomoo”。重名一时爽，查 bug 两行泪。

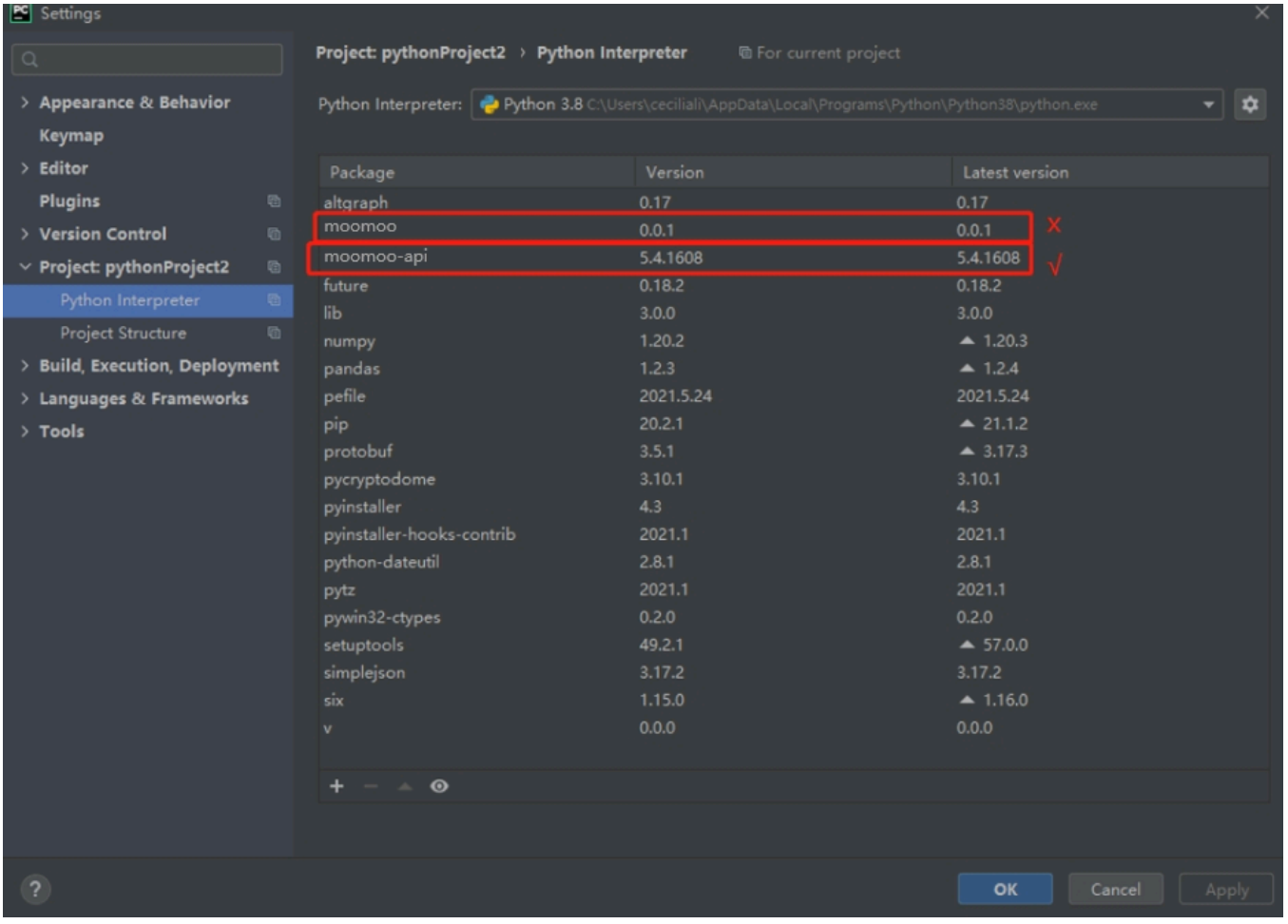
场景二：误装了一个名为“moomoo”的第三方库

moomoo API 的正确名称为 `moomoo-api`，而非“moomoo”。

如果您安装过名为“moomoo”的第三方库，请将其卸载，并 下载 **moomoo-api**。

以 PyCharm 为例：查看第三方库的安装情况。





Q5：协议加密相关

A:

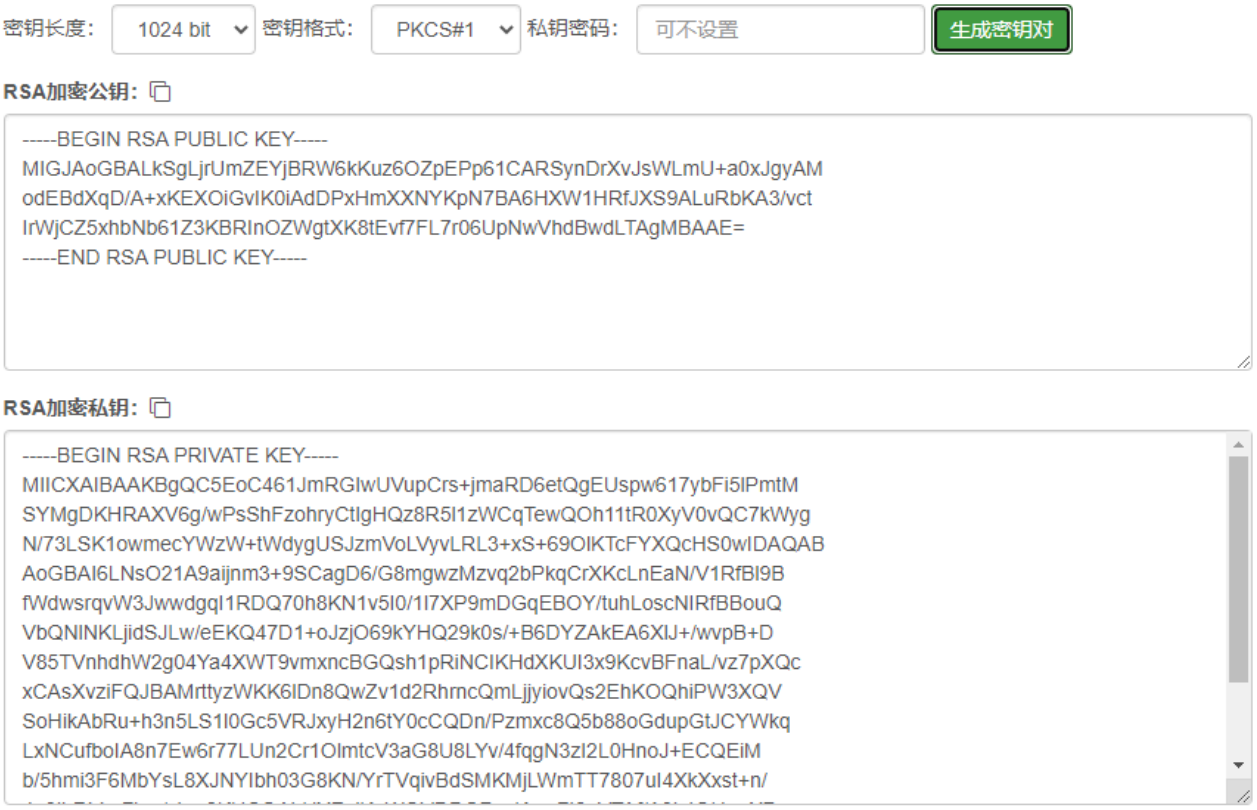
概述

您可以使用非对称加密算法 RSA，对策略程序（moomoo API）与 OpenD 之间的请求和返回内容进行加密，以保证通信安全。
如果您的策略程序（moomoo API）与 OpenD 在同一台电脑上，则通常无需加密。

协议加密流程

您可以尝试通过以下步骤解决此问题：

1. 通过第三方 web 平台自动生成密钥文件。
   - 具体方法：在 baidu 或 google 上搜索“RSA 在线生成”，**密钥格式**设置为 PKCS#1，**密钥长度**设置为 1024 bit，不需要设置私钥密码，点击**生成密钥对**。



