pybotters

MtkN1XBt

2021 年 11 月 10 日

Contents:

第 1 章

Getting Started

1.1 Installation

pybotters をお使いの環境に pip install しましょう。

```
pip install pybotters
```

注釈: パッケージアップデートの際はこちらのコマンドを利用してください。

```
pip install --upgrade pybotters
```

インストールできたらまずはコード内で pybotters パッケージを import しましょう！

```
import pybotters
```

1.2 API Client

pybotters の API クライアントは非同期 I/O のクラスです。 pybotters.Client クラスを非同期関数の中からコンテキストマネージャーで開いてインスタンスを生成しましょう。

```
async def main():
    async with pybotters.Client() as client:
        ...

asyncio.run(main())
```

自動 API 認証機能を利用するためには引数 apis に API キー・シークレットの情報を渡します。 apis の情報を渡した取引所は REST/WebSocket API を叩いた際、ホスト名を判別し自動で認証情報の付与が行われます。

```
apis = {
    'bybit': ['BYBIT_API_KEY', 'BYBIT_API_SECRET'],
    'binance': ['BINANCE_API_KEY', 'BINANCE_API_SECRET'],
    '...': ['...', '...'],
}

async def main():
    async with pybotters.Client(apis=apis) as client:
        ...

asyncio.run(main())
```

または、API 情報を JSON 形式で保存している場合、ディレクトリパスを渡すことで読み込むことが可能です。

api.json

```
{
    "bybit": ["BYBIT_API_KEY", "BYBIT_API_SECRET"],
    "binance": ["BINANCE_API_KEY", "BINANCE_API_SECRET"],
    "....": ["...", "..."]
}
```

```
async def main():
    async with pybotters.Client(apis='apis.json') as client:
        ...
```

各取引所に対応する apis のキー名は、こちらの表から設定してください。

Exchange	apis Key Name
Bybit	bybitbybit_testnet
Binance	binancebinance_testnet
FTX	ftx
Phemex	phemexphemex_testnet
BitMEX	bitmexbitmex_testnet
bitFlyer	bitflyer
GMO Coin	gmocoin
Liquid	liquid
bitbank	bitbank

1.3 REST API

REST API を利用するためには request, get, post, put, delete メソッドがあります。 いずれも非同期なので await で呼び出してください。

```
async def main():
    async with pybotters.Client(apis=apis) as client:
        r = await client.request('GET', 'https://...')
        r = await client.get('https://...', params={'foo': 'bar'})
        r = await client.post('https://...', data={'foo': 'bar'})
        r = await client.put('https://...', data={'foo': 'bar'})
        r = await client.delete('https://...', data={'foo': 'bar'})
```

注釈: HTTP リクエストの特性上、GET メソッドの場合は引数 params にパラメーター (クエリストリング) を指定します。それ以外の HTTP メソッドは引数 data にパラメーター (リクエストボディ) を指定します。

戻り値はライブラリ aiohttp.ClientResponse のインターフェースです。status プロパティで HTTP ステータスを取得できます。 json, text メソッドでレスポンスボディを取得できます。

その他のインターフェースの詳細は aiohttp のリファレンスを確認してください。

```
async def main():
    async with pybotters.Client(apis=apis) as client:
        r = await client.get('https://...', params={'foo': 'bar'})
        print(r.status)
        data = await r.json()
        print(data)
```

クライアントクラスの生成時に引数 base_url を指定しておくことでホスト名の省略が可能です。単一の取引所のみ利用する場合に便利です。 ※ base_url は WebSocket(ws_connect メソッド) の URL には適応しません。

以下は Bybit で利用する例です。

```
async def main():
    async with pybotters.Client(apis=apis, base_url='https://api.bybit.com') as client:
        r = await client.get('/v2/private/order', params={'symbol': 'BTCUSD'})
        r = await client.post('/v2/private/order/create', data={'symbol': 'BTCUSD', ...:␣
↪...})
```

クライアントクラスの生成時に引数 headers を指定しておくことでデフォルトヘッダーの指定が可能です。リクエストメソッドでも上書きで使用できます。例えば FTX のサブアカウントを利用する場合に便利です。

```
async def main():
    async with pybotters.Client(apis=apis, base_url='https://ftx.com/api', headers={'FTX-
↪SUBACCOUNT': 'my_subaccount_nickname'}) as client:
        r = await client.get('...')
        r = await client.get('...', headers={'FTX-SUBACCOUNT': 'my_alt_subaccount_
↪nickname'})
```

1.4 WebSocket API

WebSocket API を利用するためには ws_connect メソッドを利用します。メソッドは非同期なので await で呼び出してください。

```
async def main():
    async with pybotters.Client(apis=apis) as client:
        wstask = await client.ws_connect('wss://...')
```

引数 send_json, hdlr_json にそれぞれ接続時に送信するメッセージオブジェクト、受信したメッセージを処理するハンドラ関数を指定します。文字列で処理したい場合は send_str, hdlr_str を指定します。また、接続時に複数のメッセージを送信したい場合はリスト形式のデータを引数に指定します。send_json, hdlr_json どちらも指定していない場合はデフォルトで hdlr_json に pybotters.print_handler が設定され WebSocket で受信したメッセージが表示されます。

```
async def main():
    async with pybotters.Client(apis=apis) as client:
        wstask = await client.ws_connect(
            'wss://...',
            send_json={'foo': 'bar'},
            hdlr_json=pybotters.print_handler,
            # OR string
            # send_str='{"foo":"bar"}',
            # hdlr_str=pybotters.print_handler,
            # OR Multiple request
            # send_json=[{'foo': 'bar'}, {'baz': 'foobar'}],
            # send_str=['{"foo": "bar"}', '{"baz": "foobar"}'],
        )
        await wstask
```

戻り値は asyncio.Task です。開始した WebSocket タスクではコネクション切断時は自動的に再接続が行われるので、基本的には戻り値のタスクに対して操作する必要はありません。

注釈: 上記のコードを実行しても main ルーチンでは WebSocket 接続後何も処理がないためプログラムは終了してしまい、受信メッセージは print されません。(※通常であればこのあとに bot ロジックを記載するでしょう。) そこで ws_connect の戻り値は無限ループタスクなので、それを利用して await wstask とすることでプログラムの終了を防ぎハンドラの動作を確認することができます。これは pybotters で bot ロジックではなく WebSocket アプリケーションを作成する際に便利です。

1.5 DataStore

pybotters は各取引所の WebSocket で受信したメッセージを処理して扱いやすい形式で保管する DataStore クラスを実装しています。上記では単純な print ハンドラを利用しましたが、オーダー管理・ポジション自炊など本格的に WebSocket のデータを扱いたい場合は DataStore クラスのハンドラを利用しましょう。

WebSocket のデータ形式は取引所ごとに違うのでそれぞれ別のクラスを実装しています。以下は Bybit でオーダーを監視する例です。

```
async def main():
    async with pybotters.Client(apis=apis) as client:
        store = pybotters.BybitDataStore()
        wstask = await client.ws_connect(
            'wss://stream.bybit.com/realtime',
            send_json={
                'op': 'subscribe',
                'args': ['order'],
            },
            hdlr_json=store.onmessage,
        )
        # Ctrl+C to break
        while True:
            await store.wait()
            print(store.order.find())
```

上記を段階を踏んで解説しましょう。 まず最初にデータストアマネージャー クラスを生成します。このマネージャークラスは複数の データストアを持っており、いわゆる複数のテーブルを持つデータベースのようなものです。

```
store = pybotters.BybitDataStore()
```

生成したデータストアマネージャーの onmessage 関数は WebSocket 用のハンドラです。 クライアントの ws_connect メソッドの引数 hdlr_json に渡します。WebSocket 接続後、受信データがデータストアで処理され

るようになります。

```
await client.ws_connect(
    ...,
    hdlr_json=store.onmessage,
)
```

データストアには辞書のようにしてアクセスすることができます。取引所モデルによってはメンバ変数として定義してあります。

```
# dictionary access
store['order']
# member access
store.order
```

