

# Shioaji

**Usage Manual** 

# Table of contents

1. Sh	nioaji	4
1.1	Installation	4
2.		5
2.1	Streaming Market Data	5
2.2	Place Order	5
2.3	Conclusion	6
3. Er	nvironment Setup	7
3.1	Install Python Environment	7
3.2	Create Project Environment	7
4.	-	10
4.1		10
4.2		22
4.3		26
4.4		30
4.5		58
4.6	CallBack	96
4.7		104
4.8		118
4.9		119
5.		134
5.1	Shioaji	134
5.2	Login	136
5.3	Stock Order	136
5.4	Futures Order	140
5.5	Market Data	143
5.6	Future Account Info.	145
6.		146
7.		149
7.1	version: 1.2.5 (2024-10-01)	149
7.2	version: 1.2.4 (2024-08-28)	149
7.3	version: 1.2.3 (2024-03-06)	149
7.4	version: 1.2.2 (2024-01-09)	149
7.5	version: 1.2.1 (2023-12-22)	149
7.6	version: 1.2.0 (2023-12-20)	149
7.7	version: 1.1.13 (2023-11-01)	150

7.8 version: 1.1.12 (2023-08-22)	150
7.9 version: 1.1.11 (2023-08-04)	150
7.10 version: 1.1.10 (2023-07-23)	150
7.11 version: 1.1.9 (2023-07-20)	150
7.12 version: 1.1.8 (2023-07-18)	151
7.13 version: 1.1.7 (2023-07-18)	151
7.14 version: 1.1.6 (2023-06-19)	151
7.15 version: 1.1.5 (2023-06-07)	151
8.	152

# 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.

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",
)
```

# **A**e 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. Simplely 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["TXFC0"], quote_type="tick")
```

# **Q**uote Type

Currently we support two quote type you can see in shioaji.constent.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:

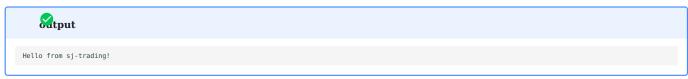
add Shioaji API to the project

```
uv add shioaji
```

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

the hello command is the entry point of the project.





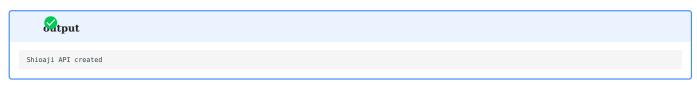
open src/sj\_trading/\_\_init\_\_.py file and copy the following content

```
import shioaji as sj

def hello():
    get_shioaji_client()

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





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

# 3.2.1 Use Jupyter Environment



# Ad the project environment to the Jupyter kernel

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

# **Q**art 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. -

# 4.1

# 4.1.1

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  $\mbox{ \ \, }$   $\mbox{ \ \, DAWHO+ \ \, }$  , to open a bank account and a securities account. open\_bank\_2
- 4. Complete bank and securities account opening.

# 4.1.2

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

- Go to the API management page in the personal service. newweb\_1
- 2. Click Add API KEY. newweb 2

newweb 3

- 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.
- 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

# **Ermission 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

# Atention

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  $\,5\,$ 



- 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

 $2.\ \mbox{Download}$  the certificate and place it into the folder that the API can read

newweb 8

# **Confirm The API Key And Certificate**

 $Continue \ with \ the \ previous \ project \ \textit{sj-trading} \ , \ add \ . \ env \ file \ in \ the \ project \ folder, \ and \ add \ the \ following \ content$ 

.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

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

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. signature

#### **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



Service Hour:

- In response to the company's information security regulations, the test service is Monday to Friday  $08:00\sim20:00$
- 18:00 ~ 20:00: Only allow Taiwan IP
- 08:00 ~ 18:00: No limit

Version Restriction:

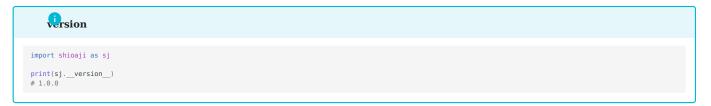
• version >= 1.2:

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



• please note the Version Restriction.

#### **LOGIN TEST**

```
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
)</pre>
```

- 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

```
Sock 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,
        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,
        {\tt account=api.stock\_account}
# place order
trade = api.place_order(contract, order)
```



```
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=<Status.Submitted: 'Submitted'>,
        order_datetime=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**

# Ature 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,
         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,
         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
```



- 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
- Future Order

CHECK IF API TESTS HAS PASSED



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.

# **Agn Status**



```
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) | Event: Session up

[FutureAccount(person_id='QBCCAIGJBJ', broker_id='F002000', account_id='9100020', signed=True, username='PAPIUSER01'),
StockAccount(person_id='QBCCAIGJBJ', 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.

ADDIV CA

- 1. Go to SinoPac Securities to download eleader eleader download
- 2. Login eleader login eleader
- 3. Select (3303)  $\,$  from the  $\,$  above eleader\_account
- 4. Click " "CA step
- 5. CA Operation steps CA\_info

#### ACTIVATE CA

- If you use simulation account, you don't have to activate CA.
- $\bullet \ \, \text{If you are a macOS user, you may subject to version-issue. We suggest you to use } \, \text{docker} \, \text{and run shioaji service on docker.} \\$

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

# Ae 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

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.

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

# 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 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
             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**

# Ad Futures Testing Content

Add the following content to src/sj\_trading/testing\_flow.py

```
from shioaji.constant import (
         {\tt FuturesPriceType},
         FuturesOCType,
def testing_futures_ordering():
    # Login to the testing environment
        # 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"],
        # 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(
               ler = 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

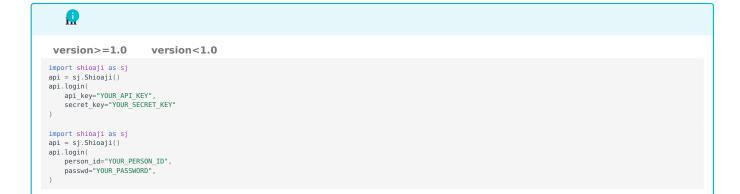
# 4.2

Login 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



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.



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

 $\bullet$  If you cannot find  $\,$  signed  $\,$  in your accounts, please refer to terms of service first.

```
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 0rder/Deal event callback (Default: True)
receive_window (int): valid duration for login execution. (Default: 30,000 ms)

person_id (str): person_id
passwd (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 timeout (Default: None)
subscribe_trade (bool): whether to subscribe 0rder/Deal event callback (Default: True)</pre>
```

# **A**rning

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.")
)</pre>
```

```
[
    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 <code>subscribe\_trade</code> in login aruguments. Default value of <code>subscribe\_trade</code> is <code>True</code>, 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
)
</pre>
```

The second one is to manually use the API subscribe trade and unsubscribe trade for specific account.

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

```
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')
```

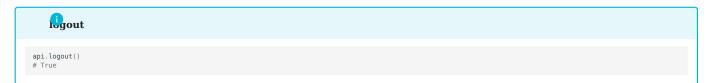
In Order object, you need to specify which account you want to place order. For more information about Order, please refer to Stock Order and Futures Order.

```
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 funciton 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.



# 4.3

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_EERET_KEY",
    contracts_timeout=10000,
)

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

• 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_contract=False,
)
api.fetch_contracts(contract_download=True)</pre>
```

### **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.





#### STOCK



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



```
Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2890',
    symbol='TSE2890',
    name=' ',
    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'>
}
```

# Stock

```
exchange (Exchange): Attributes of industry
{DES, OTC, TSE ...etc}

code (str): Id

symbol (str): Symbol

name (str): Name

category (str): Category

unit (init): 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',
    symbol='TSE2890',
    name=' ',
    category='17',
    unit=1800,
    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=' 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'
```

```
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.TX0.TX020230618000P



```
Option(
    code='TX018000R3',
    symbol='TX020230618000P',
    name=' 06 18000P',
    category='TXO',
    delivery_month='202306',
    delivery_date='2023/06/21',
    strike_price=18000,
    option_right=<0ptionRight.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 Indexs 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.Indexs.TSE["001"]
# or api.Contracts.Indexs.TSE.TSE001
```

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