2. 将生成的 **RSA 加密私钥** 复制粘贴至 txt 记事本，并保存至 OpenD 所在电脑的指定路径。

3. 在 OpenD 所在的电脑中，指定 **RSA 加密私钥** 的路径。
   - 方式一：在 可视化 OpenD 启动界面右侧的“加密私钥”一栏，指定上一步骤中放置 **RSA 加密私钥** 的路径。如下图所示：

- 方式二：在 命令行 OpenD 启动文件 OpenD.xml 中，找到参数 `rsa_private_key` ，将其配置为第 2 步中 RSA **加密私钥** 的路径。如下图所示：

```
<moomoo_opend>
    <!-- 基础参数 -->
    <!-- Basic parameters -->
        <!-- 协议监听地址,不填默认127.0.0.1 -->
        <!-- Listening address. 127.0.0.1 by default -->
        <ip>127.0.0.1</ip>
        <!-- API接口协议监听端口 -->
        <!-- API interface protocol listening port -->
        <api_port>11111</api_port>
        <!-- 登录帐号 -->
        <!-- Login account -->
        <login_account>100000</login_account>
        <!-- 登录密码32位MD5加密16进制 -->
        <!-- Login password, 32-bit MD5 encrypted hexadecimal -->
        <!-- <login_pwd_md5>6e55f158a827b1a1c4321a245aaaad88</login_pwd_md5> -->
        <!-- 登录密码明文，密码密文存在情况下只使用密文 -->
        <!-- Plain text of login password. When cypher text exists, the cypher text will be used. -->
        <login_pwd>123456</login_pwd>
        <!-- FutuOpenD语言，en：英文，chs：简体中文 -->
        <!-- FutuOpenD language. en: English, chs: Simplified Chinese -->
        <lang>chs</lang>
    <!-- 进阶参数 -->
    <!-- Advanced parameters -->
        <!-- FutuOpenD日志等级，no，debug，info，warning，error，fatal -->
        <!-- FutuOpenD log level: no, debug, info, warning, error, fatal -->
        <log_level>info</log_level>
        <!-- API推送协议格式，0：pb，1：json -->
        <!-- API push protocol format. 0: pb, 1: json -->
        <push_proto_type>0</push_proto_type>
        <!-- API订阅数据推送频率控制，单位毫秒，目前不包括K线和分时，不设置则不限制频率-->
        <!-- Data Push Frequency, in milliseconds. Candlesticks and timeframes are not included. If not se
        <!-- <qot_push_frequency>1000</qot_push_frequency> -->
        <!-- Telnet监听地址,不填默认127.0.0.1 -->
        <!-- Telnet listening address. 127.0.0.1 by default -->
        <!-- <telnet_ip>127.0.0.1</telnet_ip> -->
        <!-- Telnet监听端口 -->
        <!-- Telnet listening port -->
        <!-- <telnet_port>22222</telnet_port> -->
        <!-- API协议加密私钥文件路径,不设置则不加密 -->
        <!-- File path for private key for API protocol enctyption. If not set, it will not be encrypted.
        <!-- <rsa_private_key>D:\rsa</rsa_private_key> -->
        <!-- 是否接收到价提醒推送，0：不接收，1：接收 -->
```

4. 将第 2 步中 txt 文件另存至策略程序（moomoo API）所在电脑的指定路径， 并在策略程序中将此路径 设置为私钥路径。

5. 在策略程序（moomoo API）中启用协议加密。 启用协议加密的方式有两种，其中方式二的优先级更高。

   - 方式一：对单条的连接加密（通用）。在对 行情对象 或 交易对象 创建连接时，通过 **是否启用加密** 参数设置加密。
   - 方式二：对所有的连接加密（仅 Python）。通过 `enable_proto_encrypt` 接口设置加密，详见 这里。

> **提示**
>
> - 在 OpenD 或策略程序（moomoo API）中指定 RSA **加密私钥** 路径时，需指定至 txt 文件本身。
> - RSA 加密公钥无需保存，可通过私钥计算得到。