データストアマネージャー及びデータストアクラスは wait メソッドで WebSocket メッセージの受信があるまで待機することができます。

データストアマネージャーの wait メソッドは WebSocket で何かメッセージを受信するまで待機します。データストアの wait メソッドはそのストアに関するメッセージを受信するまで待機します。

上記の例ではオーダーしかトピックを購読していないので await store.wait() で受信を待機しています。

```
# onmessage wait
await store.wait()
# order store wait
await store.order.wait()
```

データストアは get メソッドと find メソッドでデータを参照することができます。

get メソッドは引数にデータストアのキーを指定し、一意のアイテムを取得することができます。データストアのキーは _keys メンバで確認できます。

find メソッドは引数に指定した辞書に部分一致する全てのアイテムをリストで取得することができます。指定しない場合はデータストアの全てのアイテムを取得します。

```
print(store.order._keys)
# ['order_id']

print(store.order.get({'order_id': 'aabbccdd'}))
# {'order_id': 'aabbccdd', 'symbol': 'BTCUSD', 'side': 'Buy', ...: ...}

print(store.order.get({'order_id': 'zzzzzzzz'}))
```

(次のページに続く)

(前のページからの続き)

```
# None

print(store.order.find({'symbol': 'BTCUSD', 'side': 'Buy'}))
# [
#     {'order_id': 'aabbccdd', 'symbol': 'BTCUSD', 'side': 'Buy', ...: ...},
#     {'order_id': 'eeffgghh', 'symbol': 'BTCUSD', 'side': 'Buy', ...: ...},
# ]

print(store.order.find({'order_id': 'zzzzzzzz'}))
# []

print(store.order.find())
# [
#     {'order_id': 'aabbccdd', 'symbol': 'BTCUSD', 'side': 'Buy', ...: ...},
#     {'order_id': 'eeffgghh', 'symbol': 'BTCUSD', 'side': 'Buy', ...: ...},
#     {'order_id': 'iijjkkll', 'symbol': 'BTCUSD', 'side': 'Sell', ...: ...},
# ]
```

第 2 章

Example

このページでは pybotters を利用した bot の実装例を紹介します。

2.1 Bybit インバース無期限契約

2.1.1 低頻度 bot のサンプル

ローソク足 (1 分足) を元にトレードを行う bot

```
import asyncio
import time

import pybotters



# apis
apis = {
    'bybit': ['BYBIT_API_KEY', 'BYBIT_API_SECRET']
}

async def main():
    async with pybotters.Client(apis=apis, base_url='https://api.bybit.com') as client:
        # 必要な初期処理
        ...

        # メインループ
        while True:
            # REST API データ並列リクエスト
```

(次のページに続く)

(前のページからの続き)

```
            resps = await asyncio.gather(
                client.get('/v2/public/kline/list', params={
                    'symbol': 'BTCUSD', 'interval': 1, 'from': int(time.time()) - 3600
                }),
                client.get('/v2/private/order', params={'symbol': 'BTCUSD'}),
                client.get('/v2/private/position/list', params={'symbol': 'BTCUSD'}),
            )
            kline, order, position = await asyncio.gather(*[r.json() for r in resps])

            # シグナル計算
            """
            something awesome logic...
            """
            cond = 'Whether to execute...'
            side = 'Buy or Sell...'
            qty = 'Calculated value...'

            # オーダー執行
            if cond:
                await client.post('/v2/private/order/create', data={
                    'symbol': 'BTCUSD',
                    'side': side,
                    'order_type': 'Market',
                    'qty': qty,
                    # 'price': price,
                    'time_in_force': 'GoodTillCancel',
                })

            # 待機(60秒)
            await asyncio.sleep(60.0)


# 非同期メイン関数を実行(Ctrl+Cで終了)
if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

2.1.2 高頻度 bot の場合

板情報を元にトレードを行う bot

```
import asyncio

import pybotters



# apis
apis = {
    'bybit': ['...', '...'],
}

async def main():
    async with pybotters.Client(apis=apis, base_url='https://api.bybit.com') as client:
        # データストアのインスタンスを生成する
        store = pybotters.BybitDataStore()

        # REST API 由来のデータ（オーダー・ポジション・残高）を初期データとしてデータストアに挿入する
        await store.initialize(
            client.get('/v2/private/order', params={'symbol': 'BTCUSD'}),
            client.get('/v2/private/position/list', params={'symbol': 'BTCUSD'}),
            client.get('/v2/private/wallet/balance', params={'symbol': 'BTCUSD'}),
        )

        # WebSocket 接続
        wstask = await client.ws_connect(
            'wss://stream.bybit.com/realtime',
            send_json={'op': 'subscribe', 'args': [
                'orderBookL2_25.BTCUSD',
                'trade.BTCUSD',
                'instrument_info.100ms.BTCUSD',
                'position',
                'execution',
                'order',
            ]},
            hdlr_json=store.onmessage,
        )
```

(次のページに続く)

(前のページからの続き)

```
        # WebSocket でデータを受信するまで待機
        while not all([
            len(store.orderbook),
            len(store.instrument),
        ]):
            await store.wait()

        # その他必要な初期処理
        ...

        # メインループ
        while True:
            # データ参照
            orderbook = store.orderbook.find()
            order = store.order.find()
            position = store.position_inverse.find()

            # シグナル計算
            """
            something awesome logic...
            """
            cond = 'Whether to execute...'
            side = 'Buy or Sell...'
            qty = 'Calculated value...'
            price = 'Amazing price...'

            # オーダー執行
            if cond:
                # 高頻度では重複オーダーしないようにオーダー後 WebSocket でデータ受信するまで待機させる
                # REST の応答より WebSocket のイベントの方が速い可能性があるので先にイベント待機タスクをスケジュールする
                event = asyncio.create_task(store.order.wait())
                await client.post('/v2/private/order/create', data={
                    'symbol': 'BTCUSD',
                    'side': side,
                    'order_type': 'Limit',
                    'qty': qty,
                    'price': price,
```

(次のページに続く)

(前のページからの続き)

```
                    'time_in_force': 'GoodTillCancel',
                })
                await event

            # 板情報のイベントまで待機
            await store.orderbook.wait()


# 非同期メイン関数を実行 (Ctrl+C で終了)
if __name__ == '__main__':
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        pass
```

第 3 章

Advanced Usage

3.1 apis の暗黙的な読み込み

pybotters.Client の引数 apis を指定せず以下のように暗黙的な読み込みが可能です。

3.1.1 カレントディレクトリに apis.json を配置する

Python の実行カレントディレクトリに apis.json という名前のファイルを配置することで自動的にそのファイルを読み込ます。

```
# bash
$ ls
apis.json
```

注釈: カレントディレクトリは Python 標準の os モジュールの getcwd で確認できます。

3.1.2 環境変数 PYBOTTERS_APIS にファイルパスを設定する

環境変数 PYBOTTERS_APIS に同様の形式の JSON ファイルパス設定することでそのファイルを読み込みます。カレントディレクトリ以外に配置している場合に有効です。

```
# bash
$ export PYBOTTERS_APIS=/path/to/apis.json
```

3.1.3 優先順位

以下のような優先順位で、下位のものは設定があっても無視されます。

引数 apis の明示的な指定 ＞ カレントディレクトリの apis.json ＞ 環境変数 PYBOTTERS_APIS

3.2 同期リクエスト

pybotters は requests ライブラリのようにパッケージトップレベルでリクエスト関数をサポートしています。この関数は非同期ではないので await で呼び出す必要はありません。またはレスポンスボディの取得にも await を付ける必要はありません。

上記の apis の暗黙的な読み込みと合わせることで、REPL(対話モード) で pybotters を利用する際に便利です。

```
r = pybotters.request('GET', 'https://...', apis=apis)
r = pybotters.get('https://...', params={'foo': 'bar'}, apis=apis)
r = pybotters.post('https://...', data={'foo': 'bar'}, apis=apis)
r = pybotters.put('https://...', data={'foo': 'bar'}, apis=apis)
r = pybotters.delete('https://...', data={'foo': 'bar'}, apis=apis)

print(r.text())
print(r.json())
```

注釈: 内部的には pybotters.Client の非同期関数を同期的に実行しています。リクエスト毎にセッションは閉じられる為 keep-alive 接続ではありません。

第 4 章

Exchanges

それぞれの取引所での固有の用法を記載します。

4.1 Bybit

4.1.1 Initialize

Bybit の WebSocket はオーダー・ポジション等の初期値が送信されないので REST での初期化が必要です。

初期化するにはデータストアの `initialize` メソッドを実行します。引数にはリクエストのコルーチンを可変長で渡します。

