



SHIOAJI

Shioaji

Usage Manual

yvictor, SsallyLin, yepochien

Copyright © 2025 SinoPac

Table of contents

1. Shioaji	4
1.1 Installation	4
2. Quick Start	5
2.1 Streaming Market Data	5
2.2 Place Order	5
2.3 Conclusion	6
3. Environment Setup	7
3.1 Install Python Environment	7
3.2 Create Project Environment	7
4. Tutorial - User Guide	10
4.1 Prepare	10
4.2 Login	25
4.3 Contract	29
4.4 Market Data	33
4.5 Order	66
4.6 CallBack	105
4.7 Account Data	113
4.8 Simulation Mode	128
4.9 Advanced Guide	129
5. Upgrading to 1.0	144
5.1 Shioaji	144
5.2 Login	146
5.3 Stock Order	146
5.4 Futures Order	150
5.5 Market Data	153
5.6 Future Account Info.	155
6. QA	156
7. Release Note	159
7.1 version: 1.3.0 (2025-12-17)	159
7.2 version: 1.2.9 (2025-10-29)	159
7.3 version: 1.2.8 (2025-09-10)	159
7.4 version: 1.2.7 (2025-08-13)	159
7.5 version: 1.2.6 (2025-06-16)	159
7.6 version: 1.2.5 (2024-10-01)	160
7.7 version: 1.2.4 (2024-08-28)	160

7.8 version: 1.2.3 (2024-03-06)	160
7.9 version: 1.2.2 (2024-01-09)	160
7.10 version: 1.2.1 (2023-12-22)	160
7.11 version: 1.2.0 (2023-12-20)	160
7.12 version: 1.1.13 (2023-11-01)	161
7.13 version: 1.1.12 (2023-08-22)	161
7.14 version: 1.1.11 (2023-08-04)	161
7.15 version: 1.1.10 (2023-07-23)	161
8. Use Restrictions	162

1. Shioaji

shioaji-logosinopac-logo

[PyPI - Status PyPI - Python Version PyPI - Downloads Build - Status Coverage Binder doc Telegram](#)

Shioaji is the most pythonic API for trading the Taiwan and global financial market. You can integrated your favorite Python packages such as NumPy, pandas, PyTorch or TensorFlow to build your trading model with the Shioaji API on cross-platform.

We are in early-release alpha. Expect some adventures and rough edges.

The key features are:

- Fast: High performance with c++ implement core and FPGA event broker.
- Easy: Designed to be easy to use and learn.
- Fast to code: With native python to integrate with large python ecosystem.
- Cross-Platform: The first one python trading API with Linux compatible in Taiwan.

1.1 Installation

1.1.1 Binaries

simple using pip to install

```
pip install shioaji
```

update shioaji with

```
pip install -U shioaji
```

1.1.2 uv

using uv to install

```
uv add shioaji
```

install speed version

```
uv add shioaji --extra speed
```

1.1.3 Docker Image

simple run with interactive mode in docker

```
docker run -it sinotrade/shioaji:latest
```

run with jupyter lab or notebook

```
docker run -p 8888:8888 sinotrade/shioaji:jupyter
```

2. Quick Start

Just import our API library like other popular python library and new the instance to start using our API. Login your account and activate the certification then you can start placing order.



** Please complete the Prepare before starting, including [Open Account](#), [Terms of Service](#) and [Token](#). **

Login and Activate CA

version>=1.0 version<1.0

```
import shioaji as sj

api = sj.Shioaji()
accounts = api.login("YOUR_API_KEY", "YOUR_SECRET_KEY")
api.activate_ca(
    ca_path="/c/your/ca/path/Sinopac.pfx",
    ca_passwd="YOUR_CA_PASSWORD",
    person_id="Person of this Ca",
)

import shioaji as sj

api = sj.Shioaji()
accounts = api.login("YOUR_PERSON_ID", "YOUR_PASSWORD")
api.activate_ca(
    ca_path="/c/your/ca/path/Sinopac.pfx",
    ca_passwd="YOUR_CA_PASSWORD",
    person_id="Person of this Ca",
)
```



The Certification Path

In Windows you copy the file path with \ to separate the file, you need to replace it with / .

2.1 Streaming Market Data

Subscribe the real time market data. Simply pass contract into quote `subscribe` function and give the quote type will receive the streaming data.

```
api.quote.subscribe(api.Contracts.Stocks["2330"], quote_type="tick")
api.quote.subscribe(api.Contracts.Stocks["2330"], quote_type="bidask")
api.quote.subscribe(api.Contracts.Futures["TXFCO"], quote_type="tick")
```



Currently we support two quote type you can see in `shioaji.constant.QuoteType`. The best way to use that is directly pass this enum into `subscribe` function.

2.2 Place Order

Like the above subscribing market data using the contract, then need to define the order. Pass them into `place_order` function, then it will return the trade that describe the status of your order.

```
contract = api.Contracts.Stocks["2890"]
order = api.Order(
    price=12,
    quantity=5,
```

```
    action=sj.constant.Action.Buy,  
    price_type=sj.constant.StockPriceType.LMT,  
    order_type=sj.constant.OrderType.ROD,  
)  
trade = api.place_order(contract, order)
```

2.3 Conclusion

This quickstart demonstrates how easy to use our package for native Python users. Unlike many other trading API is hard for Python developer. We focus on making more pythonic trading API for our users.

3. Environment Setup

In this section, we will introduce how to setup the python environment with `uv` for using Shioaji API. `uv` is the best solution for managing python environment on cross-platform.

3.0.1 System Requirements

Before starting, please ensure your system meets the following requirements:

- Operating System: 64-bit version of Windows, MacOS, or Linux
- Python Version: 3.8 or later
- User needs to have a Sinopac account and obtain Shioaji API permissions.

3.1 Install Python Environment

First, you need to install Python on your system. We recommend using `uv` as the Python environment and project environment management tool. And we will use `uv` to install Shioaji API in the project.



`uv` is the best solution for managing python environment on cross-platform.

3.1.1 Install uv



Linux and MacOS Windows

```
curl -LsSf https://astral.sh/uv/install.sh | sh
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
```

More information about installation and usage can be found at [uv official document](#)

3.2 Create Project Environment

First, create a project named `sj-trading`

```
uv init sj-trading --package --app --vcs git
cd sj-trading
```

The project structure will be like this:

```
sj-trading
├── README.md
├── pyproject.toml
└── src
    └── sj_trading
        └── __init__.py
```

add Shioaji API to the project

```
uv add shioaji
```

Open `pyproject.toml` file and you will see the following content

```
[project]
name = "sj-trading"
version = "0.1.0"
description = "Shioaji Trading"
readme = "README.md"
requires-python = ">=3.12"
dependencies = [
    "shioaji>=1.2.5",
]

[project.scripts]
hello = "sj_trading:hello"

[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"
```

the `hello` command is the entry point of the project.

execute hello command

```
uv run hello
```

output

```
Hello from sj-trading!
```

open `src/sj_trading/__init__.py` file and copy the following content

```
import shioaji as sj

def hello():
    get_shioaji_client()

def get_shioaji_client() -> sj.Shioaji:
    api = sj.Shioaji()
    print("Shioaji API created")
    return api
```

execute command

```
uv run hello
```

output

```
Shioaji API created
```

This is the most basic environment setup and you can start using Shioaji API now.

3.2.1 Use Jupyter Environment

Add ipykernel to the project development dependencies

```
uv add --dev ipykernel
```

**Add the project environment to the Jupyter kernel**

```
uv run ipython kernel install --user --name=sj-trading
```

**Start Jupyter**

```
uv run --with jupyter jupyter lab
```

Open `dev.ipynb` file in Jupyter and select `sj-trading` kernel to execute the command

The `hello` command we wrote earlier can be executed in this way
`jupyterlab`

If you have already opened an account, you can skip the next chapter and go to [Token & Certificate](#) to get the API Key and certificate.

4. Tutorial - User Guide

4.1 Prepare

4.1.1 Open Account

To use Shioaji, you must have a SinoPac Securities account. If you do not have a SinoPac Securities account yet, please follow the steps below to open an account:

1. To [Open Account](#) Page.

open_acct

2. If you do not have a bank account with Bank SinoPac, please open a bank account as your delivery account.

open_bank_1

3. Please select **我要開DAWHO+大戶投**, to open a bank account and a securities account.

open_bank_2

4. Complete bank and securities account opening.

4.1.2 Token & Certificate

After version 1.0, we will use Token as our login method. Please follow the steps below to apply and use.

APPLY THE API KEY

1. Go to the [API management](#) page in the personal service.

newweb_1

2. Click Add API KEY.

newweb_2

3. Use your mobile phone or email to do two-factor authentication, and the API KEY can only be established if the verification is successful.

newweb_3

4. You can set expiration time, permission, which account can be used, whether it can be used in the production environment and allowed IP list of the key.

newweb_4

Permission Description

- Market / Data : Whether to use the market / data related API
- Account : Whether to use the account related API
- Trading : Whether to use the trading related API
- Production Environment : Whether to use in the production environment

Attention

It is recommended to limit the use of IP, which can improve the security of the KEY.

5. If you add successfully, you will get the API Key and Secret Key.

newweb_5

Attention

- Please keep your key properly and do not disclose it to anyone to avoid property loss.
- The Secret Key is only obtained when the establishment is successful, and there is no way to obtain it after that, please make sure to save it.

DOWNLOAD CERTIFICATE

1. Click the Download Certificate button

newweb_7

2. Download the certificate and place it into the folder that the API can read

newweb_8

Confirm The API Key And Certificate

Continue with the previous project sj-trading , add .env file in the project folder, and add the following content

```
.env
```

```
API_KEY=<API Key>
SECRET_KEY=<Secret Key>
CA_CERT_PATH=<CA Certificate Path>
CA_PASSWORD=<CA Certificate Password>
```

the project folder structure should be like this

```
sj-trading
├── README.md
├── .env
├── pyproject.toml
└── src
    └── sj_trading
        └── __init__.py
uv.lock
```

Add the `python-dotenv` package to load the key and certificate into environment variables

```
uv add python-dotenv
```

Add the following content into `src/sj_trading/__init__.py`

```
import os
from dotenv import load_dotenv

load_dotenv()

def main():
    api = sj.Shioaji(simulation=True)
    api.login(
        api_key=os.environ["API_KEY"],
        secret_key=os.environ["SECRET_KEY"],
        fetch_contract=False
    )
    api.activate_ca(
        ca_path=os.environ["CA_CERT_PATH"],
        ca_passwd=os.environ["CA_PASSWORD"],
    )
    print("Login and activate ca success")
```

Add the `main` command into `pyproject.toml`

```
[project.scripts]
main = "sj_trading"
```

Run the `main` command

```
uv run main
```

If you see `Login and activate ca success`, it means you have successfully logged in to the simulation environment.

Next, if you have not yet completed the API usage signature, please proceed to the next chapter to complete the signature and pass the audit for the API.

4.1.3 Terms of service

Restricted by Taiwan's financial regulations, new users need to sign relevant documents and complete a test report in the simulation mode before using it in a production environment.

Sign Documents

Please refer to [sign center](#) and **read the documents carefully** before you sign.

The screenshot shows the 'Sign Documents' section of the Shioaji app. At the top, there is a red circular logo with a stylized white character inside, followed by the text '線上簽署中心' (Online Signing Center). Below that, the text '證券類' (Securities Category) is displayed. A note at the top left says '一鍵簽署合併版風險預告，簽署項目如下：' (One-click signing of the merged version of the risk disclosure, signing items include:). To the right, three green checkmarks indicate signed documents: '權證風險預告書' (Options Risk Disclosure Document), '附認股權風險預告書' (Subscribed Stock Option Risk Disclosure Document), and '交易外商股票風險預告書' (Foreign Stock Risk Disclosure Document). Below this, there are four rows of documents, each with a green ribbon indicating they have been signed:

- 已簽署 權證風險預告書**
- 已簽署 現股當沖同意書**
- 已簽署 券差借貸契約書**
- 已簽署 指數股票型ETF風險預告書**
- 已簽署 興櫃風險預告書**

In the second row:

- 已簽署 保管劃撥帳戶契約書**
- 不可簽署 融資融券契約書**
- 已簽署 ETN風險預告書**
- 已簽署 轉交換債風險預告書**
- 已簽署 交易外商股票風險預告書**

In the third row:

- 已簽署 交易日商股票風險預告書**
- 已簽署 黃金現貨風險預告書**
- 已簽署 API 證券下單簽署**
- 不可簽署 負面信用資料查詢同意書(信用戶)**
- 不可簽署 負面信用資料查詢同意書(五日型借貸)**

Test Report

To ensure that you fully understand how to use Shioaji, you need to complete the test in the simulation mode, which includes the following functions:

- `login`
- `place_order`

Attention

Service Hour:

- In response to the company's information security regulations, the test service is Monday to Friday 08:00 ~ 20:00
- 18:00 ~ 20:00: Only allow Taiwan IP
- 08:00 ~ 18:00: No limit

Version Restriction:

- version >= 1.2:
install command: `uv add shioaji` or `pip install -U shioaji`

Others:

- You should sign the API related document before you test!
- Stock and Futures account should be test separately.
- The time interval between stock place order test and futures place order test should be more than 1 second.

VERSION CHECK

version

```
import shioaji as sj
print(sj.__version__)
# 1.0.0
```

- please note the **Version Restriction**.

LOGIN TEST

Login

```
version>=1.0      version<1.0
api = sj.Shioaji(simulation=True) # Simulation Mode
api.login(
    api_key="YOUR_API_KEY",       # edit it
    secret_key="YOUR_SECRET_KEY" # edit it
)

api = sj.Shioaji(simulation=True) # Simulation Mode
api.login(
    person_id="YOUR_PERSON_ID",  # edit it
    passwd="YOUR_PASSWORD",      # edit it
)
```

- version >= 1.0: use `api_key` to login, if you haven't applied for the API Key, please refer to [Token](#) section.
- version < 1.0: use `person_id` to login.

PLACE ORDER TEST - STOCK

 Stock Order**version>=1.0 version<1.0**

```
# contract - edit it
contract = api.Contracts.Stocks.TSE["2890"]

# order - edit it
order = api.Order(
    price=18,
    quantity=1,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    account=api.stock_account
)

# place order
trade = api.place_order(contract, order)
trade

# contract - edit it
contract = api.Contracts.Stocks.TSE["2890"]

# order - edit it
order = api.Order(
    price=18,
    quantity=1,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.TFTStockPriceType.LMT,
    order_type=sj.constant.TFTOrderType.ROD,
    account=api.stock_account
)

# place order
trade = api.place_order(contract, order)
trade
```

 Out

Response Code: 0 | Event Code: 0 | Info: host '218.32.76.102:80', IP 218.32.76.102:80 (host 1 of 1) (host connection attempt 1 of 1) (total connection attempt 1 of 1) | Event: Session up

```
Trade(
    contract=Stock(...),
    order=Order(...),
    status=OrderStatus(
        id='531e27af',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        order_quantity=1,
        deals=[]
    )
)
```

- You should receive the message, Response Code: 0 | Event Code: 0 | Info: host '218.32.76.102:80' ... , which means that you have successfully connected to our testing server. The message will only appear on your first order. If you don't receive the connected message, please confirm that all the following conditions are met.
 - a. Doing test in the service hour
 - b. Version restriction
 - c. signed is not present in your account
- order status should **NOT** be Failed . If you got Failed status, please modify your order correctly and then `place_order` again.
- [Contract](#)
- [Stock Order](#)

PLACE ORDER TEST - FUTURES

Future Order**verion>=1.0 verion<1.0**

```
# near-month TXF - edit it
contract = min(
    [
        x for x in api.Contracts.Futures.TXF
        if x.code[-2:] not in ["R1", "R2"]
    ],
    key=lambda x: x.delivery_date
)

# order - edit it
order = api.Order(
    action=sj.constant.Action.Buy,
    price=15000,
    quantity=1,
    price_type=sj.constant.FuturesPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FuturesOCTYPE.Auto,
    account=api.futopt_account
)

# place order
trade = api.place_order(contract, order)
trade

# near-month TXF - edit it
contract = min(
    [
        x for x in api.Contracts.Futures.TXF
        if x.code[-2:] not in ["R1", "R2"]
    ],
    key=lambda x: x.delivery_date
)

# order - edit it
order = api.Order(
    action=sj.constant.Action.Buy,
    price=15000,
    quantity=1,
    price_type=sj.constant.FuturesPriceType.LMT,
    order_type=sj.constant.FuturesOrderType.ROD,
    octype=sj.constant.FuturesOCTYPE.Auto,
    account=api.futopt_account
)

# place order
trade = api.place_order(contract, order)
trade
```

out

```
Response Code: 0 | Event Code: 0 | Info: host '218.32.76.102:80', IP 218.32.76.102:80 (host 1 of 1) (host connection attempt 1 of 1) (total connection attempt 1 of 1) | Event: Session up

Trade(
    contract=Future(...),
    order=Order(...),
    status=OrderStatus(
        id='531e27af',
        status=<Status Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        order_quantity=1,
        deals=[]
    )
)
```

- You should receive the message, Response Code: 0 | Event Code: 0 | Info: host '218.32.76.102:80' ... , which means that you have successfully connected to our testing server. The message will only appear on your first order. If you don't receive the connected message, please confirm that all the following conditions are met.
 - Doing test in the service hour
 - Version restriction
 - signed is not present in your account
- order status should **NOT** be Failed . If you got Failed status, please modify your order correctly and then place_order again.
- [Contract](#)
- [Future Order](#)

CHECK IF API TESTS HAS PASSED

Attention

Before you check, please confirm the following conditions are met.

- Sign the API related document before you test, or you will not pass the test.
- Doing test in service hour.
- Stock accounts and Futures accounts should be tested separately.
- Waiting for reviewing your tests at least 5 minutes.

Sign Status

version>=1.0 version<1.0

```
import shioaji as sj

api = sj.Shioaji(simulation=False) # Production Mode
accounts = api.login(
    api_key="YOUR_API_KEY",          # edit it
    secret_key="YOUR_SECRET_KEY"     # edit it
)
accounts

import shioaji as sj

api = sj.Shioaji(simulation=False) # Production Mode
accounts = api.login(
    person_id="YOUR_PERSON_ID",      # edit it
    passwd="YOUR_PASSWORD",          # edit it
)
accounts
```

 **out**

```
Response Code: 0 | Event Code: 0 | Info: host '203.66.91.161:80', hostname '203.66.91.161:80' IP 203.66.91.161:80 (host 1 of 1) (host connection attempt 1 of 1) (total connection attempt 1 of 1)
| Event: Session up

[FutureAccount(person_id='QBCCAIGJB1', broker_id='F002000', account_id='9100020', signed=True, username='PAPIUSER01'),
 StockAccount(person_id='QBCCAIGJB1', broker_id='9A95', account_id='0504350', username='PAPIUSER01')]
```

- signed=True : Congrats, done! Ex: FutureAccount.
- signed=False or signed not present: the account haven't passed the api tests or haven't been signed the api documents. Ex: StockAccount.

CA

You must **apply** and **activate** the CA before `place_order`.

APPLY CA

1. Go to [SinoPac Securities](#) to download eleader



The screenshot shows the 'Download' section of the SinoPac Securities website. It includes two tables:

熱門下載

類型	下載內容
申請表單	基本資料變更(通訊辦理者需照會本人，更改戶籍地點檢附身分證影本)
電子交易	網路環境 完整環境安裝包(含JAVA安裝)
電子交易	網路環境 純環境調整包(無JAVA安裝)
電子交易	網路環境 JAVA安裝檔
電子交易	網路環境 新版憑證管理網頁元件安裝檔-32位元
電子交易	網路環境 新版憑證管理網頁元件安裝檔-64位元
電子交易	網路環境 複委託IE環境檢測
電子交易	網路環境 如何移除sun java
電子交易	網路環境 Acrobat Reader下載
電子交易	網路環境 API元件下單說明網頁(採申請制，請透過所屬業務同仁)
電子交易	網路環境 原太證憑證元件修正檔

搜尋結果

類型	下載內容
電子交易	下單平台 eLeader (直接下載)
電子交易	下單平台 好神通PLUS (直接下載)
電子交易	下單平台 手機下單
電子交易	下單平台 GLeader國外期貨版 (直接下載)
電子交易	下單平台 進階豐元寶(GPM)國外期貨版 (直接下載)
電子交易	下單平台 憑證管理小APP
電子交易	下單平台 Android TV看盤版
電子交易	下單平台 MAC 總管 (供MAC電腦 申請/更新/查詢憑證)
電子交易	網路環境 完整環境安裝包(含JAVA安裝)

2. Login eleader

永豐金證券 SinoPac Securities

Leader 易利得

身份證字號

密 碼

初始化

隱藏系統公告

完整版

[電子對帳單](#) [申請試用帳號](#) [客服中心](#) [客服信箱](#)

最新消息 [more ▶](#)

- ▶ EZTrade台股功能將於3月12日及EZTrade複委託報價將於3月20日下架通知
- ▶ 參加豐狂存股計畫抽8888元現金
- ▶ 2020/2/17~2/21, 热烈募集「復華新興亞洲3至10年期美元債券指數基金」

電子交易收單時間 [網路下單系統異常應變措施](#)

收單時間	證券交易			期權交易	
	整股下單	定盤下單	零股下單	一般交易	盤後交易
下單時間	08:30~13:30	14:00~14:30	13:40~14:30	8:30~13:45	14:50~次 交易日5:00
預約單時間	14:30~次 交易日08:30	當日13:35 ~14:00	15:30~次 交易日13:40	次交易日 6:00~8:30	無

客服專線 : 02-6630-8899 ; 0800-038-123 , 客服信箱 : service.sec@sinopac.com

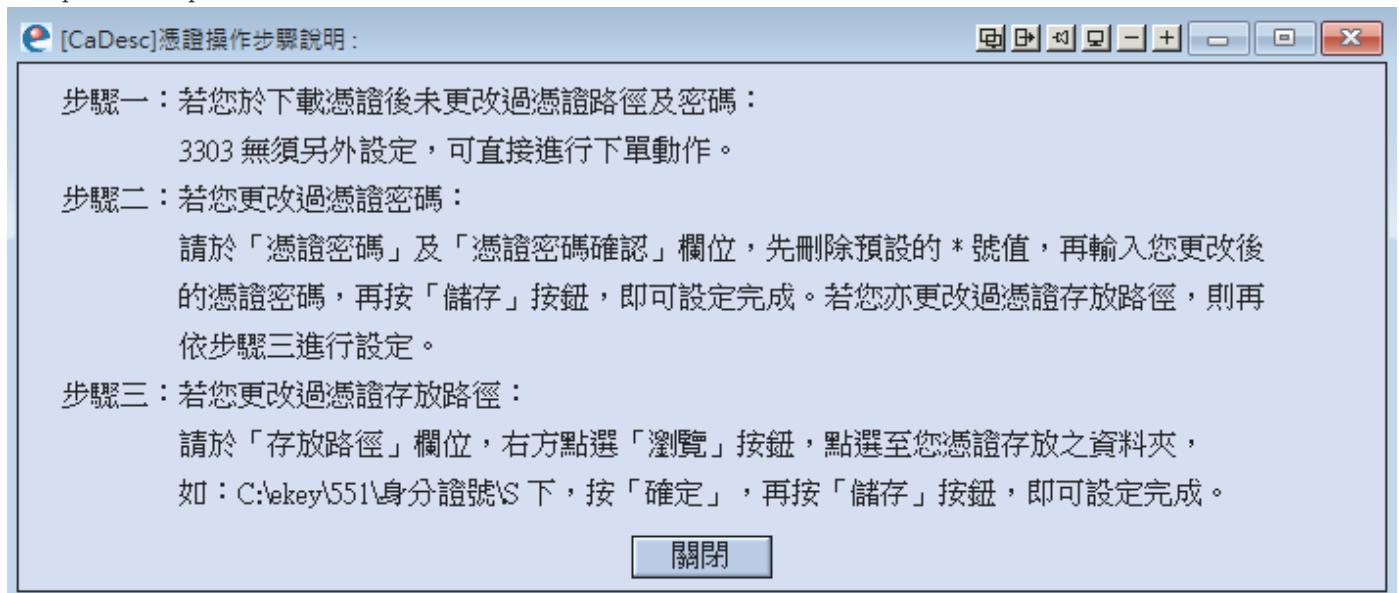
3. Select (3303)帳戶資料設定 from the 帳戶資料 above



4. Click "步驟說明"

The screenshot shows the '3303 帳戶資料設定 : (0) TR<13304>' dialog box. The title bar includes standard window controls. A red box highlights the '步驟說明' (Step Guide) button, which is located above the input fields for '身分證或統一編號', '憑證密碼', and '憑證密碼確認'. Below these fields is a '存放路徑' (Storage Path) input field with a '瀏覽' (Browse) button. At the bottom of the dialog box are two buttons: '登入密碼修改' (Change Login Password) and '憑證密碼修改' (Change Certificate Password). There is also a '儲存' (Save) button in the top right corner.

5. CA Operation steps



ACTIVATE CA

- If you use simulation account, you don't have to activate CA.
- If you are a macOS user, you may subject to version-issue. We suggest you to use [docker](#) and run shioaji service on docker.



```
result = api.activate_ca(
    ca_path="/c/your/ca/path/Sinopac.pfx",
    ca_passwd="YOUR_CA_PASSWORD",
    person_id="Person ID of this Ca",
)
print(result)
# True
```

**The Certification Path**

In Windows you copy the file path with \ to separate the file, you need to replace it with /.

Check CA expire time



```
api.get_ca_expiretime("Person ID")
```

4.1.4 Example Project For Testing Flow

First, we extend the project `sj-trading` created using `uv` in the environment creation chapter to add the testing flow part.

The complete project code can be referred to [sj-trading https://github.com/Sinotrade/sj-trading-demo](https://github.com/Sinotrade/sj-trading-demo).

You can use `git` to clone the entire environment to your local machine and use it directly.

Download Project

```
git clone https://github.com/Sinotrade/sj-trading-demo.git  
cd sj-trading-demo
```

Next, we will step by step introduce how to add the testing flow.

SHIOAJI VERSION

Get Shioaji version information

Add Version Information

Add the following content to `src/sj_trading/__init__.py`

```
def show_version() -> str:  
    print(f"Shioaji Version: {sj__version__}")  
    return sj__version__
```

Add version Command to Project

Add `version` command to `pyproject.toml`

```
[project.scripts]  
version = "sj_trading:show_version"
```

Execute `uv run version` to see the Shioaji version information

```
Shioaji Version: 1.2.0
```

STOCK TESTING**Add Stock Testing File**

Add file `testing_flow.py` to `src/sj_trading`

Add the following content

```
import shioaji as sj
from shioaji.constant import Action, StockPriceType, OrderType
import os

def testing_stock_ordering():
    # Login to the testing environment
    api = sj.Shioaji(simulation=True)
    accounts = api.login(
        api_key=os.environ["API_KEY"],
        secret_key=os.environ["SECRET_KEY"],
    )
    # Show all available accounts
    print(f"Available accounts: {accounts}")
    api.activate_ca(
        ca_path=os.environ["CA_CERT_PATH"],
        ca_passwd=os.environ["CA_PASSWORD"],
    )

    # Prepare the Contract for Ordering
    # Use 2890 Fubon Financial as an example
    contract = api.Contracts.Stocks["2890"]
    print(f"Contract: {contract}")

    # Create an Order for Ordering
    order = sj.order.StockOrder(
        action=Action.Buy, # Buy
        price=contract.reference, # Buy at the reference price
        quantity=1, # Order quantity
        price_type=StockPriceType.LMT, # Limit price order
        order_type=OrderType.ROD, # Effective for the day
        account=api.stock_account, # Use the default account
    )
    print(f"Order: {order}")

    # Send the order
    trade = api.place_order(contract=contract, order=order)
    print(f"Trade: {trade}")

    # Update the status
    api.update_status()
    print(f"Status: {trade.status}")
```

Add stock_testing Command to Project

Add `stock_testing` command to `pyproject.toml`

```
[project.scripts]
stock_testing = "sj_trading.testing_flow:testing_stock_ordering"
```

Execute `uv run stock_testing` to start testing stock ordering

FUTURES TESTING**Add Futures Testing Content**

Add the following content to `src/sj_trading/testing_flow.py`

```
from shioaji.constant import (
    FuturesPriceType,
    FuturesOCType,
)

def testing_futures_ordering():
    # Login to the testing environment
    api = sj.Shoaji(simulation=True)
    accounts = api.login(
        api_key=os.environ["API_KEY"],
        secret_key=os.environ["SECRET_KEY"],
    )
    # Show all available accounts
    print(f"Available accounts: {accounts}")
    api.activate_ca(
        ca_path=os.environ["CA_CERT_PATH"],
        ca_passwd=os.environ["CA_PASSWORD"],
    )

    # Get the contract for ordering
    # Use TXFR1 as an example
    contract = api.Contracts.Futures["TXFR1"]
    print(f"Contract: {contract}")

    # Create an Order for Ordering
    order = sj.order.FuturesOrder(
        action=Action.Buy, # Buy
        price=contract.reference, # Buy at the reference price
        quantity=1, # Order quantity
        price_type=FuturesPriceType.LMT, # Limit price order
        order_type=OrderType.ROD, # Effective for the day
        octype=FuturesOCType.Auto, # Auto select new close
        account=api.futopt_account, # Use the default account
    )
    print(f"Order: {order}")

    # Send the order
    trade = api.place_order(contract=contract, order=order)
    print(f"Trade: {trade}")

    # Update the status
    api.update_status()
    print(f"Status: {trade.status}")
```

Add futures_testing Command to Project

Add `futures_testing` command to `pyproject.toml`

```
[project.scripts]
futures_testing = "sj_trading.testing_flow:testing_futures_ordering"
```

Execute `uv run futures_testing` to start testing futures ordering

4.2 Login

Login must have a SinoPac Securities account. If you do not have a SinoPac Securities account yet. See the [document](#) for details.