Q6：为什么我获取的 DataFrame 数据，只能展示一部分？

A：打印 pandas.DataFrame 数据的时候，如果行列数过多，pandas 默认会将数据折叠，导致看起来显示不全。
因此，并不是接口返回数据真的不全。您只需要在 Python 脚本前面加上如下代码即可解决。

```
import pandas as pd
pd.options.display.max_rows=5000
pd.options.display.max_columns=5000
pd.options.display.width=1000
```

Q7：Mac 机器使用 C++ 语言的 API，遇到 “无法打开 libFTAPIChannel.dylib” 的问题

A：在对应库目录中执行以下命令即可解决: `$ xattr -r -d com.apple.quarantine libAPIChannel.dylib` 。

Q8：Python 用户，为什么在 OpenD 配置文件中设置了日志级别为 no 后，log 文件夹下仍然持续产生超大容量的日志文件?

A：OpenD 配置文件中的日志级别参数，只用来控制 OpenD 产生的日志。而 Python API 默认也会产生日志，如果您不希望希望 Python API 产生日志，可以在 Python 脚本加上如下语句：

```
logger.file_level = logging.FATAL  # 用于关闭 Python API 日志
logger.console_level = logging.FATAL  # 用于关闭 Python 运行时的控制台日志
```

Q9：对于 5.4 及以上的版本，Java API 的库名和配置方式的变更

A: * 如果您是 Java API 5.3 及以下版本的用户，在更新版本时，请注意以下变更：

配置流程的变更：

1. 通过 moomoo 官网 下载 moomoo API。
2. 解压下载好的 mmAPI 文件，`/MMAPI4J` 是 Java API 的目录，将目录结构中的 `/lib/moomoo-api-.x.y.z.jar` 添加到您的工程设置中。创建 moomoo-api 工程请参考这里。

目录结构的变更：

1. moomoo API 的 Java 版本，库名由之前的 mmapi4j.jar 变更为 `moomoo-api-x.y.z.jar`，其中 “x.y.z” 表示版本号。
2. 第三方库的引用中，去掉了 /lib/jna.jar 和 /lib/jna-platform.jar 依赖，增加了 `/lib/bcprov-jdk15on-1.68.jar` 和 `/lib/bcpkix-jdk15on-1.68.jar` 依赖。

```
```
+---mmapi4j   moomoo-api 源码，如果所用 JDK 版本不兼容可以用这里的工程
+---lib   存放公共库文件
| moomoo-api-x.y.z.jar   moomoo API 的 Java 版本
| bcprov-jdk15on-1.68.jar   第三方库，用于加解密
```
```

```
|    bcpkix-jdk15on-1.68.jar      第三方库，用于加解密
|    protobuf-java-3.5.1.jar      第三方库，用于解析 protobuf 数据
+---sample                        示例工程
+---resources                     maven 工程默认生成的目录
```

- 如果您第一次接触 moomoo API，我们提供了更便捷的通过 maven 仓库配置 Java API 的方式。配置流程请参考 这里。

Q10：Python 用户，使用 pyinstaller 打包脚本时报错：找不到 Common_pb2 模块

A：你可以尝试通过以下步骤解决此问题：

1. 假设你需要对 main.py 进行打包。使用命令行语句，运行代码：pyinstaller main.py，不要加参数 "- F"（path 为 main.py 的所在路径）

```
pyinstaller path\main.py
```