```
async def main():
    async with pybotters.Client(apis=apis, base_url='https://api.bybit.com') as client:
        store = pybotters.BybitDataStore()
        await store.initialize(
            # Inverse
            client.get('/v2/private/order', params={'symbol': 'BTCUSD'}),
            client.get('/v2/private/position/list', params={'symbol': 'BTCUSD'}),
            # USDT
            client.get('/private/linear/order/search', params={'symbol': 'BTCUSDT'}),
            client.get('/private/linear/position/list', params={'symbol': 'BTCUSDT'}),
            client.get('/v2/private/wallet/balance', params={'coin': 'USDT'}),
        )
        # Inverse
        wstask = await client.ws_connect(
            'wss://stream.bybit.com/realtime',
            send_json={'op': 'subscribe', 'args': ['position', 'order']},
```

(次のページに続く)

(前のページからの続き)

```
            hdlr_json=store.onmessage,
        )
        # USDT
        wstask = await client.ws_connect(
            'wss://stream.bybit.com/realtime_private',
            send_json={'op': 'subscribe', 'args': ['position', 'order', 'wallet']},
            hdlr_json=store.onmessage,
        )
```

対応エンドポイント

- オーダー
  - /v2/private/order
  - /private/linear/order/search
  - /futures/private/order
- 条件付きオーダー
  - /v2/private/stop-order
  - /private/linear/stop-order/search
  - /futures/private/stop-order
- ポジション
  - /v2/private/position/list
  - /private/linear/position/list
  - /futures/private/position/list
- 残高
  - /v2/private/wallet/balance

4.2 FTX

4.2.1 Subaccount

サブアカウントで取引を行う場合はヘッダーにサブアカウントの情報を指定します。(FTX API ドキュメント)

pybotters.Client またはリクエストメソッドの引数 headers に FTX-SUBACCOUNT を指定してください。

```
async def main():
    async with pybotters.Client(apis=apis, base_url='https://ftx.com/api', headers={'FTX-
↪SUBACCOUNT': 'my_subaccount_1'}) as client:
        # REST
        r = await client.get('/positions') # equal my_subaccount_1
        r = await client.get('/positions', headers={'FTX-SUBACCOUNT': 'my_subaccount_2'})
        # WebSocket
        wstask = await client.ws_connect('wss://ftx.com/ws/', send_json=...) # equal my_
↪subaccount_1
        wstask = await client.ws_connect('wss://ftx.com/ws/', send_json=..., headers={
↪'FTX-SUBACCOUNT': 'my_subaccount_2'})
```

注釈: 本来の FTX の仕様では WebSocket はヘッダーを指定するものではありませんが、pybotters がそれを判定して自動的に WebSocket の認証メッセージにサブアカウントを付与して送信しています。

4.2.2 Initialize

FTX の WebSocket はオーダーの初期値が送信されないので REST での初期化が必要です。

初期化するにはデータストアの initialize メソッドを実行します。引数にはリクエストのコルーチンを可変長で渡します。

また、データストアにポジションを用意していますが、FTX の WebSocket にはポジション情報ありません。その代わりに initialize メソッドにポジションのエンドポイントを渡すことで、WebSocket の fills チャンネル受信時に REST API を自動フェッチしてデータストアを更新する機能が有効化されます。(※ fills チャンネルを購読してください。) FTX の REST API 制限は 30 回/1 秒 と非常に緩いですが、制限が気になる場合はこの機能は利用しないでください。

```
async def main():
    async with pybotters.Client(apis=apis, base_url='https://ftx.com/api') as client:
        store = pybotters.FTXDataStore()
```

(次のページに続く)

(前のページからの続き)

```
        await store.initialize(
            client.get('/orders', params={'market': 'BTC-PERP'}),
            client.get('/positions', params={'showAvgPrice': 'true'}),
        )
        wstask = await client.ws_connect(
            'wss://ftx.com/ws/',
            send_json=[
                {'op': 'subscribe', 'channel': 'orders'},
                {'op': 'subscribe', 'channel': 'fills'},
            ],
            hdlr_json=store.onmessage,
        )
```

対応エンドポイント

- オーダー
  - /orders
  - /conditional_orders
- ポジション
  - /positions ※ fills 受信時の自動フェッチを有効化する

4.2.3 DataStore

FTX のデータストアは WebSocket から受信したデータ形式からデータストアとして扱いやすい形式に加工して格納しています。特に板情報のデータストアは以下の様に加工しています。※それ以外のデータストアは market 名を付与している程度です。

Example:

```
# input data format
{
    'asks': [[4114.25, 6.263]],
    'bids': [[4112.25, 49.29]]
}
# store.orderbook.find()
[
    {'market': 'BTC-PERP', 'side': 'sell', 'price': 4114.25, 'size': 6.263},
```

(次のページに続く)

(前のページからの続き)

```
    {'market': 'BTC-PERP', 'side': 'buy', 'price': 4112.25, 'size': 49.29}
]
```

4.3 BitMEX

4.3.1 Initialize

BitMEX の WebSocket は全ての必要な初期値が送信されます。その都合上、データストアの生成も WebSocket で初期データを受信したタイミングに行われます。(受信されるまでは None になります。)

```
async def main():
    async with pybotters.Client() as client:
        store = pybotters.BitMEXDataStore()
        wstask = await client.ws_connect(
            'wss://www.bitmex.com/realtime',
            send_json={'op': 'subscribe', 'args': ['orderBookL2_25:XBTUSD']},
            hdlr_json=store.onmessage,
        )
        print(type(store.orderbook))
        # None
        while store.orderbook is None:
            await store.wait()
        print(type(store.orderbook))
        # <class 'pybotters.store.DataStore'>
```

4.4 bitbank

4.4.1 WebSocket

bitbank の WebSocket API は Socket.IO によって実装されています 。pybotters の aiohttp 基盤なので Socket.IO クライアントには対応していません。とは言っても高レベルな Socket.IO クライアントを利用できないだけで大枠の WebSocket という規格としては同じです。なのでデータの送受信プロトコルを低レベルで記述することで接続が可能です。

サンプルコード (任意で while True 内のコメントアウトを変更してください)

```
async def main():
    async with pybotters.Client() as client:
        store = pybotters.bitbankDataStore()
        wstask = await client.ws_connect(
            'wss://stream.bitbank.cc/socket.io/?EIO=3&transport=websocket',
            send_str=[
                '42["join-room","ticker_xrp_jpy"]',
                '42["join-room","transactions_xrp_jpy"]',
                '42["join-room","depth_whole_xrp_jpy"]',
            ],
            hdlr_str=store.onmessage,
        )
        while True:
            # Transactions
            # await store.transactions.wait()
            # pybotters.print(store.transactions.find()[-1])

            # Depth
            await store.depth.wait()
            pybotters.print({k:v[:6] for k, v in store.depth.sorted().items()})

            # Ticker
            # await store.ticker.wait()
            # pybotters.print(store.ticker.find())
```

ポイントは - URL に `EIO=3&transport=websocket` といったクエリパラメーターを付与すること - 購読は `send_str` 引数を利用して配列形式の文字列に `42` プレフィックスを付けること - DataStore のハンドラは `hdlr_str` 引数を利用すること

これによって Socket.IO プロトコルで WebSocket に接続でき、DataStore のハンドラも Socket.IO の解釈に対応している為データを保管できます。

4.5 GMO Coin

4.5.1 WebSocket

GMO コインの WebSocket のチャンネル購読は 1 秒間 1 回が上限という制限があります。

pybotters の通常の仕様は *send_json* または *send_str* の送信メッセージは非同期でリクエストされます。しかし GMO コインにおいては上記のような制限があり正常にチャンネルを購読できなくなる為、pybotters は GMO コイ

ンの WebSocket を自動的に判別して制限を回避します。メッセージを送信する時に非同期処理のロックし 1 秒間の待機が行われます。またこの際正確な 1 秒間を判別する為に GMO コインの Public API `GET /public/v1/status` でサーバータイムをメッセージ送信毎にリクエストします。

この仕組みについてユーザー側で操作は不要ですが、購読するチャンネル数ごとに 1 秒間待機が発生することと、Public API のリクエストが発生することに注意してください。

第 5 章

Contributing

pybotters はオープンソースソフトウェアですので、どなたでも開発に参加できます。当プロジェクトに参加する方法や選定しているツールなどの環境構築する方法を記載します。

5.1 Table of Content

5.2 Python

pybotters は Python 3.7 以上を対象としています。 可能な限り最新の Python 3.7 でコーディングを行ってください。

5.2.1 参考 Python 3.7 のインストール～仮想環境の作成方法