4.2.1 Login

Token login

After version 1.0, we are using token as our `login` method. You can be found in [Token](#). Before version 1.0, using person id and password.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY"
)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
)
```



```
[FutureAccount(person_id='', broker_id='', account_id='', signed=True, username=''),
StockAccount(person_id='', broker_id='', account_id='', signed=True, username='')]
```

- If you cannot find `signed` in your accounts, please refer to [terms of service](#) first.

Login Arguments

version>=1.0 version<1.0

```
api_key (str): API Key
secret_key (str): Secret Key
fetch_contract (bool): whether to load contracts from cache or server (Default: True)
contracts_timeout (int): fetch contract timeout (Default: 0 ms)
contracts_cb (typing.Callable): fetch contract callback (Default: None)
subscribe_trade (bool): whether to subscribe Order/Deal event callback (Default: True)
receive_window (int): valid duration for login execution. (Default: 30,000 ms)

person_id (str): person_id
passwd (str): password
hashed (bool): whether password has been hashed (Default: False)
fetch_contract (bool): whether to load contracts from cache or server (Default: True)
contracts_timeout (int): fetch contract timeout (Default: 0 ms)
contracts_cb (typing.Callable): fetch contract callback (Default: None)
subscribe_trade (bool): whether to subscribe Order/Deal event callback (Default: True)
```

Warning

When the version is greater than 1.0, you may receive **Sign data is timeout** when login. That is, login has exceeded the effective execution time. It may be that the time difference between your computer and server is too large, you need to calibrate your computer time. Or login execution time exceeds valid time, you can increase `receive_window`.

Fetch Contracts Callback

You can use `contracts_cb` as print to check contract download status.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY",
    contracts_cb=lambda security_type: print(f"{repr(security_type)} fetch done.")
)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
    contracts_cb=lambda security_type: print(f"{repr(security_type)} fetch done.")
)
```



```
[
    FutureAccount(person_id='', broker_id='', account_id='', signed=True, username=''),
    StockAccount(person_id='', broker_id='', account_id='', signed=True, username='')
]
<SecurityType.Index: 'IND'> fetch done.
<SecurityType.Future: 'FUT'> fetch done.
<SecurityType.Option: 'OPT'> fetch done.
<SecurityType.Stock: 'STK'> fetch done.
```

Subscribe Trade

There are 2 options that you can adjust whether to subscribe trade (Order/Deal Event Callback).

The first is `subscribe_trade` in login arguments. Default value of `subscribe_trade` is `True`, and it will automatically subscribe trade from all accounts. You don't need to make any adjustments, if you would like to receive Order/Deal Events.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY",
    subscribe_trade=True
)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
    subscribe_trade=True
)
```

The second one is to manually use the API `subscribe_trade` and `unsubscribe_trade` for specific account.

subscribe trade

```
api.subscribe_trade(account)
```

unsubscribe trade

```
api.unsubscribe_trade(account)
```

4.2.2 Account

List Accounts



```
accounts = api.list_accounts()
```



```
# print(accounts)
[
    FutureAccount(person_id='PERSON_ID_1', broker_id='BROKER_ID_1', account_id='ACCOUNT_ID_1', signed=True, username='USERNAME_1'),
    FutureAccount(person_id='PERSON_ID_2', broker_id='BROKER_ID_2', account_id='ACCOUNT_ID_2', username='USERNAME_2'),
    StockAccount(person_id='PERSON_ID_3', broker_id='BROKER_ID_3', account_id='ACCOUNT_ID_3', username='USERNAME_3'),
    StockAccount(person_id='PERSON_ID_4', broker_id='BROKER_ID_4', account_id='ACCOUNT_ID_4', signed=True, username='USERNAME_4')
]
```

- If `signed` does not appear in the account list, like `ACCOUNT_ID_2` and `ACCOUNT_ID_3`, it means that the account has not signed or completed the test report in the simulation mode. Please refer to [Terms of service](#).

Default Account



```
# Futures default account
print(api.futopt_account)
```



```
FutureAccount(person_id='PERSON_ID_1', broker_id='BROKER_ID_1', account_id='ACCOUNT_ID_1', signed=True, username='USERNAME_1')
```

Set default account



```
# Default futures account switch to ACCOUNT_ID_2 from ACCOUNT_ID_1.
api.set_default_account(accounts[1])
print(api.futopt_account)
```



```
FutureAccount(person_id='PERSON_ID_2', broker_id='BROKER_ID_2', account_id='ACCOUNT_ID_2', username='USERNAME_2')
```

In Order object, you need to specify which account you want to place order. For more information about Order, please refer to [Stock Order](#) and [Futures Order](#).

```
order = api.Order(  
    price=12,  
    quantity=1,  
    action=sj.constant.Action.Buy,  
    price_type=sj.constant.StockPriceType.LMT,  
    order_type=sj.constant.OrderType.ROD,  
    order_lot=sj.constant.StockOrderLot.Common,  
    account=api.stock_account  
)
```

4.2.3 Logout

Logout function will close the connection between the client and the server.

In order to provide high quality services, starting from 2021/08/06, we've limit the [number of connections used](#). It's a good practice to logout or to terminate the program when it is not in use.

logout

```
api.logout()  
# True
```

4.3 Contract

Contract object will be used by a lot of place like place order, subscribe quote, etc.

Get Contracts

The following provides two methods to get contracts:

- method 1: After `Login` success we will start to fetch all kind of contract but fetching will not block other action. So how to know the fetch action is done ? We have status of contracts download that you can use `Contracts.status`. If you set `contracts_timeout` inside `login` set to 10000, it will block the fetch and wait 10 second until the contract is back.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY",
    contracts_timeout=10000,
)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
    contracts_timeout=10000,
)
```

- method 2: If `fetch_contract` inside `login` is set to False, it will not download contract. You can use `fetch_contracts` to download.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY",
    fetch_contract=False,
)
api.fetch_contracts(contract_download=True)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
    fetch_contract=False,
)
api.fetch_contracts(contract_download=True)
```

Contracts Information

The contracts we currently offer include: stocks, futures, options and indices. The products we provide can get more detailed information through the following ways.



```
api.Contracts
```



Contracts(Indexes=(OTC, TSE), Stocks=(OES, OTC, TSE), Futures=(BRF, CAF, CBF, CCF, CDF, CEF, CFF, CGF, CHF, CJF, CK1, CKF, CLF, CM1, CMF, CNF, CQF, CRF, CSF, CU1, CUF, CWF, CXF, CYF, CZ1, CZF, DCF, DD1, DDF, DEF, DFF, DGF, DHF, DIF, DJF, DKF, DLF, DNF, DDF, DPF, DQF, DSF, DUF, DVF, DWF, DXF, DYF, DZF, EEF, EGF, EHF, EMF, EPF, ERF, EXF, EY1, EYF, FEF, FFF, FGF, FKF, FQF, FRF, FTF, FVF, FWF, FXF, FYF, FZF, G2F, GAF, GCF, GDF, GHF, GIF, GLF, GMF, GNF, GOF, GRF, GTF, GUF, GWF, GXF, HAF, HBF, HCF, HHF, HIF, HLF, HOF, HS1, HSF, HY1, HYF, IAI, IAF, IHF, IIF, IJF, IMF, IOF, IPF, IQF, IRF, ITF, IXF, IVF, IZF, JBF, JF, JNF, JPF, JSF, JWF, JZF, KAF, KB1, KBF, KCF, KDF, KFF, KGF, KIF, KKF, KLF, KOF, KPF, KSF, KWF, LBF, LCF, LE1, LEF, LIF, LMF, LOF, LQF, LRF, LTF, LU1, LV1, LW1, LX1, LY1, MAF, MBF, MCF, MJF, MKF, MPF, MQF, NWF, MX1, MXF, MYF, NAF, NBF, NCF, NDF, NEF, NGF, NHF, NIF, NJF, NLF, NMF, NNF, NOF, NSF, NU1, NUF, NWF, NXF, NYF, NZF, OAF, OBF, OCF, ODF, OEF, OGF, OHF, OJF, OKF, OLF, OMF, OOF, OPF, QOF, ORF, OS1, OSF, OTF, OUF, OV1, OVF, OVW, OXF, PAF, PBF, PCF, PDF, PEF, PFF, PGF, PHF, P1F, PKF, PLF, PMF, PNF, POF, PPF, PQF, RHF, RTF, SPF, TSF, TGF, TJF, TXF, UDF, UNF, XAF, XBF, XEF, XIF, XJF), Options=(CAO, CBO, CCO, CDO, CEO, CFO, CGO, CHO, CJO, CKO, CLO, CMO, CNO, CQO, CRO, CSO, CXO, CZO, DCO, DEO, DFO, DGO, DHO, DJO, DKO, DLO, DNO, DOO, DPO, DQO, DSO, DUO, DVO, DWF, DZO, G1O, G2O, HCO, IJO, LOO, NYA, NYO, NZO, OAO, OBO, OCO, OJO, OKO, OOO, OZO, RHO, RTO, TEO, TFO, TGO, TX1, TXO))

STOCK



```
api.Contracts.Stocks["2890"]
# or api.Contracts.Stocks.TSE.TSE2890
```



```
Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2890',
    symbolL='TSE2890',
    name='永豐金',
    category='17',
    unit=1000,
    limit_up=19.1,
    limit_down=15.7,
    reference=17.4,
    update_date='2023/01/17',
    day_trade=<DayTrade.Yes: 'Yes'>
)
```



exchange (Exchange): Attributes of industry
 (OES, OTC, TSE ...etc)
 code (str): Id
 symbol (str): Symbol
 name (str): Name
 category (str): Category
 unit (int): Unit
 limit_up (float): Limit up
 limit_down (float): Limit down
 reference (float): Reference price
 update_date (str): Update date
 margin_trading_balance (int): Margin trading balance
 short_selling_balance (int): Short selling balance
 day_trade (DayTrade): Day trade
 {Yes, No, OnlyBuy}



```
Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2890',
    symbolL='TSE2890',
    name='永豐金',
    category='17',
    unit=1000,
    limit_up=19.1,
    limit_down=15.7,
    reference=17.4,
    update_date='2023/01/17',
    day_trade=<DayTrade.Yes: 'Yes'>
)
```

FUTURES

```
api.Contracts.Futures["TXFA3"]
# or api.Contracts.Futures.TXF.TXF202301
```



```
Future(
    code='TXFA3',
    symbol='TXF202301',
    name='臺股期貨01',
    category='TXF',
    delivery_month='202301',
    delivery_date='2023/01/30',
    underlying_kind='I',
    unit=1,
    limit_up=16417.0,
    limit_down=13433.0,
    reference=14925.0,
    update_date='2023/01/18'
)
```

**Future**

```
code (str): Id
symbol (str): Symbol
name (str): Name
category (str): Category
delivery_month (str): Delivery Month
delivery_date (str): Delivery Date
underlying_kind (str): Underlying Kind
unit (int): Unit
limit_up (float): Limit up
limit_down (float): Limit down
reference (float): Reference price
update_date (str): Update date
```

OPTIONS

```
api.Contracts.Options["TX018000R3"]
# or api.Contracts.Options.TXO.TX020230618000P
```



```
Option(
    code='TX018000R3',
    symbol='TX020230618000P',
    name='臺指選擇權06月 18000P',
    category='TXO',
    delivery_month='202306',
    delivery_date='2023/06/21',
    strike_price=18000,
    option_right=<OptionRight.Put: 'P'>,
    underlying_kind='I',
    unit=1,
    limit_up=4720.0,
    limit_down=1740.0,
    reference=3230.0,
    update_date='2023/01/18'
)
```

Option

```
code (str): Id
symbol (str): Symbol
name (str): Name
category (str): Category
delivery_month (str): Delivery Month
delivery_date (str): Delivery Date
strike_price (int or float): Strike Price
option_right (OptionRight): Option Right
underlying_kind (str): Underlying Kind
unit (int): Unit
limit_up (float): Limit up
limit_down (float): Limit down
reference (float): Reference price
update_date (str): Update date
```

INDEX

The `Index` object shows all supported index contracts, among other categories. Index contracts do not support `place_order`, but allow subscribing to market quotes. This will be discussed in the next topic.



```
api.Contracts.Indexes.TSE
```



```
TSE(TSE001, TSE015, TSE016, TSE017, TSE018, TSE019, TSE020, TSE022, TSE023, TSE024, TSE025, TSE026, TSE028, TSE029, TSE030, TSE031, TSE032, TSE033, TSE035, TSE036, TSE037, TSE038, TSE039, TSE040, TSE041, TSE042, TSE043, TSE004, TSE005)
```



```
api.Contracts.Indexes.TSE["001"]
# or api.Contracts.Indexes.TSE.TSE001
```



```
Index(
    exchange=<Exchange.TSE: 'TSE'>,
    code='001',
    symbol='TSE001',
    name='加權指數'
)
```



```
exchange (Exchange): exchange
    {OES, OTC, TSE ...etc}
code (str): Code
symbol (str): Symbol
name (str): Name
```

Contract Update Information

- 07:50 Futures contract update
- 08:00 All market contract update
- 14:45 Futures night contract update
- 17:15 Futures night contract update

4.4 Market Data

4.4.1 Streaming Market Data

Stocks

To subscribe quotes is very easy, just call `subscribe` function with `contract` which we've discussed in previous topic.

Subscribe

```
>> api.quote.subscribe?

Signature:
api.quote.subscribe(
    contract:shioaji.contracts.Contract,
    quote_type:shioaji.constant.QuoteType=<QuoteType.Tick: 'tick'>,
    intraday_odd:bool=False,
    version: shioaji.constant.QuoteVersion.v0: 'v0'>,
)
```

Quote Parameters

```
quote_type: tick price or bid/ask price to subscribe
{'tick', 'bidask'}
intraday_odd: 盤中零股
{True, False}
version: version of quote format
{'v1', 'v0'}
```

TICK

Common Stock


```
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick,
    version = sj.constant.QuoteVersion.v1
)
```



QuoteVersion.v1 QuoteVersion.v0

Response Code: 200 | Event Code: 16 | Info: TIC/v1/STK/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
Exchange.TSE
Tick(
    code = '2330',
    datetime = datetime.datetime(2021, 7, 2, 13, 16, 35, 92970),
    open = Decimal('590'),
    avg_price = Decimal('589.05'),
    close = Decimal('590'),
    high = Decimal('593'),
    low = Decimal('587'),
    amount = Decimal('590000'),
    total_amount = Decimal('8540101000'),
    volume = 1,
    total_volume = 14498,
    tick_type = 1,
    chg_type = 4,
    price_chg = Decimal('-3'),
    pct_chg = Decimal('-.505902'),
    bid_side_total_vol = 6638,
    ask_side_total_vol = 7860,
    bid_side_total_cnt = 2694,
    ask_side_total_cnt = 2705,
    closing_ oddlot_shares = 0,
    fixed_trade_vol = 0,
    suspend = 0,
    simtrade = 0,
    intraday_ odd = 0
)
```

Response Code: 200 | Event Code: 16 | Info: MKT/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
MKT/1dcdmzpcr01/TSE/2330
{
    'AmountSum': [1688787000.0],
    'Close': [593.0],
    'Date': '2021/07/01',
    'TickType': [2],
    'Time': '09:10:20.628620',
    'VolSum': [2837],
    'Volume': [1]
}
```

Intraday odd



```
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick,
    version = sj.constant.QuoteVersion.v1,
    intraday_ odd = True
)
```

 Out**QuoteVersion.v1 QuoteVersion.v0**

Response Code: 200 | Event Code: 16 | Info: TIC/v1/ODD/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
Exchange.TSE
Tick(
    code = '2330',
    datetime = datetime.datetime(2021, 7, 2, 13, 16, 55, 544646),
    open = Decimal('591'),
    avg_price = Decimal('590.24415'),
    close = Decimal('590'),
    high = Decimal('591'),
    low = Decimal('589'),
    amount = Decimal('276120'),
    total_amount = Decimal('204995925'),
    volume = 468,
    total_volume = 347307,
    tick_type = 1,
    chg_type = 4,
    price_chg = Decimal('-3'),
    pct_chg = Decimal('-.505902'),
    bid_side_total_vol = 68209,
    ask_side_total_vol = 279566,
    bid_side_total_cnt = 28,
    ask_side_total_cnt = 56,
    closing_ oddlot_shares = 0,
    fixed_trade_vol = 0,
    suspend = 0,
    simtrade = 1,
    intraday_odd = 1
)
```

Response Code: 200 | Event Code: 16 | Info: TIC/v2/*/TSE/2330/ODDLLOT | Event: Subscribe or Unsubscribe ok

```
TIC/v2/replay/TSE/2330/ODDLLOT
{
    'Date': '2021/07/01',
    'Time': '09:23:36.880878',
    'Close': '593',
    'TickType': 1,
    'Shares': 1860,
    'SharesSum': 33152,
    'Simtrade': 1
}
```

Attributes Tick**QuoteVersion.v1 QuoteVersion.v0**

```
code (str): 商品代碼
datetime (datetime): 時間
open (decimal): 開盤價
avg_price (decimal): 均價
close (decimal): 成交價
high (decimal): 最高價(自開盤)
low (decimal): 最低價(自開盤)
amount (decimal): 成交額 (NTD)
total_amount (decimal): 總成交額 (NTD)
volume (int): 成交量 (整股:張, 盤中零股:股)
total_volume (int): 總成交量 (整股:張, 盤中零股:股)
tick_type (int): 內外盤別{1: 外盤, 2: 內盤, 0: 無法判定}
chg_type (int): 漲跌註記{1: 漲停, 2: 漲, 3: 平盤, 4: 跌, 5: 跌停}
price_chg (decimal): 漲跌幅
pct_chg (decimal): 漲跌幅
bid_side_total_vol (int): 買盤成交總量 (整股:張, 盤中零股:股)
ask_side_total_vol (int): 賣盤成交總量 (整股:張, 盤中零股:股)
bid_side_total_cnt (int): 買盤成交筆數
ask_side_total_cnt (int): 賣盤成交筆數
closing_ oddlot_shares (int): 盤後零股成交股數(股)
fixed_trade_vol (int): 定盤成交量 (整股:張, 盤中零股:股)
suspend (bool): 暫停交易
simtrade (bool): 試撥
intraday_odd (int): 盤中零股 {0: 整股, 1:盤中零股}

AmountSum (:List:float): 總成交額
Close (:List:float): 成交價
Date (str): 日期 (yyyy/MM/dd)
TickType (:List:int): 內外盤別{1: 外盤, 2: 內盤, 0: 無法判定}
Time (str): 時間 (HH:mm:ss.fffff)
VolSum (:List:int): 總成交量 (張)
Volume (:List:int): 成交量 (張)
```

BIDASK**Common Stock**

```
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.BidAsk,
    version = sj.constant.QuoteVersion.v1
)
```

**QuoteVersion.v1 QuoteVersion.v0**

Response Code: 200 | Event Code: 16 | Info: QU0/v1/STK/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
Exchange.TSE
BidAsk(
    code = '2330',
    datetime = datetime.datetime(2021, 7, 1, 9, 9, 54, 36828),
    bid_price = [Decimal('593'), Decimal('592'), Decimal('591'), Decimal('590'), Decimal('589')],
    bid_volume = [248, 180, 258, 267, 163],
    diff_bid_vol = [3, 0, 0, 0],
    ask_price = [Decimal('594'), Decimal('595'), Decimal('596'), Decimal('597'), Decimal('598')],
    ask_volume = [1457, 531, 506, 90, 259],
    diff_ask_vol = [0, 0, 0, 0],
    suspend = 0,
    simtrade = 0,
    intraday_odd = 0
)
```

Response Code: 200 | Event Code: 16 | Info: QUT/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
QUT/idcdmzpcr01/TSE/2330
{
    'AskPrice': [594.0, 595.0, 596.0, 597.0, 598.0],
    'AskVolume': [1465, 532, 507, 92, 258],
    'BidPrice': [593.0, 592.0, 591.0, 590.0, 589.0],
    'BidVolume': [254, 178, 255, 268, 163],
    'Date': '2021/07/01',
    'Time': '09:09:48.447219'
}
```

Intraday odd

```
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.BidAsk,
    version = sj.constant.QuoteVersion.v1
    intraday_odd=True
)
```



QuoteVersion.v1 QuoteVersion.v0

Response Code: 200 | Event Code: 16 | Info: QU0/v1/ODD/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

```
Exchange.TSE
BidAsk(
    code = '2330',
    datetime = datetime.datetime(2021, 7, 2, 13, 17, 45, 743299),
    bid_price = [Decimal('589'), Decimal('588'), Decimal('587'), Decimal('586'), Decimal('585')],
    bid_volume = [59391, 224490, 74082, 68570, 125246],
    diff_bid_vol = [49874, 101808, 23863, 38712, 77704],
    ask_price = [Decimal('590'), Decimal('591'), Decimal('592'), Decimal('593'), Decimal('594')],
    ask_volume = [26355, 9680, 18087, 11773, 3568],
    diff_ask_vol = [13251, -14347, 39249, -20397, -10591],
    suspend = 0,
    simtrade = 1,
    intraday_odd = 1
)
```

Response Code: 200 | Event Code: 16 | Info: QU0/v2/*/TSE/2330/ODDLLOT | Event: Subscribe or Unsubscribe ok

```
QU0/v2/replay/TSE/2330/ODDLLOT
{
    'Date': '2021/07/01',
    'Time': '09:43:47.143789',
    'BidPrice': ['592', '591', '590', '589', '588'],
    'AskPrice': ['593', '594', '595', '596', '597'],
    'BidShares': [16979, 12009, 45045, 5501, 12956],
    'AskShares': [17276, 14823, 26518, 23388, 10527],
    'Simtrade': 1
}
```

Attributes



QuoteVersion.v1 QuoteVersion.v0

```
code (str): 商品代碼
datetime (datetime): 時間
bid_price (:List:decimal): 委買價
bid_volume (:List:int): 委買量 (張)
diff_bid_vol (:List:int): 買價增減量 (張)
ask_price (:List:decimal): 委賣價
ask_volume (:List:int): 委賣量
diff_ask_vol (:List:int): 賣價增減量 (張)
suspend (bool): 暫停交易
simtrade (bool): 試撮

AskPrice (:List:float): 委賣價
AskVolume (:List:int): 委賣量
BidPrice (:List:float): 委買價
BidVolume (:List:int): 委買量
Date (datetime.date): 日期 (yyyy/MM/dd)
Time (time): 時間 (HH:mm:ss.fffffffff)
```

QUOTE

Common Stock



```
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Quote,
    version = sj.constant.QuoteVersion.v1
)
```

Out

```
Response Code: 200 | Event Code: 16 | Info: QU0/v2/STK/*/TSE/2330 | Event: Subscribe or Unsubscribe ok

Exchange.TSE,
Quote(
    code='2330',
    datetime=datetime.datetime(2022, 7, 1, 10, 43, 15, 430329),
    open=Decimal('471.5'),
    avg_price=Decimal('467.91'),
    close=Decimal('461'),
    high=Decimal('474'),
    low=Decimal('461'),
    amount=Decimal('461000'),
    total_amount=Decimal('11834476000'),
    volume=1,
    total_volume=25292,
    tick_type=2,
    chg_type=4,
    price_chg=Decimal('-15'),
    pct_chg=Decimal('-3.15'),
    bid_side_total_vol=9350,
    ask_side_total_vol=15942,
    bid_side_total_cnt=2730,
    ask_side_total_cnt=2847,
    closing_oddlot_shares=0,
    closing_oddlot_close=Decimal('0.0'),
    closing_oddlot_amount=Decimal('0'),
    closing_oddlot_bid_price=Decimal('0.0'),
    closing_oddlot_ask_price=Decimal('0.0'),
    fixed_trade.vol=0,
    fixed_trade.amount=Decimal('0'),
    bid_price=[Decimal('461'), Decimal('460.5'), Decimal('460'), Decimal('459.5'), Decimal('459')],
    bid.volume=[220, 140, 994, 63, 132],
    diff_bid.vol=[-1, 0, 0, 0, 0],
    ask_price=[Decimal('461.5'), Decimal('462'), Decimal('462.5'), Decimal('463'), Decimal('463.5')],
    ask.volume=[115, 101, 103, 147, 91],
    diff_ask.vol=[0, 0, 0, 0, 0],
    avail_borrowing=9579699,
    suspend=0,
    simtrade=0
)
```

Attributes

Quote

```
code (str): 商品代碼
datetime (datetime): 時間
open (decimal): 開盤價
avg_price (decimal): 均價
close (decimal): 成交價
high (decimal): 最高價(自開盤)
low (decimal): 最低價(自開盤)
amount (decimal): 成交額 (NTD)
total_amount (decimal): 總成交額 (NTD)
volume (int): 成交量
total_volume (int): 總成交量
tick_type (int): 總外盤別
chg_type (int): 漲跌註記
price_chg (decimal): 漲跌價
pct_chg (decimal): 漲跌率
bid_side_total_vol (int): 買盤成交總量 (張)
ask_side_total_vol (int): 賣盤成交總量 (張)
bid_side_total_cnt (int): 買盤成交筆數
ask_side_total_cnt (int): 賣盤成交筆數
closing_oddlot_shares (int): 盤後零股成交股數
closing_oddlot_close (decimal): 盤後零股成交價
closing_oddlot_amount (decimal): 盤後零股成交額
closing_oddlot_bid_price (decimal): 盤後零股買價
closing_oddlot_ask_price (decimal): 盤後零股賣價
fixed_trade.vol (int): 定盤成交量 (張)
fixed_trade.amount (decimal): 定盤成交額
bid_price (:List:decimal): 買價
bid.volume (:List:int) 買量
diff_bid.vol (:List:int) 買價增減量
ask_price (:List:decimal): 賣價
ask.volume (:List:int) 賣量
diff_ask.vol (:List:int) 賣價增減量
avail_borrowing (int): 借券可用餘額
suspend (bool): 暫停交易
simtrade (bool): 試撮
```

CALLBACK

In default, we set quote callback as print function. You can modify callback function as you wish. Just remember, always avoid making calulations inside the callback function.