打包成功后，main.py 所在目录下的 /dist 中，会生成 /main 文件夹，main.exe 就在这个文件夹中。

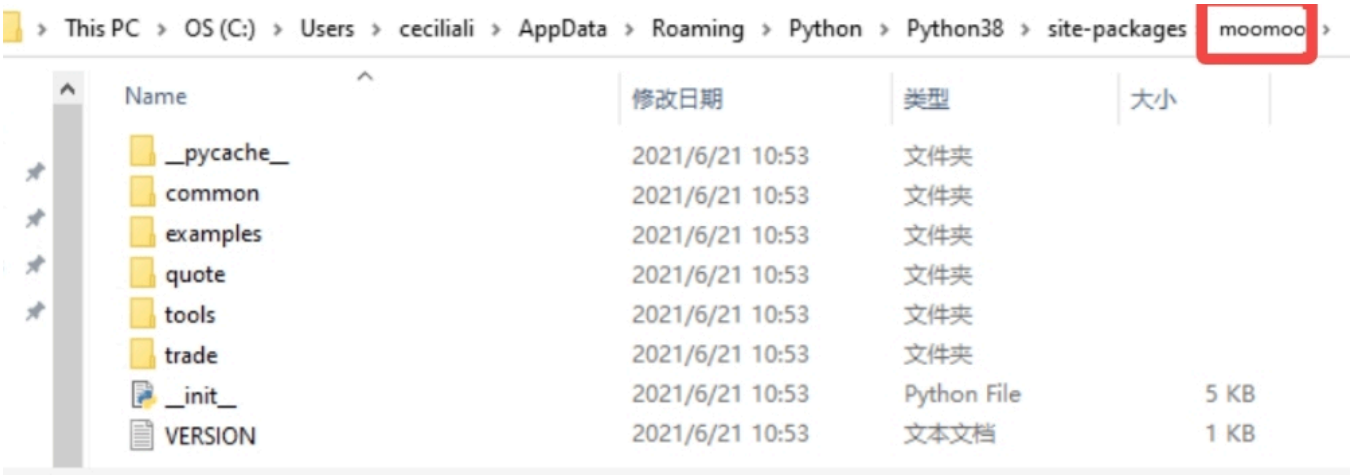
2. 运行以下代码，找到 moomoo-api 的安装目录。

```
import moomoo