```
exchange (Exchange): exchange
{0ES, OTC, TSE ...etc}
code (str): Code
symbol (str): Symbol
name (str): Name
```

# 4.4

# 4.4.1

 $To \ subscribe \ \ function \ \ with \ contract \ which \ we've \ discussed \ in \ previous \ topic.$ 

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

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

#### TICK

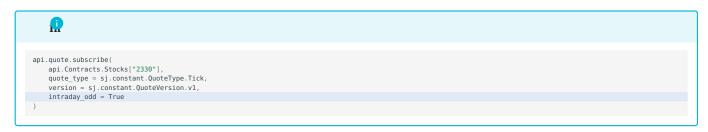
# **Common Stock**

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



# 

## Intraday odd





# QuoteVersion.v1 QuoteVersion.v0 Response Code: 200 | Event Code: 16 | Info: TIC/v1/ODD/\*/TSE/2330 | Event: Subscribe or Unsubscribe ok ${\sf Exchange.TSE}$ Tick( code = '2330', datetime = datetime.datetime(2021, 7, 2, 13, 16, 55, 544646), open = Decimal('591'), open = Decimal(1591), avg\_price = Decimal(1590.24415'), close = Decimal(1590'), high = Decimal(1591'), low = Decimal(1589'), amount = Decimal('276120'), total\_amount = Decimal('204995925'), volume = 468, total\_volume = 347307, total\_volume = 347307, tick\_type = 1, chg\_type = 4, price\_chg = Decimal('-3'), pct\_chg = Decimal('-0.505902'), bid\_side\_total\_vol= 68209, ask\_side\_total\_vol = 279566, bid\_side\_total\_cnt = 28, ask\_side\_total\_cnt = 56, closing\_oddlost\_serse = 0. closing\_oddlot\_shares = 0, fixed\_trade\_vol = 0, suspend = 0, simtrade = 1, intraday\_odd = 1 $Response\ \ Code:\ \ 200\ \ |\ \ Event\ \ Code:\ \ 16\ \ |\ \ Info:\ \ TIC/v2/*/TSE/2330/ODDLOT\ \ |\ \ Event:\ \ Subscribe\ \ or\ \ Unsubscribe\ \ ok$ TIC/v2/replay/TSE/2330/ODDLOT 'Date': '2021/07/01', 'Time': '09:23:36.880078', 'Close': '593', 'TickType': 1, 'Shares': 1860, 'SharesSum': 33152, 'Simtrade': 1

#### Attributes



open (decimal):

avg\_price (decimal):

#### QuoteVersion.v1 QuoteVersion.v0 code (str): datetime (datetime):

```
intraday_odd (int): {0: , 1: }
 AmountSum (:List:float):
 Amountum (Listifloat):
Date (str): (yyyy/MM/dd)
TickType (:Listiint): {1: , 2: , 0:
Time (str): (HH:mm:ss.ffffff)
VolSum (:Listiint): ()
Volume (:List:int): ()
```

#### BIDASK

#### Common Stock

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

#### Intraday odd

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



# 

#### **Attributes**

```
Code (str):
datetime (datetime):
bid_price (:List:decimal):
bid_volume (:List:int): ()
diff_bid_vol (:List:int): ()
ask_price (:List:decimal):
ask_volume (:List:int): ()
suspend (bool):
simtrade (bool):
AskPrice (:List:float):
AskVolume (:List:int):
BidVolume (:List:int):
Bi
```

## QUOTE

### Common Stock

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

```
at
Response Code: 200 | Event Code: 16 | Info: QUO/v2/STK/*/TSE/2330 | Event: Subscribe or Unsubscribe ok
Exchange.TSE,
      code='2330'
      avg_priceDecimal('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,
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,
```

## Attributes

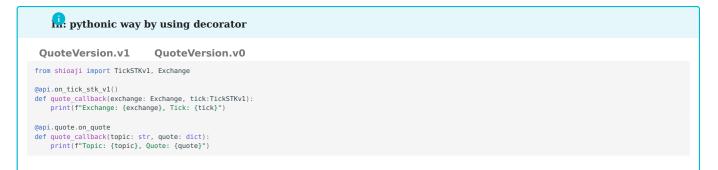
simtrade=0

```
Quote
code (str):
 datetime (datetime):
  open (decimal):
avg_price (decimal):
close (decimal):
high (decimal):
low (decimal): ( )
amount (decimal): (NTD)
total_amount (decimal):
  volume (int):
  total_volume (int):
tick_type (int):
chg_type (int):
price chg (decimal):
prtc_chg (decimal):
bid_side_total_vol (int):
ask_side_total_vol (int):
bid_side_total_cnt (int):
ask_side_total_cnt (int):
closing_oddlot_shares (int)
closing_oddlot_close (decimal):
closing_oddlot_amount (decimal):
closing_oddlot_amount (decimal):
closing_oddlot_ask_price (decimal):
fixed_trade_vol (int):
fixed_trade_amount (decimal):
bid_price_(list)decimal):
bid_price (:List:decimal):
bid_volume (:List:int)
diff_bid_vol (:List:int)
ask_price (:List:decimal):
ask_volume (:List:int)
diff_ask_vol (:List:int)
diff_ask_vol (:List:int)
avail berrowing (int):
  avail_borrowing (int):
 suspend (bool):
simtrade (bool):
```

#### 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



# traditional way

#### OuoteVersion.v1 OuoteVersion.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)
```



#### OuoteVersion.v1 OuoteVersion.v0

Exchange: Exchange.TSE, Tick: Tick(code='2330', datetime=datetime.datetime(2021, 7, 2, 13, 16, 35, 92970), open=Decimal('590'), avg\_price=Decimal('580.05'), close=Decimal('590'), high=Decimal('593'), low=Decimal('597'), amount=Decimal('590000'), total\_amount=Decimal('3540101000'), 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\_oddlot\_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('587'), Decimal('587'), Decimal('587'), Decimal('587'), Decimal('591'), Dec
```

#### Quote

# 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}")
```

# 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)
```



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('-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\_amount=Decimal('0'), bid\_price=[Decimal('461'), Decimal('460.5'), Decimal('460.5'), Decimal('459.5'), Decimal('459.5'), bid\_volume=[220, 140, 994, 63, 132], diff\_bid\_vol=[-1, 0, 0, 0, 0], ask\_price=[Decimal('461.5'), Decimal('462.5'), Decimal('462.5'), Decimal('463.5')], ask\_volume=[115, 101, 103, 147, 91], diff\_ask\_vol=[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.

 $To \ subscribe \ \ function \ \ with \ contract \ which \ we've \ discussed \ in \ previous \ topic.$ 

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

#### TICK

#### Example

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



#### QuoteVersion.v1 QuoteVersion.v0

```
Response Code: 200 | Event Code: 16 | Info: TIC/v1/F0P/*/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'),
        open = Decimal('17678'),
underlying_price = Decimal('17849.57'),
bid_side_total_vol= 32210,
ask_side_total_vol= 33218,
avg_price = Decimal('17794.663999'),
close = Decimal('17773'),
high = Decimal('17774'),
low = Decimal('17755'),
amount = Decimal('17753'),
total amount = Decimal('913790823')
         total_amount = Decimal('913790823'),
volume = 1,
total_volume = 51613,
         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],
          'TickType': [2],
'Time': '10:42:25.552000',
          'TradeAskVolSum': 33198,
'TradeBidVolSum': 32180,
          'VolSum': [51570],
'Volume': [1]
```

#### **Attributes**

```
QuoteVersion.v1 QuoteVersion.v0

code (str):
dstetime (dstetime,dstetime):
open (Decimal):
underlying price (Decimal):
bid side ctotal_vol(int): (lot)
ask_side_total_vol(int): (lot)
ask_side_total_vol(int): (lot)
ask_side_total_vol(int): (lot)
down(Decimal):
low (Decimal):
lo
```

#### BIDASK

#### Example

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



#### QuoteVersion.v1 QuoteVersion.v0

#### **Attributes**

```
BidAsk
  QuoteVersion.v1
                                                   QuoteVersion.v0
code (str):
datetime (datetime.datetime):
bid_total_vol (int): (l
ask_total_vol (int): (l
bid_price (:List:decimal):
                                             (lot)
bid_volume (:List:int):
diff_bid_vol (:List:int):
                                              (lot)
                                                       (lot)
ask_price (:List:decimal):
ask_volume (:List:int): (lot)
diff_ask_vol (:List:int):
first_derived_bid_price (decimal):
first_derived_ask_price (decimal):
                                                        (lot)
first_derived_bid_vol (int):
first_derived_ask_vol (int):
underlying_price (decimal): simtrade (int):
AskPrice (:List:float):
AskVolSum (int): (lot)
AskVolume (:List:int):
BidPrice (:List:float):
BidVolSum (int): (lot)
BidVolSum (int):
BidVolume (:List:int):
Code (str):
Date (str): (yyyy/MM/
DiffAskVol (:List:int):
                                                (lot)
DiffAskVolSum (int):
DiffBidVol (:List:int):
                                                (lot)
DiffBidVolSum (int):
FirstDerivedAskPrice (float):
FirstDerivedAskVolume (int): FirstDerivedBidPrice (float):
FirstDerivedBidVolume (int):
TargetKindPrice (float):
Time (str): (HH:mm:ss.ffffff)
```

#### 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