Tick**i: pythonic way by using decorator****QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import TickSTKv1, Exchange

@api.on_tick_stk_v1()
def quote_callback(exchange: Exchange, tick:TickSTKv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

i: traditional way**QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import TickSTKv1, Exchange

def quote_callback(exchange: Exchange, tick:TickSTKv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

api.quote.set_on_tick_stk_v1_callback(quote_callback)

def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")

api.quote.set_quote_callback(quote_callback)
```

**QuoteVersion.v1 QuoteVersion.v0**

```
Exchange: Exchange.TSE, Tick: Tick(code='2330', datetime=datetime.datetime(2021, 7, 2, 13, 16, 35, 92970), open=Decimal('590'), avg_price=Decimal('589.05'), close=Decimal('590'), high=Decimal('593'), low=Decimal('587'), amount=Decimal('590000'), total_amount=Decimal('8540101000'), volume=1, total_volume=14498, tick_type=1, chg_type=4, price_chg=Decimal('-3'), pct_chg=Decimal('-0.505902'), trade_bid_volume=6638, ask_side_total_vol=7860, bid_side_total_cnt=2694, ask_side_total_cnt=2705, closing_ordinal_shares=0, fixed_trade_vol=0, suspend=0, simtrade=0, intraday_ode=0)

Topic: MKT/*TSE/2330, Quote: {'AmountSum': [4739351000.0], 'Close': [596.0], 'Date': '2021/03/30', 'TickType': [2], 'Time': '10:01:33.349431', 'VolSum': [7932], 'Volume': [1]}
```

BidAsk**i: pythonic way by using decorator****QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import BidAskSTKv1, Exchange

@api.on_bidask_stk_v1()
def quote_callback(exchange: Exchange, bidask:BidAskSTKv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

iii: traditional way

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import BidAskSTKv1, Exchange

def quote_callback(exchange: Exchange, bidask:BidAskSTKv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

api.quote.set_on_bidask_stk_v1_callback(quote_callback)

def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")

api.quote.set_quote_callback(quote_callback)
```

Cut

QuoteVersion.v1 QuoteVersion.v0

```
Exchange: Exchange.TSE, BidAsk: BidAsk(code='2330', datetime=datetime.datetime(2021, 7, 2, 13, 17, 29, 726428), bid_price=[Decimal('589'), Decimal('588'), Decimal('587'), Decimal('586'), Decimal('585')], bid_volume=[223, 761, 1003, 809, 1274], diff_bid_vol=[0, 0, 0, 0, 0], ask_price=[Decimal('590'), Decimal('591'), Decimal('592'), Decimal('593'), Decimal('594')], ask_volume=[304, 232, 183, 242, 131], diff_ask_vol=[1, 0, 0, 0, 0], suspend=0, simtrade=0, intraday_odd=0)

Topic: QUT/idecmzpcr01/TSE/2330, Quote: {'AskPrice': [590.0, 591.0, 592.0, 593.0, 594.0], 'AskVolume': [303, 232, 183, 242, 131], 'BidPrice': [589.0, 588.0, 587.0, 586.0, 585.0], 'BidVolume': [224, 762, 1003, 809, 1274], 'Date': '2021/07/02', 'Time': '13:17:26.391840'}
```

Quote

iii: pythonic way by using decorator

```
from shioaji import QuoteSTKv1, Exchange

@api.on_quote_stk_v1()
def quote_callback(exchange: Exchange, quote:QuoteSTKv1):
    print(f"Exchange: {exchange}, Quote: {quote}")
```

iii: traditional way

```
from shioaji import QuoteSTKv1, Exchange

def quote_callback(exchange: Exchange, quote:QuoteSTKv1):
    print(f"Exchange: {exchange}, Quote: {quote}")

api.quote.set_on_quote_stk_v1_callback(quote_callback)
```

Cut

```
Exchange: TSE, Quote: Quote(code='2330', datetime=datetime.datetime(2022, 7, 1, 10, 43, 15, 430329), open=Decimal('471.5'), avg_price=Decimal('467.91'), close=Decimal('461'), high=Decimal('474'), low=Decimal('461'), amount=Decimal('461000'), total_amount=Decimal('11834476000'), volume=1, total_volume=25292, tick_type=2, chg_type=4, price_chg=Decimal('-15'), pct_chg=Decimal('-.15'), bid_side_total_vol=9350, ask_side_total_vol=15942, bid_side_total_cnt=2730, ask_side_total_cnt=2847, closing_oddlot_shares=0, closing_oddlot_close=Decimal('0.0'), closing_oddlot_amount=Decimal('0'), closing_oddlot_bid_price=Decimal('0.0'), closing_oddlot_ask_price=Decimal('0.0'), fixed_trade_vol=0, fixed_trade_amount=Decimal('0'), bid_price=[Decimal('461'), Decimal('460.5'), Decimal('460'), Decimal('459.5'), Decimal('459')], bid_volume=[220, 140, 994, 63, 132], diff_bid_vol=[-1, 0, 0, 0, 0], ask_price=[Decimal('461.5'), Decimal('462'), Decimal('462.5'), Decimal('463'), Decimal('463.5')], ask_volume=[115, 101, 103, 147, 91], diff_ask_vol=[0, 0, 0, 0, 0], avail_borrowing=9579699, suspend=0, simtrade=0)
```

- Intraday odd share the callback function with common stock.
- Advanced quote callback settings please refer to [Quote-Binding Mode](#).

Futures

To subscribe quotes is very easy, just call `subscribe` function with `contract` which we've discussed in previous topic.

Subscribe

```
api.quote.subscribe?

Signature:
    api.quote.subscribe(
        contract:shioaji.contracts.Contract,
        quote_type:shioaji.constant.QuoteType=<QuoteType.Tick: 'tick'>,
        intraday_odd:bool=False,
        version: shioaji.constant.QuoteVersion=<QuoteVersion.v0: 'v0'>,
    )
Docstring: <no docstring>
Type:      method
```

Quote Parameters:

```
quote_type: tick price or bid/ask price to subscribe
{'tick', 'bidask', 'quote'}
intraday_odd: 盤中零股
{True, False}
version: version of quote format
{'v1', 'v0'}
```

TICK

Example



```
api.quote.subscribe(
    api.Contracts.Futures.TXF['TF202107'],
    quote_type = sj.constant.QuoteType.Tick,
    version = sj.constant.QuoteVersion.v1,
)
```



QuoteVersion.v1 QuoteVersion.v0

Response Code: 200 | Event Code: 16 | Info: TIC/v1/FOP/*/TFE/TXFG1 | Event: Subscribe or Unsubscribe ok

```
Exchange.TAIFEX
Tick(
    code = 'TXFG1',
    datetime = datetime.datetime(2021, 7, 1, 10, 42, 29, 757000),
    open = Decimal('17678'),
    underlying_price = Decimal('17849.57'),
    bid_side_total_vol= 32210,
    ask_side_total_vol= 33218,
    avg_price = Decimal('17704.663999'),
    close = Decimal('17753'),
    high = Decimal('17774'),
    low = Decimal('17655'),
    amount = Decimal('17753'),
    total_amount = Decimal('913790823'),
    volume = 1,
    total_volume = 51613,
    tick_type = 0,
    chg_type = 2,
    price_chg = Decimal('41'),
    pct_chg = Decimal('0.231481'),
    simtrade = 0
)
```

Response Code: 200 | Event Code: 16 | Info: L/*/TXFG1 | Event: Subscribe or Unsubscribe ok

```
L/TFE/TXFG1
{
    'Amount': [17754.0],
    'AmountSum': [913027415.0],
    'AvgPrice': [17704.623134],
    'Close': [17754.0],
    'Code': 'TXFG1',
    'Date': '2021/07/01',
    'DiffPrice': [42.0],
    'DiffRate': [0.237127],
    'DiffType': [2],
    'High': [17774.0],
    'Low': [17655.0],
    'Open': 17678.0,
    'TargetKindPrice': 17849.57,
    'TickType': [2],
    'Time': '10:42:25.552000',
    'TradeAskVolSum': 33198,
    'TradeBidVolSum': 32180,
    'VolSum': [51570],
    'Volume': [1]
}
```

Attributes**Tick****QuoteVersion.v1 QuoteVersion.v0**

```

code (str): 商品代碼
datetime (datetime.datetime): 日期
open (Decimal): 開盤價
underlying_price (Decimal): 標的物價格
bid_side_total_vol(int): 買盤成交總量 (lot)
ask_side_total_vol(int): 賣盤成交總量 (lot)
avg_price (Decimal): 均價
close (Decimal): 成交價
high (Decimal): 最高價(自開盤)
low (Decimal): 最低價(自開盤)
amount (Decimal): 成交額 (NTD)
total_amount (Decimal): 總成交額 (NTD)
volume (int): 成交量 (lot)
total_volume (int): 總成交量 (lot)
tick_type (int): 內外盤別{1: 外盤, 2: 內盤, 0: 無法判定}
chg_type (int): 漲跌註記{1: 漲停, 2: 漲, 3: 平盤, 4: 跌, 5: 跌停}
price_chg (Decimal): 漲跌
pct_chg (Decimal): 漲跌幅 (%)
simtrade (int): 試標

Amount (list of float): 成交額 (成交價)
AmountSum (list of float): 總成交額 (總成交量)
AvgPrice (list of float): 均價
Close (list of float): 成交價
Code (str): 商品代碼
Date (str): 日期 (yyyy/MM/dd)
DiffPrice (list of float): 漲跌
DiffRate (list of float): 漲跌幅 (%)
DiffType (list of int): 漲跌註記{1: 漲停, 2: 漲, 3: 平盤, 4: 跌, 5: 跌停}
High (list of float): 最高價(自開盤)
Low (list of float): 最低價(自開盤)
Open (float): 開盤價
TargetKindPrice float: 標的物價格
TickType (:List:int): 內外盤別{1: 外盤, 2: 內盤, 0: 無法判定}
Time (str): 時間 (HH:mm:ss.fffff)
TradeAskVolSum (int): 賣盤成交總量 (lot)
TradeBidVolSum (int): 買盤成交總量 (lot)
VolSum (list of int): 總成交量 (lot)
Volume (list of int): 成交量 (lot)

```

BIDASK**Example**

```

api.quote.subscribe(
    api.Contracts.Futures.TXF['TF202107'],
    quote_type = sj.constant.QuoteType.BidAsk,
    version = sj.constant.QuoteVersion.v1
)

```



QuoteVersion.v1 QuoteVersion.v0

Response Code: 200 | Event Code: 16 | Info: QU0/v1/FOP/*/TFE/TXFG1 | Event: Subscribe or Unsubscribe ok

Exchange.TAIFEX

```
BidAsk(
    code = 'TXFG1',
    datetime = datetime.datetime(2021, 7, 1, 10, 51, 31, 999000),
    bid_total_vol = 66,
    ask_total_vol = 101,
    bid_price = [Decimal('17746'), Decimal('17745'), Decimal('17744'), Decimal('17743'), Decimal('17742')],
    bid_volume = [1, 14, 19, 17, 15],
    diff_bid_vol = [0, 1, 0, 0, 0],
    ask_price = [Decimal('17747'), Decimal('17748'), Decimal('17749'), Decimal('17750'), Decimal('17751')],
    ask_volume = [6, 22, 25, 32, 16],
    diff_ask_vol = [0, 0, 0, 0],
    first_derived_bid_price = Decimal('17743'),
    first_derived_ask_price = Decimal('17751'),
    first_derived_bid_vol = 1,
    first_derived_ask_vol = 1,
    underlying_price = Decimal('17827.94'),
    simtrade = 0
)
```

Response Code: 200 | Event Code: 16 | Info: Q/*/TXFG1 | Event: Subscribe or Unsubscribe ok

Q/TFE/TXFG1

```
{
    'AskPrice': [17747.0, 17748.0, 17749.0, 17750.0, 17751.0],
    'AskVolSum': 99,
    'AskVolume': [6, 22, 25, 31, 15],
    'BidPrice': [17746.0, 17745.0, 17744.0, 17743.0, 17742.0],
    'BidVolSum': 81,
    'BidVolume': [1, 12, 23, 25, 20],
    'Code': 'TXFG1',
    'Date': '2021/07/01',
    'DiffaskVol': [0, 0, 0, 0, 0],
    'DiffAskVolSum': 0,
    'DiffbidVol': [0, 0, 2, 0, 0],
    'DiffBidVolSum': 0,
    'FirstDerivedAskPrice': 17751.0,
    'FirstDerivedAskVolume': 1,
    'FirstDerivedBidPrice': 17743.0,
    'FirstDerivedBidVolume': 1,
    'TargetKindPrice': 17828.46,
    'Time': '10:51:29.999000'
}
```

Attributes**BidAsk****QuoteVersion.v1 QuoteVersion.v0**

```

code (str): 商品代碼
datetime (datetime.datetime): 時間
bid_total_vol (int): 委買量總計 (lot)
ask_total_vol (int): 委賣量總計 (lot)
bid_price (:List:decimal): 委買價
bid_volume (:List:int): 委買量 (lot)
diff_bid_vol (:List:int): 委買價增減量 (lot)
ask_price (:List:decimal): 委賣價
ask_volume (:List:int): 委賣量 (lot)
diff_ask_vol (:List:int): 委賣價增減量 (lot)
first_derived_bid_price (decimal): 衍生一檔委買價
first_derived_ask_price (decimal): 衍生一檔委賣價
first_derived_bid_vol (int): 衍生一檔委買量
first_derived_ask_vol (int): 衍生一檔委賣量
underlying_price (decimal): 標的物價格
simtrade (int): 試撮

AskPrice (:List:float): 委賣價
AskVolSum (int): 委賣量總計(lot)
AskVolume (:List:int): 委賣量
BidPrice (:List:float): 委買價
BidVolSum (int): 委買量總計(lot)
BidVolume (:List:int): 委買量
Code (str): 商品代碼
Date (str): 日期 (yyyy/MM/dd)
DiffAskVol (:List:int): 賣價增減量(lot)
DiffAskVolSum (int):
DiffBidVol (:List:int): 買價增減量(lot)
DiffBidVolSum (int):
FirstDerivedAskPrice (float): 衍生一檔委賣價
FirstDerivedAskVolume (int): 衍生一檔委賣量
FirstDerivedBidPrice (float): 衍生一檔委買價
FirstDerivedBidVolume (int): 衍生一檔委買量
TargetKindPrice (float): 標的物價格
Time (str): 時間 (HH:mm:ss.fffff)

```

QUOTE**Example**

```

api.quote.subscribe(
    api.Contracts.Futures.TXF['TF202512'],
    quote_type = sj.constant.QuoteType.Quote,
    version = sj.constant.QuoteVersion.v1,
)

```

✓ **Quote**

```
Response Code: 200 | Event Code: 16 | Info: QU0/v2/FOP/*/TFE/TXFL5 | Event: Subscribe or Unsubscribe ok

Exchange.TAIFEX
Quote(
    code = 'TXFL5',
    datetime = datetime.datetime(2025, 12, 11, 29, 46, 247000),
    open = Decimal('28266'),
    avg_price = Decimal('28284.374007'),
    close = Decimal('28220'),
    high = Decimal('28342'),
    low = Decimal('28192'),
    amount = Decimal('28220'),
    total_amount = Decimal('946904273'),
    volume = 0,
    total_volume = 33478,
    tick_type = 1,
    chg_type = 2,
    price_chg = Decimal('119'),
    pct_chg = Decimal('0.423472'),
    bid_side_total_vol = 15254,
    ask_side_total_vol = 17295,
    bid_side_total_cnt = 22855,
    ask_side_total_cnt = 21882,
    bid_price = [Decimal('28218'), Decimal('28217'), Decimal('28216'), Decimal('28215'), Decimal('28214')],
    bid_volume = [5, 6, 11, 6, 10],
    diff_bid_vol = [0, 0, 0, 0, 0],
    ask_price = [Decimal('28220'), Decimal('28221'), Decimal('28222'), Decimal('28223'), Decimal('28224')],
    ask_volume = [2, 8, 14, 9, 9],
    diff_ask_vol = [0, -1, 1, 0, 0],
    first_derived_bid_price = Decimal('28215'),
    first_derived_ask_price = Decimal('28222'),
    first_derived_bid_vol = 1,
    first_derived_ask_vol = 1,
    underlying_price = Decimal('28185.62'),
    simtrade = False
)
```

Attributes

_quote

```
code (str): 商品代碼
datetime (datetime.datetime): 時間
open (Decimal): 開盤價
avg_price (Decimal): 均價
close (Decimal): 成交價
high (Decimal): 最高價(自開盤)
low (Decimal): 最低價(自開盤)
amount (Decimal): 成交額 (NTD)
total_amount (Decimal): 總成交額 (NTD)
volume (int): 成交量 (lot)
total_volume (int): 總成交量 (lot)
tick_type (int): 內外盤別(1: 外盤, 2: 內盤, 0: 無法判定)
chg_type (int): 漲跌註記(1: 漲停, 2: 漲, 3: 平盤, 4: 跌, 5: 跌停)
price_chg (Decimal): 漲跌
pct_chg (Decimal): 漲跌幅 (%)
bid_side_total_vol (int): 買盤成交總量 (lot)
ask_side_total_vol (int): 賣盤成交總量 (lot)
bid_side_total_cnt (int): 買盤成交筆數
ask_side_total_cnt (int): 賣盤成交筆數
bid_price (:List:Decimal): 委買價
bid_volume (:List:int): 委買量 (lot)
diff_bid_vol (:List:int): 委買價增減量 (lot)
ask_price (:List:Decimal): 委賣價
ask_volume (:List:int): 委賣量 (lot)
diff_ask_vol (:List:int): 委賣價增減量 (lot)
first_derived_bid_price (Decimal): 衍生一檔委買價
first_derived_ask_price (Decimal): 衍生一檔委賣價
first_derived_bid_vol (int): 衍生一檔委買量
first_derived_ask_vol (int): 衍生一檔委賣量
underlying_price (Decimal): 標的物價格
simtrade (bool): 試播
```

CALLBACK

In default, we set quote callback as print function. You can modify callback function as you wish. Just remember, always avoid making calulations inside the callback function.

Tick**i: pythonic way by using decorator****QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import TickFOPv1, Exchange

@api.on_tick_fop_v1()
def quote_callback(exchange:Exchange, tick:TickFOPv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

i: traditional way**QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import TickFOPv1, Exchange

def quote_callback(exchange:Exchange, tick:TickFOPv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

api.quote.set_on_tick_fop_v1_callback(quote_callback)

def quote_callback(topic: str, tick: dict):
    print(f"Topic: {topic}, Tick: {tick}")

api.quote.set_quote_callback(quote_callback)
```

Out**QuoteVersion.v1 QuoteVersion.v0**

```
Exchange: Exchange.TAIFEX, Tick: Tick(code='TXFG1', datetime=datetime.datetime(2021, 7, 2, 13, 17, 22, 784000), open=Decimal('17651'), underlying_price=Decimal('17727.12'), trade_bid_total_vol=61550, trade_ask_volume=60914, avg_price=Decimal('17657.959752'), close=Decimal('17653'), high=Decimal('17724'), low=Decimal('17588'), amount=Decimal('35306'), total_amount=Decimal('1683421593'), volume=2, total_volume=95335, tick_type=1, chg_type=2, price_chg=Decimal('7'), pct_chg=Decimal('0.039669'), simtrade=0)
```

```
Topic: L/TFE/TXFG1, Quote: {'Amount': [17654.0], 'AmountSum': [1682856730.0], 'AvgPrice': [17657.961764], 'Close': [17654.0], 'Code': 'TXFG1', 'Date': '2021/07/02', 'DiffPrice': [8.0], 'DiffRate': [0.045336], 'DiffType': [2], 'High': [17724.0], 'Low': [17588.0], 'Open': 17651.0, 'TargetKindPrice': 17725.14, 'TickType': [1], 'Time': '13:17:16.533000', 'TradeAskVolSum': 60890, 'TradeBidVolSum': 61520, 'VolSum': [95303], 'Volume': [1]}
```

BidAsk**i: pythonic way by using decorator****QuoteVersion.v1 QuoteVersion.v0**

```
from shioaji import BidAskFOPv1, Exchange

@api.on_bidask_fop_v1()
def quote_callback(exchange:Exchange, bidask:BidAskFOPv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

iii: traditional way

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import BidAskFOPv1, Exchange

def quote_callback(exchange:Exchange, bidask:BidAskFOPv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

api.quote.set_on_bidask_fop_v1_callback(quote_callback)

def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")

api.quote.set_quote_callback(quote_callback)
```

Cut

QuoteVersion.v1 QuoteVersion.v0

```
Exchange: Exchange.TAIFEX, BidAsk: BidAsk(code='TXFG1', datetime=datetime.datetime(2021, 7, 2, 13, 18, 0, 684000), bid_total_vol=69, ask_total_vol=94, bid_price=[Decimal('17651'), Decimal('17650'), Decimal('17649'), Decimal('17648'), Decimal('17647')], bid_volume=[10, 12, 18, 18, 11], diff_bid_vol=[0, 0, 0, 0, 0], ask_price=[Decimal('17653'), Decimal('17654'), Decimal('17655'), Decimal('17656'), Decimal('17657')], ask_volume=[6, 17, 29, 22, 20], diff_ask_vol=[0, 0, 0, 0, 0], first_derived_bid_price=Decimal('17647'), first_derived_ask_price=Decimal('17657'), first_derived_bid_vol=2, first_derived_ask_vol=3, underlying_price=Decimal('17725.5'), simtrade=0)
```

```
Topic: Q/TFE/TXFG1, Quote: {'AskPrice': [17653.0, 17654.0, 17655.0, 17656.0, 17657.0], 'AskVolSum': 85, 'AskVolume': [3, 16, 24, 22, 20], 'BidPrice': [17651.0, 17650.0, 17649.0, 17648.0, 17647.0], 'BidVolSum': 67, 'BidVolume': [10, 10, 18, 18, 11], 'Code': 'TXFG1', 'Date': '2021/07/02', 'DiffAskVol': [-4, -2, 0, 0, 0], 'DiffAskVolSum': 0, 'DiffBidVol': [1, 0, 2, 0, 0], 'DiffBidVolSum': 0, 'FirstDerivedAskPrice': 17657.0, 'FirstDerivedAskVolume': 3, 'FirstDerivedBidPrice': 17647.0, 'FirstDerivedBidVolume': 2, 'TargetKindPrice': 17716.19, 'Time': '13:17:57.809000'}
```

Quote

iii: pythonic way by using decorator

```
from shioaji import QuoteFOPv1, Exchange

@api.on_quote_fop_v1()
def quote_callback(exchange:Exchange, quote:QuoteFOPv1):
    print(f"Exchange: {exchange}, Quote: {quote}")
```

Cut

```
from shioaji import QuoteFOPv1, Exchange

def quote_callback(exchange:Exchange, quote:QuoteFOPv1):
    print(f"Exchange: {exchange}, Quote: {quote}")

api.quote.set_on_quote_fop_v1_callback(quote_callback)
```

```
Exchange: Exchange.TAIFEX, Quote: Quote(code='TXFL5', datetime=datetime.datetime(2025, 12, 12, 11, 29, 46, 247000), open=Decimal('28266'), avg_price=Decimal('28284.374007'), close=Decimal('28220'), high=Decimal('28342'), low=Decimal('28192'), amount=Decimal('28220'), total_amount=Decimal('946904273'), volume=0, total_volume=33478, tick_type=1, chg_type=2, price_chg=Decimal('119'), pct_chg=Decimal('0.423472'), bid_side_total_vol=15254, ask_side_total_vol=17295, bid_side_total_cnt=22855, ask_side_total_cnt=21882, bid_price=[Decimal('28218'), Decimal('28217'), Decimal('28216'), Decimal('28215'), Decimal('28214')], bid_volume=[5, 6, 11, 6, 10], diff_bid_vol=[0, 0, 0, 0, 0], ask_price=[Decimal('28220'), Decimal('28221'), Decimal('28222'), Decimal('28223'), Decimal('28224')], ask_volume=[2, 8, 14, 9, 9], diff_ask_vol=[0, -1, 1, 0, 0], first_derived_bid_price=Decimal('28215'), first_derived_ask_price=Decimal('28222'), first_derived_bid_vol=1, first_derived_ask_vol=1, underlying_price=Decimal('28185.62'), simtrade=False)
```

- Advanced quote callback settings please refer to [Quote-Binding Mode](#).

4.4.2 Historical Market Data

Ticks

Ticks can get all day, period of time or last counts of the day. The default is get ticks of last trade day .

Ticks

```
api.ticks?

Signature:
api.ticks(
    contract: shioaji.contracts.BaseContract,
    date: str = '2022-12-26',
    query_type: shioaji.constant.TicksQueryType = <TicksQueryType.AllDay: 'AllDay'>,
    time_start: Union[str, datetime.time] = None,
    time_end: Union[str, datetime.time] = None,
    last_cnt: int = 0,
    timeout: int = 30000,
    cb: Callable[[shioaji.data.Ticks], NoneType] = None,
) -> shioaji.data.Ticks
Docstring:
get contract tick volumn
```

BY DATE

Info

```
ticks = api.ticks(
    contract=api.Contracts.Stocks["2330"],
    date="2023-01-16"
)
ticks
```

Out

```
Ticks(
    ts=[1673859600113699000, 1673859600228800000, 1673859600244294000, 1673859600308595000],
    close=[506.0, 505.0, 506.0, 506.0],
    volume=[3340, 1, 17, 2],
    bid_price=[505.0, 505.0, 506.0, 506.0],
    bid_volume=[122, 320, 60, 58],
    ask_price=[506.0, 506.0, 507.0, 507.0],
    ask_volume=[13, 22, 702, 702],
    tick_type=[1, 2, 1, 2]
)
```

Ticks

```
ts (int): timestamp
close (float): close
volume (int): volume
bid_price (float): bid price
bid_volume (int): bid volume
ask_price (float): ask price
ask_volume (int): ask volume
tick_type (int): 外內盤別(1: 外盤, 2: 內盤, 0: 無法判定)
```

To DataFrame

Info

```
import pandas as pd
df = pd.DataFrame(**ticks)
df.ts = pd.to_datetime(df.ts)
df.head()
```

 Out

ts	ask_price	close	bid_volume	volume	ask_volume	tick_type	bid_price
2023-01-16 09:00:00.113699	506	506	122	3340	13	1	505
2023-01-16 09:00:00.228800	506	505	320	1	22	2	505
2023-01-16 09:00:00.244294	507	506	60	17	702	1	506
2023-01-16 09:00:00.308595	507	506	58	2	702	2	506

RANGE TIME

 In

```
ticks = api.ticks(
    contract=api.Contracts.Stocks["2330"],
    date="2023-01-16",
    query_type=sj.constant.TicksQueryType.RangeTime,
    time_start="09:00:00",
    time_end="09:20:01"
)
ticks
```

 Out

```
Ticks(
    ts=[1673859600113699000, 1673859600228800000, 1673859600244294000, 1673859600308595000],
    close=[506.0, 505.0, 506.0, 506.0],
    volume=[3340, 1, 17, 2],
    bid_price=[505.0, 505.0, 506.0, 506.0],
    bid_volume=[122, 320, 60, 58],
    ask_price=[506.0, 506.0, 507.0, 507.0],
    ask_volume=[13, 22, 702, 702],
    tick_type=[1, 2, 1, 2]
)
```

LAST COUNT

 In

```
ticks = api.ticks(
    contract=api.Contracts.Stocks["2330"],
    date="2023-01-16",
    query_type=sj.constant.TicksQueryType.LastCount,
    last_cnt=4,
)
ticks
```

 Out

```
Ticks(
    ts=[1673859600113699000, 1673859600228800000, 1673859600244294000, 1673859600308595000],
    close=[506.0, 505.0, 506.0, 506.0],
    volume=[3340, 1, 17, 2],
    bid_price=[505.0, 505.0, 506.0, 506.0],
    bid_volume=[122, 320, 60, 58],
    ask_price=[506.0, 506.0, 507.0, 507.0],
    ask_volume=[13, 22, 702, 702],
    tick_type=[1, 2, 1, 2]
)
```

KBar

```
api.kbars?