print(moomoo.__file__)
```

```
```

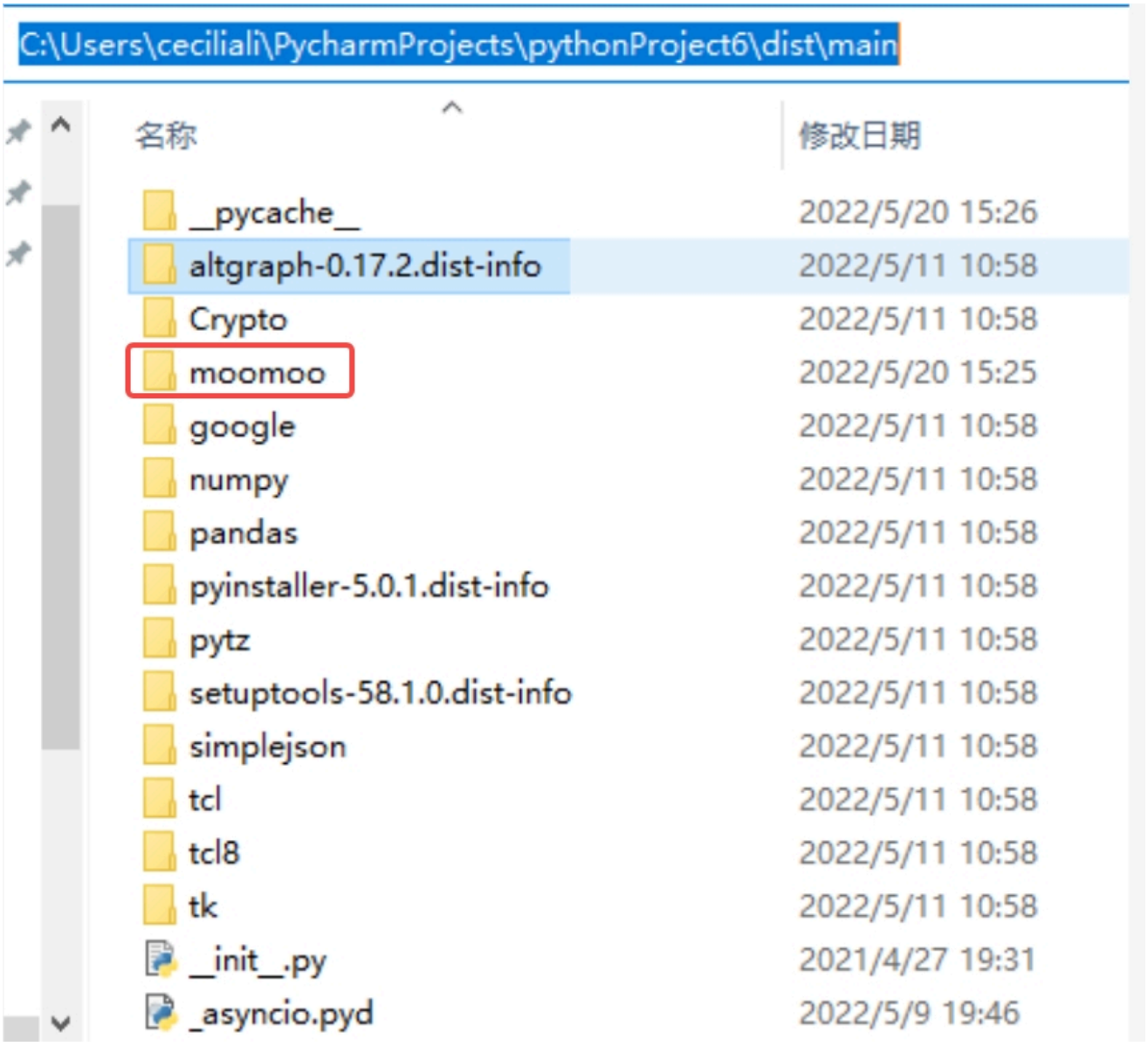
运行结果:

```
C:\Users\ceciliali\Anaconda3\lib\site-packages\moomoo\__init__.py
```