```
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}")
```

```
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)
```



#### 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/TXF61, Quote: {'Amount': [17654.0], 'AmountSum': [1682856730.0], 'AvgPrice': [17657.961764], 'Close': [17654.0], 'Code': 'TXF61', '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



#### 🔛 pythonic way by using decorator

#### QuoteVersion.v1 QuoteVersion.v0

```
from shioaii import BidAskFOPv1. Exchange
@api.on_bidask_fop_v1()
def quote_callback(exchange:Exchange, bidask:BidAskFOPv1):
    print(f"Exchange: {exchange}, BidAsk: {bidask}")
def quote_callback(topic: str, quote: dict):
    print(f"Topic: {topic}, Quote: {quote}")
```



#### H: traditional way

#### QuoteVersion.v1 QuoteVersion.v0

```
from shioaji import BidAskFOPv1, Exchange
def guote callback(exchange:Exchange, bidask:BidAskFOPv1):
   print(f"Exchange: {exchange}, BidAsk: {bidask}"
\verb"api.quote.set_on_bidask_fop_v1_callback" (\verb"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.TAIFEX, BidAsk: BidAsk(code='TXF61', 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], ask_price=[Decimal('17653'), Decimal('17653'), Decimal('17655'), Decimal('17655')], ask_volume=[6, 17, 29, 22, 20], diff_ask_vol=[0, 0, 0, 0], first_derived_bid_price=Decimal('17647'), first_derived_ask_vol=2, first_derived_ask_vol=3, underlying_price=Decimal('17725.5'), simtrade=0)
```

Topic: Q/TFE/TXF61, 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, 17648.0, 17648.0, 17648.0, 17648.0, 17648.0, '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'}

• Advanced quote callback settings please refer to Quote-Binding Mode.

### 4.4.2

#### **Ticks**

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

```
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_ent: 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

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

```
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]
}
```

```
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): {1: , 2: , 0: }
```

#### To DataFrame

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



#### RANGE TIME

```
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
```

```
Ticks(
    ts=[1673859600113699000, 1673859600228800000, 1673859600244294000, 1673859600308595000],
    close=[506.0, 505.0, 506.0, 506.0],
    volume=[3340, 1, 17, 2],
    bid_price=[505.0, 506.0, 506.0, 506.0],
    bid_volume=[122, 320, 60, 58],
    ask_price=[506.0, 506.0, 500.0, 507.0],
    ask_volume=[13, 22, 702, 702]
    tick_type=[1, 2, 1, 2]
}
```

### LAST COUNT

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

```
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, 167385984000000000],
    Open=[506.0, 505.0, 504.0],
    High=[508.0, 506.0, 506.0, 506.0],
    Low=[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()
```

$\mathcal{E}_{\mathrm{dt}}$						
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**

111000110411	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 = api.ticks(
   contract=api.Contracts.Futures.TXF.TXFR1,
   date="2020-03-22"
)
ticks
```

```
Ticks(
    ts=[1616166000030000000, 1616166000140000000, 1616166000140000000, 1616166000190000000],
    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

```
Bars
```

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

```
Kbars(
    ts=[16164027600000000000, 1616402820000000000, 161640288000000000, 161640294000000000],
    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 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.



Each snapshot can contain up to 500 contracts.

```
Signature:
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**

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

#### To DataFrame

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

<b>€</b> at								
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}
avgerage_price (float): avgerage of price
volume (int): volume
total_volume (int): total volume
amount (int): Deal amount
total_amount (int): Total deal amount
yestoday_volume (float): Volume of yestoday
buy_volume (float): Price of buy
buy_volume (float): Price of sell
sell_price (float): Price of sell
sell_volume (int): Volume of sell
sell_volume (int): Volume of sell
sell_volume (int): Volume of sell
volume_ratio (float): total_volume/yestoday_volume
```

# 4.4.4

```
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**

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

### 4.4.5

```
Credit Enquires

>> api.credit_enquires?

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

### Example

```
contracts = [
    api.Contracts.Stocks['2330'],
    api.Contracts.Stocks['2890']
]
credit_enquires = api.credit_enquires(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=1351),
CreditEnquire(update_time='2020-12-09 10:56:05', system='HF', stock_id='2330', margin_unit=1314),
CreditEnquire(update_time='2020-12-01 09:33:04', system='HF', 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
```

<b>€</b> at					
	margin_unit	short_unit	stock_id	system	update_time
0	1381	0	2330	HE	2020-12-11 13:30:13
1	1371	0	2330	НС	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	НЕ	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**

# **Credit**Enquire

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

```
Signature:
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.

```
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**

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

#### 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
```

	_					_	_	
date	code	name	ts	open	high	low	close	price_range
2023-01-17	6259		2023-01-17 11:11:41.030000	22.8	23.75	22.45	23.75	1.3
2023-01-17	6788		2023-01-17 11:19:01.924000	107	116	107	116	
2023-01-17	2540		2023-01-17 11:17:39.435000	85.2	85.2	83	85.2	2.5
2023-01-17	8478		2023-01-17 11:18:33.702000	350.5	378	347	378	3:
2023-01-17	6612		2023-01-17 11:15:32.752000	102	109	102	109	

#### **Attributes**

# ChangePercentRank

```
date (str): date
    code (str): code
    name (str): name
    ts (int): timestamp
    open (float): open price
    high (float): high price
    low (float): close price
    close (float): close price
    close (float): close price
    rick_type (int): {1: , 2: , 0: }
    change_price (float): change in price
    change_type (int): {1: , 2: , 0: }
    change_price (float): change in price
    change_type (int): {1: , 2: , 0: }
    change_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/yestoday_volume
    volume_ratio (float): total_volume/yestoday_volume
    buy_price (float): bid
    buy_volume (int): sak
    sell_volume (int): ask
    sell_volume (int): total volume that deal at bid
    bid_volumes (int): number of orders that deal at ask
    ask_volumes (int): total volume that deal at ask
    ask_volumes (int): total volume that deal at ask
    ask_volumes (int): total volume that deal at ask
```

# 4.5

### 4.5.1



First, you need to login and activate CA.

#### STOCK ORDER

```
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, RMRP) (or int) order order (LMT, RMRP) (or int) order (MT, LMR) (or int) order (MT, LMR) (or int) order (MT, LMR) (order (MT, LMR)) (order (MT, LMR))
```

### PLACE ORDER

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

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



```
version>=1.0 version<1.0

order = api.Order(
    price=17,
    quantity=3,
    action=j.constant.Action.Buy,
    price_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.StockPriceType.LMT,
    order_type=sj.constant.StockPriceType.ROD,
    order_lot=sj.constant.StockPriceTot.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.TFTStockPriceType.ROD,
    order_lot=sj.constant.TFTStockOrderLot.Common,
    # first_sell=sj.constant.TFTStockPristSell.No,
    custom_field="test",
    account=api.stock_account
)</pre>
```

# **Pace Order**

trade = api.place\_order(contract, order)
trade

# **S**at

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.

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

```
at
Trade(
       contract=Stock(
               exchange=<Exchange.TSE: 'TSE'>,
code='2890',
               symbol='TSE2890',
name='',
              name=' ',
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'>
               action=<Action.Buy: 'Buy'>,
price=17,
               quantity=3,
id='531e27af'
               seqno='000002'
ordno='000001'
               account-decount(
    account_type=<AccountType.Stock: 'S'>,
    person_id='Al23456789',
    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.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
- Filling: Part Fill

#### **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: Ca lable[[shioaji.order.Trade], NoneType] = None,
        ) -> shioaji.order.Trade

Docstring: update the order price or qty
```

#### **Update Price**

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

# Update Quantity (Reduce)

update\_order can only reduce the quantity of the order.