Signature:
api.kbars(
    contract: shioaji.contracts.BaseContract,
    start: str = '2023-01-15',
    end: str = '2023-01-16',
    timeout: int = 30000,
    cb: Callable[[shioaji.data.Kbars], NoneType] = None,
) -> shioaji.data.Kbars
Docstring:
get Kbar
```



```
kbars = api.kbars(
    contract=api.Contracts.Stocks["2330"],
    start="2023-01-15",
    end="2023-01-16",
)
kbars
```



```
Kbars(
    ts=[1673859660000000000, 1673859720000000000, 1673859780000000000, 1673859840000000000],
    Open=[506.0, 505.0, 505.0, 504.0],
    High=[508.0, 506.0, 506.0, 505.0],
    Low=[505.0, 505.0, 504.0, 504.0],
    Close=[505.0, 505.0, 504.0, 504.0],
    Volume=[5308, 1018, 543, 209]
)
```



```
ts (int): timestamp
Open (float): open price
High (float): the highest price
Low: (float): the lowest price
Close (float): close price
Volume (int): volume
```

To DataFrame



```
import pandas as pd
df = pd.DataFrame(**kbars)
df.ts = pd.to_datetime(df.ts)
df.head()
```

Out

Close	Amount	Low	Volume	ts	Open	High
505	2.68731e+09	505	5308	2023-01-16 09:01:00	506	508
505	5.14132e+08	505	1018	2023-01-16 09:02:00	505	506
504	2.74112e+08	504	543	2023-01-16 09:03:00	505	506
504	1.0542e+08	504	209	2023-01-16 09:04:00	504	505

Historical Periods**Historical Periods**

	Start Date	End Date
Index	2020-03-02	Today
Stock	2020-03-02	Today
Futures	2020-03-22	Today

Continuous Futures

Once a futures contract passes its expiration date, the contract is invalid, and it will not exist in your `api.Contracts`. In order to get historical data for expired futures contract, we provide continuous futures contracts. `R1`, `R2` are continuous near-month and next-to-near-month futures contracts respectively. They will automatically roll contracts on delivery date. You can get historical data by using `R1`, `R2` contracts, ex `api.Contracts.Futures.TXF.TXFR1`. We show examples below.

Ticks

Ticks

```
ticks = api.ticks(
    contract=api.Contracts.Futures.TXF.TXFR1,
    date="2020-03-22"
)
ticks
```

Out

```
Ticks(
    ts=[161616600003000000, 161616600014000000, 161616600014000000, 161616600019000000],
    close=[16011.0, 16013.0, 16014.0, 16011.0],
    volume=[49, 2, 2, 1],
    bid_price=[0.0, 16011.0, 16011.0, 16011.0],
    bid_volume=[0, 1, 1, 1],
    ask_price=[0.0, 16013.0, 16013.0, 16013.0],
    ask_volume=[0, 1, 1, 1]
    tick_type=[1, 1, 1, 2]
)
```

Kbars

Kbars

```
kbars = api.kbars(  
    contract=api.Contracts.Futures.TXF.TXFR1,  
    start="2023-01-15",  
    end="2023-01-16",  
)  
kbars
```

Out

```
Kbars(  
    ts=[16164027600000000000, 16164028200000000000, 16164028800000000000, 16164029400000000000],  
    Open=[16018.0, 16018.0, 16000.0, 15992.0],  
    High=[16022.0, 16020.0, 16005.0, 15999.0],  
    Low=[16004.0, 16000.0, 15975.0, 15989.0],  
    Close=[16019.0, 16002.0, 15992.0, 15994.0],  
    Volume=[1791, 864, 1183, 342]  
)
```

4.4.3 Snapshot

Snapshot is a present stock, future, option info. It contain open, high, low, close, change price, average price, volume, total volume, buy price, buy volume, sell price, sell volume and yesterday volume.

Reminder

Each snapshot can contain up to 500 contracts.

Snapshots

```
>> api.snapshots?

Signature:
api.snapshots(
    contracts: List[Union[shioaji.contracts.Option, shioaji.contracts.Future, shioaji.contracts.Stock, shioaji.contracts.Index]],
    timeout: int = 30000,
    cb: Callable[[shioaji.data.Snapshot], NoneType] = None,
) -> List[shioaji.data.Snapshot]
Docstring:
get contract snapshot info
```

Example

Info

```
contracts = [
    api.Contracts.Stocks['2330'],
    api.Contracts.Stocks['2317']
]
snapshots = api.snapshots(contracts)
snapshots
```

Out

```
[
  Snapshot(
    ts=1673620200000000000,
    code='2330',
    exchange='TSE',
    open=507.0,
    high=509.0,
    low=499.0,
    close=500.0,
    tick_type=TickType.Sell: 'Sell',
    change_price=13.5,
    change_rate=2.77,
    change_type=ChangeType.Up: 'Up',
    average_price=502.42,
    volume=48,
    total_volume=77606,
    amount=24000000,
    total_amount=38990557755,
    yesterday_volume=20963.0,
    buy_price=500.0,
    buy_volume=122.0,
    sell_price=501.0,
    sell_volume=1067,
    volume_ratio=3.7
  ),
  Snapshot(
    ts=1673620200000000000,
    code='2317',
    exchange='TSE',
    open=99.0,
    high=99.5,
    low=98.6,
    close=98.6,
    tick_type=TickType.Sell: 'Sell',
    change_price=0.0,
    change_rate=0.0,
    change_type=ChangeType.Unchanged: 'Unchanged',
    average_price=98.96,
    volume=63,
    total_volume=17809,
    amount=6211800,
    total_amount=1762344817,
    yesterday_volume=18537.0,
    buy_price=98.6,
    buy_volume=607.0,
    sell_price=98.7,
    sell_volume=4,
    volume_ratio=0.96
  )
]
```

To DataFrame

```
df = pd.DataFrame(s.__dict__ for s in snapshots)
df.ts = pd.to_datetime(df.ts)
df
```

Out

ts	code	exchange	open	high	low	close	tick_type	change_price
2023-01-13 14:30:00	2330	TSE	507	509	499	500	Sell	13.5
2023-01-13 14:30:00	2317	TSE	99	99.5	98.6	98.6	Sell	0

Attributes

Snapshot

```
ts (int): TimeStamp
code (str): Contract id
exchange (Exchange): exchange
open (float): open
high (float): high
low (float): low
close (float): close
tick_type (TickType): Close is buy or sell price
    {None, Buy, Sell}
change_price (float): change price
change_rate (float): change rate
change_type (ChangeType):
    {LimitUp, Up, Unchanged, Down, LimitDown}
average_price (float): average of price
volume (int): volume
total_volume (int): total volume
amount (int): Deal amount
total_amount (int): Total deal amount
yesterday_volume (float): Volume of yesterday
buy_price (float): Price of buy
buy_volume (float): Volume of sell
sell_price (float): Price of sell
sell_volume (int): Volume of sell
volume_ratio (float): total_volume/yesterday_volume
```

4.4.4 Short Stock Source

Short Stock Sources

```
>> api.short_stock_sources?

Signature:
api.short_stock_sources(
    contracts: List[shioaji.contracts.Stock],
    timeout: int = 5000,
    cb: Callable[[shioaji.data.ShortStockSource], NoneType] = None,
) -> List[shioaji.data.ShortStockSource]
```

Example



```
contracts = [
    api.Contracts.Stocks['2330'],
    api.Contracts.Stocks['2317']
]
short_stock_sources = api.short_stock_sources(contracts)
short_stock_sources
```



```
[  
    ShortStockSource(code='2330', short_stock_source=58260, ts=1673943433000000000),  
    ShortStockSource(code='2317', short_stock_source=75049, ts=1673943433000000000)  
]
```

To DataFrame



```
df = pd.DataFrame(s.__dict__ for s in short_stock_sources)
df.ts = pd.to_datetime(df.ts)
df
```



code	short_stock_source	ts
2330	58260	2023-01-17 08:17:13
2317	75049	2023-01-17 08:17:13

Attributes

ShortStockSource

```
code (str): contract id
short_stock_source (float): short stock source
ts (int): timeStamp
```

4.4.5 Credit Enquiries

Credit Enquiries

```
>> api.credit_enquiries?

Signature:
api.credit_enquiries(
    contracts: List[shioaji.contracts.Stock],
    timeout: int = 30000,
    cb: Callable[[shioaji.data.CreditEnquiry], NoneType] = None,
) -> List[shioaji.data.CreditEnquiry]
```

Example



```
contracts = [
    api.Contracts.Stocks['2330'],
    api.Contracts.Stocks['2890']
]
credit_enquires = api.credit_enquiries(contracts)
credit_enquires
```



```
[ CreditEnquire(update_time='2020-12-11 13:30:13', system='HE', stock_id='2330', margin_unit=1381),
CreditEnquire(update_time='2020-12-11 13:30:02', system='HC', stock_id='2330', margin_unit=1371),
CreditEnquire(update_time='2020-12-11 13:30:05', system='HN', stock_id='2330', margin_unit=1357),
CreditEnquire(update_time='2020-12-11 13:30:03', system='HF', stock_id='2330', margin_unit=1314),
CreditEnquire(update_time='2020-12-09 10:56:05', system='HE', stock_id='2890'),
CreditEnquire(update_time='2020-12-11 09:33:04', system='HN', stock_id='2890'),
CreditEnquire(update_time='2020-12-02 09:01:03', system='HF', stock_id='2890')
]
```

To DataFrame



```
df = pd.DataFrame(c.__dict__ for c in credit_enquires)
df.update_time = pd.to_datetime(df.update_time)
df
```

 **ut**

	margin_unit	short_unit	stock_id	system	update_time
0	1381	0	2330	HE	2020-12-11 13:30:13
1	1371	0	2330	HC	2020-12-11 13:30:02
2	1357	0	2330	HN	2020-12-11 14:31:19
3	1314	0	2330	HF	2020-12-11 14:31:19
4	0	0	2890	HE	2020-12-09 10:56:05
5	0	0	2890	HN	2020-12-11 09:33:04
6	0	0	2890	HF	2020-12-02 09:01:03

Attributes **CreditEnquiry**

```

update_time (str): update time
system (str): system
stock_id (str): stock id
margin_unit (int): margin unit
short_unit (int): short unit

```

4.4.6 Scanners

Scanners can use parameter of scannertype to get the rank of ChangePercent, ChangePrice, DayRange, Volume and Amount.

Scanners

```
>> api.scanners?

Signature:
api.scanners(
    scanner_type: shioaji.constant.ScannerType,
    ascending: bool = True,
    date: str = None,
    count: shioaji.shioaji.ConstrainedIntValue = 100, # 0 <= count <= 200
    timeout: int = 30000,
    cb: Callable[[List[shioaji.data.ChangePercentRank]], NoneType] = None,
) -> List[shioaji.data.ChangePercentRank]
```

Ascending is sorted from largest to smallest by default, and the value of ascending is True. Set ascending to False to sort in descending order, and the other ranking methods are the same. `count` is how many ranks you get.

Supported Scanner Type

```
ChangePercentRank: rank by price percentage change
ChangePriceRank: rank by price change
DayRangeRank: rank by day range
VolumeRank: rank by volume
AmountRank: rank by amount
```

Example

ChangePercentRank

```
scanners = api.scanners(
    scanner_type=sj.constant.ScannerType.ChangePercentRank,
    count=1
)
scanners
```

 Cut

```
[
    ChangePercentRank(
        date='2021-04-09',
        code='5211',
        name='蒙恬',
        ts=1617978600000000000,
        open=16.4,
        high=17.6,
        low=16.35,
        close=17.6,
        price_range=1.25,
        tick_type=1,
        change_price=1.6,
        change_type=1,
        average_price=17.45,
        volume=7,
        total_volume=1742,
        amount=123200,
        total_amount=30397496,
        yesterday_volume=514,
        volume_ratio=3.39,
        buy_price=17.6,
        buy_volume=723,
        sell_price=0.0,
        sell_volume=0,
        bid_orders=237,
        bid_volumes=82,
        ask_orders=33,
        ask_volumes=64
    )
]
```

To DataFrame



```
scanners = api.scanners(
    scanner_type=sj.constant.ScannerType.ChangePercentRank,
    count=5
)
df = pd.DataFrame(s._dict_ for s in scanners)
df.ts = pd.to_datetime(df.ts)
df
```

 Cut

date	code	name	ts	open	high	low	close	price_range
2023-01-17	6259	百徽	2023-01-17 11:11:41.030000	22.8	23.75	22.45	23.75	1.3
2023-01-17	6788	華景電	2023-01-17 11:19:01.924000	107	116	107	116	9
2023-01-17	2540	愛山林	2023-01-17 11:17:39.435000	85.2	85.2	83	85.2	2.2
2023-01-17	8478	東哥遊艇	2023-01-17 11:18:33.702000	350.5	378	347	378	31
2023-01-17	6612	奈米醫材	2023-01-17 11:15:32.752000	102	109	102	109	7

Attributes

ChangePercentRank

```
date (str): date
code (str): code
name (str): name
ts (int): timestamp
open (float): open price
high (float): high price
low (float): low price
close (float): close price
price_range (float): range of price
tick_type (int): 內外盤別{1: 外盤, 2: 內盤, 0: 無法判定}
change_price (float): change in price
change_type (int):
    {LimitUp, Up, Unchanged, Down, LimitDown}
average_price (float): average price
volume (int): volume
total_volume (int): total volume since market open
amount (int): amount
total_amount (int): total amount since market open
yesterday_volume (int): yesterday volume
volume_ratio (float): total_volume/yesterday_volume
buy_price (float): bid
buy_volume (int): bid volume
sell_price (float): ask
sell_volume (int): ask volume
bid_orders (int): number of orders that deal at bid
bid_volumes (int): total volume that deal at bid
ask_orders (int): number of orders that deal at ask
ask_volumes (int): total volume that deal at ask
```

4.4.7 Disposition and Attention Stocks

Disposition Stocks

Disposition Stocks

```
>> api.punish?

Signature:
api.punish(
    timeout: int = 5000,
    cb: Callable[[shioaji.data.Punish], NoneType] = None,
) -> shioaji.data.Punish
Docstring: get punish information
```

EXAMPLE



```
punish = api.punish()
punish
```



```
Punish(
    code=['2349', '2408', ...],
    start_date=[datetime.date(2025, 12, 17), datetime.date(2025, 12, 8), ...],
    end_date=[datetime.date(2025, 12, 31), datetime.date(2025, 12, 19), ...],
    updated_at=[datetime.datetime(2025, 12, 16, 18, 18, 20), datetime.datetime(2025, 12, 16, 18, 18, 20), ...],
    interval=['5分鐘', '5分鐘', ...],
    unit_limit=[10.0, 10.0, ...],
    total_limit=[30.0, 30.0, ...],
    description=['...', '...', ...],
    announced_date=[datetime.date(2025, 12, 16), datetime.date(2025, 12, 5), ...]
)
```

To DataFrame



```
df = pd.DataFrame(punish.__dict__)
df
```



code	start_date	end_date	updated_at	interval	unit_limit	total_limit	description
2349	2025-12-17	2025-12-31	2025-12-16 18:18:20	5分鐘	10.0	30.0	...
2408	2025-12-08	2025-12-19	2025-12-16 18:18:20	5分鐘	10.0	30.0	...

ATTRIBUTES **Punish**

```
code (list[str]): code
start_date (list[date]): disposition start date
end_date (list[date]): disposition end date
updated_at (list[datetime]): updated time
interval (list[str]): matching interval
unit_limit (list[float]): single order limit
total_limit (list[float]): daily order limit
description (list[str]): disposition description
announced_date (list[date]): announced date
```

Attention Stocks **Attention Stocks**

```
>> api.notice?

Signature:
api.notice(
    timeout: int = 5000,
    cb: Callable[[shioaji.data.Notice], NoneType] = None,
) -> shioaji.data.Notice
Docstring: get notice information
```

EXAMPLE

```
notice = api.notice()
notice
```

 **Cut**

```
Notice(
    code=['089508', '2349', ...],
    updated_at=[datetime.datetime(2025, 12, 16, 18, 18, 19), datetime.datetime(2025, 12, 16, 18, 18, 19), ...],
    close=[9.85, 16.2, ...],
    reason=['最近六個營業日累積收盤價漲幅達39.34% (第一款)。', '最近六個營業日累積收盤價漲幅達41.59% (第一款) ...'],
    announced_date=[datetime.date(2025, 12, 16), datetime.date(2025, 12, 16), ...]
)
```

To DataFrame



```
df = pd.DataFrame(notice.__dict__)
df
```

 **Cut**

code	updated_at	close	reason	announced_date
089508	2025-12-16 18:18:19	9.85	最近六個營業日累積收盤價漲幅達 39.34% (第一款)。	2025-12-16
2349	2025-12-16 18:18:19	16.2	最近六個營業日累積收盤價漲幅達 41.59% (第一款) ...	2025-12-16

ATTRIBUTES**Notice**

```
code (list[str]): code  
updated_at (list[datetime]): updated time  
close (list[float]): close price  
reason (list[str]): attention reason  
announced_date (list[date]): announced date
```

4.5 Order

4.5.1 Stock

Reminder

First, you need to [login](#) and [activate CA](#).

STOCK ORDER

Order Attributes

version>=1.0 version<1.0

```
price (float or int): the price of order
quantity (int): the quantity of order
action (str): order action to buy or sell
    {Buy, Sell}
price_type (str): pricing type of order
    {LMT, MKT, MKP} (限價、市價、範圍市價)
order_type (str): the type of order
    {ROD, IOC, FOK}
order_cond (str): order condition stock only
    {Cash, MarginTrading, ShortSelling} (現股、融資、融券)
order_lot (str): the type of order
    {Common, Fixing, Odd, IntradayOdd} (整股、定盤、盤後零股、盤中零股)
daytrade_short (bool): the type of first sell
    {True, False}
custom_field (str): memo field, only letters and numbers are allowed, and the maximum length is 6.
account (:obj:Account): which account to place this order
ca (binary): the ca of this order

price (float or int): the price of order
quantity (int): the quantity of order
action (str): order action to buy or sell
    {Buy, Sell}
price_type (str): pricing type of order
    {LMT, MKT, MKP} (限價、市價、範圍市價)
order_type (str): the type of order
    {ROD, IOC, FOK}
order_cond (str): order condition stock only
    {Cash, MarginTrading, ShortSelling} (現股、融資、融券)
order_lot (str): the type of order
    {Common, Fixing, Odd, IntradayOdd} (整股、定盤、盤後零股、盤中零股)
first_sell (str): the type of first sell
    {true, false}
custom_field (str): memo field, only letters and numbers are allowed, and the maximum length is 6.
account (:obj:Account): which account to place this order
ca (binary): the ca of this order
```

PLACE ORDER

Product information (`contract`) and order information (`order`) must be provided when placing an order.

Place Order

```
api.place_order?

Signature:
    api.place_order(
        contract: shioaji.contracts.Contract,
        order: shioaji.order.Order,
        timeout: int = 5000,
        cb: Callable[[shioaji.order.Trade], NoneType] = None,
    ) -> shioaji.order.Trade

Docstring:
    placing order
```

Contract

```
contract = api.Contracts.Stocks.TSE.TSE2890
```

Order

version>=1.0 version<1.0

```
order = api.Order(
    price=17,
    quantity=3,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    # daytrade_short=False,
    custom_field="test",
    account=api.stock_account
)

order = api.Order(
    price=17,
    quantity=3,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.TFTStockPriceType.LMT,
    order_type=sj.constant.TFTOrderType.ROD,
    order_lot=sj.constant.TFTStockOrderLot.Common,
    # first_sell=sj.constant.StockFirstSell.No,
    custom_field="test",
    account=api.stock_account
)
```

Place Order

```
trade = api.place_order(contract, order)
trade
```

Cut

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27af',
        seqno='000002',
        ordno='000001',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='A123456789',
            broker_id='9495',
            account_id='1234567',
            signed=True
        ),
        custom_field='test',
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        daytrade_short=False
    ),
    status=OrderStatus(
        id='531e27af',
        status=<Status.PendingSubmit: 'PendingSubmit'>,
        status_code='00',
        order_datetime=datetime.datetime(2023, 1, 12, 11, 18, 3, 867490),
        deals=[]
    )
)
```

After `place_order`, you will also receive the information sent back from the exchange. For details, please refer to [Order & Deal Event](#).

To update the `trade` status, you need to call `update_status`.

Update Status

```
api.update_status(api.stock_account)
trade
```

Get

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=DayTrade.Yes: 'Yes'
),
order=Order(
    action=<Action.Buy: 'Buy'>,
    price=17,
    quantity=3,
    id='531e27af',
    seqno='000002',
    ordno='000001',
    account=Account(
        account_type=<AccountType.Stock: 'S'>,
        person_id='A123456789',
        broker_id='9495',
        account_id='1234567',
        signed=True
),
custom_field='test',
price_type=<StockPriceType.LMT: 'LMT'>,
order_type=<OrderType.ROD: 'ROD'>,
daytrade_short=False
),
status=OrderStatus(
    id='531e27af',
    status=Status.Submitted: 'Submitted',
    status_code='00',
    order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
    order_quantity=3,
    deals=[]
)
)
```

Status of Trade

- PendingSubmit : Sending
- PreSubmitted : Reservation
- Submitted : Send Successfully
- Failed : Failed
- Cancelled : Cancelled
- Filled : Complete Fill
- PartFilled : Part Fill

UPDATE ORDER**Update Order**

```
api.update_order?

Signature:
api.update_order(
    trade: shioaji.order.Trade,
    price: Union[pydantic.types.StrictInt, float] = None,
    qty: int = None,
    timeout: int = 5000,
    cb: Callable[[shioaji.order.Trade], NoneType] = None,
) -> shioaji.order.Trade
Docstring: update the order price or qty
```

Update Price**Update Price**

```
api.update_order(trade=trade, price=17.5)
api.update_status(api.stock_account)
trade
```

Out

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=DayTrade.Yes: 'Yes'
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27af',
        seqno='000002',
        ordno='000001',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='A123456789',
            broker_id='9A95',
            account_id='1234567',
            signed=True
        ),
        custom_field='test',
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        daytrade_short=False
    ),
    status=OrderStatus(
        id='531e27af',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        modified_price=17.5,
        order_quantity=3,
        deals=[]
    )
)
```

Update Quantity (Reduce)

`update_order` can only reduce the quantity of the order.

Update Quantity

```
api.update_order(trade=trade, qty=1)
api.update_status(api.stock_account)
trade
```

Cut

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=<DayTrade.YesNo: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27af',
        seqno='000002',
        ordno='000001',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='A123456789',
            broker_id='9A95',
            account_id='1234567',
            signed=True
        ),
        custom_field='test',
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        daytrade_short=False
    ),
    status=OrderStatus(
        id='531e27af',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        order_quantity=3,
        cancel_quantity=1,
        deals=[]
    )
)
```

CANCEL ORDER

Cancel Order

```
api.cancel_order(trade)
api.update_status(api.stock_account)
trade
```

 Cut

```

Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27af',
        seqno='000002',
        ordno='000001',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='A123456789',
            broker_id='9A95',
            account_id='1234567',
            signed=True
        ),
        custom_field='test',
        price_type=<StockPriceType.LMT: 'LNT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        daytrade_short=False
    ),
    status=OrderStatus(
        id='531e27af',
        status=<Status.Cancelled: 'Cancelled'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        order_quantity=3,
        cancel_quantity=3,
        deals=[]
    )
)
)

```

DEAL

 Update Status

```

api.update_status(api.stock_account)
trade

```

 **Buy**

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name='永豐金',
        category='17',
        unit=1000,
        limit_up=19.05,
        limit_down=15.65,
        reference=17.35,
        update_date='2023/01/12',
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27af',
        seqno='000002',
        ordno='000001',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='A123456789',
            broker_id='9A95',
            account_id='1234567',
            signed=True
        ),
        custom_field='test',
        price_type=<StockPriceType.LMT: 'LNT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        daytrade_short=False
    ),
    status=OrderStatus(
        id='531e27af',
        status=<Status.Filled: 'Filled'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 11, 18, 3, 867490),
        order_quantity=3,
        deals=[
            Deal(seq='000001', price=17, quantity=3, ts=1673501631.62918)
        ]
    )
)
```

Examples

[Stock place order jupyter link](#)

ACTION **Buy**

```
order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)
```

 **Sell**

```
order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.Sell,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)
```

Daytrade Short

version>=1.0 version<1.0

```
order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.SELL,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    daytrade_short=True,
    custom_field="test",
    account=api.stock_account
)

order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.SELL,
    price_type=sj.constant.TFTStockPriceType.LMT,
    order_type=sj.constant.TFTOrderType.ROD,
    order_lot=sj.constant.TFTStockOrderLot.Common,
    first_sell=sj.constant.StockFirstSell.Yes,
    custom_field="test",
    account=api.stock_account
)
```

ROD + LMT

ROD + LMT

version>=1.0 version<1.0

```
order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.SELL,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)

order = api.Order(
    price=12,
    quantity=1,
    action=sj.constant.Action.SELL,
    price_type=sj.constant.TFTStockPriceType.LMT,
    order_type=sj.constant.TFTOrderType.ROD,
    order_lot=sj.constant.TFTStockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)
```

4.5.2 Futures and Option

Reminder

First, you need to [login](#) and [activate CA](#).

FUTURES ORDER

Order Attributes

```
price (float or int): the price of order
quantity (int): the quantity of order
action (str): order action to buy or sell
    (Buy, Sell)
price_type (str): pricing type of order
    (LMT, MKT, MKP)
order_type (str): the type of order
    (ROD, IOC, FOK)
octype (str): the type or order to open new position or close position future only
    (Auto, New, Cover, DayTrade) (自動、新倉、平倉、當沖)
account (:obj:Account): which account to place this order
ca (binary): the ca of this order
```

PLACE ORDER

Product information (`contract`) and order information (`order`) must be provided when placing an order.

Place Order

```
api.place_order?

Signature:
api.place_order(
    contract: shioaji.contracts.Contract,
    order: shioaji.order.Order,
    timeout: int = 5000,
    cb: Callable[[shioaji.order.Trade], NoneType] = None,
) -> shioaji.order.Trade

Docstring:
placing order
```

Contract

```
contract = api.Contracts.Futures.TXF.TXF202301
```

Order

version>=1.0 version<1.0

```
order = api.Order(
    action=sj.constant.Action.Buy,
    price=14400,
    quantity=3,
    price_type=sj.constant.FuturesPriceType.LNT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)

order = api.Order(
    action=sj.constant.Action.Buy,
    price=14400,
    quantity=3,
    price_type=sj.constant.FuturesPriceType.LNT,
    order_type=sj.constant.FuturesOrderType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
```

Place Order

```
trade = api.place_order(contract, order)
trade
```

Cut

```
Trade(
    contract=Future(
        code='TXXF2301',
        symbol='TXXF202301',
        name='臺股期貨01',
        category='TXXF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=14400,
        quantity=3,
        id='5efffde1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffde1',
        status=<Status.PendingSubmit: 'PendingSubmit'>,
        status_code='00',
        order_datetime=datetime.datetime(2023, 1, 12, 14, 56, 13, 995651),
        deals=[]
    )
)
```

After `place_order`, you will also receive the information sent back from the exchange. For details, please refer to [Order & Deal Event](#).

To update the `trade` status, you need to call `update_status`.

Update Status

```
api.update_status(api.futopt_account)
trade
```

✓ Cut

```
Trade(
    contract=Future(
        code='TXXF202301',
        symbol='TXXF202301',
        name='臺股期貨01',
        category='TXXF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=Action.Buy: 'Buy',
        price=14400,
        quantity=3,
        id='5efffdde1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffdde1',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),
        deals=[]
    )
)
```

>Status of Trade

- PendingSubmit : Sending
- PreSubmitted : Reservation
- Submitted : Send Successfully
- Failed : Failed
- Cancelled : Cancelled
- Filled : Complete Fill
- PartFilled : Part Fill

UPDATE ORDER

Update Order

```
api.update_order?

Signature:
    api.update_order(
        trade: shioaji.order.Trade,
        price: Union[pydantic.types.StrictInt, float] = None,
        qty: int = None,
        timeout: int = 5000,
        cb: Callable[[shioaji.order.Trade], NoneType] = None,
    ) -> shioaji.order.Trade
Docstring: update the order price or qty
```

Update Price**Update Price**

```
api.update_order(trade=trade, price=14450)
api.update_status(api.futopt_account)
trade
```

Out

```
Trade(
    contract=Future(
        code='TXFA3',
        symbol='TXF202301',
        name='臺股期貨01',
        category='TXF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=14400,
        quantity=3,
        id='5efffd1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffd1',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),
        modified_price=14450,
        order_quantity=3,
        deals=[]
    )
)
```

Update Quantity (Reduce)

`update_order` can only reduce the quantity of the order.