```
# 1. Install build dependencies.
# See pyenv Wiki
# https://github.com/pyenv/pyenv/wiki#suggested-build-environment

# 2. Get latest Python 3.7 source, extract it.
wget https://www.python.org/ftp/python/3.7.11/Python-3.7.11.tgz
# See here for the latest Python 3.7.
# https://www.python.org/downloads/source/
tar -xf Python-3.7.11.tgz

# 3. Build, Install.
cd Python-3.7.11
./configure --prefix=${HOME}/.local && make && make altinstall

# 4. Create virtual environment, activate it.
~/.local/bin/python3.7 -m venv ~/.venv/pybotters
. ~/.venv/pybotters/bin/activate
```

5.3 poetry

poetry は Python の依存関係管理とパッケージングを支援するツールです。仮想環境に開発ライブラリをインストールする為に使用します。

5.3.1 参考 poetry のインストール～依存関係インストール

```
# 1. Install poetry
# See https://python-poetry.org/docs/#installation

# 2. Install dependencies.
# Activate venv.
```

(次のページに続く)

(前のページからの続き)

```
. ~/.venv/pybotters/bin/activate
# Clone pybotters
git clone https://github.com/MtkN1/pybotters
cd pybotters
# Install.
poetry install
```

5.4 Formatter, Linter

当プロジェクトではフォーマッターは black, リンターは flake8 を採用しています。(これらは poetry によってインストールされます。)

コードをコミットする際にはこれらを適用してください。

5.4.1 参考 black の適用方法

```
# 手動で適用する場合
balck .

# VS Code で自動適用を利用する場合
# .vscode/settings.json を編集
# {
#     "python.formatting.provider": "black",
#     "editor.formatOnSave": true
# }
```

5.4.2 参考 flake8 の適用方法

```
# 手動でチェックする場合 (確認後、コードを修正してください)
flake8 .

# VS Code で自動チェックする場合
# .vscode/settings.json を編集
# {
#     "python.linting.flake8Enabled": true,
#     "python.linting.enabled": true,
```

(次のページに続く)

(前のページからの続き)

```
#     "python.linting.pylintEnabled": false
# }
```

5.5 Testing

当プロジェクトではテストに pytest を採用しています。(ライブラリは poetry によってインストールされます。)

実装したロジックに対するテストコードを作成してください。また、テストは GitHub Actions によってリモートリポジトリコミット時に自動実行されます。

5.5.1 テストの基準

- 現状、DataStore に関するテストコードは省いています。(※開発速度優先の為。正式版リリースまでには対応する予定)
- それ以外の部分についてはテストを追加してください。
- 外部との通信部分はモック化してください。

5.5.2 参考 pytest の実行方法

```
pytest
```

5.6 Pull Request

pybotters を Fork して、 develop ブランチを元にコードを作成してください。 Pull Request は同ブランチに対して行ってください。またコミットメッセージはできれば、「英文〜〜〜(#存在する関連イシュー番号)」で行ってください。

5.6.1 参考 クローン〜develop ブランチチェックアウト

```
git clone https://github.com/[YourAccountName]/pybotters
git checkout develop
```

設計思想や細かい変数名のデザインなどは、レビューし修正コードを提案します。お気軽にプルリクください！

OSS 開発にご興味がある方、是非プロジェクトにご参加ください âIJÍð§ŊřâIJÍ

5.7 Incentive(?)

将来的には取引所への Referral リンク等作成する予定です。多大な貢献をして頂いた方には Referral の収益分配するような事を行うかもしれません。(かなり先の展望です)

第 6 章

API Reference

pybotters

6.1 pybotters

Functions

`delete`(url, *[, data, apis])

`get`(url, *[, params, apis])

`post`(url, *[, data, apis])

`print_handler`(msg, ws)

`put`(url, *[, data, apis])

`request`(method, url, *[, params, data, apis])

`ws_connect`(url, *[, send_str, send_json, ...])

Classes

SyncClientResponse(method, url, *, writer, ...)

6.1.1 pybotters.SyncClientResponse

class pybotters.**SyncClientResponse**(*method: str*, *url: yarl.URL*, *, *writer: asyncio.Task[None]*, *continue100: Optional[asyncio.Future[bool]]*, *timer: aiohttp.helpers.BaseTimerContext*, *request_info: aiohttp.client_reqrep.RequestInfo*, *traces: List[Trace]*, *loop: asyncio.events.AbstractEventLoop*, *session: ClientSession*)

Methods

`__init__`(method, url, *, writer, ...)	
`close`()	
`get_encoding`()	
json(*args, **kwargs)	Read and decodes JSON response.
`raise_for_status`()	
`read`()	Read response payload.
`release`()	
`start`(connection)	Start response processing.
text(*args, **kwargs)	Read response payload and decode.
`wait_for_close`()	

Attributes

ATTRS	
charset	The value of charset part for Content-Type HTTP header.
closed	
connection	
content	
content_disposition	
content_length	The value of Content-Length HTTP header.
content_type	The value of content part for Content-Type HTTP header.
headers	
history	A sequence of of responses, if redirects occurred.
host	
links	
ok	Returns True if status is less than 400, False if not.
raw_headers	
real_url	
reason	
request_info	
status	
url	

次のページに続く

表 5 – 前のページからの続き

url_obj
version

json(**args*, ***kwargs*) → Any

Read and decodes JSON response.

text(**args*, ***kwargs*) → str

Read response payload and decode.

Modules

pybotters.auth
pybotters.client
pybotters.models
pybotters.request(method, url, *[, params, ...])
pybotters.store
pybotters.typedefs
pybotters.ws

6.1.2 pybotters.auth

Classes

Auth()
Hosts()

次のページに続く

表 7 – 前のページからの続き

Item(name, func)

pybotters.auth.Auth

class pybotters.auth.**Auth**

Methods

__init__()
binance(args, kwargs)
bitbank(args, kwargs)
bitflyer(args, kwargs)
bitmex(args, kwargs)
bybit(args, kwargs)
coincheck(args, kwargs)
ftx(args, kwargs)
gmocoin(args, kwargs)
liquid(args, kwargs)
phemex(args, kwargs)

pybotters.auth.Hosts

class pybotters.auth.**Hosts**

Methods

__init__()

Attributes

items

pybotters.auth.Item

class pybotters.auth.**Item**(*name: 'str'*, *func: 'Any'*)

Methods

__init__(name, func)

Attributes

name
func

6.1.3 pybotters.client

Classes

Client([apis, base_url])	HTTP リクエストクライアントクラス

pybotters.client.Client

class pybotters.client.**Client**(*apis: Optional[Union[dict[str, list[str]], str]] = None*, *base_url: str = ''*, ***kwargs: Any*)

HTTP リクエストクライアントクラス

注釈: 引数 apis は省略できます。

Example

```
async def main():
    async with pybotters.Client(apis={'example': ['KEY', 'SECRET']}) as client:
        r = await client.get('https://...', params={'foo': 'bar'})
        print(await r.json())
```

```
async def main():
    async with pybotters.Client(apis={'example': ['KEY', 'SECRET']}) as client:
        wstask = await client.ws_connect(
            'wss://...',
            send_json={'foo': 'bar'},
            hdlr_json=pybotters.print_handler
            )
        await wstask
        # Ctrl+C to break
```

Basic API

パッケージトップレベルで利用できる HTTP リクエスト関数です。 これらは同期関数です。 内部的に pybotters.Client をラップしています。

Example