# **p**date Quantity

api.update\_order(trade=trade, qty=1)
api.update\_status(api.stock\_account)
trade

#### CANCEL ORDER

# **L**ancel Order

api.cancel\_order(trade)
api.update\_status(api.stock\_account)
trade

```
Sat
```

```
Trade(
    contract=Stock(
    exchange=Exchange.TSE: 'TSE'>,
    code='2890',
    symbol='TSE2890',
    symbol='TSE2890',
    symbol='TSE2890',
    symbol='TSE2890',
    symbol='TSE2890',
    symbol='TYP,
    cotton='2012,
    symbol='TYP,
    cotton='2012,
    symbol='TYP,
    symbol='TYP,
```

DEAL

# **B**date Status

api.update\_status(api.stock\_account)
trade

```
at
Trade(
       de(
contract=Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2890',
    symbol='TSE2890',
               name=' ',
category='17',
               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(
                       ount=Account(
account_type=<AccountType.Stock: 'S'>,
person_id='4123456789',
broker_id='9A95',
account_id='1234567',
signed=True
               ),
custom_field='test',
price_type=<StockPriceType.LMT: 'LMT'>,
order_type=<OrderType.ROD: 'ROD'>,
daytrade_short=False
                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)
```

#### **Examples**

Stock place order jupyter link

ACTION

```
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
}
```

```
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_tot=sj.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
}
```

# 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.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.TFTStockPriceType.ROD, order\_type=sj.constant.TFTStockPriceType.ROD, order\_tsj.constant.TFTStockDriceType.ROD, order\_tsj.constant.TFTStockDriceTyp

ROD + LMT

```
version>=1.0 version<1.0

order = api.Order(
    price=12,
    quantity=1,
    action=9j.constant.Action.Sell,
    price type=9j.constant.StockPriceType.LMT,
    order type=9j.constant.StockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)

order = api.Order(
    price=12,
    quantity=1,
    action=9j.constant.Action.Sell,
    price_type=9j.constant.FTStockPriceType.LMT,
    order_type=9j.constant.FTStockPriceType.LMT,
    order_type=9j.constant.FTStockPriceType.LMT,
    order_type=9j.constant.FTStockPriceType.ROD,
    order_lot=9j.constant.TFTStockOrderLot.Common,
    custom_field="test",
    account=api.stock_account
)</pre>
```

### 4.5.2



First, you need to login and activate CA.

#### **FUTURES 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}
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} ( )
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.

```
Prace 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
```

```
Untract
```

contract = api.Contracts.Futures.TXF.TXF202301



```
version>=1.0  version<1.0

order = api.Order(
    action=sj.constant.Action.Buy,
    price=14480,
    quantity=3,
    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=14480,
    quantity=3,
    price_type=sj.constant.FuturesPriceType.LMT,
    order_type=sj.constant.FuturesOCType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)</pre>
```

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

```
at
Trade(
       contract=Future(
code='TXFA3'
              symbol='TXF202301',
name=' 01',
              name=' 01',
category='TXF',
delivery_month='202301',
delivery_date='2023/01/30',
underlying_kind='I',
unit=1
              underlying_kind='1',

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='F602000',
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 .

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



#### **Status of Trade**

• PendingSubmit: Sending

• PreSubmitted: Reservation

• Submitted: Send Successfully

• Failed: Failed

Cancelled : CancelledFilled : Complete Fill

• Filling: 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: Ca lable[[shioaji.order.Trade], NoneType] = None,
    ) -> shioaji.order.Trade
Docstring: update the order price or qty
```

#### **Update Price**

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

```
at
Trade(
        contract=Future(
               tract=buture(
code='TXFA3',
symbol='TXF202301',
name=' 01',
category='TXF',
delivery_month='202301',
delivery_date='2023/01/30',
underlying_kind='1',
unit=1.
               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='5efffdel',
seqno='000004',
ordno='000003',
                 account=Account(
                       Journ-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='5efffdel',
                status=<Status.Submitted: 'Submitted'>, status_code='00',
                order_datetime=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.

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

```
Sat
```

# CANCEL ORDER

# **l**ancel Order

api.cancel\_order(trade)
api.update\_status(api.futopt\_account)
trade

```
Trade!

contract=Future(
    code=TXFA3',
    symbol=TXF20331',
    name=' 01',
    category=TXF,
    delivery.moth='202301',
    delivery.moth='202301',
    delivery.moth='202301',
    delivery.moth='202301',
    delivery.moth='202301',
    delivery.moth='202301',
    delivery.moth='202301',
    limit_uo=16270.0,
    limit_uo=1
```

#### DEAL

# api.update\_status(api.futopt\_account) trade

#### **Examples**

Future and Option place order jupyter link

ACTION

```
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
}
```

```
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

## 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 }

### 4.5.3



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)
```

```
at
```

```
Trade(
       contract=Stock(
             exchange=<Exchange.TSE: 'TSE'>,
code='0050',
symbol='TSE0050',
name=' 50',
category='00',
              limit_up=115.8,
limit_down=94.8,
              eference=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='WA313',
               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**



Intraday Odd cannot update price.

update order can only reduce the quantity of the order.

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

### CANCEL ORDER

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



```
Trade:
contract-Stock(
exchange=Exchange.TSE: 'TSE'>,
code='0890',
symbol='TSE0890',
name=' 59',
category='00',
limit.yo='15.8,
limit.dom=94.8,
reference=105.3,
update_date='202/089/21',
wargin_trading_balance=15390,
short_selling_balance=15390,
sh
```

### 4.5.4



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.

```
api.place_comboorder?

Signature:
    api.place_comboorder(
        combo_contract: shioaji.contracts.ComboContract,
        order: shioaji.order.ComboOrder,
        timeout: int = 5000,
        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.

```
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  $update\_combo status$ .



api.update\_combostatus()
api.list\_combotrades()



```
ComboTrade(
         contract=ComboContract(
                    legs=[
ComboBase(
                                            DOBBSE(
security_type=<SecurityType.Option: 'OPT'>,
exchange=<Exchange.TAIFEX: 'TAIFEX'>,
code='TX516000L1',
symbol='TX5202112016000C',
                                          symbol='TX5202112016000C',
name=' 12W5 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.
                                           underlying_kind='1',
unit=1,
limit_up=3630.0,
limit_down=68.0,
reference=1850.0,
update_date='2021/12/23',
action=<Action.Sell: 'Sell'>),
hasan/
                                ComboBase(
                                         boBase(
security_type=<SecurityType.Option: 'OPT'>,
exchange=<Exchange.TAIFEX: 'TAIFEX'>,
code='TX516000X1',
symbol='TX5202112016000P',
name=' 12W5 16000P',
category='TX5',
delivery_monthe'202112',
delivery_date='2021/12/29',
strike_price=16000.0,
option_right=<OptionRight.Put: 'P'>,
underlying_kind='I',
unit=1,
unit=1,
                                            unit=1,
limit_up=1780.0,
                                          limit_up=1/80.0,
limit_down=0.1,
reference=0.9,
update_date='2021/12/23',
action=<Action.Sell: 'Sell'>)
          order=Order(
                    er=urder(
    action=<Action.Sell: 'Sell'>,
    price=1.0,
    quantity=1,
    id='46989de8',
                    seqno='743595'
ordno='000000'
                     drung e00000 ;
account_type=<AccountType.Future: 'F'>,
person_id='YOUR_PERSON_ID',
broker_id='F002000',
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.datetime(2021, 12, 23, 8, 46, 47),
                    msq='
                    modified_price=1.0,
deals={}
```

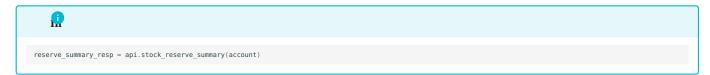
### 4.5.5

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.