Update Quantity

```
api.update_order(trade=trade, qty=1)
api.update_status(api.futopt_account)
trade
```

 **Out**

```

Trade(
    contract=Future(
        code='TXFA3',
        symbol='TF202301',
        name='臺股期貨01',
        category='TXX',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=14400,
        quantity=3,
        id='5efffd1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LNT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffd1',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),
        order_quantity=3,
        cancel_quantity=1,
        deals=[]
    )
)
)

```

CANCEL ORDER **Cancel Order**

```

api.cancel_order(trade)
api.update_status(api.futopt_account)
trade

```

Cut

```
Trade(
    contract=Future(
        code='TXFA3',
        symbol='TF202301',
        name='臺股期貨01',
        category='TTF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=Action.Buy: 'Buy',
        price=14400,
        quantity=3,
        id='5efffd1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffd1',
        status=<Status.Cancelled: 'Cancelled'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),
        order_quantity=3,
        cancel_quantity=3,
        deals=[]
    )
)
```

DEAL

Update Status

```
api.update_status(api.futopt_account)
trade
```

Buy

```
Trade(
    contract=Future(
        code='TXFA3',
        symbol='TF202301',
        name='臺股期貨01',
        category='TTF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16270.0,
        limit_down=13312.0,
        reference=14791.0,
        update_date='2023/01/12'
    ),
    order=Order(
        action=Action.Buy: 'Buy',
        price=14400,
        quantity=3,
        id='5efffd1',
        seqno='000004',
        ordno='000003',
        account=Account(
            account_type=<AccountType.Future: 'F'>,
            person_id='A123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(
        id='5efffd1',
        status=<Status.Filled: 'Filled'>,
        status_code='00',
        order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),
        order_quantity=3,
        deals=[
            Deal(seq='000001', price=14400, quantity=3, ts=1673501631.62918)
        ]
    )
)
```

Examples

Future and Option place order jupyter link

ACTION**Buy**

```
order = api.Order(
    action=sj.constant.Action.Buy,
    price=14400,
    quantity=2,
    price_type=sj.constant.FuturesPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
```

Sell

```
order = api.Order(
    action=sj.constant.Action.Sell,
    price=14400,
    quantity=2,
    price_type=sj.constant.FuturesPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
```

ROD + LMT**ROD + LMT**

```
order = api.Order(  
    action=sj.constant.Action.Sell,  
    price=1400,  
    quantity=2,  
    price_type=sj.constant.FuturesPriceType.LNT,  
    order_type=sj.constant.OrderType.ROD,  
    octype=sj.constant.FuturesOCType.Auto,  
    account=api.futopt_account  
)
```

4.5.3 Intraday Odd

Reminder

First, you need to [login](#) and activate CA.

place intraday odd order jupyter link

PLACE ORDER



```
contract = api.Contracts.Stocks.TSE.TSE0050
order = api.Order(
    price=90,
    quantity=10,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.IntradayOdd,
    account=api.stock_account,
)
trade = api.place_order(contract, order)
trade
```



```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='0050',
        symbol='TSE0050',
        name='元大台灣50',
        category='00',
        limit_up=115.8,
        limit_down=94.8,
        reference=105.3,
        update_date='2020/09/21',
        margin_trading_balance=15390,
        short_selling_balance=2,
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=90.0,
        quantity=10,
        id='38e68afe',
        seqno='482283',
        ordno='W4313',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='YOUR_PERSON_ID',
            broker_id='9A95',
            account_id='0506112',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        order_lot=<StockOrderLot.IntradayOdd: 'IntradayOdd'>
    ),
    status=OrderStatus(
        id='38e68afe',
        status=<Status.Submitted: 'Submitted'>,
        status_code='00',
        order_datetime=datetime.datetime(2020, 9, 21, 14, 38, 51),
        deals=[]
    )
)
```

UPDATE ORDER**A**ttention**Intraday Odd cannot update price.**

update_order can only reduce the quantity of the order.

Update Quantity

```
api.update_order(trade=trade, qty=2)
api.update_status(api.stock_account)
trade
```

Cut

```
Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='0050',
        symbol='TSE0050',
        name='元大台灣50',
        category='00',
        limit_up=115.8,
        limit_down=94.8,
        reference=105.3,
        update_date='2020/09/21',
        margin_trading_balance=15390,
        short_selling_balance=2,
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=90.0,
        quantity=10,
        id='9b44c3b2',
        seqno='482293',
        ordno='WA328',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='YOUR_PERSON_ID',
            broker_id='9A95',
            account_id='0506112',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        order_lot=<StockOrderLot.IntradayOdd: 'IntradayOdd'>,
        status=OrderStatus(
            id='9b44c3b2',
            status=<Status.Submitted: 'Submitted'>,
            status_code='00',
            order_datetime=datetime(2020, 9, 21, 14, 54, 36),
            cancel_quantity=2,
            deals=[]
        )
    )
)
```

CANCEL ORDER**i**Cancel Order

```
api.cancel_order(trade)
api.update_status(api.stock_account)
trade
```

 Cut

```

Trade(
    contract=Stock(
        exchange=<Exchange.TSE: 'TSE'>,
        code='0050',
        symbol='TSE0050',
        name='元大台灣50',
        category='00',
        limit_up=115.8,
        limit_down=94.8,
        reference=105.3,
        update_date='2020/09/21',
        margin_trading_balance=15390,
        short_selling_balance=2,
        day_trade=<DayTrade.Yes: 'Yes'>
    ),
    order=Order(
        action=<Action.Buy: 'Buy'>,
        price=90.0,
        quantity=10,
        id='9b44c3b2',
        seqno='482293',
        ordno='W4328',
        account=Account(
            account_type=<AccountType.Stock: 'S'>,
            person_id='YOUR_PERSON_ID',
            broker_id='9A95',
            account_id='0506112',
            signed=True
        ),
        price_type=<StockPriceType.LMT: 'LNT'>,
        order_type=<OrderType.ROD: 'ROD'>,
        order_lot=<StockOrderLot.IntradayOdd: 'IntradayOdd'>
    ),
    status=OrderStatus(
        id='9b44c3b2',
        status=<Status.Cancelled: 'Cancelled'>,
        status_code='00',
        order_datetime=datetime(2020, 9, 21, 14, 54, 36),
        cancel_quantity=10,
        deals=[]
    )
)

```

4.5.4 Combo

Reminder

First, you need to [login](#) and [activate CA](#).

PLACE COMBO ORDER.

Combo orders offer types include: **Price Call/Put Spreads**, **Time Call/Put Spreads**, **Straddles**, **Strangles**, **Conversions** and **Reversal**. Please refer to the futures exchange [document](#) for details on the combo rules.

place_comboorder

```
api.place_comboorder?

Signature:
    api.place_comboorder(
        combo_contract: shioaji.contracts.ComboContract,
        order: shioaji.order.ComboOrder,
        timeout: int = 500,
        cb: Callable[[shioaji.order.ComboTrade], NoneType] = None,
    )
Docstring:
    placing combo order
```

Product information (`contract`) and order information (`order`) must be provided when placing an order. The order of the contracts is irrelevant, only the approved combination is required.

Combo Contract

```
contract_1 = api.Contracts.Options.TX4.TX4202111017850C
contract_2 = api.Contracts.Options.TX4.TX4202111017850P
combo_contract = sj.contracts.ComboContract(
    legs=[
        sj.contracts.ComboBase(action="Sell", **contract_1.dict()),
        sj.contracts.ComboBase(action="Sell", **contract_2.dict()),
    ]
)
```

Order

```
order = api.ComboOrder(
    price_type="LMT",
    price=1,
    quantity=1,
    order_type="IOC",
    octype=sj.constant.FuturesOCType.New,
)
```



```
trade = api.place_comboorder(combo_c, order)
```

CANCEL COMBO ORDER

`Trade` is the order to be deleted, which can be obtained from the [update_combostatus](#) .



```
api.cancel_comboorder(trade)
```

UPDATE COMBO STATUS

Like `list_trades` and `update_status` concepts. Before getting the combo status, the status must be updated with `update_combostatus`.



```
api.update_combostatus()  
api.list_combotrades()
```



```
[  
    ComboTrade(  
        contract=ComboContract(  
            legs=[  
                ComboBase(  
                    security_type=<SecurityType.Option: 'OPT'>,  
                    exchange=<Exchange.TAIFEX: 'TAIFEX'>,  
                    code='TX516000L1',  
                    symbol='臺指選擇權12W5月 16000C',  
                    name='臺指選擇權12W5月 16000C',  
                    category='TX5',  
                    delivery_month='202112',  
                    delivery_date='2021/12/29',  
                    strike_price=16000.0,  
                    option_right=<OptionRight.Call: 'C'>,  
                    underlying_kind='I',  
                    unit=1,  
                    limit_up=3630.0,  
                    limit_down=68.0,  
                    reference=1850.0,  
                    update_date='2021/12/23',  
                    action=<Action.Sell: 'Sell'>),  
                ComboBase(  
                    security_type=<SecurityType.Option: 'OPT'>,  
                    exchange=<Exchange.TAIFEX: 'TAIFEX'>,  
                    code='TX516000XL1',  
                    symbol='臺指選擇權12W5月 16000P',  
                    name='臺指選擇權12W5月 16000P',  
                    category='TX5',  
                    delivery_month='202112',  
                    delivery_date='2021/12/29',  
                    strike_price=16000.0,  
                    option_right=<OptionRight.Put: 'P'>,  
                    underlying_kind='I',  
                    unit=1,  
                    limit_up=1780.0,  
                    limit_down=0.1,  
                    reference=0.9,  
                    update_date='2021/12/23',  
                    action=<Action.Sell: 'Sell'>  
                )  
            ]  
        ),  
        order=Order(  
            action=<Action.Sell: 'Sell'>,  
            price=1.0,  
            quantity=1,  
            id='46989de8',  
            seqno='743595',  
            ordno='000000',  
            account=Account(  
                account_type=<AccountType.Future: 'F'>,  
                person_id='YOUR_PERSON_ID',  
                broker_id='FO02000',  
                account_id='1234567',  
                signed=True  
            ),  
            price_type=<StockPriceType.LMT: 'LMT'>,  
            order_type=<OrderType.IOC: 'IOC'>,  
            octype=<FuturesOCType.New: 'New'>  
        ),  
        status=ComboStatus(  
            id='46989de8',  
            status=<Status.Failed: 'Failed'>,  
            status_code='9909',  
            order_datetime=datetime(2021, 12, 23, 8, 46, 47),  
            msg='可委託金額不足',  
            modified_price=1.0,  
            deals={}
        )
    )
]
```

4.5.5 Reserve Order

When Stock triggers some transaction abnormal conditions, it is necessary to `Reserve Order` in advance. Abnormal conditions include: watch out for stocks, warn about stocks, dispose of stocks, and manage stocks.

Reminder

- First, you need to [login](#) and [activate CA](#).
- Service hours are from 8:00 to 14:30 on trading days.

GET STOCK RESERVE SUMMARY STATUS



```
reserve_summary_resp = api.stock_reserve_summary(account)
```

Success

```
ReserveStocksSummaryResponse(
    response=ReserveStocksSummary(
        stocks=[
            ReserveStockSummary(
                contract=Contract(
                    security_type=<SecurityType.Stock: 'STK'>,
                    exchange=Exchange.TSE: 'TSE',
                    code='2890',
                    name='永豐金'
                ),
                available_share=5000,
                reserved_share=0
            )
        ],
        account=StockAccount(
            person_id='X123456789',
            broker_id='9A95',
            account_id='12345678',
            signed=True
        )
    )
)
```

RESERVE STOCK



```
contract = api.Contracts.Stocks["2890"]
resp = api.reserve_stock(account, contract, 1000)
```



```
ReserveStockResponse(
    response=ReserveOrderResp(
        contract=Stock(
            exchange=<Exchange.TSE: 'TSE'>,
            code='2890',
            symbol='TSE2890',
            name='永豐金'
        ),
        account=StockAccount(
            person_id='X123456789',
            broker_id='9495',
            account_id='12345678',
            signed=True),
        share=1000,
        status=True,
        info=''
    )
)
```

GET STOCK RESERVE DETAIL STATUS



```
resp = api.stock_reserve_detail(account)
```



```
ReserveStocksDetailResponse(
    response=ReserveStocksDetail(stocks=[
        ReserveStockDetail(
            contract=Contract(
                security_type=<SecurityType.Stock: 'STK'>,
                exchange=<Exchange.TSE: 'TSE'>,
                code='6153',
                name='嘉聯益'
            ),
            share=1000,
            order_ts=1638253253,
            status=True,
            info='已完成'
        )
    ],
    account=StockAccount(
        person_id='X123456789',
        broker_id='9495',
        account_id='12345678',
        signed=True)
)
```

RESERVE EARMARKING



```
contract = api.Contracts.Stocks["2890"]
resp = api.reserve_earmarking(account, contract, 1000, 15.15)
```



```
ReserveEarmarkingResponse(
    response=EarmarkingOrderResp(
        contract=Stock(
            exchange=<Exchange.TSE: 'TSE'>,
            code='2890',
            symbol='TSE2890',
            name='永豐金',
        ),
        account=StockAccount(
            person_id='X123456789',
            broker_id='9A95',
            account_id='12345678',
            signed=True
        ),
        share=1000,
        price=15.15,
        status=True,
        info='OK'
    )
)
```

GET EARMARKING DETAIL STATUS



```
api.earmarking_detail(account)
```



```
EarmarkStocksDetailResponse(
    response=EarmarkStocksDetail(
        stocks=[
            EarmarkStockDetail(
                contract=Contract(
                    security_type=<SecurityType.Stock: 'STK'>,
                    exchange=<Exchange.TSE: 'TSE'>,
                    code='2890',
                    name='永豐金'
                ),
                share=1000,
                price=15.15,
                amount=15171,
                order_ts=1638416488,
                status=False,
                info='扣款失败'),
            EarmarkStockDetail(
                contract=Contract(
                    security_type=<SecurityType.Stock: 'STK'>,
                    exchange=<Exchange.TSE: 'TSE'>,
                    code='2890',
                    name='永豐金'
                ),
                share=1000,
                price=15.15,
                amount=15171,
                order_ts=1638415662, status=True,
                info='')
        ],
        account=StockAccount(
            person_id='X123456789',
            broker_id='9A95',
            account_id='12345678',
            signed=True)
    )
)
```

EXAMPLE

Query the reserve status of all accounts under your name.



```
import shioaji as sj

api = sj.Shioaji()
accounts = api.login("YOUR_PERSON_ID", "YOUR_PASSWORD", contracts_timeout=10000)
api.activate_ca(
    ca_path="/c/your/ca/path/Sinopac.pfx",
    ca_passwd="YOUR_CA_PASSWORD",
    person_id="Person of this Ca",
)
for account in accounts:
    if account.account_type == AccountType.Stock:
        reserve_summary_resp = api.stock_reserve_summary(account)
        for reserve_stock_summary in reserve_summary_resp.response.stocks:
            if reserve_stock_summary.available_share:
                resp = api.reserve_stock(
                    account,
                    reserve_stock_summary.contract,
                    reserve_stock_summary.available_share
                )
```

4.5.6 Update Status

⚠️ Reminder

First, you need to [login](#) and [activate CA](#).

Before obtaining the `Trade` status, it must be updated with `update_status`. If you cannot successfully `update_order` or `cancel_order`, you can use `update_status` to update the specific `trade` status, and check the `OrderStatus` in `trade`, whether it is available to modify the order.

The `update_status` defaults to querying all accounts under the user's name. If you wish to inquire about a specific account, provide the account as a parameter to `account`.

💡 Update Status

```
api.update_status?
```

✅

Signature:
 `api.update_status(
 account: shioaji.account.Account = None,
 trade: shioaji.order.Trade = None,
 timeout: int = 5000,
 cb: Callable[[List[shioaji.order.Trade]], NoneType] = None,
)`
Docstring: update status of `all` trades you have

GET STOCK TRADES

💡 Get Stock Trades

```
api.update_status(api.stock_account)  
api.list_trades()
```



```
[
    Trade(
        contract=Stock(
            exchange=<Exchange.TSE: 'TSE'>,
            code='2890',
            symbol='TSE2890',
            name='永豐金',
            category='17',
            unit=1000,
            limit_up=19.05,
            limit_down=15.65,
            reference=17.35,
            update_date='2023/01/12',
            day_trade=<DayTrade.YesNo: 'Yes'>
        ),
        order=Order(
            action=<Action.Buy: 'Buy'>,
            price=17,
            quantity=3,
            id='531e27af',
            seqno='000002',
            ordno='000001',
            account=Account(
                account_type=<AccountType.Stock: 'S'>,
                person_id='A123456789',
                broker_id='9A95',
                account_id='1234567',
                signed=True
            ),
            custom_field='test',
            price_type=<StockPriceType.LMT: 'LMT'>,
            order_type=<OrderType.ROD: 'ROD'>,
            daytrade_short=True
        ),
        status=OrderStatus(
            id='531e27af',
            status=<Status.Filled: 'Filled'>,
            status_code='00',
            order_datetime=datetime.datetime(2023, 1, 12, 11, 18, 3, 867490),
            order_quantity=3,
            deals=[
                Deal(seq='000001', price=17, quantity=3, ts=1673501631.62918)
            ]
        )
    )
]
]
```

GET FUTURES TRADES



Get Futures Trades

```
api.update_status(api.futopt_account)
api.list_trades()
```

Trade

```
[  
    Trade(  
        contract=Future(  
            code='TFX3',  
            symbol='TFX202301',  
            name='臺股期貨01',  
            category='TFX',  
            delivery_month='202301',  
            delivery_date='2023/01/30',  
            underlying_kind='I',  
            unit=1,  
            limit_up=16270.0,  
            limit_down=13312.0,  
            reference=14791.0,  
            update_date='2023/01/12'  
        ),  
        order=Order(  
            action=<Action.Buy: 'Buy'>,  
            price=14400,  
            quantity=3,  
            id='5efffd1',  
            seqno='000004',  
            ordno='000003',  
            account=Account(  
                account_type=<AccountType.Future: 'F'>,  
                person_id='A123456789',  
                broker_id='F002000',  
                account_id='1234567',  
                signed=True  
            ),  
            price_type=<StockPriceType.LMT: 'LMT'>,  
            order_type=<OrderType.ROD: 'ROD'>  
        ),  
        status=OrderStatus(  
            id='5efffd1',  
            status=<Status.Filled: 'Filled'>,  
            status_code='00',  
            order_datetime=datetime(2023, 1, 12, 14, 56, 13, 995651),  
            order_quantity=3,  
            deals=[  
                Deal(seq='000001', price=14400, quantity=3, ts=1673501631.62918)  
            ]  
        )  
    )  
]
```

UPDATE SPECIFIC TRADE

Update Trade

```
# you can get trade from place_order  
# trade = api.place_order(contract, order)  
  
# or get from api.list_trades  
# trade = api.list_trades()[0]  
  
api.update_status(trade=trade)
```

TRADE STATUS

OrderStatus

```
id (str): the id uses to correlate the order object  
status (:obj:Status): the status of order {Cancelled, Filled, PartFilled, Failed, PendingSubmit, PreSubmitted, Submitted}  
status_code (str): the code of status  
order_datetime (datetime): order time  
order_quantity (int): order quantity  
modified_price (float): the price of modification  
cancel_quantity (int): the quantity of cancel  
deals (:List:Deal): information of filled order
```

Deal

```
seq (str): deal sequence number  
price (int or float): deal price  
quantity (int): deal quantity  
ts (float): deal timestamp
```

4.5.7 CallBack Info.

Stock

Order CallBack

When the exchange receives the order, it will return the callback. The callback is divided into four parts, including operation, order, status and contract. We will explain in detail below.

Order Event

version>=1.0 version<1.0

```
OrderState.StockOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '97b63e2f',
        'seqno': '267677',
        'ordno': 'IN394',
        'account': {
            'account_type': 'S',
            'person_id': '',
            'broker_id': '9A95',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 16.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'order_cond': 'Cash',
        'order_lot': 'Common',
        'custom_field': 'test'
    },
    'status': {
        'id': '97b63e2f',
        'exchange_ts': 1673576134.038,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': '137'
    },
    'contract': {
        'security_type': 'STK',
        'exchange': 'TSE',
        'code': '2890',
        'symbol': '',
        'name': '',
        'currency': 'TWD'
    }
}

OrderState.TFTOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '97b63e2f',
        'seqno': '267677',
        'ordno': 'IN394',
        'account': {
            'account_type': 'S',
            'person_id': '',
            'broker_id': '9A95',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 16.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'order_cond': 'Cash',
        'order_lot': 'Common',
        'custom_field': 'test'
    },
    'status': {
        'id': '97b63e2f',
        'exchange_ts': 1673576134.038,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': '137'
    },
    'contract': {
        'security_type': 'STK',
        'exchange': 'TSE',
        'code': '2890',
        'symbol': '',
        'name': '',
        'currency': 'TWD'
    }
}
```

OrderCallBack Info.

operation

```
op_type (str): {
    "New": new order,
    "Cancel": cancel order,
    "UpdatePrice": update price,
    "UpdateQty": update quantity
}
op_code (str): {"00": success, others: fail}
op_msg (str): error message
```

order

```
id (str): same as the trade_id in SDeal
seqn (str): sequence number
ordn (str): order number
account (dict): account info
action (str): {Buy, Sell}
price (float or int): order price
quantity (int): order quantity
order_type (str): the type of order {ROD, IOC, FOK}
price_type (str): pricing type of order {LMT, MKT, MKP}
order_cond (str): {
    Cash: 現股,
    MarginTrading: 融資,
    ShortSelling: 融券
}
order_lot (str): {
    Common: 整股,
    Fixing: 定盤,
    Odd: 盤後零股,
    IntradayOdd: 盤中零股
}
custom_field (str): memo field
```

status

```
id (str): same as the trade_id in SDeal
exchange_ts (int): exchange time
modified_price (float or int): modified price
cancel_quantity (int): cancel quantity
order_quantity (int): order quantity
web_id (str): web id
```

contract

```
security_type (str): security type
exchange (str): exchange
code (str): code id
symbol (str): symbol
name (str): name
currency (str): currency
```

DealCallBack

When the matching is successful, the exchange will send a transaction report notification. Successful matching includes partial transactions and complete transactions. You can confirm whether it is the same order from the `id` in the order callback to the `trade_id` in the deal callback.

Deal Event

version>=1.0 version<1.0

```
OrderState.StockDeal {
    'trade_id': '9c6aae2eb',
    'seqno': '269866',
    'ordno': 'IN497',
    'exchange_seq': '669915',
    'broker_id': '9A95',
    'account_id': '1234567',
    'action': 'Buy',
    'code': '2890',
    'order_cond': 'Cash',
    'order_lot': 'IntradayOdd',
    'price': 267.5,
    'quantity': 3,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1673577256.354
}

OrderState.TFTDeal {
    'trade_id': '9c6aae2eb',
    'seqno': '269866',
    'ordno': 'IN497',
    'exchange_seq': '669915',
    'broker_id': '9A95',
    'account_id': '1234567',
    'action': 'Buy',
    'code': '2890',
    'order_cond': 'Cash',
    'order_lot': 'IntradayOdd',
    'price': 267.5,
    'quantity': 3,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1673577256.354
}
```

Deal Callback Info.

```
trade_id (str): same as the id in StockOrder
seqno (str): sequence number
ordno (str): The first 5 characters is the same as ordno in StockOrder. The last 3 characters represent the deal sequence number.
exchange_seq (str): exchange sequence number
broker_id (str): broker id
account_id (str): account
action (str): {Buy, Sell}
code (str): code id
order_cond (str): {
    Cash: 現股,
    MarginTrading: 融資,
    ShortSelling: 融券
}
order_lot (str): {
    Common: 整股,
    Fixing: 定盤,
    Odd: 盤後零股,
    IntradayOdd: 盤中零股
}
price (float or int): deal price
quantity (int): deal quantity
web_id (str): web id
custom_field (str): memo field
ts (int): deal timestamp
```

Note

you "may" receive the deal event sooner than the order event due to message priority in exchange.

Handle Callback

If you would like to handle callback info, please refer to [Callback](#).

Futures

Order CallBack

When the exchange receives the order, it will return the callback. The callback is divided into four parts, including operation, order, status and contract. We will explain in detail below.

 Order Event

version>=1.0 version<1.0

```

OrderState.FuturesOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': 'fcb42a6e',
        'seqno': '585886',
        'ordno': '00',
        'account': {
            'account_type': 'F',
            'person_id': '',
            'broker_id': 'F002000',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 27000.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LNT',
        'market_type': 'Night',
        'oc_type': 'New',
        'subaccount': '',
        'combo': False
    },
    'status': {
        'id': 'fcb42a6e',
        'exchange_ts': 1764731566.0,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': 'Z'
    },
    'contract': {
        'security_type': 'FUT',
        'code': 'TXF',
        'full_code': 'TXFL5',
        'exchange': 'T1M',
        'delivery_month': '202512',
        'delivery_date': '',
        'strike_price': 0.0,
        'option_right': 'Future'
    }
}

OrderState.Order {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': 'fcb42a6e',
        'seqno': '585886',
        'ordno': '00',
        'account': {
            'account_type': 'F',
            'person_id': '',
            'broker_id': 'F002000',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 27000.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LNT',
        'market_type': 'Night',
        'oc_type': 'New',
        'subaccount': '',
        'combo': False
    },
    'status': {
        'id': 'fcb42a6e',
        'exchange_ts': 1764731566.0,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': 'Z'
    },
    'contract': {
        'security_type': 'FUT',
        'code': 'TXF',
        'full_code': 'TXFL5',
        'exchange': 'T1M',
        'delivery_month': '202512',
        'delivery_date': '',
        'strike_price': 0.0,
        'option_right': 'Future'
    }
}

```

Order CallBack Info.

operation

```
op_type (str): {
    "New": new order,
    "Cancel": cancel order,
    "UpdatePrice": update price,
    "UpdateQty": update quantity
}
op_code (str): {"00": success, others: fail}
op_msg (str): error message
```

order

```
id (str): same as the trade_id in FuturesDeal
seqn (str): sequence number
ordn (str): order number
account (dict): account info
action (str): {Buy, Sell}
price (float or int): order price
quantity (int): order quantity
order_type (str): the type of order {ROD, IOC, FOK}
price_type (str): pricing type of order {LMT, MKT, MKP}
market_type (str): {Day, Night}
oc_type (str): {New: 新倉, Cover: 平倉, Auto: 自動}
subaccount (str): subaccount
combo (bool): whether order is combo order
```

status

```
id (str): same as the trade_id in FuturesDeal
exchange_ts (int): exchange time
modified_price (float or int): modified price
cancel_quantity (int): cancel quantity
order_quantity (int): order quantity
web_id (str): web id
```

contract

```
security_type (str): security type
code (str): code id
full_code (str): code id (with delivery month)
exchange (str): exchange
delivery_month (str): delivery month
delivery_date (str): delivery date
strike_price (float): strike price
option_right (str): {Future, OptionCall, OptionPut}
```

Deal CallBack

When the matching is successful, the exchange will send a transaction report notification. Successful matching includes partial transactions and complete transactions. You can confirm whether it is the same order from the `id` in the order callback to the `trade_id` in the deal callback.