```
r = pybotters.get(
        'https://...',
        params={'foo': 'bar'},
        apis={'example': ['KEY', 'SECRET']}
    )
print(r.text())
print(r.json())
```

```
pybotters.ws_connect(
        'wss://...',
        send_json={'foo': 'bar'},
        hdlr_json=pybotters.print_handler,
        apis={'example': ['KEY', 'SECRET']}
    )
# Ctrl+C to break
```

パラメータ

- **apis** -- API キー・シークレットのデータ (optional) ex: {'exchange': ['key', 'secret']}
- **base_url** -- リクエストメソッドの url の前方に自動付加する URL(optional)
- ****kwargs** -- aiohttp.Client.request に渡されるキーワード引数 (optional)

Methods

`__init__`([apis, base_url])	param apis API キー・シークレットのデータ (optional) ex: {'exchange': ['key', 'secret']}
`close`()	
`delete`(url, *[, data])	
`get`(url, *[, params])	
`post`(url, *[, data])	

次のページに続く

表 14 – 前のページからの続き

put(url, *[, data])
request(method, url, *[, params, data]) param method　GET, POST, PUT, DELETE などの HTTP メソッド
ws_connect(url, *[, send_str, send_json, ...]) param url　WebSocket URL

request(*method: str*, *url: str*, ***, *params: Optional[Mapping[str, str]] = None*, *data: Optional[Any] = None*, ***kwargs: Any*) → aiohttp.client._RequestContextManager

パラメータ

- **method** -- GET, POST, PUT, DELETE などの HTTP メソッド
- **url** -- リクエスト URL
- **params** -- URL のクエリ文字列 (optional)
- **data** -- リクエストボディ (optional)
- **headers** -- リクエストヘッダー (optional)
- **auth** -- API 自動認証の機能の有効/無効。デフォルトで有効。auth=None を指定することで無効になります (optional)
- **kwargs** -- aiohttp.Client.request に渡されるキーワード引数 (optional)

async ws_connect(*url: str*, ***, *send_str: Optional[Union[str, list[str]]] = None*, *send_json: Any = None*, *hdlr_str: Optional[WsStrHandler] = None*, *hdlr_json: Optional[WsJsonHandler] = None*, ***kwargs: Any*) → asyncio.Task

パラメータ

- **url** -- WebSocket URL
- **send_str** -- WebSocket で送信する文字列。文字列、または文字列のリスト形式 (optional)
- **send_json** -- WebSocket で送信する辞書オブジェクト。辞書、または辞書のリスト形式 (optional)

- **hdlr_str** -- WebSocket の受信データをハンドリングする関数。第 1 引数 msg に _str_型, 第 2 引数 ws に WebSocketClientResponse 型の変数が渡されます (optional)
- **hdlr_json** -- WebSocket の受信データをハンドリングする関数。第 1 引数 msg に Any 型 (JSON-like), 第 2 引数 ws に WebSocketClientResponse 型の変数が渡されます (optional)
- **headers** -- リクエストヘッダー (optional)
- **auth** -- API 自動認証の機能の有効/無効。デフォルトで有効。auth=None を指定することで無効になります (optional)
- ****kwargs** -- aiohttp.ClientSession.ws_connect に渡されるキーワード引数 (optional)

6.1.4 pybotters.models

Modules

pybotters.models.binance

pybotters.models.bitbank

pybotters.models.bitflyer

pybotters.models.bitmex

pybotters.models.bybit

pybotters.models.experimental

pybotters.models.ftx

pybotters.models.gmocoin

pybotters.models.binance

Classes

Balance([keys, data, auto_cast])	
BinanceDataStore([auto_cast])	Binance のデータストアマネージャー (※ v0.4.0: Binance Futures USDâŞĹ-M のみ)
BookTicker([keys, data, auto_cast])	
ContinuousKline([keys, data, auto_cast])	
Kline([keys, data, auto_cast])	
Liquidation([keys, data, auto_cast])	
MarkPrice([keys, data, auto_cast])	
Order([keys, data, auto_cast])	
OrderBook([keys, data, auto_cast])	
Position([keys, data, auto_cast])	
Ticker([keys, data, auto_cast])	
Trade([keys, data, auto_cast])	

pybotters.models.binance.Balance

class pybotters.models.binance.**Balance**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])	
`find`([query])	
`get`(item)	
`wait`()	

pybotters.models.binance.BinanceDataStore

class pybotters.models.binance.**BinanceDataStore**(*auto_cast: bool = False*)

Binance のデータストアマネージャー (※ v0.4.0: Binance Futures USDâŞĹ-M のみ)

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

`balance`	
`bookticker`	

次のページに続く

表 19 – 前のページからの続き

continuouskline	
kline	
liquidation	
markprice	
order	アクティブオーダーのみ (約定・キャンセル済みは削除される)
orderbook	
position	
ticker	
trade	

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /fapi/v1/depth (DataStore: orderbook)
  - Binance API ドキュメントに従って WebSocket 接続後に initialize すること。
  - orderbook データストアの initialized が True になる。
- GET /fapi/v2/balance (DataStore: balance)
- GET /fapi/v2/positionRisk (DataStore: position)
- GET /fapi/v1/openOrders (DataStore: order)
- POST /fapi/v1/listenKey (Property: listenkey)
  - プロパティ listenkey に listenKey が格納され 30 分ごとに PUT /fapi/v1/listenKey のリクエストがスケジュールされる。

property order: *Order*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

pybotters.models.binance.BookTicker

class pybotters.models.binance.**BookTicker**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.ContinuousKline

class pybotters.models.binance.**ContinuousKline**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.Kline

class pybotters.models.binance.**Kline**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.Liquidation

class pybotters.models.binance.**Liquidation**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.MarkPrice

class pybotters.models.binance.**MarkPrice**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.Order

class pybotters.models.binance.**Order**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.OrderBook

class pybotters.models.binance.**OrderBook**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.binance.Position

class pybotters.models.binance.**Position**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.Ticker

class pybotters.models.binance.**Ticker**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.binance.Trade

class pybotters.models.binance.**Trade**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitbank

Classes

Depth([keys, data, auto_cast])	
Ticker([keys, data, auto_cast])	
Transactions([keys, data, auto_cast])	
bitbankDataStore([auto_cast])	

pybotters.models.bitbank.Depth

class pybotters.models.bitbank.**Depth**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])	
find([query])	
get(item)	
sorted([query])	
wait()	

pybotters.models.bitbank.Ticker

class pybotters.models.bitbank.**Ticker**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitbank.Transactions

class pybotters.models.bitbank.**Transactions**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitbank.bitbankDataStore

class pybotters.models.bitbank.**bitbankDataStore**(*auto_cast: bool = False*)

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

`depth`
`ticker`
`transactions`

pybotters.models.bitflyer

Classes

Board([keys, data, auto_cast])
ChildOrderEvents([keys, data, auto_cast])
ChildOrders([keys, data, auto_cast])

次のページに続く

表 36 – 前のページからの続き

Executions([keys, data, auto_cast])
ParentOrderEvents([keys, data, auto_cast])
ParentOrders([keys, data, auto_cast])
Positions([keys, data, auto_cast])
Ticker([keys, data, auto_cast])
bitFlyerDataStore([auto_cast])

pybotters.models.bitflyer.Board

class pybotters.models.bitflyer.**Board**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.bitflyer.ChildOrderEvents

class pybotters.models.bitflyer.**ChildOrderEvents**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.ChildOrders

class pybotters.models.bitflyer.**ChildOrders**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.Executions

class pybotters.models.bitflyer.**Executions**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.ParentOrderEvents

class pybotters.models.bitflyer.**ParentOrderEvents**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.ParentOrders

class pybotters.models.bitflyer.**ParentOrders**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.Positions

class pybotters.models.bitflyer.**Positions**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bitflyer.Ticker

class pybotters.models.bitflyer.**Ticker**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])	
`find`([query])	
`get`(item)	
`wait`()	

pybotters.models.bitflyer.bitFlyerDataStore

class pybotters.models.bitflyer.**bitFlyerDataStore**(*auto_cast: bool = False*)