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

### **GET STOCK RESERVE SUMMAY STATUS**



### 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='
),
account=StockAccount(
person_id='X123456789',
broker_id='9895',
account_id='12345678',
signed=True),
share=1000,
status=True,
info=''
)
}
```

### GET STOCK RESERVE DETAIL SATUS

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

### RESERVE EARMARKING

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

```
ReserveEarmarkingResponse(
    response=EarmarkingOrderResp(
    contract=Stock(
        exchange=Cxchange.TSE: 'TSE'>,
        code='2890',
        symbol='TSE2890',
        name=' ',
    ),
    account=StockAccount(
        person_id='X123456789',
        broker_id='9A95',
        account_id='12345678',
        signed=True)
    ),
    share=1800,
    price=15.15,
    status=True,
    info='0K')
}
```

### **GET EARMARKING DETAIL STATUS**



### EXAMPLE

Query the reserve status of all accounts under your name.



### 4.5.6



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.



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



api.update\_status(api.stock\_account)
api.list\_trades()

```
Sat
```

```
Trade(
    contract=Stock(
        exchange=Exchange.TSE: 'TSE'>,
        code='2889',
        symbol-'TSE2889',
        name=' ',
        category='17',
        unit=1808,
        linit_go=15.5,
        lufatt_dow=15.5,
        update_date='2823/01/12',
        day_trade=CayTrade.Yes: 'Yes'>
        },
        order=Order(
        action=*Action Buy: 'Buy'>,
        price=17,
        quantity=3,
        id='531e27ar',
        sorde="00001",
        account_Account(
        account_Account(
        account_Eype=-AccountType_Stock: 'S'>,
              person_id=Al2346788',
              broke_id='04895',
              account_id='123567',
              signed=True

custon_fald='test'.

price_type=-StockPriceType_LNT: 'LNT'>,
              order_order_type=Not: 'StockPriceType_LNT: 'LNT'>,
```

### **GET FUTURES TRADES**

### **Cet Futures Trades**

api.update\_status(api.futopt\_account)
api.list\_trades()

```
&at
```

```
Trade(
              contract=Future(
                     code='TXFA3',
symbol='TXF202301',
name=' 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'
                     action=<Action.Buy: 'Buy'>,
price=14400,
                     quantity=3,
id='5efffde1',
seqno='000004',
ordno='000003',
account=Account(
                            unit-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='5efffdel',
status=<Status.Filled: 'Filled'>,
                      status\_code='00',\\ order\_datetime=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

### # 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

### 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.

```
der Event
           version>=1.0
                                                                                                                version<1.0
OrderState.StockOrder {
                         rState.StockOrder {
  'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
            'op_mus,
},
'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': {
                                       tract': {
  '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': '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,
                          'contract': {
                                    ract': {
  'security_type': 'STK',
  'exchange': 'TSE',
  'code': '2890',
  'symbol': '',
  'name': '',
  'currency': 'TWD'
```

### 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 MarginTrading: ShortSelling: order\_lot (str): { Common: Fixing: 0dd: Intraday0dd: custom\_field (str): memo field 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

### Deal CallBack

contract

security\_type (str): security type
exchange (str): exchange
code (str): code id
symbol (str): symbol
name (str): name

currency (str): currency

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 <code>id</code> in the order callback to the trade id in the deal callback.

### leal Event

```
version<1.0
               version>=1.0
                             erState.StockDeal {
  'trade_id': '9c6ae2eb',
  'seqno': '269866',
  'ordno': 'IN497',
  'exchange_seq': '669915',
  'broker_id': '9A95',
  'account_id': '1234567',
  'acction': 'Buy',
  'code': '2890',
  'order_cond': 'Cash',
  'order_lot': 'IntradayOdd',
  'price': 267.5,
  'quantity': 3,
  'web_id': '137',
  'custom_field': 'test',
  'ts': 1673577256.354
OrderState.StockDeal {
  OrderState.TFTDeal {
    'trade_id': '9c6ae2eb',
    'seqno': '269866',
    'ordno': '1N497',
    'exchange_seq': '669915',
    'broker_id': '9A95',
    'account_id': '1234567',
    'action': 'Buy',
    'code': '2899',
    'order_cond': 'Cash',
    'order_lot': 'IntradayOdd',
    'price': 267.5,
    'quantity': 3,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1673577256.354
                                'ts': 1673577256.354
```

### Deal Callback Info. trade\_id (str): same as the id in StockOrder seqno (str): sequence number ordno (str): sequence number. 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): (Ruv. Sell) action (str): {Buy, Sell} code (str): code id order\_cond (str): { MarginTrading: ShortSelling: order\_lot (str): { Common: Fixing: Odd: IntradayOdd: } price (float or int): deal price quantity (int): deal quantity web\_id (str): web id custom\_field (str): memo field ts (int): deal timestamp

### **A**te

you "may" recieve 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.

### 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 version<1.0

```
OrderState.FuturesOrder {
                                rstate.FuturesOrder {
'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
                'op_mus,
},
'order': {
   'id': 'fcb42a6e',
   'seqno': '585886',
   'ordno': '00',
   'account': {
        'account_type': 'F',
        'person_id': '',
        'broker_id': 'F02000',
        'account_id': '1234567',
        'signed': True
}.
                                               'action': 'Buy',
'price': 14000.0,
'quantity': 1,
'order_type': 'ROD',
'price_type': 'LMT',
'market_type': 'Night',
'oc_type: 'New',
'subaccount': '',
'combo': Balse
                                                'combo': False
                   };
'status': {
    'id': 'fcb42a6e',
    'exchange_ts': 1673512283.0,
    'modified_price': 0.0,
    'cancel_quantity': 0,
    'order_quantity': 1,
    'web_id': 'Z'
                    'web__.
},
'contract': {
   'security_type': 'FUT',
   'code': 'TXF',
   'exchange': 'TM',
   'delivery_month': '202301',
   'delivery_date': '',
   'strike_price': 0.0,
   'option_right': 'Future'
       }
OrderState.FOrder {
    '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': 14000.0,
'quantity': 1,
'order_type': 'ROD',
'price_type': 'LMT',
'market_type': 'New',
'subaccount': '',
'combo': False
                                                'combo': False
                             },
'status': {
    'id': 'fcb42a6e',
    'exchange_ts': 1673512283.0,
    'modified_price': 0.0,
    'cancel_quantity': 0,
    'order_quantity': 1,
    'web_id': 'Z'
                               },
'contract': {
                                              htract': {
  'security_type': 'FUT',
  'code': 'TXF',
  'exchange': 'TIM',
  'delivery_month': '202301',
  'delivery_date': '',
  'strike_price': 0.0,
  'option_right': 'Future'
```

### 

### 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 <code>id</code> in the order callback to the <code>trade\_id</code> in the deal callback.

### leal Event

```
version>=1.0 version<1.0

OrderState.FuturesDeal {
    'trade_id': '46e0f0f0',
    'seqn0: '458955',
    'ordn0: '558.6,
    'quantity': 1,
    'ssubaccount: '7',
    'security.type: '0PT',
    'delivery_month: '202301',
    'strike_price': 13490.0,
    'ordn0: '51896,
    'ts: 1673270852.0
}
OrderState.FDeal {
    'trade_id': '46e0f0f6',
    'seqn0: '459955.00,
    'ordn0: '51896,
    'to: '1508536',
    'account_id': '1234567',
    'account_id': '1234567',
    'account_id': '1234567',
    'action1: '58e1',
    'code': '700',
    'price': 58.0,
    'quantity': 1,
    'sequiry.type: '0PT',
    'delivery.monthi: '202301',
    'strike_price': 13490.0,
    'option_fight: '0ptionhut',
    'ansket_type: '0ptionhut',
    'ansket_type: '0ptionhut',
    'ansket_type: '0ptionhut',
    'ansket_type: '0ptionhut',
    'ansket_type: '0ptionhut',
    'canbot: False,
    'ts: 1672270852.0
}</pre>
```

# 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 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): {Puture, OptionCall, OptionPut} market\_type (str): {Day, Night} ts (int): deal timestamp



you "may" recieve 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.

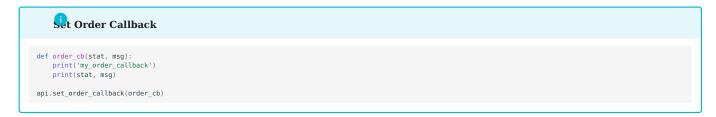
### 4.6 CallBack

### 4.6.1

Each time you place\_order, update\_order or cancel\_order, by default, you will recieve an order event (or deal event) from exchange. If you don't want recieve 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.