Deal Event

version>=1.0 version<1.0

```
OrderState.FuturesDeal {
    'trade_id': '4e6df0f6',
    'seqno': '458545',
    'ordno': 'TA0dex10',
    'exchange_seq': 'j5006396',
    'broker_id': 'F002000',
    'account_id': '1234567',
    'action': 'Sell',
    'code': 'TX1',
    'full_code': 'TX127900L5',
    'price': 25.0,
    'quantity': 1,
    'subaccount': '',
    'security_type': 'OPT',
    'delivery_month': '202512',
    'strike_price': 27900.0,
    'option_right': 'OptionCall',
    'market_type': 'Day',
    'combo': False,
    'ts': 1764685425.0
}

OrderState.FDeal {
    'trade_id': '4e6df0f6',
    'seqno': '458545',
    'ordno': 'TA0dex10',
    'exchange_seq': 'j5006396',
    'broker_id': 'F002000',
    'account_id': '1234567',
    'action': 'Sell',
    'code': 'TX1',
    'full_code': 'TX127900L5',
    'price': 25.0,
    'quantity': 1,
    'subaccount': '',
    'security_type': 'OPT',
    'delivery_month': '202512',
    'strike_price': 27900.0,
    'option_right': 'OptionCall',
    'market_type': 'Day',
    'combo': False,
    'ts': 1764685425.0
}
```

FuturesDeal

trade_id (**str**): same as the **id** in FuturesOrder
 seqno (**str**): sequence number
 ordno (**str**): The first **5** characters is the same as ordno in FuturesOrder. The last **3** characters represent the deal sequence number.
 exchange_seq (**str**): exchange sequence number
 broker_id (**str**): broker id
 account_id (**str**): account
 action (**str**): buy/sell
 code (**str**): code id
 full_code (**str**): code id (with delivery month)
 price (**float or int**): deal price
 quantity (**int**): deal quantity
 subaccount (**str**): subaccount
 security_type (**str**): security type
 delivery_month (**str**): delivery month
 strike_price (**float**): strike price
 option_right (**str**): (Future, OptionCall, OptionPut)
 market_type (**str**): {Day, Night}
 ts (**int**): deal timestamp

Note

you "may" receive the deal event sooner than the order event due to message priority in exchange.

Handle Callback

If you would like to handle callback info, please refer to [Callback](#).

4.6 CallBack

4.6.1 Order Event

Each time you `place_order`, `update_order` or `cancel_order`, by default, you will receive an order event (or deal event) from exchange. If you don't want receive both events, please refer to [Subscribe Trade](#). We also provide interface to handle order and deal events. It's extremely helpful if you are implementing your custom trading system.

Handle Order Callback

You can use `set_order_callback` to handle order/deal events. The example below shows custom order callback function (`order_cb`), which will print `my_order_callback` first and then print the order/deal event.

Set Order Callback

```
def order_cb(stat, msg):
    print('my_order_callback')
    print(stat, msg)

api.set_order_callback(order_cb)
```

Place Order

version>=1.0 version<1.0

```
contract = api.Contracts.Stocks.TSE.TSE2890
order = api.Order(
    price=16,
    quantity=1,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    order_lot=sj.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)
trade = api.place_order(contract, order)

contract = api.Contracts.Stocks.TSE.TSE2890
order = api.Order(
    price=16,
    quantity=1,
    action=sj.constant.Action.Buy,
    price_type=sj.constant.TFTStockPriceType.LMT,
    order_type=sj.constant.TFTOrderType.ROD,
    order_lot=sj.constant.TFTStockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)
trade = api.place_order(contract, order)
```

ORDER EVENT

Order Event

version>=1.0 version<1.0

```
my_order_callback
OrderState.StockOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '97b63e2f',
        'seqno': '267677',
        'ordno': 'IM394',
        'account': {
            'account_type': 'S',
            'person_id': '',
            'broker_id': '9A95',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 16.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'order_cond': 'Cash',
        'order_lot': 'Common',
        'custom_field': 'test'
    },
    'status': {
        'id': '97b63e2f',
        'exchange_ts': 1673576134.038,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': '137'
    },
    'contract': {
        'security_type': 'STK',
        'exchange': 'TSE',
        'code': '2890',
        'symbol': '',
        'name': '',
        'currency': 'TWD'
    }
}

my_order_callback
OrderState.TFTOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '97b63e2f',
        'seqno': '267677',
        'ordno': 'IM394',
        'account': {
            'account_type': 'S',
            'person_id': '',
            'broker_id': '9A95',
            'account_id': '1234567',
            'signed': True
        },
        'action': 'Buy',
        'price': 16.0,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'order_cond': 'Cash',
        'order_lot': 'Common',
        'custom_field': 'test'
    },
    'status': {
        'id': '97b63e2f',
        'exchange_ts': 1673576134.038,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        'order_quantity': 1,
        'web_id': '137'
    },
    'contract': {
        'security_type': 'STK',
        'exchange': 'TSE',
        'code': '2890',
        'symbol': '',
        'name': '',
        'currency': 'TWD'
    }
}
```

DEAL EVENT **Deal Event****version>=1.0 version<1.0**

```
my_order_callback
OrderState.StockDeal {
  'trade_id': '9c6ae2eb',
  'seqno': '269866',
  'ordno': 'IN497',
  'exchange_seq': '669915',
  'broker_id': '9A95',
  'account_id': '1234567',
  'action': 'Buy',
  'code': '2890',
  'order_cond': 'Cash',
  'order_lot': 'IntradayOdd',
  'price': 267.5,
  'quantity': 3,
  'web_id': '137',
  'custom_field': 'test',
  'ts': 1673577256.354
}
```

```
my_order_callback
OrderState.TFTDeal {
  'trade_id': '9c6ae2eb',
  'seqno': '269866',
  'ordno': 'IN497',
  'exchange_seq': '669915',
  'broker_id': '9A95',
  'account_id': '1234567',
  'action': 'Buy',
  'code': '2890',
  'order_cond': 'Cash',
  'order_lot': 'IntradayOdd',
  'price': 267.5,
  'quantity': 3,
  'web_id': '137',
  'custom_field': 'test',
  'ts': 1673577256.354
}
```

4.6.2 Event Callback

In this api, we use solace as mesh broker. This event mean the status for your client with solace connection situation. If you have no experience with networking, please skip this part, In defalut, we help you reconnect solace broker 50 times without any setting. Best way is keep your network connection alive.



```
@api.quote.on_event
def event_callback(resp_code: int, event_code: int, info: str, event: str):
    print(f'Event code: {event_code} | Event: {event}'')
```



Event code: 16 | Event: Subscribe or Unsubscribe ok

Like the quote callback, your can also set event cllback with two way.



```
api.quote.set_event_callback?
```



Signature: api.quote.set_event_callback(func:Callable[[int, int, str, str], NoneType]) -> None
Docstring: <no docstring>
Type: method

EVENT CODE

Event Code	Event Code Enumerator	Description
0	SOLCLIENT_SESSION_EVENT_UP_NOTICE	The Session is established.
1	SOLCLIENT_SESSION_EVENT_DOWN_ERROR	The Session was established and then went down.
2	SOLCLIENT_SESSION_EVENT_CONNECT_FAILED_ERROR	The Session attempted to connect but was unsuccessful.
3	SOLCLIENT_SESSION_EVENT_REJECTED_MSG_ERROR	The appliance rejected a published message.
4	SOLCLIENT_SESSION_EVENT_SUBSCRIPTION_ERROR	The appliance rejected a subscription (add or remove).
5	SOLCLIENT_SESSION_EVENT_RX_MSG_TOO_BIG_ERROR	The API discarded a received message that exceeded the Session buffer size.
6	SOLCLIENT_SESSION_EVENT_ACKNOWLEDGEMENT	The oldest transmitted Persistent/Non-Persistent message that has been acknowledged.
7	SOLCLIENT_SESSION_EVENT_ASSURED_PUBLISHING_UP	Deprecated -- see notes in solClient_session_startAssuredPublishing. The AD Handshake (that is, Guaranteed Delivery handshake) has completed; the publisher and Guaranteed messages can be sent.
8	SOLCLIENT_SESSION_EVENT_ASSURED_CONNECT_FAILED	Deprecated -- see notes in solClient_session_startAssuredPublishing. The appliance rejected the AD Handshake to start Guaranteed publishing. Use SOLCLIENT_SESSION_EVENT_ASSURED_DELIVERY_DOWN instead.
8	SOLCLIENT_SESSION_EVENT_ASSURED_DELIVERY_DOWN	Guaranteed Delivery publishing is not available. The guaranteed delivery capability on the session has been disabled by some action on the appliance.
9	SOLCLIENT_SESSION_EVENT_TE_UNSUBSCRIBE_ERROR	The Topic Endpoint unsubscribe command failed.
9	SOLCLIENT_SESSION_EVENT_DTE_UNSUBSCRIBE_ERROR	Deprecated name; SOLCLIENT_SESSION_EVENT_TE_UNSUBSCRIBE_ERROR is preferred.
10	SOLCLIENT_SESSION_EVENT_TE_UNSUBSCRIBE_OK	The Topic Endpoint unsubscribe completed.
10	SOLCLIENT_SESSION_EVENT_DTE_UNSUBSCRIBE_OK	Deprecated name; SOLCLIENT_SESSION_EVENT_TE_UNSUBSCRIBE_OK is preferred.
11	SOLCLIENT_SESSION_EVENT_CAN_SEND	The send is no longer blocked.
12	SOLCLIENT_SESSION_EVENT_RECONNECTING_NOTICE	The Session has gone down, and an automatic reconnect attempt is in progress.
13	SOLCLIENT_SESSION_EVENT_RECONNECTED_NOTICE	The automatic reconnect of the Session was successful; the Session was established again.
14	SOLCLIENT_SESSION_EVENT_PROVISION_ERROR	The endpoint create/delete command failed.
15	SOLCLIENT_SESSION_EVENT_PROVISION_OK	The endpoint create/delete command completed.
16	SOLCLIENT_SESSION_EVENT_SUBSCRIPTION_OK	The subscribe or unsubscribe operation has succeeded.
17	SOLCLIENT_SESSION_EVENT_VIRTUAL_ROUTER_NAME_CHANGED	The appliance's Virtual Router Name changed during a reconnect operation. This could render existing queue temporary topics invalid.
18	SOLCLIENT_SESSION_EVENT_MODIFYPROP_OK	The session property modification completed.
19	SOLCLIENT_SESSION_EVENT_MODIFYPROP_FAIL	The session property modification failed.

Event Code	Event Code Enumerator	Description
20	SOLCLIENT_SESSION_EVENT_REPUBLISH_UNACKED_MESSAGES	After successfully reconnecting a disconnected session, the SDK received an unknown publisher flow name response when reconnecting the GD publisher flow.

4.7 Account Data

4.7.1 Account Balance

The feature of account_balance is used to query account balance of stock account and you need to [login](#) first.



```
api.account_balance?
```



Signature:
api.account_balance(
 timeout: int = 5000,
 cb: Callable[[shioaji.position.AccountBalance], NoneType] = None,
)
Docstring: query stock account balance



```
api.account_balance()
```



AccountBalance(
 status=<FetchStatus.Fetched>,
 acc_balance=100000.0,
 date='2023-01-06 13:30:00.000000',
 errmsg=''
)



AccountBalance

status (FetchStatus): fetch status
acc_balance (float): account balance
date (str): query date
errmsg (str): error message

4.7.2 Trading Limits

Reminder

Query available on trading days from 8:30 to 15:00.

The feature of `trading_limits` is used to query trading limits of stock account and you need to [login](#) first.

Info

```
api.trading_limits?
```

Cat

Signature:
`api.trading_limits(
 account: shioaji.account.Account = None,
 timeout: int = 5000,
 cb: Callable[[shioaji.position.TradingLimits], NoneType] = None,
) -> shioaji.position.TradingLimits`
 Docstring: query stock account trading limits

Info

```
api.trading_limits(api.stock_account)
```

Cat

```
TradingLimits(  
    status=<FetchStatus.Fetched: 'Fetched'>,  
    trading_limit=1000000,  
    trading_used=0,  
    trading_available=1000000,  
    margin_limit=0,  
    margin_used=0,  
    margin_available=0,  
    short_limit=0,  
    short_used=0,  
    short_available=0  
)
```

TradingLimits

```
status (FetchStatus): fetch status  
trading_limit (int): trading limit  
trading_used (int): trading used  
trading_available (int): trading available  
margin_limit (int): margin limit  
margin_used (int): margin used  
margin_available (int): margin available  
short_limit (int): short selling limit  
short_used (int): short selling used  
short_available (int): short selling available
```

4.7.3 Margin

The feature of margin is used to query margin of futures account and you need to [login](#) first.



```
api.margin?
```



Signature:
api.margin(
 account: shioaji.account.Account = None,
 timeout: int = 5000,
 cb: Callable[[shioaji.position.Margin], NoneType] = None,
) -> shioaji.position.Margin
Docstring: query future account of margin



```
api.margin(api.futopt_account)
```



```
Margin(  
    status=<FetchStatus.Fetched: 'Fetched'>,  
    yesterday_balance=6000.0,  
    today_balance=6000.0,  
    deposit_withdrawal=0.0,  
    fee=0.0,  
    tax=0.0,  
    initial_l_margin=0.0,  
    maintenance_margin=0.0,  
    margin_call=0.0,  
    risk_indicator=999.0,  
    royalty_revenue_expenditure=0.0,  
    equity=6000.0,  
    equity_amount=6000.0,  
    option_openbuy_market_value=0.0,  
    option_opensell_market_value=0.0,  
    option_open_position=0.0,  
    option_settle_profitloss=0.0,  
    future_open_position=0.0,  
    today_future_open_position=0.0,  
    future_settle_profitloss=0.0,  
    available_margin=6000.0,  
    plus_margin=0.0,  
    plus_margin_indicator=0.0,  
    security_collateral_amount=0.0,  
    order_margin_premium=0.0,  
    collateral_amount=0.0  
)
```

Margin

```
status (FetchStatus): fetch status
yesterday_balance (float): balance of yesterday
today_balance (float): balance of today
deposit_withdrawal (float): deposit and withdrawal
fee (float): fee
tax (float): tax
initial_margin (float): margin of origin
maintenance_margin (float): margin of maintenance
margin_call (float): margin of call
risk_indicator (float): risk indicator
royalty_revenue_expenditure (float): revenue and expenditure of royalty
equity (float): equity
equity_amount (float): amount of equity
option_openbuy_market_value (float): value of option openbuy market
option_opensell_market_value (float): value of option opensell market
option_open_position (float): profit loss of open option
option_settle_profitloss (float): profit loss of settle option
future_open_position (float): profit loss of open future
today_future_open_position (float): profit loss of today open future
future_settle_profitloss (float): profit loss of settle future
available_margin (float): available margin
plus_margin (float): plus margin
plus_margin_indicator (float): indicator of plus margin
security_collateral_amount (float): amount of security collateral
order_margin_premium (float): order margin and order premium
collateral_amount (float): amount of collateral
```

4.7.4 Position

The feature of list_positions is used to query unrealized gain or loss of account and you need to [login](#) first.

Position



```
api.list_positions?
```



Signature:
`api.list_positions(
 account: shioaji.account.Account = None,
 unit: shioaji.constant.Unit = <Unit.Common: 'Common'>,
 timeout: int = 5000,
 cb: Callable[[List[Union[shioaji.position.StockPosition, shioaji.position.FuturePosition]]], NoneType] = None,
) -> List[Union[shioaji.position.StockPosition, shioaji.position.FuturePosition]]`

Docstring:
query account of unrealized gain or loss

Args:

- account (:obj:`Account`): choice the account from `listing` account (Default: stock account)

STOCKS

Common Stocks



```
api.list_positions(api.stock_account)
```



```
[  
    StockPosition(  
        id=0,  
        code='2890',  
        direction=<Action.Buy: 'Buy'>,  
        quantity=12,  
        price=2.79,  
        last_price=16.95,  
        pnl=169171.0,  
        yd_quantity=12,  
        margin_purchase_amount=0,  
        collateral=0,  
        short_sale_margin=0,  
        interest=0  
    )  
,  
]
```

To DataFrame



```
positions = api.list_positions(api.stock_account)  
df = pd.DataFrame(s._dict_ for s in positions)  
df
```

Out

id	code	direction	quantity	price	last_price	pnl	yd_quantity	cond	margin
0	0	2890	Buy	12	2.79	16.95	169172	12	

StockPosition

```
id (int): position id
code (str): contract id
direction (Action): action
    (Buy, Sell)
quantity (int): quantity
price (float): the average price
last_price (float): last price
pnl (float): unrealized profit
yd_quantity (int): yesterday
cond (StockOrderCond): Default Cash
    (Cash(現股), Netting(餘額交割), MarginTrading(融資), ShortSelling(融券), Emerging(興權))
margin_purchase_amount (int): margin_purchase_amount
collateral (int): collateral
short_sale_margin (int): short_sale_margin
interest (int): interest
```

Odd Stocks

The unit is the number of shares.

```
api.list_positions(
    api.stock_account,
    unit=sj.constant.Unit.Share
)
```

Out

```
[
    StockPosition(
        id=0,
        code='2890',
        direction=<Action.Buy: 'Buy'>,
        quantity=10000,
        price=10.1,
        last_price=12.0,
        pnl=1234.0,
        yd_quantity=10000,
        margin_purchase_amount=0,
        collateral=0,
        short_sale_margin=0,
        interest=0
    )
]
```

FUTURES AND OPTIONS

`account` is defaulted as a Stock account, and if you want to query the Futures or Options content, you need to bring in the `futopt_account`.

```
api.list_positions(api.futopt_account)
```



```
[  
    FuturePosition(  
        id=0,  
        code='TX201370J2',  
        direction=<Action.Buy: 'Buy'>,  
        quantity=3,  
        price=131.000,  
        last_price=126.0,  
        pnl=-750.0  
    )  
]
```

To DataFrame



```
positions = api.list_positions(api.futopt_account)  
df = pd.DataFrame(p._dict_ for p in positions)  
df
```



id	code	direction	quantity	price	last_price	pnl
0	TXFA3	Buy	4	14181	14375	155200

FuturePosition

```
id (int): position id  
code (str): contract id  
direction (Action): action  
    (Buy, Sell)  
quantity (int): quantity  
price (float): the average price  
last_price (float): last price  
pnl (float): unrealized profit
```

Position Detail

Using the result obtained from `list_positions`, bring the `id` into `detail_id` to query the details of that position.

STOCKS



```
api.list_position_detail?
```



Signature:
`api.list_position_detail(
 account: shioaji.account.Account = None,
 detail_id: int = 0,
 timeout: int = 5000,
 cb: Callable[[List[Union[shioaji.position.StockPositionDetail, shioaji.position.FuturePositionDetail]]], NoneType] = None,
) -> List[Union[shioaji.position.StockPositionDetail, shioaji.position.FuturePositionDetail]]`

Docstring:
query account of position detail

Args:
`account (:obj:Account):`
 choose the account from `listing` account (Default: stock account)
`detail_id (int):` the `id` is from `Position` object, `Position` is from `list_positions`



```
position_detail = api.list_position_detail(api.stock_account, 1)
position_detail
```



```
[  
    StockPositionDetail(  
        date='2023-02-22',  
        code='3558',  
        quantity=0,  
        price=1461.0,  
        last_price=1470.0,  
        dseq='WA371',  
        direction=Action.Buy: 'Buy',  
        pnl=9.0,  
        currency=<Currency.TWD: 'TWD'>,  
        fee=1.0  
    )  
)
```

To DataFrame



```
df = pd.DataFrame(pnl.__dict__ for pnl in position_detail)
df
```



date	code	quantity	price	last_price	direction	pnl	currency	fee
2023-02-22	3558	0	1461.0	WA371	Action.Buy	11.0	Currency.TWD	1.0



```
date (str): trade date
code (str): contract id
quantity (int): quantity
price (float): price
last_price (float): last price
dseq (str): detail seqno no
direction (Action): (Buy, Sell)
pnl (decimal): unrealized profit
currency (string): (NTD, USD, HKD, EUR, CAD, BAS)
fee (decimal): fee
cond (StockOrderCond): Default Cash
    (Cash, Netting, MarginTrading, ShortSelling, Emerging)
ex_dividends(int): ex-dividend amount
interest (int): interest
margintrading_amt(int): margin trading amount
collateral (int): collateral
```

FUTURES AND OPTIONS



```
position_detail = api.list_position_detail(api.fuopt_account, 0)
position_detail
```

```
[  
    FuturePositionDetail(  
        date='2023-02-14',  
        code='MXFC3',  
        quantity=1,  
        price=15611.0,  
        last_price=15541.0,  
        dseq='tA0n8',  
        direction=Action.Buy: 'Buy',  
        pnl=-3500.0,  
        currency<Currency.TWD: 'TWD',  
        entry_quantity=1  
    )  
]
```

To DataFrame

```
df = pd.DataFrame(pnl.__dict__ for pnl in position_detail)  
df
```

date	code	quantity	price	last_price	dseq	direction	pnl	curr
2023-02-14	MXFC3	1	15611.0	15541.0	tA0n8	Action.Buy	-3500.0	Currency.

```
code (str): contract id  
date (str): trade date  
quantity (int): quantity  
price (float): price  
last_price (float): last price  
dseq (str): detail seqno  
direction (Action): {Buy, Sell}  
pnl (float): unrealized profit  
currency (str): (NTD, USD, HKD, EUR, CAD, BAS)  
fee (float or int): fee  
entry_quantity(int): entry quantity
```

4.7.5 Profit Loss

You need to [login](#) first.

Profit Loss

The feature of `list_profit_loss` is used to query realized profit loss of account.



```
api.list_profit_loss?
```



Signature:

```
api.list_profit_loss(
    account: shioaji.account.Account = None,
    begin_date: str = '',
    end_date: str = '',
    unit: shioaji.constant.Unit = <Unit.Common: 'Common'>,
    timeout: int = 5000,
    cb: Callable[[List[shioaji.position.ProfitLoss]], NoneType] = None,
) -> List[shioaji.position.ProfitLoss]
```

Docstring:

```
query account of profit loss
```

Args:

```
account (:obj:`Account`):
    choice the account from listing account (Default: stock account)
begin_date (str): the start date of query profit loss (Default: today)
end_date (str): the end date of query profit loss (Default: today)
```

Enter the time interval you want to query. `begin_date` is the start time, and `end_date` is the end time. `unit` is the quantity unit, where `Common` represents whole shares and `Share` represents fractional shares.



```
profitloss = api.list_profit_loss(api.stock_account, '2020-05-05', '2020-05-30')
profitloss
```



```
[
    StockProfitLoss(
        id=0,
        code='2890',
        seqno='14816',
        dseq='1D111',
        quantity=1,
        price=10.1,
        pnl=1234.0,
        pr_ratio=0.1237,
        cond='Cash',
        date='2020-05-22'
    )
]
```

To DataFrame



```
df = pd.DataFrame(pnl.__dict__ for pnl in profitloss)
df
```



Out

id	code	cond	date	pnl	pr_ratio	price	quantity	seqno
0	2890	StockOrderCond.Cash	2020-05-22	1000.0	0.1237	10.1	1	14816



StockProfitLoss

```
id (int): use to find detail
code (str): contract id
seqno (str): seqno no
dseq (str): seqno no
quantity (int): quantity
price (float): price
pnl (float): profit and loss
pr_ratio (float): profit rate
cond (StockOrderCond): (Cash, Netting, MarginTrading, ShortSelling)
date (str): trade date
```



FutureProfitLoss

```
id (int): use to find detail
code (str): contract id
quantity (int): quantity
pnl (float): profit and loss
date (str): trade date
direction (Action): (Buy, Sell)
entry_price (int): entry price
cover_price (int): cover price
tax (int): tax
fee (int): transaction fee
```

Profit Loss Detail

The feature of `list_profit_loss_detail` is used to query profit loss detail of account. `unit` is the quantity unit, where `Common` represents whole shares and `Share` represents fractional shares.



```
api.list_profit_loss_detail?
```



Signature:
`api.list_profit_loss_detail(
 account: shioaji.account.Account = None,
 detail_id: int = 0,
 unit: shioaji.constant.Unit = <Unit.Common: 'Common'>,
 timeout: int = 5000,
 cb: Callable[[List[Union[shioaji.position.StockProfitDetail, shioaji.position.FutureProfitDetail]]], NoneType] = None,
) -> List[Union[shioaji.position.StockProfitDetail, shioaji.position.FutureProfitDetail]]`

Docstring:
query account of profit loss detail

Args:

```
account (:obj:Account):  
    choose the account from listing account (Default: stock account)  
detail_id (int): the id is from ProfitLoss object, ProfitLoss is from list_profit_loss
```



```
profitloss_detail = api.list_profit_loss_detail(api.stock_account, 2)  
profitloss_detail
```



```
[  
    StockProfitDetail(  
        date='2020-01-01',  
        code='2890',  
        quantity=1,  
        dseq='IX000',  
        fee=20,  
        tax=0,  
        currency='TWD',  
        price=10.8,  
        cost=10820,  
        rep_margintrading_amt=0,  
        rep_collateral=0,  
        rep_margin=0,  
        shortselling_fee=0,  
        ex_dividend_amt=0,  
        interest=0  
    )  
]
```

To DataFrame



```
df = pd.DataFrame(pnl.__dict__ for pnl in profitloss_detail)  
df
```



date	code	quantity	dseq	fee	tax	currency	price	cost	rep_margin
2020-01-01	2890	1	IX000	20	0	TWD	10.8	10820	



StockProfitDetail

```
date (str): trade date  
code (str): contract id  
quantity (int): quantity  
dseq (str): detail seqno no  
fee (int): fee  
tax (int): trading tax  
currency (str): {NTD, USD, HKD, EUR, CAD, BAS}  
price (float): price  
cost (int): cost of price  
rep_margintrading_amt (int): repay amount of margin trading  
rep_collateral (int): repay collateral  
rep_margin (int): repay margin  
shortselling_fee (int): fee of short selling  
ex_dividend_amt: ex-dividend amount  
interest (int): interest  
trade_type (TradeType): {Common, DayTrade}  
cond (StockOrderCond): {Cash, Netting, MarginTrading, ShortSelling}
```



FutureProfitDetail

```
date (str): trade date  
code (str): contract id  
quantity (int): quantity  
dseq (str): detail seqno no  
fee (int): fee  
tax (int): trading tax  
currency (str): {NTD, USD, HKD, EUR, CAD, BAS}  
direction (Action): {Buy, Sell}  
entry_date (str): entry date  
entry_price (int): entry price  
cover_price (int): cover price  
pnl (int): profit and loss
```

Profit Loss Summary

The feature of list_profit_loss_summary is used to query summary of profit loss for a period of time.



```
api.list_profit_loss_summary?
```



Signature:
`api.list_profit_loss_summary(
 account: shioaji.account.Account = None,
 begin_date: str = '',
 end_date: str = '',
 timeout: int = 5000,
 cb: Callable[[ProfitLossSummaryTotal], NoneType] = None,
) -> ProfitLossSummaryTotal`

Docstring:
`query summary profit loss of a period time`

Args:
`account (:obj:`Account`):
 choose the account from listing account (Default: stock account)
begin_date (str): the start date of query profit loss (Default: today)
end_date (str): the end date of query profit loss (Default: today)`

Enter the time interval you want to query. `begin_date` is the start time, and `end_date` is the end time.



```
profitloss_summary = api.list_profit_loss_summary(api.stock_account, '2020-05-05', '2020-05-30')  
profitloss_summary
```



```
ProfitLossSummaryTotal(  
    status=<FetchStatus.Fetched: 'Fetched'>,  
    profitloss_summary=[  
        StockProfitLossSummary(  
            code='2890',  
            quantity=2000,  
            entry_price=17,  
            cover_price=10,  
            pnl=-11585.0,  
            currency='NTD',  
            entry_cost=34550,  
            cover_cost=21600,  
            buy_cost=33112,  
            sell_cost=21527,  
            pr_ratio=-34.99  
        )  
    ],  
    total=ProfitLossTotal(  
        quantity=2000,  
        buy_cost=33112,  
        sell_cost=21527,  
        pnl=-11585.0,  
        pr_ratio=-34.99  
    )  
)
```

To DataFrame



```
df = pd.DataFrame(pnl._dict_ for pnl in profitloss_summary.profitloss_summary)  
df
```

Out

code	quantity	entry_price	cover_price	pnl	currency	entry_cost	cover_cost
2890	2000	17	10	-11585	NTD	34550	21600

StockProfitLossSummary

```
code (str): contract id
quantity (int): quantity
entry_price (int): price of entry
cover_price (int): price of cover
pnl (float): profit and loss
currency (str): currency
entry_cost (int): cost of entry
cover_cost (int): cost of cover
buy_cost (int): cost of buy
sell_cost (int): cost of sell
pr_ratio (float): profit rate
cond (StockOrderCond): {Cash, Netting, MarginTrading, ShortSelling}
```

FutureProfitLossSummary

```
code (str): contract id
quantity (int): quantity
entry_price (int): price of entry
cover_price (int): price of cover
pnl (float): profit and loss
currency (str): currency
direction (Action): {Buy, Sell}
tax (int): tax
fee (int): fee
```

4.7.6 Settlements

The feature of settlements is used to query settlements of stock account and you need to [login](#) first.