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
`initialize`(*aws)	
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

board
childorderevents
childorders
executions
parentorderevents
parentorders
positions
ticker

pybotters.models.bitmex

Classes

BitMEXDataStore([auto_cast])	BitMEX のデータストアマネージャー

pybotters.models.bitmex.BitMEXDataStore

class pybotters.models.bitmex.**BitMEXDataStore**(*auto_cast: bool = False*)

BitMEX のデータストアマネージャー

Methods

__init__([auto_cast])
create(name, *[, keys, data, datastore_class])

次のページに続く

表 48 – 前のページからの続き

get(name, type)	
onmessage(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
wait()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

execution	
funding	
instrument	
insurance	
liquidation	
margin	
order	アクティブオーダーのみ (約定・キャンセル済みは削除される)
orderbook	
position	
quote	
trade	
wallet	

property order: *pybotters.store.DataStore*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

pybotters.models.bybit

Classes

BybitDataStore()	Bybit のデータストアマネージャー
Execution([keys, data, auto_cast])	
Instrument([keys, data, auto_cast])	
Insurance([keys, data, auto_cast])	
Kline([keys, data, auto_cast])	
Liquidation([keys, data, auto_cast])	
Order([keys, data, auto_cast])	
OrderBook([keys, data, auto_cast])	
PositionInverse([keys, data, auto_cast])	
PositionUSDT([keys, data, auto_cast])	
StopOrder([keys, data, auto_cast])	
Trade([keys, data, auto_cast])	
Wallet([keys, data, auto_cast])	

pybotters.models.bybit.BybitDataStore

class pybotters.models.bybit.**BybitDataStore**

Bybit のデータストアマネージャー

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

`execution`	
`instrument`	
`insurance`	
`kline`	
`liquidation`	
order	アクティブオーダーのみ (約定・キャンセル済みは削除される)
`orderbook`	
position_inverse	インバース契約 (無期限/先物) 用のポジション
position_usdt	USDT 契約用のポジション
stoporder	アクティブオーダーのみ (トリガー済みは削除される)
`trade`	
`wallet`	

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /v2/private/order (DataStore: order)
- GET /private/linear/order/search (DataStore: order)
- GET /futures/private/order (DataStore: order)
- GET /v2/private/stop-order (DataStore: stoporder)
- GET /private/linear/stop-order/search (DataStore: stoporder)
- GET /futures/private/stop-order (DataStore: stoporder)
- GET /v2/private/position/list (DataStore: position_inverse)
- GET /futures/private/position/list (DataStore: position_inverse)
- GET /private/linear/position/list (DataStore: position_usdt)
- GET /v2/private/wallet/balance (DataStore: wallet)

property order: *Order*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

property position_inverse: *PositionInverse*

インバース契約 (無期限/先物) 用のポジション

property position_usdt: *PositionUSDT*

USDT 契約用のポジション

property stoporder: *StopOrder*

アクティブオーダーのみ (トリガー済みは削除される)

pybotters.models.bybit.Execution

class pybotters.models.bybit.**Execution**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Instrument

class pybotters.models.bybit.**Instrument**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Insurance

class pybotters.models.bybit.**Insurance**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Kline

class pybotters.models.bybit.**Kline**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Liquidation

class pybotters.models.bybit.**Liquidation**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Order

class pybotters.models.bybit.**Order**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.OrderBook

class pybotters.models.bybit.**OrderBook**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.bybit.PositionInverse

class pybotters.models.bybit.**PositionInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`getboth`(symbol)
`getone`(symbol)
`wait`()

pybotters.models.bybit.PositionUSDT

class pybotters.models.bybit.**PositionUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`getboth`(symbol)
`wait`()

pybotters.models.bybit.StopOrder

class pybotters.models.bybit.**StopOrder**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Trade

class pybotters.models.bybit.**Trade**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.bybit.Wallet

class pybotters.models.bybit.**Wallet**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental

Modules

pybotters.models.experimental.bybit	

pybotters.models.experimental.bybit

Classes

BybitInverseDataStore([auto_cast])	Bybit Inverse 契約のデータストアマネージャー
BybitUSDTDataStore([auto_cast])	Bybit USDT 契約のデータストアマネージャー
ExecutionInverse([keys, data, auto_cast])	
ExecutionUSDT([keys, data, auto_cast])	
InstrumentInverse([keys, data, auto_cast])	
InstrumentUSDT([keys, data, auto_cast])	
Insurance([keys, data, auto_cast])	
KlineInverse([keys, data, auto_cast])	
KlineUSDT([keys, data, auto_cast])	
LiquidationInverse([keys, data, auto_cast])	
LiquidationUSDT([keys, data, auto_cast])	
OrderBookInverse([keys, data, auto_cast])	
OrderBookUSDT([keys, data, auto_cast])	
OrderInverse([keys, data, auto_cast])	
OrderUSDT([keys, data, auto_cast])	

次のページに続く

表 66 – 前のページからの続き

PositionInverse([keys, data, auto_cast])	
PositionUSDT([keys, data, auto_cast])	
StopOrderInverse([keys, data, auto_cast])	
StopOrderUSDT([keys, data, auto_cast])	
TradeInverse([keys, data, auto_cast])	
TradeUSDT([keys, data, auto_cast])	
Wallet([keys, data, auto_cast])	

pybotters.models.experimental.bybit.BybitInverseDataStore

class pybotters.models.experimental.bybit.**BybitInverseDataStore**(*auto_cast: bool = False*)

Bybit Inverse 契約のデータストアマネージャー

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

execution	
instrument	
insurance	
kline	
liquidation	
order	アクティブオーダーのみ (約定・キャンセル済みは削除される)
orderbook	
position	インバース契約 (無期限/先物) 用のポジション
stoporder	アクティブオーダーのみ (トリガー済みは削除される)
trade	

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /v2/private/order (DataStore: order)
- GET /futures/private/order (DataStore: order)
- GET /v2/private/stop-order (DataStore: stoporder)
- GET /futures/private/stop-order (DataStore: stoporder)
- GET /v2/private/position/list (DataStore: position)
- GET /futures/private/position/list (DataStore: position)

property order: *OrderInverse*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

property position: *PositionInverse*

インバース契約 (無期限/先物) 用のポジション

property stoporder: *StopOrderInverse*

アクティブオーダーのみ (トリガー済みは削除される)

pybotters.models.experimental.bybit.BybitUSDTDataStore

class pybotters.models.experimental.bybit.**BybitUSDTDataStore**(*auto_cast: bool = False*)

Bybit USDT 契約のデータストアマネージャー

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

`execution`	
`instrument`	
`kline`	
`liquidation`	
order	アクティブオーダーのみ (約定・キャンセル済みは削除される)
`orderbook`	
position	USDT 契約用のポジション
stoporder	アクティブオーダーのみ (トリガー済みは削除される)

次のページに続く

表 70 – 前のページからの続き

trade
wallet

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /private/linear/order/search (DataStore: order)
- GET /private/linear/stop-order/search (DataStore: stoporder)
- GET /private/linear/position/list (DataStore: position)

property order: *OrderUSDT*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

property position: *PositionUSDT*

USDT 契約用のポジション

property stoporder: *StopOrderUSDT*

アクティブオーダーのみ (トリガー済みは削除される)

pybotters.models.experimental.bybit.ExecutionInverse

class pybotters.models.experimental.bybit.**ExecutionInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

pybotters.models.experimental.bybit.ExecutionUSDT

class pybotters.models.experimental.bybit.**ExecutionUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.InstrumentInverse

class pybotters.models.experimental.bybit.**InstrumentInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.InstrumentUSDT

class pybotters.models.experimental.bybit.**InstrumentUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.Insurance