ORDER EVENT

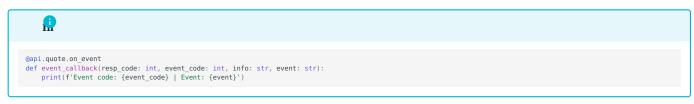
```
Ader Event
       version>=1.0
                                                                                                        version<1.0
  my_order_callback
OrderState.StockOrder {
                 'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
                  },
'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': {
   'cocurity_
                                 ntract': {
  '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',
    'segno': '267677',
    'ordno': 'IM394',
    'account': {
        'account_type': 'S',
        'person_id': '',
        'broker_id': '9A95',
        'account_id': '1234567',
        'signed': True
},
                                    'action': 'Buy',
'price': <mark>16.0</mark>,
                                   'quantity': 1,
'order_type': 'ROD',
'price_type': 'LMT',
'order_cond': 'Cash',
'order_lot': 'Common',
'custom_field': 'test'
                 },
'status': {
    'id': '97b63e2f',
    'exchange_ts': 10
                                 'id': '97h63e2f',
'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**

# version>=1.0 version<1.0 my\_order\_callback OrderState\_StackDeal { 'trade\_id': "966m2eb', 'seqno': '269866', 'orden': '269866', 'orden': '26986', 'broker\_id': "9055, 'account\_id': '129567', 'exchange\_seq': '66915', 'broker\_id': '9056', 'order\_cond': 'Cash', 'order\_cond': 'Cash', 'order\_cond': 'Cash', 'unantity': 3, 'web\_id': '137', 'custom\_faled': 'test', 't's': 'E075977256.354 } my\_order\_callback OrderState\_FFDeal { 'trade\_id': '966m2eb', 'seqno': '269866', 'orden': '18497', 'exchange\_seq': '669915', 'broker\_id': '90577256.354 } purple (Cash', 'order\_cond': 'Cash', 'order\_fond': 'Cash',

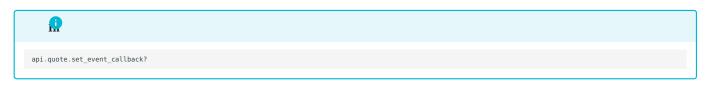
### 4.6.2

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.



```
Event code: 16 | Event: Subscribe or Unsubscribe ok
```

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



```
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 unsucces
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 remov
5	SOLCLIENT_SESSION_EVENT_RX_MSG_TOO_BIG_ERROR	The API discarded a received message that exceede Session buffer size.
6	SOLCLIENT_SESSION_EVENT_ACKNOWLEDGEMENT	The oldest transmitted Persistent/Non-Persistent methat has been acknowledged.
7	SOLCLIENT_SESSION_EVENT_ASSURED_PUBLISHING_UP	Deprecated see notes in solClient_session_startAssuredPublishing.The AD House (that is, Guaranteed Delivery handshake) has complete publisher and Guaranteed messages can be sent
8	SOLCLIENT_SESSION_EVENT_ASSURED_CONNECT_FAILED	Deprecated see notes in solClient_session_startAssuredPublishing.The applied rejected the AD Handshake to start Guaranteed published by the SOLCLIENT_SESSION_EVENT_ASSURED_DELIVED instead.
8	SOLCLIENT_SESSION_EVENT_ASSURED_DELIVERY_DOWN	Guaranteed Delivery publishing is not available. The guaranteed delivery capability on the session has be 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 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 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 recorattempt is in progress.
13	SOLCLIENT_SESSION_EVENT_RECONNECTED_NOTICE	The automatic reconnect of the Session was success 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 succeed
17	SOLCLIENT_SESSION_EVENT_VIRTUAL_ROUTER_NAME_CHANGED	The appliance's Virtual Router Name changed during reconnect operation. This could render existing quest temporary topics invalid.
18	SOLCLIENT_SESSION_EVENT_MODIFYPROP_OK	The session property modification completed.

 ${\tt SOLCLIENT\_SESSION\_EVENT\_MODIFYPROP\_FAIL}$ 

19

The session property modification failed.

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

### 4.7

### 4.7.1

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: 'Fetched'>,
    acc_balance=100000.0,
    date='2023-01-06 13:30:00.000000',
    errmsg=''
)
```

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

### 4.7.2

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_margin=0.0,
mantenance_margin=0.0,
margin_call=0.0,
risk_indicator=99.0,
royalty_revenue_expenditure=0.0,
equity_e000.0,
equity_e000.0,
equity_amount=6000.0,
option_openbuy_market_value=0.0,
option_openbuy_market_value=0.0,
option_openbuy_market_value=0.0,
option_openbuy_market_value=0.0,
option_openbuy_market_value=0.0,
option_openbuy_market_value=0.0,
option_openbuoition=0.0,
option_openbuoition=0.0,
future_open_position=0.0,
future_open_position=0.0,
future_settle_profitloss=0.0,
available_margin=0.0000.0,
plus_margin_o0,
plus_margin_o0,
plus_margin_o0,
collateral_amount=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): fee
tax (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(float): equity
equity(float): equity
equity(float): equity
equity(float): equiton
potion_openbuy_market_value (float): value of option openbuy market
option_open_position (float): profit loss of open option
potion_settle_profitloss (float): profit loss of settle option
future_open_position (float): profit loss of osen future
today_future_open_position (float): profit loss of osen future
future_settle_profitloss (float): profit loss of oteday open future
future_settle_profitloss (float): profit loss of settle future
available_margin (float): available_margin
plus_margin (float): plus margin
plus_margin_float): plus margin
plus_margin_float): indicator of plus margin
security_collateral_amount (float): amount of security collateral
order_margin_premium (float): amount of collateral
```

### 4.7.3

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
```



```
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(), Netting(), MarginTrading(), ShortSelling(), Emerging()}
    margin_purchase_amount (int): margin_purchase_amount
    collateral (int): collateral
    short_sale_margin (int): short_sale_margin
    interest
```

### Odd Stocks

The unit is the number of shares.

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

```
{
    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**

 ${\tt account}$  is defaulted as a Stock account, and if you want to query the Futures or Options content, you need to bring in the  ${\tt futopt\_account}$ .



```
FuturePosition(
    id=0,
    code='TX201370J2',
    direction=<Action.Buy: 'Buy'>,
    quantity=3,
    price=131.0000,
    last_price=126.0,
    pnl=-750.00
}
```

#### To DataFrame



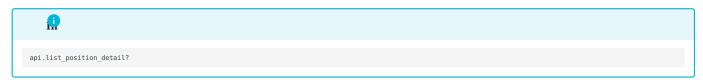


```
id (int): position id
code (str): contract id
direction (Action): action
{Buy, Sell}
quantity (int): quantity
price (float): the average price
last_price (float): last price
pnl (float): unrealized profit
```

#### **Position Detail**

Using the result obtained from list\_positions, bring the id into detail\_id to query the details of that position.

#### STOCKS



```
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):
        choice 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

#### To DataFrame



 $\label{eq:df} \begin{array}{l} df = pd.DataFrame(pnl.\_dict\_\_for\ pnl\ in\ position\_detail) \\ df \end{array}$ 



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

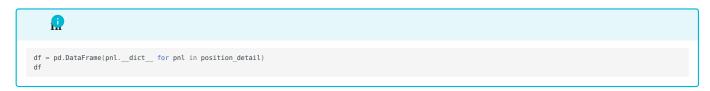
#### **FUTURES AND OPTIONS**



 $position\_detail = api.list\_position\_detail(api.futopt\_account, \ \textbf{0}) \\ position\_detail$ 

```
FuturePositionDetail(
    date='2023-02-14',
    code='MXFG3',
    quantity=1,
    price=15611.0,
    last_price=15541.0,
    dseq='tA0n8',
    direction=<Action.Buy: 'Buy'>,
    pnl=-3500.0,
    currency=<Currency.TWID: 'TWID'>,
    entry_quantity=1
    )
}
```

#### To DataFrame





```
code (str): contract id
date (str): trade date
quantity (int): quantity
price (float): price
last_price (float): last price
dseq (str): detail seqno no
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
```

#### 4.7.4

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 = '',
        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)
```

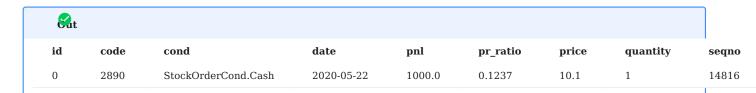
Enter the time interval you want to query. begin\_date is the start time, and end\_date is the end time. unit is the quantity unit, where Common represents whole shares and Share represents fractional shares.