Settlements



```
api.settlements?
```



Signature:
`api.settlements(
 account: shioaji.account.Account = None,
 timeout: int = 5000,
 cb: Callable[[List[shioaji.position.SettlementV1]]], NoneType] = None,
) -> List[shioaji.position.SettlementV1]`
Docstring: query stock account of settlements



```
settlements = api.settlements(api.stock_account)  
settlements
```



```
[  
    SettlementV1(date=datetime.date(2022, 10, 13), amount=0.0, T=0),  
    SettlementV1(date=datetime.date(2022, 10, 14), amount=0.0, T=1),  
    SettlementV1(date=datetime.date(2022, 10, 17), amount=0.0, T=2)  
, ]
```

To DataFrame



```
df = pd.DataFrame([s._dict_ for s in settlements]).set_index("T")  
df
```



T	date	amount
0	2022-10-13	0
1	2022-10-14	0
2	2022-10-17	0



SettlementV1

```
date (datetime.date): date of Tday  
amount (float): settlement amount  
T (int): Tday
```

4.8 Simulation Mode

Users can first familiarize themselves with the API services in the simulation mode, which can avoid the loss of property caused by operational errors in the production environment.

Simulation

```
import shioaji as sj
api = sj.Shioaji(simulation=True)
```

4.8.1 Available APIs

Data

1. quote.subscribe
2. quote.unsubscribe
3. ticks
4. kbars
5. snapshots
6. short_stock_sources
7. credit_enquires
8. scanners

Order

1. place_order
2. update_order
3. cancel_order
4. update_status
5. list_trades

Account

1. list_positions
2. list_profit_loss

4.9 Advanced Guide

4.9.1 Quote-Binding Mode

Shioaji provides quote-binding mode which you can store tick/bidask in queue, push them to redis, or submit a stop order inside quote callback function. We show examples to make you more understand how to use quote-binding mode.

Examples

BIND QUOTE TO MESSAGE QUEUE

iii: pythonic way by using decorator

```
from collections import defaultdict, deque
from shioaji import TickFOPv1, Exchange

# set context
msg_queue = defaultdict(deque)
api.set_context(msg_queue)

# In order to use context, set bind=True
@api.on_tick_fop_v1(bind=True)
def quote_callback(self, exchange:Exchange, tick:TickFOPv1):
    # append quote to message queue
    self[tick.code].append(tick)

# subscribe
api.quote.subscribe(
    api.Contracts.Futures.TXF['TXF202107'],
    quote_type = sj.constant.QuoteType.Tick,
    version = sj.constant.QuoteVersion.v1
)
```

iii: traditional way

```
def quote_callback(self, exchange:Exchange, tick:TickFOPv1):
    # append tick to context
    self[tick.code].append(tick)

# In order to use context, set bind=True
api.quote.set_on_tick_fop_v1_callback(quote_callback, bind=True)
```

Out

```
# after subscribe and wait for a few seconds ...
# print(msg_queue)
defaultdict(deque,
{
    'TXFG1': [
        Tick(code='TXFG1', datetime=datetime.datetime(2021, 7, 5, 10, 0, 21, 220000), open=Decimal('17755'), underlying_price=Decimal('17851.88'), bid_side_total_vol=34824,
        ask_side_total_vol=36212, avg_price=Decimal('17837.053112'), close=Decimal('17833'), high=Decimal('17900'), low=Decimal('17742'), amount=Decimal('17833'), total_amount=Decimal('981323314'),
        volume=1, total_volume=55016, tick_type=1, chg_type=2, price_chg=Decimal('184'), pct_chg=Decimal('1.042552'), simtrade=0),
        Tick(code='TXFG1', datetime=datetime.datetime(2021, 7, 5, 10, 0, 21, 781000), open=Decimal('17755'), underlying_price=Decimal('17851.88'), bid_side_total_vol=34825,
        ask_side_total_vol=36213, avg_price=Decimal('17837.053056'), close=Decimal('17834'), high=Decimal('17900'), low=Decimal('17742'), amount=Decimal('17834'), total_amount=Decimal('981341148'),
        volume=1, total_volume=55017, tick_type=1, chg_type=2, price_chg=Decimal('185'), pct_chg=Decimal('1.048218'), simtrade=0)
    ]
})
```

PUSH QUOTE TO REDIS STREAM

Before start, please install [redis](#) first. Below example shows how to push quote messages to redis stream.



```

import redis
import json
from shioaji import TickFOPv1, Exchange

# redis setting
r = redis.Redis(host='localhost', port=6379, db=0, decode_responses=True)

# set up context
api.set_context(r)

# In order to use context, set bind=True
@api.on_tick_fop_v1(bind=True)
def quote_callback(self, exchange:Exchange, tick:TickFOPv1):
    # push them to redis stream
    channel = 'Q:' + tick.code # ='Q:TXFG1' in this example
    self.xadd(channel, {'tick':json.dumps(tick.to_dict(raw=True))})

```



```

# after subscribe and wait for a few seconds ...
# r.xread({'Q:TXFG1':'0-0'})
[
    ['Q:TXFG1',
        [
            ('1625454940107-0',
                {'tick':
                    {'code': 'TXFG1', 'datetime': '2021-07-05T11:15:49.066000', 'open': '17755', 'underlying_price': '17904.03', 'bid_side_total_vol': 49698, 'ask_side_total_vol': 51490,
                    'avg_price': '17851.312322', 'close': '17889', 'high': '17918', 'low': '17742', 'amount': '268335', 'total_amount': '1399310819', 'volume': 15, 'total_volume': 78387, 'tick_type': 2, 'chg_type': 2, 'price_chg': '240', 'pct_chg': '1.35985', 'simtrade': 0}
                }
            ),
            ('1625454941854-0',
                {'tick':
                    {'code': 'TXFG1', 'datetime': '2021-07-05T11:15:50.815000', 'open': '17755', 'underlying_price': '17902.58', 'bid_side_total_vol': 49702, 'ask_side_total_vol': 51478,
                    'avg_price': '17851.313258', 'close': '17888', 'high': '17918', 'low': '17742', 'amount': '35776', 'total_amount': '1399346595', 'volume': 2, 'total_volume': 78389, 'tick_type': 2, 'chg_type': 2, 'price_chg': '239', 'pct_chg': '1.354184', 'simtrade': 0}
                }
            )
        ]
    ]
]

# parse redis stream
# [json.loads(x[-1]['tick']) for x in r.xread({'Q:TXFG1':'0-0'})[0][-1]]
[
    {
        'code': 'TXFG1',
        'datetime': '2021-07-05T11:15:49.066000',
        'open': '17755',
        'underlying_price': '17904.03',
        'bid_side_total_vol': 49698,
        'ask_side_total_vol': 51490,
        'avg_price': '17851.312322',
        'close': '17889',
        'high': '17918',
        'low': '17742',
        'amount': '268335',
        'total_amount': '1399310819',
        'volume': 15,
        'total_volume': 78387,
        'tick_type': 2,
        'chg_type': 2,
        'price_chg': '240',
        'pct_chg': '1.35985',
        'simtrade': 0
    },
    {
        'code': 'TXFG1',
        'datetime': '2021-07-05T11:15:50.815000',
        'open': '17755',
        'underlying_price': '17902.58',
        'bid_side_total_vol': 49702,
        'ask_side_total_vol': 51478,
        'avg_price': '17851.313258',
        'close': '17888',
        'high': '17918',
        'low': '17742',
        'amount': '35776',
        'total_amount': '1399346595',
        'volume': 2,
        'total_volume': 78389,
        'tick_type': 2,
        'chg_type': 2,
        'price_chg': '239',
        'pct_chg': '1.354184',
        'simtrade': 0
    }
]

```

STOP ORDER IMPLEMENTATION

A [stop order](#) is an order to buy or sell a security when its price moves past a particular point, ensuring a higher probability of achieving a predetermined entry or exit price, limiting the investor's loss, or locking in a profit. Once the price crosses the predefined entry/exit point, the stop order becomes a market order.

We provide an example of stop order below. **Please use at your own risk.**

Example: stop order

```

import time
from typing import Union

import shioaji as sj

class StopOrderExecutor:
    def __init__(self, api: sj.Shioaji) -> None:
        self.api = api
        self._stop_orders = {}

    def on_quote(
        self, quote: Union[sj.BidAskFOPv1, sj.BidAskSTKv1, sj.TickFOPv1, sj.TickSTKv1]
    ) -> None:
        code = quote.code
        if code in self._stop_orders:
            for stop_order in self._stop_orders[code]:
                if stop_order['executed']:
                    continue
                if hasattr(quote, "ask_price"):
                    price = 0.5 * float(
                        quote.bid_price[0] + quote.ask_price[0]
                    ) # mid price
                else:
                    price = float(quote.close) # Tick

                is_execute = False
                if stop_order["stop_price"] >= stop_order["ref_price"]:
                    if price >= stop_order["stop_price"]:
                        is_execute = True
                elif stop_order["stop_price"] < stop_order["ref_price"]:
                    if price <= stop_order["stop_price"]:
                        is_execute = True

                if is_execute:
                    self.api.place_order(stop_order["contract"], stop_order["pending_order"])
                    stop_order['executed'] = True
                    stop_order['ts_executed'] = time.time()
                    print(f"execute stop order: {stop_order}")
                else:
                    self._stop_orders[code]

    def add_stop_order(
        self,
        contract: sj.contracts.Contract,
        stop_price: float,
        order: sj.order.Order,
    ) -> None:
        code = contract.code
        snap = self.api.snapshots([contract])[0]
        # use mid price as current price to avoid illiquidity
        ref_price = 0.5 * (snap.buy_price + snap.sell_price)
        stop_order = {
            "code": contract.code,
            "stop_price": stop_price,
            "ref_price": ref_price,
            "contract": contract,
            "pending_order": order,
            "ts_create": time.time(),
            "executed": False,
            "ts_executed": 0.0
        }

        if code not in self._stop_orders:
            self._stop_orders[code] = []
        self._stop_orders[code].append(stop_order)
        print(f"add stop order: {stop_order}")

    def get_stop_orders(self) -> dict:
        return self._stop_orders

    def cancel_stop_order_by_code(self, code: str) -> None:
        if code in self._stop_orders:
            _ = self._stop_orders.pop(code)

    def cancel_stop_order(self, stop_order: dict) -> None:
        code = stop_order["code"]
        if code in self._stop_orders:
            self._stop_orders[code].remove(stop_order)
            if len(self._stop_orders[code]) == 0:
                self._stop_orders.pop(code)

    def cancel_all_stop_orders(self) -> None:
        self._stop_orders.clear()

```

- We use mid price of snapshots as our reference price to differentiate the direction of stop order.

Basically, order will be pending at your computer. The order won't be submitted to exchange until close/mid price hit the stop price. Below example shows how to submit a stop-limit order.

Set up a stop order

```
# shioaji order
contract = api.Contracts.Futures.TXF['TXF202301']
order = api.Order(
    action='Buy',
    price=14800,
    quantity=1,
    price_type='LMT',
    order_type='ROD',
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
# Stop Order Executor
soe = StopOrderExecutor(api)
soe.add_stop_order(contract=contract, stop_price=14805, order=order)
```

Cut

```
add stop order: {
    'code': 'TXFA3',
    'stop_price': 14805,
    'ref_price': 14790,
    'contract': Future(
        code='TXFA3',
        symbol='TXF202301',
        name='臺股期貨01',
        category='TXF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16241.0,
        limit_down=13289.0,
        reference=14765.0,
        update_date='2023/01/10'
    ),
    'pending_order': Order(
        action=<Action.Buy: 'Buy'>,
        price=14800,
        quantity=1,
        account=FutureAccount(person_id='A123456789', broker_id='F002000', account_id='1234567', signed=True, username='PAIUSER'),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    'ts_create': 1673329115.1056178,
    'executed': False,
    'ts_executed': 0.0
}
```

- Stop-Market Order: `price_type = 'MKT'`

Finally, we bind `StopOrderExecutor` to quote callback function. Note that you have to subscribe quote, so that stop order will be executed.

Set up context and callback function

```
from shioaji import TickFOPv1, Exchange
# set up context
api.set_context(soe)

# In order to use context, set bind=True
@api.on_tick_fop_v1(bind=True)
def quote_callback(self, exchange:Exchange, tick:TickFOPv1):
    # pass tick object to Stop Order Executor
    self.on_quote(tick)

# subscribe
api.quote.subscribe(
    contract,
    quote_type = sj.constant.QuoteType.Tick,
    version = sj.constant.QuoteVersion.v1
)
```

 **Out: Once close/mid price hit stop price**

```
execute stop order: {
    'code': 'TXFA3',
    'stop_price': 14805,
    'ref_price': 14790,
    'contract': Future(
        code='TXFA3',
        symbol='TXF202301',
        name='臺股期貨01',
        category='TXX',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        limit_up=16241.0,
        limit_down=13289.0,
        reference=14765.0,
        update_date='2023/01/10'
    ),
    'pending_order': Order(
        action=<Action.Buy: 'Buy'>,
        price=14800,
        quantity=1,
        account=FutureAccount(person_id='A123456789', broker_id='F002000', account_id='1234567', signed=True, username='PAIUSER'),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    'ts_create': 1673329115.1056178,
    'executed': True,
    'ts_executed': 1673329161.3224185
}
```

4.9.2 Non-blocking Mode

Blocking is a pattern where a function must wait for something to complete. Every function is waiting, whether it is doing I/O or doing CPU tasks. For example, if the function tries to get data from the database, it needs to stop and wait for the return result, and then continue processing the next task after receiving the return result. In contrast, non-blocking mode does not wait for operations to complete. Non-blocking mode is useful if you are trying to send batch operation in a short period of time. We provide the following examples to give you a better understanding of the difference.

To switch blocking/non-blocking mode use parameter `timeout`. Set the API parameter `timeout` to `0` for non-blocking mode. The default value of `timeout` is 5000 (milliseconds), which means the function waits for up to 5 seconds.

NON-BLOCKING PLACE ORDER

Set `timeout = 0` in `place_order` function.



```
contract = api.Contracts.Futures.TXF['TXF202301']
order = api.Order(
    action=sj.constant.Action.Sell,
    price=14000,
    quantity=1,
    price_type=sj.constant.FuturePriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FutureOCType.Auto,
    account=api.futopt_account
)
trade = api.place_order(contract, order, timeout=0)
trade
```



```
Trade(
    contract=Future(
        code='TXFA3',
        symbol='TXF202301',
        name='臺股期貨01',
        category='TXF',
        delivery_month='202301',
        delivery_date='2023/01/30',
        underlying_kind='I',
        unit=1,
        Limit_up=16241.0,
        limit_down=13289.0,
        reference=14765.0,
        update_date='2023/01/10'
    ),
    order=Order(
        action=<Action.Sell: 'Sell'>,
        price=14000,
        quantity=1,
        account=FutureAccount(
            person_id='F123456789',
            broker_id='F002000',
            account_id='1234567',
            signed=True,
            username='PAPIUSER'
        ),
        price_type=<StockPriceType.LMT: 'LMT'>,
        order_type=<OrderType.ROD: 'ROD'>
    ),
    status=OrderStatus(status=<Status.Inactive: 'Inactive'>)
)
```

The `Trade` object obtained in non-blocking mode will lack some information because the order is still in transit and has not been sent to the exchange. There are no `id` and `seqno` in the `Order` object, `id`, `status_code`, `order_datetime` and `deals` are missing in the `OrderStatus` object, and `status` is displayed as `Inactive`. In the non-blocking mode, there are two ways to obtain the above-mentioned information: `order event callback` and `non-blocking place order callback`.

Order event callback

```
OrderState_FuturesOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': 'd616839',
        'seqno': '500009',
        'ordno': '000009',
        'action': 'Sell',
        'price': 14000,
        'quantity': 1,
        'order_type': 'ROD',
        'price_type': 'LNT',
        'oc_type': 'Auto',
        'custom_field': ''
    },
    'status': {
        'id': 'd616839',
        'exchange_ts': 1673334371.492948,
        'order_quantity': 1,
        'modified_price': 0,
        'cancel_quantity': 0,
        'web_id': 'Z'
    },
    'contract': {
        'security_type': 'FUT',
        'exchange': 'TAIFEX',
        'code': 'TXFA3'
    }
}
```

Non-blocking place order callback

```
from shioaji.order import Trade

def non_blocking_cb(trade:Trade):
    print('__my_callback__')
    print(trade)

trade = api.place_order(
    contract,
    order,
    timeout=0,
    cb=non_blocking_cb # only work in non-blocking mode
)
```

Out: place order callback

```
__my_callback__
contract=Future(
    code='TXFA3',
    symbol='TXF202301',
    name='臺股期貨01',
    category='TXF',
    delivery_month='202301',
    delivery_date='2023/01/30',
    underlying_kind='I',
    unit=1,
    limit_up=16241.0,
    limit_down=13289.0,
    reference=14765.0,
    update_date='2023/01/10'
),
order=Order(
    action=Action.Sell: 'Sell'>,
    price=14000,
    quantity=1,
    id='40fd85d6',
    seqno='958433',
    ordno='ky0lg',
    account=FutureAccount(
        person_id='F123456789',
        broker_id='F002000',
        account_id='1234567',
        signed=True,
        username='PAP1USER'
    ),
    price_type=<StockPriceType.LMT: 'LMT'>,
    order_type=<OrderType.ROD: 'ROD'>
),
status=OrderStatus(
    id='40fd85d6',
    status=Status.Submitted: 'Submitted'>,
    status_code=' ',
    order_datetime=datetime(2023, 01, 10, 15, 14, 32),
    deals=[]
)
```

COMPARE BOTH MODES

In non-wait mode, executing `place_order` takes about 0.01 seconds, which is 12 times faster than the execution time in blocking mode. Although it is more efficient to place order in the non-blocking mode, the order will not take effect until the exchange receives the order.

Contract and order

```
contract = api.Contracts.Futures.TXF['TXF202301']
order = api.Order(
    action='Sell',
    price=14000,
    quantity=1,
    price_type='LMT',
    order_type='ROD',
    octype=sj.constant.FuturesOctype.Auto,
    account=api.futopt_account
)
```

Blocking

```
start_time = time.time()
api.place_order(contract, order) # block and wait for the order response
print(time.time() - start_time)
# 0.136578369140625 <- may be different
```

Non-Blocking

```
start_time = time.time()
api.place_order(contract, order, timeout=0) # non-block, the order is in transmission (inactive).
print(time.time() - start_time)
# 0.011670351028442383 <- may be different
```

Non-Blocking mode Supported Function

```
- place_order  
- update_order  
- cancel_order  
- update_status  
- list_positions  
- list_position_detail  
- list_profit_loss  
- list_profit_loss_detail  
- list_profit_loss_summary  
- settlements  
- margin  
- ticks  
- kbars
```

4.9.3 Touch Price Order

Touch Price Order

Here is a simple example that how to build your price monitor and when price touches the condition will place the order.

```
from pydantic import BaseModel

class TouchOrderCond(BaseModel):
    contract: Contract
    order: Order
    order: Order
    touch_price: float

class TouchOrder:
    def __init__(self, api: sj.Shioaji, condition: TouchOrderCond):
        self.flag = False
        self.api = api
        self.order = condition.order
        self.contract = condition.contract
        self.touch_price = condition.touch_price
        self.api.quote.subscribe(self.contract)
        self.api.quote.set_quote_callback(self.touch)

    def touch(self, topic, quote):
        price = quote["Close"][0]
        if price == self.touch_price and not self.flag:
            self.flag = True
            self.api.place_order(self.contract, self.order)
            self.api.quote.unsubscribe(self.contract)
```

Complete TouchPrice Order Extention can be found [here](#).

4.9.4 Quote Manager Basic

the whole project code can be found in [sj-trading](#), the whole example jupyter notebook can be found in [quote_manager_usage](#).

this project is created by using `uv`, if you are not familiar with how to use `uv` to create a project and manage dependencies, it is recommended to learn from the [environment setup](#) chapter.

before start writing the quote manager, we will use the Polars package to process the quote data, so we need to add it to the project dependencies, at the same time, this tutorial will have an example of how to use Polars to quickly calculate technical indicators for multiple commodities, so we also need to add the `polars_talib` package to the project dependencies.

add Polars dependencies

```
uv add polars polars_talib
```

if you are not familiar with Polars, you can refer to the [Polars official documentation](#) to learn how to use it.

`polars_talib` is a Polars extension package that provides the complete functionality of the ta-lib library in the polars expression version, allowing us to easily calculate technical indicators using Polars. It is developed by the shioaji author, and detailed usage can be found in [polars_ta_extension](#).

Polars is an efficient DataFrame package that is suitable for processing large amounts of data and can use multiple cores without any additional configuration. In this example, we can see how to use the Shioaji quote manager to obtain quote data, and use Polars for parallel computation, while converting the ticks of the commodity into K lines, and performing parallel multi-commodity technical indicator calculations.

add quote.py

add `quote.py` file in `src/sj_trading/`, and add the following code

```
import shioaji as sj
from typing import List

class QuoteManager:
    def __init__(self, api: sj.Shioaji):
        self.api = api
        self.api.quote.set_on_tick_stk_v1_callback(self.on_stk_v1_tick_handler)
        self.api.quote.set_on_tick_fop_v1_callback(self.on_fop_v1_tick_handler)
        self.ticks_stk_v1: List[sj.TickSTKv1] = []
        self.ticks_fop_v1: List[sj.TickFOPv1] = []

    def on_stk_v1_tick_handler(self, _exchange: sj.Exchange, tick: sj.TickSTKv1):
        self.ticks_stk_v1.append(tick)

    def on_fop_v1_tick_handler(self, _exchange: sj.Exchange, tick: sj.TickFOPv1):
        self.ticks_fop_v1.append(tick)
```

this part is relatively simple, let the handle func of receiving the quote data do as little as possible, we define a `QuoteManager` class, and register two callback functions in the initialization, respectively `on_stk_v1_tick_handler` and `on_fop_v1_tick_handler`, these two functions will be called when receiving the quote data, and the quote data will be stored in `ticks_stk_v1` and `ticks_fop_v1`.

add QuoteManager subscribe and unsubscribe methods

```
def __init__(self, api: sj.Shioaji):
    # skip
    self.subscribed_stk_tick: Set[str] = set()

def subscribe_stk_tick(self, codes: List[str], recover: bool = False):
    for code in codes:
        contract = self.api.Contracts.Stocks[code]
        if contract is not None and code not in self.subscribed_stk_tick:
            self.api.quote.subscribe(contract, "tick")
            self.subscribed_stk_tick.add(code)

def unsubscribe_stk_tick(self, codes: List[str]):
    for code in codes:
        contract = self.api.Contracts.Stocks[code]
        if contract is not None and code in self.subscribed_stk_tick:
            self.api.quote.unsubscribe(contract, "tick")
            self.subscribed_stk_tick.remove(code)

def unsubscribe_all_stk_tick(self):
    for code in self.subscribed_stk_tick:
        contract = self.api.Contracts.Stocks[code]
        if contract is not None:
            self.api.quote.unsubscribe(contract, "tick")
    self.subscribed_stk_tick.clear()
```

in the above code, we have added the `subscribe_stk_tick` method, this method will add the commodity codes in the incoming commodity code list to the `subscribed_stk_tick`, and call the `subscribe` method of Shioaji to subscribe to the market, `subscribed_stk_tick` is a `Set`, used to store the commodity codes that have been subscribed to avoid duplicate subscriptions and facilitate subsequent unsubscribing all subscribed commodities.

add QuoteManager get ticks method

```
def __init__(self, api: sj.Shioaji):
    # skip
    self.df_stk: pl.DataFrame = pl.DataFrame(
        [],
        schema=[
            ("datetime", pl.Datetime),
            ("code", pl.Utf8),
            ("price", pl.Float64),
            ("volume", pl.Int64),
            ("tick_type", pl.Int8),
        ],
    )

def get_df_stk(self) -> pl.DataFrame:
    poped_ticks, self.ticks_stk_v1 = self.ticks_stk_v1, []
    if poped_ticks:
        df = pl.DataFrame([tick.to_dict() for tick in poped_ticks]).select(
            pl.col("datetime", "code"),
            pl.col("close").cast(pl.Float64).alias("price"),
            pl.col("volume").cast(pl.Int64),
            pl.col("tick_type").cast(pl.Int8),
        )
        self.df_stk = self.df_stk.vstack(df)
    return self.df_stk
```

in `__init__` we define a `df_stk` Polars DataFrame, used to store all subscribed stock tick data, `get_df_stk` method will convert the `ticks_stk_v1` list to a Polars DataFrame, and return it, at this point, we have already got a DataFrame that can be used to calculate technical indicators.

`df_stk`

add QuoteManager get kbar method

```
def get_df_stk_kbar(
    self, unit: str = "1m", exprs: List[pl.Expr] = []
) -> pl.DataFrame:
    df = self.get_df_stk()
    df = df.groupby(
        pl.col("datetime").dt.truncate(unit),
        pl.col("code"),
        maintain_order=True,
    ).agg(
        pl.col("price").first().alias("open"),
        pl.col("price").max().alias("high"),
        pl.col("price").min().alias("low"),
        pl.col("price").last().alias("close"),
        pl.col("volume").sum().alias("volume"),
    )
    if exprs:
        df = df.with_columns(exprs)
    return df
```

in `get_df_stk_kbar` method, we will use `get_df_stk` method to get the Ticks DataFrame and then group the data by truncated `datetime` and `code`, and then aggregate the data in each group to get the K line data, finally, we will return the K line DataFrame. Here we remain the `exprs` parameter, allowing users to pass in additional expressions for more calculations.

In this part, we use `1m` to represent 1 minute, if you want to get 5 minutes K line, you can change the unit to `5m`, 1 hour K line can be changed to `1h`, if you want more different units, you can refer to the [truncate API documentation](#).

add custom technical indicator calculation method

```
import polars as pl
import polars_ta as plt

quote_manager.get_df_stk_kbar("5m", [
    pl.col("close").ta.ema(5).over("code").fill_nan(None).alias("ema5"),
    pltta.macd(pl.col("close"), 12, 26, 9).over("code").struct.field("macd").fill_nan(None),
])
```

in this part, we use `polars_ta` to calculate technical indicators and add them to the K line data, here we calculate `ema` and `macd` two indicators, more indicators can refer to [polars_ta_extension supported indicators list](#).

in this `polars_ta` expression, we use `over("code")` to group the data by commodity code for independent calculation of each commodity, so even if all commodities are in the same DataFrame, the calculation results are independent of each other, and this `over` partition is automatically parallel computing, so even if there are a large number of commodities, the calculation can be very fast and then using `alias` to set the field name of the calculation result as `ema5`, in the `macd` indicator, the return is a struct with multiple fields, and this part gets the `macd` field.

because this part only passes in expressions and is very lightweight, you can pass in any expressions you need according to your needs, and you can also make your own indicators using polars expression, this part just provides an interface for calculation and a simple usage example.

add QuoteManager backfill missed ticks method

```
def fetch_ticks(self, contract: BaseContract) -> pl.DataFrame:
    code = contract.code
    ticks = self.api.ticks(contract)
    df = pl.DataFrame(ticks.dict()).select(
        pl.from_epoch("ts", time_unit="ns").dt.cast_time_unit("us").alias("datetime"),
        pl.lit(code).alias("code"),
        pl.col("close").alias("price"),
        pl.col("volume").cast(pl.Int64),
        pl.col("tick_type").cast(pl.Int8),
    )
    return df

def subscribe_stk_tick(self, codes: List[str], recover: bool = False):
    for code in codes:
        # skip
        if recover:
            df = self.fetch_ticks(contract)
            if not df.is_empty():
                code_ticks = [t for t in self.ticks_stk_v1 if t.code == code]
                if code_ticks:
                    t_first = code_ticks[0].datetime
                    df = df.filter(pl.col("datetime") < t_first)
                    self.df_stk = self.df_stk.vstack(df)
                else:
                    self.df_stk = self.df_stk.vstack(df)
```

in `subscribe_stk_tick` method, we will check if the `recover` parameter is true, if it is, we will call `fetch_ticks` method to get the historical ticks data, and then use `filter` method to filter out the ticks data that have been received, and use `vstack` method to add the historical ticks data to the `df_stk` DataFrame.