class pybotters.models.experimental.bybit.**Insurance**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.KlineInverse

class pybotters.models.experimental.bybit.**KlineInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.KlineUSDT

class pybotters.models.experimental.bybit.**KlineUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.LiquidationInverse

class pybotters.models.experimental.bybit.**LiquidationInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.LiquidationUSDT

class pybotters.models.experimental.bybit.**LiquidationUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.OrderBookInverse

class pybotters.models.experimental.bybit.**OrderBookInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.experimental.bybit.OrderBookUSDT

class pybotters.models.experimental.bybit.**OrderBookUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])

次のページに続く

表 81 – 前のページからの続き

wait()

pybotters.models.experimental.bybit.OrderInverse

class pybotters.models.experimental.bybit.**OrderInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

pybotters.models.experimental.bybit.OrderUSDT

class pybotters.models.experimental.bybit.**OrderUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)

次のページに続く

表 83 – 前のページからの続き

wait()

pybotters.models.experimental.bybit.PositionInverse

class pybotters.models.experimental.bybit.**PositionInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
both(symbol)
find([query])
get(item)
one(symbol)
wait()

pybotters.models.experimental.bybit.PositionUSDT

class pybotters.models.experimental.bybit.**PositionUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`both`(symbol)
`find`([query])
`get`(item)
`one`(symbol)
`wait`()

pybotters.models.experimental.bybit.StopOrderInverse

class pybotters.models.experimental.bybit.**StopOrderInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.StopOrderUSDT

class pybotters.models.experimental.bybit.**StopOrderUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.TradeInverse

class pybotters.models.experimental.bybit.**TradeInverse**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.TradeUSDT

class pybotters.models.experimental.bybit.**TradeUSDT**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.experimental.bybit.Wallet

class pybotters.models.experimental.bybit.**Wallet**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx

Classes

FTXDataStore([auto_cast])	FTX のデータストアマネージャー
Fills([keys, data, auto_cast])	
Markets([keys, data, auto_cast])	
OrderBook([keys, data, auto_cast])	
Orders([keys, data, auto_cast])	
Positions([keys, data, auto_cast])	
Ticker([keys, data, auto_cast])	
Trades([keys, data, auto_cast])	

pybotters.models.ftx.FTXDataStore

class pybotters.models.ftx.**FTXDataStore**(*auto_cast: bool = False*)

FTX のデータストアマネージャー

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

fills	
markets	
orderbook	
orders	アクティブオーダーのみ (約定・キャンセル済みは削除される)
positions	
ticker	
trades	

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /orders (DataStore: orders)
- GET /conditional_orders (DataStore: orders)
- GET /positions (DataStore: positions)
  - fills 受信時に GET /positions の自動フェッチする機能が有効化される。

property orders: *Orders*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

pybotters.models.ftx.Fills

class pybotters.models.ftx.**Fills**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx.Markets

class pybotters.models.ftx.**Markets**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx.OrderBook

class pybotters.models.ftx.**OrderBook**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.ftx.Orders

class pybotters.models.ftx.**Orders**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx.Positions

class pybotters.models.ftx.**Positions**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx.Ticker

class pybotters.models.ftx.**Ticker**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.ftx.Trades

class pybotters.models.ftx.**Trades**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])	
`find`([query])	
`get`(item)	
`wait`()	

pybotters.models.gmocoin

Functions

`parse_datetime`(x)	

Classes

ApiType(value)	API 区分
CancelType(value)	取消区分
Channel(value)	WebSocket API チャンネル
Execution(**kwargs)	
ExecutionStore([keys, data, auto_cast])	
ExecutionType(value)	注文タイプ
GMOCoinDataStore([auto_cast])	GMO コインのデータストアマネージャー
MessageHelper()	
MessageType(value)	メッセージタイプ
Order(**kwargs)	
OrderBook(**kwargs)	

次のページに続く

表 102 – 前のページからの続き

OrderBookStore([keys, data, auto_cast])	
OrderLevel(**kwargs)	
OrderSide(value)	売買区分
OrderStatus(value)	注文ステータス
OrderStore([keys, data, auto_cast])	
OrderType(value)	取引区分
Position(**kwargs)	
PositionStore([keys, data, auto_cast])	
PositionSummary(**kwargs)	
PositionSummaryStore([keys, data, auto_cast])	
SettleType(value)	決済区分
Symbol(value)	取り扱い銘柄
Ticker(**kwargs)	
TickerStore([keys, data, auto_cast])	
TimeInForce(value)	執行数量条件
Trade(**kwargs)	
TradeStore([keys, data, auto_cast])	

pybotters.models.gmocoin.ApiType

class pybotters.models.gmocoin.**ApiType**(*value*)

API 区分

Attributes

Public
Private

pybotters.models.gmocoin.CancelType

class pybotters.models.gmocoin.**CancelType**(*value*)

取消区分

Attributes

NONE
USER
POSITION_LOSSCUT
INSUFFICIENT_BALANCE
INSUFFICIENT_MARGIN
ACCOUNT_LOSSCUT
MARGIN_CALL
MARGIN_CALL_LOSSCUT
EXPIRED_FAK
EXPIRED_FOK
EXPIRED_SOK
CLOSED_ORDER

次のページに続く

表 104 – 前のページからの続き

SOK_TAKER
PRICE_LIMIT

pybotters.models.gmocoin.Channel

class pybotters.models.gmocoin.**Channel**(*value*)

WebSocket API チャンネル

Methods

from_str(name)

Attributes

TICKER
ORDER_BOOKS
TRADES
EXECUTION_EVENTS
ORDER_EVENTS
POSITION_EVENTS
POSITION_SUMMARY_EVENTS

pybotters.models.gmocoin.Execution

class pybotters.models.gmocoin.**Execution**(***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`()	2-tuple; but raise KeyError if D is empty.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

Attributes

execution_id
order_id
symbol
side
settle_type
size
price
timestamp
loss_gain
fee
position_id
execution_type
order_price
order_size
order_executed_size
order_timestamp
time_in_force

pybotters.models.gmocoin.ExecutionStore

class pybotters.models.gmocoin.**ExecutionStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`sorted`([query])
`wait`()

pybotters.models.gmocoin.ExecutionType

class pybotters.models.gmocoin.**ExecutionType**(*value*)

注文タイプ

Attributes

MARKET
LIMIT
STOP

pybotters.models.gmocoin.GMOCoinDataStore

class pybotters.models.gmocoin.**GMOCoinDataStore**(*auto_cast: bool = False*)

GMO コインのデータストアマネージャー

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
initialize(*aws)	対応エンドポイント
`onmessage`(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
`wait`()	非同期メソッド。onmessage のイベントがあるまで待機します。

Attributes

`executions`	
`orderbooks`	
orders	アクティブオーダーのみ (約定・キャンセル済みは削除される)
`position_summary`	
`positions`	
`ticker`	
`trades`	

async initialize(**aws: Awaitable[aiohttp.client_reqrep.ClientResponse]*) → None

対応エンドポイント

- GET /private/v1/latestExecutions (DataStore: executions)
- GET /private/v1/activeOrders (DataStore: orders)
- GET /private/v1/openPositions (DataStore: positions)
- GET /private/v1/positionSummary (DataStore: position_summary)

property orders: *pybotters.models.gmocoin.OrderStore*

アクティブオーダーのみ (約定・キャンセル済みは削除される)

pybotters.models.gmocoin.MessageHelper

class pybotters.models.gmocoin.**MessageHelper**

Methods

__init__()
to_execution(data)
to_executions(data)
to_order(data)
to_orderbook(data)
to_orders(data)
to_position(data)
to_position_summaries(data)
to_position_summary(data)
to_positions(data)
to_ticker(data)

次のページに続く

表 113 – 前のページからの続き

to_tickers(data)
to_trade(data)
to_trades(data)

pybotters.models.gmocoin.MessageType

class pybotters.models.gmocoin.**MessageType**(*value*)