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

```
[
    StockProfitLoss(
        id=0,
        code='2890',
        seqno='14816',
        dseq='ID111',
        quantity=1,
        price=10.1,
        pn1=1234.0,
        pr_ratio=0.1237,
        cond='Cash',
        date='2020-05-22'
    )
]
```

#### To DataFrame

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



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

```
id (int): use to find detail
code (str): contract id
quantity (int): quantity
pnl (float): profit and loss
date (str): trade date
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):
        choice 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='1X000',
        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





```
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}
```

```
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_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):
        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.

```
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='2889',
        quantity=2800,
        entry_price=17,
        cover_price=10,
        pnl=-11585.0,
        currency='NTD',
        entry_cost=34550,
        cover_cost=21680,
        buy_cost=33112,
        sell_cost=21527,
        pr_ratio=-34.99
        )
        // total=ProfitLossTotal(
        quantity=2800,
        buy_cost=33112,
        sell_cost=21527,
        pr_ratio=-34.99
        )
    }
}
```

#### To DataFrame

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



### **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 tover
buy\_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

#### 4.7.5

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
```

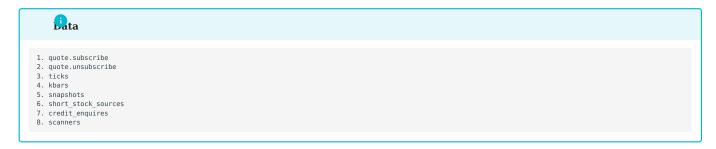
```
date (datetime.date): date of Tday
amount (float): settlement amount
T (int): Tday
```

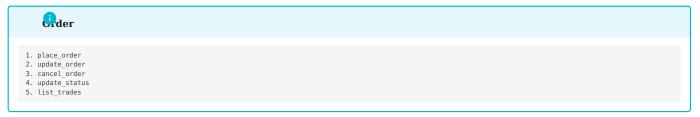
# 4.8

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.



#### 4.8.1 Available APIs







#### 4.9

#### 4.9.1

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** 

```
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)
```

#### **PUSH QUOTE TO REDIS STREAM**

Before start, please install redis first. Below example shows how to push quote massages 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 = '0;' + tick.code # = '0;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'.
"("code": "TXF61", "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'.
]
      ]
# parse redis stream # [json.loads(x[-1]['tick']) for x in r.xread({'0:TXFG1':'0-0'})[0][-1]]
                'code': 'TXFG1'
               'datetime': '2021-07-05T11:15:49.066000',
'open': '17755',
'underlying_price': '17904.03',
               'underlying_price': '17904.0'
bid_side_total_vol': 49698,
'ask_side_total_vol': 51490,
'avg_price': '17851.312322',
'close': '17889',
'high': '17918',
'low': '17742',
'amount': '268335',
'total_argunt': '1309210810',
'total_argunt': '1309210810',
               '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',
               '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,
'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.

# Rample: stop order import time from typing import Union import shioaji as sj class StopOrderExcecutor: 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']: 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</pre> 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}") self. stop orders[code] def add\_stop\_order( contract: sj.contracts.Contract, stop\_price: float, order: sj.order.Order, ) -> None: code = contract.code stode = 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 = { p\_order = { "code": contract.code, "stop\_price": stop\_price, "ref\_price": ref\_price, "contract": contract,

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

"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}")

if code in self.\_stop\_orders:
 \_ = self.\_stop\_orders.pop(code)

code = stop\_order["code"]
if code in self.\_stop\_orders:

def cancel\_stop\_order\_by\_code(self, code: str) -> None:

def cancel\_stop\_order(self, stop\_order: dict) -> None:

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()

def get\_stop\_orders(self) -> dict:
 return self.\_stop\_orders

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.

```
# 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 Excecutor
soe = StopOrderExcecutor(api)
soe.add_stop_order(contract=contract, stop_price=14805, order=order)
```

• Stop-Market Order: price\_type = 'MKT'

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

```
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 Excecutor
    self.on_quote(tick)

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

#### at: Once close/mid price hit stop price

#### 4.9.2

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.FuturesPriceType.LMT,
    order_type=sj.constant.OrderType.ROD,
    octype=sj.constant.FuturesOCType.Auto,
    account=api.futopt_account
)
trade = api.place_order(contract, order, timeout=0)
trade
```

```
Sat
contract=Future(
     symbol='TXF202301',
     name=' 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=<mark>1</mark>,
     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 sequo 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': de616839',
        'segno': '500009',
        'ordno': '600009',
        'action': '58Elt',
        'price type': '4000,
        'quantity': 1,
        'order_type': 'Nou',
        'price type': 'Will',
        'oc_type': 'Auto',
        'custom filed': '''
    },
    'status': {
        'id': 'de616839',
        'exchange ts': 1673334371.492948,
        'order_quantity': 1,
        'modified price': 0,
        'cancel_quantity': 0,
        'web_ld': ''Z'
    },
    'contract': {
        'security_type': 'FUT',
        'exchange ts': TATEEK',
        'code': 'TXFAS'
}
```

#### 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
)
```

# at: place order callback \_\_my\_callback\_\_ contract=Future( code='TXFA3' symbol='TXF202301', name=' 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' id='40fd85d6', seqno='958433', ordno='kY01g', account=FutureAccount( person\_id='F123456789', broker\_id='F002000', account\_id='1234567', signed=True, username='PAPIUSER' price\_type=<StockPriceType.LMT: 'LMT'>, order\_type=<0rderType.ROD: 'ROD'> status=OrderStatus( status=<Status.Submitted: 'Submitted'>, status\_code=' ', order\_datetime=datetime.datetime(2023, 01, 10, 15, 14, 32),

#### 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 = api.Contracts.Futures.TXF['TXF202301']
order = api.Order(
    action='Sell',
    price=14000,
    quantity=1,
    price_type='LMT',
    order_type='ROD',
    octype=s].constant.FuturesOCType.Auto,
    account=api.futopt_account
}
```

```
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
```

```
start_time = time.time()
api.place_order(contract, order, timeout=0) # non-block, the order is in transmition (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\_summary
   settlements
   margin
   ticks
   kbars

#### 4.9.3

#### **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.

Complete TouchPrice Order Extention can be found here.

#### 4.9.4

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.



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 multicommodity technical indicator calculations.

# 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 = 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.icks\_stk\_v1: List[sj.TickSTKv1] = [] self.ticks\_fop\_v1: List[sj.TickSTKv1] = [] 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  $on_stk_vl_tick_handler$  and  $on_fop_vl_tick_handler$ , these two functions will be called when receiving the quote data, and the quote data will be stored in  $ticks_stk_vl_handler$  and  $ticks_handler$  and  $ticks_han$ 

# and QuoteManager subscribe and unsubscribe methods 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 <code>subscribe\_stk\_tick</code> method, this method will add the commodity codes in the incoming commodity code list to the <code>subscribed\_stk\_tick</code>, and call the <code>subscribe</code> method of Shioaji to subscribe to the market, <code>subscribed\_stk\_tick</code> is a <code>Set</code>, used to store the commodity codes that have been subscribed to avoid duplicate subscriptions and facilitate subsequent unsubscribing all subscribed commodities.

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

# def get\_df\_stk\_kbar( self, unit: str = "lm", exprs: List[pl.Expr] = [] ) -> pl.DataFrame: df = self.get\_df\_stk() df = df.group\_by{ pl.col("adetime").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").max().alias("low"), pl.col("price").last().alias("clow"), pl.col("price").stax().alias("clow"), pl.col("volume").sum().alias("volume"), ) if exprs: df = df.with\_columns(exprs) return df

in <code>get\_df\_stk\_kbar</code> method, we will use <code>get\_df\_stk</code> method to get the Ticks DataFrame and then group the data by truncated <code>datetime</code> and <code>code</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 <code>exprs</code> 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.

```
import polars as pl
import polars_talib as plta

quote_manager.get_df_stk_kbar("5m", [
    pl.col("close").ta.ema(5).over("code").fill_nan(None).alias("ema5"),
    plta.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 <code>over("code")</code> to group the data by commodity code for independent calculation of each commodity, so even if all commodities are in the same <code>DataFrame</code>, 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 <code>alias</code> to set the field name of the calculation result as <code>ema5</code>, in the <code>macd</code> indicator, the return is a struct with multiple fields, and this part gets the <code>macd</code> 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.

#### and QuoteManager backfill missed ticks method

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.

 $\label{thm:condition} \mbox{Version 1.0 is a major release. This document assist users migrating to version 1.0.}$ 

# 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
       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:
       Ouote
        Contracts
       0rder
Init docstring:
initialize Shioaji to start trading
Args:
        simulation (bool):
       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:
        activate ca
        list_accounts
       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:
       Ouote
        Contracts
       Order
Init docstring:
initialize Shioaji to start trading
       backend (str): {http, socket}
               use http or socket as backend currently only support http, async socket backend coming soon.
        simulation (bool):
       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",
)</pre>
```

```
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

```
SockOrder
      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,
           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_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
     ) -> 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 = ''
             sequo: str = '',
ordno: str = '',
account: shioaji.account.Account = None,
custom_field: shioaji.order.ConstrainedStrValue = '',
ca: str = '',
            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'>,
```

#### 5.3.1 Order

#### Rename

- TFTStockPriceType to StockPriceType
- TFTOrderType to OrderType
- TFTStockOrderLot to StockOrderLot
- first\_sell to daytrade\_short, and type changed to Bool.