In above we have completed a quote manager that can subscribe to market data, backfill missed ticks, and calculate technical indicators. Next, we will integrate all the code and use it in a jupyter lab environment.

The complete QuoteManager can be found in [quote.py](#).

The complete example jupyter notebook can be found in [quote_manager_usage](#).

5. Upgrading to 1.0

Version 1.0 is a major release. This document assist users migrating to version 1.0.

5.1 Shioaji

Remove argument `backend`



```
import shioaji as sj
sj.Shioaji?
```

**version>=1.0 version<1.0**

```
Init signature:
sj.Shioaji(
    simulation: bool = False,
    proxies: Dict[str, str] = {},
    currency: str = 'NTD',
)
Docstring:
shioaji api

Functions:
    login
    logout
    activate_ca
    list_accounts
    set_default_account
    get_account_margin
    get_account_openposition
    get_account_settle_profitloss
    get_stock_account_funds
    get_stock_account_unreal_profitloss
    get_stock_account_real_profitloss
    place_order
    update_order
    update_status
    list_trades

Objects:
    Quote
    Contracts
    Order

Init docstring:
initialize Shioaji to start trading

Args:
    simulation (bool):
        - False: to trading on real market (just use your Sinopac account to start trading)
        - True: become simulation account(need to contract as to open simulation account)
    proxies (dict):
        specific the proxies of your https
        ex: {'https': 'your-proxy-url'}
    currency (str):
        {NTX, USX, NTD, USD, HKD, EUR, JPY, GBP}
        set the default currency for display
```

```
Init signature:
sj.Shioaji(
    backend: str = 'http',
    simulation: bool = False,
    proxies: Dict[str, str] = {},
    currency: str = 'NTD',
)
Docstring:
shioaji api
```

```
Functions:
    login
    activate_ca
    list_accounts
    set_default_account
    get_account_margin
    get_account_openposition
    get_account_settle_profitloss
    get_stock_account_funds
    get_stock_account_unreal_profitloss
    get_stock_account_real_profitloss
    place_order
    update_order
    update_status
    list_trades
```

```
Objects:
    Quote
    Contracts
    Order

Init docstring:
initialize Shioaji to start trading

Args:
    backend (str):
        {http, socket}
        use http or socket as backend currently only support http, async socket backend coming soon.
    simulation (bool):
        - False: to trading on real market (just use your Sinopac account to start trading)
        - True: become simulation account(need to contract as to open simulation account)
    proxies (dict):
        specific the proxies of your https
        ex: {'https': 'your-proxy-url'}
    currency (str):
        {NTX, USX, NTD, USD, HKD, EUR, JPY, GBP}
        set the default currency for display
```

5.2 Login

Please update your login parameters from `person_id` and `passwd` to `api_key` and `secret_key` in order to use version 1.0. You can apply for an `api_key` on the [Token](#) page.



version>=1.0 version<1.0

```
import shioaji as sj
api = sj.Shioaji()
api.login(
    api_key="YOUR_API_KEY",
    secret_key="YOUR_SECRET_KEY"
)

import shioaji as sj
api = sj.Shioaji()
api.login(
    person_id="YOUR_PERSON_ID",
    passwd="YOUR_PASSWORD",
)
```



```
[ FutureAccount(person_id='', broker_id='', account_id='', signed=True, username=''),
  StockAccount(person_id='', broker_id='', account_id='', signed=True, username='')
]
```

5.3 Stock Order

Rename `TFTStockOrder` to `StockOrder`

StockOrder

verion>=1.0 verion<1.0

```
>> sj.order.StockOrder?

Init signature:
sj.order.StockOrder(
    *,
    action: shioaji.constant.Action,
    price: Union[pydantic.types.StrictInt, float],
    quantity: shioaji.order.ConstrainedIntValue,
    id: str = '',
    seqno: str = '',
    ordno: str = '',
    account: shioaji.account.Account = None,
    custom_field: shioaji.order.ConstrainedStrValue = '',
    ca: str = '',
    price_type: shioaji.constant.StockPriceType,
    order_type: shioaji.constant.OrderType,
    order_lot: shioaji.constant.StockOrderLot = <StockOrderLot.Common: 'Common'>,
    order_cond: shioaji.constant.StockOrderCond = <StockOrderCond.Cash: 'Cash'>,
    daytrade_short: bool = False,
) -> None

>> sj.order.TFTStockOrder?

Init signature:
sj.order.TFTStockOrder(
    *,
    action: shioaji.constant.Action,
    price: Union[pydantic.types.StrictInt, float],
    quantity: shioaji.order.ConstrainedIntValue,
    id: str = '',
    seqno: str = '',
    ordno: str = '',
    account: shioaji.account.Account = None,
    custom_field: shioaji.order.ConstrainedStrValue = '',
    ca: str = '',
    price_type: shioaji.constant.TFTStockPriceType,
    order_type: shioaji.constant.TFTOrderType,
    order_lot: shioaji.constant.TFTStockOrderLot = <TFTStockOrderLot.Common: 'Common'>,
    order_cond: shioaji.constant.StockOrderCond = <StockOrderCond.Cash: 'Cash'>,
    first_sell: shioaji.constant.StockFirstSell = <StockFirstSell.No: 'false'>,
) -> None
```

5.3.1 Order

Rename

- TFTStockPriceType to StockPriceType
- TFTOrderType to OrderType
- TFTStockOrderLot to StockOrderLot
- first_sell to daytrade_short, and type changed to Bool.

Order

version>=1.0 version<1.0

```
order = api.Order(  
    price=12,  
    quantity=1,  
    action=sj.constant.Action.SELL,  
    price_type=sj.constant.StockPriceType.LMT,  
    order_type=sj.constant.OrderType.ROD,  
    order_lot=sj.constant.StockOrderLot.Common,  
    daytrade_short=True,  
    custom_field="test",  
    account=api.stock_account  
)  
  
order = api.Order(  
    price=12,  
    quantity=1,  
    action=sj.constant.Action.SELL,  
    price_type=sj.constant.TFTStockPriceType.LMT,  
    order_type=sj.constant.TFTOrderType.ROD,  
    order_lot=sj.constant.TFTStockOrderLot.Common,  
    first_sell=sj.constant.StockFirstSell.Yes,  
    custom_field="test",  
    account=api.stock_account  
)
```

5.3.2 Order Callback

Rename TFTOrder to StockOrder

Order Callback

version>=1.0 version<1.0

```
OrderState.StockOrder {
  'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
  },
  'order': {
    'id': 'c21b876d',
    'seqno': '429832',
    'ordno': 'W2892',
    'action': 'Buy',
    'price': 12.0,
    'quantity': 10,
    'order_cond': 'Cash',
    'order_lot': 'Common',
    'custom_field': 'test',
    'order_type': 'ROD',
    'price_type': 'LNT'
  },
  'status': {
    'id': 'c21b876d',
    'exchange_ts': 1583828972,
    'modified_price': 0,
    'cancel_quantity': 0,
    'web_id': '137'
  },
  'contract': {
    'security_type': 'STK',
    'exchange': 'TSE',
    'code': '2890',
    'symbol': '',
    'name': '',
    'currency': 'TWD'
  }
}

OrderState.TFTOrder {
  'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
  },
  'order': {
    'id': 'c21b876d',
    'seqno': '429832',
    'ordno': 'W2892',
    'action': 'Buy',
    'price': 12.0,
    'quantity': 10,
    'order_cond': 'Cash',
    'order_lot': 'Common',
    'custom_field': 'test',
    'order_type': 'ROD',
    'price_type': 'LNT'
  },
  'status': {
    'id': 'c21b876d',
    'exchange_ts': 1583828972,
    'modified_price': 0,
    'cancel_quantity': 0,
    'web_id': '137'
  },
  'contract': {
    'security_type': 'STK',
    'exchange': 'TSE',
    'code': '2890',
    'symbol': '',
    'name': '',
    'currency': 'TWD'
  }
}
```

5.3.3 Deal Callback

Rename TFTDeal to StockDeal

Deal Callback

version>=1.0 version<1.0

```
OrderState.StockDeal {
    'trade_id': '12ab3456',
    'exchange_seq': '123456',
    'broker_id': 'your_broker_id',
    'account_id': 'your_account_id',
    'action': <Action.Buy: 'Buy'>,
    'code': '2890',
    'order_cond': <StockOrderCond.Cash: 'Cash'>,
    'order_lot': <TFTStockOrderLot.Common: 'Common'>,
    'price': 12,
    'quantity': 10,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1583828972
}

OrderState.TFTDeal {
    'trade_id': '12ab3456',
    'exchange_seq': '123456',
    'broker_id': 'your_broker_id',
    'account_id': 'your_account_id',
    'action': <Action.Buy: 'Buy'>,
    'code': '2890',
    'order_cond': <StockOrderCond.Cash: 'Cash'>,
    'order_lot': <TFTStockOrderLot.Common: 'Common'>,
    'price': 12,
    'quantity': 10,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1583828972
}
```

5.4 Futures Order

FuturesOrder

verion>=1.0 verion<1.0

```
>> sj.order.FuturesOrder?

Init signature:
sj.order.FuturesOrder(
    *,
    action: shioaji.constant.Action,
    price: Union[pydantic.types.StrictInt, float],
    quantity: shioaji.order.ConstrainedIntValue,
    id: str = '',
    seqno: str = '',
    ordno: str = '',
    account: shioaji.account.Account = None,
    custom_field: shioaji.order.ConstrainedStrValue = '',
    ca: str = '',
    price_type: shioaji.constant.FuturesPriceType,
    order_type: shioaji.constant.OrderType,
    octype: shioaji.constant.FuturesOCType = <FuturesOCType.Auto: 'Auto'>,
) -> None

>> sj.order.FuturesOrder?

Init signature:
sj.order.FuturesOrder(
    *,
    action: shioaji.constant.Action,
    price: Union[pydantic.types.StrictInt, float],
    quantity: shioaji.order.ConstrainedIntValue,
    id: str = '',
    seqno: str = '',
    ordno: str = '',
    account: shioaji.account.Account = None,
    custom_field: shioaji.order.ConstrainedStrValue = '',
    ca: str = '',
    price_type: shioaji.constant.FuturesPriceType,
    order_type: shioaji.constant.FuturesOrderType,
    octype: shioaji.constant.FuturesOCType = <FuturesOCType.Auto: 'Auto'>,
) -> None
```

5.4.1 Order

Rename `FuturesOrderType` to `OrderType`

Order

verion>=1.0 verion<1.0

```
order = api.Order(  
    action=sj.constant.Action.Buy,  
    price=100,  
    quantity=1,  
    price_type=sj.constant.FuturesPriceType.LMT,  
    order_type=sj.constant.OrderType.ROD,  
    octype=sj.constant.FuturesOCType.Auto,  
    account=api.futopt_account  
)  
  
order = api.Order(  
    action=sj.constant.Action.Buy,  
    price=100,  
    quantity=1,  
    price_type=sj.constant.FuturesPriceType.LMT,  
    order_type=sj.constant.FuturesOrderType.ROD,  
    octype=sj.constant.FuturesOCType.Auto,  
    account=api.futopt_account  
)
```

5.4.2 Order Callback

Rename FOrder to FuturesOrder

Order Event

version>=1.0 version<1.0

```
OrderState.FuturesOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '02c347f7',
        'seqno': '956201',
        'ordno': 'KV00H',
        'action': 'Sell',
        'price': 17760.0,
        'quantity': 1,
        'order_cond': None,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'market_type': 'Night',
        'oc_type': 'New',
        'subaccount': ''
    },
    'status': {
        'id': '02c347f7',
        'exchange_ts': 1625729890,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        "web_id": "P"
    },
    'contract': {
        'security_type': 'FUT',
        'code': 'TXF',
        'exchange': 'TIM',
        'delivery_month': '202107',
        'strike_price': 0.0,
        'option_right': 'Future'
    }
}

OrderState.FOrder {
    'operation': {
        'op_type': 'New',
        'op_code': '00',
        'op_msg': ''
    },
    'order': {
        'id': '02c347f7',
        'seqno': '956201',
        'ordno': 'KV00H',
        'action': 'Sell',
        'price': 17760.0,
        'quantity': 1,
        'order_cond': None,
        'order_type': 'ROD',
        'price_type': 'LMT',
        'market_type': 'Night',
        'oc_type': 'New',
        'subaccount': ''
    },
    'status': {
        'id': '02c347f7',
        'exchange_ts': 1625729890,
        'modified_price': 0.0,
        'cancel_quantity': 0,
        "web_id": "P"
    },
    'contract': {
        'security_type': 'FUT',
        'code': 'TXF',
        'exchange': 'TIM',
        'delivery_month': '202107',
        'strike_price': 0.0,
        'option_right': 'Future'
    }
}
```

5.4.3 Deal Callback

Rename FDeal to FuturesDeal

Deal Event

version>=1.0 version<1.0

```
OrderState.FuturesDeal {
    "trade_id": "02c347f7",
    "seqno": "956344",
    "ordno": "ky00N110",
    "exchange_seq": "a0000060",
    "broker_id": "F002000",
    "account_id": "9104000",
    "action": "Sell",
    "code": "TXF",
    "price": 17650.0,
    "quantity": 4,
    "subaccount": "",
    "security_type": "FUT",
    "delivery_month": "202107",
    "strike_price": 0.0,
    "option_right": "Future",
    "market_type": "Day",
    "ts": 1625800369
}

OrderState.FDeal {
    "trade_id": "02c347f7",
    "seqno": "956344",
    "ordno": "ky00N110",
    "exchange_seq": "a0000060",
    "broker_id": "F002000",
    "account_id": "9104000",
    "action": "Sell",
    "code": "TXF",
    "price": 17650.0,
    "quantity": 4,
    "subaccount": "",
    "security_type": "FUT",
    "delivery_month": "202107",
    "strike_price": 0.0,
    "option_right": "Future",
    "market_type": "Day",
    "ts": 1625800369
}
```

5.5 Market Data

Warn

version >= 1.1 will no longer provide QuoteVersion.v0, please change to QuoteVersion.v1.

5.5.1 Callback

Tick

pythonic way by using decorator

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import TickSTKv1, Exchange

@api.on_tick_stk_v1()
def quote_callback(exchange: Exchange, tick: TickSTKv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

traditional way

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import TickSTKv1, Exchange

def quote_callback(exchange: Exchange, tick:TickSTKv1):
    print(f"Exchange: {exchange}, Tick: {tick}")

api.quote.set_on_tick_stk_v1_callback(quote_callback)

def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")

api.quote.set_quote_callback(quote_callback)
```

Cut

QuoteVersion.v1 QuoteVersion.v0

```
Exchange: Exchange.TSE, Tick: Tick(code='2330', datetime=datetime.datetime(2021, 7, 2, 13, 16, 35, 92970), open=Decimal('590'), avg_price=Decimal('589.05'), close=Decimal('590'), high=Decimal('593'), low=Decimal('587'), amount=Decimal('590000'), total_amount=Decimal('8540101000'), volume=1, total_volume=14498, tick_type=1, chg_type=4, price_chg=Decimal('-3'), pct_chg=Decimal('-0.505902'), trade_bid_volume=6638, ask_side_total_vol=7860, bid_side_total_cnt=2694, ask_side_total_cnt=2705, closing_ordinal_shares=0, fixed_trade_vol=0, suspend=0, simtrade=0, intraday_odd=0)

Topic: MKT/*TSE/2330, Quote: {'AmountSum': [4739351000.0], 'Close': [596.0], 'Date': '2021/03/30', 'TickType': [2], 'Time': '10:01:33.349431', 'VolSum': [7932], 'Volume': [1]}
```

BidAsk

pythonic way by using decorator

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import BidAskSTKv1, Exchange

@api.on_bidask_stk_v1()
def quote_callback(exchange: Exchange, bidask:BidAskSTKv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

@api.quote.on_quote
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```

traditional way

QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import BidAskSTKv1, Exchange

def quote_callback(exchange: Exchange, bidask:BidAskSTKv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")

api.quote.set_on_bidask_stk_v1_callback(quote_callback)

def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")

api.quote.set_quote_callback(quote_callback)
```

```

QuoteVersion.v1      QuoteVersion.v0

Exchange: Exchange.TSE, BidAsk: BidAsk(code='2330', datetime=datetime.datetime(2021, 7, 2, 13, 17, 29, 726428), bid_price=Decimal('589'), Decimal('588'), Decimal('587'), Decimal('586'), Decimal('585')), bid_volume=[223, 761, 1003, 809, 1274], diff_bid_vol=[0, 0, 0, 0, 0], ask_price=[Decimal('590'), Decimal('591'), Decimal('592'), Decimal('593'), Decimal('594')], ask_volume=[304, 232, 183, 242, 131], diff_ask_vol=[1, 0, 0, 0, 0], suspend=0, simtrade=0, intraday_odd=0)

Topic: QUT/idcdmpr01/TSE/2330, Quote: {'AskPrice': [590.0, 591.0, 592.0, 593.0, 594.0], 'AskVolume': [303, 232, 183, 242, 131], 'BidPrice': [589.0, 588.0, 587.0, 586.0, 585.0], 'BidVolume': [224, 762, 1003, 809, 1274], 'Date': '2021/07/02', 'Time': '13:17:26.391840'}

```

5.6 Future Account Info.

Remove functions

1. get_account_margin
2. get_account_openposition
3. get_account_settle_profitloss

Instead, you should use

1. margin
2. list_positions(api.futopt_account)
3. list_profit_loss(api.futopt_account)
4. list_profit_loss_detail(api.futopt_account)
5. list_profit_loss_summary(api.futopt_account)

For more information, please refer to [Account Data section](#).

Finally, give us support and encouragement on [GITHUB](#)

Sinotrade / Shioaji (Public)

Code Issues 33 Pull requests Actions Projects 1 Wiki Security Insights

About

Shioaji all new cross platform api for trading (跨平台證券交易API)

sinotrade.github.io/

python docker cross-platform
trading trading-api market-data
trading-platform trade trading-simulator
quote streaming-data pythonic-api

Readme 6 stars 5 watching 0 forks

6. QA

6.0.1 下單

如何下市價單(MKT)、範圍市價單(MKP)

```
order = api.Order(
    action=sj.constant.Action.Buy,
    price=0, # MKT, MKP will not use price parameter
    quantity=1,
    price_type='MKT', # change to MKT or MKP
    order_type='IOC', # MKT, MKP only accept IOC order
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
```

如何掛漲(跌)停限價ROD單

First, we need to know the limit up(limit down) price of the security. Just take a look at the `api.Contracts`, you will find the information you want.

Out

```
Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2330',
    symbol='TSE2330',
    name='台積電',
    category='24',
    unit=1000,
    limit_up=653.0,
    limit_down=535.0,
    reference=594.0,
    update_date='2021/08/27',
    margin_trading_balance=6565,
    short_selling_balance=365,
    day_trade=<DayTrade.Yes: 'Yes'>
)
```

Example place LMT and ROD order at limit up price.

Out

```
contract = api.Contracts.Stocks.TSE['TSE2330']
price = contract.limit_up
order = api.Order(
    action=sj.constant.Action.Buy,
    price=price,
    quantity=1,
    price_type='LMT',
    order_type='ROD',
    order_lot=sj.constant.StockOrderLot.Common,
    account=api.stock_account
)
```

6.0.2 行情

為什麼行情只能收幾行就斷掉了

If your code something like this, and possibly run code on cmd/terminal with `python stream.py`. Then you definitely won't get any additional ticks, since the python program has already terminated.

version>=1.0 version<1.0

```
import shioaji as sj

api = sj.Shioaji(simulation=True)
api.login('YOUR_API_KEY', 'YOUR_SECRET_KEY')
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick
)

# stream.py
import shioaji as sj

api = sj.Shioaji(simulation=True)
api.login('YOUR_PERSON_ID', '2222')
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick
)
```

If you wish your python program to survive, please modify you python script as below.

version>=1.0 version<1.0

```
# stream.py
import shioaji as sj
from threading import Event

api = sj.Shioaji(simulation=True)
api.login('YOUR_API_KEY', 'YOUR_SECRET_KEY')
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick
)

Event().wait()

# stream.py
import shioaji as sj
from threading import Event

api = sj.Shioaji(simulation=True)
api.login('YOUR_PERSON_ID', '2222')
api.quote.subscribe(
    api.Contracts.Stocks["2330"],
    quote_type = sj.constant.QuoteType.Tick
)

Event().wait()
```

6.0.3 其他

出現 Account not acceptable，可能原因如下

- 未完成[簽署](https://sinotrade.github.io/zh_TW/tutor/prepare/terms/#_1)及[API測試](https://sinotrade.github.io/zh_TW/tutor/prepare/terms/#api)。
- ['update_status'](./tutor/order/UpdateStatus)預設查詢為名下所有帳號，若想使用預設查詢方式，請確認所有帳號皆有完成簽署及測試。

如何更改shioaji.log

Please add environment variable before import shioaji. (version >= 0.3.3.dev0)

linux or Mac OS:

```
export SJ_LOG_PATH=/path/to/shioaji.log
```

windows:

```
set SJ_LOG_PATH=C:\path\to\shioaji.log
```

如何更改contracts下載路徑

Please add environment variable before import shioaji. (version >= 0.3.4.dev2)

linux or Mac OS:

```
export SJ_CONTRACTS_PATH=MY_PATH
```

windows:

```
set SJ_CONTRACTS_PATH=MY_PATH
```

python:

```
os.environ["SJ_CONTRACTS_PATH"] = MY_PATH
```

輸入密碼錯誤3次怎麼辦

線上解鎖

** Note that you only have 2 chances to unlock your account online in a day. **

** We've migrate QA site to [Shioaji Forum](#) **

7. Release Note

7.1 version: 1.3.0 (2025-12-17)

- feat: futures quote
- feat: trading_limits api

⌚ commit_id: d570ac7b

⚡
release_at: 2025-12-17 16:00:00.000

7.2 version: 1.2.9 (2025-10-29)

- feat: support py3.14
- fix: profit_loss callback error

⌚ commit_id: de2133bf

⚡
release_at: 2025-10-29 16:00:00.000

7.3 version: 1.2.8 (2025-09-10)

- feat: punish api
- feat: notice api

⌚ commit_id: 534d1ab5

⚡
release_at: 2025-09-10 16:00:00.000

7.4 version: 1.2.7 (2025-08-13)

- refactor: futures profit loss

⌚ commit_id: e417ffcc

⚡
release_at: 2025-08-13 16:00:00.000

7.5 version: 1.2.6 (2025-06-16)

- feat: support python 3.13
- feat: support linux aarch64
- fix: mac import link error
- feat: contract download event
- feat: pysolace upgrade 0.9.51 (solclient 7.33.0.3)
- chore: drop support for python 3.6

⌚ commit_id: cf4b448d

↳
release_at: 2025-06-16 16:00:00.000

7.6 version: 1.2.5 (2024-10-01)

- feat: refactor expire time of CA

⌚ commit_id: 6621685a

↳
release_at: 2024-10-01 02:25:01.723

7.7 version: 1.2.4 (2024-08-28)

- feat: support py3.12

⌚ commit_id: a287f56c

↳
release_at: 2024-08-28 16:00:00.000

7.8 version: 1.2.3 (2024-03-06)

- feat: change default site to bc
- feat: pysolace upgrade 0.9.40(solclient 7.28.0.4)
- feat: support apple silicon chip

⌚ commit_id: 8096bbac

↳
release_at: 2024-03-06 16:00:00.000

7.9 version: 1.2.2 (2024-01-09)

- fix: remove column of profitloss in future

⌚ commit_id: ca973a81

↳
release_at: 2024-01-09 02:27:31.383

7.10 version: 1.2.1 (2023-12-22)

- fix: windows inject dll issue

⌚ commit_id: 2a413848

↳
release_at: 2023-12-22 01:19:17.043

7.11 version: 1.2.0 (2023-12-20)

- feat: vpn
- feat: rust version ca

- refactor: test report flow

⌚ commit_id: 856f39ea

⚡
release_at: 2023-12-20 16:00:00.000

7.12 version: 1.1.13 (2023-11-01)

feat: impl ca.get_sign on Darwin

⌚ commit_id: 729f058e

⚡
release_at: 2023-11-01 05:36:29.553

7.13 version: 1.1.12 (2023-08-22)

- feat: usage add limit and available byte info

⌚ commit_id: cf5e4628

⚡
release_at: 2023-08-22 16:00:00.000

7.14 version: 1.1.11 (2023-08-04)

- fix: custom_field in validator for only support number and alphabet
- fix: pydantic v2 trade issue
- fix: pydantic v2 contracts cache issue

⌚ commit_id: cc1da47e

⚡
release_at: 2023-08-04 08:00:37.000

7.15 version: 1.1.10 (2023-07-23)

- feat: profit loss detail support unit

⌚ commit_id: a62d1f6a

⚡
release_at: 2023-07-23 16:00:00.000

8. Use Restrictions

In order to avoid affecting other users' connections, please follow the following usage rules.

Traffic

- Stock :

Trading volume by API in the past 30 days	Daily Traffic Limit
0	500MB
1 - 1E	2GB
> 1E	10GB

- Future :

Trading volume by API in the past 30 days	Daily Traffic Limit
0	500MB
1 - TXF 1000 contracts / MXF 4000 contracts	2GB
> TXF 1000 contracts / MXF 4000 contracts	10GB

Traffic query

```
api.usage()
```

Stat

```
UsageStatus(connections=1, bytes=41343260, limit_bytes=2147483648, remaining_bytes=2106140388)
```

```
connection: connection count.  
bytes: traffic.  
limit_bytes: limit bytes of daily.  
remaining_bytes: remaining bytes of daily.
```

Counts

- Data :

```
credit_enquire, short_stock_sources, snapshots, ticks, kbars
```

The total amount of inquiries above is limited to 50 times within 5 seconds.

During trading hours, it is prohibited to query `ticks` more than 10 times.

During trading hours, it is prohibited to query `kbars` more than 270 times.

- Portfolio :

```
list_profit_loss_detail, account_balance, list_settlements, list_profit_loss, list_positions, margin
```

The total amount of inquiries above is limited to 25 times within 5 seconds.

- Order :

```
place_order, update_status, update_qty, update_price, cancel_order
```

The total amount of inquiries above is limited to 250 times within 10 seconds.

- Subscribe :

Number of `api.subscribe()` is 200.

- Connect :

The same SinoPac Securities `person_id` can only use up to 5 connections.

note. `api.login()` create a connection.

- Login :

Up to 1000 times per day.

Warn

- If the traffic exceeds the limit, query requests for market data such as ticks, snapshots, and kbars will return empty values, while other functionalities remain unaffected.
- If the usage exceeds the limit, the service will be suspended for one minute.
- If the limit is exceeded multiple times in a row on the same day, the company will suspend the right to use the IP and ID.
- If the ID is suspended, please contact Shioaji management staff