メッセージタイプ

Attributes

NONE
ER
NOR
ROR
COR
OPR
UPR
ULR
CPR
INIT
UPDATE

次のページに続く

表 114 – 前のページからの続き

PERIODIC

pybotters.models.gmocoin.Order

class pybotters.models.gmocoin.**Order**(***kwargs*)

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

order_id
symbol
settle_type
execution_type
side
order_status
order_timestamp
price
size
executed_size
losscut_price
time_in_force
cancel_type

pybotters.models.gmocoin.OrderBook

class pybotters.models.gmocoin.**OrderBook**(***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`()	2-tuple; but raise KeyError if D is empty.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

Attributes

`asks`
`bids`
`symbol`
`timestamp`

pybotters.models.gmocoin.OrderBookStore

class pybotters.models.gmocoin.**OrderBookStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])	
`find`([query])	
`get`(item)	
`sorted`([query])	
`wait`()	

pybotters.models.gmocoin.OrderLevel

class pybotters.models.gmocoin.**OrderLevel**(***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.

次のページに続く

表 120 – 前のページからの続き

items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

symbol
side
price
size

pybotters.models.gmocoin.OrderSide

class pybotters.models.gmocoin.**OrderSide**(*value*)

売買区分

Attributes

BUY
SELL

pybotters.models.gmocoin.OrderStatus

class pybotters.models.gmocoin.**OrderStatus**(*value*)

注文ステータス

Attributes

WAITING
ORDERED
MODIFYING
CANCELLING
CANCELED
EXECUTED
EXPIRED

pybotters.models.gmocoin.OrderStore

class pybotters.models.gmocoin.**OrderStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.gmocoin.OrderType

class pybotters.models.gmocoin.**OrderType**(*value*)

取引区分

Attributes

NORMAL
LOSSCUT

pybotters.models.gmocoin.Position

class pybotters.models.gmocoin.**Position**(***kwargs*)

Methods

`__init__`(*args, **kwargs)
`clear`()
`copy`()

次のページに続く

表 126 – 前のページからの続き

fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

position_id
symbol
side
size
orderd_size
price
loss_gain

次のページに続く

表 127 – 前のページからの続き

leverage
losscut_price
timestamp

pybotters.models.gmocoin.PositionStore

class pybotters.models.gmocoin.**PositionStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

pybotters.models.gmocoin.PositionSummary

class pybotters.models.gmocoin.**PositionSummary**(***kwargs*)

Methods

__init__(*args, **kwargs)
clear()

次のページに続く

表 129 – 前のページからの続き

copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

symbol
side
average_position_rate
position_loss_gain
sum_order_quantity
sum_position_quantity

次のページに続く

表 130 – 前のページからの続き

timestamp

pybotters.models.gmocoin.PositionSummaryStore

class pybotters.models.gmocoin.**PositionSummaryStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

pybotters.models.gmocoin.SettleType

class pybotters.models.gmocoin.**SettleType**(*value*)

決済区分

Attributes

OPEN
CLOSE
LOSS_CUT

pybotters.models.gmocoin.Symbol

class pybotters.models.gmocoin.**Symbol**(*value*)

取り扱い銘柄

Attributes

BTC
ETH
BCH
LTC
XRP
BTC_JPY
ETH_JPY
BCH_JPY
LTC_JPY
XRP_JPY

pybotters.models.gmocoin.Ticker

class pybotters.models.gmocoin.**Ticker**(***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`clear`()	
`copy`()	
`fromkeys`([value])	Create a new dictionary with keys from iterable and values set to value.
`get`(key[, default])	Return the value for key if key is in the dictionary, else default.
`items`()	
`keys`()	
`pop`(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`()	2-tuple; but raise KeyError if D is empty.
`setdefault`(key[, default])	Insert key with a value of default if key is not in the dictionary.
`update`([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
`values`()	

Attributes

`ask`
`bid`
`high`
`last`

次のページに続く

表 135 – 前のページからの続き

low
symbol
timestamp
volume

pybotters.models.gmocoin.TickerStore

class pybotters.models.gmocoin.**TickerStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

`__init__`([keys, data, auto_cast])
`find`([query])
`get`(item)
`wait`()

pybotters.models.gmocoin.TimeInForce

class pybotters.models.gmocoin.**TimeInForce**(*value*)

執行数量条件

Attributes

FAK
FAS
FOK
SOK

pybotters.models.gmocoin.Trade

class pybotters.models.gmocoin.**Trade**(***kwargs*)

Methods

__init__(*args, **kwargs)	
clear()	
copy()	
fromkeys([value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.

次のページに続く

表 138 – 前のページからの続き

update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

Attributes

price
side
size
timestamp
symbol

pybotters.models.gmocoin.TradeStore

class pybotters.models.gmocoin.**TradeStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, *, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

6.1.5 pybotters.request

pybotters.**request**(*method: str*, *url: str*, ***, *params: Optional[Mapping[str, str]] = None*, *data: Any = None*, *apis: Optional[Union[dict[str, list[str]], str]] = None*, ***kwargs: Any*) → *SyncClientResponse*

6.1.6 pybotters.store

Classes

DataStore([keys, data, auto_cast])	
DataStoreManager([auto_cast])	データストアマネージャーの抽象クラスです。 データストアの作成・参照・ハンドリングなどの役割を持ちます。 それぞれの取引所のクラスが継承します。

pybotters.store.DataStore

class pybotters.store.**DataStore**(*keys: Optional[list[str]] = None*, *data: Optional[list[Item]] = None*, ***, *auto_cast: bool = False*)

Methods

__init__([keys, data, auto_cast])
find([query])
get(item)
wait()

pybotters.store.DataStoreManager

class pybotters.store.**DataStoreManager**(*auto_cast: bool = False*)

データストアマネージャーの抽象クラスです。 データストアの作成・参照・ハンドリングなどの役割を持ちます。 それぞれの取引所のクラスが継承します。

Methods

`__init__`([auto_cast])	
`create`(name, *[, keys, data, datastore_class])	
`get`(name, type)	
onmessage(msg, ws)	Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。
wait()	非同期メソッド。onmessage のイベントがあるまで待機します。

onmessage(*msg: Any*, *ws:* pybotters.ws.ClientWebSocketResponse) → None

Client クラス ws_connect メソッドの引数 send_json に渡すハンドラです。

async **wait**() → None

非同期メソッド。onmessage のイベントがあるまで待機します。

6.1.7 pybotters.typedefs

6.1.8 pybotters.ws

Functions

`pretty_modulename`(e)	
`ws_run_forever`(url, session, event, *[, ...])	

Classes

Auth()
AuthHosts()
ClientWebSocketResponse(*args, **kwargs)
Heartbeat()
HeartbeatHosts()
Item(name, func)
RequestLimit()
RequestLimitHosts()

pybotters.ws.Auth

class pybotters.ws.**Auth**

Methods

__init__()
bitflyer(ws)
ftx(ws)
liquid(ws)
phemex(ws)

pybotters.ws.AuthHosts

class pybotters.ws.**AuthHosts**

Methods

`__init__`()

Attributes

`items`

pybotters.ws.ClientWebSocketResponse

class pybotters.ws.**ClientWebSocketResponse**(**args*, ***kwargs*)

Methods

`__init__`(*args, **kwargs)	
`close`(*[, code, message])	
`exception`()	
`get_extra_info`(name[, default])	extra info from connection transport
`ping`([message])	
`pong`([message])	
`receive`([timeout])	
`receive_bytes`(*[, timeout])	

次のページに続く

表 149 – 前のページからの続き

receive_json(*[, loads, timeout])
receive_str(*[, timeout])
send_bytes(data[, compress])
send_json(data[, compress, dumps])
send_str(*args, **kwargs)

Attributes

client_notakeover
close_code
closed
compress
protocol

pybotters.ws.Heartbeat

class pybotters.ws.**Heartbeat**

Methods

__init__()
binance(ws)
bitbank(ws)

次のページに続く

表 151 – 前のページからの続き

bybit(ws)
ftx(ws)
liquid(ws)
phemex(ws)

pybotters.ws.HeartbeatHosts

class pybotters.ws.**HeartbeatHosts**

Methods

__init__()

Attributes

items

pybotters.ws.Item

class pybotters.ws.**Item**(*name: 'str'*, *func: 'Any'*)

Methods

__init__(name, func)

Attributes

name
func

pybotters.ws.RequestLimit

class pybotters.ws.**RequestLimit**

Methods

__init__()
gmocoin(ws, send_str)

pybotters.ws.RequestLimitHosts

class pybotters.ws.**RequestLimitHosts**

Methods

__init__()

Attributes

items

第 7 章

Indices and tables

- genindex
- modindex
- search

Python モジュール索引

p

索引