3. 打开上图文件夹中的 /common/pb，将所有文件全部复制到 /main 中。

4. 在 /main 中创建文件夹，命名为 moomoo，将上图文件夹中的 `VERSION.txt` 文件复制到 /main/moomoo 中。



5. 再次尝试运行 main.exe

Q11：接口调用结果正常，但其返回表现不符合预期？

A:

- 接口调用结果正常，表示富途已经成功收到并响应了您的请求，但接口返回表现可能与您的预期不符。

  例如：若您在非交易时段调用 订阅 接口，虽然您的请求可以被成功响应，并且接口调用结果正常，但在非交易时段下，交易所无行情数据变动，所以您将暂时无法收到行情数据推送，直至市场重新回到交易时段。

- 接口调用结果可以通过返回字段（定义参见：接口调用结果）查看，返回字段为 0 代表接口调用正常，非 0 代表接口调用失败。

对于 Python 用户，下面两种写法等价：

```
if ret_code == RET_OK:
```

```
if ret_code == 0:
```

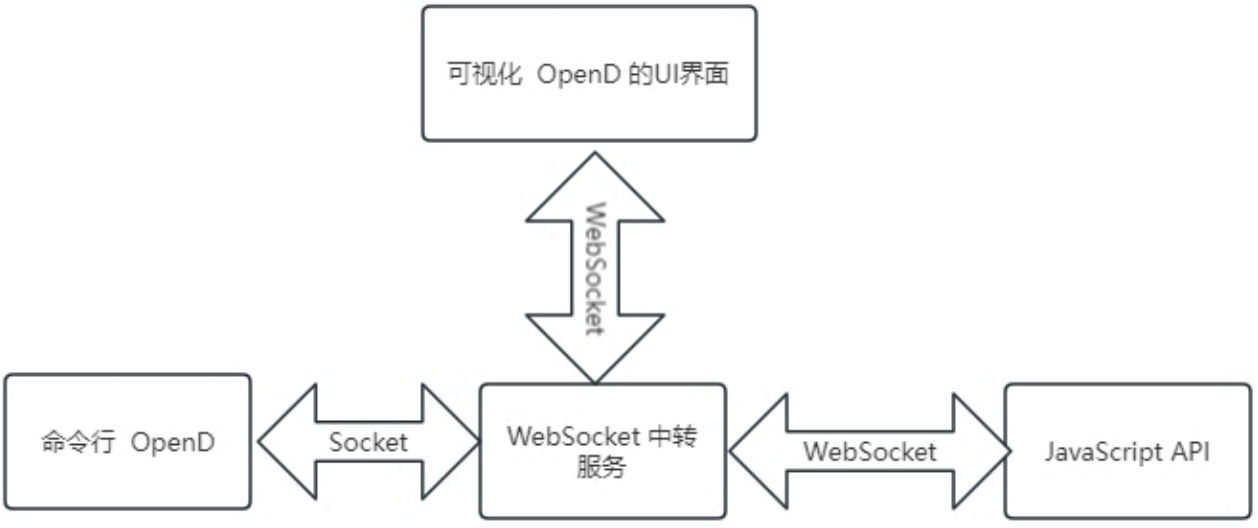
Q12：WebSocket相关

A：

概述

OpenAPI 中，WebSocket 主要用于以下两方面：

- 可视化 OpenD 中，UI 界面跟底层的命令行 OpenD 的通信使用 WebSocket 方式。
- JavaScript API 跟 OpenD 之间的通信使用 WebSocket 方式。



- 当 WebSocket 启动时，命令行 OpenD 会与 **MMWebSocket 中转服务** 建立 Socket 连接（TCP），这一连接会用到默认的 **监听地址** 和 **API 协议监听端口。**
- 同时，JavaScript API 会与 **MMWebSocket 中转服务** 建立 WebSocket 连接（HTTP），这一连接会用到 **WebSocket 监听地址** 和 **WebSocket 端口。**

使用

为保证账户安全，当 WebSocket 监听来自非本地请求时，我们强烈建议您启用 SSL 并配置 **WebSocket 鉴权密钥**。

SSL 通过在配置 **WebSocket 证书** 以及 **WebSocket 私钥** 来启用。
命令行 OpenD 可通过配置 OpenD.xml 或配置命令行参数来设置文件路径。可视化 OpenD 点击【更多选项】下拉菜单，可以看到设置项。

> **提示**
>
> 如果证书是自签的，则需要在调用 JavaScript 接口所在机器上安装该证书，或者设置不验证证书。

生成自签证书

自签证书生成详细资料不便在此文档展开，请自行查阅。
在此提供较简单可用的生成步骤：

1. 安装 openssl。

2. 修改 openssl.cnf，在 alt_names 节点下加上 OpenD 所在机器 IP 地址或域名。
   例如：IP.2 = xxx.xxx.xxx.xxx, DNS.2 = www.xxx.com
3. 生成私钥以及证书（PEM）。

证书生成参数参考如下：

```
openssl req -x509 -newkey rsa:2048 -out moomoo.cer -outform PEM -keyout moomoo.key -days 10000 -verbose -config openssl.cnf -nodes -sha256 -subj "/CN=moomoo CA" -reqexts v3_req -extensions v3_req
```

> **提示**
>
> - openssl.cnf 需要放到系统路径下，或在生成参数中指定绝对路径。
> - 注意生成私钥需要指定不设置密码（-nodes）。

附上本地自签证书以及生成证书的配置文件供测试：

- openssl.cnf
- moomoo.cer
- moomoo.key

Q13：OpenAPI 的行情和交易服务分别部署在哪里?

A：

- 行情：

平台账号	行情服务器所在地
牛牛号	腾讯云广州和香港
moomoo 号	腾讯云美国弗吉尼亚和新加坡

- 交易：

所属券商	交易服务器所在地
富途证券(香港)	香港
moomoo证券(美国)	腾讯云美国弗吉尼亚
moomoo证券(新加坡)	腾讯云新加坡

所属券商	交易服务器所在地
moomoo证券(澳大利亚)	腾讯云新加坡
moomoo证券(马来西亚)	阿里云马来西亚
moomoo证券(加拿大)	AWS加拿大
moomoo证券(日本)	腾讯云日本