```
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.IFTStockPriceType.LMT,
    order_type=sj.constant.IFTStockPriceType.LMT,
    order_type=sj.constant.IFTTStockPriceType.ROD,
    order_tot=sj.constant.IFTTOrderType.ROD,
    order_tot=sj.constant.TFTStockOrderLot.Common,
    first_sell=sj.constant.StockFirstSell.Yes,
    custom_field="test",
    account=api.stock_account")
}</pre>
```

#### 5.3.2 Order Callback

Rename TFTOrder to StockOrder

```
der Callback
         version>=1.0
                                                                                                      version<1.0
OrderState.StockOrder {
                      rState.StockOrder {
  'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
                   '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': 'LMT'
},
               },
'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': ''
                  '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': 'LMT'
},
              },
'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

```
Version>=1.0 version<1.0

OrderState.StockDeal {
    'trade_id': 'Izaba456',
    'exchange_seq: 'Izab456',
    'broker_id': 'your_broker_id',
    'account_id': 'your_account_id',
    'action': Action.Buy: 'Buy'>,
    'code': '2889',
    'order_cond': -StockOrderCod.Cash: 'Cash'>,
    'order_cond': -StockOrderCod.Cash: 'Cash'>,
    'order_bid': 'Izab.
    'yrice': 12,
    'quantify': 10,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1583828972
}

OrderState.TFTDeal {
    'trade_id': 'Izab3456',
    'exchange_seq: 'Iza456',
    'broker_id': 'your_broker_id',
    'account_id': 'your_account_id',
    'action': -Action.Buy: 'Buy'>,
    'code': '2889',
    'order_cond': -StockOrderCod.Cash: 'Cash'>,
    'order_cond': -StockOrderLot.Common: 'Common'>,
    'price': 12,
    'quantify': 10,
    'web_id': '137',
    'custom_field': 'test',
    'ts': 1583828972
}
```

#### 5.4 Futures Order

```
RturesOrder
    verion>=1.0
                                       verion<1.0
 >> sj.order.FuturesOrder?
  Init signature:
sj.order.FuturesOrder(
        action: shioaii.constant.Action.
        action: shloajl.constant.Action,
price: Union[pydantic.types.StrictInt, float],
quantity: shloaji.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 = '',
        price_type: shioaji.constant.FuturesPriceType,
order_type: shioaji.constant.FuturesOrderType,
        octype: shioaji.constant.FuturesOCType = <FuturesOCType.Auto: 'Auto'>,
```

#### 5.4.1 Order

Rename FuturesOrderType to OrderType

```
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.FuturesOCType.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.FuturesOrderType.ROD,
    octype=sj.constant.FuturesOrderType.ROD,
    account=api.futopt_account
)</pre>
```

#### 5.4.2 Order Callback

Rename FOrder to FuturesOrder

```
der Event
           version>=1.0
                                                                                                           version<1.0
OrderState.FuturesOrder {
                       'operation': {
    'op_type': 'New',
    'op_code': '00',
    'op_msg': ''
                     op_msg :
},
'order': {
    'id': '02c347f7',
    'seqno': '956201',
    'ordno': 'kY00H',
    'action': 'Sell',
    'price': 17760.0,
    'quantity': 1,
    'order_cond': None,
    'order_type': 'RDD',
    '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': {
                                     htract': {
  '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': ''
                   'op_msg': ''
},
'order': {
    'id': '02c347f7',
    'seqno': '956201',
    'ordno': 'kY00H',
    'action': 'Sell',
    'price': 17760.0,
    'quantity': 1,
    'order_cond': None,
    'order_type': 'R0D',
    '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"
                               '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

#### 5.5 Market Data



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

#### 5.5.1 Callback

Tick

# 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)
```



#### 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('597'), amount=Decimal('590000'), total_amount=Decimal('3540101000'), 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_oddlot_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

#### OuoteVersion.v1 OuoteVersion.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], 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], 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], ask_price=[Decimal('589'), Decimal('590'), Decimal('591'), Decimal('591'), Decimal('591'), Decimal('591'), Decimal('591'), Decimal('592'), Decimal('591'), Deci
```

#### 5.6 Future Account Info.

#### Remove functions

```
    get_account_margin
    get_account_openposition
    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 )
```

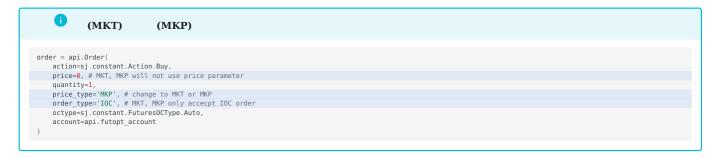
For more information, please refer to Account Data section.

Finally, give us support and encouragement on GITHUB

github

6.

#### 6.0.1





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.

```
api.Contracts.Stocks.TSE['TSE2330']
```

```
Stock(
    exchange=<Exchange.TSE: 'TSE'>,
    code='2330',
    symbol='TSE2330',
    name=' ',
    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.

```
contract = api.Contracts.Stocks.TSE['TSE2330']
price = contract.limit_up
order = api.Order(
    action=5j.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



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
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

# Account not acceptable

- [ ](https://sinotrade.github.io/zh\_TW/tutor/prepare/terms/#\_1) [API ](https://sinotrade.github.io/zh\_TW/tutor/prepare/terms/#api) [`update\_status`](../tutor/order/UpdateStatus)



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



#### contracts

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



3

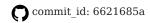
\*\* Note that you only have 2 chances to unlock your account online in a day. \*\*

\*\* We've migrate QA site to Shioaji Forum \*\*

# 7.

# 7.1 version: 1.2.5 (2024-10-01)

• feat: refactor expire time of CA



release at: 2024-10-01 02:25:01.723

## 7.2 version: 1.2.4 (2024-08-28)

• feat: support py3.12

commit\_id: a287f56c

release\_at: 2024-08-28 16:00:00.000

# 7.3 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

\$\frac{1}{2}\$ release at: 2024-03-06 16:00:00.000

#### 7.4 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

# 7.5 version: 1.2.1 (2023-12-22)

• fix: windows inject dll issue

commit\_id: 2a413848

7 release\_at: 2023-12-22 01:19:17.043

# 7.6 version: 1.2.0 (2023-12-20)

- feat: vpn
- feat: rust version ca

 $\bullet$  refactor: test report flow

commit\_id: 856f39ea

\$ release at: 2023-12-20 16:00:00.000

# 7.7 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

# 7.8 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

# 7.9 version: 1.1.11 (2023-08-04)

- fix: custom field in validator for only support number and alphabet
- fix: pydantic v2 trade issue
- $\bullet$  fix: pydantic v2 contracts cache issue
- commit\_id: cc1da47e

release\_at: 2023-08-04 08:00:37.000

#### 7.10 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

# 7.11 version: 1.1.9 (2023-07-20)

#### yanked

commit\_id: f9f03cff

7 release\_at: 2023-07-20 07:43:46.000

# 7.12 version: 1.1.8 (2023-07-18)

- feat: query usage
- feat: profit\_loss support unit
- feat: support pydantic v2
- commit\_id: 3dc8568e

release\_at: 2023-07-18 09:08:42.000

# 7.13 version: 1.1.7 (2023-07-18)

#### yanked

commit\_id: fb490a9a

f
release\_at: 2023-07-18 07:02:20.000

# 7.14 version: 1.1.6 (2023-06-19)

- feat: solace reconnect with sub 2500 user
- commit\_id: eeaeeb5f

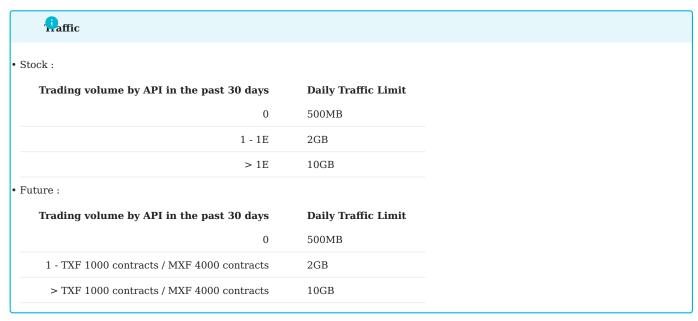
  /-
  release at: 2023-06-19 16:00:00.000

# 7.15 version: 1.1.5 (2023-06-07)

- fix: simulation sign check
- fix: handle pickle load error
- commit\_id: 1def691c

release\_at: 2023-06-07 16:00:00.000

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







## Qunts

• 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.



- 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